GOODREADS: AN ANALYSIS

Reading has been known to be quite important in developing the mind; to say the least, it improves brain connectivity and a person's communication skills. Many school education systems even implement some sort of reading into their programs. However, just because it's a part of healthy development and some school curricula does not mean that reading books is a chore and cannot be enjoyable.

This analysis focuses on classifying some books as good or bad, so that not only can people enjoy reading but also publishing companies can look at what might be profitable.

DATA

The analysis done is based on the below dataset from Kaggle. This dataset itself is based on a Goodreads list of best books ever, voted by the users of Goodreads. The list is updated every week and was scraped in 2019.
https://www.goodreads.com/list/show/1.Best_Books_Ever
https://www.kaggle.com/meetnaren/goodreads-best-books

METHOD

This is a supervised learning problem, where I sorted the dataset by its ratings and labeled the top half of the group as 'good' and the bottom half as 'bad.' Using this to train the models, I built and tuned each model, and evaluated its performance against both each other and a dummy classifier model.

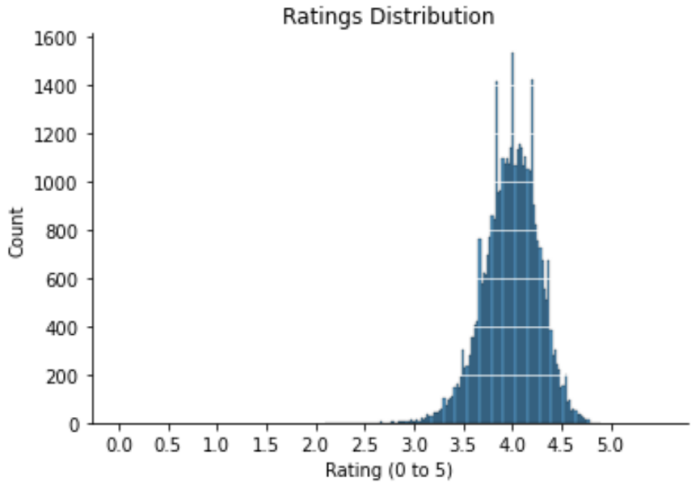
DATA CLEANING (AND FEATURE ENGINEERING)

- **Problem 1:** Missing values. There were quite a lot missing in the following features: edition, few in description, format, but importantly some were missing genres and pages.
    - Solution: Remove the ones with missing genres and pages since those are the primary features I'll be focusing on. Remove the ones with more than 3 missing features. The rest of the missing values will be filled in as 'missing' so it's not just empty.
    - Constraints: I would be losing the information from the books with a lot of missing values, which could be important. Additionally, since I manually decided to focus on genres and pages, perhaps the model will not efficiently learn which features are the best by itself.
    - Results: About 5k books were deleted, with still a good amount remaining.
- **Problem 2:** Duplicates (same books, different versions/edition, or just exact duplicates)
    - Solution: Keep only one version of the books, the one with the most amount of ratings.
    - Constraints: Loss of information since the duplicate or different edition book data is completely removed. Although there is no guarantee that the people who reviewed those duplicated entries are different people, in hindsight, it might've been better to average out the duplicated values such as ratings and perhaps summing the ratings/review counts.

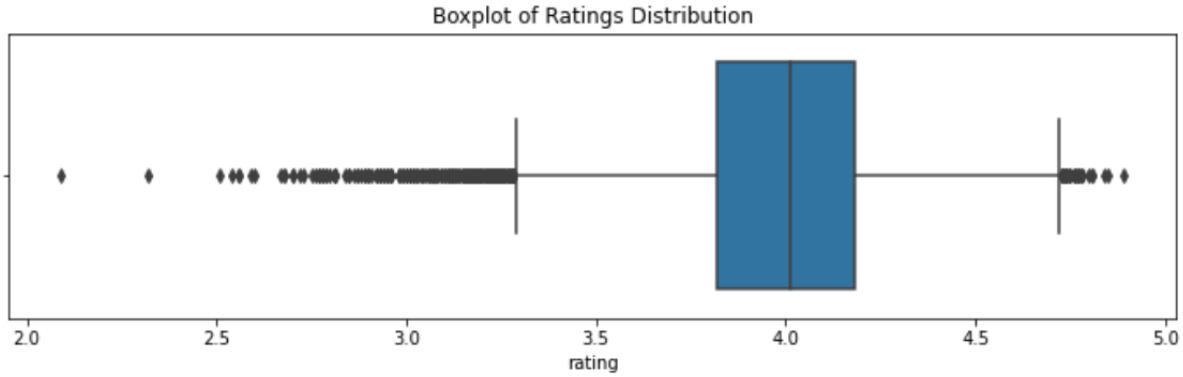
- **Problem 3:** Illogical data, including books with 0 pages or 0 ratings. Since some books were categorized as audiobooks, they naturally had 0 pages.
    - Solution: Since ratings is my main consideration of whether a book is considered 'good' or 'bad,' I kept only the books with 100 ratings. I considered 100 to be a reasonable minimum given that the median of the ratings count is 3763. The mean wasn't considered since some book pages are disproportionately large, skewing the mean to the right.
    - Constraints: There could be some information loss due to my primary focus on the book pages and ratings.
- **Problem 4:** User input format and editions created all different formats. For example: if a book was a first edition, it could be written as "First", "First Edition", "1st", etc.
    - Solution: Separate all the edition values into two categories: 1) 'First' by filtering all the editions that include the words "first" or "1"; and 2) 'Other' for all the other editions. Only the first edition was separated out because the top 4 editions were all first editions, making up a majority of the non-missing values. In order to not lose too much information from missing ones, I also added a binary feature to designate whether the edition feature was missing.
    - Constraints: Because these words were filtered through a manually created RegEx filter, some cases could have slipped through and been mistakenly added to the 'other' category.
- **Problem 5:** Sets of books (either joint works of the author, or a book series) were also included in the dataset. Since I'd like to look at only individual books, this was problematic.
    - Solution: Sets of books were dropped to the best of my abilities; any format, edition, title that included the words Box(ed), Chronicles, Sets, Collections in the title were dropped after making sure they were indeed sets of books.
    - Constraints: Because the titles, editions, formats had to be looked at individually or filtered through, there may have been books which actually were sets but not correctly identified, feeding the models erroneous inputs.
- **Problem 6:** Numerous genres, most of them holding little meaning since there would only be one book within this genre in the entire dataset.
    - Solution: Obtain the top 15 genres, and categorize the rest as 'Other'.
    - Constraints: Choosing 15 to be the threshold was a manual choice, which could present to be problematic in modeling since it may not be the optimal number.
- **Problem 7:** Different languages
    - Solution: None. For now, I've left the different languages as-is in the dataset to see how the models will do, but acknowledge this could be a potential issue.

EDA

The Ratings



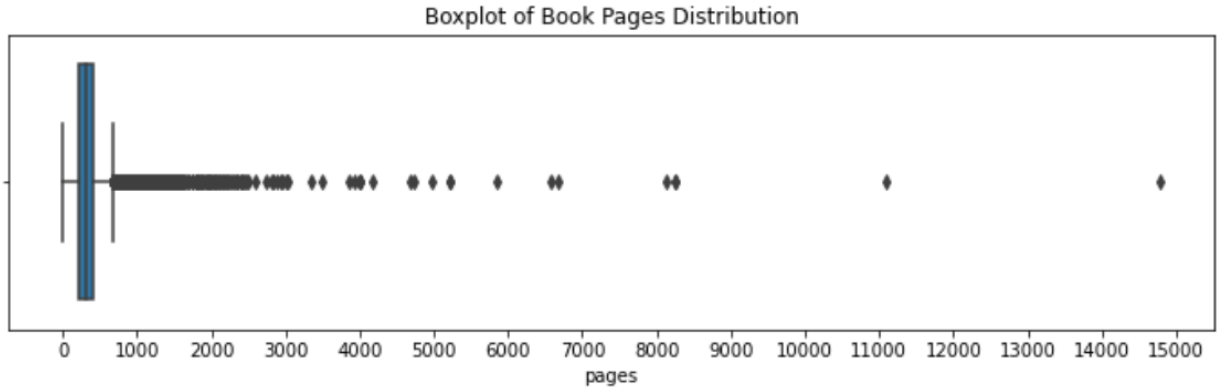

```
count    38416.000000
mean         3.991046
std          0.275888
min          2.090000
25%          3.820000
50%          4.010000
75%          4.180000
max          4.890000
Name: rating, dtype: float64
```



The ratings distribution looks relatively normal, centered around 4.0, but it is not without outliers.

The Book Pages

I acknowledge that book pages is an ambiguous number since the same novel with the same word count could have different page numbers based on the publisher, the size of the book, any illustrations it might have, etc.

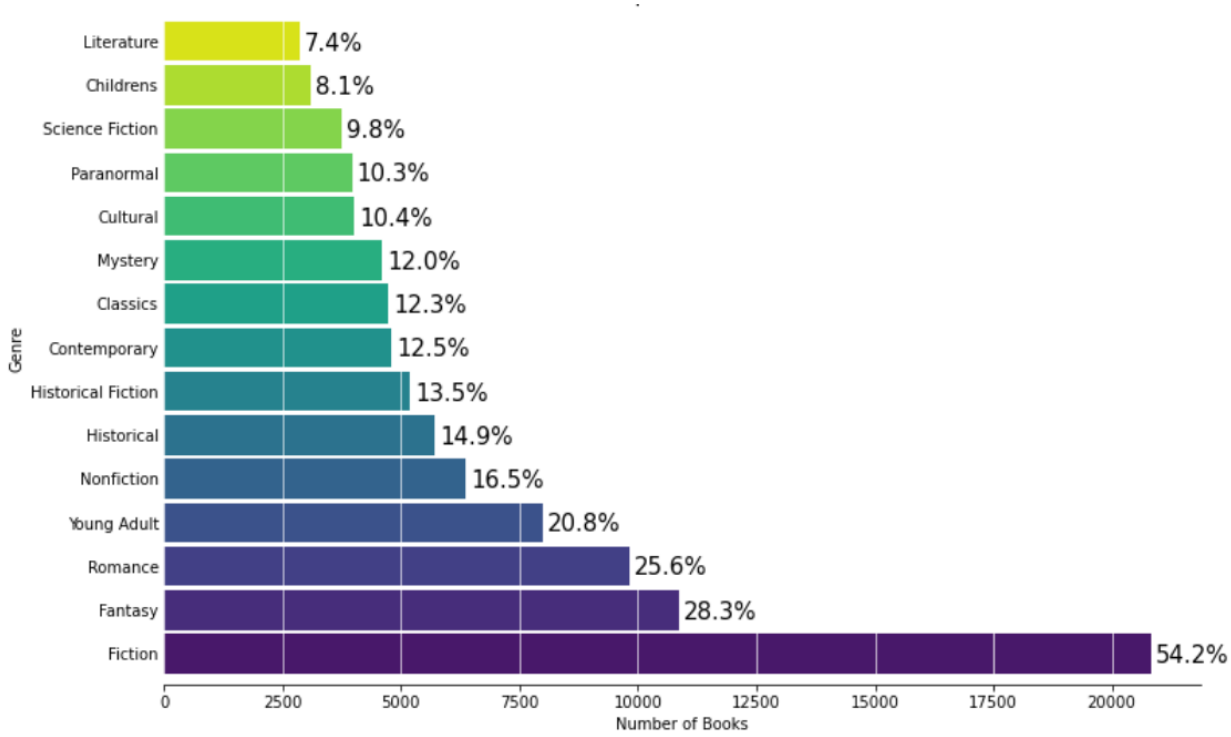


```
count    38416.000000
mean       342.581346
std        244.489347
min          0.000000
25%        226.000000
50%        320.000000
75%        404.000000
max      14777.000000
Name: pages, dtype: float64
```

The book pages are expectedly skewed to the right; half of the books had fewer than 320 pages. However, there were many outliers on the higher end, with a book going as high as 14,777 pages. Roughly half the books' pages lie between 227 and 403 pages. According to XYZ, that would take the average reader about 6.3 to 11.2 hours to read.

