BusinessCase

Lohith Kumar Kasula

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

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

Gender is denoted by a "M" for male and "F" for female

Age is chosen from the following ranges:

- 1: "Under 18"
- 18: "18-24"
- 25: "25-34"
- 35: "35-44"

- 45: "45-49"
- 50: "50-55"
- 56: "56+"

Occupation is chosen from the following choices:

- 0: "other" or not specified
- 1: "academic/educator"
- 2: "artist"
- 3: "clerical/admin"
- 4: "college/grad student"
- 5: "customer service"
- 6: "doctor/health care"
- 7: "executive/managerial"
- 8: "farmer"
- 9: "homemaker"
- 10: "K-12 student"
- 11: "lawyer"
- 12: "programmer"
- 13: "retired"
- 14: "sales/marketing"
- 15: "scientist"
- 16: "self-employed"
- 17: "technician/engineer"
- 18: "tradesman/craftsman"
- 19: "unemployed"
- 20: "writer"

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

Concepts Tested:

- Recommender Engine
- Collaborative Filtering (Item-based & User-based Approach)
- Pearson Correlation
- Nearest Neighbors using Cosine Similarity
- Matrix Factorization

Importing all the requried packages

In [ ]:

```
!pip install cmfrec
```

```
Requirement already satisfied: cmfrec in /usr/local/lib/python3.11/dist-packages
(3.5.1.post11)
Requirement already satisfied: cython in /usr/local/lib/python3.11/dist-packages
(from cmfrec) (3.0.12)
Requirement already satisfied: numpy>=1.25 in /usr/local/lib/python3.11/dist-packa
ges (from cmfrec) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (f
rom cmfrec) (1.14.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from cmfrec) (2.2.2)
Requirement already satisfied: findblas in /usr/local/lib/python3.11/dist-packages
(from cmfrec) (0.1.26.post1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
1/dist-packages (from pandas->cmfrec) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pack
ages (from pandas->cmfrec) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-pa
ckages (from pandas->cmfrec) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages
(from python-dateutil>=2.8.2->pandas->cmfrec) (1.17.0)
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from google.colab import drive
import chardet
import pickle
import re
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
from sklearn.neighbors import NearestNeighbors
from cmfrec import CMF
from sklearn.metrics import (r2_score, mean_squared_error as mse,
                             mean_absolute_error as mae,
                             root_mean_squared_error as rmse,
                              mean_absolute_percentage_error as mape)
```

In [ ]:

```
pd.set_option('display.float_format', '{:.3f}'.format)
pd.set_option('display.max_columns', None)
drive.mount('/content/drive')
path = '/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/12_Zee_Recom
for i in ['zee-movies', 'zee-ratings', 'zee-users']:
  with open(path.format(i), 'rb') as f:
      result = chardet.detect(f.read(10000))
      print(result)
```

```
Mounted at /content/drive
{'encoding': 'ISO-8859-1', 'confidence': 0.73, 'language': ''}
{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}
{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}
```

In [ ]:

```
df_movies = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessC
df_ratings = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/Business
df_users = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCa
```

Data Cleaning

In [ ]:
```
df_movies.head()
```

Out[ ]:

	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 [ ]:
```
df_movies['Genres'] = df_movies['Genres'].str.split('|')
df_exploded = df_movies.explode('Genres')
df_exploded.head()
```

Out[ ]:

	Movie ID	Title	Genres
0	1	Toy Story (1995)	Animation
0	1	Toy Story (1995)	Children's
0	1	Toy Story (1995)	Comedy
1	2	Jumanji (1995)	Adventure
1	2	Jumanji (1995)	Children's

In [ ]:
```
def get_only_title(title):
  return [title.rsplit(' ', 1)[0], title.rsplit(' ', 1)[1].strip('()')]


df_movies[['MovieTitle', 'ReleaseYear']] = df_movies.loc[:,'Title'].apply(get_only_
df_movies.drop(columns=['Title'], inplace=True)
df_movies.rename({'MovieTitle':'Title'}, axis=1, inplace=True)
```

In [ ]:
```
df_movies.head()
```

Out[ ]:

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

In [ ]:
```
genre_dummies = pd.get_dummies(df_exploded['Genres']).groupby(df_exploded.index).su
genre_dummies.head(5)
```

Out[ ]:

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-No
0	0	0	1	1	1	0	0	0	0	
1	0	1	0	1	0	0	0	0	1	
2	0	0	0	0	1	0	0	0	0	
3	0	0	0	0	1	0	0	1	0	
4	0	0	0	0	1	0	0	0	0	

In [ ]:

```
unique_genere = genre_dummies.columns
unique_genere
```

Out[ ]:

```
Index(['Action', 'Adventure', 'Animation', 'Children's', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
       'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western'],
      dtype='object')
```

In [ ]:

```
final_movie_df = df_movies.join(genre_dummies, how='inner', lsuffix='', rsuffix='')
final_movie_df.drop('Genres', axis=1, inplace=True)
final_movie_df.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Doc
0	1	Toy Story	1995	0	0	1	1	1	0	
1	2	Jumanji	1995	0	1	0	1	0	0	
2	3	Grumpier Old Men	1995	0	0	0	0	1	0	
3	4	Waiting to Exhale	1995	0	0	0	0	1	0	
4	5	Father of the Bride Part II	1995	0	0	0	0	1	0	

In [ ]:

```
df_ratings.head()
df_ratings['Timestamp'] = pd.to_datetime(df_ratings['Timestamp'], unit='s')
df_ratings['Month'] = df_ratings['Timestamp'].dt.strftime('%b')
df_ratings['Year'] = df_ratings['Timestamp'].dt.year
df_ratings['Day'] = df_ratings['Timestamp'].dt.strftime('%a')
df_ratings['Hour'] = df_ratings['Timestamp'].dt.hour
```

In [ ]:

```
df_ratings
```

Out[ ]:

	UserID	MovieID	Rating	Timestamp	Month	Year	Day	Hour
0	1	1193	5	2000-12-31 22:12:40	Dec	2000	Sun	22
1	1	661	3	2000-12-31 22:35:09	Dec	2000	Sun	22
2	1	914	3	2000-12-31 22:32:48	Dec	2000	Sun	22
3	1	3408	4	2000-12-31 22:04:35	Dec	2000	Sun	22
4	1	2355	5	2001-01-06 23:38:11	Jan	2001	Sat	23
...	...	...	...	...	...	...	...	...
1000204	6040	1091	1	2000-04-26 02:35:41	Apr	2000	Wed	2
1000205	6040	1094	5	2000-04-25 23:21:27	Apr	2000	Tue	23
1000206	6040	562	5	2000-04-25 23:19:06	Apr	2000	Tue	23
1000207	6040	1096	4	2000-04-26 02:20:48	Apr	2000	Wed	2
1000208	6040	1097	4	2000-04-26 02:19:29	Apr	2000	Wed	2

1000209 rows × 8 columns

In [ ]:

```
df_users.head()
```

Out[ ]:

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

In [ ]:

```
df_movies_ratings = pd.merge(final_movie_df, df_ratings, how='inner', left_on='Movi
df_movies_ratings.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

In [ ]:

```
df_movies_ratings.drop('MovieID', axis=1, inplace=True)
df_movies_ratings.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

In [ ]:

```
df = pd.merge(df_movies_ratings, df_users, on='UserID',how='inner')
df.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

In [ ]:

```
with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/12_Zee_Re
  pickle.dump(df, f)
```

In [ ]:

```
df.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"
                },

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

In [ ]:
```
df.head()
```

Out[ ]:

	MovieID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

EDA

In [ ]:
```
df.shape
```

Out[ ]:
```
(1000209, 32)
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 32 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   Movie ID     1000209 non-null  int64
 1   Title        1000209 non-null  object
 2   ReleaseYear  1000209 non-null  object
 3   Action       1000209 non-null  int64
 4   Adventure    1000209 non-null  int64
 5   Animation    1000209 non-null  int64
 6   Children's   1000209 non-null  int64
 7   Comedy       1000209 non-null  int64
 8   Crime        1000209 non-null  int64
 9   Documentary  1000209 non-null  int64
 10  Drama        1000209 non-null  int64
 11  Fantasy      1000209 non-null  int64
 12  Film-Noir    1000209 non-null  int64
 13  Horror       1000209 non-null  int64
 14  Musical      1000209 non-null  int64
 15  Mystery      1000209 non-null  int64
 16  Romance      1000209 non-null  int64
 17  Sci-Fi       1000209 non-null  int64
 18  Thriller     1000209 non-null  int64
 19  War          1000209 non-null  int64
 20  Western      1000209 non-null  int64
 21  UserID       1000209 non-null  int64
 22  Rating       1000209 non-null  int64
 23  Timestamp    1000209 non-null  datetime64[ns]
 24  Month        1000209 non-null  object
 25  Year         1000209 non-null  int32
 26  Day          1000209 non-null  object
 27  Hour         1000209 non-null  int32
 28  Gender       1000209 non-null  object
 29  Age          1000209 non-null  object
 30  Occupation   1000209 non-null  object
 31  Zip-code     1000209 non-null  object
dtypes: datetime64[ns](1), int32(2), int64(21), object(8)
memory usage: 236.6+ MB
```

In [ ]:

```
df.describe()
```

Out[ ]:

	Movie ID	Action	Adventure	Animation	Children's	Comedy	Crime
count	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000
mean	1865.540	0.257	0.134	0.043	0.072	0.357	0.080
min	1.000	0.000	0.000	0.000	0.000	0.000	0.000
25%	1030.000	0.000	0.000	0.000	0.000	0.000	0.000
50%	1835.000	0.000	0.000	0.000	0.000	0.000	0.000
75%	2770.000	1.000	0.000	0.000	0.000	1.000	0.000
max	3952.000	1.000	1.000	1.000	1.000	1.000	1.000
std	1096.041	0.437	0.341	0.203	0.259	0.479	0.271

In [ ]:

```
df.describe(include=['object'])
```

Out[ ]:

	Title	ReleaseYear	Month	Day	Gender	Age	Occupation	Zip-code
count	1000209	1000209	1000209	1000209	1000209	1000209	1000209	1000209
unique	3664	81	12	7	2	7	21	3439
top	American Beauty	1999	Nov	Mon	M	25-34	college/grad student	94110
freq	3428	86833	295461	173931	753769	395556	131032	3802

In [ ]:

```
df.isna().sum()
```

Out[ ]:

	0
Movie ID	0
Title	0
ReleaseYear	0
Action	0
Adventure	0
Animation	0
Children's	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
UserID	0
Rating	0
Timestamp	0
Month	0
Year	0
Day	0
Hour	0
Gender	0
Age	0
Occupation	0
Zip-code	0

dtype: int64

In [ ]:

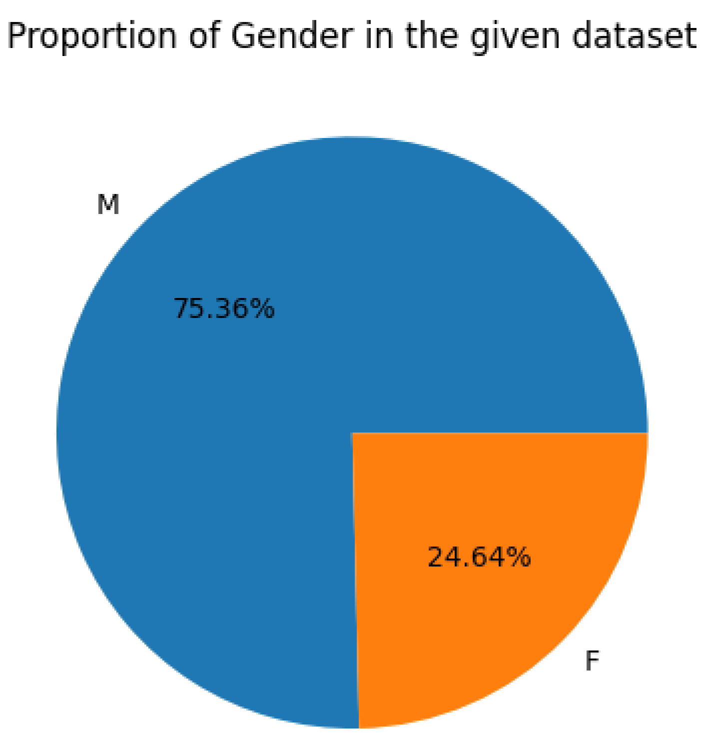
```
df.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

In [ ]:

```
plt.pie(df['Gender'].value_counts(), labels=df['Gender'].value_counts().index, autc
plt.title('Proportion of Gender in the given dataset')
plt.show()
display(df['Gender'].value_counts())
```



	count
Gender	
M	753769
F	246440

dtype: int64

Gender Distribution:

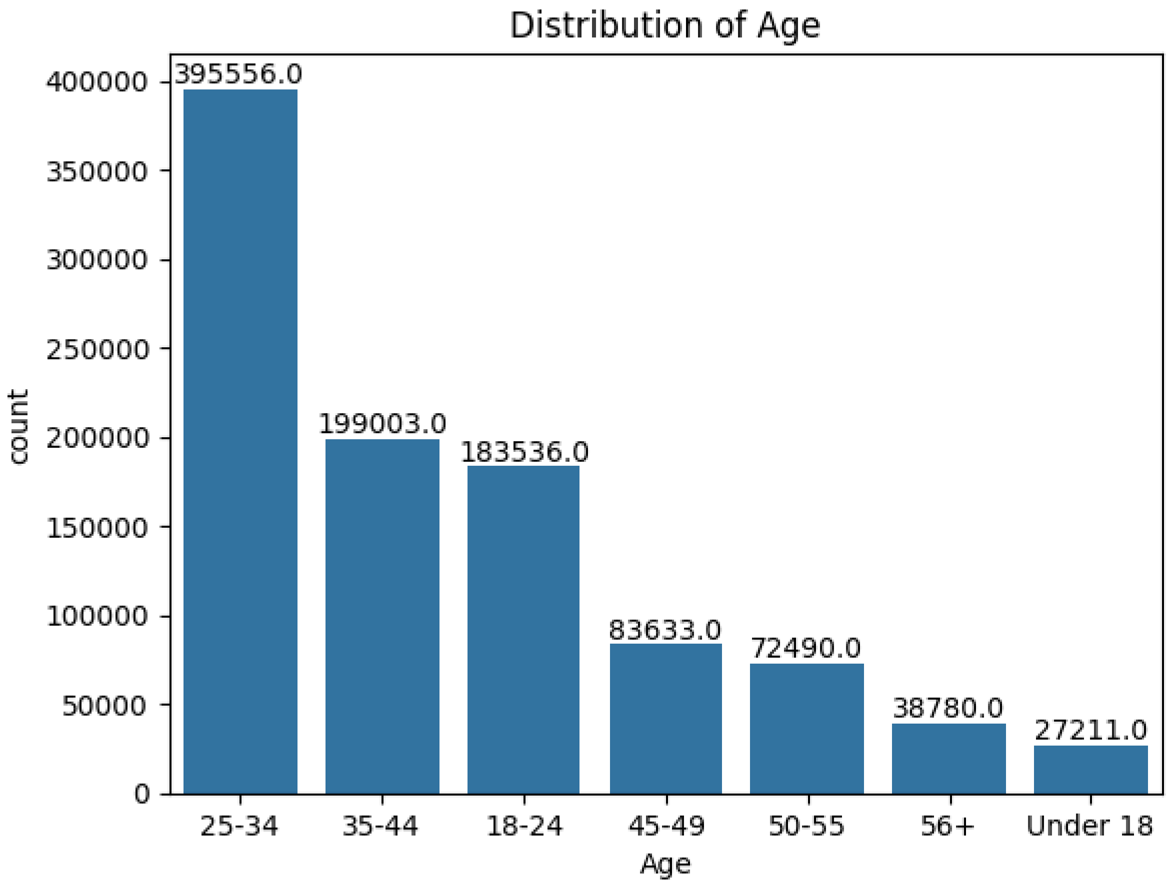
The dataset is highly imbalanced in terms of gender.

Males (M) make up 75.36% of the data.

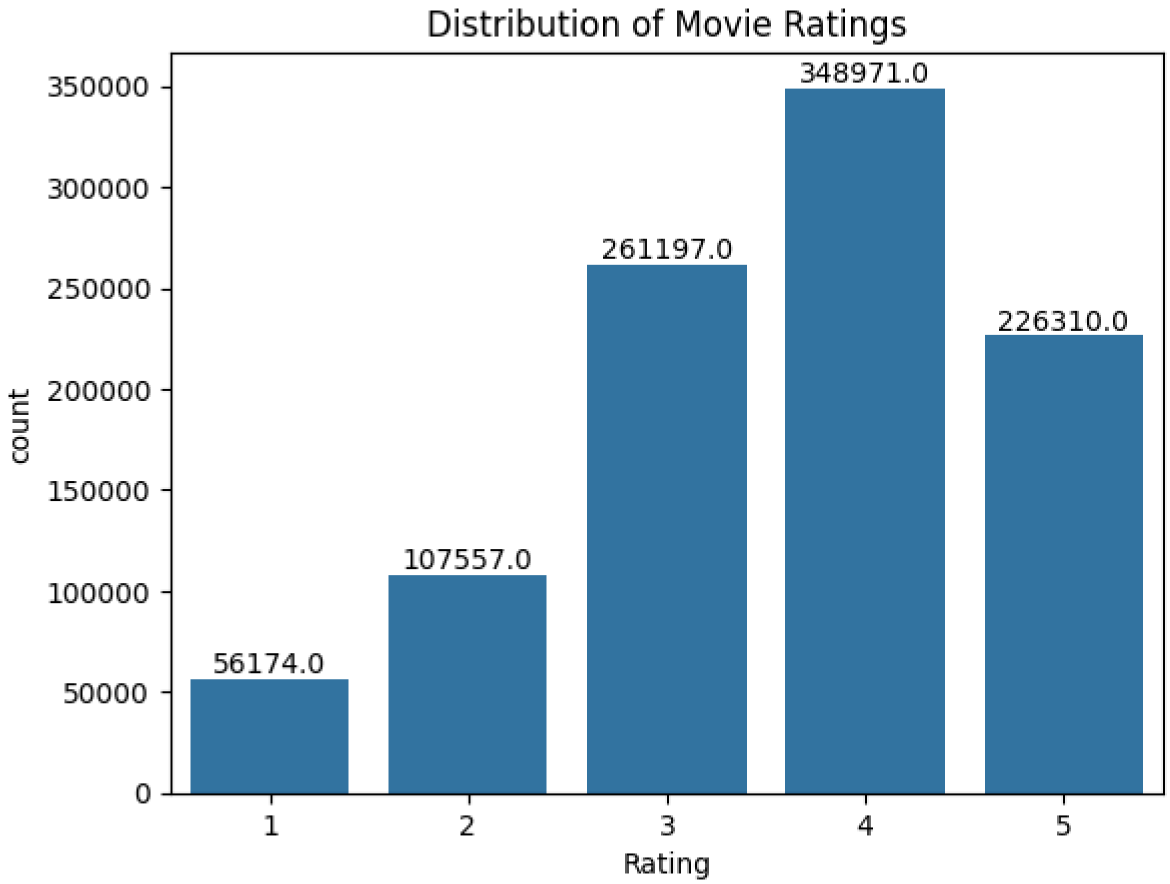
Females (F) make up only 24.64% of the data.

