- Student name: Shayan Abdul Karim Khan
- Student pace: Self Paced
- Scheduled project review date/time: July 5th 2023
- Instructor name: Abhineet Kulkarni

Summary

A recommendation system was built for a startup looking to start a quarterly book subscribtion business. The data was obtained from the Book-Crossing Community and cleaned for outliers and then further processed using different packagess, including paandas, numpy, sci-kit learn and pycountry. The data was filtered for US users and books with number of ratings greater than 19. The data with missing raatings waasas set aside to use for reecommending new books that users haven't rated yet while the rest was used to model KNN, SVD and NMF algorithms. These aalgorithms were evaluated and compared based on differet metrics. RMSE was chosen as the main metric for comparison through which SVD was chosen as the final model.

Problem Overview

A startup is planning on launching a books care package monthly subscription system where the user will be deliverd 5 books every quarter. The user will have the option to return whichever books they do not intend to keep within 7 days of receiving. Therefore it is crucial to ensure that the books delivered match the customer's preference otherwise returns can cause significant losses.

The intention is to use the ratings that users have given previous books to compute the 5 books that will be delivered. The task s to build a recommendation system that can take in certain features of the user and their history of book ratings and recommend the Top 5 books for that user.

Data Sources

The following dataset is used to create a recommendation system.

1. Ratings: https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset?select=Ratings.csv (https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset?select=Ratings.csv)
2. Books: https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset?select=Books.csv (https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset?select=Books.csv)
3. Users: https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset?select=Users.csv (https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset?select=Users.csv)

This dataset contains data on more than 200,000 users with demographic information and collected from the reputable Book-Crossing community. It has also been cleaned to a certain extent which gives us leverage to focus more on the modelling part. This dataset contains features that are readilly available and will not make it extremely difficult for the company to expand the dataset.

A recommendaation system based on less complicated data might not be highly accurate but it provides the option to add more features as needed. It is more difficult to unwind a complex model as compared to one using simple features.

Initial Data Understanding and Data Cleaning

The Book-Crossing dataset comprises 3 files.

- 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 case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three different flavours (`Image-URL-S` , `Image-URL-M` , `Image-URL-L`), i.e., small, medium, large. These URLs point to the Amazon web site.

- 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.

- Users

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

In [654]:
```
#import initial libraries
import pandas as pd
import numpy as numpy
from plotly import express as px
```

Books

First load the dataset and preview the dataset.

In [655]:
```
#load the dataset into a pandas dataframe
books = pd.read_csv("data/Books.csv")

#preview the dataset
books.head()
```

```
/var/folders/qv/0z2v23tn1f1b2fnpqqgqsxch0000gn/T/ipykernel_86122/3637232235.py:2: DtypeWarning:

Columns (3) have mixed types. Specify dtype option on import or set low_memory=False.
```

Out[655]:

	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 &amp; 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...

The ISBN is the unique identifier for every book but for recommendations, we have to look at the book titles and authors. Considering that book titles are extracted from Amazon's database, there is hiigh cconfidence that they are correct.

Image URLs won't be useful for analysis so it can be dropped. Year of Publication and Publisher are also unique charaacteristics to explore in the Exploratory Data Analysis section to understand ppulraity of individul data points.

First, lets look at the info of the books dataframe to understand the characteristics of the dataset.

In [656]:
```
#look at the info
books.info()
```

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

```
In [657]:  #check number of missing values
           books.isnull().sum()
```

```
Out[657]: ISBN                   0
          Book-Title             0
          Book-Author            1
          Year-Of-Publication    0
          Publisher              2
          Image-URL-S            0
          Image-URL-M            0
          Image-URL-L            3
          dtype: int64
```

There are very few missing values in the relevant columns. 2 in the Publisher column and 1 in the Book Author column. Considering that there are more than 250,000+ records, dropping the rows with missing data won't impact the data.

All of the columns are stored as strings. Year of Publication is also stored as a string. This will need to be changed to datetime format for proper analysis later.

```
In [658]:  #drop the Image URL columns
           books.drop(["Image-URL-S","Image-URL-M","Image-URL-L"],axis = 1,inplace = True)

           #drop the records with missing values
           books.dropna(inplace = True)

           #check info for the new dataset
           books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 271357 entries, 0 to 271359
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   ISBN                 271357 non-null  object
 1   Book-Title           271357 non-null  object
 2   Book-Author          271357 non-null  object
 3   Year-Of-Publication  271357 non-null  object
 4   Publisher            271357 non-null  object
dtypes: object(5)
memory usage: 12.4+ MB
```

Next, explore the year of publication before it is transformed to datetime format. We will cross-check whether the values stored in here are valid.

```
In [659]:  #check unique values for year of publication
           books["Year-Of-Publication"].unique()
```

```
Out[659]: 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)
```

There are two unique values that don't fit in the year of publication column. The wrong vaalues are DK Publishing Inc and Gallimard.

These values look like names of publishers. Considering that records with missing values have been dropped, this means that these records already have values for all the other columns therefore it will be best to drop these records.

We also see some outlier values like 1376, 2037 and more. We will limit the range of data that we take forward to ensure that the dataa we make recommendations on is valid. The year limitations will be 1800 to 2024. This also ensures that the books recommended are readilly available to be provided to the customer.

In [660]:
```
#remove values of publishing names
remove_value = ["DK Publishing Inc","Gallimard"]
books        = books[~books["Year-Of-Publication"].isin(remove_value)]

#cnveert to int type
books["Year-Of-Publication"] = books["Year-Of-Publication"].astype(int)

#filter the years
books = books[ books["Year-Of-Publication"] > 1800]
books = books[ books["Year-Of-Publication"] < 2024]
```

```
/var/folders/qv/0z2v23tn1f1b2fnpqqgqsxch0000gn/T/ipykernel_86122/3949624841.py:6: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [661]:
```
from datetime import datetime #import datetime

#convert to year format
books['Year-Of-Publication'] = pd.to_datetime(books['Year-Of-Publication'], format='%Y').dt.year
```

In [662]:
```
books["Year-Of-Publication"].unique()
```

Out[662]:
```
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, 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, 1911, 1904,
       1949, 1932, 1928, 1929, 1927, 1931, 1914, 1934, 1910, 1933, 1902,
       1924, 1921, 1900, 1944, 1917, 1901, 2010, 1908, 1906, 1935, 1806,
       2021, 2012, 2006, 1909, 2008, 1919, 1922, 1897])
```

In [663]:
```
#check the info of the new dataset
books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 266721 entries, 0 to 271359
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   ISBN                 266721 non-null  object
 1   Book-Title           266721 non-null  object
 2   Book-Author          266721 non-null  object
 3   Year-Of-Publication  266721 non-null  int64 
 4   Publisher            266721 non-null  object
dtypes: int64(1), object(4)
memory usage: 12.2+ MB
```

In [664]:
```
#check for null values
books.isnull().sum()
```

Out[664]:
```
ISBN                   0
Book-Title             0
Book-Author            0
Year-Of-Publication    0
Publisher              0
dtype: int64
```

With the initial data cleaning done, lets do a final check to see if there are any duplicate records

In [665]:
```
books.duplicated().sum()
```

Out[665]:
```
0
```

Now we can move onto looking at the other dataset.

Ratings

Next we can look at the Ratings dataset.

In [666]:
```
#load the data
ratings = pd.read_csv("data/Ratings.csv")
#preview the data
ratings.head()
```

Out[666]:

	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

This dataset contains the user IDs of speific users and the ISBN of the book they rated. These ISBNs can be used to merge this data with the Books dataset.

It also contains the ratings that the user assigned to every book out of 10. This will be the most cruciaal piece of information for the recommendation system.

In [667]:
```
#check info of dataset
ratings.info()
```

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

Theere are no missing values and User ID and the rating columns are integer types which we would expect. Also, the ISBN column is an object column, same as the books dataset.

There are more than 1 million records, almost double the number of books we have. Although this is large dataset, only double the number of the books dataset shows us that there might not be a lot of reviews per books. We will explore this further in the EDA section since we want to ensure that there is a minimum number of reviews per book to have confidence in the recommendations.

Lets explore the characteristics of the Ratings column.

In [668]:
```
#check statistics of the raatins column
ratings['Book-Rating'].describe()
```

Out[668]:
```
count    1.149780e+06
mean     2.866950e+00
std      3.854184e+00
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      7.000000e+00
max      1.000000e+01
Name: Book-Rating, dtype: float64
```

The mean is only ~2.9 which is really low. The quartiles give us an indication of why that might be. 50% of the values aare zero or less. Zero typically indicates that the user hass not read this book. We have to deal with these values later during EDA as we explore the distribution.

In [669]:
```
import numpy as np

# Replace zeros with NaN in column 'A'
ratings['Book-Rating'] = ratings['Book-Rating'].replace(0, np.nan)

#preview the dataset
ratings
```

Out[669]:

	User-ID	ISBN	Book-Rating
0	276725	034545104X	NaN
1	276726	0155061224	5.0
2	276727	0446520802	NaN
3	276729	052165615X	3.0
4	276729	0521795028	6.0
...	...	...	...
1149775	276704	1563526298	9.0
1149776	276706	0679447156	NaN
1149777	276709	0515107662	10.0
1149778	276721	0590442449	10.0
1149779	276723	05162443314	8.0

1149780 rows × 3 columns

In [670]:
```
#check for duplicates
ratings.duplicated().sum()
```

Out[670]: 0

Considering there are no duplicates, we can move onto evaluating the other datasets.

Users

In [671]:
```
#load the dataset
users = pd.read_csv("data/Users.csv")
#preview the dataset
users.head()
```

Out[671]:

	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

This dataset contains Age and location information on users. This can be valuable demogrpahic information for the models.Nonetheless, for location, we want to focus only on countries before we make data too granular since there might not be enough information for every city and statee/province.

In [672]:
```
#chech info
users.info()
```

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

There are more than 250,000 records in this dataset which is almost 4 times less that the ratiings dataset. This is a good indication that there are atleast more than one review available for aa good chunk of the users.

The Age column is the only one with missing data therefore these have to be handled accordingly. Lets explore the Age column to understand the information it contains. Unfortunately, we do not have enough variables to be able to impute the missing values with high confidence. Therefore we will drop these records.

In [673]:
```
#check statistics
users.describe()["Age"]
```

Out[673]:
```
count    168096.000000
mean         34.751434
std          14.428097
min           0.000000
25%          24.000000
50%          32.000000
75%          44.000000
max         244.000000
Name: Age, dtype: float64
```

Looks like the mean and standard deviation give us aa good indication of the spreaad of the data but there are 25% records for less than 24 year olds and the minimum is also 0. This definitely shows that there is anomalies in this dataset. Also, the max being 244 also showcases that there are anomalies on the higher end also.

The company wants to ideally focus its efforts on the customer base they believe will use their online services. Users under 16 are usually dependent on their parents while older people above 60 might not be active in the online space. Therefore it makees sense to first focus on a niche rather than people from all age groups. Therefore, the ages will be filtered for 16-60. This should also drop the missing values.

In [674]:
```
#filter for ovver 16
users = users[users["Age"] >= 16]

# filter for undeer 60
users =users[users["Age"] < 60]

#preeview the dataset
users.head()
```

Out[674]:

	User-ID	Location	Age
1	2	stockton, california, usa	18.0
3	4	porto, v.n.gaia, portugal	17.0
9	10	albacete, wisconsin, spain	26.0
12	13	barcelona, barcelona, spain	26.0
17	18	rio de janeiro, rio de janeiro, brazil	25.0

Cross-check to make sure that the missing values have been dropped.

In [675]:
```
#check for missing values
users.isnull().sum()
```

Out[675]:
```
User-ID      0
Location     0
Age          0
dtype: int64
```

In [676]:
```
#check the new statistics
users.describe()["Age"]
```

Out[676]:
```
count    152206.000000
mean         33.735497
std          11.296174
min          16.000000
25%          25.000000
50%          32.000000
75%          42.000000
max          59.000000
Name: Age, dtype: float64
```

Looks like the quartiles are much better spread out and showcase a proper aage range that we would expect.

Now we need to clean the location column. Since the focus needs to be on the countries to understand how the dataa varies between countires, we will filter out the countries separately.

