



A PRACTICAL REPORT ON
NATURAL LANGUAGE PROCESSING

SUBMITTED BY

SIDDHI THAKKAR

(SAP ID: 53004190041)

MScIT (PART II) – SEMESTER 4

TO UNIVERSITY OF MUMBAI

FOR

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

SPECIALISATION IN ARTIFICIAL INTELLIGENCE

YEAR: 2019-2021



SVKM'S

USHA PRAVIN GANDHI COLLEGE OF ARTS, SCIENCE AND COMMERCE

**(NAAC ACCREDITED 'A' GRADE COLLEGE) JUHU SCHEME, VILE
PARLE (W)**

MUMBAI-400056



Shri Vile Parle Kelvani Mandal's

**USHA PRAVIN GANDHI COLLEGE OF ARTS,
SCIENCE AND COMMERCE**

(Affiliated with University of Mumbai)

NAAC ACCREDITED "A" GRADE

Bhaktivedanta Swami Marg, Juhu Scheme, Vile Parle(West), Mumbai-400056



Class MSCIT-AI

SAP ID: 53004190041

CERTIFICATE

This is to certify that ~~Mr.~~/Ms. **Siddhi Thakkar** has completed the practical work in the subject **Natural Language Processing** in Semester 4 as a partial fulfilment of MSC University of Mumbai. During the academic year 2020-2021

Date: 29/5/21

Signature Of Professor:

INDEX

Practical No.	Name of the Practical	Page No
1A	Install NLTK	03
1B	Convert the given text to speech	
1C	Convert audio file Speech to Text	
2A	Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories	07
2B	Create and use your own corpora (plaintext, categorical)	
2C	Study Conditional frequency distributions	
2D	Write a program to find the most frequent noun tags.	
2E	Map Words to Properties Using Python Dictionaries	
2F	Map Words to Properties Using Python Dictionaries	
2G	Find different words from a given plain text without any space by comparing this text with a given corpus of words. Also find the score of words.	
3A	Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.	15
3B	Study lemmas, hyponyms, hypernyms, entailments	
3C	Write a program using python to find synonym and antonym of word "active" using Wordnet.	
3D	Write a program to compare two nouns	
3E	Handling stopword.	
4A	Tokenization using Python's split() function	21
4B	Tokenization using Regular Expressions (RegEx)	
4C	Tokenization using NLTK	
4D	Tokenization using the spaCy library	
4E	Tokenization using Keras	
4F	Tokenization using Gensim	
5A	Word tokenization in Hindi	23
5B	Generate similar sentences from a given Hindi text input	
5C	Identify the Indian language of a text	
6A	Part of speech Tagging and chunking of user defined text	27
6B	Named Entity recognition of user defined text	
6C	Named Entity recognition with diagram using NLTK corpus – treebank	
7A	Define grammer using nltk. Analyze a sentence using the same	31
7B	Accept the input string with Regular expression of FA: 101+	
7C	Accept the input string with Regular expression of FA: (a+b)*bba	
7D	Implementation of Deductive Chart Parsing using context free grammar and a given sentence.	
8A	Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer	36
8B	Study WordNetLemmatizer	

9	Implement Naive Bayes classifier	39
10i	A. Speech tagging using spacy	41
10i	B. Speech tagging using nltk	
10ii	A. Usage of Give and Gave in the Penn Treebank sample	
10ii	B. probabilistic parser	
10C	Malt parsing: Parse a sentence and draw a tree using malt parsing.	

Practical 1

A. Install NLTK

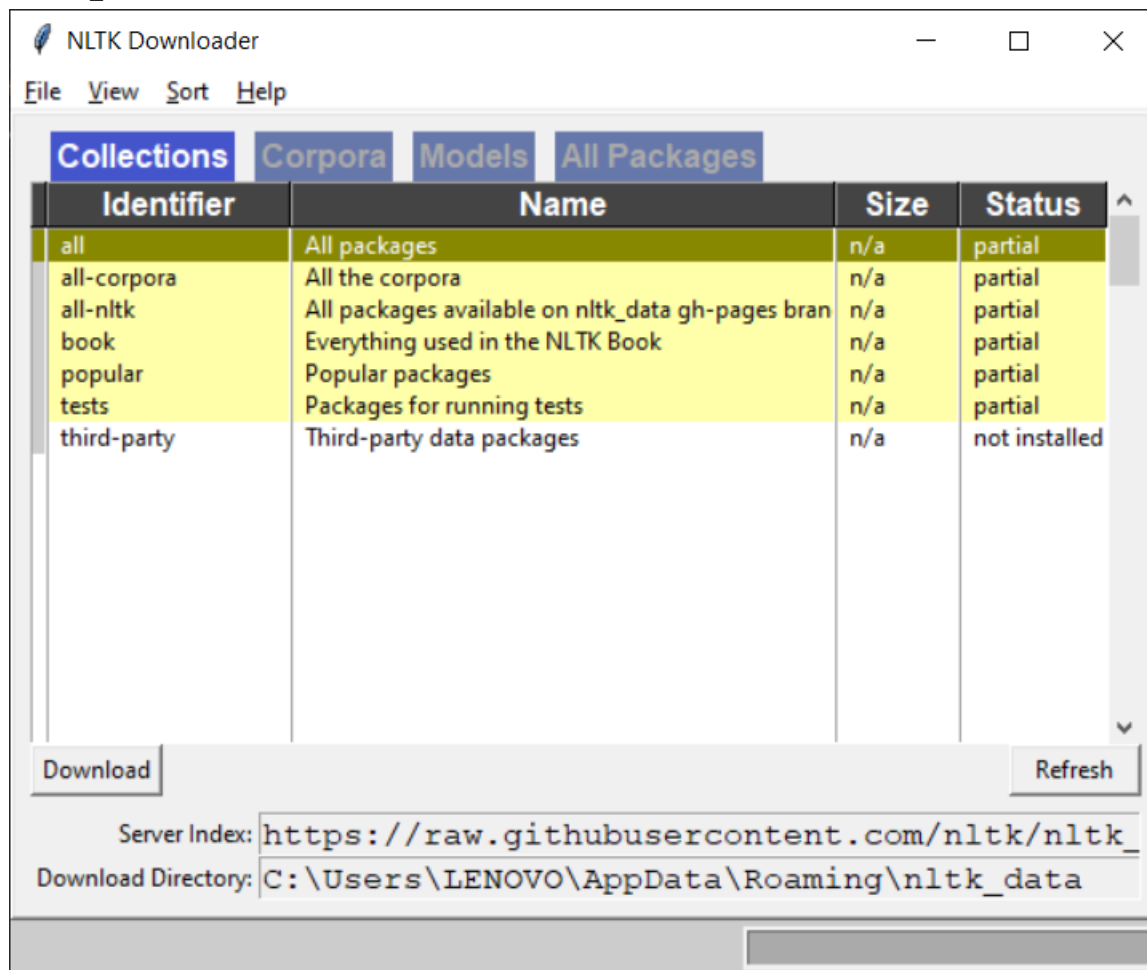
Aim: To install NLTK in Python

Concept: NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been updated for Python 3 and NLTK 3.

Code
pip install -U nltk (to install the nltk package on your machine execute this command in python console)

Output:



B. Convert the given text to speech

Aim: To convert a text(sentence or file) to an audio output and play the same.

