# NLP Project Round-1

GitHub Link:
https://github.com/mayank-gupta16/Latent-Dirichlet

## Team Name: Latent Dirichlet Members:

Saransh Mittal(20UCS174) Aagam

Jain(20UCS001)

Mayank Gupta(20UCS113)

Mihir Chaturvedi(20UCS117)

**Task:**

1. **Import the text, let's call it as T1**
2. **Perform simple text pre-processing steps**
3. **Tokenise the text T1**

4. **Analyse the frequency distribution of tokens in T1.**
5. **Create a Word Cloud of T1**
6. **Remove the stop words from T1 and then again create a word cloud**
7. **Evaluate the relationship between the word length and frequency for T1**
8. **Do PoS Tagging for T1 using anyone of the four tag sets studied in the class and get the distribution of various tags**

## Library Used

NLTK - Used for tokenizing, lemmatization and removing stopwords.

Language Word Cloud - Used to create word clouds from tokenized data.

Matplotlib- Used to Visualize our text data

Seaborn -Used to Visualize our text data

Numpy- Used for working with arrays.

Typing-To perform evaluation of the Algorithm in Entity Recognition

## IMPORT THE TEXT T1

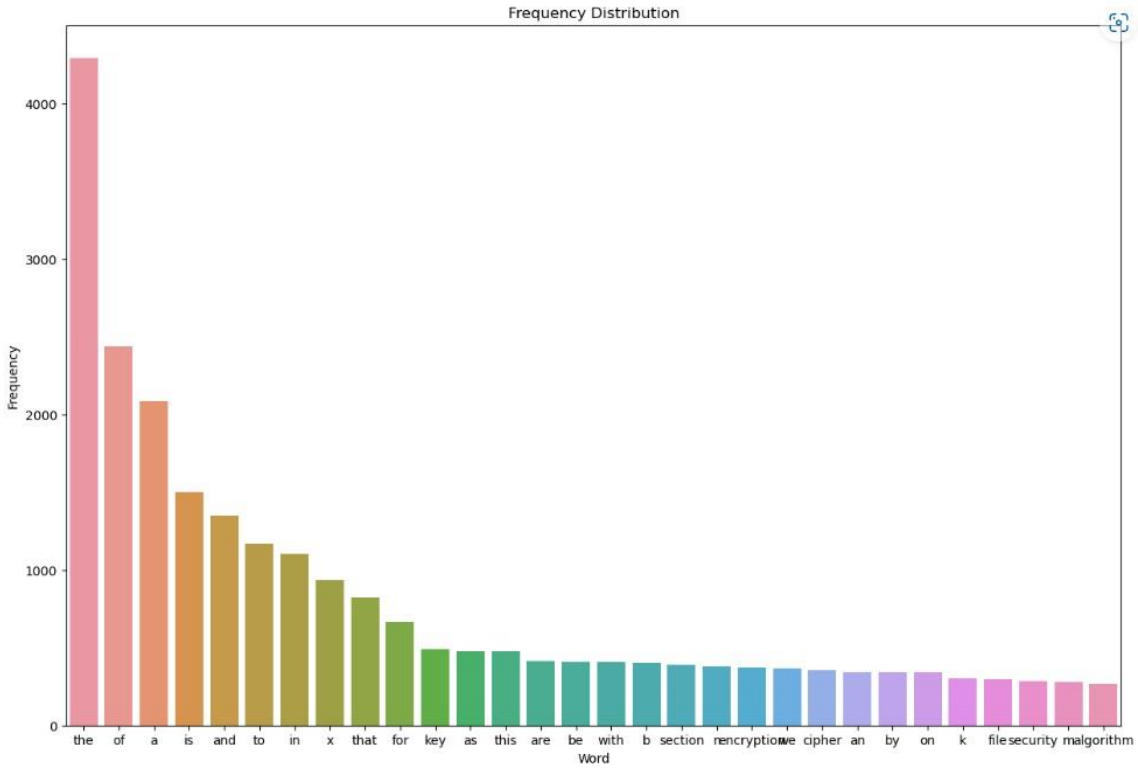
```
# Load data
file = open('T1.txt', encoding='utf-8')
text = file.read()
file.close()
```

### TEXT PREPROCESSING STEPS

- Here we have used word_tokenize of nltk library for splitting the text into words.
- Then we have converted the upper case tokens to lower case tokens .
- Then we removed the punctuations and digits.
- Then we also have removed the remaining tokens which are not alphabetical.
- Then we exclude stop words for which we have used nltk.corpus

