MINI PROJECT

Book Recommendation System

CS-BS (BATCH- 2)

-BY DEV KARAN (RA1911042010113)

RAHUL NAIR
(RA1911042010117)

SHUBH
(RA1911042010124)

TABLE OF CONTENTS

Introduction

Recommendation systems are used in hundreds of different services - everywhere from online shopping to music to movies. For instance, the online retailer Amazon had a heavy hand in developing collaborative filtering algorithms that recommend items to users. Music services like Pandora identify up to 450 uniquely identifying characteristics of songs to find music similar to that of their users' preferences. Other music streaming services, such as Spotify, heavily rely upon the music selections of similar users to make weekly song recommendations and personalized radio stations. Netflix, a popular television and movie streaming service, uses these systems to recommend movies that viewers may enjoy. We can see how recommendation systems have a surprisingly large impact on the materials consumers engage with over the course of their daily lives.

Our project the book recommendations system plans to achieve just that. It lets you keep track of what all you have read. It will suggest what you might enjoy reading next based on your past books. It saves us the time from looking for the next book to read. This will save time and energy of the user and lead them to their next favorite as soon as possible.

Background

The most common types of recommendation systems are content-based and collaborative filtering recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models Memory-based methods and Model-based methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

User-based collaborative filtering: In this model, products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie come out that Derick like, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.

I**tem-based collaborative filtering:** These systems identify similar items based on users' previous ratings. For example, if users A, B, and C gave a 5-star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A, B, and C.

Model-based methods are based on Matrix Factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve accuracy. Examples of such model-based methods include Decision trees, Rule-based Model, Bayesian Model, and latent factor models.

Content-based systems use metadata such as genre, producer, actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly, you can get music recommendations from certain artists because you liked their music. Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

Datasets to use for building recommendation systems

For this project, we used three datasets: Books ,Ratings and Users.Users don’t tend to rate or read many books, so one can imagine how inherently sparse our data was. In order to alleviate this sparsity, we used the intersection (by books) of these two datasets for our project. The intention was to increase the number of ratings each book had. The intersection of both datasets resulted in 271,361 books, for there books we have around 700,000 ratings and reviews and there are 278,859 unique users in the User dataset that we used. That gave us a total of 1,048,576 users in the intersection. However, each algorithm has constraints that caused us to filter out any users who had rated less than three books. Which allowed us to tune different variables, finding the most accurate combination without overfitting, which partitioning as described helps to prevent. The Rating dataset contains more then 700,000 book ratings. On its own, this dataset would have limited us to a approach. By providing us with text we could use to model a book, these reviews allowed us to explore a UB approach.

Singular-Value Decomposition

Singular Value Decomposition (SVD), a classical method from linear algebra is getting popular in the field of data science and machine learning. This popularity is because of its application in developing recommender systems. There are a lot of online user-centric applications such as video players, music players, e-commerce applications, etc., where users are recommended with further items to engage with.

Finding and recommending many suitable items that would be liked and selected by users is always a challenge. There are many techniques used for this task and SVD is one of those techniques. This article presents a brief introduction to recommender systems, an introduction to singular value decomposition and its implementation in movie recommendation.

Example of Singular Value Decomposition

To understand the concept, let's suppose the matrix m × n, A, collects the training data set. These sets of data will take the row for each training vector. Here, N indicates that the dimension of each vector will be very large. By feeding the A in a clustering algorithm, you will generate a fixed number of cluster centers as the output. Since 'n' is quite large, the algorithm will be unstable or take too long. So, we will utilize singular value decomposition to reduce the number of variables. We will use a transparent method for computation, considering that we are still solving the problem with un-transformed coordinate.

Problem Statement

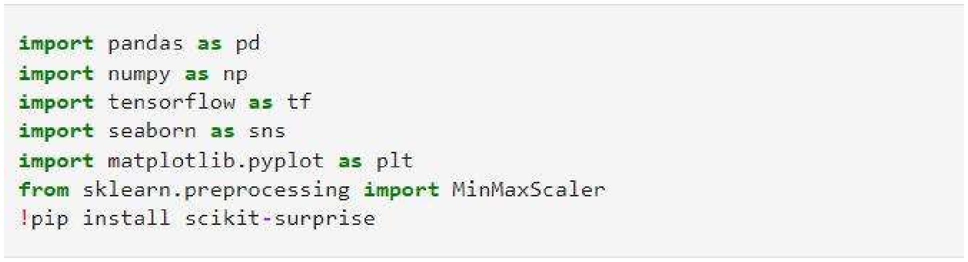
The aim of this Book Recommendation Engine, or “Book Recommender” is to provide interesting book recommendations to the user. The challenge with book recommendations, and unlike movie or music recommendation engines, is the vast amounts of book titles available, and the time investment for each of the books. Book recommenders are especially useful for the kind of readers that go into bookstores (physical or virtual) without a book title in mind.