Concept: Text-to-speech (TTS) is a type of assistive technology that reads digital text aloud. It's sometimes called "read aloud" technology. With a click of a button or the touch of a finger, TTS can take words on a computer or other digital device and convert them into audio. TTS is very helpful for kids who struggle with reading. But it can also help kids with writing and editing, and even focusing.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 20 16:48:31 2021

@author: Siddhi Thakkar
"""

from gtts import gTTS

# This module is imported so that we can
# play the converted audio
import os

# The text that you want to convert to audio
mytext = 'Practical of text to Speech performed by Siddhi Thakkar'

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.wav")

# Playing the converted file
os.system("welcome.wav")
```

Output:



C. Convert audio file Speech to Text.

Aim: To convert a given audio file to a readable text and print the text.

Concept: Speech to text conversion is the process of converting spoken words into written texts. This process is also often called speech recognition. Although these terms are almost synonymous, Speech recognition is sometimes used to describe the wider process of extracting meaning from speech, i.e. speech understanding.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 20 17:38:05 2021

@author: SiddhiThakkar
"""
import speech_recognition as sr

filename="sample.wav"

r = sr.Recognizer()
with sr.AudioFile(filename) as source:
    audio = r.record(source)

try:
    s = r.recognize_google(audio)
    print("Text: "+s)
except Exception as e:
    print("Exception: "+str(e))
```

Output:

```
In [7]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-1/1-C.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-1')
Text: summary the sides to break it therefore the you keep adequate coverage the works of places to save money baby is taking longer to getting squared away then the bank was expected during the life event company in AVN heartattack se retirement income the British were inadequate news of the saving lives are heard it has done that you naked Bond what a discussion can insert when the title of this type of song is in question or waxing or gasing needed I prevent my be personalized number work lace leather and lace work on a flat surface and smooths out this post and a separate system uses a single sirf contained Unity op shop at store holds a good mechanical isliye bad bus figures with Johar in late summer curable chairs cabinets chest down house is a set
```


Practical 2

A. Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories

Aim: To study the various corpus in python

Concept: A corpus is a large and structured set of machine-readable texts that have been produced in a natural communicative setting. Its plural is corpora. They can be derived in different ways like text that was originally electronic, transcripts of spoken language and optical character recognition, etc.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 20 22:54:57 2021

@author: SiddhiThakkar
"""

from nltk.corpus import brown
print(brown.categories())
print(brown.words(categories='news'))
print(brown.words(fileids=['cg22']))
print(brown.sents(categories=['news', 'editorial', 'reviews']))

from nltk.corpus import inaugural
print(inaugural.fileids())
print(inaugural.words())
#print(inaugural.categories())

from nltk.corpus import reuters
print(reuters.fileids())
print(reuters.words())

from nltk.corpus import udhr
print(udhr.fileids())
print(udhr.words())
```

Output:

```

In [8]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2/2-
A.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2')
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery',
'news', 'religion', 'reviews', 'romance', 'science_fiction']
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
['Does', 'our', 'society', 'have', 'a', 'runaway', '...', ...]
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', 'Atlanta's', 'recent',
'primary', 'election', 'produced', '...', 'no', 'evidence', '...', 'that', 'any', 'irregularities', 'took', 'place', '.'],
['The', 'jury', 'further', 'said', 'in', 'term-end', 'presentments', 'that', 'the', 'City', 'Executive', 'Committee', '...',
'which', 'had', 'over-all', 'charge', 'of', 'the', 'election', '...', 'deserves', 'the', 'praise', 'and', 'thanks',
'of', 'the', 'City', 'of', 'Atlanta', '...', 'for', 'the', 'manner', 'in', 'which', 'the', 'election', 'was', 'conducted',
'.'], ...]
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', '1801-Jefferson.txt', '1805-Jefferson.txt', '1809-
Madison.txt', '1813-Madison.txt', '1817-Monroe.txt', '1821-Monroe.txt', '1825-Adams.txt', '1829-Jackson.txt', '1833-
Jackson.txt', '1837-VanBuren.txt', '1841-Harrison.txt', '1845-Polk.txt', '1849-Taylor.txt', '1853-Pierce.txt', '1857-
Buchanan.txt', '1861-Lincoln.txt', '1865-Lincoln.txt', '1869-Grant.txt', '1873-Grant.txt', '1877-Hayes.txt', '1881-
Garfield.txt', '1885-Cleveland.txt', '1889-Harrison.txt', '1893-Cleveland.txt', '1897-McKinley.txt', '1901-McKinley.txt',
'1905-Roosevelt.txt', '1909-Taft.txt', '1913-Wilson.txt', '1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt',
'1929-Hoover.txt', '1933-Roosevelt.txt', '1937-Roosevelt.txt', '1941-Roosevelt.txt', '1945-Roosevelt.txt', '1949-
Truman.txt', '1953-Eisenhower.txt', '1957-Eisenhower.txt', '1961-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt',
'1973-Nixon.txt', '1977-Carter.txt', '1981-Reagan.txt', '1985-Reagan.txt', '1989-Bush.txt', '1993-Clinton.txt', '1997-
Clinton.txt', '2001-Bush.txt', '2005-Bush.txt', '2009-Obama.txt', '2013-Obama.txt', '2017-Trump.txt']
['Fellow', '-', 'Citizens', 'of', 'the', 'Senate', ...]
['test/14826', 'test/14828', 'test/14829', 'test/14832', 'test/14833', 'test/14839', 'test/14840', 'test/14841', 'test/
14842', 'test/14843', 'test/14844', 'test/14849', 'test/14852', 'test/14854', 'test/14858', 'test/14859', 'test/14860',
'test/14861', 'test/14862', 'test/14863', 'test/14865', 'test/14867', 'test/14872', 'test/14873', 'test/14875', 'test/
14876', 'test/14877', 'test/14881', 'test/14882', 'test/14885', 'test/14886', 'test/14888', 'test/14890', 'test/14891',
'test/14892', 'test/14899', 'test/14900', 'test/14903', 'test/14904', 'test/14907', 'test/14909', 'test/14911', 'test/
14912', 'test/14913', 'test/14918', 'test/14919', 'test/14921', 'test/14922', 'test/14923', 'test/14926', 'test/14928',
'test/14930', 'test/14931', 'test/14932', 'test/14933', 'test/14934', 'test/14941', 'test/14943', 'test/14949', 'test/
14951', 'test/14954', 'test/14957', 'test/14958', 'test/14959', 'test/14960', 'test/14962', 'test/14963', 'test/14964',
'test/14965', 'test/14967', 'test/14968', 'test/14969', 'test/14970', 'test/14971', 'test/14974', 'test/14975', 'test/
14978', 'test/14981', 'test/14982', 'test/14983', 'test/14984', 'test/14985', 'test/14986', 'test/14987', 'test/14988',
'test/14989', 'test/14990', 'test/14991', 'test/14992', 'test/14993', 'test/14994', 'test/14995', 'test/14996', 'test/14997', 'test/14998', 'test/14999']

```

B. Create and use your own corpora(plaintext, categorical)

Aim: To create own text corpora and read the contents of the corpora

Concept: A corpus is a large and structured set of machine-readable texts that have been produced in a natural communicative setting. Its plural is corpora. They can be derived in different ways like text that was originally electronic, transcripts of spoken language and optical character recognition, etc.

Code

```
# -*- coding: utf-8 -*-
```

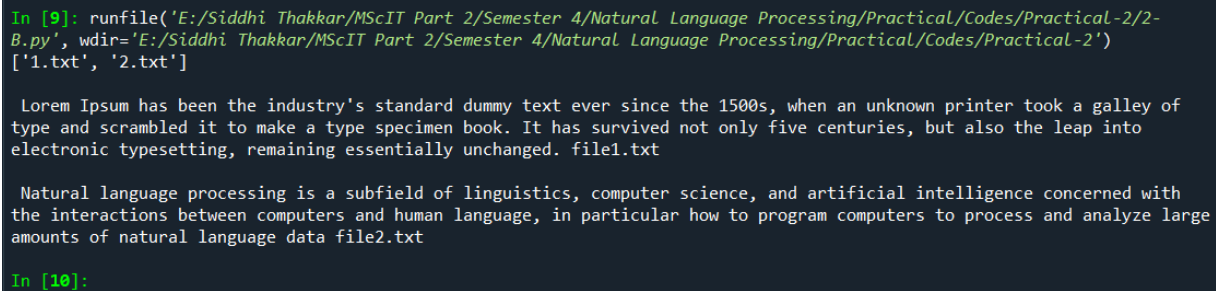
```
"""
```

```
Created on Sat May 22 19:56:05 2021
```

```
@author: Siddhi Thakkar
"""

from nltk.corpus import PlaintextCorpusReader
corpus_root = 'NewCorpus-2B/'
wordlists = PlaintextCorpusReader(corpus_root, '.*')
print(wordlists.fileids())
files=['file1.txt', 'file2.txt']
filename=0
for text in files:
    filename+=1
    with open(corpus_root+str(filename)+'.txt','r') as fout:
        print("\n",fout.read(), text)
```

Output



```
In [9]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2/2-B.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2')
['1.txt', '2.txt']

Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. file1.txt

Natural language processing is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data file2.txt

