A report submitted for

Book Recommendation System

Table of Contents

1. Introduction

1.1 Explanation of the Concept

A recommender system, is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender systems are utilized in a variety of areas, but are most commonly recognized as playlist generators for video and music services, product recommenders for online stores, or content recommenders for social media platforms.

The recommender system handles the abundance of information by filtering the most crucial information based on the information given by a user and other criteria that take into account the user's choice and interest. It determines whether a user and an item are compatible and then assumes that they are similar in order to make recommendations. There are several methods of implementing a recommender system, but most commonly, they can be broken down into two categories: collaborative filtering and content-based filtering.

Collaborative filtering approaches build a model from a user's past behavior as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in. For example, if a user A frequently purchases items that are also purchased by a user B, and user A has not purchased a specific item that user B has, the system may recommend that item to user A.

Content-based filtering approaches use item features to recommend items. For example, if a user likes a particular type of music, the system will recommend similar music based on the features of the original item. This method is based on the idea that if a user liked a particular item, they will also like items that are similar to it.

Hybrid recommenders, which combine collaborative filtering and content-based filtering methods, is a complicated algorithm that makes product recommendations based on your past behavior as well as on people that are similar to you. There are several companies that employ this strategy, such as Facebook, which displays news that is significant to you and to others in your network. LinkedIn also employs this strategy.

1.2 Explanation of the Topic

A book recommender system is a type of recommender system that suggests books to users based on their preferences and reading history.

We typically ask our friends or classmates for recommendations when we want to read a new book, or we could go through all the volumes at a library (as if we can). Even after asking around and looking, we could still be unable to locate the book we are looking for because not everyone has the same interests. For such circumstances, we require a system that considers our preferences and recommends some useful books to us.

Collaborative filtering-based book recommenders build a model from a user's past behavior as well as similar decisions made by other users. For example, if a user A frequently reads and rates books that are also read and rated highly by a user B, the system may recommend books that user B has read and rated highly but that user A has not yet read.

Content-based filtering-based book recommenders use features of the books to recommend similar books to a user. For example, if a user likes books with a certain genre or topic, the system will recommend books with similar genres or topics based on the features of the original book.

2. Background

2.1 Research

- *"Research on Book Recommendation Algorithm Based on Collaborative Filtering and Interest Degree"* by Zhi Hui Wang and De Zhi Hou [1]

Summary

It is challenging for readers to swiftly locate the books they desire while picking books due to the expansion of library resources. The importance of book recommendation systems is growing. This study suggests a collaborative filtering and interest-based book recommendation system based on the prior research. Consider the number of searches, borrowing intervals, borrowing times, and renewal rates as crucial measuring indices for the book's interest. The findings of the MAE and RMSE experiment analysis demonstrate that the approach suggested in this study converges more quickly than the conventional way.

Analysis

This study suggests a collaborative filtering and interest-based book recommendation system. Cosine similarity is used in collaborative filtering for analysis, and the interest degree employs the book's fundamental characteristics as a measuring index. It has been determined that the approach suggested in this work has a satisfactory convergence result after evaluating the statistical data of the library of Wuxi Vocational College of Science and Technology for many years utilising MAE and RMSE two measurement indicators for experimental analysis. In order to get superior convergence outcomes, the following phase aims to simultaneously improve the measurement indicators and the collaborative filtering method.

- *"Online Book Recommendation System using Collaborative Filtering" (With Jaccard Similarity)"* by Avi Rana and K. Deeba [2]

Summary

Software called the recommendation system (RS) makes suggestions for related products to customers based on their past purchases or preferences. RS looks through a vast database of items and generates a list of those that would satisfy the buyer's needs. Today, the majority of e-commerce businesses use recommendation algorithms to entice customers to make more purchases by presenting things that they are likely to enjoy. Amazon, Barnes & Noble, Flipkart, Goodreads, and other online book retailers utilise the Book Recommendation System to suggest books that customers might be enticed to purchase because they fit their preferences. Filtering, prioritising, and making correct suggestions are difficulties they encounter. To create lists of products that are comparable to the buyer's tastes, RS systems employ collaborative filtering (CF). Based on the premise that if a user has rated two books, then to a user who has rated three books, collaborative filtering Read one of these novels, and the other one is suggested (Collaboration). Due to issues with scalability, sparsity, and cold start, CF has trouble making precise suggestions.

Analysis

Even with the adaptation of a fitting algorithm for recommendation, the RS faces an obstacle because of large quantity of data that needs to be handled. According to the experimental results, the proposed algorithm with compact dataset was more accurate than existing algorithms with full datasets. In addition, JS uses the number of common users as a basis for measuring similarity, rather than the absolute ratings, as used by most existing algorithms, which gives a more accurate result. In order to provide suggestions that are more accurate, this research suggests using collaborative filtering using Jaccard similarity (JS). The foundation of JS is an index created for a pair of books. It is a ratio of common users—those who have given both books ratings—to the total number of users who have given each book a rating. The JS Index will be higher and, thus, provide better suggestions as the number of common users increases. Books with a high JS index will be at the top of the recommended books list (more recommended).

- *"Book Recommendation System Based on Combine Features of Content Based Filtering, Collaborative Filtering and Association Rule Mining"* by Tewari, A. S., Kumar, A., & Barman, A. G. [3]

Summary

Systems of recommendations are frequently employed to suggest the best items to end consumers. Today's online book retailers compete with one another in a variety of ways. One of the more powerful techniques for increasing earnings and keeping customers is the recommendation system. Books that will attract buyers must be recommended by the book recommendation system. This study proposes a method for recommending books that combines association rule mining, collaborative filtering, and content filtering.

Analysis

Most recommendation systems attempt to anticipate the buyer's interests and provide book recommendations in line with those predictions. By collaborating with other readers to filter the ratings of the books they have purchased, this book suggestion has taken into account a number of factors, including the book's quality and substance. To provide recommendations that are more persuasive, this recommender system also employs an associative model. Since the algorithm created the suggestions offline, it has no performance issues. Purpose of this book recommendation system is to recommend books to the buyer that suits their interest. This recommendation system works offline and stores recommendations in the buyer's web profile.

- *"Book Recommendation System Using Opinion Mining Technique"* by Sohail, S. S., Siddiqui, J., & Ali, R. [4]

Summary

The rate of change in daily life has also increased dramatically as a result of shifting technological trends and the rapid expansion of the Internet. These technologies have had such wide-ranging effects on society that they have touched practically every aspect of daily life. People are beginning to use internet services in their everyday lives, and they are increasingly favouring online buying for their necessities. It is highly laborious for academics, researchers, and students to choose the needed book from the vast online collections of books. In this research, we introduced a recommendation method based on opinion mining to suggest top-ranked books on several computer science disciplines. We've divided up the features for the books into categories based on client demand and feedback we've gathered from them. We evaluate the features based on user feedback as well as a number of classified qualities. Rankings are determined by allocating weights to category attributes based on their significance and usage. The top 10 novels are then listed. Millions of people who are looking for certain books should find this strategy useful.

Analysis

In this work, the product is a book, and N is 10. We plan to propose a recommender approach to recommend the top N items for some certain kind. We looked for the best books on various subjects, classify the attributes, and then rate the books based on these aspects. Numerous characteristics are given weights, and the weights are determined based on the significance of the function and the needs of the users. The top 10 novels are then listed. Only one issue, "cloud computing," is presented with results due to the page's limitation. These recommended books are thought to be among the greatest on the subject, and this effort may be beneficial for millions of clients looking for the best books.

- *"Collaborative Filtering for Book Recommendation System"* by Gautam Ramakrishnan, V. Saicharan, K. Chandrasekaran, M. V. Rathnamma and V. Venkata Ramana) [5]

Summary

One of the most crucial methods on the market now is collaborative filtering. It is common in practically all facets of the internet, including e-commerce, music, books, social networking, advertising, etc. This is because it deeply understands user demands and offers a convenient platform for users to find what they enjoy without having to search. One disadvantage of this approach is that it only takes into account the explicit feedback provided by the user in the form of a rating. Various implicit indications, such as views, read-later lists, etc., also show the user's true wants. The various methods to include implicit input into the recommendation system are proposed and compared in this research. In order to increase the number of entries in the table, the paper makes an effort to assign explicit ratings to users based on the implicit feedback that users have provided about certain books using a variety of methods.

Analysis

We showed how to leverage implicit feedback for a book recommendation system in a fresh and creative way. In conclusion, taking into account implicit evaluations outperformed the conventional approach of merely taking into account explicit ratings. Additionally, we want to enhance this approach in next works by incorporating other implicit data types that will provide a comprehensive picture of the person. Building a recommendation engine is essential to enhancing the user experience as the use of recommendation systems spreads. However, there is a fine line between gathering implicit input and protecting the user's privacy. We hope to advance the research covered in this publication in the future. Use of a user-user similarity metric is one of several potential expansions. We exclusively employed item-item similarity in our methodology. A new dataset, which provides a greater range of implicit input, might be used to test the same methodology. Due to the presence of numerous implicit feedback, the modification will need to be done.