```
In [677]:   #replace the first comma
            users["Location"] = users.Location.str.replace(",,",",")

            #use the second comma to separate out the country names
            users["Country"] = users.Location.map(lambda x : x.split(",")[-1].lower())

            #look at the unique country names
            users["Country"].unique()
```

```
Out[677]: array([' usa', ' portugal', ' spain', ' brazil', ' germany', ' mexico',
             ' china', ' canada', ' italy', ' united kingdom', ' france',
             ' netherlands', ' iraq', ' new zealand', ' india', ' ghana',
             ' switzerland', ' iran', ' bosnia and herzegovina', ' australia',
             ' sri lanka', ' belgium', ' malaysia', ' turkey', ' philippines',
             ' finland', ' norway', ' greece', ' chile', ' taiwan', ' pakistan',
             ' españa', ' denmark', ' nigeria', ' romania', ' argentina',
             ' singapore', ' vietnam', ' tunisia', ' egypt', ' uzbekistan',
             ' qatar', ' syria', ' austria', ' indonesia', '', ' sudan',
             ' saudi arabia', ' thailand', ' ireland', ' venezuela',
             ' mozambique', ' morocco', ' colombia', ' spain"', '"', ' sweden',
             ' poland', ' slovakia', ' bulgaria', ' basque country',
             ' ethiopia', ' portugal"', ' japan', ' albania', ' cuba',
             ' russia', ' nigeria"', ' jersey', ' belarus', ' cape verde"',
             ' lithuania', ' costa rica', ' guyana', ' scotland', ' jordan',
             ' la argentina', ' angola', ' algeria', ' andorra', ' kyrgyzstan',
             ' slovenia', ' ecuador', ' kuwait', ' dominican republic',
             ' turkey"', ' brunei', ' bangladesh', ' hong kong', ' panama',
             ' monterrey', ' bahrain', ' united arab emirates', ' yugoslavia',
             ' israel', ' urugua', ' peru', ' l`italia', ' dominica',
             ' honduras', ' france"', ' sierra leone', ' guatemala', ' mali',
             ' germany"', ' estonia', ' rwanda', ' trinidad and tobago',
             ' yemen', ' croatia', ' kazakhstan', ' öð¹ú', ' la france',
             ' eritrea', ' cameroon', ' india"', ' nicaragua', ' zambia',
             ' maroc', ' belize', ' south africa', ' namibia', ' uruguay',
             ' macedonia', ' argentina"', ' botswana', ' luxembourg',
             ' queenspark', ' monaco', ' samoa', ' brasil', ' guernsey',
             ' euskal herria', ' bermuda', ' georgia', ' barbados',
             ' united kingdom"', ' armenia', ' south korea', ' hungary',
             ' austria"', ' ukraine', ' iceland', ' galiza', ' suriname',
             ' jamaica', ' &#32654;&#22269;', ' afghanistan', ' latvia',
             ' u.s. virgin islands', ' catalunya', ' moldova',
             ' czech republic', ' finland"', ' burma', ' ghana"', ' ksa',
             ' tajikistan', ' azerbaijan', ' nepal', ' cote d`ivoire',
             ' maldives', ' catalunya(catalonia)', ' mã?â©xico',
             ' caribbean sea', ' peru"', ' lebanon', ' hong kong"',
             ' antigua and barbuda', ' saint vincent and the grenadines"',
             ' lleida', ' cayman islands', ' u.s.a.', ' iran"', ' u.a.e',
             ' paraguay', ' belgique', ' deutsches reich', ' catalonia',
             ' micronesia', ' cyprus', ' bahamas', ' bhutan', ' guinea-bissau',
             ' cote d`ivoire"', ' wales', ' equatorial geuinea',
             ' philippines"', ' goteborg', ' norway"', ' united states',
             ' netherlands"', ' zimbabwe', ' oman', ' bolivia', ' thailand"',
             ' ama lurra', ' hamilton', ' fiji', ' the', ' catalunya spain',
             ' malta', ' switzerland"', ' deutschland', ' papua new guinea',
             ' costa rica"', ' slovakia"', ' brazil"',
             ' saint vincent and the grenadines', ' burkina faso', ' ?ú?{',
             ' kenya', ' new zealand"', ' gabon', ' italia', ' puerto rico',
             ' north korea', ' commonwealth of northern mariana islands',
             ' mauritius', ' benin', ' colombia"', ' holy see', ' cherokee',
             ' espaã?â±a', ' la belgique', ' sweden"', ' n/a - on the road',
             ' chile"', ' algérie', ' egypt"', ' alderney', ' el salvador',
             ' republic of korea', ' côte d', ' croatia"', ' greece"',
             ' ouranos', ' denmark"', ' here and there', ' malawi', ' espaã±a',
             ' solomon islands', ' romania"', ' england', ' iceland"',
             ' lesotho', ' antarctica', ' chad', ' fifi', ' djibouti',
             ' america', ' ireland"', ' marshall islands', ' la suisse',
             ' netherlands antilles', ' méxico', ' congo', ' ä¸\xadå?½',
             ' bangladesh"', ' hungary"', ' china"', ' grenada', ' p.r.china',
             ' liberia', ' usa & canada', ' uganda', ' malaysia"',
             ' sao tome and principe"', ' vietnam"', ' poland"', ' slovenia"',
             ' sicilia', ' sri lanka"', ' san marino', ' macedonia"',
             ' china öð¹ú', ' czech republic"', ' cambodia', ' turkmenistan',
             ' hillsborough', ' greece (=hellas)', ' isle of man',
             ' channel islands', ' 5057chadwick ct.', ' far away...', ' laos',
             ' togo', ' senegal', ' sudan"', ' niger', ' guatemala"', ' orense',
             ' cape verde', ' mexico"', ' lombardia', ' strongbadia',
             ' universe', ' berguedà', ' ysa', 'lawrenceville', ' serbia',
             ' perãº', ' aotearoa', ' suisse', ' trinidad and tobago"', ' guam',
             ' burma"', ' andorra"', ' tanzania', ' saint lucia', ' n/a',
             ' tonga', ' haiti', ' roma', ' l`algérie', ' vanuatu', ' uganda"',
             ' _ brasil', ' mauritius"', ' united kindgdom', ' hungary and usa',
             ' pakistan"', ' macau', ' united state', ' the netherlands',
             ' singapore"', ' pender', ' vicenza', ' p.r.c', ' quit', ' guinea',
             ' indonesia"', ' swaziland', ' phillipines', ' trinidad', ' l',
             ' wonderful usa', ' burlington', ' madagascar', ' swazilandia',
             ' u.k.', ' santa barbara', ' mongolia', ' korea',
             ' saint kitts and nevis', ' comoros', ' morocco"', ' holland',
             ' lithuania"', ' tobago', ' venezuela"', ' madrid', ' thing',
             ' tanzania"', ' españa"', ' \\"n/a\\""', ' mozambique"',
             ' w. malaysia', ' le madagascar', ' everywhere and anywhere',
             ' chinaöð¹ú', ' galiza neghra', ' asturies', ' libya', ' palau'],
            dtype=object)
```

There are a few characters in place of country names. We can take the special character lists and replace them out to make it simpler for processing.

In [678]:
```
#delete the special characters
for special_char in """.!'",\/1234567890;&#?-{}[]()=_öð¹ú """:
    users["Country"] = users.Country.map(lambda x : x.replace(special_char,""))

#remove the white space
users["Country"] = users.Country.map(lambda x : x.strip())
```

We need to check if the country names are valid. We will use the pycountry library to validate the ccountry names and check how many records have valid country names.

In [679]:
```
import pycountry

# identify the column to check
country_column = 'Country'

# Get a set of valid country names from pycountry
valid_countries_set = list(set(country.name.lower() for country in pycountry.countries))
valid_countries_set += ["usa","russia"]

# Check and flag invalid country names in the DataFrame
users['Is_Valid_Country'] = users[country_column].apply(lambda x: x in valid_countries_set)

# Filter out the rows with invalid country names
invalid_countries = users[~users['Is_Valid_Country']]
valid_countries = users[users['Is_Valid_Country']]

# Print the invalid country names
print("Invalid Country Percentage:")
print(invalid_countries[country_column].unique())
print()
print("Invalid Country Shape",invalid_countries.shape)
print()
print("Invalid Country Record Percentage",invalid_countries.shape[0]/users.shape[0]*100)
# print(invalid_countries[country_column].unique())
print("="*80)
# Print the valid country names
print("Valid Country Percentage:")
print(valid_countries[country_column].unique())
print()
print("Valid Country Shape", valid_countries.shape)
print()
print("Valid Country Record Percentage",valid_countries.shape[0]/users.shape[0]*100)
```

```
Invalid Country Percentage:
['unitedkingdom' 'newzealand' 'iran' 'bosniaandherzegovina' 'srilanka'
 'taiwan' 'españa' 'vietnam' 'syria' '' 'saudiarabia' 'venezuela'
 'basquecountry' 'capeverde' 'costarica' 'scotland' 'laargentina'
 'dominicanrepublic' 'brunei' 'hongkong' 'monterrey' 'unitedarabemirates'
 'yugoslavia' 'urugua' 'l`italia' 'sierraleone' 'trinidadandtobago'
 'lafrance' 'maroc' 'southafrica' 'macedonia' 'queenspark' 'brasil'
 'euskalherria' 'southkorea' 'galiza' 'usvirginislands' 'catalunya'
 'moldova' 'czechrepublic' 'burma' 'ksa' 'coted`ivoire'
 'catalunyacatalonia' 'mãâ©xico' 'caribbeansea' 'antiguaandbarbuda'
 'saintvincentandthegrenadines' 'lleida' 'caymanislands' 'uae' 'belgique'
 'deutschesreich' 'catalonia' 'micronesia' 'guineabissau' 'wales'
 'equatorialgeuinea' 'goteborg' 'unitedstates' 'bolivia' 'amalurra'
 'hamilton' 'the' 'catalunyaspain' 'deutschland' 'papuanewguinea'
 'burkinafaso' 'italia' 'puertorico' 'northkorea'
 'commonwealthofnorthernmarianaislands' 'holysee' 'cherokee' 'espaãâ±a'
 'labelgique' 'naontheroad' 'algérie' 'alderney' 'elsalvador'
 'republicofkorea' 'côted' 'ouranos' 'hereandthere' 'espaã±a'
 'solomonislands' 'england' 'fifi' 'america' 'marshallislands' 'lasuisse'
 'netherlandsantilles' 'méxico' 'ä¸\xadå½' 'prchina' 'usacanada'
 'saotomeandprincipe' 'sicilia' 'sanmarino' 'hillsborough' 'greecehellas'
 'isleofman' 'channelislands' 'chadwickct' 'faraway' 'laos' 'orense'
 'lombardia' 'strongbadia' 'universe' 'berguedà' 'ysa' 'lawrenceville'
 'perãº' 'aotearoa' 'suisse' 'tanzania' 'saintlucia' 'na' 'roma'
 'l`algérie' 'unitedkindgdom' 'hungaryandusa' 'macau' 'unitedstate'
 'thenetherlands' 'pender' 'vicenza' 'prc' 'quit' 'swaziland'
 'phillipines' 'trinidad' 'l' 'wonderfulusa' 'burlington' 'swazilandia'
 'uk' 'santabarbara' 'korea' 'saintkittsandnevis' 'holland' 'tobago'
 'madrid' 'thing' 'wmalaysia' 'lemadagascar' 'everywhereandanywhere'
 'galizaneghra' 'asturies']

Invalid Country Shape (17156, 5)

Invalid Country Record Percentage 11.271566166905378
================================================================================
Valid Country Percentage:
['usa' 'portugal' 'spain' 'brazil' 'germany' 'mexico' 'china' 'canada'
 'italy' 'france' 'netherlands' 'iraq' 'india' 'ghana' 'switzerland'
 'australia' 'belgium' 'malaysia' 'turkey' 'philippines' 'finland'
 'norway' 'greece' 'chile' 'pakistan' 'denmark' 'nigeria' 'romania'
 'argentina' 'singapore' 'tunisia' 'egypt' 'uzbekistan' 'qatar' 'austria'
 'indonesia' 'sudan' 'thailand' 'ireland' 'mozambique' 'morocco'
 'colombia' 'sweden' 'poland' 'slovakia' 'bulgaria' 'ethiopia' 'japan'
 'albania' 'cuba' 'russia' 'jersey' 'belarus' 'lithuania' 'guyana'
 'jordan' 'angola' 'algeria' 'andorra' 'kyrgyzstan' 'slovenia' 'ecuador'
 'kuwait' 'bangladesh' 'panama' 'bahrain' 'israel' 'peru' 'dominica'
 'honduras' 'guatemala' 'mali' 'estonia' 'rwanda' 'yemen' 'croatia'
 'kazakhstan' 'eritrea' 'cameroon' 'nicaragua' 'zambia' 'belize' 'namibia'
 'uruguay' 'botswana' 'luxembourg' 'monaco' 'samoa' 'guernsey' 'bermuda'
 'georgia' 'barbados' 'armenia' 'hungary' 'ukraine' 'iceland' 'suriname'
 'jamaica' 'afghanistan' 'latvia' 'tajikistan' 'azerbaijan' 'nepal'
 'maldives' 'lebanon' 'paraguay' 'cyprus' 'bahamas' 'bhutan' 'zimbabwe'
 'oman' 'fiji' 'malta' 'kenya' 'gabon' 'mauritius' 'benin' 'malawi'
 'lesotho' 'antarctica' 'chad' 'djibouti' 'congo' 'grenada' 'liberia'
 'uganda' 'cambodia' 'turkmenistan' 'togo' 'senegal' 'niger' 'serbia'
 'guam' 'tonga' 'haiti' 'vanuatu' 'guinea' 'madagascar' 'mongolia'
 'comoros' 'libya' 'palau']

Valid Country Shape (135050, 5)

