# EEE 486 STATISTICAL FOUNDATIONS OF NATURAL LANGUAGE PROCESSING ASSIGNMENT 1 REPORT

**MELİH BERK YILMAZ**

**21803702**

## Table of Contents

**Abstract** .......... 3

**Introduction** .......... 3

**Corpus Consturctıon and Implementation** .......... 3

**Results** .......... 4

1. **Part F:** .......... 4
2. **Part G** .......... 6
3. **Part H** .......... 7
4. **Part I** .......... 7
5. **Part J** .......... 8
6. **Part K** .......... 10
7. **Part L** .......... 11

   ***7.1. Part G when Stop Words are Removed*** .......... *11*

   ***7.2 Part H and I when Stop Words are Removed*** .......... *11*

   ***7.3 Part J when Stop Words are Removed*** .......... *13*

   ***7.4 Part K when Stop Words are Removed*** .......... *14*

8. **Part M** .......... 15

**Conclusion** .......... 17

**References** .......... 18

**Appendıx** .......... 18

## ABSTRACT

This report provides a comprehensive analysis of various statistical laws in Natural Language Processing (NLP) by performing basic preprocessing and tokenization on selected books. The report includes analyses of the vocabulary size concerning increasing token size and word frequency based on its rank for 18 different books. Additionally, the report evaluates the impact of removing stop words from the corpus on improving Zipf's Law. Overall, this assignment aims to provide a fundamental understanding of statistical laws in NLP and their effects on the results.

## INTRODUCTION

In this assignment, the objective is to perform basic preprocessing and tokenization steps on some selected books. After the corpuses are preprocessed and tokenized, some of the statistical laws are analyzed on the dataset. Vocabulary size with respect to increasing token size and frequency of the usage of each word with respect to its rank are analyzed for 18 different books.

To form a clustered data, 9 books are selected from 3 authors, each of which has 3 books. Also, 9 books are selected from 3 genres, each of which has 3 books. Then, the relation between the books of each author and books of each genres are compared with other authors' and genres' books respectively.

Furthermore, the effect of removal of stop words from the tokenized corpus is analyzed in terms of improvement in the Zipf's Law. Basically, this assignment aims to gain the basic understanding of some of the statistical laws of Natural Language Processing (NLP), their implementations and their effects on the results.

## CORPUS CONSTURCTION AND IMPLEMENTATION

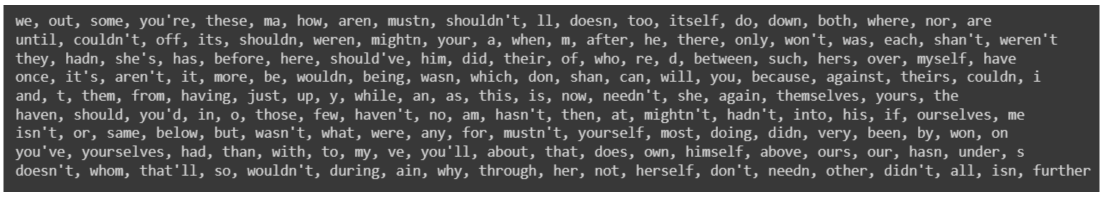
To obtain more satisfactory results, books are selected with sizes higher than 1 Mb or closer to 1 Mb. Also, all books were superficially examined for the presence of special character and the presence of any poetry. The selected authors are: Leo Tolstoy, Charlotte Bronte and Fyodor Dostoyevsky. The selected genres are: Classic Literature, Romance and Science Fiction.

- Leo Tolstoy:
  1. Anna Karenina
  2. Resurrection
  3. War and Peace
- Charlotte Bronte
  1. Jane Eyre
  2. Shirley
  3. Villette
- Fyodor Dostoyevsky
  1. Crime and Punishment
  2. The Idiot
  3. The Brothers Karamazov

- Classic Literature
  1. Les Misérables
  2. Moby Dick
  3. The Count of Monte Cristo
- Romance
  1. Emma
  2. Middlemarch
  3. The Age of Innocence
- Science Fiction
  1. A Voyage to Arcturus
  2. The Country of The Blind
  3. Varny The Vampire

After books are selected, preprocessing and tokenization parts are implemented. Firstly, punctuation marks are removed and each word is casted to lowercase. Apostrophe is directly removed, meaning that "isn't" is converted to "isnt" etc. Then, common English stop words are removed from the tokenized corpus. At the end, 18 processed and tokenized corpuses and 18 processed, tokenized and stop words removed corpuses are obtained.

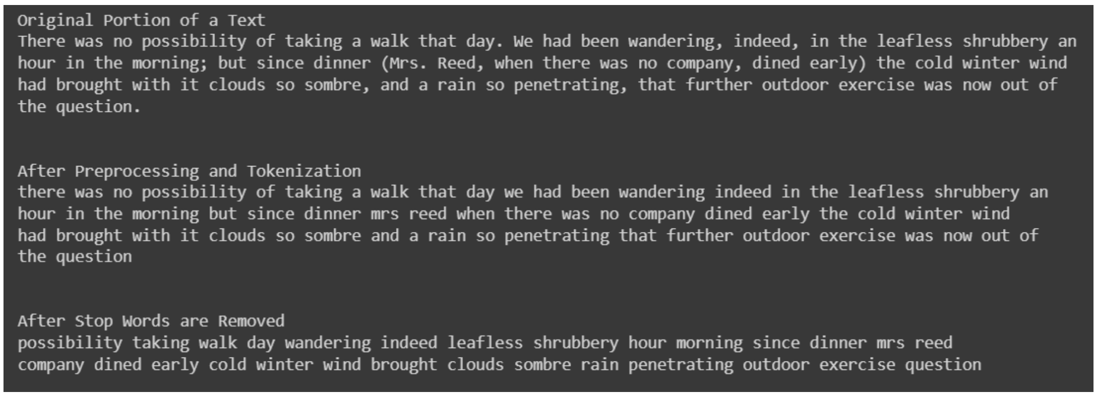
These stop words were taken from nltk library which is a common NLP library in Python. [1] This library is not directly used for any steps in the assignment, only the stop word list was downloaded and converted to a list object. All the tokenization and preprocessing steps were implemented from scratch.



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

Figure 1: Common English Stop Words List Gathered from nltk Library [1]

A paragraph from Jane Eyre by Charlotte Bronte is processed to show the intermediate steps of tokenization explained in part c and d.



```
Original Portion of a Text
There was no possibility of taking a walk that day. We had been wandering, indeed, in the leafless shrubbery an
hour in the morning; but since dinner (Mrs. Reed, when there was no company, dined early) the cold winter wind
had brought with it clouds so sombre, and a rain so penetrating, that further outdoor exercise was now out of
the question.


After Preprocessing and Tokenization
there was no possibility of taking a walk that day we had been wandering indeed in the leafless shrubbery an
hour in the morning but since dinner mrs reed when there was no company dined early the cold winter wind
had brought with it clouds so sombre and a rain so penetrating that further outdoor exercise was now out of
the question


After Stop Words are Removed
possibility taking walk day wandering indeed leafless shrubbery hour morning since dinner mrs reed
company dined early cold winter wind brought clouds sombre rain penetrating outdoor exercise question
```

Figure 2: Preprocessing and Tokenization Steps

## RESULTS

### 1. Part F:

In this part, firstly all the three books of each author are composed with each other. At the end, there are 3 corpuses one for each author. Then, each corpus is converted into a dictionary with keys correspond to words and values correspond to their frequencies. Since the dictionary is sorted by value, rank is obtained by the order of the dictionary element.

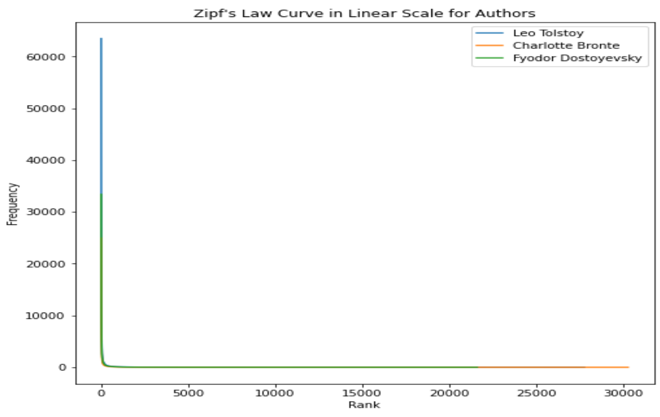


Figure 3: Zipf's Law Curve for Compose Corpus of Each Author in Linear Scale

Zipf's Law is formulized as $f(t) = C * r(t)^b$ (1), $where\ f(t)$ is the frequency of word t, $r(t)$ is the rank of the word t, $C$ is normalization constant and b is the Zipf exponent, generally between -1 and 0. Thus, the frequency of a word in a corpus is inversely proportional to its rank in the corpus. [2] Figure 3 shows the exact relation as Zipf's Law states. As the rank of the word increases its frequency decreases exponentially and vice versa. This relation can be observed clearly for all of the authors from figure 3. Note that these corpuses include stop words.

Then, each book's corpus is individually plotted for Zipf's Law.

