Link:- https://github.com/shail10/NLP-project

Language Learners

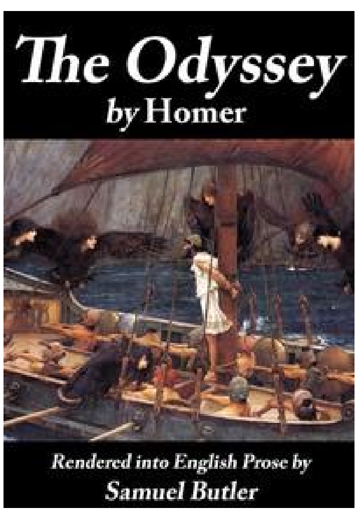
Pratik Gupta (19ucs047)
Raghav R Sharma (19ucs204)
Shail Kardani (19ucs217)

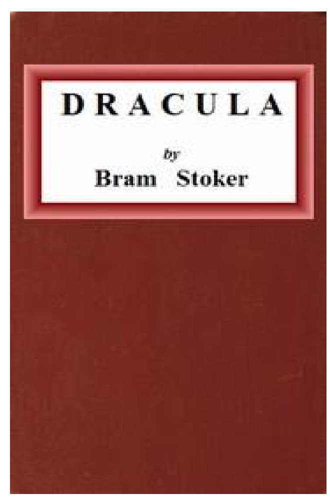
NLP PROJECT - ROUND 1

OVERVIEW

In this project, we perform text analysis on two of our chosen books from Gutenberg. “Dracula” and “Odyssey.” After that, we will apply POS Tagging on both books.

IMPORTED BOOKS





OBJECTIVES

• Import the text, let’s call it T1 and T2
• Perform simple text preprocessing steps and tokenising 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 our token.
• Remove the stop words from T1 and T2 and create a word cloud again. Comparison with word clout before the removal of stop words.
• Evaluate the word length and frequency relationship for both T1 and T2.
• Do PoS Tagging for both T1 and T2 using any tokenising of the four tag sets studied in the class and get the distribution of various tags.

Libraries Used:-

```
[ ] #imports
    import nltk
    from nltk.tokenize import word_tokenize #for tokenizing the words
    from nltk.stem import PorterStemmer #For potter stemming
    from nltk.stem import WordNetLemmatizer #for lemmatization/to get lemma of the word
    from nltk.corpus import stopwords       #To remove stop words
    from urllib.request import urlopen

    import inflect
    import re #for the purpose of regular expressions
    import matplotlib.pyplot as plt
    import seaborn as sns
    import pandas as pd
    from wordcloud import WordCloud
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
[ ] nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
[ ] p = inflect.engine()
    ps = PorterStemmer()
    lemmatizer = WordNetLemmatizer()
```

Data preprocessing

1) Importing the text using the urlib

```
[ ] #URLs for Books
    url1 = 'https://www.gutenberg.org/cache/epub/1727/pg1727.txt' #Odyssey
    url2 = 'https://www.gutenberg.org/files/345/345-0.txt'        #Dracula
```

```
[ ] #first we will read both the books and print few initial lines
    T1_odyssey = urlopen(url1).read()
    T2_dracula = urlopen(url2).read()
```

```
[ ] T1_odyssey
```

```
b'\xef\xbb\xbfThe Project Gutenberg eBook of The Odyssey, by Homer\r\n\r\n\rThis eBook is for the use of anyone anywhere in the United States and\r\nmost other parts of the world at no cost and with almost no
```

```
[ ] T2_dracula
```

```
b'\xef\xbb\xbfThe Project Gutenberg eBook of Dracula, by Bram Stoker\r\n\r\n\rThis eBook is for the use of anyone anywhere in the United States and\r\nmost other parts of the world at no cost and with almost
```

```
[ ] T1_odyssey = T1_odyssey.decode('utf-8')
```

```
[ ] T2_dracula = T2_dracula.decode('utf-8')
```

Performing simple text-preprocessing steps and tokenising both T1_odeyssey and T2_dracula

We are removing all the unnecessary text from the files.

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

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

```
[ ] T1_odyssey = discard_from_odessey(T1_odyssey)

    Discarding Before -  772
    Discarding After -  691478
```

```
[ ] T2_dracula = discard_from_dracula(T2_dracula)

    Discarding Before -  805
    Discarding After -  862449
```

```
[ ] T1_odyssey

    'THE ODYSSEY ***\r\n\r\n[Illustration]\r\n\r\n\r\n\r\n\r\n\r\nThe Odyssey\r\n\r\nby Homer\r\n\r\nrendered into English prose for the use of those who cannot read the\r\noriginal\r\n\r\nContents\r\n\r\n PREFACE TO FIRST EDITION\r\n PREFACE TO SECOND EDITION\r\n THE ODYSSEY\r\n BOOK I.\r\n BOOK II.\r\n BOOK III.\r\n BOOK IV.\r\n BOOK V.\r\n BOOK VI.\r\n BOOK VII.\r\n BOOK VIII.\r\n BOOK IX.\r\n BOOK X.\r\n BOOK XI.\r\n BOOK XII.\r\n BOOK XIII.\r\n BOOK XIV.\r\n BOOK XV.\r\n BOOK XVI.\r\n BOOK XVII.\r\n BOOK XVIII.\r\n BOOK XIX.\r\n BOOK XX.\r\n BOOK XXI.\r\n BOOK XXII.\r\n BOOK XXIII.\r\n BOOK XXIV.\r\n FOOTNOTES:\r\n\r\n\r\n\r\nAL PROFESSORE\r\nCAV. BIAGIO INGROIA,\r\nPREZIOSO ALLEATO\r\nL'AUTORE RICONOSCENTE.\r\n\r\n\r\n\r\n\r\n\r\n\r\nPREFACE TO FIRST EDITION\r\n\r\n\r\n\r\nThis translation is intended to supplement a work entitled "The\r\nAuthoress of the Odyssey", which I published in 1897. I could not give\r\nthe whole "Odyssey" in that book without making it unwieldy, I\r\ntherefore epitomised my tran…'
```

```
[ ] T2_dracula

    'DRACULA ***\r\n\r\n\r\n\r\n\r\n               DRACULA\r\n\r\n\r\n\r\n\r\n\r\n               DRACULA\r\n\r\n               _by_\r\n\r\n
    Bram Stoker\r\n\r\n               [Illustration: colophon]\r\n\r\n               NEW YORK\r\n\r\n               GROSSET & DUNLAP\r\n\r\n
    _Publishers_\r\n\r\n      Copyright, 1897, in the United States of America, according\r\n               to Act of Congress, by Bram Stoker\r\n\r\n               [_All rights reserved._]\r\n\r
    \n               PRINTED IN THE UNITED STATES\r\n               AT\r\n               THE COUNTRY LIFE PRESS, GARDEN CITY, N.Y.\r\n\r\n\r\n\r\n\r\n
    TO\r\n\r\n               MY DEAR FRIEND\r\n\r\n               HOMMY-BEG\r\n\r\n\r\n\r\n\r\n\r\nContents\r\n\r\nCHAPTER I. Jonathan Harker's …'
```

```
[ ] T1_odyssey = T1_odyssey.lower()
    T2_dracula = T2_dracula.lower()
```

Using regular expression to decontract certain words to standard form for better text understanding

```
def transforming(text):
    #removing URL
    text = re.sub(r"http\s+", "", text)

    #Decontracting most common words
    text = re.sub(r"couldn\'t", "could not", text)
    text = re.sub(r"aren\'t", "are not", text)
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)
    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
```

```
[ ] T1_odyssey = transforming(T1_odyssey)
    T2_dracula = transforming(T2_dracula)
```

Tokenising both the book

```
[ ] def tokenizing_book(text):
        tokens = text.split()
        final_word_bag = []

        for word in tokens:
            if word.isdigit():
                converted_word = p.number_to_words(word)
                final_word_bag.append(converted_word)
            else:
                final_word_bag.append(word)

        return ' '.join(final_word_bag)
```

```
[ ] T1_odyssey = tokenizing_book(T1_odyssey)
    T2_dracula = tokenizing_book(T2_dracula)
```

```
[ ] T1_odyssey
```

```
'the odyssey *** [illustration] the odyssey by homer rendered into english prose for the use of those who cannot read the original contents preface to first edition preface to second edition the odyssey bo
ok i. book ii. book iii. book iv. book v. book vi. book vii. book viii. book ix. book x. book xi. book xii. book xiii. book xiv. book xv. book xvi. book xvii. book xviii. book xix. book xx. book xxi. book
xxii. book xxiii. book xxiv. footnotes: al professore cav. biagio ingroia, prezioso alleato l'autore riconoscente. preface to first edition this translation is intended to supplement a work entitled "the a
uthoress of the odyssey", which i published in 1897. i could not give the whole "odyssey" in that book without making it unwieldy, i therefore epitomised my translation, which was already completed and whi
ch i now publish in full. i shall not here argue the two main points dealt with in the work just mentioned; i have nothing either to add to, or to withdraw from, what i have there w...'
```

```
[ ] T2_dracula
```

```
'dracula *** dracula dracula _by_ bram stoker [illustration: colophon] new york grosset & dunlap _publishers_ copyright, 1897, in the united states of america, according to act of congress, by bram stoker
[_all rights reserved._] printed in the united states at the country life press, garden city, n.y. to my dear friend hommy-beg contents chapter i. jonathan harker's journal chapter ii. jonathan harker's jo
urnal chapter iii. jonathan harker's journal chapter iv. jonathan harker's journal chapter v. letters—lucy and mina chapter vi. mina murray's journal chapter vii. cutting from "the dailygraph," eight augus
t chapter viii. mina murray's journal chapter ix. mina murray's journal chapter x. mina murray's journal chapter xi. lucy westenra's diary chapter xii. dr. seward's diary chapter xiii. dr. seward's diary c
hapter xiv. mina harker's journal chapter xv. dr. seward's diary chapter xvi. dr. seward's diary chapter xvii. dr. seward's diary chapter xviii. dr. seward's diary chapter xix. jona...'
```

Removing non-alphabetic characters

```
[ ] def remove_non_alpha(text):
        tokens = text.split()
        final_word_bag = [word for word in tokens if word.isalpha()]
        return ' '.join(final_word_bag)
```

```
[ ] T1_odyssey = remove_non_alpha(T1_odyssey)
    T2_dracula = remove_non_alpha(T2_dracula)
```

Performing Lemmatization

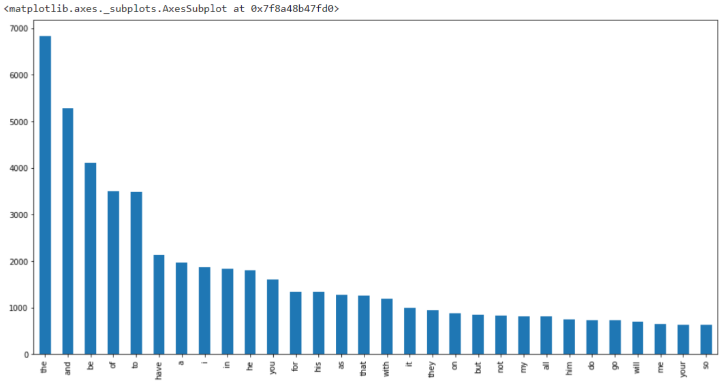
```
[ ] def lemmatize_word(text):
        word_tokens = text.split()
        lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word_tokens]
        return ' '.join(lemmas)
```

```
[ ] T1_odyssey = lemmatize_word(T1_odyssey)
    T2_dracula = lemmatize_word(T2_dracula)
```

Analysing the frequency distribution of tokens in T1_odessey and T2_dracula

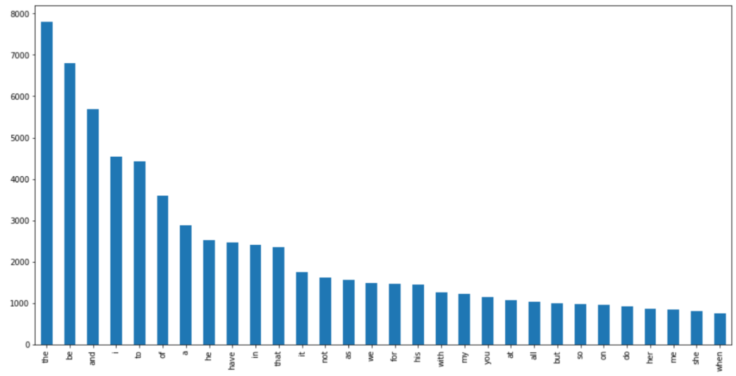
```
[ ] plt.figure(figsize=(16,8))
    tokens = word_tokenize(T1_odyssey)
    pd.Series(tokens).value_counts()[:30].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a48b47fd0>
```



```
[ ] plt.figure(figsize=(16,8))
    tokens = word_tokenize(T2_dracula)
    pd.Series(tokens).value_counts()[:30].plot(kind='bar')
```

