NLP PROJECT
ROUND-1

Github link: https://github.com/Naresh0979/NLP-Project

TEAM NAME: ELITE

Members-

Naresh Kumawat (19UCS046)

Manan Gandhi (19UCS130)

Anurag Garg (19UCS173)

PROJECT REPORT SUBMISSION

Task

1. Import the text from two books. Let’s call it T1 and T2.
2. Perform simple text preprocessing steps and tokenize the text T1 and T2.
3. Analyse the frequency distribution of tokens in T1 and T2 separately.
4. Create a Word Cloud of T1 and T2 using the token you have.
5. Remove the stopwords from T1 and T2 and create a word cloud again.
6. Compare with word clouds before the removal of stopwords.
7. Evaluate the word length and frequency relationship for both T1 and T2.
8. Do PoS Tagging for both T1 and T2 using any of the four tagset studied in the class and get various tags’ distribution.

Library Used

NLTK - Used for tokenizing, lemmatization and removing stopwords.

Language Word Cloud - Used to create word clouds from tokenized data.

Matplotlib- Used to Visualize our text data

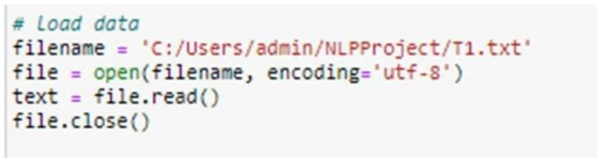
Seaborn -Used to Visualize our text data

Numpy- Used for working with arrays.

Typing-To perform evaluation of the Algorithm in Entity Recognition

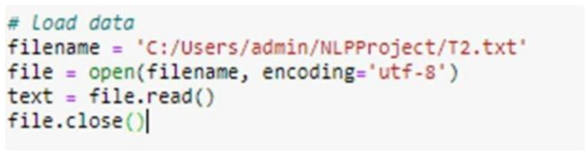
IMPORT THE TEXT T1 AND T2

<u>Code For T1</u>



```
# load data
filename = 'C:/Users/admin/NLPProject/T1.txt'
file = open(filename, encoding='utf-8')
text = file.read()
file.close()
```

<u>Code For T2</u>



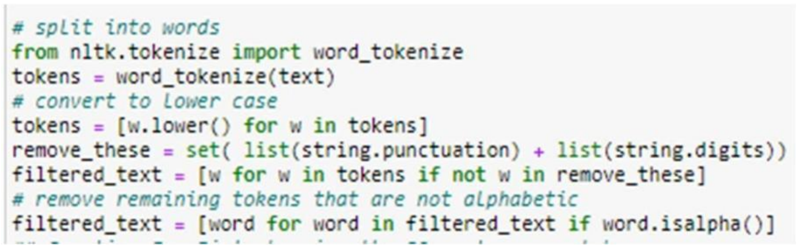
```
# load data
filename = 'C:/Users/admin/NLPProject/T2.txt'
file = open(filename, encoding='utf-8')
text = file.read()
file.close()
```

TEXT PREPROCESSING STEPS

- Here we have used word_tokenize of nltk library for splitting the text into words.
- Then we have converted the upper case tokens to lower case tokens .
- Then we removed the punctuations and digits.
- Then we also have removed the remaining tokens which are not alphabetical.
- Then we exclude stop words for which we have used nltk.corpus

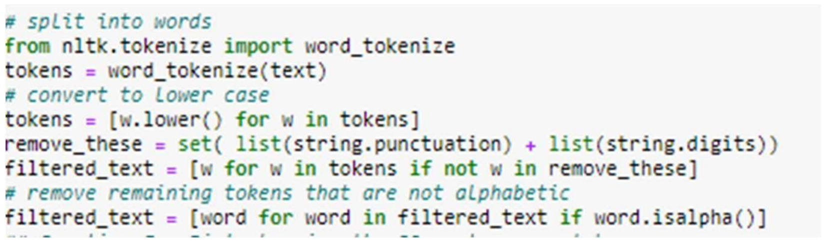
Including stopwords

<u>For T1 Code:</u>



```
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]
```

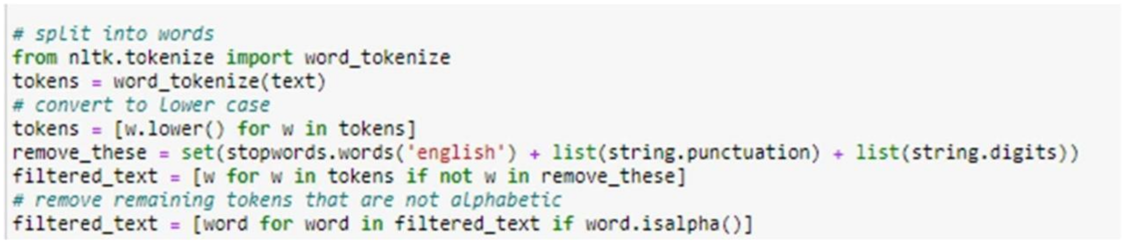
<u>For T2 Code:</u>



```
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]
```

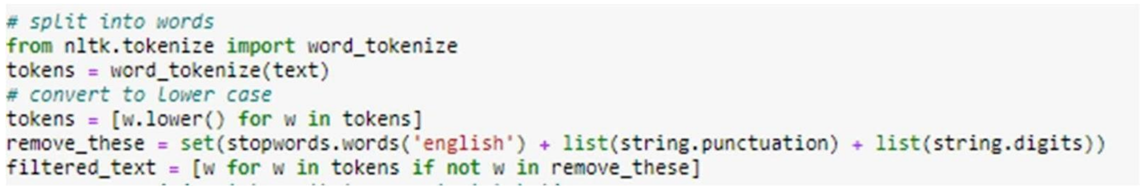
Excluding stopwords

For T1 Code:



```
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]
```

For T2 Code:

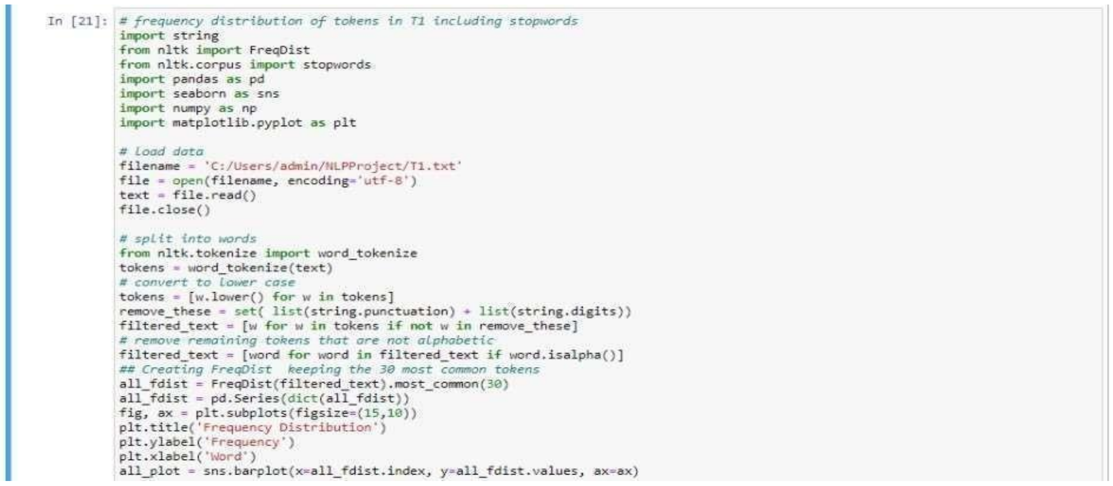


```
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
```

FREQUENCY DISTRIBUTION OF TOKENS

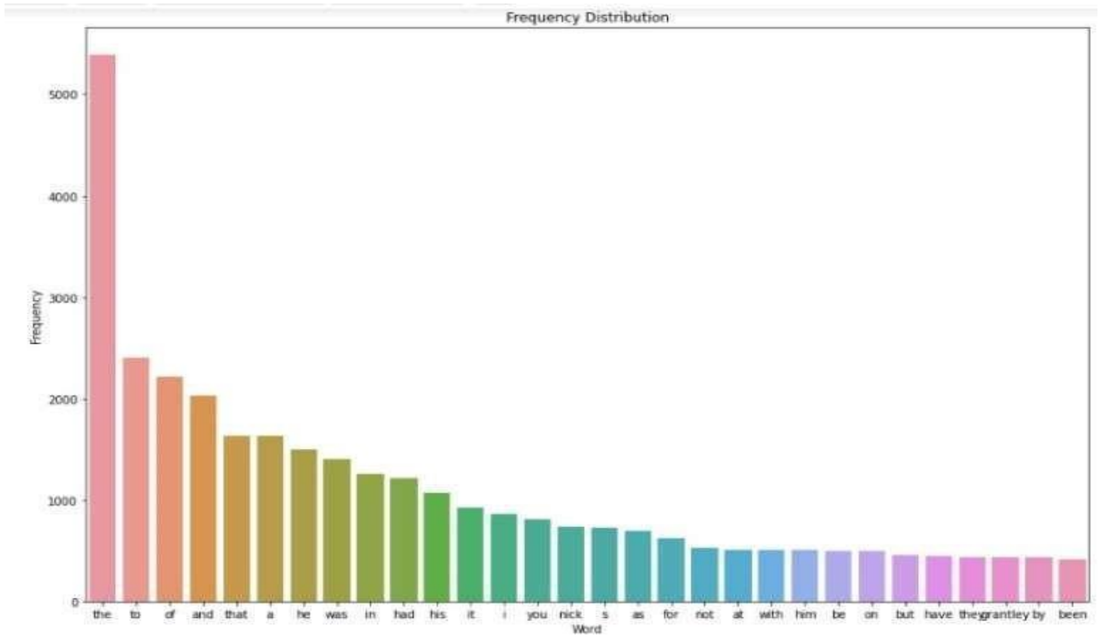
1. We have used the FreqDist function from the nltk library to distribute the 30 most common tokens.
2. For analysing purposes, we have used a bar plot

For T1(with stopwords) Code:

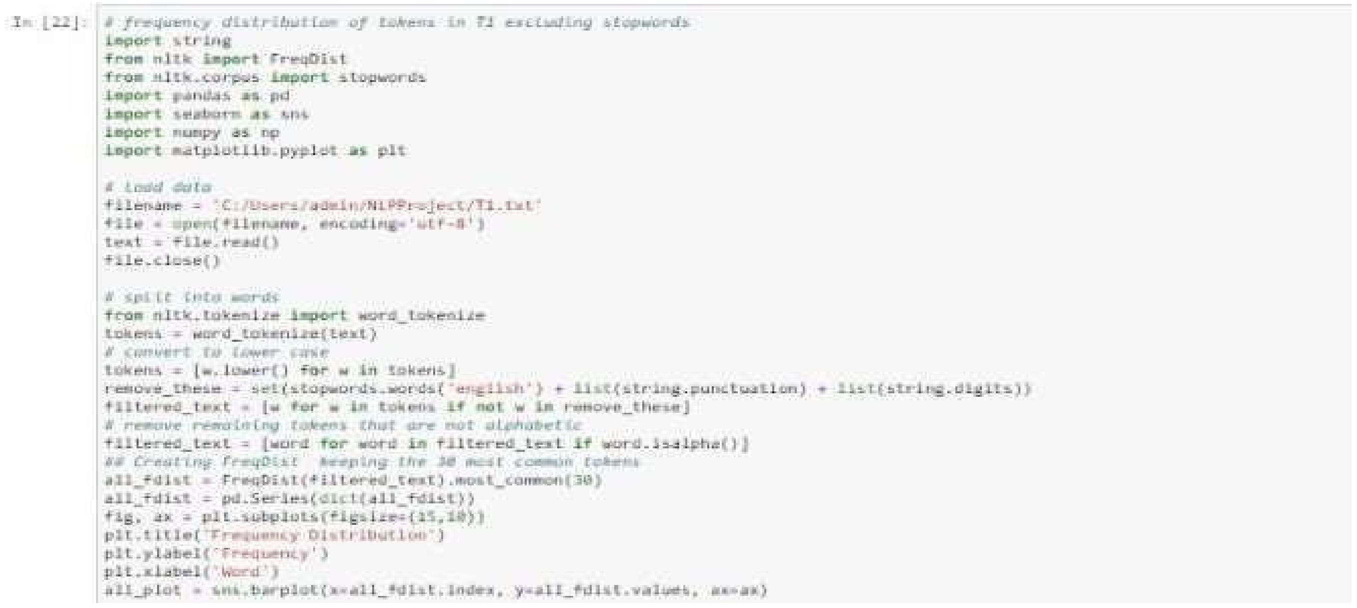


```
In [21]: # frequency distribution of tokens in T1 including stopwords
         import string
         from nltk import FreqDist
         from nltk.corpus import stopwords
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         # load data
         filename = 'C:/Users/admin/NLPProject/T1.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()

         # split into words
         from nltk.tokenize import word_tokenize
         tokens = word_tokenize(text)
         # convert to lower case
         tokens = [w.lower() for w in tokens]
         remove_these = set( list(string.punctuation) + list(string.digits))
         filtered_text = [w for w in tokens if not w in remove_these]
         # remove remaining tokens that are not alphabetic
         filtered_text = [word for word in filtered_text if word.isalpha()]
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(filtered_text).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

Output:(graphical representation)

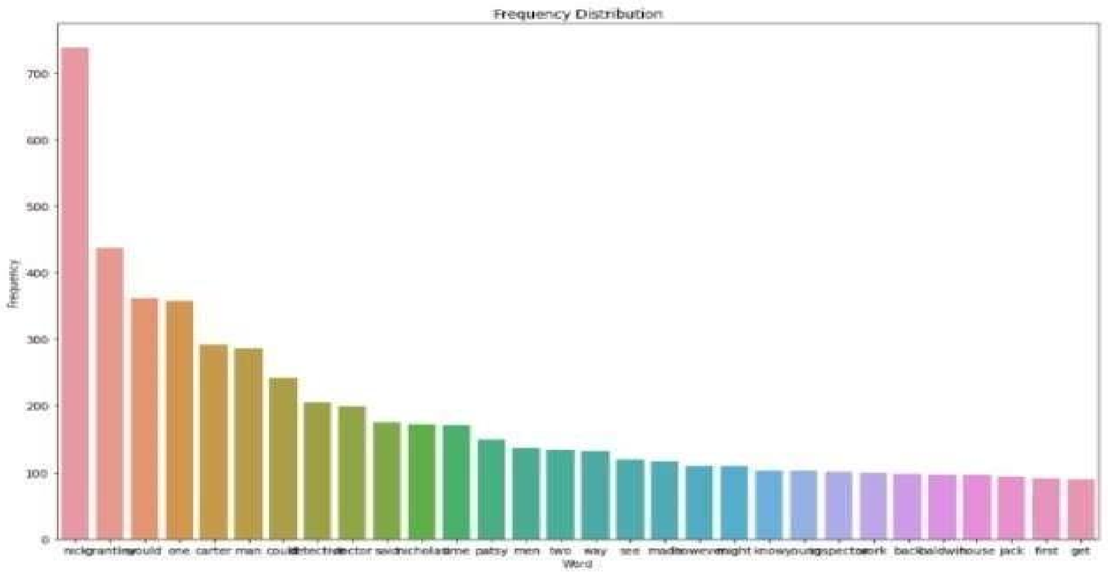


For T1 (without stopwords) Code:

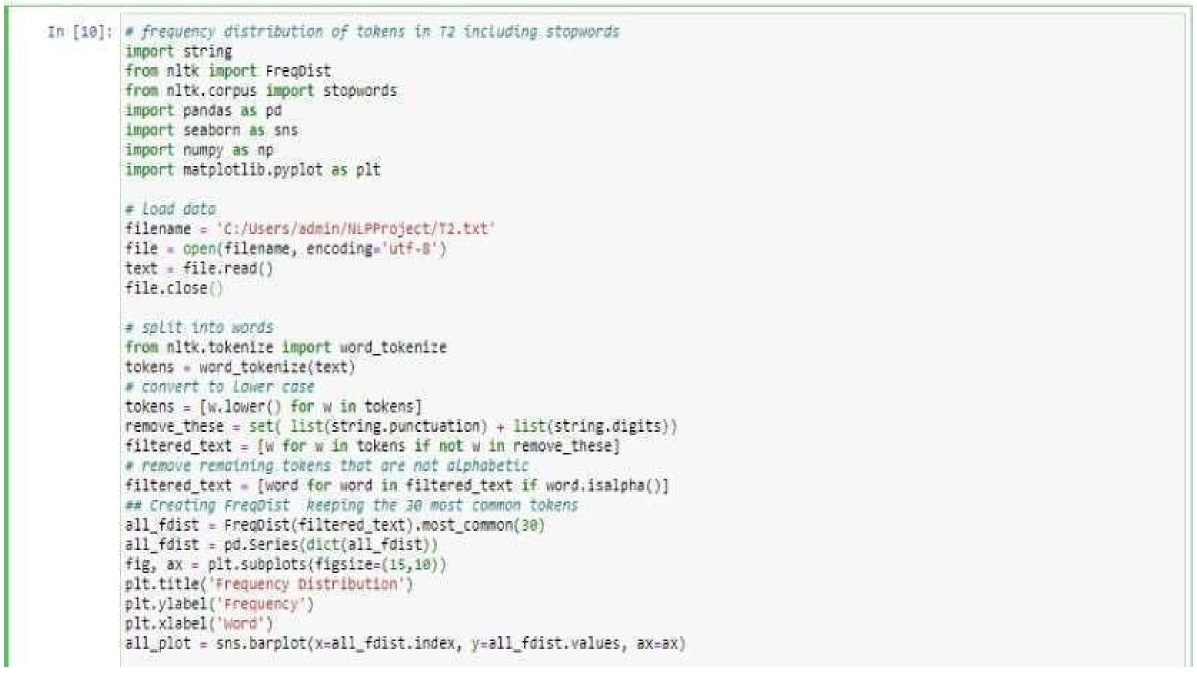


```
In [22]: # frequency distribution of tokens in T1 excluding stopwords
         import string
         from nltk import FreqDist
         from nltk.corpus import stopwords
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         # load data
         filename = 'C:/Users/admin/NLPProject/T1.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()

         # split into words
         from nltk.tokenize import word_tokenize
         tokens = word_tokenize(text)
         # convert to lower case
         tokens = [w.lower() for w in tokens]
         remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
         filtered_text = [w for w in tokens if not w in remove_these]
         # remove remaining tokens that are not alphabetic
         filtered_text = [word for word in filtered_text if word.isalpha()]
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(filtered_text).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

Output:(graphical representation)

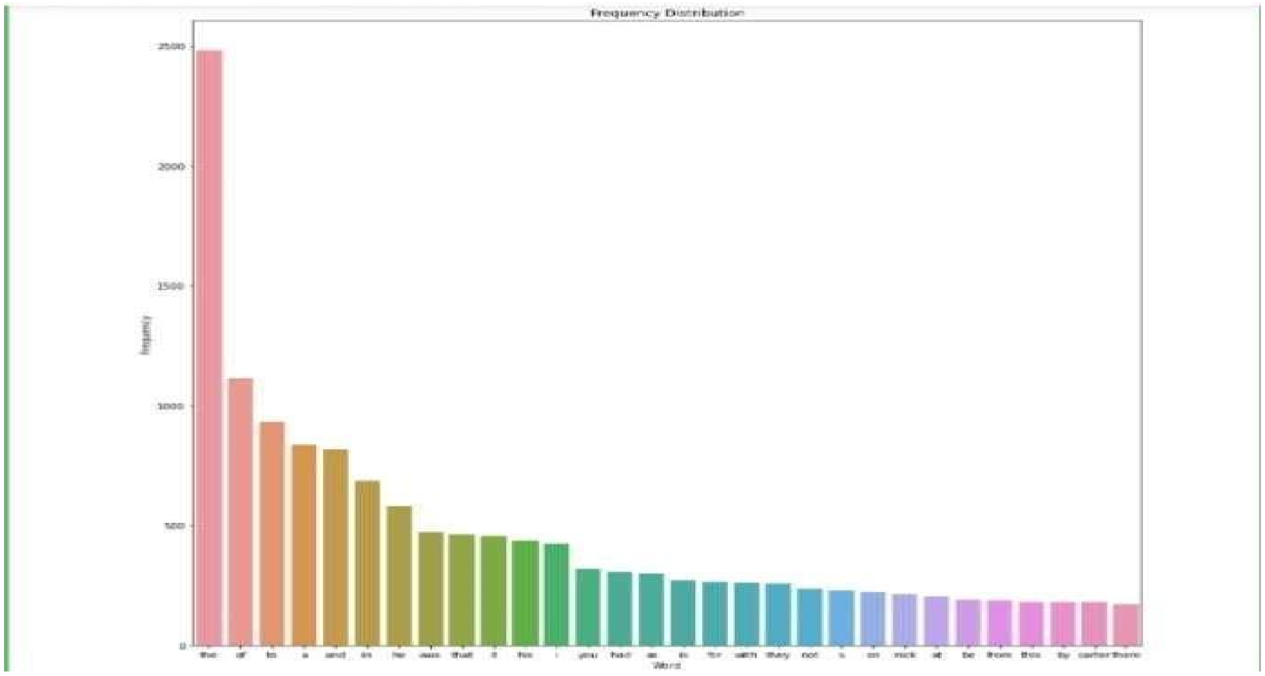


For T2(with stopwords) Code:

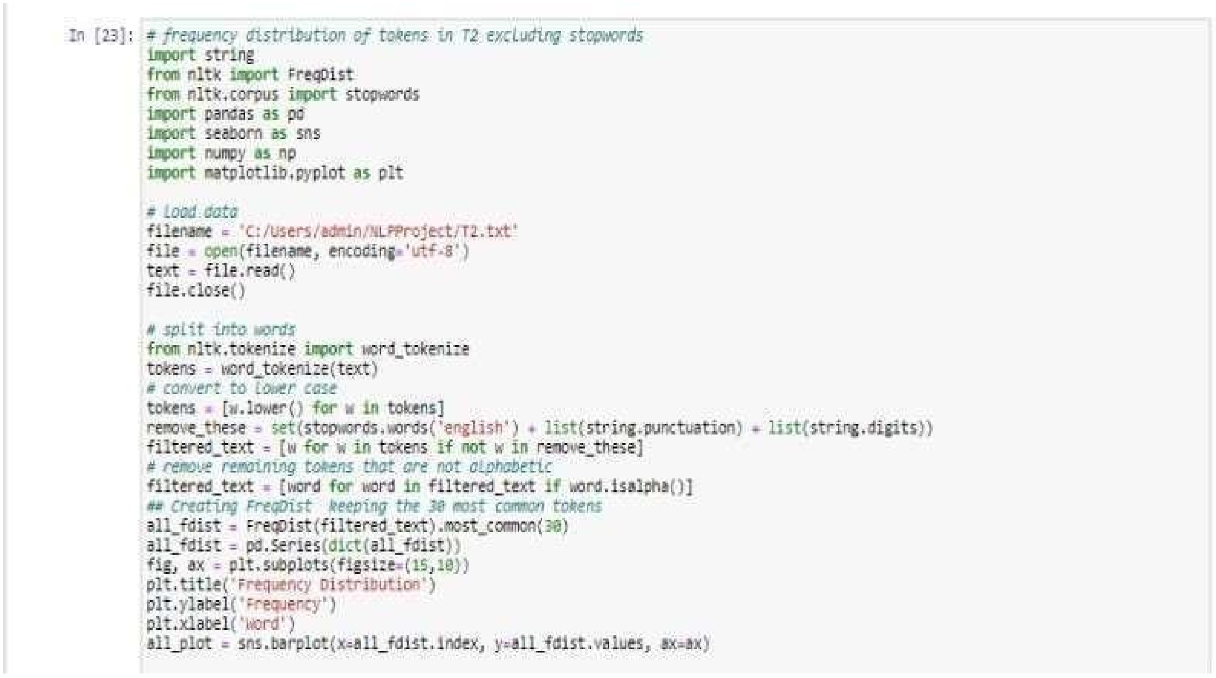


```
In [10]: # frequency distribution of tokens in T2 including stopwords
         import string
         from nltk import FreqDist
         from nltk.corpus import stopwords
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         # load data
         filename = 'C:/Users/admin/NLPProject/T2.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()

         # split into words
         from nltk.tokenize import word_tokenize
         tokens = word_tokenize(text)
         # convert to lower case
         tokens = [w.lower() for w in tokens]
         remove_these = set( list(string.punctuation) + list(string.digits))
         filtered_text = [w for w in tokens if not w in remove_these]
         # remove remaining tokens that are not alphabetic
         filtered_text = [word for word in filtered_text if word.isalpha()]
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(filtered_text).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

Output:(graphical representation)

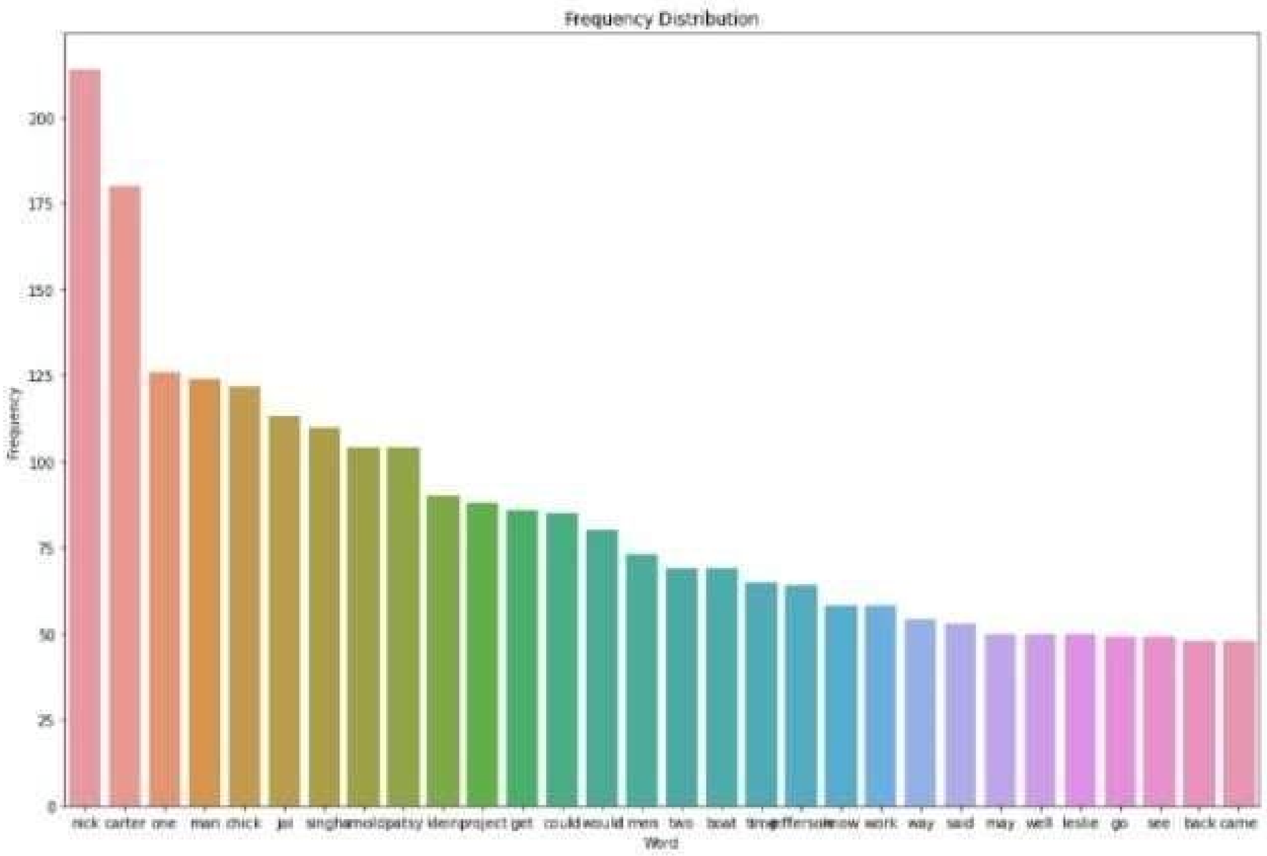


For T2 (without stopwords) Code:



```
In [23]: # frequency distribution of tokens in T2 excluding stopwords
         import string
         from nltk import FreqDist
         from nltk.corpus import stopwords
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         # Load data
         filename = 'C:/Users/admin/NLPProject/T2.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()

         # split into words
         from nltk.tokenize import word_tokenize
         tokens = word_tokenize(text)
         # convert to lower case
         tokens = [w.lower() for w in tokens]
         remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
         filtered_text = [w for w in tokens if not w in remove_these]
         # remove remaining tokens that are not alphabetic
         filtered_text = [word for word in filtered_text if word.isalpha()]
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(filtered_text).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

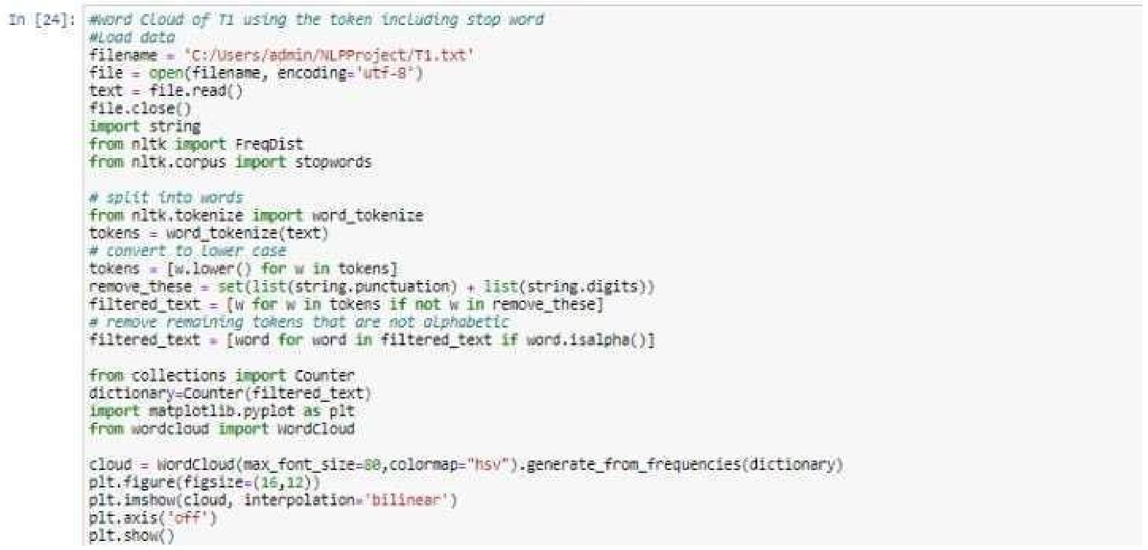
Output:(graphical representation)



Word cloud of Text using Token

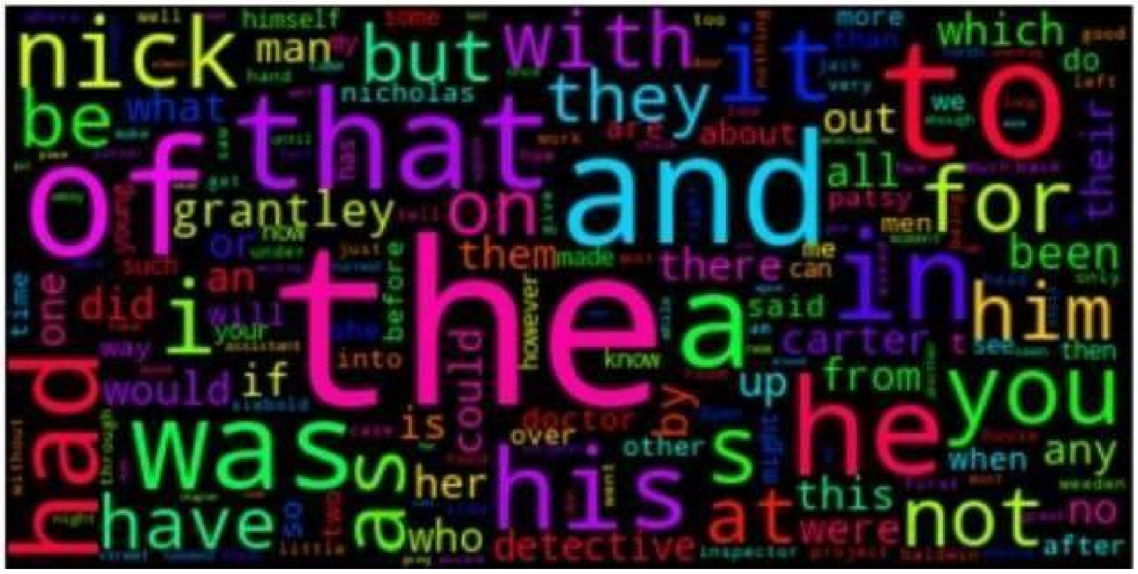
We have used the cloud function from the WordCloud library.

For T1(including stopwords) Code:

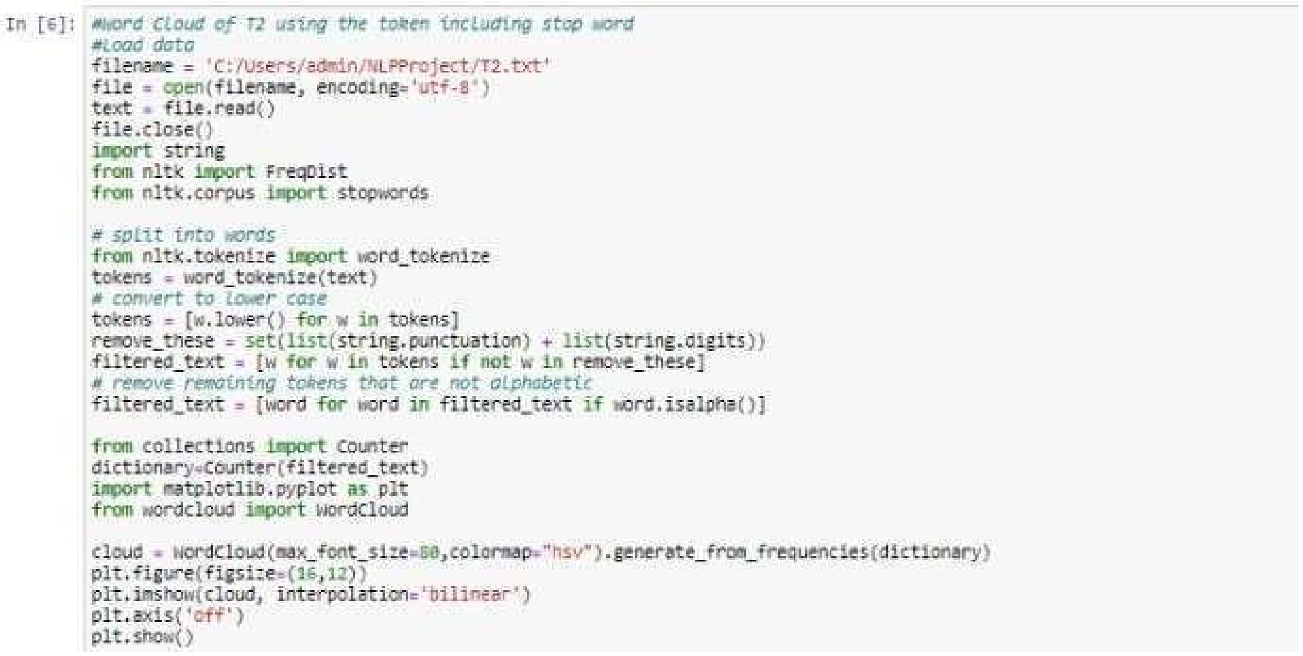


```
In [24]: #Word Cloud of T1 using the token including stop word
         #Load data
         filename = 'C:/Users/admin/NLPProject/T1.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()
         import string
         from nltk import FreqDist
         from nltk.corpus import stopwords

         # split into words
         from nltk.tokenize import word_tokenize
         tokens = word_tokenize(text)
         # convert to lower case
         tokens = [w.lower() for w in tokens]
         remove_these = set(list(string.punctuation) + list(string.digits))
         filtered_text = [w for w in tokens if not w in remove_these]
         # remove remaining tokens that are not alphabetic
         filtered_text = [word for word in filtered_text if word.isalpha()]

         from collections import Counter
         dictionary=Counter(filtered_text)
         import matplotlib.pyplot as plt
         from wordcloud import WordCloud

         cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictionary)
         plt.figure(figsize=(16,12))
         plt.imshow(cloud, interpolation='bilinear')
         plt.axis('off')
         plt.show()
```

Output:

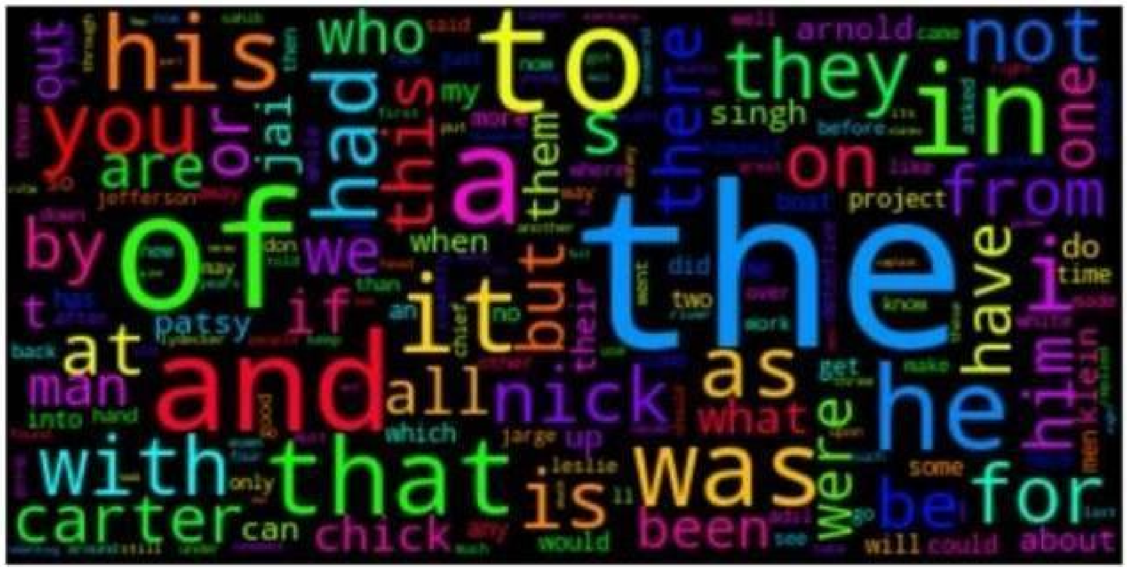


For T2(with stopwords) Code:



```
In [6]: #Word Cloud of T2 using the token including stop word
        #Load data
        filename = 'C:/Users/admin/NLPProject/T2.txt'
        file = open(filename, encoding='utf-8')
        text = file.read()
        file.close()
        import string
        from nltk import FreqDist
        from nltk.corpus import stopwords

        # split into words
        from nltk.tokenize import word_tokenize
        tokens = word_tokenize(text)
        # convert to lower case
        tokens = [w.lower() for w in tokens]
        remove_these = set(list(string.punctuation) + list(string.digits))
        filtered_text = [w for w in tokens if not w in remove_these]
        # remove remaining tokens that are not alphabetic
        filtered_text = [word for word in filtered_text if word.isalpha()]

        from collections import Counter
        dictionary=Counter(filtered_text)
        import matplotlib.pyplot as plt
        from wordcloud import WordCloud

        cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictionary)
        plt.figure(figsize=(16,12))
        plt.imshow(cloud, interpolation='bilinear')
        plt.axis('off')
        plt.show()
```

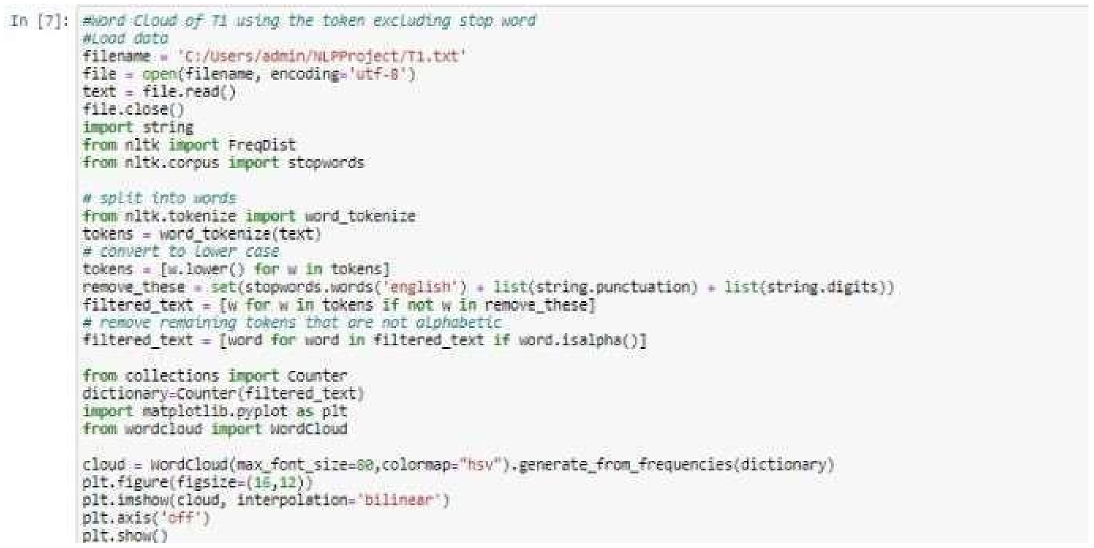
<u>Output:</u>



Inferences:

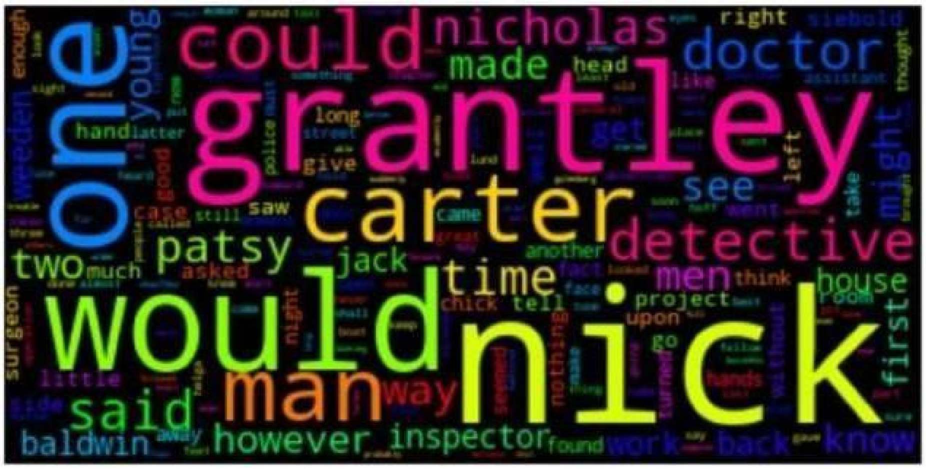
- Words like ‘of’, ‘to’ and ‘the’ are the most frequently used words in T1
- Words like ‘of’, ‘and’, ‘be’, and ‘the’ are frequently used words in T2
- These words do not contribute to the meaning of the sentence and are mostly useless for us
- These words are known as ‘stopwords’, and we need to get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

<u>For T1(without stopwords) Code:</u>

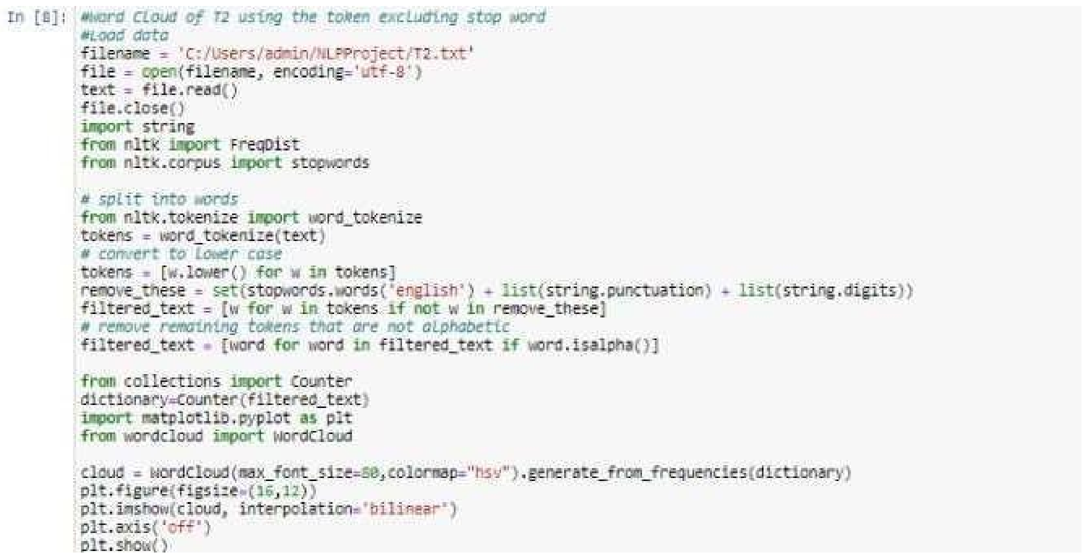


```
In [7]: #Word Cloud of T1 using the token excluding stop word
        #Load data
        filename = 'C:/Users/admin/NLPProject/T1.txt'
        file = open(filename, encoding='utf-8')
        text = file.read()
        file.close()
        import string
        from nltk import FreqDist
        from nltk.corpus import stopwords

        # split into words
        from nltk.tokenize import word_tokenize
        tokens = word_tokenize(text)
        # convert to lower case
        tokens = [w.lower() for w in tokens]
        remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
        filtered_text = [w for w in tokens if not w in remove_these]
        # remove remaining tokens that are not alphabetic
        filtered_text = [word for word in filtered_text if word.isalpha()]

        from collections import Counter
        dictionary=Counter(filtered_text)
        import matplotlib.pyplot as plt
        from wordcloud import WordCloud

        cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictionary)
        plt.figure(figsize=(16,12))
        plt.imshow(cloud, interpolation='bilinear')
        plt.axis('off')
        plt.show()
```

Output:

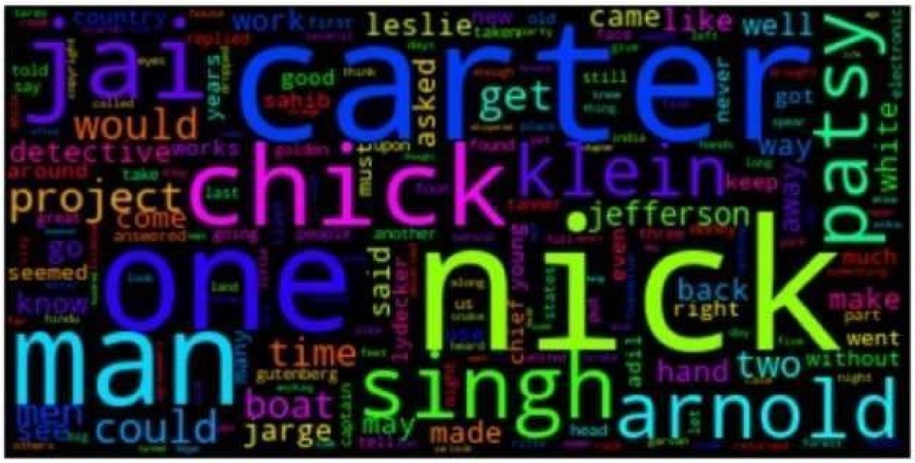


For T2(without stopwords) Code:



```
In [8]: #Word Cloud of T2 using the token excluding stop word
        #Load data
        filename = 'C:/Users/admin/NLPProject/T2.txt'
        file = open(filename, encoding='utf-8')
        text = file.read()
        file.close()
        import string
        from nltk import FreqDist
        from nltk.corpus import stopwords

        # split into words
        from nltk.tokenize import word_tokenize
        tokens = word_tokenize(text)
        # convert to lower case
        tokens = [w.lower() for w in tokens]
        remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
        filtered_text = [w for w in tokens if not w in remove_these]
        # remove remaining tokens that are not alphabetic
        filtered_text = [word for word in filtered_text if word.isalpha()]

        from collections import Counter
        dictionary=Counter(filtered_text)
        import matplotlib.pyplot as plt
        from wordcloud import WordCloud

        cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictionary)
        plt.figure(figsize=(16,12))
        plt.imshow(cloud, interpolation='bilinear')
        plt.axis('off')
        plt.show()
```

Output:

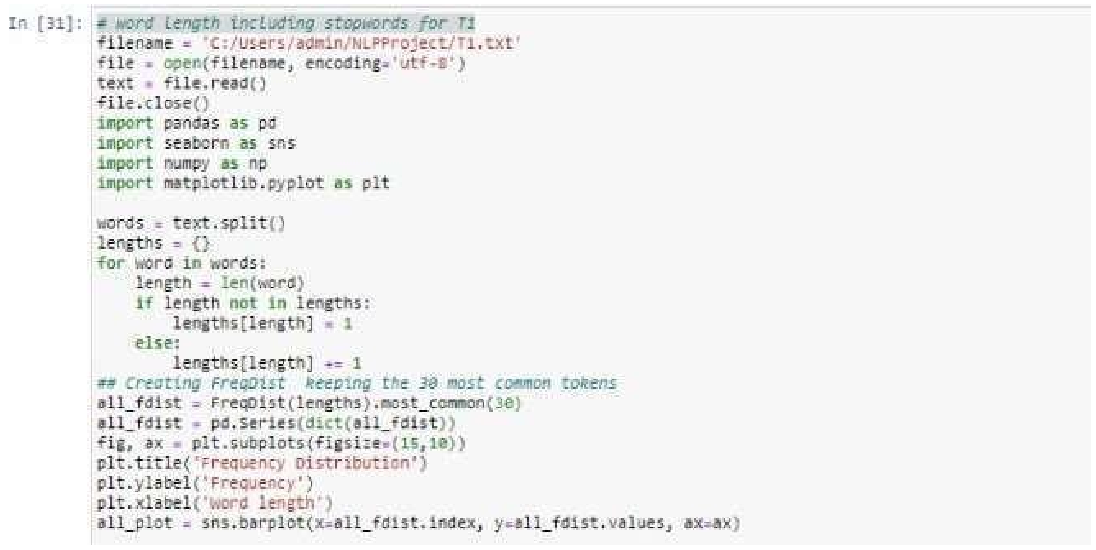


Inferences:

On excluding stop words, we get a meaningful word cloud as stop words mostly have higher frequency due to which our result might get affected. Now most of the terms that are of no meaning to us have been removed. We have gotten rid of ‘stopwords’ and can draw meaningful conclusions from the data.

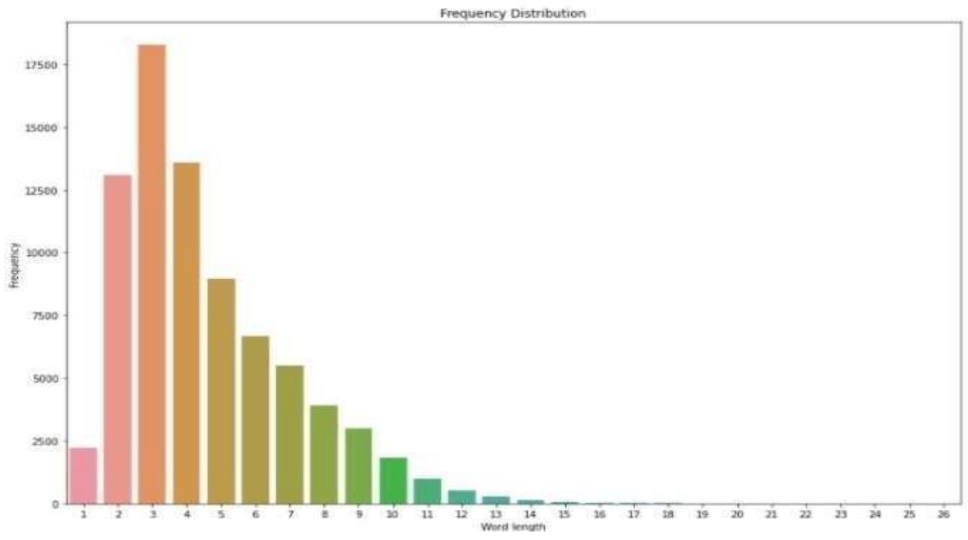
Relation between word length and frequency for Text

<u>For T1(including stopwords) Code:</u>

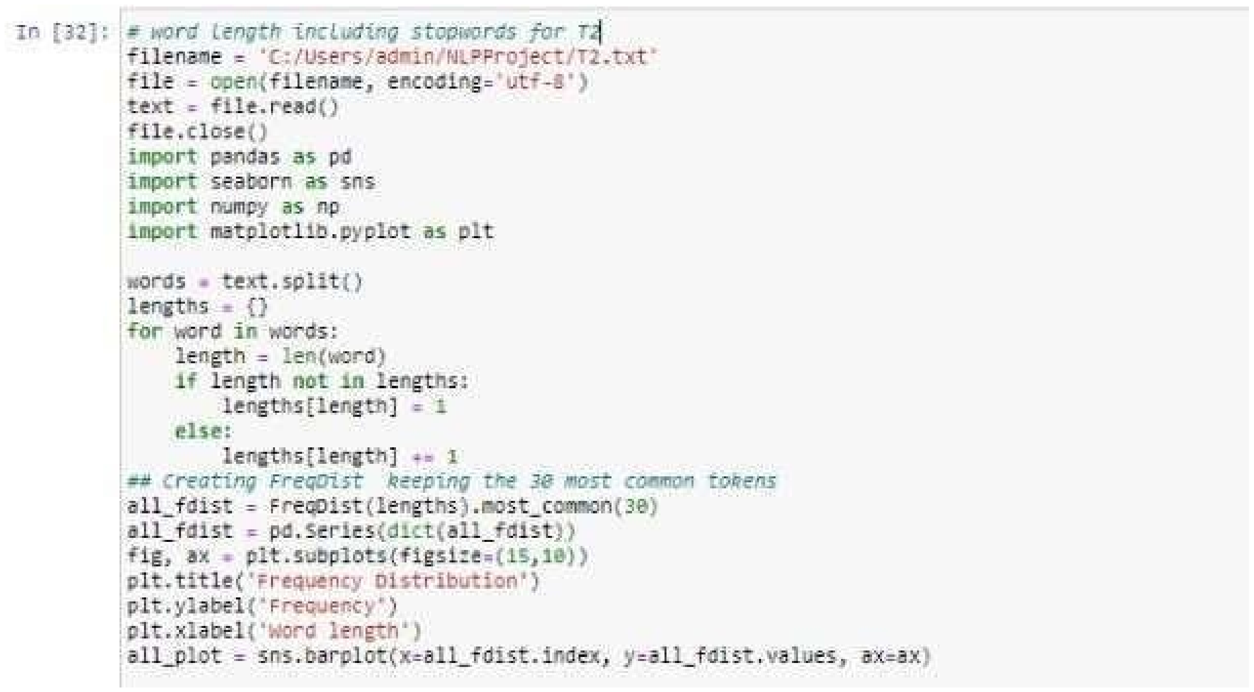


```
In [31]: # word length including stopwords for T1
         filename = 'C:/Users/admin/NLPProject/T1.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         words = text.split()
         lengths = {}
         for word in words:
             length = len(word)
             if length not in lengths:
                 lengths[length] = 1
             else:
                 lengths[length] += 1
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(lengths).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word length')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

<u>Output:(graphical representation)</u>

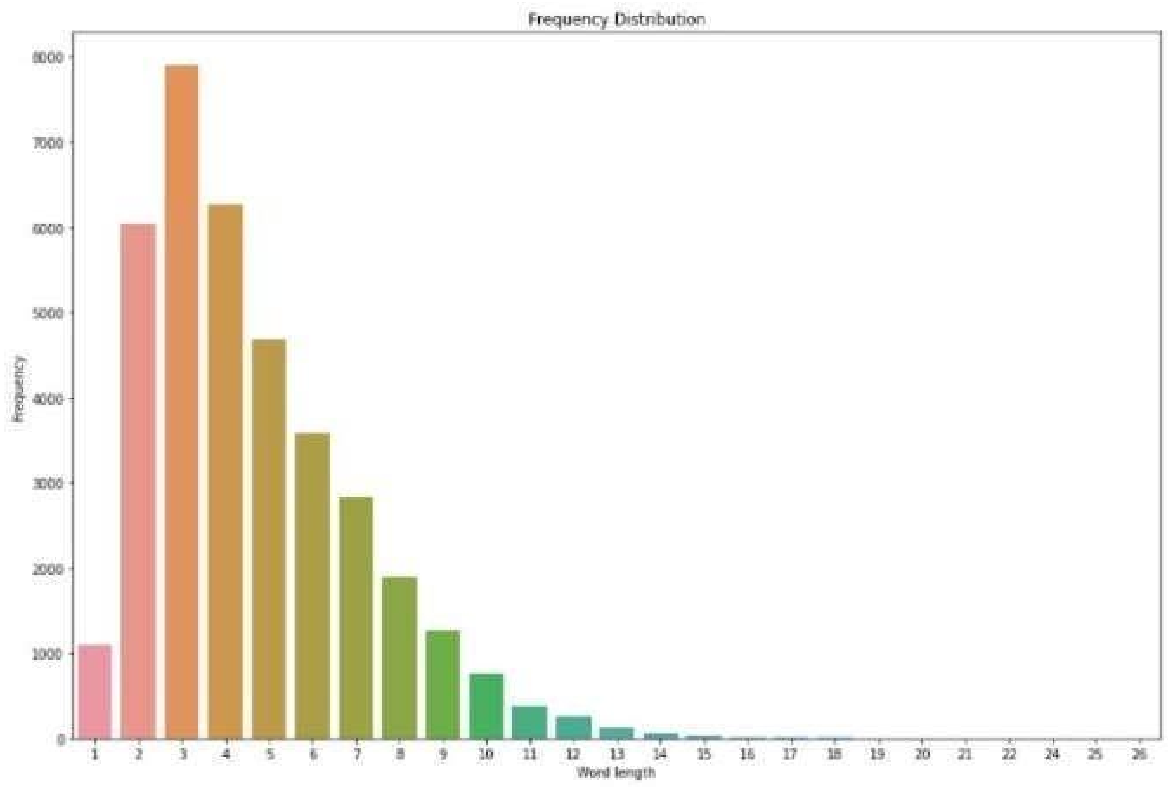


For T2(including stop words) Code:

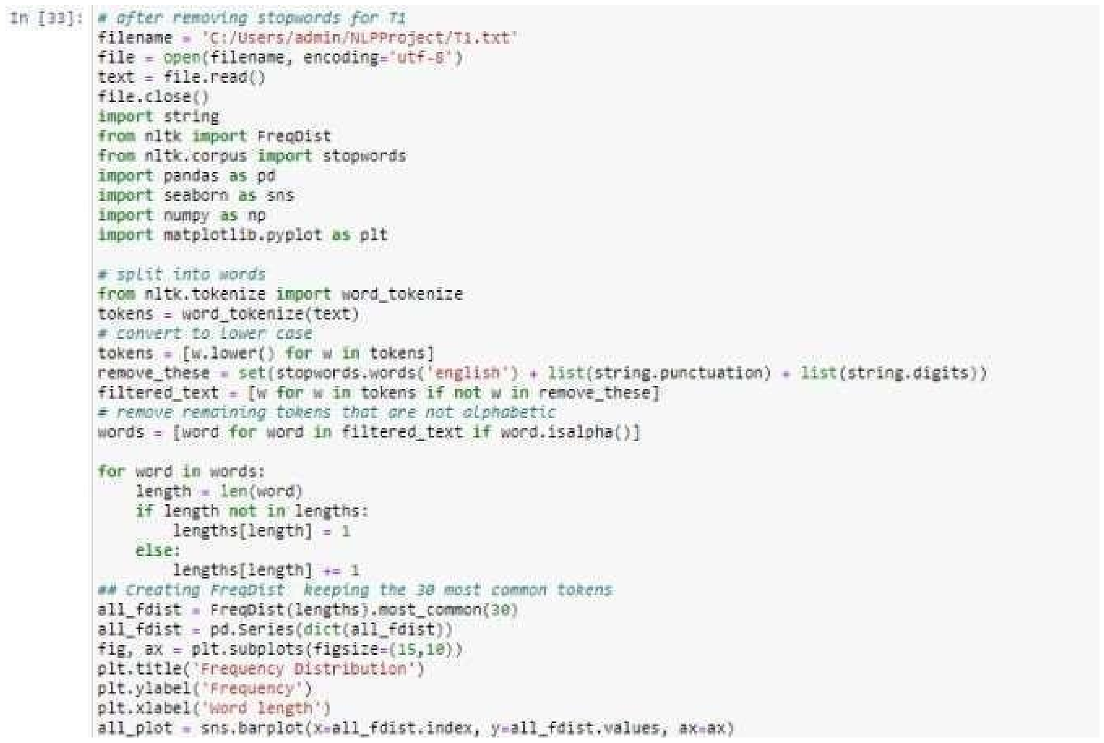


```
In [32]: # word length including stopwords for T2
         filename = 'C:/Users/admin/NLPProject/T2.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         words = text.split()
         lengths = {}
         for word in words:
             length = len(word)
             if length not in lengths:
                 lengths[length] = 1
             else:
                 lengths[length] += 1
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(lengths).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word length')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

Output:(graphical representation)

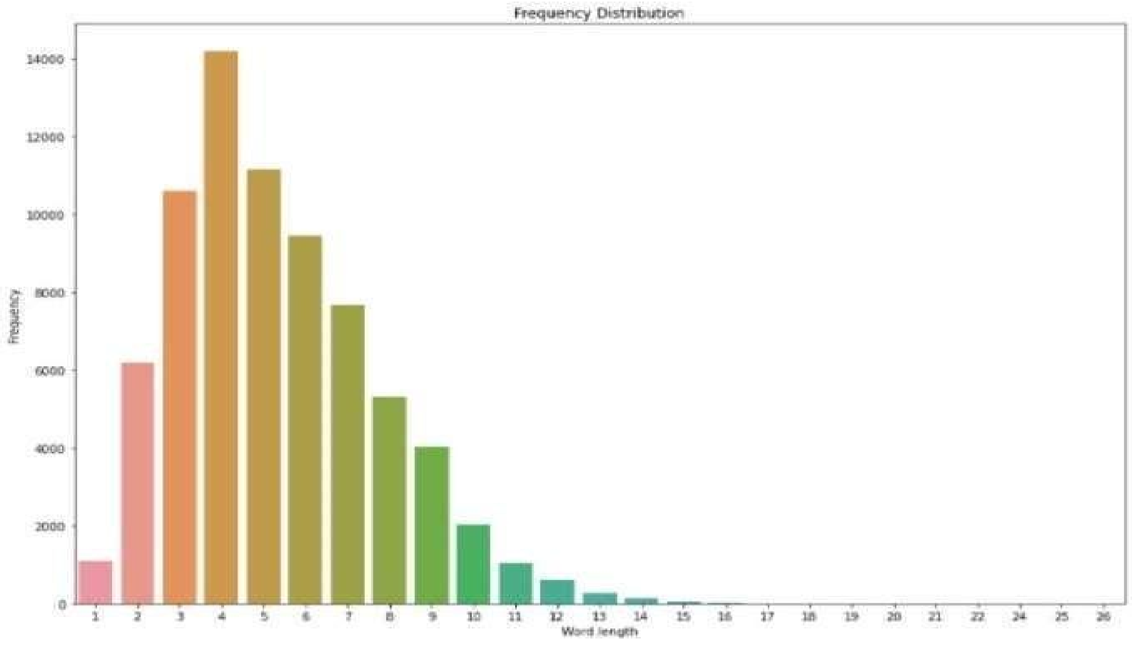


For T1(without stopwords) Code:

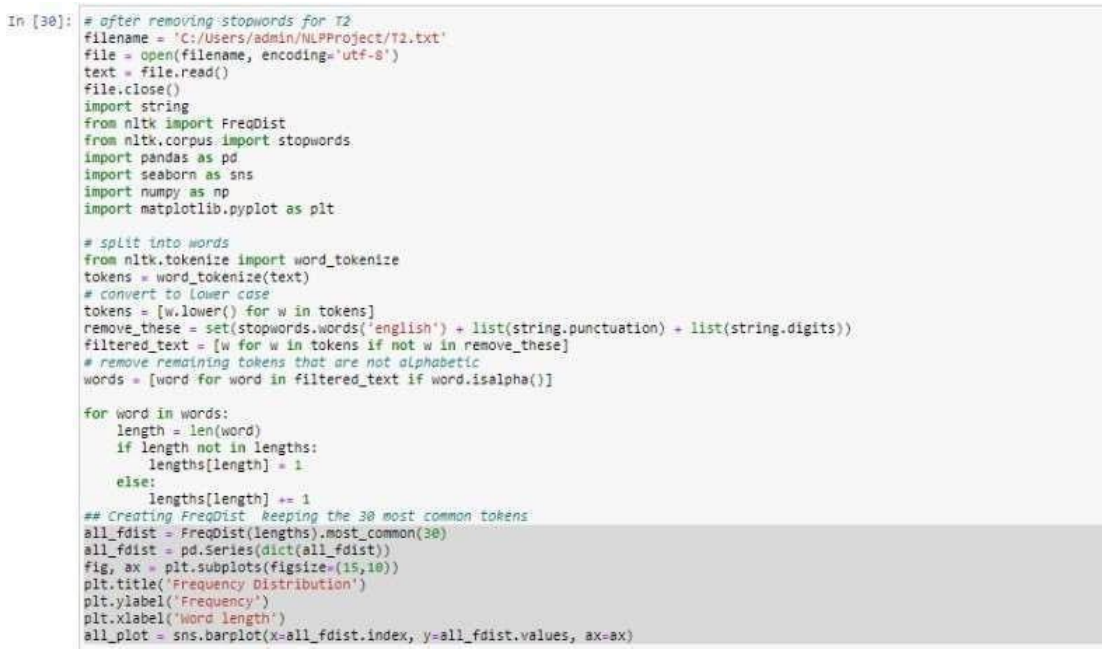


```
In [33]: # after removing stopwords for T1
         filename = 'C:/Users/admin/NLPProject/T1.txt'
         file = open(filename, encoding='utf-8')
         text = file.read()
         file.close()
         import string
         from nltk import FreqDist
         from nltk.corpus import stopwords
         import pandas as pd
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt

         # split into words
         from nltk.tokenize import word_tokenize
         tokens = word_tokenize(text)
         # convert to lower case
         tokens = [w.lower() for w in tokens]
         remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
         filtered_text = [w for w in tokens if not w in remove_these]
         # remove remaining tokens that are not alphabetic
         words = [word for word in filtered_text if word.isalpha()]

         for word in words:
             length = len(word)
             if length not in lengths:
                 lengths[length] = 1
             else:
                 lengths[length] += 1
         ## Creating FreqDist  keeping the 30 most common tokens
         all_fdist = FreqDist(lengths).most_common(30)
         all_fdist = pd.Series(dict(all_fdist))
         fig, ax = plt.subplots(figsize=(15,10))
         plt.title('Frequency Distribution')
         plt.ylabel('Frequency')
         plt.xlabel('Word length')
         all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

Output:(graphical representation)

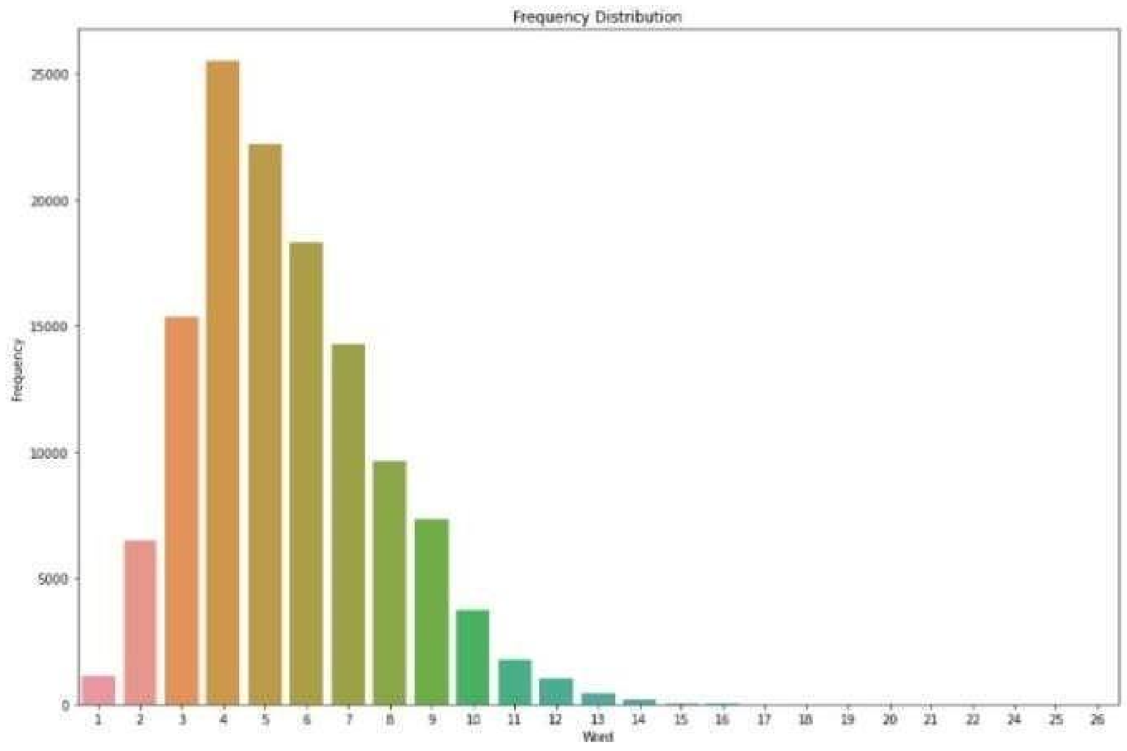


For T2(without stopwords) Code:



```
In [30]: # after removing stopwords for T2
filename = 'C:/Users/admin/NLPProject/T2.txt'
file = open(filename, encoding='utf-8')
text = file.read()
file.close()
import string
from nltk import FreqDist
from nltk.corpus import stopwords
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
words = [word for word in filtered_text if word.isalpha()]

for word in words:
    length = len(word)
    if length not in lengths:
        lengths[length] = 1
    else:
        lengths[length] += 1
## Creating FreqDist  keeping the 30 most common tokens
all_fdist = FreqDist(lengths).most_common(30)
all_fdist = pd.Series(dict(all_fdist))
fig, ax = plt.subplots(figsize=(15,10))
plt.title('Frequency Distribution')
plt.ylabel('Frequency')
plt.xlabel('Word length')
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```

Output:(graphical representation)



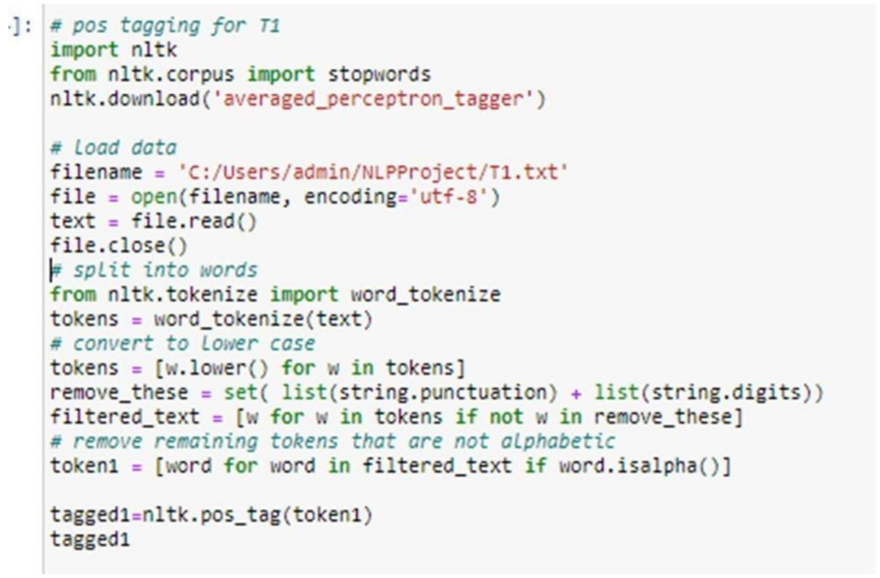
Inferences:

- The number of words of lengths 2 and 3 has significantly decreased after removing stopwords.
- There is a general trend that the highest number of words lies in the length range 3-6.
- We can infer that this is due to removing stop words like ‘be’, ‘the’, ‘of’ and ‘and’, which were the highest occurring words before removal.

POS TAGGING

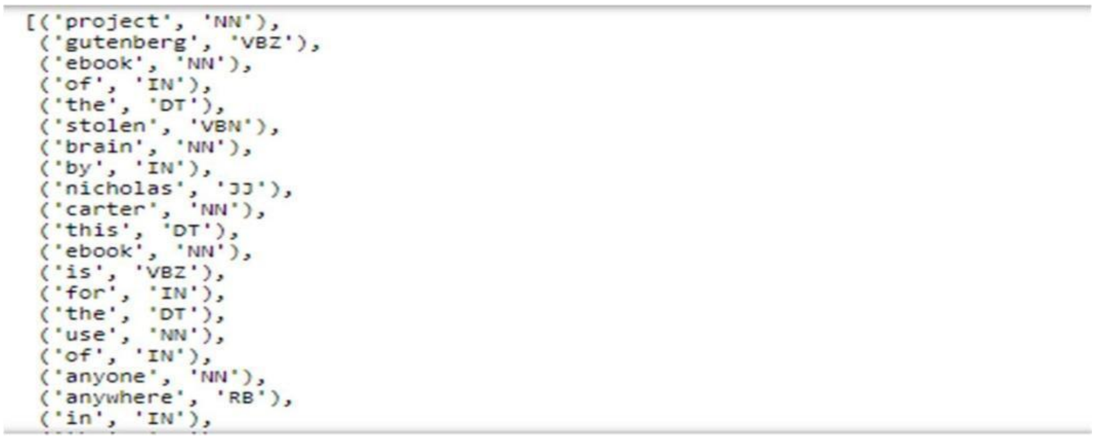
Now the main task is to give each of the tokens their tags. We have used ‘averaged_perceptron_tagger’ for POS tagging of the tickets. We have downloaded the same from nltk.

For T1 Code:



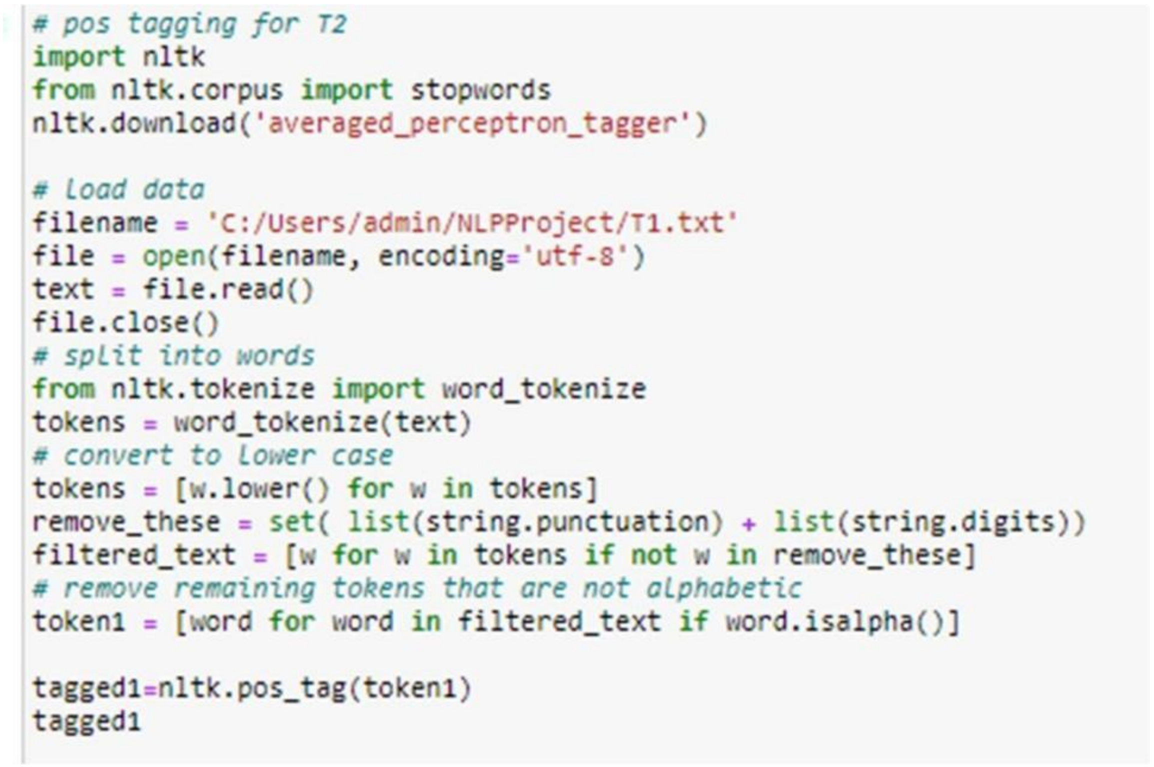
```
]: # pos tagging for T1
import nltk
from nltk.corpus import stopwords
nltk.download('averaged_perceptron_tagger')

# load data
filename = 'C:/Users/admin/NLPProject/T1.txt'
file = open(filename, encoding='utf-8')
text = file.read()
file.close()
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
token1 = [word for word in filtered_text if word.isalpha()]

tagged1=nltk.pos_tag(token1)
tagged1
```

Output:



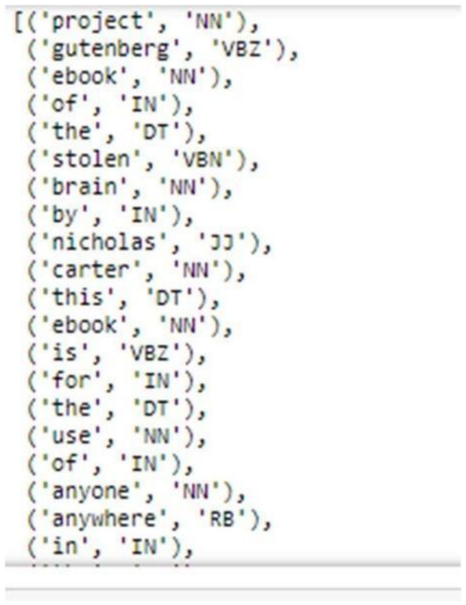
```
[('project', 'NN'),
 ('gutenberg', 'VBZ'),
 ('ebook', 'NN'),
 ('of', 'IN'),
 ('the', 'DT'),
 ('stolen', 'VBN'),
 ('brain', 'NN'),
 ('by', 'IN'),
 ('nicholas', 'JJ'),
 ('carter', 'NN'),
 ('this', 'DT'),
 ('ebook', 'NN'),
 ('is', 'VBZ'),
 ('for', 'IN'),
 ('the', 'DT'),
 ('use', 'NN'),
 ('of', 'IN'),
 ('anyone', 'NN'),
 ('anywhere', 'RB'),
 ('in', 'IN'),
```

For T2 code :



```
# pos tagging for T2
import nltk
from nltk.corpus import stopwords
nltk.download('averaged_perceptron_tagger')

# Load data
filename = 'C:/Users/admin/NLPProject/T1.txt'
file = open(filename, encoding='utf-8')
text = file.read()
file.close()
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
token1 = [word for word in filtered_text if word.isalpha()]

tagged1=nltk.pos_tag(token1)
tagged1
```

Output:



```
[('project', 'NN'),
 ('gutenberg', 'VBZ'),
 ('ebook', 'NN'),
 ('of', 'IN'),
 ('the', 'DT'),
 ('stolen', 'VBN'),
 ('brain', 'NN'),
 ('by', 'IN'),
 ('nicholas', 'JJ'),
 ('carter', 'NN'),
 ('this', 'DT'),
 ('ebook', 'NN'),
 ('is', 'VBZ'),
 ('for', 'IN'),
 ('the', 'DT'),
 ('use', 'NN'),
 ('of', 'IN'),
 ('anyone', 'NN'),
 ('anywhere', 'RB'),
 ('in', 'IN'),
```

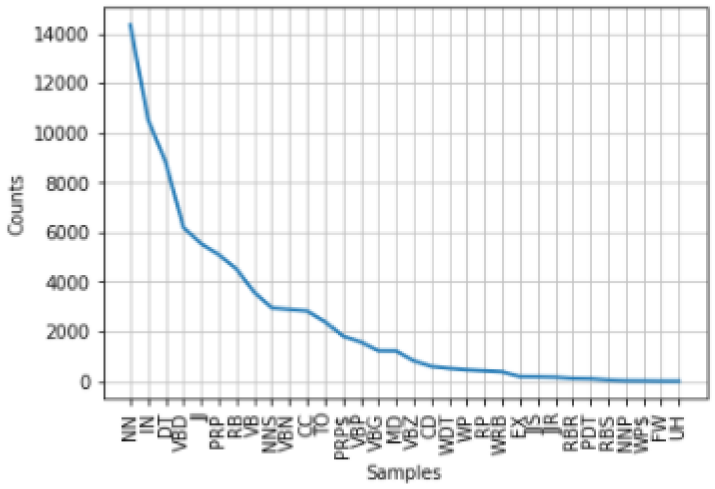
● Frequency Distribution of Tags

For T1 code:

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

```
FrequencyDist(tagged1)
```

Output:

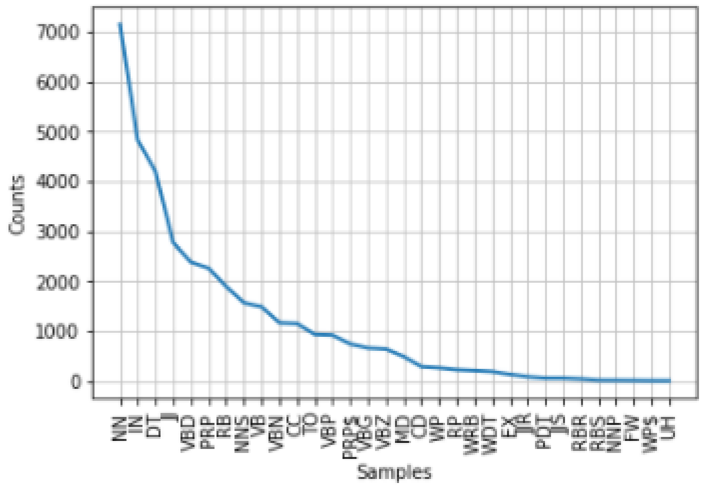


For T1 code:

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

```
FrequencyDist(tagged2)
```

Output:



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 how to Create word clouds. We have solved 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.

NLP PROJECT ROUND 2

Tasks

First Part:
1. Find the nouns and verbs in both the novels. Get the immediate categories (parent) that these words fall under in the WordNet.
2. Get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same for each novel

Second Part:
1. Recognise all persons, locations, organisations in the book.
For this, you have to do two steps:
(1) First, recognise all the entity
(2) Recognise all entity types. Use performance measures to measure the performance of the method used. For evaluation, you take a considerable amount of random passages from the novel, do manual labelling and then compare your result with it. Present the accuracy with the F1 score here.

Third Part:
1. Create TF-IDF vectors for all books, find the cosine similarity between them, and find which two books are more similar.
2. Do lemmatization of the books and recreate the TF-IDF vectors for all the books and find the cosine similarity of each pair of readers.

Library Used

NLTK - Used for tokenizing, lemmatization and removing stopwords.

Language Word Cloud - Used to create word clouds from tokenized data.

Matplotlib- Used to Visualize our text data

Seaborn -Used to Visualize our text data

Numpy- Used for working with arrays.

Typing-To perform evaluation of the Algorithm in Entity Recognition

Problem Statement And Inferences:

First part:

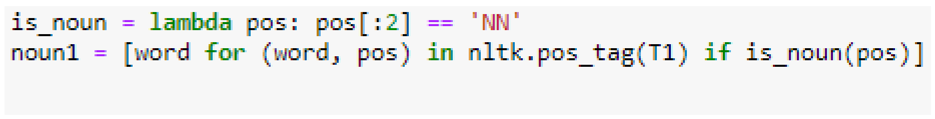
Finding NOUNS and VERBS

● Performing POS Tagging

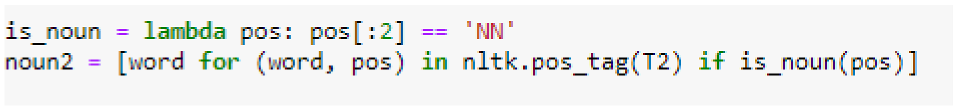
We will now perform the POS Tagging on T1 and T2 using the inbuilt function of nltk

namely pos_tag() which uses Penn Treebank tag set to perform POS tagging.

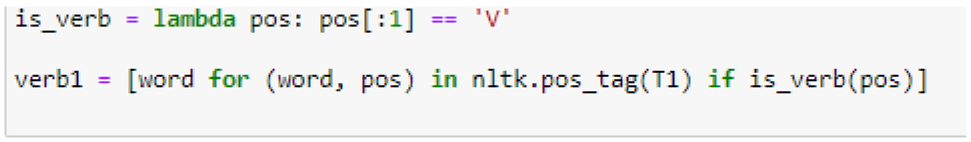
We will extract the words tagged explicitly as nouns and verbs separately from both the novels using the following code.



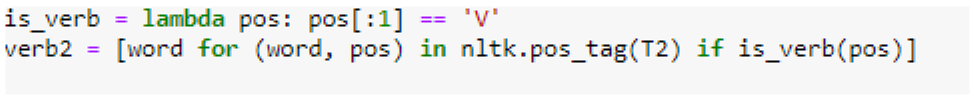
```
is_noun = lambda pos: pos[:2] == 'NN'
noun1 = [word for (word, pos) in nltk.pos_tag(T1) if is_noun(pos)]
```



```
is_noun = lambda pos: pos[:2] == 'NN'
noun2 = [word for (word, pos) in nltk.pos_tag(T2) if is_noun(pos)]
```

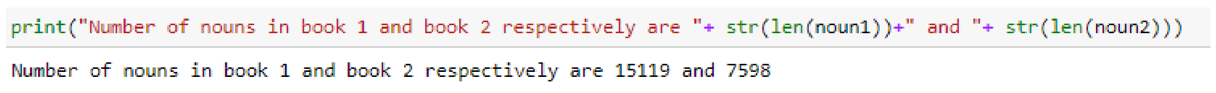


```
is_verb = lambda pos: pos[:1] == 'V'

verb1 = [word for (word, pos) in nltk.pos_tag(T1) if is_verb(pos)]
```



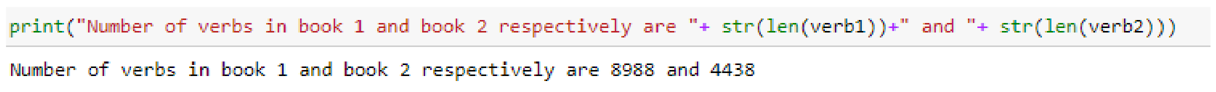
```
is_verb = lambda pos: pos[:1] == 'V'
verb2 = [word for (word, pos) in nltk.pos_tag(T2) if is_verb(pos)]
```

We will run above code to both book1 and book2, respectively, and print the total nouns and verbs in both.



```
print("Number of nouns in book 1 and book 2 respectively are "+ str(len(noun1))+" and "+ str(len(noun2)))
```

```
Number of nouns in book 1 and book 2 respectively are 15119 and 7598
```

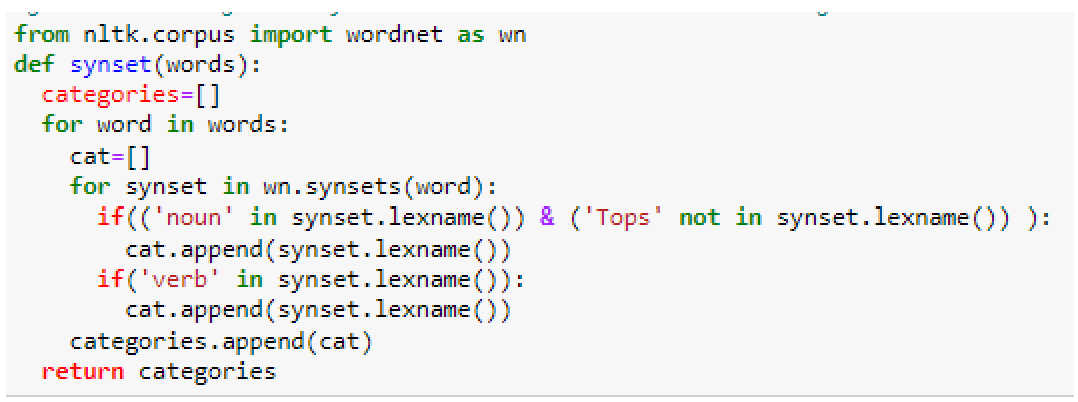


```
print("Number of verbs in book 1 and book 2 respectively are "+ str(len(verb1))+" and "+ str(len(verb2)))
```

```
Number of verbs in book 1 and book 2 respectively are 8988 and 4438
```

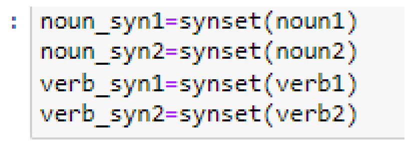
- *Get the categories that these words fall under in the WordNet.*

To retrieve the categories that each noun and verb belong to, in the wordnet synsets, we have used nltk.corpus.wordnet as it has all the tools required for this task. We have used the following function to extract categories each noun and verb belongs to. Since a noun also has synsets interpretations as verbs and vice versa hence we have included them as lists corresponding to its index in the noun and verb lists respectively.



```
from nltk.corpus import wordnet as wn
def synset(words):
  categories=[]
  for word in words:
    cat=[]
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        cat.append(synset.lexname())
      if('verb' in synset.lexname()):
        cat.append(synset.lexname())
    categories.append(cat)
  return categories
```

We will apply the above function to get 2-D lists which contain the categories that each noun and verb have been defined in the wordnet database.

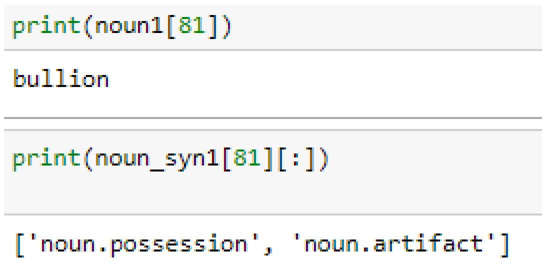


```
noun_syn1=synset(noun1)
noun_syn2=synset(noun2)
verb_syn1=synset(verb1)
verb_syn2=synset(verb2)
```

Hence noun_syn1, noun_syn2, verbsyn_1, verb_syn2 are two-dimensional. We list the categories that noun1, noun2, verb1, verb2 belong to in the wordnet synsets of nouns and verbs.

The 2d lists are indexed as noun_syn1[x][y], where x is the index of the corresponding nouns in noun1 and y is the index containing the categories it belongs to.

Eg.



```
print(noun1[81])
```

```
bullion
```

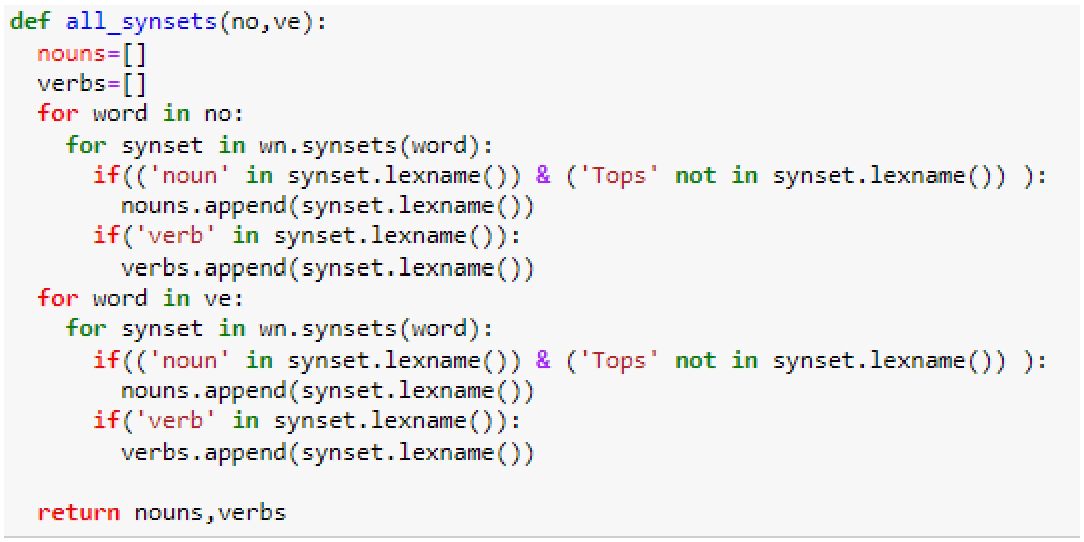
```
print(noun_syn1[81][:])
```

```
['noun.possession', 'noun.artifact']
```

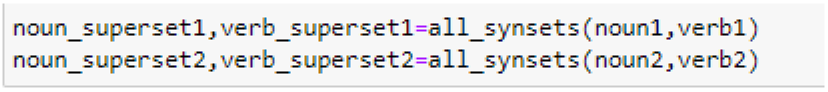
Hence a 'bullion' is both possession and artifact.

- Get the frequency of each category for each noun and verb in their corresponding and plot histogram/bar plots for each corresponding type.

To get the frequency of all the categories of nouns and verbs in the novels, we have created a set for each book that contains all the types of nouns and verbs occurring and then plotted the frequency distribution for the data.

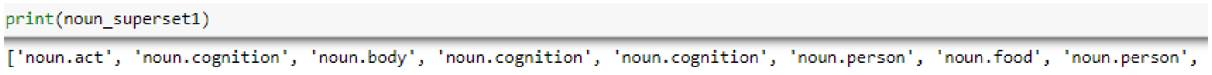


```
def all_synsets(no,ve):
  nouns=[]
  verbs=[]
  for word in no:
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        nouns.append(synset.lexname())
      if('verb' in synset.lexname()):
        verbs.append(synset.lexname())
  for word in ve:
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        nouns.append(synset.lexname())
      if('verb' in synset.lexname()):
        verbs.append(synset.lexname())

  return nouns,verbs
```



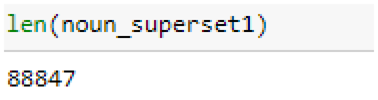
```
noun_superset1,verb_superset1=all_synsets(noun1,verb1)
noun_superset2,verb_superset2=all_synsets(noun2,verb2)
```

Here noun_superset1 contains all the different categories of nouns in novel one, and verb_superset1 has all the types of verbs.
Eg.



```
print(noun_superset1)
```

```
['noun.act', 'noun.cognition', 'noun.body', 'noun.cognition', 'noun.cognition', 'noun.person', 'noun.food', 'noun.person',
```



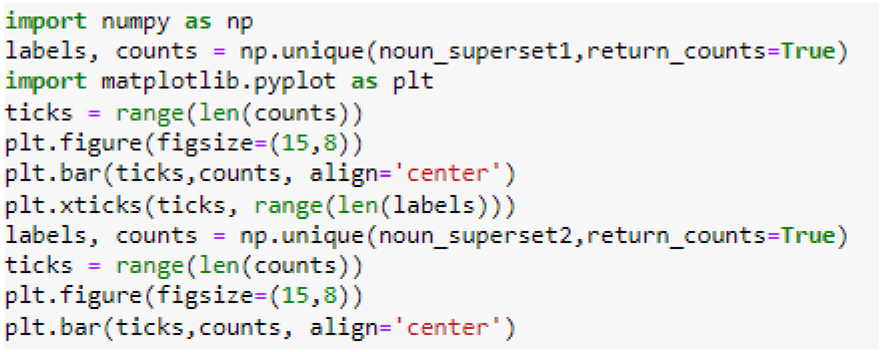
```
len(noun_superset1)
```

```
88847
```

Hence there are 88847 elements in the list.

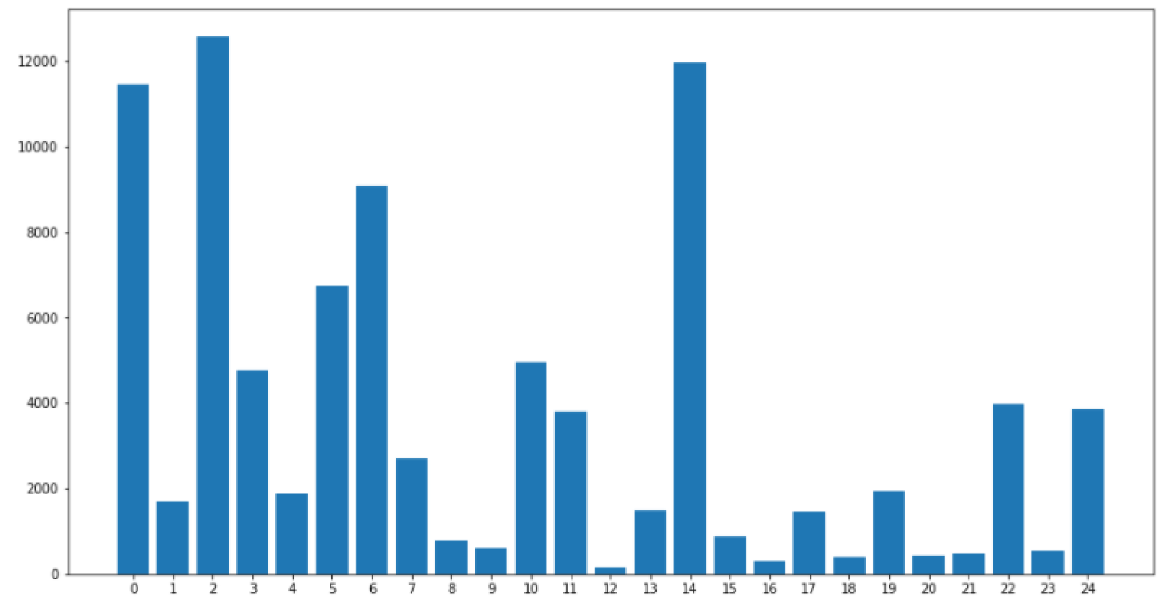
Plotting the histograms:

We have used numpy to count each type of noun and verb frequency out of 25 categories of nouns and 15 varieties of verbs from both novels.

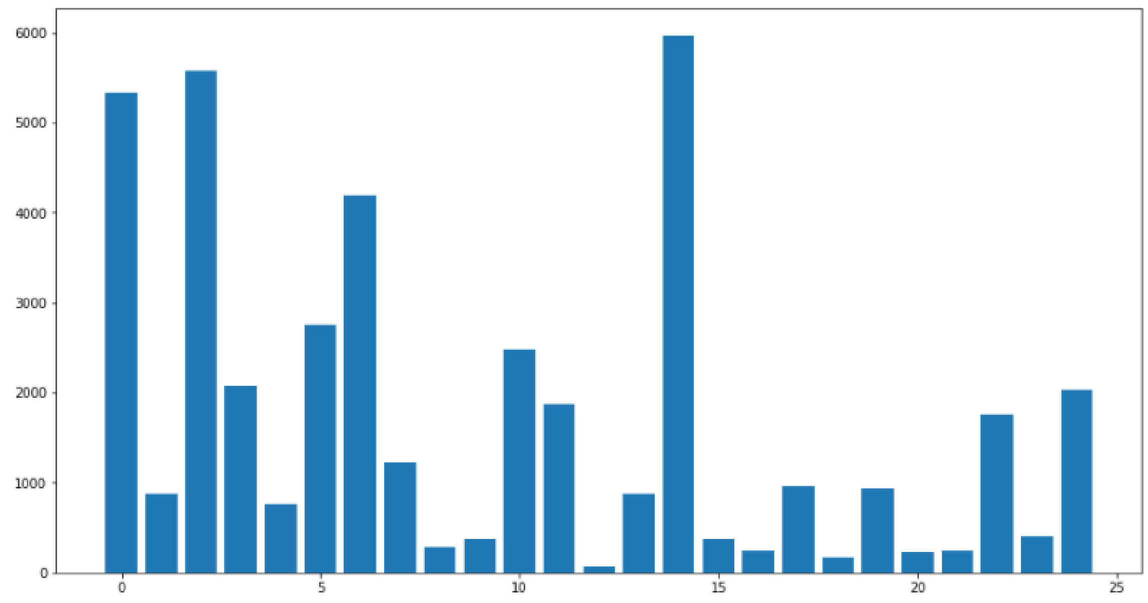


```
import numpy as np
labels, counts = np.unique(noun_superset1,return_counts=True)
import matplotlib.pyplot as plt
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
labels, counts = np.unique(noun_superset2,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
```

After plotting the graphs, we get the following plots where Y-axis is counts and x-axis are categories:

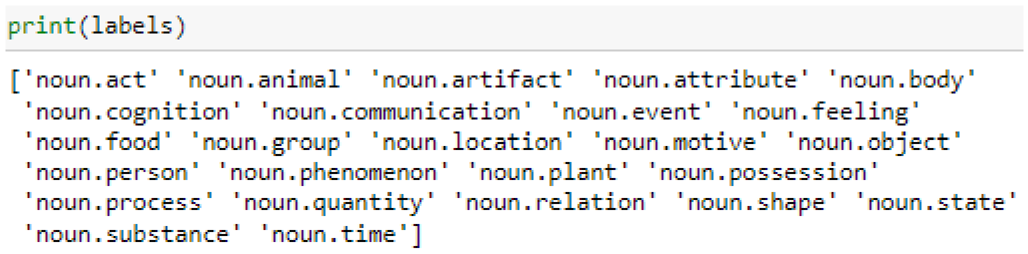


For Book 1, Nouns



For Book 2, Nouns

The categories are numbered as 0-24 in order as:

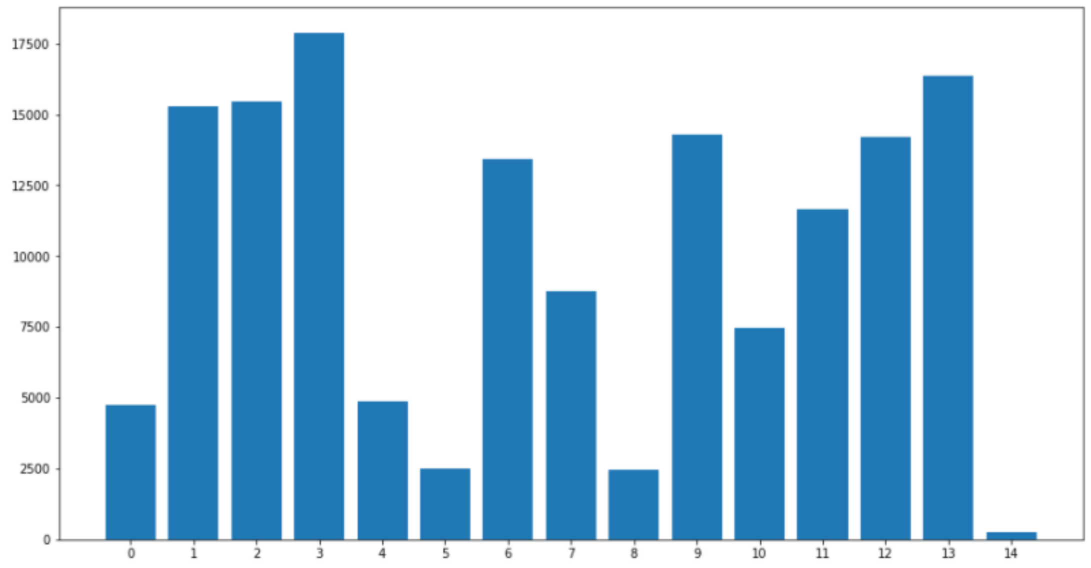


```
print(labels)
```

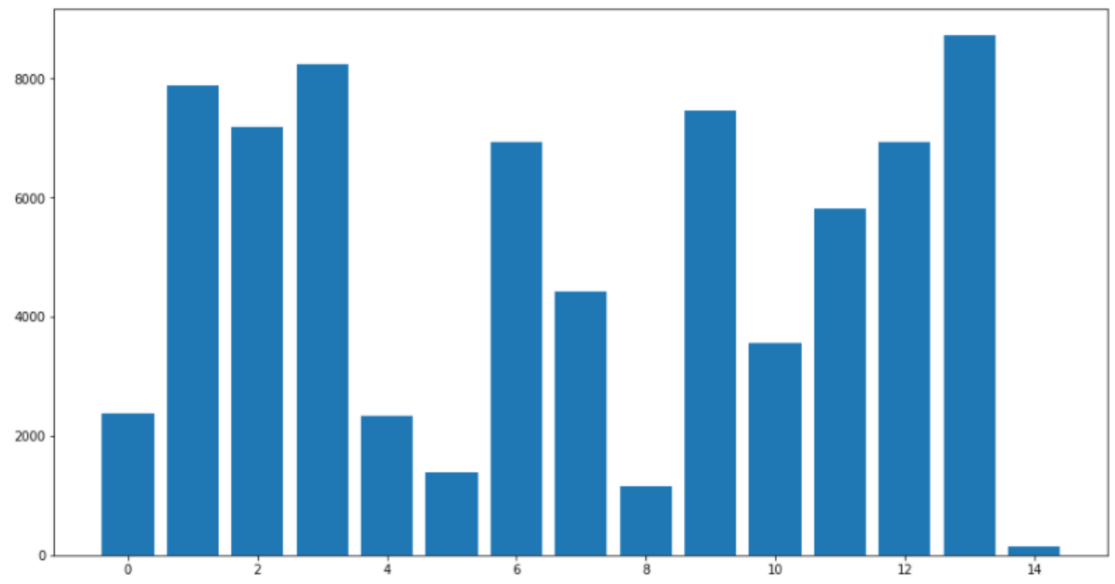
```
['noun.act' 'noun.animal' 'noun.artifact' 'noun.attribute' 'noun.body'
 'noun.cognition' 'noun.communication' 'noun.event' 'noun.feeling'
 'noun.food' 'noun.group' 'noun.location' 'noun.motive' 'noun.object'
 'noun.person' 'noun.phenomenon' 'noun.plant' 'noun.possession'
 'noun.process' 'noun.quantity' 'noun.relation' 'noun.shape' 'noun.state'
 'noun.substance' 'noun.time']
```

Here labels are arranged in the order of hierarchy as given in wordnet categories of nouns.

Similarly, we get the following plots for Verbs:



For Verbs in Book 1



For verbs in Book 2

The labels are numbered as 0-14 in the following order:

```
print(labels)
```

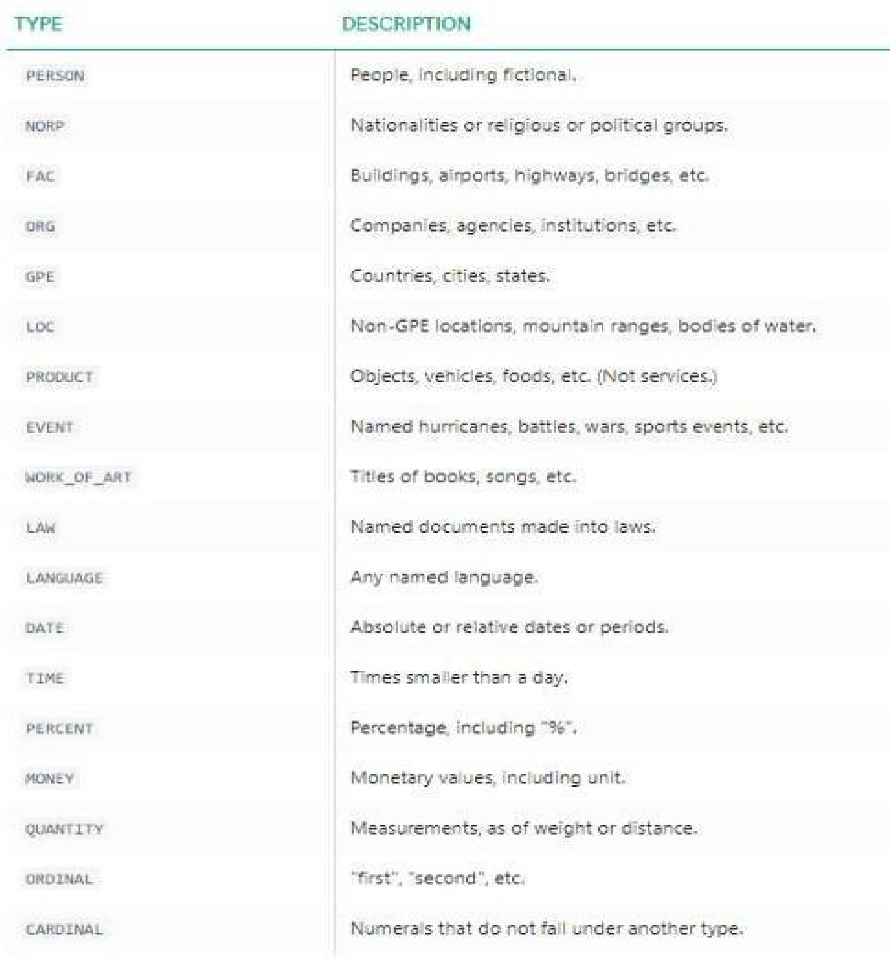
```
['verb.body' 'verb.change' 'verb.cognition' 'verb.communication'
 'verb.competition' 'verb.consumption' 'verb.contact' 'verb.creation'
 'verb.emotion' 'verb.motion' 'verb.perception' 'verb.possession'
 'verb.social' 'verb.stative' 'verb.weather']
```

Second Part:

Named Entity Recognition

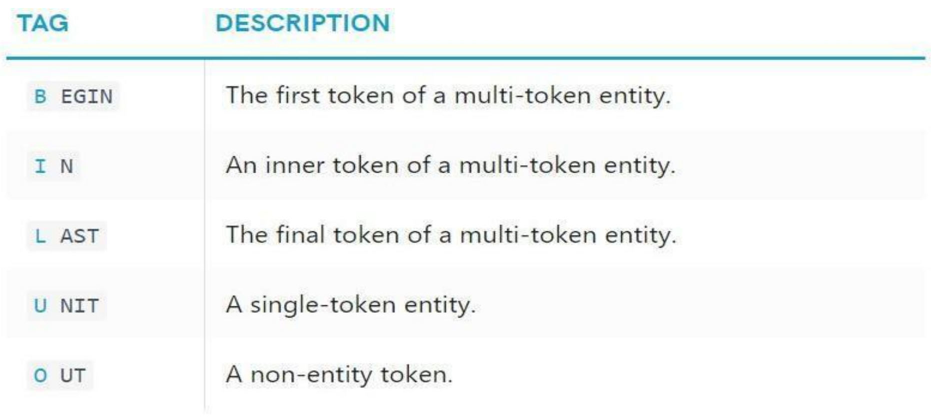
- Get the entities involved in each of the novels.

To perform Named Entity Recognition in both novels, we have used Spacy. SpaCy named entity recognition supports the following entity types:



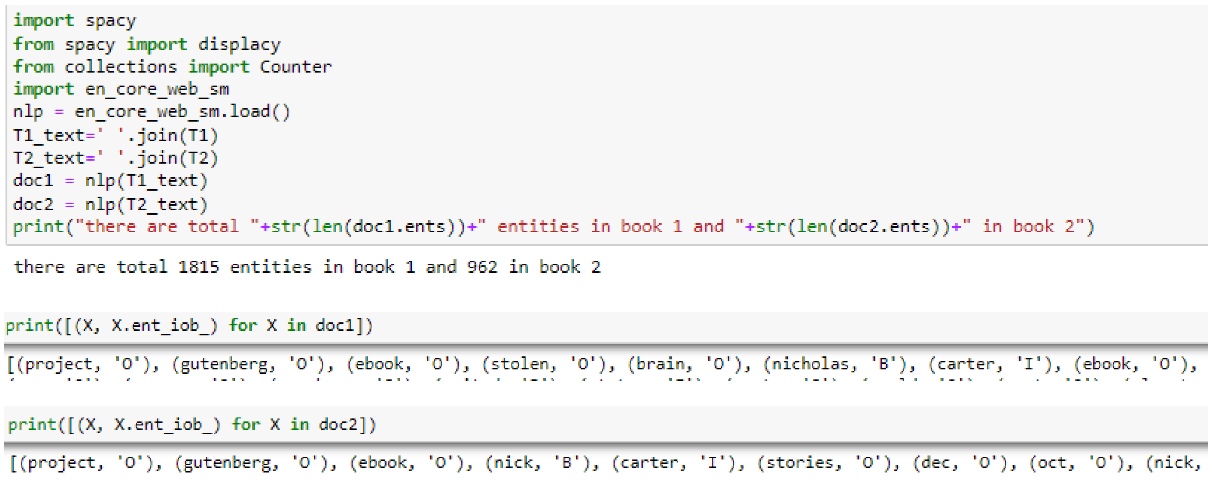
TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

The spacy uses token level entity annotation using the BILUO tagging scheme to describe the entity boundaries.



TAG	DESCRIPTION
B EGIN	The first token of a multi-token entity.
I N	An inner token of a multi-token entity.
L AST	The final token of a multi-token entity.
U NIT	A single-token entity.
O UT	A non-entity token.

First, we import spacy and get the entities involved in both novels using the methods described in the library.



```
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm.load()
T1_text=' '.join(T1)
T2_text=' '.join(T2)
doc1 = nlp(T1_text)
doc2 = nlp(T2_text)
print("there are total "+str(len(doc1.ents))+" entities in book 1 and "+str(len(doc2.ents))+" in book 2")
```

```
there are total 1815 entities in book 1 and 962 in book 2
```

```
print([(X, X.ent_iob_) for X in doc1])
```

```
[(project, 'O'), (gutenberg, 'O'), (ebook, 'O'), (stolen, 'O'), (brain, 'O'), (nicholas, 'B'), (carter, 'I'), (ebook, 'O'),
```

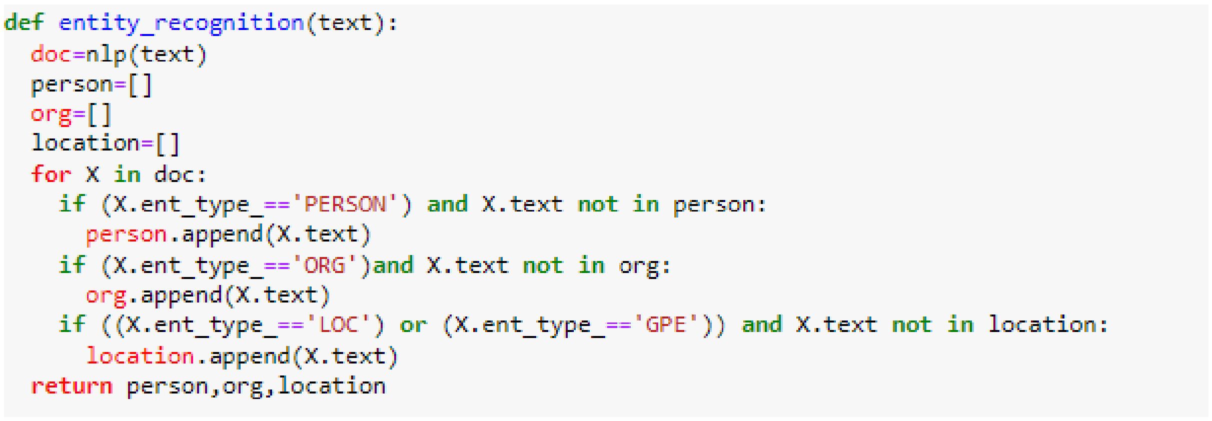
```
print([(X, X.ent_iob_) for X in doc2])
```

```
[(project, 'O'), (gutenberg, 'O'), (ebook, 'O'), (nick, 'B'), (carter, 'I'), (stories, 'O'), (dec, 'O'), (oct, 'O'), (nick,
```

As we can see, the entities are annotated using the BILUO entity scheme.

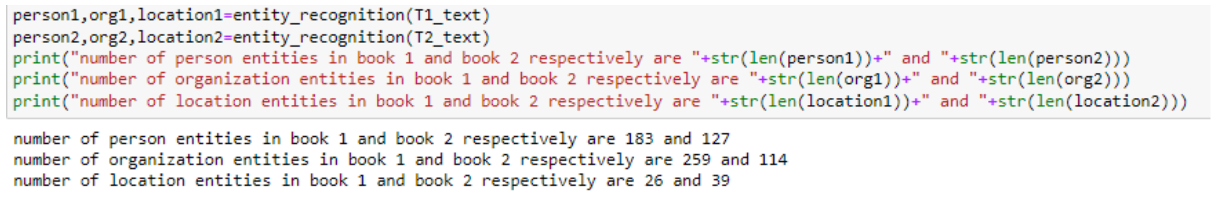
- Get Spacy's annotated entities: person,organization, and Location.

We have used the following function on the entities returned by spacy to collect the Person, Organization and Location entities in each of the novels. The ent_type function returns the type of entity annotated by Spacy and returns lists containing the above types of entities.



```
def entity_recognition(text):
  doc=nlp(text)
  person=[]
  org=[]
  location=[]
  for X in doc:
    if (X.ent_type_=='PERSON') and X.text not in person:
      person.append(X.text)
    if (X.ent_type_=='ORG')and X.text not in org:
      org.append(X.text)
    if ((X.ent_type_=='LOC') or (X.ent_type_=='GPE')) and X.text not in location:
      location.append(X.text)
  return person,org,location
```

Now collecting these entities from both books:

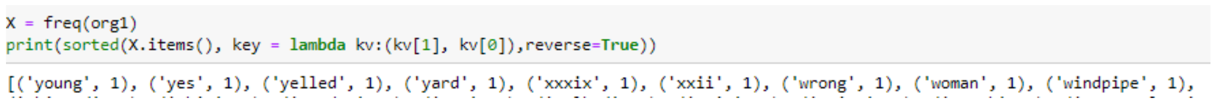


```
person1,org1,location1=entity_recognition(T1_text)
person2,org2,location2=entity_recognition(T2_text)
print("number of person entities in book 1 and book 2 respectively are "+str(len(person1))+" and "+str(len(person2)))
print("number of organization entities in book 1 and book 2 respectively are "+str(len(org1))+" and "+str(len(org2)))
print("number of location entities in book 1 and book 2 respectively are "+str(len(location1))+" and "+str(len(location2)))
```

```
number of person entities in book 1 and book 2 respectively are 183 and 127
number of organization entities in book 1 and book 2 respectively are 259 and 114
number of location entities in book 1 and book 2 respectively are 26 and 39
```

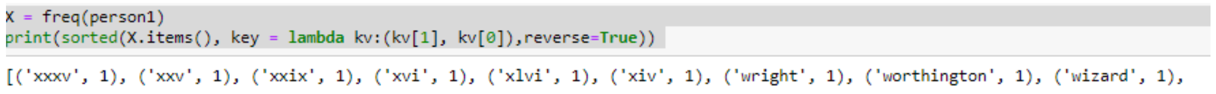
Each of these lists contains the corresponding entities.

Counting the number of occurrences of names, locations, and organisations in Book 1, we get:



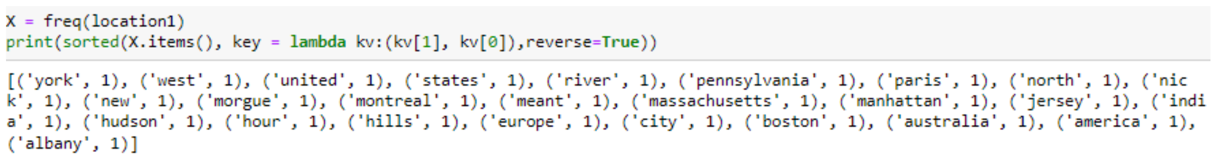
```
X = freq(org1)
print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('young', 1), ('yes', 1), ('yelled', 1), ('yard', 1), ('xxxix', 1), ('xxii', 1), ('wrong', 1), ('woman', 1), ('windpipe', 1),
```



```
X = freq(person1)
print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('xxxv', 1), ('xxv', 1), ('xxix', 1), ('xvi', 1), ('xlvi', 1), ('xiv', 1), ('wright', 1), ('worthington', 1), ('wizard', 1),
```



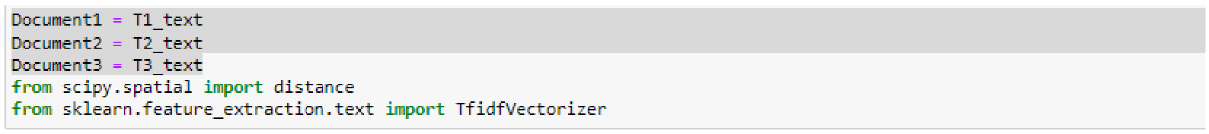
```
X = freq(location1)
print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('york', 1), ('west', 1), ('united', 1), ('states', 1), ('river', 1), ('pennsylvania', 1), ('paris', 1), ('north', 1), ('nic
k', 1), ('new', 1), ('morgue', 1), ('montreal', 1), ('meant', 1), ('massachusetts', 1), ('manhattan', 1), ('jersey', 1), ('indi
a', 1), ('hudson', 1), ('hour', 1), ('hills', 1), ('europe', 1), ('city', 1), ('boston', 1), ('australia', 1), ('america', 1),
('albany', 1)]
```

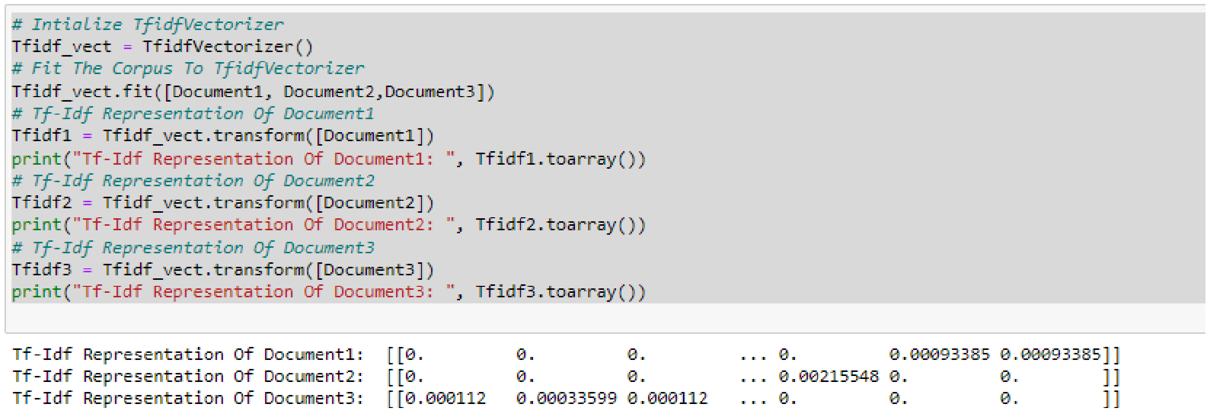
Third Part:

- Creating TF-IDF vectors for all books and finding the cosine similarity between them, and finding which two books are more similar.

  We are importing the sklearn library for creating TF-IDF vectors and the scipy library for calculating cosine similarity between two books.We used inbuilt Tfidf_vect.transform() unction to create TF-IDF vectors.

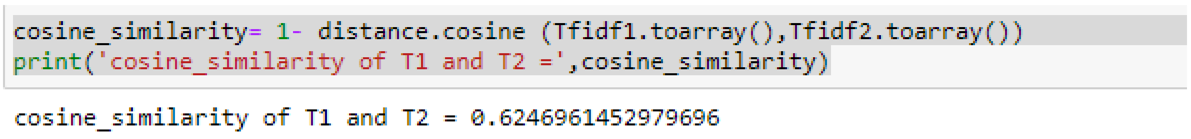


```
Document1 = T1_text
Document2 = T2_text
Document3 = T3_text
from scipy.spatial import distance
from sklearn.feature_extraction.text import TfidfVectorizer
```



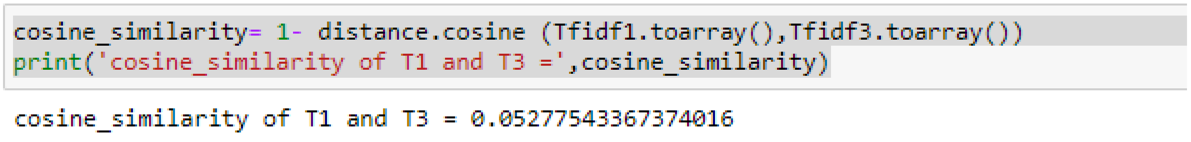
```
# Intialize TfidfVectorizer
Tfidf_vect = TfidfVectorizer()
# Fit The Corpus To TfidfVectorizer
Tfidf_vect.fit([Document1, Document2,Document3])
# Tf-Idf Representation Of Document1
Tfidf1 = Tfidf_vect.transform([Document1])
print("Tf-Idf Representation Of Document1: ", Tfidf1.toarray())
# Tf-Idf Representation Of Document2
Tfidf2 = Tfidf_vect.transform([Document2])
print("Tf-Idf Representation Of Document2: ", Tfidf2.toarray())
# Tf-Idf Representation Of Document3
Tfidf3 = Tfidf_vect.transform([Document3])
print("Tf-Idf Representation Of Document3: ", Tfidf3.toarray())
```

```
Tf-Idf Representation Of Document1:  [[0.         0.         0.         ... 0.         0.00093385 0.00093385]]
Tf-Idf Representation Of Document2:  [[0.         0.         0.         ... 0.00215548 0.         0.        ]]
Tf-Idf Representation Of Document3:  [[0.000112   0.00033599 0.000112   ... 0.         0.         0.        ]]
```



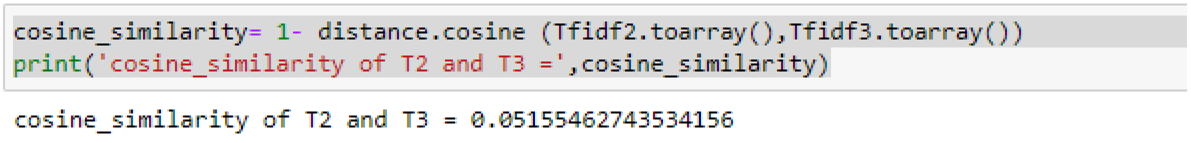
```
cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf2.toarray())
print('cosine_similarity of T1 and T2 =',cosine_similarity)
```

```
cosine_similarity of T1 and T2 = 0.6246961452979696
```



```
cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf3.toarray())
print('cosine_similarity of T1 and T3 =',cosine_similarity)
```

```
cosine_similarity of T1 and T3 = 0.05277543367374016
```



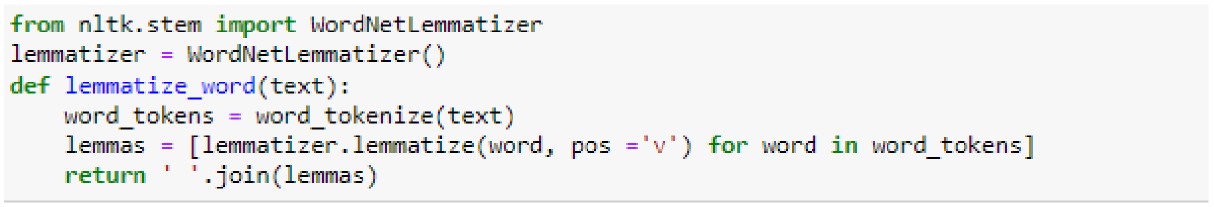
```
cosine_similarity= 1- distance.cosine (Tfidf2.toarray(),Tfidf3.toarray())
print('cosine_similarity of T2 and T3 =',cosine_similarity)
```

```
cosine_similarity of T2 and T3 = 0.05155462743534156
```

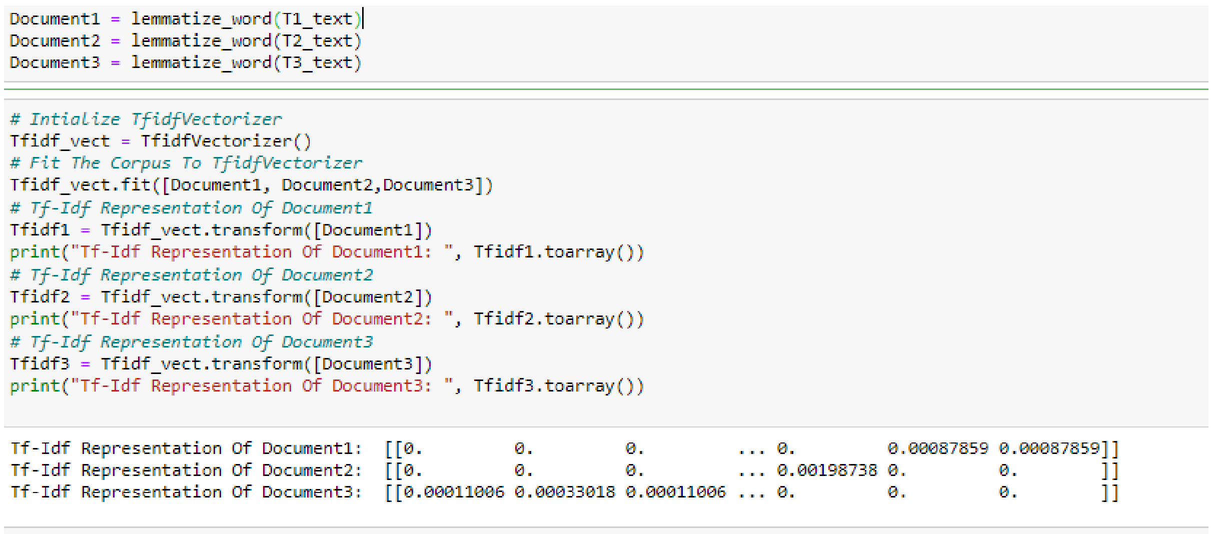
From the above cosine similarity values, we know that T1 and T2 are most similar.
The similarity order is as follow:
(T1, T2)>(T1, T3)>(T2, T3)

- Doing lemmatization of the books and creating the TF-IDF vectors for all the books and finding the cosine similarity of each pair of readers.

  We are using WordNet lemmatizer from nltk library for lemmatization.



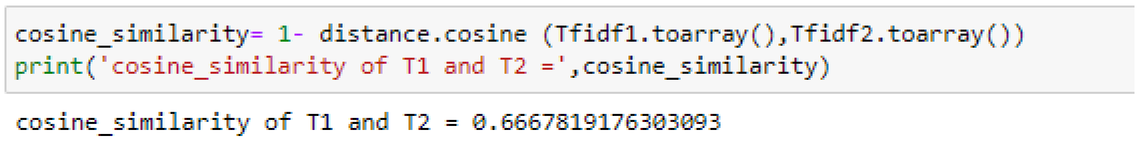
```python
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def lemmatize_word(text):
    word_tokens = word_tokenize(text)
    lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word_tokens]
    return ' '.join(lemmas)
```



```python
Document1 = lemmatize_word(T1_text)
Document2 = lemmatize_word(T2_text)
Document3 = lemmatize_word(T3_text)
```

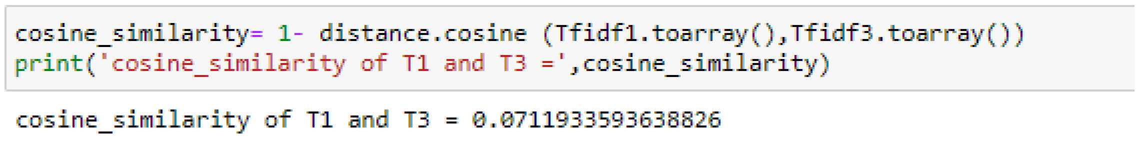
```python
# Intialize TfidfVectorizer
Tfidf_vect = TfidfVectorizer()
# Fit The Corpus To TfidfVectorizer
Tfidf_vect.fit([Document1, Document2,Document3])
# Tf-Idf Representation Of Document1
Tfidf1 = Tfidf_vect.transform([Document1])
print("Tf-Idf Representation Of Document1: ", Tfidf1.toarray())
# Tf-Idf Representation Of Document2
Tfidf2 = Tfidf_vect.transform([Document2])
print("Tf-Idf Representation Of Document2: ", Tfidf2.toarray())
# Tf-Idf Representation Of Document3
Tfidf3 = Tfidf_vect.transform([Document3])
print("Tf-Idf Representation Of Document3: ", Tfidf3.toarray())
```

```
Tf-Idf Representation Of Document1:  [[0.         0.         0.         ... 0.         0.00087859 0.00087859]]
Tf-Idf Representation Of Document2:  [[0.         0.         0.         ... 0.00198738 0.         0.        ]]
Tf-Idf Representation Of Document3:  [[0.00011006 0.00033018 0.00011006 ... 0.         0.         0.        ]]
```



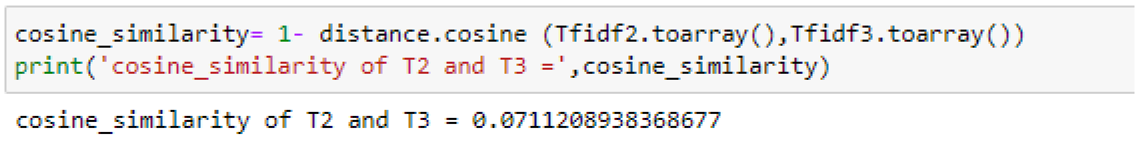
```python
cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf2.toarray())
print('cosine_similarity of T1 and T2 =',cosine_similarity)
```

```
cosine_similarity of T1 and T2 = 0.6667819176303093
```



```python
cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf3.toarray())
print('cosine_similarity of T1 and T3 =',cosine_similarity)
```

```
cosine_similarity of T1 and T3 = 0.0711933593638826
```



```python
cosine_similarity= 1- distance.cosine (Tfidf2.toarray(),Tfidf3.toarray())
print('cosine_similarity of T2 and T3 =',cosine_similarity)
```

```
cosine_similarity of T2 and T3 = 0.0711208938368677
```

After lemmatization, from above cosine similarity values we get to know that T1 and T3 are most similar.
The similarity order as follow:
(T1,T3)>(T2,T3)>(T1,T2)

Conclusion:

We have learnt how to perform Semantic analysis, POS-Tagging, Named Entity Recognition,Performance evaluation in NLP , finding TF-IDF vector and Cosine similarity from raw text data through NLP.