Project Title : Book Recommendation System

Problem Description

During the last few decades, with the rise of Youtube, Amazon, Netflix, and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys.

In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy, or anything else depending on industries).

Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. The main objective is to create a book recommendation system for users.

Data Description

The Book-Crossing dataset comprises 3 files.

Users:

Contains the users. Note that user IDs (User-ID) have been anonymized and map to integers. Demographic data is provided (Location, Age) if available. Otherwise, these fields contain NULL values.

Books:

Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (Book-Title,Book-Author, Year-Of-Publication, Publisher), obtained from Amazon Web Services. Note that in the case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three

different flavors (Image-URL-S,Image-URL-M, Image-URL-L), i.e., small, medium large. These URLs point to the Amazon website.

Ratings:

Contains the book rating information. Ratings (Book-Rating) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

Importing Libraries

```
#Import all library that will be used in entire project
!pip install scikit-surprise
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, confusion_matrix,accuracy_score,f1_score,roc_c
from sklearn.model_selection import train_test_split
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

import pandas as pd

import ast

from sklearn.metrics.pairwise import cosine_similarity

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import NearestNeighbors

from scipy.sparse import csr_matrix

from surprise import Reader, Dataset

from surprise.model_selection import train_test_split

from surprise import SVDpp, accuracy

from surprise.model_selection import cross_validate

from collections import defaultdict

from surprise import SVD, SVDpp, NMF

from surprise import SlopeOne, CoClustering

import matplotlib
```

```
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 kB 6.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-package
```

```
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp310-cp310-linux
  Stored in directory: /root/.cache/pip/wheels/a5/ca/a8/4e28def53797fdc4363ca4af740db
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.3
```

Mount Drive And Import Data

```
#Mount google drive for access of the play store dataset
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Importing users data
File_path='/content/drive/MyDrive/Capstone Project_4/'
users= pd.read_csv(File_path + 'Users.csv')

#Importing books data
books= pd.read_csv(File_path + 'Books.csv')

#Importing ratings data
ratings= pd.read_csv(File_path + 'Ratings.csv')
```

```
# First Look
users.head(10)
```

	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
5	6	santa monica, california, usa	61.0
6	7	washington, dc, usa	NaN
7	8	timmins, ontario, canada	NaN
8	9	germantown, tennessee, usa	NaN
9	10	albacete, wisconsin, spain	26.0

```
# First Look
books.head(5)
```

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

```
# First Look
ratings.head(10)
```

	User-ID	ISBN	Book-Rating
1	276726	0155061224	5
3	276729	052165615X	3
4	276729	0521795028	6
6	276736	3257224281	8
7	276737	0600570967	6
8	276744	038550120X	7
9	276745	342310538	10
16	276747	0060517794	9
19	276747	0671537458	9
20	276747	0679776818	8

```
#data information
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278858 entries, 0 to 278857
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   User-ID   278858 non-null  int64  
 1   Location  278858 non-null  object 
 2   Age       168096 non-null  float64
dtypes: float64(1), int64(1), object(1)
memory usage: 6.4+ MB
```

```
#data information
books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271360 entries, 0 to 271359
Data columns (total 8 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   ISBN                 271360 non-null  object
 1   Book-Title           271360 non-null  object
 2   Book-Author          271359 non-null  object
 3   Year-Of-Publication  271360 non-null  object
 4   Publisher            271358 non-null  object
 5   Image-URL-S          271360 non-null  object
 6   Image-URL-M          271360 non-null  object
 7   Image-URL-L          271357 non-null  object
dtypes: object(8)
memory usage: 16.6+ MB
```

```
#data information
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1149780 entries, 0 to 1149779
Data columns (total 3 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   User-ID      1149780 non-null  int64
 1   ISBN         1149780 non-null  object
 2   Book-Rating  1149780 non-null  int64
dtypes: int64(2), object(1)
memory usage: 26.3+ MB
```

Handling Missing Vaules

Users Dataset

```
# Missing Value Count Function
def show_missing():
    missing = users.columns[users.isnull().any()].tolist()
    return missing

# Missing data counts and percentage
print('Missing Data Count')
print(users[show_missing()].isnull().sum().sort_values(ascending = False))
print('--'*50)
print('Missing Data Percentage')
print(round(users[show_missing()].isnull().sum().sort_values(ascending = False)/len(users
```

```
Missing Data Count
Age    110762
dtype: int64
----------------------------------------------------------------------------------------
Missing Data Percentage
Age    39.72
dtype: float64
```

```
users['Age'].describe()
```

```
count    168096.000000
mean         34.751434
std          14.428097
min           0.000000
25%          24.000000
50%          32.000000
75%          44.000000
max         244.000000
Name: Age, dtype: float64
```

minimum age 0 and max age 244? so outliers exist in age

Age is positively skewed. Median imputation is preferable for skewed distribution(be it right or left). So we will replace nulls with median in Age column. Outliers affect the mean value of the data but have little effect on the median or mode of a given set of data so we can impute nulls with median before removing outliers.

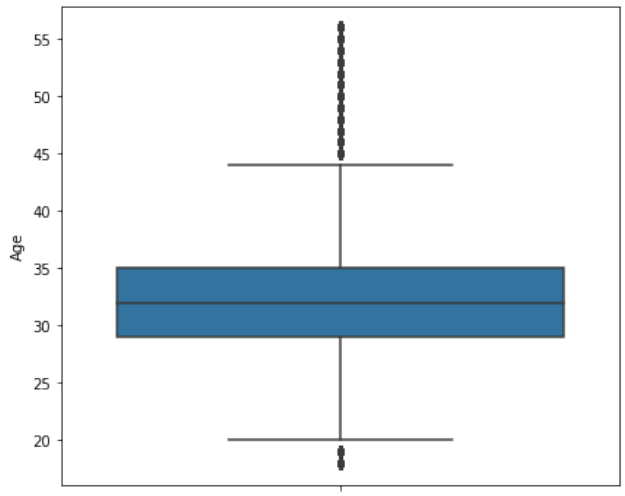
```
#Fill NaN Values in Age Column with Median
users['Age'] =users['Age'].fillna(users['Age'].median())
```

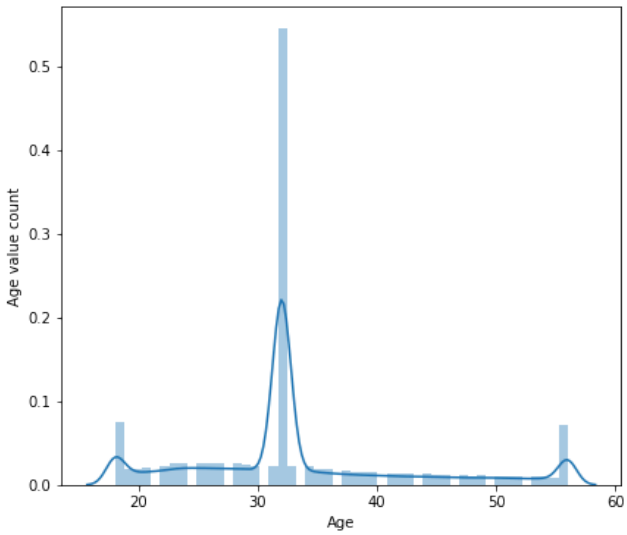
```
#Capping the outlier rows with Percentiles
upper_lim = users['Age'].quantile(.95)
lower_lim = users['Age'].quantile(.05)
users.loc[(users["Age"] > upper_lim),"Age"] = upper_lim
users.loc[(users["Age"] < lower_lim),"Age"] = lower_lim
```

```
#Boxplot and distplot of Age
plt.figure(figsize=(15,6))
plt.subplot(1, 2, 1)
fig = sns.boxplot(y=users["Age"])
fig.set_title('')
fig.set_ylabel("Age")

plt.subplot(1, 2, 2)
fig = sns.distplot(users["Age"].dropna())
fig.set_ylabel('Age value count')
fig.set_xlabel("Age")

plt.show()
```






```
users['Age'].describe()
```

```
count    278858.000000
mean         33.402570
std           9.523058
min          18.000000
25%          29.000000
50%          32.000000
75%          35.000000
max          56.000000
Name: Age, dtype: float64
```

```
users['locat']=users.Location.apply(lambda x: x.split(', '))
users['Country']=users.locat.apply(lambda x: x[2] if len(x)==3 else 'n/a')
users.drop('locat',axis=1,inplace=True)
```

Start coding or generate with AI.

