NLP Project Rounds 1 & 2

Group Name - PhiGex

Group Members -

1. Ramit Agarwal 18ucs144
2. Shaswat Kumar 18ucs125
3. Shivam kejriwal 18ucs123
4. Namit Bhandari 18ucs178

Github link for the code :-

https://github.com/Ramitphi/NLP_Project

Google Colab Notebook Link : Click to view

Index

Project Round 1

1. **Data description :-**

We have taken two text books for the data analysis. We fetched the raw text data from it and splitted it into lines.

The code for it are given below:-

```
b1 = open("14249.txt", "r")
T1= b1.read()

b2 = open("34861.txt", "rt")
T2=b2.read()
```

2. **Data Pre-processing and preparation Steps :-**

A) For removing the upper section containing the details of the book and removing the lower section containing the license,certification etc , we have written the following code .The code find the starting index of the chapter and the staring index of the end of the project and then we slice text between these two indexes to get the content for the further processing.

```
T1.find("CHAPTER")    #first found the index of the starting of
chapter I
T1.find("End of the Project") #then we will find the index of the
starting of the lower part of the book

resT1 = T1[1630:366315] #we will use slicing to slice text between
those indexes
```

B) For the removal of redundant data and changing all the uppercase letters to lowercase for normalizing it, we have written the following code.resT1 is the preprocessed text data and lower() is used to convert all the uppercase to lowercase characters.

```
#Converting the text into lowercase.
                    resT1=resT1.lower()
```

C) For Removing the running chapter names, we are using the code given below. The regular expression finds the substring chapter from the text data and then replaces it with the none character.

```
resT1 = re.sub(r'chapter \w+','',resT1)
```

D) For Removing the Punctuation and some special characters we are using the following code. In this, we are using a regex statement for the removal of all the punctuation and special characters from the text by only selecting the non-punctuation and non-special characters from the given text:-

```
resT1 = re.sub(r'[^a-zA-Z0-9\s]', '', resT1)
```

3. **Problem Statements:**

3.1 **Analyze the frequency distribution of tokens in T1 and T2 separately**

We have done the analysis of T1 and T2 by using the following code. We imported seaborn and matplotlib.pyplot for plotting different plots and graphs. We also imported pandas for helping us in manipulating the list of words or tokens.

```
#importing the libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

```
tokensT1df=pd.DataFrame(tokensT1)
tokensT2df=pd.DataFrame(tokensT2)

freqt1=tokensT1df.value_counts()
freqt2=tokensT2df.value_counts()
```

Above code divides the text into tokens named as tokensT1 and tokensT2. Using the dataframe we found the frequency of each word in both the texts.

	Token	Counts
6793	the	3988
1	a	1735
4649	of	1606
6924	to	1566
3148	he	1410

Token Count of T1

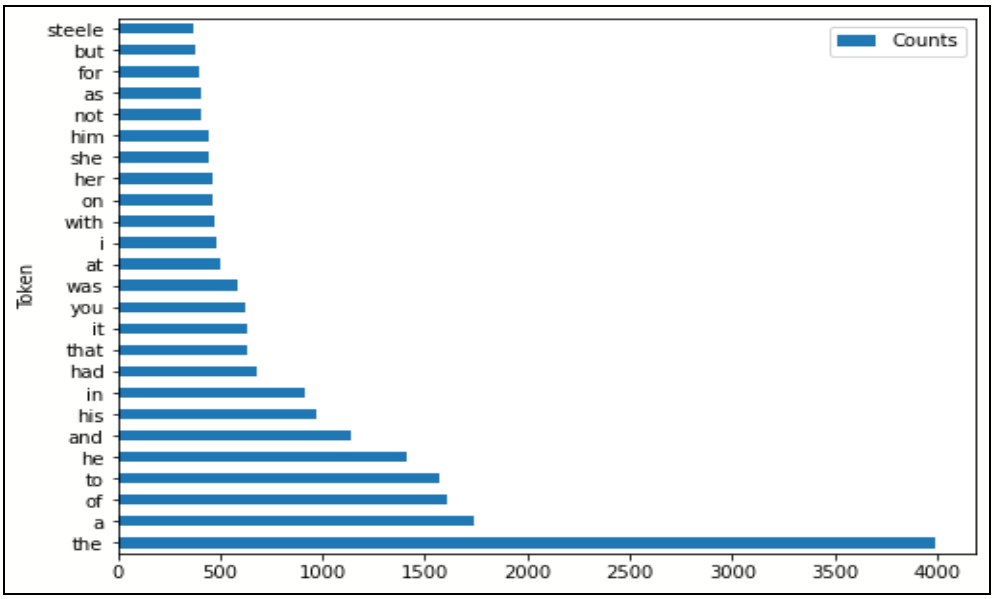
The result shown above is obtained after analysing the frequencies of Token count of T1. There are some words like 'the' that have a frequency of 3988 which is maximum among all. There are also some tokens that have lower frequency .

There is a huge difference in frequencies of tokens as we can see the words of frequency 4000 and also words of frequency 1 in the tokens list. Similar kind of processing is done for T2.

Considering top 25 tokens of the token list T1 & T2, We Plotted the frequency count for Token T1 and Token T2 by using code below-

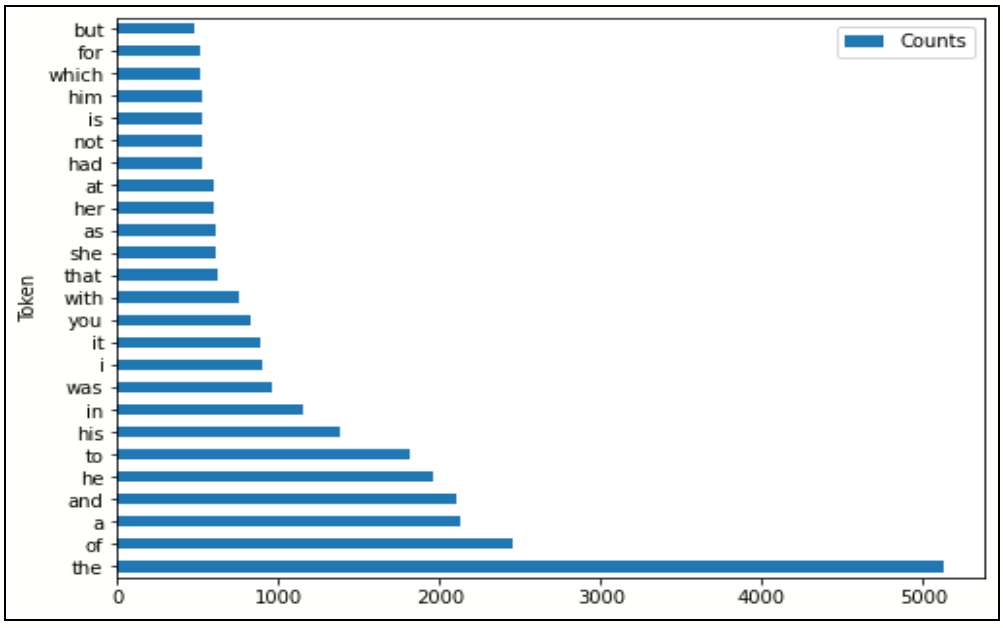
```
t1df.plot(x= 'Token',y='Counts',kind='barh',figsize=(8, 6))
t2df.plot(x= 'Token',y='Counts',kind='barh',figsize=(8, 6))
```

: **Plot of Token T1 with frequency :-**



Here X coordinate shows the frequency count and Y coordinate shows the tokens name.

: **Plot of Token T2 with frequency:-**



3.2 Create a Word Cloud of T1 and T2 using the token that you have got

We have created the word cloud from the tokens we got by using the code snippet given below

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd
comment_words = ''
stopwords = []
tokens = tokensT1
```

```
comment_words += " ".join(tokens)+" "
      wordcloud = WordCloud(width = 800,
height = 800, background_color ='white',
      stopwords = stopwords, min_font_size
      = 10).generate(comment_words)
```

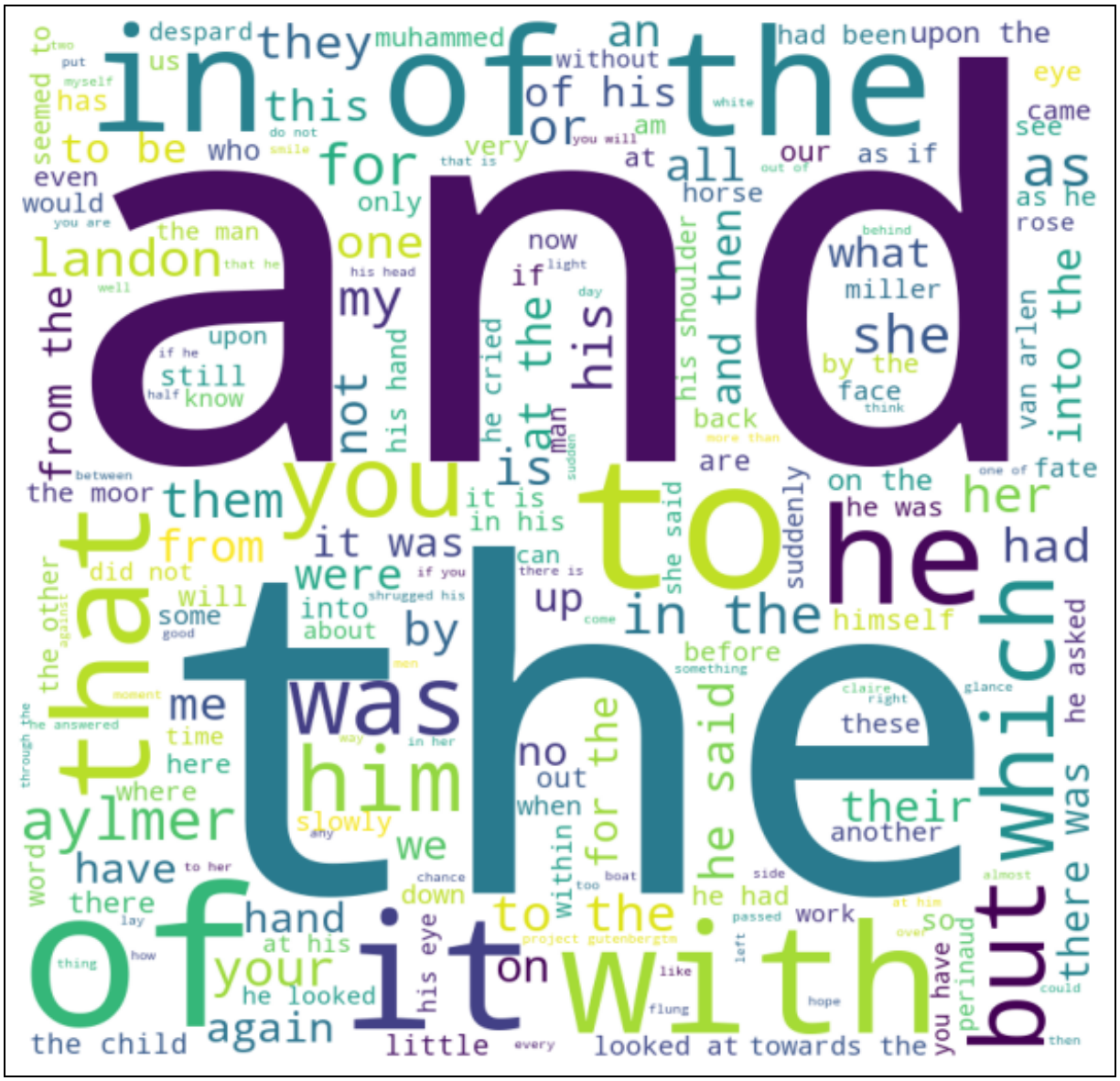
```
# plot the WordCloud image
plt.figure(figsize=(8,8), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Using the code above we imported the word-cloud library for creating the visualization of the word cloud. We also imported matplotlib.pyplot for plotting the words on a 2d sheet. We joined all the tokens in the variable named as comment words and seen in the above code snippet using join() function and then we used the inbuilt function of wordcloud and passed the parameters in it that is width and height of the sheet, background color, stopwords , and font size and after this we used the pyplot function to create the graph.

Same code is used for creating the word clouds of T2 just use the tokens of T2 in the comment_words variable.

a) **WordCloud of tokenT1 with stopwords .**

b) WordCloud of tokenT2 with stopwords .



3.3 Remove the stopwords from T1 and T2 and then again create a word cloud - what's the difference it gives when you compare it with word clouds before the removal of stopwords?

Removing StopWord

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
tokensT1withoutstop = [w for w in tokensT1 if not w in stop_words]
```

Similar stuff was done for the T2 to remove stopwords from it

Now from the code snippet that is shown above are changed to remove the stopwords and the code that we used for creating word cloud without using the stopwords is shown below:

```
comment_words = ''
tokens = tokensT1withoutstop
comment_words += " ".join(tokens)+" "

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

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

In the code snippet given above the stopwords are removed for T1

The difference between those two are: the words will not be seen in the word cloud with stop words provided that is, the work of stopword is to remove the given word in the stopword from

the word cloud.for eg: suppose the word ‘the’ is in the stopword then the word the will not be seen in the word cloud and the word-cloud without stopword will show the word ‘the’ in it.

a) Word-cloud of TokenT1 without stop words

In a text mostly stopwords have a high frequency and after removing it, we can see ‘john’ , ‘steele’ and some other words that are prominent in the word cloud which implies that these words were widely used in the book apart from the stopwords

---

b) Word-cloud of TokenT2 without stop words

In a text mostly stopwords have a high frequency and after removing it we can see ‘aylmer’ , ‘landon’ and some other words that are prominent in the word cloud which implies that these words were widely used in the book apart from the stopwords

---

3.4 Evaluate the relationship between the word length and frequency for both T1 and T2- what's your result?

The relationship between world length and frequency of both T1 and T2 can be given by the following code snippet below.

'freqt1.head()' shows the top 5 tokens with their corresponding frequency.

```
tokensT1df=pd.DataFrame(tokensT1)
tokensT2df=pd.DataFrame(tokensT2)

freqt1=tokensT1df.value_counts()
freqt1= pd.DataFrame(freqt1,columns=['freq'])
freqt1w= freqt1.reset_index()
freqt1w.head()
```

	tokens	frequency
0	the	3988
1	a	1735
2	of	1606
3	to	1566
4	he	1410

```
freqt1w['length'] = freqt1w['tokens'].apply(lambda x: len(x))
```

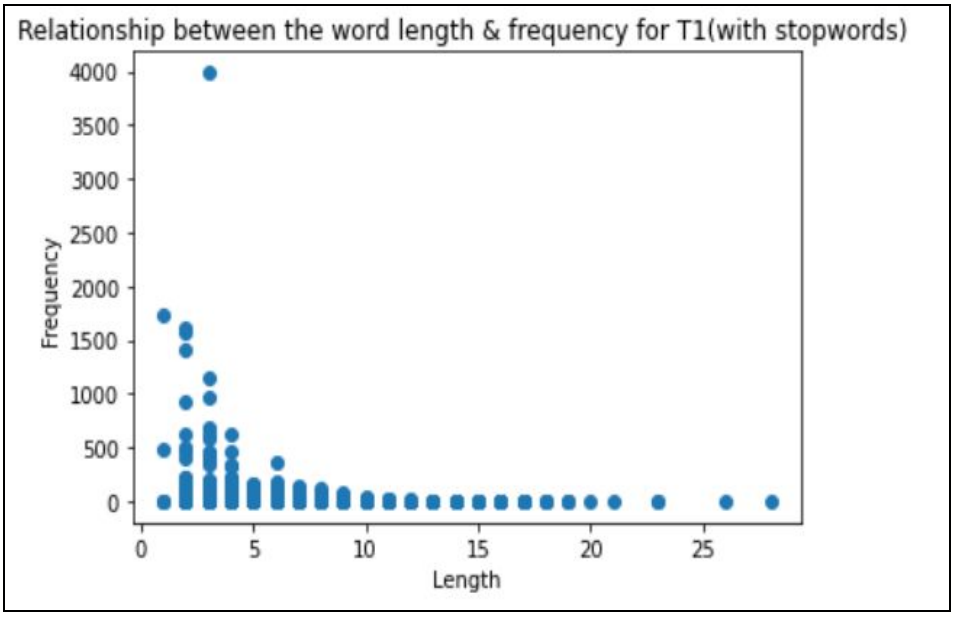
#using the apply method to find the length of all the tokens and storing it with their corresponding token using dataframe.

	tokens	frequency	length
0	the	3988	3
1	a	1735	1
2	of	1606	2
3	to	1566	2
4	he	1410	2

```
plt.scatter(freqt1w['length'], freqt1w['frequency'])

plt.title("Relationship between the word length & frequency for
T1(with stopwords) ")
plt.ylabel("Frequency")
plt.xlabel("Length")
```

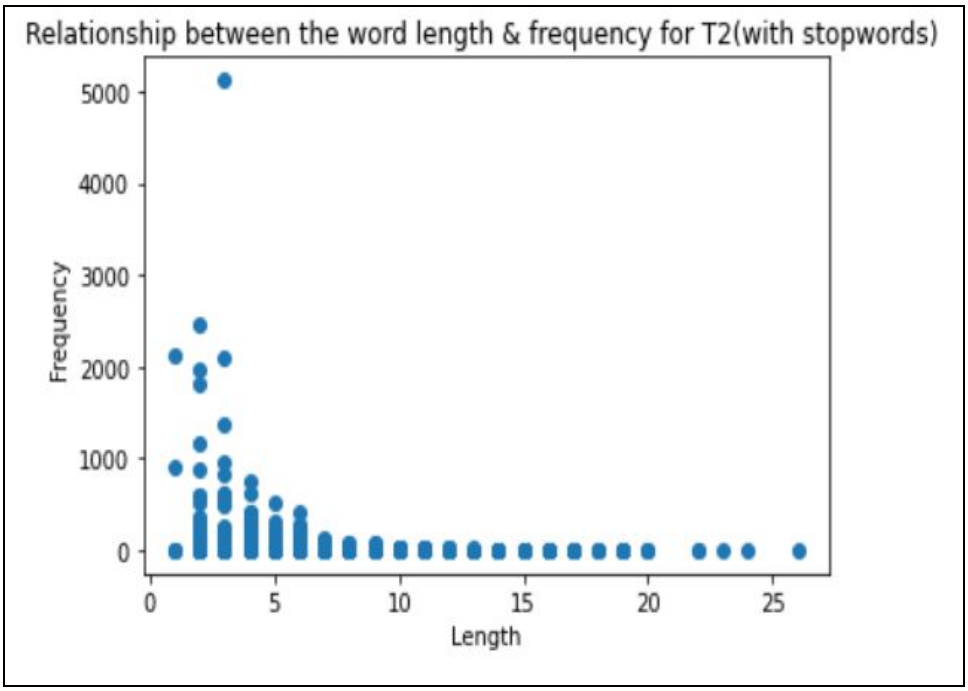
Above code snippet will have counts of all the words that will be stored in wordlen_count_list and each word-length will be plotted against its frequency . That is shown below in the graph:-

T1 graph->



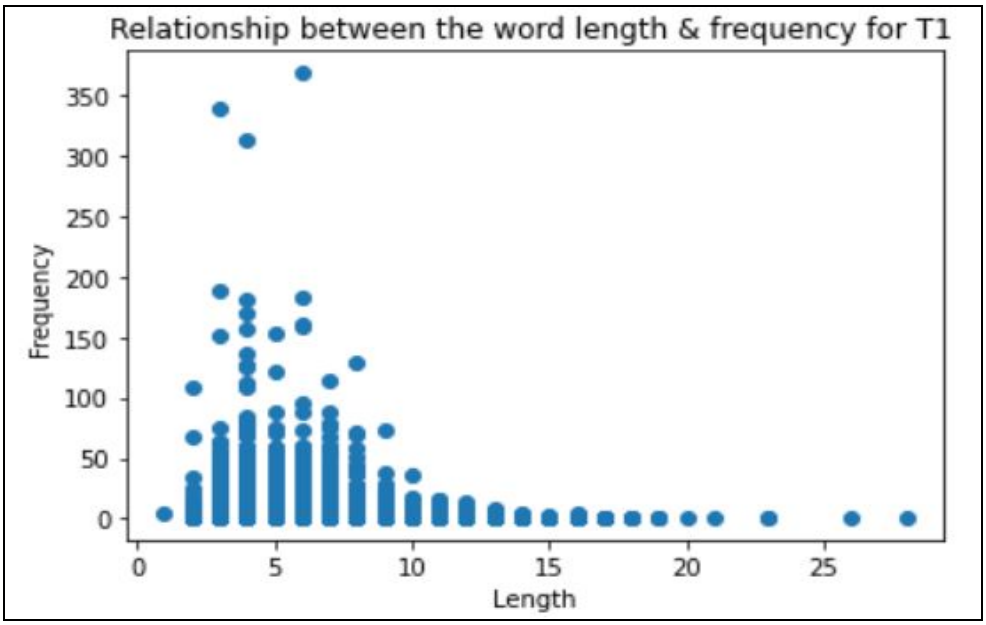
Word with world length from 1 to 5 have high frequency as compared to longer words

T2 graph->



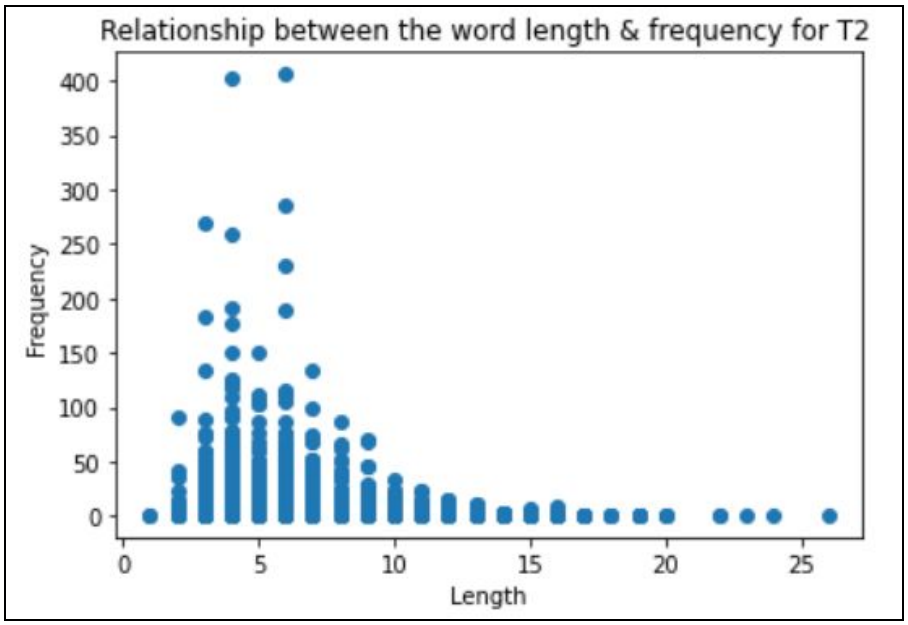
Similar plot for the T1 & T2 without Stopwords

T1(without stopwords) >>



Words from length 1 to 10 are having a good number of occurrences and word of length 6 is having the maximum occurrence .

T2(without stopwords) >>



The above plot represents the relationship between word length and it's count for tokens in T2. We can observe that words of length between 3 to 6 are having a good number of occurence as compared to words of length 20 or above it.

3.5 Do PoS Tagging for both T1 and T2 using anyone of the four tagset studied in the class and Get the distribution of various tags

We have assigned tag to every token by using function of Pos Tagging in NLTK .The Code Snippet is given below :-

We have used 'averaged_perceptron_tagger ' under nltk library for POS tagging and storing the pos tags together with their count in a list named as 't1tagcountdf' and 't2tagcountdf' .

```
import nltk;
nltk.download('averaged_perceptron_tagger');
tags_T1 = nltk.pos_tag(tokensT1)
tag1 = [ t for (w,t) in tags_T1]
t1tags, t1tagcount = np.unique(tag1, return_counts=True)
t1tagcountdf = pd.DataFrame(list(zip(t1tags, t1tagcount)),
            columns =['Tags', 'Counts']);

t1tagcountdf.sort_values(by=['Counts'],ascending=False,inplace=True)
t1toptagdf= t1tagcountdf[:30]
t1toptagdf.plot(x= 'Tags',y='Counts',kind='barh',figsize=(8, 6))
```

t1tagcountdf.head()

	Tags	Counts
0	CC	1750
1	CD	408
2	DT	6987
3	EX	98
4	FW	17

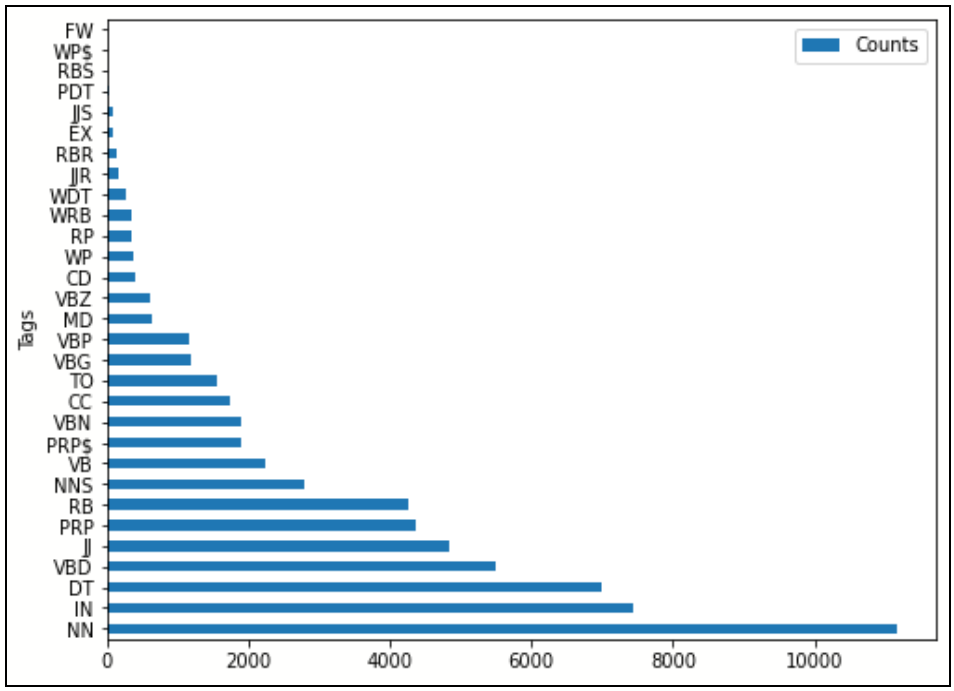
t2tagcountdf.head()

	Tags	Counts
0	CC	2811
1	CD	520
2	DT	8847
3	EX	240
4	FW	18

The image above shows the tags and their corresponding counts for both the token set T1 and T2.

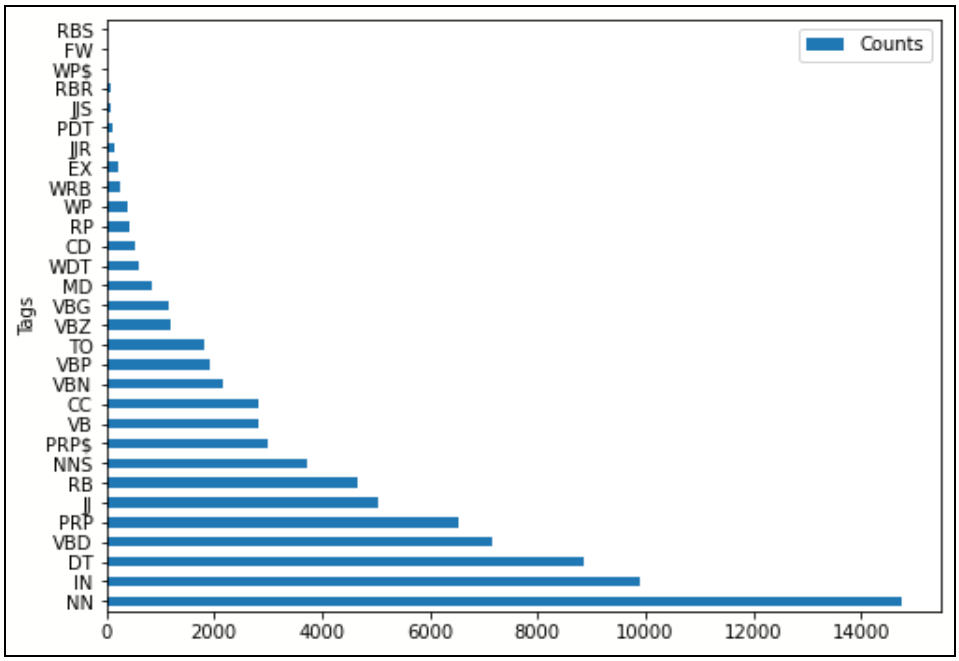
We have calculated count of each tag associated with tokens and plot of tag and frequency is given below:

-> **Plot for token T1:**



Inference : Noun occur predominantly in the text followed by preposition and by determiners

-> **Plot for Token T2:**



Project Round - 2

1. **a). 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**

Step 1 : We have used the NLTK library for POS tagging the words.

```
tags_T1 = nltk.pos_tag(tokensT1)
tags_T2 = nltk.pos_tag(tokensT2)
```

Here, tags_T1 = the POS tags results for Book 1,
tags_T2 = the POS tags results for Book 2,
tokensT1 = the list of tokens of the Book 1.
tokensT2 = the list of tokens of the Book 2

Step 2 : In next steps we have identified the noun & verb categories and then stored them separately in the list for the WordNet Classification. We conclude from observing the POS tags that all the tags that start with 'N' are Nouns and those that start with 'V' are Verbs.

```
nouns_T1=[]
for i in tags_T1:
  if(i[1][0] == 'N'):
    nouns_T1.append(i[0])
```

```
nouns_T1[:10]
```

```
['mr',
 'gilletts',
 'charge',
 'means',
 'lets',
 'decks',
 'look',
 'course',
 'charles',
 'gillett']
```

We have used a list nouns_T1 to store the nouns in Book1. From the image it is evident that the stored words are nouns.For example gilletts , charge , charles are all forms of Nouns.

Similarly, verb_T1 is used to store the extracted results of the verbs in Book 1, verb_T2 , nouns_T2 are used to store the verbs and nouns in book 2 respectively.

Step 3 : We downloaded the wordnet from the NLTK library and then imported it from the NLTK corpus.

```
nltk.download('wordnet')
```

```
from nltk.corpus import wordnet
```

Step 4 : Using the WordNet the nouns and verbs got divided into the respective categories like in book 1, 'mr' categories to noun.communication and 'gilletts' categories to noun.act as shown in code snippet below.

```
nt1=[]

for i in nouns_T1:
  syn =  wordnet.synsets(i,pos=wordnet.NOUN)
  if len(syn)>0:
    nt1.append(syn[0].lexname())

print(list(zip(nouns_T1,nt1)))
```

```
[('mr', 'noun.communication'), ('gilletts', 'noun.act'),
```

Here, nt1 represents the list that contains the final categorisation of the nouns in Book1.

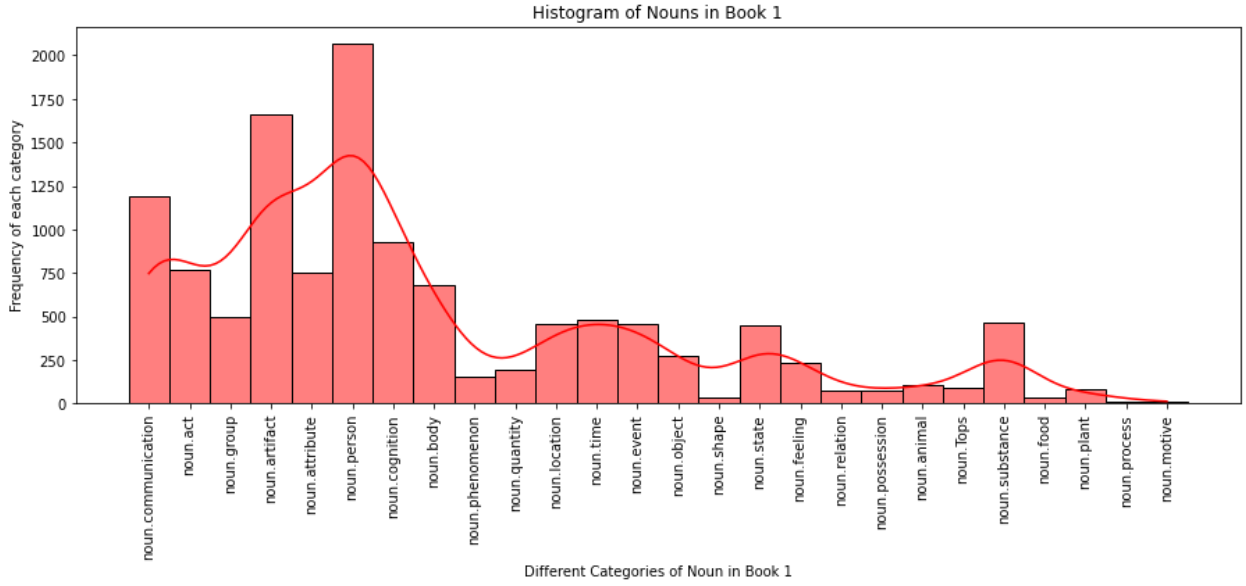
```
vt1=[]

for i in verb_T1:
  syn =  wordnet.synsets(i,pos=wordnet.VERB)
  if len(syn)>0:
    vt1.append(syn[0].lexname())
#print(vt1)
print(list(zip(verb_T1,vt1)))
```

```
[('mdeah', 'verb.motion'), ('go', 'verb.possession'),
```

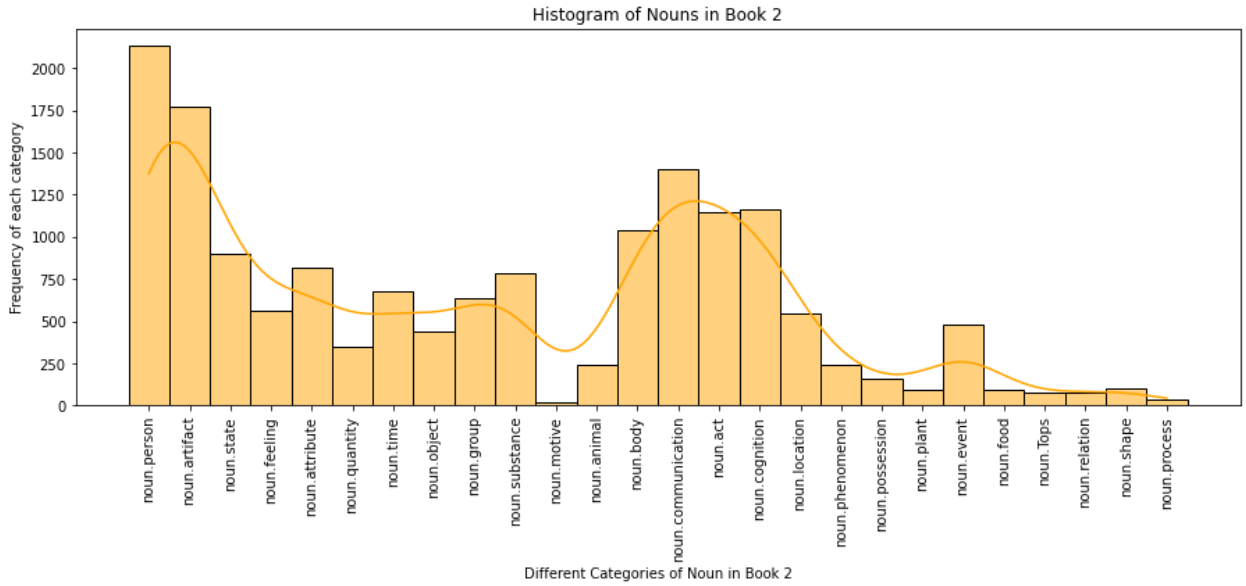
The image above shows the categorisation of verbs in Book1 and the results are stored in the list vt1. For example : go categories to verb.possession , mdeah categories to verb.motion and so on.Similarly the categorisation is done for the verbs in book2 and nouns in book2 and the results are stored in list nt2 for nouns in book2 and vt2 for verbs in book2

1. **b) Get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same for each novels**

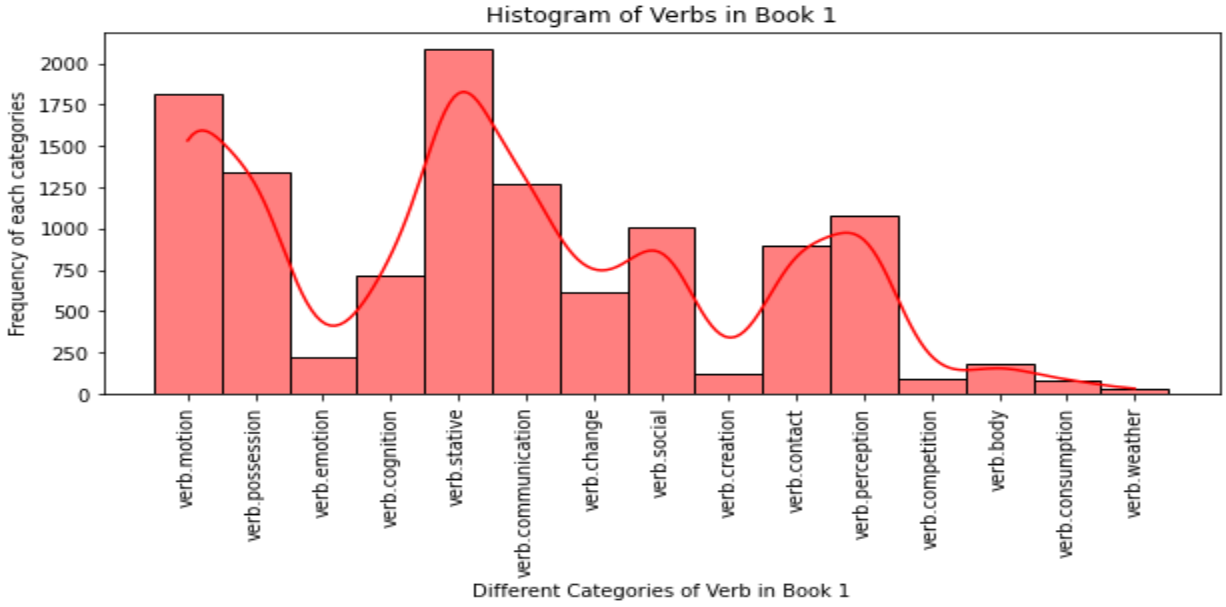
We plotted histogram for the frequency plotting and we can see from the histogram that for book T1 , noun.communication has a frequency of around 1220+ and noun.person has more that 2000 frequency and which is the most in book 1 , and noun.motives has the least number of frequency count that is close to 2.



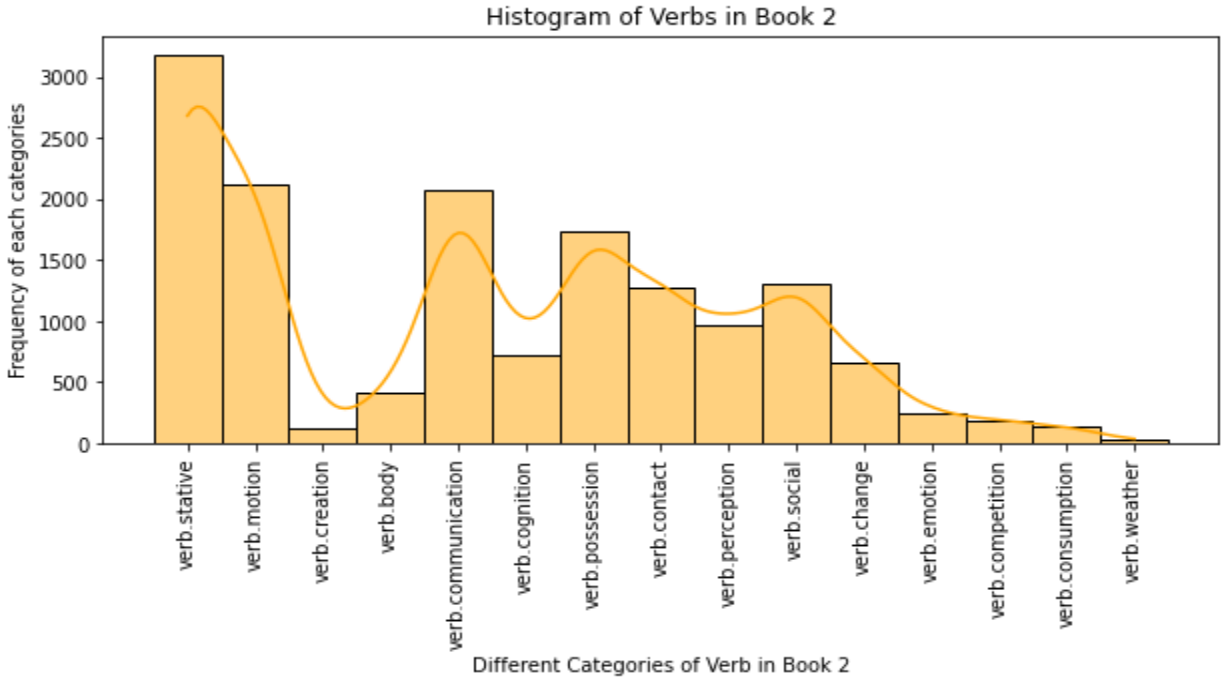
In book 2 the maximum frequency count is of noun.person and that is more than 2000 and least count is for noun.motive.



For the verb also we plotted the frequency count in the form of histogram. For the book 1 , Verb.motion has 1750+ frequency count , verb.stative has 2000+ frequency count which is maximum of all and verb.weather has the least count which is not more than 10.



Similarly For the book 2, as we can see in the histogram, verb.stative has the largest frequency count and that is more than 3000 and verb.weather has the least count among all.



2. **a) Recognise all Persons, Location, Organisation (Types given in Fig 22.1) in book. For this you have to do two steps: (1) First recognise all the entities and then (2) recognise all entity types.**

The Spacy NER system contains a word embedding strategy using sub word features and "Bloom" embed, and a deep convolutional neural network with residual connections.

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

For implementing this we imported **spacy** , 'en_core_web_sm' is the pretrained model that spacy uses and with the help of this library we recognised all the entities in book 1 and later for book 2.

```
book1 = T1
entity_tag_book1 = []
lst1 = ner(book1)
for word in lst1.ents:
  x = word.text
  y = word.label_
  entity_tag_book1.append([x,y])
```

```
entity_tag_book1[:10]
```

```
[['Half', 'CARDINAL'],
 ['Frederic S. Isham', 'PERSON'],
 ['eBook', 'ORG'],
 ['eBook', 'ORG'],
 ['Half', 'CARDINAL'],
 ['Frederic S.', 'PERSON'],
 ['December 3, 2004', 'DATE'],
 ['14249', 'MONEY'],
 ['English', 'LANGUAGE'],
 ['Kevin Handy', 'PERSON']]
```

As we can see in the image that all types of entities are recognised like CARDINAL, PERSON, ORG, DATE , MONEY and many more and for evaluating it.

As we can see in the image , T1 is the raw text of book 1 and entity_tag_book1 is the list that has the entity name as well as it's corresponding entity type. And the first 10 items of the list are shown in this image. Here 'half' is referred to CARDINAL , 'eBooK' is referred to 'ORG' and so on.

Similarly we have used the raw text data of book 2 and stored the entities name and their types in the entity_tag_book2 list.

2. b) Use performance measures to measure the performance of the method used - For evaluation you take a considerable amount of random passages from the Novel, do a manual labelling and then compare your result with it. Present the accuracy here

: Evaluating the accuracy of the Named Entity Recognition of Book 1.

```
s3 = nerT1[:3000]
s3
```

Here, nerT1 is the whole preprocessed text of Book 1 after removal of the headers and the footers and s3 is the first 3000 characters of Book 1 used for the Named Entity Recognition.

As we can see from the image entity_tag_passage_book1 is a list that shows the entities with their labels marked by the model itself . For example 'gilletts' is a name of a person so it has been labeled as PERSON and australia is a country so it is labeled as GPE.

```
entity_tag_passage_book1 = []
lst3 = ner(s3)
for word in lst3.ents:
  x = word.text
  y = word.label_
  entity_tag_passage_book1.append([x,y])

entity_tag_passage_book1
```

```
[['gilletts', 'PERSON'],
 ['charles althoughdo', 'PERSON'],
 ['gillett', 'PERSON'],
 ['one', 'CARDINAL'],
 ['sixth', 'ORDINAL'],
 ['australia', 'GPE'],
 ['charles', 'PERSON'],
 ['one', 'CARDINAL'],
 ['first', 'ORDINAL'],
 ['charles\n\n', 'PERSON'],
 ['mladymlord', 'PERSON'],
 ['gillett', 'PERSON'],
 ['captain macpherson', 'PERSON'],
 ['one', 'CARDINAL'],
 ['nelson', 'PERSON'],
 ['macpherson', 'PERSON'],
 ['charles', 'PERSON'],
 ['charles party', 'PERSON'],
 ['macpherson', 'PERSON'],
 ['macpherson', 'PERSON'],
 ['charles', 'PERSON']]
```

Total number of entities marked by the model is 21 in which 'mladymlord" is marked incorrectly because it is not a Person.

The text below represents the first 3000 characters of the text of Book 1 and in the figure all the Bold words represent the Entities marked manually.There are a total 24 entities that we have found in the passage and the Model shown above catches 21 entities.

Fig : The sample text of the Book 1 for the evaluation of accuracy

mr **gilletts** charge
by all means mudah let's go down between decks and have a look at them.

of course if you wish sir **charles althoughdo** you think we shall be edified mr **gillett.**

that depends lady and the speaker a man with official manners and ferret like eyes shifted from **one** foot to another on what degree or particular class of **criminal** your ladyship would be interested in he added if in the ordinary category of skittle sharper or thimblerigger with a suspicion of mild scorn then i do not imagine your ladyship would find much attraction in the present cargo but on the other hand in a livelier tone if your ladyship has any curiosity or shall we say a psychological bent regarding the real outandouter the excursion should be to your liking for rubbing his hands a properer lot of cutthroats and bad magsamen it has never been my privilege to escort across the equator and this is my **sixth** trip to **australia** how interesting how very interesting the lady's voice floated languidly sir **charles** is quite right we must really go down at any rate it will be a change after having been shut up so long in that terrible stateroom.

One moment mlady there's a little formality that must be observed **first** formality and the lady who was of portly appearance and uncertain age gazed from the speaker standing deferentially before her to a man of size weight and importance seated in a comfortable chair at her side what does he mean sir **charles** regulations **mladym lord** was the answer no one allowed on the prisoners deck without the captains permission there he is now then be good enough to beckon to him said the lady .

but this mr **gillett** agent of the **police** discreetly declined to do captain **macpherson** was a man not to be beckoned to by any **one** much less by him as he stood squarely in the center of the ship he looked like a mariner capable of commanding his boat and all the people aboard indeed some of the characteristics of his vessel seemed to have entered into his own makeup the man matched the craft broad nosed wide of beam big massive obstinate looking the lord **nelson** plowed aggressively through the seas with every square sail tugging hard at her sturdy masts she smote and overrode the waves and beating them down maintained an unvarying stubborn poise but although she refused to vacillate or shuffle to the wooing efforts of the uneasy waters she progressed not without noise and other foamed and fumed mightily at the bow and left behind her a wake receding almost as far as the eyes might reach captain **macpherson** looked after the bubbles cast his glance aloft at the bulging patches of white and then condescended to observe the agent of the **police** who had silently approached
sir **charles** and lady and sir **charles party** have expressed captain **macpherson** the desire to obtain permission to visit the prisoners deck
captain **macpherson** looked toward sir **charles** and his lady the other passengers lounging around them a little girl at the rail her hair blown windward a splash of go

The numbers of total entities that needs to be marked = 24

Total number of entities labelled by the model = 21

Number of the correctly labelled entities = 20

Precision (**p**) = Total correctly labelled entities by the total number of labeled entities

= 20 / 21 = 0.9523

Recall (**r**) = Total correctly labelled entities by the total entities that need to be labelled.

= 20 / 24 = 0.833

F- measure : F_1 score = 0.8886 .

: Evaluating the accuracy of the Named Entity Recognition of Book 2

```
s4 = nerT2[:3000]
s4
```

Here, nerT2 is the whole preprocessed text of Book 2 after removal of the headers and the footers and s4 is the first 3000 characters of Book 2 used for the Named Entity Recognition.

As we can see from the image entity_tag_passage_book2 is a list that shows the entities with their labels marked by the model itself . For example John Aylmer's is a name of a person so it has been labeled as PERSON and thirty seconds is a duration of time so it is labeled as TIME.

```
entity_tag_passage_book2 = []
lst4 = ner(s4)
for word in lst4.ents:
  x = word.text
  y = word.label_
  entity_tag_passage_book2.append([x,y])

entity_tag_passage_book2
```

```
[["john aylmer's", 'PERSON'],
 ['thirty seconds', 'TIME'],
 ['bab al sok', 'PERSON'],
 ['earl', 'GPE'],
 ['second', 'ORDINAL'],
 ['american', 'NORP'],
 ['atlantic', 'LOC'],
 ['moroccan', 'NORP'],
 ['dozen', 'CARDINAL'],
 ['first', 'ORDINAL'],
 ['american', 'NORP'],
 ['american', 'NORP'],
 ['cook', 'PERSON'],
 ['moghreb-al-aksa', 'ORG'],
 ['first', 'ORDINAL'],
 ['two', 'CARDINAL'],
 ['third', 'ORDINAL'],
 ['jew', 'ORG'],
 ['moslem', 'NORP'],
 ['christian', 'NORP'],
 ['one', 'CARDINAL'],
 ['two', 'CARDINAL']]
```

Total number of entities marked by the model is 22 in which 'bab al sok' is marked incorrectly because it is not a Person, it is a monument and 'Jew' is not an organisation, in this case it is NORP.

The text below represents the first 3000 characters of the text of Book 2 and in the figure all the Bold words represent the Entities marked manually.There are a total 28 entities that we have found in the passage and the Model shown above catches 22 entities.

Fig : The sample text of the Book 2 for the evaluation of accuracy

the lady of the pier

It was not the muleteers shove, slight but significant though it was, which produced **John Aylmer's** shrug of irritation. his resentment was directed at himself. He realized that he had been guilty of a gaucherie. for **thirty seconds** he had been standing halted in the main street of **tangier**, a rock of obstruction to all the rabble traffic which passes between the **bab al masha** and the **bab al sok**, staring at--what? at a pretty woman.

he reddened under his tan. The muleteers shoulder had displaced him for purely practical reasons, for, indeed, almost benevolent ones, for the mules would have been capable of obtaining with their teeth what their guardian had obtained by mere weight of his body. but aylmer felt that by accepted social standards a kick would not have been more than his due. had he not been behaving like some cub of a cockney clerk at an **earl's** court exhibition? his lips moved. He was muttering excuses of himself to himself, and knew that they were valid, but that an onlooker would have had no clue. for it was not her prettiness which had drawn his attention to the girl.it took no **second** glance to assure him that she was no countrywoman of his, but an **american**. Her features had the clean regularity, her complexion the pale, unfurrowed smoothness which is kept intact on the western side of the **atlantic** and there alone. The **moroccan** sunlight was proving in a **dozen** places the mistake the shadows made when they dulled the gold of her hair to brown. her eyes matched the waters of the unrippled bay. though he recognized these things, they had not, in the first place, attracted **Aylmer's** attention. **american** girls--pretty **american** girls--are no rarity in **tangier** since mr. **cook** threw over **maghreb-al-aqsa** the aegis of his protection. under ordinary circumstances he would have looked, approved, and, without altering his stride, passed on. but here was something which appealed to the inherited instincts of a gentleman. What was it? apprehension.

he felt no reasonable doubt on the subject. among this girl's natural attributes, he told himself, were placidity, content, self-reliance. The **first two** were wanting. the **third** was strained. there was almost a sense of furtiveness in the glances which she turned to throw not only about but, occasionally, behind her. frankly, she was afraid.

His interest fed upon observation. He glanced at her more narrowly, he observed her surroundings. He drew aside out of the mid-street traffic, and under pretence of lighting a **cigarette**, halted again in the shadow of an awning.

She was not alone. she held by the hand a small, alert-looking child--a boy, who watched the passers-by with the happy, unconcentrated interest of childhood. his eyes reviewed his surroundings without any of the surprise of unaccustomedness; obviously the scene was not strange to him. he smiled at **jew** and **moslem**, **christian** and **infidel**, with a pleasant patronage which **one** or **two** itinerant pedlars and

The numbers of total entities that needs to be marked = 28

Total number of entities labelled by the model = 22

Number of the correctly labelled entities = 20

Precision (**p**) = Total correctly labelled entities by the total labeled entities

= 20 / 22 = 0.909.

Recall (**r**) = Total correctly labelled entities by the total entities that need to be labelled.

= 20 / 28 = 0.714

F- measure : F_1 score = 0.799

Part 3. Extract the relationship between the entities (mainly the characters involved in the novel). Try to do this as much as possible

```
# load spaCy model
nlp = spacy.load("en_core_web_sm")


text =nerT1

# create a spaCy object
doc = nlp(text)
```

Spacy provides a Tokenizer, a POS-tagger and a Named Entity Recognizer and uses word embedding strategy that is fast and accurate. It uses the word embeddings to do the task. spaCy uses word embeddings for its NER model, which is a multilayer CNN.

We are loading the spacy model for english and then creating a doc object suitable for spacy libraries from our text

This is how doc looks like :

```
doc[:100]
```

```
mr gilletts charge


by all means mdeah lets go down between decks and have a look at
them

of course if you wish sir charles althoughdo you think we shall be
edified mr gillett

that depends mladyand the speaker a man with official manners
and ferretlike eyes shifted from one foot to anotheron what
degree or particular class of criminal your ladyship would be
interested in he added if in the ordinary category of skittle
sharper or thimblerigger with a suspicion of mild scorn then i do
not imagine
```

Pattern 1 :

```
pattern = [ {'ENT_TYPE':'PERSON'},


            {'LOWER': 'and','OP':"?"},
            {'POS':'DET'},
           {'POS':'NOUN'}]
```

We are defining a pattern which we will be extracting from the text. The pattern defined as : the first words should be an entity with type *'person'* followed by the text *'and'* and then followed by the *determine*r and ends with a *noun*.

```
matcher2 = Matcher(nlp.vocab)
matcher2.add("matching_1", None, pattern)

matches = matcher2(doc)
print(len(matches))
```

```
28
```

We used the matcher object from the spacy library to match sequences of tokens based on pattern rules.

After running the following code snippet we found 28 matches.

```
for i in range(27):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

```
macpherson the desire
charles and his lady
frisco the pride
ho the ship
joe and the police
mdear whose decisions
steele your crossexamination
wray whose estate
charles and his wife
steele the words
charles and his wife
steele her eyes
steele his face
interposed a man
wray his business
wray and some one
steele her look
gillett these men
gillett your lordship
steele a voice
steele her brows
divert her thoughts
dennis the man
steele a spark
joe a prodigal
steele the bloodshot
tom the man
```

The 28 relations are stored in the python object matches. For example: Charles and his lady is one of the output patterns which follows the given pattern rule 1, Another examples are frisco the pride , jo and the police and so on. These are showing the relationship between two entities.

Pattern 2 :

```
pattern = [ {'ENT_TYPE':'PERSON','OP':"?"},


            {'LOWER': 'and','OP':"?"},
            {'POS':'DET'},
           {'ENT_TYPE':'GPE'},
           {'POS':'NOUN','OP':"?"}]

matcher3 = Matcher(nlp.vocab)
matcher3.add("matching_1", None, pattern)

matches = matcher3(doc)
print(len(matches))

9
```

Again we are adding some new patterns like 'pos : NOUN , OP:? ' (here OP stands for optional) and running the entity match algorithm, and we found some 9 matches.

The 9 new matches are shown in the image shown aside.

```
for i in range(8):
   span = doc[matches[i][1]:matches[i][2]]
   print(span)

a london
all london
a shadowa
that wasi
the london
no blotter
your london
the mississippi
```

We found the country names which are being preceded by the Determiners for example 'a london' , 'the london' and so on.We found out in the text used there no relationship of a person with the GPE.

Pattern 3:

```
pattern = [

            {'ENT_TYPE':'PERSON'},
            {'POS':'DET','OP':"?"},
            {'LOWER': 'has', 'OP':"?"},
            {'LOWER': 'lives', 'OP':"?"},
            {'ENT_TYPE':'GPE','OP':"?"},
            {'ENT_TYPE':'ORG','OP':"?"},
            {'LOWER': 'on', 'OP':"?"},
            {'POS':'NOUN'}]
```

Now again, we used some new patterns to obtain some more relationship between entities

```
#matcher.add("matching_1", None, pattern)
matcher5 = Matcher(nlp.vocab)
matcher5.add("matching_1", None, pattern)

#matches = matcher(doc)
matches = matcher5(doc)
print(len(matches))
```

```
68
```

After using the above pattern in the Matcher object we obtained 68 new relationships.

```
steeles
steele her eyes
saccharine sweetness
crimson others
steeles coat
steele his face
steeles breast
wrays eyes
interposed a man
gilletts manner
wray his business
steeles library
afterwardwastrel nomad
himto step
pon honor
steeles face
charles words
norman period
charles pipe
charles exclamation
steele her look
steeles arm
gillett these men
gillett your lordship
steele a voice
steele her brows
dingy metropolis
divert her thoughts
george gold
steeles excuses
steeles room
rosemary villa
dennis the man
steeles pulses
steele notice
```

Those **68 relationships** are shown in the image below . For example: '**Steele her eyes**' , 'interposed a man' , 'wrays eyes',' crimson others' and so on.

We found the relations where the **PERSON** entity is followed by **det/noun.**

Pattern 4 :

```
3# Matcher class object
matcher2 = Matcher(nlp.vocab)

#define the pattern
pattern = [{'ENT_TYPE':'PERSON'},
           {'POS':'NOUN','OP':"?"},
           {'POS':'NOUN','OP':"?"},

            {'POS':'DET','OP':"?"},
           {'POS':'VERB'}
           ]

matcher2.add("matching_1", None, pattern)

matches = matcher2(doc)
print(len(matches))
```

```
377
```

Now again we defined some more patterns and used them in the matcher object and ran the algorithm . We got 377 matches.

Those 30 matches are shown in this image . **We observed that the noun is followed by a verb** . For example: 'Gillett paused' , 'Charles assisted' , 'Charles laughed' , 'Nelson new' and so on.

```
for i in range(30):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

```
nelson plowed
macpherson looked
macpherson looked
macpherson frowned
nelson knew
charles addressed
gillett tell
dick turpin
voice rose
gillett paused
charles assisted
charles laughed
laughed got
jocelyn the expostulating
gillett seized
nelson minded
macpherson staggered
macpherson recovered
macpherson sends
charles expostulated
nelson fell
obrien get
macpherson waited
sam killed
ho ho
nelson followed
steele conduct
steele comes
jocelyn returned
you gazing
```

Pattern 5:

```
#define the pattern
pattern = [

          {'POS':'NOUN'},
          {'LOWER': 'agent', 'OP':"?"},
          {'LOWER': 'of'},
          {'POS':'DET'},
          {'POS': 'NOUN'}]
```

Now again we defined some new patterns and used it in similar fashion to generate some more new matches .

We created a matcher object.

We will match the given pattern with our book text to find relations

```
# Matcher class object
matcher = Matcher(nlp.vocab)
matcher.add("matching_1", None, pattern)

matches = matcher(doc)
```

```
len(matches)
```

```
300
```

.

```
for i in range(20):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

We are first printing 20 such matches.

```
agent of the police
center of the ship
characteristics of his vessel
agent of the police
exercise of that ingenuity
experience of the world
face of the police
case of the rum
members of his party
closeness of the air
beating of the waves
stillness of the place
pity of the spectacle
face of the convict
voice of the police
tones of the governors
approach of the party
expression of his iron
seething of the sea
cursing of the convicts
```

Pattern 6:

```
#define the pattern
pattern = [
           {'POS':'NOUN'},
           {'LOWER': 'and'},


           {'POS': 'NOUN'}]

matcher1 = Matcher(nlp.vocab)
matcher1.add("matching_1", None, pattern)

matches = matcher1(doc)
print(len(matches))
```

```
122
```

Now we will look for some other patterns

[NOUN] and [NOUN]

We found 122 such matches

```
for i in range(30):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

```
weight and importance
theory and practice
stars and stripes
sodden and brutish
turmoil and uproar
water and provision
wraps and greatcoats
spirits and liqueurs
courage and power
energy and effort
darkness and confusion
anger and hatred
psychiatry and anthropology
witness and barristers
day and one
intelligence and intensity
ship and turtle
papers and letters
dukes and earls
trills and pizzicatos
rank and condition
wife and party
knowledge and care
pros and cons
family and hers
manner and others
dear and sir
poets and historians
sunshine and air
flowers and sips
```

30 such matching phrases are as follow

Reading processes test of book 2

```
text =nerT2

# create a spaCy object
doc = nlp(text)
```

```
doc[:300]
```

```
THE LADY OF THE PIER


It was not the muleteer s shove  slight but significant though it was
which produced John Aylmer s shrug of irritation  His resentment was
directed at himself  He realized that he had been guilty of a gaucherie
For thirty seconds he had been standing halted in the main street of
Tangier  a rock of obstruction to all the rabble traffic which passes
between the Bab al Marsa and the Bab al Sok  staring at  what

At a pretty woman

He reddened under his tan  The muleteer s shoulder had displaced him for
purely practical reasons  for  indeed  almost benevolent ones  for the
mules would have been capable of obtaining with their teeth what their
guardian had obtained by mere weight of his body  But Aylmer felt that
by accepted social standards a kick would not have been more than his
due  Had he not been behaving like some cub of a cockney clerk at an
Earl s Court Exhibition  His lips moved  He was muttering excuses of
himself to himself  and knew that they were valid  but that an onlooker
would have had no clue to them

For it was not her prettiness which had drawn his attention to the girl
It took no second glance to assure him that she was no countrywoman of
his  but an American  Her features had the clean regularity  her
complexion the pale  unfurrowed smoothness which is kept intact on the
western side of the Atlantic and there alone  The Moroccan sunlight was
proving
```

Now we are going to perform similar actions for fetching the relationships from book 2.

Pattern 1 :

```
pattern = [ {'ENT_TYPE':'PERSON'},

            {'LOWER': 'and','OP':"?"},
            {'POS':'DET'},
           {'POS':'NOUN'}]
```

```
matcherT2 = Matcher(nlp.vocab)
matcherT2.add("matching_1", None, pattern)

matches = matcherT2(doc)
print(len(matches))
```

```
13
```

13 matching phrases were found after running the algorithm.

```
for i in range(13):
    span = doc[matches[i][1]:matches[i][2]]
    print(span)
```

```
Aksa the aegis
Aylmer and his companion
Awara and the camp
Absalaam and the horses
Sidi every reason


 The general
  The question


 A matter
Speak your part
Perinaud and his escort
  The bill
Doubtless another half
Arlen and her companion
```

The matching patterns are shown in the image. Some of the relations are 'aksa the aegis', 'Aylmer and his companion' .

Pattern 2 :

```
pattern = [ {'ENT_TYPE':'PERSON','OP':"?"},


            {'LOWER': 'and','OP':"?"},
            {'POS':'DET'},
           {'ENT_TYPE':'GPE'},
           {'POS':'NOUN','OP':"?"}]

matcherT3 = Matcher(nlp.vocab)
matcherT3.add("matching_1", None, pattern)

matches = matcherT3(doc)
print(len(matches))
```

```
19
```

19 matching patterns found

```
for i in range(19):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

```
a Briton
that New
the Villa
the Waterport
the Waterport
the States
the Waterport
the morocco
the Moroquin
the Moroquin coast


 The Van
The Van
the Villa
which Daoud
a Hercules
the mast
The
A Newcastle
A Newcastle collier
```

The matching patterns are shown in the image . Some of the relations are 'a Briton' , 'the waterport' , 'the states' , 'the morocco'. **We are observing that a determiner is followed by a noun entity.**

Pattern 3 :

```
pattern = [

            {'ENT_TYPE':'PERSON'},
            {'POS':'DET','OP':"?"},
            {'LOWER': 'has', 'OP':"?"},
            {'LOWER': 'lives', 'OP':"?"},
            {'ENT_TYPE':'GPE','OP':"?"},
            {'ENT_TYPE':'ORG','OP':"?"},
            {'LOWER': 'on', 'OP':"?"},
            {'POS':'NOUN'}]
```

```
#matcher.add("matching_1", None, pattern)
matcherT5 = Matcher(nlp.vocab)
matcherT5.add("matching_1", None, pattern)


matches = matcherT5(doc)
print(len(matches))
```

```
49
```

49 matching phrases were found.

The matching patterns are shown in the given image

```
 The general
Mocking cries
mes enfants
le troisieme
Monsieur has reason
  The question
s face
Berber tribesmen
s return


 A matter
Aylmer s
Mademoiselle forgets
Speak your part
s contraband
Lipari group
  The bill
Mequinez steel
Whispered voices
Muhammed doesn
Doubtless another half
s family
Aylmer forbore
s cousin
Isn t
lights dusky
Moored boats
s hair
Claire knelt
Isn t
Melilla port
s eyes
s face
Claire s
```

Pattern 4 :

```
3# Matcher class object
matcherT6 = Matcher(nlp.vocab)

#define the pattern
pattern = [{'ENT_TYPE':'PERSON'},
           {'POS':'NOUN','OP':"?"},
           {'POS':'NOUN','OP':"?"},

            {'POS':'DET','OP':"?"},
           {'POS':'VERB'},
           {'POS':'NOUN','OP':"?"}
           ]

matcherT6.add("matching_1", None, pattern)

matches = matcherT6(doc)
print(len(matches))
```

```
409
```

There are 409 matching phrases that we obtained after running the algorithm.

Some of the matching phrases are shown in the given image: For example ,'Aylmer felt' , 'Cook threw' , 'John wriggled' . **We observed that a noun is followed by the verb**

```
for i in range(30):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

```
Aylmer felt
Cook threw
Aylmer saw
Jan encourages
Jan snatches
John wriggled
Aylmer turned
Thou hast
Aylmer throw
Selim will
Selim jumping
John will
Aylmer tightened
Rattier looked
Aylmer smiled
Aylmer dropped
Rattier lifted
Aylmer reined
Anstruther sends
Anstruther sends breakfast
Aylmer slid
Sidis will
Aylmer spurred
Aylmer maintained
Aylmer told
Aylmer swerved
Aylmer felt
Aylmer looked
crimson smear
jaw must
```

Pattern 5:

```
matcherT7 = Matcher(nlp.vocab)

#define the pattern
pattern = [ {'POS':'PRON'},
           {'POS':'NOUN'},

            {'POS':'DET','OP':"?"},
           {'POS':'VERB'},
        {'POS':'NOUN','OP':"?"},

           ]

matcherT7.add("matching_1", None, pattern)

matches = matcherT7(doc)
len(matches)
```

```
9
```

```
for i in range(len(matches)):
  span = doc[matches[i][1]:matches[i][2]]
  print(span)
```

```
I m getting
he d come
you purpose going
s ear served
I m talking
you d call
I m going
I m coming
I m hurted
```

There are 9 matching phrases

We can see that the matches we obtained are in the form 'I am getting ' , 'I am talking' and so on.

References:

- https://spacy.io/api/matcher
- https://www.nltk.org/
- https://web.stanford.edu/~jurafsky/slp3/
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
- https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d
- Class Video Lectures and Slides