CAPSTONE PROJECT ON
BOOK RECOMMENDATION SYSTEM

CHAPTER 1- OBJECTIVES AND INTRODUCTION

1.1 OBJECTIVE :

- The main objective is to create a machine learning model to recommend relevant books to users based on popularity and user interests.
- In addition to the ML Model prediction, we also have taken into account the book recommendation for a totally new user.

1.2 INTRODUCTION:

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).

Recommendation 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.

Since, here we are trying to recommend books to users based on their past purchases or ratings the user gave previously, we are basically trying different models like Popularity based recommender system, Collaborative filtering based recommender system (user-item or item-item) etc. We will be using the Popularity based recommender system to deal with the cold start problem, where we do not have history of past purchases of a particular user or where the user is totally new.

The next chapters have the following sections which are the steps involved for solving the Book Recommendation System,

Section 1 - Data collection
Section 2 - Data preparation
Section 3 - Exploratory data analysis
Section 4 - Feature Engineering
Section 5 - Working different models
Section 6 - Evaluating model

CHAPTER 2 - DATA COLLECTION AND DATA PREPARATION

2.1 Data Summary:

We are using Book-Crossing dataset to train and test our recommendation system. Book-Crossings is a book ratings dataset compiled by Cai-Nicolas Ziegler. It contains 1.1 million ratings of 270,000 books by 90,000 users. The ratings are on a scale from 1 to 10. The Book-Crossing dataset comprises 3 files.

● Users

This .csv file 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

2.2 Data Collection

Before building any machine learning model, it is vital to understand what the data is, and what are we trying to achieve. Data exploration reveals the hidden trends and insights and data preprocessing makes the data ready for use by ML algorithms.

So, let's begin. . .

To proceed with the problem dealing first we will load our dataset that is given to us in a .csv file into a dataframe.

Mount the drive and load the csv file into a dataframe.

```
# Mounting Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#Loading data
path='/content/drive/MyDrive/AlmaBetter/Capstone_Project/Capstone Project III/'
users=pd.read_csv(path+'Users.csv')
users.head()
```

```
#Books data
books=pd.read_csv(path+'Books.csv')
books.head()
```

```
#Ratings Data
ratings=pd.read_csv(path+'Ratings.csv')
ratings.head()
```

After mounting the drive the next step is to import the required libraries. Python has a wide number of libraries which makes the work easier. Here pandas, numpy, matlplotlib, seaborn, math,sklearn etc.

CHAPTER 3- EXPLORATORY DATA ANALYSIS

The primary goal of EDA is to support the analysis of data prior to making any conclusions. It may aid in the detection of apparent errors, as well as a deeper understanding of data patterns, the detection of outliers or anomalous events, and the discovery of interesting relationships between variables.

3.1 Duplication:

In our further analysis we found that the dataset has no duplicate entries.

3.2 Dimension of dataset

```
# dimension of dataset
print(f'''\t  Book_df shape is {books.shape}
          Ratings_df shape is {ratings.shape}
          Users_df shape is {users.shape}''')
```

```
Book_df shape is (271360, 8)
Ratings_df shape is (1149780, 3)
Users_df shape is (278858, 3)
```

3.3 Users_Dataset:

In the users_dataset we have the following feature variables.

- User-ID (unique for each user)
- Location (contains city, state and country separated by commas)
- Age

Out of these features, User-ID is unique for each user, Location contains city, state and country separated by commas and we have Age given across each user.

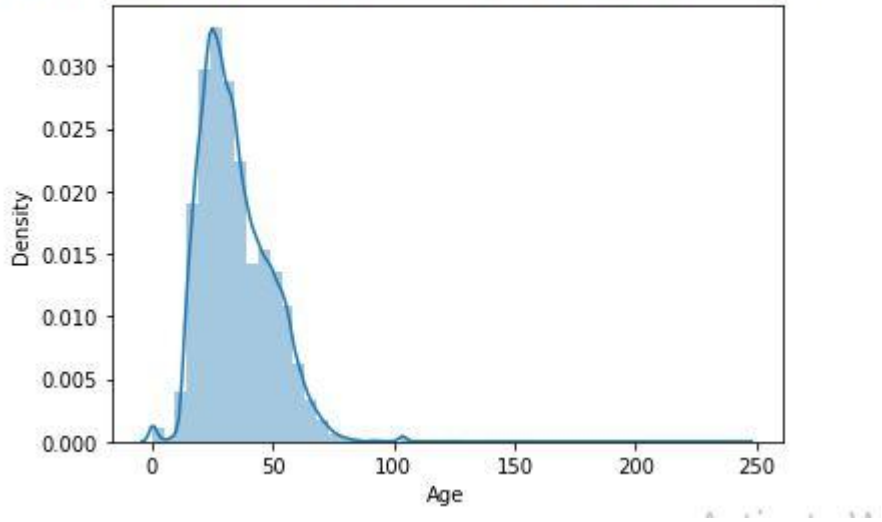
	userID	Location	Age
0	1	nyc, new york, usa	34
1	2	stockton, california, usa	18
2	3	moscow, yukon territory, russia	34
3	4	porto, v.n.gaia, portugal	17
4	5	farnborough, hants, united kingdom	34

Age:

Let's understand the Age distribution of the given user dataset. By using a distplot for the Age column we can get the distribution of ages and the density. Below is the distplot.

```
1 sns.distplot(users.Age)
```

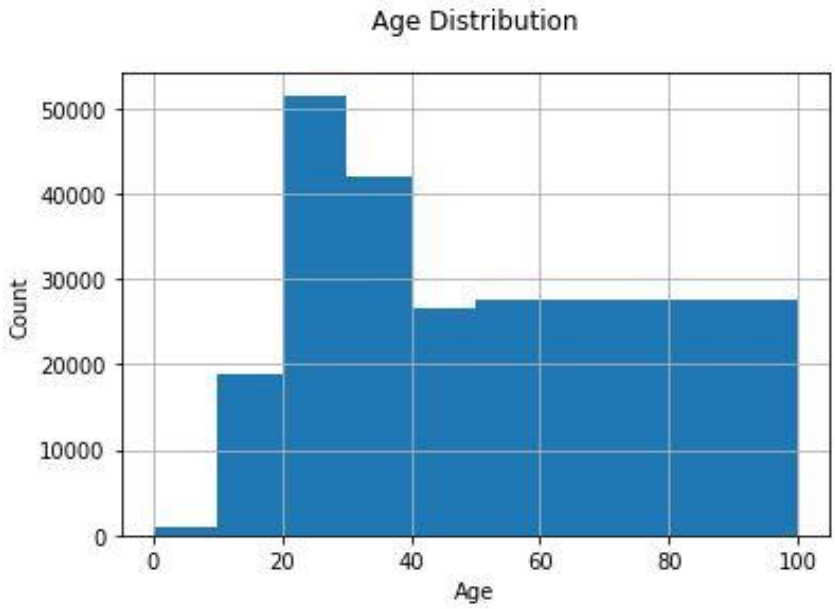
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a11ac00d0>
```



From the given distplot we can see that we have outliers in the Age column, we will treat these outliers in the coming section of outlier treatment.

Now let's have a look at the age distribution in a range of 0 to 100 by plotting a histogram.

From the given histogram plot for age, we see that the distribution is right skewed. This information will be used further in imputing the Null values in the Age column.

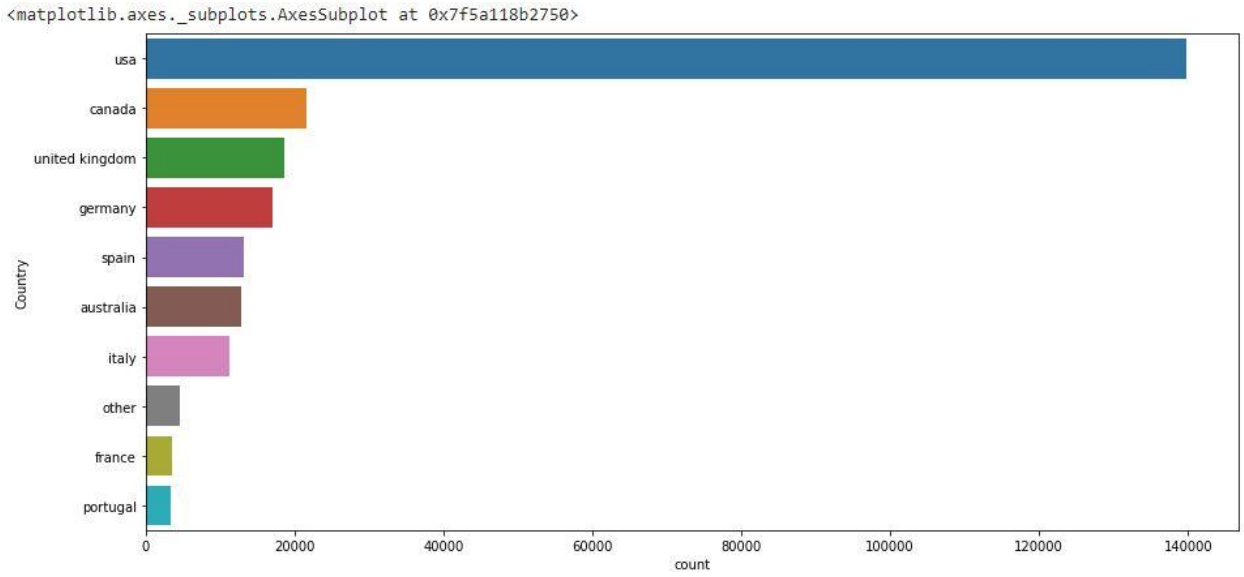


Location:

Now let's deep dive into our location column. This column has city, state and country separated by commas. We will first segregate these into different columns and we will introduce a new column "Country" so that we can analyse on the basis of the country of different users. The following code will separate the Country from the location.

```
for i in users:
    users['Country']=users.Location.str.extract(r'\,+\s?(\w*\s?\w*)\"*$')
```

There are mis-spellings in some of the country names. We will first correct these and then plot the top 10 countries from where we have the maximum number of users. The following countplot shows the top 10 countries, here we analyzed that the maximum number of users belong to the USA.



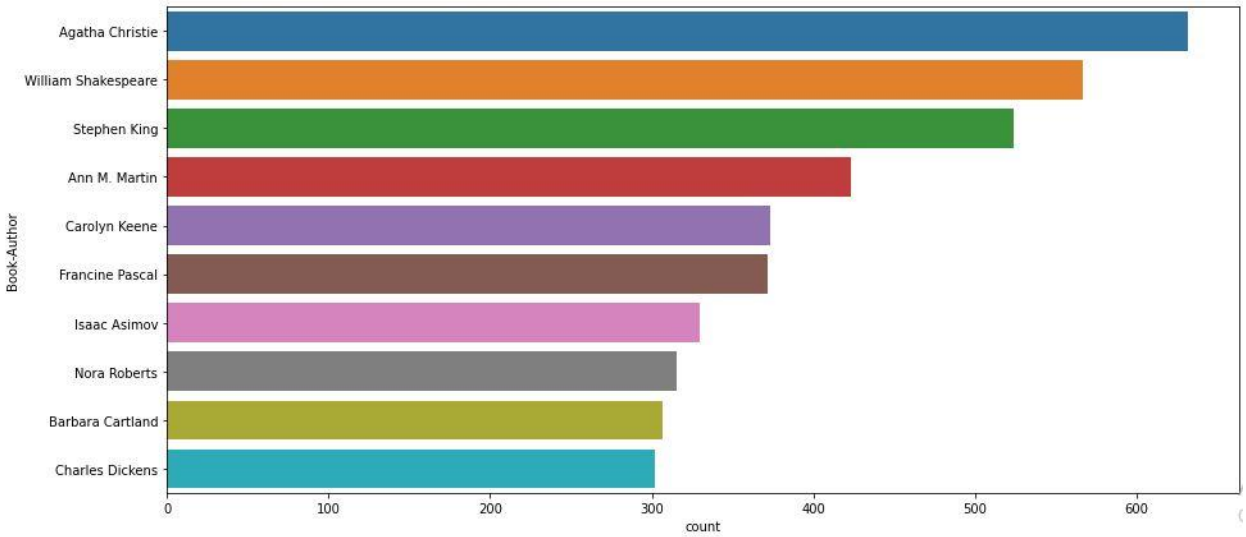
3.4 Book Dataset:

In the books_dataset we have the following feature variables.

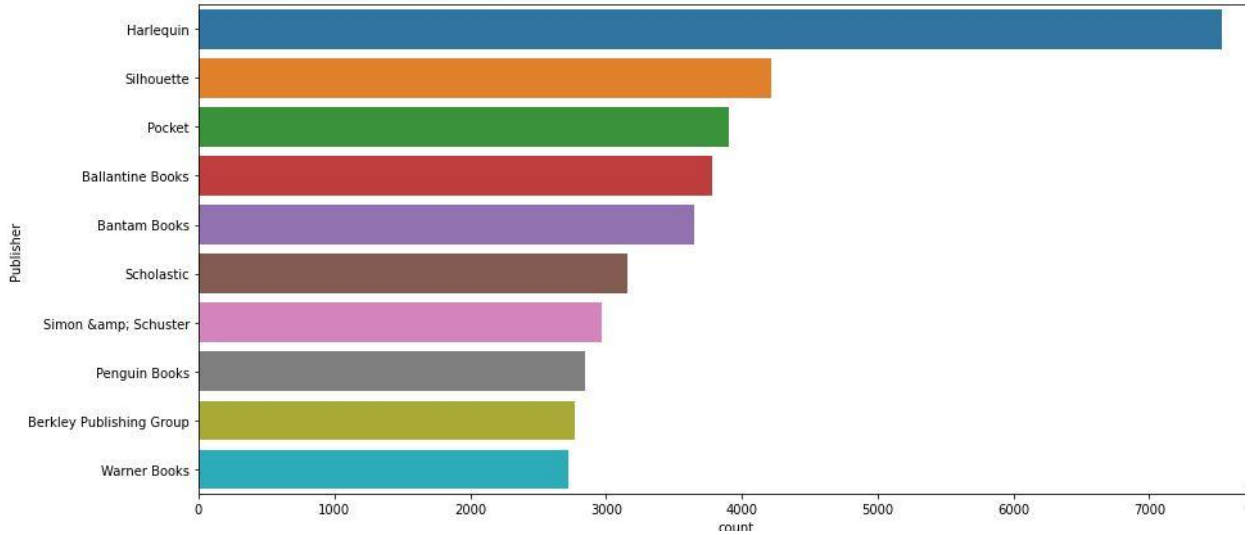
- ISBN (unique for each book)
- Book-Title
- Book-Author
- Year-Of-Publication
- Publisher
- Image-URL-S
- Image-URL-M
- Image-URL-L

From the count plot, let's find the top 10 Book-Author and top 10 Book-Publishers. Further we find that both the plots are skewed and the maximum number of books are from top 10 Book-Authors and top 10 Book-Publishers.

<matplotlib.axes._subplots.AxesSubplot at 0x7f5a11a56990>



<matplotlib.axes._subplots.AxesSubplot at 0x7f5a1194a3d0>



3.5 Ratings Dataset:

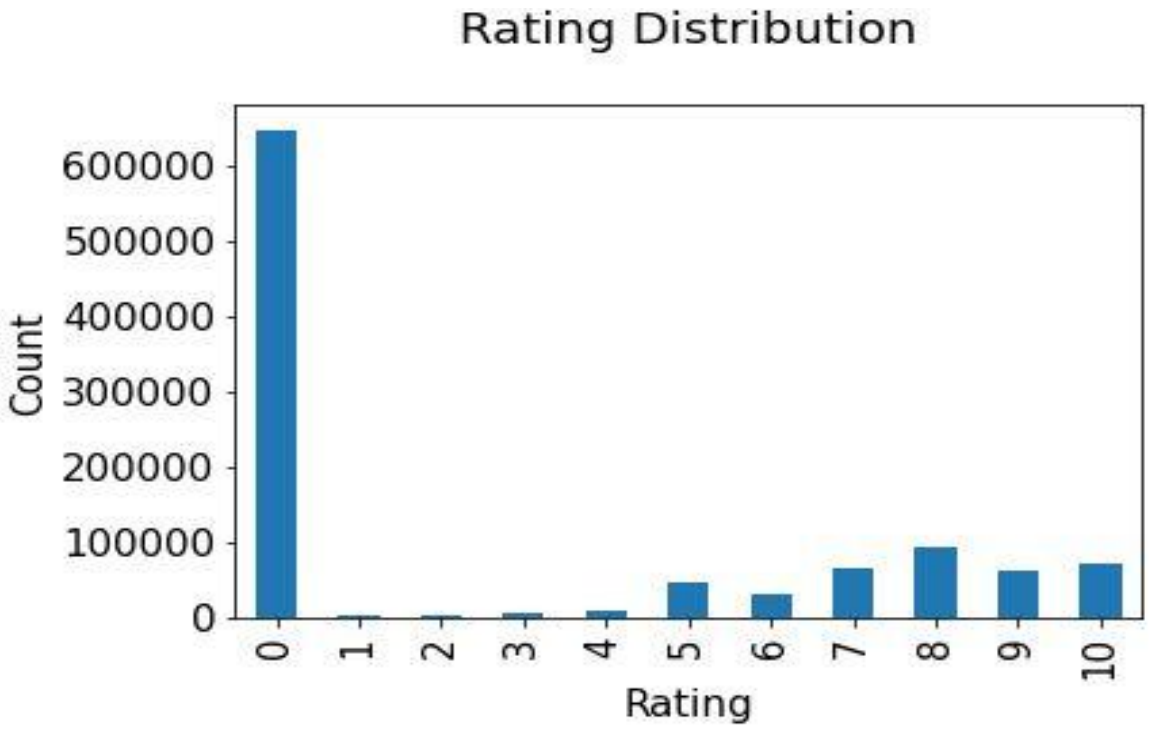
In the ratings_dataset we have the following feature variables.

- User-ID
- ISBN
- Book-Rating

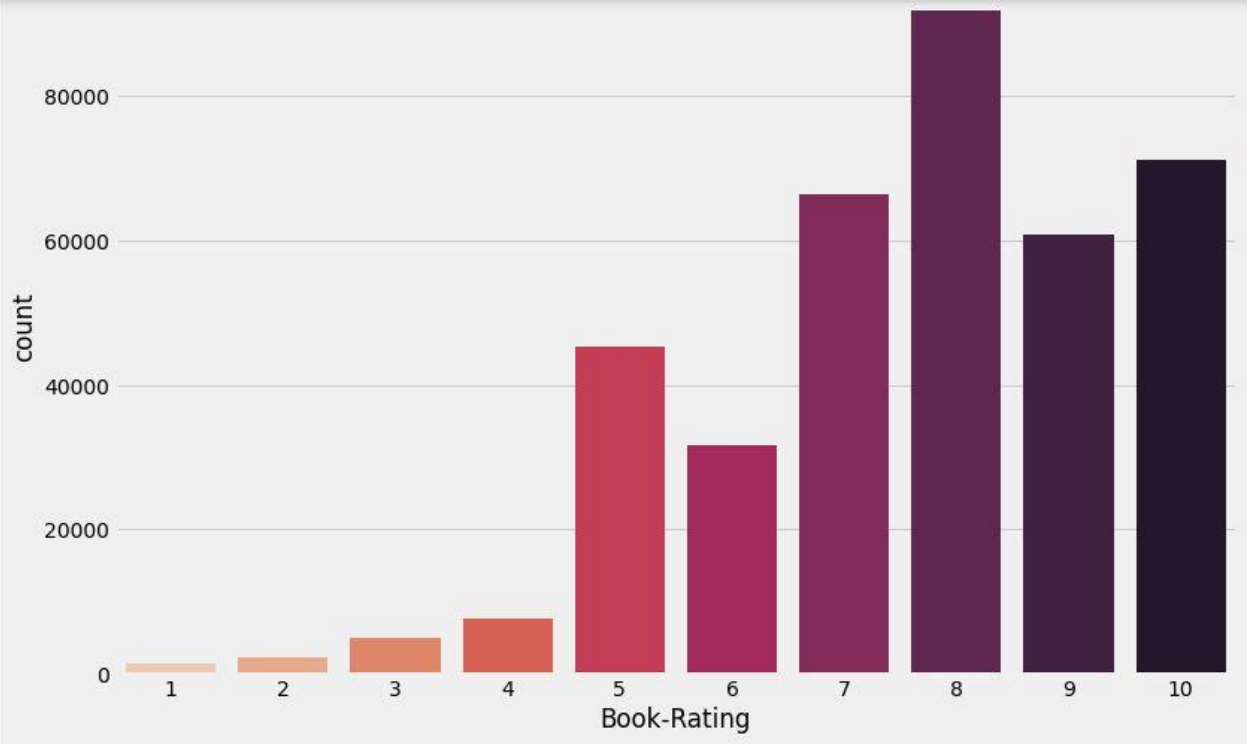
	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5

Let’s find the distribution of ratings frequency in our ratings dataset. From the following frequency plot, we find that most of the ratings are 0 which is implicit rating. The following code snippet will give us the frequency distribution.

```
plt.rc("font", size=15)
ratings_new['Book-Rating'].value_counts(sort=False).plot(kind='bar')
plt.title('Rating Distribution\n')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```



The ratings are very unevenly distributed, and the vast majority of ratings are 0 .As quoted in the description of the dataset - BX-Book-Ratings contains the book rating information. Ratings are either explicit, expressed on a scale from 1-10 higher values denoting higher appreciation, or implicit, expressed by 0. Hence segregating implicit and explicit ratings datasets.



It can be observed that higher ratings are more common amongst users and rating 8 has been rated the highest number of times.

3.4 Missing data:

Once you have raw data, you must deal with issues such as missing data and ensure that the data is prepared for forecasting models in such a way that it is amenable to them.

There were missing values in some columns in our dataset. The following code snippet gave us the idea about missing values in different columns in users_dataset.

```
def missing_values(df):
    mis_val=df.isnull().sum()
    mis_val_percent=round(df.isnull().mean().mul(100),2)
    mz_table=pd.concat([mis_val,mis_val_percent],axis=1)
    mz_table=mz_table.rename(
    columns={df.index.name:'col_name',0:'Missing Values',1:'% of Total Values'})
    mz_table['Data_type']=df.dtypes
    mz_table=mz_table.sort_values('% of Total Values',ascending=False)
    return mz_table.reset_index()
```

	index	Missing Values	% of Total Values	Data_type
0	Age	110762	39.72	float64
1	User-ID	0	0.00	int64
2	Location	0	0.00	object

Age has around 39% missing values which we will impute in further sections.

```
1 books.isna().sum()
```

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

We have 1 Null value in Book-Author and 2 Null values in Publisher columns of our Books dataframe, these Null values we will be treating in further sections.

```
 #   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
```

In our Ratings_df we do not have any null values.

3.5 Data Cleaning:

In the data cleaning section we will drop off the features which do not contribute towards making a good recommendation system.

Books_df:

We have already seen in our EDA part that the Image-URL-S, Image-URL-M and Image-URL-L do not contribute towards our final goal. So, we will drop off these columns and also dropping these columns we will not be losing any information.

CHAPTER 4 : FEATURE ENGINEERING

Handling missing values :

Values that are reported as missing may be due to a variety of factors. These lack of answers would be considered missing values. The researcher may leave the data or do data imputation to replace them. Suppose the number of cases of missing values is extremely small; then, an expert researcher may drop or omit those values from the analysis. But here in our case we had a lot of data points which were having missing values in different feature columns.

4.1 Imputing Values:

Since, we have missing values in some feature variables, we need to impute them.

- Year-Of-Publication
- Book-Author
- Publisher
- Age

Year-Of-Publication:

From the analysis part we get that the Year-Of_publication was wrongly mentioned for some of the rows. Diving deep into the Books_df we got to know that for these rows there was actually a column mismatch.

ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
078946697X	DK Readers: Creating the X-Men, How It All Beg...	2000	DK Publishing Inc	http://images.amazon.com/images/P/078946697X.0...
0789466953	DK Readers: Creating the X-Men, How Comic Book...	2000	DK Publishing Inc	http://images.amazon.com/images/P/0789466953.0...

As it can be seen from above, there are some incorrect entries in the Year-Of-Publication field. It looks like Publisher names 'DK Publishing Inc' and

'Gallimard' have been incorrectly loaded as Year-Of-Publication in the dataset due to some errors in the csv file.
Also, since these entries were few, we could manually correct the column values for these particular rows. The following code snippet shows the imputation.

```
#From above, it is seen that bookAuthor is incorrectly loaded with bookTitle, hence making required corrections
#ISBN '0789466953'
books.loc[books.ISBN == '0789466953','Year-Of-Publication'] = 2000
books.loc[books.ISBN == '0789466953','Book-Author'] = "James Buckley"
books.loc[books.ISBN == '0789466953','Publisher'] = "DK Publishing Inc"
books.loc[books.ISBN == '0789466953','Book-Title'] = "DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)"

#ISBN '078946697X'
books.loc[books.ISBN == '078946697X','Year-Of-Publication'] = 2000
books.loc[books.ISBN == '078946697X','Book-Author'] = "Michael Teitelbaum"
books.loc[books.ISBN == '078946697X','Publisher'] = "DK Publishing Inc"
books.loc[books.ISBN == '078946697X','Book-Title'] = "DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)"

#rechecking
books.loc[(books.ISBN == '0789466953') | (books.ISBN == '078946697X'),:]
#corrections done
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
	078946697X	DK Readers: Creating the X-Men, How It All Beg...	Michael Teitelbaum	2000	DK Publishing Inc
	0789466953	DK Readers: Creating the X-Men, How Comic Book...	James Buckley	2000	DK Publishing Inc

Also, for Year-Of-Publication we observed that the year mentioned was beyond 2020 for some entries whereas the dataset was created in 2004. So, for the anomalous entries we first filled them with Nan values. Then we also observed that the Year-Of_publication graph was right skewed so we imputed the Nan values with the median. The code snippet will give a better idea.

```
books.loc[(books['Year-Of-Publication'] > 2006) | (books['Year-Of-Publication'] == 0),'Year-Of-Publication'] = np.NAN

#replacing NaNs with median value of Year-Of-Publication
books['Year-Of-Publication'].fillna(round(books['Year-Of-Publication'].median()), inplace=True)
```

For Book-Author and Publisher missing data since there were a very few missing values we could search the internet and impute the values.

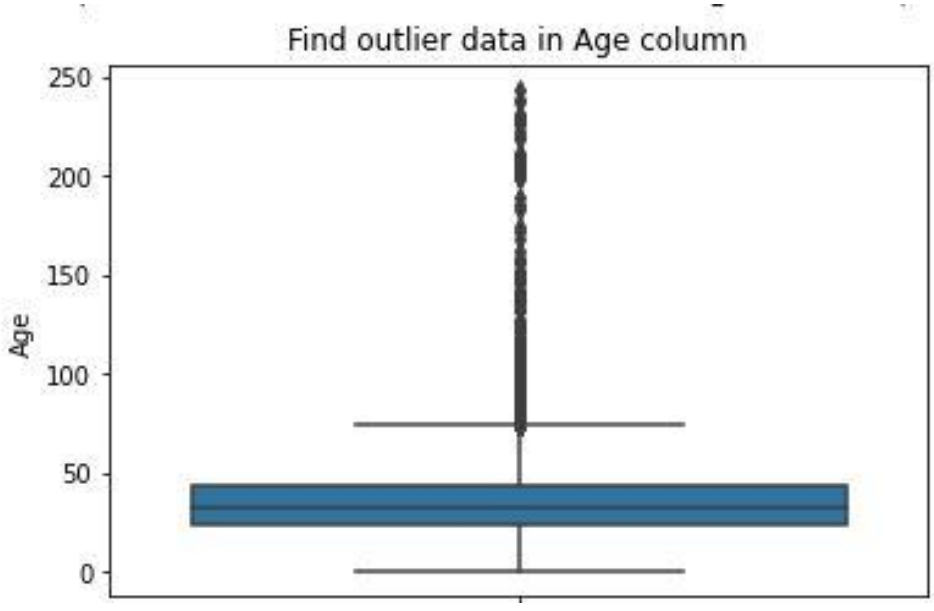
Age:

Outliers and Imputation:
An outlier is an observation of a data point that lies an abnormal distance from other values in a given population. It is an abnormal observation during the Data Analysis stage, that data point lies far away from other values.

What method to use for finding outliers?

Univariate method: I believe you're familiar with Univariate analysis, playing around one variable/feature from the given data set. Here to look at the Outlier we're going to apply the BOX plot to understand the nature of the Outlier and where it is exactly.

Now, let's deep dive into the data distribution and find if we have outliers or not. There are outliers in the Age feature of Users_df.



From the given box plot we found that there were outliers in the Age column and the Age range exceeded beyond 100 which we all know is not sensible at all. Also in the EDA part we inferred that 40% of the data in the Age column was Nan.

Now let's see how to treat outliers and missing values.

Having a look at the distribution of our outliers and the missing number of values we came to the point that we cannot afford to just drop off these Nan and outliers because even these users have given ratings and we have User-Item interactions for these users. So, we concluded not to drop off these values, instead we first imputed the outliers with Nan values then we imputed all the Nan values with some sensible number.

But, how to decide with what value to impute??

Here we took an interesting approach. Instead of simply imputing with a single value we grouped our user information based on country and took the median of the age then we imputed these values in place of NaNs. Also, we decided to impute with median because earlier we saw that the age distribution is positively skewed.

The following code snippet will give you better understanding.

```
# outlier data became NaN
users.loc[(users.Age > 100) | (users.Age < 5), 'Age'] = np.nan
```

Age value's below 5 and above 100 do not make much sense for our book rating case...hence replacing these by NaNs. Next step is to impute these NaNs with median grouped on country values.

```
users['Age'] = users['Age'].fillna(users.groupby('Country')['Age'].transform('median'))
```

CHAPTER 5: Popularity Based Approach

It is a type of recommendation system which works on the principle of popularity and or anything which is in trend. These systems check about the books which are in trend or are most popular among the users and directly recommend them.

For example, if a product is often purchased by most people then the system will get to know that that product is most popular so for every new user who just signed it, the system will recommend that product to that user also and chances become high that the new user will also purchase that.

Why Is this model relevant?

The answer to this is the **Cold-Start Problem.** Cold start is a potential problem in computer-based information systems which involves a degree of automated data modelling. Specifically, it concerns the issue that the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information. The cold start problem is a well known and well researched problem for recommender systems. Recommender systems form a specific type of information filtering (IF) technique that attempts to present information items (e-commerce, films, music, books, news, images, web pages) that are likely of interest to the user. Typically, a recommender system compares the user's profile to some reference characteristics. These characteristics may be related to item characteristics (content-based filtering) or the user's social environment and past behavior (collaborative filtering). Depending on the system, the user can be associated with various kinds of interactions: ratings, bookmarks, purchases, likes, number of page visits etc.

There are three cases of cold start:

- **New community:** refers to the start-up of the recommender, when, although a catalogue of items might exist, almost no users are present and the lack of user interaction makes it very hard to provide reliable recommendations
- **New item:** a new item is added to the system, it might have some content information but no interactions are present
- **New user:** a new user registers and has not provided any interaction yet, therefore it is not possible to provide personalized recommendations

A popularity based model does not suffer from cold start problems which means on day 1 of the business also it can recommend products on various different filters. There is

no need for the user's historical data. The popularity index used for our books dataset was **weighted rating**.

$$WR = [(v * R)/(v + m)] + [(m * c)/(v + m)]$$

where,

v is the number of votes for the books;

m is the minimum votes required to be listed in the chart;

R is the average rating of the book; and

C is the mean vote across the whole report.

The function used to calculate **C** and **m** is given below.

```
C= Final_Dataset['Avg_Rating'].mean()
m= Final_Dataset['Total_No_Of_Users_Rated'].quantile(0.90)
Top_Books = Final_Dataset.loc[Final_Dataset['Total_No_Of_Users_Rated'] >= m]
print(f'C={C} , m={m}')
Top_Books.shape
```

```
C=7.626700569504765 , m=64.0
(38570, 11)
```

Here we used 90th percentile as our cutoff. In other words, for a book to feature in the charts, it must have more votes than at least 90% of the books in the list.
We see that there are 38570 books which qualify to be in this list. Now, we need to calculate our metric for each qualified book. To do this, we will define a function, **weighted_rating()** and define a new feature score, of which we'll calculate the value by applying this function to our DataFrame of qualified books:

```
def weighted_rating(x, m=m, C=C):
    v = x['Total_No_Of_Users_Rated']
    R = x['Avg_Rating']
    return (v/(v+m) * R) + (m/(m+v) * C)


Top_Books['Score'] = Top_Books.apply(weighted_rating,axis=1)


#Sorting books based on score calculated above
Top_Books = Top_Books.sort_values('Score', ascending=False)
```

Creation of Weighted Ratings

Using this popularity metric we can calculate the top books that could be recommended to a user.

	Book-Title	Total_No_Of_Users_Rated	Avg_Rating	Score
0	Harry Potter and the Goblet of Fire (Book 4)	137	9.262774	8.741835
1	Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))	313	8.939297	8.716469
2	Harry Potter and the Order of the Phoenix (Book 5)	206	9.033981	8.700403
3	To Kill a Mockingbird	214	8.943925	8.640679
4	Harry Potter and the Prisoner of Azkaban (Book 3)	133	9.082707	8.609690
5	The Return of the King (The Lord of the Rings, Part 3)	77	9.402597	8.596517
6	Harry Potter and the Prisoner of Azkaban (Book 3)	141	9.035461	8.595653
7	Harry Potter and the Sorcerer's Stone (Book 1)	119	8.983193	8.508791
8	Harry Potter and the Chamber of Secrets (Book 2)	189	8.783069	8.490549
9	Harry Potter and the Chamber of Secrets (Book 2)	126	8.920635	8.484783
10	The Two Towers (The Lord of the Rings, Part 2)	83	9.120482	8.470128
11	Harry Potter and the Goblet of Fire (Book 4)	110	8.954545	8.466143
12	The Fellowship of the Ring (The Lord of the Rings, Part 1)	131	8.839695	8.441584
13	The Hobbit : The Enchanting Prelude to The Lord of the Rings	161	8.739130	8.422706
14	Ender's Game (Ender Wiggins Saga (Paperback))	117	8.837607	8.409441
15	Tuesdays with Morrie: An Old Man, a Young Man, and Life's Greatest Lesson	200	8.615000	8.375412
16	Charlotte's Web (Trophy Newbery)	68	9.073529	8.372037
17	Dune (Remembering Tomorrow)	75	8.973333	8.353301
18	A Prayer for Owen Meany	181	8.607735	8.351465
19	Fahrenheit 451	164	8.628049	8.346969

Hence it was observed that the **Harry Potter series** was the most popular book series. Other than that books like **To Kill A Mockingbird**, **The Lord of the Rings** and **Dune** were among the top books.

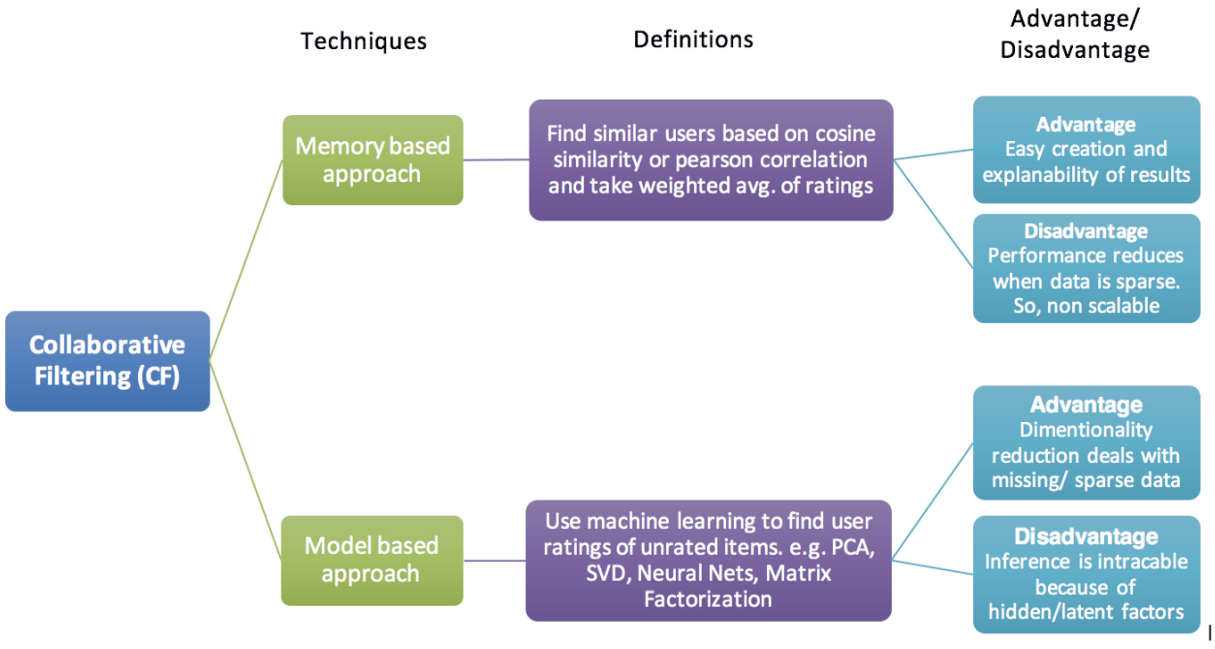
The **Popularity Based Recommender** provides a general chart of recommended

movies to all the users. They **are not sensitive to the interests and tastes of a particular user**. It is not personalised and the system would recommend the same sort of products/movies which are solely based upon popularity to every other user. Hence we need to try and work out other kinds of recommender systems.

CHAPTER 6: Collaborative Filtering

It is considered to be one of the very smart recommender systems that work on the similarity between different users and also items that are widely used as an e-commerce website and also online movie websites. It checks about the taste of similar users and makes recommendations.

The similarity is not restricted to the taste of the user, moreover there can be consideration of similarity between different items also. The system will give more efficient recommendations if we have a large volume of information about users and items. There are various types of collaborative filtering techniques as mentioned in the diagram given below.



Types of collaborative filtering approaches

6.1 Model Based Approach:

Model-based recommendation systems involve building a model based on the dataset of ratings. In other words, we extract some information from the dataset, and use that as a "model" to make recommendations without having to use the complete dataset every time. This approach potentially offers the benefits of both speed and scalability.

Model based collaborative approaches only rely on user-item interactions information and assume a latent model supposed to explain these interactions. For example, matrix factorisation algorithms consists in decomposing the huge and sparse user-item interaction matrix into a product of two smaller and dense matrices: a user-factor matrix (containing users representations) that multiplies a factor-item matrix (containing items representations).

Model based approach hence involves building machine learning algorithms to predict user's ratings. They involve dimensionality reduction methods that reduce high dimensional matrices containing an abundant number of missing values with a much smaller matrix in lower-dimensional space. To understand this further lets understand what matrix factorization is.

6.2 Matrix Factorization

The main assumption behind matrix factorisation is that there exists a pretty low dimensional latent space of features in which we can represent both users and items and such that the interaction between a user and an item can be obtained by computing the dot product of corresponding dense vectors in that space.

For example, consider that we have a user-book rating matrix. In order to model the interactions between users and books, we can assume that:

- there exist some features describing (and telling apart) pretty well books.
- these features can also be used to describe user preferences (high values for features the user likes, low values otherwise)

However we don't want to give explicitly these features to our model (as it could be done for content based approaches that we will describe later). Instead, we prefer to let the system discover these useful features by itself and make its own representations of both users and items. As they are learned and not given, extracted features taken individually have a mathematical meaning but no intuitive interpretation (and, so, are difficult, if not impossible, to understand as human). However, it is not unusual to end up having structures emerging from that type of algorithm being extremely close to intuitive decomposition that humans could think about. Indeed, the consequence of such factorisation is that close users in terms of preferences as well as close items in terms of characteristics end up having close representations in the latent space.

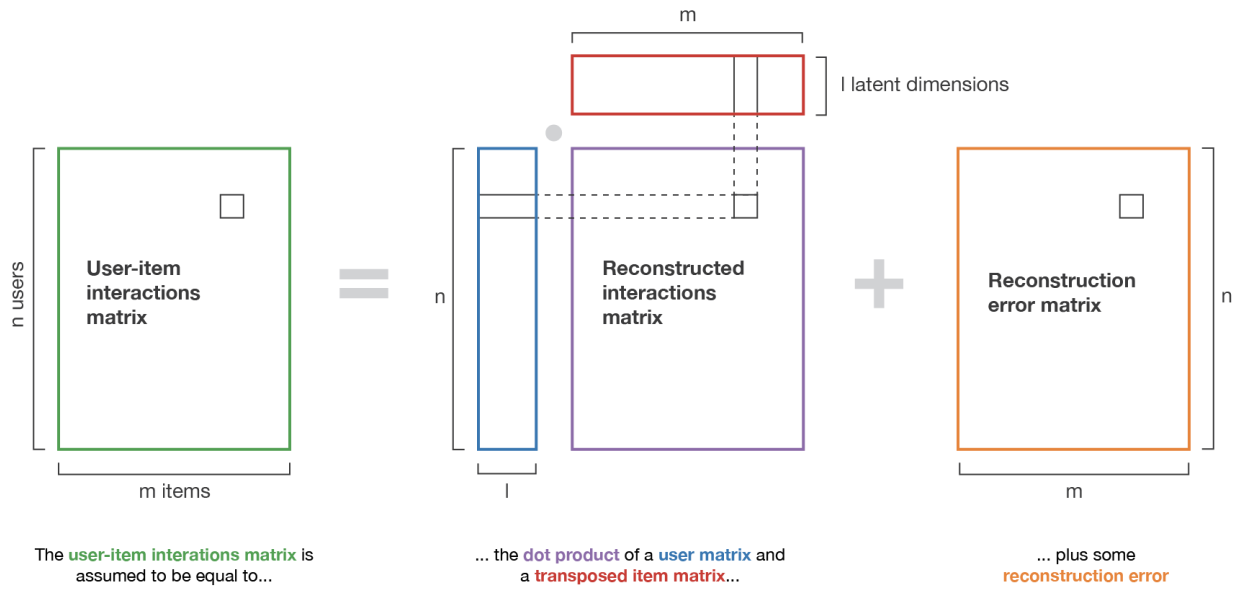


Illustration of the matrix factorization method.

Our analysis will focus on book recommendations based on Book-Crossing dataset. To reduce the dimensionality of the dataset and avoid running into memory error we will focus on users with at least 3 ratings and top 10% most frequently rated books. After filtering out the users the total number of records came down to 1.6 lakhs.

The two techniques used for matrix factorization here are SVD and NMF.

6.2.1 Non-Negative Matrix Factorization (NMF):

Non-negative matrix factorization also non-negative matrix approximation is a group of algorithms in multivariate analysis and linear algebra where a matrix V is factorized into (usually) two matrices W and H, with the property that all three matrices have no negative elements. This non-negativity makes the resulting matrices easier to inspect. Also, in applications such as processing of audio spectrograms or muscular activity, non-negativity is inherent to the data being considered. Since the problem is not exactly solvable in general, it is commonly approximated numerically.

This algorithm is used in a vast number of fields including image processing, text mining, clustering, collaborative filtering, and community detection. Its most significant assets are speed, ease of interpretation and versatility.

It's quite simple: you put your users as columns and products/ratings as rows of an array (let's call it V). As values, you should put adequate statistics like a number of purchases or ratings. Our goal in NMF is to approximate this matrix by the dot product of two arrays W and H. Dimensions of the arrays are defined by dimensions of V and number of components we set to the algorithm. If V has n rows and m columns and we want to decompose it to k components, then W has n rows, and k columns and H has k rows and m columns. Of course usually, it's impossible to reconstruct the initial matrix precisely. We want to be as "close" as possible to the initial array. To measure the distance, we can use the Frobenius norm.

Using NMF for our test data we observed the following results:

```
model_nmf = NMF()
cv_results_nmf = cross_validate(model_nmf, data, cv=3)
pd.DataFrame(cv_results_nmf).mean()
```

```
test_rmse    2.622481
test_mae     2.240744
fit_time     9.238085
test_time    0.485544
dtype: float64
```

We observe that the **RMSE and MAE scores are quite low**.

6.2.2 Singular Value Decomposition (SVD):

Singular value decomposition also known as the SVD algorithm is used as a collaborative filtering method in recommendation systems. SVD is a matrix factorization method that is used to reduce the features in the data by reducing the dimensions from N to K where (K<N).

For the part of the recommendation, the only part which is taken care of is matrix factorization that is done with the user-item rating matrix. Matrix-factorization is all about taking 2 matrices whose product is the original matrix. Vectors are used to represent item ‘qi’ and user ‘pu’ such that their dot product is the expected rating as given below

$$\text{expected rating} = \hat{r}_{ui} = q_i^T p_u$$

qi’ and ‘pu’ can be calculated in such a way that the square error difference between the dot product of user and item and the original ratings in the user-item matrix is least.

Why SVD?

There are 3 primary reasons for that:

- It’s very efficient
- The basis is hierarchical, ordered by relevance
- It tends to perform quite well for most data sets

Implementing SVD on our dataset along with cross validation involving 3 folds i.e. cv=3, we can see the following results:

```
model_svd = SVD()
cv_results_svd = cross_validate(model_svd, data, cv=3)
pd.DataFrame(cv_results_svd).mean()
```

```
test_rmse    1.602729
test_mae     1.239371
fit_time     6.056137
test_time    0.640078
```

Thus we observe that the SVD model performs considerably **better than the NMF model** in terms of very l**ow RMSE and MAE scores for the test set** as well as in terms of **lower test time taken for execution**.

Since SVD gave us better results, let's try and understand the SVD model results. The predicted rating and the Absolute Error for each user-item is calculated and stored as given below.

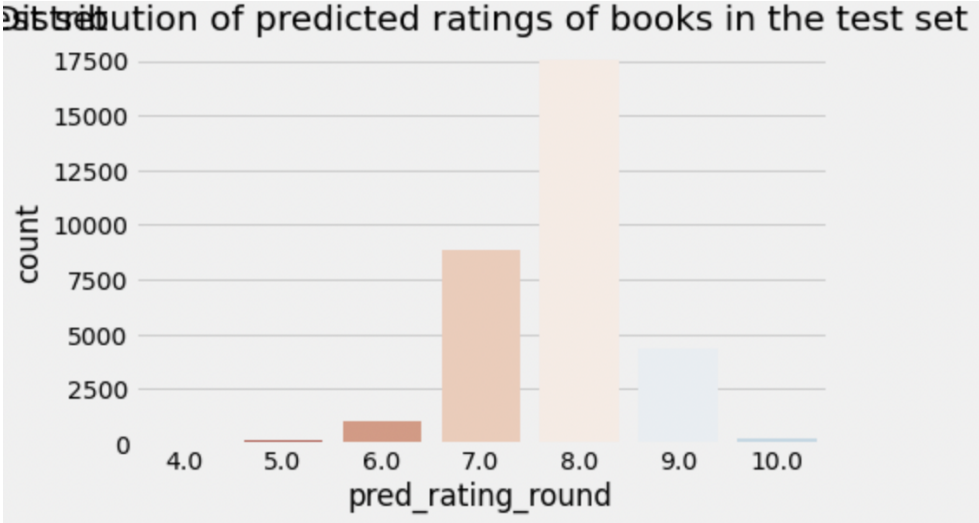
	user_id	isbn	actual_rating	pred_rating	impossible	pred_rating_round	abs_err
30500	240700	0064407446	10.0	6.984772	False	7.0	3.015228
16607	146113	0061009059	7.0	7.789755	False	8.0	0.789755
8498	273979	0786884142	7.0	6.840120	False	7.0	0.159880
11152	25359	157145697X	8.0	7.984568	False	8.0	0.015432
21034	32195	0515120863	5.0	6.951631	False	7.0	1.951631

Dataframe showing Actual And PredictedRatings with the Error Factor

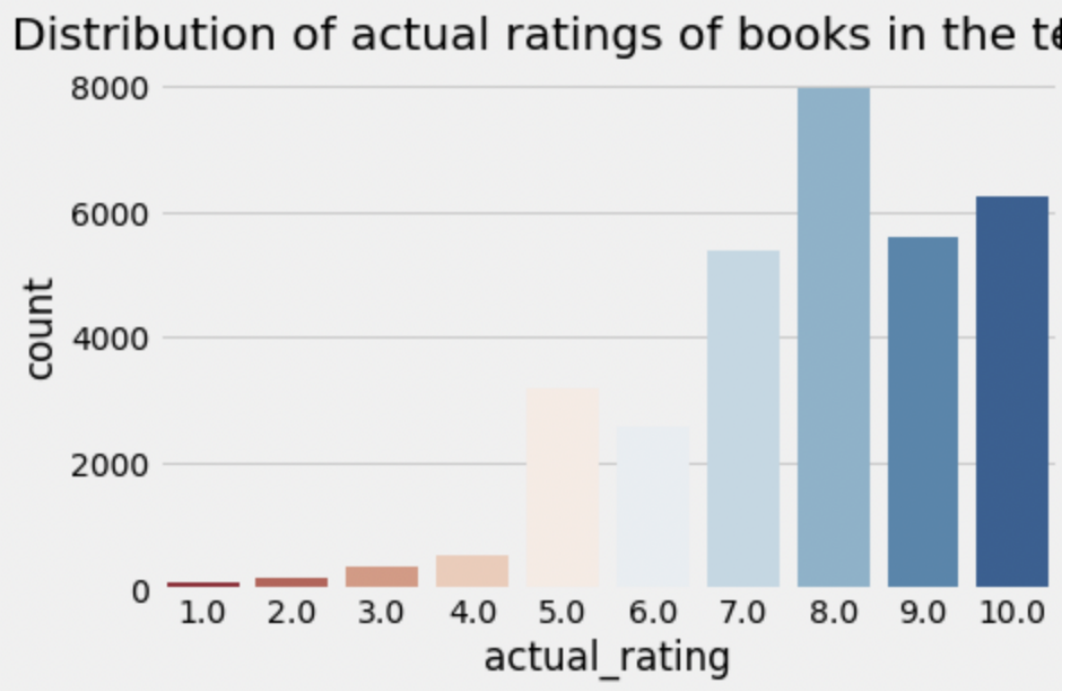
Distribution of actual and predicted ratings in the test set According to the distribution of actual ratings of books in the test set, the biggest part of users give positive scores - between 7 and 10.

The mode equals 8 but count of ratings 7, 9, 10 is also noticeable. The distribution of predicted ratings in the test set is visibly different.

One more time, 8 is a mode but scores 7, 9 and 10 are clearly less frequent. It shows that the recommender system is not perfect and it cannot reflect the real distribution of book ratings. The graphical representation of actual and predicted ratings is as follows.



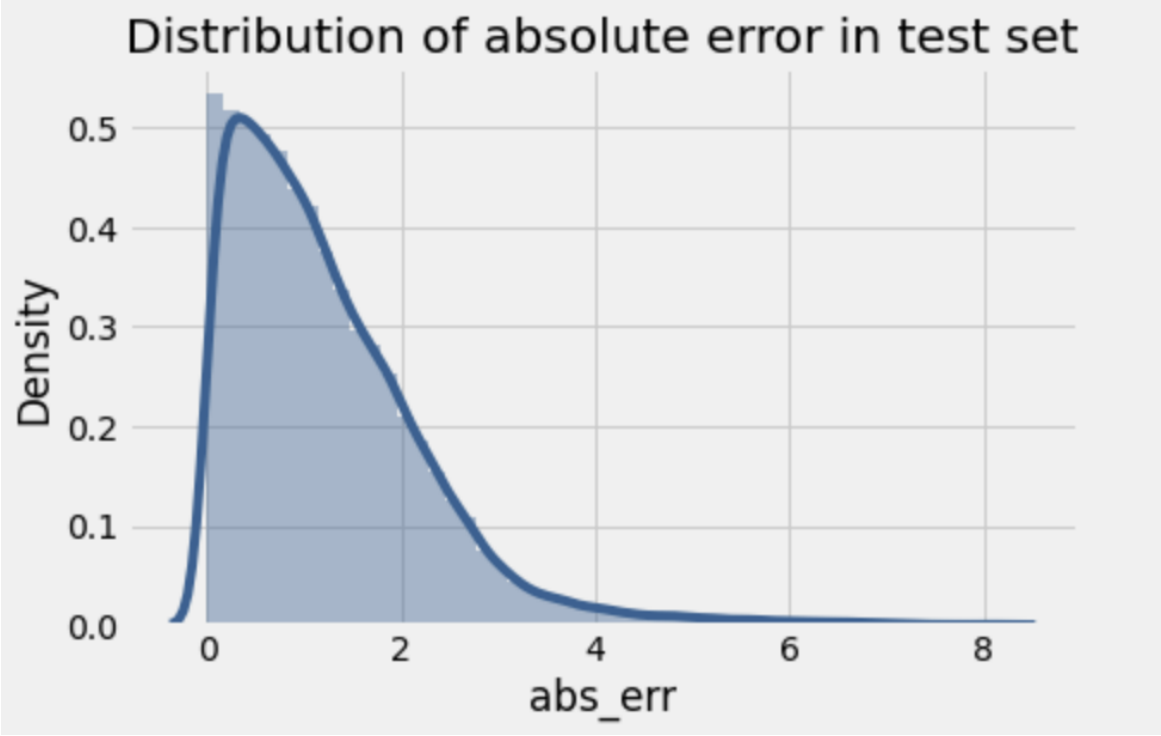
Predicted Rating Distribution



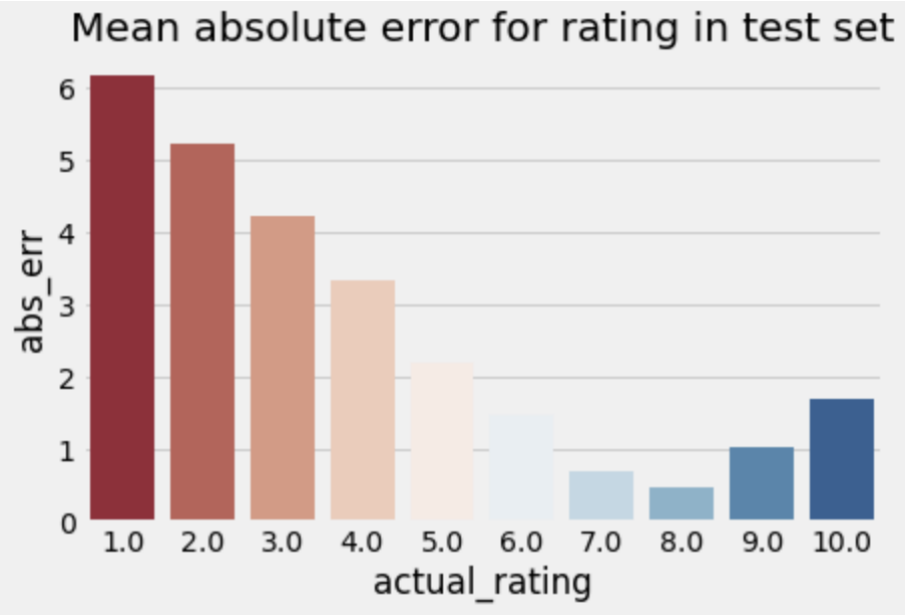
Actual Rating Distribution

Also, the distribution of absolute errors is right-skewed, showing that the majority of errors is small: between 0 and 1. There is a long tail that indicates that there are several observations for which the absolute error was close to 10.

How good/bad the model is with predicting certain scores?
As expected from the above charts, the model deals very well with predicting score = 8 (the most frequent value). The further the rating from score = 8, the higher the absolute error. The biggest errors happen to observations with scores 1 or 2 which indicates that probably the model is predicting high ratings for those observations.



Distribution of Absolute Error



Error vs Actual Ratings

Analysis of predicted ratings of a particular user:

For this part of the analysis, the **user with id 193458 was selected**. By analyzing book ratings by this user, it can be noted that he/she likes diverse types of readings: English romantic novels (Pride and Prejudice, Sense and Sensibility), fantasy (Narnia) as well as historical novels (Schindler's List). Among the recommended books there are other works from Narnia's series, two historical novels and one romance which correlates with the user's previous preferences.

Actual Top Rated Books

Predicted Top Rated Books

Hence we can observe that there is a very little difference between the prediction and actual ratings.

6.3 Memory Based Approach:

The main characteristics of user-user and item-item approaches is that they use only information from the user-item interaction matrix and they assume no model to produce new recommendations.

6.3.1 User-User:

In order to make a new recommendation to a user, the user-user method roughly tries to identify users with the most similar “interactions profile” (nearest neighbours) in order to suggest items that are the most popular among these neighbours (and that are “new” to our user). This method is said to be “user-centred” as it represents users based on their interactions with items and evaluates distances between users.

Assume that we want to make a recommendation for a given user. First, every user can be represented by its vector of interactions with the different items (“its line” in the interaction matrix). Then, we can compute some kind of “similarity” between our user of interest and every other user. That similarity measure is such that two users with similar interactions on the same items should be considered as being close. Once similarities to every user have been computed, we can keep the k-nearest-neighbours to our user and then suggest the most popular items among them (only looking at the items that our reference user has not interacted with yet).

Notice that, when computing similarity between users, the number of “common interactions” (how many items have already been considered by both users?) should be considered carefully! Indeed, most of the time, we want to avoid that someone that only has one interaction in common with our reference user could have a 100% match and be considered as being “closer” than someone having 100 common interactions and agreeing “only” on 98% of them. So, we consider that two users are similar if they have interacted with a lot of common items in the same way (similar rating, similar time hovering…).

The following diagram gives a good illustration of user-user interaction.

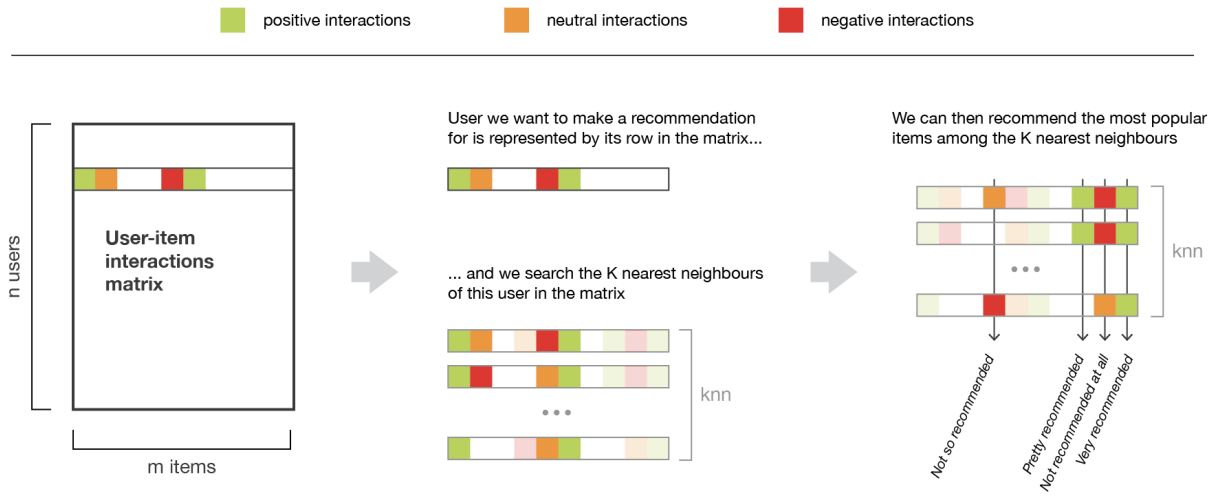


Illustration of the user-user method

For the scope of our problem, in-order to avoid sparsity, we are only considering the users who have interacted for more than 5 books. After creating the pivot matrix we use the recommender.

```
class CFRecommender:

    MODEL_NAME = 'Collaborative Filtering'

    def __init__(self, cf_predictions_df):
        self.cf_predictions_df = cf_predictions_df

    def get_model_name(self):
        return self.MODEL_NAME

    def recommend_items(self, user_id, items_to_ignore=[], topn=10):
        # Get and sort the user's predictions
        sorted_user_predictions = self.cf_predictions_df[user_id].sort_values(ascending=False).reset_index().rename(columns={user_id: 'recStrength'})

        # Recommend the highest predicted rating content that the user hasn't seen yet.
        recommendations_df = sorted_user_predictions[~sorted_user_predictions['ISBN'].isin(items_to_ignore)].sort_values('recStrength', ascending = False)

        return recommendations_df
```

Evaluation of this model is a bit different from the usual evaluation metrics in general. In Recommender Systems, there are a set of metrics commonly used for evaluation. We choose to work with Top-N accuracy metrics, which evaluates the accuracy of the top recommendations provided to a user, compared to the items the user has actually interacted with in the test set. **This evaluation method works as follows**:

- For each user
- For each item the user has interacted in test set
- Sample 100 other items the user has never interacted with.
- Ask the recommender model to produce a ranked list of recommended items, from a set composed of one interacted item and the 100 non-interacted items
- Compute the Top-N accuracy metrics for this user and interacted item from the recommendations ranked list
- Aggregate the global Top-N accuracy metrics

Below mentioned is a list of all evaluations performed for this model.

```
Enter User ID from above list for book recommendation  69078
Recommendation for User-ID =  69078
         ISBN                                          Book-Title  recStrength
0  0446310786                              To Kill a Mockingbird        0.842
1  0345370775                                       Jurassic Park        0.802
2  0312966970          Four To Score (A Stephanie Plum Novel)        0.675
3  0316769487                            The Catcher in the Rye        0.673
4  0345361792                           A Prayer for Owen Meany        0.646
5  0440214041                                The Pelican Brief        0.621
6  044021145X                                        The Firm        0.617
7  0440211727                                   A Time to Kill        0.617
8  0060928336  Divine Secrets of the Ya-Ya Sisterhood: A Novel        0.606
9  0312924585                              Silence of the Lambs        0.600
```

For a random set of users, the recall strength associated with its results is mentioned above. User **0446310786** has high recall capacity for certain books like **To Kill a MockingBird** meaning there is a high chance of this book being recommended.

Similarly, for users lower than that have lesser recall strength meaning the corresponding book being recommended has a lower chance.

Similarly **some other metrics** used are:

1. **Recall@K**: Shows proportion of relevant items found in the top-k recommendations.
2. **Hit@K**: Proportion of the seen data being present in the top-k recommendations.

```
Global metrics:
{'modelName': 'Collaborative Filtering', 'recall@5': 0.2357298474945534, 'recall@10': 0.3057371096586783}
```

	hits@5_count	hits@10_count	interacted_count	recall@5	recall@10	User-ID
10	252	343	1389	0.181	0.247	11676
31	189	245	1138	0.166	0.215	98391
45	17	30	380	0.045	0.079	189835
30	83	104	369	0.225	0.282	153662
70	29	33	236	0.123	0.140	23902
7	30	49	204	0.147	0.240	235105
47	22	32	203	0.108	0.158	76499
50	23	35	193	0.119	0.181	171118
42	55	68	192	0.286	0.354	16795
43	23	31	188	0.122	0.165	248718

Why does user-user not work that well?

- This method requires a high computation power as there are extremely high numbers of users as compared to items.
- Also the possibility of users changing with time has much higher probability than items changing. Hence we can say item-item is much more deterministic in nature.

6.3.2 Item-Item:

To make a new recommendation to a user, the idea of the item-item method is to find items similar to the ones the user already “positively” interacted with. Two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is said to be “item-centred” as it represents items based on interactions users had with them and evaluates distances between those items.

Assume that we want to make a recommendation for a given user. First, we consider the item this user liked the most and represent it (as all the other items) by its vector of interaction with every user (“its column” in the interaction matrix). Then, we can compute similarities between the “best item” and all the other items. Once the similarities have been computed, we can then keep the k-nearest-neighbours to the selected “best item” that are new to our user of interest and recommend these items. Notice that in order to get more relevant recommendations, we can do this job for more than only the user’s favourite item and consider the n preferred items instead. In this case, we can recommend items that are close to several of these preferred items.

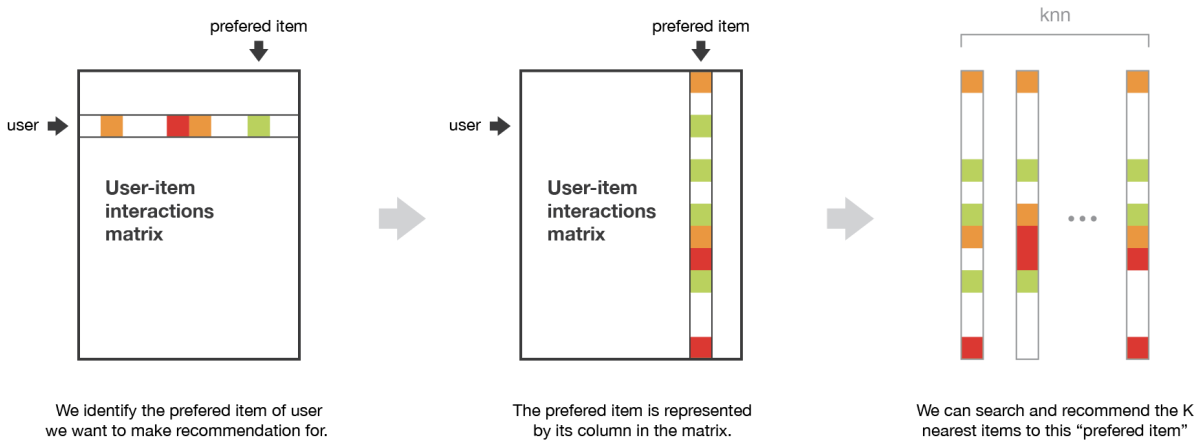


Illustration of the item-item method

For the scope of our project, we used the **K-Nearest Neighbours** algorithm. kNN is a machine learning algorithm to find clusters of similar users based on common book ratings, and make predictions using the average rating of top-k nearest neighbors. We convert our table to a 2D matrix, and fill the missing values with zeros (since we will calculate distances between rating vectors). We then transform the values(ratings) of the matrix data frame into a scipy sparse matrix for more efficient calculations.

To find nearest neighbours:

We use unsupervised algorithms with sklearn.neighbors. The algorithm we use to compute the nearest neighbors is “brute”, and we specify “metric=cosine” so that the algorithm will calculate the cosine similarity between rating vectors. Finally, we fit the model.

```
In [35]: us_canada_user_rating_pivot = us_canada_user_rating.pivot(index = 'bookTitle', columns = 'userID', values = 'bookRating').fillna(
         us_canada_user_rating_matrix = csr_matrix(us_canada_user_rating_pivot.values)

In [36]: from sklearn.neighbors import NearestNeighbors

         model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
         model_knn.fit(us_canada_user_rating_matrix)

Out[36]: NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                          metric_params=None, n_jobs=1, n_neighbors=5, p=2, radius=1.0)
```

Cosine similarity measures the **similarity** between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. Hence we can understand how 2 books are similar.

Testing model and making Recommendations:

In this step, the kNN algorithm measures distance to determine the “closeness” of instances. It then classifies an instance by finding its nearest neighbors, and picks the most popular class among the neighbors.

```
In [44]: query_index = np.random.choice(us_canada_user_rating_pivot.shape[0])
         distances, indices = model_knn.kneighbors(us_canada_user_rating_pivot.iloc[query_index, :].reshape(1, -1), n_neighbors = 6)

         for i in range(0, len(distances.flatten())):
             if i == 0:
                 print('Recommendations for {0}:\n'.format(us_canada_user_rating_pivot.index[query_index]))
             else:
                 print('{0}: {1}, with distance of {2}:'.format(i, us_canada_user_rating_pivot.index[indices.flatten()[i]], distances.flat

Recommendations for The Green Mile: Coffey's Hands (Green Mile Series):

1: The Green Mile: Night Journey (Green Mile Series), with distance of 0.26063737394209996:
2: The Green Mile: The Mouse on the Mile (Green Mile Series), with distance of 0.2911623754404248:
3: The Green Mile: The Bad Death of Eduard Delacroix (Green Mile Series), with distance of 0.2959542871302775:
4: The Two Dead Girls (Green Mile Series), with distance of 0.30596709534565514:
5: The Green Mile: Coffey on the Mile (Green Mile Series), with distance of 0.37646848777592923:
```

Recommendations for Harry Potter and the Sorcerer's Stone (Book 1)

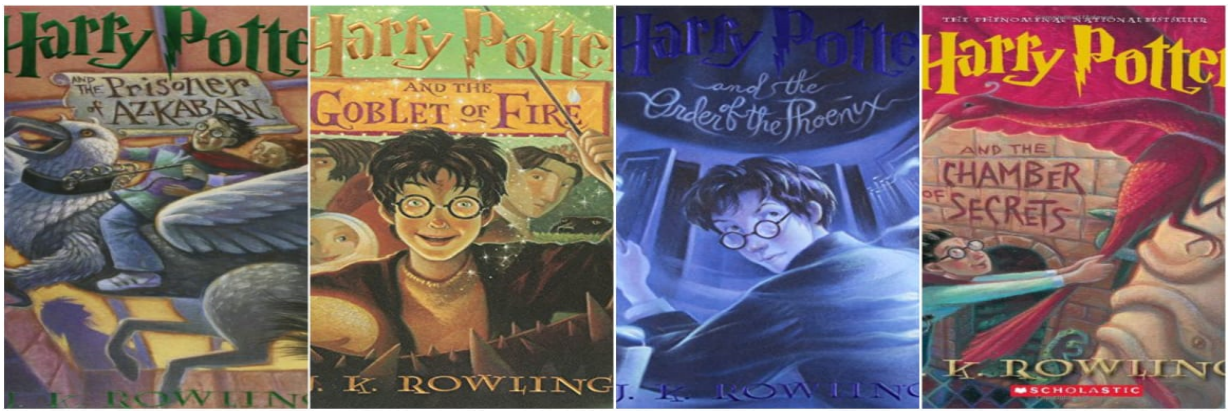


Illustration of the item-item method

What do we observe after implementing both the memory based techniques?

The user-user method is based on the search of similar users in terms of interactions with items. As, in general, every user has only interacted with a few items, it makes the method pretty sensitive to any recorded interactions (high variance). On the other hand, as the final recommendation is only based on interactions recorded for users similar to our user of interest, we obtain more personalized results (low bias).

Conversely, the item-item method is based on the search of similar items in terms of user-item interactions. As, in general, a lot of users have interacted with an item, the neighbourhood search is far less sensitive to single interactions (lower variance). As a counterpart, interactions coming from every kind of user (even users very different from our reference user) are then considered in the recommendation, making the method less personalised (more biased). Thus, this approach is less personalized than the user-user approach but more robust.

CONCLUSION

- In EDA, the Top-10 most rated books were essentially **novels**. Books like **The Lovely Bone** and **The Secret Life of Bees** were very well perceived.

- Majority of the readers were of the **age bracket 20-35** and most of them came from North American and European countries namely **USA, Canada, UK, Germany and Spain**.

- If we look at the ratings distribution, **most of the books have high rating**s with maximum books being rated 8. Ratings below 5 are few in number.

- Author with the most books was **Agatha Christie, William Shakespeare and Stephen King**.

- For modelling, it was observed that for **model based** collaborative filtering SVD technique worked way better than NMF with lower Mean Absolute Error (MAE) .

- Amongst the memory based approach, **item-item CF performed better** than **user-user CF** because of lower computation requirements .

CHALLENGES

- **Handling of sparsity** was a major challenge as well since the user interactions were not present for the majority of the books.

- Understanding the metric for evaluation was a challenge as well.

- Since the data consisted of text data, **data cleaning** was a major challenge in features like **Location** etc..

- **Decision making on missing value imputations and outlier treatment** was quite challenging as well.

FUTURE SCOPE

- Given more information regarding the books dataset, namely features like Genre, Description etc, we could implement a content-filtering based recommendation system and compare the results with the existing collaborative-filtering based system.

- We would like to explore various clustering approaches for clustering the users based on Age, Location etc., and then implement voting algorithms to recommend items to the user depending on the cluster into which it belongs.