If a reader has rated highly “Harry Potter — The Goblet of Fire (2000)” it may not come as a surprise he/she will also like “Harry Potter — The Order of the Phoenix (2003)”. While the book recommendation engine may provide a higher accuracy by predicting this title, it may not be very useful to the user of the engine.

The aim of the book recommender is not to provide a high accuracy, instead, difficult to quantify insightful book recommendations. It can be built using 3 algorithms: -
1.Collaborative Filtering
2.Using Cosine Similarity
3.Using KNN Algorithm

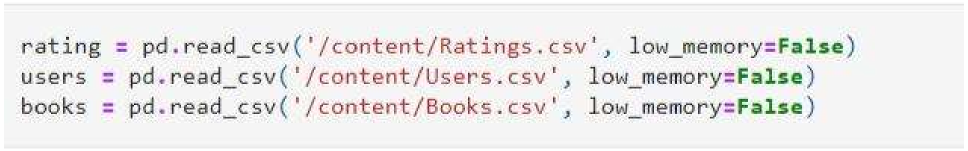
METHOD 1: USING COLLABORATIVE FILTERING

->The first thing we need to do is to import pandas and numpy.



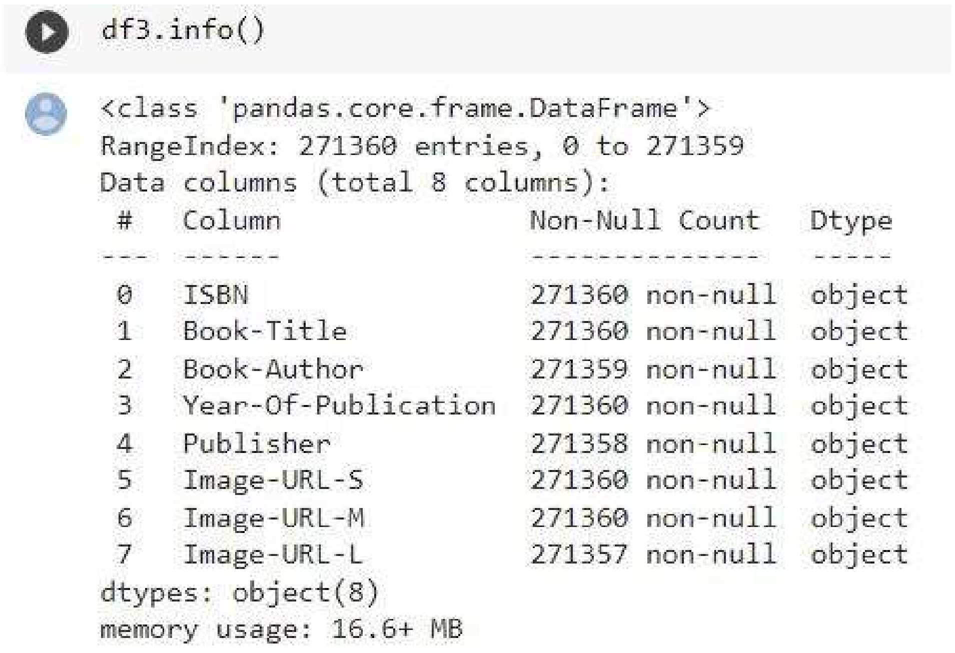
```
import pandas as pd
import numpy as np
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
!pip install scikit-surprise
```

->We load the dataset a books, users, and ratings.



```
rating = pd.read_csv('/content/Ratings.csv', low_memory=False)
users = pd.read_csv('/content/Users.csv', low_memory=False)
books = pd.read_csv('/content/Books.csv', low_memory=False)
```

