AIWR ASSIGNMENT 2

RECOMMENDER SYSTEM USING A CORPUS

NAME: Riya Jha

SRN: PES1UG20CS344

NAME: Akanksha Kurapati

SRN: PES1UG20CS228

NAME: Ananya Menon

SRN: PES1UG20CS043

NAME: Hamsini S

SRN: PES1UG20CS157

Problem Statement

An example of a recommendation system that employs data analysis and machine learning algorithms to give people personalised book recommendations is a book recommendation system. Online book retailers, libraries, and other businesses that offer their clients book-related services can use these systems.

Introduction

A book recommendation system's objective is to offer users recommendations that are catered to their tastes and interests. This can be done by looking at user information including past purchases, browsing patterns, and book ratings. For more accurate recommendations, the system can additionally take into account additional elements like genre, author, and publication date.

Dataset

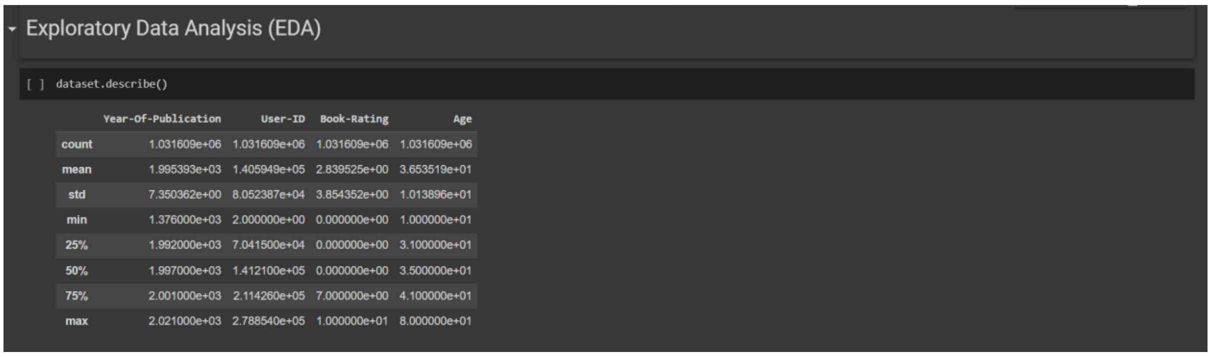
Link to the dataset - http://www2.informatik.uni-freiburg.de/~cziegler/BX/

The Book-Crossing dataset is a collection of data about the book ratings provided by users in the Book-Crossing community. This dataset was collected by Cai-Nicolas Ziegler in August and September 2004, over a period of four weeks, with the permission of Ron Hornbaker, the CTO of Humankind Systems.

The dataset includes information about 278,858 users, who have been anonymized, but with demographic information such as age, gender, and location. Additionally, the dataset provides 1,149,780 ratings, which are either explicit (i.e., the user has given a rating on a scale of 1 to 10) or implicit (i.e., the rating is inferred from the user's behaviour, such as the frequency of reading a particular book).

Overall, the dataset contains information about 271,379 books that have been rated by the users in the Book-Crossing community. This dataset can be used for various purposes, such as building recommendation systems, understanding user behaviour, and analysing reading habits across different demographics.

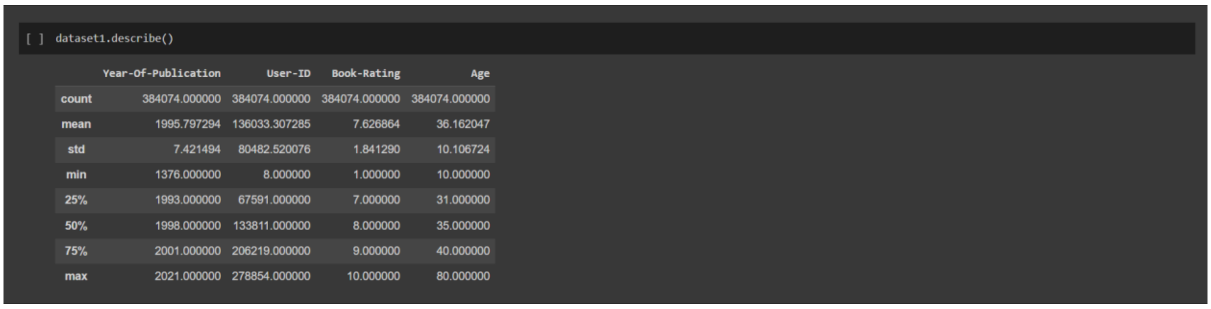
EDA



Exploratory Data Analysis (EDA)

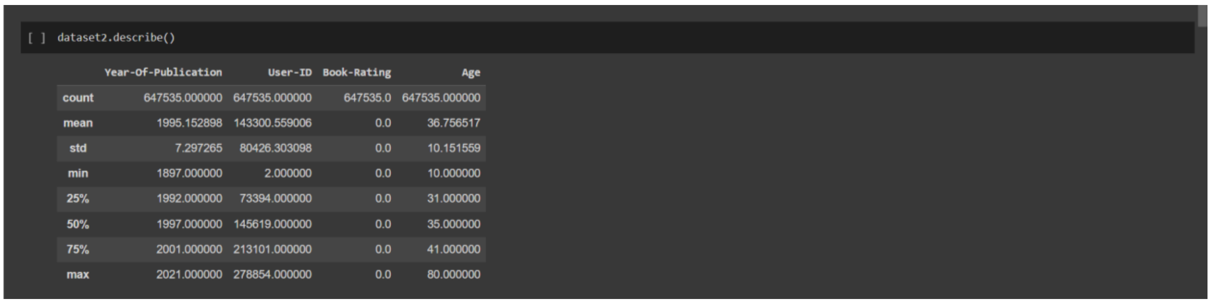
```
[ ] dataset.describe()
```

	Year-Of-Publication	User-ID	Book-Rating	Age
count	1.031609e+06	1.031609e+06	1.031609e+06	1.031609e+06
mean	1.995393e+03	1.405949e+05	2.839525e+00	3.653519e+01
std	7.350362e+00	8.052387e+04	3.854352e+00	1.013896e+01
min	1.376000e+03	2.000000e+00	0.000000e+00	1.000000e+01
25%	1.992000e+03	7.041500e+04	0.000000e+00	3.100000e+01
50%	1.997000e+03	1.412100e+05	0.000000e+00	3.500000e+01
75%	2.001000e+03	2.114260e+05	7.000000e+00	4.100000e+01
max	2.021000e+03	2.788540e+05	1.000000e+01	8.000000e+01



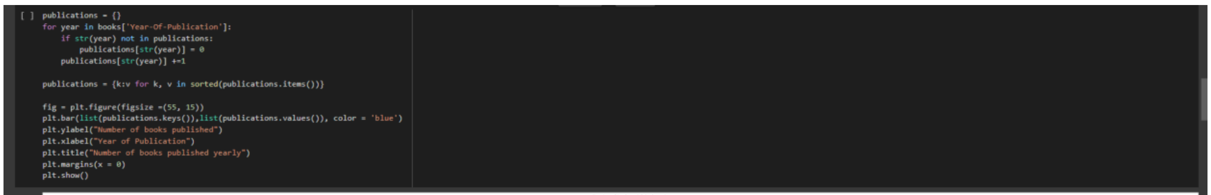
```
[ ] dataset1.describe()
```

	Year-Of-Publication	User-ID	Book-Rating	Age
count	384074.000000	384074.000000	384074.000000	384074.000000
mean	1995.797294	136033.307285	7.626864	36.162047
std	7.421494	80482.520076	1.841290	10.106724
min	1376.000000	8.000000	1.000000	10.000000
25%	1993.000000	67591.000000	7.000000	31.000000
50%	1998.000000	133811.000000	8.000000	35.000000
75%	2001.000000	206219.000000	9.000000	40.000000
max	2021.000000	278854.000000	10.000000	80.000000

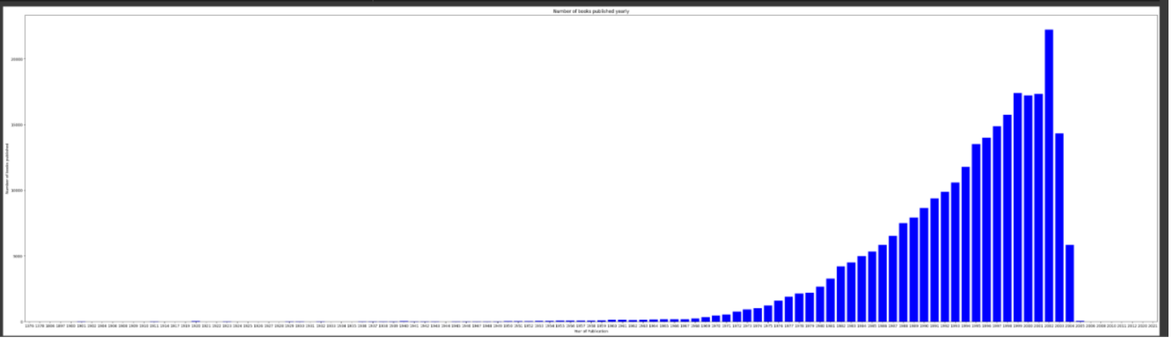


```
[ ] dataset2.describe()
```

	Year-Of-Publication	User-ID	Book-Rating	Age
count	647535.000000	647535.000000	647535.0	647535.000000
mean	1995.152898	143300.559006	0.0	36.756517
std	7.297265	80426.303098	0.0	10.151559
min	1897.000000	2.000000	0.0	10.000000
25%	1992.000000	73394.000000	0.0	31.000000
50%	1997.000000	145619.000000	0.0	35.000000
75%	2001.000000	213101.000000	0.0	41.000000
max	2021.000000	278854.000000	0.0	80.000000

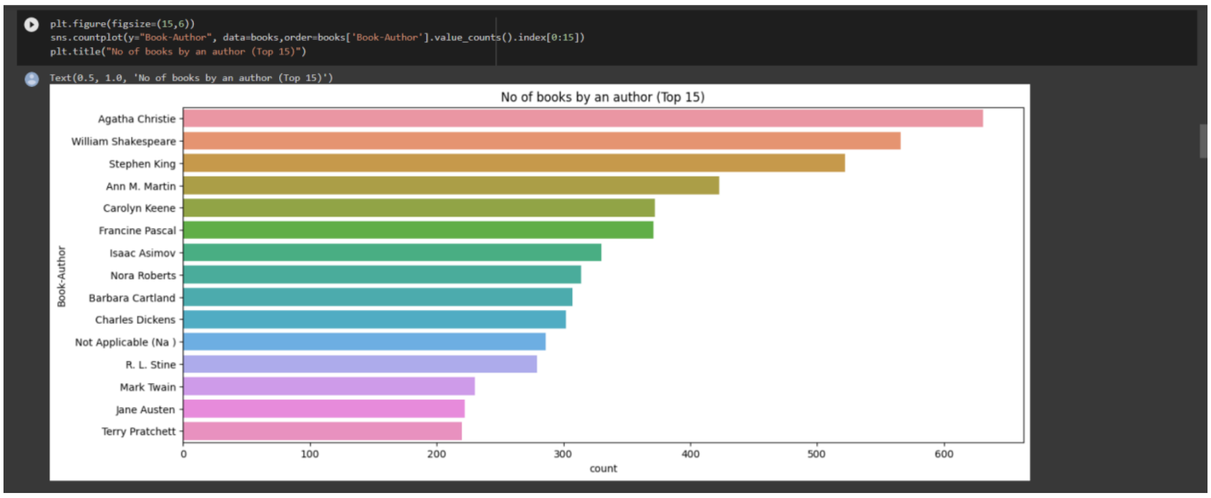


```
[ ] publications = {}
    for year in books['Year-Of-Publication']:
        if str(year) not in publications:
            publications[str(year)] = 0
        publications[str(year)] +=1

    publications = {k:v for k, v in sorted(publications.items())}

    fig = plt.figure(figsize =(55, 15))
    plt.bar(list(publications.keys()),list(publications.values()), color = 'blue')
    plt.ylabel("Number of books published")
    plt.xlabel("Year of Publication")
    plt.title("Number of books published yearly")
    plt.margins(x = 0)
    plt.show()
```



- The first four lines of code initialize an empty dictionary called publications and loop through each value in the 'Year-Of-Publication' column of the books Data Frame.
- For each year, the code checks if the year is already a key in the publications dictionary, and if not, it adds the year as a key and sets the value to zero. Then, it increments the count of books published in that year by one.

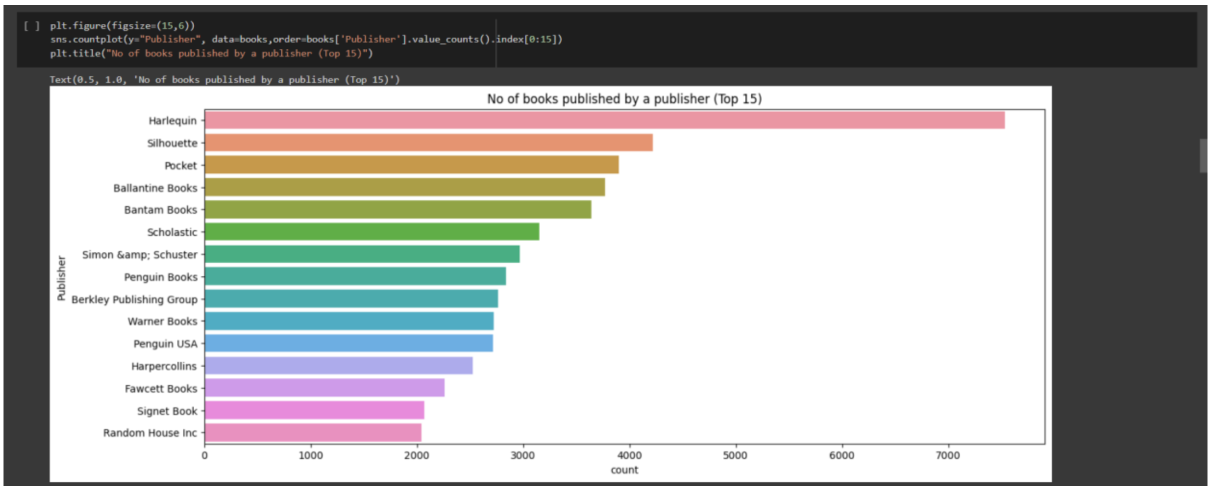
- The fifth line of code sorts the publications dictionary by year, in ascending order.
- The next six lines of code create the actual bar chart using the matplotlib library. The fig variable initializes a figure with a specified size, and the plt.bar function is called with the list of keys (years) and values (counts) from the publications dictionary.
- The color of the bars is set to blue using the color parameter. The plt.ylabel, plt.xlabel, and plt.title functions are used to label the axes and title of the chart. Finally, plt.margins sets the margins of the x-axis to zero to remove any extra space around the edges of the chart, and plt.show displays the chart.



```
plt.figure(figsize=(15,6))
sns.countplot(y="Book-Author", data=books,order=books['Book-Author'].value_counts().index[0:15])
plt.title("No of books by an author (Top 15)")
```

```
Text(0.5, 1.0, 'No of books by an author (Top 15)')
```

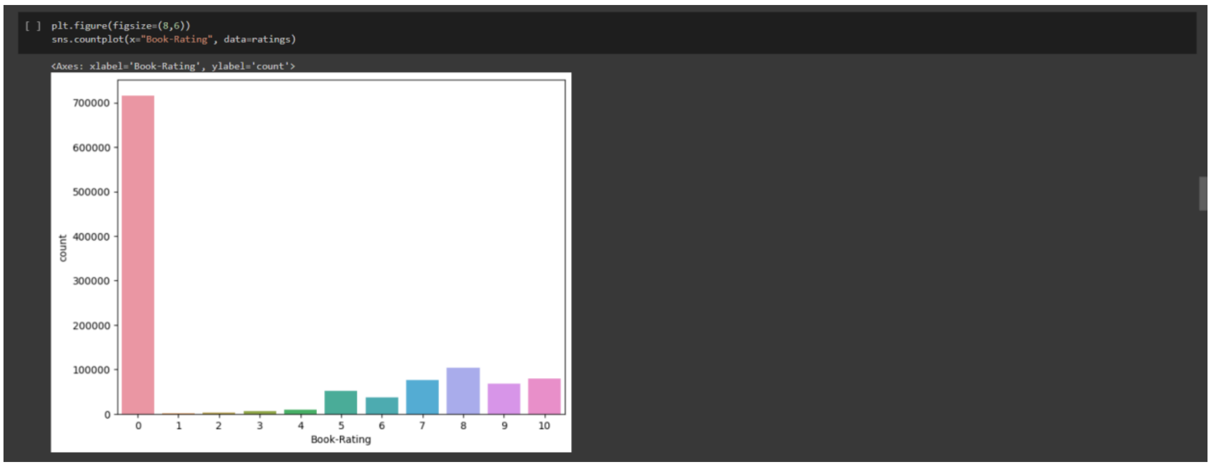
- The first line of code sets the figure size of the plot to be 15 inches wide and 6 inches tall using the figsize parameter.
- The second line of code creates the actual count plot using the sns.countplot function from the Seaborn library. The y parameter specifies the variable to be plotted on the y-axis, which in this case is the "Book-Author" column of the books Data Frame. The data parameter specifies the Data Frame to be used for the plot.
- The order parameter sorts the authors by the number of books they have written in descending order (i.e., the most prolific authors are shown first) using the value_counts method of the books['Book-Author'] Series. The index parameter selects only the top 15 authors from the sorted list.
- The third line of code sets the title of the plot to "No of books by an author (Top 15)" using the plt.title function.



```
[ ] plt.figure(figsize=(15,6))
    sns.countplot(y="Publisher", data=books,order=books['Publisher'].value_counts().index[0:15])
    plt.title("No of books published by a publisher (Top 15)")
```

```
Text(0.5, 1.0, 'No of books published by a publisher (Top 15)')
```

- The first line of code sets the figure size of the plot to be 15 inches wide and 6 inches tall using the figsize parameter.
- The second line of code creates the actual count plot using the sns.countplot function from the Seaborn library. The y parameter specifies the variable to be plotted on the y-axis, which in this case is the "Publisher" column of the books Data Frame.
- The data parameter specifies the Data Frame to be used for the plot. The order parameter sorts the publishers by the number of books they have published in descending order (i.e., the most prolific publishers are shown first) using the value_counts method of the books ['Publisher'] Series. The index parameter selects only the top 15 publishers from the sorted list.
- The third line of code sets the title of the plot to "No of books published by a publisher (Top 15)" using the plt.title function

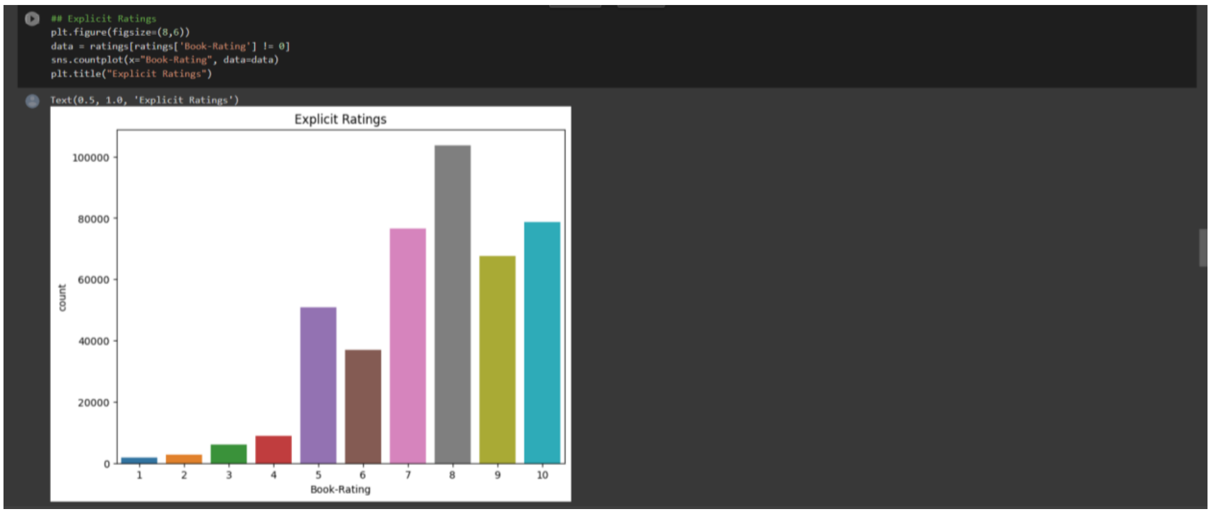


```
[ ] plt.figure(figsize=(8,6))
    sns.countplot(x="Book-Rating", data=ratings)
```

```
<Axes: xlabel='Book-Rating', ylabel='count'>
```

- The first line of code sets the figure size of the plot to be 8 inches wide and 6 inches tall using the figsize parameter.
- The second line of code creates the actual count plot using the sns.countplot function from the Seaborn library. The x parameter specifies the variable to be plotted on the x-axis, which in this case is the "Book-Rating" column of the ratings Data Frame. The data parameter specifies the Data Frame to be used for the plot.

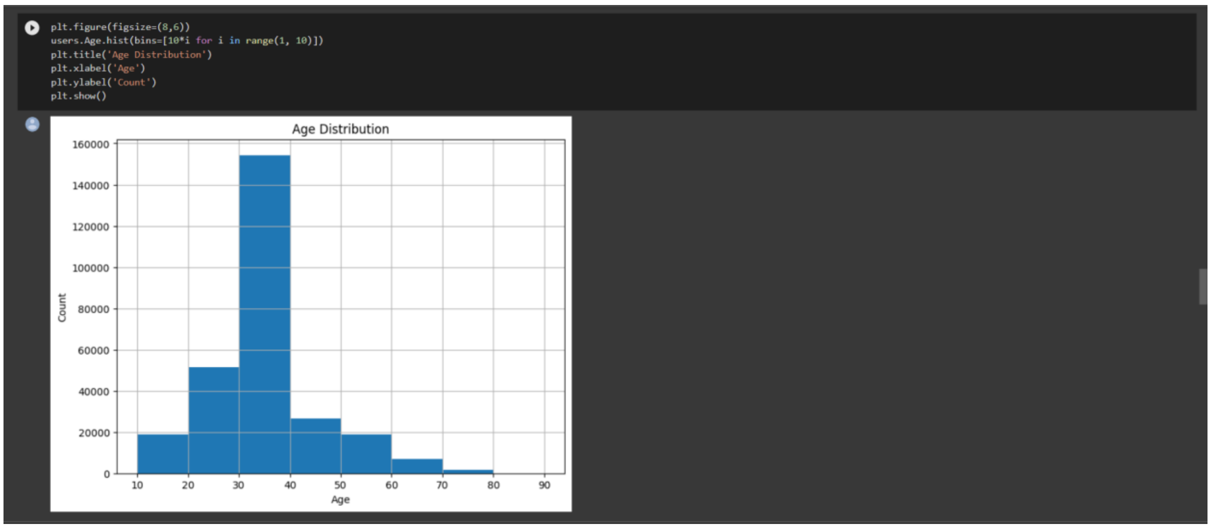
- The resulting plot shows a bar for each possible rating value, with the height of each bar indicating how many times that rating value appears in the ratings dataset.



```
## Explicit Ratings
plt.figure(figsize=(8,6))
data = ratings[ratings['Book-Rating'] != 0]
sns.countplot(x="Book-Rating", data=data)
plt.title("Explicit Ratings")
```

```
Text(0.5, 1.0, 'Explicit Ratings')
```

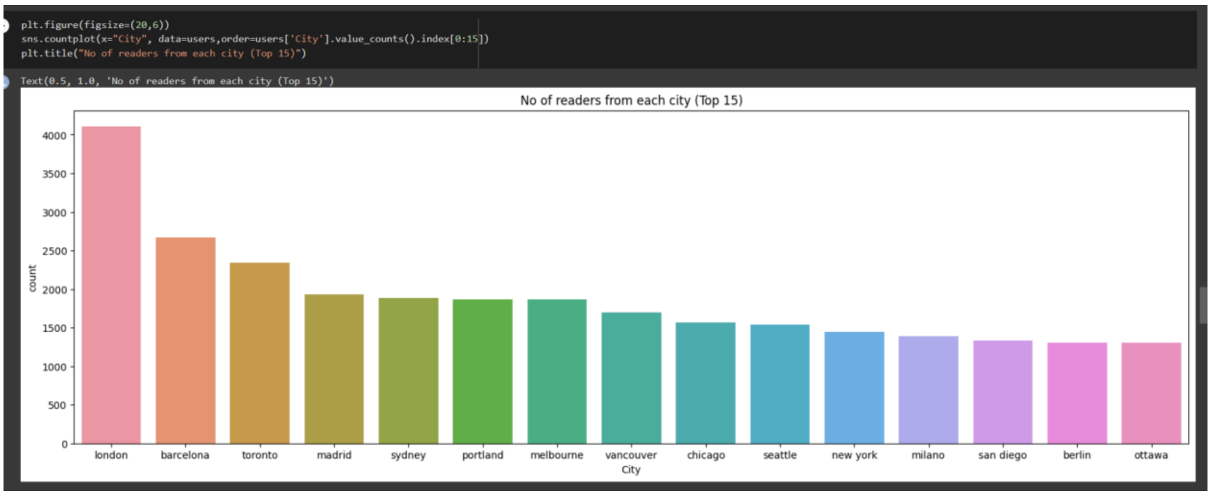
- The first line of code sets the figure size of the plot to be 8 inches wide and 6 inches tall using the figsize parameter.
- The second line of code creates a new Data Frame called data that contains only the rows from the ratings Data Frame where the "Book-Rating" column is not equal to zero. This is done using boolean indexing, where ratings['Book-Rating'] != 0 returns a boolean mask that is True for rows where the rating is not equal to zero, and False otherwise.
- The third line of code creates the actual count plot using the sns.countplot function from the Seaborn library. The x parameter specifies the variable to be plotted on the x-axis, which in this case is the "Book-Rating" column of the data Data Frame. The data parameter specifies the Data Frame to be used for the plot.
- The fourth line of code sets the title of the plot to "Explicit Ratings" using the plt.title function.



```
plt.figure(figsize=(8,6))
users.Age.hist(bins=[10*i for i in range(1, 10)])
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

- The first line of code sets the figure size of the plot to be 8 inches wide and 6 inches tall using the figsize parameter.

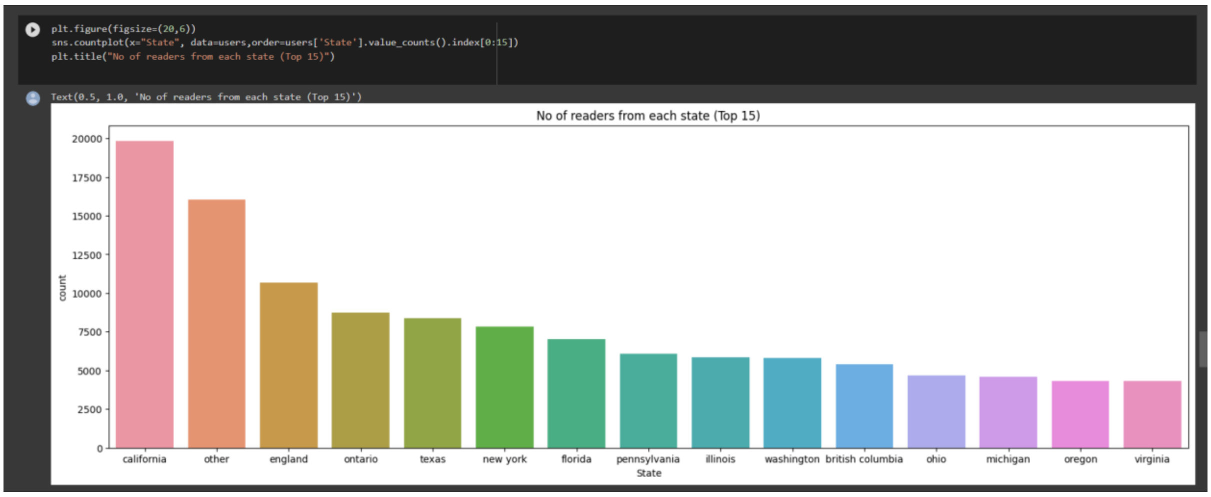
- The second line of code calls the hist method on the Age column of the users DataFrame. This creates a histogram of the ages with the specified bins. In this case, the bins parameter specifies a list of bin edges, where each bin is 10 years wide. For example, the first bin goes from 10 to 19 years old, the second bin goes from 20 to 29 years old, and so on.
- The third line of code sets the title of the plot to "Age Distribution" using the plt.title function.
- The fourth line of code sets the label of the x-axis to "Age" using the plt.xlabel function.
- The fifth line of code sets the label of the y-axis to "Count" using the plt.ylabel function.
- The final line of code displays the plot using the plt.show function.



```
plt.figure(figsize=(20,6))
sns.countplot(x="City", data=users,order=users['City'].value_counts().index[0:15])
plt.title("No of readers from each city (Top 15)")
```

```
Text(0.5, 1.0, 'No of readers from each city (Top 15)')
```

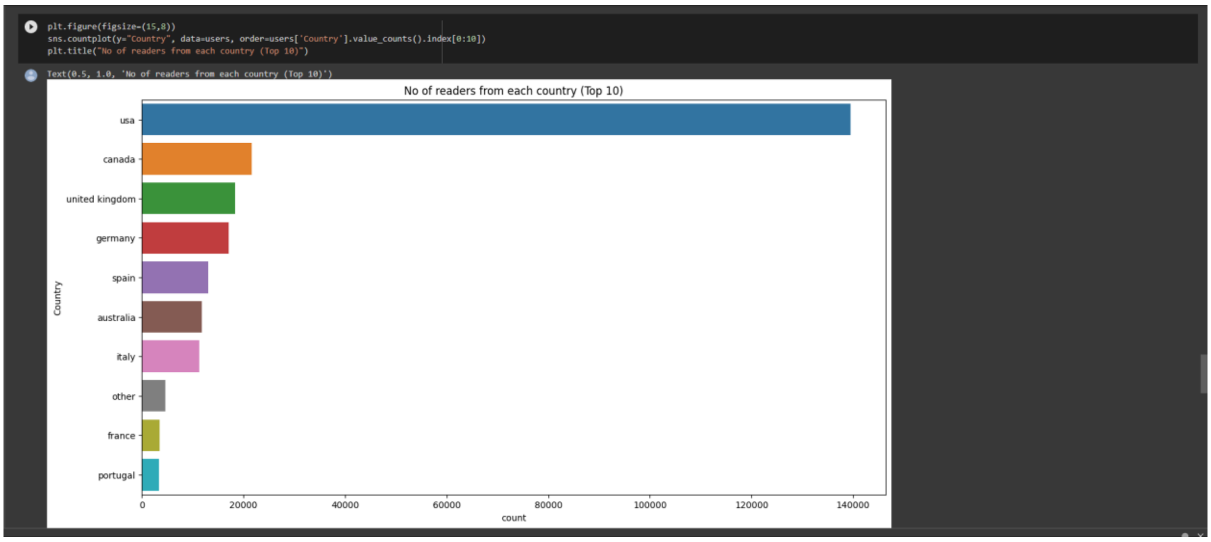
- The code is setting the size of the figure to be 20 units wide and 6 units tall using the figsize parameter in the plt.figure() function call.
- The sns.countplot() function is then used to create the bar chart. The x parameter specifies the column in the dataset that contains the city names. The data parameter specifies the dataset being used. The order parameter is setting the order of the bars to be plotted based on the frequency of cities in the dataset. Specifically, the top 15 cities with the most readers are being plotted.
- Finally, the plt.title() function is being used to set the title of the chart to "No of readers from each city (Top 15)".



```
plt.figure(figsize=(20,6))
sns.countplot(x="State", data=users,order=users['State'].value_counts().index[0:15])
plt.title("No of readers from each state (Top 15)")
```

```
Text(0.5, 1.0, 'No of readers from each state (Top 15)')
```

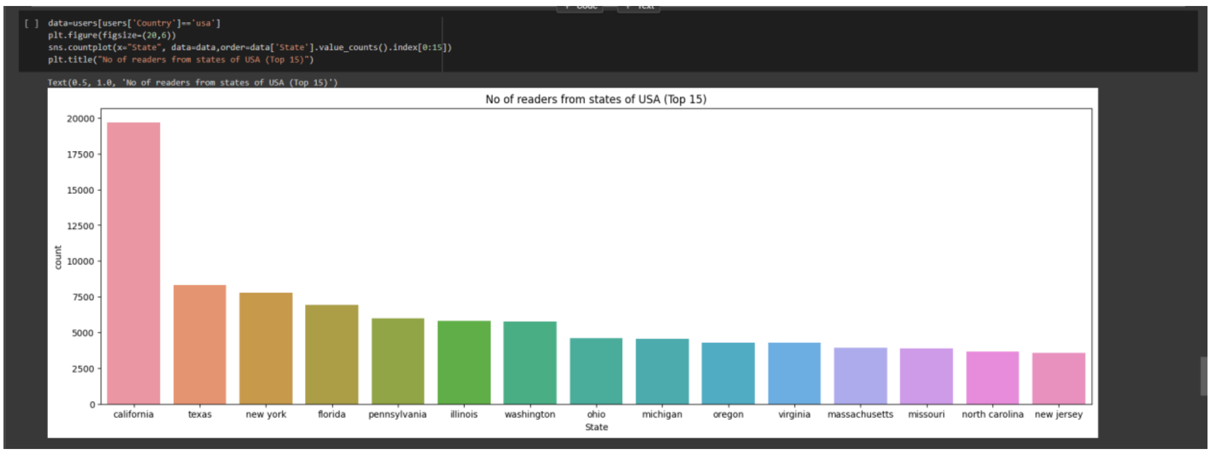
- The code is setting the size of the figure to be 20 units wide and 6 units tall using the figsize parameter in the plt.figure() function call.
- The sns.countplot() function is then used to create the bar chart. The x parameter specifies the column in the dataset that contains the city names. The data parameter specifies the dataset being used. The order parameter is setting the order of the bars to be plotted based on the frequency of cities in the dataset. Specifically, the top 15 states with the most readers are being plotted.
- Finally, the plt.title() function is being used to set the title of the chart to "No of readers from each state (Top 15)".



```
plt.figure(figsize=(15,8))
sns.countplot(y="Country", data=users, order=users['Country'].value_counts().index[0:10])
plt.title("No of readers from each country (Top 10)")
```

```
Text(0.5, 1.0, 'No of readers from each country (Top 10)')
```

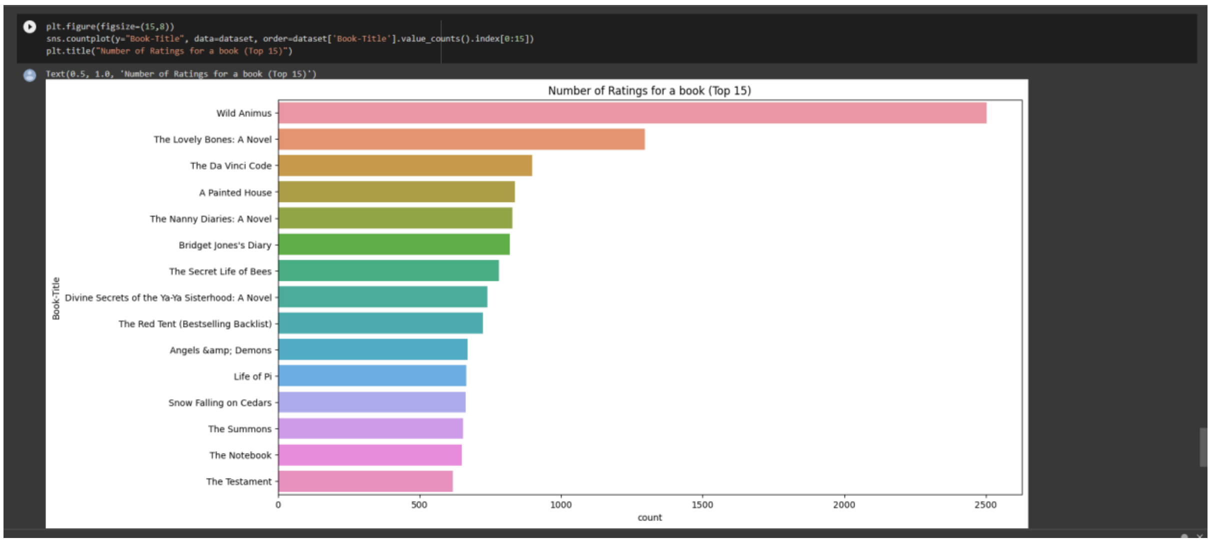
- The code is creating a bar chart that is 15 inches wide and 8 inches tall using the figure() method.
- The countplot() method is used to create a bar chart of the "Country" column in the "users" dataset. The "order" parameter is being used to specify the order in which the countries are displayed on the y-axis of the bar chart. Specifically, it is ordering the countries by the number of readers in each country, from highest to lowest, and then selecting only the top 10 countries using index[0:10].
- The title() method is used to set the title of the chart to "No of readers from each country (Top 10)".



```
data=users[users['Country']=='usa']
plt.figure(figsize=(20,6))
sns.countplot(x="State", data=data,order=data['State'].value_counts().index[0:15])
plt.title("No of readers from states of USA (Top 15)")
```

```
Text(0.5, 1.0, 'No of readers from states of USA (Top 15)')
```

- The code is selecting a subset of the "users" dataset where the "Country" column is equal to "usa". The resulting subset of the data is stored in a variable called "data".
- After that, the code is using Seaborn to create a bar chart that shows the number of readers from each state in the "data" subset of the "users" dataset. The countplot() method is used to create the bar chart with the "State" column in the "data" subset of the "users" dataset.
- The "order" parameter is being used to specify the order in which the states are displayed on the x-axis of the bar chart. Specifically, it is ordering the states by the number of readers in each state, from highest to lowest, and then selecting only the top 15 states using index[0:15].
- The title() method is used to set the title of the chart to "No of readers from states of USA (Top 15)".



```
plt.figure(figsize=(15,8))
sns.countplot(y="Book-Title", data=dataset, order=dataset['Book-Title'].value_counts().index[0:15])
plt.title("Number of Ratings for a book (Top 15)")
```

```
Text(0.5, 1.0, 'Number of Ratings for a book (Top 15)')
```

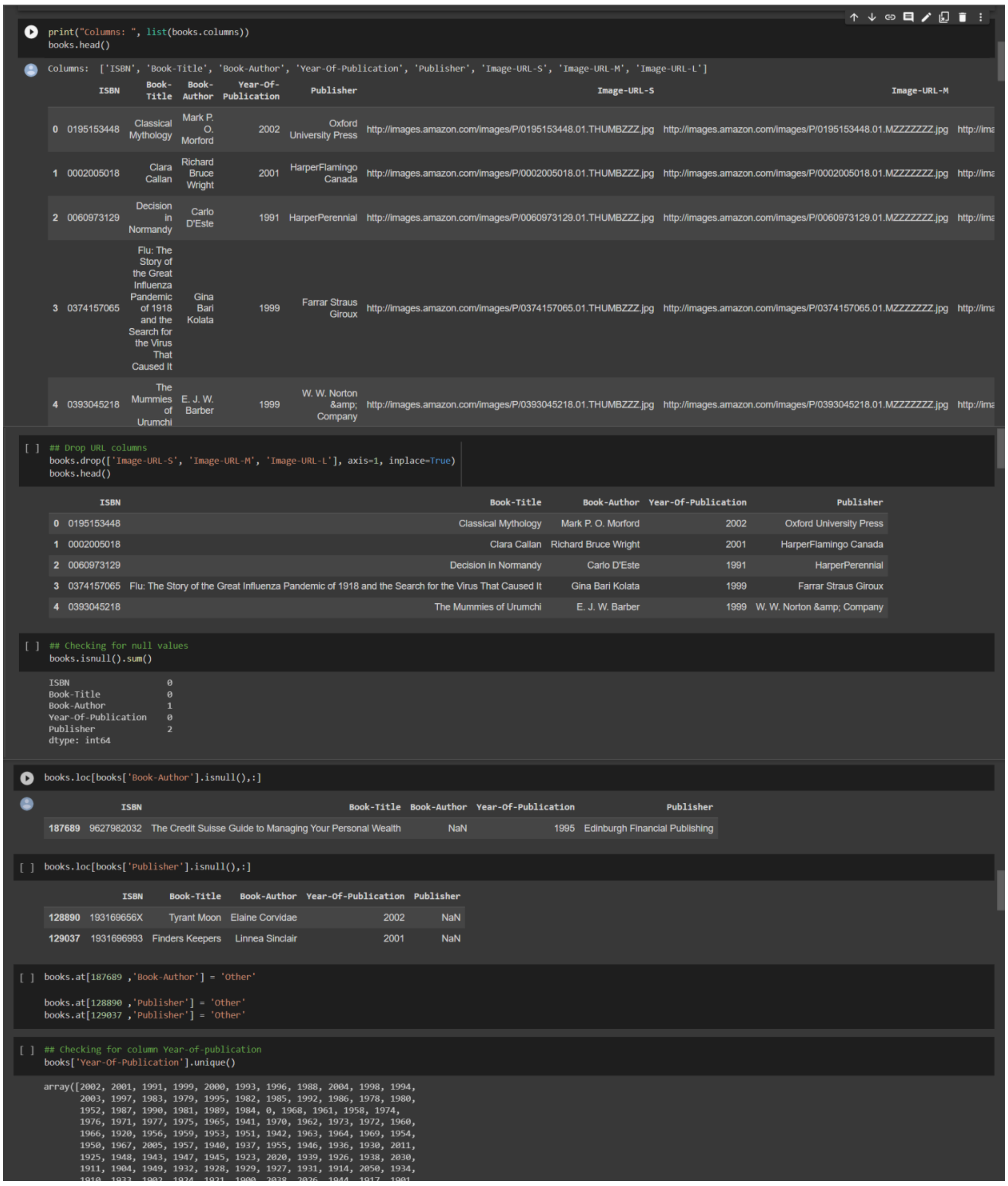
- The code is creating a bar chart that shows the number of ratings for each book in a dataset called "dataset". The code is creating a bar chart that is 15 inches wide and 8 inches tall using the figure() method.

- The countplot() method is used to create a bar chart of the "Book-Title" column in the "dataset" dataset. The "order" parameter is being used to specify the order in which the books are displayed on the y-axis of the bar chart. Specifically, it is ordering the books by the number of ratings for each book, from highest to lowest, and then selecting only the top 15 books using index[0:15].

- The title() method is used to set the title of the chart to "Number of Ratings for a book (Top 15)"

Pre-Processing

Books dataset:



```
print("Columns: ", list(books.columns))
books.head()
```

```
Columns:  ['ISBN', 'Book-Title', 'Book-Author', 'Year-Of-Publication', 'Publisher', 'Image-URL-S', 'Image-URL-M', 'Image-URL-L']
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0195153448.01.MZZZZZZZ.jpg	http://ima
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0002005018.01.MZZZZZZZ.jpg	http://ima
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0060973129.01.MZZZZZZZ.jpg	http://ima
3	0374157065	Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused It	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0374157065.01.MZZZZZZZ.jpg	http://ima
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton &amp; Company	http://images.amazon.com/images/P/0393045218.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0393045218.01.MZZZZZZZ.jpg	http://ima

```
[ ] ## Drop URL columns
    books.drop(['Image-URL-S', 'Image-URL-M', 'Image-URL-L'], axis=1, inplace=True)
    books.head()
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	0374157065	Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused It	Gina Bari Kolata	1999	Farrar Straus Giroux
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton &amp; Company

```
[ ] ## Checking for null values
    books.isnull().sum()
```

```
ISBN                   0
Book-Title             0
Book-Author            1
Year-Of-Publication    0
Publisher              2
dtype: int64
```

```
books.loc[books['Book-Author'].isnull(),:]
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
187689	9627982032	The Credit Suisse Guide to Managing Your Personal Wealth	NaN	1995	Edinburgh Financial Publishing

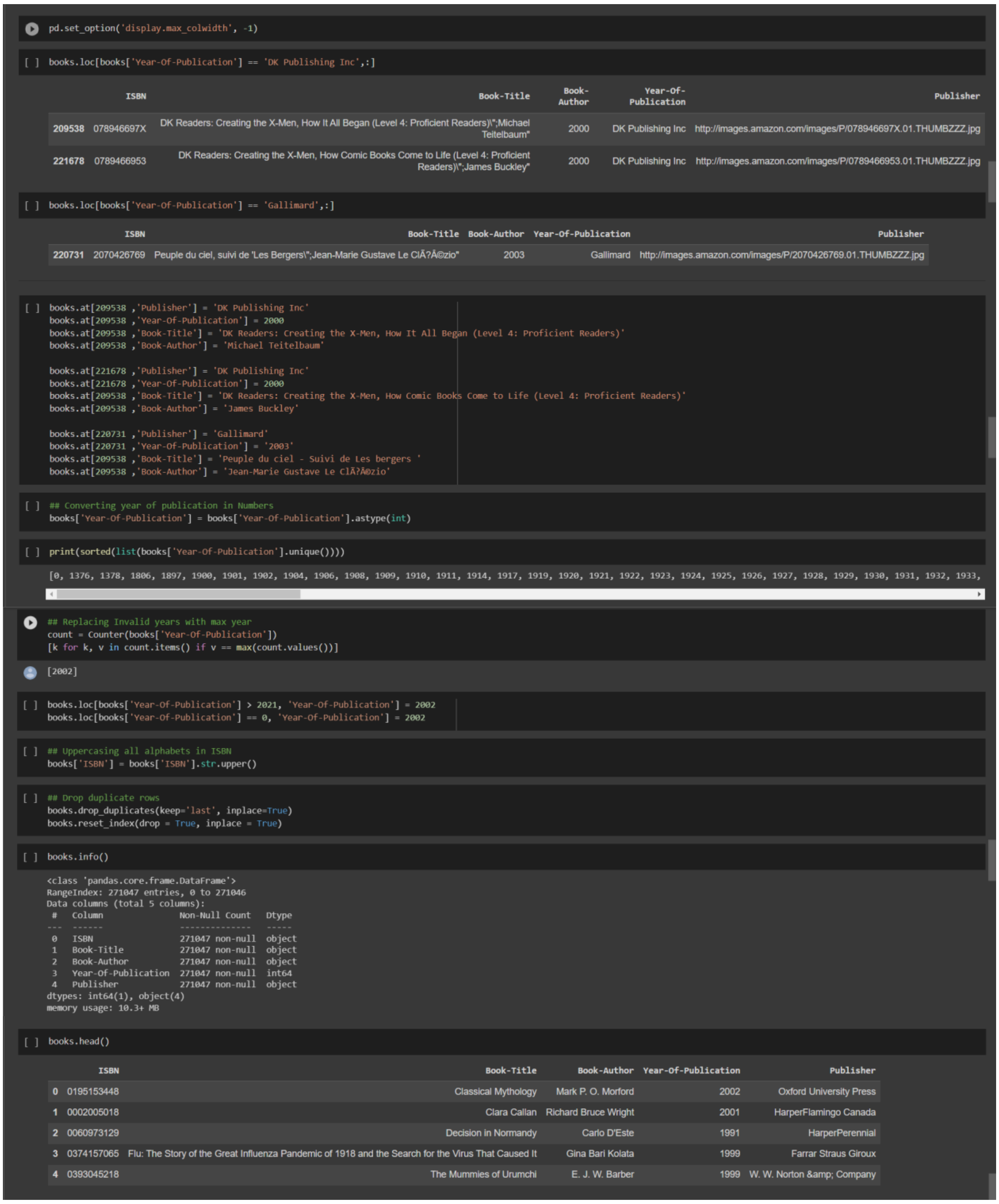
```
[ ] books.loc[books['Publisher'].isnull(),:]
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
128890	193169656X	Tyrant Moon	Elaine Corvidae	2002	NaN
129037	1931696993	Finders Keepers	Linnea Sinclair	2001	NaN

```
[ ] books.at[187689 ,'Book-Author'] = 'Other'

    books.at[128890 ,'Publisher'] = 'Other'
    books.at[129037 ,'Publisher'] = 'Other'
```

```
[ ] ## Checking for column Year-of-publication
    books['Year-Of-Publication'].unique()
```

```
array([2002, 2001, 1991, 1999, 2000, 1993, 1996, 1988, 2004, 1998, 1994,
       2003, 1997, 1983, 1979, 1995, 1982, 1985, 1992, 1986, 1978, 1980,
       1952, 1987, 1990, 1981, 1989, 1984, 0, 1968, 1961, 1958, 1974,
       1976, 1971, 1977, 1975, 1965, 1941, 1970, 1962, 1973, 1972, 1960,
       1966, 1920, 1956, 1959, 1953, 1951, 1942, 1963, 1964, 1969, 1954,
       1950, 1967, 2005, 1957, 1940, 1937, 1955, 1946, 1936, 1930, 2011,
       1925, 1948, 1943, 1947, 1945, 1923, 2020, 1939, 1926, 1938, 2030,
       1911, 1904, 1949, 1932, 1928, 1929, 1927, 1931, 1914, 2050, 1934,
       1910, 1933, 1902, 1924, 1921, 1900, 2038, 2026, 1944, 1917, 1901
```



```
pd.set_option('display.max_colwidth', -1)
```

```
books.loc[books['Year-Of-Publication'] == 'DK Publishing Inc',:]
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
209538	078946697X	DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)\";Michael Teitelbaum"	2000	DK Publishing Inc	http://images.amazon.com/images/P/078946697X.01.THUMBZZZ.jpg
221678	0789466953	DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)\";James Buckley"	2000	DK Publishing Inc	http://images.amazon.com/images/P/0789466953.01.THUMBZZZ.jpg

```
books.loc[books['Year-Of-Publication'] == 'Gallimard',:]
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
220731	2070426769	Peuple du ciel, suivi de 'Les Bergers\";Jean-Marie Gustave Le ClÃ?Â©zio"	2003	Gallimard	http://images.amazon.com/images/P/2070426769.01.THUMBZZZ.jpg

```
books.at[209538 ,'Publisher'] = 'DK Publishing Inc'
books.at[209538 ,'Year-Of-Publication'] = 2000
books.at[209538 ,'Book-Title'] = 'DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)'
books.at[209538 ,'Book-Author'] = 'Michael Teitelbaum'

books.at[221678 ,'Publisher'] = 'DK Publishing Inc'
books.at[221678 ,'Year-Of-Publication'] = 2000
books.at[209538 ,'Book-Title'] = 'DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)'
books.at[209538 ,'Book-Author'] = 'James Buckley'

books.at[220731 ,'Publisher'] = 'Gallimard'
books.at[220731 ,'Year-Of-Publication'] = '2003'
books.at[209538 ,'Book-Title'] = 'Peuple du ciel - Suivi de Les bergers '
books.at[209538 ,'Book-Author'] = 'Jean-Marie Gustave Le ClÃ?Â©zio'
```

```
## Converting year of publication in Numbers
books['Year-Of-Publication'] = books['Year-Of-Publication'].astype(int)
```

```
print(sorted(list(books['Year-Of-Publication'].unique())))
```

```
[0, 1376, 1378, 1806, 1897, 1900, 1901, 1902, 1904, 1906, 1908, 1909, 1910, 1911, 1914, 1917, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933,
```

```
## Replacing Invalid years with max year
count = Counter(books['Year-Of-Publication'])
[k for k, v in count.items() if v == max(count.values())]
```

```
[2002]
```

```
books.loc[books['Year-Of-Publication'] > 2021, 'Year-Of-Publication'] = 2002
books.loc[books['Year-Of-Publication'] == 0, 'Year-Of-Publication'] = 2002
```

```
## Uppercasing all alphabets in ISBN
books['ISBN'] = books['ISBN'].str.upper()
```

```
## Drop duplicate rows
books.drop_duplicates(keep='last', inplace=True)
books.reset_index(drop = True, inplace = True)
```

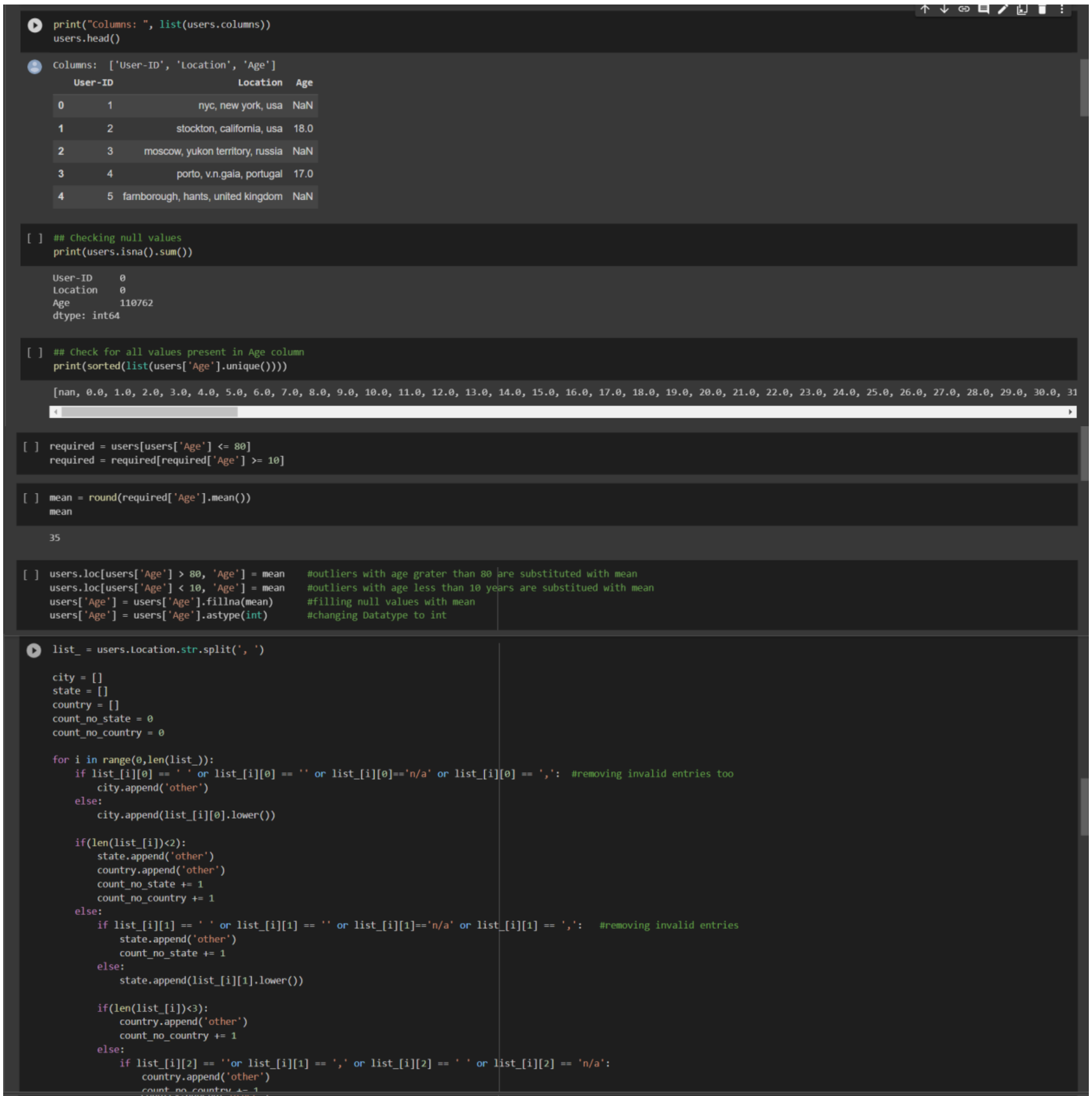
```
books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271047 entries, 0 to 271046
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   ISBN                 271047 non-null  object
 1   Book-Title           271047 non-null  object
 2   Book-Author          271047 non-null  object
 3   Year-Of-Publication  271047 non-null  int64 
 4   Publisher            271047 non-null  object
dtypes: int64(1), object(4)
memory usage: 10.3+ MB
```

```
books.head()
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	0374157065	Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused It	Gina Bari Kolata	1999	Farrar Straus Giroux
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton &amp; Company

Users dataset:



```
print("Columns: ", list(users.columns))
users.head()
```

```
Columns:  ['User-ID', 'Location', 'Age']
```

	User-ID	Location	Age
0	1	nyc, new york, usa	NaN
1	2	stockton, california, usa	18.0
2	3	moscow, yukon territory, russia	NaN
3	4	porto, v.n.gaia, portugal	17.0
4	5	farnborough, hants, united kingdom	NaN

```
## Checking null values
print(users.isna().sum())
```

```
User-ID      0
Location     0
Age          110762
dtype: int64
```

```
## Check for all values present in Age column
print(sorted(list(users['Age'].unique())))
```

```
[nan, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31
```

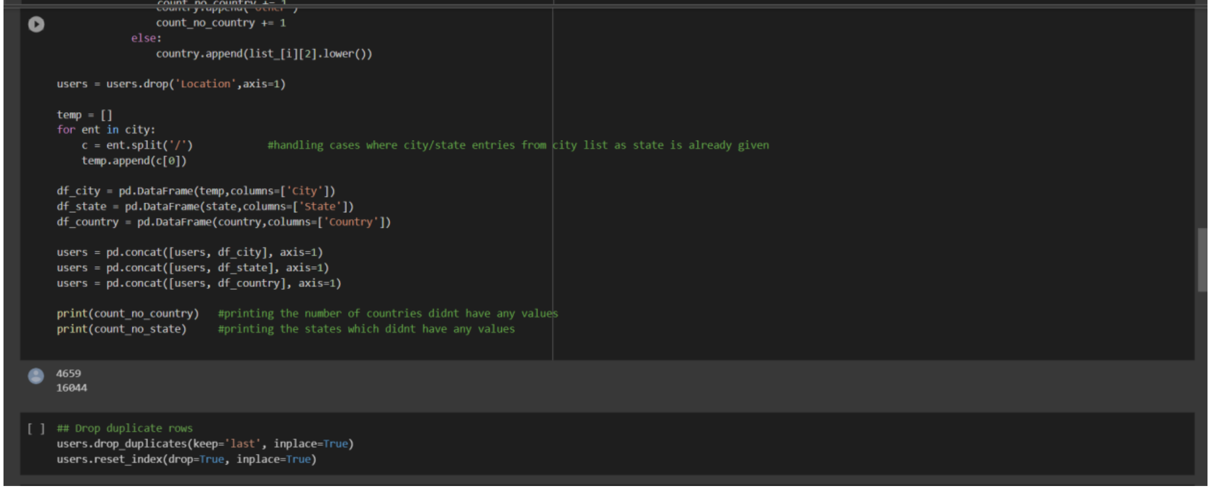
```
required = users[users['Age'] <= 80]
required = required[required['Age'] >= 10]
```

```
mean = round(required['Age'].mean())
mean
```

```
35
```

```
users.loc[users['Age'] > 80, 'Age'] = mean     #outliers with age grater than 80 are substituted with mean
users.loc[users['Age'] < 10, 'Age'] = mean     #outliers with age less than 10 years are substitued with mean
users['Age'] = users['Age'].fillna(mean)       #filling null values with mean
users['Age'] = users['Age'].astype(int)        #changing Datatype to int
```

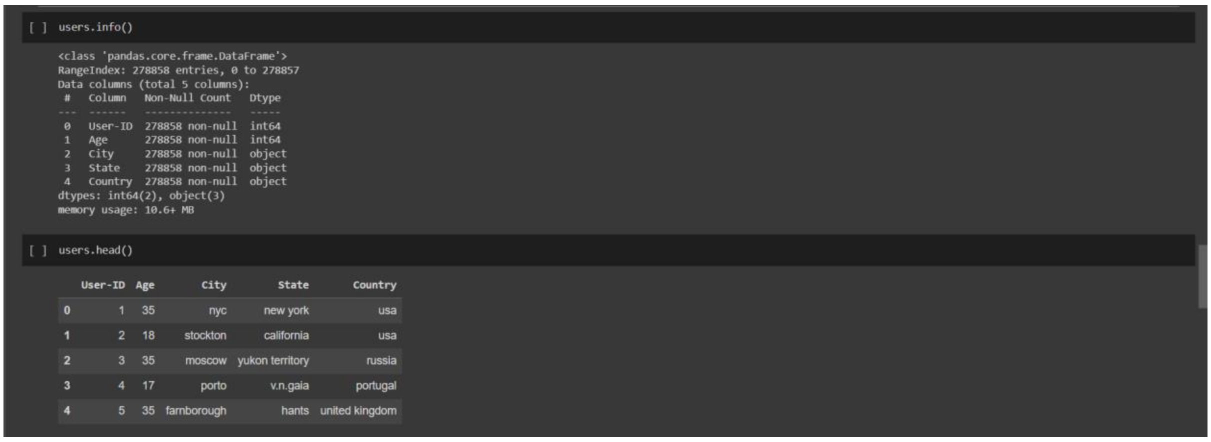
```
list_ = users.Location.str.split(', ')

city = []
state = []
country = []
count_no_state = 0
count_no_country = 0

for i in range(0,len(list_)):
    if list_[i][0] == ' ' or list_[i][0] == '' or list_[i][0]=='n/a' or list_[i][0] == ',':  #removing invalid entries too
        city.append('other')
    else:
        city.append(list_[i][0].lower())

    if(len(list_[i])<2):
        state.append('other')
        country.append('other')
        count_no_state += 1
        count_no_country += 1
    else:
        if list_[i][1] == ' ' or list_[i][1] == '' or list_[i][1]=='n/a' or list_[i][1] == ',':   #removing invalid entries
            state.append('other')
            count_no_state += 1
        else:
            state.append(list_[i][1].lower())

        if(len(list_[i])<3):
            country.append('other')
            count_no_country += 1
        else:
            if list_[i][2] == ''or list_[i][1] == ',' or list_[i][2] == ' ' or list_[i][2] == 'n/a':
                country.append('other')
                count_no_country += 1
```



```
                count_no_country += 1
            else:
                country.append(list_[i][2].lower())

users = users.drop('Location',axis=1)

temp = []
for ent in city:
    c = ent.split('/')            #handling cases where city/state entries from city list as state is already given
    temp.append(c[0])

df_city = pd.DataFrame(temp,columns=['City'])
df_state = pd.DataFrame(state,columns=['State'])
df_country = pd.DataFrame(country,columns=['Country'])

users = pd.concat([users, df_city], axis=1)
users = pd.concat([users, df_state], axis=1)
users = pd.concat([users, df_country], axis=1)

print(count_no_country)   #printing the number of countries didnt have any values
print(count_no_state)     #printing the states which didnt have any values
```

```
4659
16044
```

```
## Drop duplicate rows
users.drop_duplicates(keep='last', inplace=True)
users.reset_index(drop=True, inplace=True)
```

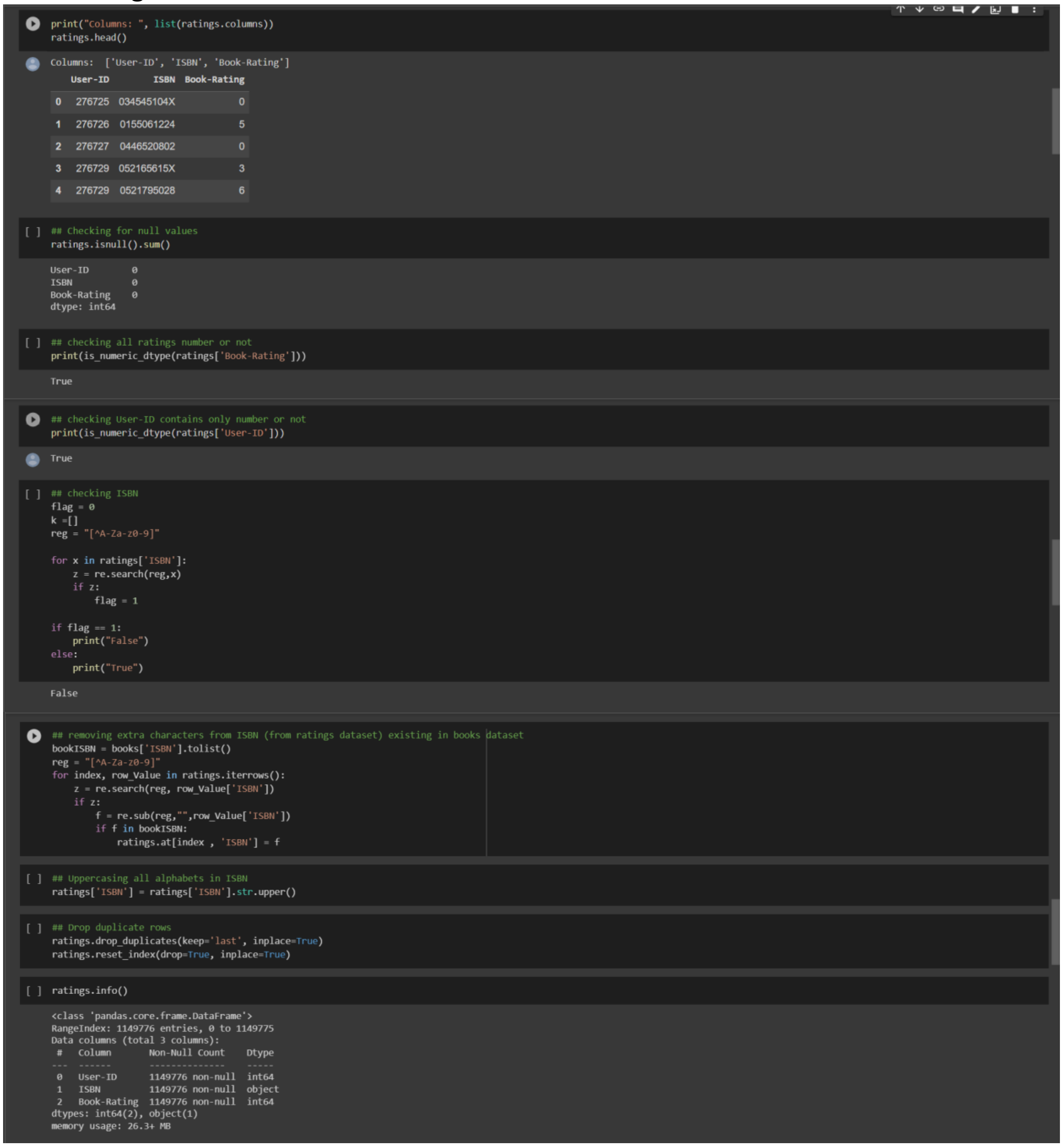


```
[ ] users.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 278858 entries, 0 to 278857
    Data columns (total 5 columns):
     #   Column   Non-Null Count   Dtype
    ---  ------   --------------   -----
     0   User-ID  278858 non-null  int64
     1   Age      278858 non-null  int64
     2   City     278858 non-null  object
     3   State    278858 non-null  object
     4   Country  278858 non-null  object
    dtypes: int64(2), object(3)
    memory usage: 10.6+ MB
```

```
[ ] users.head()
```

	User-ID	Age	City	State	Country
0	1	35	nyc	new york	usa
1	2	18	stockton	california	usa
2	3	35	moscow	yukon territory	russia
3	4	17	porto	v.n.gaia	portugal
4	5	35	farnborough	hants	united kingdom

Book rating dataset:



```
print("Columns: ", list(ratings.columns))
ratings.head()

Columns:  ['User-ID', 'ISBN', 'Book-Rating']
```

	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

```
[ ] ## Checking for null values
    ratings.isnull().sum()

    User-ID        0
    ISBN           0
    Book-Rating    0
    dtype: int64
```

```
[ ] ## checking all ratings number or not
    print(is_numeric_dtype(ratings['Book-Rating']))

    True
```

```
## checking User-ID contains only number or not
print(is_numeric_dtype(ratings['User-ID']))

True
```

