NLP PROJECT

TEAM NAME : DELTA

S.NO	TEAM MEMBER	ROLL NUMBERS
1	ARAV KHANDELWAL	18UCS140
2	HEMAKSHI MANCHANDIA	18UCS141
3	ROHIT AGRAWAL	18UCS094
4	TOSHI MUDGAL	18UCS064

Submitted To: Dr. Sakthi Balan

Round - 1

LINK FOR CODE REPOSITORY : https://github.com/toshimudgal/NLP-project-round-1-
This repository only contains Round-1 code. Round 2 code link can be found at (page no: 26) of this document.

Below is the code along with the problem statement , explanation and required outputs (due to the size of output at some places we have shown a part of output only) . We have used Google Collab as a platform to write our code for the same.

Data Description

We have installed the following books from Link - http://www.gutenberg.org

1. A Tale of Two Cities, by Charles Dicken(T1) - https://dev.gutenberg.org/ebooks/98
2. Pride and Prejudice, by Jane Austen(T2) - https://dev.gutenberg.org/ebooks/1342

Size of T1 is 770 KB
Size of T2 is 767 KB

```
twords = text1.split()

print('Number of words in book 1 :', len(twords))
```

```
twords2 = text2.split()

print('Number of words in book 2 :', len(twords2))
```

Word Count of T1 : 141441
Word Count of T2: 125854

```
number_of_characters = len(text1)

print('Number of characters in Book 1 with spaces :', number_of_characters)
```

```
number_of_characters = len(text2)
```

```
print('Number of characters in Book 2 with spaces :', number_of_characters)
```

Character count(including whitespaces) T1 : 775953
Character count(including whitespaces) T2 : 773396

```
data = text1.replace(" ","")

#get the length of the data
number_of_characters = len(data)

print('Number of characters in Book 1 without spaces :', number_of_characters)
```

```
data = text2.replace(" ","")

number_of_characters = len(data)

print('Number of characters in Book 2 without spaces :', number_of_characters)
```

Character count(excluding whitespaces) T1 : 599602
Character count(excluding whitespaces) T2 : 551370

```
num_lines = 0
with open('/content/drive/My Drive/python/nlp project/A Tale of Two Cities by
Charles Dickens.txt', 'r') as f:
    for line in f:
        num_lines += 1
print("Number of lines in book 1:")
print(num_lines)
```

```
num_lines = 0
with open('/content/drive/My Drive/python/nlp project/Pride and Prejudice by
Jane Austen.txt' ,'r') as f:
    for line in f:
        num_lines += 1
print("Number of lines in book 2:")
print(num_lines)
```

Lines Count of T1: 16272
Lines Count of T2: 14593

Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import nltk
```

1. We have import matplotlib.pyplot library for plotting the graph
2. The Natural Language Toolkit (NLTK) contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.
3. We have import pandas library for data analysis

➢ Import the text, as T1 and T2 (books that you have downloaded)

Import Book

```
f= open('/content/drive/My Drive/python/nlp project/A Tale of Two Cities by Charles Dickens.txt', 'r')
text1=f.read()
```

- Opening T1(A Tale of Two Cities by Charles Dickens) in read mode in f and assigning it to variable text1.

```
k= open('/content/drive/My Drive/python/nlp project/Pride and Prejudice by Jane Austen.txt', 'r')
text2=k.read()
```

- Opening T2(Pride and Prejudice by Jane Austen) in read mode in k and assigning it to variable text2.

Connecting Google collab to Drive

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

- By this we are connecting gdrive to google collab so that we can further use the text files uploaded on gdrive . Our drive is now mounted onto /content/drive/MyDrive.

➢ Perform simple text pre-processing steps

Data Preprocessing

```
import nltk
nltk.download('punkt')

import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
import re
import string
def text_cleaning(text):
  # to remove special chars
  text= text.lower()
  text= re.sub('\[.*?\]', '', text)
  text = re.sub("\\W"," ",text)
  # to remove links
  text = re.sub('https?://\S+|www\.\S+', '', text)
  #to remove punctuations
  text = re.sub('<.*?>+', '', text)
  text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
  text = re.sub('\n', '', text)
  text = re.sub('\w*\d\w*', '', text)
  text = re.sub("chapter ","",text)
  return text
```

- We have performed various data pre-processing steps by creating a function text_cleaning(text):
    - Converting into lower case
    - Removed some special characters like : \[.*?\]
    - Removed any character which is not a Unicode word character
    - Removed links in the form: https?://\S+|www\.\S+
    - Removed other punctuation marks etc.
- We have found that word “chapter” is a **running word** so we removed it by using python code: `re.sub("chapter ","",text)`

```
text1=text_cleaning(text1)
text2=text_cleaning(text2)
```

- We create two variables text1 and text2 to store the result of function text_cleaning(text).

```
print(text1)
```

```
the project gutenberg ebook of a tale of two cities  by charles dickens
this ebook is for the use of anyone anywhere at no cost and with almost no
restrictions whatsoever   you may copy it  give it away or re use it under
the terms of the project gutenberg license included with this ebook or
online at www gutenberg org   title  a tale of two cities        a story of
the french revolution  author  charles dickens  release date  january
posting date  november    last updated  march     language  english
character set encoding  utf       start of this project gutenberg ebook a
tale of two cities         produced by judith boss         a tale of two
cities  a story of the french revolution  by charles dickens   contents
book the first  recalled to life       i      the period      ii     the
mail      iii    the night shadows      iv     the preparation      v
the wine shop      vi     the shoemaker        book the second  the golden
thread       i      five years later
```

```
print(text2)
```

```
the project gutenberg ebook of pride and prejudice  by jane austen  this ebook
is for the use of anyone anywhere at no cost and with almost no restrictions
whatsoever   you may copy it  give it away or re use it under the terms of the
project gutenberg license included with this ebook or online at www gutenberg
org   title  pride and prejudice  author  jane austen  release date  august
last updated  november      language  english  character set encoding  utf
start of this project gutenberg ebook pride and prejudice         produced by
anonymous volunteers  and david widger  there is an illustrated edition of this
title which may viewed at ebook   cover           pride and prejudice        by
jane austen          contents
```

➢ Tokenize the text T1 and T2

```
token1=word_tokenize(text1)
token2=word_tokenize(text2)
```

- We have called predefined function word_tokenize which is included in nltk library and created 2 variables token1 and token2 to store the result of the above function.

```
token1
```

```
['the',
 'project',
 'gutenberg',
 'ebook',
 'of',
 'a',
 'tale',
 'of',
 'two',
 'cities',
 'by',
 'charles',
 'dickens',
 'this',
 'ebook',
 'is',
 'for',
 'the',
 'use',
 'of',
 'anyone',
 'anywhere',
 'at',
 'no',
 'cost',
 'and',
 'with',
 'almost',
 'no',
 'restrictions',
 'whatsoever',
 'you',
 'may',
 'copy',
 'it',
 'give',
 'it',
 'away',
 'Or',......]
```

token2

```
['the',
 'project',
 'gutenberg',
 'ebook',
 'of',
 'pride',
 'and',
 'prejudice',
 'by',
 'jane',
 'austen',
 'this',
 'ebook',
 'is',
 'for',
 'the',
 'use',
 'of',
 'anyone',
 'anywhere',
 'at',
 'no',
 'cost',
 'and',
 'with',
 'almost',
 'no',
 'restrictions',
 'whatsoever',
 'you',
 'may',
 'copy',
 'it',
 'give',
 'it',
 'away',
 'or',
 're',.....]
```

➢ Analyze the frequency distribution of tokens in T1 and T2 separately

```
from nltk.probability import FreqDist
fdist1 = FreqDist(token1)
fdist2 = FreqDist(token2)
```

- We have imported FreqDist library from nltk.probability for finding frequency distribution of token.
- Then, we have called a predefined function FreqDist(token) for frequency distribution and created 2 variables fdist1 and fdist2 to store the output of the same.