Books Dataset

```
# Missing Value Count Function
def show_missing():
    missing = books.columns[books.isnull().any()].tolist()
    return missing

# Missing data counts and percentage
print('Missing Data Count')
print(books[show_missing()].isnull().sum().sort_values(ascending = False))
print('--'*50)
print('Missing Data Percentage')
print(round(books[show_missing()].isnull().sum().sort_values(ascending = False)/len(books
```

```
Missing Data Count
Image-URL-L    3
Publisher      2
Book-Author    1
dtype: int64
----------------------------------------------------------------------------------------
Missing Data Percentage
Image-URL-L    0.0
Publisher      0.0
Book-Author    0.0
dtype: float64
```

```
#Dropping Nulls of Books data as they are negligible
books.dropna(inplace=True)
```

```
#Unique year values
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,
       2010, 1908, 1906, 1935, 1806, 2021, '2000', '1995', '1999', '2004',
       '2003', '1990', '1994', '1986', '1989', '2002', '1981', '1993',
       '1983', '1982', '1976', '1991', '1977', '1998', '1992', '1996',
       '0', '1997', '2001', '1974', '1968', '1987', '1984', '1988',
       '1963', '1956', '1970', '1985', '1978', '1973', '1980', '1979',
       '1975', '1969', '1961', '1965', '1939', '1958', '1950', '1953',
       '1966', '1971', '1959', '1972', '1955', '1957', '1945', '1960',
       '1967', '1932', '1924', '1964', '2012', '1911', '1927', '1948',
       '1962', '2006', '1952', '1940', '1951', '1931', '1954', '2005',
       '1930', '1941', '1944', '1943', '1938', '1900', '1942', '1923',
       '1920', '1933', '1909', '1946', '2008', '1378', '2030', '1936',
```

```
       '1947', '2011', '2020', '1919', '1949', '1922', '1897', '2024',
       '1376', '1926', '2037'], dtype=object)
```

```
#converting Reviews type into integer
books['Year-Of-Publication'] = pd.to_numeric(books['Year-Of-Publication'])
```

```
#Capping the outlier rows with Percentiles
upper_lim = books['Year-Of-Publication'].quantile(.95)
lower_lim = books['Year-Of-Publication'].quantile(.05)
books.loc[(books["Year-Of-Publication"] > upper_lim),"Year-Of-Publication"] = upper_lim
books.loc[(books["Year-Of-Publication"] < lower_lim),"Year-Of-Publication"] = lower_lim
```

```
#Upper limit of year of publication feature
upper_lim
```

```
2003.0
```

```
#lower limit of year of publication feature
lower_lim
```

```
1976.0
```

Ratings Dataset

```
# Missing Value Count Function
def show_missing():
    missing = ratings.columns[ratings.isnull().any()].tolist()
    return missing

# Missing data counts and percentage
print('Missing Data Count')
print(ratings[show_missing()].isnull().sum().sort_values(ascending = False))
print('--'*50)
print('Missing Data Percentage')
print(round(ratings[show_missing()].isnull().sum().sort_values(ascending = False)/len(rat
```

```
Missing Data Count
Series([], dtype: float64)
----------------------------------------------------------------------------------------
Missing Data Percentage
Series([], dtype: float64)
```

Cleaning Data

```
#users data head
users.head(1)
```

	User-ID	Location	Age	Country
0	1	nyc, new york, usa	32.0	usa

```
#shape of users dataframe
users.shape
```

```
(278858, 4)
```

```
#head of rating dataframe
ratings.head(1)
```

	User-ID	ISBN	Book-Rating
0	276725	034545104X	0

```
#shape of rating dataframe
ratings.shape
```

```
(1149780, 3)
```

Merging users and ratings dataset on User-ID column

```
#Merging users and rating dataframe
data_users_ratings=pd.merge(users,ratings, on='User-ID')
```

Merging combined data of users and ratings with books data on ISBN column

```
#Merging both data
merged=pd.merge(books,data_users_ratings, on='ISBN')
```

```
#info
merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1031129 entries, 0 to 1031128
Data columns (total 13 columns):
 #   Column               Non-Null Count    Dtype 
---  ------               --------------    ----- 
 0   ISBN                 1031129 non-null  object
 1   Book-Title           1031129 non-null  object
 2   Book-Author          1031129 non-null  object
 3   Year-Of-Publication  1031129 non-null  int64 
 4   Publisher            1031129 non-null  object
 5   Image-URL-S          1031129 non-null  object
 6   Image-URL-M          1031129 non-null  object
 7   Image-URL-L          1031129 non-null  object
 8   User-ID              1031129 non-null  int64 
 9   Location             1031129 non-null  object
```

```
 10  Age                  1031129 non-null  float64
 11  Country              1031129 non-null  object
 12  Book-Rating          1031129 non-null  int64
dtypes: float64(1), int64(3), object(9)
memory usage: 110.1+ MB
```

```
# Missing Value Count Function
def show_missing():
    missing = ratings.columns[ratings.isnull().any()].tolist()
    return missing

# Missing data counts and percentage
print('Missing Data Count')
print(ratings[show_missing()].isnull().sum().sort_values(ascending = False))
print('--'*50)
print('Missing Data Percentage')
print(round(ratings[show_missing()].isnull().sum().sort_values(ascending = False)/len(rat
```

```
Missing Data Count
Series([], dtype: float64)
------------------------------------------------------------------------------------
Missing Data Percentage
Series([], dtype: float64)
```

```
#head of all 3 merged dataframe
merged.head()
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/in
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/in
2	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/in
3	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/in
4	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/in

```
#dropping unnecessary features
merged.drop(columns=['Image-URL-S','Image-URL-M','Image-URL-L'], axis=1,inplace=True)


#Head of merged data frame
merged.head()
```

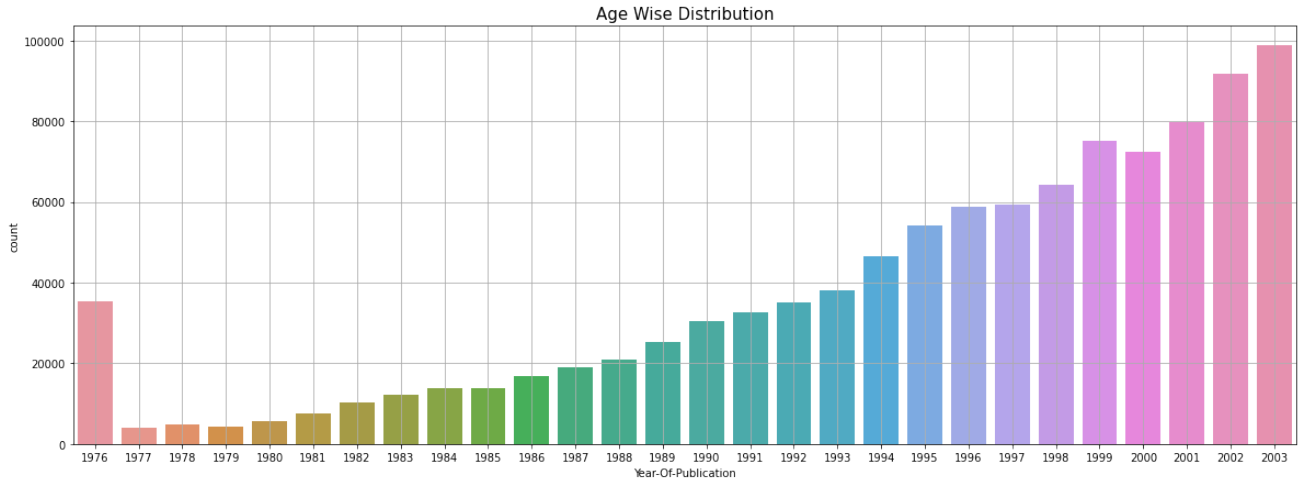
	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	User-ID	Location	Age	C
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	2	stockton, california, usa	18.0	
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	8	timmins, ontario, canada	32.0	
2	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	11400	ottawa, ontario, canada	49.0	
3	0002005018	Clara	Richard	2001	HarperFlamingo	11676	n/a, n/a,	32.0	

