Sabrina Alves, Cody Gunter, Ebrahim Moosa, Lyndah Mupfunya
Project 4, Group 4

Final Paper

Introduction

The aim of our project is to create a machine learning recommendation engine that will allow people to get book recommendations based on a book that they enjoy. We are using the Goodreads Best Books Ever dataset from Kaggle to build our model off of. Our inspiration came by looking at the Anime Recommender that had been done by a previous bootcamp, as well as our own personal love for reading. When looking for data sources, we found two other book recommender systems that gave us ideas on how to tackle the problem. Our expectation from the model is that based on the recommendation model we use, to get 10 books that are similar to the input book.

Natural Language Processing: Data Preprocessing

Introduction

Natural Language Processing (NLP) is "a branch of artificial intelligence that helps computers understand, interpret and manipulate human language," as stated by SAS[1]. For the scope of this project, NLP allows us to analyze the book descriptions available in the data set for term frequency and sentiment in order to build a book recommendation list for the user (which is a content-based filter). Unlike the KNN model, this analysis will not be based on any numeric data, such as ratings, reviews, or rating count. Recommendations will be from similarity on book descriptions and genre.

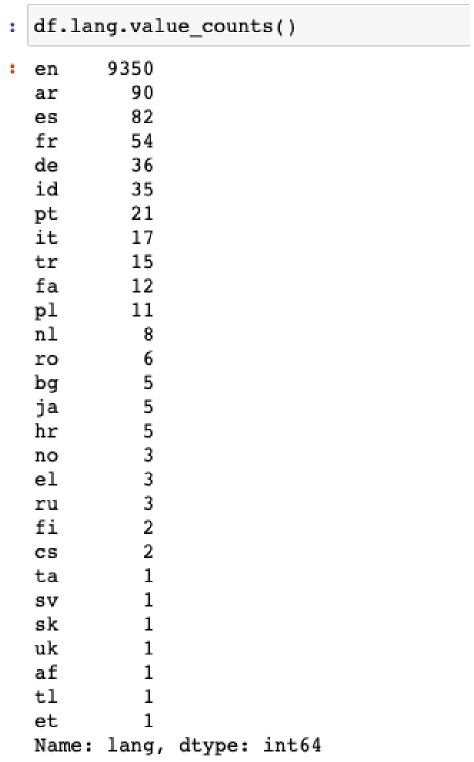
[1] https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html

Chosen Columns

Because NLP is text driven, the columns from the data set that will be used for analysis are Genre and bookDesc. The other columns included are key identifiers or other additional information needed for our user experience. These columns are: url, bookTitle, bookImage, and Author.

Data Filters

Some of the books in the data set were non-english titles. Using the langdetect library[2], we identified the 9,350 books that appeared to be in the english language (language breakdown pictured with code). We then dropped the rest.

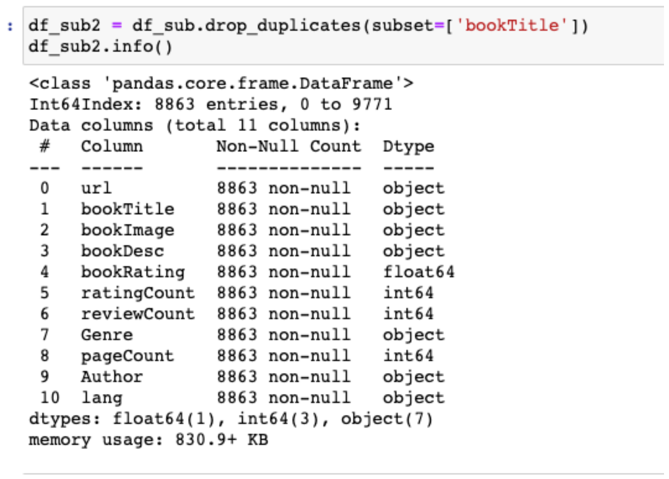


```
: df.lang.value_counts()

: en    9350
  ar      90
  es      82
  fr      54
  de      36
  id      35
  pt      21
  it      17
  tr      15
  fa      12
  pl      11
  nl       8
  ro       6
  bg       5
  ja       5
  hr       5
  no       3
  el       3
  ru       3
  fi       2
  cs       2
  ta       1
  sv       1
  sk       1
  uk       1
  af       1
  tl       1
  et       1
  Name: lang, dtype: int64
```

