NLP PROJECT

Round 1

Team: Random 4

Submitted to: **Dr. Sakthi Balan**

Submitted By:

Aneesh Khunteta (20ucs020)

Viral Bambori (20ucs230)

Vinay Sunil Bhagwat (20ucs227)

Chiroshree Tiwari (20ucs058)

Chinnaga Somasekhar (20ucs056)

ACKNOWLEDGEMENT

We would like to express our thanks of gratitude to our professor, Dr. Sakthi Balan, who gave us the opportunity to do this project which helped in increasing our knowledge and skills. Thank you so much for your continuous support and presence whenever needed.

We would like to thank everyone who is involved in the project directly or indirectly and helped us in finishing this project within time. At last, we are always thankful to the Almighty for always showing us the path.

Aim

The aim of this part of the project is to obtain PoS tagging for a computer science book that we used in one of our courses.

Code Link: https://github.com/ANEESH-2003/NLP-PROJECT-RANDOM_4/blob/main/final_code.py

Git Link: https://github.com/ANEESH-2003/NLP-PROJECT-RANDOM_4

Data Description

The book which we have used in our project is:

Understanding Cryptography, by Christof Paar · Jan Pelzl

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

```
#Importing Files

f1 = open("Cryptography_by_Cristopher.txt", "r", encoding="utf-8")

# Main Content extractor

# For taking the main lines of the book 1 : 'Cryptography_by_Cristopher'

with open("Cryptography_by_Cristopher.txt", encoding="utf-8") as book:
    lines_1 = book.readlines()

begin_index = lines_1.index("Chapter 1\n")
end_index = len(lines_1) - 1 - lines_1[::-1].index("Certificates\n")

print("The main content of the book for text T1 is from line numbers {} to {}".format(
    begin_index, end_index))

lines_1 = lines_1[begin_index:end_index]
```

Output is as follows:

```
In [2]: runfile('C:/Users/HP/OneDrive/Desktop/NLP Project/
original.py', wdir='C:/Users/HP/OneDrive/Desktop/NLP Project')
The main content of the book for text T1 is from line numbers 640
to 18802
```

Number of words in the book are:

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

```
number of words in T1: 116089
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Firstly, we have imported some libraries for various tasks

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

Importing the Text files, calling it T1.

- We have imported the text file 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 “Understanding Cryptography” Code for the same is given below.

```
from nltk.corpus import stopwords
from collections import defaultdict
#from wordcloud import WordCloud, STOPWORDS
from nltk.probability import FreqDist as FD
import string
import re
from nltk.tokenize import sent_tokenize, word_tokenize
import pandas as pnds
import matplotlib.pyplot as mplp
import nltk
import operator
nltk.download('punkt')

#Importing Files

f1 = open("Cryptography_by_Cristopher.txt", "r", encoding="utf-8")
```

Text Pre processing

- For text/data pre-processing 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]. ○ 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:

```
34
35  def empty_line_remover(lines):  # function for removing the running words & empty lines
36
37      chapter_pattern = r"Chapter [IVX]+"
38
39      temp = []
40      for line in lines:
41          is_valid = ((line == '\n') or re.match(chapter_pattern, line))
42          # If the line is neither a chapter number nor a part heading nor an empty line
43          if(not is_valid):
44              temp.append(line)          # include it in the final list
45
46      return temp
47
48
49  lines_1 = empty_line_remover(lines_1)
50
51
52  T1 = ''.join(lines_1)  # joining the lines to string
53
54
55
```

● Code for text processor:

```
59
60  def text_preprocessor(text):  # Function to clean books
61
62      text = text.lower()
63      text = re.sub('https?://\S+|www\.\S+', '', text)  # to remove links
64      text = re.sub('\s', '_', text)            # Replacing spaces with '_'
65      # Removing non-alphanumeric characters
66      text = re.sub(r'\W+', '', text)
67      text = re.sub('_', ' ', text)             # Replacing spaces with '_'
68      return text
69
70  T1 = text_preprocessor(T1)
71  print(T1)  # printing the cleaned text of book T1
72
```

● Output after this process for T1 file is:

```
In [5]: runfile('C:/Users/HP/OneDrive/Desktop/NLP Project/original.py', wdir='C:/Users/HP/OneDrive/Desktop/NLP Project')
The main content of the book for text T1 is from line numbers 640 to 18802
number of words in T1: 116089
chapter 1 introduction to cryptography and data security this section will introduce the most important terms of modern cryptology
and will teach an important lesson about proprietary vs openly known algorithms we will also introduce modular arithmetic which is
also of major importance in publickey cryptography    in this chapter you will learn  the general rules of cryptography  key length
for short medium and longterm security  the difference between different types of attacks against ciphers  afew historical ciphers
and on the way we will learn about modular arithmetic    which is of major importance for modern cryptography as well  why one
should only use wellestablished encryption algorithms c paar j pelzl understanding cryptography                    1 doi
10100797836420410 13 1 c  springerverlag berlin heidelberg 2010 18 2                                                  1
introduction to cryptography and data security 11 overview of cryptology and this book if we hear the word cryptography our first
associations might be email encryption secure website access smart cards for banking applications or code breaking during world war
ii such as the famous attack against the german enigma encryption machine fig 11 fig 11 the german enigma encryption machine
reproduced with permission from the deutsches museum munich     cryptography seems closely linked to modern electronic communicatio
how ever cryptography is a rather old business with early examples dating back to about 2000 bc when nonstandard secret
hieroglyphics were used in ancient egypt since egyptian days cryptography has been used in one form or the other in many if not mos
cultures that developed written language for instance there are doc umented cases of secret writing in ancient greece namely the
scytale of sparta fig 12 or the famous caesar cipher in ancient rome about which we will learn later in this chapter this book
however strongly focuses on modern cryptographic                                    thescytal
eisatran                                        spositio                                        ncipher fig 12
scytale of sparta methods and also teaches many data security issues and their relationship with cryp tography     lets now have a
look at the field of cryptography fig 13 the first thing 19 11 overview of cryptology and this book
```

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_tokenize function for T1 text is:

```
73  token1 = word_tokenize(T1)  # Tokenizing the book T1
74  print(token1)
75
```

```
['chapter', '1', 'introduction', 'to', 'cryptography', 'and',
'data', 'security', 'this', 'section', 'will', 'introduce',
'the', 'most', 'important', 'terms', 'of', 'modern',
'cryptology', 'and', 'will', 'teach', 'an', 'important',
'lesson', 'about', 'proprietary', 'vs', 'openly', 'known',
'algorithms', 'we', 'will', 'also', 'introduce', 'modular',
'arithmetic', 'which', 'is', 'also', 'of', 'major',
'importance', 'in', 'publickey', 'cryptography', 'in',
'this', 'chapter', 'you', 'will', 'learn', 'the', 'general',
'rules', 'of', 'cryptography', 'key', 'lengths', 'for',
'short', 'medium', 'and', 'longterm', 'security', 'the',
'difference', 'between', 'different', 'types', 'of',
'attacks', 'against', 'ciphers', 'afew', 'historical',
'ciphers', 'and', 'on', 'the', 'way', 'we', 'will', 'learn',
'about', 'modular', 'arithmetic', 'which', 'is', 'of',
'major', 'importance', 'for', 'modern', 'cryptography', 'as',
'well', 'why', 'one', 'should', 'only', 'use',
'wellestablished', 'encryption', 'algorithms', 'c', 'paar',
'j', 'pelzl', 'understanding', 'cryptography', '1', 'doi',
'10100797836420410 13', '1', 'c', 'springerverlag', 'berlin',
'heidelberg', '2010', '18', '2', '1', 'introduction', 'to',
'cryptography', 'and', 'data', 'security', '11', 'overview',
'of', 'cryptology', 'and', 'this', 'book', 'if', 'we',
'hear', 'the', 'word', 'cryptography', 'our', 'first',
```

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 Frequency Distribution for the T1 tokenized text is:

```
fd1 = FD(token1)  # frequency distribution of token 1
print(fd1)

# Frequency distribution

K = 8443
list_fd1 = dict(sorted(fd1.items(), key=operator.itemgetter(1), reverse=True))
list_output_fd1 = dict(list(list_fd1.items())[0: K])

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

```
<FreqDist with 8443 samples and 113834 outcomes>
K highest frequency words for text T1 are : {'the': 7474, 'of': 3107, 'is': 2691, 'a': 2556, 'in': 2222, 'and': 1850, 'to': 1819, 'for': 1345, 'that': 1304, 'are': 1126,
'we': 1116, 'key': 1086, 'with': 911, 'as': 848, 'this': 823, 'be': 802, 'can': 714, 'which': 695, 'an': 694, '1': 600, 'by': 584, 'it': 523, 'not': 481, 'on': 447,
'encryption': 443, 'if': 414, '2': 410, 'have': 407, 'block': 404, 'mod': 388, 'bit': 387, 'cipher': 378, 'one': 363, 'or': 345, 'p': 330, 'message': 327, 'used': 323,
'x': 312, 'two': 302, 'from': 301, 'ciphers': 296, 'hash': 292, 'all': 287, 'rsa': 287, 'algorithm': 285, 'security': 283, 'at': 280, 'bits': 270, 'attack': 265, 'there'
264, 'aes': 263, 'only': 262, 'has': 254, 'signature': 240, 'such': 239, '3': 235, 'des': 234, 'function': 229, 'also': 226, 'example': 220, 'output': 217, 'more': 216,
'length': 212, 'first': 210, 'bob': 210, 'group': 210, 'stream': 210, 'ie': 208, 'compute': 207, 'alice': 203, 'attacks': 200, '0': 199, 'since': 198, 'was': 197, 'many'
191, 'now': 190, 'problem': 189, 'given': 187, 'algorithms': 185, 'number': 185, 'how': 183, 'other': 183, 'keys': 182, 'order': 182, 'cryptography': 180, 'where': 180,
'using': 178, 'b': 178, 'each': 175, 'prime': 175, 'value': 173, 'fig': 172, 'publickey': 171, 'use': 165, 'plaintext': 163, 'digital': 162, 'note': 161, 'input': 161,
'elliptic': 161, 'scheme': 160, 'but': 157, 'ciphertext': 157, '4': 155, 'table': 153, 'following': 153, 'most': 150, 'oscar': 149, 'k': 149, 'round': 148, 'decryption':
147, 'curve': 147, 'operation': 146, 'will': 145, 'n': 145, 'element': 144, 'they': 143, 'public': 143, 'however': 142, 'functions': 142, '11': 139, 'about': 137,
'should': 137, 'both': 135, 'elements': 135, 'than': 135, 'assume': 134, 'case': 130, 'same': 130, 'mode': 130, 'discrete': 129, 'these': 127, 'eg': 126, 'symmetric': 12
'very': 125, 's': 125, 'secure': 124, 'even': 124, 'addition': 124, 'multiplication': 123, 'data': 121, 'practice': 121, 'do': 120, 'c': 119, 'what': 119, 'some': 119,
'm': 119, '10': 119, 'against': 118, '12': 118, '5': 118, 'inverse': 118, 'field': 117, 'based': 117, 'every': 117, 'logarithm': 117, 'values': 116, 'over': 115, '13':
110, 'possible': 110, 'point': 109, 'time': 108, 'he': 108, '8': 107, 'e': 107, 'secret': 105, 'd': 105, 'signatures': 104, 'any': 104, '15': 104, 'protocol': 103,
'often': 103, 'three': 102, 'does': 102, 'random': 100, 'q': 100, 'numbers': 100, 'sect': 99, 'attacker': 99, 'schemes': 98, 'standard': 98, 'important': 97, 'thus': 97,
'theorem': 97, 'i': 95, 'integer': 94, 'you': 93, 'look': 93, 'y': 93, 'private': 93, 'known': 92, 'between': 92, 'different': 92, 'second': 92, 'must': 92, '32': 92,
'x1': 92, '7': 91, 'let': 91, 'been': 90, 'instance': 90, 'no': 90, 'shown': 90, 'computed': 90, 'several': 89, '6': 89, 'x2': 89, 'chapter': 87, 'lets': 87, 'fact': 87,
'so': 87, 'long': 86, 'section': 84, 'elgamal': 84, 'layer': 83, 'applications': 82, 'g': 82, 'though': 82, '16': 81, 'cyclic': 81, 'large': 80, 'method': 79, 'choose':
79, 'hence': 78, 'session': 78, 'introduced': 77, 'messages': 77, 'new': 77, 'authentication': 77, 'result': 77, 'operations': 77, 'into': 76, 'need': 76, 'follows': 76,
'cryptographic': 75, 'r': 75, 'its': 75, 'modulo': 75, 'curves': 75, '14': 74, 'would': 74, 'consider': 74, 'encrypted': 74, 'implementation': 74, 'small': 74, 'hardware
74, 'were': 73, 'us': 73, 'size': 73, 'properties': 72, 'here': 72, 'after': 72, 'called': 71, '9': 71, 'perform': 71, '128': 71, 'arithmetic': 70, 'above': 70, 'blocks'
69, 'system': 69, 'then': 69, 'bruteforce': 68, 'search': 68, 'computing': 68, 'sbox': 68, 'substitution': 67, 'xi': 67, 'parameters': 67, 'encrypt': 67, 'practical': 6
'definition': 65, 'means': 65, 'integers': 65, 'lengths': 64, '01': 64, 'iv': 64, 'groups': 64, 'collision': 63, 'cryptosystems': 62, 'find': 62, 'out': 62, 'simple': 62
'much': 62, 't': 62, 'show': 62, 'introduction': 61, 'asymmetric': 61, 'provide': 61, 'finite': 61, 'verification': 61, 'way': 60, 'because': 60, 'set': 60, 'macs': 60,
'general': 59, 'still': 59, 'zp': 59, 'modular': 58, 'know': 58, 'another': 58, '64': 58, 'generate': 58, 'byte': 58, 'lfsr': 58, 'a': 58, 'whether': 57, 'software': 57,
'fields': 57, 'state': 57, 'see': 56, '26': 56, 'f': 56, 'verify': 56, 'generated': 56, 'primitive': 56, 'short': 55, 'well': 54, 'why': 54, 'when': 54, 'basic': 54,
'want': 54, 'needed': 54, 'ecc': 54, 'problems': 53, 'steps': 53, 'last': 53, 'diffiehellman': 53, 'polynomial': 53, 'might': 52, 'through': 52, 'h': 51, 'certain': 51,
'form': 50, 'results': 50, 'like': 50, 'easily': 50, 'found': 50, '256': 50, 'shift': 50, 'single': 50, 'si': 50, '03': 50, 'modern': 49, 'chosen': 49, 'process': 49,
'23': 49, 'feedback': 49, 'generation': 49, 'equation': 49, 'four': 49, '02': 49, 'sha1': 49, 'during': 48, 'tion': 48, 'again': 48, 'your': 48, 'modulus': 48, 'valid':
```

● 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.

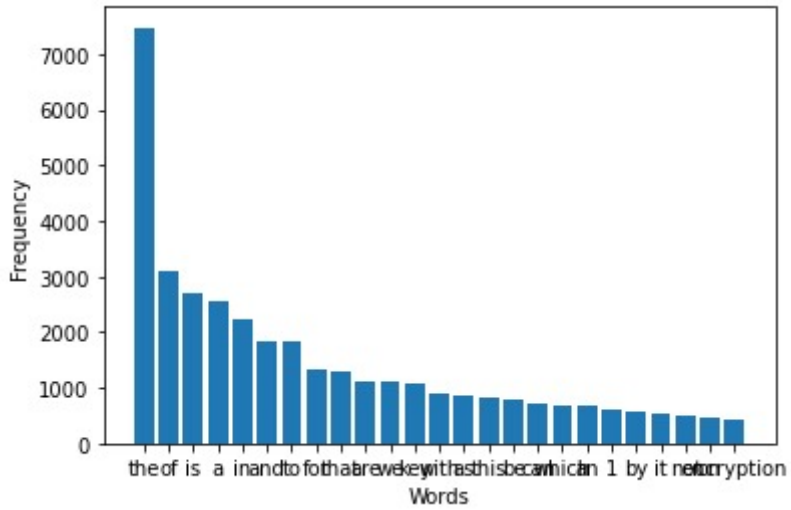
● Code for the same is:

```
K = 25
list_fd1 = dict(sorted(fd1.items(), key=operator.itemgetter(1), reverse=True))
list_output_fd1 = dict(list(list_fd1.items())[0: K])

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


# bar graph of word-frequency of fd1
mplp.bar(list_output_fd1.keys(), list_output_fd1.values())
mplp.xlabel('Words')
mplp.ylabel('Frequency')
mplp.show()
```

● Bar graph for the output of T1 is:



Creating word cloud for the text T1:

● 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

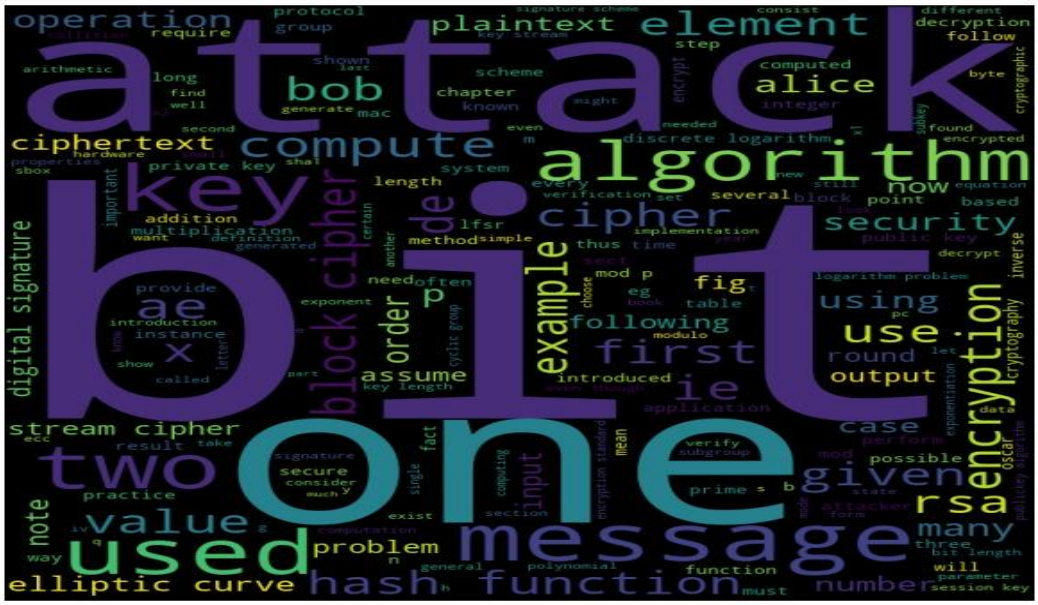
● Code :

```
# Word Cloud Formation

# wordcloud of T1
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:



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:

```
#StopWords Removal and Cloud Forming

nltk.download('stopwords')

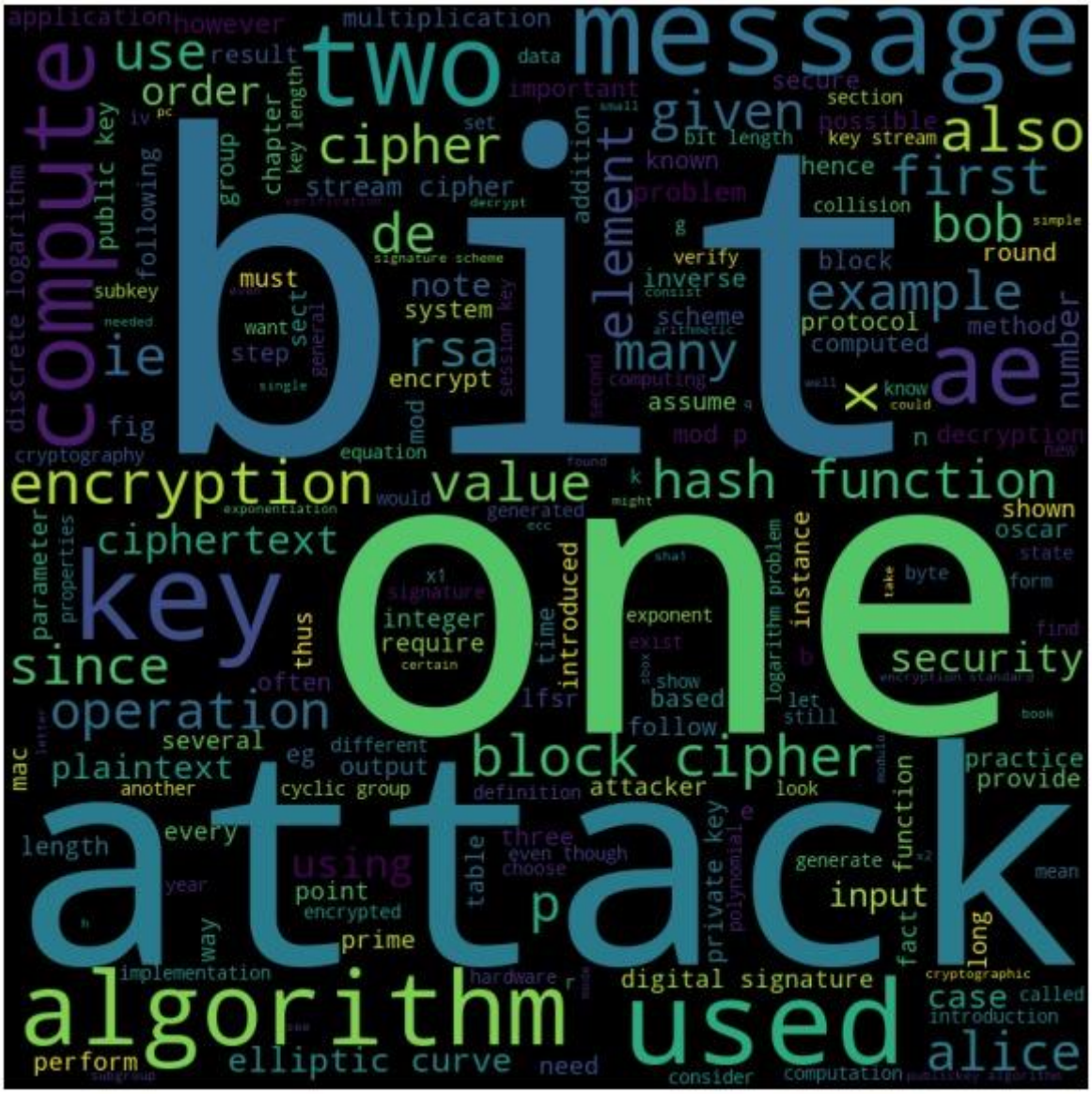
all_stopwords = stopwords.words('english')
```

● 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:

```
114  # wordcloud of T1 after removing stop word
115  word_cloud_instance = WordCloud(width=800, height=800, background_color='black',
116                                  stopwords=all_stopwords, min_font_size=8).generate(T1)
117
118  mplp.figure(figsize=(8, 8), facecolor=None)
119  mplp.imshow(word_cloud_instance)
120  mplp.axis("off")
121  mplp.tight_layout(pad=0)
122  mplp.show()
123
```

● Word cloud after removing stopwords for text T1 is:



Relationship between the word length and frequency for T1:

● 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

```
words = {}


def word_counter(text):  # function for matching number of words to its length

    for word in text.split():

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

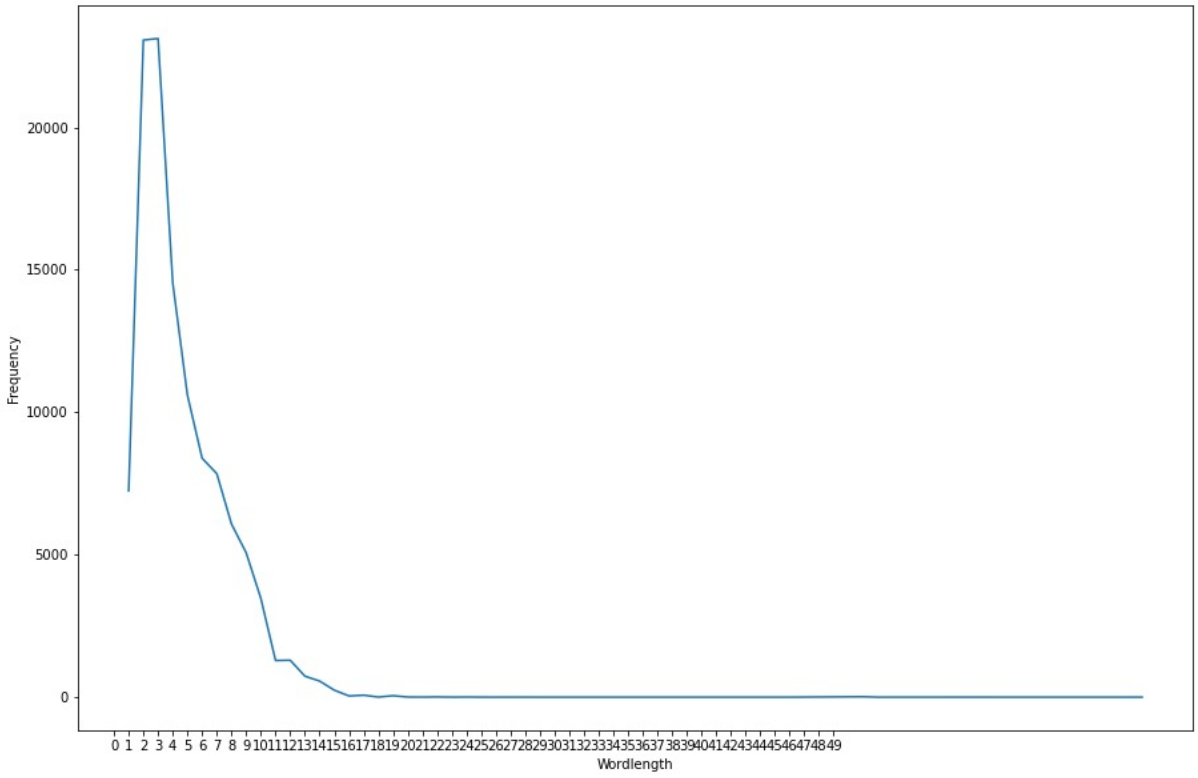
● For Text T1-

● 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:



PoS Tagging

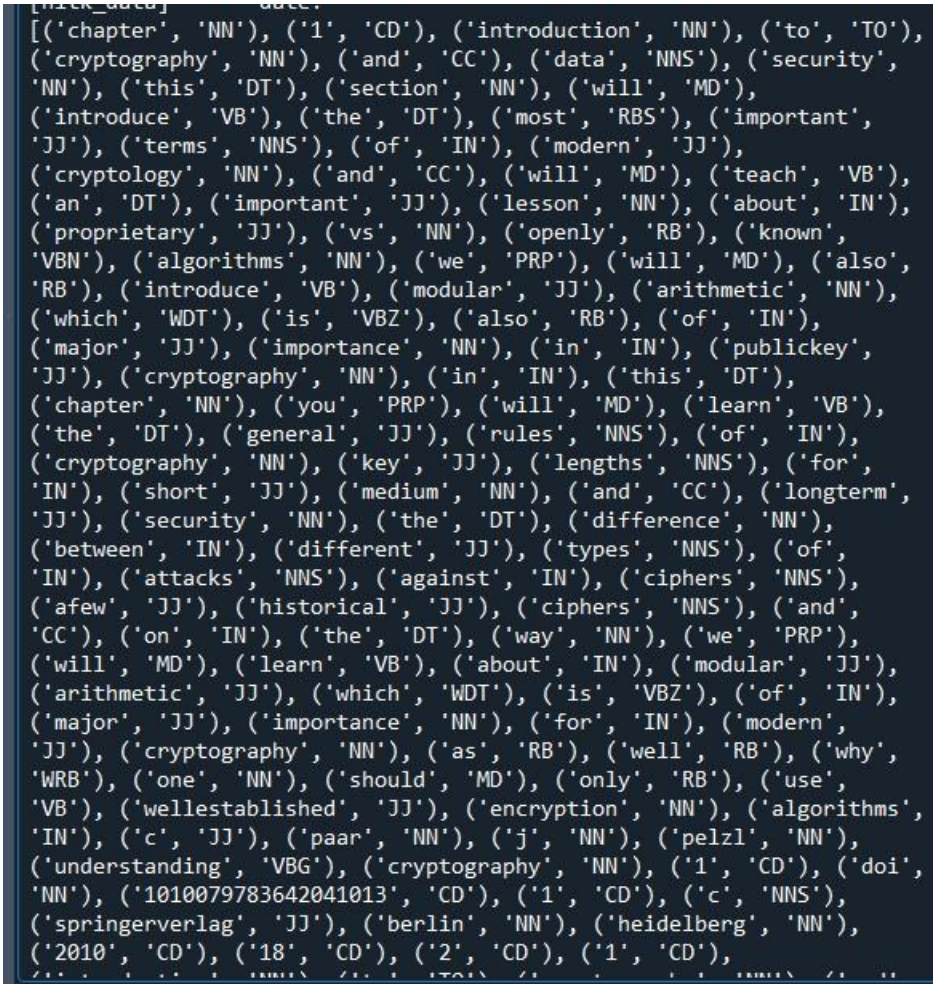
● Now the main task comes to give each of the tokens their tags. We have used 'averaged_perceptron_tagger' for POS tagging of the tokens. We have downloaded the same from nltk.

● Code for T1:

```
# PoS Tagging
nltk.download('averaged_perceptron_tagger')

tagged1 = nltk.pos_tag(token1)  # doing pos tagging of token1
tagged1
```

● Output for T1:



```
[('chapter', 'NN'), ('1', 'CD'), ('introduction', 'NN'), ('to', 'TO'),
('cryptography', 'NN'), ('and', 'CC'), ('data', 'NNS'), ('security',
'NN'), ('this', 'DT'), ('section', 'NN'), ('will', 'MD'),
('introduce', 'VB'), ('the', 'DT'), ('most', 'RBS'), ('important',
'JJ'), ('terms', 'NNS'), ('of', 'IN'), ('modern', 'JJ'),
('cryptology', 'NN'), ('and', 'CC'), ('will', 'MD'), ('teach', 'VB'),
('an', 'DT'), ('important', 'JJ'), ('lesson', 'NN'), ('about', 'IN'),
('proprietary', 'JJ'), ('vs', 'NN'), ('openly', 'RB'), ('known',
'VBN'), ('algorithms', 'NN'), ('we', 'PRP'), ('will', 'MD'), ('also',
'RB'), ('introduce', 'VB'), ('modular', 'JJ'), ('arithmetic', 'NN'),
('which', 'WDT'), ('is', 'VBZ'), ('also', 'RB'), ('of', 'IN'),
('major', 'JJ'), ('importance', 'NN'), ('in', 'IN'), ('publickey',
'JJ'), ('cryptography', 'NN'), ('in', 'IN'), ('this', 'DT'),
('chapter', 'NN'), ('you', 'PRP'), ('will', 'MD'), ('learn', 'VB'),
('the', 'DT'), ('general', 'JJ'), ('rules', 'NNS'), ('of', 'IN'),
('cryptography', 'NN'), ('key', 'JJ'), ('lengths', 'NNS'), ('for',
'IN'), ('short', 'JJ'), ('medium', 'NN'), ('and', 'CC'), ('longterm',
'JJ'), ('security', 'NN'), ('the', 'DT'), ('difference', 'NN'),
('between', 'IN'), ('different', 'JJ'), ('types', 'NNS'), ('of',
'IN'), ('attacks', 'NNS'), ('against', 'IN'), ('ciphers', 'NNS'),
('afew', 'JJ'), ('historical', 'JJ'), ('ciphers', 'NNS'), ('and',
'CC'), ('on', 'IN'), ('the', 'DT'), ('way', 'NN'), ('we', 'PRP'),
('will', 'MD'), ('learn', 'VB'), ('about', 'IN'), ('modular', 'JJ'),
('arithmetic', 'JJ'), ('which', 'WDT'), ('is', 'VBZ'), ('of', 'IN'),
('major', 'JJ'), ('importance', 'NN'), ('for', 'IN'), ('modern',
'JJ'), ('cryptography', 'NN'), ('as', 'RB'), ('well', 'RB'), ('why',
'WRB'), ('one', 'NN'), ('should', 'MD'), ('only', 'RB'), ('use',
'VB'), ('wellestablished', 'JJ'), ('encryption', 'NN'), ('algorithms',
'IN'), ('c', 'JJ'), ('paar', 'NN'), ('j', 'NN'), ('pelzl', 'NN'),
('understanding', 'VBG'), ('cryptography', 'NN'), ('1', 'CD'), ('doi',
'NN'), ('1010079783642041013', 'CD'), ('1', 'CD'), ('c', 'NNS'),
('springerverlag', 'JJ'), ('berlin', 'NN'), ('heidelberg', 'NN'),
('2010', 'CD'), ('18', 'CD'), ('2', 'CD'), ('1', 'CD'),
```

Distribution of various tags for text T1:

● Code is:

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

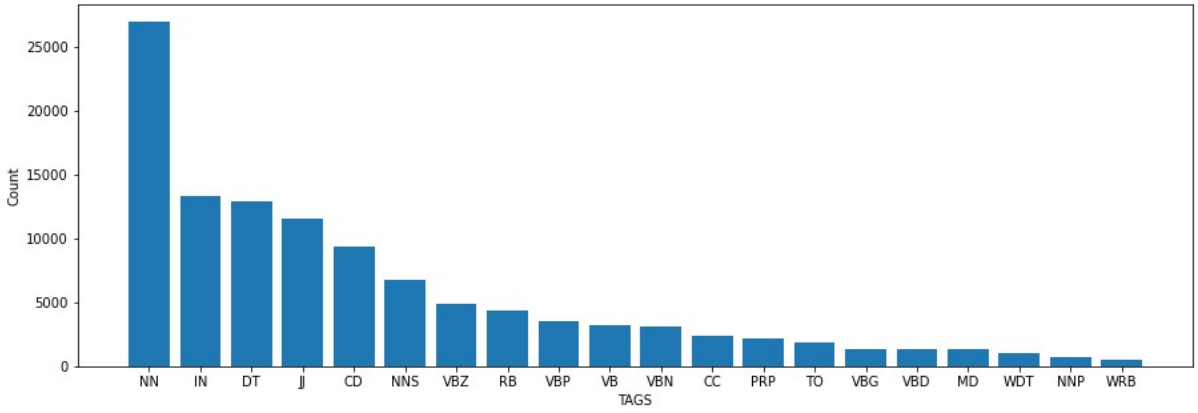
sorted_d1 = dict(
    sorted(dict1.items(), key=operator.itemgetter(1), reverse=True))

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

mplp.bar(out1.keys(), out1.values())
mplp.xlabel('TAGS')
mplp.ylabel('Count')
mplp.show()
```

● 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:



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 'one' 'bit' 'and' 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

GIT Link: https://github.com/ANEESH-2003/NLP-PROJECT-RANDOM_4

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

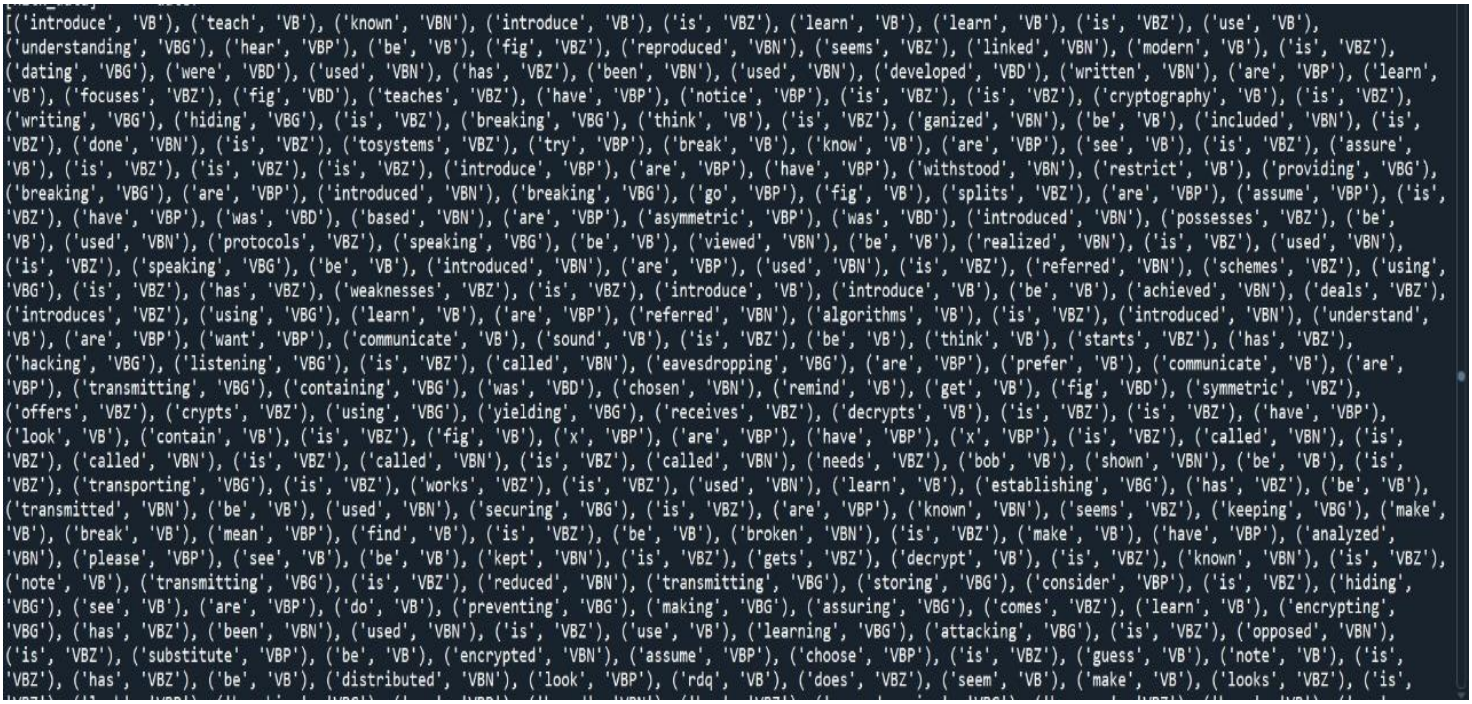
For finding nouns and verbs , we looped through the whole tagged text and using conditioning we have separated nouns and verbs from the text. For text T1

Code :

```
176
177  # After the Pos Tagging we need to find the  verbs and nouns from the tagged text
178
179  T1_nouns = []
180  T1_verbs = []
181
182  for i in tagged1:
183    if(i[1] == "NN" or i[1] == "NNS" or i[1] == "NNP" or i[1] =="NNPS"):
184      T1_nouns.append(i)
185    elif (i[1]=="VB" or i[1]=="VBD" or i[1]=="VBG" or i[1]=="VBN" or i[1]=="VBP" or i[1]=="VBZ"):
186      T1_verbs.append(i)
187
188  print(T1_nouns)
189  print(T1_verbs)
190
```

Output:

```
[('chapter', 'NN'), ('introduction', 'NN'), ('cryptography', 'NN'), ('data', 'NNS'), ('security', 'NN'), ('section', 'NN'), ('terms', 'NNS'), ('cryptology', 'NN'),
('lesson', 'NN'), ('vs', 'NN'), ('algorithms', 'NN'), ('arithmetic', 'NN'), ('importance', 'NN'), ('cryptography', 'NN'), ('chapter', 'NN'), ('rules', 'NNS'),
('cryptography', 'NN'), ('lengths', 'NNS'), ('medium', 'NN'), ('security', 'NN'), ('difference', 'NN'), ('types', 'NNS'), ('attacks', 'NNS'), ('ciphers', 'NNS'),
('ciphers', 'NNS'), ('way', 'NN'), ('importance', 'NN'), ('cryptography', 'NN'), ('one', 'NN'), ('encryption', 'NN'), ('paar', 'NN'), ('j', 'NN'), ('pelzl', 'NN'),
('cryptography', 'NN'), ('doi', 'NN'), ('c', 'NNS'), ('berlin', 'NN'), ('heidelberg', 'NN'), ('introduction', 'NN'), ('cryptography', 'NN'), ('data', 'NNS'), ('security',
'NN'), ('overview', 'NN'), ('cryptology', 'NN'), ('book', 'NN'), ('word', 'NN'), ('cryptography', 'NN'), ('associations', 'NNS'), ('encryption', 'NN'), ('secure', 'NN'),
('access', 'NN'), ('cards', 'NNS'), ('banking', 'NN'), ('applications', 'NNS'), ('code', 'NN'), ('breaking', 'NN'), ('world', 'NN'), ('war', 'NN'), ('ii', 'NN'),
('attack', 'NN'), ('enigma', 'NN'), ('encryption', 'NN'), ('machine', 'NN'), ('fig', 'NN'), ('enigma', 'NN'), ('encryption', 'NN'), ('machine', 'NN'), ('permission',
'NN'), ('deutsches', 'NNS'), ('museum', 'NN'), ('munich', 'NN'), ('cryptography', 'NN'), ('communication', 'NN'), ('cryptography', 'NN'), ('business', 'NN'), ('examples',
'NNS'), ('bc', 'NN'), ('nonstandard', 'NN'), ('hieroglyphics', 'NNS'), ('egypt', 'NN'), ('days', 'NNS'), ('cryptography', 'NN'), ('form', 'NN'), ('cultures', 'NNS'),
('language', 'NN'), ('instance', 'NN'), ('cases', 'NNS'), ('writing', 'NN'), ('greece', 'NN'), ('scytale', 'NN'), ('sparta', 'NN'), ('fig', 'NN'), ('caesar', 'NN'),
('cipher', 'NN'), ('rome', 'NN'), ('chapter', 'NN'), ('book', 'NN'), ('thescytal', 'NN'), ('eisatran', 'NN'), ('spositio', 'NN'), ('scytale', 'NN'), ('methods', 'NNS'),
('data', 'NNS'), ('security', 'NN'), ('issues', 'NNS'), ('relationship', 'NN'), ('cryp', 'NN'), ('tography', 'NN'), ('lets', 'NNS'), ('look', 'NN'), ('field', 'NN'),
('cryptography', 'NN'), ('fig', 'NN'), ('thing', 'NN'), ('overview', 'NN'), ('cryptology', 'NN'), ('book', 'NN'), ('fig', 'NN'), ('overview', 'NN'), ('field', 'NN'),
('cryptology', 'NN'), ('term', 'NN'), ('cryptology', 'NN'), ('cryptology', 'NN'), ('splits', 'NNS'), ('branches', 'NNS'), ('cryptography', 'NN'), ('science', 'NN'),
('goal', 'NN'), ('ing', 'NN'), ('message', 'NN'), ('cryptanalysis', 'NN'), ('science', 'NN'), ('art', 'NN'), ('cryptosystems', 'NNS'), ('code', 'NN'), ('breaking', 'NN'),
('intelligence', 'NN'), ('community', 'NN'), ('crime', 'NN'), ('classification', 'NN'), ('scien', 'NN'), ('tific', 'NN'), ('discipline', 'NN'), ('cryptanalysis', 'NN'),
('researchers', 'NNS'), ('academia', 'NN'), ('nowadays', 'NNS'), ('cryptanalysis', 'NN'), ('importance', 'NN'), ('cryp', 'NN'), ('people', 'NNS'), ('methods', 'NNS'),
('discussion', 'NN'), ('issue', 'NN'), ('cryptanalysis', 'NN'), ('way', 'NN'), ('cryptosystem', 'NN'), ('part', 'NN'), ('cryptology', 'NN'), ('focus', 'NN'), ('book',
'NN'), ('cryptography', 'NN'), ('crypto', 'NN'), ('algorithms', 'NN'), ('detail', 'NN'), ('algorithms', 'NN'), ('cryptanalysis', 'NN'), ('time', 'NN'), ('cases', 'NNS'),
('decades', 'NNS'), ('case', 'NN'), ('cryptanalysis', 'NN'), ('ourselves', 'NNS'), ('results', 'NNS'), ('respect', 'NN'), ('crypto', 'NN'), ('al', 'NN'), ('gorithms',
'NN'), ('factoring', 'NN'), ('record', 'NN'), ('lets', 'NNS'), ('cryptography', 'NN'), ('branches', 'NNS'), ('algorithms', 'NNS'), ('people', 'NNS'), ('cryptography',
'NN'), ('parties', 'NNS'), ('encryption', 'NN'), ('decryption', 'NN'), ('method', 'NN'), ('share', 'NN'), ('key', 'NN'), ('cryptography', 'NN'), ('times', 'NNS'),
('methods', 'NNS'), ('ciphers', 'NNS'), ('use', 'NN'), ('data', 'NNS'), ('encryption', 'NN'), ('integrity', 'NN'), ('check', 'NN'), ('messages', 'NNS'), ('algorithms',
'NN'), ('type', 'NN'), ('cipher', 'NN'), ('whitfield', 'NN'), ('diffie', 'NN'), ('martin', 'NN'), ('hellman', 'NN'), ('ralph', 'NN'), ('merkle', 'NN'), ('cryptography',
'NN'), ('user', 'NN'), ('key', 'NN'), ('cryptog', 'NN'), ('raphy', 'NN'), ('key', 'NN'), ('algorithms', 'NN'), ('applications', 'NNS'), ('signatures', 'NNS'),
('establishment', 'NN'), ('data', 'NNS'), ('cryption', 'NN'), ('cryptographic', 'NN'), ('crypto', 'NN'), ('protocols', 'NNS'), ('deal', 'NN'), ('plication', 'NN'),
('algorithms', 'NN'), ('symmetric', 'NN'), ('algorithms', 'NN'), ('introduction', 'NN'), ('cryptography', 'NN'), ('data', 'NNS'), ('security', 'NN'), ('building', 'NN'),
('blocks', 'NNS'), ('applications', 'NNS'), ('communication', 'NN'), ('transport', 'NN'), ('layer', 'NN'), ('security', 'NN'), ('tls', 'NN'), ('scheme', 'NN'), ('web',
'NN'), ('browser', 'NN'), ('example', 'NN'), ('protocol', 'NN'), ('hash', 'NN'), ('functions', 'NNS'), ('form', 'NN'), ('class', 'NN'), ('algorithms', 'NN'), ('time',
'NN'), ('share', 'NN'), ('properties', 'NNS'), ('ciphers', 'NNS'), ('majority', 'NN'), ('applications', 'NNS'), ('systems', 'NNS'), ('algorithms', 'NN'), ('functions',
'NNS'), ('reason', 'NN'), ('families', 'NNS'), ('algorithms', 'NN'), ('strengths', 'NNS'), ('focus', 'NN'), ('book', 'NN'), ('algorithms', 'NN'), ('hash', 'NN'),
('functions', 'NNS'), ('security', 'NN'), ('protocols', 'NNS'), ('establishment', 'NN'), ('protocols', 'NNS'), ('crypto', 'NN'), ('protocols', 'NNS'), ('confidentiality',
'NN'), ('data', 'NNS'), ('integrity', 'NN'), ('data', 'NNS'), ('tion', 'NN'), ('data', 'NNS'), ('identification', 'NN'), ('cryptography', 'NN'), ('section', 'NN'),
```



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:

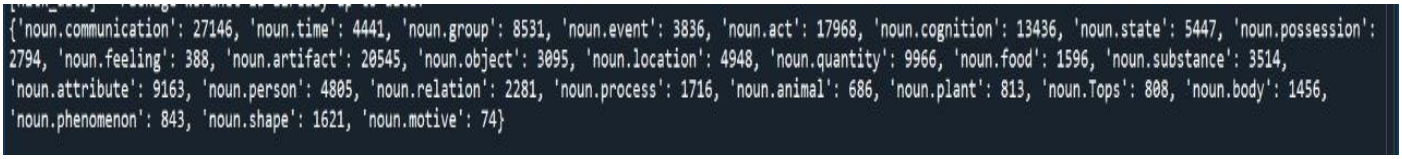
```
# importing wordnet from nltk corpus
from nltk.corpus import wordnet
nltk.download('wordnet')

# Categorizing Nouns (25 Categories)
T1_nouns_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

print(T1_nouns_dict)
```

Output:



For Verbs code:

```
208
209  # Now Categorizing the Verbs (16 Categories)
210  T1_verbs_dict = {}
211  for i in T1_verbs:
212    syn = wordnet.synsets(i[0])
213    for j in syn:
214      if j.lexname()[0]=='v':
215        if j.lexname() in T1_verbs_dict:
216          T1_verbs_dict[j.lexname()]+=1
217        else:
218          T1_verbs_dict[j.lexname()]=1
219
220  print(T1_verbs_dict)
221
```

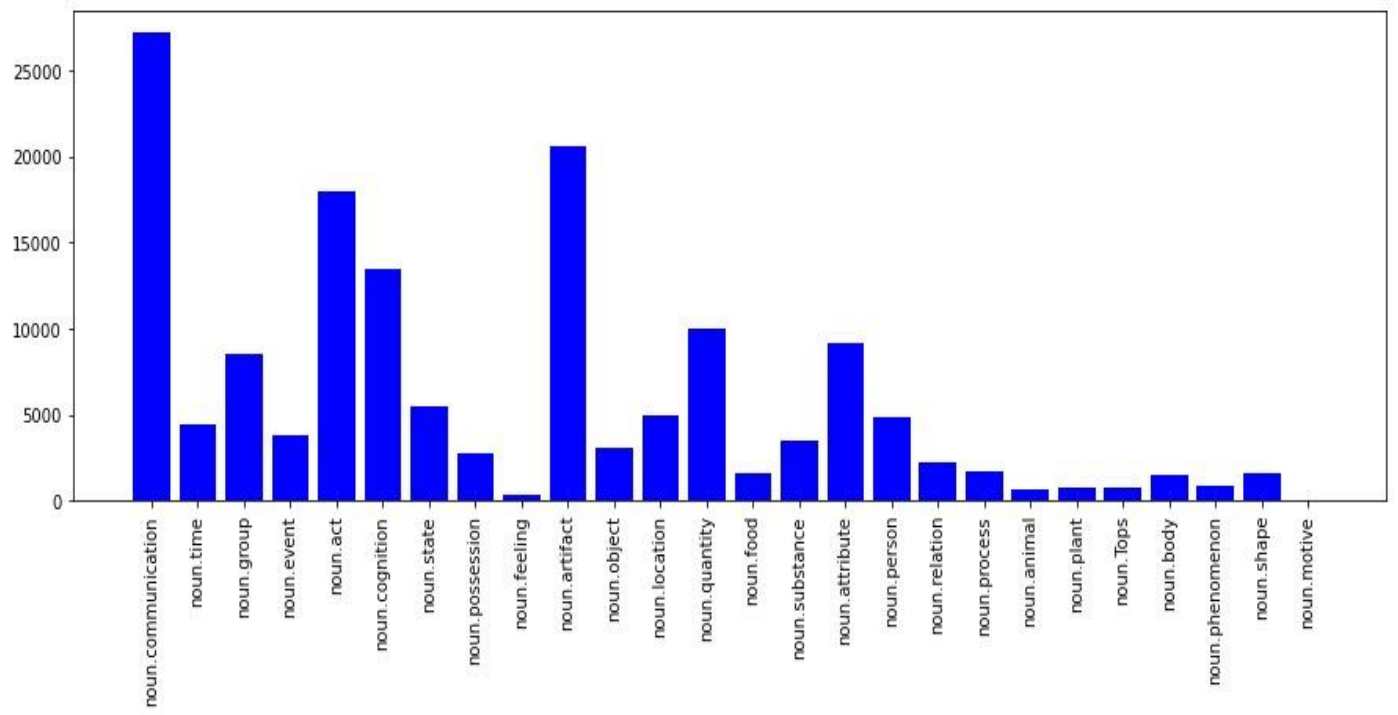
Output:

```
[nltk_data] Package wordnet is already up-to-date!
{'verb.communication': 13500, 'verb.creation': 8031, 'verb.change': 9369, 'verb.motion': 5805, 'verb.contact': 5222, 'verb.cognition': 10386, 'verb.stative': 65108,
'verb.social': 16183, 'verb.possession': 17388, 'verb.consumption': 4438, 'verb.perception': 4210, 'verb.body': 4886, 'verb.competition': 1520, 'verb.emotion': 590,
'verb.weather': 13}
```

Histograms for novel T1

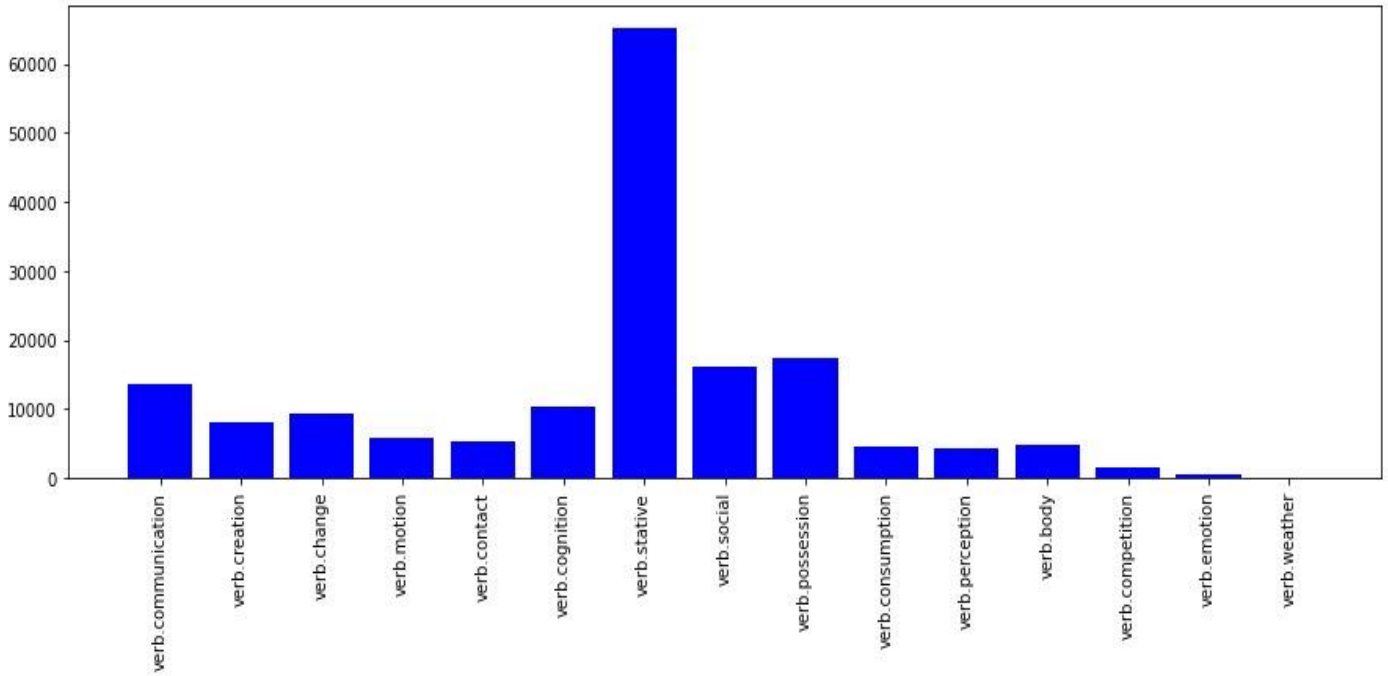
Nouns:

```
219
220  #print(T1_verbs_dict)
221
222  # Histogram for Nouns
223
224  mplp.bar(T1_nouns_dict.keys(), T1_nouns_dict.values(),color='b')
225  y_pos = range(len(T1_nouns_dict.keys()))
226  mplp.xticks(y_pos, T1_nouns_dict.keys(), rotation=90)
227  mplp.show()
228
229
```



Verbs:

```
229
230  # Histogram for Verbs
231
232  mplp.bar(T1_verbs_dict.keys(), T1_verbs_dict.values(),color='b')
233  y_pos = range(len(T1_verbs_dict.keys()))
234  mplp.xticks(y_pos, T1_verbs_dict.keys(), rotation=90)
235  mplp.show()
236
```



Part-2

Named Entity Recognition (NER)

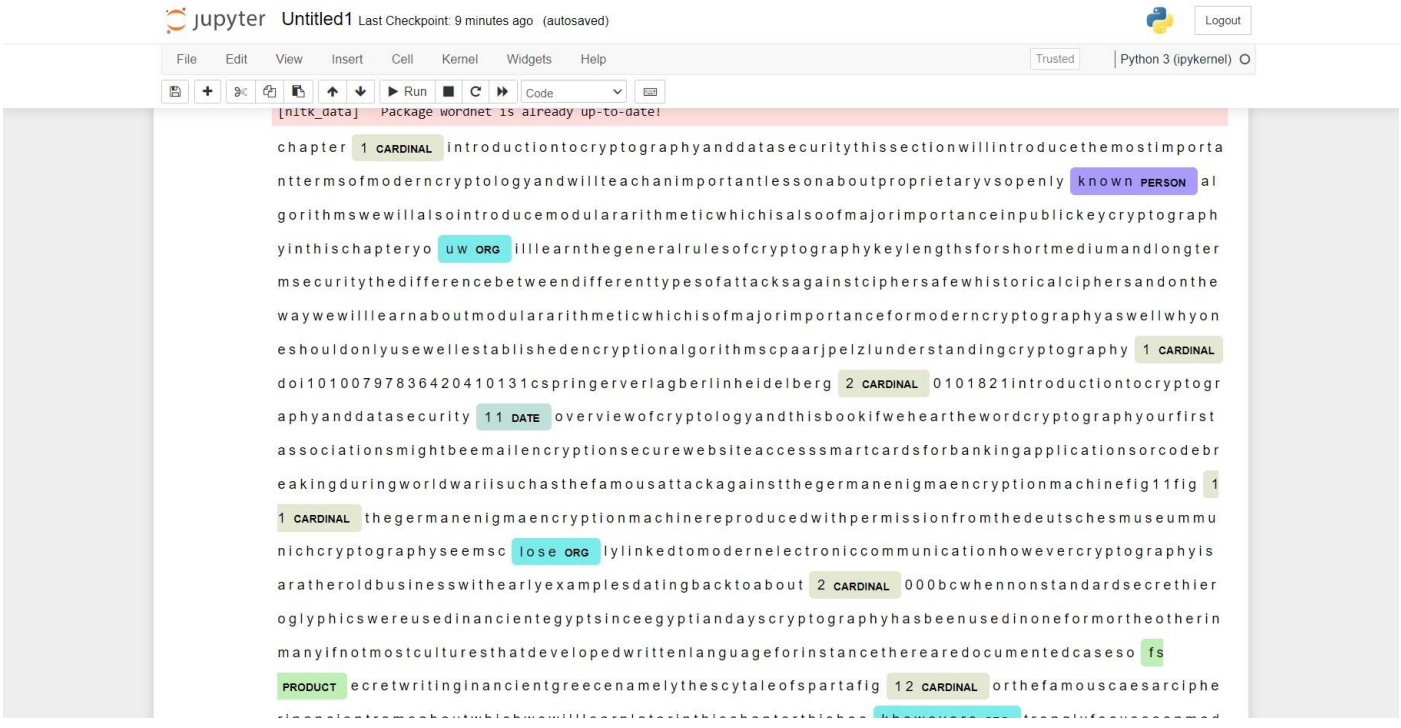
It is a sub part of Information extraction and retrieval. We have used the ‘spacy’ library of python for the same. Using ‘en_core_web_sm.load()’ inbuilt function we have named or labelled all the text of a novel. Spacy supports PERSON, NORP (nationalities, religious and political groups), FAC (buildings, 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, ORDINAL and CARDINAL entities.

```
237
238  # Second Part
239  # Named Entity Recognition (NER)
240  # import spacy , displacy and en_core_web_sm for the same
241  import spacy
242  from spacy import displacy
243  import en_core_web_sm
244  from collections import Counter
245  NER = en_core_web_sm.load()
246
247  doc1 = NER(T1)
248  print([(X.text, X.label_) for X in doc1.ents])
249
250
```

```
[('chapter 1', 'LAW'), ('berlin', 'GPE'), ('2010', 'DATE'), ('11', 'CARDINAL'), ('first', 'ORDINAL'), ('world war ii', 'EVENT'), ('german', 'NORP'), ('11', 'CARDINAL'),
('11', 'CARDINAL'), ('german', 'NORP'), ('about 2000', 'CARDINAL'), ('egypt', 'GPE'), ('egyptian days', 'DATE'), ('one', 'CARDINAL'), ('doc', 'ORG'), ('12', 'CARDINAL'),
('rome', 'GPE'), ('12', 'CARDINAL'), ('13', 'CARDINAL'), ('first', 'ORDINAL'), ('19 11', 'QUANTITY'), ('3', 'MONEY'), ('13', 'CARDINAL'), ('two', 'CARDINAL'), ('13',
'CARDINAL'), ('several decades', 'DATE'), ('13', 'CARDINAL'), ('three', 'CARDINAL'), ('two', 'CARDINAL'), ('1976', 'DATE'), ('1976', 'DATE'), ('whitfield diffie martin
hellman', 'PERSON'), ('ralph merkle', 'PERSON'), ('20 4                                                    ', 'PERCENT'), ('11', 'CARDINAL'), ('third',
'ORDINAL'), ('12', 'CARDINAL'), ('121', 'CARDINAL'), ('two', 'CARDINAL'), ('14', 'CARDINAL'), ('lan communica', 'PERSON'), ('awifi communica', 'PERSON'), ('two',
'CARDINAL'), ('the next few years', 'DATE'), ('1', 'CARDINAL'), ('21 12', 'CARDINAL'), ('14', 'CARDINAL'), ('15', 'CARDINAL'), ('15', 'CARDINAL'), ('15', 'CARDINAL'),
('15', 'CARDINAL'), ('22', 'CARDINAL'), ('13', 'CARDINAL'), ('1', 'CARDINAL'), ('2', 'CARDINAL'), ('122', 'CARDINAL'), ('one', 'CARDINAL'), ('one', 'CARDINAL'), ('23 12',
'CARDINAL'), ('12', 'CARDINAL'), ('bnnb                                    hcc hwwhbsqvqbre hwq vhlq', 'ORG'), ('first', 'ORDINAL'), ('ci', 'GPE'), ('first', 'ORDINAL'),
('121', 'CARDINAL'), ('kk', 'GPE'), ('keys ki', 'PERSON'), ('52', 'CARDINAL'), ('several decades', 'DATE'), ('24', 'CARDINAL'), ('first', 'ORDINAL'), ('one', 'CARDINAL'),
('26', 'CARDINAL'), ('al phabet', 'ORG'), ('25', 'CARDINAL'), ('26', 'CARDINAL'), ('25321', 'DATE'), ('26', 'CARDINAL'), ('hundreds of thousands', 'CARDINAL'), ('several
decades', 'DATE'), ('second', 'ORDINAL'), ('first', 'ORDINAL'), ('second', 'ORDINAL'), ('english', 'LANGUAGE'), ('1', 'CARDINAL'), ('english', 'LANGUAGE'), ('about 13',
'CARDINAL'), ('second', 'ORDINAL'), ('about 9', 'CARDINAL'), ('third', 'ORDINAL'), ('about 8', 'CARDINAL'), ('11', 'CARDINAL'), ('2', 'CARDINAL'), ('english', 'LANGUAGE')
('european', 'NORP'), ('3', 'CARDINAL'), ('one', 'CARDINAL'), ('three', 'CARDINAL'), ('three', 'CARDINAL'), ('13', 'CARDINAL'), ('12', 'CARDINAL'), ('noon', 'TIME'), ('25
13', 'CARDINAL'), ('9', 'CARDINAL'), ('11', 'CARDINAL'), ('english', 'LANGUAGE'), ('00817', 'DATE'), ('00150', 'DATE'), ('00751', 'DATE'), ('01270', 'DATE'), ('00633',
'DATE'), ('00906', 'DATE'), ('00276', 'DATE'), ('00077', 'CARDINAL'), ('00197', 'DATE'), ('00241', 'CARDINAL'), ('13', 'CARDINAL'), ('131', 'CARDINAL'), ('british',
'NORP'), ('world war ii', 'EVENT'), ('german', 'NORP'), ('alan turing', 'PERSON'), ('16 26 10', 'MONEY'), ('16', 'CARDINAL'), ('channels2', 'GPE'), ('2', 'CARDINAL'),
('german', 'NORP'), ('200', 'CARDINAL'), ('27 13', 'CARDINAL'), ('1883', 'DATE'), ('131', 'CARDINAL'), ('132', 'CARDINAL'), ('the 1990s', 'DATE'), ('two', 'CARDINAL'),
('1', 'CARDINAL'), ('122', 'CARDINAL'), ('28', 'CARDINAL'), ('2', 'CARDINAL'), ('80bit', 'CARDINAL'), ('1024bit', 'CARDINAL'), ('rsa rsa', 'PERSON'), ('12', 'CARDINAL'),
```

Extracting text for Manual Labelling for T1

```
250  # Extracting text for Manual Labelling for T1
251
252  para_T1 = T1[0:5000]
253  doc1 = NER(para_T1)
254  displacy.render(doc1, jupyter=True, style='ent')
255
```



```
255
256  comp_labels_1 = [x.label_ for x in doc1.ents]
257  print(comp_labels_1)
258
```

```
['CARDINAL', 'PERSON', 'ORG', 'CARDINAL', 'CARDINAL', 'DATE', 'CARDINAL', 'ORG', 'CARDINAL', 'PRODUCT', 'CARDINAL', 'ORG', 'QUANTITY', 'CARDINAL', 'DATE', 'PRODUCT']

In [32]:
```

Manual labelled and performance measures for T1

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.

```
258
259  # Manual labelled and performance measures for the text
260
261  from sklearn import metrics
262  manual_labels_1 =['ORDINAL','NULL','NULL','CARDINAL','PERSON','DATE','ORDINAL','NULL','CARDINAL','NULL','CARDI
263
264  print(metrics.classification_report(manual_labels_1,comp_labels_1))
265
266
```

For text T1 the performance metrics are:

```
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

    CARDINAL       0.57      1.00      0.73         4
        DATE       1.00      1.00      1.00         2
        NULL       0.00      0.00      0.00         5
     ORDINAL       0.00      0.00      0.00         2
         ORG       0.00      0.00      0.00         0
      PERSON       0.00      0.00      0.00         1
     PRODUCT       0.00      0.00      0.00         0
    QUANTITY       1.00      1.00      1.00         1
 WORK_OF_ART       0.00      0.00      0.00         1

    accuracy                           0.44        16
   macro avg       0.29      0.33      0.30        16
weighted avg       0.33      0.44      0.37        16
```

Accuracy is 44%.

Part-3

The

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:

-