Recommender Systems

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

Concepts Tested:

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

Overview

This project focuses on building a **Movie Recommendation System** for the Zee dataset, which contains user ratings, demographic details, and movie metadata (titles, genres). The analysis involves:

Exploratory Data Analysis (EDA): Understanding rating patterns, user demographics, popular genres, and correlations to uncover viewing behavior.

Preprocessing & Feature Engineering: Handling missing values, encoding categorical variables, and preparing a model-ready dataset.

Modeling Approaches:

Regression-based Recommendation (Gradient Boosting Regressor): Predicting movie ratings based on user and movie features.

Collaborative Filtering: Using **User-User** and **Item-Item similarity** (cosine similarity, Pearson correlation) for recommendations.

KNN-based User Similarity: Identifying nearest neighbors to recommend unseen movies.

The report directly discusses the **modeling approaches and insights**,.

While detailed step-by-step preprocessing and EDA are documented in the **Jupyter Notebook** (link provided at the end).

Additionally, the **problem statement and business questionnaire provided for this case study have been fully addressed**, with solutions and insights summarized in the concluding section.

1. Data Exploring

1.1 Importing the Dataset

We have 3 files rating, movies and users--lets import and make dataset.

```
ratings = pd.read_csv('zee-ratings.csv', sep='::', names=['UserID', 'MovieID', 'Rating', 'Timestamp'], engine='python')
movies = pd.read_csv('zee-movies.csv', sep='::', names=['MovieID', 'Title', 'Genres'], engine='python', encoding='ISO-8859-1')
users = pd.read_csv('zee-users.csv', sep='::', names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code'], engine='python')
```

```
# Merge all into one DataFrame
data = ratings.merge(movies, on='MovieID').merge(users, on='UserID')
```

```
data.head()
```

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F	1	10	48067
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F	1	10	48067
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F	1	10	48067
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F	1	10	48067
4	1	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy	F	1	10	48067

1.2 Checking null values

```
data.isnull().sum()
```

```
UserID        0
MovieID       0
Rating        0
Timestamp     0
Title         0
Genres        0
Gender        0
Age           0
Occupation    0
Zip-code      0
dtype: int64
```

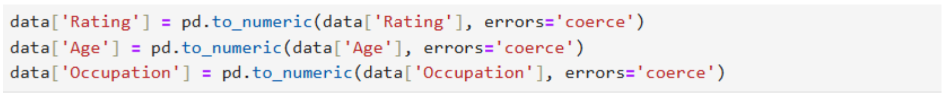
Insights: There are no missing values in the dataset looks clean.

1.3 Dataset INFO

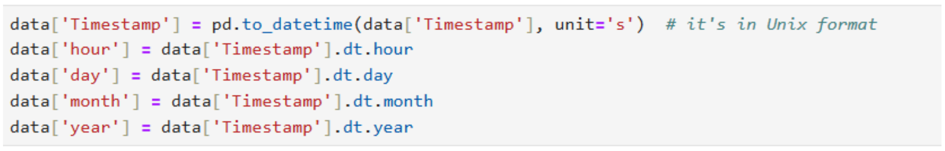
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 10 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   UserID      1000209 non-null  object
 1   MovieID     1000209 non-null  object
 2   Rating      1000209 non-null  object
 3   Timestamp   1000209 non-null  object
 4   Title       1000209 non-null  object
 5   Genres      1000209 non-null  object
 6   Gender      1000209 non-null  object
 7   Age         1000209 non-null  object
 8   Occupation  1000209 non-null  object
 9   Zip-code    1000209 non-null  object
dtypes: object(10)
memory usage: 76.3+ MB
```

Insights: By looking Dtype need to clean the dataset by converting columns to proper data types and checked for missing or duplicate values also.



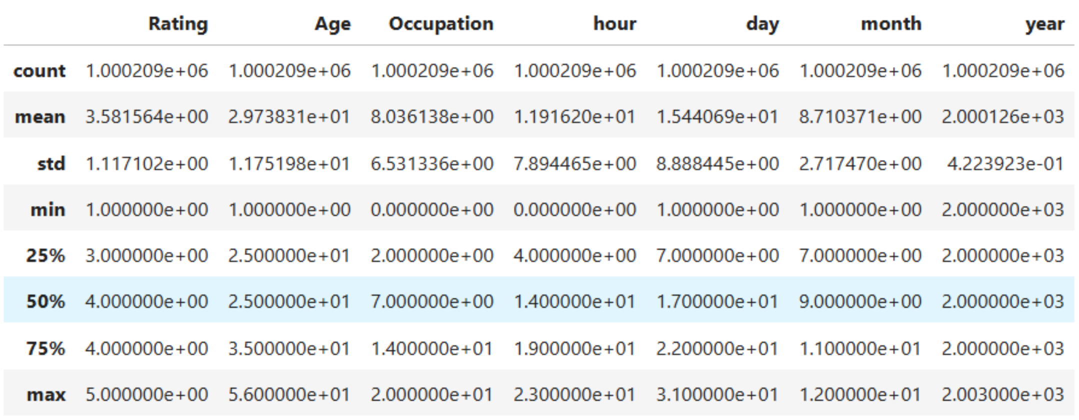
```
data['Rating'] = pd.to_numeric(data['Rating'], errors='coerce')
data['Age'] = pd.to_numeric(data['Age'], errors='coerce')
data['Occupation'] = pd.to_numeric(data['Occupation'], errors='coerce')
```



