Problem Statement

- Zee Recommender Systems represents an ambitious venture by Zee to enhance user experience through personalized movie recommendations.
- The focus is on leveraging user ratings and similarities among users to create a robust, personalized movie recommender system.
- Utilizing a comprehensive dataset of movie ratings, user demographics, and movie details, Zee aims to develop a system that can accurately predict user preferences and suggest movies accordingly.
- The insights gained from this system are expected to drive user engagement, increase satisfaction, and foster a more intuitive user experience.

Data Dictionary

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
- 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"

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

Loading dependencies and dataset

In [218…

```
# !pip install cmfrec

import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

import warnings
# warnings.filterwarnings("ignore", message="Attempting to use more than 1 t
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.set_option('display.max_columns', None)

# from scipy.stats import levene, f_oneway, kruskal
# from scipy.stats import ttest_ind
# from scipy.stats import chi2_contingency
# from statsmodels.graphics.gofplots import qqplot

from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from cmfrec import CMF



from sklearn.metrics import mean_squared_error as mse, mean_absolute_percent
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Dataset: Users

In [219…

```
users = pd.read_fwf('./data/users.dat', encoding='ISO-8859-1', header=None)
users.head()
```

Out[219]:

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

In [220…

```
users_cln1 = users.drop([0], axis=0)
users_cln1['User_ID'] = users_cln1[0].str.split('::').str[0]
users_cln1['Gender'] = users_cln1[0].str.split('::').str[1]
users_cln1['Age'] = users_cln1[0].str.split('::').str[2]
users_cln1['Occupation'] = users_cln1[0].str.split('::').str[3]
users_cln1['Zipcode'] = users_cln1[0].str.split('::').str[4]
users_cln1.drop(0, axis=1, inplace=True)


users_cln1.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_cln1.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_cln1.reset_index(drop=True, inplace=True)
users_cln1.head()
```

Out[220]:

	User_ID	Gender	Age	Occupation	Zipcode
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

Dataset: Movies

In [221…

```
movies = pd.read_fwf('./data/movies.dat', encoding='ISO-8859-1', header=None
movies.head()
```

Out[221]:

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

In [222…

```
movies_cln1 = movies.drop([1, 2], axis=1).drop([0], axis=0)
movies_cln1['Movie_ID'] = movies_cln1[0].str.split('::').str[0]
movies_cln1['Title'] = movies_cln1[0].str.split('::').str[1]
movies_cln1['Genres'] = movies_cln1[0].str.split('::').str[2]
movies_cln1.drop(0, axis=1, inplace=True)
movies_cln1.reset_index(drop=True, inplace=True)
movies_cln1.head()
```

Out[222]:

	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

Dataset: Ratings

In [223…
```
ratings = pd.read_fwf('./data/ratings.dat', encoding='ISO-8859-1', header=No
ratings.head()
```

Out[223]:

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

In [224…
```
ratings_cln1 = ratings.drop([0], axis=0)
ratings_cln1['User_ID'] = ratings_cln1[0].str.split('::').str[0]
ratings_cln1['Movie_ID'] = ratings_cln1[0].str.split('::').str[1]
ratings_cln1['Rating'] = ratings_cln1[0].str.split('::').str[2]
ratings_cln1['Timestamp'] = ratings_cln1[0].str.split('::').str[3]
ratings_cln1.drop(0, axis=1, inplace=True)
ratings_cln1.reset_index(drop=True, inplace=True)
ratings_cln1.head()
```

Out[224]:

	User_ID	Movie_ID	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

Data Cleaning, ReFormatting the datasets, Basic EDA:

Users

In [225…
```
users_cln1.info()
```

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

In [226…
```
# Checking duplicates
users_cln1.loc[users_cln1.duplicated()]
```

Out[226]:

	User_ID	Gender	Age	Occupation	Zipcode

In [227…
```
# users_cln1['User_ID'] = users_cln1['User_ID'].astype('int64')
users_cln1['User_ID'].nunique()
```

Out[227]:
```
6040
```

In [228…
```
users_cln1['Gender'].value_counts()
```

Out[228]:
```
M    4331
F    1709
Name: Gender, dtype: int64
```

In [229…
```
users_cln1['Age'].value_counts()
```

Out[229]:
```
25-34       2096
35-44       1193
18-24       1103
45-49        550
50-55        496
56 Above     380
Under 18     222
Name: Age, dtype: int64
```

In [230…
```
users_cln1['Occupation'].value_counts()
```

Out[230]:
```
college/grad student    759
other                   711
executive/managerial    679
academic/educator       528
technician/engineer     502
programmer              388
sales/marketing         302
writer                  281
artist                  267
self-employed           241
doctor/health care      236
k-12 student            195
clerical/admin          173
scientist               144
retired                 142
lawyer                  129
customer service        112
homemaker                92
unemployed               72
tradesman/craftsman      70
farmer                   17
Name: Occupation, dtype: int64
```

In [231…
```
users_cln1['Zipcode'].nunique()
```

Out[231]:
```
3439
```

Movies

In [232…
```
movies_cln1.head()
```

Out[232]:

	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

In [233…
```
movies_cln1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Movie_ID  3883 non-null   object
 1   Title     3883 non-null   object
 2   Genres    3858 non-null   object
dtypes: object(3)
memory usage: 91.1+ KB
```

In [234…
```
# Checking duplicates
movies_cln1.loc[movies_cln1.duplicated()]
```

Out[234]:

	Movie_ID	Title	Genres

In [235…
```
# movies_cln1['Movie_ID'] = movies_cln1['Movie_ID'].astype('int64')
movies_cln1['Movie_ID'].nunique()
```

Out[235]:
```
3883
```

In [236…
```
movies_cln1['Title'].nunique()
```

Out[236]:
```
3883
```

Reformatting movie genres

In [237…
```
movies_cln2_explode = movies_cln1.copy()
movies_cln2_explode['Genres'] = movies_cln2_explode['Genres'].str.split('|')
movies_cln2_explode = movies_cln2_explode.explode(['Genres'])
movies_cln2_explode = movies_cln2_explode.reset_index().drop('index', axis=1
movies_cln2_explode
```

Out[237]:

	Movie_ID	Title	Genres
0	1	Toy Story (1995)	Animation
1	1	Toy Story (1995)	Children's
2	1	Toy Story (1995)	Comedy
3	2	Jumanji (1995)	Adventure
4	2	Jumanji (1995)	Children's
...	...	...	...
6361	3949	Requiem for a Dream (2000)	Drama
6362	3950	Tigerland (2000)	Drama
6363	3951	Two Family House (2000)	Drama
6364	3952	Contender, The (2000)	Drama
6365	3952	Contender, The (2000)	Thriller

6366 rows × 3 columns

In [238…
```
movies_cln2_explode.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6366 entries, 0 to 6365
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Movie_ID  6366 non-null   object
 1   Title     6366 non-null   object
 2   Genres    6341 non-null   object
dtypes: object(3)
memory usage: 149.3+ KB
```

In [239…
```
movies_cln2_explode.isna().sum()
```

Out[239]:
```
Movie_ID     0
Title        0
Genres      25
dtype: int64
```

In [240…
```
# movies_cln2_explode['Movie_ID'] = movies_cln2_explode['Movie_ID'].astype(
movies_cln2_explode['Movie_ID'].nunique()
```

Out[240]:
```
3883
```

In [241…
```
movies_cln2_explode['Genres'].nunique()
```

Out[241]:
```
63
```

In [242…
```
def clean_genre(word):
    clean_genre_dict = {
    'Action': ['A', 'Acti', 'Action'],
    'Adventure': ['Adv', 'Advent', 'Adventu', 'Adventur'],
    'Animation' : ['Animati'],
    'Children': ["Children's", "Childre", 'Chi', 'Chil', 'Childr', 'Childre
    'Documentary': ['Docu', 'Documen', 'Document', 'Documenta'],
    'Drama': ['D', 'Dr','Dram'],
    'Comedy': ['Com', 'Come', 'Comed'],
    'Fantasy': ['F', 'Fant', 'Fantas'],
    'Horror': ['Horr', 'Horro'],
    'Musical': ['Music'],
    'Romance': ['R', 'Ro', 'Rom', 'Roma', 'Roman'],
    'Sci-Fi': ['S', 'Sci', 'Sci-', 'Sci-F'],
    'Thriller': ['Th', 'Thri', 'Thrille'],
    'War': ['Wa'],
    'Western': ['We','Wester']
    }
    for key in clean_genre_dict:
        if word in clean_genre_dict[key]:
            return key
    return word


movies_cln2_explode['Genres'] = movies_cln2_explode['Genres'].apply(clean_ge
movies_cln2_explode['Genres'].value_counts()
```

Out[242]:
```
Drama          1585
Comedy         1189
Action          503
Thriller        488
Romance         462
Horror          340
Adventure       282
Sci-Fi          265
Children        249
Crime           210
War             139
Documentary     124
Musical         113
Mystery         105
Animation       104
Western          68
Fantasy          63
Film-Noir        44
                  8
Name: Genres, dtype: int64
```

In [243…
```
movies_cln2_explode.loc[movies_cln2_explode['Genres']=='', 'Genres'] = np.na
movies_cln2_explode.isna().sum()
```

Out[243]:
```
Movie_ID     0
Title        0
Genres      33
dtype: int64
```

In [244…
```
genre_mode = movies_cln2_explode['Genres'].mode()[0]
movies_cln2_explode['Genres'].fillna(genre_mode, inplace=True)
movies_cln2_explode.isna().sum()
```

Out[244]:
```
Movie_ID    0
Title       0
Genres      0
dtype: int64
```

In [245…
```
movies_cln2_explode['Genres'].nunique()
```

Out[245]:
```
18
```

In [246…
```
movies_cln2_explode['Genres'].value_counts()
```

Out[246]:
```
Drama          1618
Comedy         1189
Action          503
Thriller        488
Romance         462
Horror          340
Adventure       282
Sci-Fi          265
Children        249
Crime           210
War             139
Documentary     124
Musical         113
Mystery         105
Animation       104
Western          68
Fantasy          63
Film-Noir        44
Name: Genres, dtype: int64
```

In [247…
```
# Checking duplicates after data cleaning
movies_cln2_explode.loc[movies_cln2_explode.duplicated()]
```

Out[247]:

	Movie_ID	Title	Genres
450	265	Like Water for Chocolate (Como agua para choco...	Drama
2153	1306	Until the End of the World (Bis ans Ende der W...	Drama
4913	3000	Princess Mononoke, The (Mononoke Hime) (1997)	Action

In [248…
```
movies_cln2_explode.drop_duplicates(inplace=True)
```

Reformatting movie genres: Multi Hot Encoding

In [249…
```
movies_cln2_explode.head()
```

Out[249]:

	Movie_ID	Title	Genres
0	1	Toy Story (1995)	Animation
1	1	Toy Story (1995)	Children
2	1	Toy Story (1995)	Comedy
3	2	Jumanji (1995)	Adventure
4	2	Jumanji (1995)	Children

In [250…

```
movie_genre_map = ~movies_cln2_explode.pivot(index='Movie_ID', columns='Gen
movie_genre_map = movie_genre_map.astype('int64')
movie_genre_map.columns = [tup[-1] for tup in movie_genre_map.columns]
movie_genre_map
```

Out[250]:

	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama
Movie_ID								
1	0	0	1	1	1	0	0	0
10	1	1	0	0	0	0	0	0
100	0	0	0	0	0	0	0	1
1000	0	0	0	0	0	1	0	0
1001	0	0	0	0	1	0	0	0
…	…	…	…	…	…	…	…	…
994	0	0	0	0	0	0	0	1
996	1	0	0	0	0	0	0	1
997	0	0	0	0	0	0	0	1
998	1	0	0	0	0	1	0	0
999	0	0	0	0	0	1	0	0

3883 rows × 18 columns

Extracting Release_Year

In [251…

```
movies_cln1.head()
```

Out[251]:

	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

In [252…

```
movies_cln3 = movies_cln1.copy()
movies_cln3.drop('Genres', axis=1, inplace=True)
```

```
movies_cln3['Release_Year'] = movies_cln3['Title'].str[-6:]
movies_cln3['Release_Year'].unique()
```

Out[252]:
```
array(['(1995)', '(1994)', '(1996)', 'in the', '(1976)', '(1993)',
       '(1992)', '(1988)', '(1967)', '1964):', '(1977)', ' der B',
       '(1965)', ' (1995', '(1982)', '(1962)', '(1990)', '(1991)',
       '(1989)', '(1937)', '(1940)', '(1969)', '(1981)', '(1973)',
       '(1970)', '(1960)', '(1955)', '(1956)', '(1959)', '(1968)',
       '(1980)', '(1975)', '(1986)', '1995):', '(1948)', '(1943)',
       'the Bo', '(1950)', '(1946)', '(1987)', '(1997)', 'n Arta',
       '(1974)', 'ans la', '(1958)', '(1949)', '(1972)', '(1998)',
       '(1933)', '(1952)', '(1951)', '(1957)', '(1961)', '(1954)',
       '(1934)', '(1944)', '(1963)', '(1942)', '(1941)', '(1964)',
       '(1953)', '(1939)', '(1947)', '(1945)', '(1938)', '(1935)',
       '(1936)', '(1926)', '(1932)', '(1930)', '(1971)', '(1979)',
       '(1966)', '(1978)', '(1985)', '(1983)', '(1984)', '(1931)',
       '(1922)', 'l) (19', '(1927)', '(1929)', ') (195', '(1928)',
       '(1925)', '(1923)', '(1999)', "re d'A", 'Polar)', ' (1998',
       ' (1965', '1981):', '(1919)', ' (1984', 'and) (', '(2000)',
       'en) (1', '(1920)', '(1921)', 'ndront', 'i) (19', 'he (19',
       '1989):', 've (19'], dtype=object)
```

In [253…
```
movies_cln3.loc[movies_cln3['Release_Year']=='and) (', 'Title'].values
```

Out[253]:
```
array(['March of the Wooden Soldiers (a.k.a. Laurel & Hardy in Toyland)
('],
      dtype=object)
```

In [254…
```
movies_cln3.loc[movies_cln3['Release_Year']=='he (19', 'Title'].values
```

Out[254]:
```
array(['Decline of Western Civilization Part II: The Metal Years, The (1
9'],
      dtype=object)
```

In [255…
```
movies_cln3['Release_Year'] = movies_cln3['Release_Year'].str.strip('() :')
pd.Series(movies_cln3['Release_Year'].unique()).sort_values().to_list()
```

Out[255]:
```
['1919',
 '1920',
 '1921',
 '1922',
 '1923',
 '1925',
 '1926',
 '1927',
 '1928',
 '1929',
 '1930',
 '1931',
 '1932',
 '1933',
 '1934',
 '1935',
 '1936',
 '1937',
 '1938',
 '1939',
 '1940',
 '1941',
 '1942',
 '1943',
 '1944',
 '1945',
 '1946',
 '1947',
 '1948',
 '1949',
 '195',
 '1950',
 '1951',
 '1952',
 '1953',
 '1954',
 '1955',
 '1956',
 '1957',
 '1958',
 '1959',
 '1960',
 '1961',
 '1962',
 '1963',
 '1964',
 '1965',
 '1966',
 '1967',
 '1968',
 '1969',
 '1970',
 '1971',
 '1972',
 '1973',
 '1974',
 '1975',
 '1976',
 '1977',
 '1978',
 '1979',
 '1980',
 '1981',
 '1982',
```

```
 '1983',
 '1984',
 '1985',
 '1986',
 '1987',
 '1988',
 '1989',
 '1990',
 '1991',
 '1992',
 '1993',
 '1994',
 '1995',
 '1996',
 '1997',
 '1998',
 '1999',
 '2000',
 'Polar',
 'and',
 'ans la',
 'der B',
 'en) (1',
 'he (19',
 'i) (19',
 'in the',
 'l) (19',
 'n Arta',
 'ndront',
 "re d'A",
 'the Bo',
 've (19']
```

In [256…

```
def clean_release_year(year):
    unclean_year_list = [
    'Polar',
    'and',
    'ans la',
    'der B',
    'en) (1',
    'he (19',
    'i) (19',
    'in the',
    'l) (19',
    'n Arta',
    'ndront',
    "re d'A",
    'the Bo',
    've (19'
    ]

    for phr in unclean_year_list:
        if year == '195':
            return '1958'
        elif year in unclean_year_list:
            return np.nan
    return year

movies_cln3['Release_Year'] = movies_cln3['Release_Year'].apply(clean_releas
movies_cln3['Release_Year'].value_counts()
```

Out[256]:
```
1996    344
1995    342
1998    334
1997    315
1999    283
       ...
1923      3
1919      3
1922      2
1920      2
1921      1
Name: Release_Year, Length: 81, dtype: int64
```

In [257…
```
movies_cln3.isna().sum()
```

Out[257]:
```
Movie_ID         0
Title            0
Release_Year    14
dtype: int64
```

In [258…
```
median_release_year = movies_cln3['Release_Year'].median()
movies_cln3['Release_Year'].fillna(median_release_year, inplace=True)
movies_cln3.isna().sum()
```

Out[258]:
```
Movie_ID        0
Title           0
Release_Year    0
dtype: int64
```

In [259…
```
def release_year_encode(year):
    if year<1930:
        return '20s'
    elif 1930<=year<1940:
        return '30s'
    elif 1940<=year<1950:
        return '40s'
    elif 1950<=year<1960:
        return '50s'
    elif 1960<=year<1970:
        return '60s'
    elif 1970<=year<1980:
        return '70s'
    elif 1980<=year<1990:
        return '80s'

    return '90s'

