NLP PROJECT

ROUND-1 REPORT

TEAM NAME : **Nlp_Learners**

Team Members :MOHIT JAIN (19UCS050)
AYUSH KABRA (19UCS113)
ARYAN VYAS (19UCS117)

Github repository:
https://github.com/Aryan-Vyas/NlpLearners_NlpProject

Books used :

1. A town is drowning….By Frederik Pohl : T1
2. Nick Carter Stories….By Nick Carter : T2

Objectives :

1. Import the text from two books, let's call it as T1 and T2.
2. Perform simple text pre-processing steps and tokenize the text T1 and T2.
3. Analyze the frequency distribution of tokens in T1 and T2 separately.
4. Create a Word Cloud of T1 and T2 using the token that you have got.
5. Remove the stopwords from T1 and T2 and then again create a word cloud.
6. Compare with word clouds before the removal of stopwords.
7. Evaluate the relationship between the word length and frequency for both T1 and T2.
8. 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

Python Libraries used in this project :

NLTK : Used for Tokenizing, Lemmatization and Removing Stopwords .
Re : Used to remove URLs and Decontract Contractions in English Language .
Wordcloud : Used to create WordClouds from Tokenized Data .
Matplotlib : Used to Visualize our text data .

Importing the texts from the books :

```
In [2]: f1 = open("E:/T1.txt",'r')
        f2 = open("E:/T2.txt",'r')
```

```
t1 = f1.read()
```

```
t2 = f2.read()
```

DATA PREPROCESSING STEPS

1. Removing not needed text of Book : We will remove the documentation part of the text that is of no use to us.
2. Remove chapter name : We will remove chapter names as given in the task.
3. Convert all data to lowercase : We will convert all text data to lowercase.
4. Remove link : We will remove urls using regular expression .
5. Remove Punctuations .
6. Lemmatization : We do Lemmatization with the help of WordNetLemmatizer() function .

Functions :

```
def remove_not_needed_text(t):
    start = t.find('*** START OF THE PROJECT ')
    end = t.find('*** END OF THE PROJECT ')
    t = t[start:end]
    return t
```

```
def remove_chap(t):
    t=re.sub(r"CHAPTER [A-Z]+","",t)
    return t
```

```
def to_lowercase(t):
  return t.lower()
```

```
def remove_apos(t):
    t = re.sub(r"won\'t", "will not", t)
    t = re.sub(r"can\'t", "can not", t)
    t = re.sub(r"don\'t", "do not", t)

    t = re.sub(r"\'s", " is", t)
    t = re.sub(r"\'ll", " will", t)
    t = re.sub(r"\'ve", " have", t)
    t = re.sub(r"\'m", " am", t)
    t = re.sub(r"n\'t", " not", t)
    t = re.sub(r"\'re", " are", t)

    return t
```

```
def remove_link(t):
    return re.sub(r"http\S+", "", t)
```

```
def remove_punc_num(t):
    t = re.sub("[^\w\s]","",t)
    t = re.sub("[0-9]","",t)
    return t
```

```python
def tokenize(t):
    token = nltk.word_tokenize(t)
    return token
```

```python
def lemmatize_word(token):
    lemmatizer = WordNetLemmatizer()
    token1 = []
    for word in token:
        token1.append(lemmatizer.lemmatize(word))
    return token1
```

Data Preparation

We apply all the functionalities we added above , tokenize our data and prepare our data for analysis:

For T1

In [17]:
```python
t1 = f1.read()
```

In [18]:
```python
t1 = remove_not_needed_text(t1)
```

In [19]:
```python
t1 = remove_chap(t1)
```

In [20]:
```python
t1 = to_lowercase(t1)
```

In [21]:
```python
t1 = remove_apos(t1)
```

In [22]:
```python
t1 = remove_link(t1)
```

In [23]:
```python
t1 = remove_punc_num(t1)
```

In [24]:
```python
token = tokenize(t1)
```

In [25]:
```python
token = lemmatize_word(token)
```

For T2

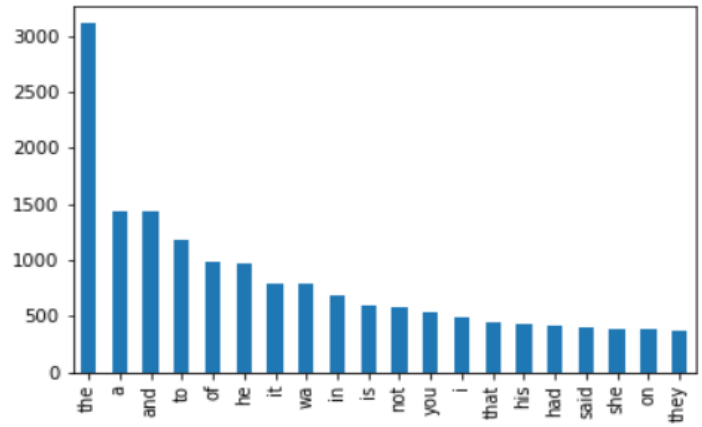
```
In [33]: t2 = f2.read()

In [34]: t2 = remove_not_needed_text(t2)

In [35]: t2 = remove_chap(t2)

In [36]: t2 = to_lowercase(t2)

In [37]: t2 = remove_apos(t2)

In [38]: t2 = remove_link(t2)

In [39]: t2 = remove_punc_num(t2)

In [40]: token = tokenize(t2)

In [41]: token = lemmatize_word(token)
```

Problem Statements and Inferences

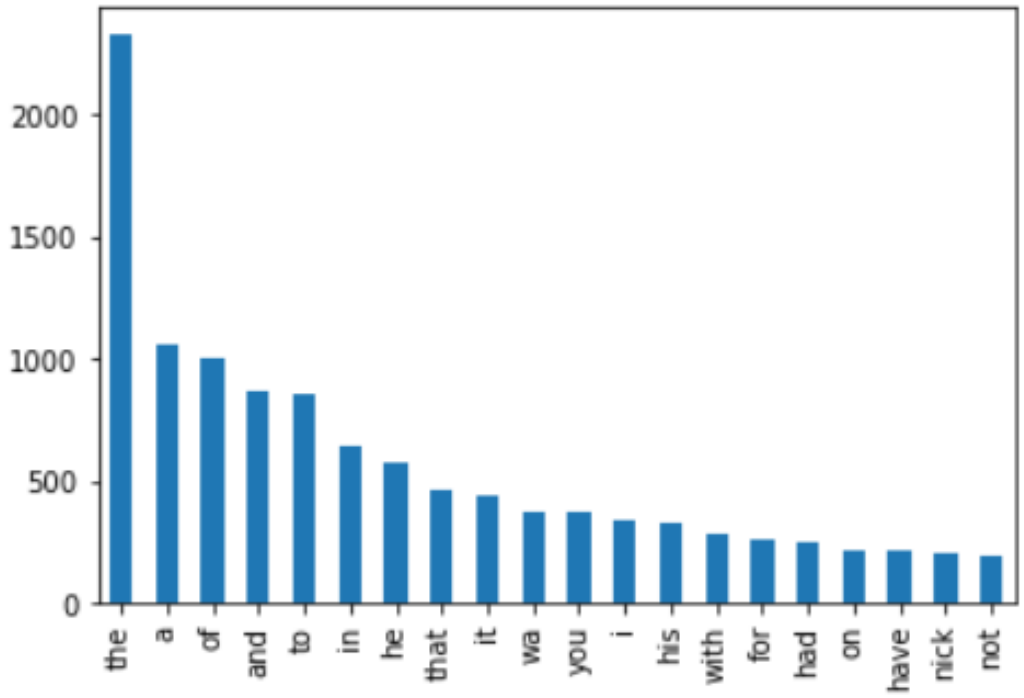
- Analyze the frequency distribution of tokens in T1 and T2 separately : We use python library pandas . First we tokenize the given data, and then we plot a bar graph of most frequent words .

```
def freqdist(token):
    fdist = nltk.FreqDist(token)
    return fdist
```

