INDEX

PRACTICAL NO: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 intall nltk
#py -m pip install --upgrade pip
#nltk.download('punkt')
#nltk.download('wordnet')

from nltk.tokenize import word_tokenize
text="God is Great! I won a lottery."
print("The words are",word_tokenize(text))

from nltk.tokenize import sent_tokenize
text="God is Great! I won a lottery." print("The
sentences are",sent_tokenize(text))
```

OUTPUT:

```
The words are ['God', 'is', 'Great', '!', 'I', 'won', 'a', 'lottery', '.']
The sentences are ['God is Great!', 'I won a lottery.']
```

PRACTICAL NO: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 steaming 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 preprocessing 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.

Applications of lemmatization are:

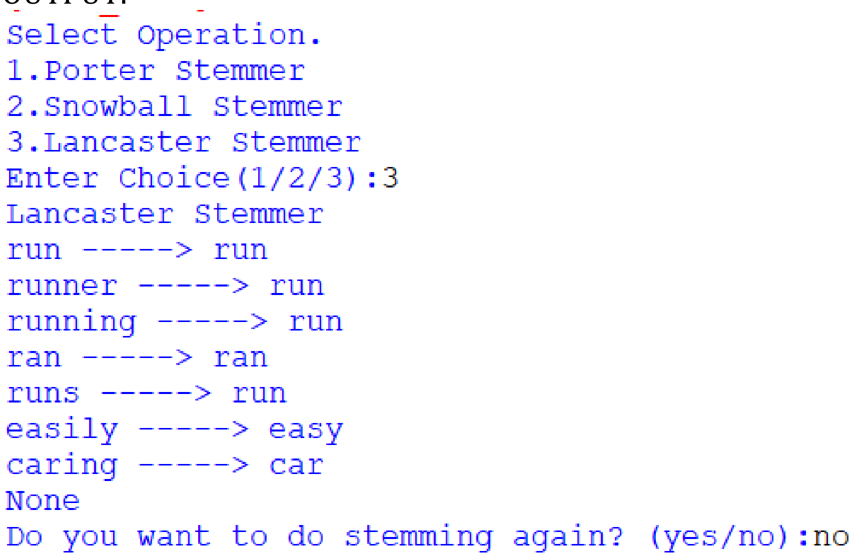
- ✔ Used in comprehensive retrieval systems like search engines.
- ✔ Used in compact indexing

- ✔ STEMMING
  :

CODE:
```
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','caring']

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:



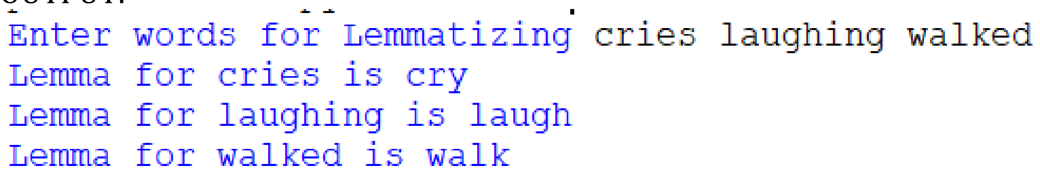
```
Select Operation.
1.Porter Stemmer
2.Snowball Stemmer
3.Lancaster Stemmer
Enter Choice(1/2/3):3
Lancaster Stemmer
run -----> run
runner -----> run
running -----> run
ran -----> ran
runs -----> run
easily -----> easy
caring -----> car
None
Do you want to do stemming again? (yes/no):no
```

✔ Lemmatization

CODE:

```
import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text =input("Enter words for Lemmatizing")
tokenization= nltk.word_tokenize(text)

# v verb, a adjective, n noun in lemmatize parameter
for w in tokenization:
   print("Lemma for {} is {}".format(w,wordnet_lemmatizer.lemmatize(w,'v')))
```

OUTPUT:



```
Enter words for Lemmatizing cries laughing walked
Lemma for cries is cry
Lemma for laughing is laugh
Lemma for walked is walk
```

PRACTICAL NO: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:

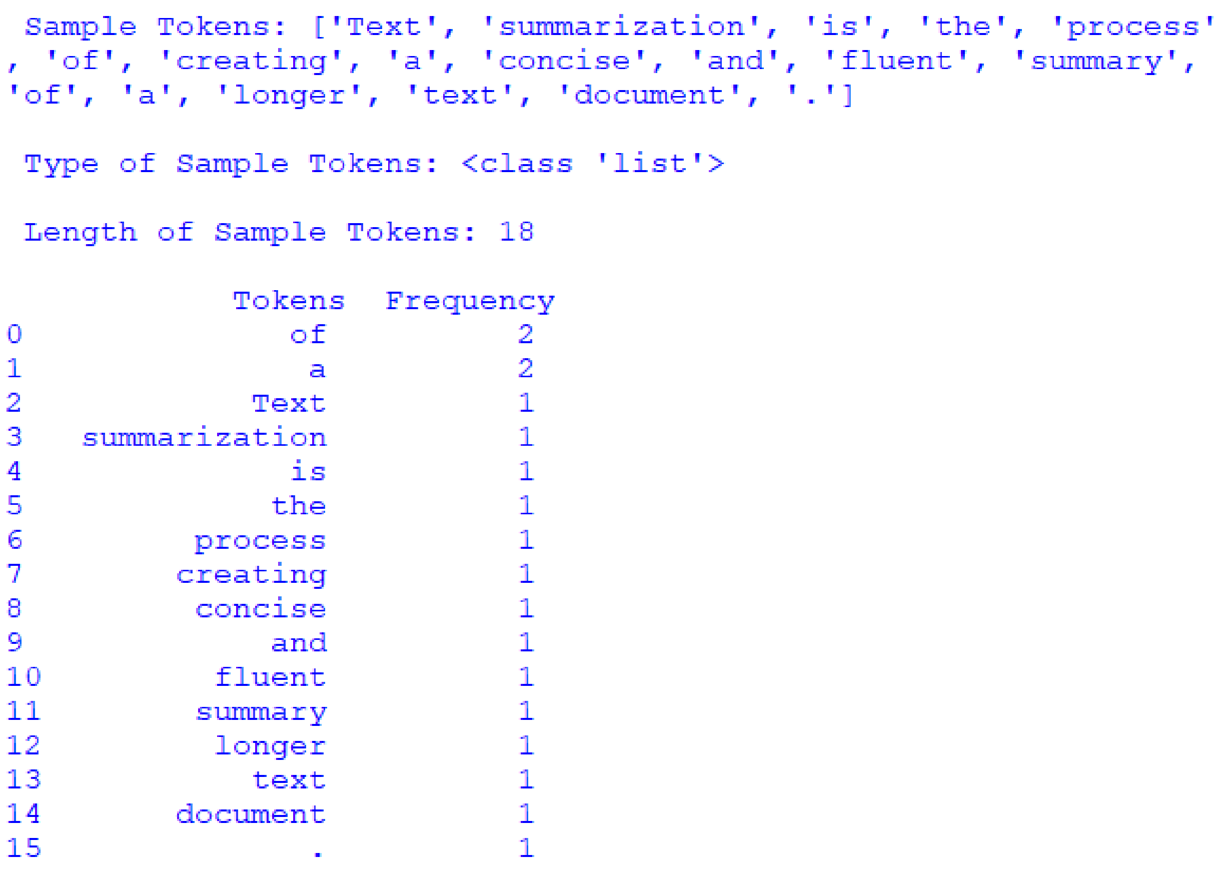
```
Text summarization is the process of
creating a concise and fluent summary
of a longer text document.
```

CODE:

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

f = open("D:/salma/salma1.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: ['Text', 'summarization', 'is', 'the', 'process'
, 'of', 'creating', 'a', 'concise', 'and', 'fluent', 'summary',
'of', 'a', 'longer', 'text', 'document', '.']

 Type of Sample Tokens: <class 'list'>

 Length of Sample Tokens: 18

            Tokens  Frequency
0               of          2
1                a          2
2             Text          1
3    summarization          1
4               is          1
5              the          1
6          process          1
7         creating          1
8          concise          1
9              and          1
10          fluent          1
11         summary          1
12          longer          1
13            text          1
14        document          1
15               .          1
```

```
 Bigrams: [('Text', 'summarization'), ('summarization', 'is'), (
'is', 'the'), ('the', 'process'), ('process', 'of'), ('of', 'cre
ating'), ('creating', 'a'), ('a', 'concise'), ('concise', 'and')
, ('and', 'fluent'), ('fluent', 'summary'), ('summary', 'of'), (
'of', 'a'), ('a', 'longer'), ('longer', 'text'), ('text', 'docum
ent'), ('document', '.')]

 Trigrams: [('Text', 'summarization', 'is'), ('summarization', '
is', 'the'), ('is', 'the', 'process'), ('the', 'process', 'of'),
('process', 'of', 'creating'), ('of', 'creating', 'a'), ('creati
ng', 'a', 'concise'), ('a', 'concise', 'and'), ('concise', 'and'
, 'fluent'), ('and', 'fluent', 'summary'), ('fluent', 'summary',
'of'), ('summary', 'of', 'a'), ('of', 'a', 'longer'), ('a', 'lon
ger', 'text'), ('longer', 'text', 'document'), ('text', 'documen
t', '.')]

 N-grams(4): [('Text', 'summarization', 'is', 'the'), ('summariz
ation', 'is', 'the', 'process'), ('is', 'the', 'process', 'of'),
('the', 'process', 'of', 'creating'), ('process', 'of', 'creatin
g', 'a'), ('of', 'creating', 'a', 'concise'), ('creating', 'a',
'concise', 'and'), ('a', 'concise', 'and', 'fluent'), ('concise'
, 'and', 'fluent', 'summary'), ('and', 'fluent', 'summary', 'of'
), ('fluent', 'summary', 'of', 'a'), ('summary', 'of', 'a', 'lon
ger'), ('of', 'a', 'longer', 'text'), ('a', 'longer', 'text', 'd
ocument'), ('longer', 'text', 'document', '.')]
```

PRACTICAL NO: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

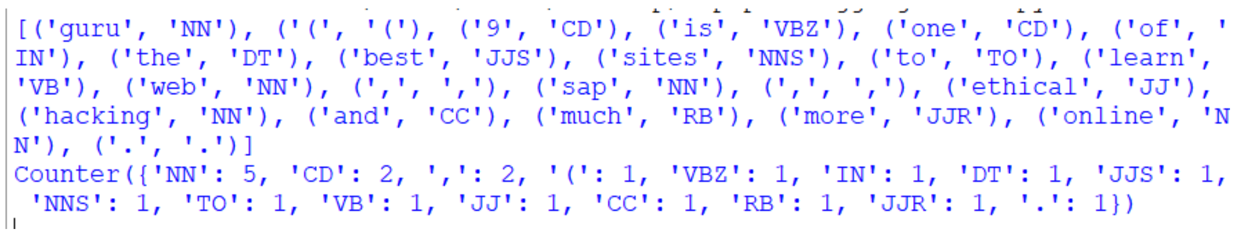
Transformation based tagging is also called Brill tagging. It is an instance of the transformation-based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state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

CODE:

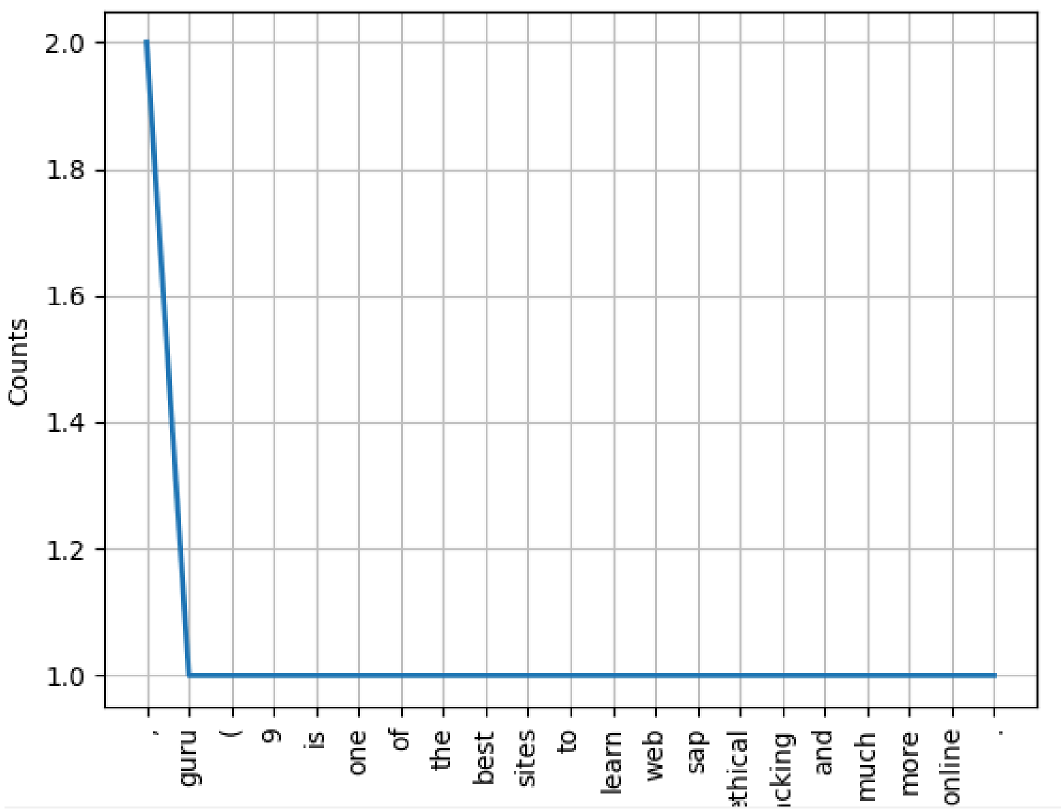
```
import nltk
from collections import Counter

text="Guru(9 is one of the best sites to learn WEB,SAP,Ethical Hacking and much more online."
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)
```

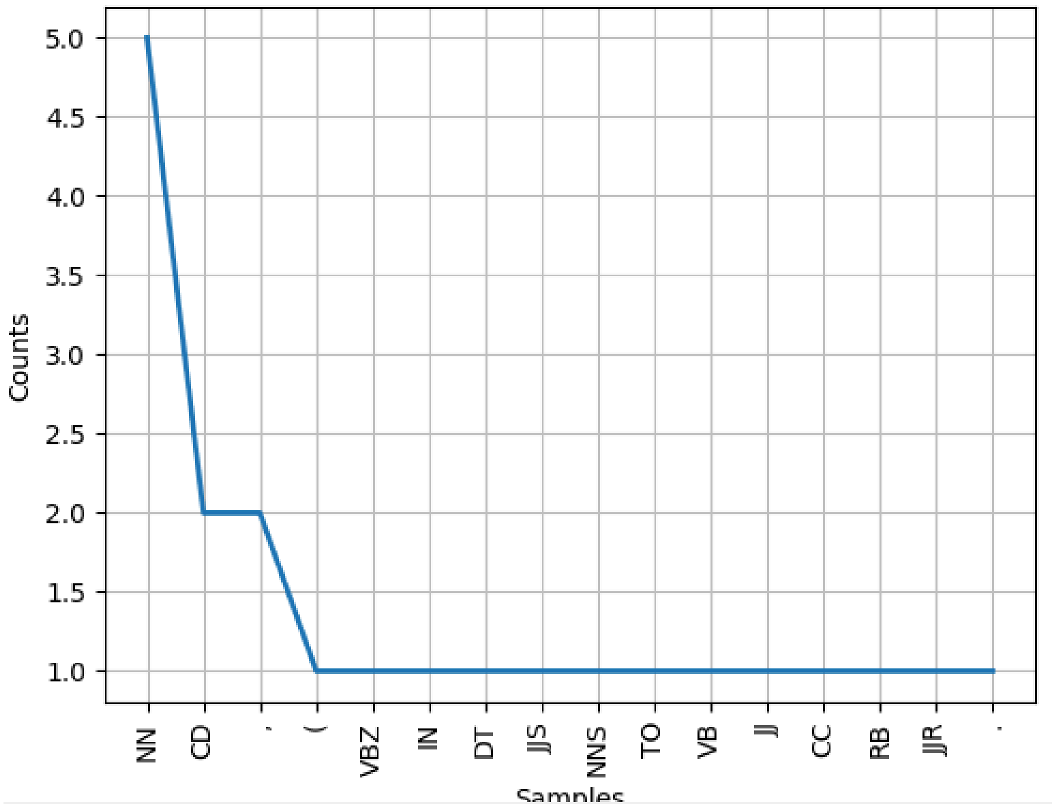
```
###for tag in tags:
### print(tag)
print(counts)

fd=nltk.FreqDist(tokens)
fd.plot()

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

OUTPUT:



```
[('guru', 'NN'), ('(', '('), ('9', 'CD'), ('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', 'N
N'), ('.', '.')]
Counter({'NN': 5, 'CD': 2, ',': 2, '(': 1, 'VBZ': 1, 'IN': 1, 'DT': 1, 'JJS': 1,
 'NNS': 1, 'TO': 1, 'VB': 1, 'JJ': 1, 'CC': 1, 'RB': 1, 'JJR': 1, '.': 1})
```




5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Counts
NN
CD
,
(
VBZ
IN
DT
JJS
NNS
TO
VB
JJ
CC
RB
JJR
.
Samples

PRACTICAL NO: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

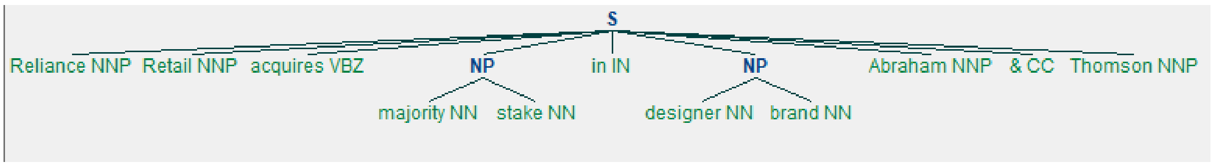
It is used to implement the task of parsing. It may be defined as the software component designed for taking input data (text) and giving structural representation of the input after checking for correct syntax as per formal grammar. It also builds a data structure generally in the form of parse tree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

CODE:

```
import nltk
nltk.download('averaged_perceptron_tagger'
)

from nltk import pos_tag,word_tokenize,RegexpParser
sentence='Reliance Retail acquires majority stake in designer brand Abraham &
Thomson'
tokens=word_tokenize(sentence)
tags=pos_tag(tokens)
grammer="NP:{<NN>?<DT>*<NN>}"
chunker=RegexpParser(grammer)
result=chunker.parse(tags)
print(result)
result.draw()
```

OUTPUT:

```
(S
  Reliance/NNP
  Retail/NNP
  acquires/VBZ
  (NP majority/NN stake/NN)
  in/IN
  (NP designer/NN brand/NN)
  Abraham/NNP
  &/CC
  Thomson/NNP)
```



PRACTICAL NO: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.

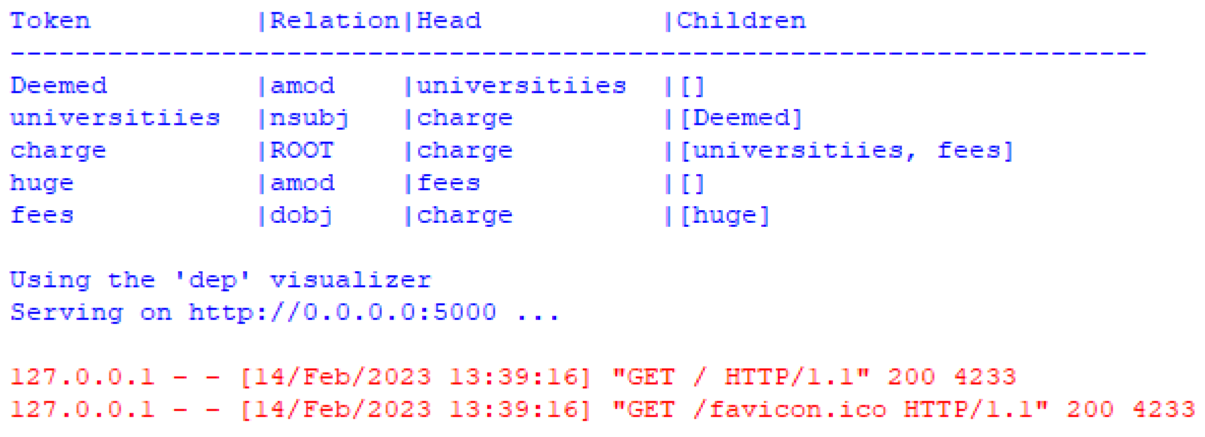
Module:

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

CODE:

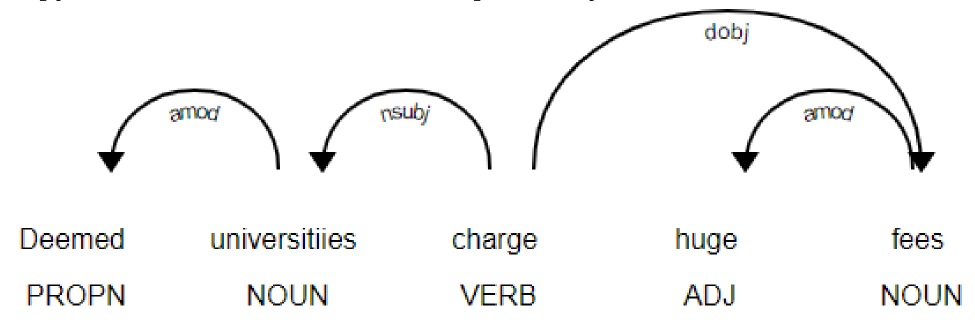
```
#spacy download en_core_web_sm (install on cmd)
import spacy
from spacy import displacy

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:



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

Using the 'dep' visualizer
Serving on http://0.0.0.0:5000 ...

127.0.0.1 - - [14/Feb/2023 13:39:16] "GET / HTTP/1.1" 200 4233
127.0.0.1 - - [14/Feb/2023 13:39:16] "GET /favicon.ico HTTP/1.1" 200 4233
```

Copy the URL on web to see the dependency



PRACTICAL NO: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.

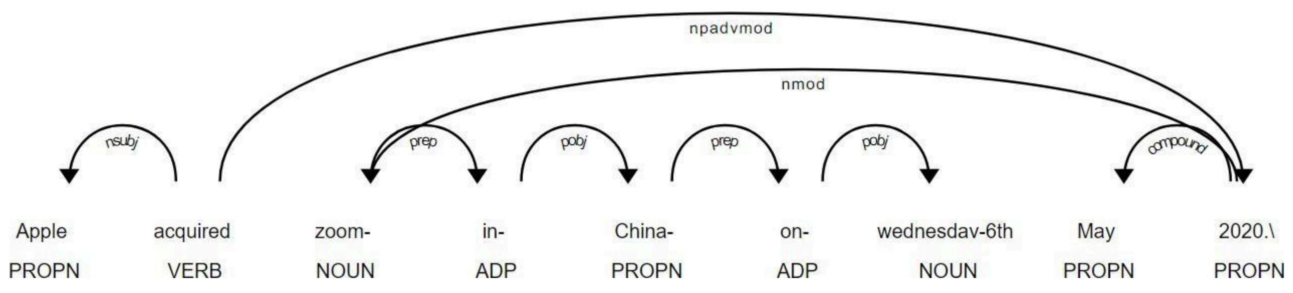
CODE:

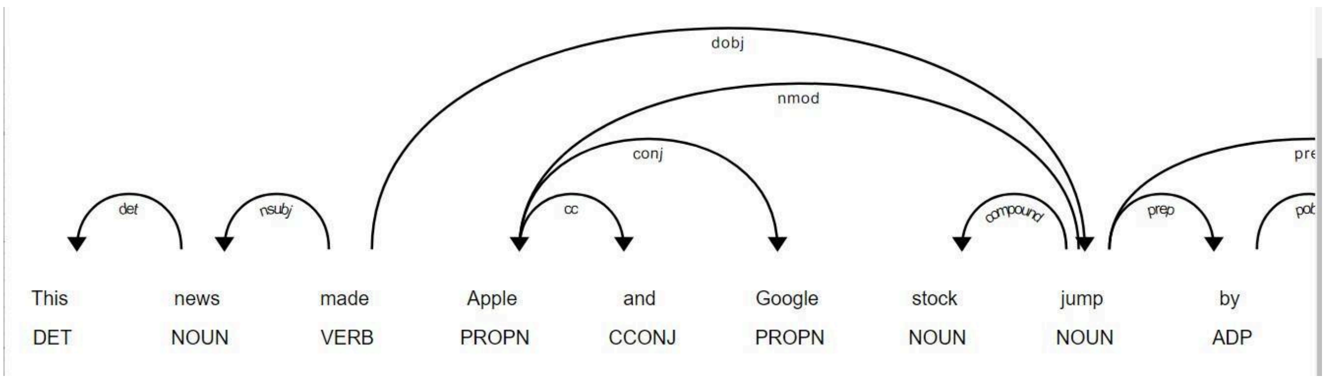
```
import spacy
import pandas as
pd
from spacy import displacy

NER = spacy.load("en_core_web_sm")
text = "Apple acquired zoom-in-China-on-wednesdav-6th May 2020.\ This news made
Apple and Google stock jump by 5% on Dow Jones Indexin -the \ United Starps of
America"

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})
```

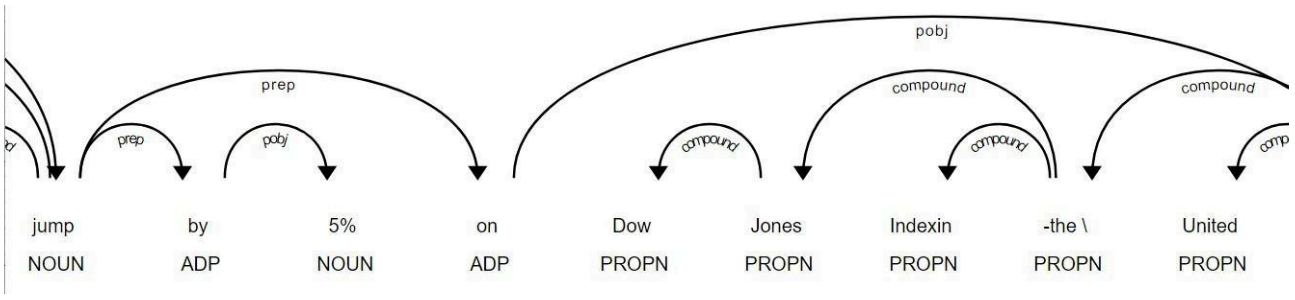
```
print(df)
displacy.serve(doc, style='dep', options={'distance': 120})
displacy.render(doc, style="ent")
```

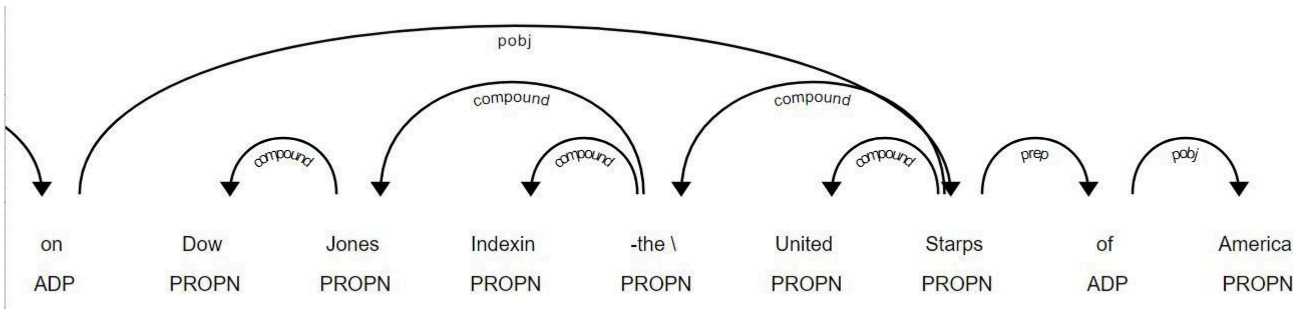
OUTPUT:

```
                           Entities   Labels  Position_Start  Position_End
0                           (Apple)      ORG               0             5
1                           (China)      GPE              23            28
2                     (May, 2020.\)     DATE              46            56
3                           (Apple)      ORG              72            77
4                          (Google)      ORG              82            88
5                            (5, %)  PERCENT             103           105
6             (Dow, Jones, Indexin)      ORG             109           126
7  (United, Starps, of, America)        ORG             134           158

Using the 'dep' visualizer
Serving on http://0.0.0.0:5000 ...
```








pobj
compound
compound
compound
compound
compound
prep
pobj
on ADP
Dow PROPN
Jones PROPN
Indexin PROPN
-the \ PROPN
United PROPN
Starps PROPN
of ADP
America PROPN

PRACTICAL NO:8

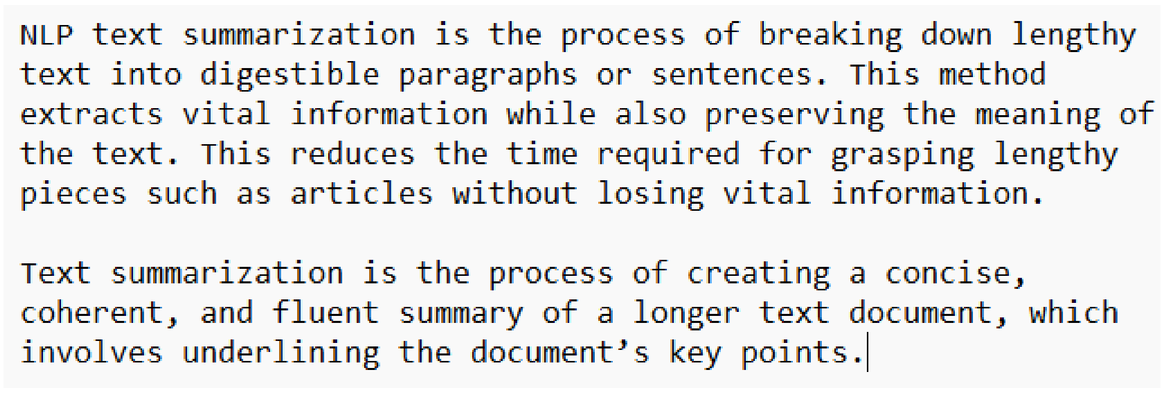
AIM: Write a program to Implement Text Summarization for the given sample text.

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:



```
NLP text summarization is the process of breaking down lengthy
text into digestible paragraphs or sentences. This method
extracts vital information while also preserving the meaning of
the text. This reduces the time required for grasping lengthy
pieces such as articles without losing vital information.

Text summarization is the process of creating a concise,
coherent, and fluent summary of a longer text document, which
involves underlining the document’s key points.
```

CODE:

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

f = open("D:/salma/salma.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:

```
Text summarization is the process of creating a
concise, coherent, and fluent summary of a longe
r text document, which involves underlining the
documentâ€™s key points.
```

Practical 9

AIM: Demonstrate the use of NLP in designing Virtual Assistants.

Apply LSTM, build conversational Bots.

THEORY:

Chat – Chat is a class that contains complete logic for processing the text data which the chatbot receives and find useful information out of it.

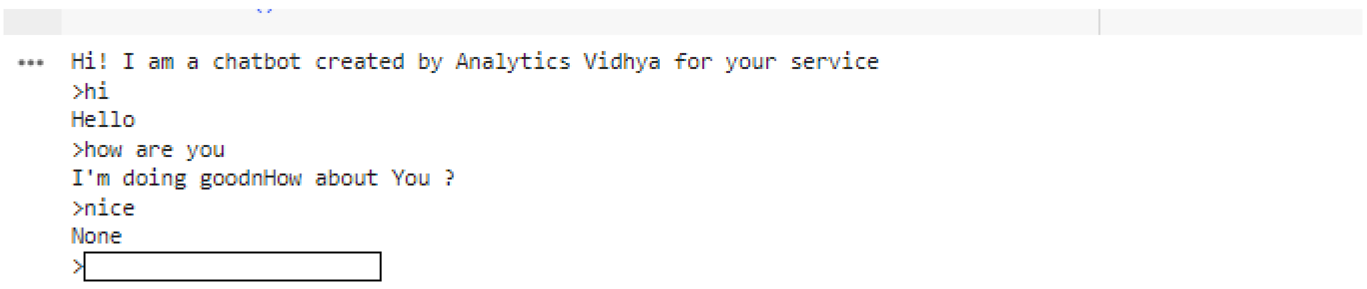
Reflections – Another import we have done is reflections which is a dictionary containing basic input and corresponding outputs. You can also create your own dictionary with more responses you want. if you print reflections it will be something like this.

CODE:

```
import nltk

from nltk.chat.util import Chat

reflections = {

"i am" : "you are",

"i was" : "you were",

"i" : "you",

"i'm" : "you are",

"i'd" : "you would",

"i've" : "you have",

"i'll" : "you will",

"my" : "your",

"you are" : "I am",

"you were" : "I was",

"you've" : "I have",

"you'll" : "I will",
"your" : "my",

"yours" : "mine",

"you" : "me",

"me" : "you"

}


pairs = [

[

r"my name is (.*)",

["Hello %1, How are you today ?",]

],

[

r"hi|hey|hello",

["Hello", "Hey there",]

],

[

r"what is your name ?",

["I am a bot created by Analytics Vidhya. you can call me crazy!",]

],

[

r"how are you ?",

["I'm doing goodnHow about You ?",]

],

[

r"sorry (.*)",

["Its alright","Its OK, never mind",]

],

[

r"I am fine",

["Great to hear that, How can I help you?",]
],

[

r"i'm (.*) doing good",

["Nice to hear that","How can I help you?:)",]

],

[

r"(.*) age?",

["I'm a computer program dudenSeriously you are asking me this?",]

],

[

r"what (.*) want ?",

["Make me an offer I can't refuse",]

],

[

r"(.*) created ?",

["Raghav created me using Python's NLTK library ","top secret ;)",]

],

[

r"(.*) (location|city) ?",

['Indore, Madhya Pradesh',]

],

[

r"how is weather in (.*)?",

["Weather in %1 is awesome like always","Too hot man here in %1","Too cold man here
in %1","Never even heard about %1"]

],



[

r"i work in (.*)?",

["%1 is an Amazing company, I have heard about it. But they are in huge loss these
```

```
days.",]

],

[

r"(.*)raining in (.*)",

["No rain since last week here in %2","Damn its raining too much here in %2"]

],

[

r"how (.*) health(.*)",

["I'm a computer program, so I'm always healthy ",]

],

[

r"(.*) (sports|game) ?",

["I'm a very big fan of Football",]

],

[

r"who (.*) sportsperson ?",

["Messy","Ronaldo","Roony"]

],

[

r"who (.*) (moviestar|actor)?",

["Brad Pitt"]

],

[

r"i am looking for online guides and courses to learn data science, can you
suggest?",

["Crazy_Tech has many great articles with each step explanation along with code, you
can explore"]

],

[

r"quit",
["BBye take care. See you soon :) ","It was nice talking to you. See you soon :)"]

],

]

def chat():

  print("Hi! I am a chatbot created by Analytics Vidhya for your service")

  chat = Chat(pairs, reflections)

  chat.converse()

#initiate the conversation

if __name__ == "__main__":

          chat()
```

OUTPUT:



```
Hi! I am a chatbot created by Analytics Vidhya for your service
>hi
Hello
>how are you
I'm doing goodnHow about You ?
>nice
None
>
```