movies_cln3['Release_Year_Original'] = movies_cln3['Release_Year']
movies_cln3['Release_Year'] = movies_cln3['Release_Year'].apply(lambda x: re
movies_cln3['Release_Year'].value_counts()
```

Out[259]:
```
90s    2446
80s     596
70s     244
60s     190
50s     168
40s     126
30s      76
20s      37
Name: Release_Year, dtype: int64
```

Creating map b/w Movie_ID and Title

In [260…
```
movies_title_map = movies_cln3.copy()
movies_title_map.drop('Release_Year', axis=1, inplace=True)
movies_title_map.set_index('Movie_ID', inplace=True)
movies_title_map.head()
```

Out[260]:

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

In [261…
```
movie_id_title_map = dict(zip(movies_title_map.index, movies_title_map['Tit
print('First 5 key-value pairs:')
print('-'*50)
list(movie_id_title_map.items())[:5]
```

Out[261]:
```
First 5 key-value pairs:
--------------------------------------------------
 [('1', 'Toy Story (1995)'),
  ('2', 'Jumanji (1995)'),
  ('3', 'Grumpier Old Men (1995)'),
  ('4', 'Waiting to Exhale (1995)'),
  ('5', 'Father of the Bride Part II (1995)')]
```

Rating

In [262…
```
ratings_cln1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype 
---  ------     --------------    ----- 
 0   User_ID    1000209 non-null  object
 1   Movie_ID   1000209 non-null  object
 2   Rating     1000209 non-null  object
 3   Timestamp  1000209 non-null  object
dtypes: object(4)
memory usage: 30.5+ MB
```

In [263…
```
# Checking duplicates (across all columns)
ratings_cln1.loc[ratings_cln1.duplicated()]
```

Out[263]:

User_ID	Movie_ID	Rating	Timestamp

In [264…
```
# Checking duplicates (across subsets of columns)
ratings_cln1.loc[ratings_cln1.duplicated(subset=['User_ID', 'Movie_ID', 'Ra
```

Out[264]:

User_ID	Movie_ID	Rating	Timestamp

In [265…
```
# Checking duplicates (across subsets of columns)
ratings_cln1.loc[ratings_cln1.duplicated(subset=['User_ID', 'Movie_ID'])]
```

Out[265]:

User_ID	Movie_ID	Rating	Timestamp

In [ ]:

In [266…
```
# ratings_cln1['User_ID'] = ratings_cln1['User_ID'].astype('int64')
ratings_cln1['User_ID'].nunique()
```

Out[266]:
```
6040
```

In [267…
```
# ratings_cln1['Movie_ID'] = ratings_cln1['Movie_ID'].astype('int64')
ratings_cln1['Movie_ID'].nunique()
```

Out[267]:
```
3706
```

In [268…
```
ratings_cln1['Rating'] = ratings_cln1['Rating'].astype('int64')
ratings_cln1['Rating'].nunique()
```

Out[268]:
```
5
```

Extracting Features related to Watch event:

In [269…
```
ratings_cln1['Timestamp'] = ratings_cln1['Timestamp'].astype('int64')
```

In [270…
```
ratings_cln1['watch_date'] = pd.to_datetime(ratings_cln1['Timestamp'].apply(
ratings_cln1['watch_dow'] = ratings_cln1['Timestamp'].apply(lambda x: dateti
ratings_cln1['watch_hour'] = ratings_cln1['Timestamp'].apply(lambda x: date
```

In [271…
```
ratings_cln1['watch_date'].dt.year.value_counts()
```

Out[271]:
```
2000    904175
2001     68628
2002     24053
2003      3353
Name: watch_date, dtype: int64
```

In [272…
```
ratings_cln1['watch_date'].dt.month.value_counts()
```

Out[272]:
```
11    294083
8     190458
12    119472
7      95928
5      72863
6      61286
9      56259
10     46374
1      23560
4      19310
2      12180
3       8436
Name: watch_date, dtype: int64
```

In [273…
```
ratings_cln1['watch_date'].dt.day.value_counts()[:10]
```

Out[273]:
```
20    76813
21    58135
3     54819
19    44505
22    44008
2     38977
8     37218
6     35007
4     34870
1     33400
Name: watch_date, dtype: int64
```

In [274…
```
ratings_cln1['watch_dow'].value_counts()
```

Out[274]:
```
0    164527
1    159606
2    145677
3    139360
4    138858
6    136650
5    115531
Name: watch_dow, dtype: int64
```

In [275…
```
ratings_cln1['watch_hour'].value_counts()
```

Out[275]:
```
3     64092
8     62149
2     61393
1     60417
9     60256
0     59589
7     57888
6     56632
23    56378
5     53082
10    49288
22    49202
4     49146
21    47372
11    36161
20    34034
12    31898
19    24309
13    22657
14    18190
18    17159
15    10322
17    10203
16     8392
Name: watch_hour, dtype: int64
```

Visualizing the datasets before merging

In [276…
```
ratings_cln1.head(3)
```

Out[276]:

	User_ID	Movie_ID	Rating	Timestamp	watch_date	watch_dow	watch_hour
0	1	1193	5	978300760	2001-01-01	0	3
1	1	661	3	978302109	2001-01-01	0	4
2	1	914	3	978301968	2001-01-01	0	4

In [277…
```
users_cln1.head(3)
```

Out[277]:

	User_ID	Gender	Age	Occupation	Zipcode
0	1	F	Under 18	k-12 student	48067
1	2	M	56 Above	self-employed	70072
2	3	M	25-34	scientist	55117

In [278…
```
movies_cln3.head(3)
```

Out[278]:

	Movie_ID	Title	Release_Year	Release_Year_Original
0	1	Toy Story (1995)	90s	1995
1	2	Jumanji (1995)	90s	1995
2	3	Grumpier Old Men (1995)	90s	1995

In [279…
```
movie_genre_map.head(3)
```

Out[279]:

Movie_ID	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama
1	0	0	1	1	1	0	0	0
10	1	1	0	0	0	0	0	0
100	0	0	0	0	0	0	0	1

Merging all the datasets

In [280…
```
df_merged = pd.merge(ratings_cln1, users_cln1, on='User_ID', how='inner')
df_merged = pd.merge(df_merged, movies_cln3[['Movie_ID', 'Release_Year']], o
df_merged = pd.merge(df_merged, movie_genre_map, on='Movie_ID', how='inner')
df_merged = df_merged[['User_ID', 'Gender', 'Age', 'Occupation', 'Zipcode',
                       'Movie_ID', 'Release_Year', 'Action', 'Adventure', 'A
                       'Comedy', 'Crime','Documentary', 'Drama', 'Fantasy',
                       'Musical','Mystery', 'Romance', 'Sci-Fi', 'Thriller',
                       'Timestamp', 'watch_date', 'watch_dow', 'watch_hour',
                       'Rating'
                      ]]
df_merged.head()
```

Out[280]:

	User_ID	Gender	Age	Occupation	Zipcode	Movie_ID	Release_Year	Action
0	1	F	Under 18	k-12 student	48067	1193	70s	0
1	2	M	56 Above	self-employed	70072	1193	70s	0
2	12	M	25-34	programmer	32793	1193	70s	0
3	15	M	25-34	executive/managerial	22903	1193	70s	0
4	17	M	50-55	academic/educator	95350	1193	70s	0

In [281…
```
df_merged.shape
```

Out[281]:
```
(1000209, 30)
```

In [282…
```
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 30 columns):
 #   Column        Non-Null Count    Dtype         
---  ------        --------------    -----         
 0   User_ID       1000209 non-null  object        
 1   Gender        1000209 non-null  object        
 2   Age           1000209 non-null  object        
 3   Occupation    1000209 non-null  object        
 4   Zipcode       1000209 non-null  object        
 5   Movie_ID      1000209 non-null  object        
 6   Release_Year  1000209 non-null  object        
 7   Action        1000209 non-null  int64         
 8   Adventure     1000209 non-null  int64         
 9   Animation     1000209 non-null  int64         
 10  Children      1000209 non-null  int64         
 11  Comedy        1000209 non-null  int64         
 12  Crime         1000209 non-null  int64         
 13  Documentary   1000209 non-null  int64         
 14  Drama         1000209 non-null  int64         
 15  Fantasy       1000209 non-null  int64         
 16  Film-Noir     1000209 non-null  int64         
 17  Horror        1000209 non-null  int64         
 18  Musical       1000209 non-null  int64         
 19  Mystery       1000209 non-null  int64         
 20  Romance       1000209 non-null  int64         
 21  Sci-Fi        1000209 non-null  int64         
 22  Thriller      1000209 non-null  int64         
 23  War           1000209 non-null  int64         
 24  Western       1000209 non-null  int64         
 25  Timestamp     1000209 non-null  int64         
 26  watch_date    1000209 non-null  datetime64[ns]
 27  watch_dow     1000209 non-null  int64         
 28  watch_hour    1000209 non-null  int64         
 29  Rating        1000209 non-null  int64         
dtypes: datetime64[ns](1), int64(22), object(7)
memory usage: 236.6+ MB
```

In [283…
```
df_merged.isna().sum()
```

Out[283]:
```
User_ID         0
Gender          0
Age             0
Occupation      0
Zipcode         0
Movie_ID        0
Release_Year    0
Action          0
Adventure       0
Animation       0
Children        0
Comedy          0
Crime           0
Documentary     0
Drama           0
Fantasy         0
Film-Noir       0
Horror          0
Musical         0
Mystery         0
Romance         0
Sci-Fi          0
Thriller        0
War             0
Western         0
Timestamp       0
watch_date      0
watch_dow       0
watch_hour      0
Rating          0
dtype: int64
```

Watch History: EDA + Feature Engineering

Univariate Analysis

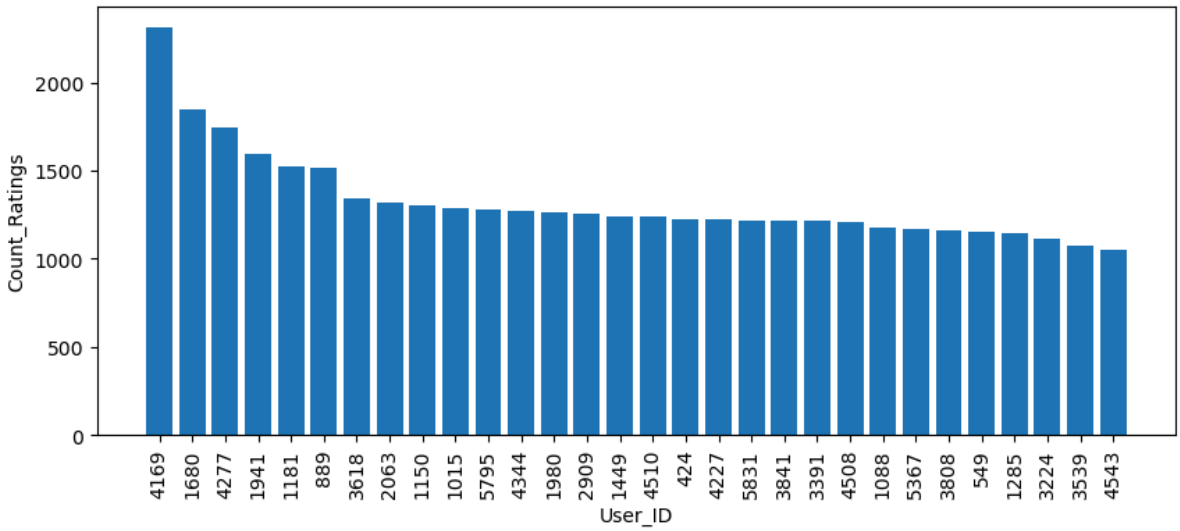
Related to User Metadata

In [284…
```
df_merged['User_ID'].nunique()
```

Out[284]:
```
6040
```

Users with Most Ratings

In [285…
```
plt.figure(figsize=(10, 4))
plt.bar(x=df_merged['User_ID'].value_counts()[:30].index, height=df_merged[
plt.xticks(rotation=90)
plt.xlabel('User_ID')
plt.ylabel('Count_Ratings')
plt.show()
```



User/Rating Split across Gender

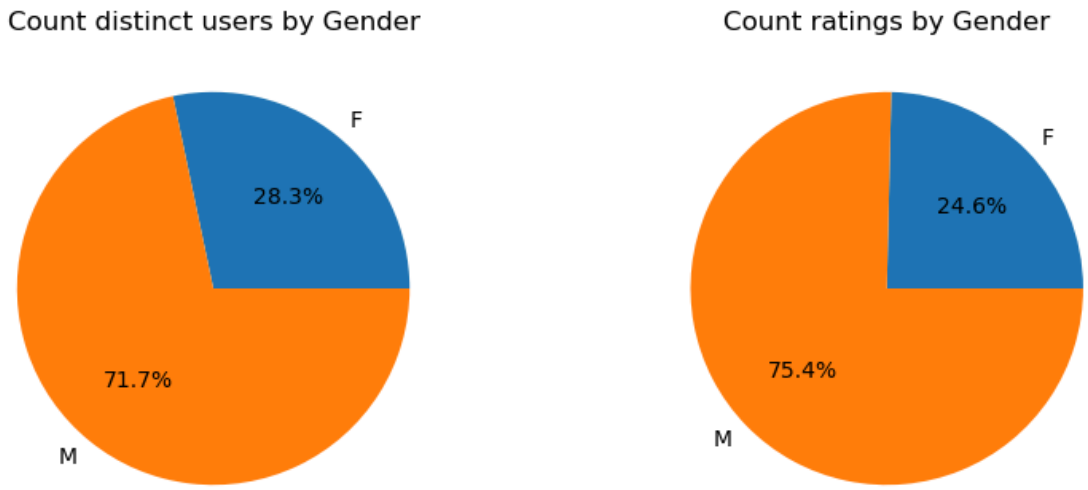
In [286…
```
cnt_users_by_gender = df_merged.groupby('Gender')['User_ID'].nunique()
cnt_users_by_gender
```

Out[286]:
```
Gender
F    1709
M    4331
Name: User_ID, dtype: int64
```

In [287…
```
cnt_ratings_by_gender = df_merged.groupby('Gender')['Rating'].count()
cnt_ratings_by_gender
```

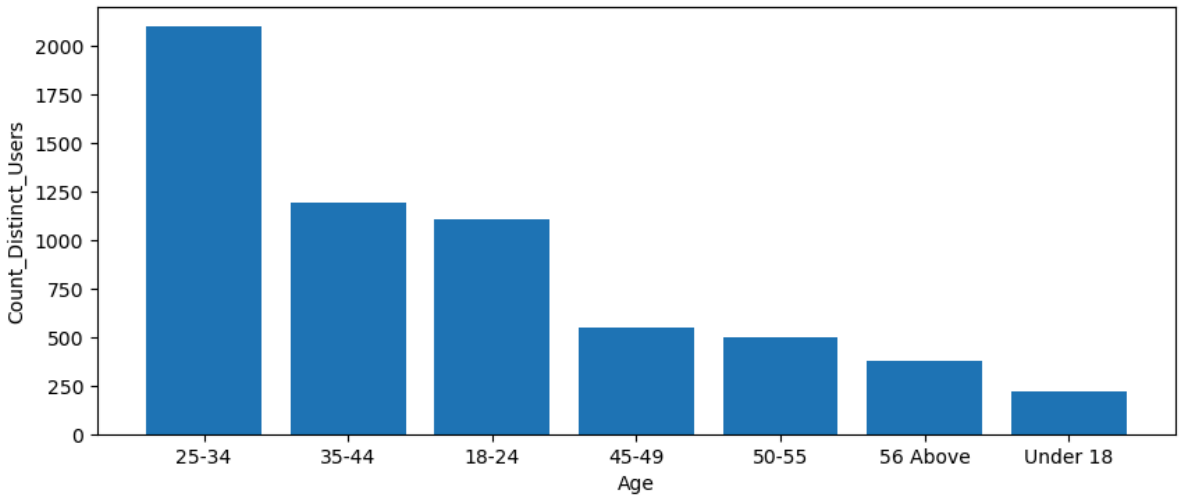
Out[287]:
```
Gender
F    246440
M    753769
Name: Rating, dtype: int64
```

In [288…
```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
axs[0].pie(cnt_users_by_gender, labels=cnt_users_by_gender.index, autopct='%
axs[0].set_title('Count distinct users by Gender')

axs[1].pie(cnt_ratings_by_gender, labels=cnt_ratings_by_gender.index, autopc
axs[1].set_title('Count ratings by Gender')

plt.show()
```

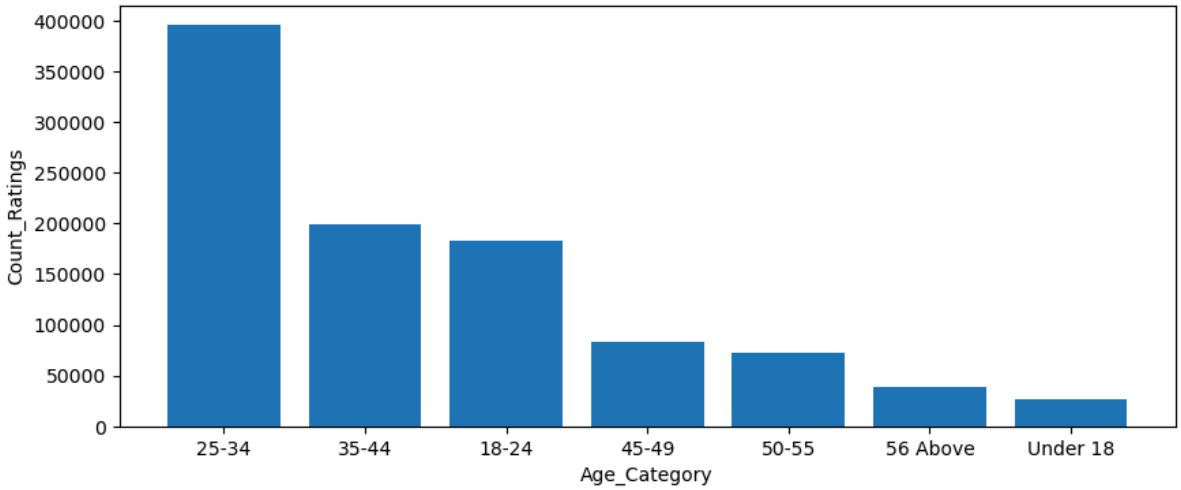


User/Rating Split across Age

In [289…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged.groupby('Age')['User_ID'].nunique().sort_values(ascending=
        df_merged.groupby('Age')['User_ID'].nunique().sort_values(ascending=
# plt.xticks(rotation=90)
plt.xlabel('Age')
plt.ylabel('Count_Distinct_Users')
plt.show()
```

