LNMIIT
The LNM Institute of Information Technology

NLP Project

NLP Project Round - 1

Team Name : Three's Company

Aayush Mehta 19ucs100

Ram Ahuja 19ucs017

Vaibhav Gupta 19ucs115

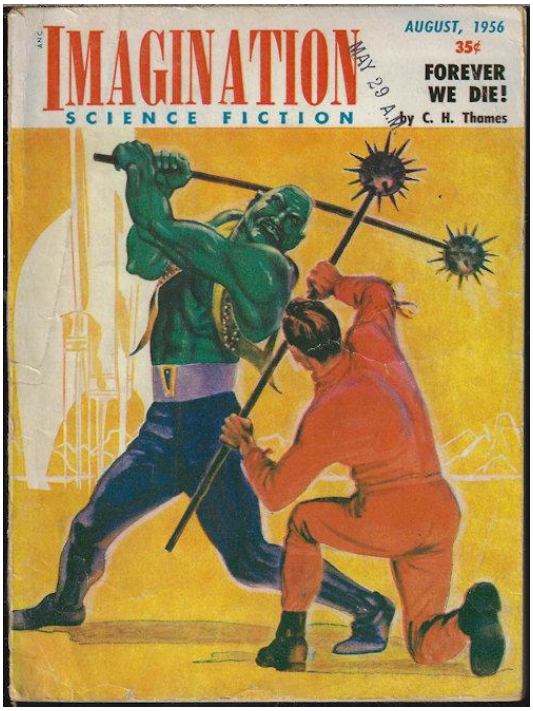
GitHub Repo :

https://github.com/Aayushmehta2001/NLP_Project

Overview :

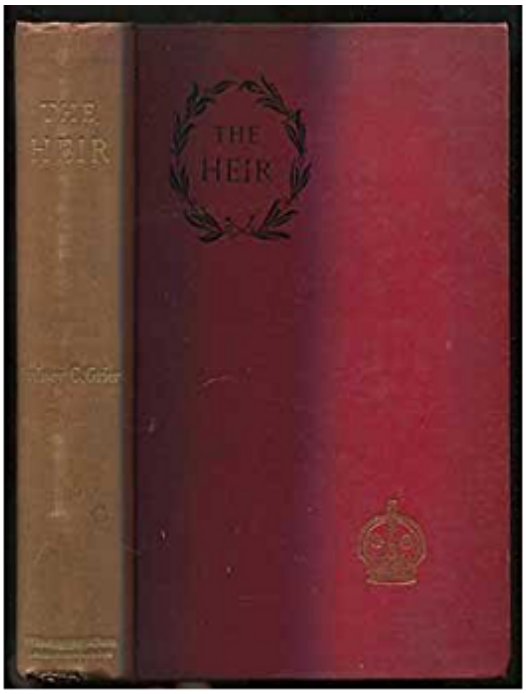
In this project we perform textual analysis on two of our chosen books “Traitor's Choice” and “ The Heir” . After that we will apply POS tagging on both the books . We will do all this using NLP techniques, with the help of python libraries.

Books used :



Traitor's Choice

By Paul W. Fairman



The Heir

By Sydney C. Grier

Goals :

- Import the text from two books, let's call it as T1 and T2.
- Perform simple text pre-processing steps and tokenize the text T1 and T2.
- Analyze the frequency distribution of tokens in T1 and T2 separately.
- Create a Word Cloud of T1 and T2 using the token that you have got.
- Remove the stopwords from T1 and T2 and then again create a word cloud.
- Compare with word clouds before the removal of stopwords.
- Evaluate the relationship between the word length and frequency for both T1 and T2.
- Do PoS Tagging for both T1 and T2 using any of the four tagset studied in the class and Get the distribution of various tags .

Python Libraries Used :

- Urllib - Used to fetch text data from Gutenberg URLs
- NLTK - Used for Tokenizing, Lemmatization and Removing Stopwords
- Re - Used to remove URLs and Decontract Contractions in English Language
- Word Cloud - Used to create WordClouds from Tokenized Data

Inflect - Used to replace numbers with words

- Maplotlib - Used to Visualize our text data

Data Preprocessing Steps :

1. Discard Useless Portion of book -

We will Discard the Documentation part of the book that is of no use to us.

```
[ ]  def discard_useless_part (text):
       sidx = text.find('*** START OF THE PROJECT ')
       eidx = text.find('*** END OF THE PROJECT ')
       print("Discarding Before - ", sidx)
       print("Discarding After - ", eidx)
       text = text[sidx:eidx]
       return text
```

2. Convert all data to Lowercase -

We will convert all text data to lowercase, as the case does not contribute much to the meaning of data.

```
def to_lower(text):
  return text.lower()
```

3. Converting Number to Words -

For this, we use inflect Python Library which has a function p.number_to_words that will give us the English equivalent of a number using basic mapping techniques.

```
def num2word(text):
  list_of_words = text.split()
  modified_text = []

  for word in list_of_words:
    if word.isdigit():
      number_in_word = p.number_to_words(word)
      modified_text.append(number_in_word)
    else:
      modified_text.append(word)

  return ' '.join(modified_text)
```

4. Removing Contractions and Punctuations -

We will do this using a Python Library re that will help us apply regular expressions on our data as desired.

```
def decontracted(text):
    # specific
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)

    # general
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text
```

```
def remove_punctuation(text):
  tokens = word_tokenize(text)
  words = [word for word in tokens if word.isalpha()]
  return ' '.join(words)
```

5. Removing URLs -

Again, we would do this using re

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

6. Lemmatization -

We do Lemmatization with the help of WordNetLemmatizer() function from nltk.stem which gives the lemmatized form of all verbs

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

Data Preparation :

We apply all the functionalities we added above and Prepare our data for analysis.

```
def PreProcessedBook(url):
  book = read_book(url)
  print_book_title_and_length(book)
  text = decode_book(book)
  text = discard_useless_part(text)
  text = to_lower(text)
  text = remove_URL(text)
  text = decontracted(text)
  text = num2word(text)
  text = remove_punctuation(text)
  text = lemmatize_word(text)
  return (text)
```

```
book1_text = PreProcessedBook(url1)
book2_text = PreProcessedBook(url2)
```

Problem Statements and Inferences :