2.2 Coursework 1 Review

[1.4 – Types of Recommendation Systems]

Examples of recommendation systems include Netflix's ability to suggest movies and Amazon's ability to suggest things based on previous purchases and browsing history.

The common types of recommendation systems are:

- Collaborative recommendation system
- Content-based recommendation system
- Hybrid recommendation system

[1.5 - Problem Domain]

While shopping in an online store, for instance, we are often presented with a bewildering array of things from which to choose, making it a more difficult task to settle on a final purchase. This issue is remedied by recommendation systems, which do data analysis on massive databases and then provide suggestions for user-specific services. Different methods for building recommendation systems, such as collaborative filtering, content-based filtering, and hybrid filtering, have emerged in recent years. Business expansion is a common use of recommendation systems. Both consumers and large corporations may benefit from them. On a technical level, they lessen the transaction costs associated with handling the results of an internet search. And as we've seen, these kinds of solutions have proven to be useful in facilitating the decision-making process. They contribute to the company's bottom line by proposing products the consumer would find useful.

As far as methods go, collaborative filtering is by far the most advanced and often used. Using the preferences of other users who share the active user's demographic and social context, collaborative filtering makes product recommendations. The usage of collaborative recommender systems has spread across several industries. Ringo, a web-based social information filtering system, uses collaborative filtering to generate user profiles inferred from music ratings.

[2.1.3 Advantages of the topic / problem domain]

- **Improved user experience**

  Recommendation algorithms aid in simulating the personalized care and in-store shopping experience that a knowledgeable sales associate may deliver to a hesitant online consumer.

- **Increase in Sales**

  Personalizing the shopping experience and highlighting relevant items leads to increased sales volume, higher average order value, and greater customer lifetime value.

- **Decision making**

  By tracking sales and customers and generating in-depth statistics, businesses may better inform their marketing, supply chain, and pricing decisions.

- **Revenue Growth**

  Considering the aforementioned factors, recommendation systems have the potential to be lucrative sources of revenue. Product recommendation solutions may help increase marketing-spend efficiency by 10-30% and revenue by 5-15%, as stated in McKinsey's 2019 article The Future of Personalization.

[2.1.4 Disdvantages of the topic / problem domain]

- **The cold-start issue**

  The data availability of similar users is crucial to the operation of collaborative filtering systems. If we were to create a brand-new recommendation system, we'd have no user data to work with at all. First, we could try out content filtering, and if it goes well, we could go on to collaborative filtering.

- **Scalability**

  The algorithms struggle to handle a growing user base. With 10 million customers and 100 thousand movies, you'd need a sparse matrix with 1 trillion components.

- **Insufficient accurate data**

  Ratings provided by humans are not always accurate, which might lead to inaccurate input data. Users' actions matter more than their ratings. The best answer here is item-based recommendations.

[2.1.5 - Dataset]

Datasets for this project has been extracted from the internet and it has the following attributes:

- **Books:** It has 8 columns; ISBN, Book title, Book author, Year of publication, Publisher, and three columns for Book cover Image URLs representing three different versions (small, medium, and large).
- **Users:** Contains the user's information. It consists of 3 columns UserID, Location, and age.
- **Ratings:** Contains the information on ratings of the books. It consists of 3 columns UserID, ISBN, and Book Rating.

3. Solution

3.1 Explanation of the Solution

We have implemented several Techniques for this recommendation system.

- **Popularity Based Recommendation:**
    - Popular in the Whole Collection: We have sorted the dataset according to the total ratings each of the books have received in non-increasing order and then recommended top n books.
    - Popular at a Given Place: The dataset was filtered according to a given place (city, state, or country) and then sorted according to total ratings they have received by the users in decreasing order of that place and recommended top n books.
    - Books by the Same Author, Publisher of Given Book Name: For this model, we have sorted the books by rating for the same author and same publisher of the given book and recommended top n books.
    - *Popular* Books Yearly: This is the most basic model in which we have grouped all the books published in the same year and recommended the top-rated book yearly.

- **Recommendation using Average Weighted Rating:** We have calculated the weighted score using the below formula for all the books and recommended the books with the highest score.

$$score = t/(t+m) * a + m/(m+t) * c$$

where,

t represents the total number of ratings received by the book

m represents the minimum number of total ratings considered to be included

a represents the average rating of the book and,

c represents the mean rating of all the books.

- **User-Item Collaborative Filtering Recommendation:** Collaborative Filtering Recommendation System works by considering user ratings and finds cosine similarities in ratings by several users to recommend books. To implement this, we took only those books' data that have at least 50 ratings in all.

- **Correlation Based Recommendation:** For this model, we have created the correlation matrix considering only those books which have total ratings of more than 50. Then a user-book rating matrix is created. For the input book using the correlation matrix, top books are recommended.

- **Nearest Neighbour Based Recommendation:** To train the Nearest Neighbours model, we have created a compressed sparse row matrix taking ratings of each Book by each User individually. This matrix is used to train the Nearest Neighbours model and then to find n nearest neighbors using the cosine similarity metric.

- **Content Based Recommendation:** This system recommends books by calculating similarities in Book Titles. For this, TF-IDF feature vectors were created for unigrams and bigrams of Book-Titles; only those books' data has been considered which are having at least 80 ratings.

- **Hybrid Approach (Collaborative + Content) Recommendation:** A hybrid recommendation system was built using the combination of both content-based filtering and collaborative filtering systems. A percentile score is given to the results obtained from both content and collaborative filtering models and is combined to recommend top n books.

3.2 Pseudocode

1. Load the GWO culture X_i (i = 1, 2, ..., n) // X_i represents random cluster positions in Movielens dataset.
2. Initialize r, R, and Q // r, R and Q are coefficient points.
3. Estimate the appropriateness of each explorer negotiator
4. X_α =finest explorer negotiator
5. X_β =2nd finest explorer negotiator
6. X_δ =3rd finest explorer negotiator
7. While (z < Most number of iterations) // z represents current iteration.
    - 7.1 for respective explorer negotiator
        - 7.1.1 Refresh the spot of the present explorer negotiator.
    - 7.2 end for
    - 7.3 Refresh r, R and Q.
    - 7.4 Determine the appropriateness of entire explorer negotiators.
    - 7.5 Refresh X_α, X_β, & X_δ
    - 7.6 z=z+1
8. end while
9. Return X_α // represents positions of centroids given by the grey wolf optimizer for Movielens data.
10. Randomly select cluster centre // Fuzzy c-means.

F matrix represents an association between clusters for Movielens dataset of users.

11. Load F= [f_{lm}] matrix, F(0)
12. Estimate the f_{lm} using:

$$f_{lm}^{m} = 1 \Big/ \sum_{k=1}^{c} \left[\frac{\|(X_l - c_m)\|}{\|(X_l - c_k)\|} \right] \frac{2}{n-1}$$ // represents association between i^{th} and j^{th} cluster in Movielens.

13. set k=0
14. At k-step: determine the midpoints B(k)=[c_m] with F(k)

$b_m = \frac{\sum_{l=1}^{N} f_{lm}^{n} x_l}{\sum_{l=1}^{N} f_{lm}^{n}}$ // represents the new position of the jth cluster for Movielens dataset.

15. Refresh F(k), F(k+1)

$$f_{lm}^{m} = 1 \Big/ \sum_{k=1}^{c} \left[\frac{\|(X_l - c_m)\|}{\|(X_l - c_k)\|} \right] \frac{2}{n-1}$$

16. If || F(k+1) - F(k)||< ε, then discontinue; else go back to step 12.
17. Return newly formed clusters and cluster centers for the Movielens users.