For T1 :



For T2 :

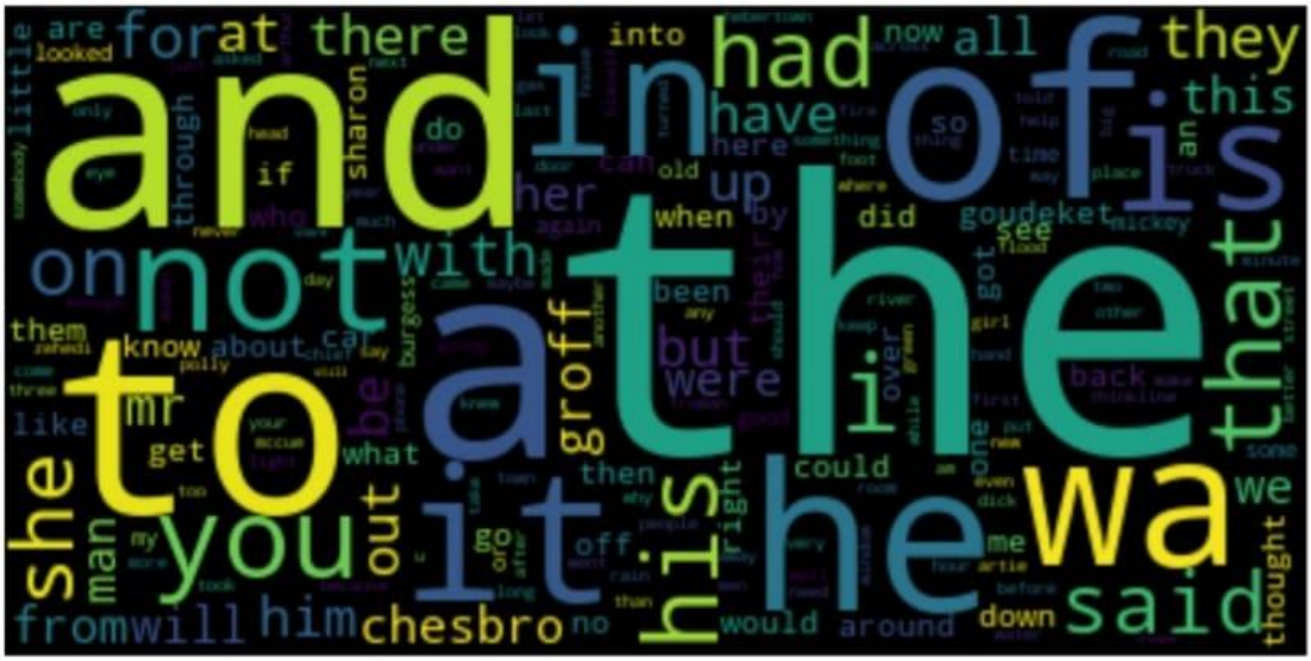


Generating a word cloud from T1 and T2 :

For this we use python library wordcloud and its function WordCloud.

```
def wordcloud(fdist):
    wordcloud = WordCloud()
    wordcloud.generate_from_frequencies(frequencies=fdist)
    plt.figure(figsize=(20,20))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

Wordcloud for T1 :



Wordcloud for T2 :

Inferences :

- Words like ‘and’, ‘a’, ‘of’, ‘to’ and ‘the’ are the most frequently used words in T1 .
- Words like ‘and’, ‘the’, ‘he’, ‘to’’ are frequently used words in T2 .
- The words seen in wordcloud are called stop words .

Generating new word clouds after removing stopwords :

To remove stopwords, we use STOPWORDS function in nltk.

```
def wordcloud_withoutstop(t):
    wc1 = WordCloud(
          height = 1000,
          width = 1000 ,
          stopwords = STOPWORDS,
          collocations = False,
          background_color='white'
         )
    wc1.generate(t)
    plt.figure(figsize=(15,15))
    plt.imshow(wc1)
```

Wordcloud for T1 after removing stopwords :

Wordcloud for T2 after removing stopwords :

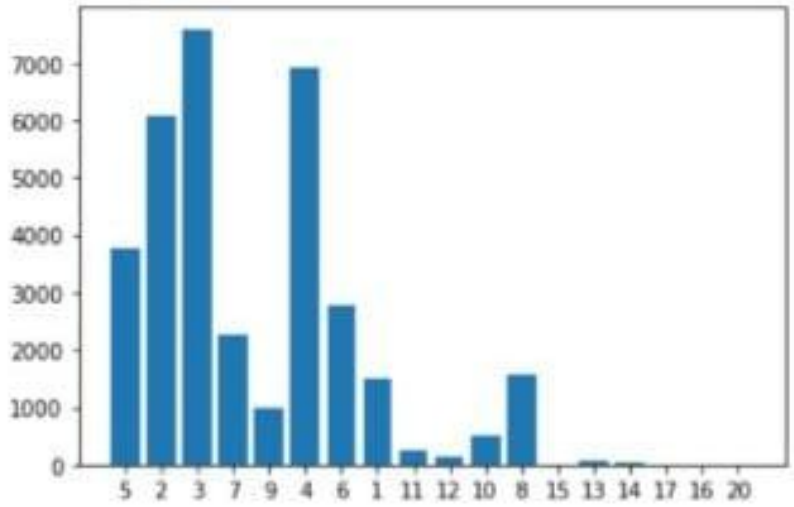
Inferences :

- Now the stopwords are removed .
- New words like ‘sharon’, ‘groff’ in T1 and ‘nick’, ‘carter’, ‘floyd’, etc. can be seen in wordcloud which indicates the name of characters and other important stuffs around whom the book revolves.
- Therefore, after removing stopwords we get more meaning and understanding of the book.

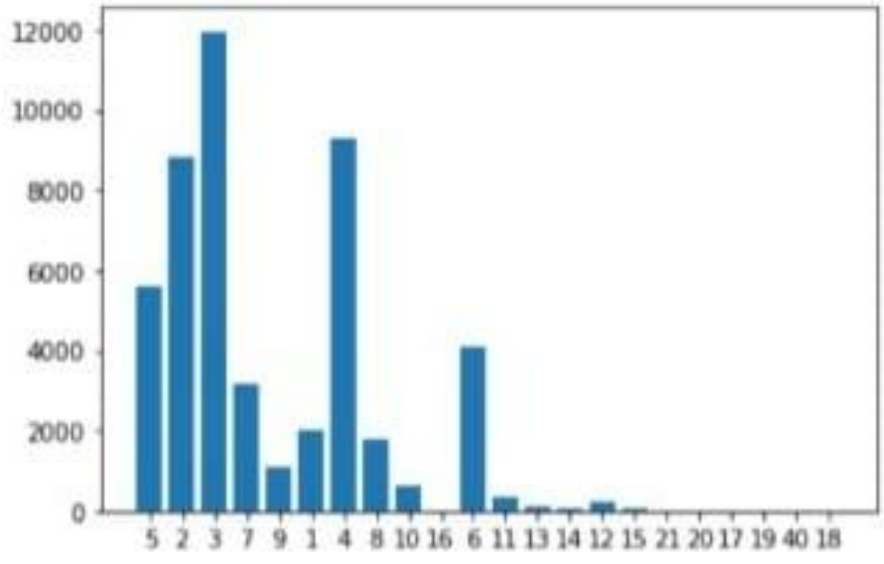
Evaluating the relationship between the word length and frequency for both T1 and T2 :

```
def freq_wrt_len(dataf):
    dict1 = {}
    for i in range(0,len(dataf.index)):
        if len(dataf.iloc[i,0]) not in dict1:
            dict1[len(dataf.iloc[i,0])] = dataf.iloc[i,1]
        else:
            dict1[len(dataf.iloc[i,0])] += dataf.iloc[i,1]
    print(dict1)
    length = list(dict1.keys())
    freq = list(dict1.values())
    plt.bar(range(len(dict1)),freq,tick_label=length)
    plt.show()
```

For T1 :


7000
6000
5000
4000
3000
2000
1000
0
5 2 3 7 9 4 6 1 11 12 10 8 15 13 14 17 16 20

For T2 :


12000
10000
8000
6000
4000
2000
0
5 2 3 7 9 1 4 8 10 16 6 11 13 14 12 15 21 20 17 19 40 18

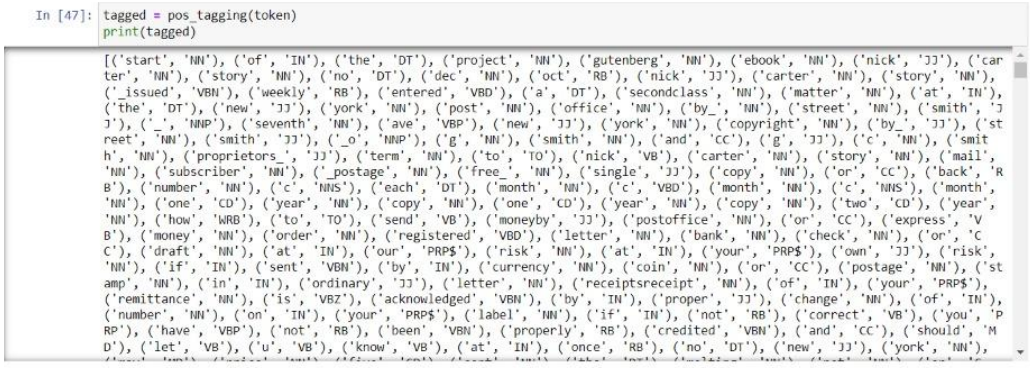
POS Tagging :

We will now perform the POS Tagging on T1 and T2 using inbuilt functions of nltk namely post_tag() which uses Penn Treebank tag set to perform POS tagging.

```
def pos_tagging(token):
    tagged = nltk.pos_tag(token)
    return tagged
```

For T1 :

For T2 :

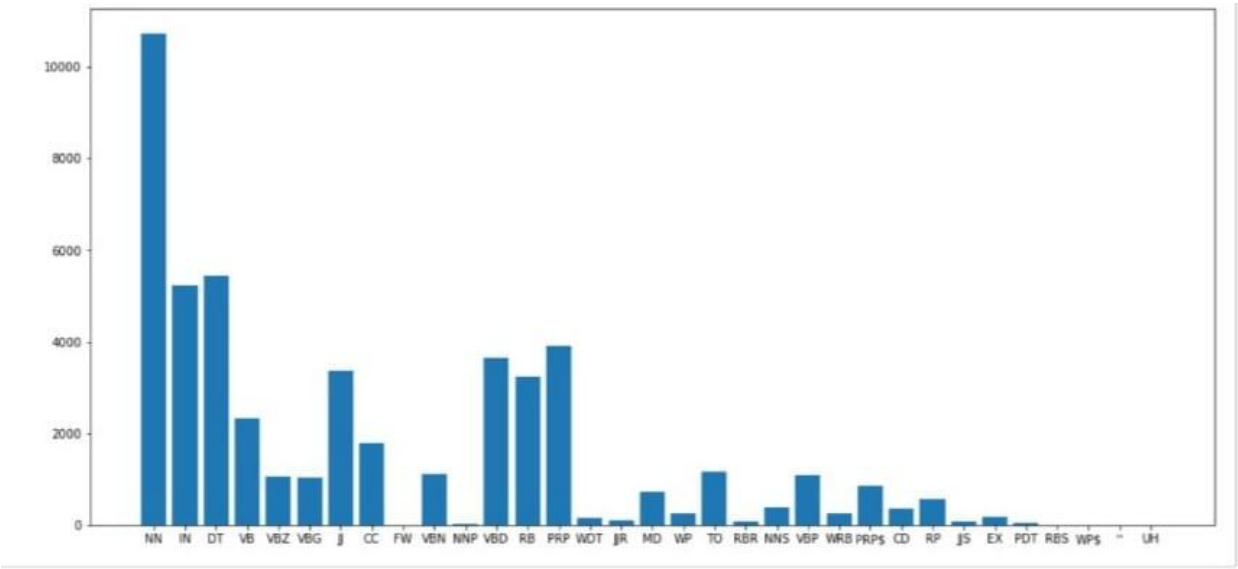


Frequency Distribution of Tags :

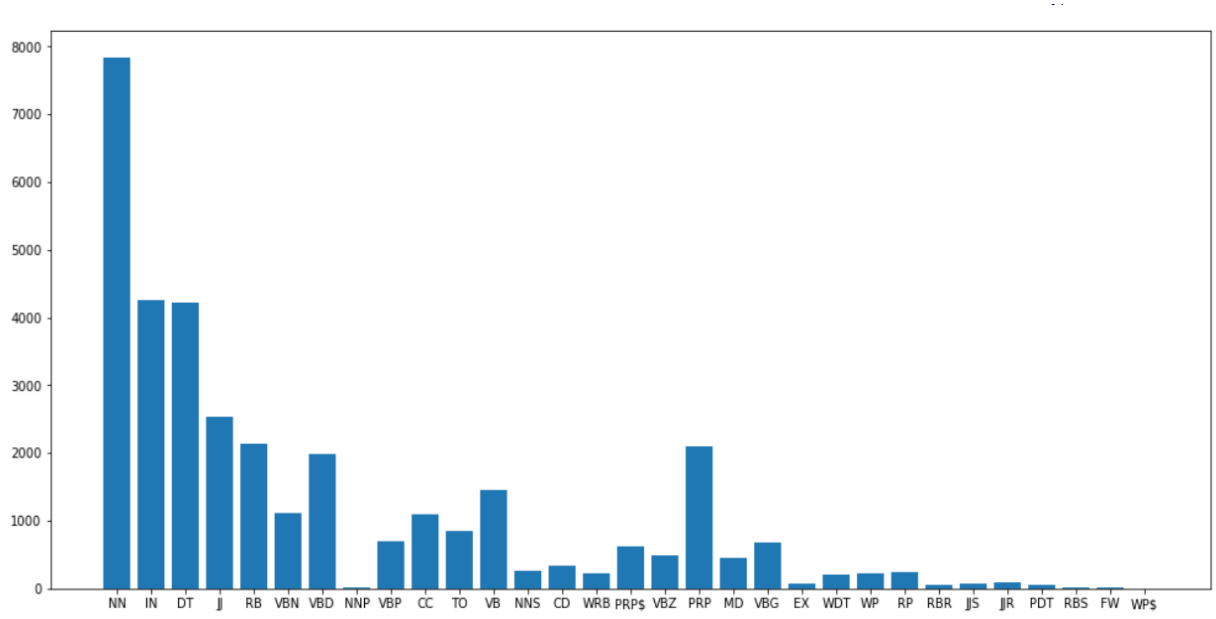
Now, we plot the frequency distribution of tags after POS tagging on T1 and T2. For this we take the help of Counter() function from collections python library and FreqDist() function from nltk python library .

```
def tagwithfreq(tagged):
    from collections import Counter
    the_count = Counter(tag for _, tag in tagged)
    print(the_count)
    tag = list(the_count.keys())
    freq = list(the_count.values())
    plt.figure(figsize=(17, 8))
    plt.bar(range(len(the_count)),freq,tick_label=tag)
    plt.show()
```

FOR T1 :



FOR T2 :



Inferences :

From the above results we infer that the highest occurring tag is ‘NN’, and ‘Determinant’ Tags are on the lower frequency side. This is largely due to the removal of stopwords before POS Tagging.

Project Round - 2

Books used :

1. A town is drowning….By Frederik Pohl : B1
2. Nick Carter Stories….By Nick Carter : B2
3. Upside Down or Backwards… By W. C. Tuttle: B3

Objectives :

- Import the text from two books, let's call it as B1 and B2. Perform simple text pre-processing steps and tokenize the text B1 and B2.
- Find the nouns and verbs in both the novels. Get the immediate categories (parent) that these words fall under in the WordNet.
- 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.
- 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 entity .
    2. recognise all entity types.

  Use performance measures to measure the performance of the method used, use random samples to measure the performance.
- Import the text from three books, let's call it as B1 , B2 and B3 .
- Create TF-IDF vectors for all books and find the cosine similarity between each of them and find which two books are more similar.

- Do lemmatization of the books and recreate the TF-IDF vectors for all the books and find the cosine similarity of each pair of books.

Python Libraries used :

NLTK : Used for Tokenizing, implementing wordnet,POS tagging,etc.
Spacy : To perform Entity recognition in text .
Numpy : To get frequency distributions of nouns and verbs .
Matplotlib : Used to Visualize our text data .
Re : Used to remove URLs and Decontract Contractions in English Language .
Typing : To perform evaluation of the Algorithm in Entity Recognition.
Sklearn : Used for machine learning and statistical modeling .
Inflect : Correctly generate plurals, singular nouns, ordinals , indefinite articles; convert numbers to words.
Pandas : Offers data structures and operations for manipulating numerical tables and time series .

Problem Statement And Inferences :

- ☐ **Finding nouns and verbs**
    - **Preprocessing of B1 and B2 :**

In [15]:
```
##Preprocessing on b1
```

In [16]:
```
b1 = f1.read()
b1 = remove_not_needed_text(b1)
b1 = remove_chap(b1)
b1 = to_lowercase(b1)
b1 = remove_apos(b1)
b1 = remove_link(b1)
b1 = remove_punc_num(b1)
b1 = lemmatize_word(b1)
```

In [17]:
```
##preprocessing on b2
```

In [18]:
```
b2 = f2.read()
b2 = remove_not_needed_text(b2)
b2 = remove_chap(b2)
b2 = to_lowercase(b2)
b2 = remove_apos(b2)
b2 = remove_link(b2)
b2 = remove_punc_num(b2)
b2 = lemmatize_word(b2)
```

- **Performing POS Tagging :**
  Now we'll perform the POS Tagging on T1 and T2 using inbuilt functions of nltk namely pos_tag() which uses Penn Treebank tag set to perform POS tagging. We will extract the words which are tagged explicitly as nouns and verbs separately from both the novels using the following functions .

In [19]:
```
def noun(text):
  is_noun = lambda pos: pos[:1] == 'N'
  tokenized = nltk.word_tokenize(text)
  nouns = [word for (word, pos) in nltk.pos_tag(tokenized) if is_noun(pos)]
  return nouns
```

In [20]:
```
noun1=noun(b1)
noun2=noun(b2)
```

In [22]:
```
def verb(text):
  is_verb = lambda pos: pos[:1] == 'V'
  tokenized = nltk.word_tokenize(text)
  verbs = [word for (word, pos) in nltk.pos_tag(tokenized) if is_verb(pos)]
  return verbs
```

In [23]:
```
verb1=verb(b1)
verb2=verb(b2)
```

We will apply the above functions to both B1 and B2 respectively and print the total nouns and verbs in both of them .

```
In [21]: print("Number of nouns in book 1 and book 2 respectively are "+ str(len(noun1))+" and "+ str(len(noun2)))
```

```
Number of nouns in book 1 and book 2 respectively are 11653 and 8625
```

```
In [24]: print("Number of verbs in book 1 and book 2 respectively are "+ str(len(verb1))+" and "+ str(len(verb2)))
```

```
Number of verbs in book 1 and book 2 respectively are 9445 and 5683
```

☐ Get the categories that these words fall under in the WordNet.

To retrieve the categories that each noun and verb belong to, in the wordnet synsets, we have used nltk.corpus.wordnet as it has all the tools required for this task. We have used the following function to extract categories each noun and verb belongs to. Since a noun also has synsets interpretations as verbs and vice versa. hence we have included them as lists corresponding to its index in the noun and verb lists respectively.

```
In [25]: from nltk.corpus import wordnet as wn
```

```
In [26]: def synset(words):
           categories=[]
           for w in words:
             cat=[]
             for synset in wn.synsets(w):
               if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
                 cat.append(synset.lexname())
               if('verb' in synset.lexname()):
                 cat.append(synset.lexname())
             categories.append(cat)
           return categories
```

We will apply the above function to get 2-D lists which contain the categories that each noun and verb have been defined in the wordnet database.

```
In [27]: noun_synset1=synset(noun1)
         noun_synset2=synset(noun2)
         verb_synset1=synset(verb1)
         verb_synset2=synset(verb2)
```

Hence noun_synset1, noun_synset2, verbsynset_1,verb_synset2 are 2 - Dimensional lists which contain the categories that noun1 , noun2 , verb1 , verb2 belong to in the wordnet synsets of nouns and verbs.

The 2D lists are indexed as noun_synset1[x][y] where x is the index of corresponding noun in noun1 and y is the index containing the categories it belongs to.

Example :

```
In [28]: print(noun1[58])
         persons
```

```
In [29]: print(noun_synset1[58][:])
         ['noun.body', 'noun.communication']
```

- [ ] **Get the frequency of each category for each noun and verb in their corresponding and plot histogram / bar plots for each corresponding categories.**

To get the frequency of all the categories of nouns and verbs in the novels, we have created a set for each novel which contains all the types of nouns and verbs occurring and then plotting the frequency distribution for the data.

```
In [30]: def all_synsets(no,ve):
           nouns=[]
           verbs=[]
           for word in no:
             for synset in wn.synsets(word):
               if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
                 nouns.append(synset.lexname())
               if('verb' in synset.lexname()):
                 verbs.append(synset.lexname())
           for word in ve:
             for synset in wn.synsets(word):
               if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
                 nouns.append(synset.lexname())
               if('verb' in synset.lexname()):
                 verbs.append(synset.lexname())
           return nouns,verbs
```

Here , noun_superset1 contains all the different categories of nouns in novel 1 and verb_superset1 contains all the categories of verbs in novel 1 (T1) .

Example :

```
In [32]: print(noun_superset1)
['noun.event', 'noun.time', 'noun.act', 'noun.act', 'noun.act', 'noun.location', 'noun.communication', 'noun.attribute', 'nou
n.act', 'noun.cognition', 'noun.person', 'noun.location', 'noun.group', 'noun.location', 'noun.person', 'noun.location', 'nou
n.group', 'noun.location', 'noun.person', 'noun.quantity', 'noun.attribute', 'noun.quantity', 'noun.quantity', 'noun.quantit
```

```
In [33]: len(noun_superset1)
Out[33]: 75171
```

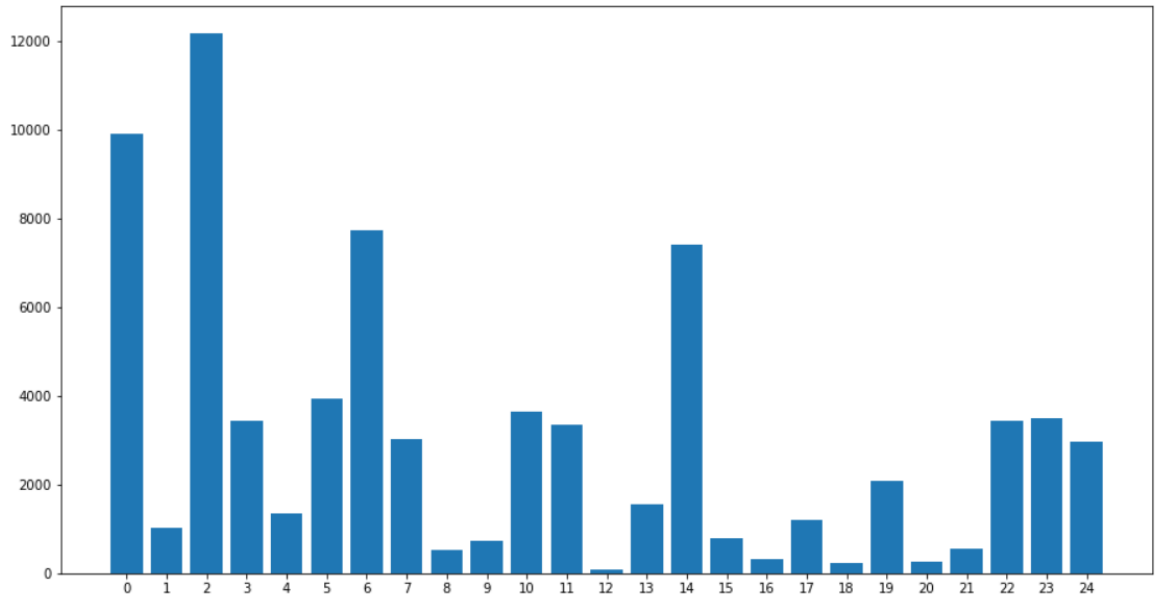
Hence there are 75171 elements in the list.

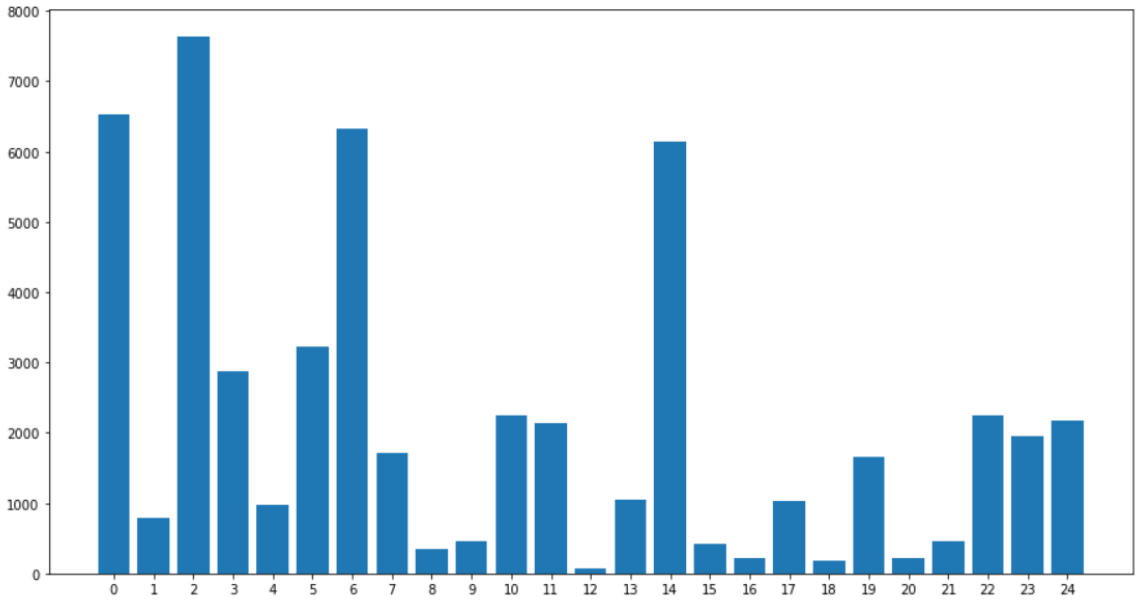
Plotting the histograms :

We have used numpy to count the frequency of each type of noun and verb out of 25 categories of nouns and 15 categories of verbs from both the novels.

```
In [34]: import numpy as np
labels, counts = np.unique(noun_superset1,return_counts=True)
import matplotlib.pyplot as plt
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
labels, counts = np.unique(noun_superset2,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
```

After plotting the graphs , we get the following plots where Y axis is counts and x axis are categories :





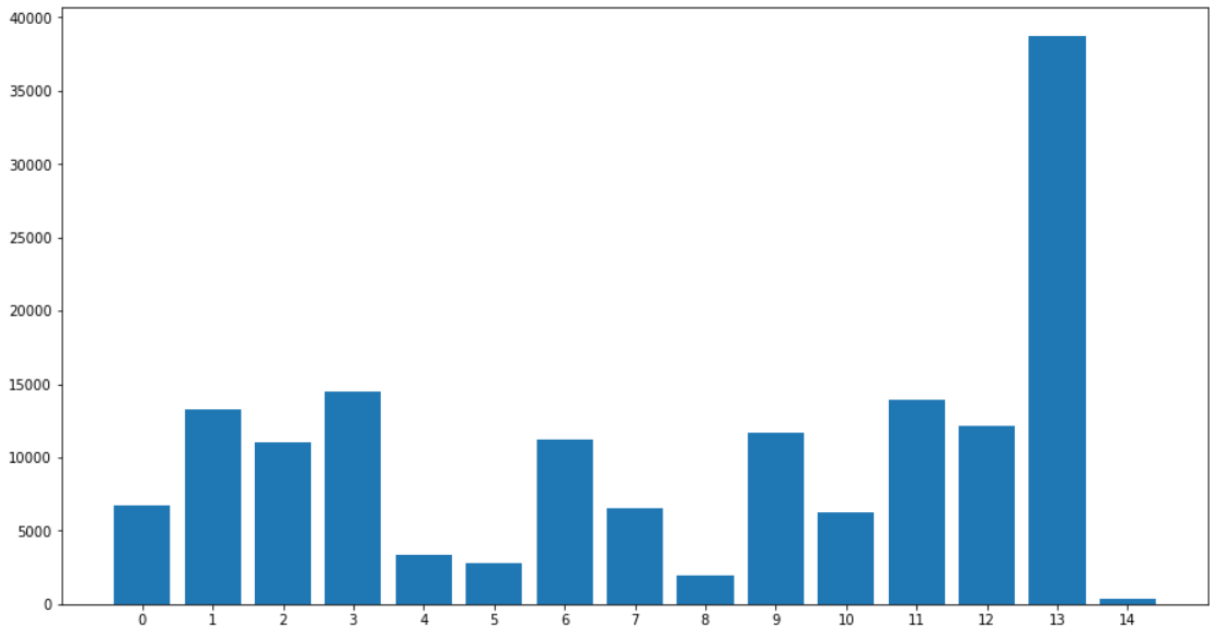
The categories are numbered as 0-24 in the order:

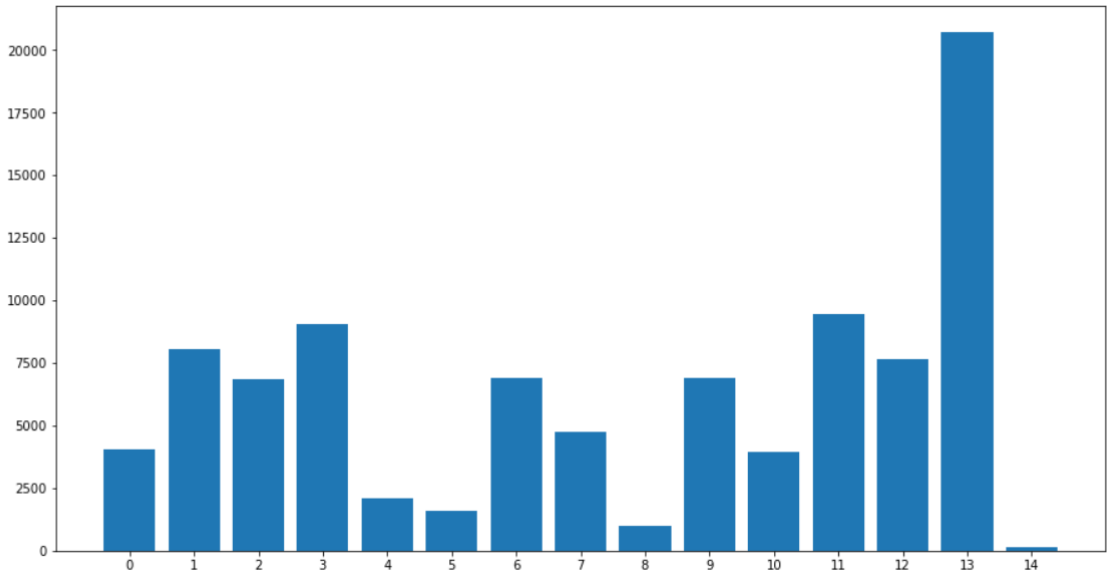
```
In [35]: print(labels)

['noun.act' 'noun.animal' 'noun.artifact' 'noun.attribute' 'noun.body'
 'noun.cognition' 'noun.communication' 'noun.event' 'noun.feeling'
 'noun.food' 'noun.group' 'noun.location' 'noun.motive' 'noun.object'
 'noun.person' 'noun.phenomenon' 'noun.plant' 'noun.possession'
 'noun.process' 'noun.quantity' 'noun.relation' 'noun.shape' 'noun.state'
 'noun.substance' 'noun.time']
```

Here labels are arranged in the order of hierarchy as given in wordnet categories of nouns.

Similarly we get the following **plots for Verbs** :

In [36]:
```
labels, counts = np.unique(verb_superset1,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
labels, counts = np.unique(verb_superset2,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
```





☐ Named Entity Recognition :

- Get the entities involved in each of the novels.

  To perform Named Entity Recognition in both the novels we have used *Spacy* . The Spacy uses token-level entity annotation using the BILUO tagging scheme to describe the entity boundaries.

  First we import spacy and get the entities involved in both the novels using the methods described in the library.

In [19]:
```
import spacy
from spacy import displacy
from collections import Counter
```

In [20]:
```
import en_core_web_sm
nlp = en_core_web_sm.load()
```

In [40]:
```
d1 = nlp(b1)
d2 = nlp(b2)
print("There are total "+str(len(d1.ents))+" entities in Book 1 and "+str(len(d2.ents))+" in Book 2")
```

```
There are total 838 entities in Book 1 and 777 in Book 2
```

```
In [41]: print([(X, X.ent_iob_) for X in d1])
```

```
[(start, 'O'), (of, 'O'), (the, 'O'), (project, 'O'), (gutenberg, 'O'), (ebook, 'O'), (a, 'O'), (town, 'O'), (be, 'O'), (drow
n, 'O'), (a, 'O'), (town, 'O'), (be, 'O'), (drown, 'O'), (by, 'O'), (frederik, 'O'), (pohl, 'O'), (and, 'O'), (c, 'O'), (m,
```

```
In [42]: print([(X, X.ent_iob_) for X in d2])
```

```
[(start, 'O'), (of, 'O'), (the, 'O'), (project, 'O'), (gutenberg, 'O'), (ebook, 'O'), (nick, 'B'), (carter, 'I'), (stories,
'O'), (no, 'O'), (dec, 'O'), (oct, 'O'), (nick, 'B'), (carter, 'I'), (stories, 'O'), (_, 'O'), (issued, 'O'), (weekly, 'B'),
```

- Get the entities which are annotated as Person,Organization and Location by Spacy. :
  We have used the following function on the entities returned by spacy to collect the Person, Organization and Location entities in each of the novels respectively. The ent_type function returns the type of entity annotated by Spacy and hence return lists containing the above types of entities.

```
In [43]: def entity_recognition(text):
           doc=nlp(text)
           person=[]
           org=[]
           location=[]
           for X in doc:
             if (X.ent_type_=='PERSON') and X.text not in person:
               person.append(X.text)
             if (X.ent_type_=='ORG')and X.text not in org:
               org.append(X.text)
             if ((X.ent_type_=='LOC') or (X.ent_type_=='GPE')) and X.text not in location:
               location.append(X.text)
           return person,org,location
```

Now collecting these entities from both books:

```
In [44]: person1,org1,location1=entity_recognition(b1)
         person2,org2,location2=entity_recognition(b2)
         print("number of person entities in book 1 and book 2 respectively are "+str(len(person1))+" and "+str(len(person2)))
         print("number of organization entities in book 1 and book 2 respectively are "+str(len(org1))+" and "+str(len(org2)))
         print("number of location entities in book 1 and book 2 respectively are "+str(len(location1))+" and "+str(len(location2)))
```

```
number of person entities in book 1 and book 2 respectively are 44 and 69
number of organization entities in book 1 and book 2 respectively are 42 and 51
number of location entities in book 1 and book 2 respectively are 34 and 49
```

Each of these lists contain the corresponding entities. Counting the number of occurrences of names, locations, and organizations in Book 1, we get :

```
In [47]: X = freq(person1)
         print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('zehedi', 1), ('zeckendorf', 1), ('wheedle', 1), ('wax', 1), ('watch', 1), ('wake', 1), ('von', 1), ('thud', 1), ('syrian',
1), ('sullivan', 1), ('suffer', 1), ('starkman', 1), ('sorry', 1), ('sharon', 1), ('scold', 1), ('sam', 1), ('nod', 1), ('neuma
nn', 1), ('murray', 1), ('morgan', 1), ('mickey', 1), ('lutz', 1), ('loafer', 1), ('jay', 1), ('jack', 1), ('itwait', 1), ('hom
estill', 1), ('hill', 1), ('henry', 1), ('harry', 1), ('haggarty', 1), ('groff', 1), ('george', 1), ('froman', 1), ('ed', 1),
('e', 1), ('dickie', 1), ('dave', 1), ('copy', 1), ('chesbro', 1), ('brissim', 1), ('brayer', 1), ('arthur', 1), ('anderson',
1)]
```

```
In [48]: X = freq(location1)
         print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('york', 1), ('wifei', 1), ('west', 1), ('washington', 1), ('valley', 1), ('usa', 1), ('thud', 1), ('the', 1), ('street', 1),
('springfield', 1), ('pittsburgh', 1), ('philadelphia', 1), ('pennsylvania', 1), ('palestine', 1), ('ohio', 1), ('new', 1), ('m
orrisania', 1), ('missouri', 1), ('mississippi', 1), ('lowder', 1), ('jersey', 1), ('italy', 1), ('hollywood', 1), ('hill', 1),
('hacienda', 1), ('fort', 1), ('detroit', 1), ('delaware', 1), ('connecticut', 1), ('city', 1), ('china', 1), ('chicago', 1),
('burgess', 1), ('america', 1)]
```

```
In [49]: X = freq(org1)
         print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('white', 1), ('usa', 1), ('twoandahalfton', 1), ('the', 1), ('take', 1), ('snap', 1), ('sharon', 1), ('red', 1), ('quartermas
ter', 1), ('of', 1), ('nowit', 1), ('mrs', 1), ('mimeograph', 1), ('mean', 1), ('lincoln', 1), ('inc', 1), ('house', 1), ('hil
l', 1), ('hebertown', 1), ('haggarty', 1), ('groff', 1), ('goudeket', 1), ('goddam', 1), ('gallop', 1), ('froman', 1), ('ford',
1), ('fifth', 1), ('fcc', 1), ('face', 1), ('cross', 1), ('congressman', 1), ('congress', 1), ('civil', 1), ('chesbro', 1), ('b
ook', 1), ('be', 1), ('battalion', 1), ('ballantine', 1), ('avenue', 1), ('authority', 1), ('arthur', 1), ('aeronautics', 1)]
```

- **Performance of The Entity Recognition model on the dataset:**

  To measure the performance of the model we used on the dataset, we took some random sample passages from the novel and labeled them manually for these entities. After doing manual labeling , we compared them with the labels assigned to them by Spacy's Entity Recognition Algorithm.
  Random Passages :

```
passage_1 = "Mickey Groff was a big man and not used to si
```

```
passage_2="it's hotter than Avenue A, Mrs. Goudeket, and c
```

```
passage_3="letters to friends in Palestine , wistful lette
```

```
passage_4="Goudeket's winter cruise to visit Palestine, ne
```

Manually Labelled :

```
handlabel_1 = ['PER','O','O','O','O','O'
handlabel_2 = ['O','O','O','LOC','PER',
handlabel_3 = ['O','O','O','O','GPE','O'
handlabel_4 = ['PER','O','O','O','O','GP
```

After passing the samples to the spacy's algorithm we collect the entities labelled by :

```
labels_1 = ([(X.label_) for X in nlp(passage_1).ents])
labels_2 = ([(X.label_) for X in nlp(passage_2).ents])
labels_3 = ([(X.label_) for X in nlp(passage_3).ents])
labels_4 = ([(X.label_) for X in nlp(passage_4).ents])
```

Now , to Calculate the performance metric we have used precision,recall and the f1 scores of the predictions. To calculate these metrics, we created a confusion matrix using the following code :

```
from sklearn import metrics
# Printing the confusion matrix
# The columns will show the instances predicted for each label,
# and the rows will show the actual number of instances for each label.
print(metrics.confusion_matrix(handlabel, labels , labels=["PER", "ORG", "GPE","LOC","O"]))
# Printing the precision and recall, among other metrics
print(metrics.classification_report(handlabel, labels, labels=["PER", "ORG", "GPE","LOC","O"]))
```

And then calculated the scores from it :

For B1 :

	precision	recall	f1-score	support
PER	0.71	0.69	0.70	68.0
ORG	0.21	0.22	0.22	3.0
LOC	0.75	0.81	0.78	11.0
GPE	0.79	0.83	0.81	23.0
O	0.95	0.93	0.98	552.0
avg/total	0.68	0.69	0.70	657.0

For B2 :

	precision	recall	f1-score	support
PER	0.65	0.64	0.78	58.0
ORG	0.31	0.28	0.25	8.0
LOC	0.68	0.79	0.81	13.0
GPE	0.74	0.84	0.86	30.0
O	0.90	0.91	0.96	540.0
avg/total	0.65	0.69	0.73	649.0

Therefore , we get f1- score of 0.70 for first book (B1) and 0.73 for second book (B2) .

☐ Create TF-IDF vectors for all books and find the cosine similarity :

- **Firstly , we will import the texts from the three books :**

```
f1 = open("E:/T1.txt",'r')
f2 = open("E:/T2.txt",'r')
f3 = open("E:/T3.txt",'r')
```

```
b1 = f1.read()
b2 = f2.read()
b3 = f3.read()
```

- **We create a corpus for the three books**

```
In [17]: doc_corpus=[b1,b2,b3]
```

- **We use tf-idf Vectorizer module for finding the available features in the corpus:**

```
In [19]: vec=TfidfVectorizer(stop_words='english')
```

```
In [20]: matrix=vec.fit_transform(doc_corpus)
```

```
In [22]: print("Feature Names n",vec.get_feature_names())
```

```
r', 'trajectory', 'tramped', 'tramping', 'tramps', 'transaction', 'transcontinental', 'transcribed', 'transcribers', 'transfe
r', 'transferred', 'transfigured', 'transformed', 'transient', 'transients', 'transistor', 'transmission', 'transmitter', 'tr
anspired', 'transport', 'transportation', 'trap', 'trapdoor', 'trapped', 'trapper', 'trapping', 'traps', 'traveling', 'travel
s', 'treacherous', 'treachery', 'treasure', 'treasury', 'treated', 'treatment', 'tree', 'trees', 'treeshaded', 'trembling',
```

- **We generate a matrix containing tf-idf for each book and extract the tf-idf values corresponding to each book :**

```
In [23]: print("Sparse Matrix n",matrix.shape,"n",matrix.toarray())
```

```
Sparse Matrix n (3, 9564) n [[0.00156157 0.         0.00156157 ... 0.00468472 0.00156157 0.        ]
 [0.         0.00500151 0.         ... 0.         0.         0.00250075]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

```
In [24]: import scipy
```

```
In [36]: array = matrix.toarray()
```

```
In [37]: tfidf_b1 = array[0]
         tfidf_b2 = array[1]
         tfidf_b3 = array[2]
```

```
In [38]: print(tfidf_b1)
```

```
[0.00156157 0.         0.00156157 ... 0.00468472 0.00156157 0.        ]
```

```
In [39]: print(tfidf_b2)
```

```
[0.         0.00500151 0.         ... 0.         0.         0.00250075]
```

```
In [40]: print(tfidf_b3)
```

```
[0. 0. 0. ... 0. 0. 0.]
```

```
In [43]: print(tfidf_b3[510])
```

```
0.004050680669534677
```

Finding cosine similarity :

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them.

Using scipy library we find the cosine similarity for each book

In [46]:
```
#Cosine_similarity between b1 and b2
cosine_similarity = 1 - spatial.distance.cosine(tfidf_b1, tfidf_b2)
print(cosine_similarity)
```

```
0.3406396620370207
```

In [47]:
```
#Cosine_similarity between b1 and b3
cosine_similarity = 1 - spatial.distance.cosine(tfidf_b1, tfidf_b3)
print(cosine_similarity)
```

```
0.11372010996665316
```

In [48]:
```
#Cosine_similarity between b2 and b3
cosine_similarity = 1 - spatial.distance.cosine(tfidf_b2, tfidf_b3)
print(cosine_similarity)
```

```
0.08900553550644419
```

Inferences :

Before performing lemmatization , the values of cosine similarity between Book 1 and Book 2 is highest among others . Therefore , we can say Book 1 and Book 2 are more similar .

Now for the next part we perform , lemmatization of the books and recreate the TF-IDF vectors for all the books and find the cosine similarity of each pair of books.

- **Lemmatizing the given books :**

```
In [95]: #Lemmatization of the books and recreating tfidf vectors

In [50]: lemmatizer = WordNetLemmatizer()

In [53]: def tokenize(t):
             token = nltk.word_tokenize(t)
             return token

In [51]: def lemmatize_word(text):
             word_tokens = tokenize(text)
             lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word_tokens]
             return ' '.join(lemmas)

In [54]: b1 = lemmatize_word(b1)

In [56]: b2 = lemmatize_word(b2)
         b3 = lemmatize_word(b3)
```

- **We create a corpus for the three books :**

```
In [57]: doc_corpus=[b1,b2,b3]
```

- **We use tf-idf Vectorizer module for finding the available features in the corpus:**

```
In [58]: vec=TfidfVectorizer(stop_words='english')

In [59]: matrix=vec.fit_transform(doc_corpus)
```

```
In [60]: print("Feature Names n",vec.get_feature_names())
uardsmen', 'guess', 'guest', 'guests', 'guff', 'guide', 'guilt', 'guilty', 'gulf', 'gulfport', 'gullied', 'gulp', 'gumshoe',
'gun', 'gunman', 'gunshot', 'gunsize', 'gurgle', 'gush', 'gusher', 'gut', 'gutenberg', 'gutted', 'gutter', 'guy', 'gym', 'gym
nasium', 'ha', 'habit', 'hacienda', 'hadand', 'hades', 'hadnt', 'haemoangioma', 'hag', 'haggarty', 'hah', 'hail', 'hair', 'ha
```

- **We generate a matrix containing tf-idf for each book and extract the tf-idf values corresponding to each book :**

```
In [61]: print("Sparse Matrix n",matrix.shape,"n",matrix.toarray())

Sparse Matrix n (3, 7569) n [[0.00147642 0.         0.00147642 ... 0.00295284 0.00590567 0.        ]
 [0.         0.00475426 0.         ... 0.         0.         0.00237713]
 [0.         0.         0.         ... 0.         0.         0.        ]]

In [62]: array = matrix.toarray()

In [63]: tfidf_b1 = array[0]
         tfidf_b2 = array[1]
         tfidf_b3 = array[2]

In [64]: print(tfidf_b1)

[0.00147642 0.         0.00147642 ... 0.00295284 0.00590567 0.        ]

In [65]: print(tfidf_b2)

[0.         0.00475426 0.         ... 0.         0.         0.00237713]

In [66]: print(tfidf_b3)

[0. 0. 0. ... 0. 0. 0.]

In [91]: print(tfidf_b3[4])

0.0040688023525317345
```

Finding cosine similarity(after lemmatizing books) :

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them. Using scipy library we find the cosine similarity for each book

In [92]:
```
#Cosine_similarity between b1 and b2
cosine_similarity = 1 - spatial.distance.cosine(tfidf_b1, tfidf_b2)
print(cosine_similarity)
```

```
0.4032543173522589
```

In [93]:
```
#Cosine_similarity between b1 and b3
cosine_similarity = 1 - spatial.distance.cosine(tfidf_b1, tfidf_b3)
print(cosine_similarity)
```

```
0.19128910895758378
```

In [94]:
```
#Cosine_similarity between b2 and b3
cosine_similarity = 1 - spatial.distance.cosine(tfidf_b2, tfidf_b3)
print(cosine_similarity)
```

```
0.15053706598819994
```

Inferences :

After lemmatization also , the values of cosine similarity between Book 1 and Book 2 is highest among others . Therefore , we can say Book 1 and Book 2 are more similar .