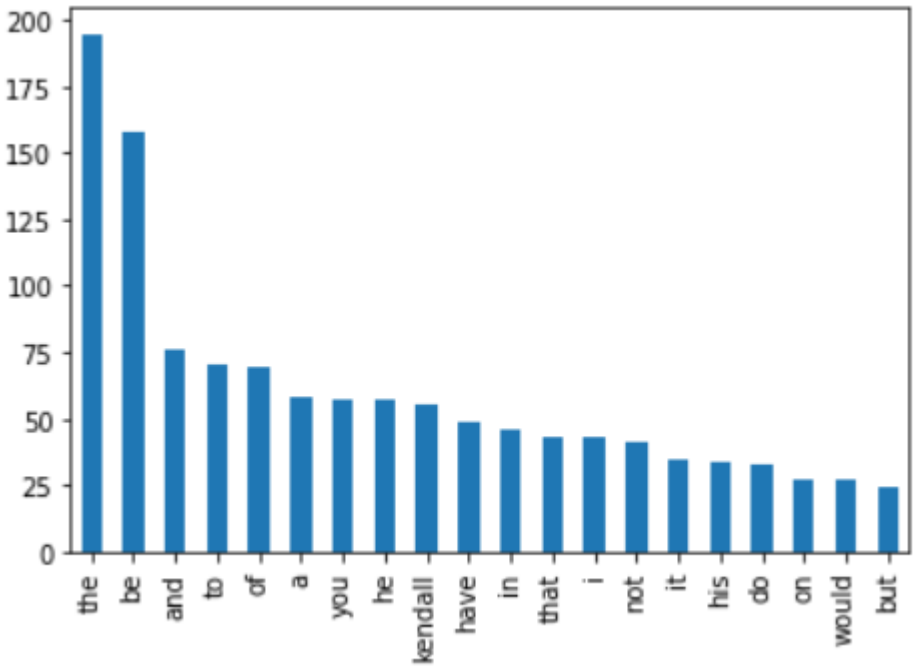
- Analyze the frequency distribution of tokens in T1 and T2 separately.

For this step, we take help of python library pandas
First we tokenize the given data, and then we plot a histogram of top 10 most frequent words.

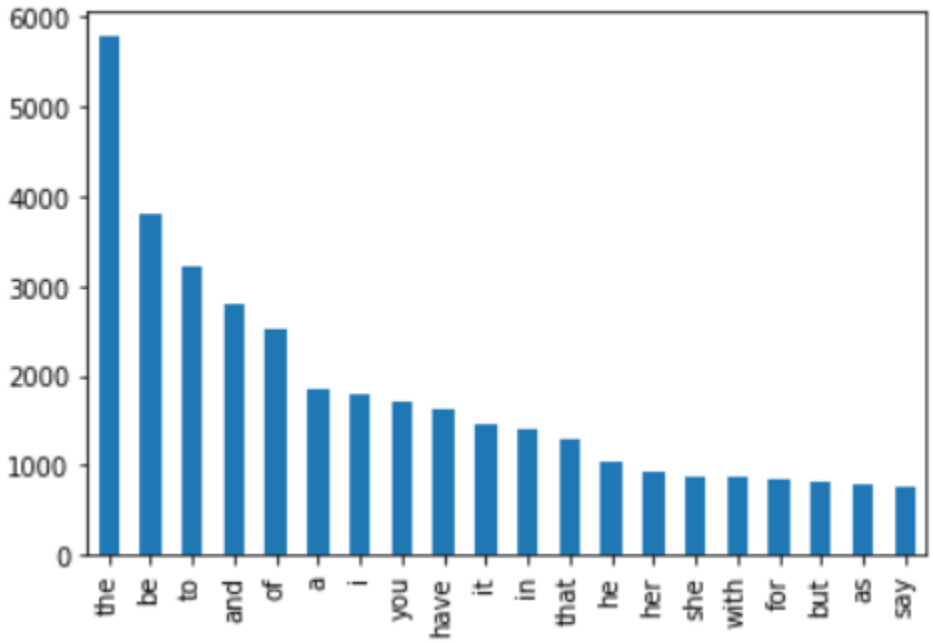
```
word_tokens1 = word_tokenize(book1_text)
pd.Series(word_tokens1).value_counts()[:20].plot(kind='bar')
```

```
word_tokens2 = word_tokenize(book2_text)
pd.Series(word_tokens2).value_counts()[:20].plot(kind='bar')
```

Frequency distribution of T1 :



Frequency distribution of T2 :



- **Generating a word cloud from T1 and T2**

For this task we take help of python library **'wordcloud'** and its function '**WordCloud()**' It helps in generating wordclouds from a list of Tags from Text data.

```
# Generate word cloud
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1,
                      background_color='salmon', colormap='Pastel1',stopwords= [],
                        collocations=False).generate(' '.join(word_tokens1))
# Plot
plot_cloud(wordcloud)
```

Wordcloud of T1

Wordcloud of T2

Inferences

We infer from the above visualizations that

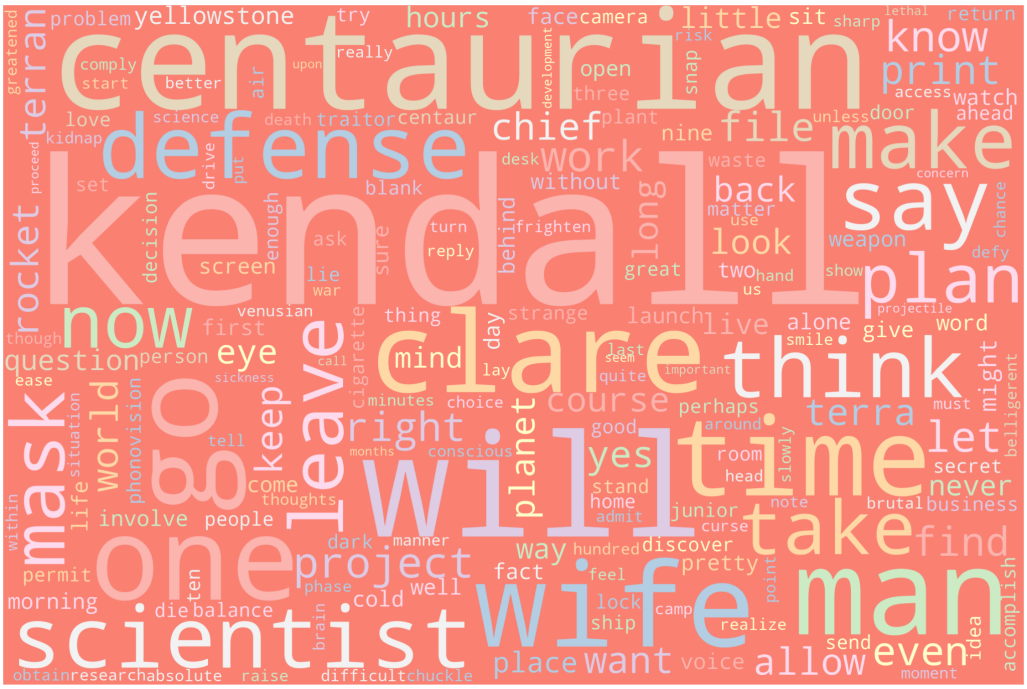
- Words like ‘and’, ‘be’ and ‘the’ are the most frequently used words in T1.
- Words like ‘to’, ‘be’ and ‘the’ are the most frequently used words in T2.
- These words do not contribute to the meaning of the sentence and are mostly useless for us.
- These words are known as ‘stopwords’ and we need to get them out of it.

● Generating new word clouds after removing stopwords

To remove stopwords, we use an inbuilt function in nltk called STOPWORDS.

```
def remove_stopwords(tokens):
    return [word for word in tokens if word not in STOPWORDS]
```

```
T1 = remove_stopwords(word_tokens1)
T2 = remove_stopwords(word_tokens2)
```



Wordcloud of T1(after Removing Stopwords)

Wordcloud of T2(after Removing Stopwords)

Inferences

We infer from the above visualizations that

● Now most of the words that were of no meaning to us have been removed.

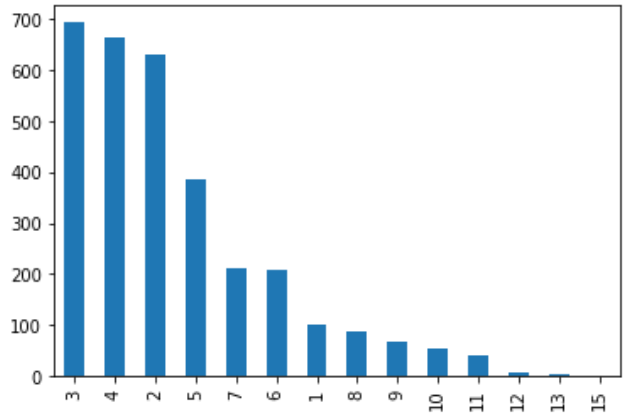
● New words like ‘kendall’, ‘centaurian’ and ‘clare’ now dominate in frequency in T1 which sort of reveals the name of characters of the book around whom the story
will revolve.

● Words like ‘say’, ‘zoe, ‘one’ and ‘maurice’ are the new frequently used words in T2, though it is tough to make inferences based on this information but we are able to roughly guess that there is some ‘don’ and ‘professor’ in the story.

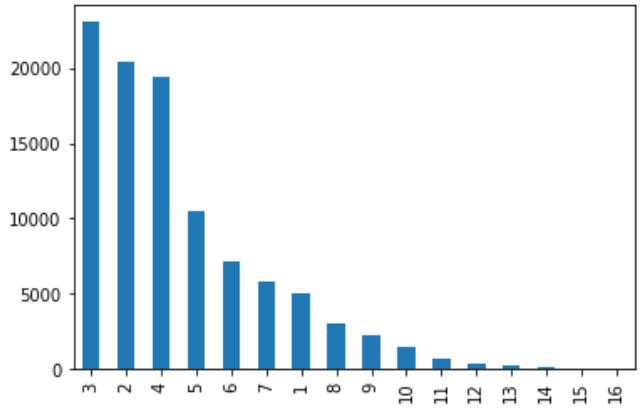
● We have gotten rid of ‘stopwords’ and are now able to draw meaningful conclusions from the data.

● As a part of the project, we would also like to evaluate the relationship between the word length and frequency for both T1 and T2 both before and after the removal of stopwords.

Before

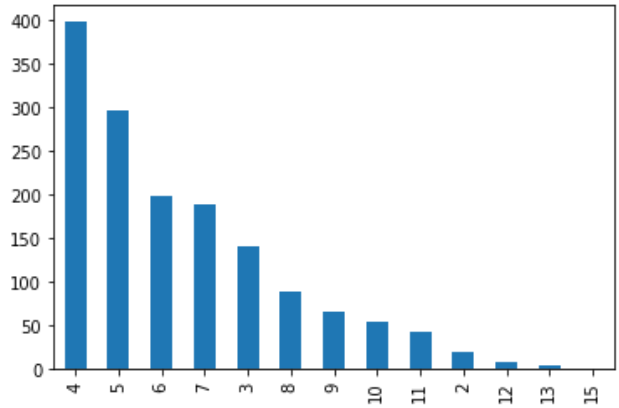


T1

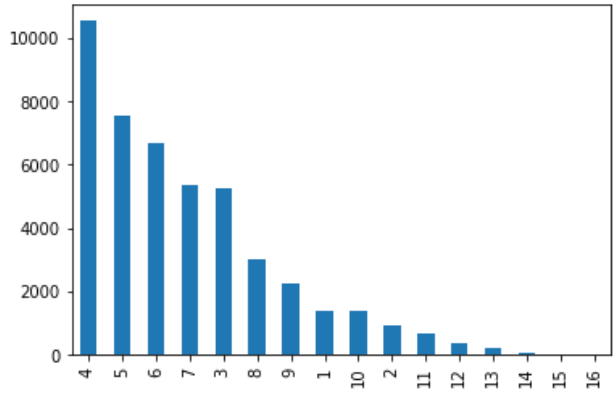


T2

After



T1



T2

Inferences

We infer from the above visualizations that

- The number of words of length 1, 2 and 3 have significantly decreased after the removal of stopwords.
- We can clearly infer that this is due to the removal of stopwords like ’a’ ,‘be’, ‘the’, ‘of’ and ‘and’ which were the highest occurring words before removal.

• Performing 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 tag_treebank(tokens):
  tagged=nltk.pos_tag(tokens)
  return tagged
```

```
book1_tags = tag_treebank(T1)
book2_tags = tag_treebank(T2)
```

```
print(book1_tags)
```

('start', 'NN'), ('project', 'NN'), ('gutenberg', 'NN'), ('ebook', 'NN'), ('traitor', 'NN'), ('choice', 'NN'), ('traitor', 'NN'), ('choice', 'NN'), ('paul', 'NN'), ('fairman', 'NN'), ('kendall', 'NN'), ('difficult', 'JJ'), ('decision', 'NN'), ('make', 'VBP'), ('defy', 'NN'), ('alien', 'NN'), ('clare', 'NN'), ('face', 'NN'), ('horrible', 'JJ'), ('death', 'NN'), ('comply', 'NN'), ('whole', 'JJ'), ('planet', 'NN'), ('must', 'MD'), ('die', 'VB'), ('transcriber', 'NNP'), ('note', 'NN'), ('etext', 'NN'), ('produce', 'VBP'), ('imagination', 'NN'), ('stories', 'NNS'), ('science', 'NN'), ('fantasy', 'NN'), ('august', 'VBP'), ('one', 'CD'), ('thousand', 'CD'), ('nine', 'CD'), ('hundred', 'VBN'), ('extensive', 'JJ'), ('research', 'NN')

```
print(book2_tags)
```

('start', 'NN'), ('project', 'NN'), ('gutenberg', 'NN'), ('ebook', 'VBP'), ('heir', 'PRP$'), ('heir', 'NN'), ('sydney', 'NN'), ('grier', 'NN'), ('author', 'NN'),

('uncrowned', 'VBD'), ('king', 'VBG'), ('warden', 'JJ'), ('march', 'NN'), ('etc', 'NN'), ('illustrations', 'NNS'), ('george', 'VBP'), ('percy', 'JJ'), ('balkan', 'JJ'), ('series', 'NN'), ('william', 'NN'), ('blackwood', 'NN'), ('sons', 'NNS'), ('edinburgh', 'VBP'), ('london', 'JJ'), ('mcmvi', 'JJ'), ('image', 'NN'), ('caption', 'NN'), ('arm', 'NN'), ('grip', 'NN'), ('one', 'CD'), ('brigands', 'VBZ'), ('trudge', 'NN'), ('silently', 'RB'), ('beside', 'JJ'), ('content', 'NN'), ('de', 'IN'), ('jure', 'NN'), ('ii', 'NN'), ('stock', 'NN'), ('emperors', 'NNS'), ('iii', 'VBP'), ('orient', 'JJ').

• 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.

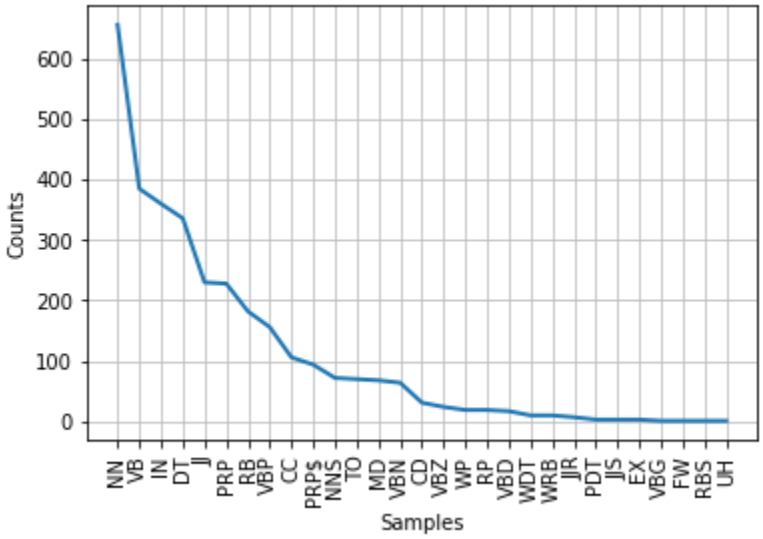
```
from collections import Counter
def get_counts(tags):
  counts = Counter( tag for word,  tag in tags)
  return counts
```

```
def FrequencyDist(tags):
  wfd = nltk.FreqDist(t for (w,t) in tags)
  wfd
  wfd.plot(50)
```

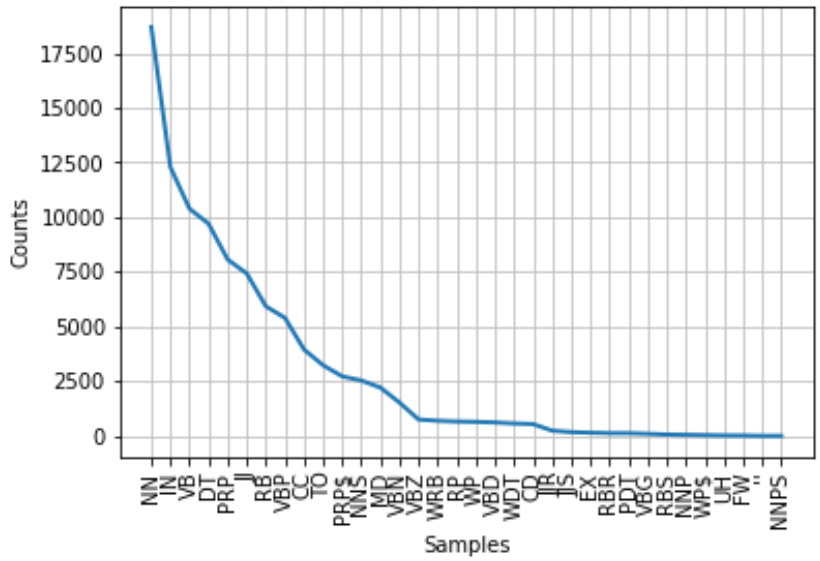
```
book1_pos_count=get_counts(book1_tags)
book2_pos_count=get_counts(book2_tags)
```

```
FrequencyDist(book1_tags)
FrequencyDist(book2_tags)
```

Frequency Distribution of Tags in T1



Frequency Distribution of Tags in 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.

Conclusion :

We have learnt how to perform Text Preprocessing, Tokenization on Text Data and then answer all the Problems given to us in NLP Project Round 1 with the use of Plots and Visualisations and learnt to draw meaningful inferences from it.

Project Round - 2

Team Name : Three's Company

Aayush Mehta 19ucs100

Ram Ahuja 19ucs017

Vaibhav Gupta 19ucs115

GitHub Repo :

https://github.com/Aayushmehta2001/NLP_Project

Goals :

First Part :

- 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 novel.

Second Part :

- 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. 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 with F1 score here.

Third Part:

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

- Urllib - Used to fetch text data from Gutenberg URLs
- NLTK - Used for Tokenizing, Lemmatization and Removing Stopwords
- Re - Used to remove URLs and Decontract Contractions in English Language
- Matplotlib - Used to Visualize our text data
- Spacy- To perform Entity recognition in text
- Numpy- To get frequency distributions of nouns and verbs
- Typing-To perform evaluation of the Algorithm in Entity Recognition.
- Pandas - Pandas is used to analyze data.
- Sklearn.metrics - The **sklearn.metrics** module implements functions assessing prediction error for specific purposes.
- TfidfVectorizer - Convert a collection of raw documents to a matrix of TF-IDF features.
- Cosine_similarity - Compute cosine similarity between samples in X and Y. Cosine similarity, or the cosine kernel, computes similarity as the normalized dot product of X and Y.
- WordNetLemmatizer - In order to lemmatize, you need to create an instance of the WordNetLemmatizer() and call the lemmatize() function on a single word.
- Nltk.stem - Interfaces used to remove morphological affixes from words, leaving only the word stem
- Nltk.corpus - The modules in this package provide functions that can be used to read corpus files in a variety of formats.

Problem Statements and Inferences :

PART - 1

NOUNS and VERBS Detection:

1. Find the nouns and verbs in both the novels.

● Performing POS Tagging

We will now 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.

```
[ ]  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
```

```
[ ]  noun1=noun(book1_text)
     noun2=noun(book2_text)
```

```
[ ]  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
```

```
[ ]  verb1=verb(book1_text)
     verb2=verb(book2_text)
```

We will apply the above functions to both book1 and book2 respectively and print the total nouns and verbs in both of them.

```
[ ]  print("Number of nouns in book 1 are "+ str(len(noun1)))

     Number of nouns in book 1 are 728
```

```
[ ]  print("Number of nouns book 2 are "+str(len(noun2)))

     Number of nouns book 2 are 21271
```

```
[ ]  print("Number of verbs in book 1 are "+ str(len(verb1)))

     Number of verbs in book 1 are 647
```

```
[ ]  print("Number of verbs in book 2 are "+ str(len(verb2)))

     Number of verbs in book 2 are 18773
```

2. 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.

```
[ ]  #gives the categories of nouns or verb that the word belongs to
     from nltk.corpus import wordnet as wn
     def synset(words):
       categories=[]
       for word in words:
         cat=[]
         for synset in wn.synsets(word):
           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.

```
[ ]  noun_syn1=synset(noun1)
     noun_syn2=synset(noun2)
     verb_syn1=synset(verb1)
     verb_syn2=synset(verb2)
```

Hence noun_syn1, noun_syn2, verbsyn_1,verb_syn2 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_syn1[x][y] where x is the index of the corresponding noun in noun1 and y is the index containing the categories it belongs to.

Eg -

```
[ ]  print(noun1[88])

     course
```

Eg -

```
[ ]  print(noun_syn1[88][:])

     ['noun.act', 'noun.group', 'noun.location', 'noun.act', 'noun.object', 'noun.group'
```

Hence a ‘course’ is either an act, group, location, object.

3. 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.

```
[ ]  #GIVES TOTAL NOUN LEXNAMES AND TOTAL VERB LEXNAMES FOR FREQUENCY DISTRIBUTIONS
     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
```

```
[ ]  noun_superset1,verb_superset1=all_synsets(noun1,verb1)
     noun_superset2,verb_superset2=all_synsets(noun2,verb2)
```

Eg-

```
[ ]  print(noun_superset1)

     ['noun.event', 'noun.time', 'noun.act', 'noun.act', 'noun.act', 'noun.location', 'noun.communication', 'noun.attribute',
```

```
[ ]  len(noun_superset1)

     4990
```

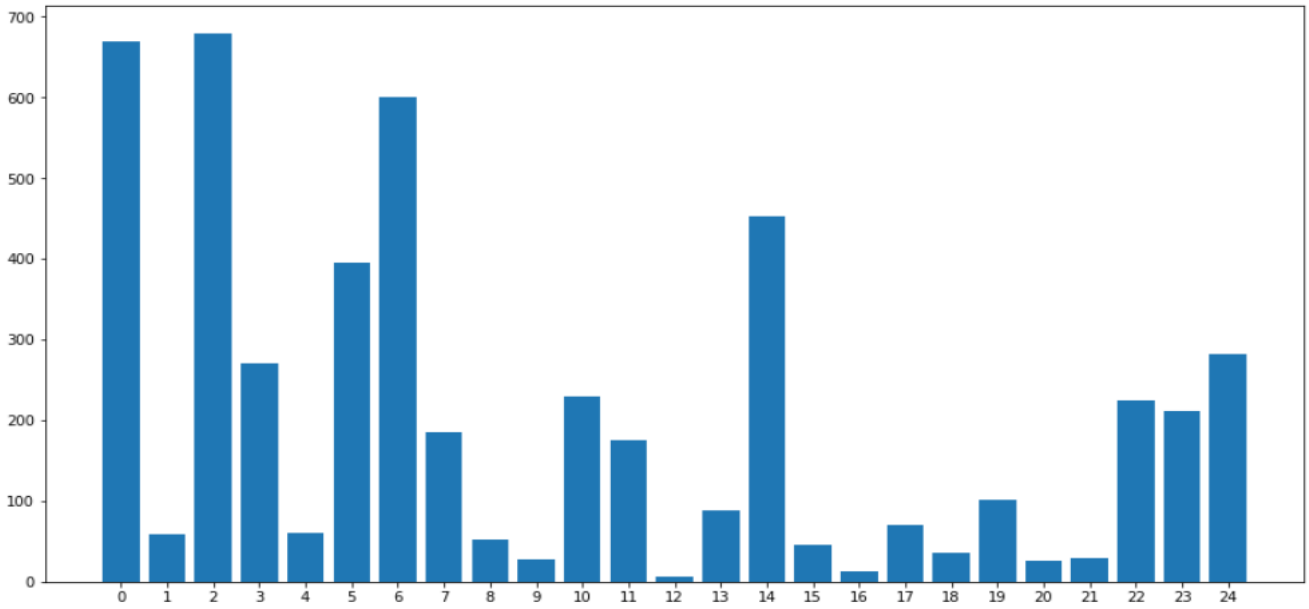
Hence there are **4990** elements in the list.

Plotting the histograms:

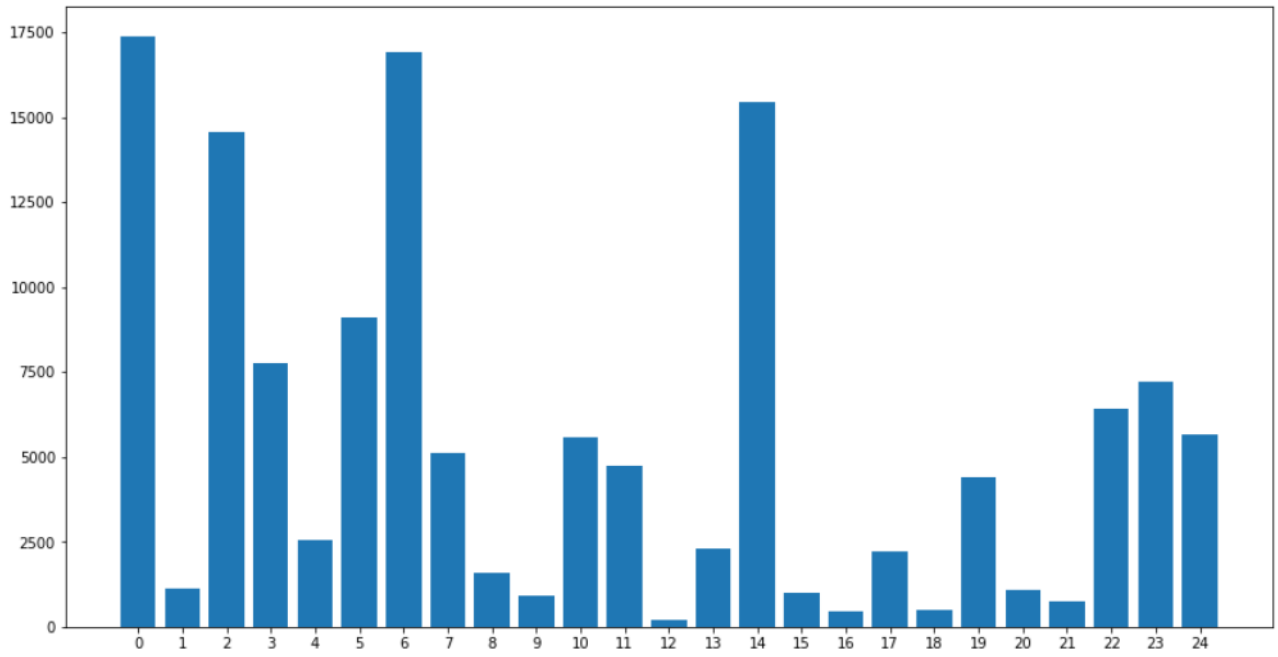
We have used numpy to count the frequency of each type of noun and verb out of 25 categories of nouns and 15 categories of verbs from both the novels.

```
[ ]  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:



For Book 1 Nouns



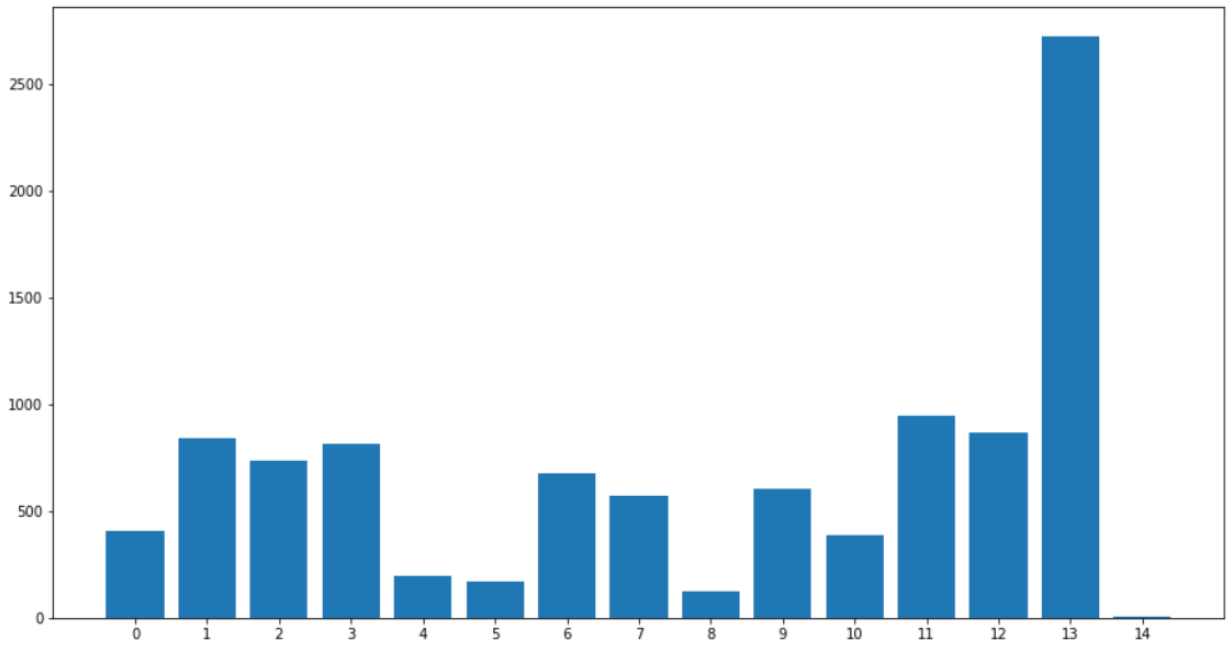
For Book 2 Nouns

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

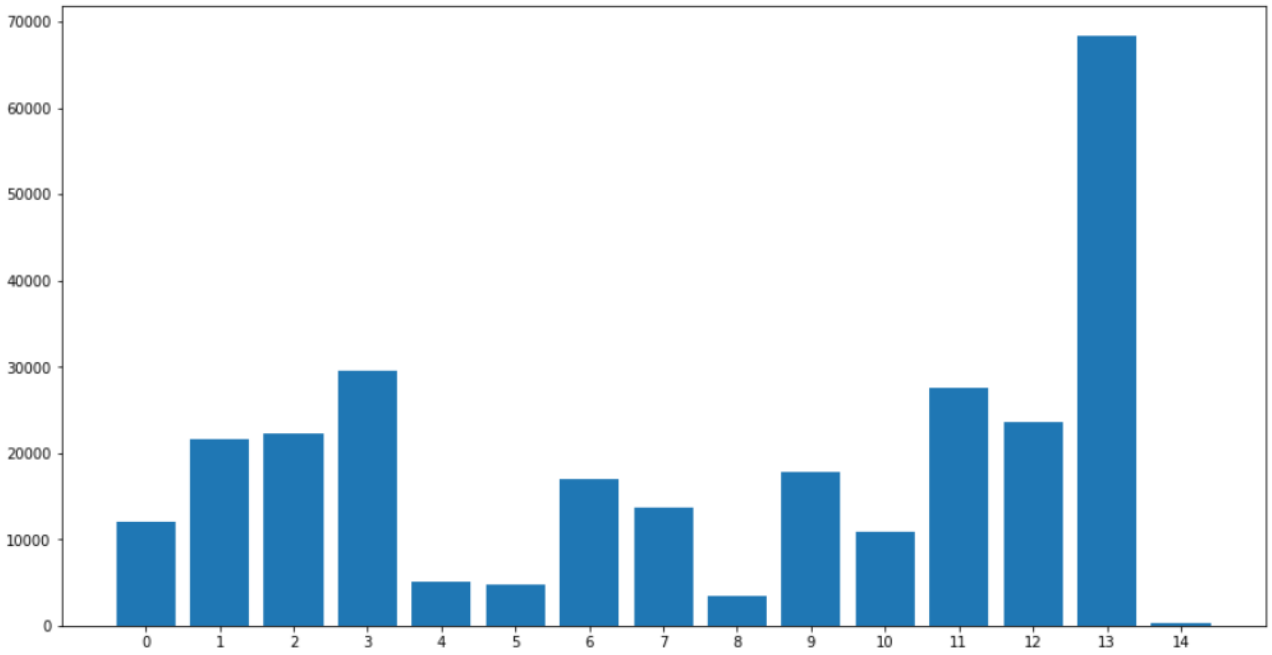
```
[ ]  print(labels)
```

```
['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:



For Verbs in Book 1



For Verbs in Book 2

And the labels are numbered as 0-14 in the following order.

```
print(labels)
```

```
['verb.body' 'verb.change' 'verb.cognition' 'verb.communication'
 'verb.competition' 'verb.consumption' 'verb.contact' 'verb.creation'
 'verb.emotion' 'verb.motion' 'verb.perception' 'verb.possession'
 'verb.social' 'verb.stative' 'verb.weather']
```

PART -2

Named Entity Recognition

1. Get the entities involved in each of the novels.

To perform Named Entity Recognition in both the novels we have used Spacy. SpaCy's named entity recognition has been trained on the OntoNotes 5 corpus and it supports the following entity types:

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.

CARDINAL	Numerals that do not fall under another type.

The Spacy uses token-level entity annotation using the BILUO tagging scheme to describe the entity boundaries. Where,

TAG	DESCRIPTION
B EGIN	The first token of a multi-token entity.
I N	An inner token of a multi-token entity.
L AST	The final token of a multi-token entity.
U NIT	A single-token entity.
O UT	A non-entity token.

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

```
[ ]  import spacy
     from spacy import displacy
     from collections import Counter
     import en_core_web_sm
     nlp = en_core_web_sm.load()
     doc1 = nlp(book1_text)
     doc2 = nlp(book2_text)
```

```
[ ]  print("there are total "+str(len(doc1.ents))+" entities in book 1 ")

     there are total 109 entities in book 1
```

```
[ ]  print(" there are total " +str(len(doc2.ents))+ " entities in book 2 ")

      there are total 2356 entities in book 2
```

```
[ ] print([(X, X.ent_iob_) for X in doc1])

    [(start, 'O'), (of, 'O'), (the, 'O'), (project, 'O'), (gutenberg, 'O'), (ebook, 'O'), (traitor, 'O'), (be, 'O'), (choice, 'O'),
```

```
[ ] print([(X, X.ent_iob_) for X in doc2])

    [(start, 'O'), (of, 'O'), (the, 'O'), (project, 'O'), (gutenberg, 'O'), (ebook, 'O'), (the, 'O'), (heir, 'O')
```

As we can see the entities are annotated using the BILUO entity scheme.

3. Get the entities which are annotated 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.

```
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:

```
[ ] person1,org1,location1=entity_recognition(book1_text)
    person2,org2,location2=entity_recognition(book2_text)
    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 8 and 247
    number of organization entities in book 1 and book 2 respectively are 12 and 74
    number of location entities in book 1 and book 2 respectively are 1 and 65
```

Each of these lists contain the corresponding entities.

Counting the number of occurrences of names, locations, and organizations in Book 1, we get :

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

    [('yellowstone', 1), ('reed', 1), ('paul', 1), ('lie', 1), ('kendall', 1), ('hunt', 1), ('fairman', 1), ('clare', 1)]
```

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

    [('kendall', 1)]
```

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

    [('tray', 1), ('the', 1), ('research', 1), ('project', 1), ('ordnance', 1), ('nova', 1), ('kendall', 1), ('instructions', 1),
```

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.

```
def metrics(truth, run):
    t = set(truth)
    r = set(run)
    intersection = r & t
    True_positive = float(len(intersection))
    if float(len(run)) >= float(True_positive):
        False_positive = len(run) - True_positive
    else:
        False_positive = True_positive - len(run)
    True_negative = 0
    if len(truth) >= len(run):
        False_negative = len(truth) - len(run)
    else:
        False_negative = 0
    accuracy = (float(True_positive) + float(True_negative)) / (float(True_positive) + float(True_negative) + float(False_positive) + float(False_negative))
    precision = float(True_positive) / (float(True_positive) + float(False_positive))
    recall = float(True_positive) / (float(True_positive) + float(False_negative))
    F_measure = (2 * recall * precision) / (recall + precision)
    print("Accuracy: ", accuracy)
    print("Recall: ", recall)
    print("Precision: ", precision)
    print("F-measure: ", F_measure)
    d = {'Predicted Negative': [True_negative, False_negative],
         'Predicted Positive': [False_positive, True_positive]}
    metricsdf = pd.DataFrame(d, index=['Negative Cases', 'Positive Cases'])
    return metricsdf
```

Result :

Two lists were made, namely “truth” and “run”. The truth list contains the manually labelled entities and the 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	
		Yes	No
ACTUAL CLASS	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 formulas :

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 :

```
[40] print('Maunally labelled PERSON entities : ')
     print(set(truth))
     print('')
     print('Algorithm labelled PERSON entities : ')
     print(set(run))
     print('')
     print('Evaluation :')
     print(metrics(truth, run))
```

```
Maunally labelled PERSON entities :
{'devil', 'Blonde', 'Kendall', 'Centuarians', 'Clare', 'Terra'}

Algorithm labelled PERSON entities :
{'Blonde', 'Clare'}

Evaluation :
Accuracy:  0.3333333333333333
Recall:  0.4
Precision:  0.6666666666666666
F-measure:  0.5
                Predicted Negative  Predicted Positive
Negative Cases                   0                 1.0
Positive Cases                   3                 2.0
```

Paragraph-2 :

```
[45] print('Maunally labelled PERSON entities : ')
     print(set(truth))
     print('')
     print('Alogorithm labelled PERSON entities : ')
     print(set(run))
     print('')
     print('Evaluation :')
     print(metrics(truth, run))
     print('-----------------------------------------------------------------------------------')
```

```
Maunally labelled PERSON entities :
{'Teffany', 'Nicholas', 'Scythia', 'Emathia', 'Panagiotis', 'Zoe', 'Abbey', 'Maurice'}

Alogorithm labelled PERSON entities :
{'Isn', 'Nicholas', 'Scythia', 'Emathia', 'Panagiotis', 'Zoe'}

Evaluation :
Accuracy:  0.625
Recall:  0.7142857142857143
Precision:  0.8333333333333334
F-measure:  0.7692307692307692
                Predicted Negative  Predicted Positive
Negative Cases                   0                 1.0
Positive Cases                   2                 5.0
-----------------------------------------------------------------------------------
```

PART - 3

Books used -

B1 - Traitor's Choice. by Paul W. Fairman (used above)
B2 - The Heir By Sydney C. Grier (used above)
B3 - Law Rustlers by W. C. Tuttle . (New book taken.)

Algorithms Used -

1. tf*idf Algorithm - TF*IDF is an information retrieval technique that weighs a term's frequency (TF) and its inverse document frequency (IDF). Each word or term that occurs in the text has its respective TF and IDF score.
The product of the TF and IDF scores of a term is called the TF*IDF weight of that term. Put simply, the higher the TF*IDF score (weight), the rarer the term is in a given document and vice versa.

2. **Cosine Similarity Algorithm** - The Cosine Similarity procedure computes similarity between all pairs of items. It is a symmetrical algorithm, which means that the result from computing the similarity of Item A to Item B is the same as computing the similarity of Item B to Item A. We can therefore compute the score for each pair of nodes once. We don't compute the similarity of items to themselves.

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

Before Lemmatization -

1. Book1 & Book2 -

- TF-IDF vector -
  Here, we have used sklearn.feature_extraction.text as a Library and TfidfVectorizer as a package to create the TF-IDF vector.

```
from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)
```

	abandon	abandoning	abashed	abbey	abbot	able	abodes	abortive	about	above	abraham	abroad	abruptly	absence	absent	absently	absolute	absolute
text_book1	0.000000	0.000000	0.000000	0.000000	0.000000	0.003255	0.000000	0.000000	0.003255	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004575	0.006510	0.000
text_book2	0.000302	0.000151	0.000151	0.000603	0.000302	0.004615	0.000151	0.000151	0.020283	0.003771	0.000151	0.000452	0.000603	0.000754	0.000452	0.000000	0.000537	0.002

2 rows × 8322 columns

- Cosine Similarity Matrix -
  Here, we have used sklearn.metrics.pairwise as a Library and cosine_similarity as a Package.

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['Book1','Book2'])
```

	Book1	Book2
text_book1	1.000000	0.892343
text_book2	0.892343	1.000000

2. Book1 & Book3 -

- TF-IDF vector -

	able	abolish	about	above	absently	absolute	absolutely	absurd	accept	accepts	access	accomplished	accorded	accusing	across	action	active	actors
text_book1	0.003242	0.00000	0.003242	0.000000	0.004557	0.009113	0.00000	0.004557	0.00000	0.00000	0.009113	0.01367	0.004557	0.004557	0.003242	0.00000	0.00000	0.004557
text_book2	0.001622	0.00114	0.030014	0.005701	0.000000	0.000000	0.00114	0.000000	0.00342	0.00114	0.000000	0.00000	0.000000	0.000000	0.008923	0.00342	0.00114	0.000000

2 rows × 2401 columns

- Cosine Similarity Matrix -

	Book1	Book3
text_book1	1.000000	0.777991
text_book2	0.777991	1.000000

3. Book2 & Book3 -

- TF-IDF vector -

	abandon	abandoning	abashed	abbey	abbot	able	abodes	abolish	abortive	about	above	abraham	abroad	abruptly	absence	absent	absolute	absolute
text_book1	0.000302	0.000151	0.000151	0.000604	0.000302	0.004618	0.000151	0.000000	0.000151	0.020299	0.002685	0.000151	0.000453	0.000604	0.000755	0.000453	0.000755	0.0015
text_book2	0.000000	0.000000	0.000000	0.000000	0.000000	0.001647	0.000000	0.001157	0.000000	0.030467	0.004117	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0008

2 rows × 8785 columns

- Cosine Similarity Matrix -

	Book2	Book3
text_book1	1.00000	0.84377
text_book2	0.84377	1.00000

2. 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.

For Lemmatization of sentences first we have to use nltk.stem and nltk.corpus as Library and need to import WordNetLemmatizer and wordnet as a Package.

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
lemmatizer = WordNetLemmatizer()
def nltk2wn_tag(nltk_tag):
  if nltk_tag.startswith('J'):
    return wordnet.ADJ
  elif nltk_tag.startswith('V'):
    return wordnet.VERB
  elif nltk_tag.startswith('N'):
    return wordnet.NOUN
  elif nltk_tag.startswith('R'):
    return wordnet.ADV
  else:
    return None
```

Here, we have defined the function for Lemmatization of sentences. Here , we used pos.tag() function.

```
[ ]  def lemmatize_sentence(sentence):
       nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
       wn_tagged = map(lambda x: (x[0], nltk2wn_tag(x[1])), nltk_tagged)
       res_words = []
       for word, tag in wn_tagged:
         if tag is None:
           res_words.append(word)
         else:
           res_words.append(lemmatizer.lemmatize(word, tag))
       return " ".join(res_words)
```

After Lemmatization -

1. Book1 & Book2 -

- TF-IDF vector -

	abandon	abashed	abbey	abbot	able	abode	abortive	about	above	abraham	abroad	abruptly	absence	absent	absently	absolute	absolutely	absorb
text_book1	0.000000	0.00000	0.000000	0.000000	0.002914	0.00000	0.00000	0.002914	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.004095	0.005828	0.000000	0.000000
text_book2	0.000419	0.00014	0.000558	0.000279	0.004269	0.00014	0.00014	0.018764	0.003488	0.00014	0.000419	0.000558	0.000698	0.000419	0.000000	0.000496	0.001954	0.000279

2 rows × 6482 columns

- Cosine Similarity Matrix -

	book1	book2
text_book1	1.000000	0.907687
text_book2	0.907687	1.000000

2. Book1 & Book3 -

- TF-IDF vector -

	able	abolish	about	above	absently	absolute	absolutely	absurd	accept	access	accomplish	accomplished	accord	accuse	across	act	action	
text_book1	0.002904	0.000000	0.002904	0.000000	0.004081	0.008163	0.000000	0.004081	0.000000	0.008163	0.008163	0.004081	0.004081	0.004081	0.002904	0.000000	0.000000	0.
text_book2	0.001582	0.001111	0.029261	0.005557	0.000000	0.000000	0.001111	0.000000	0.004446	0.000000	0.000000	0.000000	0.000000	0.000000	0.008699	0.001111	0.003334	0

2 rows × 1983 columns

- Cosine Similarity Matrix -

	book1	book3
text_book1	1.000000	0.778361
text_book2	0.778361	1.000000

3. Book2 & Book3 -

- TF-IDF vector -

	abandon	abashed	abbey	abbot	able	abode	abolish	abortive	about	above	abraham	abroad	abruptly	absence	absent	absolute	absolutely	absorb
text_book1	0.000419	0.00014	0.000559	0.000279	0.004273	0.00014	0.000000	0.00014	0.018780	0.002484	0.00014	0.000419	0.000559	0.000698	0.000419	0.000698	0.001391	0.000279
text_book2	0.000000	0.00000	0.000000	0.000000	0.001595	0.00000	0.001121	0.00000	0.029516	0.003989	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000798	0.000000

2 rows × 6812 columns

- Cosine Similarity Matrix -

	book2	book3
text_book1	1.000000	0.848678
text_book2	0.848678	1.000000

Inferences

- Before Lemmatization we found that Book1 & Book2 are more similar than other combinations of Books.
- After Lemmatization we found that Book1 & Book2 are more similar than other combinations of Books.

Conclusion

We have learnt how to perform Semantic analysis, POS-Tagging, Named Entity Recognition, Performance evaluation in NLP and answered the questions in Project Round 2 successfully.

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