Batu Arda Düzgün
21802633

**Abstract:**

In this work Zipf's Law and the relationship between the token size in the corpus and the vocabulary size is analyzed over books from different author and books from different genres. Three books from three different authors and three different books from three different genres, in total 18 books are selected for the analyses. The books are preprocessed than tokenized and analyzed to see the Zip's Law and the relationship between the token size in the corpus. From these analyses a classification method for the books were devised.

**Introduction:**

The main topics of this assignment are Zipfs law and the relationship between the token size in the corpus and the vocabulary size is analyzed. Zipfs law is about how there are few words used very frequently and a lot of words used rarely [1]. It is a law applicable to many different distributions as it is part of human nature to take the path of least effort. Because of this it is a well-researched research area with many articles on the topic [2]. This law is quite important for all application of NLP is because of NLP becomes very difficult. As no matter how much data we have some data will always be spare due to the nature of this law making these words hard to learn in a direct method.

In this work we analyze multiple different corpora to find how they are effected but the Zipf's law. Additionally the same book used in the previous analysis are used in vocabulary growth analysis. These analysis results are used to cluster the books based on their writer then books based on their genres. Then finally a randomly generated corpus is generated to test if Zipfs law holds for it as well. How and why Zipfs law holds for randomly generated text as well is explored in the article of Wentian Li [3]

**Corpus Construction and Implementation:**

I used 18 books in total. Half of these books were chosen based on their other and the other half were chosen based on the books genres. Nine corpora's are chosen based on their authors by picking three books from three authors each. Then nine books are chosen based on their respective genres by picking three books from three different genres each. In total these make up the 18 corpora used in this report.

The corpora chosen for the author are listed below:

- Charles Dickens
    - A Child's History of England
    - Bleak House
    - Great Expectations
- Fyodor Dostoyevsky
    - Crime and Punishment
    - The Brothers Karamazov
    - The Possessed (The Devils)
- Graf Leo Tolstoy
    - Anna Karenina
    - Resurrection
    - War and Peace

The corpora chosen for the different genres are listed below:

- Fantasy
  - Four Arthurian Romances by active 12th century de Troyes Chrétien
  - Le Morte d'Arthur Volume 1 by Sir Thomas Malory
  - The Night Land by William Hope Hodgson
- Horror
  - Dracula by Bram Stoker
  - The Mysteries of Udolpho by Ann Ward Radcliffe
  - Varney the Vampire; Or, the Feast of Blood by Prest and Rymer
- Romance
  - Don Quixote by Miguel de Cervantes Saavedra
  - Notre-Dame de Paris by Victor Hugo
  - The Man in the Iron Mask by Alexandre Dumas

The books for authors are chosen such that their UTF-8 files are larger then 1MBs. Also I tried to pick books which are well known. This severely limited the authors which could be chosen as there are few writer which has writer 3 book that are that long. Again the books with specific genres are chosen chosen such that their UTF-8 files are larger then 1MBs. And again I tried to pick books which are similar in nature or books that are famous.

I used Python to pre-process the books. For the pre-process I removed the all of the punctuation marks and cast the texts to lower. I removed apostrophe by placing a space in their place, so as an example “isn’t” becomes “isn t”. Then saved the books for easier use later. Then I removed the re-read the preprocessed corpuses and removed the stop-words from then and separately saved the stop-word-removed version of the corpuses. In total I had 36 corpuses 18 of them are pre-processed and 18 of them pre-processed and stop-word-removed. For the stop-word’s list I used the stop-word list for the NLTK. The list is given below [4]:

```
"ourselves", "hers", "between", "yourself", "but", "again", "there", "about",
"once", "during", "out", "very", "having", "with", "they", "own", "an", "be",
"some", "for", "do", "its", "yours", "such", "into", "of", "most", "itself",
"other", "off", "is", "s", "am", "or", "who", "as", "from", "him", "each",
"the", "themselves", "until", "below", "are", "we", "these", "your", "his",
"through", "don", "nor", "me", "were", "her", "more", "himself", "this",
"down", "should", "our", "their", "while", "above", "both", "up", "to",
"ours", "had", "she", "all", "no", "when", "at", "any", "before", "them",
"same", "and", "been", "have", "in", "will", "on", "does", "yourselves",
"then", "that", "because", "over", "why", "so", "can", "did", "not", "now",
"under", "he", "you", "herself", "has", "just", "where", "too", "only",
"myself", "which", "those", "i", "after", "few", "whom", "t", "being", "if",
"theirs", "my", "against", "a", "by", "doing", "it", "how", "further", "was",
"here", "than"
```

As an example three level of pre-process for the book are given below.

The plain text:

A Chancery judge once had the kindness to inform me, as one of a company of some hundred and fifty men and women not labouring under any suspicions of lunacy, that the Court of Chancery, though the shining subject of much popular prejudice (at which point I thought the judge's eye had a cast in my direction), was almost immaculate.

Only pre-processed:

a chancery judge once had the kindness to inform me as one of a company of some hundred and fifty men and women not labouring under any suspicions of lunacy that the court of chancery though the shining subject of much popular prejudice at which point i thought the judges eye had a cast in my direction was almost immaculate

Stop-word-removed:

chancery judge kindness inform one company hundred fifty men women labouring suspicions lunacy court chancery though shining subject much popular prejudice point thought judges eye cast direction almost immaculate

**Results:**

The parts (a), (b), (c), (d) and (e) do not have outputs as they are only about downloading the corpus and preprocessing the downloaded corpus. I will talked about these parts in the previous section already and I will re-mention them again as they come up. The rest of the parts, each have their own sub-heading with their respective plots and comments

- **Part f:**

First of all I need to mention that I saved the result of part e which were dictionaries containing word types along with their frequencies for different books in a dictionary object. In this part I created a function which can merge two dictionary objects that contain word types along with their frequencies. I used this function to merge the three corpuses of each author to create their respective large corpuses. Then I plotted the requested plots.

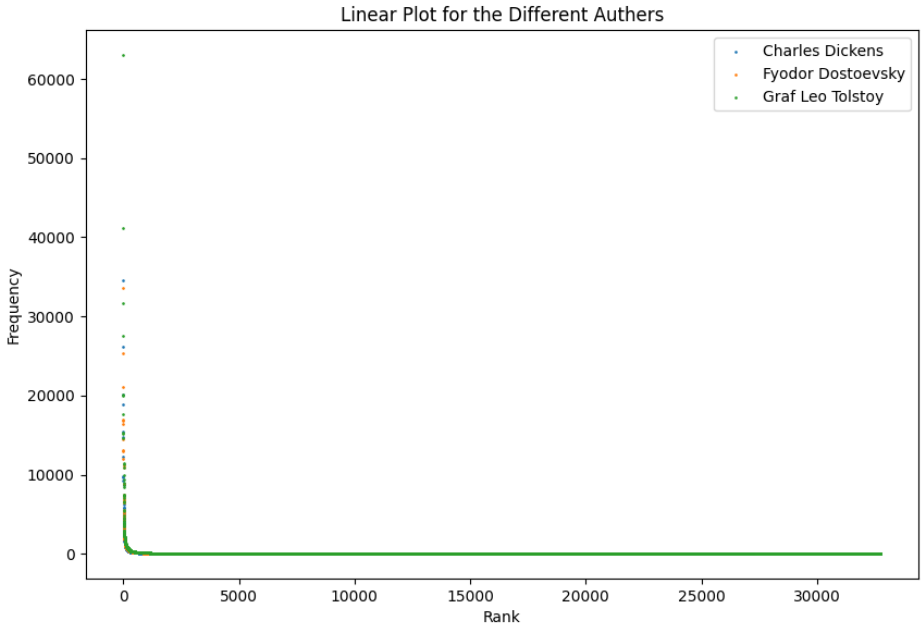


*Figure 1: This figure is the plot of word frequency rank vs. frequency plot for the large corpuses of the three authors (Charles Dickens, Fyodor Dostoevsky, Graf Leo Tolstoy) for the stop word including version of the their corpuses. This plot is primarily draw to show the Zipf's law. George Zipf found out that in all corpora there are few words used very frequently and a lot of words used rarely. In particular, he noted, the n-th most frequent word has a frequency proportional to the negative power of its rank (n). Let $r_n$ be the rank of the word n-th word and $f_n$ be the frequency of the n-th word. Then the Zip's law is $r_n \approx C \cdot f_n^{-b}$. This plots are coherent with that result.*

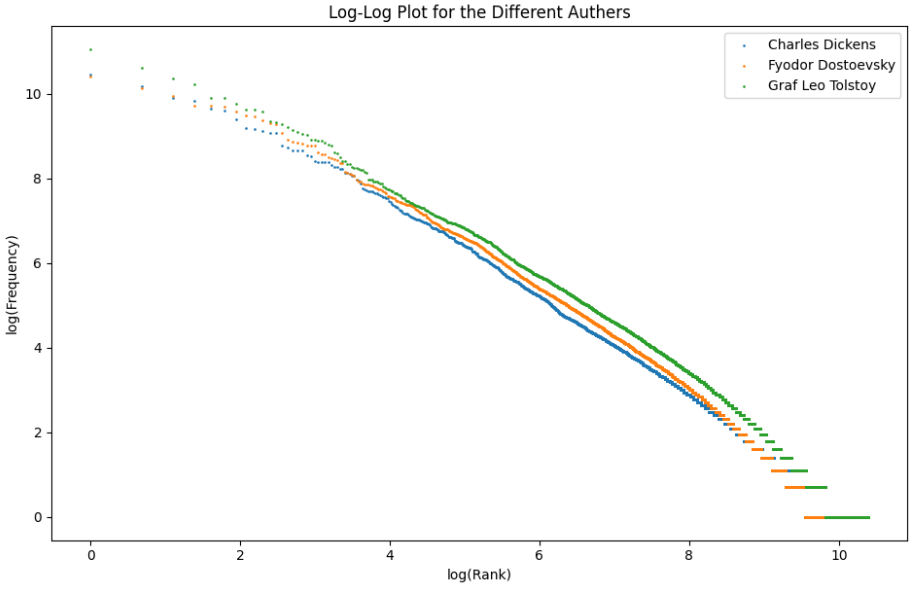


*Figure 2: This figure is the plot of log (word frequency rank) vs. log(frequency) plot for the large corpuses of the three authors (Charles Dickens, Fyodor Dostoevsky, Graf Leo Tolstoy) for the stop word including version of the their corpuses. As previously seen in Figure 1 the results are coherent with the Zipf's law. However this plot is even better at showing the negative power relationship as the log-log plots have negative linear shape as expected from a log of a negative power. Also interestingly the graphs slops are approximately -1. Even though there are no explanations for it, this is something which has been noted before about naturel corpuses.*

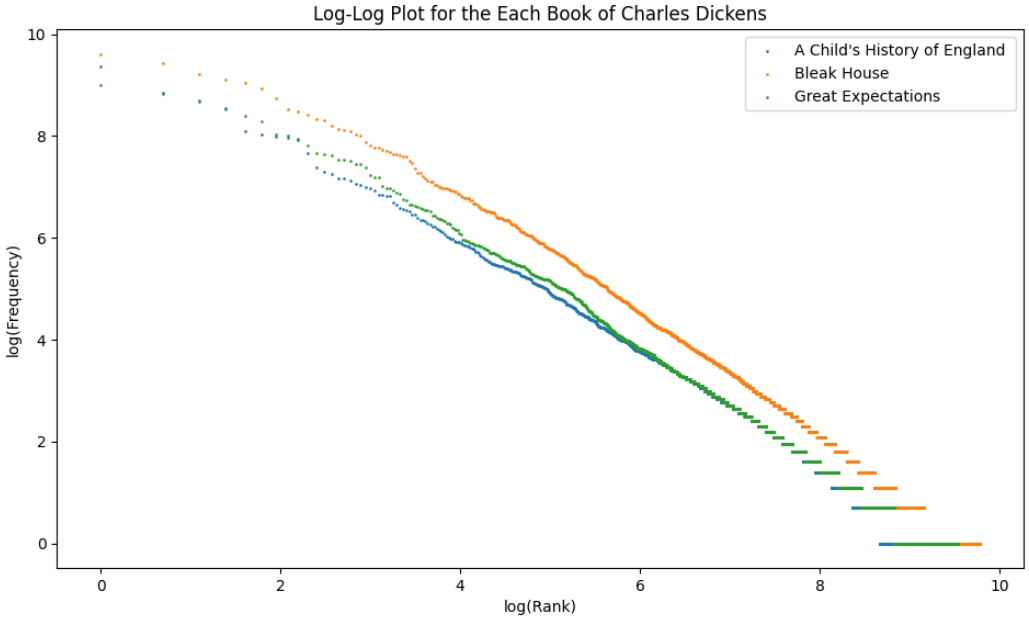


*Figure 3: This figure is the plot of log (word frequency rank) vs. log(frequency) plot for the three books of the Charles Dickens, for the stop word including version of the his corpuses. This plot shows that not only the large corpus of the authors fallows the Zipfs's law but also Charles Dickens's books individually are in accordance with Zipf's law. This is because all the curves have a negative slope showing the negative power relation between the rank and frequency.*

