NLP Project Round - 1

Team Name : Three's Company

Ram Ahuja *19ucs017*

Aayush Mehta *19ucs100*

Vaibhav Gupta *19ucs115*

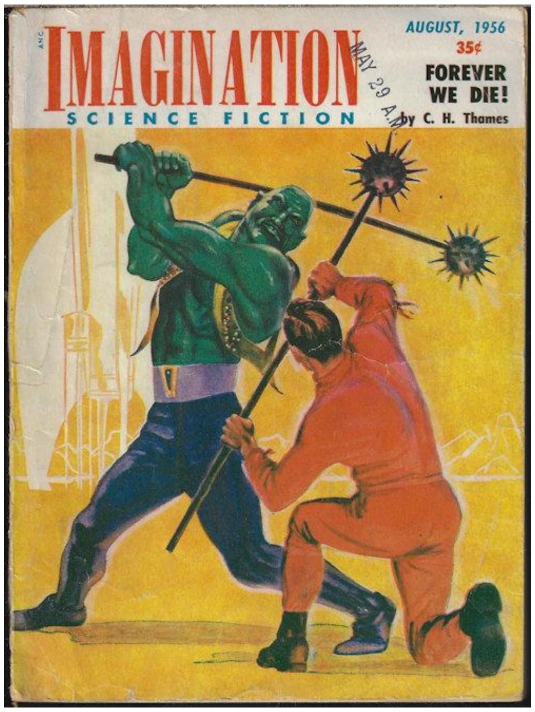
GitHub Repo :

https://github.com/Aayushmehta2001/NLP_Project

Overview :

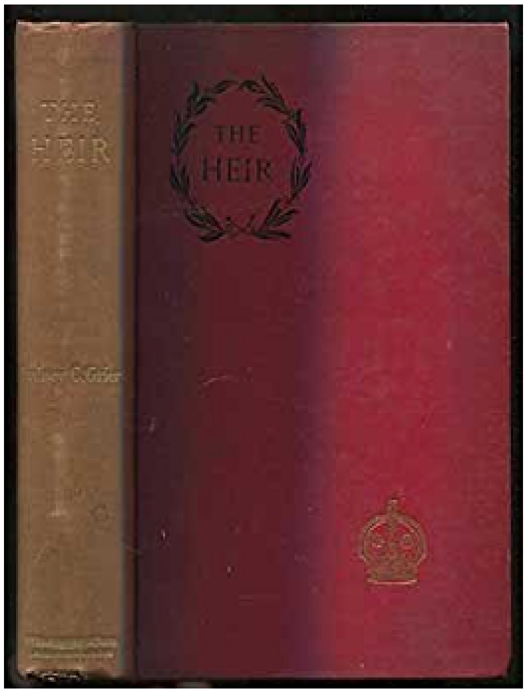
In this project we perform textual analysis on two of our chosen books "Traitor's Choice" and " The Heir" . After that we will apply POS tagging on both the books . We will do all this using NLP techniques, with the help of python libraries.

Books used :



Traitor's Choice

By Paul W. Fairman



The Heir

By Sydney C. Grier

Goals :

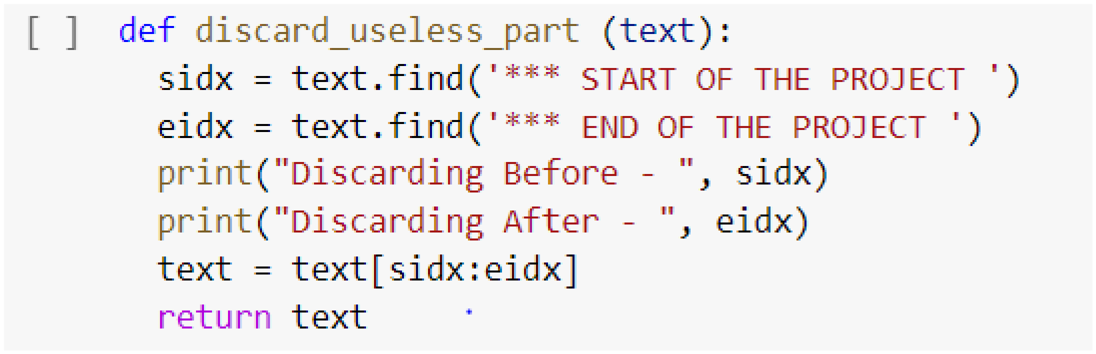
1. Import the text from two books, let's call it as T1 and T2.
2. Perform simple text pre-processing steps and tokenize the text T1 and T2.
3. Analyze the frequency distribution of tokens in T1 and T2 separately.
4. Create a Word Cloud of T1 and T2 using the token that you have got.
5. Remove the stopwords from T1 and T2 and then again create a word cloud.
6. Compare with word clouds before the removal of stopwords.
7. Evaluate the relationship between the word length and frequency for both T1 and T2.
8. Do PoS Tagging for both T1 and T2 using anyone of the four tagset studied in the class and Get the distribution of various tags .

Python Libraries Used :

- Urllib - Used to fetch text data from Gutenberg URLs
- NLTK - Used for Tokenizing, Lemmatization and Removing Stopwords
- Re - Used to remove URLs and Decontract Contractions in English Language
- Word Cloud - Used to create WordClouds from Tokenized Data Inflect - Used to replace numbers with words
- Maplotlib - Used to Visualize our text data

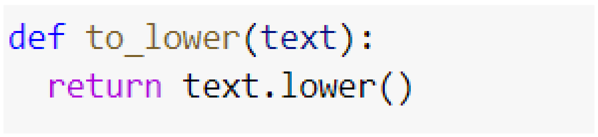
Data Preprocessing Steps :

1. **Discard Useless Portion of book -**
   We will Discard the Documentation part of the book that is of no use to us.



```python
def discard_useless_part (text):
  sidx = text.find('*** START OF THE PROJECT ')
  eidx = text.find('*** END OF THE PROJECT ')
  print("Discarding Before - ", sidx)
  print("Discarding After - ", eidx)
  text = text[sidx:eidx]
  return text
```

2. **Convert all data to Lowercase -**
   We will convert all text data to lowercase, as the case does not contribute much to the meaning of data.



```python
def to_lower(text):
  return text.lower()
```

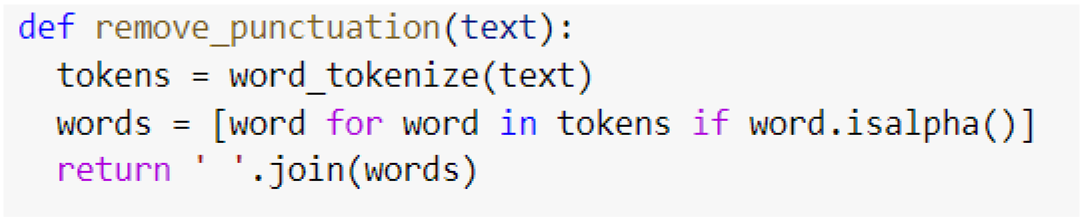
3. **Converting Number to Words -**
   For this, we use inflect Python Library which has a function p.number_to_words that will give us the English equivalent of a number using basic mapping techniques.

```
def num2word(text):
  list_of_words = text.split()
  modified_text = []

  for word in list_of_words:
    if word.isdigit():
      number_in_word = p.number_to_words(word)
      modified_text.append(number_in_word)
    else:
      modified_text.append(word)

  return ' '.join(modified_text)
```

4. Removing Contractions and Punctuations -

We will do this using a Python Library **re** that will help us apply regular expressions on our data as desired.

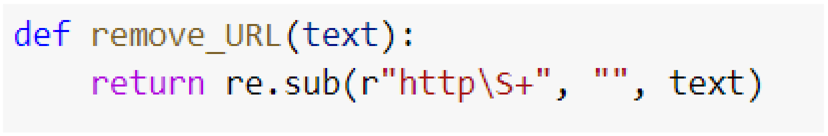
```
def decontracted(text):
    # specific
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)

    # general
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text
```



```python
def remove_punctuation(text):
  tokens = word_tokenize(text)
  words = [word for word in tokens if word.isalpha()]
  return ' '.join(words)
```

5. Removing URLs -

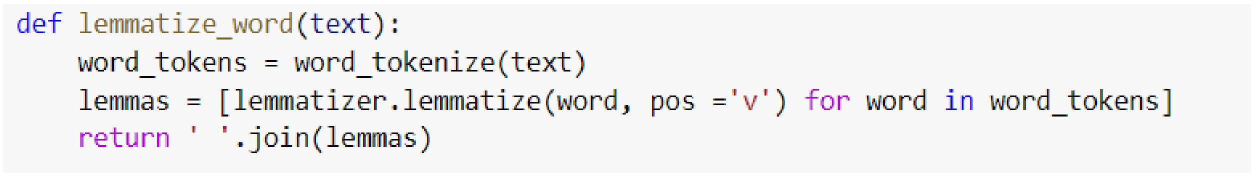
Again, we would do this using **re**



```python
def remove_URL(text):
    return re.sub(r"http\S+", "", text)
```

6. Lemmatization -

We do Lemmatization with the help of **WordNetLemmatizer()** function from **nltk.stem** which gives the lemmatized form of all verbs



```python
def lemmatize_word(text):
    word_tokens = word_tokenize(text)
    lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word_tokens]
    return ' '.join(lemmas)
```

Data Preparation :

We apply all the functionalities we added above and Prepare our data for analysis.

```
def PreProcessedBook(url):
  book = read_book(url)
  print_book_title_and_length(book)
  text = decode_book(book)
  text = discard_useless_part(text)
  text = to_lower(text)
  text = remove_URL(text)
  text = decontracted(text)
  text = num2word(text)
  text = remove_punctuation(text)
  text = lemmatize_word(text)
  return (text)
```

```
book1_text = PreProcessedBook(url1)
book2_text = PreProcessedBook(url2)
```

Problem Statements and Inferences :

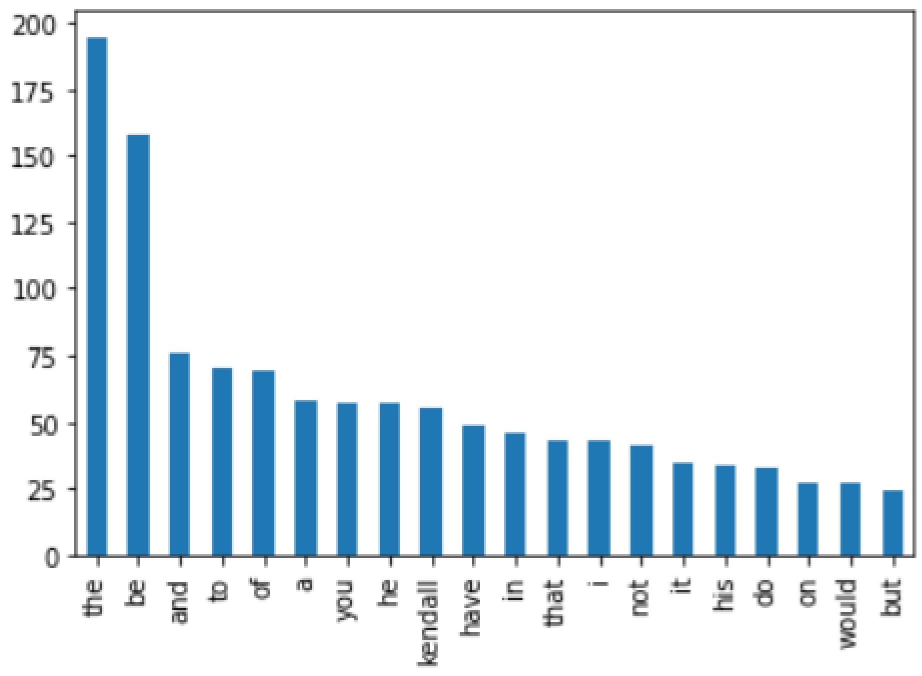
- **Analyze the frequency distribution of tokens in T1 and T2 separately.**

  For this step, we take help of python library pandas
  First we tokenize the given data, and then we plot a histogram of top 10 most frequent
  words.

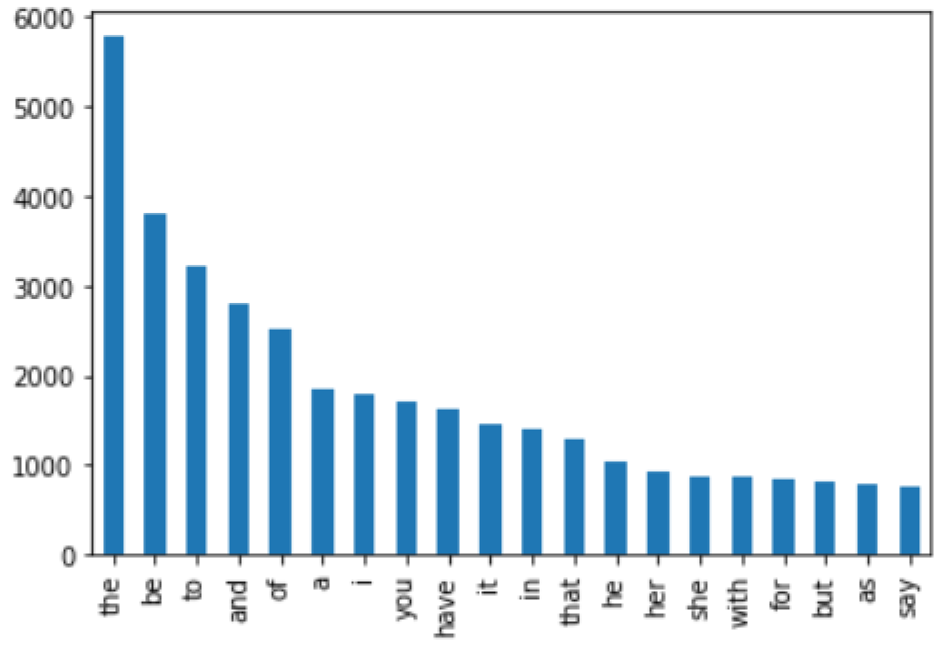
```
word_tokens1 = word_tokenize(book1_text)
pd.Series(word_tokens1).value_counts()[:20].plot(kind='bar')
```

```
word_tokens2 = word_tokenize(book2_text)
pd.Series(word_tokens2).value_counts()[:20].plot(kind='bar')
```

Frequency distribution of T1 :



Frequency distribution of T2 :



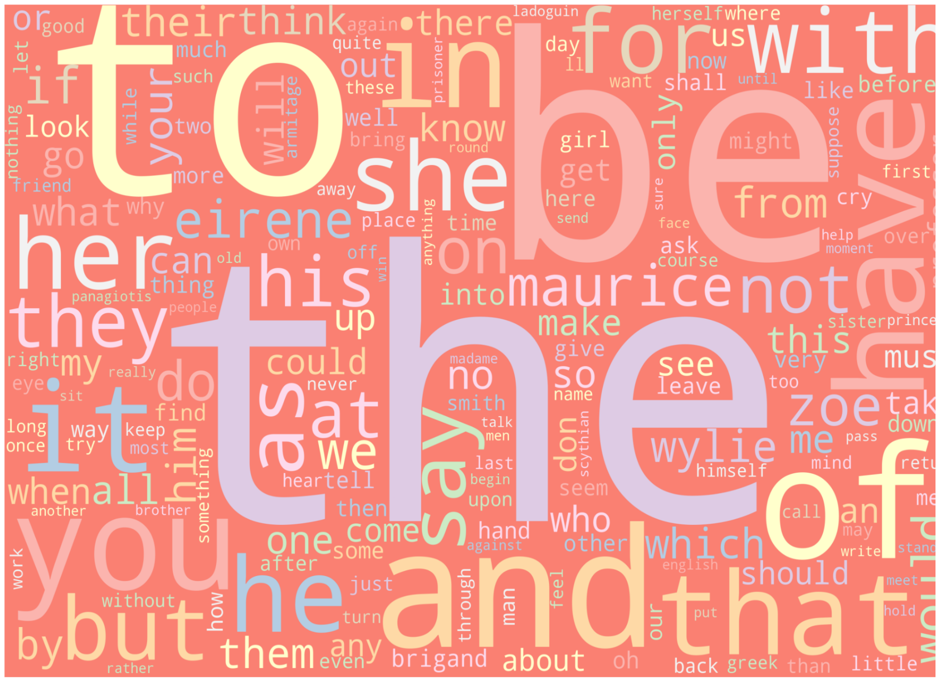
- **Generating a word cloud from T1 and T2**

  For this task we take help of python library **'wordcloud'** and its function '**WordCloud()**' It helps in generating wordclouds from a list of Tags from Text data.

```
# Generate word cloud
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1,
                      background_color='salmon', colormap='Pastel1',stopwords= [],
                        collocations=False).generate(' '.join(word_tokens1))
# Plot
plot_cloud(wordcloud)
```

the
be
and
to
you
of
he
kendall
that
centaurian
have
his
in
it
not
will
would
with
wife
for
as
they
say
time
right
clare
her
scientist
defense
project
man
but
do
out
on
think
what
we
your
this
after
get
yellowstone
terra
planet

Wordcloud of T1



Wordcloud of T2

Inferences

We infer from the above visualizations that

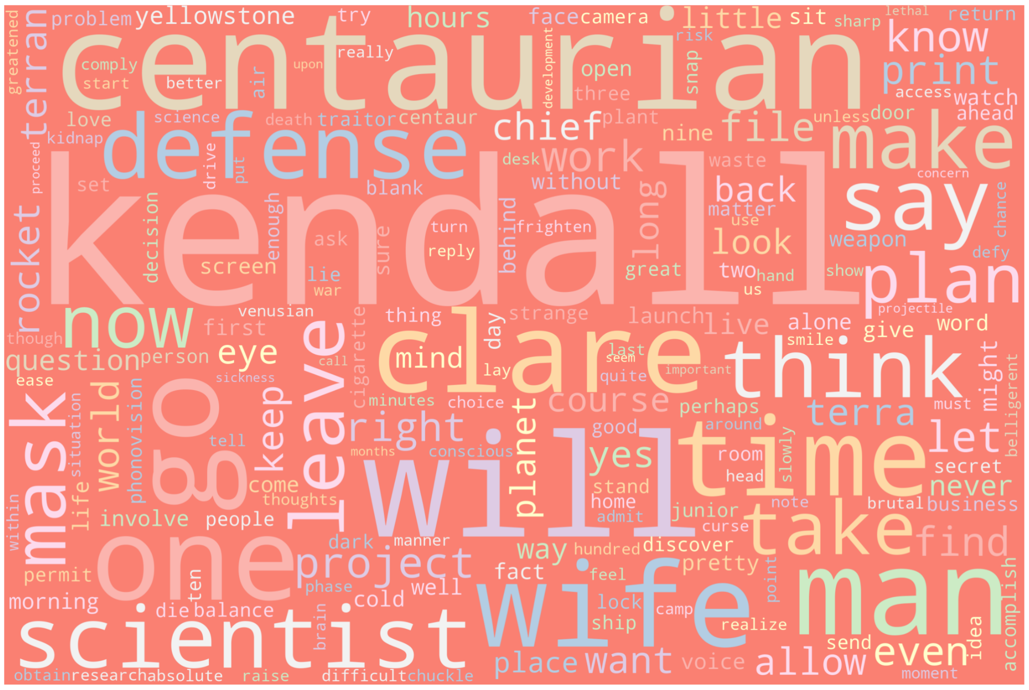
- Words like ‘and’, ‘be’ and ‘the’ are the most frequently used words in T1
- Words like ‘to’, ‘be’ and ‘the’ are the most frequently used words in T2
- These words do not contribute to the meaning of the sentence and are mostly useless for us
- These words are known as ‘stopwords’ and we need to get them out of it

- **Generating new word clouds after removing stopwords**

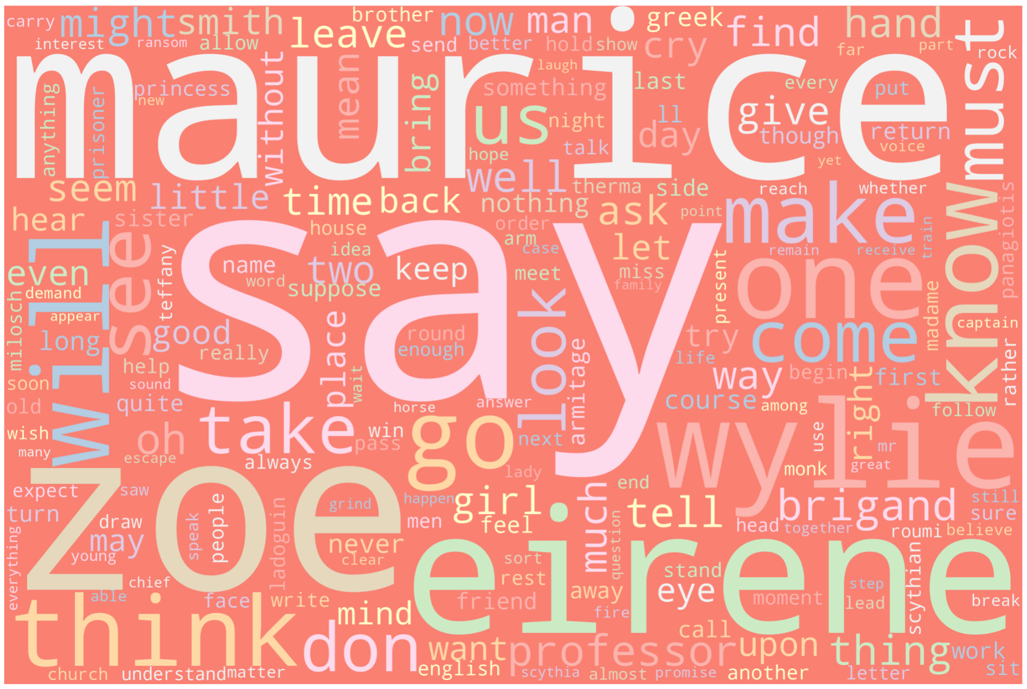
  To remove stopwords, we use an inbuilt function in nltk called STOPWORDS.

```
def remove_stopwords(tokens):
    return [word for word in tokens if word not in STOPWORDS]
```

```
T1 = remove_stopwords(word_tokens1)
T2 = remove_stopwords(word_tokens2)
```



Wordcloud of T1(after Removing Stopwords)



Wordcloud of T2(after Removing Stopwords)

Inferences

We infer from the above visualizations that

● Now most of the words that were of no meaning to us have been removed.

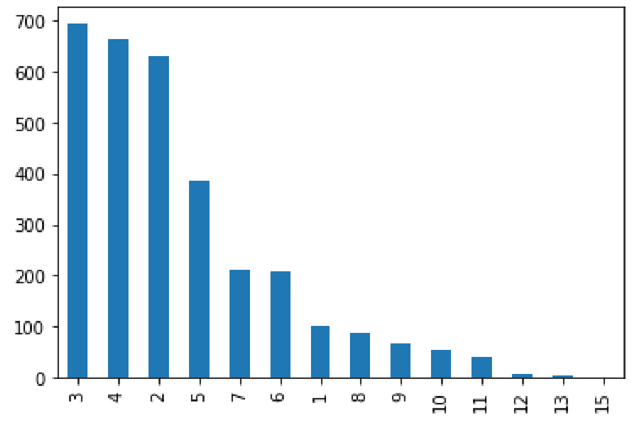
● New words like 'kendall', 'centaurian' and 'clare' now dominate in frequency in T1 which sort of reveals the name of characters of the book around whom the story will revolve.

● Words like 'say', 'zoe, 'one' and 'maurice' are the new frequently used words in T2, though it is tough to make inferences based on this information but we are able to roughly guess that there is some 'don' and 'professor' in the story.

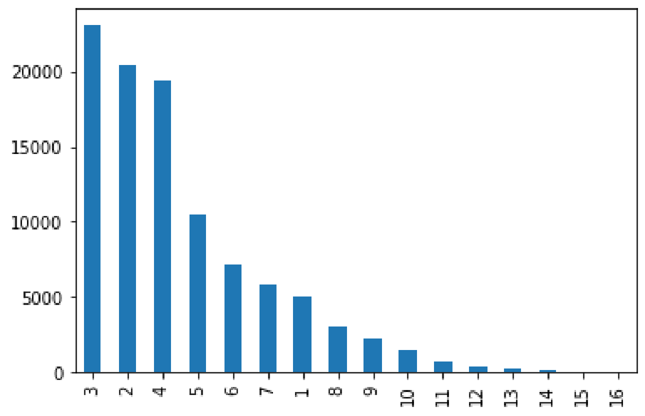
● We have gotten rid of 'stopwords' and are now able to draw meaningful conclusions from the data.

● **As a part of the project, we would also like to evaluate the relationship between the word length and frequency for both T1 and T2 both before and after the removal of stopwords.**

Before

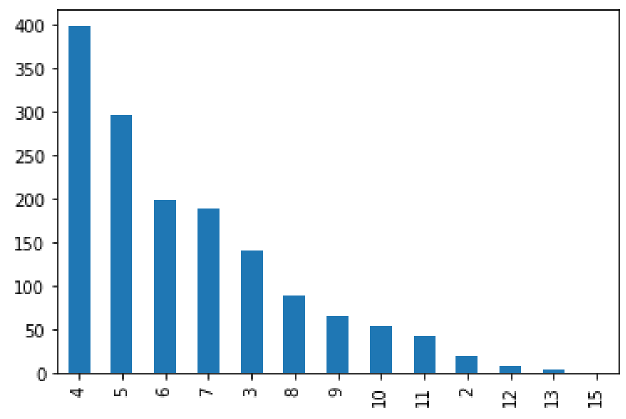


T1

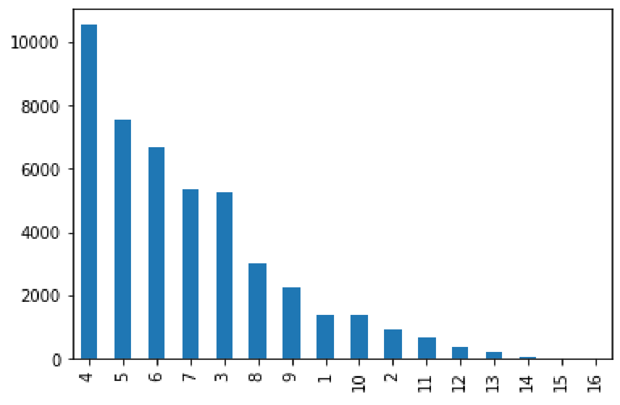


T2

After



T1



T2

Inferences

We infer from the above visualizations that

● The number of words of length 1, 2 and 3 have significantly decreased after the removal of stopwords.

● We can clearly infer that this is due to the removal of stopwords like ’a’ ,‘be’, ‘the’, ‘of’ and ‘and’ which were the highest occurring words before removal.

● Performing POS Tagging

We will now perform the POS Tagging on T1 and T2 using inbuilt functions of **nltk** namely **post_tag()** which uses Penn Treebank tag set to perform POS tagging.

```
def tag_treebank(tokens):
  tagged=nltk.pos_tag(tokens)
  return tagged
```

```
book1_tags = tag_treebank(T1)
book2_tags = tag_treebank(T2)
```

```
print(book1_tags)
```

```
('start', 'NN'), ('project', 'NN'), ('gutenberg', 'NN'), ('ebook', 'NN'),
('traitor', 'NN'), ('choice', 'NN'), ('traitor', 'NN'), ('choice', 'NN'),
('paul', 'NN'), ('fairman', 'NN'), ('kendall', 'NN'), ('difficult',
'JJ'), ('decision', 'NN'), ('make', 'VBP'), ('defy', 'NN'), ('alien',
'NN'), ('clare', 'NN'), ('face', 'NN'), ('horrible', 'JJ'), ('death',
'NN'), ('comply', 'NN'), ('whole', 'JJ'), ('planet', 'NN'), ('must',
'MD'), ('die', 'VB'), ('transcriber', 'NNP'), ('note', 'NN'), ('etext',
'NN'), ('produce', 'VBP'), ('imagination', 'NN'), ('stories', 'NNS'),
('science', 'NN'), ('fantasy', 'NN'), ('august', 'VBP'), ('one', 'CD'),
('thousand', 'CD'), ('nine', 'CD'), ('hundred', 'VBN'), ('extensive',
'JJ'), ('research', 'NN')
```

```
print(book2_tags)
```

```
('start', 'NN'), ('project', 'NN'), ('gutenberg', 'NN'), ('ebook',
'VBP'), ('heir', 'PRP$'), ('heir', 'NN'), ('sydney', 'NN'),
('grier', 'NN'), ('author', 'NN'), ('uncrowned', 'VBD'), ('king',
'VBG'), ('warden', 'JJ'), ('march', 'NN'), ('etc', 'NN'),
('illustrations', 'NNS'), ('george', 'VBP'), ('percy', 'JJ'),
('balkan', 'JJ'), ('series', 'NN'), ('william', 'NN'), ('blackwood',
'NN'), ('sons', 'NNS'), ('edinburgh', 'VBP'), ('london', 'JJ'),
('mcmvi', 'JJ'), ('image', 'NN'), ('caption', 'NN'), ('arm', 'NN'),
('grip', 'NN'), ('one', 'CD'), ('brigands', 'VBZ'), ('trudge',
'NN'), ('silently', 'RB'), ('beside', 'JJ'), ('content', 'NN'),
('de', 'IN'), ('jure', 'NN'), ('ii', 'NN'), ('stock', 'NN'),
('emperors', 'NNS'), ('iii', 'VBP'), ('orient', 'JJ').
```

- **Frequency Distribution of Tags**

  Now, we plot the frequency distribution of tags after POS tagging on T1 and T2. For this we take the help of **Counter()** function from **collections** python library and **FreqDist()** function from nltk python library.

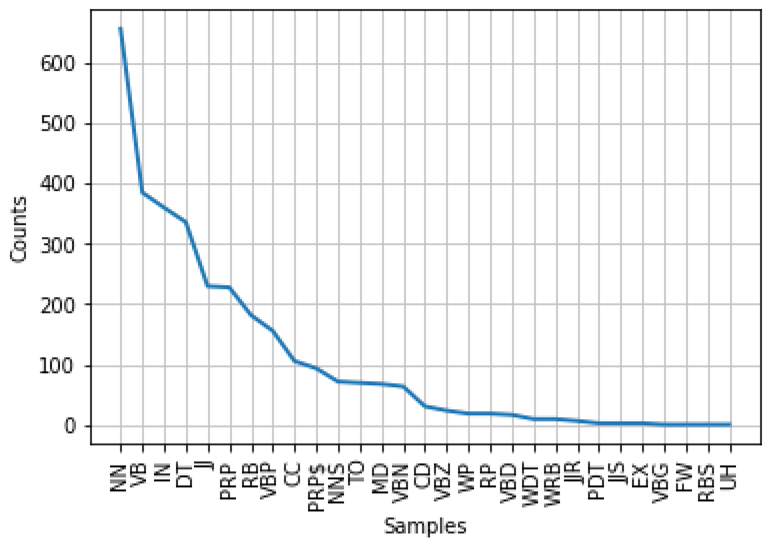
```
from collections import Counter
def get_counts(tags):
  counts = Counter( tag for word,  tag in tags)
  return counts
```

```
def FrequencyDist(tags):
  wfd = nltk.FreqDist(t for (w,t) in tags)
  wfd
  wfd.plot(50)
```

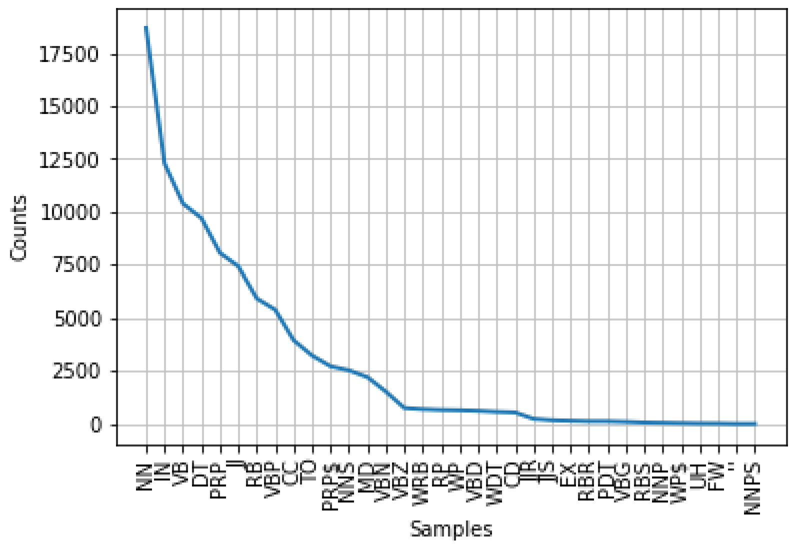
```
book1_pos_count=get_counts(book1_tags)
book2_pos_count=get_counts(book2_tags)
```

```
FrequencyDist(book1_tags)
FrequencyDist(book2_tags)
```

Frequency Distribution of Tags in T1



Frequency Distribution of Tags in T2



Inferences

From the above results we infer that the highest occurring tag is 'NN', and 'Determinant' Tags are on the lower frequency side. This is largely due to the removal of stopwords before POS Tagging.

Conclusion :

We have learnt how to perform Text Preprocessing, Tokenization on Text Data and then answer all the Problems given to us in NLP Project Round 1 with the use of Plots and Visualisations and learnt to draw meaningful inferences from it.