NLP Project Report

Under the Guidance of Dr. Sakthi Balan

Team Name: **LexicallyFlexible**

GitHub Link Repo: https://github.com/RishabhJ01/nlp-project-2021

Team Members:

Rishabh Jain (19UCS176)

Tapomay Singh Khatri (19UCS064)

Vishnu Bhat (19UCS058)

INDEX

S.No	Title	Page No.
1	System Requirements and Books Used	3
2	Import The Text	4
3	Perform Simple Text Pre-processing	4
4	Tokenize the Text	5
5	Analyze the Frequency Distribution	6
6	Creating Word Clouds	8
7	Evaluate the Relationship between the Word Length and Frequency after Removing Stop Words	12
8	Do PoS Tagging	15

System Requirements:

1. Python 3.x
2. Libraries: NLTK, matplotlib, wordcloud, re, NLTK.corpus, json, ssl, urllib
3. Jupyter Notebook
4. OS: Windows / Linux / Mac OS

Books Used:

Book 1:
Title: Pride and Prejudice
Author: Jane Austen
Link: https://www.gutenberg.org/files/1342/1342-0.txt

Book 2:
Title: Gulliver's Travel
Author: Jonathan Swift
Link: https://www.gutenberg.org/files/829/829-0.txt

Project Round-1

Problem statement - 1:

Import the text; let's call it data1 and data2.

We have imported Book-1 as data1 and Book-2 as data2. We have used **"*json*"** and **"*urllib*"** libraries to scrap the texts from the given links. Further we decoded it to convert the text to string.

Problem statement - 2:

Perform simple text pre-processing.

1. Changing the text to lowercase.
2. Removing running sections.
3. Removing special characters.
4. Removing Chapter Headlines.
5. Removing digits.
6. Removing implicit next line characters - '\n'.

Problem statement - 3:

Tokenize the texts.

Tokenization is basically dividing the text into smaller units called tokens. They can be words, characters, or subwords. It helps in understanding the context and interpreting the meaning behind the sequence of tokens. For example “miserable animals presume*”,* will be tokenized into - [“*miserable*”, “*animals*”, “*presume*”].

1. ‘*data1*’ and ‘*data2*’ are tokenized using the ***nltk.word_tokenize*** function.
2. **‘*token1*’** contains tokens of ‘*data1*’ i.e. Book-1.
3. ‘***token2*’** contains tokens of ‘*data2*’ i.e. Book-2.

Inference:

So now the imported texts **‘*data1*’** and **‘*data2*’** will be tokenized as explained above using ***nltk.word_tokenize*** function. Tokens created are now stored in ‘*token1*’ for ‘*data1*’ and ‘*token2*’ for ‘*data2*’.

Problem statement - 4:

Analyze the frequency distribution of tokens separately.

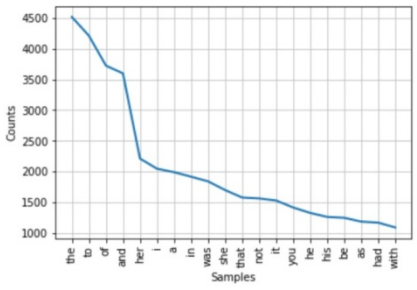
Formed the Frequency Distribution using FreqDist function of NLTK library and then plotted it.

Token1:

In [9]:
```
# Frequency Distribution for token 1
fdist1 = FreqDist(token1)
print(fdist1.most_common(20))
```

```
[('the', 4511), ('to', 4207), ('of', 3722), ('and', 3596), ('her', 2215), ('i', 2050), ('a', 1994), ('in', 1921), ('was', 1845), ('she', 1704), ('that', 1583), ('not', 1568), ('it', 1534), ('you', 1421), ('he', 1333), ('his', 1268), ('be', 1254), ('as', 1192), ('had', 1175), ('with', 1097)]
```

In [12]:
```
fdist1.plot(20)
```

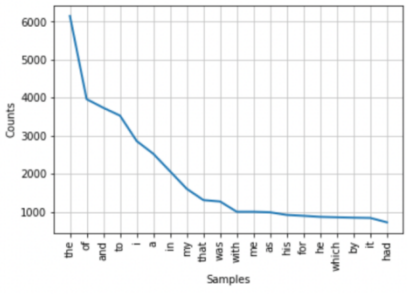


Token2:

In [10]:
```
# frequency Distribution for token 2
fdist2 = FreqDist(token2)
print(fdist2.most_common(20))
```

```
[('the', 6151), ('of', 3965), ('and', 3738), ('to', 3532), ('i', 2864), ('a', 2527), ('in', 2068), ('my', 1608), ('that', 1310), ('was', 1274), ('with', 1003), ('me', 1003), ('as', 989), ('his', 920), ('for', 897), ('he', 870), ('which', 858), ('by', 848), ('it', 841), ('had', 726)]
```

In [13]:
```
fdist2.plot(20)
```



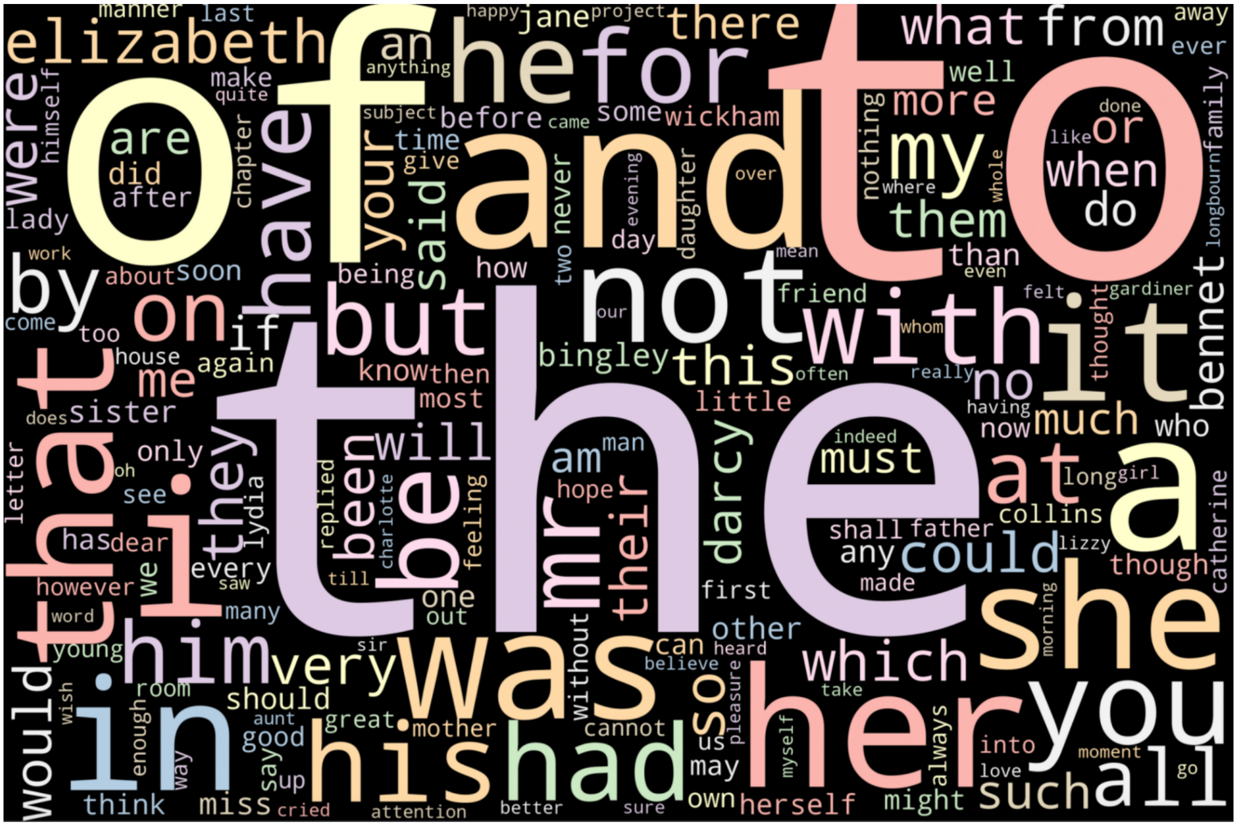
Problem statement - 5:

Create a wordcloud before and after removing the stop words.

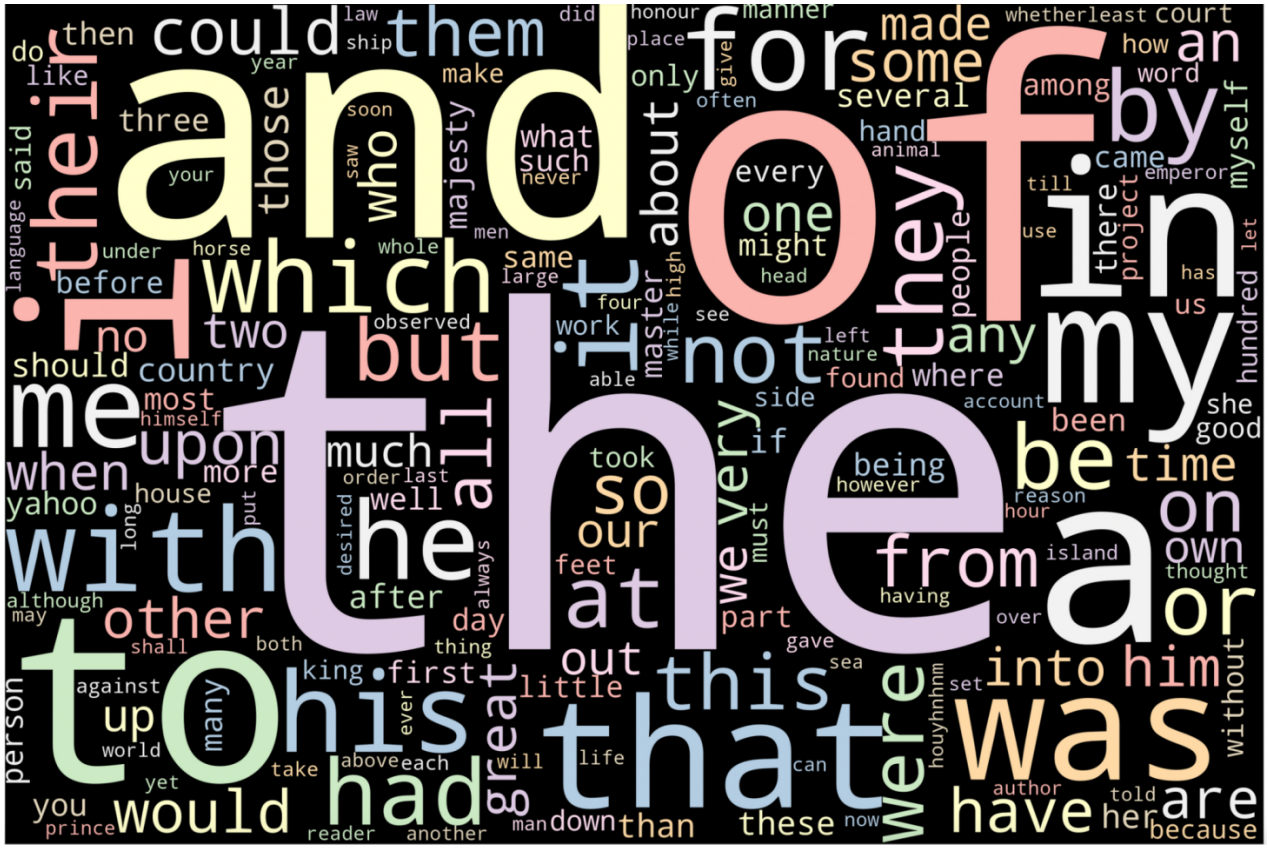
After analyzing the frequency distribution of the tokens, wordclouds are created using these tokens. A wordcloud is a visual representation of text data wherein each word is pictured with its frequency, i.e. higher the frequency, larger the size of the word. **Stopwords** - These are most commonly occurring words because of the grammar rules which hinder the visualization and useful computations.

Before removing stopwords:

Book-1:


the
to
of
and
he
for
her
she
was
his
had
in
i
that
be
it
a
not
but
with
you
mr
have
elizabeth
there
what
from
an
are
were
by
on
my
them
when
do
or
more
your
said
this
which
him
very
all
such
they
their
darcy
bingley
could
at
much
must
will
been
so
any
wickham
jane
well
away
ever
manner
last
happy
project
make
quite
anything
subject
before
some
came
time
give
nothing
more
did
after
lady
himself
chapter
never
evening
daughter
over
where
whole
like
done
family
longbourn
day
being
how
two
than
mean
even
friend
felt
gardiner
bennet
thought
whom
our
soon
about
work
come
too
if
house
again
know
then
this
often
really
no
having
me
most
little
indeed
now
who
sister
does
only
man
oh
letter
see
am
long
girl
catherine
hope
lydia
replied
charlotte
feeling
collins
shall
father
lizzy
has
dear
every
though
we
however
saw
till
first
made
one
out
word
many
morning
other
sir
young
heard
believe
would
room
should
without
take
wish
enough
aunt
great
mother
cannot
pleasure
myself
always
into
moment
way
good
say
us
may
love
go
think
miss
sup
cried
attention
better
sure
own
herself
might

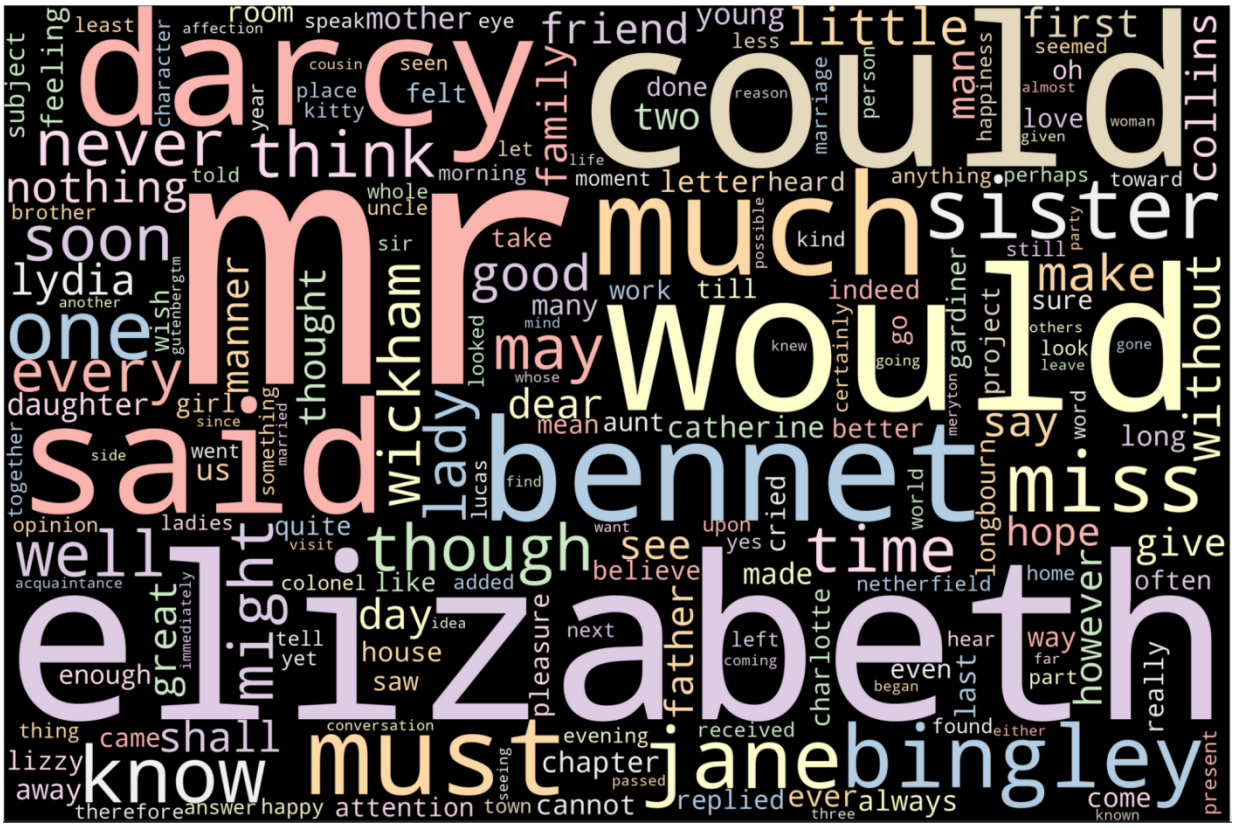
Book-2:



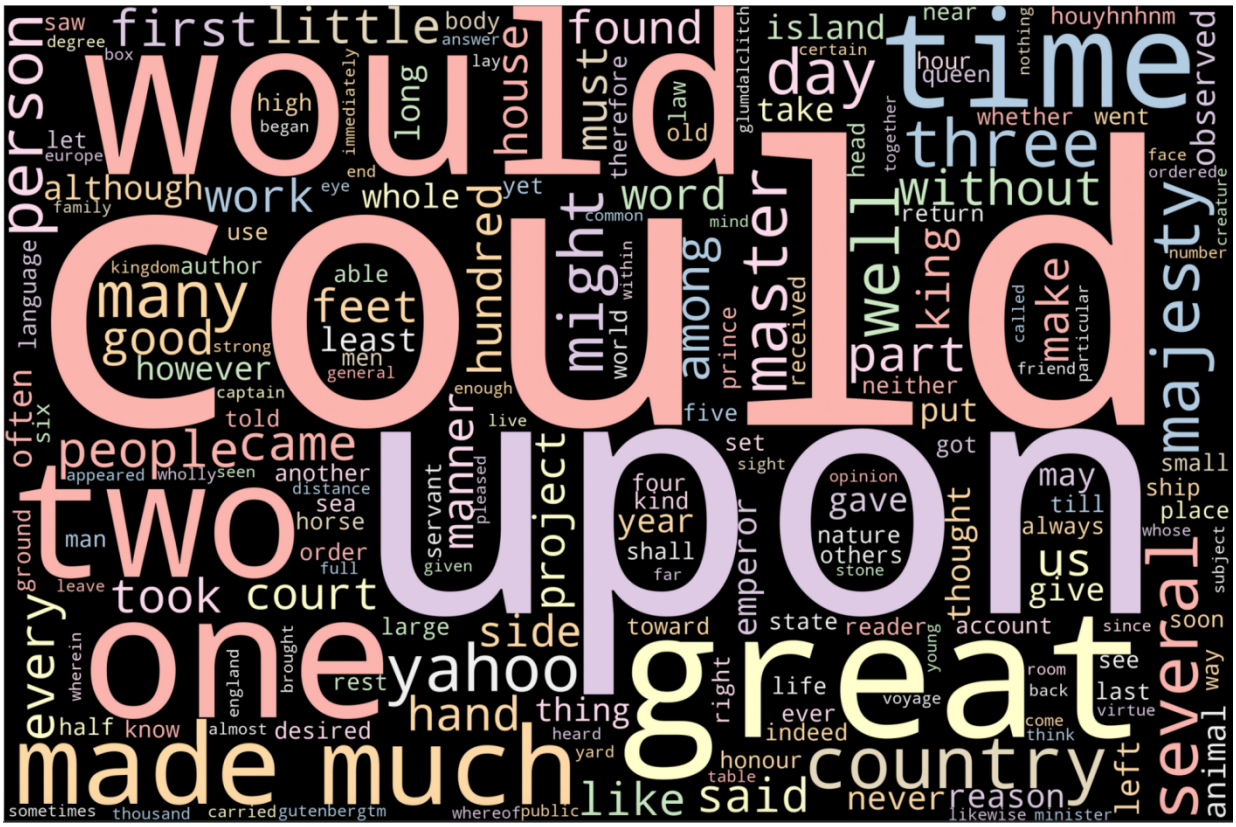
After removing stopwords:

NLTK library provides a list of the stopwords of the English language. So with the help of this list, we omitted out the stopwords from our tokens.

Book-1:



Book-2:



Inference:

As it can be clearly seen in above pictures of wordcloud before removing the stopwords, the larger words are *‘and’*, *‘of’*, *‘the’*, *‘to’*, etc. These are the stopwords which occur very frequently (size of a word in a word cloud is determined according to its frequency) and these words are not essential for our analysis. So after removing the stopwords, we can ensure that other words which are in context with our books show up in the word cloud.

Problem statement - 6:

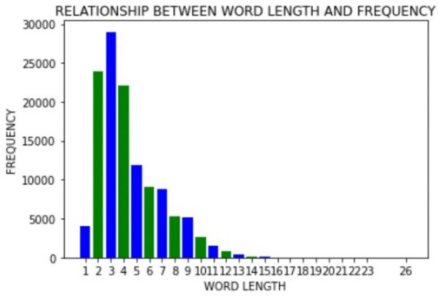
Evaluate the relationship between the word length and frequency after removing stopwords.

Firstly we created an empty list and stored the number of words for each length and then plotted it using Frequency Distribution and matplotlib.

Before Removing Stopwords:-

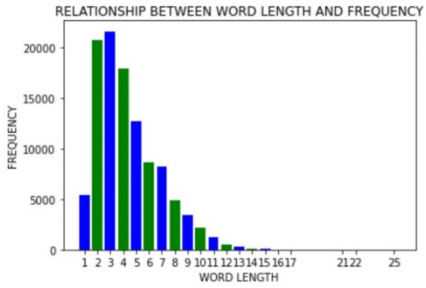
Book-1:

```
In [14]: plot_relationship(fdist1)
```



Book-2:

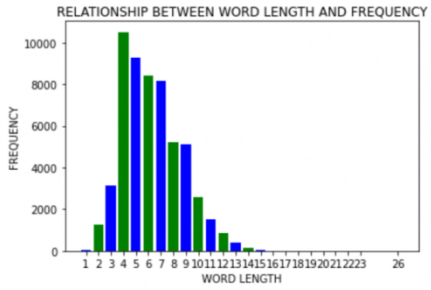
In [15]:
```
plot_relationship(fdist2)
```



After Removing Stopwords:-

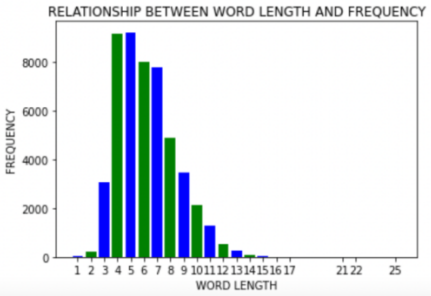
Book-1:

In [21]:
```
# For Data 1
plot_relationship(FreqDist(rem1))
```



Book-2:

In [22]:
```
# For Data 2
plot_relationship(FreqDist(rem2))
```



Inference:

Frequency of words of smaller length decreased significantly after removing the stopwords as generally these stopwords are of smaller length, e.g. - *'to', 'the',* etc.

Problem statement - 7:

Do PoS Tagging using any of the four tagset studied in the class and get the distribution of various tags.

PoS stands for Parts of Speech. PoS tagging means - to tag each word with its part of speech - like noun, verb, adverb, adjective, etc. Though there are some third party libraries which can be used to do PoS tagging in just one line of code, there is a very fancy algorithm which is doing all the work behind this. Basically it uses *HMM (Hidden Markov Model)* to tag each word. The crux of the algorithm is to find the most probable tagset for the given set of words among all the possible tagsets. It uses the *Viterbi* algorithm which is based on the dynamic programming paradigm to efficiently find the most probable tagset. There are many available tagsets which can be used for this purpose. For this project, we have used the *Treebank* tagset.*Tree bank* tagset includes *36* PoS tags like Coordinating Conjunction (CC), Determiner (DT) etc.

Note:- Below plots are for the 20 most occurring tags.

Book-1:-

PoS tagging analysis

```
In [30]: FreqDist(tags1)

Out[30]: FreqDist({'NN': 17754, 'JJ': 10157, 'RB': 5250, 'VBD': 4328, 'NNS': 4103, 'VB': 2420, 'VBP': 2234, 'VBG': 2176, 'VBN': 2147, 'MD': 1932,
...})
```

Book-2:-

PoS tagging analysis

```
In [31]: FreqDist(tags2)

Out[31]: FreqDist({'NN': 14486, 'JJ': 9178, 'NNS': 6175, 'VBD': 4104, 'RB': 3455, 'VBN': 1922, 'VBP': 1899, 'VBG': 1724, 'VB': 1596, 'IN': 1390,
...})
```

Project Round-2

Books Used:

Book 1:
Title: Pride and Prejudice Author:
Jane Austen
Link: https://www.gutenberg.org/files/1342/1342-0.txt

Book 2:
Title: Gulliver's Travel Author:
Jonathan Swift
Link: https://www.gutenberg.org/files/829/829-0.txt

Part - 1

Problem statement 1:-

Find the nouns and verbs in both the novels. Get the categories that these words fall under in the WordNet. Note that there are 25 categories and 16 categories for Nouns and Verbs respectively.

Get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same for each novel.

- We first imported the books, preprocessed them, tokenized them, removed the stop words and then applied the PoS tagging to them (detailed explanation of these steps can be found above).

- Using PoS Tags, we extracted the nouns and verbs.

- For the extraction of nouns, we ran a for loop over the list of PoS tagged tuples which we got after PoS tagging. If the tag starts with ‘N’, then the corresponding word was appended in the nouns list. Similarly if the tag starts with ‘V’, the word belongs to the verbs list. Had the tag been directly compared with the string “NN”, some of the words having tags such as “NNP” would have been left out from our nouns list. Similar is the case with verbs.

- Now we have the nouns and verbs list. For each word in this list, its category was extracted from the word net.

WordNet :- It is the lexical database i.e. dictionary for different languages, specifically designed for natural language processing. For each word in the list, the list of its synsets is extracted using the function *wordnet.synsets(word)*.

Synsets :- It is a set of synonyms which share a common meaning. Moreover it is a simple interface that is present in NLTK to look up words in WordNet. Some of the words have only one Synset and some have several.

Now using the 1st element of this list of synsets, the category was extracted using the function *lexname().*

Noun categories present in the wordnet :-

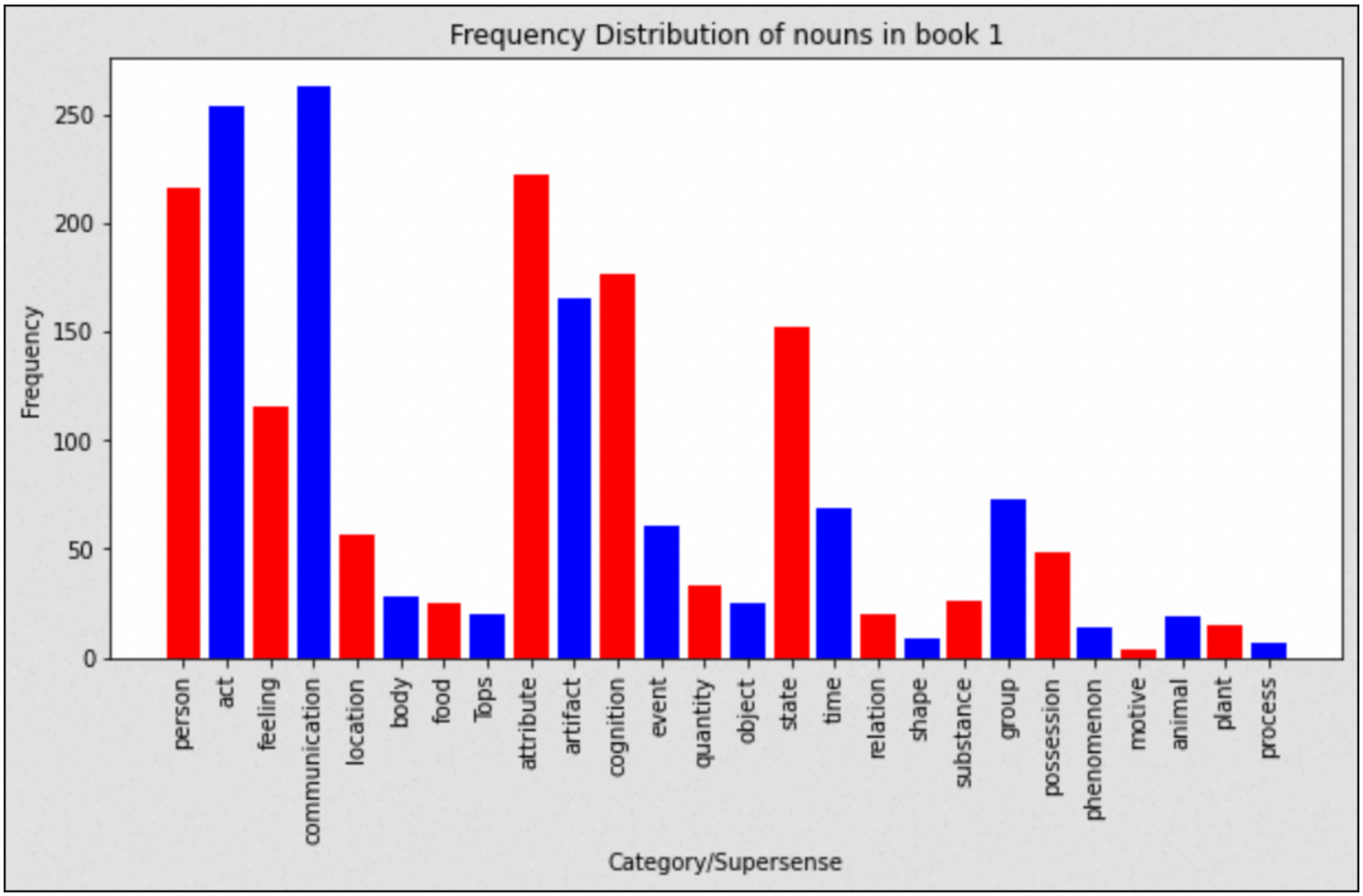
01	**tops**	unique beginner for nouns
02	**act**	nouns denoting acts or actions
03	**animal**	nouns denoting animals
04	**artifact**	nouns denoting man-made objects
05	**attribute**	nouns denoting attributes of people and objects
06	**body**	nouns denoting body parts
07	**cognition**	nouns denoting cognitive processes and contents
08	**communication**	nouns denoting communicative processes and contents
09	**event**	nouns denoting natural events

10	**feeling**	nouns denoting feelings and emotions
11	**food**	nouns denoting foods and drinks
12	**group**	nouns denoting groupings of people or objects
13	**location**	nouns denoting spatial position
14	**motive**	nouns denoting goals
15	**object**	nouns denoting natural objects (not man-made)
16	**person**	nouns denoting people
17	**phenomenon**	nouns denoting natural phenomena
18	**plant**	nouns denoting plants
19	**possession**	nouns denoting possession and transfer of possession
20	**process**	nouns denoting natural processes
21	**quantity**	nouns denoting quantities and units of measure
22	**relation**	nouns denoting relations between people or things or ideas
23	**shape**	nouns denoting two and three dimensional shapes
24	**state**	nouns denoting stable states of affairs
25	**substance**	nouns denoting substances
26	**time**	nouns denoting time and temporal relations

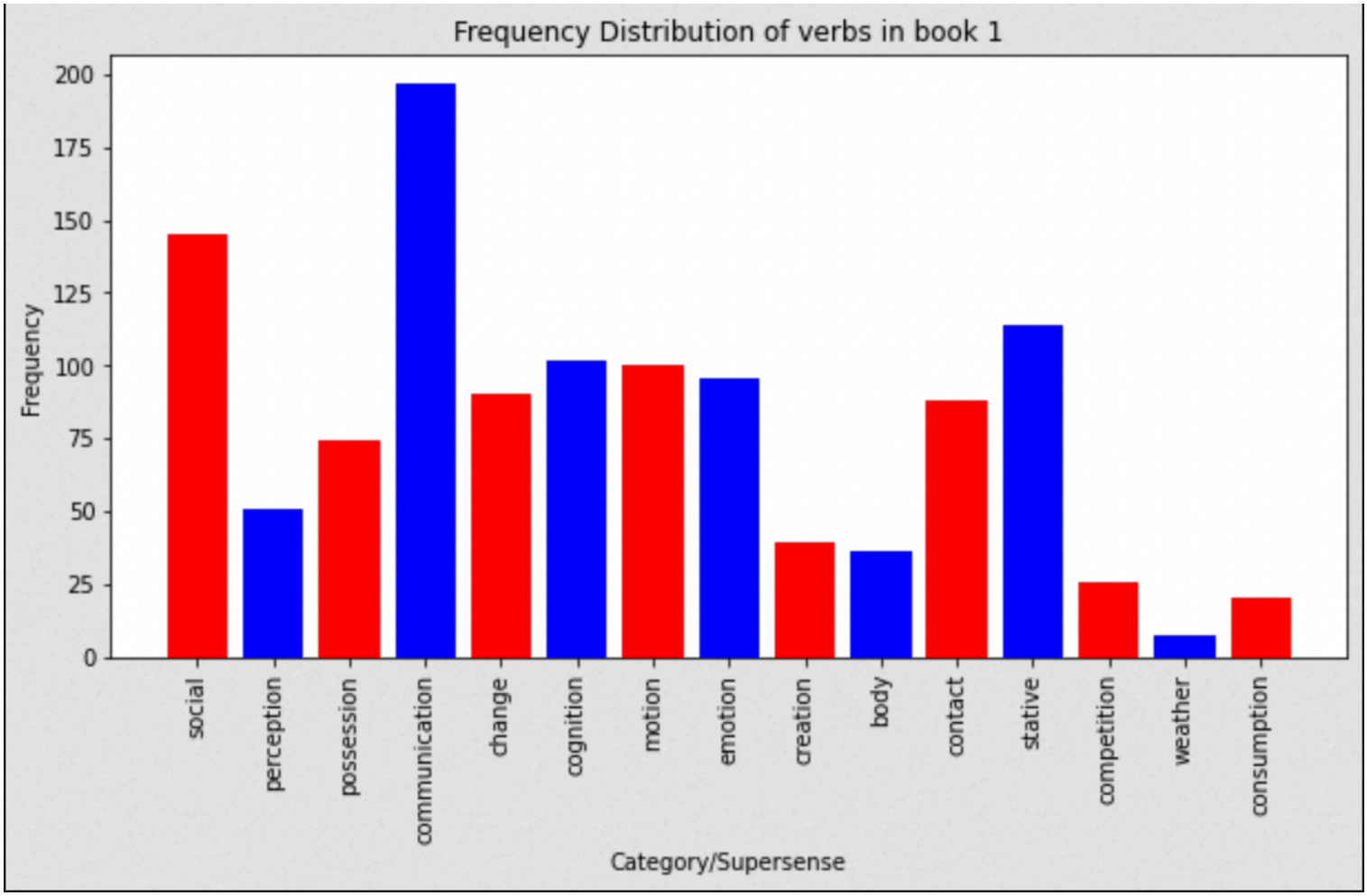
Verb categories present in the wordnet :-

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

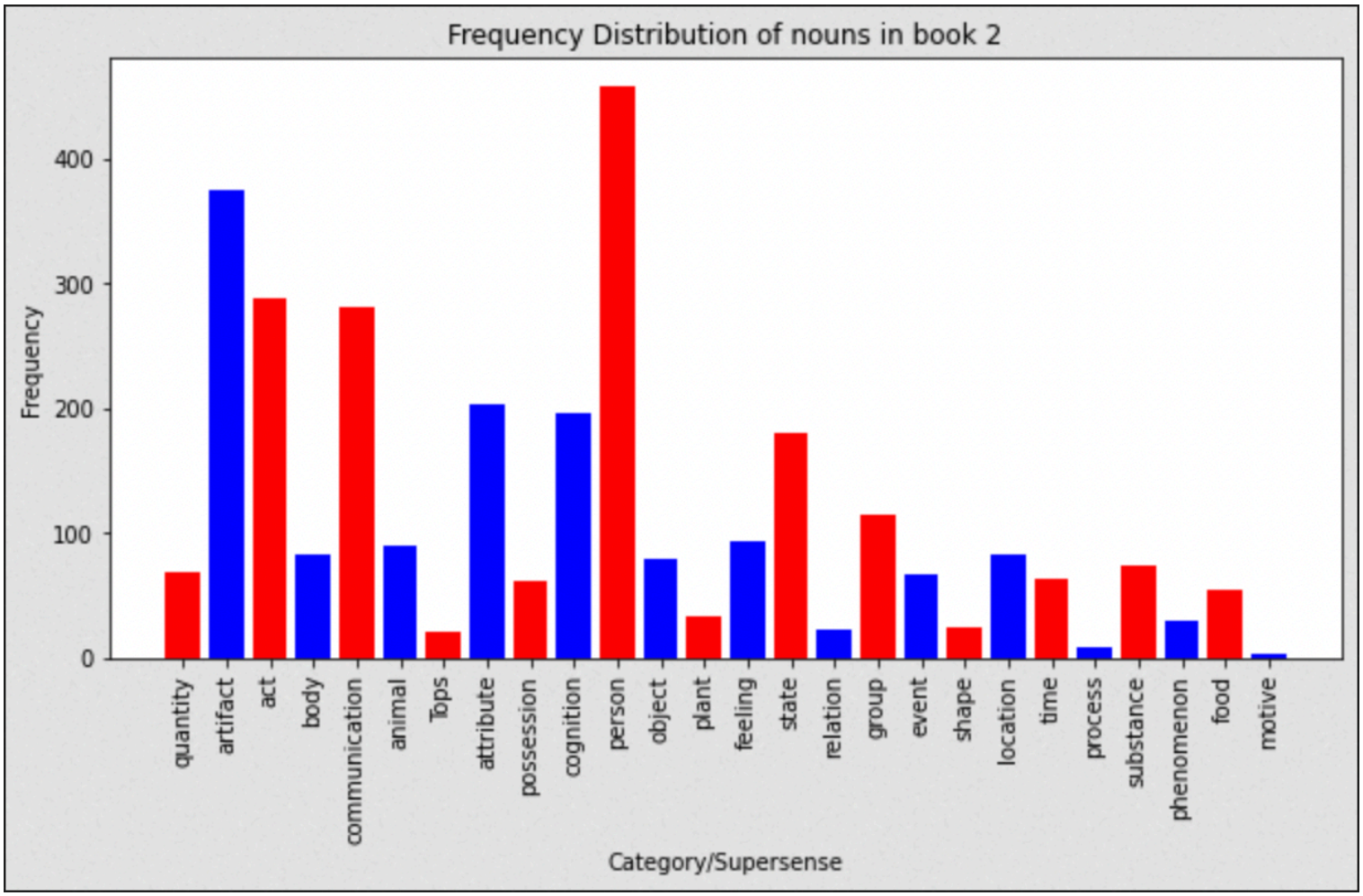
Frequency plot of the noun categories of the **nouns** of **book-1**



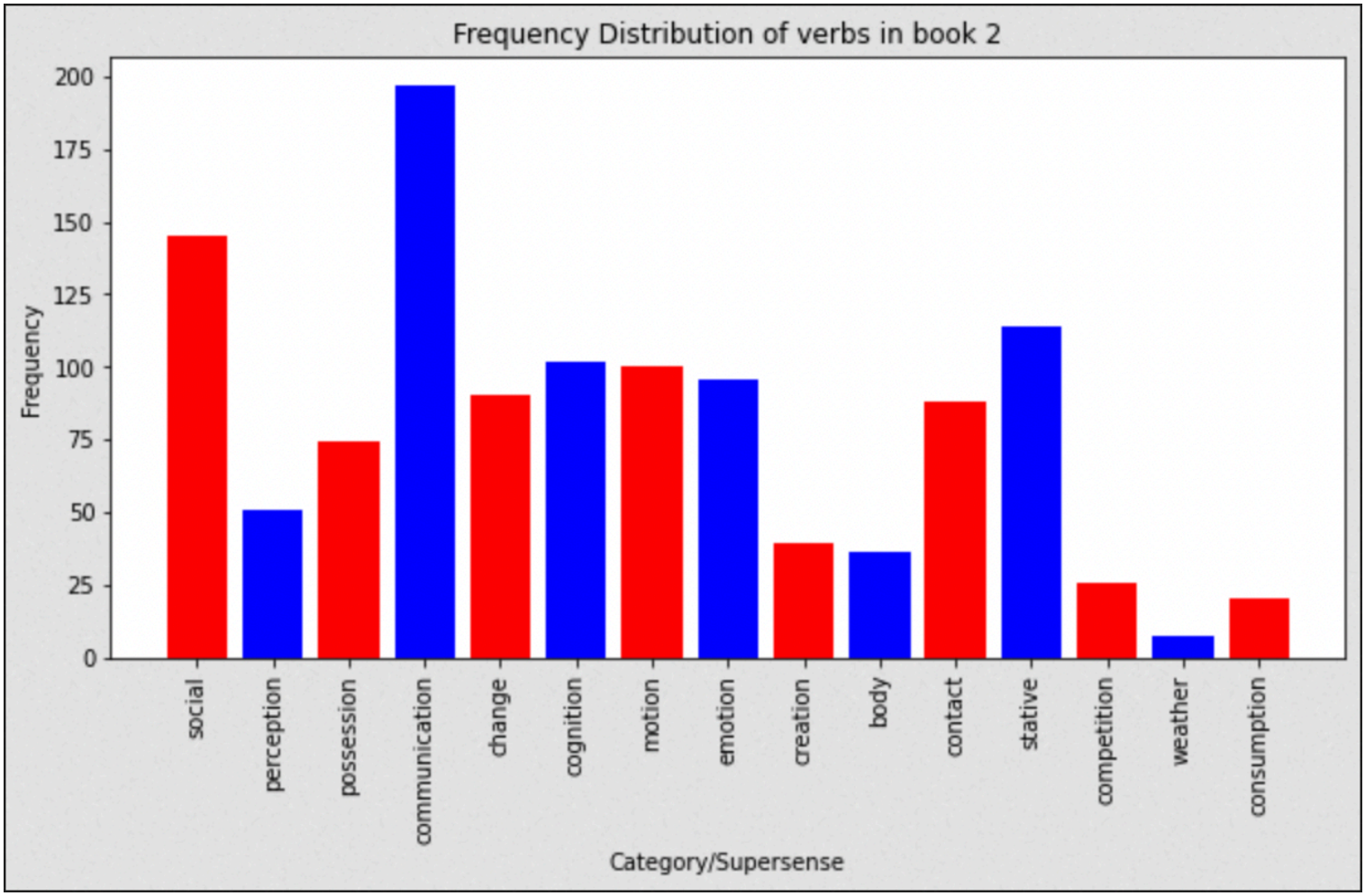
Frequency plot of the verb categories of the **verbs** of **book-1**



Frequency plot of the noun categories of the **nouns** of **book-2**



Frequency plot of the verb categories of the **verbs** of **book-2**



Part-2

Problem Statement :-

Recognise all the named entities in both the books.

Steps:

- First without doing any preprocessing, we perform tokenization for both the books.
- Both the books are POS tagged using nltk library.
- Now, on this tagged data, we perform entity recognition using *ne_chunk()* function.
- All the used entities are recorded for both the books.

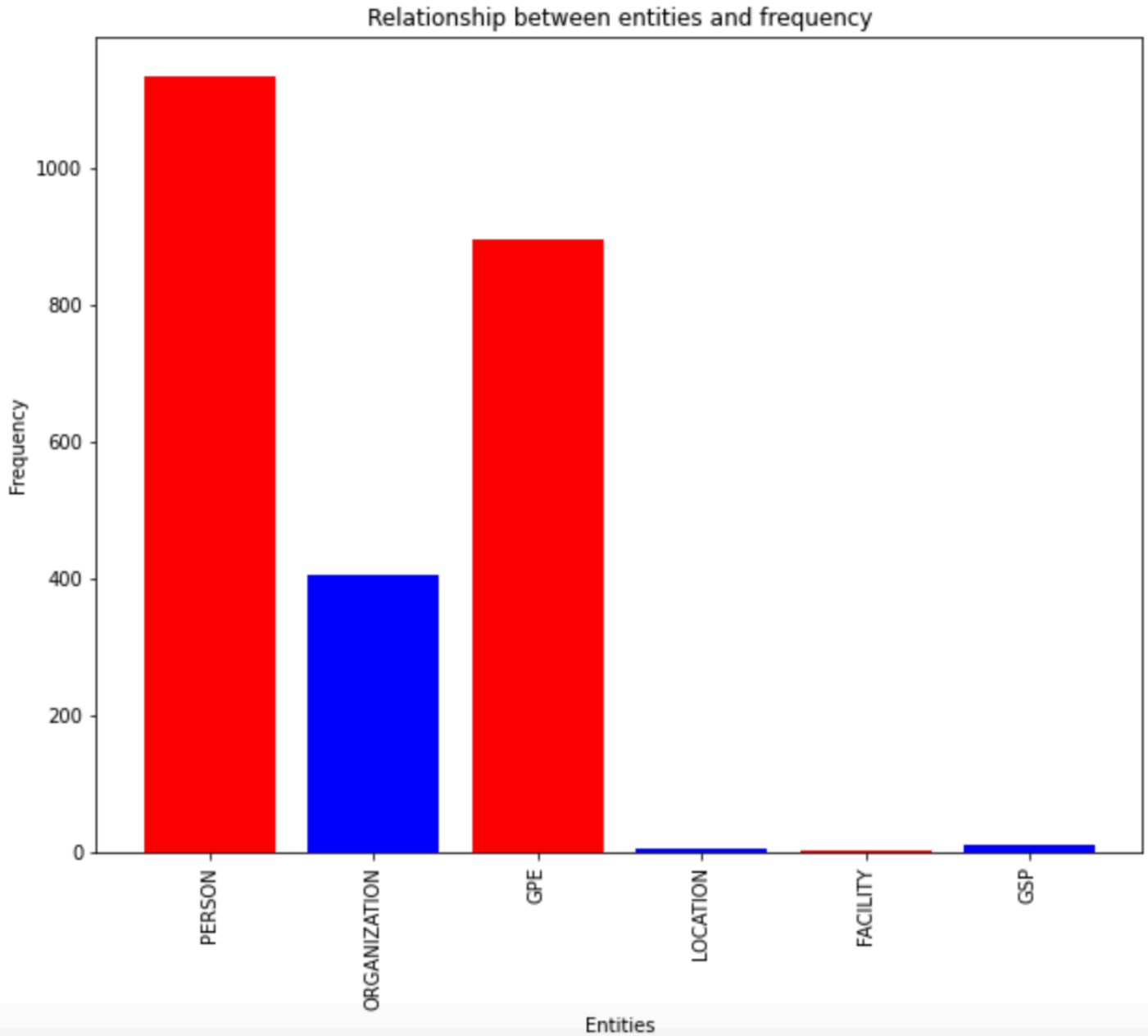
Entity Recognition:

- Sequence labeling is used to perform Named Entity Recognition (NER).
- First we encode our training data with IOB tags. This is done manually by domain experts.
- Set of features are associated with each token to be labeled.
- When an adequate set of features are extracted from a training set, it is encoded in a form appropriate to train a machine learning based sequence classifier.
- These extracted features are augmented with our earlier IOB scheme with more columns.
- Now a sequence classifier can be trained.

VISUALIZATION :-

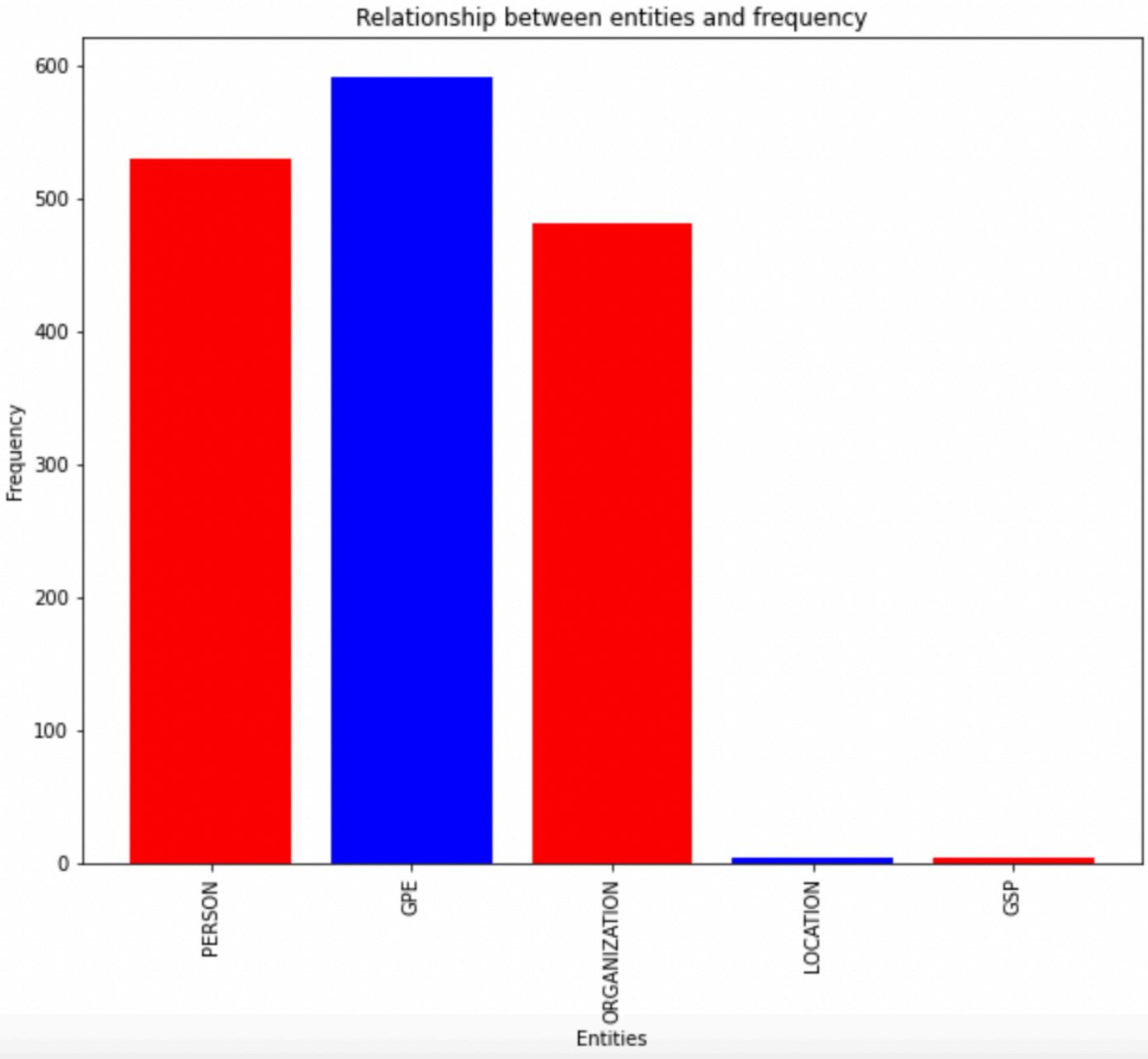
BOOK 1:

```
Plotting entities vs. frequency chart for book-1
```



BOOK 2:

```
Plotting entities vs. frequency chart for book-2
```



Performance Evaluation :

- We need a perfectly labeled data to compare our classifier with.
- We selected two random passages from each text book and manually tagged it ourselves with the right entity. For this purpose we used a particular entity, i.e., PER (person).
- Then we used our classifier to tag these passages.
- We created a function *(metrics)* to get the True Positives, False Negatives, False positives and True negatives.
- Using these values we found out the accuracy, precision, recall and f-measure of our data.

Result :

Two lists were made, namely “*truth”* and “*run”*. The *truth* list contains the manually labeled entities and *run* list contains the entities that were recognized by the algorithm. (The entity used for this purpose is Person).

The confusion matrix was made:-

	PREDICTED CLASS		
ACTUAL CLASS		Yes	No
	Yes	*TP*	*FP*
	No	*TN*	*FN*

TP refers to true positive, FP refers to false positive, FN refers to false negative and TN refers to true negative. TP is the count of positive records which are classified as positive. FP is the count of positive records which are not classified as positive. TN is the count of negative records which are classified as negative. FN is the count of negative records which are not classified as negative.

These values were calculated from the *truth* and *run* lists. Using these values, *accuracy*, *recall*, *precision* and *F-measure* are calculated via following formulae:

Accuracy = (TP + TN) / (TP + TN + FP + FN)
Recall = TP / (TP + FP)
Precision = TP / (TP + TN)
F-measure = (2 * recall * precision) / (recall + precision)

```
Running NER on random paragraphs from books to evaluate the algorithm

Paragraph-1 :
Maunally labelled PERSON entities :
{'Mr. Denny', 'Kitty', 'Bingley', 'Mr. Darcy', 'Mr. Wickham', 'Elizabeth', 'Miss Bennet', 'Lydia'}

Algorithm labelled PERSON entities :
{'Mr. Denny', 'Kitty', 'Darcy', 'Bingley', 'Mr. Wickham', 'Mr. Darcy', 'Miss Bennet', 'Elizabeth'}

Evaluation :
Accuracy:  0.5384615384615384
Recall:  1.0
Precision:  0.5384615384615384
F-measure:  0.7000000000000001
               Predicted Negative  Predicted Positive
Negative Cases                  0                 6.0
Positive Cases                  0                 7.0
```

```
Paragraph-2 :
Maunally labelled PERSON entities :
{'Slamecksan', 'Blefuscu', 'Lustrog', 'Tramecksan', 'Lilliput'}

Alogorithm labelled PERSON entities :
{'Blefuscu', 'Lustrog', 'Reldresal'}

Evaluation :
Accuracy:  0.4
Recall:  0.5
Precision:  0.6666666666666666
F-measure:  0.5714285714285715
               Predicted Negative  Predicted Positive
Negative Cases                  0                 1.0
Positive Cases                  2                 2.0
```

Part-3

Adding The third book: Myths and Legends of Ancient Greece and Rome, by E.M. Berens

URL: https://www.gutenberg.org//cache/epub/22381/pg22381.txt

Problem Statement:-

Create TF-IDF vectors for all books and find the cosine similarity between each of them and find which two books are more similar

We first imported the books, preprocessed it, removed stop words and vectorise it and compare them

TF-IDF Vectorisation and Cosine Similarity:-

- We will be using the ‘numpy’ library and the module ‘sklearn.feature_extraction’
- Sklearn.feature_extraction is used for embedding the feature vectors. I provides different types of embedding techniques like TF-IDF, CountVectorizer, Word2Vec etc.
- We will be using Tdidfvectorizer.
- We will be removing all the stop words during the vectorisation.
- We will be computing them in the form of matrix and process it using Pandas.DataFrame

Results:

	10	100	101	102	103	104	105	106	107	108	...	zealots	zealous	zenith	zephyrs	zephyrus	z
B1	0.000739	0.000000	0.000000	0.0000	0.0000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.0000	0.000000	0.00
B2	0.000622	0.000000	0.000000	0.0000	0.0000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.001054	0.001602	0.001054	0.0000	0.000000	0.00
B3	0.001949	0.004401	0.004401	0.0022	0.0011	0.007701	0.011002	0.007701	0.007701	0.009902	...	0.000000	0.000837	0.000000	0.0011	0.007701	0.00

3 rows × 16545 columns

For computing cosine similarity, we will be using cosine_similarity().
The Similarity between the books are as follows

Book1 - Book2 = 0.47258612
Book2 - Book3 = 0.47534742
Book1 - Book3 = 0.2607772

This shows that book2 is way more similar to book3 and book1 and book3 are the least similar

Lemmatising the texts

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

After lemmatization, we created their TF-IDF vector matrix. There, we can see a quite significant change.

Results:

	10	100	101	102	103	104	105	106	107	108	...	zealot	zealous	zenith	zephyr	zephyrus
B1	0.000573	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000
B2	0.000469	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000795	0.001209	0.000795	0.000000	0.000000
B3	0.001303	0.002941	0.002941	0.00147	0.000735	0.005146	0.007351	0.005146	0.005146	0.006616	...	0.000000	0.000559	0.000000	0.000735	0.005146

3 rows × 14657 columns

The output is:

Book1 - Book2: 0.61686472
Book2 - Book3: 0.69378468
Book3 - Book1: 0.52198365

This time also, Book2 is the most similar to Book3 and Book1 is least similar to Book3.

References:-

- NLP class lectures
- https://www.nltk.org/api/nltk.sem.html?highlight=extract_rels#nltk.sem.relextract.extract_rels
- https://www.nltk.org/api/nltk.chunk.html?highlight=ne_chunk#nltk.chunk.ne_chunk
- https://www.nltk.org/book/ch05.html
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html