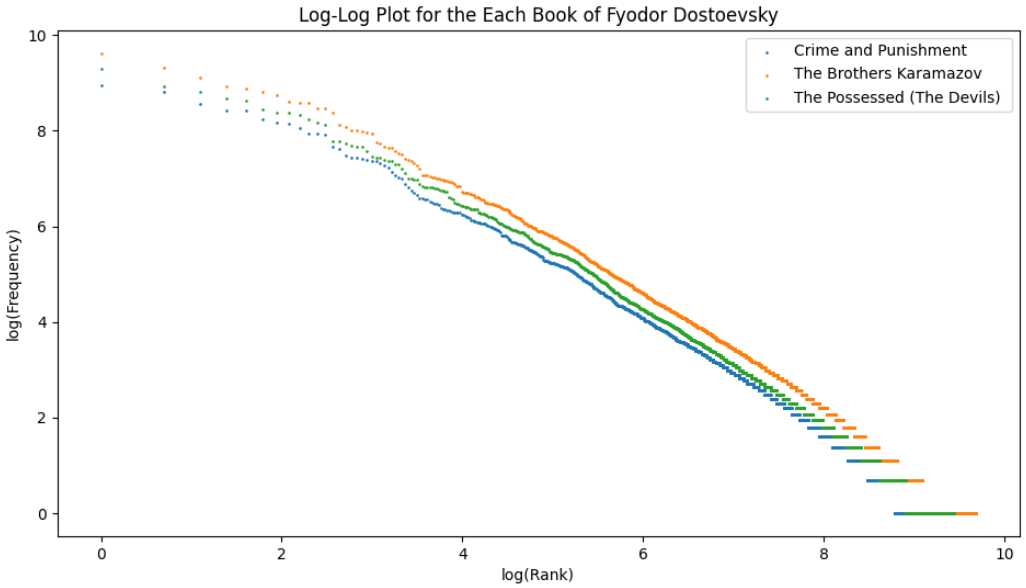


*Figure 4: This figure is the plot of log (word frequency rank) vs. log(frequency) plot for the three books of the Fyodor Dostoevsky, for the stop word including version of the his corpuses. This plot shows that not only the large corpus of the authors fallows the Zipf's law but also Fyodor Dostoevsky's books individually are in accordance with Zipf's law. This is because all the curves have a negative slope showing the negative power relation between the rank and frequency.*

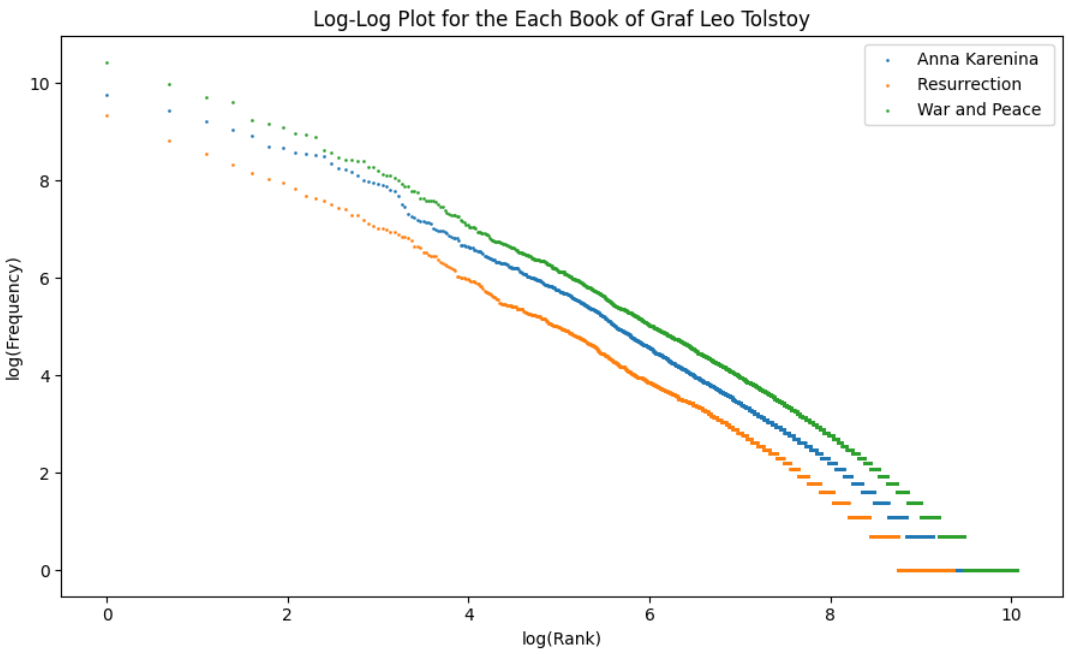


*Figure 5: This figure is the plot of log (word frequency rank) vs. log(frequency) plot for the three books of the Graf Leo Tolstoy, for the stop word including version of the his corpuses. This plot shows that not only the large corpus of the authors fallows the Zipf's law but also Graf Leo Tolstoy's books individually are in accordance with Zipf's law. This is because all the curves have a negative slope showing the negative power relation between the rank and frequency.*

The plots seem to show the books chosen according to their author's all are coherent with the Zipf's law. Also the same authors books seem to have similar Zipf's law curves. However, using corpus analysis for book classification will be further discussed in part k.

- **Part g:**

In this part the relation between the corpus size and the token size are examined. For this tasked I counted the number of unique words once every 5000, 10000, 15000, 20000, … words until the book is over. To do this I first tokenized the corpus and took increasingly larger sets of if to analyze. Then I turned the samples to sets and counted the number of elements. I repeated this process for each author's large corpus.

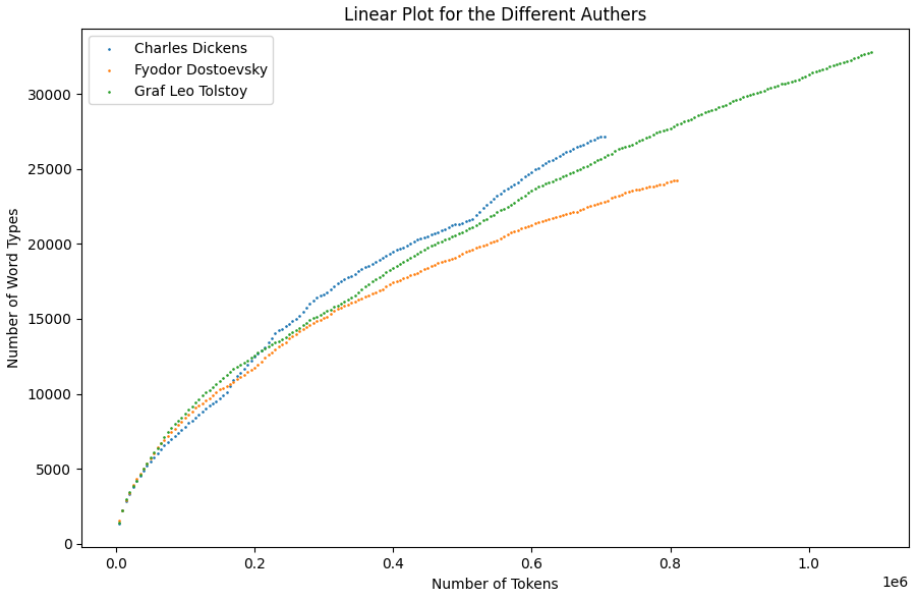


*Figure 6: This figure is the plot of number of token vs. number of unique words (corpus size) for the large corpuses of the three authors (Charles Dickens, Fyodor Dostoevsky, Graf Leo Tolstoy) for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. It is easy to see there is a $C_1 x^{C_2}$ relation between the number of tokens and the corpus size. This relation is quite logical as in the beging every word is a new word so the corpus size grows rapidly but as the number of tokens increase the most words become repeating words and the growth of the corpse size decreases. There are clear jumps in the curve for Charles Dickens. These anomalies can be attributed to the point where the different corpora's are joined. Most likely each new book has a lot of new names in it making the corpus size jump up.*

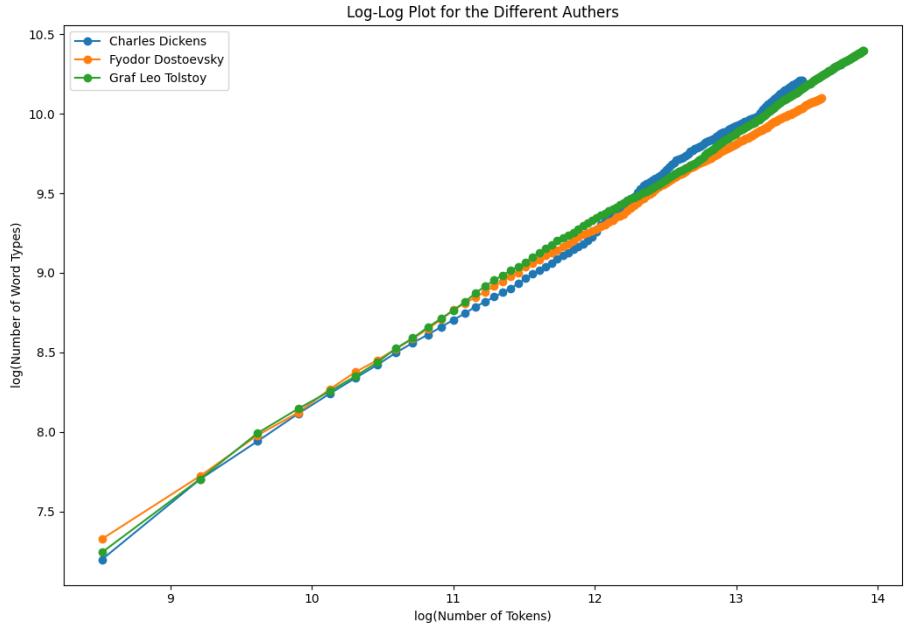


*Figure 7: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for the large corpuses of the three authors (Charles Dickens, Fyodor Dostoevsky, Graf Leo Tolstoy) for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. As expected from Figure 6 these plots are linear lines. It is even easier to see there is a $C_1 x^{C_2}$ relation between the number of tokens and the corpus size ass the log-log plot is a line. This relation is quite logical as in the initially every word is a new word so the corpus size grows rapidly but as the number of tokens increase the most words become repeating words and the growth of the corpse size decreases.*

The relationship between the number of token and the corpus size is a exponential one ($C_1 x^{C_2}$). This relation ship can be represented with $C_1 x^{C_2}$ where $C_1$ and $C_2$ are constants. $C_1$ effects the bias and $C_2$ effects the slop of the plots in Figure 7. This relationship will be further examined in part h. As we will further discus in that part this information could even be used to differentiate different books.

- **Part h:**

In this part the relation between the corpus size and the token size are examined for each book of each author and the sloops of the curves are found and shown on the plots (the slope finding is part of part i but they are also shown on this parts curves). For this task I once again counted the number of unique words once every 5000, 10000, 15000, 20000, … words until the books are over. To do this I first tokenized the corpus and took increasingly larger sets of if to analyze. Then I turned the samples to sets and counted the number of elements. I repeated this process for each book of each author.

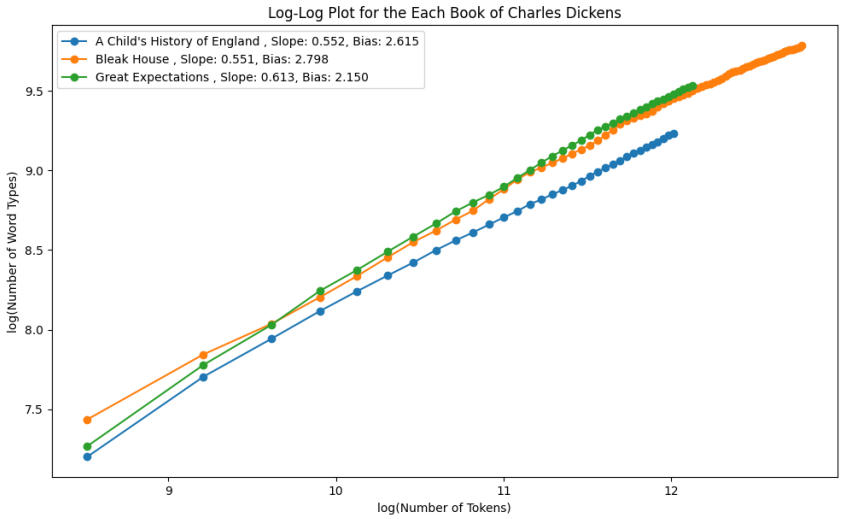


*Figure 8: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book of Charles Dickens for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expected from Figure 7 not only the large corpus show a $C_1 x^{C_2}$ relation but each of the books on their own show the same relationship. Additionally there is no big dislocation on the plots. This time, each plot is one book and because there are no merging there are no sudden influx of names in them. Also it is important to note that the curves are quite close to each other and as such there book can be clustered together by this. This will be further discussed in part k.*

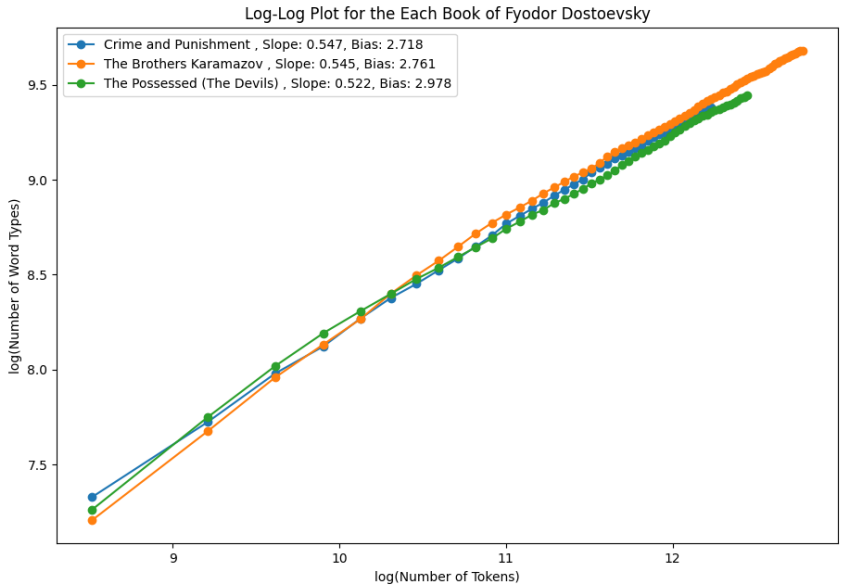


*Figure 9: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book of Fyodor Dostoevsky for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expected from Figure 7 not only the large corpus show a $C_1 x^{C_2}$ relation but each of the books on their own show the same relationship. Additionally there is no big dislocation on the plots. This time, each plot is one book and because there are no merging there are no sudden influx of names in them. Also it is important to note that the curves are quite close to each other and as such there book can be clustered together by this. This will be further discussed in part k.*

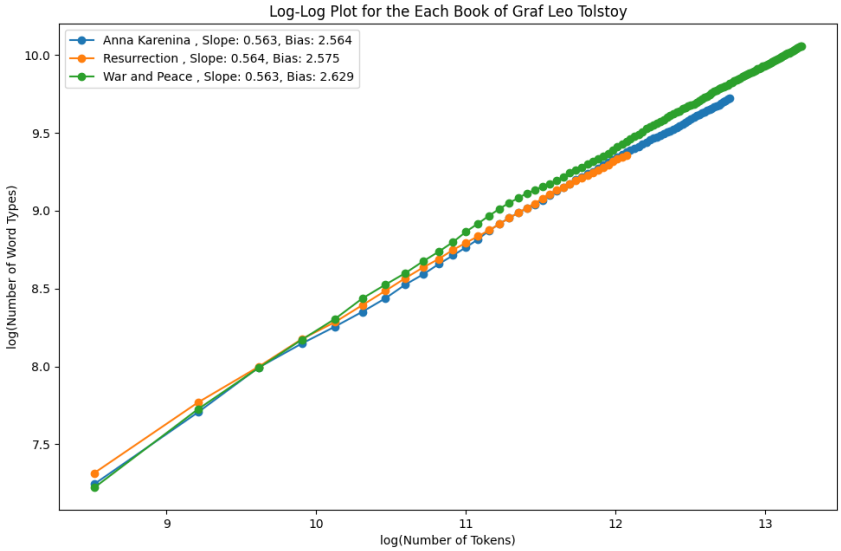


*Figure 10: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book of Graf Leo Tolstoy for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expected from Figure 7 not only the large corpus show a $C_1 x^{C_2}$ relation but each of the books on their own show the same relationship. Additionally there is no big dislocation on the plots. This time, each plot is one book and because there are no merging there are no sudden influx of names in them. Also it is important to note that the curves are quite close to each other and as such there book can be clustered together by this. This will be further discussed in part k.*

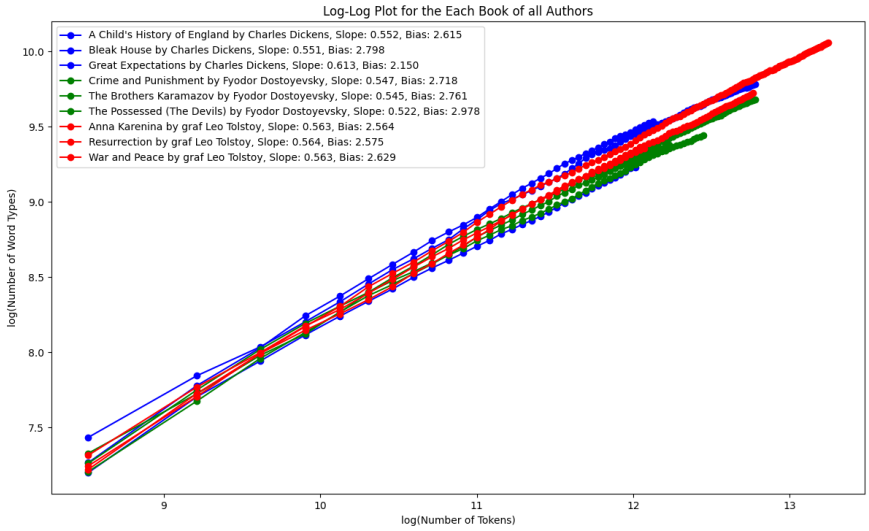


*Figure 11: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book of all the authors for the stop word including version of the their corpuses. Once again the slope and the bias of each curve is given in its legend. As expected from Figure 7 not only the large corpus show a $C_1 x^{C_2}$ relation but each of the books on their own show the same relationship. Additionally there is no big dislocation on the plots. Also it is important to note that the curves are quite close to each other and as such there book can be clustered together by this and this is easier to see in this plot. This will be further discussed in part k*

- **Part i:**

In this part the best fitting lines for the log(number of token) vs. log(corpus size) for each book of all the authors are found and displayed in a table. I used the line fitting function of Pythons internal library in this part.

| Author | Books | Slope | Bias |
|---|---|---|---|
| Charles Dickens | A Child's History of England | 0.552 | 2.615 |
| | Bleak House | 0.551 | 2.798 |
| | Great Expectations | 0.613 | 2.150 |
| Fyodor Dostoyevsky | Crime and Punishment | 0.547 | 2.718 |
| | The Brothers Karamazov | 0.545 | 2.761 |
| | The Possessed (The Devils) | 0.522 | 2.978 |
| Graf Leo Tolstoy | Anna Karenina | 0.563 | 2.564 |
| | Resurrection | 0.564 | 2.575 |
| | War and Peace | 0.563 | 2.629 |

*Table 1: This is the table of slope and bias for the log(number of token) vs. log(corpus size) curves for each book of all the authors for the stop word including version of the their corpuses. Also it is important to note that the curves are quite close to each a specially the books by Graf Leo Tolstoy. This will be further discussed in part k*

- **Part j:**

This part is same as part h and part i but it is for corpuses from different genres instead of authors. So, in this part the relation between the corpus size and the token size are examined for each book from all the genres and the sloops of the curves are found and shown on the plots. For this task I once again counted the number of unique words once every 5000, 10000, 15000, 20000, … words until the books are over.

To do this I first tokenized the corpus and took increasingly larger sets of if to analyze. Then I turned the samples to sets and counted the number of elements. I repeated this process for each book.

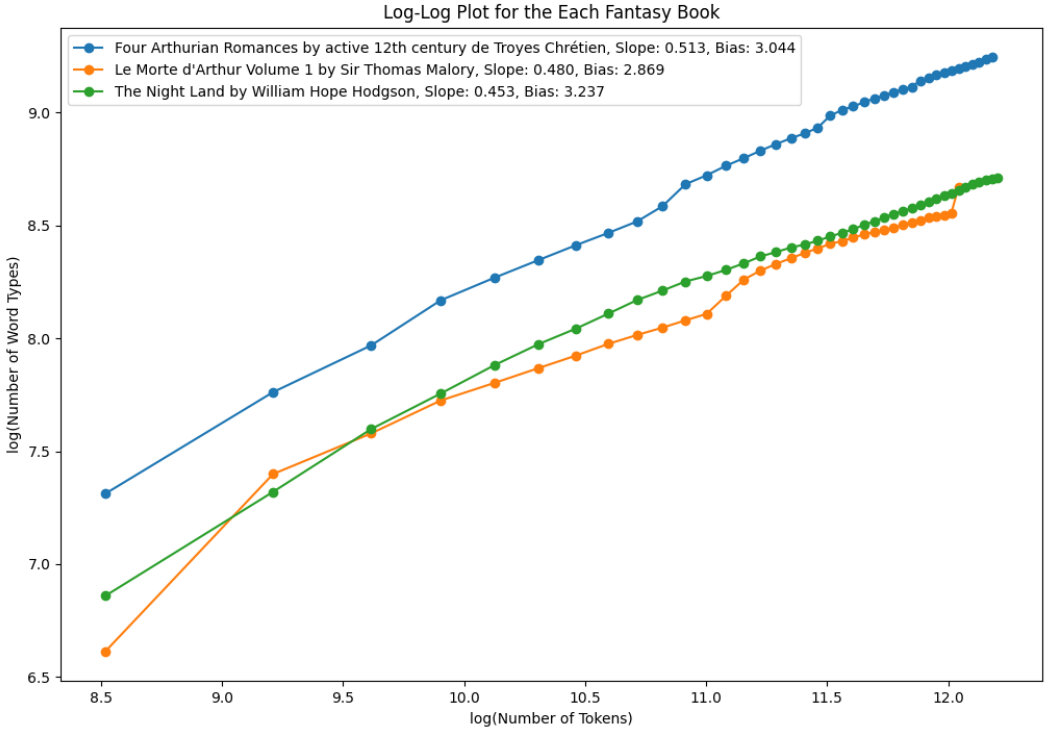


*Figure 12: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book in the fantasy genre for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expect from part h these books also have the $C_1 x^{C_2}$. Clustering of these curves as well will discussed in part k.*

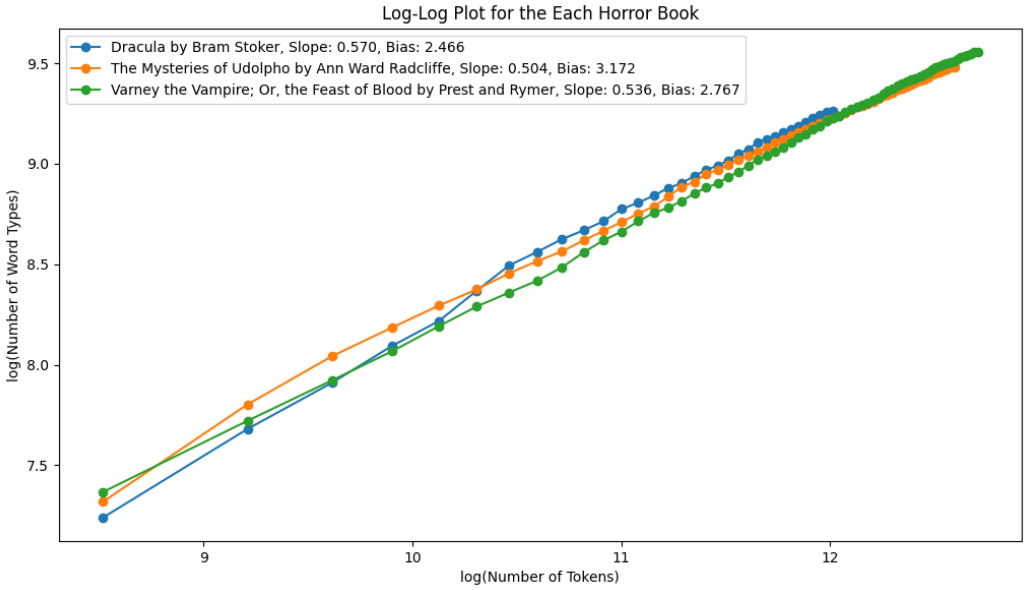


*Figure 13: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book in the horror genre for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between*