In [10]:
```

C. Study Conditional frequency distributions Study of tagged corpora with methods like tagged_sents, tagged_words.

Aim: To study tagged_sents and tagged_words in nltk.corpus

Concept: These tools are well known and well used to perform sentence and word tokenizations. Brown corpus is used here in this practical and the categories that are used belong to news and universal.

Code

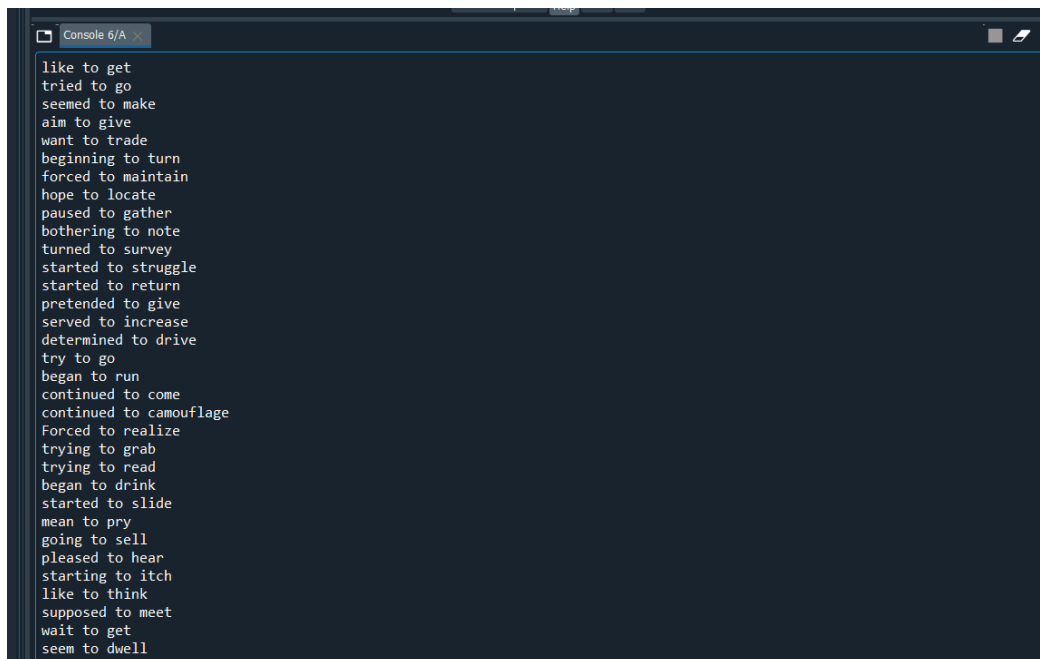
```
# -*- coding: utf-8 -*-
"""
Created on Fri May 21 17:47:47 2021

@author: SiddhiThakkar
"""
import nltk
from nltk.corpus import brown
def process(sentence):
    for (w1,t1), (w2,t2), (w3,t3) in nltk.trigrams(sentence):
        if (t1.startswith('V') and t2 == 'TO' and t3.startswith('V')):
            print(w1, w2, w3)

for tagged_sent in brown.tagged_sents():
    process(tagged_sent)

brown_news_tagged = brown.tagged_words(categories='news', tagset='universal')
data = nltk.ConditionalFreqDist((word.lower(), tag)
for (word, tag) in brown_news_tagged)
for word in sorted(data.conditions()):
    if len(data[word]) > 3:
        tags = [tag for (tag, _) in data[word].most_common()]
        print(word, ''.join(tags))
```

Output



```
like to get
tried to go
seemed to make
aim to give
want to trade
beginning to turn
forced to maintain
hope to locate
paused to gather
bothering to note
turned to survey
started to struggle
started to return
pretended to give
served to increase
determined to drive
try to go
began to run
continued to come
continued to camouflage
Forced to realize
trying to grab
trying to read
began to drink
started to slide
mean to pry
going to sell
pleased to hear
starting to itch
like to think
supposed to meet
wait to get
seem to dwell
forced to dwell
```

D. Write a program to find the most frequent noun tags.

Aim: to find the most frequently occurring noun tags and their count in the corpus

Concept: The NOUN tag is intended for common nouns only. See PROPN for proper nouns and PRON for pronouns. Note that some verb forms such as gerunds and infinitives may share properties and usage of nouns and verbs. Depending on language and context, they may be classified as either VERB or NOUN

Code

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 21 17:59:40 2021

@author: Siddhi Thakkar
"""

import nltk
from nltk.corpus import brown

tagged = brown.tagged_words(tagset='universal')

noundist = nltk.FreqDist(w2 for ((w1, t1), (w2, t2)) in
    nltk.bigrams(brown.tagged_words(tagset="universal")))
    if w1.lower() == "the" and t2 == "NOUN")
print(noundist.most_common(10))
```

Output

```
In [11]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2/2-D.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2')
[('world', 346), ('time', 250), ('way', 236), ('end', 206), ('fact', 194), ('state', 190), ('man', 176), ('door', 172), ('house', 152), ('city', 127)]

In [12]:
```

E. Map Words to Properties Using Python Dictionaries

Aim: To show the properties of word using dictionaries in python.

Concept: Semantic properties or meaning properties are those aspects of a linguistic unit, such as a morpheme, word, or sentence, that contribute to the meaning of that unit.

Code

```
import nltk
text = nltk.word_tokenize("The night is cold, with a lot of stars in the sky and huge tress alongsides of the road")
tokenized=nltk.pos_tag(text)
print(tokenized)
```

Output

```
In [1]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2/2-E.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2')
{'The': 'DT', 'night': 'NN', 'is': 'VBZ', 'cold': 'JJ', ',': ',', 'with': 'IN', 'a': 'DT', 'lot': 'NN', 'of': 'IN', 'stars': 'NNS', 'in': 'IN', 'the': 'DT', 'sky': 'NN', 'and': 'CC', 'huge': 'JJ', 'tress': 'NN', 'alongsides': 'NNS', 'road': 'NN'}
In [2]:
```

F. Study DefaultTagger, Regular expression tagger, UnigramTagger

Aim: To study the defaultTagger, regular expression tagger and unigram tagger using nltk

Concept: The nltk. tagger module defines TaggerI, a general interface for tagging texts. This interface is used by all taggers. It defines a single method, tag, which assigns a tag to each token in a list, and returns the resulting list of tagged tokens

Code

```
# -*- coding: utf-8 -*-
"""
```

```

Created on Fri May 21 18:10:09 2021
@author: Siddhi Thakkar
"""
import nltk
from nltk.corpus import brown
tokens = 'John saw 3 polar bears '.split()
default_tagger = nltk.DefaultTagger('NN')
print(default_tagger.tag(tokens))
brown_tagged_sents = brown.tagged_sents(categories='news')
print(default_tagger.evaluate(brown_tagged_sents))

patterns = [
(r'.*ing$', 'VBG'),      # gerunds
(r'.*ed$', 'VBD'),      # simple past
(r'.*es$', 'VBZ'),      # 3rd singular present
(r'.*ould$', 'MD'),     # modals
(r'.*\s$', 'NNS$'),     # possessive nouns
(r'.*s$', 'NNS'),       # plural nouns
(r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
(r'.*', 'NN')           # nouns (default)
]

regex_tagger = nltk.RegexpTagger(patterns)
exp=regex_tagger.tag("An example of Regular Expression tagger performed by Siddhi Thakkar 53004190041".split())
print(exp)
brown_tagged_sents = brown.tagged_sents(categories='news')
print(regex_tagger.evaluate(brown_tagged_sents))
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
print(unigram_tagger.tag(brown_sents[2007]))
print(unigram_tagger.evaluate(brown_tagged_sents))

```

Output

```

In [13]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2/2-F.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-2')
[('John', 'NN'), ('saw', 'NN'), ('3', 'NN'), ('polar', 'NN'), ('bears', 'NN'), ('.', 'NN')]
0.13089484257215028
[('An', 'NN'), ('example', 'NN'), ('of', 'NN'), ('Regular', 'NN'), ('Expression', 'NN'), ('tagger', 'NN'), ('performed', 'VBD'), ('by', 'NN'), ('Siddhi', 'NN'), ('Thakkar', 'NN'), ('53004190041', 'CD')]
0.20186168625812995
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ('apartments', 'NNS'), ('are', 'BER'), ('of', 'IN'), ('the', 'AT'), ('terrace', 'NN'), ('type', 'NN'), (',', ','), ('being', 'BEG'), ('on', 'IN'), ('the', 'AT'), ('ground', 'NN'), ('floor', 'NN'), ('so', 'QL'), ('that', 'CS'), ('entrance', 'NN'), ('is', 'BEZ'), ('direct', 'JJ'), ('.', '.')]
0.9349006503968017

```

G. Find different words from a given plain text without any space by comparing this text with a given corpus of words.

Also find the score of words.