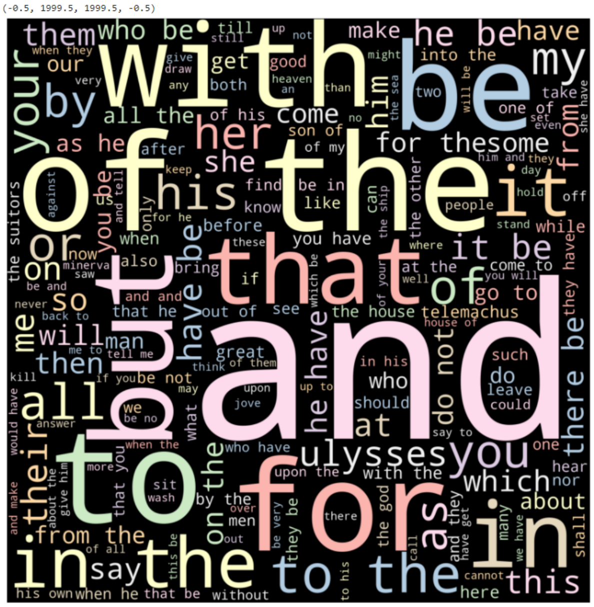
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a47cbd490>
```



Generating Word Cloud

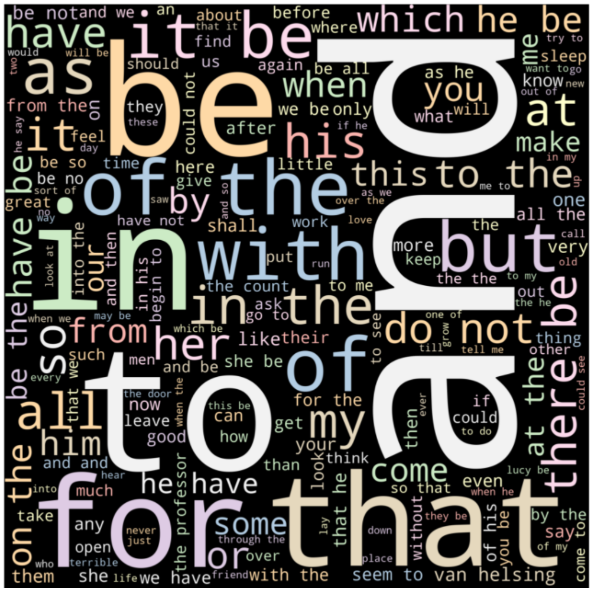
```
plt.figure(figsize=(15,15))
wordcloud = WordCloud(width = 2000, height = 2000,background_color ='black',stopwords = [], colormap='Pastel1').generate(T1_odyssey)
plt.imshow(wordcloud)
plt.axis('off')
```

```
(-0.5, 1999.5, 1999.5, -0.5)
```



```python
plt.figure(figsize=(18,15))
wordcloud = WordCloud(width = 2000, height = 2000,background_color ='black',stopwords = [], colormap='Pastel1').generate(T2_dracula)
plt.imshow(wordcloud)
plt.axis('off')
```

```
(-0.5, 1999.5, 1999.5, -0.5)
```



From the above word cloud visualisation, we can infer that the most frequently are mainly stopped words like ‘to’, ‘of’, ‘be’,’ the’. These words do not contribute to the meaning of the sentence. These stop words can easily be removed, resulting in a better word cloud.

Removing the STOPWORDS and again generating Word Cloud to identify the potential differences between the word clouds before and after removing the STOPWORDS.

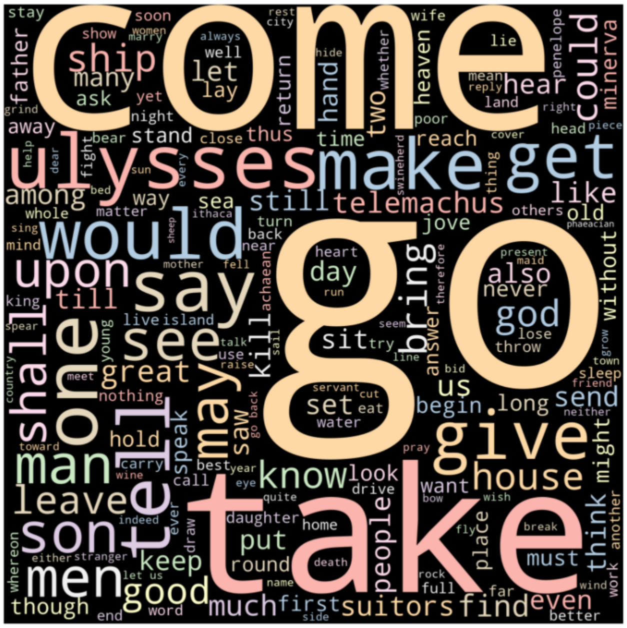
```python
stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    tokens = word_tokenize(text)
    tokens = [words for words in tokens if words not in stop_words]
    return ' '.join(tokens)
```

```python
T1_odyssey_ = remove_stopwords(T1_odyssey)
T2_dracula_ = remove_stopwords(T2_dracula)
```

```python
plt.figure(figsize=(15,15))
wordcloud = WordCloud(width = 2000, height = 2000,background_color ='black',stopwords = [], colormap='Pastel1').generate(T1_odyssey_)
plt.imshow(wordcloud)
plt.axis('off')
```

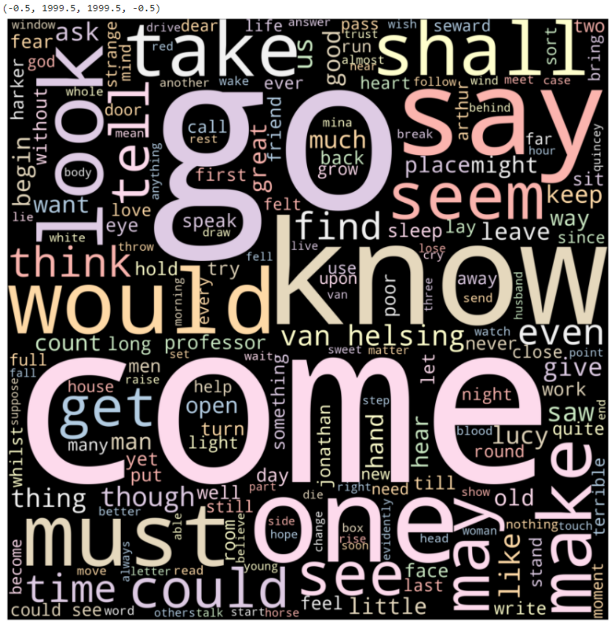
```
plt.figure(figsize=(15,15))
wordcloud = WordCloud(width = 2000, height = 2000,background_color ='black',stopwords = [], colormap='Pastel1').generate(T1_odyssey_)
plt.imshow(wordcloud)
plt.axis('off')
```

```
(-0.5, 1999.5, 1999.5, -0.5)
```




```
plt.figure(figsize=(18,15))
wordcloud = WordCloud(width = 2000, height = 2000,background_color ='black',stopwords = [], colormap='Pastel1').generate(T2_dracula_)
plt.imshow(wordcloud)
plt.axis('off')
```

```
(-0.5, 1999.5, 1999.5, -0.5)
```



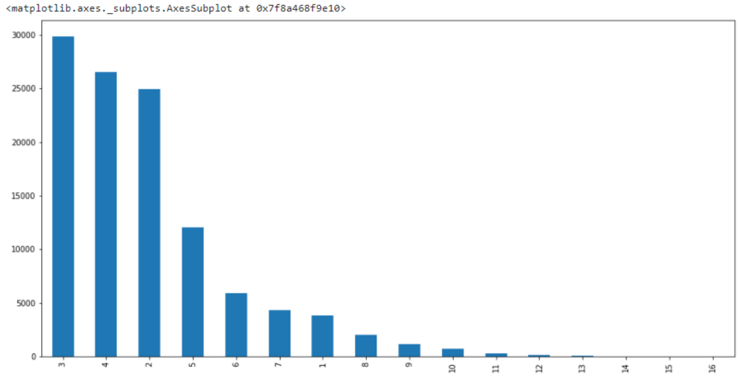
The words with no meaning are removed from the text after removing all the stop words.
Now more relevant words can be displayed with word clouds with words like ‘would’, ‘know’, ‘come’,’take’.

Evaluating the relationship between the word length and frequency for both T1_odessey and T2_dracula

Before removing StopWords

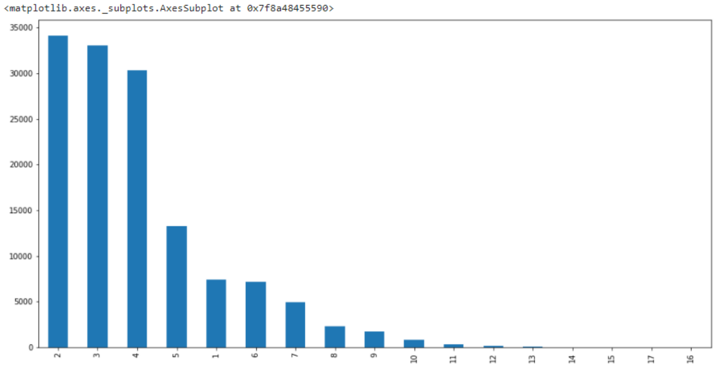
```
# For T1_odessey
plt.figure(figsize=(16,8))
tokens = word_tokenize(T1_odyssey)
length = [len(words) for words in tokens]
pd.Series(length).value_counts()[:30].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a468f9e10>
```




```
# For T2_dracula
plt.figure(figsize=(16,8))
tokens = word_tokenize(T2_dracula)
length = [len(words) for words in tokens]
pd.Series(length).value_counts()[:30].plot(kind='bar')
```

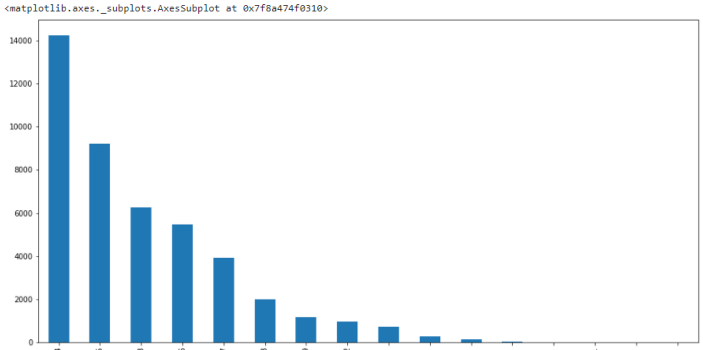
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a48455590>
```



After removing StopWords

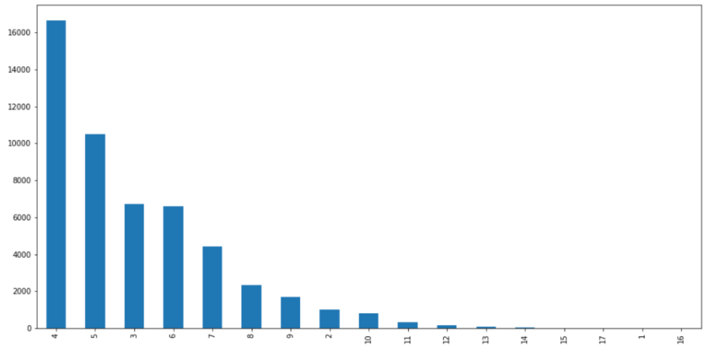
```
# For T1_odessey
plt.figure(figsize=(16,8))
tokens = word_tokenize(T1_odyssey_)
length = [len(words) for words in tokens]
pd.Series(length).value_counts()[:30].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a474f0310>
```



```
# For T2_dracula
plt.figure(figsize=(16,8))
tokens = word_tokenize(T2_dracula_)
length = [len(words) for words in tokens]
pd.Series(length).value_counts()[:30].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a48172910>
```



The number of words of 2 and 3 has been decreased after removing stopwords. This is since stopping words like ‘be’, ‘of” have been removed. Apart from that, new comments have emerged, the stop words of length 3,4,5.

POS Tagging

```
[ ] from collections import Counter
```

```
[ ] def tag_treebank(text):
        tokenized=nltk.word_tokenize(text)
        tagged=nltk.pos_tag(tokenized)
        return tagged
```

```
[ ] def get_counts(tags):
        counts = Counter( tag for word,  tag in tags)
        return counts
```

```
[ ] nltk.download('averaged_perceptron_tagger')

    [nltk_data] Downloading package averaged_perceptron_tagger to
    [nltk_data]     /root/nltk_data...
    [nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
    True
```

```
[ ] odessey_tags=tag_treebank(T1_odyssey)
    odessey_pos_count=get_counts(odessey_tags)
    dracula_tags=tag_treebank(T2_dracula)
    dracula_pos_count=get_counts(dracula_tags)
```

```
[ ] len(odessey_pos_count)
```

```
32
```

```
[ ] odessey_pos_count
```

```
Counter({'$': 2,
         'CC': 6782,
         'CD': 774,
         'DT': 11036,
         'EX': 189,
         'FW': 31,
         'IN': 16283,
         'JJ': 7104,
         'JJR': 297,
         'JJS': 201,
         'MD': 2551,
         'NN': 20058,
         'NNP': 50,
         'NNS': 1344,
         'PDT': 248,
         'PRP': 8903,
         'PRP$': 2649,
         'RB': 5066,
         'RBR': 93,
         'RBS': 65,
         'RP': 668,
         'TO': 3479,
         'VB': 12382,
         'VBD': 755,
         'VBG': 204,
         'VBN': 2292,
         'VBP': 5173,
         'VBZ': 621,
         'WDT': 653,
         'WP': 959,
         'WP$': 43,
         'WRB': 981})
```

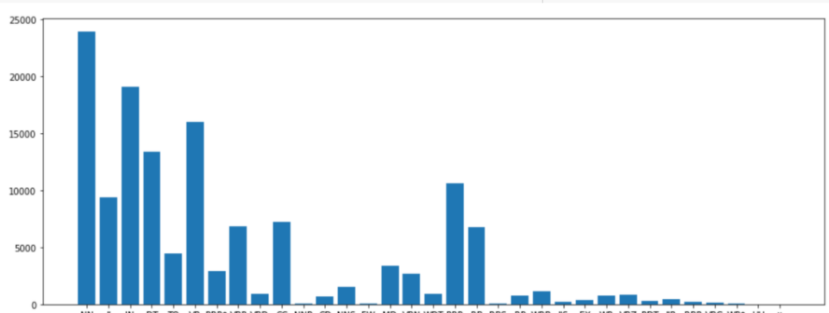
```
[ ] len(dracula_pos_count)
```

```
33
```

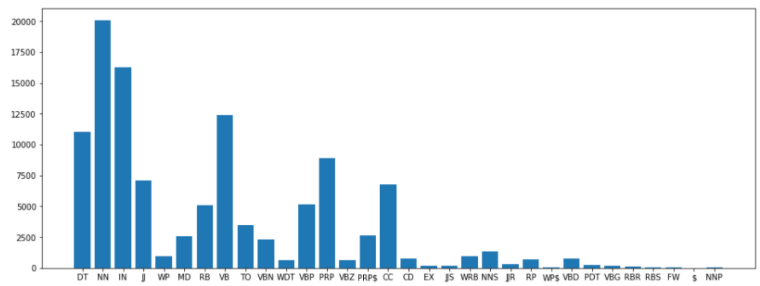
```
[ ] dracula_pos_count
```

```
Counter({"''": 1,
         'CC': 7210,
         'CD': 650,
         'DT': 13401,
         'EX': 362,
         'FW': 50,
         'IN': 19083,
         'JJ': 9373,
         'JJR': 426,
         'JJS': 210,
         'MD': 3394,
         'NN': 23928,
         'NNP': 26,
         'NNS': 1504,
         'PDT': 310,
         'PRP': 10607,
         'PRP$': 2915,
         'RB': 6733,
         'RBR': 192,
         'RBS': 47,
         'RP': 776,
         'TO': 4429,
         'UH': 2,
         'VB': 16014,
         'VBD': 886,
         'VBG': 102,
         'VBN': 2685,
         'VBP': 6873,
         'VBZ': 795,
         'WDT': 894,
         'WP': 719,
         'WP$': 30,
         'WRB': 1133})
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(16,6))
plt.bar(range(len(dracula_pos_count)), list(dracula_pos_count.values()), align='center')
plt.xticks(range(len(dracula_pos_count)), list(dracula_pos_count.keys()))


plt.show()
```



```python
import matplotlib.pyplot as plt

plt.figure(figsize=(16,6))
plt.bar(range(len(odessey_pos_count)), list(odessey_pos_count.values()), align='center')
plt.xticks(range(len(odessey_pos_count)), list(odessey_pos_count.keys()))


plt.show()
```



We can infer that the highest occurring tag is ‘NN’, and ‘Determinant’ Tags are on the lower frequency side. This is primarily due to the removal of stopwords before POS Tagging.

CONCLUSION:-

We have learnt how to perform Text Processing, Tokenization and answer everything we asked of us using proper imports and libraries like pandas, matplotlib, NLTK.

NLP PROJECT - ROUND 2

OBJECTIVES

First Part -

1.) Find the nouns and verbs in both novels. Get the immediate categories (parent) that these words fall under in the WordNet.
2) 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 -

1) Recognise all Persons, Locations, organisations in the book. For this you have to do two steps:
(a) First recognise all the entities and then
(b) 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 manual labelling and then compare your result with it. Present the accuracy with the F1 score here.

Third Part -

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.
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.

Libraries Used:-

Urllib - Used to fetch test data from Gutenberg text URL.

NLTK - We have used this library for tokenization, POS tagging, using wordnet, etc.

Re - This library generates regular expressions that are used to remove URLs and other unwanted passages from the text

Matplotlib, seaborn - Plotting libraries used for plotting bar graphs and visualizing the data.

Spacy - To for entity recognition in text

Numpy = Provides support with large multi-dimensional arrays to get frequency distributions of nouns and verbs.

Task 1-

<u>Finding nouns and verbs in both the novel</u>

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(book):
        # In pos tagging nounds have the NN part. For examples prorper nouns are represented as NNP
        is_noun = lambda pos_tags: pos_tags[:2] == 'NN'
        #Next we will tokenize the book
        tokens = word_tokenize(book)
        #Filter the book and find all the nouns and append them to single list called nouns
        nouns = [word for (word, pos_tags) in nltk.pos_tag(tokens) if is_noun(pos_tags)]
        return nouns
```

```
[ ] noun_odyssey = noun(T1_odyssey)
    noun_dracula = noun(T2_dracula)
```

```
[ ] print('Number of Nouns in the book Odyssey are: ',len(noun_odyssey))
    print('Number of Nouns in the book Dracula are: ',len(noun_dracula))

    Number of Nouns in the book Odyssey are:  19423
    Number of Nouns in the book Dracula are:  22486
```

```
[ ] def verb(book):
        is_verb = lambda pos_tags: pos_tags[:1] == 'V'
        tokens = nltk.word_tokenize(book)
        verbs = [word for (word, pos_tags) in nltk.pos_tag(tokens) if is_verb(pos_tags)]
        return verbs
```

```
[ ] verb_odyssey = verb(T1_odyssey)
    verb_dracula = verb(T2_dracula)
```

```
[ ] print('Number of Verbs in the book Odyssey are: ',len(verb_odyssey))
    print('Number of Verbs in the book Dracula are: ',len(verb_dracula))

    Number of Verbs in the book Odyssey are:  20769
    Number of Verbs in the book Dracula are:  26312
```

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

To retrieve the categories that each noun and verb belong to will use sysnet. Sysnet is a grouping of synonyms of words that expresses the same concept. 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.

```
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
```

```
[65] noun_syn1=synset(noun_odyssey)
     noun_syn2=synset(noun_dracula)
     verb_syn1=synset(verb_odyssey)
     verb_syn2=synset(verb_dracula)
```

Hence noun_syn1, noun_syn2, verbsyn_1,verb_syn2 are 2-dimensional lists that 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.

```
noun_odyssey[57]

'question'

[67] noun_syn1[57][:]

['noun.communication',
 'noun.communication',
 'noun.communication',
 'noun.attribute',
 'noun.communication',
 'noun.communication',
 'verb.communication',
 'verb.communication',
 'verb.communication',
 'verb.communication',
 'verb.communication']
```

Here, we have attempted to find out the Hypernyms of each categories Verb and Noun. Sometimes we have to go many levels up the hypernyms in order to get what we want.

```
[ ] def hypernym(words):
        hyper=dict()
        for word in words:
            h = str(wn.synsets(word))
            if word not in hyper:
                hyper[word] = []
                hyper[word].append(h)
        return hyper
```

```
hp_noun_odyssey = hypernym(noun_odyssey)
hp_noun_dracula = hypernym(noun_dracula)
hp_verb_odyssey = hypernym(verb_odyssey)
hp_verb_dracula = hypernym(verb_dracula)
```

```
[ ] n = noun_odyssey[60]
    hp_noun_odyssey[n]

    ["[Synset('seashore.n.01'), Synset('coast.n.02'), Synset('coast.n.03'), Synset('slide.n.05'), Synset('coast.v.01')]"]
```

```
[ ] n

    'coast'
```

Getting the frequency of each category for each noun and verb and plotting histogram for the same.

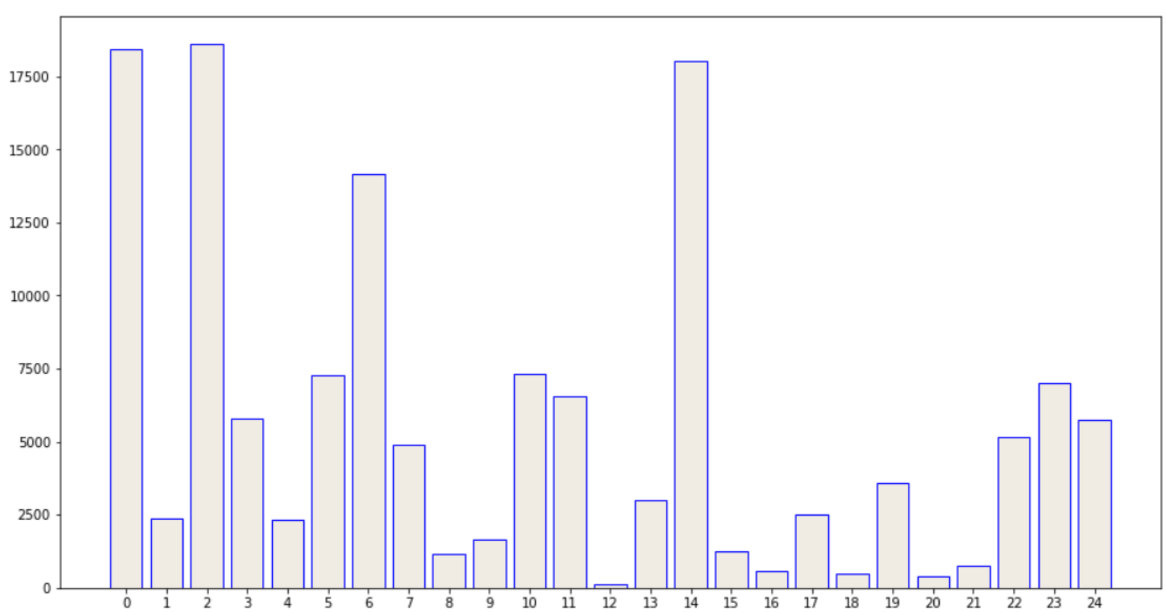
```
#GIVES TOTAL NOUN LEXNAMES AND TOTAL VERB LEXNAMES FOR FREQUENCY DISTRIBUTIONS
def all_synsets(no,ve):
    nouns=[]
    verbs=[]
    for word in no:
        for synset in wn.synsets(word):
            if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname())):
                nouns.append(synset.lexname())
            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_super_odyssey,verb_super_odyssey=all_synsets(noun_odyssey,verb_odyssey)
    noun_super_dracula,verb_super_dracula=all_synsets(noun_dracula,verb_dracula)
```

```
[ ] print(len(noun_super_odyssey))
    print(len(verb_super_odyssey))
    print(len(noun_super_dracula))
    print(len(verb_super_dracula))

    139276
    362308
    165696
    428402
```

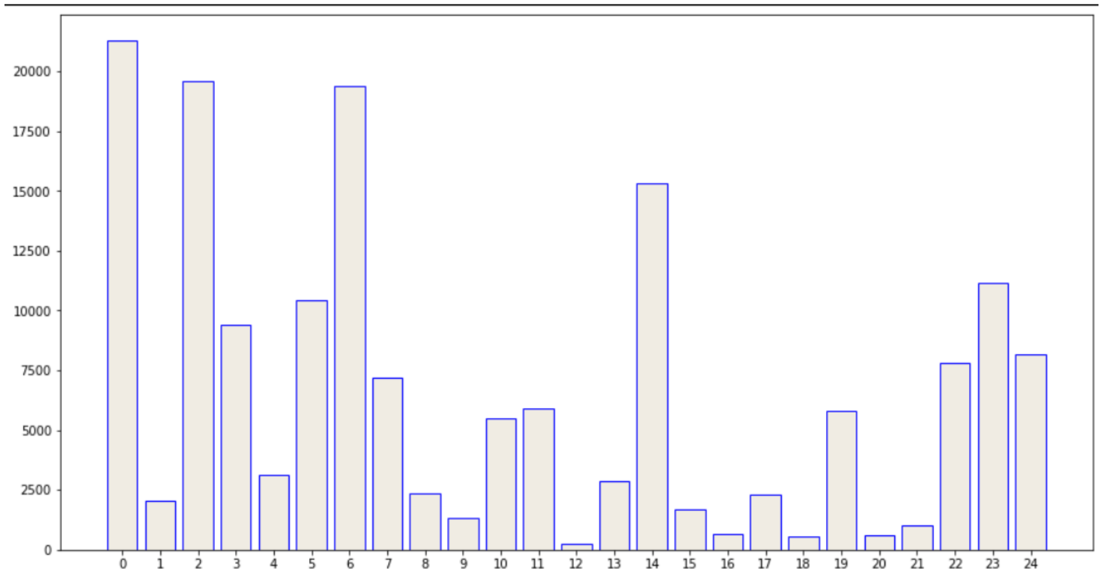
A) i)For Odyssey (noun):

```
labels1, counts = np.unique(noun_super_odyssey,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center',color=['#F0ECE3'],edgecolor='blue')
plt.xticks(ticks, range(len(labels1)))
```



ii) For Dracula (Noun):

```
labels1, counts = np.unique(noun_super_dracula,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center',color=['#F0ECE3'],edgecolor='blue')
plt.xticks(ticks, range(len(labels1)))
```

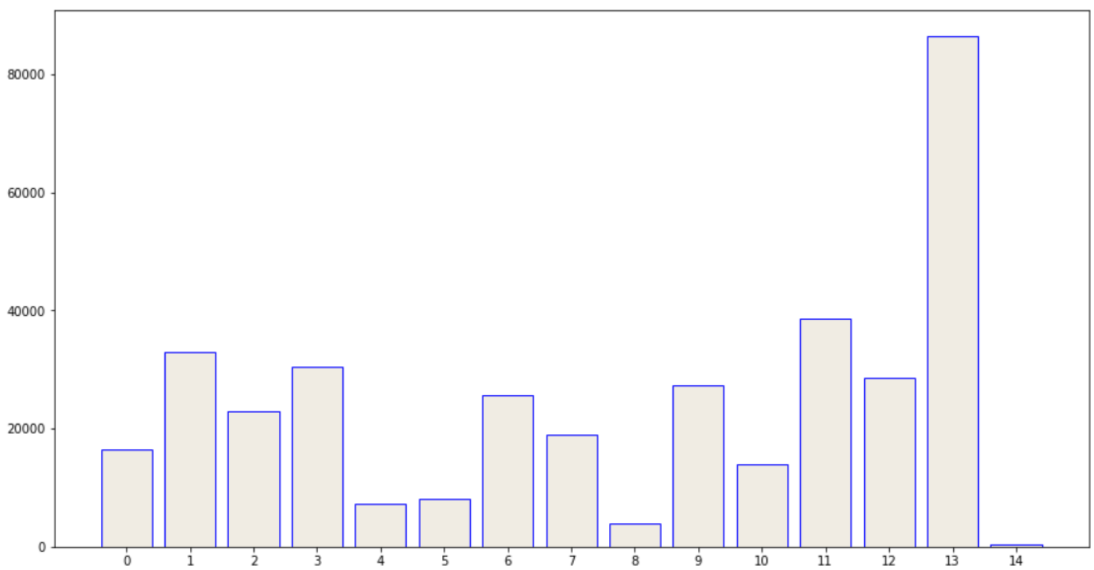


Labels for noun:

```
['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']
```

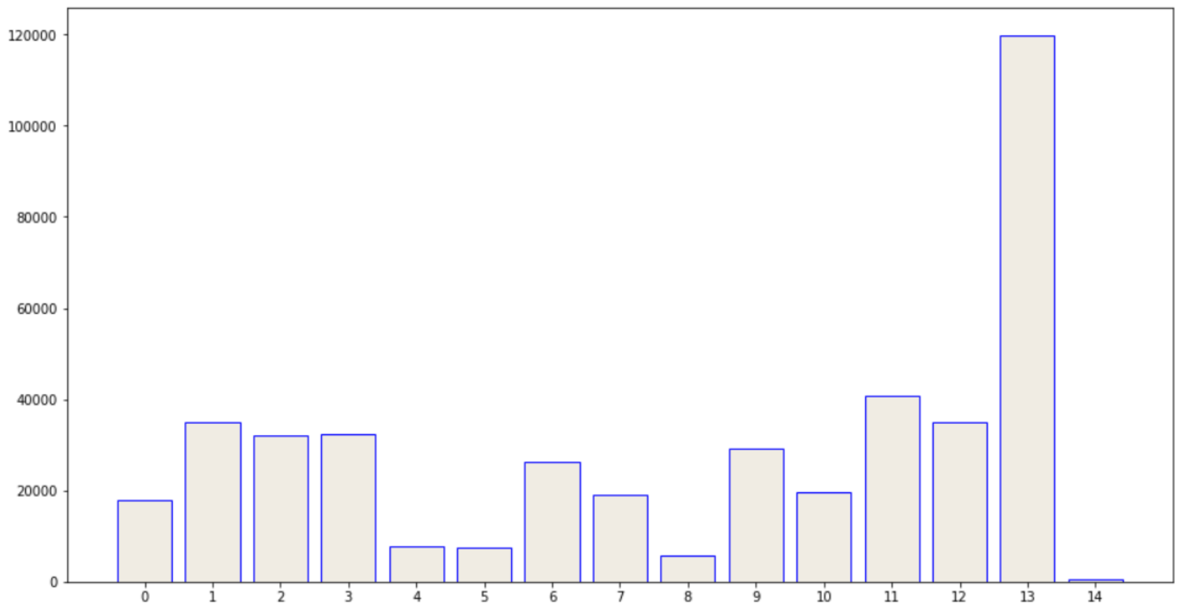
B) i)For Odyssey (Verb):

```
labels2, counts = np.unique(verb_super_odyssey,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center',color=['#F0ECE3'],edgecolor='blue')
plt.xticks(ticks, range(len(labels2)))
```



ii)For Dracula (Verb):

```
[78] labels2, counts = np.unique(verb_super_dracula,return_counts=True)
     ticks = range(len(counts))
     plt.figure(figsize=(15,8))
     plt.bar(ticks,counts, align='center',color=['#F0ECE3'],edgecolor='blue')
     plt.xticks(ticks, range(len(labels2)))
```



Labels for Verbs:

```
['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']
```

Task 2-

In task 2, we performed Named Entity Recognition using Spacy. Spacy's named entity recognition has been trained on the OntoNotes 5 corpus and it supports a wide range of numerical and named entities like person, organization, language, event, etc.

First we import Spacy and get named entities in both the books:

```
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
```

```
[8] nlp = en_core_web_sm.load()
```

```
odyssey = nlp(T1_odyssey)
dracula = nlp(T2_dracula)
print('There are total '+str(len(odyssey.ents))+' entities in Odyssey and '+str(len(dracula.ents))+' in Dracula')
```

```
There are total 4368 entities in Odyssey and 4948 in Dracula
```

Next we create a function, to extract the entities which are annotated as Person, Organization, and Location.

```
[9] 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
```

Printing out the number of entities that are annotated as Person, Organization, and Location.

```
person1,org1,location1=entity_recognition(T1_odyssey)
person2,org2,location2=entity_recognition(T2_dracula)
print("number of person entities in Odyssey and Dracula respectively are "+str(len(person1))+" and "+str(len(person2)))
print("number of organization entities in Odyssey and Descularespectively are "+str(len(org1))+" and "+str(len(org2)))
print("number of location entities in Odyssey and Dracula respectively are "+str(len(location1))+" and "+str(len(location2)))
```

```
number of person entities in Odyssey and Dracula respectively are 362 and 345
number of organization entities in Odyssey and Descularespectively are 270 and 267
number of location entities in Odyssey and Dracula respectively are 212 and 186
```

Task 3

We have taken Pride and Prejudice as our third book.

To do the third task, first, we import raw books without any preprocessing.

```
url1 = 'https://www.gutenberg.org/cache/epub/1727/pg1727.txt' #Odyssey
url2 = 'https://www.gutenberg.org/files/345/345-0.txt'        #Dracula
url3 = 'https://www.gutenberg.org/files/1342/1342-0.txt' ##Pride and Prejudice
```

```
[30] T1_odyssey= urlopen(url1).read()
     T2_dracula = urlopen(url2).read()
     T3_PNP = urlopen(url3).read()
```

```
[31] T1_odyssey = T1_odyssey.decode('utf-8')
     T2_dracula = T2_dracula.decode('utf-8')
     T3_PNP = T3_PNP.decode('utf-8')
```

We install the necessary libraries for finding TF-IDF and cosine similarity values between the books and using these libraries we find the aforementioned values respectively.

```
[32] from scipy.spatial import distance
     from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[33] Doc1 = T1_odyssey
     Doc2 = T2_dracula
     Doc3 = T3_PNP
```

```
# Intialize TfidfVectorizer
Tfidf_vect = TfidfVectorizer()
# Fit The Corpus To TfidfVectorizer
Tfidf_vect.fit([Doc1, Doc2,Doc3])
# Tf-Idf Representation Of Document1
Tfidf1 = Tfidf_vect.transform([Doc1])
print("Tf-Idf Representation Of Odyssey:- ", Tfidf1.toarray())
# Tf-Idf Representation Of Document2
Tfidf2 = Tfidf_vect.transform([Doc2])
print("Tf-Idf Representation Of Dracula:- ", Tfidf2.toarray())
# Tf-Idf Representation Of Document3
Tfidf3 = Tfidf_vect.transform([Doc3])
print("Tf-Idf Representation Of Pride and Prejudice:- ", Tfidf3.toarray())
```

```
Tf-Idf Representation Of Odyssey:-  [[0.0001603  0.0001603  0.00027141 ... 0.00027141 0.0001357  0.0001357 ]]
Tf-Idf Representation Of Dracula:-  [[0.00013557 0.00040672 0.         ... 0.         0.         0.        ]]
Tf-Idf Representation Of Pride and Prejudice:-  [[9.16222103e-05 1.83244421e-04 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]]
```

```
[35] cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf2.toarray())
     print('Cosine_similarity of Odyssey and Dracula  =',cosine_similarity)
```

```
Cosine_similarity of Odyssey and Dracula  = 0.9647228764011149
```

```
[36] cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf3.toarray())
     print('Cosine_similarity of Odyssey and PNP =',cosine_similarity)
```

```
Cosine_similarity of Odyssey and PNP = 0.9077758928203683
```

```
[37] cosine_similarity= 1- distance.cosine (Tfidf2.toarray(),Tfidf3.toarray())
     print('Cosine_similarity of Dracula and PNP =',cosine_similarity)
```

```
Cosine_similarity of Dracula and PNP = 0.925848002366286
```

Now, we apply lemmatization to each book and find out TF-IDF and cosine similarities between them respectively.

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

```
[39] Doc1 = lemmatize_word(T1_odyssey)
     Doc2 = lemmatize_word(T2_dracula)
     Doc3 = lemmatize_word(T3_PNP)
```

```
[40] # Intialize TfidfVectorizer
     Tfidf_vect = TfidfVectorizer()
     # Fit The Corpus To TfidfVectorizer
     Tfidf_vect.fit([Doc1, Doc2,Doc3])
     # Tf-Idf Representation Of Document1
     Tfidf1 = Tfidf_vect.transform([Doc1])
     print("Tf-Idf Representation Of Odyssey: ", Tfidf1.toarray())
     # Tf-Idf Representation Of Document2
     Tfidf2 = Tfidf_vect.transform([Doc2])
     print("Tf-Idf Representation Of Dracula: ", Tfidf2.toarray())
     # Tf-Idf Representation Of Document3
     Tfidf3 = Tfidf_vect.transform([Doc3])
     print("Tf-Idf Representation Of Pride and Prejudice : ", Tfidf3.toarray())
```

```
Tf-Idf Representation Of Odyssey:  [[0.00015038 0.00015038 0.00025461 ... 0.00025461 0.00012731 0.00012731]]
Tf-Idf Representation Of Dracula:  [[0.00012423 0.0003727  0.         ... 0.         0.         0.        ]]
Tf-Idf Representation Of Pride and Prejudice :  [[8.11506557e-05 1.62301311e-04 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]]
```

```
[41] cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf2.toarray())
     print('Cosine_similarity of Odyssey and Dracula =',cosine_similarity)
```

```
Cosine_similarity of Odyssey and Dracula = 0.9667378683060702
```

```
[42] cosine_similarity= 1- distance.cosine (Tfidf1.toarray(),Tfidf3.toarray())
     print('Cosine_similarity of Odyssey and PNP=',cosine_similarity)
```

```
Cosine_similarity of Odyssey and PNP= 0.913038746067504
```

```
[43] cosine_similarity= 1- distance.cosine (Tfidf2.toarray(),Tfidf3.toarray())
     print('Cosine_similarity of Dracula and PNP =',cosine_similarity)
```

```
Cosine_similarity of Dracula and PNP = 0.9368810668148123
```

Result of task 3-

- Odyssey and Dracula are most similar among all pairs before and after lemmatization and Odyssey and Pride and Prejudice are the least similar.
- As we can infer when we applied lemmatization cosine similarity between two books increases.