*the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expect from part h these books also have the $C_1 x^{C_2}$ relation. Clustering of these curves as well will discussed in part k.*

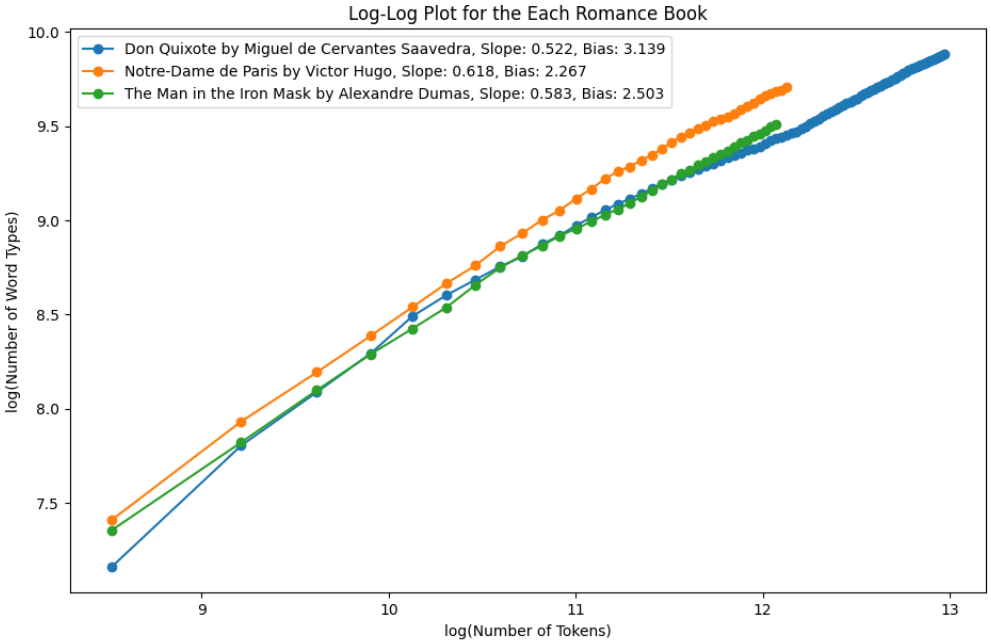


*Figure 14: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book in the romance genre for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expect from part h these books also have the $C_1 x^{C_2}$ relation. Clustering of these curves as well will discussed in part k.*

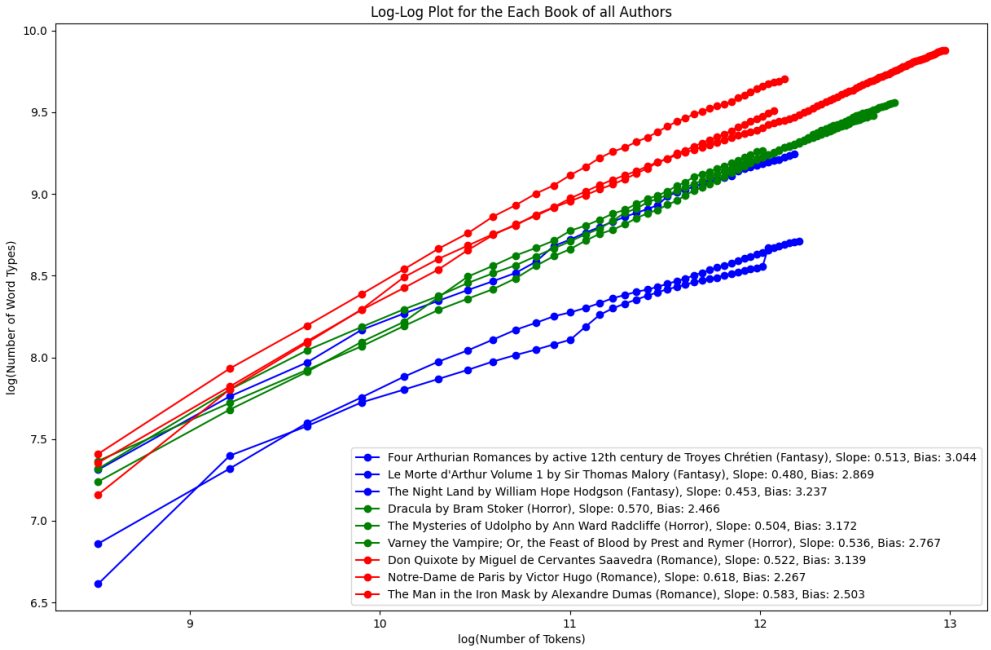


*Figure 15: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book in all the genres for the stop word including version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The slope and the bias of each curve is given in its legend. As expect from part h these books also have the $C_1 x^{C_2}$ relation. Clustering of these curves as well will discussed in part k however it should be noted that other than an exception in the fantasy genre these books seem very easy to distinguish from each other.*

| Genre | Books | Slope | Bias |
|---|---|---|---|
| Fantasy | Four Arthurian Romances | 0.513 | 3.044 |
| | Le Morte d'Arthur Volume 1 | 0.480 | 2.869 |
| | The Night Land | 0.453 | 3.237 |
| Horror | Dracula | 0.570 | 2.466 |
| | The Mysteries of Udolpho | 0.504 | 3.172 |
| | Varney the Vampire | 0.536 | 2.767 |
| Romance | Don Quixote | 0.522 | 3.139 |
| | Notre-Dame de Paris | 0.618 | 2.267 |
| | The Man in the Iron Mask | 0.583 | 2.503 |

*Table 1: This is the table of slope and bias for the log(number of token) vs. log(corpus size) curves for each book from all the genres for the stop word including version of the their corpuses. Also it is important to note that the curves from the same genre are quite close to each This will be further discussed in part k*

As expected books chosen based on their genres have very similar behaviors to the books chosen for their authors.

- **Part k:**

We could deduce from the results of part (g), (h), (i), (j) that a clustering methodologies could be deduced for books so that the members of each group belong to the same author or literary type. From Figure 11 we would see the plots almost cluster on the graph according to their authors with the only exception being A Child's History of England by Charles Dickens looking like it is a book by Fyodor Dostoyevsky. Also Table 1 reviles that Graf Leo Tolstoy has books almost the same slope (only a difference on 0.001 difference) and very close bias. These results show authors writing style definitely effect their plots slop and bias and this is truer for some writers. If writers with distinctive writing styles are chosen than a clustering methodologies which you automatically group books according to their authors is defiantly possible. However such a system wouldn't work for all authors as some writers change their writing styles drastically between books.

From Figure 15 we could see that books from different genres are very clearly separate themselves from each other. The only expectation to this behavior is the Four Arthurian Romances by active 12th century de Troyes Chrétien looks like a romance book. However, as the books name suggest this book might be more heavily leaning towards the romance genre then the Fantasy genre. So this should not even be seen as a mistake of the algorithm but rather its success. However, it is still important remember I deliberately tried to pick book genres quite different from each other so a different group of genres which are fundamentally closer to each other may not give as successful results. Still it is clear different genres impose different writing styles so strongly to their book that a simple test such as this one could be detected them.

It is very important to note the success of this clustering methodology can only be seen for low number of clusters. As there are only two degrees of freedom (bias and slope) if there are five author as suggested by the guideline than it is almost inevitable for different author books to heavily overlap.

- **Part I:**

In this part we discuss the effects of removing the stop-words in parts (g), (h), (i), (j), (k). I repeated all the previous parts for this part with the stop-word-removed corpora. I only placed some of the plots to compare with the previous plots.

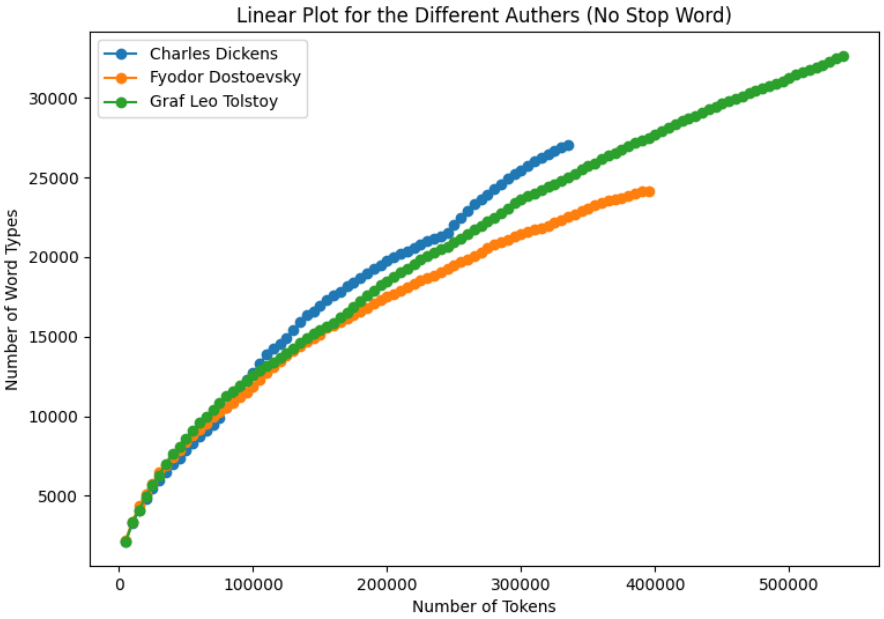


*Figure 16: This figure is the plot of number of token vs. number of unique words (corpus size) for the large corpuses of the three authors (Charles Dickens, Fyodor Dostoevsky, Graf Leo Tolstoy) for the stop word removed version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size, and how it differs from the stop word included version. It is easy to see there is very little difference. Stop words are the most commons words so removing them slightly increases the slope of the graphs yet it doesn't affect the relationship between the plots in a significant way.*

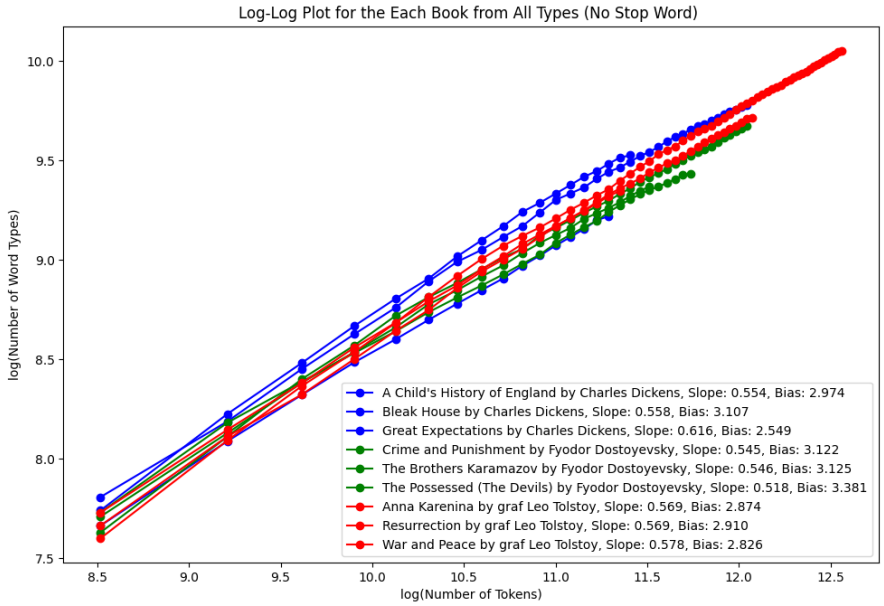


*Figure 17: This figure is the plot of log(number of token) vs. log(number of unique words (corpus size)) for each book of all the authors for the stop word removed version of the their corpuses. Once again the slope and the bias of each curve is given in its*

*legend. This plot is draw to examine how it differs from the stop word included version. It is easy to see there is very little difference. Stop words are the most commons words so removing them slightly increases the slope of the graphs yet it doesn't affect the relationship between the plots in a significant way. This is easier to see in this plot as the slops of the lines are written in the legend.*

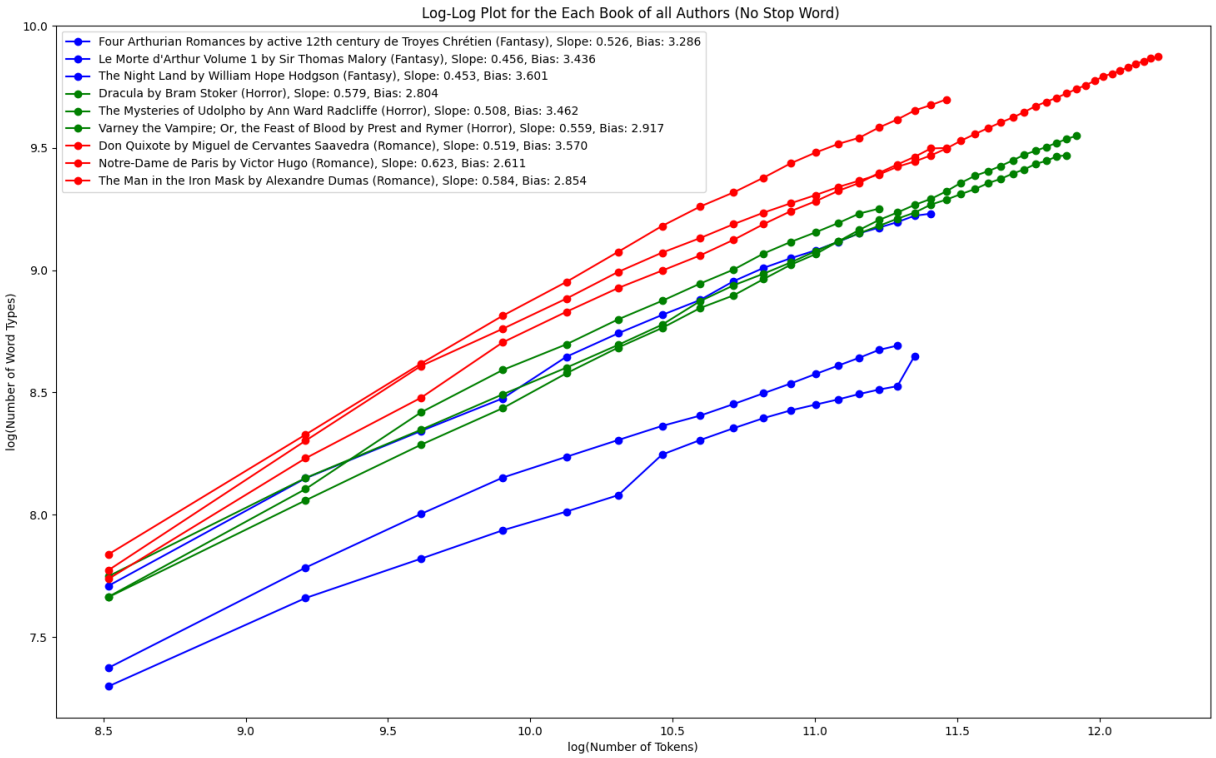


*Figure 18: This figure is the plot of log(number of token) vs. log(corpus size) for each book in all the genres for the stop word removed version of the their corpuses. This plot is primarily draw to analyze the relationship between the number of token and corpus size and how it differs form its stop word included counter part. The slope and the bias of each curve is given in its legend. Stop words are the most commons words so removing them slightly increases the slope of the graphs yet it doesn't affect the relationship between the plots in a significant way. This is easier to see in this plot as the slops of the lines are written in the legend.*

In the end, it seems removing the stop words has little effect on the in parts (g), (h), (i), (j), (k). The plots almost look identical to their stop-word included counterparts. The only significant difference is because stop words are the most commons, removing them slightly increases the slope of the graphs. As an example the three books of Graf Leo Tolstoy slopes change from Anna Karenina: 0.563, Resurrection: 0.564 and War and Peace: 0.563 to Anna Karenina: 0.569, Resurrection: 0.569, War and Peace: 0.578. So the slopes are very close to the previous slope values only slightly larger for the stop-word-removed corpora just as we expected. Normally stop-word removal is quite important and impactful for NLP tasks, so how insignificant it is for these is quite interesting.

- **Part m:**

In this part a randomly generated corpus with a size approximately equivalent to the combined size of two novels is crated and analyzed. A randomly generated corpus is defined in the paper of Wentian Li [5]. It is defined as a sequence of randomly drawn sequence of characters from a poll of $(M + 1)$

characters with replacement. There are no colaration between the draws so each character could be seen as an i.i.d. random variable. The character poll of $(M + 1)$ character has one "blank space" in it and any sequence of characters between two consecutive "blank space" are words. And a string of "blank space" is not a word. Under this definition I crated a random corpus using python which is around 2.5 MB which is around double the size of the average novels size. I used $M = 26$ as there are 26 letters in English. Then I repeated the steps on part f and part g to draw the plots to see if it exhibits Zipfian behavior or the behavior seen in part g.

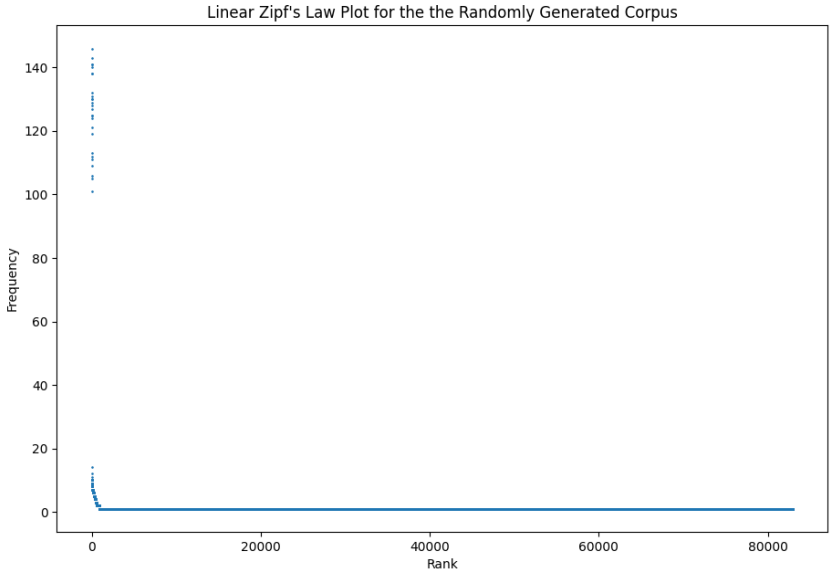


*Figure 19: This figure is the plot of word frequency rank vs. frequency plot for the large corpuses of the randomly generated corpus. This plot is primarily draw to show the Zipf's law. It seems to show Zipfian behavior. However, It should be noted that for M = 26 a corpus of size 2.5 MB is too small to analyze properly.*

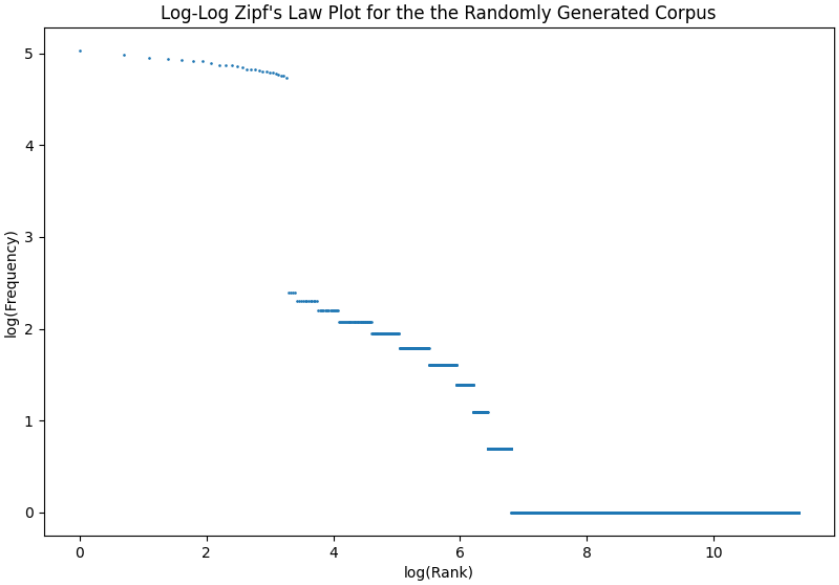


*Figure 20: This figure is the plot of log(word frequency rank) vs. log(frequency) plot for the large corpuses of the randomly generated corpus. This plot is primarily draw to show the Zipf's law. It seems to show Zipfian behavior however it is a lot more segmented. This is probably because for a corpus with M = 26 a size of 2.5 MB is too small to analyze properly.*

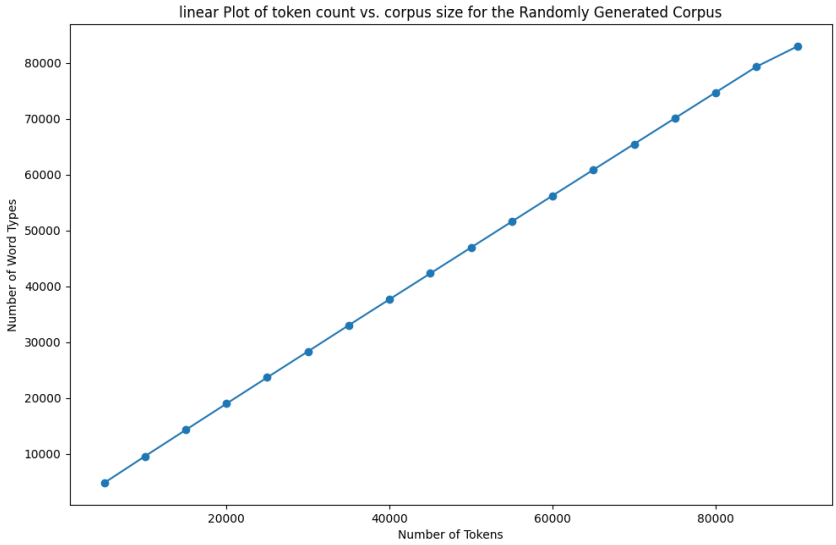


*Figure 21: This figure is the plot of number of token vs. number of unique words (corpus size) for randomly generated corpus. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The relation is almost completely linear and it doesn't resemble the $C_1 x^{C_2}$ plot from part g. This is probably because for a corpus with M = 26 a size of 2.5 MB is too small to analyze properly.*

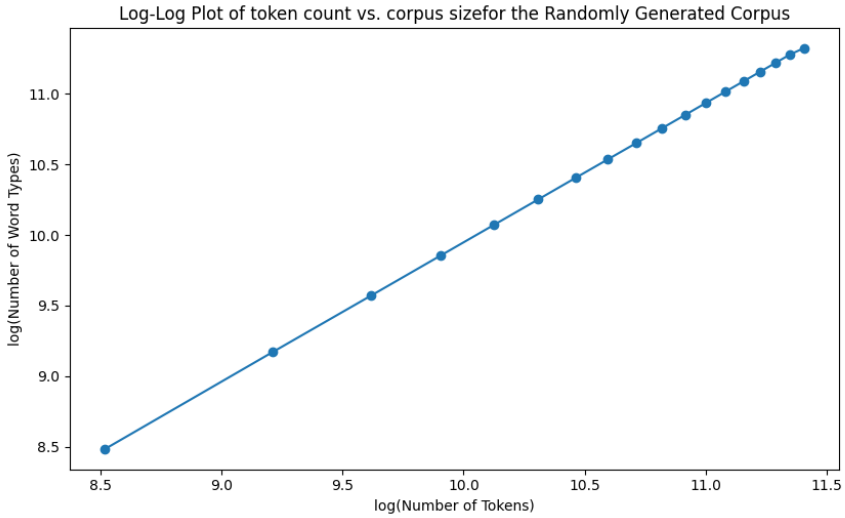


*Figure 22: This figure is the plot of log(number of token) vs. log(corpus size) for randomly generated corpus. This plot is primarily draw to analyze the relationship between the number of token and corpus size. The relation is linear again but it is not because of the same reason as the plots from part g. This is probably because for a corpus with M = 26 a size of 2.5 MB is too small to analyze properly.*

As I mentioned multiple times 2.5 MB is too small to analyze a randomly generated corpus with $M = 26$. So I placed some plots for $M = 3$ to show it.

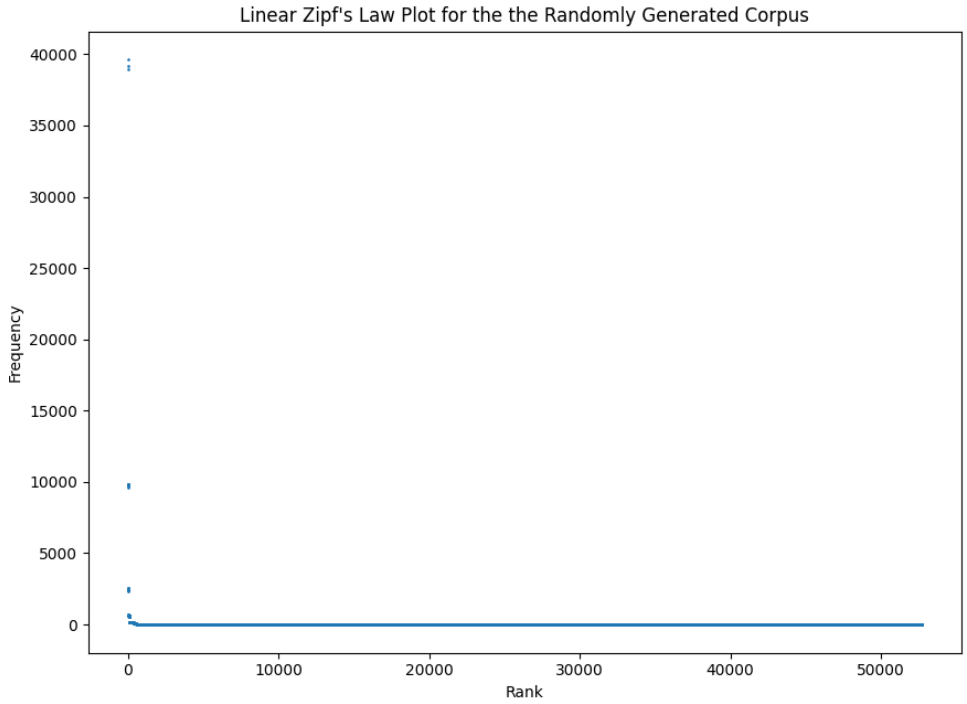


*Figure 23: This figure is the plot of word frequency rank vs. frequency plot for the large corpuses of the randomly generated corpus for only three characters. This plot is primarily draw to show the Zipf's law. It seems to show Zipfian behavior.*

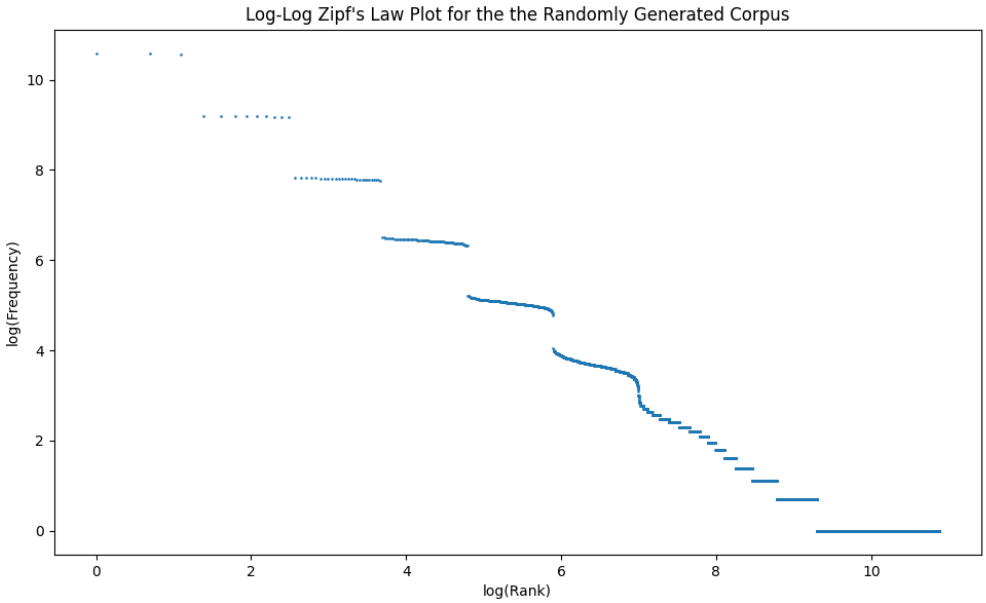


*Figure 24: This figure is the plot of log(word frequency rank) vs. log(frequency) plot for the large corpuses of the randomly generated corpus for only three characters. This plot is primarily draw to show the Zipf's law. It seems to show Zipfian behavior however it is a little bit more segmented.*

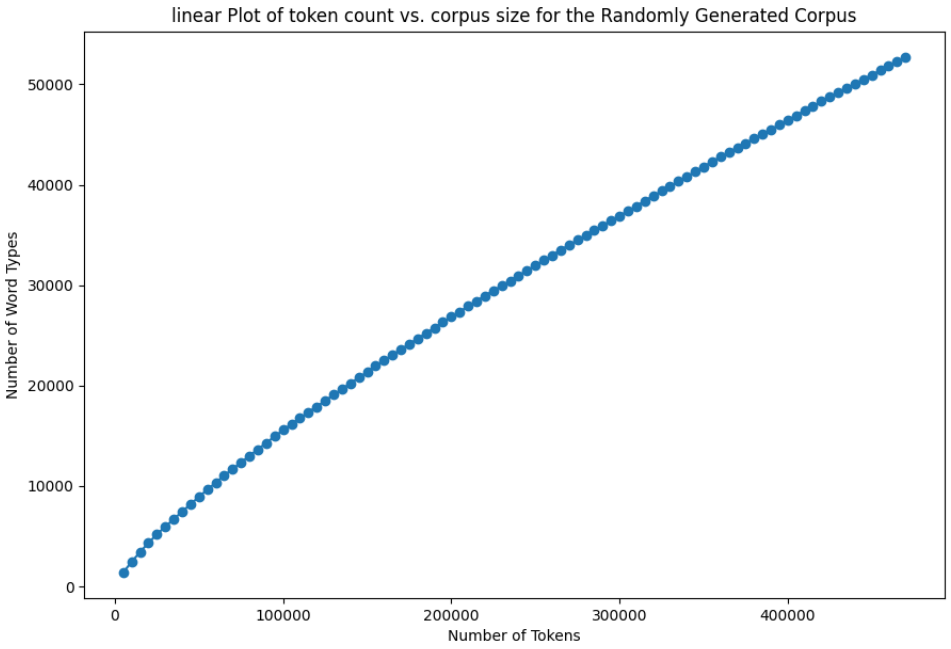


*Figure 25: This figure is the plot of number of token vs. number of unique words (corpus size) for randomly generated corpus for only three characters. This plot is primarily draw to analyze the relationship between the number of token and corpus size. This is like the $C_1 x^{C_2}$ plot from part g.*

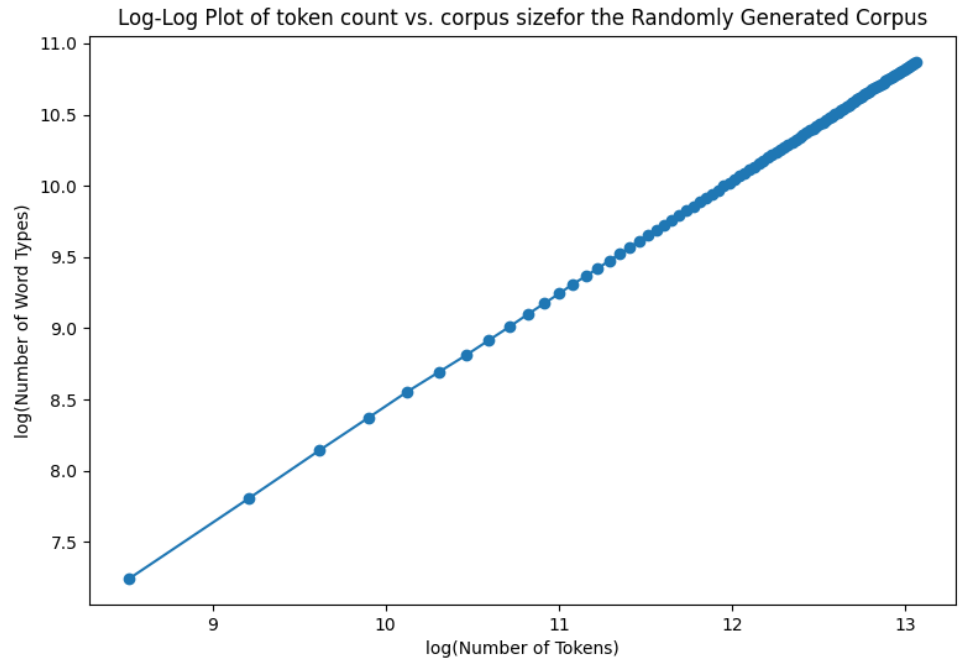


*Figure 26: This figure is the plot of log(number of token) vs. log(corpus size) for randomly generated corpus for only three characters. This plot is primarily draw to analyze the relationship between the number of token and corpus size. As expected from figure 25 it is linear like the plots from part g.*

The randomly generated corpus acts very much like a book written with normal language in this tests if the randomly generated text is generated using a low number of different characters. I believe the same

effect could be captured by using 26 characters as well however a bigger corpus is needed. I can't test this a too large corpuses take a lot of time to generate and they can't fit in my computers RAM.

**Discussions & Conclusions:**

The main objective of this assignment was to analyze the Zipf's law and the relationship between the corpus size and the token size. In my experiments I saw that books from famous writers obeyed the Zipf's law even when their books are joined. I saw that corpus size and the token size relation of a corpus is something meaningful and distinctive. Different writers can have different and sometimes very consistent that corpus size and the token size relation in their books. I also saw corpus size and the token size could be used to distinguish books from different genres. However, these corpus classifications have a limit as each relation has two degrees of freedom if too many class are tried to be distinguished some are bound to overlap. I saw how little effect stop-word removal had on the analysis of the previous parts. Finally we tested the same experiments on a randomly generated corpora. The results showed that under these analysis there is little difference between a randomly generated corpora and a novel, as the randomly generated corpora showed Zipfian behavior and the same corpus size and the token size relation as the novels. The two most interesting outcomes of the assignment for me were: The random generated corpus shows Zipfian behavior. This shows that the Zipf's law is not as mysteries and deep as it might seem at first glance but rather it is something rooted in probability. And how corpus size and the token size relation could be a distinguishing feature of a corpus.

**References:**

[1] G. K. Zipf, Human behavior and the principle of least effort: An ntroduction to human ecology, Martino Fine Books, 1949.

[2] X. Gabaix, "Zipf's Law for Cities: An Explanation," *The Quarterly Journel of Economics,* vol. 114, no. 3, pp. 739-767, 2002.

[3] W. Li, "Random texts exhibit Zipf's-law-like word frequency distribution," *IEEE Transactions on Information Theory,* vol. 38, no. 6, pp. 1842-1845, 1992.

[4] "removing stop words nltk python," Geeks for Geeks, 31 May 2021. [Online]. Available: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/. [Accessed 18 March 2022].

[5] W. Li, "Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution," *IEEE Transactions on Information Theory,,* vol. 38, no. 6, pp. 1842-1845, 1992.

**Appendix:**

```
import os
import string
import random as rd
import numpy as np
import matplotlib.pyplot as plt

books_path = "books"
PP_books_path = "PP_books"
NSW_PP_books_path = "NSW_PP_books"

authors = "Authors"
genres = "Types"


random_corpus_file_name = "random_corpus.txt"


book_classes = [authors, genres]

writer_names = ["Charles Dickens", "Fyodor Dostoevsky", "Graf Leo Tolstoy"]
genra_names = ["Fantasy", "Horror", "Romance"]

vocabulary_file = {PP_books_path: {authors: {writer_names[0]: [],
writer_names[1]: [], writer_names[2]: []},
                                   genres: {genra_names[0]: [],
genra_names[1]: [], genra_names[2]: []}},
                   NSW_PP_books_path: {authors: {writer_names[0]: [],
writer_names[1]: [], writer_names[2]: []},
                                       genres: {genra_names[0]: [],
genra_names[1]: [], genra_names[2]: []}}}

corpus_size_file = {PP_books_path: {authors: {writer_names[0]: [],
writer names[1]: [], writer names[2]: []},
                                    genres: {genra_names[0]: [],
genra_names[1]: [], genra_names[2]: []}},
                    NSW_PP_books_path: {authors: {writer_names[0]: [],
writer_names[1]: [], writer_names[2]: []},
                                        genres: {genra_names[0]: [],
genra_names[1]: [], genra_names[2]: []}}}

corpus_size_slope_file = {PP_books_path: {authors: {writer_names[0]: [],
writer_names[1]: [], writer_names[2]: []},
                                          genres: {genra_names[0]: [],
genra_names[1]: [], genra_names[2]: []}},
                          NSW_PP_books_path: {authors: {writer_names[0]: [],
writer_names[1]: [], writer_names[2]: []},
                                              genres: {genra_names[0]: [],
genra_names[1]: [], genra_names[2]: []}}}

stop_words = ["ourselves", "hers", "between", "yourself", "but", "again",
"there", "about", "once", "during", "out",
              "very", "having", "with", "they", "own", "an", "be", "some",
"for", "do", "its", "yours", "such",
              "into", "of", "most", "itself", "other", "off", "is", "s",
"am", "or", "who", "as", "from", "him",
```

```
                "each", "the", "themselves", "until", "below", "are", "we",
"these", "your", "his", "through", "don",
                "nor", "me", "were", "her", "more", "himself", "this", "down",
"should", "our", "their", "while",
                "above", "both", "up", "to", "ours", "had", "she", "all",
"no", "when", "at", "any", "before", "them",
                "same", "and", "been", "have", "in", "will", "on", "does",
"yourselves", "then", "that", "because",
                "over", "why", "so", "can", "did", "not", "now", "under",
"he", "you", "herself", "has", "just",
                "where", "too", "only", "myself", "which", "those", "i",
"after", "few", "whom", "t", "being", "if",
                "theirs", "my", "against", "a", "by", "doing", "it", "how",
"further", "was", "here", "than"]


def Preproces(input_text, remove_stopwords):

    # removing the punctuation and apostrophe
    temp_text = input_text.translate(str.maketrans("’", " ", "\’" +
string.punctuation))

    # making it all lower case
    temp_text = temp_text.lower()

    if remove_stopwords:

        text_words = temp_text.split()
        NSW_text_words = [word for word in text_words if word.lower() not in
stop_words]
        temp_text = ' '.join(NSW_text_words)

    return temp_text

def Pre_process_books():

    for book_class in book_classes:

        for class_name in os.listdir(f"{books_path}\{book_class}"):

            temp_book_names =
os.listdir(f"{books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:

                f =
open(f"{books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8', errors='replace')

                temp_text = ""

                for line in f.readlines():

                    temp_text += line

                f.close()
```

```
                original_text = temp_text

                # preprocessing
                temp_text = Preproces(original_text, False)

                # saving the file
                f =
open(f"{PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'w',
encoding='utf-8')
                f.write(temp_text)

                f.close()


                # preprocessing
                temp_text = Preproces(original_text, True)

                # saving the file
                f =
open(f"{NSW_PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'w',
encoding='utf-8')
                f.write(temp_text)

                f.close()

def CountFrequency(my_list):
    # Creating an empty dictionary
    freq = {}
    for item in my_list:
        if (item in freq):
            freq[item] += 1
        else:
            freq[item] = 1

    return freq

def MergeDictionary(dic1, dic2):

    for item in dic2:

        if (item in dic1):

            dic1[item] += dic2[item]

        else:

            dic1[item] = dic2[item]

    return dic1

def Vocabulary_counter(word_list, step_size):

    keep_counting = True
    step_count = 1

    counts_list = []
```

```
    while (step_count * step_size < len(word_list)):

        temp_set = set(word_list[:step_count * step_size])
        counts_list.append(len(temp_set))

        step_count += 1

    temp_set = set(word_list)
    counts_list.append(len(temp_set))

    return counts_list



def Part_e():
    # for only pre processed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{PP_books_path}\{book_class}"):

            temp_book_names =
os.listdir(f"{PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:
                f =
open(f"{PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                temp_text = ""

                for line in f.readlines():
                    temp_text += line

                f.close()

                temp_text_words = temp_text.split()

                temp_dictinory = CountFrequency(temp_text_words)

                temp_sorted_dictionary = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}


vocabulary_file[PP_books_path][book_class][class_name].append(temp_sorted_dict
ionary)

    """
    # for pre processed and stop word removed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{NSW_PP_books_path}\{book_class}"):

            temp_book_names =
os.listdir(f"{NSW_PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:
                f =
```

```
open(f"{NSW_PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                temp_text = ""

                for line in f.readlines():
                    temp_text += line

                f.close()

                temp_text_words = temp_text.split()

                temp_dictinory = CountFrequency(temp_text_words)

                temp_sorted_dictionary = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}


vocabulary_file[NSW_PP_books_path][book_class][class_name].append(temp_sorted_
dictionary)
    """

def Part_f():

    # creating the merge of books of each author
    preprocess_type = "PP_books"
    book_class = "Authors"
    for class_name in vocabulary_file[preprocess_type][book_class]:

        temp_sumed_dictionary = {}

        for dictionary in
vocabulary_file[preprocess_type][book_class][class_name]:

            temp_sumed_dictionary = MergeDictionary(temp_sumed_dictionary,
dictionary)


        temp_sorted_dictionary = {k: v for k, v in
sorted(temp_sumed_dictionary.items(), key=lambda item: item[1], reverse=True)}


vocabulary_file[preprocess_type][book_class][class_name].append(temp_sorted_di
ctionary)


    # ploting the linear plot for authers
    legend_names = []
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        dictinory =
vocabulary_file[preprocess_type][book_class][class_name][3]

        freqs = list(dictinory.values())
        ranks = list(range(1, len(dictinory.keys()) + 1))
```

```
        plt.scatter(ranks, freqs, s=0.8, label=class_name)


    plt.xlabel("Rank")
    plt.ylabel("Frequency")
    plt.title("Linear Plot for the Different Authers")
    plt.legend(legend_names)
    plt.show()


    # ploting the log-log plot for authers
    legend_names = []
    for class name in vocabulary file[preprocess type][book class]:

        legend_names.append(class_name)

        dictinory =
vocabulary_file[preprocess_type][book_class][class_name][3]

        freqs = list(dictinory.values())
        ranks = list(range(1, len(dictinory.keys()) + 1))

        plt.scatter(np.log(ranks), np.log(freqs), s=0.8, label=class_name)


    plt.xlabel("log(Rank)")
    plt.ylabel("log(Frequency)")
    plt.title("Log-Log Plot for the Different Authers")
    plt.legend(legend_names)
    plt.show()


    # ploting the log-log plots for each authers books
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names = [name.split("by")[0] for name in legend_names]

        for i in range(3):
            dictinory =
vocabulary_file[preprocess_type][book_class][class_name][i]

            freqs = list(dictinory.values())
            ranks = list(range(1, len(dictinory.keys()) + 1))

            plt.scatter(np.log(ranks), np.log(freqs), s=0.8, label=class_name)

        plt.xlabel("log(Rank)")
        plt.ylabel("log(Frequency)")
        plt.title(f"Log-Log Plot for the Each Book of {class_name}")
        plt.legend(legend_names)
        plt.show()

def Part_g():
```

```
    # for only pre processed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{PP_books_path}\{book_class}"):

            temp_text = ""

            temp_book_names =
os.listdir(f"{PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:
                f =
open(f"{PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                for line in f.readlines():
                    temp_text += line

                f.close()

            temp_text_words = temp_text.split()

            temp_word_counts = Vocabulary_counter(temp_text_words, 5000)


corpus_size_file[PP_books_path][book_class][class_name].append(temp_word_count
s)




    #ploting the plots
    preprocess_type = "PP_books"
    book_class = "Authors"

    # ploting the linear plot for authers
    legend_names = []
    for class_name in corpus_size_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][0]
        number_of_tokens = np.array(range(1, len(number_of_words_types) + 1))
* 5000

        plt.scatter(number_of_tokens, number_of_words_types , s=0.8,
label=class_name)


    plt.xlabel("Number of Tokens")
    plt.ylabel("Number of Word Types")
    plt.title("Linear Plot for the Different Authers")
    plt.legend(legend_names)
    plt.show()

    # ploting the log-log plot for authers
```

```
    legend_names = []
    for class_name in corpus_size_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][0]
        number_of_tokens = np.array(range(1, len(number_of_words_types) + 1))
* 5000

        plt.plot(np.log(number_of_tokens), np.log(number_of_words_types), "-
o", label=class_name)


    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title("Log-Log Plot for the Different Authers")
    plt.legend(legend_names)
    plt.show()

def Part_h_i():

    # for only pre processed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{PP_books_path}\{book_class}"):

            temp_book_names =
os.listdir(f"{PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:

                temp_text = ""

                f =
open(f"{PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                for line in f.readlines():
                    temp_text += line

                f.close()

                temp_text_words = temp_text.split()

                temp_word_counts = Vocabulary_counter(temp_text_words, 5000)


corpus_size_file[PP_books_path][book_class][class_name].append(temp_word_count
s)

                #calculating the slope:
                temp_number_of_tokens = np.array(range(1,
len(temp_word_counts) + 1)) * 5000

                log_temp_word_counts = np.log(temp_word_counts)
                log_temp_number_of_tokens = np.log(temp_number_of_tokens)
```

```
                temp_slop, temp_bias = np.polyfit(log_temp_number_of_tokens,
log_temp_word_counts, 1)


corpus_size_slope_file[PP_books_path][book_class][class_name].append((temp_slo
p, temp_bias))




    #ploting the plots
    preprocess_type = "PP_books"
    book class = "Authors"


    # ploting the log-log plots for each authers books
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names = [name.split("by")[0] for name in legend_names]

        for i in [-3, -2, -1]:

            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]

            legend_names[i] += f", Slope: {slope:.3f}, Bias: {bias:.3f}"


        plt.xlabel("log(Number of Tokens)")
        plt.ylabel("log(Number of Word Types)")
        plt.title(f"Log-Log Plot for the Each Book of {class_name}")
        plt.legend(legend_names)
        plt.show()


    # ploting the log-log plots for all authers books
    colors = ["b", "g", "r"]
    color_index = 0

    legend_names = []
    curve_number = 0

    for class_name in vocabulary_file[preprocess_type][book_class]:

        temp_legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
```

```
        legend_names += [name.split(".txt")[0] for name in temp_legend_names]

        for i in [-3, -2, -1]:

            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", color=colors[color_index], label=class_name)

            slope, bias =
corpus size slope file[preprocess type][book class][class name][i]

            legend_names[curve_number] += f", Slope: {slope:.3f}, Bias:
{bias:.3f}"

            curve_number += 1

        color_index += 1

    print("*** slope bias table for the only preprocesed authors ***")
    for i in legend_names:
        print(i)
    print("\n\n")

    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title(f"Log-Log Plot for the Each Book of all Authors")
    plt.legend(legend_names)
    plt.show()

def Part_j():

    #only filling the files if they are empty so we can run this part alone.
    if len(corpus_size_slope_file[PP_books_path][authors][writer_names[0]]) ==
0:

        # for only pre processed books
        for book_class in book_classes:

            for class_name in os.listdir(f"{PP_books_path}\{book_class}"):

                temp_book_names =
os.listdir(f"{PP_books_path}\{book_class}\{class_name}")

                for temp_book_name in temp_book_names:

                    temp_text = ""

                    f =
open(f"{PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                    for line in f.readlines():
                        temp_text += line
```

```
                    f.close()

                    temp_text_words = temp_text.split()

                    temp_word_counts = Vocabulary_counter(temp_text_words,
5000)


corpus_size_file[PP_books_path][book_class][class_name].append(temp_word_count
s)

                    # calculating the slope:
                    temp number of tokens = np.array(range(1,
len(temp_word_counts) + 1)) * 5000

                    log_temp_word_counts = np.log(temp_word_counts)
                    log_temp_number_of_tokens = np.log(temp_number_of_tokens)

                    temp_slop, temp_bias =
np.polyfit(log_temp_number_of_tokens, log_temp_word_counts, 1)


corpus_size_slope_file[PP_books_path][book_class][class_name].append((temp_slo
p, temp_bias))

        """
        # for pre processed and stop word removed books
        for book_class in book_classes:

            for class_name in os.listdir(f"{NSW_PP_books_path}\{book_class}"):

                temp_book_names =
os.listdir(f"{NSW_PP_books_path}\{book_class}\{class_name}")

                for temp_book_name in temp_book_names:

                    temp_text = ""

                    f =
open(f"{NSW_PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                    for line in f.readlines():
                        temp_text += line

                    f.close()

                    temp_text_words = temp_text.split()

                    temp_word_counts = Vocabulary_counter(temp_text_words,
5000)


corpus_size_file[NSW_PP_books_path][book_class][class_name].append(temp_word_c
ounts)
```

```
                    # calculating the slope:
                    temp_number_of_tokens = np.array(range(1,
len(temp_word_counts) + 1)) * 5000

                    log_temp_word_counts = np.log(temp_word_counts)
                    log_temp_number_of_tokens = np.log(temp_number_of_tokens)

                    temp_slop, temp_bias =
np.polyfit(log_temp_number_of_tokens, log_temp_word_counts, 1)


corpus_size_slope_file[NSW_PP_books_path][book_class][class_name].append((temp
_slop, temp_bias))
        """

    # ploting the plots
    preprocess_type = "PP_books"
    book_class = "Types"

    # ploting the log-log plots for each type books
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names = [name.split(".txt")[0] for name in legend_names]

        for i in [-3, -2, -1]:
            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]

            legend_names[i] += f", Slope: {slope:.3f}, Bias: {bias:.3f}"

        plt.xlabel("log(Number of Tokens)")
        plt.ylabel("log(Number of Word Types)")
        plt.title(f"Log-Log Plot for the Each {class_name} Book")
        plt.legend(legend_names)
        plt.show()



    # ploting the log-log plots for each all types of books
    colors = ["b", "g", "r"]
    color_index = 0

    legend_names = []
    curve_number = 0

    for class_name in vocabulary_file[preprocess_type][book_class]:
```

```
        temp_legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names += [name.split(".txt")[0] for name in temp_legend_names]

        for i in [-3, -2, -1]:
            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", color=colors[color_index], label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]

            legend_names[curve_number] += f" ({class_name}), Slope:
{slope:.3f}, Bias: {bias:.3f}"

            curve_number += 1

        color_index += 1

    print("*** slope bias table for the only preprocesed authors ***")
    for i in legend_names:
        print(i)
    print("\n\n")

    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title(f"Log-Log Plot for the Each Book of all Authors")
    plt.legend(legend_names)
    plt.show()

def Part_1():

    # *************************
    # *** creating the files ***
    # *************************
    # part e
    # for pre processed and stop word removed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{NSW_PP_books_path}\{book_class}"):

            temp_book_names =
os.listdir(f"{NSW_PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:
                f =
open(f"{NSW_PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                temp_text = ""

                for line in f.readlines():
                    temp_text += line
```

```
                f.close()

                temp_text_words = temp_text.split()

                temp_dictinory = CountFrequency(temp_text_words)

                temp_sorted_dictionary = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}


vocabulary_file[NSW_PP_books_path][book_class][class_name].append(temp_sorted_
dictionary)

    # part g
    # for pre processed and stop word removed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{NSW_PP_books_path}\{book_class}"):

            temp_text = ""

            temp_book_names =
os.listdir(f"{NSW_PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:
                f =
open(f"{NSW_PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                for line in f.readlines():
                    temp_text += line

                f.close()

            temp_text_words = temp_text.split()

            temp_word_counts = Vocabulary_counter(temp_text_words, 5000)


corpus_size_file[NSW_PP_books_path][book_class][class_name].append(temp_word_c
ounts)

    # part h_i
    # for pre processed and stop word removed books
    for book_class in book_classes:

        for class_name in os.listdir(f"{NSW_PP_books_path}\{book_class}"):

            temp_book_names =
os.listdir(f"{NSW_PP_books_path}\{book_class}\{class_name}")

            for temp_book_name in temp_book_names:

                temp_text = ""

                f =
```

```
open(f"{NSW_PP_books_path}\{book_class}\{class_name}\{temp_book_name}", 'r',
encoding='utf-8')

                for line in f.readlines():
                    temp_text += line

                f.close()

                temp_text_words = temp_text.split()

                temp_word_counts = Vocabulary_counter(temp_text_words, 5000)


corpus_size_file[NSW_PP_books_path][book_class][class_name].append(temp_word_c
ounts)

                #calculating the slope:
                temp_number_of_tokens = np.array(range(1,
len(temp_word_counts) + 1)) * 5000

                log_temp_word_counts = np.log(temp_word_counts)
                log_temp_number_of_tokens = np.log(temp_number_of_tokens)

                temp_slop, temp_bias = np.polyfit(log_temp_number_of_tokens,
log_temp_word_counts, 1)


corpus_size_slope_file[NSW_PP_books_path][book_class][class_name].append((temp
_slop, temp_bias))



    # ************************
    # *** drawing the plots ***
    # ************************
    # part f
    # creating the merge of books of each author
    preprocess_type = "NSW_PP_books"
    book_class = "Authors"
    for class_name in vocabulary_file[preprocess_type][book_class]:

        temp_sumed_dictionary = {}

        for dictionary in
vocabulary_file[preprocess_type][book_class][class_name]:

            temp_sumed_dictionary = MergeDictionary(temp_sumed_dictionary,
dictionary)


        temp_sorted_dictionary = {k: v for k, v in
sorted(temp_sumed_dictionary.items(), key=lambda item: item[1], reverse=True)}


vocabulary_file[preprocess_type][book_class][class_name].append(temp_sorted_di
ctionary)
```

```
    # ploting the linear plot for authers
    legend_names = []
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        dictinory =
vocabulary_file[preprocess_type][book_class][class_name][3]

        freqs = list(dictinory.values())
        ranks = list(range(1, len(dictinory.keys()) + 1))

        plt.scatter(ranks, freqs, s=0.8, label=class name)


    plt.xlabel("Rank")
    plt.ylabel("Frequency")
    plt.title("Linear Plot for the Different Authers (No Stop Word)")
    plt.legend(legend_names)
    plt.show()


    # ploting the log-log plot for authers
    legend_names = []
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        dictinory =
vocabulary_file[preprocess_type][book_class][class_name][3]

        freqs = list(dictinory.values())
        ranks = list(range(1, len(dictinory.keys()) + 1))

        plt.scatter(np.log(ranks), np.log(freqs), s=0.8, label=class_name)


    plt.xlabel("log(Rank)")
    plt.ylabel("log(Frequency)")
    plt.title("Log-Log Plot for the Different Authers (No Stop Word)")
    plt.legend(legend_names)
    plt.show()


    # ploting the log-log plots for each authers books
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names = [name.split("by")[0] for name in legend_names]

        for i in range(3):
            dictinory =
vocabulary_file[preprocess_type][book_class][class_name][i]

            freqs = list(dictinory.values())
```

```
            ranks = list(range(1, len(dictinory.keys()) + 1))

            plt.scatter(np.log(ranks), np.log(freqs), s=0.8, label=class_name)

        plt.xlabel("log(Rank)")
        plt.ylabel("log(Frequency)")
        plt.title(f"Log-Log Plot for the Each Book of {class_name} (No Stop
Word)")
        plt.legend(legend_names)
        plt.show()



    # part g
    #ploting the plots
    preprocess_type = "NSW_PP_books"
    book_class = "Authors"

    # ploting the linear plot for authers
    legend_names = []
    for class_name in corpus_size_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][0]
        number_of_tokens = np.array(range(1, len(number_of_words_types) + 1))
* 5000

        plt.plot(number_of_tokens, number_of_words_types, "o-",
label=class_name)


    plt.xlabel("Number of Tokens")
    plt.ylabel("Number of Word Types")
    plt.title("Linear Plot for the Different Authers (No Stop Word)")
    plt.legend(legend_names)
    plt.show()

    # ploting the log-log plot for authers
    legend_names = []
    for class_name in corpus_size_file[preprocess_type][book_class]:

        legend_names.append(class_name)

        number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][0]
        number_of_tokens = np.array(range(1, len(number_of_words_types) + 1))
* 5000

        plt.plot(np.log(number_of_tokens), np.log(number_of_words_types), "o-
", label=class_name)


    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title("Log-Log Plot for the Different Authers (No Stop Word)")
```

```
    plt.legend(legend_names)
    plt.show()


    # part h_i
    #ploting the plots
    preprocess_type = "NSW_PP_books"
    book_class = "Authors"


    # ploting the log-log plots for each authers books
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names = [name.split("by")[0] for name in legend_names]

        for i in [-3, -2, -1]:

            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]

            legend_names[i] += f", Slope: {slope:.3f}, Bias: {bias:.3f}"

        plt.xlabel("log(Number of Tokens)")
        plt.ylabel("log(Number of Word Types)")
        plt.title(f"Log-Log Plot for the Each Book of {class_name} (No Stop
Word)")
        plt.legend(legend_names)
        plt.show()


    # ploting the log-log plots for all authers books
    colors = ["b", "g", "r"]
    color_index = 0

    legend_names = []
    curve_number = 0

    for class_name in vocabulary_file[preprocess_type][book_class]:

        temp_legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names += [name.split(".txt")[0] for name in temp_legend_names]

        for i in [-3, -2, -1]:

            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
```

```
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", color=colors[color_index], label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]

            legend_names[curve_number] += f", Slope: {slope:.3f}, Bias:
{bias:.3f}"

            curve_number += 1

        color_index += 1

    print("*** slope bias table for the preprocesed and (No Stop Word) authors
***")
    for i in legend_names:
        print(i)
    print("\n\n")

    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title(f"Log-Log Plot for the Each Book from All Types (No Stop Word)
")
    plt.legend(legend_names)
    plt.show()




    # part j
    # ploting the plots
    preprocess_type = "NSW_PP_books"
    book_class = "Types"

    # ploting the log-log plots for each type books
    for class_name in vocabulary_file[preprocess_type][book_class]:

        legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names = [name.split(".txt")[0] for name in legend_names]

        for i in [-3, -2, -1]:
            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]
```

```
            legend_names[i] += f", Slope: {slope:.3f}, Bias: {bias:.3f}"

        plt.xlabel("log(Number of Tokens)")
        plt.ylabel("log(Number of Word Types)")
        plt.title(f"Log-Log Plot for the Each {class_name} Book (No Stop Word)
")
        plt.legend(legend_names)
        plt.show()

    # ploting the log-log plots for each all types of books
    colors = ["b", "g", "r"]
    color_index = 0

    legend names = []
    curve_number = 0

    for class_name in vocabulary_file[preprocess_type][book_class]:

        temp_legend_names =
os.listdir(f"{preprocess_type}\{book_class}\{class_name}")
        legend_names += [name.split(".txt")[0] for name in temp_legend_names]

        for i in [-3, -2, -1]:
            number_of_words_types =
corpus_size_file[preprocess_type][book_class][class_name][i]
            number_of_tokens = np.array(range(1, len(number_of_words_types) +
1)) * 5000

            plt.plot(np.log(number_of_tokens), np.log(number_of_words_types),
"o-", color=colors[color_index], label=class_name)

            slope, bias =
corpus_size_slope_file[preprocess_type][book_class][class_name][i]

            legend_names[curve_number] += f" ({class_name}), Slope:
{slope:.3f}, Bias: {bias:.3f}"

            curve_number += 1

        color_index += 1

    print("*** slope bias table for the preprocesed and (No Stop Word) book
types ***")
    for i in legend_names:
        print(i)
    print("\n\n")

    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title(f"Log-Log Plot for the Each Book of all Authors (No Stop Word)
")
    plt.legend(legend_names)
    plt.show()

def Random_corpus_generator():

    alfabe = "abcdefghijklmnopqrstuvwxyz "
```

```
    random_corpora = ''.join(rd.choice(alfabe) for _ in range(2500000))
    f = open(f"{random_corpus_file_name}", 'w', encoding='utf-8')
    f.write(random_corpora)
    f.close()

def Part_m():

    temp_text = ""

    f = open(f"{random_corpus_file_name}", 'r', encoding='utf-8')

    for line in f.readlines():
        temp text += line

    f.close()

    temp_text_words = temp_text.split()

    # ploting the log(token count) vs. log(corpus size) plot for random corpus
    number_of_words_types = Vocabulary_counter(temp_text_words, 5000)
    number_of_tokens = np.array(range(1, len(number_of_words_types) + 1)) *
5000

    plt.plot(np.log(number_of_tokens), np.log(number_of_words_types), "o-",
label="Random Corpus")
    plt.xlabel("log(Number of Tokens)")
    plt.ylabel("log(Number of Word Types)")
    plt.title("Log-Log Plot of token count vs. corpus sizefor the Randomly
Generated Corpus")
    plt.show()

    # ploting the token count vs. corpus size plot for random corpus
    plt.plot(number_of_tokens, number_of_words_types, "o-", label="Random
Corpus")
    plt.xlabel("Number of Tokens")
    plt.ylabel("Number of Word Types")
    plt.title("linear Plot of token count vs. corpus size for the Randomly
Generated Corpus")
    plt.show()



    dictinory = CountFrequency(temp_text_words)
    sorted_dictionary = {k: v for k, v in sorted(dictinory.items(), key=lambda
item: item[1], reverse=True)}
    freqs = list(sorted_dictionary.values())
    ranks = list(range(1, len(sorted_dictionary.keys()) + 1))


    # ploting the linear Zips law plot for random corpus
    plt.scatter(ranks, freqs, s=0.8, label="Randomly Generated Corpus")
    plt.xlabel("Rank")
    plt.ylabel("Frequency")
    plt.title("Linear Zipf's Law Plot for the the Randomly Generated Corpus")
    plt.show()
```

```
    # ploting the log-log Zips law plot for random corpus
    plt.scatter(np.log(ranks), np.log(freqs), s=0.8, label="Randomly Generated
Corpus")
    plt.xlabel("log(Rank)")
    plt.ylabel("log(Frequency)")
    plt.title("Log-Log Zipf's Law Plot for the the Randomly Generated Corpus")
    plt.show()



#pre_process_books()


Part_e() # creats vocabulary files carrying the word types along with their
frequencies.

Part_f() # plots vocabulary files carrying the word types along with their
frequencies.

Part_g() # creats type-token files for combined corpora and plots it

Part_h_i() # creats type-token files for each book, finds their slops and
plots it for authers

Part_j() # creats type-token files for each book, finds their slops and plots
it for book types

Part_l() # does all the previus parts but for the no stop word corpus.


Random_corpus_generator()

Part_m() # plot the type-token for the randomly generated corpora
```