Valid Country Record Percentage 88.72843383309463
```

Looks like almost 89% of the data has valid country names. We will keep these records and drop the other 11% that does not have valid country names.

In [680]:
```
#droop the column with the Indicator
users = valid_countries.drop(["Is_Valid_Country"],axis = 1)

#preview the dataset
users.head()
```

Out[680]:

	User-ID	Location	Age	Country
1	2	stockton, california, usa	18.0	usa
3	4	porto, v.n.gaia, portugal	17.0	portugal
9	10	albacete, wisconsin, spain	26.0	spain
12	13	barcelona, barcelona, spain	26.0	spain
17	18	rio de janeiro, rio de janeiro, brazil	25.0	brazil

We can drop the loccation column and keep the Country column. If we need to make our data further granulaar, we can add this back in.

In [681]:
```
#drop location column
users.drop(['Location'],axis=1, inplace=True)
```

In [682]:
```
#preview the dataset
users.head()
```

Out[682]:

	User-ID	Age	Country
1	2	18.0	usa
3	4	17.0	portugal
9	10	26.0	spain
12	13	26.0	spain
17	18	25.0	brazil

In [683]:
```
#check for duplicates
users.duplicated().sum()
```

Out[683]: 0

In [684]:
```
#check dataframe info
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 135050 entries, 1 to 278851
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   User-ID  135050 non-null  int64  
 1   Age      135050 non-null  float64
 2   Country  135050 non-null  object 
dtypes: float64(1), int64(1), object(1)
memory usage: 4.1+ MB
```

With no duplicate values and no missing values, we can move onto combining the three datasets for EDA.

Data Preparation

To prepare the data for EDA, we will have to combine the different datasets. Lets take a look at how these datasetss vary in their shapes.

In [685]:
```
#check shape
ratings.shape
```
Out[685]:
```
(1149780, 3)
```
In [686]:
```
#check shape
users.shape
```
Out[686]:
```
(135050, 3)
```
In [687]:
```
#check shape
books.shape
```
Out[687]:
```
(266721, 5)
```
We will first filter out the ratings dataset to keep the user IDs that we filtered in the users dataset to ensure that we have the same userss that we have information for. We will do the same things for books.

In [688]:
```
#filter for user IDs
ratings = ratings[ratings["User-ID"].isin(users["User-ID"].unique())]

#filter for book ISBN
ratings = ratings[ratings["ISBN"].isin(books["ISBN"].unique())]
```
In [689]:
```
#check new shape
ratings.shape
```
Out[689]:
```
(653360, 3)
```
Looks like we cut down the dataset in half. it is still 3 times the size of the the books dataset and aalmost 5 times the size of the users dataset.

In [690]:
```
#merge ratings and books
rating_books = ratings.merge(books, how='inner', on='ISBN')
```
In [691]:
```
#preview the dataset
rating_books.head()
```
Out[691]:

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher
0	276727	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books
1	638	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books
2	3363	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books
3	7158	0446520802	10.0	The Notebook	Nicholas Sparks	1996	Warner Books
4	8253	0446520802	10.0	The Notebook	Nicholas Sparks	1996	Warner Books

In [692]:
```
#check shape
rating_books.shape
```
Out[692]:
```
(653360, 7)
```
In [693]:
```
#check for missing values
rating_books.isna().sum()
```
Out[693]:
```
User-ID                     0
ISBN                        0
Book-Rating            421494
Book-Title                  0
Book-Author                 0
Year-Of-Publication         0
Publisher                   0
dtype: int64
```

In [694]:
```
#chek for duplicate values
rating_books.duplicated().sum()
```
Out[694]:
```
0
```

Considering there are no missing or duplicate values, we don't need to clean the data for bad data significantly anymore. Before we merge our last dataset, lets make sure that the olumnss we filtered out diidn't have any anomalous data leaked in.

In [695]:
```
#check info
rating_books.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 653360 entries, 0 to 653359
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   User-ID              653360 non-null  int64  
 1   ISBN                 653360 non-null  object 
 2   Book-Rating          231866 non-null  float64
 3   Book-Title           653360 non-null  object 
 4   Book-Author          653360 non-null  object 
 5   Year-Of-Publication  653360 non-null  int64  
 6   Publisher            653360 non-null  object 
dtypes: float64(1), int64(2), object(4)
memory usage: 39.9+ MB
```

In [696]:
```
#check year of publication filter
rating_books['Year-Of-Publication'].describe()
```
Out[696]:
```
count    653360.000000
mean       1995.373893
std           7.330963
min        1897.000000
25%        1992.000000
50%        1997.000000
75%        2001.000000
max        2021.000000
Name: Year-Of-Publication, dtype: float64
```

The min and max values are within the ranges that the data was filtered for. Lets join the user dataset to this and finalize the dataset for EDA.

In [697]:
```
#merge user dataset
user_rating_books = rating_books.merge(users, how='inner', on='User-ID')

#preview the dataset
user_rating_books.head()
```
Out[697]:

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Age	Country
0	276727	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books	16.0	australia
1	638	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books	20.0	usa
2	638	0316666343	10.0	The Lovely Bones: A Novel	Alice Sebold	2002	Little, Brown	20.0	usa
3	638	0375400699	10.0	Love in the Time of Cholera (Everyman's Librar...	GABRIEL GARCIA MARQUEZ	1997	Everyman's Library	20.0	usa
4	638	0385504209	10.0	The Da Vinci Code	Dan Brown	2003	Doubleday	20.0	usa

```
In [698]:  #check for missing values
            user_rating_books.isna().sum()

Out[698]:  User-ID                     0
           ISBN                        0
           Book-Rating            421494
           Book-Title                  0
           Book-Author                 0
           Year-Of-Publication         0
           Publisher                   0
           Age                         0
           Country                     0
           dtype: int64
```

```
In [699]:  #check for duplicates
            user_rating_books.duplicated().sum()

Out[699]:  0
```

With no missing or duplicate values, this dataset does not need any basic cleaning. We can move onto ensuring that our fiilters for Age hold and that there is no data that has leaked in.

```
In [700]:  #check info
            user_rating_books.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 653360 entries, 0 to 653359
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   User-ID              653360 non-null  int64  
 1   ISBN                 653360 non-null  object 
 2   Book-Rating          231866 non-null  float64
 3   Book-Title           653360 non-null  object 
 4   Book-Author          653360 non-null  object 
 5   Year-Of-Publication  653360 non-null  int64  
 6   Publisher            653360 non-null  object 
 7   Age                  653360 non-null  float64
 8   Country              653360 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 49.8+ MB
```

```
In [701]:   #check age value counts
            user_rating_books['Age'].value_counts()
```

```
Out[701]: 33.0    31394
29.0    27135
28.0    24754
30.0    24643
32.0    24477
34.0    24278
31.0    23570
36.0    23533
25.0    20664
44.0    20478
26.0    20474
38.0    20174
27.0    19946
43.0    19541
37.0    19474
35.0    18527
47.0    17889
23.0    17322
24.0    17151
52.0    17038
39.0    15953
46.0    15930
41.0    14698
40.0    14529
49.0    13498
51.0    13131
45.0    11186
22.0    10673
42.0    10035
21.0     9360
54.0     9022
58.0     8689
50.0     8648
57.0     8199
56.0     8128
18.0     8125
48.0     7239
53.0     6604
20.0     5981
55.0     5617
17.0     5078
19.0     4917
16.0     2966
59.0     2692
Name: Age, dtype: int64
```

```
In [702]:   #check age statistics
            user_rating_books['Age'].describe()
```

```
Out[702]: count    653360.000000
mean         36.233371
std          10.301461
min          16.000000
25%          28.000000
50%          35.000000
75%          44.000000
max          59.000000
Name: Age, dtype: float64
```

Lastly, before we start performing EDA, we want to ensure that the ratings we are basing book recommendations on are valid. Filtering out the dataset based on a minimum number of reviews for a book is a common practice in collaborative filtering-based recommendation systems. It can help improve the quality and reliability of recommendations by ensuring that books with insufficient user feedback are not included in the system.

In [703]:
```
#group by book unique identifier ISBN and aggregate number of book ratings
num_reviews = user_rating_books.groupby("ISBN").count()

#check value counts for number of book ratings
num_reviews["Book-Rating"].value_counts()
```

Out[703]:
```
0      98254
1      73220
2      14988
3       5952
4       2985
       ...
326        1
124        1
90         1
88         1
73         1
Name: Book-Rating, Length: 134, dtype: int64
```

The zeroes are NaN values that we already discussed and will deal with later. For now, lets move onto exploring the distribution of the counts. We know from domain knowlege that 100 reviews are convincing benchmark for users to consistently identify a good book. Lets check the distribution of books wiht number of reviews greater than 100.

In [704]:
```
#plot a histogram
px.histogram(num_reviews[num_reviews['Book-Rating']>99]['Book-Rating'],
             title="Number of Reviews distribution",nbins=20)
```

There is a very small percentage of books with 100 oor greater number of reviews. Lets take a look at books with less than 100 number of reviews but greater than zero.

```
In [705]:   #plot histogram
            px.histogram(num_reviews[(num_reviews['Book-Rating']<100) & num_reviews['Book-Rating']>0]['Book-Rating'],
                         title="Number of Reviews distribution",nbins=20)
```

The majority of books have less than 100 reviews and between 1-4 reviews.

We can do “CI” calculations systematically for a large number of books at a fixed level of confidence: 95%. What this means is, that we can expect the “true” ratings for the books (after thousands of further ratings) to still lie within those earlier Confidence Intervals in 95% of cases.

We can run a bootstrap method to see which number of ratings will work. We will run the maximum number of ratings in the bins in the ggraph up there to see how they will perform and how wide spread the confidence interval can be.

In [706]:
```
#create raanges for the max limits of the bins
for votes in range(4,40,5):
    indices = num_reviews[num_reviews['Book-Rating']==votes].index #extract indices of books
    ci_low = [] #create array for lower bounds of ci
    ci_up = [] #create array for upper bounds of ci

    #iterate through the books
    for index in indices:
        rats = user_rating_books[(user_rating_books['ISBN']==index) &
                      (~user_rating_books['Book-Rating'].isna())]['Book-Rating'].values

        # Perform bootstrapping
        bootstrapped_means = []
        num_resamples = 1000
        for _ in range(num_resamples):
            resampled_data = np.random.choice(rats, size=len(rats), replace=True)
            bootstrapped_mean = np.mean(resampled_data)
            bootstrapped_means.append(bootstrapped_mean)

        # Calculate confidence interval
        ci_lower, ci_upper = np.percentile(bootstrapped_means, [2.5, 97.5])
        ci_low.append(ci_lower)
        ci_up.append(ci_upper)

    #calculate the means
    mean_ci_low = np.mean(ci_low)
    mean_ci_up = np.mean(ci_up)

    # Print the confidence interval
    print('----------------------------------------------------------------------------------------------')
    print("Number of Ratings: ", votes)
    print("Confidence Interval: {:.2f} to {:.2f}".format(mean_ci_low, mean_ci_up))
    print("Out of a rating of 10, the true rating can lie in the following % range: ",
          (mean_ci_up-mean_ci_low)*100/10)
    print('----------------------------------------------------------------------------------------------')
```

```
---------------------------------------------------------------------------------------------
Number of Ratings:  4
Confidence Interval: 6.48 to 8.96
Out of a rating of 10, the true rating can lie in the following % range:  24.830883584589618
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  9
Confidence Interval: 6.80 to 8.72
Out of a rating of 10, the true rating can lie in the following % range:  19.243306559571618
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  14
Confidence Interval: 6.98 to 8.64
Out of a rating of 10, the true rating can lie in the following % range:  16.61239495798319
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  19
Confidence Interval: 7.16 to 8.62
Out of a rating of 10, the true rating can lie in the following % range:  14.624197689345317
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  24
Confidence Interval: 7.08 to 8.37
Out of a rating of 10, the true rating can lie in the following % range:  12.908909574468073
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  29
Confidence Interval: 7.21 to 8.40
Out of a rating of 10, the true rating can lie in the following % range:  11.91594827586207
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  34
Confidence Interval: 7.19 to 8.28
Out of a rating of 10, the true rating can lie in the following % range:  10.857620320855625
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
Number of Ratings:  39
Confidence Interval: 7.14 to 8.18
Out of a rating of 10, the true rating can lie in the following % range:  10.368131868131858
---------------------------------------------------------------------------------------------
```

Having 19 or more ratings brings down the range of the ratings confidence interval within a narrow range of less than 15% difference. This shows us that a book having more than 19 votes in this dataset has a rating that can be reliaably used to make predictions. We will filter out to use books that have greater than 19 votes.

Also, the book ratings are seen to remain comparitively high as compared to the scale of 1-10. Lets see how many books do we truly have ratings less than 6.

In [707]:
```
#check value counts for less than 6 ratings
user_rating_books[user_rating_books['Book-Rating'] <6]['Book-Rating'].value_counts()
```

Out[707]:
```
5.0    23887
4.0     4324
3.0     2876
2.0     1322
1.0      754
Name: Book-Rating, dtype: int64
```

In [708]:
```
#check total length
len(user_rating_books)
```

Out[708]:
```
653360
```

We can see that books with ratings of 5 has aa good number but there arae very few books with ratings less than 5. Therefore. we can drop these records and only keep the ones greater than or equal to 5.

```
In [709]:   #filter for ratings greater thaaan 5 and the NaN values
            user_rating_books = user_rating_books[(user_rating_books['Book-Rating'] >= 5) |
                                                  (user_rating_books['Book-Rating'].isna())]
```

After we are done filtering for aall other data points, we will filter out our dataset to keep only the books that have greater than 19 reviews.

```
In [710]:   #check info of the new dataset
            user_rating_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 644084 entries, 0 to 653359
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   User-ID              644084 non-null  int64  
 1   ISBN                 644084 non-null  object 
 2   Book-Rating          222590 non-null  float64
 3   Book-Title           644084 non-null  object 
 4   Book-Author          644084 non-null  object 
 5   Year-Of-Publication  644084 non-null  int64  
 6   Publisher            644084 non-null  object 
 7   Age                  644084 non-null  float64
 8   Country              644084 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 49.1+ MB
```

```
In [711]:   #check for duplicates
            user_rating_books.duplicated().sum()

Out[711]: 0
```

With all of this done, we can move onto Exploratory Data Analysis to understand more about the data we are working with. Moreover, we can use matrix factorization like Alternating Least Squares (ALS) to fill up missing values in a dataset for ratings through matrix completion. The idea is to factorize the user-item rating matrix using ALS and then use the learned factors to estimate the missing values. We can do this with the modelling part.

EDA

```
In [712]:   #check info of the new dataset
            user_rating_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 644084 entries, 0 to 653359
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   User-ID              644084 non-null  int64  
 1   ISBN                 644084 non-null  object 
 2   Book-Rating          222590 non-null  float64
 3   Book-Title           644084 non-null  object 
 4   Book-Author          644084 non-null  object 
 5   Year-Of-Publication  644084 non-null  int64  
 6   Publisher            644084 non-null  object 
 7   Age                  644084 non-null  float64
 8   Country              644084 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 49.1+ MB
```

Lets start with looking at the Age data to see what the distribution of the Ages are.

```
In [713]:  #plot Age against number of reviews
           px.histogram(user_rating_books['Age'],title="User's Age Distribution")
```

The ages data is right skewed with a long tail. Most of the customers are in between 25 years old and 40 years old which is the ideal niche to focus on. These age groups are people who have money, are more inclined to use ecommercce and online services, and generally have the ability to read a lot more.

Lets check Top 10 countries that we have the data for.

In [716]:
```
#group the data
country_data = user_rating_books.groupby("Country").size().sort_values(ascending= False).head(10)

#separate index and values for names
names = country_data.index
values = country_data.values

#plot pie chart
px.pie(values = values, names = names,title = "Countries Total Users %")
```

In [717]:
```
#check Country value percentages
user_rating_books["Country"].value_counts(normalize=True)
```

Out[717]:
```
usa          0.780740
canada       0.093825
germany      0.027236
spain        0.018945
australia    0.018676
               ...
angola       0.000002
ethiopia     0.000002
algeria      0.000002
zambia       0.000002
uganda       0.000002
Name: Country, Length: 103, dtype: float64
```

Looks like almost 84% of the data is from US customers. This is a highly biased dataset leaning towards USA. Rather than trying to train the model for other countries, it would be more useful to have more reliable recommendation system for a single country and find more data for the other countries before making aaa recommendation system for their users.

In [718]:
```
#set the countries to filter for
countries = ['usa']

#filter for the countries to keep
user_rating_books = user_rating_books[user_rating_books['Country'].isin(countries)]

#preview the dataset
user_rating_books.head()
```

Out[718]:

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Age	Country
1	638	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books	20.0	usa
2	638	0316666343	10.0	The Lovely Bones: A Novel	Alice Sebold	2002	Little, Brown	20.0	usa
3	638	0375400699	10.0	Love in the Time of Cholera (Everyman's Librar...	GABRIEL GARCIA MARQUEZ	1997	Everyman's Library	20.0	usa
4	638	0385504209	10.0	The Da Vinci Code	Dan Brown	2003	Doubleday	20.0	usa
5	638	0679746048	7.0	Girl, Interrupted	SUSANNA KAYSEN	1994	Vintage	20.0	usa

In [719]:
```
#check country values
user_rating_books['Country'].value_counts()
```

Out[719]:
```
usa    502862
Name: Country, dtype: int64
```

In [720]:
```
#check info
user_rating_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 502862 entries, 1 to 653358
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   User-ID              502862 non-null  int64
 1   ISBN                 502862 non-null  object
 2   Book-Rating          164882 non-null  float64
 3   Book-Title           502862 non-null  object
 4   Book-Author          502862 non-null  object
 5   Year-Of-Publication  502862 non-null  int64
 6   Publisher            502862 non-null  object
 7   Age                  502862 non-null  float64
 8   Country              502862 non-null  object
dtypes: float64(2), int64(2), object(5)
memory usage: 38.4+ MB
```

Lets look at the book ratings to understand their distribution better.

In [721]:
```
#plot book ratings
px.histogram(user_rating_books['Book-Rating'],title="Distribution of Book Ratings")
```

The ratings are left skewed with most of them lying between 7 and 10. This shows that most people have rated books on a higher scale with 8 with the most popular one.

Next we will check the average ratings of title.

In [722]:
```
#group by title and find the avergaae book ratings
avg_rating_book = user_rating_books.groupby("ISBN").mean()["Book-Rating"].sort_values(ascending=False)
```

In [723]:
```
#plot average book ratings
px.histogram(avg_rating_book,title="Distribution of Average Book Ratings", nbins=7)
```

In [724]:
```
avg_rating_book.mean()
```

Out[724]:
```
7.925450320566293
```

The average book ratings are almost normally distributed which shows that even though people generally were seen giving higher ratings, on an average, for every title, it has balanced out.

In [725]:
```
#check info
user_rating_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 502862 entries, 1 to 653358
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   User-ID              502862 non-null  int64  
 1   ISBN                 502862 non-null  object 
 2   Book-Rating          164882 non-null  float64
 3   Book-Title           502862 non-null  object 
 4   Book-Author          502862 non-null  object 
 5   Year-Of-Publication  502862 non-null  int64  
 6   Publisher            502862 non-null  object 
 7   Age                  502862 non-null  float64
 8   Country              502862 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 38.4+ MB
```

```
In [726]:   import plotly.subplots as sp

            # Create subplots
            fig = sp.make_subplots(rows=3, cols=2)

            # Add histograms to subplots
            fig.add_trace(px.histogram(user_rating_books[user_rating_books['Book-Rating'] == 5]['Age']).data[0], row=1, col=1)
            fig.add_trace(px.histogram(user_rating_books[user_rating_books['Book-Rating'] == 6]['Age']).data[0], row=1, col=2)
            fig.add_trace(px.histogram(user_rating_books[user_rating_books['Book-Rating'] == 7]['Age']).data[0], row=2, col=1)
            fig.add_trace(px.histogram(user_rating_books[user_rating_books['Book-Rating'] == 8]['Age']).data[0], row=2, col=2)
            fig.add_trace(px.histogram(user_rating_books[user_rating_books['Book-Rating'] == 9]['Age']).data[0], row=3, col=1)
            fig.add_trace(px.histogram(user_rating_books[user_rating_books['Book-Rating'] == 10]['Age']).data[0], row=3, col=2)

            # Update layout with chart titles
            fig.update_layout(
                showlegend=False,
                title_text="Histograms of Ages Distribution for Different Book Ratings",
                title_font_size=24,
                title_x=0.5,
                title_y=0.95,
                # Set individual subplot titles
                annotations=[
                    dict(
                        text="Book Rating 5",
                        x=0.17,
                        y=0.83,
                        font=dict(size=14),
                        showarrow=False,
                        xref="paper",
                        yref="paper",
                        align="left"
                    ),
                    dict(
                        text="Book Rating 6",
                        x=0.83,
                        y=0.83,
                        font=dict(size=14),
                        showarrow=False,
                        xref="paper",
                        yref="paper",
                        align="left"
                    ),
                    dict(
                        text="Book Rating 7",
                        x=0.17,
                        y=0.47,
                        font=dict(size=14),
                        showarrow=False,
                        xref="paper",
                        yref="paper",
                        align="left"
                    ),
                    dict(
                        text="Book Rating 8",
                        x=0.83,
                        y=0.47,
                        font=dict(size=14),
                        showarrow=False,
                        xref="paper",
                        yref="paper",
                        align="left"
                    ),
                    dict(
                        text="Book Rating 9",
                        x=0.17,
                        y=0.12,
                        font=dict(size=14),
                        showarrow=False,
                        xref="paper",
                        yref="paper",
                        align="left"
```

```
        ),
        dict(
            text="Book Rating 10",
            x=0.83,
            y=0.12,
            font=dict(size=14),
            showarrow=False,
            xref="paper",
            yref="paper",
            align="left"
        )
    ]
)

# Update layout
fig.update_layout(showlegend=False)

# Show the figure
fig.show()
```

For every rating we have a similaar distribution of age groups we have records for. As we saw earlieer, most of our data is coming from 25-40 year olds which is the main niche that we can rely on as the customer base.

Nonetheless, lets explore how do the different age groups rate. There is a possibility that some aage groups are stricter raters than other or vice versa. Lets explore this.

```
In [727]:  # Asigned temp data
            data = user_rating_books

            # Create age categories using a temporary column
            data['Age_Category'] = pd.cut(data['Age'], bins=range(16, 61, 4), right=False).astype(str)

            # Calculate average ratings for each age category
            avg_ratings = data.groupby('Age_Category')['Book-Rating'].mean().reset_index()

            # Create the plot using Plotly Express
            fig = px.bar(avg_ratings, x='Age_Category', y='Book-Rating', labels={'Age_Category': 'Age Group', 'Rating': 'Average Rating'}, title = "Average Rating VS Age Group for ")

            # Customize the plot layout
            fig.update_layout(title='Average Ratings by Age Group', xaxis_title='Age Group', yaxis_title='Average Rating')

            # Show the plot
            fig.show()
```

All of the age groups rate very close to each other, remaining mostly close to 8. This is a good sign that there are no drastic skewness with any age groups.

Lets check to see how do different age groups differ in their choice of books relative to when they were published.

In [728]:
```python
import plotly.express as px

# Assigned temp data
data = user_rating_books

# Group the data by age groups and count the number of books published in a certain year
grouped_data = data.groupby(['Age_Category', 'Year-Of-Publication']).size().reset_index(name='Number of Books')

# Create a bar chart
fig = px.bar(grouped_data, x='Age_Category', y='Number of Books', color='Year-Of-Publication',
             title='Number of Books Published by Age Group and Year')

# Set axes labels
fig.update_layout(
    xaxis_title='Age Group',
    yaxis_title='Number of Books Published'
)

# Show the figure
fig.show()
```

We don't have anomalies with any age groups associations to years of publications.

Next lets check what the average age of readers is for the top 20 highest rated authors.

```
In [729]:     #filter for top 20
              data = user_rating_books.groupby(['Book-Author']).mean().sort_values(['Book-Rating'],ascending=False).head(20)

              #plot
              px.bar(data['Age'], title = "Average Age VS Top 20 Rated Book Authors")
```

Mostly the ages are between 30-40. This is making the customer persona clearer. The biggest customer base will be between 25-40 year old.

Now lets check which authors have written the most books. We can check if there is some trend to the number of books written and higher ratings.

In [730]:
```
#filter for top 20
data = user_rating_books.groupby(['Book-Author']).mean().sort_values(['Book-Rating'],ascending=False).head(20)

#plot
px.bar(data['Book-Rating'], title = "Average Rating for Top 20 Rated Book Authors")
```

```
In [731]:   #groupby aand filter for authors
            Author = user_rating_books.groupby("Book-Author").size().sort_values(ascending = False).head(20)
            Author = pd.DataFrame({"Book-Author":Author.index,"No of Books Written":Author.values})

            #plot
            px.bar(data_frame = Author, y="Book-Author", x="No of Books Written" , title = "Top 20 Most books written by Author")
```

We don't see any definite correlation between the most books written and the average ratings. Mostly, if an author has written more books, doesn't necessarily mean that theyir books have higher average ratigs. Nonetheless, we do see some of the higher rated authors in the above list also.

Next, lets investigate publishers in the same way. We will first check the average rating per publisher and the number of books published.

In [732]:
```
#groupby and filter for top 20
publisher = user_rating_books.groupby("Publisher").mean().sort_values(['Book-Rating'],ascending=False).head(20)

#plot
px.bar(publisher['Book-Rating'], title = "Average Rating for Top 20 Rated Book Publishers")
```

```
In [733]:   #groupby and filter for top 20
            publisher = user_rating_books.groupby("Publisher").size().sort_values(ascending = False).head(20)
            publisher = pd.DataFrame({"Publisher":publisher.index,"No of Books Published":publisher.values})

            #plot
            px.bar(data_frame = publisher, y="Publisher", x="No of Books Published" , title = "Top 20 Most books Published by Publisher")
```

Similar to authors, most of the publishers who have published the most books do not have the best average ratings. This can be due to a number of reasons but one thing is cleaar thta quantity does not translate out to quality.

Now we will go ahead and filter out the books that have less than 19 reviews.

```
In [734]:   #groupby ISBN and calculate numebr of reviews
            num_reviews = user_rating_books.groupby('ISBN').count()

            #extract indices of books
            indices = num_reviews[num_reviews['Book-Rating']>19].index

            #filter to keep the books with the correct indices
            user_rating_books = user_rating_books[(user_rating_books['ISBN'].isin(indices))]

            #preview
            user_rating_books.head()
```

Out[734]:

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Age	Country	Age_Category
1	638	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books	20.0	usa	[20, 24)
2	638	0316666343	10.0	The Lovely Bones: A Novel	Alice Sebold	2002	Little, Brown	20.0	usa	[20, 24)
4	638	0385504209	10.0	The Da Vinci Code	Dan Brown	2003	Doubleday	20.0	usa	[20, 24)
5	638	0679746048	7.0	Girl, Interrupted	SUSANNA KAYSEN	1994	Vintage	20.0	usa	[20, 24)
7	638	0670892963	7.0	Bridget Jones : The Edge of Reason	Helen Fielding	2000	Viking Books	20.0	usa	[20, 24)

We will filter out the missing values into te recommendation set that we will use to predict book ratings and recommend the books that useers haven't read.

In [735]:
```
#filter for the datasets with missing ratings
rec_set = user_rating_books[user_rating_books['Book-Rating'].isna()]

#preview data
rec_set.head()
```

Out[735]:

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Age	Country	Age_Category
1	638	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books	20.0	usa	[20, 24)
12	638	0743206045	NaN	Daddy's Little Girl	Mary Higgins Clark	2002	Simon &amp; Schuster	20.0	usa	[20, 24)
69	638	080411868X	NaN	Welcome to the World, Baby Girl!	Fannie Flagg	1999	Ballantine Books	20.0	usa	[20, 24)
74	3363	0446520802	NaN	The Notebook	Nicholas Sparks	1996	Warner Books	29.0	usa	[28, 32)
75	3363	002542730X	NaN	Politically Correct Bedtime Stories: Modern Ta...	James Finn Garner	1994	John Wiley &amp; Sons Inc	29.0	usa	[28, 32)

In [736]:
```
#check info
rec_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40644 entries, 1 to 636960
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   User-ID              40644 non-null  int64
 1   ISBN                 40644 non-null  object
 2   Book-Rating          0 non-null      float64
 3   Book-Title           40644 non-null  object
 4   Book-Author          40644 non-null  object
 5   Year-Of-Publication  40644 non-null  int64
 6   Publisher            40644 non-null  object
 7   Age                  40644 non-null  float64
 8   Country              40644 non-null  object
 9   Age_Category         40644 non-null  object
dtypes: float64(2), int64(2), object(6)
memory usage: 3.4+ MB
```

In [737]:
```
#drop records with missing ratings
user_rating_books = user_rating_books[~user_rating_books['Book-Rating'].isna()]

#preview dataset
user_rating_books.head()
```

Out[737]:

	User-ID	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Age	Country	Age_Category
2	638	0316666343	10.0	The Lovely Bones: A Novel	Alice Sebold	2002	Little, Brown	20.0	usa	[20, 24)
4	638	0385504209	10.0	The Da Vinci Code	Dan Brown	2003	Doubleday	20.0	usa	[20, 24)
5	638	0679746048	7.0	Girl, Interrupted	SUSANNA KAYSEN	1994	Vintage	20.0	usa	[20, 24)
7	638	0670892963	7.0	Bridget Jones : The Edge of Reason	Helen Fielding	2000	Viking Books	20.0	usa	[20, 24)
8	638	0316776963	10.0	Me Talk Pretty One Day	David Sedaris	2001	Back Bay Books	20.0	usa	[20, 24)

In [738]:
```
#check info
user_rating_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27476 entries, 2 to 636959
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   User-ID              27476 non-null  int64  
 1   ISBN                 27476 non-null  object 
 2   Book-Rating          27476 non-null  float64
 3   Book-Title           27476 non-null  object 
 4   Book-Author          27476 non-null  object 
 5   Year-Of-Publication  27476 non-null  int64  
 6   Publisher            27476 non-null  object 
 7   Age                  27476 non-null  float64
 8   Country              27476 non-null  object 
 9   Age_Category         27476 non-null  object 
dtypes: float64(2), int64(2), object(6)
memory usage: 2.3+ MB
```

A big chunk of our data has missing ratings as compared to the available values for us. This iis not ideal sicne we would ideally want more data to train our model on. Nonetheless, we will see how our models perform with the available data

In [739]:
```
# #make a temporary copy
# temp_data = user_rating_books.copy()

# #drop rowss with missing values
# temp_data.dropna(inplace=True)

# #groupby ISSBN and find mean
# temp_data = temp_data.groupby("ISBN").mean()

# #round the data
# temp_data['Book-Rating'] = temp_data['Book-Rating'].round()

# #preview the data
# temp_data['Book-Rating']
```

In [740]:
```
# for index in temp_data.index:
#     value = temp_data['Book-Rating'].loc[index]

#     user_rating_books['Book-Rating'] = user_rating_books.groupby('ISBN')['Book-Rating'].fillna(value)
```

Data Limitations

Before we get into Modelling, there are a few limitiations of the Data that we should be aware of.

```
1. There is very few demogrphic information for the users available to develop a solid user persona to base recommendations on.

2. There is no information available for book genres that can be used for devveloping more informaative vectors about books.

3. The data is only focused for the US population. If the customer base needs to be expaanded to other ountries, than more data from the other countries will have to b
e gathered.

4. The missing rating records are almost double in number as compared to the records with non-missing ratings. This is not ideal sice we would like to have a dataset w
hich is larger than the missing values dataset . Wiith more missing values, there will definiteely be areas or aspects that our model isn't trained to handle.

5. The general bias of marketing and promotion is not considered in this dataset. Through Social Sciences, it is known

6. There are very few low ratings available while most ratings arae on the higher side.

7. Only ratings on a scale of 1-10 are available. There are no reviews available that would also provide a qulittaive analysis of what specifically did the user like.
```

Model Preprocessing

In [741]:
```
#import relevant libraries
from surprise import Dataset, Reader

from surprise.model_selection import train_test_split
from sklearn import model_selection

from imblearn.under_sampling import RandomUnderSampler

from sklearn.model_selection import train_test_split,RandomizedSearchCV,GridSearchCV
from tqdm import tqdm
```

Lets start off with separating the data. We will use the ISBN to identify our books. We can later use the predicted ISBN to extract the Book title name. Since the ISBN is the unique identifier, thata will be the best feature to use.

In [742]:
```
#separate the columns to use
ml_ratings = user_rating_books[['User-ID',
                                'ISBN',
                                'Book-Author',
                                'Year-Of-Publication',
                                'Publisher',
                                'Age',
                                'Country',
                                'Age_Category',
                                'Book-Rating']]

#check for duplicates
ml_ratings.duplicated().sum()
```

Out[742]:
```
0
```

In [743]:
```
#check info
ml_ratings.info()
```

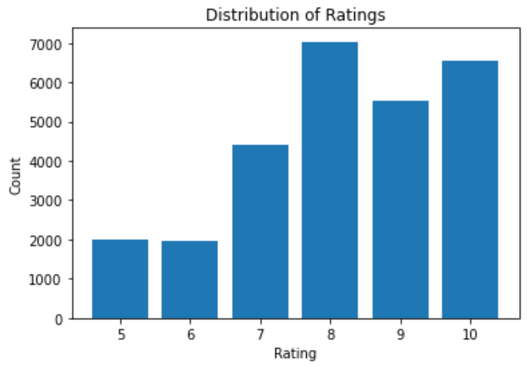
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27476 entries, 2 to 636959
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   User-ID              27476 non-null  int64  
 1   ISBN                 27476 non-null  object 
 2   Book-Author          27476 non-null  object 
 3   Year-Of-Publication  27476 non-null  int64  
 4   Publisher            27476 non-null  object 
 5   Age                  27476 non-null  float64
 6   Country              27476 non-null  object 
 7   Age_Category         27476 non-null  object 
 8   Book-Rating          27476 non-null  float64
dtypes: float64(2), int64(2), object(5)
memory usage: 2.1+ MB
```

Recall that the missing values in Book-Rating need to bee handled through matrix factorization. Lets see again what the distribution of the ratings looks like currently.

In [744]:
```
import matplotlib.pyplot as plt

# Count the occurrences of each rating value
rating_counts = ml_ratings['Book-Rating'].value_counts()

# Plot the distribution of ratings
plt.bar(rating_counts.index, rating_counts.values)
plt.xlabel('Rating')
plt.ylabel('Count')
plt.title('Distribution of Ratings')
plt.show()
```



Next, we will split our dataset into three sets; Train set, Validation Set, Test Set. The traain set will be used to traain the models and the validation set will be cossequently used to test these models. After the best performing model haas been selected, we will use the test set to run the metrics and do a final evaluation.

In [745]:
```
# Import the necessary libraries
from surprise import Reader, Dataset
from sklearn.model_selection import train_test_split

#split into train/valid sets and test sets
train_valid_set, test = model_selection.train_test_split(ml_ratings,
                                                         test_size=0.2,
                                                         random_state = 23)

# Create a Reader object with the rating scale
reader = Reader(rating_scale=(5, 10))

# Load the data from the train_valid_set DataFrame into a Surprise Dataset
data = Dataset.load_from_df(train_valid_set[['User-ID', 'ISBN', 'Book-Rating']], reader)

# Convert Surprise Dataset to a pandas DataFrame
df = pd.DataFrame(data.raw_ratings, columns=['User-ID', 'ISBN', 'Book-Rating', 'timestamp'])

# Split the data into train_valid and test sets
train_df, valid_df = train_test_split(df, test_size=0.25, random_state=42)

# Convert the train, valid and test DataFrames back to Surprise Dataset
trainset = Dataset.load_from_df(train_df[['User-ID', 'ISBN', 'Book-Rating']], reader)
validset = Dataset.load_from_df(valid_df[['User-ID', 'ISBN', 'Book-Rating']], reader)
testset = Dataset.load_from_df(test[['User-ID', 'ISBN', 'Book-Rating']], reader)

#Build the Trainset object
train_set = trainset.build_full_trainset()

# Build the validset object
valid_set = validset.build_full_trainset().build_testset()

# Build the Testset object
test_set = testset.build_full_trainset().build_testset()
```

In [746]:
```
#preview train set
train_set
```

Out[746]:
```
<surprise.trainset.Trainset at 0x7f92bd83ed30>
```

With traain, validation and test sets available, lets move on to modelling our data.

Modelling and Evaluation

In [747]:
```
#import libraries
from surprise import SVD , NMF , KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline
from surprise.model_selection import RandomizedSearchCV
from surprise import accuracy
```

We will be using the following methods for modelling:

```
1. KNN

2. SVD

3. NMF
```

KNNWithMeans, KNNWithZScore, and KNNBaseline are collaborative filtering algorithms based on the K-Nearest Neighbors approach. They leverage the similarities between users or items to make predictions. KNNWithMeans considers the mean ratings, KNNWithZScore incorporates normalization of ratings, and KNNBaseline integrates a baseline rating estimation. These algorithms are effective when the data exhibits localized similarities and can provide accurate recommendations based on nearest neighbors.

SVD (Singular Value Decomposition) is a popular matrix factorization technique widely used in recommendation systems. It can effectively capture latent factors in the data by decomposing the user-item rating matrix into lower-dimensional matrices. SVD performs well in reducing noise and capturing underlying patterns, making it a reliable choice for recommendation modeling.

NMF (Non-Negative Matrix Factorization) is another matrix factorization method that imposes non-negativity constraints on the factor matrices. It is particularly useful when dealing with non-negative data, such as ratings or counts. NMF can provide meaningful latent factors and help in generating accurate recommendations.

To handle missing values in the book ratings, the **ALS (Alternating Least Squares) algorithm can be utilized in conjunction with KNN modeling**. ALS is a matrix factorization method that iteratively fills in the missing values by estimating latent factors. It can be applied before the train-test split, ensuring that the imputation process is based solely on the training data.

Alternatively, **SVD and NMF can handle missing values on their own**. These matrix factorization methods inherently impute missing values as part of the factorization process. By considering the available ratings and latent factors, they can estimate missing values and generate recommendations accordingly.

The evaluation metrics that we will use to evluate the performance of these models are listed below with theeir pros and cons.

FCP (Fraction of Concordant Pairs) is a commonly used evaluation metric for collaborative filtering algorithms. It measures the fraction of pairs of items or users where the predicted ordering of preferences is consistent with the observed ordering. FCP is advantageous because it is less sensitive to overall rating values and focuses more on the relative rankings. However, a limitation of FCP is that it does not consider the magnitude of the predicted ratings.

RMSE (Root Mean Squared Error) is a popular evaluation metric that calculates the square root of the average of squared differences between predicted and actual ratings. RMSE penalizes larger prediction errors more heavily and provides a good overall measure of accuracy. However, RMSE is sensitive to outliers and can be influenced by extreme ratings, which might not be desirable in certain cases.

MSE (Mean Squared Error) is similar to RMSE, but it does not take the square root. Instead, it calculates the average of squared differences between predicted and actual ratings. MSE is useful for comparing models and identifying the best-performing one. However, like RMSE, it is sensitive to outliers and may not provide an intuitive understanding of the prediction errors in terms of the original rating scale.

MAE (Mean Absolute Error) is an evaluation metric that calculates the average of absolute differences between predicted and actual ratings. MAE provides a more interpretable measure of prediction errors in terms of the original rating scale. It is less sensitive to outliers compared to RMSE and MSE. However, MAE treats all prediction errors equally, which means it does not differentiate between small and large errors.

In summary, FCP is a suitable metric for collaborative filtering algorithms as it focuses on relative rankings. RMSE and MSE provide overall accuracy measures but are sensitive to outliers. MAE is more interpretable and less sensitive to outliers, but it treats all errors equally. The choice of evaluation metric depends on the specific requirements and priorities of the recommendation system.

In [748]:
```
final_models = {
    "Name":[],
    "Model":[],
    "FCP":[],
    "RMSE":[],
    "MSE":[],
    "MAE":[],
}
```

Baseline Model

We will start with a baseline model. We will KNN Basic for the baselinne model. We won't be doing aany paraameter optimizations. It can handle both numerical and categorical features, accommodating a diverse range of book characteristics. The algorithm does not assume a specific data distribution, allowing it to adapt to changing user tastes over time without retraining. However, KNN may struggle with high-dimensional data and can be computationally expensive for large datasets. While it serves as a solid starting point, more advanced algorithms can be employed to improve accuracy and scalability.

In [749]:
```
algo = KNNBasic()

# Train the model on the training set
algo.fit(train_set)

# Evaluate the model on the test set
predictions = algo.test(valid_set)

# Calculate FCP, RMSE, MSE & MAE
rmse = accuracy.rmse(predictions)
mae  = accuracy.mae(predictions)
mse  = accuracy.mse(predictions)
fcp  = accuracy.fcp(predictions)

final_models["Name"].append("KNNBasic")
final_models["Model"].append(algo)
final_models["FCP"].append(fcp)
final_models["RMSE"].append(rmse)
final_models["MSE"].append(mse)
final_models["MAE"].append(mae)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.6180
MAE:  1.2702
MSE: 2.6179
FCP:  0.5885
```

RMSE (Root Mean Square Error): RMSE measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. In this case, the RMSE value of 1.6180 indicates that, on average, the model's predictions for book ratings deviate from the actual ratings by approximately 1.6180 on the 5-10 scale. Lower RMSE values indicate better performance, as it means the model's predictions are closer to the actual ratings.

MAE (Mean Absolute Error): MAE also measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. The MAE value of 1.2702 suggests that, on average, the model's predictions deviate from the true ratings by approximately 1.2702 on the 5-10 scale. Similarly to RMSE, lower MAE values indicate better performance.

MSE (Mean Squared Error): MSE is another measure of the prediction accuracy, but it focuses on the squared differences between the model's predicted ratings and the actual ratings. The MSE value of 2.6179 represents the average of these squared differences. Like RMSE and MAE, lower MSE values indicate better performance.

FCP (Fraction of Concordant Pairs): FCP is a different type of evaluation metric often used in recommendation systems. It measures the proportion of pairs of items where the model correctly predicts the relative order of ratings. In this case, the FCP value of 0.5885 suggests that approximately 58.85% of the pairs are correctly ranked by the model. Higher FCP values indicate better performance, as it means the model is more accurate in predicting the relative rankings of book ratings.

Overall, based on these evaluation metrics, it seems that the book recommendation model is performing reasonably well. The RMSE, MAE, and MSE values indicate that, on average, the model's predictions deviate from the true ratings by around 1.5 to 2 units on the 5-10 scale. The FCP value of 0.5885 suggests that the model is able to correctly rank the relative order of book ratings in approximately 58.85% of cases.

We will try other KNN models to see if we can improve the performance.

KNN Baseline

The KNN Baseline model is a preferable choice over the KNN Basic model due to its ability to enhance performance in several ways. Unlike the KNN Basic model, which relies solely on the similarity between instances to make predictions, the KNN Baseline model incorporates additional information by considering the baselines of users and items. By estimating the overall rating tendencies of users and items, the KNN Baseline model effectively reduces the impact of outliers and accounts for the inherent biases present in the dataset. This incorporation of baseline estimates allows for a more accurate and robust prediction process, leading to improved performance in recommendation systems. Consequently, the KNN Baseline model is considered a valuable upgrade to the KNN Basic model as it leverages additional information to deliver enhanced predictions and better overall performance.

We will also optimize the model's parameters to fine tune performance.

The **'k' parameter** determines the number of neighbors considered when making predictions, and trying different values such as 10, 20, 30, and 40 allows us to find the optimal value that balances accuracy and computational efficiency.

The **'sim_options' parameter** specifies the similarity metric to be used, with options like 'msd' (mean squared difference), 'cosine', and 'pearson'. By testing these different metrics, we can identify the one that best captures the relationships between users and items in the dataset.

The **'min_support' parameter** determines the minimum number of common items required for two users to be considered neighbors. Evaluating this parameter with values like 3 and 5 helps us understand the impact of item overlap on the quality of recommendations.

The **'user_based' parameter** being set to 'True' indicates that the model uses a user-based collaborative filtering approach, where recommendations are based on similarities between users. This setting can be compared with item-based collaborative filtering to determine which approach yields better results.

Finally, setting **"verbose"** to 'False' ensures that the model does not produce excessive output during training and evaluation, keeping the process more streamlined.

By searching for the optimal combination of these parameters, we can fine-tune the KNN Baseline model to achieve improved performance and generate more accurate recommendations.

In [750]:
```python
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'k': [10, 20, 30, 40],
    'sim_options' : {
        'name'       : ['msd','cosine','pearson'],
        'min_support': [3,5],
        'user_based' : [True]
        },
    "verbose" : [False]
    }

# Perform grid search with cross-validation
grid = RandomizedSearchCV(KNNBaseline, param_distributions=param_grid, measures=['rmse'], cv=5)

# Fit the grid search object to the data
grid.fit(data)

# Get the best RMSE score and parameters
print("Best RMSE score:", grid.best_score['rmse'])
print("Best parameters:", grid.best_params['rmse'])

# Train the model on the full training set with the best parameters
algo = grid.best_estimator['rmse']
algo.fit(train_set)

# Evaluate the best model on the test set
predictions = algo.test(valid_set)

# Calculate FCP, RMSE, MSE & MAE
rmse = accuracy.rmse(predictions)
mae  = accuracy.mae(predictions)
mse  = accuracy.mse(predictions)
fcp  = accuracy.fcp(predictions)


final_models["Name"].append("KNNBaseline")
final_models["Model"].append(algo)
final_models["FCP"].append(fcp)
final_models["RMSE"].append(rmse)
final_models["MSE"].append(mse)
final_models["MAE"].append(mae)
```

```
Best RMSE score: 1.3878838636093893
Best parameters: {'k': 20, 'sim_options': {'name': 'pearson', 'min_support': 5, 'user_based': True}, 'verbose': False}
RMSE: 1.4017
MAE:  1.1309
MSE: 1.9648
FCP:  0.5516
```

RMSE (Root Mean Square Error): RMSE measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. In this case, the RMSE value of 1.4017 indicates that, on average, the model's predictions for book ratings deviate from the actual ratings by approximately 1.4017 on the 5-10 scale. Lower RMSE values indicate better performance, as it means the model's predictions are closer to the actual ratings.

MAE (Mean Absolute Error): MAE also measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. The MAE value of 1.1309 suggests that, on average, the model's predictions deviate from the true ratings by approximately 1.1309 on the 5-10 scale. Similarly to RMSE, lower MAE values indicate better performance.

MSE (Mean Squared Error): MSE is another measure of the prediction accuracy, but it focuses on the squared differences between the model's predicted ratings and the actual ratings. The MSE value of 1.9648 represents the average of these squared differences. Like RMSE and MAE, lower MSE values indicate better performance.

FCP (Fraction of Concordant Pairs): FCP is a different type of evaluation metric often used in recommendation systems. It measures the proportion of pairs of items where the model correctly predicts the relative order of ratings. In this case, the FCP value of 0.5516 suggests that approximately 55.16% of the pairs are correctly ranked by the model. Higher FCP values indicate better performance, as it means the model is more accurate in predicting the relative rankings of book ratings.

Based on these evaluation metrics, it appears that the book recommendation model is performing fairly well. The RMSE, MAE, and MSE values indicate that, on average, the model's predictions deviate from the true ratings by around 1.4 to 2 units on the 5-10 scale. The FCP value of 0.5516 suggests that the model is able to correctly rank the relative order of book ratings in approximately 55.16% of cases.

KNN with Means

In [751]:
```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'k': [10, 20, 30, 40],
    'sim_options' : {
        'name'        : ['msd','cosine','pearson'],
        'min_support': [3,5],
        'user_based' : [True]
        },
    "verbose" : [False]
    }

# Perform grid search with cross-validation
grid = RandomizedSearchCV(KNNWithMeans, param_distributions=param_grid, measures=['rmse'], cv=5)

# Fit the grid search object to the data
grid.fit(data)

# Get the best RMSE score and parameters
print("Best RMSE score:", grid.best_score['rmse'])
print("Best parameters:", grid.best_params['rmse'])

# Train the model on the full training set with the best parameters
algo = grid.best_estimator['rmse']
algo.fit(train_set)

# Evaluate the best model on the test set
predictions = algo.test(valid_set)

# Calculate FCP, RMSE, MSE & MAE
rmse = accuracy.rmse(predictions)
mae  = accuracy.mae(predictions)
mse  = accuracy.mse(predictions)
fcp  = accuracy.fcp(predictions)


final_models["Name"].append("KNNWithMeans")
final_models["Model"].append(algo)
final_models["FCP"].append(fcp)
final_models["RMSE"].append(rmse)
final_models["MSE"].append(mse)
final_models["MAE"].append(mae)
```

```
Best RMSE score: 1.4875738130036062
Best parameters: {'k': 30, 'sim_options': {'name': 'pearson', 'min_support': 5, 'user_based': True}, 'verbose': False}
RMSE: 1.5040
MAE:  1.1604
MSE: 2.2619
FCP:  0.7114
```

RMSE (Root Mean Square Error): RMSE measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. In this case, the RMSE value of 1.5040 indicates that, on average, the model's predictions for book ratings deviate from the actual ratings by approximately 1.5040 on the 5-10 scale. Lower RMSE values indicate better performance, as it means the model's predictions are closer to the actual ratings.

MAE (Mean Absolute Error): MAE also measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. The MAE value of 1.1604 suggests that, on average, the model's predictions deviate from the true ratings by approximately 1.1604 on the 5-10 scale. Similarly to RMSE, lower MAE values indicate better performance.

MSE (Mean Squared Error): MSE is another measure of the prediction accuracy, but it focuses on the squared differences between the model's predicted ratings and the actual ratings. The MSE value of 2.2619 represents the average of these squared differences. Like RMSE and MAE, lower MSE values indicate better performance.

FCP (Fraction of Concordant Pairs): FCP is a different type of evaluation metric often used in recommendation systems. It measures the proportion of pairs of items where the model correctly predicts the relative order of ratings. In this case, the FCP value of 0.7114 suggests that approximately 71.14% of the pairs are correctly ranked by the model. Higher FCP values indicate better performance, as it means the model is more accurate in predicting the relative rankings of book ratings.

Based on these evaluation metrics, it appears that the book recommendation model is performing reasonably well. The RMSE, MAE, and MSE values indicate that, on average, the model's predictions deviate from the true ratings by around 1.4 to 2.3 units on the 5-10 scale. The FCP value of 0.7114 suggests that the model is able to correctly rank the relative order of book ratings in approximately 71.14% of cases.

We can try and see if we can improve performance throough KNN with Z Score.

KNN with Z Score

KNN with Z score is a superior model compared to KNN Baseline as it brings additional advantages to enhance performance. While KNN Baseline incorporates baseline estimates to mitigate biases and outliers, KNN with Z score further improves the model by standardizing the ratings across the dataset. By transforming the ratings into z-scores, which represent the number of standard deviations an individual rating is from the mean, KNN with Z score eliminates the influence of varying rating scales and normalizes the data. This normalization enables better comparison and similarity calculations between instances, resulting in more accurate predictions. Moreover, by taking into account the entire rating distribution rather than just the mean and biases, KNN with Z score offers a more comprehensive and nuanced understanding of the data, leading to improved performance in recommendation systems. Therefore, KNN with Z score is a favorable choice over KNN Baseline, as it leverages standardized ratings to enhance accuracy, comparability, and the overall performance of the model.

The parameters optimized are listed below.

The **'k' parameter** determines the number of neighbors considered when making predictions, and trying values like 10, 20, 30, and 40 allows us to find the optimal balance between accuracy and computational efficiency.

The **'sim_options' parameter** specifies the similarity metric to be used, with options such as 'msd' (mean squared difference), 'cosine', and 'pearson'. By testing these different similarity metrics, we can identify the one that best captures the relationships between instances when taking into account the standardized ratings.

The **'min_support' parameter** determines the minimum number of common items required for two users to be considered neighbors. Evaluating this parameter with values like 3 and 5 helps us understand the impact of item overlap on the quality of recommendations when using z-scored ratings.

The **'user_based' parameter** being set to 'True' indicates that the model uses a user-based collaborative filtering approach, where recommendations are based on similarities between users. This setting can be compared with item-based collaborative filtering to determine which approach yields better results when utilizing z-scored ratings.

Finally, setting **"verbose" to 'False'** ensures that the model does not produce excessive output during training and evaluation, making the optimization process more streamlined.

By searching for the optimal combination of these parameters, we can fine-tune the KNN with Z score model to achieve improved performance and generate more accurate recommendations, specifically by considering the standardized ratings for enhanced similarity calculations and prediction accuracy.

In [752]:

```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'k': [10, 20, 30, 40],
    'sim_options' : {
        'name'       : ['msd','cosine','pearson'],
        'min_support': [3,5],
        'user_based' : [True]
        },
    "verbose" : [False]
    }

# Perform grid search with cross-validation
grid = RandomizedSearchCV(KNNWithZScore, param_distributions=param_grid, measures=['rmse'], cv=5)

# Fit the grid search object to the data
grid.fit(data)

# Get the best RMSE score and parameters
print("Best RMSE score:", grid.best_score['rmse'])
print("Best parameters:", grid.best_params['rmse'])

# Train the model on the full training set with the best parameters
algo = grid.best_estimator['rmse']
algo.fit(train_set)

# Evaluate the best model on the test set
predictions = algo.test(valid_set)

# Calculate FCP, RMSE, MSE & MAE
rmse = accuracy.rmse(predictions)
mae  = accuracy.mae(predictions)
mse  = accuracy.mse(predictions)
fcp  = accuracy.fcp(predictions)


final_models["Name"].append("KNNWithZScore")
final_models["Model"].append(algo)
final_models["FCP"].append(fcp)
final_models["RMSE"].append(rmse)
final_models["MSE"].append(mse)
final_models["MAE"].append(mae)
```

```
Best RMSE score: 1.4956894345343532
Best parameters: {'k': 40, 'sim_options': {'name': 'pearson', 'min_support': 5, 'user_based': True}, 'verbose': False}
RMSE: 1.5035
MAE:  1.1601
MSE: 2.2604
FCP:  0.7121
```

RMSE (Root Mean Square Error): RMSE measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. In this case, the RMSE value of 1.5078 indicates that, on average, the model's predictions for book ratings deviate from the actual ratings by approximately 1.5078 on the 5-10 scale. Lower RMSE values indicate better performance, as it means the model's predictions are closer to the actual ratings.

MAE (Mean Absolute Error): MAE also measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. The MAE value of 1.1639 suggests that, on average, the model's predictions deviate from the true ratings by approximately 1.1639 on the 5-10 scale. Similarly to RMSE, lower MAE values indicate better performance.

MSE (Mean Squared Error): MSE is another measure of the prediction accuracy, but it focuses on the squared differences between the model's predicted ratings and the actual ratings. The MSE value of 2.2734 represents the average of these squared differences. Like RMSE and MAE, lower MSE values indicate better performance.

FCP (Fraction of Concordant Pairs): FCP is a different type of evaluation metric often used in recommendation systems. It measures the proportion of pairs of items where the model correctly predicts the relative order of ratings. In this case, the FCP value of 0.7089 suggests that approximately 70.89% of the pairs are correctly ranked by the model. Higher FCP values indicate better performance, as it means the model is more accurate in predicting the relative rankings of book ratings.

Based on these evaluation metrics, it appears that the book recommendation model is performing reasonably well. The RMSE, MAE, and MSE values indicate that, on average, the model's predictions deviate from the true ratings by around 1.5 to 2.3 units on the 5-10 scale. The FCP value of 0.7089 suggests that the model is able to correctly rank the relative order of book ratings in approximately 70.89% of cases.

Next, we will try Matrix Factorization methods to see if we can get better results.

SVD (Singular Value Decomposition)

The first Matrix Factorization method we will use is Singular Value Decomposition (SVD). SVD is a highly suitable model for a book recommendation system due to its ability to capture latent factors and uncover meaningful patterns in user-item interactions. In a book recommendation system, SVD can decompose the user-item rating matrix into three separate matrices representing users, latent factors, and items. By reducing the dimensionality of the original matrix, SVD can effectively capture the underlying characteristics of both users and books. This enables the model to identify similarities between users with similar reading preferences and recommend books based on those patterns. SVD's low-rank approximation also helps address the sparsity issue commonly encountered in recommendation systems. Moreover, SVD provides interpretable factors that represent different book genres, topics, or user preferences, making it possible to explain the recommendations to users. Overall, SVD's ability to extract latent factors, handle sparsity, and offer interpretability makes it a powerful and effective model for book recommendation systems.

We will also be optimizing parameters for SVD. The parameters we will be optimizing are listed below.

'n_epochs' parameter: This parameter determines the number of iterations or epochs the model goes through during training. Trying different values such as 10, 20, and 30 allows us to find the optimal number of epochs that balances convergence and computational efficiency. Increasing the number of epochs can potentially improve the model's accuracy, but it may also increase the risk of overfitting.

'lr_all' parameter: This parameter represents the learning rate, which determines the step size taken during model optimization. The learning rate influences how quickly the model adapts to the training data and finds the optimal solution. Trying different values like 0.002, 0.005, and 0.01 enables us to identify the learning rate that leads to the best convergence and minimizes the loss function.

'reg_all' parameter: This parameter controls the regularization strength, which helps prevent overfitting by penalizing large parameter values. Regularization is essential for generalization and robustness of the model. Exploring values such as 0.2, 0.4, and 0.6 allows us to find the optimal level of regularization that strikes a balance between model complexity and avoiding overfitting.

By searching for the optimal combination of these parameters within the provided grid, we can fine-tune the SVD model to achieve improved performance and generate more accurate recommendations. This optimization process allows us to strike the right balance between convergence, learning rate, and regularization, leading to enhanced accuracy and better overall performance of the recommendation system.

In [753]:
```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_epochs': [10, 20, 30],
    'lr_all': [0.002, 0.005, 0.01],
    'reg_all': [0.2, 0.4, 0.6]
}

# Perform grid search with cross-validation
grid = RandomizedSearchCV(SVD, param_distributions=param_grid, measures=['rmse'], cv=5)

# Fit the grid search object to the data
grid.fit(data)

# Get the best RMSE score and parameters
print("Best RMSE score:", grid.best_score['rmse'])
print("Best parameters:", grid.best_params['rmse'])

# Train the model on the full training set with the best parameters
algo = grid.best_estimator['rmse']
algo.fit(train_set)

# Evaluate the best model on the test set
predictions = algo.test(valid_set)

# Calculate FCP, RMSE, MSE & MAE
rmse = accuracy.rmse(predictions)
mae  = accuracy.mae(predictions)
mse  = accuracy.mse(predictions)
fcp  = accuracy.fcp(predictions)


final_models["Name"].append("SVD")
final_models["Model"].append(algo)
final_models["FCP"].append(fcp)
final_models["RMSE"].append(rmse)
final_models["MSE"].append(mse)
final_models["MAE"].append(mae)
```

```
Best RMSE score: 1.3620999963748541
Best parameters: {'n_epochs': 20, 'lr_all': 0.01, 'reg_all': 0.4}
RMSE: 1.3731
MAE:  1.1049
MSE: 1.8853
FCP:  0.5483
```

RMSE (Root Mean Square Error): RMSE measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. In this case, the RMSE value of 1.3726 indicates that, on average, the model's predictions for book ratings deviate from the actual ratings by approximately 1.3726 on the 5-10 scale. Lower RMSE values indicate better performance, as it means the model's predictions are closer to the actual ratings.

MAE (Mean Absolute Error): MAE also measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. The MAE value of 1.0999 suggests that, on average, the model's predictions deviate from the true ratings by approximately 1.0999 on the 5-10 scale. Similarly to RMSE, lower MAE values indicate better performance.

MSE (Mean Squared Error): MSE is another measure of the prediction accuracy, but it focuses on the squared differences between the model's predicted ratings and the actual ratings. The MSE value of 1.8839 represents the average of these squared differences. Like RMSE and MAE, lower MSE values indicate better performance.

FCP (Fraction of Concordant Pairs): FCP is a different type of evaluation metric often used in recommendation systems. It measures the proportion of pairs of items where the model correctly predicts the relative order of ratings. In this case, the FCP value of 0.5420 suggests that approximately 54.20% of the pairs are correctly ranked by the model. Higher FCP values indicate better performance, as it means the model is more accurate in predicting the relative rankings of book ratings.

Based on these evaluation metrics, it appears that the book recommendation model is performing reasonably well. The RMSE, MAE, and MSE values indicate that, on average, the model's predictions deviate from the true ratings by around 1.1 to 1.9 units on the 5-10 scale. The FCP value of 0.5420 suggests that the model is able to correctly rank the relative order of book ratings in approximately 54.20% of cases.

NMF (Non-Negative Matrix Factorization)

NMF (Non-Negative Matrix Factorization) can outperform SVD in certain situations. NMF's advantage lies in its ability to handle non-negative data effectively, making it suitable for applications where negative values are not meaningful. Moreover, NMF often produces more interpretable factors, allowing for a better understanding of the underlying patterns. It is particularly valuable in fields like topic modeling and text analysis. Additionally, NMF's non-negative constraints enhance robustness to outliers and noise in the data.

We will also be optimizing parameters for SVD. The parameters we will be optimizing are listed below.

n_factors' parameter: This parameter determines the number of latent factors used to represent the original matrix. Trying different values such as 10, 20, and 30 allows us to find the optimal number of factors that capture the underlying patterns in the data. Increasing the number of factors can potentially improve the model's ability to represent the complexity of the data, but it may also lead to overfitting.

'n_epochs' parameter: This parameter represents the number of iterations or epochs the model goes through during training. Trying different values like 10, 20, and 30 allows us to find the optimal number of epochs that balances convergence and computational efficiency. Increasing the number of epochs can potentially improve the model's accuracy, but it may also increase the risk of overfitting.

'reg_pu' and 'reg_qi' parameters: These parameters control the regularization strength for user factors (reg_pu) and item factors (reg_qi). Regularization helps prevent overfitting by penalizing large parameter values. By exploring values such as 0.2, 0.4, and 0.6, we can find the optimal level of regularization that balances model complexity and overfitting prevention.

By searching for the optimal combination of these parameters within the provided grid, we can fine-tune the NMF model to achieve improved performance and generate more accurate recommendations. This optimization process allows us to strike the right balance between the number of factors, the number of epochs, and the regularization terms, leading to enhanced accuracy and better overall performance of the recommendation system.

In [754]:
```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_factors': [10, 20, 30],
    'n_epochs': [10, 20, 30],
    'reg_pu': [0.2, 0.4, 0.6],
    'reg_qi': [0.2, 0.4, 0.6]
}

# Perform grid search with cross-validation
grid = RandomizedSearchCV(NMF, param_distributions=param_grid, measures=['rmse', 'mae'], cv=5)

# Fit the grid search object to the data
grid.fit(data)

# Get the best RMSE and MAE scores and parameters
print("Best RMSE score:", grid.best_score['rmse'])
print("Best MAE score:", grid.best_score['mae'])
print("Best parameters:", grid.best_params['rmse'])

# Train the model on the full training set with the best parameters
algo = grid.best_estimator['rmse']
algo.fit(train_set)

# Evaluate the best model on the test set
predictions = algo.test(valid_set)

# Calculate FCP, RMSE, MSE & MAE
rmse = accuracy.rmse(predictions)
mae  = accuracy.mae(predictions)
mse  = accuracy.mse(predictions)
fcp  = accuracy.fcp(predictions)


final_models["Name"].append("NMF")
final_models["Model"].append(algo)
final_models["FCP"].append(fcp)
final_models["RMSE"].append(rmse)
final_models["MSE"].append(mse)
final_models["MAE"].append(mae)
```

```
Best RMSE score: 1.543049731654391
Best MAE score: 1.238902456381926
Best parameters: {'n_factors': 30, 'n_epochs': 30, 'reg_pu': 0.2, 'reg_qi': 0.6}
RMSE: 1.5540
MAE:  1.2520
MSE: 2.4151
FCP:  0.5837
```

RMSE (Root Mean Square Error): RMSE measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. In this case, the RMSE value of 1.5829 indicates that, on average, the model's predictions for book ratings deviate from the actual ratings by approximately 1.5829 on the 5-10 scale. Lower RMSE values indicate better performance, as it means the model's predictions are closer to the actual ratings.

MAE (Mean Absolute Error): MAE also measures the average magnitude of the differences between the model's predicted ratings and the actual ratings. The MAE value of 1.2697 suggests that, on average, the model's predictions deviate from the true ratings by approximately 1.2697 on the 5-10 scale. Similarly to RMSE, lower MAE values indicate better performance.

MSE (Mean Squared Error): MSE is another measure of the prediction accuracy, but it focuses on the squared differences between the model's predicted ratings and the actual ratings. The MSE value of 2.5057 represents the average of these squared differences. Like RMSE and MAE, lower MSE values indicate better performance.

FCP (Fraction of Concordant Pairs): FCP is a different type of evaluation metric often used in recommendation systems. It measures the proportion of pairs of items where the model correctly predicts the relative order of ratings. In this case, the FCP value of 0.5824 suggests that approximately 58.24% of the pairs are correctly ranked by the model. Higher FCP values indicate better performance, as it means the model is more accurate in predicting the relative rankings of book ratings.

Based on these evaluation metrics, it appears that the book recommendation model is performing reasonably well. The RMSE, MAE, and MSE values indicate that, on average, the model's predictions deviate from the true ratings by around 1.3 to 2.5 units on the 5-10 scale. The FCP value of 0.5824 suggests that the model is able to correctly rank the relative order of book ratings in approximately 58.24% of cases.

Final Model

We ran 6 models with 4 different evaluatin metrics. We will choose the best performing metric based on 2 metrics; FCP and RMSE, with FCP taking precedence.

When evaluating a book recommendation system, different metrics can be used to assess the model's performance. RMSE (Root Mean Square Error) is often considered a better metric than FCP (Fraction of Concordant Pairs), MSE (Mean Squared Error), and MAE (Mean Absolute Error) in certain contexts.

RMSE measures the average magnitude of the differences between predicted and actual ratings, considering both the magnitude and direction of errors. It provides a measure of overall accuracy, penalizing larger errors due to squaring and enabling comparison between models. FCP, on the other hand, focuses on the relative order of ratings and evaluates the model's ability to rank items correctly. While useful for ranking, FCP does not consider the magnitude of errors or absolute accuracy. MSE computes the average of squared differences and is sensitive to outliers, but its interpretation may be less intuitive without the square root operation. MAE, which measures the average absolute difference, is less sensitive to outliers but does not heavily penalize large errors. FCP would have been a good metric to use if we were worried about the ranking of the top 5 books amongst each other but since all the top 5 books would be sent to the customer, FCP won't reaally be beneficial.

Therefore we will use RMSE to choose which models to use since we will be using the predicted raatings to determine the top 5 books to use.

In [755]:
```
#sort for RMSE
final_models = pd.DataFrame(final_models).sort_values(by=['RMSE'])#.sort_values(by='RMSE')

#preview results
final_models
```

Out[755]:

	Name	Model	FCP	RMSE	MSE	MAE
4	SVD	<surprise.prediction_algorithms.matrix_factori...	0.548315	1.373078	1.885345	1.104876
1	KNNBaseline	<surprise.prediction_algorithms.knns.KNNBaseli...	0.551601	1.401719	1.964816	1.130863
3	KNNWithZScore	<surprise.prediction_algorithms.knns.KNNWithZS...	0.712085	1.503478	2.260447	1.160131
2	KNNWithMeans	<surprise.prediction_algorithms.knns.KNNWithMe...	0.711412	1.503965	2.261911	1.160396
5	NMF	<surprise.prediction_algorithms.matrix_factori...	0.583653	1.554047	2.415064	1.251961
0	KNNBasic	<surprise.prediction_algorithms.knns.KNNBasic ...	0.588548	1.617996	2.617913	1.270156

SVD is the best performing model with regards to *RMSE*. There will be a deviation of approximately 1.4 ratings which can be drastic in certain scenarios. To improve performance further, there will be more data needed.

Final Model Results

In [756]:
```
#select the final model
best_model = final_models["Model"].head(1).values[0]
```

In [757]:
```
# Evaluate the best model on the test set
valid_predictions = best_model.test(valid_set)


# Calculate FCP, RMSE
valid_rmse_score = accuracy.rmse(valid_predictions)
valid_mae_score = accuracy.mae(valid_predictions)
valid_mse_score = accuracy.mse(valid_predictions)
valid_fcp_score = accuracy.fcp(valid_predictions)
```

```
RMSE: 1.3731
MAE:  1.1049
MSE: 1.8853
FCP:  0.5483
```

```
In [758]:
    #make predictions
    test_predications = []
    for user_id,book_title,ratings in test_set:
        result = best_model.predict(uid = user_id, iid = book_title, r_ui = ratings)
        test_predications.append(result)

    # Calculate test scores
    test_rmse_score = accuracy.rmse(test_predications)
    test_mae_score = accuracy.mae(test_predications)
    test_mse_score = accuracy.mse(test_predications)
    test_fcp_score = accuracy.fcp(test_predications)
```

```
RMSE: 1.3543
MAE:  1.0931
MSE: 1.8343
FCP:  0.5697
```

```
In [759]:
    #combine results from valdaatiion set and test set evaluation
    valid_set_metrics = {
        "Metrics":["FCP","RMSE","MSE","MAE"],
        "Valid Set":[valid_fcp_score, valid_rmse_score, valid_mse_score, valid_mae_score],
        "Test Set":[test_fcp_score, test_rmse_score, test_mse_score, test_mae_score]
    }

    #convert to daraframe
    valid_set_metrics = pd.DataFrame(valid_set_metrics)
```

```
In [760]:
    #preview
    valid_set_metrics
```

Out[760]:

	Metrics	Valid Set	Test Set
0	FCP	0.548315	0.569722
1	RMSE	1.373078	1.354347
2	MSE	1.885345	1.834256
3	MAE	1.104876	1.093088

Looks like RMSE actually improved on the test set. Nonetheless, a swing of 1.3 ratings can push a book from a below average level to an above average level and change the complete outlook of how a book is perceived by the recommendation system. Lets proceed with using our recommendation model and taking a look at the books recommended.

```
In [761]:
    #predict ratings
    pred_ratings = []
    for user_id,book_title,ratings in test_set:
        result = best_model.predict(uid = user_id, iid = book_title, r_ui = ratings).est
        pred_ratings.append(result)

    #convert surprise test set to dataframe
    df_test = pd.DataFrame(test_set, columns=["User-ID","ISBN","Book-Rating"])

    #store predicted ratings
    df_test["Predicted Ratings"] = pred_ratings

    #preview ratings
    df_test.head()
```

Out[761]:

	User-ID	ISBN	Book-Rating	Predicted Ratings
0	253826	0553572997	8.0	7.985177
1	134403	0060938455	9.0	8.407789
2	134403	0440998050	9.0	8.865747
3	208751	0142001740	6.0	8.355166
4	208751	0375727132	8.0	7.748765

Recommendation System

Now, we will create a function to recommend the top 5 books using the final model selected above and leveraging collaborative filtering. Lets create a function for the recommendation system and then we will test it out.

In [762]:
```python
def recommendation(model, user_id: int, top: int):
    # Get all unique book ISBNs
    all_books = rec_set["ISBN"].unique()

    # Get all records of books that the user haas not rated yet
    prev_user_ratings = rec_set[rec_set["User-ID"] == user_id][["ISBN", "Book-Rating"]]

    # Initialize a dictionary to store book recommendations, including ISBN, title, and scores
    recommendation = {k: [] for k in ["book", "title", "score"]}

    # Iterate over each book ISBN
    for book in all_books:
        # Check if the user has previously rated the book
        if ((prev_user_ratings['ISBN'] == book).sum() == 1):
            rating = prev_user_ratings[prev_user_ratings["ISBN"] == book]

            # Append book ISBN and title to recommendations
            recommendation["book"].append(book)
            recommendation["title"].append(rec_set[rec_set['ISBN'] == book]['Book-Title'].iloc[0])

            # predict the estimated rating using the provided r_ui value
            est = model.predict(uid=user_id, iid=book).est
            recommendation["score"].append(est)
#         else:
#             # If the user has no previous rating, predict the estimated rating without providing the r_ui value
#             est = model.predict(uid=user_id, iid=book).est
#             recommendation["score"].append(est)

    # Convert the recommendation dictionary into a pandas DataFrame and sort it based on the "score" column
    rec = pd.DataFrame(recommendation).sort_values(by="score", ascending=False)

    # Print the top recommended books
    print(f"Top {top} Books recommended:")
    for k, i in enumerate(rec["title"].head(top).values):
        print(k + 1, i)

    # Return the top recommended book titles as a pandas Series with the index reset, and the full recommendation DataFrame
    return rec["title"].head(top).reset_index(drop=True), pd.DataFrame(rec)
```

Lets test out the function on a user ID and check the results. We will compare the recmmended books with the top books that the user rated originally. It might be possible that the top 5 recommended books are not the same as the original top rated books by the user. We will use domain knowledge to understand how close the recommended books are to the original top choices.

```
In [763]:   #define user id to test
            usid = 40889

            #call the function
            data, rec = recommendation(model = best_model ,user_id = usid, top= 5)

            #extract the ISBNs for the user with ratings of 10
            isbn = ml_ratings[(ml_ratings['User-ID'] == usid)&(ml_ratings['Book-Rating']==10)].sort_values(
                                                                        'Book-Rating', ascending=False)['ISBN']

            #print separatio results
            print('')
            print('-----------------------------------------------------------------------------------------------------')
            print('')
            print('Top Books originally rated by user:')

            #initialize counter
            counter = 0

            #run for loop to print the original top titles
            for val in (isbn):
                ttl = user_rating_books[(user_rating_books['ISBN']==val)]['Book-Title'].iloc[0]
                counter+=1
                print(counter, ttl)
```

```
Top 5 Books recommended:
1 Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))
2 Seabiscuit: An American Legend
3 Harry Potter and the Chamber of Secrets (Book 2)
4 Tribulation Force: The Continuing Drama of Those Left Behind (Left Behind No. 2)
5 Face the Fire (Three Sisters Island Trilogy)

-----------------------------------------------------------------------------------------------------

Top Books originally rated by user:
1 The Purpose-Driven Life: What on Earth Am I Here For?
2 The Hobbit : The Enchanting Prelude to The Lord of the Rings
3 Tuesdays with Morrie: An Old Man, a Young Man, and Life's Greatest Lesson
4 The Partner
5 Sea Swept (Quinn Brothers (Paperback))
6 Message in a Bottle
7 Rising Tides
8 Chicken Soup for the Soul (Chicken Soup for the Soul)
9 Heart of the Sea (Irish Trilogy)
10 The Lord of the Rings (Movie Art Cover)
11 The Deep End of the Ocean (Oprah's Book Club (Hardcover))
12 Jewels of the Sun (Irish Trilogy)
```

The top 5 recommended books include "Harry Potter and the Sorcerer's Stone," "Seabiscuit: An American Legend," "Harry Potter and the Chamber of Secrets," "Tribulation Force: The Continuing Drama of Those Left Behind," and "The Hunt for Red October."

When comparing these recommendations to the user's original highly rated books, we observe a shift in genres and themes. The original highly rated books cover a range of genres, including self-help, fantasy, contemporary fiction, and romance. In contrast, the recommended books emphasize popular titles from the fantasy, thriller, and adventure genres.

The recommended books, particularly the inclusion of the "Harry Potter" series, indicate that the system has identified highly regarded and widely beloved books that have resonated with a large audience. These recommendations offer a departure from the user's previous reading preferences, introducing them to new and popular series that have captivated readers worldwide.

While the recommended books may differ from the user's original highly rated books in terms of genre, they can be considered high-quality recommendations. These titles have gained acclaim for their captivating storytelling, engaging plots, and enduring popularity. By suggesting these popular and well-regarded books, the recommendation system aims to provide the user with the opportunity to explore widely appreciated literary works and potentially discover new favorites within these genres.

It's important to note that the quality of recommendations is subjective and dependent on individual reading preferences. The user's reception of the recommended books may vary, influenced by their personal tastes and openness to exploring different genres. Nevertheless, the inclusion of popular and acclaimed titles in the recommendations suggests that the system has identified widely recognized books that have the potential to engage and captivate readers.

Nonetheless, with such divergence, it is difficult to ascertain how successful the subscriptiion service might be.

Next Steps & Recommendation

While the recommendation system provides valuable good recommendations, there is a lot of room for improvement. The RMSE scores are still relatively high and there were a lot more records in the missing ratings dataset as compared to the known ratings dataset. The performance can be significaantly improved by gathering more data and building granular user and product personas, such as genre, demographic and other information. This would give the model more detailed information to predict from.

Moreover, incrporating a hybrid approach of content based and user based collaborative filtering would be beneficial. Currently, user-based collaborative filtering is used but using content based would also help. For content based filtering, it would be also be important to gether moree data.

Also, the current models can be further optimized using more parameters to find a further optimal solutions. This can help bring the RMSE score further down and help to improve the model performance.

Lastly, incorporating a solution for the cold start problem would allow incorporation of new users who do not have prior rating data. A feedback loop can also be added to update the ratings as users return or rate the new boooks. This way a constanly updated model will improvee performance based on new data.