Natural Language Processing

CSE 3201

Round – 1 & 2 Combined

Project Report by

Team: Ctrl Alt Elite

Nirnay Korde(20UCS133)

Raghav Khanna(20UCS153)

Sourav Jain(20UCS198)

Prateek Jain(20UCS145)

Course Instructor

Dr. Sakthi Balan Muthiah

Associate Professor, Dept. of CSE

Department of Computer Science Engineering

The LNM Institute of Information Technology, Jaipur

Abstract

Natural Language Processing is a subfield of artificial intelligence that deals with interactions between computers and human users. It is basically to find a way to make the computers process and analyze the natural language data.

This project aims to use the existing libraries of Python, to know the various algorithms used behind the processing of text, and also to get a clear understanding of how the algorithms are used in real life.
This report has a summary of the various steps included during the code part of our research, along with visualizations by graphs for easy inferences for a better understanding of the text as well.

Table of Contents

Round – 1

- **The complete code is on our GitHub repository. Link to our repository:**
  Ctrl Alt Elite

Problem Statement

This project involves various questions to be solved, which centers around various NLP topics such as,

a) Clean the Data
b) Tokenization of text
c) Creation of Word Cloud
d) Giving appropriate tags to the word, PoS Tagging

Data Visualization is also a key part of this project, which we use to get a clear understanding of the text so that we can accordingly prepare for the text processing steps. It is also used to get various inferences of the data after processing for an easy understanding.

All of the Language Processing in this project is done using the **Python** programming language, which has a variety of libraries available for the same. We'll be using the Natural Language Toolkit, commonly known as the **NLTK library** for the text processing, along with data visualization libraries such as **matplotlib.pyplot, WordCloud.**

Data Description

The text used for this project is a course book downloaded in text format of the Operating System Concepts 9th Edition : Abraham Silberschatz . We've used this book for the purpose.

Book Description –
No. of Characters: 17,20,614
No. of Words: 2,99,138

Text Pre-Processing Steps

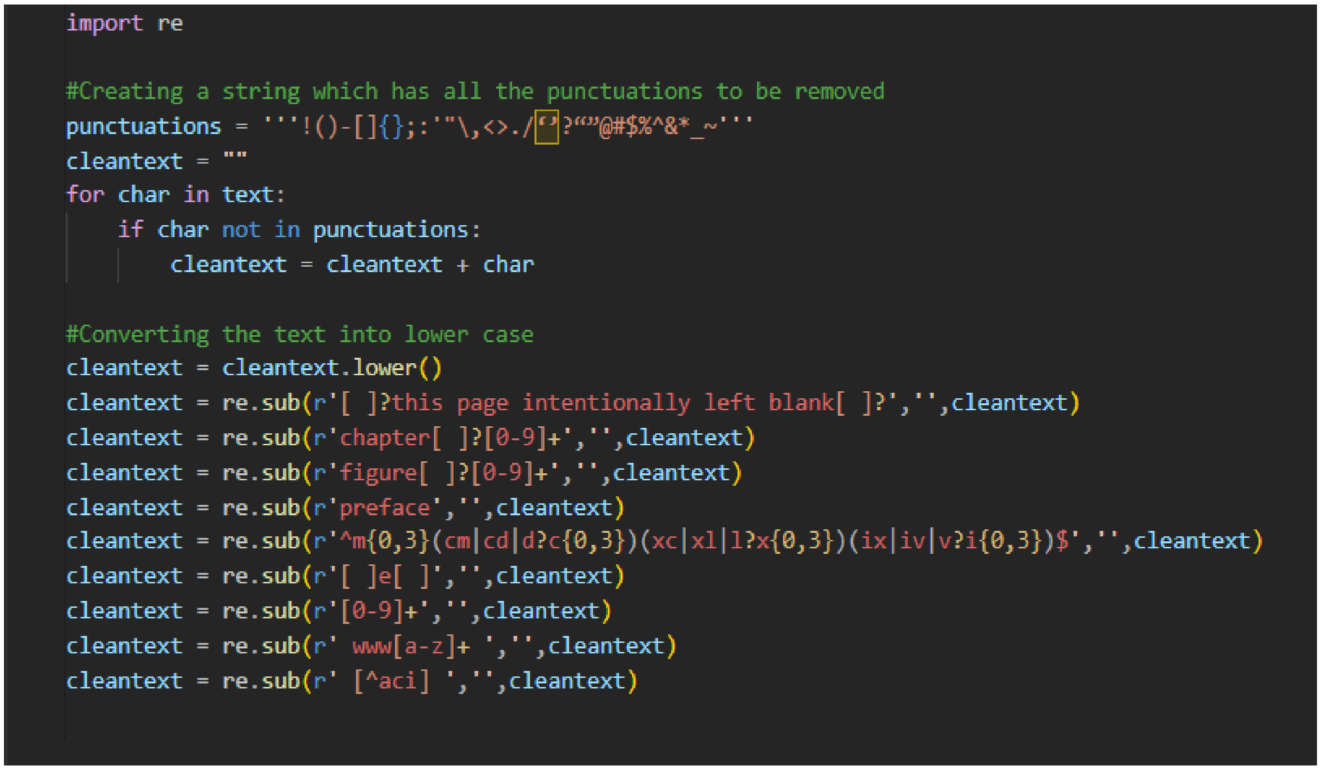
Text Pre-processing is an important part of Natural Language Processing tasks. It helps to make the data in a more digestible form, which can thus be easily worked upon by the machine learning algorithms.
We used the Python libraries

- **‘nltk’ for text processing**
- **‘matplotlib’ for visualization.**

The tasks in Pre-processing we’ve performed in our code are:

a) Removing content irrelevent to overall book’s theme.
b) The following sections has been removed- copyright, preface, index, credits.
c) **Removing Punctuations and Normalization**:
    - We firstly removed all the punctuation characters such as “ **!, (,), ;** ” etc characters from the entire text novel.
    - Next, we normalized the text.
    - This means we convert the text in a standard form which becomes easy to process.
    - We turned the entire text into lower case letters.

- Converting text into clean-text and removing irrelevant text like figure numbers, roman numbers, links etc. with the help of Regular Expressions(RE).



```python
import re

#Creating a string which has all the punctuations to be removed
punctuations = '''!()-[]{};:'"\,<>./‘’?“”@#$%^&*_~'''
cleantext = ""
for char in text:
    if char not in punctuations:
        cleantext = cleantext + char

#Converting the text into lower case
cleantext = cleantext.lower()
cleantext = re.sub(r'[ ]?this page intentionally left blank[ ]?','',cleantext)
cleantext = re.sub(r'chapter[ ]?[0-9]+','',cleantext)
cleantext = re.sub(r'figure[ ]?[0-9]+','',cleantext)
cleantext = re.sub(r'preface','',cleantext)
cleantext = re.sub(r'^m{0,3}(cm|cd|d?c{0,3})(xc|xl|l?x{0,3})(ix|iv|v?i{0,3})$','',cleantext)
cleantext = re.sub(r'[ ]e[ ]','',cleantext)
cleantext = re.sub(r'[0-9]+','',cleantext)
cleantext = re.sub(r' www[a-z]+ ','',cleantext)
cleantext = re.sub(r' [^aci] ','',cleantext)
```

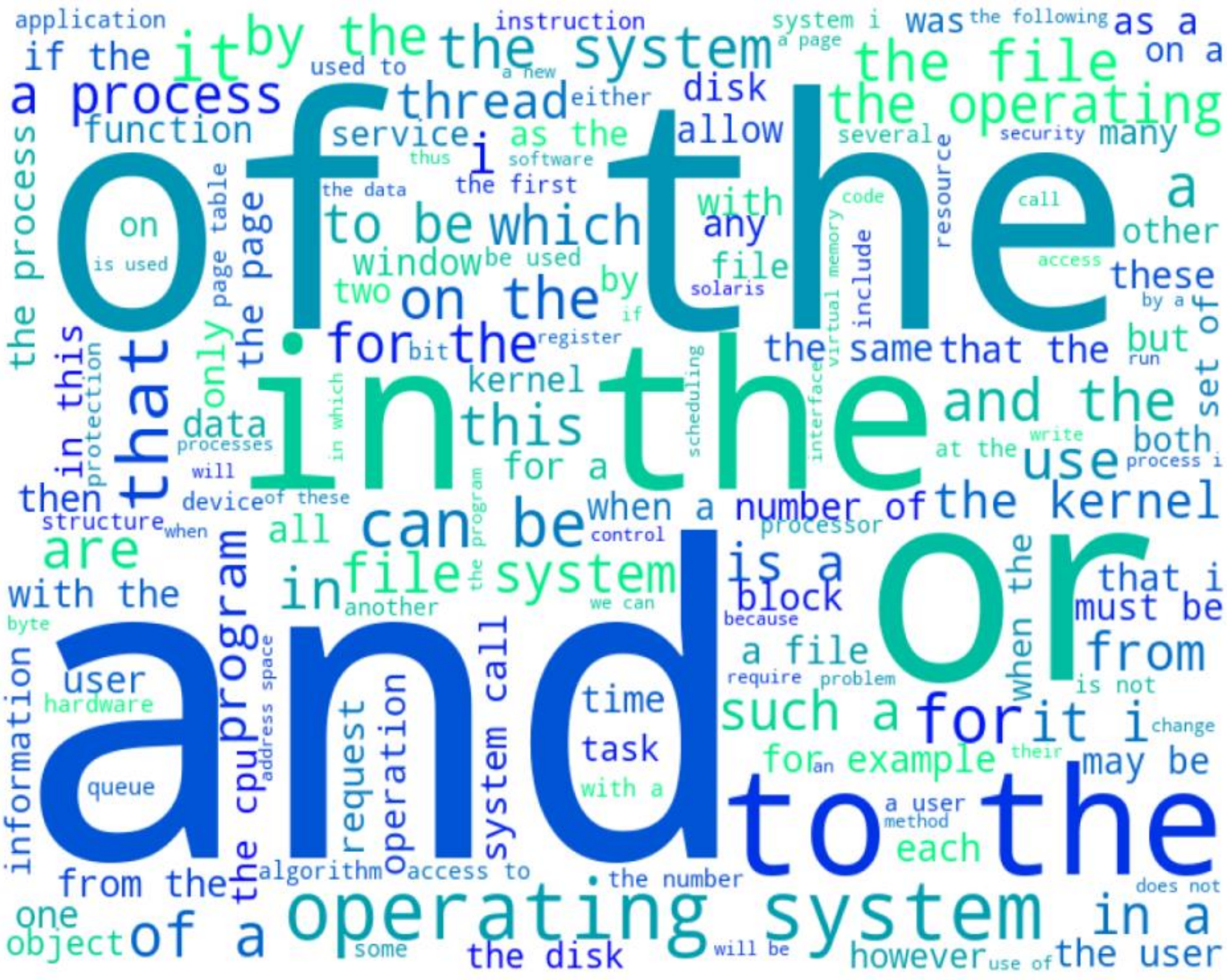
- Length of cleantext : 16,60,217

➢ We can see that the difference between text and clean text is 60,397. This much length of text was occupied by the text which is not of our interest in this project.

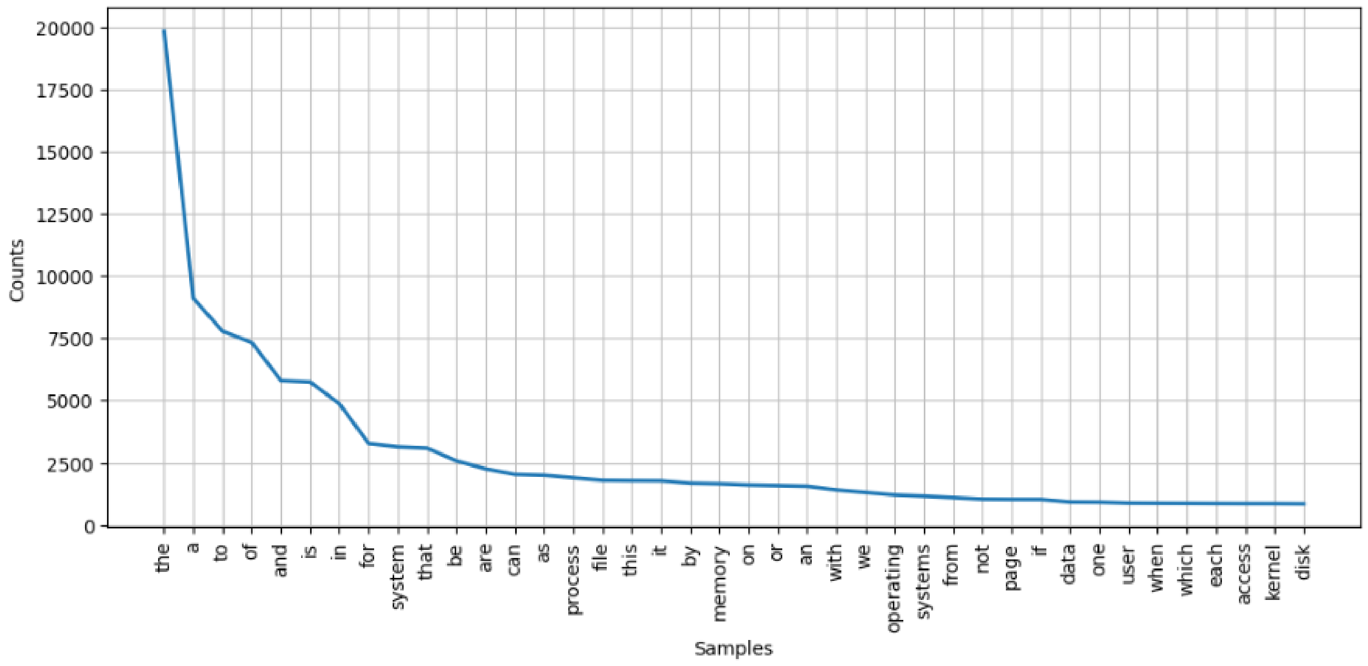
d) Next, we tokenize the cleaned text using the function **‘word_tokenize’** from the **nltk.tokenize library.**

- Tokenizers divide strings into lists of substrings.
- This particular tokenizer ‘word_tokenize’ requires the **Punkt sentence tokenization model** to be installed.
- This Punkt sentence tokenizer divides text into a list of sentences by using an unsupervised algorithm to build a model for words.

- **WordCloud of Cleantext without removing stopwords.**



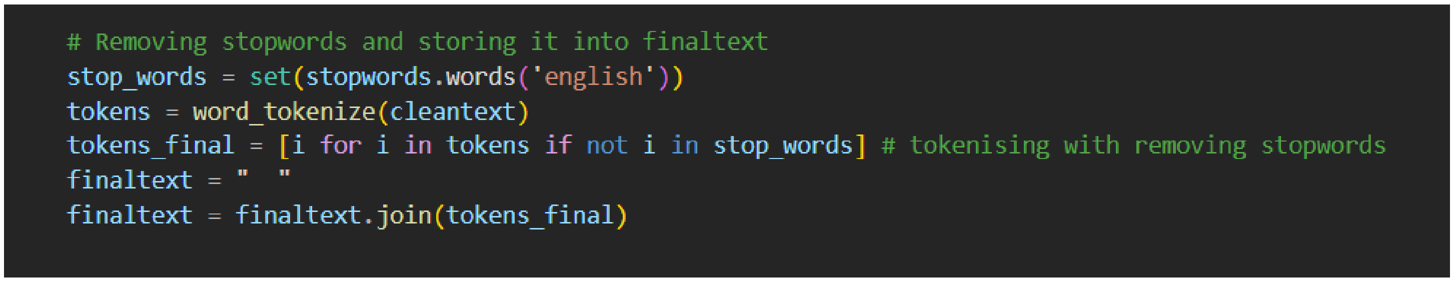
- **Frequency distribution of cleantext without removing stopwords**



➢ **INFERENCE:**

- As we expected that the cleantext without removing the stopwords will have high frequency of words like **'the','a','to'** etc. which can be seen through wordcloud and frequency distribution graph.

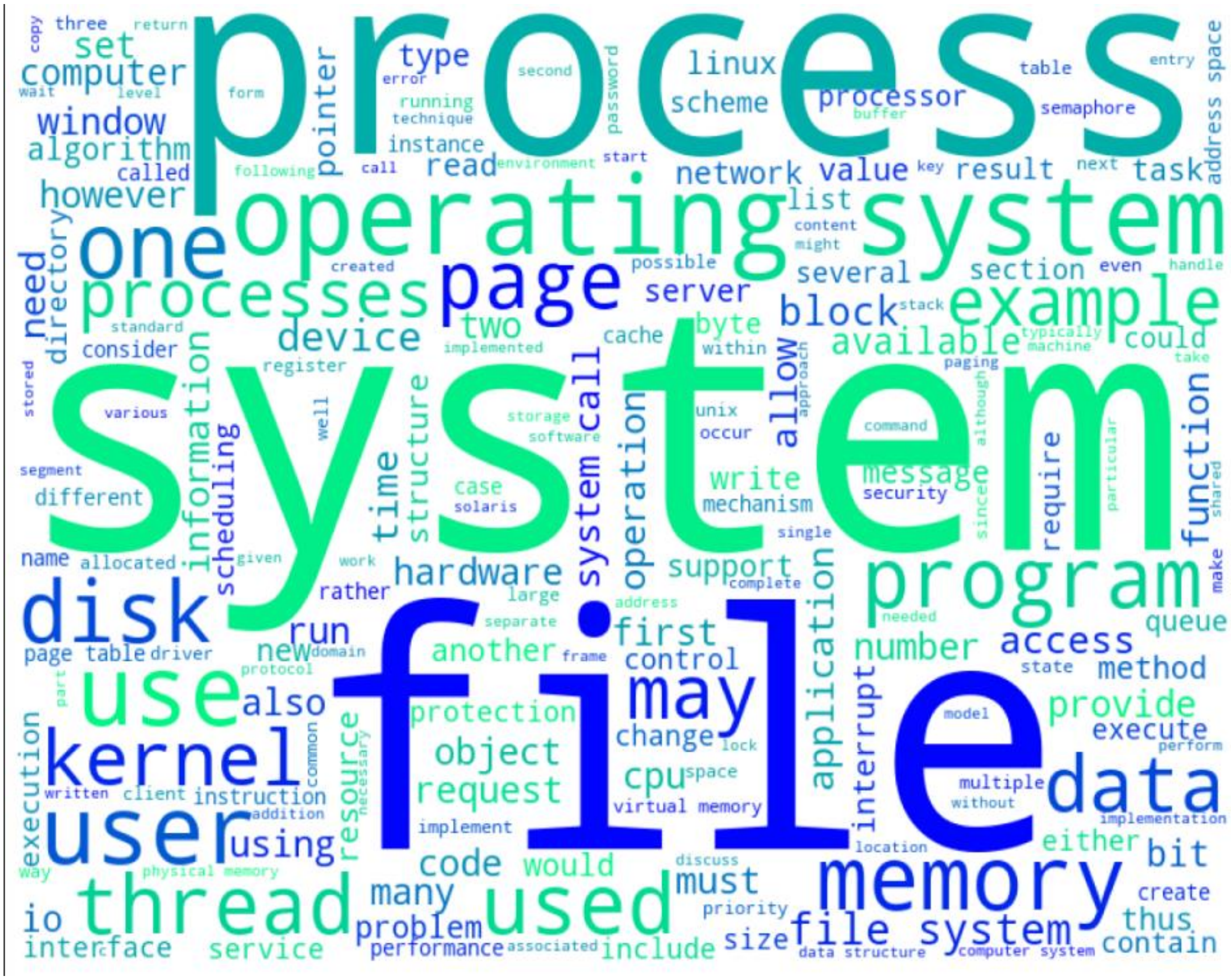
e) Removing Stopwords from Cleantext



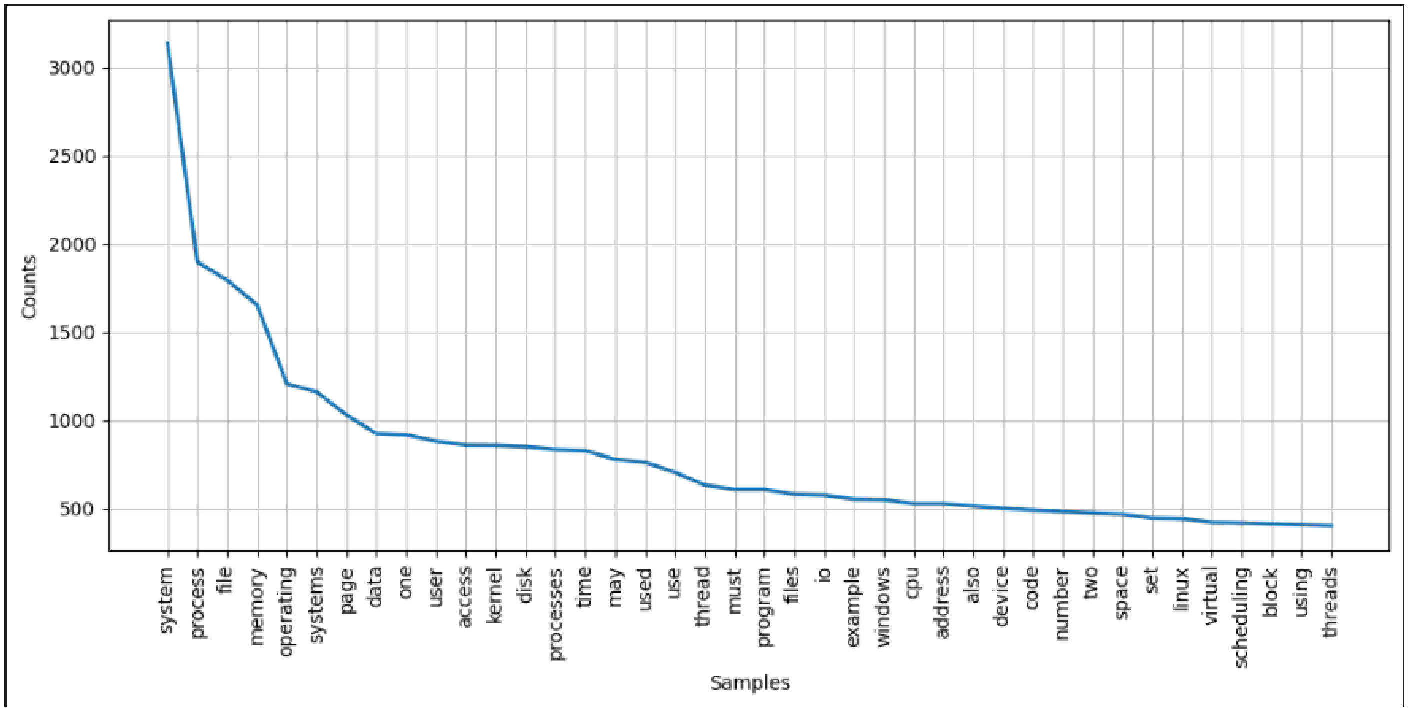
```
# Removing stopwords and storing it into finaltext
stop_words = set(stopwords.words('english'))
tokens = word_tokenize(cleantext)
tokens_final = [i for i in tokens if not i in stop_words] # tokenising with removing stopwords
finaltext = "  "
finaltext = finaltext.join(tokens_final)
```

- Length of finaltext : 13,90,527
- We use **nltk.download(“stopwords”)** to download the list of stopwords provided by nltk.
- Then, we match each token with the stopword and remove the words which gets matched with the stopwords. This way we create our finaltext.

- **Wordcloud of finaltext (cleantext without stopwords)**



- **Frequency distribution of Finaltext**



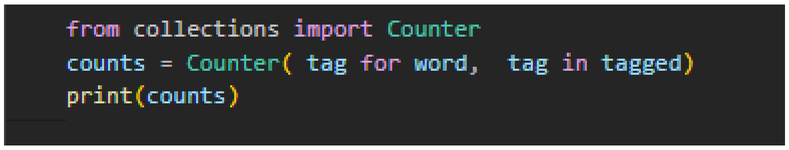
➢ **INFERENCE:**

- After the removal of stopwords , we get the finaltext .
- With the help of WordCloud and frequency distribution we can infer what words appears the most in the text.
- As our text is an Operating System Book, we can see that words like **'System', 'process', 'file' and 'memory'** appears the most.

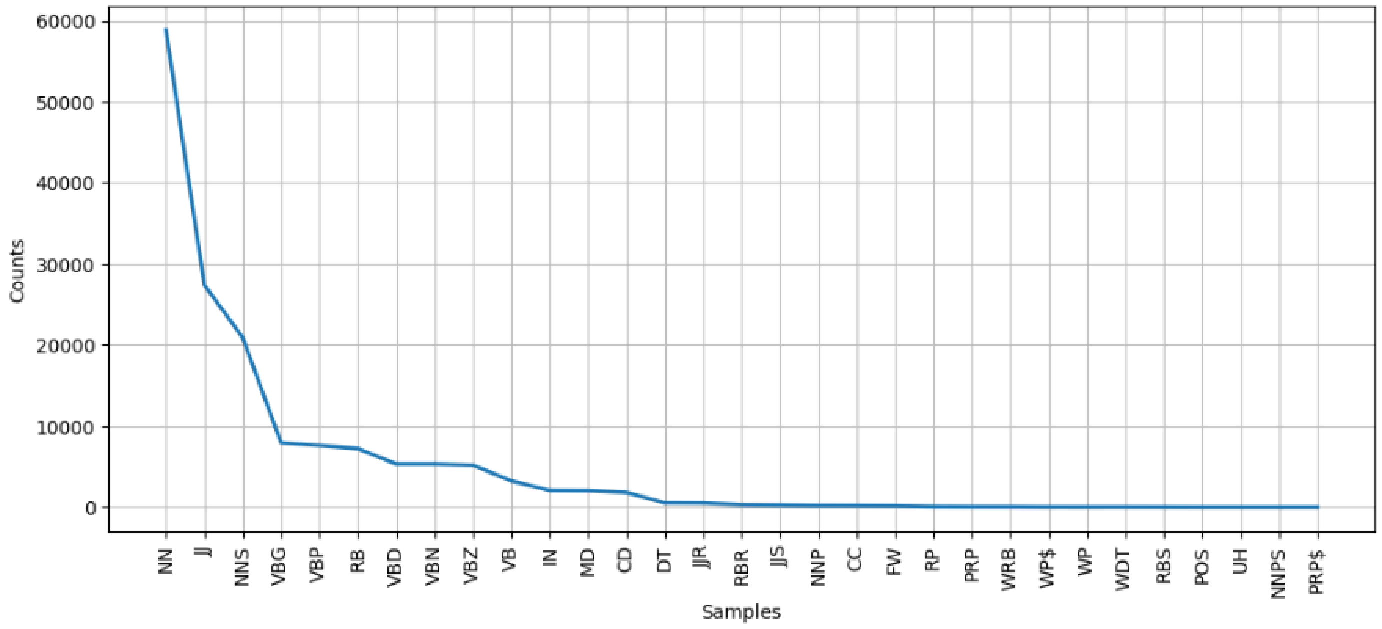
PoS Tagging

After we thoroughly clean our data, we proceed with the text processing part. Here, we have to assign appropriate Part-of-Speech Tags to the tokens.

For this, we use “**nltk.pos_tag( )**” from the NLTK library of python.



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

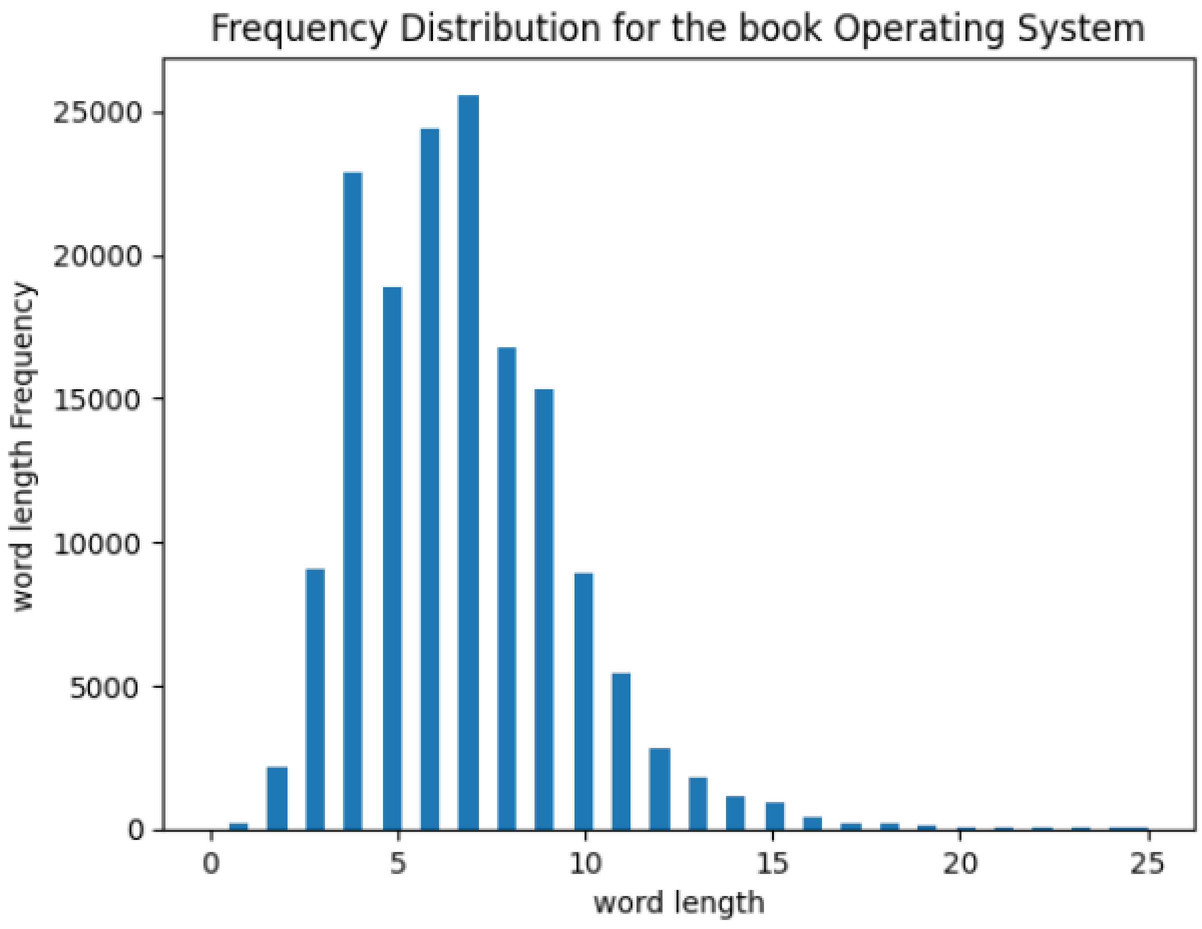


Relationship between the word length and frequency:

Here, we analyze a relationship between the word length and how many words with such word length occurs.

- Using **len( )** function we calculate the length of each token
- Then we plot a graph for frequency of such word lengths using **matplotlib.pyplot.**

➢ **INFERENCE:** We can see that majority of words have their length in Between **4** to **10** letters.



Output for Frequency Distribution for the book Operating System

ROUND 2

The text used for this project is a course book downloaded in text format of the Operating System Concepts 9th Edition : Abraham Silberschatz . We've used this book for the purpose.

- **The complete code is on our GitHub repository. Link to our repository:**
  Ctrl Alt Elite

This Round has been divided into the following three parts –

First Part

Problem Statement – This part can be further divided into two subparts –

a.) In this, we have to find the nouns and verbs in the book. Then, we have to get the categories that these words fall under in the WordNet.

b.) In this, we have to get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same.

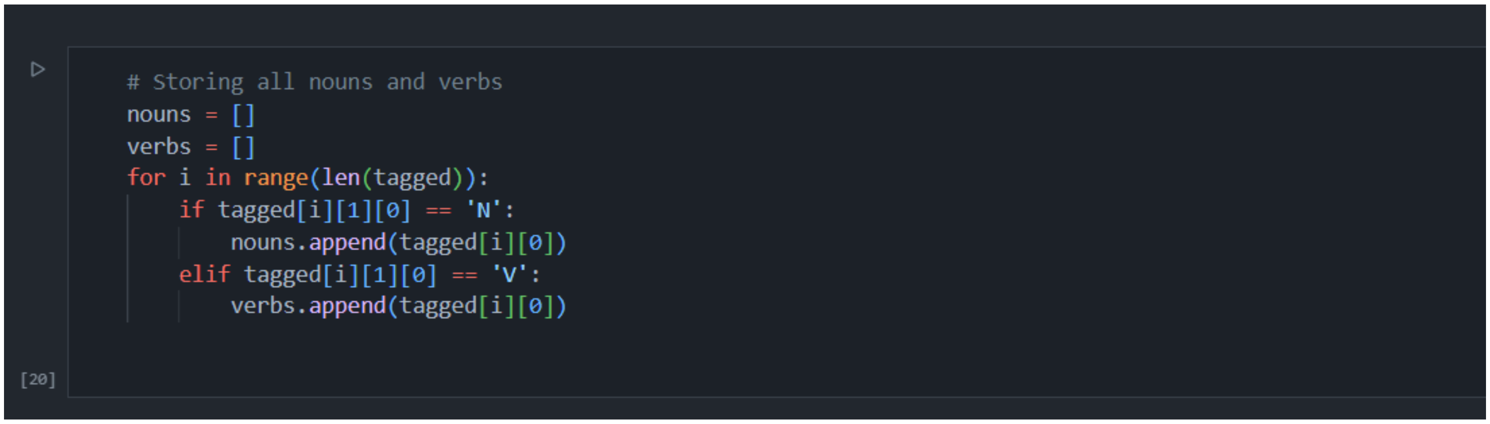
Libraries Used – POS Tagging (Same as Round-1), Wordnet

Wordnet Description –

WordNet is a lexical database for the English language, which was created by Princeton, and is part of the NLTK corpus. You can use WordNet alongside the NLTK module to find the meanings of words, synonyms, antonyms, and more.

Steps Involved –

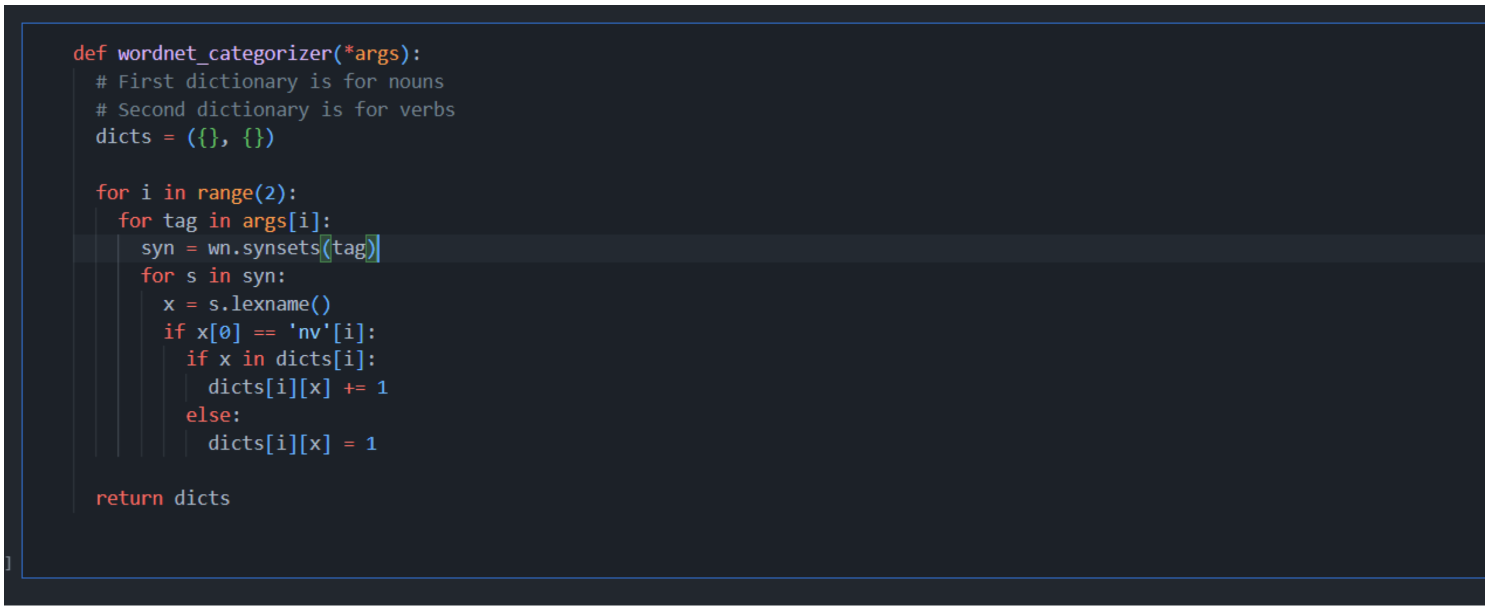
1.) **Store all the Nouns and Verbs from the "tagged"** ("tagged" is a variable containing the tags from round-1)



```
# Storing all nouns and verbs
nouns = []
verbs = []
for i in range(len(tagged)):
    if tagged[i][1][0] == 'N':
        nouns.append(tagged[i][0])
    elif tagged[i][1][0] == 'V':
        verbs.append(tagged[i][0])
```

[20]

2.) Create Dictionary for each, Nouns and Verbs –



```
def wordnet_categorizer(*args):
    # First dictionary is for nouns
    # Second dictionary is for verbs
    dicts = ({}, {})

    for i in range(2):
        for tag in args[i]:
            syn = wn.synsets(tag)
            for s in syn:
                x = s.lexname()
                if x[0] == 'nv'[i]:
                    if x in dicts[i]:
                        dicts[i][x] += 1
                    else:
                        dicts[i][x] = 1

    return dicts
```

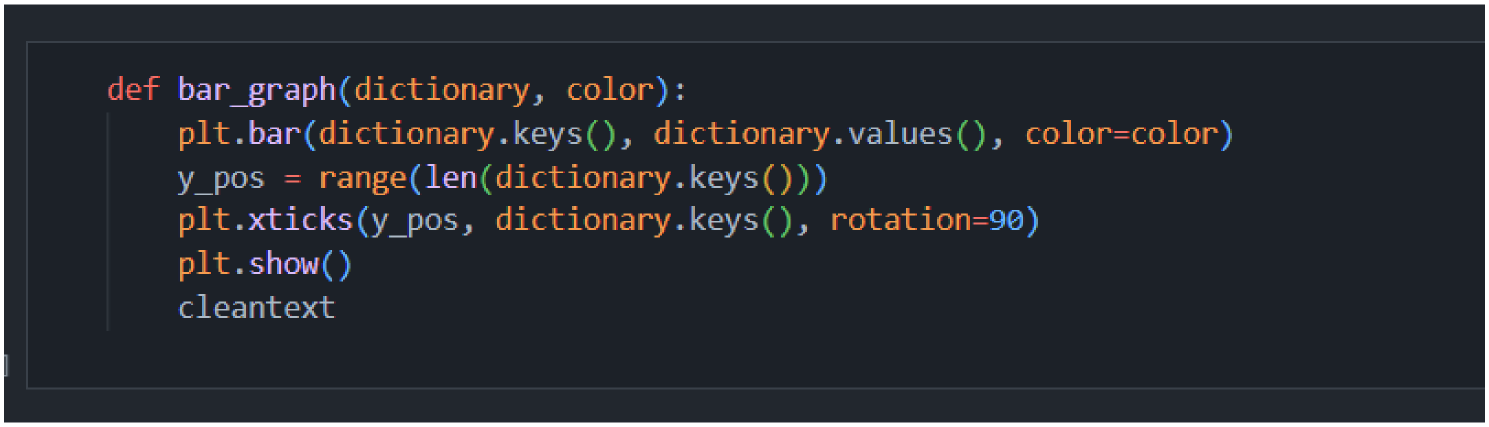
For this, we made a custom function that uses the function **“synset”** from wordnet.

Synset is a special kind of a simple interface that is present in **NLTK** to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept. Some of the words have only one Synset and some have several.

Every synset has a method **“lexname”.** We have used this to create dictionaries.

3.) Create Bar Graphs

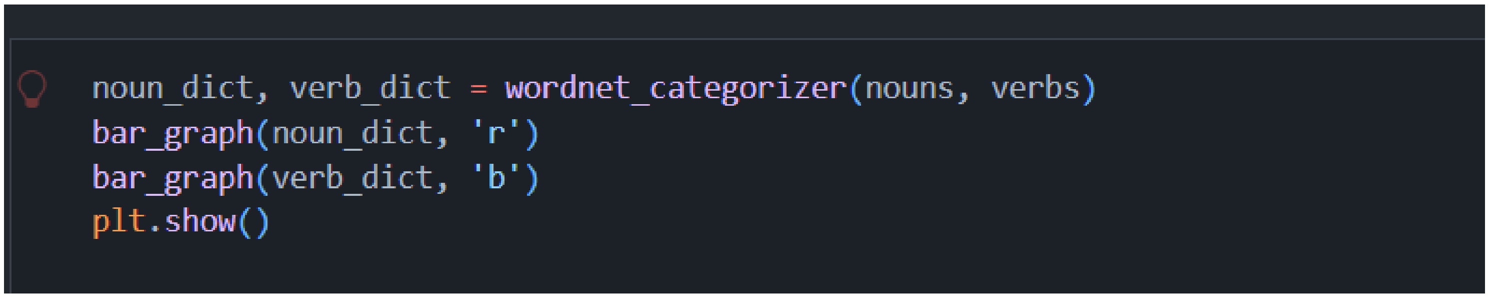
For creating bar graphs, we have built a custom function that takes input a dictionary and a color –



```
def bar_graph(dictionary, color):
    plt.bar(dictionary.keys(), dictionary.values(), color=color)
    y_pos = range(len(dictionary.keys()))
    plt.xticks(y_pos, dictionary.keys(), rotation=90)
    plt.show()
    cleantext
```

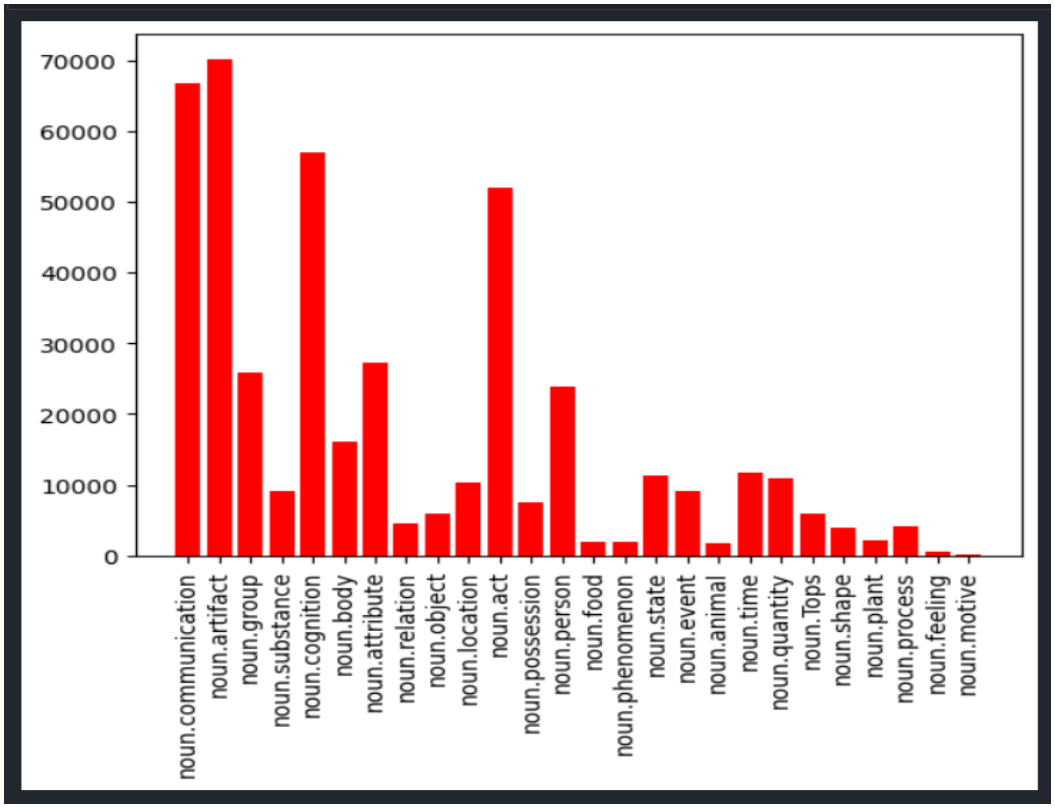
"plt" is the imported Matplotlib library.

Result –

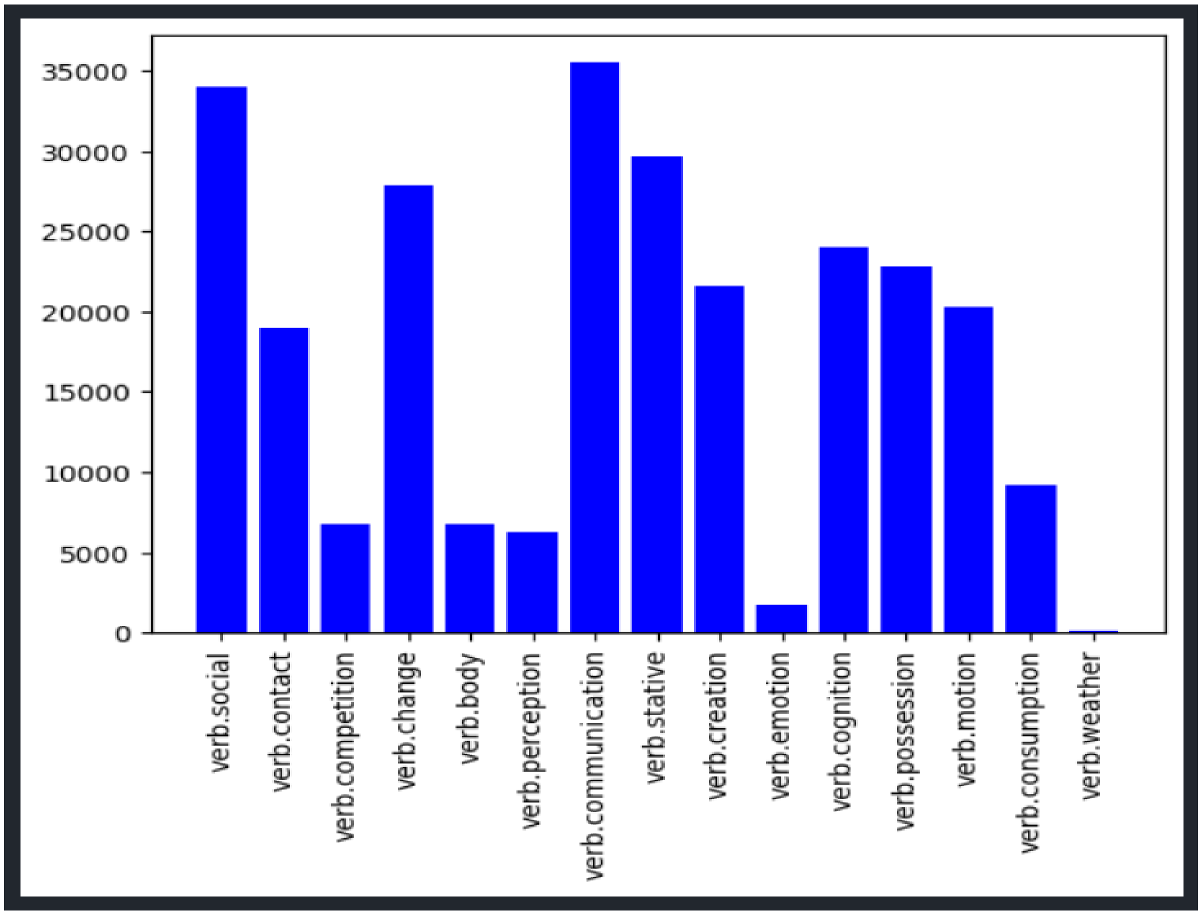


```
noun_dict, verb_dict = wordnet_categorizer(nouns, verbs)
bar_graph(noun_dict, 'r')
bar_graph(verb_dict, 'b')
plt.show()
```

Noun –


70000
60000
50000
40000
30000
20000
10000
0
noun.communication
noun.artifact
noun.group
noun.substance
noun.cognition
noun.body
noun.attribute
noun.relation
noun.object
noun.location
noun.act
noun.possession
noun.person
noun.food
noun.phenomenon
noun.state
noun.event
noun.animal
noun.time
noun.quantity
noun.Tops
noun.shape
noun.plant
noun.process
noun.feeling
noun.motive

Verb –


35000
30000
25000
20000
15000
10000
5000
0
verb.social
verb.contact
verb.competition
verb.change
verb.body
verb.perception
verb.communication
verb.stative
verb.creation
verb.emotion
verb.cognition
verb.possession
verb.motion
verb.consumption
verb.weather

Second Part

Problem Statement – This part can be further divided into two subparts –

a.) Recognize all the entities
b.) Recognize all the entity types.

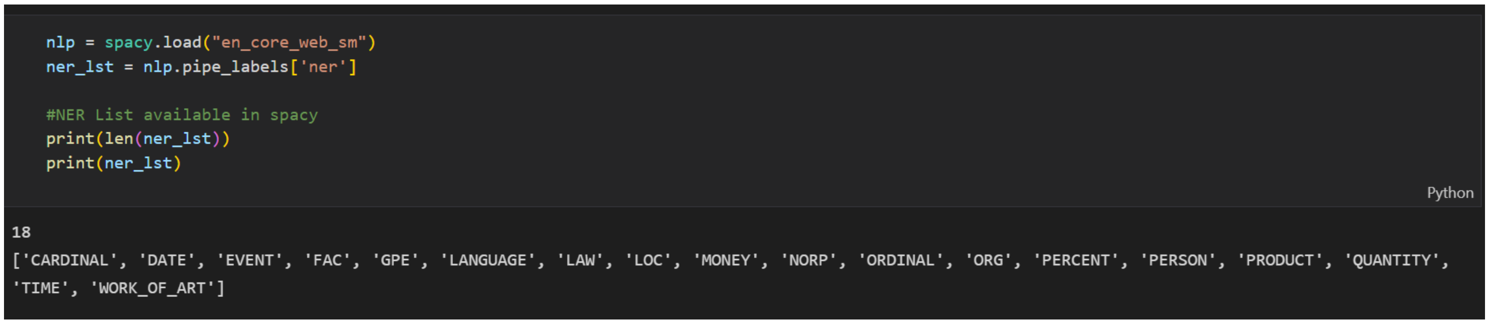
Libraries Used – Spacy, displacy

Spacy Description – (Entity Recognizer)

The entity recognizer identifies non-overlapping labelled spans of tokens. The transition-based algorithm used encodes certain assumptions that are effective for "traditional" named entity recognition tasks, but may not be a good fit for every span identification problem. Specifically, the loss function optimizes for whole entity accuracy, so if your inter-annotator agreement on boundary tokens is low, the component will likely perform poorly on your problem. The transition-based algorithm also assumes that the most decisive information about your entities will be close to their initial tokens. If your entities are long and characterized by tokens in their middle, the component will likely not be a good fit for your task.
[Source - https://spacy.io/api/entityrecognizer]

Steps Involved –

1.) Recognize the Entity Types in the Library used.



```
nlp = spacy.load("en_core_web_sm")
ner_lst = nlp.pipe_labels['ner']

#NER List available in spacy
print(len(ner_lst))
print(ner_lst)
```

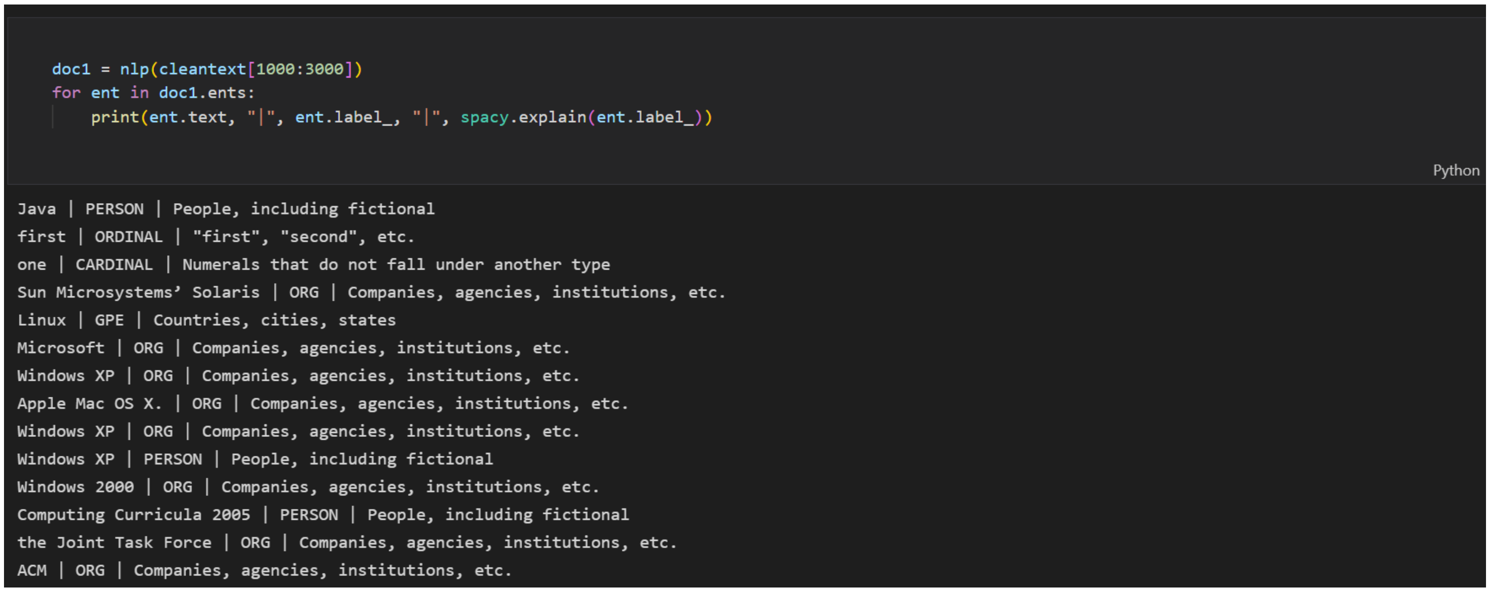
Python

```
18
['CARDINAL', 'DATE', 'EVENT', 'FAC', 'GPE', 'LANGUAGE', 'LAW', 'LOC', 'MONEY', 'NORP', 'ORDINAL', 'ORG', 'PERCENT', 'PERSON', 'PRODUCT', 'QUANTITY',
'TIME', 'WORK_OF_ART']
```

As we can see that ‘spacy’ has 18 entity types.

2.) Do Entity Recognition and Entity Type Recognition using the library

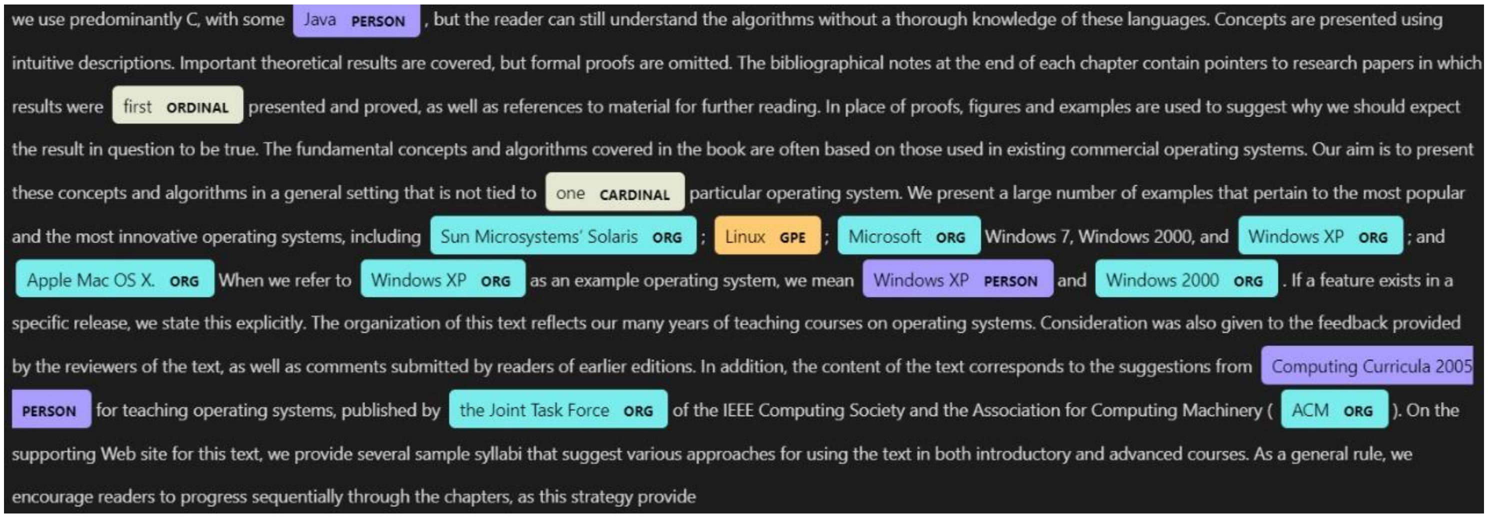
As the document we are using here is very big to be processed, we use an example part from the text as a test.
Here is the result -



```python
doc1 = nlp(cleantext[1000:3000])
for ent in doc1.ents:
    print(ent.text, "|", ent.label_, "|", spacy.explain(ent.label_))
```

Python

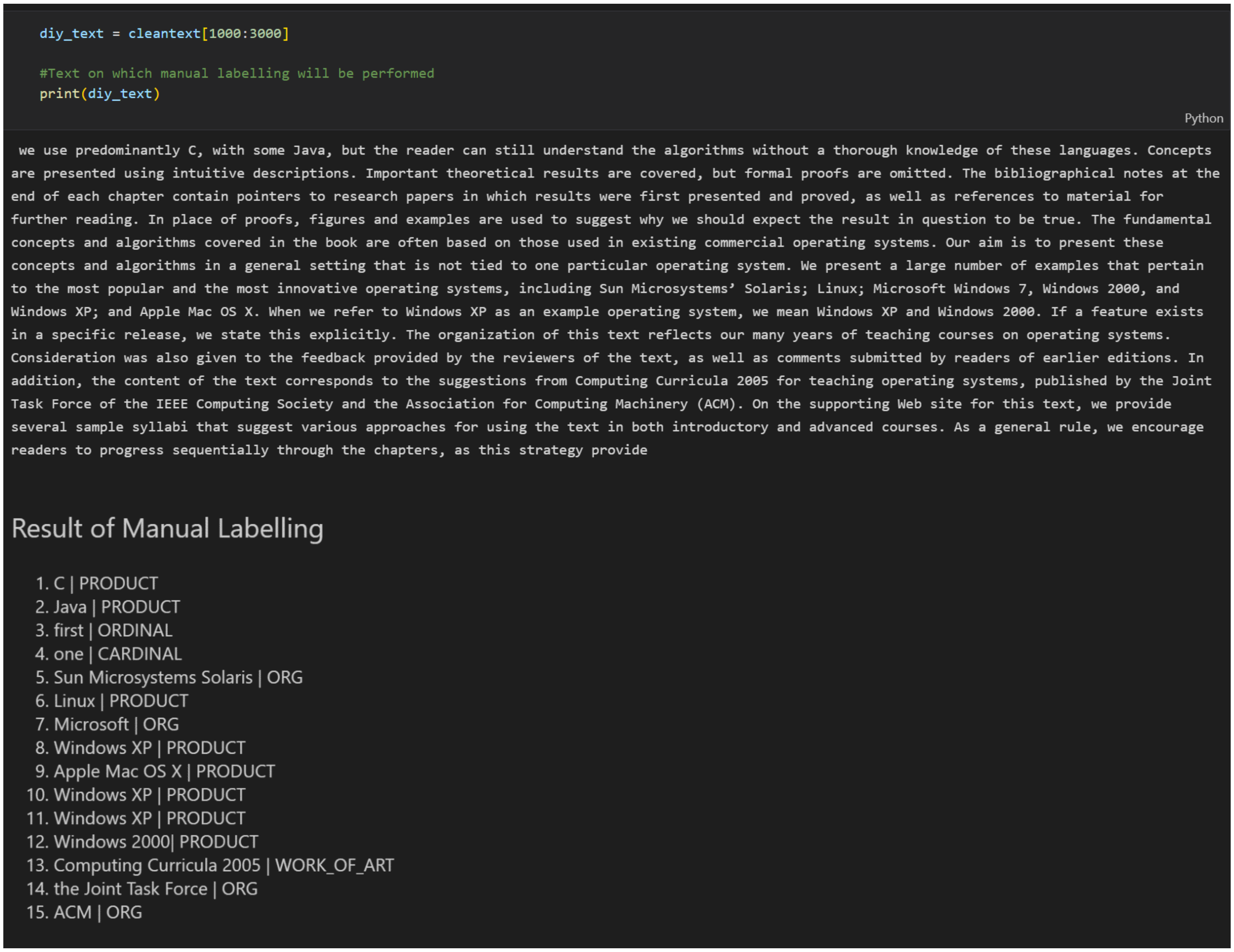
```
Java | PERSON | People, including fictional
first | ORDINAL | "first", "second", etc.
one | CARDINAL | Numerals that do not fall under another type
Sun Microsystems’ Solaris | ORG | Companies, agencies, institutions, etc.
Linux | GPE | Countries, cities, states
Microsoft | ORG | Companies, agencies, institutions, etc.
Windows XP | ORG | Companies, agencies, institutions, etc.
Apple Mac OS X. | ORG | Companies, agencies, institutions, etc.
Windows XP | ORG | Companies, agencies, institutions, etc.
Windows XP | PERSON | People, including fictional
Windows 2000 | ORG | Companies, agencies, institutions, etc.
Computing Curricula 2005 | PERSON | People, including fictional
the Joint Task Force | ORG | Companies, agencies, institutions, etc.
ACM | ORG | Companies, agencies, institutions, etc.
```



we use predominantly C, with some Java PERSON , but the reader can still understand the algorithms without a thorough knowledge of these languages. Concepts are presented using intuitive descriptions. Important theoretical results are covered, but formal proofs are omitted. The bibliographical notes at the end of each chapter contain pointers to research papers in which results were first ORDINAL presented and proved, as well as references to material for further reading. In place of proofs, figures and examples are used to suggest why we should expect the result in question to be true. The fundamental concepts and algorithms covered in the book are often based on those used in existing commercial operating systems. Our aim is to present these concepts and algorithms in a general setting that is not tied to one CARDINAL particular operating system. We present a large number of examples that pertain to the most popular and the most innovative operating systems, including Sun Microsystems’ Solaris ORG ; Linux GPE ; Microsoft ORG Windows 7, Windows 2000, and Windows XP ORG ; and Apple Mac OS X. ORG When we refer to Windows XP ORG as an example operating system, we mean Windows XP PERSON and Windows 2000 ORG . If a feature exists in a specific release, we state this explicitly. The organization of this text reflects our many years of teaching courses on operating systems. Consideration was also given to the feedback provided by the reviewers of the text, as well as comments submitted by readers of earlier editions. In addition, the content of the text corresponds to the suggestions from Computing Curricula 2005 PERSON for teaching operating systems, published by the Joint Task Force ORG of the IEEE Computing Society and the Association for Computing Machinery (ACM ORG). On the supporting Web site for this text, we provide several sample syllabi that suggest various approaches for using the text in both introductory and advanced courses. As a general rule, we encourage readers to progress sequentially through the chapters, as this strategy provide

3.) Do Manual Labelling –

Here, we use the same text (as above) and do manual labelling -



```python
diy_text = cleantext[1000:3000]

#Text on which manual labelling will be performed
print(diy_text)
```

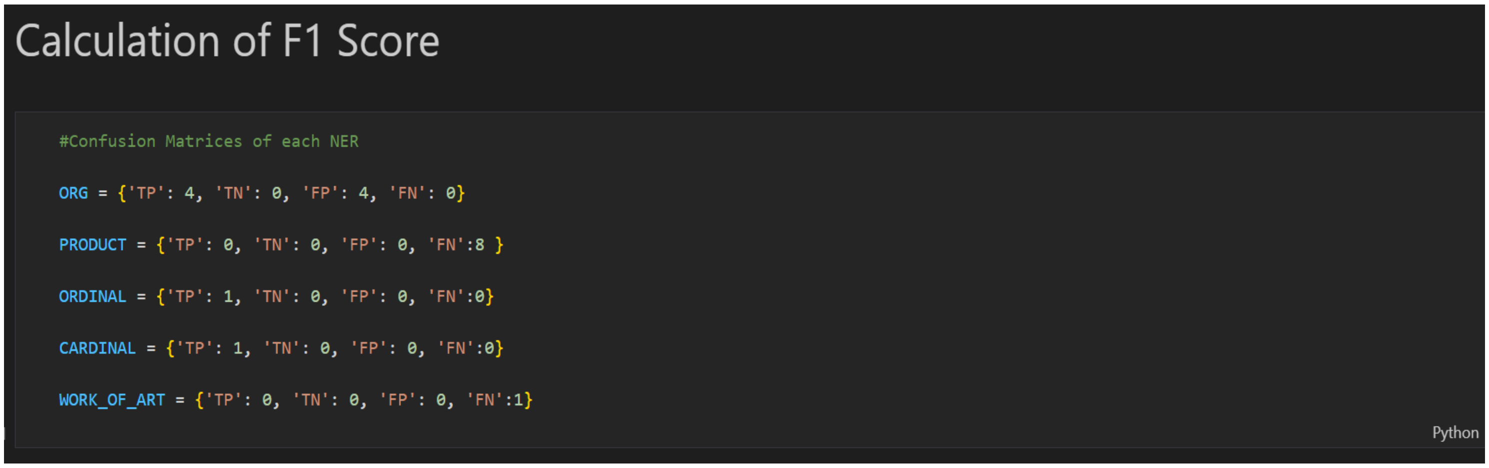
Python

```
 we use predominantly C, with some Java, but the reader can still understand the algorithms without a thorough knowledge of these languages. Concepts
are presented using intuitive descriptions. Important theoretical results are covered, but formal proofs are omitted. The bibliographical notes at the
end of each chapter contain pointers to research papers in which results were first presented and proved, as well as references to material for
further reading. In place of proofs, figures and examples are used to suggest why we should expect the result in question to be true. The fundamental
concepts and algorithms covered in the book are often based on those used in existing commercial operating systems. Our aim is to present these
concepts and algorithms in a general setting that is not tied to one particular operating system. We present a large number of examples that pertain
to the most popular and the most innovative operating systems, including Sun Microsystems' Solaris; Linux; Microsoft Windows 7, Windows 2000, and
Windows XP; and Apple Mac OS X. When we refer to Windows XP as an example operating system, we mean Windows XP and Windows 2000. If a feature exists
in a specific release, we state this explicitly. The organization of this text reflects our many years of teaching courses on operating systems.
Consideration was also given to the feedback provided by the reviewers of the text, as well as comments submitted by readers of earlier editions. In
addition, the content of the text corresponds to the suggestions from Computing Curricula 2005 for teaching operating systems, published by the Joint
Task Force of the IEEE Computing Society and the Association for Computing Machinery (ACM). On the supporting Web site for this text, we provide
several sample syllabi that suggest various approaches for using the text in both introductory and advanced courses. As a general rule, we encourage
readers to progress sequentially through the chapters, as this strategy provide
```

Result of Manual Labelling

1. C | PRODUCT
2. Java | PRODUCT
3. first | ORDINAL
4. one | CARDINAL
5. Sun Microsystems Solaris | ORG
6. Linux | PRODUCT
7. Microsoft | ORG
8. Windows XP | PRODUCT
9. Apple Mac OS X | PRODUCT
10. Windows XP | PRODUCT
11. Windows XP | PRODUCT
12. Windows 2000| PRODUCT
13. Computing Curricula 2005 | WORK_OF_ART
14. the Joint Task Force | ORG
15. ACM | ORG

4.) Caculation F-Score –

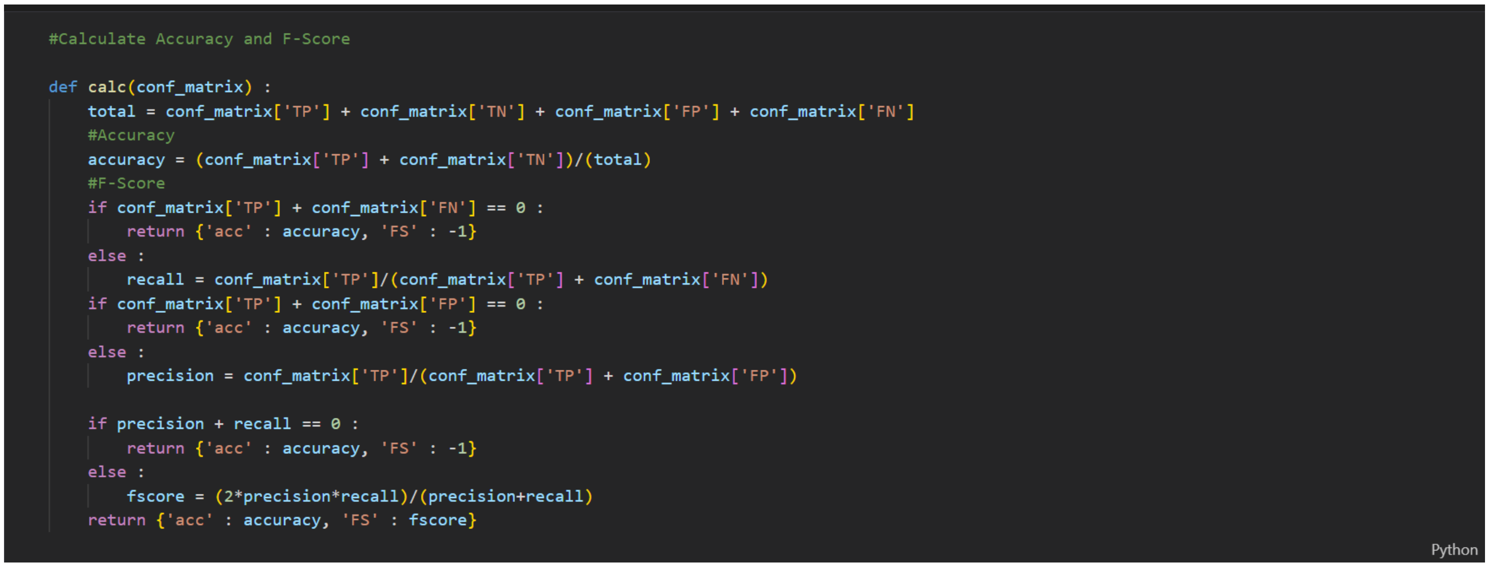


Calculation of F1 Score

```python
#Confusion Matrices of each NER

ORG = {'TP': 4, 'TN': 0, 'FP': 4, 'FN': 0}

PRODUCT = {'TP': 0, 'TN': 0, 'FP': 0, 'FN':8 }

ORDINAL = {'TP': 1, 'TN': 0, 'FP': 0, 'FN':0}

CARDINAL = {'TP': 1, 'TN': 0, 'FP': 0, 'FN':0}

WORK_OF_ART = {'TP': 0, 'TN': 0, 'FP': 0, 'FN':1}
```

Python

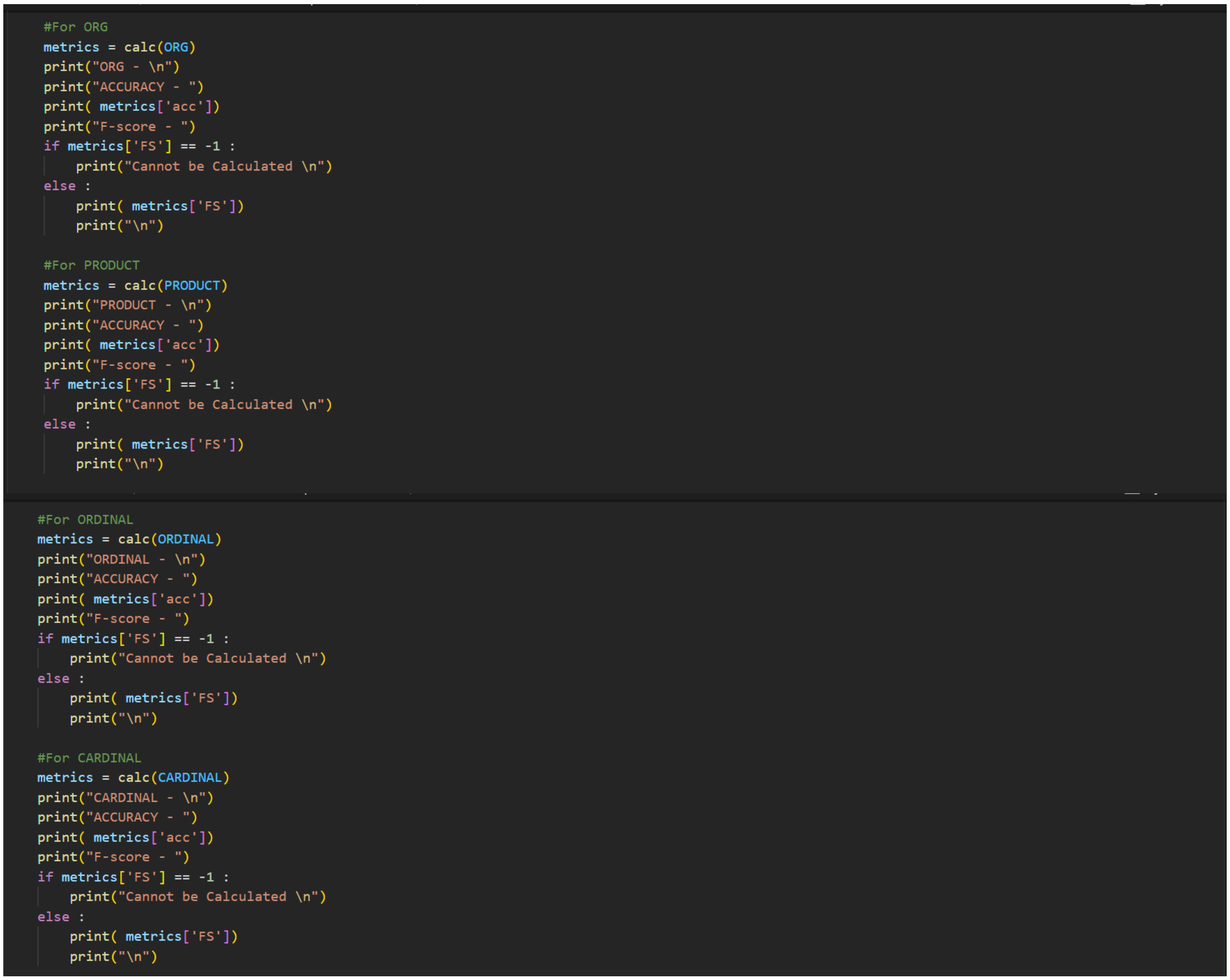
We have made a custom function here that takes a confusion matrix as an input And returns the accuracy and F-score for the same.



```python
#Calculate Accuracy and F-Score

def calc(conf_matrix) :
    total = conf_matrix['TP'] + conf_matrix['TN'] + conf_matrix['FP'] + conf_matrix['FN']
    #Accuracy
    accuracy = (conf_matrix['TP'] + conf_matrix['TN'])/(total)
    #F-Score
    if conf_matrix['TP'] + conf_matrix['FN'] == 0 :
        return {'acc' : accuracy, 'FS' : -1}
    else :
        recall = conf_matrix['TP']/(conf_matrix['TP'] + conf_matrix['FN'])
    if conf_matrix['TP'] + conf_matrix['FP'] == 0 :
        return {'acc' : accuracy, 'FS' : -1}
    else :
        precision = conf_matrix['TP']/(conf_matrix['TP'] + conf_matrix['FP'])

    if precision + recall == 0 :
        return {'acc' : accuracy, 'FS' : -1}
    else :
        fscore = (2*precision*recall)/(precision+recall)
    return {'acc' : accuracy, 'FS' : fscore}
```

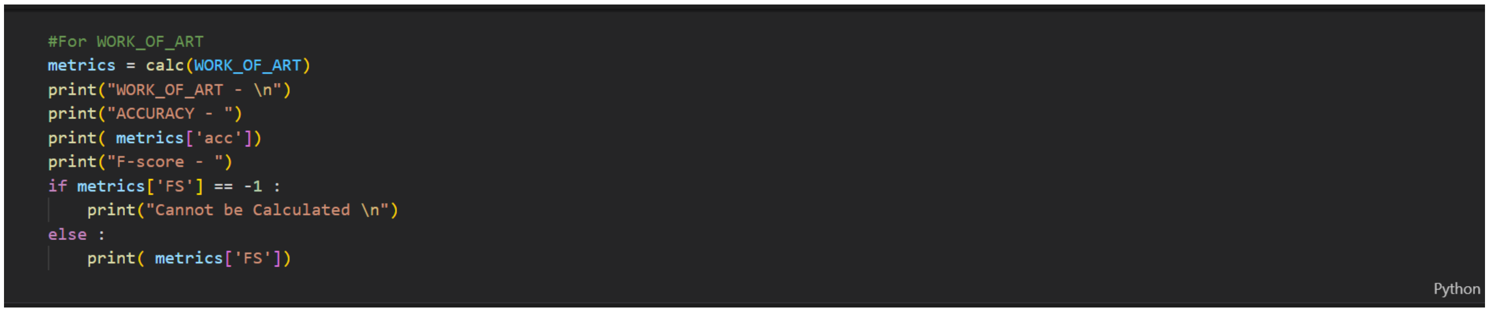
Python

Now, we calculate these for each entity -



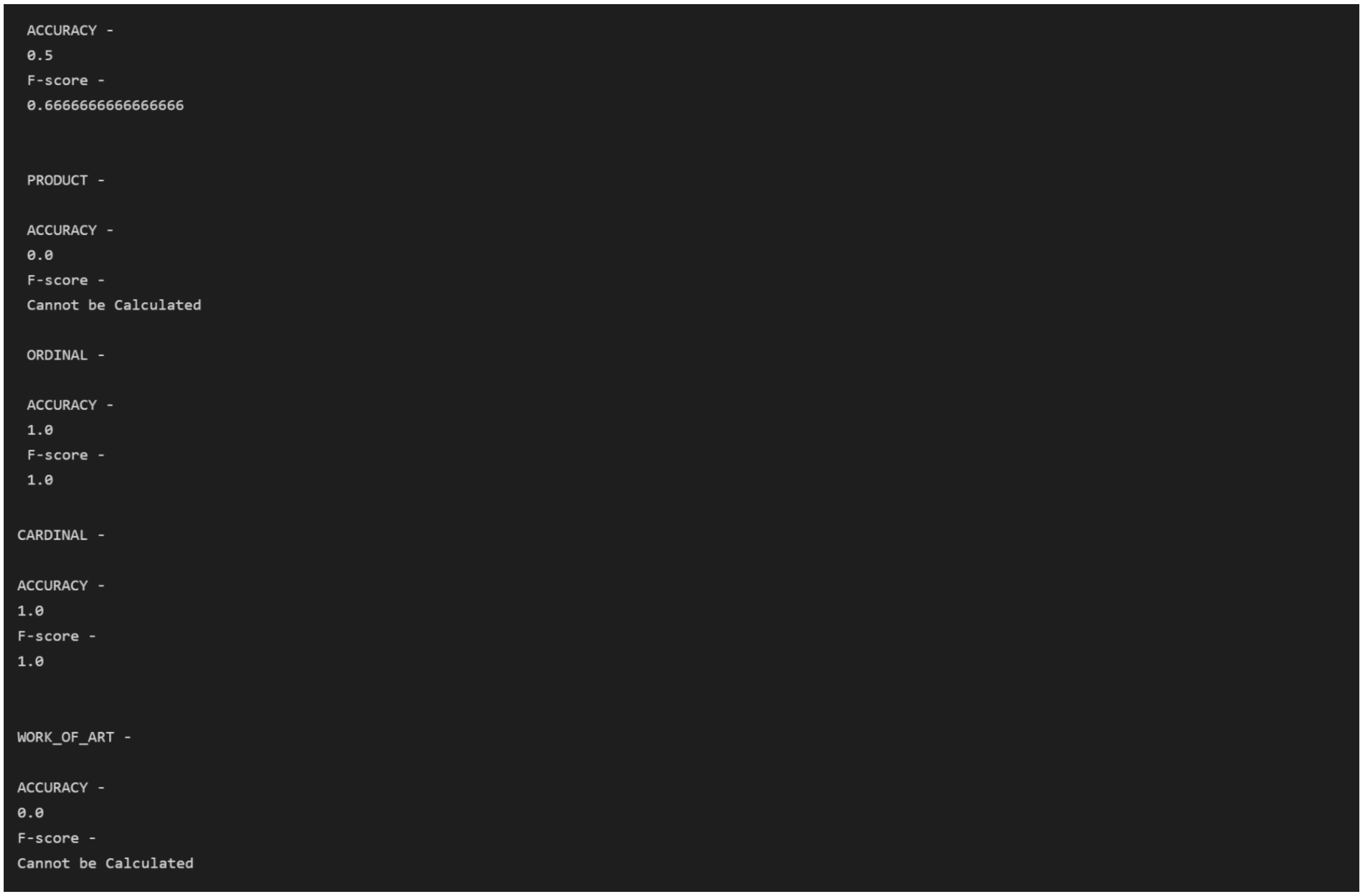
```python
#For ORG
metrics = calc(ORG)
print("ORG - \n")
print("ACCURACY - ")
print( metrics['acc'])
print("F-score - ")
if metrics['FS'] == -1 :
    print("Cannot be Calculated \n")
else :
    print( metrics['FS'])
    print("\n")

#For PRODUCT
metrics = calc(PRODUCT)
print("PRODUCT - \n")
print("ACCURACY - ")
print( metrics['acc'])
print("F-score - ")
if metrics['FS'] == -1 :
    print("Cannot be Calculated \n")
else :
    print( metrics['FS'])
    print("\n")
```

```python
#For ORDINAL
metrics = calc(ORDINAL)
print("ORDINAL - \n")
print("ACCURACY - ")
print( metrics['acc'])
print("F-score - ")
if metrics['FS'] == -1 :
    print("Cannot be Calculated \n")
else :
    print( metrics['FS'])
    print("\n")

#For CARDINAL
metrics = calc(CARDINAL)
print("CARDINAL - \n")
print("ACCURACY - ")
print( metrics['acc'])
print("F-score - ")
if metrics['FS'] == -1 :
    print("Cannot be Calculated \n")
else :
    print( metrics['FS'])
    print("\n")
```



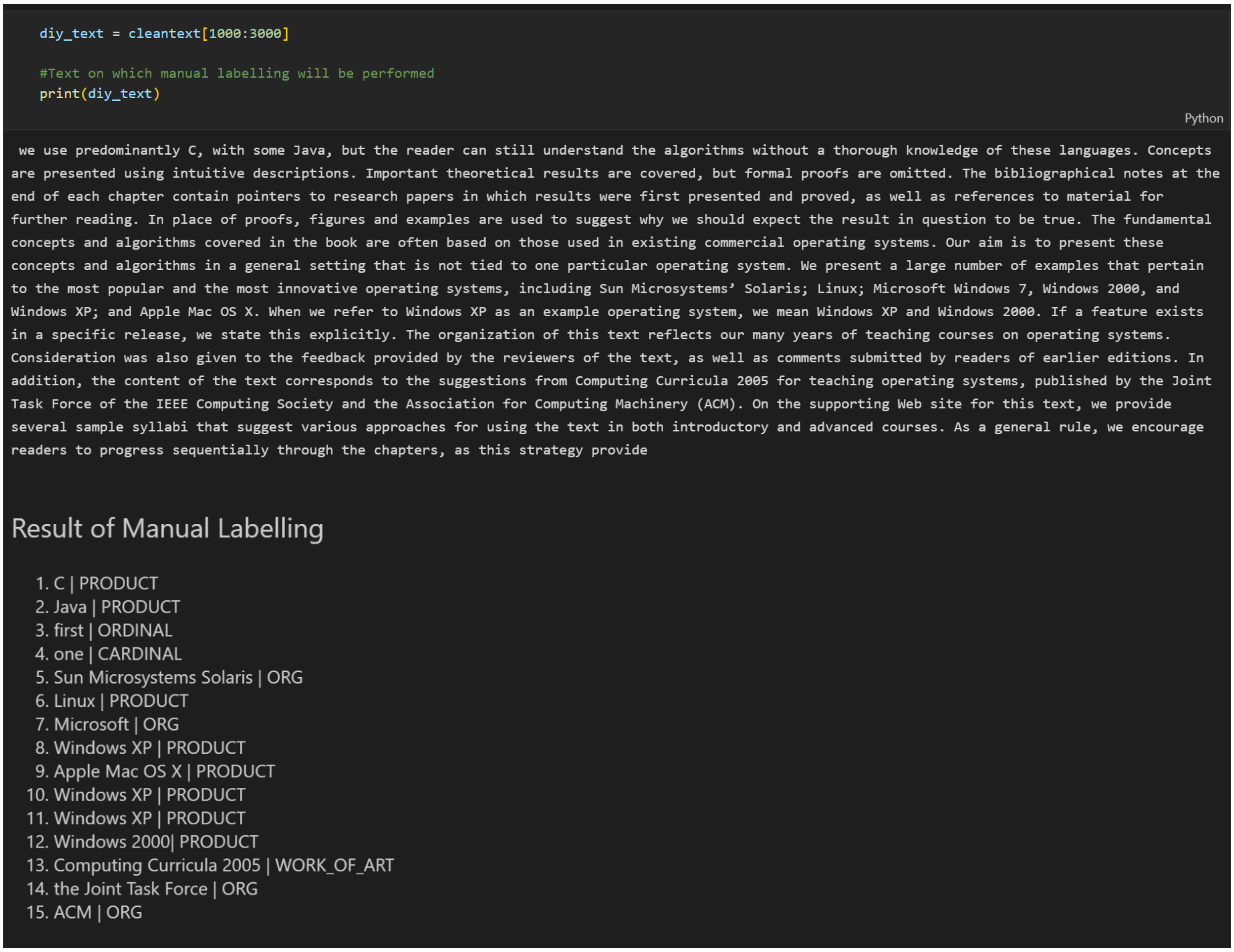
```python
#For WORK_OF_ART
metrics = calc(WORK_OF_ART)
print("WORK_OF_ART - \n")
print("ACCURACY - ")
print( metrics['acc'])
print("F-score - ")
if metrics['FS'] == -1 :
    print("Cannot be Calculated \n")
else :
    print( metrics['FS'])
```

Python



```
 ACCURACY -
 0.5
 F-score -
 0.6666666666666666


 PRODUCT -

 ACCURACY -
 0.0
 F-score -
 Cannot be Calculated

 ORDINAL -

 ACCURACY -
 1.0
 F-score -
 1.0

CARDINAL -

ACCURACY -
1.0
F-score -
1.0


WORK_OF_ART -

ACCURACY -
0.0
F-score -
Cannot be Calculated
```

Third Part



```python
diy_text = cleantext[1000:3000]

#Text on which manual labelling will be performed
print(diy_text)
```

Python

```
 we use predominantly C, with some Java, but the reader can still understand the algorithms without a thorough knowledge of these languages. Concepts are presented using intuitive descriptions. Important theoretical results are covered, but formal proofs are omitted. The bibliographical notes at the end of each chapter contain pointers to research papers in which results were first presented and proved, as well as references to material for further reading. In place of proofs, figures and examples are used to suggest why we should expect the result in question to be true. The fundamental concepts and algorithms covered in the book are often based on those used in existing commercial operating systems. Our aim is to present these concepts and algorithms in a general setting that is not tied to one particular operating system. We present a large number of examples that pertain to the most popular and the most innovative operating systems, including Sun Microsystems’ Solaris; Linux; Microsoft Windows 7, Windows 2000, and Windows XP; and Apple Mac OS X. When we refer to Windows XP as an example operating system, we mean Windows XP and Windows 2000. If a feature exists in a specific release, we state this explicitly. The organization of this text reflects our many years of teaching courses on operating systems. Consideration was also given to the feedback provided by the reviewers of the text, as well as comments submitted by readers of earlier editions. In addition, the content of the text corresponds to the suggestions from Computing Curricula 2005 for teaching operating systems, published by the Joint Task Force of the IEEE Computing Society and the Association for Computing Machinery (ACM). On the supporting Web site for this text, we provide several sample syllabi that suggest various approaches for using the text in both introductory and advanced courses. As a general rule, we encourage readers to progress sequentially through the chapters, as this strategy provide
```

Result of Manual Labelling

1. C | PRODUCT
2. Java | PRODUCT
3. first | ORDINAL
4. one | CARDINAL
5. Sun Microsystems Solaris | ORG
6. Linux | PRODUCT
7. Microsoft | ORG
8. Windows XP | PRODUCT
9. Apple Mac OS X | PRODUCT
10. Windows XP | PRODUCT
11. Windows XP | PRODUCT
12. Windows 2000| PRODUCT
13. Computing Curricula 2005 | WORK_OF_ART
14. the Joint Task Force | ORG
15. ACM | ORG

❖ We obtained the above Entities and Entity types from the paragraph we chose for NER.

❖ Now we are going to find the relationships amongst the entities present in the paragraph.

- We can decide the relationship between the entities from the presence of particular constructions in a constituent structure.

Entity	Entity type	Entity	Entity type	Relationship
MICROSOFT	ORGANISATION	WINDOWS XP	PRODUCT	Windows XP ***PartOf*** Microsoft
MICROSOFT	ORGANISATION	WINDOWS 2000	PRODUCT	Windows 2000 ***PartOf*** Microsoft
THE JOINT TASK FORCE	ORGANISATION	COMPUTING CURRICULA 2005	WORK_OF_ART	The Joint Task Force ***PRODUCES*** Computing Curricula 2005
ACM	ORGANISATION	COMPUTING CURRICULA 2005	WORK_OF_ART	ACM ***PRODUCES*** Computing Curricula 2005