2023

A Novel Content Based Recommendation System for Australian Schools

JAYDEN DZIERBICKI

WORD LIMIT: 2000 +/- 30% (MAX 2600 – EXCLUDING FIGURES, TABLES AND CODE SECTIONS IN REPORT)

DOCUMENT WORD COUNT: 2594

Contents

Background

Since the 1950s, natural language processing (NLP) has emerged as a key intersection between artificial intelligence and linguistics, with applications like sentiment analysis and text classification. This paper explores NLP for constructing a content-based recommendation system. Content-based filtering aims to recommend items to users based on item attributes. Here, we define an item as an Australian school textbook and its attributes as description, author, and category, while the user is someone seeking a subject-specific textbook for a particular grade. We conducted analysis using Jupyter Notebook with Python 3.9.7. To accomplish our goal, we developed an optimized K-Nearest Neighbour model, evaluated the F1 score, selected the top two recommendations from the test data, and assessed overall recommendations by calculating the mean F1 score based on our defined user.

Data

Overview

Data was obtained from the MA5851 Assessment 1 folder on learn JCU and downloaded on 19th March 2023. The data was saved locally and imported for data pre-processing and analysis. The data included information about the ISBN of a textbooks used in Australian schools, as well as additional variables that identify a school, the state for the school, year level, and subject area for which the book is assumed to be used in; noting that subject may be inaccurate. We summarise the data in table 1.

Table 1 provides a summary of the sample data, which originally comprised 5 columns and 1804 rows with diverse data types.

	School_ID	State	Year	Subject	ISBN
Number of rows	1804	1804	1804	1804	1804
Data type	Int64	object	Int64	Object	Int64
Unique observations	40	8	13	30	1071

Data Pre-processing and Explorative Data Analysis: Sample data

We took four pre-processing steps on the sample data to ensure its accuracy. We corrected errors, removed duplicates, and converted School_ID, State, Year, and Subject to categorical data types. We suspected duplicate entries were caused by multiple classes for the same subject in a school. We observed 'English' was listed three different ways and we renamed them as 'ENGLISH' to standardize.

Data science frequently presents the challenge of having data in an unsuitable format, highlighting significant effort to transform it. To address this issue in our NLP classification recommendation model, we proposed a solution involving the creation of a data frame with a unique ISBN in each row and a second column listing the known users of the corresponding textbook (figure 1). This approach allows us to evaluate our recommendation model later by comparing our designated user as the ground truth (Datagen, ND). Through utilizing the known users of each textbook, we can compare the model's recommended textbooks and evaluate it via this novel approach (Datagen, ND).

School_ID	State	Year	Subjects	ISBN
2	VIC	8	LANGUAGES	9780007485505
2	VIC	12	LANGUAGES	9780060787325
2	VIC	11	LANGUAGES	9780060787325

ISBN	User_type
9780060787325	[USER_TYPE_1, USER_TYPE_2]
9780007485505	[USER_TYPE_1]

Figure 1: Displays the transformation applied to the JCU sample data to render it in a usable format for evaluating the recommender model. The user type was utilized as the ground truth for this purpose.

Define our user

We consulted the literature to gain domain expertise on the Australian education system via desktop research, this helped us be contextually informed, highlighting the importance of subject matter expertise in data science (Yablon, 2020). For our recommendation system we selected users based on subject and year level as our ground truth (Moam Grammer, 2015; Datagen, ND). We counted the number of defined users per ISBN and found that some textbooks spanned multiple year levels, which through domain experience is not an issue. However, we discovered that five specific ISBNs spanned vastly different subjects, and through desktop research discovered these items being dictionaries and blank notebooks (figure 2). Therefore, we removed them from our analysis*, resulting in 1066 unique ISBNs. Although this may introduce bias, our primary objective was to recommend textbooks rather than everyday items such as dictionaries.

	ISBN	User_type	user_type_length
201	9780190303488	[ENGLISH_6, LANGUAGES_9, ENGLISH_5, LANGUAGES_...	11
443	9780730389422	[FOOD TECHNOLOGY_8, LANGUAGES_9, MUSIC_9, LANG...	9
1070	9798708474995	[SCIENCE_8, MATHEMATICS_10, LANGUAGES_10, SCIE...	8
451	9780732979966	[COMPUTER SCIENCE_12, LANGUAGES_10, MATHEMATIC...	8
851	9781741353501	[ENGLISH_1, ENGLISH_6, ENGLISH_5, MATHEMATICS_...	6
385	9780648237327	[ENGLISH_1, ENGLISH_12, ENGLISH_0, MATHEMATICS...	5
638	9781118489291	[GEOGRAPHY_10, SCIENCE_9, SCIENCE_10, HISTORY_...	5

Figure 2: Represents the user count per ISBN, where it should be noted that the ISBNs presented are not actual but used for illustrative purposes of count. For the list of actual ISBNs, please refer to the code attachment.

Explorative Data Analysis (EDA) – JCU Sample

We undertook EDA on the JCU sample data; at this stage we elected to retain all ISBNs for API data retrieval, as we would later remove data based on user groupings limiting the scope of the recommender and allowing flexibility to change scope of our defined user, we summarise our findings in table 2.

Table 2 summarizes the results of the initial exploratory data analysis (EDA) conducted on the provided sample data, along with a recommendation to utilize the data as our defined user.

Variable	Comments	Used in defining user
Subject	We observe core subjects such as English, mathematics and languages are some of the most common textbooks used in Australian schools, accounting for a total of 1,011 observations in the sample data, highlighting the model possibly might not generalise well to other subjects.	Yes, limitations include low sample count such as some subjects occurring once.
State	The observation count for Victoria, NSW, and Qld in the dataset aligns with Australia's population distribution (ABS, 2022). These states have dedicated textbooks for the Higher School Certificate (HSC) and Victorian Certificate of Education (VCE).	No, we know states such as Tasmania through our own domain knowledge utilise textbooks from HSC/VCE. If recommender was state specific would limit scope too much
Year	We observe that most textbooks in the dataset are used in high school or college level classes, with a smaller proportion of textbooks being used in earlier years. Highlighting the model possibly might not generalise well to all year levels.	Yes, though we need to keep in mind that some subjects can span multiple year levels and this is usually not an issue. In addition, low count earlier learning
School	We cannot provide insights on the distribution of textbooks by school since we do not have additional insight into school data and elect to exclude this from the user grouping.	No, we did not have additional data on the schools and elected not to include it

We observe two issues in figure 3; visually we have cohorts which are underpresented, and cohorts which are overepresented. This could result in a biased model towards the overepsented user group leadining to comprimised performance for the underepsented user. This can be addresssed in machine learning via multiple arprpoaches such as stratified sampling, k-fold cross validation or simply aknlowedging it as a limitation; the most important thing is that a data scientists be transparent about such limitations (Yu-Yen Ou et al., 2006).

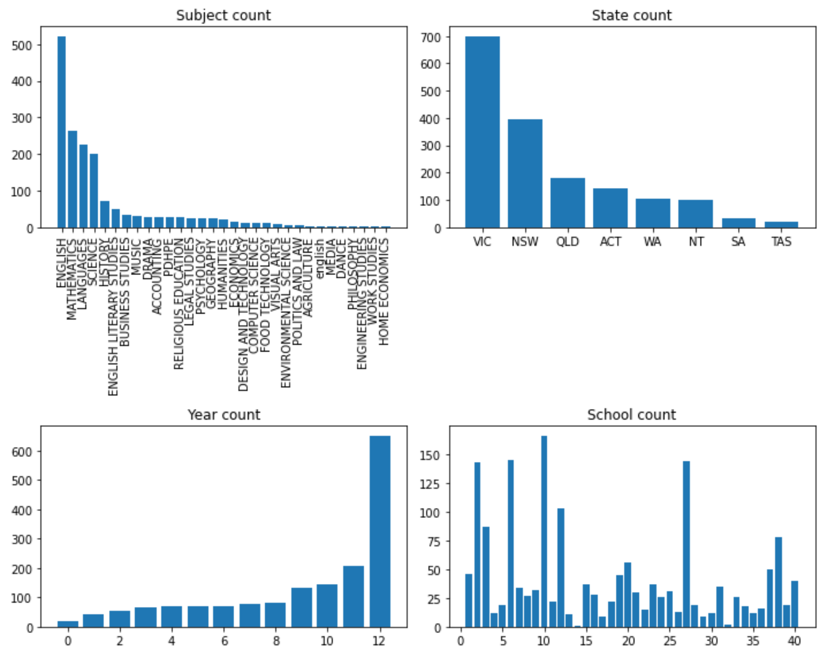


Figure 3: illustrates the distribution of categorical variables in the pre-processed dataset after eliminating duplicates. The results indicate that core subjects have the highest frequency, and the data set is dominated by larger states. Moreover, the count is higher for high school and college levels when compared to earlier years. Although there is no further information on schools, it is hypothesized that the institutions with spikes in the data are probably larger schools, high schools, or colleges.

API Data Acquisition

APIs are essential tools for data scientists working in the technology field. Data scientists use APIs to retrieve data. We retrieved text data using the ISBN/titles which allows us to enrich our data. Understanding the basics of HTTP, RESTful services, and data formats such as JSON and XML is crucial for effective API use; as well as using API documentation (Juviler, 2022). It is important to be aware of API limitations, such as rate limiting and data accuracy issues, and how to mitigate them through techniques like partial requests (Juviler, 2022).

Table 3: Outlines the APIs utilized for enriching our data, as well as the limitations and variables extracted from them, code utilised supplied in appendix.

APIs	Variables extracted	Justification	Limitations
Google Books	Title; publisher, authors; description and categories	Initial API request made on 1066 unique ISBNs. Utilised partial request as documented in technical guide.	- Failed to retrieve all required data, 653 ISBNs incomplete. - Needed to include a rate limiting step
Trove	Title; publisher; authors categories and description	653 ISBNs did not contain complete information, attempted to enhance through title search on trove.	- Categories had noise, could be used as somewhat of a description - Failed to retrieve all required data

Google & Trove API

We utilized Google Books API to gather information (table 3) though due to server limitations we had to implement the **time.sleep()** function to limit the number of requests made to the server. We utilized the 'fields' parameter to reduce the amount of information we requested from the server (Google, Performance tips), ensuring efficient data retrieval.

We found that the necessary data was not always available via Google Books API, resulting in missing information. This is a common occurrence during data retrieval, and it is crucial for data scientists to prepare contingencies, such as utilising other APIs. We used Trove API to perform two requests - the first one to retrieve missing titles for ISBNs not found in Google, and the second search based on book titles; this allowed us to minimise unnecessary API calls. However, we encountered a constraint with Trove where we could not obtain descriptions when searching for ISBNs alone as the snippet feature mainly returned empty values for ISBNs, hence the need for title search. This underscores the significance of data scientists tackling challenges from multiple angles and experimenting with different search parameters to ensure an enriched dataset.

API Request Standardization, Augmentation & Additional Pre-processing

We utilized NLTK and BeautifulSoup to standardize and clean HTML elements in the columns of both Google and Trove APIs, ensuring consistency in data format. To avoid potential errors caused by differences in metadata quality, we standardized each column separately before augmenting them into a single corpus. This was crucial as author names appeared differently in Trove and Google, which could have led to errors when merging names during our EDA which we depict in figure 4.

