International Skill Development Corporation
Skills for Tomorrow

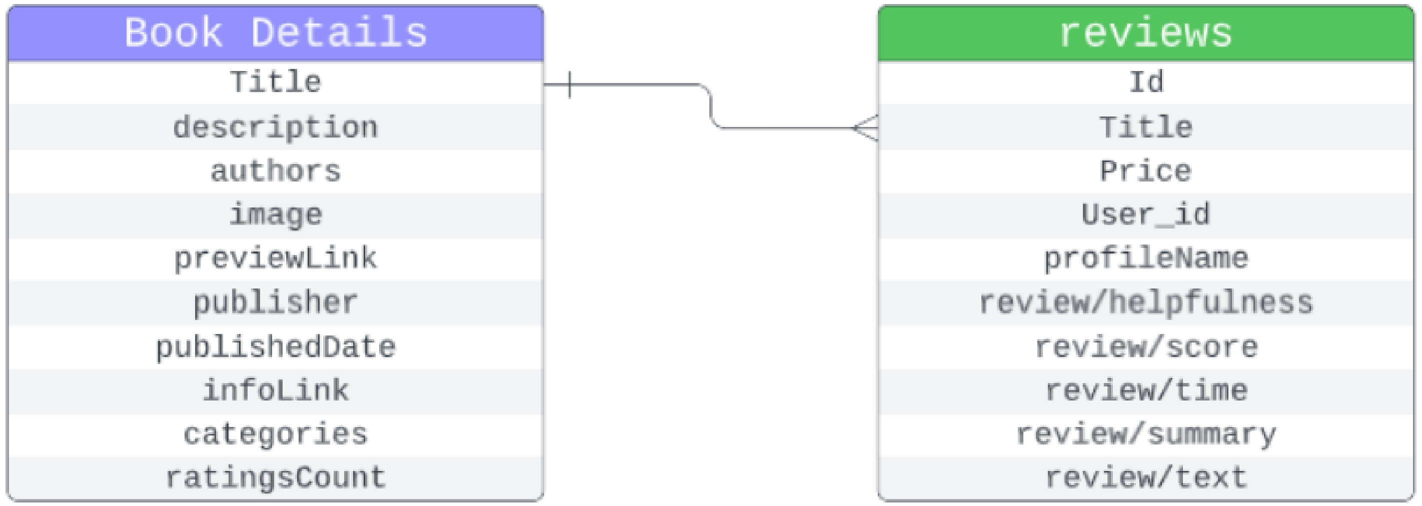
Amazon Books Reviews

Recommendation Project

About Dataset

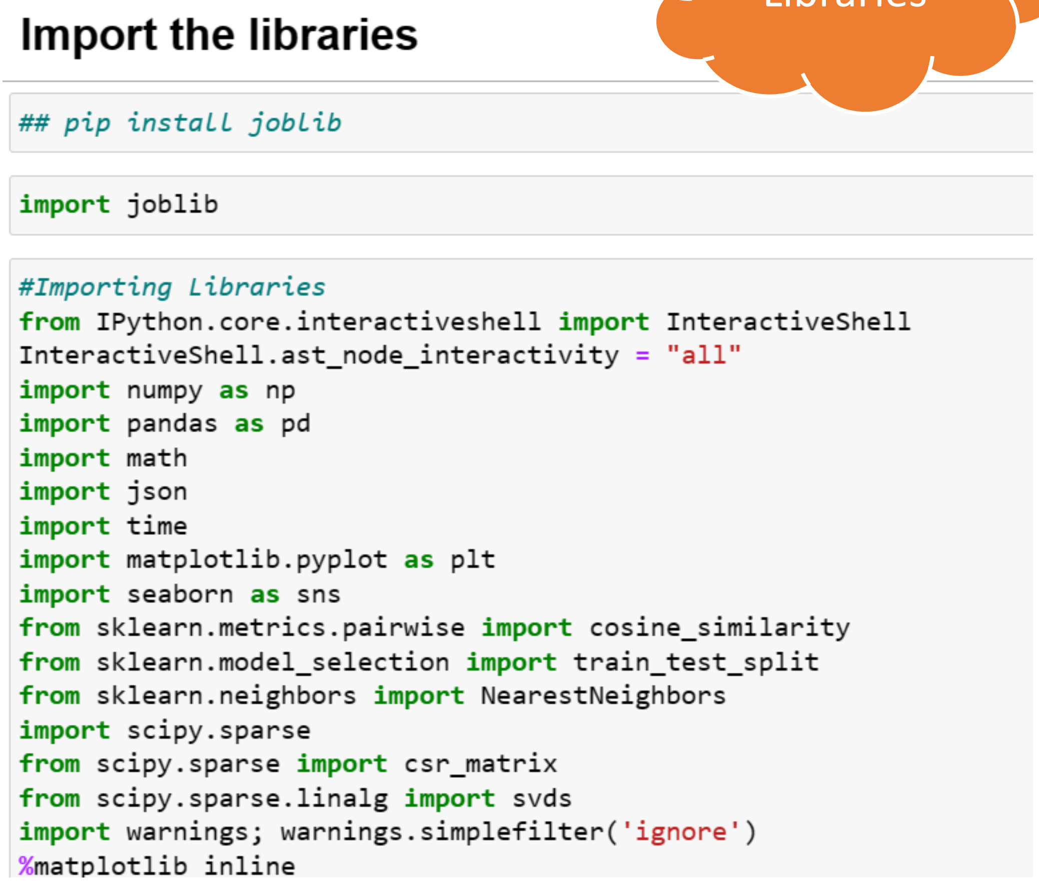
https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews?select=Books_rating.csv

https://nijianmo.github.io/amazon/index.html#complete-data



Features	Description
id	The Id of Book
Title	Book Title
Price	The price of Book
User_id	Id of the user who rates the book
profileName	Name of the user who rates the book
review/helpfulness	helpfulness rating of the review, e.g. 2/3
review/score	rating from 0 to 5 for the book
review/time	time of given the review
review/summary	the summary of a text review
review/text	the full text of a review

Importing Libraries


Import the libraries

```
## pip install joblib
```

```
import joblib
```

```
#Importing Libraries
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import numpy as np
import pandas as pd
import math
import json
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
import scipy.sparse
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds
import warnings; warnings.simplefilter('ignore')
%matplotlib inline
```

The **joblib library** in Python plays a crucial role in saving and loading objects efficiently, especially for large-scale data processing and machine learning tasks. Its main role is to provide a simple and efficient way to serialize Python objects to disk and reload them later, eliminating the need to recompute or recreate those objects from scratch.

The key role of joblib can be summarized as follows:

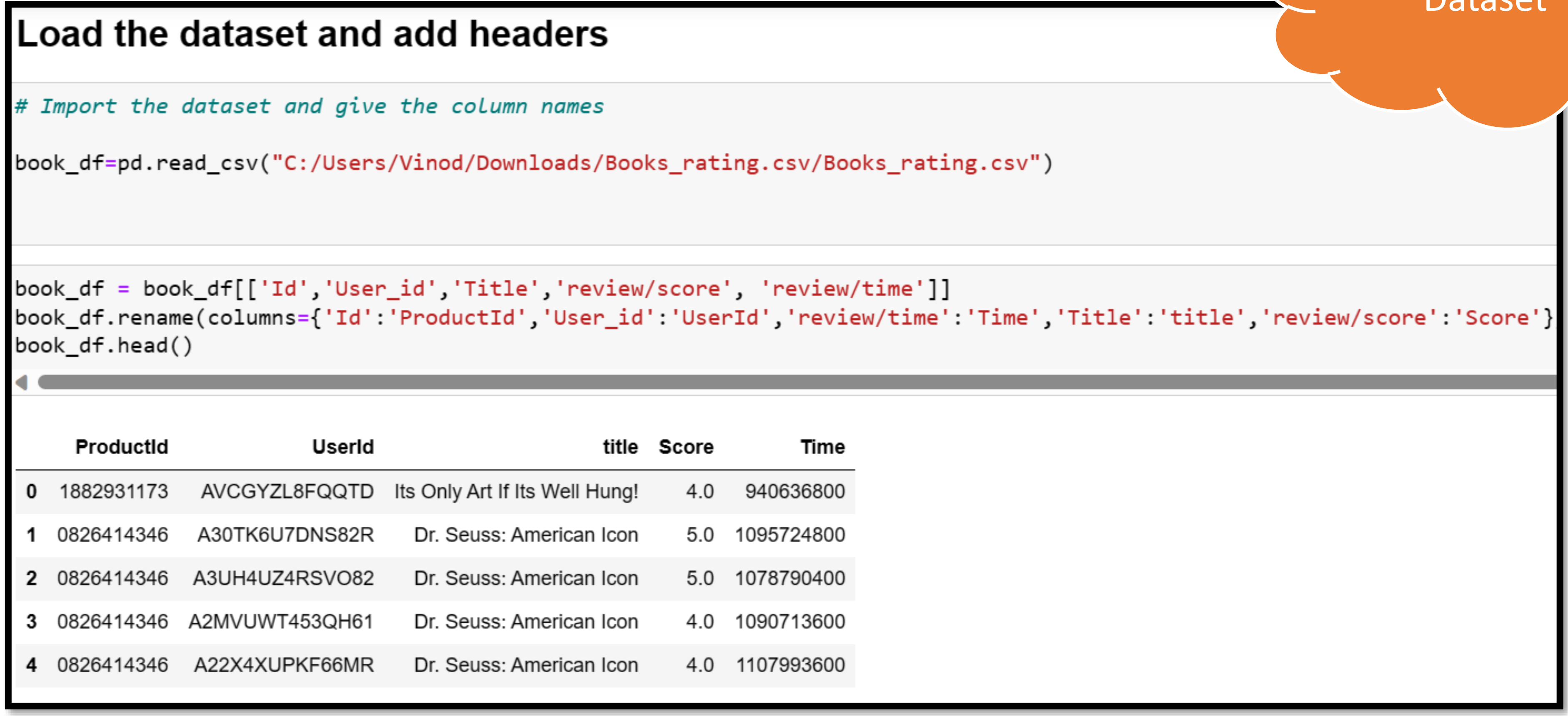
1. Object Serialization: joblib allows you to save and load Python objects, such as machine learning models, NumPy arrays, and other data structures, to disk. This is useful for preserving the state of objects and sharing them across different Python sessions or with other users.

2. Efficient Storage: joblib uses a compressed binary format to store objects, optimizing disk space usage and enabling faster read and write operations. It efficiently handles large objects, preventing excessive memory consumption.

3. Parallel Computing: joblib provides utilities for parallel computing, allowing you to execute multiple tasks simultaneously across multiple cores or processors. It simplifies parallelizing your code and efficiently leveraging the available computational resources.

4. Caching: joblib includes a caching mechanism that enables you to memoize function calls and cache the results. This is particularly useful when working with computationally expensive functions or when you want to avoid recomputing results for the same inputs.

Importing Dataset



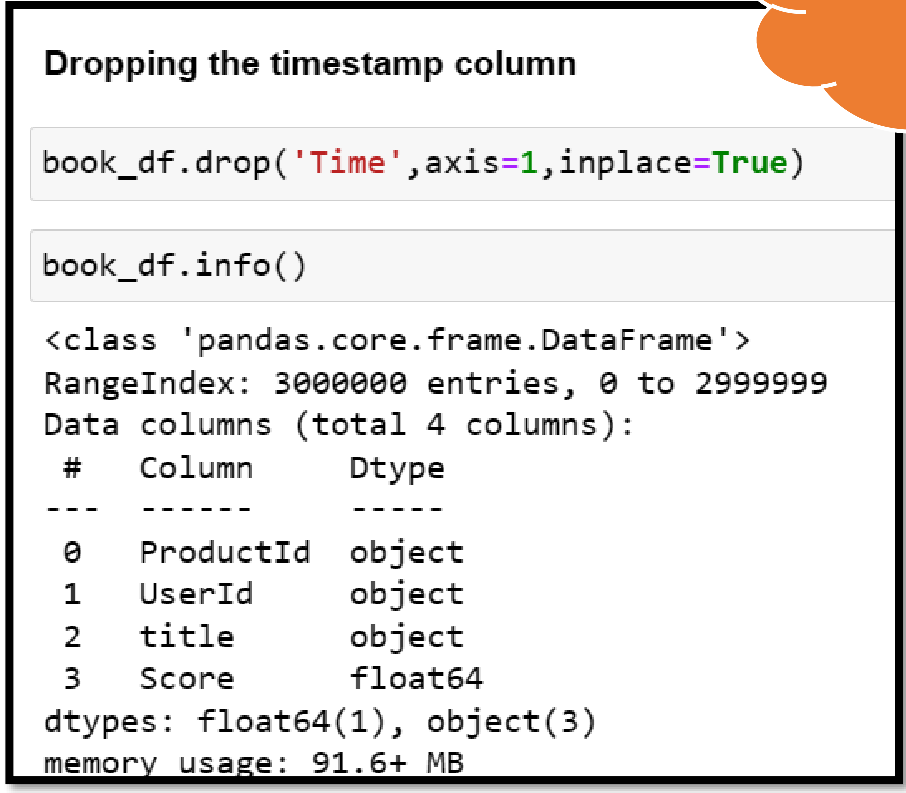
Load the dataset and add headers

```
# Import the dataset and give the column names

book_df=pd.read_csv("C:/Users/Vinod/Downloads/Books_rating.csv/Books_rating.csv")
```

```
book_df = book_df[['Id','User_id','Title','review/score', 'review/time']]
book_df.rename(columns={'Id':'ProductId','User_id':'UserId','review/time':'Time','Title':'title','review/score':'Score'}
book_df.head()
```

	ProductId	UserId	title	Score	Time
0	1882931173	AVCGYZL8FQQTD	Its Only Art If Its Well Hung!	4.0	940636800
1	0826414346	A30TK6U7DNS82R	Dr. Seuss: American Icon	5.0	1095724800
2	0826414346	A3UH4UZ4RSVO82	Dr. Seuss: American Icon	5.0	1078790400
3	0826414346	A2MVUWT453QH61	Dr. Seuss: American Icon	4.0	1090713600
4	0826414346	A22X4XUPKF66MR	Dr. Seuss: American Icon	4.0	1107993600

Dropping “Time” column

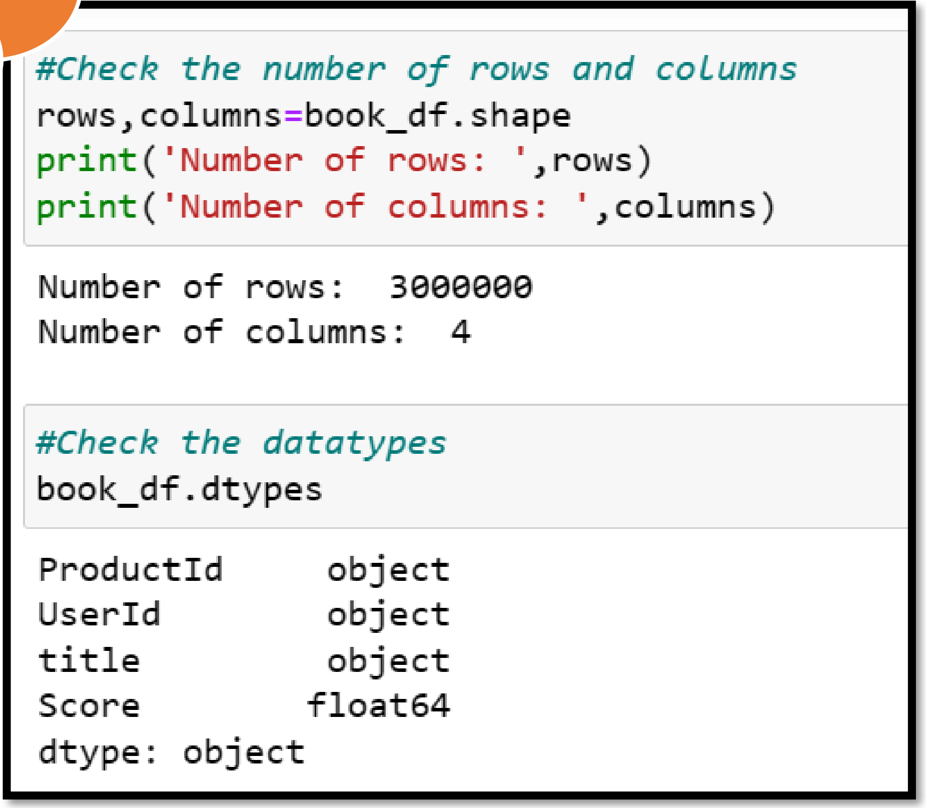


Dropping the timestamp column

```
book_df.drop('Time',axis=1,inplace=True)
```

```
book_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000000 entries, 0 to 2999999
Data columns (total 4 columns):
 #   Column     Dtype  
---  ------     -----  
 0   ProductId  object 
 1   UserId     object 
 2   title      object 
 3   Score      float64
dtypes: float64(1), object(3)
memory usage: 91.6+ MB
```

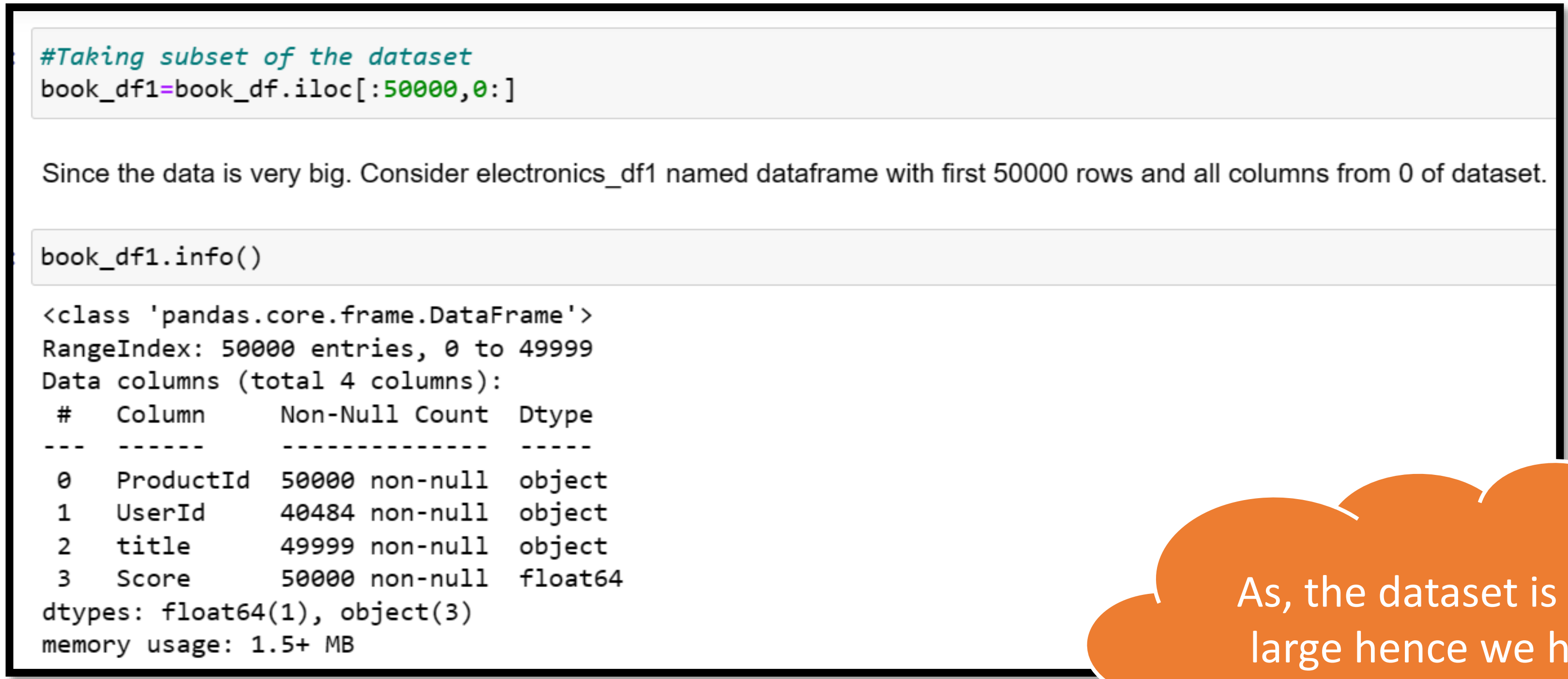


```
#Check the number of rows and columns
rows,columns=book_df.shape
print('Number of rows: ',rows)
print('Number of columns: ',columns)
```

```
Number of rows:  3000000
Number of columns:  4
```

```
#Check the datatypes
book_df.dtypes
```

```
ProductId     object
UserId        object
title         object
Score        float64
dtype: object
```



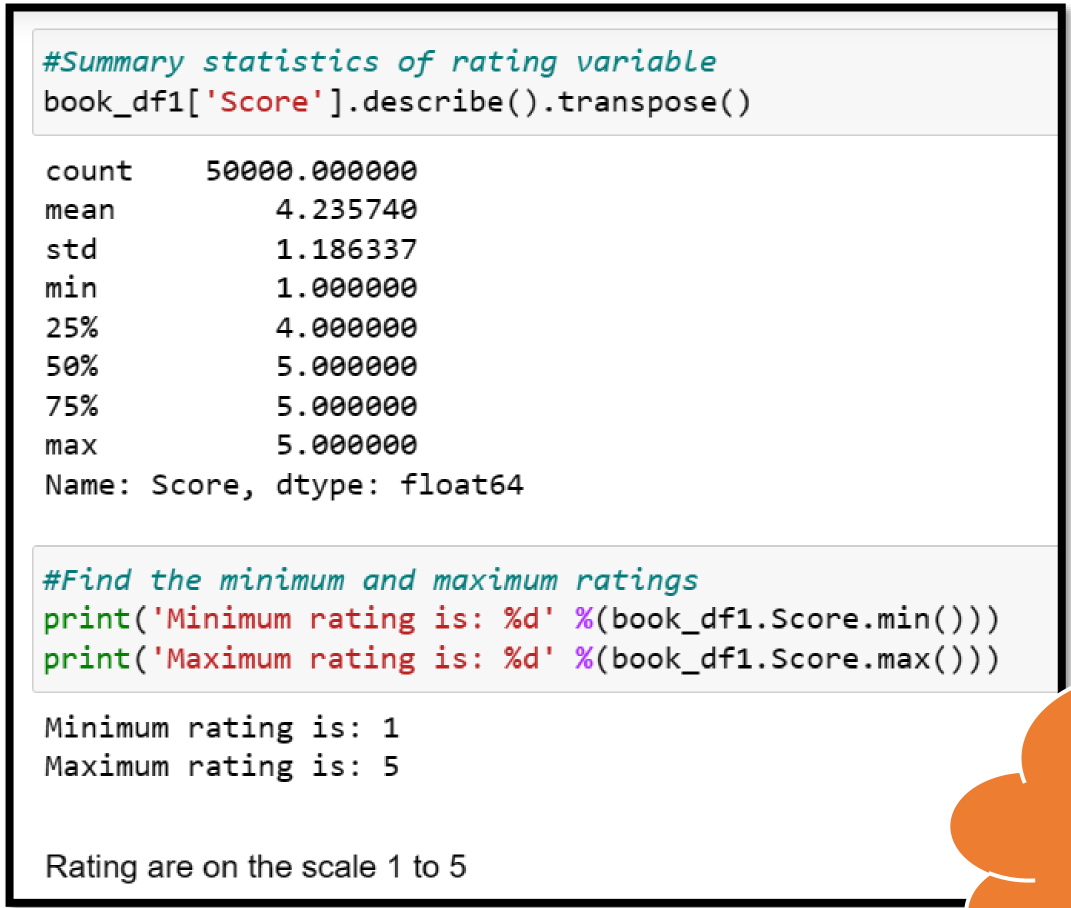
```
#Taking subset of the dataset
book_df1=book_df.iloc[:50000,0:]
```

Since the data is very big. Consider electronics_df1 named dataframe with first 50000 rows and all columns from 0 of dataset.

```
book_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ProductId  50000 non-null  object 
 1   UserId     40484 non-null  object 
 2   title      49999 non-null  object 
 3   Score      50000 non-null  float64
dtypes: float64(1), object(3)
memory usage: 1.5+ MB
```

As, the dataset is very large hence we have shorten the data.



```
#Summary statistics of rating variable
book_df1['Score'].describe().transpose()
```

```
count    50000.000000
mean         4.235740
std          1.186337
min          1.000000
25%          4.000000
50%          5.000000
75%          5.000000
max          5.000000
Name: Score, dtype: float64
```

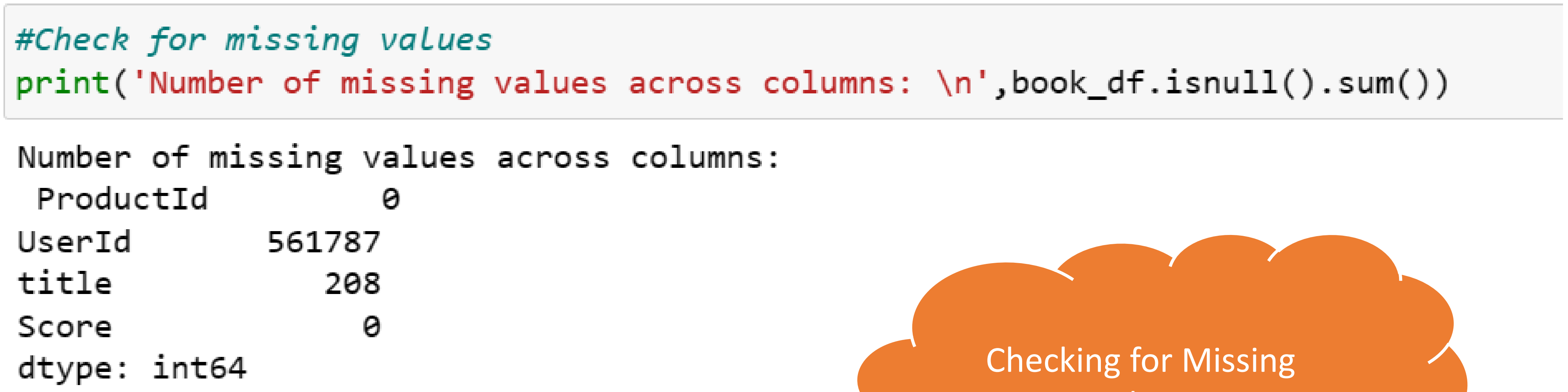
```
#Find the minimum and maximum ratings
print('Minimum rating is: %d' %(book_df1.Score.min()))
print('Maximum rating is: %d' %(book_df1.Score.max()))
```

```
Minimum rating is: 1
Maximum rating is: 5
```

Rating are on the scale 1 to 5

Summary of Score ranging from 1 to 5.

Handling Missing values



```python
#Check for missing values
print('Number of missing values across columns: \n',book_df.isnull().sum())
```

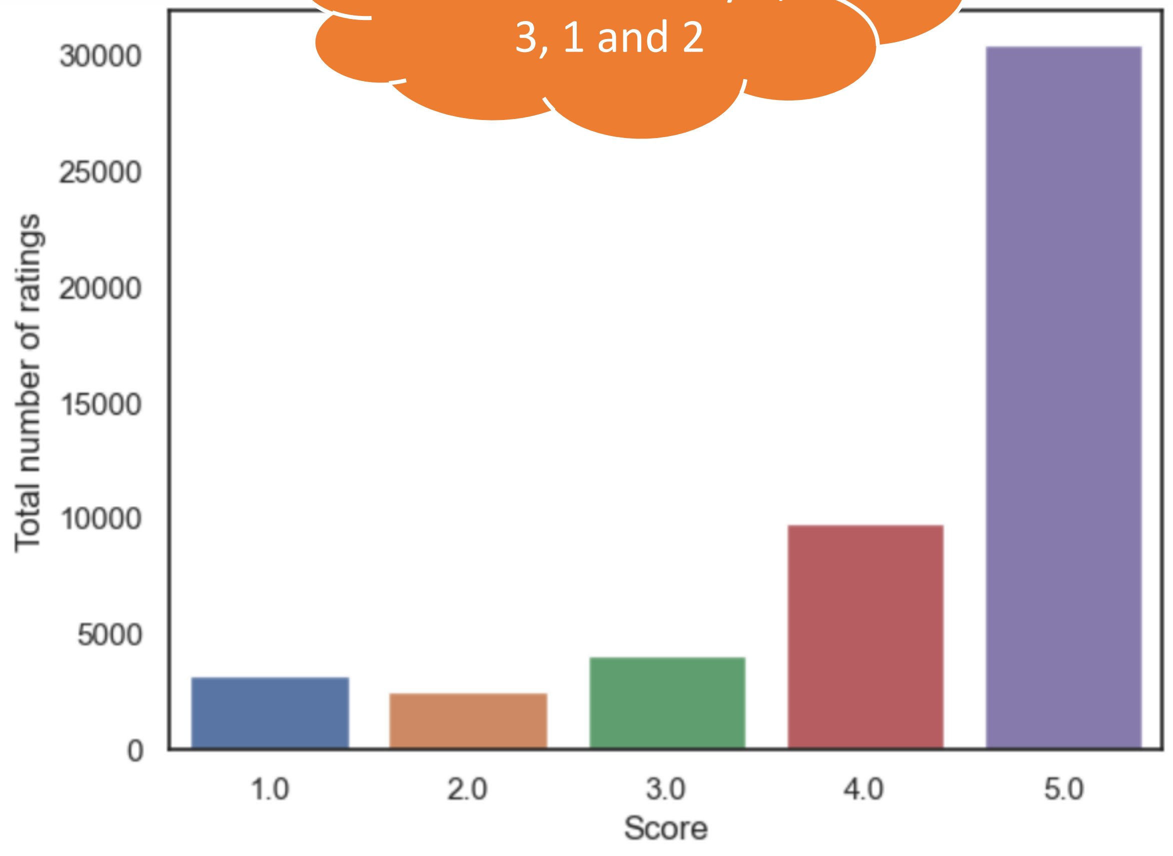
```
Number of missing values across columns:
 ProductId         0
UserId       561787
title           208
Score             0
dtype: int64
```

Checking for Missing Values

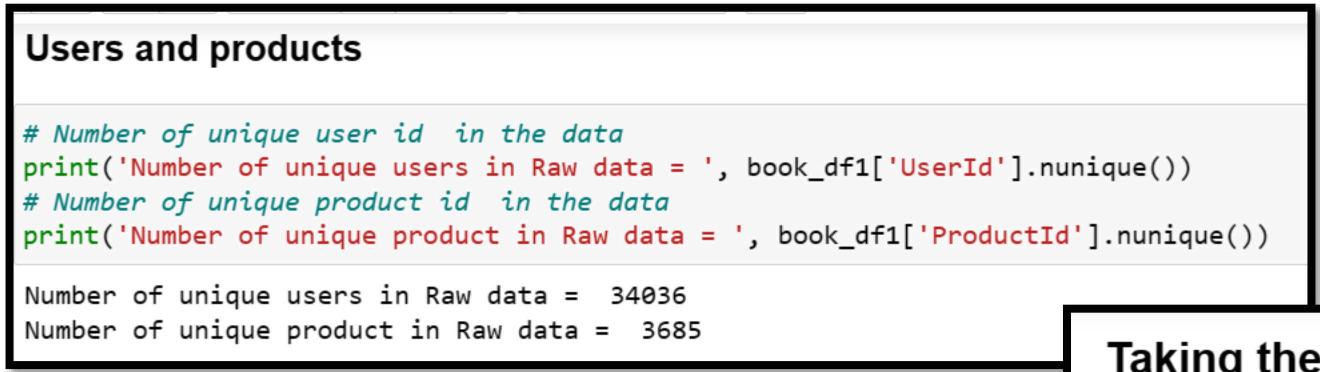
Ratings

```
import seaborn as sns
import matplotlib.pyplot as plt

# Check the distribution of the rating
sns.set(style="white")
g = sns.countplot(x="Score", data=book_df1)
g.set_ylabel("Total number of ratings")

plt.show()
```

```
Text(0, 0.5, 'Total number of ratings')
```



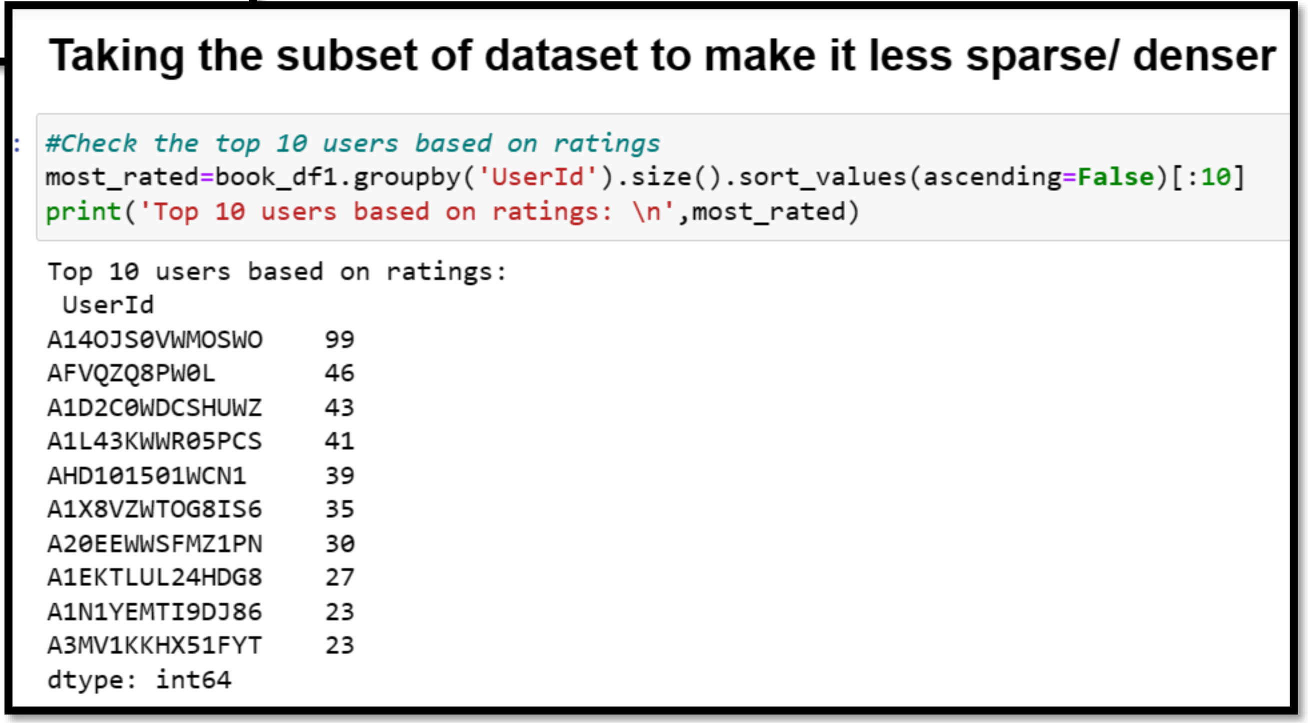
Maximum rating is of 5 followed by 4, 3, 1 and 2



Users and products

```python
# Number of unique user id  in the data
print('Number of unique users in Raw data = ', book_df1['UserId'].nunique())
# Number of unique product id  in the data
print('Number of unique product in Raw data = ', book_df1['ProductId'].nunique())
```

```
Number of unique users in Raw data =  34036
Number of unique product in Raw data =  3685
```

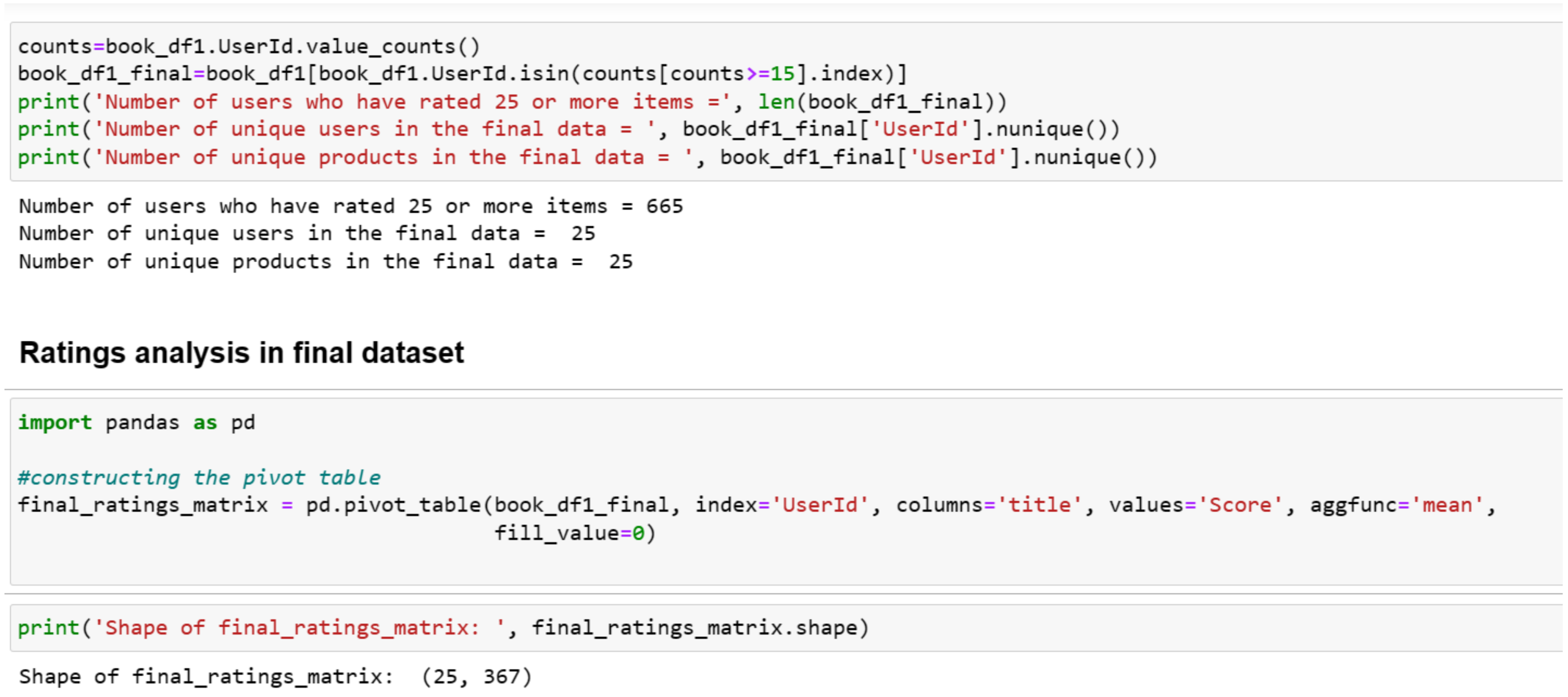


Taking the subset of dataset to make it less sparse/ denser

```python
#Check the top 10 users based on ratings
most_rated=book_df1.groupby('UserId').size().sort_values(ascending=False)[:10]
print('Top 10 users based on ratings: \n',most_rated)
```

```
Top 10 users based on ratings:
 UserId
A14OJS0VWMOSWO    99
AFVQZQ8PW0L       46
A1D2C0WDCSHUWZ    43
A1L43KWWR05PCS    41
AHD101501WCN1     39
A1X8VZWTOG8IS6    35
A20EEWWSFMZ1PN    30
A1EKTLUL24HDG8    27
A1N1YEMTI9DJ86    23
A3MV1KKHX51FYT    23
dtype: int64
```

Top 10 users based on ratings (1 -5)



```python
counts=book_df1.UserId.value_counts()
book_df1_final=book_df1[book_df1.UserId.isin(counts[counts>=15].index)]
print('Number of users who have rated 25 or more items =', len(book_df1_final))
print('Number of unique users in the final data = ', book_df1_final['UserId'].nunique())
print('Number of unique products in the final data = ', book_df1_final['UserId'].nunique())
```

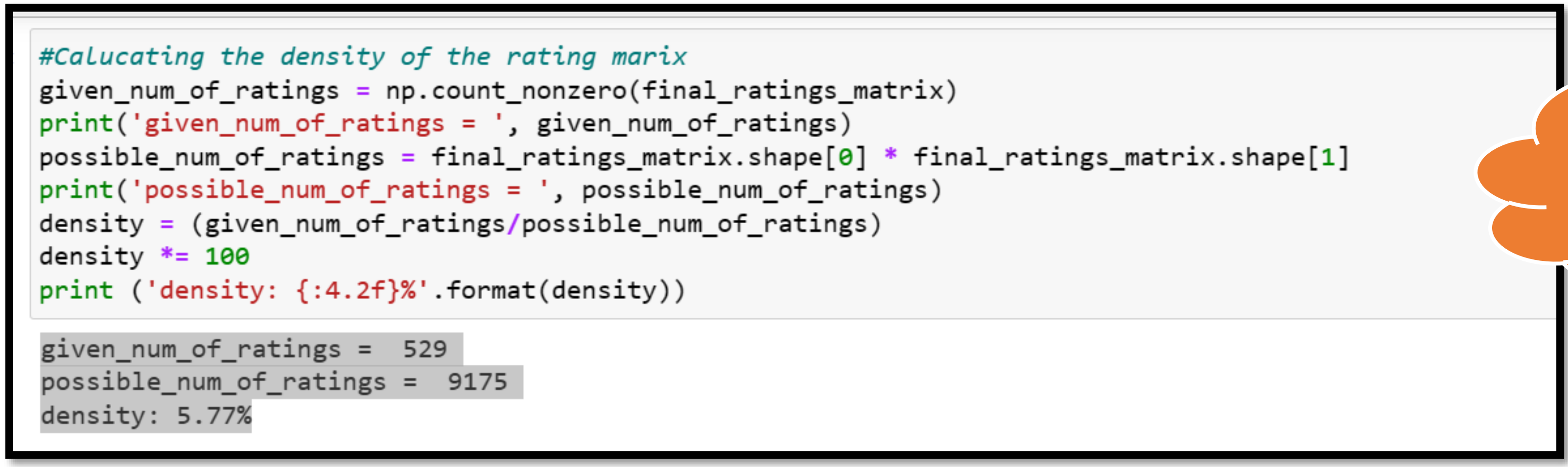
```
Number of users who have rated 25 or more items = 665
Number of unique users in the final data =  25
Number of unique products in the final data =  25
```

Ratings analysis in final dataset

```python
import pandas as pd

#constructing the pivot table
final_ratings_matrix = pd.pivot_table(book_df1_final, index='UserId', columns='title', values='Score', aggfunc='mean',
                                      fill_value=0)
```

```python
print('Shape of final_ratings_matrix: ', final_ratings_matrix.shape)
```

```
Shape of final_ratings_matrix:  (25, 367)
```

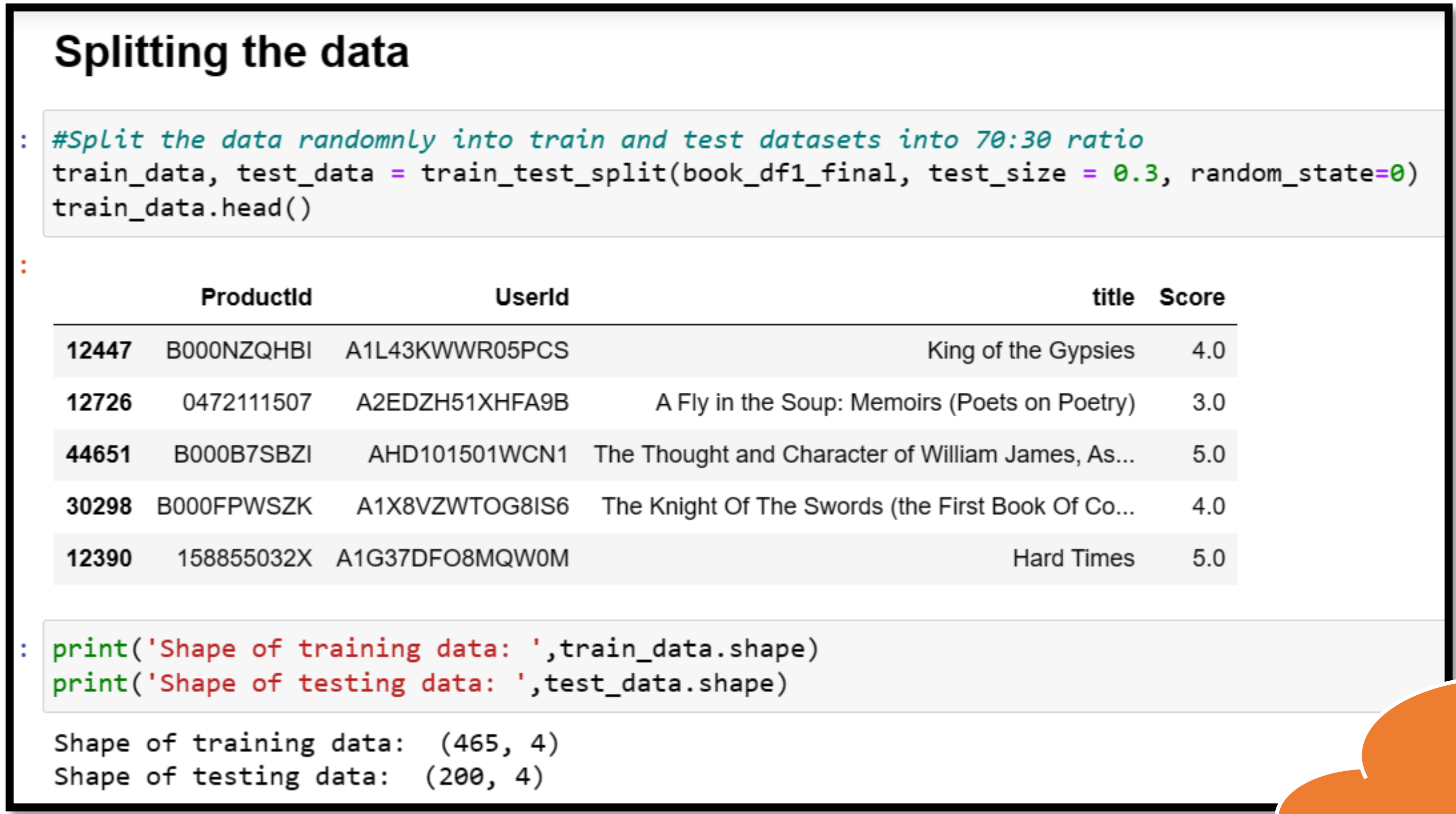


```
#Calucating the density of the rating marix
given_num_of_ratings = np.count_nonzero(final_ratings_matrix)
print('given_num_of_ratings = ', given_num_of_ratings)
possible_num_of_ratings = final_ratings_matrix.shape[0] * final_ratings_matrix.shape[1]
print('possible_num_of_ratings = ', possible_num_of_ratings)
density = (given_num_of_ratings/possible_num_of_ratings)
density *= 100
print ('density: {:4.2f}%'.format(density))
```

```
given_num_of_ratings =  529
possible_num_of_ratings =  9175
density: 5.77%
```

Density: 5.77 %

- In this case, we have a given number of ratings of 529, which means there are 529 observed ratings in the dataset. The possible number of ratings, on the other hand, is 9175, indicating the total number of ratings that could potentially exist in the dataset if all items were rated by all users.
- The density is calculated by dividing the given number of ratings by the possible number of ratings and multiplying the result by 100 to express it as a percentage. In this case, the density is 5.77%.
- The density value represents the proportion of observed ratings in relation to the total number of possible ratings.



Splitting the data

```
#Split the data randomnly into train and test datasets into 70:30 ratio
train_data, test_data = train_test_split(book_df1_final, test_size = 0.3, random_state=0)
train_data.head()
```

	ProductId	UserId	title	Score
12447	B000NZQHBI	A1L43KWWR05PCS	King of the Gypsies	4.0
12726	0472111507	A2EDZH51XHFA9B	A Fly in the Soup: Memoirs (Poets on Poetry)	3.0
44651	B000B7SBZI	AHD101501WCN1	The Thought and Character of William James, As...	5.0
30298	B000FPWSZK	A1X8VZWTOG8IS6	The Knight Of The Swords (the First Book Of Co...	4.0
12390	158855032X	A1G37DFO8MQW0M	Hard Times	5.0

```
print('Shape of training data: ',train_data.shape)
print('Shape of testing data: ',test_data.shape)
```

```
Shape of training data:  (465, 4)
Shape of testing data:  (200, 4)
```

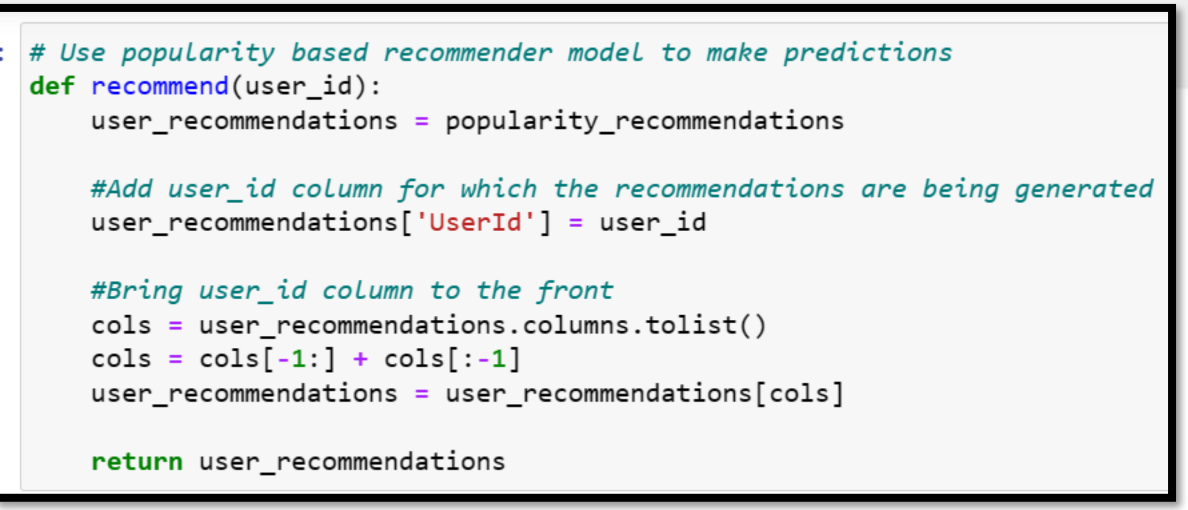
Splitting the Dataset

Building Popularity Recommder model

```python
#Count of user_id for each unique product as recommendation score
train_data_grouped = train_data.groupby('ProductId').agg({'UserId': 'count'}).reset_index()
train_data_grouped.rename(columns = {'UserId': 'Score'},inplace=True)
train_data_grouped.head(50)
```

```python
#Sort the products on recommendation score
train_data_sort = train_data_grouped.sort_values(['Score', 'ProductId'], ascending = [0,1])

#Generate a recommendation rank based upon score
train_data_sort['rank'] = train_data_sort['Score'].rank(ascending=0, method='first')

#Get the top 5 recommendations
popularity_recommendations = train_data_sort.head(5)
popularity_recommendations
```

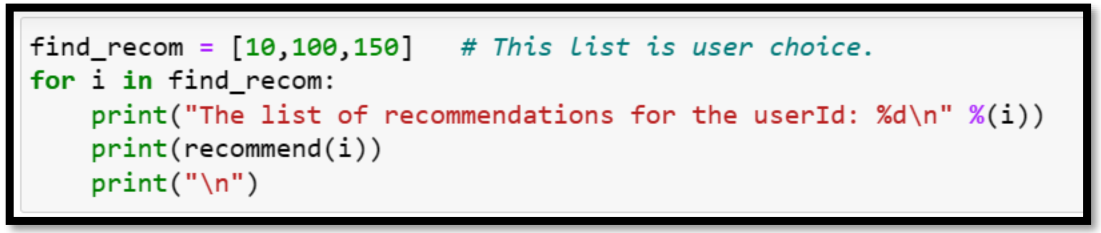
	ProductId	Score	rank
122	1581180012	16	1.0
74	0786197005	13	2.0
227	B000JJ9ISM	11	3.0
159	9562910334	10	4.0
274	B000TZ19TC	10	5.0



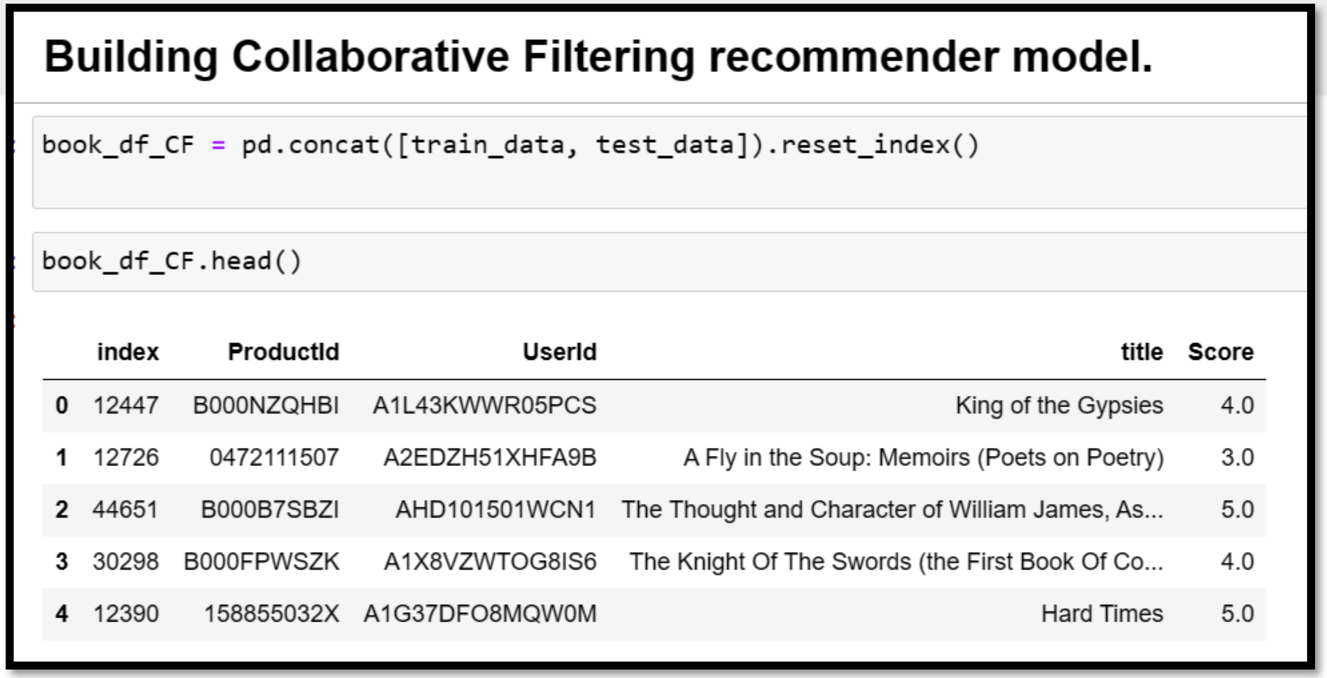
```python
# Use popularity based recommender model to make predictions
def recommend(user_id):
    user_recommendations = popularity_recommendations

    #Add user_id column for which the recommendations are being generated
    user_recommendations['UserId'] = user_id

    #Bring user_id column to the front
    cols = user_recommendations.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    user_recommendations = user_recommendations[cols]

    return user_recommendations
```

The given code snippet implements a popularity-based recommender model to make predictions for a specific user.

- The function recommends (user_id) takes a user_id as input.
- It assigns the popularity_recommendations (previously calculated recommendations based on item popularity) to the variable user_recommendations.
- It adds a new column 'UserId' to the user_recommendations DataFrame, indicating the user for whom the recommendations are being generated.
- It rearranges the columns of the DataFrame, bringing the 'UserId' column to the front.
- Finally, it returns the updated user_recommendations DataFrame containing the recommendations for the specified user.


```
find_recom = [10,100,150]   # This list is user choice.
for i in find_recom:
    print("The list of recommendations for the userId: %d\n" %(i))
    print(recommend(i))
    print("\n")
```

The provided lists of recommendations are generated for different user IDs (10, 100, and 150) using the popularity-based recommender model. Each list contains the top 5 recommendations for the respective user.
For all three users, the recommendations are the same, suggesting a consistent popularity pattern in the dataset.
The recommendations are presented in a tabular format, with each row representing a recommended item. The columns include:
•UserId: The ID of the user for whom the recommendations are generated (10, 100, or 150).
•ProductId: The ID of the recommended product.
•Score: The popularity score or ranking assigned to the product based on its popularity among all users.
•Rank: The rank of the recommendation within the list of recommendations.
In this case, the top recommendation for all users is the product with the ProductId '1581180012', followed by '0786197005', 'B000JJ9ISM', '9562910334', and 'B000TZ19TC' respectively.

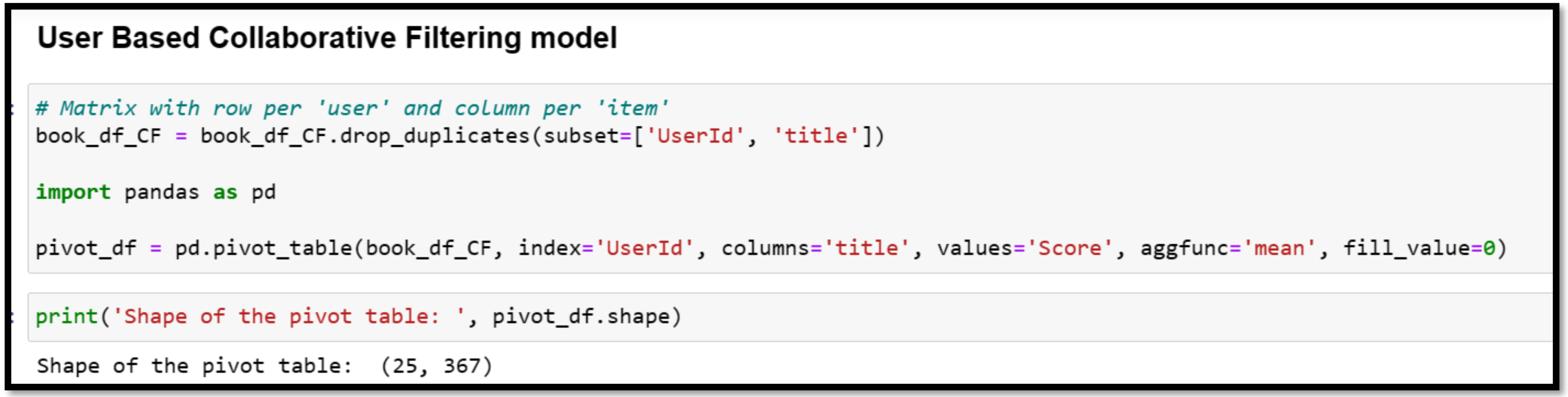


Building Collaborative Filtering recommender model.

```
book_df_CF = pd.concat([train_data, test_data]).reset_index()
```

```
book_df_CF.head()
```

	index	ProductId	UserId	title	Score
0	12447	B000NZQHBI	A1L43KWWR05PCS	King of the Gypsies	4.0
1	12726	0472111507	A2EDZH51XHFA9B	A Fly in the Soup: Memoirs (Poets on Poetry)	3.0
2	44651	B000B7SBZI	AHD101501WCN1	The Thought and Character of William James, As...	5.0
3	30298	B000FPWSZK	A1X8VZWTOG8IS6	The Knight Of The Swords (the First Book Of Co...	4.0
4	12390	158855032X	A1G37DFO8MQW0M	Hard Times	5.0



User Based Collaborative Filtering model

```
# Matrix with row per 'user' and column per 'item'
book_df_CF = book_df_CF.drop_duplicates(subset=['UserId', 'title'])

import pandas as pd

pivot_df = pd.pivot_table(book_df_CF, index='UserId', columns='title', values='Score', aggfunc='mean', fill_value=0)
```

```
print('Shape of the pivot table: ', pivot_df.shape)
```

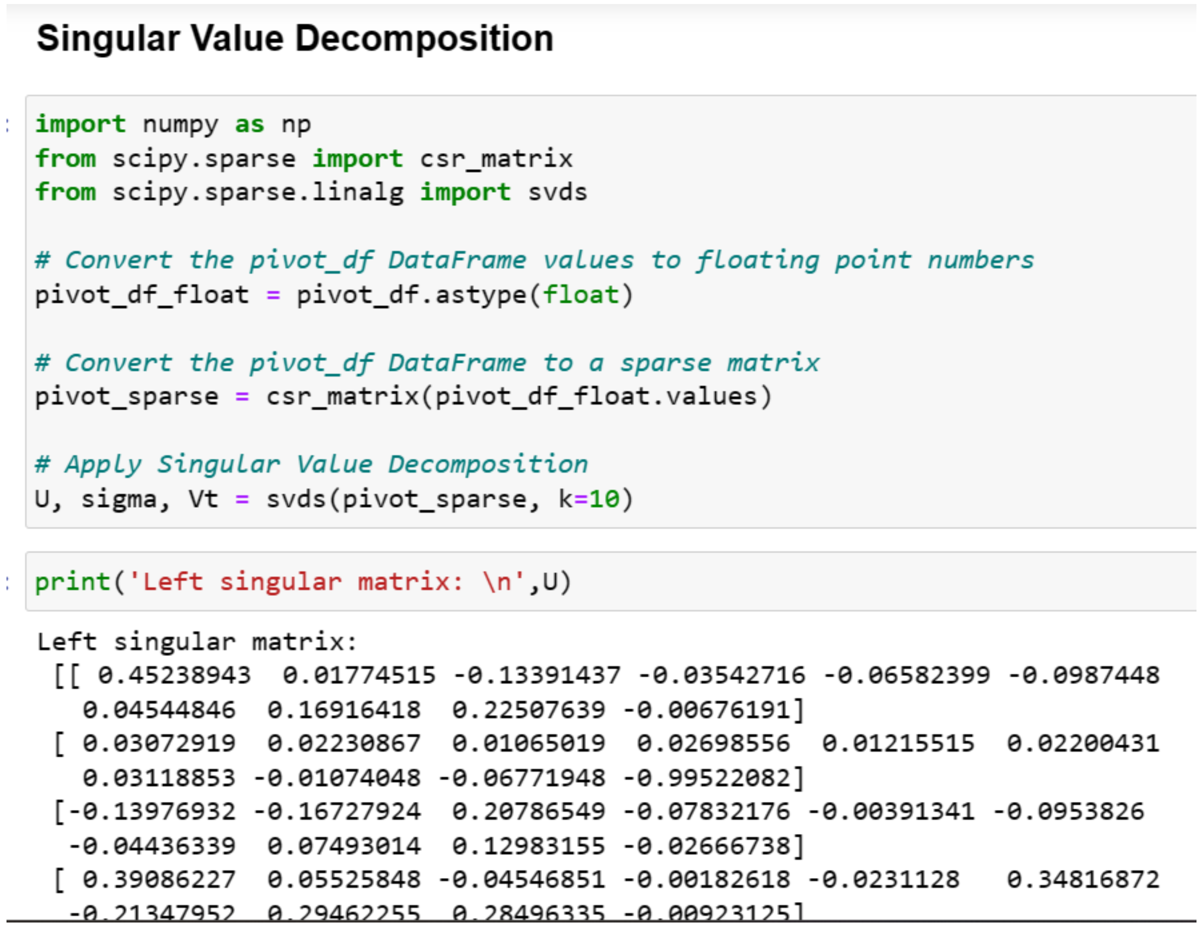
```
Shape of the pivot table:  (25, 367)
```

```python
pivot_df.set_index(['user_index'], inplace=True)
# Actual ratings given by users
pivot_df.head()
```

title	'night, Mother: A Play (Mermaid Dramabook)	.NET Multithreading	1984	4 Secrets of High Performing Organizations: Beyond the Flavor of the Month to Lasting Results	4 Zinas: A Story of Mothers and Daughters on the Mormon Frontier	A Brief Conversion and Other Stories (Karen and Michael Braziller Books)	A Fly in the Soup: Memoirs (Poets on Poetry)	A Peaceful Realm : The Rise And Fall of the Indus Civilization	A Princess of Mars	A Study in Scarlet and The Sign of Four	...	Voyage from yesteryear	WIND RIDER'S OATH	We	Ni (Si M
user_index															
0	0	0	0	0	0	0	0	0	0	0	...	0	0	4	
1	0	5	0	5	5	5	0	5	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	4	0	...	0	0	0	
3	0	0	5	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 367 columns

- Left Singular Matrix (U): The left singular matrix represents the relationships between users and latent factors. In this case, it is a matrix of shape (number of users, number of latent factors). Each row of the matrix corresponds to a user, and each column represents a latent factor. The values in the matrix indicate the strength of association between a user and a latent factor. The values can range from negative to positive, indicating the direction and magnitude of the association.
- Singular Values (Sigma): The singular values are the diagonal elements of the sigma matrix. They represent the importance or significance of each latent factor. The singular values are sorted in descending order, indicating the relative contribution of each latent factor to the overall variability in the data. Larger singular values correspond to more influential or important latent factors.
- Right Singular Matrix (V^T): The right singular matrix represents the relationships between items and latent factors. It has a shape of (number of latent factors, number of items). Each row represents a latent factor, and each column corresponds to an item. The values in the matrix indicate the association between a latent factor and an item. Similar to the left singular matrix, the values can range from negative to positive.

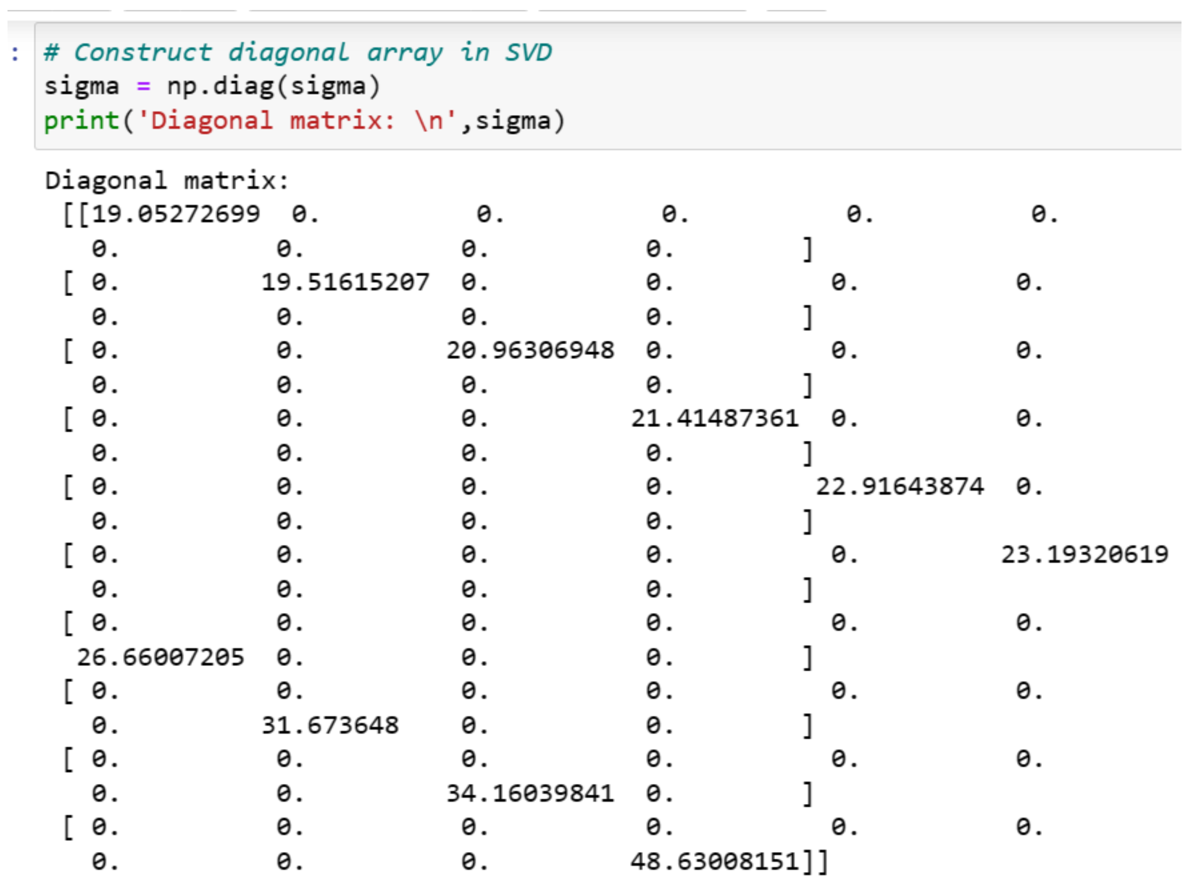


Singular Value Decomposition

```python
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds

# Convert the pivot_df DataFrame values to floating point numbers
pivot_df_float = pivot_df.astype(float)

# Convert the pivot_df DataFrame to a sparse matrix
pivot_sparse = csr_matrix(pivot_df_float.values)

# Apply Singular Value Decomposition
U, sigma, Vt = svds(pivot_sparse, k=10)
```

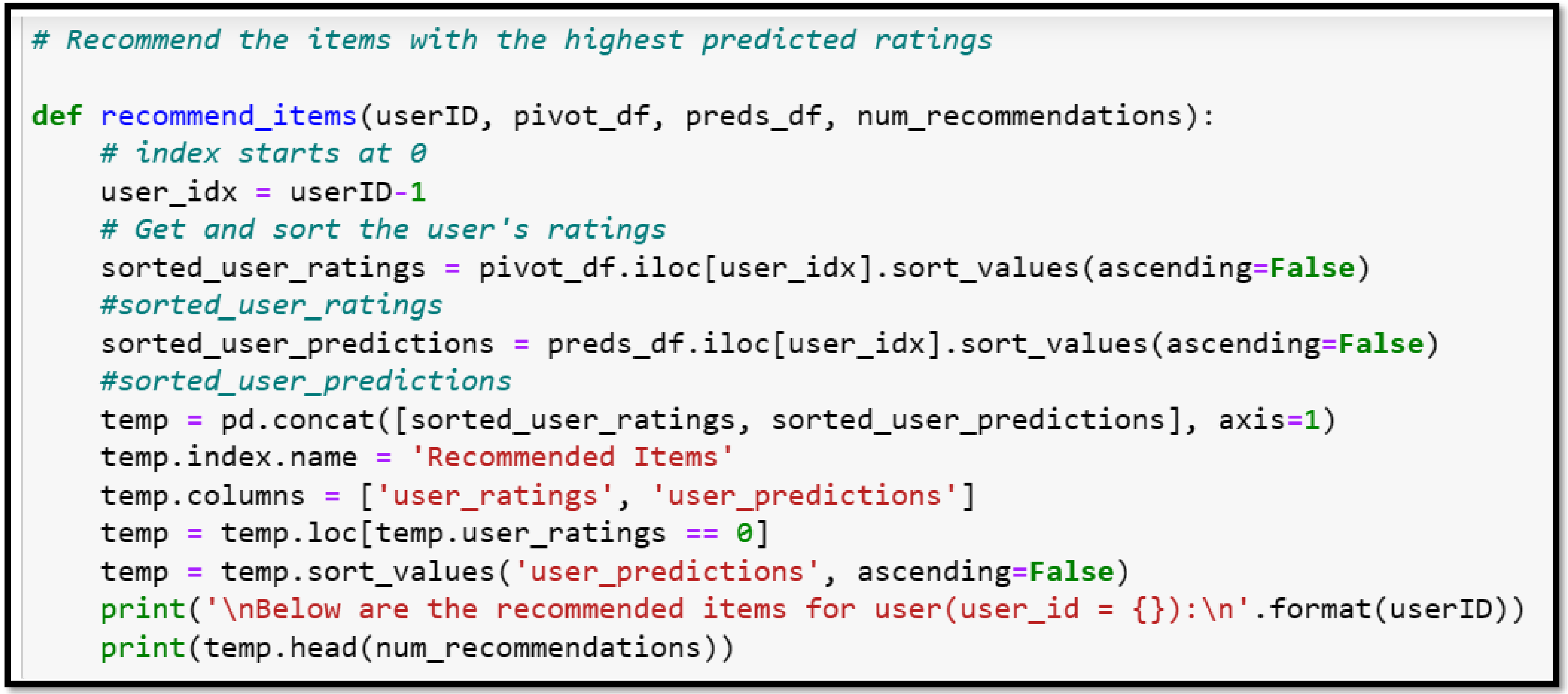
```python
print('Left singular matrix: \n',U)
```

```
Left singular matrix: 
 [[ 0.45238943  0.01774515 -0.13391437 -0.03542716 -0.06582399 -0.0987448
   0.04544846  0.16916418  0.22507639 -0.00676191]
 [ 0.03072919  0.02230867  0.01065019  0.02698556  0.01215515  0.02200431
   0.03118853 -0.01074048 -0.06771948 -0.99522082]
 [-0.13976932 -0.16727924  0.20786549 -0.07832176 -0.00391341 -0.0953826
  -0.04436339  0.07493014  0.12983155 -0.02666738]
 [ 0.39086227  0.05525848 -0.04546851 -0.00182618 -0.0231128   0.34816872
  -0.21347952  0.29462255  0.28496335 -0.00923125]
```

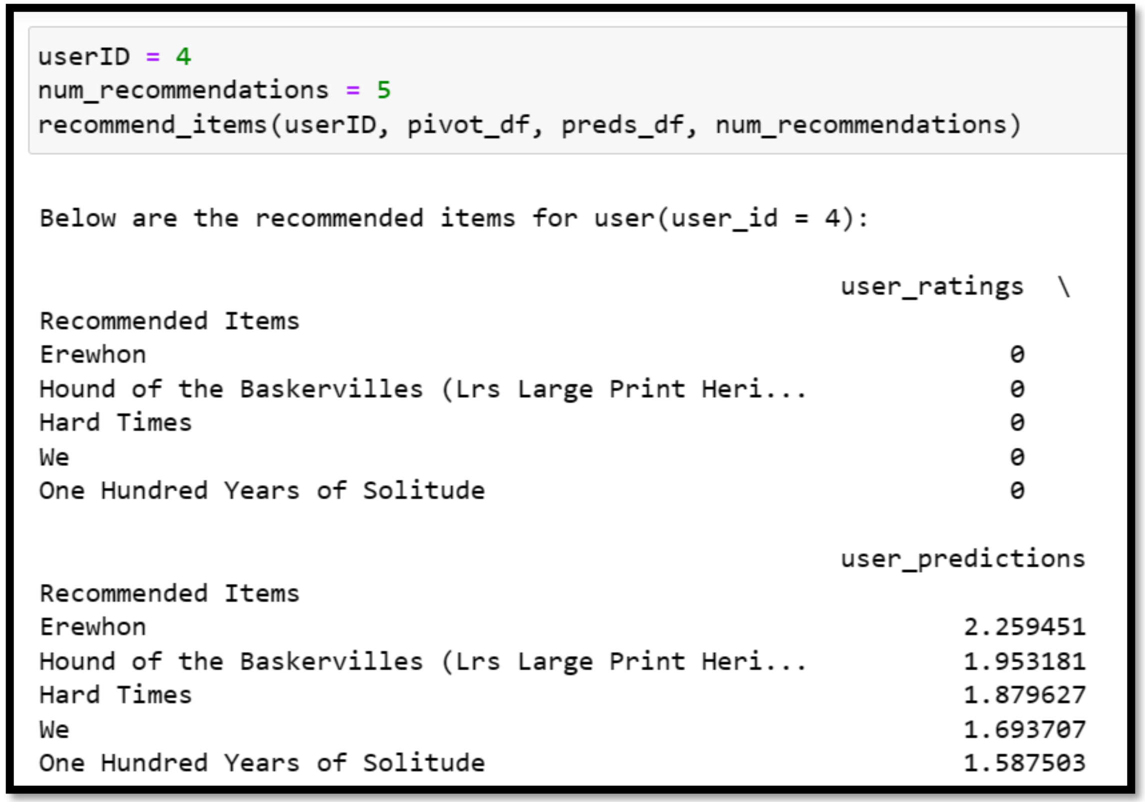


```python
# Construct diagonal array in SVD
sigma = np.diag(sigma)
print('Diagonal matrix: \n',sigma)
```

```
Diagonal matrix: 
 [[19.05272699  0.          0.          0.          0.          0.
   0.          0.          0.          0.        ]
 [ 0.         19.51615207  0.          0.          0.          0.
   0.          0.          0.          0.        ]
 [ 0.          0.         20.96306948  0.          0.          0.
   0.          0.          0.          0.        ]
 [ 0.          0.          0.         21.41487361  0.          0.
   0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.         22.91643874  0.
   0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.         23.19320619
   0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
  26.66007205  0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.         31.673648    0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.         34.16039841  0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.          0.         48.63008151]]
```

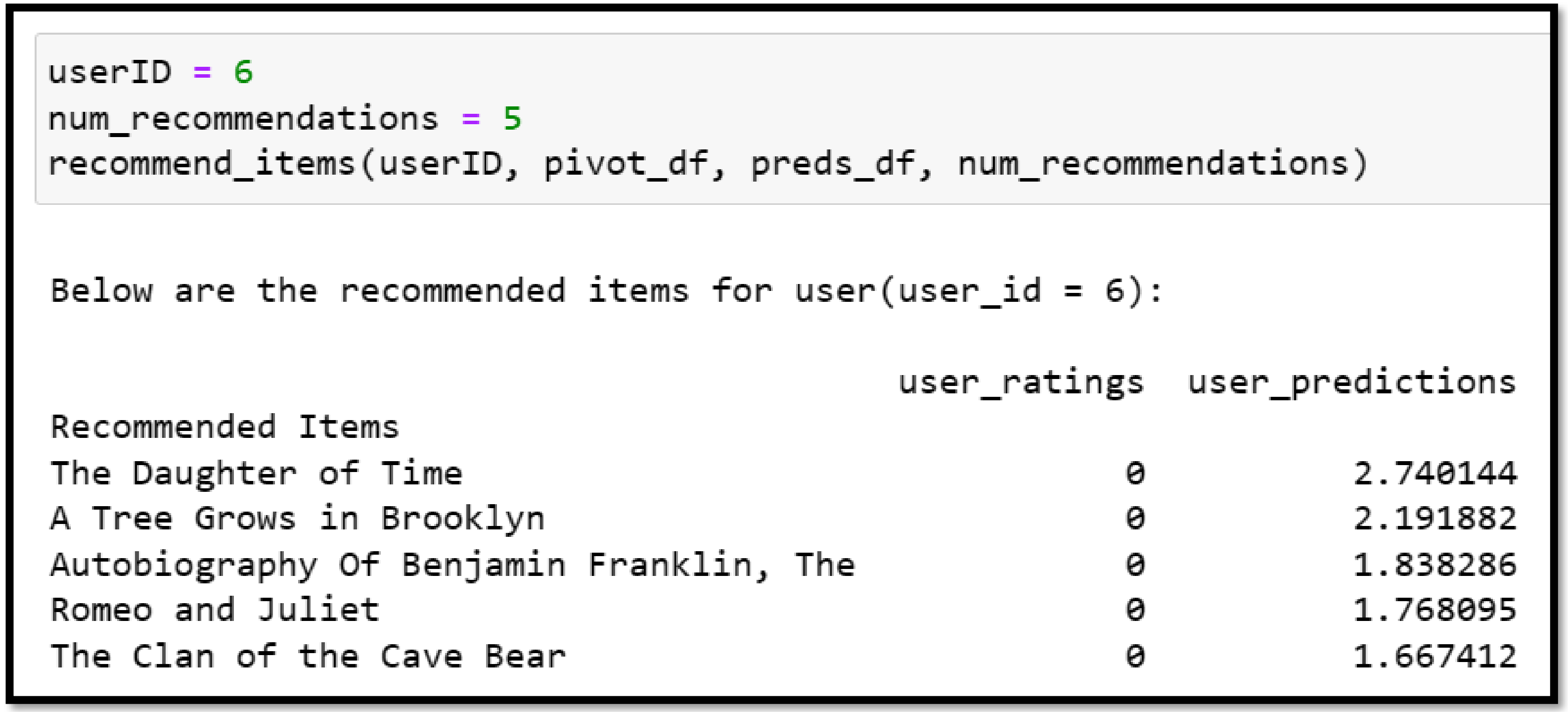


```python
# Recommend the items with the highest predicted ratings

def recommend_items(userID, pivot_df, preds_df, num_recommendations):
    # index starts at 0
    user_idx = userID-1
    # Get and sort the user's ratings
    sorted_user_ratings = pivot_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_ratings
    sorted_user_predictions = preds_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_predictions
    temp = pd.concat([sorted_user_ratings, sorted_user_predictions], axis=1)
    temp.index.name = 'Recommended Items'
    temp.columns = ['user_ratings', 'user_predictions']
    temp = temp.loc[temp.user_ratings == 0]
    temp = temp.sort_values('user_predictions', ascending=False)
    print('\nBelow are the recommended items for user(user_id = {}):\n'.format(userID))
    print(temp.head(num_recommendations))
```



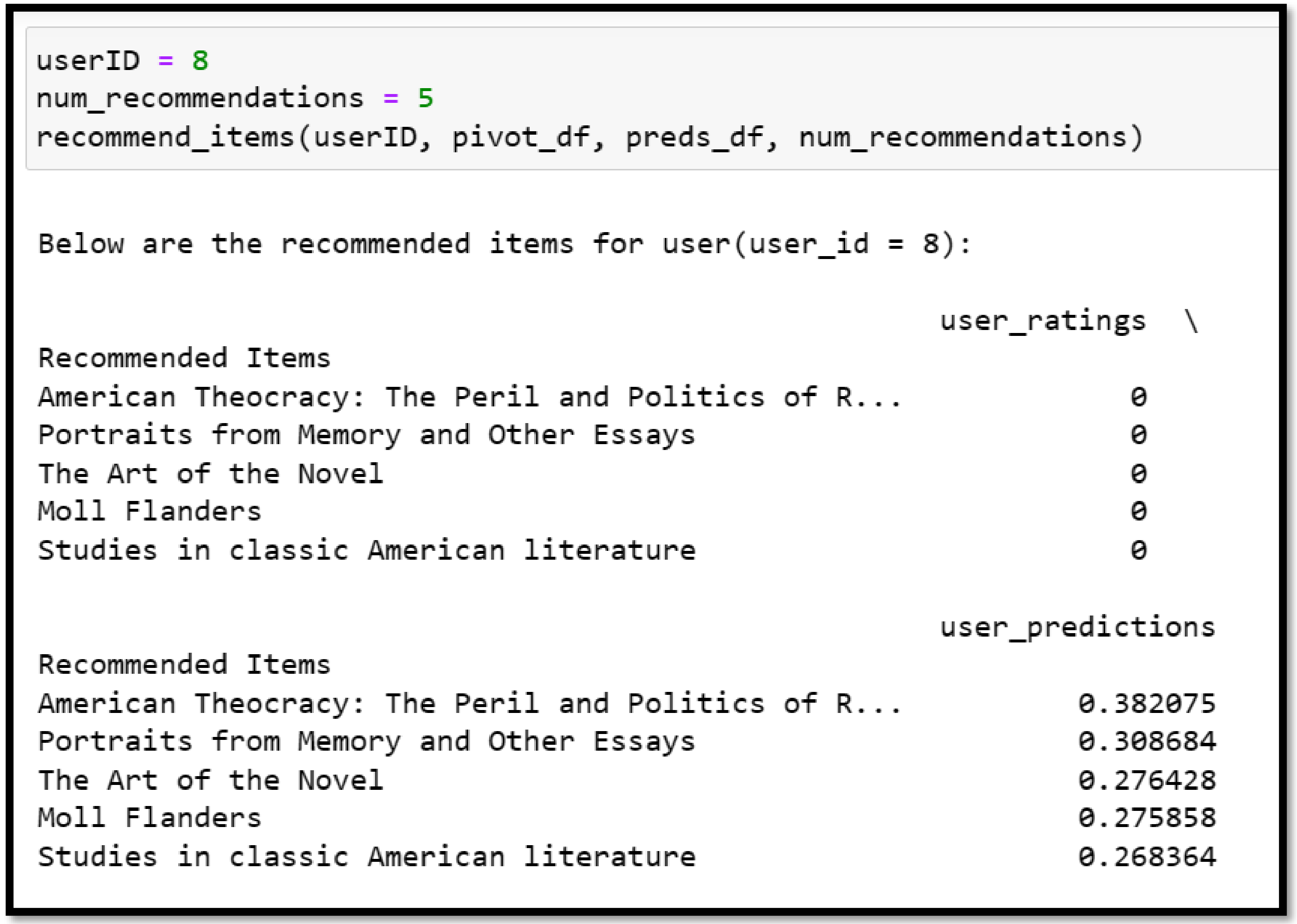
```
userID = 4
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
Below are the recommended items for user(user_id = 4):

                                                    user_ratings  \
Recommended Items
Erewhon                                                        0
Hound of the Baskervilles (Lrs Large Print Heri...             0
Hard Times                                                     0
We                                                             0
One Hundred Years of Solitude                                  0

                                                    user_predictions
Recommended Items
Erewhon                                                     2.259451
Hound of the Baskervilles (Lrs Large Print Heri...          1.953181
Hard Times                                                  1.879627
We                                                          1.693707
One Hundred Years of Solitude                               1.587503
```



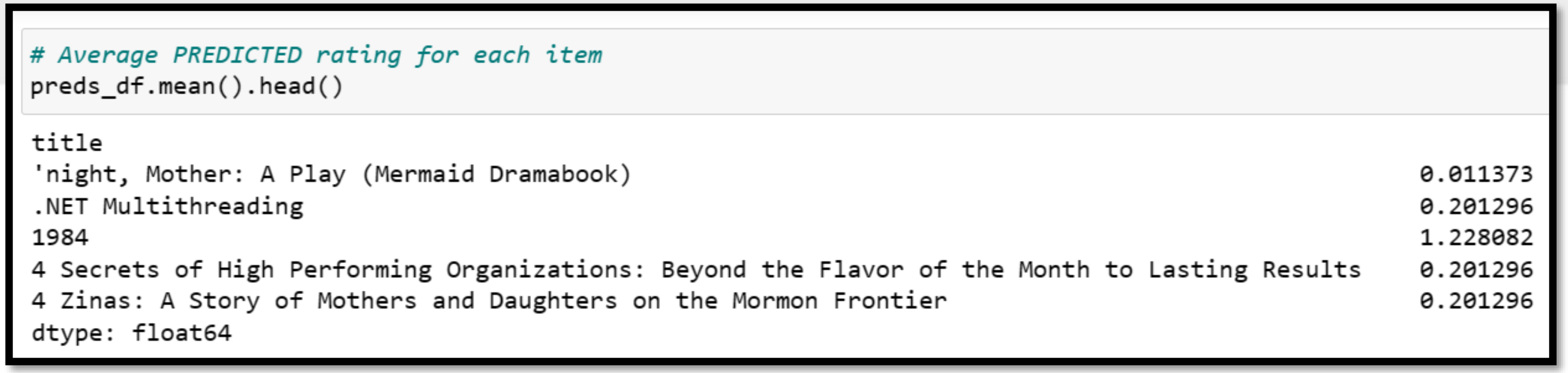
```
userID = 6
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
Below are the recommended items for user(user_id = 6):

                                         user_ratings  user_predictions
Recommended Items
The Daughter of Time                                0          2.740144
A Tree Grows in Brooklyn                            0          2.191882
Autobiography Of Benjamin Franklin, The             0          1.838286
Romeo and Juliet                                    0          1.768095
The Clan of the Cave Bear                           0          1.667412
```



```
userID = 8
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
Below are the recommended items for user(user_id = 8):

                                                   user_ratings  \
Recommended Items
American Theocracy: The Peril and Politics of R...             0
Portraits from Memory and Other Essays                         0
The Art of the Novel                                           0
Moll Flanders                                                  0
Studies in classic American literature                         0

                                                   user_predictions
Recommended Items
American Theocracy: The Peril and Politics of R...          0.382075
Portraits from Memory and Other Essays                      0.308684
The Art of the Novel                                        0.276428
Moll Flanders                                               0.275858
Studies in classic American literature                      0.268364
```


```
# Average PREDICTED rating for each item
preds_df.mean().head()
```

```
title
'night, Mother: A Play (Mermaid Dramabook)                                              0.011373
.NET Multithreading                                                                     0.201296
1984                                                                                    1.228082
4 Secrets of High Performing Organizations: Beyond the Flavor of the Month to Lasting Results    0.201296
4 Zinas: A Story of Mothers and Daughters on the Mormon Frontier                        0.201296
dtype: float64
```

Here are the explanations for the recommended items:

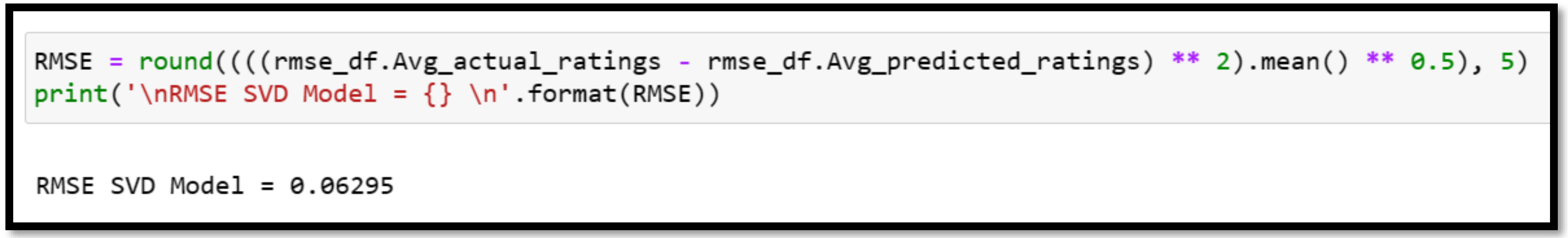
1.'night, Mother: A Play (Mermaid Dramabook): The predicted rating for this item is 0.011373. This suggests that the system believes the user has a very low preference or interest in this play.

2..NET Multithreading: The predicted rating for this item is 0.201296. This indicates a relatively low predicted preference or interest in this topic related to multithreading in the .NET framework.

3.1984: The predicted rating for this item is 1.228082. This suggests a relatively high predicted preference or interest in the novel "1984" by George Orwell.

4.4 Secrets of High Performing Organizations: Beyond the Flavor of the Month to Lasting Results: The predicted rating for this item is 0.201296. This indicates a relatively low predicted preference or interest in this book about high-performing organizations.

5.4 Zinas: A Story of Mothers and Daughters on the Mormon Frontier: The predicted rating for this item is 0.201296. This suggests a relatively low predicted preference or interest in this book about mothers and daughters in a Mormon frontier setting.



```
RMSE = round((((rmse_df.Avg_actual_ratings - rmse_df.Avg_predicted_ratings) ** 2).mean() ** 0.5), 5)
print('\nRMSE SVD Model = {} \n'.format(RMSE))
```

```
RMSE SVD Model = 0.06295
```

SVD (Singular Value Decomposition) model used in the collaborative filtering recommendation system achieved an RMSE (Root Mean Square Error) value of 0.06295. This indicates that the model has good accuracy in predicting the ratings of items in the dataset.

The RMSE value represents the average difference between the predicted ratings and the actual ratings. A lower RMSE suggests that the predicted ratings are closer to the actual ratings. In this case, the RMSE of 0.06295 indicates that, on average, the predicted ratings from the SVD model have a difference of approximately 0.06295 from the actual ratings.

Summarising insights

From the practical aspect of analyzing Amazon book reviews using the techniques mentioned, the following conclusions can be drawn:

1.Collaborative filtering, specifically matrix factorization with Singular Value Decomposition (SVD), is an effective approach for building recommendation systems based on user preferences and item associations. It allows for personalized recommendations by identifying latent factors that capture the underlying patterns in the data.

2.The preprocessing steps, such as data cleaning and transformation, are crucial for ensuring the quality and usability of the dataset. Removing duplicates, handling missing values, and converting the data into a suitable format (e.g., pivot table) are essential for accurate analysis and modeling.

3.By applying SVD, we can reduce the dimensionality of the user-item rating matrix and capture the most important factors that drive user preferences. The resulting low-rank approximation allows for efficient computation and provides a foundation for making personalized recommendations.

4.The evaluation metrics, such as Root Mean Squared Error (RMSE), provide a quantitative measure of the model's predictive accuracy. A lower RMSE indicates better performance in predicting user ratings and suggests that the model is capturing the underlying patterns well.

5.The recommended items for users, based on the SVD model, can help users discover new books that align with their interests. These recommendations can enhance the user experience, increase engagement, and potentially lead to higher sales and customer satisfaction.

6.The insights gained from the SVD model, such as the left singular matrix (U) and right singular matrix (V^T), provide valuable information about user preferences, item associations, and latent factors. This knowledge can be leveraged for targeted marketing, inventory management, and further understanding customer behavior.

Manisha on Amazon Book Recommendation
manisha@isdcglobal.org.uk
+91-7754837447