After this, we dropped duplicate titles. This dropped some times that had the different authors, such as *Apollyon* by Jennifer L. Armentrout and Tim LaHaye. We needed to make this decision due to time constraints, however, since we had 487 duplicate rows that we could manually review, nor could we make our user-function on our website complex enough to differentiate between two of the same titles. To drop rows, we dropped on the bookTitle column and .dropduplicates, as seen in the screenshot below:

[2] https://pypi.org/project/langdetect/



```
: df_sub2 = df_sub.drop_duplicates(subset=['bookTitle'])
  df_sub2.info()

  <class 'pandas.core.frame.DataFrame'>
  Int64Index: 8863 entries, 0 to 9771
  Data columns (total 11 columns):
   #   Column       Non-Null Count  Dtype  
  ---  ------       --------------  -----  
   0   url          8863 non-null   object 
   1   bookTitle    8863 non-null   object 
   2   bookImage    8863 non-null   object 
   3   bookDesc     8863 non-null   object 
   4   bookRating   8863 non-null   float64
   5   ratingCount  8863 non-null   int64  
   6   reviewCount  8863 non-null   int64  
   7   Genre        8863 non-null   object 
   8   pageCount    8863 non-null   int64  
   9   Author       8863 non-null   object 
   10  lang         8863 non-null   object 
  dtypes: float64(1), int64(3), object(7)
  memory usage: 830.9+ KB
```

At this stage, only 5,000 rows were kept in the data set due to size constraints of the website. The 5,000 titles were randomly chosen (using random state 42) in a jupyter notebook.

Keyword Extraction

Using Rake (from the NLTK library), stopwords, punctuation, and white space were removed from the bookDesc and bookTitle column. All words were made lowercase and made a list for easier future processing. This was done so that text analysis is not done on common words such as the, on, as, a, and so on. The code was derived from a movie-recommender example[3]

This process did not remove all symbols or numbers from the data. To address this issue, the following function was used:

[3] https://www.kdnuggets.com/2019/11/content-based-recommender-using-natural-language-processing-nlp.html

```
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

def preprocess(sentence):
    sentence=str(sentence)
    sentence = sentence.lower()
    sentence=sentence.replace('{html}',"")
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', sentence)
    rem_url=re.sub(r'http\S+', '',cleantext)
    rem_num = re.sub('[0-9]+', '', rem_url)
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = tokenizer.tokenize(rem_num)
    filtered_words = [w for w in tokens if len(w) > 2 if not w in stopwords.words('english')]
    stem_words=[stemmer.stem(w) for w in filtered_words]
    lemma_words=[lemmatizer.lemmatize(w) for w in stem_words]
    return " ".join(filtered_words)


df_new2['clean_keywords']=df_new2['key_words'].map(lambda s:preprocess(s))
```

In the function, symbols, urls, and numbers were removed. Keywords were also processed using PorterStemmer and WordNetLemmatizer libraries.

The Porter stemmer algorithm is “a process for removing the commoner morphological and inflexional endings from words in English,” as stated by the official PorterStemmer library page.[4] In other words, it finds the root/stem/base words of terms in our keyword lists and replaces them with the stem word, such as replacing *making* with *make*.

WordNetLemmatizer is a more advanced text mining library than PorterStemmer.[5] It is often considered a better alternative, but using them in conjunction with one another is common. Here, the keywords are lemmatized after they were stemmed to make sure that a wide enough net was cast to convert all of the relevant words in the dataset.

Bag of Words

Bag of Words in NLP is a technique of text modeling. It places all relevant words in the data (after processing) in a vector to be used for NLP models.[6]

[4] https://tartarus.org/martin/PorterStemmer
[5] https://aparnamishra144.medium.com/lemmatization-in-nlp-using-wordnetlemmatizer-420a444a50d
[6] https://www.mygreatlearning.com/blog/bag-of-words

For this data set, the following preprocessing took place:

- Genre: made lowercase and one word
- Book Title: made lowercase, tokenized, and stop words dropped
- Book Description: made lowercase, tokenized, and stop words dropped (described above)

These columns were added to our “Bag of Words” column.

The disadvantages of this technique is that: it ignores the location information of the word, ignores the semantics (such as sarcasm), and, in the same vein, treats synonyms separately from each other (booklover and bibliophile can be interchangeable, but would not be treated as such with this technique).

Sentiment Analysis

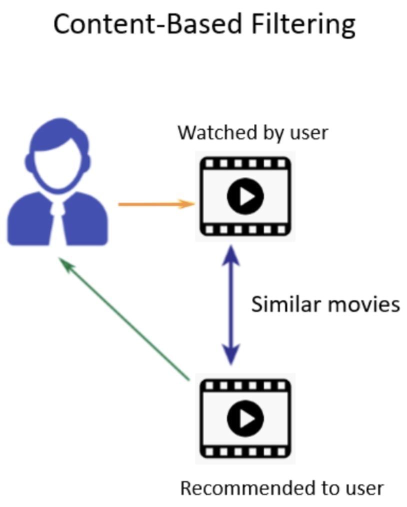
Sentiment analysis focuses on “defining the opinions, attitudes, and even emoticons in a corpus of texts. The range of established sentiments significantly varies from one method to another.”[7] Text Blob is a Python package that generates polarity and subjectivity scores. Polarity is a float within the range [-1, 1], which returns negative sentiments (-1) or positive sentiments (1). A 0 is a neutral sentiment. It will also evaluate a subjectivity score between 0 and 1, where 0 is objective and 1 is subjective.

These scores were used in the KNN model filter.

Natural Language Processing: Countvectorizer Recommender Model

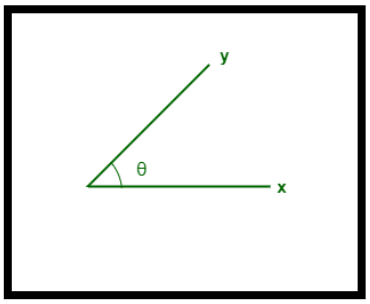
[7] https://stackabuse.com/sentiment-analysis-in-python-with-textblob/

The models we chose to use for NLP are content-based filtering. This means that recommendations are generated based on product metadata, rather than input from other users, which is collaborative filtering.[8]



CountVectorizer is a simple frequency counter for each word in the Bag of Words column. It will generate a matrix containing a count of occurance for all of the words for each of the "documents", or book descriptions. A cosine_similarity function can then be applied to compare similarities of counts between books.

Cosine_similarity is helpful for determining how similar data objects are, irrespective of their size.[9] In other words, even if the similar data objects are far apart by the Euclidean distance because of the size, there could be a small angle between them. The smaller the angle, the greater the similarity. This is a much better way of measuring similarity for a non-clustered data set.



Cosine Similarity between two vectors

We ultimately chose to include the TF-IDF Recommender instead for our website for its prioritization of more targeted and rarer terms.

Natural Language Processing: TF-IDF Recommender Model

[8] https://www.kdnuggets.com/2019/11/content-based-recommender-using-natural-language-processing-nlp.html

[9] https://www.geeksforgeeks.org/cosine-similarity/

TF (term Frequency) is another name for the CountVectorizer listed above.

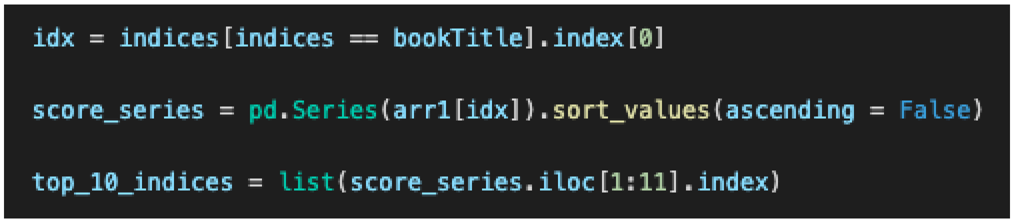
IDF (Inverse Document Frequency) is the inverse of CountVectorizer in that it will produce higher scores for the words more infrequently used in the entire matrix– this infrequency is compared to the other words from all of the documents. This means that pronouns in our book descriptions, such as “New York”, “Hogwarts”, will have higher scores than more frequently used words like “fight” and “survive”.

The TF-IDF scores produced will be a combined score. It will give higher scores to terms that occur frequently in a single/few document(s) than a term that frequently shows up across all documents.[10] TF-IDf is considered one of the best metrics to determine how significant a term is, as “words with higher scores of weight are deemed to be more significant”.[11]

Cosine similarity is a way of comparing the scores of other documents in the matrix. As stated previously, cosine similarity is a metric used to determine the similarity between two documents (book descriptions), irrespective of size. Machine Learning Plus explains cosine similarity as, “When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.”[12]

Cosine similarity will be a numpy array (matrix) with a similarity score between each of the documents. The similarity score generated is how similar the documents are. In the code snippet below, the user input is located on the index, and then the score in the cosine matrix (arr1) are sorted from greatest to least.

[10] https://hackernoon.com/document-term-matrix-in-nlp-count-and-tf-idf-scores-explained
[11] https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
[12] https://www.machinelearningplus.com/nlp/cosine-similarity/



```
idx = indices[indices == bookTitle].index[0]

score_series = pd.Series(arr1[idx]).sort_values(ascending = False)

top_10_indices = list(score_series.iloc[1:11].index)
```

This model, due especially to scores for recurring pronouns, is highly vulnerable to franchises in the dataset, such as the Harry Potter and Twilight franchise, which have several books with franchise buzz words. However, we chose this model because it also doesn't get overwhelmed by more mundane words– such as "fight"– that are not removed as stop words the way CountVectorizer does. It ends up being more reliable, especially for single releases or for the less franchised books on the list.

Natural Language Processing: Launching on Flask App

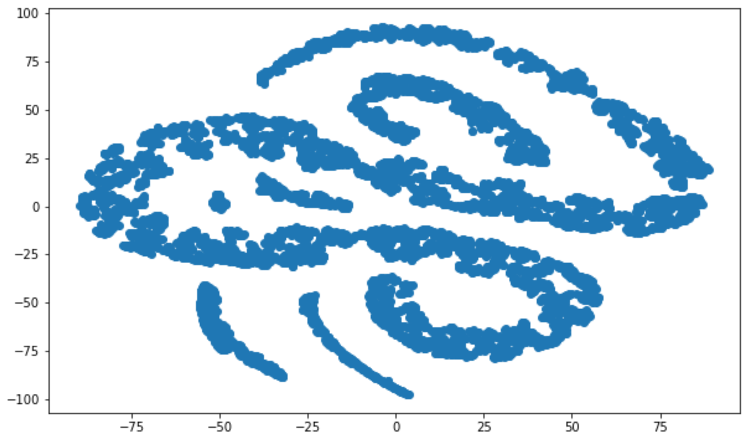
Due to GitHub size constraints (100mb file sizes), we could not store the cosine similarity matrix as a document. To get around it, we published the matrix to an AWS S3 bucket and accessed the information with s3fs. This will allow us to avoid processing time whenever a new search is loaded.

The only user filter/input is the book title. Other filters, at this time, were unfeasible and complicated the results. What is returned to the users (from the random sample data set) are the *most* related books in the entire list. User inputs could result in inaccurate or poor recommendations due to the constraints placed on the return.

Machine Learning KNN Model

Due to the scope and type of problem this project is tackling, a KNN model was selected as the best model to recommend book titles to a user. KNN stands for k-nearest neighbors and is ‘is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.’[13] In other words, the algorithm works by evaluating numeric features in the data set, and predicts the clustering of the data points. Due to the model only evaluating numeric data, we will only be using features/ columns that can be easily one-hot encoded and that are already in numeric form. When you ask the model to give you ten recommendations based on a data point, in our case a book, it pulls the ten nearest neighbors to that original data point.

The columns selected to become features for the KNN model differ from the TF-IDR recommender, numeric data is used rather than words. For this reason columns that could not be one-hot encoded or not in numeric format already were dropped. This left us with the book ratings, count of reviews, count of ratings, the sentiment polarity and subjectivity, and the one-hot encoded columns of genre and author. The three numeric fields that we started with, rating, count of ratings, and count of reviews were scaled. To look at clustering of the books, principal component analysis (PCA) was used on the same three columns that were scaled. PCA is the process of computing the principal components to reduce the dimensionality of the data[14]. The PCA columns along with the one-hot encoded genre and author



13 https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point.

14 https://builtin.com/data-science/step-step-explanation-principal-component-analysis

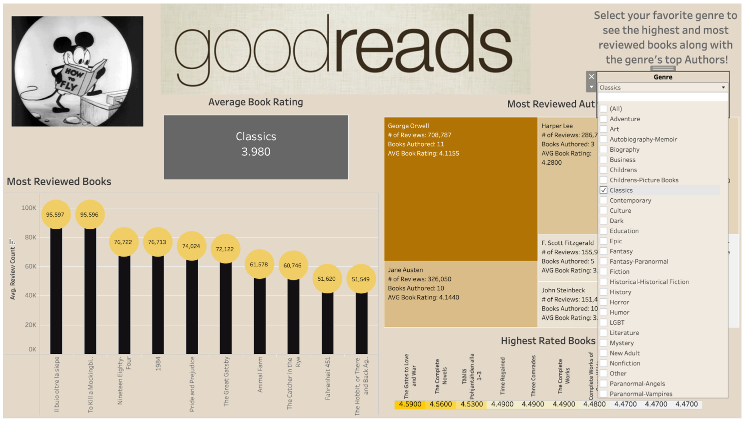
columns were concatenated to produce a dataframe with all the columns needed to be run through the model. To visualize the clusters, the dataframe was transformed using t-sne to reduce dimensionality to just two dimensions. This reduced data is plotted onto a graph to give us a visual representation of the clusters.

For our recommender, we wanted to pull the ten closest neighbors to the book that the user is inputting. Due to the model spitting out the same book that is being asked to compare, the number of neighbors we are asking the model to show is actually eleven. Since this model does not take into account the book description and book name, there is a very small chance that the model will recommend books that are in the same series, thus eliminating obvious choices that the user may already know of.

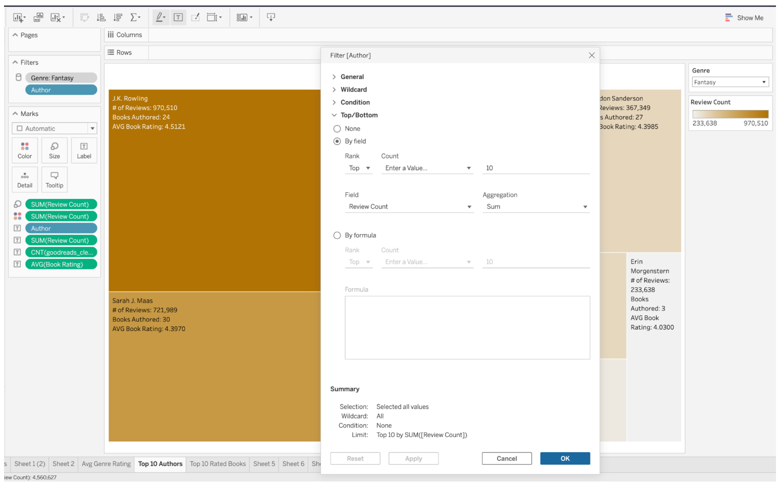
Integrating the model into the Flask application proved to be more difficult than originally thought. The response the ajax request was receiving from the model was not returning as a dictionary but rather as a massive string with the content being in the form of a dictionary. Once this was resolved, the final step was making sure that the UI matched that of the other model and the general theme.

Tableau

Our Website has 2 interactive tableau dashboards. Both dashboards respond to a genre filter where you can select any genre that is contained in the goodreads dataset and both pages will display 4 visualizations. All visualizations across these two pages respond to any change in the genre selector, whether it be representing one genre, multiple or all but one.



The first dashboard presents the top 10 in the most reviewed books, the most reviewed authors, and the



Filters: Genre: Fantasy; Author

SUM(Review Count)
SUM(Review Count)
Author
SUM(Review Count)
CNT(goodreads_cle...
AVG(Book Rating)

J.K. Rowling
# of Reviews: 970,510
Books Authored: 24
AVG Book Rating: 4.5121

Sarah J. Maas
# of Reviews: 721,989
Books Authored: 30
AVG Book Rating: 4.3970

don Sanderson
Reviews: 367,349
s Authored: 27
Book Rating: 4.3985

Erin Morgenstern
# of Reviews: 233,638
Books Authored: 3
AVG Book Rating: 4.0300

Genre: Fantasy

Review Count: 233,638 – 970,510

iew Count): 4,560,627

Filter [Author]

- General
- Wildcard
- Condition
- Top/Bottom
  - ( ) None
  - (●) By field
    - Rank: Top | Count: Enter a Value... | 10
    - Field: Review Count | Aggregation: Sum
  - ( ) By formula
    - Rank: Top | Count: Enter a Value... | 10
    - Formula:

Summary

Selection: Selected all values
Wildcard: All
Condition: None
Limit: Top 10 by SUM([Review Count])

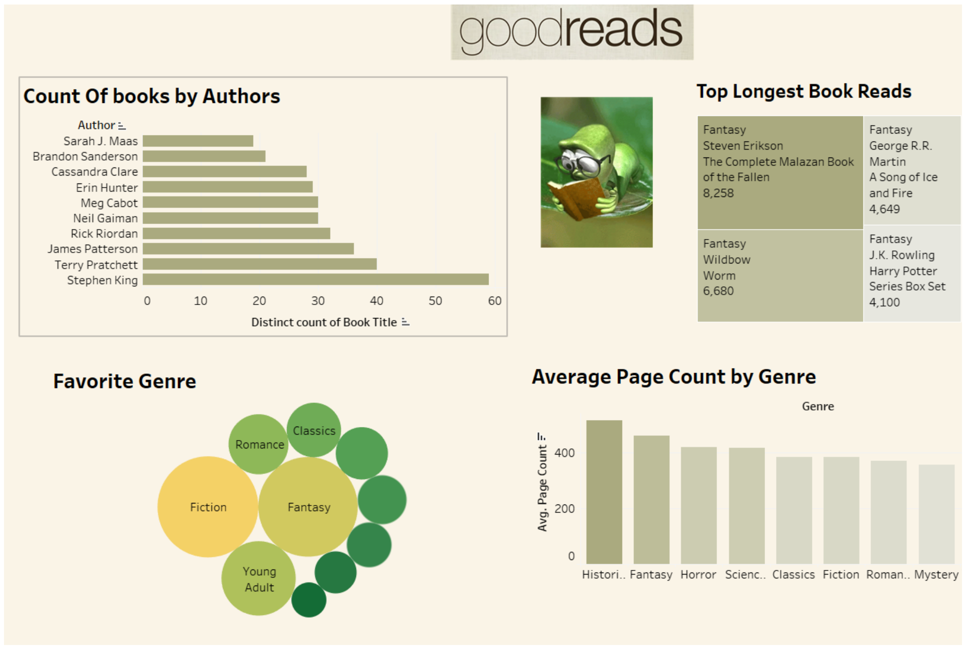
highest rated books. For the color theme, shades of brown, yellow and gold gives the page the aesthetic of wooden bookcases filled with books. The author visualization is a highlight table displaying the author, number of reviews, books authored, and their average book rating. The size of each chunk of the table each author represents is determined by the total number of reviews for each author divided by the total number of reviews of the cumulative top 10. The greater review count presents a larger area in the table. Books with the highest review count are represented by a lollipop chart with the number of reviews presented on each lollipop. Top 10 highest rated books are presented in a highlight bar in the lower right corner of the page, with the higher rating in a brighter colored bar. Lastly, the genre's average book rating is displayed underneath the goodreads title in a gray box.

In building this dashboard, it was a challenge to get the 3 top 10 visualizations to respond with their respective top 10 at the same time with different measures. This was resolved by adding a second filter to each sheet used in the dashboard.
All 3 of the added filters displayed the top 10 of the measure involved while keeping the genre filter as the primary filter. Keeping the genre filter as the primary filter allowed the new filter to respond to the genre selection, then display the top 10.

Our second tableau dashboard also generates 4 visualizations contingent on the genre f filter. To contrast the wood aesthetic, this dashboard has shades of green. The top 10 books for each genre by page count, top 10 authors by measure of books written, and the average page count and number of books in the dataset for each genre is displayed.

2nd Dashboard

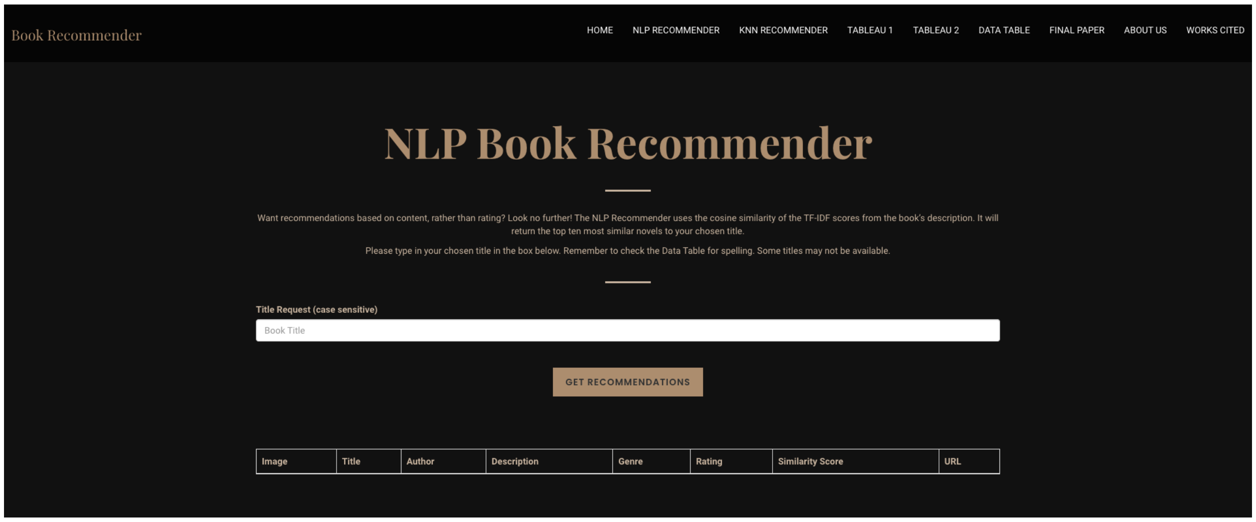
The 2nd dashboard was exploring four different visualizations that were aimed at exploring the Genre as well as the count of books by author and top longest reads. The visualizations are aimed at establishing correlations and dynamics and to see if genre can determine. Between all the features the filtering was done by Genre and the visualizations change based on the tree map, favorite genre and average page count per genre. The first horizontal bar chart shows the author with the largest distinct count of books and this can be reflected in ascending and descending order. The top longest book reads shows the average page count of the longest books. Bubble charts reflect the light colors being the more popular genre and the darker colors being the less popular genres. The Fiction genre reflects as being the most popular but also can fall under a very general genre that can fall under any category. It generally shows fiction being more popular than non fiction. The bar graph is also filtered by the top 10 genre and reflects that the longest book reads are reflected in darker shades and tend to be the historical books and fantasy and the shortest book reads tend to be the non-fiction books and childrens books. All the visualizations are relatable on both dashboards.



The challenges with building this visualization was building a friendly filter that can give accurate results while interchanging the charts. There was also a challenge in the data source as the data needed further cleaning to bring out accurate results. The focus of the visualization was to influence book recommendations based on longest book reads , page counts of book reads based on genre and show the popular genres. The dashboard was embedded and hosted on Tableau Public.

<u>Website:</u>

Our website consists of nine pages. Apart from the home page, the other 8 other pages consist of the 2 models, the 2 tableau dashboards, the data table, about us, final paper, and work cite page. The same design themes are shared throughout the pages, this includes a black background with beige text. The following is an image of our recommender page, which shows the color palette used:



NLP Book Recommender

Want recommendations based on content, rather than rating? Look no further! The NLP Recommender uses the cosine similarity of the TF-IDF scores from the book's description. It will return the top ten most similar novels to your chosen title.

Please type in your chosen title in the box below. Remember to check the Data Table for spelling. Some titles may not be available.

Title Request (case sensitive)

Book Title

GET RECOMMENDATIONS

Image	Title	Author	Description	Genre	Rating	Similarity Score	URL

In our recommender models, the user can type a title into the search bar and generate the top ten recommendations from that model. With each search the table regenerates, providing the user a fresh table with new recommendations.

The navigation bar appears on the top of the page and scrolls with the user allowing them to access it at any point. The Tableau pages use the Tableau API to embed the dashboards into the page. The Tableau dashboards are also interactable allowing users to select what genres they want to compare.

The data table uses D3 to create the table. The CSV that is used for the table is the one used for the NLP recommender. This was done due to memory constraints that we were running up against.

The work cited and about us pages use HTML to display both text and images. The Final Paper page will be embedded using an iframe.

Conclusion:

This project has been a resounding success when it comes to building a machine learning model. We hope that users can rely on our website for their book exploration needs. Whether they’re exploring the most popular books by genre, want a book recommended based on content, or a book recommender based on popularity metrics, we hope we can build a one-stop shop for our users.

Limitations and future work:

We would like to allow the user to enter titles without needing to be concerned by the case sensitivity. Additionally, we would like users to have an autofill bar that allows them to type in a title and choose additional information, such as author, which would allow us to have duplicates in our data set.

Also, we would like our recommendation tables to be longer and sortable, so that the user can sort through number of reviews to find more “indie” titles, as an example, or have a higher chance of finding a new title with more options available.

Due to memory constraints we were limited to the amount of books we could use in our project. This memory constraint also affected our data table since we could not keep our original cleaned csv stored in Heroku. In future work we would like to be able to present our data in a more clean way rather than using the csv for the NLP recommender. Additionally, we could not make the theme we chose compatible with a bootstrap responsive table with pages and search capabilities. We like to make sure we have that in a subsequent launch.

We would also like to find a proper resolution to books with the same title but different authors. In our data we dropped all books with duplicate names due to there being so few, but with potential real world application of our model, we would like to find a way to keep them.