NLP_PROJECT 1

TEAM NAME: **N3**
TEAM MEMBERS: Nallamaddi Sritej Reddy (17ucs094)
Lohit Reddy Arrabothu (17ucs082)
Bharat Rawat (16ucc025)

TITLE: **“**The Adventures of Sherlock Holmes”
[http://www.gutenberg.org/ebooks/1661]
AUTHOR: Arthur Conan Doyle

NUMBER OF WORDS: 98332 **NUMBER OF UNIQUE_WORDS:** 8169
NUMBER OF CHARACTERS: 479745 **Number Of Sentences** :4834

PREPROCESSING STEPS:

1. We removed all the Roman numbers because they were only preceding Chapter names and contents of the book.
2. Removing all the chapter names with regular expressions as they are repeated several times in the book.
3. Replacing apostrophes with hidden words.

   For example: Replacing “i’m” with “I am”
   “he’s” with “he is”
4. Replacing tab spaces with single space and double line spaces with single line space.
5. We used PorterStemmer to apply process of Stemming over all the words.It reduces a word to its root word.

   For example: refundable to refund, liking to like and so on
6. We then removed all the non-alphanumeric symbols just before extracting tokens from the textual data.

   We used this regular expression to do so,

   t=re.sub(r”[^\w]”, “ ”, text)

Importing the text:

We named the book that we used as ‘nlptext’.
Character set encoding of the book was in ‘utf-8’.
We obtained the textual data from book with below command

```
T =  open('nlptext.txt', encoding = 'utf-8', errors = 'ignore').read()
```

Pre-processing the text:

```
T_strip = T.strip()
T = T_strip.replace('\n\n','\n')
T = T.replace('\n\n','\n')
T = T.replace('\t',' ')
```

- Regular expression for removing roman numbers
  ```
  text=re.sub(r"[^\w][ivx]+\."," ",text)
  ```

- We used the following command to remove chapter names, here “a scandal in bohemia” is the chapter name. Similar command is applied for all chapter names.
  ```
  text=re.sub(r"a scandal in bohemia","",text)
  ```

- R.E for substituting apostrophes with hidden words.
  ```
  text = re.sub(r"i'm", "I am", text)
  text = re.sub(r"he's", "he is", text)
  ```

- R.E for removing all the non-alphanumeric symbols.
  ```
  t = re.sub(r"[^\w]", " ", text)
  ```
  Substituting lines with spaces.
  ```
  t = re.sub(r"\n" , " ",t)
  ```

Tokenizing the text:

```
words = t.split(" ")
```
(we extracted tokens from text by splitting w.r.t space)

Analysing the frequency distribution of tokens:

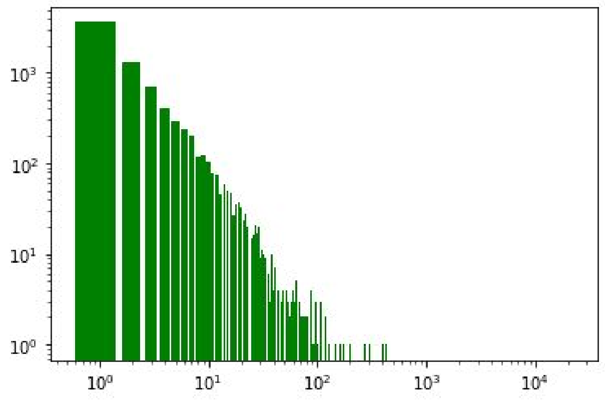
- Extracting unique words from words.

```
        unique_words = {}
                for i in words:
                        if i not in unique_words:
                                unique_words[i] = 1
                        else:
                                unique_words[i] +=1
```

- Extracting Frequency of unique words.

```
        freq_distr = {}
                for i,j in unique_words.items():
                        if j not in freq_distr:
                        freq_distr[j] = 1
                else:
                        freq_distr[j] +=1
```

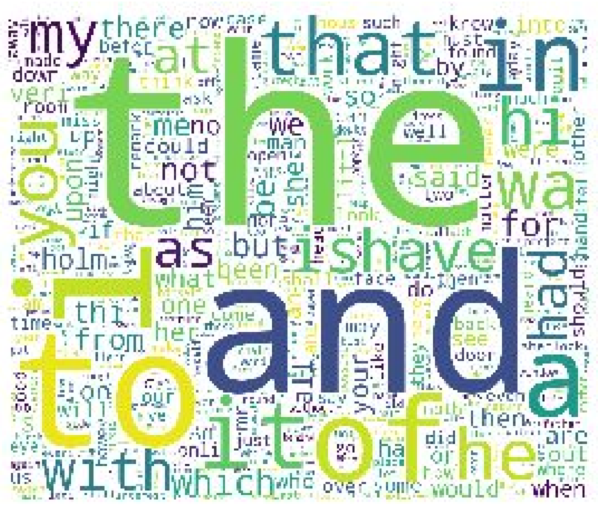
- We are using log scale over this bar graph to make it visually more analyzable.Words which are occurring very less number of times are large in number whereas words which are occurring many times are few



```
plt.bar(freq_distr.keys(), freq_distr.values(), color='g')
plt.yscale('log')
plt.xscale('log')
plt.show()
```

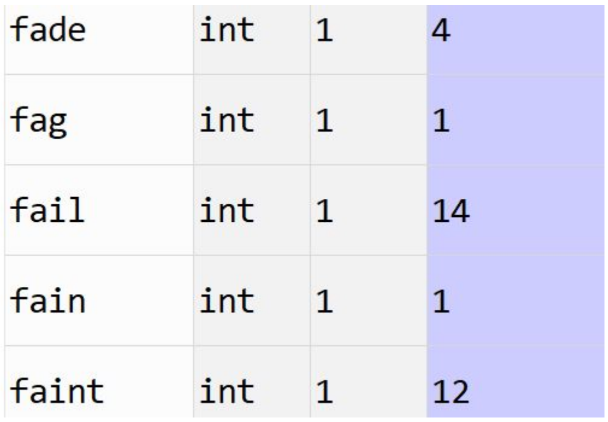
Creating a word cloud of T:

```
from wordcloud import WordCloud
        wordcloud = WordCloud(max_words=1200,
                background_color = 'white',
                width = 1200,
                height = 1000,).generate_from_frequencies(s_words)
        plt.imshow(wordcloud)
        plt.axis('off')
        plt.show()
```



Words and their frequency:

Unique_words are stored in this way, where the word “fade” has been seen 4 times ,similarly for other words their frequency is represented.



fade	int	1	4
fag	int	1	1
fail	int	1	14
fain	int	1	1
faint	int	1	12

(Screenshot from unique_word dictionary)

Removing Stop words:

(We extracted list of stop words from a website and stored it in a textfile ‘stop_words’.)

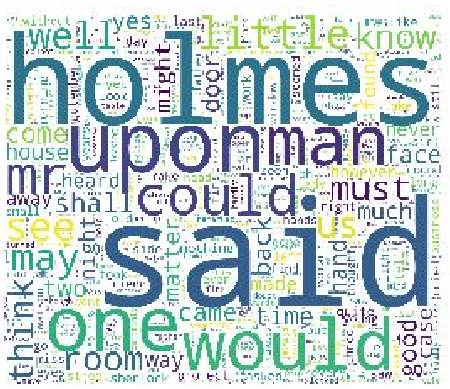
```
swords =  open('stop_words.txt', encoding = 'utf-8', errors = 'ignore').read()
swords=swords.replace('\n',' ')
s=swords.split(" ")
nostop_w={}
```

(if any word in unique_words matched with the word in stop words then we didn’t take it into our new words dictionary.)

```
for i,j in unique_words.items():
    if i not in s:
        nostop_w[i]=j
```

“There's been a significant difference in total number of words after removing stop words since they will be used in almost all sentences. We can even observe the word cloud after removing stop words which shows completely different words after removing them.”

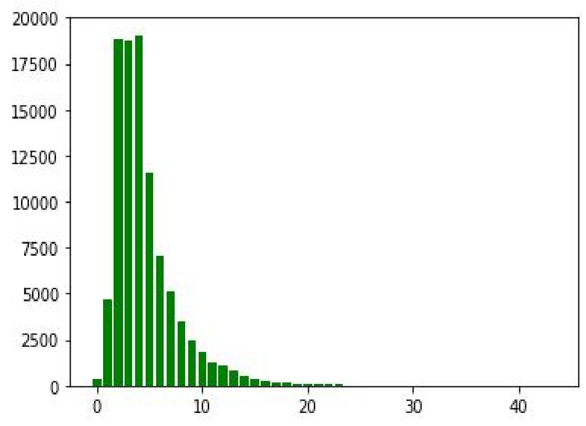
```
wordcloud1 = WordCloud(max_words=1200,
                       background_color = 'white',
                       width = 1200,
                       height = 1000,).generate_from_frequencies(nostop_w)
plt.imshow(wordcloud1)
plt.axis('off')
plt.show()
```



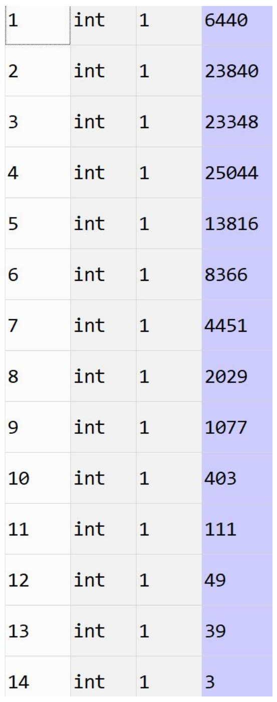
(word cloud after removing stop words.)

Word Length VS Frequency:

```
f_words={}
for i,j in unique_words.items() :
        if len(i) not in f_words:
            f_words[len(i)]=j        ( key represents length of word and
        else:                          value says no.of such words )
            f_words[len(i)]+=j
plt.bar(f_words.keys(), f_words.values(), color='g')
plt.show()
```



(word len in x-axis its freq in y-axis)



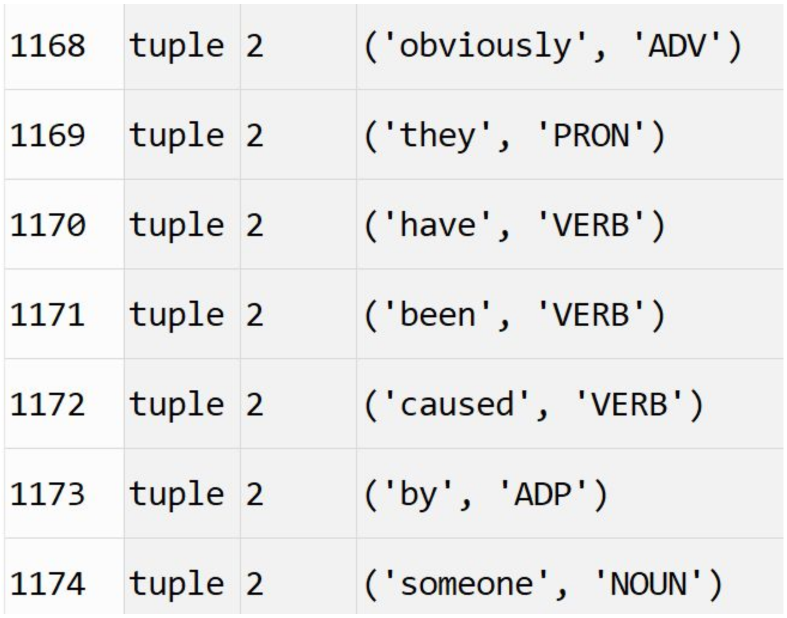
1	int	1	6440
2	int	1	23840
3	int	1	23348
4	int	1	25044
5	int	1	13816
6	int	1	8366
7	int	1	4451
8	int	1	2029
9	int	1	1077
10	int	1	403
11	int	1	111
12	int	1	49
13	int	1	39
14	int	1	3

(screenshot of f_words dictionary.)

PoS tagging (using brown corpus):

We are taking training set(from brown corpus) from 100th sentence till end and test set from the beginning to 100th sentence. Perceptron tagger with load given as false is used because it tags in the way we train it. We are training tagger over training set and after testing, it gave 97.4% accuracy.

```
from nltk.corpus import brown
token = word_tokenize(text)
brown_news_tagged = brown.tagged_sents(categories='news',
tagset='universal')
brown_train = brown_news_tagged[100:]
brown_test = brown_news_tagged[:100]
from nltk.tag.perceptron import PerceptronTagger
tagger = PerceptronTagger(load=False)
tagger.train(brown_train)
accuracy=tagger.evaluate(brown_test)  (accuracy came out to be 0.974 i.e, 97.4%.)
p=tagger.tag(token)
```

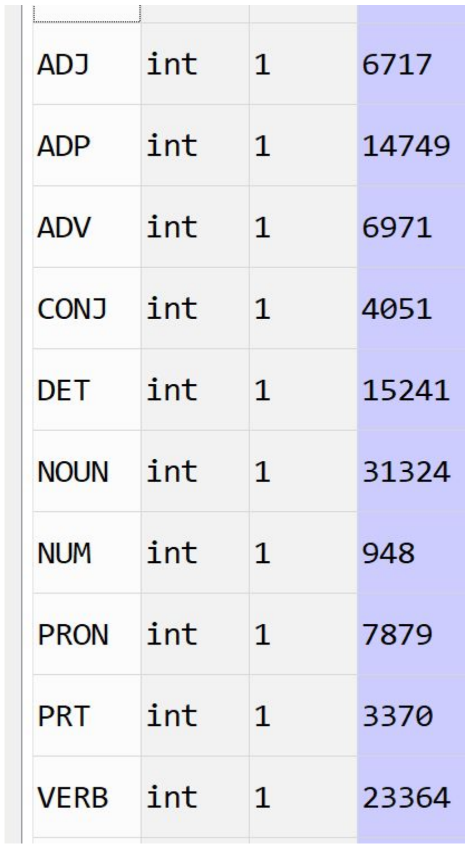


1168	tuple	2	('obviously', 'ADV')
1169	tuple	2	('they', 'PRON')
1170	tuple	2	('have', 'VERB')
1171	tuple	2	('been', 'VERB')
1172	tuple	2	('caused', 'VERB')
1173	tuple	2	('by', 'ADP')
1174	tuple	2	('someone', 'NOUN')

(screenshot of some of tagged words using trained tagger.)

Distribution Of Various Tags:

Below data is frequency distribution of Various tags in our text .
By analyzing it we can easily say that there are a lot of nouns and verbs tags compared to other tags.



ADJ	int	1	6717
ADP	int	1	14749
ADV	int	1	6971
CONJ	int	1	4051
DET	int	1	15241
NOUN	int	1	31324
NUM	int	1	948
PRON	int	1	7879
PRT	int	1	3370
VERB	int	1	23364

There are 31,324 nouns , 23,364 verbs out of 98,332 words in our textual data.

CODE:

```
import re
import matplotlib.pyplot as plt
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize            #including all the needed modules
from wordcloud import WordCloud
from nltk.corpus import brown
from nltk.tag.perceptron import PerceptronTagger
from collections import Counter

T =  open('nlptext.txt', encoding = 'utf-8', errors = 'ignore').read()  #importing text
T_strip = T.strip()
T = T_strip.replace('\n\n','\n')
T = T.replace('\n\n','\n')
T = T.replace('\t',' ')

text = re.sub('\*','',T)
text= re.sub('_','',text)                                          #pre-processing steps
text = text.lower()
text=re.sub(r"[^\w][ivx]+\."," ",text )
text=re.sub(r"a scandal in bohemia","",text )
text=re.sub(r"the red-headed league","",text )
text=re.sub(r"a case of identity","",text )
text=re.sub(r"the boscombe valley mystery","",text )
text=re.sub(r"the five orange pips","",text )
text=re.sub(r"the man with the twisted lip","",text )
text=re.sub(r"the adventure of the blue carbuncle","",text )
text=re.sub(r"the adventure of the speckled band","",text )
text=re.sub(r"the adventure of the engineer’s thumb","",text )
text=re.sub(r"the adventure of the noble bachelor","",text )
text=re.sub(r"the adventure of the beryl coronet","",text )
text=re.sub(r"the adventure of the copper beeches","",text )

text = re.sub(r"i'm", "i am", text)
text = re.sub(r"he's", "he is", text)
text = re.sub(r"she's", "she is", text)
text = re.sub(r"it's", "it is", text)
text = re.sub(r"that's", "that is", text)
text = re.sub(r"what's", "what is", text)
text = re.sub(r"where's", "where is", text)
text = re.sub(r"who's","who is",text)
text = re.sub(r"how's","how is",text)
text = re.sub(r"'s","",text)
text = re.sub(r"\'ll", " will", text)
text = re.sub(r"\'re", " are", text)
text = re.sub(r"\'ve", " have", text)
text = re.sub(r"\'re", " are", text)
text = re.sub(r"\'d", " would", text)
text = re.sub(r"won't", "will not", text)
text = re.sub(r"can't", "can not", text)
text = re.sub(r"ain't","am not",text)
text = re.sub(r"\'t"," not",text)

t = re.sub(r"[^\w]", " ", text)
t = re.sub(r"\n" , " ",t)
words = t.split(" ")                    #obtaining the tokens

char_num=0                              #total no.of char in text
for i in words :
   char_num+=len(i)

unique_words = {}                       #extracting Unique words
for i in words:
   if i not in unique_words:
      unique_words[i] = 1
   else:
     unique_words[i] +=1
freq_distr = {}                       #Freq distribution of words w.r.t no.of occurences
for i,j in unique_words.items():
   if j not in freq_distr:
      freq_distr[j] = 1
   else:
      freq_distr[j] +=1
plt.bar(freq_distr.keys(), freq_distr.values(), color='g')
plt.yscale('log')
plt.xscale('log')
plt.show()
```

```
ps = PorterStemmer()                    #using PorterStemmer for obtaining root words
s_words={}
for i in words:
    if ps.stem(i) not in s_words :
        s_words[ps.stem(i)]=1
    else :
        s_words[ps.stem(i)]+=1

f_words={}                                                  #freq of words w.r.t word length
for i,j in s_words.items() :
    if len(i) not in f_words:
        f_words[len(i)]=j
    else :
        f_words[len(i)]+=j
plt.bar(f_words.keys(), f_words.values(), color='g')
plt.show()
wordcloud = WordCloud(max_words=1200,      #extracting word cloud w.r.t word freq
            background_color = 'white',
               width = 1200
               height = 1000,
            ).generate_from_frequencies(s_words)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()

swords =  open('stop_words.txt', encoding = 'utf-8', errors = 'ignore').read()
swords=swords.replace('\n',' ')
s=swords.split(" ")
nostop_w={}
for i,j in unique_words.items():                           #removing stop words
    if i not in s:
        nostop_w[i]=j
wordcloud1 = WordCloud(max_words=1200,     #word cloud after stop words removal
                            background_color = 'white',
         width = 1200,
            height = 1000,
            ).generate_from_frequencies(nostop_w)
plt.imshow(wordcloud1)
plt.axis('off')
plt.show()                          #we are using sentences under news category because it
token= word_tokenize(text)  #has various forms of sentences suitable for training tagger
brown_news_tagged = brown.tagged_sents(categories='news', tagset='universal')
brown_train = brown_news_tagged[100:]          #training set
brown_test = brown_news_tagged[:100]           #testing set
tagger = PerceptronTagger(load=False)
tagger.train(brown_train)                         #training tagger over brown corpus
accuracy=tagger.evaluate(brown_test)  (accuracy came out to be 0.974 i.e, 97.4%.)
p=tagger.tag(token)
counts = Counter( tag for word,  tag in p)
```
****

Functionalities of various packages included:

- **Import re**
  This package is used for implementing regular expressions in the code.
- **import matplotlib.pyplot as plt**
  This specific import line merely imports the module "matplotlib.pyplot" and binds to the name "plt".
- **from nltk.stem import PorterStemmer**
  From natural language toolkit we are using porter stemmer algorithm to convert the various forms of words to their root forms.
  'nltk' contains text processing libraries for tokenization
  (from nltk.tokenize import word_tokenize)
- **from nltk.tag.perceptron import PerceptronTagger**
  Perceptron tagger predicts output based on the current weights and inputs
  Update its weights, if the prediction != the label.
- **from wordcloud import WordCloud**
  For generating wordcloud in Python, modules needed are – matplotlib and wordcloud.
- **from nltk.corpus import brown**
  The modules in this package provide functions that can be used to read corpus files in a variety of formats. We are using brown corpus for our tagging.
- **from collections import Counter**
  Counter is a container that keeps track of how many times equivalent values are added. Counter supports three forms of initialization. Its constructor can be called with a sequence of items, a dictionary containing keys and counts, or using keyword arguments mapping string names to counts.

NLP_PROJECT 2

TEAM NAME: N3
TEAM MEMBERS: Nallamaddi Sritej Reddy (17ucs094)
Lohit Reddy Arrabothu (17ucs082)
Bharat Rawat (16ucc025)
TITLE: "The Rod and Gun Club"
[http://www.gutenberg.org/ebooks/60838]

AUTHOR: Harry Castlemon
NUMBER OF WORDS: 79984
NUMBER OF UNIQUE_WORDS: 5778
NUMBER OF CHARACTERS: 325611 **Number Of Sentences**: 3078

Pre-processing steps:

1. We removed all the Roman numbers because they were only preceding Chapter names and contents of the book.
2. Removing all the chapter names with regular expressions as they are repeated several times in the book.
3. Replacing apostrophes with hidden words.
For example: Replacing "how's" with "how is","where's" with "where is".
4. Replacing tab spaces with single space and double line spaces with single line space.
5. We used PorterStemmer to apply process of Stemming over all the words.It reduces a word to its root word.
For example: refundable to refund, liking to like and so on
6. We then removed all the non-alphanumeric symbols just before extracting tokens from the textual data. We used this regular expression to do so, t=re.sub(r"[^\w]", " ", text).

Importing the text:

We named the book that we used as ‘nlptext3’. Character set encoding of the book was in ‘utf-8’. We obtained the textual data from book with below command

```
        T = open('nlptext3.txt', encoding = 'utf-8', errors = 'ignore').read()
```

Pre-processing the text:

```
T_strip = T.strip()
T = T_strip.replace('\n\n','\n')
T = T.replace('\n\n','\n')
T = T.replace('\t',' ')
```

● Regular expression for removing roman numbers

```
                text=re.sub(r"[^\w][ivx]+\."," ",text)
```

● We used the following command to remove chapter names, here “a scandal in bohemia” is the chapter name. Similar command is applied for all chapter names.

```
        text=re.sub(r"some disgusted boys","",text)
```

● R.E for substituting apostrophes with hidden words.

```
 text = re.sub(r"i'm", "I am", text)     text = re.sub(r"he's", "he is", text)
```

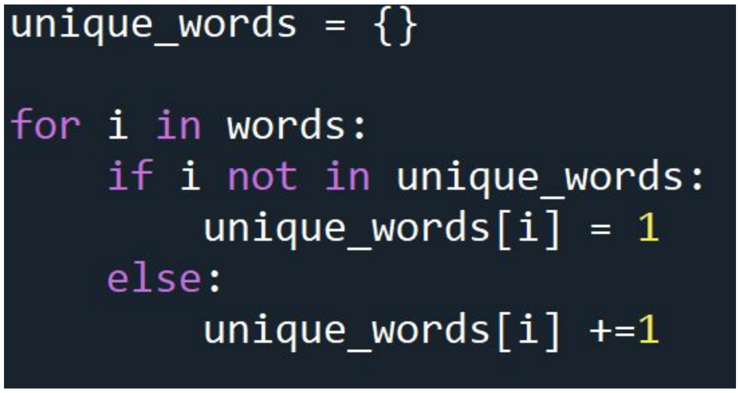
● R.E for removing all the non-alphanumeric symbols. t = re.sub(r"[^\w]", " ", text)
Substituting lines with spaces. t = re.sub(r"\n" , " ",t)

Tokenizing the text:

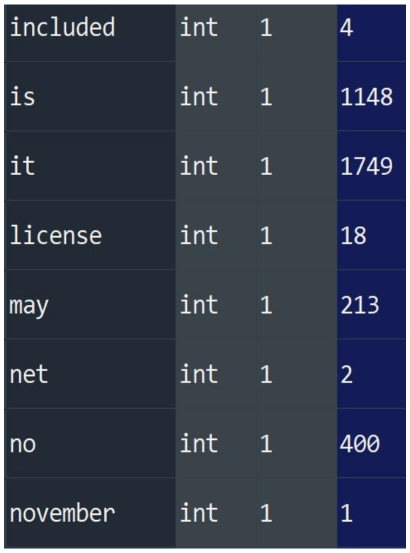
words = t.split(" ") (we extracted tokens from text by splitting w.r.t space)

<u>Analysing the frequency distribution of tokens:</u>

- <u>Extracting unique words from words.</u>



```
unique_words = {}

for i in words:
    if i not in unique_words:
        unique_words[i] = 1
    else:
        unique_words[i] +=1
```

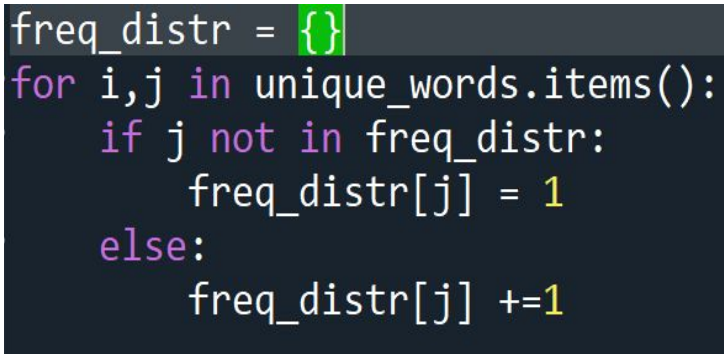


included	int	1	4
is	int	1	1148
it	int	1	1749
license	int	1	18
may	int	1	213
net	int	1	2
no	int	1	400
november	int	1	1

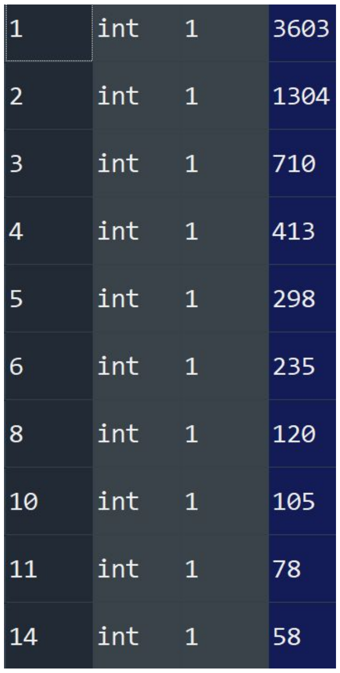
(ss of unique words dictionary)

Unique words dictionary has each word as its key and its frequency as its value. In the above screenshot ‘included’ is occuring 4 times and ‘it’ is appearing 1749 times.

- <u>Extracting Frequency of unique word</u>



```
freq_distr = {}
for i,j in unique_words.items():
    if j not in freq_distr:
        freq_distr[j] = 1
    else:
        freq_distr[j] +=1
```

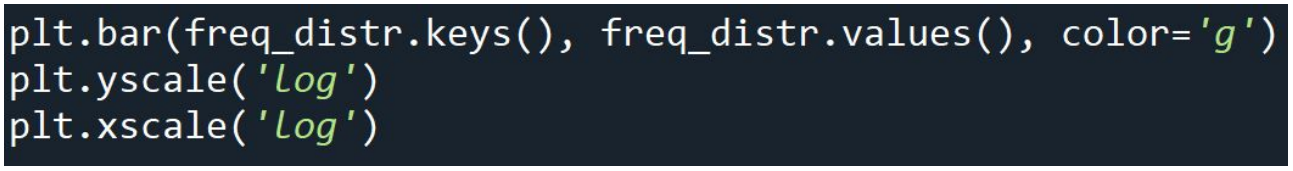


1	int	1	3603
2	int	1	1304
3	int	1	710
4	int	1	413
5	int	1	298
6	int	1	235
8	int	1	120
10	int	1	105
11	int	1	78
14	int	1	58

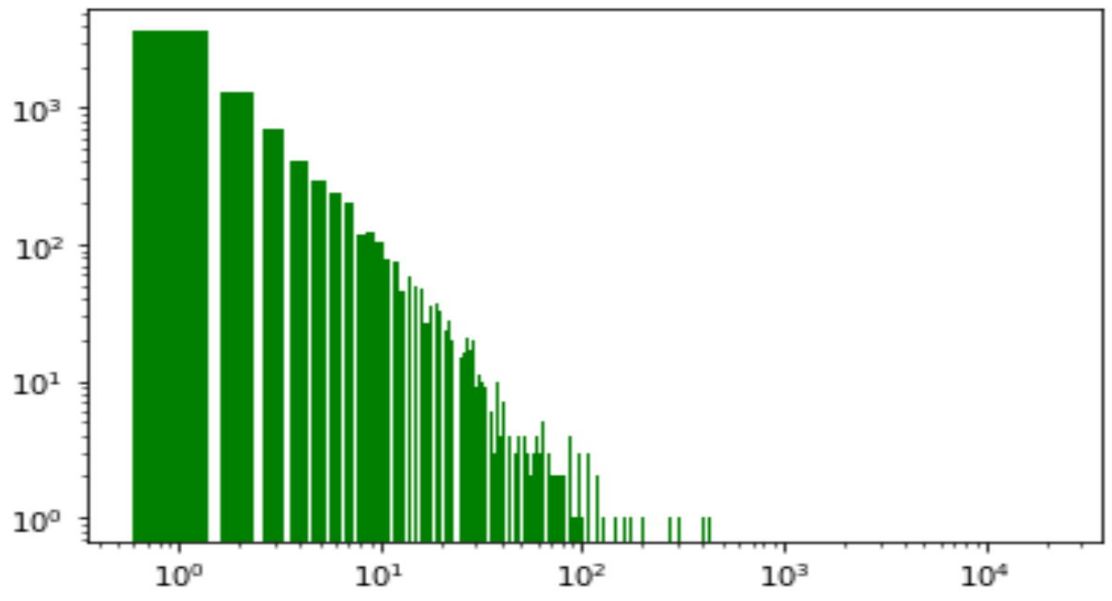
(s.s-1)

In freq_distr we are storing word occurrences as keys and number of such words as values. For key 1 in the above screenshot (s.s-1) value is 3603 which is implying that number of words that occurred only once in the text are 3603 in number.
Similarly number of words that occurred twice are 1304 in number.

- We are using log scale over this bar graph to make it visually more analyzable.

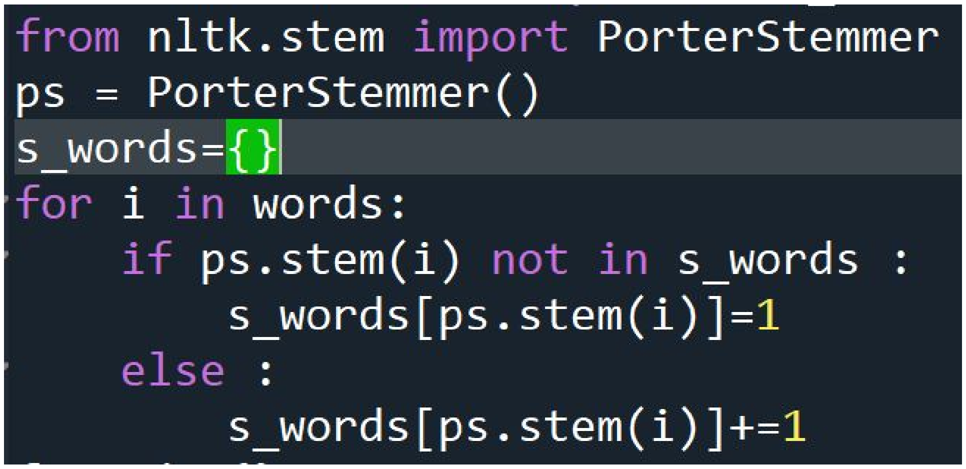


```
plt.bar(freq_distr.keys(), freq_distr.values(), color='g')
plt.yscale('log')
plt.xscale('log')
```



- Words which are occurring very less number of times are large in number whereas words which are occurring many times are few.

Applying stemming on words:



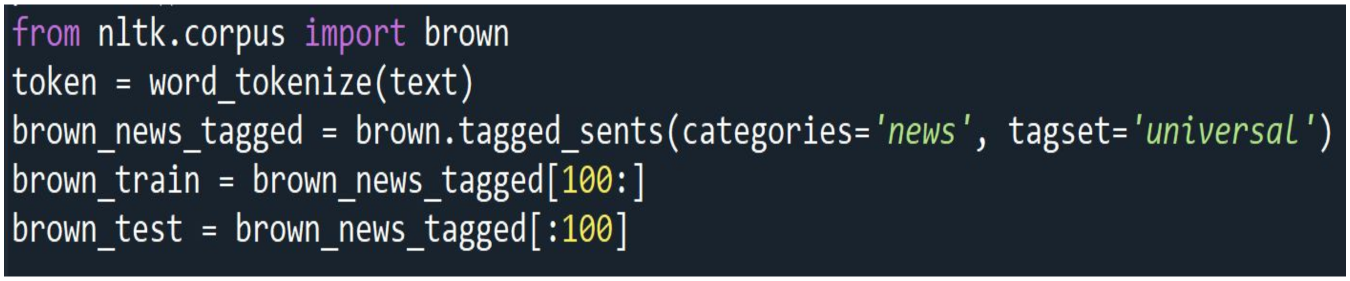
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
s_words={}
for i in words:
    if ps.stem(i) not in s_words :
        s_words[ps.stem(i)]=1
    else :
        s_words[ps.stem(i)]+=1
```

We are using Porter Stemmer algorithm for applying stemming on the words. In s_words stem form of word is stored as key and values as its frequency.

Stemming won’t be preferred when we want to apply POS tags on words because for parts of speech recognition tense of a word is vital.Stemming may alter the pos tags of words.

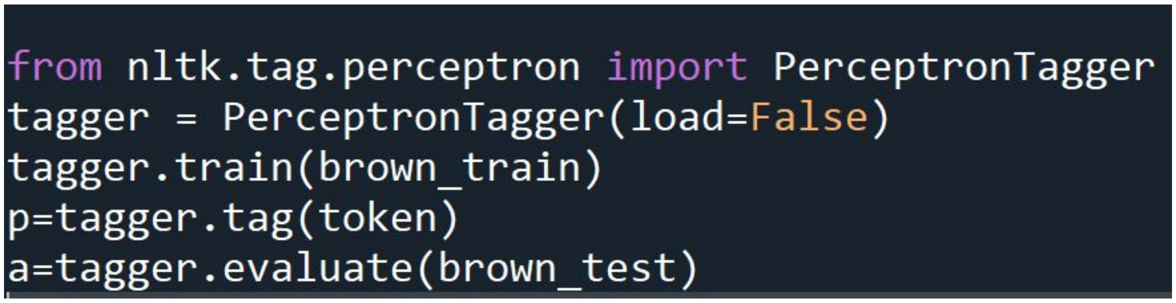
PoS tagging (using brown corpus):

We are taking training set(from brown corpus) from 100th sentence till end and test set from the beginning to 100th sentence.



```python
from nltk.corpus import brown
token = word_tokenize(text)
brown_news_tagged = brown.tagged_sents(categories='news', tagset='universal')
brown_train = brown_news_tagged[100:]
brown_test = brown_news_tagged[:100]
```

We are using categories as news because it has various topics which could help us in training a better tagger.

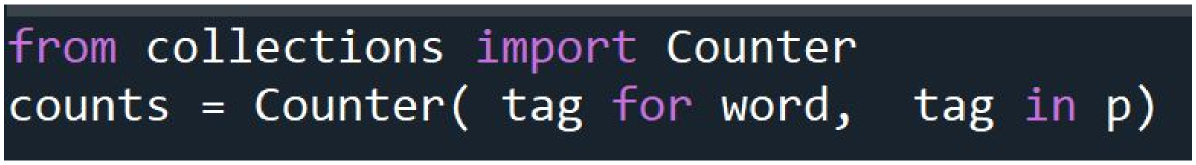


```python
from nltk.tag.perceptron import PerceptronTagger
tagger = PerceptronTagger(load=False)
tagger.train(brown_train)
p=tagger.tag(token)
a=tagger.evaluate(brown_test)
```

Perceptron tagger with load given as false is used because it tags in the way we train it. We are training tagger over training set and after testing on test set, it gave 97.4% accuracy.

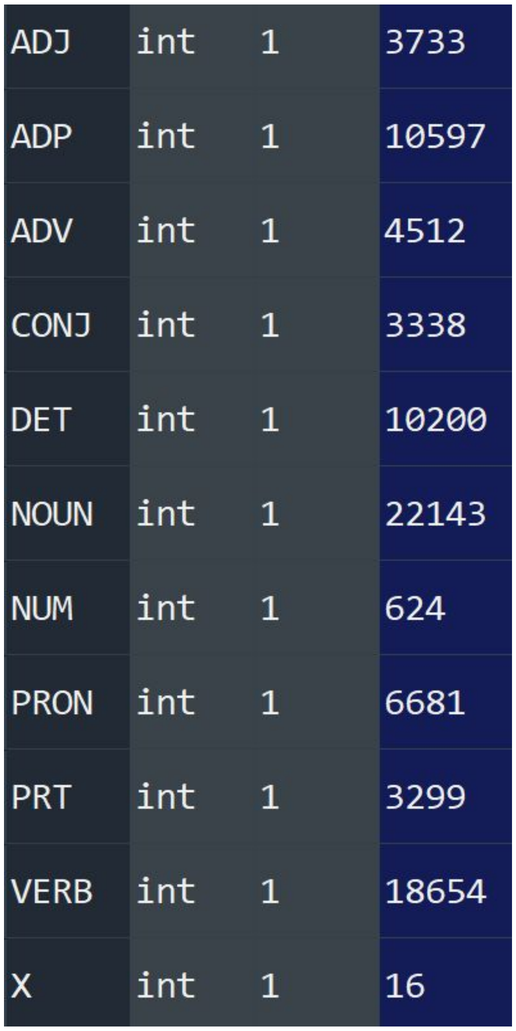
In “p” we have all the pos tags w.r.t each word stored in tuples format.

Obtaining distribution of Pos tags:



```
from collections import Counter
counts = Counter( tag for word,  tag in p)
```

Counter is a container that keeps track of how many times equivalent values are added.

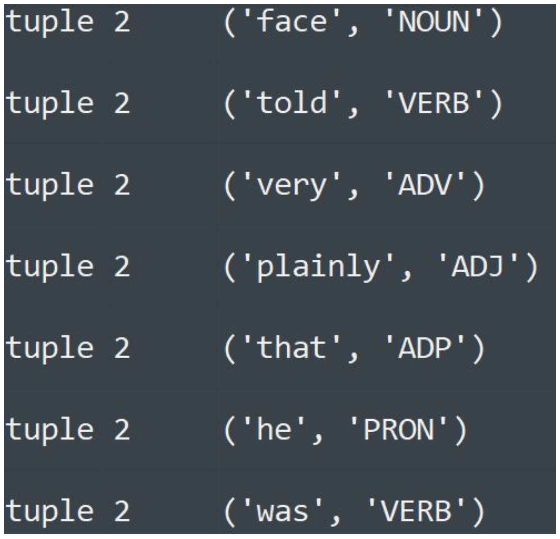


ADJ	int	1	3733
ADP	int	1	10597
ADV	int	1	4512
CONJ	int	1	3338
DET	int	1	10200
NOUN	int	1	22143
NUM	int	1	624
PRON	int	1	6681
PRT	int	1	3299
VERB	int	1	18654
X	int	1	16

(ss of **counts**)

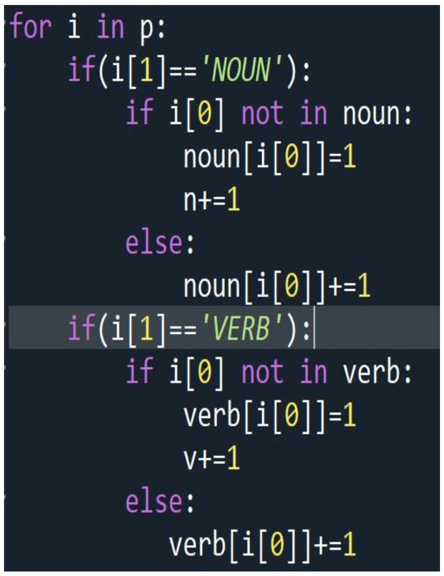
Here key is Pos tag and values are corresponding frequency of that tag.
There are 3733 adjectives, 4512 adverbs. Similarly frequency of other Pos tags can be understood.

Part 1: Finding the Nouns and Verbs:-



```
tuple 2     ('face', 'NOUN')
tuple 2     ('told', 'VERB')
tuple 2     ('very', 'ADV')
tuple 2     ('plainly', 'ADJ')
tuple 2     ('that', 'ADP')
tuple 2     ('he', 'PRON')
tuple 2     ('was', 'VERB')
```

(p is getting stored in this tuples form**)**

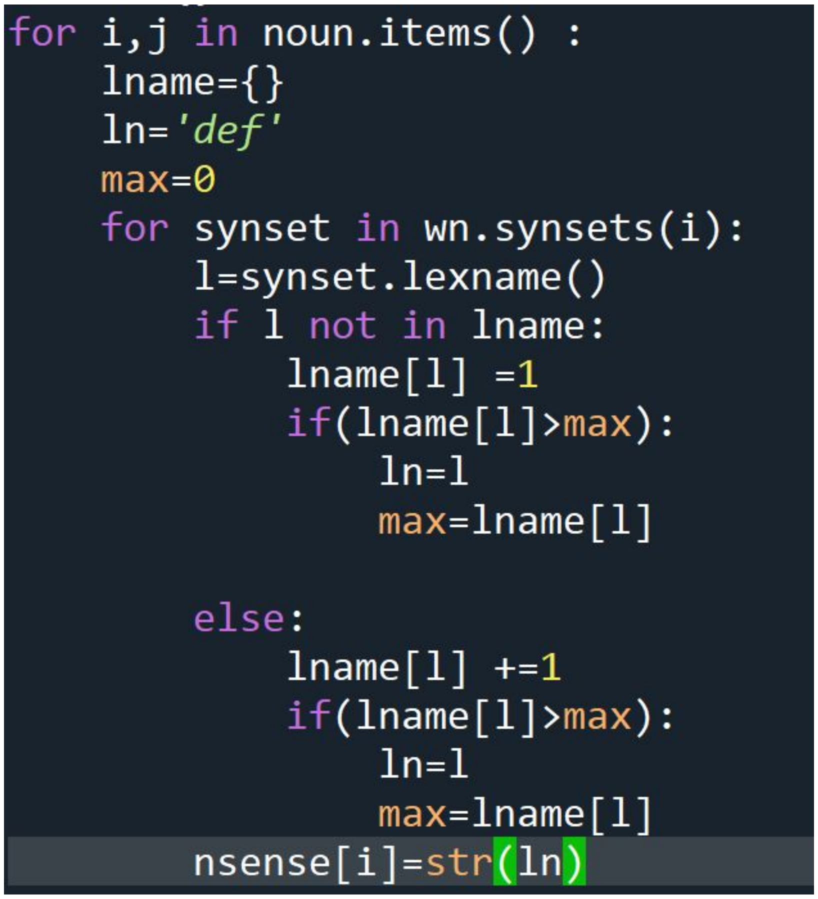


```
for i in p:
    if(i[1]=='NOUN'):
        if i[0] not in noun:
            noun[i[0]]=1
            n+=1
        else:
            noun[i[0]]+=1
    if(i[1]=='VERB'):
        if i[0] not in verb:
            verb[i[0]]=1
            v+=1
        else:
          verb[i[0]]+=1
```

(i[0] represents a word with i[1] as its respective pos tag. Here we are storing all the nouns with their frequencies in a list named **noun**. Similarly for Verbs.)

Now we have all the nouns and verbs list stored separately.

Finding the categories that these words fall under in the Wordnet:

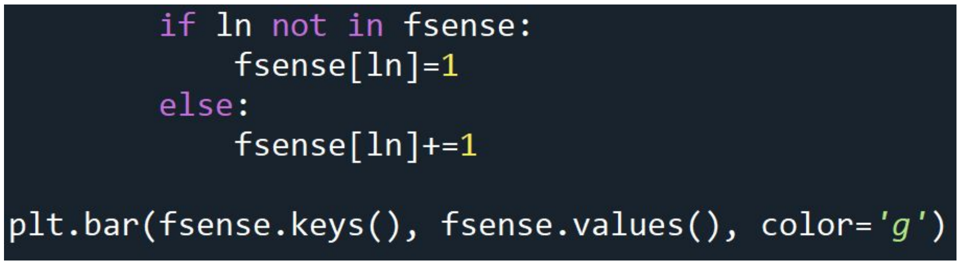


```
for i,j in noun.items() :
    lname={}
    ln='def'
    max=0
    for synset in wn.synsets(i):
        l=synset.lexname()
        if l not in lname:
            lname[l] =1
            if(lname[l]>max):
                ln=l
                max=lname[l]

        else:
            lname[l] +=1
            if(lname[l]>max):
                ln=l
                max=lname[l]
        nsense[i]=str(ln)
```

(here i is the word and j is its frequency. ‘wn.synsets(i)’ gives synonyms of that word.**)**
synset.lexname() gives us the category of that word. We are storing categories for that word w.r.t all its synonyms in **lname** and we are picking the category which is occuring most for its synonyms. In **nsense** we are storing the noun word as key and category of it as value.

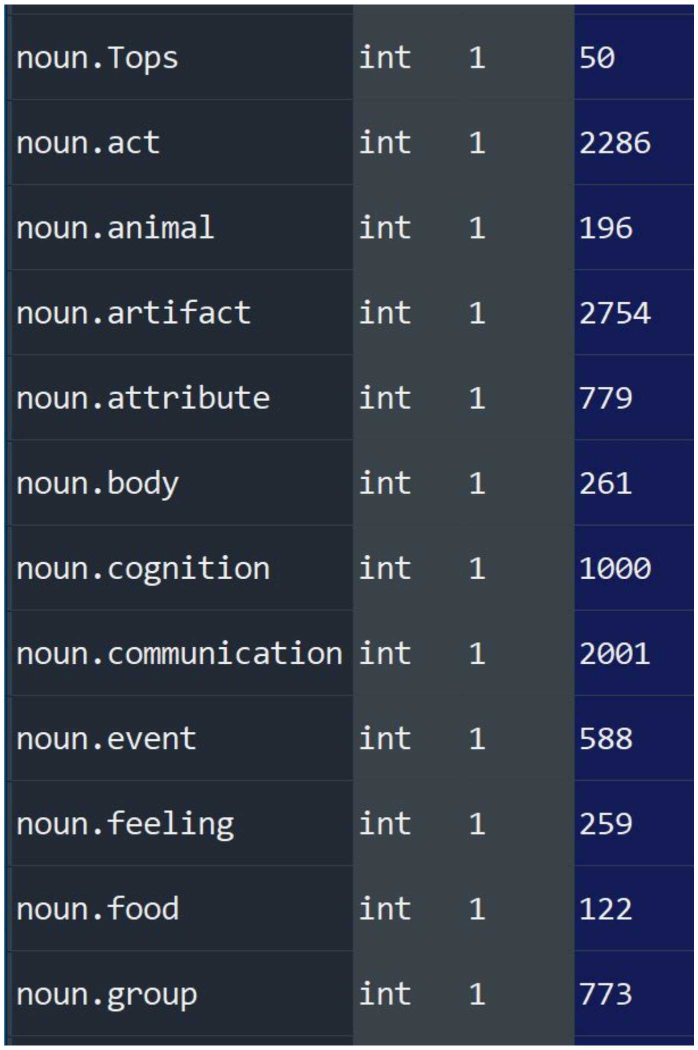
Now we have all the nouns and its category in **nsense.**

Frequency of categories:



```
        if ln not in fsense:
            fsense[ln]=1
        else:
            fsense[ln]+=1

plt.bar(fsense.keys(), fsense.values(), color='g')
```

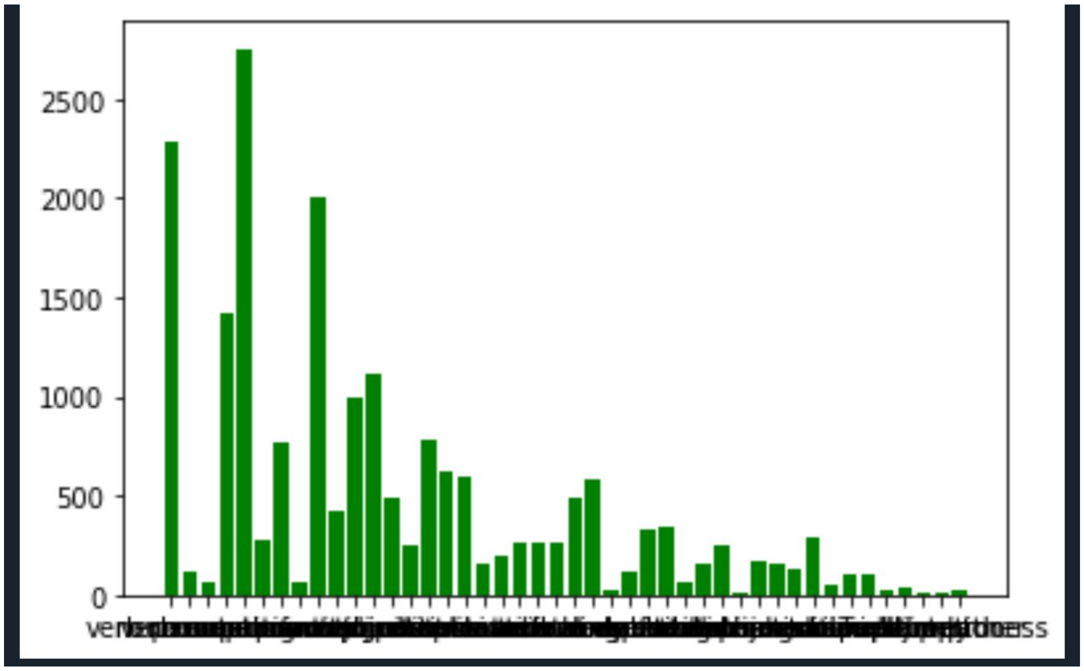
In fsense we are keeping categories as key and values as its frequency.
There are 44 different categories totally with 25 categories for nouns itself,



noun.Tops	int	1	50
noun.act	int	1	2286
noun.animal	int	1	196
noun.artifact	int	1	2754
noun.attribute	int	1	779
noun.body	int	1	261
noun.cognition	int	1	1000
noun.communication	int	1	2001
noun.event	int	1	588
noun.feeling	int	1	259
noun.food	int	1	122
noun.group	int	1	773

(screenshot of some of those categories with their frequencies, fsense keys and their values are shown in this screenshot **)**

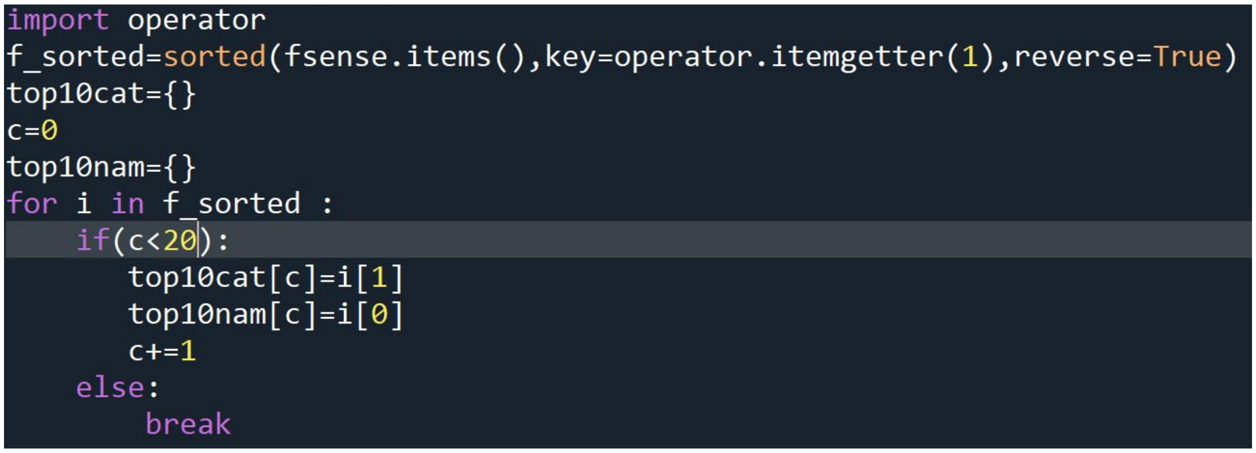
Histogram of Categories vs Frequency



(x-axis represents categories, y-axis represents frequency**)**
X-axis is keys of fsense (category names) , y-axis is values w.r.t keys of fsense(frequency of each category)

But in this graph as categories in x-axis are large in number(total 44) we are unable to visualize for which category a bar belongs to .

So we are sorting the frequencies and taking top 20 categories w.r.t their occurrences



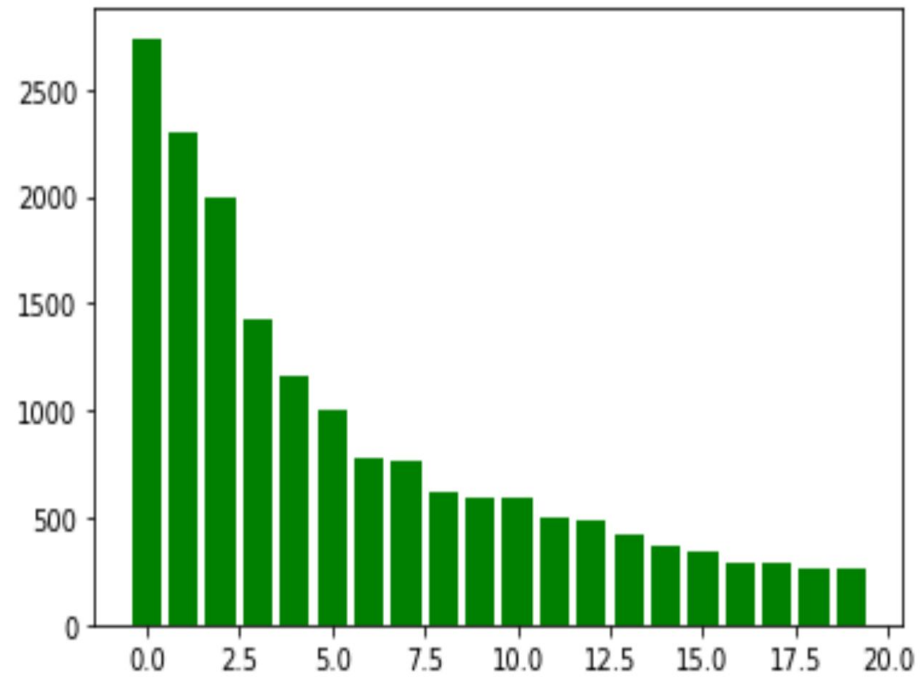
```python
import operator
f_sorted=sorted(fsense.items(),key=operator.itemgetter(1),reverse=True)
top10cat={}
c=0
top10nam={}
for i in f_sorted :
    if(c<20):
        top10cat[c]=i[1]
        top10nam[c]=i[0]
        c+=1
    else:
         break
```

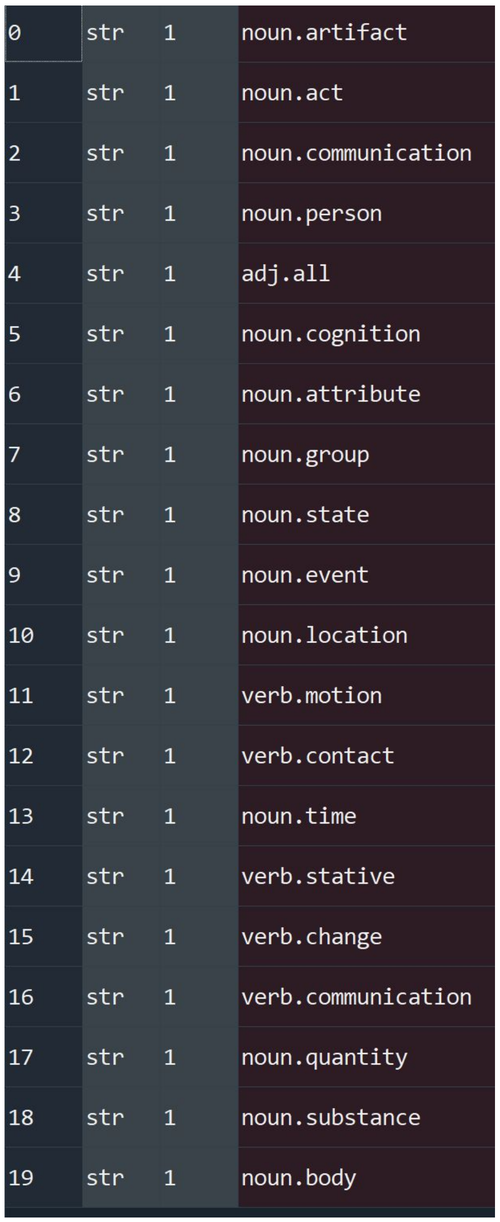
In **f_sorted** we are storing sorted form of **fsense** w.r.t values .

Operator module is used for multiple level sorting .We are using operator.itemgetter(1) for sorting here. As we are sorting w.r.t values '1' is passed into the function. If we wanted to sort according to keys then itemgetter(0) would have been used .

In **top10nam** key represents index which is assigned to each of the top20 categories and value represents category name whereas in **top10cat** key represents index and value represents frequency .So now in histogram x-axis represents index corresponding to a category and y-axis gives frequency w.r.t to index.

```
plt.bar(top10cat.keys(), top10cat.values(), color='g')
```



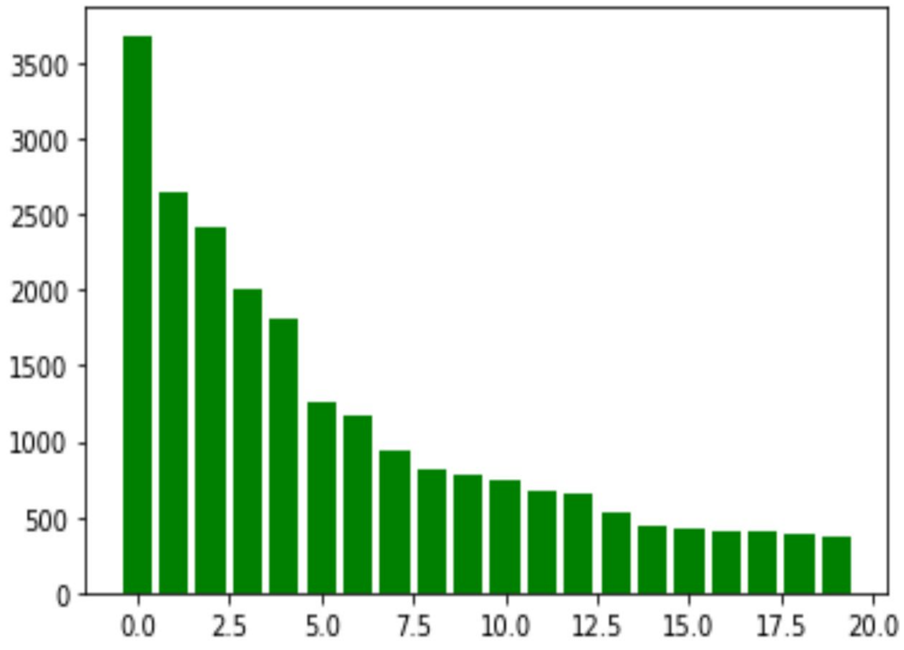


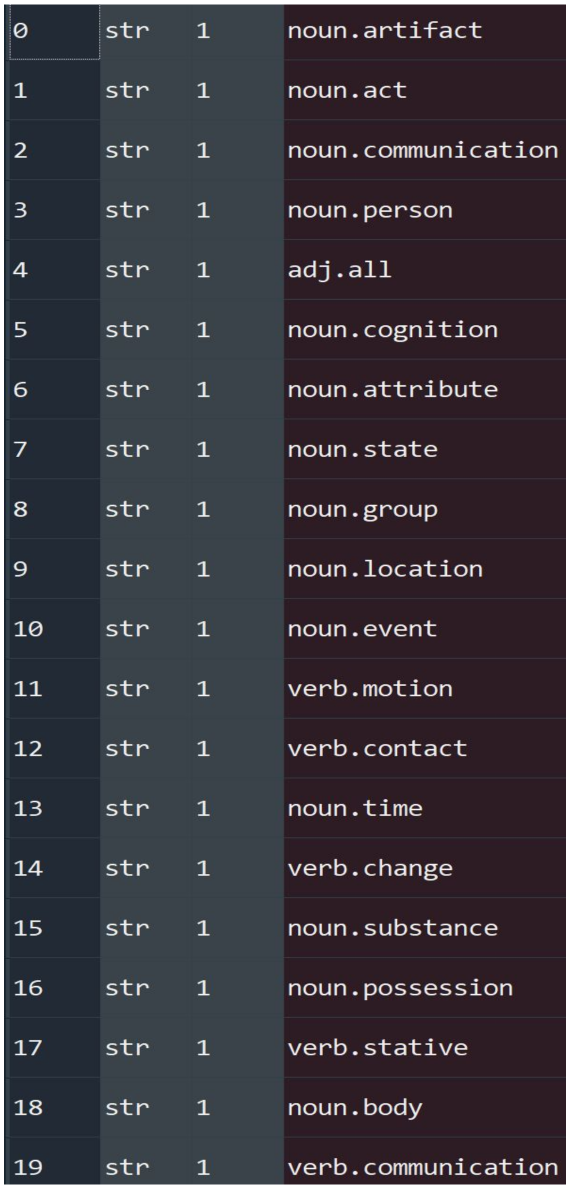
0	str	1	noun.artifact
1	str	1	noun.act
2	str	1	noun.communication
3	str	1	noun.person
4	str	1	adj.all
5	str	1	noun.cognition
6	str	1	noun.attribute
7	str	1	noun.group
8	str	1	noun.state
9	str	1	noun.event
10	str	1	noun.location
11	str	1	verb.motion
12	str	1	verb.contact
13	str	1	noun.time
14	str	1	verb.stative
15	str	1	verb.change
16	str	1	verb.communication
17	str	1	noun.quantity
18	str	1	noun.substance
19	str	1	noun.body

Plot of top10cat.keys() vs top10cat.values(Book2) **This table represents top10nam{} (Left most column shows indices and right most column are categories)**

Above plot shows the frequency of occurences of top 20 categories of noun words in Book 2.
Analyzation of above plot should be done in this way ….
index 0 in x-axis represents noun.artifact which is shown in the table beside the plot.
The bar marked with index 5 represents frequency of noun.cognition (Since in table ,5th index is marked by noun.cognition).

Similar plots for book 1 :

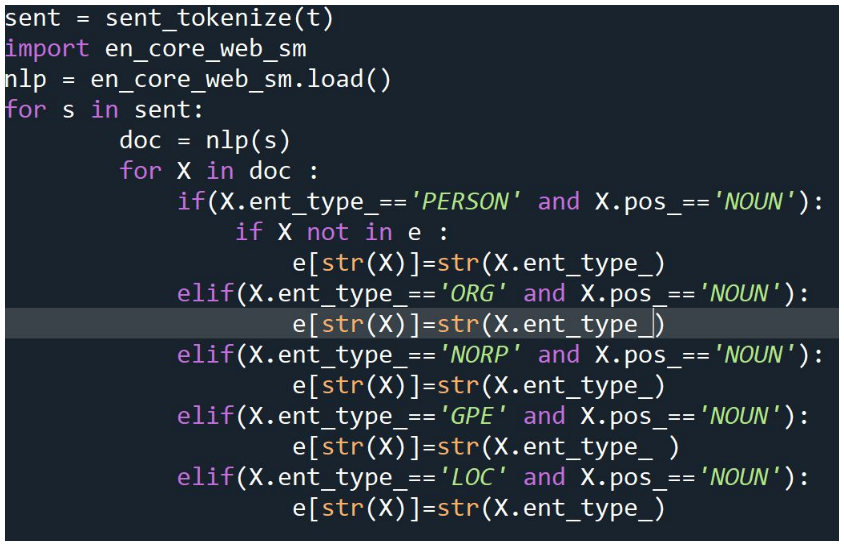




0	str	1	noun.artifact
1	str	1	noun.act
2	str	1	noun.communication
3	str	1	noun.person
4	str	1	adj.all
5	str	1	noun.cognition
6	str	1	noun.attribute
7	str	1	noun.state
8	str	1	noun.group
9	str	1	noun.location
10	str	1	noun.event
11	str	1	verb.motion
12	str	1	verb.contact
13	str	1	noun.time
14	str	1	verb.change
15	str	1	noun.substance
16	str	1	noun.possession
17	str	1	verb.stative
18	str	1	noun.body
19	str	1	verb.communication

There's only a slight change in top20 categories in book1 compared to book2. For Example We can see a swap between (7,8) positions and (9,10) positions .
Compared to book2 ,book1 has a huge difference between first and second category.

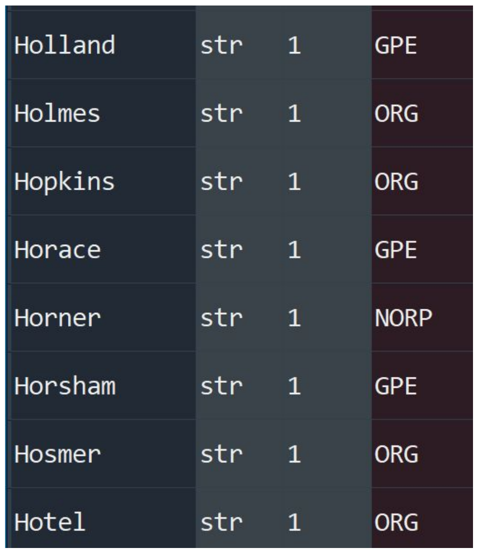
Part -2 : Recognising all the named entities:



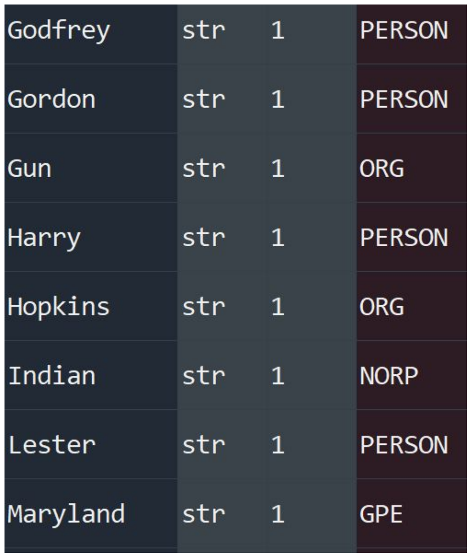
```
sent = sent_tokenize(t)
import en_core_web_sm
nlp = en_core_web_sm.load()
for s in sent:
        doc = nlp(s)
        for X in doc :
            if(X.ent_type_=='PERSON' and X.pos_=='NOUN'):
                if X not in e :
                    e[str(X)]=str(X.ent_type_)
            elif(X.ent_type_=='ORG' and X.pos_=='NOUN'):
                    e[str(X)]=str(X.ent_type_)
            elif(X.ent_type_=='NORP' and X.pos_=='NOUN'):
                    e[str(X)]=str(X.ent_type_)
            elif(X.ent_type_=='GPE' and X.pos_=='NOUN'):
                    e[str(X)]=str(X.ent_type_ )
            elif(X.ent_type_=='LOC' and X.pos_=='NOUN'):
                    e[str(X)]=str(X.ent_type_)
```

(en_core_web_sm is a small english model trained on return web test, which can provide us with entity, word dependency information.**)**
Firstly we are tokenizing the text w.r.t sentences and then we are obtaining dependency parse and named entities from this sentence by using en_core_web_sm and storing all this information in variable **doc.** If entity type is either PERSON, ORGANISATION, NORP, GPE, LOC and their pos tag is noun then we are storing them in a list e[] with key as word and values as its entity type.

There are **429** named entities in this book1 and only **323** entities in book2:



Holland	str	1	GPE
Holmes	str	1	ORG
Hopkins	str	1	ORG
Horace	str	1	GPE
Horner	str	1	NORP
Horsham	str	1	GPE
Hosmer	str	1	ORG
Hotel	str	1	ORG

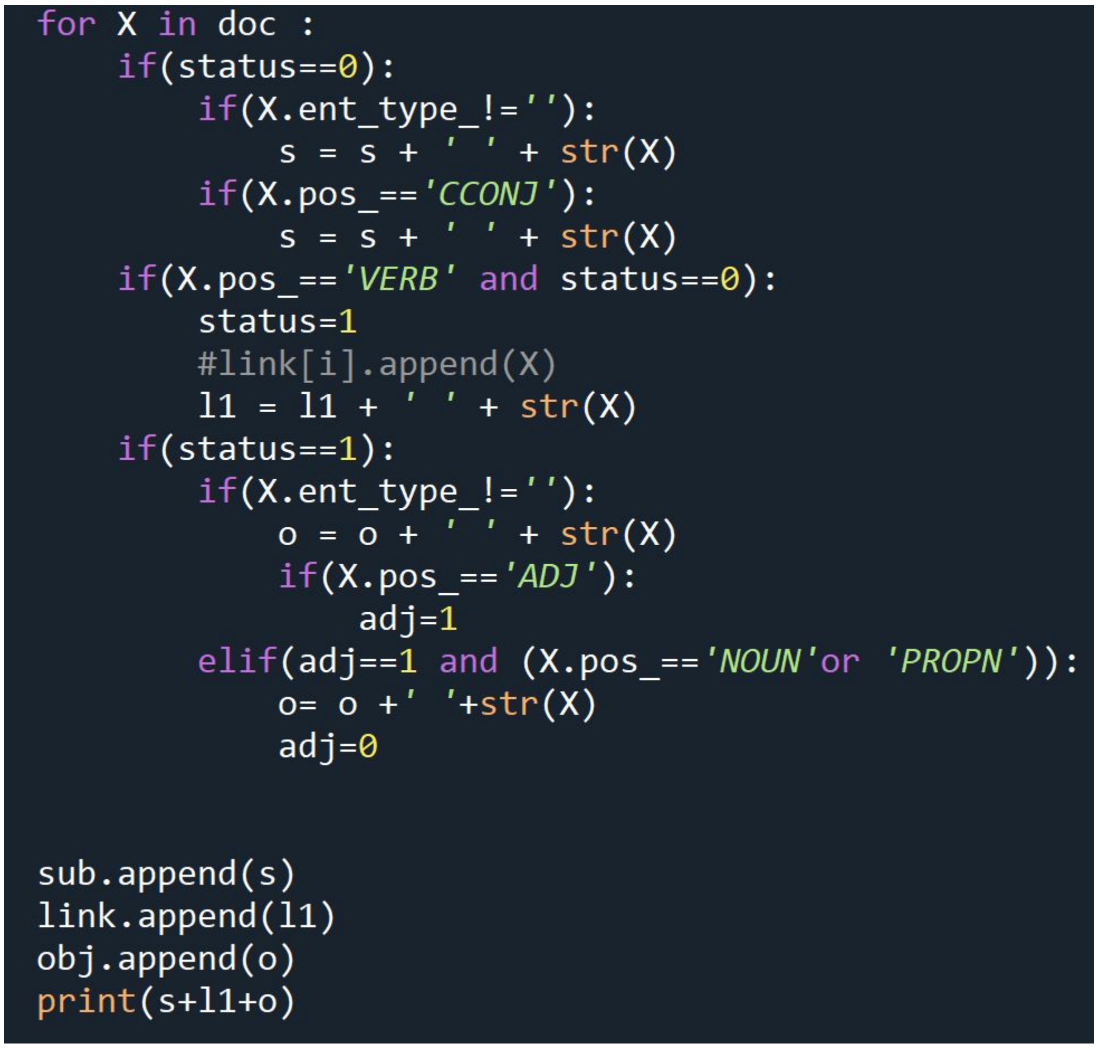


Godfrey	str	1	PERSON
Gordon	str	1	PERSON
Gun	str	1	ORG
Harry	str	1	PERSON
Hopkins	str	1	ORG
Indian	str	1	NORP
Lester	str	1	PERSON
Maryland	str	1	GPE

(Screenshot of words with their entity types .Left screenshot is some of entities from book1 and Right screenshot is some of entities from book2)

Additional Exercises: Relation between the entities:-

1st method:



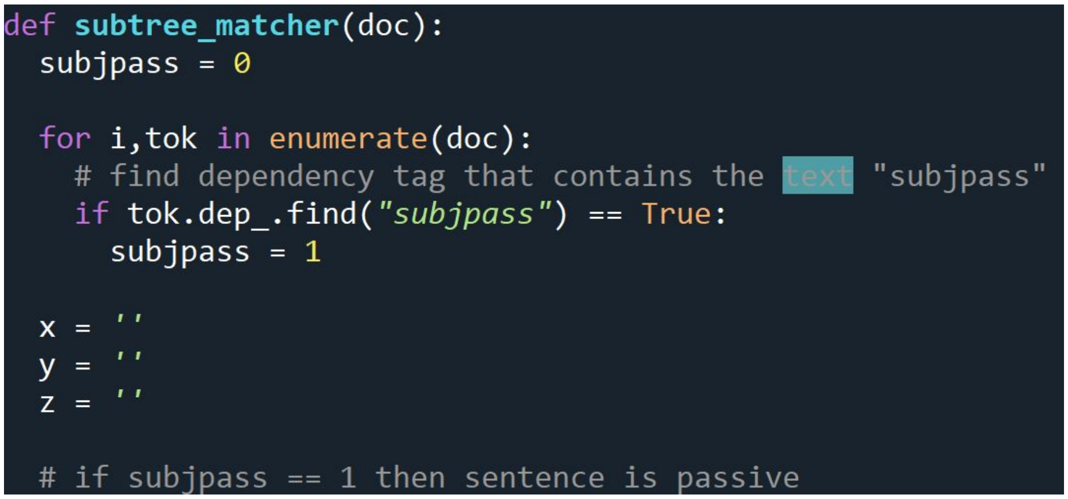
```
for X in doc :
    if(status==0):
        if(X.ent_type_!=''):
            s = s + ' ' + str(X)
        if(X.pos_=='CCONJ'):
            s = s + ' ' + str(X)
    if(X.pos_=='VERB' and status==0):
        status=1
        #link[i].append(X)
        l1 = l1 + ' ' + str(X)
    if(status==1):
        if(X.ent_type_!=''):
            o = o + ' ' + str(X)
            if(X.pos_=='ADJ'):
                adj=1
        elif(adj==1 and (X.pos_=='NOUN'or 'PROPN')):
            o= o +' '+str(X)
            adj=0


sub.append(s)
link.append(l1)
obj.append(o)
print(s+l1+o)
```

For each sentence we are taking all the dependencies and entity types in variable doc .For each word in this doc, firstly we are checking if we have already seen a verb or not. If we haven’t seen a verb then status equals to 0. So when status is 0 and we have seen an entity then we are assuming it as our subject and storing it in (sub). If there is a conjunction followed by this subject then we are including it in (sub). (Assuming that there could be a subject like Jenny and Luci).

If we see a verb then we are changing the status variable to 1 and including this variable into a variable link. From now on when we see entity type in this sentence then we are assuming it as our object for this sentence and storing it in a variable ‘obj’.
 But we are not following this method because it doesn’t work well with passive forms of sentences.

2nd method:

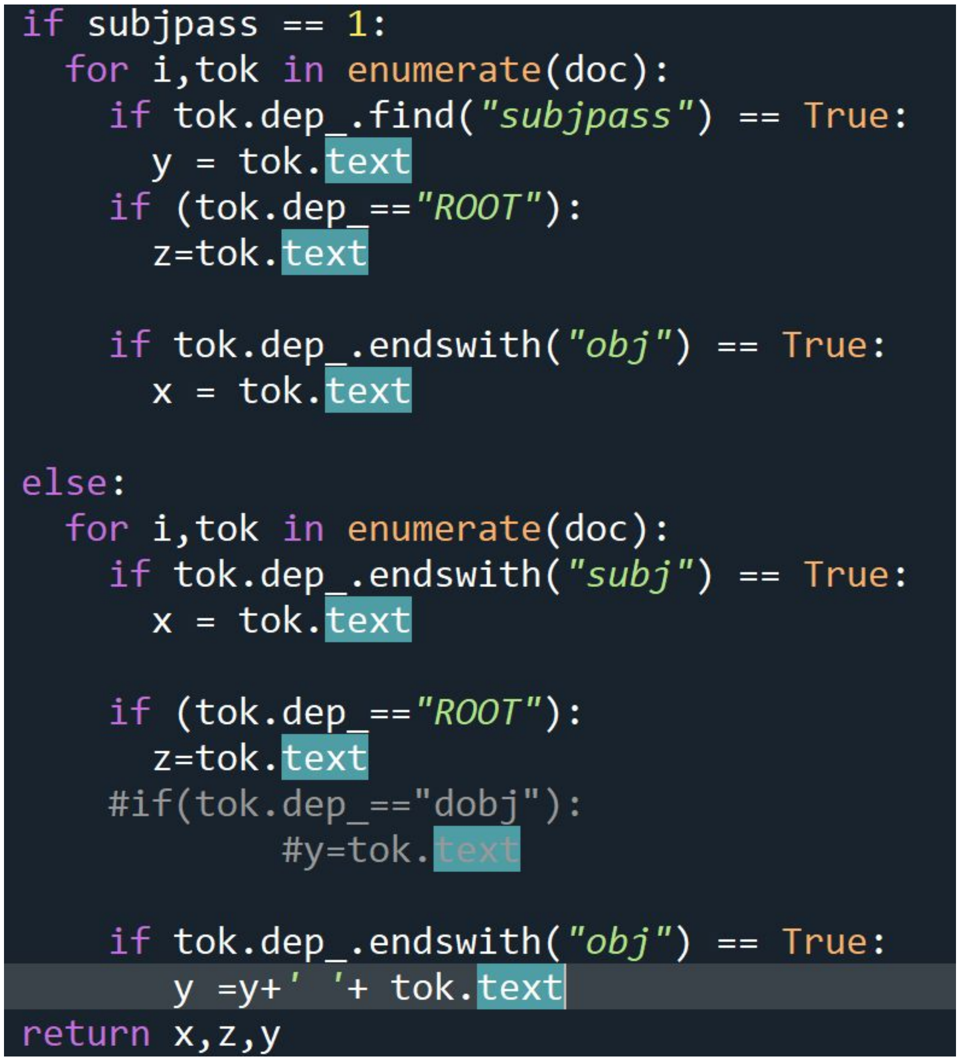
We are following rule based approach again but this time with including word dependencies.



```
def subtree_matcher(doc):
  subjpass = 0

  for i,tok in enumerate(doc):
    # find dependency tag that contains the text "subjpass"
    if tok.dep_.find("subjpass") == True:
      subjpass = 1

  x = ''
  y = ''
  z = ''

  # if subjpass == 1 then sentence is passive
```

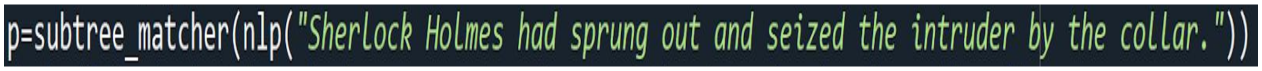
Initially we are taking sentence is in active form and taking “subjpass” variable value as 0 indicating active form.
Now we are checking in the sentence if there is a word tagged with dependency as “subjpass”. If there is any such word then we are changing variable to 1, indicating that it is a passive sentence.

‘x’ is used to store all the subjects, ‘y’ is used to store all the objects and ‘z’ is used to store all the root verbs.



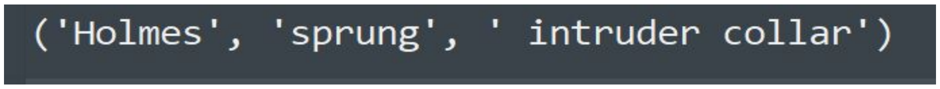
```python
if subjpass == 1:
  for i,tok in enumerate(doc):
    if tok.dep_.find("subjpass") == True:
      y = tok.text
    if (tok.dep_=="ROOT"):
      z=tok.text

    if tok.dep_.endswith("obj") == True:
      x = tok.text

else:
  for i,tok in enumerate(doc):
    if tok.dep_.endswith("subj") == True:
      x = tok.text

    if (tok.dep_=="ROOT"):
      z=tok.text
    #if(tok.dep_=="dobj"):
            #y=tok.text

    if tok.dep_.endswith("obj") == True:
       y =y+' '+ tok.text
return x,z,y
```

For active sentences we are directly taking x as word with dependency tag as 'subj' and z as word with dependency tag as "ROOT" and taking y as a word with dependency tag obj.
 For passive sentences we are taking y i.e obj as word with dependency "subpass, similarly z as word with dependency tag as "ROOT" and taking x i.e subj as a word with dependency tag obj.

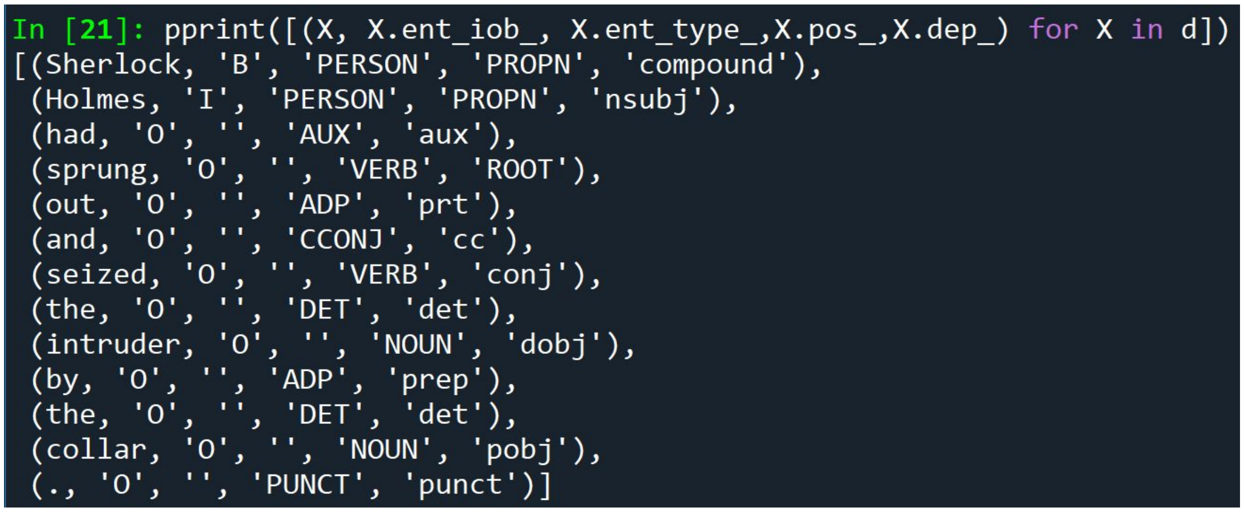


```python
p=subtree_matcher(nlp("Sherlock Holmes had sprung out and seized the intruder by the collar."))
```

We gave one of the sentences in the book1 as an input to the subtree_matcher function which is defined above, the output is as follows:



```
('Holmes', 'sprung', ' intruder collar')
```



```
In [21]: pprint([(X, X.ent_iob_, X.ent_type_,X.pos_,X.dep_) for X in d])
[(Sherlock, 'B', 'PERSON', 'PROPN', 'compound'),
 (Holmes, 'I', 'PERSON', 'PROPN', 'nsubj'),
 (had, 'O', '', 'AUX', 'aux'),
 (sprung, 'O', '', 'VERB', 'ROOT'),
 (out, 'O', '', 'ADP', 'prt'),
 (and, 'O', '', 'CCONJ', 'cc'),
 (seized, 'O', '', 'VERB', 'conj'),
 (the, 'O', '', 'DET', 'det'),
 (intruder, 'O', '', 'NOUN', 'dobj'),
 (by, 'O', '', 'ADP', 'prep'),
 (the, 'O', '', 'DET', 'det'),
 (collar, 'O', '', 'NOUN', 'pobj'),
 (., 'O', '', 'PUNCT', 'punct')]
```

Pprint is prettyprint which gives output in a better understandable way.

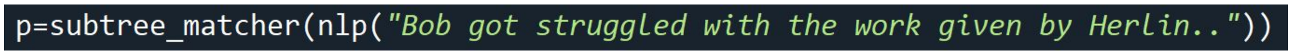
X is the word **X.ent_iob_** gives **(I,O,B)** tags w.r.t the word

X.ent_type_ gives entity type of the word . If word is not an entity then it will be empty.

X.dep_ gives dependency tag of the word

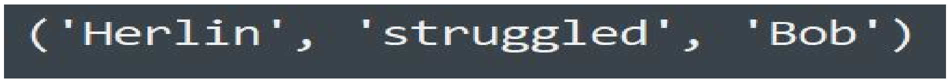
X.pos_ gives pos tag of the word.

An Example of passive sentence from our book2:



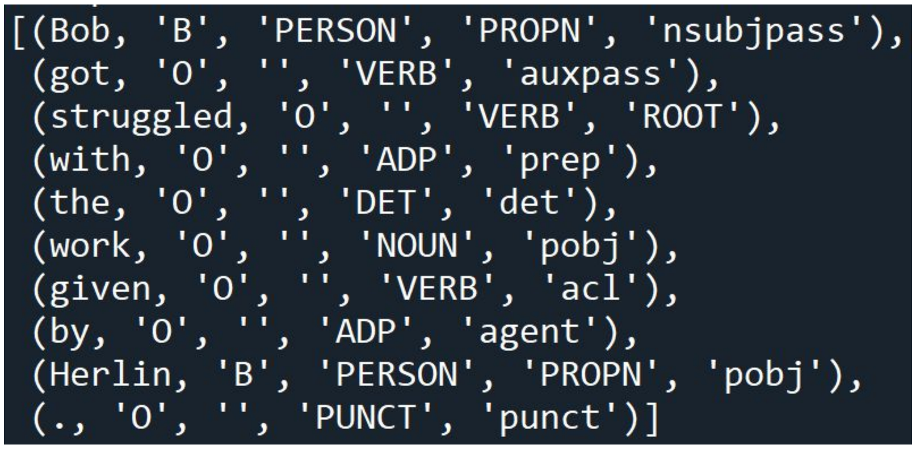
```
p=subtree_matcher(nlp("Bob got struggled with the work given by Herlin.."))
```

Output is obtained in perfect way as we expected as follows:



```
('Herlin', 'struggled', 'Bob')
```

All word_dependencies,IOB tags,entity types in that sentence are in this way:



```
[(Bob, 'B', 'PERSON', 'PROPN', 'nsubjpass'),
 (got, 'O', '', 'VERB', 'auxpass'),
 (struggled, 'O', '', 'VERB', 'ROOT'),
 (with, 'O', '', 'ADP', 'prep'),
 (the, 'O', '', 'DET', 'det'),
 (work, 'O', '', 'NOUN', 'pobj'),
 (given, 'O', '', 'VERB', 'acl'),
 (by, 'O', '', 'ADP', 'agent'),
 (Herlin, 'B', 'PERSON', 'PROPN', 'pobj'),
 (., 'O', '', 'PUNCT', 'punct')]
```

Bob is tagged with '**nsubpass**' word dependency means he's subject in passive sentence. And he got '**Person**' entity tag with 'proper noun' Pos tag.Struggled is recognised as **Root** verb in sentence.

Herlin is also recognised as '**Person**' but with obj dependency tag.

Code:

```
import re
import matplotlib.pyplot as plt
from nltk.stem import PorterStemmer
from nltk.corpus import brown
from nltk.tag.perceptron import PerceptronTagger
from wordcloud import WordCloud                         #all the libraries that are being used
from collections import Counter
from pprint import pprint
import en_core_web_sm
import operator
from nltk.corpus import wordnet as wn
from nltk.tokenize import sent_tokenize, word_tokenize
                                                        #importing the text from text file.
T =  open('nlptext3.txt', encoding = 'utf-8', errors = 'ignore').read()

T_strip = T.strip(' ')                  #strip removes leading and trailing characters based on argument
T = T_strip.replace('\n\n','\n')
T = T.replace('\n\n','\n')
T = T.replace('\t',' ')                 #pre-processing steps

text = re.sub('\*','',T)
text= re.sub('_','',text)
text = text.lower()
text =re.sub(r"chapter","",text)
text=re.sub(r"[^\w][ivx]+\."," ",text )             #removing roman numbers
text=re.sub(r"some disgusted boys","",text )
text=re.sub(r"birds of a feather","",text )         #removing chapter names
text=re.sub(r"lester brigham's idea","",text )
text=re.sub(r"flight and pursuit","",text )
text=re.sub(r"don’s encounter with the tramp","",text )
text=re.sub(r"about various things","",text )
text=re.sub(r"a test of courage","",text )


text = re.sub(r"i'm", "i am", text)
text = re.sub(r"he's", "he is", text)
text = re.sub(r"she's", "she is", text)
text = re.sub(r"it's", "it is", text)
text = re.sub(r"that's", "that is", text)
text = re.sub(r"what's", "what is", text)
text = re.sub(r"where's", "where is", text)
text = re.sub(r"who's","who is",text)
text = re.sub(r"how's","how is",text)
text = re.sub(r"'s","",text)
text = re.sub(r"\'ll", " will", text)
text = re.sub(r"\'re", " are", text)
text = re.sub(r"\'ve", " have", text)
text = re.sub(r"\'re", " are", text)
text = re.sub(r"\'d", " would", text)
text = re.sub(r"won't", "will not", text)
text = re.sub(r"can't", "can not", text)
text = re.sub(r"ain't","am not",text)
text = re.sub(r"\'t"," not",text)

t = re.sub(r"[^\w]", " ", text)
t = re.sub(r"\n" , " ",t)
words = t.split()

char_num=0
for i in words :                          #To find number of characters in text
   char_num+=len(i)

unique_words = {}
for i in words:
   if i not in unique_words:              #extracting unique_words from text with their frequency
      unique_words[i] = 1
   else:
      unique_words[i] +=1

freq_distr = {}
for i,j in unique_words.items():
   if j not in freq_distr:                #frequency of word occurences
      freq_distr[j] = 1
   else:
      freq_distr[j] +=1
```

```
plt.bar(freq_distr.keys(), freq_distr.values(), color='g')
plt.yscale('log')
plt.xscale('log')
plt.show()

ps = PorterStemmer()
s_words={}                              #stemming of words using porterStemmer
for i in words:
   if ps.stem(i) not in s_words :
      s_words[ps.stem(i)]=1
   else :
      s_words[ps.stem(i)]+=1

f_words={}                              #extracting wordlengths and their frequency relations
for i,j in s_words.items() :
   if len(i) not in f_words:
      f_words[len(i)]=j
   else :
      f_words[len(i)]+=j
plt.bar(f_words.keys(), f_words.values(), color='g')
plt.show()

wordcloud = WordCloud(max_words=1200,               #producing a word Cloud
                     background_color = 'white',
                     width = 1200,
                     height = 1000,
                     ).generate_from_frequencies(s_words)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
swords =  open('stop_words.txt', encoding = 'utf-8', errors = 'ignore').read()
swords=swords.replace('\n',' ')
s=swords.split(" ")
nostop_w={}

for i,j in unique_words.items():                    #removing stop-words
   if i not in s:
      nostop_w[i]=j
```

```
wordcloud1 = WordCloud(max_words=1200
                       background_color = 'white',
                       width = 1200,
                       height = 1000,
                           ).generate_from_frequencies(nostop_w)
plt.imshow(wordcloud1)

token = word_tokenize(text)
brown_news_tagged = brown.tagged_sents(categories='news', tagset='universal')
brown_train = brown_news_tagged[100:]          #training-set
brown_test = brown_news_tagged[:100]           #test-set

tagger = PerceptronTagger(load=False)          #declaring tagger
tagger.train(brown_train)                      #and training it over the training set
p=tagger.tag(token)
a=tagger.evaluate(brown_test)                  #evaluating it gave 97.4% accuracy on test set

counts = Counter( tag for word,  tag in p)     #obtaining frequency of POS tags

noun={}
n=0
verb={}
v=0
for i in p:
   if(i[1]=='NOUN'):                           #Extracting all the nouns and verbs from text
      if i[0] not in noun:
         noun[i[0]]=1
         n+=1
      else:
         noun[i[0]]+=1
   if(i[1]=='VERB'):
      if i[0] not in verb:
         verb[i[0]]=1
         v+=1
      else:
        verb[i[0]]+=1
nsense={}
fsense={}
vsense={}
```

```
fvsense={}
for i,j in noun.items() :
   lname={}
   ln='def'
   max=0
   for synset in wn.synsets(i):                  #categorizing nouns based on senses in Wordnet
      l=synset.lexname()
      if l not in lname:
         lname[l] =1
         if(lname[l]>max):
            ln=l
            max=lname[l]

      else:
         lname[l] +=1
         if(lname[l]>max):
            ln=l
            max=lname[l]
      nsense[i]=str(ln)
      if ln not in fsense:
         fsense[ln]=1
      else:
         fsense[ln]+=1

f_sorted=sorted(fsense.items(),key=operator.itemgetter(1),reverse=True)
top10cat={}                                   #sorting the Categories based on their frequency
c=0
top10nam={}
for i in f_sorted :
   if(c<20):                          #Extracting Top20 most occurring ‘Categories’ for graph plotting
     top10cat[c]=i[1]
     top10nam[c]=i[0]
     c+=1
   else:
      break


for i,j in verb.items() :
   lname={}
   ln='def'
   max=0                                   #categorizing verbs based on senses in Wordnet
   for synset in wn.synsets(i):
      l=synset.lexname()
      if l not in lname:
         lname[l] =1
         if(lname[l]>max):
            ln=l
            max=lname[l]

      else:
         lname[l] +=1
         if(lname[l]>max):
            ln=l
            max=lname[l]
      vsense[i]=str(ln)
      if ln not in fvsense:
         fvsense[ln]=1
      else:
         fvsense[ln]+=1

e={}
t = re.sub(r"[^\w]", " ", T)
sent = sent_tokenize(t)
nlp = en_core_web_sm.load()                #Recognising entity types of words
for s in sent:
      doc = nlp(s)
      for X in doc :
         if(X.ent_type_=='PERSON' and X.pos_=='NOUN'):
            if X not in e :
               e[str(X)]=str(X.ent_type_)
         elif(X.ent_type_=='ORG'):
               e[str(X)]=str(X.ent_type_)
         elif(X.ent_type_=='NORP'):
               e[str(X)]=str(X.ent_type_)
         elif(X.ent_type_=='GPE' ):
               e[str(X)]=str(X.ent_type_ )
         elif(X.ent_type_=='LOC' ):
               e[str(X)]=str(X.ent_type_)
```

```
sub=[]
link=[]
obj=[]
for s in sent:
     nlp = en_core_web_sm.load()
    doc = nlp(s)
    #pprint([(X, X.ent_iob_, X.ent_type_,X.pos_) for X in doc])
    s = ''
    i=0

    l1 = ''

    o = ''
    adj=0
    status=0                              #extracting entity relationship **method-1**
    for X in doc :
        if(status==0):
            if(X.ent_type_!=''):          #if entity type is not null and we haven't seen a verb till now then that
                s = s + ' ' + str(X)      # entity is added into subject
            if(X.pos_=='CCONJ'):
                s = s + ' ' + str(X)
        if(X.pos_=='VERB' and status==0):      #after we saw a verb we are making it as link
            status=1                             #between sub and obj
            #link[i].append(X)
            l1 = l1 + ' ' + str(X)
        if(status==1):
            if(X.ent_type_!=''):                 #when we see entity after seeing verb then we add it to object
                o = o + ' ' + str(X)
                if(X.pos_=='ADJ'):
                    adj=1
            elif(adj==1 and (X.pos_=='NOUN'or 'PROPN')):
                o= o +' '+str(X)
                adj=0
    sub.append(s)
    link.append(l1)
    obj.append(o)              #giving out the relationship at the end .But this method won't
print(s+l1+o)                  # work well for passive sentences ...so method-2 has been done.
```

```
def subtree_matcher(doc):                    #method-2 for Entity Relationships extraction
  subjpass = 0

  for i,tok in enumerate(doc):
    # find dependency tag that contains the text "subjpass"
    if tok.dep_.find("subjpass") == True:
      subjpass = 1                             #detecting if a sentence passive or not .
                                               #by searching for a word tagged with subpass dependency
  x = ''
  y = ''
  z = ''                        #if a passive subject is detected then sentence is passive else active

                                # if subjpass == 1 then sentence is passive
  if subjpass == 1:
    for i,tok in enumerate(doc):
      if tok.dep_.find("subjpass") == True:
        y = tok.text
      if (tok.dep_=="ROOT"):          #making subject as object and object as subject if sentence is passive
        z=tok.text                             #keeping link as word tagged with ROOT dependency

      if tok.dep_.endswith("obj") == True:
        x = tok.text

  else:
    for i,tok in enumerate(doc):
      if tok.dep_.endswith("subj") == True:
        x = tok.text                            #if sentence is active then just extract subj,obj and Root verb
      if (tok.dep_=="ROOT"):
        z=tok.text

        if tok.dep_.endswith("obj") == True:
         y =y+' '+ tok.text
  return x,z,y                                 #returning subject(x),Root verb(z) and object(y)
```

Functionalities of various packages included:

We are writing functionalities of packages that are additionally added in project-2,i.e which aren’t described in report-1.

- **from pprint import pprint:** pprint contains a “pretty printer” for producing aesthetically pleasing representations of our data . we are using it mainly at last part of project to print dependencies,pos tags,entity types,iob tags of a word all at the same time in a neat way.

- **import en_core_web_sm:** this model includes the language data, as well as binary weight to enable spaCy to make predictions for part-of-speech tags, dependencies and named entities.This module is mainly used by us for doing additional exercise.

- **import operator:** Operator module is used for multiple level sorting .We are using operator.itemgetter( ) for sorting here .When data is in tuples form and we want to sort it according to one of the values then we use operator module .
  Let’s say a tuple be in this format (x,y) where x represents one type variable and y represents another.
  If we want to sort according to x ,that is first in tuples then we use operator.itemgetter(0) .if want to sort it w.r.t ‘y’ then we use operator.itemgetter(1)

- **from nltk.corpus import wordnet as wn:** WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing. We are using it specifically to obtain synsets and assign categories for nouns and verbs that we extracted from text.

- **from nltk.tokenize import sent_tokenize, word_tokenize:** sent_tokenizer simply divides a text according to sentences i.e each token is a sentence belonging to the text, similarly for word_tokenizer each token is a word belonging to the text.

Conclusion:

Words which are occurring very less number of times are large in number whereas words which are occurring many times are few. Smaller words and larger words are less in number whereas medium sized words are more.Removal of stop words result in a reduction of the huge number of words.Word cloud formed after their removal has a totally different set of words.In our text we can observe from tag frequencies that nouns and verbs are very large in number compared to other tags. Trained tagger always needs to be tested on a different set of sentences other than the sentences on which it is trained. In our case we trained it on ‘brown_sentences[100:]’ and tested it on ‘brown_sentences[:100]’ and accuracy of our tagger came out to be 97.4%.

In both the books, noun.artifact is the most frequent category of nouns.
In the first book there are 3600+ noun.artifacts whereas in second book there are 2754 nouns belonging to artifact category. Most of the frequency rankings of these categories are the same in both books swapping rankings of very few categories. There are more number of entities in book-1(429) compared to book-2(323). For extracting relations between entities we used two methods and both are **Rule Based Approaches**. In the first method we are solely using Pos tag and Entity_type for extracting relations but since we can’t recognize passive sentences through Pos tags alone we followed a second method. In this method we are using dependencies between the words to find if a sentence is passive or not. If the dependency of a word is ‘ROOT’ verb then we are making it as a link which connects subject and object in the sentence. We are also obtaining IOB tags of words in this method.