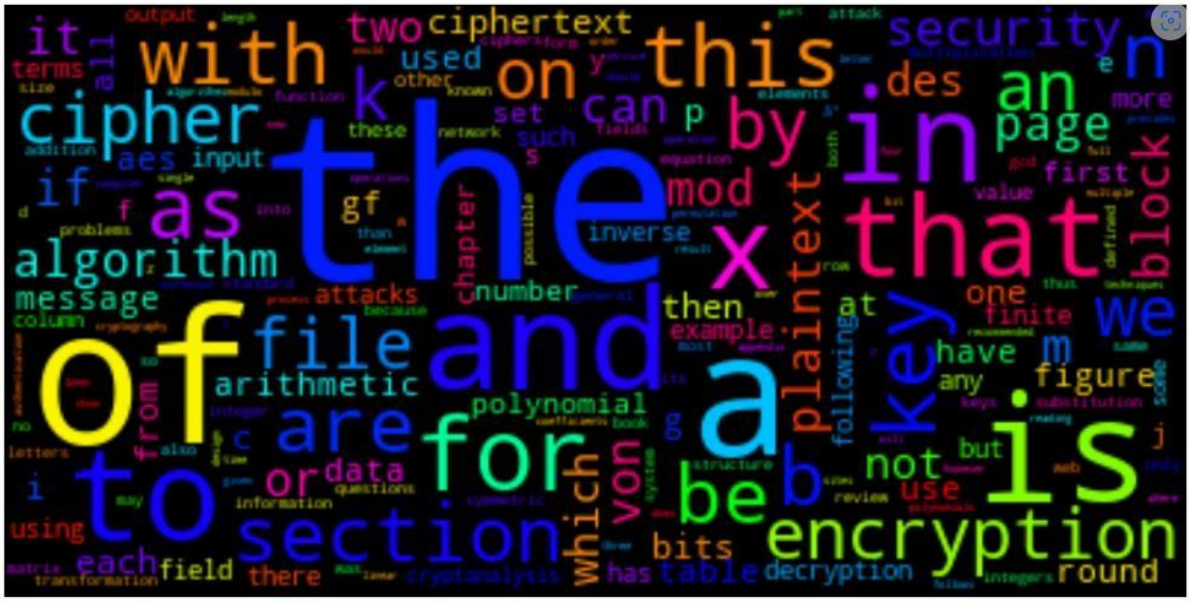
## Frequency Distribution Including stopwords

```
# frequency distribution of tokens in T1 including stopwords
import string
from nltk import FreqDist
from nltk.corpus import stopwords
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import nltk
#nltk.download('all')

# load data
file = open('T1.txt', encoding='utf-8')
text = file.read()
file.close()

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]
## Creating FreqDist  keeping the 30 most common tokens
all_fdist = FreqDist(filtered_text).most_common(30)
all_fdist = pd.Series(dict(all_fdist))
fig, ax = plt.subplots(figsize=(15,10))
plt.title('Frequency Distribution')
plt.ylabel('Frequency')
plt.xlabel('Word')
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```



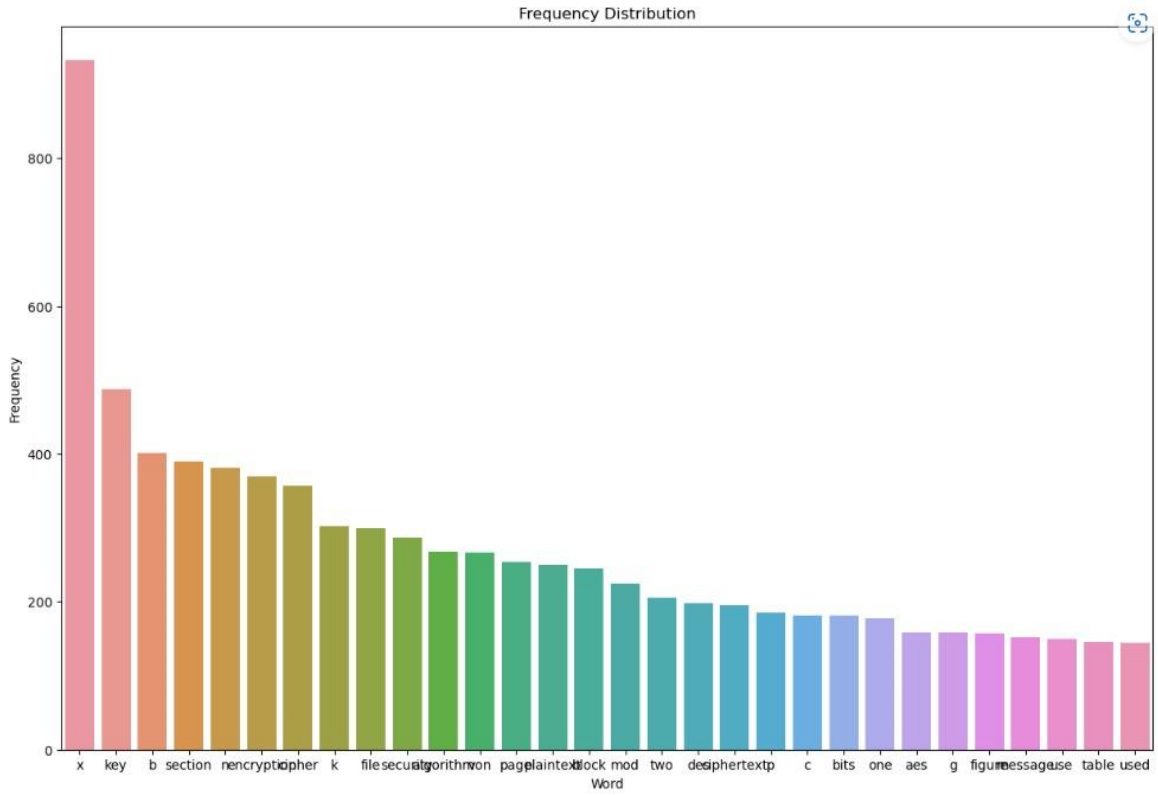

## Word Cloud Including stopwords

```
import string
from nltk import FreqDist
from nltk.corpus import stopwords

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]

from collections import Counter
dictionary=Counter(filtered_text)
import matplotlib.pyplot as plt
from wordcloud import WordCloud

cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictionary)
plt.figure(figsize=(16,12))
plt.imshow(cloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```




## Frequency Distribution excluding stopwords

```
import string
from nltk import FreqDist
from nltk.corpus import stopwords
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]
## Creating FreqDist  keeping the 30 most common tokens
all_fdist = FreqDist(filtered_text).most_common(30)
all_fdist = pd.Series(dict(all_fdist))
fig, ax = plt.subplots(figsize=(15,10))
plt.title('Frequency Distribution')
plt.ylabel('Frequency')
plt.xlabel('Word')
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```




## Word Cloud Exluding stopwords

```
import string
from nltk import FreqDist
from nltk.corpus import stopwords

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]

from collections import Counter
dictionary=Counter(filtered_text)
import matplotlib.pyplot as plt
from wordcloud import WordCloud

cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictionary)
plt.figure(figsize=(16,12))
plt.imshow(cloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```


## Inferences:

- Words like ‘of’, ‘a’ and ‘the’ are the most frequently used words in T1
- Words like ‘of’, ‘and’, ‘a’, and ‘the’ are frequently used words in T2
- These words do not contribute to the meaning of the sentence and are mostly useless for us

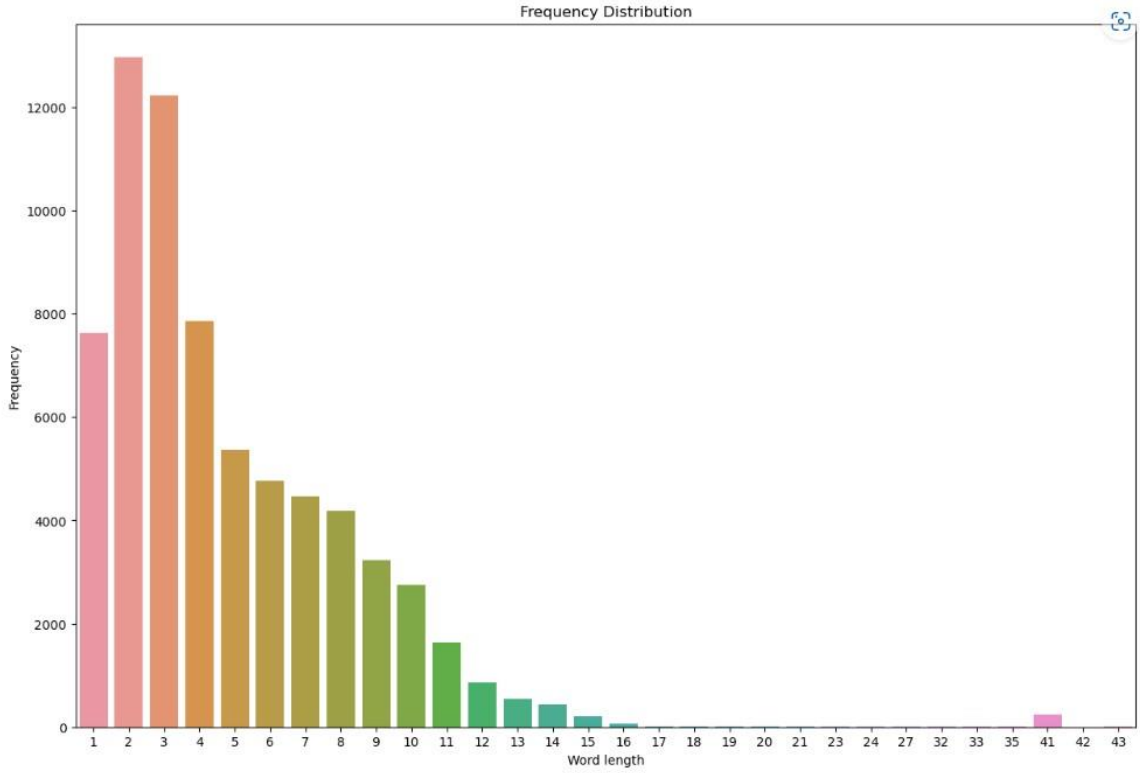
These words are known as ‘stopwords’, and we need to get rid of them

On excluding stop words, we get a meaningful word cloud as stop words mostly have higher frequency due to which our result might get affected. Now most of the terms that are of no meaning to us have been removed. We have gotten rid of ‘stopwords’ and can draw meaningful conclusions from the data.

## Relation between word length and frequency for Text

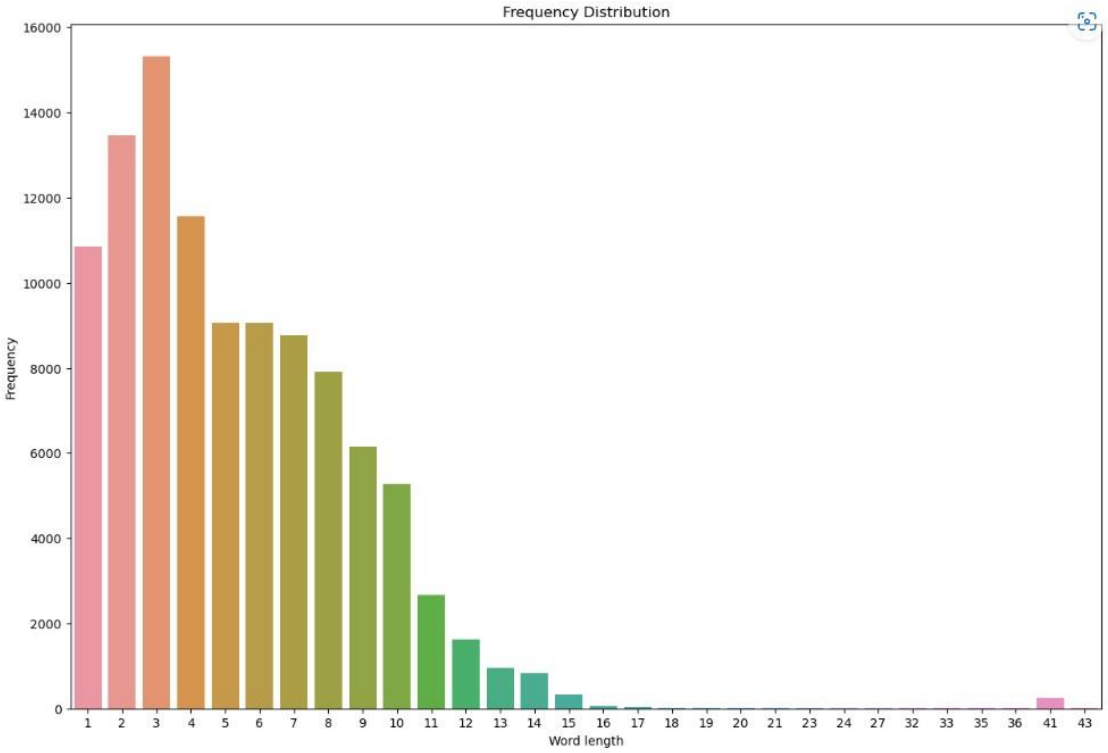
Including Stopwords:

```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

words = text.split()
lengths = {}
for word in words:
    length = len(word)
    if length not in lengths:
        lengths[length] = 1
    else:
        lengths[length] += 1
## Creating FreqDist  keeping the 30 most common tokens
all_fdist = FreqDist(lengths).most_common(30)
all_fdist = pd.Series(dict(all_fdist))
fig, ax = plt.subplots(figsize=(15,10))
plt.title('Frequency Distribution')
plt.ylabel('Frequency')
plt.xlabel('Word length')
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```




Excluding StopWords:

```
import string
from nltk import FreqDist
from nltk.corpus import stopwords
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set(stopwords.words('english') + list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
words = [word for word in filtered_text if word.isalpha()]

for word in words:
    length = len(word)
    if length not in lengths:
        lengths[length] = 1
    else:
        lengths[length] += 1
## Creating FreqDist  keeping the 30 most common tokens
all_fdist = FreqDist(lengths).most_common(30)
all_fdist = pd.Series(dict(all_fdist))
fig, ax = plt.subplots(figsize=(15,10))
plt.title('Frequency Distribution')
plt.ylabel('Frequency')
plt.xlabel('Word length')
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
```




## Inferences:

- The number of words of length 1 has significantly increased after removing stopwords as now it will also count varialbles like ‘x’ in it.
- There is a general trend that the highest number of words lies in the length range 3-6.
- We can infer that this is due to removing stop words like ‘a’, ‘the’, ‘of’ and ‘and’, which were the highest occurring words before removal.

- **POS TAGGING**
- Now the main task is to give each of the tokens their tags. We have used
- ‘averaged_perceptron_tagger’ for POS tagging of the tickets. We have downloaded the same from nltk.

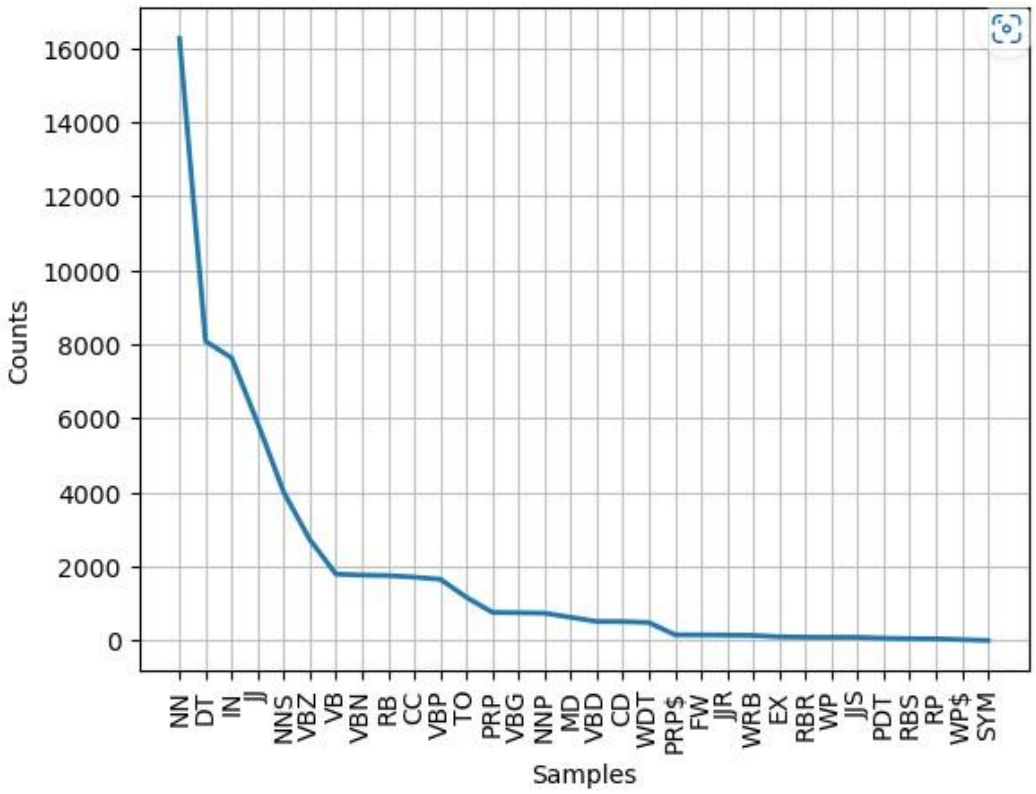
```
from nltk.corpus import stopwords

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
token1 = [word for word in filtered_text if word.isalpha()]

tagged1=nltk.pos_tag(token1)
tagged1
```

```
[('cryptography', 'NN'),
 ('and', 'CC'),
 ('network', 'NN'),
 ('security', 'NN'),
 ('principles', 'NNS'),
 ('and', 'CC'),
 ('practices', 'NNS'),
 ('fourth', 'JJ'),
 ('edition', 'NN'),
 ('cryptography', 'NN'),
 ('and', 'CC'),
 ('network', 'NN'),
 ('security', 'NN'),
 ('principles', 'NNS'),
 ('and', 'CC'),
 ('practices', 'NNS'),
 ('fourth', 'JJ'),
 ('edition', 'NN'),
 ('by', 'IN'),
 ('william', 'JJ'),
 ('stallings', 'NNS'),
 ('publisher', 'NN'),
 ('prentice', 'NN'),
 ('hall', 'NN'),
 ('pub', 'NN'),
 ('date', 'NN'),
 ('november', 'JJ'),
 ('print', 'NN'),
 ('print', 'NN'),
 ('etext', 'RBR'),
 ('table', 'NN'),
 ('of', 'IN'),
 ('contents', 'NNS'),
 ('etext', 'JJ'),
```

Frequency Distribution of Tags:

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

```
FrequencyDist(tagged1)
```




## Inferences

From the above results we infer that the highest occurring tag is 'NN', and 'Determinant' Tags are on the lower frequency side. This is largely due to the removal of stopwords before POS Tagging.

## Conclusion:

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

# NLP PROJECT
# ROUND-2

## Tasks

First Part:

1. Find the nouns and verbs in both the novels. Get the immediate categories (parent) that these words fall under in the WordNet.
2. Get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same for each novel

Second Part:

1. Recognise all persons, locations, organisations in the book.

For this, you have to do two steps:

(1) First, recognise all the entity

(2) Recognise all entity types. Use performance measures to measure the performance of the method used. For evaluation, you take a considerable amount of random passages from the novel, do manual labelling and then compare your result with it.

Present the accuracy with the F1 score here.

Third Part:

1. For extracting the relationship between the entities from the book - what are the features necessary for this? Use the ideas given in the book and presented in the class to augment the data of Entities and augment that data by extracting additional features and build the table and present it.

## Library Used

NLTK - Used for tokenizing, lemmatization and removing stopwords.

Language Word Cloud - Used to create word clouds from tokenized data.

Matplotlib- Used to Visualize our text data

Seaborn -Used to Visualize our text data

Numpy- Used for working with arrays.

Typing-To perform evaluation of the Algorithm in Entity Recognition

# Problem Statement And Inferences:

## First part:

Finding NOUNS and VERBS

- Performing POS Tagging

We will now perform the POS Tagging on T1 and T2 using the inbuilt function of nltk

namely pos_tag() which uses Penn Treebank tag set to perform POS tagging.

We will extract the words tagged explicitly as nouns and verbs separately from both the novels using the following code.

```
from nltk.tokenize import word_tokenize
import string
from nltk.corpus import stopwords
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import nltk
from textblob import TextBlob
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import re
import inflect
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from nltk import FreqDist
from collections import Counter
#nltk.download('all')

# load data
file = open('T1.txt', encoding='utf-8')
text = file.read()
file.close()

# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
# convert to lower case
tokens = [w.lower() for w in tokens]
remove_these = set( list(string.punctuation) + list(string.digits))
filtered_text = [w for w in tokens if not w in remove_these]
# remove remaining tokens that are not alphabetic
filtered_text = [word for word in filtered_text if word.isalpha()]
```

```
# function to test if something is a noun
is_noun = lambda pos: pos[:2] == 'NN'
noun1 = [word for (word, pos) in nltk.pos_tag(filtered_text) if is_noun(pos)]
noun1
```

```
['cryptography',
 'network',
 'security',
 'principles',
 'practices',
 'edition',
 'cryptography',
 'network',
 'security',
 'principles',
 'practices',
 'edition',
 'stallings',
 'publisher',
 'prentice',
 'hall',
 'pub',
 'date',
 'print',
```

```
print("Number of nouns in book "+ str(len(noun1)))
```

```
Number of nouns in book 20991
```

```
is_verb = lambda pos: pos[:1] == 'V'

verb1 = [word for (word, pos) in nltk.pos_tag(filtered_text) if is_verb(pos)]
print("Number of verbs in book are "+ str(len(verb1)))
```

```
Number of verbs in book are 9204
```

- *Get the categories that these words fall under in the WordNet.*

To retrieve the categories that each noun and verb belong to, in the wordnet synsets, we have used nltk.corpus.wordnet as it has all the tools required for this task. We have used the following function to extract categories each noun and verb belongs to. Since a noun also has synsets interpretations as verbs and vice versa hence we have included them as lists corresponding to its index in the noun and verb lists respectively.

```
from nltk.corpus import wordnet as wn
def synset(words):
  categories=[]
  for word in words:
    cat=[]
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        cat.append(synset.lexname())
      if('verb' in synset.lexname()):
        cat.append(synset.lexname())
    categories.append(cat)
  return categories
```

We will apply the above function to get 2-D lists which contain the categories that each noun and verb have been defined in the wordnet database.

```
noun_syn1=synset(noun1)
verb_syn1=synset(verb1)
```

Hence noun_syn1, verbsyn_1 are two-dimensional. We list the categories that noun1, , verb1 belong to in the wordnet synsets of nouns and verbs.

The 2d lists are indexed as noun_syn1[x][y], where x is the index of the corresponding nouns in noun1 and y is the index containing the categories it belongs to.

- Get the frequency of each category for each noun and verb in their corresponding and plot histogram/bar plots for each corresponding type.

To get the frequency of all the categories of nouns and verbs in the novels, we have created a set for each book that contains all the types of nouns and verbs occurring and then plotted the frequency distribution for the data.

```
def all_synsets(no,ve):
  nouns=[]
  verbs=[]
  for word in no:
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        nouns.append(synset.lexname())
      if('verb' in synset.lexname()):
        verbs.append(synset.lexname())
  for word in ve:
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        nouns.append(synset.lexname())
      if('verb' in synset.lexname()):
        verbs.append(synset.lexname())

  return nouns,verbs
```

```
noun_superset1,verb_superset1=all_synsets(noun1,verb1)
```

*Here noun_superset1 contains all the different categories of nouns and verb_superset1 has all the types of verbs.* Eg.

```
print(noun_superset1)
```

```
n', 'noun.cognition', 'noun.artifact', 'noun.location', 'noun.shape', 'noun.group', 'noun.artifact', 'noun.quantity', 'n
oun.cognition', 'noun.artifact', 'noun.artifact', 'noun.artifact', 'noun.artifact', 'noun.act', 'noun.person', 'noun.ani
mal', 'noun.person', 'noun.act', 'noun.quantity', 'noun.food', 'noun.location', 'noun.artifact', 'noun.state', 'noun.ac
t', 'noun.artifact', 'noun.act', 'noun.artifact', 'noun.act', 'noun.cognition', 'noun.phenomenon', 'noun.artifact', 'nou
n.artifact', 'noun.quantity', 'noun.time', 'noun.substance', 'noun.substance', 'noun.substance', 'noun.substance', 'nou
n.quantity', 'noun.quantity', 'noun.communication', 'noun.communication', 'noun.communication', 'noun.artifact', 'noun.s
ubstance', 'noun.quantity', 'noun.communication', 'noun.quantity', 'noun.time', 'noun.substance', 'noun.substance', 'nou
n.substance', 'noun.substance', 'noun.quantity', 'noun.quantity', 'noun.communication', 'noun.communication', 'noun.comm
unication', 'noun.artifact', 'noun.quantity', 'noun.time', 'noun.substance', 'noun.substance', 'noun.substance', 'noun.s
ubstance', 'noun.quantity', 'noun.quantity', 'noun.communication', 'noun.communication', 'noun.communication', 'noun.art
ifact', 'noun.food', 'noun.substance', 'noun.substance', 'noun.substance', 'noun.substance', 'noun.act', 'noun.act', 'no
un.person', 'noun.person', 'noun.person', 'noun.act', 'noun.location', 'noun.quantity', 'noun.substance', 'noun.quantit
y', 'noun.quantity', 'noun.quantity', 'noun.communication', 'noun.artifact', 'noun.act', 'noun.attribute', 'noun.attribu
te', 'noun.process', 'noun.act', 'noun.act', 'noun.attribute', 'noun.substance', 'noun.communication', 'noun.substance',
'noun.quantity', 'noun.communication', 'noun.act', 'noun.substance', 'noun.communication', 'noun.substance', 'noun.quant
ity', 'noun.communication', 'noun.act', 'noun.artifact', 'noun.possession', 'noun.cognition', 'noun.event', 'noun.relati
on', 'noun.act', 'noun.phenomenon', 'noun.possession', 'noun.artifact', 'noun.artifact', 'noun.possession', 'noun.posses
sion', 'noun.attribute', 'noun.substance', 'noun.location', 'noun.substance', 'noun.attribute', 'noun.state', 'noun.rela
tion', 'noun.relation', 'noun.communication', 'noun.cognition', 'noun.act', 'noun.attribute', 'noun.attribute', 'noun.pr
ocess', 'noun.act', 'noun.act', 'noun.attribute', 'noun.location', 'noun.person', 'noun.act', 'noun.attribute', 'noun.st
```

```
len(noun_superset1)
```

```
124783
```

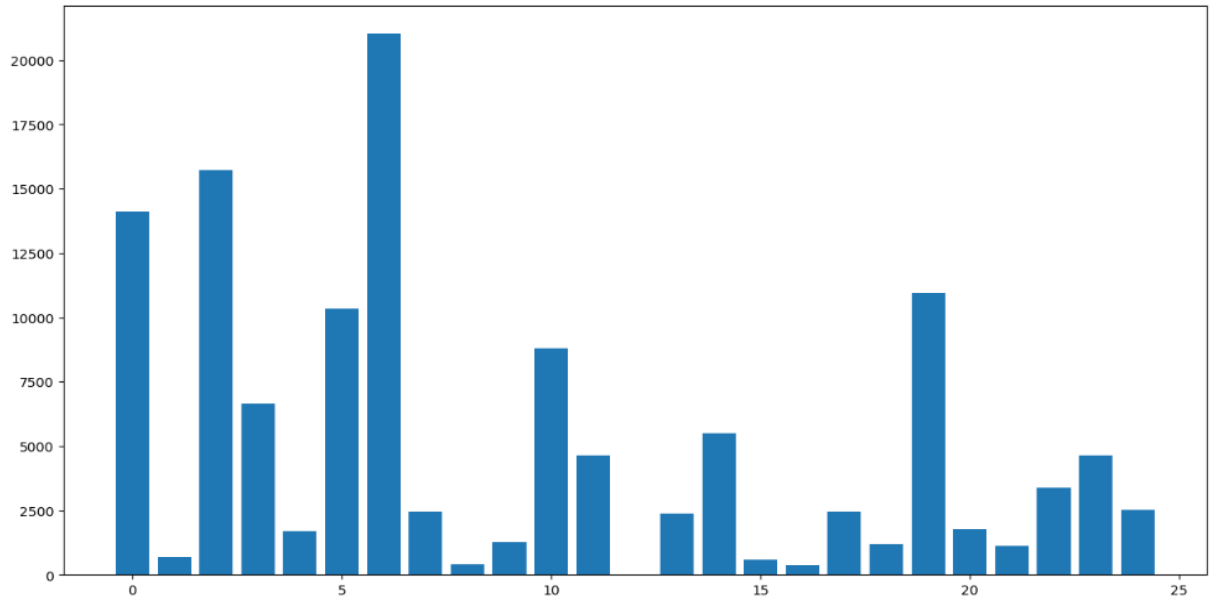
Hence there are 124783 elements in the list.

## Plotting the histograms:

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

```
import numpy as np
labels, counts = np.unique(noun_superset1,return_counts=True)
import matplotlib.pyplot as plt
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
```

```
<BarContainer object of 25 artists>
```



```
print(labels)
```

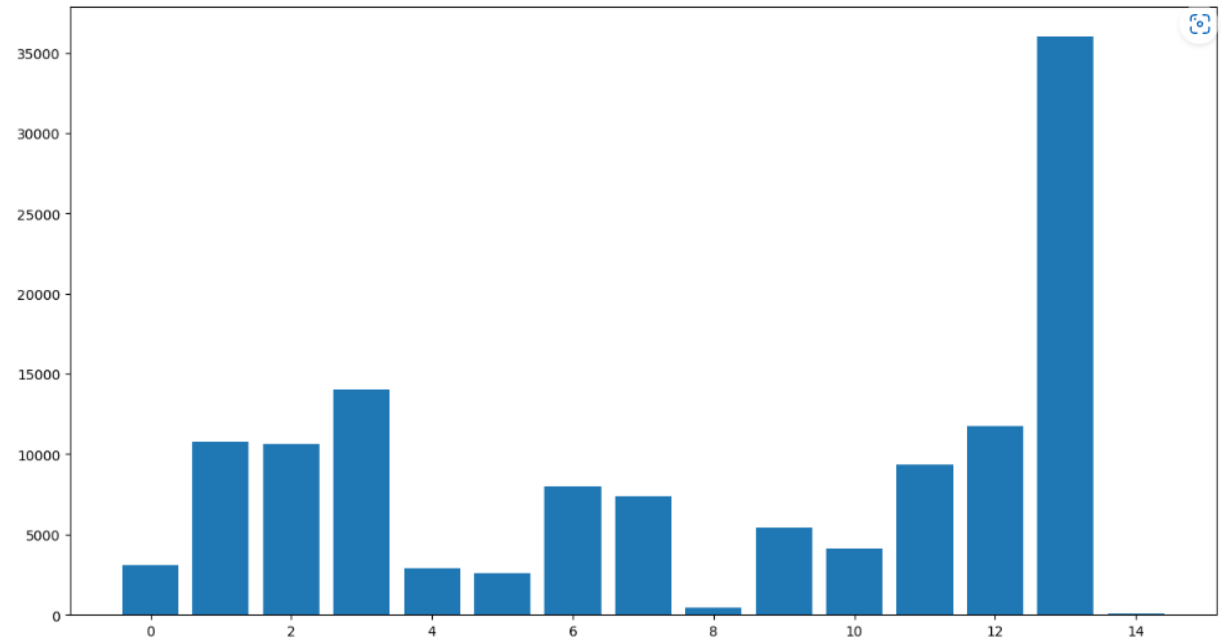
```
['noun.act' 'noun.animal' 'noun.artifact' 'noun.attribute' 'noun.body'
 'noun.cognition' 'noun.communication' 'noun.event' 'noun.feeling'
 'noun.food' 'noun.group' 'noun.location' 'noun.motive' 'noun.object'
 'noun.person' 'noun.phenomenon' 'noun.plant' 'noun.possession'
 'noun.process' 'noun.quantity' 'noun.relation' 'noun.shape' 'noun.state'
 'noun.substance' 'noun.time']
```

Here labels are arranged in the order of hierarchy as given in wordnet categories of nouns.

Similarly, we get the following plots for Verbs:

```
labels, counts = np.unique(verb_superset1,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
```

```
<BarContainer object of 15 artists>
```




labels are numbered as 0-14 in the following order:

```
print(labels)
```

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

## Second Part:

### Named Entity Recognition

- Get the entities involved in each of the novels.

To perform Named Entity Recognition in both novels, we have used Spacy.
SpaCy named entity recognition supports the following entity types:

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

The spacy uses token level entity annotation using the BILUO tagging scheme to describe the entity boundaries.

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

First, we import spacy and get the entities using the methods described in the library.

```
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
spacy.load('en_core_web_sm')
nlp = en_core_web_sm.load()
T1_text=' '.join(filtered_text)
doc1 = nlp(T1_text)
print("there are total "+str(len(doc1.ents))+" entities in book")
```

```
there are total 1142 entities in book
```

```
T1_text
```

```
'cryptography and network security principles and practices fourth edition cryptography and network security principles
and practices fourth edition by william stallings publisher prentice hall pub date november print print etext table of c
ontents etext index pages in this age of viruses and hackers of electronic eavesdropping and electronic fraud security i
s paramount as the disciplines of cryptography and network security have matured more practical readily available applic
ations to enforce network security have developed this text provides a practical survey of both the principles and pract
ice of cryptography and network security first the basic issues to be addressed by a network security capability are exp
lored through a tutorial and survey of cryptography and network security technology then the practice of network securit
y is explored via practical applications that have been implemented and are in use today file table of contents cryptogr
aphy and network security principles and practices fourth edition by william stallings publisher prentice hall pub date
november print print etext table of contents etext index pages copyright notation xi preface xiii objectives xiii intend
ed audience xiii plan of the book xiv internet services for instructors and students xiv projects for teaching cryptogra
phy and network security xiv what new in the fourth edition xv acknowledgments xvi chapter reader guide section outline
of this book section roadmap section internet and web resources chapter introduction section security trends section the
osi security architecture section security attacks section security services section security mechanisms section a model
for network security section recommended reading and web sites section key terms review questions and problems part one
symmetric ciphers chapter classical encryption techniques section symmetric cipher model file von table of contents sect
ion substitution techniques section transposition techniques section rotor machines section steganography section recomm
ended reading and web sites section key terms review questions and problems chapter block ciphers and the data encryptio
n standard section block cipher principles section the data encryption standard section the strength of des section diff
```

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

```
[(cryptography, 'O'), (and, 'O'), (network, 'O'), (security, 'O'), (principles, 'O'), (and, 'O'), (practices, 'O'), (fou
rth, 'B'), (edition, 'O'), (cryptography, 'O'), (and, 'O'), (network, 'O'), (security, 'O'), (principles, 'O'), (and,
'O'), (practices, 'O'), (fourth, 'B'), (edition, 'O'), (by, 'O'), (william, 'B'), (stallings, 'I'), (publisher, 'O'), (p
rentice, 'B'), (hall, 'I'), (pub, 'I'), (date, 'O'), (november, 'O'), (print, 'O'), (print, 'O'), (etext, 'O'), (table,
'O'), (of, 'O'), (contents, 'O'), (etext, 'O'), (index, 'O'), (pages, 'O'), (in, 'O'), (this, 'O'), (age, 'O'), (of,
'O'), (viruses, 'O'), (and, 'O'), (hackers, 'O'), (of, 'O'), (electronic, 'O'), (eavesdropping, 'O'), (and, 'O'), (elect
ronic, 'O'), (fraud, 'O'), (security, 'O'), (is, 'O'), (paramount, 'O'), (as, 'O'), (the, 'O'), (disciplines, 'O'), (of,
'O'), (cryptography, 'O'), (and, 'O'), (network, 'O'), (security, 'O'), (have, 'O'), (matured, 'O'), (more, 'O'), (pract
ical, 'O'), (readily, 'O'), (available, 'O'), (applications, 'O'), (to, 'O'), (enforce, 'O'), (network, 'O'), (security,
'O'), (have, 'O'), (developed, 'O'), (this, 'O'), (text, 'O'), (provides, 'O'), (a, 'O'), (practical, 'O'), (survey,
'O'), (of, 'O'), (both, 'O'), (the, 'O'), (principles, 'O'), (and, 'O'), (practice, 'O'), (of, 'O'), (cryptography,
'O'), (and, 'O'), (network, 'O'), (security, 'O'), (first, 'B'), (the, 'O'), (basic, 'O'), (issues, 'O'), (to, 'O'), (b
e, 'O'), (addressed, 'O'), (by, 'O'), (a, 'O'), (network, 'O'), (security, 'O'), (capability, 'O'), (are, 'O'), (explore
d, 'O'), (through, 'O'), (a, 'O'), (tutorial, 'O'), (and, 'O'), (survey, 'O'), (of, 'O'), (cryptography, 'O'), (and,
'O'), (network, 'O'), (security, 'O'), (technology, 'O'), (then, 'O'), (the, 'O'), (practice, 'O'), (of, 'O'), (network,
'O'), (security, 'O'), (is, 'O'), (explored, 'O'), (via, 'O'), (practical, 'O'), (applications, 'O'), (that, 'O'), (hav
e, 'O'), (been, 'O'), (implemented, 'O'), (and, 'O'), (are, 'O'), (in, 'O'), (use, 'O'), (today, 'B'), (file, 'O'), (tab
le, 'O'), (of, 'O'), (contents, 'O'), (cryptography, 'O'), (and, 'O'), (network, 'O'), (security, 'O'), (principles,
'O'), (and, 'O'), (practices, 'O'), (fourth, 'B'), (edition, 'O'), (by, 'O'), (william, 'B'), (stallings, 'I'), (publish
```

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

- Get Spacy's annotated entities: person,organization, and Location.

We have used the following function on the entities returned by spacy to collect the Person, Organization and Location entities. The ent_type function returns the type of entity annotated by Spacy and returns lists containing the above types of entities.

```
def entity_recognition(text):
  doc=nlp(text)
  person=[]
  org=[]
  location=[]
  for X in doc:
    if (X.ent_type_=='PERSON') and X.text not in person:
      person.append(X.text)
    if (X.ent_type_=='ORG')and X.text not in org:
      org.append(X.text)
    if ((X.ent_type_=='LOC') or (X.ent_type_=='GPE')) and X.text not in location:
      location.append(X.text)
  return person,org,location
```

Now collecting these entities :

```
person1,org1,location1=entity_recognition(T1_text)
print("number of person entities in book  are "+str(len(person1)))
print("number of organization entities in book are "+str(len(org1)))
print("number of location entities in book are "+str(len(location1)))
```

```
number of person entities in book  are 160
number of organization entities in book are 151
number of location entities in book are 32
```

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

```
def freq(str_list):
    unique_words = set(str_list)
    counts = {}
    for words in unique_words :
        counts[words] = str_list.count(words)
    return counts
```

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

```
[('zicvtwqngkzeiigasxstslvvwla', 1), ('zealand', 1), ('z', 1), ('y', 1), ('xiv', 1), ('xcsma', 1), ('x', 1), ('wesleyan', 1), ('watermarking', 1), ('washington', 1), ('w', 1), ('von', 1), ('v', 1), ('upper', 1), ('university', 1), ('u', 1), ('transformation', 1), ('toga', 1), ('thus', 1), ('therefore', 1), ('then', 1), ('the', 1), ('terada', 1), ('technology', 1), ('substitution', 1), ('subnib', 1), ('state', 1), ('standards', 1), ('standard', 1), ('sqgkc', 1), ('sqapu', 1), ('signal', 1), ('set', 1), ('senate', 1), ('section', 1), ('schulzrinne', 1), ('santa', 1), ('saddle', 1), ('rsa', 1), ('routo', 1), ('river', 1), ('representatives', 1), ('reinhold', 1), ('rei', 1), ('purdue', 1), ('pub', 1), ('proceeds', 1), ('prentice', 1), ('pqtyj', 1), ('playfair', 1), ('pearson', 1), ('party', 1), ('pairs', 1), ('p', 1), ('oregon', 1), ('one', 1), ('officer', 1), ('of', 1), ('nyberg', 1), ('nsa', 1), ('nist', 1), ('nibbles', 1), ('new', 1), ('navy', 1), ('national', 1), ('n', 1), ('muir', 1), ('monoalphabetic', 1), ('mod', 1), ('mixcolumns', 1), ('mexico', 1), ('medeiros', 1), ('mathematical', 1), ('m', 1), ('london', 1), ('logon', 1), ('lloyd', 1), ('linear', 1), ('kxjey', 1), ('korner', 1), ('kodak', 1), ('koc', 1), ('key', 1), ('kemal', 1), ('k', 1), ('james', 1), ('isr', 1), ('ipsec', 1), ('institution', 1), ('institute', 1), ('ins', 1), ('information', 1), ('inc', 1), ('identities', 1), ('ibm', 1), ('house', 1), ('hartford', 1), ('hall', 1), ('group', 1), ('gf', 1), ('george', 1), ('gcd', 1), ('ga', 1), ('g', 1), ('florida', 1), ('finite', 1), ('financial', 1), ('file', 1), ('field', 1), ('fed', 1), ('euclidean', 1), ('etsqu', 1), ('equation', 1), ('england', 1), ('el', 1), ('educationjapan', 1), ('education', 1), ('digital', 1), ('designated', 1), ('demonstrate', 1), ('dea', 1), ('de', 1), ('d', 1), ('cryptogram', 1), ('cristina', 1), ('corp', 1), ('continue', 1), ('congress', 1), ('columbia', 1), ('code', 1), ('clara', 1), ('cambridge', 1), ('c', 1), ('bureau', 1), ('breno', 1), ('boston', 1), ('bits', 1), ('bin', 1), ('bicakci', 1), ('barrie', 1), ('association', 1), ('artech', 1), ('army', 1), ('an', 1), ('american', 1), ('america', 1), ('amateur', 1), ('algorithm', 1), ('aes', 1), ('additive', 1), ('ac', 1)]
```

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

```
[('y', 1), ('xv', 1), ('xor', 1), ('xiv', 1), ('x', 1), ('wimsey', 1), ('william', 1), ('waterloo', 1), ('wang', 1), ('walter', 1), ('w', 1), ('von', 1), ('vernam', 1), ('van', 1), ('v', 1), ('university', 1), ('u', 1), ('tzu', 1), ('tuchman', 1), ('trentacoste', 1), ('tracy', 1), ('tom', 1), ('sun', 1), ('stiglic', 1), ('stallings', 1), ('smith', 1), ('singh', 1), ('simon', 1), ('shamir', 1), ('scribner', 1), ('schaefer', 1), ('sarah', 1), ('sanjay', 1), ('s', 1), ('rotnib', 1), ('roger', 1), ('robin', 1), ('richard', 1), ('re', 1), ('rcon', 1), ('rao', 1), ('qzjka', 1), ('quinn', 1), ('pu', 1), ('prentice', 1), ('philip', 1), ('ph', 1), ('peter', 1), ('paul', 1), ('patricia', 1), ('parker', 1), ('page', 1), ('outerbridge', 1), ('nichols', 1), ('nicholas', 1), ('nakahara', 1), ('n', 1), ('muir', 1), ('moseley', 1), ('modulo', 1), ('mod', 1), ('meyer', 1), ('mcgibney', 1), ('mcdowell', 1), ('max', 1), ('mauborgne', 1), ('mason', 1), ('marketing', 1), ('marcia', 1), ('mac', 1), ('ma', 1), ('m', 1), ('luiz', 1), ('lucas', 1), ('lisa', 1), ('lidl', 1), ('lee', 1), ('l', 1), ('kumar', 1), ('krizanc', 1), ('kris', 1), ('kk', 1), ('kim', 1), ('keys', 1), ('key', 1), ('kernan', 1), ('kenselaar', 1), ('kennedy', 1), ('katzenbeisser', 1), ('kahn', 1), ('k', 1), ('joshua', 1), ('joseph', 1), ('jorge', 1), ('john', 1), ('joan', 1), ('jntak', 1), ('jimmy', 1), ('jeroen', 1), ('james', 1), ('j', 1), ('integer', 1), ('howard', 1), ('horton', 1), ('holt', 1), ('holden', 1), ('hill', 1), ('hcrkiabie', 1), ('hallà', 1), ('hall', 1), ('h', 1), ('graaf', 1), ('goto', 1), ('gilbert', 1), ('george', 1), ('gaj', 1), ('g', 1), ('friedrich', 1), ('filho', 1), ('file', 1), ('f', 1), ('edward', 1), ('ecs', 1), ('e', 1), ('dunkelberger', 1), ('dunigan', 1), ('doyle', 1), ('dlog', 1), ('diwhu', 1), ('director', 1), ('dexter', 1), ('decryption', 1), ('de', 1), ('david', 1), ('danny', 1), ('daly', 1), ('curricula', 1), ('correa', 1), ('conan', 1), ('computing', 1), ('colin', 1), ('clausewitz', 1), ('cipher', 1), ('christianna', 1), ('chapter', 1), ('carl', 1), ('camille', 1), ('bruce', 1), ('brown', 1), ('brian', 1), ('brandon', 1), ('belfanti', 1), ('balenson', 1), ('av', 1), ('asymmetric', 1), ('arthur', 1), ('appendix', 1), ('anton', 1), ('andrew', 1), ('andre', 1)]
```

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

```
[('york', 1), ('washington', 1), ('vernam', 1), ('united', 1), ('toronto', 1), ('tokyo', 1), ('the', 1), ('stein', 1), ('states', 1), ('page', 1), ('north', 1), ('new', 1), ('moscow', 1), ('malaysia', 1), ('london', 1), ('laguna', 1), ('kx', 1), ('kong', 1), ('jersey', 1), ('japan', 1), ('iii', 1), ('hong', 1), ('hills', 1), ('germany', 1), ('galois', 1), ('egypt', 1), ('cetin', 1), ('canada', 1), ('belgium', 1), ('australia', 1), ('asia', 1), ('america', 1)]
```