```
fdist1
```

```
FreqDist({'the': 8230,
          'project': 87,
          'gutenberg': 93,
          'ebook': 10,
          'of': 4139,
          'a': 3017,
          'tale': 7,
          'two': 230,
          'cities': 7,
          'by': 595,
          'charles': 103,
          'dickens': 4,
          'this': 601,
          'is': 853,
          'for': 987,
          'use': 31,
          'anyone': 5,
          'anywhere': 6,
          'at': 1050,
          'no': 553,
          'cost': 6,
          'and': 5067,
          'with': 1358,
          'almost': 49,
          'restrictions': 2,
```

```
          'whatsoever': 3,
          'you': 1497,
          'may': 142,
          'copy': 12,
          'it': 2082,
          'give': 50,
          'away': 137,
          'or': 442,
          're': 13,
          'under': 148,
          'terms': 26,
          'license': 17,
          'included': 5,
          'online': 4,....})
```

```
fdist2
```

```
FreqDist({'the': 4507,
          'project': 88,
          'gutenberg': 93,
          'ebook': 11,
          'of': 3731,
          'pride': 53,
          'and': 3660,
          'prejudice': 11,
          'by': 658,
          'jane': 295,
          'austen': 4,
          'this': 496,
          'is': 887,
          'for': 1086,
          'use': 28,
          'anyone': 29,
          'anywhere': 3,
          'at': 803,
          'no': 500,
          'cost': 7,
          'with': 1100,
          'almost': 61,
          'restrictions': 2,
          'whatsoever': 2,
          'you': 1428,
          'may': 209,
          'copy': 12,
          'it': 1550,
          'give': 127,
```

```
          'away': 120,
          'or': 377,
          're': 6,
          'under': 39,
          'terms': 43,.....})
```

- As the output of the function FreqDist(token) is in dictionary data type , so we can directly use the dictionary data type in our further commands

```
import operator
sorted_d1 = dict( sorted(fdist1.items(),
key=operator.itemgetter(1),reverse=True))
sorted_d2 = dict( sorted(fdist2.items(),
key=operator.itemgetter(1),reverse=True))
```

- We sorted both dict fdist1 and fdist2 according to values in descending order.
- Due to the enormous size of output we decided to limit(slicing) our dictionary to the first 20 elements and construct a bar graph for the dictionary.

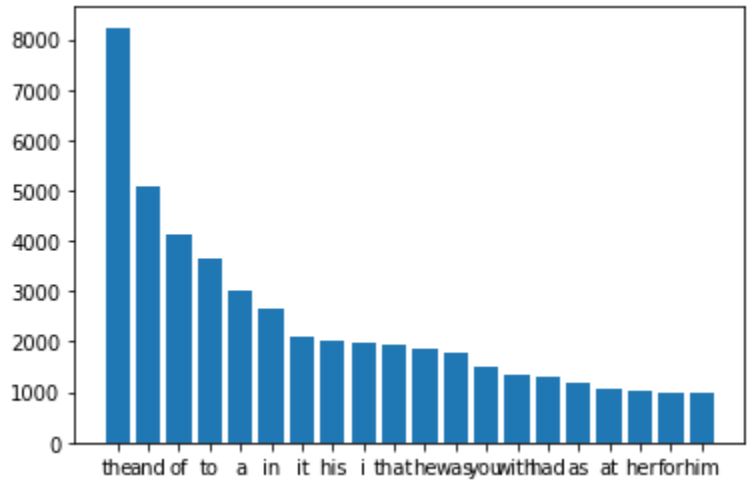
```
K =20
# Using items() + list slicing
# Get first K items in dictionary
out1 = dict(list(sorted_d1.items())[0: K])
out2 = dict(list(sorted_d2.items())[0: K])
# printing result
print("Dictionary limited by K is : " + str(out1))
print("Dictionary limited by K is : " + str(out2))
```

Output-

```
Dictionary limited by K is : {'the': 8230, 'and': 5067, 'of': 4139, 'to': 3651,
'a': 3017, 'in': 2660, 'it': 2082, 'his': 2011, 'i': 1990, 'that': 1956, 'he':
1860, 'was': 1774, 'you': 1497, 'with': 1358, 'had': 1306, 'as': 1174, 'at':
1050, 'her': 1045, 'for': 987, 'him': 976}

Dictionary limited by K is : {'the': 4507, 'to': 4243, 'of': 3731, 'and': 3660,
'her': 2225, 'i': 2070, 'a': 2007, 'in': 1937, 'was': 1847, 'she': 1710,
'that': 1594, 'not': 1566, 'it': 1550, 'you': 1428, 'he': 1339, 'his': 1271,
'be': 1260, 'as': 1192, 'had': 1177, 'with': 1100}
```

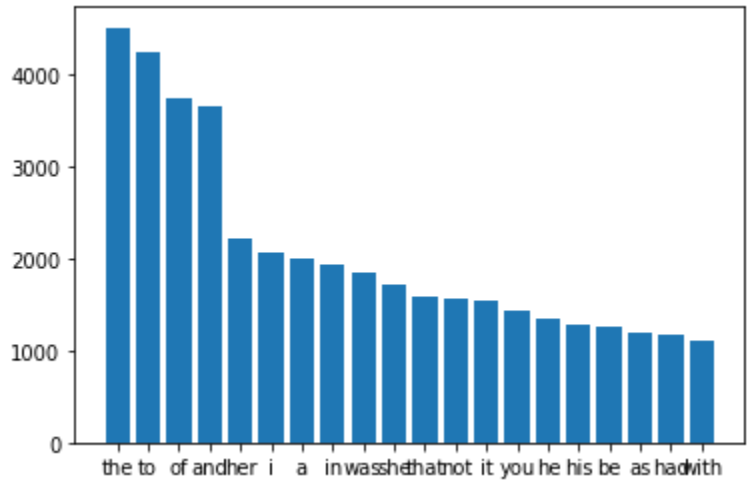
- Output of both dictionaries after limiting to first 20 elements.

```
a_dictionary = out1
keys = a_dictionary.keys()

values = a_dictionary.values()

plt.bar(keys, values)
```



- Bar graph for the frequency distribution of token1 (first 20 elements)
- y-axis: Frequency of tokens
- x-axis: tokens

```
a_dictionary = out2
keys = a_dictionary.keys()
values = a_dictionary.values()

plt.bar(keys, values)
```



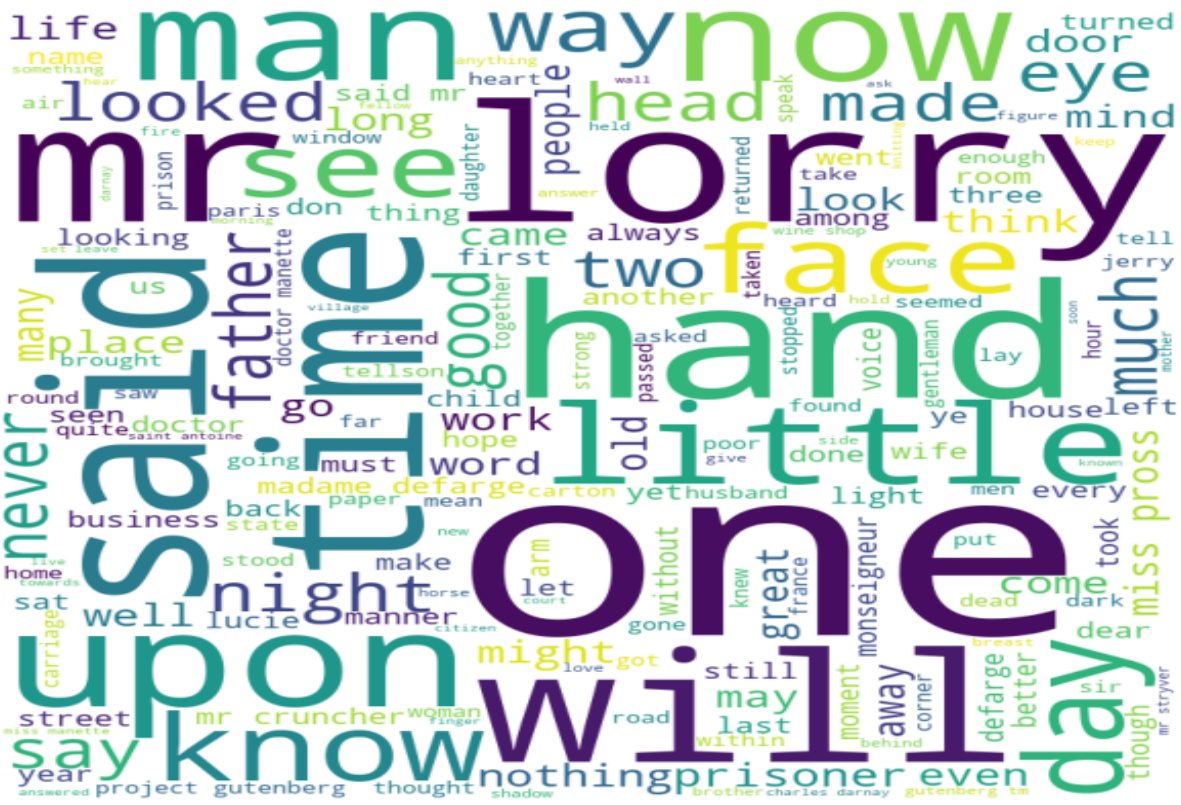
- Bar graph for the frequency distribution of token2 (first 20 elements)
- y-axis: Frequency of token
- x-axis: tokens

➢ Creating a Word Cloud of T1 and T2 using the tokens we got

```
from wordcloud import WordCloud, STOPWORDS
```

```

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(text1)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

- We have imported WordCloud and STOPWORDS library from wordcloud module for constructing wordcloud and removal of stopwords .
- Then we have used plt (matplotlib) to display our wordcloud for text1.



- Similarly using plt (matplotlib) we have displayed wordcloud for text2.

```
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(text2)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

➢ Remove the stopwords from T1 and T2 and then again create a word cloud

Stop words are often **removed** from the text before training deep learning and machine learning models since **stop words** occur in abundance, hence providing little to no unique information that can be used for classification or clustering.

- We have imported the stopwords library from nltk corpus module .
- We defined a function `stopwords_cleaned` which runs a for loop through each and every word of the sentence and if that word isn't found in the stopwords it is appended and so we get the desired (stopword free) text in res.

```
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords = set(nltk.corpus.stopwords.words())

def stopwords_cleaned(sentence):
    res = []
    for word in sentence:
```

```
        if word not in stopwords:
            res.append(word)
    return res
```

```
token3=stopwords_cleaned(token1)
token4=stopwords_cleaned(token2)
```

- We imported FreqDist from the nltk probability module.
- The FreqDist class is used to encode “frequency distributions”, which count the number of times that each outcome of an experiment occurs.
- Then we sorted the distributions in descending order.

```
from nltk.probability import FreqDist
fdist3 = FreqDist(token3)
fdist4 = FreqDist(token4)

sorted_d1 = dict( sorted(fdist3.items(),
key=operator.itemgetter(1),reverse=True))
sorted_d2 = dict( sorted(fdist4.items(),
key=operator.itemgetter(1),reverse=True))
print('Dictionary in descending order by value : ',sorted_d1)
print('Dictionary in descending order by value : ',sorted_d2)
```

Output-

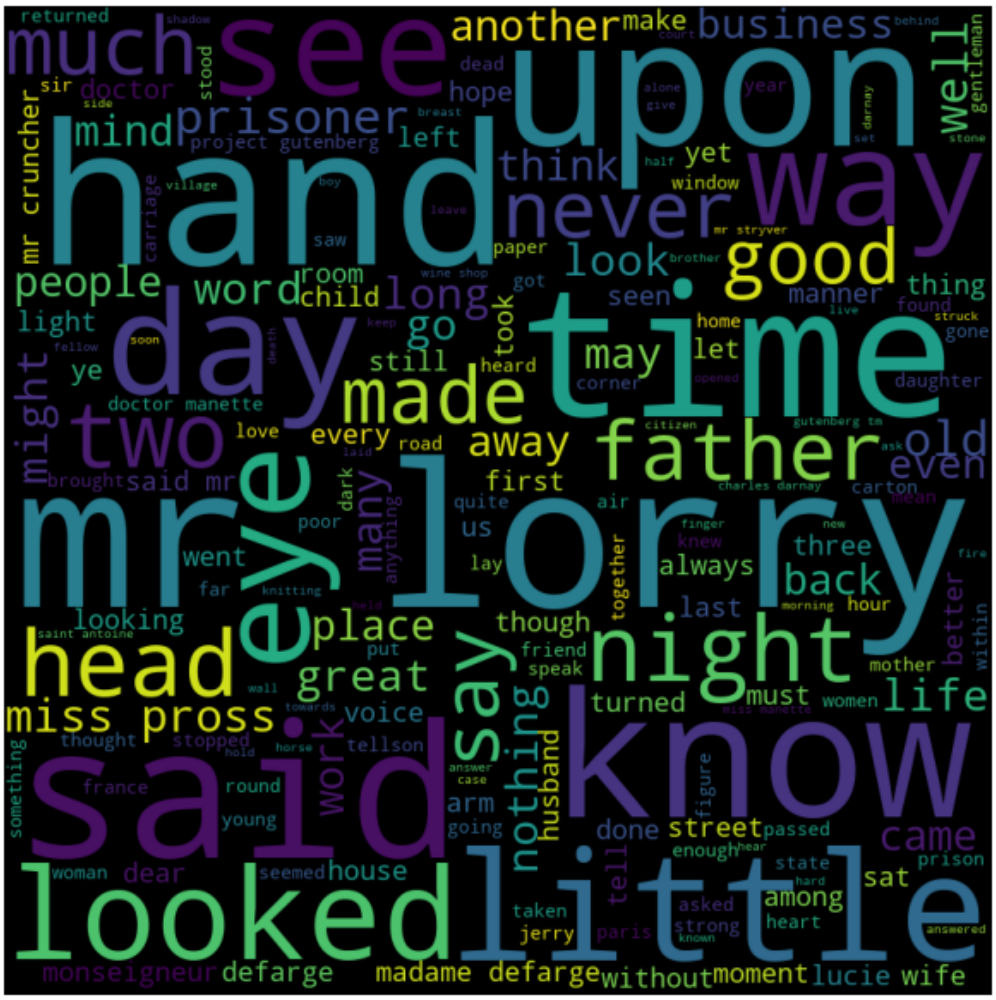
```
Dictionary in descending order by value :  {'said': 661, 'mr': 622, 'lorry':
369, 'would': 343, 'defarge': 302, 'upon': 291, 'could': 283, 'little': 267,
'time': 267,.......}
Dictionary in descending order by value :  {'mr': 786, 'elizabeth': 635,
'could': 527, 'would': 471, 'darcy': 418, 'said': 401, 'mrs': 343, 'much': 329,
'bennet': 323,..........}
```

```
comment_words = ''
comment_words += " ".join(token3)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                min_font_size = 10).generate(comment_words)
```

```
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

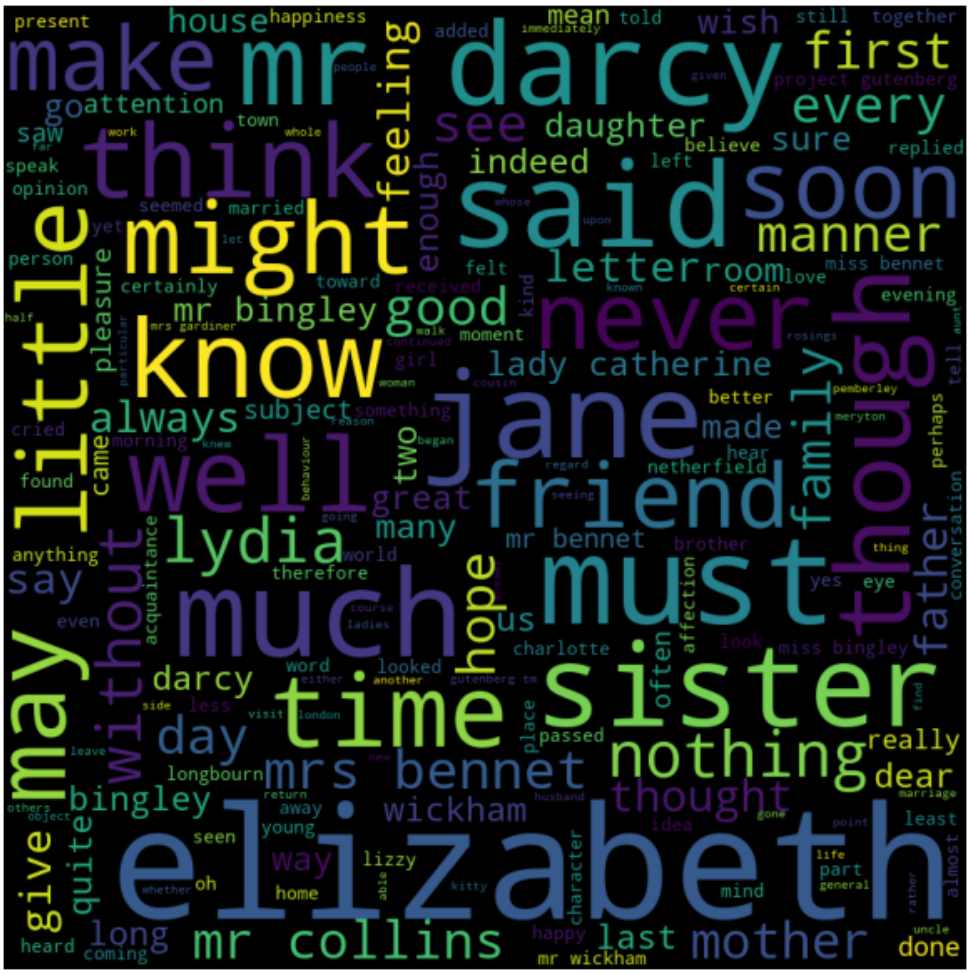
- We join all the words (these words are token 3 which is the text after removal of stopwords) using the .join function and store it to the variable ‘comment_words’ . The variable ‘comment_words’ now contains all the words in a single long text necessary to generate our word cloud.

- Next we use plt (matplotlib) function to display the wordcloud for token3 and then token4 respectively .



- Similarly we make a word cloud after the removal of stop words from T2.

```
comment_words = ''
comment_words += " ".join(token4)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



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

```
words1 = ''
words1 += " ".join(token3)+" "
words2 = ''
words2 += " ".join(token4)+" "
```

- We join all the words (these words are token 3 which is the text after removal of stopwords) using the .join function and store it to the variable ‘words1’ and ‘words2 ‘. These variables now contain all the words in a single long text .

```
from collections import defaultdict
x = {}
def get_word_counts(text):
    for word in text.split():
        if(len(word) not in x):
          x[len(word)]=1
        else:
          x[len(word)]+=1
```

- To evaluate the relation between word length and frequency we have created the function get_word_counts in which we calculate the frequency that how many words of a particular length are occurring . We send our stopwords free text words1 and words2 here in this function and after that we store in a dictionary (X) the key value pair where

  KEY : Word length
  VALUE : Frequency of words of that length

- We display x for both of them after passing words1 and words2 to get_word_count function .

```
(get_word_counts(words1))
print(x)
```

Output-

```
{1: 6082,
 2: 26669,
 3: 38092,
 4: 39329,
 5: 27328,
 6: 22140,
```

```
 7: 17873,
 8: 11766,
 9: 7649,
 10: 4316,
 11: 2192,
 12: 1138,
 13: 544,
 14: 156,
 15: 48,
 16: 10,
 17: 2}
```

```
(get_word_counts(words2))
print(x)
```

Output-

```
{1: 6104,
 2: 27852,
 3: 40650,
 4: 49585,
 5: 36717,
 6: 30663,
 7: 25939,
 8: 16870,
 9: 12829,
 10: 6744,
 11: 3591,
 12: 1953,
 13: 937,
 14: 263,
 15: 80,
 16: 15,
 17: 5}
```

- For better understanding we have created a table for the result recorded :

Word Length	Frequency of particular word length in T1(after removal of stopwords)	Frequency of particular word length in T2(after removal of stopwords)
1	6082	6104

2	26669	27852
3	38092	40650
4	39329	49585
5	27328	36717
6	22140	30663
7	17873	25939
8	11766	16870
9	7649	12829
10	4316	6744
11	2192	3591
12	1138	1953
13	544	937
14	156	263
15	48	80
16	10	15
17	2	5

OBSERVATIONS :

- We observe from the above table that as the word length increases first the frequency increases upto 4 length words or so after that we observe a sharp decline in frequencies in both the cases (T1 & T2) .
- The rate of decline is higher than the rate of increase that was observed for the initial (smaller) word lengths .
- Therefore in our text samples we can conclude that the maximum number of words are of 3-4 length .

➢ POS Tagging for T1 and T2

POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag, based on its context and definition.

```
nltk.download('averaged_perceptron_tagger')
```

- averaged_perceptron_tagger is used for tagging words with their parts of speech (POS).

```
tagged1 = nltk.pos_tag(token3)
print(tagged1)
```

- A part-of-speech tagger, or **POS-tagger**, processes a sequence of words, and attaches a part of speech tag to each word .
- We have used the predefined function `nltk.pos_tag` for finding tags.

Output-

```
[('project', 'NN'), ('gutenberg', 'VBZ'), ('ebook', 'VB'), ('two', 'CD'),
('cities', 'NNS'), ('charles', 'NNS'), ('dickens', 'NNS'), ('ebook',
'VBP'), ('use', 'IN'), ('anyone', 'NN'), ('anywhere', 'RB'), ('cost',
'VBZ'), ('almost', 'RB'), ('restrictions', 'NNS'), ('whatsoever', 'VBP'),
('may', 'MD'), ('copy', 'VB'), ('give', 'VB'), ('away', 'RP'), ('use',
'NN'), ('terms', 'NNS'), ('project', 'VBP'), ('gutenberg', 'JJ'),
('license', 'NN'), ('included', 'VBD'), ('ebook', 'JJ'), ('online', 'NN'),
('www', 'NN'), ('gutenberg', 'NN'), ('org', 'JJ'), ('title', 'NN'), ('two',
'CD'), ('cities', 'NNS'), ('story', 'NN'), ('french', 'JJ'), ('revolution',
'NN'), ('author', 'NN'), ('charles', 'VBZ'), ('dickens', 'VBZ'),
('release', 'NN'), ('date', 'NN'), ('january', 'JJ'), ('posting', 'NN'),
('date', 'NN'), ('november', 'IN'), ('last', 'JJ'),....]
```

- Similarly we do pos tagging for T2.

```
tagged2 = nltk.pos_tag(token4)
print(tagged2)
```

- We have used the predefined function `nltk.pos_tag` for finding tags.

Output-

```
[('project', 'NN'), ('gutenberg', 'NN'), ('ebook', 'NN'), ('pride', 'NN'),
('prejudice', 'NN'), ('jane', 'NN'), ('austen', 'VBP'), ('ebook', 'NN'),
('use', 'NN'), ('anyone', 'NN'), ('anywhere', 'RB'), ('cost', 'VBZ'),
('almost', 'RB'), ('restrictions', 'NNS'), ('whatsoever', 'VBP'), ('may',
'MD'), ('copy', 'VB'), ('give', 'VB'), ('away', 'RP'), ('use', 'NN'),
('terms', 'NNS'), ('project', 'VBP'), ('gutenberg', 'JJ'), ('license',
'NN'), ('included', 'VBD'), ('ebook', 'JJ'), ('online', 'NN'), ('www',
'NN'), ('gutenberg', 'NN'), ('org', 'NN'), ('title', 'NN'), ('pride',
'NN'), ('prejudice', 'NN'), ('author', 'NN'), ('jane', 'NN'), ('austen',
```

```
'JJ'), ('release', 'NN'), ('date', 'NN'), ('august', 'NN'), ('last', 'JJ'),
('updated', 'JJ'), ('november', 'JJ'), ('language', 'NN'), ('english',
'JJ'), ('character', 'NN'), ('set', 'VBN'), ('encoding', 'VBG'), ('utf',
'JJ'), ('start', 'NN'), ('project', 'NN'), ('gutenberg', 'NN'), ('ebook',
'NN'), ('pride', 'NN'), ('prejudice', 'NN'), ('produced', 'VBD'),.....]
```

➢ Get the distribution of various tags for both T1 and T2

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

```
dit2 = {}
for a,b in tagged2:
  if(b not in dit2):
    dit2[b]=1
  else:
    dit2[b]+=1
```

- We created 2 dictionary dit and dit2 for tagged1 and tagged2 respectively.
- Here dictionary is used to store the frequency of various tags.

```
sorted_d1 = dict( sorted(dit.items(), key=operator.itemgetter(1),reverse=True))
sorted_d2 = dict( sorted(dit2.items(),
key=operator.itemgetter(1),reverse=True))
print('Dictionary in descending order by value : ',sorted_d1)
print('Dictionary in descending order by value : ',sorted_d2)
```

- We sorted both the get_word_counts in descending order by values so as to make our illustrations better and easier to understand .

```
Dictionary in descending order by value :  {'NN': 21091, 'JJ': 12077,
```

```
'VBD': 6003, 'NNS': 5326, 'RB': 4879, 'VBG': 2943, 'VBN': 2467, 'VBP':
2161, 'VB': 1734, 'IN': 1516, 'MD': 1140, 'CD': 573, 'VBZ': 558, 'DT': 284,
'JJR': 281, 'JJS': 211, 'RBR': 186, 'PRP': 123, 'RP': 94, 'FW': 65, 'WP$':
46, 'NNP': 34, 'CC': 34, 'UH': 26, 'WDT': 16, 'RBS': 11, 'WRB': 7, 'PRP$':
2, 'PDT': 2, 'POS': 1, 'WP': 1, 'NNPS': 1}
Dictionary in descending order by value :  {'NN': 17765, 'JJ': 10189, 'RB':
5249, 'VBD': 4289, 'NNS': 3722, 'VB': 2331, 'VBG': 2206, 'VBN': 2134,
'VBP': 2045, 'MD': 1937, 'IN': 1081, 'VBZ': 536, 'CD': 333, 'JJS': 311,
'DT': 292, 'JJR': 245, 'RBR': 240, 'PRP': 139, 'FW': 86, 'CC': 80, 'RP':
61, 'WP$': 59, 'WDT': 39, 'UH': 28, 'WRB': 25, 'NNP': 18, 'PRP$': 7, 'WP':
6, 'RBS': 4, 'PDT': 4, 'POS': 2}
```

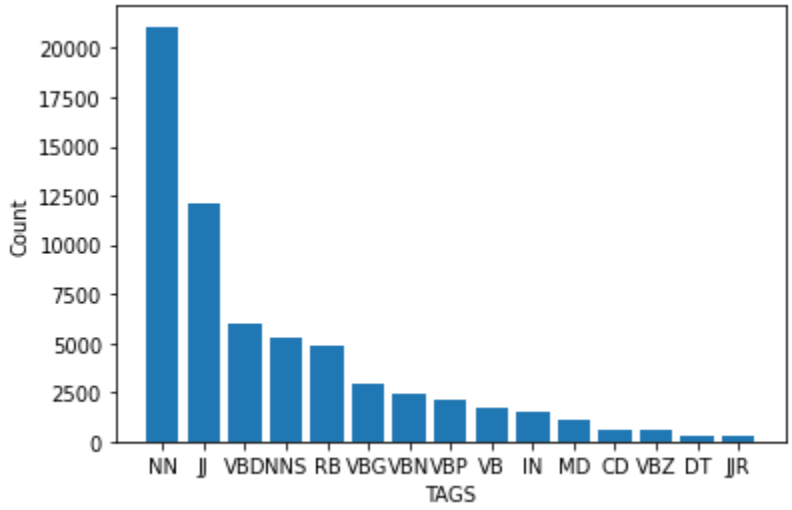
```
K =15
# Using items() + list slicing
# Get first K items in dictionary
out1 = dict(list(sorted_d1.items())[0: K])
out2 = dict(list(sorted_d2.items())[0: K])
# printing result
print("Dictionary limited by K is : " + str(out1))
print("Dictionary limited by K is : " + str(out2))
```

- Next we printed a part of these (K=15) by using slicing operators [ ] and then we made the frequency distribution for both out1 and out 2 (variables which have sorted and sliced dictionary . We have used the inbuilt function of matplotlib library to create the following distribution charts .

```
Dictionary limited by K is : {'NN': 21091, 'JJ': 12077, 'VBD': 6003, 'NNS':
5326, 'RB': 4879, 'VBG': 2943, 'VBN': 2467, 'VBP': 2161, 'VB': 1734, 'IN':
1516, 'MD': 1140, 'CD': 573, 'VBZ': 558, 'DT': 284, 'JJR': 281}

Dictionary limited by K is : {'NN': 17765, 'JJ': 10189, 'RB': 5249, 'VBD':
4289, 'NNS': 3722, 'VB': 2331, 'VBG': 2206, 'VBN': 2134, 'VBP': 2045, 'MD':
1937, 'IN': 1081, 'VBZ': 536, 'CD': 333, 'JJS': 311, 'DT': 292}
```

```
a_dictionary = out1
keys = a_dictionary.keys()
values = a_dictionary.values()
plt.bar(keys, values)
plt.xlabel('TAGS')
plt.ylabel('Count')
plt.show()
```

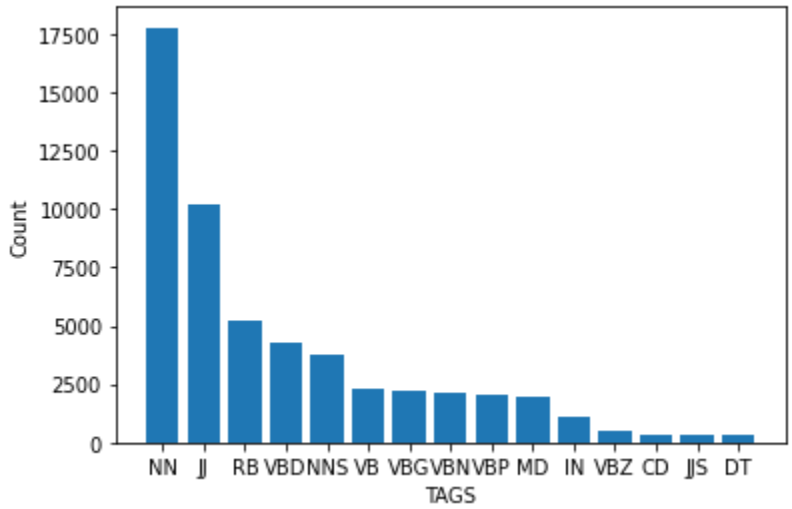
- We have constructed a bar graph for tags vs frequency of tagged1 using matplotlib.pyplot.



- Bar graph for the frequency distribution of tagged1 (first 15 elements)
- y-axis: Frequency of tags
- x-axis: tags

```
a_dictionary = out2
keys = a_dictionary.keys()
values = a_dictionary.values()
plt.bar(keys, values)
plt.xlabel('TAGS')
plt.ylabel('Count')
plt.show()
```

- We have constructed a bar graph for tags vs frequency of tagged2 using matplotlib.pyplot.



- Bar graph for the frequency distribution of tagged2 (first 15 elements)
- y-axis: Frequency of tags
- x-axis: tags

NLP PROJECT ROUND - 2

First Part:

1. ***Find the nouns and verbs in both the novels. Get the categories that these words fall under in the WordNet. Note that there are 25 categories and 16 categories for Nouns and Verbs respectively.***
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, Location, Organisation etc. For this we have to do two steps: (1) First recognise all the entities and then (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 a manual labelling and then compare your result with it. Present the accuracy here.***

Third Part:

1. ***Extract the relationship between the entities (mainly the characters involved in the novel). Try to do this as much as possible.***

LINK FOR CODE REPOSITORY : https://github.com/Rohitag14/NLP-Project-Delta

---

➢Finding the nouns and verbs in both the novels

- Previously , we have done POS tagging for both our novels .
- Now we do conditioning of our result as we want only nouns and verbs for further study
- For **nouns** we have used the following tags-
    - **NN** : used for singular nouns
    - **NNS** : used for plural nouns
    - **NNP** : used for proper nouns
    - **NNPS** : used for plural proper nouns
- For **verbs** we have used the following tags-
    - **VB** : used for verbs
    - **VBG** : used for gerunds
    - **VBD** : used for past tense of verbs
    - **VBN** : used for past participle verbs
    - **VBP** : used for present tense of verbs
    - **VBZ** : used for present tense of verbs with 3rd person singular
- We store these conditioned results in a list named **net1** and we will use this for further analysis.

➢ Categorising Noun and Verb

The WordNet is basically a NLTK corpus reader . It allows us to search data conceptually . It divides the lexicon into 5 categories of Nouns, Verbs, Adjectives , Adverbs and Function Words . We have therefore downloaded the Wordnet and imported it in our code for further use.

- For each of the words in **net1** (verbs and nouns) we use wordnet to categorise them in categories .
- In Wordnet we have 25 and 16 categories for Nouns and Verbs respectively .

- To categorise them in these first we run a loop in **net1** and use **synset.lexname().**
- Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet.
- This will categorise our words (**net1**) something like this(as shown below).
- This result is stored in the form of a string.
- This process is repeated for book2 and we store the results .

```
noun.communication
noun.communication
noun.communication
noun.communication
noun.communication
noun.cognition
noun.act
noun.communication
noun.communication
noun.cognition
verb.creation
verb.cognition
verb.cognition
verb.perception
verb.cognition
verb.social
verb.communication
verb.communication
verb.stative
Verb.social
verb.communication
noun.state
noun.possession
noun.communication
verb.social
verb.contact
verb.social
verb.contact
verb.competition
verb.communication
verb.body
verb.contact
Verb.competition
```

➢ Finding the frequency of each category for all verbs and nouns recorded

- On this string we run a loop which finds **“.”** character and if the substring till **“.”** is noun we store the category in a dictionary named dict1 and if the substring is a verb we store it in a separate dictionary dict2 .
- Now in these dictionaries we have the frequency of each category of noun in dict 1 and for each category of verb in dict 2.
- Similar process was repeated for book2.

- The results recorded for book1 are shown below.

Nouns-

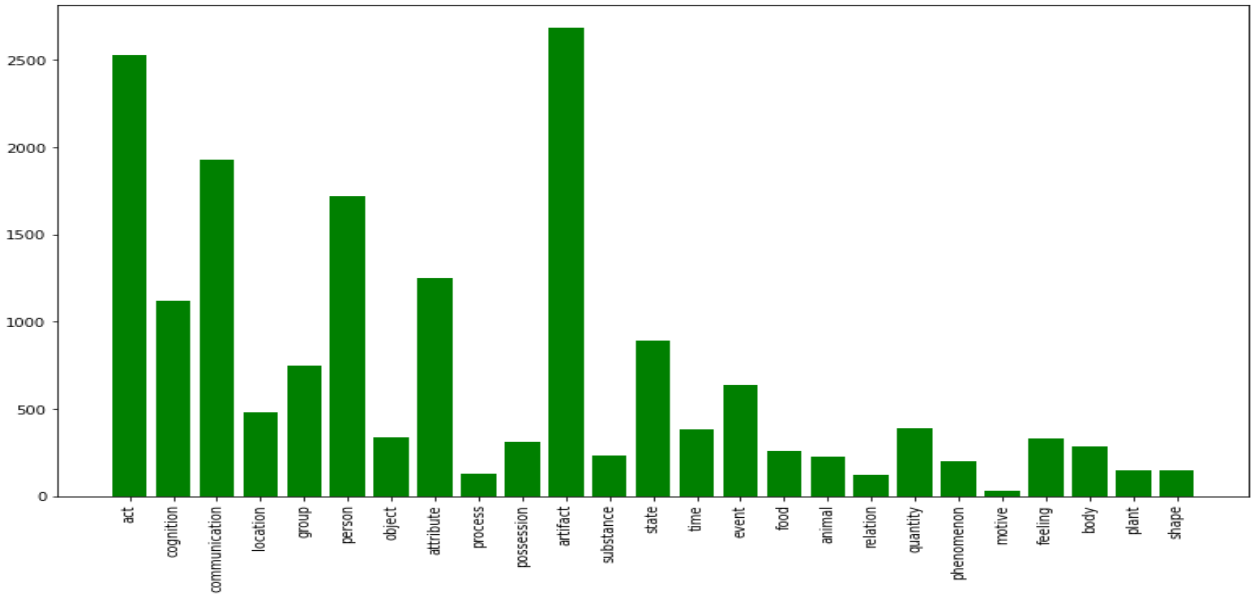
```
{'act': 2532,
 'animal': 227,
 'artifact': 2684,
 'attribute': 1251,
 'body': 285,
 'cognition': 1123,
 'communication': 1928,
 'event': 635,
 'feeling': 333,
 'food': 260,
 'group': 751,
 'location': 478,
 'motive': 29,
 'object': 337,
 'person': 1719,
 'phenomenon': 201,
 'plant': 149,
 'possession': 313,
 'process': 130,
 'quantity': 388,
 'relation': 119,
 'shape': 149,
 'state': 891,
 'substance': 234,
 'time': 383}
```

Verbs-

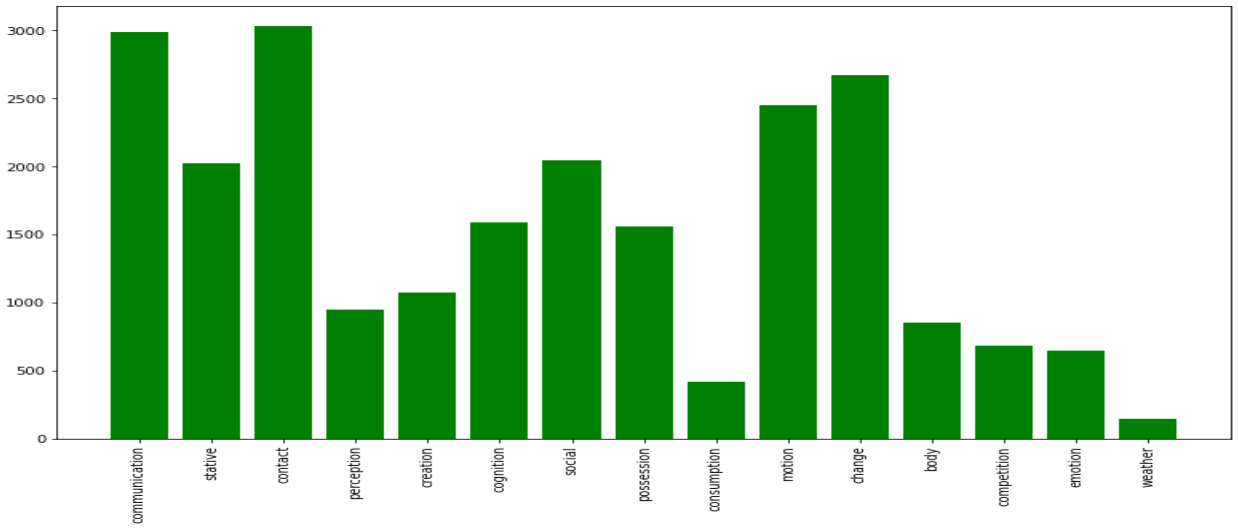
```
{'body': 854,
 'change': 2668,
 'cognition': 1590,
 'communication': 2989,
 'competition': 681,
 'consumption': 419,
 'contact': 3029,
 'creation': 1073,
 'emotion': 646,
 'motion': 2446,
 'perception': 944,
 'possession': 1555,
 'social': 2043,
 'stative': 2025,
 'weather': 145}
```

➢ **Analysing the frequencies of each category for both Verb and Noun**

- **Novel 1**

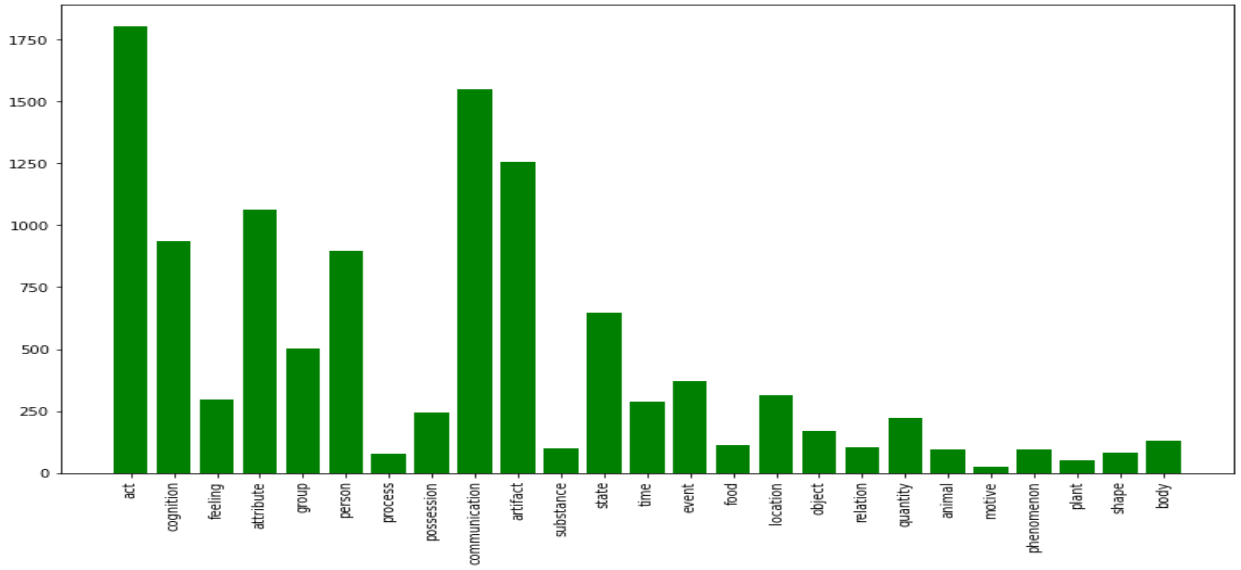


The graph above depicts Frequency for each category for all nouns

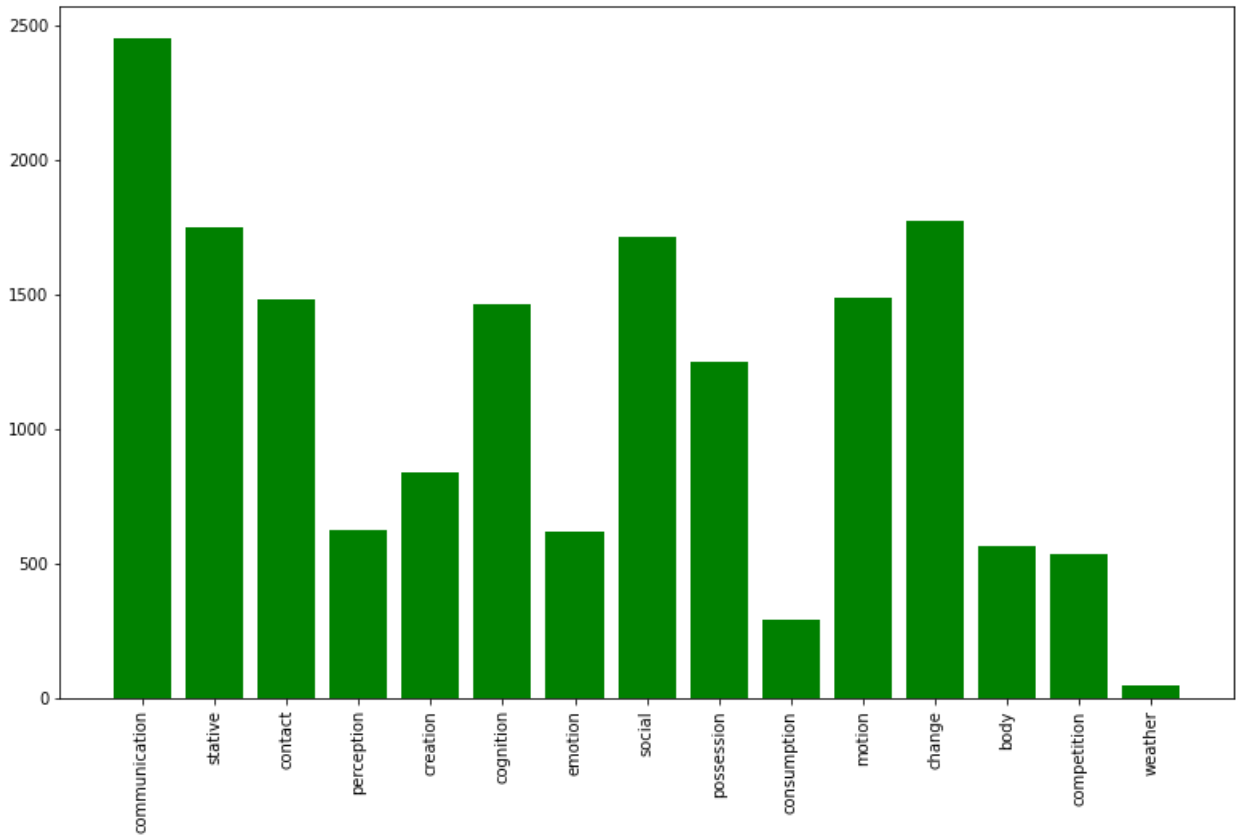


The graph above depicts Frequency for each category for each Verb

- **Novel 2**



The graph above depicts Frequency for each category for all nouns



The graph below depicts Frequency for each category for each Verb

➢Named Entity Recognition (NER)

- It is a subtask of Information Extraction that is used to identify and categorize key information in the text like name, person, location, organization, etc.
- We have imported a new library **spaCy** for this task.
- spaCy supports the following entity types:
    - 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.
- After this we have also imported **en_core_web_sm** and define a function `nlp = en_core_web_sm.load()`
    - This assigns context-specific token vectors, POS tags, dependency parse and named entities.
- Now we have passed book1 to this function and the output is the NER in the form (text, label)
- Some part of the output is shown below-

```
[('two', 'CARDINAL'),
 ('charles dickens', 'PERSON'),
 ('gutenberg', 'GPE'),
 ('www gutenberg', 'FAC'),
 ('two', 'CARDINAL'),
 ('french', 'NORP'),
 ('charles dickens', 'PERSON'),
 ('date january', 'DATE'),
 ('november', 'DATE'),
 ('march', 'DATE'),
 ('english', 'LANGUAGE'),
 ('two', 'CARDINAL'),
 ('judith boss', 'PERSON'),
 ('two', 'CARDINAL'),
 ('the french revolution', 'EVENT'),
 ('charles dickens', 'PERSON'),
 ('first', 'ORDINAL'),
 ('second', 'ORDINAL'),
 ('five years later', 'DATE'),
 ('hundreds', 'CARDINAL'),
 ('two', 'CARDINAL'),
 ('one night', 'TIME'),
```

```
 ('nine days', 'DATE'),
 ('xix', 'PERSON'),
 ('xx', 'PERSON'),
 ('fifty two', 'CARDINAL'),
 ('first', 'ORDINAL'),
 ('the spring', 'DATE'),
 ('england', 'GPE'),
 ('france', 'GPE'),
 ('the year', 'DATE'),
 ('england', 'GPE'),
 ('southcott', 'PERSON')...]
```

- Printing the frequency of each label in the book1

```
Counter({'PERSON': 883, 'CARDINAL': 745, 'DATE': 375, 'TIME': 308, 'GPE':
305, 'ORDINAL': 165, 'NORP': 97, 'ORG': 91, 'FAC': 65, 'LOC': 48,
'PRODUCT': 44, 'LANGUAGE': 41, 'QUANTITY': 23, 'MONEY': 2, 'EVENT': 1})
```

- Pass the book2 in the nlp function defined above we get output showing NER.
- Some part of output is shown below-

```
[('jane', 'PERSON'),
 ('gutenberg', 'GPE'),
 ('www gutenberg', 'FAC'),
 ('jane austen release', 'PERSON'),
 ('november', 'DATE'),
 ('english', 'LANGUAGE'),
 ('david widger', 'PERSON'),
 ('jane austen', 'PERSON'),
 ('first', 'ORDINAL'),
 ('one day', 'DATE'),
 ('netherfield park', 'FAC'),
 ('bennet', 'PERSON'),
 ('bennet', 'PERSON'),
 ('netherfield', 'GPE'),
 ('the north of england', 'LOC'),
 ('monday', 'DATE'),
 ('four', 'CARDINAL'),
 ('the end of next week', 'DATE'),
 ('four or five thousand', 'DATE'),
 ('bennet', 'PERSON'),
 ('one', 'CARDINAL'),
```

```
('bingley', 'PERSON'),
('five', 'CARDINAL'),
('bingley', 'PERSON'),
('one', 'CARDINAL'),
('william', 'PERSON'),
('lucas', 'GPE'),
('bingley', 'PERSON'),
('lizzy', 'PERSON'),....]
```

- Printing the frequency of each label in the book2

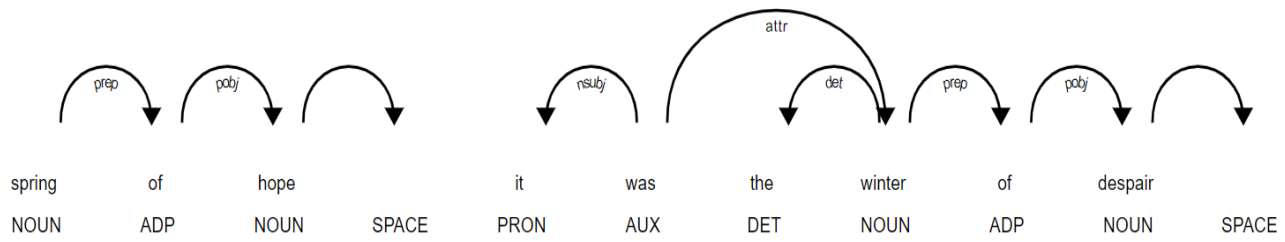
```
Counter({'PERSON': 2368, 'CARDINAL': 396, 'DATE': 347, 'GPE': 320, 'TIME':
217, 'ORDINAL': 161, 'ORG': 114, 'PRODUCT': 57, 'QUANTITY': 42, 'FAC': 28,
'NORP': 17, 'LOC': 8, 'LANGUAGE': 1})
```

➢ Entity depiction

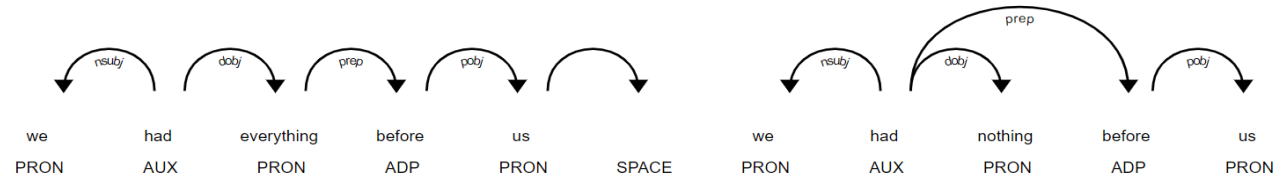
❑ Book1

- Selecting a random line from the book and then showing entity depiction

Part 1

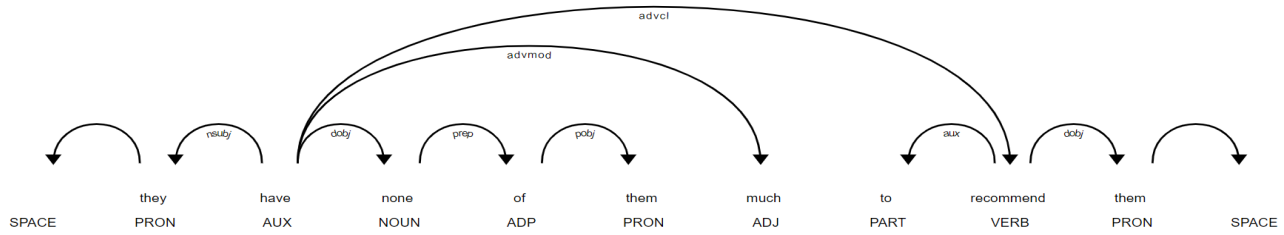


Part 2

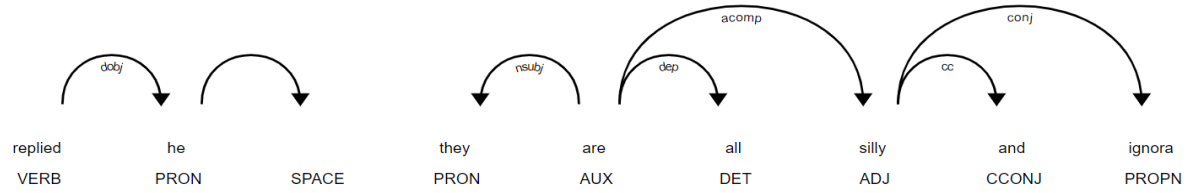


❑ Book2

Part 1



Part 2



➢ Comparing result

❑ Book1

- Picking the random paragraph from the book

```
s="It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were conceded to England at that favoured period, as at
this. Mrs. Southcott had recently attained her five-and-twentieth blessed
birthday, of whom a prophetic private in the Life Guards had heralded the
sublime appearance by announcing that arrangements were made for the swallowing
up of London and Westminster. Even the Cock-lane ghost had been laid only a
round dozen of years, after rapping out its messages, as the spirits of this
very year last past (supernaturally deficient in originality) rapped out
theirs. Mere messages in the earthly order of events had lately come to the
English Crown and People, from a congress of British subjects in America:
which, "
```

- **Output shown by the algorithm**

```
the year DATE
England GPE
Southcott PERSON
five CARDINAL
the Life Guards WORK_OF_ART
London GPE
Westminster GPE
dozen of years DATE
this very year last past DATE
English LANGUAGE
British NORP
America GPE
```

- **Manually labelling output**

```
It was the year DATE of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were conceded to England GPE at that favoured period, as
at this. Mrs. Southcott PERSON had recently attained her five CARDINAL
-and-twentieth blessed birthday, of whom a prophetic private in the Life Guards
WORK_OF_ART had heralded the sublime appearance by announcing that arrangements
were made for the swallowing up of London GPE and Westminster GPE .  Even the
Cock-lane ghost had been laid only a round dozen of years DATE , after rapping
out its messages, as the spirits of this very year last past DATE
(supernaturally deficient in originality) rapped out theirs. Mere messages in
the earthly order of events had lately come to the English NORP Crown and
People, from a congress of British NORP subjects in America GPE : which,
```

- Algorithm shows English as LANGUAGE but it should be NORP because here it refers to the nationality.

- Calculate Accuracy
    - We have created a list which contains the correct output(labelling) and a list of output given by the algorithm.
    - Sort both the list and then compare each value.
    - If any value is present in the correct list but not in the output shown by the algorithm then increase the wrong count.
- So the accuracy will be (11/12) *100 = 91.67 %.

❑ **Book2**

- Picking the random paragraph from the book

```
 s2= [""" Within a short walk of Longbourn lived a family with whom the Bennets
were particularly intimate. Sir William Lucas had been formerly in trade in
Meryton, where he had made a tolerable fortune, and risen to the honour of
knighthood by an address to the king during his mayoralty.The distinction had
perhaps been felt too strongly. It had given him a disgust to his business, and
to his residence in a small market town; and, in quitting them both, he had
removed with his family to a house about a mile from Meryton, denominated from
that period Lucas Lodge, where he could think with pleasure of his own
importance, and, unshackled by business, occupy himself solely in being civil
to all the world. For, though elated by his rank, it did not render him
supercilious; on the contrary, he was all attention to everybody. By nature
inoffensive, friendly, and obliging, his presentation at St. James's had made
him courteous. Lady Lucas was a very good kind of woman, not too clever to be a
valuable neighbour to Mrs. Bennet. They had several children. The eldest of
them, a sensible, intelligent young woman, about twenty-seven, was Elizabeth's
intimate friend. That the Miss Lucases and the Miss Bennets should meet to talk
over a ball was absolutely necessary; and the morning after the assembly
brought the former to Longbourn to hear and to communicate.'YOU began the
evening well, Charlotte,' said Mrs. Bennet with civil self-command to Miss
Lucas. 'YOU were Mr.Bingley's first choice.'
"""]
```

- **Output shown by the algorithm**

```
Longbourn PERSON
William Lucas PERSON
Meryton PERSON
Meryton PERSON
Lucas Lodge PERSON
St. James GPE
Lady Lucas PERSON
Bennet PERSON
about twenty-seven CARDINAL
Elizabeth PERSON
Lucases PERSON
the morning TIME
Longbourn PERSON
Charlotte PERSON
Bennet PERSON
Lucas PERSON
```

```
Bingley PERSON
first ORDINAL
```

- **Manually labelling output**

```
Within a short walk of Longbourn FAC lived a family with whom the Bennets were
particularly intimate. Sir William Lucas PERSON had been formerly in trade in
Meryton PERSON , where he had made a tolerable fortune, and risen to the honour
of knighthood by an address to the king during his mayoralty.The distinction
had perhaps been felt too strongly. It had given him a disgust to his business,
and to his residence in a small market town; and, in quitting them both, he had
removed with his family to a house about a mile from Meryton PERSON ,
denominated from that period Lucas Lodge PERSON , where he could think with
pleasure of his own importance, and, unshackled by business, occupy himself
solely in being civil to all the world. For, though elated by his rank, it did
not render him supercilious; on the contrary, he was all attention to
everybody. By nature inoffensive, friendly, and obliging, his presentation at
St. James GPE 's had made him courteous.
Lady Lucas PERSON was a very good kind of woman, not too clever to be a
valuable neighbour to Mrs. Bennet PERSON . They had several children. The
eldest of them, a sensible, intelligent young woman, about twenty-seven
CARDINAL , was Elizabeth PERSON 's intimate friend. That the Miss Lucases
PERSON and the Miss Bennets should meet to talk over a ball was absolutely
necessary; and the morning TIME after the assembly brought the former to
Longbourn FAC to hear and to communicate. 'YOU PERSON began the evening well,
Charlotte PERSON ,' said Mrs. Bennet PERSON with civil self-command to Miss
Lucas PERSON . 'You were Mr. Bingley PERSON 's first ORDINAL choice.'
```

- Algorithm shows Longbourn as PERSON but it should be FAC because here it refers to the home of the Bennet family.

- Calculate Accuracy
    - We have created a list which contains the correct output(labelling) and a list of output given by the algorithm.
    - Sort both the list and then compare each value.
    - If any value is present in the correct list but not in the output shown by the algorithm then increase the wrong count.
- So the accuracy will be (16/18) *100 = 88.88 %.

➢ Extracting relationship between entities

- The process of extracting relations from the text which occurs between entities is known as relation extraction.
- We have used kaggle instead of google colab for this part because the library which we have imported is very huge and is not accessible by the colab. That's why we have used kaggle.
- We have used open IE (open information extraction) for this part. This refers to the extraction of structured relations from text . It doesn't need the schemas(relations) to be pre-defined.
- We first searched for the NLP Core desired functionality in our directories as well as .zip files . CoreNLP is a Java implementation of an open IE system as described in the stanford paper we have used as reference .
- Following this search if it doesn't succeed we have downloaded it from the source directory of Stanford . We put this in our target directory and set the environment .
- Using the annotator openie ,read the text thoroughly ,locate the target entities , and labelled the same as subject, relation, object in triples .
- We have picked starting 10,000 characters because the processing of the whole text of the book is not supported by the kaggle.

The following two images show the result as displayed by triples list variable for book1 and book2 respectively .

❑ Book1

```
Starting server with command: java -Xmx8G -cp /root/.stanfordnlp_resources/stanford-corenlp-4.1.0/* edu.stanford.nlp.pipeline.StanfordC
oreNLPServer -port 9000 -timeout 60000 -threads 5 -maxCharLength 100000 -quiet True -serverProperties corenlp_server-0a35d2471cbe4c87.p
rops -preload openie
|- {'subject': 'road', 'relation': 'is with', 'object': 'mutinous intent of taking back to blackheath reins'}
|- {'subject': 'you', 'relation': 'give away', 'object': 'it'}
|- {'subject': 'hundreds', 'relation': 'is in', 'object': 'town viii monseigneur'}
|- {'subject': 'anyone', 'relation': 'is with', 'object': 'almost no restrictions whatsoever'}
|- {'subject': 'farmer', 'relation': 's', 'object': 'boy of sixpence'}
|- {'subject': 'shooter', 'relation': 's', 'object': 'hill'}
|- {'subject': 'guard', 'relation': 'is in', 'object': 'combination'}
|- {'subject': 'stand', 'relation': 'is with', 'object': 'wary'}
|- {'subject': 'city tradesman', 'relation': 'is in', 'object': 'light'}
|- {'subject': 'gorgon', 'relation': 's', 'object': 'head'}
|- {'subject': 'king', 'relation': 'is with', 'object': 'fair face on throne of france in countries'}
|- {'subject': 'common way', 'relation': 'is in', 'object': 'midst of them'}
|- {'subject': 'british subjects', 'relation': 'is in', 'object': 'america'}
|- {'subject': 'everything', 'relation': 'was', 'object': 'steaming'}
|- {'subject': 'fair face', 'relation': 'is in', 'object': 'countries'}
|- {'subject': 'fought battles', 'relation': 'is with', 'object': 'their turnkeys'}
|- {'subject': 'steaming mist', 'relation': 'is in', 'object': 'hollows'}
|- {'subject': 'highwayman', 'relation': 'is in', 'object': 'dark'}
|- {'subject': 'mere messages', 'relation': 'is in', 'object': 'earthly order of events'}
|- {'subject': 'town viii monseigneur', 'relation': 'is in', 'object': 'country'}
|- {'subject': 'you', 'relation': 'give', 'object': 'it'}
```

❑ Book2

```
Starting server with command: java -Xmx8G -cp /root/.stanfordnlp_resources/stanford-corenlp-4.1.0/* edu.stanford.nlp.pipeline.StanfordC
oreNLPServer -port 9000 -timeout 60000 -threads 5 -maxCharLength 100000 -quiet True -serverProperties corenlp_server-afe5354e1a8f4813.p
rops -preload openie
|- {'subject': 'his design', 'relation': 'is in', 'object': 'settling here design nonsense'}
|- {'subject': 'heaven', 'relation': 'for', 'object': 'sake'}
|- {'subject': 'anyone', 'relation': 'is with', 'object': 'almost no restrictions whatsoever'}
|- {'subject': 'fortnight', 'relation': 's', 'object': 'acquaintance'}
|- {'subject': 'single man', 'relation': 'is in', 'object': 'possession of good fortune'}
|- {'subject': 'discretion', 'relation': 'is in', 'object': 'her coughs'}
|- {'subject': 'his', 'relation': 'name', 'object': 'bingley'}
```

➢ References

1. https://github.com/philipperemy/Stanford-OpenIE-Python/blob/master/README.md
2. https://wordnet.princeton.edu/
3. https://nlp.stanford.edu/software/openie.html
4. https://spacy.io/api/doc
5. https://www.geeksforgeeks.org/generating-word-cloud-python/

****

THE END

****