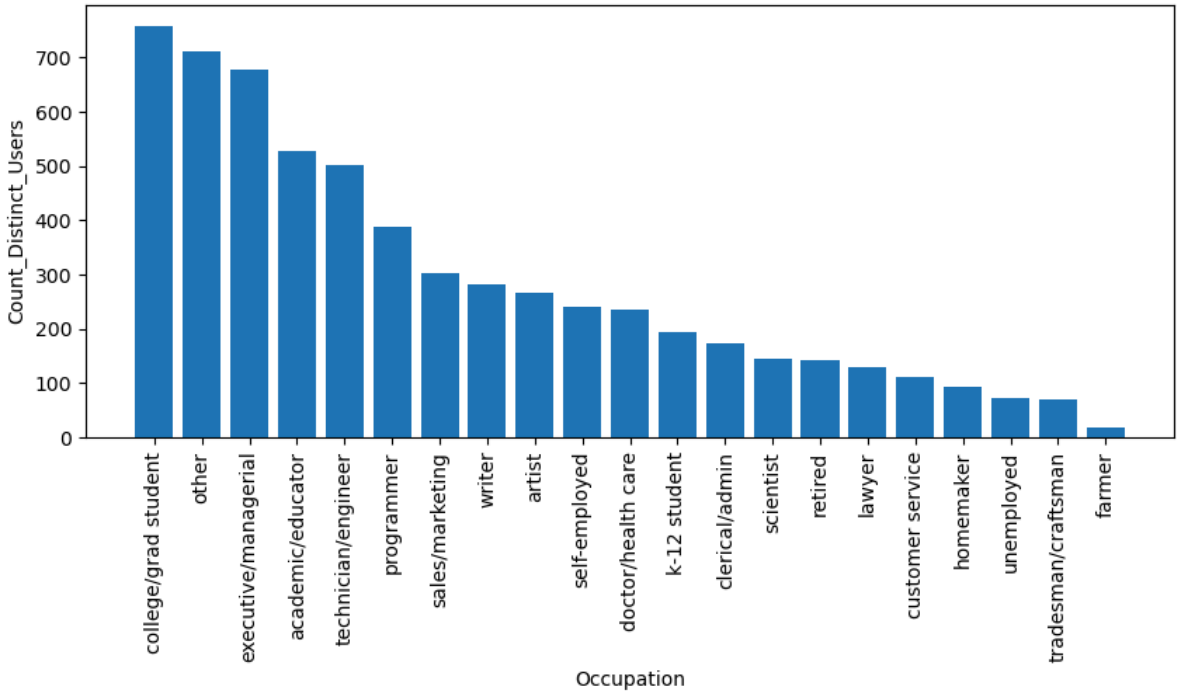


In [290…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['Age'].value_counts().index, df_merged['Age'].value_counts
plt.xlabel('Age_Category')
plt.ylabel('Count_Ratings')
plt.show()
```

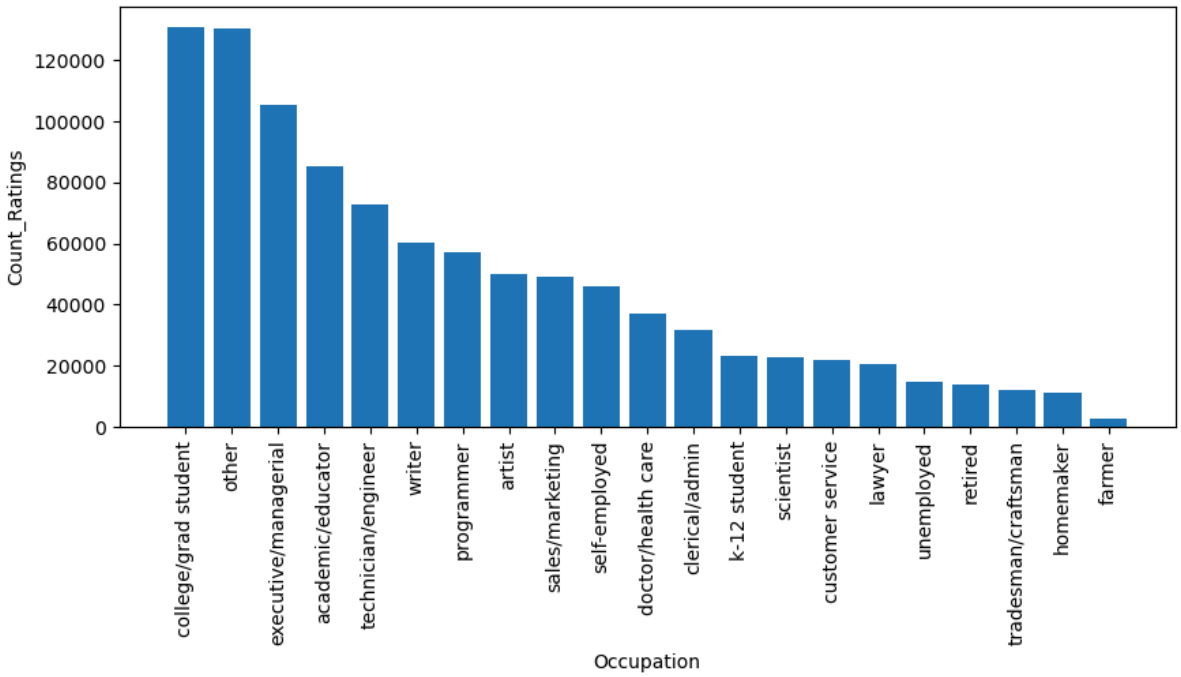


User/Rating Split across Occupation

In [291…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged.groupby('Occupation')['User_ID'].nunique().sort_values(asc
        df_merged.groupby('Occupation')['User_ID'].nunique().sort_values(asc
plt.xticks(rotation=90)
plt.xlabel('Occupation')
plt.ylabel('Count_Distinct_Users')
plt.show()
```



In [292…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['Occupation'].value_counts().index, df_merged['Occupation
plt.xticks(rotation=90)
plt.xlabel('Occupation')
plt.ylabel('Count_Ratings')
plt.show()
```



In [293…
```
# plt.figure(figsize=(10, 4))
# plt.bar(df_merged['Zipcode'].value_counts()[:20].index, df_merged['Zipcode
# plt.xticks(rotation=90)
# plt.xlabel('Zipcode')
# plt.ylabel('Count_Ratings')
# plt.show()
```

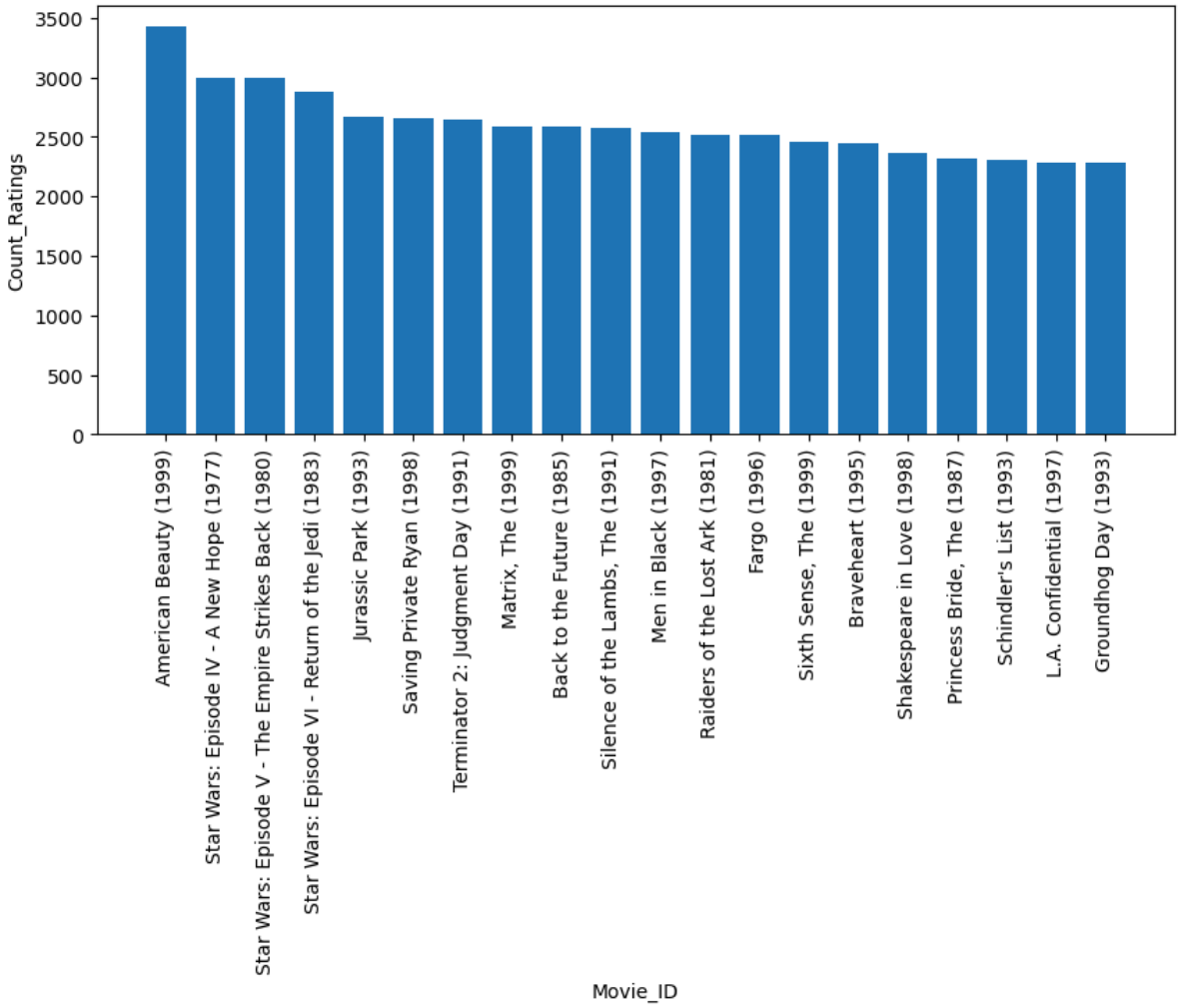
Related to Movie Metadata

In [294…
```
df_merged['Movie_ID'].nunique()
```

Out[294]: 3706

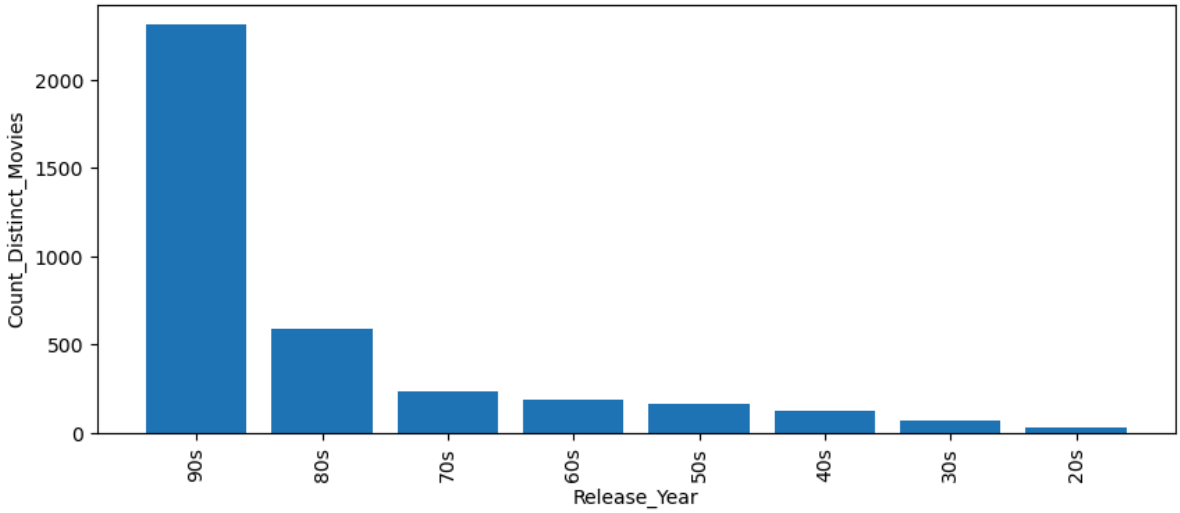
Most Rated Movies

In [295…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['Movie_ID'].value_counts()[:20].index,
        df_merged['Movie_ID'].value_counts()[:20].values)
plt.xlabel('Movie_ID')
plt.ylabel('Count_Ratings')
current_xlabels = df_merged['Movie_ID'].value_counts()[:20].index
new_xlabels = [movie_id_title_map[idx] for idx in current_xlabels]
plt.xticks(ticks=range(len(current_xlabels)), labels=new_xlabels)
plt.xticks(rotation=90)
plt.show()
```

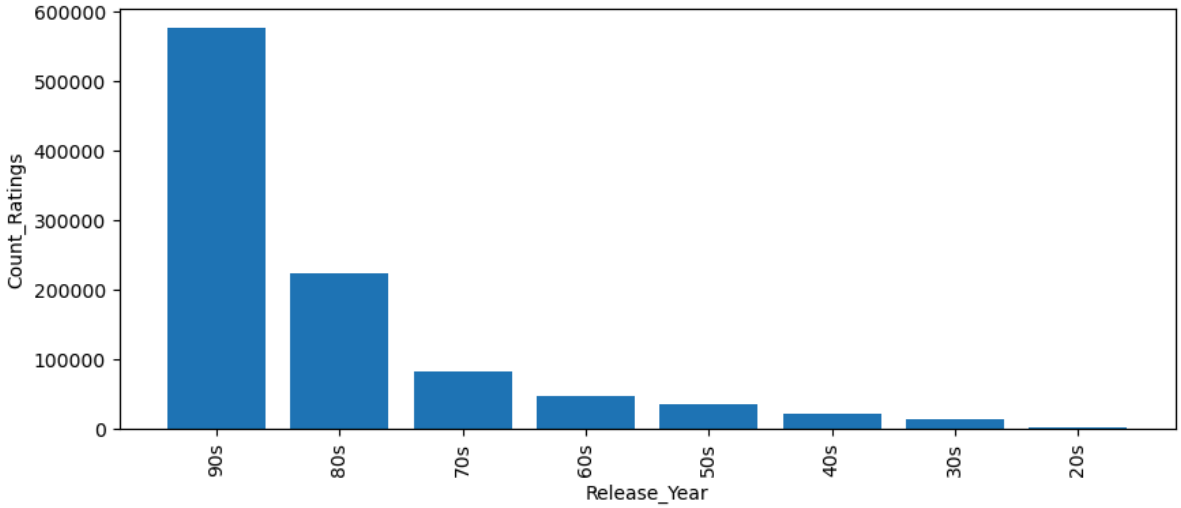


Movie/Rating split across Release Year (Release Decade)

In [296…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged.groupby('Release_Year')['Movie_ID'].nunique().sort_values(
        df_merged.groupby('Release_Year')['Movie_ID'].nunique().sort_values(
plt.xlabel('Release_Year')
plt.ylabel('Count_Distinct_Movies')
plt.xticks(rotation=90)
plt.show()
```

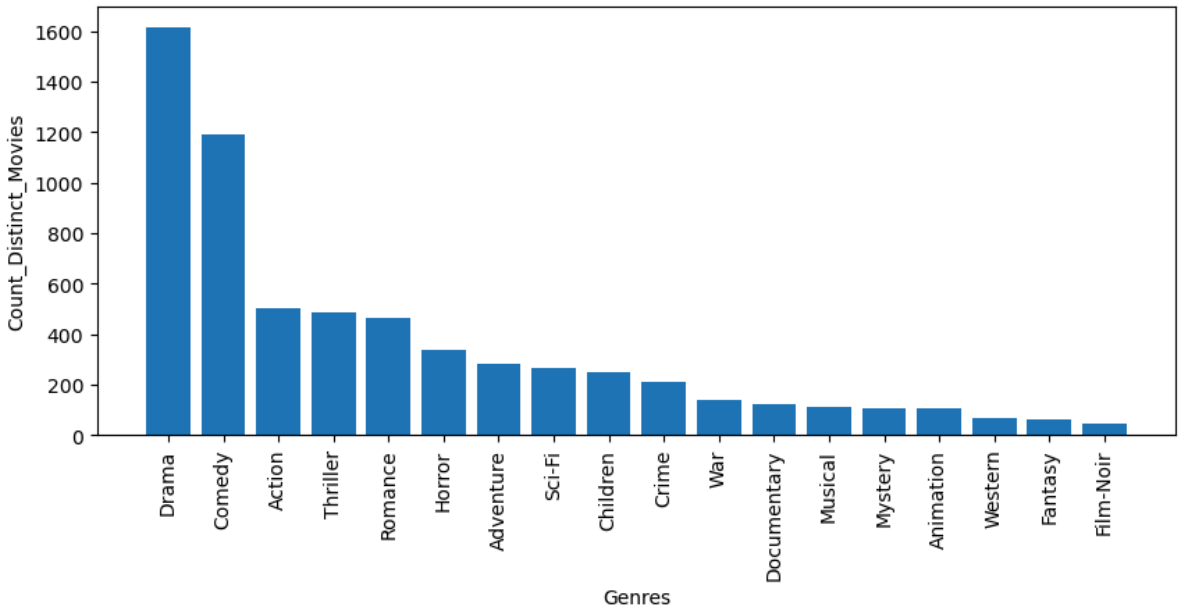


In [297…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['Release_Year'].value_counts()[:20].index,
        df_merged['Release_Year'].value_counts()[:20].values)
plt.xlabel('Release_Year')
plt.ylabel('Count_Ratings')
plt.xticks(rotation=90)
plt.show()
```



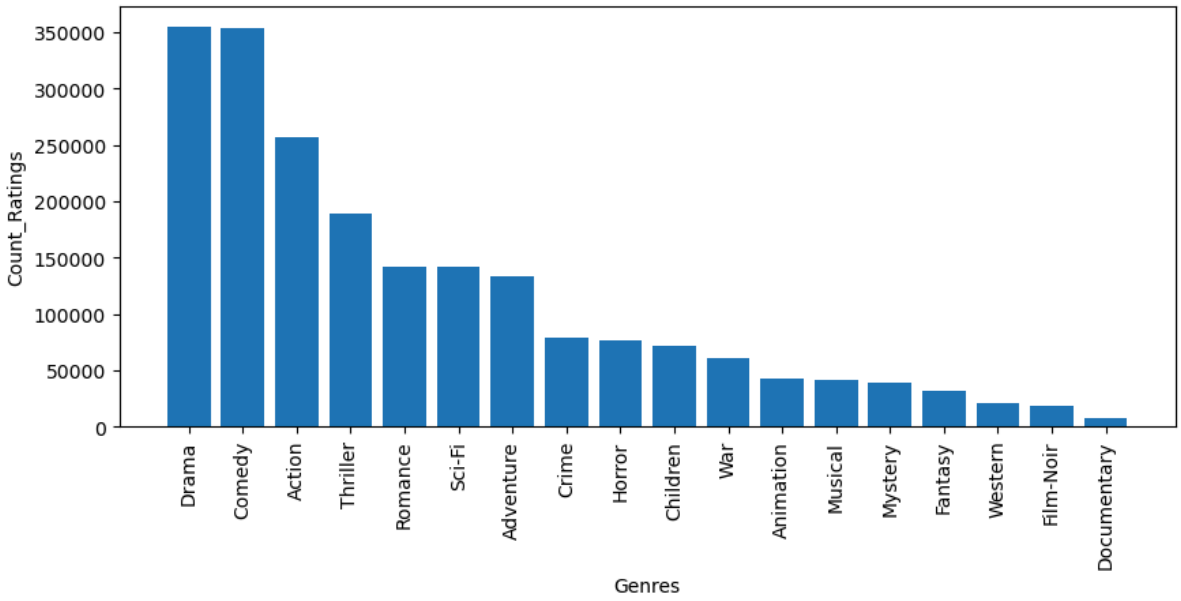
Movie/Rating split across Genre

In [298…
```
plt.figure(figsize=(10, 4))
plt.bar(movies_cln2_explode.groupby('Genres')['Movie_ID'].nunique().sort_va
        movies_cln2_explode.groupby('Genres')['Movie_ID'].nunique().sort_va
plt.xlabel('Genres')
plt.ylabel('Count_Distinct_Movies')
plt.xticks(rotation=90)
plt.show()
```



In [299…
```
cnt_ratings_by_genre = df_merged[['Action', 'Adventure', 'Animation', 'Chil
                                  'Crime', 'Documentary', 'Drama', 'Fantasy
                                  'Horror', 'Musical', 'Mystery', 'Romance',
                                  'Thriller', 'War', 'Western']].sum(axis=0)
```

