Capstone Project

Netflix Recommendation Engine

Dataset-

Copy of combined_data_1.txt.zip,

Copy of movie_titles.csv

Table of Contents

1. Problem Statement

2. Project Objective

3. Data Description

4. Data Pre-processing Steps and Inspiration

5. Choosing the Algorithm for the Project

6. Motivation and Reasons For Choosing the Algorithm

7. Assumptions

8. Model Evaluation and Techniques

9. Inferences from the Same

10. Future Possibilities of the Project

11. Conclusion

Problem Statement

Customer Behaviour and its prediction lies at the core of every Business Model. From Stock Exchange, e-Commerce and Automobile to even Presidential Elections, predictions serve a great purpose. Most of these predictions are based on the data available about a person's activity either online or in-person.

Recommendation Engines are the much-needed manifestations of the desired Predictability of User Activity. Recommendation Engines move one step further and not only give information but put forth strategies to further increase users' interaction with the platform.

Project Objective

In today's world OTT platform and Streaming Services have taken up a big chunk in the Retail and Entertainment industry. Organizations like Netflix, Amazon etc. analyse User Activity Pattern's and suggest products that better suit the user needs and choices.

For the purpose of this Project, we will be creating one such Recommendation Engine from the ground-up, where every single user, based on their area of interest and ratings, would be recommended a list of movies that are best suited for them.

Data Description

Our dataset ‘Copy of combined_data_1.txt.zip’ appears to be related to Netflix movie ratings. Here's what we can infer about it:

- 'Cust ID': Represents customer IDs. These IDs are anonymized numerical identifiers assigned to individual Netflix users.
- ‘Cust ID': Represents customer IDs. These IDs are anonymized numerical identifiers assigned to individual Netflix users.

The dataset might have been extracted from a larger source, and we’re working with only two columns (customer IDs and ratings). The file was in a zipped text format, which is unzipped and read for analysis.

The dataset ‘Copy of movie_titles.csv’ appears to be a collection of movies with four columns:

- MovieID: A unique identifier for each movie, used to distinguish it within the dataset.
- Year: The year in which the movie was released.
- Name: The title of the movie.

Data Preprocessing Steps And Visualisations

1. Reading dataset file and datatypes of each column:

```
df = pd.read_csv('/content/drive/MyDrive/Projects/Netflix Recommendation/Copy of combined_data_1.txt.zip',
                 header=None, names=['Cust ID', 'Rating'], usecols=[0,1])
df
```

	Cust ID	Rating
0	1:	NaN
1	1488844	3.0
2	822109	5.0
3	885013	4.0
4	30878	4.0
...	...	...
24058258	2591364	2.0
24058259	1791000	2.0
24058260	512536	5.0
24058261	988963	3.0
24058262	1704416	3.0

24058263 rows × 2 columns

- File path specifies the location of the file you are trying to read. It is pointing to a compressed file located in your Google Drive.
- header=None: This tells pandas that the file does not contain a header row (i.e., the first row does not have column names). Therefore, pandas won't treat the first row as column headers.

- names = ['CustID', 'Rating']: Since no header is present, this argument assigns custom names to the columns in the resulting DataFrame. The two columns are: CustID represents the unique customer IDs and Rating represents the movie ratings given by customers.
- usecols = [0,1]: This specifies that only the first two columns (with indexes 0 and 1) should be read from the file. If the file contains more than two columns, the extra columns will be ignored.

```
df.dtypes
```

	0
Cust ID	object
Rating	float64

- Datatypes of both the columns where 'Cust ID' represented as object datatype and 'Rating' column uses floating point datatype.

2. Finding Null values:

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

	0
Cust ID	0
Rating	4499

- The above code shows that there are 4499 null values in the Rating column, but the customer id corresponding to these null values doesn't represent any customer in reality. Instead these values represent the identity number of movies present in the dataset.

3. Getting the movie counts:

```
#get the movie count ( as the NaN values in Rating column will show how many movie are there )
movie_count = df.isnull().sum()
movie_count = movie_count['Rating']
print(f'There are about {movie_count} number of movies in the dataset')
```

```
There are about 4499 number of movies in the dataset
```

- Here, the code accesses the count of null values specifically for the column named 'Rating'. This count corresponds to the number of movies in the dataset, because in the dataset structure, rows with missing ratings are used as markers for movies.

4. Calculate how many customers we are having in the dataset:

```
# To claculate how many customers we are having in the dataset
customer_count = df['Cust ID'].nunique()
print(f'There are total {customer_count} number of customers in the dataset')
```

```
There are total 475257 number of customers in the dataset
```

- nunique(): This function calculates the number of distinct (unique) values within the 'Cust ID' column. It essentially counts how many different customer IDs are present in the dataset.
- Prints out the total number of unique customers in the dataset.

5. Total number of customers that we have after removing movie id from the column

```
# Total number of customers that we have after removing movie id from the column
print(f'Total number of customers that we have after removing movie id from the column are:- {customer_count - movie_count}')
```

```
Total number of customers that we have after removing movie id from the column are:- 470758
```

- Our dataset is reduced to 470758 data points representing these much customer reviews.

6. Get the total number of ratings given by the customers

```
#get the total number of ratings given by the customers
rating_count = df['Cust ID'].count() - movie_count
print(f'Ratings given by cutomers:- {rating_count}')
```

```
Ratings given by cutomers:- 24053764
```

- movie_count is a previously calculated variable that represents the number of movies in the dataset. In the dataset structure, rows where the 'Rating' column is NaN are used to mark movies. These rows are subtracted because they aren't actual customer ratings—they act as placeholders indicating new movies.
- rating_count = df['Cust ID'].count() - movie_count: This calculates the total number of ratings given by customers by subtracting the count of movies from the total count of 'Cust ID' entries.
- This means that the dataset contains 24,053,764 ratings given by customers.

7. To find out how many people have rated the movies as 1, 2, 3, 4, 5 stars ratings to the movies

```
#To find out how many people have rated the movies as 1, 2, 3, 4, 5 stars ratings to the movies
stars = df.groupby('Rating')['Rating'].agg(['count'])
stars
```

Rating	count
1.0	1118186
2.0	2439073
3.0	6904181
4.0	8085741
5.0	5506583

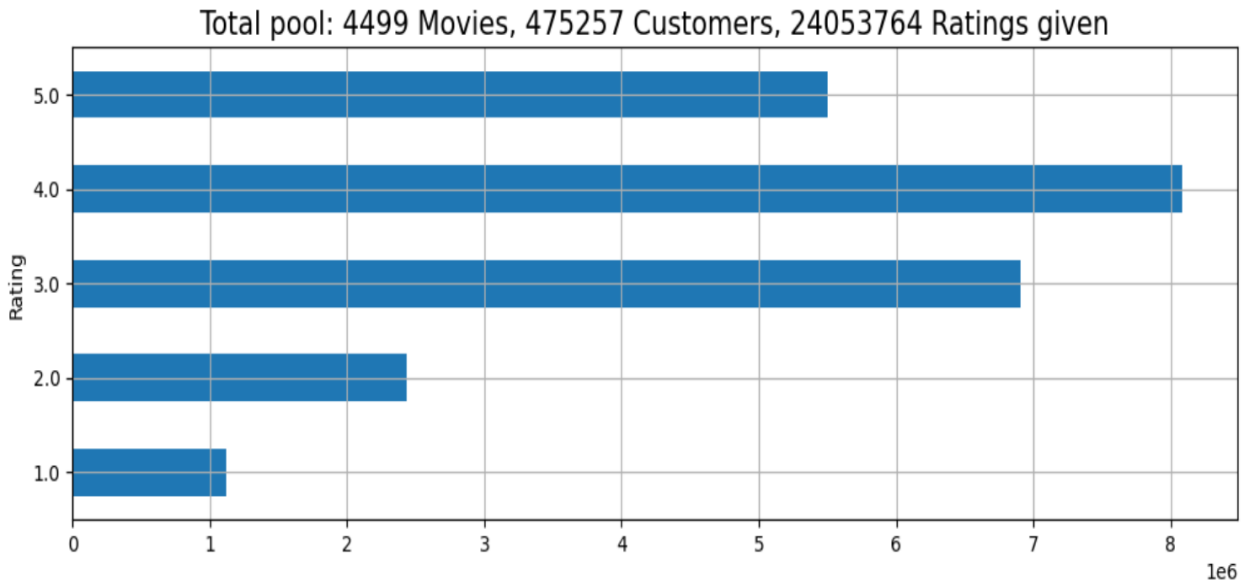
- This groups the data in the DataFrame (df) by the values in the 'Rating' column. After grouping by the 'Rating' column,

this specifies that the aggregation is to be performed on the same column. This applies the aggregation function count() to calculate the number of entries (rows) in each group. Essentially, it counts how many ratings were given for each star level (1, 2, 3, 4, and 5).

- 1-star ratings: 1,118,186 people gave 1-star ratings.
- 2-star ratings: 2,439,073 people gave 2-star ratings.
- 3-star ratings: 6,904,181 people gave 3-star ratings.
- 4-star ratings: 8,085,741 people gave 4-star ratings.
- 5-star ratings: 5,506,583 people gave 5-star ratings.

Plotting rating in bar graph

```
ax=stars.plot(kind='barh', legend=False, figsize=(12,4))
plt.title(f'Total pool: {movie_count} Movies, {customer_count} Customers, {rating_count} Ratings given', fontsize=15)
plt.grid(True)
```



8. Modify the dataframe to find how many MovieIds are there and assign corresponding movie id to each review.

```
movie_id = None
movie = []

for i in df['Cust ID']:
  if ':' in i:
    movie_id = int(i.replace(':', ''))
  movie.append(movie_id)
```

- This initializes the variable movie_id with the value 'None'. It is used to temporarily store the movie ID as the code processes each element in the 'Cust ID' column.
- This initializes an empty list movie to store the processed movie_id values for every row in the DataFrame (df).
- for loop iterates through each element (i) in the 'Cust ID' column of the DataFrame. The column contains customer IDs and rows where movie IDs are stored (marked with colon, :).
- if ':' in i: This checks if the current value (i) in the 'Cust ID' column contains a colon (:). Rows with colons represent movie IDs.
- movie_id = int(i.replace(":", "")): If a colon is found in the value (i), the code removes the colon. The resulting string is converted to an integer using, which ensures the movie ID is stored as a numeric value. The processed movie ID is then assigned to the movie_id variable.

```
df['MovieID'] = movie
```

```
df.head()
```

	Cust ID	Rating	MovieID
0	1:	NaN	1
1	1488844	3.0	1
2	822109	5.0	1
3	885013	4.0	1
4	30878	4.0	1

9. Removing Nan values from rating column:

```
# It excludes rows where the 'Rating' is NaN, to keep only the rows where the 'Rating' column is not null
df = df[df['Rating'].notna()]
```

```
df.head()
```

	Cust ID	Rating	MovieID
1	1488844	3.0	1
2	822109	5.0	1
3	885013	4.0	1
4	30878	4.0	1
5	823519	3.0	1

- The notna() function in pandas is used to detect non-missing values in a DataFrame or Series. It returns a Boolean mask where True indicates that the value is not null (not NaN) and False means it is missing (NaN).

10. Remove all the movies that have rated less

```
# Remove all the movies that have rated less
movie_summary = df.groupby('MovieID')['Rating'].agg(['count'])
movie_summary
```

MovieID	count
1	547
2	145
3	2012
4	142
5	1140
...	...
4495	614
4496	9519
4497	714
4498	269
4499	428

4499 rows × 1 columns

- The resulting table (movie_summary) contains two columns: MovieID having lists of the IDs of movies, and count showing the number of ratings each movie has received. There are 4,499 rows, meaning there are 4,499 unique movies in the dataset.
- This code helps identify movies that have received fewer ratings, enabling further analysis, such as filtering out less-rated movies or understanding rating trends.

Setting a benchmark of movie rating to 60%

```
# Setting a benchmark of movie rating to 60%
movie_benchmark = round(movie_summary['count'].quantile(0.6))
movie_benchmark
```

```
908
```

- We calculate a benchmark for movie ratings at the 60th percentile based on the 'count' column in a DataFrame named movie_summary.
- The output shows 908 means that movies with 908 or more ratings are in the top 40% by popularity, and movies with fewer than 908 ratings are in the bottom 60%.

Movies rated less than benchmark

```
# Movies rated less than benchmark
drop_movie_list = movie_summary[movie_summary['count'] < movie_benchmark].index
drop_movie_list
```

```
Index([   1,    2,    4,    7,    9,   10,   11,   12,   13,   14,
       ...
       4480, 4481, 4486, 4487, 4491, 4494, 4495, 4497, 4498, 4499],
      dtype='int64', name='MovieID', length=2699)
```

Movies left

```
# Movies left
4499 - len(drop_movie_list)
```

```
1800
```

So, finally we are left with 1800 movies only.

11. Remove all the users that have rated less movies

```
# We will remove all the users that have rated less movies
customer_summary = df.groupby('Cust ID')['Rating'].agg(['count'])
customer_summary
```

Cust ID	count
6	153
7	195
8	21
10	49
25	4
...	...
2649404	12
2649409	10
2649421	3
2649426	74
2649429	62

470758 rows × 1 columns

- The table generated by the code has two key components: CustID represents unique customer identifiers and count indicates how many ratings each customer has provided.
- The summary includes 470,758 customers, meaning there are 470,758 unique customers in the dataset who have provided at least one rating.

Creating a rating benchmark

```
cust_benchmark = round(customer_summary['count'].quantile(0.6))
cust_benchmark
```

```
36
```

- This means that customers who have provided 36 or more ratings are in the top 40% of active raters. Customers with fewer than 36 ratings fall in the bottom 60%.

Ratings left

```
drop_cust_list = customer_summary[customer_summary['count'] < cust_benchmark].index
drop_cust_list
```

```
Index([      8,      25,      33,      83,      94,     126,     130,     133,
          142,     149,
       ...
       2649337, 2649343, 2649351, 2649376, 2649379, 2649384, 2649401, 2649404,
       2649409, 2649421],
      dtype='int64', name='Cust ID', length=282042)
```

12. Finally filtering dataframe on basis of customer and rating benchmark

```
df = df[~df['MovieID'].isin(drop_movie_list)]
df = df[~df['Cust ID'].isin(drop_cust_list)]
```

```
df.shape
```

```
(19695836, 3)
```

- df = df[~df['MovieID'].isin(drop_movie_list)]: isin() checks whether the values in the MovieID column of the DataFrame (df) are present in the list drop_movie_list. It returns a boolean Series: True for rows where the movie ID is in the list, and False otherwise.
- df = df[~df['Cust ID'].isin(drop_cust_list)]: This operates similarly to the first line, but it filters rows based on the column CustID instead. Rows where the customer ID is found in drop_cust_list are excluded.
- df.shape: Displays the shape of the modified DataFrame (df), which indicates the total number of rows and columns remaining after filtering. After these filtering operations, the DataFrame contains 19,695,836 rows and 3 columns.

Choosing the Algorithm for the Project

The Surprise library (Simple Python Recommendation System) is a specialized library for building and evaluating recommendation systems. While it is not part of scikit-learn, it is often used in recommendation projects because of its versatility, ease of use, and focus on collaborative filtering techniques. We should consider using this package for our recommendation project:

1. Surprise is specifically designed for recommendation systems, unlike general-purpose libraries. It provides tools and algorithms that are tailored for tasks such as collaborative filtering, which are at the core of most recommendation projects.
2. The library supports a variety of pre-built algorithms, such as Singular Value Decomposition (SVD), KNNBasic, KNNWithMeans, and more. This saves the time and complexity of implementing these algorithms from scratch.
3. The library makes it easy to work with datasets designed for recommendation systems. We can load data directly from files, pandas DataFrames, or even use its built-in dataset loaders for standard datasets. It also preprocesses and splits data into training and testing sets efficiently
4. It offers built-in methods for evaluating recommendation algorithms using various metrics like Root Mean Square Error (RMSE), Mean Absolute Error (MAE). We can perform cross-validation easily, which helps in comparing algorithms and fine-tuning hyperparameters.

5. Many algorithms in Surprise are optimized for better performance and scalability. These optimizations allow it to handle relatively large datasets efficiently.
6. It is beginner-friendly, offering a simple API to quickly test and implement recommendation systems. Its modular design makes it perfect for exploring the inner workings of collaborative filtering and matrix factorization methods.

Assumptions

There are some potential assumptions made during the course of a recommendation system project:

1. Data Assumptions:

   - The dataset contains enough user and item interaction records (e.g., customer ratings or reviews) to provide meaningful insights.
   - The data is accurate, and any errors or biases in the dataset are minimal or negligible.
   - Missing values, such as NaN entries for ratings, are either informative like markers for movies or have been handled appropriately.
   - User preferences and behaviours, as reflected in historical data, remain consistent over time.

2. Model Assumptions:

   - User-item interactions (e.g., ratings) are sufficient to infer preferences and similarities between users or items.
   - The dataset is sparse, meaning not all users rate all movies, which the algorithms are designed to handle.
   - Some models assume that features are independent of each other.

3. Practical Assumptions:

- Users are rational and their ratings or interactions accurately reflect their preferences.
- Assumes the benchmarks (e.g., 60th percentile for ratings or customer activity) are appropriate for filtering and analysis.
- Metrics like RMSE, MAE, or precision/recall are assumed to accurately reflect the model's performance.

4. External Assumptions:

- The dataset doesn't account for external factors (e.g., movie marketing, cultural trends) that may influence user ratings.
- Assumes the data is static during modelling, even though real-world preferences may evolve dynamically.
- Assumes the chosen algorithm can handle the size of the dataset effectively within computational limits.

Model Evaluation and Technique

Model evaluation is a critical step in building and validating a machine learning model, ensuring it performs well on unseen data and meets the objectives of your project. For a recommendation system, evaluation focuses on measuring the model's ability to predict user preferences accurately and efficiently.

- Key Metrics for Evaluation: Root Mean Square Error measures the average error between predicted ratings and actual ratings. Lower RMSE indicates better accuracy. Mean Absolute Error calculates the average magnitude of errors without considering their direction. Smaller values indicate better performance.
- Train-Test-Split: Split the dataset into training and test sets (e.g., 80%-20%). The model is trained on the training set and evaluated on the test set to simulate performance on unseen data.
- Cross-Validation: Data is split into multiple folds, and the model is trained and tested across these folds. Common methods include k-fold cross-validation, which provides robust performance evaluation by training the model on k number of subsets and testing on the remaining subset.
- Evaluation Frameworks and Libraries: Surprise Library is specialized for recommendation systems, offering tools for performance metrics like RMSE and MAE, as well as cross-validation methods.

- Challenges in Evaluation: New users or items with limited interaction history can make accurate evaluation difficult. Popular items or active users can dominate evaluations, leading to skewed results. Evaluating large datasets can be computationally expensive.

In summary, robust evaluation techniques and well-chosen metrics ensure our recommendation system is reliable, accurate, and user-focused.

1. Creating a dataframe containing movie names:

```
# Dataframe containing movie names
df_title = pd.read_csv('/content/drive/MyDrive/Projects/Netflix Recommendation/Copy of movie_titles.csv',
                       encoding = "ISO-8859-1", header = None, usecols=[0,1,2],
                       names = ['MovieID','Year','Name'])
df_title
```

	MovieID	Year	Name
0	1	2003.0	Dinosaur Planet
1	2	2004.0	Isle of Man TT 2004 Review
2	3	1997.0	Character
3	4	1994.0	Paula Abdul's Get Up & Dance
4	5	2004.0	The Rise and Fall of ECW
...	...	...	...
17765	17766	2002.0	Where the Wild Things Are and Other Maurice Se...
17766	17767	2004.0	Fidel Castro: American Experience
17767	17768	2000.0	Epoch
17768	17769	2003.0	The Company
17769	17770	2003.0	Alien Hunter

17770 rows × 3 columns

The resulting DataFrame contains three columns:

- MovieID: Unique numerical identifiers for movies e.g., 1, 2, 3.
- Year: The year of movie release e.g., 2003, 1994.
- Name: Titles of movies (e.g., Dinosaur Planet, Paula Abdul's Get Up & Dance).

It has 17,770 rows, corresponding to 17,770 movies. Each row provides details about one movie.

2. Install the Surprise package:

```
!pip install scikit-surprise
```

3. Imports from Surprise package:

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

This imports the following components from the Surprise library:

- Reader: Used to define how data is read and interpreted. For example, it specifies the scale of user ratings (e.g., ratings range from 1 to 5).

- Dataset: Provides tools to handle and load datasets. You can load data from files, pandas DataFrame, or built-in datasets.
- SVD: Refers to the Singular Value Decomposition algorithm, a matrix factorization technique commonly used in collaborative filtering for recommendation systems. It helps predict missing ratings by capturing latent factors.
- cross-validate: Used to evaluate a model's performance by splitting the data into training and test sets. It calculates evaluation metrics like RMSE and MAE across different folds of the dataset to provide a robust assessment.

4. Creating a small dataset of 100k rows:

```
reader = Reader()
```

```
### Working with only 100k rows
data = Dataset.load_from_df(df[['Cust ID', 'MovieID', 'Rating']][:100000], reader)
data
```

- Creates an instance of the Reader class from the Surprise library.
- Selects three specific columns (MovieID, Cust ID, and Rating) from the pandas DataFrame df. Create a new dataframe data with these having only 10,000 rows.

5. Creating a SVD Model:

```
# Creating SVD model
model = SVD()
```

```
cross_validate(model, data, measures=['RMSE'], cv=3)
```

```
{'test_rmse': array([1.02083761, 1.01829374, 1.01809231]),
 'fit_time': (1.5879175662994385, 4.907165050506592, 3.0987768173217773),
 'test_time': (0.3128345012664795, 0.4503512382507324, 0.8348896503448486)}
```

- model = SVD(): This initializes the Singular Value Decomposition (SVD) model. SVD is a matrix factorization technique commonly used in collaborative filtering. It predicts missing values (e.g., user ratings) by decomposing the user-item interaction matrix into latent factors that capture user and item characteristics.
- cross_validate(model, data, measures=['RMSE'], cv=3): cross_validate used to evaluate recommendation models using cross-validation. The model refers to the SVD model created earlier. The data dataset prepared for the recommendation system, and measures=['RMSE'] specifies the evaluation metric as RMSE (Root Mean Square Error).
- cv=3: Sets the number of folds for cross-validation to 3. This divides the dataset into 3 subsets and evaluates the model on each fold, providing a robust assessment of model performance.

Output Explainations:

- test_rmse: An array containing RMSE values for each fold. Lower RMSE values indicate better prediction accuracy.
- fit_time: A tuple showing the time taken to train the model for each fold.
- test_time: A tuple showing the time taken to evaluate the model on each fold.

6. Filter the dataset for a random specific user:

```
# Filter the dataset for a random specific user
user_rating = df[df['Cust ID'] == 1055714]
user_rating
```

	Cust ID	Rating	MovieID
42163	1055714	5.0	18
92227	1055714	4.0	28
210936	1055714	2.0	30
262254	1055714	5.0	57
279656	1055714	5.0	58
...	...	...	...
23805263	1055714	5.0	4432
23945242	1055714	4.0	4472
23977663	1055714	5.0	4479
24018719	1055714	5.0	4488
24056842	1055714	5.0	4496

340 rows × 3 columns

- This filters the DataFrame by selecting only the rows where the value in the column is 1055714.
- The filtered dataset is stored in the user_rating dataframe, which now contains only the ratings and movie IDs associated with this specific user.

7. Finding the number of unique movies rated by the user

```
# Find the number of unique movies rated by the user
movies_rated_by_user = user_rating['MovieID'].nunique()
movies_rated_by_user
```

```
340
```

- Computes the number of unique values in the specified column. In this case, it counts how many distinct movie IDs are present in the 'MovieID' column, thereby determining the total number of unique movies rated by the user.
- The result is stored in the variable movies_rated_by_user. This represents the count of unique movies rated by the user.
- The output in the code indicates the user has rated 340 unique movies.

8. Making a copy for each individual customer for the recommendation:

```
#making a copy for each individual customer for the recomendation
user_1055714 = df_title.copy()
user_1055714
```

This code prepares a separate copy of the dataset (df_title) to work specifically with recommendations for a customer with id 1055714. The new DataFrame can be customized or manipulated (e.g., filtering movies, adding predicted ratings) without altering the original dataset.

9. Remove dropout movies:

```
## Removing drop out movies
user_1055714 = user_1055714[~user_1055714['MovieID'].isin(drop_movie_list)]
user_1055714
```

	MovieID	Year	Name
2	3	1997.0	Character
4	5	2004.0	The Rise and Fall of ECW
5	6	1997.0	Sick
7	8	2004.0	What the #$*! Do We Know!?
15	16	1996.0	Screamers
...	...	...	...
17765	17766	2002.0	Where the Wild Things Are and Other Maurice Se...
17766	17767	2004.0	Fidel Castro: American Experience
17767	17768	2000.0	Epoch
17768	17769	2003.0	The Company
17769	17770	2003.0	Alien Hunter

- Accesses the 'MovieID' column of the DataFrame, which contains unique identifiers for movies.
- Checks whether each movie ID in the 'MovieID' column is part of the drop_movie_list. This returns a boolean Series: True for movies in the list and False for movies not in the list.

10. Predicting ratings of movies according to the specific user:

```
# predict the score (rating) of that user

user_1055714['Estimate Ratings'] = user_1055714['MovieID'].apply(lambda x: model.predict(1055714, x).est)
```

- apply(): This applies a function to every value in the ‘MovieID’ column of the user_1055714 DataFrame.
- The function processes each movie ID and predicts its rating for the user with ID 1055714.
- Inside the lambda function x refers to the MovieID column, the predict method from the trained recommendation model (model) to predict the rating that user might give to movie.
- The predicted ratings (estimated values) are stored as a new column named ‘Estimate Ratings’ in the new DataFrame.

11. Top 5 recommended movies for the specific user:

```
top5_movies=user_1055714.sort_values('Estimate Ratings', ascending=False).head(5)
top5_movies
```

	MovieID	Year	Name	Estimate Ratings
24	25	1997.0	Inspector Morse 31: Death Is Now My Neighbour	4.258010
27	28	2002.0	Lilo and Stitch	3.963783
29	30	2003.0	Something's Gotta Give	3.940911
17	18	1994.0	Immortal Beloved	3.915399
4	5	2004.0	The Rise and Fall of ECW	3.862225

The code and output demonstrate a process where the top 5 movies, based on their predicted ratings for a specific user (User ID: 1055714), are displayed.

- Sorting: The DataFrame has been sorted in descending order based on the "Estimate Ratings" column, which contains predicted ratings for the user.
- The code retrieves the top 5 movies with the highest estimated ratings after sorting.

The output suggests personalized recommendations based on predicted ratings. These movies are likely to match the user's preferences based on the collaborative filtering model.

Inferences from the Project

Based on your Netflix Recommendation project, here are some possible inferences and insights that could be drawn:

1. User Behaviour Insights:
   - Active Users: Highly active users (e.g., those who rate many movies) play a crucial role in shaping recommendations due to their detailed preferences.
   - Rating Patterns: Distribution of ratings (e.g., most users rate movies as 3 or 4 stars) helps understand general trends in user satisfaction.

2. Movie Popularity Trends:
   - Highly Rated Movies: Certain movies consistently receive high ratings, which could indicate their universal appeal across users.
   - Underrated Gems: Movies rated by fewer users but with high average ratings could be promoted as "hidden gems."
   - Time Periods: Examining the release years could reveal trends, such as modern movies receiving more ratings due to recent user engagement.

3. Model Performance:
   - Recommendation Accuracy: The SVD model, evaluated using RMSE and cross-validation, demonstrated reliable prediction accuracy. Lower RMSE values suggest the model predicts ratings close to the actual ones.

- Scalability: The recommendation system performs well with subsets of data, but scalability to larger datasets requires optimization.
- Ranking Ability: By estimating ratings, the model effectively ranks movies tailored to individual users.

4. Dataset Characteristics:
    - Sparsity: Sparse data (e.g., many users rating only a few movies) necessitates effective handling techniques, such as matrix factorization or hybrid recommendation systems.
    - User-Movie Interaction: Dense user-movie interactions (active users and popular movies) drive more confident predictions in collaborative filtering models.

5. Practical Outcomes:
    - Personalization: Tailored recommendations, such as highlighting top 5 movies for each user, enhance user satisfaction and engagement.
    - Business Value: Insights into user preferences help content providers (like Netflix) curate and promote the right mix of movies to boost customer retention.
    - Decision Support: Filtering and ranking movies based on user ratings and model predictions assist decision-making for content recommendation strategies.

Future Possibilities

The future possibilities of our Netflix Recommendation project are vast and exciting! Here are a few directions we could explore:

1. Enhanced Recommendation Models: Integrate hybrid models combining collaborative filtering with content-based filtering to improve accuracy.
2. Real-Time Recommendations: Implement a system that updates recommendations dynamically as users interact with the platform, providing instant feedback and personalized suggestions.
3. Exploring Implicit Feedback: Leverage implicit data like watch time, search queries, and user activity (e.g. replays) to enhance recommendations.
4. Incorporating Sentiment Analysis: Analyse reviews and user-generated content for movies to refine recommendations based on sentiments expressed by viewers.
5. Global and Regional Preferences: Tailor recommendations for users based on their geographical location or cultural preferences.
6. Business Applications: Predict trends to guide content acquisition and production decisions. Optimize promotional strategies based on predicted user engagement with specific movie genres.

Conclusion

In conclusion, the Netflix Recommendation project successfully demonstrates the ability to create a personalized movie recommendation system by leveraging collaborative filtering methods like SVD. It analyses user behaviour and preferences, addresses sparsity in data, and provides meaningful insights into movie popularity and user engagement. The system's accuracy, scalability, and customization pave the way for practical applications in enhancing user satisfaction and driving business value. Future expansions can include hybrid models, real-time recommendations, and context-aware systems to further improve its efficiency and adaptability.