Figure 1 - Pseudocode of a Collaborative Recommender System

3.3 Diagrams

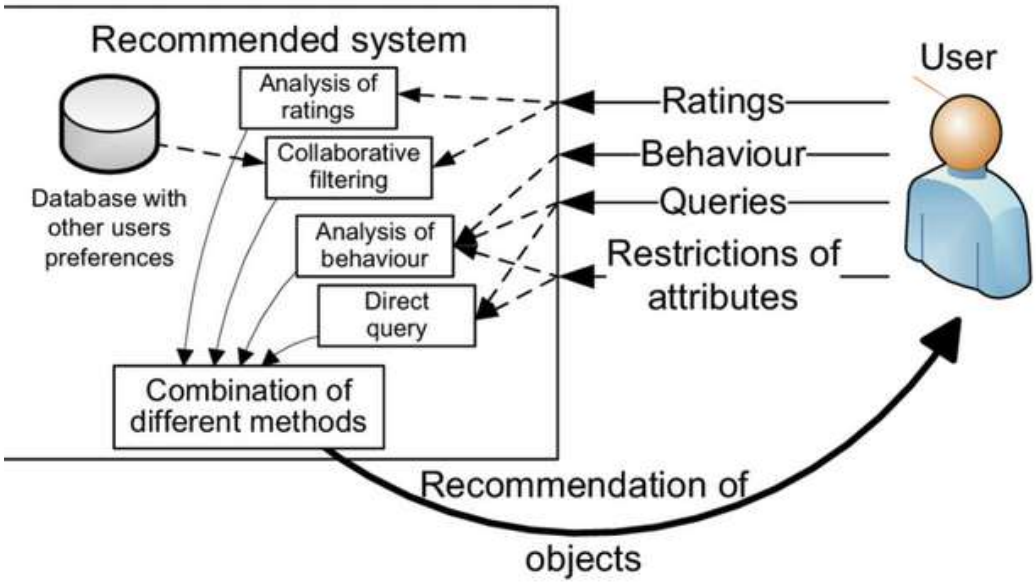


Figure 2 - Structure of a Recommender System

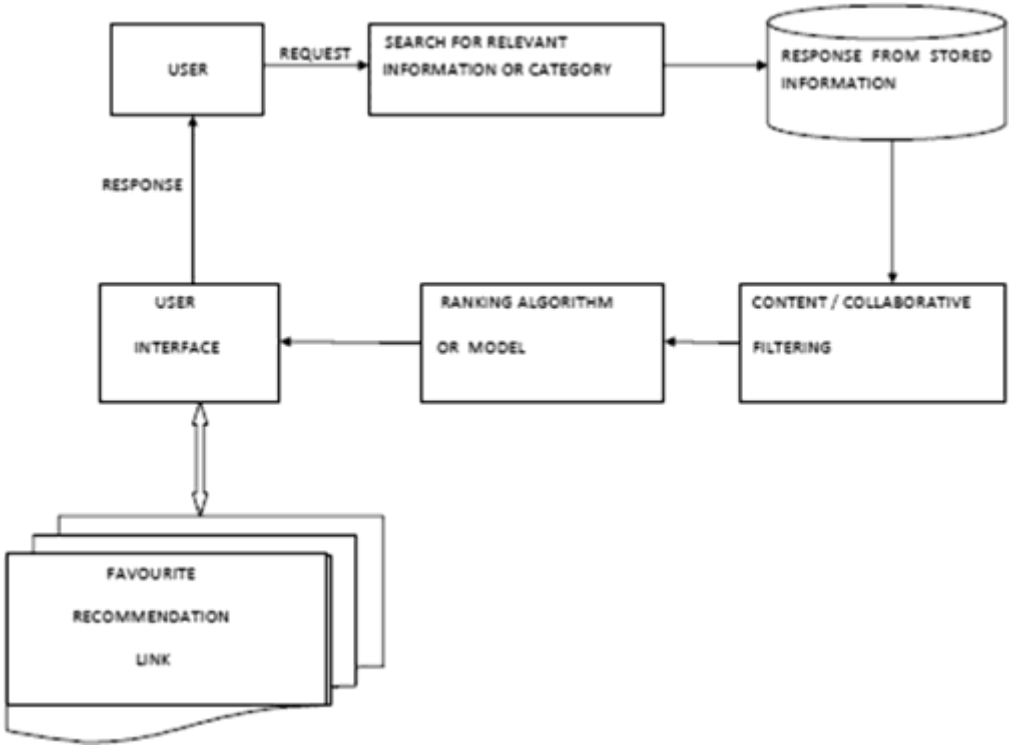


Figure 3 - Basic Diagram of Book Recommendation System

3.4 Explanation of Development Process

Here’s a high-level basic overview of the steps required to implement a user-based collaborative recommender system.

1. Collect and organize information on users and products

This is the crucial initial step. You must be aware of your users' identities and technologies. In our case, our users are book readers so we need to find out more information about the readers, and the books that are available. **This process is generally referred to as an ETL (extract-transform-load).**

As you collect the information, it's important to organize the material into some sort of standard form. With a standard form, a customer and the products he or she uses can be compared easily to other customers and other products.

2. Compare User A to all other users

Using those standard forms, you next design a function that compares User A to all other users. This function should create a set of users that are most similar to User A. Using common machine learning libraries like Python's scikit-learn, we are able to use the **Nearest Neighbours algorithm** out of the box on our transformed data to compute this user set.

This algorithm uses some of the most common distance measures (e.g., Euclidian) in linear algebra to compute the similarity between a set of vectors and a single query vector.

3. Create a function that finds products that User A has not used, but which similar users have

In this portion, we will find users that have similar interests to that of User A, and we will examine their vector to determine books which are new to user A, but are used by similar users.

This can be done using basic set theory operations on the set of Books used by the neighbours, and the set of Books used by User A:

{Books Used by Similar Users} – {Books Used by User A} = {Books to Recommend to User A}

These products are ones that a user-based recommender algorithm could recommend to the user. The intuition originates from the fact that because the users have used similar Books to User A, the additional Books that they have used may also be of interest to User A.

4. Rank and recommend

If we want to interest User A in new products, we'll increase our chances of success by assigning a higher rank to products that customers similar to User A already use.

We can extend the recommendation system by ranking the recommended items to User A. The greater the number of similar customers using a Klip, the higher the rank that Klip gets assigned.

The intuition behind this is that if many of the similar customers use a specific product, it is likely that User A will find it useful as well.

The system will then recommend to User A the items that rank highest on the list.

5. Evaluate and test

We can assume that the recommender is going to get it right the first time. So, we will test the accuracy of the recommendations the system generates by using the original collection of users and their products from Step 1.

Hence, we select a few users to act as “test users” to be compared to the remaining users.

For each test user, we remove some of the Books we know they have used. After processing them with Steps 2-5, we then see if the recommended results match those that were removed.

If their extracted items are recommended to them again, we know the system is working accurately.

3.5 Results

3.5.1 Importing Required Libraries

```
import re
import pickle
import operator
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
from scipy.sparse import csr_matrix
from pandas.api.types import is_numeric_dtype
from sklearn.neighbors import NearestNeighbors
from sklearn.feature_extraction import DictVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV

import warnings
warnings.filterwarnings("ignore")
```

3.5.2 Importing the Dataset

Dataset

```
books = pd.read_csv(r"/content/drive/MyDrive/Books Datasets/Books.csv", delimiter=';', error_b
users = pd.read_csv(r"/content/drive/MyDrive/Books Datasets/Users.csv", delimiter=';', error_b
ratings = pd.read_csv(r"/content/drive/MyDrive/Books Datasets/Book-Ratings.csv", delimiter=';'

print("Books Data:    ", books.shape)
print("Users Data:    ", users.shape)
print("Books-ratings: ", ratings.shape)
```

```
Books Data:     (271360, 8)
Users Data:     (278858, 3)
Books-ratings:  (1149780, 3)
```

3.5.3 Data pre-processing

Pre-Processing Books Data

Display and Cleaning the Data: This involves filling and/or dropping null values, dropping the un-important columns, and removing outlier values.

Pre-processing

Books Dataset Pre-processing

In [10]:
```
print("Columns: ", list(books.columns))
books.head()
```

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

Out[10]:

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	Image-URL-L
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0195153448.0...	http://images.amazon.com/images/P/0195153448.0...	http://images.amazon.com/images/P/0195153448.0...
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.0...	http://images.amazon.com/images/P/0002005018.0...	http://images.amazon.com/images/P/0002005018.0...
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.0...	http://images.amazon.com/images/P/0060973129.0...	http://images.amazon.com/images/P/0060973129.0...
3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.0...	http://images.amazon.com/images/P/0374157065.0...	http://images.amazon.com/images/P/0374157065.0...
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton & Company	http://images.amazon.com/images/P/0393045218.0...	http://images.amazon.com/images/P/0393045218.0...	http://images.amazon.com/images/P/0393045218.0...