Aim: To find out the occurrences of each word in the given text and checks its occurrence score

Concept: Word lists by frequency are lists of a language's words grouped by frequency of occurrence within some given text corpus, either by levels or as a ranked list, serving the purpose of vocabulary acquisition

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 23 16:31:48 2021

@author: SiddhiThakkar
"""

def word_count(str):
    counts = dict()
    str=str.lower()
    words = str.split()

    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

    return counts

print( word_count('The Quick Brown fox jumps over the lazy dog.'))
```

Output

```
In [14]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/
Practical-2/2G.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/
Practical-2')
{'the': 2, 'quick': 1, 'brown': 1, 'fox': 1, 'jumps': 1, 'over': 1, 'lazy': 1, 'dog.': 1}
```


Practical 3

A. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.

Aim: To study the word net dictionary and methods of sysnet

Concept: WordNet is a lexical database of semantic relations between words in more than 200 languages. WordNet links words into semantic relations including synonyms, hyponyms, and meronyms. The synonyms are grouped into synsets with short definitions and usage examples

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 12:22:47 2021

@author: SiddhiThakkar
"""

from nltk.corpus import wordnet
syn = wordnet.synsets('hello')[0]

print ("Synset name : ", syn.name())

# Defining the word
print ("\nSynset Definition : ", syn.definition())

# list of phrases that use the word in context
print ("\nSynset example : ", syn.examples())

synonyms = []
antonyms = []

for syn in wordnet.synsets("good"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print("\nSysnet Synonyms : ",set(synonyms))
print("\nSysnet Antonyms : ",set(antonyms))
```

Ouput

```
In [15]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3/3-A.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3')
Synset name : hello.n.01

Synset Definition : an expression of greeting

Synset example : ['every morning they exchanged polite hellos']

Synset Synonyms : {'adept', 'soundly', 'unspoilt', 'practiced', 'respectable', 'upright', 'undecomposed', 'skillful', 'trade_good', 'effective', 'goodness', 'full', 'well', 'good', 'unspoiled', 'serious', 'secure', 'beneficial', 'salutary', 'dear', 'ripe', 'just', 'proficient', 'estimable', 'sound', 'thoroughly', 'in_force', 'near', 'skilful', 'safe', 'honorable', 'expert', 'right', 'commodity', 'in_effect', 'honest', 'dependable'}

Synset Antonyms : {'bad', 'ill', 'badness', 'evil', 'evilness'}

In [16]:
```

B. Study lemmas, hyponyms, hypernyms, entailments

Aim: To study lemmas, hyponyms, hypernyms and entailments from wordnet

Concept: In morphology and lexicography, a lemma is the canonical form, dictionary form, or citation form of a set of words. In English, for example, break, breaks, broke, broken and breaking are forms of the same lexeme, with break as the lemma by which they are indexed. In linguistics, hyponymy is a semantic relation between a hyponym denoting a subtype and a hypernym or hyperonym denoting a supertype. In other words, the semantic field of the hyponym is included within that of the hypernym. In simpler terms, a hyponym is in a type-of relationship with its hypernym. An entailment is a deduction or implication, that is, something that follows logically from or is implied by something else.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 12:42:30 2021

@author: SiddhiThakkar
"""
from nltk.corpus import wordnet as wn
for syn in wn.synsets("good"):
    print(syn.lemmas())
```

```

machine_that_prints = wn.synset('printer.n.03')
print("Hyponyms: ",sorted([lemma.name() for synset in machine_that_prints.hyponyms() for lemma in
synset.lemmas()]))
print("Hypernyms: ",[lemma.name() for synset in machine_that_prints.hypernyms() for lemma in
synset.lemmas()])
print("Entailments: ",wn.synset('eat.v.01').entailments())

```

Ouput

```

04.in_force'])
[Lemma('good.s.15.good')]
[Lemma('good.s.16.good'), Lemma('good.s.16.serious')]
[Lemma('good.s.17.good'), Lemma('good.s.17.sound')]
[Lemma('good.s.18.good'), Lemma('good.s.18.salutary')]
[Lemma('good.s.19.good'), Lemma('good.s.19.honest')]
[Lemma('good.s.20.good'), Lemma('good.s.20.undecomposed'), Lemma('good.s.20.unspoiled'), Lemma('good.s.20.unspoilt')]
[Lemma('good.s.21.good')]
[Lemma('well.r.01.well'), Lemma('well.r.01.good')]
[Lemma('thoroughly.r.02.thoroughly'), Lemma('thoroughly.r.02.soundly'), Lemma('thoroughly.r.02.good')]
Hyponyms: ['Addressograph', 'addressing_machine', 'character-at-a-time_printer', 'character_printer',
'electrostatic_printer', 'impact_printer', 'line-at-a-time_printer', 'line_printer', 'page-at-a-time_printer',
'page_printer', 'printer', 'serial_printer', 'thermal_printer', 'typesetting_machine']
Hypernyms: ['machine']
Entailments: [Synset('chew.v.01'), Synset('swallow.v.01')]

```

C. Write a program using python to find synonym and antonym of word "active" using Wordnet

Aim: To find the antonym and synonym of the word “active” using wordnet

Concept: WordNet’s structure makes it a useful tool for computational linguistics and natural language processing. WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings.

Code

```

# -*- coding: utf-8 -*-
"""
Created on Sat May 22 12:49:24 2021

@author: SiddhiThakkar
"""

from nltk.corpus import wordnet
synonyms = []
antonyms = []

```

```

for syn in wordnet.synsets("active"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print("\nSysnet Synonyms : ",set(synonyms))
print("\nSysnet Antonyms : ",set(antonyms))

```

Ouput

```

In [17]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3/3-C.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3')

Sysnet Synonyms : {'active_voice', 'active', 'active_agent', 'fighting', 'participating', 'dynamic', 'alive', 'combat-ready'}

Sysnet Antonyms : {'extinct', 'quiet', 'passive_voice', 'dormant', 'passive', 'stative', 'inactive'}

```

D. Compare two nouns

Aim: To compare two nouns and find out its similarity

Concept: Comparative adjectives are used to compare one noun to another noun. In these instances, only two items are being compared. For example, someone might say that "the blue bird is angrier than the robin." Superlative adjectives are used to compare three or more nouns.

Code

```

# -*- coding: utf-8 -*-
"""
Created on Sat May 22 13:02:32 2021

@author: Siddhi Thakkar
"""

from nltk.corpus import wordnet
print("\nComparing ship and boat:")
n1 = wordnet.synset('ship.n.01')
n2 = wordnet.synset('boat.n.01')
print(n1.wup_similarity(n2))
print("\nComparing bus and boat:")

```

```
n1 = wordnet.synset('bus.n.01')
n2 = wordnet.synset('boat.n.01')
print(n1.wup_similarity(n2))
print("\nComparing red and green:")
n1 = wordnet.synset('red.n.01')
n2 = wordnet.synset('green.n.01')
print(n1.wup_similarity(n2))
```

Ouput

```
In [18]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3/3-
0.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3')

Comparing ship and boat:
0.9090909090909091

Comparing bus and boat:
0.7

Comparing red and green:
0.875
```

E. Handling stopword. Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List. Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List. Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

Aim: To remove stop words from a given sentence using nltk, genism and spacy library

Concept: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For example, the words like the, he, have etc. Such words are already captured this in corpus named corpus.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 14:21:43 2021

@author: SiddhiThakkar
"""

from nltk.corpus import stopwords
import nltk
from nltk.tokenize import word_tokenize
text = "Siddhi Thakkar likes reading novels. one of her most favourite genre is mystery"
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]
print(tokens_without_sw)
filtered_sentence = (" ").join(tokens_without_sw)
print("NLTK: ",filtered_sentence)
from gensim.parsing.preprocessing import remove_stopwords
filtered_sentence = remove_stopwords(text)
print("Genism:",filtered_sentence)
import spacy
sp = spacy.load('C:/Users/LENOVO/AppData/Local/Programs/Python/Python38/Lib/site-packages/en_core_web_sm/en_core_web_sm-3.0.0')
all_stopwords = sp.Defaults.stop_words
text_tokens = word_tokenize(text)
tokens_without_sw= [word for word in text_tokens if not word in all_stopwords]
filtered_sentence = (" ").join(tokens_without_sw)
print("Spacy: ",filtered_sentence)
```

Ouput

```
In [20]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3/3
E.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-3')
['Siddhi', 'Thakkar', 'likes', 'reading', 'novels', '.', 'favourite', 'genre', 'mystery']
NLTK: Siddhi Thakkar likes reading novels . favourite genre mystery
Genism: Siddhi Thakkar likes reading novels. favourite genre mystery
Spacy: Siddhi Thakkar likes reading novels . favourite genre mystery
```

Practical 4: Text Tokenization

a. Tokenization using Python's split() function

b. Tokenization using Regular Expressions (RegEx)

c. Tokenization using NLTK

d. Tokenization using the spaCy library

e. Tokenization using Keras

f. Tokenization using Gensim

Aim: Performing tokenization on a sentence using split(), regex, nltk, spacy, keras and genism library

Concept: Tokens are the building blocks of Natural Language. Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 15:48:17 2021

@author: SiddhiThakkar
"""
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import word_tokenize
import spacy
from keras.preprocessing.text import text_to_word_sequence
from gensim.utils import tokenize
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization
and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately
developed
liquid-fuel launch vehicle to orbit the Earth."""
# Splits at space
```

```

split_token=text.split()
print("Tokenization using split function: ",split_token)
# Create a reference variable for Class RegexpTokenizer
tk = RegexpTokenizer("\s+", gaps = True)
# Use tokenize method
regex_token = tk.tokenize(text)
print("\n\nRegular expression tokenizer: ",regex_token)
print("\n\nNLTK tokenization: ",word_tokenize(text))
sp = spacy.load('C:/Users/LENOVO/AppData/Local/Programs/Python/Python38/Lib/site-
packages/en_core_web_sm/en_core_web_sm-3.0.0')
spacy_token=sp(text)
spacyToken=[]
for word in spacy_token:
    spacyToken.append(word.text)
print("\n\nSpacy tokenization: ",spacyToken)
keras_token = text_to_word_sequence(text)
print("\n\nKeras Tokenization:",keras_token)
print("\n\nGenism Tokenization:",list(tokenize(text)))

```

Ouput

```

In [21]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-4/4-
A,B,C,D,E,F.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-4')
Tokenization using split function: ['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to',
'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining',
'city', 'on', 'Mars.', 'In', '2008,', 'SpaceX's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-
fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth.']

Regular expression tokenizer: ['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become',
'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on',
'Mars.', 'In', '2008,', 'SpaceX's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-fuel', 'launch',
'vehicle', 'to', 'orbit', 'the', 'Earth.']

NLTK tokenization: ['Founded', 'in', '2002', ',', 'SpaceX', '', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become',
'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on',
'Mars', '.', 'In', '2008', ',', 'SpaceX', '', 's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-
fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']

Spacy tokenization: ['Founded', 'in', '2002', ',', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become',
'a', 'spacefaring', 'civilization', 'and', 'a', 'multi', '-', 'planet', '\n', 'species', 'by', 'building', 'a', 'self', '-',
'sustaining', 'city', 'on', 'Mars', '.', 'In', '2008', ',', 'SpaceX', 's', 'Falcon', '1', 'became', 'the', 'first', 'privately',
'developed', '\n', 'liquid', '-', 'fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']

Keras Tokenization: ['founded', 'in', '2002', 'spacex's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a',
'spacefaring', 'civilization', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city',
'on', 'mars', 'in', '2008', 'spacex's', 'falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid', 'fuel',
'launch', 'vehicle', 'to', 'orbit', 'the', 'earth']

Genism Tokenization: ['Founded', 'in', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a',
'spacefaring', 'civilization', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city',
'on', 'Mars', 'In', 'SpaceX', 's', 'Falcon', 'became', 'the', 'first', 'privately', 'developed', 'liquid', 'fuel', 'launch',
'vehicle', 'to', 'orbit', 'the', 'Earth']

```


Practical 5: Import NLP Libraries for Indian Languages and perform:

A. Word tokenization in Hindi

Aim: To perform word tokenization on a text in Hindi language.

Concept: Word tokenization is the process of splitting a large sample of text into words. This is a requirement in natural language processing tasks where each word needs to be captured and subjected to further analysis like classifying and counting them for a particular sentiment etc.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 13:11:28 2021

@author: SiddhiThakkar
"""

import nltk
from nltk.tokenize import tokenize
"""
from nltk.tokenize import setup
setup('hi')
"""

hindi_text = """प्राचीन काल में विक्रमादित्य नाम के एक आदर्श राजा हुआ करते थे।
अपने साहस, पराक्रम और शौर्य के लिए राजा विक्रम मशहूर थे।
ऐसा भी कहा जाता है कि राजा विक्रम अपनी प्राजा के जीवन के दुख दर्द जानने के लिए रात्री के पहर में भेष बदल
कर नगर में घूमते थे।"""

# tokenize(input text, language code)
print(tokenize(hindi_text, "hi"))
```

Ouput

```
In [22]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-5/5-A.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-5')
['प्राचीन', 'काल', 'में', 'विक्रमादित्य', 'नाम', 'के', 'एक', 'आदर्श', 'राजा', 'हुआ', 'करते', 'थे', '।', 'अपने', 'साहस', '।', 'पराक्रम', 'और', 'शौर्य', 'के', 'लिए', 'राजा',
'विक्रम', 'मशहूर', 'थे', '।', 'ऐसा', 'भी', 'कहा', 'जाता', 'है', 'कि', 'राजा', 'विक्रम', 'अपनी', 'प्रा', 'जा', 'के', 'जीवन', 'के', 'दुख', 'दर्द', 'जानने', 'के', 'लिए', 'रात्री',
'के', 'पहर', 'में', 'भेष', 'बदल', 'कर', 'नगर', 'में', 'घूमते', 'थे', '।']

In [23]:
```

B. Generate similar sentences from a given Hindi text input

Aim: To get similar words of a text in Hindi language.

Concept: . The definition of similar is two things that have characteristics that resemble each other but are not exactly alike.

Code

```
from nltk.nltk import get_similar_sentences
import torch
torch.manual_seed(1)
# get similar sentences to the one given in hindi
output = get_similar_sentences('मैं आज बहुत खुश हूँ', 5, 'hi')

print(output)
```

Output

```
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class '
warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class '
warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class '
warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class '
warnings.warn(msg, SourceChangeWarning)
['मैं आज काफी खुश हूँ', 'मैं आज काफी खुश हूँ', 'मैं आज अत्यधिक खुश हूँ', 'मैं आज बहुत खुश हूँ', 'मैं आज बहुत खुश हूँ']
```

C. Identify the Indian language of a text

Aim: To identify the language of a text.

Concept: In natural language processing, language identification or language guessing is the problem of determining which natural language given content is in. Computational approaches to this problem view it as a special case of text categorization, solved with various statistical methods.

Code

```
from nltk.nltk import identify_language
#Identify the Language of given text
identify_language('मी विद्यार्थी आहे')
```

Output

```
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class 'torch.nn.modules.dropout.Dropout' has
warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class 'torch.nn.modules.linear.Linear' has cl
warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:493: SourceChangeWarning: source code of class 'torch.nn.modules.activation.ReLU' has
warnings.warn(msg, SourceChangeWarning)
'marathi'
```

Practical 6: Illustrate part of speech tagging.

A. Part of speech Tagging and chunking of user defined text.

Aim: To perform part of speech tagging and chunk the data

Concept: POS tagging is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 17:34:33 2021

@author: SiddhiThakkar
"""
import nltk

text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization
and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately
developed
liquid-fuel launch vehicle to orbit the Earth."""

from nltk import pos_tag
from nltk import RegexpParser

text = text.split()
print("After Split:", text)

tokens_tag = pos_tag(text)
print("\nAfter Token:", tokens_tag)

patterns = """mychunk: {<NN.?>*<VBD>*<JJ.?>*<CC>?}"""
chunker = RegexpParser(patterns)

print("\nAfter Regex:", chunker)

output = chunker.parse(tokens_tag)

print("\nAfter Chunking", output)
```

Ouput:

```

In [23]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-6/6-A.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-6')
After Split: ['Founded', 'in', '2002', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring',
'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on', 'Mars.', 'In',
'2008', 'SpaceX's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-fuel', 'launch', 'vehicle', 'to',
'orbit', 'the', 'Earth.']

After Token: [('Founded', 'VBN'), ('in', 'IN'), ('2002', 'CD'), ('SpaceX's', 'NNP'), ('mission', 'NN'), ('is', 'VBZ'), ('to',
'TO'), ('enable', 'VB'), ('humans', 'NNS'), ('to', 'TO'), ('become', 'VB'), ('a', 'DT'), ('spacefaring', 'JJ'), ('civilization',
'NN'), ('and', 'CC'), ('a', 'DT'), ('multi-planet', 'JJ'), ('species', 'NNS'), ('by', 'IN'), ('building', 'VBG'), ('a', 'DT'),
('self-sustaining', 'JJ'), ('city', 'NN'), ('on', 'IN'), ('Mars.', 'NNP'), ('In', 'IN'), ('2008', 'CD'), ('SpaceX's', 'NNP'),
('Falcon', 'NNP'), ('1', 'CD'), ('became', 'VBD'), ('the', 'DT'), ('first', 'JJ'), ('privately', 'RB'), ('developed', 'VBN'),
('liquid-fuel', 'JJ'), ('launch', 'NN'), ('vehicle', 'NN'), ('to', 'TO'), ('orbit', 'VB'), ('the', 'DT'), ('Earth.', 'NNP')]

After Regex: chunk.RegexpParser with 1 stages:
RegexpChunkParser with 1 rules:
<ChunkRule: '<NN.?>*<VBD>*<JJ.?>*<CC>?>'>

After Chunking (S
Founded/VBN
in/IN
2002,/CD
(mychunk SpaceX's/NNP mission/NN)
is/VBZ
to/TO
enable/VB
(mychunk humans/NNS)
to/TO
become/VB
a/DT
(mychunk spacefaring/JJ)
(mychunk civilization/NN and/CC)
a/DT
(mychunk multi-planet/JJ)
(mychunk species/NNS)
by/IN
2008,/CD
1,/CD
became/VBD
the/DT
first/JJ
privately/RB
developed/VBN
liquid-fuel/JJ
launch/NN
vehicle/NN
to/TO
orbit/VB
the/DT
Earth./NNP)

```

B. Named Entity recognition of user defined text.

Aim: To create NER on user defined text

Concept: Named entity recognition (NER) is probably the first step towards information extraction that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

Code

```

# -*- coding: utf-8 -*-
"""
Created on Sat May 22 17:40:40 2021

@author: SiddhiThakkar
"""

import nltk

```

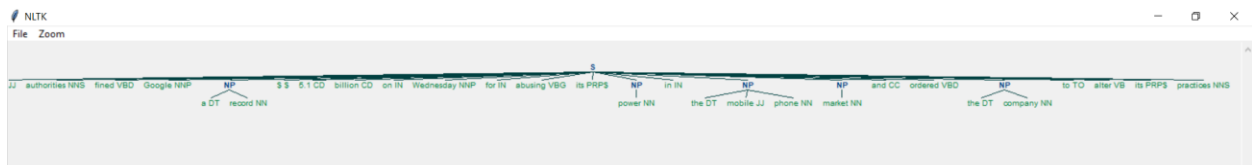
```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
```

ex = 'European authorities fined Google a record \$5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter its practices'

```
def preprocess(sent):
    sent = nltk.word_tokenize(sent)
    sent = nltk.pos_tag(sent)
    return sent
sent = preprocess(ex)
print(sent)
pattern = 'NP: {<DT>?<JJ>*<NN>}'
cp = nltk.RegexpParser(pattern)
cs = cp.parse(sent)
print(cs)
cs.draw()
```

Ouput

```
In [29]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-6/6-B.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-6')
[('European', 'JJ'), ('authorities', 'NNS'), ('fined', 'VBD'), ('Google', 'NNP'), ('a', 'DT'), ('record', 'NN'), ('$ ', '$ '),
('5.1', 'CD'), ('billion', 'CD'), ('on', 'IN'), ('Wednesday', 'NNP'), ('for', 'IN'), ('abusing', 'VBG'), ('its', 'PRP$'),
('power', 'NN'), ('in', 'IN'), ('the', 'DT'), ('mobile', 'JJ'), ('phone', 'NN'), ('market', 'NN'), ('and', 'CC'), ('ordered',
'VBD'), ('the', 'DT'), ('company', 'NN'), ('to', 'TO'), ('alter', 'VB'), ('its', 'PRP$'), ('practices', 'NNS')]
(S
  European/JJ
  authorities/NNS
  fined/VBD
  Google/NNP
  (NP a/DT record/NN)
  $/$
  5.1/CD
  billion/CD
  on/IN
  Wednesday/NNP
  for/IN
  abusing/VBG
  its/PRP$
  (NP power/NN)
  in/IN
  (NP the/DT mobile/JJ phone/NN)
  (NP market/NN)
  and/CC
  ordered/VBD
  (NP the/DT company/NN)
  to/TO
  alter/VB
  its/PRP$
  practices/NNS)
```



C. Named Entity recognition with diagram using NLTK corpus – treebank

Aim: To create NER using Treebank corpus

Concept: In linguistics, a treebank is a parsed text corpus that annotates syntactic or semantic sentence structure. The construction of parsed corpora in the early 1990s revolutionized computational linguistics, which benefitted from large-scale empirical data.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 23 21:23:34 2021

@author: Siddhi Thakkar
"""

import nltk

cp = nltk.RegexpParser('<CHUNK: {<V.*> <TO> <V.*>}>')
treebank = nltk.corpus.treebank
for sent in treebank.tagged_sents():
    tree = cp.parse(sent)
    for subtree in tree.subtrees():
        if subtree.label() == 'CHUNK': print(subtree)
```

Ouput

```
In [30]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/
Practical/Codes/Practical-6/6-C.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural
Language Processing/Practical/Codes/Practical-6')
(CHUNK came/VBD to/TO work/VB)
(CHUNK returned/VBN to/TO favor/VB)
(CHUNK admits/VBZ to/TO mixed/VBN)
(CHUNK led/VBN to/TO even/VB)
```

Practical 7

A. Define grammer using nltk. Analyze a sentence using the same.

Aim: Creating a context free grammar, and parsing it

Concept: Context free grammar is a formal grammar which is used to generate all possible strings in a given formal language. Context free grammar G can be defined by four tuples as: $G = (V, T, P, S)$

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 17:56:12 2021
@author: Siddhi Thakkar
"""

import nltk
groucho_grammar = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'T'
VP -> V NP | VP PP
Det -> 'a' | 'my'
N -> 'cake' | 'birthday'
V -> 'baked'
P -> 'on'
""")
sent = ['T', 'baked', 'a', 'cake', 'on', 'my', 'birthday']
parser = nltk.ChartParser(groucho_grammar)
for tree in parser.parse(sent):
    print(tree)
```

Ouput

```
In [31]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7/7-A.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7')
(S
 (NP I)
 (VP
  (VP (V baked) (NP (Det a) (N cake)))
  (PP (P on) (NP (Det my) (N birthday)))))
(S
 (NP I)
 (VP
  (V baked)
  (NP (Det a) (N cake) (PP (P on) (NP (Det my) (N birthday))))))
```


B. Accept the input string with Regular expression of FA:

101+

Aim: To match string only with the given regular expression

Concept: A regular expression is a sequence of characters that specifies a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 17:50:48 2021

@author: Siddhi Thakkar
"""

def FA(s):
    #if the length is less than 3 then it can't be accepted, Therefore end the process.
    if len(s)<3:
        return "Rejected"
    #first three characters are fixed. Therefore checking them using index
    if s[0]=='1':
        if s[1]=='0':
            if s[2]=='1':
                # After index 2 only "1" can appear. Therefore break the process if any other character is
                detected
                for i in range(3,len(s)):
                    if s[i]!='1':
                        return "Rejected"
                return "Accepted"
            return "Rejected"
        return "Rejected"
    return "Rejected"
inputs=['1','10101','101','10111','01010','', '10110000000"]
for i in inputs:
    print(FA(i))
```

Ouput

```
In [33]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7/7-B.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7')
Rejected
Rejected
Accepted
Accepted
Rejected
Rejected
Rejected
```

C. Accept the input string with Regular expression of FA: (a+b)*bba

Aim: To match string only with the given regular expression

Concept: A regular expression is a sequence of characters that specifies a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 17:53:38 2021

@author: Siddhi Thakkar
"""

def FA(s):
    size=0
    #scan complete string and make sure that it contains only 'a' & 'b'
    for i in s:
        if i=='a' or i=='b':
            size+=1
        else:
            return "Rejected"
    #After checking that it contains only 'a' & 'b'
    #check it's length it should be 3 atleast
    if size>=3:
        #check the last 3 elements
        if s[size-3]=='b':
            if s[size-2]=='b':
                if s[size-1]=='a':
                    return "Accepted"
                return "Rejected"
            return "Rejected"
        return "Rejected"
    return "Rejected"

inputs=['bba', 'ababbba', 'abba','abb', 'baba','bbb','']
for i in inputs:
    print(FA(i))
```

Ouput

```
In [34]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7/7-C.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7')
Accepted
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
```

D. Implementation of Deductive Chart Parsing using context free grammar and a given sentence

Aim: To implement deductive chart parsing using context free grammar

Concept: Deductive reasoning, also called deductive logic, is the process of reasoning from one or more general statements regarding what is known to reach a logically certain conclusion. Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar. The term parsing comes from Latin pars, meaning part

Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 24 19:48:50 2021

@author: Siddhi Thakkar
"""

import nltk
nltk.parse.chart.demo(2, print_times=False, trace=1, sent='I saw a dog', numparses=1)
```

Ouput

```

In [12]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7/7-D.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-7')
* Sentence:
I saw a dog
['I', 'saw', 'a', 'dog']

* Strategy: Bottom-up

|, I, , saw, , a, , dog, |
|-----|, , , | [0:1] 'I'
|, [-----], , , | [1:2] 'saw'
|, , [-----], , | [2:3] 'a'
|, , , [-----], | [3:4] 'dog'
|> , , , | [0:0] NP -> * 'I'
| [-----], , , | [0:1] NP -> * 'I' *
|> , , , | [0:0] S -> * NP VP
| [-----], , , | [0:0] NP -> * NP PP
| [-----], , , | [0:1] S -> NP * VP
| [-----], , , | [0:1] NP -> NP * PP
|, > , , | [1:1] Verb -> * 'saw'
|, [-----], , , | [1:2] Verb -> 'saw' *
|, > , , | [1:1] VP -> * Verb NP
|, > , , | [1:1] VP -> * Verb
|, [-----], , , | [1:2] VP -> Verb * NP
|, [-----], , , | [1:2] VP -> Verb *
|, > , , | [1:1] VP -> * VP PP
| [-----], , , | [0:2] S -> NP VP *
|, [-----], , , | [1:2] NP -> VP * PP
|, > , , | [2:2] Det -> * 'a'
|, , [-----], , | [2:3] Det -> 'a' *
|, > , , | [2:2] NP -> * Det Noun
|, [-----], , , | [2:3] NP -> Det * Noun
|, , > , , | [3:3] Noun -> * 'dog'
|, , [-----], , | [3:4] Noun -> 'dog' *
|, , [-----], , | [2:4] NP -> Det Noun *
|, > , , | [2:2] S -> * NP VP

```

Practical 8

A. Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer

Aim: To study different types of stemmers

Concept: Stemming is basically removing the suffix from a word and reduce it to its root word. We use these suffix to create a new word from original stem word. Here is the link to official docs of NLTK on Stemming. The stem of the verb wait is wait: it is the part that is common to all its inflected variants

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 19:21:35 2021

@author: Siddhi Thakkar
"""

from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
from nltk.stem import RegexpStemmer
from nltk.stem.snowball import SnowballStemmer

from nltk.tokenize import word_tokenize

ps = PorterStemmer()
ls = LancasterStemmer()
rs = RegexpStemmer('ing$s$s$e$|able$', min=4)
ss = SnowballStemmer(language='english')
# choose some words to be stemmed
sentence = "Reading is enjoying a world of fantasy"
words = word_tokenize(sentence)

for w in words:
    print("\n",w, "<== Porter Stemmer ==> ", ps.stem(w))
    print(w, "<== Lancaster Stemmer ==> ", ls.stem(w))
    print(w, "<== RegEx Stemmer ==> ", rs.stem(w))
    print(w, "<== Snowball Stemmer ==> ", ss.stem(w))
```

Ouput

```
Console 6/A x
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-8')

Reading <== Porter Stemmer ==>: read
Reading <== Lancaster Stemmer ==>: read
Reading <== RegEx Stemmer ==>: Read
Reading <== Snowball Stemmer ==>: read

is <== Porter Stemmer ==>: is
is <== Lancaster Stemmer ==>: is
is <== RegEx Stemmer ==>: is
is <== Snowball Stemmer ==>: is

enjoying <== Porter Stemmer ==>: enjoy
enjoying <== Lancaster Stemmer ==>: enjoy
enjoying <== RegEx Stemmer ==>: enjoy
enjoying <== Snowball Stemmer ==>: enjoy

a <== Porter Stemmer ==>: a
a <== Lancaster Stemmer ==>: a
a <== RegEx Stemmer ==>: a
a <== Snowball Stemmer ==>: a

world <== Porter Stemmer ==>: world
world <== Lancaster Stemmer ==>: world
world <== RegEx Stemmer ==>: world
world <== Snowball Stemmer ==>: world

of <== Porter Stemmer ==>: of
of <== Lancaster Stemmer ==>: of
of <== RegEx Stemmer ==>: of
of <== Snowball Stemmer ==>: of

fantasy <== Porter Stemmer ==>: fantasi
fantasy <== Lancaster Stemmer ==>: fantasy
fantasy <== RegEx Stemmer ==>: fantasy
fantasy <== Snowball Stemmer ==>: fantasi
```

B. Study WordNetLemmatizer

Aim: To study wordnet lemmatizer

Concept: Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 19:32:37 2021

@author: Siddhi Thakkar
"""

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
# Init the Wordnet Lemmatizer
lemmatizer = WordNetLemmatizer()
# Lemmatize Single Word
sentence = "He makes a very delicious meal. It costs nothing to be good"
words = word_tokenize(sentence)
for w in words:
    print(w, " ==> ", lemmatizer.lemmatize(w))
```

Ouput

```
In [37]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-8/8-B.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-8')
He ==>: He
makes ==>: make
a ==>: a
very ==>: very
delicious ==>: delicious
meal ==>: meal
. ==>: .
It ==>: It
costs ==>: cost
nothing ==>: nothing
to ==>: to
be ==>: be
good ==>: good
```

Practical 9: Implement Naive Bayes classifier

Aim: To implement naïve bayes classifier on gender prediction

Concept: Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 22 19:39:46 2021

@author: Siddhi Thakkar
"""

import pandas as pd
import numpy as np
from sklearn.naive_bayes import MultinomialNB
import requests
from bs4 import BeautifulSoup
import matplotlib as mlp
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
mlp.rcParams.update({'font.family': 'Open Sans', 'font.size': 16})

names = pd.read_csv("NationalNames.csv", dtype = {'Count': np.int32})
names = names.fillna(0)
print(names.head())
namechart = names.groupby(['Name', 'Gender'], as_index = False)['Count'].sum()
print(namechart.head(5))
namechartdiff = namechart.reset_index().pivot('Name', 'Gender', 'Count')
namechartdiff = namechartdiff.fillna(0)
namechartdiff["Mpercent"] = ((namechartdiff["M"] - namechartdiff["F"])/(namechartdiff["M"] +
namechartdiff["F"]))
namechartdiff['gender'] = np.where(namechartdiff['Mpercent'] > 0.001, 'male', 'female')
print(namechartdiff.head())
char_vectorizer = CountVectorizer(analyzer='char', ngram_range=(2, 2))
X = char_vectorizer.fit_transform(namechartdiff.index)
```



```

X = X.tocsc()
y = (namechartdiff.gender == 'male').values.astype(np.int)

itrain, itest = train_test_split(range(namechartdiff.shape[0]), train_size=0.7)
mask=np.ones(namechartdiff.shape[0], dtype='int')
mask[itrain]=1
mask[itest]=0
mask = (mask==1)
Xtrainthis=X[mask]
Ytrainthis=y[mask]
Xtestthis=X[~mask]
Ytestthis=y[~mask]
clf = MultinomialNB(alpha = 1)
clf.fit(Xtrainthis, Ytrainthis)
training_accuracy = clf.score(Xtrainthis,Ytrainthis)
test_accuracy = clf.score(Xtestthis,Ytestthis)

print("Training Accuracy",training_accuracy)
print("Testing Accuracy",test_accuracy)
def lookup(x):
    str(x)
    new = char_vectorizer.transform([x])
    y_pred = clf.predict(new)
    if (y_pred == 1):
        print("This is most likely a male name!")
    else:
        print("This is most likely a female name!")

lookup("Beena")

```

Ouput

```

In [38]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-9/9.py',
wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/Practical/Codes/Practical-9')

```

	Id	Name	Year	Gender	Count
0	1	Mary	1880	F	7065
1	2	Anna	1880	F	2604
2	3	Emma	1880	F	2003
3	4	Elizabeth	1880	F	1939
4	5	Minnie	1880	F	1746

	Name	Gender	Count
0	Aaban	M	72
1	Aabha	F	21
2	Aabid	M	5
3	Aabriella	F	10
4	Aadam	M	196

Gender	F	M	Mpercent	gender
Name				
Aaban	0.0	72.0	1.0	male
Aabha	21.0	0.0	-1.0	female
Aabid	0.0	5.0	1.0	male
Aabriella	10.0	0.0	-1.0	female
Aadam	0.0	196.0	1.0	male

```

Training Accuracy 0.7407260886765467
Testing Accuracy 0.7402989313735933
This is most likely a female name!

```

Practical 10

A-i. Speech tagging using spacy

Aim: To perform pos tagging in spacy

Concept: In corpus linguistics, part-of-speech tagging, also called grammatical tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 29 14:30:46 2021

@author: SiddhiThakkar
"""
import spacy
sp = spacy.load('C:/Users/LENOVO/AppData/Local/Programs/Python/Python38/Lib/site-packages/en_core_web_sm/en_core_web_sm-3.0.0')
sen = sp(u"I like to play football. I hated it in my childhood though")
print(sen.text)
print(sen[7].pos_)
print(sen[7].tag_)
print(spacy.explain(sen[7].tag_))
for word in sen:
    print(f'{word.text}:{12}} {word.pos_}:{10}} {word.tag_}:{8}} {spacy.explain(word.tag_)})')
```

Ouput

```
In [8]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/
Practical/Codes/Practical-10/10-i-A.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/
Natural Language Processing/Practical/Codes/Practical-10')

2021-05-29 15:57:01.657934: W tensorflow/stream_executor/platform/default/dso_loader.cc:
60] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not
found
2021-05-29 15:57:01.668220: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore
above cudart dlderror if you do not have a GPU set up on your machine.
I like to play football. I hated it in my childhood though
VERB
VBD
verb, past tense
I          PRON      PRP      pronoun, personal
like       VERB      VBP      verb, non-3rd person singular present
to         PART      TO       infinitival "to"
play       VERB      VB       verb, base form
football   NOUN      NN       noun, singular or mass
.          PUNCT      .        punctuation mark, sentence closer
I          PRON      PRP      pronoun, personal
hated      VERB      VBD      verb, past tense
it         PRON      PRP      pronoun, personal
in         ADP       IN       conjunction, subordinating or preposition
my         PRON      PRP$     pronoun, possessive
childhood  NOUN      NN       noun, singular or mass
though     ADV       RB       adverb
```

B-i. Speech tagging using nltk

Aim: To perform pos tagging in nltk

Concept: In corpus linguistics, part-of-speech tagging, also called grammatical tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 29 14:34:39 2021

@author: LENOVO

import nltk
tokens=nltk.word_tokenize("Can we get information on admission for the academic year 2021")
print("Parts of speech:",nltk.pos_tag(tokens))
```

Ouput

```
In [9]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/
Practical/Codes/Practical-10/10-i-B.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/
Natural Language Processing/Practical/Codes/Practical-10')
Parts of speech: [('Can', 'MD'), ('we', 'PRP'), ('get', 'VB'), ('information', 'NN'),
('on', 'IN'), ('admission', 'NN'), ('for', 'IN'), ('the', 'DT'), ('academic', 'JJ'),
('year', 'NN'), ('2021', 'CD')]
```

A-ii. Usage of Give and Gave in the Penn Treebank sample

Aim: To show usage of give and gave on treebank

Concept: In linguistics, a treebank is a parsed text corpus that annotates syntactic or semantic sentence structure. The construction of parsed corpora in the early 1990s revolutionized computational linguistics, which benefitted from large-scale empirical data

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 29 14:39:52 2021
```

```

@author: SiddhiThakkar
"""

import nltk
def give(t):
    return t.label() == 'VP' and len(t) > 2 and t[1].label() == 'NP'\
           and (t[2].label() == 'PP-DTV' or t[2].label() == 'NP')\
           and ('give' in t[0].leaves() or 'gave' in t[0].leaves())
def sent(t):
    return ''.join(token for token in t.leaves() if token[0] not in '*-0')
def print_node(t, width):
    output = "%s %s: %s / %s: %s" %\
              (sent(t[0]), t[1].label(), sent(t[1]), t[2].label(), sent(t[2]))
    if len(output) > width:
        output = output[:width] + "..."
    print(output)

for tree in nltk.corpus.treebank.parsed_sents():
    for t in tree.subtrees(give):
        print_node(t, 72)

```

Ouput

```

In [10]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/
Practical/Codes/Practical-10/10-ii-A.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester
4/Natural Language Processing/Practical/Codes/Practical-10')
gave NP: the chefs / NP: a standing ovation
give NP: advertisers / NP: discounts for maintaining or increasing ad sp...
give NP: it / PP-DTV: to the politicians
gave NP: them / NP: similar help
give NP: them / NP:
give NP: only French history questions / PP-DTV: to students in a Europe...
give NP: federal judges / NP: a raise
give NP: consumers / NP: the straight scoop on the U.S. waste crisis
gave NP: Mitsui / NP: access to a high-tech medical product
give NP: Mitsubishi / NP: a window on the U.S. glass industry
give NP: much thought / PP-DTV: to the rates she was receiving , nor to ...
give NP: your Foster Savings Institution / NP: the gift of hope and free...
give NP: market operators / NP: the authority to suspend trading in futu...
gave NP: quick approval / PP-DTV: to $ 3.18 billion in supplemental appr...
give NP: the Transportation Department / NP: up to 50 days to review any...
give NP: the president / NP: such power
give NP: me / NP: the heebie-jeebies
give NP: holders / NP: the right , but not the obligation , to buy a cal...
gave NP: Mr. Thomas / NP: only a `` qualified '' rating , rather than ``...
give NP: the president / NP: line-item veto power

```

B-ii. Statistical parsing: probabilistic parser

Aim: To show usage of probabilistic parser

Concept: Probabilistic parsing is using dynamic programming algorithms to compute the most likely parse(s) of a given sentence, given a statistical model of

the syntactic structure of a language. Models have been developed for parsing in several languages other than English, including Chinese, Arabic, and German.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 29 11:33:01 2021

@author: Siddhi Thakkar
"""

import nltk
from nltk import PCFG

grammar = PCFG.fromstring("""
NP -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
NNS -> "men" [0.1] | "women" [0.2] | "children" [0.3] | NNS CC NNS [0.4]
JJ -> "old" [0.4] | "young" [0.6]
CC -> "and" [0.9] | "or" [0.1]
""")

print(grammar)

viterbi_parser = nltk.ViterbiParser(grammar)

token = "old men and women".split()

obj = viterbi_parser.parse(token)

print("Output: ")
for x in obj:
    print(x)
```

Output

```
In [12]: runfile('E:/Siddhi Thakkar/MScIT Part 2/Semester 4/Natural Language Processing/
Practical/Codes/Practical-10/10-B.py', wdir='E:/Siddhi Thakkar/MScIT Part 2/Semester 4/
Natural Language Processing/Practical/Codes/Practical-10')
Grammar with 11 productions (start state = NP)
NP -> NNS [0.5]
NP -> JJ NNS [0.3]
NP -> NP CC NP [0.2]
NNS -> 'men' [0.1]
NNS -> 'women' [0.2]
NNS -> 'children' [0.3]
NNS -> NNS CC NNS [0.4]
JJ -> 'old' [0.4]
JJ -> 'young' [0.6]
CC -> 'and' [0.9]
CC -> 'or' [0.1]
Output:
(NP (JJ old) (NNS (NNS men) (CC and) (NNS women))) (p=0.000864)
```

C. Malt parsing: Parse a sentence and draw a tree using malt parsing.

Aim: to perform malt parsing

Concept: MaltParser is a system for data-driven dependency parsing, which can be used to induce a parsing model from treebank data and to parse new data using an induced model. MaltParser is developed by Johan Hall, Jens Nilsson and Joakim Nivre at Växjö University and Uppsala University, Sweden

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 29 15:01:58 2021

@author: SiddhiThakkar
"""
from nltk.parse import malt
mp =
malt.MaltParser('C:/Users/LENOVO/AppData/Local/Programs/Python/Python38/maltparser-
1.7.2', 'C:/Users/LENOVO/AppData/Local/Programs/Python/Python38/engmalt.linear-
1.7.mco')#file
#t = mp.parse_one('I shot an elephant in my pajamas '.split()).tree()
t = mp.parse_one('I am studing NLP subject in my final semester.'.split()).tree()
print(t)
print(t.draw())
```

Output

