NLP Project Round 1 Report by Team :

LaMDA

Adarsh Singh 20UCS008

Atishay Jain 20UCS042

Milind Sharma 20UCS119

Kuldeep 20UCS103

Date - 30/10/2022

GitHub Link : LaMDA

TABLE OF CONTENTS

1.0 Data Description :

For this Project we imported a book in text format in order to perform text analysis using NLP techniques. We tokenized and lemmatized the imported text file, analyzed the frequency distribution and performed PoS tagging on the text file. We have chosen the prescribed textbook for Operating System i.e. "**A. Silberschatz, P.B. Galvin and G. Gagne, Operating System Concepts, 8th ed. Wiley, 2009**".
For the tasks given in this project we have used the **nltk** module which is a suite of libraries and programs for symbolic and statistical natural language processing for English written in Python. And we have imported some modules and functions in order to perform different activities such as preprocessing, tokenizing, removing stop words etc. And after performing such operations we have plotted frequency distribution and created word clouds.

2.0 Data Preprocessing :

First, we need to import the text from the pdf file . In other words, convert the .pdf file to .txt file after removing all the figures,images and tables present in the book.

We first import the.pdf file for the book and then using PyPDF2 library in Python we get the .txt file using the .extract_text() function.

Link for the extracted text file and source code (Github) : - Link

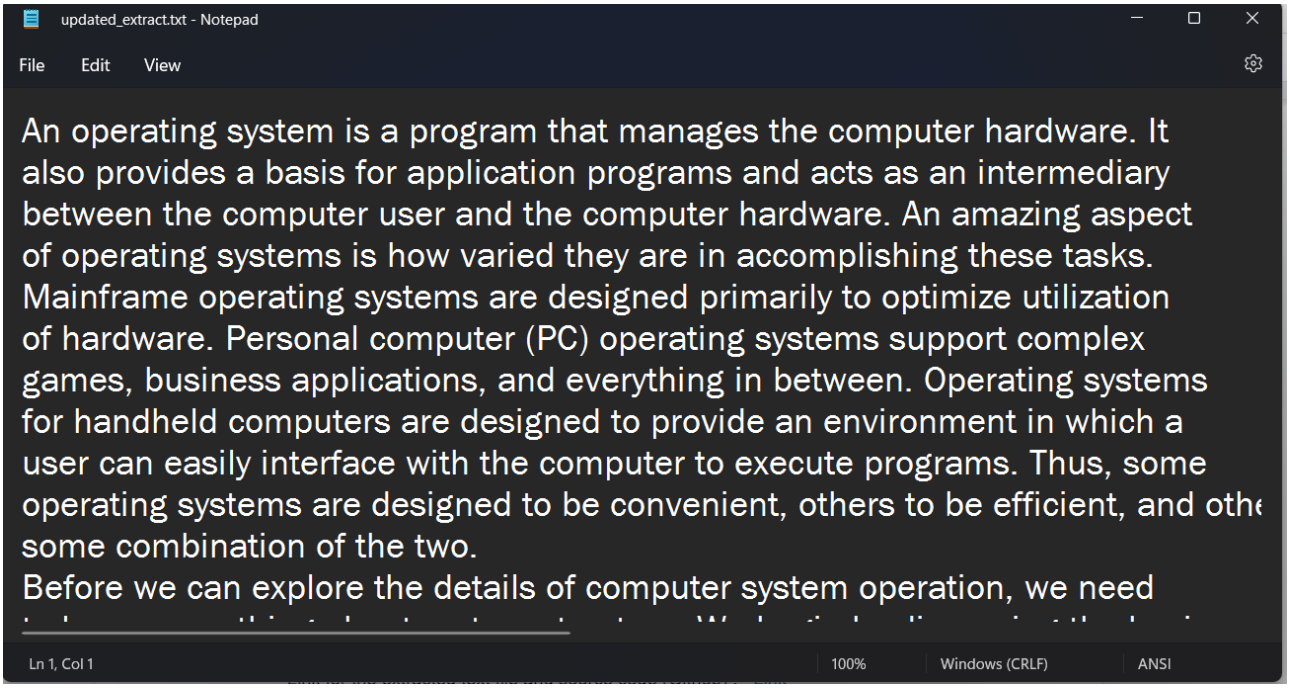


Fig 2.1 Screenshot of text file after extraction of text from .pdf file

3.0 Data Preparation :

Now, we convert the whole text to lowercase , remove all the special characters and remove any white spaces present in the text.

Then, we perform lemmatization on the text and extract the final processed.txt file.

Link for the extracted text file and source code (Github) : - Link

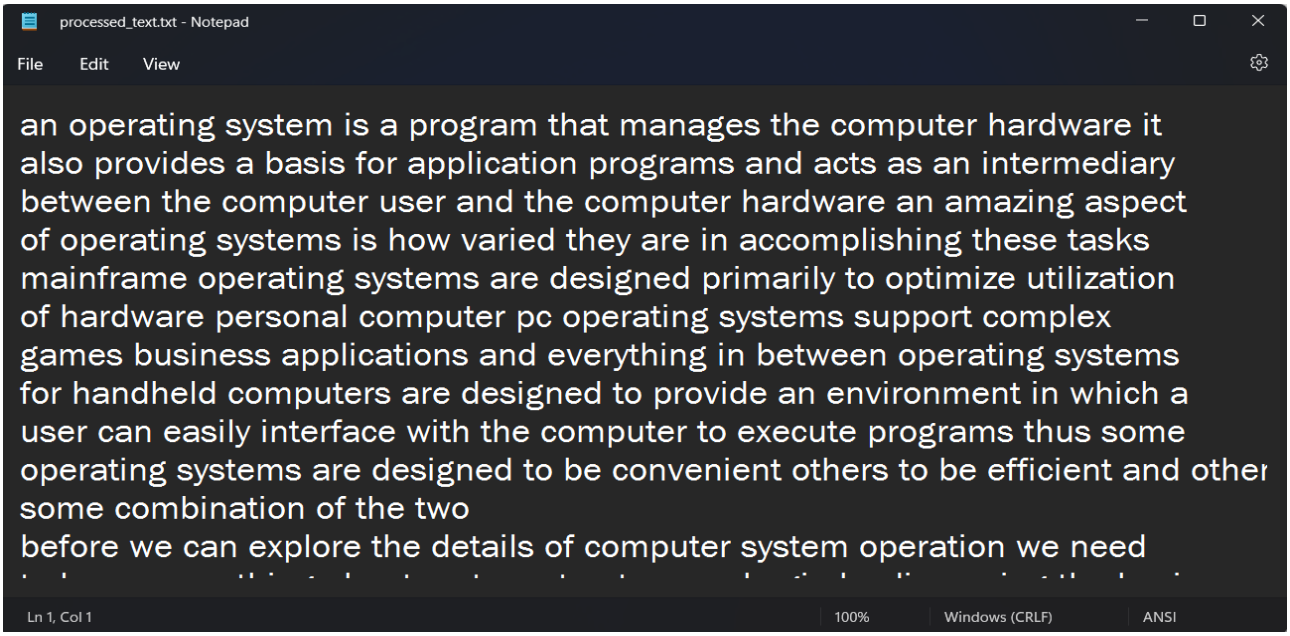


Fig 3.1 Screenshot of text file after above operations

After this , we tokenize the text file using word_tokenize() function imported from nltk.

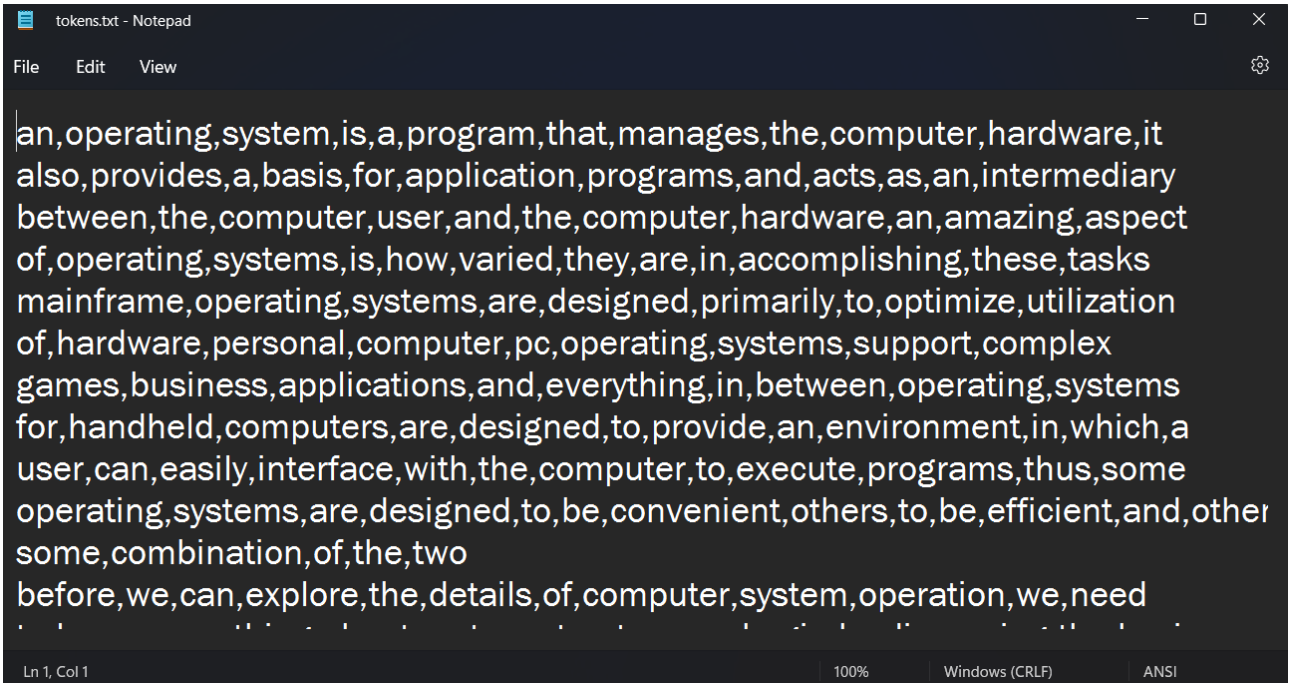


Fig 3.2 Screenshot of text file after tokenization

4.0 Problem Statement :

In this project our main objective is to perform frequency analysis of tokens obtained in the above sections.
We need to :

- Create Words Clouds , before and after removing stop words
- Evaluate the relationship between word length and frequency
- Perform POS tagging for the text

5.0 Frequency Distribution :

5.1 Frequencies of tokens :

```
Out[51]: [('the', 25301),
          ('a', 12227),
          ('to', 10126),
          ('of', 9403),
          ('is', 7545),
          ('and', 7544),
          ('in', 6443),
          ('for', 4304),
          ('system', 4273),
          ('that', 4133),
          ('be', 3330),
          ('are', 2916),
          ('file', 2678),
          ('process', 2620),
          ('can', 2611),
          ('as', 2607),
          ('it', 2382),
          ('this', 2280),
          ('on', 2140),
          ('by', 2103)]
```

Fig 5.1 Screenshot representing 20 most frequent words with their frequency in the processed text

5.2 Plot:

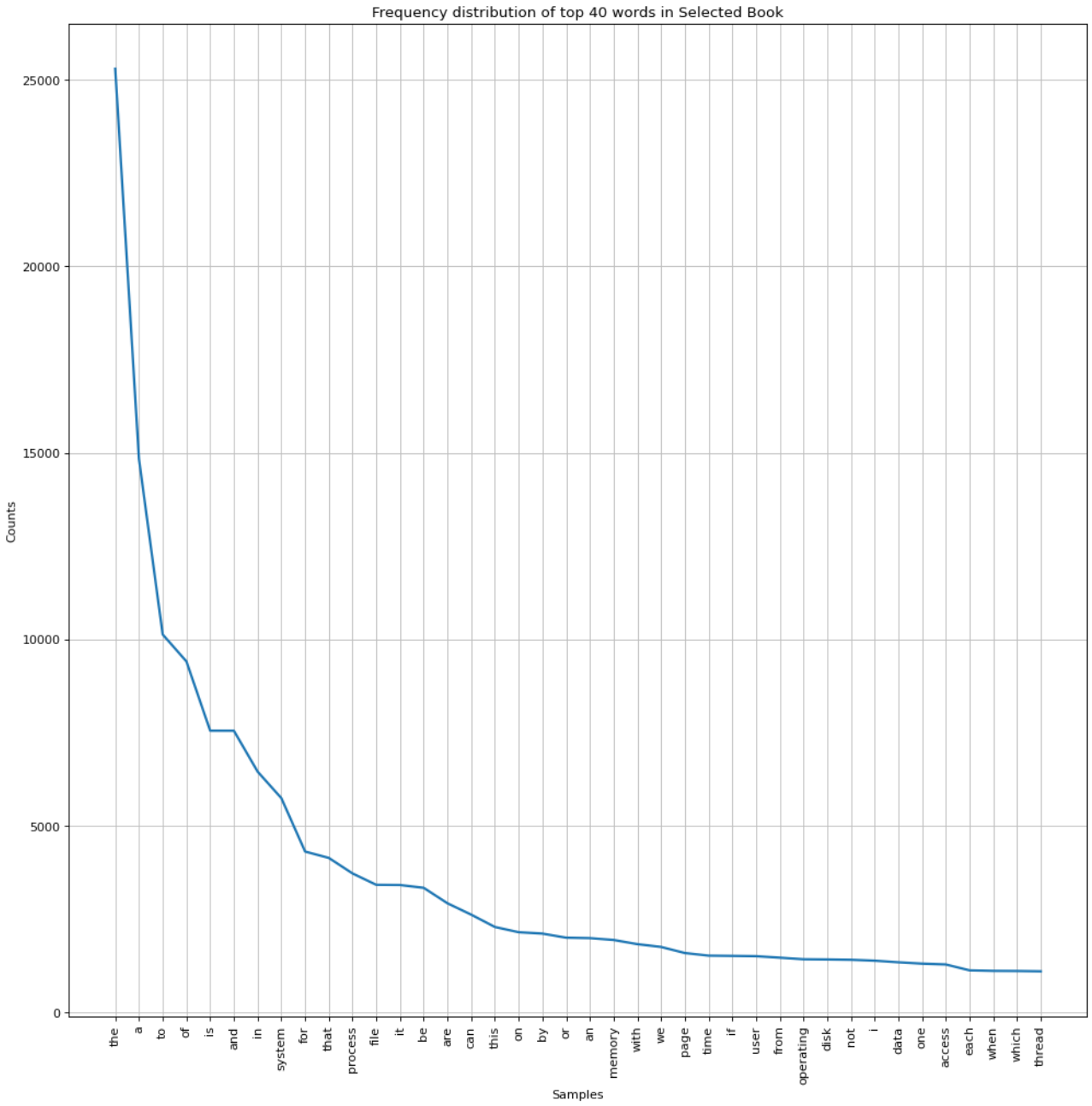


Fig 5.2 Plot representing the 20 most frequent words

5.3 Inference:

17 out of 20 most frequent words in the book are **stop words**. **3** words that are not stop words are **system ,file and process**. Thus, we can infer the type of content in the book with this information. We can say that the stop words make up a substantial fraction of all the words.

6.0 Word Cloud :

6.1 Word Cloud using all the tokens :

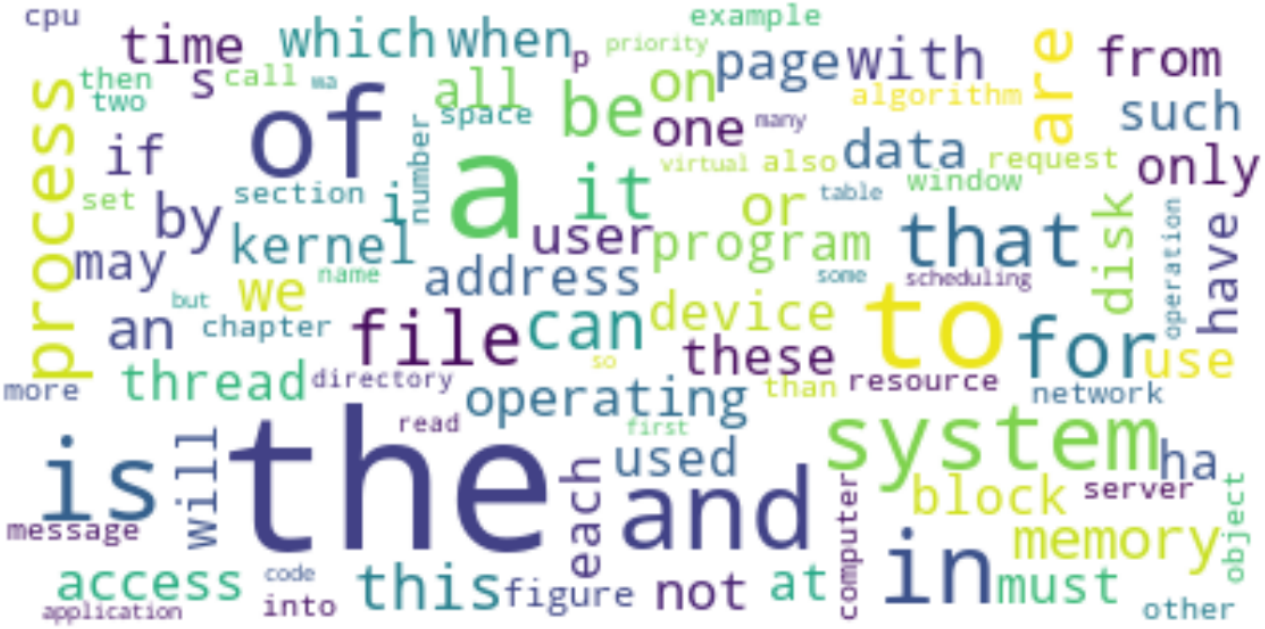


Fig 6.1 Word Cloud of all the tokens

6.2 Word Cloud after removing all stop words :

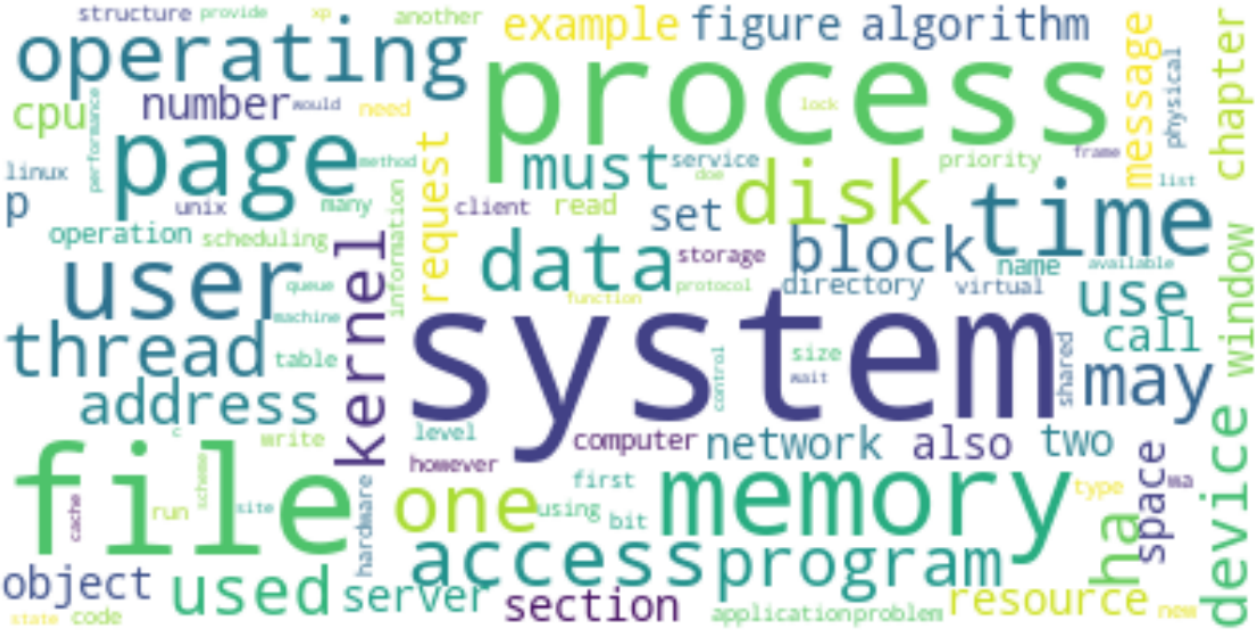


Fig 6.1 Word Cloud after removing stop words

6.3 Inference :

We observe that in fig 6.1 major space is occupied by the stop words since they are the most occurring words in the file.
But in fig 6.2 , major space is occupied by words like system , process, file, data , etc. Thus we can say that this book on Operating Systems mainly focuses on how the system manages different processes.Also, it talks about the file management system.

7.0 Relationship b/w word length and frequency :

```
OrderedDict([(1, 21136),
             (2, 59015),
             (3, 62075),
             (4, 59093),
             (5, 28390),
             (6, 37675),
             (7, 34318),
             (8, 20946),
             (9, 18595),
             (10, 8753),
             (11, 7250),
             (12, 2198),
             (13, 1558),
             (14, 832),
             (15, 251),
             (16, 70),
             (17, 22),
             (18, 16),
             (19, 9),
             (20, 12),
             (21, 2),
             (22, 3),
             (23, 2),
             (24, 3),
             (25, 2),
             (26, 2),
             (29, 1)])
```

Fig 7.1 Frequency based on word length with stop words

```
OrderedDict([(1, 3045),
             (2, 4593),
             (3, 14262),
             (4, 37142),
             (5, 23426),
             (6, 36916),
             (7, 33086),
             (8, 20945),
             (9, 18592),
             (10, 8724),
             (11, 7250),
             (12, 2198),
             (13, 1558),
             (14, 832),
             (15, 251),
             (16, 70),
             (17, 22),
             (18, 16),
             (19, 9),
             (20, 12),
             (21, 2),
             (22, 3),
             (23, 2),
             (24, 3),
             (25, 2),
             (26, 2),
             (29, 1)])
```

Fig 7.2 Frequency based on word length without stop words

7. 1 Plot (with stop words) :

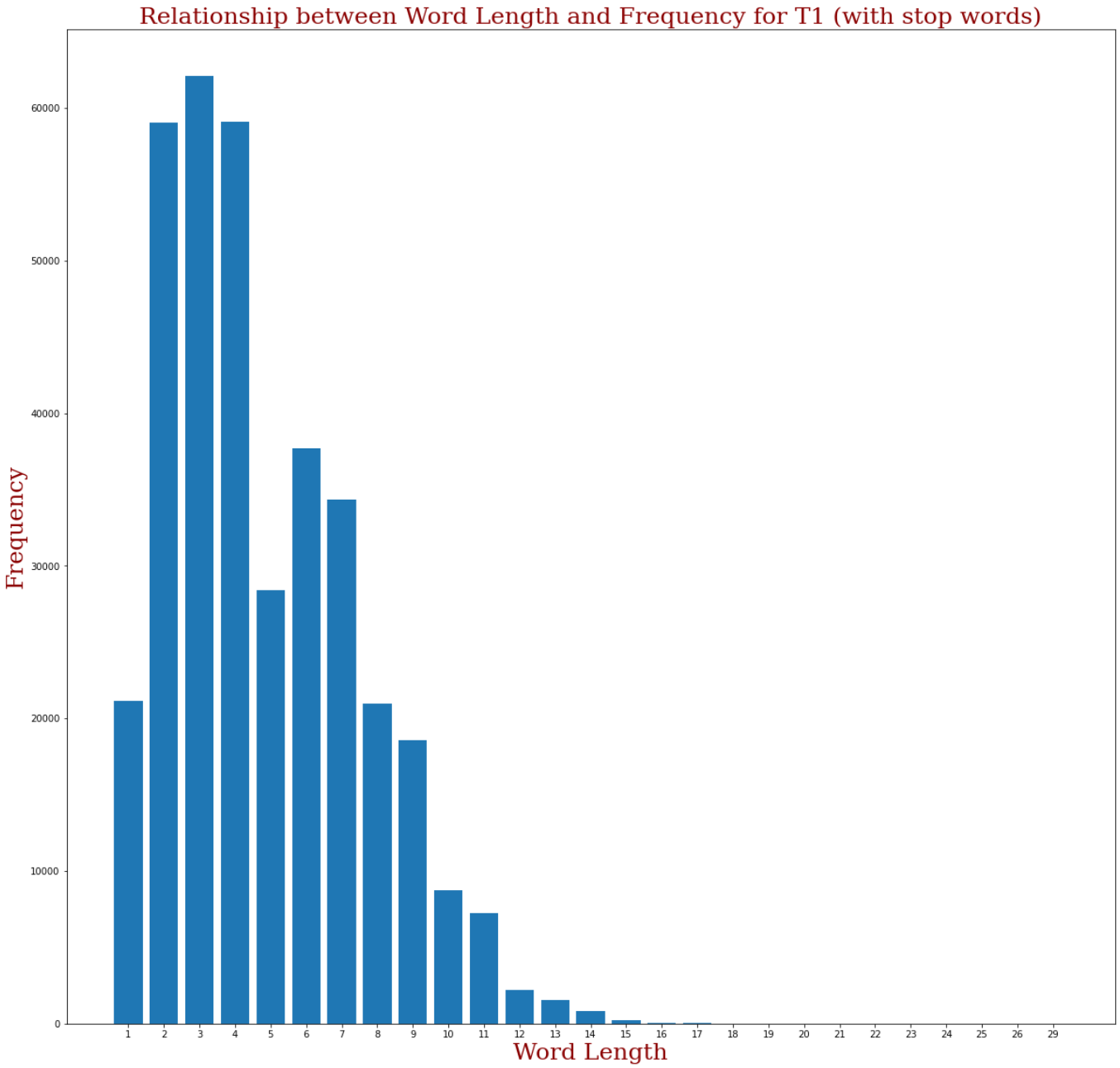


Fig 7.3 Plot of data in fig 7.1

7.2 Plot (without stop words):

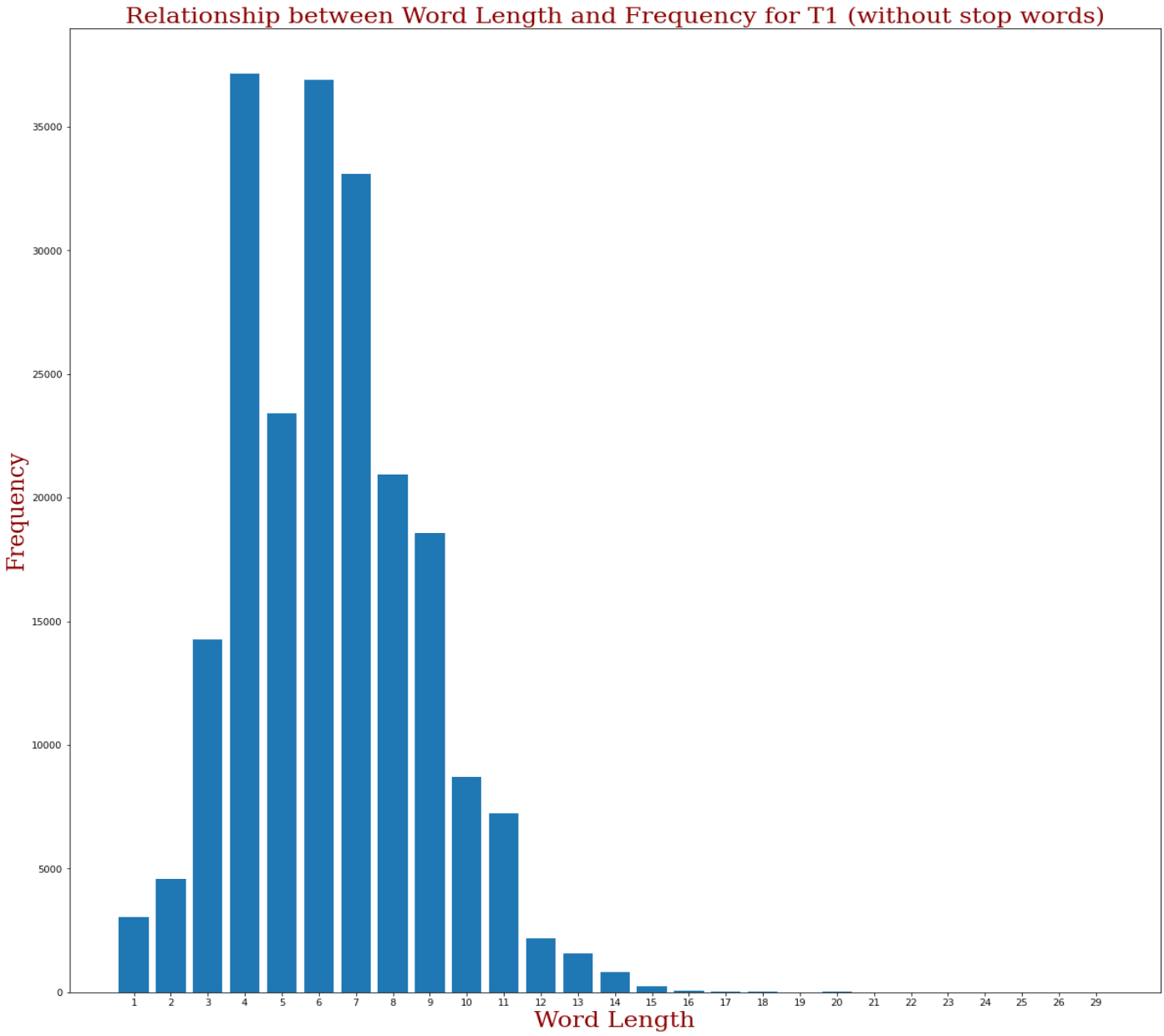


Fig 7.4 Plot of data in fig 7.2

7.3 Inference :

In fig 7.1, words of sizes 2,3 and 4 are most frequent. This is due to the fact that stop words occur very frequently in the text file.
After removing the stop words, we can see in fig 7.2 that words of length 4 and 6 have very high frequency. This may be due to the fact that words like system and file have high frequency in the text file.

8.0 POS Tagging :

We used nltk.pos-tag() method for PoS tagging using the Penn Treebank Tagset. We obtained the frequency distribution for the different tags for the texts. We also plotted this distribution as a bar chart.

```
OrderedDict([('NN', 92505), ('DT', 47648), ('IN', 41378), ('JJ', 32195), ('NNS', 25560), ('VBZ', 16467), ('VB', 14953), ('VBN', 12
796), ('RB', 12719), ('CC', 10344), ('TO', 10126), ('VBP', 9397), ('VBG', 8300), ('MD', 6461), ('PRP', 5242), ('WDT', 2900), ('VB
D', 2810), ('CD', 2502), ('WRB', 1906), ('PRP$', 1551), ('JJR', 1152), ('RBR', 556), ('JJS', 554), ('RP', 474), ('WP', 388), ('PD
T', 346), ('EX', 306), ('NNP', 290), ('RBS', 234), ('FW', 115), ('WP$', 46), ('SYM', 5), ("''", 2), ('$', 1)])
```

Fig 8.1 Figure showing frequency of different tags

8.1 Plot :

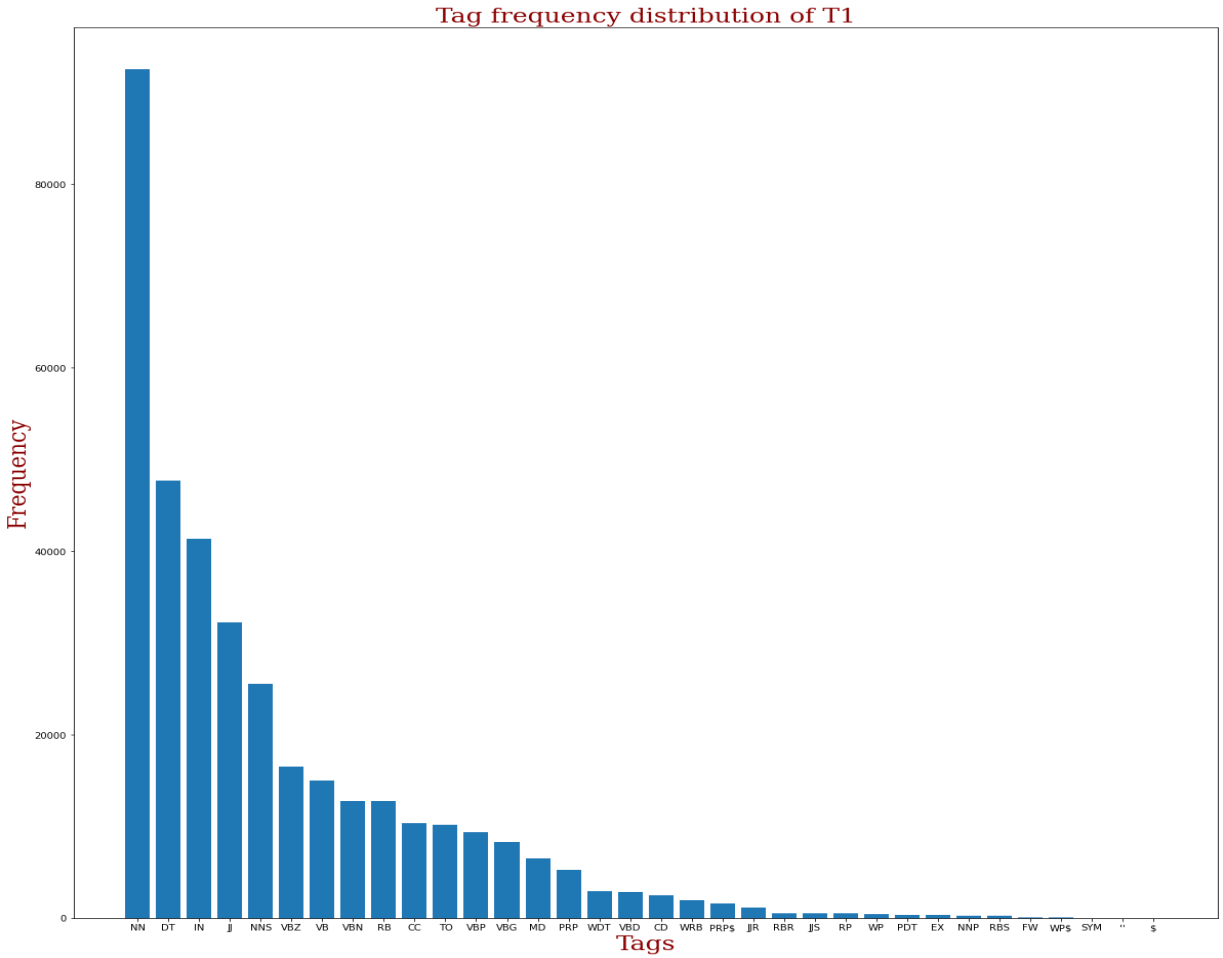


Fig 8.2 Plot representing above data

8.2 Inference :

We find that most frequent tags are :

- NN : Nouns (92505)
- DT : Determiners (47648)
- IN : Propositions (41378)

Thus, we get an idea about the type of content written in the book

9.0 References :

[1] Stackoverflow - How to extract text from pdf file

[2] Stackoverflow - How to tokenize text input in Python

[3] Youtube - Stats Wire

[4] GeeksForGeeks - How to remove stop words using nltk

[5] Medium.com - Text preprocessing in Python

NLP Project Round 2 Report by Team :

LaMDA

Adarsh Singh 20UCS008

Atishay Jain 20UCS042

Milind Sharma 20UCS119

Kuldeep 20UCS103

Date - 20/12/2022

Kaggle NB Link : LaMDA

TABLE OF CONTENTS

1.0 Python Libraries Used :

- **Matplotlib :**

  Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK

- **NLTK** :

  The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

  - **Word_tokenize :**

    Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string:

  - **WordNetLemmatizer :**

    Lemmatize `word` using WordNet's built-in morphy function.

    Returns the input word unchanged if it cannot be found in WordNet

  - **FreqDist** :

    A frequency distribution for the outcomes of an experiment. A frequency distribution records the number of times each outcome of an experiment has occurred.

  - **Wordnet :**

    WordNet is a lexical database for the English language, which was created by Princeton, and is part of the NLTK corpus

- **Spacy** :

  spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. The library is published under the MIT license and its main developers are Matthew Honnibal and Ines Montani, the founders of the software company Explosion.

2.0 Recognising Noun and Verbs :

- We plotted the frequency distribution of the nouns and verbs in the text under different categories as per WordNet.

- Categories for Verbs :

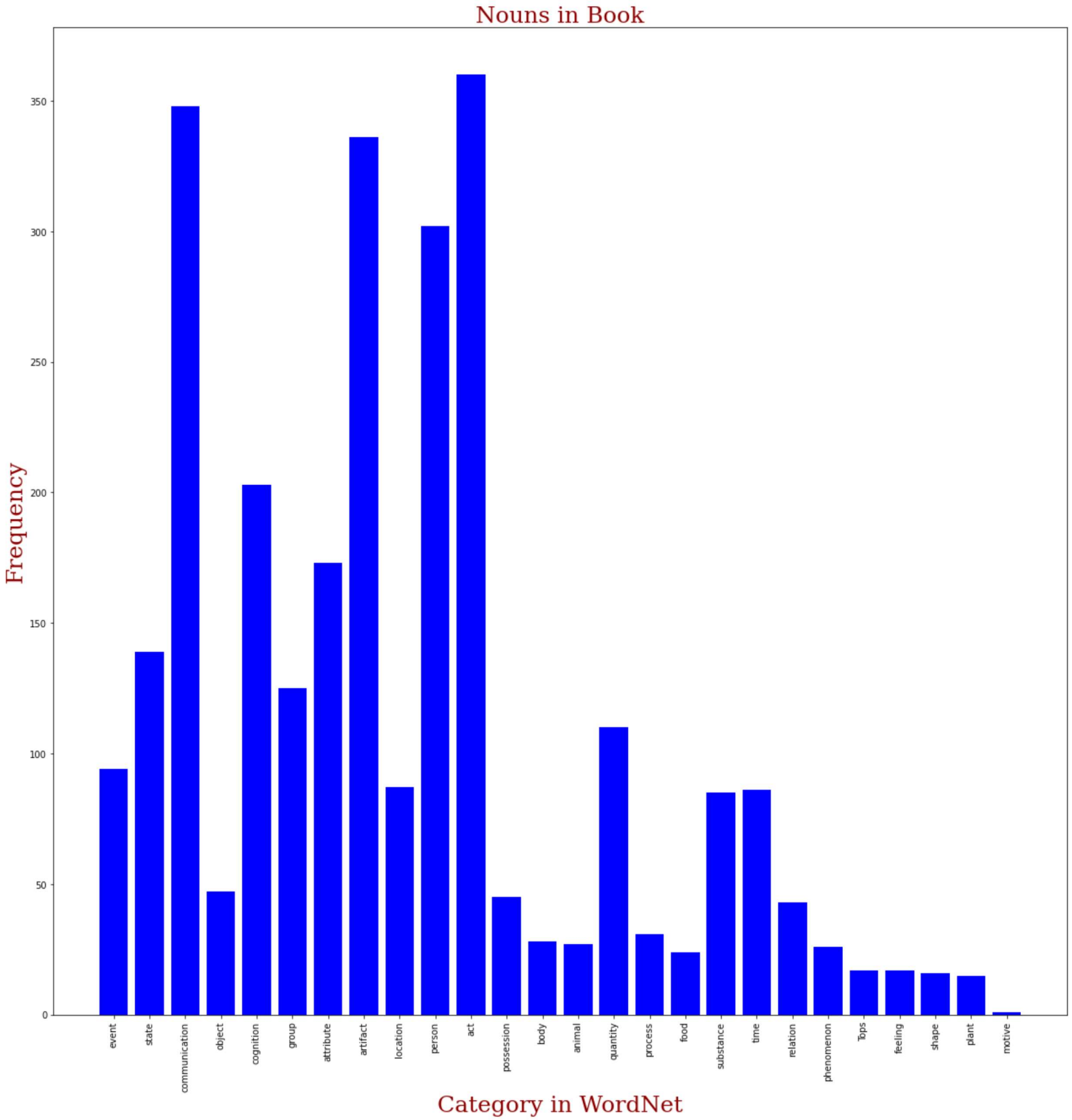
verb.body	verbs of grooming, dressing and bodily care
verb.change	verbs of size, temperature change, intensifying, etc
verb.cognition	verbs of thinking, judging, analyzing, doubting
verb.communication	verbs of telling, asking, ordering, singing
verb.competition	verbs of fighting, athletic activities
verb.consumption	verbs of eating and drinking
verb.contact	verbs of touching, hitting, tying, digging
verb.creation	verbs of sewing, baking, painting, performing
verb.emotion	verbs of feeling
verb.motion	verbs of walking, flying, swimming
verb.perception	verbs of seeing, hearing, feeling
verb.possession	verbs of buying, selling, owning
verb.social	verbs of political and social activities and events
verb.stative	verbs of being, having, spatial relations
verb.weather	verbs of raining, snowing, thawing, thundering

- Categories for Nouns :

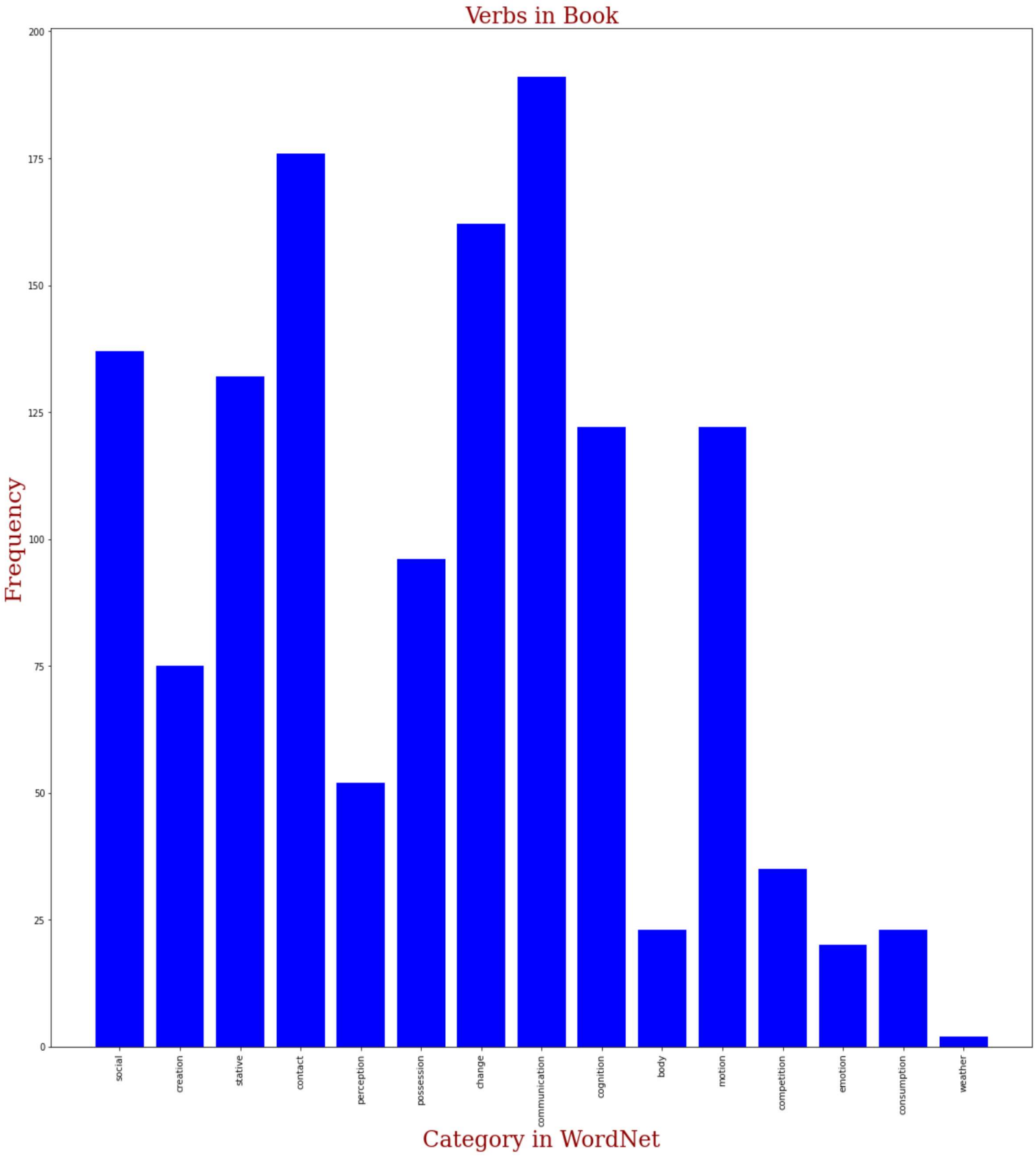
noun.Tops	unique beginner for nouns
noun.act	Nouns denoting acts or action
noun.animal	Nouns denoting animal
noun.artifact	nouns denoting man-made objects
noun.attribute	nouns denoting attributes of people and objects
noun.body	Nouns denoting body parts
noun.cognition	nouns denoting cognitive processes and contents
noun.communication	nouns denoting communicative processes and contents
noun.even	nouns denoting natural events
noun.feeling	nouns denoting feelings and emotions
noun.food	nouns denoting foods and drinks
noun.group	nouns denoting groupings of people or objects
noun.location	nouns denoting spatial position
noun.motive	Nouns denoting goals
noun.object	nouns denoting natural objects (not man-made)
noun.person	Nouns denoting people
noun.phenomenon	nouns denoting natural phenomena
noun.plant	Nouns denoting plants
noun.possession	nouns denoting possession and transfer of possession

noun.process	nouns denoting natural processes
noun.quantity	nouns denoting quantities and units of measure
noun.relation	nouns denoting relations between people or things or ideas
noun.shape	nouns denoting two and three dimensional shapes
noun.state	nouns denoting stable states of affairs
noun.substance	nouns denoting substances
noun.time	nouns denoting time and temporal relations

- When we plot the graph of the frequency distribution, where we plot Category in Wordnet on the x-axis and we plot frequency for the category on the y-axis, we find that some of the most frequent noun categories are: **act, communication, attract**.



- When we plot the graph of the frequency distribution, where we plot Category in Wordnet on the x-axis and we plot frequency for the category on the y-axis, we find that some of the most frequent verb categories are: **communication, contact, change**..



3.0 Recognising Entities

● To identify all Person, Location, and organizations in the book we have used Named Entity Recognition (NER) in python. We have Python Library **Spacy** for this.

● We pre-processed the text as above for this purpose, that is, we only reduced the text to the content of the book, expanded contractions, and removed chapter numbers from the text of the book.

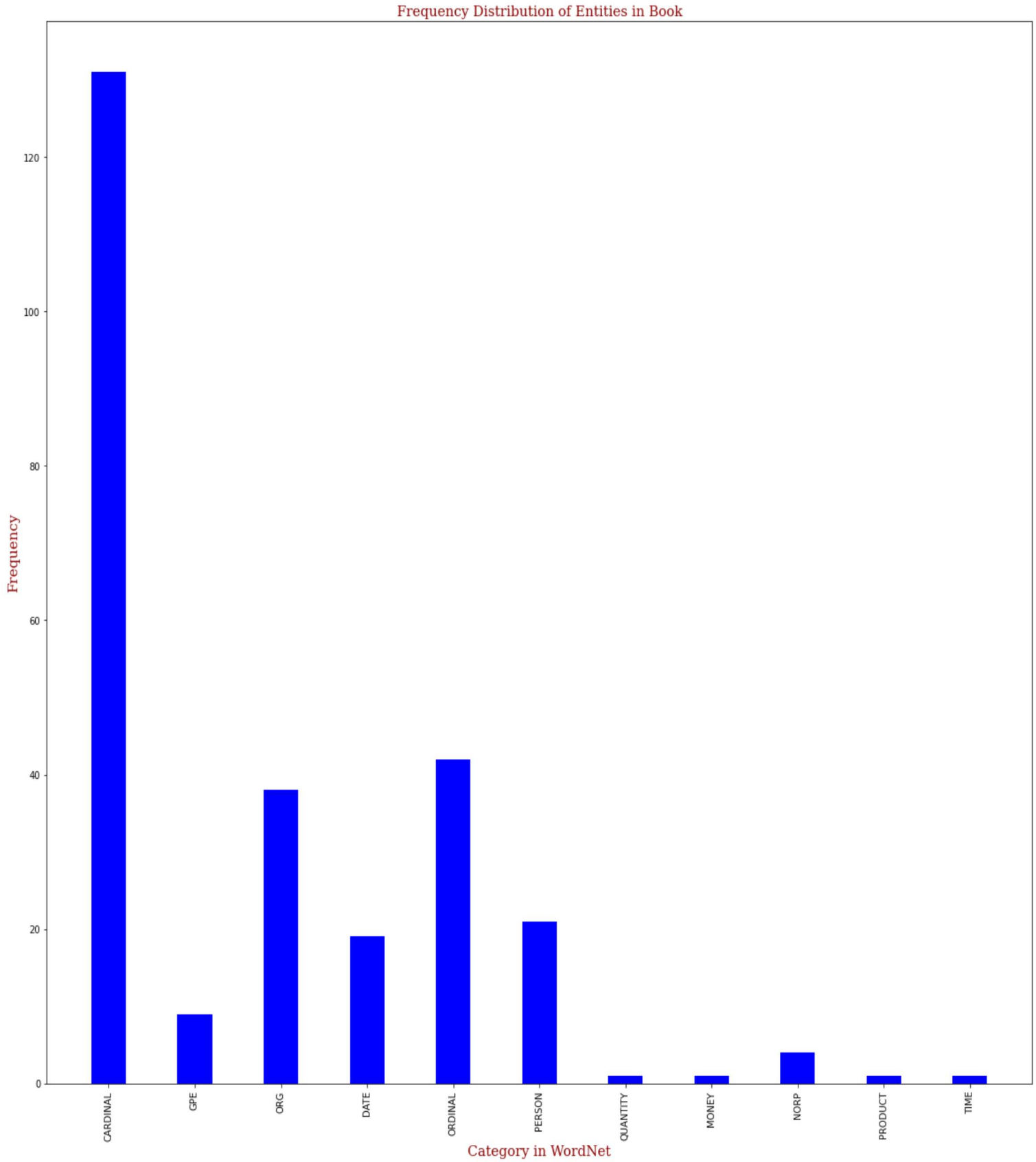
● The bar graph plotted using the FreqDist() of the above list was then used to indicate the number of different types of entities in the text.

● The named entity types reported are :

- CARDINAL
- ORGANIZATION
- PERSON
- LOCATION
- GPE(Geo-Political Entity)
- FACILITY
- DATE
- ORDINAL
- QUANTITY
- MONEY
- PRODUCT
- TIME.

● The bar graphs are attached:

- When we plot the graph of the frequency distribution of the entities in the book, where we plot Entity type on the x-axis and we plot frequency on the y- axis, we find that :
    - Most frequent entity type : Cardinal
    - Least frequent is of type : Quantity,Money,Product,Time.



3.1 Performance Evaluation

- We selected 3 random passages from the book and manually identified the entities present in them.
- We also extracted the entities from these passages using the Spacy library, as done above for the whole book.
- The passages can be found in the code.
- For example :

```
text2 = "in one approach the command interpreter itself contains the code to execute the command for example a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call in this case the number of comn lands that can be given determines the size of the command interpreter since each command requires its own implementing code an alternative approach used by unix among other operating systems implements most commands through system programs in this case the command interpreter does not understand the cmnmand in any way it merely uses the command to identify a file to be loaded into memory and executed thus the unix command to delete a file rm file txt would search for a file called rm load the file into memory and execute it with the parameter file txt the function associated with the rm command would be defined completely by the code in the file rm in this way programmers can add new commands to the system easily by creating new files with the proper figure the bourne shell command interpreter in solaris i names the command interpreter program which can be small does not have to be changed for new commands to be added graphical user interfaces a second strategy for interfacing with the operating system is through a user friendly graphical user interface or cui here rather than entering commands directly via a command line interface users employ a mouse based window and nl enu system characterized by a metaphor the user moves the mouse to position its pointer on images or on the screen the desktop that represent programs files directories and system functions depending on the mouse pointer s location clicking a button on the mouse can invoke a program select a file or directory known as a folder or pull down a menu that contains commands graphical user interfaces first appeared due in part to research taking place in the early s at xerox parc research facility the first ui appeared on the xerox alto computer in however graphical interfaces became more widespread with the advent of apple macintosh computers in the s the user interface for the macintosh operating system mac os has undergone various changes over the years the most significant being the adoption of the aqua interface that appeared with mac os x microsoft s first version of windows version was based on the addition of a ui interface to the ms dos operating system later versions of windows have made cosmetic changes in the appearance of the ui along with several enhancements in its functionality including windows explorer chapter traditionally unix systems have been dominated by command line inter."
```

MANUAL :
CARDINAL: one
ORDINAL: first, second
ORGANIZATION: xerox, macintosh, mac, windows, Microsoft, Apple

CODE :
CARDINAL: one
ORDINAL: first, second
ORGANIZATION: xerox, mac
DATE: years

	Predicted Positive	**Predicted Negative**
Actual Positive	6	4
Actual Negative	1	0

Precision: 0.857
Recall: 0.60
F1 Score: 0.705

Passage Number	**Precision**	**Recall**	**F1 Score**
1	0.64	0.70	0.67
2	0.857	0.60	0.705
3	0.75	0.43	0.55

- Average Value of above measures:
  - Precision: 0.749
  - Recall: 0.576
  - F1 Score: 0.641

- We can see that the spacy entity recognition is not very accurate, however, its performance is quite decent.

4.0 Third Part

With named entity recognition, the most important step is to identify the features that will be useful for relation classification, these feature are classified as follows: -

Features of named entity themselves : -

1) Named entity types of the two words.
2) Concatenation of two entity types.
3) Head words of the arguments.
4) Bag of words.

Features derived from the words in the text : -

1) Bag of words between two entities
2) Stemmed version of the entities
3) Distance in words between the arguments
4) Number of entities between the arguments.

Features derived from the syntactic structure of the sentence : -

1) Constituent tree paths.
2) Tree distance between the arguments.
3) Base syntactic chunk path.

To extract relationship between entities, we took a sample text and extracted the bag of words

```
text = """in one approach the command interpreter itself contains the code to
execute the command for example a command to delete a file may cause
the command interpreter to jump to a section of its code that sets up the
parameters and makes the appropriate system call in this case the number of
common lands that can be given determines the size of the command interpreter
since each command requires its own implementing code
an alternative approach used by unix among other operating systems
implements most commands through system programs in this case the
command interpreter does not understand the command in any way it merely
uses the command to identify a file to be loaded into memory and executed
thus the unix command to delete a file
rm file txt
would search for a file called rm load the file into memory and execute it with
the parameter file txt the function associated with the rm command would
be defined completely by the code in the file rm in this way programmers can
add new commands to the system easily by creating new files with the proper
figure the bourne shell command interpreter in solaris i
names the command interpreter program which can be small does not have
to be changed for new commands to be added."""
```

```
def find_bag_of_words(text, entity1, entity2):
    # Tokenize the text into words
    words = nltk.word_tokenize(text)

    # Remove stop words and punctuation
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words and word.isalpha()]

    # Find the indices of the two entities
    entity1_index = words.index(entity1)
    entity2_index = words.index(entity2)

    # Extract the bag of words between the two entities
    bag_of_words = words[entity1_index + 1:entity2_index]

    return bag_of_words
```

We now obtain the following bag of words: -

bag of words:
{command, interpreter, code, execute, delete, file, system call, number of commands, unix, system programs, memory, execute, parameter, function, programmer, new commands, system, bourne shell, solaris}

Finding the constituent path : -

```
def find_constituent_path(text, entity1, entity2):
    # Parse the text into a tree structure
    tree = nltk.Tree.fromstring(text)

    # Find the two entities in the tree and extract their corresponding subtrees
    entity1_subtree = tree.subtrees(lambda t: t.label() == entity1)[0]
    entity2_subtree = tree.subtrees(lambda t: t.label() == entity2)[0]

    # Find the lowest common ancestor (LCA) of the two entities
    lca = tree.common_ancestor(entity1_subtree, entity2_subtree)

    # Extract the constituent path by traversing the tree from the LCA to each of the two entities
    path = []
    current_node = lca
    while current_node != entity1_subtree:
        path.append(current_node.label())
        current_node = current_node[0]
    path.append(entity1)
    current_node = lca
    while current_node != entity2_subtree:
        path.append(current_node.label())
        current_node = current_node[-1]
    path.append(entity2)

    return path
```

We now find the syntactic structure of the sample text : -

```
def find_syntactic_structure(text, entity1, entity2):
    # Parse the text with the language model
    doc = nlp(text)

    # Initialize a list to store the syntactic structures
    structures = []

    # Iterate over the entities and the dependencies between them
    for entity in doc:
        if entity.text == entity1:
            for child in entity.children:
                if child.text == entity2:
                    structures.append(child.dep_)

    return structures
```

We now find the relation between two entities, “system” and “solaris” and summarize the result in the table as follows : -

Entity based features Entity 1 type Entity 1 head Entity 2 type Entity 2 head	 [OBJ] system [PROD] solaris
Word based features Between entity bag of words Word before entity 1 Word after entity 2	 {easily, by, creating, new, files, with, the, proper, figure, the, bourne, shell, command, interpreter, in} the *NONE*
Syntactic features Constituent Path Base syntactic chunk path	 NP ↑ NP ↑ S ↑ S ↓ NP NP → NP → PP → NP → VP → NP → NP

5.0 References :

[1] Stackoverflow - How to extract text from pdf file

[2] Stackoverflow - How to tokenize text input in Python

[3] Youtube - Stats Wire

[4] GeeksForGeeks - How to remove stop words using nltk

[5] Medium.com - Text preprocessing in Python

[6] Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition, Daniel Jurafsky and James H. Martin, Pearson, 2nd edition, 2014.