```
np.set_printoptions(threshold=10)


# unique values of ISBN feature
merged.ISBN.unique()
```

```
array(['0195153448', '0002005018', '0060973129', ..., '006008667X',
       '0192126040', '0767409752'], dtype=object)
```

```
#barplot of age and its counts
plt.figure(figsize=(20,7))
sns.countplot(x = merged['Year-Of-Publication'])
plt.title('Age Wise Distribution',fontsize=15)
plt.grid()
plt.show()
```




```
#Unique values of Year-Of-Publication'
merged['Year-Of-Publication'].unique()
```

```
array([2002, 2001, 1991, ..., 1989, 1984, 1977])
```

```
#pd.set_option('display.max_rows', 50000)
```

```
#Country value counts
merged.Country.value_counts()
```

```
usa                                   745812
canada                                 92954
n/a                                    37573
united kingdom                         32007
germany                                27654
                                       ...
hungary"                                   1
c                                          1
samoa                                      1
zambia                                     1
usa (currently living in england)          1
Name: Country, Length: 281, dtype: int64
```

Duplicates

```
#checking duplicates
merged.duplicated().any()
```

```
False
```

EDA

```
#Head of merged dataframe
merged.head(2)
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	User-ID	Location	Age	C
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	2	stockton, california, usa	18.0	
			Richard				timmins		

Top 10 sold books

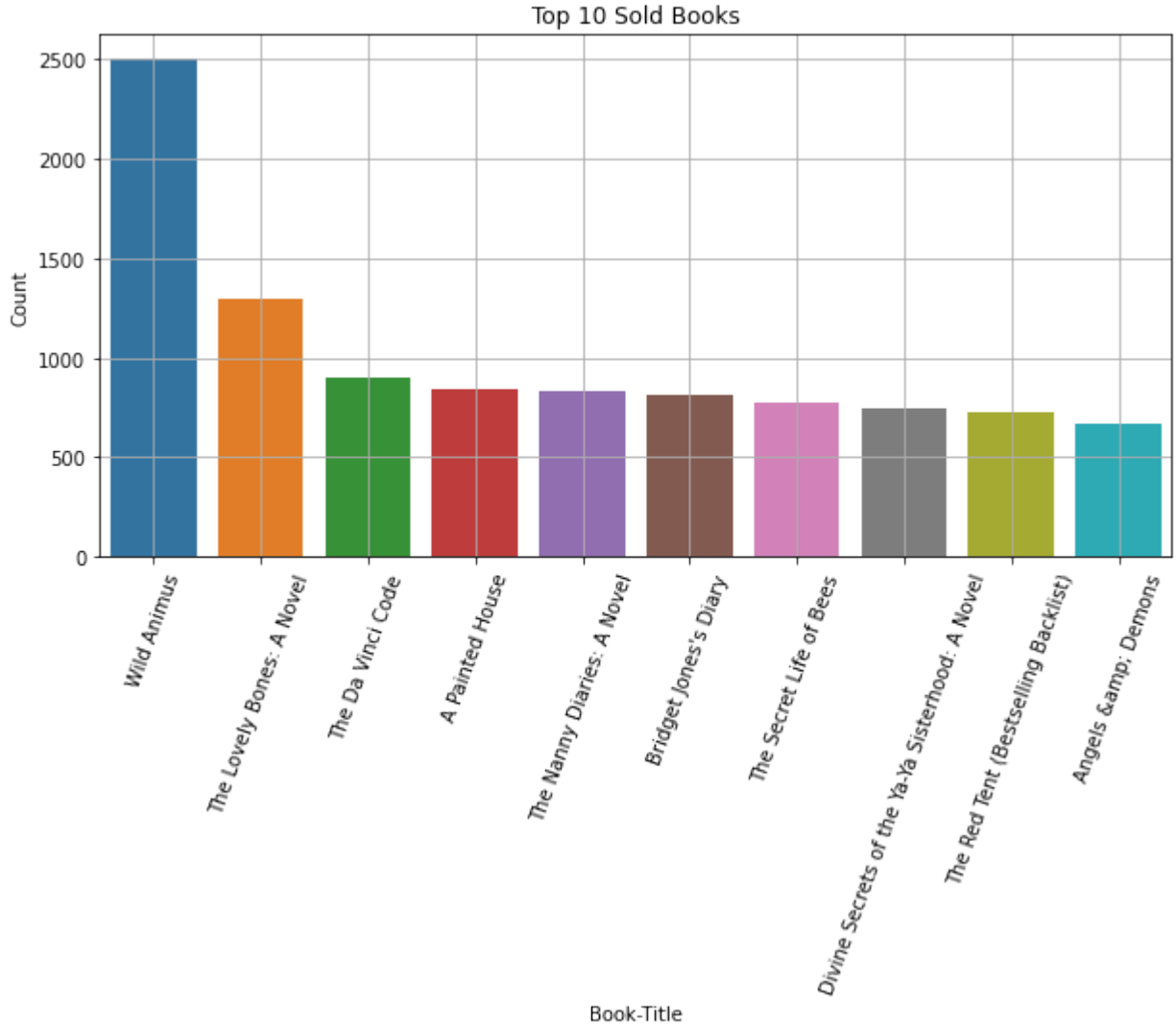
```
# Top 10 sold Books
Top10_Book=merged['Book-Title'].value_counts().reset_index().head(10)
Top10_Book.rename(columns={'index':'Book-Title','Book-Title':'Count'},inplace=True)
Top10_Book
```

	Book-Title	Count
0	Wild Animus	2502
1	The Lovely Bones: A Novel	1295
2	The Da Vinci Code	898
3	A Painted House	838
4	The Nanny Diaries: A Novel	828
5	Bridget Jones's Diary	815
6	The Secret Life of Bees	774
7	Divine Secrets of the Ya-Ya Sisterhood: A Novel	740
8	The Red Tent (Bestselling Backlist)	723
9	Angels & Demons	670

Wild Animus is the best-selling book

```
#barplot of Top 10 sold Books
plt.rcParams['figure.figsize'] = (10, 5)
sns.barplot(Top10_Book['Book-Title'],Top10_Book['Count'])
plt.xticks(rotation=70, horizontalalignment="center")
plt.grid()
plt.title('Top 10 Sold Books')
```

```
Text(0.5, 1.0, 'Top 10 Sold Books')
```



books with same title but different author

```
#books with same title but different author
books['Book-Title'].value_counts()
```

```
Selected Poems
27
Little Women
24
Wuthering Heights
21
Dracula
20
The Secret Garden
20

..
```

```
On a Clear Day You Can See General Motors: John Z. De Lorean's Look Inside the
Automotive Giant                                        1
What Every Kid Should Know
1
The Seventh Enemy (A Brady Coyne Mystery)
1
A Brace of Skeet
1
A Guided Tour of Rene Descartes' Meditations on First Philosophy with Complete
Translations of the Meditations by Ronald Rubin     1
Name: Book-Title, Length: 242130, dtype: int64
```

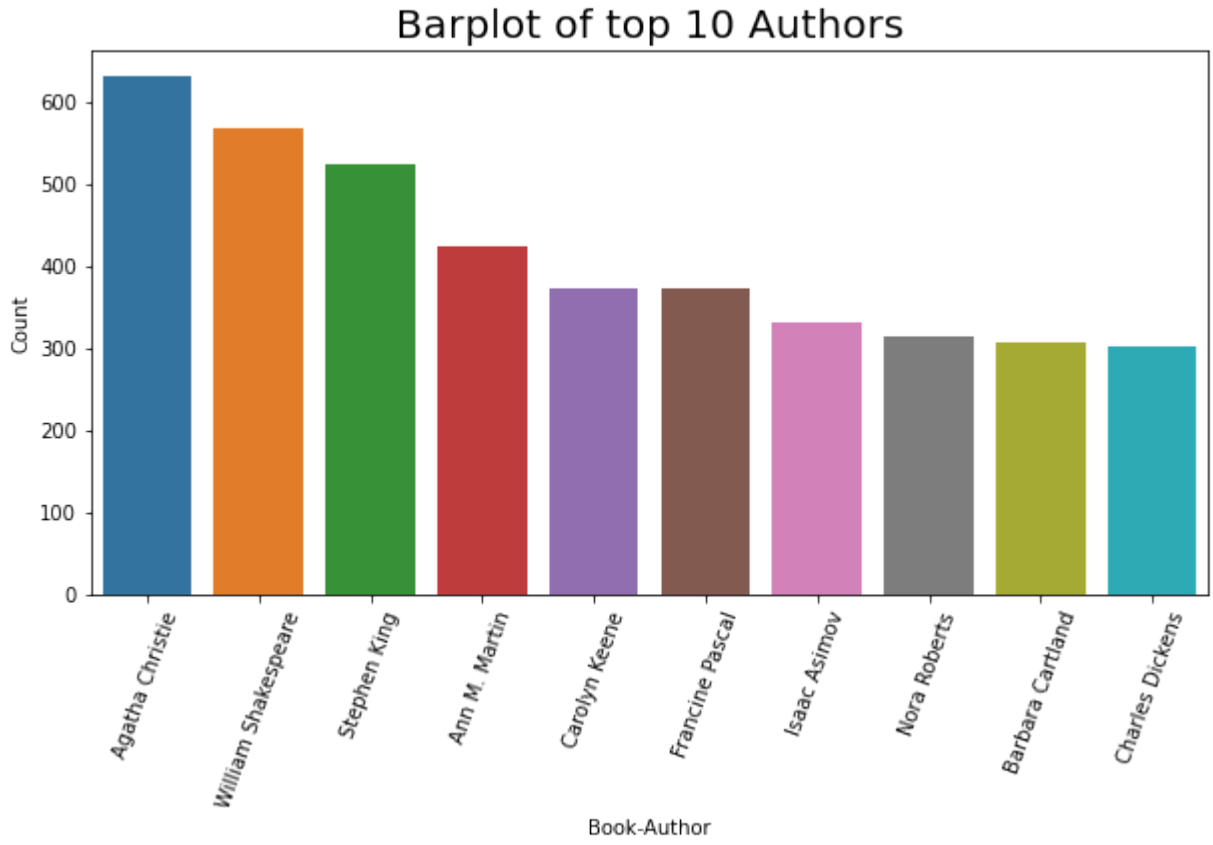
Top 10 author

```
#Top 10 author with most books written
Top10_author=books['Book-Author'].value_counts().reset_index().head(10)
Top10_author.rename(columns={'index':'Book-Author','Book-Author':'Count'},inplace=True)
Top10_author
```

	Book-Author	Count
0	Agatha Christie	632
1	William Shakespeare	567
2	Stephen King	524
3	Ann M. Martin	423
4	Carolyn Keene	373
5	Francine Pascal	372
6	Isaac Asimov	330
7	Nora Roberts	315
8	Barbara Cartland	307
9	Charles Dickens	302

```
#barplot of top 10 Authors with most books written
sns.barplot(x="Book-Author",y="Count",data=Top10_author)
plt.xticks(rotation=70, horizontalalignment="center")
plt.title("Barplot of top 10 Authors",fontsize=20)
```

```
Text(0.5, 1.0, 'Barplot of top 10 Authors')
```

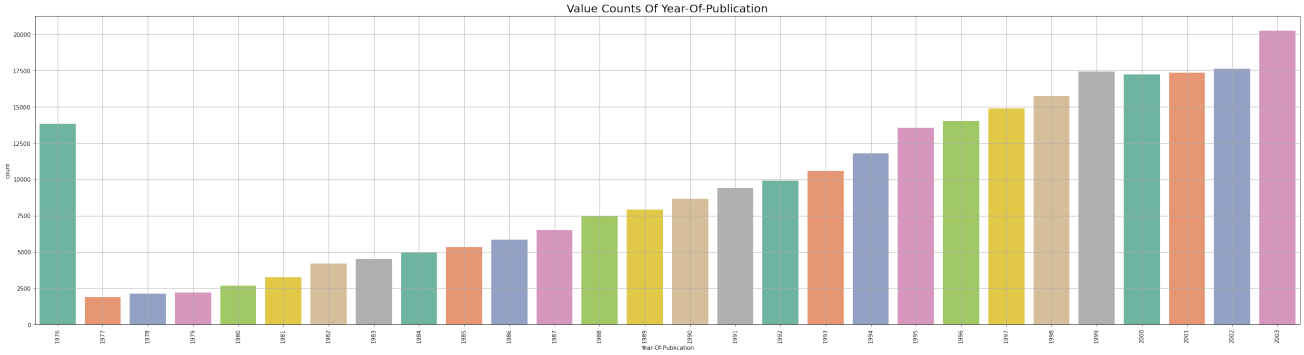


Value Counts Of Year-Of-Publication

```
#Value Counts Of Year-Of-Publication
plt.rcParams['figure.figsize'] = (40,10)
fig = sns.countplot(x=books['Year-Of-Publication'],palette="Set2")
plt.xticks(rotation=90, horizontalalignment="center")
plt.grid()

plt.title("Value Counts Of Year-Of-Publication",fontsize=20)

plt.show()
```



Top 10 publisher

```
#Top 10 publisher
Top10_publisher=books['Publisher'].value_counts().reset_index().head(10)
Top10_publisher.rename(columns={'index':'Publisher','Publisher':'Count'},inplace=True)
Top10_publisher
```

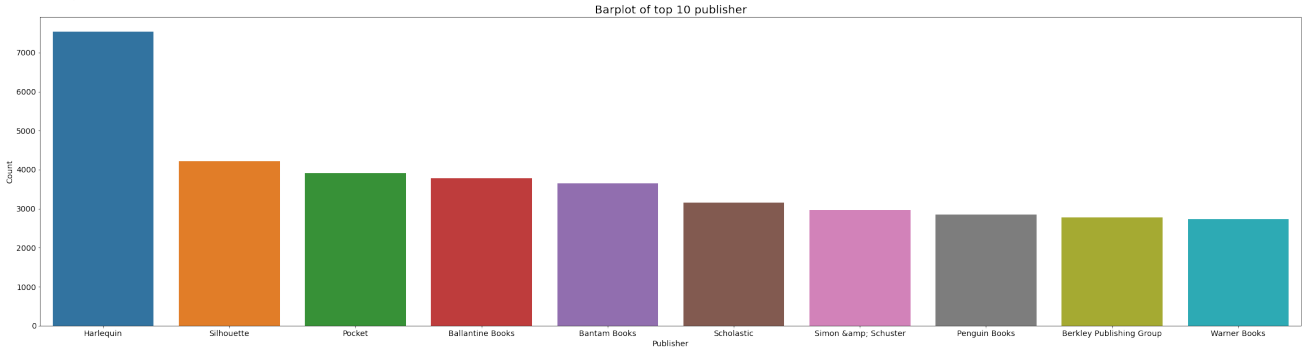
	Publisher	Count
0	Harlequin	7535
1	Silhouette	4220
2	Pocket	3905
3	Ballantine Books	3783
4	Bantam Books	3646
5	Scholastic	3160
6	Simon & Schuster	2971
7	Penguin Books	2844
8	Berkley Publishing Group	2771
9	Warner Books	2727

Harlequin published the most books

```
#barplot of top 10 publisher
sns.barplot(x="Publisher",y="Count",data=Top10_publisher)

plt.title("Barplot of top 10 publisher",fontsize=20)
```

```
Text(0.5, 1.0, 'Barplot of top 10 publisher')
```

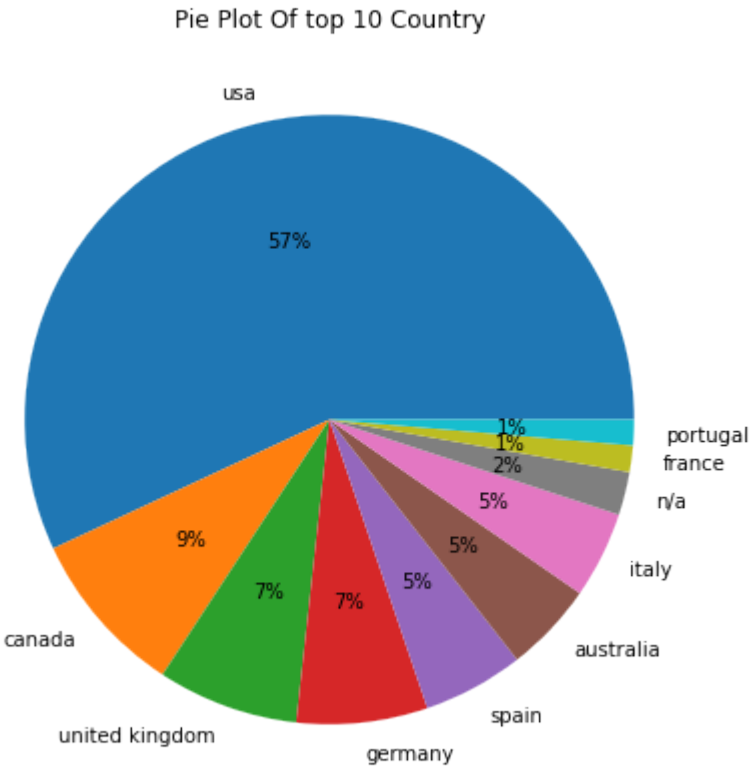


Top 10 Country

```
#value counts of country
Top10_Country=users['Country'].value_counts().reset_index().head(10)
Top10_Country.rename(columns={'index':'Country','Country':'Count'},inplace=True)
Top10_Country
```

	Country	Count
0	usa	139421
1	canada	21601
2	united kingdom	18314
3	germany	17024
4	spain	13096
5	australia	11730
6	italy	11244
7	n/a	5617
8	france	3440
9	portugal	3312

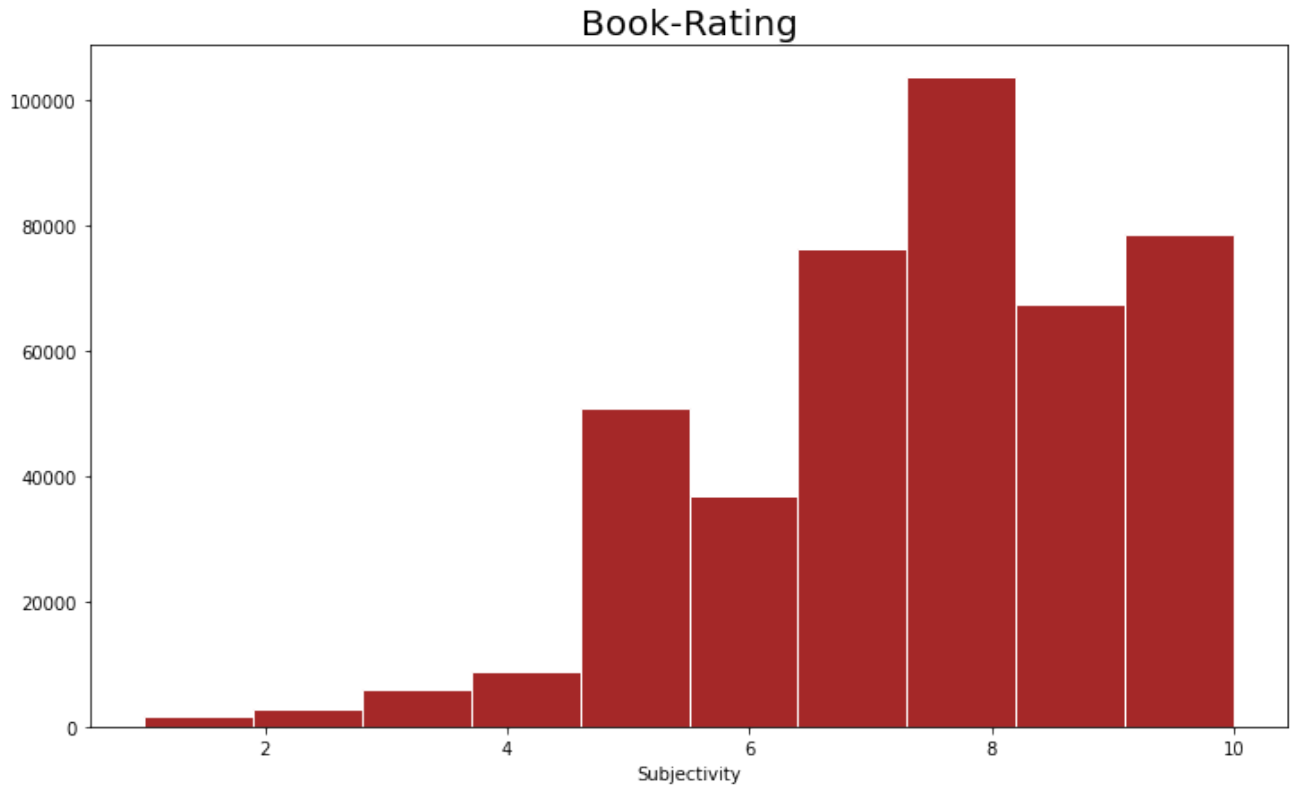
```
#Pie Plot Of Top10_Country
fig= plt.figure(figsize=(20,7))
plt.pie(Top10_Country['Count'],labels=Top10_Country['Country'],autopct='%.0f%%')
plt.title('Pie Plot Of top 10 Country')
plt.show()
```



More than 50% readers are from USA

Histogram Of Book-Ratings(Excluding 0 rating count)

```
#Histogram Of Book-Rating excluding rating equal to zero
ratings= ratings[ratings['Book-Rating'] != 0]
plt.figure(figsize=(12,7))
plt.xlabel("Subjectivity")
plt.title("Book-Rating",fontsize=20)
plt.hist(ratings['Book-Rating'],color="brown",edgecolor="white")
plt.show()
```




```
#Description of book rating
ratings['Book-Rating'].describe()
```

```
count    433671.000000
mean          7.601066
std           1.843798
min           1.000000
25%           7.000000
50%           8.000000
75%           9.000000
max          10.000000
Name: Book-Rating, dtype: float64
```

Book-Ratings are negatively distributed with median rating of 8

Collaborative filtering models Used

Collaborative filtering methods Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called “user-item interactions matrix”.

Data Preparation

```
#Rating head
ratings.head(1)
```

	User-ID	ISBN	Book-Rating
1	276726	0155061224	5

```
#Rating data with exclusion of Books with rating 0
ratings= ratings[ratings['Book-Rating'] != 0]


#Merging dataframe rating and books on ISBN
df=pd.merge(ratings,books, on='ISBN')
```

Implementing KNN

Books which are rated by atleast 10 users

```
# Books interactionn count
books_interactions_count_df = df.groupby(['ISBN', 'User-ID']).size().groupby('ISBN').size
print('# of books: %d' % len(books_interactions_count_df))

# Books with enough interactions
books_with_enough_interactions_df = books_interactions_count_df[books_interactions_count_
print('# of books with at least 10 interactions: %d' % len(books_with_enough_interactions
print(books_with_enough_interactions_df.head(5))
```

```
# of books: 149832
# of books with at least 10 interactions: 5444
         ISBN
0  0002558122
1  000648302X
2  000649840X
3  0006547834
4  0006550576
```

Users which have rated atleast 25 different books

```
# Users interactionn count
users_interactions_count_df = df.groupby(['User-ID', 'ISBN']).size().groupby('User-ID').s
print('# of users: %d' % len(users_interactions_count_df))

# Users with enough interactions
users_with_enough_interactions_df = users_interactions_count_df[users_interactions_count_
print('# of users with at least 25 interactions: %d' % len(users_with_enough_interactions
print(users_with_enough_interactions_df.head(5))
```

```
# of users: 68091
# of users with at least 25 interactions: 2604
```

```
   User-ID
0      254
1      507
2      638
3      643
4      651
```

```
#Users with enough interactions
print('# of interactions: %d' % len(df))
interactions_from_selected_users_df = df.merge(users_with_enough_interactions_df,
               how = 'right',
               on = 'User-ID')
print('# of interactions from users with at least 25 interactions: %d' % len(interactions
```

```
     # of interactions: 383838
     # of interactions from users with at least 25 interactions: 202455
```

Dataframe of Users and Books with enough interactions

```
#Users and Books with enough interactions
print('# of interactions: %d' % len(df))
interactions_from_selected_books_and_users_df= interactions_from_selected_users_df.merge(
print('# of interactions from users with at least 25 interactions and books with at least
```

```
     # of interactions: 383838
     # of interactions from users with at least 25 interactions and books with at least 10
```

```
#interactions from selected books and users dataframe
interactions_from_selected_books_and_users_df.head(5)
```

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	
0	254	0671021001	7	She's Come Undone (Oprah's Book Club)	Wally Lamb	1998	Pocket	http://images.ama
1	2977	0671021001	9	She's Come Undone (Oprah's Book Club)	Wally Lamb	1998	Pocket	http://images.ama
2	6563	0671021001	2	She's Come Undone (Oprah's Book Club)	Wally Lamb	1998	Pocket	http://images.ama
3	8253	0671021001	10	She's Come Undone (Oprah's Book Club)	Wally Lamb	1998	Pocket	http://images.ama
4	11718	0671021001	8	She's Come Undone (Oprah's Book Club)	Wally Lamb	1998	Pocket	http://images.ama

```
#Shape of interactions from selected books and users dataframe
interactions_from_selected_books_and_users_df.shape
```

```
(63541, 10)
```

```
#aggregating all the interactions of users and applying log transformation to rating
import math
def smooth_user_preference(x):
    return math.log(1+x, 2)

interactions_full_df1 = interactions_from_selected_books_and_users_df.groupby(['User-ID',
print('# of unique user/item interactions: %d' % len(interactions_full_df1))
interactions_full_df = interactions_from_selected_books_and_users_df.groupby(['User-ID',
print('# of unique user/item interactions: %d' % len(interactions_full_df))
interactions_full_df.head(5)
```

```
# of unique user/item interactions: 63228
# of unique user/item interactions: 63541
```

	User-ID	ISBN	Book-Rating
0	254	0060934700	3.321928
1	254	0064471047	3.000000
2	254	0066238501	2.584963
3	254	0142001740	3.321928
4	254	0380730448	3.169925

```
#Creating a sparse pivot table
df_user_item_matrix = interactions_full_df.pivot(index='ISBN',columns='User-ID',values='B
user_item_matrix_sparse = csr_matrix(df_user_item_matrix.values)
df_user_item_matrix1 = interactions_full_df1.pivot(index='User-ID',columns='Book-Title',v
df_user_item_matrix1=df_user_item_matrix1.transpose()
user_item_matrix_sparse1 = csr_matrix(df_user_item_matrix1.values)
user_item_matrix_sparse1=csr_matrix(df_user_item_matrix1.values)
```

Model Building

```
#Fitting Model
model = NearestNeighbors(n_neighbors=30, metric='cosine', algorithm='brute', n_jobs=-1)

model.fit(user_item_matrix_sparse1)
```

```
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=30)
```

Recommendations for randomly selected book

```
#Recommendations for randomly selected book
query_index = np.random.choice(df_user_item_matrix1.shape[0])
distances, indices = model.kneighbors(df_user_item_matrix1.iloc[query_index, :].values.re

for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for Book {0}:\n'.format(df_user_item_matrix1.index[query_i
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, df_user_item_matrix1.index[indi
```

```
Recommendations for Book The Genesis Code:

1: The Worst-Case Scenario Survival Handbook, with distance of 0.6519719690823778:
2: Undue Influence, with distance of 0.6618984157764958:
3: Finding Fish: A Memoir, with distance of 0.6748876904816117:
4: Due di due (Bestsellers), with distance of 0.6748876904816117:
5: Schachnovelle, with distance of 0.6748876904816117:
6: Schnee, der auf Zedern fÃ?Â¤llt., with distance of 0.6748876904816117:
7: Jade Peony, with distance of 0.6748876904816117:
8: Rama Revealed (Bantam Spectra Book), with distance of 0.6748876904816117:
```

```
9: Artemis Fowl., with distance of 0.6748876904816117:
10: Ravage, with distance of 0.6748876904816117:
11: Eine Frau, Eine Wohnung, Ein Roman, with distance of 0.6748876904816117:
12: Oceano Mare, with distance of 0.6748876904816117:
13: Das Echo., with distance of 0.6748876904816117:
14: Garden of Rama, with distance of 0.6748876904816117:
15: Buddha of Suburbia, with distance of 0.6748876904816117:
```

Model building and recommendation for perticular book

```
#Model building and recommendation for perticular book
model = NearestNeighbors(n_neighbors=30, metric='cosine', algorithm='brute', n_jobs=-1)

model.fit(user_item_matrix_sparse)

index_to_book = dict()

df_titles_book = df.set_index('ISBN').loc[df_user_item_matrix.index]

count = 0

for index, row in df_titles_book.iterrows():

    index_to_book[count]=row['Book-Title']

    count +=1


def recommender(model, user_item_matrix_sparse, df_book, number_of_recommendations, book_

    main_title = index_to_book[book_index]

    dist, ind = model.kneighbors(user_item_matrix_sparse[book_index], n_neighbors=number_

    dist = dist[0].tolist()

    ind = ind[0].tolist()

    titles = []

    for index in ind:

        titles.append(index_to_book[index])

    recommendations = list(zip(titles,dist))

    # sort recommendations

    recommendations_sorted = sorted(recommendations, key = lambda x:x[1])

    # reverse recommendations, leaving out the first element

    recommendations_sorted.reverse()

    recommendations_sorted = recommendations_sorted[:-1]

    print("Recommendations for Book {}: ".format(main_title))

    count = 0

    for (title, distance) in recommendations_sorted:

        count += 1

        print('{}. {}, recommendation score = {}'.format(count, title, round(distance,5))
```

```
recommender(model, user_item_matrix_sparse, df, 10, 10)
```

```
Recommendations for Book Before and After:
1. Tishomingo Blues, recommendation score = 0.81735
2. Waiting : The True Confessions of a Waitress, recommendation score = 0.81501
3. Soul Mountain, recommendation score = 0.80676
4. Perfect Murder, Perfect Town, recommendation score = 0.80452
5. Politically Correct Holiday Stories: For an Enlightened Yuletide Season, recommend
6. A Promising Man (and About Time, Too), recommendation score = 0.8012
7. When He Was Wicked (Bridgerton Family Series), recommendation score = 0.80114
8. All-American Girl, recommendation score = 0.79553
9. Night Watch, recommendation score = 0.79382
10. A Cook's Tour : Global Adventures in Extreme Cuisines, recommendation score = 0.7
```