->Data info

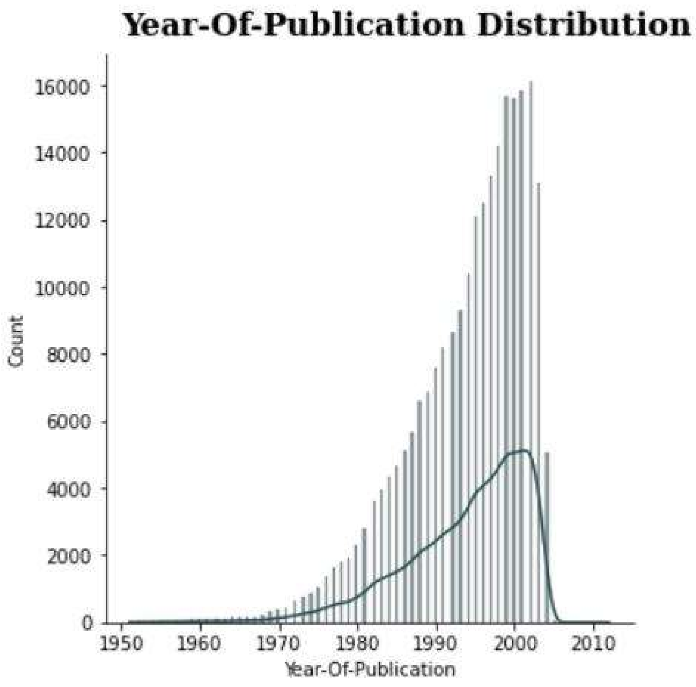


```
df3.info()
```

```
<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
```

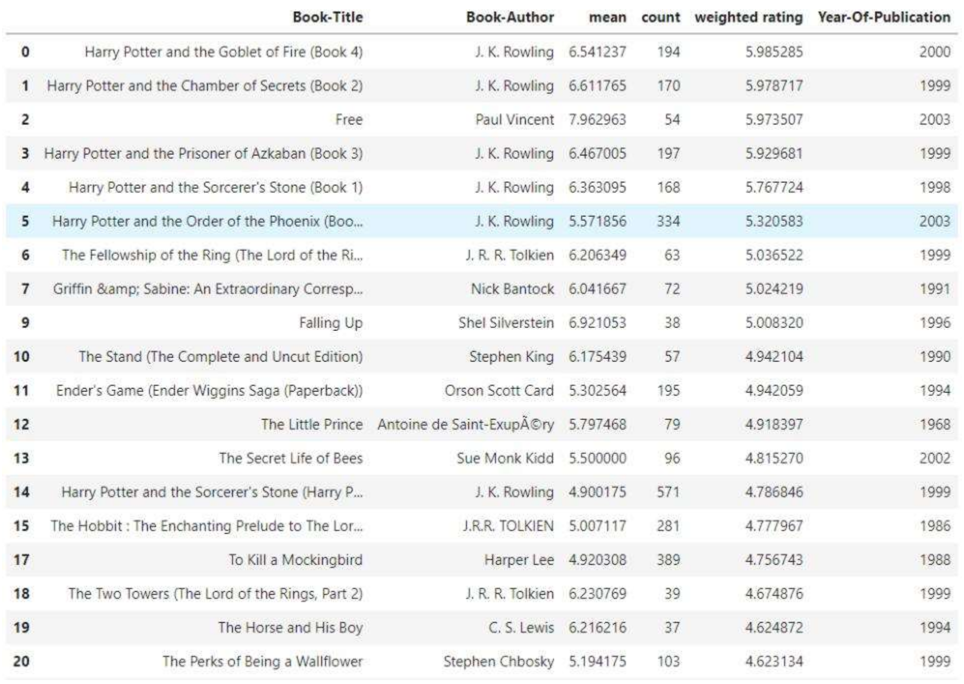
->Now let's see the no of book with respect to `Year Of Publication`

```
books = books[~books['Year-Of-Publication'].isin(['DK Publishing Inc',
'Gallimard'])]books['Year-Of-Publication'] =
pd.to_numeric(books['Year-Of-Publication'])

df3 = df3[(df3['Year-Of-Publication']>1950) & (df3['Year-Of-
Publication']<=2016)]

plot_distribution('Year-Of-Publication', df3)
```



->Generate score based on mean rating and total number of times the book is rated

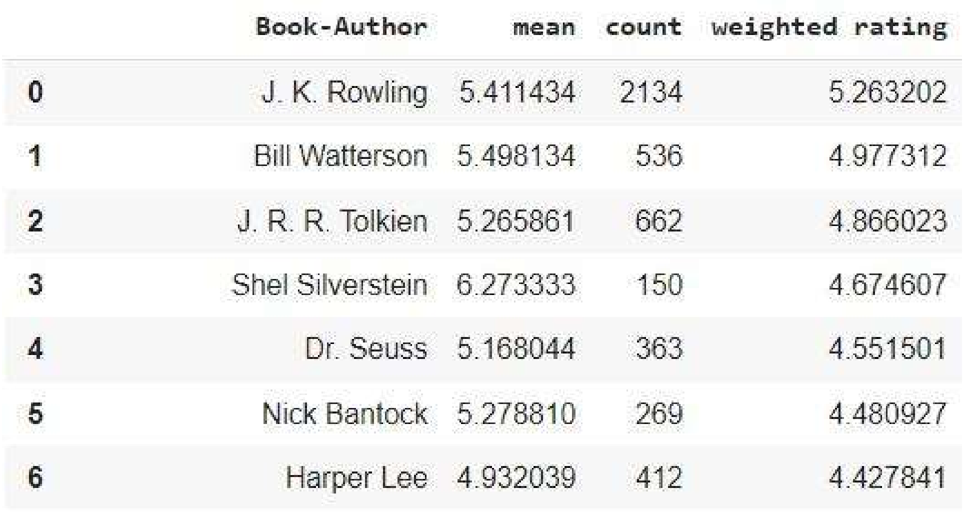
```
data = df1.groupby('ISBN').agg(['mean', 'count'])['Book-Rating'].reset_index()

m = data['count'].quantile(0.99) # minimum votes required to be listed in the Top

250 data = data[data['count']>m]

print('m =', m)

print(data.shape)

R = data['mean'] # average for the book (mean) = (Rating)

v = data['count'] # number of votes for the book = (votes)

C = data['mean']. mean() # mean vote across all books

data['weighted rating'] = (v/(v+m))*R + (m/(v+m))*C

data = data.sort_values('weighted rating', ascending=False).reset_index(drop=True)

    data = pd.merge(data, df3, on='ISBN')[['Book-Title', 'Book-Author', 'mean', 'count', 'weighted
                    rating', 'Year-Of-Publication']].drop_duplicates('Book-Title').iloc[:20] data
```



	Book-Title	Book-Author	mean	count	weighted rating	Year-Of-Publication
0	Harry Potter and the Goblet of Fire (Book 4)	J. K. Rowling	6.541237	194	5.985285	2000
1	Harry Potter and the Chamber of Secrets (Book 2)	J. K. Rowling	6.611765	170	5.978717	1999
2	Free	Paul Vincent	7.962963	54	5.973507	2003
3	Harry Potter and the Prisoner of Azkaban (Book 3)	J. K. Rowling	6.467005	197	5.929681	1999
4	Harry Potter and the Sorcerer's Stone (Book 1)	J. K. Rowling	6.363095	168	5.767724	1998
5	Harry Potter and the Order of the Phoenix (Boo...	J. K. Rowling	5.571856	334	5.320583	2003
6	The Fellowship of the Ring (The Lord of the Ri...	J. R. R. Tolkien	6.206349	63	5.036522	1999
7	Griffin &amp; Sabine: An Extraordinary Corresp...	Nick Bantock	6.041667	72	5.024219	1991
9	Falling Up	Shel Silverstein	6.921053	38	5.008320	1996
10	The Stand (The Complete and Uncut Edition)	Stephen King	6.175439	57	4.942104	1990
11	Ender's Game (Ender Wiggins Saga (Paperback))	Orson Scott Card	5.302564	195	4.942059	1994
12	The Little Prince	Antoine de Saint-ExupÃ©ry	5.797468	79	4.918397	1968
13	The Secret Life of Bees	Sue Monk Kidd	5.500000	96	4.815270	2002
14	Harry Potter and the Sorcerer's Stone (Harry P...	J. K. Rowling	4.900175	571	4.786846	1999
15	The Hobbit : The Enchanting Prelude to The Lor...	J.R.R. TOLKIEN	5.007117	281	4.777967	1986
17	To Kill a Mockingbird	Harper Lee	4.920308	389	4.756743	1988
18	The Two Towers (The Lord of the Rings, Part 2)	J. R. R. Tolkien	6.230769	39	4.674876	1999
19	The Horse and His Boy	C. S. Lewis	6.216216	37	4.624872	1994
20	The Perks of Being a Wallflower	Stephen Chbosky	5.194175	103	4.623134	1999

->Generate score based on mean rating and total number of times the author is rated

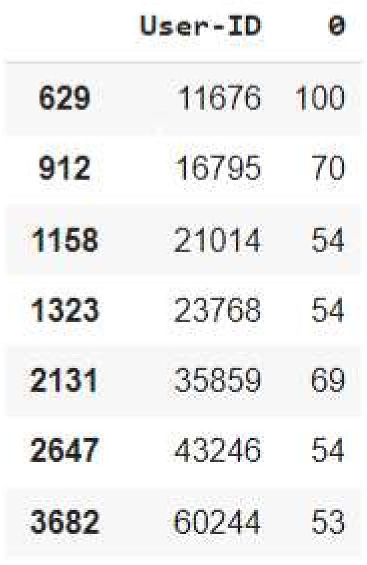
```
df3 = df3.drop_duplicates(['Book-Author', 'Book-Title'])

data = pd.merge(df3, df1, on='ISBN')[['Book-Author', 'Book-Rating', 'Book-Title', 'ISBN']]

data = data.groupby('Book-Author').agg(['mean', 'count'])['Book-Rating'].reset_index()

m = data['count'].quantile(0.99) # minimum votes required to be listed in the Top

250 data = data[data['count']>m]

print('m =', m)

print(data.shape)

R = data['mean'] # average for the author (mean) = (Rating)

v = data['count'] # number of votes for the author = (votes)

C = data['mean'].mean() # mean vote across all authors

data['weighted rating'] = (v/(v+m))*R + (m/(v+m))*C

data = data.sort_values('weighted rating', ascending=False).reset_index(drop=True)

data.iloc[:20]
```



	Book-Author	mean	count	weighted rating
0	J. K. Rowling	5.411434	2134	5.263202
1	Bill Watterson	5.498134	536	4.977312
2	J. R. R. Tolkien	5.265861	662	4.866023
3	Shel Silverstein	6.273333	150	4.674607
4	Dr. Seuss	5.168044	363	4.551501
5	Nick Bantock	5.278810	269	4.480927
6	Harper Lee	4.932039	412	4.427841

->Get user-ID for users who have read more than 50 books

```
temp = df[~df.isna()].count(axis=1).reset_index()

temp[temp[0]>50]
```



	User-ID	0
629	11676	100
912	16795	70
1158	21014	54
1323	23768	54
2131	35859	69
2647	43246	54
3682	60244	53

->Import from Surprise Library Reader to get the rating scale, Dataset to load the data and SVD () to act as the algorithm to train the data.

```
In [54]: from surprise import Reader, Dataset, SVD
         from surprise.model_selection import train_test_split, cross_validate
```

```
In [55]: reader = Reader(rating_scale=(0, 10))
         surprise_data = Dataset.load_from_df(data[['User-ID', 'ISBN', 'Book-Rating']], reader)
         trainset, testset = train_test_split(surprise_data, test_size=0.25)
```

```
In [56]: benchmark = []
         for algorithm in [SVD()]:
             results = cross_validate(algorithm, surprise_data, measures=['RMSE'], cv=3, verbose=False)
             tmp = pd.DataFrame.from_dict(results).mean(axis=0)
             tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], index=['Algorithm']))
             benchmark.append(tmp)

         pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

Out[56]:

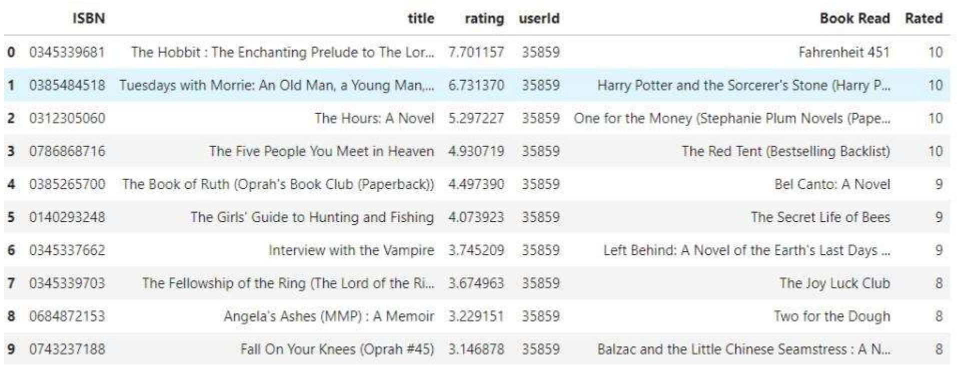
	test_rmse	fit_time	test_time
Algorithm			
SVD	3.921047	1.680262	0.107684

```
In [57]: svd = SVD()
         svd.fit(trainset)
```

```
Out[57]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fd26e853210>
```

Get high rated books by user

```
index_val = 2131
userId = df.index[index_val]
books = []
ratings = []
titles = []

for isbn in df.iloc[index_val][df.iloc[index_val].isna()].index:
    books.append(isbn)
    title = data[data['ISBN']==isbn]['Book-Title'].values[0]
    titles.append(title)
    ratings.append(svd.predict(userId, isbn).est)

prediction = pd.DataFrame({'ISBN':books, 'title':titles, 'rating':ratings,
'userId':userId})
prediction = prediction.sort_values('rating',
ascending=False).iloc[:10].reset_index(drop=True)

temp = data[data['User-ID']==df.index[index_val]].sort_values(
    'Book-Rating', ascending=False)[['Book-Rating', 'Book-Title', 'User-
ID']].iloc[:10].reset_index(drop=True)
prediction['Book Read'] = temp['Book-Title']
prediction['Rated']= temp['Book-Rating']
prediction
```



	ISBN	title	rating	userId	Book Read	Rated
0	0345339681	The Hobbit : The Enchanting Prelude to The Lor...	7.701157	35859	Fahrenheit 451	10
1	0385484518	Tuesdays with Morrie: An Old Man, a Young Man,...	6.731370	35859	Harry Potter and the Sorcerer's Stone (Harry P...	10
2	0312305060	The Hours: A Novel	5.297227	35859	One for the Money (Stephanie Plum Novels (Pape...	10
3	0786868716	The Five People You Meet in Heaven	4.930719	35859	The Red Tent (Bestselling Backlist)	10
4	0385265700	The Book of Ruth (Oprah's Book Club (Paperback))	4.497390	35859	Bel Canto: A Novel	9
5	0140293248	The Girls' Guide to Hunting and Fishing	4.073923	35859	The Secret Life of Bees	9
6	0345337662	Interview with the Vampire	3.745209	35859	Left Behind: A Novel of the Earth's Last Days ...	9
7	0345339703	The Fellowship of the Ring (The Lord of the Ri...	3.674963	35859	The Joy Luck Club	8
8	0684872153	Angela's Ashes (MMP) : A Memoir	3.229151	35859	Two for the Dough	8
9	0743237188	Fall On Your Knees (Oprah #45)	3.146878	35859	Balzac and the Little Chinese Seamstress : A N...	8

METHOD 2: USING COSINE SIMILARITY

Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

Cosine similarity function is declared to find the most similar books to a particular chosen book using the angles between them. It doesn't depend on Euclidean distance like KNN Algorithm.

Algorithm: -

->In the second model, the book, user, and rating data are merged according to the ISBN of the books and the user-id of each user.
->Duplicate columns for the new data are removed.
->A pivot matrix is created for the data to pivot the columns and replace NA with 0.
->SVD is used for matrix factorization of the pivot matrix.
->Cosine similarity function is declared to find the most similar books to a particular chosen book using the angles between them. It doesn't depend on Euclidean distance like KNN Algorithm.
Libraries used:
->Pandas
->NumPy
->SciPy

```
import numpy as np
import pandas as pd
```

```
book_df=pd.read_csv('/content/Books.csv')
ratings_df = pd.read_csv('/content/Ratings.csv').sample(40000)
user_df = pd.read_csv('/content/Users.csv')
user_rating_df = ratings_df.merge(user_df, left_on = 'User-ID', right_on = 'User-ID')
```

```
book_user_rating = book_df.merge(user_rating_df, left_on = 'ISBN',right_on = 'ISBN')
book_user_rating = book_user_rating[['ISBN', 'Book-Title', 'Book-Author', 'User-ID', 'Book-Rating']]
book_user_rating.reset_index(drop=True, inplace = True)
```

```
d ={}
for i,j in enumerate(book_user_rating.ISBN.unique()):
    d[j] =i
book_user_rating['unique_id_book'] = book_user_rating['ISBN'].map(d)
```

```
users_books_pivot_matrix_df = book_user_rating.pivot(index='User-ID',
                                                     columns='unique_id_book',
                                                     values='Book-Rating').fillna(0)
```

```
def top_cosine_similarity(data, book_id, top_n=10):
    index = book_id
    book_row = data[index, :]
    magnitude = np.sqrt(np.einsum('ij, ij -> i', data, data))
    similarity = np.dot(book_row, data.T) / (magnitude[index] * magnitude)
    sort_indexes = np.argsort(-similarity)
    return sort_indexes[:top_n]

def similar_books(book_user_rating, book_id, top_indexes):
    print('Recommendations for {0}: \n'.format(
    book_user_rating[book_user_rating.unique_id_book == book_id]['Book-Title'].values[0]))
    for id in top_indexes + 1:
        print(book_user_rating[book_user_rating.unique_id_book == id]['Book-Title'].values[0])
```

```
k = 50
m_id =2594
top_n = 3
sliced = Vt.T[:, :k] # representative data

similar_books(book_user_rating, 2594, top_cosine_similarity(sliced, m_id, top_n))
```

METHOD 3: USING KNN ALGORITHM

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

ALGORITHM: -

->In the third model, after pre-processing data, user and rating data are merged according to the ISBN of the books and the user-id of each user.

->'Image-URL-S', 'Image-URL-M', 'Image-URL-L' columns are removed as we do not need them, and they might skew the data.

->We calculate the Manhattan distance, Euclidean distance and Minkowski distance.

->We find the nearest neighbour to implement a simple popularity-based Recommendation system.

->It shows us Manhattan distance between user 192762 with 500 other users' distance to suggest book.

Libraries used:

->NumPy

->Pandas

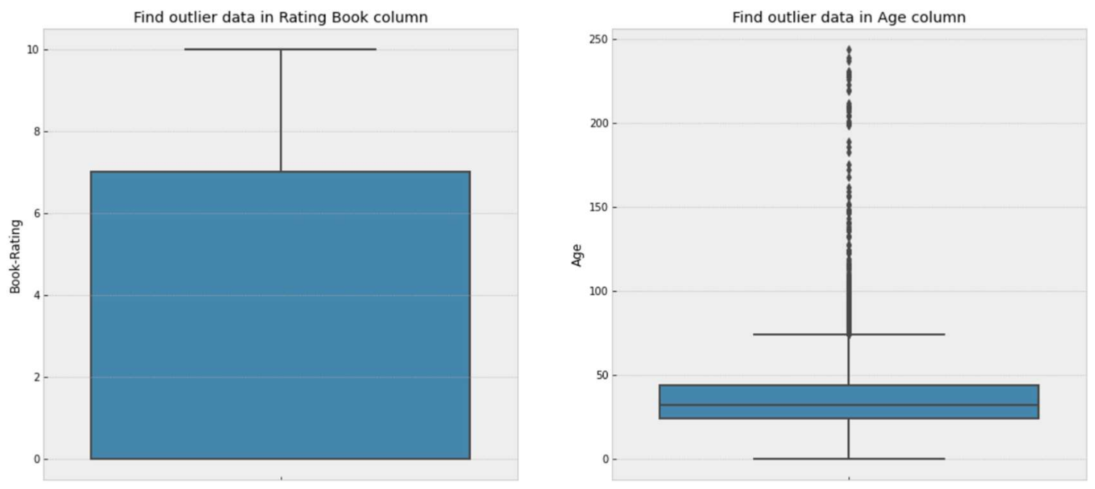
->Matplotlib

->Seaborn

->Missingno

OUTPUT: -

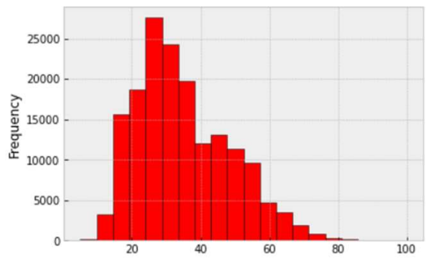
```
In [13]:  f,ax=plt.subplots(1,2,figsize=(18,8))
          sns.boxplot(y='Book-Rating', data=Ratings_df,ax=ax[0])
          ax[0].set_title('Find outlier data in Rating Book column')
          sns.boxplot(y='Age', data=Users_df,ax=ax[1])
          ax[1].set_title('Find outlier data in Age column')

Out[13]:  Text(0.5, 1.0, 'Find outlier data in Age column')
```



In [35]:
```
# outlier data became NaN
Users_df.loc[(Users_df.Age > 100 ) | (Users_df.Age < 5),'Age']=np.nan
```

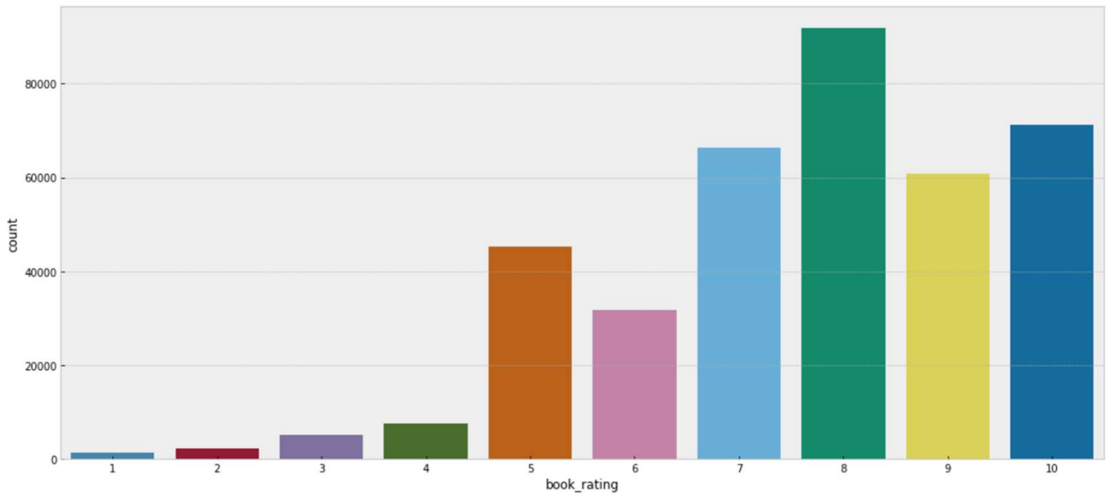
In [36]:
```
Users_df.Age.plot.hist(bins=20,edgecolor='black',color='red')
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34732d8390>
```



In [61]:
```
fig, ax = plt.subplots(figsize=(18,8))
sns.countplot(data=ratings_1to10,x='book_rating',ax=ax)
```

Out[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34715f6d90>
```



In [71]:

```python
def recommend(username):
    """Give list of recommendations"""
    # first find nearest neighbor
    nearest=computeNearestNeighbor(username)[0][1]
    recommendations=[]
    # now find bands neighbor rated that user didn't
    neighborRatings = dataset.loc[dataset.user_id==nearest].Book_Title.tolist()
    userRatings = dataset.loc[dataset.user_id==username].Book_Title.tolist()
    for artist in neighborRatings:
        if not artist in userRatings:
            recommendations.append((artist,int(dataset[(dataset.Book_Title==artist) & (dataset.user_id==nearest)].book_rating)))
    return sorted(recommendations,key=lambda artistTuple : artistTuple[1],reverse=True)
print(recommend(192762))
```

```
[('Goodbye to the Buttermilk Sky', 7), ('The Witchfinder (Amos Walker Mystery Series)', 6), ('More Cunning Than Man: A Social History of Rats and Man',
6), ('Clara Callan', 5), ("Where You'll Find Me: And Other Stories", 5), ('The Middle Stories', 5), ('Jane Doe', 5)]
```

LIBRARIES USED

PANDAS

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

LIBRARY FEATURES: -

- DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation[6] and frequency conversions, moving window statistics, moving window linear regressions, date shifting and lagging.
- Provides data filtration.

NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

SURPRISE

Surprise is a Python scikit for building and analysing recommender systems that deal with explicit rating data.

Surprise **was designed with the following purposes in mind**:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on documentation, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.

- Alleviate the pain of Dataset handling. Users can use both built-in datasets (Movielens, Jester), and their own custom datasets.
- Provide various ready-to-use prediction algorithms such as baseline algorithms, neighbourhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson...) are built-in.
- Make it easy to implement new algorithm ideas.
- Provide tools to evaluate, analyse and compare the algorithms' performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by scikit-learn excellent tools), as well as exhaustive search over a set of parameters.

The name SurPRISE stands for Simple Python Recommendation System Engine.

MATPLOTLIB

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats .
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

SEABORN

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes, and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

MISSINGNO

In the case of a real-world dataset, it is very common that some values in the dataset are missing. We represent these missing values as NaN (Not a Number) values. But to build a good machine learning model our dataset should be complete. That's why we use some imputation techniques to replace the NaN values with some probable values. But before doing that we need to have a good understanding of how the NaN values are distributed in our dataset.

Missingno library offers a very nice way to visualize the distribution of NaN values. Missingno is a Python library and compatible with Pandas.

SCIKIT-LEARN

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn.

Some popular groups of models provided by scikit-learn include:

- **Clustering**: for grouping unlabeled data such as KMeans.

- **Cross Validation**: for estimating the performance of supervised models on unseen data.
- **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
- **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- **Ensemble methods**: for combining the predictions of multiple supervised models.
- **Feature extraction**: for defining attributes in image and text data.
- **Feature selection**: for identifying meaningful attributes from which to create supervised models.
- **Parameter Tuning**: for getting the most out of supervised models.
- **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
- **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

REFERENCES

https://www.kaggle.com/arashnic/book-recommendation-dataset
https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/
https://en.wikipedia.org/wiki/Pandas_(software)
https://matplotlib.org/
https://www.machinelearningplus.com/nlp/cosine-similarity/
https://www.geeksforgeeks.org/python-visualize-missing-values-nan-values-using-missingno-library/
http://surpriselib.com/
https://numpy.org/doc/stable/user/whatisnumpy.html
https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning