NLP PROJECT FINAL REPORT

Team Name : Brute Force

Team Members

- Ansh Mehta : 17ucs027
- Shubham Pabuwal : 17ucs157
- Abhinav Tiwari : 17ucs007

Index

ROUND 1 - BOOK 01

1. Importing and Reading The Book 01

Title- Psychotherapy
Author- Hugo Münsterberg
Words- 109,765
Characters (no spaces)- 540,321
Characters (with spaces)- 647,282
Paragraphs- 9,875
Lines- 10,838

In [50]:
```
import nltk
import matplotlib
from nltk.corpus import wordnet as wn
#nltk.download('averaged_perceptron_tagger')
```

In [51]:
```
file = open("Psychotherapy by Hugo Münsterberg.txt","r",encoding="utf-8-sig")
```

In [3]:
```
text = file.read()
```

2. Text-Preprocessing

All the lines which are not required in this model are removed manually. Lines that are removed are- start-114, 188-251, 439-454, 860-868, 1525-1533, 2241-2249, 3199-3207, 3975-3990, 4584-4592, 5253-5261, 5888-5896, 7543-7551, 8046-8062, 8710-8720, 9258-9264, 9945-end.

In [4]:
```
postString = text.split("\n")
text  = " ".join(postString)
```

In [5]:
```
text = text.lower()
#print(text)
```

To remove any HTML tags from the downloaded text file we use following regular expression

It is useful beacuse html tags are just for the representation of the file and they do not contribute anything in this mode l.

In [6]:
```
import re
text = re.sub('<[^<]+?>','',text)
```

Converting Unicodes(ä, ś, đ, è, â, ã, etc.) into corresponding alphabet

Because we are using a PoS tagger which uses english dictionary

In [7]:
```
import unicodedata
text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
```

Converting Contraction words into their original form

For e.g.: converting "what's" --> "what is"
It is helpful because only removing punctuations marks first will not help us.
eg. can't --> cant(if we remove punctuation marks without expanding words,doesn't have any meaning in english dictionary)
can't --> cannot(if we expand word and it has meaning in english dictionary)

In [8]:
```
from contractions import contractions_dict


for word in contractions_dict.keys():
    if word in text:
        text = text.replace(word, contractions_dict[word])

text_expanded = text
```

Removing punctuation marks from the text

e.g. ",","?","!",etc.

In [9]:
```
import string
table = str.maketrans('','',string.punctuation)
text = text.translate(table)
```

3. Tokenizing the input text

Tokenization is the process by which big quantity of text is divided into smaller parts called tokens.
These tokens are very useful for finding such patterns as well as is considered as a base step for stemming and lemmatization.
We use the method word_tokenize() to split a string (passed as argument) into list of words.

In [10]:
```
words = nltk.word_tokenize(text)
```

NLTK in python has a function FreqDist which gives you the frequency of each unique word within a text

Frequency distributions are generally constructed by running a number of experiments, and incrementing the count for a sample every time it is an outcome of an experiment.

In [11]:
```
from nltk.probability import FreqDist
fdist = FreqDist(words)
```

Graph of 10 most appering words in the text

plot function in nltk.probability.FreqDist creates a frequency distribution plot of the most common words. Takes a no as parameter specifying no of most common words to show in plot.

In [12]:
```
fdist.plot(10)
```

```
<Figure size 640x480 with 1 Axes>
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d387b01708>
```

In [13]:
```
for key,val in list(fdist.most_common())[:10]:
    print (str(key) + ':' + str(val))
```

```
the:8837
of:4770
and:3114
to:2894
a:2281
in:2208
is:1681
that:1537
which:1281
it:1019
```

Percentage of stop words in the original text

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. In this model we are using a predefined set of stopwords present in nltk library.

In [14]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
stopwords_text=[]
stopwords_text = [w for w in words if w in stop_words]
print("Precentage of stopwords -",len(stopwords_text)*100/len(words),"%")
```

```
Precentage of stopwords - 51.657594314659406 %
```

4. Word Cloud

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

Word Cloud with stopwords in the text

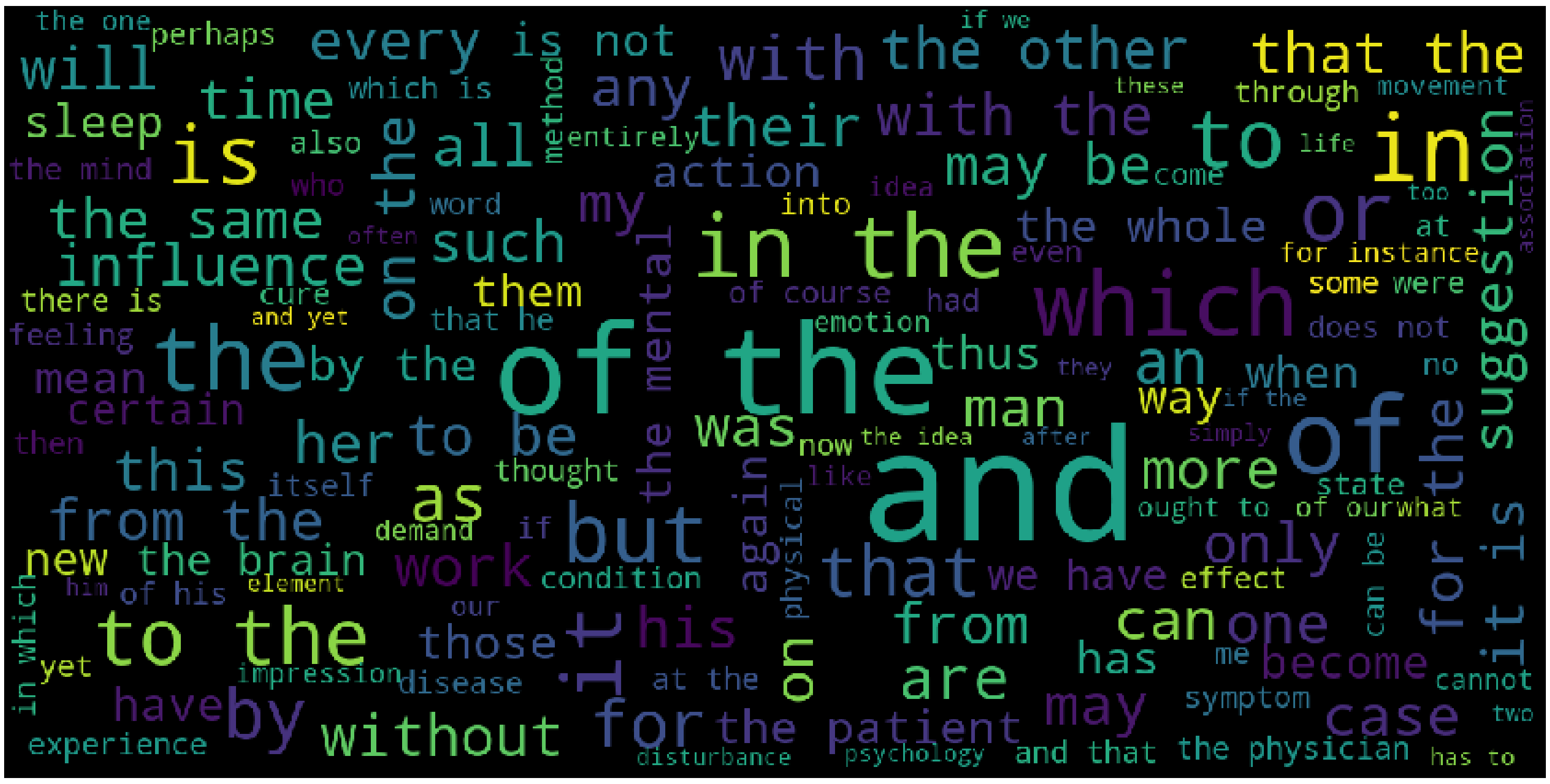
In [15]:
```
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image
```

Created a function named generateWordCloud which takes two arguments first is the pre processed text and second is the title of the image. Function uses predefined function WordCloud in wordcloud library.

In [16]:
```
def generateWordCloud(data,s):
    plt.figure(figsize = (15,15))
    wc = WordCloud(background_color = 'black', max_words =150,  max_font_size = 40,stopwords=[],scale=3)
    wc.generate(data)
    plt.imshow(wc)
    plt.title(s,y=1.03,fontdict = {'fontsize' : 30,'fontstyle':'italic'})
    plt.axis('off')
```

```
In [17]: generateWordCloud(text,'Most Frequent Words with StopWords')
```

Most Frequent Words with StopWords



Word coud without stop words in the text

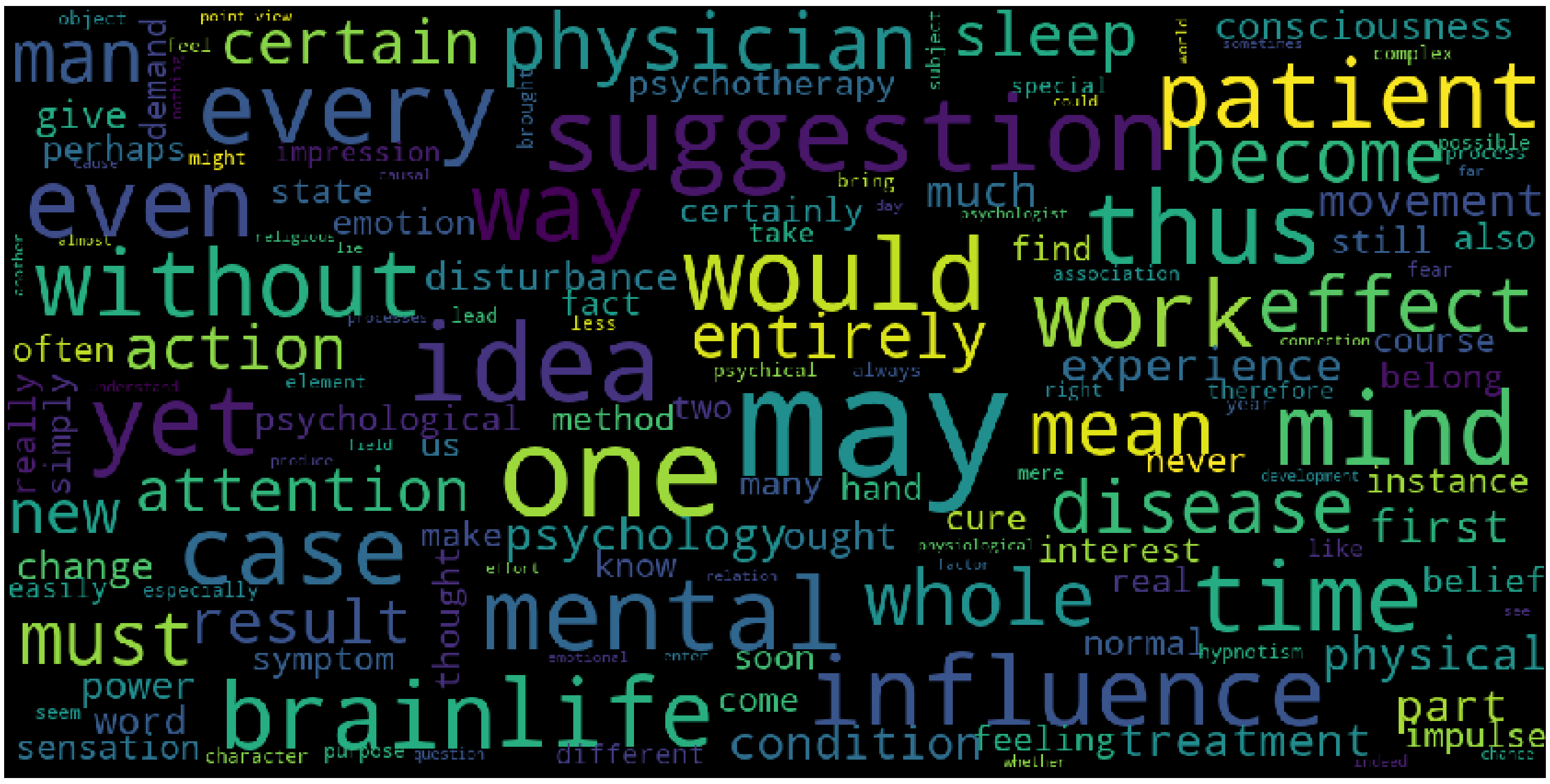
```
In [18]: words_stop_removed = []

         for w in words:
             if w not in stop_words:
                 words_stop_removed.append(w)
```

```
In [19]: text_stop_removed = " ".join(words_stop_removed)
```

```
In [20]: generateWordCloud(text_stop_removed,'Most Frequent Words without StopWords')
```

Most Frequent Words without StopWords



Difference between the two WordClouds

In the first wordcloud with stopwords as most of the text consist of stop words so as we can see in the image most of the words were the,of,is and we didn't got any information regrading the context of the text of the file that we are using but in the second case word cloud without stopwords we got an image in which words which are related to the text or are most frequently used in the text are given so we can have a idea of what the text is about like as we can see words like physician,mental,mind,brain,patient,suggestions so we can conclude that this text is regarding physcology.

5. Relation between word length and frequency

Calculating lenght of each tokenize word in the text

with the help of function we created a list lenght_word which shows length of each word present in the words list.

In [21]:
```
def lengthOfEachWord(words):
    length_word = []
    for w in words:
        length_word.append(len(w))
    return length_word
```

In [22]:
```
#store count of each tokenized word
length_word = lengthOfEachWord(words)
print("Length of first 10 words",length_word[:10])
```

```
Length of first 10 words [4, 6, 2, 13, 7, 2, 1, 6, 2, 5]
```

Calculating frequency of each length

In [23]:
```
#storing frequency of each length in length_count list
# index 0 contains count of length=1
# index 1 contains count of length=2
def countOfEachLength(length_word):
    length_count = [0]*(max(length_word))
    for i in length_word:
        length_count[i-1] += 1
    return length_count
```

In [24]:
```
length_count = countOfEachLength(length_word)

for i in range(len(length_count)):
    print('length ',i+1,' has occured ',length_count[i],' times')
```

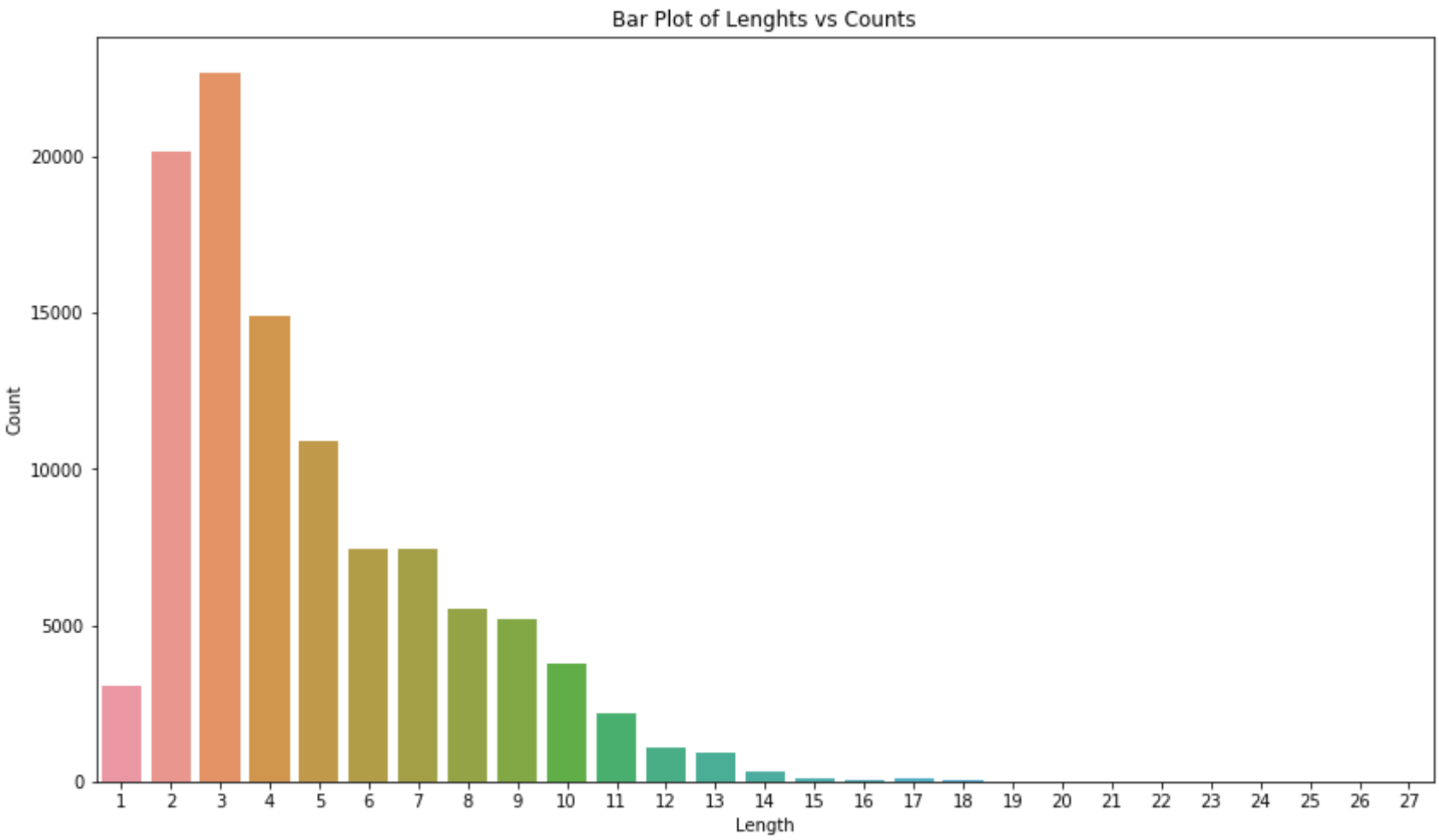
```
length  1  has occured  3043  times
length  2  has occured  20169  times
length  3  has occured  22693  times
length  4  has occured  14880  times
length  5  has occured  10912  times
length  6  has occured  7435  times
length  7  has occured  7414  times
length  8  has occured  5502  times
length  9  has occured  5187  times
length  10  has occured  3785  times
length  11  has occured  2161  times
length  12  has occured  1078  times
length  13  has occured  931  times
length  14  has occured  336  times
length  15  has occured  116  times
length  16  has occured  43  times
length  17  has occured  103  times
length  18  has occured  19  times
length  19  has occured  7  times
length  20  has occured  0  times
length  21  has occured  1  times
length  22  has occured  0  times
length  23  has occured  0  times
length  24  has occured  0  times
length  25  has occured  0  times
length  26  has occured  0  times
length  27  has occured  1  times
```

Plot of Frequecy of each length

In [25]:
```
import seaborn as sns
import pandas as pd
```

In [26]:
```
#availableLength : list which store the lengths - 1,2,3...
#created a data frame b/w the lengths and their counts in the corpus.
availableLengths = [i for i in range(1,max(length_word)+1)]
df = pd.DataFrame({'length':availableLengths,'counts': length_count})
```

In [27]:
```
plt.figure(figsize = (14,8))
x = sns.barplot(x=availableLengths,y=length_count,data=df)
x.set_title('Bar Plot of Lenghts vs Counts')
x.set(xlabel='Length', ylabel='Count')
plt.show()
```



Correlation constant of frequency and length of words

A correlation coefficient is a statistical measure of the degree to which changes to the value of one variable predict change to the value of another. In positively correlated variables, the value increases or decreases in tandem. In negatively correlated variables, the value of one increases as the value of the other decreases.

In [28]:
```
df['length'].corr(df['counts'])
```

Out[28]: -0.7432639486947982

Conclusion-

As we can easily see from the graph and the correlation constant we can come to a conclusion that frequency and length of words are inversly proportional to each.

6. PoS Tagging

It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

In [29]:
```
#text expanded is text with puctuation as they are important for PoS tagging
sentenceList = nltk.sent_tokenize(text_expanded)
tagged = []
print('No. of Sentences: ',len(sentenceList))
#print(sentenceList)
```

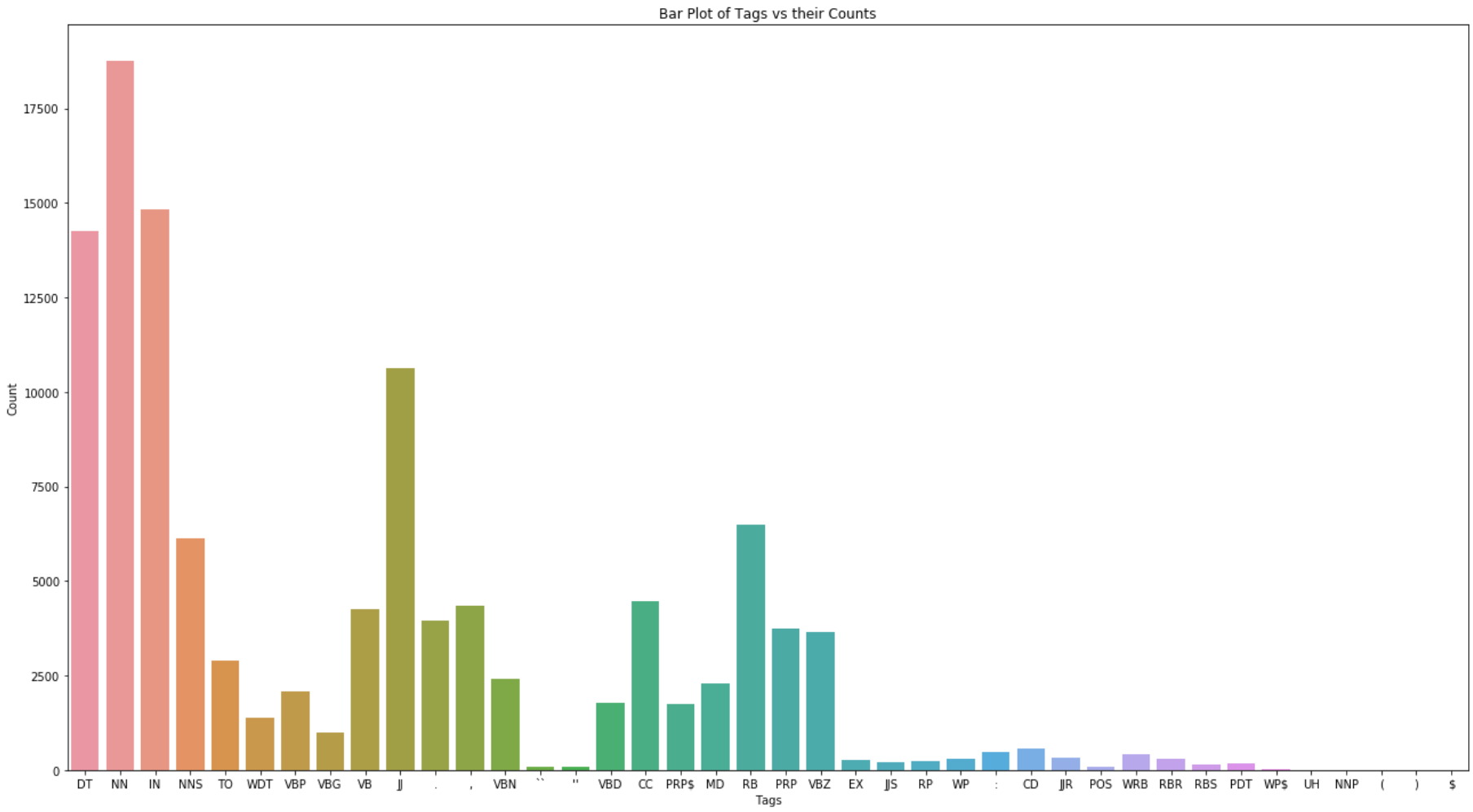
```
No. of Sentences:  3948
```

In [30]:
```
for sentence in sentenceList:
    wordList = nltk.word_tokenize(sentence)
    taggedSentence = nltk.pos_tag(wordList,lang='eng')
    tagged.append(taggedSentence)
```

In [31]:
```
countTagged = {}
c = 0
for i in tagged:
    for j in i:
        countTagged[j[1]] = countTagged.get(j[1],0) + 1
        c+=1
```

In [32]:
```
df1 = pd.DataFrame({'length':list(countTagged.keys()),'counts':list(countTagged.values())})
```

In [33]:
```
plt.figure(figsize = (22,12))
x = sns.barplot(x=list(countTagged.keys()),y=list(countTagged.values()),data=df1)
x.set_title('Bar Plot of Tags vs their Counts')
x.set(xlabel='Tags', ylabel='Count')
plt.show()
```



ROUND 2

7. Finding Noun and Verbs in the Novel

Using above PoS tagged list

To find out noun and verbs present in the novel we use our PoS tagged List tagged(created above). There are 4 and 6 different type of nouns and verbs respectively PoS tags-

Tag	Description
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present

In [52]:
```
taggedWords = []
for sentence in tagged:
  for w in sentence:
    taggedWords.append(w)
```

In [53]:
```
nouns=[]
verbs=[]
for word,pos in taggedWords:
    if (pos == 'NN' or pos == 'NNP' or pos == 'NNS' or pos == 'NNPS'):
         nouns.append(word)
    if(pos == 'VB' or pos == 'VBD' or pos == 'VBG' or pos == 'VBN' or pos == 'VBP' or pos == 'VBZ'):
        verbs.append(word)
print('No. of Nouns',len(nouns))
print("No. of Verbs",len(verbs))
```

```
No. of Nouns 24914
No. of Verbs 15183
```

So we use this Knowledge to built two Lists
- Nouns
- Verbs
which contains all the words present in the text which are noun and verb respectively.

Issue

In the above approach most of the words present in the text have more than one sense.

Like Volume which has 6 senses of noun. Hence simply taking the first sense would not give us correct results.

In [59]:
```
print("SENSES OF VOLUME")
for sense in wn.synsets('volume'):
    print()
    print(sense.lexname(),"--",sense.definition())
```

```
SENSES OF VOLUME

noun.quantity -- the amount of 3-dimensional space occupied by an object

noun.attribute -- the property of something that is great in magnitude

noun.artifact -- physical objects consisting of a number of pages bound together

noun.communication -- a publication that is one of a set of several similar publications

noun.quantity -- a relative amount

noun.attribute -- the magnitude of sound (usually in a specified direction)
```

Solution

So we should apply Word Sense Disambiguation on the text sentence(not PoS tagged sentences).
So that we can have only 1 sense for a word in the sentence.

.Also in WordNet, a word's sense is stored as volume.n.01, we can easily distinguish whether the sense it refers to is a Noun or a Verb.

Step 1 (Disambiguate Text)

(Storing disambiguated sentence in sentenceDisambiguated list)

We use pywsd which is Python implementations of Word Sense Disambiguation technologies. It includes

- **Lesk algorithms (includes original Lesk, adapted Lesk and simple Lesk)**
- **Baseline algorithms (random sense, first sense, Most Frequent Sense)**
  ** We have use it's all-words implementation (using simple-lesk) for WSD.**

In [66]:
```
from pywsd import disambiguate

def disambiguateText():
    sentenceDisambiguated = []

    for sentence in sentenceList:
        sentenceDisambiguated.append(disambiguate(sentence))

    return sentenceDisambiguated
```

In [87]:
```
sentenceDisambiguated = disambiguateText()
print(sentenceDisambiguated[0])
```

```
[('this', None), ('volume', Synset('book.n.02')), ('on', None), ('psychotherapy', Synset('psychotherapy.n.02')), ('be
longs', Synset('belong.v.03')), ('to', None), ('a', None), ('series', Synset('series.n.07')), ('of', None), ('books',
Synset('script.n.01')), ('which', None), ('i', None), ('am', None), ('writing', Synset('publish.v.03')), ('to', Non
e), ('discuss', Synset('discourse.v.01')), ('for', None), ('a', None), ('wider', Synset('across-the-board.s.01')),
('public', Synset('public.n.02')), ('the', None), ('practical', Synset('practical.s.04')), ('applications', Synset('a
pplication.n.02')), ('of', None), ('modern', Synset('modern.s.05')), ('ology', Synset('ology.n.01')), ('.', None)]
```

Step 2 (Find lexical category)

Converting disambiguated sentences into a list lexName which is a tuple consisting of a word with its WordNet lexicographer file name.

A WordNet lexicographer file name specifies for each word its class(noun,verb,adjective or adverb) and its type.

In [74]:
```
def findLexName():
    lexName = []
    for sentence in sentenceDisambiguated:
        for word in sentence:
            if type(word[1])==type(None):
                continue
            lexName.append((word[0],word[1].lexname()))
    return lexName
```

In [77]:
```
lexName = findLexName()
```

In [88]:
```
print(lexName[0])
```

```
('volume', 'noun.artifact')
```

Step 3 (Store in different list - Nouns/Verbs)

nounCatCount(dictionary) containing all the noun categories with their count in the text, and verbCatCount(dictionary) containing all the verb categories with their count in the text.

In [81]:
```
nounCatCount = {}
verbCatCount = {}

for word in lexName:
    if word[1][0]=='n':
        nounCatCount[word[1]] = nounCatCount.get(word[1],0)+1
    elif word[1][0]=='v':
        verbCatCount[word[1]] = verbCatCount.get(word[1],0)+1

print('No of Categories for Verbs: ',(len(verbCatCount)))
print('No of Categories for Nouns: ',len((nounCatCount)))
```

```
No of Categories for Verbs:  15
No of Categories for Nouns:  26
```

In [86]:
```
for cat in verbCatCount.keys():
    print(cat,' ',verbCatCount[cat])
```

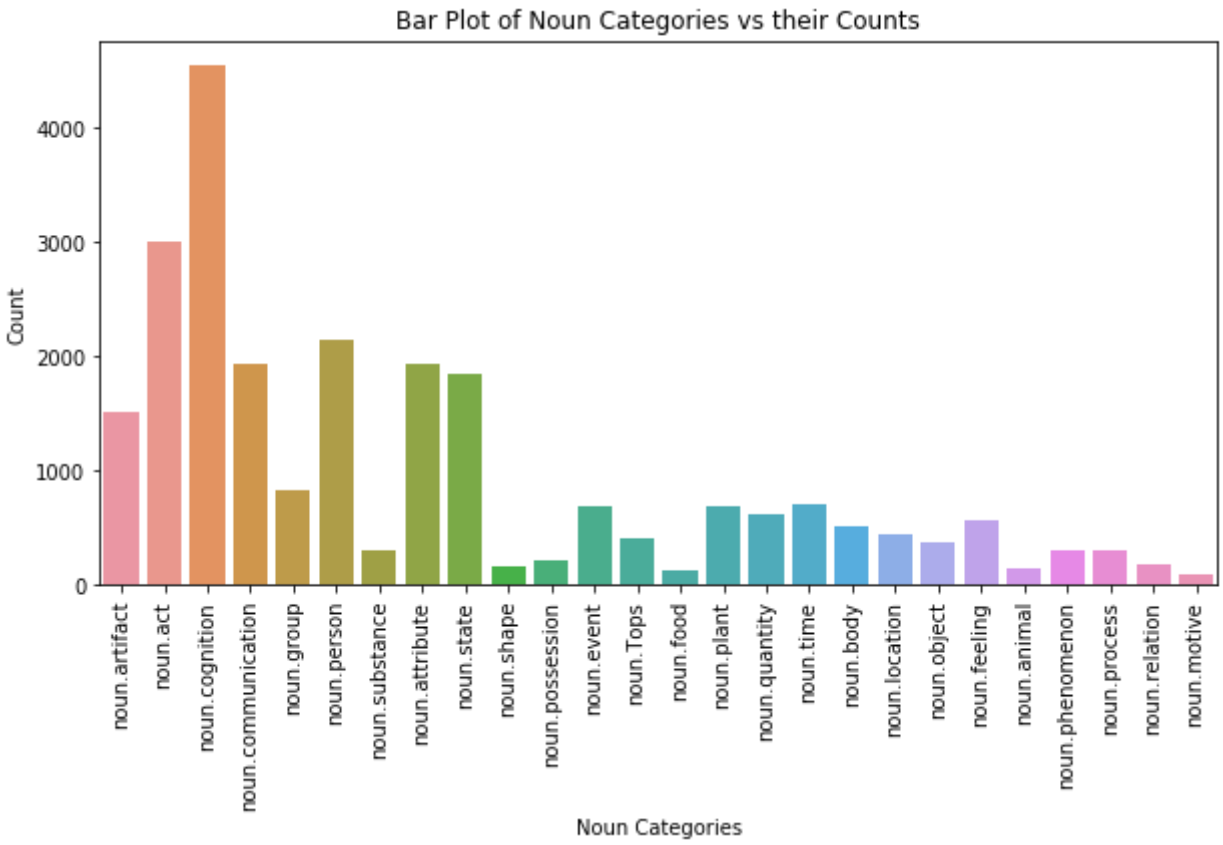
```
verb.stative   1394
verb.creation   450
verb.communication   1269
verb.social   796
verb.cognition   1253
verb.body   356
verb.contact   618
verb.consumption   252
verb.change   1315
verb.motion   585
verb.possession   546
verb.perception   575
verb.emotion   230
verb.competition   186
verb.weather   9
```

We see that there are 26 categories for nouns. There is one extra category i.e noun.Tops which is unique beginner for nouns.

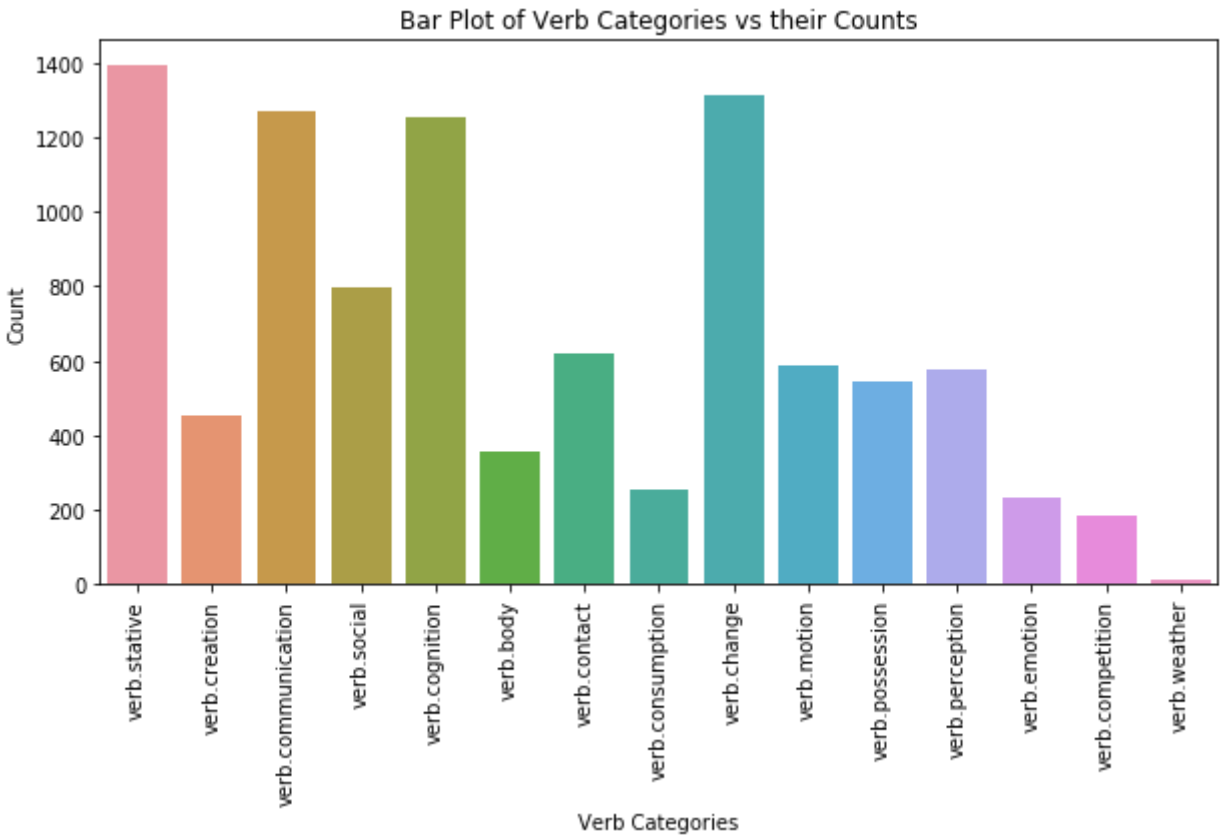
Step 4 (Plotting)

In [129]:
```
def barPlot(dictt,st):
    df1 = pd.DataFrame({'length':list(dictt.keys()),'counts':list(dictt.values())})
    plt.figure(figsize = (10,5))
    x = sns.barplot(x=list(dictt.keys()),y=list(dictt.values()),data=df1)
    x.set_title('Bar Plot of '+st+ ' vs their Counts')
    plt.xticks(rotation=90)
    x.set(xlabel=st, ylabel='Count')
    plt.show()
```

In [130]:
```
barPlot(nounCatCount,'Noun Categories')
```



In [131]:
```
barPlot(verbCatCount,'Verb Categories')
```



8. Named Entity Recognition

Named-entity recognition (NER) is process or subtask of information extraction that seeks to locate and classify named entity mention in the unstructured text into pre-defined categories such as people, organisation, location, geo-political entity, facility and vehicles.

spaCy is regarded as the fastest NLP framework in Python, with single optimized functions for each of the NLP tasks it implements. Being easy to learn and use, one can easily perform simple tasks using a few lines of code. spaCy's models can be installed as Python packages.

We have used ScaCy for our NER task.

SpaCy's named entity recognition has been trained on the OntoNotes 5.
It uses BILUO annotation instead of BIO scheme.

- **B: The first token of a multi-token entity.**
- **I: An inner token of a multi-token entity.**
- **L: The final token of a multi-token entity.**
- **U: A single-token entity.**
- **O: A non-entity token.**

In [92]:
```
import spacy
```

About Model used for NER

Downloaded the spacy NER model package through terminal using python -m spacy download en_core_web_sm command.

Here model is loaded in *ner_model* variable.

The model used is en_core_web_sm

1. **Type: Model capabilities (e.g. core for general-purpose model with vocabulary, syntax, entities and word vectors, or depent for only vocab, syntax and entities).**
2. **Genre: Type of text the model is trained on, e.g. web or news.**
3. **Size: Model size indicator, sm, md or lg. **
   Hence en_core_web_sm is a small English model trained on written web text (blogs, news, comments), that includes vocabulary, vectors, syntax and entities.

   (Here model is loaded in *ner_model* variable.)

In [103]:
```
from collections import Counter
ner_model = spacy.load(r'C:\Users\mahav\Anaconda3\Lib\site-packages\en_core_web_sm\en_core_web_sm-2.2.5')
```

Step 1. Applying the Model

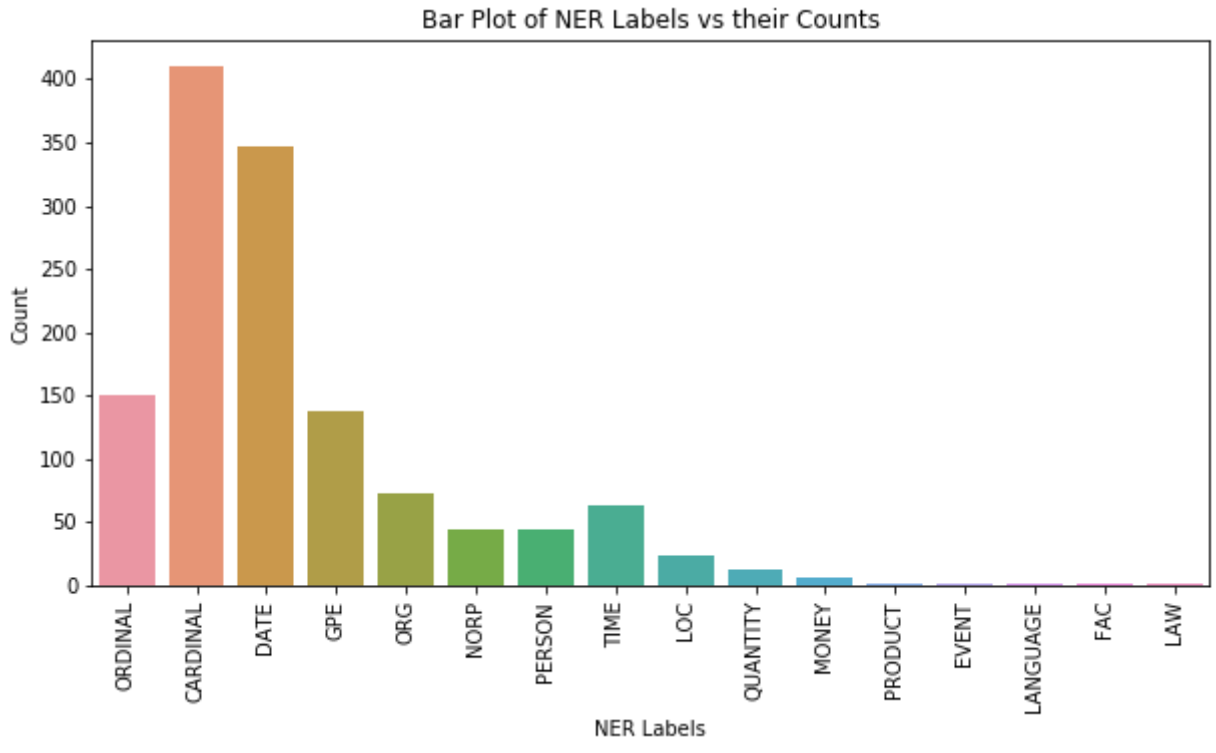
Models trained on the OntoNotes 5 corpus support the following entity types:

1. **PERSON : People, including fictional.**
2. **NORP : Nationalities or religious or political groups.**
3. **FAC : Buildings, airports, highways, bridges, etc.**
4. **ORG : Companies, agencies, institutions, etc.**
5. **GPE : Countries, cities, states.**
6. **LOC : Non-GPE locations, mountain ranges, bodies of water.**
7. **PRODUCT : Objects, vehicles, foods, etc. (Not services.)**
8. **EVENT : Named hurricanes, battles, wars, sports events, etc.**
9. **WORK_OF_ART : Titles of books, songs, etc.**
10. **LAW : Named documents made into laws.**
11. **LANGUAGE : Any named language.**
12. **DATE : Absolute or relative dates or periods.**
13. **TIME : Times smaller than a day.**
14. **PERCENT : Percentage, including ”%“.**
15. **MONEY : Monetary values, including unit.**
16. **QUANTITY : Measurements, as of weight or distance.**
17. **ORDINAL : “first”, “second”, etc.**
18. **CARDINAL : Numerals that do not fall under another type.**

NER on each sentence of the corpus.

In [144]:
```
labels = []
labCount = {}
for sent in sentenceList:
    doc = ner_model(sent)
    for entity in doc.ents:
        labels.append((entity.text,entity.label_))
```

In [145]:
```
for text,label in labels:
    labCount[label] = labCount.get(label,0)+1
barPlot(labCount,"NER Labels")
```



We see that most of labels assigned are CARDINAL , DATE

In [146]:
```
print(labels[:10])
```

```
[('first', 'ORDINAL'), ('a hundred', 'CARDINAL'), ('first', 'ORDINAL'), ('first', 'ORDINAL'), ('first', 'ORDINAL'),
('second', 'ORDINAL'), ('third', 'ORDINAL'), ('one', 'CARDINAL'), ('recent years', 'DATE'), ('my student days', 'DAT
E')]
```

Step 2. Testing

For testing we have hand labeled 3 paragraphs from the text. Below are the paragraphs which will be handlabeled.

In [163]:
```
para1 = """One word for my personal right to deal with these questions, as too much illegitimate psychotherapeutics i
s heard to-day. For me, the relation between psychology and medicine is not a chance chapter of my science to which I
have turned simply in following up the various sides of applied psychology. And still less have I turned to it becaus
e it has become the fashion in recent years. On the contrary, it has been an important factor in all my work since my
student days. I have been through five years of regular medical studies, three years in Leipzig and two years in Heid
elberg; I have an M.D. degree from the University of Heidelberg."""
```

In [164]:
```
para2 = """This letter was written on the twenty-third of January, 1908. I replied to him at once that he certainly o
ught not to come from the Pacific to the Atlantic, but that I wanted him to write to me much more about that first oc
currence. As he was evidently right in considering that episode as the starting point of his troublesome association
s, I supposed that these associated ideas had not yet become independent but were still the effect of that first "com
plex." Therefore I wanted to bring that to complete discharge. Accordingly I wrote him to think himself once more int
o that happening of years ago, to pass through it with all the power of his imagination, to describe it to me then in
as full a statement as possible and to express in the letter also his conviction that there was no reason to avoid th
e eyes of his superior, that he might have looked straight into his face. As soon as he got my reply, he wrote to me
 on the sixth of February a description of that first episode, filling nineteen pages, telling me all about his relat
ions to those various men and every minute detail was brought clearly to consciousness again. I did not add anything
 further, but the expected occurred."""
```

In [165]:
```
para3 = """On the eighteenth of February, he writes to me: "In the last week or ten days, the writer has noted a deci
ded improvement regarding mental condition. The result is a new interest in life. If you can spare the time, would li
ke to have you write me a few lines. Gratefully yours." At the end of the month he writes: "Received your letter abou
t half an hour ago. Hasten to assure you with a great deal of pleasure that I am feeling much better. Since sending y
ou the letter regarding the first case, I have noticed day by day an improvement." On the eighth of March: "Since wri
ting you last I have noticed a gradual improvement. It has given me wonderful encouragement." On the tenth of March:
 "Just a line to say that I am still improving." On the twelfth of April: "I desire to say that since the taking up o
f treatment with you, life has had a far different appearance to me than it has had for the last ten years." On the t
wenty-first of April: "Since my first letter to you, there has been such an improvement that I have accepted a positi
on which carries with it much responsibility."""
```

Handlabling Named Entites

In [281]:
```
handlabeled1 = [
    ('One word for my personal right to deal with these questions, as too much illegitimate psychotherapeutics is hea
rd to-day.',
     [[0, 3, 'CARDINAL']]),

    ('For me, the relation between psychology and medicine is not a chance chapter of my science to which I have turn
ed simply in following up the various sides of applied psychology.',
     []),

    ('And still less have I turned to it because it has become the fashion in recent years.',
     [(72,84,"DATE")]),

     ('On the contrary, it has been an important factor in all my work since my student days.',
      []),

     ('I have been through five years of regular medical studies, three years in Leipzig and two years in Heidelberg;
I have an M.D. degree from the University of Heidelberg.',
      [[20,30,"DATE"],[59,70,"DATE"],[74,81,'GPE'],[86,95,'DATE'],[99,109,'GPE'],[142,166,'ORG']])
    ]


handlabeled2 = [
    ('This letter was written on the twenty-third of January, 1908.',
     [[27, 60, 'DATE']]),

    ('I replied to him at once that he certainly ought not to come from the Pacific to the Atlantic, but that I wante
d him to write to me much more about that first occurrence.',
     [[70,77,'LOC'],[85,93,'LOC'],[153,158,'ORDINAL']]),


    ('As he was evidently right in considering that episode as the starting point of his troublesome associations, I
 supposed that these associated ideas had not yet become independent but were still the effect of that first "comple
x".',
     [[213,218,"ORDINAL"]]),

     ('Therefore I wanted to bring that to complete discharge.',
      []),

     ('Accordingly I wrote him to think himself once more into that happening of years ago, to pass through it with a
ll the power of his imagination, to describe it to me then in as full a statement as possible and to express in the l
etter also his conviction that there was no reason to avoid the eyes of his superior, that he might have looked strai
ght into his face.',
      [[74,83,"DATE"]]),

     ('As soon as he got my reply, he wrote to me on the sixth of February a description of that first episode, filli
ng nineteen pages, telling me all about his relations to those various men and every minute detail was brought clearl
y to consciousness again.',
       [[50,67,'DATE'],[90,95,'ORDINAL'],[113,121,'ORDINAL']]),
    ]
```

Comparing Named entity recognised by spaCy model with Handlabled Paragraph using Scorer

We have used 2 inbuilt spaCy libraries for testing.

- **Scorer: Class in SpaCy which stores evaluation scores.**
- **GoldParse : A collection for training annotations.**

In [ ]:
```
from spacy.gold import GoldParse
from spacy.scorer import Scorer
from tabulate import tabulate
```

(Importing libraries)

In [291]:
```
def print_results(results):
    print('F-Score : ',round(results['ents_f'],2))
    print('Precsion : ',round(results['ents_p'],2))
    print('Recall : ',round(results['ents_r'],2),'\n')

    print('EVALUATING PER ENTITY TYPE\n')
    data = []
    for key in results['ents_per_type'].keys():
        data.append([key,round(results['ents_per_type'][key]['f'],2),round(results['ents_per_type'][key]['f'],2),roun
d(results['ents_per_type'][key]['f'],2)])

    print(tabulate(data, headers=['Entity Type', 'Precsion','Recall','F1-Score']))
```

In [292]:
```
def evaluate(ner_model, examples):
    scorer = Scorer()

    for ex in range(len(examples)):
        doc_gold_text = ner_model.make_doc(examples[ex][0])
        gold = GoldParse(doc_gold_text, entities=examples[ex][1])
        pred_value = ner_model(examples[ex][0])
        scorer.score(pred_value, gold)
    return scorer.scores
```

In [293]:
```
def evaluate_hand_labeled_para(handlabled):
    results = evaluate(ner_model, handlabled)
    print_results(results)
```

evaluate function

1. **scorer : Stores all the scores of sentences.**
2. **Iterate through all sentences in handlabled text.**
3. **Make gold corpus of hand labled paragraph.**
4. **Apply spaCy model on the sentences.**
5. **Compare(score) predicted and the gold standard.**

printing results(scores)

Inside scorer we have various attributes which we can evaluate. We have listed the following in our results :

1. **Total F1 Score =** $\frac{2*Precision*Recall}{Precision+Recall}$
2. **Total Prescion =** $\frac{TruePositive}{TruePositive+TrueNegative}$
3. **Total Recall =** $\frac{TruePositive}{TruePositive+FlaseNegative}$
4. **Per Entity**
   - **A. Precsion**
   - **B. Recall**
   - **C. F1 Score**

In [295]:
```
evaluate_hand_labeled_para(handlabeled1)
```

```
F-Score :  77.78
Precsion :  70.0
Recall :  87.5

EVALUATING PER ENTITY TYPE

Entity Type      Precsion    Recall    F1-Score
-------------  ----------  --------  ----------
CARDINAL           100       100         100
DATE                88.89     88.89       88.89
GPE                 80        80          80
ORG                  0         0           0
```

In [296]:
```
evaluate_hand_labeled_para(handlabeled2)
```

```
F-Score :  73.68
Precsion :  70.0
Recall :  77.78

EVALUATING PER ENTITY TYPE

Entity Type      Precsion    Recall    F1-Score
-------------  ----------  --------  ----------
DATE                33.33     33.33       33.33
LOC                100       100         100
ORDINAL             88.89     88.89       88.89
```

9. Extracting Relationship between Entities

Relationship extraction is the task of extracting semantic relationships from a text. Extracted relationships usually occur between two or more entities of a certain type (e.g. Person, Organisation, Location) and fall into a number of semantic categories (e.g. married to, employed by, lives in).

For our task we use the following text from the book.

Text 1

In [368]:
```
TEXTS = [
    """One word for my personal right to deal with these questions, as too much illegitimate psychotherapeutics is he
ard to-day. For me, the relation between psychology and medicine is not a chance chapter of my science to which I hav
e turned simply in following up the various sides of applied psychology. And still less have I turned to it because i
t has become the fashion in recent years. On the contrary, it has been an important factor in all my work since my st
udent days. I have been through five years of regular medical studies, three years in Leipzig and two years in Heidel
berg; I have an M.D. degree from the University of Heidelberg. In my first year as docent in a German university twen
ty years ago, I gave throughout the winter semester before several hundred students a course in hypnotism and its med
ical application. It was probably the first university course on hypnotism given anywhere. Since that time I have nev
er ceased to work psychotherapeutically in the psychological laboratory. Yet that must not be misunderstood. I have n
o clinic, and while by principle I have never hypnotized anyone for mere experiment's sake but always only for medica
l purposes, yet I adjust my practical work entirely to the interests of my scientific study. The limitations of my ti
me force me to refuse the psychotherapeutic treatment of any case which has not a certain scientific interest for me,
and of the many hundreds whom I have helped in the laboratory, no one ever had to pay anything. Thus my practical wor
k has strictly the character of laboratory research."""
]
```

Below are the named entities extracted using the model.

In [369]:
```
def ner_text():
    doc = ner_model(TEXTS[0])
    for entity in doc.ents:
        print(entity.label_,'  ',entity.text)
ner_text()
```

```
CARDINAL     One
DATE     recent years
DATE     my student days
DATE     five years
DATE     three years
GPE     Leipzig
DATE     two years
GPE     Heidelberg
GPE     M.D.
ORG     the University of Heidelberg
DATE     my first year
NORP     German
DATE     twenty years ago
DATE     winter
CARDINAL     several hundred
ORDINAL     first
CARDINAL     the many hundreds
```

Lets see the 2 GPE entities : Leipzig and Heidelberg.

Since in our book the named entity PERSON are very less and very sparse, we extract relationships b/w GPE

In [370]:
```
def filter_spans(spans):
    get_sort_key = lambda span: (span.end - span.start, -span.start)
    sorted_spans = sorted(spans, key=get_sort_key, reverse=True)
    result = []
    seen_tokens = set()
    for span in sorted_spans:
        if span.start not in seen_tokens and span.end - 1 not in seen_tokens:
            result.append(span)
            seen_tokens.update(range(span.start, span.end))
    result = sorted(result, key=lambda span: span.start)
    return result
```

In [371]:
```
def main(ner_model):
    nlp = ner_model
    print("Processing %d texts" % len(TEXTS))

    for text in TEXTS:
        doc = nlp(text)
        relations = extract_gpe_relations(doc)
        for r1, r2 in relations:
            print("{:<10}\t{}\t{}".format(r1.text, r2.ent_type_, r2.text))
```

We extract relations between phrases and entities using spaCy's named entity recognizer and the dependency parse.

In [372]:
```
def extract_gpe_relations(doc):
    spans = list(doc.ents) + list(doc.noun_chunks)
    spans = filter_spans(spans)
    with doc.retokenize() as retokenizer:
        for span in spans:
            retokenizer.merge(span)

    relations = []
    for gp in filter(lambda w: w.ent_type_ == "GPE", doc):
        if gp.dep_ in ("attr", "dobj"):
            subject = [w for w in money.head.lefts if w.dep_ == "nsubj"]
            if subject:
                subject = subject[0]
                relations.append((subject, gp))
        elif gp.dep_ == "pobj" and gp.head.dep_ == "prep":
            relations.append((gp.head.head, gp))
    return relations
```

In [373]:
```
main(ner_model)
```

```
Processing 1 texts
three years     GPE     Leipzig
two years       GPE     Heidelberg
```

- **Leipzig relates to the DATE entity three years**
- **Heidelberg related to the DATE entity two years**

I have been through five years of regular medical studies, three years in Leipzig and two years in Heidelberg;

So our extraction were indeeed correct!.

Text 2

In [374]:
```
TEXTS = [
    """This letter was written on the twenty-third of January, 1908. I replied to him at once that he certainly ought not to come from the Pacific to the Atlantic, but that I wanted him to write to me much more about that first occurrence. As he was evidently right in considering that episode as the starting point of his troublesome associations, I supposed that these associated ideas had not yet become independent but were still the effect of that first "complex." Therefore I wanted to bring that to complete discharge. Accordingly I wrote him to think himself once more into that happening of years ago, to pass through it with all the power of his imagination, to describe it to me then in as full a statement as possible and to express in the letter also his conviction that there was no reason to avoid the eyes of his superior, that he might have looked straight into his face. As soon as he got my reply, he wrote to me on the sixth of February a description of that first episode, filling nineteen pages, telling me all about his relations to those various men and every minute detail was brought clearly to consciousness again. I did not add anything further, but the expected occurred.
    """]
```

In [375]:
```
ner_text()
```

```
DATE    the twenty-third of January, 1908
LOC     Pacific
LOC     Atlantic
ORDINAL     first
ORDINAL     first
DATE    years ago
ORDINAL     sixth
DATE    February
ORDINAL     first
ORDINAL     nineteen
```

Lets see the 2 LOC entities : Pacific and Atlantic.

Here too named entity PERSON are very less and very sparse, we extract relationships b/w LOC

In [376]:
```
def main(ner_model):
    nlp = ner_model
    print("Processing %d texts" % len(TEXTS))

    for text in TEXTS:
        doc = nlp(text)
        relations = extract_loc_relations(doc)
        for r1, r2 in relations:
            print("{:<10}\t{}\t{}".format(r1.text, r2.ent_type_, r2.text))
```

In [377]:
```
def extract_loc_relations(doc):
    spans = list(doc.ents) + list(doc.noun_chunks)
    spans = filter_spans(spans)
    with doc.retokenize() as retokenizer:
        for span in spans:
            retokenizer.merge(span)

    relations = []
    for loc in filter(lambda w: w.ent_type_ == "LOC", doc):
        if loc.dep_ in ("attr", "dobj"):
            subject = [w for w in loc.head.lefts if w.dep_ == "nsubj"]
            if subject:
                subject = subject[0]
                relations.append((subject, loc))
        elif loc.dep_ == "pobj" and loc.head.dep_ == "prep":
            relations.append((loc.head.head, loc))
    return relations
```

In [378]:
```
main(ner_model)
```

```
Processing 1 texts
come            LOC     the Pacific
come            LOC     the Atlantic
```

- **the Pacific relates to come phrase.**
- **the Atlantic also relates to the same come phrase.**

""I replied to him at once that he certainly ought not to come from the Pacific to the Atlantic, but that I wanted him to write to me much more about that first occurrence.""

But the Atlantic should refer to from the Pacific

Text 3

In [395]:
```
TEXTS = ["""As soon as the psychophysical principles are understood, there is indeed no difficulty in going from the
 simplest experience to those spectacular ones where we may suggest to the profoundly hypnotized person that he is a
 little child or that he is George Washington. In the one case, he will speak and cry and play and write as in his pr
esent imagination a child would behave; in the other case, he will pose in an attitude which he may have seen in a pi
cture of Washington. There is nothing mysterious and his utterances are completely dependent upon his own ideas, whic
h may be very different from the real wisdom of a Washington and the real unwisdom of a child. I may suggest to him t
o be the Czar, by that he will not become able to speak Russian. In the same way I may suggest changes of the surroun
dings; he may take my room for the river upon which he paddles his canoe, or for the orchard in which he picks apples
from my bookshelves."""]
```

In [396]:
```
ner_text()
```

```
PERSON    George Washington
CARDINAL    one
GPE    Washington
GPE    Washington
ORG    Czar
NORP    Russian
```

In [397]:
```
def main(ner_model):
    nlp = ner_model
    print("Processing %d texts" % len(TEXTS))

    for text in TEXTS:
        doc = nlp(text)
        relations = extract_per_relations(doc)
        for r1, r2 in relations:
            print("{:<10}\t{}\t{}".format(r1.text, r2.ent_type_, r2.text))
```

In [404]:
```
def extract_per_relations(doc):
    spans = list(doc.ents) + list(doc.noun_chunks)
    spans = filter_spans(spans)
    with doc.retokenize() as retokenizer:
        for span in spans:
            retokenizer.merge(span)

    relations = []
    for per in filter(lambda w: w.ent_type_ == "PERSON", doc):
        if per.dep_ in ("attr", "dobj"):
            subject = [w for w in per.head.lefts if w.dep_ == "nsubj"]
            if subject:
                subject = subject[0]
                relations.append((subject, per))
        elif per.dep_ == "pobj" and per.head.dep_ == "prep":
            relations.append((per.head.head, per))
    return relations
```

In [405]:
```
main(ner_model)
```

```
Processing 1 texts
he              PERSON  George Washington
```

there is indeed no difficulty in going from the simplest experience to those spectacular ones where we may suggest to the profoundly hypnotized person that he is a little child or that he is George Washington.

Extraction was correct!

ROUND 1 - BOOK 02

1. Importing and Reading The Book 02

Title- The Life, Crime and Capture of John Wilkes Booth
Author- George Alfred Townsend
Words- 42,077
Characters (no spaces)- 198,917
Characters (with spaces)- 237,296
Paragraphs- 3,699
Lines- 4,261

In [406]:
```
import nltk
import matplotlib
```

In [408]:
```
file = open("The Life, Crime and Capture of John Wilkes Booth.txt","r",encoding="utf-8-sig")
```

In [409]:
```
text = file.read()
```

2. Text-Preprocessing

All the lines which are not required in this model are removed manually. Removed lines includes part before Explanatory, all the chapters name and all the illustrations and all the heading like prefatory, explanatory and the last part of the book which contains end of the project part

In [410]:
```
postString = text.split("\n")
text  = " ".join(postString)
```

In [411]:
```
text = text.lower()
```

To remove any HTML tags from the downloaded text file we use following regular expression

It is useful beacuse html tags are just for the representation of the file and they do not contribute anything in this model.

In [412]:
```
import re
text = re.sub('<[^<]+?>','',text)
```

Converting Unicodes(ä, ś, đ, è, â, ā, etc.) into corresponding alphabet

Because we are using a PoS tagger which uses english dictionary

In [413]:
```
import unicodedata
text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
```

Converting Contraction words into their original form

For e.g.: converting "what's" --> "what is"
It is helpful because only removing punctuations marks first will not help us.
eg. can't --> cant(if we remove punctuation marks without expanding words,doesn't have any meaning in english dictionary)
can't --> cannot(if we expand word and it has meaning in english dictionary)

In [414]:
```
from contractions import contractions_dict


for word in contractions_dict.keys():
    if word in text:
        text = text.replace(word, contractions_dict[word])

text_expanded = text
```

Removing punctuation marks from the text

e.g. ",","?","!",etc.

In [415]:
```
import string
table = str.maketrans('','',string.punctuation)
text = text.translate(table)
```

3. Tokenizing the input text

Tokenization is the process by which big quantity of text is divided into smaller parts called tokens.
These tokens are very useful for finding such patterns as well as is considered as a base step for stemming and lemmatization.
We use the method word_tokenize() to split a string (passed as argument) into list of words.

In [416]:
```
words = nltk.word_tokenize(text)
```

NLTK in python has a function FreqDist which gives you the frequency of each unique word within a text

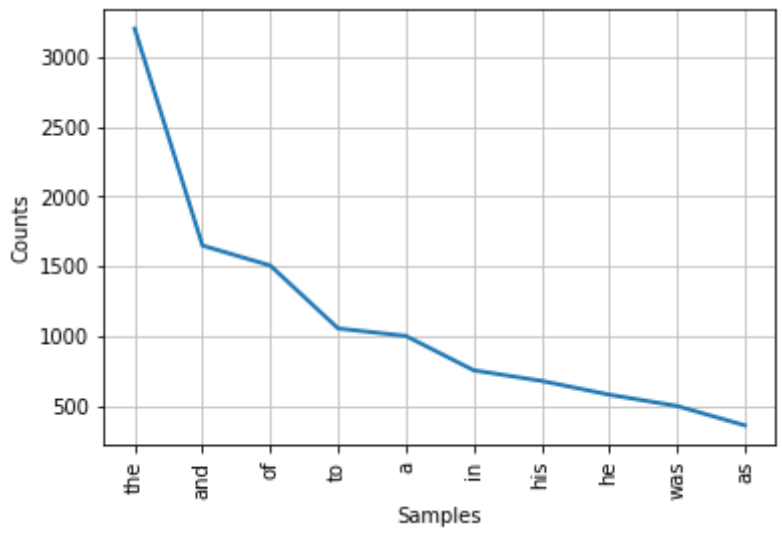
Frequency distributions are generally constructed by running a number of experiments, and incrementing the count for a sample every time it is an outcome of an experiment.

In [417]:
```
from nltk.probability import FreqDist
fdist = FreqDist(words)
```

Graph of 10 most appering words in the text

plot function in nltk.probability.FreqDist creates a frequency distribution plot of the most common words. Takes a no as parameter specifying no of most common words to show in plot.

In [418]:
```
fdist.plot(10)
```



Out[418]: <matplotlib.axes._subplots.AxesSubplot at 0x1d3cdee5f48>

In [419]:
```
for key,val in list(fdist.most_common())[:10]:
    print (str(key) + ':' + str(val))
```

```
the:3202
and:1647
of:1504
to:1053
a:999
in:753
his:677
he:578
was:497
as:358
```

Percentage of stop words in the original text

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. In this model we are using a predefined set of stopwords present in nltk library.

In [420]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
stopwords_text=[]
stopwords_text = [w for w in words if w in stop_words]
print("Precentage of stopwords -",len(stopwords_text)*100/len(words),"%")
```

```
Precentage of stopwords - 48.90321659687857 %
```

4. Word Cloud

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

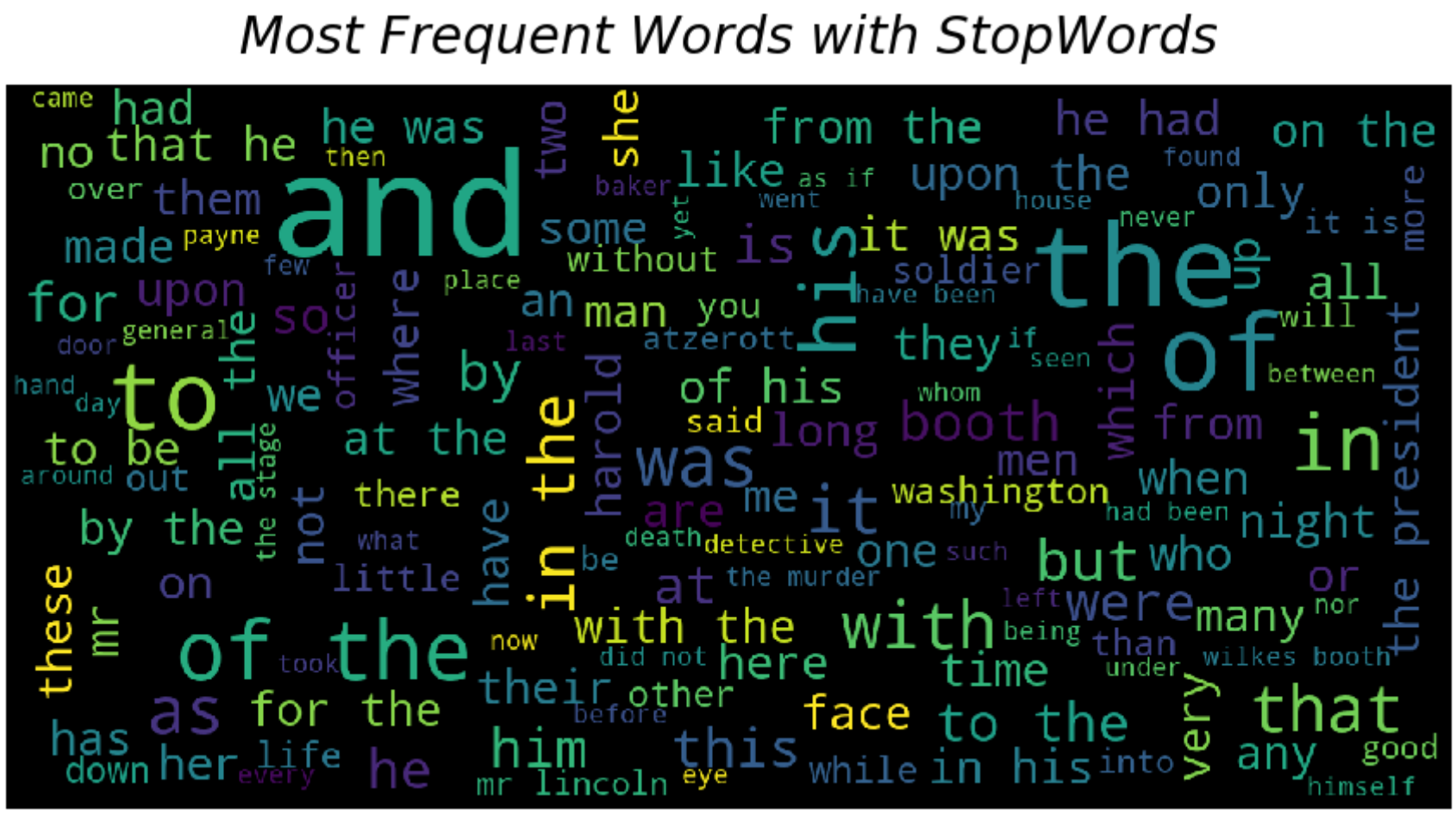
Word Cloud with stopwords in the text

In [421]:
```
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image
```

Created a function named generateWordCloud which takes two arguments first is the pre processed text and second is the title of the image. Function uses predefined function WordCloud in wordcloud library.

In [422]:
```
def generateWordCloud(data,s):
    plt.figure(figsize = (15,15))
    wc = WordCloud(background_color = 'black', max_words =150,  max_font_size = 40,stopwords=[],scale=3)
    wc.generate(data)
    plt.imshow(wc)
    plt.title(s,y=1.03,fontdict = {'fontsize' : 30,'fontstyle':'italic'})
    plt.axis('off')
```

In [423]:
```
generateWordCloud(text,'Most Frequent Words with StopWords')
```



Word coud without stop words in the text

In [424]:
```
words_stop_removed = []

for w in words:
    if w not in stop_words:
        words_stop_removed.append(w)
```

In [425]:
```
text_stop_removed = " ".join(words_stop_removed)
```

In [426]:
```
generateWordCloud(text_stop_removed,'Most Frequent Words without StopWords')
```

Difference between the two WordClouds

In the first wordcloud with stopwords as most of the text consist of stop words so as we can see in the image most of the words were the,of,is and we didn't got any information regrading the context of the text of the file that we are using but in the second case word cloud without stopwords we got an image in which words which are related to the text or are most frequently used in the text are given so we can have a idea of what the text is about like as we can see words like man, booth, murder, blood, time, office, house so we can conclude that this text is regarding a person's life.

5. Relation between word length and frequency

Calculating lenght of each tokenize word in the text

with the help of function we created a list lenght_word which shows length of each word present in the words list.

In [427]:
```
def lengthOfEachWord(words):
    length_word = []
    for w in words:
        length_word.append(len(w))
    return length_word
```

In [428]:
```
#store count of each tokenized word
length_word = lengthOfEachWord(words)
print("Length of first 10 words",length_word[:10])
```

```
Length of first 10 words [3, 4, 3, 3, 6, 2, 3, 7, 5, 6]
```

Calculating frequency of each length

In [429]:
```
#storing frequency of each length in length_count list
# index 0 contains count of length=1
# index 1 contains count of length=2
def countOfEachLength(length_word):
    length_count = [0]*(max(length_word))
    for i in length_word:
        length_count[i-1] += 1
    return length_count
```

In [430]:
```
length_count = countOfEachLength(length_word)

for i in range(len(length_count)):
    print('length ',i+1,' has occured ',length_count[i],' times')
```

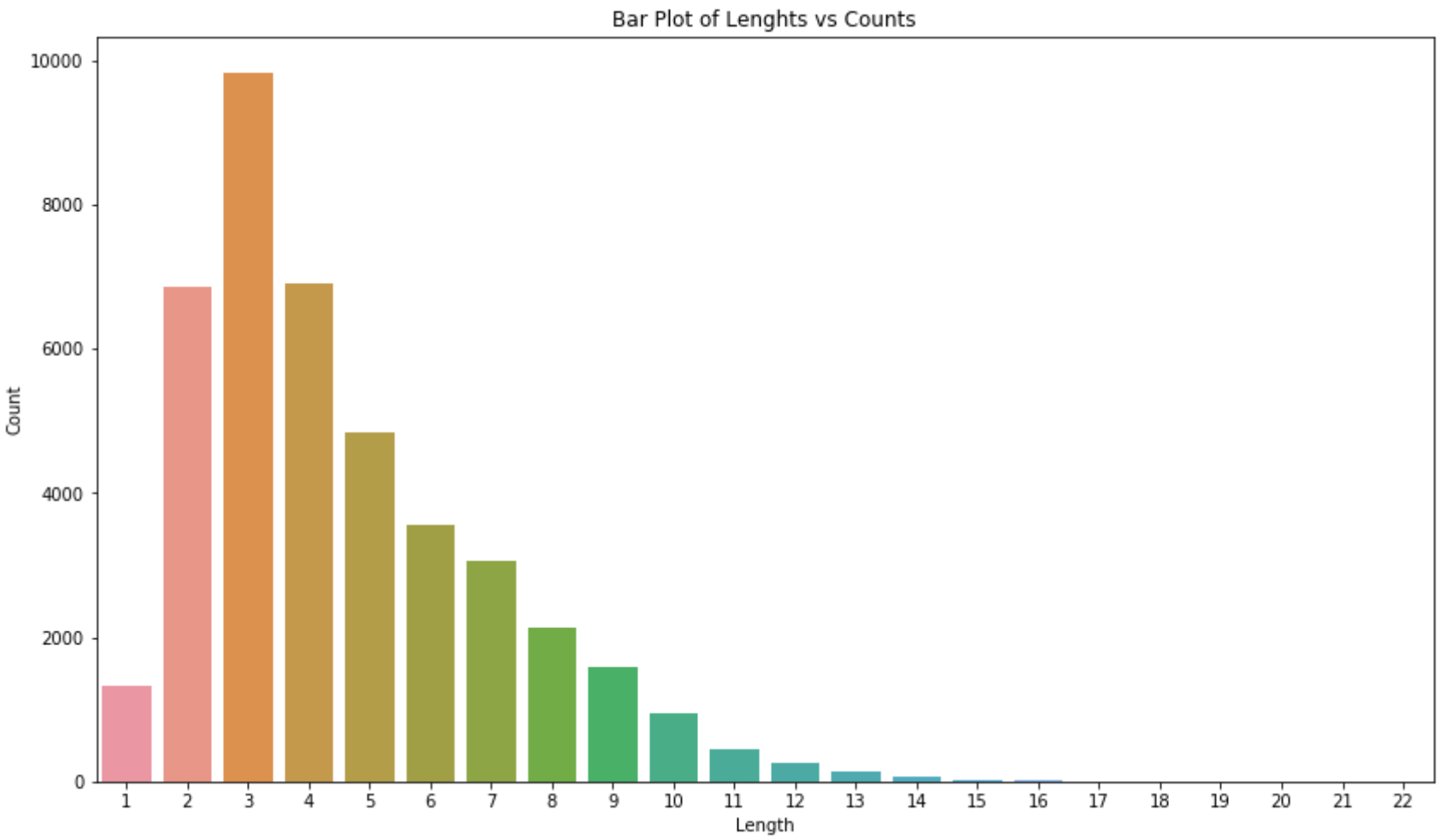
```
length  1  has occured  1334  times
length  2  has occured  6862  times
length  3  has occured  9838  times
length  4  has occured  6918  times
length  5  has occured  4834  times
length  6  has occured  3547  times
length  7  has occured  3071  times
length  8  has occured  2138  times
length  9  has occured  1594  times
length  10  has occured  954  times
length  11  has occured  443  times
length  12  has occured  250  times
length  13  has occured  139  times
length  14  has occured  66  times
length  15  has occured  27  times
length  16  has occured  8  times
length  17  has occured  6  times
length  18  has occured  1  times
length  19  has occured  0  times
length  20  has occured  0  times
length  21  has occured  1  times
length  22  has occured  1  times
```

Plot of Frequecy of each length

In [431]:
```
import seaborn as sns
import pandas as pd
```

In [432]:
```
#availableLength : list which store the lengths - 1,2,3...
#created a data frame b/w the lengths and their counts in the corpus.
availableLengths = [i for i in range(1,max(length_word)+1)]
df = pd.DataFrame({'length':availableLengths,'counts': length_count})
```

In [433]:
```
plt.figure(figsize = (14,8))
x = sns.barplot(x=availableLengths,y=length_count,data=df)
x.set_title('Bar Plot of Lenghts vs Counts')
x.set(xlabel='Length', ylabel='Count')
plt.show()
```



Correlation constant of frequency and length of words

A correlation coefficient is a statistical measure of the degree to which changes to the value of one variable predict change to the value of another. In positively correlated variables, the value increases or decreases in tandem. In negatively correlated variables, the value of one increases as the value of the other decreases.

In [434]:
```
df['length'].corr(df['counts'])
```

Out[434]:
```
-0.7592747862765812
```

Conclusion-

As we can easily see from the graph and the correlation constant we can come to a conclusion that frequency and length of words are inversly proportional to each.

6. PoS Tagging

It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

In [435]:
```
#text expanded is text with puctuation as they are important for PoS tagging
sentenceList = nltk.sent_tokenize(text_expanded)
tagged = []
print('No. of Sentences: ',len(sentenceList))
#print(sentenceList)
```

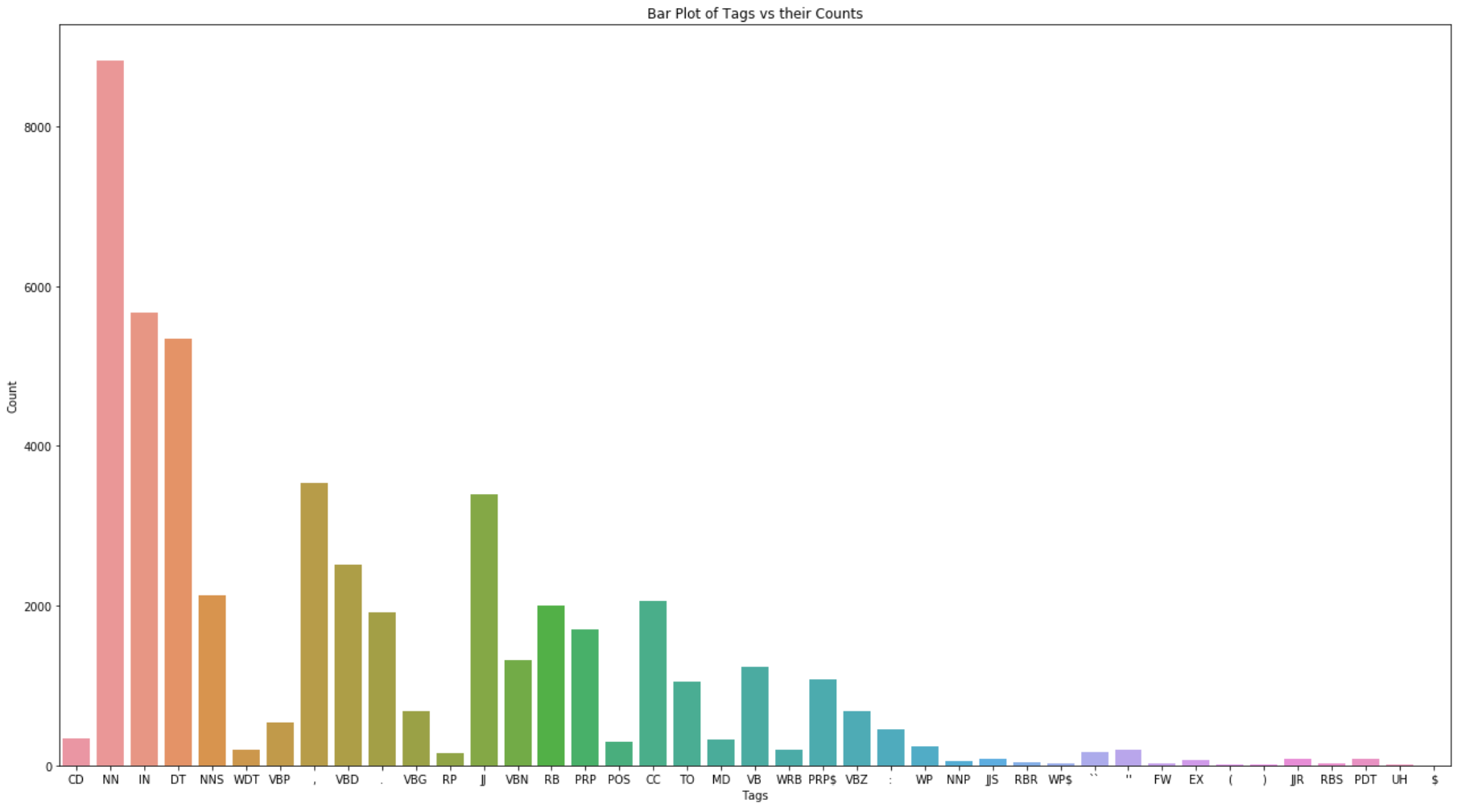
```
No. of Sentences:  1913
```

In [436]:
```
for sentence in sentenceList:
    wordList = nltk.word_tokenize(sentence)
    taggedSentence = nltk.pos_tag(wordList,lang='eng')
    tagged.append(taggedSentence)
```

In [437]:
```
countTagged = {}
c = 0
for i in tagged:
    for j in i:
        countTagged[j[1]] = countTagged.get(j[1],0) + 1
        c+=1
```

In [438]:
```
df1 = pd.DataFrame({'length':list(countTagged.keys()),'counts':list(countTagged.values())})
```

In [439]:
```
plt.figure(figsize = (22,12))
x = sns.barplot(x=list(countTagged.keys()),y=list(countTagged.values()),data=df1)
x.set_title('Bar Plot of Tags vs their Counts')
x.set(xlabel='Tags', ylabel='Count')
plt.show()
```



ROUND 2

7. Finding Noun and Verbs in the Novel

To find out noun and verbs present in the novel we use our PoS tagged List tagged which contains all the sentences of the novel tagged and stored all the words(tokenize) with their tags in taggedWords list. There are 4 and 6 different type of nouns and verbs respectively PoS tags-

Tag	Description
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present

So we use this Knowledge to built two Lists- nouns,verbs which contains all the words present in the text which are noun and verb respectively.

In [444]:
```
taggedWords = []
for i in tagged:
  for j in i:
    taggedWords.append(j)
```

In [445]:
```
nouns=[]
verbs=[]
for word,pos in taggedWords:
    if (pos == 'NN' or pos == 'NNP' or pos == 'NNS' or pos == 'NNPS'):
         nouns.append(word)
    if(pos == 'VB' or pos == 'VBD' or pos == 'VBG' or pos == 'VBN' or pos == 'VBP' or pos == 'VBZ'):
        verbs.append(word)
print('No. of Nouns',len(nouns))
print("No. of Verbs",len(verbs))
```

```
No. of Nouns 11013
No. of Verbs 6950
```

So we use this Knowledge to built two Lists
- Nouns
- Verbs
which contains all the words present in the text which are noun and verb respectively.

Issue

In the above approach most of the words present in the text have more than one sense.

Like Volume which has 6 senses of noun. Hence simply taking the first sense would not give us correct results.

In [446]:
```
print("SENSES OF VOLUME")
for sense in wn.synsets('volume'):
    print()
    print(sense.lexname(),"--",sense.definition())
```

```
SENSES OF VOLUME

noun.quantity -- the amount of 3-dimensional space occupied by an object

noun.attribute -- the property of something that is great in magnitude

noun.artifact -- physical objects consisting of a number of pages bound together

noun.communication -- a publication that is one of a set of several similar publications

noun.quantity -- a relative amount

noun.attribute -- the magnitude of sound (usually in a specified direction)
```

Solution

So we should apply Word Sense Disambiguation on the text sentence(not PoS tagged sentences).
So that we can have only 1 sense for a word in the sentence.

.Also in WordNet, a word's sense is stored as volume.n.01, we can easily distinguish whether the sense it refers to is a Noun or a Verb.

Step 1 (Disambiguate Text)

(Storing disambiguated sentence in sentenceDisambiguated list)

We use pywsd which is Python implementations of Word Sense Disambiguation technologies. It includes

- **Lesk algorithms (includes original Lesk, adapted Lesk and simple Lesk)**
- **Baseline algorithms (random sense, first sense, Most Frequent Sense)**
  ** We have use it's all-words implementation (using simple-lesk) for WSD.**

In [447]:
```
from pywsd import disambiguate

def disambiguateText():
    sentenceDisambiguated = []

    for sentence in sentenceList:
        sentenceDisambiguated.append(disambiguate(sentence))

    return sentenceDisambiguated
```

In [448]:
```
sentenceDisambiguated = disambiguateText()
print(sentenceDisambiguated[0])
```

```
[('one', Synset('one.s.06')), ('year', Synset('year.n.03')), ('ago', Synset('ago.s.01')), ('the', None), ('writer', S
ynset('writer.n.02')), ('of', None), ('the', None), ('letters', Synset('letter.n.04')), ('which', None), ('follow', S
ynset('follow.v.18')), (',', None), ('visited', Synset('visit.v.08')), ('the', None), ('battle', Synset('struggle.n.0
1')), ('field', Synset('field.n.15')), ('of', None), ('waterloo', Synset('waterloo.n.03')), ('.', None)]
```

Step 2 (Find lexical category)

Converting disambiguated sentences into a list lexName which is a tuple consisting of a word with its WordNet lexicographer file name.

A WordNet lexicographer file name specifies for each word its class(noun,verb,adjective or adverb) and its type.

In [449]:
```
def findLexName():
    lexName = []
    for sentence in sentenceDisambiguated:
        for word in sentence:
            if type(word[1])==type(None):
                continue
            lexName.append((word[0],word[1].lexname()))
    return lexName
```

In [450]:
```
lexName = findLexName()
```

In [451]:
```
print(lexName[0])
```

```
('one', 'adj.all')
```

Step 3 (Store in different list - Nouns/Verbs)

nounCatCount(dictionary) containing all the noun categories with their count in the text, and
verbCatCount(dictionary) containing all the verb categories with their count in the text.

In [452]:
```
nounCatCount = {}
verbCatCount = {}

for word in lexName:
    if word[1][0]=='n':
        nounCatCount[word[1]] = nounCatCount.get(word[1],0)+1
    elif word[1][0]=='v':
        verbCatCount[word[1]] = verbCatCount.get(word[1],0)+1

print('No of Categories for Verbs: ',(len(verbCatCount)))
print('No of Categories for Nouns: ',len((nounCatCount)))
```

```
No of Categories for Verbs:  15
No of Categories for Nouns:  26
```

In [453]:
```
for cat in verbCatCount.keys():
    print(cat,' ',verbCatCount[cat])
```

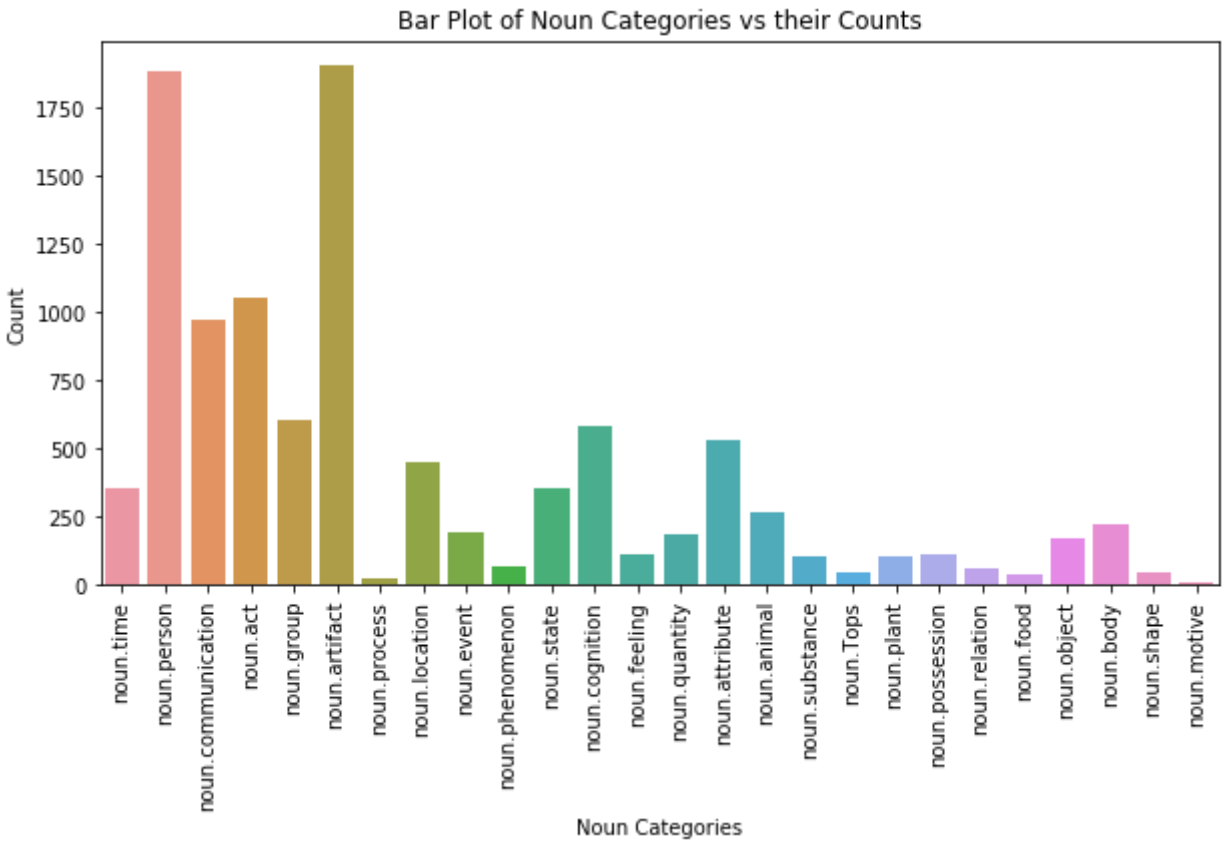
```
verb.social   424
verb.change   445
verb.perception   285
verb.possession   231
verb.stative   499
verb.communication   720
verb.cognition   373
verb.creation   175
verb.contact   528
verb.motion   611
verb.emotion   127
verb.body   159
verb.competition   156
verb.consumption   65
verb.weather   19
```

We see that there are 26 categories for nouns. There is one extra category i.e noun.Tops which is unique beginner for nouns.

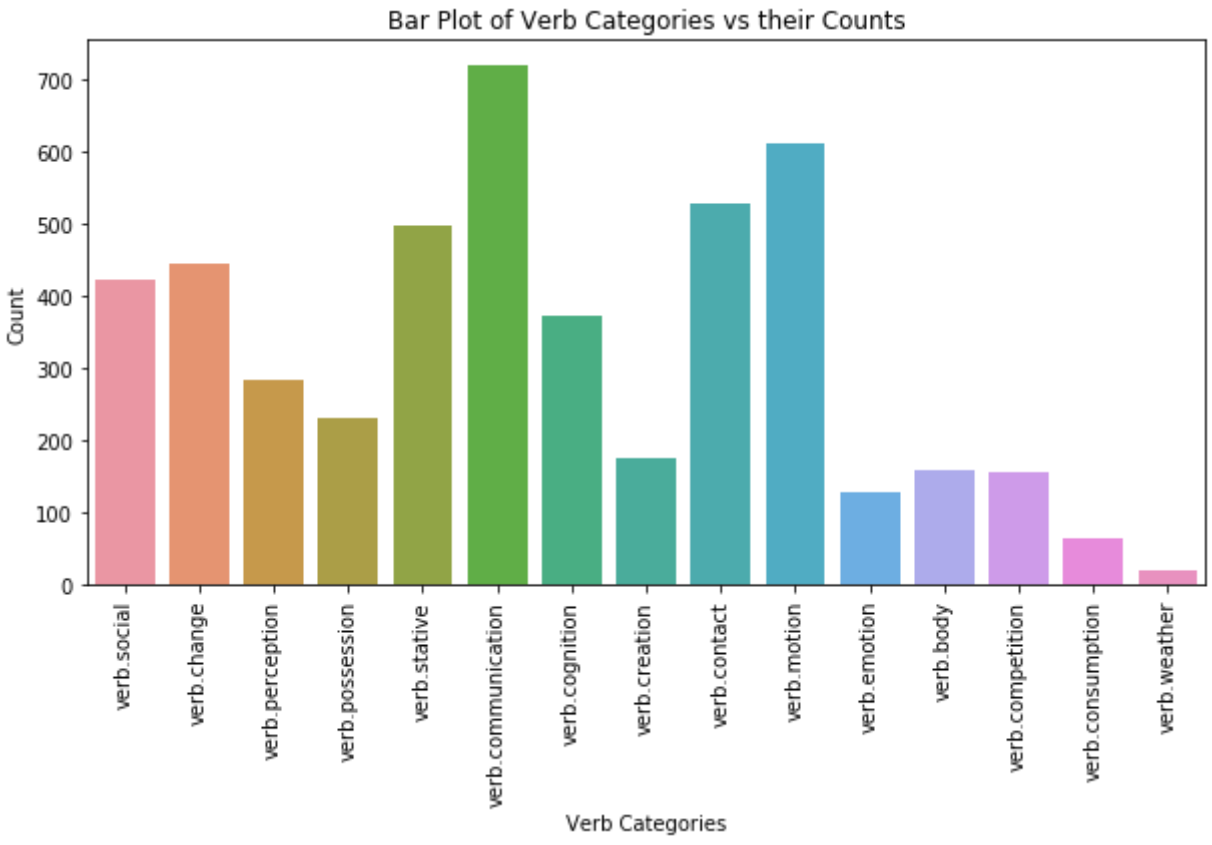
Step 4 (Plotting)

In [454]:
```
def barPlot(dictt,st):
    df1 = pd.DataFrame({'length':list(dictt.keys()),'counts':list(dictt.values())})
    plt.figure(figsize = (10,5))
    x = sns.barplot(x=list(dictt.keys()),y=list(dictt.values()),data=df1)
    x.set_title('Bar Plot of '+st+ ' vs their Counts')
    plt.xticks(rotation=90)
    x.set(xlabel=st, ylabel='Count')
    plt.show()
```

In [457]:
```
barPlot(nounCatCount,'Noun Categories')
```



In [458]:
```
barPlot(verbCatCount,'Verb Categories')
```



8. Named Entity Recognition

Named-entity recognition (NER) is process or subtask of information extraction that seeks to locate and classify named entity mention in the unstructured text into pre-defined categories such as people, organisation, location, geo-political entity, facility and vehicles.

spaCy is regarded as the fastest NLP framework in Python, with single optimized functions for each of the NLP tasks it implements. Being easy to learn and use, one can easily perform simple tasks using a few lines of code. spaCy's models can be installed as Python packages.

We have used ScaCy for our NER task.

SpaCy's named entity recognition has been trained on the OntoNotes 5.
It uses BILUO annotation instead of BIO scheme.

- **B: The first token of a multi-token entity.**
- **I: An inner token of a multi-token entity.**
- **L: The final token of a multi-token entity.**
- **U: A single-token entity.**
- **O: A non-entity token.**

In [459]:
```
import spacy
```

About Model used for NER

Downloaded the spacy NER model package through terminal using python -m spacy download en_core_web_sm command.

Here model is loaded in *ner_model* variable.

The model used is en_core_web_sm

1. **Type: Model capabilities (e.g. core for general-purpose model with vocabulary, syntax, entities and word vectors, or depent for only vocab, syntax and entities).**
2. **Genre: Type of text the model is trained on, e.g. web or news.**
3. **Size: Model size indicator, sm, md or lg. **
   Hence en_core_web_sm is a small English model trained on written web text (blogs, news, comments), that includes vocabulary, vectors, syntax and entities.

   (Here model is loaded in *ner_model* variable.)

In [460]:
```
from collections import Counter
ner_model = spacy.load(r'C:\Users\mahav\Anaconda3\Lib\site-packages\en_core_web_sm\en_core_web_sm-2.2.5')
```

Step 1. Applying the Model

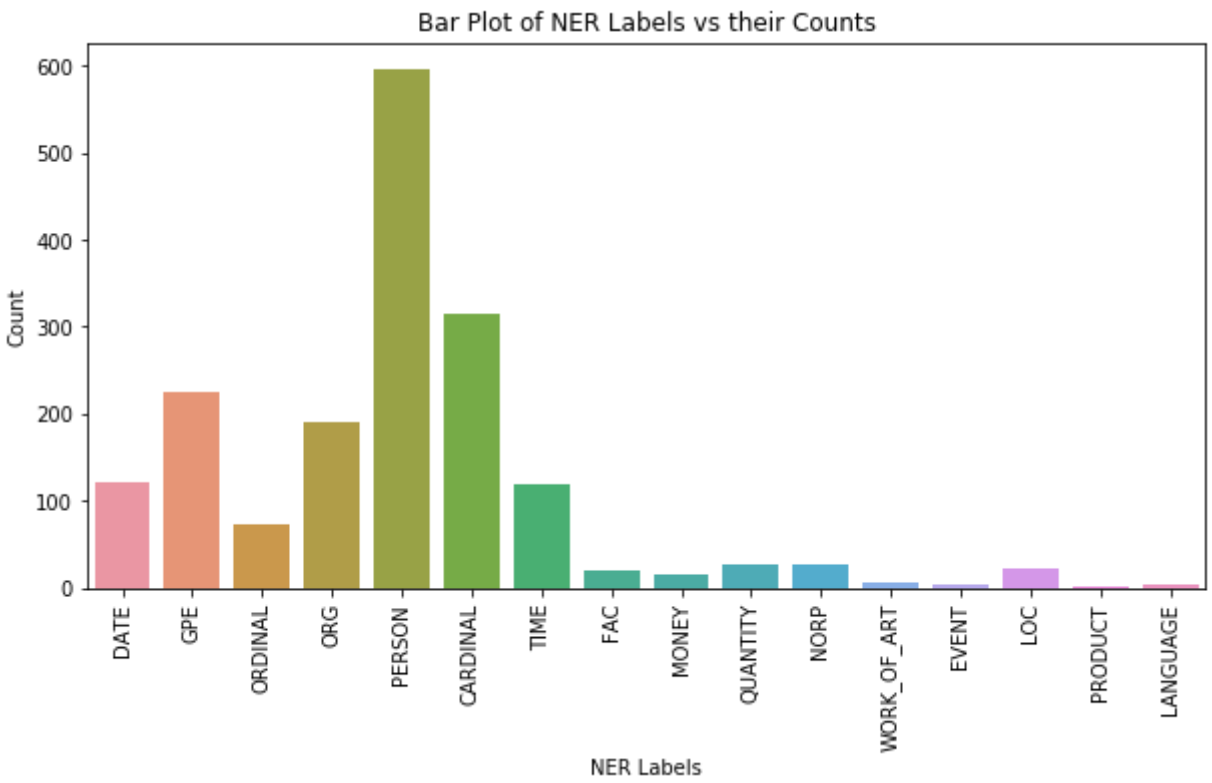
Models trained on the OntoNotes 5 corpus support the following entity types:

1. **PERSON : People, including fictional.**
2. **NORP : Nationalities or religious or political groups.**
3. **FAC : Buildings, airports, highways, bridges, etc.**
4. **ORG : Companies, agencies, institutions, etc.**
5. **GPE : Countries, cities, states.**
6. **LOC : Non-GPE locations, mountain ranges, bodies of water.**
7. **PRODUCT : Objects, vehicles, foods, etc. (Not services.)**
8. **EVENT : Named hurricanes, battles, wars, sports events, etc.**
9. **WORK_OF_ART : Titles of books, songs, etc.**
10. **LAW : Named documents made into laws.**
11. **LANGUAGE : Any named language.**
12. **DATE : Absolute or relative dates or periods.**
13. **TIME : Times smaller than a day.**
14. **PERCENT : Percentage, including ”%“.**
15. **MONEY : Monetary values, including unit.**
16. **QUANTITY : Measurements, as of weight or distance.**
17. **ORDINAL : “first”, “second”, etc.**
18. **CARDINAL : Numerals that do not fall under another type.**

NER on each sentence of the corpus.

In [464]:
```
labels = []
labCount = {}
for sent in sentenceList:
    doc = ner_model(sent)
    for entity in doc.ents:
        labels.append((entity.text,entity.label_))
```

In [462]:
```
for text,label in labels:
    labCount[label] = labCount.get(label,0)+1
barPlot(labCount,"NER Labels")
```



We see that most of labels assigned are CARDINAL , DATE

In [463]:
```
print(labels[:10])
```

```
[('one year ago', 'DATE'), ('waterloo', 'GPE'), ('wellington', 'GPE'), ('first', 'ORDINAL'), ('the day', 'DATE'), ('w
aterloo', 'ORG'), ('assassination', 'ORG'), ('lincoln', 'PERSON'), ('_', 'CARDINAL'), ('washington', 'GPE')]
```

Step 2. Testing

For testing we have hand labeled 3 paragraphs from the text. Below are the paragraphs which will be handlabeled.

In [483]:
```
para1="""Secretary Stanton, just arrived from the bedside of Mr. Seward, asked Surgeon-General Barnes what was Mr. Li
ncoln's condition. "I fear, Mr. Stanton, that there is no hope." "O, no, general; no, no;" and the man, of all other
s, apparently strange to tears, sank down beside the bed, the hot, bitter evidences of an awful sorrow trickling thro
ugh his fingers to the floor. Senator Sumner sat on the opposite side of the bed, holding one of the President's hand
s in his own, and sobbing with kindred grief. Secretary Welles stood at the foot of the bed, his face hidden, his fra
me shaken with emotion. General Halleck, Attorney-General Speed, Postmaster-General Dennison, M. B. Field, Assistant
 Secretary of the Treasury, Judge Otto, General Meigs, and others, visited the chamber at times, and then retired. Mr
s. Lincoln--but there is no need to speak of her. Mrs. Senator Dixon soon arrived, and remained with her through the
 night. All through the night, while the horror-stricken crowds outside swept and gathered along the streets, while t
he military and police were patrolling and weaving a cordon around the city; while men were arming and asking each ot
her, "What victim next?" while the telegraph was sending the news from city to city over the continent, and while the
two assassins were speeding unharmed upon fleet horses far away--his chosen friends watched about the death-bed of th
e highest of the nation. Occasionally Dr. Gurley, pastor of the church where Mr. Lincoln habitually attended, knelt d
own in prayer. Occasionally Mrs. Lincoln and her sons, entered, to find no hope and to go back to ceaseless weeping.
 Members of the cabinet, senators, representatives, generals, and others, took turns at the bedside. Chief-Justice Ch
ase remained until a late hour, and returned in the morning. """
```

In [484]:
```
para2 = """Secretary McCulloch remained a constant watcher until 5 A. M. Not a gleam of consciousness shone across th
e visage of the President up to his death--a quiet, peaceful death at last--which came at twenty-two minutes past sev
en A. M. Around the bedside at this time were Secretaries Stanton, Welles, Usher, Attorney-General Speed, Postmaster-
General Dennison, M. B. Field, Assistant Secretary of the Treasury, Judge Otto, Assistant Secretary of the Interior,
 General Halleck, General Meigs, Senator Sumner, F. R. Andrews, of New-York, General Todd, of Dacotah, John Hay, priv
ate secretary, Governor Oglesby, of Illinois, General Farnsworth, Mrs. and Miss Kenny, Miss Harris, Captain Robert Li
ncoln, son of the President, and Drs. E. W. Abbott, R. K. Stone, C. D. Gatch, Neal Hall, and Leiberman. Rev. Dr. Gurl
ey, after the event, knelt with all around in prayer, and then, entering the adjoining room where were gathered Mrs.
 Lincoln, Captain Robert Lincoln, Mr. John Hay, and others, prayed again. Soon after 9 o'clock the remains were place
d in a temporary coffin and conveyed to the White House under a small escort."""
```

Named Entities recognised from the paragraph by spaCy model.

In [510]:
```
doc = ner_model(para1)
for entity in doc.ents:
    print(entity.label_,'  ',entity.text)
```

```
PERSON    Stanton
PERSON    Seward
PERSON    Lincoln
PERSON    Stanton
PERSON    Sumner
CARDINAL    one
PERSON    kindred grief
PERSON    Welles
PERSON    Halleck
PERSON    Speed
PERSON    Postmaster
PERSON    Dennison
PERSON    M. B. Field
ORG    Treasury
PERSON    Otto
PERSON    Meigs
PERSON    Lincoln
PERSON    Dixon
CARDINAL    two
PERSON    Gurley
PERSON    Lincoln
PERSON    Lincoln
ORG    Chase
TIME    a late hour
TIME    morning
```

In [511]:
```
doc = ner_model(para2)
for entity in doc.ents:
    print(entity.label_,'  ',entity.text)
```

```
PERSON    McCulloch
CARDINAL    5
QUANTITY    twenty-two minutes past seven
PERSON    Secretaries Stanton
PERSON    Speed
PERSON    Postmaster
PERSON    Dennison
PERSON    M. B. Field
ORG    Treasury
PERSON    Otto
PERSON    Halleck
PERSON    Meigs
PERSON    Sumner
PERSON    F. R. Andrews
GPE    New-York
PERSON    Todd
GPE    Dacotah
PERSON    John Hay
PERSON    Oglesby
GPE    Illinois
PERSON    Farnsworth
PERSON    Kenny
PERSON    Harris
PERSON    Robert Lincoln
PERSON    E. W. Abbott
PERSON    R. K. Stone
PERSON    C. D. Gatch
FAC    Neal Hall
PERSON    Leiberman
PERSON    Gurley
PERSON    Lincoln
PERSON    Robert Lincoln
PERSON    John Hay
TIME    9 o'clock
ORG    the White House
```

Handlabling Named Entites in the paragharph and storing the handlabeled para in handlabled variable

In [502]:
```
handlabled1=[
    ("Secretary Stanton, just arrived from the bedside of Mr. Seward, asked Surgeon-General Barnes what was Mr. Linco
ln's condition.",
        [[10,17,'PERSON'],[56,62,'PERSON'],[106,113,'PERSON']]),
    ("I fear, Mr.Stanton, that there is no hope.' 'O, no, general; no, no;' and the man, of all others, apparently st
range to tears, sank down beside the bed, the hot, bitter evidences of an awful sorrow trickling through his fingers
 to the floor.",
        [[13,20,'PERSON']]),
    ("Senator Sumner sat on the opposite side of the bed, holding one of the President's hands in his own, and sobbin
g with kindred grief.",
        [[8,15,'PERSON'],[60,63,'CARDINAL']]),
    ("Secretary Welles stood at the foot of the bed, his face hidden, his frame shaken with emotion.",
        [[10,15,'PERSON']]),
    ("General Halleck, Attorney-General Speed, Postmaster-General Dennison, M. B. Field, Assistant Secretary of the T
reasury, Judge Otto, General Meigs, and others, visited the chamber at times, and then retired.",
        [[8,15,'PERSON'],[60,67,'PERSON'],[70,81,'PERSON'],[126,130,'PERSON'],[140,145,'PERSON']]),
    ("Mrs. Lincoln--but there is no need to speak of her.",[[5,13,'PERSON']]),
    ("Mrs. Senator Dixon soon arrived, and remained with her through the night.",
        [[13,18,'PERSON']]),
    ("All through the night, while the horror-stricken crowds outside swept and gathered along the streets, while the
military and police were patrolling and weaving a cordon around the city; while men were arming and asking each othe
r, 'What victim next?' while the telegraph was sending the news from city to city over the continent, and while the t
wo assassins were speeding unharmed upon fleet horses far away--his chosen friends watched about the death-bed of the
highest of the nation.",
        [[12,21,'TIME'],[344,347,'CARDINAL']]),
    ("Occasionally Dr. Gurley, pastor of the church where Mr. Lincoln habitually attended, knelt down in prayer.",
        [[17,23,'PERSON'],[56,63,'PERSON']],),
    ("Occasionally Mrs. Lincoln and her sons, entered, to find no hope and to go back to ceaseless weeping.",
        [[18,25,'PERSON']]),
    ("Members of the cabinet, senators, representatives, generals, and others, took turns at the bedside.",
        []),
    ("Chief-Justice Chase remained until a late hour, and returned in the morning.",
        [[14,19,'PERSON'],[35,46,'TIME'],[61,75,'TIME']])
]
```

In [512]:
```
handlabled2 = [
    ("Secretary McCulloch remained a constant watcher until 5 A. M.",
        [[10,19,'PERSON'],[48,61,'TIME']]),
    ("Not a gleam of consciousness shone across the visage of the President up to his death--a quiet, peaceful death
 at last--which came at twenty-two minutes past seven A. M.",
        [[134,152,'TIME'],[153,169,'TIME']]),
    ("Around the bedside at this time were Secretaries Stanton, Welles, Usher, Attorney-General Speed, Postmaster-Gen
eral Dennison, M. B. Field, Assistant Secretary of the Treasury, Judge Otto, Assistant Secretary of the Interior, Gen
eral Halleck, General Meigs, Senator Sumner, F. R. Andrews, of New-York, General Todd, of Dacotah, John Hay, private
 secretary, Governor Oglesby, of Illinois, General Farnsworth, Mrs. and Miss Kenny, Miss Harris, Captain Robert Linco
ln, son of the President, and Drs. E. W. Abbott, R. K. Stone, C. D. Gatch, Neal Hall, and Leiberman.",
        [[49,56,'PERSON'],[58,64,'PERSON'],[66,71,'PERSON'],[90,94,'PERSON'],[116,124,'PERSON'],[132,137,'PERSON'],[1
82,186,'PERSON'],[233,240,'PERSON'],[250,255,'PERSON'],[265,271,'PERSON'],[273,286,'PERSON'],[291,299,'GPE'],[309,313
,'PERSON'],[318,325,'GPE'],[327,335,'PERSON'],[365,372,'PERSON'],[377,385,'GPE'],[395,405,'PERSON'],[421,426,'PERSON'
],[433,439,'PERSON'],[449,463,'PERSON'],[496,508,'PERSON'],[510,521,'PERSON'],[523,534,'PERSON'],[536,545,'PERSON'],[
551,560,'PERSON']]),
    ("Rev. Dr. Gurley, after the event, knelt with all around in prayer, and then, entering the adjoining room where
 were gathered Mrs. Lincoln, Captain Robert Lincoln, Mr. John Hay, and others, prayed again.",
        [[9,15,'PERSON'],[130,137,'PERSON'],[147,161,'PERSON'],[167,175,'PERSON']]),
    ("Soon after 9 o'clock the remains were placed in a temporary coffin and conveyed to the White House under a smal
l escort.",
        [[11,20,'TIME'],[83,97,'ORG']])
]
```

Comparing Named entity recognised by spaCy model with Handlabled Paragraph using Scorer

We have used 2 inbuilt spaCy libraries for testing.

- **Scorer: Class in SpaCy which stores evaluation scores.**
- **GoldParse : A collection for training annotations.**

In [513]:
```
from spacy.gold import GoldParse
from spacy.scorer import Scorer
from tabulate import tabulate
```

(Importing libraries)

In [514]:
```
def print_results(results):
    print('F-Score : ',round(results['ents_f'],2))
    print('Precsion : ',round(results['ents_p'],2))
    print('Recall : ',round(results['ents_r'],2),'\n')

    print('EVALUATING PER ENTITY TYPE\n')
    data = []
    for key in results['ents_per_type'].keys():
        data.append([key,round(results['ents_per_type'][key]['f'],2),round(results['ents_per_type'][key]['f'],2),roun
d(results['ents_per_type'][key]['f'],2)])

    print(tabulate(data, headers=['Entity Type', 'Precsion','Recall','F1-Score']))
```

In [515]:
```
def evaluate(ner_model, examples):
    scorer = Scorer()

    for ex in range(len(examples)):
        doc_gold_text = ner_model.make_doc(examples[ex][0])
        gold = GoldParse(doc_gold_text, entities=examples[ex][1])
        pred_value = ner_model(examples[ex][0])
        scorer.score(pred_value, gold)
    return scorer.scores
```

In [521]:
```
def evaluate_hand_labeled_para(handlabled):
    results = evaluate(ner_model, handlabled)
    print_results(results)
```

evaluate function

1. **scorer : Stores all the scores of sentences.**
2. **Iterate through all sentences in handlabled text.**
3. **Make gold corpus of hand labled paragraph.**
4. **Apply spaCy model on the sentences.**
5. **Compare(score) predicted and the gold standard.**

printing results(scores)

Inside scorer we have various attributes which we can evaluate. We have listed the following in our results :

1. **Total F1 Score =** $\frac{2*Precision*Recall}{Precision+Recall}$
2. **Total Prescion =** $\frac{TruePositive}{TruePositive+TrueNegative}$
3. **Total Recall =** $\frac{TruePositive}{TruePositive+FlaseNegative}$
4. **Per Entity**
   - **A. Precsion**
   - **B. Recall**
   - **C. F1 Score**

In [522]:
```
evaluate_hand_labeled_para(handlabled1)
```

```
F-Score :  83.33
Precsion :  83.33
Recall :  83.33

EVALUATING PER ENTITY TYPE

Entity Type      Precsion    Recall    F1-Score
-------------  ----------  --------  ----------
PERSON             93.33     93.33       93.33
CARDINAL          100       100         100
TIME               66.67     66.67       66.67
ORG                 0         0           0
```

In [523]:
```
evaluate_hand_labeled_para(handlabled2)
```

```
F-Score :  66.67
Precsion :  71.43
Recall :  62.5

EVALUATING PER ENTITY TYPE

Entity Type      Precsion    Recall    F1-Score
-------------  ----------  --------  ----------
PERSON               100       100         100
CARDINAL               0         0           0
TIME                   0         0           0
```

9. Extracting Relationship between Entities

Relationship extraction is the task of extracting semantic relationships from a text. Extracted relationships usually occur between two or more entities of a certain type (e.g. Person, Organisation, Location) and fall into a number of semantic categories (e.g. married to, employed by, lives in).

For our task we use the following text from the book.

Text 1

In [608]:
```
TEXTS = [
"""Secretary Stanton, just arrived from the bedside of Mr. Seward, asked Surgeon-General Barnes what was Mr. Lincol
n's condition. "I fear, Mr. Stanton, that there is no hope." "O, no, general; no, no;" and the man, of all others, ap
parently strange to tears, sank down beside the bed, the hot, bitter evidences of an awful sorrow trickling through h
is fingers to the floor. Senator Sumner sat on the opposite side of the bed, holding one of the President's hands in
 his own, and sobbing with kindred grief. Secretary Welles stood at the foot of the bed, his face hidden, his frame s
haken with emotion. General Halleck, Attorney-General Speed, Postmaster-General Dennison, M. B. Field, Assistant Secr
etary of the Treasury, Judge Otto, General Meigs, and others, visited the chamber at times, and then retired. Mrs. Li
ncoln--but there is no need to speak of her. Mrs. Senator Dixon soon arrived, and remained with her through the nigh
t. All through the night, while the horror-stricken crowds outside swept and gathered along the streets, while the mi
litary and police were patrolling and weaving a cordon around the city; while men were arming and asking each other,
 "What victim next?" while the telegraph was sending the news from city to city over the continent, and while the two
assassins were speeding unharmed upon fleet horses far away--his chosen friends watched about the death-bed of the hi
ghest of the nation. Occasionally Dr. Gurley, pastor of the church where Mr. Lincoln habitually attended, knelt down
 in prayer. Occasionally Mrs. Lincoln and her sons, entered, to find no hope and to go back to ceaseless weeping. Mem
bers of the cabinet, senators, representatives, generals, and others, took turns at the bedside. Chief-Justice Chase
 remained until a late hour, and returned in the morning."""
]
```

Below are the named entities extracted using the model.

In [609]:
```
def ner_text():
    doc = ner_model(TEXTS[0])
    for entity in doc.ents:
        print(entity.label_,'  ',entity.text)
ner_text()
```

```
PERSON    Stanton
PERSON    Seward
PERSON    Lincoln
PERSON    Stanton
PERSON    Sumner
CARDINAL    one
PERSON    kindred grief
PERSON    Welles
PERSON    Halleck
PERSON    Speed
PERSON    Postmaster
PERSON    Dennison
PERSON    M. B. Field
ORG    Treasury
PERSON    Otto
PERSON    Meigs
PERSON    Lincoln
PERSON    Dixon
CARDINAL    two
PERSON    Gurley
PERSON    Lincoln
PERSON    Lincoln
ORG    Chase
TIME    a late hour
TIME    morning
```

Lets see the relation b/w PERSON

In [610]:
```
def filter_spans(spans):
    get_sort_key = lambda span: (span.end - span.start, -span.start)
    sorted_spans = sorted(spans, key=get_sort_key, reverse=True)
    result = []
    seen_tokens = set()
    for span in sorted_spans:
        if span.start not in seen_tokens and span.end - 1 not in seen_tokens:
            result.append(span)
        seen_tokens.update(range(span.start, span.end))
    result = sorted(result, key=lambda span: span.start)
    return result
```

In [611]:
```
def main(ner_model):
    nlp = ner_model
    print("Processing %d texts" % len(TEXTS))

    for text in TEXTS:
        doc = nlp(text)
        relations = extract_per_relations(doc)
        for r1, r2 in relations:
            print("{:<10}\t{}\t{}".format(r1.text, r2.ent_type_, r2.text))
```

We extract relations between phrases and entities using spaCy's named entity recognizer and the dependency parse.

In [612]:
```
def extract_per_relations(doc):
    spans = list(doc.ents) + list(doc.noun_chunks)
    spans = filter_spans(spans)
    with doc.retokenize() as retokenizer:
        for span in spans:
            retokenizer.merge(span)

    relations = []
    for per in filter(lambda w: w.ent_type_ == "PERSON", doc):
        if per.dep_ in ("attr", "dobj"):
            subject = [w for w in per.head.lefts if w.dep_ == "nsubj"]
            if subject:
                subject = subject[0]
                relations.append((subject, per))
        elif per.dep_ == "pobj" and per.head.dep_ == "prep":
            relations.append((per.head.head, per))
    return relations
```

In [613]:
```
main(ner_model)
```

```
Processing 1 texts
the bedside     PERSON  Mr. Seward
sobbing         PERSON  kindred grief
```

- **Mr. Seward relates to the from the bedside.**
- **kindred grief was wrongly tagged as a PERSON**

Secretary Stanton, just arrived from the bedside of Mr. Seward, asked Surgeon-General Barnes what was Mr. Lincoln's condition.

Text 2

In [614]:
```
TEXTS = ["""The last man, who sits on the extreme right of the prisoners, is Mr. Sam. Arnold. He is, perhaps, the bes
t looking of the prisoners, and theleast implicated. He has a solid, pleasant face; has been a rebel soldier, foolish
ly committed himself to Booth, with perhaps no intention to do a crime, recanted in pen and ink, and was made a natio
nal character. Had he recanted by word of mouth he might have saved himself unpleasant dreams. This shows everybody t
he absurdity of writing what they can so easily say. The best thing Arnold ever wrote was his letter to Booth refusin
g to engage in murder. Yet this recantation is more in evidence against than then his original purpose."""
        ]
```

In [615]:
```
ner_text()
```

```
PERSON    Sam
PERSON    Arnold
PERSON    Booth
PERSON    Arnold
```

Lets see the relations b/w PERSON's

In [616]:
```
main(ner_model)
```

```
Processing 1 texts
The last man    PERSON  Mr. Sam
committed       PERSON  Booth
```

- **Mr. Sam relates to the last man.**
- **Booth here refers to ARTIFACT but was wrongly recognized by the model as a Person.**

""The last man, who sits on the extreme right of the prisoners, is Mr. Sam. Arnold. He is, perhaps, the best looking of the prisoners, and the least implicated. He has a solid, pleasant face; has been a rebel soldier, foolishly committed himself to Booth""

First extraction done by the extracter was suprisingly very well!.

References

1. https://spacy.io/api for NER, Extracting Relations
2. https://github.com/alvations/pywsd for WSD
3. NLTK
4. WordNet

Thanks!

Team BruteForce-

1. Abhinav Tiwari- 17ucs007
2. Ansh Mehta- 17ucs027
3. Shubham Pabuwal- 17ucs157