Project Title	**Movie Recommender System**
Tools	**Google Colab**
Domain	**Data Analyst**
Project Difficulties Level	**intermediate**

Dataset: The dataset is available at the given link.

Click here to download data set

Movies Recommender System

About Dataset: The dataset contains metadata for 45,000 movies from the Full MovieLens Dataset, covering films released up to July 2017. It includes cast, crew, plot keywords, budget, revenue, release dates, languages, production companies, and TMDB vote statistics. Additionally, it features 26 million ratings from 270,000 users, with ratings ranging from 1 to 5. The data is sourced from the official GroupLens website.

movies_metadata.csv: This is the main file with information about 45,000 movies from the Full MovieLens dataset. It includes details like movie posters, background images, budget, earnings, release dates, languages, and the countries and companies that made the movies.

1. Importing Libraries: Essential libraries are imported for data handling (pandas, numpy) and visualization (seaborn, matplotlib).

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
```

2. Uploading Dataset: Allows you to upload CSV files from your local machine to the Colab environment for analysis.

```
from google.colab import files

uploaded = files.upload()
```

3. Reading the Datasets: Loads the movies metadata and ratings data into Pandas DataFrames for further processing.

```
movies = pd.read_csv('movies_metadata.csv', low_memory=False)

ratings = pd.read_csv('ratings_small.csv')
```

4. Displaying Dataset Info: Provides column-wise info such as data types and non-null counts to understand the structure of the datasets.

```
print("Movies Dataset Info:")

print(movies.info())
```

```
Movies Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 45466 entries, 0 to 45465

Data columns (total 24 columns):

 #   Column                 Non-Null Count  Dtype

---  ------                 --------------  -----

 0   adult                  45466 non-null  object

 1   belongs_to_collection  4494 non-null   object

 2   budget                 45466 non-null  object

 3   genres                 45466 non-null  object
```

```
 4   homepage              7782 non-null   object
 5   id                    45466 non-null  object
 6   imdb_id               45449 non-null  object
 7   original_language     45455 non-null  object
 8   original_title        45466 non-null  object
 9   overview              44512 non-null  object
 10  popularity            45461 non-null  object
 11  poster_path           45080 non-null  object
 12  production_companies  45463 non-null  object
 13  production_countries  45463 non-null  object
 14  release_date          45379 non-null  object
 15  revenue               45460 non-null  float64
 16  runtime               45203 non-null  float64
 17  spoken_languages      45460 non-null  object
 18  status                45379 non-null  object
 19  tagline               20412 non-null  object
 20  title                 45460 non-null  object
 21  video                 45460 non-null  object
 22  vote_average          45460 non-null  float64
 23  vote_count            45460 non-null  float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
None
```

print("\nRatings Dataset Info:")

print(ratings.info())

```
Ratings Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100004 entries, 0 to 100003
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   userId     100004 non-null  int64
 1   movieId    100004 non-null  int64
 2   rating     100004 non-null  float64
 3   timestamp  100004 non-null  int64
dtypes: float64(1), int64(3)
```

memory usage: 3.1 MB

None

5. Checking for Missing Values: Identifies columns with missing values to inform any necessary data cleaning steps.

```
print("\nMissing Values in Movies Dataset:")
print(movies.isnull().sum())
```

Missing Values in Movies Dataset:

adult	0
belongs_to_collection	40972
budget	0
genres	0
homepage	37684
id	0
imdb_id	17
original_language	11
original_title	0
overview	954
popularity	5
poster_path	386
production_companies	3
production_countries	3
release_date	87
revenue	6
runtime	263
spoken_languages	6
status	87
tagline	25054
title	6
video	6
vote_average	6
vote_count	6

dtype: int64

```
print("\nMissing Values in Ratings Dataset:")
```

print(ratings.isnull().sum())

Missing Values in Ratings Dataset:

userId 0

movieId 0

rating 0

timestamp 0

dtype: int64

6. Previewing the Datasets: Displays the first few rows of each dataset to get an initial look at the data.

print("\nMovies Dataset Preview:")

print(movies.head())

Movies Dataset Preview:

adult belongs_to_collection budget \

0 False {'id': 10194, 'name': 'Toy Story Collection', ... 30000000

1 False NaN 65000000

2 False {'id': 119050, 'name': 'Grumpy Old Men Collect... 0

3 False NaN 16000000

4 False {'id': 96871, 'name': 'Father of the Bride Col... 0

genres \

0 [{'id': 16, 'name': 'Animation'}, {'id': 35, '...

1 [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...

2 [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...

3 [{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...

4 [{'id': 35, 'name': 'Comedy'}]

homepage id imdb_id original_language \

0 http://toystory.disney.com/toy-story 862 tt0114709 en

1 NaN 8844 tt0113497 en

2 NaN 15602 tt0113228 en

3 NaN 31357 tt0114885 en

4 NaN 11862 tt0113041 en

original_title \

0 Toy Story

1 Jumanji

2 Grumpier Old Men

3 Waiting to Exhale

4 Father of the Bride Part II

overview ... release_date \

0 Led by Woody, Andy's toys live happily in his 1995-10-30

1 When siblings Judy and Peter discover an encha... ... 1995-12-15

2 A family wedding reignites the ancient feud be... ... 1995-12-22

3 Cheated on, mistreated and stepped on, the wom... ... 1995-12-22

4 Just when George Banks has recovered from his 1995-02-10

revenue runtime spoken_languages \

0 373554033.0 81.0 [{'iso_639_1': 'en', 'name': 'English'}]

1 262797249.0 104.0 [{'iso_639_1': 'en', 'name': 'English'}, {'iso...

2 0.0 101.0 [{'iso_639_1': 'en', 'name': 'English'}]

3 81452156.0 127.0 [{'iso_639_1': 'en', 'name': 'English'}]

4 76578911.0 106.0 [{'iso_639_1': 'en', 'name': 'English'}]

status tagline \

0 Released NaN

1 Released Roll the dice and unleash the excitement!

2 Released Still Yelling. Still Fighting. Still Ready for...

3 Released Friends are the people who let you be yourself...

4 Released Just When His World Is Back To Normal... He's ...

title video vote_average vote_count

0 Toy Story False 7.7 5415.0

1 Jumanji False 6.9 2413.0

2 Grumpier Old Men False 6.5 92.0

3 Waiting to Exhale False 6.1 34.0

4 Father of the Bride Part II False 5.7 173.0

[5 rows x 24 columns]

print("\nRatings Dataset Preview:")

print(ratings.head())

Ratings Dataset Preview:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

7. Converting Budget and Revenue to Numeric: Converts the budget and revenue fields to numeric format while handling non-numeric values.

```
movies['budget'] = pd.to_numeric(movies['budget'], errors='coerce')
movies['revenue'] = pd.to_numeric(movies['revenue'], errors='coerce')
```

8. Summary Statistics for Ratings: Provides summary statistics (mean, min, max, etc.) for movie ratings to understand rating trends.

```
print("\nRatings Dataset Statistics:")
print(ratings['rating'].describe())
```

Ratings Dataset Statistics:

count	100004.000000
mean	3.543608
std	1.058064
min	0.500000
25%	3.000000
50%	4.000000
75%	4.000000
max	5.000000

Name: rating, dtype: float64

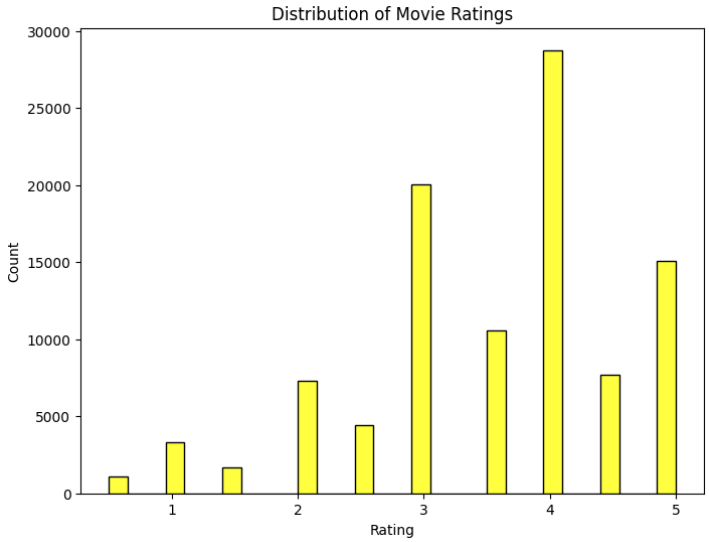
9. Plotting Rating Distribution: Visualizes how user ratings are distributed across different movies.

```
plt.figure(figsize=(8, 6))
sns.histplot(ratings['rating'], bins=30, kde=False, color='yellow')
```

plt.title('Distribution of Movie Ratings')

plt.xlabel('Rating')

plt.ylabel('Count')



10. Plotting Number of Ratings per Movie: Shows how frequently different movies have been rated to identify popular and unpopular titles.

ratings_per_movie = ratings.groupby('movieId').size()

plt.figure(figsize=(8, 6))

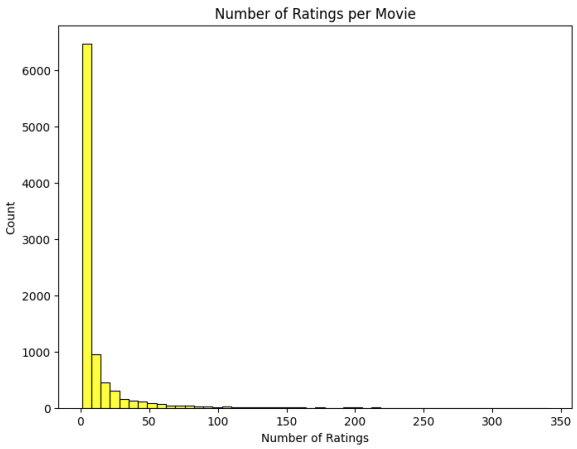
sns.histplot(ratings_per_movie, bins=50, kde=False, color='yellow')

plt.title('Number of Ratings per Movie')

plt.xlabel('Number of Ratings')

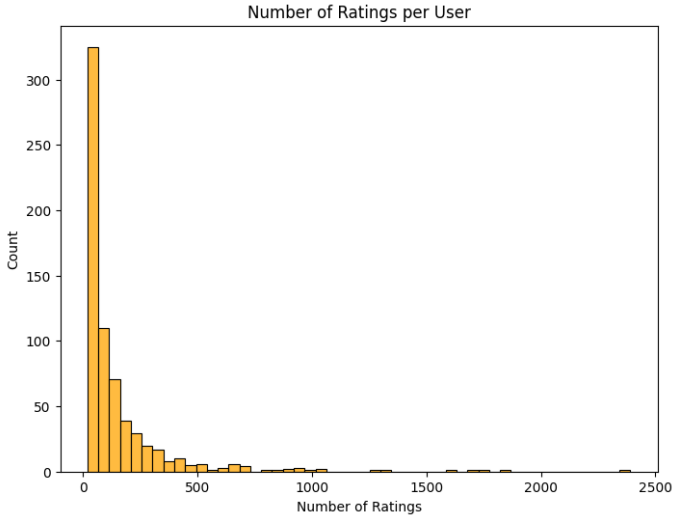
plt.ylabel('Count')

plt.show()



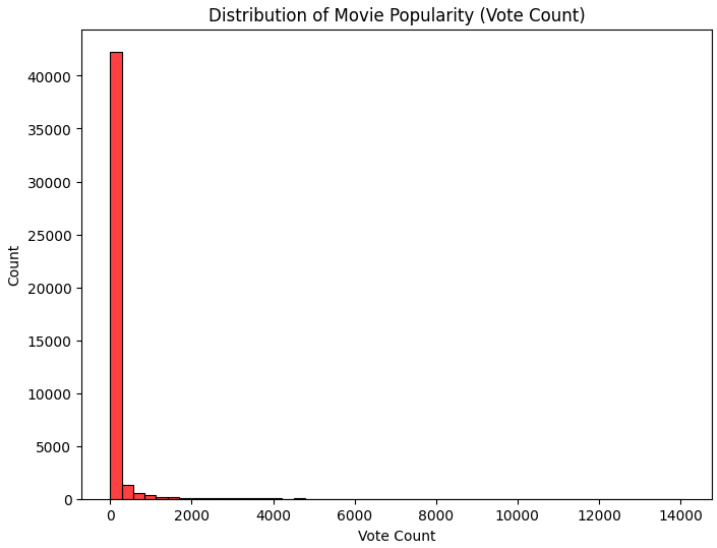
11. Plotting Number of Ratings per User: Analyzes how many ratings each user has given to assess user activity.

```
ratings_per_user = ratings.groupby('userId').size()

plt.figure(figsize=(8, 6))

sns.histplot(ratings_per_user, bins=50, kde=False, color='orange')

plt.title('Number of Ratings per User')

plt.xlabel('Number of Ratings')

plt.ylabel('Count')

plt.show()
```



12. Vote Count Distribution: Visualizes the popularity of movies based on the number of votes received.

```
movies['vote_count'] = pd.to_numeric(movies['vote_count'], errors='coerce')

plt.figure(figsize=(8, 6))

sns.histplot(movies['vote_count'].dropna(), bins=50, kde=False, color='red')

plt.title('Distribution of Movie Popularity (Vote Count)')

plt.xlabel('Vote Count')

plt.ylabel('Count')

plt.show()
```



13. Installing Surprise Library: Installs the Surprise library for building and evaluating recommendation systems.

```
!pip install surprise
```

14. Loading and Preparing Data for Surprise: Converts the ratings DataFrame into a format compatible with Surprise for model training.

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

# Load the ratings dataset into Surprise's format
reader = Reader(rating_scale=(0.5, 5))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

15. Creating and Evaluating SVD Model: Builds a matrix factorization model using SVD and evaluates its performance using cross-validation.

```
svd = SVD()

# Evaluate the model using cross-validation
print("\nCross-Validation Results:")
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Cross-Validation Results:

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

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8942	0.8946	0.8943	0.8958	0.9018	0.8962	0.0029
MAE (testset)	0.6898	0.6876	0.6890	0.6892	0.6944	0.6900	0.0023
Fit time	1.59	2.70	2.52	1.57	1.57	1.99	0.51
Test time	0.12	0.11	0.26	0.12	0.23	0.17	0.07

{'test_rmse': array([0.89420321, 0.89459953, 0.89434847, 0.89578459, 0.90182156]),
'test_mae': array([0.68982268, 0.68764913, 0.6889683 , 0.68923709, 0.69436281]),
'fit_time': (1.5904841423034668,
2.695606231689453,
2.515707015991211,
1.5687916278839111,
1.5691964626312256),
'test_time': (0.1160435676574707,
0.1069185733795166,
0.2638585567474365,
0.11686420440673828,
0.23031830787658691)}

16. Training the SVD Model: Fits the SVD model on the full training dataset to prepare it for making predictions.

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7ce17287f710>
```

17. Predicting a Single Rating: Uses the trained SVD model to predict how a user would rate a specific movie.

```
user_id = 1
movie_id = 10  # Replace with a valid movie ID from your dataset
prediction = svd.predict(user_id, movie_id)
print(f"\nPredicted rating for user {user_id} and movie {movie_id}: {prediction.est}")
```

```
Predicted rating for user 1 and movie 10: 2.5926260967236137
```

18. Defining a Recommendation Function: Creates a function that recommends the top N movies for a given user based on predicted ratings.

```
def recommend_movies(user_id, num_recommendations=10):
    """
    Recommends the top N movies for a given user based on predicted ratings.

    Parameters:
    - user_id: The ID of the user to recommend movies for.
    - num_recommendations: Number of movies to recommend.

    Returns:
    - DataFrame with top recommended movies and their predicted ratings.
    """
    # List all unique movie IDs
    movie_ids = movies['id'].dropna().unique()

    # Predict ratings for all movies the user hasn't rated yet
    movie_ratings = []
    for movie_id in movie_ids:
        try:
            prediction = svd.predict(user_id, int(movie_id))
            movie_ratings.append((movie_id, prediction.est))
        except ValueError:
            continue

    # Sort movies by predicted rating in descending order
    sorted_ratings = sorted(movie_ratings, key=lambda x: x[1], reverse=True)[:num_recommendations]

    # Get movie titles for the recommended IDs
    recommended_movies = pd.DataFrame(sorted_ratings, columns=['movieId', 'predicted_rating'])
    recommended_movies = recommended_movies.merge(movies, left_on='movieId', right_on='id', how='left')

    return recommended_movies[['title', 'predicted_rating']]
```

19. Generating and Displaying Recommendations: Takes a user ID as input and outputs their top 10 movie recommendations using the trained model.

10

Top 10 Recommended Movies:

	title	predicted_rating
0	Sleepless in Seattle	4.594937
1	While You Were Sleeping	4.538354
2	Bonnie and Clyde	4.524048
3	Once Were Warriors	4.512739
4	Broken Blossoms	4.506576
5	Lonely Hearts	4.487654
6	The Good Thief	4.473470
7	The Million Dollar Hotel	4.448557
8	License to Wed	4.436276
9	Hard Target	4.415794

Conclusion: In this project, we analyzed movie data and user ratings from the Full MovieLens dataset. We cleaned the data, visualized trends, and explored user and movie rating patterns. Using the Surprise library, we built a recommendation model with SVD to predict user ratings. The model was evaluated using cross-validation and showed good performance. Finally, we created a function to recommend top movies for any user. This project shows how machine learning can help in building personalized movie recommendations.