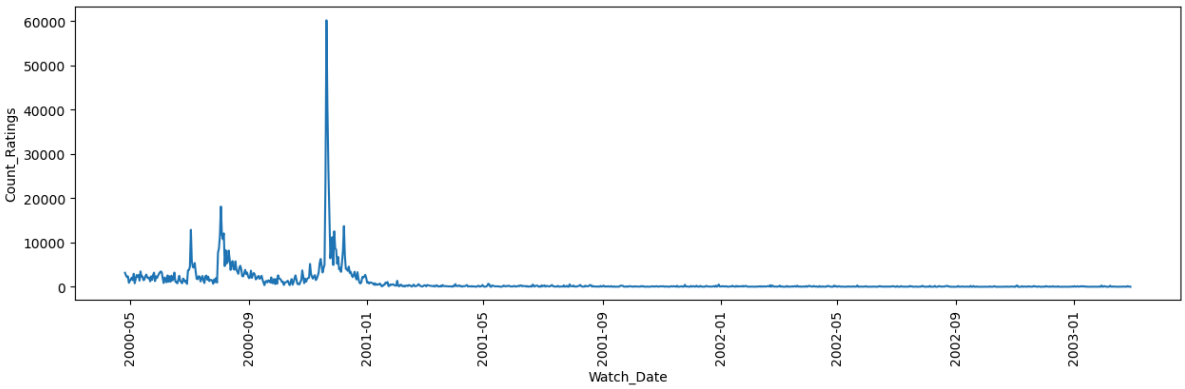
In [300…
```
plt.figure(figsize=(10, 4))
plt.bar(cnt_ratings_by_genre.index, cnt_ratings_by_genre.values)
plt.xlabel('Genres')
plt.ylabel('Count_Ratings')
plt.xticks(rotation=90)
plt.show()
```



Related to Watch Event Timestamp

Count of Ratings Day-on-Day

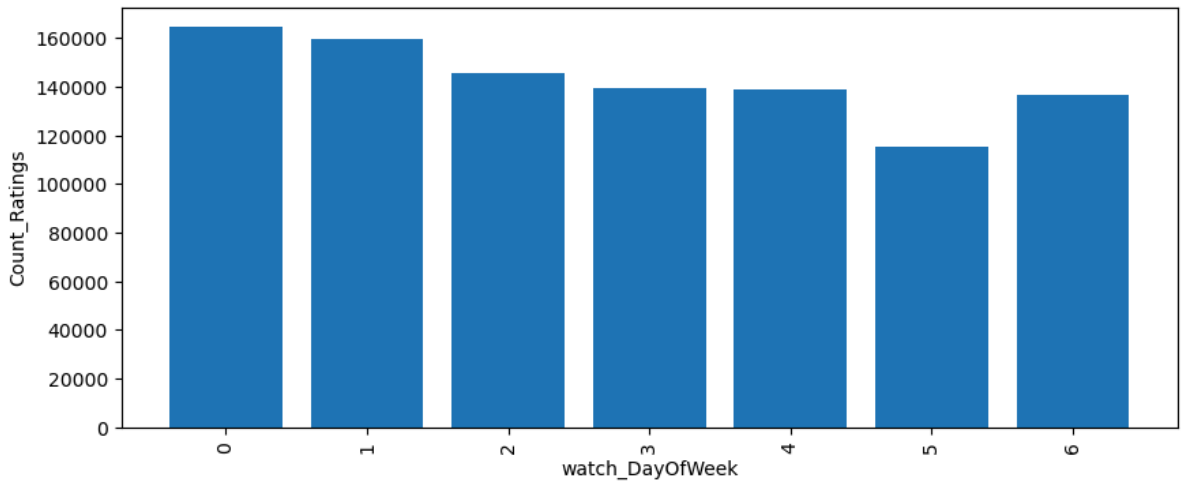
In [301…
```
plt.figure(figsize=(15, 4))
plt.plot(df_merged['watch_date'].value_counts().sort_index())
plt.xlabel('Watch_Date')
plt.ylabel('Count_Ratings')
plt.xticks(rotation=90)
plt.show()
```



Rating split across Day of Week

In [302…

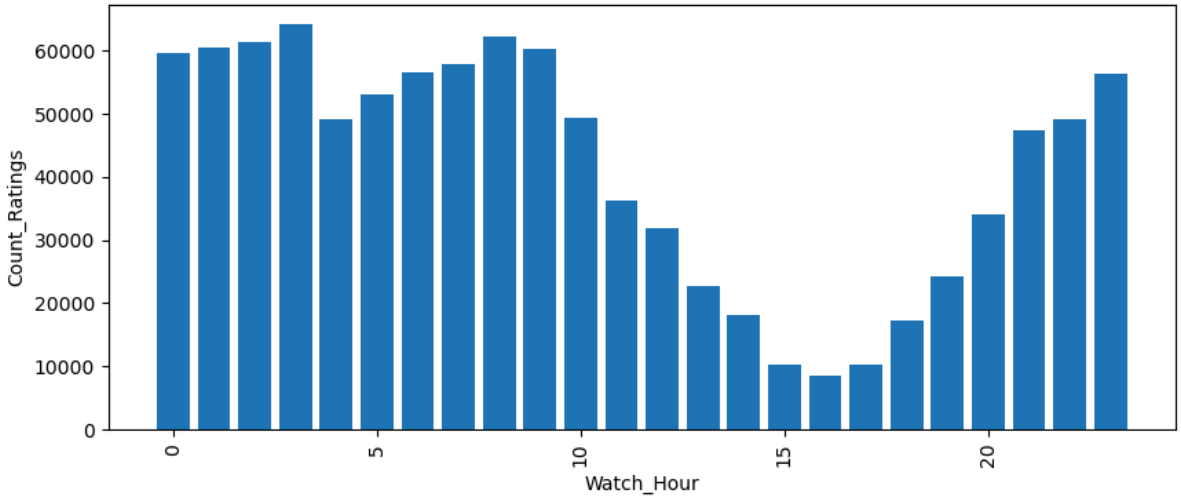
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['watch_dow'].value_counts().sort_index().index,
        df_merged['watch_dow'].value_counts().sort_index().values)
plt.xlabel('watch_DayOfWeek')
plt.ylabel('Count_Ratings')
plt.xticks(rotation=90)
plt.show()
```



Rating slit across Watch Hour

In [303…

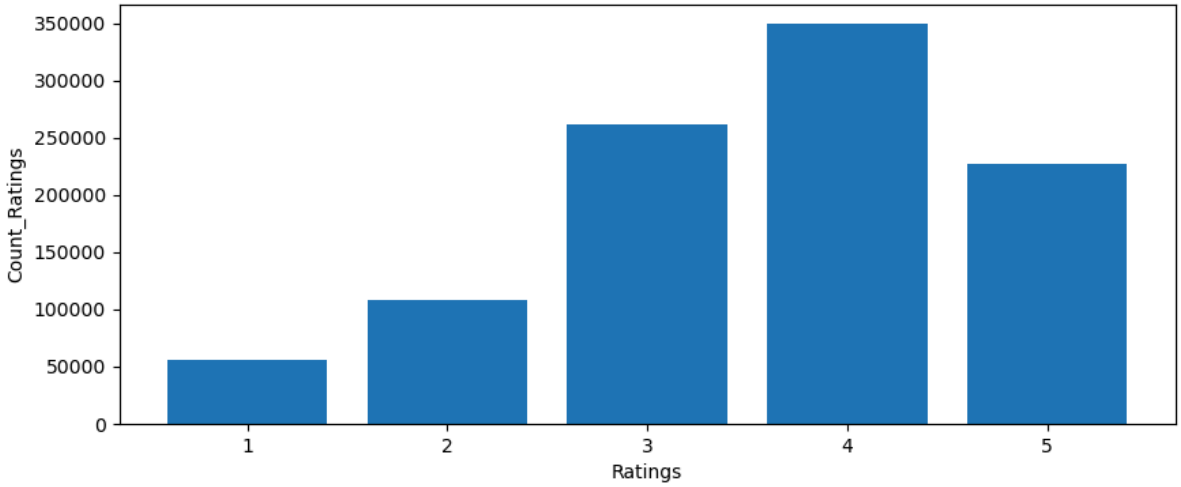
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['watch_hour'].value_counts().sort_index().index,
        df_merged['watch_hour'].value_counts().sort_index().values)
plt.xlabel('Watch_Hour')
plt.ylabel('Count_Ratings')
plt.xticks(rotation=90)
plt.show()
```



Related to Ratings

Count of Rating per Rating Value

In [304…
```
plt.figure(figsize=(10, 4))
plt.bar(df_merged['Rating'].value_counts().sort_index().index,
        df_merged['Rating'].value_counts().sort_index().values)
plt.xlabel('Ratings')
plt.ylabel('Count_Ratings')
# plt.xticks(rotation=90)
plt.show()
```



In [305…
```
df_merged['Rating'].describe()
```

Out[305]:
```
count    1.000209e+06
mean     3.581564e+00
std      1.117102e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: Rating, dtype: float64
```

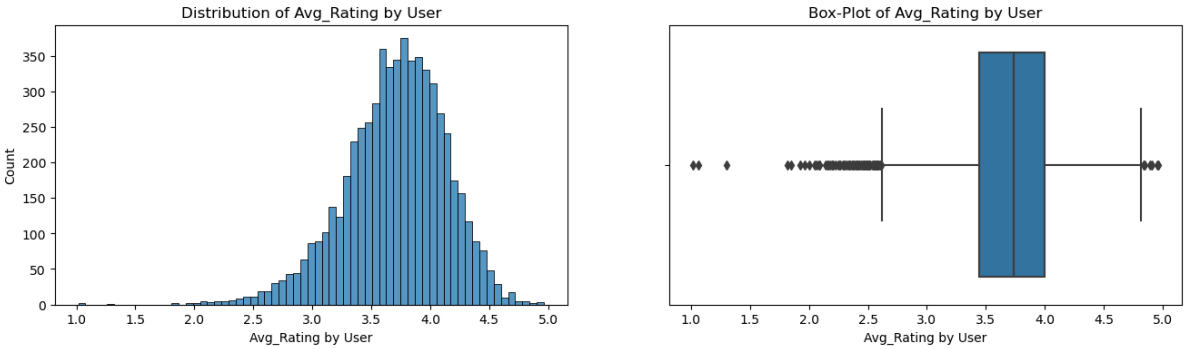
Bivariate Analysis

Average rating by user (User_Overall_Bias)

In [306…
```
mean_rating_by_user = df_merged.groupby('User_ID')['Rating'].mean()
mean_rating_by_user.name = 'Avg_Rating_by_User'
mean_rating_by_user.describe()
```

Out[306]:
```
count    6040.000000
mean        3.702705
std         0.429622
min         1.015385
25%         3.444444
50%         3.735294
75%         4.000000
max         4.962963
Name: Avg_Rating_by_User, dtype: float64
```

In [307…
```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 4))
sns.histplot(mean_rating_by_user, ax=axs[0])
axs[0].set_title('Distribution of Avg_Rating by User')
axs[0].set_xlabel('Avg_Rating by User')
sns.boxplot(x=mean_rating_by_user, ax=axs[1])
axs[1].set_title('Box-Plot of Avg_Rating by User')
axs[1].set_xlabel('Avg_Rating by User')
plt.show()
```

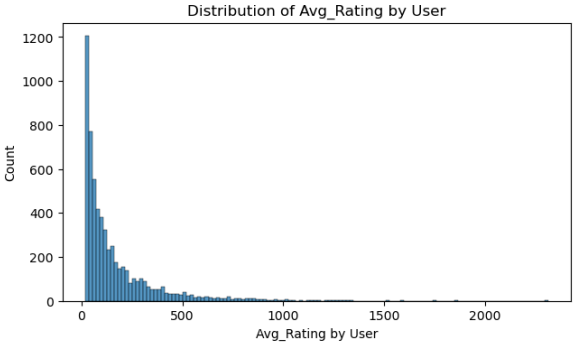


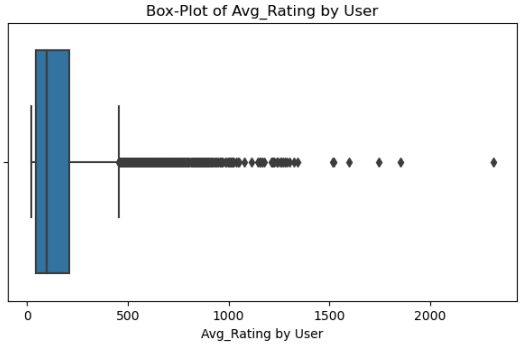
Count of rating by user

In [308…
```
cnt_rating_by_user = df_merged.groupby('User_ID')['Rating'].count()
cnt_rating_by_user.name = 'Count_Rating_by_User'
cnt_rating_by_user.describe()
```

Out[308]:
```
count    6040.000000
mean      165.597517
std       192.747029
min        20.000000
25%        44.000000
50%        96.000000
75%       208.000000
max      2314.000000
Name: Count_Rating_by_User, dtype: float64
```

In [309…
```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 4))
sns.histplot(cnt_rating_by_user, ax=axs[0])
axs[0].set_title('Distribution of Avg_Rating by User')
axs[0].set_xlabel('Avg_Rating by User')
sns.boxplot(x=cnt_rating_by_user, ax=axs[1])
axs[1].set_title('Box-Plot of Avg_Rating by User')
axs[1].set_xlabel('Avg_Rating by User')
plt.show()
```





In [310…
```
# plt.figure(figsize=(8, 4))
# plt.scatter(cnt_rating_by_user, mean_rating_by_user, alpha=0.1)
# plt.ylabel('Mean_Rating_by_User')
# plt.xlabel('Count_Rating_by_User')
```

In [311…
```
# df_merged_fe = pd.merge(df_merged_fe, mean_rating_by_user, left_on='User_I
# df_merged_fe = pd.merge(df_merged_fe, cnt_rating_by_user, left_on='User_ID
# df_merged_fe
```

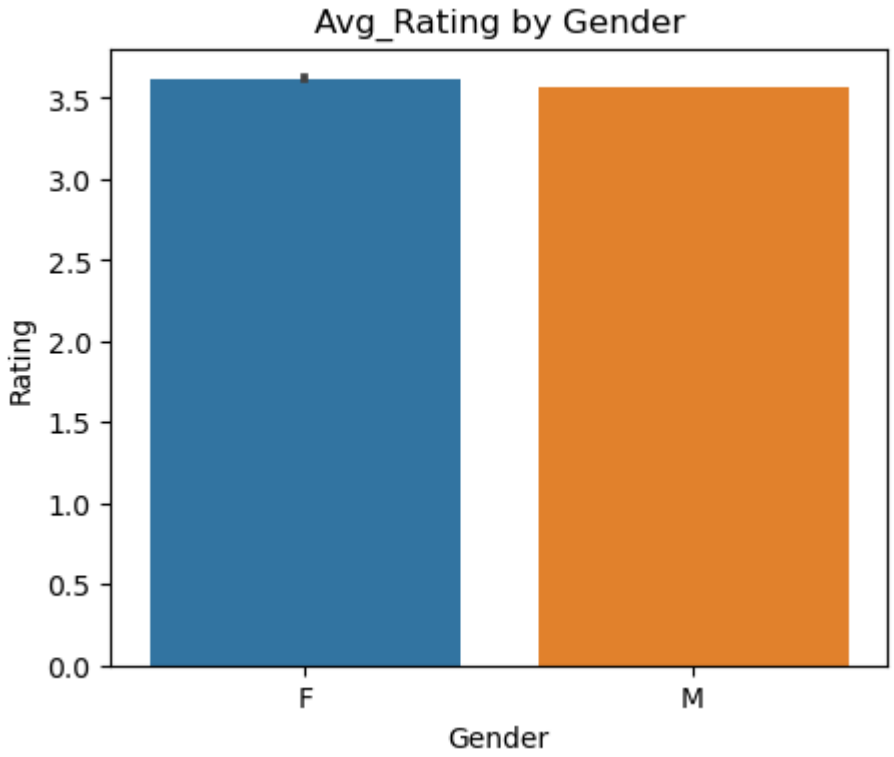
Average rating by gender

In [312…
```
df_merged.groupby('Gender')['Rating'].describe()
```

Out[312]:

Gender	count	mean	std	min	25%	50%	75%	max
F	246440.0	3.620366	1.111228	1.0	3.0	4.0	4.0	5.0
M	753769.0	3.568879	1.118724	1.0	3.0	4.0	4.0	5.0

In [313…
```
fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(5, 4))
sns.barplot(data=df_merged, y='Rating', x='Gender', ax=axs)
axs.set_title('Avg_Rating by Gender')
axs.set_xlabel('Gender')
plt.show()
```



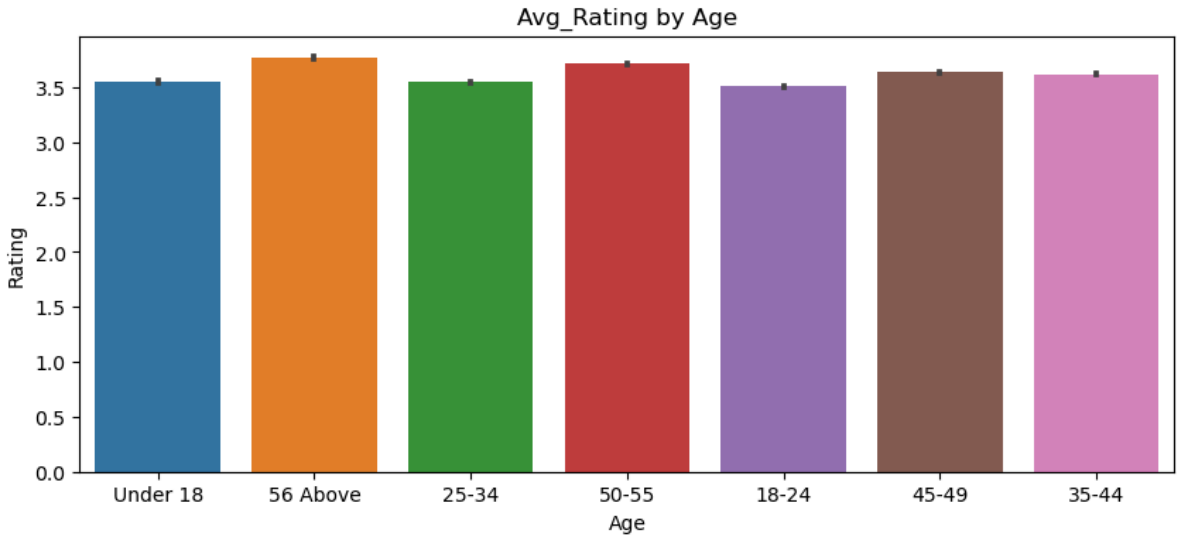
Average rating by age

In [314…
```
df_merged.groupby('Age')['Rating'].describe()
```

Out[314]:

