NLP Project Round-1

GitHub Link: ***The full project code and related files are available in the following gitHub link: https://github.com/Durvesh021/TextNex-Harnessing-NLP-for-Book-Analysis***

Introduction

- **Data Description:**

  Data used for the project is the book, “A Concise Introduction to Software Engineering”, by Pankaj Jalote.

- **Data Preprocessing Steps**

  Steps undertaken before processing the data are:-

  1. Removal of images, tables, figures, running sections and chapter name
  2. Removal of extra space

- **Data Preparation**

  After performing preprocessing steps, we tokenize the text data.

  1. Tokenization
  2. Lemmatization
  3. Stemming

- **Problem Statement**

  Do PoS Tagging for the tokens of the “A Concise Introduction to Software Engineering”, by Pankaj Jalote” Book using TreeBank Tagset.

Data Preprocessing Steps

1. Removal of Images, Tables, Figures , Running section and Chapter Name :
   - To remove the images, tables and figures we extracted paragraphs from the book. Thus, only text data was left in the docx.
   - To remove the running sections and chapter names, we wrote a regular expression for strings having some text followed by a single digit and ‘.’ for eg(1. Chapter name). Thereafter we searched the text data to match the regex and did not include those strings in our newly formed doc file. The regular expression for the following task is as mentioned below:

     `'[1-9] [A-Za-z ]+'`
   - Labels for the figures, given in our Data is of the form “Figure 1.1 : Figure name”.To eliminate the Figures Labels, Labels were matched with the regular expression and were not included in the processed Data file.The regular expression for the following task is as mentioned below:

     `'Figure [0-9].[0-9]:[A-Za-z .]+'`
   - Labels for the Tables, given in our Data is of the form “Table 1.1 : Table name”.To eliminate the Table Labels, Labels were matched with the regular expression and were not included in the processed Data file.The regular expression for the following task is as mentioned below:

     `'Table [0-9].[0-9]:[A-Za-z .]+'`

2. Removal of extra space :
   - Extra spaces in our raw data were removed by excluding the paragraph having less than 50 characters:

     if len(text)>50:

     doc.add_paragraph(text)

Python Libraries used here are:

- **docx** - It provides functions to handle document files. Main functions include, creating a new docx file, adding headers and paragraphs to it, etc.

- **re** - It helps with regular expressions. A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

```python
#*****Removing figures, images, tables, headings from book******##
import docx
import re
doc = docx.Document()
def readtxt(filename):
    doc1 = docx.Document(filename)
    for para in doc1.paragraphs:
        newtext = para.text
        text = re.sub(r'[1-9] [A-Za-z ]+', '', newtext)
        text = re.sub(r'Figure [0-9].[0-9]:[A-Za-z .]+', '', text)
        text = re.sub(r'Table [0-9].[0-9]:[A-Za-z .]+', '', text)
        if len(text)>50:
            doc.add_paragraph(text)

readtxt('data.docx')
doc.save('preProcessedData.docx')
```

Figure: Code for Pre-Processing the Data

Data Preparation Steps

1. Tokenization:-

```
#**********************Tokenization*********************##
import nltk
fullTokens = []
def tokenization(filename):
    doc1 = docx.Document(filename)
    for para in doc1.paragraphs:
        nltk_tokens = nltk.word_tokenize(para.text)
        for token in nltk_tokens:
            fullTokens.append(token)

tokenization('preProcessedData.docx')
with open("tokens.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(fullTokens))
```

Figure: Code for Tokenization

Tokenization is performed for each paragraph in our text data file using the nltk library's word_tokenize function, and all tokens are saved in a list called fullTokens.

2. Lemmatization:-

```
##********************Lemmatization*******************##
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
punctuations="?:!.,;)(][\"}{*"
lemmaList = []
for word in fullTokens:
    if word not in punctuations:
        lemmaList.append(wordnet_lemmatizer.lemmatize(word))

with open("tokensAfterlemma.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(lemmaList))
```

Figure: Code for Lemmatization

The tokens from list ‘fullTokens’ are given as input to the WordNetLemmatizer, it returns the tokens after lemmatization, we add that token to the lemmalist.

3. Stemming:-

```python
# ##*********************Stemming************************##
from nltk.stem import PorterStemmer
stemList = []
ps = PorterStemmer()
for w in lemmaList:
    stemList.append(ps.stem(w))

with open("tokensafterStem.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(stemList))

```

Figure: Code for Stemming

The tokens from list ‘lemmaList’ are given as input to the PorterStemmer(), it returns the tokens after stemming, we add that token to the stemList.

Python Library Used:-

nltk - NLTK, or Natural Language Toolkit, is a Python package that you can use for NLP. A lot of the data that you could be analyzing is unstructured data and contains human-readable text. Before you can analyze that data programmatically, you first need to preprocess it.

Functions used are:-

1. *Word_tokenize()* – To tokenize the text.
2. *WordNetLemmatizer()* - To lemmatize the tokens.
3. *PorterStemmer()* - To do stemming on tokens.

Graphs and visual inference

- ***Calculating the frequency of tokens:-***

```python
# ##*************frequency distribution******************##
freq = {}
def freqCount(List):
    for token in List:
        if(token in freq):
            freq[token] +=1
        else:
            freq[token] = 1

freqCount(stemList)
file = open('freq.txt','w',encoding='utf-8')
for key, value in freq.items():
    file.write(key+" :"+str(value)+"\n")
```

Figure: Code for calculating frequency of Tokens

Frequency is calculated by iterating over all tokens in stemList. It is stored in a dictionary named ‘*freq*’.

- ***Bar Plot for Top 50 frequent words:-***

```python
#******************bar plot for top 50 frequent words*******************#
import collections
import itertools

sorted_x = sorted(freq.items(), key=lambda kv: kv[1],reverse=True)
sorted_dict = collections.OrderedDict(sorted_x)

top = dict(itertools.islice(sorted_dict.items(), 50))
import matplotlib.pyplot as plt
sizes = list(top.values())
labels = list(top.keys())
plt.barh(labels,sizes)
plt.yticks(fontsize=7.5)
plt.title("Word vs Frequency")
plt.savefig("barGraph"+".png", bbox_inches='tight')
plt.show()
```

Figure: Code for generating Bar plot of Top 50 frequent words (including stopWords) and its frequency

'freq' dictionary is sorted in the decreasing order of the magnitude of frequency values which is stored in *'sorted_x'.*

Python Library Used:-

collections - They are containers used to store data, commonly known as data structures. Lists, tuples, arrays, dictionaries, etc. Python has a built-in collections module that provides additional data structures for data collections.

itertools - It is a Python module used to iterate over data structures that can be skipped in for loops. This module serves as a fast and memory efficient tool, used alone or in combination to form iterator algebras.

matplotlib.pyplot - It is a comprehensive library for creating static, animated and interactive visualizations in Python. Matplotlib makes simple things easy and hard things possible. Create interactive characters that can zoom, pan and update.
Functions used are:-

1. *collections.OrderedDict() -* to track the order in which items were added.
2. *itertools.islice()* - to handle iterators for top 50 frequent words.
3. *plt.barh(labels,sizes)* - to specify that the bar chart is to be plotted by using the labels column as the Y-axis, and the sizes as the X-axis.
4. *plt.yticks()* - to specify the font size of words on Y-axis.
5. *plt.title()* - to specify the title to the bar chart.
6. *plt.savefig* - to save the graph figure in the storage.
7. *plt.show()* - to show the output of the bar chart.

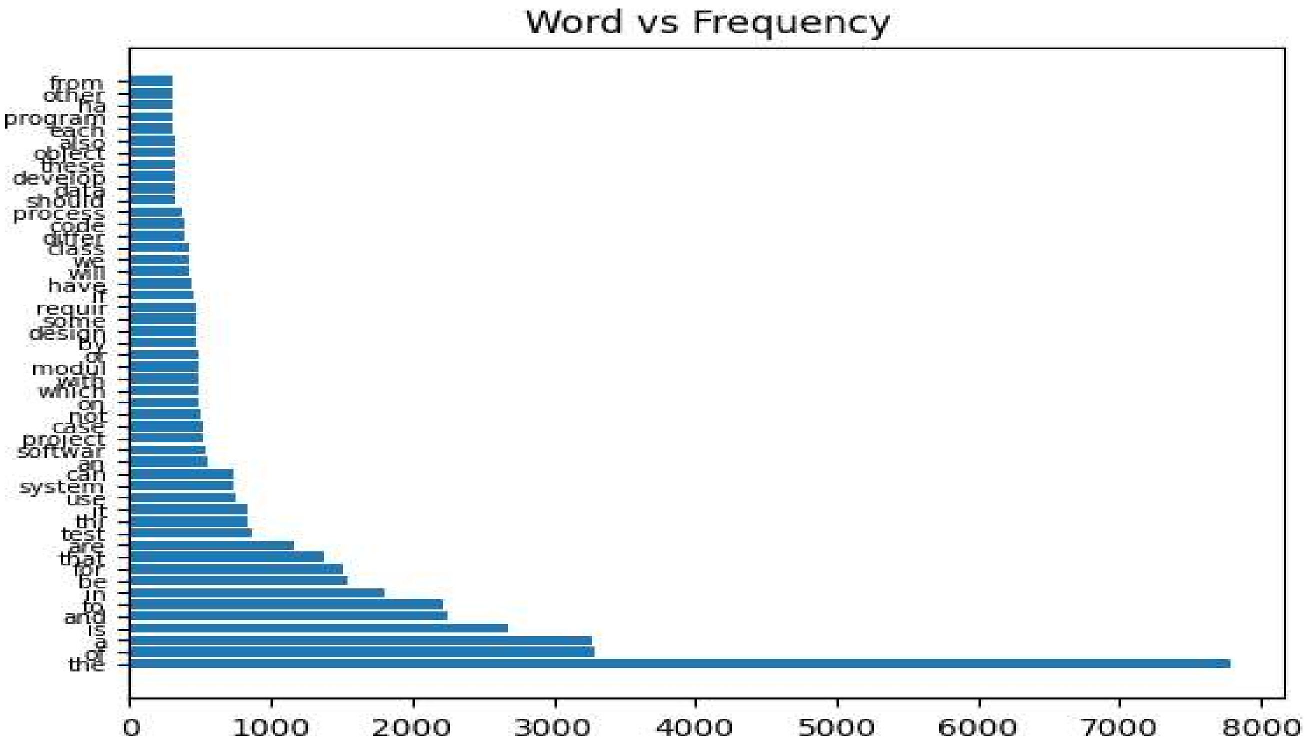


Figure: Bar Plot for Top 50 frequent words including stopWords and its frequency

- ***Word Cloud for all the tokens in our Data file:-***

```
# ##*******************Word Cloud*************************##
from wordcloud import WordCloud
wcloud = WordCloud().generate_from_frequencies(freq)
plt.figure()
plt.imshow(wcloud, interpolation="bilinear")
plt.axis("off")
plt.savefig("WordCloudWithStopWords"+".png", bbox_inches='tight')
plt.show()
```

Figure: Code for Generating Word Cloud of all the Tokens(Including StopWords)

The *‘freq’* dictionary is used to create the word cloud for the set of words **with stop words included**.

Python Library Used:-

WordCloud - WordCloud is a technique that displays the most frequently used words in a given text. We can create word cloud by importing this library and then by selecting the Dataset and selecting the Text and amount of text for Word Cloud by using the function *WordCloud.generate_from_frequencies(freq).*

Function(s) Used:-

1. *WordCloud.generate_from_frequencies(freq) -* take a dictionary of words and their frequencies (here dictionary name is *freq*) and create a word cloud from the counts.
2. *plt.figure()* - used to create a new figure.
3. *plt.imshow()* - matplotlib function imshow() creates an image from a 2-dimensional numpy array.
4. *plt.axis()* - used to set some axis properties to the graph.
5. *plt.savefig()* - to save the graph figure in the storage.
6. *plt.show()* - to show the output of the bar chart.

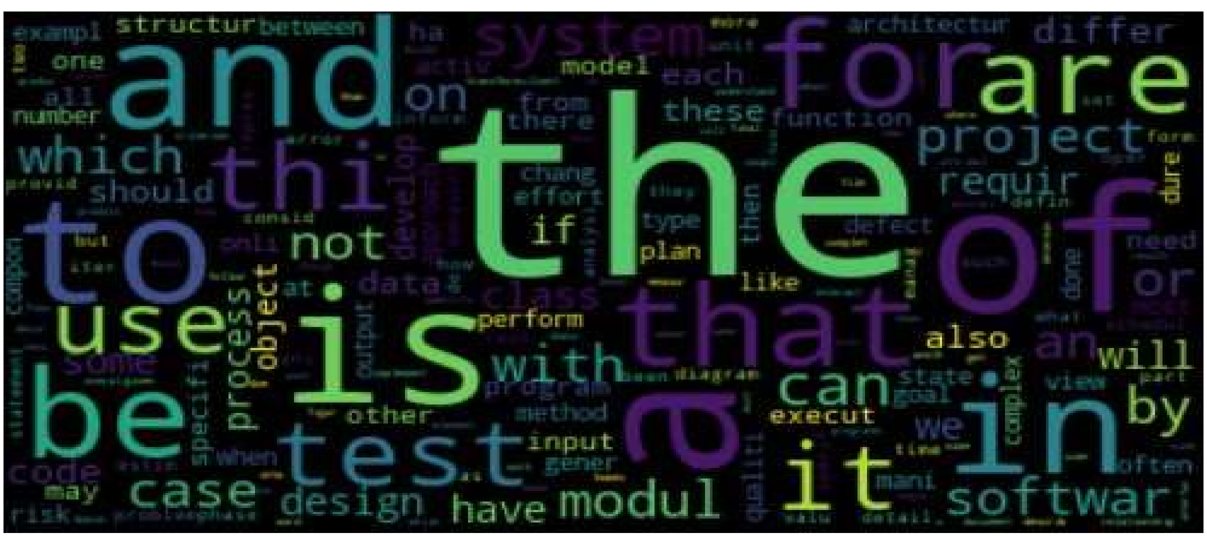


Figure: Word Cloud of all the Tokens(including Stop Words)

- ***Removing Stop Words:-***

```python
# # ##*******************Remove Stop Words******************##
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
filteredList = []
for w in stemList:
    if w not in stop_words:
        filteredList.append(w)
with open("tokensWithoutStopWords.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(filteredList))
```

Figure: Code for removing StopWords

- ***Frequency Distribution After Removing Stop Words:-***

Frequency is calculated by iterating over all tokens in stemList after removing stop words. It is stored in a dictionary named '*freq1'.*

```python
# # ##*************Frequency Distribution After Removing Stopwords**************##
freq1 = {}
def freqCount(list):
    for token in list:
        if(token in freq1):
            freq1[token] +=1
        else:
            freq1[token] = 1
freqCount(filteredList)
```

Figure: Code for frequency distribution after removing stopwords

- ***Bar Plot for Top 50 frequent words after removing stop words:-***

'freq' dictionary is sorted in the decreasing order of the magnitude of frequency values which is stored in *'sorted_y'.*

```python
import collections
import itertools

sorted_y = sorted(freq1.items(), key=lambda kv: kv[1],reverse=True)
sorted_dict1 = collections.OrderedDict(sorted_y)

top1 = dict(itertools.islice(sorted_dict1.items(), 50))
import matplotlib.pyplot as plt
sizes1 = list(top1.values())
labels1 = list(top1.keys())
plt.barh(labels1,sizes1)
plt.yticks(fontsize=7.5)
plt.title("Word vs Frequency")
plt.savefig("barGraph2"+".png", bbox_inches='tight')
plt.show()
```

Figure: Code for Bar Plot for Top 50 frequent words after removing stop words

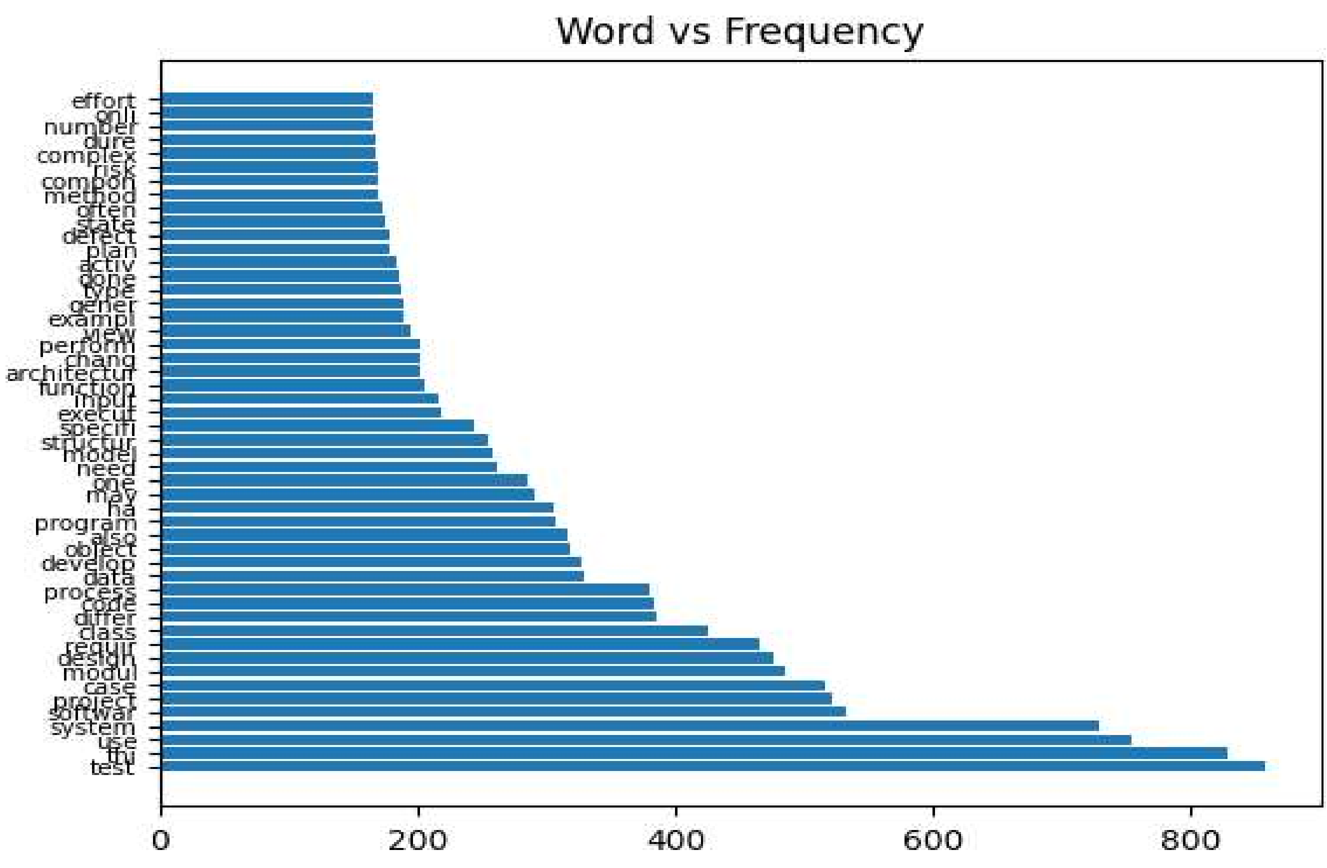


Figure: Bar Plot of Top 50 frequent words(excluding stopWords) and their Frequency

- ***Word Cloud for all the tokens in our Data file after removing stop words:-***

The *'freq'* dictionary is used to create the word cloud for the set of words **after removing stop words**.

```
#*****************Word Cloud after removing stop words*************************#
wcloud = WordCloud().generate_from_frequencies(freq1)
plt.figure()
plt.imshow(wcloud, interpolation="bilinear")
plt.axis("off")
plt.savefig("WordCloud"+".png", bbox_inches='tight')
plt.show()

```

Figure: Code for Word Cloud of all the Tokens excluding StopWords

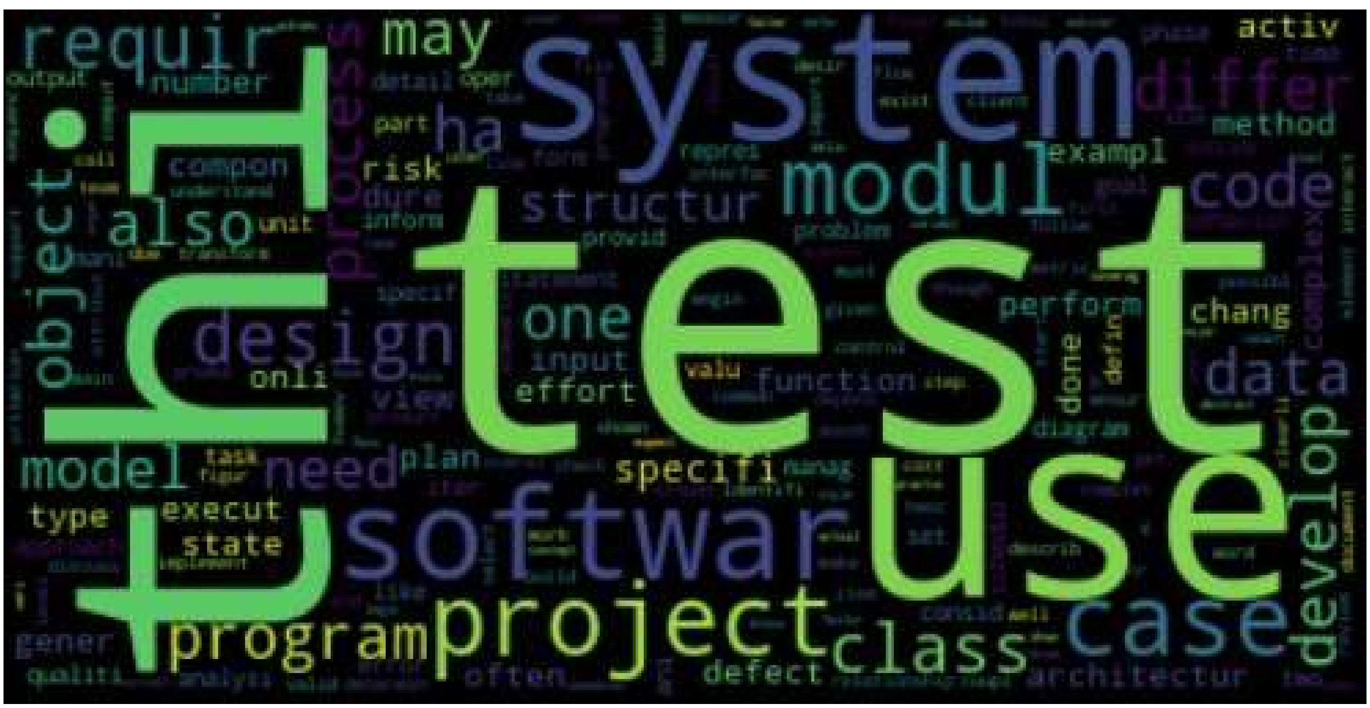


Figure: Word Cloud of all the Tokens(excluding StopWords)

- ***Relationship between word length and frequency and It's Graph Plotting:-***

This function uses stemList and stores length of words with its corresponding frequency of occurrence and stores it in a txt file named freqWithLength.

```
# ##******************RelationShip b/w word Length and frequency*********************##
from collections import Counter
counts = Counter(len(word) for word in stemList)
wordLengths = []
frequencies = []
file = open('freqWithLength.txt','w',encoding='utf-8')
file.write(" Len     Freq")
for length in range(1, max(counts.keys()) + 1):
    file.write("\n")
    wordLengths.append(length)
    frequencies.append(counts.get(length,0))
    file.write(f'{length:4d} {counts.get(length, 0):6d}')

#*******************Graph Plotting****************************#

plt.barh(wordLengths,frequencies)
plt.yticks(fontsize=7.5)
plt.title("Word length v/s Frequency")
plt.savefig("barGraph"+".png", bbox_inches='tight')
plt.show()
```

Figure:Code for generating the bar plot for word length and frequency

Python Library Used:-

Counter - It is a dict subclass for counting hashable objects. A collection that stores items as dictionary keys and their counts as dictionary values.

Function(s) used:-

1. *Counter(len(word) for word in stemList)* - it returns the number of times an object appears in a list.

After this, the program is plotting the bar graph between word length and frequency.

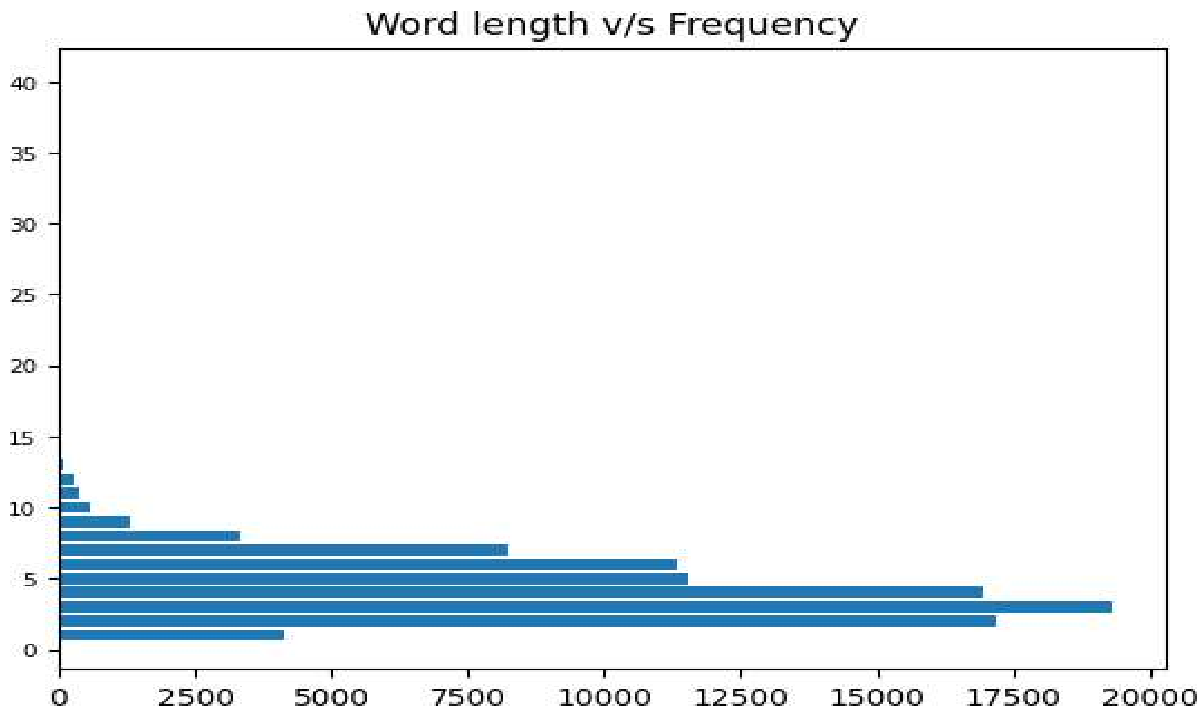


Figure: Bar Plot of Word length and Frequency

POS Tagging

This function uses fiteredList() for the final **POS tagging using TreeBank Tagset**, which is then stored in a txt file named as taggedTokens.

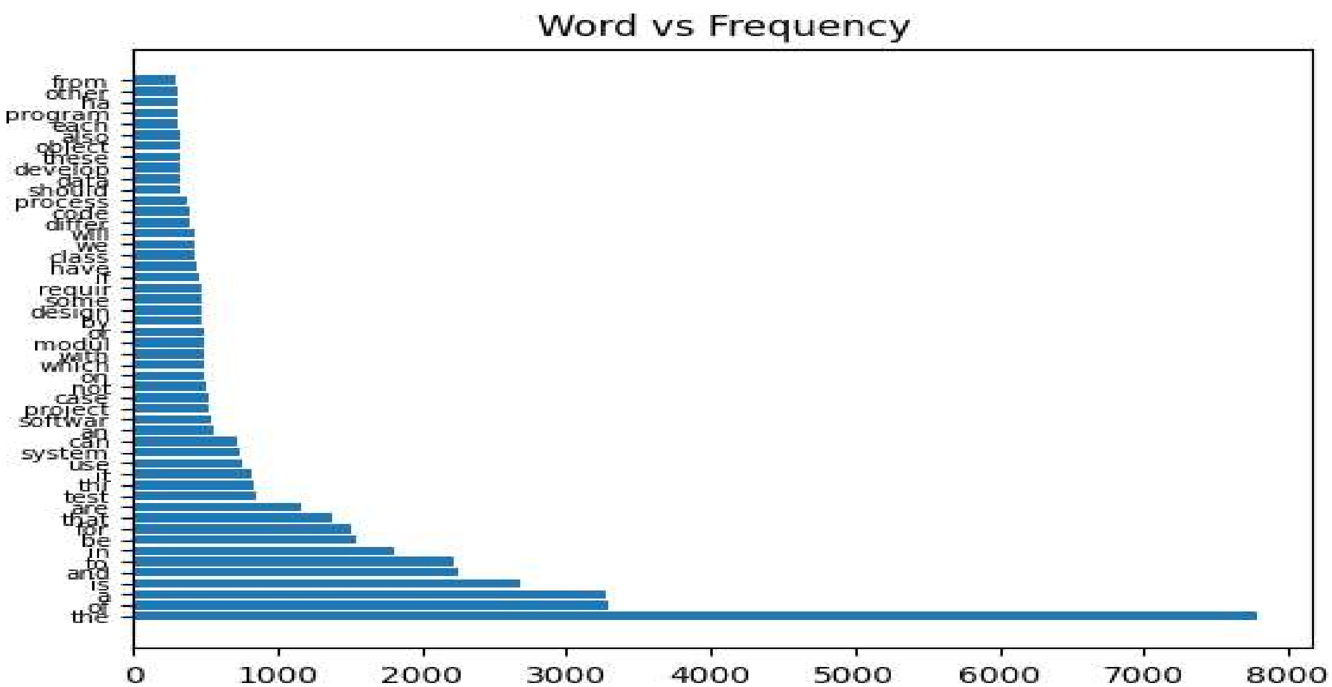
```
# ##********************POS Tagging*************************************##
from nltk import tag
taggedTokens = tag.pos_tag(filteredList)
with open("taggedTokens.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(" ".join(tup) for tup in taggedTokens))
```

Figure: Code for POS Tagging using TreeBank Tagset

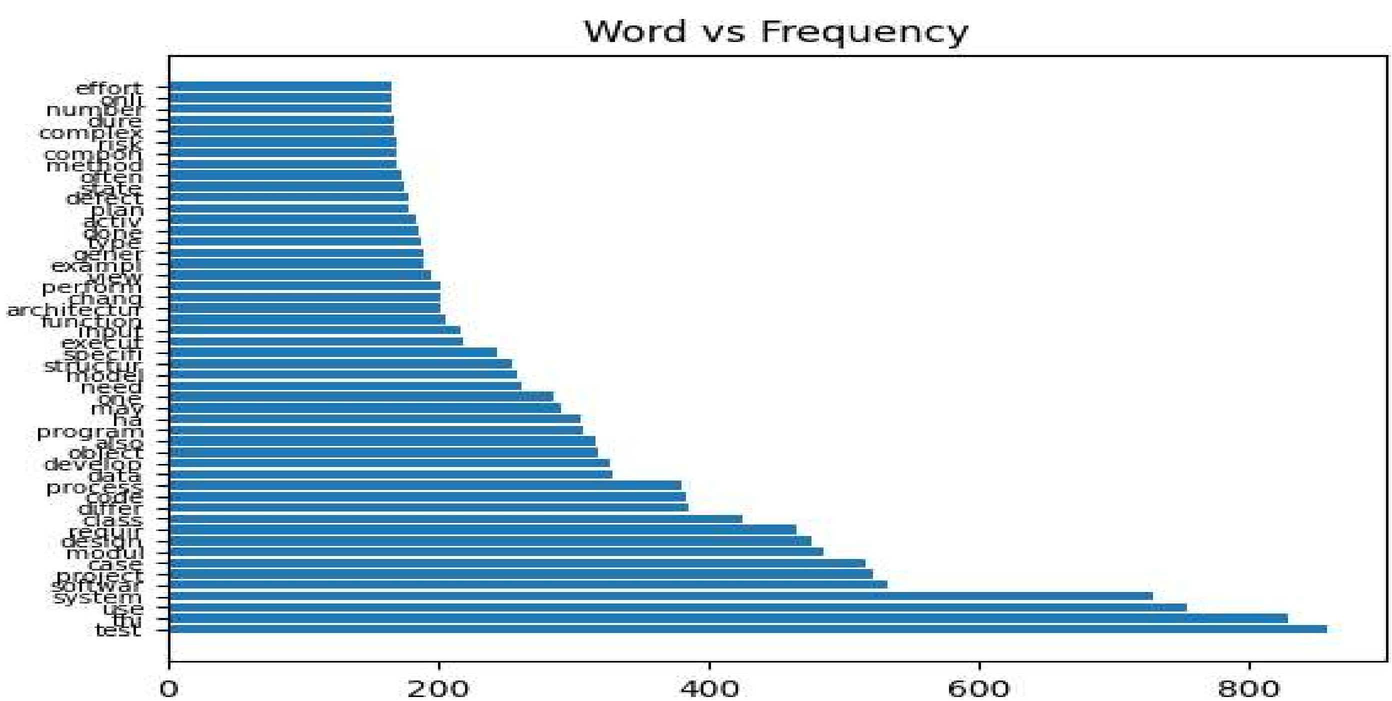
Observation

❖ **Difference between the Bar Plot of Word vs Frequency with stopWords and Without StopWords**

- *With stop words:-*



- *Without stopWords:-*

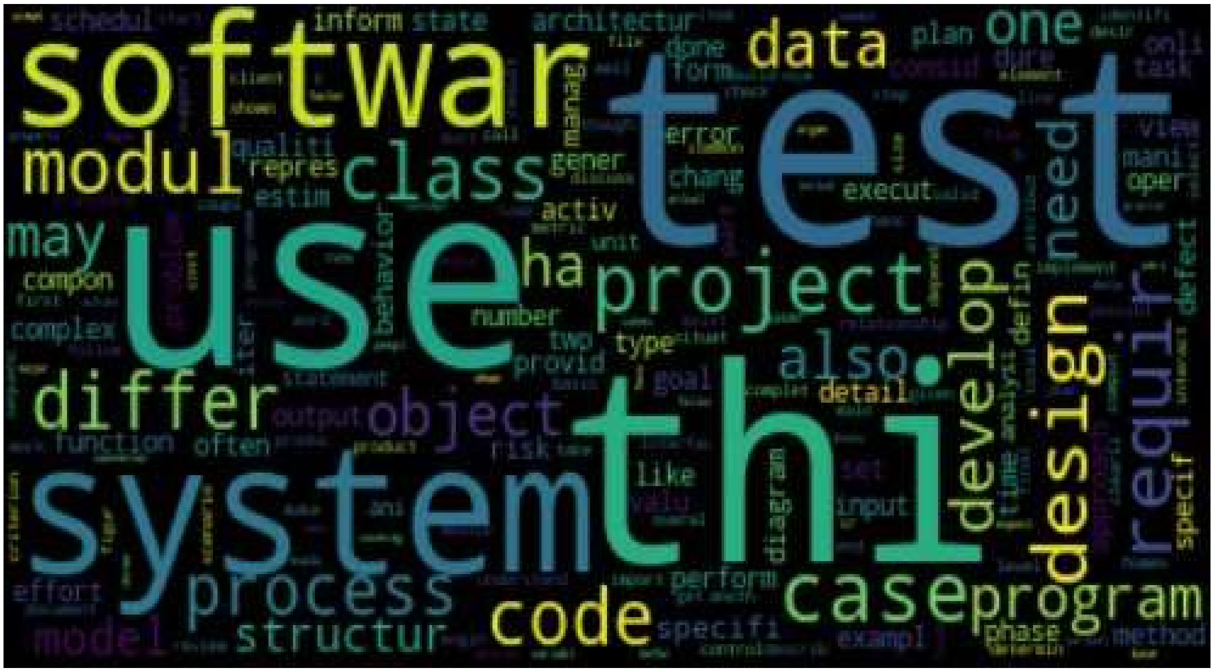


- *The Bar Plot of word vs frequency including stop words contains **'the'** with the highest frequency with count of 7800 approximately.*
- *The words such as **'of'**, **'a'**, **'is'**, **'and'**, **'to'**, **'be'**,. ... are having the frequency around 1000-3500 which consist of a very large set of our corpus.*
- *The Bar plot of word vs frequency excluding stop words contains **'test'** with the highest frequency with the count of 820 approximately.*
- *By this observation we can conclude that most of the set of our data (65%) comprises stopWords.*

❖ ***<u>Difference between Word Cloud with and without stop words</u>***

- ***With stop words:-***

- ***Without stopWords:-***



DIfference:-

*Before removing stopwords words like, **'the', 'is', 'a', 'to'** , etc are having large font sizes in the word cloud. Whereas, after removing stop words, **'test', 'use', 'systems', 'thi'** ,etc,. have large font sizes.*

- ***GitHub Link:***

  The project round 1 code is available in the following gitHub link:
  https://github.com/manavjangid5/The_Markovs_NLP_Project/blob/master/Python_Code/nlp_project_round_1.py

NLP Project Round-2

Introduction

- **Part 1:**
  - Finding the nouns and verbs in the book.
  - Getting the categories of the words that fall under in the WordNet.
  - Getting the frequency of each category for each noun and verb in their corresponding hierarchies
  - Providing Visual Inferences of the frequency of each category for each noun and verb in their corresponding hierarchies.

- **Part 2 :**
  - Recognizing all entities and their types of Data set.
  - Recognizing all entities and their types of Test set.
  - Using performance measures to measure the performance of the method used.

- **Part 3:**
  - Using the ideas given in the book and presented in the class to augment the data of Entities
  - Augmenting that data by extracting additional features and building the table to present it.

Course of Action

Part-1

- **Finding the nouns and verbs in the book:**

  ➔ We have used POS tags to find nouns and verbs. The tokens with tags(NN, NNP, NNS, NNPS) are added to a list containing **nouns**.

```
# -------------------------------------------
nouns = []
for word, pos in taggedTokens1:
    if (pos == 'NN' or pos == 'NNP' or pos == 'NNS' or pos == 'NNPS'):
        if (nouns.count(word) == 0):
            nouns.append(word)

with open("nouns.txt", "w", encoding='utf-8') as outfile:
    outfile.write("\n".join(nouns))
# ---------------------------------------------
```

  ➔ The tokens with POS tags (VBD, VBG, VBN, VBZ) are added to a list containing **verbs**.

```

verbs = []
for word, pos in taggedTokens1:
    if (pos == 'VB' or pos == 'VBD' or pos == 'VBG' or pos == 'VBN' or pos == 'VBZ'):
        if (verbs.count(word) == 0):
            verbs.append(word)

with open("verbs.txt", "w", encoding='utf-8') as outfile:
    outfile.write("\n".join(verbs))
# ---------------------------------------------
```

- **Getting the categories of the words that fall under in the WordNet:**

➔ There are 25 categories of nouns and 16 categories of verbs. We can find them in lexname() attribute of synset of particular token.

➔ <u>Python Module Used :</u>

◆ **Wordnet.synset :** 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. Some of the words have only one Synset and some have several.

```python
nounlex = {}
for noun in nouns:
    syn = wordnet.synsets(noun)
    for s in syn:
        nounlex[noun] = s.lexname()
# ----------------------------------------------------
verblex = {}
for verb in verbs:
    syn = wordnet.synsets(verb)
    for s in syn:
        verblex[verb] = s.lexname()
# --------------------------------------------------------

file = open('freqlex.txt', 'w', encoding='utf-8')
for key, value in nounlex.items():
    file.write(key+" :"+value+"\n")
for key, value in verblex.items():
    file.write(key+" :"+value+"\n")
# ----------------------------------------------------------
```

➔ We are storing the the nouns and verbs tokens and its corresponding categories in **nounlex** and **verblex** dictionary respectively.

➔ Generating a text file named **freqlex** to list all the nouns and verbs token and their corresponding categories.

- **Getting the frequency of each category for each noun and verb in their corresponding hierarchies :**

➔ We are using **nounlex** and **verblex** dictionaries for finding frequencies for noun and verb categories respectively.

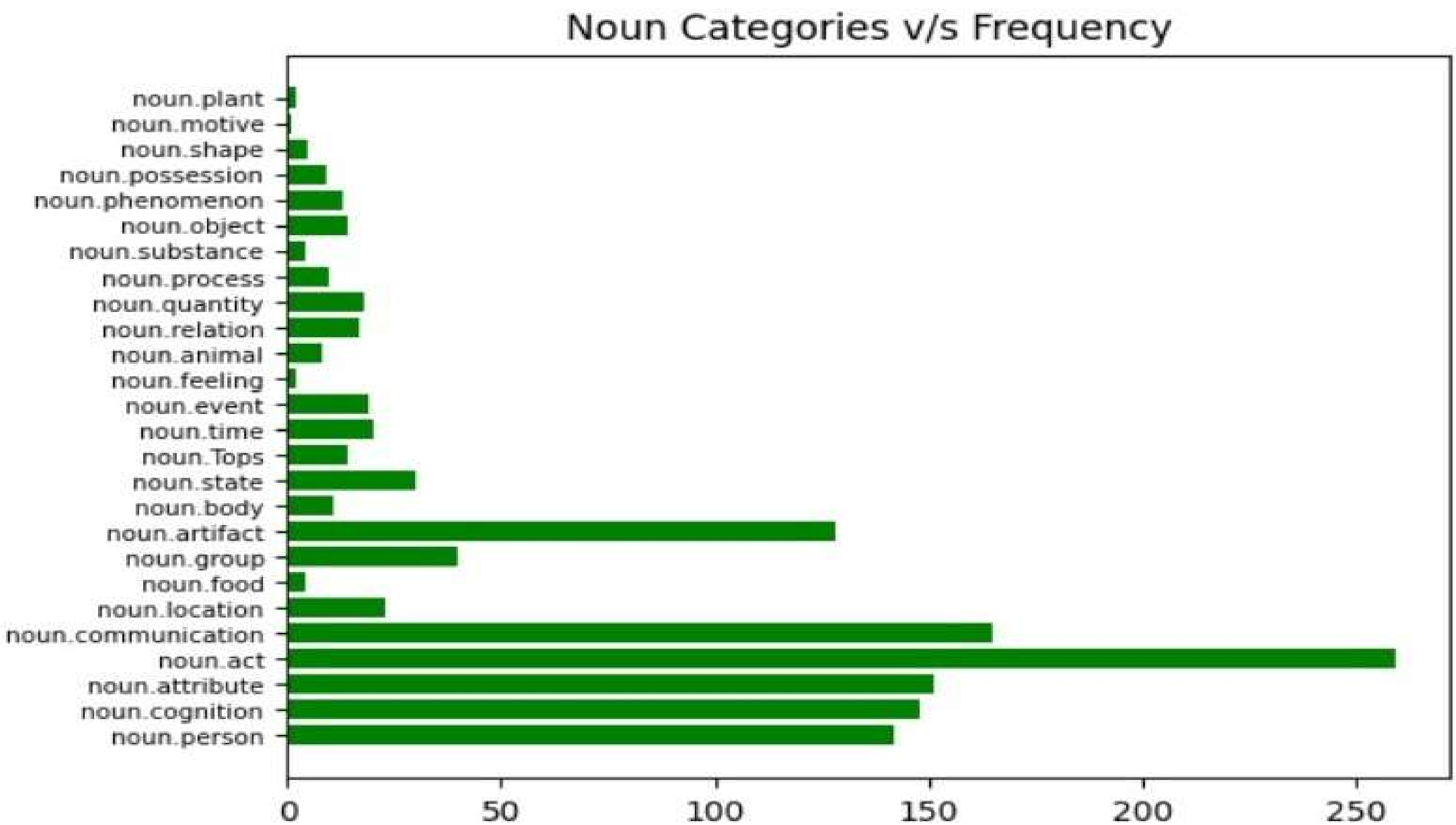
➔ Stored them in the **nounFreq** and **verbFreq** respectively.

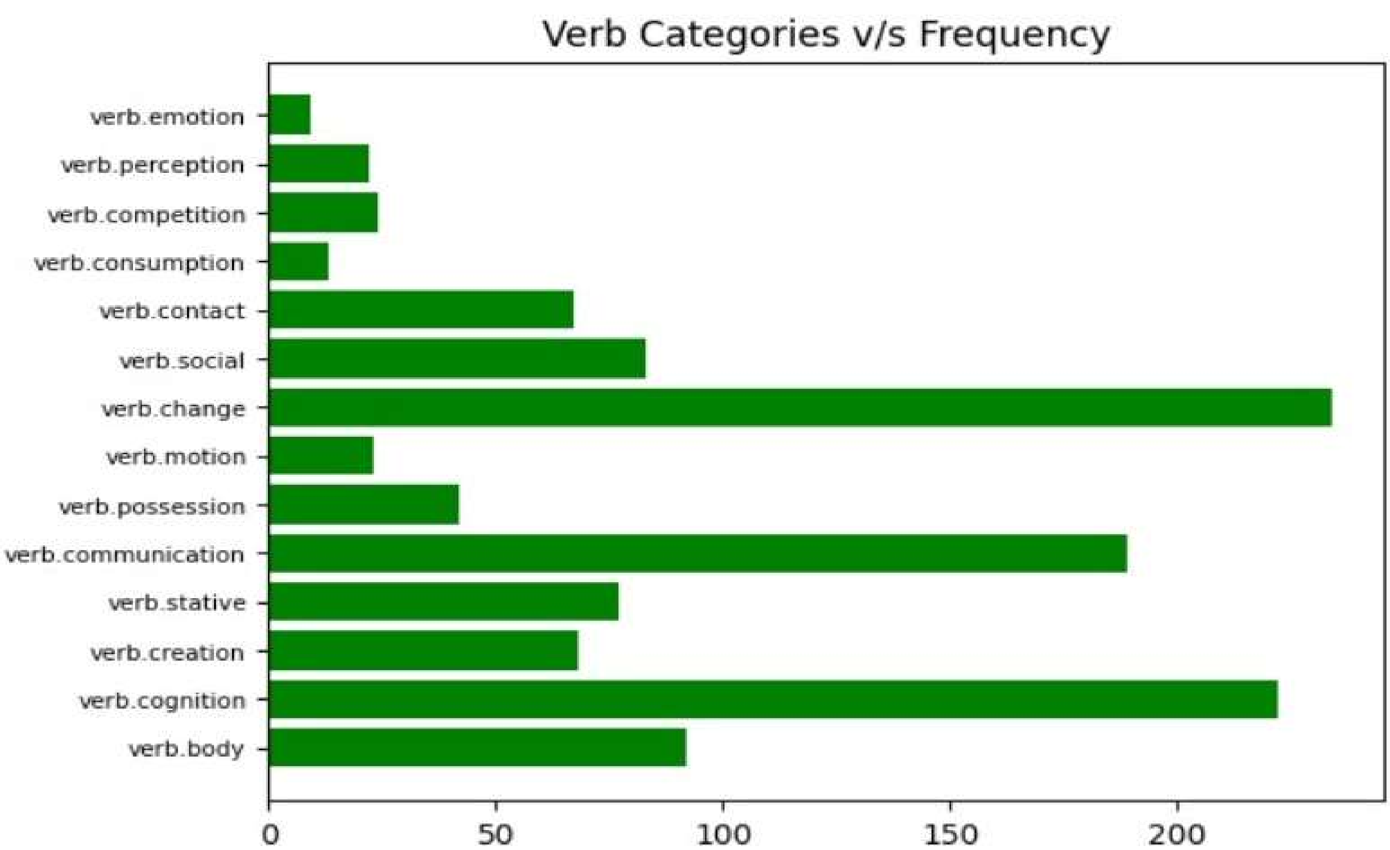
```
nounFreq = {}

for key, value in nounlex.items():
    if (value[0:4] == "noun"):
        if (value in nounFreq):
            nounFreq[value] += 1
        else:
            nounFreq[value] = 1
# ------------------------------------------------------------

verbFreq = {}

for key, value in verblex.items():
    if (value[0:4] == "verb"):
        if (value in verbFreq):
            verbFreq[value] += 1
        else:
            verbFreq[value] = 1

# ------------------------------------------------------------
```

- **Visual Inferences of the frequency of each category for each noun and verb in their corresponding hierarchies:**

➔ Following image shows code for the bar plot of noun Categories v/s frequency and Verb category v/s Frequency.

```
plt.barh(list(nounFreq.keys()), nounFreq.values(), color='g')
plt.yticks(fontsize=7.5)
plt.title("Noun Categories v/s Frequency")
plt.savefig("BarPlot_NounCategories_Frequencies"+".png", bbox_inches='tight')
plt.show()

plt.barh(list(verbFreq.keys()), verbFreq.values(), color='g')
plt.yticks(fontsize=7.5)
plt.title("Verb Categories v/s Frequency")
plt.savefig("BarPlot_verbCategories_Frequencies"+".png", bbox_inches='tight')
plt.show()
```


Noun Categories v/s Frequency
noun.plant
noun.motive
noun.shape
noun.possession
noun.phenomenon
noun.object
noun.substance
noun.process
noun.quantity
noun.relation
noun.animal
noun.feeling
noun.event
noun.time
noun.Tops
noun.state
noun.body
noun.artifact
noun.group
noun.food
noun.location
noun.communication
noun.act
noun.attribute
noun.cognition
noun.person
0
50
100
150
200
250


Verb Categories v/s Frequency
verb.emotion
verb.perception
verb.competition
verb.consumption
verb.contact
verb.social
verb.change
verb.motion
verb.possession
verb.communication
verb.stative
verb.creation
verb.cognition
verb.body
0
50
100
150
200

Part-2

- ## Recognizing all entities and their types of Data set:

  - First of all we will import the **spaCy** library in our code :
    - SpaCy Library includes features such as **Tokenization**, **POS Tagging**, **Dependency Parsing**, **Lemmatization**, **NER**, **Entity linking**, **Training** and many more.

  - Loading all the components of **en_core_web_sm** and storing it in the variable named **nlp.**
    - **En_core_web_sm** is an English pipeline optimized for CPU which consist of the components: **tok2vec, tagger, parser, senter, ner, attribute_ruler, lemmatizer**.

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

  - Creating the list named (**NER1**) of all the unique tokens and their respective NER tags and storing it in a text file named **NameEntity.txt**.

```
doc2 = docx.Document("preProcessedData.docx")
NER1 = []

for para in doc2.paragraphs:
    doc = nlp(para.text)
    for e in doc.ents:
        ents = e.text + " " + e.label_
        if (ents not in NER1):
            NER1.append(ents)
with open("NameEntity.txt", "w", encoding='utf-8') as outfile:
    outfile.write("\n".join(NER1))
```

➔ Creating a list named **nounNER** which contains the NER tagging of all nouns present in our data set.

```
nounNER = []
for word in nouns:
    doc = nlp(word)
    for e in doc.ents:
        ents = e.text + " " + e.label_
        nounNER.append(ents)
# print(nounNER)
with open("nounNameEntity.txt", "w", encoding='utf-8') as outfile:
    outfile.write("\n".join(nounNER))
```

➔ Creating a list named **verbNER** which contains the NER tagging of all verbs present in our data set.

```
verbNER = []
for word in verbs:
    doc = nlp(word)
    for e in doc.ents:
        ents = e.text + " " + e.label_
        verbNER.append(ents)
# print(verbNER)
with open("verbNameEntity.txt", "w", encoding='utf-8') as outfile:
    outfile.write("\n".join(verbNER))
```

- ## Recognizing all entities and their types of Test set :

    ➔ We have considered the random 3 pages of our pre-processed data as the test data.

    ➔ The Test Data file is named **TestData.docx**.

➔ Creating the list named (**NER2**) of all the unique tokens of our Test data file and their respective NER tags and storing it in a text file named as **NameEntityTest.txt**.

```
NER2 = []

for para in doc2.paragraphs:
    doc = nlp(para.text)
    for e in doc.ents:
        ents = e.text + " " + e.label_
        if (ents not in NER2):
            NER2.append(ents)

with open("NameEntityTest.txt", "w", encoding='utf-8') as outfile:
    outfile.write("\n".join(NER2))
```

➔ Creating the predicted List and Actual List of NER tags of the Test data set.

➔ Storing the Predicted List and Actual List in a text file named as **predictedTest.txt** and **actualTest.txt** respectively.

```
with open('predictedTest.txt', "r") as word_list:
    predictedList = word_list.read().split('\n')
print(predictedList)
with open('actualTest.txt', "r") as word_list:
    actualList = word_list.read().split('\n')
print(actualList)
```

- **Using performance measures to measure the performance of the method used :**

  ➔ We have included **sklearn.metrics** library which provides following function:
  f1_score: for calculation of the F1 score for the model.
  accuracy_score: for calculation of the accuracy score for the model.

  ➔ Functions are implemented using actualList, predictedList as the parameters passed to both the functions as shown in the code.

```
# ---------------------------------Calculating F1 score and Accuracy---------------------------

f1 = f1_score(actualList, predictedList, average='micro')
print("F1 score for the model is: ",f1)

acc = accuracy_score(actualList, predictedList)
print("Accuracy score for the model is: ",acc)
```

OUTPUT for F1 and Accuracy score respectively

```
F1 score for the model is:  0.8000000000000002
Accuracy score for the model is:  0.8
```

Part - 3

- **Recognizing relationship between entities:**

  ➔ Firstly, identify the domains of the entities. Assign each of them a serial number. Here we have done domain specification.

DOMAINS:

```
DOMAINS:

Department of Computer Science and Engineering IIT Delhi    1
India   2
Ian Mackie  3
cole Polytechnique  4
France  5
University of Sussex    6
UK  7
Samson Abramsky    8
University of Oxford    9
Chris Hankin    10
Imperial College    11
London  12
Dexter Kozen    13
Cornell University  14
Andrew Pitts    15
University of Cambridge 16
Hanne Riis Nielson  17
Technical University of Denmark    18
Denmark Steven Skiena   19
Stony Brook University    20
USA    21
David Zhang    22
The Hong Kong Polytechnic University    23
Hong Kong    24
Software Engineering    25
one    26
sumedha    27
shikha    28
sunanda    29
approximately 10000    30
Java    31
5000    32
1000    33
LOC    34
as low as 100    35
```

- ## Extracting additional features and presenting the table for the classes and relations:

  - ➔ Now we have classified them into respective domains.

CLASSES:

```
CLASSES:

ORG={1,4,6,9,11,14,16,18,20,23,25,34}
PERSON={3,8,10,13,15,17,19,22,27,28,29}
GPE={2,5,7,12,21,24}
CARDINAL={26,30,32,33,35}
LANGUAGE={31}
```

  - ➔ Now we have specified the relationships between entities.

RELATIONS:

```
RELATIONS:

situated_in = {<1,2>, <4,5>, <6,7>, <9,7>, <12,7>, <11,12>, <14,21>, <16,7>, <20,21>, <23,24>}
works_at = {<3,4>, <8,9>, <10,11>, <13,14>, <15,16>, <17,18>, <19,20>, <22,23>}
remains = {<25,26>}
chapter_problem_domain_of = {<26,25>}
daughter_of = {<27,28>, <29,28>}
lines_of = {<30,31>}
value_of = {<32,34>, <33,34>, <35,34>}
```

  - ➔ We have extracted the above shown data and stored it in the text file named **relationTable.txt**
    The code for printing the above text file is shown below.

```
f = open('relationTable.txt', 'r')
content = f. read()
print(content)
f.close()
```

CONCLUSION:

We have processed different algorithms and acknowledged the working of Natural Language Processing algorithms.
We got to know the different data preprocessing steps like tokenization, lemmatization, stemming which prepared our raw data for making the word cloud.
Word cloud functions are used to create the word cloud with and without stop words. POS tagging with treebank tagset is performed.

Then in the second round of the project we have determined the nouns and verbs in the book. Gets the category for that word in WordNet. Get the frequency of each category of each noun and verb within each hierarchy. Visually infer the frequency of each category of each noun and verb within each hierarchy.

After that we have recognized all entities and their types of Data set. Recognizing all entities and their types of Test set. Using performance measures to measure the performance of the method used.

Then using the ideas given in the book and presented in the class to augment the data of Entities. Augmenting that data by extracting additional features and building the table to present it.

- ***GitHub Link:***
  The project round 1 and round 2 combined code is available in the following gitHub link: https://github.com/Durvesh021/TextNex-Harnessing-NLP-for-Book-Analysisython_Code/nlp_project_R1_R2.py