ISBN	Author	Publisher
9780006755234	['Diana Wynne Jones']	HarperCollins UK
9780076716753	['Charles William McLoughlin', 'Marlyn Thompson', 'Dinah Zike', 'Ralph M. Feather', 'Glencoe/McGraw-Hill']	

ISBN	Author	Publisher
9780006755234	[dianawynnejones]	harpercollinsUK
9780076716753	[charleswilliammcLoughlin, marlynthompson]	

Figure 4: we can observe that the author names were merged to facilitate exploratory data analysis (EDA) and ensure that individuals such as 'William Smith' and 'William Shakespeare' are counted separately. This step is crucial to avoid duplicating the count of an author with the same first name.

Similarly, Trove returned HTML text for descriptions, while Google returned a cleaner output. It is crucial for data scientists to be aware of metadata quality and variations in consistency between sources when making API requests, which can improve the retention and recall of the data (Hider, 2018). We engineered new features for Google and Trove representing the cleaned text of each column using the functions listed in table 4.

Table 4: presents the utilization of NLTK and Beautiful Soup to clean and pre-process our text data for analysis in our NLP recommender model. The objective was to diminish the noise in the data by incorporating NLP theory such as Zipf's theory.

Custom function	Purpose	Package dependencies	Application
# function for text cleaning def clean_text(text): # remove backslash-apostrophe text = re.sub("\'", "", text) # remove everything except alphabets text = re.sub("[^a-zA-Z]"," ",text) # remove whitespaces text = ' '.join(text.split()) # convert text to lowercase text = text.lower() return text	The purpose of this is to standardise the text in columns, this allows the machine to treat each word as….	Required natural language tool kit	Google API feature engineering: - Clean_categories - Clean_title - Clean_author (refer to figure 5) - Clean_publisher - Clean_description Trove API feature engineering - Clean_categories - Clean_title - Clean_author (refer to figure 5) - Clean_publisher - Clean description (HTML cleaning)
def clean_html(text): # Remove HTML tags soup = BeautifulSoup(text, 'html.parser') clean_text = soup.get_text() # Remove any remaining non-alphabetic characters and convert to lowercase clean_text = re.sub('[^a-zA-Z]', ' ', clean_text).lower() # Remove any extra whitespace clean_text = ' '.join(clean_text.split()) return clean_text	The purpose of this is to standardise the description text obtained from Trove API due to the extraction of HTML string retrieved from Trove **description** [Teaching THRASS whole <b...	Required Beautiful Soup as NLTK did not have a function to clean HTML that we could locate	
def remove_stopwords_and_stem(text): stemmer = PorterStemmer() tokens = word_tokenize(text.lower())	The purpose of this is to standardise the text in columns, this allows the machine to…	Required natural language tool kit	

no_stopword_text = [stemmer.stem(w) for w in tokens if not w in stop_words] return ' '.join(no_stopword_text)			

Zipf's law is a statistical law that suggests that if we plot a graph between a word's frequency of occurrence and its rank in a corpus, we will observe an inverse relationship between the two. Zipf's law proposes that there are three types of words in a document (Wisdomml, 2022), which we visualize in Figure 5:

- Stop words: These are frequently occurring words such as ‘and’, ‘an’ etc.
- Significant words: these words have a moderate frequency in the document and contribute actual meaning to the text
- Rare words: These words do not occur frequently and tend to have lesser importance

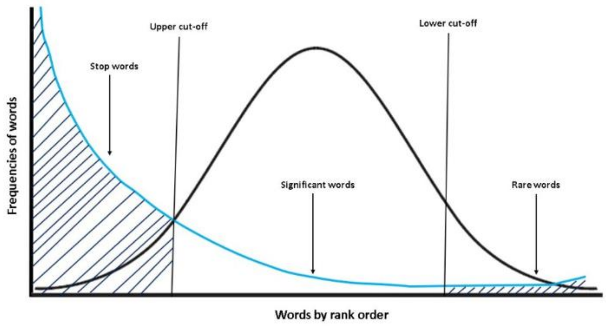


Figure 5: provides a graphical representation of Zipf's law, as described by Wisdomml in 2022.

Cleaning and removing stop words from text data is a common practice in NLP that is based on the theory that certain words are used frequently in human language but do not contribute much to the overall meaning of a text. By removing these stop words, we can focus on the more important words that convey the primary message of the text, the significant words (figure 5). The decision to remove stop words should be carefully balanced in the context of the task (Wilame, 2023). If done incorrectly can lead to inaccurate interpretations of the text, such as in the case of the phrase "The movie was not good at all," where removing stop words would result in "movie good," which is a wrong interpretation. In our case we determined that removing stop words would not be an issue based on our domain knowledge. We see visually that our text contains many stop words (figure 5).

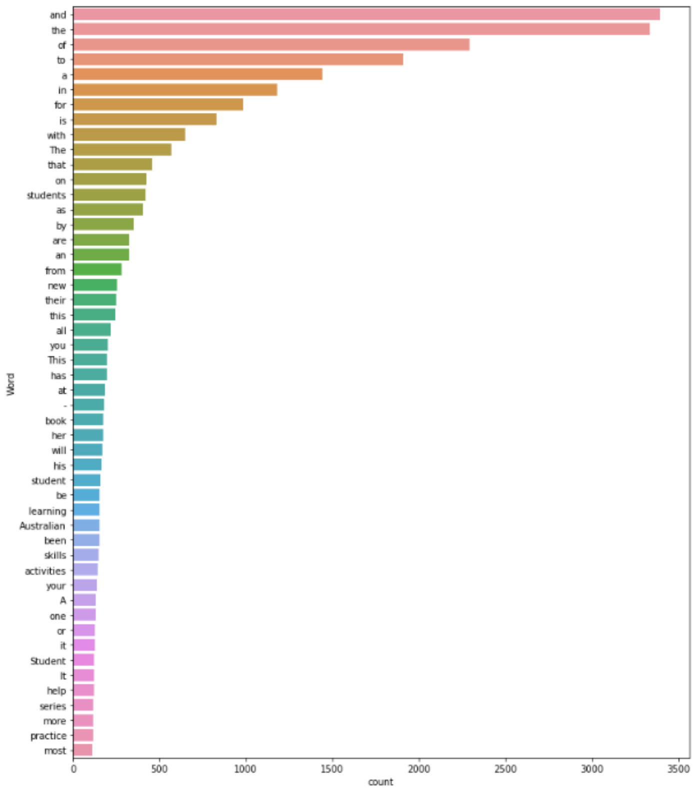


Figure 6: depicts the exploratory data analysis (EDA) conducted on the text data from Google API, which revealed a significant frequency of stop words. Although not presented in this figure, a similar trend was also observed in the text data from Trove as part of the pre-processing step.

At this stage both our Google and Trove data was in a similar structure and we combined the sources together as summarised by the rules in table 5 via the **pd.merge()** function. We removed ISBNs which did not contain a title, resulting in 948 unique ISBNs.

Table 5: presents the rules implemented for merging data from Trove and Google to enhance our corpus, along with any limitations and the number of missing observations.

Variable name	Rule	Number of missing variables	Note
Clean title	If Google missing merge Trove	0	
Clean Author	If google missing merge Trove	81	
Clean Publisher	If Google missing merge Trove	498	
Clean Categories	If Trove missing merge Google	235	Categories from Trove contained elements of both categories and description like text. This minimises missing description information
Clean description	If Google missing merge Trove	280 *0	*If merged with categories from Trove categories column we obtain 0 missing observations. We highlight that Trove categories also included elements of descriptive like text in addition to the snippet feature returned from trove API

Explorative Data Analysis – API Google & Trove

We conducted some EDA on our clean text, we explored most common author, most common category and most common publisher and observed the following characteristics (figure 7). In addition, we conducted further EDA on text training/test set in the next section. In addition, figure 7 suggests and support the idea that our data overrepresents those core subjects such as English and maths which we observed in figure 3 highlighting possibility of a biased model.

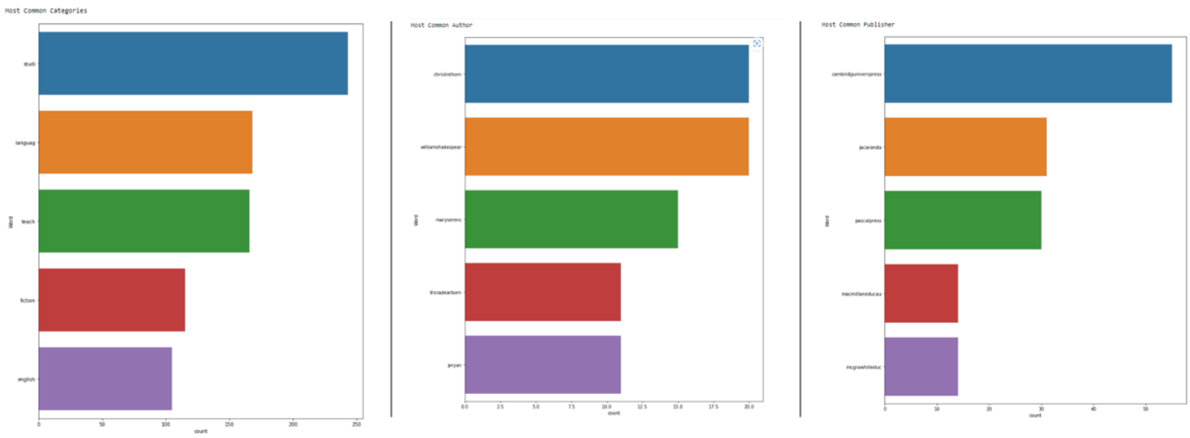


Figure 7: showcases the exploratory data analysis (EDA) performed on the top five most common categories (left), author (middle), and publisher (right) of the cleaned corpus. The distribution of corpus texts was found to favor the core subjects discussed earlier, potentially indicating a biased model.

Model Development

We elected to utilise the F1 score as our metric of choice when evaluating the performance of our model, this can be defined formally with the following equations:

$$F1 = 2 * \frac{Precision * Recall}{Precsision + Recall}$$

$$Precesion = \frac{Number\ of\ relevant\ recommendations}{Total\ number\ of\ recommendations}$$

$$Recall = \frac{Number\ of\ relevant\ recommendations}{Total\ number\ of\ relevant\ recommendations\ in\ the\ ground\ truth}$$

We compared our model's performance with a basic naive model that assumes every user is recommended the same textbook. The naive model's test F1 score of 0.098 was obtained by assuming each user would be recommended two textbooks from the most common user type ENGLISH_12 (figure 8). It serves as a crucial baseline for data scientists to assess the performance of advanced models (Nair, 2022).

```
Mean F1 Score of NAIVE Books: 0.09879629629629628
```

	Test_ISBN	Test_User	Recommended_ISBNs	Recommended_Users	F1_Score
0	9780947225698	[DESIGN AND TECHNOLOGY_12]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
1	9780316382007	[ENGLISH_4, ENGLISH_5]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
2	9780571336173	[ENGLISH_12]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	1.0
3	9780725334659	[ENGLISH_6]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
4	9780141359410	[ENGLISH_7, ENGLISH_8]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
...	...	...	...	...	...
175	9780190323226	[ENGLISH_6]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
176	9780733970665	[LANGUAGES_11]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
177	9780224025720	[ENGLISH_3, ENGLISH_4]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
178	9780980874921	[SCIENCE_12, COMPUTER SCIENCE_12]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0
179	9781741353372	[MATHEMATICS_2]	9780571336173, 9780099462217	[[ENGLISH_12], [ENGLISH_12]]	0.0

180 rows × 5 columns

Figure 8: A naïve recommendation model is presented that recommends the same two books to all users using the test set. The recommendation is solely based on the most frequent user category, which serves as a baseline evaluation tool with a score of 0.098 for assessing the performance of our recommendation model.