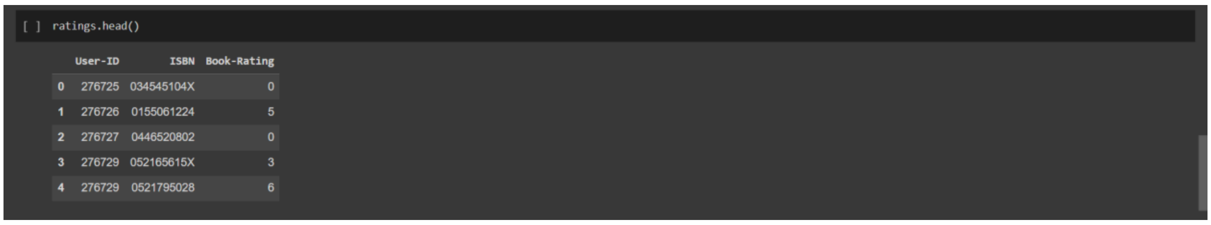
```
[ ] ## checking ISBN
    flag = 0
    k =[]
    reg = "[^A-Za-z0-9]"

    for x in ratings['ISBN']:
        z = re.search(reg,x)
        if z:
            flag = 1

    if flag == 1:
        print("False")
    else:
        print("True")

    False
```

```
## removing extra characters from ISBN (from ratings dataset) existing in books dataset
bookISBN = books['ISBN'].tolist()
reg = "[^A-Za-z0-9]"
for index, row_Value in ratings.iterrows():
    z = re.search(reg, row_Value['ISBN'])
    if z:
        f = re.sub(reg,"",row_Value['ISBN'])
        if f in bookISBN:
            ratings.at[index , 'ISBN'] = f
```

```
[ ] ## Uppercasing all alphabets in ISBN
    ratings['ISBN'] = ratings['ISBN'].str.upper()
```

```
[ ] ## Drop duplicate rows
    ratings.drop_duplicates(keep='last', inplace=True)
    ratings.reset_index(drop=True, inplace=True)
```

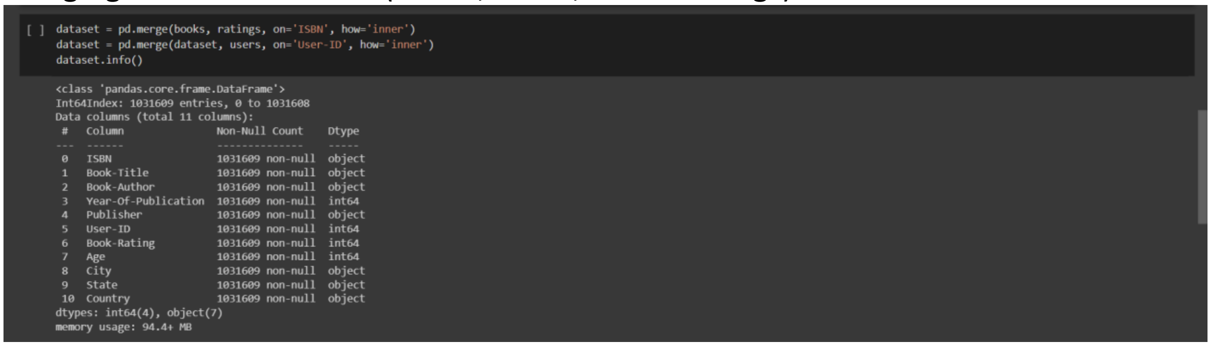
```
[ ] ratings.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 1149776 entries, 0 to 1149775
    Data columns (total 3 columns):
     #   Column       Non-Null Count    Dtype
    ---  ------       --------------    -----
     0   User-ID      1149776 non-null  int64
     1   ISBN         1149776 non-null  object
     2   Book-Rating  1149776 non-null  int64
    dtypes: int64(2), object(1)
    memory usage: 26.3+ MB
```



```
[ ] ratings.head()
```

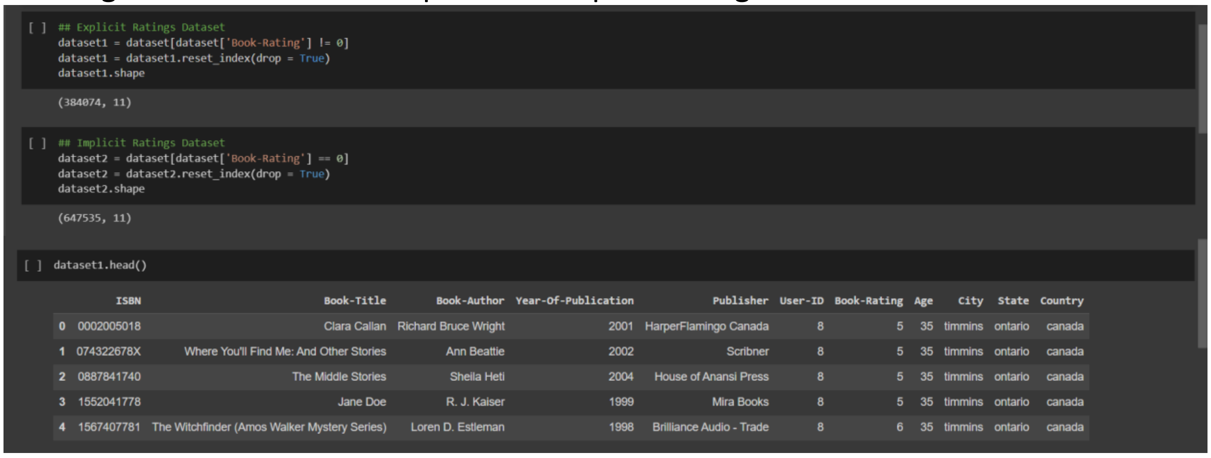
	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

Merging of all three tables (Books, Users, Books-ratings):



```
[ ] dataset = pd.merge(books, ratings, on='ISBN', how='inner')
    dataset = pd.merge(dataset, users, on='User-ID', how='inner')
    dataset.info()

    <class 'pandas.core.frame.DataFrame'>
    Int64Index: 1031609 entries, 0 to 1031608
    Data columns (total 11 columns):
     #   Column               Non-Null Count    Dtype
    ---  ------               --------------    -----
     0   ISBN                 1031609 non-null  object
     1   Book-Title           1031609 non-null  object
     2   Book-Author          1031609 non-null  object
     3   Year-Of-Publication  1031609 non-null  int64
     4   Publisher            1031609 non-null  object
     5   User-ID              1031609 non-null  int64
     6   Book-Rating          1031609 non-null  int64
     7   Age                  1031609 non-null  int64
     8   City                 1031609 non-null  object
     9   State                1031609 non-null  object
     10  Country              1031609 non-null  object
    dtypes: int64(4), object(7)
    memory usage: 94.4+ MB
```

Dividing the data based on implicit and explicit ratings dataset:

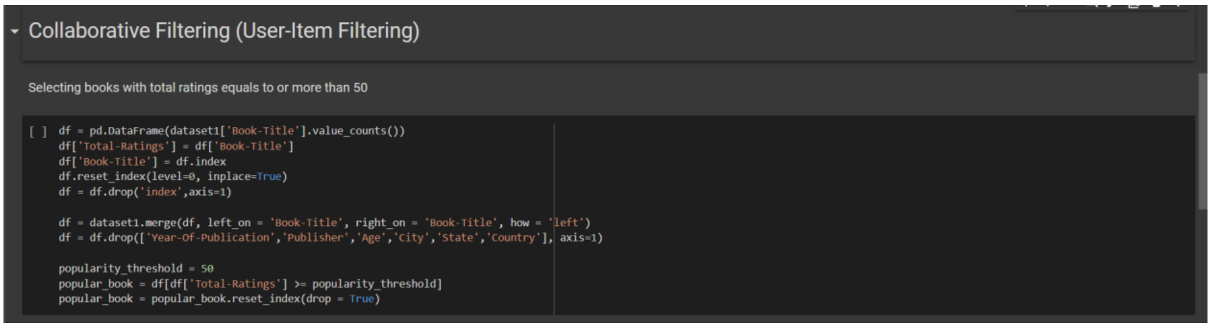


```
[ ] ## Explicit Ratings Dataset
    dataset1 = dataset[dataset['Book-Rating'] != 0]
    dataset1 = dataset1.reset_index(drop = True)
    dataset1.shape

    (384074, 11)

[ ] ## Implicit Ratings Dataset
    dataset2 = dataset[dataset['Book-Rating'] == 0]
    dataset2 = dataset2.reset_index(drop = True)
    dataset2.shape

    (647535, 11)

[ ] dataset1.head()
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	User-ID	Book-Rating	Age	City	State	Country
0	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	8	5	35	timmins	ontario	canada
1	074322678X	Where You'll Find Me: And Other Stories	Ann Beattie	2002	Scribner	8	5	35	timmins	ontario	canada
2	0887841740	The Middle Stories	Sheila Heti	2004	House of Anansi Press	8	5	35	timmins	ontario	canada
3	1552041778	Jane Doe	R. J. Kaiser	1999	Mira Books	8	5	35	timmins	ontario	canada
4	1567407781	The Witchfinder (Amos Walker Mystery Series)	Loren D. Estleman	1998	Brilliance Audio - Trade	8	6	35	timmins	ontario	canada

Methodology

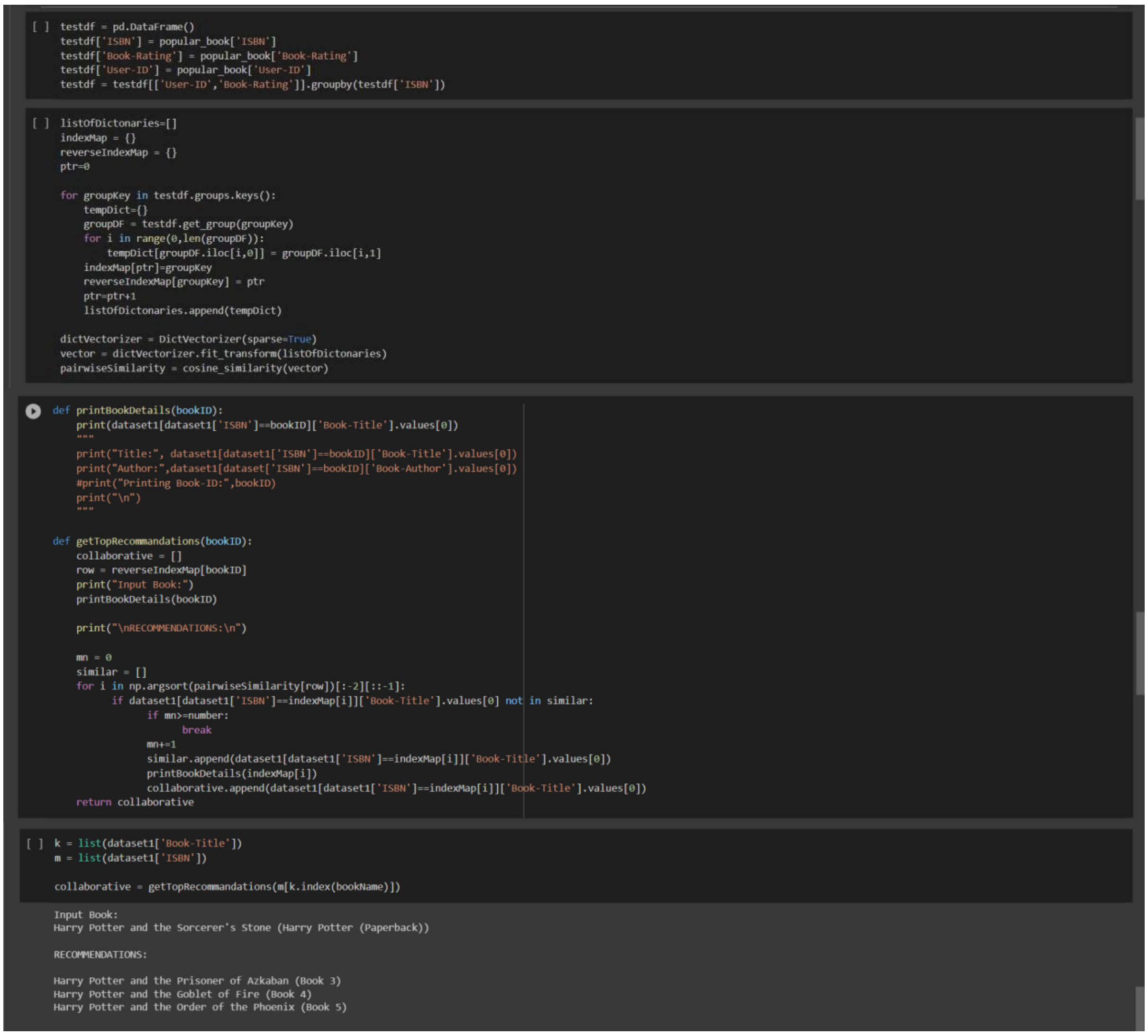
1. Correlation based
2. Neighbourhood based
3. Content based



Collaborative Filtering (User-Item Filtering)

Selecting books with total ratings equals to or more than 50

```
[ ] df = pd.DataFrame(dataset1['Book-Title'].value_counts())
    df['Total-Ratings'] = df['Book-Title']
    df['Book-Title'] = df.index
    df.reset_index(level=0, inplace=True)
    df = df.drop('index',axis=1)

    df = dataset1.merge(df, left_on = 'Book-Title', right_on = 'Book-Title', how = 'left')
    df = df.drop(['Year-Of-Publication','Publisher','Age','City','State','Country'], axis=1)

    popularity_threshold = 50
    popular_book = df[df['Total-Ratings'] >= popularity_threshold]
    popular_book = popular_book.reset_index(drop = True)
```



```
[ ] testdf = pd.DataFrame()
    testdf['ISBN'] = popular_book['ISBN']
    testdf['Book-Rating'] = popular_book['Book-Rating']
    testdf['User-ID'] = popular_book['User-ID']
    testdf = testdf[['User-ID','Book-Rating']].groupby(testdf['ISBN'])
```

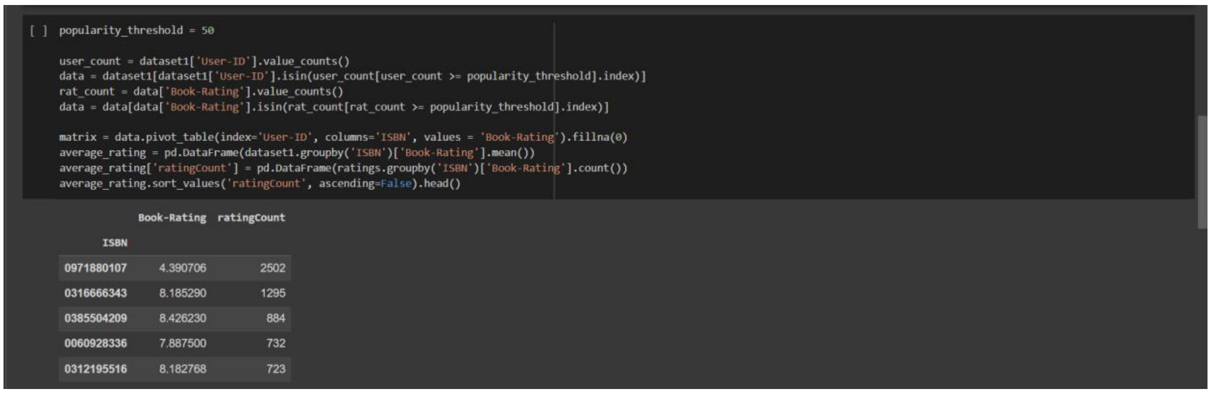
```
[ ] listOfDictonaries=[]
    indexMap = {}
    reverseIndexMap = {}
    ptr=0

    for groupKey in testdf.groups.keys():
        tempDict={}
        groupDF = testdf.get_group(groupKey)
        for i in range(0,len(groupDF)):
            tempDict[groupDF.iloc[i,0]] = groupDF.iloc[i,1]
        indexMap[ptr]=groupKey
        reverseIndexMap[groupKey] = ptr
        ptr=ptr+1
        listOfDictonaries.append(tempDict)

    dictVectorizer = DictVectorizer(sparse=True)
    vector = dictVectorizer.fit_transform(listOfDictonaries)
    pairwiseSimilarity = cosine_similarity(vector)
```

```
def printBookDetails(bookID):
    print(dataset1[dataset1['ISBN']==bookID]['Book-Title'].values[0])
    """
    print("Title:", dataset1[dataset1['ISBN']==bookID]['Book-Title'].values[0])
    print("Author:",dataset1[dataset['ISBN']==bookID]['Book-Author'].values[0])
    #print("Printing Book-ID:",bookID)
    print("\n")
    """

def getTopRecommandations(bookID):
    collaborative = []
    row = reverseIndexMap[bookID]
    print("Input Book:")
    printBookDetails(bookID)

    print("\nRECOMMENDATIONS:\n")

    mn = 0
    similar = []
    for i in np.argsort(pairwiseSimilarity[row])[:-2][::-1]:
          if dataset1[dataset1['ISBN']==indexMap[i]]['Book-Title'].values[0] not in similar:
                if mn>=number:
                      break
                mn+=1
                similar.append(dataset1[dataset1['ISBN']==indexMap[i]]['Book-Title'].values[0])
                printBookDetails(indexMap[i])
                collaborative.append(dataset1[dataset1['ISBN']==indexMap[i]]['Book-Title'].values[0])
    return collaborative
```

```
[ ] k = list(dataset1['Book-Title'])
    m = list(dataset1['ISBN'])

    collaborative = getTopRecommandations(m[k.index(bookName)])
```

```
Input Book:
Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))

RECOMMENDATIONS:

Harry Potter and the Prisoner of Azkaban (Book 3)
Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Order of the Phoenix (Book 5)
```

1. *Correlation based*



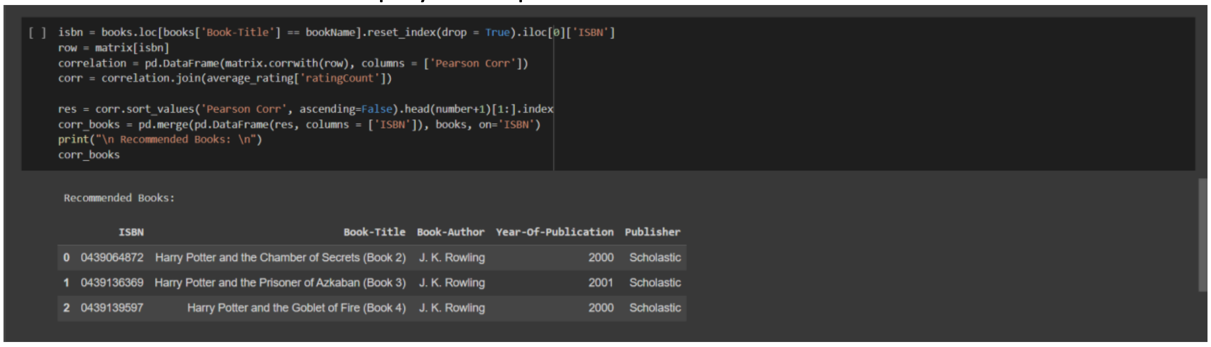
```
[ ] popularity_threshold = 50

    user_count = dataset1['User-ID'].value_counts()
    data = dataset1[dataset1['User-ID'].isin(user_count[user_count >= popularity_threshold].index)]
    rat_count = data['Book-Rating'].value_counts()
    data = data[data['Book-Rating'].isin(rat_count[rat_count >= popularity_threshold].index)]

    matrix = data.pivot_table(index='User-ID', columns='ISBN', values = 'Book-Rating').fillna(0)
    average_rating = pd.DataFrame(dataset1.groupby('ISBN')['Book-Rating'].mean())
    average_rating['ratingCount'] = pd.DataFrame(ratings.groupby('ISBN')['Book-Rating'].count())
    average_rating.sort_values('ratingCount', ascending=False).head()
```

	Book-Rating	ratingCount
ISBN		
0971880107	4.390706	2502
0316666343	8.185290	1295
0385504209	8.426230	884
0060928336	7.887500	732
0312195516	8.182768	723

- The variable "popularity_threshold" is set to 50, which means that only books with 50 or more ratings and users who have rated 50 or more books will be included in the analysis.
- First, the code is counting the number of times each user appears in the "User-ID" column of the "dataset1" dataset using the value_counts() method. Then, it filters the "dataset1" dataset to only include rows where the "User-ID" appears at least

"popularity_threshold" number of times using the isin() method.

- Next, the code is counting the number of times each rating value appears in the "Book-Rating" column of the filtered dataset using the value_counts() method. Then, it filters the filtered dataset to only include rows where the "Book-Rating" appears at least "popularity_threshold" number of times using the isin() method.
- After that, the code creates a pivot table called "matrix" using the filtered dataset. The pivot table has users as the index, books as the columns, and book ratings as the values. The pivot table is filled with zeros where there is no rating data.
- Finally, the code calculates the average rating and the number of ratings for each book in the original "dataset1" dataset using the groupby() method. The resulting DataFrame is stored in "average_rating" variable. The sort_values() method is used to sort the books by the number of ratings in descending order, and the head() method is used to display the top-rated books



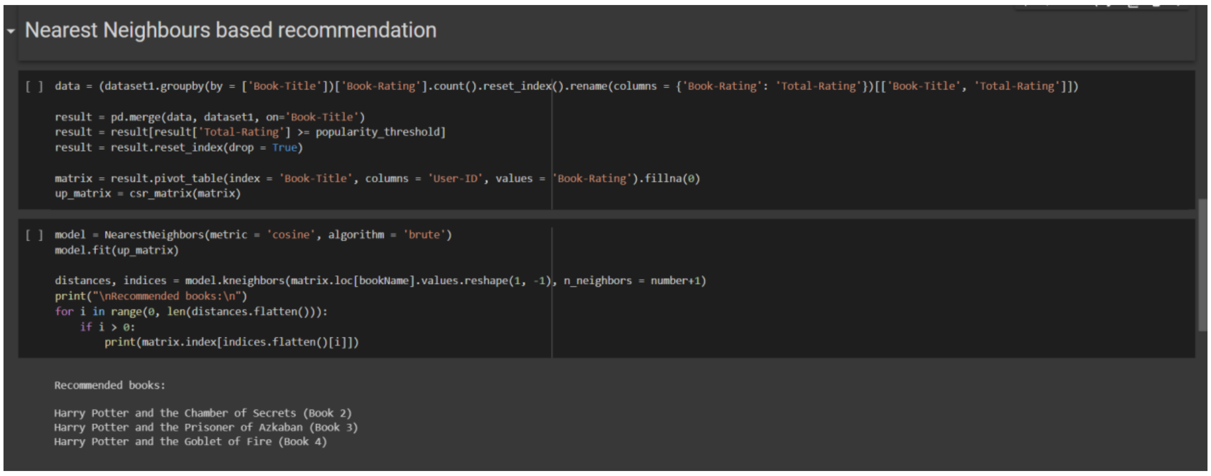
```
[ ] isbn = books.loc[books['Book-Title'] == bookName].reset_index(drop = True).iloc[0]['ISBN']
    row = matrix[isbn]
    correlation = pd.DataFrame(matrix.corrwith(row), columns = ['Pearson Corr'])
    corr = correlation.join(average_rating['ratingCount'])

    res = corr.sort_values('Pearson Corr', ascending=False).head(number+1)[1:].index
    corr_books = pd.merge(pd.DataFrame(res, columns = ['ISBN']), books, on='ISBN')
    print("\n Recommended Books: \n")
    corr_books
```

```
Recommended Books:
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
0	0439064872	Harry Potter and the Chamber of Secrets (Book 2)	J. K. Rowling	2000	Scholastic
1	0439136369	Harry Potter and the Prisoner of Azkaban (Book 3)	J. K. Rowling	2001	Scholastic
2	0439139597	Harry Potter and the Goblet of Fire (Book 4)	J. K. Rowling	2000	Scholastic

- The code begins by using the loc[] method to filter the "books" dataset by the "Book-Title" column, searching for the book with the name "bookName". Once the matching book is found, the reset_index() method is used to reset the index of the resulting DataFrame, and the iloc[] method is used to select the first row (i.e., the row with index 0). The ISBN value from this row is then stored in the "isbn" variable.
- Next, the code selects the row in the "matrix" pivot table that corresponds to the book with the specified "isbn" using the following code: row = matrix[isbn].
- Then, the code calculates the Pearson correlation between the selected book and all the other books in the dataset using the corrwith() method of the "matrix" pivot table. The resulting correlation values are stored in a Data Frame called "correlation", with the column named "Pearson Corr".
- The "corr" Data Frame is created by joining the "correlation" Data Frame with the "average_rating" Data Frame using the join() method. This new Data Frame contains the Pearson correlation coefficients for each book, as well as the number of ratings each book has received.
- Finally, the code selects the top "number+1" books with the highest Pearson correlation coefficients (excluding the book itself) using the sort_values() method of the "corr" Data Frame, and the head() method to select the top "number+1" books. The resulting index values (i.e., the ISBN numbers) are stored in the "res" variable.
- The code then merges the "res" Data Frame with the "books" Data Frame on the "ISBN" column using the merge() method to create a new Data Frame called "corr_books". This new Data Frame contains the recommended books, including their ISBN, book title, author, and year of publication.
- Finally, the code prints the recommended books Data Frame with the message "Recommended Books".

2. *Neighborhood based*



Nearest Neighbours based recommendation

```
[ ] data = (dataset1.groupby(by = ['Book-Title'])['Book-Rating'].count().reset_index().rename(columns = {'Book-Rating': 'Total-Rating'})[['Book-Title', 'Total-Rating']])

    result = pd.merge(data, dataset1, on='Book-Title')
    result = result[result['Total-Rating'] >= popularity_threshold]
    result = result.reset_index(drop = True)

    matrix = result.pivot_table(index = 'Book-Title', columns = 'User-ID', values = 'Book-Rating').fillna(0)
    up_matrix = csr_matrix(matrix)
```

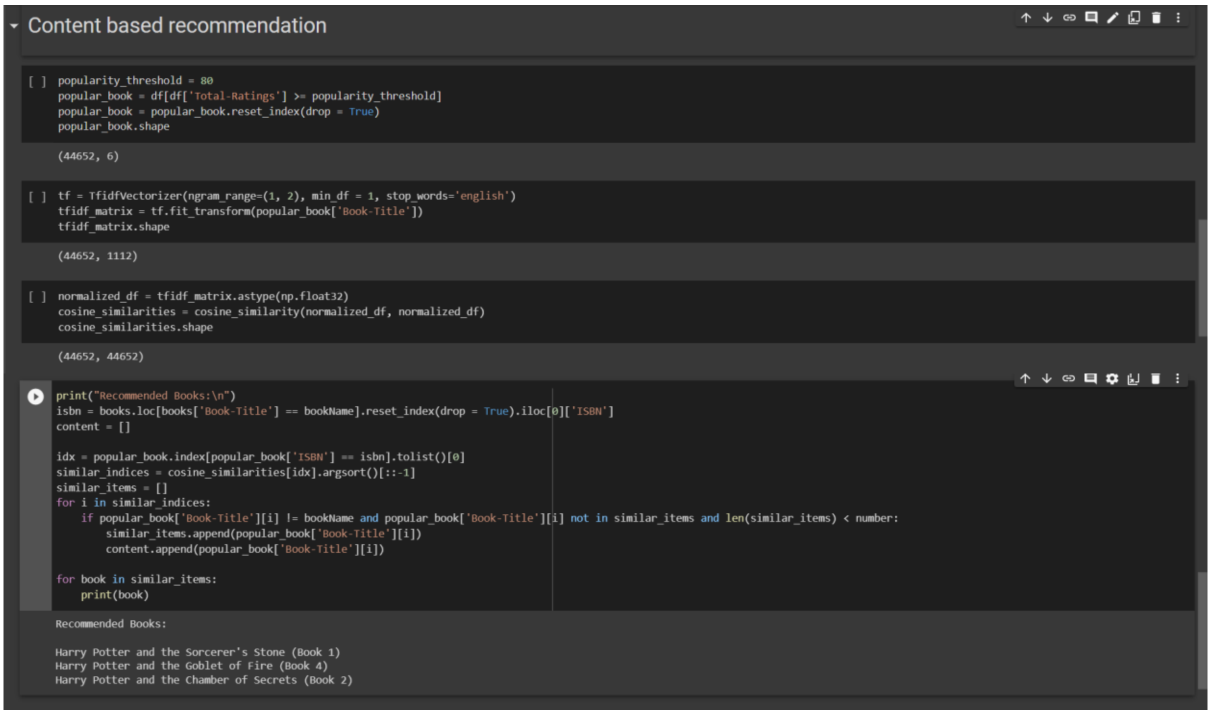
```
[ ] model = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
    model.fit(up_matrix)

    distances, indices = model.kneighbors(matrix.loc[bookName].values.reshape(1, -1), n_neighbors = number+1)
    print("\nRecommended books:\n")
    for i in range(0, len(distances.flatten())):
        if i > 0:
            print(matrix.index[indices.flatten()[i]])
```

```
Recommended books:

Harry Potter and the Chamber of Secrets (Book 2)
Harry Potter and the Prisoner of Azkaban (Book 3)
Harry Potter and the Goblet of Fire (Book 4)
```