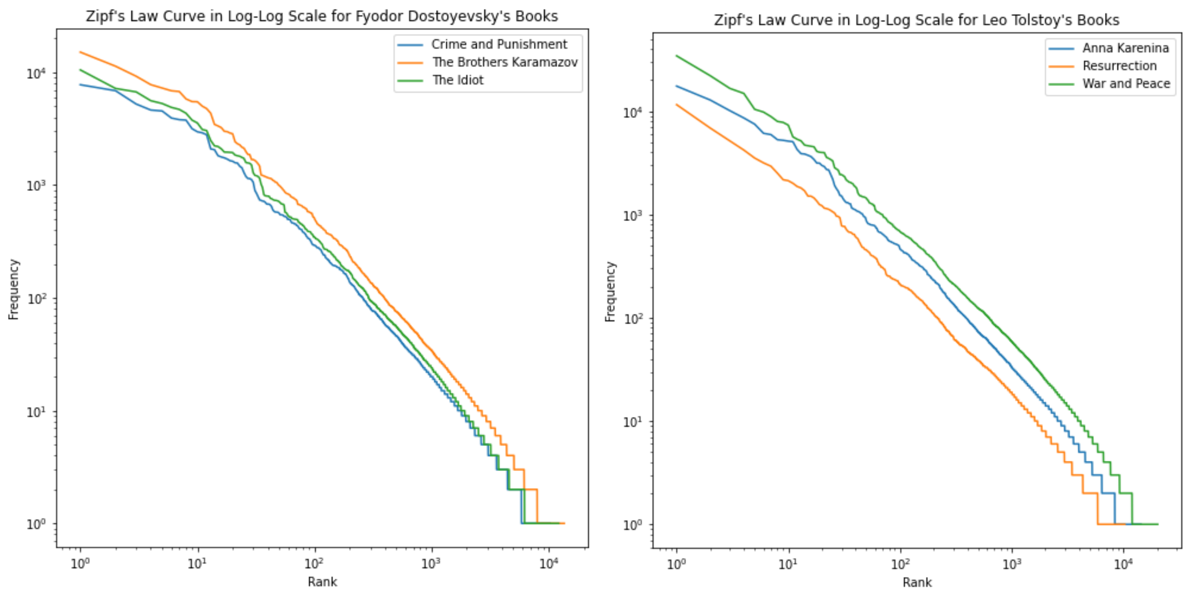


Figure 4: Zipf's Law Plot for Books of Each Author in Log-Log Scale

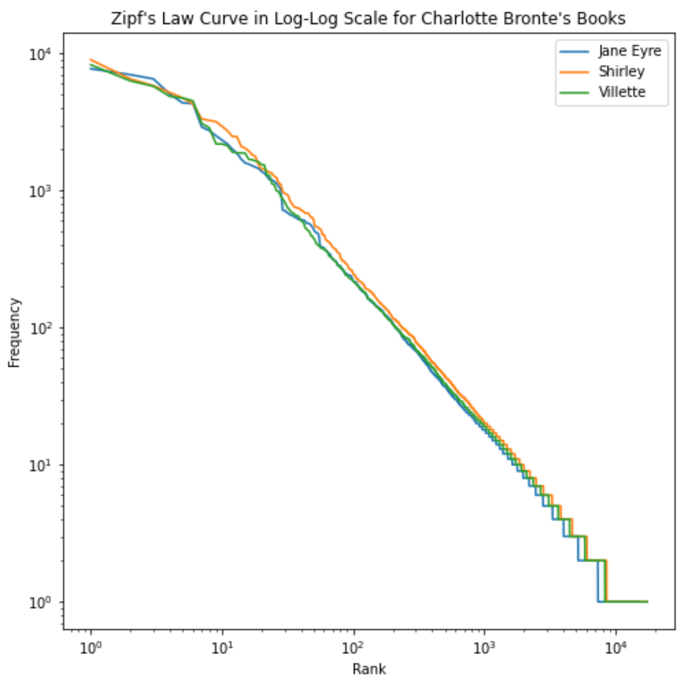


Figure 4 (cont'd): Zipf's Law Plot for Books of Each Author in Log-Log Scale

Again, these Zipf's Law Curves for each book obeys the rule. This can be understood well taking the logarithm of both sides of the equation (1).

$$\log f(t) = \log(C) + b * \log r(t) \ (2)$$

The equation (2) is a linear relationship with slope $b$ and interception $\log(C)$ in the logarithmic scale. It is clear that from figure 4, the slop of the curves are near -1. As a results, all of these 9 curves obey the Zipf's Law as it is expected.

## 2. Part G

In this part, the relation between the token size and vocabulary size is examined. The number of unique words is kept track for every 10,000 words. Then, the resulting list is plotted for normal scale and log-log scale.

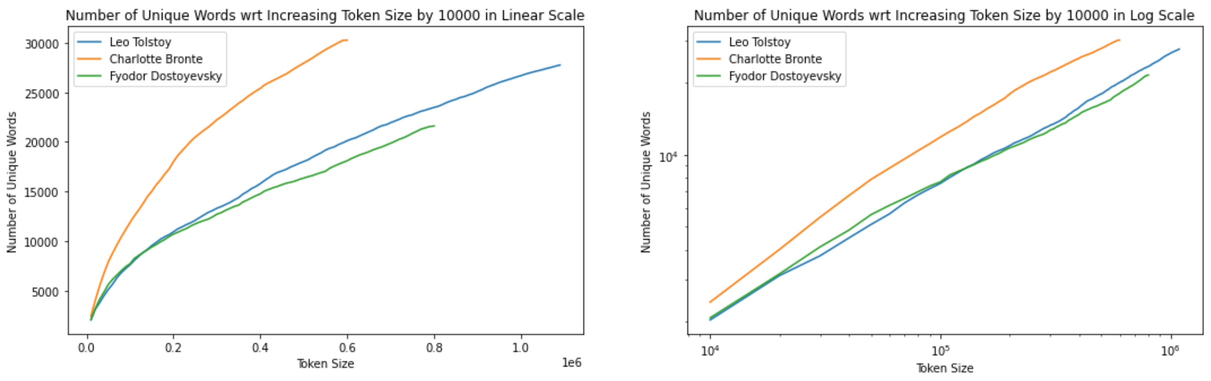


Figure 5: Number of Unique Words w.r.t Increasing Token Size by 10,000 in Normal *(Left)* and Log-Log *(Right)* Scale

As it is seen from figure 5 *(left)*, the number of unique words increases with a decreasing rate as the token size increases. This is normal since the possibility of usage of different word for higher token size is low. Figure 5 *(right)* shows the same relation in the logarithmic scale. In this plot the relation is linear since the curves in the left plot seem like logarithmic plots. This relation can be formulized as: $V = k * T^b$ (3), where $V$ is the vocabulary size, $k$ is the constant, $T$ is the token size and b is the exponent term which is between 0 and 1. Let's apply the same procedure as in the first equation, take the logarithm of both sides. Then,

$$\log V = \log k + b * \log T \ (4)$$

As a result, there is a linear relation between the vocabulary size w.r.t increasing token size in logarithmic scale with a slope of b, which verifies the figure 5 *(right)*

This relation (equation 3) is called as Heap's Law. Basically, Heaps' law states that as the amount of text data increases, the rate at which new unique words are discovered decreases. In other words, the more text that is added, the less likely it is to encounter new words that haven't been seen before. [3]

The parameters in this relation is constant k and exponent b. The parameter k is the scaling factor that determines the overall slope of the relation between V (vocabulary size) and T (token size). The parameter b is called "vocabulary exponent" determines the rate of growth of vocabulary size w.r.t increasing token size or corpus size.

## 3. Part H

In this part, the relation between the vocabulary size w.r.t increasing token size is examined for all of the books of authors individually. The relation is plotted in the log-log scale. Also, the same relation for the composition of corpuses of each author is wanted, but it is shown in the figure 5 *(right)* and explained in the previous part.

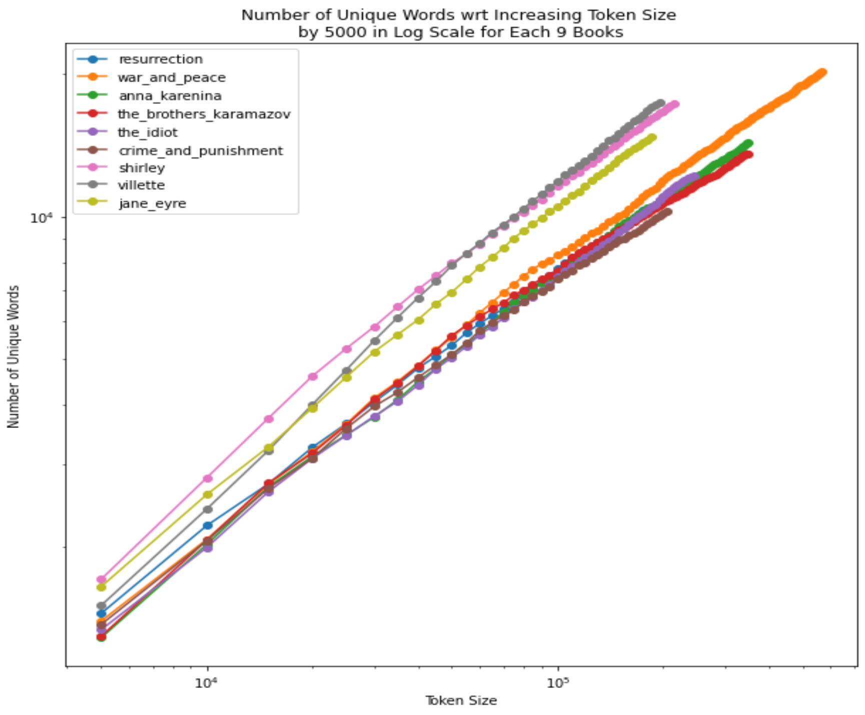


Figure 6: Number of Unique Words w.r.t Increasing Token Size by 5,000 in Log-Log Scale for Each 9 Books

As seen from figure 6, the relation between the number of unique words w.r.t increasing token size is linear in logarithmic scale, as it is expected from the Heap's Law. Even though the corpuses are not composed, they obey the Heap's Law with their individual corpuses as well since the length of each corpus is sufficient to observe the relation. As aforementioned, Figure 5 *(right)* shows the same relation for the composition of corpuses of each author. Since each book individually obeys the Heap's Law, their composition obeys as well.

## 4. Part I

In this part, the best fitting lines are found for the line representing the relation between the number of unique words w.r.t increasing token size for each book. When nine books relation curve and their corresponding best fit curves are plotted in the same figure, it seems confusing, therefore only three of them (one for each author) is presented in the figure.

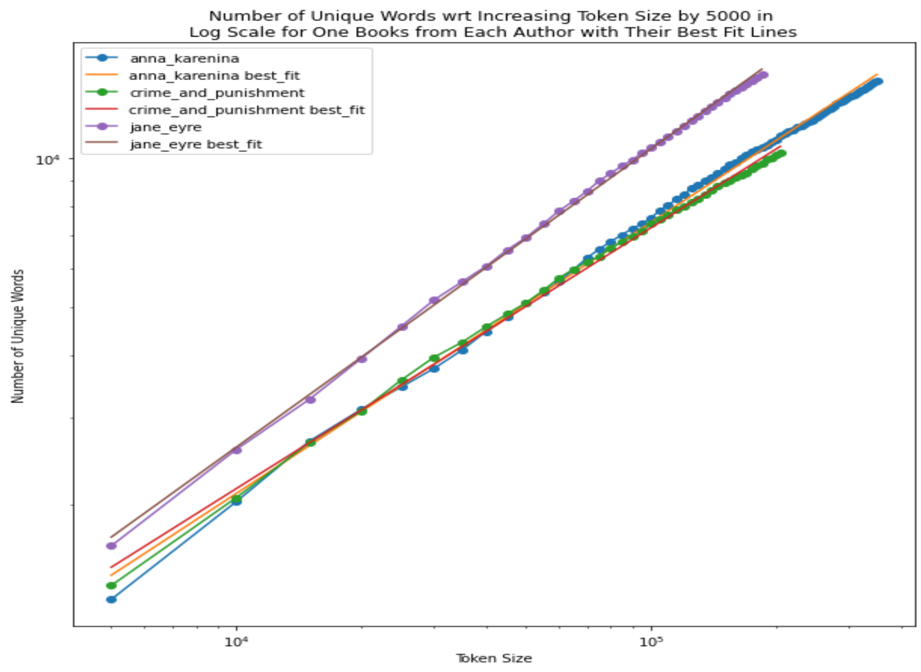


Figure 7: The Best Fitting Lines for the Curves Representing The Relation between The Vocabulary size w.r.t Increasing Token Size for One Book from Each Author

| | Authors | Book Names | Interception | Slope |
|---|---|---|---|---|
| 0 | Leo Tolstoy | Resurrection | 1.183568 | 0.540576 |
| 1 | Leo Tolstoy | War and Peace | 1.167179 | 0.547414 |
| 2 | Leo Tolstoy | Anna Karenina | 1.134301 | 0.547377 |
| 3 | Fyodor Dostoyevsky | The Brothers Karamazov | 1.313780 | 0.511828 |
| 4 | Fyodor Dostoyevsky | The Idiot | 1.053626 | 0.563567 |
| 5 | Fyodor Dostoyevsky | Crime and Punishment | 1.227881 | 0.526434 |
| 6 | Charlotte Bronte | Shirley | 1.127332 | 0.586299 |
| 7 | Charlotte Bronte | Villette | 0.779380 | 0.657954 |
| 8 | Charlotte Bronte | Jane Eyre | 1.009208 | 0.601987 |

Table 1: Table of Parameters of Best Fitting Lines for Each Book

As it is found in the equation (4), the curves are linear in the logarithmic scale with a slope of b, which is a real number between 0 and 1. The results in the table 1 are consistent with our findings from Heap's Law. Mainly, the slopes are between $[0.5, 0.7]$. This is expected since the growth rate of vocabulary size decreases as token size increases.

## 5. Part J

In this part, the same procedures in Part H and I are applied for the 9 books corresponding to three different genres. Firstly, the relation between the vocabulary size w.r.t increasing token size is examined for each book.

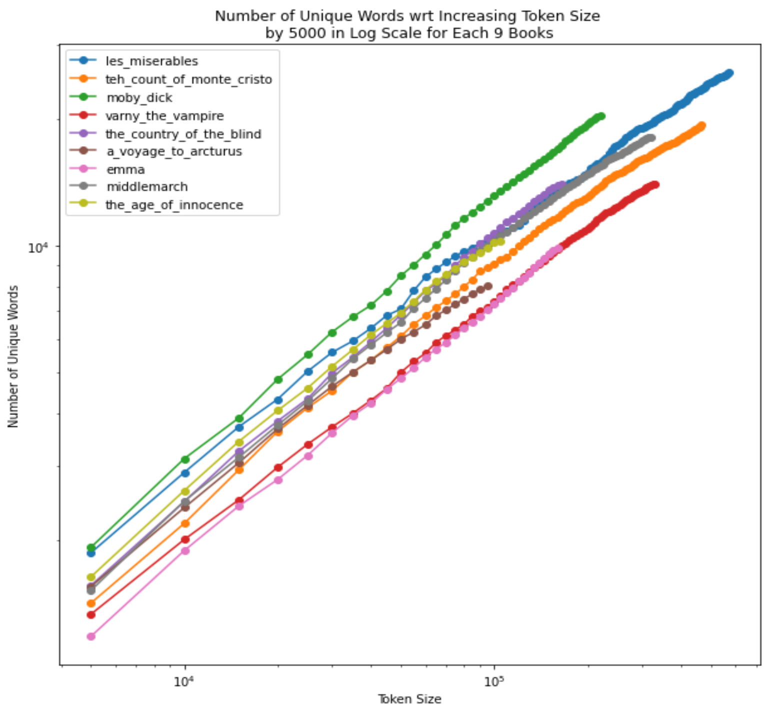


Figure 8: Number of Unique Words w.r.t Increasing Token Size by 5,000 in Log-Log Scale for Each 9 Books

As it is seen from the figure 8, there is a linear relation between the token size and vocabulary size in logarithmic plot as Heap's Law states from equation (4). The results are consistent with the Heap's Law and the results in the Part H.

Then, the best fitting lines for each curve is found. Again, only the best fitting lines with their corresponding curves for one book from each genre is represented to prevent confusion in the figure.

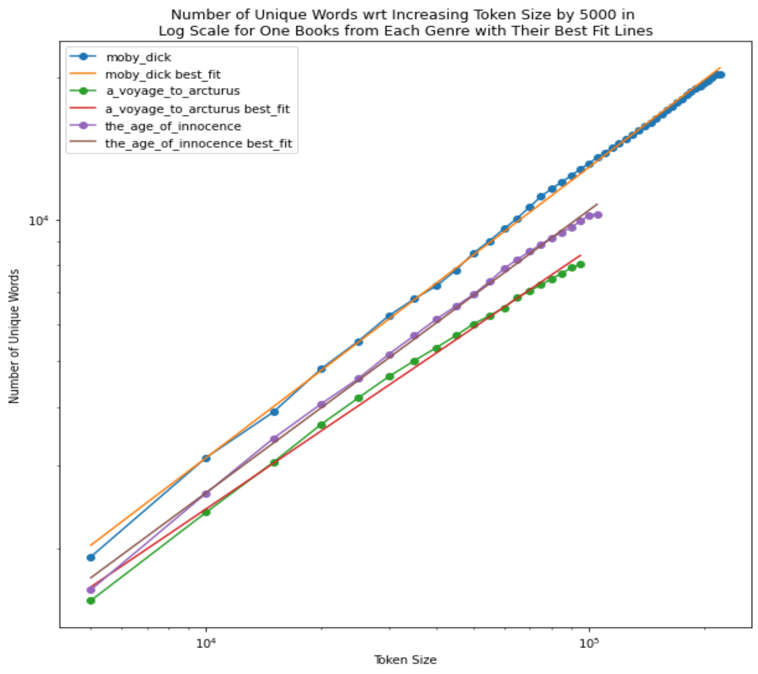
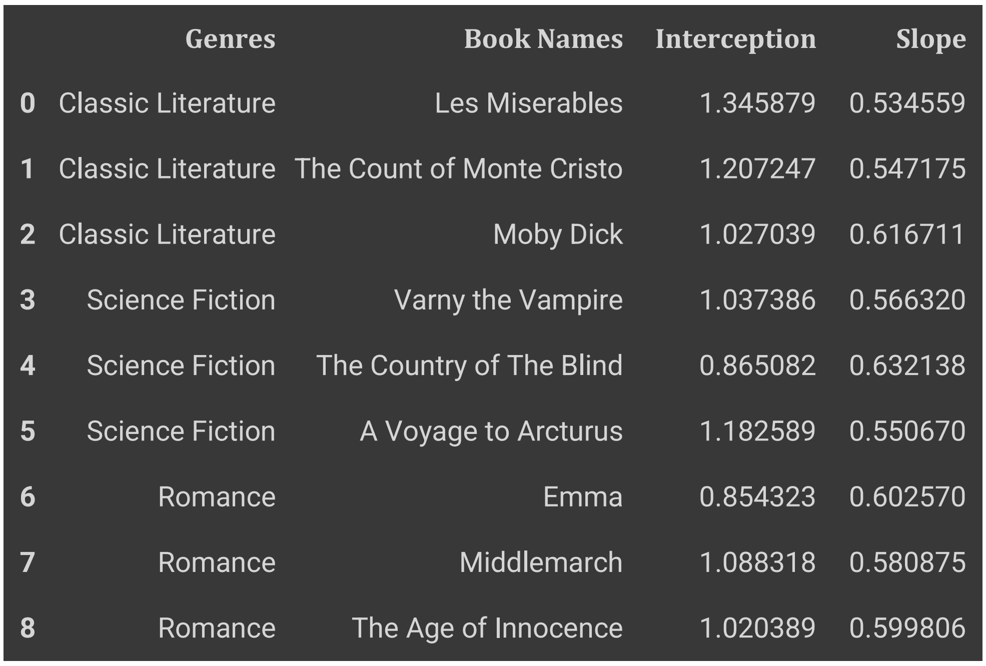


Figure 9: The Best Fitting Lines for the Curves Representing The Relation between The Vocabulary size w.r.t Increasing Token Size for One Book from Each Genre



| | Genres | Book Names | Interception | Slope |
|---|---|---|---|---|
| 0 | Classic Literature | Les Miserables | 1.345879 | 0.534559 |
| 1 | Classic Literature | The Count of Monte Cristo | 1.207247 | 0.547175 |
| 2 | Classic Literature | Moby Dick | 1.027039 | 0.616711 |
| 3 | Science Fiction | Varny the Vampire | 1.037386 | 0.566320 |
| 4 | Science Fiction | The Country of The Blind | 0.865082 | 0.632138 |
| 5 | Science Fiction | A Voyage to Arcturus | 1.182589 | 0.550670 |
| 6 | Romance | Emma | 0.854323 | 0.602570 |
| 7 | Romance | Middlemarch | 1.088318 | 0.580875 |
| 8 | Romance | The Age of Innocence | 1.020389 | 0.599806 |

Table 2: Table of Parameters of Best Fitting Lines for Each Book

As it is found in the equation (4), the curves are linear in the logarithmic scale with a slope of b, which is a real number between 0 and 1. The results in the table 1 are consistent with our findings from Heap's Law. Mainly, the slopes are between $[0.5, 0.7]$. This is expected since the growth rate of vocabulary size decreases as token size increases. Consequently, both books of different authors and genres obey the Heap's Law.

## 6. Part K

The author based books can be clustered using the results from Part G, H, I and J with some exception cases. For example, Charlotte Bronte's books can be totally differentiated from other authors' books as seen in figure 6. This can be observed from the table 1 as well. The slopes and interceptions of Charlotte Bronte's books are very close to each other and considerably higher than the others. On the other hand, Leo Tolstoy's and Fyodor Dostoyevsky's books cannot be clustered into 2 clusters easily. There are still differences between their slopes, interceptions and curve shapes. However, it is not as exact as in the case of Charlotte Bronte. The slopes and interceptions of Leo Tolstoy's books are very close to each other, in that way, one can cluster his books and differentiate from Fyodor Dostoyevsky.

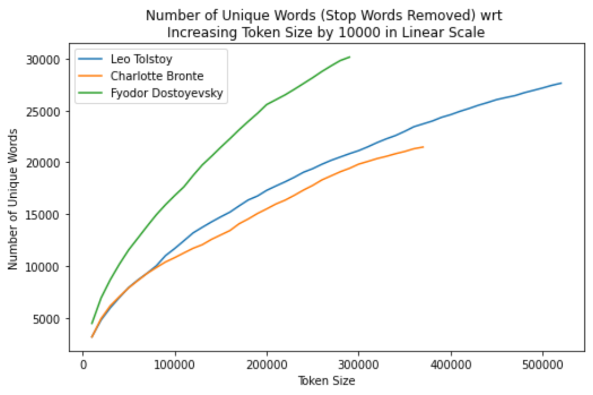
The genre based books cannot be clustered easily using the results from Part G, H, I and J. Actually, this would be the case if the genres are completely unrelated for example horror and philosophy. Since Romance, Classic Literature and Science Fiction (this is not that much) are related to each other, the separation is not exact, at least from these results.

In general, the number of author or type of genre is not a parameter for clustering. Of course, if the number of author or type of genre is too high, the clustering method would have a difficulty to converge with using only these findings. However, the main criteria for clear separation of clusters is not dependent on the number of type, rather it is dependent on the relation and uniqueness of the books of authors or genres. As it is seen in this example, three authors can be easily clustered since each of them has their unique ways of writing. However, since the type of genres are quite correlated to each other, the separation is not that exact. Nevertheless, using only the results of slope, interception and vocabulary size vs token size relation wouldn't be adequate for clear separation of clusters for maybe 5, 6 authors or genres.

## 7. Part L

In this part, the effect of removal of stop words is examined. The Part G, H, I, J and K are applied again to corpuses without stop words.

### *7.1. Part G when Stop Words are Removed*



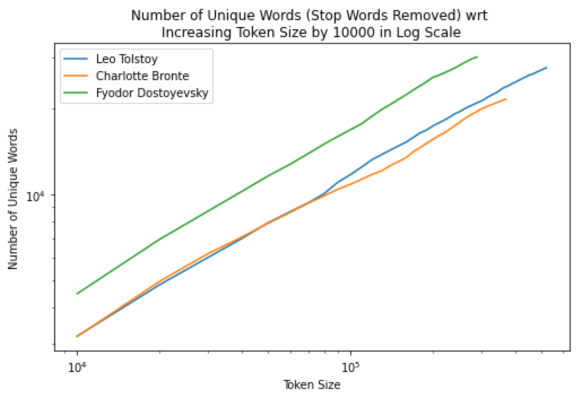


Figure 10: Number of Unique Words w.r.t Increasing Token Size by 10,000 in Normal *(Left)* and Log-Log *(Right)* Scale

As it can be seen from figure 10 and figure 5, there is not much a difference as it is expected. Because Heap's Law doesn't depend on the presence of the stop words. It is law stating that the relation between the corpus size and vocabulary size. Therefore, the only difference is the number of unique words, which only differs by 179 since the length of the stop words list is 179.

### *7.2 Part H and I when Stop Words are Removed*

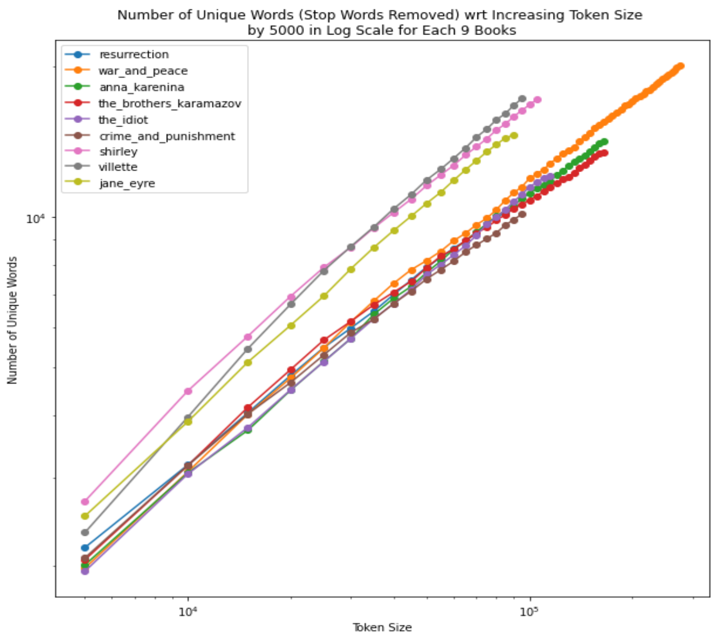


Figure 11: Number of Unique Words w.r.t Increasing Token Size by 5,000 in Log-Log Scale for Each 9 Books

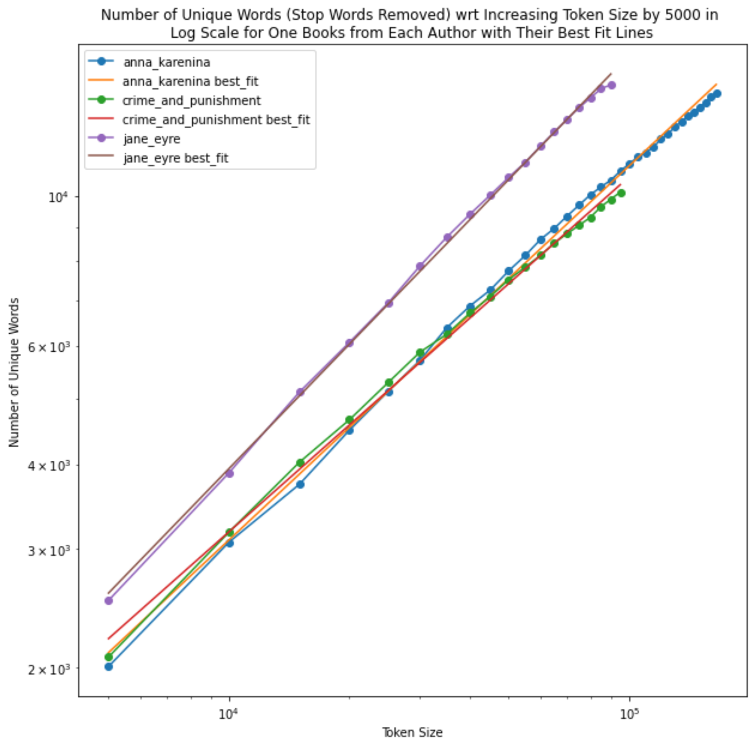
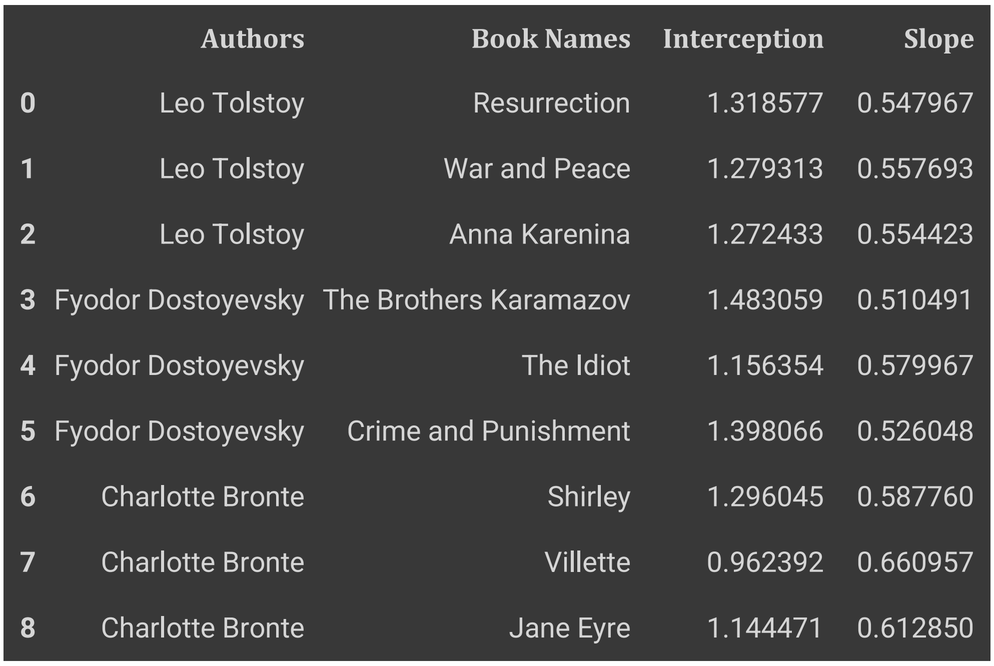


Figure 12: The Best Fitting Lines for the Curves Representing The Relation between The Vocabulary size w.r.t Increasing Token Size for One Book from Each Author



| | Authors | Book Names | Interception | Slope |
|---|---|---|---|---|
| 0 | Leo Tolstoy | Resurrection | 1.318577 | 0.547967 |
| 1 | Leo Tolstoy | War and Peace | 1.279313 | 0.557693 |
| 2 | Leo Tolstoy | Anna Karenina | 1.272433 | 0.554423 |
| 3 | Fyodor Dostoyevsky | The Brothers Karamazov | 1.483059 | 0.510491 |
| 4 | Fyodor Dostoyevsky | The Idiot | 1.156354 | 0.579967 |
| 5 | Fyodor Dostoyevsky | Crime and Punishment | 1.398066 | 0.526048 |
| 6 | Charlotte Bronte | Shirley | 1.296045 | 0.587760 |
| 7 | Charlotte Bronte | Villette | 0.962392 | 0.660957 |
| 8 | Charlotte Bronte | Jane Eyre | 1.144471 | 0.612850 |

Table 3: Table of Parameters of Best Fitting Lines for Each Book

As seen from figure 11-12 and figure 6-7; and table 1 and 3, there is not much difference between the curves and their best fitting lines. There are minor changes in their slope, however, results are not completely different from each other. Still, Charlotte Bronte's books are quite differentiable from other authors' books as seen in figure 11. The relation between the slopes for different authors are consistent with the table 1.

### *7.3 Part J when Stop Words are Removed*

In this part, part H and I will be repeated for genres based books without stop words.

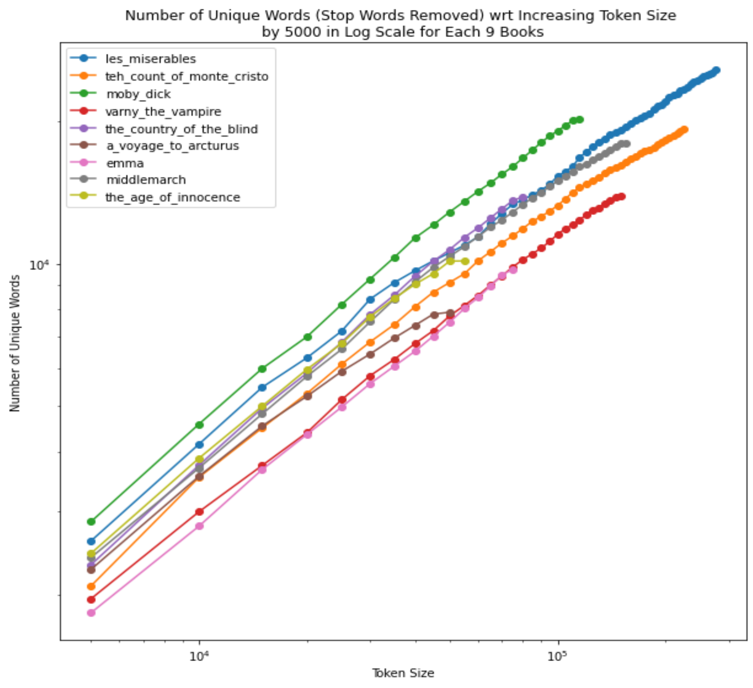


Figure 13: Number of Unique Words w.r.t Increasing Token Size by 5,000 in Log-Log Scale for Each 9 Books

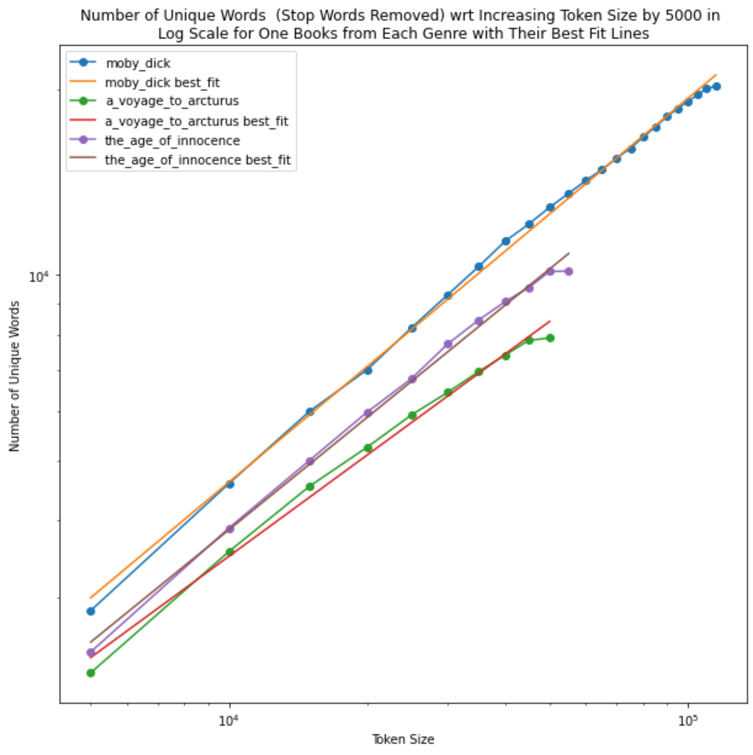
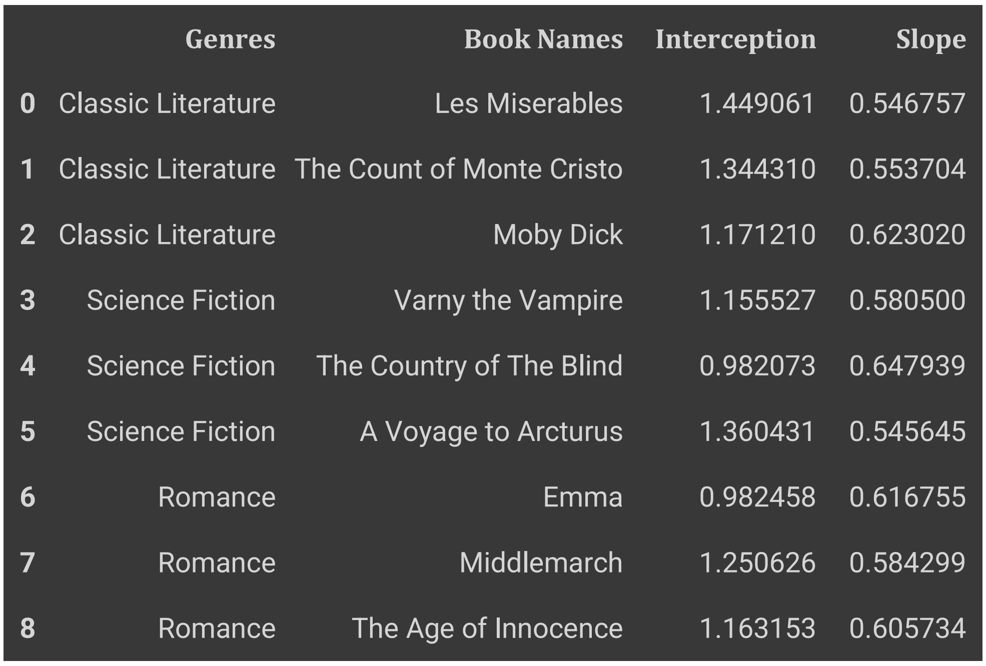


Figure 14: The Best Fitting Lines for the Curves Representing The Relation between The Vocabulary size w.r.t Increasing Token Size for One Book from Each Genre



| | Genres | Book Names | Interception | Slope |
|---|---|---|---|---|
| 0 | Classic Literature | Les Miserables | 1.449061 | 0.546757 |
| 1 | Classic Literature | The Count of Monte Cristo | 1.344310 | 0.553704 |
| 2 | Classic Literature | Moby Dick | 1.171210 | 0.623020 |
| 3 | Science Fiction | Varny the Vampire | 1.155527 | 0.580500 |
| 4 | Science Fiction | The Country of The Blind | 0.982073 | 0.647939 |
| 5 | Science Fiction | A Voyage to Arcturus | 1.360431 | 0.545645 |
| 6 | Romance | Emma | 0.982458 | 0.616755 |
| 7 | Romance | Middlemarch | 1.250626 | 0.584299 |
| 8 | Romance | The Age of Innocence | 1.163153 | 0.605734 |

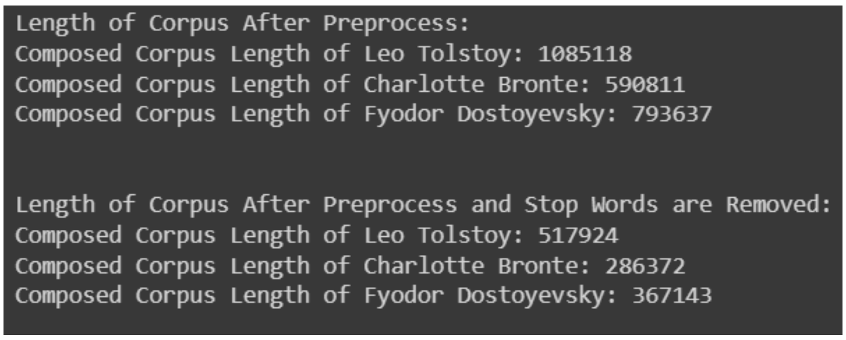
Table 4: Table of Parameters of Best Fitting Lines for Each Book

As seen from figure 13-14 and figure 8-9; and table 2 and 4, there is not much difference between the curves and their best fitting lines. There are minor changes in their slope, however, results are not completely different from each other. Still, each genre cannot be easily differentiated from each other using the results with stop words removed as seen in figure 13. The relation between the slopes for different authors are consistent with the table 1.

### *7.4 Part K when Stop Words are Removed*

As seen in Section 7.1-7.3, there is not much a difference between the results of normal corpus and stop words removed corpus. Therefore, the results of part K doesn't changed. Figure 11 shows the clear separation of Charlette Bronte's books and the table 1 shows the separation of Leo Tolstoy's and Fyodor Dostoyevsky's books based on the differences in the slopes of their corresponding books.

Furthermore, both figure 13 and table 4 shows that there is no clear separation between the genres as in the part K because of the fact that these genres are quite related. Only the Classical Literature is partially differentiable from other genres. As a result, removing the stop words doesn't change or affect the overall results. However, it greatly decrease (almost the half or more) the size of the corpus and results in a very efficient way of implementation of preprocessing and tokenization algorithms.



```
Length of Corpus After Preprocess:
Composed Corpus Length of Leo Tolstoy: 1085118
Composed Corpus Length of Charlotte Bronte: 590811
Composed Corpus Length of Fyodor Dostoyevsky: 793637


Length of Corpus After Preprocess and Stop Words are Removed:
Composed Corpus Length of Leo Tolstoy: 517924
Composed Corpus Length of Charlotte Bronte: 286372
Composed Corpus Length of Fyodor Dostoyevsky: 367143
```

Figure 15: The Length of The Composed Corpus with Normal and Stop Words Removed Type

As seen from figure 15 and Part 7.1-7.4, removing stop words both decreases the size of the corpus, implementation time of the algorithms and doesn't negatively affect the overall results.

## 8. Part M

In this part, a randomly generated corpus is created to analyze the presence of the Zipf's and Heap's Law for random corpus. To do this, a random corpus is created based on the algorithm presented in the paper of W. Li. [4] In this algorithm, firstly $(M + 1)$ symbols are created with one being a blank space. Then, a uniform random generator generates a number between 1 and $M + 1$. Assume it generated i, then, $i^{th}$ symbol is added to initial empty string. After a specific length of string is filled by symbols, corpus is created by taking each sequence of symbols between two blank spaces as a word. [4] Using this algorithm, a corpus with 300,000 words is created to analyze the statistical laws. Firstly, the procedure in Part F is implemented, the frequency of each word and its rank are found and plotted in both linear and logarithmic scale.

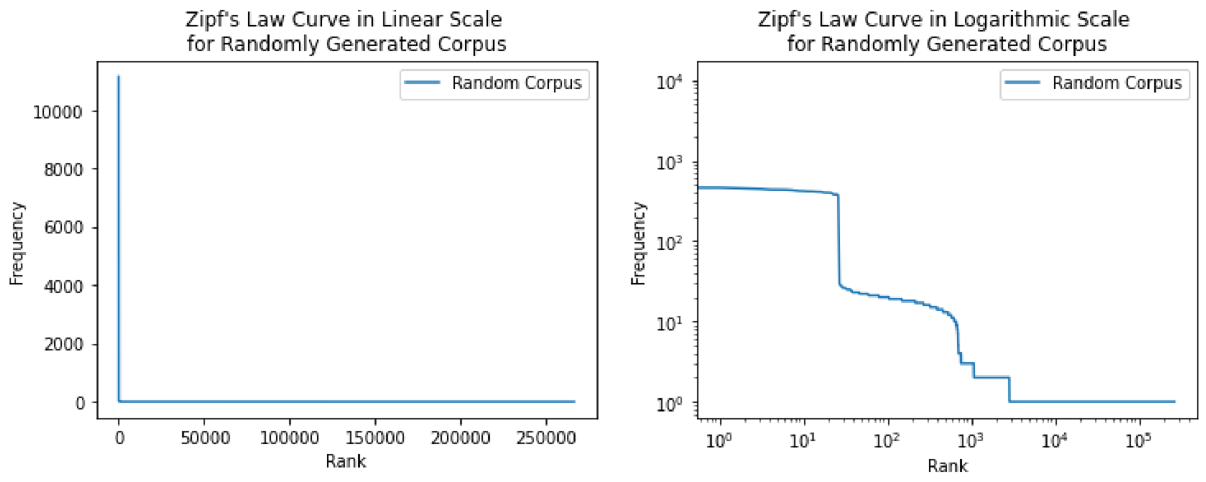


Figure 16: Zipf's Law Curve for Randomly Generated Corpus in Linear *(Left)* and Logarithmic *(Right)* Scale

Figure 16 *(Left)* seems like the random corpus obeys the Zipf's Law, however, the logarithmic plot (Figure 16 right) shows that it doesn't completely obey the Zipfian behavior. It seems that some of the words are used with a very high frequency whereas some of the words are used with a very low frequency. However, there is a huge difference between the frequency of the words whose ranks are in the middle of frequent and rare ones. the decrease of frequency of a word doesn't linearly drops in the logarithmic scale. This is because of the fact that the words are completely meaningless and random. Therefore, there are lots of possibilities of creating new word. Actually, the length of a word is limited with 6 to prohibit the increase in the randomness of a word. Otherwise, this curve would be worse. Still, it obeys the Zipf's Law partially, however, much more words are needed for random corpus to fully obey the Zipf's Law compared to the corpus of natural languages.

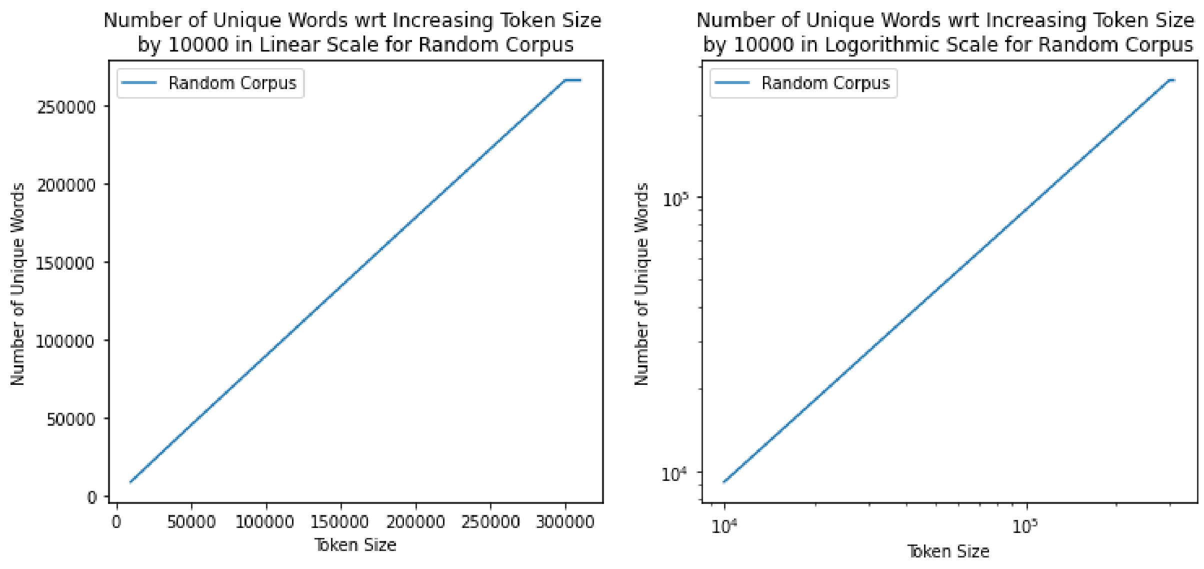


Figure 17: Vocabulary Size w.r.t Increasing Token Size in Linear *(Left)* and Logarithmic *(Right)* Scale

As seen in figure 17 *(left),* the relation between the vocabulary size and increasing token size is equal to the equation (3) for the case $b = 1$. However, in Heap's Law, the expectation is that b is less than 1. Therefore, Heap's Law cannot be fully observed in this case as well. It partially obeys the rule, however, it is expected that the growth rate of vocabulary size decreases as token size increases.

As aforementioned, this stems from the fact that the possibility of creating new word is quite high for random corpus even though the maximum length is limited with 6. If the corpus length is increased to much higher numbers such as 10 million, the Zipf's and Heap's Law can be clearly examined.

For example, a random corpus with a length of 300,000 and a maximum word length restricted to 3 is created. The number of unique words is decreased from $26^3 = 17576$ to $26^6 \sim 300\ million.$

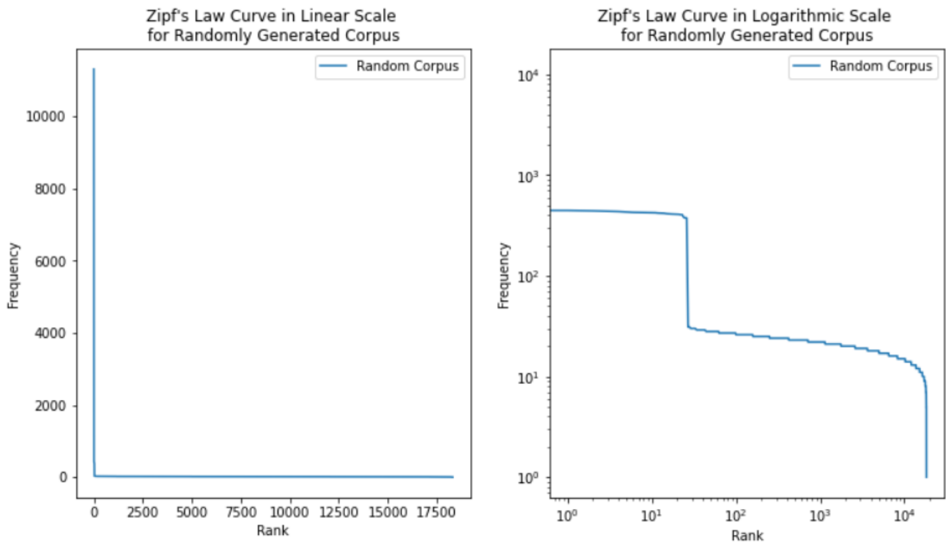


Figure 18: Zipf's Law Curve for Randomly Generated Corpus in Linear *(Left)* and Logarithmic *(Right)* Scale

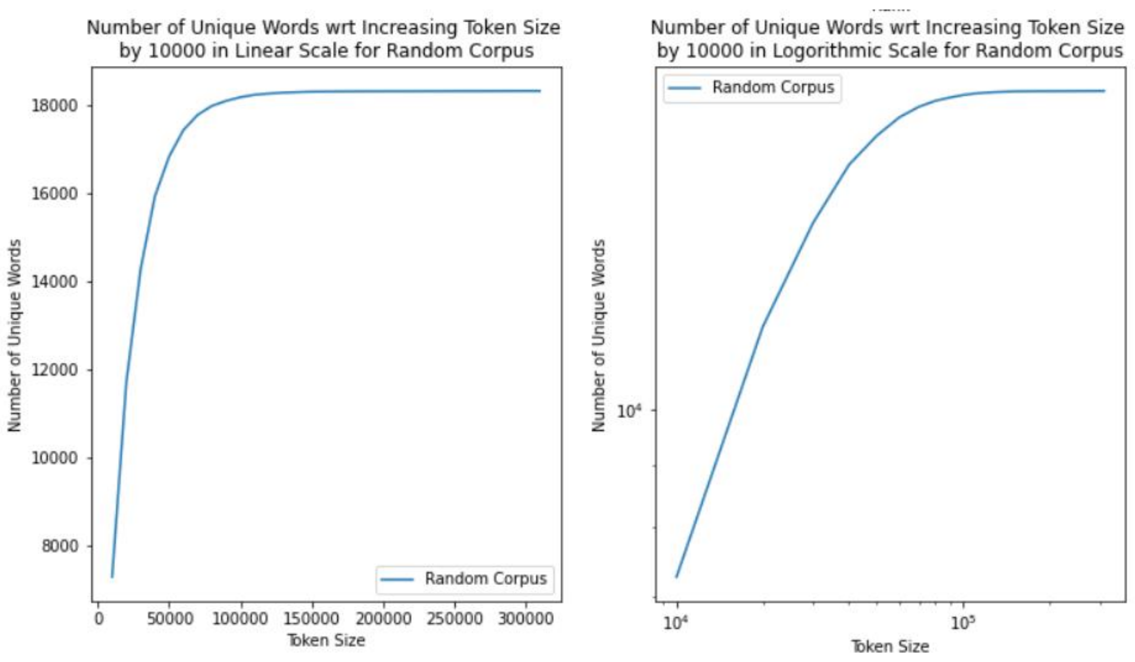


Figure 19: Vocabulary Size w.r.t Increasing Token Size in Linear *(Left)* and Logarithmic *(Right)* Scale

Figure 18 and 19 obeys the Zipf's Law and Heap's Law respectively. Still, there is a high decrease in a short interval in figure 18, but the rest of the figure is consistent. In figure 19, the growth rate of vocabulary size decreases rapidly as it is stated by the Heap's Law.

Consequently, randomly generated corpus obeys both the Zipf's and Heap's Law if and only if the length of the corpus is sufficient enough. This sufficiency rate is quite higher compared to the corpus created from a natural language.

## CONCLUSION

In conclusion, this assignment aimed to perform basic preprocessing and tokenization steps on selected books, analyze statistical laws on the dataset, and compare the relation between books of different authors and genres. Through analyzing vocabulary size and word frequency, we gained a better understanding of the statistical laws of natural language processing. Additionally, the effect of removing stop words was explored on Zipf's and Heap's Law. The results illustrated that there is not much difference between the results of normal and stop words removed corpuses. Therefore, it is concluded that working with stop words removed corpus is more time and memory efficient. Furthermore, a randomly generated corpus was created and it was examined through some of the essential statistical laws of NLP. It is shown that the word requirement of randomly generated corpus is quite larger compared to corpus created from natural languages to observe the Zipf's and Heap's Law. By completing this assignment, we have gained valuable insights into the fundamental concepts of NLP and their practical applications in data analysis.

## REFERENCES

[1] 262588213843476, “NLTK's list of English stopwords,” *Gist*. [Online]. Available: https://gist.github.com/sebleier/554280. [Accessed: 12-Mar-2023].

[2] “Zipf's Law,” *Wikipedia*, 04-Feb-2023. [Online]. Available: https://en.wikipedia.org/wiki/Zipf%27s_law. [Accessed: 12-Mar-2023].

[3] “Heaps' law,” *Wikipedia*, 18-Feb-2023. [Online]. Available: https://en.wikipedia.org/wiki/Heaps%27_law. [Accessed: 12-Mar-2023].

[4] “W. Li, ”Random texts exhibit Zipf’s-law-like word frequency distribution,” in IEEE Transactions on Information Theory, vol. 38, no. 6, pp. 1842- 1845, Nov. 1992.” [Accessed: 12-Mar-2023].

## APPENDIX

```
# -*- coding: utf-8 -*-
"""EEE 486 Assignment 1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1szLpr-TXSt0Q7VVwvaNlAT8NYp8MSSLv

# **Imports**
"""

import numpy as np
import pandas as pd
import string
import time
import os

from matplotlib import pyplot as plt

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

from google.colab import drive
drive.mount('/content/drive')

!unzip -q /content/drive/MyDrive/books.zip -d /content/
!mv "/content/books/authors/Charlotee Bronte"
"/content/books/authors/Charlotte Bronte"

"""# **Util Functions**"""
```

```python
def tokenization(book_path, stop_word_removal = True):
    book = []
    with open(book_path, 'r') as file:

        stop_words_list = list(set(stopwords.words('english')))

        punctuation_marks = ['.', ',', '?', '!', ';', ':',"’", "'", '"', '(',
')', '[', ']', '-', '—', '/',"“", "”"]
        punctuation_dict = {ord(punctuation): None for punctuation in
punctuation_marks}
        for line in file.readlines():
            line = line.strip().lstrip('\ufeff').lower()
            line = line.translate(punctuation_dict)
            line = line.translate(str.maketrans("’", " ", "‘’" +
string.punctuation))
            line = line.split(" ")
            line = [x for x in line if x not in [""]]
            book.extend(line)

    if stop_word_removal:
        book_without_stop_words = [word for word in book if word not in
stop_words_list]
        return book, book_without_stop_words
    else:
        return book

def create_frequency_dict(book, diction = False):
    word_types, word_frequencies = np.unique(book, return_counts = True)
    word_types = list(word_types)
    word_frequencies = list(word_frequencies)
    # sort by rank
    word_types, word_frequencies = zip(*sorted(zip(word_types,
word_frequencies), key=lambda x: x[1], reverse=True))

    if diction:
        word_freq_dict = dict(zip(word_types, word_frequencies))
        return word_freq_dict
    else:
        return word_types, word_frequencies

def book_compose(dct, stop_word = 0):
    list1, list2, list3 = [], [], []

    for author, books in dct.items():
        for book, values in books.items():
            if author == "Charlotte Bronte":
                list1.extend(values[stop_word])
            elif author == "Fyodor Dostoyevsky":
                list2.extend(values[stop_word])
            elif author == "Leo Tolstoy":
                list3.extend(values[stop_word])
    return list1, list2, list3

def count_word_types(corpus, size):
    num_unique_words = []
    num_iter = len(corpus) // size
    for batch in range(num_iter):
        num_unique_words.append(len(np.unique(corpus[:(batch+1)*size])))

    num_unique_words.append(len(np.unique(corpus)))
    return num_unique_words

def zipf_plot(list_dicts, label_list, scale_type, title):
    plt.figure(figsize=(8, 8))

    books_freqs = []
    ranks_list = []

    for diction in list_dicts:
        freqs = list(diction.values())
        books_freqs.append(freqs)

        ranks = range(1, len(freqs)+1)
        ranks_list.append(ranks)

    if scale_type == "linear":
        for i, (ranks, freqs) in enumerate(zip(ranks_list, books_freqs)):
            plt.plot(ranks, freqs)

        plt.xlabel('Rank')
        plt.ylabel('Frequency')
        plt.title("Zipf's Law Curve in Linear Scale " + title)
        plt.legend(label_list)
        plt.show()

    if scale_type == "log-log":
        for i, (ranks, freqs) in enumerate(zip(ranks_list, books_freqs)):
            plt.loglog(ranks, freqs)

        plt.xlabel('Rank')
        plt.ylabel('Frequency')
        plt.title("Zipf's Law Curve in Log-Log Scale " + title)
        plt.legend(label_list)
        plt.show()

def random_generated_corpus(size_by_len=None, size_by_word=None, max_len =
None, alphabet=None):
    if alphabet is None:
        alphabet = "abcdefghijklmnopqrstuvwxyz "
    symbols = list(alphabet)
    text = ""
    low = 0
    high = len(symbols)

    if size_by_len is not None:
        for i in range(size_by_len):
            text += symbols[np.random.randint(low, high, size=1).item()]

    elif size_by_word is not None:
        while text.count(" ") < size_by_word:
            if max_len is not None:
                if text.count(" ") > 0:
                    word_len = len(text) - text.rfind(" ") - 1
                    if word_len == max_len:
                        text += " "
            text += symbols[np.random.randint(low, high, size=1).item()]

    corpus = text.split(" ")
    return corpus

"""# **Preprocess of All Books**"""

path_authors = "books/authors/"
path_genres = "books/genres/"

authors = os.listdir(path_authors)
genres = os.listdir(path_genres)

author_tokenized = {}
genres_tokenized = {}

# dictionary of dictionaries, inner dictionary has two values first one is
# tokenized corpus second one is tokenized corpus with stop words removed
for author in authors:
    dict_books = {}
    author_books = os.listdir(path_authors + author + "/")
    for book in author_books:
        book_path = path_authors + author + "/" + book
        dict_books[book[:-4]] = tokenization(book_path,
stop_word_removal=True)

    author_tokenized[author] = dict_books

for genre in genres:
    dict_books = {}
    genre_books = os.listdir(path_genres + genre + "/")
    for book in genre_books:
        book_path = path_genres + genre + "/" + book
        dict_books[book[:-4]] = tokenization(book_path,
stop_word_removal=True)

    genres_tokenized[genre] = dict_books

composed_books_charlot, composed_books_dost, composed_books_tols =
book_compose(author_tokenized, stop_word = 0)
composed_books_charlot_rmvd, composed_books_dost_rmvd,
composed_books_tols_rmvd = book_compose(author_tokenized, stop_word = 1)
print("Length of Corpus After Preprocess:")
print(f"Composed Corpus Length of Leo Tolstoy: {len(composed_books_tols)}")
print(f"Composed Corpus Length of Charlotte Bronte:
{len(composed_books_charlot)}")
print(f"Composed Corpus Length of Fyodor Dostoyevsky:
{len(composed_books_dost)}")
print("\n")
print("Length of Corpus After Preprocess and Stop Words are Removed:")
print(f"Composed Corpus Length of Leo Tolstoy:
{len(composed_books_tols_rmvd)}")
print(f"Composed Corpus Length of Charlotte Bronte:
{len(composed_books_charlot_rmvd)}")
print(f"Composed Corpus Length of Fyodor Dostoyevsky:
{len(composed_books_dost_rmvd)}")

"""# **Part F.1**"""

tols_freq_dict = create_frequency_dict(composed_books_tols, diction = True)
charlot_freq_dict = create_frequency_dict(composed_books_charlot, diction =
True)
dost_freq_dict = create_frequency_dict(composed_books_dost, diction = True)

author_dicts = [tols_freq_dict, charlot_freq_dict, dost_freq_dict]
label_authors = ["Leo Tolstoy", "Charlotte Bronte", "Fyodor Dostoyevsky"]

zipf_plot(list_dicts=author_dicts, label_list=label_authors, scale_type =
"linear", title = "for Authors")

"""# **Part F.2**"""

anna_dict = create_frequency_dict(author_tokenized["Leo
Tolstoy"]["anna_karenina"][0], diction = True)
resur_dict = create_frequency_dict(author_tokenized["Leo
Tolstoy"]["resurrection"][0], diction = True)
war_dict = create_frequency_dict(author_tokenized["Leo
Tolstoy"]["war_and_peace"][0], diction = True)

crime_dict = create_frequency_dict(author_tokenized["Fyodor
Dostoyevsky"]["crime_and_punishment"][0], diction = True)
```

```python
brother_dict = create_frequency_dict(author_tokenized["Fyodor
Dostoyevsky"]["the_brothers_karamazov"][0], diction = True)
idiot_dict = create_frequency_dict(author_tokenized["Fyodor
Dostoyevsky"]["the_idiot"][0], diction = True)

jane_dict = create_frequency_dict(author_tokenized["Charlotte
Bronte"]["jane_eyre"][0], diction = True)
shir_dict = create_frequency_dict(author_tokenized["Charlotte
Bronte"]["shirley"][0], diction = True)
vil_dict = create_frequency_dict(author_tokenized["Charlotte
Bronte"]["villette"][0], diction = True)


leo_list = [anna_dict, resur_dict, war_dict]
leo_label_list = ["Anna Karenina", "Resurrection", "War and Peace"]

dost_list = [crime_dict, brother_dict, idiot_dict]
dost_label_list = ["Crime and Punishment", "The Brothers Karamazov", "The
Idiot"]

char_list = [jane_dict, shir_dict, vil_dict]
char_label_list = ["Jane Eyre", "Shirley", "Villette"]

zipf_plot(list_dicts=leo_list, label_list=leo_label_list, scale_type = "log-
log", title = "for Leo Tolstoy's Books")
zipf_plot(list_dicts=dost_list, label_list=dost_label_list, scale_type = "log-
log", title = "for Fyodor Dostoyevsky's Books")
zipf_plot(list_dicts=char_list, label_list=char_label_list, scale_type = "log-
log", title = "for Charlotte Bronte's Books")

"""# **Part G**"""

# get composed books from part f1
size = 10000

num_unique_words_tols = count_word_types(composed_books_tols, size = size)
num_unique_words_dost = count_word_types(composed_books_dost, size = size)
num_unique_words_charlot = count_word_types(composed_books_charlot, size =
size)

range_tols = np.arange(size, (len(num_unique_words_tols)+1)*size, size)
range_dost = np.arange(size, (len(num_unique_words_dost)+1)*size, size)
range_charlot = np.arange(size, (len(num_unique_words_charlot)+1)*size, size)

plt.figure(figsize=(18, 5))
plt.subplot(1,2,1)
plt.plot(range_tols, num_unique_words_tols)
plt.plot(range_dost, num_unique_words_dost)
plt.plot(range_charlot, num_unique_words_charlot)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Leo Tolstoy", "Charlotte Bronte", "Fyodor Dostoyevsky"])
plt.title(f"Number of Unique Words wrt Increasing Token Size by {size} in
Linear Scale")

plt.subplot(1,2,2)
plt.loglog(range_tols, num_unique_words_tols)
plt.loglog(range_dost, num_unique_words_dost)
plt.loglog(range_charlot, num_unique_words_charlot)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Leo Tolstoy", "Charlotte Bronte", "Fyodor Dostoyevsky"])
plt.title(f"Number of Unique Words wrt Increasing Token Size by {size} in Log
Scale")
plt.show()

"""# **Part H**"""

size = 5000

authors = []
books_author = []
word_freqs_author = []
x_ranges_author = []

plt.figure(figsize=(10,10))
for author in author_tokenized.keys():
    authors.append(author)
    for book in author_tokenized[author].keys():
        y = count_word_types(author_tokenized[author][book][0], size = size)
        x = range_tols = np.arange(size, (len(y)+1)*size, size)

        books_author.append(book)
        word_freqs_author.append(y)
        x_ranges_author.append(x)

        plt.loglog(x, y, label = book, marker = "o")

plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words wrt Increasing Token Size \nby {size} in
Log Scale for Each 9 Books")

"""# **Part I**"""

coefficients_author = []
```

```
for x_range, freqs in zip(x_ranges_author, word_freqs_author):
    x_log = np.log10(x_range)
    y_log = np.log10(freqs)
    coeffs = np.polyfit(x_log, y_log, 1)
    coefficients_author.append(coeffs)

plt.figure(figsize=(10,10))
for i in range(2,9,3):
    label1 = books_author[i]
    label2 = books_author[i] + " best_fit"
    plt.loglog(x_ranges_author[i], word_freqs_author[i], label = label1,
marker = "o")
    plt.loglog(x_ranges_author[i], 10**coefficients_author[i][1] *
x_ranges_author[i]**coefficients_author[i][0], label = label2)

coefficients_author = np.concatenate(coefficients_author).reshape(-1, 2)

coeff_table = pd.DataFrame(columns=["Authors" ,"Book Names", "Interception",
"Slope"])
coeff_table["Authors"] = np.repeat(list(author_tokenized.keys()), 3)
coeff_table["Book Names"] = books_author
coeff_table["Interception"] = coefficients_author[:,1]
coeff_table["Slope"] = coefficients_author[:,0]


plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words wrt Increasing Token Size by {size} in
\nLog Scale for One Books from Each Author with Their Best Fit Lines")
plt.show()
print("\n")
coeff_table

"""# **Part J**"""

size = 5000

genres = []
books_genres = []
word_freqs_genres = []
x_ranges_genres = []

plt.figure(figsize=(10,10))
for genre in genres_tokenized.keys():
    genres.append(genre)
    for book in genres_tokenized[genre].keys():
        y = count_word_types(genres_tokenized[genre][book][0], size = size)
        x = np.arange(size, (len(y)+1)*size, size)
```

```python
        books_genres.append(book)
        word_freqs_genres.append(y)
        x_ranges_genres.append(x)

        plt.loglog(x, y, label = book, marker = "o")

plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words wrt Increasing Token Size \nby {size} in
Log Scale for Each 9 Books")

coefficients_genres = []

for x_range, freqs in zip(x_ranges_genres, word_freqs_genres):
    x_log = np.log10(x_range)
    y_log = np.log10(freqs)
    coeffs = np.polyfit(x_log, y_log, 1)
    coefficients_genres.append(coeffs)

plt.figure(figsize=(10,10))
for i in range(2,9,3):
    label1 = books_genres[i]
    label2 = books_genres[i] + " best_fit"
    plt.loglog(x_ranges_genres[i], word_freqs_genres[i], label = label1,
marker = "o")
    plt.loglog(x_ranges_genres[i], 10**coefficients_genres[i][0] *
x_ranges_genres[i]**coefficients_genres[i][1], label = label2)

coefficients_genres = np.concatenate(coefficients_genres).reshape(-1, 2)

coeff_table = pd.DataFrame(columns=["Genres", "Book Names", "Interception",
"Slope"])
coeff_table["Genres"] = np.repeat(list(genres_tokenized.keys()), 3)
coeff_table["Book Names"] = books_genres
coeff_table["Interception"] = coefficients_genres[:,1]
coeff_table["Slope"] = coefficients_genres[:,0]


plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words wrt Increasing Token Size by {size} in
\nLog Scale for One Books from Each Genre with Their Best Fit Lines")
plt.show()
print("\n")
coeff_table
```

```
"""# **Part L**

**Part G when Stop Words are Removed**
"""

# get composed books from part f1
size = 10000

num_unique_words_tols_rmvd = count_word_types(composed_books_tols_rmvd, size =
size)
num_unique_words_dost_rmvd = count_word_types(composed_books_dost_rmvd, size =
size)
num_unique_words_charlot_rmvd = count_word_types(composed_books_charlot_rmvd,
size = size)

range_tols = np.arange(size, (len(num_unique_words_tols_rmvd)+1)*size, size)
range_dost = np.arange(size, (len(num_unique_words_dost_rmvd)+1)*size, size)
range_charlot = np.arange(size, (len(num_unique_words_charlot_rmvd)+1)*size,
size)

plt.figure(figsize=(18, 5))
plt.subplot(1,2,1)
plt.plot(range_tols, num_unique_words_tols_rmvd)
plt.plot(range_dost, num_unique_words_dost_rmvd)
plt.plot(range_charlot, num_unique_words_charlot_rmvd)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Leo Tolstoy", "Charlotte Bronte", "Fyodor Dostoyevsky"])
plt.title(f"Number of Unique Words (Stop Words Removed) wrt \nIncreasing Token
Size by {size} in Linear Scale")

plt.subplot(1,2,2)
plt.loglog(range_tols, num_unique_words_tols_rmvd)
plt.loglog(range_dost, num_unique_words_dost_rmvd)
plt.loglog(range_charlot, num_unique_words_charlot_rmvd)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Leo Tolstoy", "Charlotte Bronte", "Fyodor Dostoyevsky"])
plt.title(f"Number of Unique Words (Stop Words Removed) wrt \nIncreasing Token
Size by {size} in Log Scale")
plt.show()

"""**Part *H and I* when Stop Words are Removed**"""

size = 5000

authors = []
books_author = []
word_freqs_author = []
```

```python
x_ranges_author = []

plt.figure(figsize=(10,10))
for author in author_tokenized.keys():
    authors.append(author)
    for book in author_tokenized[author].keys():
        y = count_word_types(author_tokenized[author][book][1], size = size)
        x = range_tols = np.arange(size, (len(y)+1)*size, size)

        books_author.append(book)
        word_freqs_author.append(y)
        x_ranges_author.append(x)

        plt.loglog(x, y, label = book, marker = "o")

plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words (Stop Words Removed) wrt Increasing Token
Size \nby {size} in Log Scale for Each 9 Books")
plt.show()

coefficients_author = []

for x_range, freqs in zip(x_ranges_author, word_freqs_author):
    x_log = np.log10(x_range)
    y_log = np.log10(freqs)
    coeffs = np.polyfit(x_log, y_log, 1)
    coefficients_author.append(coeffs)

plt.figure(figsize=(10,10))
for i in range(2,9,3):
    label1 = books_author[i]
    label2 = books_author[i] + " best_fit"
    plt.loglog(x_ranges_author[i], word_freqs_author[i], label = label1,
marker = "o")
    plt.loglog(x_ranges_author[i], 10**coefficients_author[i][1] *
x_ranges_author[i]**coefficients_author[i][0], label = label2)

coefficients_author = np.concatenate(coefficients_author).reshape(-1, 2)

coeff_table = pd.DataFrame(columns=["Authors" ,"Book Names", "Interception",
"Slope"])
coeff_table["Authors"] = np.repeat(list(author_tokenized.keys()), 3)
coeff_table["Book Names"] = books_author
coeff_table["Interception"] = coefficients_author[:,0]
coeff_table["Slope"] = coefficients_author[:,1]
```

```
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words (Stop Words Removed) wrt Increasing Token
Size by {size} in \nLog Scale for One Books from Each Author with Their Best
Fit Lines")
plt.show()
print("\n")
coeff_table

"""**Part *J* when Stop Words are Removed**"""

size = 5000

genres = []
books_genres = []
word_freqs_genres = []
x_ranges_genres = []

plt.figure(figsize=(10,10))
for genre in genres_tokenized.keys():
    genres.append(genre)
    for book in genres_tokenized[genre].keys():
        y = count_word_types(genres_tokenized[genre][book][1], size = size)
        x = np.arange(size, (len(y)+1)*size, size)

        books_genres.append(book)
        word_freqs_genres.append(y)
        x_ranges_genres.append(x)

        plt.loglog(x, y, label = book, marker = "o")

plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words (Stop Words Removed) wrt Increasing Token
Size \nby {size} in Log Scale for Each 9 Books")

coefficients_genres = []

for x_range, freqs in zip(x_ranges_genres, word_freqs_genres):
    x_log = np.log10(x_range)
    y_log = np.log10(freqs)
    coeffs = np.polyfit(x_log, y_log, 1)
    coefficients_genres.append(coeffs)

plt.figure(figsize=(10,10))
for i in range(2,9,3):
    label1 = books_genres[i]
    label2 = books_genres[i] + " best_fit"
    plt.loglog(x_ranges_genres[i], word_freqs_genres[i], label = label1, 
marker = "o")
    plt.loglog(x_ranges_genres[i], 10**coefficients_genres[i][1] * 
x_ranges_genres[i]**coefficients_genres[i][0], label = label2)

coefficients_genres = np.concatenate(coefficients_genres).reshape(-1, 2)

coeff_table = pd.DataFrame(columns=["Book Names", "Interception", "Slope"])
coeff_table["Book Names"] = books_genres
coeff_table["Interception"] = coefficients_genres[:,0]
coeff_table["Slope"] = coefficients_genres[:,1]


plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend()
plt.title(f"Number of Unique Words  (Stop Words Removed) wrt Increasing Token 
Size by {size} in \nLog Scale for One Books from Each Genre with Their Best 
Fit Lines")
plt.show()
print("\n")
coeff_table

"""# **Part M**"""

word_size = 10000
t1 = time.time()
randomly_generated_corpus = []
for i in range(30):
    randomly_generated_corpus.extend(random_generated_corpus(size_by_word=word
_size, max_len = 6, alphabet = None))
    if i % 5 == 0:
        t2 = time.time()
        print(i , round(t2-t1, 2))
t3 = time.time()
print(f"To create {word_size*30} words took {round(t3-t1, 2)} secs")
print(len(randomly_generated_corpus))

word_size = 10000
t1 = time.time()
random_corpus = []
for i in range(40):
    random_corpus.extend(random_generated_corpus(size_by_word=word_size, 
max_len = 8, alphabet = None))
    if i % 5 == 0:
        t2 = time.time()
        print(i , round(t2-t1, 2))
t3 = time.time()
```

```
print(f"To create {word_size*40} words took {round(t3-t1, 2)} secs")
print(len(random_corpus))

random_word_types, random_word_freqs =
create_frequency_dict(randomly_generated_corpus, diction = False)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.plot(random_word_freqs, label = "Random Corpus")
plt.xlabel('Rank')
plt.ylabel('Frequency')
plt.title("Zipf's Law Curve in Linear Scale \nfor Randomly Generated Corpus")
plt.legend()

plt.subplot(122)
plt.loglog(random_word_freqs, label = "Random Corpus")
plt.xlabel('Rank')
plt.ylabel('Frequency')
plt.title("Zipf's Law Curve in Logarithmic Scale \nfor Randomly Generated
Corpus")
plt.legend()
plt.show()

size = 10000

num_unique_words_random = count_word_types(randomly_generated_corpus, size =
size)
range_random = np.arange(size, (len(num_unique_words_random)+1)*size, size)

plt.figure(figsize=(12,5))
plt.subplot(121)
plt.plot(range_random, num_unique_words_random)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Random Corpus"])
plt.title(f"Number of Unique Words wrt Increasing Token Size \nby {size} in
Linear Scale for Random Corpus")

plt.subplot(122)
plt.loglog(range_random, num_unique_words_random)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Random Corpus"])
plt.title(f"Number of Unique Words wrt Increasing Token Size \nby {size} in
Logorithmic Scale for Random Corpus")
plt.show()

word_size = 10000
t1 = time.time()
```

```python
random_corpus_short = []
for i in range(30):
    random_corpus_short.extend(random_generated_corpus(size_by_word=word_size,
max_len = 3, alphabet = None))
    if i % 5 == 0:
        t2 = time.time()
        print(i , round(t2-t1, 2))
t3 = time.time()
print(f"To create {word_size*30} words took {round(t3-t1, 2)} secs")
print(len(random_corpus_short))

random_word_types, random_word_freqs =
create_frequency_dict(random_corpus_short, diction = False)

plt.figure(figsize=(12,14))
plt.subplot(221)
plt.plot(random_word_freqs, label = "Random Corpus")
plt.xlabel('Rank')
plt.ylabel('Frequency')
plt.title("Zipf's Law Curve in Linear Scale \nfor Randomly Generated Corpus")
plt.legend()

plt.subplot(222)
plt.loglog(random_word_freqs, label = "Random Corpus")
plt.xlabel('Rank')
plt.ylabel('Frequency')
plt.title("Zipf's Law Curve in Logarithmic Scale \nfor Randomly Generated
Corpus")
plt.legend()


size = 10000

num_unique_words_random = count_word_types(random_corpus_short, size = size)
range_random = np.arange(size, (len(num_unique_words_random)+1)*size, size)


plt.subplot(223)
plt.plot(range_random, num_unique_words_random)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Random Corpus"])
plt.title(f"Number of Unique Words wrt Increasing Token Size \nby {size} in
Linear Scale for Random Corpus")

plt.subplot(224)
plt.loglog(range_random, num_unique_words_random)
plt.xlabel("Token Size")
plt.ylabel("Number of Unique Words")
plt.legend(["Random Corpus"])
plt.title(f"Number of Unique Words wrt Increasing Token Size \nby {size} in
Logorithmic Scale for Random Corpus")
plt.show()
```