SVD

```
minimum_rating = min(interactions_full_df['Book-Rating'].values)

maximum_rating = max(interactions_full_df['Book-Rating'].values)


reader = Reader(rating_scale=(minimum_rating,maximum_rating))

data = Dataset.load_from_df(interactions_full_df[['User-ID', 'ISBN', 'Book-Rating']], rea
```

Train Test Split And Model Building

```
from surprise import accuracy, Dataset, SVD
from surprise.model_selection import train_test_split


# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=0.25)

# We'll use the famous SVD algorithm
algo = SVD()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)

# Then compute MAE
accuracy.mae(predictions)
```

```
RMSE: 0.3049
MAE:  0.2179
0.21792023149026263
```

```
interactions_full_df.head(1)
```

	User-ID	ISBN	Book-Rating
0	254	0060934700	3.321928

```
user_id = '254'

isbn = '0060934700'

prediction = algo.predict(uid=user_id, iid=isbn)

print("Predicted rating of user with id {} for movie with id {}: {}".format(user_id, isbn
```

```
Predicted rating of user with id 254 for movie with id 0060934700: 3.134
```

Log transforamtion is applied to ratings

predictions

```
# Predictions- actual and estimated
predictions
```

```
[Prediction(uid=91103, iid='0385505833', r_ui=3.1699250014423126,
est=2.6435948250746617, details={'was_impossible': False}),
 Prediction(uid=11676, iid='0345402308', r_ui=3.0, est=3.008480935223256,
details={'was_impossible': False}),
 Prediction(uid=172742, iid='0449219364', r_ui=3.4594316186372978,
est=3.320280468405256, details={'was_impossible': False}),
 Prediction(uid=220278, iid='043942089X', r_ui=3.1699250014423126,
est=3.194934430207404, details={'was_impossible': False}),
 Prediction(uid=126604, iid='0060932139', r_ui=3.0, est=2.8455181642563177,
details={'was_impossible': False}),
 Prediction(uid=254201, iid='0671739743', r_ui=3.4594316186372978,
est=3.3197414235742295, details={'was_impossible': False}),
 Prediction(uid=220688, iid='1558531025', r_ui=3.3219280948873626,
est=3.4594316186372978, details={'was_impossible': False}),
 Prediction(uid=47856, iid='0451451430', r_ui=3.0, est=2.7429160949633324,
details={'was_impossible': False}),
 Prediction(uid=129716, iid='0061020680', r_ui=3.3219280948873626,
est=3.452394599199206, details={'was_impossible': False}),
 Prediction(uid=163570, iid='0451172817', r_ui=3.4594316186372978,
est=3.374233954545587, details={'was_impossible': False}),
 Prediction(uid=143175, iid='0425133516', r_ui=3.3219280948873626,
est=3.0558817571042565, details={'was_impossible': False}),
 Prediction(uid=225087, iid='0060921145', r_ui=3.4594316186372978,
est=3.264019559952603, details={'was_impossible': False}),
 Prediction(uid=140069, iid='0446603775', r_ui=2.807354922057604,
est=3.0473785143585954, details={'was_impossible': False}),
```

```
 Prediction(uid=38273, iid='0312976275', r_ui=2.584962500721156,
est=2.986520541008787, details={'was_impossible': False}),
 Prediction(uid=158295, iid='0553275704', r_ui=2.807354922057604,
est=2.830312489713341, details={'was_impossible': False}),
 Prediction(uid=193898, iid='0345453816', r_ui=3.3219280948873626,
est=3.2663426515592886, details={'was_impossible': False}),
 Prediction(uid=123257, iid='0671759329', r_ui=3.1699250014423126,
est=3.2013517172307457, details={'was_impossible': False}),
 Prediction(uid=11224, iid='0061009059', r_ui=3.0, est=2.9158600046238674,
details={'was_impossible': False}),
 Prediction(uid=190925, iid='0395927218', r_ui=3.1699250014423126,
est=3.191457964812869, details={'was_impossible': False}),
 Prediction(uid=157273, iid='0440419468', r_ui=2.584962500721156,
est=3.1164308920246553, details={'was_impossible': False}),
 Prediction(uid=277427, iid='0441627404', r_ui=3.4594316186372978,
est=3.3059879602060556, details={'was_impossible': False}),
 Prediction(uid=251844, iid='0156006391', r_ui=3.1699250014423126,
est=3.0900698749558706, details={'was_impossible': False}),
 Prediction(uid=170077, iid='0345432401', r_ui=2.584962500721156,
est=3.0748015250777474, details={'was_impossible': False}),
 Prediction(uid=157799, iid='0671695126', r_ui=3.0, est=3.0806689697011422,
details={'was_impossible': False}),
 Prediction(uid=154345, iid='1573226882', r_ui=3.1699250014423126,
est=2.8684201915128313, details={'was_impossible': False}),
 Prediction(uid=239736, iid='0836213319', r_ui=3.4594316186372978,
est=3.20167291151265, details={'was_impossible': False}),
 Prediction(uid=115435, iid='0553375407', r_ui=2.807354922057604,
est=3.2032385611698, details={'was_impossible': False}),
 Prediction(uid=105517, iid='0316666343', r_ui=3.3219280948873626,
est=2.8148003591568327, details={'was_impossible': False}),
 Prediction(uid=146348, iid='0553569074', r_ui=3.1699250014423126,
est=3.3907440975125613, details={'was_impossible': False}),
```

SVDpp

Train Test Split And Model Building

```
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=0.25)

# We'll use the famous SVD algorithm
algo = SVDpp()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)

# Then compute MAE
accuracy.mae(predictions)
```

```
RMSE: 0.3003
MAE:  0.2102
0.21024747039974506
```

```
interactions_full_df.head(1)
```

	User-ID	ISBN	Book-Rating
0	254	0060934700	3.321928

```
user_id = '254'

isbn = '0060934700'

prediction = algo.predict(uid=user_id, iid=isbn)

print("Predicted rating of user with id {} for movie with id {}: {}".format(user_id, isbn
```

```
Predicted rating of user with id 254 for movie with id 0060934700: 3.151
```

```
# Predictions- actual and estimated
predictions
```

```
[Prediction(uid=47316, iid='0156528207', r_ui=3.4594316186372978,
est=3.3704100060527034, details={'was_impossible': False}),
 Prediction(uid=270713, iid='0375705171', r_ui=3.3219280948873626,
est=3.1507594204638076, details={'was_impossible': False}),
 Prediction(uid=208568, iid='0345339681', r_ui=3.1699250014423126,
est=2.9448033758715018, details={'was_impossible': False}),
 Prediction(uid=150979, iid='0385730586', r_ui=3.3219280948873626,
est=3.231151342292123, details={'was_impossible': False}),
 Prediction(uid=3363, iid='0446672211', r_ui=3.4594316186372978,
est=3.4594316186372978, details={'was_impossible': False}),
 Prediction(uid=68984, iid='0842329293', r_ui=3.1699250014423126,
est=3.020481388841952, details={'was_impossible': False}),
 Prediction(uid=230522, iid='074322535X', r_ui=3.1699250014423126,
est=3.2230782572509247, details={'was_impossible': False}),
 Prediction(uid=270605, iid='0452281032', r_ui=3.0, est=2.910570288467437,
details={'was_impossible': False}),
 Prediction(uid=30273, iid='0373250142', r_ui=3.1699250014423126,
est=3.2001357545204265, details={'was_impossible': False}),
 Prediction(uid=239423, iid='0345416260', r_ui=3.4594316186372978,
est=3.090662597210539, details={'was_impossible': False}),
 Prediction(uid=68760, iid='0373484224', r_ui=2.0, est=3.014884638463251,
details={'was_impossible': False}),
 Prediction(uid=194676, iid='0312974256', r_ui=3.1699250014423126,
est=2.978206946645483, details={'was_impossible': False}),
 Prediction(uid=102647, iid='0061094129', r_ui=3.1699250014423126,
est=3.0652288496741136, details={'was_impossible': False}),
 Prediction(uid=266466, iid='0028604199', r_ui=3.3219280948873626,
est=2.863638398441883, details={'was_impossible': False}),
 Prediction(uid=235105, iid='0525947299', r_ui=3.3219280948873626,
est=3.2326666043725236, details={'was_impossible': False}),
 Prediction(uid=11676, iid='0440216842', r_ui=3.0, est=2.9860268544076702,
details={'was_impossible': False}),
```

```
 Prediction(uid=264996, iid='0671003755', r_ui=3.1699250014423126,
est=3.008886338547793, details={'was_impossible': False}),
 Prediction(uid=23547, iid='0140262032', r_ui=2.321928094887362,
est=3.126564544405117, details={'was_impossible': False}),
 Prediction(uid=117873, iid='0312171838', r_ui=3.1699250014423126,
est=3.141966258877104, details={'was_impossible': False}),
 Prediction(uid=69156, iid='0743222008', r_ui=2.321928094887362,
est=3.180485017418699, details={'was_impossible': False}),
 Prediction(uid=204591, iid='0553569910', r_ui=3.0, est=3.108127071687056,
details={'was_impossible': False}),
 Prediction(uid=11676, iid='0553284789', r_ui=2.321928094887362,
est=3.124353175969844, details={'was_impossible': False}),
 Prediction(uid=26593, iid='0446611212', r_ui=3.4594316186372978,
est=3.282091094338445, details={'was_impossible': False}),
 Prediction(uid=227447, iid='0920668372', r_ui=3.4594316186372978,
est=3.244117534566919, details={'was_impossible': False}),
 Prediction(uid=78553, iid='0671737791', r_ui=3.3219280948873626,
est=3.3612404958833015, details={'was_impossible': False}),
 Prediction(uid=222296, iid='0312150601', r_ui=3.1699250014423126,
est=2.9959670925425743, details={'was_impossible': False}),
 Prediction(uid=275970, iid='0446391301', r_ui=3.0, est=3.2414955095840994,
details={'was_impossible': False}),
 Prediction(uid=133689, iid='0440170796', r_ui=3.1699250014423126,
est=2.943411228123349, details={'was_impossible': False}),
 Prediction(uid=129368, iid='0385420161', r_ui=3.3219280948873626,
```

Start coding or generate with AI.

NMF

Train Test Split And Model Building

```
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=0.25)

# We'll use the famous SVD algorithm
algo = NMF()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)

# Then compute MAE
accuracy.mae(predictions)
```

```
RMSE: 0.3382
MAE:  0.2436
0.24361194608328227
```

```
interactions_full_df.head(1)
```

	User-ID	ISBN	Book-Rating
0	254	0060934700	3.321928

```
user_id = '254'

isbn = '0060934700'

prediction = algo.predict(uid=user_id, iid=isbn)

print("Predicted rating of user with id {} for movie with id {}: {}".format(user_id, isbn
```

```
Predicted rating of user with id 254 for movie with id 0060934700: 3.122
```

```
# Predictions- actual and estimated
predictions
```

```
[Prediction(uid=197687, iid='0373825013', r_ui=3.0, est=2.901101017358445,
details={'was_impossible': False}),
 Prediction(uid=218121, iid='0449208281', r_ui=3.4594316186372978,
est=3.2676018404049723, details={'was_impossible': False}),
 Prediction(uid=11629, iid='0345368991', r_ui=2.584962500721156,
est=2.8598943139607016, details={'was_impossible': False}),
 Prediction(uid=250962, iid='0345351525', r_ui=3.4594316186372978,
est=3.1800068432871287, details={'was_impossible': False}),
 Prediction(uid=117384, iid='0385484518', r_ui=2.584962500721156,
est=3.1580377279818603, details={'was_impossible': False}),
 Prediction(uid=86202, iid='068484267X', r_ui=2.584962500721156,
est=3.1857766129613676, details={'was_impossible': False}),
 Prediction(uid=6238, iid='0804115613', r_ui=3.3219280948873626,
est=3.4594316186372978, details={'was_impossible': False}),
 Prediction(uid=52853, iid='0440212561', r_ui=3.3219280948873626,
est=3.1995686703536883, details={'was_impossible': False}),
 Prediction(uid=210587, iid='0452284449', r_ui=3.0, est=2.9389260561206036,
details={'was_impossible': False}),
 Prediction(uid=53220, iid='0425182908', r_ui=3.1699250014423126,
est=2.771120509294062, details={'was_impossible': False}),
 Prediction(uid=123883, iid='0515136530', r_ui=2.584962500721156,
est=2.6230475523745977, details={'was_impossible': False}),
 Prediction(uid=57234, iid='0671250671', r_ui=3.1699250014423126,
est=2.8752458468265107, details={'was_impossible': False}),
 Prediction(uid=113904, iid='0553292722', r_ui=3.0, est=2.9269090550372123,
details={'was_impossible': False}),
 Prediction(uid=251422, iid='0440222656', r_ui=3.0, est=3.0849451089359827,
details={'was_impossible': False}),
 Prediction(uid=110887, iid='044661162X', r_ui=2.584962500721156,
est=2.2800607609826153, details={'was_impossible': False}),
 Prediction(uid=259259, iid='034538475X', r_ui=3.4594316186372978,
est=3.145830406089521, details={'was_impossible': False}),
 Prediction(uid=213126, iid='0345417623', r_ui=3.0, est=3.172068627925214,
details={'was_impossible': False}),
 Prediction(uid=88693, iid='0671004549', r_ui=3.0, est=3.0851103349998255,
details={'was_impossible': False}),
 Prediction(uid=117384, iid='0451521285', r_ui=2.584962500721156,
```

```
est=3.0767173017961342, details={'was_impossible': False}),
 Prediction(uid=234828, iid='0345384911', r_ui=3.1699250014423126,
est=2.933023718722259, details={'was_impossible': False}),
 Prediction(uid=137589, iid='0440213290', r_ui=3.1699250014423126,
est=2.988456332603984, details={'was_impossible': False}),
 Prediction(uid=263877, iid='0312961324', r_ui=3.0, est=2.995719735619176,
details={'was_impossible': False}),
 Prediction(uid=267635, iid='0345391055', r_ui=3.1699250014423126,
est=3.140821914251984, details={'was_impossible': False}),
 Prediction(uid=82511, iid='0553568760', r_ui=3.1699250014423126,
est=2.84398299304295, details={'was_impossible': False}),
 Prediction(uid=69078, iid='0894805770', r_ui=3.0, est=3.229978266349879,
details={'was_impossible': False}),
 Prediction(uid=56399, iid='0345350499', r_ui=3.4594316186372978,
est=3.351975975139548, details={'was_impossible': False}),
 Prediction(uid=146803, iid='0399141464', r_ui=3.0, est=2.678352634016068,
details={'was_impossible': False}),
 Prediction(uid=265751, iid='0684872153', r_ui=3.4594316186372978,
est=3.333654923997899, details={'was_impossible': False}),
 Prediction(uid=107784, iid='0451128370', r_ui=2.584962500721156,
```

SlopeOne

Train Test Split And Model Building

```
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=0.25)

# We'll use the famous SVD algorithm
algo = SlopeOne()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)

# Then compute MAE
accuracy.mae(predictions)
```

```
RMSE: 0.3873
MAE:  0.2694
0.269443908648527
```

```
interactions_full_df.head(1)
```

	User-ID	ISBN	Book-Rating
0	254	0060934700	3.321928

```
user_id = '254'

isbn = '0060934700'
```