Book Rating Analysis

Dasun Wellawalage
DSC 680
https://github.com/dasun27/Bellevue/blob/gh-pages/index.md

Business Problem

As more and more users have become online readers and it is important to find books that are appealing to you. It is often hard to figure out how good a book is by its cover. This is where the user ratings and recommendation systems come into play. It is crucial that online reading platform vendors get recommend good books to their users, so they keep coming back to the site. In this analysis, I am trying to build a simple book recommendation system that can be extended to a real-world scenario.

Method

For this analysis I am using a dataset taken from Kaggle. This data set consists of three separate data files and one of them has 900k+ records. Altogether there are 8 variables and only 5 of them are useful to work with as others are related to images and we are not going to perform any image analysis here. I am using both content based and collaborative filtering to build the recommendation system.

1. Loading Data – This should have been a simple task. Unfortunately, in this case one of the CSV files had semicolon as the column separator and the same file had many string values that contained semicolons. Parsing through the file to extract all the columns accurately took an enormous amount of time and effort.

```
users = pd.read_csv('BX-Users.csv', sep=';', encoding='latin-1')

users.head()

import csv



datafile = open('BX-Books.csv', 'r', encoding='latin-1')

myreader = csv.reader(datafile)
```

```
data = []


for row in myreader:
    line = ''
    for col in range(22): # Checking for first 22 columns
        if row[col] != '':
            line = line + row[col]
        else:
            break
      data.append(line.strip())
datafile.close()


# removing ampersands
new_data = []
remove_list = ['Edith Delatush',
 'Dylan Thomas',
 'Dougal Robertson',
 …. Rest of the list omitted for brevity ]


for n in data:
    append = True
    for x in remove_list:
        if x in n:
            append = False

            break

    if append:

        new_data.append(n.replace("&amp;", "&").replace(";:", ":").replace(" ; ", " ").replace("; ", "
").replace("&lt;", "<").replace(';\\', ":").replace('g;', "g"))


# Creating the dataframe from list

books = pd.DataFrame([sub.split(";") for sub in new_data])


# Removing empty columns

cols = [8]

books.drop(books.columns[cols],axis=1,inplace=True)


# extracting the column names form the first line

books.columns=books.iloc[0]

books = books[1:]


# Removing quotes from column names

rm_quote = lambda x: x.replace('"', '')

books = books.rename(columns=rm_quote)


# Removing quotes from the entire dataset

books = books.applymap(lambda x: x.replace('"', '') if (isinstance(x, str)) else x)

books.head()
```

2. Data Cleanup – Had to cleanup some null values and replace them with other values such as the average for the ‘Age’ column. Deleted two lines that had ,none. Values for most of the columns.

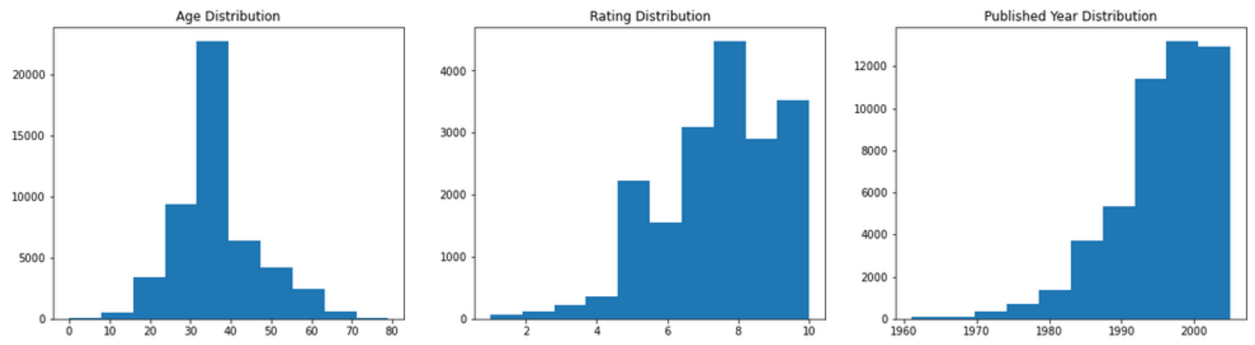
```
users.isnull().sum()
users['Age'] = users['Age'].fillna(34.5)

books.isnull().sum()
books.drop(books.index[84125], inplace=True)
books.reset_index(inplace=True)
```

3. Merging datasets – Three separate datasets for users, books, and ratings were merged into one for ease of analysis.

```
df = pd.merge(users, ratings, on='User-ID', left_index=False)
df = pd.merge(df, books, on='ISBN', left_index=False, right_index=False)
del df['index']
df.head()
```

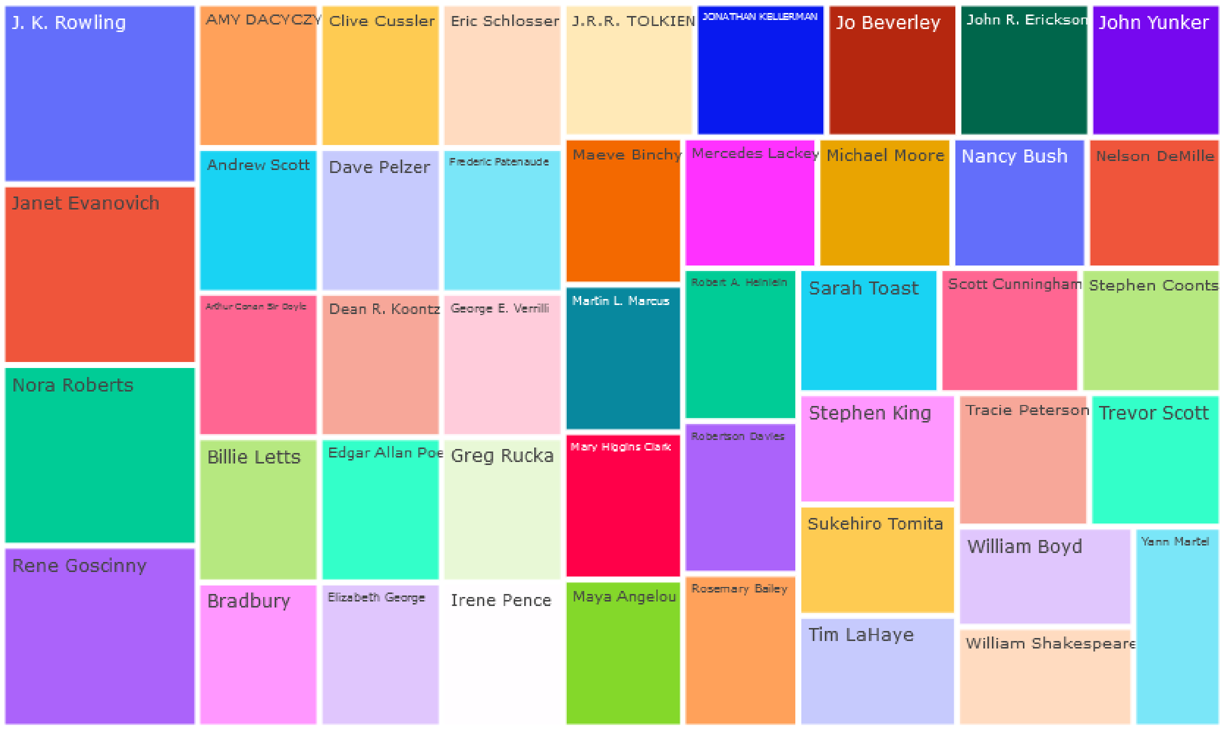
4. Next, I started exploring the dataset by using visualization techniques.

```
df = df.sample(n = 50000)
temp_Age = df[df['Age'] < 80]
temp_Book_Rating = df[df['Book-Rating'] > 0]
temp_YoP = df[df['Year-Of-Publication'] > 1960][df['Year-Of-Publication'] < 2010]

fig, axes = plt.subplots(nrows = 1, ncols = 3, figsize=(20, 5))
axes[0].hist(df[df['Age'] < 80]['Age'])
axes[0].set_title("Age Distribution")
axes[1].hist(df[df['Book-Rating'] > 0]['Book-Rating'])
axes[1].set_title("Rating Distribution")
axes[2].hist(df[df['Year-Of-Publication'] > 1960][df['Year-Of-Publication'] < 2010]['Year-Of-
Publication'])
axes[2].set_title("Published Year Distribution")
```



