

# **Natural Language Processing Practical**

**Subject Code :- 57722**

A Practical Report Submitted in fulfillment of

the Degree of

**MASTER OF SCIENCE**

**In**

**COMPUTER SCIENCE**

**Year 2024 – 2025**

By

**Ms. Mansi Sanjay Jadhav**

**(Application Id :- MU0027920240004612)**

**Seat Number :- 8176101**

Semester II

Under the Guidance of

**Prof. Smita Patil**



Centre for Distance and Online Education

Vidya Nagari, Kalina, Santacruz East 400098.

**University of Mumbai**

PCP Center [Satish Pradhan Dnyanasadhana

College, Thane 400604]



Centre for Distance and Online Education  
Vidya Nagari, Kalina, Santacruz East 400098.

**CERTIFICATE**

This is to certify that

**Ms. Mansi Sanjay Jadhav** , Application Id :- MU0027920240004612 ,

Student of Master of Science in Computer Science has Satisfactorily

Completed the Practical in

**Natural Language Processing**

Name

Ms Mansi Sanjay Jadhav

Application Id

MU0027920224000612

---

Subject In-Charge

---

Examiner

# INDEX

Sr. No.	Practical	Date	Page No.	Sign
1	Write a program to implement Sentence Segmentation & Word Tokenization			
2	Write a program to implement Stemming & Lemmatization			
3	Write a program to implement a Tri-Gram Model			
4	Write a program to implement POS Tagging			
5	Write a program to implement Syntactic Parsing of a given text			
6	Write a program to implement Dependency Parsing of a given text			
7	Write a program to implement Named Entity Recognition (NER)			
8	Write a program to implement Text Summarization for the given sample text			

## Practical 1

**Aim:** Write a program to implement Sentence Segmentation & Word Tokenization

**Theory:** Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

**Sentence Tokenization:** Sentence tokenization is the process of splitting text into individual sentences.

**Word Tokenization:** Word tokenization is the most common version of tokenization. It takes natural breaks, like pauses in speech or spaces in text, and splits the data into its respective words using delimiters (characters like ‘,’ or ‘;’ or ““,””). While this is the simplest way to separate speech or text into its parts.

**Modules:** NLTK contains a module called tokenize() which further classifies into two sub-categories:

- Word tokenize: We use the word\_tokenize() method to split a sentence into tokens or words.
- Sentence tokenize: We use the sent\_tokenize() method to split a document or paragraph into sentences

**Code:**

```
#pip install nltk
```

```
import nltk
```

```
s=open("sadiq.txt")
```

```
content=s.read()
```

```
sentences=nltk.sent_tokenize(content)
```

```
for sentence in sentences:
```

```
    print("\nSentence is:",sentence)
```

```
    print("\nTokens are:",nltk.word_tokenize(sentence))
```

```
    print()
```

## Output:

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\SADIQ\MSc\SEM 2\NLP\PRAC\Prac 1\a.py =====

Sentence is: We used to be close, but people can go.

Tokens are: ['We', 'used', 'to', 'be', 'close', ',', 'but', 'people', 'can', 'go', '.', '.']

Sentence is: From people you know to people you don't.

Tokens are: ['From', 'people', 'you', 'know', 'to', 'people', 'you', 'do', 'n't', '.', '.']

Sentence is: And what hurts the most is people can go.

Tokens are: ['And', 'what', 'hurts', 'the', 'most', 'is', 'people', 'can', 'go', '.', '.']

Sentence is: From people you know to people you don't.

Tokens are: ['From', 'people', 'you', 'know', 'to', 'people', 'you', 'do', 'n't', '.', '.']
>>>
```

## Practical 2

**Aim:** Write a program to Implement Stemming & Lemmatization.

### Theory:

#### What is Stemming?

Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat.

Search engines use stemming for indexing the words. That's why rather than storing all forms of a word, a search engine can store only the stems. In this way, stemming reduces the size of the index and increases retrieval accuracy.

#### Modules:

NLTK has **PorterStemmer** class with the help of which we can easily implement Porter Stemmer algorithms for the word we want to stem. This class knows several regular word forms and suffixes with the help of which it can transform the input word to a final stem.

NLTK has **LancasterStemmer** class with the help of which we can easily implement Lancaster Stemmer algorithms for the word we want to stem.

NLTK has **SnowballStemmer** class with the help of which we can easily implement Snowball Stemmer algorithms. It supports 15 non-English languages. In order to use this stemming class, we need to create an instance with the name of the language we are using and then call the stem() method.

#### What is Lemmatization?

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.

Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meanings to one word.

Text pre-processing includes both Stemming as well as Lemmatization.

Many times, people find these two terms confusing. Some treat these two as the same. Actually, lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

**Code:****Stemming:**

```
import nltk  
nltk.download('averaged_perceptron_tagger')  
from nltk.stem import PorterStemmer  
from nltk.stem import SnowballStemmer  
from nltk.stem import LancasterStemmer  
words = ['run', 'runner', 'running', 'ran', 'runs', 'easily', 'fairly']
```

```
def portstemming (words):  
    ps=PorterStemmer()  
    print("Porter Stemmer")  
    for word in words:  
        print(word, "--->", ps.stem(word))
```

```
def snowballstemming (words):  
    snowball = SnowballStemmer (language='english')  
    print("Snowball Stemmer")  
    for word in words:  
        print(word, "--->", snowball.stem(word))
```

```
def lancasterstemming (words):  
    lancaster = LancasterStemmer()  
    print("Lancaster Stemmer")  
    for word in words:  
        print(word, "--->", lancaster.stem(word))
```

```
print("Select operation.")
print("1.Porter Stemmer")
print("2.Snowball Stemmer")
print("3. Lancaster Stemmer")

while True:
    choice = input("Enter choice (1/2/3): ")
    if choice in ('1', '2', '3'):
        if choice == '1':
            print(portstemming(words))
        elif choice == '2':
            print(snowballstemming (words))
        elif choice == '3':
            print(lancasterstemming (words))

        next_calculation = input("Do you want to do stemming again? (yes/no): ")
        if next_calculation == "no":
            break
    else:
        print("Invalid Input")
```



## Output:

```
Select operation.
1.Porter Stemmer
2.Snowball Stemmer
3.Lancaster Stemmer
Enter choice (1/2/3): 2
Snowball Stemmer
run ---> run
runner ---> runner
running ---> run
ran ---> ran
runs ---> run
easily ---> easili
fairly ---> fair
None
Do you want to do stemming again? (yes/no): yes
Enter choice (1/2/3): 1
Porter Stemmer
run ---> run
runner ---> runner
running ---> run
ran ---> ran
runs ---> run
easily ---> easili
fairly ---> fairli
None
Do you want to do stemming again? (yes/no): yes
Enter choice (1/2/3): 3
Lancaster Stemmer
run ---> run
runner ---> run
running ---> run
ran ---> ran
runs ---> run
easily ---> easy
fairly ---> fair
None
Do you want to do stemming again? (yes/no): no
```

### **Lemmatization:**

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = input("Enter words for lemmatizing: ")
tokenization = nltk.word_tokenize(text)

for w in tokenization:
    print("Lemma for {} is {}".format(w,wordnet_lemmatizer.lemmatize(w,'v')))
```

### **Output:**

```
>>> Enter words for lemmatizing: punching eaten listened cries
      Lemma for punching is punch
      Lemma for eaten is eat
      Lemma for listened is listen
      Lemma for cries is cry
```

## Practical 3

**Aim:** Write a program to implement a Tri-Gram Model

**Theory:**

**Tri-Gram Model:**

- A trigram model is a statistical language model used in natural language processing (NLP) that predicts the probability of the next word in a sequence, given the two previous words.
- It is based on the Markov assumption that the probability of a word depends only on the preceding two words, and not on any earlier context.
- The trigram model can be used in various NLP tasks such as language modeling, speech recognition, machine translation, and text classification.
- It is a simple yet effective model that can capture some of the syntactic and semantic dependencies between words in a sentence.
- However, it suffers from the data sparsity problem, as the number of distinct trigrams in a large corpus can be very large, and many of them may not occur at all.
- Various techniques such as smoothing and backoff can be used to address this problem.

**Text File:**

```
Natural Language Processing refers to AI  
method of communicating using a natural  
language. Natural Language examples are english.
```

**Code:**

```
import nltk  
  
nltk.download('punkt')  
  
from nltk.tokenize import word_tokenize  
  
from nltk import FreqDist  
  
import pandas as pd
```

```
f = open("sadiq.txt")
sample = f.read()

sample_tokens = nltk.word_tokenize(sample)
print('\n Sample Tokens:',sample_tokens)
print('\n Type of Sample Tokens:',type(sample_tokens))
print('\n Length of Sample Tokens:',len(sample_tokens))

sample_freq =FreqDist(sample_tokens)
tokens=[]
sf=[]

for i in sample_freq:
    tokens.append(i)
    sf.append(sample_freq[i])

df = pd.DataFrame({'Tokens':tokens,'Frequency':sf})
print('\n',df)

print('\n Bigrams:',list(nltk.bigrams(sample_tokens)))
print('\n Trigrams:',list(nltk.trigrams(sample_tokens)))
print('\n N-grams(4):',list(nltk.ngrams(sample_tokens,4)))
```

## Output:

```
Sample Tokens: ['Natural', 'Language', 'Processing', 'refers', 'to', 'AI', 'method', 'of', 'communicating', 'using', 'a', 'natural', 'language', '.', 'Natural', 'Language', 'examples', 'are', 'english', '.']
```

```
Type of Sample Tokens: <class 'list'>
```

```
Length of Sample Tokens: 20
```

	Tokens	Frequency
0	Natural	2
1	Language	2
2	.	2
3	Processing	1
4	refers	1
5	to	1
6	AI	1
7	method	1
8	of	1
9	communicating	1
10	using	1
11	a	1
12	natural	1
13	language	1
14	examples	1
15	are	1
16	english	1

```
Bigrams: [('Natural', 'Language'), ('Language', 'Processing'), ('Processing', 'refers'), ('refers', 'to'), ('to', 'AI'), ('AI', 'method'), ('method', 'of'), ('of', 'communicating'), ('communicating', 'using'), ('using', 'a'), ('a', 'natural'), ('natural', 'language'), ('language', '.'), ('.', 'Natural'), ('Natural', 'Language'), ('Language', 'examples'), ('examples', 'are'), ('are', 'english'), ('english', '.')]

Trigrams: [('Natural', 'Language', 'Processing'), ('Language', 'Processing', 'refers'), ('Processing', 'refers', 'to'), ('refers', 'to', 'AI'), ('to', 'AI', 'method'), ('AI', 'method', 'of'), ('method', 'of', 'communicating'), ('of', 'communicating', 'using'), ('communicating', 'using', 'a'), ('using', 'a', 'natural'), ('a', 'natural', 'language'), ('natural', 'language', '.'), ('language', '.', 'Natural'), ('.', 'Natural', 'Language'), ('Natural', 'Language', 'examples'), ('Language', 'examples', 'are'), ('examples', 'are', 'english'), ('are', 'english', '.')]

N-grams(4): [('Natural', 'Language', 'Processing', 'refers'), ('Language', 'Processing', 'refers', 'to'), ('Processing', 'refers', 'to', 'AI'), ('refers', 'to', 'AI', 'method'), ('to', 'AI', 'method', 'of'), ('AI', 'method', 'of', 'communicating'), ('method', 'of', 'communicating', 'using'), ('of', 'communicating', 'using', 'a'), ('communicating', 'using', 'a', 'natural'), ('using', 'a', 'natural', 'language'), ('a', 'natural', 'language', '.'), ('natural', 'language', '.', 'Natural'), ('language', '.', 'Natural', 'Language'), ('.', 'Natural', 'Language', 'examples'), ('Natural', 'Language', 'examples', 'are'), ('Language', 'examples', 'are', 'english'), ('examples', 'are', 'english', '.')]

```

## **Practical 4**

**Aim:** Write a program to Implement POS Tagging.

**Theory:**

**What is POS Tagging?**

- Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of-speech, semantic information and so on.
- Part-of-Speech (PoS) tagging may be defined as the process of assigning one of the parts of speech to the given word. It is generally called POS tagging.
- In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech.
- We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub-categories.
- Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging and Transformation based tagging.

**Rule-based POS Tagging:**

One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag.

**Stochastic POS Tagging:**

The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.

**Transformation-based Tagging:**

Transformation based tagging is also called Brill tagging. It is an instance of the transformation-based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

**Code:**

```
import nltk
from collections import Counter
text="Guru99 is one of the best sites to learn WEB,SAP,Ethical Hacking and
much more online.My name is Sadiq. And I'm working on a project"
```

```
lower_case=text.lower()
tokens=nltk.word_tokenize(lower_case)
tags=nltk.pos_tag(tokens)
print(tags)
counts=Counter(tag for word,tag in tags)
```

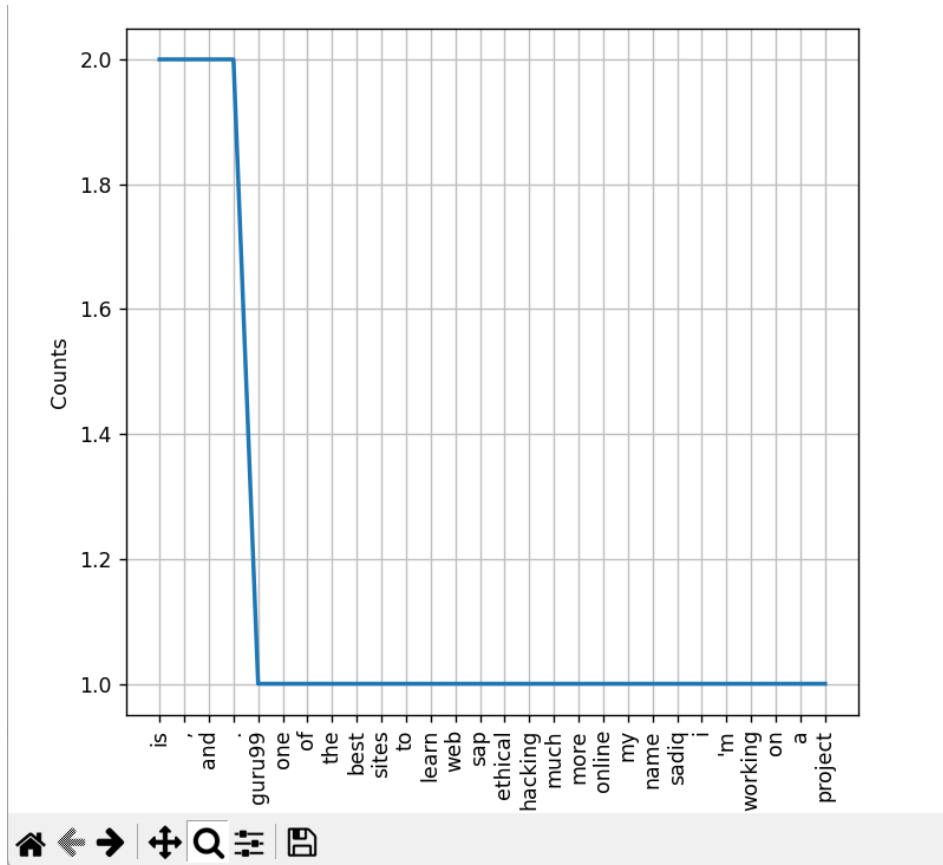
```
###for tag in tags:
### print(tag)
print(counts)
fd=nltk.FreqDist(tokens)
fd.plot()
```

```
fd1=nltk.FreqDist(counts)
fd1.plot()
```

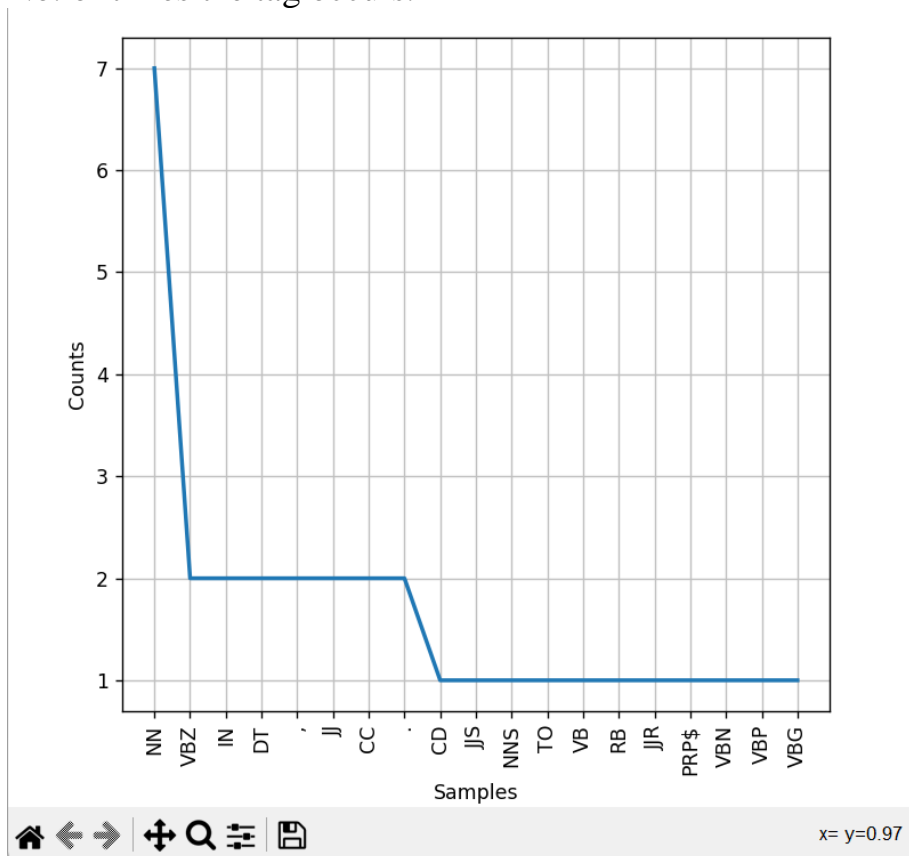
**Output:**

```
===== RESTART: D:\SADIQ\MSc\SEM 2\NLP\PRAC\Prac 4\prac4.py =====
[('guru99', 'NN'), ('is', 'VBZ'), ('one', 'CD'), ('of', 'IN'), ('the', 'DT'), ('best', 'JJS'), ('sites', 'NNS'), ('to', 'TO'), ('learn', 'VB'), ('web', 'NN'), ('', ' '), ('sap', 'NN'), ('', ' '), ('ethical', 'JJ'), ('hacking', 'NN'), ('and', 'CC'), ('much', 'RB'), ('more', 'JJR'), ('online.my', 'JJ'), ('name', 'NN'), ('is', 'VBZ'), ('sadiq', 'VBN'), ('.', '.'), ('and', 'CC'), ('i', 'JJ'), ('m', 'VBP'), ('working', 'VBG'), ('on', 'IN'), ('a', 'DT'), ('project', 'NN')]
Counter({'NN': 6, 'JJ': 3, 'VBZ': 2, 'IN': 2, 'DT': 2, ' ': 2, 'CC': 2, 'CD': 1, 'JJS': 1, 'NNS': 1, 'TO': 1, 'VB': 1, 'RB': 1, 'JJR': 1, 'VBN': 1, '.': 1, 'VBP': 1, 'VBG': 1})
```

No. of times the word occurs.



No. of times the tag occurs.





## Practical 5

**Aim:** Write a program to Implement Syntactic Parsing of a given text

### Theory:

#### What is Syntactic Parsing?

- Syntactic analysis or parsing or syntax analysis is the third phase of NLP.
- The purpose of this phase is to draw exact meaning, or you can say dictionary meaning from the text.
- Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar.

#### Concept of Parser

- It is used to implement the task of parsing. It may be defined as the software component designed for taking input data (text) and giving structural representation of the input after checking for correct syntax as per formal grammar.
- It also builds a data structure generally in the form of parse tree or abstract syntax tree or other hierarchical structure.

### Code:

```
import nltk

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

from nltk import pos_tag, word_tokenize, RegexpParser

sample_text="My little sister is playing with our pet cat and I am working"

tagged=pos_tag(word_tokenize(sample_text))

#Extract all parts of speech from any text

chunker =RegexpParser("""NP:{<DT>?<JJ>*<NN>}

                        P:{<IN>}

                        V:{<V.*>}

                        PP:{<P> <NP>}
```

```

VP:{<V> <NP|PP> *}
""")

```

#print all parts of speech in above sentence.

```
output = chunker.parse(tagged)
```

```
print("After Extracting\n",output)
```

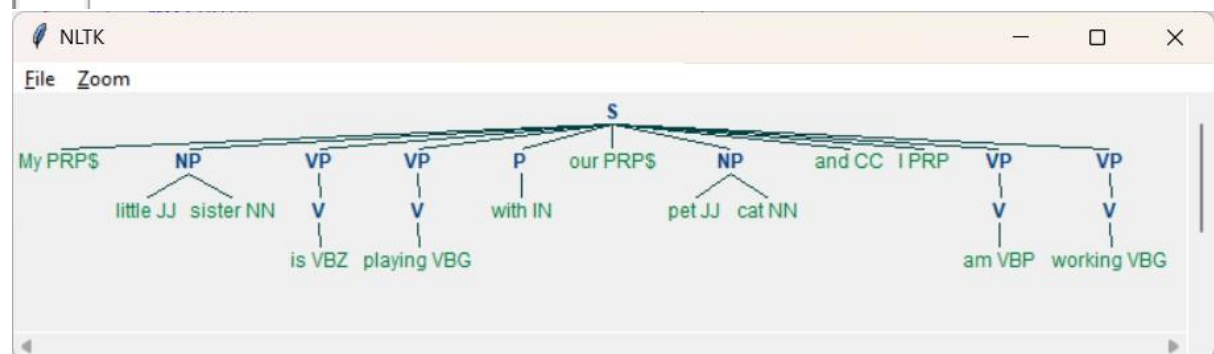
```
output.draw()
```

## Output:

```

[('My', 'PRP$', 'My'), ('little', 'JJ', 'little'), ('sister', 'NN', 'sister'), ('is', 'VBZ', 'is'), ('playing', 'VBG', 'playing'), ('with', 'IN', 'with'), ('our', 'PRP$', 'our'), ('pet', 'JJ', 'pet'), ('cat', 'NN', 'cat'), ('and', 'CC', 'and'), ('I', 'PRP', 'I'), ('am', 'VBP', 'am'), ('working', 'VBG', 'working')]
After Extracting
(S
  My/PRP$
  (NP little/JJ sister/NN)
  (VP (V is/VBZ))
  (VP (V playing/VBG))
  (P with/IN)
  our/PRP$
  (NP pet/JJ cat/NN)
  and/CC
  I/PRP
  (VP (V am/VBP))
  (VP (V working/VBG)))

```



## Practical 6

**Aim:** Write a program to Implement Dependency Parsing of a given text

### Theory:

#### Dependency Parsing:

The term Dependency Parsing (DP) refers to the process of examining the dependencies between the phrases of a sentence in order to determine its grammatical structure.

A sentence is divided into many sections based mostly on this. The process is based on the assumption that there is a direct relationship between each linguistic unit in a sentence. These hyperlinks are called dependencies.

#### Module:

spaCy is a library for advanced Natural Language Processing in Python and Cython.

### Code:

```
#pip install -U spacy
#python -m spacy download en_core_web_sm

import spacy

from spacy import displacy

nlp = spacy.load("en_core_web_sm")

sentence = 'Deemed universitiies charge huge fees'

doc =nlp(sentence)

print("{:<15}|{:<8}|{:<15}|{:<20}".format('Token','Relation','Head','Children'))

print('-'*70)

for token in doc:

    print("{:15}|{:<8}|{:<15}|{:<20}".format(str(token.text),str(token.dep_),
str(token.head.text),str([child for child in token.children])))

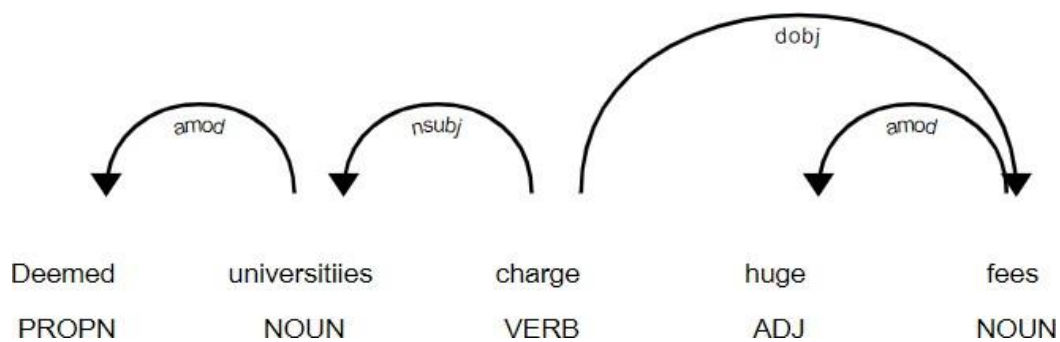
# Use displacy to visualize the dependency

displacy.serve(doc,style='dep',options={'distance':120})
```

## Output:

As output was not properly displayed in IDLE, I have used jupyter notebook.

Token	Relation	Head	Children
Deemed	amod	universitiies	[]
universitiies	nsubj	charge	[Deemed]
charge	ROOT	charge	[universitiies, fees]
huge	amod	fees	[]
fees	dobj	charge	[huge]



## Practical 7

**Aim:** Write a program to implement Named Entity Recognition (NER)

### Theory:

#### What is NER?

Named-entity recognition is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into predefined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

#### What is NER used for?

Named entity recognition (NER) helps you easily identify the key elements in a text, like names of people, places, brands, monetary values, and more.

#### How do NER work?

Named Entity Recognition is a process where an algorithm takes a string of text (sentence or paragraph) as input and identifies relevant nouns (people, places, and organizations) that are mentioned in that string.

#### What is spacy used for?

spacy is a free, open-source library for NLP in Python. It's written in Cython and is designed to build information extraction or natural language understanding systems.

### Code:

```
import spacy
```

```
import pandas as pd
```

```
from spacy import displacy
```

```
NER = spacy.load("en_core_web_sm")
```

```
text = "The Indian Space Reasearch Organisation or is the national space agency  
in India. Established on 24th July 2023 and operates 99% space program"
```

```
doc = NER(text)
```

```
entities = []
```

```
labels = []
```

```

position_start = []
position_end = []

for ent in doc.ents:
    entities.append(ent)
    labels.append(ent.label_)
    position_start.append(ent.start_char)
    position_end.append(ent.end_char)

df = pd.DataFrame({'Entities': entities, 'Labels': labels, 'Position_Start':
position_start, 'Position_End': position_end})

print(df)

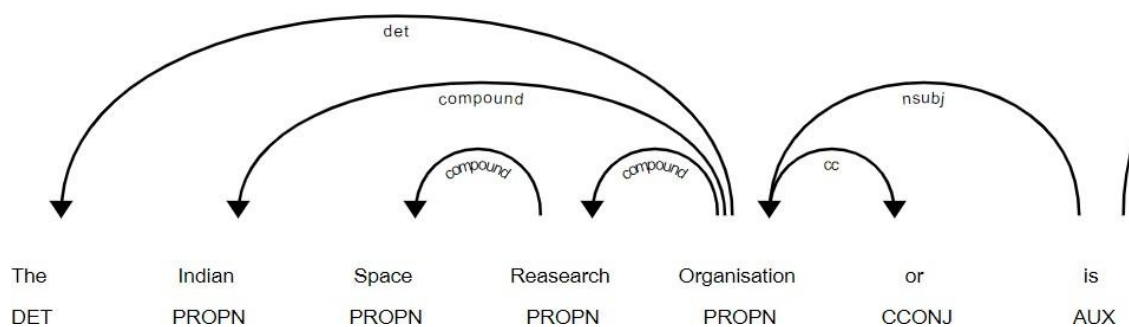
displacy.serve(doc, style='dep', options={'distance': 120})
displacy.render(doc, style="ent")

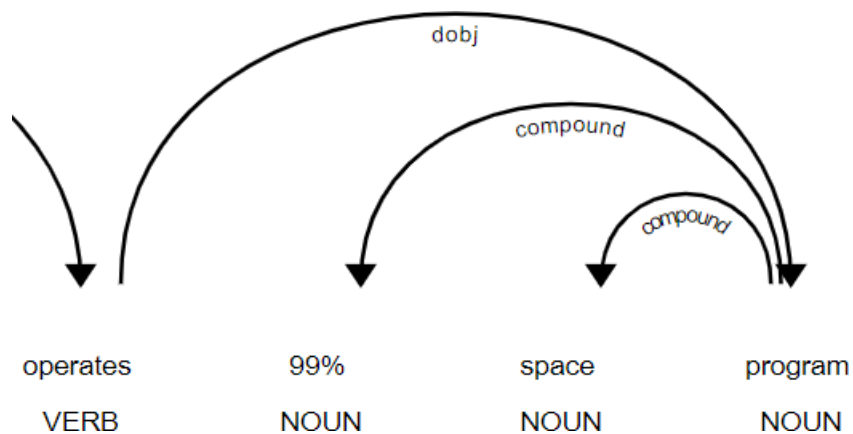
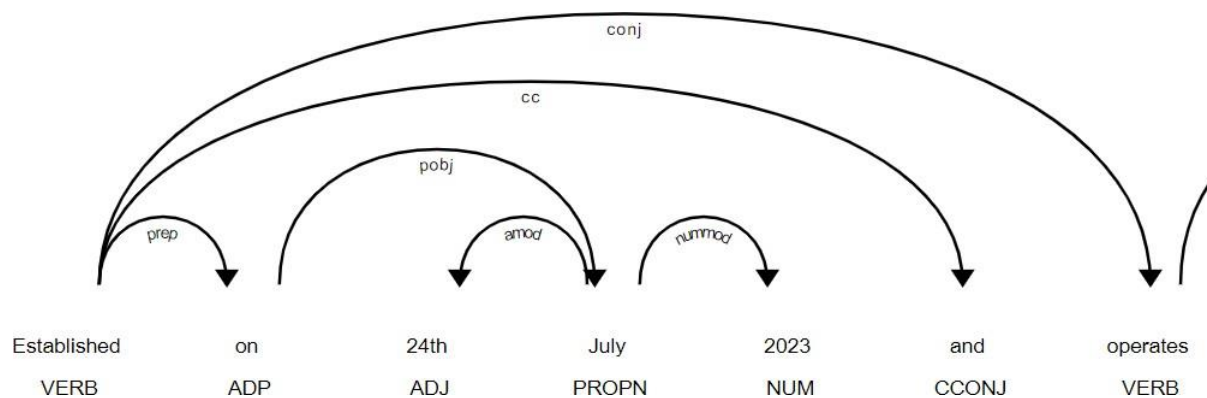
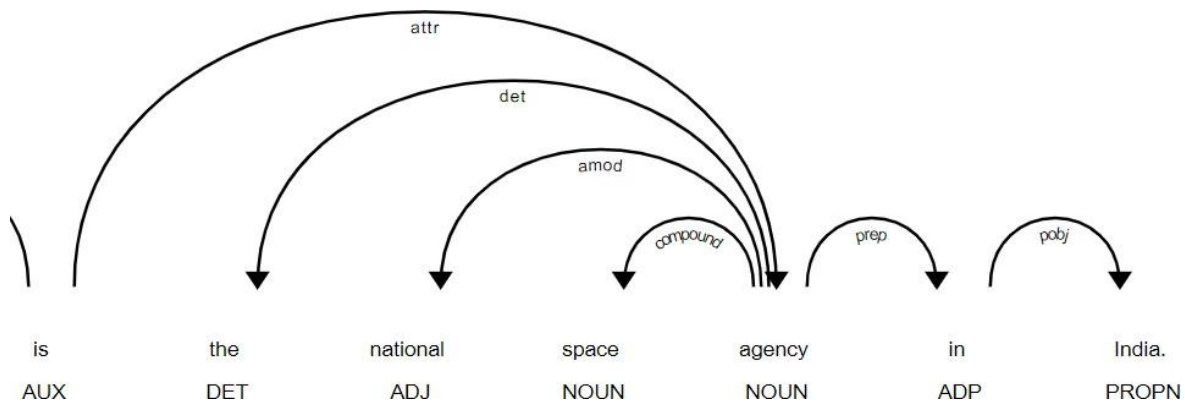
```

## Output:

As output was not properly displayed in IDLE, I have used jupyter notebook.

	Entities	Labels	Position_Start	Position_End
0	(The, Indian, Space, Reasearch, Organisation)	ORG	0	39
1	(India)	GPE	75	80
2	(24th, July, 2023)	DATE	97	111
3	(99, %)	PERCENT	125	128





## Practical 8

**Aim:** Write a program to Implement Text Summarization for the given sample text.

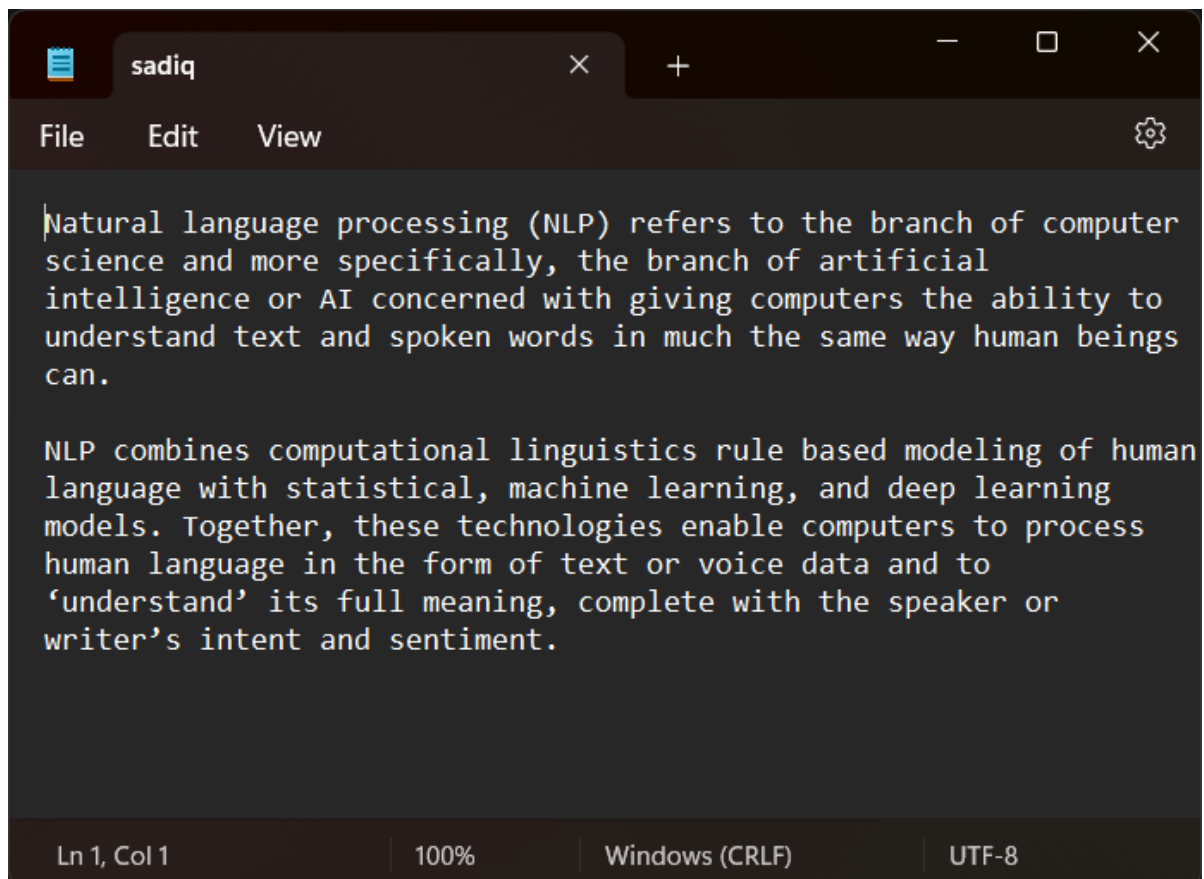
### Theory:

Text summarization is the process of creating a shorter version of a longer text while preserving its main points and essential meaning. The goal of text summarization is to reduce the amount of time and effort required to understand the content of a text, making it easier to digest and extract the key information.

There are two main types of text summarization: extractive and abstractive. Extractive summarization involves selecting and combining the most important sentences or phrases from the original text, while abstractive summarization involves generating new sentences that capture the meaning of the original text.

Text summarization is used in various fields, including news and media, research, and education. It can be done manually or with the help of automated tools and algorithms, such as natural language processing (NLP) techniques and machine learning models.

### Text File:



The screenshot shows a text editor window with a dark theme. The window title is 'sadiq'. The menu bar includes 'File', 'Edit', 'View', and a settings icon. The text content is as follows:

```
Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
```

```
NLP combines computational linguistics rule based modeling of human language with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment.
```

The status bar at the bottom shows 'Ln 1, Col 1', '100%', 'Windows (CRLF)', and 'UTF-8'.



**Code:**

```
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.corpus import stopwords
```

```
f = open("sadiq.txt")
text = f.read()
```

```
words = word_tokenize(text)
sents = sent_tokenize(text)
stopwords = set(stopwords.words('english'))
```

```
freqTable = dict()
```

```
for word in words:
    word = word.lower()
    if word in stopwords:
        continue
    elif word in freqTable:
        freqTable[word]+=1
    else:
        freqTable[word]=1
```

```
sentValue = dict()
```

```
for sent in sents:
    for word,freq in freqTable.items():
        if word in sent.lower():
```

```

        if sent in sentValue:
            sentValue[sent]+=freq
        else:
            sentValue[sent]=freq

sumValues = 0
for s in sentValue:
    sumValues+=sentValue[s]

avg = int(sumValues/len(sents))

summary = "
for sent in sents:
    if (sent in sentValue) and (sentValue[sent]>1.2*avg):
        summary+=" "+sent
    print(summary)

```

### Output:

```

===== RESTART: D:\SADIQ\MSC\SEM 2\NLP\PRAC\Prac 8\prac8.py =====
=====
Natural language processing (NLP) refers to the branch of computer science
and more specifically, the branch of artificial intelligence or AI concerne
d with giving computers the ability to understand text and spoken words in
much the same way human beings can.
..

```