In [ ]:
```
def annotate(ax, rotation = False):
    for patch in ax.patches:  # Loop through each bar
        if rotation:  # For horizontal bars
            x = patch.get_width()  # Get the width (value of the bar)
            y = patch.get_y() + patch.get_height() / 2  # Center the annotation ver
            ax.annotate(f"{x}", (x + 0.5, y), ha='left', va='center')  # Adjust pos
        else:  # For vertical bars
            x = patch.get_x() + patch.get_width() / 2  # Center the annotation hori
            y = patch.get_height()  # Get the height (value of the bar)
            ax.annotate(f"{y}", (x, y + 0.5), ha='center', va='bottom')  # Adjust p
```

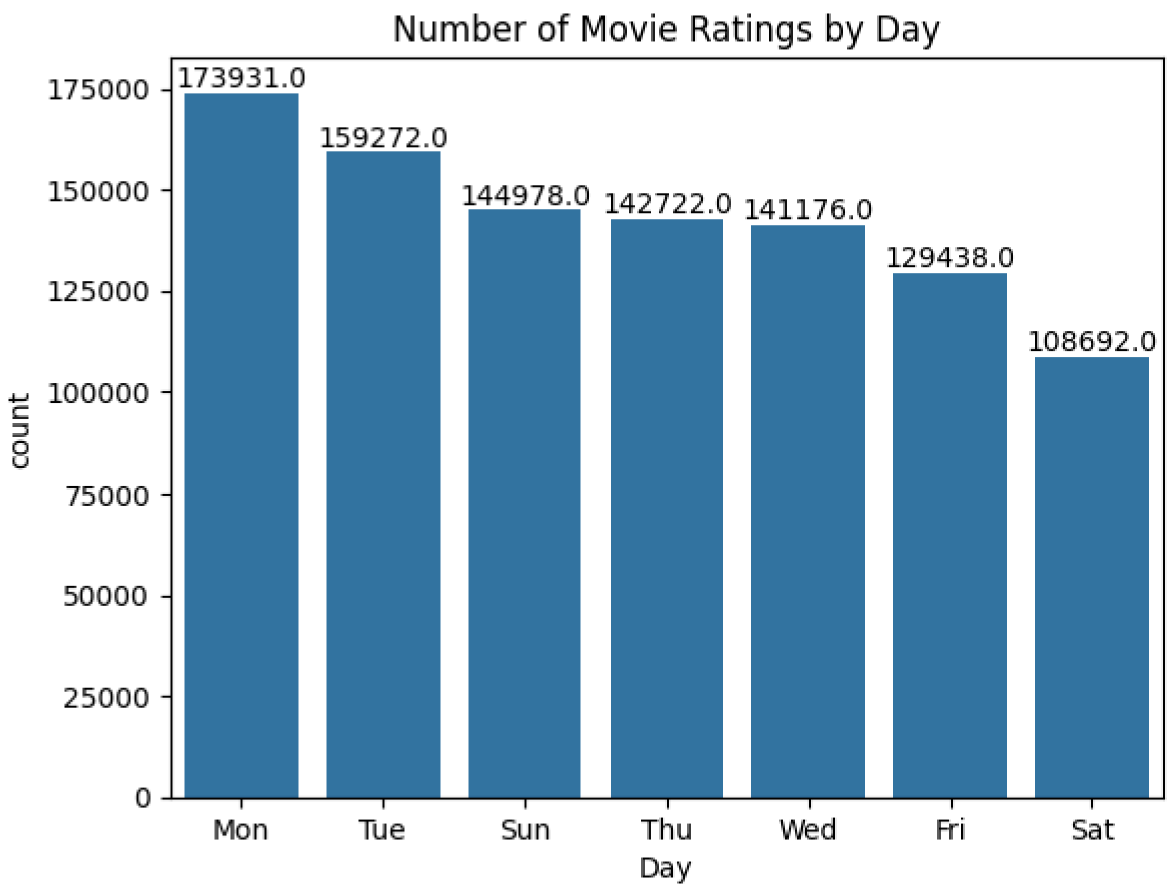
In [ ]:
```
plt.title('Distribution of Age')
ax = sns.barplot(df.Age.value_counts().reset_index(), x='Age', y='count')
annotate(ax)
plt.show()
```



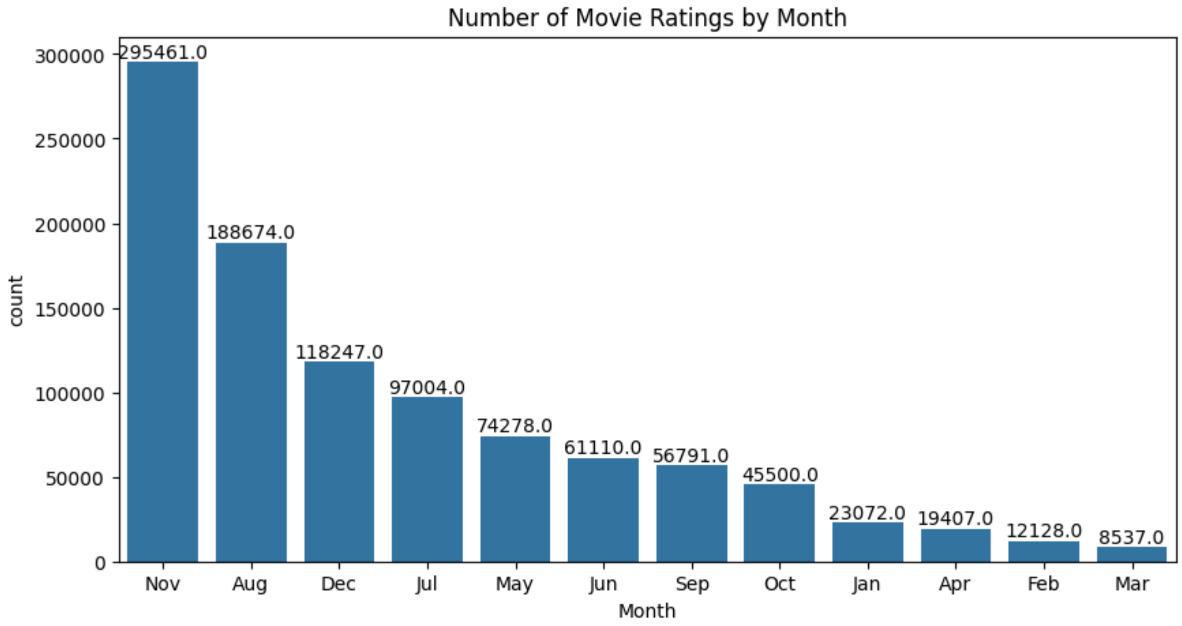
In [ ]:
```
plt.title('Distribution of Movie Ratings')
ax = sns.barplot(df.Rating.value_counts().reset_index(), x='Rating', y='count')
annotate(ax)
plt.show()
```



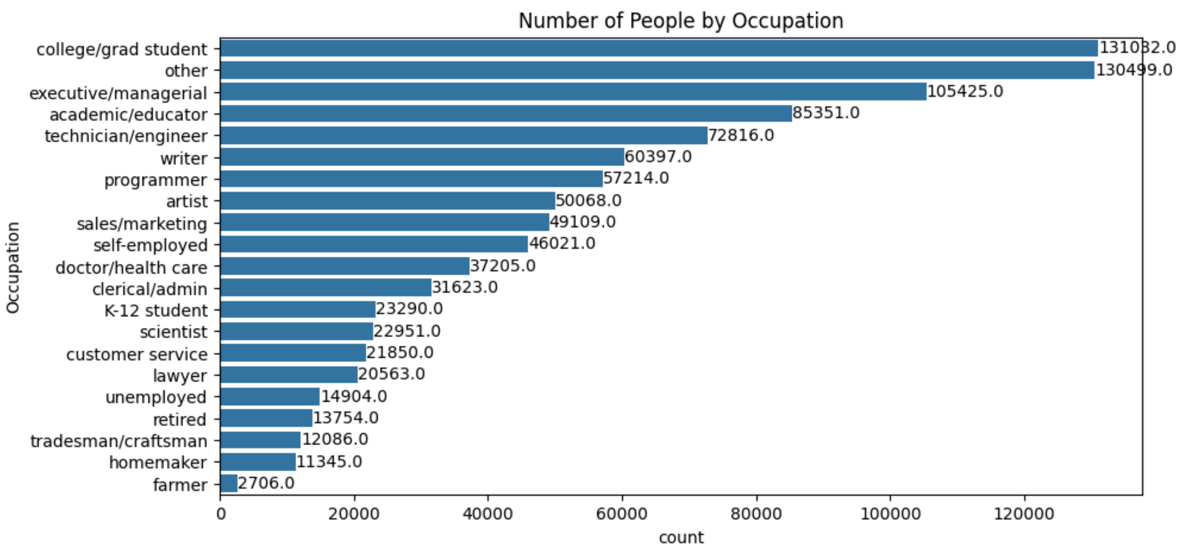
In [ ]:
```
plt.title('Number of Movie Ratings by Day')
ax = sns.barplot(df.Day.value_counts().reset_index(), x='Day', y='count')
annotate(ax)
plt.show()
```



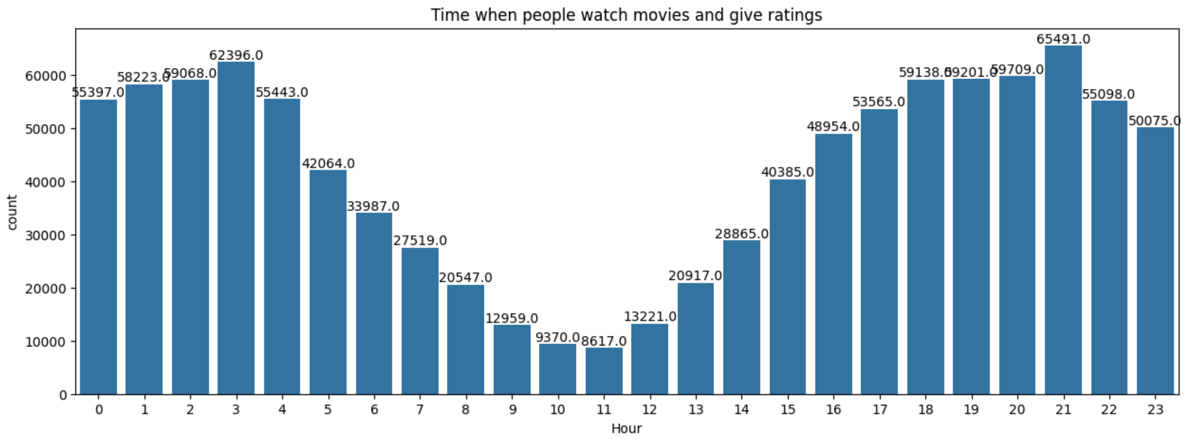
In [ ]:
```
plt.figure(figsize=(10,5))
ax = sns.barplot(df.Month.value_counts().reset_index(), x='Month', y='count')
plt.title('Number of Movie Ratings by Month')
annotate(ax)
plt.show()
```



In [ ]:
```
plt.figure(figsize=(10,5))
ax = sns.barplot(df.Occupation.value_counts().reset_index(), y='Occupation', x='cou
plt.title('Number of People by Occupation')
annotate(ax, rotation=True)
plt.show()
```

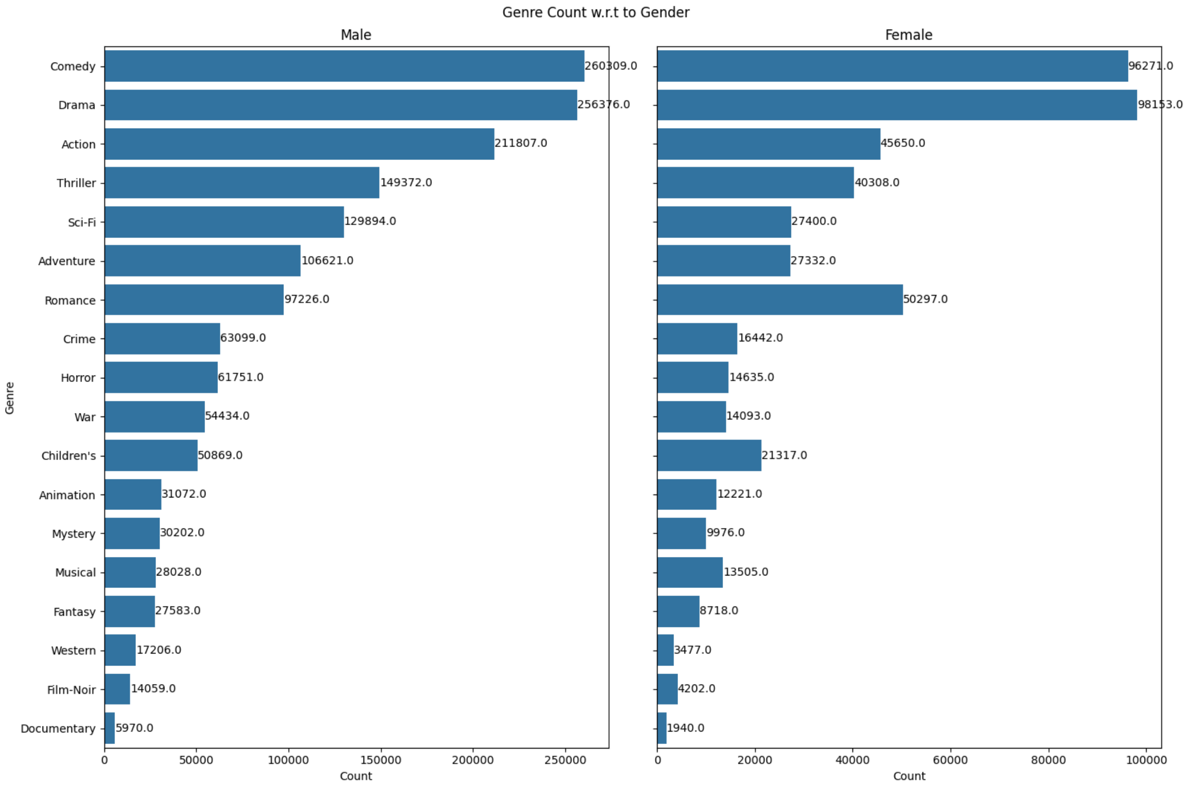


In [ ]:
```
plt.figure(figsize=(15,5))
plt.title('Time when people watch movies and give ratings')
ax = sns.barplot(df.Hour.value_counts().reset_index(), x='Hour', y='count')
annotate(ax)
plt.show()
```



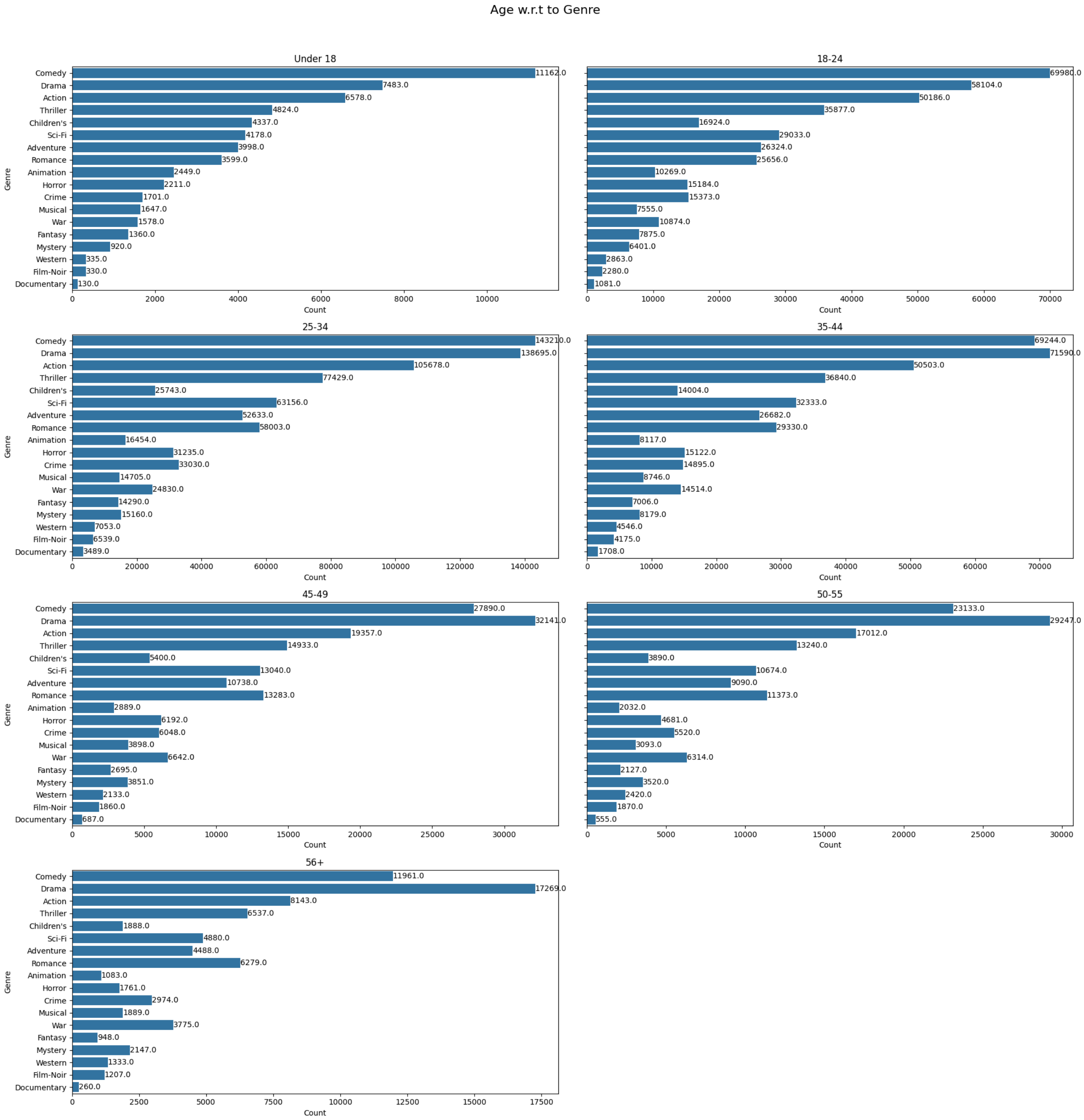
In [ ]:

```
def get_gender_count_by_genre(_df, genre):
    _df = _df[_df[genre] == 1].groupby('Gender')['UserID'].count()
    _df.name = genre
    return _df

genre_df_list = []
for i in range(len(unique_genere)):
    _genre_count_df = get_gender_count_by_genre(df, unique_genere[i])
    genre_df_list.append(_genre_count_df)

gender_genre_df = pd.concat(genre_df_list, axis=1)
gender_genre_df.reset_index(inplace=True)
melted_gender_genre_df = gender_genre_df.melt(id_vars=['Gender'], var_name='Genre',


fig, axes = plt.subplots(1, 2, figsize=(15, 10), sharey=True)
plt.suptitle('Genre Count w.r.t to Gender')
for idx, gender in enumerate(['M', 'F']):

    temp_df = melted_gender_genre_df[melted_gender_genre_df['Gender'] == gender]
    temp_df = temp_df.drop(columns=['Gender'])
    temp_df.sort_values(by='Count', ascending=False, inplace=True)

    ax = axes[idx]
    _gender = 'Male' if gender == 'M' else 'Female'
    sns.barplot(data=temp_df, x='Count', y='Genre', ax=ax)
    ax.set_title(f'{_gender}')
    ax.set_xlabel('Count')
    ax.set_ylabel('Genre')

    annotate(ax, rotation=True)

plt.tight_layout()
plt.show()
```



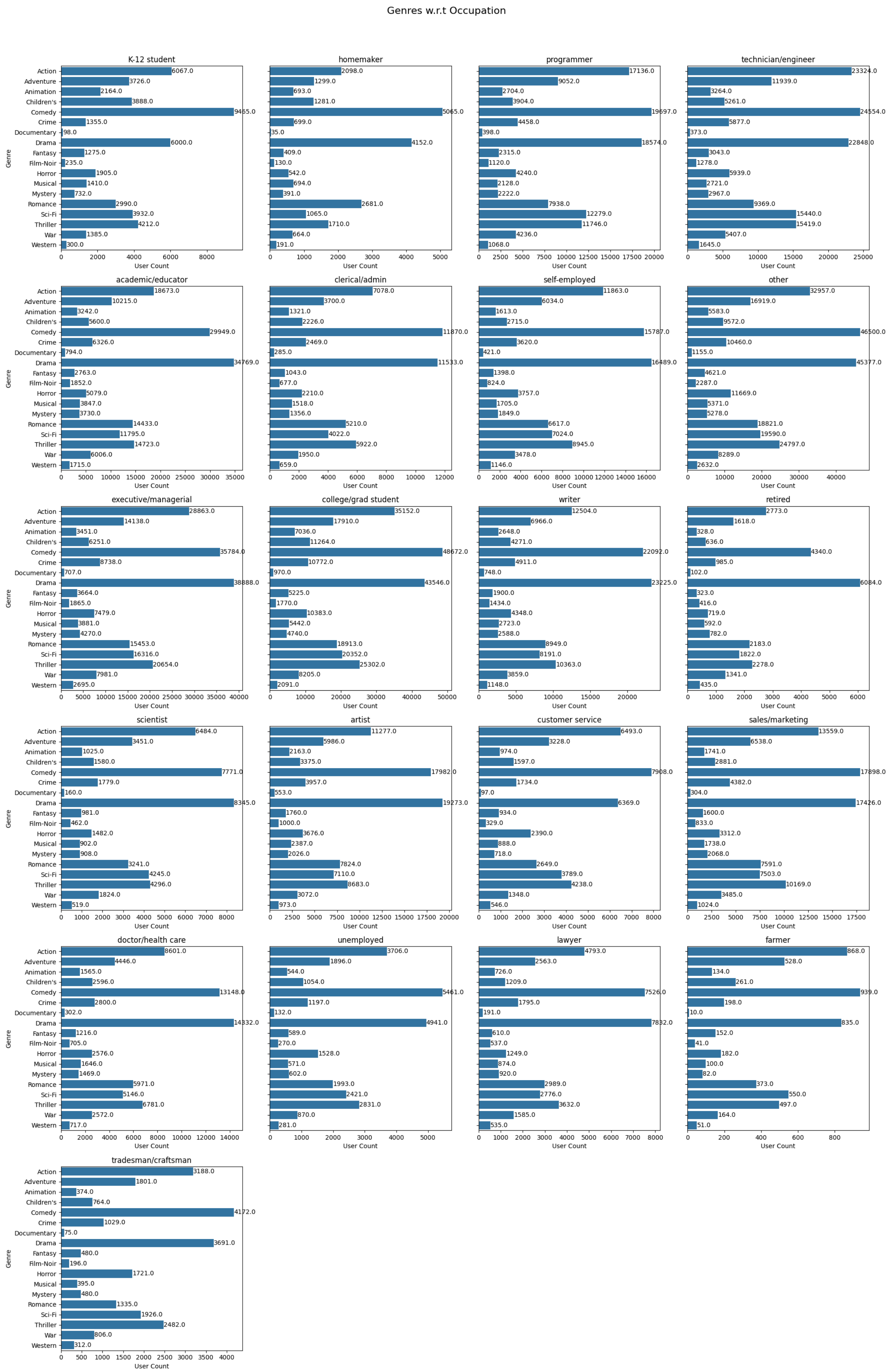
In [ ]:
```
def get_genre_count_by_feature(_df, genre, feature, feature_order):
    _df = _df[_df[genre] == 1].groupby(feature)['UserID'].count()
    _df.name = genre
    _df = _df.reset_index()
    _df[feature] = pd.Categorical(_df[feature], categories=feature_order, ordered=T
    _df = _df.sort_values(feature)
    _df = _df.set_index('Age')
    return _df

age_genre_list = []
custom_age_order = ['Under 18', '18-24', '25-34', '35-44', '45-49', '50-55', '56+']
for i in range(len(unique_genere)):
    age_genre_list.append(get_genre_count_by_feature(df, unique_genere[i], 'Age', c

df_final = pd.concat(age_genre_list, axis=1)
df_final = df_final.T
df_final.index.name = 'Genre'

# Plotting logic
n_cols = 2
n_rows = (len(df_final.columns) + 1) // 2  # Ceiling division to ensure enough rows
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 20), sharey=True)
axes = axes.flatten()  # Flatten to easily index axes in a loop

for i, age_group in enumerate(df_final.columns):
    temp_df = df_final[[age_group]].sort_values(by=age_group, ascending=False)
    temp_df = temp_df.reset_index()

    # Plot on the corresponding subplot
    ax = axes[i]
    sns.barplot(data=temp_df, x=age_group, y='Genre', ax=ax)
    ax.set_title(f'{age_group}')
    ax.set_xlabel('Count')
    ax.set_ylabel('Genre')


    annotate(ax, rotation=True)
```

```
# Remove empty subplots if any
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

fig.suptitle('Age w.r.t to Genre', fontsize=16, y=1.02)  # Adjust y to avoid overla
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```



In [ ]:
```
def get_genre_count_by_occupation(_df, occupation, genres, genre_order):
    # Filter for the specific occupation
    _df = _df[_df['Occupation'] == occupation]
    # Count users for each genre where genre column == 1
    counts = []
    for genre in genres:
        count = _df[_df[genre] == 1]['UserID'].count()
        counts.append(count)
    # Create DataFrame with genre counts
    result_df = pd.DataFrame({
        'Genre': genres,
        'Count': counts
    })
    # Apply categorical order to genres (optional, for consistency)
    result_df['Genre'] = pd.Categorical(result_df['Genre'], categories=genre_order,
    # Sort by count for plotting
    result_df = result_df.sort_values('Count', ascending=False)
```

```python
    return result_df

# List of occupations
custom_occupation_order = [
    'K-12 student', 'homemaker', 'programmer', 'technician/engineer',
    'academic/educator', 'clerical/admin', 'self-employed', 'other',
    'executive/managerial', 'college/grad student', 'writer',
    'retired', 'scientist', 'artist', 'customer service',
    'sales/marketing', 'doctor/health care', 'unemployed', 'lawyer',
    'farmer', 'tradesman/craftsman'
]

# Plotting logic
n_cols = 4  # Exactly 4 columns per row
n_rows = (len(custom_occupation_order) + n_cols - 1) // n_cols  # Ceiling division
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 5 * n_rows), sharey=True)
axes = axes.flatten()  # Flatten for easy indexing

for i, occupation in enumerate(custom_occupation_order):
    # Get genre counts for the current occupation
    temp_df = get_genre_count_by_occupation(df, occupation, unique_genere, unique_g

    # Plot on the corresponding subplot
    ax = axes[i]
    sns.barplot(data=temp_df, x='Count', y='Genre', ax=ax)
    ax.set_title(f'{occupation}')
    ax.set_xlabel('User Count')
    ax.set_ylabel('Genre')

    annotate(ax, rotation=True)

# Remove empty subplots (last 3 in 6x4 grid)
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

fig.suptitle('Genres w.r.t Occupation', fontsize=16, y=1.02)  # Figure-level title
plt.tight_layout()  # Adjust layout
plt.show()
```


Genres w.r.t Occupation
K-12 student
homemaker
programmer
technician/engineer
academic/educator
clerical/admin
self-employed
other
executive/managerial
college/grad student
writer
retired
scientist
artist
customer service
sales/marketing
doctor/health care
unemployed
lawyer
farmer
tradesman/craftsman
Action
Adventure
Animation
Children's
Comedy
Crime
Documentary
Drama
Fantasy
Film-Noir
Horror
Musical
Mystery
Romance
Sci-Fi
Thriller
War
Western
Genre
User Count

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 32 columns):
 #   Column       Non-Null Count    Dtype         
---  ------       --------------    -----         
 0   Movie ID     1000209 non-null  int64         
 1   Title        1000209 non-null  object        
 2   ReleaseYear  1000209 non-null  object        
 3   Action       1000209 non-null  int64         
 4   Adventure    1000209 non-null  int64         
 5   Animation    1000209 non-null  int64         
 6   Children's   1000209 non-null  int64         
 7   Comedy       1000209 non-null  int64         
 8   Crime        1000209 non-null  int64         
 9   Documentary  1000209 non-null  int64         
 10  Drama        1000209 non-null  int64         
 11  Fantasy      1000209 non-null  int64         
 12  Film-Noir    1000209 non-null  int64         
 13  Horror       1000209 non-null  int64         
 14  Musical      1000209 non-null  int64         
 15  Mystery      1000209 non-null  int64         
 16  Romance      1000209 non-null  int64         
 17  Sci-Fi       1000209 non-null  int64         
 18  Thriller     1000209 non-null  int64         
 19  War          1000209 non-null  int64         
 20  Western      1000209 non-null  int64         
 21  UserID       1000209 non-null  int64         
 22  Rating       1000209 non-null  int64         
 23  Timestamp    1000209 non-null  datetime64[ns]
 24  Month        1000209 non-null  object        
 25  Year         1000209 non-null  int32         
 26  Day          1000209 non-null  object        
 27  Hour         1000209 non-null  int32         
 28  Gender       1000209 non-null  object        
 29  Age          1000209 non-null  object        
 30  Occupation   1000209 non-null  object        
 31  Zip-code     1000209 non-null  object        
dtypes: datetime64[ns](1), int32(2), int64(21), object(8)
memory usage: 236.6+ MB
```

In [ ]:

```
for i in df.columns:
  print(i, df[i].nunique())
```

```
Movie ID 3706
Title 3664
ReleaseYear 81
Action 2
Adventure 2
Animation 2
Children's 2
Comedy 2
Crime 2
Documentary 2
Drama 2
Fantasy 2
Film-Noir 2
Horror 2
Musical 2
Mystery 2
Romance 2
Sci-Fi 2
Thriller 2
War 2
Western 2
UserID 6040
Rating 5
Timestamp 458455
Month 12
Year 4
Day 7
Hour 24
Gender 2
Age 7
Occupation 21
Zip-code 3439
```

In [ ]:

```
df.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

In [ ]:

```
top_rated_2021 = df[(df['Year'] == 2001) &  (df['Rating'] == 5) &  (df['Month'] ==
top_rated_2021
```

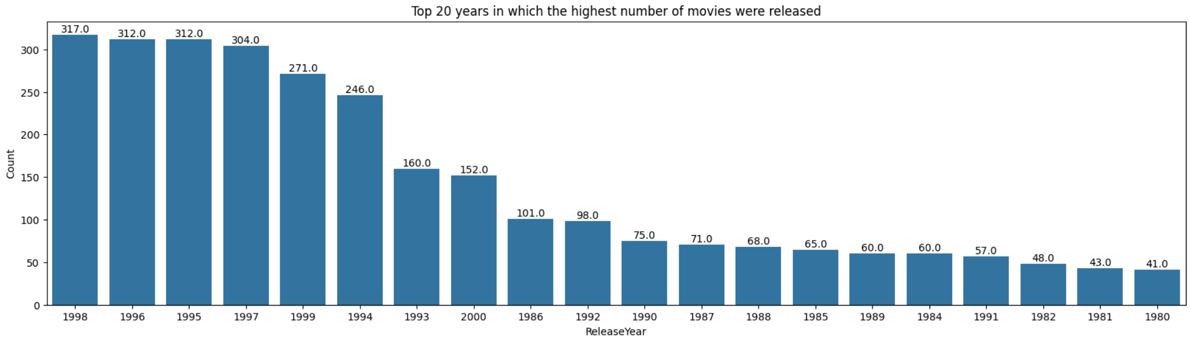
Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime
0	1	Toy Story	1995	0	0	1	1	1	0
6	1	Toy Story	1995	0	0	1	1	1	0
171	1	Toy Story	1995	0	0	1	1	1	0
210	1	Toy Story	1995	0	0	1	1	1	0
430	1	Toy Story	1995	0	0	1	1	1	0
...	...	...	...	...	...	...	...	...	...
999261	3948	Meet the Parents	2000	0	0	0	0	1	0
999455	3949	Requiem for a Dream	2000	0	0	0	0	0	0
999535	3949	Requiem for a Dream	2000	0	0	0	0	0	0
999699	3949	Requiem for a Dream	2000	0	0	0	0	0	0
999781	3951	Two Family House	2000	0	0	0	0	0	0

3464 rows × 32 columns

In [ ]:

```
year_unique_titles_df = df[['Title','ReleaseYear']].drop_duplicates()
year_unique_titles_df = year_unique_titles_df.sort_values(by='ReleaseYear')
year_move_count_df = year_unique_titles_df.groupby(by='ReleaseYear')['Title'].count
year_move_count_df.name = 'Count'
year_move_count_df = year_move_count_df.sort_values(ascending=False)
year_move_count_df = year_move_count_df.head(40).reset_index()
plt.figure(figsize=(20,5))
plt.title('Top 20 years in which the highest number of movies were released')
ax = sns.barplot(year_move_count_df.head(20), x=year_move_count_df.columns[0], y=ye
annotate(ax)
plt.show()
```



Modelling Recommender system

User-Movie Matrix

In [ ]:
```
user_movie_matrix  = pd.pivot_table(df, index='UserID', columns='Title', values='Ra
user_movie_matrix.head(10)
```

Out[ ]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The
UserID										
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	4.000	NaN	NaN	NaN	NaN	3.000	4.000

In [ ]:
```
user_movie_matrix.fillna(0, inplace=True)
```

In [ ]:
```
user_movie_matrix.head()
```

Out[ ]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The
UserID										
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

In [ ]:

```
user_movie_matrix.shape
```

Out[ ]:

```
(6040, 3664)
```

Recommender System based on Pearson Correlation (Item-based)

In [ ]:

```
user_movie_matrix.columns[:10]
```

Out[ ]:

```
Index(['$1,000,000 Duck', ''Night Mother', ''Til There Was You', ''burbs, The',
       '...And Justice for All', '1-900', '10 Things I Hate About You',
       '101 Dalmatians', '12 Angry Men', '13th Warrior, The'],
      dtype='object', name='Title')
```

In [ ]:

```
user_movie_matrix.loc[:,"'Til There Was You"]
```

Out[ ]:

	'Til There Was You
UserID	
1	0.000
2	0.000
3	0.000
4	0.000
5	0.000
...	...
6036	0.000
6037	0.000
6038	0.000
6039	0.000
6040	0.000

6040 rows × 1 columns

dtype: float64

In [ ]:
```
similar_movies = user_movie_matrix.corrwith(user_movie_matrix.loc[:,"'Til There Was
similar_movies = similar_movies.sort_values(ascending=False)
similar_movies[1:].head(5)
```

Out[ ]:

	0
Title	
If Lucy Fell	0.267
Picture Perfect	0.256
To Gillian on Her 37th Birthday	0.241
Mad Love	0.231
Practical Magic	0.230

dtype: float64

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

In [ ]:
```
sim_df.iloc[1:, :]
```

Out[ ]:

	Correlation
Title	
If Lucy Fell	0.267
Picture Perfect	0.256
To Gillian on Her 37th Birthday	0.241
Mad Love	0.231
Practical Magic	0.230
...	...
Kelly's Heroes	-0.014
Boat, The (Das Boot)	-0.014
Good, The Bad and The Ugly, The	-0.014
High Plains Drifter	-0.016
Magnum Force	-0.016

3663 rows × 1 columns

Item-Item Similarity

In [ ]:

```
item_sim = cosine_similarity(user_movie_matrix.T)
item_sim
```

Out[ ]:

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

In [ ]:

```
item_sim_mat = pd.DataFrame(item_sim, index=user_movie_matrix.columns, columns=user
item_sim_mat.head()
```

Out[ ]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13 Warri T
Title										
$1,000,000 Duck	1.000	0.072	0.037	0.079	0.061	0.000	0.059	0.190	0.095	0.0
'Night Mother	0.072	1.000	0.115	0.116	0.160	0.000	0.077	0.137	0.111	0.0
'Til There Was You	0.037	0.115	1.000	0.099	0.066	0.080	0.128	0.129	0.079	0.0
'burbs, The	0.079	0.116	0.099	1.000	0.144	0.000	0.192	0.250	0.171	0.1
...And Justice for All	0.061	0.160	0.066	0.144	1.000	0.000	0.075	0.179	0.205	0.1

User-User similarity matrix

In [ ]:
```
user_sim = cosine_similarity(user_movie_matrix)
user_sim
```

Out[ ]:
```
array([[1.        , 0.09638153, 0.12060981, ..., 0.        , 0.17460369,
        0.13359025],
       [0.09638153, 1.        , 0.1514786 , ..., 0.06611767, 0.0664575 ,
        0.21827563],
       [0.12060981, 0.1514786 , 1.        , ..., 0.12023352, 0.09467506,
        0.13314404],
       ...,
       [0.        , 0.06611767, 0.12023352, ..., 1.        , 0.16171426,
        0.09930008],
       [0.17460369, 0.0664575 , 0.09467506, ..., 0.16171426, 1.        ,
        0.22833237],
       [0.13359025, 0.21827563, 0.13314404, ..., 0.09930008, 0.22833237,
        1.        ]])
```

In [ ]:
```
user_sim_mat = pd.DataFrame(user_sim, index=user_movie_matrix.index, columns=user_m
user_sim_mat.head()
```

Out[ ]:

UserID	1	2	3	4	5	6	7	8	9	10	11	12	13	
UserID														
1	1.000	0.096	0.121	0.132	0.090	0.179	0.060	0.138	0.226	0.255	0.130	0.110	0.124	0.0
2	0.096	1.000	0.151	0.171	0.114	0.101	0.306	0.211	0.190	0.228	0.197	0.096	0.317	0.0
3	0.121	0.151	1.000	0.151	0.063	0.075	0.138	0.078	0.126	0.214	0.174	0.084	0.277	0.0
4	0.132	0.171	0.151	1.000	0.045	0.014	0.130	0.101	0.094	0.121	0.068	0.066	0.196	0.0
5	0.090	0.114	0.063	0.045	1.000	0.047	0.126	0.221	0.261	0.118	0.221	0.045	0.118	0.1

In [ ]:
```
user_movie_matrix.T.values
```

Out[ ]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Item based approach using Neighbors algorithm and Cosine Similarity

In [ ]:
```
csr_mat = sparse.csr_matrix(user_movie_matrix.T.values)
csr_mat
```

Out[ ]:
```
<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 997085 stored elements and shape (3664, 6040)>
```

In [ ]:
```
knn = NearestNeighbors(n_neighbors=5, metric='cosine', n_jobs=-1)
knn.fit(csr_mat)
```

Out[ ]:
```
NearestNeighbors
NearestNeighbors(metric='cosine', n_jobs=-1)
```

In [ ]:
```
movie_name = "'Til There Was You"
movie_index = user_movie_matrix.columns.get_loc(movie_name)
```

In [ ]:
```
distances, indices = knn.kneighbors(user_movie_matrix[movie_name].values.reshape(1,
```

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

```
Recommendations for the movie: 'Til There Was You

1: If Lucy Fell, with distance of 0.726
2: Picture Perfect, with distance of 0.735
3: To Gillian on Her 37th Birthday, with distance of 0.751
4: Practical Magic, with distance of 0.759
5: Mad Love, with distance of 0.763
6: Something to Talk About, with distance of 0.764
7: Circle of Friends, with distance of 0.766
8: Beautician and the Beast, The, with distance of 0.77
9: Evening Star, The, with distance of 0.772
10: How to Make an American Quilt, with distance of 0.774
```

Matrix Factorization

In [ ]:
```
df.head()
```

Out[ ]:

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Docume
0	1	Toy Story	1995	0	0	1	1	1	0	
1	1	Toy Story	1995	0	0	1	1	1	0	
2	1	Toy Story	1995	0	0	1	1	1	0	
3	1	Toy Story	1995	0	0	1	1	1	0	
4	1	Toy Story	1995	0	0	1	1	1	0	

In [ ]:
```
df_movies = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessC
```

In [ ]:
```
df_ratings.columns
```

Out[ ]:
```
Index(['UserID', 'MovieID', 'Rating', 'Timestamp', 'Month', 'Year', 'Day',
       'Hour'],
      dtype='object')
```

In [ ]:
```
rm = df_ratings.pivot(index = 'UserID', columns ='MovieID', values = 'Rating').fill
rm.head()
```

Out[ ]:

MovieID	1	2	3	4	5	6	7	8	9	10	11	12	13	
UserID														
1	5.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	C
2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	C
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	C
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	C
5	0.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	C

In [ ]:
```
rm_raw = df_ratings[['UserID', 'MovieID', 'Rating']].copy()
rm_raw.columns = ['UserId', 'ItemId', 'Rating']
rm_raw.head(2)
```

Out[ ]:

	UserId	ItemId	Rating
0	1	1193	5
1	1	661	3

In [ ]:
```
model = CMF(method="als", k=2, lambda_=0.1, user_bias=False, item_bias=False, verbo
model.fit(rm_raw)
```

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

In [ ]:
```
model.A_.shape, model.B_.shape
```

Out[ ]:
```
((6040, 2), (3706, 2))
```

In [ ]:
```
rm_raw.Rating.mean(), model.glob_mean_
```

Out[ ]:
```
(np.float64(3.581564453029317), 3.581564426422119)
```

In [ ]:
```
rm.shape
```

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

In [ ]:
```
rm__ = np.dot(model.A_, model.B_.T) + model.glob_mean_
# mse(rm.values[rm > 0], rm__[rm > 0])**0.5
print('RMSE :' ,round(rmse(rm.values[rm > 0], rm__[rm > 0]),2))
print('MSE :',round( mse(rm.values[rm > 0], rm__[rm > 0]),2))
print('MAPE :', round(mape(rm.values[rm > 0], rm__[rm > 0]),2))
```

```
RMSE : 1.3
MSE : 1.7
MAPE : 0.38
```

Overlap

In [ ]:
```
top_items = model.topN(user=1, n=10)
df_movies.loc[df_movies['Movie ID'].isin(top_items)]
```

Out[ ]:

	Movie ID	Title	Genres
638	643	Peanuts - Die Bank zahlt alles (1996)	Comedy
883	895	Venice/Venice (1992)	Drama
1397	1421	Grateful Dead (1995)	Documentary
2754	2823	Spiders, The (Die Spinnen, 1. Teil: Der Golden...	Action|Drama
2842	2911	Grandfather, The (El Abuelo) (1998)	Drama
3264	3333	Killing of Sister George, The (1968)	Drama
3311	3380	Railroaded! (1947)	Film-Noir
3462	3531	All the Vermeers in New York (1990)	Comedy|Drama|Romance
3748	3818	Pot O' Gold (1941)	Comedy|Musical
3822	3892	Anatomy (Anatomie) (2000)	Horror

In [ ]:

```
top_items = model.topN(user=10, n=10)
df_movies.loc[df_movies['Movie ID'].isin(top_items)]
```

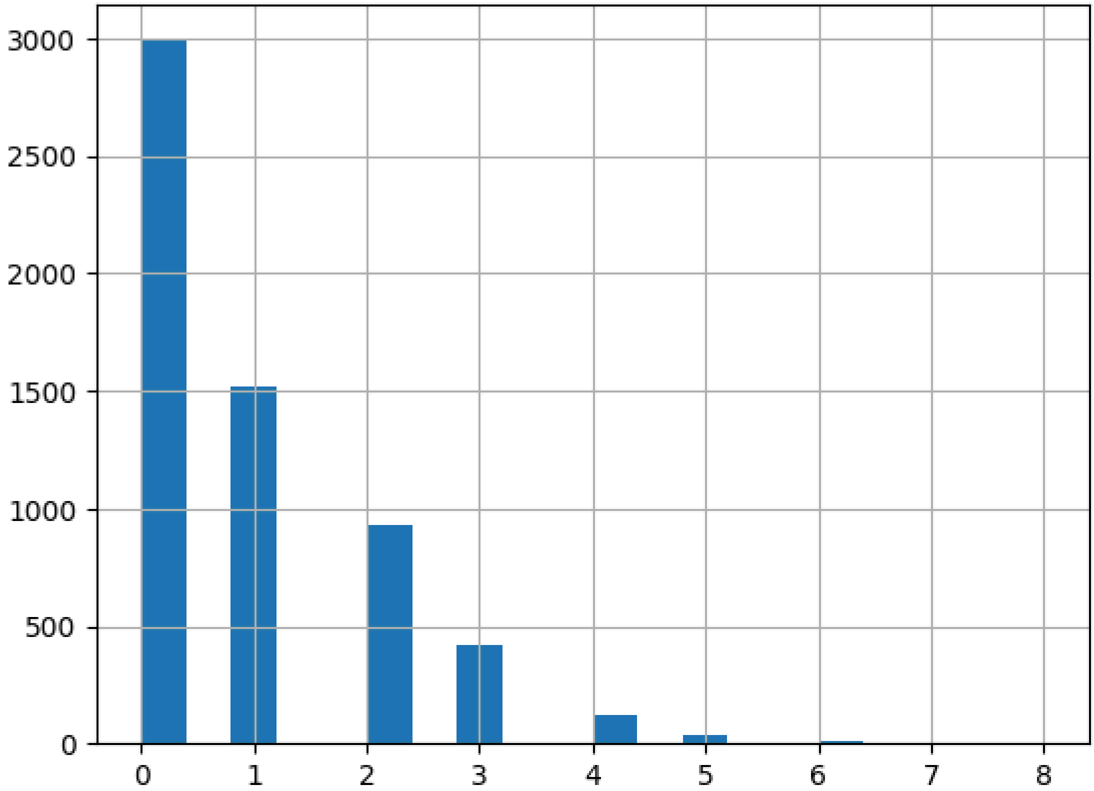
Out[ ]:

	Movie ID	Title	Genres
638	643	Peanuts - Die Bank zahlt alles (1996)	Comedy
883	895	Venice/Venice (1992)	Drama
1397	1421	Grateful Dead (1995)	Documentary
2469	2538	Dancemaker (1998)	Documentary
2754	2823	Spiders, The (Die Spinnen, 1. Teil: Der Golden...	Action|Drama
2842	2911	Grandfather, The (El Abuelo) (1998)	Drama
3311	3380	Railroaded! (1947)	Film-Noir
3462	3531	All the Vermeers in New York (1990)	Comedy|Drama|Romance
3748	3818	Pot O' Gold (1941)	Comedy|Musical
3822	3892	Anatomy (Anatomie) (2000)	Horror

In [ ]:

```
overlap= []
num_rec = []
n = 20
for user in df_ratings['UserID'].unique():
    top_items = model.topN(user=user, n=n)
    user_movies = df_ratings.loc[(df_ratings['UserID']==user)]['MovieID']
    valid_rec = set(top_items).intersection(set(user_movies)) # I can only measure

    _ = len(set(df_ratings.loc[df_ratings['UserID']==user].sort_values(by='Rating',
    overlap.append(_)
    num_rec.append(len(valid_rec))

print('avg_perc_overlap:', np.array(overlap).mean() / np.array(num_rec).mean())
pd.Series(overlap).hist(bins=20)
plt.show()
```

```
avg_perc_overlap: 0.34503222512921955
```

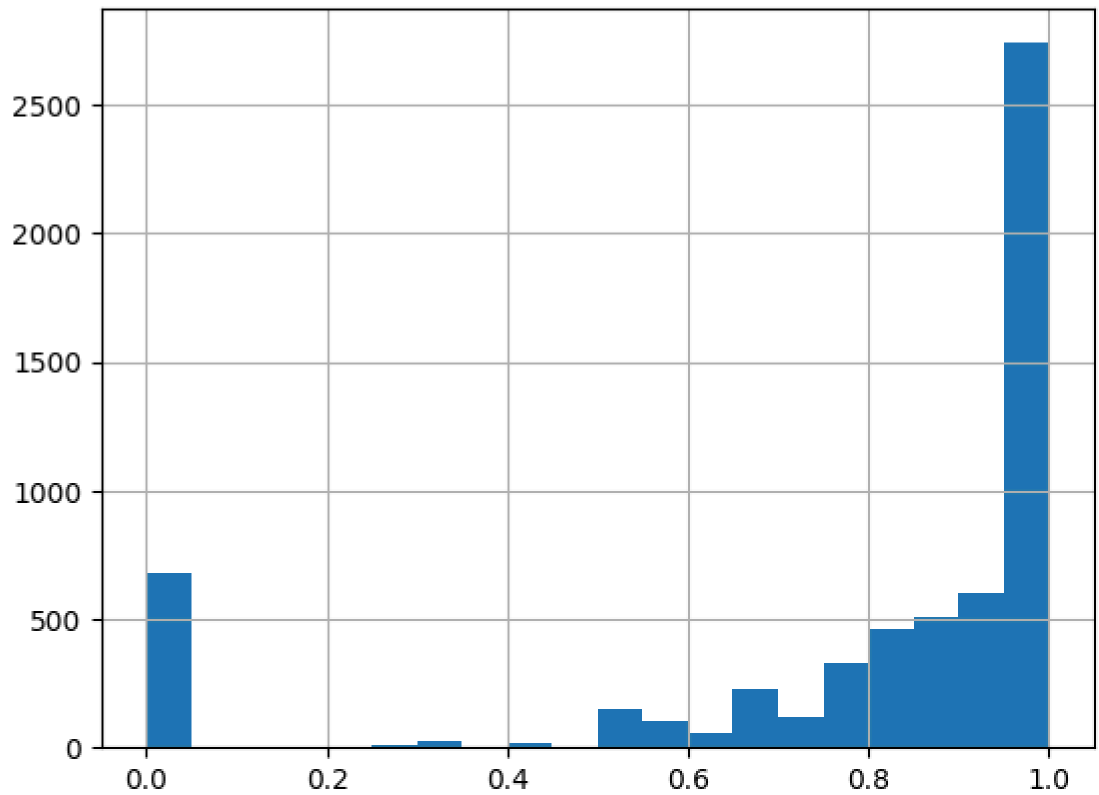


K-precision

In [ ]:

```
overlap=[]
for user in df_ratings['UserID'].unique():
    recommendations = model.topN(user=user, n=100)
    user_movies = df_ratings.loc[(df_ratings['UserID']==user)]['MovieID']
    valid_rec = set(recommendations).intersection(set(user_movies)) # I can only me
    relevant_items = df_ratings.loc[(df_ratings['UserID']==user) & (df_ratings['Rat
    try:
        _ = len(set(recommendations).intersection(set(relevant_items))) / len(valid
    except:
        _ = 0
    overlap.append(_)

overlap = np.array(overlap)
print('avg:', overlap.mean())
pd.Series(overlap).hist(bins=20)
plt.show()
```

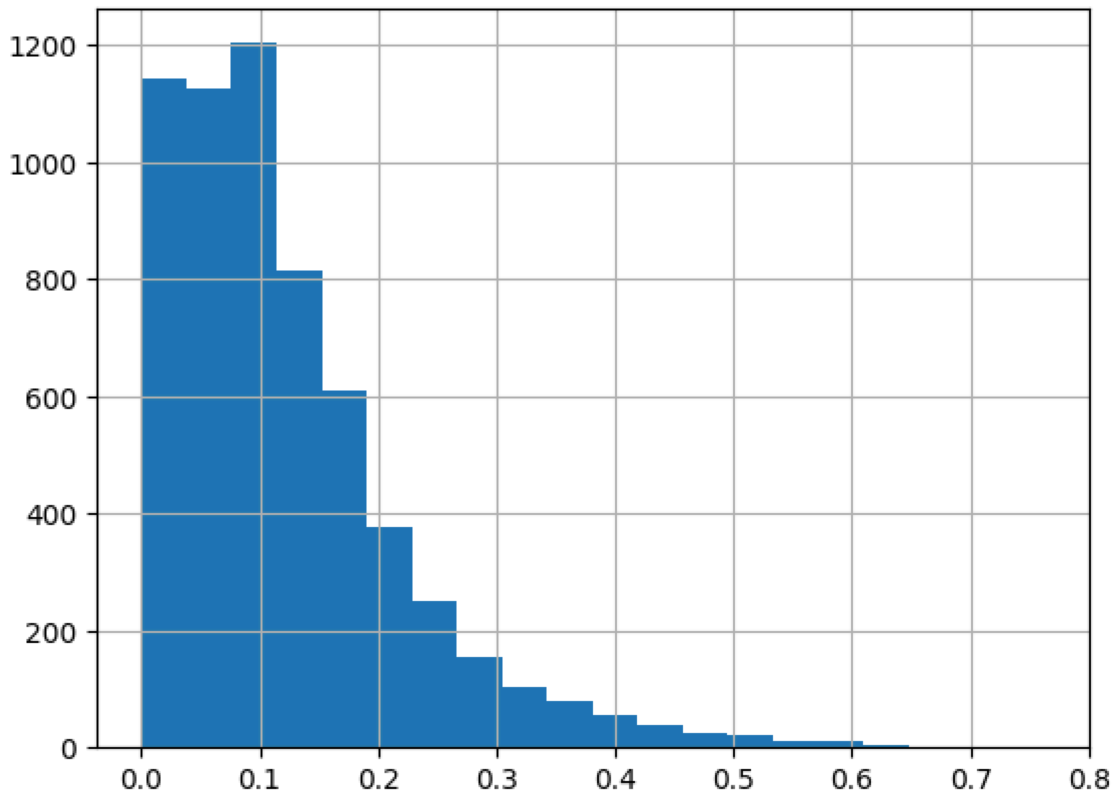
```
avg: 0.7941254267523916
```



In [ ]:

```
overlap=[]
for user in df_ratings['UserID'].unique():
    recommendations = model.topN(user=user, n=100)
    user_movies = df_ratings.loc[(df_ratings['UserID']==user)]['MovieID']
    valid_rec = set(recommendations).intersection(set(user_movies)) # I can only me
    relevant_items = df_ratings.loc[(df_ratings['UserID']==user) & (df_ratings['Rat
    try:
        _ = len(set(recommendations).intersection(set(relevant_items))) / len(set(r
    except:
        _ = 0
    overlap.append(_)

overlap = np.array(overlap)
print('avg:', overlap.mean())
pd.Series(overlap).hist(bins=20)
plt.show()
```

```
avg: 0.1206196870105706
```



Recommender System based Pearson Correlation (User-based)

In [ ]:
```
similar_users = user_movie_matrix.T.corrwith(user_movie_matrix.T.loc[:,5])
similar_users = similar_users.sort_values(ascending=False)
similar_users[1:].index[:20]
```

Out[ ]:
```
Index([1484, 5452,  281, 3538, 1407, 5749, 5826, 5718, 5496, 3240, 1636, 2918,
       1255, 4607,  225,  944, 1104, 2870, 5047, 4995],
      dtype='int64', name='UserID')
```

In [ ]:
```
user_movies_watched = user_movie_matrix.T.loc[:, 5]  # User ID 4 in your example
user_movies_watched  = user_movies_watched.astype('int')
movies_already_watched = user_movies_watched[user_movies_watched != 0].index

top_similar_users = similar_users.index[:10]  # top 10 similar users

similar_users_movies = user_movie_matrix.loc[top_similar_users]

movie_recommendation_scores = similar_users_movies.mean(axis=0)
movie_recommendation_scores.drop(movies_already_watched, inplace=True)

recommended_movies = movie_recommendation_scores.sort_values(ascending=False).head(
recommended_movie_name = recommended_movies.index

print('Rcommended movies are :\n')
for i in recommended_movie_name:
  print(i)
```

```
Rcommended movies are :

Shakespeare in Love
Fugitive, The
Boogie Nights
To Die For
Clerks
Toy Story 2
Crying Game, The
What's Eating Gilbert Grape
Groundhog Day
Terminator 2: Judgment Day
```

In [ ]: