NLP PROJECT

PoS Tagging

Team : Triple Strike

Submitted to: **Dr. Sakthi Balan**

Submitted By:

Saurav Somani 19UCS177

Shreyansh Jain 19UCS003

Shikhar Nigam 19UCS146

ACKNOWLEDGEMENT

We are grateful to our respected teacher, Dr. Sakthi Balan, whose provided knowledge in classes benefited us in completing this project successfully. Thank you so much for your continuous support and presence whenever needed.

Last but not the least, we would like to thank everyone who is involved in the project directly or indirectly.

And we are always thankful to The Great Almighty for always having a blessing on us.

Aim

The aim of this part of the project is to obtain PoS tagging for two books downloaded from project gutenberg.

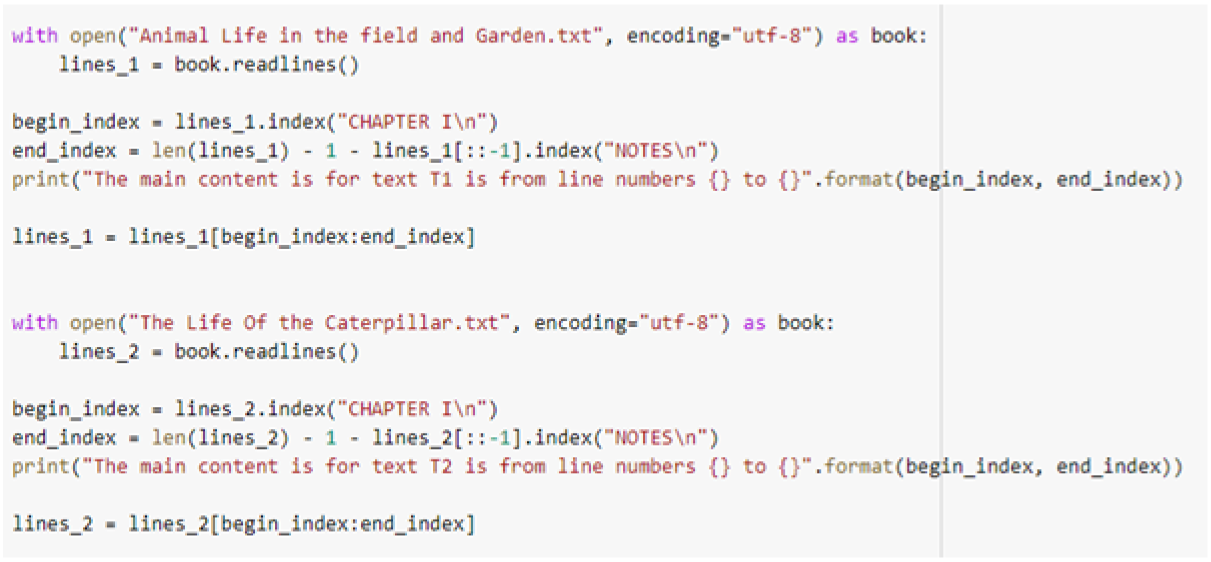
Code Link: https://github.com/Saurav-Somani/NLP_Project/blob/main/code.py

Data Description

The two books which we have used in our project is:

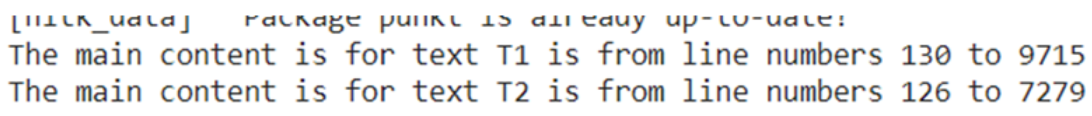
- Animal Life in field and garden, by Jean- Henri Fabre as T1
  https://www.gutenberg.org/cache/epub/66755/pg66755.txt
- The Life of the Caterpillar, by Jean- Henri Fabre as T2
  https://www.gutenberg.org/cache/epub/66762/pg66762.txt

The lines from which the real book starts is been coded here as below:



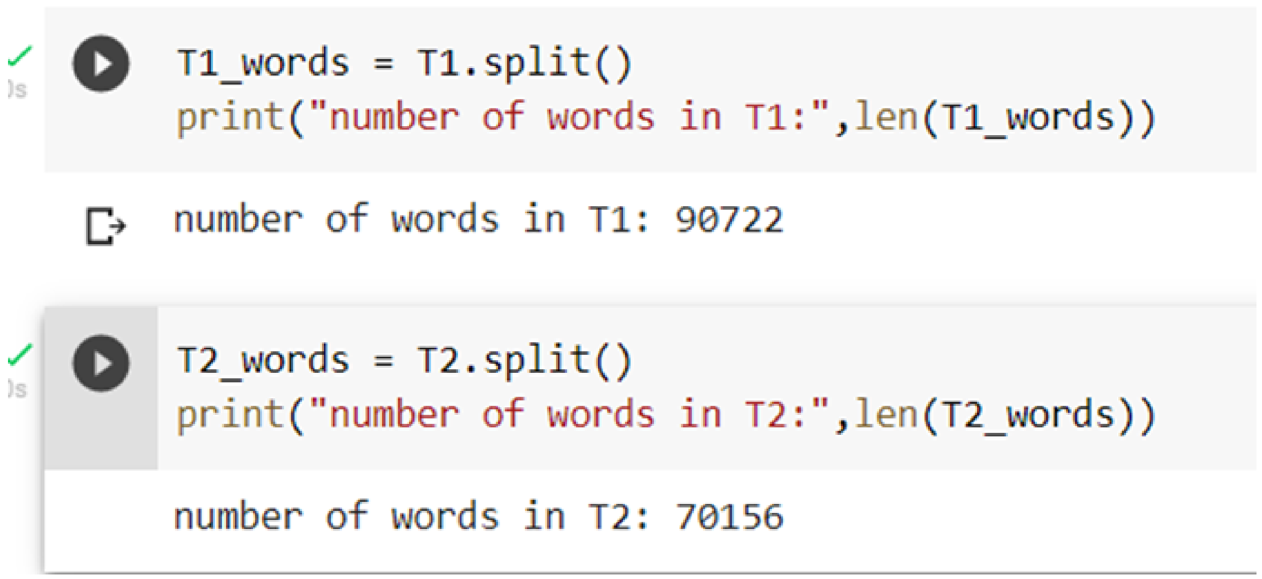
```
with open("Animal Life in the field and Garden.txt", encoding="utf-8") as book:
    lines_1 = book.readlines()

begin_index = lines_1.index("CHAPTER I\n")
end_index = len(lines_1) - 1 - lines_1[::-1].index("NOTES\n")
print("The main content is for text T1 is from line numbers {} to {}".format(begin_index, end_index))

lines_1 = lines_1[begin_index:end_index]


with open("The Life Of the Caterpillar.txt", encoding="utf-8") as book:
    lines_2 = book.readlines()

begin_index = lines_2.index("CHAPTER I\n")
end_index = len(lines_2) - 1 - lines_2[::-1].index("NOTES\n")
print("The main content is for text T2 is from line numbers {} to {}".format(begin_index, end_index))

lines_2 = lines_2[begin_index:end_index]
```

Output is as follows:



```
[nltk_data]   Package punkt is already up-to-date!
The main content is for text T1 is from line numbers 130 to 9715
The main content is for text T2 is from line numbers 126 to 7279
```

Number of words in both of the books are:



```
T1_words = T1.split()
print("number of words in T1:",len(T1_words))
```

```
number of words in T1: 90722
```

```
T2_words = T2.split()
print("number of words in T2:",len(T2_words))
```

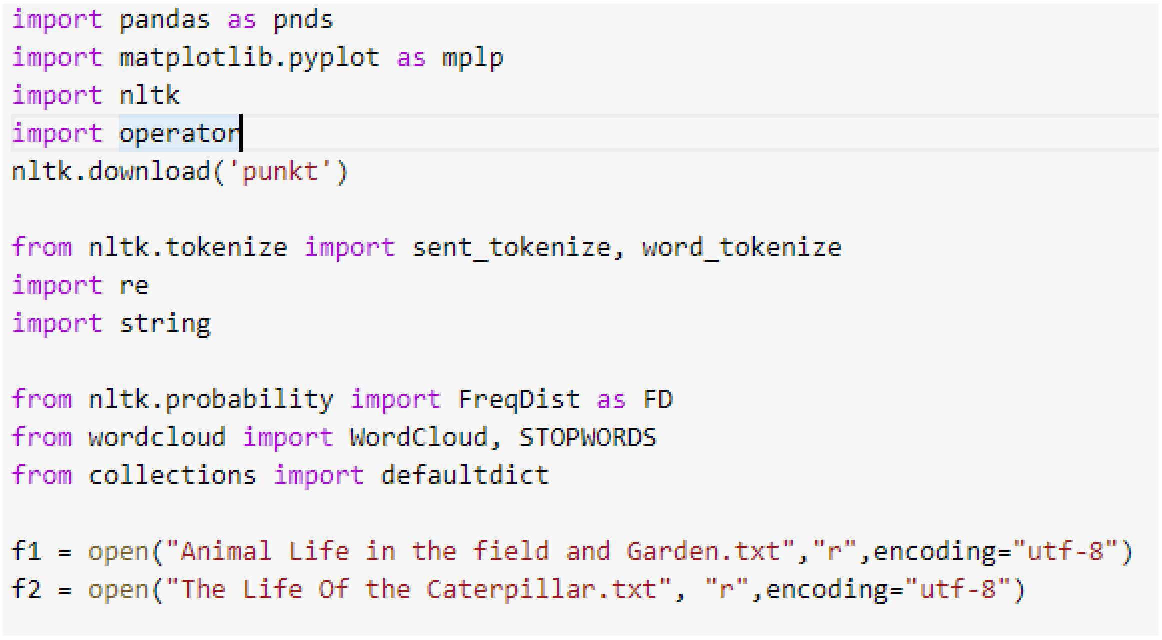
```
number of words in T2: 70156
```

Firstly importing some libraries for various tasks

- Importing 'pandas' module for data analysis and machine learning tasks.
- Importing 'nltk'(natural language toolkit) for the tasks related to NLP.
- Importing 'matplotlib.pyplot' for graphs and frequency analysis.

Importing the Text files, calling it T1 and T2.

- We have imported the 2 text files using open() method of python in read mode. And then for further processing of data we have converted the whole text file into a string using read() function. T1 is for 'A Tale of Two Cities' & T2 is for 'Pride & Prejudice' Code for the same is given below.



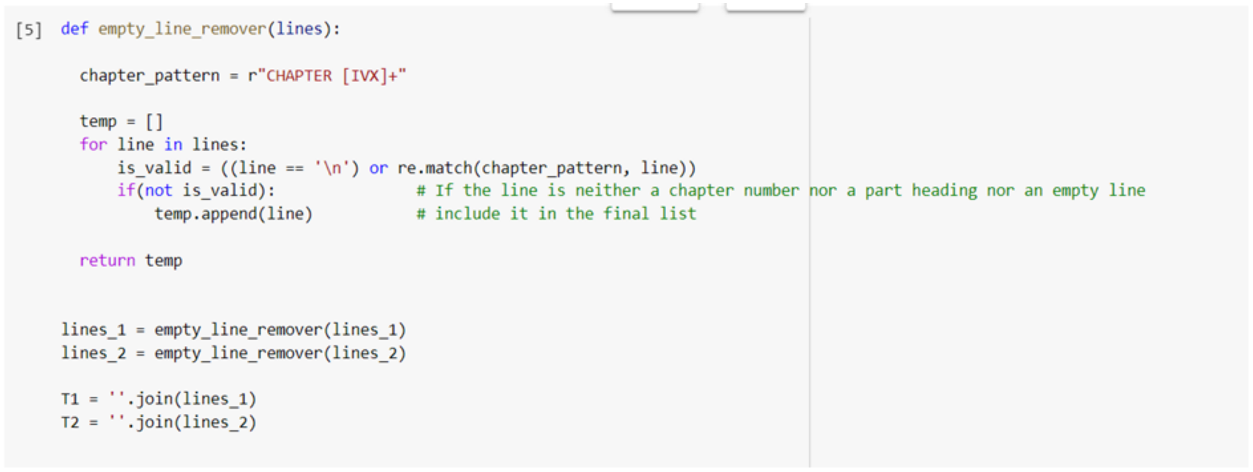
```
import pandas as pnds
import matplotlib.pyplot as mplp
import nltk
import operator
nltk.download('punkt')

from nltk.tokenize import sent_tokenize, word_tokenize
import re
import string

from nltk.probability import FreqDist as FD
from wordcloud import WordCloud, STOPWORDS
from collections import defaultdict

f1 = open("Animal Life in the field and Garden.txt","r",encoding="utf-8")
f2 = open("The Life Of the Caterpillar.txt", "r",encoding="utf-8")
```

Text Pre processing

- For text/data preprocessing we will be using inbuilt functions of ‘nltk’ libraries.
- We will be removing them by defining a regex for the same.
- This process involves
  - Converting the whole text in lower case.
  - Removing all the punctuations,numbers and all the irrelevant characters regex for this is [^a-zA-Z].
  - Removing all the links using regex [https?://\S+|www\.\S+’]
  - Also removing the running words

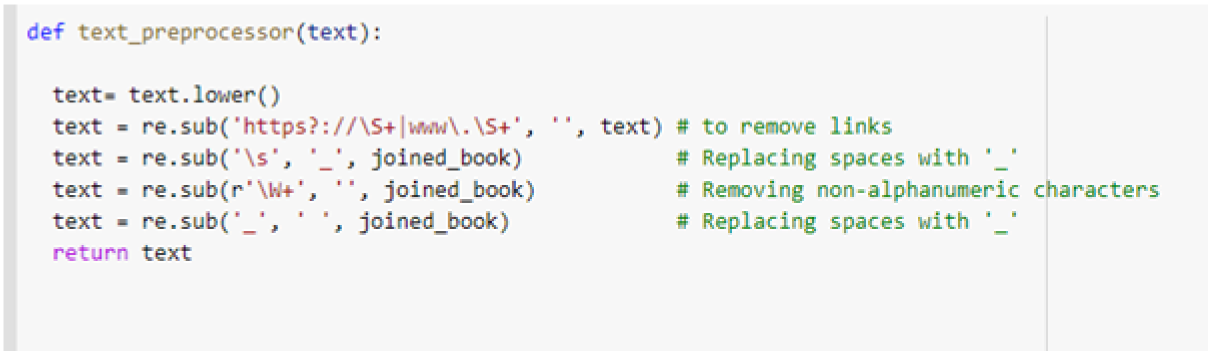
- Here chapter is the running word hence we will remove them and replace them with a space .Also we have removed all the empty lines. We have made a function empty_line_remover().
- We did all this using the function text_preprocessor(text), this is a function defined by us. Code for the same is

- Code for removal of empty lines and running words:



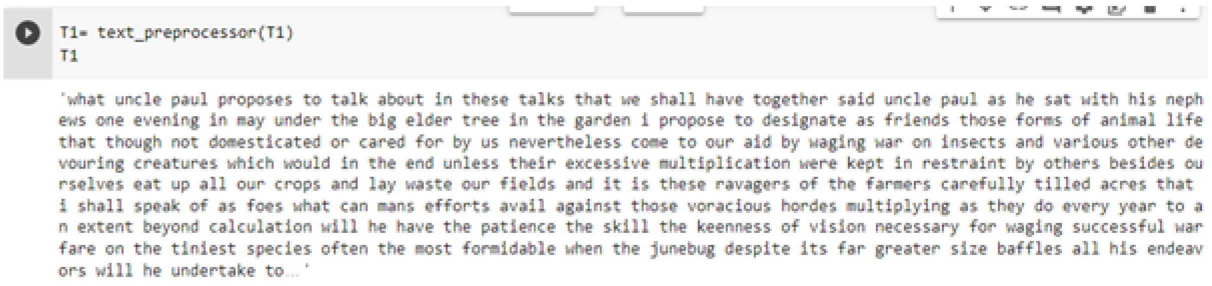
```
[5] def empty_line_remover(lines):

      chapter_pattern = r"CHAPTER [IVX]+"

      temp = []
      for line in lines:
          is_valid = ((line == '\n') or re.match(chapter_pattern, line))
          if(not is_valid):               # If the line is neither a chapter number nor a part heading nor an empty line
              temp.append(line)           # include it in the final list

      return temp


    lines_1 = empty_line_remover(lines_1)
    lines_2 = empty_line_remover(lines_2)

    T1 = ''.join(lines_1)
    T2 = ''.join(lines_2)
```

- Code for text_processor:



```
def text_preprocessor(text):

  text= text.lower()
  text = re.sub('https?://\S+|www\.\S+', '', text) # to remove links
  text = re.sub('\s', '_', joined_book)           # Replacing spaces with '_'
  text = re.sub(r'\W+', '', joined_book)           # Removing non-alphanumeric characters
  text = re.sub('_', ' ', joined_book)            # Replacing spaces with '_'
  return text
```

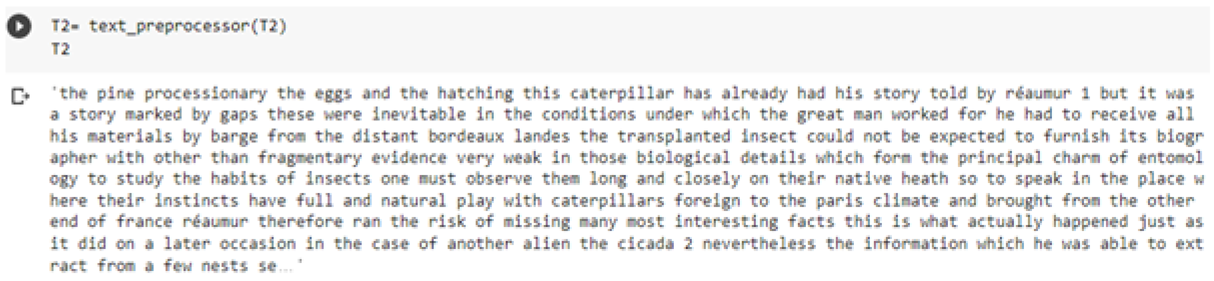
- Output after this process for T1 file is:



```
T1= text_preprocessor(T1)
T1
```

```
'what uncle paul proposes to talk about in these talks that we shall have together said uncle paul as he sat with his neph
ews one evening in may under the big elder tree in the garden i propose to designate as friends those forms of animal life
that though not domesticated or cared for by us nevertheless come to our aid by waging war on insects and various other de
vouring creatures which would in the end unless their excessive multiplication were kept in restraint by others besides ou
rselves eat up all our crops and lay waste our fields and it is these ravagers of the farmers carefully tilled acres that
i shall speak of as foes what can mans efforts avail against those voracious hordes multiplying as they do every year to a
n extent beyond calculation will he have the patience the skill the keenness of vision necessary for waging successful war
fare on the tiniest species often the most formidable when the junebug despite its far greater size baffles all his endeav
ors will he undertake to...'
```

- Output after this process for T2 file is:



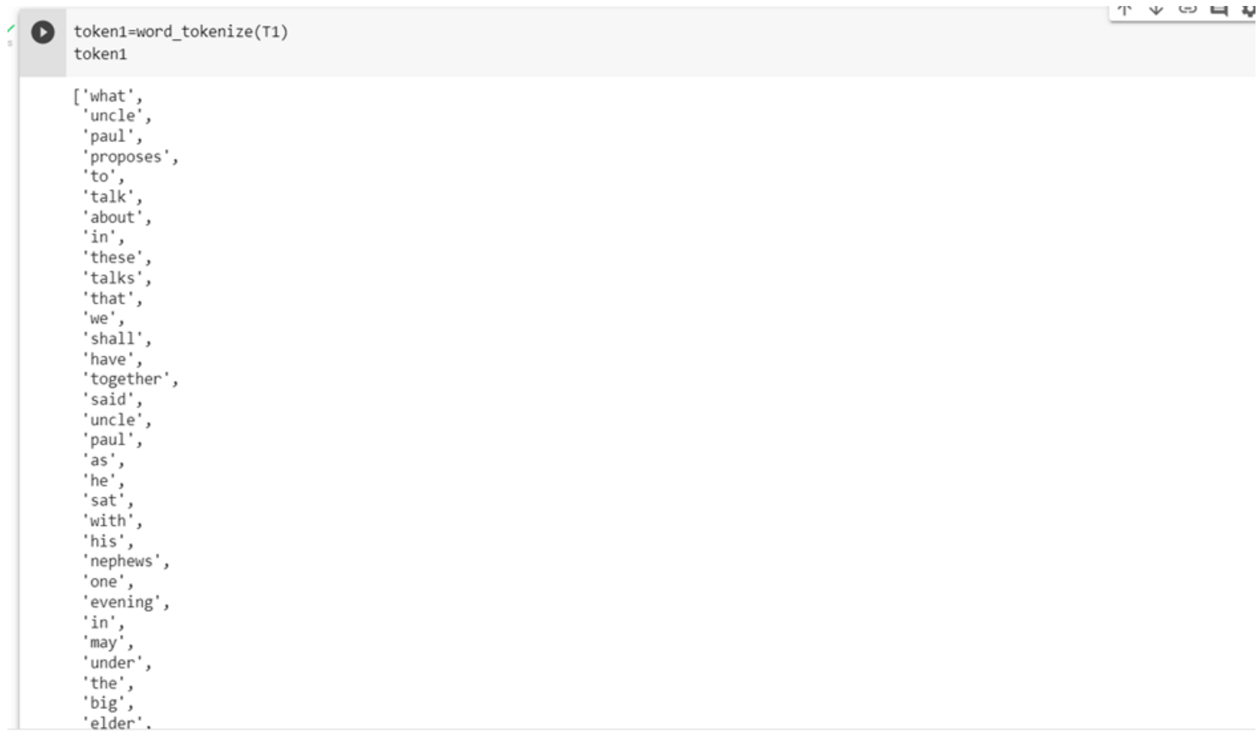
```
T2= text_preprocessor(T2)
T2
```

```
'the pine processionary the eggs and the hatching this caterpillar has already had his story told by réaumur 1 but it was
a story marked by gaps these were inevitable in the conditions under which the great man worked for he had to receive all
his materials by barge from the distant bordeaux landes the transplanted insect could not be expected to furnish its biogr
apher with other than fragmentary evidence very weak in those biological details which form the principal charm of entomol
ogy to study the habits of insects one must observe them long and closely on their native heath so to speak in the place w
here their instincts have full and natural play with caterpillars foreign to the paris climate and brought from the other
end of france réaumur therefore ran the risk of missing many most interesting facts this is what actually happened just as
it did on a later occasion in the case of another alien the cicada 2 nevertheless the information which he was able to ext
ract from a few nests se...'
```

Tokenization

- Now tokenizing the texts using the predefined function word_tokenize() of nltk module. The syntax for it is var = word_tokenize(T1)

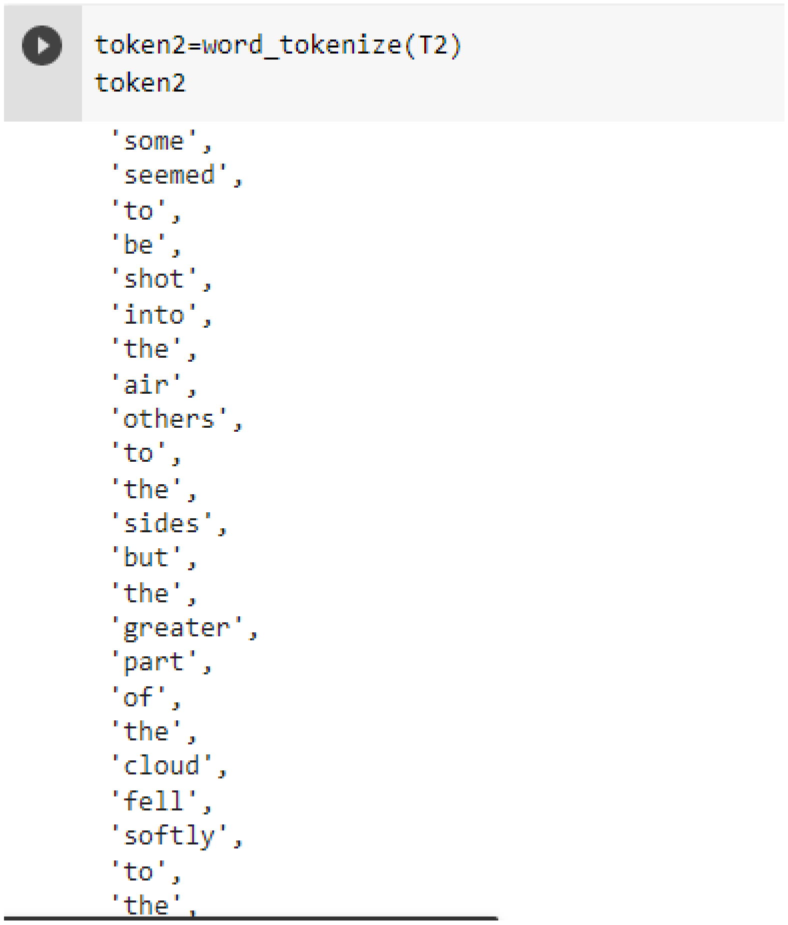
- Output after passing the pre processed text in the word_toeknize function for T1 text is:



```
token1=word_tokenize(T1)
token1
```

```
['what',
 'uncle',
 'paul',
 'proposes',
 'to',
 'talk',
 'about',
 'in',
 'these',
 'talks',
 'that',
 'we',
 'shall',
 'have',
 'together',
 'said',
 'uncle',
 'paul',
 'as',
 'he',
 'sat',
 'with',
 'his',
 'nephews',
 'one',
 'evening',
 'in',
 'may',
 'under',
 'the',
 'big',
 'elder',
```

- Output after passing the pre processed text in the word_toeknize function for T2 text is:



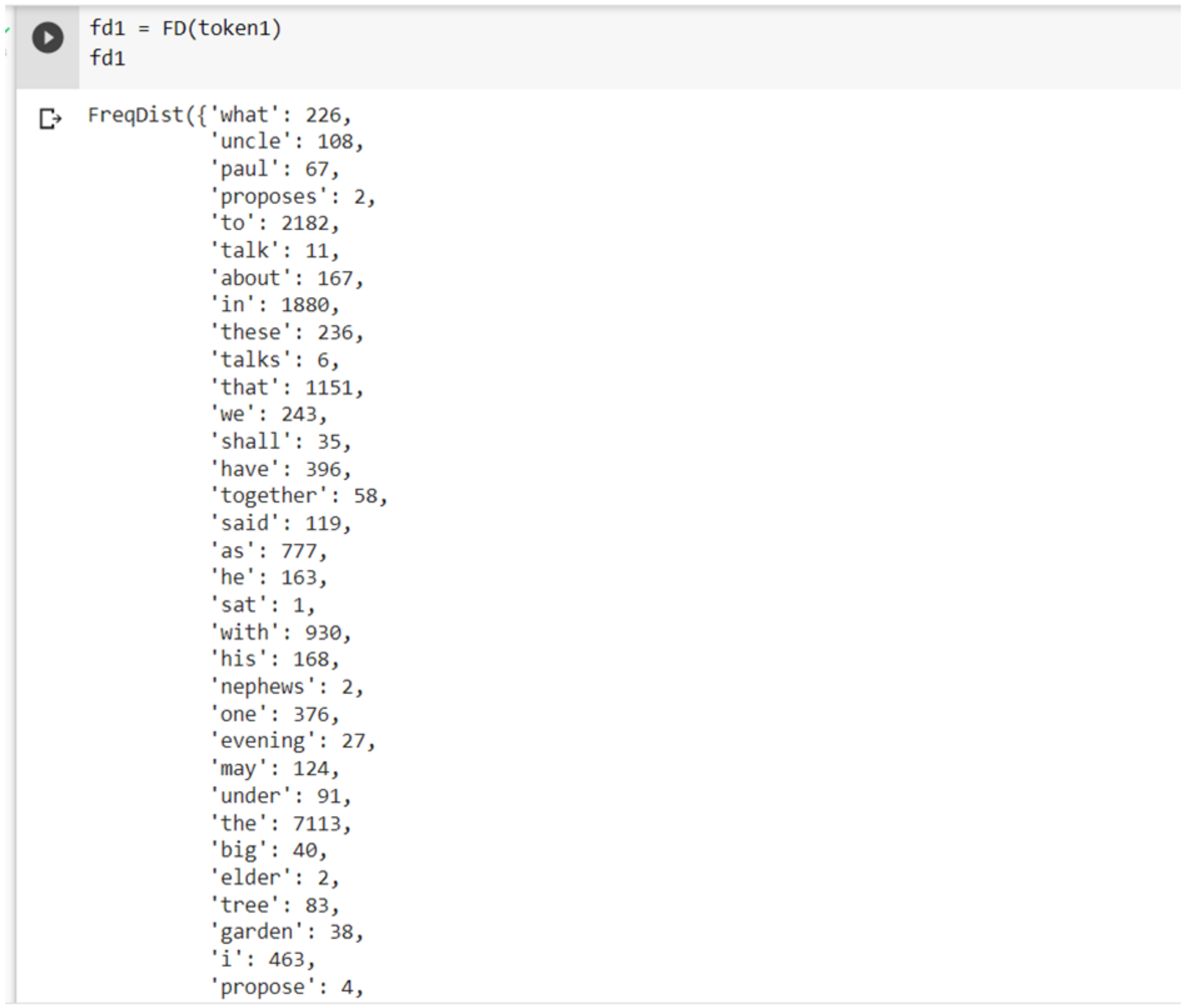
```
token2=word_tokenize(T2)
token2
```

```
'some',
'seemed',
'to',
'be',
'shot',
'into',
'the',
'air',
'others',
'to',
'the',
'sides',
'but',
'the',
'greater',
'part',
'of',
'the',
'cloud',
'fell',
'softly',
'to',
'the',
```

Frequency Analysing

- For frequency analysis we have to import the FreqDist module from nltk.probability.
- Now calling the inbuilt function FreqDist() to calculate the frequency of the tokenized text. To get the output we convert the output from FreqDist() to a dictionary. The output for text1 is below

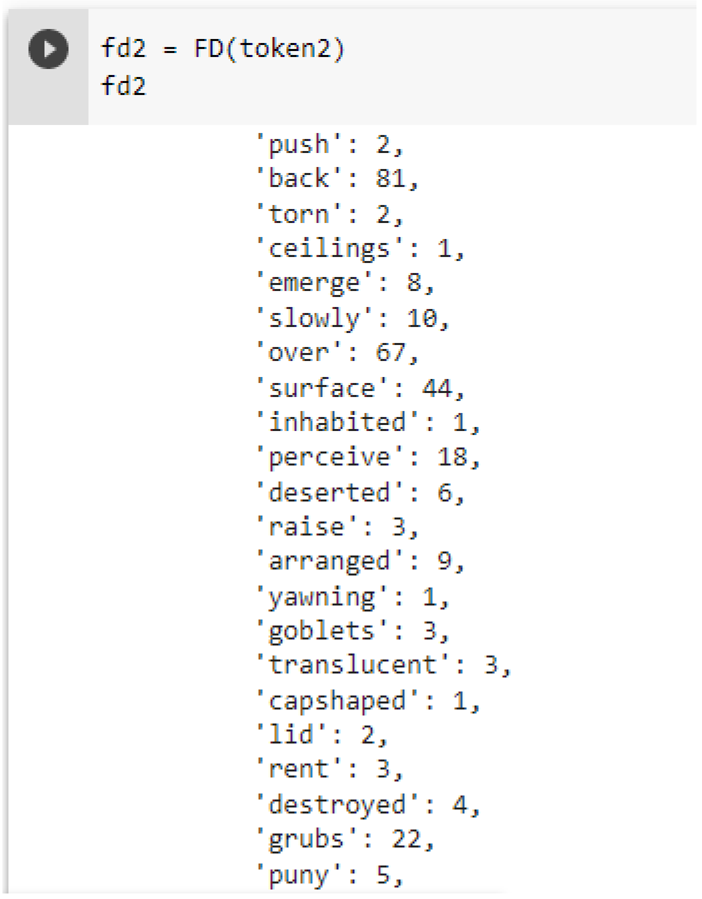
- The output of FreqDist for the T1 tokenized text is:



```
fd1 = FD(token1)
fd1
```

```
FreqDist({'what': 226,
          'uncle': 108,
          'paul': 67,
          'proposes': 2,
          'to': 2182,
          'talk': 11,
          'about': 167,
          'in': 1880,
          'these': 236,
          'talks': 6,
          'that': 1151,
          'we': 243,
          'shall': 35,
          'have': 396,
          'together': 58,
          'said': 119,
          'as': 777,
          'he': 163,
          'sat': 1,
          'with': 930,
          'his': 168,
          'nephews': 2,
          'one': 376,
          'evening': 27,
          'may': 124,
          'under': 91,
          'the': 7113,
          'big': 40,
          'elder': 2,
          'tree': 83,
          'garden': 38,
          'i': 463,
          'propose': 4,
```

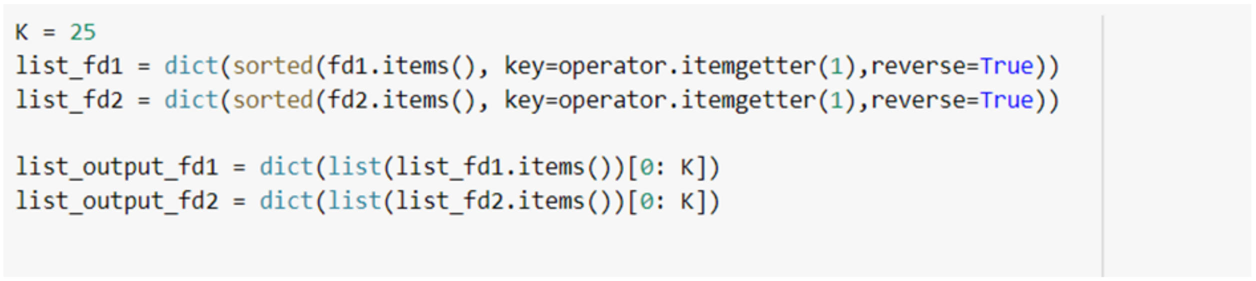
- The output of FreqDist for the T2 tokenized text is:



```
fd2 = FD(token2)
fd2
```

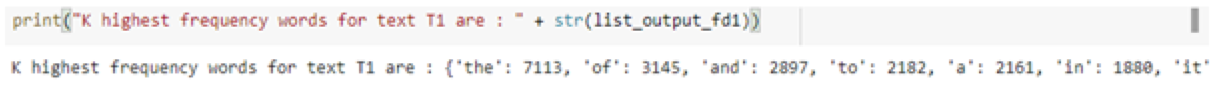
```
                'push': 2,
                'back': 81,
                'torn': 2,
                'ceilings': 1,
                'emerge': 8,
                'slowly': 10,
                'over': 67,
                'surface': 44,
                'inhabited': 1,
                'perceive': 18,
                'deserted': 6,
                'raise': 3,
                'arranged': 9,
                'yawning': 1,
                'goblets': 3,
                'translucent': 3,
                'capshaped': 1,
                'lid': 2,
                'rent': 3,
                'destroyed': 4,
                'grubs': 22,
                'puny': 5,
```

- As there are many words in the tokenized text so sorting them in descending order of their frequencies for making a bar graph. We have restricted the output to the top 25 words.
- Code for the same is:



```
K = 25
list_fd1 = dict(sorted(fd1.items(), key=operator.itemgetter(1),reverse=True))
list_fd2 = dict(sorted(fd2.items(), key=operator.itemgetter(1),reverse=True))

list_output_fd1 = dict(list(list_fd1.items())[0: K])
list_output_fd2 = dict(list(list_fd2.items())[0: K])
```

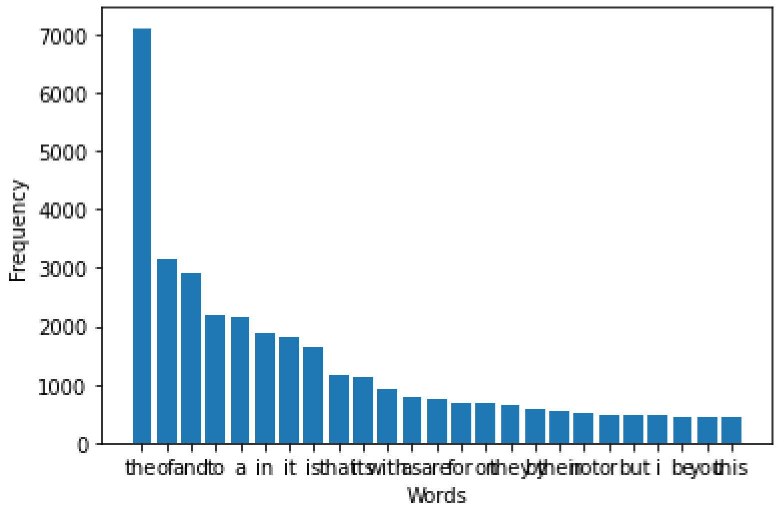
- Output for Text T1:



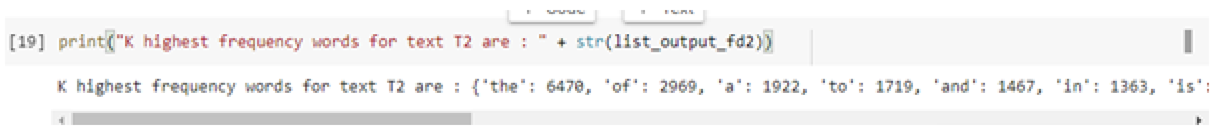
```
print("K highest frequency words for text T1 are : " + str(list_output_fd1))
```

```
K highest frequency words for text T1 are : {'the': 7113, 'of': 3145, 'and': 2897, 'to': 2182, 'a': 2161, 'in': 1880, 'it'
```

- Bar graph for the output(of first 25 elements) of T1 is:



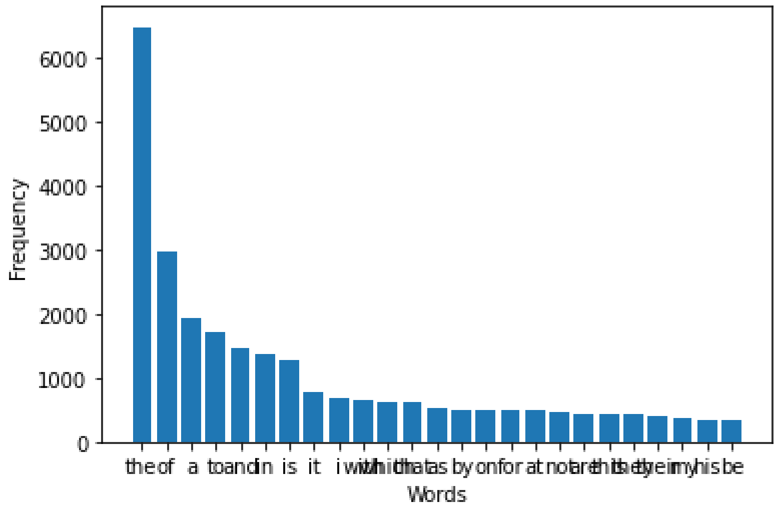
- Output for K highest frequency for text T1 is



```
[19] print("K highest frequency words for text T2 are : " + str(list_output_fd2))
```

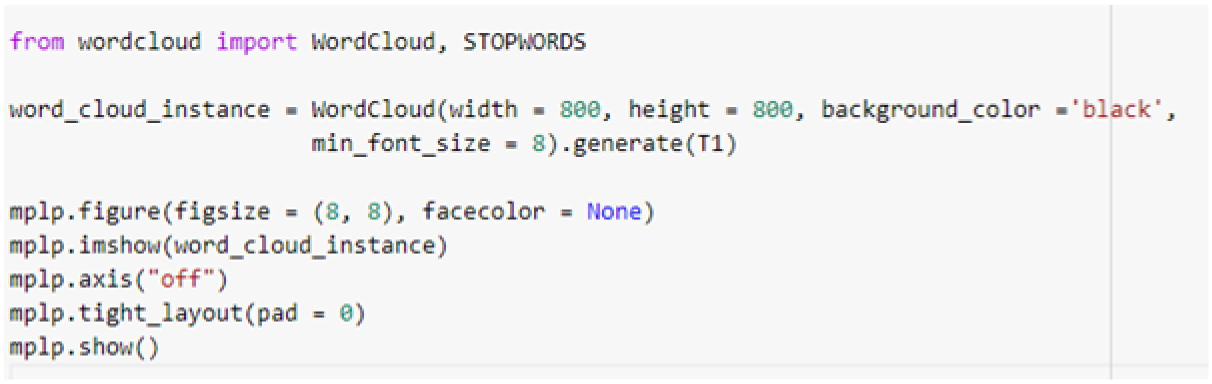
```
K highest frequency words for text T2 are : {'the': 6470, 'of': 2969, 'a': 1922, 'to': 1719, 'and': 1467, 'in': 1363, 'is':
```

- Bar graph for the output(of first 25 elements) of Text T2 is:



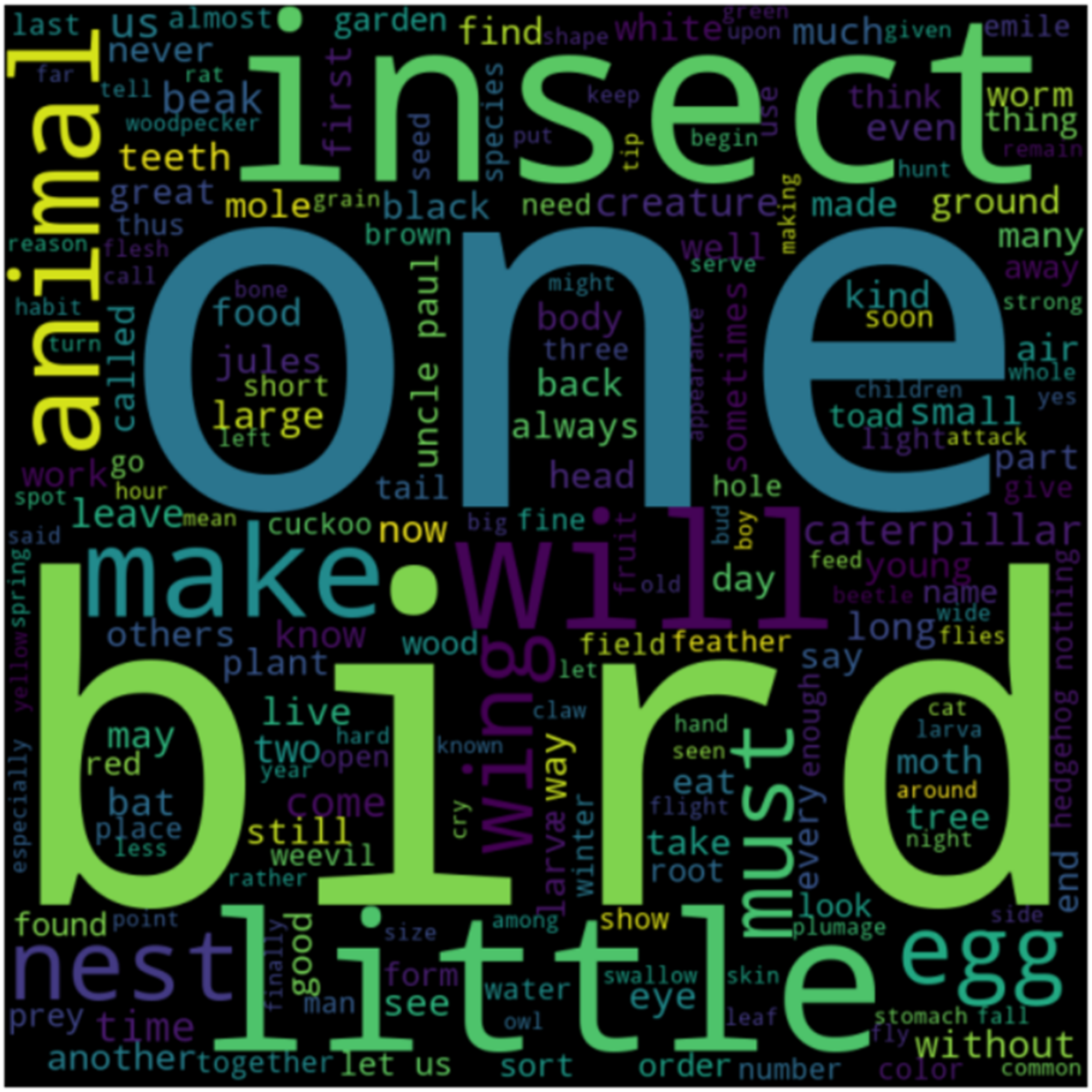
Creating word cloud for the text T1 and text T2

- For creating the word cloud we will be importing ‘wordcloud’ module of python. And than using inbuilt functions to generate the word cloud and the code is in the code snippet below
- **Code :**

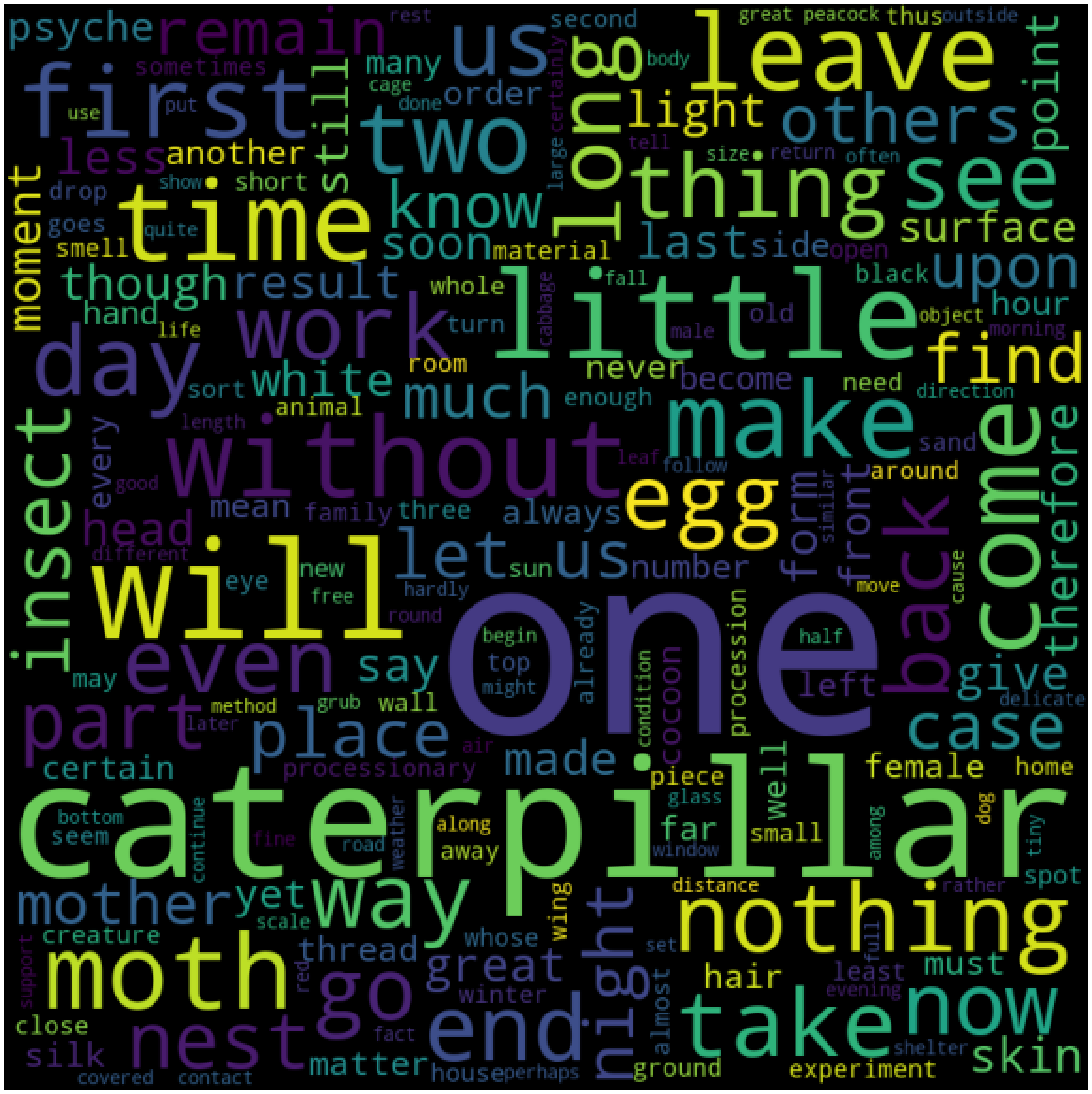


```python
from wordcloud import WordCloud, STOPWORDS

word_cloud_instance = WordCloud(width = 800, height = 800, background_color ='black',
                      min_font_size = 8).generate(T1)

mplp.figure(figsize = (8, 8), facecolor = None)
mplp.imshow(word_cloud_instance)
mplp.axis("off")
mplp.tight_layout(pad = 0)
mplp.show()
```

- **Word cloud for text T1:**



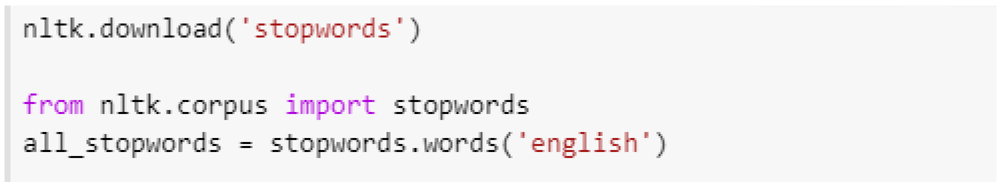
- **Word cloud for text T2:**



Removal of stopwords

- Stopwords are basically the words which give no information helpful for text processing. For our purpose we will use the stopwords of english. For the same we will download the stopwords module from the nltk module.

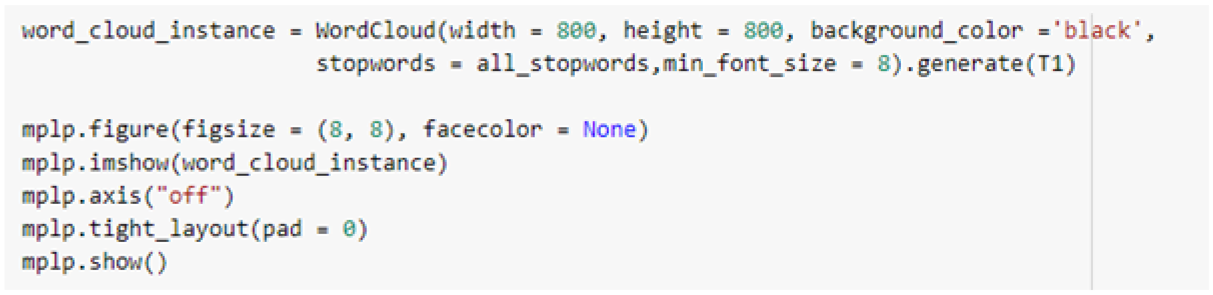
- **Code for the same is:**



```
nltk.download('stopwords')

from nltk.corpus import stopwords
all_stopwords = stopwords.words('english')
```

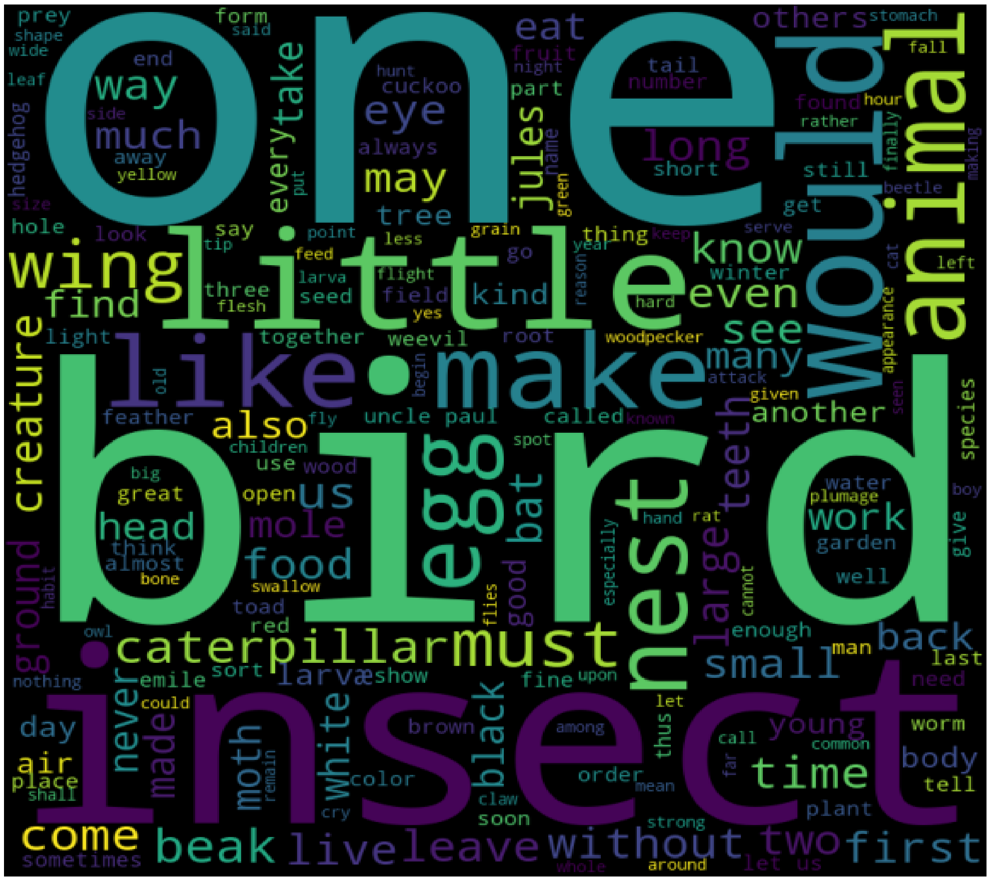
- Wordcloud after removing the stopwords will be formed in the same way as previously formed, just an addition of new attribute of what all the words not to include.

- **Code is:**

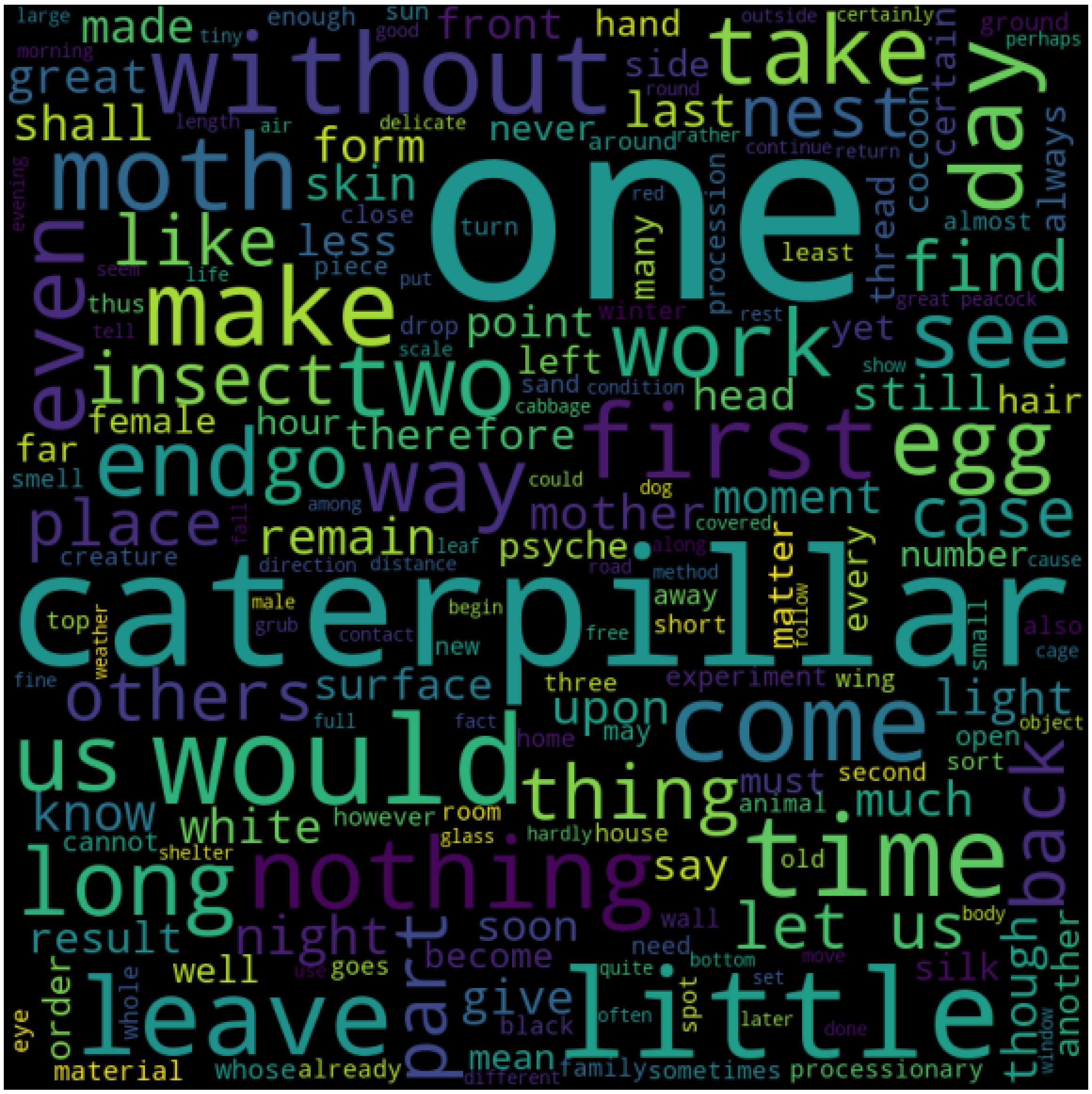


```
word_cloud_instance = WordCloud(width = 800, height = 800, background_color ='black',
                      stopwords = all_stopwords,min_font_size = 8).generate(T1)

mplp.figure(figsize = (8, 8), facecolor = None)
mplp.imshow(word_cloud_instance)
mplp.axis("off")
mplp.tight_layout(pad = 0)
mplp.show()
```

- **Word cloud after removing stopwords for text T1 is:**



- **Word cloud after removing stopwords for text T2 is:**



Relationship between the word length and frequency for both T1 and T2

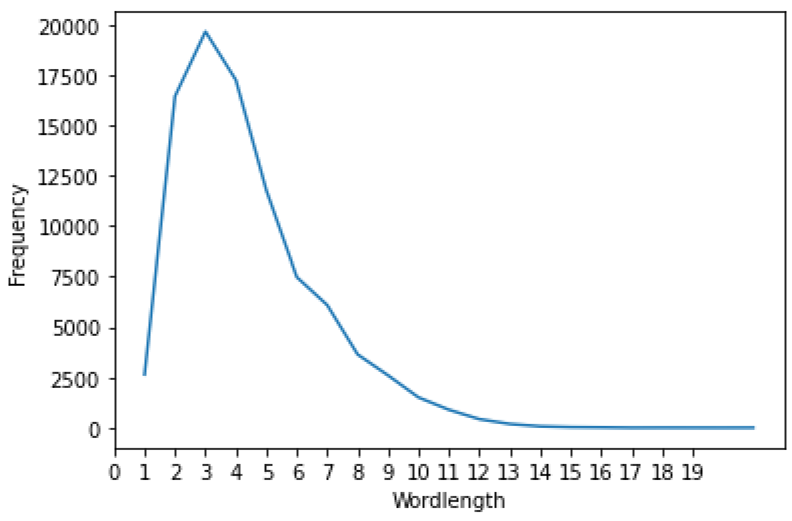
- For this task we have made a function word_counter() which takes the text as input and makes a map, which contains the frequency of words of particular length. We will show the relationship using a line graph.
- Code for the same is

```
from collections import defaultdict

words = {}

def word_counter(text):


    for word in text.split():

        if(len(word) not in words):
          words[len(word)]=1
        else:
          words[len(word)]+=1
```

- **For Text T1-**
    - **Code:**

```
word_counter(T1)

list_count_t1 = sorted(words.items())
x1,y1=zip(*list_count_t1)
mplp.plot(x1,y1)
mplp.xticks(range(0,20))
mplp.rcParams["figure.figsize"] = (10,5)
mplp.xlabel("Wordlength")
mplp.ylabel("Frequency")
mplp.show()
```

- **Output graph is:**

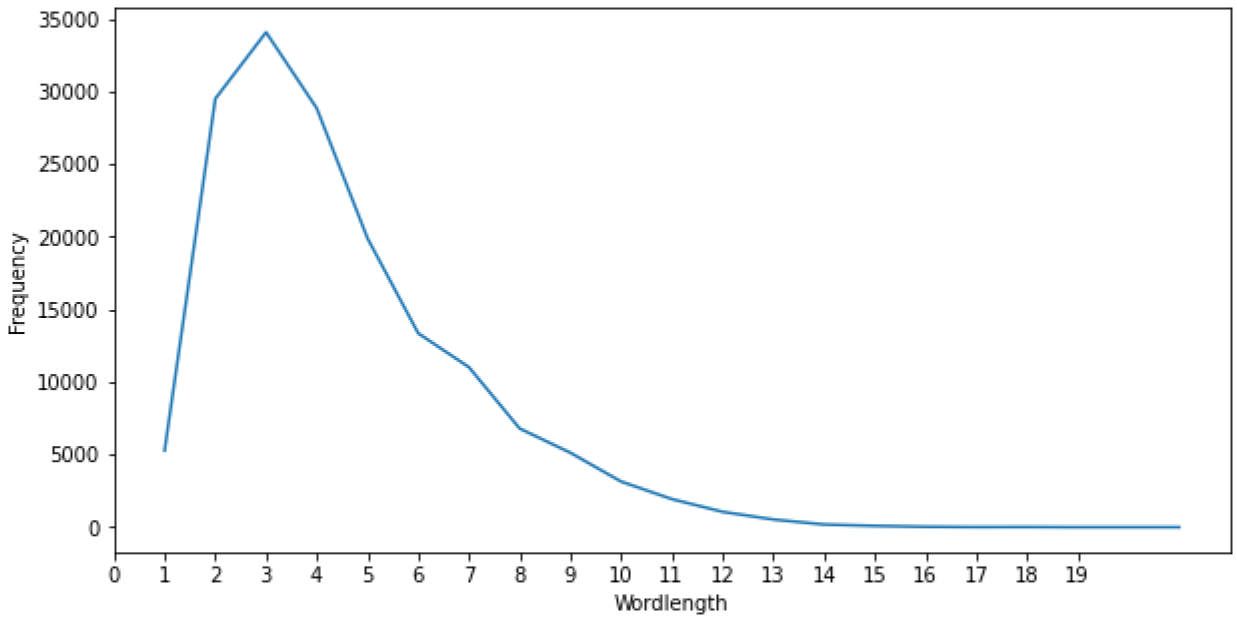


- **For Text T2-**

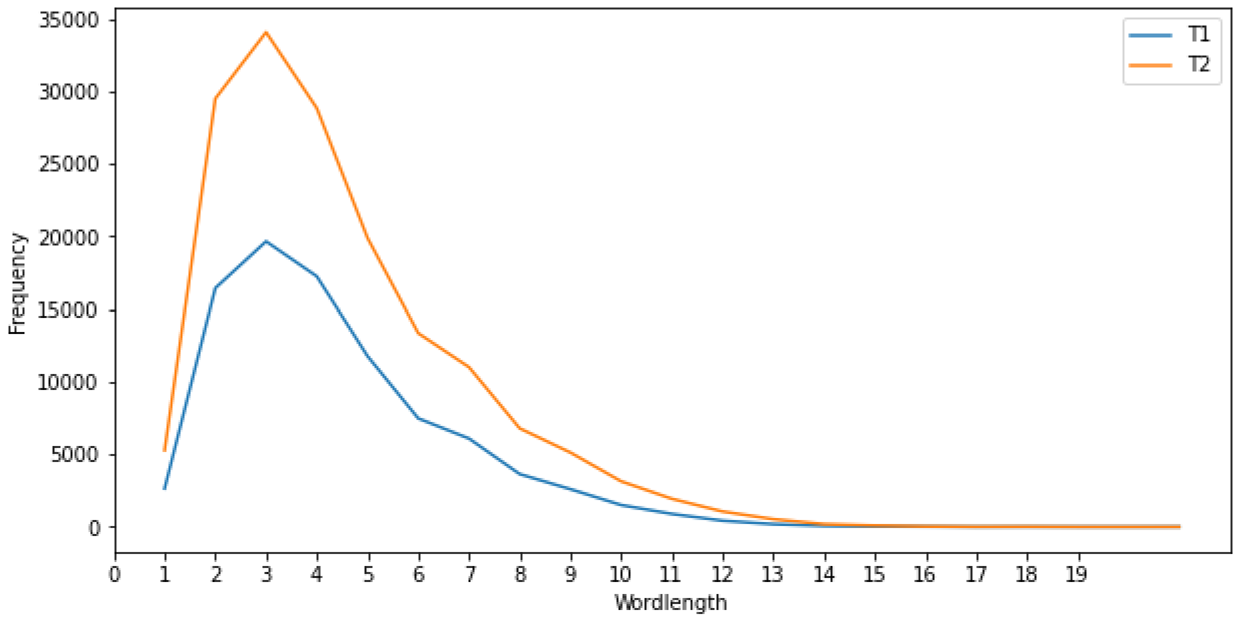
- **Code is :**

```
word_counter(T2)

list_count_t2 = sorted(words.items())
x1,y1=zip(*list_count_t2)
mplp.plot(x1,y1)
mplp.xticks(range(0,20))
mplp.rcParams["figure.figsize"] = (10,5)
mplp.xlabel("Wordlength")
mplp.ylabel("Frequency")
mplp.show()
```

- **Output graph is :**



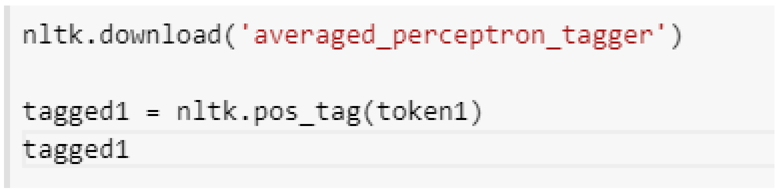
- **Combined graph of both T1 & T2 is :**



PoS Tagging

- Now the main task comes to give each of the tokens their tags. We have used 'averaged_perceptron_tagger' for POS tagging of the tokens. We have downloaded the same from nltk.

- **Code for T1:**



```
nltk.download('averaged_perceptron_tagger')

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

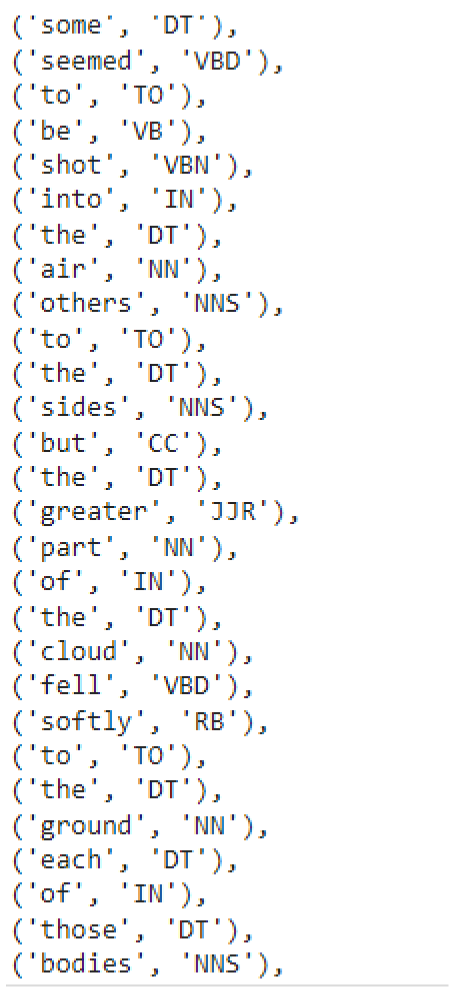
- **Output for T1:**

```
('special', 'JJ'),
('tools', 'NNS'),
('just', 'RB'),
('as', 'RB'),
('does', 'VBZ'),
('any', 'DT'),
('work', 'NN'),
('done', 'VBN'),
('by', 'IN'),
('man', 'NN'),
('now', 'RB'),
('among', 'IN'),
('the', 'DT'),
('innumerable', 'JJ'),
('trades', 'NNS'),
('of', 'IN'),
('animals', 'NNS'),
('there', 'EX'),
('is', 'VBZ'),
('one', 'CD'),
('that', 'WDT'),
('is', 'VBZ'),
('common', 'JJ'),
('to', 'TO'),
('all', 'DT'),
('without', 'IN'),
('exception', 'VBP'),
('the', 'DT'),
```

- **Code for text T2:**

```
tagged2 = nltk.pos_tag(token2)
tagged2
```

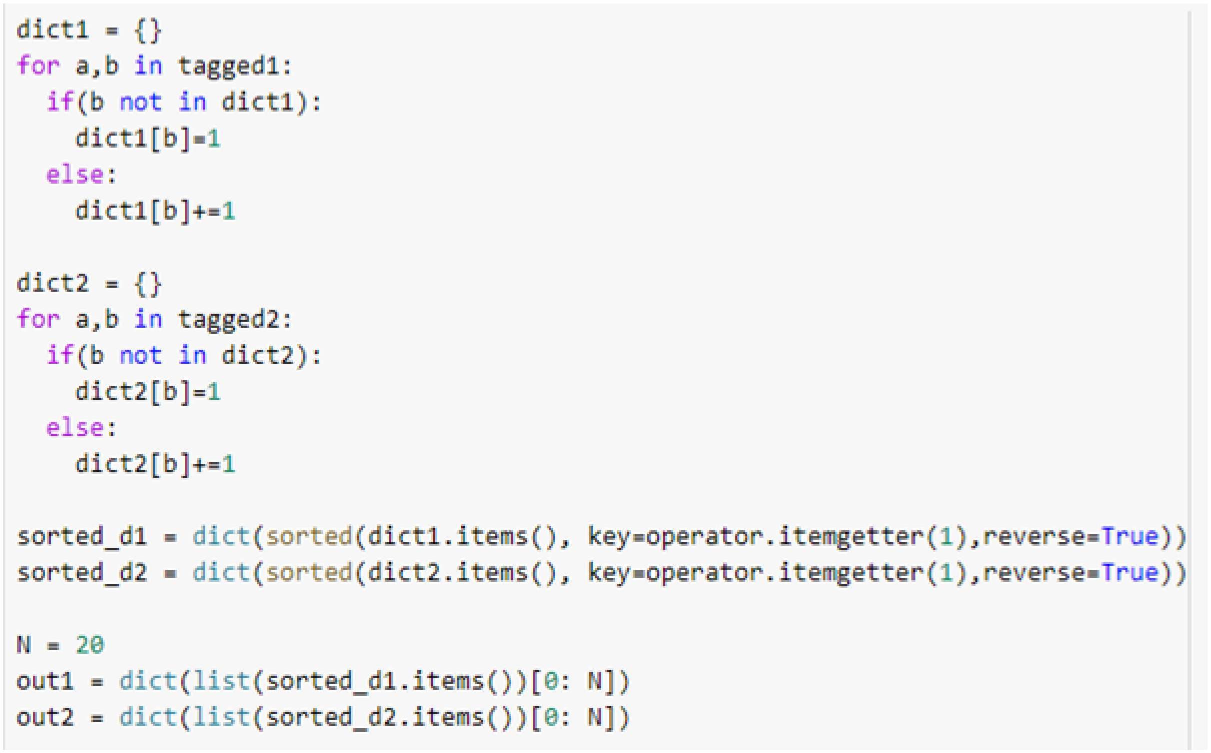
- **Output for T2:**



```
('some', 'DT'),
('seemed', 'VBD'),
('to', 'TO'),
('be', 'VB'),
('shot', 'VBN'),
('into', 'IN'),
('the', 'DT'),
('air', 'NN'),
('others', 'NNS'),
('to', 'TO'),
('the', 'DT'),
('sides', 'NNS'),
('but', 'CC'),
('the', 'DT'),
('greater', 'JJR'),
('part', 'NN'),
('of', 'IN'),
('the', 'DT'),
('cloud', 'NN'),
('fell', 'VBD'),
('softly', 'RB'),
('to', 'TO'),
('the', 'DT'),
('ground', 'NN'),
('each', 'DT'),
('of', 'IN'),
('those', 'DT'),
('bodies', 'NNS'),
```

Distribution of various tags for text T1 and T2 :

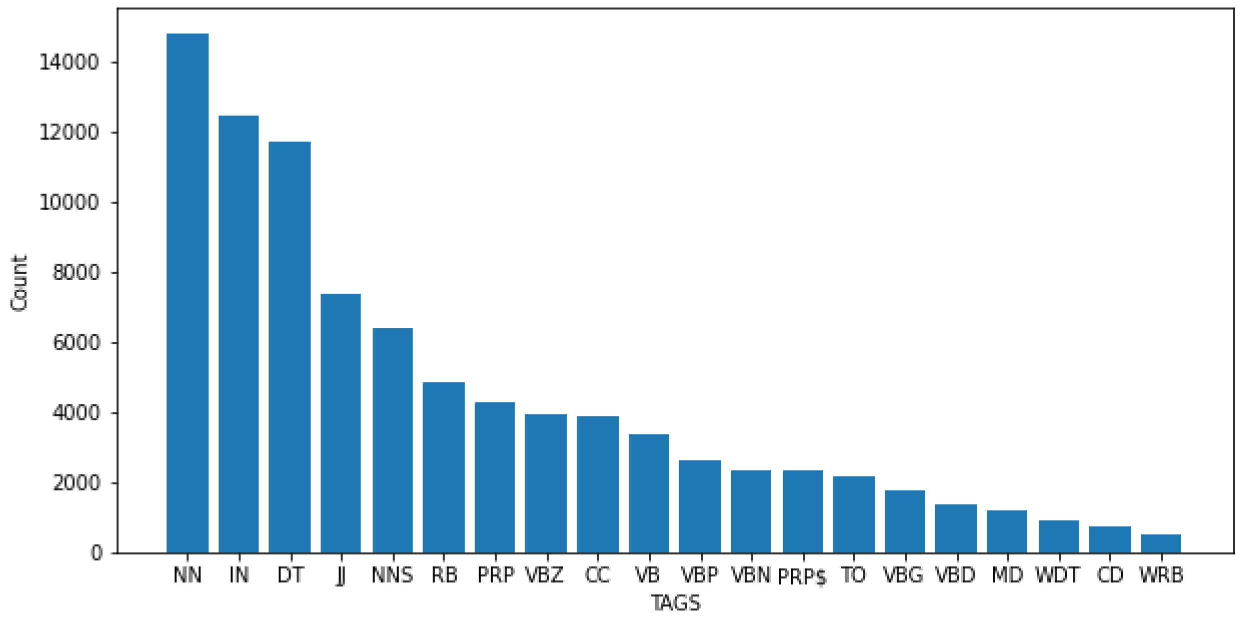
- **Code is:**



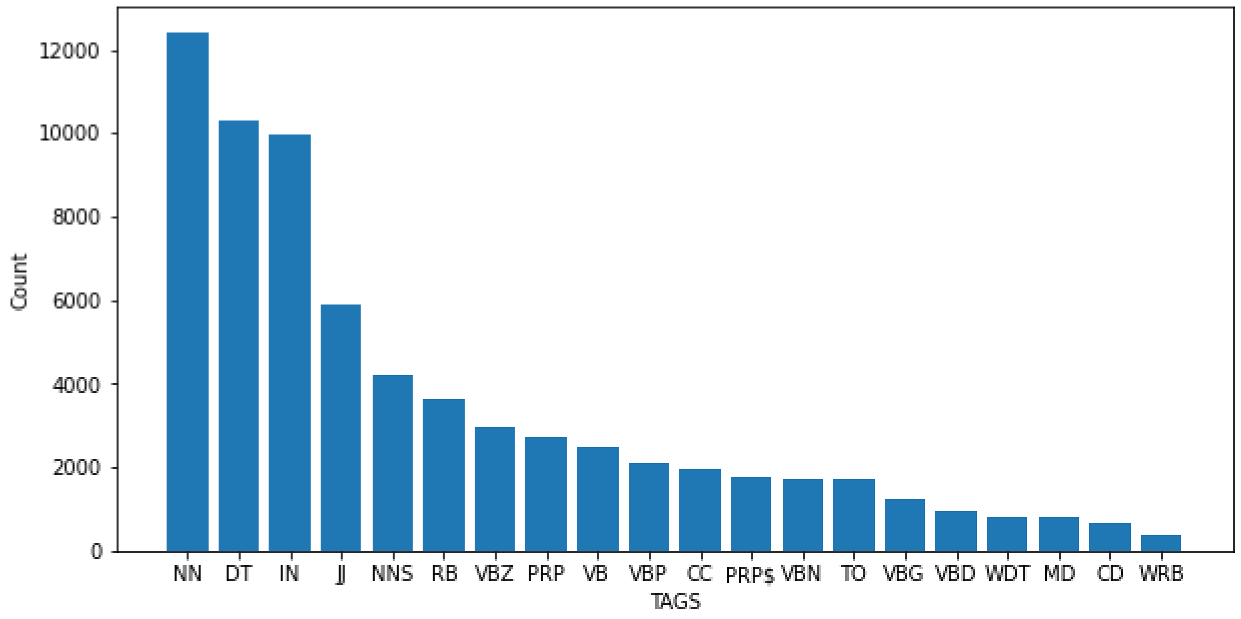
```
dict1 = {}
for a,b in tagged1:
  if(b not in dict1):
    dict1[b]=1
  else:
    dict1[b]+=1

dict2 = {}
for a,b in tagged2:
  if(b not in dict2):
    dict2[b]=1
  else:
    dict2[b]+=1

sorted_d1 = dict(sorted(dict1.items(), key=operator.itemgetter(1),reverse=True))
sorted_d2 = dict(sorted(dict2.items(), key=operator.itemgetter(1),reverse=True))

N = 20
out1 = dict(list(sorted_d1.items())[0: N])
out2 = dict(list(sorted_d2.items())[0: N])
```

- Output of the POS tagging distribution is shown by constructing bar graphs. We have included only the top 20 POS tags and we have sorted it for getting the desired output.

- **Output for T1 :**



- **Output for T2 -**



Conclusions

Through this project, we were able to learn various new concepts such as wordclouds, tokenization, PoS tagging, etc. We also learned about new techniques to implement these concepts and gain insights from the data.

We were also able to derive a few inferences on the texts:

- Smaller words are more frequent as compared to larger words.
- The most common word length is 3. This is evident by frequent occurrences of words like ‘and’, ‘the’, ‘was’, etc.
- Among all the kinds of words used, nouns are the most common among them. It is followed by prepositions and determinants.

NLP Project Round-2

Code Link: https://github.com/Saurav-Somani/NLP_Project/blob/main/NLP_Project.ipynb

Part-1

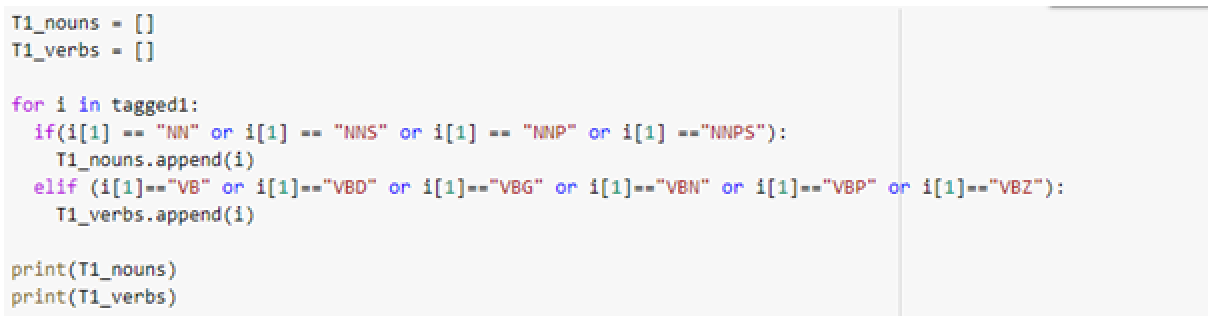
We previously did the PoS tagging of all the tokens. Now from the tagged set, we will make a list of all the nouns and verbs.

There are 4 types of nouns in the PoS tagged set. “NN”, ”NNP”, ”NNPS” & “NNS”.

There are 6 types of verbs in the PoS tagged set. “VB”, ”VBD”, ”VBN”, ”VBG”, ”VBP” & “VBZ”.

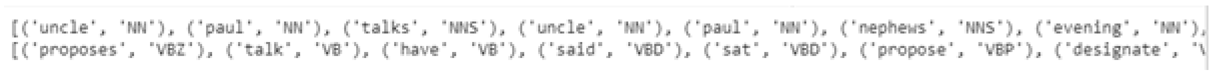
Finding Nouns and Verbs

For finding nouns and verbs , we looped through the whole tagged text and using conditioning we have separated nouns and verbs from the text.For text T1
Code :



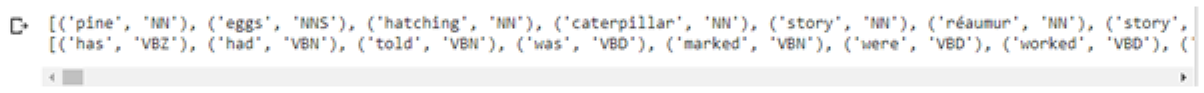
```
T1_nouns = []
T1_verbs = []

for i in tagged1:
  if(i[1] == "NN" or i[1] == "NNS" or i[1] == "NNP" or i[1] =="NNPS"):
    T1_nouns.append(i)
  elif (i[1]=="VB" or i[1]=="VBD" or i[1]=="VBG" or i[1]=="VBN" or i[1]=="VBP" or i[1]=="VBZ"):
    T1_verbs.append(i)

print(T1_nouns)
print(T1_verbs)
```

Output:



```
[('uncle', 'NN'), ('paul', 'NN'), ('talks', 'NNS'), ('uncle', 'NN'), ('paul', 'NN'), ('nephews', 'NNS'), ('evening', 'NN'),
[('proposes', 'VBZ'), ('talk', 'VB'), ('have', 'VB'), ('said', 'VBD'), ('sat', 'VBD'), ('propose', 'VBP'), ('designate', '\
```

Similarly, we did for text T2.Output for it is



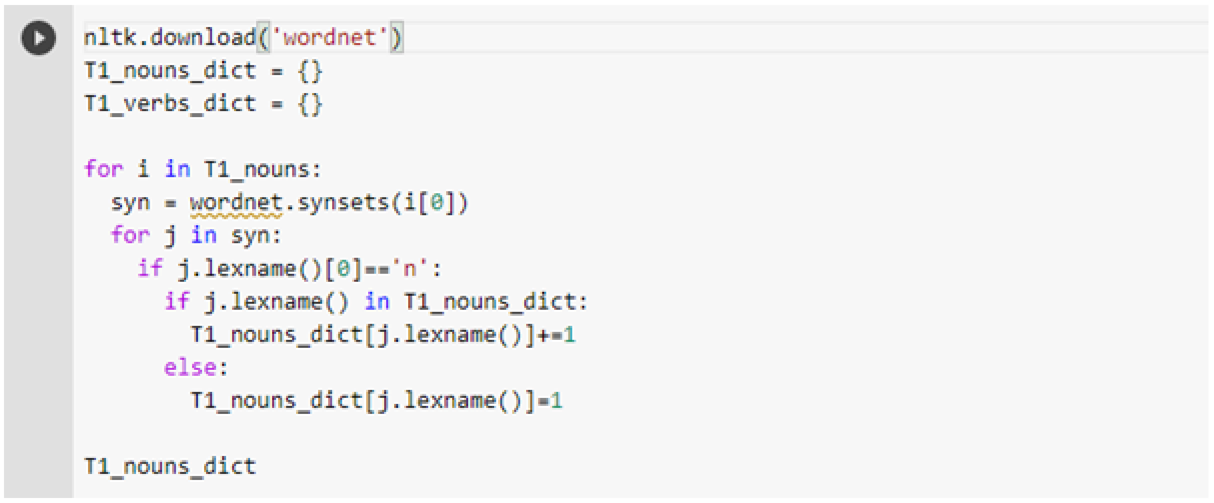
```
[('pine', 'NN'), ('eggs', 'NNS'), ('hatching', 'NN'), ('caterpillar', 'NN'), ('story', 'NN'), ('réaumur', 'NN'), ('story',
[('has', 'VBZ'), ('had', 'VBN'), ('told', 'VBN'), ('was', 'VBD'), ('marked', 'VBN'), ('were', 'VBD'), ('worked', 'VBD'), ('
```

Categorizing and frequencing the nouns and verbs

For this we have used the ‘wordnet’ corpus reader of NLTK library. It helps to categorize the PoS tags into parent tags. It has 25 categories of nouns and 16 of verbs. We have used ‘synset’ for this purpose. Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept.

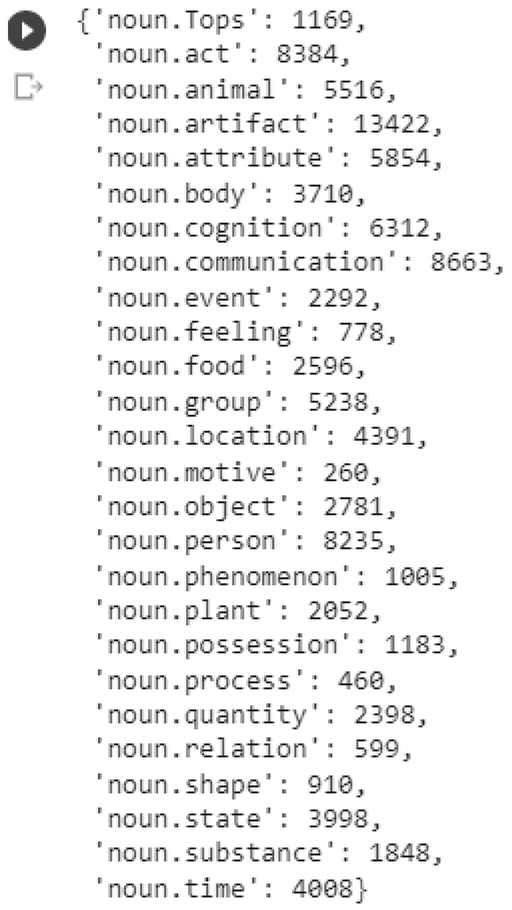
For Text T1:

For Nouns code:



```python
nltk.download('wordnet')
T1_nouns_dict = {}
T1_verbs_dict = {}

for i in T1_nouns:
  syn = wordnet.synsets(i[0])
  for j in syn:
    if j.lexname()[0]=='n':
      if j.lexname() in T1_nouns_dict:
        T1_nouns_dict[j.lexname()]+=1
      else:
        T1_nouns_dict[j.lexname()]=1

T1_nouns_dict
```

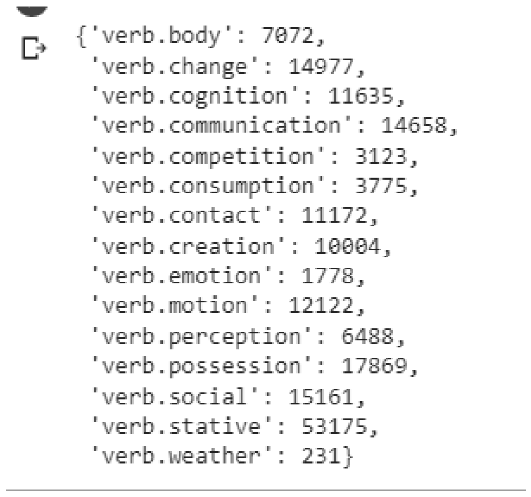
Output:



```
{'noun.Tops': 1169,
 'noun.act': 8384,
 'noun.animal': 5516,
 'noun.artifact': 13422,
 'noun.attribute': 5854,
 'noun.body': 3710,
 'noun.cognition': 6312,
 'noun.communication': 8663,
 'noun.event': 2292,
 'noun.feeling': 778,
 'noun.food': 2596,
 'noun.group': 5238,
 'noun.location': 4391,
 'noun.motive': 260,
 'noun.object': 2781,
 'noun.person': 8235,
 'noun.phenomenon': 1005,
 'noun.plant': 2052,
 'noun.possession': 1183,
 'noun.process': 460,
 'noun.quantity': 2398,
 'noun.relation': 599,
 'noun.shape': 910,
 'noun.state': 3998,
 'noun.substance': 1848,
 'noun.time': 4008}
```

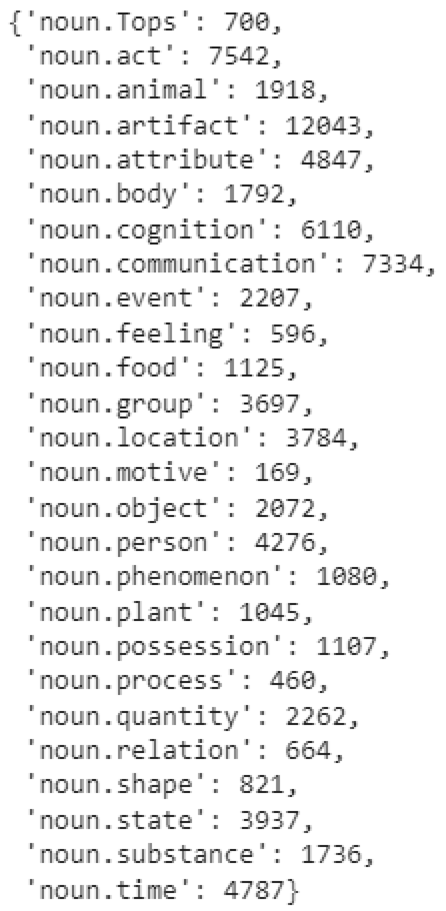
Similar code for verbs just change in one line from " j.lexname()[0] == `n` " to " j.lexname()[0] == `v` ".

Output:



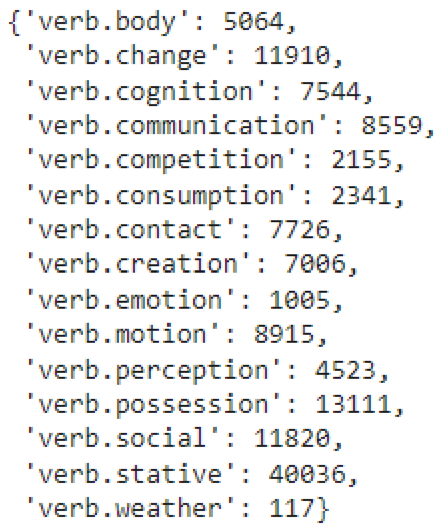
```
{'verb.body': 7072,
 'verb.change': 14977,
 'verb.cognition': 11635,
 'verb.communication': 14658,
 'verb.competition': 3123,
 'verb.consumption': 3775,
 'verb.contact': 11172,
 'verb.creation': 10004,
 'verb.emotion': 1778,
 'verb.motion': 12122,
 'verb.perception': 6488,
 'verb.possession': 17869,
 'verb.social': 15161,
 'verb.stative': 53175,
 'verb.weather': 231}
```

Similarly for Text T2. Output for nouns categorizing:



```
{'noun.Tops': 700,
 'noun.act': 7542,
 'noun.animal': 1918,
 'noun.artifact': 12043,
 'noun.attribute': 4847,
 'noun.body': 1792,
 'noun.cognition': 6110,
 'noun.communication': 7334,
 'noun.event': 2207,
 'noun.feeling': 596,
 'noun.food': 1125,
 'noun.group': 3697,
 'noun.location': 3784,
 'noun.motive': 169,
 'noun.object': 2072,
 'noun.person': 4276,
 'noun.phenomenon': 1080,
 'noun.plant': 1045,
 'noun.possession': 1107,
 'noun.process': 460,
 'noun.quantity': 2262,
 'noun.relation': 664,
 'noun.shape': 821,
 'noun.state': 3937,
 'noun.substance': 1736,
 'noun.time': 4787}
```

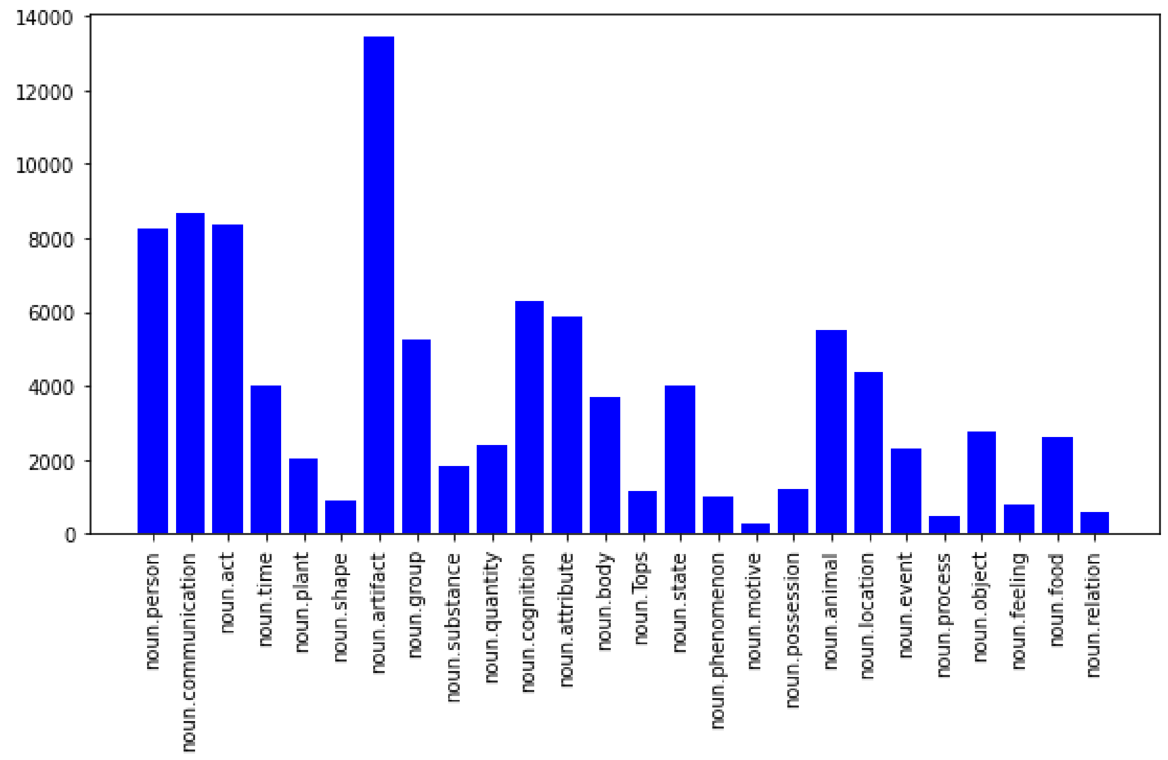
Output for verbs:



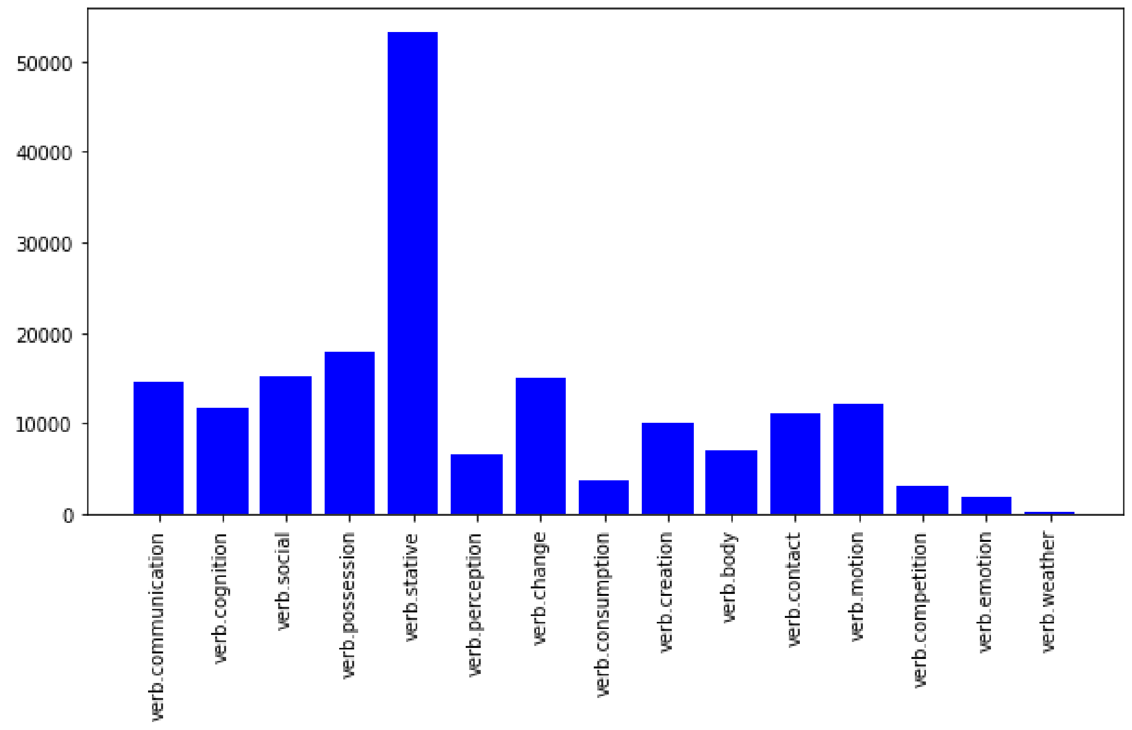
```
{'verb.body': 5064,
 'verb.change': 11910,
 'verb.cognition': 7544,
 'verb.communication': 8559,
 'verb.competition': 2155,
 'verb.consumption': 2341,
 'verb.contact': 7726,
 'verb.creation': 7006,
 'verb.emotion': 1005,
 'verb.motion': 8915,
 'verb.perception': 4523,
 'verb.possession': 13111,
 'verb.social': 11820,
 'verb.stative': 40036,
 'verb.weather': 117}
```

Histograms for novel T1

Nouns:

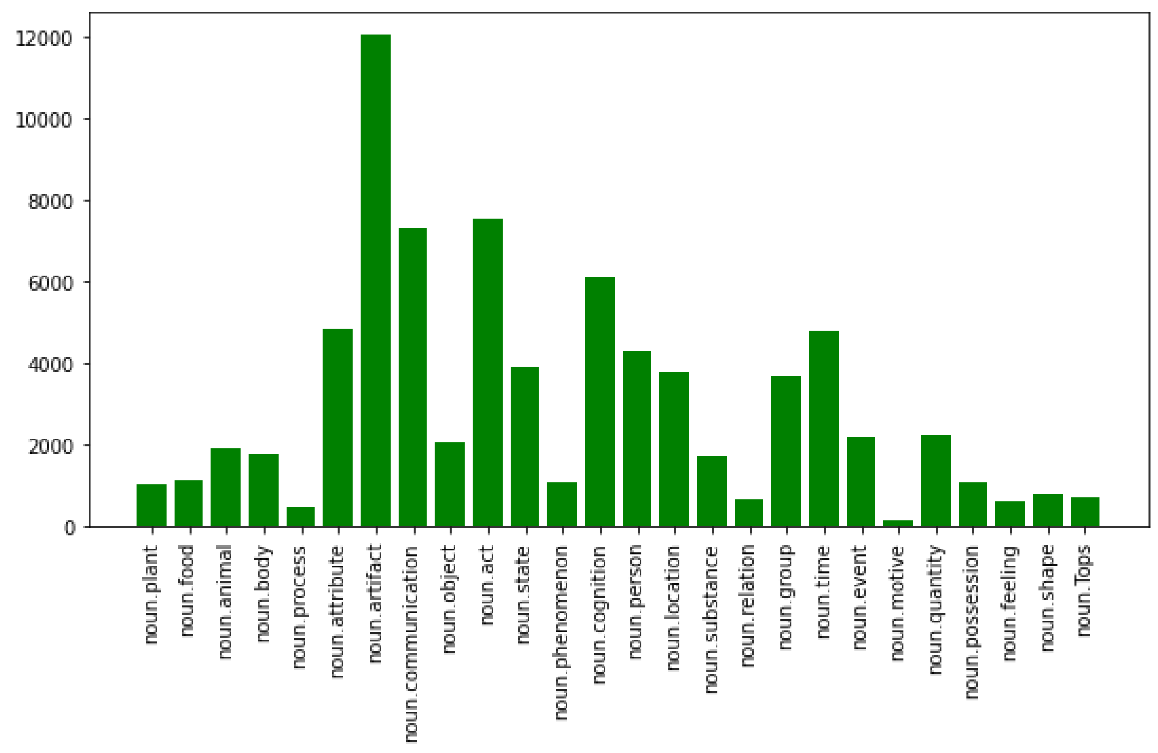


Verbs:



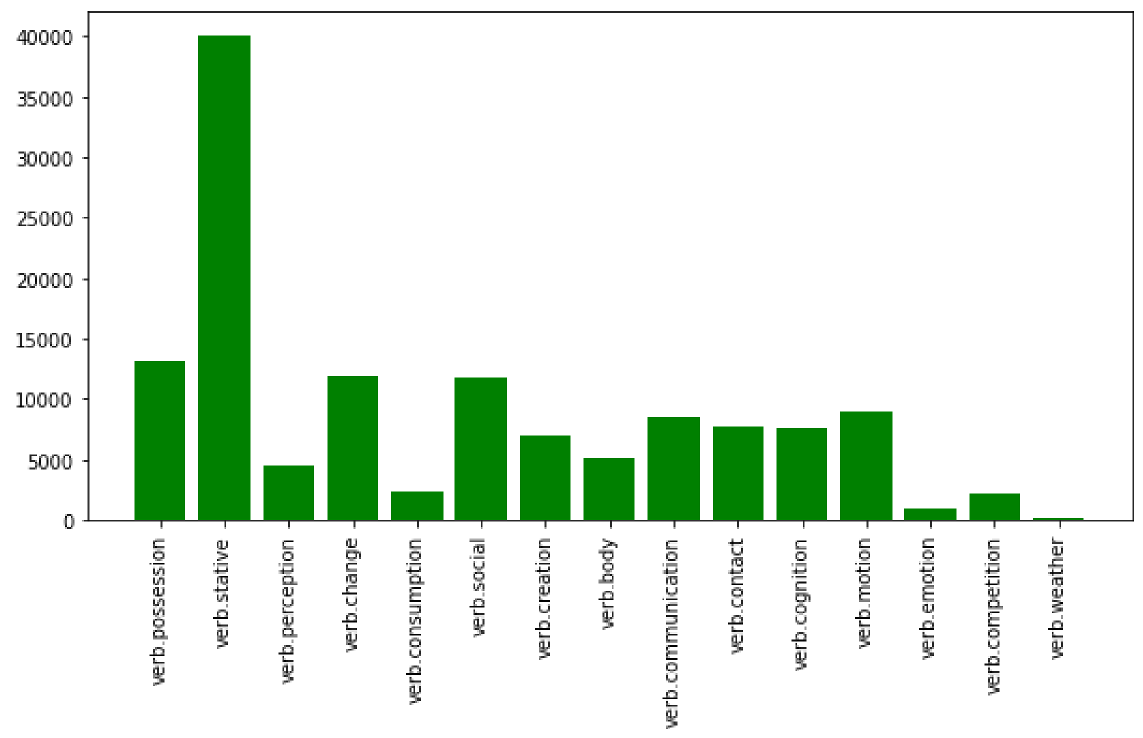
Histograms for novel T2

Nouns:



12000
10000
8000
6000
4000
2000
0
noun.plant
noun.food
noun.animal
noun.body
noun.process
noun.attribute
noun.artifact
noun.communication
noun.object
noun.act
noun.state
noun.phenomenon
noun.cognition
noun.person
noun.location
noun.substance
noun.relation
noun.group
noun.time
noun.event
noun.motive
noun.quantity
noun.possession
noun.feeling
noun.shape
noun.Tops

Verbs:



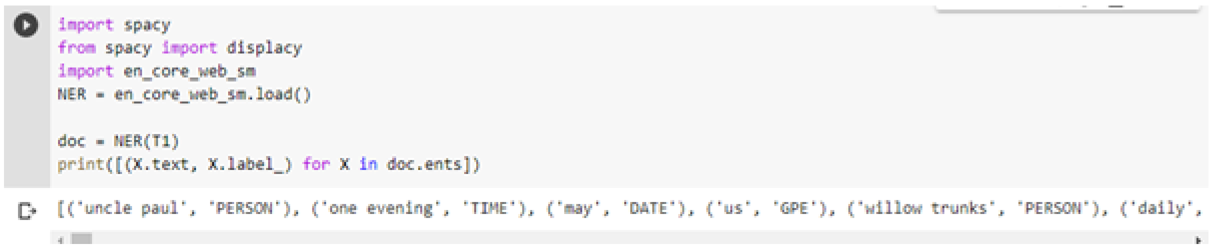
40000
35000
30000
25000
20000
15000
10000
5000
0
verb.possession
verb.stative
verb.perception
verb.change
verb.consumption
verb.social
verb.creation
verb.body
verb.communication
verb.contact
verb.cognition
verb.motion
verb.emotion
verb.competition
verb.weather

Part-2

Named Entity Recognition (NER)

It is a sub part of Information extraction and retrieval. We have used the 'spacy' library of python for the same. Using 'en_core_web_sm.load()' inbuilt function we have named or labelled all the text of a novel.Spacy supports PERSON, NORP (nationalities, religious and political groups), FAC (buildings, airports etc.), ORG (organizations), GPE (countries, cities etc.), LOC (mountain ranges, water bodies etc.), PRODUCT (products), EVENT (event names), WORK_OF_ART (books, song titles), LAW (legal document titles), LANGUAGE (named languages), DATE, TIME, PERCENT, MONEY, QUANTITY, ORDINAL and CARDINAL entities.

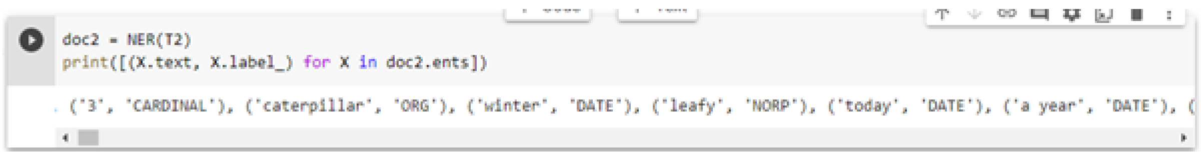
Output for T1:



```
import spacy
from spacy import displacy
import en_core_web_sm
NER = en_core_web_sm.load()

doc = NER(T1)
print([(X.text, X.label_) for X in doc.ents])
```

```
[('uncle paul', 'PERSON'), ('one evening', 'TIME'), ('may', 'DATE'), ('us', 'GPE'), ('willow trunks', 'PERSON'), ('daily',
```

Output for T2:



```
doc2 = NER(T2)
print([(X.text, X.label_) for X in doc2.ents])
```

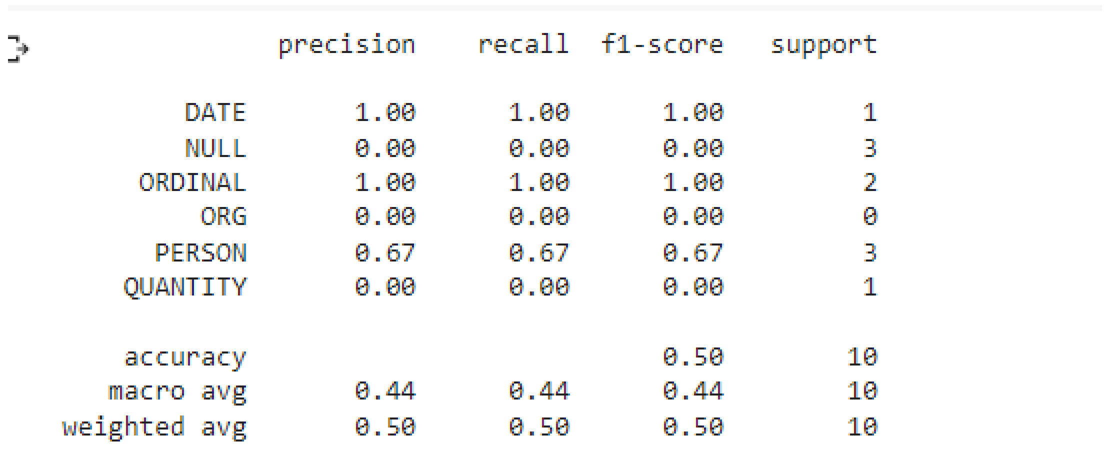
```
. ('3', 'CARDINAL'), ('caterpillar', 'ORG'), ('winter', 'DATE'), ('leafy', 'NORP'), ('today', 'DATE'), ('a year', 'DATE'), (
```

Performance metrics

For finding the accuracy and f-measures we have used the 'metrics' module from the 'sklearn' library of python. It provides all the necessary functionalities needed for calculating the accuracy,recall,precision and f1 measure.

For this purpose we have randomly chosen a passage of code and we have first passed it through Spacy's NER function to get all the entities,the result of this will be the predicted one. For actual labels we have manually labelled the same passage and than using classification_report() function of metrics we have calculated the performance matrix.

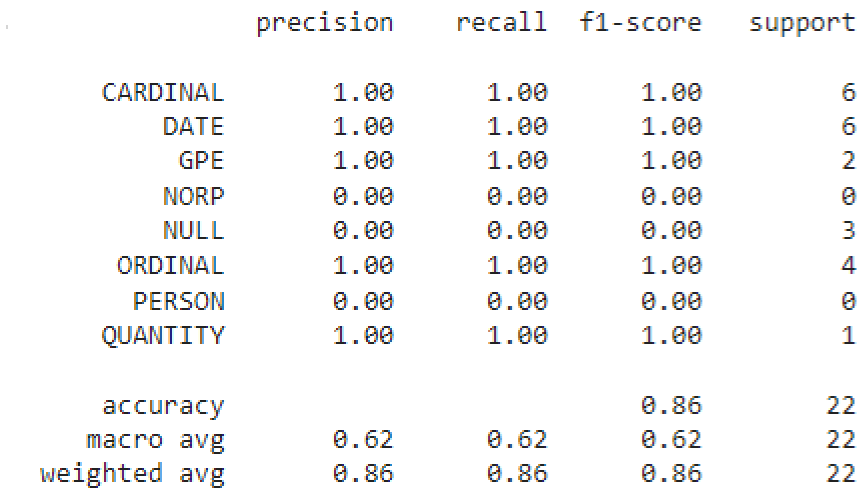
For text T1 the performance metrics are:



```
              precision    recall  f1-score   support

        DATE       1.00      1.00      1.00         1
        NULL       0.00      0.00      0.00         3
     ORDINAL       1.00      1.00      1.00         2
         ORG       0.00      0.00      0.00         0
      PERSON       0.67      0.67      0.67         3
    QUANTITY       0.00      0.00      0.00         1

    accuracy                           0.50        10
   macro avg       0.44      0.44      0.44        10
weighted avg       0.50      0.50      0.50        10
```

Accuracy is 50%.

For text T2 the performance metrics are:



```
              precision    recall  f1-score   support

    CARDINAL       1.00      1.00      1.00         6
        DATE       1.00      1.00      1.00         6
         GPE       1.00      1.00      1.00         2
        NORP       0.00      0.00      0.00         0
        NULL       0.00      0.00      0.00         3
     ORDINAL       1.00      1.00      1.00         4
      PERSON       0.00      0.00      0.00         0
    QUANTITY       1.00      1.00      1.00         1

    accuracy                           0.86        22
   macro avg       0.62      0.62      0.62        22
weighted avg       0.86      0.86      0.86        22
```

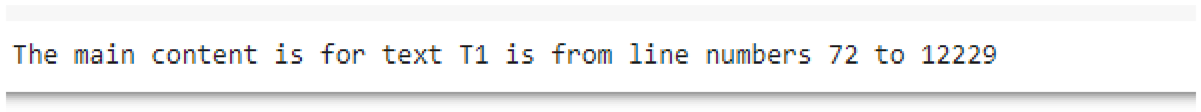
Accuracy is 86%.

Part-3

Adding the third book

The Third book is GreenSea Island by Victor Bridges.

Main content is

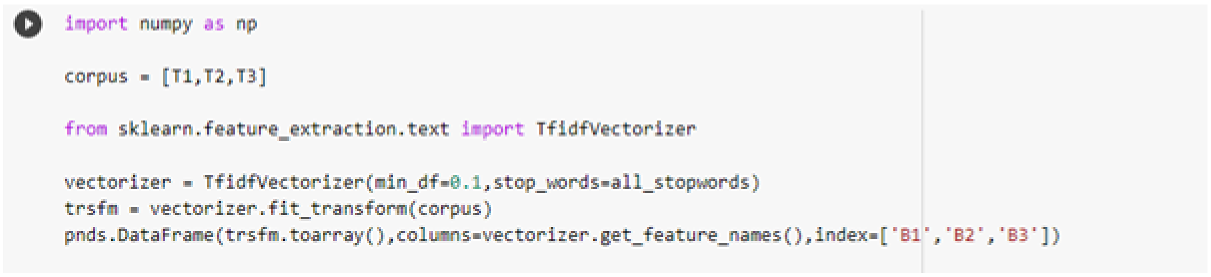


```
The main content is for text T1 is from line numbers 72 to 12229
```

Tf-Idf Vectorization and Cosine Similarity

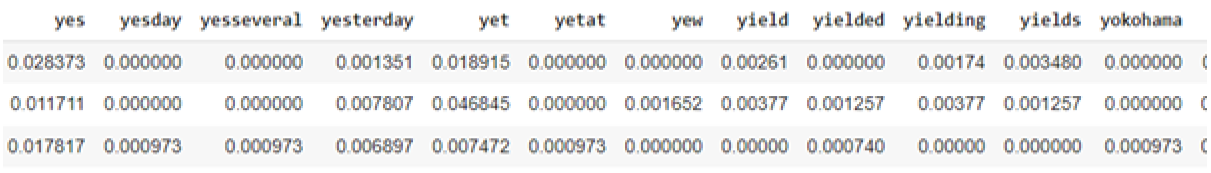
For this task, we will be using the ‘numpy’ library and the module ‘sklearn.feature_extraction’. Sklearn.feature_extraction is used for embedding the feature vectors. It provides different types of embedding techniques like TF-IDF,CountVectorizer,Word2Vec etc. We will be using the Tfidfvectorizer class of the sklearn.feature_extraction. We have removed all the stopwords during the vectorization as they are not useful for finding similarity between 2 texts. We have computed it in a form of matrix and will process it in that form only using Pandas.DataFrame.

Code:



```
import numpy as np

corpus = [T1,T2,T3]

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=0.1,stop_words=all_stopwords)
trsfm = vectorizer.fit_transform(corpus)
pnds.DataFrame(trsfm.toarray(),columns=vectorizer.get_feature_names(),index=['B1','B2','B3'])
```

Output:



```
     yes    yesday  yesseveral  yesterday       yet     yetat       yew    yield   yielded  yielding    yields  yokohama
0.028373  0.000000    0.000000   0.001351  0.018915  0.000000  0.000000  0.00261  0.000000   0.00174  0.003480  0.000000
0.011711  0.000000    0.000000   0.007807  0.046845  0.000000  0.001652  0.00377  0.001257   0.00377  0.001257  0.000000
0.017817  0.000973    0.000973   0.006897  0.007472  0.000973  0.000000  0.00000  0.000740   0.00000  0.000000  0.000973
```

The matrix trsfm contains 3 rows and 16039 columns.

For computing cosine similarity we will use the inbuilt function cosine_similarity() from sklearn.metrics.pairwise class. Cosine similarity is the measure of similarity of the 2 documents.

Similarity results are as:

B1 - B2 = 0.6769565

B2 - B3 = 0.50249855

B1 - B3 = 0.47390829

By this we come to know that books B1 and B2 are very similar while B1 and B3 are least similar.

Lemmatizing the texts

Lemmatization is the algorithmic process of finding the lemma of a word depending on its meaning. Lemmatization usually refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word, which is known as the lemma.We have used the ‘WordNetLemmatizer’ module for lemmatization.

After lemmatizing all the three books we have again created their Tfidf vector matrix and there is quite a significant change. The number of columns has been reduced from 16039 to 14142.

Output for similarity matrix after this is :

B1 - B2 = 0.68816024

B1 - B3 = 0.40573885

B2 - B3 = 0.43669932

We can say that this is true as B1 and B2 both the books are related to animals and both are written by the same author while the last one is the book based on the island.

Conclusions

Through this project we are able to learn some very advanced concepts like Wordnet, how to use it, Named Entity Recognition and vectorization. We learnt their practical applications,when and how they are used.

Overall we learnt a lot of new things and practical application of the concepts learnt in NLP course.

References

- https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba
- https://www.analyticsvidhya.com/blog/2021/05/how-to-build-word-cloud-in-python/
- https://www.einfochips.com/blog/nlp-text-vectorization/
- https://www.educative.io/edpresso/how-to-create-a-confusion-matrix-in-python-using-scikit-learn
- https://plotly.com/python/