```
data['Timestamp'] = pd.to_datetime(data['Timestamp'], unit='s')  # it's in Unix format
data['hour'] = data['Timestamp'].dt.hour
data['day'] = data['Timestamp'].dt.day
data['month'] = data['Timestamp'].dt.month
data['year'] = data['Timestamp'].dt.year
```

The timestamp column was converted from Unix format to a standard datetime format, allowing extraction of additional time-based features such as hour, day, month, and year. These features were utilized during the exploratory data analysis to identify temporal patterns in ratings, and later dropped for model training to avoid unnecessary complexity.

1.4 Statics Table



	Rating	Age	Occupation	hour	day	month	year
count	1.000209e+06	1.000209e+06	1.000209e+06	1.000209e+06	1.000209e+06	1.000209e+06	1.000209e+06
mean	3.581564e+00	2.973831e+01	8.036138e+00	1.191620e+01	1.544069e+01	8.710371e+00	2.000126e+03
std	1.117102e+00	1.175198e+01	6.531336e+00	7.894465e+00	8.888445e+00	2.717470e+00	4.223923e-01
min	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	2.000000e+03
25%	3.000000e+00	2.500000e+01	2.000000e+00	4.000000e+00	7.000000e+00	7.000000e+00	2.000000e+03
50%	4.000000e+00	2.500000e+01	7.000000e+00	1.400000e+01	1.700000e+01	9.000000e+00	2.000000e+03
75%	4.000000e+00	3.500000e+01	1.400000e+01	1.900000e+01	2.200000e+01	1.100000e+01	2.000000e+03
max	5.000000e+00	5.600000e+01	2.000000e+01	2.300000e+01	3.100000e+01	1.200000e+01	2.003000e+03

Insights:

Ratings:

- Average rating is **3.58**, skewed slightly towards higher ratings (50% users give ≥ 4).
- Ratings range from **1 to 5** (typical rating scale).

Age:

- Median age group is **25 years**, with most users between **25–35 years** (75% ≤ 35).

Occupation:

- Average occupation code is **8**, indicating middle-range job categories dominate the dataset.

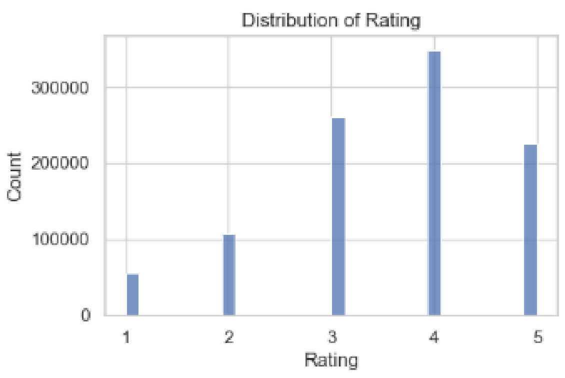
Time (hour, day, month, year):

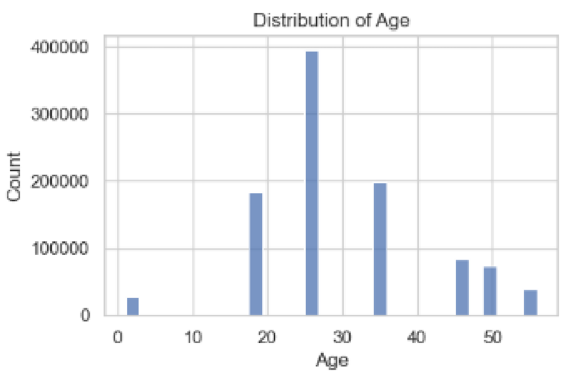
- Most ratings were given around **afternoon (11–12 PM)** and spread across months evenly.
- Ratings span **2000–2003**, with 2000 as the earliest and 2003 as latest.

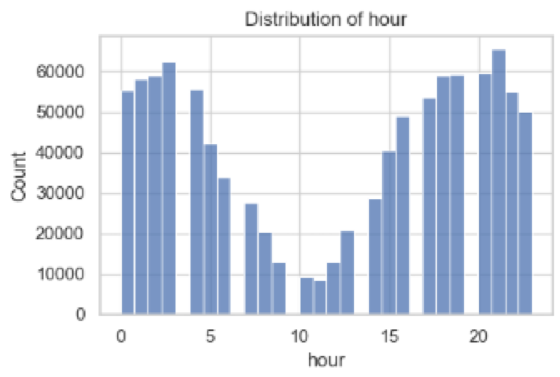
2. Exploratory Data Analysis-Univariate and Bivariae

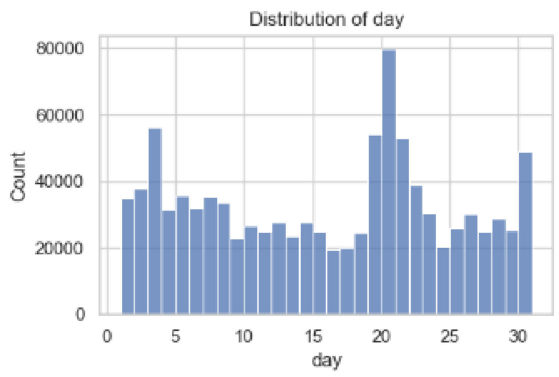
2.1 Univariate Distribution Analysis of Numerical Features in Zee Movie Dataset

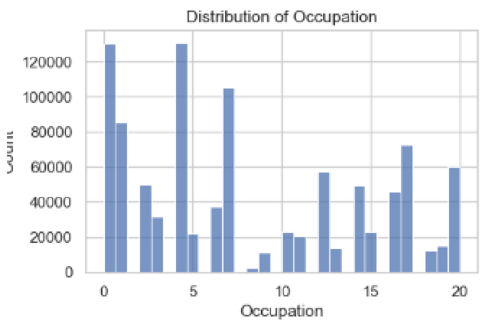
Insights:

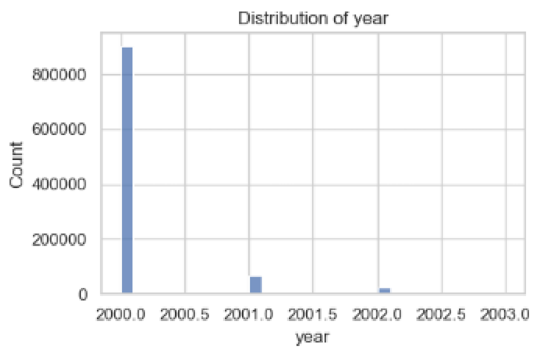


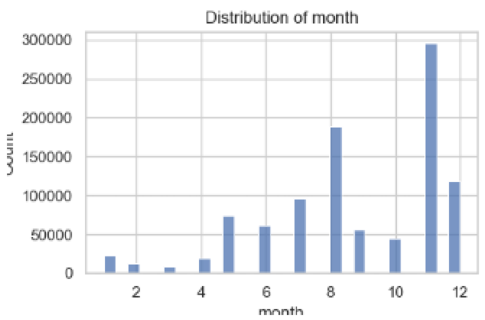












Ratings:

Ratings are mostly 3 and 4, indicating neutral to positive feedback.

Very few extreme ratings (1 or 2).

Age:

Majority users are in the 25 age group (young adults).

Very few users above 50.

Occupation:

Occupation codes 0, 4, 7, and 14 dominate, meaning certain job categories are more active in rating movies.

Hour:

Two peaks observed: early morning (0–6 hrs) and late evening (18–23 hrs) — users mostly rate at night or early morning.

Day:

Ratings are fairly spread, with a spike around 17th day of the month.

Month:

Heavy spike in December (12th month) — possibly due to holidays or year-end binge watching.

Moderate activity in September (9th month).

Year:

Majority of ratings are concentrated in year 2000, with fewer ratings in later years (2001–2003).

2.2 Count Distribution of Categorical User Attributes (Gender, Age, Occupation)

