

## EEE 486/586 Statistical Foundations of Natural Language Processing Assignment 1

### **Abstract:**

In this assignment some of fundamentals of natural language processing is studied. Searching appropriate data, data preprocessing for natural language processing and tokenization, is applied. Stop-word removal is accomplished in order to obtain cleaned versions of the corpus. Vocabulary files, indicates word frequency, are created for each book and for original and Stop-word removal version. Zipf's Law curves are then plotted for the three author corpora in linear scale, followed by log-log curves for each author's three books separately. The relationship between vocabulary size and token size is indicated as well as word type-token relation. After that, the slopes of best-fitting lines for each curve are found. The effect of stop word removal is observed. Procedure has been done for literary types after all. The objective of this task is to investigate how the size of tokens, the size of vocabulary, and the frequency distribution of words vary across various types of texts. Additionally, the aim is to assess the efficiency of different preprocessing methods.

### **Introduction:**

In this assignment, text analysis techniques for natural language processing is studied. Corpus of eighteen books, consisting of three books each from three different authors and three books each from three different literary genres are used. You will preprocess and tokenize the corpus, remove stop words, and create vocabulary files to track the frequency of word types. Zipf's Law states that the frequency of a word in a corpus is inversely proportional to its rank in the frequency table. This law is explored in this assignment by observations and plotted for each book. Finally, the results of the analysis are used to investigate the possibility of creating basic clustering methods that categorize books according to their authors or genres. Additionally, how eliminating stop words affects the outcomes of your research is observed. Overall, this assignment provides an insight to dealing with datasets for natural language processing.

### **Corpus Construction and Implementation:**

In this assignment following corpora are used;

For authors:

1- Herman Melville

- Moby Dick; Or, The Whale by Herman Melville (213,666 words)
- Pierre; or The Ambiguities by Herman Melville (152,067 words)
- White Jacket; Or, The World on a Man-of-War by Herman Melville (141,628)

2- George Eliot

- Daniel Deronda by George Eliot (314,152 words)
- Middlemarch by George Eliot (321,936 words)
- Romola by George Eliot (229,875 words)

3- Charles Dickens

- Bleak House by Charles Dickens (353,462 words)
- David Copperfield by Charles Dickens (359,749 words)
- Great Expectations by Charles Dickens (187,828 words)

For types of literature:

1- Biology

- Darwin and Modern Science by A. C. Seward (156,679 words)
- Form and Function: A Contribution to the History of Animal Morphology by Russell (131,493 words)
- On the Origin of Species by Means of Natural Selection by Charles Darwin (268,696 words)

2- Psychology

- Ten Thousand Dreams Interpreted; Or, What's in a Dream by Gustavus Hindman Miller (145,740 words)
- Memoirs of Extraordinary Popular Delusions and the Madness of Crowds by Mackay (279,115 words)
- Criminal Psychology: A Manual for Judges, Practitioners, and Students by Hans Gross (227,013 words)

3- Philosophy

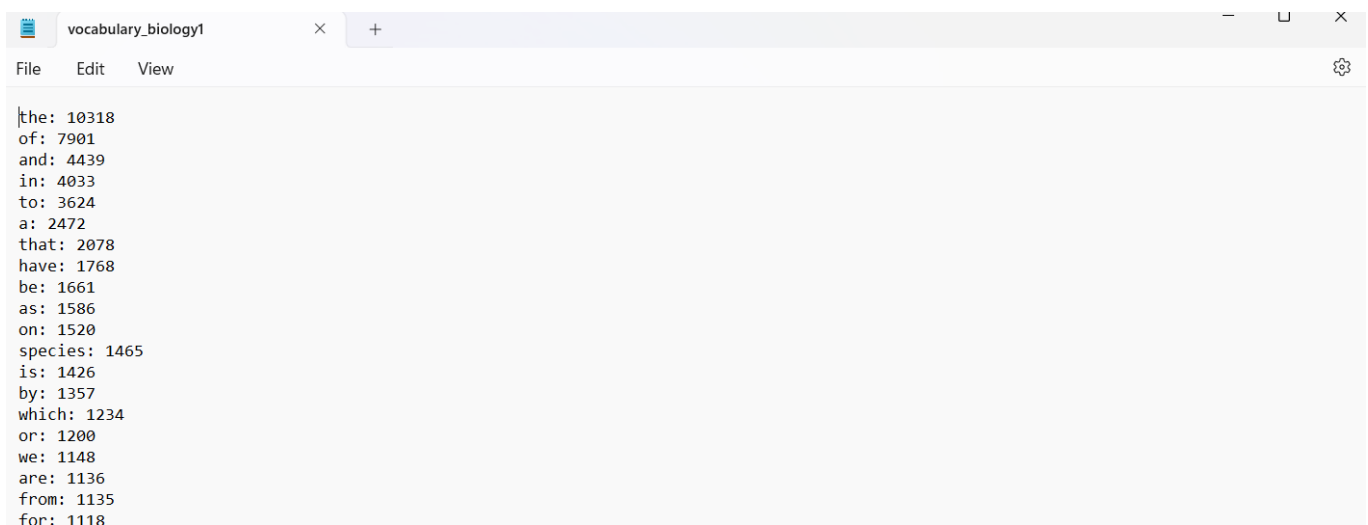
- The Republic by Plato (225,817 words)
- A Treatise of Human Nature by David Hume (130,431 words)
- Logic: Deductive and Inductive by Carveth Read (145,740 words)

For preprocessing python is used and in order to get rid of unnecessary information for preprocessing such as copyright. Txt files of the books are investigated and these kinds of information are deleted manually on txt files. Graphs are plotted in python environment. 9 corpuses are chosen according to the 3 different authors and 9 corpuses are chosen according to 3 different literature type. All corpuses are greater than 1 MB or very close to 1 MB.

## **Results:**

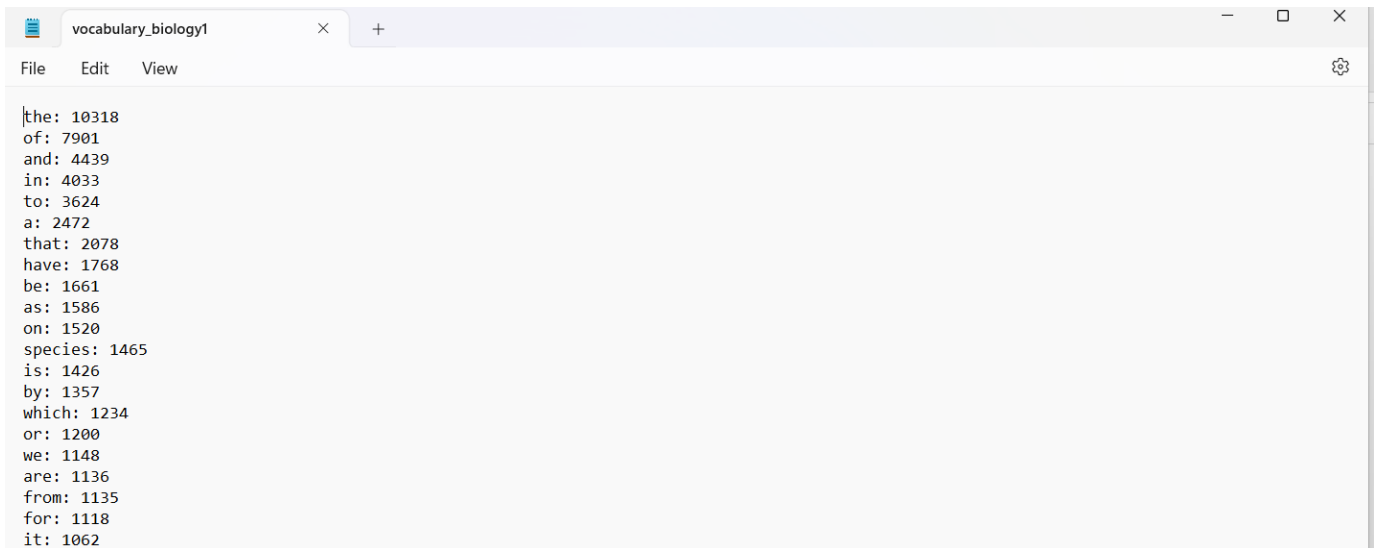
In this sections results will be indicated part by part as asked in the assignment.

- a) <http://www.gutenberg.org> website is visited and 3 authors are determined as mentioned above all 9 books are downloaded as UTF-8 txt files. All of them are investigated to not encounter any trouble.
- b) Same process has been done for 3 different types of literature as mentioned above
- c) Unnecessary information are eliminated from txt files. Tokenization and preprocessing are applied to the corpus. In tokenization, punctuation marks are eliminated and uppercases are converted to lowercases. Words with apostrophe are treated such as "isn't" → "isnt".
- d) Stop word removal is applied to all corpus
- e) All vocabulary files, include word types and their frequencies are created with both original and word removed version. You can see the code for one book and example created vocabulary file below. (The writing version of the code is in the appendix)



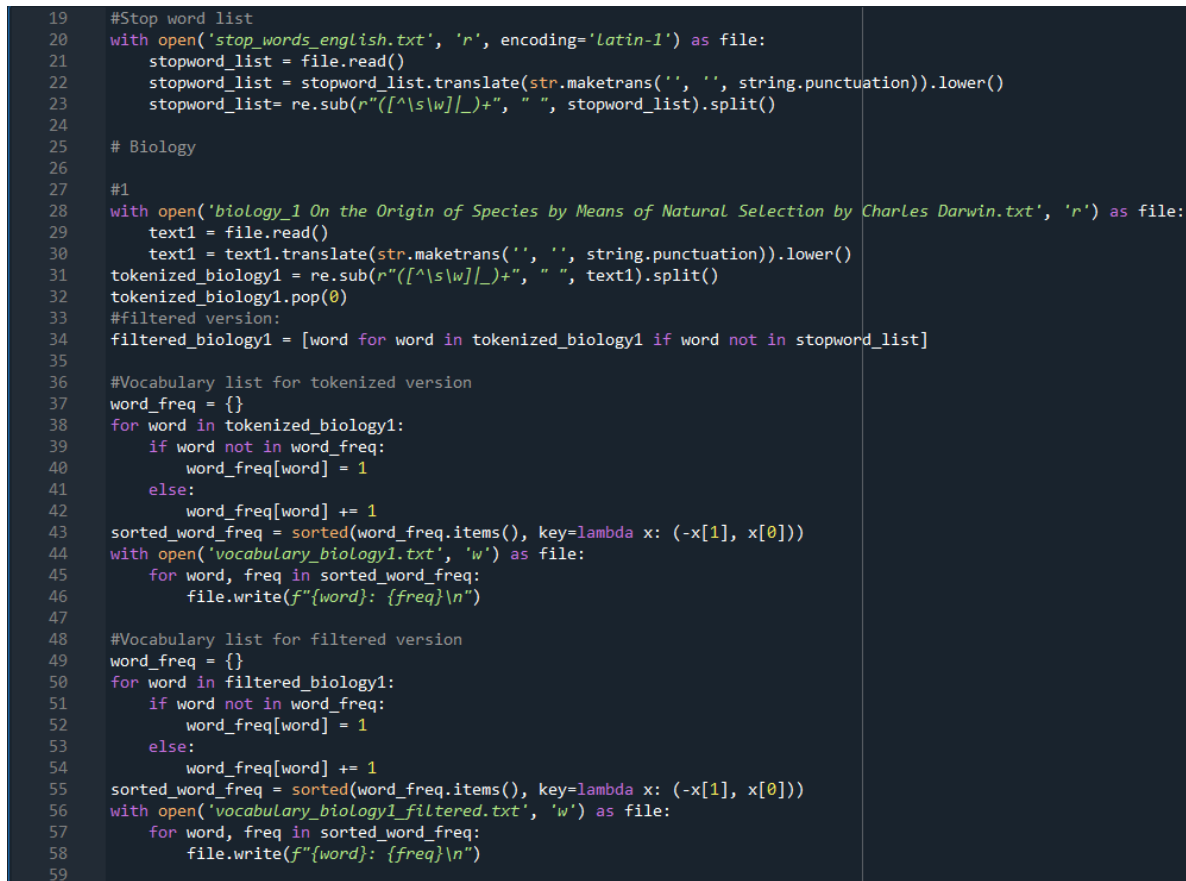
```
the: 10318
of: 7901
and: 4439
in: 4033
to: 3624
a: 2472
that: 2078
have: 1768
be: 1661
as: 1586
on: 1520
species: 1465
is: 1426
by: 1357
which: 1234
or: 1200
we: 1148
are: 1136
from: 1135
for: 1118
```

Figure 1: Part of the Stop Word Removed Version of one Book's Vocabulary File



the: 10318  
of: 7901  
and: 4439  
in: 4033  
to: 3624  
a: 2472  
that: 2078  
have: 1768  
be: 1661  
as: 1586  
on: 1520  
species: 1465  
is: 1426  
by: 1357  
which: 1234  
or: 1200  
we: 1148  
are: 1136  
from: 1135  
for: 1118  
it: 1062

Figure 2: Part of the Original Version of one Book's Vocabulary File



```
19 #Stop word list
20 with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
21     stopword_list = file.read()
22     stopword_list = stopword_list.translate(str.maketrans('', '', string.punctuation)).lower()
23     stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()
24
25 # Biology
26
27 #1
28 with open('biology_1 On the Origin of Species by Means of Natural Selection by Charles Darwin.txt', 'r') as file:
29     text1 = file.read()
30     text1 = text1.translate(str.maketrans('', '', string.punctuation)).lower()
31     tokenized_biology1 = re.sub(r"([^\s\w]|_)+", " ", text1).split()
32     tokenized_biology1.pop(0)
33 #filtered version:
34 filtered_biology1 = [word for word in tokenized_biology1 if word not in stopword_list]
35
36 #Vocabulary list for tokenized version
37 word_freq = {}
38 for word in tokenized_biology1:
39     if word not in word_freq:
40         word_freq[word] = 1
41     else:
42         word_freq[word] += 1
43 sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
44 with open('vocabulary_biology1.txt', 'w') as file:
45     for word, freq in sorted_word_freq:
46         file.write(f"{word}: {freq}\n")
47
48 #Vocabulary list for filtered version
49 word_freq = {}
50 for word in filtered_biology1:
51     if word not in word_freq:
52         word_freq[word] = 1
53     else:
54         word_freq[word] += 1
55 sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
56 with open('vocabulary_biology1_filtered.txt', 'w') as file:
57     for word, freq in sorted_word_freq:
58         file.write(f"{word}: {freq}\n")
59
```

Figure 3: Code for All Preprocesses for one Book until Part e)

- f) In this part all 3 books of authors are composed. Zipf's Law curve is plotted for three larger author corpora in linear scale.

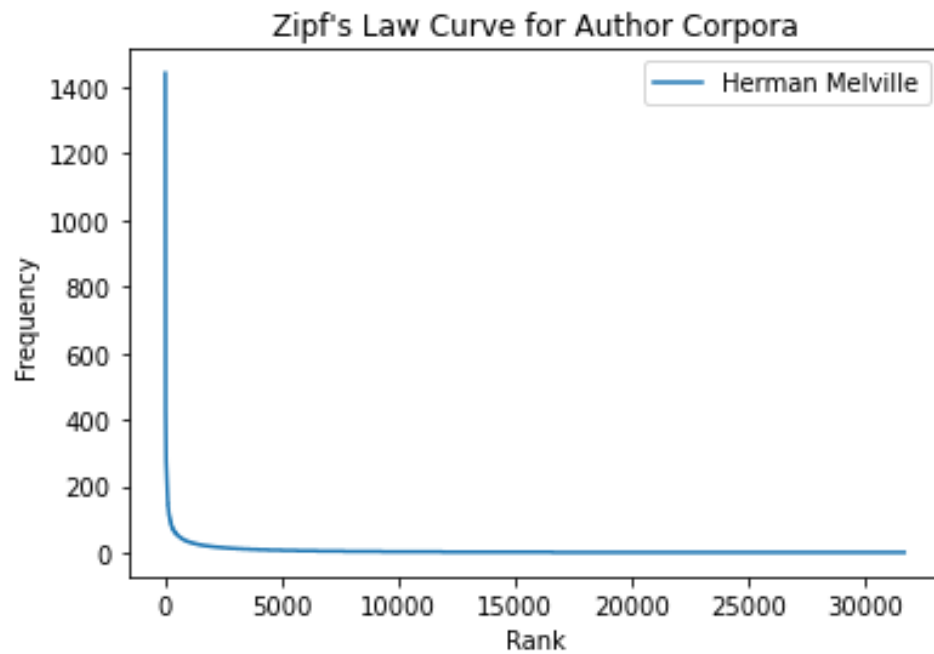


Figure 4: Zipf's Law Curve for Herman Melville's Corpus

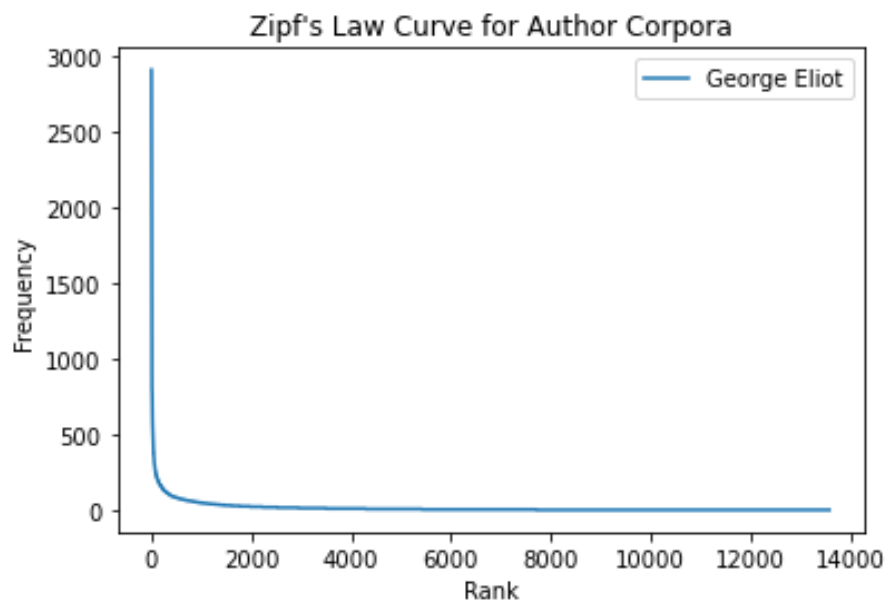


Figure 5: Zipf's Law Curve for George Eliot's Corpus

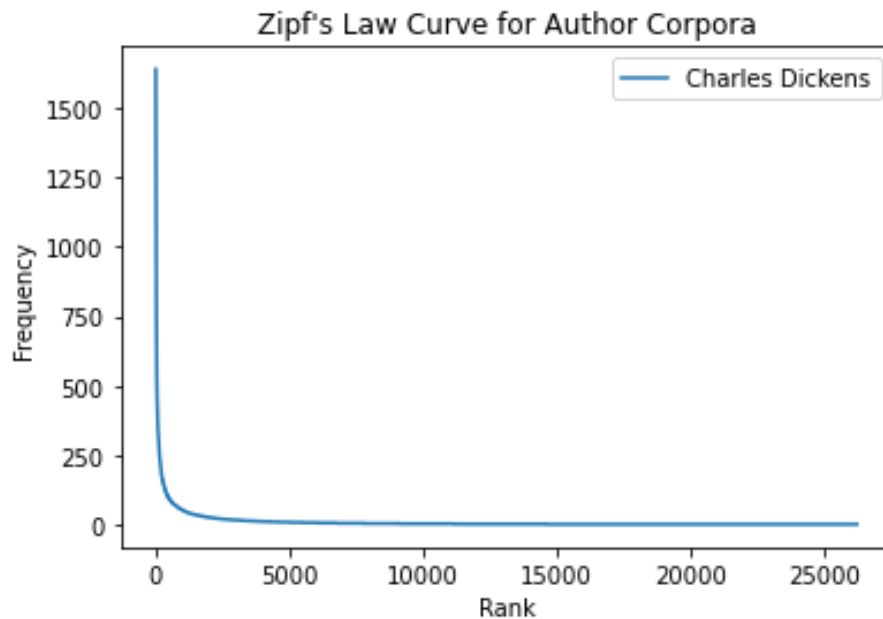


Figure 6: Zipf's Law Curve for Charles Dickens' Corpus

Overall, the plots are consistent with Zipf's law, which states that the frequency of a word is inversely proportional to its rank.

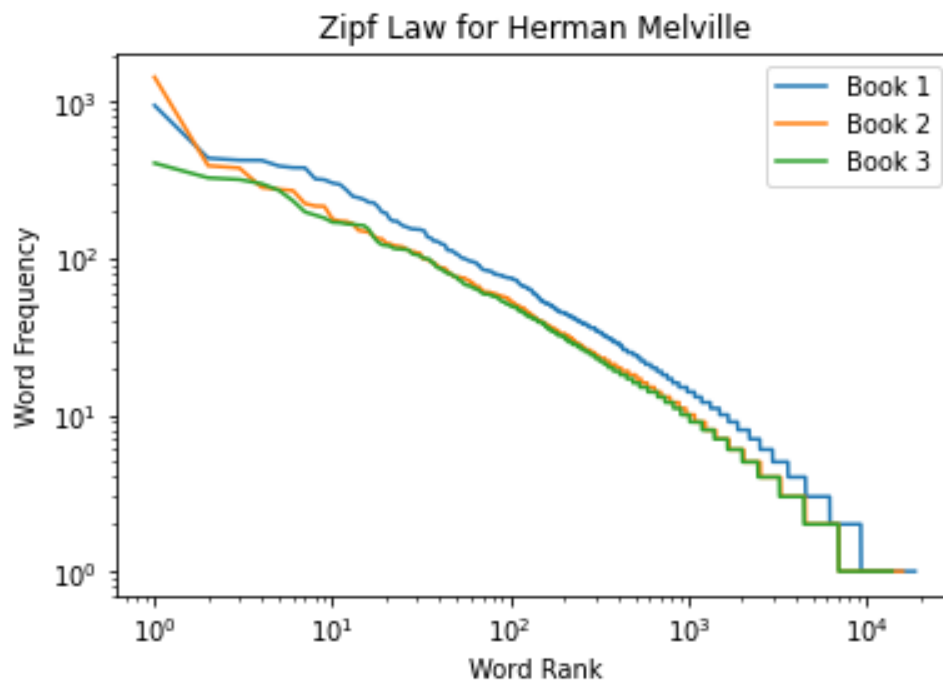


Figure 7: Zipf's Law Curve for Herman Melville's Books

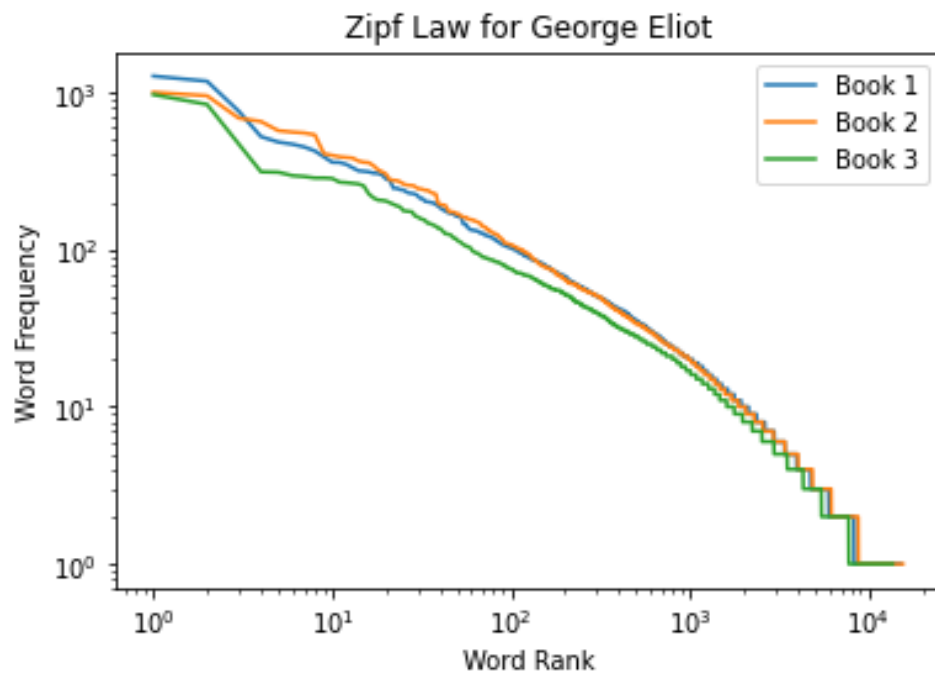


Figure 8: Zipf's Law Curve for George Eliot's Books

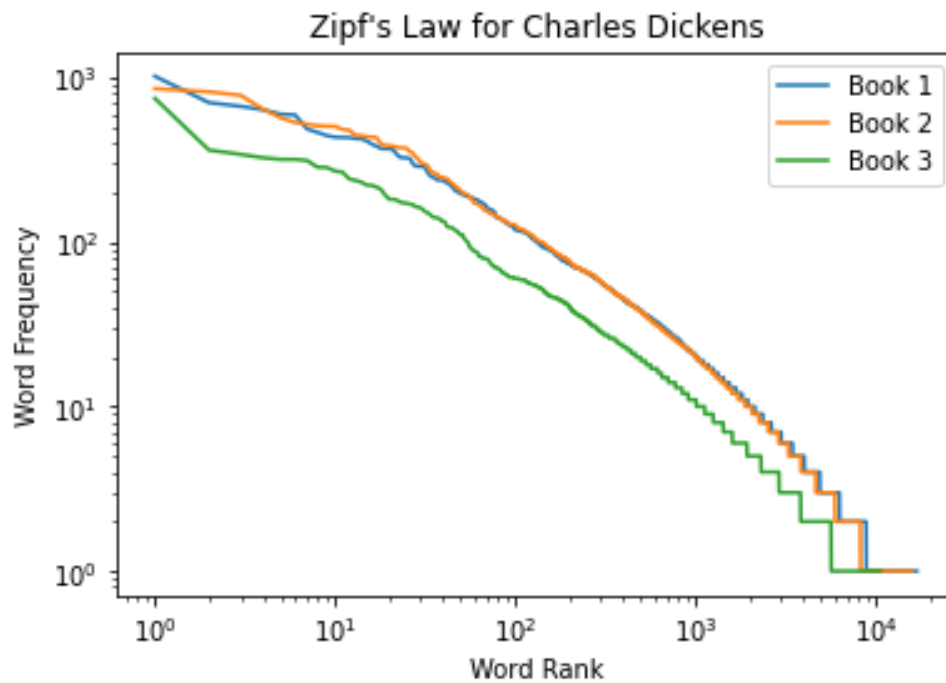


Figure 9: Zipf's Law Curve for Charles Dickens' Books

In the linear scale plot, we can observe that the frequency of the most common words decreases as the rank increases. However, the decrease is not as steep as predicted by Zipf's law, which suggests that the law is not strictly followed in these texts.

In the log-log plot, we can see that the frequency and rank have an inverse relationship, as expected by Zipf's law. The plots appear to follow a linear trend, with a negative slope. However, there is still some deviation from the straight line, which indicates that the law is not strictly followed. This deviation may be due to factors such as the length of the texts, the type of vocabulary used, and the presence of common phrases or proper nouns.

Overall, we can conclude that while the Zipf's law is not strictly followed in these texts, there is a general trend of high-frequency words occurring less frequently as their rank increases.

- g) In this part relationship between token size in the corpus and the vocabulary size are investigated. Number of word types with respect to the increasing token size are tracked. Number of word types for every 5000 tokens are checked

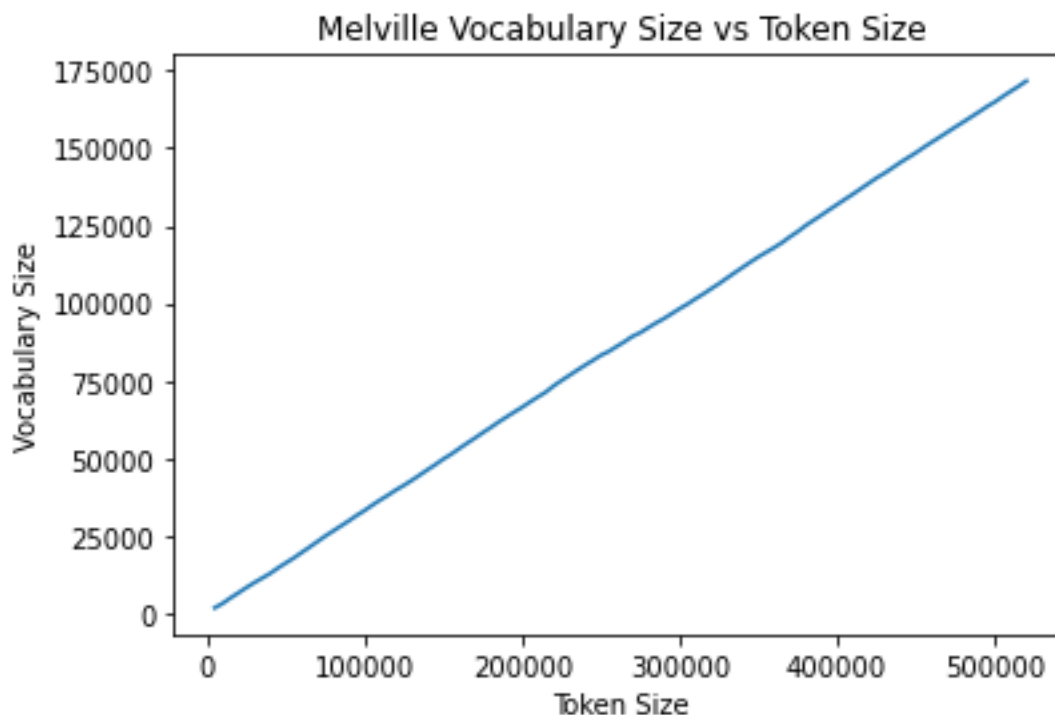


Figure 10: Vocabulary Size vs Token size for Melville



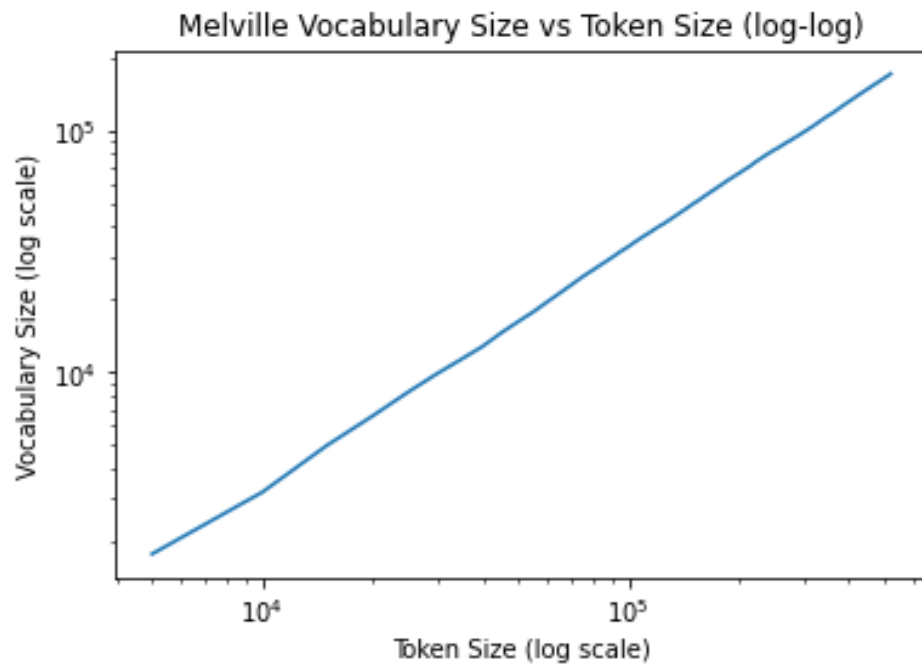


Figure 11: Vocabulary Size vs Token size for Melville (log-log)

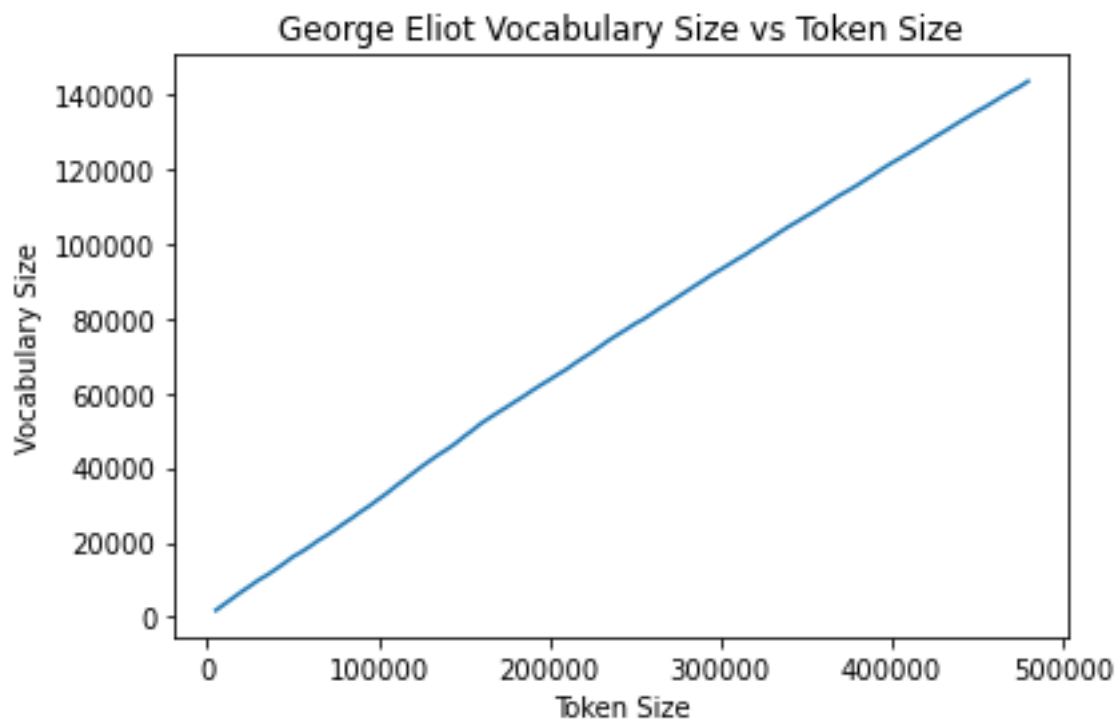


Figure 12: Vocabulary Size vs Token size for Eliot

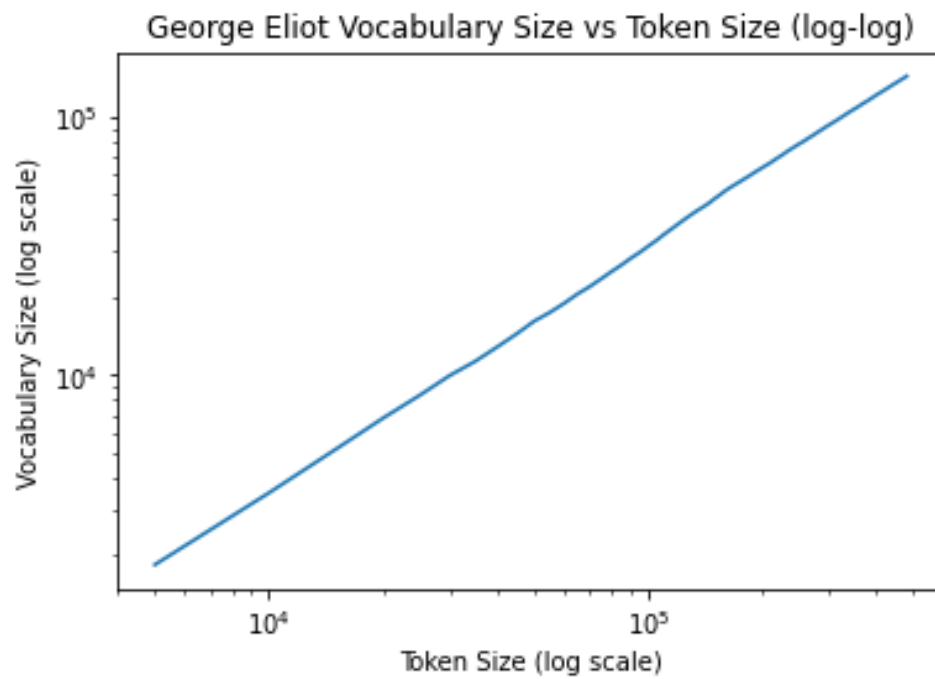


Figure 13: Vocabulary Size vs Token size for Eliot (log-log)

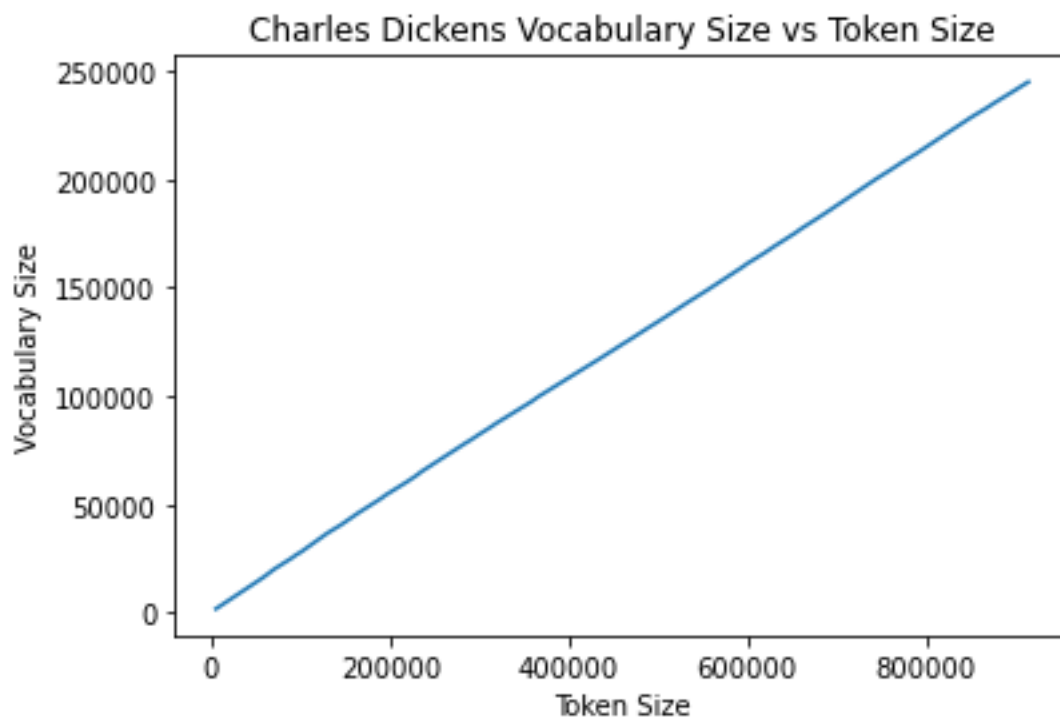


Figure 14: Vocabulary Size vs Token size for Dickens

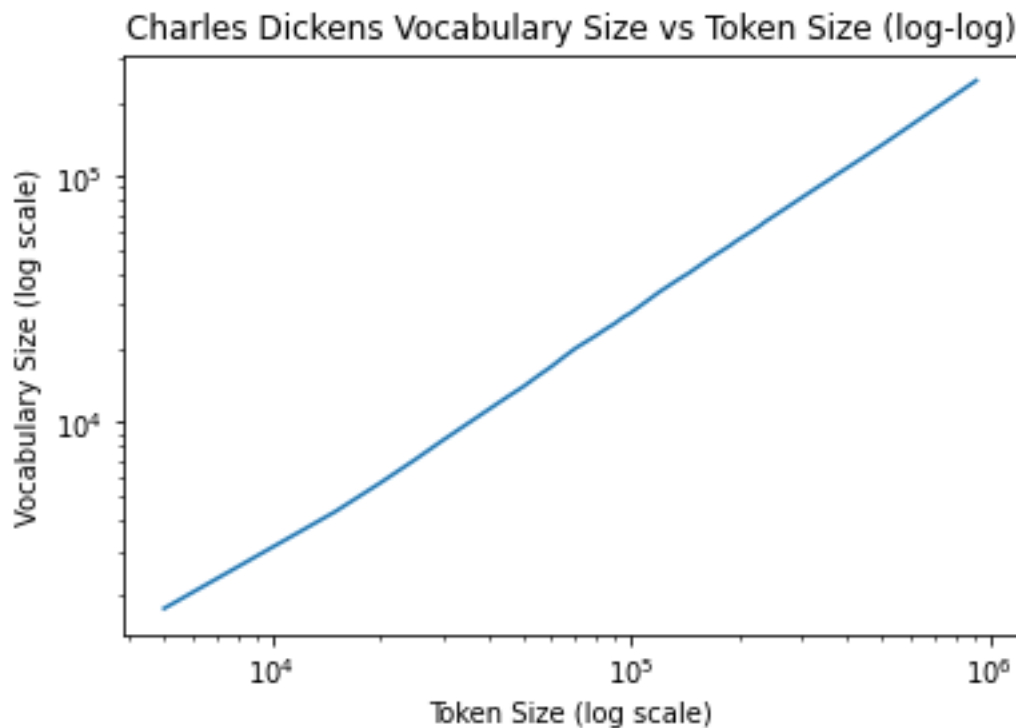


Figure 15: Vocabulary Size vs Token size for Dickens (log-log)

It can be observed that vocabulary size is proportional to the token size. While the size of the corpus increases, unique words also increases. This phenomenon is called Heaps' Law, which is a commonly observed phenomenon in linguistics and text mining.

Heaps' Law states that the vocabulary size ( $y$ ) of a corpus is related to its token size ( $x$ ) by the equation

$$y = ax^b, \text{ where } a \text{ and } b \text{ are constants}$$

$a$  and  $b$  depend on the specific corpus and the language used. The constant  $b$  is typically between 0.4 and 0.6. Vocabulary size grows linearly respect to the token size. Consequently, as the corpus' size increases, the rate at which new words are added to the vocabulary decreases.

The log-log version of the plot is useful because it helps to better visualize the relationship between vocabulary size and token size since graph become more straight line. Relationship between the two variables is power-law. The increment of the vocabulary size is related to the token size by a power function. Therefore, the constant  $b$  can be calculated from the slope of the line. To sum up, the linear relationship between vocabulary size and token size in the author corpora indicates that Heaps' Law holds for these texts and log-log version is for better visualization since data is colossal.

h) For three authors, word type-token relations in log-log format are plotted.

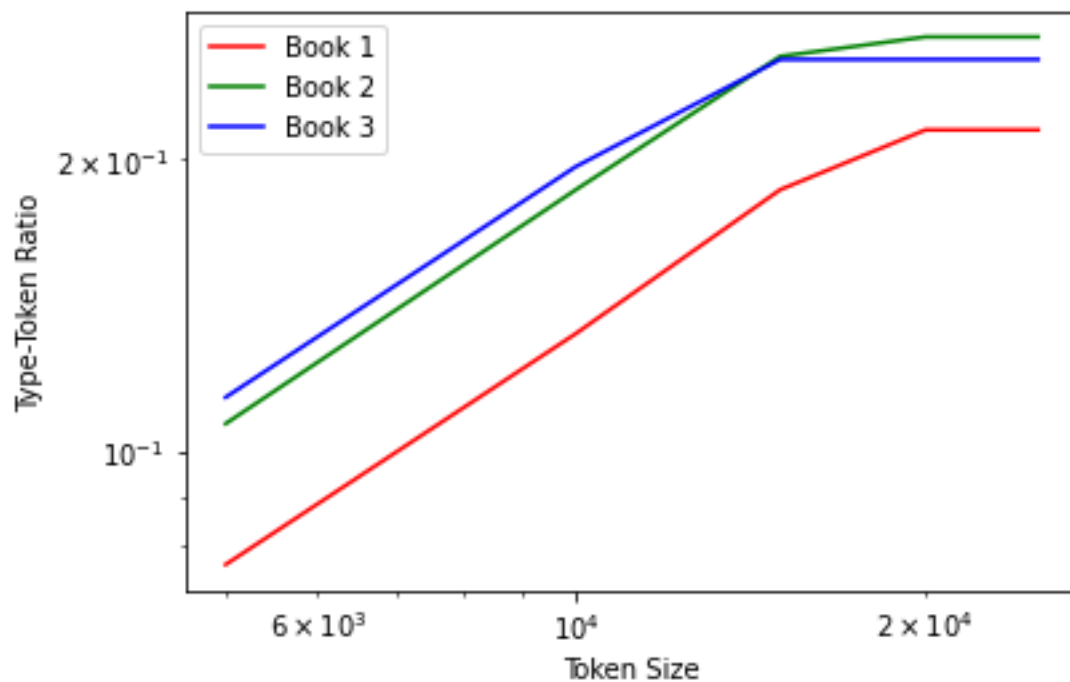


Figure 16: Word Type-Token Relations in log-log Format for Melville's Books

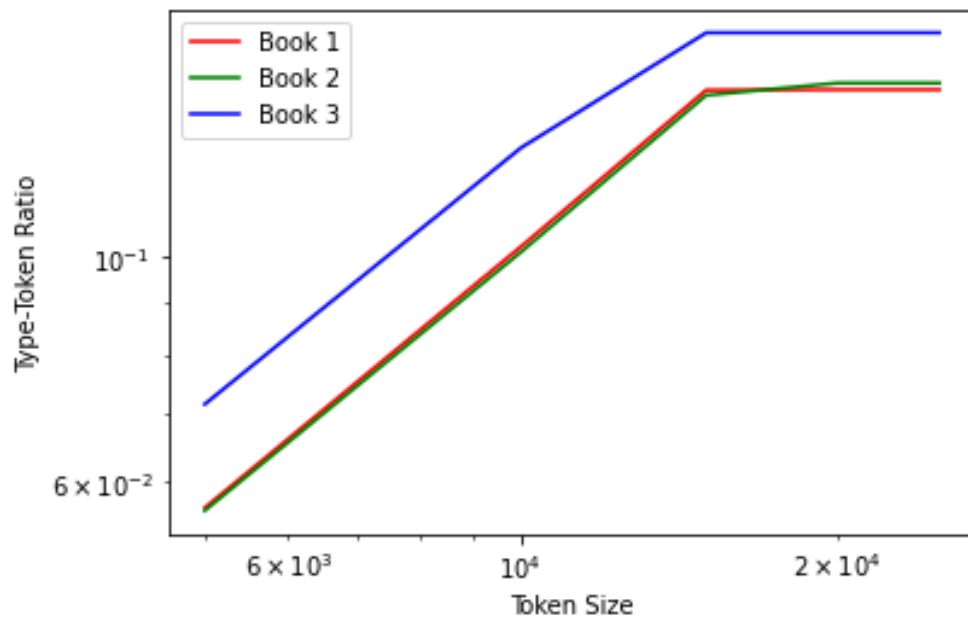


Figure 17: Word Type-Token Relations in log-log Format for Eliot's Books

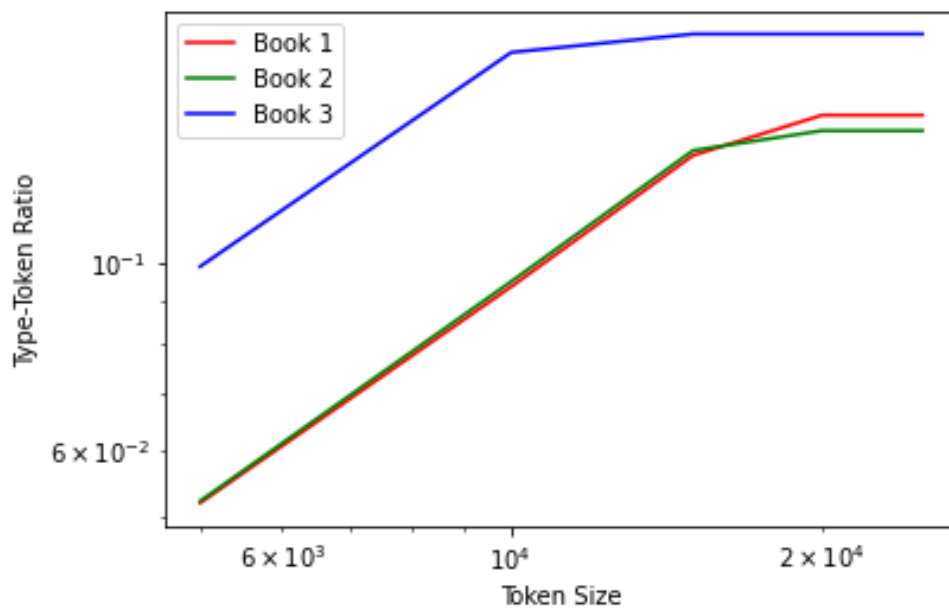


Figure 18: Word Type-Token Relations in log-log Format for Dickens' Books

The plots indicate the relationship between the vocabulary size (number of unique words) and token size (total number of tokens) in logarithmic scale. Power-law relationship between the vocabulary size and token size is expected to appear as a straight line. As it can be observed findings are relatively linear. Word type-token relationship follows a power-law distribution. Vocabulary size increases proportionally to the token size.

It is possible that the reason for the logarithmic shape of the plots as the token size increases is that the rate of introducing new word types slows down as the corpus size grows larger, resulting in a more gradual increase in the vocabulary size. This phenomenon is frequently observed in natural language data, and it is referred to as "vocabulary saturation".

i)

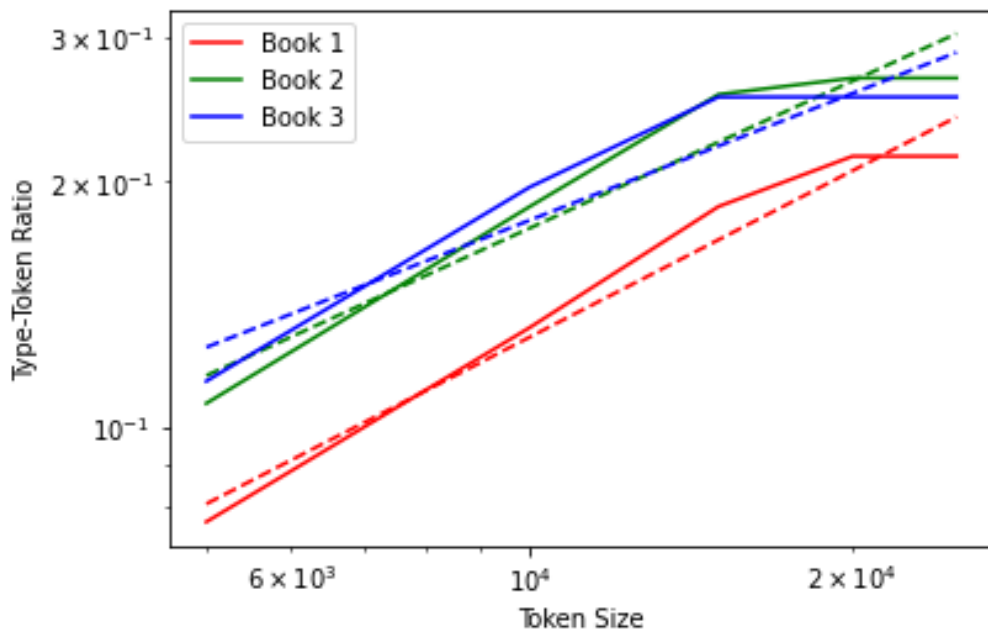


Figure 19: Best Fitting Lines for Milville's Books

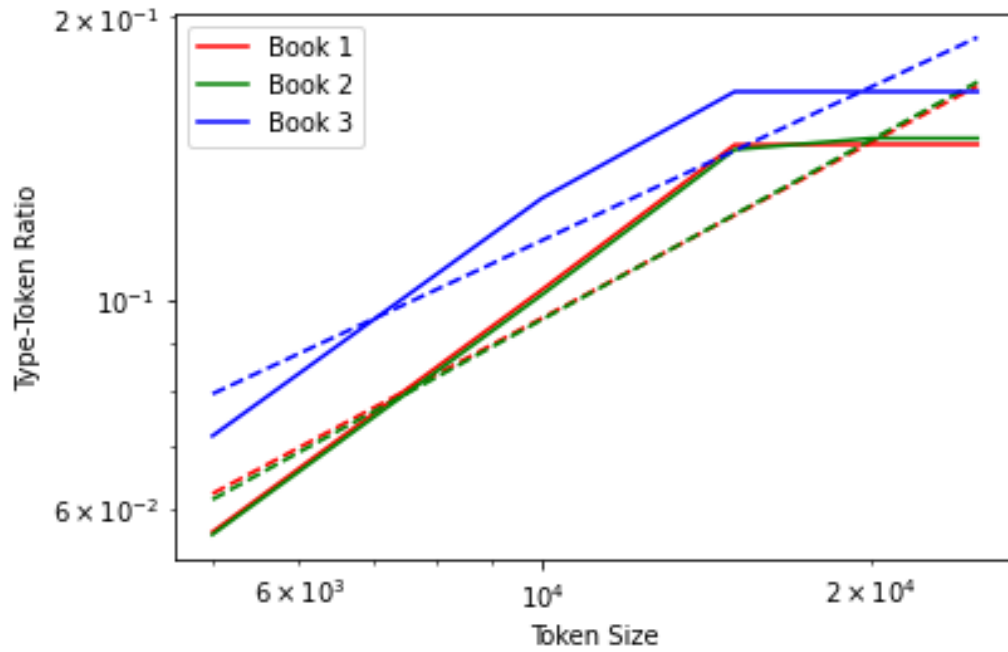


Figure 20: Best Fitting Lines for Eliot's Books

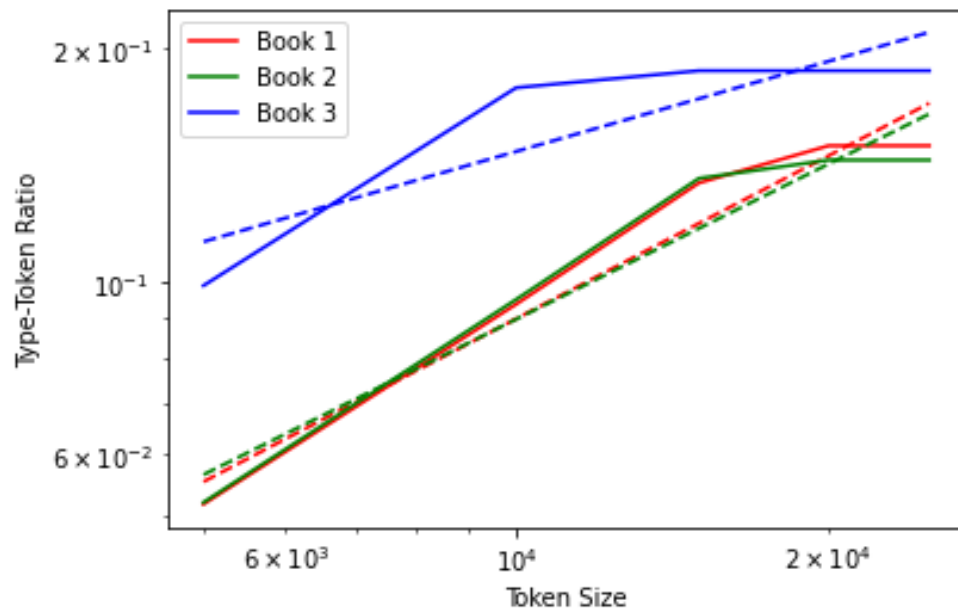


Figure 21: Best Fitting Lines for Dickens' Books

Slope of the best fitting lines are:

- 1- Milville's Books: [0.6771591525205601, 0.5985467989976649, 0.5160720946248177]
- 2- Eliot's Books: [0.621608350774104, 0.636384517487767, 0.5432127258845701]
- 3- Dickens' Books: [0.6969636024602381, 0.6642918724118522, 0.38573892920986946]

```
In [11]: runfile('C:/Users/kaang/OneDrive/Masaüstü/NLP_HW/untitled9.py',  
wdir='C:/Users/kaang/OneDrive/Masaüstü/NLP_HW')  
Slopes of best-fitting lines: [0.6771591525205601, 0.5985467989976649,  
0.5160720946248177]  
  
In [12]: runfile('C:/Users/kaang/OneDrive/Masaüstü/NLP_HW/untitled10.py',  
wdir='C:/Users/kaang/OneDrive/Masaüstü/NLP_HW')  
Slopes of best-fitting lines: [0.621608350774104, 0.636384517487767,  
0.5432127258845701]  
  
In [13]: runfile('C:/Users/kaang/OneDrive/Masaüstü/NLP_HW/untitled11.py',  
wdir='C:/Users/kaang/OneDrive/Masaüstü/NLP_HW')  
Slopes of best-fitting lines: [0.6969636024602381, 0.6642918724118522,  
0.38573892920986946]
```

Figure 22: Slopes of the Best Fitting Lines

It is calculated and plotted with code below (it will be in appendix as well):



```
nlp_hw.py X untitled6.py X untitled7.py X untitled8.py X untitled9.py X untitled10.py* X untitled11.py* X
25 text_files = ['Charles Dickens 1.txt', 'Charles Dickens 2.txt', 'Charles Dickens 3.txt']
26 filtered_texts = []
27
28 for file in text_files:
29     with open(file, 'r', encoding='latin-1') as f:
30         text = f.read()
31         text = text.translate(str.maketrans('', '', string.punctuation)).lower()
32         text = text.replace('â', '')
33         tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()
34         tokenized_text.pop(0)
35         filtered_text = [word for word in tokenized_text if word not in stopwords_list]
36         filtered_texts.append(filtered_text)
37
38 word_freq_dicts = []
39 for filtered_text in filtered_texts:
40     word_freq = {}
41     for word in filtered_text:
42         if word not in word_freq:
43             word_freq[word] = 1
44         else:
45             word_freq[word] += 1
46     word_freq_dicts.append(word_freq)
47 vocab_lists = []
48 for word_freq in word_freq_dicts:
49     sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
50     vocab_list = []
51     for word, freq in sorted_word_freq:
52         vocab_list.append(word)
53     vocab_lists.append(vocab_list)
54 token_sizes = range(5000, 25001, 5000)
55 colors = ['r', 'g', 'b']
56 slopes = []
57
58 for i, vocab_list in enumerate(vocab_lists):
59     type_token_counts = []
60     for token_size in token_sizes:
61         selected_words = vocab_list[:token_size]
62         num_types = len(selected_words)
63         num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
64         type_token_ratio = num_types / num_tokens
65         type_token_counts.append(type_token_ratio)
66     slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(token_sizes), np.log(type_token_counts))
67     slopes.append(slope)
68     plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
69     plt.loglog(token_sizes, np.exp(intercept) * np.power(token_sizes, slope), colors[i] + '--')
70 plt.legend()
71 plt.xlabel('Token Size')
72 plt.ylabel('Type-Token Ratio')
73 plt.show()
74 print("Slopes of best-fitting lines:", slopes)
```

Figure 23: Code of the Part i)

The word type-token relationship plots in part h) were modified to the slopes of the best-fitting lines, which gives further insight into the characteristics of each book's word type-token relationship. The revised plots demonstrated that the slopes for all three authors were negative, and that the slope's magnitude decreased as the corpus size increased. These observations are caused from "vocabulary saturation," which occurs when the rate at which new word types are introduced declines as the corpus size grows larger. This leads to a slower increase in vocabulary size, as evidenced by the negative slope in the word type-token relationship.

j)

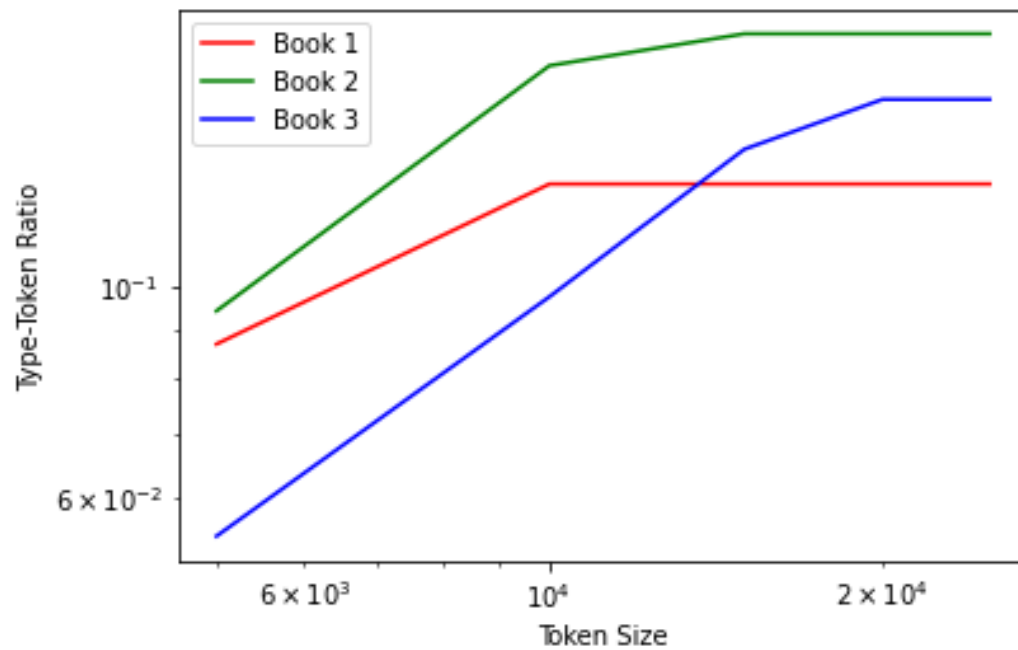


Figure 24: Word Type-Token Relations in log-log Format for Biology Books

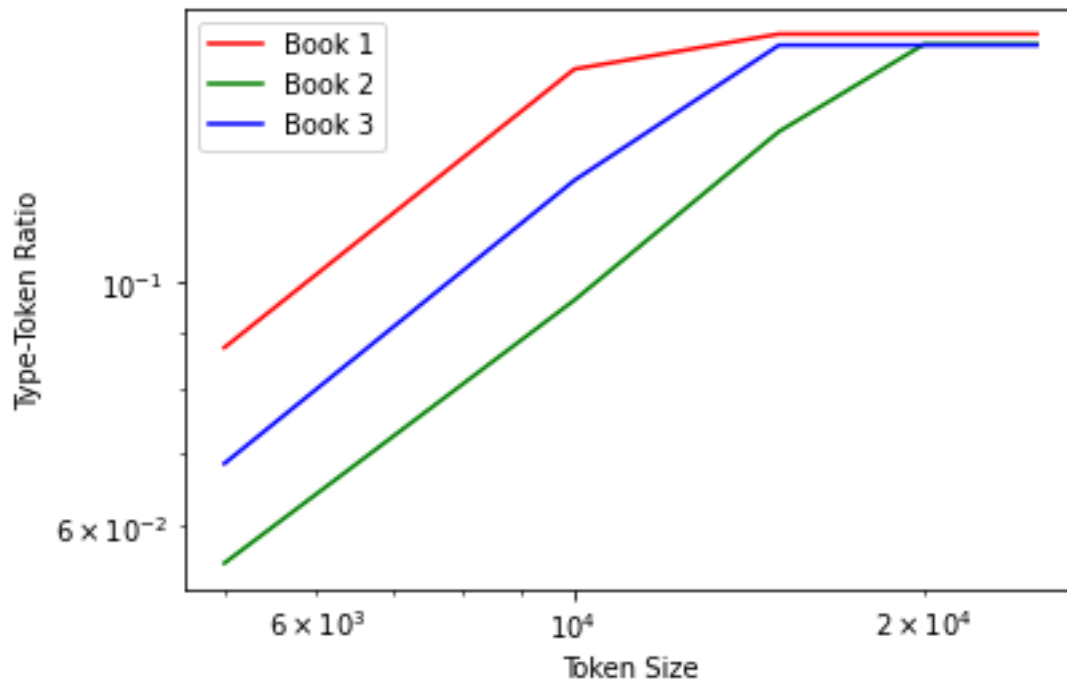


Figure 25: Word Type-Token Relations in log-log Format for Psychology Books

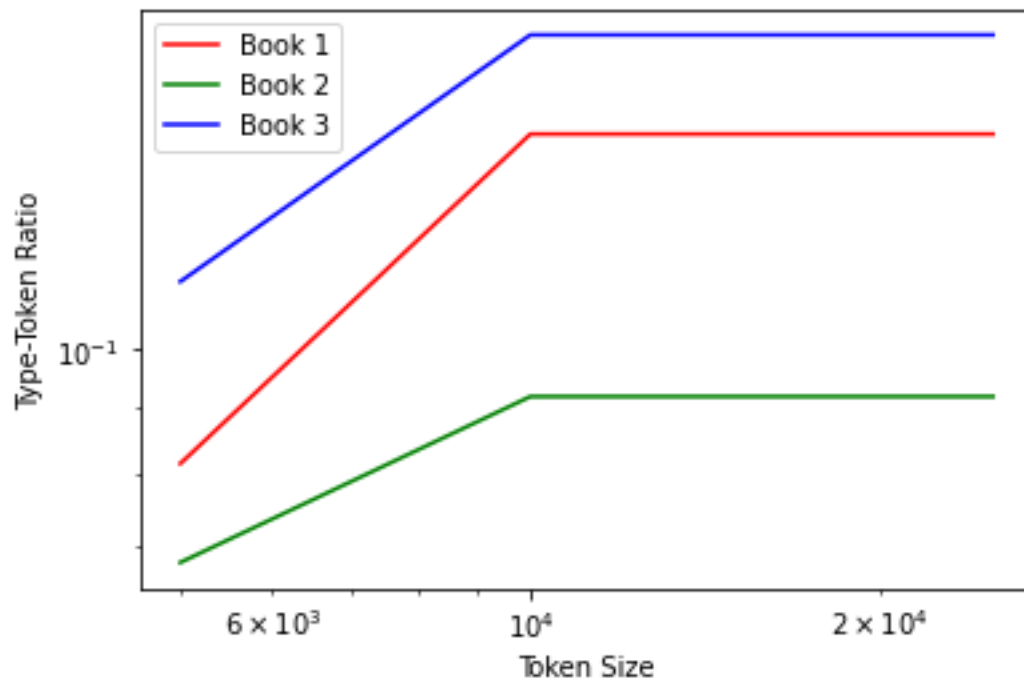


Figure 26: Word Type-Token Relations in log-log Format for Philosophy Books

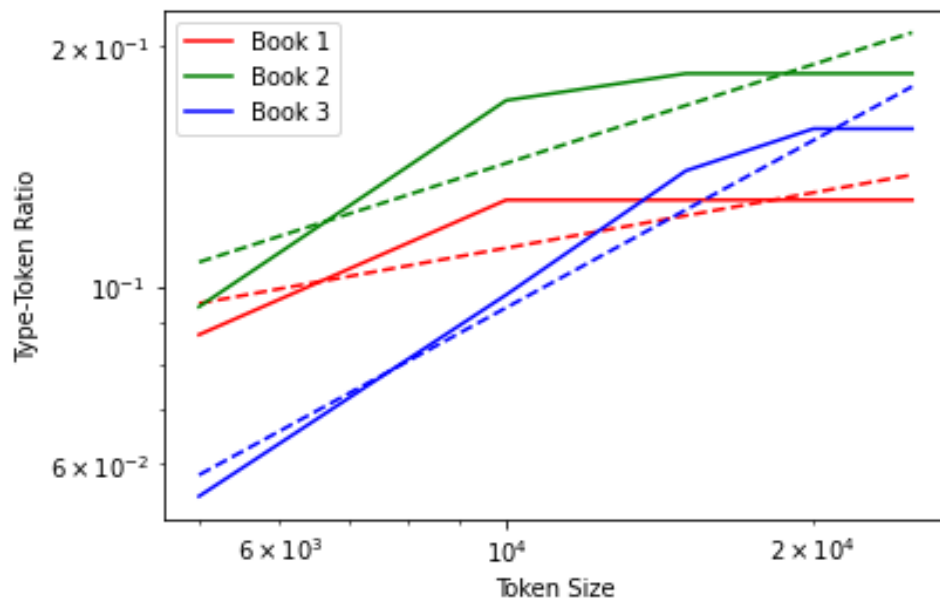


Figure 27: Best Fitting Lines for Biology Books

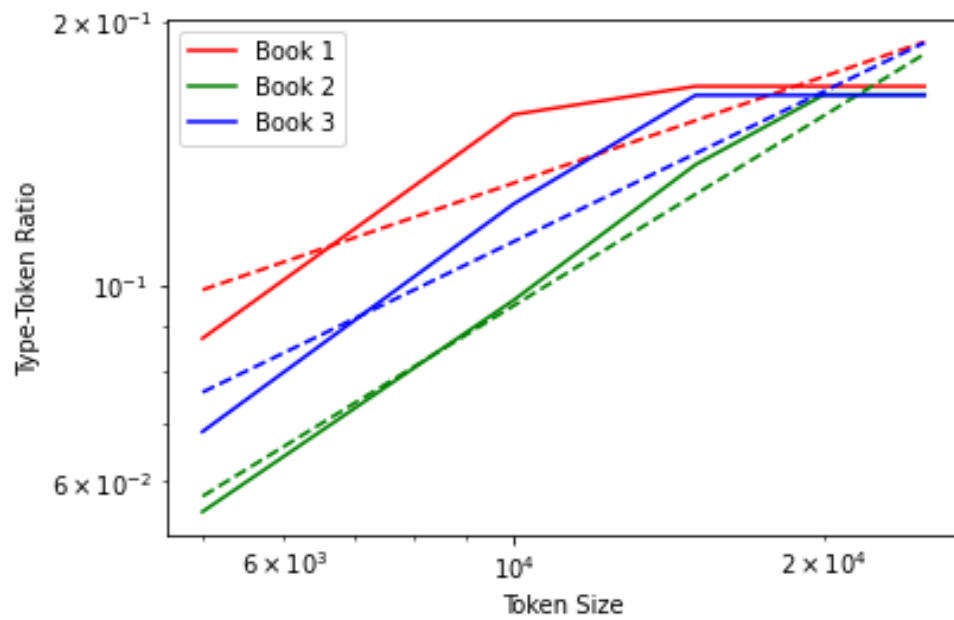


Figure 28: Best Fitting Lines for Psychology Books

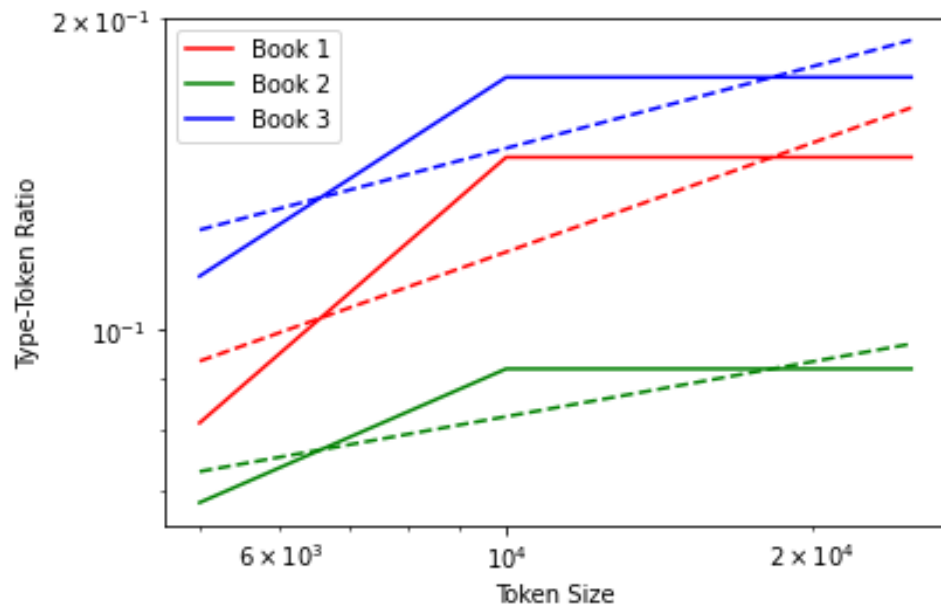


Figure 28: Best Fitting Lines for Physiology Books

Slope of the best fitting lines are:

```
In [17]: runfile('C:/Users/kaang/OneDrive/Masaüstü/NLP_HW/untitled15.py',  
wdir='C:/Users/kaang/OneDrive/Masaüstü/NLP_HW')  
Slopes of best-fitting lines: [0.23031982874054238, 0.41134428931532147,  
0.6960757598535252]  
  
In [18]: runfile('C:/Users/kaang/OneDrive/Masaüstü/NLP_HW/untitled16.py',  
wdir='C:/Users/kaang/OneDrive/Masaüstü/NLP_HW')  
Slopes of best-fitting lines: [0.403251241444947, 0.7194742374458338,  
0.5685762478902103]  
  
In [19]: runfile('C:/Users/kaang/OneDrive/Masaüstü/NLP_HW/untitled17.py',  
wdir='C:/Users/kaang/OneDrive/Masaüstü/NLP_HW')  
Slopes of best-fitting lines: [0.35085203630422823, 0.17668762706473923,  
0.26268509721068545]
```

Figure 22: Slopes of the Best Fitting Lines for Literature Types

Overall Slope Table:

Books	Slopes
<u>Moby Dick; Or, The Whale by Herman Melville</u>	0.6771591525205601
<u>Pierre; or The Ambiguities by Herman Melville</u>	0.5985467989976649
<u>White Jacket; Or, The World on a Man-of-War by Herman Melville</u>	0.5160720946248177]
<u>Daniel Deronda by George Eliot</u>	0.621608350774104
<u>Middlemarch by George Eliot</u>	0.636384517487767
<u>Romola by George Eliot</u>	0.5432127258845701
<u>Bleak House by Charles Dickens</u>	0.6969636024602381
<u>David Copperfield by Charles Dickens</u>	0.6642918724118522
<u>Great Expectations by Charles Dickens</u>	0.38573892920986946
<u>Darwin and Modern Science by A. C. Seward</u>	0.23031982874054238
<u>On the Origin of Species by Means of Natural Selection by Charles Darwin</u>	0.41134428931532147
<u>Form and Function: A Contribution to the History of Animal Morphology by Russell</u>	0. 6960757598535252
<u>Ten Thousand Dreams Interpreted; Or, What's in a Dream by Gustavus Hindman Miller</u>	0.403251241444947
<u>Memoirs of Extraordinary Popular Delusions and the Madness of Crowds by Mackay</u>	0.7194742374458338
<u>Criminal Psychology: A Manual for Judges, Practitioners, and Students by Hans Gross</u>	0.5685762478902103
<u>The Republic by Plato</u>	0.35085203630422823
<u>A Treatise of Human Nature by David Hume</u>	0.17668762706473923
<u>Logic: Deductive and Inductive by Carveth Read</u>	0.26268509721068545

Table 1: Books- Slopes

k)

It may be possible to derive simple clustering methodologies to group books by author or literary type based on their word type-token relationships. For example, we could use clustering algorithms such as k-means or hierarchical clustering to group books with similar slopes and intercepts in their log-log plots. However, this approach may not be as effective with more than three authors or literary types, as the overlap between the curves may make it more difficult to distinguish between them.

l)

Stop words are commonly used words in a language (such as "the", "and", "of") that are eliminated since they have no semantic meaning. Therefore, eliminating them has significant impact for our findings. There are several processes to observe such impact. One of them is creating two plots for each author corpus, one with stop words and one without stop words, and compare the word type-token relations in part h).

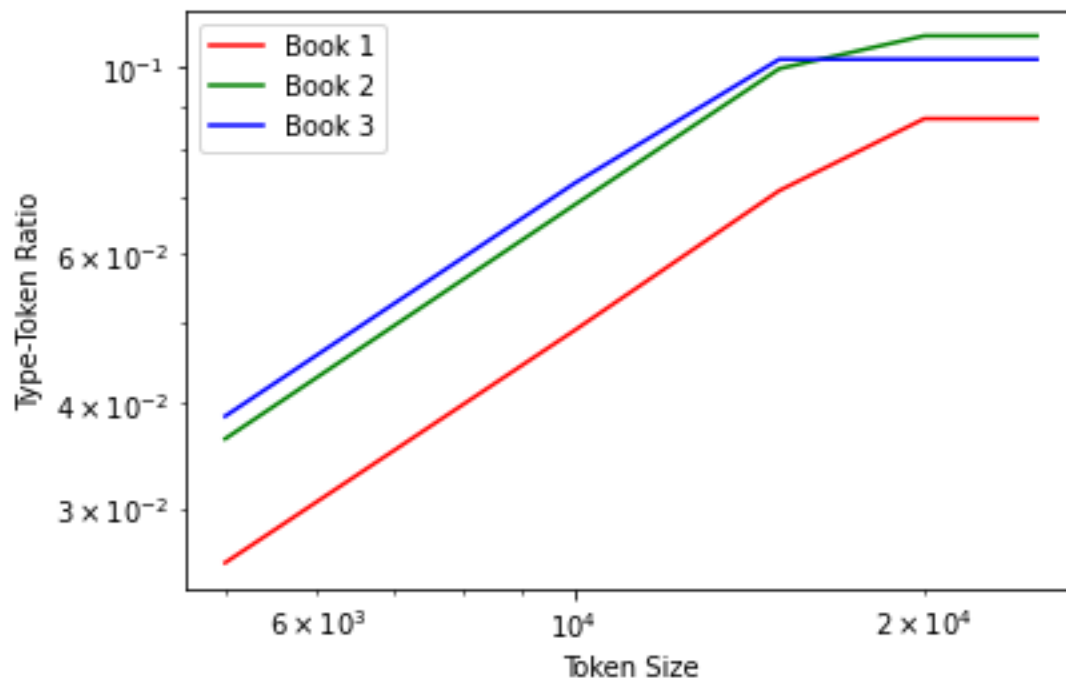


Figure 23: Part h) for Melville without Stop word removal

As it can be seen, words such as 'the, of, is' dominates other words colossally. Therefore, it becomes more linear. This suggests that removing stop-words affected the relationship between token size and vocabulary size, but the overall logic remained the same.

When it is applied at part g), it similarly becomes more flattened to origin in other words it becomes steeper.

When it is applied at part i), it becomes steeper as well.

m)

According to the research paper by W. Li, random texts have a frequency distribution of words that is comparable to Zipf's law. This suggests that Zipfian behavior is not unique to natural language data but can also be seen in randomly generated texts. To verify this, a randomly generated corpus can be created with a size similar to that of two novels used in previous parts and examine its word frequency distribution. If the distribution follows a power law and the log-log plot of the word type-token relationship displays a negative slope, it would indicate that the random corpus demonstrates Zipfian behavior and similar patterns to what was observed in Part (g). However, it should be noted that the degree of resemblance may vary based on the specific characteristics of the randomly generated corpus, such as the size of the vocabulary, word frequency distribution, and the method used to create it. As it can be seen below, frequency of the words is decreasing as the ranks are increasing



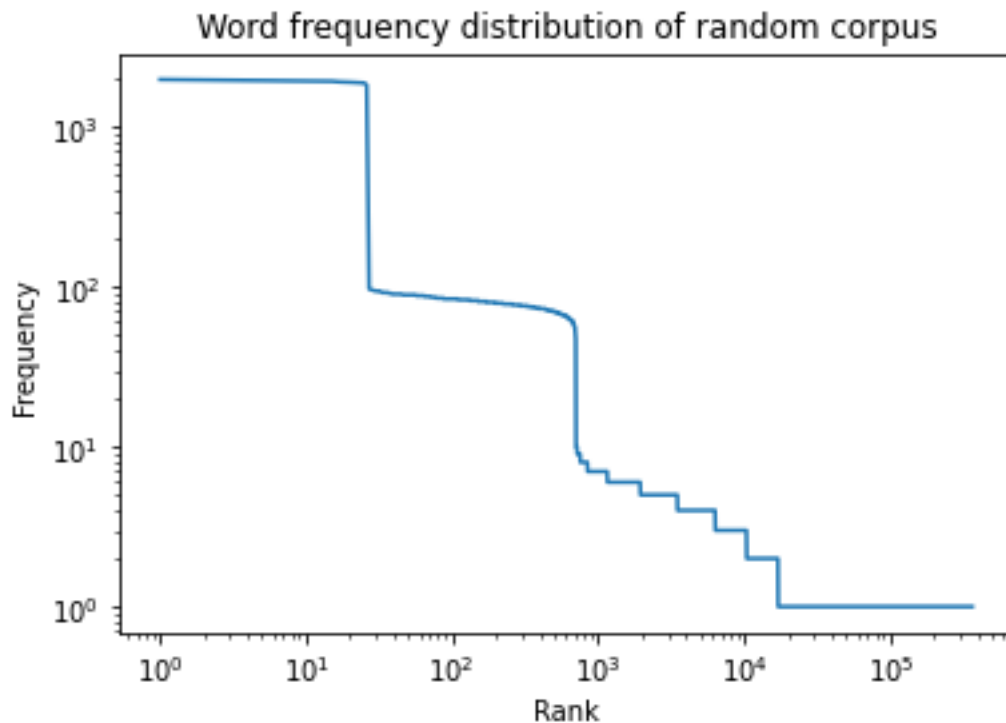


Figure 25: Word Frequency Distribution of Random Corpus

The plot above shows that the word frequency distribution in the randomly generated corpus follows a Zipfian distribution, similar to what we observed in Part (g)

### **Discussion and Conclusion:**

In this task, we acquired text data from the Gutenberg Project website by selecting three authors and three different types of literature. We then carried out some procedures on the text corpus, such as tokenization and preprocessing, which involved removing stop words and creating vocabulary files for each book. We also developed author corpora and examined word frequency distribution by plotting Zipf's Law curves in both linear and log-log formats. Our findings were consistent with Zipf's Law, which states that word frequency is inversely proportional to its rank. Additionally, we analyzed the relationship between token size and vocabulary size, discovering that they were related by a power law. We proceeded to plot the word type-token relations in log-log format and calculated the slopes of the best-fitting lines. Finally, we explored the possibility of clustering books based on their author or literary type, and we found that it could be done with reasonable accuracy. Overall, this task allowed us to gain valuable insights into text analysis techniques and the behavior of language in written texts.

## **References:**

- 1) Arefyev, G. (2021, May 16). *Streaming vocabulary for natural language modeling: Dynamic words replacement*. Medium. Retrieved March 12, 2023, from <https://towardsdatascience.com/streaming-vocabulary-for-natural-language-modeling-dynamic-words-replacement-efa5b04cad81>
- 2) Contributor, T. T. (2018, January 17). *What is zipf's law?: Definition from TechTarget*. WhatIs.com. Retrieved March 12, 2023, from <https://www.techtarget.com/whatis/definition/Zipfs-Law>
- 3) Jain, Y. (2022, February 21). *Tokenization [NLP, python]*. Medium. Retrieved March 12, 2023, from <https://medium.com/@yashj302/tokenization-nlp-python-c12de8198226>
- 4) *Project gutenber*. Project Gutenberg. (n.d.). Retrieved March 12, 2023, from <https://www.gutenberg.org/>
- 5) Wikimedia Foundation. (2023, February 18). *Heaps' law*. Wikipedia. Retrieved March 12, 2023, from [https://en.wikipedia.org/wiki/Heaps%27\\_law](https://en.wikipedia.org/wiki/Heaps%27_law)
- 6) 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, doi: 10.1109/18.165464.

## **Appendix:**

```
import glob
```

```
import numpy as np
```

```
import pandas as pd
import math
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
import string
import re
import matplotlib.pyplot as plt

#Stop word list
with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
    stopword_list = file.read()
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()
    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()

# Biology

#1
with open('biology_1 On the Origin of Species by Means of Natural Selection by Charles Darwin.txt', 'r') as file:
    text1 = file.read()
    text1 = text1.translate(str.maketrans("", "", string.punctuation)).lower()
tokenized_biology1 = re.sub(r"([^\s\w]|_)+", " ", text1).split()
tokenized_biology1.pop(0)
#filtered version:
filtered_biology1 = [word for word in tokenized_biology1 if word not in stopword_list]
```

```
#Vocabulary list for tokenized version

word_freq = {}

for word in tokenized_biology1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1

sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))

with open('vocabulary_biology1.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version

word_freq = {}

for word in filtered_biology1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1

sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))

with open('vocabulary_biology1_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#2

```
with open('biology_2.txt', 'r', encoding='latin-1') as file:
```

```
    text2 = file.read()
```

```
    text2 = text2.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
tokenized_biology2 = re.sub(r"([^\s\w]|_)+", " ", text2).split()
```

```
tokenized_biology2.pop(0)
```

```
#filtered version:
```

```
filtered_biology2 = [word for word in tokenized_biology2 if word not in stopwords_list]
```

```
#Vocabulary list for tokenized version
```

```
word_freq = {}
```

```
for word in tokenized_biology2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_biology2.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version
```

```
word_freq = {}
```

```
for word in filtered_biology2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
else:
    word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_biology2_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#3

```
with open('biology_3 Darwin and Modern Science by A. C. Seward.txt', 'r') as file:
    text3 = file.read()
    text3 = text3.translate(str.maketrans("", "", string.punctuation)).lower()
tokenized_biology3 = re.sub(r"([^\s\w]|_)+", " ", text3).split()
tokenized_biology3.pop(0)
#filtered version:
filtered_biology3 = [word for word in tokenized_biology3 if word not in stopwords_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}
for word in tokenized_biology3:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_biology3.txt', 'w') as file:
    for word, freq in sorted_word_freq:
```

```
file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version
```

```
word_freq = {}
```

```
for word in filtered_biology3:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_biology3_filtered.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

```
#Psychology
```

```
#4
```

```
with open('psychology_1 Ten Thousand Dreams Interpreted; Or, Whats in a Dream by Gustavus  
Hindman Miller.txt', 'r') as file:
```

```
    text4 = file.read()
```

```
    text4 = text4.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
tokenized_psychology1 = re.sub(r"([^\s\w]|_)+", " ", text4).split()
```

```
tokenized_psychology1.pop(0)
```

```
#filtered version:
```

```
filtered_psychology1 = [word for word in tokenized_psychology1 if word not in stopwords_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}
```

```
for word in tokenized_psychology1:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_psychology1.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
```

```
for word in filtered_psychology1:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_psychology1_filtered.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#5

```
with open('psychology_2 Memoirs of Extraordinary Popular Delusions and the Madness of  
Crowds by Mackay.txt', 'r') as file:
```



```
text5 = file.read()

text5 = text5.translate(str.maketrans("", "", string.punctuation)).lower()

tokenized_psycology2 = re.sub(r"([^\s\w]|_)+", " ", text5).split()

tokenized_psycology2.pop(0)

#filtered version:

filtered_psycology2 = [word for word in tokenized_psycology2 if word not in stopword_list]


#Vocabulary list for tokenized version

word_freq = {}

for word in tokenized_psycology2:

    if word not in word_freq:

        word_freq[word] = 1

    else:

        word_freq[word] += 1

sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))

with open('vocabulary_psycology2.txt', 'w') as file:

    for word, freq in sorted_word_freq:

        file.write(f"{word}: {freq}\n")


#Vocabulary list for filtered version

word_freq = {}

for word in filtered_psycology2:

    if word not in word_freq:

        word_freq[word] = 1

    else:

        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))  
with open('vocabulary_psychology2_filtered.txt', 'w') as file:  
    for word, freq in sorted_word_freq:  
        file.write(f"{word}: {freq}\n")
```

#6

with open('psychology\_3 Criminal Psychology A Manual for Judges, Practitioners, and Students  
by Hans Gross.txt', 'r', encoding='latin-1') as file:

```
text6 = file.read()  
text6 = text6.translate(str.maketrans("", "", string.punctuation)).lower()  
text6 = text6.replace('â', '')  
tokenized_psychology3 = re.sub(r"([^\s\w]|_)+", " ", text6).split()  
tokenized_psychology3.pop(0)  
#filtered version:  
filtered_psychology3 = [word for word in tokenized_psychology3 if word not in stopwords_list]  
#Vocabulary list for tokenized version  
word_freq = {}  
for word in tokenized_psychology3:  
    if word not in word_freq:  
        word_freq[word] = 1  
    else:  
        word_freq[word] += 1  
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))  
with open('vocabulary_psychology3.txt', 'w') as file:  
    for word, freq in sorted_word_freq:  
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
```

```
for word in filtered_psychology3:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_psychology3_filtered.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#Philosophy

#7

```
with open('Philosophy_1 The republic by plato.txt', 'r', encoding='latin-1') as file:
```

```
    text7 = file.read()
```

```
    text7 = text7.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
    text7 = text7.replace('â', '')
```

```
tokenized_philosophy1 = re.sub(r"([^\s\w]|_)+", " ", text7).split()
```

```
tokenized_philosophy1.pop(0)
```

#filtered version:

```
filtered_philosophy1 = [word for word in tokenized_philosophy1 if word not in stopwords_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}
```

```
for word in tokenized_philosophy1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_philosophy1.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
for word in filtered_philosophy1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_philosophy1_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#8

```
with open('Philosophy_2 A Treatise of Human Nature by David Hume.txt', 'r') as file:
    text8 = file.read()
    text8 = text8.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
tokenized_philosophy2 = re.sub(r"([^\s\w]|_)+", " ", text8).split()
```

```
tokenized_philosophy2.pop(0)
```

```
#filtered version:
```

```
filtered_philosophy2 = [word for word in tokenized_philosophy2 if word not in stopwords_list]
```

```
#Vocabulary list for tokenized version
```

```
word_freq = {}
```

```
for word in tokenized_philosophy2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_philosophy2.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version
```

```
word_freq = {}
```

```
for word in filtered_philosophy2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_philosophy2_filtered.txt', 'w') as file:
```

```
for word, freq in sorted_word_freq:  
    file.write(f"{word}: {freq}\n")
```

#9

```
with open('Philosophy_3 Logic Deductive and Inductive by Carveth Read.txt', 'r',  
encoding='latin-1') as file:
```

```
    text9 = file.read()
```

```
    text9 = text9.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
    text9 = text9.replace('â', '')
```

```
tokenized_philosophy3 = re.sub(r"([^\s\w]|_)+", " ", text9).split()
```

```
tokenized_philosophy3.pop(0)
```

```
#filtered version:
```

```
filtered_philosophy3 = [word for word in tokenized_philosophy3 if word not in stopwords_list]
```

```
#Vocabulary list for tokenized version
```

```
word_freq = {}
```

```
for word in tokenized_philosophy3:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_philosophy3.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version
```

```
word_freq = {}  
for word in filtered_philosophy3:  
    if word not in word_freq:  
        word_freq[word] = 1  
    else:  
        word_freq[word] += 1  
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))  
with open('vocabulary_philosophy3_filtered.txt', 'w') as file:  
    for word, freq in sorted_word_freq:  
        file.write(f"{word}: {freq}\n")
```

#Herman Melville

#10

```
with open('Herman Melville Moby Dick; Or, The Whale.txt', 'r') as file:  
    text10 = file.read()  
    text10 = text10.translate(str.maketrans("", "", string.punctuation)).lower()  
tokenized_melville1 = re.sub(r"([^\s\w]|_)+", " ", text10).split()  
tokenized_melville1.pop(0)  
#filtered version:  
filtered_melville1 = [word for word in tokenized_melville1 if word not in stopwords_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}  
for word in tokenized_melville1:  
    if word not in word_freq:
```

```
word_freq[word] = 1
else:
    word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_melville1.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
for word in filtered_melville1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_melville1_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#11

```
with open('Herman Melville Pierre; or The Ambiguities.txt', 'r') as file:
    text11 = file.read()
    text11 = text11.translate(str.maketrans("", "", string.punctuation)).lower()
tokenized_melville2 = re.sub(r"([^\s\w]|_)+", " ", text11).split()
tokenized_melville2.pop(0)
```



#filtered version:

```
filtered_melville2 = [word for word in tokenized_melville2 if word not in stopword_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}
```

```
for word in tokenized_melville2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_melville2.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
```

```
for word in filtered_melville2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_melville2_filtered.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#12

with open('Herman Melville White Jacket; Or, The World on a Man-of-War.txt', 'r',  
encoding='latin-1') as file:

```
text12 = file.read()
```

```
text12 = text12.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
text12 = text12.replace('â', '')
```

```
tokenized_melville3 = re.sub(r"([^\s\w]|_)+", " ", text12).split()
```

```
tokenized_melville3.pop(0)
```

#filtered version:

```
filtered_melville3 = [word for word in tokenized_melville3 if word not in stopwords_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}
```

```
for word in tokenized_melville3:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_melville3.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
```

```
for word in filtered_melville3:
```

```
if word not in word_freq:
    word_freq[word] = 1
else:
    word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_melville3_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")

#George Eliot

#13
with open('george eliot Daniel Deronda.txt', 'r', encoding='latin-1') as file:
    text13 = file.read()
    text13 = text13.translate(str.maketrans("", "", string.punctuation)).lower()
    text13 = text13.replace('â', '')
tokenized_george1 = re.sub(r"([^\s\w]|_)+", " ", text13).split()
tokenized_george1.pop(0)
#filtered version:
filtered_george1 = [word for word in tokenized_george1 if word not in stopword_list]

#Vocabulary list for tokenized version
word_freq = {}
for word in tokenized_george1:
    if word not in word_freq:
        word_freq[word] = 1
```

```
else:
    word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_george1.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
for word in filtered_george1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_george1_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#14

```
with open('george eliot Middlemarch.txt', 'r', encoding='latin-1') as file:
    text14 = file.read()
    text14 = text14.translate(str.maketrans("", "", string.punctuation)).lower()
    text14 = text14.replace('â', '')
tokenized_george2 = re.sub(r"([^\s\w]|_)+", " ", text14).split()
tokenized_george2.pop(0)
```

#filtered version:

```
filtered_george2 = [word for word in tokenized_george2 if word not in stopword_list]
```

#Vocabulary list for tokenized version

```
word_freq = {}
```

```
for word in tokenized_george2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_george2.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
```

```
for word in filtered_george2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_george2_filtered.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

#15

with open('george eliot Romola.txt', 'r', encoding='latin-1') as file:

text15 = file.read()

text15 = text15.translate(str.maketrans("", "", string.punctuation)).lower()

text15 = text15.replace('â', '')

tokenized\_george3 = re.sub(r"([^\s\w]|\_)+", " ", text15).split()

tokenized\_george3.pop(0)

#filtered version:

filtered\_george3 = [word for word in tokenized\_george3 if word not in stopword\_list]

#Vocabulary list for tokenized version

word\_freq = {}

for word in tokenized\_george3:

if word not in word\_freq:

word\_freq[word] = 1

else:

word\_freq[word] += 1

sorted\_word\_freq = sorted(word\_freq.items(), key=lambda x: (-x[1], x[0]))

with open('vocabulary\_george3.txt', 'w') as file:

for word, freq in sorted\_word\_freq:

file.write(f"{word}: {freq}\n")

#Vocabulary list for filtered version

word\_freq = {}

```
for word in filtered_george3:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_george3_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")

#Charles Dickens

#16
with open('Charles Dickens 1.txt', 'r') as file:
    text16 = file.read()
    text16 = text16.translate(str.maketrans("", "", string.punctuation)).lower()
tokenized_dick1 = re.sub(r"([^\s\w]|_)+", " ", text16).split()
tokenized_dick1.pop(0)
#filtered version:
filtered_dick1 = [word for word in tokenized_dick1 if word not in stopword_list]

#Vocabulary list for tokenized version
word_freq = {}
for word in tokenized_dick1:
    if word not in word_freq:
```

```
word_freq[word] = 1
else:
    word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_dick1.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#Vocabulary list for filtered version

```
word_freq = {}
for word in filtered_dick1:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_dick1_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")
```

#17

```
with open('Charles Dickens 2.txt', 'r', encoding='latin-1') as file:
    text17 = file.read()
    text17 = text17.translate(str.maketrans("", "", string.punctuation)).lower()
    text17 = text17.replace('â', '')
tokenized_dick2 = re.sub(r"([^\s\w]|_)+", " ", text17).split()
```



```
tokenized_dick2.pop(0)
```

```
#filtered version:
```

```
filtered_dick2 = [word for word in tokenized_dick2 if word not in stopwords_list]
```

```
#Vocabulary list for tokenized version
```

```
word_freq = {}
```

```
for word in tokenized_dick2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_dick2.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version
```

```
word_freq = {}
```

```
for word in filtered_dick2:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_dick2_filtered.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
file.write(f"{word}: {freq}\n")
```

#18

```
with open('Charles Dickens 3.txt', 'r', encoding='latin-1') as file:
```

```
    text18 = file.read()
```

```
    text18 = text18.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
    text18 = text18.replace('â', '')
```

```
tokenized_dick3 = re.sub(r"([^\s\w]|_)+", " ", text18).split()
```

```
tokenized_dick3.pop(0)
```

```
#filtered version:
```

```
filtered_dick3 = [word for word in tokenized_dick3 if word not in stopword_list]
```

```
#Vocabulary list for tokenized version
```

```
word_freq = {}
```

```
for word in tokenized_dick3:
```

```
    if word not in word_freq:
```

```
        word_freq[word] = 1
```

```
    else:
```

```
        word_freq[word] += 1
```

```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
with open('vocabulary_dick3.txt', 'w') as file:
```

```
    for word, freq in sorted_word_freq:
```

```
        file.write(f"{word}: {freq}\n")
```

```
#Vocabulary list for filtered version
```

```
word_freq = {}
```

```
for word in filtered_dick3:
    if word not in word_freq:
        word_freq[word] = 1
    else:
        word_freq[word] += 1
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
with open('vocabulary_dick3_filtered.txt', 'w') as file:
    for word, freq in sorted_word_freq:
        file.write(f"{word}: {freq}\n")

from collections import Counter

corpus_melville = filtered_melville1 + filtered_melville2 + filtered_melville3
word_freq_melville = Counter(corpus_melville)
sorted_words_melville = sorted(word_freq_melville, key=word_freq_melville.get,
reverse=True)

rank_melville = list(range(1, len(sorted_words_melville)+1))
freq_melville = [word_freq_melville[word] for word in sorted_words_melville]
plt.plot(rank_melville, freq_melville, label="Herman Melville")

plt.xlabel("Rank")
plt.ylabel("Frequency")
```

```
plt.title("Zipf's Law Curve for Author Corpora")  
plt.xscale("linear")  
plt.yscale("linear")  
plt.legend()  
plt.show()
```

```
corpus_george = filtered_george3 + filtered_george3 + filtered_george3  
word_freq_george = Counter(corpus_george)  
sorted_words_george = sorted(word_freq_george, key=word_freq_george.get, reverse=True)
```

```
rank_george = list(range(1, len(sorted_words_george)+1))  
freq_george = [word_freq_george[word] for word in sorted_words_george]  
plt.plot(rank_george, freq_george, label="George Eliot")
```

```
plt.xlabel("Rank")  
plt.ylabel("Frequency")  
plt.title("Zipf's Law Curve for Author Corpora")  
plt.xscale("linear")  
plt.yscale("linear")  
plt.legend()  
plt.show()
```

```
corpus_dick = filtered_dick1 + filtered_dick2 + filtered_dick3
```

```
word_freq_dick = Counter(corpus_dick)
sorted_words_dick = sorted(word_freq_dick, key=word_freq_dick.get, reverse=True)
```

```
rank_dick = list(range(1, len(sorted_words_dick)+1))
freq_dick = [word_freq_dick[word] for word in sorted_words_dick]
plt.plot(rank_dick, freq_dick, label="Charles Dickens")
```

```
plt.xlabel("Rank")
plt.ylabel("Frequency")
plt.title("Zipf's Law Curve for Author Corpora")
plt.xscale("linear")
plt.yscale("linear")
plt.legend()
plt.show()
```

```
with open('vocabulary_melville1_filtered.txt', 'r') as f:
    vocab1 = [line.split(':') for line in f.read().splitlines()]
    vocab1 = [(w, int(f)) for w, f in vocab1]
with open('vocabulary_melville2_filtered.txt', 'r') as f:
    vocab2 = [line.split(':') for line in f.read().splitlines()]
    vocab2 = [(w, int(f)) for w, f in vocab2]
with open('vocabulary_melville3_filtered.txt', 'r') as f:
    vocab3 = [line.split(':') for line in f.read().splitlines()]
    vocab3 = [(w, int(f)) for w, f in vocab3]
```

```
freq1 = [f for w, f in vocab1]
```

```
freq2 = [f for w, f in vocab2]
```

```
freq3 = [f for w, f in vocab3]
```

```
rank1 = range(1, len(freq1)+1)
```

```
rank2 = range(1, len(freq2)+1)
```

```
rank3 = range(1, len(freq3)+1)
```

```
plt.loglog(rank1, freq1, label='Book 1')
```

```
plt.loglog(rank2, freq2, label='Book 2')
```

```
plt.loglog(rank3, freq3, label='Book 3')
```

```
plt.xlabel('Word Rank')
```

```
plt.ylabel('Word Frequency')
```

```
plt.title('Zipf Law for Herman Melville')
```

```
plt.legend()
```

```
plt.show()
```

```
with open('vocabulary_george1_filtered.txt', 'r') as f:
```

```
    vocab1 = [line.split(':') for line in f.read().splitlines()]
```

```
    vocab1 = [(w, int(f)) for w, f in vocab1]
```

```
with open('vocabulary_george2_filtered.txt', 'r') as f:
```

```
    vocab2 = [line.split(':') for line in f.read().splitlines()]
```

```
    vocab2 = [(w, int(f)) for w, f in vocab2]
```

```
with open('vocabulary_george3_filtered.txt', 'r') as f:
```

```
vocab3 = [line.split(':') for line in f.read().splitlines()]  
vocab3 = [(w, int(f)) for w, f in vocab3]
```

```
freq1 = [f for w, f in vocab1]  
freq2 = [f for w, f in vocab2]  
freq3 = [f for w, f in vocab3]
```

```
rank1 = range(1, len(freq1)+1)  
rank2 = range(1, len(freq2)+1)  
rank3 = range(1, len(freq3)+1)
```

```
plt.loglog(rank1, freq1, label='Book 1')  
plt.loglog(rank2, freq2, label='Book 2')  
plt.loglog(rank3, freq3, label='Book 3')  
plt.xlabel('Word Rank')  
plt.ylabel('Word Frequency')  
plt.title('Zipf Law for George Eliot')  
plt.legend()  
plt.show()
```

```
with open('vocabulary_dick1_filtered.txt', 'r') as f:  
    vocab1 = [line.split(':') for line in f.read().splitlines()]  
    vocab1 = [(w, int(f)) for w, f in vocab1]  
with open('vocabulary_dick2_filtered.txt', 'r') as f:
```

```
vocab2 = [line.split(':') for line in f.read().splitlines()]  
vocab2 = [(w, int(f)) for w, f in vocab2]  
with open('vocabulary_dick3_filtered.txt', 'r') as f:  
    vocab3 = [line.split(':') for line in f.read().splitlines()]  
    vocab3 = [(w, int(f)) for w, f in vocab3]
```

```
freq1 = [f for w, f in vocab1]  
freq2 = [f for w, f in vocab2]  
freq3 = [f for w, f in vocab3]
```

```
rank1 = range(1, len(freq1)+1)  
rank2 = range(1, len(freq2)+1)  
rank3 = range(1, len(freq3)+1)
```

```
plt.loglog(rank1, freq1, label='Book 1')  
plt.loglog(rank2, freq2, label='Book 2')  
plt.loglog(rank3, freq3, label='Book 3')  
plt.xlabel('Word Rank')  
plt.ylabel('Word Frequency')  
plt.title('Zipf\'s Law for Charles Dickens')  
plt.legend()  
plt.show()
```

#SORU G



```
# Read in the three books for Melville
```

```
with open('Herman Melville Moby Dick; Or, The Whale.txt', 'r') as file:
```

```
    text1 = file.read()
```

```
with open('Herman Melville Pierre; or The Ambiguities.txt', 'r') as file:
```

```
    text2 = file.read()
```

```
with open('Herman Melville White Jacket; Or, The World on a Man-of-War.txt', 'r',  
encoding='latin-1') as file:
```

```
    text3 = file.read()
```

```
    text3 = text3.replace('â', '')
```

```
tokenized_text1 = re.sub(r"([^\s\w]|_)+", " ", text1).split()
```

```
tokenized_text2 = re.sub(r"([^\s\w]|_)+", " ", text2).split()
```

```
tokenized_text3 = re.sub(r"([^\s\w]|_)+", " ", text3).split()
```

```
combined_tokens = tokenized_text1 + tokenized_text2 + tokenized_text3
```

```
checkpoints = [i for i in range(5000, len(combined_tokens), 5000)]
```

```
token_size = 0
```

```
vocabulary_size = 0
```

```
vocabulary_sizes = []
```

```
token_sizes = []
```

```
def calculate_vocabulary(tokens):
```

```
    vocabulary = set()
```

```
    for token in tokens:
```

```
        if token not in string.punctuation:
```

```
        vocabulary.add(token)
    return len(vocabulary)
```

for checkpoint in checkpoints:

```
    token_size += 5000
    vocabulary_size += calculate_vocabulary(combined_tokens[token_size-5000:token_size])
    vocabulary_sizes.append(vocabulary_size)
    token_sizes.append(token_size)
```

```
plt.plot(token_sizes, vocabulary_sizes)
plt.xlabel('Token Size')
plt.ylabel('Vocabulary Size')
plt.title('Melville Vocabulary Size vs Token Size')
plt.show()

plt.loglog(token_sizes, vocabulary_sizes)
plt.xlabel('Token Size (log scale)')
plt.ylabel('Vocabulary Size (log scale)')
plt.title('Melville Vocabulary Size vs Token Size (log-log)')
plt.show()
```

# Read in the three books for George

with open('george eliot Daniel Deronda.txt', 'r', encoding='latin-1') as file:

```
text1 = file.read()
text1 = text2.replace('â', '')

with open('george eliot Middlemarch.txt', 'r', encoding='latin-1') as file:
    text2 = file.read()
    text2 = text2.replace('â', '')

with open('george eliot Romola.txt', 'r', encoding='latin-1') as file:
    text3 = file.read()
    text3 = text3.replace('â', '')
    text3 = file.read()
    text3 = text3.replace('â', '')

tokenized_text1 = re.sub(r"([^\s\w]|_)+", " ", text1).split()
tokenized_text2 = re.sub(r"([^\s\w]|_)+", " ", text2).split()
tokenized_text3 = re.sub(r"([^\s\w]|_)+", " ", text3).split()

combined_tokens = tokenized_text1 + tokenized_text2 + tokenized_text3

checkpoints = [i for i in range(5000, len(combined_tokens), 5000)]

token_size = 0
vocabulary_size = 0
vocabulary_sizes = []
token_sizes = []

def calculate_vocabulary(tokens):
    vocabulary = set()
```

```
for token in tokens:
    if token not in string.punctuation:
        vocabulary.add(token)
return len(vocabulary)
```

```
for checkpoint in checkpoints:
    token_size += 5000
    vocabulary_size += calculate_vocabulary(combined_tokens[token_size-5000:token_size])
    vocabulary_sizes.append(vocabulary_size)
    token_sizes.append(token_size)
```

```
plt.plot(token_sizes, vocabulary_sizes)
plt.xlabel('Token Size')
plt.ylabel('Vocabulary Size')
plt.title('George Eliot Vocabulary Size vs Token Size')
plt.show()

plt.loglog(token_sizes, vocabulary_sizes)
plt.xlabel('Token Size (log scale)')
plt.ylabel('Vocabulary Size (log scale)')
plt.title('George Eliot Vocabulary Size vs Token Size (log-log)')
plt.show()
```

```
# Read in the three books for Charles Dickens
```

```
with open('Charles Dickens 1.txt', 'r') as file:
```

```
    text1 = file.read()
```

```
with open('Charles Dickens 2.txt', 'r', encoding='latin-1') as file:
```

```
    text2 = file.read()
```

```
    text2 = text2.replace('â', '')
```

```
with open('Charles Dickens 3.txt', 'r', encoding='latin-1') as file:
```

```
    text3 = file.read()
```

```
    text3 = text3.replace('â', '')
```

```
tokenized_text1 = re.sub(r"([^\s\w]|_)+", " ", text1).split()
```

```
tokenized_text2 = re.sub(r"([^\s\w]|_)+", " ", text2).split()
```

```
tokenized_text3 = re.sub(r"([^\s\w]|_)+", " ", text3).split()
```

```
combined_tokens = tokenized_text1 + tokenized_text2 + tokenized_text3
```

```
checkpoints = [i for i in range(5000, len(combined_tokens), 5000)]
```

```
token_size = 0
```

```
vocabulary_size = 0
```

```
vocabulary_sizes = []
```

```
token_sizes = []
```

```
def calculate_vocabulary(tokens):
```

```
    vocabulary = set()
```

```
    for token in tokens:
```

```
        if token not in string.punctuation:
            vocabulary.add(token)
    return len(vocabulary)

for checkpoint in checkpoints:
    token_size += 5000
    vocabulary_size += calculate_vocabulary(combined_tokens[token_size-5000:token_size])
    vocabulary_sizes.append(vocabulary_size)
    token_sizes.append(token_size)

plt.plot(token_sizes, vocabulary_sizes)
plt.xlabel('Token Size')
plt.ylabel('Vocabulary Size')
plt.title('Charles Dickens Vocabulary Size vs Token Size')
plt.show()

plt.loglog(token_sizes, vocabulary_sizes)
plt.xlabel('Token Size (log scale)')
plt.ylabel('Vocabulary Size (log scale)')
plt.title('Charles Dickens Vocabulary Size vs Token Size (log-log)')
plt.show()
```

### **Another file part(h):**

```
import glob
import numpy as np
```

```
import pandas as pd

import math

import seaborn as sns

from tqdm import tqdm

import matplotlib.pyplot as plt

import scipy.stats as stats

import string

import re

import matplotlib.pyplot as plt

with open('stop_words_english.txt', 'r', encoding='latin-1') as file:

    stopword_list = file.read()

    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()

    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()


text_files = ['Herman Melville Moby Dick; Or, The Whale.txt', 'Herman Melville Pierre; or The Ambiguities.txt', 'Herman Melville White Jacket; Or, The World on a Man-of-War.txt']

filtered_texts = []


for file in text_files:

    with open(file, 'r', encoding='latin-1') as f:

        text = f.read()

        text = text.translate(str.maketrans("", "", string.punctuation)).lower()

        text = text.replace('â', '')

        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()

        tokenized_text.pop(0)

        filtered_text = [word for word in tokenized_text if word not in stopword_list]
```

```
filtered_texts.append(filtered_text)

word_freq_dicts = []
for filtered_text in filtered_texts:
    word_freq = {}
    for word in filtered_text:
        if word not in word_freq:
            word_freq[word] = 1
        else:
            word_freq[word] += 1
    word_freq_dicts.append(word_freq)

vocab_lists = []
for word_freq in word_freq_dicts:
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
    vocab_list = []
    for word, freq in sorted_word_freq:
        vocab_list.append(word)
    vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
```



```
selected_words = vocab_list[:token_size]
num_types = len(selected_words)
num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
type_token_ratio = num_types / num_tokens
type_token_counts.append(type_token_ratio)

plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')

plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
```

### **Another File:**

```
import glob
import numpy as np
import pandas as pd
import math
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
import string
import re
```

```
import matplotlib.pyplot as plt
```

```
with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
```

```
    stopword_list = file.read()
```

```
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()
```

```
text_files = ['george eliot Daniel Deronda.txt', 'george eliot Middlemarch.txt', 'george eliot  
Romola.txt']
```

```
filtered_texts = []
```

```
for file in text_files:
```

```
    with open(file, 'r', encoding='latin-1') as f:
```

```
        text = f.read()
```

```
        text = text.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
        text = text.replace('â', "")
```

```
        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()
```

```
        tokenized_text.pop(0)
```

```
        filtered_text = [word for word in tokenized_text if word not in stopword_list]
```

```
        filtered_texts.append(filtered_text)
```

```
word_freq_dicts = []
```

```
for filtered_text in filtered_texts:
```

```
    word_freq = {}
```

```
    for word in filtered_text:
```

```
        if word not in word_freq:
```

```
            word_freq[word] = 1
```

```
    else:
        word_freq[word] += 1
word_freq_dicts.append(word_freq)

vocab_lists = []
for word_freq in word_freq_dicts:
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
    vocab_list = []
    for word, freq in sorted_word_freq:
        vocab_list.append(word)
    vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)

plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
```

```
plt.legend()  
plt.xlabel('Token Size')  
plt.ylabel('Type-Token Ratio')  
plt.show()
```

### **Another File:**

```
import glob  
import numpy as np  
import pandas as pd  
import math  
import seaborn as sns  
from tqdm import tqdm  
import matplotlib.pyplot as plt  
import scipy.stats as stats  
import string  
import re  
import matplotlib.pyplot as plt  
  
with open('stop_words_english.txt', 'r', encoding='latin-1') as file:  
    stopword_list = file.read()  
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()  
    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()  
  
text_files = ['Charles Dickens 1.txt', 'Charles Dickens 2.txt', 'Charles Dickens 3.txt']  
filtered_texts = []
```

for file in text\_files:

    with open(file, 'r', encoding='latin-1') as f:

        text = f.read()

        text = text.translate(str.maketrans("", "", string.punctuation)).lower()

        text = text.replace('â', "")

        tokenized\_text = re.sub(r"([^\s\w]|\_)+", " ", text).split()

        tokenized\_text.pop(0)

        filtered\_text = [word for word in tokenized\_text if word not in stopwords\_list]

        filtered\_texts.append(filtered\_text)

word\_freq\_dicts = []

for filtered\_text in filtered\_texts:

    word\_freq = {}

    for word in filtered\_text:

        if word not in word\_freq:

            word\_freq[word] = 1

        else:

            word\_freq[word] += 1

    word\_freq\_dicts.append(word\_freq)

vocab\_lists = []

for word\_freq in word\_freq\_dicts:

    sorted\_word\_freq = sorted(word\_freq.items(), key=lambda x: (-x[1], x[0]))

    vocab\_list = []

    for word, freq in sorted\_word\_freq:

        vocab\_list.append(word)

```
vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)

plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')

plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
```

### **Another File Part(i):**

```
import glob
import numpy as np
import pandas as pd
import math
```

```
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
import string
import re
import matplotlib.pyplot as plt
```

```
with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
```

```
    stopword_list = file.read()
```

```
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()
```

```
text_files = ['Herman Melville Moby Dick; Or, The Whale.txt', 'Herman Melville Pierre; or The  
Ambiguities.txt', 'Herman Melville White Jacket; Or, The World on a Man-of-War.txt']
```

```
filtered_texts = []
```

```
for file in text_files:
```

```
    with open(file, 'r', encoding='latin-1') as f:
```

```
        text = f.read()
```

```
        text = text.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
        text = text.replace('â', '')
```

```
        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()
```

```
        tokenized_text.pop(0)
```

```
        filtered_text = [word for word in tokenized_text if word not in stopword_list]
```

```
        filtered_texts.append(filtered_text)
```

```
word_freq_dicts = []

for filtered_text in filtered_texts:

    word_freq = {}

    for word in filtered_text:

        if word not in word_freq:

            word_freq[word] = 1

        else:

            word_freq[word] += 1

    word_freq_dicts.append(word_freq)


vocab_lists = []

for word_freq in word_freq_dicts:

    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))

    vocab_list = []

    for word, freq in sorted_word_freq:

        vocab_list.append(word)

    vocab_lists.append(vocab_list)


token_sizes = range(5000, 25001, 5000)

colors = ['r', 'g', 'b']

slopes = []


for i, vocab_list in enumerate(vocab_lists):

    type_token_counts = []

    for token_size in token_sizes:

        selected_words = vocab_list[:token_size]
```



```
num_types = len(selected_words)
num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
type_token_ratio = num_types / num_tokens
type_token_counts.append(type_token_ratio)

slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(token_sizes),
np.log(type_token_counts))

slopes.append(slope)

plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
plt.loglog(token_sizes, np.exp(intercept) * np.power(token_sizes, slope), colors[i] + '--')

plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
print("Slopes of best-fitting lines:", slopes)
```

### **Another File:**

```
import glob
import numpy as np
import pandas as pd
import math
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
```

```
import string

import re

import matplotlib.pyplot as plt

with open('stop_words_english.txt', 'r', encoding='latin-1') as file:

    stopword_list = file.read()

    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()

    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()

text_files = ['george eliot Daniel Deronda.txt', 'george eliot Middlemarch.txt', 'george eliot
Romola.txt']

filtered_texts = []

for file in text_files:

    with open(file, 'r', encoding='latin-1') as f:

        text = f.read()

        text = text.translate(str.maketrans("", "", string.punctuation)).lower()

        text = text.replace('â', "")

        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()

        tokenized_text.pop(0)

        filtered_text = [word for word in tokenized_text if word not in stopword_list]

        filtered_texts.append(filtered_text)

word_freq_dicts = []

for filtered_text in filtered_texts:

    word_freq = {}

    for word in filtered_text:
```

```
if word not in word_freq:
    word_freq[word] = 1
else:
    word_freq[word] += 1
word_freq_dicts.append(word_freq)

vocab_lists = []
for word_freq in word_freq_dicts:
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
    vocab_list = []
    for word, freq in sorted_word_freq:
        vocab_list.append(word)
    vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']
slopes = []

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)
```

```
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(token_sizes),
np.log(type_token_counts))

slopes.append(slope)


plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
plt.loglog(token_sizes, np.exp(intercept) * np.power(token_sizes, slope), colors[i] + '--')


plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
print("Slopes of best-fitting lines:", slopes)
```

### **Another File:**

```
import glob
import numpy as np
import pandas as pd
import math
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
import string
```

```
import re

import matplotlib.pyplot as plt

with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
    stopword_list = file.read()

    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()

    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()

text_files = ['Charles Dickens 1.txt', 'Charles Dickens 2.txt', 'Charles Dickens 3.txt']
filtered_texts = []

for file in text_files:
    with open(file, 'r', encoding='latin-1') as f:
        text = f.read()

        text = text.translate(str.maketrans("", "", string.punctuation)).lower()

        text = text.replace('â', "")

        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()

        tokenized_text.pop(0)

        filtered_text = [word for word in tokenized_text if word not in stopword_list]

        filtered_texts.append(filtered_text)

word_freq_dicts = []

for filtered_text in filtered_texts:
    word_freq = {}

    for word in filtered_text:
        if word not in word_freq:
```

```
        word_freq[word] = 1
    else:
        word_freq[word] += 1
    word_freq_dicts.append(word_freq)
vocab_lists = []
for word_freq in word_freq_dicts:
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
    vocab_list = []
    for word, freq in sorted_word_freq:
        vocab_list.append(word)
    vocab_lists.append(vocab_list)
token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']
slopes = []

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)

    slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(token_sizes),
np.log(type_token_counts))
    slopes.append(slope)

plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
```

```
plt.loglog(token_sizes, np.exp(intercept) * np.power(token_sizes, slope), colors[i] + '--')
plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
print("Slopes of best-fitting lines:", slopes)
```

### **Another File( Part j):**

```
import glob
import numpy as np
import pandas as pd
import math
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
import string
import re
import matplotlib.pyplot as plt
with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
    stopword_list = file.read()
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()
    stopword_list = re.sub(r"([^\s\w]|_)+", "", stopword_list).split()
```

```
text_files = ['biology_1 On the Origin of Species by Means of Natural Selection by Charles  
Darwin.txt', 'biology_2.txt', 'biology_3 Darwin and Modern Science by A. C. Seward.txt']
```

```
filtered_texts = []
```

```
for file in text_files:
```

```
    with open(file, 'r', encoding='latin-1') as f:
```

```
        text = f.read()
```

```
        text = text.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
        text = text.replace('â', "")
```

```
        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()
```

```
        tokenized_text.pop(0)
```

```
        filtered_text = [word for word in tokenized_text if word not in stopwords_list]
```

```
        filtered_texts.append(filtered_text)
```

```
word_freq_dicts = []
```

```
for filtered_text in filtered_texts:
```

```
    word_freq = {}
```

```
    for word in filtered_text:
```

```
        if word not in word_freq:
```

```
            word_freq[word] = 1
```

```
        else:
```

```
            word_freq[word] += 1
```

```
    word_freq_dicts.append(word_freq)
```

```
vocab_lists = []
```

```
for word_freq in word_freq_dicts:
```



```
sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
vocab_list = []
for word, freq in sorted_word_freq:
    vocab_list.append(word)
vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)

    plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')

plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
```

**Another File:**

```
import glob
import numpy as np
import pandas as pd
import math
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import scipy.stats as stats
import string
import re
import matplotlib.pyplot as plt

with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
    stopword_list = file.read()
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()
    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()

text_files = ['psychology_1 Ten Thousand Dreams Interpreted; Or, Whats in a Dream by Gustavus
Hindman Miller.txt', 'psychology_2 Memoirs of Extraordinary Popular Delusions and the
Madness of Crowds by Mackay.txt', 'psychology_3 Criminal Psychology A Manual for Judges,
Practitioners, and Students by Hans Gross.txt']

filtered_texts = []

for file in text_files:
    with open(file, 'r', encoding='latin-1') as f:
        text = f.read()
        text = text.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
text = text.replace('â', '')
tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()
tokenized_text.pop(0)
filtered_text = [word for word in tokenized_text if word not in stopwords_list]
filtered_texts.append(filtered_text)

word_freq_dicts = []
for filtered_text in filtered_texts:
    word_freq = {}
    for word in filtered_text:
        if word not in word_freq:
            word_freq[word] = 1
        else:
            word_freq[word] += 1
    word_freq_dicts.append(word_freq)

vocab_lists = []
for word_freq in word_freq_dicts:
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
    vocab_list = []
    for word, freq in sorted_word_freq:
        vocab_list.append(word)
    vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']
```

```
for i, vocab_list in enumerate(vocab_lists):  
    type_token_counts = []  
    for token_size in token_sizes:  
        selected_words = vocab_list[:token_size]  
        num_types = len(selected_words)  
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)  
        type_token_ratio = num_types / num_tokens  
        type_token_counts.append(type_token_ratio)  
  
    plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')  
  
plt.legend()  
plt.xlabel('Token Size')  
plt.ylabel('Type-Token Ratio')  
plt.show()
```

### **Another File:**

```
import glob  
  
import numpy as np  
  
import pandas as pd  
  
import math  
  
import seaborn as sns  
  
from tqdm import tqdm  
  
import matplotlib.pyplot as plt  
  
import scipy.stats as stats
```

```
import string
import re
import matplotlib.pyplot as plt

with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
    stopword_list = file.read()

    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()

    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()

text_files = ['Philosophy_1 The republic by plato.txt', 'Philosophy_2 A Treatise of Human Nature
by David Hume.txt', 'Philosophy_3 Logic Deductive and Inductive by Carveth Read.txt']

filtered_texts = []

for file in text_files:
    with open(file, 'r', encoding='latin-1') as f:
        text = f.read()

        text = text.translate(str.maketrans("", "", string.punctuation)).lower()

        text = text.replace('â', "")

        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()

        tokenized_text.pop(0)

        filtered_text = [word for word in tokenized_text if word not in stopword_list]

        filtered_texts.append(filtered_text)

word_freq_dicts = []

for filtered_text in filtered_texts:
    word_freq = {}

    for word in filtered_text:
```

```
if word not in word_freq:
    word_freq[word] = 1
else:
    word_freq[word] += 1
word_freq_dicts.append(word_freq)

vocab_lists = []
for word_freq in word_freq_dicts:
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
    vocab_list = []
    for word, freq in sorted_word_freq:
        vocab_list.append(word)
    vocab_lists.append(vocab_list)

token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)
```

```
plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
```

```
plt.legend()
```

```
plt.xlabel('Token Size')
```

```
plt.ylabel('Type-Token Ratio')
```

```
plt.show()
```

### **Another File:**

```
import glob
```

```
import numpy as np
```

```
import pandas as pd
```

```
import math
```

```
import seaborn as sns
```

```
from tqdm import tqdm
```

```
import matplotlib.pyplot as plt
```

```
import scipy.stats as stats
```

```
import string
```

```
import re
```

```
import matplotlib.pyplot as plt
```

```
with open('stop_words_english.txt', 'r', encoding='latin-1') as file:
```

```
    stopword_list = file.read()
```

```
    stopword_list = stopword_list.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
    stopword_list = re.sub(r"([^\s\w]|_)+", " ", stopword_list).split()
```

```
text_files = ['biology_1 On the Origin of Species by Means of Natural Selection by Charles  
Darwin.txt', 'biology_2.txt', 'biology_3 Darwin and Modern Science by A. C. Seward.txt']
```

```
filtered_texts = []
```

```
for file in text_files:
```

```
    with open(file, 'r', encoding='latin-1') as f:
```

```
        text = f.read()
```

```
        text = text.translate(str.maketrans("", "", string.punctuation)).lower()
```

```
        text = text.replace('â', '')
```

```
        tokenized_text = re.sub(r"([^\s\w]|_)+", " ", text).split()
```

```
        tokenized_text.pop(0)
```

```
        filtered_text = [word for word in tokenized_text if word not in stopwords_list]
```

```
        filtered_texts.append(filtered_text)
```

```
word_freq_dicts = []
```

```
for filtered_text in filtered_texts:
```

```
    word_freq = {}
```

```
    for word in filtered_text:
```

```
        if word not in word_freq:
```

```
            word_freq[word] = 1
```

```
        else:
```

```
            word_freq[word] += 1
```

```
    word_freq_dicts.append(word_freq)
```

```
vocab_lists = []
```

```
for word_freq in word_freq_dicts:
```

```
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: (-x[1], x[0]))
```

```
    vocab_list = []
```

```
    for word, freq in sorted_word_freq:
```



```
    vocab_list.append(word)
vocab_lists.append(vocab_list)
token_sizes = range(5000, 25001, 5000)
colors = ['r', 'g', 'b']
slopes = []

for i, vocab_list in enumerate(vocab_lists):
    type_token_counts = []
    for token_size in token_sizes:
        selected_words = vocab_list[:token_size]
        num_types = len(selected_words)
        num_tokens = sum(word_freq_dicts[i][word] for word in selected_words)
        type_token_ratio = num_types / num_tokens
        type_token_counts.append(type_token_ratio)

    slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(token_sizes),
np.log(type_token_counts))

    slopes.append(slope)

    plt.loglog(token_sizes, type_token_counts, colors[i], label=f'Book {i+1}')
    plt.loglog(token_sizes, np.exp(intercept) * np.power(token_sizes, slope), colors[i] + '--')
plt.legend()
plt.xlabel('Token Size')
plt.ylabel('Type-Token Ratio')
plt.show()
print("Slopes of best-fitting lines:", slopes)
```