In [11]:
```
## Drop URL columns
books.drop(['Image-URL-S', 'Image-URL-M', 'Image-URL-L'], axis=1, inplace=True)
books.head()
```

Out[11]:

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

Filling Null Values

In [12]:
```
## Checking for null values
books.isnull().sum()
```

Out[12]:
```
ISBN                 0
Book-Title           0
Book-Author          1
Year-Of-Publication  0
Publisher            2
dtype: int64
```

In [13]:
```
books.loc[books['Book-Author'].isnull(),:]
```

Out[13]:

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

In [14]:
```
books.loc[books['Publisher'].isnull(),:]
```

Out[14]:

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

Converting Year of publication into Numbers

In [15]:
```
books.at[187689 ,'Book-Author'] = 'Other'

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

In [16]:
```
## Checking for column Year-of-publication
books['Year-Of-Publication'].unique()
```

Out[16]:
```
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', 'DK Publishing Inc', '1943', '1938',
       '1900', '1942', '1923', '1920', '1933', 'Gallimard', '1909',
       '1946', '2008', '1378', '2030', '1936', '1947', '2011', '2020',
       '1919', '1949', '1922', '1897', '2024', '1376', '1926', '2037'],
      dtype=object)
```

```
In [ ]: pd.set_option('display.max_colwidth', -1)
```

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

Out[11]:

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

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

Out[12]:

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

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

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

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

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

Replacing Invalid Years with Max year

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

```
[0, 1376, 1378, 1806, 1897, 1900, 1901, 1902, 1904, 1906, 1908, 1909, 1910, 1911, 1914, 1917, 1919, 1920, 1921, 1922, 1923, 1
924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944,
1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965,
1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986,
1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2008,
2010, 2011, 2012, 2020, 2021, 2024, 2026, 2030, 2037, 2038, 2050]
```

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

Out[16]: [2002]

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

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

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

```
In [ ]: books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271047 entries, 0 to 271046
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ISBN                 271047 non-null  object
 1   Book-Title           271047 non-null  object
 2   Book-Author          271047 non-null  object
 3   Year-Of-Publication  271047 non-null  int32
 4   Publisher            271047 non-null  object
dtypes: int32(1), object(4)
memory usage: 9.3+ MB
```

Pre-Processing Users Data

Display and Cleaning the Data: This involves filling and/or dropping null values, dropping the un-important columns, and removing outlier values.

Users Dataset Pre-processing

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

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

Out[22]:

	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

In [ ]:
```
## Checking null values
print(users.isna().sum())
```

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

Check for all values present in Age Column

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

```
[nan, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.
0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 4
2.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0,
63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0, 81.0, 82.0, 83.0,
84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0, 100.0, 101.0, 102.0, 103.0, 1
04.0, 105.0, 106.0, 107.0, 108.0, 109.0, 110.0, 111.0, 113.0, 114.0, 115.0, 116.0, 118.0, 119.0, 123.0, 124.0, 127.0, 128.0,
132.0, 133.0, 136.0, 137.0, 138.0, 140.0, 141.0, 143.0, 146.0, 147.0, 148.0, 151.0, 152.0, 156.0, 157.0, 159.0, 162.0, 168.0,
172.0, 175.0, 183.0, 186.0, 189.0, 199.0, 200.0, 201.0, 204.0, 207.0, 208.0, 209.0, 210.0, 212.0, 219.0, 220.0, 223.0, 226.0,
228.0, 229.0, 230.0, 231.0, 237.0, 239.0, 244.0]
```

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

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

Out[26]: 35

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

Handling cases where city/state entries from city list as state is already given

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

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

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

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

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

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

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

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

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

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

```
4659
16044
```

Dropping Duplicate Rows

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

```
In [ ]: users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278858 entries, 0 to 278857
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype 
---  ------   --------------   ----- 
 0   User-ID  278858 non-null  int64 
 1   Age      278858 non-null  int32 
 2   City     278858 non-null  object
 3   State    278858 non-null  object
 4   Country  278858 non-null  object
dtypes: int32(1), int64(1), object(3)
memory usage: 9.6+ MB
```

Pre-Processing Books Ratings Data

Display and Cleaning the Data: This involves filling and/or dropping null values, dropping the un-important columns, and removing outlier values.

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

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

Out[32]:

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

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

```
Out[33]: User-ID        0
         ISBN           0
         Book-Rating    0
         dtype: int64
```

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

```
True
```

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

```
True
```

Checking User-ID and ISBN

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

```
True
```

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

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

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

```
False
```

Removing extra characters from ISBN (from ratings dataset) existing in books dataset

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

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

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

In [ ]:
```
ratings.info()
```

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

Merging all 3 Tables

Merging of all three Tables

Merging Books, Users and Rating Tables in One

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1031609 entries, 0 to 1031608
Data columns (total 11 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   ISBN                 1031609 non-null  object
 1   Book-Title           1031609 non-null  object
 2   Book-Author          1031609 non-null  object
 3   Year-Of-Publication  1031609 non-null  int32
 4   Publisher            1031609 non-null  object
 5   User-ID              1031609 non-null  int64
 6   Book-Rating          1031609 non-null  int64
 7   Age                  1031609 non-null  int32
 8   City                 1031609 non-null  object
 9   State                1031609 non-null  object
 10  Country              1031609 non-null  object
dtypes: int32(2), int64(2), object(7)
memory usage: 86.6+ MB
```

Divide the Data on the basis of Ratings

Divide complete data on the basis of Implicit and Explicit ratings datasets

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

Out[43]: (384074, 11)

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

Out[44]: (647535, 11)

In [ ]:
```
dataset1.head()
```

Out[45]:

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

Data Visualization

Data Visualization

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

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

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

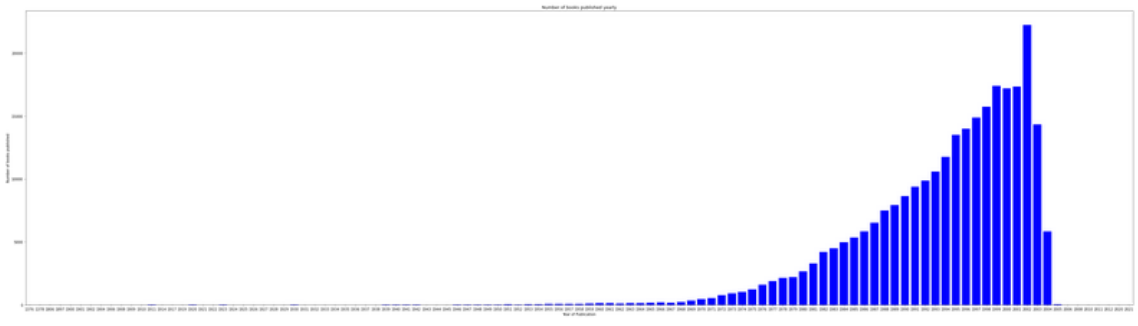


Figure 4 - Number of books published vs Year of Publication

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

Out[47]:
```
Text(0.5, 1.0, 'No of books by an author (Top 15)')
```

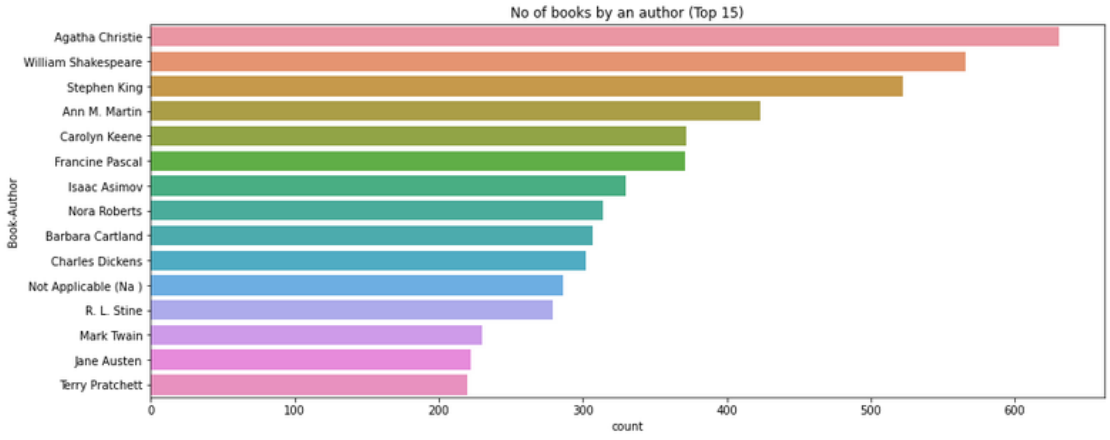
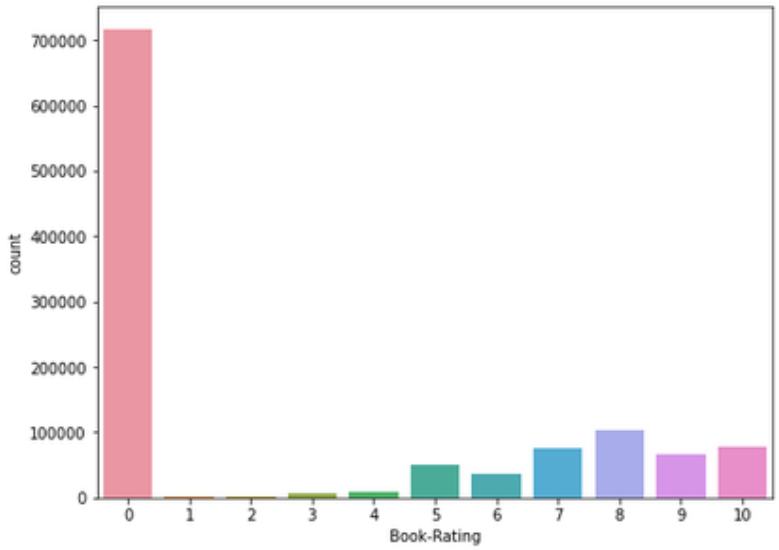


Figure 5 - Number of books by an Author (Top 15)

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

```
Out[49]: <AxesSubplot:xlabel='Book-Rating', ylabel='count'>
```




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

```
Out[50]: Text(0.5, 1.0, 'Explicit Ratings')
```

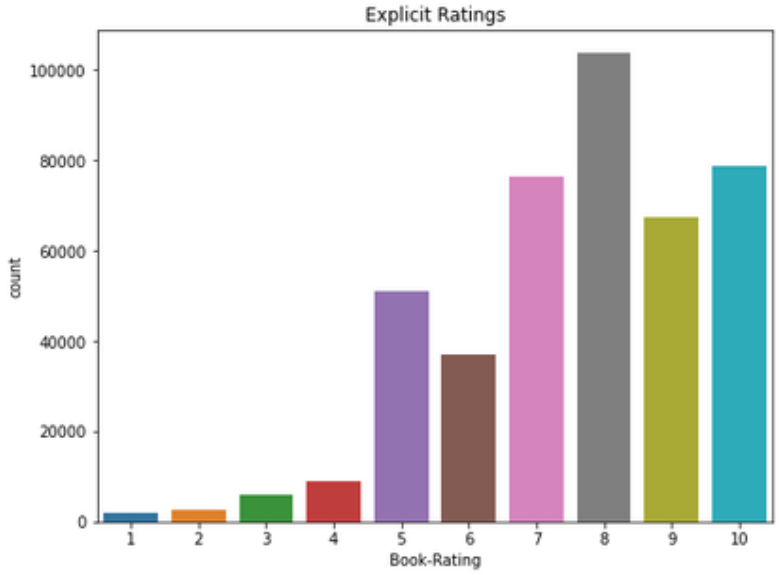


Figure 6 - Book Ratings

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

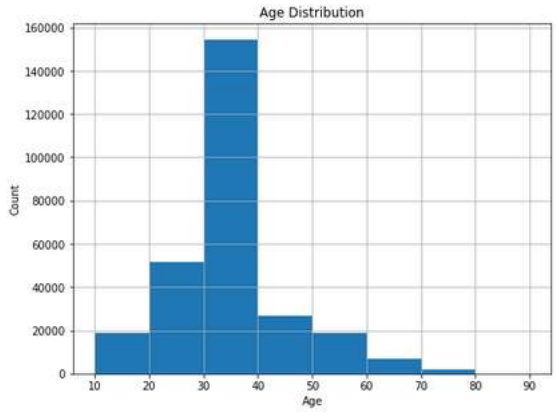
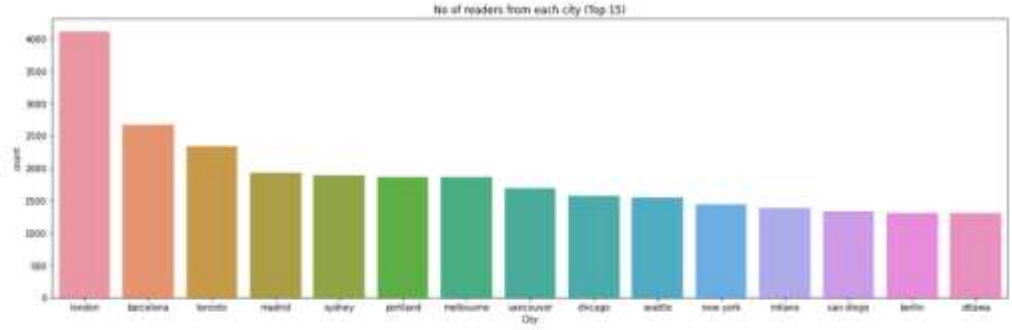


Figure 7 - Age Distribution of users

```
In [ ]: plt.figure(figsize=(20,6))
        sns.countplot(x="City", data=users,order=users['City'].value_counts().index[0:15])
        plt.title("No of readers from each city (Top 15)")
Out[53]: Text(0.5, 1.0, 'No of readers from each city (Top 15)')
```




```
In [ ]: plt.figure(figsize=(20,6))
        sns.countplot(x="State", data=users,order=users['State'].value_counts().index[0:15])
        plt.title("No of readers from each state (Top 15)")
Out[53]: Text(0.5, 1.0, 'No of readers from each state (Top 15)')
```

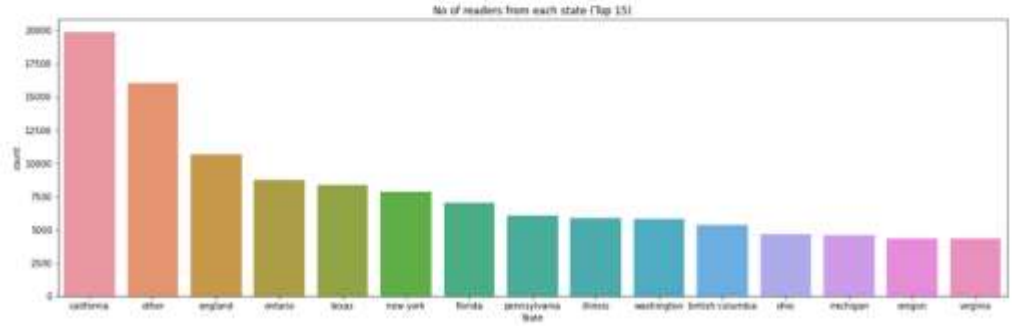


Figure 8 - Age Distribution of users

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

```
Out[54]: Text(0.5, 1.0, 'No of readers from each country (Top 10)')
```

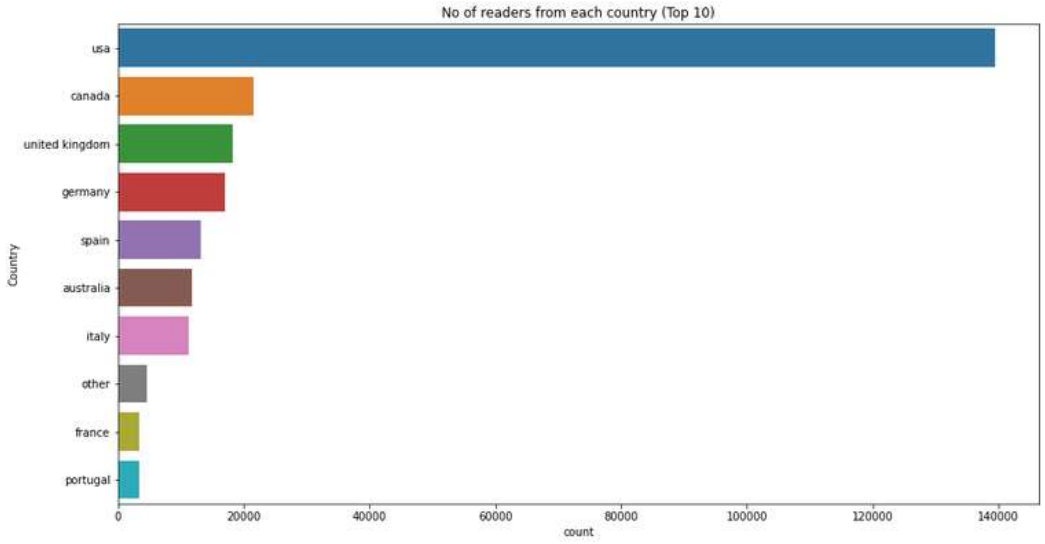


Figure 9 – Number of readers from each country

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

```
Out[55]: Text(0.5, 1.0, 'No of readers from states of USA (Top 15)')
```

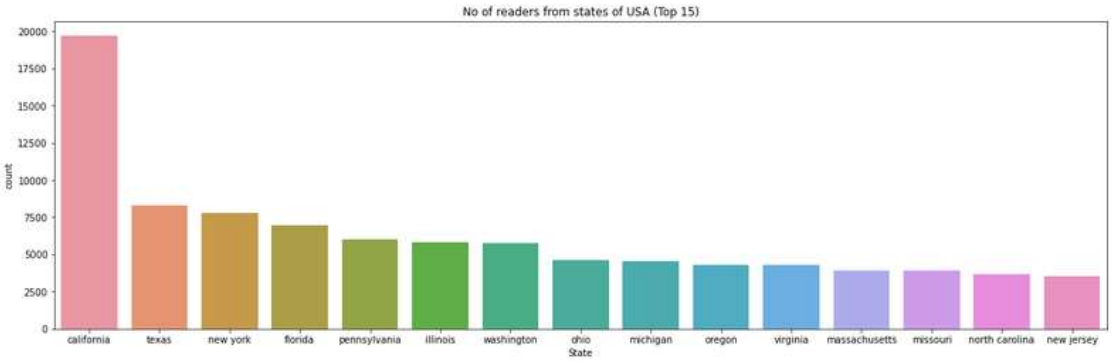


Figure 10 – Number of readers from USA

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

```
Out[56]: Text(0.5, 1.0, 'Number of Ratings for a book (Top 15)')
```

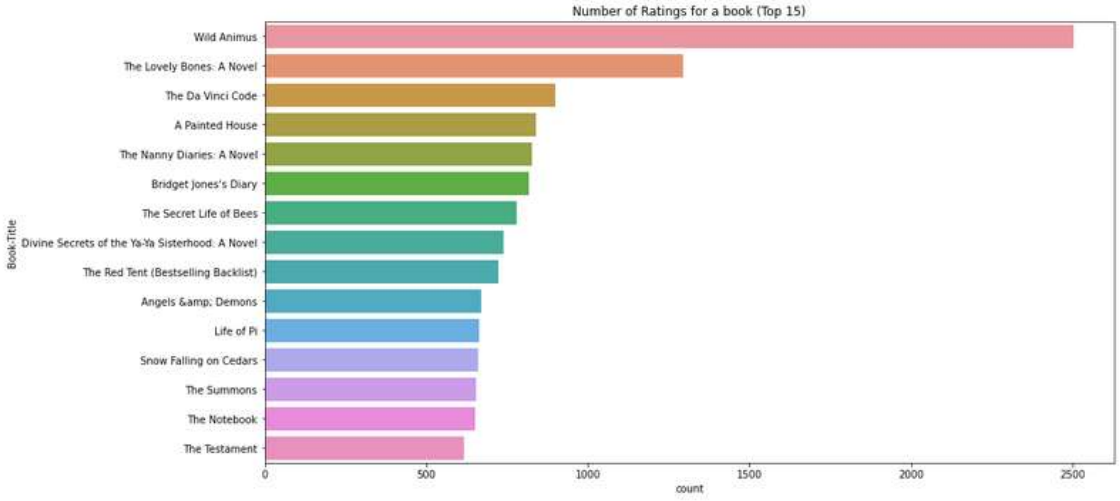


Figure 10 – Number of ratings for a Book

Various Implementations of Recommendation System (See 3.1)

1) Implementation of Popularity Based Recommendation (Whole Collection)

Recommendation Systems

In [ ]:
```
bookName = input("Enter a book name: ")
number = int(input("Enter number of books to recommend: "))

# Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))
```

```
Enter a book name: Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))
Enter number of books to recommend: 5
```

1. Popularity Based (Top In whole collection)

In [ ]:
```
def popularity_based(dataframe, n):
    if n >= 1 and n <= len(dataframe):
        data = pd.DataFrame(dataframe.groupby('ISBN')['Book-Rating'].count()).sort_values('Book-Rating', ascending=False).head(n)
        result = pd.merge(data, books, on='ISBN', left_index = True)
        return result
    return "Invalid number of books entered!!"
```

In [ ]:
```
print("Top", number, "Popular books are: ")
popularity_based(dataset1, number)
```

```
Top 5 Popular books are:
```

Out[59]:

	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher
401	0316666343	707	The Lovely Bones: A Novel	Alice Sebold	2002	Little, Brown
26	0971880107	581	Wild Animus	Rich Shapero	2004	Too Far
734	0385504209	488	The Da Vinci Code	Dan Brown	2003	Doubleday
513	0312195516	383	The Red Tent (Bestselling Backlist)	Anita Diamant	1998	Picador USA
1086	0060928336	320	Divine Secrets of the Ya-Ya Sisterhood: A Novel	Rebecca Wells	1997	Perennial

2) Implementation of Popularity Based Recommendation (Given Place)

2. Popularity Based (Top In a given place)

In [ ]:
```
def search_unique_places(dataframe, place):
    place = place.lower()

    if place in list(dataframe['City'].unique()):
        return dataframe[dataframe['City'] == place]
    elif place in list(dataframe['State'].unique()):
        return dataframe[dataframe['State'] == place]
    elif place in list(dataframe['Country'].unique()):
        return dataframe[dataframe['Country'] == place]
    else:
        return "Invalid Entry"
```

In [ ]:
```
place = input("Enter the name of place: ")
data = search_unique_places(dataset1, place)

if isinstance(data, pd.DataFrame):
    data = popularity_based(data, number)

data
```

```
Enter the name of place: India
```

Out[61]:

	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher
26	0971880107	3	Wild Animus	Rich Shapero	2004	Too Far
166	0671047612	2	Skin And Bones	Franklin W. Dixon	2000	Aladdin
164	0486284735	2	Pride and Prejudice (Dover Thrift Editions)	Jane Austen	1995	Dover Publications
9596	8171670407	2	Inscrutable Americans	Mathur Anurag	1996	South Asia Books
72379	0006944035	1	Secret Island / Secret Mountain (Two-in-ones)	Enid Blyton	1994	HarperCollins Publishers

3) Implementation of Books by same Author, Publisher of given book name

3. Books by same author, publisher of given book name

In [ ]:
```
def printBook(k, n):
    z = k['Book-Title'].unique()
    for x in range(len(z)):
        print(z[x])
        if x >= n-1:
            break
```

In [ ]:
```
def get_books(dataframe, name, n):
    print("\nBooks by same Author:\n")
    au = dataframe['Book-Author'].unique()

    data = dataset1[dataset1['Book-Title'] != name]

    if au[0] in list(data['Book-Author'].unique()):
        k2 = data[data['Book-Author'] == au[0]]
    k2 = k2.sort_values(by=['Book-Rating'])
    printBook(k2, n)

    print("\n\nBooks by same Publisher:\n")
    au = dataframe['Publisher'].unique()

    if au[0] in list(data['Publisher'].unique()):
        k2 = pd.DataFrame(data[data['Publisher'] == au[0]])
    k2=k2.sort_values(by=['Book-Rating'])
    printBook(k2, n)
```

In [ ]:
```
if bookName in list(dataset1['Book-Title'].unique()):
    d = dataset1[dataset1['Book-Title'] == bookName]
    get_books(d, bookName, number)
else:
    print("Invalid Book Name!!")
```

```
Books by same Author:

Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Order of the Phoenix (Book 5)
Harry Potter y el cÃ¡liz de fuego
Harry Potter and the Chamber of Secrets (Book 2)
Harry Potter and the Sorcerer's Stone (Book 1)


Books by same Publisher:

The Seeing Stone
The Slightly True Story of Cedar B. Hartley: Who Planned to Live an Unusual Life
Harry Potter and the Chamber of Secrets (Harry Potter)
The Story of the Seagull and the Cat Who Taught Her To Fly
Book! Book! Book!
```

4) Implementation of Books popular Yearly

4. Books popular Yearly

In [ ]:
```
data = pd.DataFrame(dataset1.groupby('ISBN')['Book-Rating'].count()).sort_values('Book-Rating', ascending=False)
data = pd.merge(data, books, on='ISBN', left_index = True)

years = set()
indices = []
for ind, row in data.iterrows():
    if row['Year-Of-Publication'] in years:
        indices.append(ind)
    else:
        years.add(row['Year-Of-Publication'])

data = data.drop(indices)
data = data.drop('Book-Rating', axis = 1)
data = data.sort_values('Year-Of-Publication')

pd.set_option("display.max_rows", None, "display.max_columns", None)
data
```

Out[65]:

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
253437	964442011X	Tasht-i khun	IsmaÂ°il Fasih	1376	Nashr-i Alburz
227222	9643112136	Dalan-i bihisht (Dastan-i Irani)	Nazi Safavi	1378	Intisharat-i Quqnus
171529	0781228956	Complete Works 10 Volumes [2,6,7,8,9] (Notable American Authors)	Benjamin Franklin	1806	Reprint Services Corp
211547	1551103982	The Cycling Adventures of Coconut Head: A North American Odyssey	Ted Schredd	1900	Graphic Arts Center Pub Co
262204	0671397214	JOY OF MUSIC P	Leonard Bernstein	1901	Fireside
102242	0373226888	Tommy's Mom	Linda O. Johnston	1902	Harlequin
45599	038528120X	CATCH 22	JOSEPH HELLER	1904	Delta

5) K-Nearest Neighbors

8. Nearest Neighbours Based

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

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

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

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

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

```
Recommended books:

Harry Potter and the Chamber of Secrets (Book 2)
Harry Potter and the Prisoner of Azkaban (Book 3)
Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Order of the Phoenix (Book 5)
The Fellowship of the Ring (The Lord of the Rings, Part 1)
```

6) Implementation of Average Weighted Ratings

In [ ]:
```
def avgRating(newdf, df):
    newdf['Average Rating'] = 0
    for x in range(len(newdf)):
        l = list(df.loc[df['Book-Title'] == newdf['Book-Title'][x]]['Book-Rating'])
        newdf['Average Rating'][x] = sum(l)/len(l)
    return newdf

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

# df = avgRating(df, dataset1)
# df.to_pickle('weightedData')
df = pd.read_pickle('weightedData')
```

In [ ]:
```
## C - Mean vote across the whole
C = df['Average Rating'].mean()

## Minimum number of votes required to be in the chart
m = df['Total-Ratings'].quantile(0.90)
```

In [ ]:
```
def weighted_rating(x, m=m, C=C):
    v = x['Total-Ratings']    #v - number of votes
    R = x['Average Rating']   #R - Average Rating
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [ ]:
```
df = df.loc[df['Total-Ratings'] >= m]

df['score'] = df.apply(weighted_rating, axis=1)
df = df.sort_values('score', ascending=False)

print("Recommended Books:-\n")
df.head(number)
```

```
Recommended Books:-
```

Out[69]:

	Book-Title	Total-Ratings	Average Rating	score
4794	Postmarked Yesteryear: 30 Rare Holiday Postcards	11	10	9.189906
7272	The Sneetches and Other Stories	8	10	9.002961
17	Harry Potter and the Prisoner of Azkaban (Book 3)	277	9	8.971768
28	Harry Potter and the Goblet of Fire (Book 4)	247	9	8.968407
42	Harry Potter and the Order of the Phoenix (Book 5)	211	9	8.963141

7) Collaborative Filtering

Selecting books with total ratings equals to or more than 50 (Because of availability of limited resources)

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

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

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

User - Item Collaborative Filtering

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

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

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

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

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

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

    print("\nRECOMMENDATIONS:\n")

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

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

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

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

RECOMMENDATIONS:

Harry Potter and the Prisoner of Azkaban (Book 3)
Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Order of the Phoenix (Book 5)
Harry Potter and the Chamber of Secrets (Book 2)
Fried Green Tomatoes at the Whistle Stop Cafe
```

8) Correlation Based

In [ ]:
```
popularity_threshold = 50

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

matrix = data.pivot_table(index='User-ID', columns='ISBN', values = 'Book-Rating').fillna(0)
```

In [ ]:
```
average_rating = pd.DataFrame(dataset1.groupby('ISBN')['Book-Rating'].mean())
average_rating['ratingCount'] = pd.DataFrame(ratings.groupby('ISBN')['Book-Rating'].count())
average_rating.sort_values('ratingCount', ascending=False).head()
```

Out[76]:

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

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

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

```
Recommended Books:
```

Out[77]:

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
0	0439064872	Harry Potter and the Chamber of Secrets (Book 2)	J. K. Rowling	2000	Scholastic
1	0439136369	Harry Potter and the Prisoner of Azkaban (Book 3)	J. K. Rowling	2001	Scholastic
2	0439139597	Harry Potter and the Goblet of Fire (Book 4)	J. K. Rowling	2000	Scholastic
3	0804115613	Fried Green Tomatoes at the Whistle Stop Cafe	Fannie Flagg	2000	Ballantine Books
4	0439139600	Harry Potter and the Goblet of Fire (Book 4)	J. K. Rowling	2002	Scholastic Paperbacks

9) Content Based

9. Content Based

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

Out[80]: (44652, 6)

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

Out[81]: (44652, 1112)

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

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

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

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

```
Recommended Books:

Harry Potter and the Sorcerer's Stone (Book 1)
Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Chamber of Secrets (Book 2)
Harry Potter and the Prisoner of Azkaban (Book 3)
Harry Potter and the Order of the Phoenix (Book 5)
```

10) Hybrid Approach (Content+Collaborative) Using percentile

10. Hybrid Approach (Content+Collaborative) Using percentile

In [ ]:
```
z = list()
k = float(1/number)
for x in range(number):
      z.append(1-k*x)

dictISBN = {}
for x in collaborative:
      dictISBN[x] = z[collaborative.index(x)]

for x in content:
    if x not in dictISBN:
        dictISBN[x] = z[content.index(x)]
    else:
        dictISBN[x] += z[content.index(x)]

ISBN = dict(sorted(dictISBN.items(),key=operator.itemgetter(1),reverse=True))
w=0
print("Input Book:\n")
print(bookName)
print("\nRecommended Books:\n")
for x in ISBN.keys():
    if w>=number:
        break
    w+=1
    print(x)
```

```
Input Book:

Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))

Recommended Books:

Harry Potter and the Goblet of Fire (Book 4)
Harry Potter and the Prisoner of Azkaban (Book 3)
Harry Potter and the Sorcerer's Stone (Book 1)
Harry Potter and the Chamber of Secrets (Book 2)
Harry Potter and the Order of the Phoenix (Book 5)
```

3.5.4 Separating Train and Test Values

Splitting Data and Applying KNN, SVD Algorithms on the data

Machine Learning - Model Selection

In [49]:
```
# creating a surprise object

reader = Reader(rating_scale=(0, 10))
data   = Dataset.load_from_df(rating[['User-ID','Book-Title','Book-Rating']], reader)


# Split the data into training & testing sets. Python's surprise documentation has the steps detailed out
# https://surprise.readthedocs.io/en/stable/FAQ.html

raw_ratings = data.raw_ratings
import random
random.shuffle(raw_ratings)                # shuffle dataset

threshold   = int(len(raw_ratings)*0.8)

train_raw_ratings = raw_ratings[:threshold] # 80% of data is trainset
test_raw_ratings  = raw_ratings[threshold:] # 20% of data is testset

data.raw_ratings = train_raw_ratings        # data is now the trainset
trainset         = data.build_full_trainset()
testset          = data.construct_testset(test_raw_ratings)
```

KNN (K Nearest Neighbours), memory based approach

This algorithm takes into consideration up-to 'K' nearest users (in user based collaborative filtering) or 'K' nearest items (in item based collaborative filtering) for making recommendations. By default, the algorithm is 'user-based', and k is 40 (kmin is 1). This means ratings of 40 nearest users are considered while recommending an an item to a user. Some variants of this algorithm include WithMeans, WithZscore & Baseline wherein the average rating of users, or the normalized ZScores of ratings or the baseline rating are also considered as the system generates recommendations

SVD (Singular Value Decomposition), model based approach

This algorithm takes a matrix factorization approach. The user-item rating matrix is factorized into smaller dimension user & item matrices consisting of latent factors (hidden characteristics). By default, number of latent factors is 100. These latent factors are able to capture the known user-item rating preference & in the process are able to predict an estimated rating for all user-item pair where user has not yet rated an item

In [50]:
```
# Trying KNN (K-Nearest Neighbors) & SVD (Singluar Value decomposition) algorithms using default model parameters

models=[KNNBasic(),KNNWithMeans(),KNNWithZScore(),KNNBaseline(),SVD()]
results = {}

for model in models:
    # perform 5 fold cross validation
    # evaluation metrics: mean absolute error & root mean square error
    CV_scores = cross_validate(model, data, measures=["MAE","RMSE"], cv=5, n_jobs=-1)

    # storing the average score across the 5 fold cross validation for each model
    result = pd.DataFrame.from_dict(CV_scores).mean(axis=0).\
             rename({'test_mae':'MAE', 'test_rmse': 'RMSE'})
    results[str(model).split("algorithms.")[1].split("object ")[0]] = result
```

MAE and RMSE of models

```
In [51]: performance_df = pd.DataFrame.from_dict(results)
         print("Model Performance: \n")
         performance_df.T.sort_values(by='RMSE')
```

```
Model Performance:
```

Out[51]:

	MAE	RMSE	fit_time	test_time
knns.KNNWithMeans	2.349277	3.288073	0.089588	1.120883
knns.KNNBaseline	2.352778	3.293826	0.120104	0.866165
matrix_factorization.SVD	2.389929	3.308103	0.461079	0.098767
knns.KNNWithZScore	2.325618	3.314696	0.170141	0.857288
knns.KNNBasic	2.440217	3.495338	0.057461	0.576574

- KNNWithMeans has the least RMSE (root mean square error) among KNN algorithms
- The model fit_time is the maximum for SVD but the model test_time is the least

3.5.6 Identifying the best SVM (SVC) paramaters using GridSearchCV

Tuning KNN

- KNNWithMeans has the least RMSE (root mean square error) among KNN algorithms
- The model fit_time is the maximum for SVD but the model test_time is the least

Machine Learning – Hyperparameter tuning with GridSearchCV

KNNWithMeans

- user_based: By default, this model parameter is 'True'. The other option 'False', corresponds to an item based approach
- min_support: This refers to number of items to consider (in user based approach) or number of users to consider (in item based approach) to calculate similarity before setting it to 0
- name: This refers to the distance measure that KNNWithMeans utilizes for calculating similarity. By default, the value is set as 'MSD' i.e., Mean Squared Distance. One other popular distance measure for rating based data is 'cosine' or angular distance. The cosine distance enables calculation of similarity among items & users accounting for inherent rating bias amongst users. E.g., users who like item2 twice as much as item1 may rate items as '8' & '4' if they are generous with their ratings but rate it only '4' & '2' if they are more stringent raters. MSD measures similarity based on the absolute ratings and will not be able to capture this inherent rating bias described above, however, cosine distance measure will be able to capture the same

```
In [52]: # Hyperparameter tuning - KNNWithMeans

param_grid = { 'sim_options' : {'name': ['msd','cosine'], \
                                'min_support': [3,5], \
                                'user_based': [False, True]}
             }

gridsearchKNNWithMeans = GridSearchCV(KNNWithMeans, param_grid, measures=['mae', 'rmse'], \
                                      cv=5, n_jobs=-1)

gridsearchKNNWithMeans.fit(data)

print(f'MAE Best Parameters:  {gridsearchKNNWithMeans.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchKNNWithMeans.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchKNNWithMeans.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchKNNWithMeans.best_score["rmse"]}\n')
```

Tuning SVD

- Post Hyperparameter Tuning with GridSearchCV, the best parameters are found to be different for MAE & RMSE metrics
- 'Cosine' distance measure, min_support of 3 & user_based : False i.e., item based approach have been chosen for building recommendations

The logic/code below can be modified to make recommendations using 'MSD' distance & user based method if needed

In [ ]:
```
# Model fit & prediction - KNNWithMeans

sim_options = {'name':'cosine','min_support':3,'user_based':False}
final_model = KNNWithMeans(sim_options=sim_options)

# Fitting the model on trainset & predicting on testset, printing test accuracy
pred = final_model.fit(trainset).test(testset)

print(f'\nUnbiased Testing Performance:')
print(f'MAE: {accuracy.mae(pred)}, RMSE: {accuracy.rmse(pred)}')
```

- The MAE & RMSE metrics for testset are comparable with what was obtained using cross validation & hyperparameter tuning stages with trainset. Chosen model hence, generalizes well

SVD

- n_factors: This refers to number of latent factors (hidden characteristics) for matrix factorization/ dimensionality reduction. By default, the value is 100
- n_epochs: This refers to number of iterations of stochiastic gradient descent procedure, utilized by SVD for learning the parameters and minimizing error
- lr_all & reg_all: i.e., learning rate and regularization rate. Learning rate is the step size of the said (above) SGD algorithm whereas regularization rate prevents overlearning, so the model may generalize well on data it has not yet seen. By default these values are set as 0.005 & 0.02

In [ ]:
```
# Hyperparameter tuning - SVD

param_grid = {"n_factors": range(10,100,20),
              "n_epochs" : [5, 10, 20],
              "lr_all"   : [0.002, 0.005],
              "reg_all"  : [0.2, 0.5]}

gridsearchSVD = GridSearchCV(SVD, param_grid, measures=['mae', 'rmse'], cv=5, n_jobs=-1)

gridsearchSVD.fit(data)

print(f'MAE Best Parameters:  {gridsearchSVD.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchSVD.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchSVD.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchSVD.best_score["rmse"]}\n')
```

- Post Hyperparameter Tuning with GridSearchCV, the best parameters are found to be different for MAE & RMSE metrics
- 'n_factors':50, 'n_epochs':10, 'lr_all':0.005 & 'reg_all': 0.2 have been chosen for building recommendations

In [ ]:
```
# Model fit & prediction - SVD

final_model = SVD(n_factors=50, n_epochs=10, lr_all=0.005, reg_all= 0.2)

# Fitting the model on trainset & predicting on testset, printing test accuracy
pred = final_model.fit(trainset).test(testset)

print(f'\nUnbiased Testing Performance')
print(f'MAE: {accuracy.mae(pred)}, RMSE: {accuracy.rmse(pred)}')
```

- The MAE & RMSE metrics for testset are comparable with what was obtained using cross validation & hyperparameter tuning stages with trainset. Chosen model hence again, generalizes well

SVD

- n_factors: This refers to number of latent factors (hidden characteristics) for matrix factorization/ dimensionality reduction. By default, the value is 100
- n_epochs: This refers to number of iterations of stochiastic gradient descent procedure, utilized by SVD for learning the parameters and minimizing error
- lr_all & reg_all: i.e., learning rate and regularization rate. Learning rate is the step size of the said (above) SGD algorithm whereas regularization rate prevents overlearning, so the model may generalize well on data it has not yet seen. By default these values are set as 0.005 & 0.02

In [ ]:
```
# Hyperparameter tuning - SVD

param_grid = {"n_factors": range(10,100,20),
              "n_epochs" : [5, 10, 20],
              "lr_all"   : [0.002, 0.005],
              "reg_all"  : [0.2, 0.5]}

gridsearchSVD = GridSearchCV(SVD, param_grid, measures=['mae', 'rmse'], cv=5, n_jobs=-1)

gridsearchSVD.fit(data)

print(f'MAE Best Parameters:  {gridsearchSVD.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchSVD.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchSVD.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchSVD.best_score["rmse"]}\n')
```

- Post Hyperparameter Tuning with GridSearchCV, the best parameters are found to be different for MAE & RMSE metrics
- 'n_factors': 50, 'n_epochs': 10, 'lr_all' 0.005 & 'reg_all': 0.2 have been chosen for building recommendations

In [ ]:
```
# Model fit & prediction - SVD

final_model = SVD(n_factors=50, n_epochs=10, lr_all=0.005, reg_all= 0.2)

# Fitting the model on trainset & predicting on testset, printing test accuracy
pred = final_model.fit(trainset).test(testset)

print(f'\nUnbiased Testing Performance')
print(f'MAE: {accuracy.mae(pred)}, RMSE: {accuracy.rmse(pred)}')
```

- The MAE & RMSE metrics for testset are comparable with what was obtained using cross validation & hyperparameter tuning stages with trainset. Chosen model hence again, generalizes well

4. Conclusion

4.1 Analysis

The rate of change in daily life has also increased dramatically as a result of shifting technological trends and the rapid expansion of the Internet. These technologies have had such wide-ranging effects on society that they have touched practically every aspect of daily life. People are beginning to use internet services in their everyday lives, and they are increasingly favouring online buying for their necessities. It is highly laborious for academics, researchers, and students to choose the needed book from the vast online collections of books. In this report, we introduced a recommendation method based on several methods, to suggest the best books across a variety of users.

4.2 Solution in the Real World

Recommendation systems are one of the biggest uses of AI in the world today. Giants like Netflix, YouTube, Amazon and many others use these systems to gain the attention of customers, and enhance the sales. This recommendation system is essentially useful for Book Selling Websites, and it has a lot of potential in the industry,

4.3 Further Work

Improving the proposed recommendation system is a task that the experts and developers should do interactively to tune the system. The objective is that the system can adapt to the collaboration process. However, future works need to be done. The automatic recommender together with the pedagogical support module will automatically offer students and tutors warnings and recommendations about the collaboration process. Moreover, the structure of the model was built manually with the help of the author.

5. REFERENCES

- https://www.hindawi.com/journals/wcmc/2021/7036357/ [1]
- https://iopscience.iop.org/article/10.1088/1742-6596/1362/1/012130/pdf [2]
- https://ieeexplore.ieee.org/abstract/document/6779375 [3]
- https://ieeexplore.ieee.org/abstract/document/6637421 [4]
- **https://link.springer.com/chapter/10.1007/978-981-15-0184-5_29** [5]
- **https://www.researchgate.net/figure/Basic-diagram-of-book-recommendation-system_fig3_326579202**
- **https://www.sciencedirect.com/science/article/abs/pii/S0957417413005095**
- **https://ashima96.medium.com/building-a-book-recommendation-system-a98c58a4f1bb**