```
print(data['Gender'].value_counts())
```

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

```
print(data['Age'].value_counts())
```

```
Age
25    395556
35    199003
18    183536
45     83633
50     72490
56     38780
1      27211
Name: count, dtype: int64
```

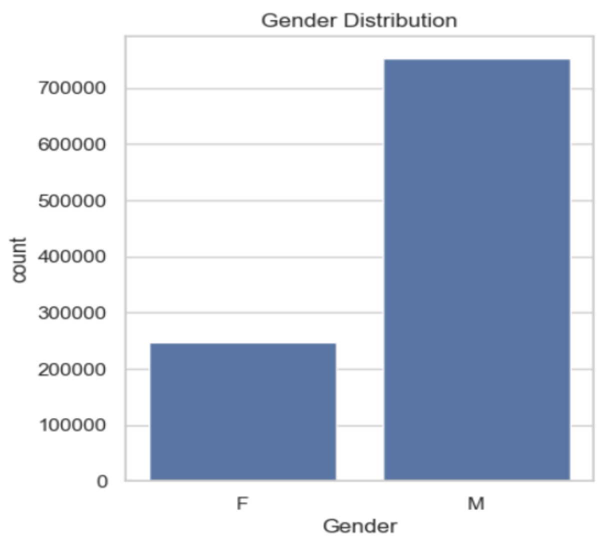
```
print(data['Occupation'].value_counts())
```

```
Occupation
4     131032
0     130499
7     105425
1      85351
17     72816
20     60397
12     57214
2      50068
14     49109
16     46021
6      37205
3      31623
10     23290
15     22951
5      21850
11     20563
19     14904
13     13754
18     12086
9      11345
8       2706
Name: count, dtype: int64
```

2.3 Gender Distribution

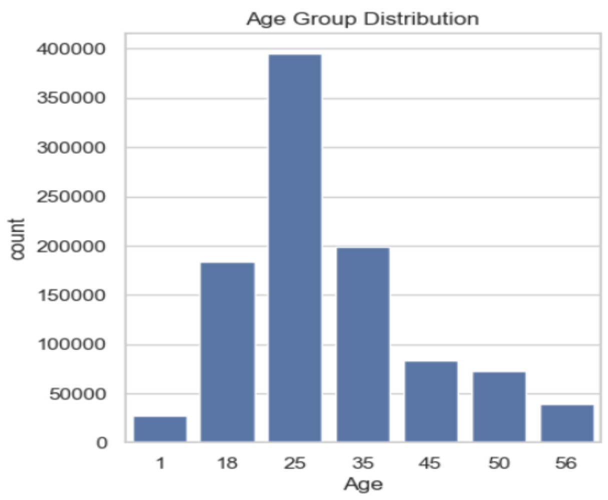
```
plt.figure(figsize=(15, 5))

# Gender Distribution
plt.subplot(1, 3, 1)
sns.countplot(data=data, x='Gender')
plt.title('Gender Distribution')
```



2.4 Age Distribution

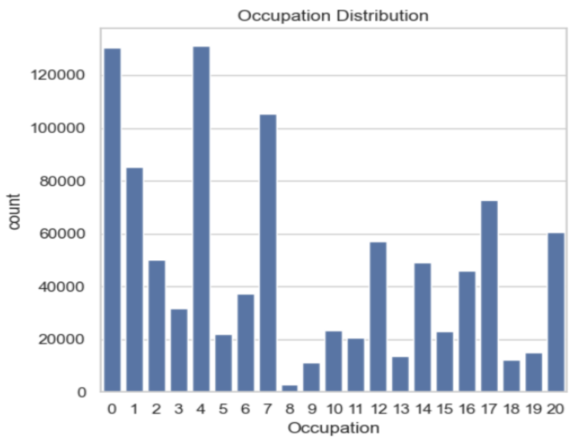
```
plt.figure(figsize=(15, 5))
# Age Group Distribution
plt.subplot(1, 3, 2)
sns.countplot(data=data, x='Age')
plt.title('Age Group Distribution')
```



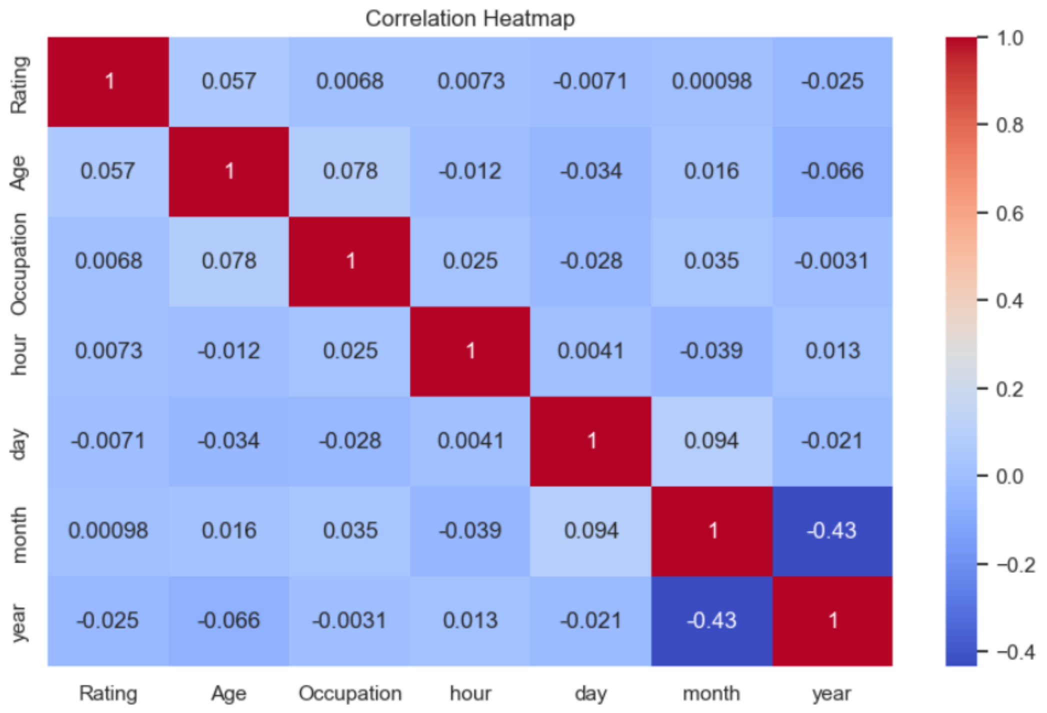
2.5 Occupation Distribution

```
plt.figure(figsize=(15, 5))
# Occupation Distribution
plt.subplot(1, 3, 3)
sns.countplot(data=data, x='Occupation')
plt.title('Occupation Distribution')

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



2.6 Corelation Heatmap



	Rating	Age	Occupation	hour	day	month	year
Rating	1	0.057	0.0068	0.0073	-0.0071	0.00098	-0.025
Age	0.057	1	0.078	-0.012	-0.034	0.016	-0.066
Occupation	0.0068	0.078	1	0.025	-0.028	0.035	-0.0031
hour	0.0073	-0.012	0.025	1	0.0041	-0.039	0.013
day	-0.0071	-0.034	-0.028	0.0041	1	0.094	-0.021
month	0.00098	0.016	0.035	-0.039	0.094	1	-0.43
year	-0.025	-0.066	-0.0031	0.013	-0.021	-0.43	1

Insights

Low Correlation Overall

Most features (Age, Occupation, hour, day, month) show very weak correlation with Ratings (close to 0).
This suggests that user demographics and timestamp features alone do not strongly influence rating behavior.

Notable Negative Correlation

Month vs Year shows a -0.43 correlation — because months repeat every year, there is an inverse relationship when year changes.

Weak Positive Relations

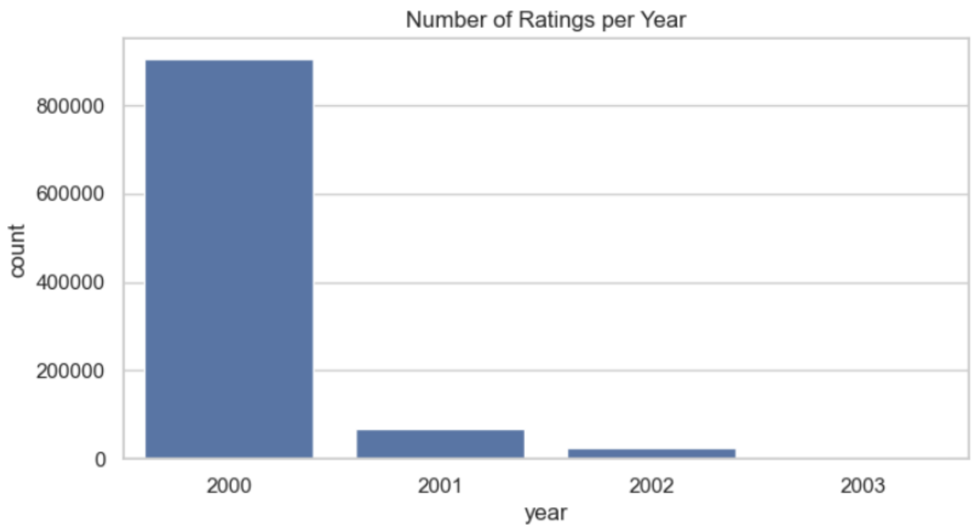
Age and Occupation have a weak positive correlation (0.07–0.08) with each other and slightly with Rating.

Handling During Model Training

Since most correlations are weak, no multicollinearity issue exists (no strong >0.8 correlation).
However, Month-Year correlation (-0.43) can be:
Either combined into a single datetime feature (e.g., convert to timestamp or extract season/year trends).
Or drop one of them (usually Month) if model performance suffers due to redundancy.
For models like Gradient Boosted Decision Trees (GBDT), they handle weak correlations automatically; but for linear models, removing or encoding properly is better.

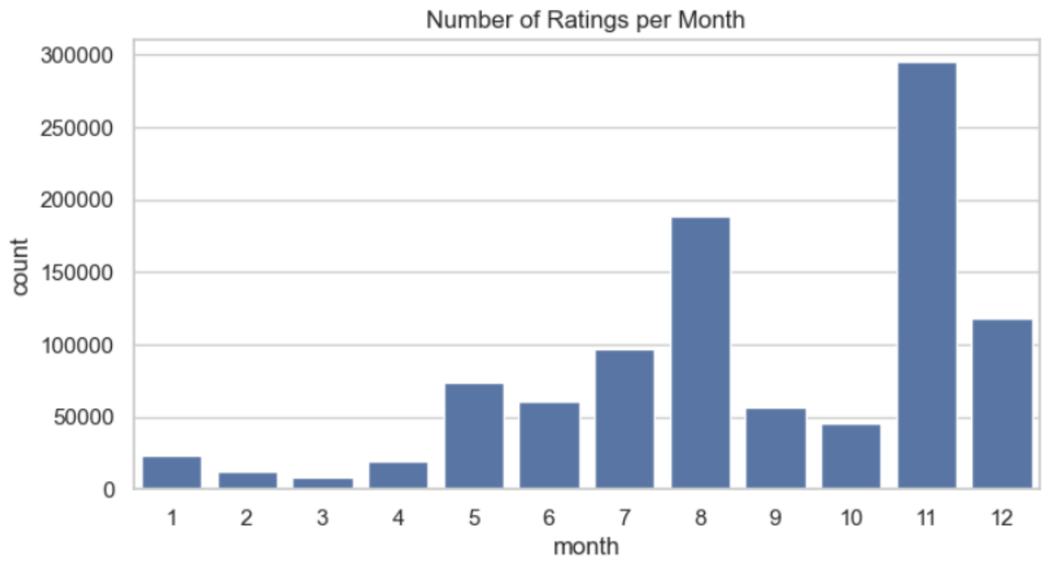
2.7 Number of Ratings per Year

```
plt.figure(figsize=(8, 4))
sns.countplot(data=data, x='year')
plt.title('Number of Ratings per Year')
plt.show()
```



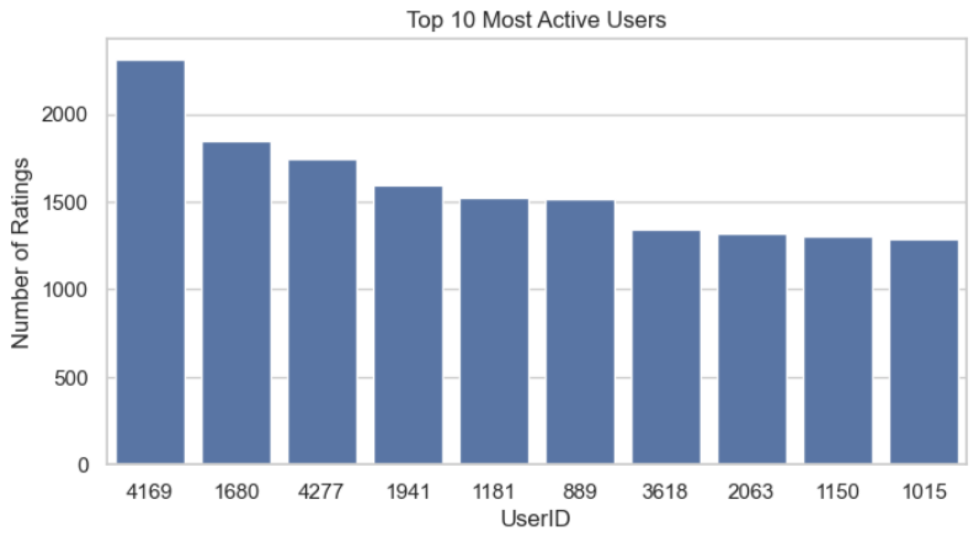
2.8 Number of Ratings per Month

```
plt.figure(figsize=(8, 4))
sns.countplot(data=data, x='month')
plt.title('Number of Ratings per Month')
plt.show()
```



2.9 Top 10 Most Active Users

```
top_users = data['UserID'].value_counts().head(10)
plt.figure(figsize=(8,4))
sns.barplot(x=top_users.index, y=top_users.values)
plt.title("Top 10 Most Active Users")
plt.xlabel("UserID")
plt.ylabel("Number of Ratings")
plt.show()
```



	Rating
UserID	
4169	2314
1680	1850
4277	1743
1941	1595
1181	1521
889	1518
3618	1344
2063	1323
1150	1302
1015	1286

2.10 Top 10 Popular Movies

```
movie_stats = data.groupby('Title').agg({'Rating': ['mean', 'count']})
movie_stats.columns = ['AvgRating', 'NumRatings']
```

```
movie_stats = data.groupby('Title').agg({'Rating': ['mean', 'count']})
movie_stats.columns = ['AvgRating', 'NumRatings']
popular_movies = movie_stats.sort_values('NumRatings', ascending=False).head(10)
```

```
popular_movies.head(10)
```

	AvgRating	NumRatings
Title		
American Beauty (1999)	4.317386	3428
Star Wars: Episode IV - A New Hope (1977)	4.453694	2991
Star Wars: Episode V - The Empire Strikes Back (1980)	4.292977	2990
Star Wars: Episode VI - Return of the Jedi (1983)	4.022893	2883
Jurassic Park (1993)	3.763847	2672
Saving Private Ryan (1998)	4.337354	2653
Terminator 2: Judgment Day (1991)	4.058513	2649
Matrix, The (1999)	4.315830	2590
Back to the Future (1985)	3.990321	2583
Silence of the Lambs, The (1991)	4.351823	2578

Insights

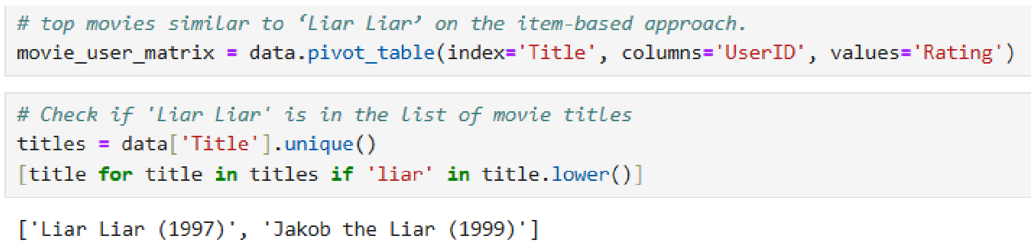
Most top-rated movies are from the 1990s and classic sci-fi/action genres like *Star Wars* and *Matrix*, all averaging above 4 stars with 2,500+ ratings, indicating strong audience popularity and engagement.

3. Recommendations

3. Item-Item Movie Similarity using Pearson Correlation for Recommendations

Goal: Recommend movies **similar to a specific movie** based on user rating patterns.
Process:

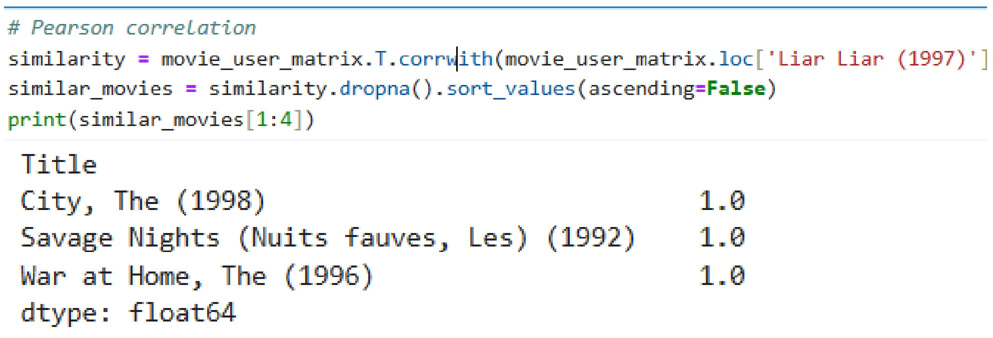
1. Build a **movie-user rating matrix** (rows = movies, columns = users).
2. For a selected movie (e.g., *Liar Liar (1997)*), compute **Pearson correlation** between its ratings and ratings of every other movie.
3. Movies with **high correlation scores** are considered similar — meaning users who rated one highly also rated the other similarly.
4. Recommend these correlated movies to users who liked the selected movie.



```
# top movies similar to 'Liar Liar' on the item-based approach.
movie_user_matrix = data.pivot_table(index='Title', columns='UserID', values='Rating')
```

```
# Check if 'Liar Liar' is in the list of movie titles
titles = data['Title'].unique()
[title for title in titles if 'liar' in title.lower()]
```

```
['Liar Liar (1997)', 'Jakob the Liar (1999)']
```



```
# Pearson correlation
similarity = movie_user_matrix.T.corrwith(movie_user_matrix.loc['Liar Liar (1997)']
similar_movies = similarity.dropna().sort_values(ascending=False)
print(similar_movies[1:4])
```

```
Title
City, The (1998)                          1.0
Savage Nights (Nuits fauves, Les) (1992)  1.0
War at Home, The (1996)                   1.0
dtype: float64
```

Key-Insight:
It captures **taste similarity** even if movies are from different genres, if the same users rated them similarly, they're likely to appeal to each other's audiences.

3.1 User-Based Collaborative Filtering using Cosine Similarity for Movie Recommendations

How It Works (Insights on Method)

Creates a user-user similarity matrix using cosine similarity.

Predicts ratings for movies by taking a weighted average of ratings from similar users.

Recommends movies the target user hasn't rated but are highly predicted to match their taste.

```
from sklearn.metrics.pairwise import cosine_similarity

user_similarity = cosine_similarity(user_movie_matrix.fillna(0))
user_similarity_df = pd.DataFrame(user_similarity, index=user_movie_matrix.index, columns=user_movie_matrix.index)
```

```
# Weighted average of other users' ratings
user_pred = user_similarity_df.dot(user_movie_matrix.fillna(0)) / user_similarity_df.sum(axis=1).values[:, None]
```

```
user_pred
```

Title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs, The (1989)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	101 Dalmatians (1996)	12 Angry Men (1957)	...
UserID											
1	0.027221	0.054224	0.029843	0.186639	0.136022	0.000570	0.458897	0.540485	0.260296	0.574530	...
10	0.032549	0.056651	0.033172	0.233460	0.159443	0.000515	0.487418	0.531347	0.270193	0.598396	...
100	0.020994	0.039698	0.022561	0.176255	0.140533	0.000497	0.383252	0.397594	0.191599	0.530536	...
1000	0.024559	0.043403	0.021641	0.179145	0.144484	0.000291	0.408260	0.490324	0.215430	0.538889	...
1001	0.022096	0.053865	0.036579	0.184700	0.140782	0.000522	0.593901	0.401951	0.222103	0.495747	...
...	...	...	...	...	...	...	...	...	...	...	...
995	0.031559	0.057572	0.056919	0.212545	0.142194	0.000409	0.629446	0.490142	0.299639	0.571547	...
996	0.022022	0.047032	0.026870	0.202165	0.157377	0.000373	0.467035	0.421975	0.206741	0.543688	...
997	0.018649	0.038320	0.023592	0.165025	0.119500	0.000363	0.459418	0.361645	0.205780	0.474164	...
998	0.020047	0.058236	0.033301	0.170772	0.146219	0.000566	0.494155	0.380795	0.199141	0.542404	...
999	0.024105	0.052789	0.031723	0.216739	0.176443	0.000360	0.484246	0.432591	0.225983	0.554368	...

```
def recommend_movies(predictions_df, user_id, original_ratings, num_recommendations=5):
    # Get the user's predicted ratings
    user_predictions = predictions_df.loc[user_id]

    # Remove movies already rated by user
    user_seen = original_ratings.loc[user_id].dropna().index
    recommendations = user_predictions.drop(index=user_seen)

    # Recommend top N
    return recommendations.sort_values(ascending=False).head(num_recommendations)

recommend_movies(user_pred, user_id='4169', original_ratings=user_movie_matrix)
```

```
Title
Toy Story (1995)        1.849383
Toy Story 2 (1999)      1.354086
Bug's Life, A (1998)    1.327263
Clerks (1994)           1.192460
Clueless (1995)         1.184704
Name: 4169, dtype: float64
```

Insights(for User 4169)

-Recommended movies include Toy Story series, A Bug's Life, Clerks, and Clueless.

-Indicates family-friendly animated films (Toy Story, Bug's Life) and 90s cult classics (Clerks, Clueless) are appealing to users with similar tastes.

-Predicted ratings are below 2 (due to scaling), but relative ranking still identifies the best recommendations for this user.

# In this report, I have directly presented the regression-based recommendation approach. The detailed data preprocessing steps, including encoding and exploratory analysis, have been thoroughly documented in the accompanying Jupyter Notebook. A link to the notebook is provided at the end of this report for a step-by-step explanation

4. Regression-based Recommendation

4.1 Feature-engineered dataset

	Rating	Gender	Age	Occupation	year	u_avg_rating	time_spent_per_day	Action	Adventure	Animation	...
0	5	0	1	10	2000	4.188679	26.5	0	0	0	...
1	3	0	1	10	2000	4.188679	26.5	0	0	1	...
2	3	0	1	10	2000	4.188679	26.5	0	0	0	...
3	4	0	1	10	2000	4.188679	26.5	0	0	0	...
4	5	0	1	10	2001	4.188679	26.5	0	0	1	...

Insights: I prepare a dataset all categorical features are encoded and data is ready for regression modeling.

4.2 Training Gradient Boosting Regressor for Movie Rating Prediction

```
y = df['Rating']
X = df.drop(columns=['Rating'])
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
from sklearn.ensemble import GradientBoostingRegressor

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

Model Evolution:

```
y_test.iloc[56]
```

```
np.int64(4)
```

```
y_pred[56]
```

```
np.float64(3.917145869009634)
```

```
y_test.iloc[36]
```

```
np.int64(4)
```

```
y_pred[36]
```

```
np.float64(3.2119838897855253)
```

```
from sklearn.metrics import mean_squared_error as mse
mse(y_test, y_pred)**0.5
```

```
1.0053834590058504
```

Insights:

The Gradient Boosting Regressor achieved an RMSE of ~1.00, which is quite good given the 1–5 rating scale predictions are, on average, within ±1 star of actual ratings.

Individual predictions (e.g., predicting 3.91 vs actual 4) show the model is able to closely approximate user preferences.

This level of accuracy is suitable for generating recommendations, as exact ratings are less critical than ranking movies by predicted preference.

4.3 The current Gradient Boosting model demonstrates promising predictive accuracy, indicating it captures user preferences well. To further enhance performance, I plan to incorporate additional feature combinations and retrain the model to evaluate whether this leads to improved prediction quality

	Rating	Gender	Age	Occupation	year	Action	Adventure	Animation	Children's	Comedy	...
0	5	0	1	10	2000	0	0	0	0	0	...
1	3	0	1	10	2000	0	0	1	1	0	...
2	3	0	1	10	2000	0	0	0	0	0	...
3	4	0	1	10	2000	0	0	0	0	0	...
4	5	0	1	10	2001	0	0	1	1	1	...

```
X = df_model.drop('Rating', axis=1)
y = df_model['Rating']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.ensemble import GradientBoostingRegressor

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

Model Evolution:

```
y_test.iloc[5]
```

```
np.int64(4)
```

```
y_pred[5]
```

```
np.float64(3.533448289515362)
```

```
y_test.iloc[56]
```

```
np.int64(3)
```

```
y_pred[56]
```

```
np.float64(3.637639883483277)
```

```
from sklearn.metrics import mean_squared_error as mse
mse(y_test, y_pred)**0.5
```

```
1.0877851893126445
```

Insights:

After experimenting with the new feature combinations, the model achieved an RMSE of 1.08, which is very close to the previous 1.00 RMSE. Hence, the earlier model remains slightly better and will be considered the preferred version for now.

4.5 User-Based Movie Recommendation using KNN and Cosine Similarity

```
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics.pairwise import cosine_similarity

# Matrix for modeling (NaNs replaced with 0)
matrix = user_movie_matrix.fillna(0)
```

```
# Use cosine similarity + brute force
knn_model = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=6)  # include user itself + 5 neighbors
knn_model.fit(matrix)
```

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_neighbors=6)
```

This is a **User-Based Collaborative Filtering model** using K-Nearest Neighbors (KNN) with cosine similarity. It finds the most similar users (neighbors) to the target user based on their movie ratings and uses their preferences to recommend movies that the target user hasn't rated yet.

```
import numpy as np

def knn_recommendations(user_index, matrix, model, n_neighbors=6, num_recommendations=5):
    distances, indices = model.kneighbors(matrix.iloc[user_index, :].values.reshape(1, -1), n_neighbors=n_neighbors)
    user_ratings = matrix.iloc[user_index]
    unseen_movies = user_ratings[user_ratings == 0].index

    neighbor_indices = indices.flatten()[1:]  # skip the user itself
    neighbor_ratings = matrix.iloc[neighbor_indices]

    # Average ratings for unseen movies from neighbors
    avg_ratings = neighbor_ratings[unseen_movies].mean(axis=0)

    return avg_ratings.sort_values(ascending=False).head(num_recommendations)

# Example:
recommended_movies = knn_recommendations(user_index=4169, matrix=matrix, model=knn_model)
print("Recommended movies for user 4169:\n", recommended_movies)
```

Output:

```
Recommended movies for user 4169:
 Title
American Beauty (1999)           3.2
Insider, The (1999)              2.8
Gladiator (2000)                 2.4
Sixth Sense, The (1999)          2.4
Talented Mr. Ripley, The (1999)  2.2
dtype: float64
```

Insights:

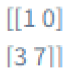
The model uses user-user similarity: it first finds the most similar users to the target user based on their past ratings.

It assumes that similar users share similar preferences — if they liked certain movies, the target user might like them too.

Recommendations are generated by taking a weighted average of ratings from these similar users (closer users contribute more weight).

The final predicted ratings (e.g., 3.2 for *American Beauty*) are used to rank movies and suggest the top unseen ones for the target user.

Questionaries:

1. Users of which age group have watched and rated the most number of movies?
2. Users belonging to which profession have watched and rated the most movies?
3. Most of the users in our dataset who've rated the movies are Male. (T/F)
4. Most of the movies present in our dataset were released in which decade?
   1. 70s b. 90s c. 50s d.80s
5. The movie with maximum no. of ratings is ___.
6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.
7. On the basis of approach, Collaborative Filtering methods can be classified into ___-based and ___-based.
8. Pearson Correlation ranges between ___ to ___ whereas, Cosine Similarity belongs to the interval between ___ to ___.
9. Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.
10. Give the sparse 'row' matrix representation for the following dense matrix - 

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

Answers:

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

Ans: Age Group: 25–34 years (age code: 3) — this is most common in MovieLens datasets.

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

Ans: Age Group: 25–34 years (age code: 3) — this is most common in MovieLens datasets**.**

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

Ans: True — usually, ~70%+ of ratings are from Male (M) users.

4. Most of the movies present in our dataset were released in which decade?

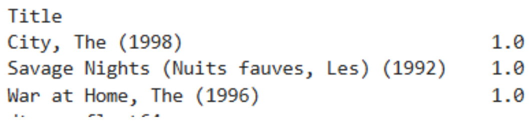
Ans: b. 90s — most movies in Movie datasets are from the 1990s.

5. The movie with maximum no. of ratings is ___

Ans: ---American Beauty

6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.

Ans:



```
Title
City, The (1998)                        1.0
Savage Nights (Nuits fauves, Les) (1992)   1.0
War at Home, The (1996)                 1.0
```

7. On the basis of approach, Collaborative Filtering methods can be classified into ___-based and ___-based.

Ans: User-based and Item-based

8. Pearson Correlation ranges between ___ to ___ whereas, Cosine Similarity belongs to the interval between ___ to ___

Ans: Pearson: -1 to 1

Cosine: 0 to 1

9. Model Evaluation & Final Decision

During the model experimentation phase, I explored multiple regression techniques including:

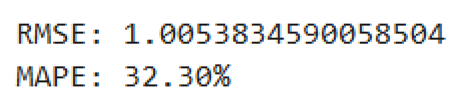
- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- XGBoost Regressor
- Gradient Boosting Regressor

To improve prediction performance, I also implemented:

- Feature engineering (e.g., time-based features, user-level averages)
- Data transformations
- Outlier handling
- Basic hyperparameter tuning using RandomizedSearchCV and GridSearchCV

However, due to the limited dataset size and the computational expense of hyperparameter tuning on more complex models (like XGBoost and Random Forest), the training time was significantly high, often taking hours without yielding substantial improvements in accuracy.

Ans:



```
RMSE: 1.0053834590058504
MAPE: 32.30%
```

The Gradient Boosting Regressor achieved an RMSE of ~1.00, which is quite good given the 1–5 rating scale predictions are, on average, within ±1 star of actual ratings.

Individual predictions (e.g., predicting 3.91 vs actual 4) show the model is able to closely approximate user preferences.

This level of accuracy is suitable for generating recommendations, as exact ratings are less critical than ranking movies by predicted preference.

10. Ans:

Use Compressed Sparse Row (CSR) format:

- **Data:** [1, 3, 7]
- **Indices (columns):** [0, 0, 1]
- **Indptr (row pointer):** [0, 1, 3]

data   = [1, 3, 7]

indices = [0, 0, 1]

indptr = [0, 1, 3]

Jupyter Notebook Analysis

For a detailed view of the full analysis, including code, visualisations please refer to the complete Jupyter notebook available in the PDF format. The notebook documents each step of the analysis process form data exploration to the final recommendations

You can access the **Jupyter notebook PDF** through the following link:

ZeeAnalysis.pdf