Genres

Many of the genres in the original dataset appear only once, and would not have much of a meaning/impact in the model due to its scarcity. Because of this, I focused on only the top 15 genres and grouped the rest together into a "Other" category.



Fiction reigned the dataset, making up a whopping 54.2% of the entire dataset. At almost half, fantasy was second, followed by romance. Keeping in mind that books may be of multiple genres, a majority of books (91.63%) consisted of a genre within the top 15 *and* a genre that's not listed in the top 15.

Looking at the average ratings in each genre, historical fiction is the top genre with a 4.05872 average rating. The average rating of the #1 genre, fiction, is in the middle in 7th out of the top 15 genres, at 3.97937.

Authors

Sometimes authors are simply great writers, and most of their works become successful. As such, it could be important to explore the authors' information.

Some authors seem to have really great average ratings such as Jo-Anne McArthur with a 4.84 rating. Upon closer look, however, some have written only one book. As such, it can't be determined whether they are indeed great writers or if they were lucky. Exploring authors who've written more than three books, the best average rated author in the dataset is Bill Watterson with 4.693571. This is expected since the author of the Calvin and Hobbes comics!

There are also authors in the dataset with numerous works. Reigning with 146 works, legendary Stephen King is at the top of that list, followed by the well-known detective author Agatha Christie.

FEATURE ENGINEERING

Authors

Multitudes of books have multiple authors, leading to a list of authors as feature. This would cause the same problem as the original genre values, as there's a lot of features with little value to the model due to their scarcity. As such, I was not going to use the authors' names as a feature in my model. However, I still wanted to feed some of that information to the model, so I created the following features for the number of authors a book has:

1. Number of authors
2. Binary feature of whether the book has more than one author (0 for no, 1 for yes)
3. Binary feature of whether the book has more than three authors (0 for no, 1 for yes)
4. Binary feature of whether the book has more than five authors (0 for no, 1 for yes)
5. Binary feature of whether the book has more than ten authors (0 for no, 1 for yes)

Average Author Rating

In this portion, I make the big assumption that the first author listed is the primary and most important author. As such, I created four features to capture some information about the authors:

1. First author's average book rating
2. First author's number of works
3. Total authors' average book ratings
4. Total authors' numbers of books

MODELS

After cleaning and feature engineering, I sorted the data by its ratings and created my dependent variable by splitting the top half (better rated half) as 'good' and the bottom half as 'bad.' The ratings feature was then removed from the dataset before any modeling.

Since my initial models were not performing well, I tried various others as well. Although some of them didn't necessarily need scaling of the data, I used a Min-Max scaler anyway to see if it would make a difference due to some models' poor performances.

Baseline

A baseline was created with a dummy classifier that generates predictions uniformly at random. Expectedly, its accuracy was about 50%.

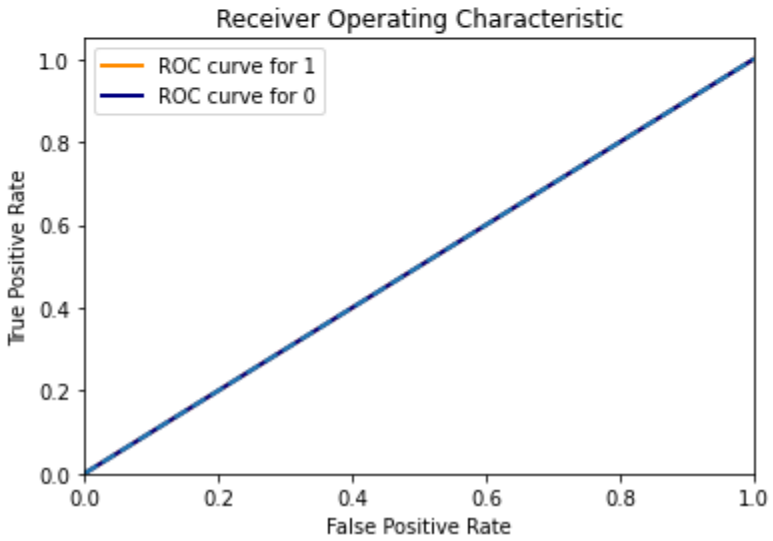
```
DummyClassifier(random_state=42, strategy='uniform')
Model Fit Time:  0.0019948482513427734 s
Model Prediction Time:  0.0009963512420654297 s

              precision    recall  f1-score   support

         0.0       0.51      0.51      0.51      5766
         1.0       0.50      0.50      0.50      5759

    accuracy                           0.50     11525
   macro avg       0.50      0.50      0.50     11525
weighted avg       0.50      0.50      0.50     11525

[[2914 2852]
 [2855 2904]]
AUC for classifying as good (1): 0.5
AUC for classifying as bad (0): 0.5.
```



Logistic Regression

The first model I tested was a logistic regression, tuning some of its parameters using GridSearchCV.

Interestingly enough, running the same GridSearchCV of the logistic regression model on the scaled data resulted in the same results as the dummy classifier. Logistic regression on the as-is data resulted in a minimally better test RMSE. However, its accuracy for 'bad' books was not as great.

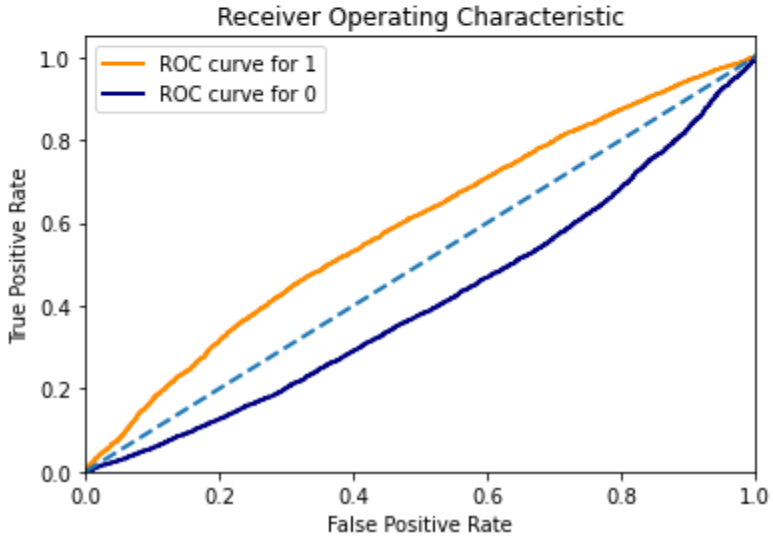
```
LogisticRegression(C=100.0, multi_class='ovr')
Model Fit Time:  0.3331129550933838 s
Model Prediction Time:  0.002985715866088867 s

              precision    recall  f1-score   support

         0.0       0.56      0.57      0.57      5766
         1.0       0.56      0.56      0.56      5759

    accuracy                           0.56     11525
   macro avg       0.56      0.56      0.56     11525
weighted avg       0.56      0.56      0.56     11525

[[3276 2490]
 [2525 3234]]
AUC for classifying as good (1): 0.5905990274041801
AUC for classifying as bad (0): 0.40940097259582.
```



Random Forest

I then tried to fit a random forest classifier, hoping this would perform better than the dummy classifier and logistic regression. However, it actually did worse. Since random forests are scale-invariant, the scaled data did not make much of a difference.

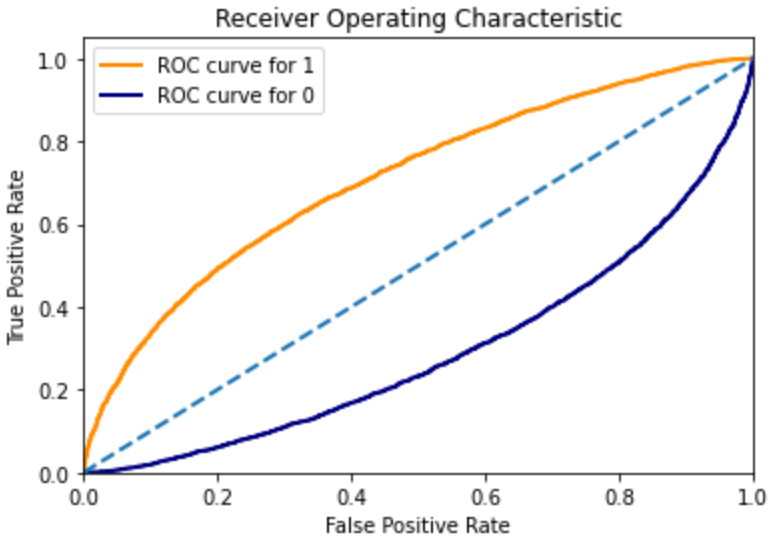
```
RandomForestClassifier(criterion='entropy', max_depth=25, n_estimators=500,
                       random_state=42)
Model Fit Time:  56.12944531440735 s
Model Prediction Time:  3.57208514213562 s

              precision    recall  f1-score   support

         0.0       0.64      0.68      0.66      5766
         1.0       0.66      0.62      0.64      5759

    accuracy                           0.65     11525
   macro avg       0.65      0.65      0.65     11525
weighted avg       0.65      0.65      0.65     11525

[[3940 1826]
 [2200 3559]]
AUC for classifying as good (1): 0.7077095905083823
AUC for classifying as bad (0): 0.2922904094916178.
```



ADA Boost

Next I tried using a Ada Boost Classifier, tuning it once again with a GridSearchCV. Since ADA Boost is also scale-invariant, the scaled data did not make much of a difference in its performance. Also surprisingly, its performance was worse than the dummy classifier.

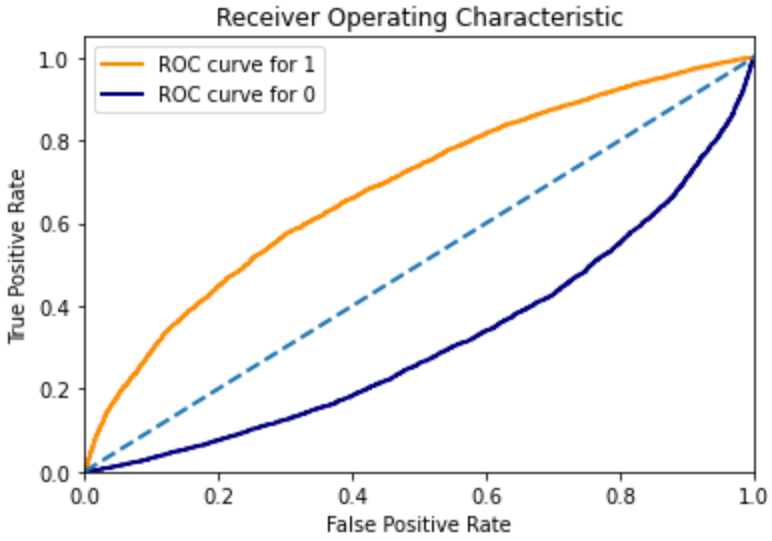
```
AdaBoostClassifier(learning_rate=0.3, n_estimators=200, random_state=42)
Model Fit Time:  8.274667739868164 s
Model Prediction Time:  0.6522541046142578 s

              precision    recall  f1-score   support

         0.0       0.63      0.66      0.64      5766
         1.0       0.64      0.60      0.62      5759

    accuracy                           0.63     11525
   macro avg       0.63      0.63      0.63     11525
weighted avg       0.63      0.63      0.63     11525

[[3813 1953]
 [2276 3483]]
AUC for classifying as good (1): 0.6829056777438707
AUC for classifying as bad (0): 0.3170943222561293.
```



KNN

None of the above models were performing as I would have liked, so I tried a KNN model. Since this exploits similarities, I used the scaled data. In order to decrease the computational time, I tuned the model with a RandomizedSearchCV as opposed to a GridSearchCV. Given that there is generally minimal differences in results between the two, I opted for the lower computational time.

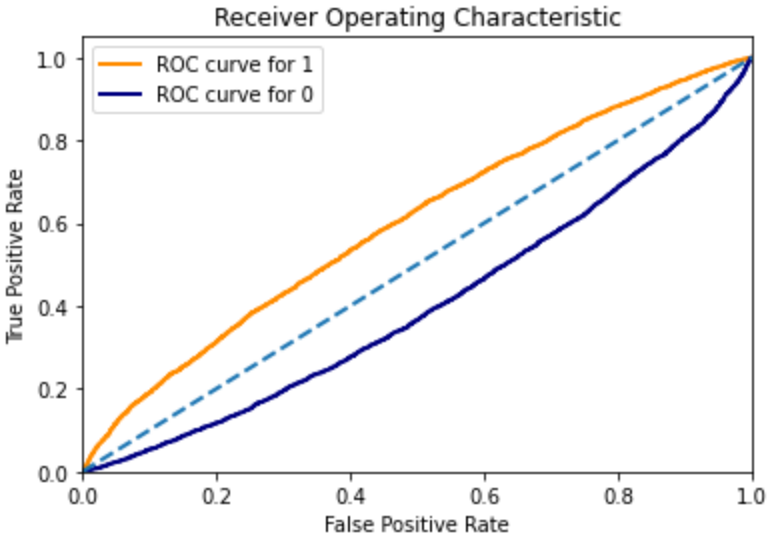
```
KNeighborsClassifier(leaf_size=50, n_neighbors=50, weights='distance')
Model Fit Time:  1.7871971130371094 s
Model Prediction Time:  23.68520498275757 s

              precision    recall  f1-score   support

         0.0       0.57      0.57      0.57      5766
         1.0       0.57      0.57      0.57      5759

    accuracy                           0.57     11525
   macro avg       0.57      0.57      0.57     11525
weighted avg       0.57      0.57      0.57     11525

[[3262 2504]
 [2476 3283]]
AUC for classifying as good (1): 0.5982563779734711
AUC for classifying as bad (0): 0.401743622026529.
```



MODEL SUMMARY AND CONCLUSIONS

Model	F-1 Score [0, 1]	Prediction Time (s)	Fit Time (s)
Baseline	0.51, 0.50	0.001995	0.000996
Logistic Regression	0.00, 0.67	0.333113	0.002986
Random Forest	0.66, 0.64	56.12945	3.572085
Ada Boost	0.64, 0.62	8.274668	0.652254
KNN	0.60, 0.57	1.787197	23.685205

Note: 0 is denoted to be a ‘bad’ book while 1 is denoted to be a ‘good’ book. Without taking into account the dummy classifier, the best performing score in each category is highlighted.

From the above, it looks like the logistic regression is classifying all the books as good books, which wouldn't make a useful model.

Although none of the models performed well enough to really be implemented, the random forest model seems to be the best given its F-1 score. Its higher precision and recall could better predict which books are actually good. The random forest, however, has the longest fit execution time; hopefully this isn't a strong issue since the prediction execution time is considerably lower although it's not one of the lowest. In the end, more work is needed on these models to improve their performances since they are only marginally better than the dummy classifier.

FUTURE IMPROVEMENTS

Because the current models are not performing well, additional features could be added to improve the model. For example, NLP could be applied to the books' descriptions feature in the original dataset and be used in the model. Since it would be NLP, there would need to be a lot of data cleaning. Computer vision could be used to analyze the book cover feature in the original dataset as well.

The dataset could be further expanded to add revenues for each book, to see whether the model's good or bad approach truly translates well into a profitable scenario.

The initial data could also be better cleaned, such as scaling the input data (of at least the book pages) with tanh before normalization so the numbers are cleaner for the models to use; this will better deal with the outliers as well. Or remove fewer entries when dealing with missing features.

An extension and separate project could be a book recommendation system, using both the original features of the dataset but also the predicted values of good or bad books.