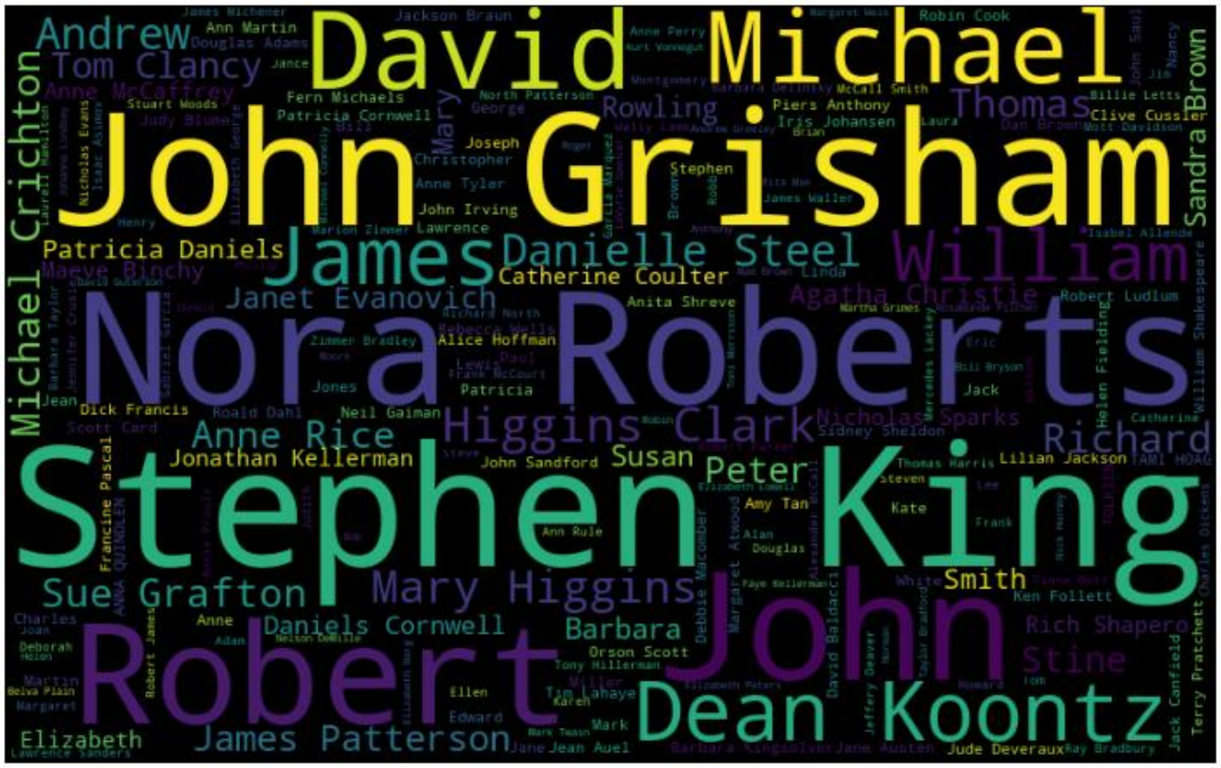
5. I used a tree map to visualize top books and top authors.

```
top_rated = df.sort_values('Book-Rating', ascending=False)
tf_top_rated = top_rated[:25]
fig = px.treemap(tf_top_rated, path=['Book-Title'], values='Book-Rating',title='Top Rated Books',
width=1000, height=700)
fig.show()

fifty_top_authors = top_rated[:50]
fig = px.treemap(fifty_top_authors, path=['Book-Author'], values='Book-Rating',title='Top
Authors', width=1000, height=700)
fig.show()
```



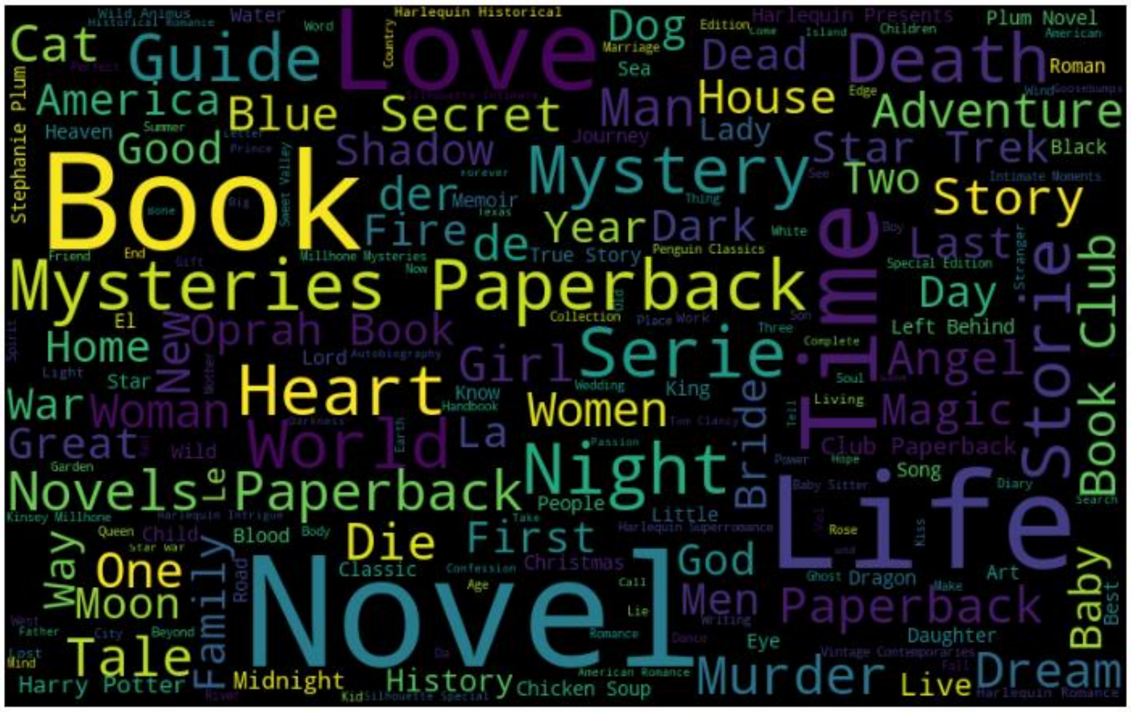
6. I also used wordclouds to better visualize top authors and books.

```
stop_words=set(STOPWORDS)

def wordcloud(string):
    wc = WordCloud(width=800,height=500,mask=None,random_state=21,
max_font_size=110,stopwords=stop_words).generate(string)
    fig=plt.figure(figsize=(16,8))
    plt.axis('off')
    plt.imshow(wc)

# word cloud for authors
authors = " ".join(df['Book-Author'])
wordcloud(authors)
```



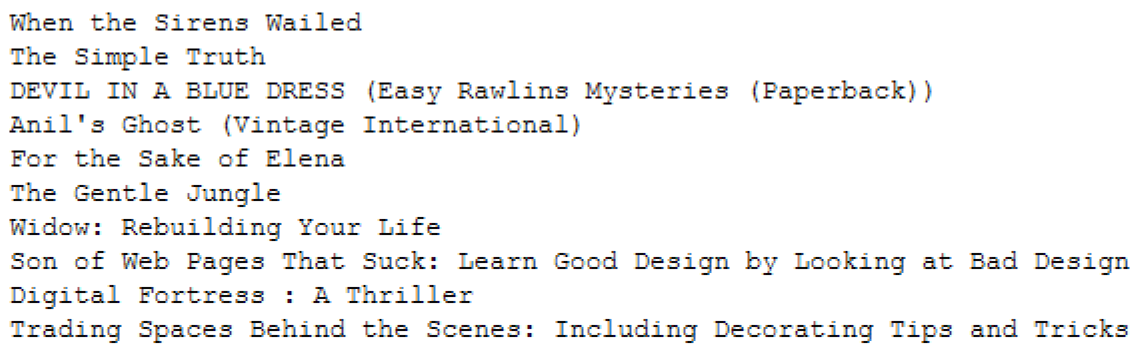
word cloud for books

```
books = " ".join(df['Book-Title'])
wordcloud(books)
```



7. Next, I started working on my models. I used cosine similarity for content-based filtering. Again, I had to pick a tiny subset of the dataset as my machine couldn't handle at least 10% of the dataset with these models. I also used the TFIDF vectorizer with method.

```
subdf = df.sample(n = 2000)
content = subdf[['Book-Title','Book-Author','Book-Rating']]
content = content.astype(str)
content['content'] = content['Book-Title'] + ' ' + content['Book-Author'] + ' ' + content['Book-
Rating']
content = content.reset_index()
indices = pd.Series(content.index, index=content['Book-Title'])

tfidf = TfidfVectorizer(stop_words='english')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(content['Book-Author'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape

cosine_sim_author = linear_kernel(tfidf_matrix, tfidf_matrix)

def get_recommendations_books(title, cosine_sim=cosine_sim_author):
    idx = indices[title]

    # Get the pairwsie similarity scores of all books with that book
    sim_scores = list(enumerate(cosine_sim_author[idx]))

    # Sort the books based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar books
    sim_scores = sim_scores[1:11]
    # Get the book indices
    book_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar books
    return list(content['Book-Title'].iloc[book_indices])



def author_book_shows(book):
    for book in book:
        print(book)

books = get_recommendations_books('The Prodigy', cosine_sim_author)
author_book_shows(books)
```



```
When the Sirens Wailed
The Simple Truth
DEVIL IN A BLUE DRESS (Easy Rawlins Mysteries (Paperback))
Anil's Ghost (Vintage International)
For the Sake of Elena
The Gentle Jungle
Widow: Rebuilding Your Life
Son of Web Pages That Suck: Learn Good Design by Looking at Bad Design
Digital Fortress : A Thriller
Trading Spaces Behind the Scenes: Including Decorating Tips and Tricks
```

8. Then I tried the content-based filtering again, but this time with the count vectorizer.

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(content['content'])

cosine_sim_content = cosine_similarity(count_matrix, count_matrix)

def get_recommendations(title, cosine_sim=cosine_sim_content):
    idx = indices[title]

    # Get the pairwsie similarity scores of all books with that book
    sim_scores = list(enumerate(cosine_sim_content[idx]))

    # Sort the books based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar books
    sim_scores = sim_scores[1:11]

    # Get the book indices
    book_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar books
    return list(content['Book-Title'].iloc[book_indices])

def book_shows(book):
    for book in book:
        print(book)

books = get_recommendations('The Prodigy', cosine_sim_content)
book_shows(books)
```

```
Deception on His Mind
Deception on His Mind
The Remarkable Women of the Bible Growth: And Their Message for Your Life Today
Almost: A Novel
Winter Fire
Watership Down
Museum Pieces
Moving Target
Destinations South
Too Hot To Handle
```

9. Finally, I used the collaborative filtering method. Since we have a really sparse dataset I removed zero rated entries from ‘users’ dataframe before merging it with others.

```
nonzero_ratings = ratings[ratings['Book-Rating'] != 0]
df = pd.merge(users, nonzero_ratings, on='User-ID', left_index=False)
df = pd.merge(df, books, on='ISBN', left_index=False, right_index=False)
del df['index']
df.head()
```

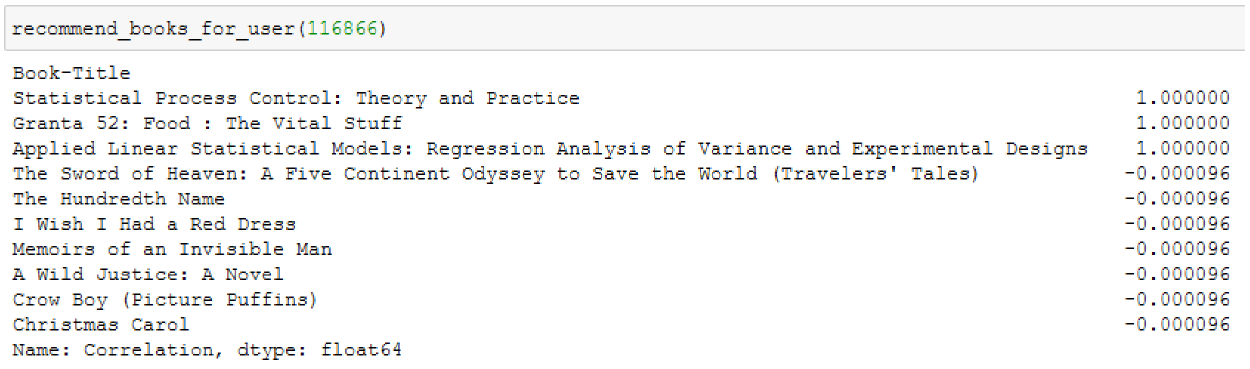
10. Next, I picked a rando sample of 20k records from the original merged dataframe as the original dataframe was too big for my machine to process.

```
subdf = df.sample(n = 20000)
rating = subdf.pivot_table(index='User-ID', values='Book-Rating', columns='Book-
Title').fillna(0)
```

11. There are multiple methods that can be used with collaborative filtering. They belong to two categories namely, memory based and model based. Here I am trying the memory-based model first with an item based approach and then with a user based approach.

```
def user_rated_books(userID):
    temp = subdf[subdf['User-ID'] == userID][['Book-Rating', 'Book-Title']]

    # picking only the books where user has rated 3 or above
    temp = temp[temp['Book-Rating'] >= 3]

    # if the user has not rated any book 3 or above
    if temp.empty:
        temp = temp[temp['Book-Rating']]

    # if the user has not rated any book at all
    if temp.empty:
        temp = temp[temp['Book-Rating']]

    return temp

def recommend_similar_books(book):
    temp_ratings = rating[book]
    books_like_temp = rating.corrwith(temp_ratings)
    corr_temp = pd.DataFrame(books_like_temp, columns=['Correlation'])
    corr_temp.dropna(inplace=True)
    return corr_temp.sort_values('Correlation', ascending=False).head(10)


def recommend_books_for_user(userID):
    book_df = pd.DataFrame()
    rated_book_list = user_rated_books(userID)

    if rated_book_list.empty:
        print("This user has no rated any book to correlate")
    elif len(rated_book_list) == 1:
        recommend_similar_books(rated_book_list['Book-Title'].values[0])
    else:
        if len(rated_book_list) > 3:
            rated_book_list = rated_book_list.sort_values('Book-Rating',
ascending=False)[:3]

        for x in rated_book_list['Book-Title'].values:
            book_df = book_df.append(recommend_similar_books(x))

        book_df['Book-Title'] = book_df.index
        book_df = book_df.drop_duplicates(subset='Book-Title', keep="first")
        return book_df.sort_values('Correlation', ascending=False)['Correlation'].head(10)
```

12. Below is an output from a test case. The idea behind this approach is that we pick books where the user has rated more than 3. This is an arbitrary value. You can set it for a much higher value if you want. Then we find books that have a very good correlation with that book based on the pivot table we produced before.



```
recommend_books_for_user(116866)
```

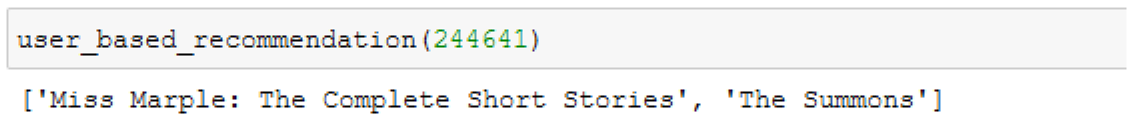
```
Book-Title
Statistical Process Control: Theory and Practice                                       1.000000
Granta 52: Food : The Vital Stuff                                                      1.000000
Applied Linear Statistical Models: Regression Analysis of Variance and Experimental Designs    1.000000
The Sword of Heaven: A Five Continent Odyssey to Save the World (Travelers' Tales)    -0.000096
The Hundredth Name                                                                    -0.000096
I Wish I Had a Red Dress                                                              -0.000096
Memoirs of an Invisible Man                                                           -0.000096
A Wild Justice: A Novel                                                               -0.000096
Crow Boy (Picture Puffins)                                                            -0.000096
Christmas Carol                                                                       -0.000096
Name: Correlation, dtype: float64
```

13. Next, I tried the user-based approach. Here again I pick some books that the give user has rated high. Then I find other users who have rated the same book similar or higher. Then I find similar or higher rated books for those users and combine them into a single list. There is a lot finer tuning needed for this phase, for a real-world implementation.

```
def find_similar_users(userID):
   temp = subdf[subdf['User-ID'] == userID][['User-ID', 'Book-Rating', 'Book-Title']]
   temp = temp.sort_values('Book-Rating', ascending=False).head(3)

   # picking only the books where user has rated 3 or above
   temp = temp[temp['Book-Rating'] >= 3]

   user_list = []

   # if the user has not rated any book 3 or above
   if len(temp) < 3:
      val = len(temp)
   else:
      val = 3

   if not temp.empty:
      for n in range(val):
         rating = temp['Book-Rating'].values[n]
         temp = subdf[subdf['Book-Title'] == temp['Book-Title'].values[0]]
```

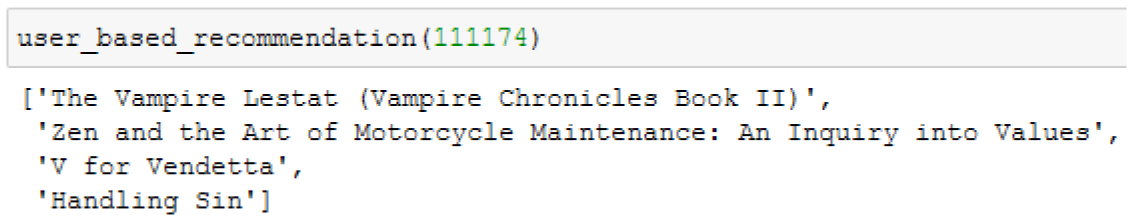
```
            temp = temp[temp['User-ID'] != userID]

            if not temp.empty:
                temp2 = temp[temp['Book-Rating'] == rating]
                if temp2.empty:
                    temp2 = temp[temp['Book-Rating'] > rating]

            if not temp2.empty:
                user_list.append(temp2['User-ID'][:3].values)
    return user_list[0]

def get_books_from_similar_users(user_list):
    book_list = []
    for x in user_list:
        temp = subdf[subdf['User-ID'] == x][['Book-Rating', 'Book-Title']]
        temp = temp.sort_values('Book-Rating', ascending=False)['Book-Title'].head(3)
        temp = list(temp.values)
        for n in temp:
            book_list.append(n)

    book_list = list(set(book_list))

    return book_list[:10]

def user_based_recommendation(userID):
    user_list = list(find_similar_users(userID))

    if user_list:
        return get_books_from_similar_users(user_list)
    else:
        print("Not enough information to make user based recommendations for this user"
```

14. Here are two test cases for the user-based approach.



```
user_based_recommendation(244641)
```

```
['Miss Marple: The Complete Short Stories', 'The Summons']
```



```
user_based_recommendation(111174)
```

```
['The Vampire Lestat (Vampire Chronicles Book II)',
 'Zen and the Art of Motorcycle Maintenance: An Inquiry into Values',
 'V for Vendetta',
 'Handling Sin']
```

15. I couldn’t do any analysis on the accuracy of these models. Using metrics such as precision and recall we can measure some amount of accuracy but eventually how successful your recommendations are going to depend on the end user input, which is the best way to measure your models.

Q & A

- Where is this data taken from ?
  Dataset was obtained from Kaggle. However, the source of data is book-crossing.com.

- Did you have enough resources to do all the required computations ?
  I could perform initial data exploration on the complete dataset with 900k+ records. However, vectorizing algorithms couldn’t be run on the complete dataset. In fact, I had to pick a very small subset to build the model.

- How did you select features ?
  Feature selection wasn’t that hard as the number of variables were small. Besides, this is a book recommendation system hence the features are almost always going to include book titles, authors and similar variables.

- What is the reason behind choosing these specific algorithms to build models ?

The two most common method to build recommendation systems are content based filtering and collaborative filtering which are what I used for my analysis.

- Which method resulted in better recommendations ?
  Ideally it should have been the Collaborative method as that is the generally preferred method. In my case, I got good results with content filtering but not so with collaborative filtering. This can be due to the small subset I had to pick or simply an error in my analysis.

- Do you always have to convert categorical variables to numeric ?
  Most ML algorithms will require you to do this as these are statistical models working with numbers. However, some models will include some encoding to achieve this while some algorithms like Naïve Bayes can directly use categorical features.

- Is it better to use more dependent features when building models ?
  Not always. Sometimes, adding more variables into the model will decrease the accuracy. This is something you have to figure out by trial and error.

- Is it easier to do this analysis using R ?
  It's a matter of taste. Some users will like the built-in features and packages in R while others who already have a programming background might prefer Python.

- How good are your system's recommendations ?
  It is difficult to say how good they are without any additional user feedback. If I had used the complete dataset, I would have had some data to compare with.

- How can you improve these models ?
  I need to figure out the reason why collaborative filtering didn't produce good results and work on that model. Also, I can test the models in a better resourced environment using the complete dataset.

Conclusion

Recommendation systems play a major role in online streaming, movies, retail, and almost any industry. Any e-commerce platform relies heavily on recommendations to increase their sales and in the case of streaming music and videos, to keep users coming back to the site. Book recommendation is no different. Every reader has their own taste and they would be happier if they can find books of similar type. In this analysis I used both content based filtering and collaborative filtering to recommend books to users. Content based method worked well with a single variable as well as with multiple. Collaborative method produced unpredictable results. This is an area that I have to improve.

References

- https://rpubs.com/mswofford/goordreads

  This is an analysis on book ratings from Goodreads which is a popular literary social media site that catalogs books, provides a platform for ratings and reviews, and helps users find their next book to read based on their interests and the recommendations of other users.

- https://techxplore.com/news/2018-09-behavior-goodreads-amazon-bestsellers.html

  This is an analysis of book reading behavior on Goodreads to predict Amazon Bestsellers.

- https://www.readkong.com/page/analyzing-social-book-reading-behavior-on-goodreads-and-how-5273289?p=2

  This is an analysis of Social Book Reading Behavior on Goodreads and how it predicts Amazon Best Sellers.

- https://www.kaggle.com/ammukp/bookreviews-visualization

  This is a visualization effort on the book ratings dataset found on Kaggle.

- https://www.kaggle.com/viktorpolevoi/book-crossing-data-preparation-viz

  This is a data preparation & visualization notebook on the book ratings dataset found on Kaggle.

- https://www.kaggle.com/chirantansayan/book-crossing-eda-and-recommender

  This is an effort to build a collaborative filter-based recommendation system for this book ratings dataset found on Kaggle.

- https://www.kaggle.com/akashdotcom/book-crossing-eda-and-recommendation-model-knn

  This is another effort to build a book recommendation system using KNN for this book ratings dataset found on Kaggle.

- https://www.kaggle.com/drfrank/book-review-ratings-data-analysis-visualization

This is a complete analysis on this book rating dataset that uses many different analytics techniques.

- https://towardsdatascience.com/building-a-content-based-book-recommendation-engine-9fd4d57a4da

  This is an effort to build a content-based book recommendation system.

- https://towardsdatascience.com/my-journey-to-building-book-recommendation-system-5ec959c41847

  This is another effort to build a book recommendation system using collaborative filtering.

Appendix I – Dataset

Dataset - https://www.kaggle.com/ruchi798/bookcrossing-dataset

There is no codebook. Below are the variables.

- Users File
  - User ID
  - Location
  - Age

- Books File
  - ISBN
  - Title
  - Author
  - Year of Publication
  - Publisher
  - Images

- Ratings File
  - User ID
  - ISBN
  - Rating

Appendix II – Complete Code

Importing Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import os
import numpy as np
import plotly.graph_objs as go
import plotly_express as px
from wordcloud import WordCloud,STOPWORDS
import re
import string
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

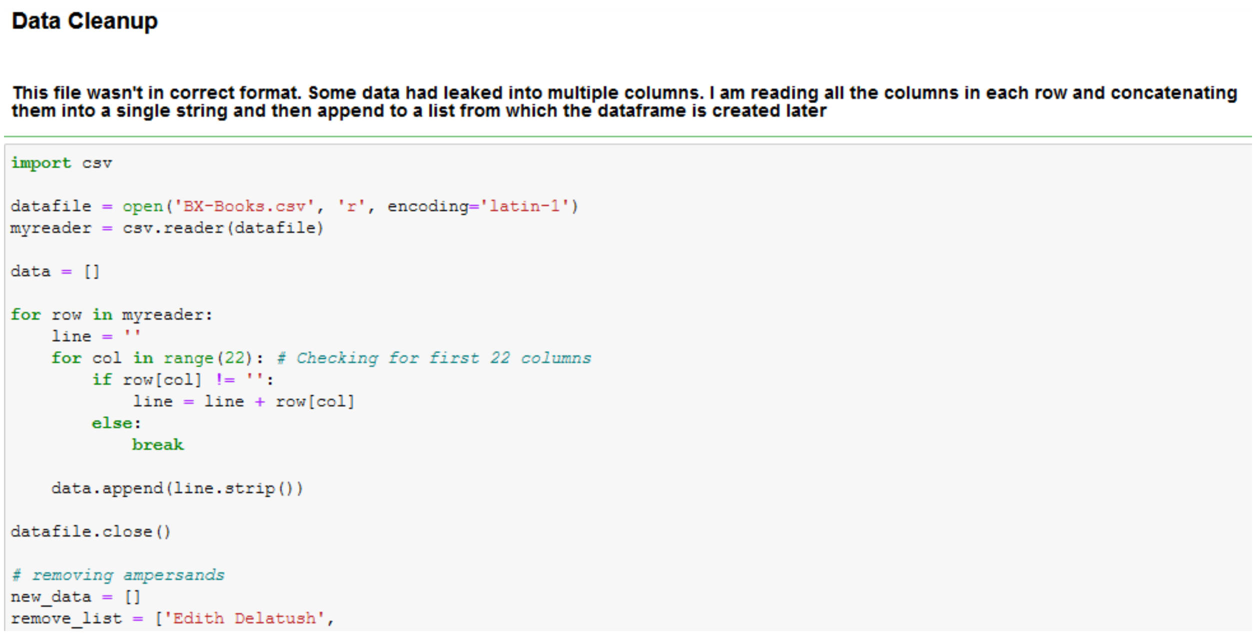
Loading Data

```
users = pd.read_csv('BX-Users.csv', sep=';', encoding='latin-1')
users.head()
```

	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

```
ratings = pd.read_csv('BX-Book-Ratings.csv', sep=';', encoding='latin-1')
ratings.head()
```

	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

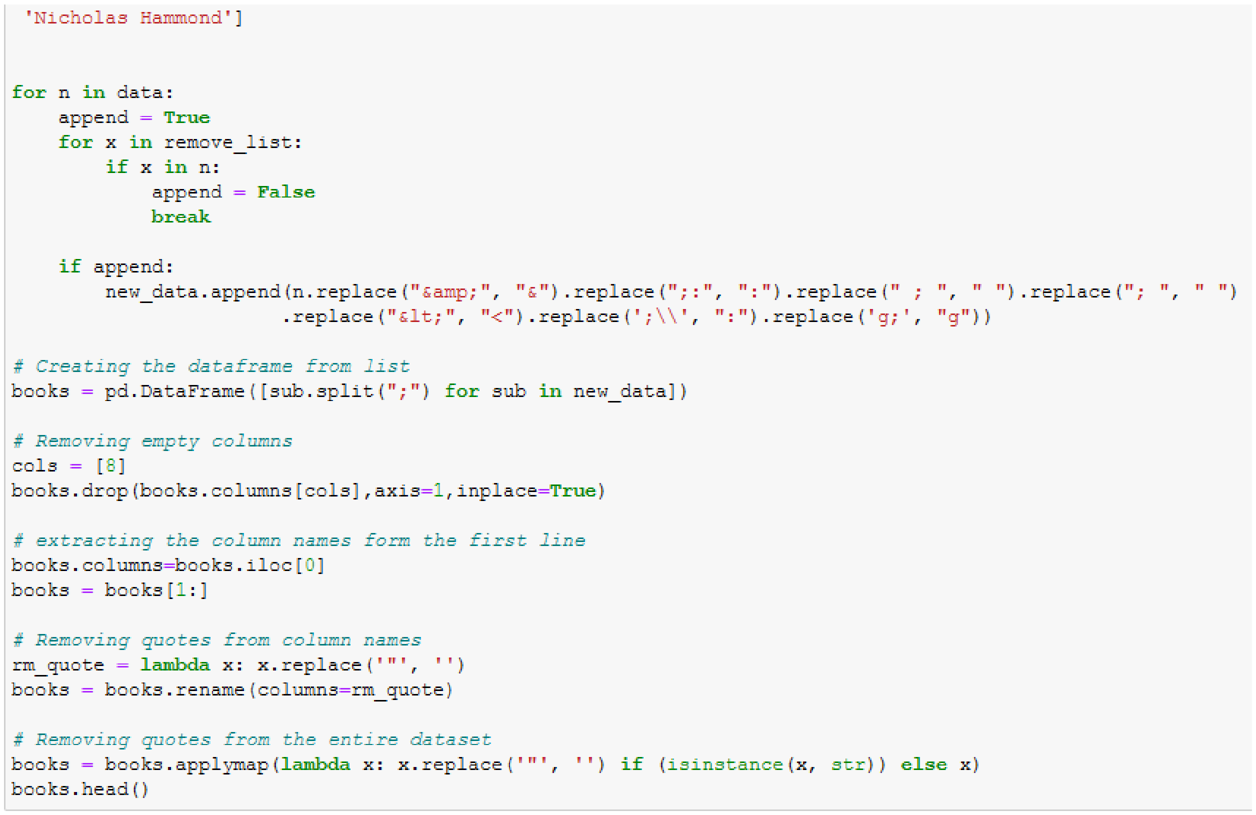


Data Cleanup

This file wasn't in correct format. Some data had leaked into multiple columns. I am reading all the columns in each row and concatenating them into a single string and then append to a list from which the dataframe is created later

```python
import csv

datafile = open('BX-Books.csv', 'r', encoding='latin-1')
myreader = csv.reader(datafile)

data = []

for row in myreader:
    line = ''
    for col in range(22): # Checking for first 22 columns
        if row[col] != '':
            line = line + row[col]
        else:
            break

    data.append(line.strip())

datafile.close()

# removing ampersands
new_data = []
remove_list = ['Edith Delatush',
```

…. Omitted complete list for brevity



```python
 'Nicholas Hammond']


for n in data:
    append = True
    for x in remove_list:
        if x in n:
            append = False
            break

    if append:
        new_data.append(n.replace("&amp;", "&").replace(";:", ":").replace(" ; ", " ").replace("; ", " ")
                       .replace("&lt;", "<").replace(';\\', ":").replace('g;', "g"))

# Creating the dataframe from list
books = pd.DataFrame([sub.split(";") for sub in new_data])

# Removing empty columns
cols = [8]
books.drop(books.columns[cols],axis=1,inplace=True)

# extracting the column names form the first line
books.columns=books.iloc[0]
books = books[1:]

# Removing quotes from column names
rm_quote = lambda x: x.replace('"', '')
books = books.rename(columns=rm_quote)

# Removing quotes from the entire dataset
books = books.applymap(lambda x: x.replace('"', '') if (isinstance(x, str)) else x)
books.head()
```

Checking for null values

```
users.isnull().sum()
```

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

```
# Replacing the null values in 'Age' column with the average
users['Age'].mean()
```

```
34.75143370454978
```

```
## Check if there is any user with 'Age' 34.5 so we can replace NaNs with a unique value
print(users[users['Age'].astype(str).str.contains('34.5')])
```

```
Empty DataFrame
Columns: [User-ID, Location, Age]
Index: []
```

```
# replacing
users['Age'] = users['Age'].fillna(34.5)
users.isnull().sum()
```

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

```
ratings.isnull().sum()
```

```
User-ID        0
ISBN           0
Book-Rating    0
dtype: int64
```

```
books.isnull().sum()
```

```
0
ISBN                   0
Book-Title             0
Book-Author            2
Year-Of-Publication    2
Publisher              2
Image-URL-S            2
Image-URL-M            2
Image-URL-L            2
dtype: int64
```

```
books[books['Book-Author'].isnull()]
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	Image-URL-L
84126	0553274740	You Are a Monster (Choose Your Own Adventure	None	None	None	None	None	None
236773	0824209141	Terrorism in the United States (Reference Shel...	None	None	None	None	None	None

```
books.drop(books.index[84125], inplace=True)
books[books['Book-Author'].isnull()]
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	Image-URL-L
236773	0824209141	Terrorism in the United States (Reference Shel...	None	None	None	None	None	None

```
books.drop(books.index[236771], inplace=True)
books[books['Book-Author'].isnull()]
```

ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	Image-URL-L

```
books.reset_index(inplace=True)
```

Merging the 3 datasets

```
df = pd.merge(users, ratings, on='User-ID', left_index=False)
df = pd.merge(df, books, on='ISBN', left_index=False, right_index=False)
del df['index']
df.head()
```

	User-ID	Location	Age	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	I
0	2	stockton, california, usa	18.0	0195153448	0	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.a /P/019
1	8	timmins, ontario, canada	34.5	0002005018	5	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.a /P/000

```
# Coverting to int
df['Year-Of-Publication'] = df['Year-Of-Publication'].astype('int64')
```

Data Exploration

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 995406 entries, 0 to 995405
Data columns (total 12 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   User-ID              995406 non-null  int64  
 1   Location             995406 non-null  object 
 2   Age                  995406 non-null  float64
 3   ISBN                 995406 non-null  object 
 4   Book-Rating          995406 non-null  int64  
 5   Book-Title           995406 non-null  object 
 6   Book-Author          995406 non-null  object 
 7   Year-Of-Publication  995406 non-null  int64  
 8   Publisher            995406 non-null  object 
 9   Image-URL-S          995406 non-null  object 
 10  Image-URL-M          995406 non-null  object 
 11  Image-URL-L          995406 non-null  object 
dtypes: float64(1), int64(3), object(8)
memory usage: 68.3+ MB
```

Picking a sample of 50k rows as the original dataset is too resource intensive to work with

```
df = df.sample(n = 50000)
```

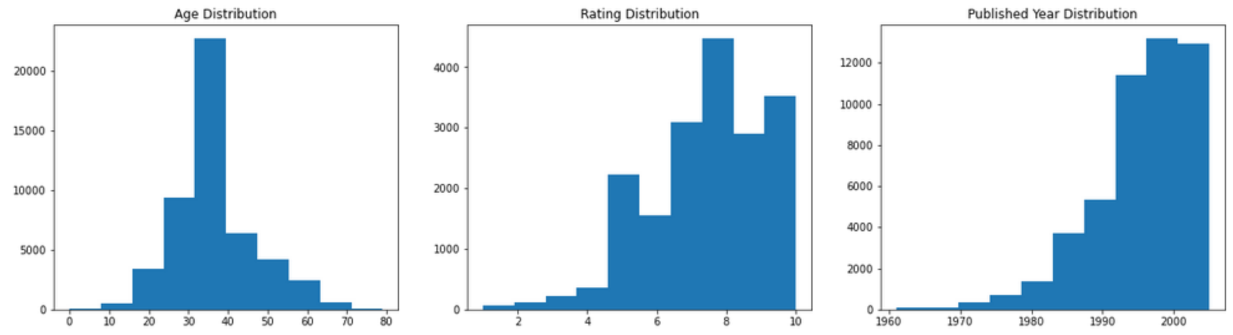
```
temp_Age = df[df['Age'] < 80]
temp_Book_Rating = df[df['Book-Rating'] > 0]
temp_YoP = df[df['Year-Of-Publication'] > 1960][df['Year-Of-Publication'] < 2010]
```

```
<ipython-input-22-153787c18270>:3: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  temp_YoP = df[df['Year-Of-Publication'] > 1960][df['Year-Of-Publication'] < 2010]
```

```
fig, axes = plt.subplots(nrows = 1, ncols = 3, figsize=(20, 5))

axes[0].hist(df[df['Age'] < 80]['Age'])
axes[0].set_title("Age Distribution")
axes[1].hist(df[df['Book-Rating'] > 0]['Book-Rating'])
axes[1].set_title("Rating Distribution")
axes[2].hist(df[df['Year-Of-Publication'] > 1960][df['Year-Of-Publication'] < 2010]['Year-Of-Publication'])
axes[2].set_title("Published Year Distribution")
```

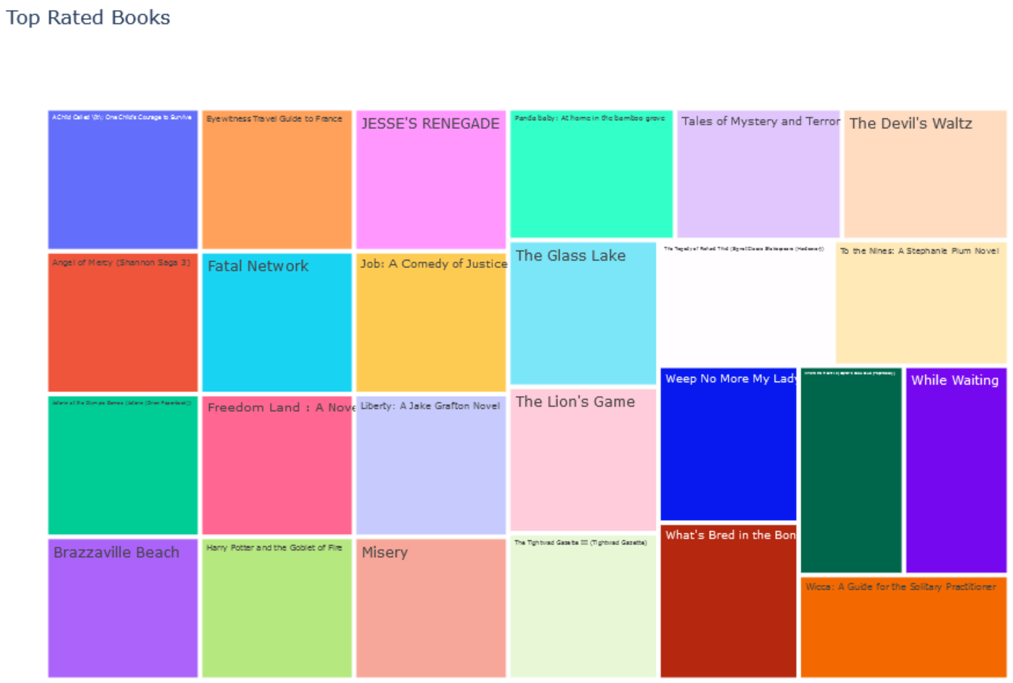
```
<ipython-input-23-34bd7ef8314a>:7: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  axes[2].hist(df[df['Year-Of-Publication'] > 1960][df['Year-Of-Publication'] < 2010]['Year-Of-Publication'])
```

```
Text(0.5, 1.0, 'Published Year Distribution')
```



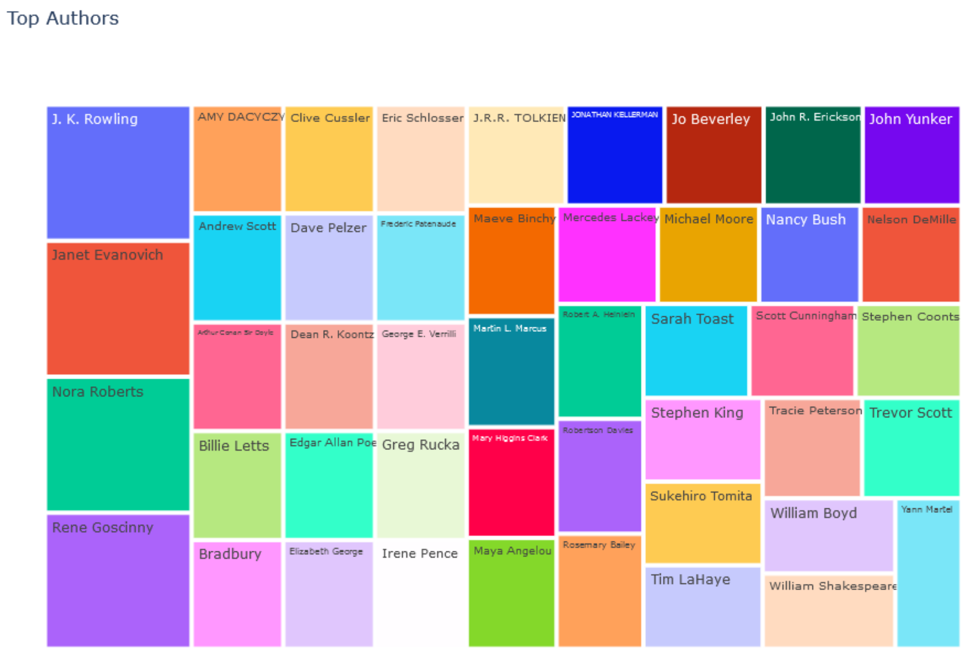
Finding top rated books

```python
top_rated = df.sort_values('Book-Rating', ascending=False)
tf_top_rated = top_rated[:25]
fig = px.treemap(tf_top_rated, path=['Book-Title'], values='Book-Rating',title='Top Rated Books', width=1000, height=700)
fig.show()
```



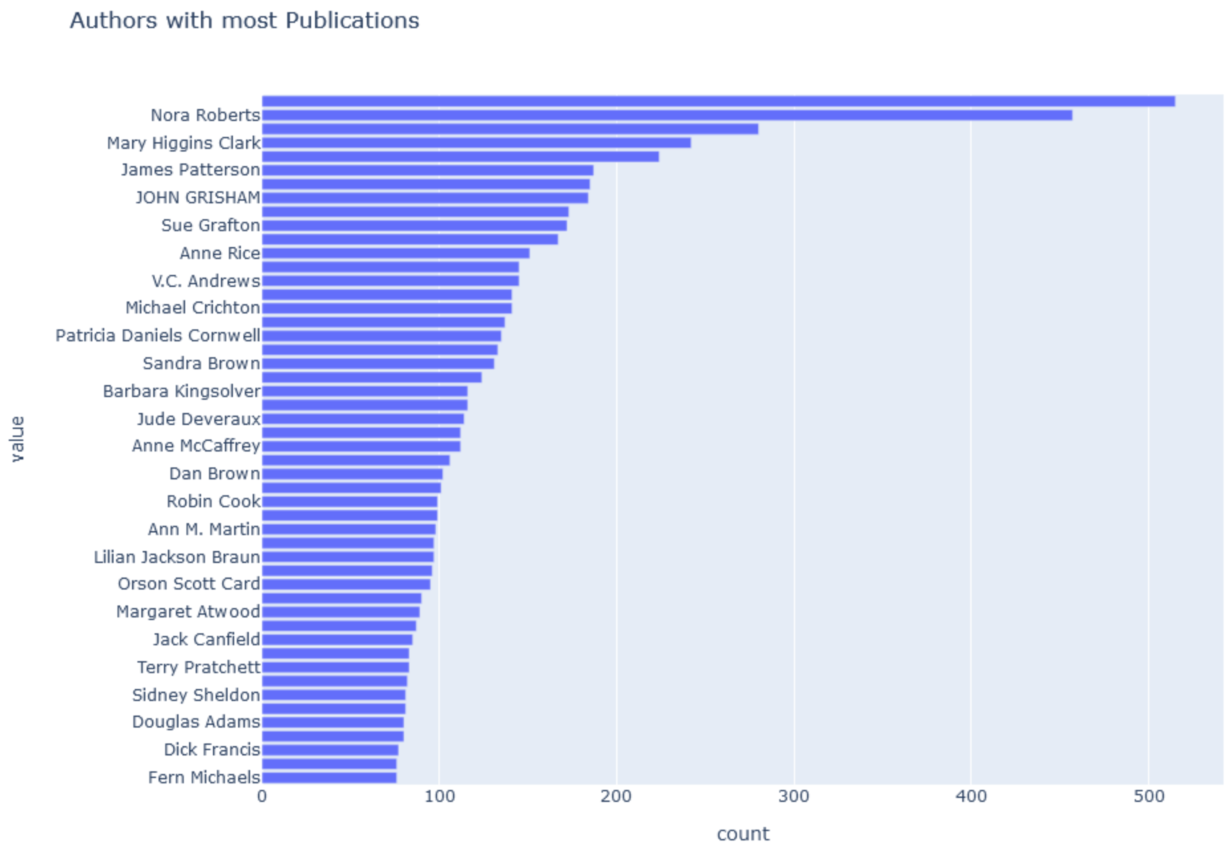
Finding top authors

```python
fifty_top_authors = top_rated[:50]
fig = px.treemap(fifty_top_authors, path=['Book-Author'], values='Book-Rating',title='Top Authors', width=1000, height=700)
fig.show()
```



Finding authors with most number of publications

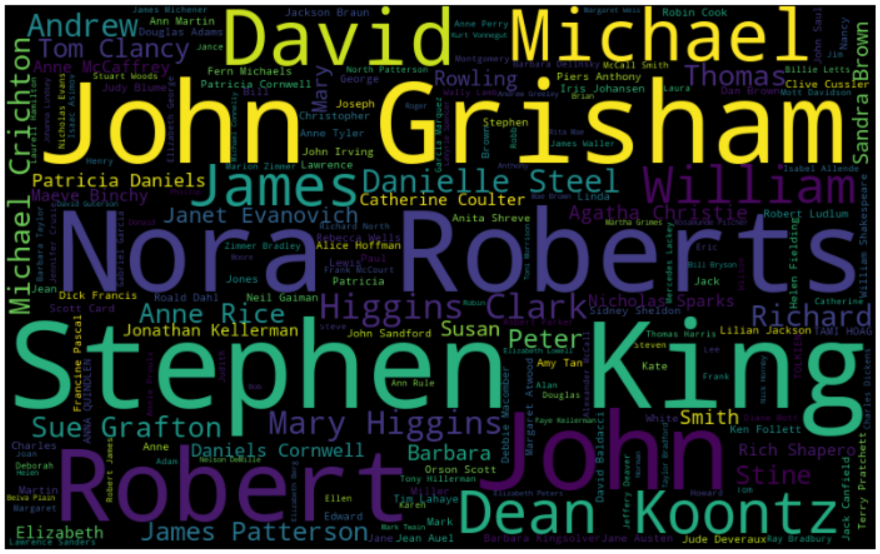
```
top_author_counts = df['Book-Author'].value_counts().reset_index()
top_author_counts.columns = ['value', 'count']
top_author_counts['value'] = top_author_counts['value']
top_author_counts = top_author_counts.sort_values('count')
fig = px.bar(top_author_counts.tail(50), x="count", y="value", title='Authors with most Publications',
             orientation='h', width=1000, height=700)
fig.show()
```



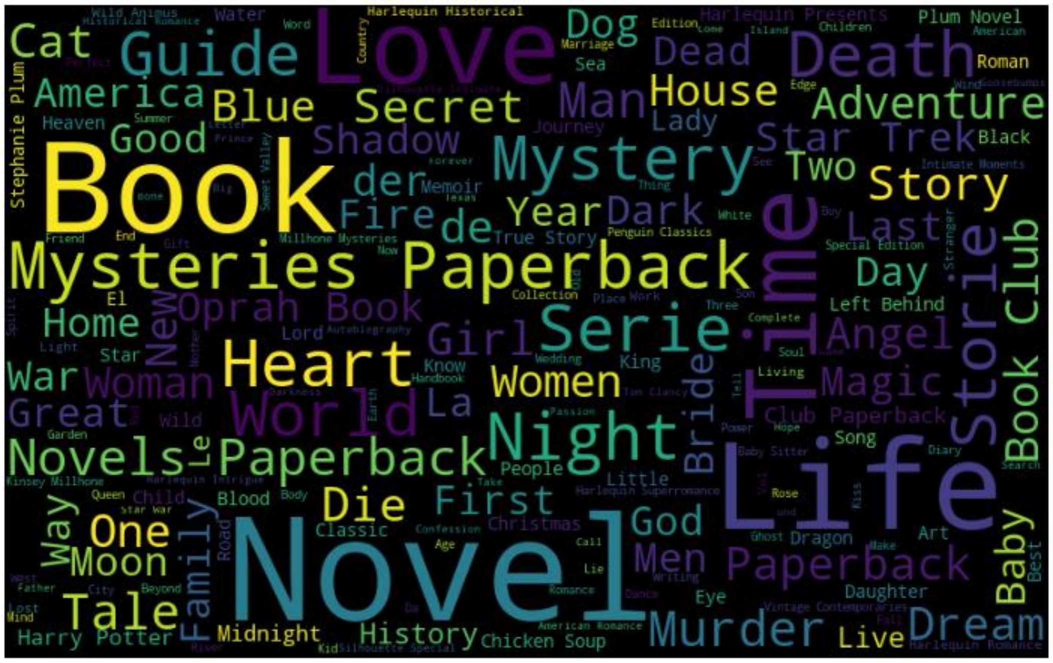
Wordclouds

```
stop_words=set(STOPWORDS)

def wordcloud(string):
    wc = WordCloud(width=800,height=500,mask=None,random_state=21, max_font_size=110,stopwords=stop_words).generate(string)
    fig=plt.figure(figsize=(16,8))
    plt.axis('off')
    plt.imshow(wc)
```

```
# word cloud for authors
authors = " ".join(df['Book-Author'])
wordcloud(authors)
```



```
# word cloud for books
books = " ".join(df['Book-Title'])
wordcloud(books)
```



Content Based Filtering

```
# picking an even smaller dataset as my PC cannot compute these models for a 50k dataset
subdf = df.sample(n = 2000)
```

```
content = subdf[['Book-Title','Book-Author','Book-Rating']]
content = content.astype(str)
content['content'] = content['Book-Title'] + ' ' + content['Book-Author'] + ' ' + content['Book-Rating']
content = content.reset_index()
indices = pd.Series(content.index, index=content['Book-Title'])
```

Recommendation based on author

```
tfidf = TfidfVectorizer(stop_words='english')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(content['Book-Author'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

```
(2000, 1917)
```

```
cosine_sim_author = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
def get_recommendations_books(title, cosine_sim=cosine_sim_author):
    idx = indices[title]

    # Get the pairwsie similarity scores of all books with that book
    sim_scores = list(enumerate(cosine_sim_author[idx]))

    # Sort the books based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar books
    sim_scores = sim_scores[1:11]

    # Get the book indices
    book_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar books
    return list(content['Book-Title'].iloc[book_indices])


def author_book_shows(book):
    for book in book:
        print(book)
```

```python
books = get_recommendations_books('The Prodigy', cosine_sim_author)
author_book_shows(books)
```

```
When the Sirens Wailed
The Simple Truth
DEVIL IN A BLUE DRESS (Easy Rawlins Mysteries (Paperback))
Anil's Ghost (Vintage International)
For the Sake of Elena
The Gentle Jungle
Widow: Rebuilding Your Life
Son of Web Pages That Suck: Learn Good Design by Looking at Bad Design
Digital Fortress : A Thriller
Trading Spaces Behind the Scenes: Including Decorating Tips and Tricks
```

```python
books = get_recommendations_books('For the Sake of Elena', cosine_sim_author)
author_book_shows(books)
```

```
Deception on His Mind
The Remarkable Women of the Bible Growth: And Their Message for Your Life Today
Deception on His Mind
Murder Makes Waves (Southern Sisters Mysteries (Paperback))
Mary Queen of Scotland & The Isles : A Novel
A House Somewhere:  Tales of Life Abroad
Radical robots: Can you be replaced? (A Novabook)
Shadow Moon (Chronicles of the Shadow War No 1)
Sand & Musset: Lettres d'amour
The Cable Car Murder
```

Recommendation based on multiple content

```python
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(content['content'])

cosine_sim_content = cosine_similarity(count_matrix, count_matrix)
```

```python
def get_recommendations(title, cosine_sim=cosine_sim_content):
    idx = indices[title]

    # Get the pairwsie similarity scores of all books with that book
    sim_scores = list(enumerate(cosine_sim_content[idx]))

    # Sort the books based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar books
    sim_scores = sim_scores[1:11]

    # Get the book indices
    book_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar books
    return list(content['Book-Title'].iloc[book_indices])

def book_shows(book):
    for book in book:
        print(book)
```

```python
books = get_recommendations('The Prodigy', cosine_sim_content)
book_shows(books)
```

```
When the Sirens Wailed
The Simple Truth
DEVIL IN A BLUE DRESS (Easy Rawlins Mysteries (Paperback))
Anil's Ghost (Vintage International)
For the Sake of Elena
The Gentle Jungle
Widow: Rebuilding Your Life
Son of Web Pages That Suck: Learn Good Design by Looking at Bad Design
Digital Fortress : A Thriller
Trading Spaces Behind the Scenes: Including Decorating Tips and Tricks
```

```python
books = get_recommendations('For the Sake of Elena', cosine_sim_content)
book_shows(books)
```

```
Deception on His Mind
Deception on His Mind
The Remarkable Women of the Bible Growth: And Their Message for Your Life Today
Almost: A Novel
Winter Fire
Watership Down
Museum Pieces
Moving Target
Destinations South
Too Hot To Handle
```

Collaborative Filtering

Since most of the rating values are 0 which can affect our filtering model, I am going to remove 0 ratings from the original 'ratings' data frame and re join with other columns

```
nonzero_ratings = ratings[ratings['Book-Rating'] != 0]
df = pd.merge(users, nonzero_ratings, on='User-ID', left_index=False)
df = pd.merge(df, books, on='ISBN', left_index=False, right_index=False)
del df['index']
df.head()
```

	User-ID	Location	Age	ISBN	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M	Ima
0	8	timmins, ontario, canada	34.5	0002005018	5	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.am /P/00020
1	11676	n/a, n/a, n/a	34.5	0002005018	8	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.am /P/00020
2	67544	toronto, ontario, canada	30.0	0002005018	8	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.am /P/00020
3	116866	ottawa, ,	34.5	0002005018	9	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.am /P/00020
4	123629	kingston, ontario, canada	34.5	0002005018	9	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.am /P/00020

```
# using 20k records from the list
subdf = df.sample(n = 20000)
rating = subdf.pivot_table(index='User-ID', values='Book-Rating', columns='Book-Title').fillna(0)
rating.head()
```

Book-Title	Earth Prayers From around the World: 365 Prayers Poems and Invocations for Honoring the Earth	!Yo!	$oft Money: The True Power in Our Nation's Capital	'Round the Corner (Sister Circle)	'Salem's Lot	01-01-00: A Novel of the Millennium	08/15 Heute	10 Lb. Penalty	10 Reasons to Abolish the Imf & World Bank (Open Media Pamphlet Series)	100 Carols for Choirs	...	how to stop time : heroin from A to Z	il Paradiso Degli Orchi	stardust	thepurplebook: The Definitive Guide to Exceptional Online Shopping	why I'm like this : True Stories	Â¿Quis llevad que
User-ID																	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
56	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
85	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
92	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

5 rows × 14717 columns

Item based approach

I am using one book as an example to test the collaborative method where we recommend books to a user based on other books like the one he has rated

```python
def user_rated_books(userID):
    temp = subdf[subdf['User-ID'] == userID][['Book-Rating', 'Book-Title']]

    # picking only the books where user has rated 3 or above
    temp = temp[temp['Book-Rating'] >= 3]

    # if the user has not rated any book 3 or above
    if temp.empty:
        temp = temp[temp['Book-Rating']]

    # if the user has not rated any book at all
    if temp.empty:
        temp = temp[temp['Book-Rating']]

    return temp

def recommend_similar_books(book):
    temp_ratings = rating[book]
    books_like_temp = rating.corrwith(temp_ratings)
    corr_temp = pd.DataFrame(books_like_temp, columns=['Correlation'])
    corr_temp.dropna(inplace=True)
    return corr_temp.sort_values('Correlation', ascending=False).head(10)

def recommend_books_for_user(userID):
    book_df = pd.DataFrame()
    rated_book_list = user_rated_books(userID)

    if rated_book_list.empty:
        print("This user has no rated any book to correlate")
    elif len(rated_book_list) == 1:
        recommend_similar_books(rated_book_list['Book-Title'].values[0])
    else:
        if len(rated_book_list) > 3:
            rated_book_list = rated_book_list.sort_values('Book-Rating', ascending=False)[:3]

        for x in rated_book_list['Book-Title'].values:
            book_df = book_df.append(recommend_similar_books(x))

        book_df['Book-Title'] = book_df.index
        book_df = book_df.drop_duplicates(subset='Book-Title', keep="first")
        return book_df.sort_values('Correlation', ascending=False)['Correlation'].head(10)
```

```
recommend_books_for_user(116866)
```

```
Book-Title
Statistical Process Control: Theory and Practice                                  1.000000
Granta 52: Food : The Vital Stuff                                                 1.000000
Applied Linear Statistical Models: Regression Analysis of Variance and Experimental Designs    1.000000
The Sword of Heaven: A Five Continent Odyssey to Save the World (Travelers' Tales)    -0.000096
The Hundredth Name                                                               -0.000096
I Wish I Had a Red Dress                                                         -0.000096
Memoirs of an Invisible Man                                                      -0.000096
A Wild Justice: A Novel                                                          -0.000096
Crow Boy (Picture Puffins)                                                       -0.000096
Christmas Carol                                                                  -0.000096
Name: Correlation, dtype: float64
```

```
recommend_books_for_user(98391)
```

```
Book-Title
Sea of Bones                                                                 1.0
Dissolution                                                                  1.0
The Duet: A Novel                                                            1.0
The Fall of Light                                                            1.0
The Forever Kiss                                                             1.0
The Fort at River's Bend : Book Five of The Camulod Chronicles (Camulod Chronicles)    1.0
True North                                                                   1.0
Lady in Waiting                                                              1.0
The Good House : A Novel                                                     1.0
Killer Blonde                                                                1.0
Name: Correlation, dtype: float64
```

User based approach

```python
def find_similar_users(userID):
    temp = subdf[subdf['User-ID'] == userID][['User-ID', 'Book-Rating', 'Book-Title']]
    temp = temp.sort_values('Book-Rating', ascending=False).head(3)

    # picking only the books where user has rated 3 or above
    temp = temp[temp['Book-Rating'] >= 3]

    user_list = []

    # if the user has not rated any book 3 or above
    if len(temp) < 3:
        val = len(temp)
    else:
        val = 3

    if not temp.empty:
        for n in range(val):
            rating = temp['Book-Rating'].values[n]
            temp = subdf[subdf['Book-Title'] == temp['Book-Title'].values[0]]
            temp = temp[temp['User-ID'] != userID]

            if not temp.empty:
                temp2 = temp[temp['Book-Rating'] == rating]
                if temp2.empty:
                    temp2 = temp[temp['Book-Rating'] > rating]

            if not temp2.empty:
                user_list.append(temp2['User-ID'][:3].values)
    return user_list[0]

def get_books_from_similar_users(user_list):
    book_list = []
    for x in user_list:
        temp = subdf[subdf['User-ID'] == x][['Book-Rating', 'Book-Title']]
        temp = temp.sort_values('Book-Rating', ascending=False)['Book-Title'].head(3)
        temp = list(temp.values)
        for n in temp:
            book_list.append(n)

    book_list = list(set(book_list))

    return book_list[:10]
```

```python
def user_based_recommendation(userID):
    user_list = list(find_similar_users(userID))

    if user_list:
        return get_books_from_similar_users(user_list)
    else:
        print("Not enough information to make user based recommendations for this user")
```

```python
user_based_recommendation(244641)
```

```
['Miss Marple: The Complete Short Stories', 'The Summons']
```

```python
user_based_recommendation(111174)
```

```
['The Vampire Lestat (Vampire Chronicles Book II)',
 'Zen and the Art of Motorcycle Maintenance: An Inquiry into Values',
 'V for Vendetta',
 'Handling Sin']
```

For testing purposes we can use below query on the data frame to find book titles that have been rated by more than one user. This is useful since most books have zero ratings.

subdf[subdf.duplicated(subset='Book-Title', keep='first')]