- The first step is to group the "dataset1" Data Frame by the "Book-Title" column, and count the number of ratings for each book using the groupby() and count() methods. The resulting Data Frame is assigned to the "data" variable, and only the "Book-Title" and "Total-Rating" columns are kept.
- Next, the code merges the "data" Data Frame with the "dataset1" Data Frame on the "Book-Title" column using the merge() method. The resulting Data Frame is assigned to the "result" variable.
- The code then filters out any rows in the "result" Data Frame where the "Total-Rating" value is less than the "popularity_threshold" variable. This is done by selecting only the rows where "Total-Rating" is greater than or equal to "popularity_threshold", and assigning the result back to the "result" variable. The reset_index() method is then used to reset the index of the resulting Data Frame.
- Next, the code creates a pivot table from the "result" Data Frame, where the rows represent each book, the columns represent each user, and the values represent the book ratings. The pivot table is created using the pivot_table() method, and the resulting Data Frame is assigned to the "matrix" variable. Any missing values in the pivot table are filled with 0 using the fillna() method.
- Finally, the code converts the pivot table into a sparse matrix using the csr_matrix() function from the "scipy.sparse" module. This is done to save memory and increase performance when working with large datasets. The resulting sparse matrix is assigned to the "up_matrix" variable.
- The code is using the k-nearest neighbors (KNN) algorithm to find the "number+1" most similar books to a given book ("bookName") in the "matrix" Data Frame, based on the cosine similarity metric.
- First, an instance of the "Nearest Neighbours" class is created, with the cosine similarity metric and brute-force algorithm. Then, the "fit()" method of the "model" object is called on the "up_matrix" sparse matrix to fit the KNN model to the data.
- Next, the "k-neighbors()" method of the "model" object is used to find the k-nearest neighbors of the input book, where k is set to "number+1". The distances and indices of the nearest neighbors are returned, and are assigned to the "distances" and "indices" variables, respectively.
- Finally, a loop is used to iterate over the indices of the nearest neighbors, and the book titles are printed out using the "indices.flatten()[i]" expression to get the index

of the book in the "matrix" Data Frame. The loop starts at index 1, since the first index will always be the input book itself.

3. *Content based*


Content based recommendation

```
[ ] popularity_threshold = 80
    popular_book = df[df['Total-Ratings'] >= popularity_threshold]
    popular_book = popular_book.reset_index(drop = True)
    popular_book.shape
```

```
(44652, 6)
```

```
[ ] tf = TfidfVectorizer(ngram_range=(1, 2), min_df = 1, stop_words='english')
    tfidf_matrix = tf.fit_transform(popular_book['Book-Title'])
    tfidf_matrix.shape
```

```
(44652, 1112)
```

```
[ ] normalized_df = tfidf_matrix.astype(np.float32)
    cosine_similarities = cosine_similarity(normalized_df, normalized_df)
    cosine_similarities.shape
```

```
(44652, 44652)
```

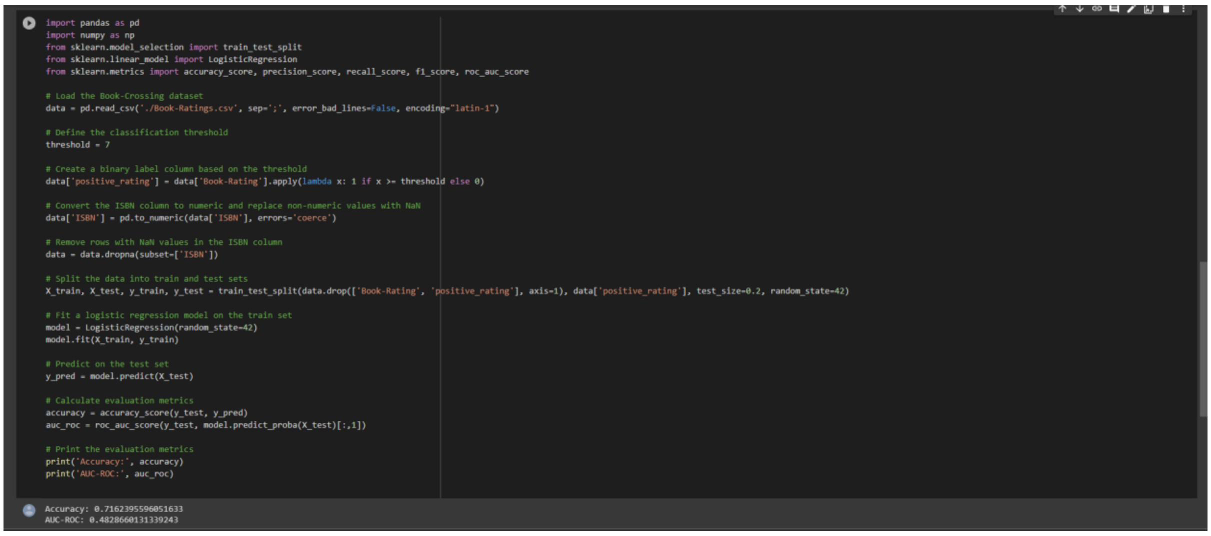
```
print("Recommended Books:\n")
isbn = books.loc[books['Book-Title'] == bookName].reset_index(drop = True).iloc[0]['ISBN']
content = []

idx = popular_book.index[popular_book['ISBN'] == isbn].tolist()[0]
similar_indices = cosine_similarities[idx].argsort()[::-1]
similar_items = []
for i in similar_indices:
    if popular_book['Book-Title'][i] != bookName and popular_book['Book-Title'][i] not in similar_items and len(similar_items) < number:
        similar_items.append(popular_book['Book-Title'][i])
        content.append(popular_book['Book-Title'][i])

for book in similar_items:
    print(book)
```

```
Recommended Books:

Harry Potter and the Sorcerer's Stone (Book 1)
Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Chamber of Secrets (Book 2)
```

- This code performs content-based recommendation using cosine similarity to find books that are similar to the given book based on their titles.
- First, the code sets a popularity threshold, selects books with ratings greater than or equal to that threshold, and creates a TF-IDF matrix of the book titles. Then, the cosine similarity between each pair of books in the matrix is calculated, resulting in a cosine similarity matrix.
- Next, given a book name, the code finds its corresponding ISBN and its index in the popular_book DataFrame. It then retrieves the row of cosine similarity values for the given book from the cosine similarity matrix, sorts it in descending order, and selects the top number books that are most similar to the given book.
- Finally, the code prints the recommended books that are most similar to the given book.
- isbn = books.loc[books['Book-Title'] == bookName].reset_index(drop = True).iloc[0]['ISBN']: Finds the ISBN of the input book name from the 'books' dataset.
- content = []: Initializes an empty list 'content'.
- idx = popular_book.index[popular_book['ISBN'] == isbn].tolist()[0]: Finds the index of the book in the 'popular_book' dataset that matches the ISBN of the input book.
- similar_indices = cosine_similarities[idx].argsort()[::-1]: Calculates the cosine similarity between the input book and all other books in the 'popular_book' dataset, and returns the indices of the books in descending order of similarity.
- similar_items = []: Initializes an empty list 'similar_items'.
- for i in similar_indices:: Loops through the indices of the similar books.

- if popular_book['Book-Title'][i] != bookName and popular_book['Book-Title'][i] not in similar_items and len(similar_items) < number:: If the book at the current index is not the input book, has not already been added to 'similar_items', and the number of books in 'similar_items' is less than the requested number, then:
- similar_items.append(popular_book['Book-Title'][i]): Adds the title of the similar book to 'similar_items'.
- content.append(popular_book['Book-Title'][i]): Adds the title of the similar book to 'content'.
- for book in similar_items:: Loops through the titles of the similar books.
- print(book): Prints the title of each similar book.

Evaluation Metrics



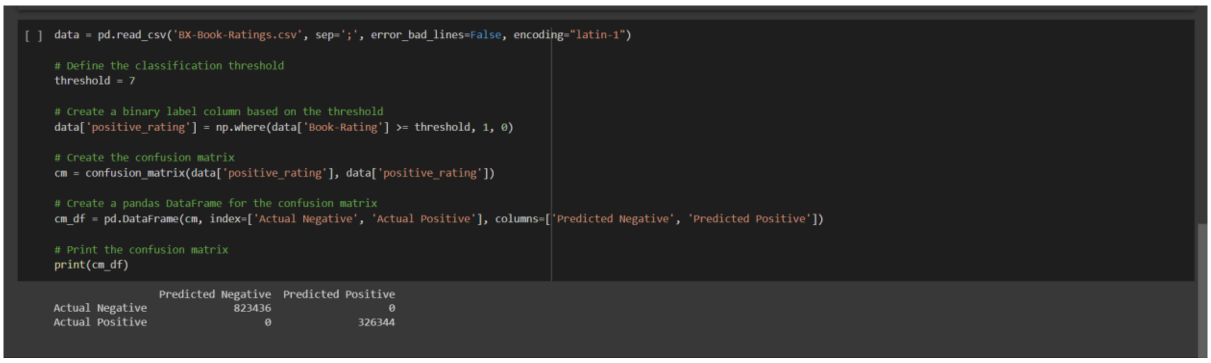
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Load the Book-Crossing dataset
data = pd.read_csv('./Book-Ratings.csv', sep=';', error_bad_lines=False, encoding="latin-1")

# Define the classification threshold
threshold = 7

# Create a binary label column based on the threshold
data['positive_rating'] = data['Book-Rating'].apply(lambda x: 1 if x >= threshold else 0)

# Convert the ISBN column to numeric and replace non-numeric values with NaN
data['ISBN'] = pd.to_numeric(data['ISBN'], errors='coerce')

# Remove rows with NaN values in the ISBN column
data = data.dropna(subset=['ISBN'])

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(data.drop(['Book-Rating', 'positive_rating'], axis=1), data['positive_rating'], test_size=0.2, random_state=42)

# Fit a logistic regression model on the train set
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
auc_roc = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])

# Print the evaluation metrics
print('Accuracy:', accuracy)
print('AUC-ROC:', auc_roc)
```

```
Accuracy: 0.7162395596051633
AUC-ROC: 0.4828660131339243
```

- The code is performing a binary classification task on the Book-Crossing dataset, where the goal is to predict whether a book rating is positive or not based on the ISBN and other features of the book.
- The code first loads the dataset into a Pandas DataFrame using the read_csv() function. It then defines a threshold value of 7 for the book rating, and creates a binary label column (positive_rating) based on whether the rating is greater than or equal to the threshold.
- Next, the code converts the ISBN column to numeric values using the pd.to_numeric() function, and replaces any non-numeric values with NaN. It then drops any rows with NaN values in the ISBN column using the dropna() function.
- The code then splits the dataset into training and test sets using the train_test_split() function from scikit-learn. It uses the logistic regression algorithm (LogisticRegression()) from scikit-learn to fit a model on the training set, and then predicts the binary labels for the test set using the predict() method.
- Finally, the code calculates two evaluation metrics for the model: accuracy and AUC-ROC. Accuracy is calculated using the accuracy_score() function, which compares the predicted binary labels with the actual binary labels in the test set.
- AUC-ROC is calculated using the roc_auc_score() function, which calculates the area under the receiver operating characteristic curve (ROC-AUC) for the model's predicted probabilities.

- The evaluation metrics show that the model achieves an accuracy of 0.716 and an AUC-ROC of 0.483. This means that the model's performance is better than random guessing, but it is not very accurate in predicting positive ratings based on the ISBN and other features of the book.

Confusion Matrix



```
[ ] data = pd.read_csv('BX-Book-Ratings.csv', sep=';', error_bad_lines=False, encoding="latin-1")

    # Define the classification threshold
    threshold = 7

    # Create a binary label column based on the threshold
    data['positive_rating'] = np.where(data['Book-Rating'] >= threshold, 1, 0)

    # Create the confusion matrix
    cm = confusion_matrix(data['positive_rating'], data['positive_rating'])

    # Create a pandas DataFrame for the confusion matrix
    cm_df = pd.DataFrame(cm, index=['Actual Negative', 'Actual Positive'], columns=['Predicted Negative', 'Predicted Positive'])

    # Print the confusion matrix
    print(cm_df)
```

```
                 Predicted Negative  Predicted Positive
Actual Negative              823436                   0
Actual Positive                   0              326344
```

- The Book-Crossing dataset is loaded from a CSV file using the read_csv() function. A threshold value of 7 is defined for the book rating, and a binary label column (positive_rating) is created based on whether the rating is greater than or equal to the threshold using the NumPy np.where() function.

- However, the confusion matrix that is created using confusion_matrix() function is created incorrectly. The same positive_rating column is used as both the actual and predicted labels in the function. This results in a confusion matrix where the true positives and true negatives are both equal to the total number of data points, and the false positives and false negatives are both equal to zero