We merged our user ground truth data with our corpus of text using the **pd.merge()** function to prepare the data for analysis (figure 9).

	ISBN	User_type
0	2019052211324	[MUSIC_12]
1	9780006755234	[ENGLISH_9]
2	9780007141425	[ENGLISH_7]
3	9780007456208	[ENGLISH_4]
4	9780007485505	[LANGUAGES_8, LANGUAGES_9]
...	...	...
1061	9789814629294	[MATHEMATICS_3]
1062	9789814779098	[COMPUTER SCIENCE_7]
1063	9789814792530	[LANGUAGES_11]
1064	9789814792547	[LANGUAGES_11]
1065	9789839494488	[LANGUAGES_12]

	isbn	title	clean_description	authors	publisher	categories
0	9781741250879	victorian target handwrit		janepinsk triciadearborn joryan stephenmichael...		penmanship studi teach primari victoria penman...
1	9780648237327	kluwel home read yellow level	suitabl lower primari student	stankluzek andrewcoldwel		
2	9781742990682	wilfrid gordon mcdonald partridg	wilfrid gordon mcdonald partridg live next doo...	memfox	omnibu book	boy
3	9781741351750	imath	imath investig base numeraci program comprehen...	maryserenc lenaford chrislinthom		mathemat
4	9781742152196	nelson math australian curriculum	nelson math australian curriculum program supp...	glendabradley paulinerog		mathemat
...	...	...	...	...	...	...
985	9780730365464	jacaranda math quest australian curriculum e l...	jacaranda math quest ac jacaranda math quest a...		jacaranda	
986	9781107628199	physic ib diploma coursebook free onlin materi	physic ib diploma sixth edit cover full requir...	k tsoko	cambridg univers press	educ
987	9780415684811	global polit	nd edit global polit new introduct continu pro...	jennyedkin		polit scienc
988	9781420229431	queensland senior physic educ	queensland senior physic educ specif written s...	glennamezdroz suedicken geoffhosford	macmillan educ au	physic educ train
989	9781471858161	edexcel level french includ		karineharrington kirstythathapudi rodhar wendy...		

	isbn	clean_text	User_type
1	9780648237327	kluwel home read yellow level suitabl lower pr...	[ENGLISH_2, ENGLISH_0, MATHEMATICS_1, ENGLISH_...
27	9780190312480	handwrit first victoria foundat oxford handwri...	[ENGLISH_1, ENGLISH_12]
31	9781741353501	kluwel home read orang level suitabl upper lev...	[ENGLISH_6, ENGLISH_4, ENGLISH_2, MATHEMATICS_...
49	9781563160332	iqra arab reader textbook arab languag reader ...	[LANGUAGES_3, LANGUAGES_2, LANGUAGES_12]
88	9780190302689	australian primari integr dictionari thesauru ...	[ENGLISH_3, ENGLISH_4, ENGLISH_5]
...	...	...	...
981	9781316422953	nelson peak perform nelson peak perform write ...	[MATHEMATICS_12]
984	9781107587434	cambridg vce health human develop unit pack de...	[MATHEMATICS_12]
985	9780730365464	jacaranda math quest australian curriculum lea...	[MATHEMATICS_12]
986	9781107628199	physic ib diploma coursebook free onlin materi...	[SCIENCE_11]
989	9781471858161	edexcel level french includ karineharrington k...	[LANGUAGES_12]

Figure 9: illustrates the merging of our JCU sample data, utilizing the ground truth method, with our text corpus. Through trial and error, we generated a clean text field that would be utilized in our recommender model, as described in Table 5.

We introduced a new feature called "clean text," which was a combination of the title, categories, and description (figure 9). Through trial-and-error process we tested various combinations of text features and selected the one that resulted in the highest test F1 score for our out-of-box model (default parameters). Table 5 summarizes the variation in F1 scores resulting from different feature selections. We excluded users from our model in which there were less than 5 observations resulting in 900 unique data points.

Table 5: outlines the assessment of the impact of API data on a vanilla K-Nearest Neighbors (KNN) model by retaining essential features such as title categories, description, author, and publisher. The results showed that the best-performing vanilla model was a combination of title, categories, and description only.

Clean text	Test F1 Score (Untuned model – set seed applied)
Title	0.22
Title + Description	0.25
Title + Authors	0.23
Title + Publisher	0.22
Title + Categories	0.25
Title + Categories + Description	0.266
Title + Categories + Description + Authors	0.263
Title + Categories + Description + Authors + Publisher	0.25

To ensure reproducibility, we set a random seed of 123 using the **random.seed()** function and split our data into training and testing sets using the **train_test_split()** function from scikit-learn's model_selection module, with a test size of 20%; we will discuss our attempt to handle unbalanced data towards the end. We used the **MultiLabelBinarizer** function to encode the User_type variable as a binary matrix, which is necessary for multi-class classification problems. We calculated summary statistics for our corpus, which provides insights into the distribution of documents between our training and testing sets. We have 720 documents in our training set and 180 in our testing set. The total number of unique words in our vocabulary is 5419, and the number of tokens is 3534 for the training set and 861 for the testing set. The mean number of tokens per document in the training set is 4.91, while in the testing set, it is 4.79. These statistics can help us understand the characteristics of our corpus and help influence parameters such as max features for **TfidfVectorizer().**

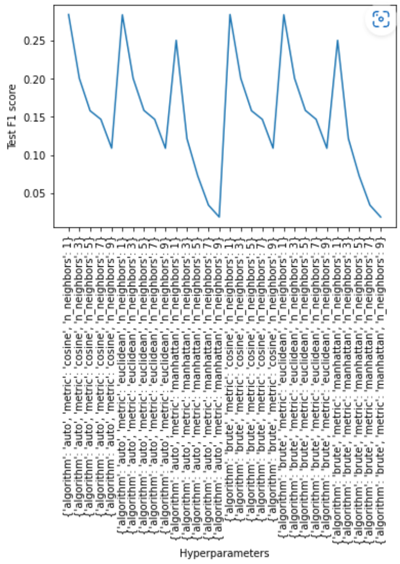
```
Number of documents in train set: 720
Number of documents in test set: 180
Number of words in vocabulary: 5419
Number of tokens in train set: 3534.921523238345
Number of tokens in test set: 861.3041535668966
Mean number of tokens per document in train set: 4.909613226719923
Mean number of tokens per document in test set: 4.785023075371648
```

Figure 10: displays the exploratory data analysis (EDA) conducted on the test and train data corpus, utilizing the clean text optimized for the model discussed in Table 5.

We used the **TfidfVectorizer()** function to transform our text data into a numerical matrix of TF-IDF (term frequency-inverse document frequency) features. This assigns a weight to each term in the corpus based on its frequency and informativeness according to Zipf's law. By doing so, we were able to represent each document as a vector in the feature space and train the K-Nearest Neighbours model to find the nearest neighbours for a given query document using clean text. We vectorized the text using TF-IDF instead of a bag-of-words representation because TF-IDF considers the importance of each word in the document based on its frequency and informativeness, while a bag-of-words representation treats all words equally. This allows the model to better capture the important words and their relationships in the document and can improve classification (Yang, 2017).

The K-Nearest Neighbours model is widely used classification algorithm known for its simplicity and flexibility (Guo et al., 2003). It involves finding the k nearest neighbours to a query point based on a distance metric and then predicting the class based on the majority of neighbours. This approach has been used in previous recommender systems, making it a suitable choice for our project (Li, 2017). Although we explored the use of Random Forest with altered **class_weight** parameter to handle unbalanced data and potentially overcome the limitations of our K-Nearest Neighbours model as a lazy learner, the random forest model resulted in a lower F1 score of 0.23. We attempted to optimize the model through grid search, but this did not improve performance. These results highlight the complexity of data science and the fact that not all models are suitable for every dataset. To identify the best model for a given task, it is important to experiment with different approaches and perform thorough evaluations.

We employed grid search to optimize the hyperparameters of our K-Nearest Neighbours model. This involved defining a hyperparameter grid with possible values for the number of neighbours, the algorithm, and the distance metric. Figure 11 implies that poor combinations of hyperparameters can significantly impact performance. To perform the cross-validated grid search with 5 folds and the F1 weighted scoring metric, we used the **GridSearchCV()** function.




```
Best hyperparameters: {'algorithm': 'auto', 'metric': 'cosine', 'n_neighbors': 1}
Best TEST F1 score: 0.2839678724352491
```

Figure 11 illustrates the process of tuning our KNN model to select the optimal parameters for training. Based on the results of the grid search, the best model was achieved using the following hyperparameters: algorithm = Auto, distance metric = Cosine, and number of neighbours = 1.

To assess the performance of our model, we computed various classification metrics such as F1 score, precision, recall, and accuracy for both the training and testing sets. We observed indications of overfitting, which is a common machine learning issue. This was evident in Figure 12 where all training metrics significantly outperformed the testing metrics.

```
Performance metrics on classification of text books
Test F1 score: 0.3949 - Train F1 score: 0.8847
Test Precision: 0.3859 - Train Precision: 0.8763
Test Recall: 0.4043 - Train Recall: 0.8931623931623932
Test Accuracy: 0.2778 - Train Accuracy: 0.8819
```

Figure 12: the performance metrics for our model are presented, revealing a Test F1 score of 0.3949 and a Train F1 score of 0.8847, along with other metrics indicating potential overfitting.

We evaluated the performance of our recommendation system using the F1 score using our ground truth user defined previously. To make recommendations, we utilized our trained model to identify the nearest neighbours for a given query document and returned the top 2 recommendations with their cosine similarity scores, allowing us to provide relevant recommendations based on distance. Figure 13 shows that our recommender system achieved an overall mean F1 score of 0.38 with 2 recommendations returned. This outperformed the naive model, which obtained an F1 score of only 0.09 (Figure 8). It is important to note that there may be errors in the subject data and that some recommendations could be valid.

```
Performance metric on recomnedation system using test data only
Mean F1 Score of Books: 0.3871296296296295
```

	Test_ISBN	Test_User	Recommended_ISBNs	Recommended_Users	Similarity_Scores	F1_Score
0	9780947225698	[DESIGN AND TECHNOLOGY_12]	[9780947225704, 9780947225667]	[[DESIGN AND TECHNOLOGY_12], [DESIGN AND TECHN...	[0.3470323168185492, 0.3470323168185492]	1.000000
1	9780316382007	[ENGLISH_5, ENGLISH_4]	[9780553294385, 9780571056866]	[[ENGLISH_12], [ENGLISH_9, ENGLISH_8, ENGLISH_...	[0.36038968076248845, 0.1952351696183009]	0.000000
2	9780571336173	[ENGLISH_12]	[9780140422108, 9781740818377]	[[ENGLISH_12], [ENGLISH_12, ENGLISH_11]]	[0.2703387591360915, 0.19368248829545653]	1.000000
3	9780725334659	[ENGLISH_6]	[9781740202954, 9781740202992]	[[ENGLISH_0], [ENGLISH_2]]	[1.0, 1.0]	0.000000
4	9780141359410	[ENGLISH_7, ENGLISH_8]	[9780702235467, 9780099462217]	[[ENGLISH_5], [ENGLISH_12]]	[0.14024223984426398, 0.13828405674889854]	0.000000
...	...	...	...	...	...	...
175	9780190323226	[ENGLISH_6]	[9780190323172, 9780190323219]	[[ENGLISH_1, ENGLISH_3], [MATHEMATICS_5, ENGLI...	[1.0, 1.0]	0.000000
176	9780733970665	[LANGUAGES_11]	[9780733970672, 9780733969027]	[[LANGUAGES_11], [LANGUAGES_12]]	[1.0, 0.6976481859626077]	0.666667
177	9780224025720	[ENGLISH_4, ENGLISH_3]	[9780142401088, 9780380807345]	[[ENGLISH_4, ENGLISH_3], [ENGLISH_7]]	[0.1431992535545552, 0.1290509209858326]	0.666667
178	9780980874921	[SCIENCE_12, COMPUTER SCIENCE_12]	[9780170196826, 9781108413473]	[[LANGUAGES_9], [HISTORY_12]]	[0.34315459263548487, 0.3397013761691934]	0.000000
179	9781741353372	[MATHEMATICS_2]	[9781741353402, 9780190322809]	[[MATHEMATICS_5, ENGLISH_5, ENGLISH_4], [MATHE...	[1.0, 1.0]	0.666667

Figure 11: Demonstrates our optimized model that suggests two books based on Test_ISBN using cosine similarity and test data. However, there is a potential limitation in evaluating the model's ability using ground truth when subjects cover multiple year levels. For instance, ISBN 9780141359410 is linked with ENGLISH_7 and ENGLISH_3, but the recommendations are utilized by ENGLISH_5 and ENGLISH_12 users. As a result, the model's match might be deemed unsuccessful, whereas it could actually be appropriate in reality.

To address the issue of unbalanced data, we limited the scope of our recommender system by excluding users who occurred less than 30 times. While this decision improved the performance of our model by improving our F1 score to 0.55 (figure 13), it meant that our model would only be applicable to a smaller subset of users listed in figure 12, resulting in a more biased and limited scope model.

```
449
HISTORY_12: 31
ENGLISH_4: 34
SCIENCE_11: 34
ENGLISH_10: 44
ENGLISH_11: 45
MATHEMATICS_12: 47
ENGLISH_9: 48
LANGUAGES_12: 56
SCIENCE_12: 73
ENGLISH_12: 106
```

Figure 12: demonstrates our attempt to enhance the balance of our model by excluding observations where the user count was low. Consequently, we reduced our dataset by half and restricted the scope of our new model.By limiting the scope of users, we were able to enhance the performance of our recommender system, as illustrated in figure 15. However, to further improve our study, we suggest collecting more data for non-core subjects and increasing the sampling rate of lower year levels.

Figure 3 shows that high school and core subjects in Australia have the largest user base, suggesting that a more targeted, narrower model could be more beneficial than a broad one resulting in better performance (figure 13). Thus, the question of whether we need a narrow or broad model should be considered.

```
Performance metric on recomnedation system using test data only
Mean F1 Score of Books: 0.558641975308642
```

	Test_ISBN	Test_User	Recommended_ISBNs	Recommended_Users	Similarity_Scores	F1_Score
0	9781488612015	[SCIENCE_12]	[9781316422953, 9781488621468]	[[MATHEMATICS_12], [MATHEMATICS_12]]	[1.0, 0.2094100955772895]	0.000000
1	9780060787325	[LANGUAGES_11, LANGUAGES_9, LANGUAGES_8, LANGU...	[9780008320065, 9780190311308]	[[LANGUAGES_12], [ENGLISH_9, ENGLISH_10]]	[0.43563784685175366, 0.3295526980295602]	0.166667
2	9781857989380	[ENGLISH_12]	[9780375866272, 9780142401088]	[[ENGLISH_10], [ENGLISH_4, ENGLISH_3]]	[0.22430691179580398, 0.16016164395853494]	0.000000
3	9780063074293	[LANGUAGES_12]	[9780008320065, 9787100059459]	[[LANGUAGES_12], [LANGUAGES_12]]	[0.5284786723214503, 0.5037369131978402]	1.000000
4	9780143104407	[ENGLISH_11]	[9780573040191, 9780143129516]	[[ENGLISH_9], [ENGLISH_12]]	[1.0, 0.2652319806242265]	0.000000
...	...	...	...	...	...	...
85	9780140180909	[ENGLISH_12]	[9780679781516, 9780970181671]	[[ENGLISH_12], [ENGLISH_12]]	[0.13118939357293158, 0.12085380050869765]	1.000000
86	9781316502648	[SCIENCE_9, SCIENCE_12, SCIENCE_11, SCIENCE_10]	[9781316502662, 9781108908689]	[[MATHEMATICS_11, MATHEMATICS_12], [SCIENCE_12]]	[0.9459768201711796, 0.6334949501650804]	0.333333
87	9780812550702	[ENGLISH_9]	[9780140364521, 9780076716753]	[[ENGLISH_4], [SCIENCE_12]]	[0.20173461277018112, 0.14928824756166992]	0.000000
88	9780863158186	[MATHEMATICS_12]	[9781925489552, 9781925489552]	[[MATHEMATICS_11, MATHEMATICS_12], [MATHEMATIC...	[0.5105209905047333, 0.5105209905047333]	1.000000
89	9780190319342	[ENGLISH_7, ENGLISH_9, ENGLISH_8, ENGLISH_10]	[9780190308674, 9780190311308]	[[ENGLISH_7, ENGLISH_8, ENGLISH_10], [ENGLISH_...	[0.4130102564189829, 0.2911504433986467]	0.750000

Figure 13: displays our second optimized model that recommends two books based on Test_ISBN utilizing cosine similarity and using a model which was trained on a more balanced dataset. Nevertheless, a significant limitation of this approach is that it is solely applicable to users listed in Figure 14, though raises the question if a more narrow model is better suited given the larger user base.

Concluding Remarks

In conclusion, our experience in developing a book recommendation system utilizing NLP techniques has demonstrated the potential of this tool for data scientists. The process involved multiple steps, including data acquisition, cleaning, and NLP pre-processing, as well as selecting optimal features and hyperparameters. We were able to improve our model's performance further by removing users with low sample counts to address the unbalanced nature of the data. However, our model still had limitations, including overfitting and potential bias from user exclusions or the accuracy of our ground truth user. Despite these limitations, our model outperformed the naive approach, with an F1 score of 0.38 compared to 0.09; or a limited model with an F1 score of 0.55. Future studies may benefit from incorporating additional data sources or redefining user groups to improve model performance. Overall, our experience highlights the potential of NLP-powered recommendation systems to provide valuable insights and recommendations through a unique approach.

Reference

Datagen (no date) Ground truth in machine learning: Process & key challenges, Datagen. Available at: https://datagen.tech/guides/data-training/ground-truth/ (Accessed: March 27, 2023).

Department of Education (no date) Australian curriculum, Department of Education. Available at: https://www.education.gov.au/australian-curriculum (Accessed: March 22, 2023).

Google (no date) Performance tips | google books apis | google developers, Google. Google. Available at: https://developers.google.com/books/docs/v1/performance (Accessed: March 29, 2023).

Guo, G. et al. (2003) KNN model-based approach in classification, SpringerLink. Springer Berlin Heidelberg. Available at: https://link.springer.com/chapter/10.1007/978-3-540-39964-3_62 (Accessed: March 29, 2023).

Hider, P. (2018) Information resource description: Creating and managing metadata, Amazon. Facet Publ. Available at: https://www.amazon.com/Information-Resource-Description-Creating-Managing/dp/1783302232 (Accessed: March 28, 2023).

Juviler, J. (2022) Rest apis: How they work and what you need to know, HubSpot Blog. HubSpot. Available at: https://blog.hubspot.com/website/what-is-rest-api (Accessed: March 29, 2023).

Li, S. (2017) Building a book Recommender System – the basics, KNN and matrix factorization, DataScience+. Available at: https://datascienceplus.com/building-a-book-recommender-system-the-basics-knn-and-matrix-factorization/ (Accessed: March 29, 2023).

Moam Grammer (2015) VCE Subjects Comparable to NSW HSC Subjects. Available at: https://www.moamagrammar.nsw.edu.au/wp-content/uploads/2015/02/UAC-subject-conversion.pdf (Accessed: March 22, 2023).

Nair, A. (2022) Baseline models: Your guide for model building, Medium. Towards Data Science. Available at: https://towardsdatascience.com/baseline-models-your-guide-for-model-building-1ec3aa244b8d (Accessed: February 20, 2023)

National, state and territory population, September 2022 (2022) Australian Bureau of Statistics. Available at: https://www.abs.gov.au/statistics/people/population/national-state-and-territory-population/latest-release (Accessed: March 22, 2023).

Wilame (2023) Why is removing stop words not always a good idea, Medium. Medium. Available at: https://medium.com/@limavallantin/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214 (Accessed: March 29, 2023).

Wisdomml (2022) What are stop words in NLP and why we should remove them?, Wisdom ML. Available at: https://wisdomml.in/what-are-stopwords-in-nlp-and-why-we-should-remove-them/ (Accessed: March 28, 2023).

Yablon, D. (2020) Machine learning 3: The importance of subject matter expertise. Available at: https://analyticalscience.wiley.com/do/10.1002/micro.3507 (Accessed: March 29, 2023).

Yu-Yen Ou, Hao-Geng Hung and Yen-Jen Oyang (2006) “A study of supervised learning with multivariate analysis on unbalanced datasets,” The 2006 IEEE International Joint Conference on Neural Network Proceedings [Preprint]. Available at: https://doi.org/10.1109/ijcnn.2006.247014.

Yang, Y. (2017) “Research and realization of internet public opinion analysis based on improved TF - IDF algorithm,” 2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES) [Preprint]. Available at: https://doi.org/10.1109/dcabes.2017.24.

Appendix – Python Code

```
# ## Background
# Natural language processing (NLP) began in the 1950s as the intersection between artificial intelligence and linguistics became prominent. There are various applications of NLP, with familiar applications such as sentiment analysis and text classification. For the purpose of this paper we will be exploring the application of NLP to build a content-based recommendation system. The goal of content-based filtering is to make an item-based recommendation to a user based on attributes of the item. For our example we will define an item as a textbook used in an Australian school and the attribute being features such as (1) description, (2) author and (3) publisher when available; with our user being defined as someone who needs a textbook for a particular subject (i.e. Year 12 English). All analysis was conducted using Jupyter Notebook with Python version 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)].

# In[1]:


# Import required packages
import sys
import random
import nltk
import pandas as pd
import matplotlib.pyplot as plt
import requests
import json
import tqdm
import time
import datetime
import numpy as np
import seaborn as sns
from collections import Counter
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
from sklearn.preprocessing import MultiLabelBinarizer
import warnings
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.exceptions import UndefinedMetricWarning
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, precision_score, recall_score,
accuracy_score
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from bs4 import BeautifulSoup
import re

# Download NLTK resources
nltk.download('stopwords')
nltk.download('punkt')

# Define stop words
stop_words = set(stopwords.words('english'))
stop_words.add('etc/')
```

```
# Define file paths
file_path = 'C:/Users/jayde/OneDrive - James Cook
University/Desktop/MA5851/Assignment 1/'
file_name = 'MA5851_SP82_2023_A1_Data.xlsx'

# Functions for text cleaning and analysis

def clean_html(text):
    # Remove HTML tags
    soup = BeautifulSoup(text, 'html.parser')
    clean_text = soup.get_text()
    # Remove any remaining non-alphabetic characters and convert to
lowercase
    clean_text = re.sub('[^a-zA-Z]', ' ', clean_text).lower()
    # Remove any extra whitespace
    clean_text = ' '.join(clean_text.split())
    return clean_text

def clean_text(text):
    # Remove backslash-apostrophe
    text = re.sub("\'", "", text)
    # Remove everything except alphabets
    text = re.sub("[^a-zA-Z]"," ",text)
    # Remove whitespaces
    text = ' '.join(text.split())
    # Convert text to lowercase
    text = text.lower()
    return text

def remove_stopwords_and_stem(text):
    # Remove stop words and stem words
    stemmer = PorterStemmer()
    tokens = word_tokenize(text.lower())
    no_stopword_text = [stemmer.stem(w) for w in tokens if not w in
stop_words]
    return ' '.join(no_stopword_text)

def freq_words(x, terms = 30):
    # Get frequency distribution of words in text
    all_words = ' '.join([text for text in x])
    all_words = all_words.split()
    fdist = nltk.FreqDist(all_words)
    words_df = pd.DataFrame({'word':list(fdist.keys()),
'count':list(fdist.values())})
    # Select top n most frequent words and plot them
    d = words_df.nlargest(columns="count", n=terms)
    plt.figure(figsize=(12,15))
    ax = sns.barplot(data=d, x="count", y="word")
    ax.set(ylabel='Word')
    plt.show()
print(sys.version)


# # Data overview:
# Data was obtained from the MA5851 Assessment 1 folder on learn JCU and
downloaded on 19th March 2023. The data was saved locally and imported into
python for data pre-processing and analysis. The provided data includes
information about the ISBN numbers of textbooks used in Australian schools,
as well as additional variables such as the unique identifier for the
school, the state in which the school resides in, the year level for which
the text book is used for, and the subject area for which the book is
```

```
assumed to be used in; noting that subject area information may be
inaccurate.
# Table 1 summarised the data dimensions and data types for each column.
The initial data set contained 5 columns and 1804 rows.
#

# In[2]:


# Load in isbn data sheet
isbn_data = pd.read_excel(f"{file_path}{file_name}",
sheet_name='ISBN_Data')
user_data = isbn_data
isbn_data.head()  # Cofirm loaded  correctly


# In[3]:


num_rows = isbn_data.shape[0]
num_cols = isbn_data.shape[1]

print('Number of rows:', num_rows)
print('Number of columns:', num_cols)

# print column names and data types
print(isbn_data.dtypes)


# # Sample Data Pre-Processing / Sample Data Analysis (EDA)
# - Step 1: Correct data type
# - Step 2: Duplicate observations (multiple classes at a school possibly)
# - Step 3: Review and correct subjects
# - Step 4: Do we remove low observations based on our user?
#
#
# To optimize our NLP recommendation system, we performed four pre-
processing steps on the sample data from the JCU folder. These steps
ensured that the data was of the correct data type, free of duplications,
and had any errors reviewed and corrected. We converted School_ID, State,
Year, and Subject to categorical data types and removed 127 duplicates from
the dataset. Although we hypothesized that this could be due to a school
having multiple classes for a particular subject. We also reviewed the
distribution of subjects and noticed that English was listed in three
different ways. To standardize the data, we renamed ‘english’ and ‘ENGLISH
LITERARY STUDIES’ as ‘ENGLISH’ and assumed that all other subjects appeared
correctly.
# In data science, a common challenge is that data is often not initially
in a usable format, which requires significant time to transform. We
suggested a solution for our NLP classification recommendation model, which
involved creating a data frame with a unique ISBN in each row and a second
column listing known users of the corresponding textbook. This would enable
us to recommend textbooks based on features such as author, description,
title, and category and evaluate the model's performance using a ground
truth method (Datagen, ND). We can use the known users of each textbook to
compare the model's recommended textbooks and assess its accuracy and
effectiveness (Datagen, ND).
#

# In[4]:
```

```
# STEP 1
# Convert data type to categorical as required
isbn_data['School_ID'] = isbn_data['School_ID'].astype('category')
isbn_data['State'] = isbn_data['State'].astype('category')
isbn_data['Year'] = isbn_data['Year'].astype('category')
isbn_data['Subject'] = isbn_data['Subject'].astype('category')

# Confrim above worked
print(isbn_data.dtypes)


# In[5]:


# STEP 2
num_rows_before = isbn_data.shape[0]

isbn_data_2 = isbn_data.drop_duplicates(keep='first')

num_rows_after = isbn_data_2.shape[0]
num_rows_dropped = num_rows_before - num_rows_after

print('Number of rows dropped:', num_rows_dropped)

print(isbn_data_2)


# In[6]:


# EDA: Produce some visuals to get an undertsanding for the data
subject_count = isbn_data_2['Subject'].value_counts()
state_count = isbn_data_2['State'].value_counts()
year_count = isbn_data_2['Year'].value_counts()
school_count = isbn_data_2['School_ID'].value_counts()


# Create subplots for each count variable
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

# Subject count
axs[0, 0].bar(subject_count.index, subject_count.values)
axs[0, 0].set_title('Subject count')
axs[0, 0].tick_params(axis='x', rotation=90) # Add this line to rotate the
labels


# State count
axs[0, 1].bar(state_count.index, state_count.values)
axs[0, 1].set_title('State count')

# Year count
axs[1, 0].bar(year_count.index, year_count.values)
axs[1, 0].set_title('Year count')

# School count
axs[1, 1].bar(school_count.index, school_count.values)
axs[1, 1].set_title('School count')

# Adjust spacing between subplots
```

```
plt.tight_layout()

# Show the plot
plt.show()
print(subject_count)


# In[7]:


# Step 3: Correct subjects
isbn_data_2['Subject'] = isbn_data_2['Subject'].str.strip()

# Replace 'english' and 'ENGLISH LITERARY STUDIES' with 'ENGLISH'
isbn_data_2.loc[isbn_data_2['Subject'].isin(['english', 'ENGLISH LITERARY
STUDIES']), 'Subject'] = 'ENGLISH'

# Count the number of occurrences of each subject
subject_count = isbn_data_2['Subject'].value_counts()
print(subject_count)

# Identify subjects with fewer than 10 occurrences
#mask = isbn_data_2['Subject'].isin(subject_count.index[subject_count <
10])

# Filter the rows of the DataFrame using the mask
#isbn_data_2 = isbn_data_2[~mask]

# Print the updated value counts of the Subject column
#subject_count = isbn_data_2['Subject'].value_counts()
#print(subject_count)


# # Who are our users of NLP recomendation?
# - user_1: Could be based on year & subject
# - user_2: Could be based on subject
# - user_3: Could be based on year level
#
# We consulted the literature to gain domain expertise on the Australian
education system, which helped us create contextually informed models for
more accurate predictions. For our recommendation system, we selected users
based on subject and year level to recommend textbooks appropriate for
their teaching level for our ground truth (Moam Grammer, 2015; Datagen,
ND). To validate our approach, we counted the number of defined users per
ISBN and found that some subject textbooks spanned multiple year levels.
However, during desktop research, we discovered that five specific ISBNs
also spanned vastly different subjects, such as dictionaries and blank
notebooks (as shown in Figure 2 for illustrative purposes). Therefore, we
removed them from our analysis to reduce noise & API calls in the next
section, resulting in 1066 unique ISBNs. At this stage we retained all
users.

# In[8]:



isbn_data_3 = isbn_data_2[['Year', 'Subject', 'ISBN']]
isbn_data_3['Year'] = isbn_data_3['Year'].astype(str)  # convert Year
column to string data type
isbn_data_3['Subject'] = isbn_data_3['Subject'].astype(str)  # convert
Subject column to string data type
```

```
isbn_user_binary = isbn_data_3.groupby('ISBN').apply(lambda x:
list(set(x['Subject'] + '_' + x['Year'])))
#isbn_user_binary = isbn_data_3.groupby('ISBN').apply(lambda x:
list(set(x['Subject'])))
#isbn_user_binary = isbn_data_3.groupby('ISBN').apply(lambda x:
list(set(x['Subject'])))
isbn_user_binary = isbn_user_binary.reset_index()
isbn_user_binary = isbn_user_binary.rename(columns={0: 'User_type'})
print(isbn_user_binary)


# Get the counts of each user type and sort by frequency
counts =
isbn_user_binary['User_type'].explode().value_counts().sort_values()

# Plot the top 10 and bottom 10 user types
plt.barh(counts.index[-10:], counts[-10:])
plt.barh(counts.index[:10], counts[:10])

# Add labels and title
plt.xlabel('Frequency')
plt.ylabel('User Type')
plt.title('Top and Bottom 10 User Types by Frequency')

# Display the plot
plt.show()

user_type_counts = isbn_user_binary['User_type'].explode().value_counts()
print(user_type_counts)



# In[ ]:





# In[9]:


# Sort the DataFrame by the length of the 'User_type' column in descending
order
sorted_df = isbn_user_binary.sort_values(by=['User_type'], key=lambda x:
x.str.len(), ascending=False)

# Filter where more then 3 eleements in dict
filtered_df = sorted_df[sorted_df['User_type'].apply(lambda x: len(x)) > 3]

filtered_df

# Possible ISBNs to inviestage
# 9780190303488
# 9780730389422
# 9798708474995
# 9780732979966
# 9780648237334
# Why are these an issue:
# - The subjects listed appear quite different to one another.
# - given this list is small we will do some analysis to make a decsion to
drop or keep: this will also reduce API calls
```

```
# In[10]:


# Create a list of ISBNs to be removed based on desktop research
isbns_to_remove = [9780190303488, 9780730389422, 9798708474995,
9780732979966, 9780648237334]

# Filter out the rows containing these ISBNs
isbn_user_binary =
isbn_user_binary[~isbn_user_binary['ISBN'].isin(isbns_to_remove)]


isbn_list = isbn_user_binary["ISBN"].tolist() # Used for API Call
# We are now happy with the format of our sample data and will save it out
for use later
isbn_user_binary.to_csv('jcu_sample_data_proccessed.csv', index = False)


# In[11]:


# ISBN list for API call
len(isbn_list)


# # PART 2: API collection of data
# APIs are essential tools for data scientists working in the technology
field. They are used in various online activities, such as sharing posts
and making payments. Data scientists can use APIs to retrieve data,
including text data using the ISBN which we will demonstrate to enrich our
data pool. Understanding the basics of HTTP, RESTful services, and data
formats such as JSON and XML is crucial for effective API use. It is also
important to be aware of API limitations, such as rate limiting and data
accuracy issues, and how to mitigate them through techniques like partial
requests and caching. Consulting technical documentation before making API
requests is vital.

# In[270]:


def get_google_book_info(isbn):
    endpoint = "https://www.googleapis.com/books/v1/volumes"

    # Add field to request to limit object return
    params = {"q": "isbn="+str(isbn),
              "maxResults": 1,
              "fields": "totalItems,items(volumeInfo(title, authors,
publishedDate, publisher, description, categories))" # Partial request
    }

    # Send GET request to API
    response = requests.get(endpoint, params=params).json()

    if "items" in response: # Check if "items" key is present in response,
this prevents crashing
        if (response["totalItems"] > 0):
            for book in response["items"]:
                volume = book["volumeInfo"]
                title = volume["title"]
```

```
                # Not all API request will contain the following:
                if "authors" in volume.keys():
                    authors = volume['authors']
                else:
                    authors = ""
                if "publishedDate" in volume.keys():
                    published = volume["publishedDate"]
                else:
                    published = ""
                if "publisher" in volume.keys():
                    publisher = volume["publisher"]
                else:
                    publisher = ""
                if "description" in volume.keys():
                    description = volume["description"]
                else:
                    description = ""
                if "categories" in volume.keys():
                    categories = volume["categories"]
                else:
                    categories = ""

                return {"isbn":isbn, "title": title, "authors": authors,
"published": published, "publisher": publisher, "description":description,
"categories": categories }
        else:
            return {"isbn":isbn, "title": "", "authors": "", "published":
"", "publisher": "", "description": "", "categories": "" }
    else:
        return {"isbn":isbn, "title": "", "authors": "", "published": "",
"publisher": "", "description": "", "categories": "" }

print(get_google_book_info(9780385486804))


# In[78]:


# Save as dataframe all output; and keep track of errors
import time
import datetime # Used to keep track of star/end time

book_data = []

start_time = datetime.datetime.now()
# Display progress bar as we list through unique list
for isbn in tqdm.tqdm(isbn_list, desc=f"Retrieving book information for
ISBNs {isbn_list[0]} to {isbn_list[-1]}"):
    book_info = get_google_book_info(isbn)
    book_data.append(book_info)
    time.sleep(4) # limit API, as would fail to connect due to multiple
requests - limitation


end_time = datetime.datetime.now()

# Calculate the time taken to loop through all ISBNs
time_taken = end_time - start_time
print(f"Time taken: {time_taken}") # 1 hour and 23 mins
```

```
# Convert dict to dataframe using pandas and save csv for faster/easier
retrival if needed later
df = pd.DataFrame(book_data)
df.to_csv('google_api_data.csv', index=False)


# In[ ]:





# # Clean Google API data
# To ensure uniformity of data format, we utilized NLTK and BeautifulSoup
for HTML element cleaning to standardize and clean the columns of both the
Google and Trove APIs. We developed custom functions as specified at the
start of the script. To prevent potential errors stemming from disparities
in metadata quality, we standardized the columns individually before
consolidating them into a single document.
#
# - Convert to lower
# - merge words, this is because 'James Smith' and 'James Brown' would
result in 2 counts for james when they are different authors

# In[71]:



df_google = pd.read_csv('google_api_book_detail.csv', na_values=['NA', '-
999'])
# fill NaN values with empty string
df_google.fillna('', inplace=True)

# For our analysis later
df_google['clean_categories'] =
df_google['categories'].astype(str).apply(lambda x: clean_text(x))
df_google['clean_categories'] =
df_google['clean_categories'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))

df_google['clean_title'] = df_google['title'].astype(str).apply(lambda x:
clean_text(x))
df_google['clean_title'] =
df_google['clean_title'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))

# Treat author as one unique name, Trove only returned 1x Author
df_google['clean_author'] = df_google['authors'].str.replace(' ', '')
df_google['clean_author'] =
df_google['clean_author'].astype(str).apply(lambda x: clean_text(x))
df_google['clean_author'] =
df_google['clean_author'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))

df_google['clean_publisher'] =
df_google['publisher'].astype(str).apply(lambda x: clean_text(x))
df_google['clean_publisher'] =
df_google['clean_publisher'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))

df_google['clean_description'] =
```

```
df_google['description'].astype(str).apply(lambda x: clean_text(x))
df_google['clean_description'] =
df_google['clean_description'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))

freq_words(df_google['description'], 50)


# In[66]:


df2 = pd.read_csv('google_api_book_detail.csv', na_values=['NA', '-999'])
missing_values_count_google_book = df2.isna().sum()
print(missing_values_count_google_book)

# Missing data: We want to try and locate description data
missing_description_mask = df2['description'].isna()
missing_description_list = df2[missing_description_mask]['title'].tolist()

# We want to try and locate title data:
missing_title_mask = df2['title'].isna()
missing_title_list = df2[missing_title_mask]['isbn'].tolist()


# # Trove API Request
# During our analysis, we discovered that the information we needed was not
available in the Google Books API, and we encountered missing data. This is
a common challenge in data retrieval, and it is important for data
scientists to have backup plans in place to minimize the number of missing
observations, such as using other APIs. To enhance our data, we used the
Trove API to extract the book title when the description was missing and
the ISBN when the title was missing. We performed two Trove API requests,
the first one to retrieve missing titles for ISBNs that were not found in
Google, and the second one to search for titles using the results from the
initial Trove search and the Google search. However, we faced a limitation
with Trove, as we were unable to obtain a description when searching using
ISBN, as the snippet feature mostly returned empty values for ISBNs. This
highlights the importance of data scientists approaching challenges from
different angles to ensure a complete and enriched dataset is obtained and
trying different search parameters.

# In[166]:


# The goal here is to try and find description information as we consider
this the most important
def get_trove_book_info_title(book_name):
    # Define trove connection
    endpoint = 'https://api.trove.nla.gov.au/v2/result'

    # Define params
    params = {
        'q': str(book_name), # Can search on ISBN
        'zone': 'book', # Search in the book zone
        'key': "3df38mcts5aquk4l", # API key
        'encoding': 'json', # Default is xml
        'include' : 'tags=comments,workversions'
    }

    response = requests.get(endpoint, params=params)
    data = response.json()
```

```
    try:
        books = data['response']['zone'][0]['records']['work']
        if len(books) > 0:
            book = books[0]
            snippet = book.get('snippet', '')
            if 'version' in book and len(book['version']) > 0:
                version = book['version'][0]
                if 'record' in version and 'publisher' in
version['record']:
                    publisher = version['record']['publisher']
                else:
                    publisher = ''
                if 'record' in version and 'issued' in version['record']:
                    published = version['record']['issued']
                else:
                    published = ''
                if 'record' in version and 'subject' in version['record']
and len(version['record']['subject']) > 0:
                    categories = []
                    for subject in version['record']['subject']:
                        if isinstance(subject, dict) and 'value' in
subject:
                            categories.append(subject['value'])
                        else:
                            categories.append(subject)
                else:
                    categories = ''
            else:
                publisher = ''
                published = ''
                categories = ''
            author = book.get('contributor', '')
            if author:
                author_name = author[0].split(',')[0].strip()  # get last
name
                if len(author[0].split(',')) > 1:
                    author_first_initial =
author[0].split(',')[1].strip().split()[0]  # get first name/initial
                    formatted_author = f"{author_first_initial}
{author_name}"  # combine first and last name to match google API
                else:
                    formatted_author = author_name
            else:
                formatted_author = ''
            return {'title': book_name, 'description': snippet,
'publisher': publisher, 'published': published, 'authors':
formatted_author, 'categories': categories}
        else:
            return None
    except KeyError:
        return None



# In[69]:


# Extract title from trove, then we will do another search on titles to
find description via snippet
def get_trove_book_info_isbn(isbn):
```

```
    # Define trove connection
    endpoint = 'https://api.trove.nla.gov.au/v2/result'

    # Define params
    params = {
        'q': str(isbn), # Can search on ISBN
        'zone': 'book', # Search in the book zone
        'key': "3df38mcts5aquk4l", # API key
        'encoding': 'json', # Default is xml
        'include' : 'tags=comments,workversions'
    }
    response = requests.get(endpoint, params=params)
    data = response.json()
    if 'work' in data['response']['zone'][0]['records']:
        book_info = data['response']['zone'][0]['records']['work']
        book = book_info[0]
        full_title = book.get('title', '')
        title = full_title.split(' / ')[0] #
        return{'isbn': isbn, 'title': title}
get_trove_book_info_isbn(9781316606735)



book_info_list = []

start_time = datetime.datetime.now()
for isbn in tqdm.tqdm(missing_title_list, desc=f"Retrieving book
information for ISBNs {missing_title_list[0]} to {missing_title_list[-
1]}"):
    try:
        book_info = get_trove_book_info_isbn(isbn)
        book_info_list.append(book_info)
    except:
        print(f'ISBN ERROR {isbn}')
    time.sleep(2) # Prevent getting blocked on server
end_time = datetime.datetime.now()
time_taken = end_time - start_time
print(f"Time taken: {time_taken}")

book_info_list = list(filter(None, book_info_list))
trove_title_data = pd.DataFrame(book_info_list)
titles_list = trove_title_data['title'].tolist()
titles_list


# In[181]:


# Now we have missing titles from google found in trove, we will search for
the reminder information using title only

book_info_list = []

start_time = datetime.datetime.now()
for title in tqdm.tqdm(titles_list, desc=f"Retrieving book information for
missing books"):
    try:
        # Search for book information using title
        book_info = get_trove_book_info_title(title)

        # Find the ISBN associated with the title from df2
```

```
        isbn = trove_title_data.loc[trove_title_data['title'] == title,
'isbn'].values[0]

        # Add the ISBN to the book information dictionary
        book_info['isbn'] = isbn

        # Append the book information to the list
        book_info_list.append(book_info)
    except:
        print(f'Error retrieving book information for {title}')

    # Sleep to prevent getting blocked by the server
    time.sleep(2)

end_time = datetime.datetime.now()
time_taken = end_time - start_time
print(f"Time taken: {time_taken}")

df_trove_description = pd.DataFrame(book_info_list)
df_trove_description


# In[70]:



book_info_list = []


start_time = datetime.datetime.now()
for title in tqdm.tqdm(missing_description_list, desc=f"Retrieving book
information for missing books"):
    try:
        # Search for book information using title
        book_info = get_trove_book_info_title(title)

        # Find the ISBN associated with the title from df2
        isbn = df2.loc[df2['title'] == title, 'isbn'].values[0]

        # Add the ISBN to the book information dictionary
        book_info['isbn'] = isbn

        # Append the book information to the list
        book_info_list.append(book_info)
    except:
        print(f'Error retrieving book information for {title}')

    # Sleep to prevent getting blocked by the server
    time.sleep(2)

end_time = datetime.datetime.now()
time_taken = end_time - start_time
print(f"Time taken: {time_taken}")

df_google_missing = pd.DataFrame(book_info_list)
df_google_missing


# In[212]:
```

```
# Merge our trove data together
#combined_trove_df = pd.concat([df_trove_description, df_google_missing], ignore_index=True)
#combined_trove_df.to_csv('trove_api_data.csv', index=False)


# In[209]:


#combined_trove_df


# # Clean trove API data
# To ensure uniformity of data format, we utilized NLTK and BeautifulSoup for HTML element cleaning to standardize and clean the columns of both the Google and Trove APIs. We developed custom functions as specified at the start of the script. To prevent potential errors stemming from disparities in metadata quality, we standardized the columns individually before consolidating them into a single document.

# In[73]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
import requests
import re
from collections import Counter
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize


# Read in Trove API data
df_trove = pd.read_csv('trove_api_data.csv', na_values=['NA', '-999'])

# Fill missing values with empty string
df_trove.fillna('', inplace=True)



df_trove['clean_categories'] = df_trove['categories'].astype(str).apply(lambda x: clean_text(x))
df_trove['clean_categories'] = df_trove['clean_categories'].astype(str).apply(lambda x: remove_stopwords_and_stem(x))

df_trove['clean_title'] = df_trove['title'].astype(str).apply(lambda x: clean_text(x))
df_trove['clean_title'] = df_trove['clean_title'].astype(str).apply(lambda x: remove_stopwords_and_stem(x))

# Treat author as one unique name, Trove only returned 1x Author
df_trove['clean_author'] = df_trove['authors'].astype(str).apply(lambda x: clean_text(x))
df_trove['clean_author'] = df_trove['clean_author'].astype(str).apply(lambda x: remove_stopwords_and_stem(x))
df_trove['clean_author'] = df_trove['clean_author'].str.replace(' ', '')
```

```
df_trove['clean_publisher'] =
df_trove['publisher'].astype(str).apply(lambda x: clean_text(x))
df_trove['clean_publisher'] =
df_trove['clean_publisher'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))

df_trove['clean_description'] = df_trove['description'].apply(clean_html) #
HTML enconced, need to clean
freq_words(df_trove['clean_description'], 50)
df_trove['clean_description'] =
df_trove['clean_description'].astype(str).apply(lambda x: clean_text(x))
df_trove['clean_description'] =
df_trove['clean_description'].astype(str).apply(lambda x:
remove_stopwords_and_stem(x))



df_trove


# In[ ]:





# # Augment Google API with Trove
# At this stage both our Google and Trove data was in a similar structure,
and we combined the two sources together as summarised by the rules below
via the pd.merge() function.
#
#
# Merge based on thse rules:
# - Clean title: If Google missing merge Trove
# - Clean Author: If google missing merge Trove
# - Clean Publisher: If Google missing merge Trove
# - Clean Categories: If Trove missing merge Google
# - Clean description: If Google missing merge Trove
#
#
#


# In[74]:


merged_df = pd.merge(df_google, df_trove, on='isbn', how = 'outer')
merged_df
print(merged_df.columns)


# Merge google, trove
merged_df = pd.merge(df_google, df_trove, on='isbn', how = 'outer')

# fill in missing values with data from the other data frame
merged_df['clean_title_x'].fillna(merged_df['clean_title_y'], inplace=True)
merged_df['clean_author_x'].fillna(merged_df['clean_author_y'],
inplace=True)
merged_df['published_x'].fillna(merged_df['published_y'], inplace=True)
merged_df['clean_publisher_x'].fillna(merged_df['clean_publisher_y'],
inplace=True)
merged_df['clean_categories_y'].fillna(merged_df['clean_categories_x'],
```

```
inplace=True) # Will retain categories y as doment as it has more info
then google, this will be useful if description is missing
merged_df['clean_description_x'].fillna(merged_df['clean_description_y'],
inplace=True)

#merged_df.to_csv('merged_df.csv', index=False)
# drop the redundant columns from the merged data frame
# drop the redundant columns from the merged data frame
merged_df.drop(['clean_title_y', 'clean_author_y', 'published_y',
'clean_publisher_y', 'clean_categories_x' , 'clean_description_y'], axis=1,
inplace=True)





# Rename the columns to remove the suffix '_y' or '_x'
merged_df.rename(columns={
    'clean_title_x': 'title',
    'clean_author_x': 'authors',
    'clean_published_x': 'published',
    'clean_publisher_x': 'publisher',
    'clean_categories_y': 'categories',
    'clean_description_x': 'clean_description'
}, inplace=True)


# drop the rows where 'title' is missing
merged_df = merged_df[['isbn', 'title', 'clean_description', 'authors',
'publisher', 'categories']]
merged_df['cat_des'] = merged_df['clean_description'].astype(str) + ' ' +
merged_df['categories'].astype(str)

merged_df.dropna(subset=['title'], inplace=True)

merged_df.replace('', np.nan, inplace=True)

# Drop rows with missing titles
merged_df.dropna(subset=['title'], inplace=True) # We consider this crucial
for our ML model

# Count the number of unique ISBNs
unique_isbns = merged_df['isbn'].nunique()

# Print the results
print('Number of rows after dropping missing titles:', len(merged_df))
print('Number of unique ISBNs:', unique_isbns)

na_count = merged_df.isna().sum()
print(na_count)
len(merged_df)
merged_df.to_csv('clean_merged_api_data.csv', index=False)


# # Deciding on features
# We introduced a new feature called "clean text," which was a combination
of the title, categories, and description. Through a trial-and-error
process, we tested various combinations of text features and selected the
one that resulted in the highest test F1 score for our out-of-box model
(default parameters).
```

```
# In[12]:


merged_df = pd.read_csv('clean_merged_api_data.csv')


# In[13]:


# Create a text column:
#merged_df['text'] = merged_df['title'].astype(str) + ' ' +
merged_df['authors'].astype(str) + ' ' +
merged_df['publisher'].astype(str) + ' ' +
merged_df['description'].astype(str)+ ' ' +
merged_df['categories'].astype(str)
merged_df['clean_text1'] = merged_df['title'].astype(str)
merged_df['clean_text2'] = merged_df['title'].astype(str) + ' ' +
merged_df['clean_description'].astype(str)
merged_df['clean_text3'] = merged_df['title'].astype(str) + ' ' +
merged_df['authors'].astype(str)
merged_df['clean_text4'] = merged_df['title'].astype(str) + ' ' +
merged_df['categories'].astype(str)
merged_df['clean_text5'] = merged_df['title'].astype(str) + ' ' +
merged_df['publisher'].astype(str)
merged_df['clean_text6'] = merged_df['title'].astype(str) + ' ' +
merged_df['categories'].astype(str) + ' ' +
merged_df['clean_description'].astype(str)
merged_df['clean_text7'] = merged_df['title'].astype(str) + ' ' +
merged_df['categories'].astype(str) + ' ' +
merged_df['clean_description'].astype(str) + ' ' +
merged_df['authors'].astype(str)
merged_df['clean_text8'] = merged_df['title'].astype(str) + ' ' +
merged_df['categories'].astype(str) + ' ' +
merged_df['clean_description'].astype(str) + ' ' +
merged_df['authors'].astype(str)+ ' ' + merged_df['publisher'].astype(str)


# Through tiral and error clean_text_6 was the best combination based on
validation F1



# # EDA On Text Corpus
# At this stage we conducted some EDA on our clean text, we explored most
common author, most common category and most common publisher and observed
the following characteristics (fig 7):
# -    Most common Authors: Chris Linthorn & William Shakespear
# -    Most common Category: Languages and Studies
# -    Most common publisher: Cambridge University Press
#

# In[14]:


# Fill na so we can do EDA
merged_df['title'] = merged_df['title'].fillna('')
merged_df['clean_description'] = merged_df['clean_description'].fillna('')
merged_df['authors'] = merged_df['authors'].fillna('')
merged_df['publisher'] = merged_df['publisher'].fillna('')
merged_df['categories'] = merged_df['categories'].fillna('')
print("Most Common Author")
```

```
freq_words(merged_df['authors'], 5)


# In[15]:


print("Most Common Categories")
freq_words(merged_df['categories'], 5)


# In[16]:


merged_df['merged_publisher'] = merged_df['publisher'].str.replace(' ', '')
print("Most Common Publisher")
freq_words(merged_df['merged_publisher'], 5)


# In[17]:




# Reviewing graph we need to remove some words which have not been removed
stop_words.add('nan')
stop_words.add('e')


# Best model
merged_df['clean_text'] = merged_df['clean_text6'].apply(lambda x:
remove_stopwords_and_stem(x))


# In[18]:


freq_words(merged_df['clean_text'], 100)


# # Developing model
# In this stage of our analysis, we merged our user ground truth data with
our corpus of text using the pd.merge() function.
#

# In[19]:


df = pd.read_csv('jcu_sample_data_proccessed.csv', converters={'User_type':
eval})
merged_df_analysis = pd.merge(merged_df, df, left_on='isbn', right_on =
'ISBN', how = 'left')


# In[20]:


merged_df_analysis = merged_df_analysis[['isbn', 'clean_text',
'User_type']]
merged_df_analysis = merged_df_analysis.dropna(subset=['User_type']) # drop
rows with missing values - 5 we removed earlier
merged_df_analysis
```

```
# In[ ]:




# # KNN Model Development

# In[22]:


import collections

# Get a list of all user types
user_types = [user_type for row in merged_df_analysis['User_type'] for
user_type in row]

# Count the occurrences of each user type
user_type_counts = Counter(user_types)

# Remove user types that occur less than 10 times
user_type_counts = {k: v for k, v in user_type_counts.items() if v >= 5}

# Filter the rows of the dataframe to only include user types that occur at
least 10 times
filtered_df = merged_df_analysis[
    merged_df_analysis['User_type'].apply(
        lambda x: any(user_type in user_type_counts for user_type in x)
    )
]
print(len(filtered_df))

# Sort the user type counts by value in ascending order
sorted_counts = sorted(user_type_counts.items(), key=lambda x: x[1])

# Print the first 10 items (i.e., the user types with the smallest counts)
for user_type, count in sorted_counts[:20]:
    print(f"{user_type}: {count}")

# 5 Good Accuracy: Through a grid search we tuned number of users
filtered_df


# In[ ]:




# In[23]:


# Split the data into train and test sets
random.seed(123)

# Filter out UndefinedMetricWarning warnings
warnings.filterwarnings("ignore", category=UndefinedMetricWarning)

data = filtered_df.copy()
train_data, test_data = train_test_split(data, test_size=0.2,
```

```
random_state=10)


# Vectorize the clean_text using TF-IDF
vectorizer = TfidfVectorizer(max_features = 10000)
#vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_data['clean_text'])
X_test = vectorizer.transform(test_data['clean_text'])

# Encode the User_type using MultiLabelBinarizer
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(train_data['User_type'])
y_test = mlb.transform(test_data['User_type'])


# Get the vocabulary and number of words
vocabulary = vectorizer.get_feature_names()
num_words = len(vocabulary)

# Calculate the number of tokens in train and test sets
num_train_tokens = X_train.sum()
num_test_tokens = X_test.sum()

# Calculate the mean number of tokens per document
mean_train_tokens = num_train_tokens / len(train_data)
mean_test_tokens = num_test_tokens / len(test_data)

# Print summary statistics
print("Number of documents in train set:", len(train_data))
print("Number of documents in test set:", len(test_data))
print("Number of words in vocabulary:", num_words)
print("Number of tokens in train set:", num_train_tokens)
print("Number of tokens in test set:", num_test_tokens)
print("Mean number of tokens per document in train set:",
mean_train_tokens)
print("Mean number of tokens per document in test set:", mean_test_tokens)


# In[ ]:





# In[24]:


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer

# Split the data into train and test sets
random.seed(123)
train_data, test_data = train_test_split(filtered_df, test_size=0.2,
random_state=10)

# Create a TfidfVectorizer instance with desired parameters
vectorizer = TfidfVectorizer(stop_words='english')

# Fit the vectorizer on the training data
vectorizer.fit(train_data['clean_text'].values)
```

```python
# Get the vocabulary and number of words
vocabulary = vectorizer.get_feature_names()
num_words = len(vocabulary)

# Define the max_features parameter based on the number of words
max_features = min(num_words, 1000)

# Set the max_features parameter and transform the text data
vectorizer.max_features = max_features
X_train = vectorizer.transform(train_data['clean_text'].values)
X_test = vectorizer.transform(test_data['clean_text'].values)

# Encode the User_type using MultiLabelBinarizer
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(train_data['User_type'])
y_test = mlb.transform(test_data['User_type'])

# Calculate the number of tokens in train and test sets
num_train_tokens = X_train.sum()
num_test_tokens = X_test.sum()

# Calculate the mean number of tokens per document
mean_train_tokens = num_train_tokens / len(train_data)
mean_test_tokens = num_test_tokens / len(test_data)

# Print summary statistics
print("Number of documents in train set:", len(train_data))
print("Number of documents in test set:", len(test_data))
print("Number of words in vocabulary:", num_words)
print("Number of tokens in train set:", num_train_tokens)
print("Number of tokens in test set:", num_test_tokens)
print("Mean number of tokens per document in train set:",
mean_train_tokens)
print("Mean number of tokens per document in test set:", mean_test_tokens)


# In[ ]:





# In[25]:


# TUNE OUR MODEL PARAMATER
# Filter out UndefinedMetricWarning warnings
warnings.filterwarnings("ignore", category=UndefinedMetricWarning)
random.seed(123)

# Train our model
# Define the hyperparameters to tune
import random
random.seed(123)

param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9],
    'algorithm': ['auto', 'brute'],
    'metric': ['cosine', 'euclidean', 'manhattan']
}
```

```python
# Perform the grid search
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid,
cv=5, scoring='f1_weighted')
grid_search.fit(X_train, y_train)

# Get the results and the corresponding F1 scores
results = grid_search.cv_results_
f1_scores = results['mean_test_score']

# Plot the F1 scores for each combination of hyperparameters
fig, ax = plt.subplots()
ax.plot(range(len(f1_scores)), f1_scores)
ax.set_xticks(range(len(f1_scores)))
ax.set_xticklabels([str(params) for params in results['params']])
ax.set_xticklabels([str(params) for params in results['params']],
rotation=90)
ax.set_xlabel('Hyperparameters')
ax.set_ylabel('Test F1 score')
plt.show()

# Get the best F1 score and its corresponding hyperparameters
best_f1_score = grid_search.best_score_
best_params = grid_search.best_params_

# Print the best hyperparameters and their F1 score
print(f"Best hyperparameters: {best_params}")
print(f"Best TEST F1 score: {best_f1_score}")


# In[29]:


# WHAT ABOUT RANDOM FOREST? Could it be better at classifying?
# Train our model
# Define the hyperparameters to tune
random.seed(123)


param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False],
    'class_weight': ['balanced']
}

# Perform the grid search
grid_search = GridSearchCV(RandomForestClassifier(), param_grid=param_grid,
cv=5, scoring='f1_weighted')
grid_search.fit(X_train, y_train)

# Get the results and the corresponding F1 scores
results = grid_search.cv_results_
f1_scores = results['mean_test_score']

# Plot the F1 scores for each combination of hyperparameters
fig, ax = plt.subplots()
ax.plot(range(len(f1_scores)), f1_scores)
ax.set_xticks(range(len(f1_scores)))
```

```
ax.set_xticklabels([str(params) for params in results['params']])
ax.set_xticklabels([str(params) for params in results['params']],
rotation=90)
ax.set_xlabel('Hyperparameters')
ax.set_ylabel('Test F1 score')
plt.show()

# Get the best F1 score and its corresponding hyperparameters
best_f1_score = grid_search.best_score_
best_params = grid_search.best_params_

# Print the best hyperparameters and their F1 score
print(f"Best hyperparameters: {best_params}")
print(f"Best TEST F1 score: {best_f1_score}")


# We elected to procced with KK based on above results, even thouhg we have
unabalnced data - though we will address this with removing more data
points later*

# In[48]:


# BUILD OUR MODEL AND EVALUATE
# Filter out UndefinedMetricWarning warnings
warnings.filterwarnings("ignore", category=UndefinedMetricWarning)
warnings.filterwarnings("ignore", category=UserWarning)
random.seed(123)


# Train a K-Nearest Neighbors model
model = KNeighborsClassifier(metric='cosine', algorithm='auto', n_neighbors
= 1)
#model = KNeighborsClassifier(metric='cosine', algorithm='auto')
model.fit(X_train, y_train)


# Predict on the test set
y_pred = model.predict(X_test)

# Predict on the train set
y_pred_train = model.predict(X_train)

# Calculate the metrics
f1_test = f1_score(y_test, y_pred, average='micro')
f1_train = f1_score(y_train, y_pred_train, average='micro')
precision_test = precision_score(y_test, y_pred, average='micro')
precision_train = precision_score(y_train, y_pred_train, average='micro')
recall_test = recall_score(y_test, y_pred, average='micro')
recall_train = recall_score(y_train, y_pred_train, average='micro')
accuracy_test = accuracy_score(y_test, y_pred)
accuracy_train = accuracy_score(y_train, y_pred_train)

# Print the performance metrics
print("Performance metrics on classification of text books")
print(f"Test F1 score: {f1_test:.4f} - Train F1 score: {f1_train:.4f}")
print(f"Test Precision: {precision_test:.4f} - Train Precision:
{precision_train:.4f}")
print(f"Test Recall: {recall_test:.4f} - Train Recall: {recall_train}")
print(f"Test Accuracy: {accuracy_test:.4f} - Train Accuracy:
{accuracy_train:.4f}")
```

```
print()



# Update the recommendation function to return the top n recommendations
and their similarity scores (KNN)
def recommend_books(model, vectorized_text, n=2):
    distances, indices = model.kneighbors(vectorized_text, n_neighbors=n)
    similarity_scores = 1 - distances
    return indices, similarity_scores

# Make recommendations for the test set
n_recommendations = 2
test_indices, similarity_scores = recommend_books(model, X_test,
n_recommendations)



# Section 2: Calculate F1 score for each test book and store the
recommended books in a DataFrame
# create empty lists for storing the data
test_isbns = []
test_users = []
recommended_isbns = []
recommended_users = []
f1_scores = []
similarity_scores_list = []

# Loop through each row of the test set and append recommended books to the
list
for i in range(len(test_data)):
    test_isbn = test_data.iloc[i]['isbn']
    test_user = test_data.iloc[i]['User_type']
    recommended_isbns_row = []
    recommended_users_row = []
    similarity_scores_row = []

    for idx, sim_score in zip(test_indices[i], similarity_scores[i]):
        rec_isbn = train_data.iloc[idx]['isbn']
        rec_user = train_data.iloc[idx]['User_type']
        #rec_user = ["ENGLISH_12"] # This line makes it a Naive Model, we
assign everyone an ENGLISH_12 BOOK
        recommended_isbns_row.append(rec_isbn)
        recommended_users_row.append(rec_user)
        similarity_scores_row.append(sim_score)

    # Binarize the test_user and recommended_users_row
    binarized_test_user = mlb.transform([test_user])[0]
    binarized_recommended_users = mlb.transform(recommended_users_row)

    # Calculate the F1 score for this row
    f1 = f1_score([binarized_test_user]*len(binarized_recommended_users),
binarized_recommended_users, average='weighted')

    # Append the data to the corresponding lists
    test_isbns.append(test_isbn)
    test_users.append(test_user)
    recommended_isbns.append(recommended_isbns_row)
    recommended_users.append(recommended_users_row)
    f1_scores.append(f1)
    similarity_scores_list.append(similarity_scores_row)
```

```python
# Create a dictionary to represent the recommended books for each test book
recommended_books_dict = {
    'Test_ISBN': test_isbns,
    'Test_User': test_users,
    'Recommended_ISBNs': recommended_isbns,
    'Recommended_Users': recommended_users,
    'Similarity_Scores': similarity_scores_list,
    'F1_Score': f1_scores
}

# Convert the dictionary to a pandas dataframe
recommended_books_df = pd.DataFrame(recommended_books_dict)

# Get the mean value of the 'f1_score' column
mean_f1_score = recommended_books_df['F1_Score'].mean()

# Print the dataframe
print("Performance metric on recomnedation system using test data only")
print('Mean F1 Score of Books:', mean_f1_score)
recommended_books_df


# In[ ]:





# # NAIVE MODEL
# How well is out model? Lets assume all users get recomnded the same text
book, or the same USERY_TYPE book. In our example the most common user type
is a year 12 English Book. We simply assume the recomneder model simply
assings everyone a Year 12 Enlgish book

# In[49]:


# NAIVE MODEL
# Section 2: Calculate F1 score for each test book and store the
recommended books in a DataFrame
# create empty lists for storing the data
test_isbns = []
test_users = []
recommended_isbns = []
recommended_users = []
f1_scores = []
similarity_scores_list = []

# Loop through each row of the test set and append recommended books to the
list
for i in range(len(test_data)):
    test_isbn = test_data.iloc[i]['isbn']
    test_user = test_data.iloc[i]['User_type']
    recommended_isbns_row = []
    recommended_users_row = []
    similarity_scores_row = []

    for idx, sim_score in zip(test_indices[i], similarity_scores[i]):
        rec_user = ["ENGLISH_12"] # This line makes it a Naive Model, we
assign everyone an ENGLISH_12 BOOK
```

```python
        recommended_isbns_row.append(rec_isbn)
        recommended_users_row.append(rec_user)

    # Binarize the test_user and recommended_users_row
    binarized_test_user = mlb.transform([test_user])[0]
    binarized_recommended_users = mlb.transform(recommended_users_row)

    # Calculate the F1 score for this row
    f1 = f1_score([binarized_test_user]*len(binarized_recommended_users),
binarized_recommended_users, average='weighted')

    # Append the data to the corresponding lists
    test_isbns.append(test_isbn)
    test_users.append(test_user)
    recommended_isbns.append(recommended_isbns_row)
    recommended_users.append(recommended_users_row)
    f1_scores.append(f1)
    similarity_scores_list.append(similarity_scores_row)

# Create a dictionary to represent the recommended books for each test book
recommended_books_dict = {
    'Test_ISBN': test_isbns,
    'Test_User': test_users,
    'Recommended_ISBNs': '9780571336173, 9780099462217',
    'Recommended_Users': recommended_users,
    'F1_Score': f1_scores
}

# Convert the dictionary to a pandas dataframe
recommended_books_df = pd.DataFrame(recommended_books_dict)

# Get the mean value of the 'f1_score' column
mean_f1_score = recommended_books_df['F1_Score'].mean()

# Print the dataframe
print('Mean F1 Score of NAIVE Books:', mean_f1_score)
recommended_books_df


# # Can we improve our model?
# - Simplify model and base user on subject level only
# - Limit scope of user base, such as 15 users min required - how does this
impact the model
#
# We found that if we remove users with less then 15 observations in the
data set that we improve both our model and naive model. Though, this
limits the scope of analysis. This can be done just by toggling the start
of the model and changing the value for filteigng

# In[ ]:
```