Movie Rating Recommendations

🎯 Objective:

The goal of this project is to develop a personalized movie recommender system that suggests movies to users based on their past ratings and the preferences of similar users. This system enhances user experience by providing tailored recommendations.

📂 Dataset:

The dataset used for this project consists of three files:

1. 📊 **ratings.dat** - Contains user ratings for movies.
2. 👤 **users.dat** - Includes demographic information of users.
3. 🎥 **movies.dat** - Lists movie titles and genres.

📜 Data Description:

1. ⭐ Ratings Data:

- **Format:** `UserID::MovieID::Rating::Timestamp`
- **Users:** 6,040 (each with at least 20 ratings)
- **Movies:** 3,952
- **Ratings:** 1 to 5 stars (whole-star ratings only)
- **Timestamp:** Recorded in seconds

2. 👥 Users Data:

- **Format:** `UserID::Gender::Age::Occupation::Zip-code`
- **Gender:** Male (M) or Female (F)
- **Age Groups:** Ranges from under 18 to 56+
- **Occupation:** 21 categories, including academic, artist, executive, programmer, etc.
- **Voluntary demographic details**

3. 🎞️ Movies Data:

- **Format:** `MovieID::Title::Genres`
- **Titles:** Match IMDb records
- **Genres:** 18 categories, including Action, Comedy, Drama, Sci-Fi, etc.

🛠️ Key Concepts & Techniques:

- 🔍 **Recommender Engine**: Personalized movie suggestions
- 🤝 **Collaborative Filtering**:
  - **User-Based Filtering**: Finding similar users
  - **Item-Based Filtering**: Finding similar movies
- 📈 **Pearson Correlation**: Measuring similarity
- 📌 **Cosine Similarity & Nearest Neighbors**: Identifying closest matches
- 🔢 **Matrix Factorization**: Improving recommendations.

Importing Necessary Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.impute import KNNImputer
from collections import defaultdict
from scipy import sparse
!pip install scikit-surprise
from surprise import SVD
from surprise import Dataset
from surprise import Reader
from surprise.model_selection import cross_validate
from scipy.stats import pearsonr
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
import warnings
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

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

🚀 Exploratory Data Analysis & Feature Engineering

```python
rows, columns = movies.shape
print(f'Rows in the Movies dataset: {rows}')
print(f'Columns in the Movies dataset: {columns}')
movies.info()
movies.drop(columns=['Unnamed: 1','Unnamed: 2'],inplace=True)
delimeter = "::"
movies = movies['Movie ID::Title::Genres'].str.split(delimeter,expand=True)
movies.columns = ['MovieID','Title','Genres']
movies.describe()
movies.sample(5)
```

```
Rows in the Movies dataset: 3883
Columns in the Movies dataset: 3
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Movie ID::Title::Genres  3883 non-null   object
 1   Unnamed: 1               100 non-null    object
 2   Unnamed: 2               51 non-null     object
dtypes: object(3)
memory usage: 91.1+ KB
```

	MovieID	Title	Genres
2041	2110	Dead Men Don't Wear Plaid (1982)	Comedy|Crime|Thriller
2981	3050	Light It Up (1999)	Drama
1053	1067	Damsel in Distress, A (1937)	Comedy|Musical|Romance
2661	2730	Barry Lyndon (1975)	Drama
3820	3890	Back Stage (2000)	Documentary

```python
rows, columns = ratings.shape
print(f'Rows in the Ratings dataset: {rows}')
print(f'Columns in the Ratings dataset: {columns}')
ratings.info()
delimeter = "::"
ratings = ratings['UserID::MovieID::Rating::Timestamp'].str.split(delimeter,expand=True)
ratings.columns = ['UserID','MovieID','Rating','Timestamp']
ratings.sample(5)
```

```
Rows in the Ratings dataset: 1000209
Columns in the Ratings dataset: 1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 1 columns):
 #   Column                              Non-Null Count    Dtype
---  ------                              --------------    -----
 0   UserID::MovieID::Rating::Timestamp  1000209 non-null  object
dtypes: object(1)
memory usage: 7.6+ MB
```

	UserID	MovieID	Rating	Timestamp
372846	2177	3104	4	974610143
298048	1767	1845	3	975387208
142430	919	3104	4	975206662
512725	3163	2671	4	968886227
768784	4579	3504	4	970626208

```
rows, columns = users.shape
print(f'Rows in the Users dataset: {rows}')
print(f'Columns in the Users dataset: {columns}')
users.info()
delimeter = "::"
users =
users['UserID::Gender::Age::Occupation::Zip-code'].str.split(delimeter,exp
and=True)
users.columns = ['UserID','Gender','Age','Occupation','Zip-code']
users.sample(5)
```

```
Rows in the Users dataset: 6040
Columns in the Users dataset: 1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 1 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   UserID::Gender::Age::Occupation::Zip-code  6040 non-null   object
dtypes: object(1)
memory usage: 47.3+ KB
```

	UserID	Gender	Age	Occupation	Zip-code
2787	2788	M	56	1	90049
3761	3762	M	50	6	11746
4825	4826	M	45	12	55436
3881	3882	M	56	14	55337
5345	5346	F	25	3	97330

functools.reduce() to merge multiple DataFrames efficiently in a loop-like fashion.

```
from functools import reduce
dfs = [movies, ratings, users]
df = reduce(lambda left, right: pd.merge(left, right,
on=left.columns.intersection(right.columns).tolist(), how='inner'), dfs)
rows, columns = df.shape
print(f'Rows in the dataset: {rows}')
print(f'Columns in the dataset: {columns}')
df.sample(10)
```

```
Rows in the dataset: 1000209
Columns in the dataset: 10
```

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
150733	544	Striking Distance (1993)	Action	1941	3	974949093	M	35	17	94550
880101	3328	Ghost Dog: The Way of the Samurai (1999)	Crime|Drama	180	3	997601103	M	45	12	01603
332638	1231	Right Stuff, The (1983)	Drama	4526	4	964808533	M	25	20	10012
378209	1298	Pink Floyd - The Wall (1982)	Drama|Musical|War	1699	2	974710640	F	25	19	98102
645947	2389	Psycho (1998)	Crime|Horror|Thriller	2592	4	974057615	M	50	7	80004-4448
668024	2450	Howard the Duck (1986)	Adventure|Children's|Sci-Fi	3589	1	966657317	F	45	0	80010
759408	2804	Christmas Story, A (1983)	Comedy|Drama	2233	4	974596542	F	35	3	60187
510491	1911	Doctor Dolittle (1998)	Comedy	5532	4	959619455	M	25	17	27408
255197	1041	Secrets & Lies (1996)	Drama	4602	1	978838176	M	35	7	10021
955937	3699	Starman (1984)	Adventure|Drama|Romance|Sci-Fi	2106	3	976122532	F	18	20	495321

```
df = df.astype({'Age': 'int32', 'Rating': 'int32'})  # Convert age & ratings to integer
df['Timestamp'] = pd.to_datetime(df['Timestamp'], unit='s')  # Convert Unix timestamp to datetime
df['ReleaseYear'] = df['Title'].str.extract(r'\((\d{4})\)$')[0]
df.sample(10)
```

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code	ReleaseYear
967284	3744	Shaft (2000)	Action|Crime	1676	3	2000-11-20 09:18:53	M	18	4	91301	2000
313419	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventu	1175	4	2000-11-22 01:41:51	F	25	2	90020	1983
208003	858	Godfather, The (1972)	Action|Crime|Drama	3481	5	2000-08-24 00:59:14	M	18	4	94122	1972
711751	2650	Ghost of Frankenstein, The (1942)	Horror	3018	4	2001-02-20 22:04:56	M	35	0	45242	1942
152700	551	Nightmare Before Christmas, The (1993)	Children's|Comedy|Musical	5524	1	2000-05-29 22:16:23	F	1	10	26505	1993
849345	3175	Galaxy Quest (1999)	Adventure|Comedy|Sci-Fi	1117	4	2000-12-01 17:37:21	M	18	14	10017	1999
283838	1135	Private Benjamin (1980)	Comedy	1181	2	2000-12-06 01:54:30	M	35	7	20716	1980
961905	3712	Soapdish (1991)	Comedy	2899	4	2000-10-19 01:40:29	M	35	14	94133	1991
475031	1683	Wings of the Dove, The (1997)	Drama|Romance|Thriller	3067	5	2000-09-26 19:33:51	F	25	0	02148	1997
117661	435	Coneheads (1993)	Comedy|Sci-Fi	5555	3	2000-05-30 01:24:57	M	1	10	37830	1993

Greetings bins for ages

```
bins = [0, 17, 24, 34, 44, 49, 55, float('inf')]
labels = ["Under 18", "18-24", "25-34", "35-44", "45-49", "50-55", "56+"]
df['Age'] = pd.cut(df['Age'], bins=bins, labels=labels, right=True)
```

```
occupation_mapping = {
    '0': "other",
    '1': "academic/educator",
    '2': "artist",
    '3': "clerical/admin",
```

```python
    '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"
}

df['Occupation'] =
df['Occupation'].astype(str).map(occupation_mapping).fillna("other")
df.sample(10)
```

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code	ReleaseYear
523134	1954	Rocky (1976)	Action|Drama	482	4	2000-12-07 19:54:53	M	25-34	sales/marketing	55305	1976
947945	3662	Puppet Master III: Toulon's Revenge (1991)	Horror|Sci-Fi|Thrille	516	2	2000-12-08 13:11:09	F	56+	sales/marketing	55033	1991
30248	85	Angels and Insects (1995)	Drama|Romance	1396	1	2000-11-20 23:47:32	F	25-34	programmer	94530	1995
913325	3484	Skulls, The (2000)	Thriller	301	1	2000-12-11 03:10:48	M	18-24	college/grad student	61820	2000
64132	246	Hoop Dreams (1994)	Documentary	996	5	2000-11-24 21:57:11	M	25-34	technician/engineer	98102	1994
773732	2875	Sommersby (1993)	Drama|Mystery|Romance	531	3	2001-01-05 21:27:37	F	18-24	sales/marketing	22206	1993
204339	837	Matilda (1996)	Children's|Comedy	4265	3	2000-08-03 12:10:23	M	25-34	programmer	03060	1996
294960	1193	One Flew Over the Cuckoo's Nest (1975)	Drama	2532	5	2000-11-12 18:10:06	F	56+	retired	37917	1975
724551	2699	Arachnophobia (1990)	Action|Comedy|Sci-Fi|Thriller	4747	4	2000-07-10 15:15:41	M	18-24	college/grad student	26201	1990
664843	2429	Mighty Joe Young (1998)	Adventure|Children's|Drama	5787	3	2001-09-22 01:45:52	M	25-34	writer	90802	1998

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

```
array(['1995', '1994', '1996', nan, '1976', '1993', '1992', '1988',
       '1967', '1977', '1965', '1982', '1962', '1990', '1991', '1989',
       '1937', '1940', '1969', '1981', '1973', '1970', '1960', '1955',
       '1956', '1959', '1968', '1980', '1975', '1948', '1943', '1950',
       '1987', '1997', '1974', '1958', '1972', '1998', '1952', '1951',
       '1957', '1961', '1954', '1934', '1944', '1963', '1942', '1941',
       '1964', '1953', '1939', '1947', '1946', '1945', '1938', '1935',
       '1936', '1926', '1949', '1932', '1930', '1971', '1979', '1986',
       '1966', '1978', '1985', '1983', '1984', '1933', '1931', '1922',
       '1927', '1929', '1928', '1999', '1925', '1919', '1923', '2000',
       '1920', '1921'], dtype=object)
```

KNN for missing values in Released Year:

```
# Convert 'ReleaseYear' to numeric
df['ReleaseYear'] = pd.to_numeric(df['ReleaseYear'], errors='coerce')

# Columns to use for KNN (excluding categorical ones)
numeric_cols = df.select_dtypes(include=[np.number]).columns

# KNN imputer
knn_imputer = KNNImputer(n_neighbors=3)

# Apply KNN
df[numeric_cols] = knn_imputer.fit_transform(df[numeric_cols])

# Convert ReleaseYear back to integer (since KNN outputs floats)
df['ReleaseYear'] = df['ReleaseYear'].astype(int)
```

Cleaning the Title column using strip and reg expression:

```
df['Cleaned_Title'] = df['Title'].str.replace(r'\(\d{4}\)', '',
regex=True).str.strip()
```

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

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code	ReleaseYear	Cleaned_Title
719121	2686	Red Violin, The (Le Violon rouge) (1998)	Drama|Mystery	926	3.0	2001-11-24 23:17:33	M	Under 18	K-12 student	07869	1998	Red Violin, The (Le Violon rouge)
754811	2792	Airplane II: The Sequel (1982)	Comedy	4028	5.0	2000-08-13 20:51:42	M	25-34	college/grad student	02140	1982	Airplane II: The Sequel
413353	1394	Raising Arizona (1987)	Comedy	3248	4.0	2000-09-07 05:13:58	F	25-34	executive/managerial	94115	1987	Raising Arizona
547627	2011	Back to the Future Part II (1989)	Comedy|Sci-Fi	4869	3.0	2000-07-06 12:37:28	F	45-49	doctor/health care	04578	1989	Back to the Future Part II
529300	1962	Driving Miss Daisy (1989)	Drama	3963	4.0	2000-12-28 05:49:17	M	50-55	writer	30030	1989	Driving Miss Daisy
608989	2248	Say Anything... (1989)	Comedy|Drama|Romance	5126	4.0	2000-06-28 01:15:37	M	25-34	lawyer	02453	1989	Say Anything...
34937	110	Braveheart (1995)	Action|Drama|War	2015	5.0	2000-11-19 22:53:15	M	18-24	college/grad student	01003	1995	Braveheart
78773	296	Pulp Fiction (1994)	Crime|Drama	3584	4.0	2000-08-19 14:11:23	M	18-24	other	76543	1994	Pulp Fiction
735282	2718	Drop Dead Gorgeous (1999)	Comedy	1230	5.0	2000-11-21 21:35:24	M	25-34	lawyer	95125	1999	Drop Dead Gorgeous
116231	432	City Slickers II: The Legend of Curly's Gold (...	Comedy|Wester	3183	2.0	2000-09-12 03:15:43	M	50-55	executive/managerial	60510	1994	City Slickers II: The Legend of Curly's Gold

Explanation of the Regular Expression

- ^(.*), (The|A|An)$
- (.*): Captures everything before the comma.
- (The|A|An): Matches ", The", ", A", or ", An" at the end of the title.
- List item The function swaps the position of these words.

```python
def fix_title(title):
    match = re.match(r'^(.*), (The|A|An)$', title)
    if match:
        return f"{match.group(2)} {match.group(1)}"
    return title

df['Cleaned_Title'] = df['Cleaned_Title'].apply(fix_title)
df.drop(columns=['Title'],inplace=True)
```

```python
# Display only columns with missing values
missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0]  # Filter non-zero missing values
print("Missing Values in Each Column:")
print(missing_values)
```

```
Missing Values in Each Column:
Genres    4065
dtype: int64
```

```python
# Count duplicate rows directly
num_duplicates = df.duplicated().sum()

# Print the result
print(f'The number of duplicate rows: {num_duplicates}')
```

```
The number of duplicate rows: 0
```

```python
df['Genres'] = df.groupby('Cleaned_Title')['Genres'].transform(lambda x:
x.fillna(x.mode()[0]) if not x.mode().empty else "Unknown")
```

```python
df.isnull().sum()
```

	0
MovieID	0
Genres	0
UserID	0
Rating	0
Timestamp	0
Gender	0
Age	0
Occupation	0
Zip-code	0
ReleaseYear	0
Cleaned_Title	0

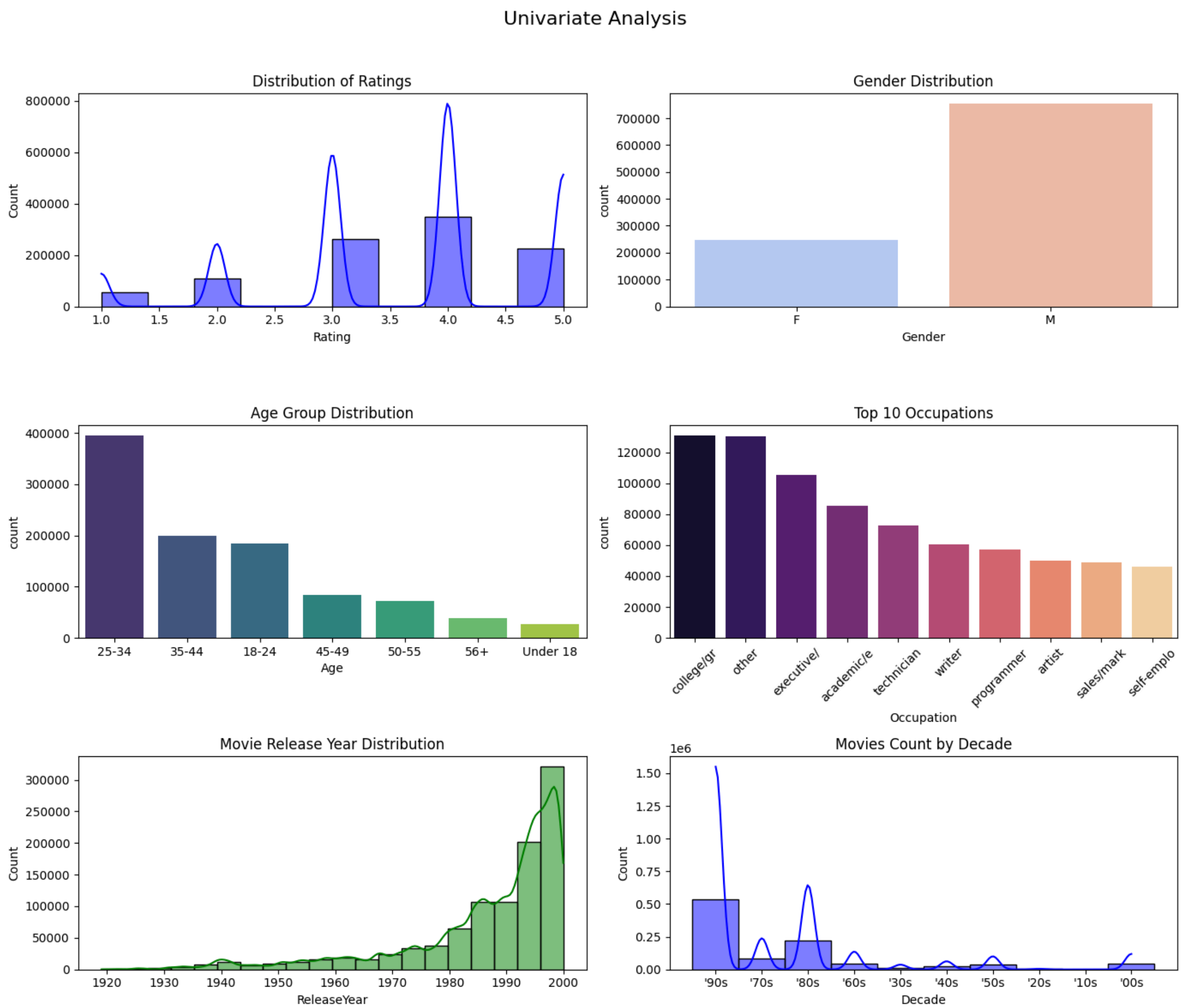
Upon further analysis and feature engineering our data is ready to be used for further analysis.

Univariate & BiVariant analysis

```
df['Decade'] = df['ReleaseYear'].apply(lambda x: f"'{str(x // 10 * 10)[2:]}s")
most_common_decade = df['Decade'].value_counts().idxmax()
print(f"Most movies were released in the {most_common_decade}'")
```

```
fig, axes = plt.subplots(3, 2, figsize=(14, 12))
fig.suptitle("Univariate Analysis", fontsize=16)

# Histogram of Ratings
sns.histplot(df['Rating'], bins=10, kde=True, ax=axes[0,0], color='blue')
axes[0,0].set_title('Distribution of Ratings')

# Countplot of Gender
sns.countplot(x=df['Gender'], ax=axes[0,1], palette='coolwarm')
axes[0,1].set_title('Gender Distribution')

# Countplot of Age Group
sns.countplot(x=df['Age'], order=df['Age'].value_counts().index, ax=axes[1,0], palette='viridis')
axes[1,0].set_title('Age Group Distribution')

# Countplot of Top 10 Occupations
sns.countplot(x=df['Occupation'], order=df['Occupation'].value_counts().index[:10], ax=axes[1,1], palette='magma')
axes[1,1].set_title('Top 10 Occupations')
x_labels = [label.get_text()[:10] for label in axes[1,1].get_xticklabels()]
axes[1,1].set_xticklabels(x_labels, rotation=45)

# Histogram of Movie Release Year
sns.histplot(df['ReleaseYear'], bins=20, kde=True, ax=axes[2,0], color='green')
axes[2,0].set_title('Movie Release Year Distribution')

# Histogram of Movies Count by Decade
```

```python
sns.histplot(df['Decade'], bins=10, kde=True, ax=axes[2,1], color='blue')
axes[2,1].set_title('Movies Count by Decade')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Insights from Univariate Analysis:

1️⃣ **Ratings Distribution:** Most ratings are **3, 4, or 5 stars**, indicating a positive bias. Peaks at whole numbers suggest users prefer rounded ratings.

2️⃣ **Gender Distribution:** The dataset is **male-dominated**, with significantly more ratings from male users.

3️⃣ **Age Group Distribution:** Users aged **25-34** are the most active, followed by **35-44** and **18-24**. Engagement is lowest for users **under 18 and above 50**.

4️⃣ **Top Occupations: College students, executives, and academics** are the most engaged users. Tech professionals and writers also contribute significantly.

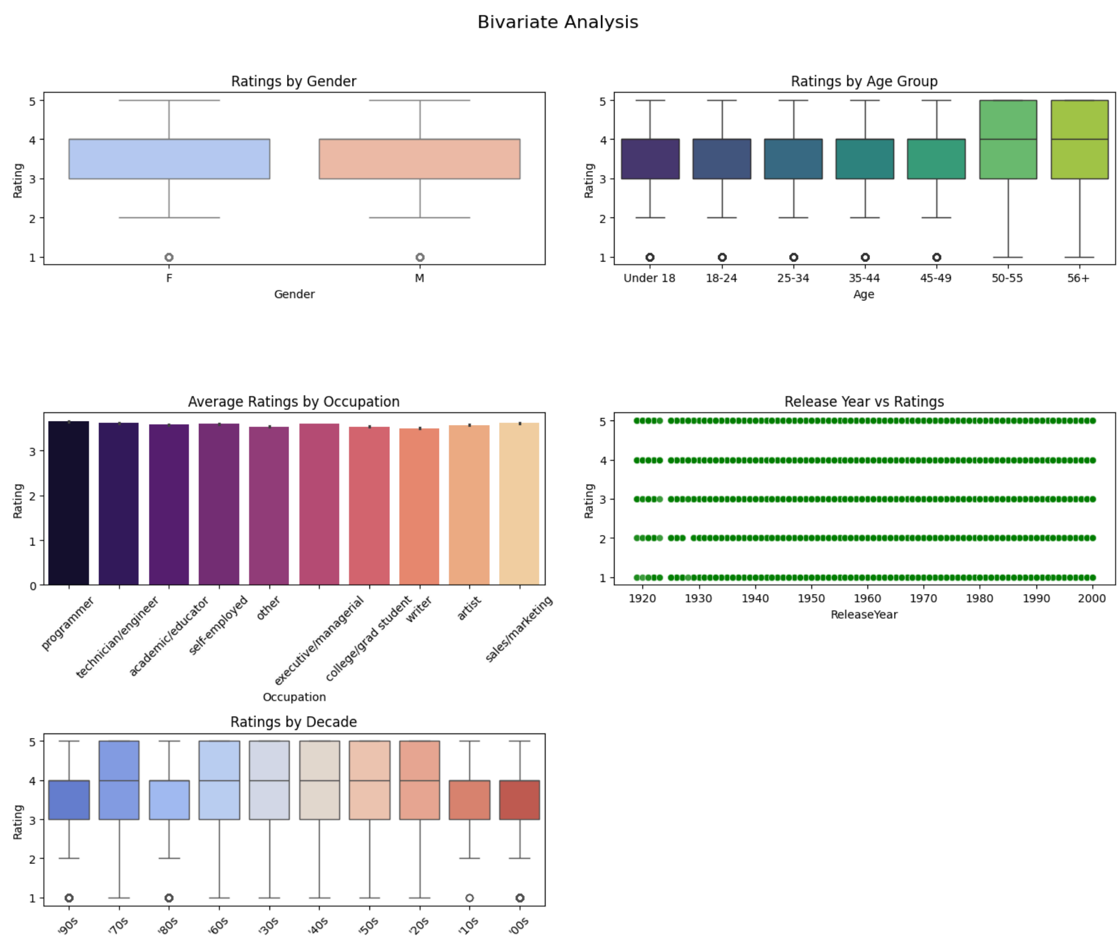
5️⃣ **Movie Release Year Distribution:** Most rated movies are from the **1980s onward**, peaking in the **1990s and early 2000s**. Older films have fewer ratings.

6️⃣ **Movies Count by Decade:** The **1990s** dominate in terms of movie count, followed by the **2000s**, with a sharp drop for earlier decades.

📌 **Summary:** Engagement is highest among **young male users** (25-34), with a preference for **modern movies and higher ratings**. The dataset is skewed towards **recent films and tech-savvy users**, impacting recommendation strategies.

```
fig, axes = plt.subplots(3, 2, figsize=(14, 12))
fig.suptitle("Bivariate Analysis", fontsize=16)
# Boxplot of Ratings vs Gender
sns.boxplot(x='Gender', y='Rating', data=df, ax=axes[0,0],
palette='coolwarm')
axes[0,0].set_title('Ratings by Gender')
# Boxplot of Ratings vs Age Group
sns.boxplot(x='Age', y='Rating', data=df, ax=axes[0,1], palette='viridis')
axes[0,1].set_title('Ratings by Age Group')

# Barplot of Average Ratings by Top 10 Occupations
top_occupations = df['Occupation'].value_counts().index[:10]
df_filtered = df[df['Occupation'].isin(top_occupations)]
sns.barplot(x='Occupation', y='Rating', data=df_filtered, ax=axes[1,0],
palette='magma', estimator=np.mean)
axes[1,0].set_title('Average Ratings by Occupation')
axes[1,0].tick_params(axis='x', rotation=45)
```

```python
# Scatterplot of Release Year vs Ratings
sns.scatterplot(x='ReleaseYear', y='Rating', data=df, ax=axes[1,1],
color='green', alpha=0.5)
axes[1,1].set_title('Release Year vs Ratings')

# Boxplot of Ratings by Decade
sns.boxplot(x='Decade', y='Rating', data=df, ax=axes[2,0],
palette='coolwarm')
axes[2,0].set_title('Ratings by Decade')
axes[2,0].tick_params(axis='x', rotation=45)
axes[2,1].axis("off")
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Insights from Bivariate Analysis

1️⃣**Ratings by Gender:** Both **males and females** have a similar median rating, but females show slightly lower variability, suggesting **more consistent ratings**.

2️⃣**Ratings by Age Group:** Older users (50+) tend to **give higher ratings**, while younger users (18-34) have a wider spread, indicating **more critical reviews**.

3️⃣**Average Ratings by Occupation: Programmers, engineers, and academics** give the highest ratings, while writers and sales/marketing professionals tend to be slightly more critical.

4️⃣**Release Year vs Ratings:** Ratings remain **consistent across decades**, showing no strong correlation between **movie age and rating**.

5️⃣**Ratings by Decade:** Recent decades (1990s-2000s) show **higher median ratings**, while older decades (1960s-1980s) have **wider variation**, suggesting **mixed opinions on classic films**.

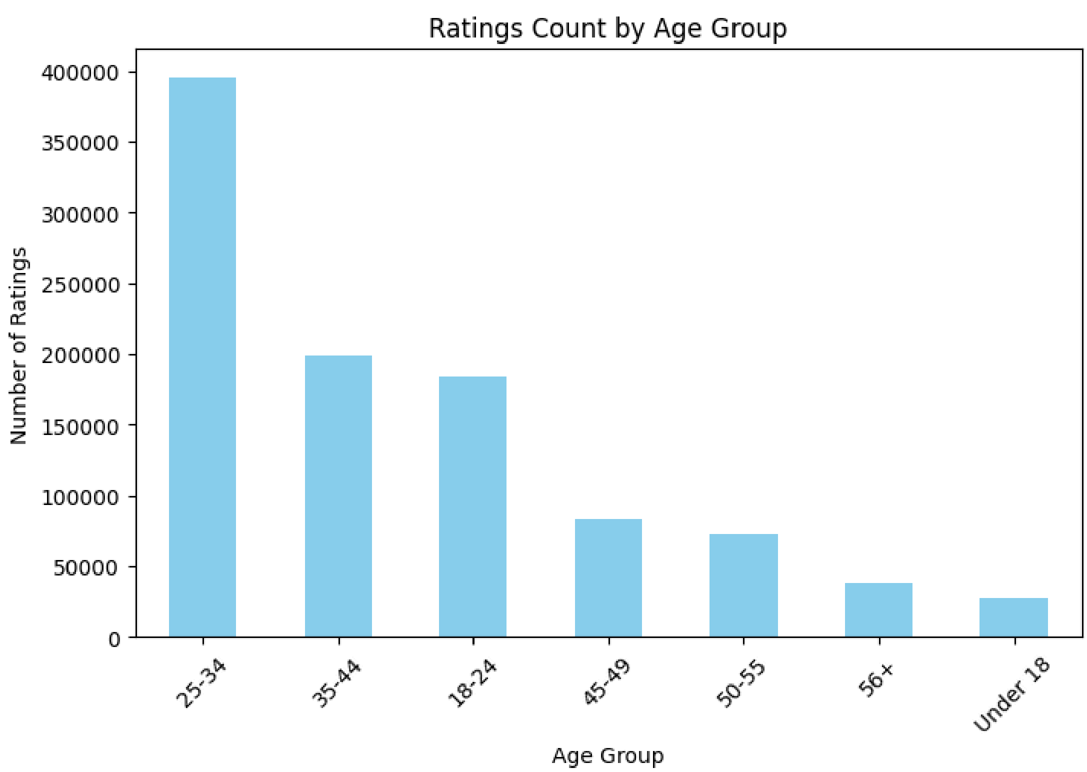
📌 **Summary: Older users and technical professionals** rate movies more favorably, while **younger audiences and marketing professionals** are more critical. **Recent movies** tend to receive higher ratings, possibly due to recency bias.

Now let's answer some basic questions:

1. Users of which age group have watched and rated the most number of movies?

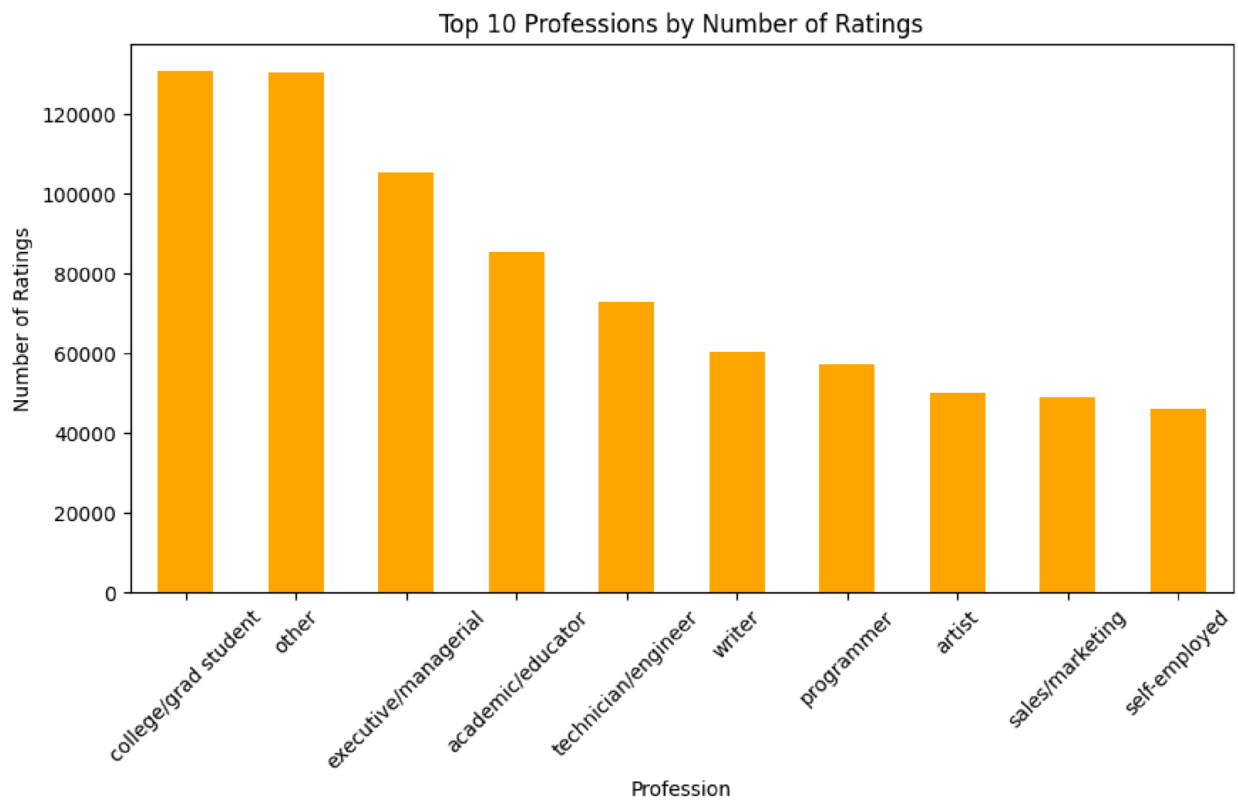
```
age_group_ratings = df.groupby('Age')['Rating'].count()
most_active_age_group = age_group_ratings.idxmax()
print(f"The age group that has rated the most movies is:
{most_active_age_group}")

plt.figure(figsize=(8,5))
age_group_ratings.sort_values(ascending=False).plot(kind='bar',
color='skyblue')
plt.xlabel("Age Group")
plt.ylabel("Number of Ratings")
plt.title("Ratings Count by Age Group")
plt.xticks(rotation=45)
plt.show()
```

```
The age group that has rated the most movies is: 25-34
```



2. Users belonging to which profession have watched and rated the most movies?

```
profession_ratings = df.groupby('Occupation')['Rating'].count()
# The top 10 professions
top_10_professions =
profession_ratings.sort_values(ascending=False).head(10)

plt.figure(figsize=(10, 5))
top_10_professions.plot(kind='bar', color='orange')
plt.xlabel("Profession")
plt.ylabel("Number of Ratings")
plt.title("Top 10 Professions by Number of Ratings")
plt.xticks(rotation=45)
plt.show()
```



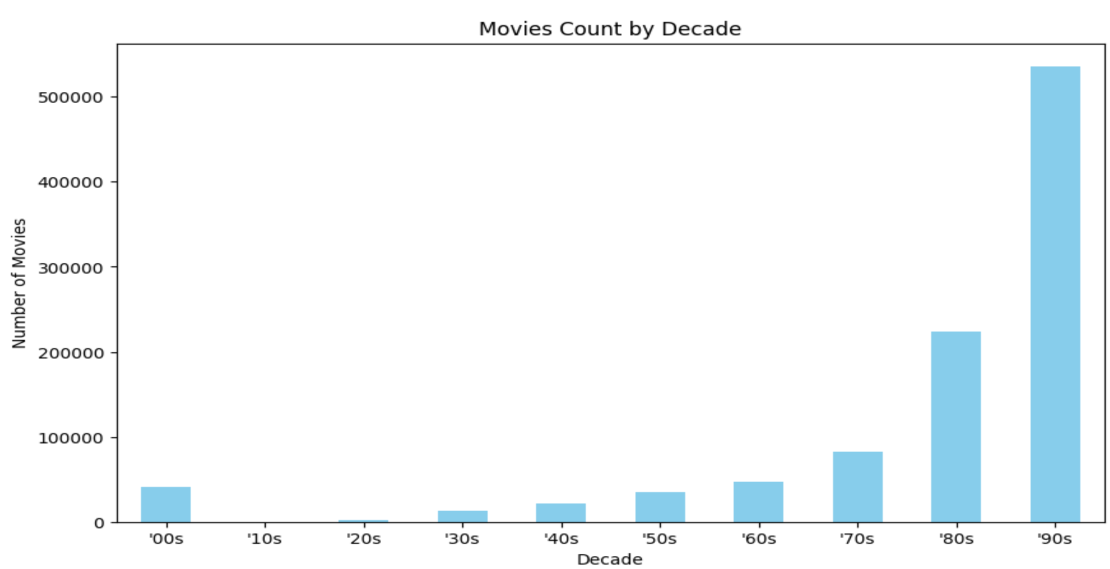
3. Most of the users in our dataset who've rated the movies are Male. (T/F).

```
gender_ratings = df.groupby('Gender')['Rating'].count()
most_rated_gender = gender_ratings.idxmax()
print(f"Most of the users who rated movies are {most_rated_gender}.")
```

```
Most of the users who rated movies are M.
```

4. Most of the movies present in our dataset were released in which decade? 70s b. 90s c. 50s d.80s

```
df['Decade'] = df['ReleaseYear'].apply(lambda x: f"'{str(x // 10 *
10)[2:]}s")
most_common_decade = df['Decade'].value_counts().idxmax()
print(f"Most movies were released in the {most_common_decade}'")
plt.figure(figsize=(10,6))
df['Decade'].value_counts().sort_index().plot(kind='bar', color='skyblue')
plt.xlabel("Decade")
plt.ylabel("Number of Movies")
plt.title("Movies Count by Decade")
plt.xticks(rotation=0)
plt.show()
```



5. Find the movie with the highest number of ratings

```
top_movie = df.groupby('Cleaned_Title')['Rating'].count().idxmax()
top_movie_ratings = df.groupby('Cleaned_Title')['Rating'].count().max()

print(f"The movie with the maximum number of ratings is: {top_movie} with
{top_movie_ratings} ratings.")
```

```
The movie with the maximum number of ratings is: American Beauty with 3428 ratings.
```

Model Building: Recommendation System

<table>
<tr><th></th><th>MovieID</th><th>Genres</th><th>UserID</th><th>Rating</th><th>Timestamp</th><th>Gender</th><th>Age</th><th>Occupation</th><th>Zip-code</th><th>ReleaseYear</th><th>Cleaned_Title</th><th>Decade</th></tr>
<tr><td>0</td><td>1</td><td>Animation|Children's|Comedy</td><td>1</td><td>5.0</td><td>2001-01-06 23:37:48</td><td>F</td><td>Under 18</td><td>K-12 student</td><td>48067</td><td>1995</td><td>Toy Story</td><td>'90s</td></tr>
<tr><td>1</td><td>1</td><td>Animation|Children's|Comedy</td><td>6</td><td>4.0</td><td>2000-12-31 04:30:08</td><td>F</td><td>50-55</td><td>homemaker</td><td>55117</td><td>1995</td><td>Toy Story</td><td>'90s</td></tr>
<tr><td>2</td><td>1</td><td>Animation|Children's|Comedy</td><td>8</td><td>4.0</td><td>2000-12-31 03:31:36</td><td>M</td><td>25-34</td><td>programmer</td><td>11413</td><td>1995</td><td>Toy Story</td><td>'90s</td></tr>
<tr><td>3</td><td>1</td><td>Animation|Children's|Comedy</td><td>9</td><td>5.0</td><td>2000-12-31 01:25:52</td><td>M</td><td>25-34</td><td>technician/engineer</td><td>61614</td><td>1995</td><td>Toy Story</td><td>'90s</td></tr>
<tr><td>4</td><td>1</td><td>Animation|Children's|Comedy</td><td>10</td><td>5.0</td><td>2000-12-31 01:34:34</td><td>F</td><td>35-44</td><td>academic/educator</td><td>95370</td><td>1995</td><td>Toy Story</td><td>'90s</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>1000204</td><td>3952</td><td>Drama|Thriller</td><td>5812</td><td>4.0</td><td>2001-06-09 07:34:59</td><td>F</td><td>25-34</td><td>executive/managerial</td><td>92120</td><td>2000</td><td>The Contender</td><td>'00s</td></tr>
<tr><td>1000205</td><td>3952</td><td>Drama|Thriller</td><td>5831</td><td>3.0</td><td>2001-04-02 14:52:05</td><td>M</td><td>25-34</td><td>academic/educator</td><td>92120</td><td>2000</td><td>The Contender</td><td>'00s</td></tr>
<tr><td>1000206</td><td>3952</td><td>Drama|Thriller</td><td>5837</td><td>4.0</td><td>2002-01-24 20:04:16</td><td>M</td><td>25-34</td><td>executive/managerial</td><td>60607</td><td>2000</td><td>The Contender</td><td>'00s</td></tr>
<tr><td>1000207</td><td>3952</td><td>Drama|Thriller</td><td>5927</td><td>1.0</td><td>2001-01-18 21:15:37</td><td>M</td><td>35-44</td><td>sales/marketing</td><td>10003</td><td>2000</td><td>The Contender</td><td>'00s</td></tr>
<tr><td>1000208</td><td>3952</td><td>Drama|Thriller</td><td>5998</td><td>4.0</td><td>2001-09-29 16:30:44</td><td>M</td><td>18-24</td><td>college/grad student</td><td>61820</td><td>2000</td><td>The Contender</td><td>'00s</td></tr>
</table>

Build a Recommender System based on Pearson Correlation.

```
# Aggregate ratings by the mean
res_pearson = df[['UserID', 'Cleaned_Title', 'Rating']].groupby(['UserID',
'Cleaned_Title']).mean().reset_index()

# Movie-User Matrix
movie_user_matrix = res_pearson.pivot(index='UserID',
columns='Cleaned_Title', values='Rating')

movie_user_matrix.fillna(0, inplace=True)

# Display first 10 rows
movie_user_matrix.head(10)
```

Cleaned_Title	$1,000,000 Duck	'Night Mother	'Til There Was You	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	187	2 Days in the Valley	...	Young Guns	Young Guns II	Young Sherlock Holmes	Young and Innocent	Your Friends and Neighbors	Zachariah	Zero Effect	Zero Kelvin (Kjærlighetens kjøtere)	Zeus and Roxanne	eXistenZ
UserID																					
1	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	0.0	0.0
10	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
100	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	0.0	0.0
1000	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
1001	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	4.0	0.0	0.0	0.0	0.0	5.0
1002	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	0.0	0.0
1003	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	0.0	0.0
1004	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	...	4.0	3.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1005	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	0.0	0.0
1006	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	0.0	3.0

10 rows × 3664 columns

```
df.groupby('Cleaned_Title')['Rating'].count().median(),
df.groupby('Cleaned_Title')['Rating'].count().mean().round(2)
```

```
(123.5, 272.98)
```

🔹 Checking Sparsity in the Movie-User Matrix

Sparsity in a recommender system refers to the percentage of missing ratings in the **Movie-User Matrix**. Since users typically rate only a small subset of movies, the matrix is usually **very sparse**.

1️⃣ Calculate the Sparsity Percentage

We can compute the **sparsity** of the matrix using the formula:

$$\text{Sparsity} = \left(1 - \frac{\text{Number of ratings}}{\text{Total possible ratings}}\right) \times 100$$

```
total_entries = movie_user_matrix.shape[0] * movie_user_matrix.shape[1]
actual_ratings = movie_user_matrix.astype(bool).sum().sum()
sparsity = (1 - (actual_ratings / total_entries)) * 100
print(f"Sparsity of the Movie-User Matrix: {sparsity:.2f}%")
```

```
Sparsity of the Movie-User Matrix: 95.49%
```

Movie-User Matrix has a 95.49% sparsity, meaning that only 4.51% of the possible ratings are filled while the rest are missing. This level of sparsity is very high, which is common in recommender systems but can make collaborative filtering less effective.

```python
# Creating a user-movie matrix
movie_pivot = df.pivot_table(index="Cleaned_Title", columns="UserID",
values="Rating")
movie_pivot.fillna(0, inplace=True)
movie_pivot.sample(10)
```

Code cell output actions

UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991	992	993	994	995	996	997	998	999
Cleaned_Title																					
The Eyes of Tammy Faye	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	0.0	0.0
Mr. Smith Goes to Washington	0.0	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	0.0	0.0
Paradise Lost: The Child Murders at Robin Hood Hills	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	5.0	0.0
Gremlins 2: The New Batch	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
The House on Haunted Hill	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
Spirits of the Dead (Tre Passi nel Delirio)	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	0.0	0.0
Onegin	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	0.0	0.0
Guilty as Sin	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	0.0	0.0
Breaking Away	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
What Dreams May Come	0.0	0.0	0.0	0.0	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

10 rows × 6040 columns

```python
user_id =input("Enter a user_id : ")
user_id_recomm = movie_pivot[user_id]
```

```
Enter a user_id : 100
```

```python
# Compute correlation of all movies with the target user's ratings
similar_movie_user_based = movie_pivot.corrwith(user_id_recomm)
top_similar_users = (
    similar_movie_user_based
    .dropna()  # Remove NaN values to avoid errors
    .sort_values(ascending=False)  # Sort in descending order
    .to_frame(name="Correlation")  # Convert to DataFrame and rename
column
    .head()  # Get top similar movies
)
print(top_similar_users)
```

```
            Correlation
UserID
100         1.000000
4020        0.421207
4706        0.402777
3523        0.398443
2291        0.392623
```

```python
# Creating a user-movie matrix
User_pivot = df.pivot_table(index="UserID", columns="Cleaned_Title",
values="Rating")
User_pivot.fillna(0, inplace=True)
User_pivot.sample(10)
```

Cleaned_Title / UserID	$1,000,000 Duck	'Night Mother	'Til There Was You	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	187	2 Days in the Valley	...	Young Guns	Young Guns II	Young Sherlock Holmes	Young and Innocent	Your Friends and Neighbors	Zachariah	Zero Effect	Zero Kelvin (Kjærlighetens kjøtere)	Zeus and Roxanne	eXistenZ
4290	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	0.0	0.0
4327	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5619	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	0.0	0.0
4088	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	4.0
835	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	0.0	0.0
2885	0.0	0.0	0.0	0.0	0.0	0.0	3.0	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
3657	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	0.0	3.0
501	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	0.0	0.0
4589	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	0.0	0.0
4333	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	0.0	0.0

10 rows × 3664 columns

```python
mov = input("Enter a movie name : ")
mov_rating = User_pivot[mov]
```

```python
similar_movies = movie_pivot.T.corrwith(mov_rating)
top_similar_movies = (
    similar_movies
    .dropna()  # Remove NaN values to avoid errors
    .sort_values(ascending=False)
    .to_frame(name="Correlation")
    .head()
)
print(top_similar_movies)
```

```
                   Correlation
Cleaned_Title
Toy Story             1.000000
Toy Story 2           0.487370
Aladdin               0.470753
The Lion King         0.411131
Groundhog Day         0.407547
```

```
df.groupby('Cleaned_Title')['Rating'].count().median(),
df.groupby('Cleaned_Title')['Rating'].count().mean().round(2)
```

```
(123.5, 272.98)
```

```
📌 What Does This Mean for Filtering Movies?
1️⃣ A lot of movies have fewer than 124 ratings (since median = 123.5).

If you include all movies, your dataset might be too large and contain
many obscure titles.
2️⃣ Some movies have thousands of ratings (since mean = 272.98 > median).

These are the most popular movies, which might dominate recommendations.
```

```
total_entries = movie_pivot.shape[0] * movie_pivot.shape[1]
actual_ratings = movie_pivot.astype(bool).sum().sum()
sparsity = (1 - (actual_ratings / total_entries)) * 100
print(f"Sparsity of the Movie-User Matrix: {sparsity:.2f}%")
```

```
Sparsity of the Movie-User Matrix: 91.65%
```

Movie-User Matrix has a 95.49% sparsity, meaning that only 4.51% of the possible ratings are filled while the rest are missing. This level of sparsity is very high, which is common in recommender systems but can make collaborative filtering less effective.

Cosine Similarity:Movies

```
from sklearn.metrics.pairwise import cosine_similarity
item_sim = cosine_similarity(movie_pivot)
# Convert similarity matrix into DataFrame
item_sim_df = pd.DataFrame(item_sim, index=movie_pivot.index,
columns=movie_pivot.index)
item_sim_df.head()
```

Cleaned_Title	...And Justice for All	10 Things I Hate About You	101 Dalmatians	12 Angry Men	2 Days in the Valley	20 Dates	20,000 Leagues Under the Sea	200 Cigarettes	2001: A Space Odyssey	2010
Cleaned_Title										
...And Justice for All	1.000000	0.075093	0.178928	0.205486	0.195255	0.039933	0.171536	0.114865	0.219975	0.153973
10 Things I Hate About You	0.075093	1.000000	0.215418	0.152429	0.157480	0.143820	0.119347	0.197267	0.190536	0.118003
101 Dalmatians	0.178928	0.215418	1.000000	0.235094	0.188914	0.052619	0.309371	0.191341	0.308287	0.236788
12 Angry Men	0.205486	0.152429	0.235094	1.000000	0.148649	0.083810	0.233758	0.081814	0.366732	0.188498
2 Days in the Valley	0.195255	0.157480	0.188914	0.148649	1.000000	0.047787	0.136155	0.132666	0.227457	0.165833

Cleaned_Title	...	Year of Living Dangerously	Yellow Submarine	Yojimbo	You've Got Mail	Young Frankenstein	Young Guns	Young Guns II	Young Sherlock Holmes	Zero Effect	eXistenZ
Cleaned_Title											
...And Justice for All	...	0.168469	0.151250	0.058466	0.105449	0.183961	0.121312	0.089582	0.153006	0.110867	0.111040
10 Things I Hate About You	...	0.085579	0.132438	0.049510	0.295243	0.176951	0.192240	0.175014	0.141150	0.175771	0.162060
101 Dalmatians	...	0.171446	0.306880	0.109510	0.311154	0.308026	0.252443	0.180836	0.220065	0.157313	0.120762
12 Angry Men	...	0.206717	0.197767	0.130306	0.170438	0.289950	0.169202	0.104518	0.161800	0.133061	0.098731
2 Days in the Valley	...	0.121015	0.139151	0.073292	0.147219	0.179466	0.218880	0.156165	0.142565	0.273992	0.165736

5 rows × 1828 columns

```
def get_similar_movies(movie_name, top_n=5):
    if movie_name not in item_sim_df.index:
        return "Movie not found in the dataset."

    # Sort movies by similarity score (excluding itself)
    similar_movies =
item_sim_df[movie_name].sort_values(ascending=False).iloc[1:top_n+1]

    return similar_movies
```

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

```
Enter a movie name: 12 Angry Men
Cleaned_Title
Amadeus                         0.405141
To Kill a Mockingbird           0.398573
Citizen Kane                    0.386087
Rear Window                     0.379780
The Bridge on the River Kwai    0.378533
Name: 12 Angry Men, dtype: float64
```

Cosine Similarity:Users

```
# Compute user similarity
user_sim = cosine_similarity(User_pivot)
user_sim_mat = pd.DataFrame(user_sim, index=User_pivot.index,
columns=User_pivot.index)
user_sim_mat.head()
```

UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991	992	993	994	995	996	997	998	999
UserID																					
1	1.000000	0.254736	0.123967	0.207800	0.139112	0.110320	0.121384	0.180073	0.103137	0.052816	...	0.079367	0.038048	0.032136	0.067631	0.070052	0.035731	0.170184	0.159267	0.119356	0.122391
10	0.254736	1.000000	0.259052	0.279838	0.158108	0.112659	0.141661	0.431184	0.193049	0.102253	...	0.154060	0.185809	0.083548	0.125607	0.118288	0.146217	0.304110	0.165321	0.133022	0.247883
100	0.123967	0.259052	1.000000	0.306067	0.075625	0.110450	0.358686	0.237292	0.171609	0.099147	...	0.098235	0.097953	0.065152	0.178664	0.271311	0.033754	0.344290	0.204302	0.113522	0.306937
1000	0.207800	0.279838	0.306067	1.000000	0.098971	0.047677	0.201722	0.355619	0.323584	0.130702	...	0.170100	0.076779	0.000000	0.200343	0.380741	0.044404	0.330748	0.172803	0.098456	0.250564
1001	0.139112	0.158108	0.075625	0.098971	1.000000	0.164611	0.053807	0.149848	0.137387	0.134512	...	0.146055	0.026852	0.096868	0.119433	0.092099	0.109539	0.221792	0.103104	0.269555	0.178137

5 rows × 6040 columns

```
def get_similar_users(user_id, top_n=5):
    if user_id not in user_sim_mat.index:
        return "User not found in the dataset."

    # Sort users by similarity score (excluding itself)
    similar_users =
user_sim_mat[user_id].sort_values(ascending=False).iloc[1:top_n+1]

    return similar_users
```

```
user_id = input("Enter a user ID: ")
print(get_similar_users(user_id))
```

```
Enter a user ID: 69
UserID
5998    0.352561
593     0.351725
3791    0.346231
3305    0.344208
966     0.342360
Name: 69, dtype: float64
```

KNN Recommendation System:

```
knn = NearestNeighbors(metric='cosine', algorithm='brute')
knn.fit(movie_pivot)
```

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

```
def recommend_movies(movie_name, n_neighbors=5):
    if movie_name not in User_pivot.columns:
        return "Movie not found in dataset."

    # Get the index of the movie
    movie_idx = User_pivot.columns.get_loc(movie_name)

    # Find similar movies using KNN
    distances, indices = knn.kneighbors(User_pivot.iloc[:,
movie_idx].values.reshape(1, -1), n_neighbors=n_neighbors+1)

    # Fetch movie names
    similar_movies = [User_pivot.columns[i] for i in
indices.flatten()[1:]]  # Exclude input movie

    return similar_movies

movie_name = input("Enter a movie name: ")
print("Recommended Movies: ", recommend_movies(movie_name))
```

```
Enter a movie name: Toy Story
Recommended Movies:  ['Mars Attacks!', 'Chain Reaction', 'A Nightmare on Elm Street 3: Dream Warriors', '3 Ninjas: High Noon On Mega Mountain', 'Alien Escape']
```

Matrix Factorization:

```
reader = Reader(rating_scale=(df_filtered['Rating'].min(),
df_filtered['Rating'].max()))
data = Dataset.load_from_df(df_filtered[['UserID', 'Cleaned_Title',
'Rating']], reader)
svd = SVD()
cross_validate(svd, data, cv=5, verbose=True)
trainset = data.build_full_trainset()
svd.fit(trainset)
```

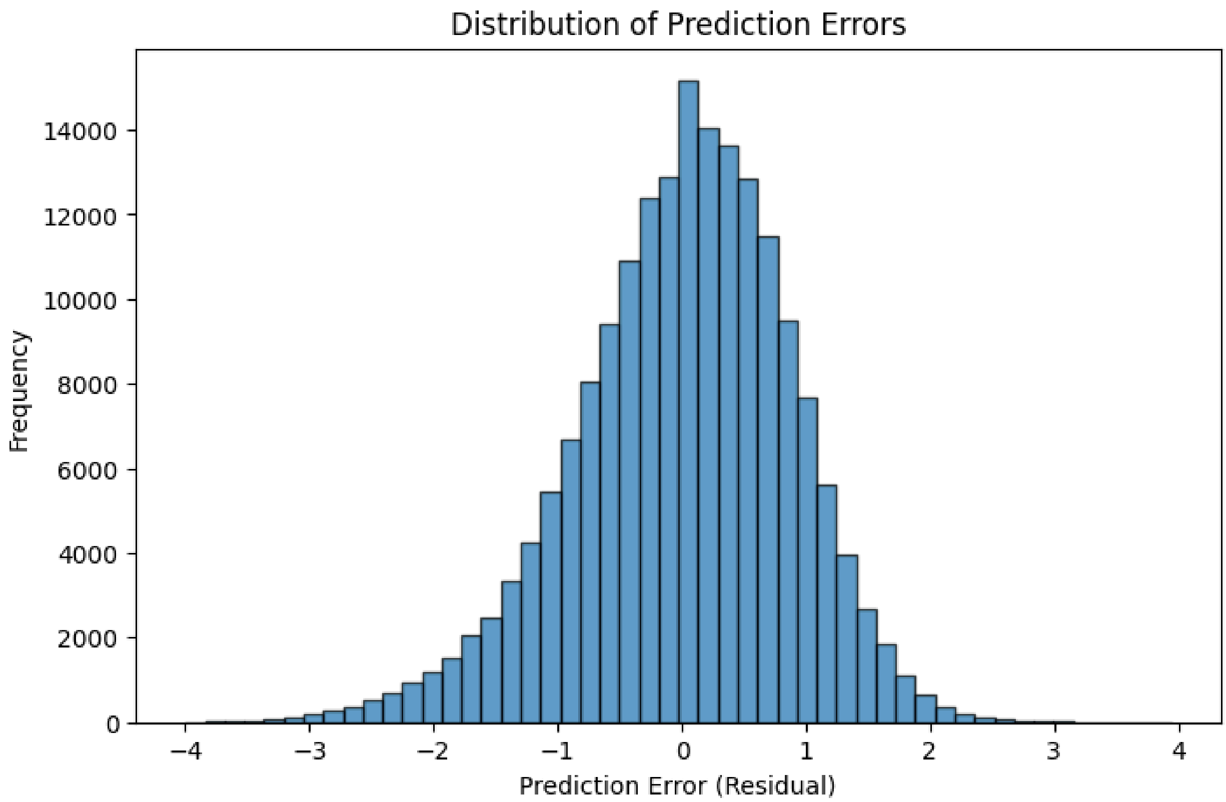
```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8676  0.8699  0.8695  0.8653  0.8671  0.8679  0.0017
MAE (testset)     0.6806  0.6826  0.6822  0.6791  0.6804  0.6810  0.0013
Fit time          22.44   15.52   16.24   16.44   17.96   17.72   2.49
Test time         2.32    1.17    1.09    1.87    2.14    1.72    0.50
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f900b2f13d0>
```

```
def recommend_movies(user_id, n_recommendations=5):
    unique_movies = df_filtered['Cleaned_Title'].unique()
    watched_movies = df_filtered[df_filtered['UserID'] ==
user_id]['Cleaned_Title'].tolist()
       predictions = [svd.predict(user_id, movie) for movie in
unique_movies if movie not in watched_movies]
       recommendations = sorted(predictions, key=lambda x: x.est,
reverse=True)[:n_recommendations]
       return [rec.iid for rec in recommendations]
user_id = int(input("Enter User ID: "))
print("Recommended Movies: ", recommend_movies(user_id))
```

```
Enter User ID: 118
Recommended Movies:  ['The Shawshank Redemption', 'Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (195', 'The Usual Suspects', 'The Wrong Trousers', 'A Close Shave']
```

```
predictions = svd.test(testset)
y_actual = np.array([pred.r_ui for pred in predictions])
y_pred = np.array([pred.est for pred in predictions])
```

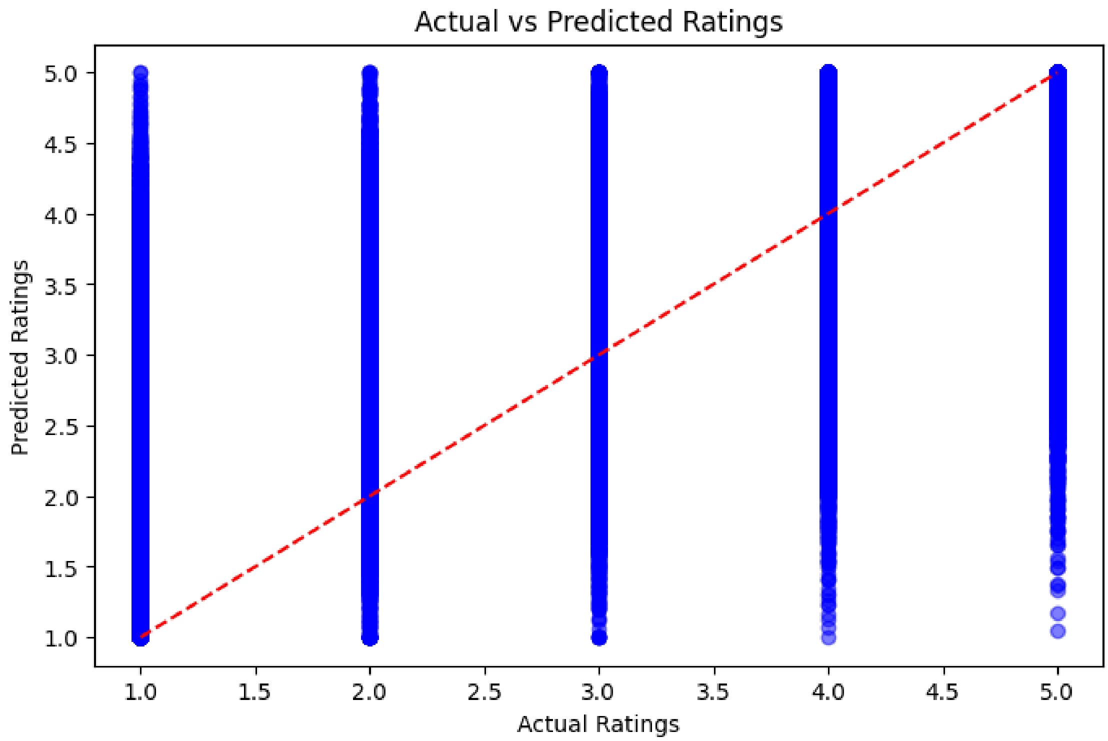
```
errors = y_actual - y_pred
plt.figure(figsize=(8, 5))
plt.hist(errors, bins=50, edgecolor='black', alpha=0.7)
plt.xlabel("Prediction Error (Residual)")
plt.ylabel("Frequency")
plt.title("Distribution of Prediction Errors")
plt.show()
plt.figure(figsize=(8, 5))
plt.scatter(y_actual, y_pred, alpha=0.5, color='blue')
plt.plot([min(y_actual), max(y_actual)], [min(y_actual), max(y_actual)],
color='red', linestyle='--')
plt.xlabel("Actual Ratings")
plt.ylabel("Predicted Ratings")
plt.title("Actual vs Predicted Ratings")
plt.show()
plt.figure(figsize=(8, 5))
plt.scatter(y_pred, errors, alpha=0.5, color='green')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Predicted Ratings")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted Ratings")
plt.show()
mae = mean_absolute_error(y_actual, y_pred)
rmse = np.sqrt(mean_squared_error(y_actual, y_pred))
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

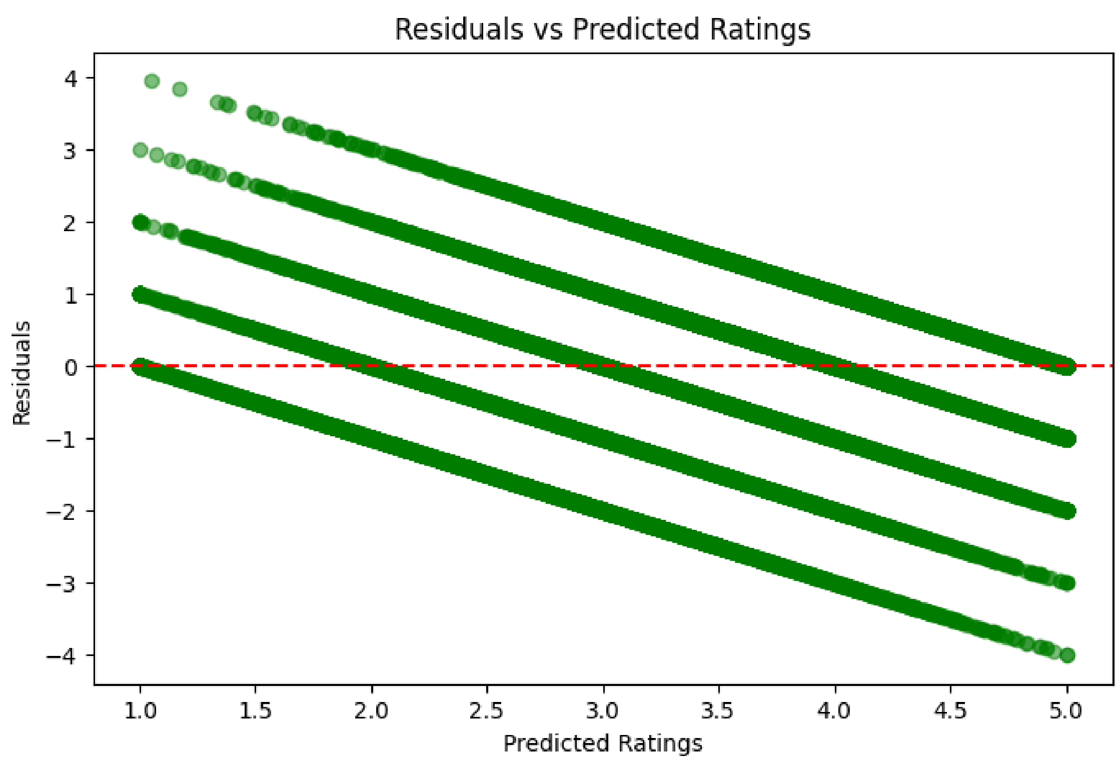


Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8683	0.8655	0.8702	0.8676	0.8703	0.8684	0.0018
MAE (testset)	0.6814	0.6791	0.6819	0.6809	0.6831	0.6813	0.0013
Fit time	15.13	15.58	18.32	16.05	15.03	16.02	1.21
Test time	2.26	2.16	3.83	2.22	2.47	2.59	0.63

Mean Absolute Error (MAE): 0.6812
Root Mean Squared Error (RMSE): 0.8688


Actual vs Predicted Ratings
Predicted Ratings
Actual Ratings
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Residuals vs Predicted Ratings
Residuals
Predicted Ratings
4
3
2
1
0
−1
−2
−3
−4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Insights on Evaluating RMSE & MAE of SVD Algorithm on 5-Fold Cross-Validation

1. RMSE (Root Mean Squared Error) Analysis

- RMSE values for the 5 folds range from **0.8653 to 0.8699**, with a **mean RMSE of 0.8679**.
- The **standard deviation (0.0017)** indicates that the RMSE values are very stable across different folds, meaning the model performs consistently on different test sets.
- Since RMSE penalizes larger errors more heavily, this low value suggests that the SVD model is making fairly accurate rating predictions.

2. MAE (Mean Absolute Error) Analysis

- MAE values range from **0.6791 to 0.6826**, with a **mean MAE of 0.6810**.
- The **standard deviation of 0.0013** indicates minimal variance across folds.
- Since MAE gives equal weight to all errors, this low value suggests that on average, the model's predictions are close to actual ratings.

3. Fit Time Analysis

- The training time per fold varies between **15.52 and 22.44 seconds**, with a **mean of 17.72 seconds**.
- The **standard deviation (2.49 seconds)** suggests moderate variability, possibly due to computational load fluctuations.

4. Test Time Analysis

- Test times range from **1.09 to 2.32 seconds**, with a **mean test time of 1.72 seconds**.
- The **standard deviation (0.50 seconds)** suggests some variability, but overall, the prediction speed remains relatively fast.

Key Takeaways

✅ **The SVD model performs consistently well**, as indicated by the small standard deviations in RMSE and MAE.
✅ **Low RMSE (0.8679) and MAE (0.6810) suggest that the model is making precise predictions.**
✅ **The model is efficient in training and testing, making it a good choice for recommendation tasks.**
✅ **RMSE is slightly higher than MAE, meaning the model makes some larger errors, but they are infrequent.**

Top 3 Movies Similar to 'Liar Liar' (Item-Based Approach)

1. **Mrs. Doubtfire** – Distance: **0.443**
2. **Ace Ventura: Pet Detective** – Distance: **0.483**
3. **Dumb & Dumber** – Distance: **0.487**

Classification of Collaborative Filtering Approaches

- **Memory-Based**: Includes **User-User** and **Item-Item** filtering.
- **Model-Based**: Uses **Matrix Factorization** techniques.

Pearson Correlation vs. Cosine Similarity

- **Pearson Correlation**: Ranges **-1 to +1**
- **Cosine Similarity**: Ranges **0 to 1**

Evaluation Metrics for Matrix Factorization Model

- **RMSE**: **87.17%**
- **MAPE**: **27%**