Age	count	mean	std	min	25%	50%	75%	max
18-24	183536.0	3.507573	1.165970	1.0	3.0	4.0	4.0	5.0
25-34	395556.0	3.545235	1.127175	1.0	3.0	4.0	4.0	5.0
35-44	199003.0	3.618162	1.078101	1.0	3.0	4.0	4.0	5.0
45-49	83633.0	3.638062	1.065385	1.0	3.0	4.0	4.0	5.0
50-55	72490.0	3.714512	1.061380	1.0	3.0	4.0	5.0	5.0
56 Above	38780.0	3.766632	1.062551	1.0	3.0	4.0	5.0	5.0
Under 18	27211.0	3.549520	1.208417	1.0	3.0	4.0	4.0	5.0

In [315…
```
fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(10, 4))
sns.barplot(data=df_merged, y='Rating', x='Age', ax=axs)
axs.set_title('Avg_Rating by Age')
axs.set_xlabel('Age')
# axs.tick_params(axis='x', rotation=90)
plt.show()
```



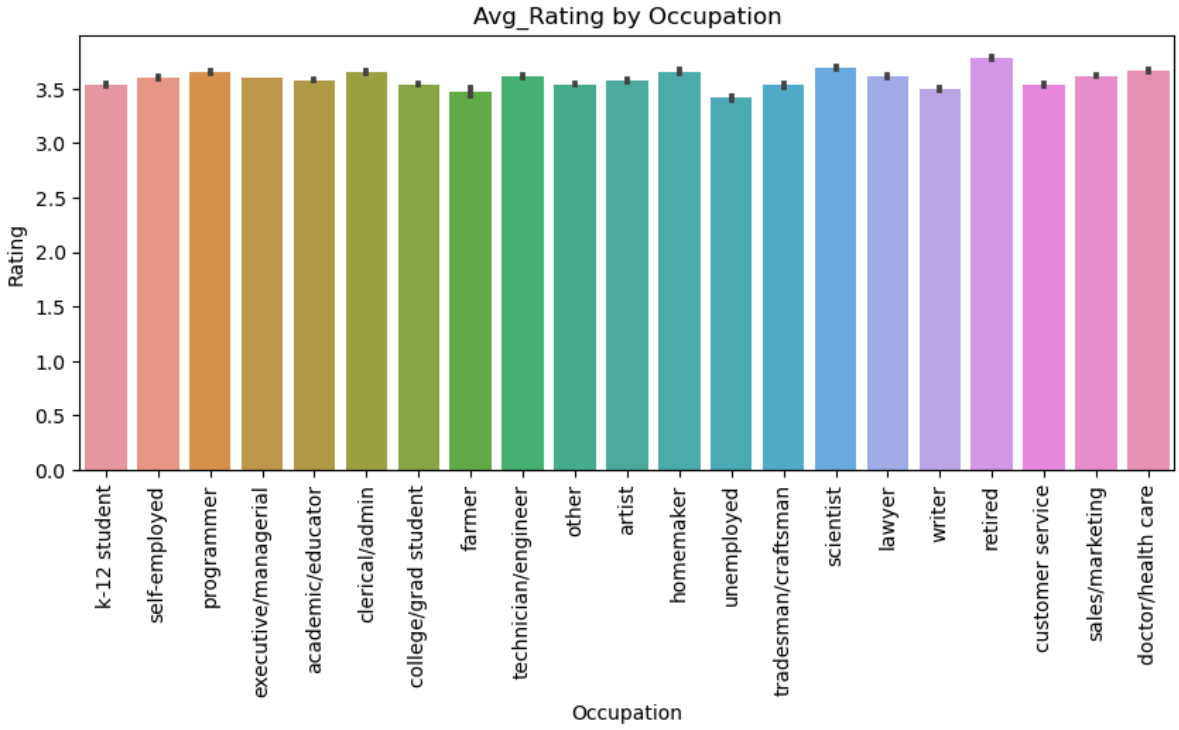
Average rating by occupation

In [316…
```
df_merged.groupby('Occupation')['Rating'].describe()
```

Out[316]:

Occupation	count	mean	std	min	25%	50%	75%	max
academic/educator	85351.0	3.576642	1.107499	1.0	3.0	4.0	4.0	5.0
artist	50068.0	3.573081	1.139391	1.0	3.0	4.0	4.0	5.0
clerical/admin	31623.0	3.656516	1.097272	1.0	3.0	4.0	4.0	5.0
college/grad student	131032.0	3.536793	1.165478	1.0	3.0	4.0	4.0	5.0
customer service	21850.0	3.537529	1.096425	1.0	3.0	4.0	4.0	5.0
doctor/health care	37205.0	3.661578	1.109623	1.0	3.0	4.0	4.0	5.0
executive/managerial	105425.0	3.599772	1.083684	1.0	3.0	4.0	4.0	5.0
farmer	2706.0	3.466741	1.103097	1.0	3.0	4.0	4.0	5.0
homemaker	11345.0	3.656589	1.033470	1.0	3.0	4.0	4.0	5.0
k-12 student	23290.0	3.532675	1.224677	1.0	3.0	4.0	4.0	5.0
lawyer	20563.0	3.617371	1.148651	1.0	3.0	4.0	4.0	5.0
other	130499.0	3.537544	1.126175	1.0	3.0	4.0	4.0	5.0
programmer	57214.0	3.654001	1.083001	1.0	3.0	4.0	4.0	5.0
retired	13754.0	3.781736	1.037604	1.0	3.0	4.0	5.0	5.0
sales/marketing	49109.0	3.618481	1.092451	1.0	3.0	4.0	4.0	5.0
scientist	22951.0	3.689774	1.066078	1.0	3.0	4.0	4.0	5.0
self-employed	46021.0	3.596575	1.074638	1.0	3.0	4.0	4.0	5.0
technician/engineer	72816.0	3.613574	1.075178	1.0	3.0	4.0	4.0	5.0
tradesman/craftsman	12086.0	3.530117	1.065832	1.0	3.0	4.0	4.0	5.0
unemployed	14904.0	3.414050	1.219017	1.0	3.0	4.0	4.0	5.0
writer	60397.0	3.497392	1.154047	1.0	3.0	4.0	4.0	5.0

In [317…
```
fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(10, 4))
sns.barplot(data=df_merged, y='Rating', x='Occupation', ax=axs)
axs.set_title('Avg_Rating by Occupation')
axs.set_xlabel('Occupation')
axs.tick_params(axis='x', rotation=90)
plt.show()
```

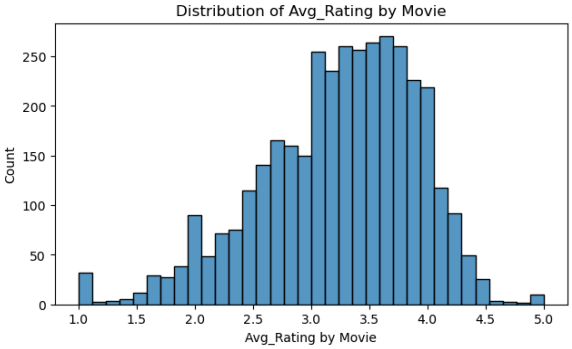


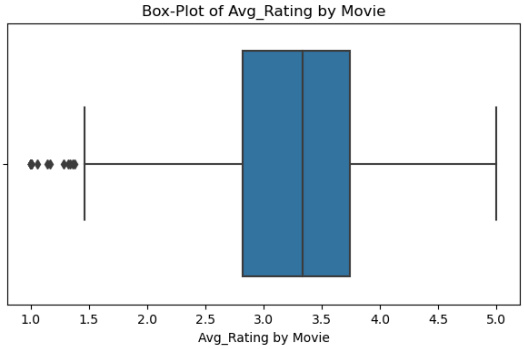
Average rating by movie (Movie_Overall_Bias)

In [318…
```
mean_rating_by_movie = df_merged.groupby('Movie_ID')['Rating'].mean()
mean_rating_by_movie.name = 'Avg_Rating_by_Movie'
mean_rating_by_movie.describe()
```

Out[318]:
```
count    3706.000000
mean        3.238892
std         0.672925
min         1.000000
25%         2.822705
50%         3.331546
75%         3.740741
max         5.000000
Name: Avg_Rating_by_Movie, dtype: float64
```

In [319…
```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 4))
sns.histplot(mean_rating_by_movie, ax=axs[0])
axs[0].set_title('Distribution of Avg_Rating by Movie')
axs[0].set_xlabel('Avg_Rating by Movie')
sns.boxplot(x=mean_rating_by_movie, ax=axs[1])
axs[1].set_title('Box-Plot of Avg_Rating by Movie')
axs[1].set_xlabel('Avg_Rating by Movie')
plt.show()
```



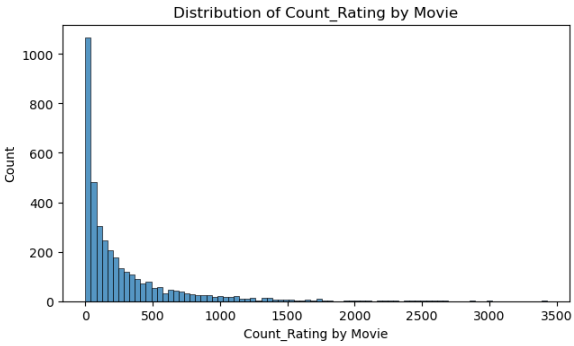


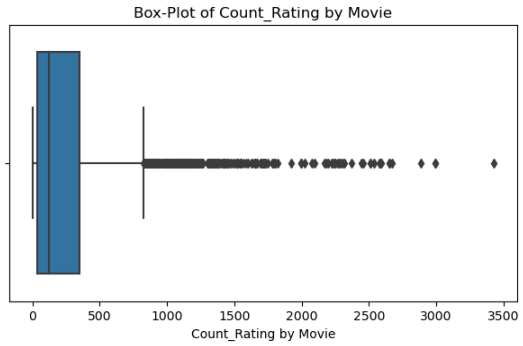
Count of ratings by movie (Movie_Overall_Popularity)

In [320…
```
cnt_rating_by_movie = df_merged.groupby('Movie_ID')['Rating'].count()
cnt_rating_by_movie.name = 'Count_Rating_by_Movie'
cnt_rating_by_movie.describe()
```

Out[320]:
```
count    3706.000000
mean      269.889099
std       384.047838
min         1.000000
25%        33.000000
50%       123.500000
75%       350.000000
max      3428.000000
Name: Count_Rating_by_Movie, dtype: float64
```

In [321…
```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 4))
sns.histplot(cnt_rating_by_movie, ax=axs[0])
axs[0].set_title('Distribution of Count_Rating by Movie')
axs[0].set_xlabel('Count_Rating by Movie')
sns.boxplot(x=cnt_rating_by_movie, ax=axs[1])
axs[1].set_title('Box-Plot of Count_Rating by Movie')
axs[1].set_xlabel('Count_Rating by Movie')
plt.show()
```





Average rating by release_year

In [322…
```
df_merged.groupby('Release_Year')['User_ID'].nunique()
```

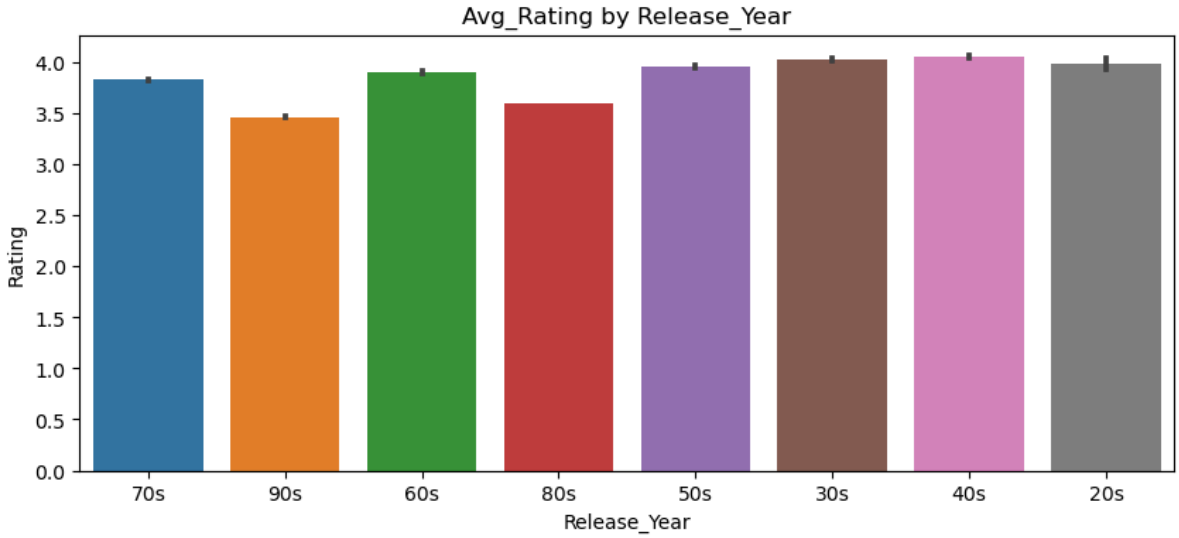
Out[322]:
```
Release_Year
20s     967
30s    3161
40s    3570
50s    4179
60s    4699
70s    5368
80s    5920
90s    6040
Name: User_ID, dtype: int64
```

In [323…
```
df_merged.groupby('Release_Year')['Rating'].describe()
```

Out[323]:

Release_Year	count	mean	std	min	25%	50%	75%	max
20s	1741.0	3.985640	1.059885	1.0	3.0	4.0	5.0	5.0
30s	12648.0	4.024905	0.969005	1.0	3.0	4.0	5.0	5.0
40s	21501.0	4.051160	0.957218	1.0	3.0	4.0	5.0	5.0
50s	35232.0	3.959469	0.984425	1.0	3.0	4.0	5.0	5.0
60s	47188.0	3.902708	0.985047	1.0	3.0	4.0	5.0	5.0
70s	82354.0	3.831641	1.049540	1.0	3.0	4.0	5.0	5.0
80s	224004.0	3.594940	1.092328	1.0	3.0	4.0	4.0	5.0
90s	575541.0	3.462603	1.138483	1.0	3.0	4.0	4.0	5.0

In [324…
```
fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(10, 4))
sns.barplot(data=df_merged, y='Rating', x='Release_Year', ax=axs)
axs.set_title('Avg_Rating by Release_Year')
axs.set_xlabel('Release_Year')
# axs.tick_params(axis='x', rotation=90)
plt.show()
```



Average rating by genres

In [325…
```
genres = ['Action', 'Adventure', 'Animation', 'Children', 'Comedy', 'Crime',
          'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musica
          'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']
```

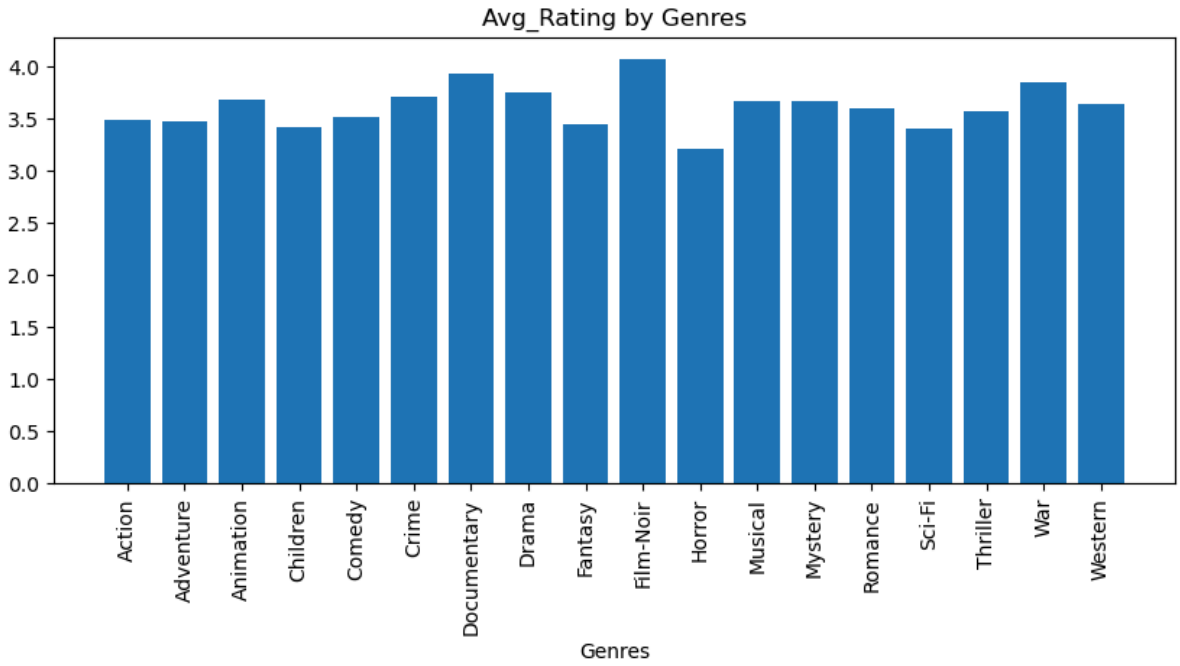
In [326…
```
list_ser_genre = []
for genre in genres:
    ser_genre = df_merged.loc[df_merged[genre]==1, 'Rating'].describe()
    ser_genre.name=genre
    list_ser_genre.append(ser_genre)
```

In [327…
```
pd.DataFrame(list_ser_genre)
```

Out[327]:

	count	mean	std	min	25%	50%	75%	max
Action	256829.0	3.488570	1.133025	1.0	3.0	4.0	4.0	5.0
Adventure	133342.0	3.477666	1.129302	1.0	3.0	4.0	4.0	5.0
Animation	42948.0	3.681149	1.082286	1.0	3.0	4.0	4.0	5.0
Children	71887.0	3.427365	1.158120	1.0	3.0	4.0	4.0	5.0
Comedy	353555.0	3.522711	1.120291	1.0	3.0	4.0	4.0	5.0
Crime	79387.0	3.708781	1.077714	1.0	3.0	4.0	5.0	5.0
Documentary	7799.0	3.930632	1.034756	1.0	3.0	4.0	5.0	5.0
Drama	354849.0	3.760986	1.048219	1.0	3.0	4.0	5.0	5.0
Fantasy	32166.0	3.453678	1.124553	1.0	3.0	4.0	4.0	5.0
Film-Noir	18261.0	4.075188	0.932666	1.0	4.0	4.0	5.0	5.0
Horror	76267.0	3.216450	1.225183	1.0	2.0	3.0	4.0	5.0
Musical	41416.0	3.665105	1.100931	1.0	3.0	4.0	5.0	5.0
Mystery	39550.0	3.674690	1.086100	1.0	3.0	4.0	5.0	5.0
Romance	142275.0	3.596233	1.067592	1.0	3.0	4.0	4.0	5.0
Sci-Fi	142072.0	3.412587	1.154543	1.0	3.0	4.0	4.0	5.0
Thriller	188501.0	3.568161	1.106694	1.0	3.0	4.0	4.0	5.0
War	61056.0	3.857115	1.076818	1.0	3.0	4.0	5.0	5.0
Western	20683.0	3.637770	1.099845	1.0	3.0	4.0	4.0	5.0

In [328…
```
mean_rating_by_genres_lst = [df_merged.loc[df_merged[genre]==1, 'Rating'].me
mean_rating_by_genres = pd.Series(mean_rating_by_genres_lst, index=genres)
# mean_rating_by_genres

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(10, 4))
axs.bar(mean_rating_by_genres.index, mean_rating_by_genres.values)
plt.title('Avg_Rating by Genres')
plt.xlabel('Genres')
plt.xticks(rotation=90)
plt.show()
```



Multivariate Analysis

Capturing genre specific popularity at user level

In [329…
```
df_merged_melt = df_merged.melt(id_vars=['User_ID', 'Movie_ID', 'Rating'],
                                value_vars=['Action', 'Adventure', 'An
                                            'Comedy', 'Crime', 'Documer
                                            'Film-Noir', 'Horror', 'Mus
                                            'Romance', 'Sci-Fi', 'Thri
                                var_name='Genres',
                                value_name='IsGenre'
                               )
df_merged_melt.head()
```

Out[329]:

	User_ID	Movie_ID	Rating	Genres	IsGenre
0	1	1193	5	Action	0
1	2	1193	5	Action	0
2	12	1193	4	Action	0
3	15	1193	4	Action	0
4	17	1193	5	Action	0

In [330…
```
df_user_genre_popularity = pd.crosstab(index=df_merged_melt['User_ID'], colu
                                       values=df_merged_melt['IsGenre'], agg
df_user_genre_popularity.columns.name = 'Prob_User_Genres'
df_user_genre_popularity.head()
```

Out[330]:

Prob_User_Genres	Action	Adventure	Animation	Children	Comedy	Crime	Docum
User_ID							
1	0.043478	0.043478	0.156522	0.173913	0.121739	0.017391	0.0
10	0.095182	0.082256	0.038778	0.074031	0.212691	0.017626	0.0
100	0.300971	0.160194	0.000000	0.004854	0.053398	0.029126	0.0
1000	0.247664	0.074766	0.098131	0.084112	0.074766	0.023364	0.0
1001	0.065549	0.025915	0.028963	0.028963	0.178354	0.039634	0.0

In [ ]:

Capturing genre specific bias at user level

In [331…

```
df_merged_melt_rtng_mean = df_merged_melt.loc[df_merged_melt['IsGenre']==1]
df_user_genre_bias = df_merged_melt_rtng_mean.pivot(index='User_ID', columns
df_user_genre_bias.columns = [col[-1] for col in df_user_genre_bias.columns]
df_user_genre_bias.columns.name = 'Mean_Rating_User_Genres'
df_user_genre_bias.head()
```

Out[331]:

Mean_Rating_User_Genres	Action	Adventure	Animation	Children	Comedy	Crime
User_ID						
1	4.200000	4.000000	4.111111	4.250000	4.142857	4.000000
10	3.913580	4.114286	4.303030	4.142857	4.132597	3.600000
100	2.951613	3.090909	NaN	5.000000	2.727273	3.166667
1000	4.113208	3.937500	4.047619	4.000000	4.125000	4.600000
1001	3.116279	3.235294	3.842105	3.684211	3.547009	3.653846

In [332…

```
# Imputing empty cells with global mean rating
mean_rtng_overall = df_merged['Rating'].mean()
df_user_genre_bias.fillna(mean_rtng_overall, inplace=True)
df_user_genre_bias.head()
```

Out[332]:

Mean_Rating_User_Genres	Action	Adventure	Animation	Children	Comedy	Crime
User_ID						
1	4.200000	4.000000	4.111111	4.250000	4.142857	4.000000
10	3.913580	4.114286	4.303030	4.142857	4.132597	3.600000
100	2.951613	3.090909	3.581564	5.000000	2.727273	3.166667
1000	4.113208	3.937500	4.047619	4.000000	4.125000	4.600000
1001	3.116279	3.235294	3.842105	3.684211	3.547009	3.653846

In [333…

```
# # Cross Join1
# x_test = users_cln1.iloc[0:2].copy()
```

```
# x_test['key'] = 0
# y_test = movie_genre_map.iloc[0:2].reset_index().copy()
# y_test['key'] = 0
# pd.merge(x_test, y_test, on='key', how='inner')

# # Cross Join2
# x_test = users_cln1.iloc[0:2].copy()
# y_test = movie_genre_map.iloc[0:2].reset_index().copy()
# pd.merge(x_test, y_test, how='cross')
```

Generating all possible combinations of User & Movies:

In [334…
```
df_cross_merge_user_movie = pd.merge(users_cln1['User_ID'], movies_cln3['Mov
df_cross_merge_user_movie.shape
```

Out[334]:
```
(23453320, 2)
```

In [335…
```
movie_genre_map_cp1 = movie_genre_map.copy()
movie_genre_map_cp1['Cnt_Genres_by_Movie'] = movie_genre_map_cp1.sum(axis=1)
movie_genre_map_cp1.head(2)
```

Out[335]:

Movie_ID	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama
1	0	0	1	1	1	0	0	0
10	1	1	0	0	0	0	0	0

In [336…
```
df_cross_merge_user_movie_genre = pd.merge(df_cross_merge_user_movie, movie_
df_cross_merge_user_movie_genre.sort_values(by=['User_ID', 'Movie_ID'], inp
df_cross_merge_user_movie_genre.head()
```

Out[336]:

	User_ID	Movie_ID	Action	Adventure	Animation	Children	Comedy	Crime	Do
0	1	1	0	0	1	1	1	0	
54360	1	10	1	1	0	0	0	0	
591920	1	100	0	0	0	0	0	0	
5961480	1	1000	0	0	0	0	0	1	
5967520	1	1001	0	0	0	0	1	0	

Feature: User_Genre_Popular_Bias

- This feature is based on the the user's behavioural pattern since it takes into account 'the most watched' or popular genres of the user
- Thus for any given user U & any given a movie M (and its related genre), we calculate a bias score which denotes the popularity of the movie M in the context of user U

In [337…
```
df_cm_user_gen_pop = pd.merge(df_cross_merge_user_movie_genre, df_user_genre
df_cm_user_gen_pop['User_Genre_Popular_Bias'] = (df_cm_user_gen_pop['Action
                                                 df_cm_user_gen_pop['Adventu
```

```
                                                    df_cm_user_gen_pop['Animat
                                                    df_cm_user_gen_pop['Childre
                                                    df_cm_user_gen_pop['Comedy
                                                    df_cm_user_gen_pop['Crime']
                                                    df_cm_user_gen_pop['Documer
                                                    df_cm_user_gen_pop['Drama']
                                                    df_cm_user_gen_pop['Fantasy
                                                    df_cm_user_gen_pop['Film-No
                                                    df_cm_user_gen_pop['Horror
                                                    df_cm_user_gen_pop['Musica
                                                    df_cm_user_gen_pop['Mystery
                                                    df_cm_user_gen_pop['Romance
                                                    df_cm_user_gen_pop['Sci-Fi
                                                    df_cm_user_gen_pop['Thrille
                                                    df_cm_user_gen_pop['War'] >
                                                    df_cm_user_gen_pop['Western
                                                   )/df_cm_user_gen_pop['Cnt_Ge

df_cm_user_gen_pop.drop(['Action_p', 'Adventure_p','Animation_p', 'Children_
                         'Documentary_p', 'Drama_p', 'Fantasy_p', 'Film-Noi
                         'Mystery_p', 'Romance_p', 'Sci-Fi_p', 'Thriller_p',

df_cm_user_gen_pop.head()
```

Out[337]:

	User_ID	Movie_ID	Action	Adventure	Animation	Children	Comedy	Crime	Do
0	1	1	0	0	1	1	1	0	
54360	1	10	1	1	0	0	0	0	
591920	1	100	0	0	0	0	0	0	
5961480	1	1000	0	0	0	0	0	1	
5967520	1	1001	0	0	0	0	1	0	

Feature: User_Genre_Rating_Bias

- This feature is based on the the user's behavioural pattern since it takes into account 'the most liked' or 'highly rated' genres of the user
- Thus for any given user U & any given a movie M (and its related genre), we calculate a bias score which denotes the rateabilty of the movie M in the context of user U

In [338…

```
df_cm_user_gen_bias = pd.merge(df_cm_user_gen_pop, df_user_genre_bias, left_

df_cm_user_gen_bias['User_Genre_Rating_Bias'] = (df_cm_user_gen_bias['Actior
                                                 df_cm_user_gen_bias['Adver
                                                 df_cm_user_gen_bias['Anima
                                                 df_cm_user_gen_bias['Child
                                                 df_cm_user_gen_bias['Comed
                                                 df_cm_user_gen_bias['Crime
                                                 df_cm_user_gen_bias['Docur
                                                 df_cm_user_gen_bias['Drama
                                                 df_cm_user_gen_bias['Fanta
                                                 df_cm_user_gen_bias['Film-
                                                 df_cm_user_gen_bias['Horro
                                                 df_cm_user_gen_bias['Music
                                                 df_cm_user_gen_bias['Myste
                                                 df_cm_user_gen_bias['Roman
```

```
                                                    df_cm_user_gen_bias['Sci-F
                                                    df_cm_user_gen_bias['Thri
                                                    df_cm_user_gen_bias['War'
                                                    df_cm_user_gen_bias['Weste
                                                   )/df_cm_user_gen_bias['Cnt_

df_cm_user_gen_bias.drop(['Action_b', 'Adventure_b','Animation_b', 'Children
                          'Documentary_b', 'Drama_b', 'Fantasy_b', 'Film-Noi
                          'Mystery_b', 'Romance_b', 'Sci-Fi_b', 'Thriller_b

df_cm_user_gen_bias.head()
```

Out[338]:

	User_ID	Movie_ID	Action	Adventure	Animation	Children	Comedy	Crime	Do
0	1	1	0	0	1	1	1	0	
54360	1	10	1	1	0	0	0	0	
591920	1	100	0	0	0	0	0	0	
5961480	1	1000	0	0	0	0	0	1	
5967520	1	1001	0	0	0	0	1	0	

Features: User_Rating_Bias, Movie_Rating_Bias, Movie_Popular_Bias

- These features are more general and represent the overall bias factors operating at a user and movie level
  - User_Rating_Bias: The mean of all the ratings given by a particular user
  - Movie_Rating_Bias: The mean of all the ratings given for a particular movie
  - Movie_Popular_Bias: The count of the number of times a movie has been watched across all users

In [339…

```
df_user_rtng_bias = pd.DataFrame(mean_rating_by_user)
df_user_rtng_bias.reset_index(inplace=True)
df_user_rtng_bias.rename({'Avg_Rating_by_User':'User_Rating_Bias'}, axis=1,
df_user_rtng_bias.head(2)
```

Out[339]:

	User_ID	User_Rating_Bias
0	1	4.188679
1	10	4.114713

In [340…

```
df_movie_rtng_pop_bias = pd.DataFrame(mean_rating_by_movie)
df_movie_rtng_pop_bias.rename({'Avg_Rating_by_Movie':'Movie_Rating_Bias'}, a
df_movie_rtng_pop_bias['Movie_Popular_Bias'] = 100*cnt_rating_by_movie/cnt_r
df_movie_rtng_pop_bias.reset_index(inplace=True)
df_movie_rtng_pop_bias.head(2)
```

Out[340]:

	Movie_ID	Movie_Rating_Bias	Movie_Popular_Bias
0	1	4.146846	0.207657
1	10	3.540541	0.088781

In [341…
```
print('Unique users:', df_cm_user_gen_bias['User_ID'].nunique())
print('Unique users with atleast 1 or more ratings:', df_user_rtng_bias['Use
print('-'*100)
print('Unique movies:', df_cm_user_gen_bias['Movie_ID'].nunique())
print('Unique movies with atleast 1 or more ratings:', df_movie_rtng_pop_bia
```

```
Unique users: 6040
Unique users with atleast 1 or more ratings: 6040
----------------------------------------------------------------------------
-------------------------
Unique movies: 3883
Unique movies with atleast 1 or more ratings: 3706
```

In [342…
```
df_cm_final = pd.merge(df_cm_user_gen_bias, df_user_rtng_bias, how='left')
df_cm_final = pd.merge(df_cm_final, df_movie_rtng_pop_bias, how='left')
df_cm_final = pd.merge(df_cm_final, movies_cln3[['Movie_ID', 'Release_Year']
df_cm_final.head(2)
```

Out[342]:

	User_ID	Movie_ID	Action	Adventure	Animation	Children	Comedy	Crime	Documenta
0	1	1	0	0	1	1	1	0	
1	1	10	1	1	0	0	0	0	

In [343…
```
df_cm_final = df_cm_final[['User_ID', 'User_Rating_Bias',
                           'Movie_ID', 'Movie_Rating_Bias', 'Movie_Popular_B
                           'Release_Year','Action', 'Adventure', 'Animation
                           'Crime', 'Documentary', 'Drama', 'Fantasy', 'Filn
                           'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War
                           'User_Genre_Rating_Bias', 'User_Genre_Popular_Bia
df_cm_final.head()
```

Out[343]:

	User_ID	User_Rating_Bias	Movie_ID	Movie_Rating_Bias	Movie_Popular_Bias	Release_Y
0	1	4.188679	1	4.146846	0.207657	
1	1	4.188679	10	3.540541	0.088781	
2	1	4.188679	100	3.062500	0.012797	
3	1	4.188679	1000	3.050000	0.002000	
4	1	4.188679	1001	NaN	NaN	

In [344…
```
df_cm_final.isna().sum()
```

Out[344]:
```
User_ID                      0
User_Rating_Bias             0
Movie_ID                     0
Movie_Rating_Bias      1069080
Movie_Popular_Bias     1069080
Release_Year                 0
Action                       0
Adventure                    0
Animation                    0
Children                     0
Comedy                       0
Crime                        0
Documentary                  0
Drama                        0
Fantasy                      0
Film-Noir                    0
Horror                       0
Musical                      0
Mystery                      0
Romance                      0
Sci-Fi                       0
Thriller                     0
War                          0
Western                      0
User_Genre_Rating_Bias       0
User_Genre_Popular_Bias      0
dtype: int64
```

In [345…
```
# Distinct #Movies with no ratings
df_cm_final.loc[df_cm_final['Movie_Rating_Bias'].isna(), 'Movie_ID'].nunique
```

Out[345]:
```
177
```

In [346…
```
# Global mean rating
mean_rtng_overall
```

Out[346]:
```
3.581564453029317
```

In [347…
```
# Movie_Rating_Bias: Imputing those movies which are never rated by any use
df_cm_final['Movie_Rating_Bias'].fillna(mean_rtng_overall, inplace=True)
# Movie_Popular_Bias: Since these movies has never been watched by any user,
df_cm_final['Movie_Popular_Bias'].fillna(mean_rtng_overall, inplace=True)
```

In [348…
```
df_cm_final.isna().sum()
```

Out[348]:
```
User_ID                  0
User_Rating_Bias         0
Movie_ID                 0
Movie_Rating_Bias        0
Movie_Popular_Bias       0
Release_Year             0
Action                   0
Adventure                0
Animation                0
Children                 0
Comedy                   0
Crime                    0
Documentary              0
Drama                    0
Fantasy                  0
Film-Noir                0
Horror                   0
Musical                  0
Mystery                  0
Romance                  0
Sci-Fi                   0
Thriller                 0
War                      0
Western                  0
User_Genre_Rating_Bias   0
User_Genre_Popular_Bias  0
dtype: int64
```

Selecting only those rows for which we have ground truth

- The dataset (df_cm_final) has all possible combinations across all users and all movies
- However a specific user will not have watched all movies, similarly a specific movie will not be watched by all users
- Thus we create a subset df_cm_final retaining only those rows (user & movie combinations) for which we have the ground truth
- This data set can be used for building a regression based recommender system or even in scenarios where ensemble techniques are used

In [349…
```
df_cm_final.head(3)
```

Out[349]:

	User_ID	User_Rating_Bias	Movie_ID	Movie_Rating_Bias	Movie_Popular_Bias	Release_Y
0	1	4.188679	1	4.146846	0.207657	
1	1	4.188679	10	3.540541	0.088781	
2	1	4.188679	100	3.062500	0.012797	

In [350…
```
df_cm_final.shape
```

Out[350]:
```
(23453320, 26)
```

In [351…
```
df_cm_final_gt = pd.merge(df_cm_final, df_merged[['User_ID', 'Movie_ID', 'Ra
df_cm_final_gt.sort_values(by=['User_ID', 'Movie_ID'], inplace=True)
df_cm_final_gt.reset_index(drop=True, inplace=True)
df_cm_final_gt.head(3)
```

Out[351]:

	User_ID	User_Rating_Bias	Movie_ID	Movie_Rating_Bias	Movie_Popular_Bias	Release_
0	1	4.188679	1	4.146846	0.207657	
1	1	4.188679	1022	3.755633	0.057688	
2	1	4.188679	1028	3.894164	0.101079	

In [352…
```
df_cm_final_gt.shape
```

Out[352]:
```
(1000209, 27)
```

Recommender System

In [353…
```
ratings_cln2 = ratings_cln1[['User_ID', 'Movie_ID', 'Rating']].copy()
ratings_cln2['User_ID'] = ratings_cln2['User_ID'].astype('int64')
ratings_cln2['Movie_ID'] = ratings_cln2['Movie_ID'].astype('int64')
ratings_cln2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 3 columns):
 #   Column    Non-Null Count    Dtype
---  ------    --------------    -----
 0   User_ID   1000209 non-null  int64
 1   Movie_ID  1000209 non-null  int64
 2   Rating    1000209 non-null  int64
dtypes: int64(3)
memory usage: 22.9 MB
```

User-Item Interaction Matrix

In [354…
```
rm = ratings_cln2[['User_ID', 'Movie_ID', 'Rating']].pivot(index='User_ID',
rm.head()
```

Out[354]:

Movie_ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
User_ID															
1	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
5	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

In [355…
```
rm.shape
```

Out[355]:
```
(6040, 3706)
```

In [356…
```
sparsity = ((~rm.isna()).sum()).sum()/(rm.shape[0]*rm.shape[1])
print('Sparsity of interaction matrix:', sparsity)
```

```
Sparsity of interaction matrix: 0.044683625622312845
```

Smaller Interaction Matrix: Random 600 users + Top100 popular movies

In [357…
```
top100_popular_movies = ratings_cln2.groupby('Movie_ID')['Rating'].count().s
```

In [358…
```
rm_small = rm.loc[:, top100_popular_movies].sample(600)
rm_small.head()
```

Out[358]:

Movie_ID	2858	260	1196	1210	480	2028	589	2571	1270	593	1580	1198	608
User_ID													
2417	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
505	4.0	4.0	NaN	NaN	4.0	5.0	4.0	5.0	5.0	5.0	4.0	5.0	NaN
5100	4.0	5.0	5.0	5.0	4.0	5.0	5.0	5.0	4.0	5.0	4.0	4.0	5.0
3083	4.0	NaN	5.0	NaN	NaN	NaN	5.0	NaN	NaN	NaN	5.0	NaN	5.0
5245	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [359…
```
rm_small.shape
```

Out[359]:
```
(600, 100)
```

In [360…
```
sparsity_small = ((~rm_small.isna()).sum()).sum()/(rm_small.shape[0]*rm_sma
print('Sparsity of interaction matrix (small):', sparsity_small)
```

```
Sparsity of interaction matrix (small): 0.304
```

Collaborative Filtering (using Interaction Matrix): Pearson Correlation Based

- Usually it is a common practice to impute the sparse interaction matrix with 0s
- However such imputations are ok if we are using metrics like cosine_similarity to quantify similairty
- However, if our choice of similarity metric is pearson correlation coefficient, imputing with 0s can distort the coefficient values
- Hence in this case we will impute the sparse matrix with the overall mean rating

In [361…
```
mean_rtng_overall
```

Out[361]:
```
3.581564453029317
```

In [362…
```
rm1 = rm.copy().fillna(mean_rtng_overall)
rm1
```

Out[362]:

Movie_ID	1	2	3	4	5	6	7	8
User_ID								
1	5.000000	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
2	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
3	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
4	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
5	3.581564	3.581564	3.581564	3.581564	3.581564	2.000000	3.581564	3.581564
...	...	...	...	...	...	...	...	..
6036	3.581564	3.581564	3.581564	2.000000	3.581564	3.000000	3.581564	3.581564
6037	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
6038	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
6039	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564
6040	3.000000	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564	3.581564

6040 rows × 3706 columns

Item Based

In [363…

```
rm1_item_sim = rm1.corr()
rm1_item_sim
```

Out[363]:

Movie_ID	1	2	3	4	5	6	7	
Movie_ID								
1	1.000000	0.015527	0.002277	0.021245	-0.018760	0.033016	0.005899	0
2	0.015527	1.000000	0.059506	0.021200	0.105647	0.006502	0.083732	0
3	0.002277	0.059506	1.000000	0.097239	0.156499	0.037292	0.087474	0
4	0.021245	0.021200	0.097239	1.000000	0.180140	-0.003043	0.055317	0
5	-0.018760	0.105647	0.156499	0.180140	1.000000	0.004174	0.094320	0
...	...	...	...	...	...	...	...	
3948	0.046124	0.045094	0.046648	0.033252	0.049393	0.022927	0.012479	0
3949	0.037766	-0.036210	-0.038425	-0.082715	-0.044831	0.034091	-0.028240	-0
3950	0.020647	-0.029596	0.019643	0.013035	0.003841	-0.010237	0.013107	0
3951	0.014129	-0.008659	0.018385	0.016084	0.011670	-0.004647	0.026387	-0
3952	0.031255	-0.011943	0.011830	0.006241	-0.014079	0.024443	-0.022061	-0

3706 rows × 3706 columns

In [364…

```
movies_title_map_cp = movies_title_map[['Title']].copy()
movies_title_map_cp.reset_index(inplace=True)
movies_title_map_cp['Movie_ID'] = movies_title_map_cp['Movie_ID'].astype('in
movies_title_map_cp.set_index('Movie_ID', inplace=True)
# movies_title_map_cp.head()
```

Examples of Recomendations given for a Movie

In [365…

```
# Example1
item = 1
print('Top3 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm1_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top3 Similar Movies to: Toy Story (1995)
--------------------------------------------------
```

Out[365]:

Movie_ID	Title
3114	Toy Story 2 (1999)
588	Aladdin (1992)
2355	Bug's Life, A (1998)

In [366…
```
# Example2
item = 858
print('Top3 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm1_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top3 Similar Movies to: Godfather, The (1972)
--------------------------------------------------
```

Out[366]:

Movie_ID	Title
1221	Godfather: Part II, The (1974)
1213	GoodFellas (1990)
1208	Apocalypse Now (1979)

In [367…
```
# Example3
item = 10
print('Top3 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm1_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top3 Similar Movies to: GoldenEye (1995)
--------------------------------------------------
```

Out[367]:

Movie_ID	Title
1722	Tomorrow Never Dies (1997)
3082	World Is Not Enough, The (1999)
3639	Man with the Golden Gun, The (1974)

Item Based: Quantifying Performance (Experimental)

In [368…
```
# rm_small1 = rm_small.copy().fillna(mean_rtng_overall)
# rm_small1
```

In [369…
```
# rm_small1_item_sim = rm_small1.corr()
# rm_small1_item_sim
```

In [370…
```
# def predict_rating(user, item, rm1, rm1_item_sim, k=3):

#     topk_similar_movie_corr = rm1_item_sim.loc[item].drop(item).sort_value
#     topk_similar_movie_idxs = rm1_item_sim.loc[item].drop(item).sort_value
#     ratings_similar_movies = rm1.loc[user, topk_similar_movie_idxs]
#     predicted_rating = (ratings_similar_movies.values*topk_similar_movie_

#     return predicted_rating

# predict_rating(5, 6, rm1, rm1_item_sim)
```

In [371…
```
# rm_small1_predict = np.zeros(rm_small1.shape)

# for i in range(0, rm_small1.shape[0]):
#     for j in range(0, rm_small1.shape[1]):
```

```
#             if np.isnan(rm_small.iloc[i, j]):
#                 continue
#             user_id = rm_small1.iloc[i, :].name
#             item_id = rm_small1.iloc[:, j].name
#             rm_small1_predict[i, j] = predict_rating(user_id, item_id, rm_
```

In [372…
```
# rm_small1.shape, rm_small1_predict.shape
```

In [373…
```
# rm_small.values[~rm_small.isna()]
```

In [374…
```
# rm_small1_predict[~rm_small.isna()]
```

In [375…
```
# mse(rm_small.values[~rm_small.isna()], rm_small1_predict[~rm_small.isna()]
```

Collaborative Filtering (using Interaction Matrix): Cosine Similarity Based

In [376…
```
rm2 = rm.copy().fillna(0)
rm2
```

Out[376]:

Movie_ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	1
User_ID																		
1	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
5	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
6036	0.0	0.0	0.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	3.0	4.0	0.
6037	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.
6038	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
6039	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
6040	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.

6040 rows × 3706 columns

Item Based

In [377…
```
rm2_item_sim = pd.DataFrame(cosine_similarity(rm2.T), index=rm2.columns, co
rm2_item_sim
```

Out[377]:

Movie_ID	1	2	3	4	5	6	7	
Movie_ID								
1	1.000000	0.390349	0.267943	0.178789	0.256569	0.347373	0.301490	0.12570
2	0.390349	1.000000	0.240946	0.155457	0.249970	0.244827	0.262772	0.19653
3	0.267943	0.240946	1.000000	0.192788	0.308290	0.187020	0.292230	0.09212
4	0.178789	0.155457	0.192788	1.000000	0.271990	0.125170	0.220024	0.04955
5	0.256569	0.249970	0.308290	0.271990	1.000000	0.148114	0.305107	0.09551
...	...	...	...	...	...	...	...	
3948	0.309676	0.213650	0.190575	0.118902	0.174554	0.236447	0.191689	0.09038
3949	0.186633	0.140781	0.104837	0.096318	0.092403	0.201419	0.117660	0.08052
3950	0.093479	0.087013	0.062258	0.022588	0.051633	0.115331	0.059262	0.08497
3951	0.042829	0.026063	0.010073	0.024769	0.010750	0.029136	0.036102	0.07214
3952	0.182691	0.122185	0.097786	0.095154	0.112835	0.222836	0.138879	0.04552

3706 rows × 3706 columns

In [378…
```
rm2_item_sim.shape
```

Out[378]:
```
(3706, 3706)
```

Examples of Recomendations given for a Movie

In [379…
```
# Example1
item = 1
print('Top5 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm2_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top5 Similar Movies to: Toy Story (1995)
--------------------------------------------------
```

Out[379]:

Movie_ID	Title
3114	Toy Story 2 (1999)
1265	Groundhog Day (1993)
588	Aladdin (1992)
2355	Bug's Life, A (1998)
1270	Back to the Future (1985)

In [380…
```
# Example2
item = 858
print('Top5 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm2_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top5 Similar Movies to: Godfather, The (1972)
--------------------------------------------------
```

Out[380]:

Movie_ID	Title
1221	Godfather: Part II, The (1974)
260	Star Wars: Episode IV - A New Hope (1977)
1196	Star Wars: Episode V - The Empire Strikes Back...
1198	Raiders of the Lost Ark (1981)
608	Fargo (1996)

In [381…

```
# Example3
item = 10
print('Top5 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm2_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top5 Similar Movies to: GoldenEye (1995)
--------------------------------------------------
```

Out[381]:

Movie_ID	Title
1722	Tomorrow Never Dies (1997)
1370	Die Hard 2 (1990)
733	Rock, The (1996)
648	Mission: Impossible (1996)
349	Clear and Present Danger (1994)

User Based

In [382…

```
rm2_user_sim = pd.DataFrame(cosine_similarity(rm2), index=rm2.index, columns
rm2_user_sim
```

Out[382]:

User_ID	1	2	3	4	5	6	7	8
User_ID								
1	1.000000	0.096382	0.120610	0.132455	0.090158	0.179222	0.059678	0.13824
2	0.096382	1.000000	0.151479	0.171176	0.114394	0.100865	0.305787	0.20333
3	0.120610	0.151479	1.000000	0.151227	0.062907	0.074603	0.138332	0.07765
4	0.132455	0.171176	0.151227	1.000000	0.045094	0.013529	0.130339	0.10085
5	0.090158	0.114394	0.062907	0.045094	1.000000	0.047449	0.126257	0.22081
...	...	...	...	...	...	...	...	...
6036	0.186329	0.228241	0.143264	0.170583	0.293365	0.093583	0.122441	0.22740
6037	0.135979	0.206274	0.107744	0.127464	0.172686	0.065788	0.111673	0.14439
6038	0.000000	0.066118	0.120234	0.062907	0.020459	0.065711	0.000000	0.01924
6039	0.174604	0.066457	0.094675	0.064634	0.027689	0.167303	0.014977	0.04466
6040	0.133590	0.218276	0.133144	0.137968	0.241437	0.083436	0.080680	0.14812

6040 rows × 6040 columns

In [383…
```
rm2_user_sim.shape
```

Out[383]: (6040, 6040)

In [384…
```
users_cln2 = users_cln1.copy()
users_cln2['User_ID'] = users_cln2['User_ID'].astype('int64')
users_cln2.set_index('User_ID', inplace=True)
# users_cln2.head()
```

Examples of Recomendations given for a User

In [385…
```
# Example1
item = 1
print('Top5 Similar Users to:')
print('-'*50)
print(users_cln2.loc[[item]])
print('-'*75)
similar_user_idxs = rm2_user_sim.loc[item].drop(item).sort_values(ascending=
print(users_cln2.loc[similar_user_idxs])
```

```
Top5 Similar Users to:
--------------------------------------------------
        Gender       Age    Occupation Zipcode
User_ID
1            F  Under 18  k-12 student   48067
---------------------------------------------------------------------------
        Gender    Age            Occupation Zipcode
User_ID
5343         F  25-34             homemaker   60201
5190         F  35-44                 other   85024
1481         M  35-44   technician/engineer   77096
1283         F  18-24     academic/educator   94607
5705         F  18-24  college/grad student   90024
```

In [386…
```
# Example2
item = 100
```

```
print('Top5 Similar Users to:')
print('-'*50)
print(users_cln2.loc[[item]])
print('-'*75)
similar_user_idxs = rm2_user_sim.loc[item].drop(item).sort_values(ascending=
print(users_cln2.loc[similar_user_idxs])
```

```
Top5 Similar Users to:
--------------------------------------------------
         Gender    Age           Occupation Zipcode
User_ID
100           M  35-44  technician/engineer   95401
---------------------------------------------------------------------------
         Gender    Age            Occupation Zipcode
User_ID
4020          M  25-34    doctor/health care   77034
2186          M  18-24      customer service   41018
778           M  18-24   technician/engineer   32694
729           M  35-44            programmer   85284
3523          M  35-44  executive/managerial   53404
```

Collaborative Filtering: Matrix Factorization

In [387…

```
user_movie_raw = ratings_cln2[['User_ID', 'Movie_ID', 'Rating']].copy()
user_movie_raw.columns = ['UserId', 'ItemId', 'Rating']  # CMFREQ lib requi
user_movie_raw.head(2)
```

Out[387]:

	UserId	ItemId	Rating
0	1	1193	5
1	1	661	3

In [388…

```
print("Shape:", user_movie_raw.shape)
print("No.of Users:", user_movie_raw['UserId'].nunique())
print("No.of Items:", user_movie_raw['UserId'].nunique())
```

```
Shape: (1000209, 3)
No.of Users: 6040
No.of Items: 6040
```

In [389…

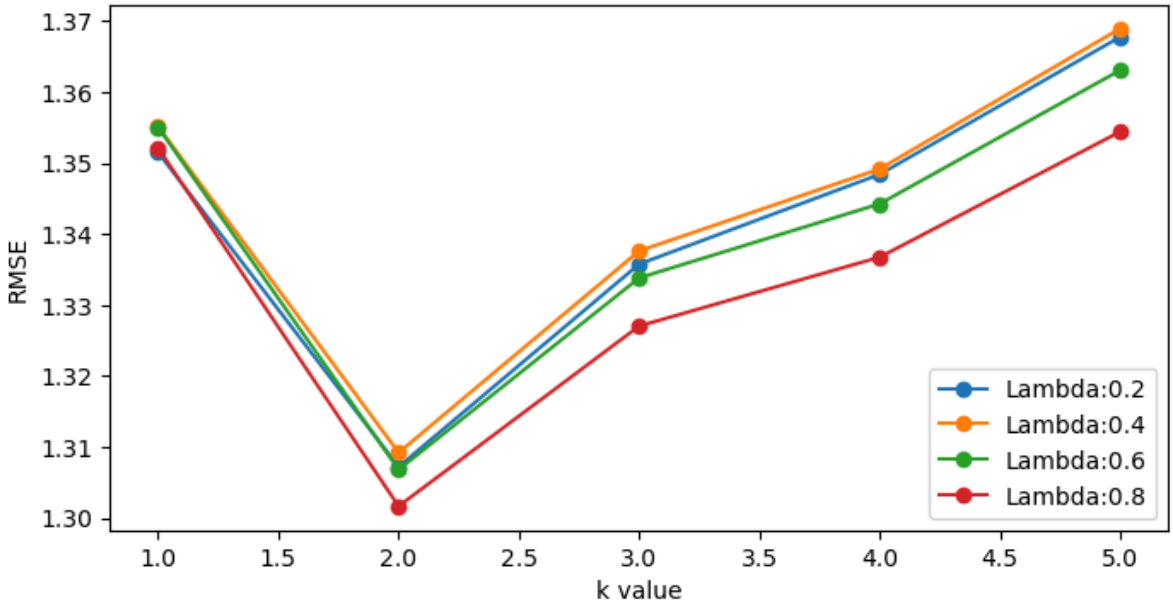
```
k_values = {}
rmse_values = {}
for lamb in np.arange(0, 1, 0.2):
    lamb = np.round(lamb, 1)
    k_values[lamb] = []
    rmse_values[lamb] = []
    print('Lambda:', lamb)
    print('-'*75)
    for k in range(1, 6):
        model = CMF(method="als", k=k, lambda_=lamb, user_bias=False, item_b
        model.fit(user_movie_raw)
        rm_predict = np.dot(model.A_, model.B_.T) + model.glob_mean_

        k_values[lamb].append(k)
        rmse_val = mse(rm.values[~rm.isna()], rm_predict[~rm.isna()])**0.5
        rmse_values[lamb].append(rmse_val)

        print('K value:', k)
        print('RMSE:', rmse_val)
        print('-'*50)
```

```
Lambda: 0.0
--------------------------------------------------------------------------------
K value: 1
RMSE: 315.89820983585577
--------------------------------------------------
K value: 2
RMSE: 18.604744214183327
--------------------------------------------------
K value: 3
RMSE: 44.16604410405676
--------------------------------------------------
K value: 4
RMSE: 4.398419658108807
--------------------------------------------------
K value: 5
RMSE: 1.807819315706532
--------------------------------------------------
Lambda: 0.2
--------------------------------------------------------------------------------
K value: 1
RMSE: 1.3514910493531889
--------------------------------------------------
K value: 2
RMSE: 1.3071860851977433
--------------------------------------------------
K value: 3
RMSE: 1.3357527558945386
--------------------------------------------------
K value: 4
RMSE: 1.34840416447673
--------------------------------------------------
K value: 5
RMSE: 1.367774190289871
--------------------------------------------------
Lambda: 0.4
--------------------------------------------------------------------------------
K value: 1
RMSE: 1.3551732240458125
--------------------------------------------------
K value: 2
RMSE: 1.3091821532935732
--------------------------------------------------
K value: 3
RMSE: 1.3376075393952749
--------------------------------------------------
K value: 4
RMSE: 1.349168327965947
--------------------------------------------------
K value: 5
RMSE: 1.3689596189929532
--------------------------------------------------
Lambda: 0.6
--------------------------------------------------------------------------------
K value: 1
RMSE: 1.3550255367859383
--------------------------------------------------
K value: 2
RMSE: 1.3067879360387464
--------------------------------------------------
K value: 3
RMSE: 1.333804636536503
--------------------------------------------------
K value: 4
RMSE: 1.3442466158175466
```

```
--------------------------------------------------
K value: 5
RMSE: 1.3630821061094816
--------------------------------------------------
Lambda: 0.8
--------------------------------------------------------------------------------
K value: 1
RMSE: 1.3521190355579715
--------------------------------------------------
K value: 2
RMSE: 1.3016349475953861
--------------------------------------------------
K value: 3
RMSE: 1.3270216150498662
--------------------------------------------------
K value: 4
RMSE: 1.3367303268311526
--------------------------------------------------
K value: 5
RMSE: 1.3544221030885133
--------------------------------------------------
```

In [390…

```
plt.figure(figsize=(8, 4))
for key in k_values:
    if key!=0:
        plt.plot(k_values[key], rmse_values[key], label=f'Lambda:{key}', ma

plt.xlabel('k value')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```



Observations:

- d=2 & Lambda=0.8 seems to be the best combination which achieved the lowest RMSE
- However we want to proceed with d=4 & Lmabda=0.8 (No logic here, just experimenting!)

In [391…

```
model = CMF(method="als", k=4, lambda_=0.8, user_bias=False, item_bias=False
model.fit(user_movie_raw)
```

Out[391]:
```
Collective matrix factorization model
(explicit-feedback variant)
```

In [392…
```
model.A_.shape, model.B_.T.shape
```

Out[392]:
```
((6040, 4), (4, 3706))
```

In [393…
```
user_movie_raw.Rating.mean(), model.glob_mean_
```

Out[393]:
```
(3.581564453029317, 3.581564426422119)
```

In [394…
```
rm_predict = np.matmul(model.A_, model.B_.T) + model.glob_mean_
df_rm_predict = pd.DataFrame(rm_predict, index=rm.index, columns=rm.columns)
rm_predict
```

Out[394]:
```
array([[4.0729756, 3.8602662, 4.3779488, ..., 3.5075934, 3.8226469,
        3.6209924],
       [4.219084 , 2.8883936, 4.6355476, ..., 3.0789406, 4.3790426,
        3.6565714],
       [4.028888 , 3.0897815, 3.6879978, ..., 2.171103 , 4.104062 ,
        3.6488369],
       ...,
       [4.0688944, 3.4428487, 4.0918393, ..., 3.490091 , 3.816106 ,
        3.6603944],
       [4.3060856, 3.3458445, 4.1515956, ..., 3.3004184, 3.8801773,
        3.7107055],
       [4.516684 , 3.1514907, 3.5695343, ..., 4.3034716, 3.297753 ,
        3.859365 ]], dtype=float32)
```

In [395…
```
print('Original Interaction Matrix:', rm.shape)
print('Predicted Interaction Matrix:', rm_predict.shape)
```

```
Original Interaction Matrix: (6040, 3706)
Predicted Interaction Matrix: (6040, 3706)
```

In [396…
```
print('RMSE:', mse(rm.values[~rm.isna()], rm_predict[~rm.isna()])**0.5)
```

```
RMSE: 1.3367303268311526
```

In [397…
```
print('MAPE:', mape(rm.values[~rm.isna()], rm_predict[~rm.isna()]))
```

```
MAPE: 0.3774345350419918
```

How to find the top recommendations for any user:

- Let us take the example of User 5145 (We can do this for any user)

In [398…
```
# # Movies watched by User1 along with ratings
# rm.loc[5145].loc[~rm.loc[5145].isna()].sort_values(ascending=False)
```

In [399…
```
# Movie titles watched by user 5145
movies_title_map_cp.loc[rm.loc[5145].loc[~rm.loc[5145].isna()].sort_values(a
```

Out[399]:

Movie_ID	Title
3742	Battleship Potemkin, The (Bronenosets Potyomki...
2959	Fight Club (1999)
2762	Sixth Sense, The (1999)
2858	American Beauty (1999)
2997	Being John Malkovich (1999)
3176	Talented Mr. Ripley, The (1999)
3145	Cradle Will Rock, The (1999)
3052	Dogma (1999)
2152	Air Bud: Golden Receiver (1998)
2891	Happy, Texas (1999)
3081	Sleepy Hollow (1999)
3147	Green Mile, The (1999)
3175	Galaxy Quest (1999)
1518	Breakdown (1997)
3082	World Is Not Enough, The (1999)
3113	End of Days (1999)
2267	Mortal Thoughts (1991)
2907	Superstar (1999)
1845	Zero Effect (1998)
3053	Messenger: The Story of Joan of Arc, The (1999)

In [400…

```
# Top10 recommendations for User 5145
movies_title_map_cp.loc[df_rm_predict.loc[5145].loc[rm.loc[5145].isna()].so
```

Out[400]:

Movie_ID	Title
3331	My Tutor (1983)
3342	Birdy (1984)
2235	One Man's Hero (1999)
2213	Waltzes from Vienna (1933)
3654	Guns of Navarone, The (1961)
3660	Puppet Master (1989)
688	Operation Dumbo Drop (1995)
2814	Bat, The (1959)
2424	You've Got Mail (1998)
1627	U Turn (1997)

In [401…
```
# # Top10 recommendations for User 5145 using topN method --> ideally should
# movies_title_map_cp.loc[model.topN(user=5145, n=10)]
```

Collaborative Filtering: (using Latent Matrices post MF)

Item Based: Using Latent Matrix for Movies (d=4)

In [402…
```
item_latent_d4 = model.B_
item_latent_d4
```

Out[402]:
```
array([[ 0.1915074 , -1.10358   , -0.35639656,  0.13128413],
       [ 0.07175454, -0.36977997,  0.6010349 ,  0.38018137],
       [-0.6346922 , -1.0043632 , -0.00551532, -0.43241844],
       ...,
       [ 0.3407038 , -0.04203251,  1.105523  , -0.85018444],
       [-0.5075365 , -0.14688273, -0.4393353 , -0.14947638],
       [ 0.1127795 , -0.17441656, -0.06933313,  0.02275571]],
      dtype=float32)
```

In [403…
```
item_latent_d4.shape
```

Out[403]:
```
(3706, 4)
```

In [404…
```
mf_d4_item_sim = pd.DataFrame(cosine_similarity(item_latent_d4), index=rm.c
mf_d4_item_sim.head()
```

Out[404]:

Movie_ID	1	2	3	4	5	6	7	
Movie_ID								
1	1.000000	0.270563	0.623284	0.439669	0.712570	0.923004	0.710485	0.95488
2	0.270563	1.000000	0.155415	0.056569	0.368033	0.098797	0.043221	0.2046
3	0.623284	0.155415	1.000000	0.877790	0.922257	0.654342	0.947669	0.7849
4	0.439669	0.056569	0.877790	1.000000	0.905456	0.645658	0.927858	0.6842
5	0.712570	0.368033	0.922257	0.905456	1.000000	0.785781	0.941889	0.8654

In [405…
```
mf_d4_item_sim.shape
```

Out[405]:
```
(3706, 3706)
```

Examples of Recommendations given for a Movie

In [406…
```
# Example1
item = 1
print('Top5 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = mf_d4_item_sim.loc[item].drop(item).sort_values(ascend
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top5 Similar Movies to: Toy Story (1995)
--------------------------------------------------
```

Out[406]:

Movie_ID	Title
22	Copycat (1995)
549	Thirty-Two Short Films About Glenn Gould (1993)
2754	Deadtime Stories (1987)
553	Tombstone (1993)
448	Fearless (1993)

In [407…
```
# Example2
item = 858
print('Top5 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = mf_d4_item_sim.loc[item].drop(item).sort_values(ascend
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top5 Similar Movies to: Godfather, The (1972)
--------------------------------------------------
```

Out[407]:

Movie_ID	Title
3698	Running Man, The (1987)
2644	Dracula (1931)
2140	Dark Crystal, The (1982)
3050	Light It Up (1999)
1806	Paulie (1998)

Model Evaluation: Experimental

In [408…
```
# # Upon Matrix Completion: Ratings For User 1
# np.dot(model.A_[0], model.B_.T)
```

In [409…
```
# # Top10 Movies recommeded by engine
# mov_reco = np.argsort((np.dot(model.A_[0], model.B_.T))+model.glob_mean_)
# mov_reco
```

In [410…
```
# # movies actually seen by user
# mov_interact = ratings_cln2.loc[ratings_cln2['User_ID']==1, 'Movie_ID'].va
# mov_interact
```

In [411…
```
# # Overall: How many movies out of the 40 recommended movies has the user a
# set(mov_reco).intersection(set(mov_interact))
```

User Based: Using Latent Matrix for Users (d=4)

In [412…
```
user_latent_d4 = model.A_
user_latent_d4
```

Out[412]:
```
array([[-0.4466898 , -0.5507423 ,  0.12025386,  0.09159722],
       [-0.43711087, -0.44867653, -0.907722  , -0.74213314],
       [-0.20742509, -0.08313342, -1.011801  ,  0.26431417],
       ...,
       [-0.01818586, -0.3808764 , -0.29536688, -0.26494044],
       [ 0.10105851, -0.5028569 , -0.5290204 , -0.29185537],
       [ 1.5068572 , -0.51305264, -0.58820194, -0.98475206]],
      dtype=float32)
```

In [413…
```
user_latent_d4.shape
```

Out[413]:
```
(6040, 4)
```

In [414…
```
mf_d4_user_sim = pd.DataFrame(cosine_similarity(user_latent_d4), index=rm.ir
mf_d4_user_sim.head()
```

Out[414]:

User_ID	1	2	3	4	5	6	7	
User_ID								
1	1.000000	0.275178	0.052850	0.211678	-0.623089	0.345141	0.841349	0.443
2	0.275178	1.000000	0.598120	0.578122	0.128722	0.819345	0.346082	0.182
3	0.052850	0.598120	1.000000	0.092747	-0.019293	0.279105	0.506546	0.544
4	0.211678	0.578122	0.092747	1.000000	0.619011	0.180089	0.233024	0.431
5	-0.623089	0.128722	-0.019293	0.619011	1.000000	-0.267008	-0.466072	0.054

In [415…
```
mf_d4_user_sim.shape
```

Out[415]:
```
(6040, 6040)
```

Examples of Recommendations given for a User

In [416…
```
# Example1
item = 1
print(f'Top5 Similar Users to: User{item}')
print('-'*50)
print(users_cln2.loc[[item]])
print('-'*75)
similar_user_idxs = mf_d4_user_sim.loc[item].drop(item).sort_values(ascendi
print(users_cln2.loc[similar_user_idxs])
```

```
Top5 Similar Users to: User1
--------------------------------------------------
        Gender       Age    Occupation Zipcode
User_ID
1            F  Under 18  k-12 student   48067
---------------------------------------------------------------------------
        Gender       Age          Occupation Zipcode
User_ID
3699         F  56 Above      clerical/admin   30127
4816         F     45-49  doctor/health care   04240
3840         F     25-34               other   02176
343          F     35-44      clerical/admin   55127
5363         M  56 Above             retired   30673
```

Regression Based

In [417…
```
df_cm_final_gt.shape
```

Out[417]:
```
(1000209, 27)
```

In [418…
```
df_cm_final_gt.head()
```

Out[418]:

	User_ID	User_Rating_Bias	Movie_ID	Movie_Rating_Bias	Movie_Popular_Bias	Release_
0	1	4.188679	1	4.146846	0.207657	
1	1	4.188679	1022	3.755633	0.057688	
2	1	4.188679	1028	3.894164	0.101079	
3	1	4.188679	1029	3.688380	0.056788	
4	1	4.188679	1035	3.931973	0.088182	

In [419…
```
# Working with a smaller sample of origianl data
df_cm_final_gt_sample = df_cm_final_gt.sample(50000).copy()
df_cm_final_gt_sample.shape
```

Out[419]:
```
(50000, 27)
```

Stratified Train-Test Split

In [420…
```
X = df_cm_final_gt_sample.drop(['User_ID', 'Movie_ID', 'Release_Year', 'Rat
y = df_cm_final_gt_sample['Rating']
```

In [421…
```
y.value_counts(normalize=True)
```

Out[421]:
```
4    0.34978
3    0.26196
5    0.22734
2    0.10610
1    0.05482
Name: Rating, dtype: float64
```

In [422…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
print('Train data:', X_train.shape, y_train.shape)
print('Train data:', X_test.shape, y_test.shape)
```

```
Train data: (35000, 23) (35000,)
Train data: (15000, 23) (15000,)
```

In [423…
```
y_train.value_counts(normalize=True)
```

Out[423]:
```
4    0.349771
3    0.261971
5    0.227343
2    0.106086
1    0.054829
Name: Rating, dtype: float64
```

In [424…
```
y_test.value_counts(normalize=True)
```

Out[424]:
```
4    0.349800
3    0.261933
5    0.227333
2    0.106133
1    0.054800
Name: Rating, dtype: float64
```

Scaling Data

In [425…
```
std_scaler = StandardScaler()
X_train_scaled = std_scaler.fit_transform(X_train)
X_test_scaled = std_scaler.transform(X_test)
```

Fitiing a Random Forest Regressor

In [426…
```
mse_scorer = make_scorer(mse)
rfr1 = RandomForestRegressor(random_state=42)
cross_val_rfr1 = cross_val_score(rfr1, X_train_scaled, y_train, cv=5, scorir
```

In [427…
```
print('RMSE:', cross_val_rfr1**0.5)
print('-'*70)
pd.Series(cross_val_rfr1**0.5).describe()
```

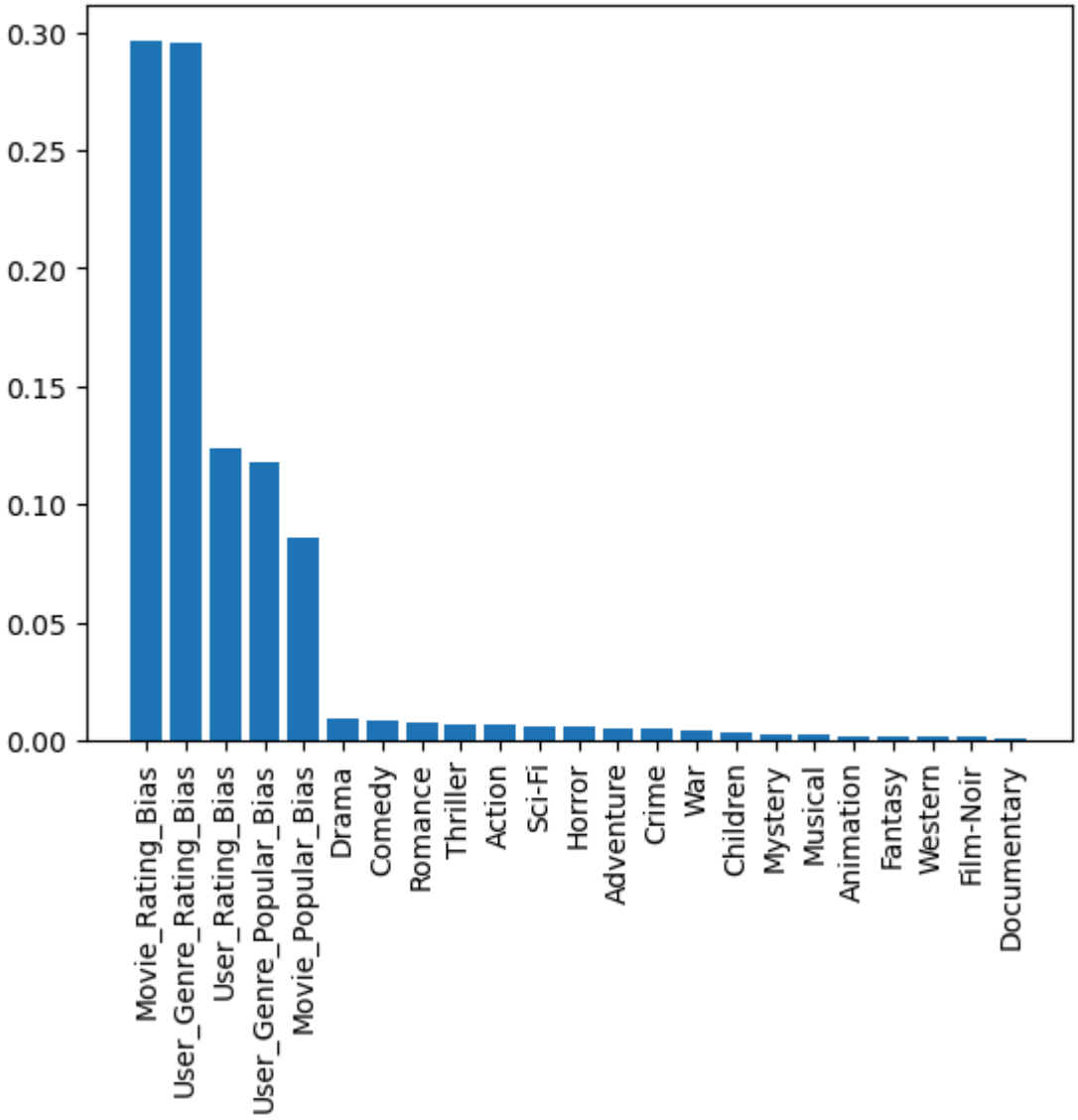
```
RMSE: [0.89851094 0.88975928 0.90556148 0.89349839 0.88380277]
----------------------------------------------------------------------
```

Out[427]:
```
count    5.000000
mean     0.894227
std      0.008306
min      0.883803
25%      0.889759
50%      0.893498
75%      0.898511
max      0.905561
dtype: float64
```

Feature Importances

- All 5 of our self created features turned out to be important -- Job well done!

In [428…
```
rfr1.fit(X_train_scaled, y_train)
ser_feat_imp = pd.Series(dict(zip(X_train.columns, rfr1.feature_importances_
plt.bar(ser_feat_imp.index, ser_feat_imp.values)
plt.xticks(rotation=90)
plt.show()
```



Questions and Answers:

- Q: Users of which age group have watched and rated the most number of movies?
  - A: age group 25-35
- Q: Users belonging to which profession have watched and rated the most movies?
  - A:College Graduate Students and Other category
- Q: Most of the users in our dataset who've rated the movies are Male. (T/F)
  - A: Male
- Q: Most of the movies present in our dataset were released in which decade?
  - A: 90s
- Q: The movie with maximum no. of ratings is ___.
  - A: American Beauty
- Q: Name the top 3 movies similar to 'Liar Liar' on the item-based approach.
  - A: Shown below

- Q: On the basis of approach, Collaborative Filtering methods can be classified into - ***based and*** -based.
  - A: Memory based and Model based
- Q: Pearson Correlation ranges between ***to*** whereas, Cosine Similarity belongs to the interval between ***to*** .
  - A: Pearson Correlation ranges between -1 to +1
- Q: Cosine Similarity belongs to the interval between -1 to 1
  - A: Cosine Similarity ranges between -1 to +1
- Q: Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.
  - RMSE: 1.33, MAPE: 0.38
- Q: Give the sparse 'row' matrix representation for the following dense matrix: [[1 0], [3 7]]
  - A: Shown below

In [429…
```
# Finding movies similar to 'Liar Liar'
movies_title_map_cp.loc[movies_title_map_cp['Title'].str.split().apply(lambd
```

Out[429]:

Movie_ID	Title
1485	Liar Liar (1997)
2882	Jakob the Liar (1999)

In [430…
```
# Approach: CF using Original Interaction Matrix (Item-Based)
item = 1485
print('Top3 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = rm2_item_sim.loc[item].drop(item).sort_values(ascending
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top3 Similar Movies to: Liar Liar (1997)
--------------------------------------------------
```

Out[430]:

Movie_ID	Title
500	Mrs. Doubtfire (1993)
344	Ace Ventura: Pet Detective (1994)
231	Dumb & Dumber (1994)

In [431…
```
# Approach: CF using Latent Matrices (Item-Based)
item = 1485
print('Top3 Similar Movies to:', movies_title_map_cp.loc[item][0])
print('-'*50)
similar_movie_idxs = mf_d4_item_sim.loc[item].drop(item).sort_values(ascend
movies_title_map_cp.loc[similar_movie_idxs]
```

```
Top3 Similar Movies to: Liar Liar (1997)
--------------------------------------------------
```

Out[431]:

Movie_ID	Title
2271	Permanent Midnight (1998)
988	Grace of My Heart (1996)
2810	Perfect Blue (1997)

In [432…

```
# sparse ‘row’ matrix representation for the following dense matrix: [[1 0],

from scipy.sparse import csr_matrix
dense_matrix = [[1,0],
                [3,7]]

sparse_matrix = csr_matrix(dense_matrix)
print(sparse_matrix.data)
print(sparse_matrix.indices)
print(sparse_matrix.indptr)
```

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

In [ ]:

In [ ]:

In [ ]: