Round1

December 15, 2019

Team Name: ASH
Team Members: Ayush Gupta(17ucs042), Harshit Garg(17ucs064), Shubhi Rustagi(17ucs158)

1 Book Details

Book Title - Off Sandy Hook, and other stories Author's Name - Richard Dehan Subjects - Fiction, Short stories Number of Sentences - 11202 Number of Words - 98822 Character set encoding - UTF-8

This is part of NLP-Projects Phase-1. In this project, we have used the following libraries and toolkits: * **nltk** - A leading platform in Python designed to provide easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries. These libraries are used for classification, tokenisation, lemmatisation etc. * **string** - This module consists of basic operations that can be performed on string. * **WordNetLemmatizer** - It is a package for lemmatization. Lemmatization is the process of grouping together the words having similar meaning to a particular word. * **stopwords** - Set of commonly used words that are mostly ignored in text processing. * **matplotlib** - Library used for data visualisation. In other words, it is used for plotting the relationship among different components. * **re** - Library used for operations on regular expressions.

To begin, we import all the necssary libraries.

[2]:
```
import nltk
import string
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import re
from nltk.corpus import brown
```

Now, open the book and read it in a variable *file*. Further, we read the contents of the book in another variable *T*. We declare a variable *test* equal to *T* in case we require the original text later in the process as *T* will undergo text processing.

[7]:
```
#Reading Book
file = open("Sandy_Hook_NLP.txt",encoding="utf8")

# variable T
T = file.read()
```

```python
# variable test
test=T

# Printing the contents of the book stored in T
print(T[:1000])
```

```
Project Gutenberg's Off Sandy Hook and other stories, by Richard Dehan

This eBook is for the use of anyone anywhere in the United States and most
other parts of the world at no cost and with almost no restrictions
whatsoever.  You may copy it, give it away or re-use it under the terms of
the Project Gutenberg License included with this eBook or online at
www.gutenberg.org.  If you are not located in the United States, you'll have
to check the laws of the country where you are located before using this ebook.

Title: Off Sandy Hook and other stories

Author: Richard Dehan

Release Date: October 8, 2019 [EBook #60452]

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK OFF SANDY HOOK AND OTHER STORIES ***




Produced by Richard Tonsing and the Online Distributed
Proofreading Team at http://www.pgdp.net (This file was
produced from images generously made available by The
Internet Archive)






                              OFF SANDY HOOK
```

Further, punctuation marks are removed using ***maketrans()*** function and ***translate()*** function. First, *maketrans()* maps the string.punctuation to None and creates a mapping table. Then, using *translate()*, all the punctuations in T are replaced by None according to the mapping table.

[12]:
```
translator = str.maketrans("","",string.punctuation)
T=T.translate(translator)
print(T[:1000])
```

```
                                      1

                                           15

                  A                35

                                  44

                                60

                  S               68

                         81

                             95

                  A S          107

                                      119

                                133

                  A              143

                  A                154

                                     164

                                            180

                                             193

                                           205

                  A              219
```

```
                    S               230

                              241

                    S          264

                    S          276

                                         287
```

Removal of the acknowledgement section is done by using the substitution function and regular expressions.

[11]:
```
# Removing acknowledgement
T = re.sub("Project[\s\S]*CONTENTS","",T)
print(T[:1000])
```

```
                                       1

                                            15

                   A                36

                                   44

                                 60

                   S               68

                          81

                              95

                   A S          107

                                       119

                                 133

                   A              143

                   A                154
```

```
                                     164

                                            180

                                             193

                                           205

                  A              219

                   S               230

                             241

                   S          264

                   S          276

                                        287
```

Similarly as above, we remove the chapter names from the index of the book.

[13]:
```python
# Removing Chapter Names
T=re.sub("[A-Z]{2,}","",T)
print(T[:1000])
```

```
                                      1

                                           15

                  A                35

                                  44

                                32

                  S               68

                         81

                             95

                  A S          107
```

```
                                      119

                                133

                  A              143

                  A                154

                                     164

                                            180

                                             193

                                           205

                  A              219

                   S               230

                             241

                   S          264

                   S          276

                                        287
```

Now, we remove the Chapter Numbers using the substitution function and regular expression.

[14]:
```python
# Removing Chapter Numbers
T=re.sub("[0-9]+","",T)
print(T[:1000])
```

```
                  A
```

```
                 S


                 A S


                 A
                 A




                 A
                  S

                  S
                  S

                 A
```

Conversion of the text to lower case for the ease of analysis.

[15]:
```python
# Converting text to lowercase
T = T.lower()
print(T[:1000])
```

a

s

a s

a

a

a

s

s

s

```
                     a
```

[16]:
```
# splitting the lines and joining them into one.
T = " ".join(T.split())
wordcloudT=T
print(T[:1000])
```

```
 a   s  a s a a a s s s a   on board the rampatina liner eleven
days and a half out from liverpool the usual terrific sensation created by the
appearance of the pilotyacht prevailed necks were craned and toes were trodden
on as the steamer slackened speed and a line dexterously thrown by a
bluejerseyed deckhand was caught by somebody aboard the yacht the pilot not
insensible to the fact of his being a personage of note carefully divested his
bearded countenance of all expression as he saluted the captain and taking from
the deckstewards obsequiously proffered salver a glass containing fourfingers
of neat bourbon whisky concealed its contents about his person without
perceptible emotion and went up with the first officer upon the upper bridge as
the relieved skipper plunged below the telegraphs clicked their messagethe
leviathan hulk of the liner quivered and began to forge slowly ahead and an
intelligentlooking thinlipped badlyshaved young man in a bowler tweeds and str
```

Tokenisation is splitting a string into words, thus, forming a list of words known as tokens. For tokenisation, we use a library from *nltk* known as *tokenise*. The library consists of function *word_tokenize()* which takes the string, here *T* as the parameter and returns the list as shown in the output.

[17]:
```
# Tokenisation
from nltk.tokenize import word_tokenize

T = word_tokenize(T)
print(T[:1000])
```

```
['\ufeff', 'a', '', '', '', '', 's', '', 'a', '', 's', 'a', 'a', 'a', '',
's', '', 's', '', 's', 'a', '', '', '', '', 'on', 'board', 'the',
'rampatina', 'liner', 'eleven', 'days', 'and', 'a', 'half', 'out', 'from',
'liverpool', 'the', 'usual', 'terrific', 'sensation', 'created', 'by', 'the',
'appearance', 'of', 'the', 'pilotyacht', 'prevailed', 'necks', 'were', 'craned',
'and', 'toes', 'were', 'trodden', 'on', 'as', 'the', 'steamer', 'slackened',
'speed', 'and', 'a', 'line', 'dexterously', 'thrown', 'by', 'a', 'bluejerseyed',
'deckhand', 'was', 'caught', 'by', 'somebody', 'aboard', 'the', 'yacht', 'the',
'pilot', 'not', 'insensible', 'to', 'the', 'fact', 'of', 'his', 'being', 'a',
'personage', 'of', 'note', 'carefully', 'divested', 'his', 'bearded',
'countenance', 'of', 'all', 'expression', 'as', 'he', 'saluted', 'the',
'captain', 'and', 'taking', 'from', 'the', 'decksteward', '', 's',
'obsequiously', 'proffered', 'salver', 'a', 'glass', 'containing',
'fourfingers', 'of', 'neat', 'bourbon', 'whisky', 'concealed', 'its',
'contents', 'about', 'his', 'person', 'without', 'perceptible', 'emotion',
'and', 'went', 'up', 'with', 'the', 'first', 'officer', 'upon', 'the', 'upper',
'bridge', 'as', 'the', 'relieved', 'skipper', 'plunged', 'below', 'the',
'telegraphs', 'clicked', 'their', 'messagethe', 'leviathan', 'hulk', 'of',
'the', 'liner', 'quivered', 'and', 'began', 'to', 'forge', 'slowly', 'ahead',
'and', 'an', 'intelligentlooking', 'thinlipped', 'badlyshaved', 'young', 'man',
'in', 'a', 'bowler', 'tweeds', 'and', 'striped', 'necktie', 'introduced',
'himself', 'to', 'the', 'second', 'officer', 'as', 'an', 'emissary', 'of',
'the', 'press', '', 'mr', 'cyrus', 'k', 'pillson', 'new', 'york', 'yeller',
'pleased', 'to', 'know', 'you', 'sir', '', 'said', 'the', 'second', 'officer',
'', 'step', 'into', 'the', 'smokeroom', 'this', 'way', 'barsteward', 'a',
'brandy', 'cocktail', 'for', 'me', 'and', 'you', 'sir', 'order', 'whatever',
'you', 'are', 'most', 'in', 'the', 'habit', 'of', 'hoisting', 'whisky',
'straight', 'now', 'sir', 'happy', 'to', 'afford', 'you', 'what', 'information',
'i', 'can', '', '', 'i', 'presume', '', 'observed', 'the', 'young',
'gentleman', 'of', 'the', 'press', 'settling', 'himself', 'on', 'the',
'springy', 'morocco', 'cushions', 'and', 'accepting', 'the', 'second',
'officer', '', 's', 'polite', 'offer', 'of', 'a', 'green', 'havana', 'of',
'the', 'strongest', 'kind', '', 'that', 'you', 'have', 'had', 'a', 'smooth',
'passage', 'considerin', '', 'the', 'time', 'of', 'year', '', '', 'smooth',
'', 'the', 'second', 'officer', 'carefully', 'reversed', 'in', 'his', 'reply',
'the', 'pressman', '', 's', 'remark', '', 'well', 'yes', 'the', 'time', 'of',
'year', 'considered', 'a', 'smooth', 'passage', 'i', 'take', 'it', 'we', 'have',
'had', '', '', 'no', 'fogs', '', 'interrogated', 'the', 'young', 'gentleman',
'clicking', 'the', 'elastic', 'band', 'of', 'a', 'notebook', 'which',
'projected', 'from', 'his', 'breastpocket', '', 'fogs', 'no', '', 'said',
'the', 'second', 'officer', '', 'you', 'didn', '', 't', 'chance', '',
'pursued', 'the', 'young', 'gentleman', 'of', 'the', 'press', 'taking', 'his',
'short', 'drink', 'from', 'the', 'steward', '', 's', 'salver', 'and',
'throwing', 'it', 'contemptuously', 'down', 'his', 'throat', '', 'to', 'fall',
'in', 'with', 'a', 'berg', 'off', 'the', 'bank', 'did', 'you', '', '', 'not',
'a', 'smell', 'of', 'one', '', 'replied', 'the', 'second', 'officer', 'with',
'decision', '', 'ran', 'into', 'a', 'derelict', 'hencoop', 'perhaps', '',
'persisted', 'the', 'young', 'gentleman', 'concealing', 'the', 'worn', 'sole',
'of', 'a', 'wearied', 'boot', 'from', 'the', 'searching', 'glare', 'of', 'the',
'electric', 'light', 'by', 'tucking', 'it', 'underneath', 'him', '', 'or',
'an', 'old', 'lady', '', 's', 'bonnetbox', 'or', 'a', 'rubber', 'doll', 'some',
'woman', '', 's', 'baby', 'had', 'lost', 'overboard', 'no', '', 'he',
'echoed', 'as', 'the', 'second', 'officer', 'shook', 'his', 'head', '', 'then',
'how', 'in', 'thunder', 'did', 'you', 'manage', 'to', 'lose', 'twenty', 'feet',
'of', 'your', 'portrail', '', '', 'carried', 'away', '', 'said', 'the',
'second', 'officer', 'offering', 'the', 'young', 'press', 'gentleman', 'a',
'light', '', 'no', 'thanks', 'always', 'eat', 'mine', '', 'said', 'the',
'young', 'press', 'gentleman', 'gracefully', '', 'matter', 'of', 'taste', '',
'observed', 'the', 'second', 'officer', 'blowing', 'blue', 'rings', '', 'i',
'guess', 'so', 'and', 'i', '', 've', 'a', 'taste', 'for', 'knowing', 'how',
'you', 'came', '', 'said', 'the', 'young', 'pressman', '', 'to', 'part',
'with', 'that', 'twenty', 'foot', 'of', 'rail', '', '', 'carried', 'away',
'', 'said', 'the', 'second', 'officer', '', 'i', 'kin', 'see', 'that', '',
'retorted', 'the', 'visitor', '', 'it', 'was', 'carried', 'away', '', 'said',
'the', 'second', 'officer', '', 'by', 'an', 'elephant', '', '', 'a', 'pet',
'you', 'had', 'running', 'about', 'aboard', '', 'queried', 'the', 'pressman',
'with', 'imperturbable', 'coolness', '', 'a', 'passenger', '', 'returned',
'the', 'second', 'officer', 'with', 'equal', 'calm', 'there', 'was', 'a',
'snap', 'and', 'the', 'pressman', '', 's', 'notebook', 'was', 'open', 'on',
'his', 'knee', 'the', 'pencil', 'vibrated', 'over', 'the', 'virgin', 'page',
'when', 'a', 'curious', 'utterance', 'between', 'a', 'wail', 'a', 'cough',
'and', 'a', 'roar', 'made', 'the', 'hand', 'that', 'held', 'it', 'start', '',
'yarrrr', 'ohowgh', 'yarr', '', 'the', 'melancholy', 'sound', 'came', 'from',
'without', 'borne', 'on', 'the', 'cool', 'breeze', 'of', 'a', 'late',
'afternoon', 'in', 'march', 'through', 'the', 'open', 'ventilators', '',
'might', 'that', '', 'queried', 'the', 'young', 'gentleman', 'of', 'the',
'press', '', 'be', 'an', 'expression', 'of', 'opinion', 'on', 'the', 'part',
'of', 'the', 'elephant', '', '', 'lord', 'love', 'you', 'no', '', 'said',
'the', 'second', 'officer', '', 'it', '', 's', 'the', 'leopard', '', 'he',
'added', 'after', 'a', 'second', '', 's', 'pause', '', 'or', 'the', 'puma',
'', '', 'do', 'you', 'happen', 'to', 'have', 'a', 'menagerie', 'aboard', '',
'inquired', 'the', 'pressman', 'making', 'a', 'note', 'in', 'shorthand', '',
'no', 'sir', 'the', 'beastselephants', 'leopards', 'and', 'a', 'box', 'of',
'cobrasare', 'invoiced', 'from', 'the', 'london', 'docks', 'to', 'a',
'wealthy', 'amateur', 'in', 'new', 'york', 'state', 'not', 'an', 'iron', 'king',
'or', 'a', 'corn', 'king', 'or', 'a', 'cotton', 'king', 'or', 'a', 'pickle',
'king', 'or', 'a', 'kerosene', 'king', '', 'said', 'the', 'second', 'officer',
'with', 'a', 'steady', 'upper', 'lip', '', 'but', 'a', 'chewinggum', 'king',
'', '', 'if', 'you', 'mean', 'shadland', 'c', 'mcoster', '', 'said', 'the',
'pressman', '', 'my', 'mother', 'is', 'his', 'cousin', 'they', 'used', 'to',
'chew', 'gum', 'together', 'in', 'school', 'recess', 'sir', 'little',
'guessing', 'that', 'shad', 'would', 'one', 'day', 'soar', 'on', 'wings',
'made', 'of', 'that', 'article', 'to', 'the', 'realms', 'of', 'gilded',
'plutocracy', '', '', 'i', 'rather', 'imagine', 'the', 'name', 'you',
'mention', 'to', 'be', 'the', 'right', 'one', '', 'said', 'the', 'second',
'officer', 'cautiously', '', 'but', 'i', 'won', '', 't', 'commit', 'myself',
'the', 'beasts', 'shipped', 'from', 'liverpool', 'are', 'intended', 'as', 'a',
'present', 'for', 'the', 'purchaser', '', 's', 'infant', 'daughter', 'on',
'her', 'fifth', 'birthday', '', '', 'yarrrr', 'ohowgh', 'ohowgh', '',
'again', 'the', 'coughing', 'roar', 'vibrated', 'through', 'the', 'smokeroom',
'then', 'the', 'chorus', 'of', '', 'hail', 'columbia', '', 'rose', 'from',
'the', 'promenade', 'deck', 'where', 'the', 'lady', 'passengers', 'were',
'assembled', 'ready', 'to', 'wave', 'starred', 'and', 'striped', 'silk',
'pockethandkerchiefs', 'and', 'exchange', 'patriotic', 'sentiments', 'at',
'the', 'first', 'glimpse', 'of', 'land', '', 'it', '', 's', 'not', 'what',
'i', 'should', 'call', 'a', 'humly', 'voice', 'that', 'of', 'the', 'leopard',
'', 'observed', 'the', 'pressman', 'controlling', 'a', 'slight', 'shiver', '',
'children', 'have']
```

Lemmatization is grouping together of the different forms of a word having similar meaning.

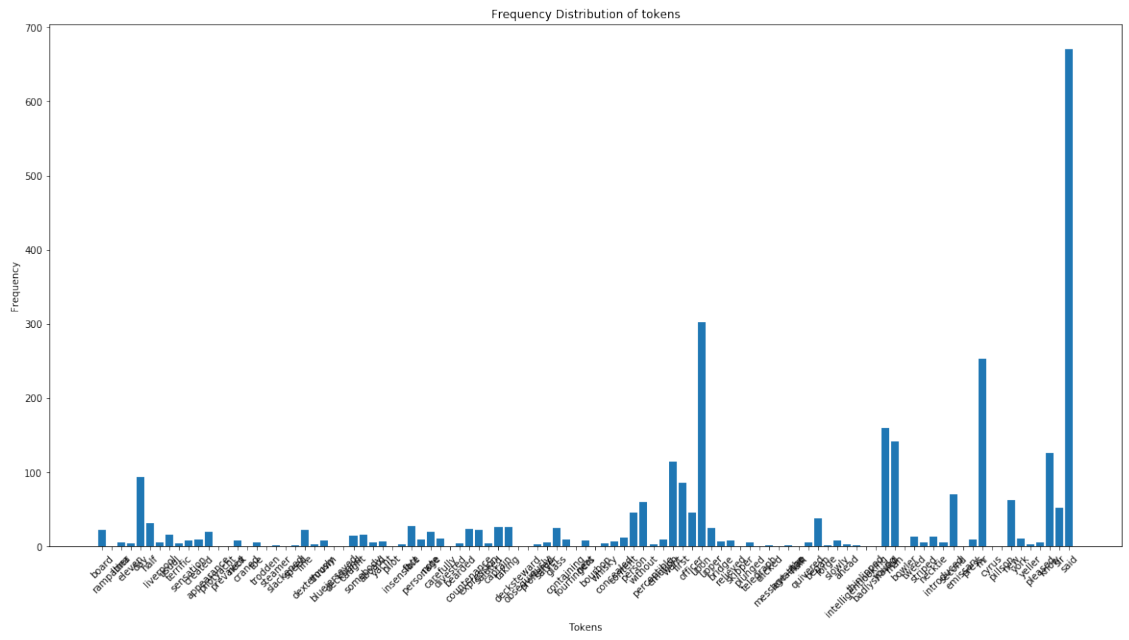
[18]:
```
# Lemmatisation
lemmatizer = WordNetLemmatizer()

# forming a set of stopwords from english language
stop_words = set(stopwords.words('english'))

# creating an empty list
lemmatized_T=[]

# traversing all the words
for word in T:
    # if the word has a length greater than or equal to 2 and is not a stopword
    if len(word) >= 2 and word not in stop_words:
        # then we append the word into the list lemmatized_T after performing␣
 ↪lemmatization
        # using the lemmatize() function
        lemmatized_T.append(lemmatizer.lemmatize(word))

# printing the list of lemmatized words
print(lemmatized_T[:1000])
```

```
['board', 'rampatina', 'liner', 'eleven', 'day', 'half', 'liverpool', 'usual',
'terrific', 'sensation', 'created', 'appearance', 'pilotyacht', 'prevailed',
'neck', 'craned', 'toe', 'trodden', 'steamer', 'slackened', 'speed', 'line',
'dexterously', 'thrown', 'bluejerseyed', 'deckhand', 'caught', 'somebody',
'aboard', 'yacht', 'pilot', 'insensible', 'fact', 'personage', 'note',
'carefully', 'divested', 'bearded', 'countenance', 'expression', 'saluted',
'captain', 'taking', 'decksteward', 'obsequiously', 'proffered', 'salver',
'glass', 'containing', 'fourfingers', 'neat', 'bourbon', 'whisky', 'concealed',
'content', 'person', 'without', 'perceptible', 'emotion', 'went', 'first',
'officer', 'upon', 'upper', 'bridge', 'relieved', 'skipper', 'plunged',
'telegraph', 'clicked', 'messagethe', 'leviathan', 'hulk', 'liner', 'quivered',
'began', 'forge', 'slowly', 'ahead', 'intelligentlooking', 'thinlipped',
'badlyshaved', 'young', 'man', 'bowler', 'tweed', 'striped', 'necktie',
'introduced', 'second', 'officer', 'emissary', 'press', 'mr', 'cyrus',
'pillson', 'new', 'york', 'yeller', 'pleased', 'know', 'sir', 'said', 'second',
'officer', 'step', 'smokeroom', 'way', 'barsteward', 'brandy', 'cocktail',
'sir', 'order', 'whatever', 'habit', 'hoisting', 'whisky', 'straight', 'sir',
'happy', 'afford', 'information', 'presume', 'observed', 'young', 'gentleman',
'press', 'settling', 'springy', 'morocco', 'cushion', 'accepting', 'second',
'officer', 'polite', 'offer', 'green', 'havana', 'strongest', 'kind', 'smooth',
'passage', 'considerin', 'time', 'year', 'smooth', 'second', 'officer',
'carefully', 'reversed', 'reply', 'pressman', 'remark', 'well', 'yes', 'time',
'year', 'considered', 'smooth', 'passage', 'take', 'fog', 'interrogated',
'young', 'gentleman', 'clicking', 'elastic', 'band', 'notebook', 'projected',
'breastpocket', 'fog', 'said', 'second', 'officer', 'chance', 'pursued',
'young', 'gentleman', 'press', 'taking', 'short', 'drink', 'steward', 'salver',
'throwing', 'contemptuously', 'throat', 'fall', 'berg', 'bank', 'smell', 'one',
'replied', 'second', 'officer', 'decision', 'ran', 'derelict', 'hencoop',
'perhaps', 'persisted', 'young', 'gentleman', 'concealing', 'worn', 'sole',
'wearied', 'boot', 'searching', 'glare', 'electric', 'light', 'tucking',
'underneath', 'old', 'lady', 'bonnetbox', 'rubber', 'doll', 'woman', 'baby',
'lost', 'overboard', 'echoed', 'second', 'officer', 'shook', 'head', 'thunder',
'manage', 'lose', 'twenty', 'foot', 'portrail', 'carried', 'away', 'said',
'second', 'officer', 'offering', 'young', 'press', 'gentleman', 'light',
'thanks', 'always', 'eat', 'mine', 'said', 'young', 'press', 'gentleman',
'gracefully', 'matter', 'taste', 'observed', 'second', 'officer', 'blowing',
'blue', 'ring', 'guess', 'taste', 'knowing', 'came', 'said', 'young',
'pressman', 'part', 'twenty', 'foot', 'rail', 'carried', 'away', 'said',
'second', 'officer', 'kin', 'see', 'retorted', 'visitor', 'carried', 'away',
'said', 'second', 'officer', 'elephant', 'pet', 'running', 'aboard', 'queried',
'pressman', 'imperturbable', 'coolness', 'passenger', 'returned', 'second',
'officer', 'equal', 'calm', 'snap', 'pressman', 'notebook', 'open', 'knee',
'pencil', 'vibrated', 'virgin', 'page', 'curious', 'utterance', 'wail', 'cough',
'roar', 'made', 'hand', 'held', 'start', 'yarrrr', 'ohowgh', 'yarr',
'melancholy', 'sound', 'came', 'without', 'borne', 'cool', 'breeze', 'late',
'afternoon', 'march', 'open', 'ventilator', 'might', 'queried', 'young',
'gentleman', 'press', 'expression', 'opinion', 'part', 'elephant', 'lord',
'love', 'said', 'second', 'officer', 'leopard', 'added', 'second', 'pause',
'puma', 'happen', 'menagerie', 'aboard', 'inquired', 'pressman', 'making',
'note', 'shorthand', 'sir', 'beastselephants', 'leopard', 'box', 'cobrasare',
'invoiced', 'london', 'dock', 'wealthy', 'amateur', 'new', 'york', 'state',
'iron', 'king', 'corn', 'king', 'cotton', 'king', 'pickle', 'king', 'kerosene',
'king', 'said', 'second', 'officer', 'steady', 'upper', 'lip', 'chewinggum',
'king', 'mean', 'shadland', 'mcoster', 'said', 'pressman', 'mother', 'cousin',
'used', 'chew', 'gum', 'together', 'school', 'recess', 'sir', 'little',
'guessing', 'shad', 'would', 'one', 'day', 'soar', 'wing', 'made', 'article',
'realm', 'gilded', 'plutocracy', 'rather', 'imagine', 'name', 'mention',
'right', 'one', 'said', 'second', 'officer', 'cautiously', 'commit', 'beast',
'shipped', 'liverpool', 'intended', 'present', 'purchaser', 'infant',
'daughter', 'fifth', 'birthday', 'yarrrr', 'ohowgh', 'ohowgh', 'coughing',
'roar', 'vibrated', 'smokeroom', 'chorus', 'hail', 'columbia', 'rose',
'promenade', 'deck', 'lady', 'passenger', 'assembled', 'ready', 'wave',
'starred', 'striped', 'silk', 'pockethandkerchiefs', 'exchange', 'patriotic',
'sentiment', 'first', 'glimpse', 'land', 'call', 'humly', 'voice', 'leopard',
'observed', 'pressman', 'controlling', 'slight', 'shiver', 'child', 'queer',
'taste', 'said', 'second', 'officer', 'well', 'old', 'spot', 'lively', 'bingo',
'dead', 'bingo', 'queried', 'pressman', 'bingo', 'elephant', 'said', 'second',
'officer', 'passing', 'palm', 'brown', 'right', 'hand', 'upper', 'lip',
'pressman', 'made', 'rapid', 'note', 'particular', 'deathbed', 'scene',
'likely', 'interest', 'youwhy', 'welcome', 'em', 'white', 'said', 'pressman',
'warmly', 'licking', 'pencil', 'elephant', 'die', 'seasickness', 'said',
'second', 'officer', 'calmly', 'seen', 'thing', 'worth', 'seeingmyself',
'said', 'pressman', 'enviously', 'seasick', 'elephant', 'professional',
'ladynurse', 'attendance', 'said', 'second', 'officer', 'complete', 'stem',
'stern', 'print', 'gown', 'white', 'apron', 'flyaway', 'caprigging', 'ward',
'shoe', 'pressman', 'grunted', 'lack', 'interest', 'doubled', 'corner',
'smokeroom', 'divan', 'notebook', 'balanced', 'bulging', 'shirtfront', 'made',
'furious', 'note', 'second', 'officer', 'waited', 'pencil', 'seemed', 'hungry',
'fed', 'little', 'information', 'girl', 'came', 'aboard', 'liverpool',
'mackintosh', 'holdall', 'little', 'black', 'shiny', 'bag', 'went', 'noticed',
'passing', 'sort', 'way', 'freshcolored', 'tidylooking', 'young', 'woman',
'rather', 'plump', 'bow', 'air', 'though', 'meant', 'get', 'full', 'money',
'worth', 'elevenpound', 'fare', 'cheap', 'tariff', 'filled', 'passengerlists',
'fairly', 'full', 'long', 'score', 'thing', 'attend', 'special', 'derrick',
'rigged', 'sling', 'elephant', 'cage', 'aboard', 'capital', 'one', 'sound',
'indian', 'teak', 'strengthened', 'steelmust', 'cost', 'mint', 'money',
'stowed', 'lot', 'sweat', 'swearing', 'promenade', 'deck', 'abaft', 'funnel',
'bolting', 'ring', 'specially', 'screwed', 'deck', 'passing', 'wire', 'hawser',
'across', 'top', 'made', 'fast', 'port', 'starboard', 'davit', 'rigging',
'weatherscreens', 'double', 'tarpaulin', 'keep', 'bingo', 'warm', 'dry',
'beast', 'shipped', 'lee', 'forward', 'cabin', 'skylight', 'got', 'job',
'quiet', 'ladylike', 'voice', 'elbow', 'say', 'please', 'officer', 'regard',
'patient', 'wish', 'know', 'ask', 'purser', 'said', 'rather', 'snappishly',
'hot', 'worried', 'headstewardess', 'asked', 'say', 'voice', 'calm',
'determined', 'way', 'referred', 'well', 'say', 'mistake', 'say', 'young',
'ladyfor', 'young', 'lady', 'hospital', 'nurse', 'besides', 'neatly', 'rigged',
'usual', 'uniform', 'mistake', 'allotted', 'bedroom', 'groundfloor', 'far',
'patient', 'possibly', 'hear', 'call', 'night', 'went', 'breeze', 'played',
'white', 'silk', 'bonnetstrings', 'wavy', 'little', 'kink', 'soft', 'brown',
'hair', 'framed', 'forehead', 'want', 'move', 'upper', 'floor', 'mean',
'promenade', 'deck', 'madam', 'say', 'smoothing', 'grin', 'though', 'well',
'enough', 'used', 'odd', 'bungle', 'landfolks', 'make', 'name', 'thing', 'sea',
'flying', 'pencil', 'stopped', 'pressman', 'looked', 'turning', 'shortened',
'cigar', 'teeth', 'come', 'elephant', 'asked', 'said', 'second', 'officer',
'mean', 'promenade', 'deck', 'say', 'patient', 'occupy', 'one', 'cabin', 'port',
'starboard', 'side', 'may', 'ask', 'number', 'name', 'smiled', 'brightly',
'eye', 'teeth', 'making', 'sort', 'flash', 'together', 'cabin', 'say', 'sleep',
'cage', 'patient', 'bingo', 'elephant', 'great', 'pierpont', 'morgan',
'ejaculated', 'pressman', 'previously', 'flying', 'pencil', 'became', 'almost',
'invisible', 'extreme', 'rapidity', 'plied', 'drop', 'perspiration', 'broke',
'upon', 'sallow', 'forehead', 'glory', 'cried', 'another', 'man', 'thought',
'worth', 'run', 'tackle', 'wallowing', 'old', 'tub', 'touched', 'cap', 'went',
'second', 'officer', 'keeping', 'professionally', 'could', 'surprise', 'felt',
'understand', 'madam', 'asked', 'elephant', 'nurse', 'nodded', 'another',
'bright', 'smile', 'told', 'nurse', 'amy', 'st', 'baalam', 'nursing',
'association', 'london', 'specially', 'engaged', 'american', 'gentleman',
'bought', 'elephant', 'shadland', 'mcoster', 'prompted', 'pressman',
'without', 'looking', 'attend', 'animal', 'voyage', 'understood', 'principal',
'patient', 'condition', 'permitted', 'nurse', 'amy', 'pay', 'leopard',
'attention', 'capable', 'appreciating', 'pressure', 'point', 'ohowgh',
'coughed', 'voice', 'outside', 'yarr', 'ohowgh', 'smell', 'land', 'guess',
'said', 'pressman', 'nigger', 'suggested', 'second', 'officer', 'ought',
'heard', 'bingo', 'three', 'day', 'mersey', 'fair', 'wind', 'smooth', 'sea',
'first', 'nothing', 'delighted', 'lady', 'child', 'board', 'like', 'feeding',
'apple', 'nut', 'biscuit', 'thing', 'prigged', 'saloon', 'table', 'seaair',
'must', 'sharpened', 'beast', 'appetite', 'suppose', 'old', 'trunk', 'snorking',
'round', 'day', 'purser', 'naturally', 'wild', 'said', 'must', 'put', 'away',
'hogshead', 'good', 'thing', 'addition', 'allowance', 'hay', 'bread',
'beetroot', 'grain', 'cabbage', 'sugar', 'ca', 'temper', 'asked', 'pressman',
'mild', 'milk', 'kind', 'beast', 'ever', 'breathed', 'elephant', 'lot',
'breathing', 'said', 'second', 'officer', 'lady', 'gentleman', 'upperdeck',
'cabin', 'used', 'complain', 'snoring', 'night', 'nurse', 'amy', 'said',
'people', 'complain', 'anything', 'em', 'like', 'smell', 'elephantwhich',
'allow', 'happened']
```

Analysis of the frequency of various tokens

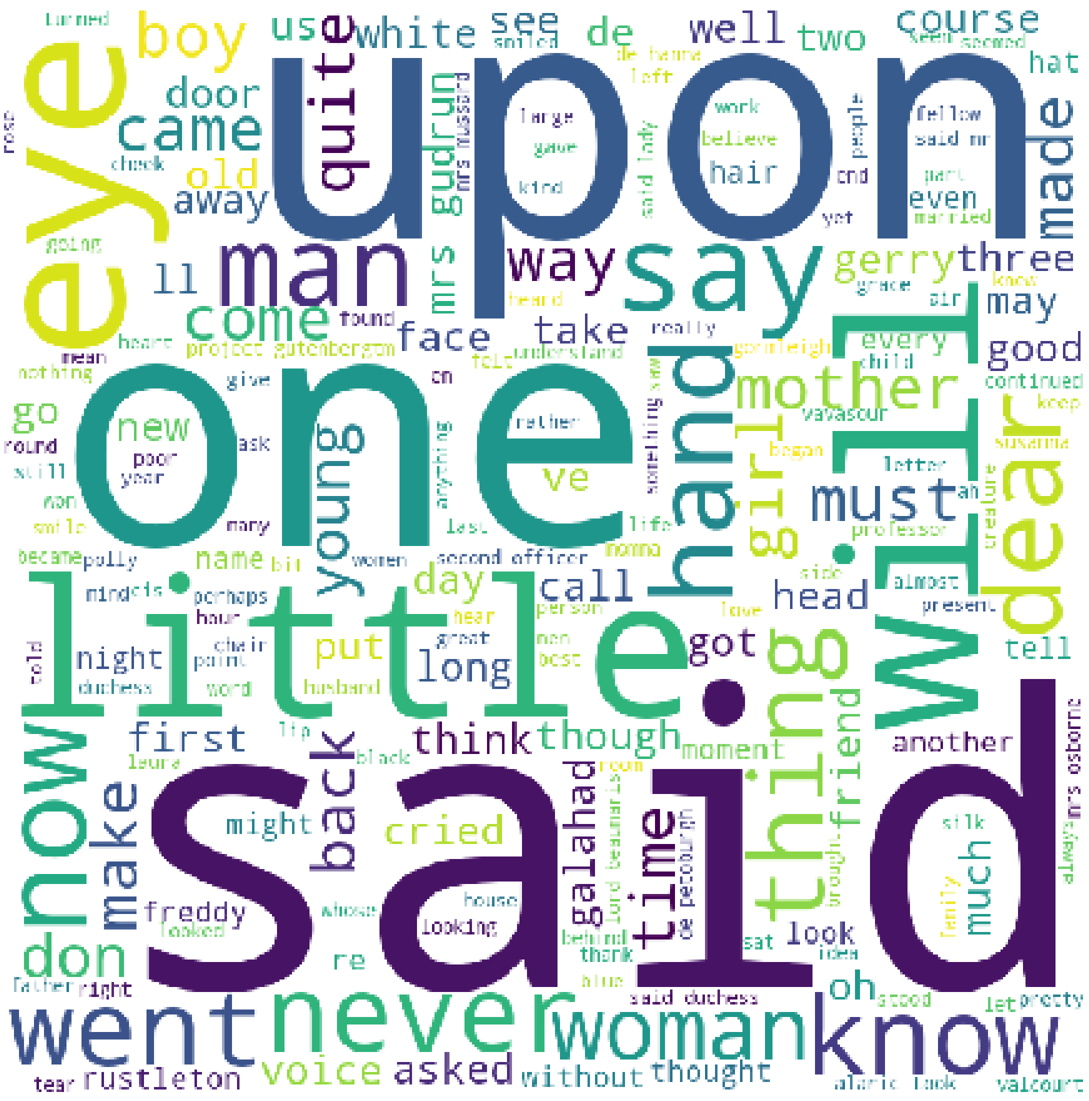
There were a large number of tokens so we just plotted the first 100 tokens

[66]:
```python
# Evaluating frequency distribution of tokens
# The nltk library function FreqDist() returns a dictionary containing key␣
 ↪value pairs where values are the frequency of
# the keys. Here keys are the Tokens.
freq_dist = nltk.FreqDist(lemmatized_T)

# creating a list of the tokens
keys = []
for key in freq_dist.keys():
    if len(keys)<=100:
        keys.append(key)
    else:
        break

# creating a list of the frequency of the various tokens
values = []
for value in freq_dist.values():
    if len(values)<=100:
        values.append(value)
    else:
        break

# plotting a bar plot diagram of the frequency distribution
plt.figure(figsize=(20,10))
plt.bar(keys,values)
plt.xlabel("Tokens")
plt.xticks(rotation=45)
plt.ylabel("Frequency")
plt.title("Frequency Distribution of tokens")
plt.show()
```



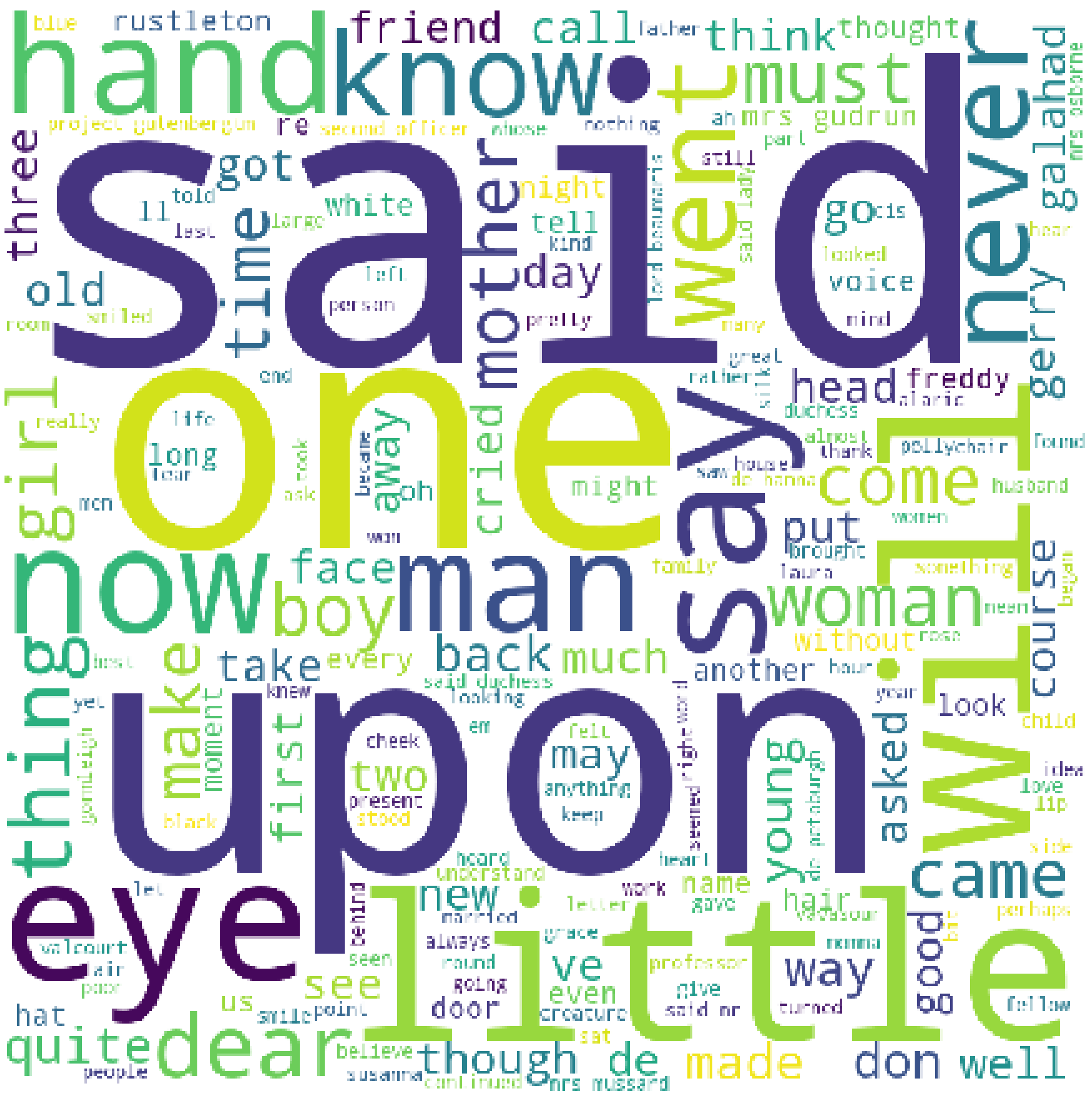
WordCloud is a representation of the textual data according to their frequency. It basically emphasizes the keywords used in the text.

[25]:
```
# WordCloud without removing Stopwords
# We used WORDCLOUD package to create a wordcloud. The function WordCloud takes␣
↪in the size and color of the word cloud
# and then returns a worldcloud object based on the frequency of words.

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

plt.figure(figsize=(8,8),facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")

plt.show()
```



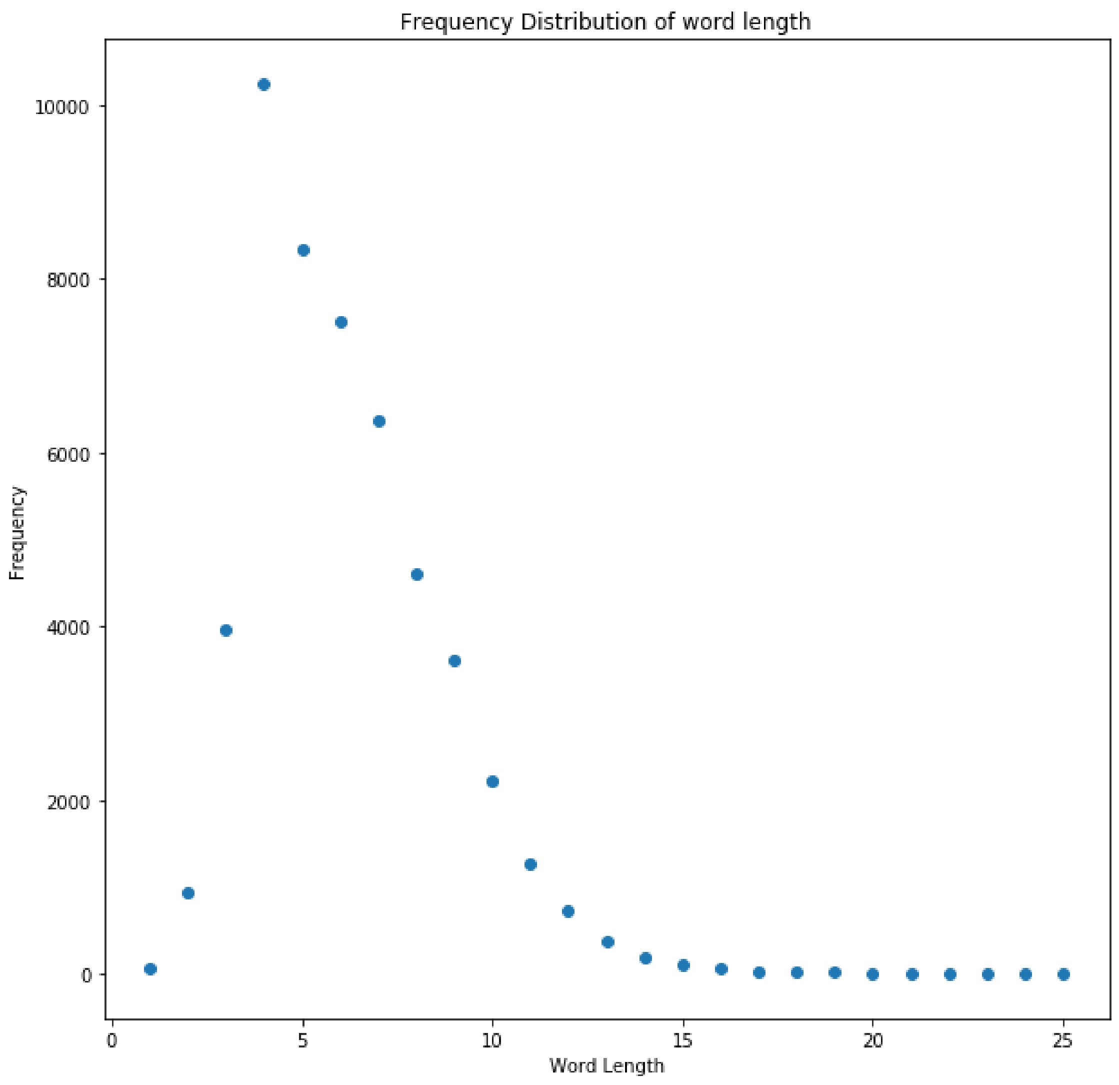
Removing stopwords decreases noise in the data and so other words get more emphasis in the wordcloud

[26]:
```python
# WordCloud after removing Stopwords
wordcloud = WordCloud(width = 800, height=800,
                    background_color='white',
                     stopwords = set(STOPWORDS),
                    min_font_size=10).generate(wordcloudT)

plt.figure(figsize=(8,8),facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")

plt.show()
```



[53]:
```
#Frequency distribution of Word Length
#Result: As the word length increases the frequency of that word decreases. So␣
 ↪large length word occurs less in the book as
#         compared to small length words.
len_list={}

for i in range(len(lemmatized_T)):

    if len(lemmatized_T[i]) not in len_list:
        len_list[len(lemmatized_T[i])] = 1

    else:
        len_list[len(lemmatized_T[i])] += 1
```

```
keys = list(len_list.keys())
values = list(len_list.values())

plt.figure(figsize=(10,10))
plt.scatter(keys,values)
plt.xlabel("Word Length")
plt.ylabel("Frequency")
plt.title("Frequency Distribution of word length")
plt.show()
```



[32]:
```
# using brown corpus for POS tagging. We used nltk again for this. The tagset␣
↪we used is universal so the number of tags
```

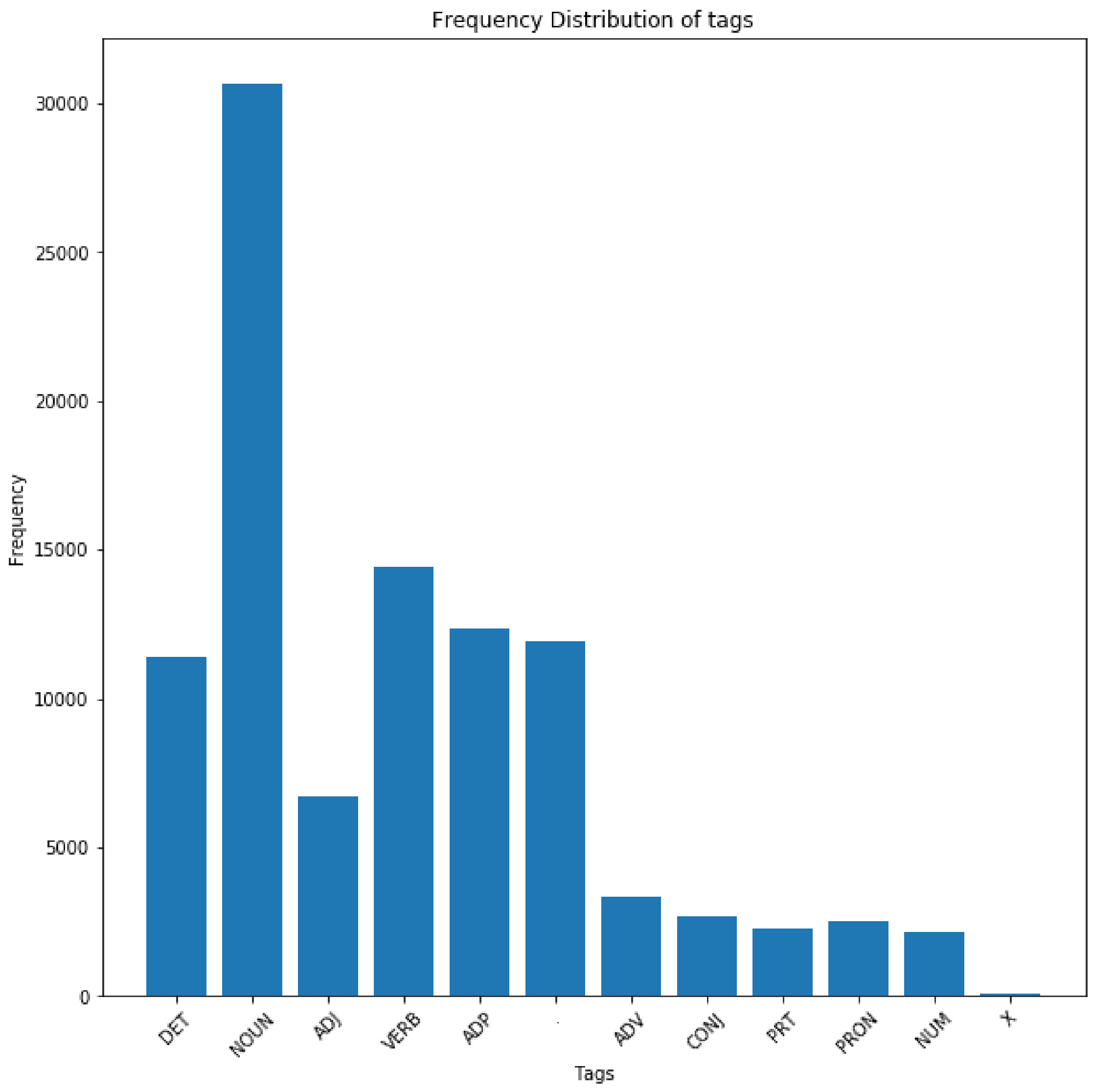
```
# will be less because in universal tagset, many tags are clustered into one␣
 ↪tag and then the pos tagging is performed.
brown_news_words = brown.tagged_words(categories='news', tagset='universal')
brown_news_words
```

[32]:
```
[('The', 'DET'), ('Fulton', 'NOUN'), ...]
```

[35]:
```
# Frequency distribution of POS tags
fdistw = nltk.FreqDist([t for (w, t) in brown_news_words])
fdistw
```

[35]:
```
FreqDist({'NOUN': 30654, 'VERB': 14399, 'ADP': 12355, '.': 11928, 'DET': 11389,
'ADJ': 6706, 'ADV': 3349, 'CONJ': 2717, 'PRON': 2535, 'PRT': 2264, ...})
```

[48]:
```
# Representation of frequrncy distribution of POS tags using brown corpus
# creating a list of the tokens
keys = list(fdistw.keys())

# creating a list of the frequency of the various tokens
values = list(fdistw.values())

# plotting a bar plot diagram of the frequency distribution
plt.figure(figsize=(10,10))
plt.bar(keys,values)
plt.xlabel("Tags")
plt.xticks(rotation=45)
plt.ylabel("Frequency")
plt.title("Frequency Distribution of tags")
plt.show()
```


Frequency Distribution of tags
Frequency
Tags
30000
25000
20000
15000
10000
5000
0
DET
NOUN
ADJ
VERB
ADP
.
ADV
CONJ
PRT
PRON
NUM
X

Round2

December 15, 2019

1 Book Details

Book 1

Book Title - Off Sandy Hook, and other stories Author's Name - Richard Dehan Language: English Subjects - Fiction, Short stories Number of Sentences - 11202 Number of Words - 98822 Character set encoding - UTF-8

Book 2

Book Title - A Book Author's Name - Djuna Barnes Language: English Subjects - Fiction Number of Sentences - 1967 Number of Words - 43897 Character set encoding - UTF-8

This is part of NLP-Projects Round-2. In this project, we have used the following libraries and toolkits: * **nltk** - A leading platform in Python designed to provide easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries. These libraries are used for classification, tokenisation, lemmatisation etc. * **string** - This module consists of basic operations that can be performed on string. * **WordNetLemmatizer** - It is a package for lemmatization. Lemmatization is the process of grouping together the words having similar meaning to a particular word. * **stopwords** - Set of commonly used words that are mostly ignored in text processing. * **matplotlib** - Library used for data visualisation. In other words, it is used for plotting the relationship among different components. * **re** - Library used for operations on regular expressions. * **spaCy** - It is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It is designed specifically for production use and helps in building applications that process large volumes of text. * **en_core_web_sm** - This Library is part of spacy which allows your application to handle a model like any other package dependency. * **seqeval** - A Python framework for sequence labeling evaluation. It is used to evaluate the performance of chunking tasks such as named-entity recognition, part-of-speech tagging, semantic role labeling etc.

To begin, we import all the necssary libraries.

[1]:
```
import nltk
import string
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.corpus import brown
import matplotlib.pyplot as plt
import numpy as np
import re
```

2 Preprocessing

Book 1

Now, open the book and read it in a variable *file*. Further, we read the contents of the book in another variable *T*. We declare a variable *T2* equal to *T* in case we require the original text later in the process as *T* will undergo text processing.

[2]:
```python
#Reading Book
file = open("Sandy_Hook_NLP.txt",encoding="utf8")

# variable T
T = file.read()

# variable T2
T2=T

# Printing the contents of the book stored in T
print(T[:1000])
```

```
Project Gutenberg's Off Sandy Hook and other stories, by Richard Dehan

This eBook is for the use of anyone anywhere in the United States and most
other parts of the world at no cost and with almost no restrictions
whatsoever.  You may copy it, give it away or re-use it under the terms of
the Project Gutenberg License included with this eBook or online at
www.gutenberg.org.  If you are not located in the United States, you'll have
to check the laws of the country where you are located before using this ebook.

Title: Off Sandy Hook and other stories

Author: Richard Dehan

Release Date: October 8, 2019 [EBook #60452]

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK OFF SANDY HOOK AND OTHER STORIES ***




Produced by Richard Tonsing and the Online Distributed
Proofreading Team at http://www.pgdp.net (This file was
produced from images generously made available by The
Internet Archive)
```

```
                    OFF SANDY HOOK
```

Removal of the acknowledgement section and transcriber's note is done by using the substitution function and regular expressions.

[3]:
```python
# Removing acknowledgement
T = re.sub("Project[\s\S]*CONTENTS","",T)
T = re.sub("TRANSCRIBER[\s\S]*","",T)
print(T[:1000])
```

```
                                 PAGE
          OFF SANDY HOOK            1

          GEMINI                   15

          A DISH OF MACARONI       31

          FREDDY & C^{IE}        44

          UNDER THE ELECTRICS      60

          VALCOURTS GRIN         68

          THE EVOLUTION OF THE FAIREST   81

          THE REVOLT OF RUSTLETON        95

          A DYSPEPTICS TRAGEDY         107

          RENOVATION                  119

          THE BREAKING PLACE          133

          A LANCASHIRE DAISY          143

          A PITCHED BATTLE            154
```

```
THE TUG OF WAR                164

GAS!                          180

AIR                           193

SIDE!                         205

A SPIR
```

Similarly as above, we remove the chapter names from the index of the book.

[4]:
```
# Removing Chapter Names
T=re.sub("[A-Z]{2,}","",T)
print(T[:1000])
```

```




                    1

                         15

A                 31

 & C^{}                 44

              60

S                 68

       81

           95

A S           107

                    119

              133

A              143

A                154

                   164
```

```
!                                180

                                 193

!                              205

A                   219

 S                    230

               241

 S             264

 S             276

                         28
```

Now, we remove the Chapter Numbers using the substitution function and regular expression.

[5]:
```python
# Removing Chapter Numbers
T=re.sub("[0-9]+","",T)
print(T[:1000])
```

```
A

 & C^{}



S






A S
```

```
A

A


!


!

A

 S


 S

 S


A
```

Splitting the lines and joining them into one.

[6]:
```
# splitting the lines and joining them into one.
T = " ".join(T.split())
T=T.replace(T[:51],'')
T = T.replace('', '')
T = T.replace('', '')
T = T.replace("s", '')
T = T.replace("d", '')
T = T.replace("", '')
T2=T
print(T[:1000])
```

```
 On board the Rampatina liner, eleven days and a half out from Liverpool, the
usual terrific sensation created by the appearance of the pilot-yacht prevailed.
Necks were craned and toes were trodden on as the steamer slackened speed, and a
line dexterously thrown by a blue-jerseyed deck-hand was caught by somebody
aboard the yacht. The pilot, not insensible to the fact of his being a personage
of note, carefully divested his bearded countenance of all expression as he
saluted the Captain, and taking from the deck-steward obsequiously proffered
```

```
salver a glass containing four-fingers of neat Bourbon whisky, concealed its
contents about his person without perceptible emotion, and went up with the
First Officer upon the upper bridge as the relieved skipper plunged below. The
telegraphs clicked their messagethe leviathan hulk of the liner quivered and
began to forge slowly ahead, and an intelligent-looking, thin-lipped, badly-
shaved young man in a bowler, tweeds, and striped necktie, intro
```

Tokenisation is splitting a string into words, thus, forming a list of words known as tokens. For tokenisation, we use a library from *nltk* known as *tokenise*. The library consists of function *word_tokenize()* which takes the string, here *T* as the parameter and returns the list as shown in the output. We did this because it helps in easily interpreting the text by analyzing the tokens present in the it.

[7]:
```
# Tokenisation
from nltk.tokenize import word_tokenize

T = word_tokenize(T)
print(T[:100])
```

```
['On', 'board', 'the', 'Rampatina', 'liner', ',', 'eleven', 'days', 'and', 'a',
'half', 'out', 'from', 'Liverpool', ',', 'the', 'usual', 'terrific',
'sensation', 'created', 'by', 'the', 'appearance', 'of', 'the', 'pilot-yacht',
'prevailed', '.', 'Necks', 'were', 'craned', 'and', 'toes', 'were', 'trodden',
'on', 'as', 'the', 'steamer', 'slackened', 'speed', ',', 'and', 'a', 'line',
'dexterously', 'thrown', 'by', 'a', 'blue-jerseyed', 'deck-hand', 'was',
'caught', 'by', 'somebody', 'aboard', 'the', 'yacht', '.', 'The', 'pilot', ',',
'not', 'insensible', 'to', 'the', 'fact', 'of', 'his', 'being', 'a',
'personage', 'of', 'note', ',', 'carefully', 'divested', 'his', 'bearded',
'countenance', 'of', 'all', 'expression', 'as', 'he', 'saluted', 'the',
'Captain', ',', 'and', 'taking', 'from', 'the', 'deck-steward', 'obsequiously',
'proffered', 'salver', 'a', 'glass', 'containing']
```

Lemmatization is grouping together of the different forms of a word having similar meaning that is reducing the inflected forms and sometimes derivationally related forms of a word to a common base form to ease the analysis of the document. It is better than stemming because it dosen't reduce the word in a crude way.

[8]:
```
# Lemmatisation
lemmatizer = WordNetLemmatizer()

# forming a set of stopwords from english language
stop_words = set(stopwords.words('english'))

# creating an empty list
lemmatized_T=[]

# traversing all the words
for word in T:
    # if the word has a length greater than or equal to 2 and is not a stopword
    if len(word) >= 2 and word not in stop_words:
        # then we append the word into the list lemmatized_T after performing␣
↪lemmatization
        # using the lemmatize() function
        lemmatized_T.append(lemmatizer.lemmatize(word))

# printing the list of lemmatized words
print(lemmatized_T[:100])
```

```
['On', 'board', 'Rampatina', 'liner', 'eleven', 'day', 'half', 'Liverpool',
'usual', 'terrific', 'sensation', 'created', 'appearance', 'pilot-yacht',
'prevailed', 'Necks', 'craned', 'toe', 'trodden', 'steamer', 'slackened',
'speed', 'line', 'dexterously', 'thrown', 'blue-jerseyed', 'deck-hand',
'caught', 'somebody', 'aboard', 'yacht', 'The', 'pilot', 'insensible', 'fact',
'personage', 'note', 'carefully', 'divested', 'bearded', 'countenance',
'expression', 'saluted', 'Captain', 'taking', 'deck-steward', 'obsequiously',
'proffered', 'salver', 'glass', 'containing', 'four-fingers', 'neat', 'Bourbon',
'whisky', 'concealed', 'content', 'person', 'without', 'perceptible', 'emotion',
'went', 'First', 'Officer', 'upon', 'upper', 'bridge', 'relieved', 'skipper',
'plunged', 'The', 'telegraph', 'clicked', 'messagethe', 'leviathan', 'hulk',
'liner', 'quivered', 'began', 'forge', 'slowly', 'ahead', 'intelligent-looking',
'thin-lipped', 'badly-shaved', 'young', 'man', 'bowler', 'tweed', 'striped',
'necktie', 'introduced', 'Second', 'Officer', 'emissary', 'Press', 'Mr.',
'Cyrus', 'K.', 'Pillson']
```

From Graph it is clearly visible that contact category occures the most in verbs.
Book 2
For Book 2, we use the same steps of preprcessing as for Book 1.

[9]:
```
#Reading Book 2
file = open("book2.txt",encoding="utf8")

# variable S
S = file.read()

# variable S2
S2=S

# Printing the contents of the book stored in S
print(S[:1000])
```

```
The Project Gutenberg EBook of A Book, by Djuna Barnes

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever.  You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
www.gutenberg.org.  If you are not located in the United States, you'll
have to check the laws of the country where you are located before using
```

```
this ebook.



Title: A Book

Author: Djuna Barnes

Release Date: December 11, 2019 [EBook #60904]

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK A BOOK ***




Produced by Richard Tonsing, Tim Lindell, and the Online
Distributed Proofreading Team at http://www.pgdp.net (This
book was produced from images made available by the
HathiTrust Digital Library.)








                                  A BOOK


[Illustration]




                                  A BOO
```

[10]:
```
# Removing acknowledgement
S = re.sub("Project[\s\S]*CONTENTS","",S)
S = re.sub("TRANSCRIBER[\s\S]*","",S)
print(S[:1000])
```

```
The
```

```
                                PAGE

   A NIGHT AMONG THE HORSES        1

   THREE FROM THE EARTH           15

   THE VALET                      31

   TO THE DOGS                    44

   BEYOND THE END                 59

   PASTORAL                       74

   OSCAR                          76

   ANTIQUE                       103

   KATRINA SILVERSTAFF           104

   HUSH BEFORE LOVE              116

   THE ROBINS HOUSE             117

   PARADISE                      131

   NO-MANS-MARE                 132

   SIX SONGS OF KHALIDINE        145

   THE DOVE                      147

   MOTHER                        164

   SONG IN AUTUMN
```

[11]:
```
# Removing Chapter Names
S=re.sub("[A-Z]{2,}","",S)
print(S[:1000])
```

```
The
```

```
                  A               1

                                   15

                                            31

                                           44

                                        59

                                            74

                                               76

                                            103

                                 104

                                     116

                   S                 117

                                           131

                  -S-                    132

                                145

                                            147

                                             164

                                       172

                                          173

                                            179

                                       180

                  ID              194
```

[12]:
```
# Removing Chapter Numbers
S=re.sub("[0-9]+","",S)
print(S[:1000])
```

```
The
```

A

S

-S-

ID

[13]:
```python
# splitting the lines and joining them into one.
S = " ".join(S.split())
S=S.replace(S[:62],'')
S = S.replace('', '')
S = S.replace('', '')
S = S.replace("s", '')
S = S.replace("d", '')
S = S.replace("", '')
S2=S
print(S[:1000])
```

```
rtrait Portrait Drawing Portrait Study Portrait A A Toward dusk, in the Summer
of the year, a man dressed in a frock coat and top hat, and carrying a cane,
crept through the underbrush bordering the corral of the Buckler farm. As he
moved, small twigs snapped, fell and were silent. His knees were green from
wounded shrubbery and grass, and his outspread hands tore unheeded plants. His
wrists hurt him and he rested from time to time, always caring for his hat and
knotted yellow cane, blowing through his moustache. Dew had been falling,
covering the twilight leaves like myriad faces damp with the perspiration of the
struggle for existence, and half a mile away, standing out against the darkness
of the night, a grove of white birches shimmered like teeth in a skull. He heard
the creaking of a gate, and the splashing of late rain into the depths of a dark
cistern. His heart ached with the nearness of the earth, the faint murmur of it
moving upon itself, like a sleeper who turns to throw an
```

[14]:
```python
# Tokenisation
from nltk.tokenize import word_tokenize

S = word_tokenize(S)
print(S[:100])
```

```
['rtrait', 'Portrait', 'Drawing', 'Portrait', 'Study', 'Portrait', 'A', 'A',
'Toward', 'dusk', ',', 'in', 'the', 'Summer', 'of', 'the', 'year', ',', 'a',
'man', 'dressed', 'in', 'a', 'frock', 'coat', 'and', 'top', 'hat', ',', 'and',
'carrying', 'a', 'cane', ',', 'crept', 'through', 'the', 'underbrush',
'bordering', 'the', 'corral', 'of', 'the', 'Buckler', 'farm', '.', 'As', 'he',
'moved', ',', 'small', 'twigs', 'snapped', ',', 'fell', 'and', 'were', 'silent',
'.', 'His', 'knees', 'were', 'green', 'from', 'wounded', 'shrubbery', 'and',
'grass', ',', 'and', 'his', 'outspread', 'hands', 'tore', 'unheeded', 'plants',
'.', 'His', 'wrists', 'hurt', 'him', 'and', 'he', 'rested', 'from', 'time',
'to', 'time', ',', 'always', 'caring', 'for', 'his', 'hat', 'and', 'knotted',
'yellow', 'cane', ',', 'blowing']
```

[15]:
```
# Lemmatisation
lemmatizer2 = WordNetLemmatizer()

# forming a set of stopwords from english language
stop_words2 = set(stopwords.words('english'))

# creating an empty list
lemmatized_S=[]

# traversing all the words
for word in S:
    # if the word has a length greater than or equal to 2 and is not a stopword
    if len(word) >= 2 and word not in stop_words2:
        # then we append the word into the list lemmatized_T after performing␣
↪lemmatization
        # using the lemmatize() function
        lemmatized_S.append(lemmatizer2.lemmatize(word))

# printing the list of lemmatized words
print(lemmatized_S[:100])
```

```
['rtrait', 'Portrait', 'Drawing', 'Portrait', 'Study', 'Portrait', 'Toward',
'dusk', 'Summer', 'year', 'man', 'dressed', 'frock', 'coat', 'top', 'hat',
'carrying', 'cane', 'crept', 'underbrush', 'bordering', 'corral', 'Buckler',
'farm', 'As', 'moved', 'small', 'twig', 'snapped', 'fell', 'silent', 'His',
'knee', 'green', 'wounded', 'shrubbery', 'grass', 'outspread', 'hand', 'tore',
'unheeded', 'plant', 'His', 'wrist', 'hurt', 'rested', 'time', 'time', 'always',
'caring', 'hat', 'knotted', 'yellow', 'cane', 'blowing', 'moustache', 'Dew',
'falling', 'covering', 'twilight', 'leaf', 'like', 'myriad', 'face', 'damp',
'perspiration', 'struggle', 'existence', 'half', 'mile', 'away', 'standing',
'darkness', 'night', 'grove', 'white', 'birch', 'shimmered', 'like', 'teeth',
'skull', 'He', 'heard', 'creaking', 'gate', 'splashing', 'late', 'rain',
'depth', 'dark', 'cistern', 'His', 'heart', 'ached', 'nearness', 'earth',
'faint', 'murmur', 'moving', 'upon']
```

3 PART 1

3.1 Finding nouns, verbs, their categories and their frequecies

Book 1

[16]:
```
# Seprating nouns and verbs from lemmatized words
from nltk.corpus import wordnet as wn
nouns = []
verbs = []
for word in lemmatized_T:
    for synset in wn.synsets(word):
        if "noun" in synset.lexname() and word not in nouns:
            nouns.append(word)

        elif "verb" in synset.lexname() and word not in verbs:
            verbs.append(word)
```

[17]:
```
print(nouns[:100])
```

```
['board', 'liner', 'eleven', 'day', 'half', 'Liverpool', 'sensation',
'appearance', 'Necks', 'toe', 'steamer', 'speed', 'line', 'somebody', 'yacht',
'pilot', 'fact', 'personage', 'note', 'countenance', 'expression', 'Captain',
'taking', 'salver', 'glass', 'Bourbon', 'whisky', 'content', 'person',
'emotion', 'First', 'Officer', 'upper', 'bridge', 'skipper', 'telegraph',
'leviathan', 'hulk', 'forge', 'young', 'man', 'bowler', 'tweed', 'necktie',
'Second', 'emissary', 'Press', 'Mr.', 'Cyrus', 'York', 'know', 'sir', 'step',
'way', 'brandy', 'cocktail', 'order', 'habit', 'Whisky', 'straight', 'Now',
'information', 'gentleman', 'settling', 'morocco', 'cushion', 'offer', 'green',
'Havana', 'kind', 'smooth', 'passage', 'time', 'year', 'Smooth', 'reply',
'Pressman', 'remark', 'Well', 'yes', 'take', 'No', 'fog', 'elastic', 'band',
'notebook', 'Fogs', 'chance', 'pursued', 'short', 'drink', 'steward', 'throat',
'fall', 'berg', 'Bank', 'smell', 'one', 'decision', 'derelict']
```

[18]:
```
print(verbs[:100])
```

```
['board', 'created', 'prevailed', 'Necks', 'craned', 'toe', 'trodden',
'steamer', 'slackened', 'speed', 'line', 'thrown', 'caught', 'yacht', 'pilot',
'note', 'divested', 'bearded', 'countenance', 'saluted', 'Captain', 'taking',
'proffered', 'glass', 'containing', 'concealed', 'content', 'went', 'Officer',
'bridge', 'relieved', 'skipper', 'plunged', 'telegraph', 'clicked', 'hulk',
'quivered', 'began', 'forge', 'man', 'striped', 'introduced', 'Second', 'Press',
'Pleased', 'know', 'said', 'step', 'order', 'habit', 'hoisting', 'afford',
'presume', 'observed', 'settling', 'cushion', 'accepting', 'offer', 'green',
'smooth', 'time', 'Smooth', 'reversed', 'reply', 'remark', 'Well', 'considered',
'take', 'fog', 'interrogated', 'clicking', 'band', 'projected', 'Fogs',
'chance', 'pursued', 'short', 'drink', 'throwing', 'fall', 'Bank', 'smell',
'replied', 'Ran', 'persisted', 'concealing', 'worn', 'sole', 'wearied', 'boot',
'searching', 'glare', 'light', 'tucking', 'rubber', 'baby', 'lost', 'echoed',
'shook', 'head']
```

Creating a dictionary of words and category of nouns or verbs they belong to.

[19]:
```
# Nouns
noun_dic = {}
lt = []
for noun in nouns:
    l = []
    for synset in wn.synsets(noun):
        if "noun" in synset.lexname():
            if synset.lexname()[5:] not in l:
                l.append(synset.lexname()[5:])

    noun_dic[noun] = l
```

[20]:
```
j = 0
for i in noun_dic:
    print(i,":", noun_dic[i])
    j +=1
    if j == 50:
        break
```

```
board : ['group', 'substance', 'artifact', 'food']
liner : ['act', 'artifact']
eleven : ['quantity', 'group']
day : ['time', 'state', 'person']
half : ['quantity', 'time']
Liverpool : ['location']
sensation : ['cognition', 'person', 'feeling', 'state']
appearance : ['attribute', 'event', 'act', 'cognition']
Necks : ['body', 'object', 'food', 'artifact']
toe : ['body', 'artifact']
steamer : ['food', 'artifact', 'animal']
speed : ['time', 'attribute', 'act', 'relation', 'artifact']
line : ['group', 'communication', 'shape', 'phenomenon', 'location',
'cognition', 'artifact', 'act', 'quantity', 'possession']
somebody : ['Tops']
yacht : ['artifact']
pilot : ['person', 'communication', 'cognition', 'artifact']
fact : ['cognition', 'communication', 'state']
personage : ['person']
note : ['communication', 'attribute', 'possession', 'state']
countenance : ['attribute', 'communication', 'body']
expression : ['attribute', 'communication', 'process', 'act']
Captain : ['person']
taking : ['act']
salver : ['artifact']
glass : ['substance', 'artifact', 'quantity']
Bourbon : ['person', 'food', 'group']
whisky : ['food']
content : ['group', 'communication', 'relation', 'quantity', 'cognition',
'state', 'artifact']
person : ['Tops', 'body', 'communication']
emotion : ['feeling']
First : ['relation', 'quantity', 'time', 'act', 'communication', 'artifact']
Officer : ['person']
upper : ['artifact']
bridge : ['artifact', 'relation', 'body', 'act']
skipper : ['person']
```

```
telegraph : ['artifact']
leviathan : ['person']
hulk : ['person', 'artifact']
forge : ['artifact']
young : ['animal', 'person', 'group']
man : ['person', 'animal', 'location', 'artifact']
bowler : ['person', 'artifact']
tweed : ['artifact']
necktie : ['artifact']
Second : ['time', 'act', 'relation', 'quantity', 'person', 'communication',
'artifact']
emissary : ['person']
Press : ['state', 'communication', 'artifact', 'group', 'act']
Mr. : ['communication']
Cyrus : ['person']
York : ['group']
```

[21]:
```python
# Verbs
verb_dic = {}
for verb in verbs:
    l = []
    for synset in wn.synsets(verb):
        if "verb" in synset.lexname():
            if synset.lexname()[5:] not in l:
                l.append(synset.lexname()[5:])
    verb_dic[verb] = l

j = 0
for i in verb_dic:
    print(i,":", verb_dic[i])
    j +=1
    if j == 50:
        break
```

```
board : ['motion', 'stative', 'consumption']
created : ['creation', 'social']
prevailed : ['stative', 'competition', 'communication']
Necks : ['contact']
craned : ['body']
toe : ['motion', 'contact']
trodden : ['motion', 'contact', 'possession']
steamer : ['motion']
slackened : ['change']
speed : ['motion', 'change']
line : ['stative', 'contact', 'change']
thrown : ['contact', 'motion', 'communication', 'creation', 'emotion',
'cognition']
```

```
caught : ['cognition', 'perception', 'contact', 'motion', 'possession',
'competition', 'change', 'social', 'body', 'weather', 'emotion', 'creation']
yacht : ['motion']
pilot : ['motion']
note : ['communication', 'perception']
divested : ['possession', 'social', 'change']
bearded : ['stative']
countenance : ['communication']
saluted : ['consumption', 'communication', 'perception']
Captain : ['social']
taking : ['social', 'possession', 'motion', 'contact', 'change', 'cognition',
'stative', 'perception', 'communication', 'consumption', 'competition', 'body']
proffered : ['possession']
glass : ['possession', 'perception', 'contact', 'change']
containing : ['stative', 'social', 'competition']
concealed : ['perception']
content : ['consumption', 'emotion']
went : ['motion', 'social', 'change', 'stative', 'perception', 'contact',
'consumption', 'competition', 'cognition']
Officer : ['communication']
bridge : ['stative', 'contact', 'motion']
relieved : ['body', 'social', 'emotion', 'communication', 'possession',
'change']
skipper : ['social']
plunged : ['contact', 'motion', 'change', 'cognition']
telegraph : ['communication']
clicked : ['motion', 'perception', 'contact', 'communication', 'cognition']
hulk : ['stative']
quivered : ['motion']
began : ['change', 'stative', 'communication', 'social']
forge : ['creation', 'motion']
man : ['social', 'competition']
striped : ['contact', 'possession', 'body', 'change']
introduced : ['communication', 'creation', 'change', 'motion', 'contact',
'cognition']
Second : ['social']
Press : ['contact', 'communication', 'stative', 'motion', 'creation', 'social',
'competition']
Pleased : ['emotion']
know : ['cognition', 'contact']
said : ['communication', 'stative']
step : ['motion', 'social', 'possession', 'contact', 'change']
order : ['communication', 'social', 'change', 'cognition']
habit : ['body']
```

Creating a dictionary of frequency of category of nouns and verbs.

[22]:
```
# Nouns
noun_cate_dic = {}

for i in noun_dic:
    for j in noun_dic[i]:
        if j not in noun_cate_dic:
            noun_cate_dic[j] = 1
        else:
            noun_cate_dic[j] +=1

print(noun_cate_dic)
```

```
{'group': 533, 'substance': 278, 'artifact': 1635, 'food': 333, 'act': 1440,
'quantity': 320, 'time': 252, 'state': 647, 'person': 1278, 'location': 400,
'cognition': 700, 'feeling': 214, 'attribute': 794, 'event': 458, 'body': 259,
'object': 292, 'animal': 287, 'relation': 118, 'communication': 1140, 'shape':
150, 'phenomenon': 143, 'possession': 237, 'Tops': 58, 'process': 101, 'motive':
17, 'plant': 215}
```

[23]:
```
# Verbs
verb_cate_dic = {}

for i in verb_dic:
    for j in verb_dic[i]:
        if j not in verb_cate_dic:
            verb_cate_dic[j] = 1
        else:
            verb_cate_dic[j] +=1

print(verb_cate_dic)
```

```
{'motion': 1219, 'stative': 980, 'consumption': 345, 'creation': 669, 'social':
1200, 'competition': 480, 'communication': 1512, 'contact': 1623, 'body': 703,
'possession': 851, 'change': 1380, 'emotion': 565, 'cognition': 845,
'perception': 624, 'weather': 107}
```

Book 2

[24]:
```
# Seprating nouns and verbs from lemmatized words
nouns2 = []
verbs2 = []
for word in lemmatized_S:
    for synset in wn.synsets(word):
        if "noun" in synset.lexname() and word not in nouns2:
            nouns2.append(word)

        elif "verb" in synset.lexname() and word not in verbs2:
            verbs2.append(word)
```

[25]:
```
print(nouns2[:100])
```

```
['Portrait', 'Drawing', 'Study', 'dusk', 'Summer', 'year', 'man', 'frock',
'coat', 'top', 'hat', 'cane', 'underbrush', 'corral', 'Buckler', 'farm', 'As',
'small', 'twig', 'fell', 'knee', 'green', 'wounded', 'shrubbery', 'grass',
'hand', 'tore', 'plant', 'wrist', 'hurt', 'time', 'caring', 'yellow', 'blowing',
'moustache', 'Dew', 'covering', 'twilight', 'leaf', 'like', 'myriad', 'face',
'damp', 'perspiration', 'struggle', 'existence', 'half', 'mile', 'standing',
'darkness', 'night', 'grove', 'white', 'birch', 'teeth', 'skull', 'He',
'creaking', 'gate', 'splashing', 'rain', 'depth', 'dark', 'cistern', 'heart',
'nearness', 'earth', 'faint', 'murmur', 'sleeper', 'turn', 'throw', 'arm',
'beloved', 'frog', 'skunk', 'cabbage', 'John', 'thrust', 'deep', 'bosom', 'It',
'heavy', 'prison', 'sin', 'one', 'may', 'punishment', 'feeling', 'way', 'high',
'plank', 'fence', 'sensing', 'finger', 'lay', 'head', 'ground', 'sleep', 'eye']
```

[26]:
```
print(verbs2[:100])
```

```
['Drawing', 'Study', 'dusk', 'Summer', 'man', 'dressed', 'frock', 'coat', 'top',
'hat', 'carrying', 'cane', 'crept', 'bordering', 'corral', 'farm', 'moved',
'twig', 'snapped', 'fell', 'green', 'wounded', 'grass', 'hand', 'tore', 'plant',
'hurt', 'rested', 'time', 'caring', 'knotted', 'yellow', 'blowing', 'falling',
'covering', 'leaf', 'like', 'face', 'damp', 'struggle', 'standing', 'white',
'birch', 'shimmered', 'heard', 'creaking', 'gate', 'splashing', 'rain', 'ached',
'earth', 'faint', 'murmur', 'moving', 'turn', 'throw', 'arm', 'frog', 'began',
'moaning', 'skunk', 'cabbage', 'thrust', 'bosom', 'seemed', 'wondered', 'sin',
'suffer', 'went', 'feeling', 'reached', 'plank', 'fence', 'sensing', 'finger',
'lay', 'resting', 'head', 'ground', 'tired', 'wanted', 'sleep', 'searched',
'straightened', 'turned', 'eye', 'star', 'thought', 'live', 'shake', 'dog',
'barking', 'dim', 'light', 'kept', 'winking', 'tree', 'swung', 'square', 'came']
```

[27]:
```
# Creating a dictionary of words and category of nouns they belong to.
noun_dic2 = {}
for noun in nouns2:
    d = []
    for synset in wn.synsets(noun):
        if "noun" in synset.lexname():
            if synset.lexname()[5:] not in d:
                d.append(synset.lexname()[5:])
    noun_dic2[noun] = d
```

[28]:
```
j = 0
for i in noun_dic2:
    print(i,":", noun_dic2[i])
    j +=1
    if j == 50:
        break
```

```
Portrait : ['communication', 'artifact']
Drawing : ['communication', 'artifact', 'act']
Study : ['act', 'cognition', 'communication', 'artifact', 'person']
dusk : ['time']
Summer : ['time']
year : ['time', 'group']
man : ['person', 'animal', 'location', 'artifact']
frock : ['artifact']
coat : ['artifact', 'animal']
top : ['location', 'time', 'state', 'attribute', 'artifact']
hat : ['artifact', 'act']
cane : ['artifact', 'plant']
underbrush : ['group']
corral : ['artifact']
Buckler : ['artifact']
farm : ['artifact']
As : ['substance', 'location', 'quantity', 'communication', 'body']
small : ['body', 'attribute']
twig : ['plant']
fell : ['substance', 'artifact', 'act']
knee : ['body', 'animal', 'artifact']
green : ['attribute', 'location', 'person', 'object', 'food', 'artifact']
wounded : ['group']
shrubbery : ['location', 'group']
grass : ['plant', 'person', 'food', 'artifact']
hand : ['body', 'person', 'communication', 'cognition', 'location', 'group',
'artifact', 'quantity', 'animal', 'act']
tore : ['artifact']
plant : ['artifact', 'Tops', 'person', 'cognition']
wrist : ['body']
hurt : ['state', 'feeling', 'event', 'act']
time : ['event', 'time', 'Tops', 'attribute']
caring : ['feeling']
yellow : ['attribute']
blowing : ['process']
moustache : ['body']
Dew : ['substance']
covering : ['object', 'artifact', 'act']
twilight : ['time', 'phenomenon', 'state']
leaf : ['plant', 'communication', 'artifact']
like : ['cognition']
myriad : ['quantity']
face : ['body', 'attribute', 'artifact', 'person', 'location', 'communication']
damp : ['state']
perspiration : ['body', 'process']
struggle : ['act']
existence : ['state', 'object']
half : ['quantity', 'time']
```

```
mile : ['quantity', 'event']
standing : ['state', 'communication', 'act']
darkness : ['state', 'location', 'cognition', 'attribute']
```

[29]:
```python
# Creating a dictionary of words and category of verbs they belong to.
verb_dic2 = {}
for verb in verbs2:
    d = []
    for synset in wn.synsets(verb):
        if "verb" in synset.lexname():
            if synset.lexname()[5:] not in d:
                d.append(synset.lexname()[5:])
    verb_dic2[verb] = d

j = 0
for i in verb_dic2:
    print(i,":", verb_dic2[i])
    j +=1
    if j == 50:
        break
```

```
Drawing : ['contact', 'possession', 'cognition', 'motion', 'creation',
'communication', 'emotion', 'consumption', 'competition', 'change', 'stative',
'social', 'body']
Study : ['cognition', 'perception']
dusk : ['change']
Summer : ['stative']
man : ['social', 'competition']
dressed : ['body', 'contact', 'change', 'motion', 'creation']
frock : ['body']
coat : ['contact', 'body']
top : ['stative', 'motion', 'possession', 'contact', 'change']
hat : ['body', 'possession']
carrying : ['contact', 'stative', 'motion', 'communication', 'social', 'change',
'competition', 'cognition', 'possession', 'creation', 'consumption', 'body']
cane : ['contact']
crept : ['motion']
bordering : ['contact', 'stative', 'possession']
corral : ['contact']
farm : ['social', 'possession', 'creation']
moved : ['motion', 'social', 'body', 'change', 'emotion', 'creation',
'possession', 'competition', 'communication']
twig : ['change', 'cognition']
snapped : ['communication', 'contact', 'change', 'motion', 'perception',
'emotion']
fell : ['contact', 'motion', 'change', 'stative', 'weather', 'social',
'possession']
```

```
green : ['change']
wounded : ['body', 'emotion']
grass : ['competition', 'contact', 'consumption', 'communication']
hand : ['possession', 'motion']
tore : ['contact', 'motion', 'body']
plant : ['contact', 'creation', 'possession', 'cognition']
hurt : ['perception', 'body', 'emotion', 'change']
rested : ['contact', 'communication', 'stative', 'body', 'change', 'social']
time : ['change', 'cognition']
caring : ['emotion', 'social']
knotted : ['creation', 'contact']
yellow : ['change']
blowing : ['body', 'weather', 'motion', 'perception', 'change', 'social',
'possession', 'contact', 'communication']
falling : ['motion', 'change', 'stative', 'weather', 'social', 'possession',
'contact']
covering : ['contact', 'stative', 'communication', 'motion', 'competition',
'social', 'perception', 'possession', 'body']
leaf : ['perception', 'motion', 'body']
like : ['emotion', 'cognition']
face : ['communication', 'competition', 'stative', 'motion', 'perception',
'contact']
damp : ['perception', 'change']
struggle : ['social', 'contact', 'motion', 'competition']
standing : ['contact', 'stative', 'cognition', 'competition']
white : ['change']
birch : ['contact']
shimmered : ['weather', 'stative']
heard : ['perception', 'cognition', 'social']
creaking : ['perception']
gate : ['possession', 'contact', 'change']
splashing : ['contact', 'motion', 'perception']
rain : ['weather']
ached : ['perception', 'emotion']
```

[30]:
```
# Creating a dictionary of frequency of category of nouns.
noun_cate_dic2 = {}

for i in noun_dic2:
    for j in noun_dic2[i]:
        if j not in noun_cate_dic2:
            noun_cate_dic2[j] = 1
        else:
            noun_cate_dic2[j] +=1

print(noun_cate_dic2)
```

```
{'communication': 485, 'artifact': 752, 'act': 728, 'cognition': 369, 'person':
```

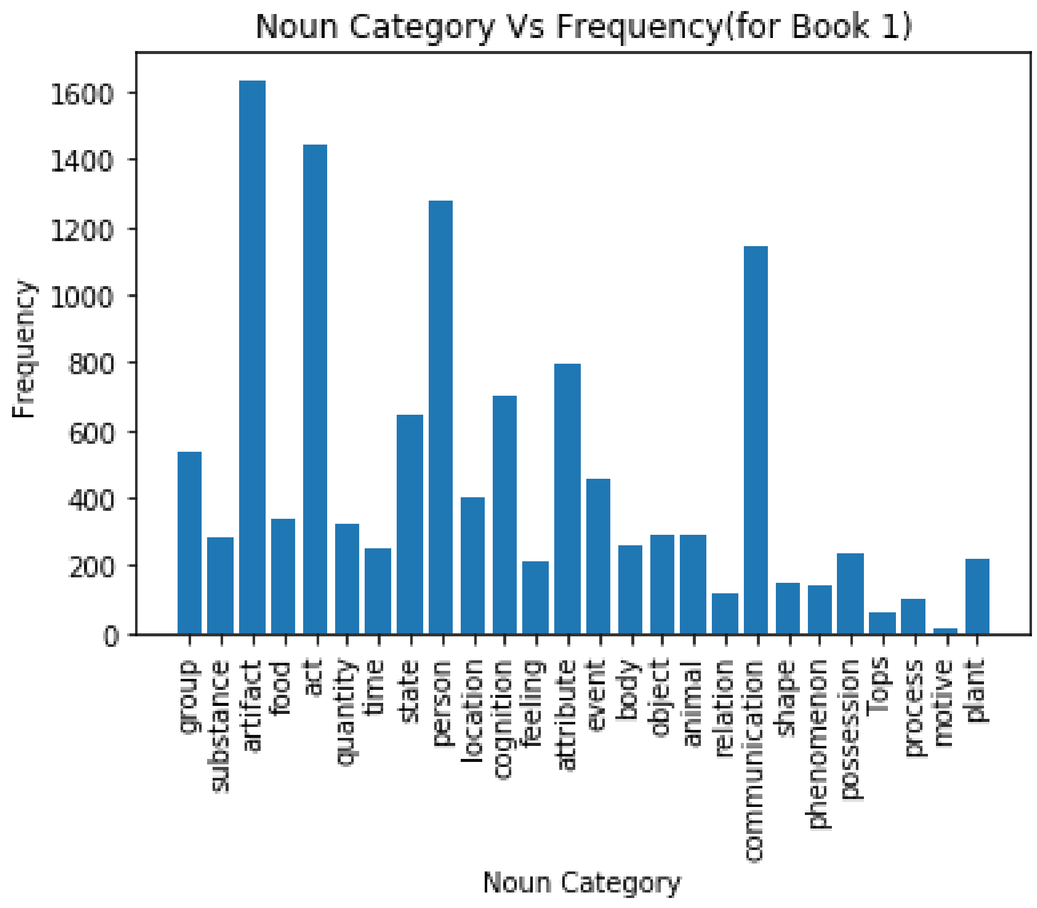
```
501, 'time': 152, 'group': 241, 'animal': 159, 'location': 209, 'state': 354,
'attribute': 424, 'plant': 104, 'substance': 125, 'quantity': 165, 'body': 169,
'object': 172, 'food': 145, 'Tops': 37, 'feeling': 140, 'event': 239, 'process':
58, 'phenomenon': 77, 'possession': 98, 'shape': 75, 'relation': 40, 'motive':
11}
```

[31]:
```
# Creating a dictionary of frequency of category of verbs.
verb_cate_dic2 = {}

for i in verb_dic2:
    for j in verb_dic2[i]:
        if j not in verb_cate_dic2:
            verb_cate_dic2[j] = 1
        else:
            verb_cate_dic2[j] +=1

print(verb_cate_dic2)
```

```
{'contact': 866, 'possession': 423, 'cognition': 501, 'motion': 674, 'creation':
352, 'communication': 729, 'emotion': 304, 'consumption': 206, 'competition':
284, 'change': 747, 'stative': 583, 'social': 630, 'body': 412, 'perception':
372, 'weather': 61}
```

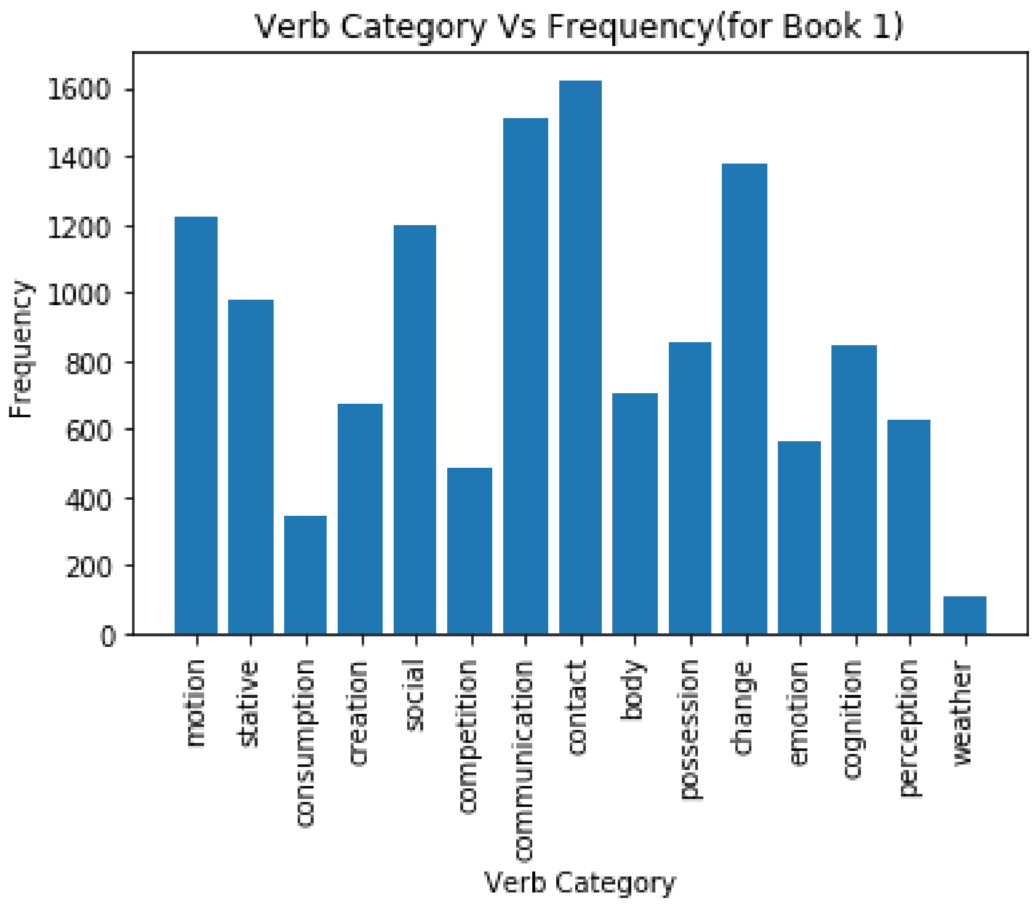
3.2 Ploting graphs

[32]:
```
# Bar graph of Noun Category and Frequency in Book1
plt.figure()
plt.bar(noun_cate_dic.keys(),noun_cate_dic.values())
plt.title('Noun Category Vs Frequency(for Book 1)')
plt.xlabel('Noun Category')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.show()
```



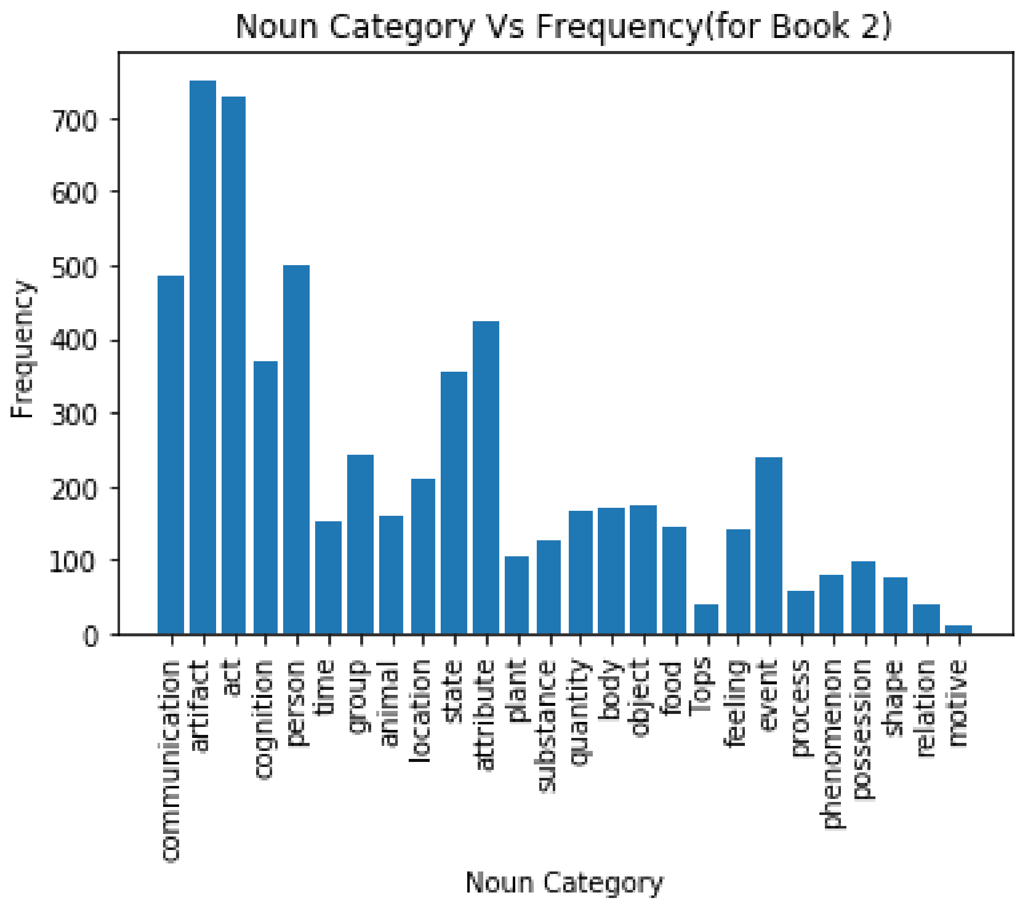
From Graph it is clearly visible that artifact category occures the most in nouns.

[33]:
```
# Bar graph of Verb Category and Frequency in Book1
plt.figure()
plt.bar(verb_cate_dic.keys(),verb_cate_dic.values())
plt.title('Verb Category Vs Frequency(for Book 1)')
plt.xlabel('Verb Category')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.show()
```



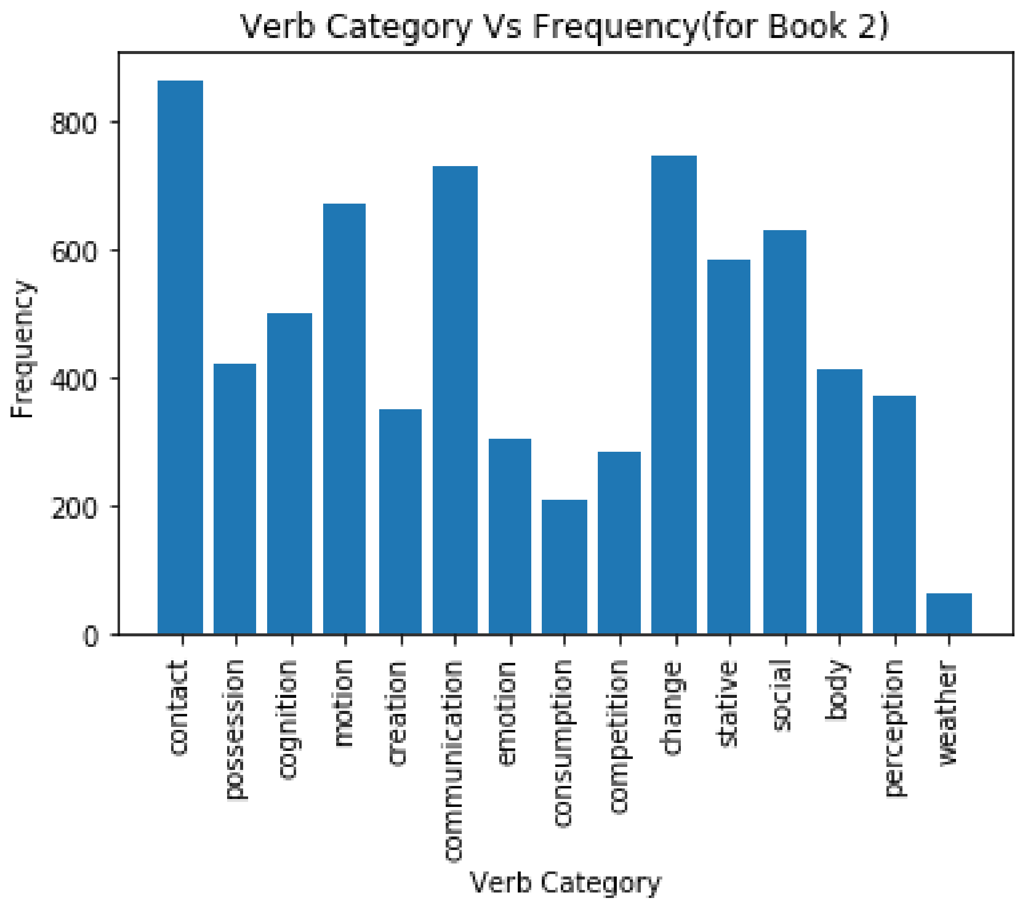
We are using bar graph instead of histogram as the x-axis of the data has discrete values.

[34]:
```
# Bar graph of Noun Category and Frequency in Book2
plt.figure()
plt.bar(noun_cate_dic2.keys(),noun_cate_dic2.values())
plt.title('Noun Category Vs Frequency(for Book 2)')
plt.xlabel('Noun Category')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.show()
```



From Graph it is clearly visible that artifact category occures the most in nouns.

[35]:
```
# Bar graph of Verb Category and Frequency in Book2
plt.figure()
plt.bar(verb_cate_dic2.keys(),verb_cate_dic2.values())
plt.title('Verb Category Vs Frequency(for Book 2)')
plt.xlabel('Verb Category')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.show()
```



From Graph it is clearly visible that contact category occures the most in verbs.

4 PART 2

4.1 Entity recongition and Classifiaction

First we import required libraries

[36]:
```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk import ne_chunk

import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm.load()

from seqeval.metrics import accuracy_score
from seqeval.metrics import classification_report
from seqeval.metrics import f1_score
```

Book 1

We use nltk for pos tagging and tokenizing the text which is required later for chunking and IOB tagging

[37]:
```
tagged_T2 = nltk.word_tokenize(T2)
tagged_T2 = nltk.pos_tag(tagged_T2)
print(tagged_T2[:100])
```

```
[('On', 'IN'), ('board', 'NN'), ('the', 'DT'), ('Rampatina', 'NNP'), ('liner',
'NN'), (',', ','), ('eleven', 'JJ'), ('days', 'NNS'), ('and', 'CC'), ('a',
'DT'), ('half', 'NN'), ('out', 'IN'), ('from', 'IN'), ('Liverpool', 'NNP'),
(',', ','), ('the', 'DT'), ('usual', 'JJ'), ('terrific', 'NN'), ('sensation',
'NN'), ('created', 'VBN'), ('by', 'IN'), ('the', 'DT'), ('appearance', 'NN'),
('of', 'IN'), ('the', 'DT'), ('pilot-yacht', 'NN'), ('prevailed', 'VBD'), ('.',
'.'), ('Necks', 'NNS'), ('were', 'VBD'), ('craned', 'VBN'), ('and', 'CC'),
('toes', 'NNS'), ('were', 'VBD'), ('trodden', 'JJ'), ('on', 'IN'), ('as', 'IN'),
('the', 'DT'), ('steamer', 'NN'), ('slackened', 'VBD'), ('speed', 'NN'), (',',
','), ('and', 'CC'), ('a', 'DT'), ('line', 'NN'), ('dexterously', 'RB'),
('thrown', 'VBN'), ('by', 'IN'), ('a', 'DT'), ('blue-jerseyed', 'JJ'), ('deck-
hand', 'NN'), ('was', 'VBD'), ('caught', 'VBN'), ('by', 'IN'), ('somebody',
'NN'), ('aboard', 'IN'), ('the', 'DT'), ('yacht', 'NN'), ('.', '.'), ('The',
'DT'), ('pilot', 'NN'), (',', ','), ('not', 'RB'), ('insensible', 'JJ'), ('to',
'TO'), ('the', 'DT'), ('fact', 'NN'), ('of', 'IN'), ('his', 'PRP$'), ('being',
'VBG'), ('a', 'DT'), ('personage', 'NN'), ('of', 'IN'), ('note', 'NN'), (',',
','), ('carefully', 'RB'), ('divested', 'VBD'), ('his', 'PRP$'), ('bearded',
'JJ'), ('countenance', 'NN'), ('of', 'IN'), ('all', 'DT'), ('expression', 'NN'),
('as', 'IN'), ('he', 'PRP'), ('saluted', 'VBD'), ('the', 'DT'), ('Captain',
'NNP'), (',', ','), ('and', 'CC'), ('taking', 'VBG'), ('from', 'IN'), ('the',
'DT'), ('deck-steward', 'JJ'), ('obsequiously', 'RB'), ('proffered', 'VBN'),
('salver', 'RP'), ('a', 'DT'), ('glass', 'NN'), ('containing', 'VBG')]
```

Using nltk function *ne_chunk* for chunking the text and named entity recognition

[38]:
```
results = ne_chunk(tagged_T2)
print(results[:100])
```

```
[('On', 'IN'), ('board', 'NN'), ('the', 'DT'), Tree('GPE', [('Rampatina',
'NNP')]), ('liner', 'NN'), (',', ','), ('eleven', 'JJ'), ('days', 'NNS'),
('and', 'CC'), ('a', 'DT'), ('half', 'NN'), ('out', 'IN'), ('from', 'IN'),
Tree('GPE', [('Liverpool', 'NNP')]), (',', ','), ('the', 'DT'), ('usual', 'JJ'),
('terrific', 'NN'), ('sensation', 'NN'), ('created', 'VBN'), ('by', 'IN'),
('the', 'DT'), ('appearance', 'NN'), ('of', 'IN'), ('the', 'DT'), ('pilot-
yacht', 'NN'), ('prevailed', 'VBD'), ('.', '.'), ('Necks', 'NNS'), ('were',
'VBD'), ('craned', 'VBN'), ('and', 'CC'), ('toes', 'NNS'), ('were', 'VBD'),
('trodden', 'JJ'), ('on', 'IN'), ('as', 'IN'), ('the', 'DT'), ('steamer', 'NN'),
('slackened', 'VBD'), ('speed', 'NN'), (',', ','), ('and', 'CC'), ('a', 'DT'),
('line', 'NN'), ('dexterously', 'RB'), ('thrown', 'VBN'), ('by', 'IN'), ('a',
'DT'), ('blue-jerseyed', 'JJ'), ('deck-hand', 'NN'), ('was', 'VBD'), ('caught',
'VBN'), ('by', 'IN'), ('somebody', 'NN'), ('aboard', 'IN'), ('the', 'DT'),
('yacht', 'NN'), ('.', '.'), ('The', 'DT'), ('pilot', 'NN'), (',', ','), ('not',
'RB'), ('insensible', 'JJ'), ('to', 'TO'), ('the', 'DT'), ('fact', 'NN'), ('of',
'IN'), ('his', 'PRP$'), ('being', 'VBG'), ('a', 'DT'), ('personage', 'NN'),
('of', 'IN'), ('note', 'NN'), (',', ','), ('carefully', 'RB'), ('divested',
'VBD'), ('his', 'PRP$'), ('bearded', 'JJ'), ('countenance', 'NN'), ('of', 'IN'),
('all', 'DT'), ('expression', 'NN'), ('as', 'IN'), ('he', 'PRP'), ('saluted',
'VBD'), ('the', 'DT'), Tree('GPE', [('Captain', 'NNP')]), (',', ','), ('and',
'CC'), ('taking', 'VBG'), ('from', 'IN'), ('the', 'DT'), ('deck-steward', 'JJ'),
('obsequiously', 'RB'), ('proffered', 'VBN'), ('salver', 'RP'), ('a', 'DT'),
('glass', 'NN'), ('containing', 'VBG')]
```

Creating a parse tree from the tagged text

[39]:
```
pattern = 'NP: {<DT>?<JJ>*<NN>}'
cp = nltk.RegexpParser(pattern)
cs = cp.parse(tagged_T2)
print(cs[:100])
```

```
[('On', 'IN'), Tree('NP', [('board', 'NN')]), ('the', 'DT'), ('Rampatina',
'NNP'), Tree('NP', [('liner', 'NN')]), (',', ','), ('eleven', 'JJ'), ('days',
'NNS'), ('and', 'CC'), Tree('NP', [('a', 'DT'), ('half', 'NN')]), ('out', 'IN'),
('from', 'IN'), ('Liverpool', 'NNP'), (',', ','), Tree('NP', [('the', 'DT'),
('usual', 'JJ'), ('terrific', 'NN')]), Tree('NP', [('sensation', 'NN')]),
('created', 'VBN'), ('by', 'IN'), Tree('NP', [('the', 'DT'), ('appearance',
'NN')]), ('of', 'IN'), Tree('NP', [('the', 'DT'), ('pilot-yacht', 'NN')]),
('prevailed', 'VBD'), ('.', '.'), ('Necks', 'NNS'), ('were', 'VBD'), ('craned',
'VBN'), ('and', 'CC'), ('toes', 'NNS'), ('were', 'VBD'), ('trodden', 'JJ'),
('on', 'IN'), ('as', 'IN'), Tree('NP', [('the', 'DT'), ('steamer', 'NN')]),
('slackened', 'VBD'), Tree('NP', [('speed', 'NN')]), (',', ','), ('and', 'CC'),
Tree('NP', [('a', 'DT'), ('line', 'NN')]), ('dexterously', 'RB'), ('thrown',
'VBN'), ('by', 'IN'), Tree('NP', [('a', 'DT'), ('blue-jerseyed', 'JJ'), ('deck-
hand', 'NN')]), ('was', 'VBD'), ('caught', 'VBN'), ('by', 'IN'), Tree('NP',
[('somebody', 'NN')]), ('aboard', 'IN'), Tree('NP', [('the', 'DT'), ('yacht',
'NN')]), ('.', '.'), Tree('NP', [('The', 'DT'), ('pilot', 'NN')]), (',', ','),
('not', 'RB'), ('insensible', 'JJ'), ('to', 'TO'), Tree('NP', [('the', 'DT'),
('fact', 'NN')]), ('of', 'IN'), ('his', 'PRP$'), ('being', 'VBG'), Tree('NP',
[('a', 'DT'), ('personage', 'NN')]), ('of', 'IN'), Tree('NP', [('note', 'NN')]),
(',', ','), ('carefully', 'RB'), ('divested', 'VBD'), ('his', 'PRP$'),
Tree('NP', [('bearded', 'JJ'), ('countenance', 'NN')]), ('of', 'IN'), Tree('NP',
[('all', 'DT'), ('expression', 'NN')]), ('as', 'IN'), ('he', 'PRP'), ('saluted',
'VBD'), ('the', 'DT'), ('Captain', 'NNP'), (',', ','), ('and', 'CC'), ('taking',
'VBG'), ('from', 'IN'), ('the', 'DT'), ('deck-steward', 'JJ'), ('obsequiously',
'RB'), ('proffered', 'VBN'), ('salver', 'RP'), Tree('NP', [('a', 'DT'),
('glass', 'NN')]), ('containing', 'VBG'), ('four-fingers', 'NNS'), ('of', 'IN'),
('neat', 'JJ'), ('Bourbon', 'NNP'), Tree('NP', [('whisky', 'NN')]), (',', ','),
('concealed', 'VBD'), ('its', 'PRP$'), ('contents', 'NNS'), ('about', 'IN'),
('his', 'PRP$'), Tree('NP', [('person', 'NN')]), ('without', 'IN'), Tree('NP',
[('perceptible', 'JJ'), ('emotion', 'NN')]), (',', ','), ('and', 'CC')]
```

Performing IOB tagging using nltk

[40]:
```
from nltk.chunk import tree2conlltags
from pprint import pprint
iob_tagged = tree2conlltags(cs)
pprint(iob_tagged[:50])
```

```
[('On', 'IN', 'O'),
 ('board', 'NN', 'B-NP'),
 ('the', 'DT', 'O'),
 ('Rampatina', 'NNP', 'O'),
 ('liner', 'NN', 'B-NP'),
 (',', ',', 'O'),
 ('eleven', 'JJ', 'O'),
 ('days', 'NNS', 'O'),
 ('and', 'CC', 'O'),
 ('a', 'DT', 'B-NP'),
 ('half', 'NN', 'I-NP'),
 ('out', 'IN', 'O'),
 ('from', 'IN', 'O'),
 ('Liverpool', 'NNP', 'O'),
 (',', ',', 'O'),
 ('the', 'DT', 'B-NP'),
 ('usual', 'JJ', 'I-NP'),
 ('terrific', 'NN', 'I-NP'),
 ('sensation', 'NN', 'B-NP'),
 ('created', 'VBN', 'O'),
 ('by', 'IN', 'O'),
 ('the', 'DT', 'B-NP'),
 ('appearance', 'NN', 'I-NP'),
 ('of', 'IN', 'O'),
 ('the', 'DT', 'B-NP'),
 ('pilot-yacht', 'NN', 'I-NP'),
 ('prevailed', 'VBD', 'O'),
 ('.', '.', 'O'),
 ('Necks', 'NNS', 'O'),
 ('were', 'VBD', 'O'),
 ('craned', 'VBN', 'O'),
 ('and', 'CC', 'O'),
 ('toes', 'NNS', 'O'),
 ('were', 'VBD', 'O'),
 ('trodden', 'JJ', 'O'),
 ('on', 'IN', 'O'),
 ('as', 'IN', 'O'),
 ('the', 'DT', 'B-NP'),
 ('steamer', 'NN', 'I-NP'),
 ('slackened', 'VBD', 'O'),
 ('speed', 'NN', 'B-NP'),
 (',', ',', 'O'),
 ('and', 'CC', 'O'),
 ('a', 'DT', 'B-NP'),
 ('line', 'NN', 'I-NP'),
 ('dexterously', 'RB', 'O'),
 ('thrown', 'VBN', 'O'),
 ('by', 'IN', 'O'),
 ('a', 'DT', 'B-NP'),
 ('blue-jerseyed', 'JJ', 'I-NP')]
```

Representing the IOB tags using print function

[41]:
```
j = 0
for word, pos, ner in iob_tagged:
    print(word, pos, ner)
    j +=1
    if j == 50:
        break
```

```
On IN O
board NN B-NP
the DT O
Rampatina NNP O
liner NN B-NP
, , O
eleven JJ O
days NNS O
and CC O
a DT B-NP
half NN I-NP
out IN O
from IN O
Liverpool NNP O
, , O
the DT B-NP
usual JJ I-NP
terrific NN I-NP
sensation NN B-NP
created VBN O
by IN O
the DT B-NP
appearance NN I-NP
```

```
of IN O
the DT B-NP
pilot-yacht NN I-NP
prevailed VBD O
. . O
Necks NNS O
were VBD O
craned VBN O
and CC O
toes NNS O
were VBD O
trodden JJ O
on IN O
as IN O
the DT B-NP
steamer NN I-NP
slackened VBD O
speed NN B-NP
, , O
and CC O
a DT B-NP
line NN I-NP
dexterously RB O
thrown VBN O
by IN O
a DT B-NP
blue-jerseyed JJ I-NP
```

Below we are using spacy library for named entity recognition, classification and IOB tagging

***nlp* function of spacy converts the text to tokens, does IOB tagging and entity recongnition and classification**

[42]:
```
T2 = nlp(T2)
```

Below are the IOB tags and entity types of the text

[43]:
```
for X in T2[6350:6360]:
    print(X, X.ent_iob_, X.ent_type_)
```

```
weeds O
were O
eminently O
becoming O
. O
Dear O
Captain O
Ranking O
```

```
, O
how O
```

We extract all the labels of the entity types, labeled by spacy

[44]:
```
labels = [x.label_ for x in T2.ents]
```

***Counter* function of spacy counts all the types of labels in the text**

[45]:
```
Counter(labels)
```

[45]:
```
Counter({'PERSON': 2265,
         'DATE': 262,
         'GPE': 370,
         'PRODUCT': 72,
         'CARDINAL': 580,
         'ORDINAL': 162,
         'NORP': 261,
         'ORG': 562,
         'QUANTITY': 54,
         'TIME': 148,
         'WORK_OF_ART': 51,
         'LOC': 63,
         'FAC': 131,
         'MONEY': 13,
         'LANGUAGE': 24,
         'LAW': 12,
         'EVENT': 15})
```

Book 2

We use nltk for pos tagging and tokenizing the text which is required later for chunking and IOB tagging

[46]:
```
tagged_S2 = nltk.word_tokenize(S2)
tagged_S2 = nltk.pos_tag(tagged_S2)
tagged_S2[:100]
```

[46]:
```
[('rtrait', 'NN'),
 ('Portrait', 'NNP'),
 ('Drawing', 'NNP'),
 ('Portrait', 'NNP'),
 ('Study', 'NNP'),
 ('Portrait', 'NNP'),
 ('A', 'NNP'),
 ('A', 'NNP'),
 ('Toward', 'NNP'),
 ('dusk', 'NN'),
 (',', ','),
 ('in', 'IN'),
('the', 'DT'),
('Summer', 'NNP'),
('of', 'IN'),
('the', 'DT'),
('year', 'NN'),
(',', ','),
('a', 'DT'),
('man', 'NN'),
('dressed', 'VBN'),
('in', 'IN'),
('a', 'DT'),
('frock', 'NN'),
('coat', 'NN'),
('and', 'CC'),
('top', 'JJ'),
('hat', 'NN'),
(',', ','),
('and', 'CC'),
('carrying', 'VBG'),
('a', 'DT'),
('cane', 'NN'),
(',', ','),
('crept', 'VBD'),
('through', 'IN'),
('the', 'DT'),
('underbrush', 'JJ'),
('bordering', 'VBG'),
('the', 'DT'),
('corral', 'NN'),
('of', 'IN'),
('the', 'DT'),
('Buckler', 'NNP'),
('farm', 'NN'),
('.', '.'),
('As', 'IN'),
('he', 'PRP'),
('moved', 'VBD'),
(',', ','),
('small', 'JJ'),
('twigs', 'NNS'),
('snapped', 'VBD'),
(',', ','),
('fell', 'VBD'),
('and', 'CC'),
('were', 'VBD'),
('silent', 'JJ'),
('.', '.'),
 ('His', 'PRP$'),
 ('knees', 'NNS'),
 ('were', 'VBD'),
 ('green', 'JJ'),
 ('from', 'IN'),
 ('wounded', 'VBN'),
 ('shrubbery', 'NN'),
 ('and', 'CC'),
 ('grass', 'NN'),
 (',', ','),
 ('and', 'CC'),
 ('his', 'PRP$'),
 ('outspread', 'JJ'),
 ('hands', 'NNS'),
 ('tore', 'RB'),
 ('unheeded', 'JJ'),
 ('plants', 'NNS'),
 ('.', '.'),
 ('His', 'PRP$'),
 ('wrists', 'NNS'),
 ('hurt', 'VBP'),
 ('him', 'PRP'),
 ('and', 'CC'),
 ('he', 'PRP'),
 ('rested', 'VBD'),
 ('from', 'IN'),
 ('time', 'NN'),
 ('to', 'TO'),
 ('time', 'NN'),
 (',', ','),
 ('always', 'RB'),
 ('caring', 'VBG'),
 ('for', 'IN'),
 ('his', 'PRP$'),
 ('hat', 'NN'),
 ('and', 'CC'),
 ('knotted', 'VBD'),
 ('yellow', 'JJ'),
 ('cane', 'NN'),
 (',', ','),
 ('blowing', 'VBG')]
```

Using nltk function *ne_chunk* for chunking the text and named entity recognition

[47]:
```
results2 = ne_chunk(tagged_S2)
print(results2[:100])
```

```
[('rtrait', 'NN'), Tree('PERSON', [('Portrait', 'NNP'), ('Drawing', 'NNP'),
('Portrait', 'NNP'), ('Study', 'NNP'), ('Portrait', 'NNP')]), ('A', 'NNP'),
('A', 'NNP'), ('Toward', 'NNP'), ('dusk', 'NN'), (',', ','), ('in', 'IN'),
('the', 'DT'), Tree('ORGANIZATION', [('Summer', 'NNP')]), ('of', 'IN'), ('the',
'DT'), ('year', 'NN'), (',', ','), ('a', 'DT'), ('man', 'NN'), ('dressed',
'VBN'), ('in', 'IN'), ('a', 'DT'), ('frock', 'NN'), ('coat', 'NN'), ('and',
'CC'), ('top', 'JJ'), ('hat', 'NN'), (',', ','), ('and', 'CC'), ('carrying',
'VBG'), ('a', 'DT'), ('cane', 'NN'), (',', ','), ('crept', 'VBD'), ('through',
'IN'), ('the', 'DT'), ('underbrush', 'JJ'), ('bordering', 'VBG'), ('the', 'DT'),
('corral', 'NN'), ('of', 'IN'), ('the', 'DT'), Tree('ORGANIZATION', [('Buckler',
'NNP')]), ('farm', 'NN'), ('.', '.'), ('As', 'IN'), ('he', 'PRP'), ('moved',
'VBD'), (',', ','), ('small', 'JJ'), ('twigs', 'NNS'), ('snapped', 'VBD'), (',',
','), ('fell', 'VBD'), ('and', 'CC'), ('were', 'VBD'), ('silent', 'JJ'), ('.',
'.'), ('His', 'PRP$'), ('knees', 'NNS'), ('were', 'VBD'), ('green', 'JJ'),
('from', 'IN'), ('wounded', 'VBN'), ('shrubbery', 'NN'), ('and', 'CC'),
('grass', 'NN'), (',', ','), ('and', 'CC'), ('his', 'PRP$'), ('outspread',
'JJ'), ('hands', 'NNS'), ('tore', 'RB'), ('unheeded', 'JJ'), ('plants', 'NNS'),
('.', '.'), ('His', 'PRP$'), ('wrists', 'NNS'), ('hurt', 'VBP'), ('him', 'PRP'),
('and', 'CC'), ('he', 'PRP'), ('rested', 'VBD'), ('from', 'IN'), ('time', 'NN'),
('to', 'TO'), ('time', 'NN'), (',', ','), ('always', 'RB'), ('caring', 'VBG'),
('for', 'IN'), ('his', 'PRP$'), ('hat', 'NN'), ('and', 'CC'), ('knotted',
'VBD'), ('yellow', 'JJ'), ('cane', 'NN'), (',', ','), ('blowing', 'VBG'),
('through', 'IN'), ('his', 'PRP$'), ('moustache', 'NN'), ('.', '.')]
```

Creating a parse tree from the tagged text

[48]:
```python
pattern = 'NP: {<DT>?<JJ>*<NN>}'
cp2 = nltk.RegexpParser(pattern)
cs2 = cp2.parse(tagged_S2)
print(cs2[:100])
```

```
[Tree('NP', [('rtrait', 'NN')]), ('Portrait', 'NNP'), ('Drawing', 'NNP'),
('Portrait', 'NNP'), ('Study', 'NNP'), ('Portrait', 'NNP'), ('A', 'NNP'), ('A',
'NNP'), ('Toward', 'NNP'), Tree('NP', [('dusk', 'NN')]), (',', ','), ('in',
'IN'), ('the', 'DT'), ('Summer', 'NNP'), ('of', 'IN'), Tree('NP', [('the',
'DT'), ('year', 'NN')]), (',', ','), Tree('NP', [('a', 'DT'), ('man', 'NN')]),
('dressed', 'VBN'), ('in', 'IN'), Tree('NP', [('a', 'DT'), ('frock', 'NN')]),
Tree('NP', [('coat', 'NN')]), ('and', 'CC'), Tree('NP', [('top', 'JJ'), ('hat',
'NN')]), (',', ','), ('and', 'CC'), ('carrying', 'VBG'), Tree('NP', [('a',
'DT'), ('cane', 'NN')]), (',', ','), ('crept', 'VBD'), ('through', 'IN'),
('the', 'DT'), ('underbrush', 'JJ'), ('bordering', 'VBG'), Tree('NP', [('the',
'DT'), ('corral', 'NN')]), ('of', 'IN'), ('the', 'DT'), ('Buckler', 'NNP'),
Tree('NP', [('farm', 'NN')]), ('.', '.'), ('As', 'IN'), ('he', 'PRP'), ('moved',
'VBD'), (',', ','), ('small', 'JJ'), ('twigs', 'NNS'), ('snapped', 'VBD'), (',',
','), ('fell', 'VBD'), ('and', 'CC'), ('were', 'VBD'), ('silent', 'JJ'), ('.',
'.'), ('His', 'PRP$'), ('knees', 'NNS'), ('were', 'VBD'), ('green', 'JJ'),
('from', 'IN'), ('wounded', 'VBN'), Tree('NP', [('shrubbery', 'NN')]), ('and',
'CC'), Tree('NP', [('grass', 'NN')]), (',', ','), ('and', 'CC'), ('his',
'PRP$'), ('outspread', 'JJ'), ('hands', 'NNS'), ('tore', 'RB'), ('unheeded',
'JJ'), ('plants', 'NNS'), ('.', '.'), ('His', 'PRP$'), ('wrists', 'NNS'),
('hurt', 'VBP'), ('him', 'PRP'), ('and', 'CC'), ('he', 'PRP'), ('rested',
'VBD'), ('from', 'IN'), Tree('NP', [('time', 'NN')]), ('to', 'TO'), Tree('NP',
[('time', 'NN')]), (',', ','), ('always', 'RB'), ('caring', 'VBG'), ('for',
'IN'), ('his', 'PRP$'), Tree('NP', [('hat', 'NN')]), ('and', 'CC'), ('knotted',
'VBD'), Tree('NP', [('yellow', 'JJ'), ('cane', 'NN')]), (',', ','), ('blowing',
'VBG'), ('through', 'IN'), ('his', 'PRP$'), Tree('NP', [('moustache', 'NN')]),
('.', '.'), ('Dew', 'NNP'), ('had', 'VBD'), ('been', 'VBN')]
```

Performing IOB tagging using nltk

[49]:
```python
from nltk.chunk import conlltags2tree, tree2conlltags
from pprint import pprint
iob_tagged2 = tree2conlltags(cs2)
pprint(iob_tagged2[:50])
```

```
[('rtrait', 'NN', 'B-NP'),
 ('Portrait', 'NNP', 'O'),
 ('Drawing', 'NNP', 'O'),
 ('Portrait', 'NNP', 'O'),
 ('Study', 'NNP', 'O'),
 ('Portrait', 'NNP', 'O'),
 ('A', 'NNP', 'O'),
 ('A', 'NNP', 'O'),
 ('Toward', 'NNP', 'O'),
 ('dusk', 'NN', 'B-NP'),
 (',', ',', 'O'),
 ('in', 'IN', 'O'),
 ('the', 'DT', 'O'),
 ('Summer', 'NNP', 'O'),
 ('of', 'IN', 'O'),
 ('the', 'DT', 'B-NP'),
 ('year', 'NN', 'I-NP'),
 (',', ',', 'O'),
 ('a', 'DT', 'B-NP'),
 ('man', 'NN', 'I-NP'),
 ('dressed', 'VBN', 'O'),
 ('in', 'IN', 'O'),
 ('a', 'DT', 'B-NP'),
 ('frock', 'NN', 'I-NP'),
 ('coat', 'NN', 'B-NP'),
 ('and', 'CC', 'O'),
 ('top', 'JJ', 'B-NP'),
 ('hat', 'NN', 'I-NP'),
 (',', ',', 'O'),
 ('and', 'CC', 'O'),
 ('carrying', 'VBG', 'O'),
 ('a', 'DT', 'B-NP'),
 ('cane', 'NN', 'I-NP'),
 (',', ',', 'O'),
 ('crept', 'VBD', 'O'),
 ('through', 'IN', 'O'),
 ('the', 'DT', 'O'),
 ('underbrush', 'JJ', 'O'),
 ('bordering', 'VBG', 'O'),
 ('the', 'DT', 'B-NP'),
 ('corral', 'NN', 'I-NP'),
 ('of', 'IN', 'O'),
 ('the', 'DT', 'O'),
 ('Buckler', 'NNP', 'O'),
 ('farm', 'NN', 'B-NP'),
 ('.', '.', 'O'),
 ('As', 'IN', 'O'),
 ('he', 'PRP', 'O'),
 ('moved', 'VBD', 'O'),
 (',', ',', 'O')]
```

Representing the IOB tags using print function

[50]:
```
j = 0
for word, pos, ner in iob_tagged2:
    print(word, pos, ner)
    j +=1
    if j == 10:
        break
```

```
rtrait NN B-NP
Portrait NNP O
Drawing NNP O
Portrait NNP O
Study NNP O
Portrait NNP O
A NNP O
A NNP O
Toward NNP O
dusk NN B-NP
```

Below we are using spacy library for named entity recognition, classification and IOB tagging

***nlp* function of spacy converts the text to tokens, does IOB tagging and entity recongnition and classification**

[51]:
```
S2 = nlp(S2)
```

Below are the IOB tags and entity types of the text

[52]:
```
for X in S2[6350:6360]:
    print(X, X.ent_iob_, X.ent_type_)
```

```
 O
 O
 O
Oh O
, O
my O
God O
! O
[ O
_ O
```

We extract all the labels of the entity types, labeled by spacy

[53]:
```
labels2 = [x.label_ for x in S2.ents]
```

***Counter* function of spacy counts all the types of labels in the text**

[54]:
```
Counter(labels2)
```

[54]:
```
Counter({'ORG': 92,
         'TIME': 67,
         'DATE': 120,
         'PERSON': 380,
         'QUANTITY': 9,
         'LOC': 25,
         'CARDINAL': 229,
         'FAC': 9,
         'NORP': 52,
         'GPE': 61,
         'PRODUCT': 6,
         'ORDINAL': 30,
         'MONEY': 1,
         'WORK_OF_ART': 16,
         'EVENT': 2,
         'LANGUAGE': 7})
```

5 Performanace Measures

We used seqeval library for finding the F-score and accuracy of our classifier
BOOK 1

Since we only require the entity types specified in fig. 22.1 of chapter 22 we only extract those in our predicted entity list. Below only a part of the text (from 6350 to 6750 character) is shown because we use the same for finding the F-score

[55]:
```
entity_pred = []

for X in T2[6350:6750]:
    if X.ent_type_ == "GPE" or X.ent_type_ == "PERSON" or X.ent_type_ == "ORG"␣
 ↪or X.ent_type_ == "FAC" or X.ent_type_ == "LOC":
        entity_pred.append('B-'+X.ent_type_)
    elif X.ent_type_=="":
        entity_pred.append('O')

print(entity_pred)
```

```
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-GPE', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'B-FAC', 'B-FAC', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PERSON',
'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'O',
'O', 'B-PERSON', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O']
```

Now we convert this predicted entity list into a list of list manually such that each list element consists of only one entity

[56]:

```
entity_pred = [['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],␣
 ↪['B-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],['B-PERSON', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O'], ['B-FAC', 'I-FAC', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON',␣
 ↪'O', 'O'], ['B-PERSON', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O']]
```

We now create a entity type list of the actual values in the paragraph which we tagged manually

[57]:

```
```

```
entity_true = [['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],␣
 ↪['B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],['B-PERSON',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-LOC', 'I-LOC', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O'], ['B-PERSON', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON',␣
 ↪'I-PERSON', 'O', 'O'], ['B-PERSON', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O']]
```

Finding the F-score between actual entity types and predicted entity types

[58]:
```
f1_score(entity_true, entity_pred)
```

[58]:
```
0.8888888888888888
```

Below is the accuracy score for the same

[59]:
```
accuracy_score(entity_true, entity_pred)
```

[59]:
```
0.9924433249370277
```

BOOK 2

Since we only require the entity types specified in fig. 22.1 of chapter 22 we only extract those in our predicted entity list. Below only a part of the text (from 7050 to 7500 character) is shown because we use the same for finding the F-score

[60]:
```
entity_pred2 = []

for X in S2[7050:7500]:
    if X.ent_type_ == "GPE" or X.ent_type_ == "PERSON" or X.ent_type_ == "ORG"␣
 ↪or X.ent_type_ == "FAC" or X.ent_type_ == "LOC":
        entity_pred2.append(X.ent_type_)
    elif X.ent_type_=="":
        entity_pred2.append('O')

print(entity_pred2)
```

```
['ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'ORG', 'ORG', 'ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'PERSON', 'PERSON', 'PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'PERSON', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'GPE', 'O', 'O', 'O', 'O', 'PERSON', 'PERSON', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'PERSON', 'PERSON', 'PERSON', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'GPE', 'O', 'O', 'PERSON', 'PERSON', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'ORG', 'ORG', 'ORG', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'GPE', 'GPE', 'GPE', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
```

Now we convert this predicted entity list into a list of list manually such that each list element consists of only one entity

[61]:
```
```

```
entity_pred2 = [['O', 'O', 'O', 'O', 'B-ORG', 'I-ORG', 'I-ORG', 'I-ORG', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'O',␣
 ↪'O'], ['B-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-LOC', 'O', 'O'], ['B-ORG', 'I-ORG',␣
 ↪'I-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],␣
 ↪['B-GPE', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-LOC', 'O', 'O'], ['B-PERSON',␣
 ↪'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O'], ['B-ORG', 'I-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],␣
 ↪['B-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O']]
```

We now create a entity type list of the actual values in the paragraph which we tagged manually

[62]:
```
```

```
entity_true2 = [['O', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'I-PERSON',␣
 ↪'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['O', 'O',␣
 ↪'O', 'O'], ['B-PERSON', 'I-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'I-PERSON', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-LOC', 'O', 'O'],␣
 ↪['B-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON',␣
 ↪'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O'], ['B-GPE', 'O', 'O', 'O', 'O'], ['B-PERSON',␣
 ↪'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],␣
 ↪['B-LOC', 'O', 'O'], ['B-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON',␣
 ↪'I-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['B-PERSON', 'I-PERSON',␣
 ↪'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',␣
 ↪'O']]
```

Finding the F-score between actual entity types and predicted entity types

[63]:
```
f1_score(entity_true2, entity_pred2)
```

[63]:
```
0.64
```

Below is the accuracy score for the same

[64]:
```
accuracy_score(entity_true2, entity_pred2)
```

[64]:
```
0.9640449438202248
```

6 Additonal Exercise

We were not able to find pre-built algorithms or classifiers for relation extraction so we tried to do it manualy. Following are some of the relations we extracted in both the books.

Book 1 [“Rampatina Liner”, “Liverpool”] - Part-Of [“pilot”, “yatch”] - Affiliation [“Pressman”, “notebook”] - Affiliation [“Pandemonium”, “Mid-Ocean”] - Part-Of

Book 2 [“Corral”, “Buckler Farm”] - Part-Of [“Thorns”, “Crown”] - Part-Of [“fields”, “house”] - Geospatial [“Vera Sovna”, “England”] - Affiliation [“townsfolk”, “landholders”] - Affiliations [“Cornwall”, “Hucksteppe”] - Geospatial [“Storm”, “Rain”] - Affiliations

[ ]: