

## Practical 1

### A. Install NLTK Python 3.12.4 Installation on Windows.

**Step 1:** Go to link <https://www.python.org/downloads/>, and select the latest version for windows.  
Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

**Step 2:** Click on the Windows installer (64 bit)

**Step 3:** Select Customize Installation

**Step 4:** Click NEXT

**Step 5:** In next screen

1. Select the advanced options

2. Give a Custom installation location. Keep the default folder as c:\Program files\Python311.9

3. Click Install

**Step 6:** Click Close button once install is done.

**Step 7:** open command prompt window and run the following commands:

```
>>> import nltk
```

Python 3.12.4 Installation on Windows.

Step 1: Step 1) Go to link <https://www.python.org/downloads/> and select the latest version for windows.



Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

Release version	Release date	Click for more	
Python 3.12.4	June 6, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.12.3	April 9, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.11.9	April 2, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.10.14	March 19, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.9.19	March 19, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.8.19	March 19, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.11.8	Feb. 6, 2024	<a href="#">Download</a>	<a href="#">Release Notes</a>

[View older releases](#)

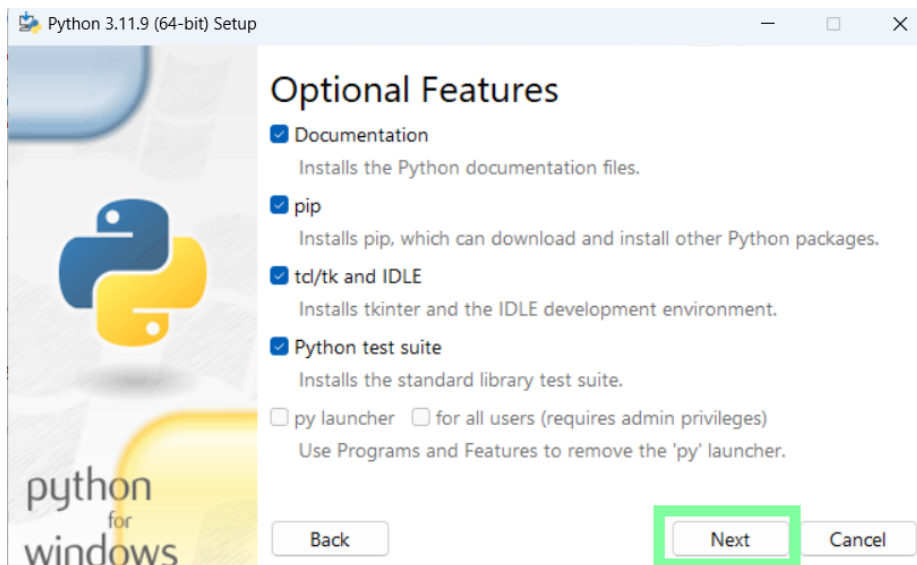
Step 2) Click on the Windows installer (64 bit)

Step 3) Select Customize Installation.



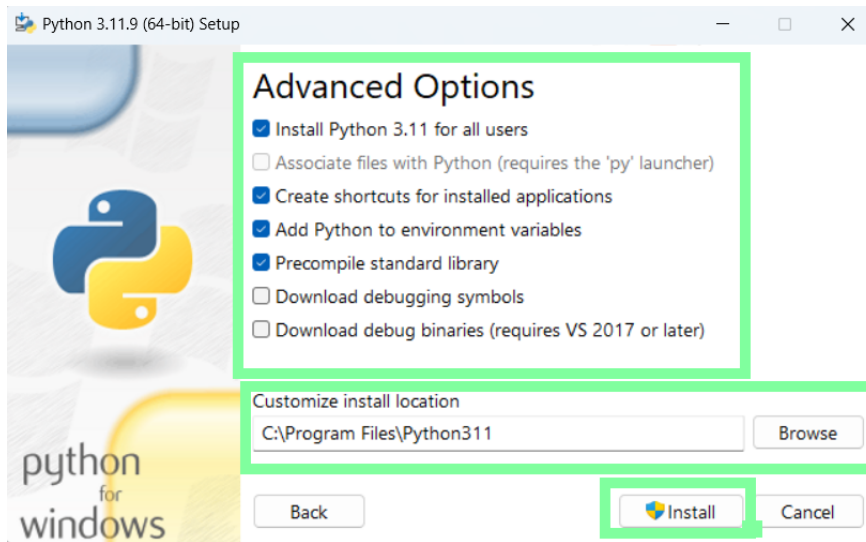
Note: Tick the checkbox of Add python.exe to Path.

Step 5: Click on Next.



Step 5) In next screen

1. Select the advanced options
2. Give a Custom install location. Keep the default folder as c:\Program files\Python311.9
3. Click Install



Step 6) Click Close button once install is done.

Step 7) Open Command Prompt and write pip install nltk.

```
Command Prompt
Microsoft Windows [Version 10.0.22631.3810]
(c) Microsoft Corporation. All rights reserved.

C:\Users\schau>pip install nltk
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nltk in c:\users\schau\appdata\roaming\python\python311\site-packages (3.8.1)
Requirement already satisfied: click in c:\users\schau\appdata\roaming\python\python311\site-packages (from nltk) (8.1.3)
Requirement already satisfied: joblib in c:\users\schau\appdata\roaming\python\python311\site-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in c:\users\schau\appdata\roaming\python\python311\site-packages (from nltk) (2022.10.31)
Requirement already satisfied: tqdm in c:\users\schau\appdata\roaming\python\python311\site-packages (from nltk) (4.65.0)
Requirement already satisfied: colorama in c:\users\schau\appdata\roaming\python\python311\site-packages (from click->nltk) (0.4.6)
```

**B. Convert the given text to speech.****Code:**

```
# Import the required module for text to speech conversion

!pip install gtts
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 = "Hello Everyone!My name is Rahul"

# 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("welcomeNK.mp3")
# Playing the converted file
#os.system("mpg321 welcomeNK.mp3")
```

**Output:**

```
Requirement already satisfied: gtts in c:\program files\python38\lib\site-packages (2.5.1)
Requirement already satisfied: requests<3,>=2.27 in c:\program files\python38\lib\site-packages (from gtts) (2.31.0)
Requirement already satisfied: click<8.2,>=7.1 in c:\program files\python38\lib\site-packages (from gtts) (8.1.7)
Requirement already satisfied: colorama in c:\program files\python38\lib\site-packages (from click<8.2,>=7.1->gtts) (0.4.6)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\program files\python38\lib\site-packages (from requests<3,>=2.27->gtts) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in c:\program files\python38\lib\site-packages (from requests<3,>=2.27->gtts) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\program files\python38\lib\site-packages (from requests<3,>=2.27->gtts) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\program files\python38\lib\site-packages (from requests<3,>=2.27->gtts) (2023.7.22)
```

 welcomeNK.mp3

1 minute ago

**Note:** welcomeNLP.mp3 audio file is getting created and it plays the file with play sound() method, while running the program.

**C. Convert audio file Speech to Text.****Source code:**

Note: required to store the input file " welcomeNK.mp3" in the current folder before running the program.

**Code:**

```
!pip install SpeechRecognition
!pip install pydub
```

```
import speech_recognition as sr # Importing the SpeechRecognition library.
```

```
from pydub import AudioSegment # Importing the AudioSegment class from the pydub library.

# Initialize the recognizer
recognizer = sr.Recognizer() # Creating a Recognizer instance for speech recognition.

# Load the MP3 audio file
audio_file_mp3 = "/content/welcomeNK.mp3" # Setting the path to the input MP3 audio file.

# Load audio file as AudioSegment
audio = AudioSegment.from_mp3(audio_file_mp3) # Loading the MP3 audio file as an
AudioSegment object.


# Export AudioSegment to WAV format
audio_file_wav = "/content/welcomeNK.wav" # Setting the path for the output WAV audio file.
audio.export(audio_file_wav, format="wav") # Exporting the AudioSegment to WAV format.

# Load WAV audio file as audio data
with sr.AudioFile(audio_file_wav) as source: # Opening the WAV audio file as an AudioFile object.
    audio_data = recognizer.record(source) # Recording the audio data from the WAV file.

# Recognize speech using Google Speech Recognition
try:
    text = recognizer.recognize_google(audio_data) # Using Google Speech Recognition to recognize
speech from the audio data.
    print("Recognized speech:", text) # Printing the recognized speech.
except sr.UnknownValueError:
    print("Speech recognition could not understand the audio") # Handling unrecognized speech.
except sr.RequestError as e:
    print("Could not request results from Google Speech Recognition service; {0}".format(e)) #
Handling request errors.
```

**Output:**

Recognized speech: hello everyone my name is Rahul

 welcomeNK.wav

## Practical 2

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

**Code:**

```
!pip install nltk
```

```
import nltk # Importing the NLTK library
from nltk.corpus import brown # Importing the Brown corpus from NLTK
nltk.download('brown')
# Displaying the file identifiers of the Brown corpus
print ('File ids of brown corpus\n', brown.fileids())

# Loading the words from the 'ca01' file of the Brown corpus
ca01 = brown.words('ca01')

# Displaying the first few words from the 'ca01' file
print('\nca01 has following words:\n', ca01[:10]) # Displaying first 10 words

# Counting the total number of words in the 'ca01' file
print('\nca01 has', len(ca01), 'words')

# Displaying the categories or files in the Brown corpus
print ('\n\nCategories or file in brown corpus:\n')
print (brown.categories())

# Displaying statistics for each text in the Brown corpus
print ('\n\nStatistics for each text:\n')
print('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\t\tFileName')

# Looping over all file identifiers in the Brown corpus
for fileid in brown.fileids():
    num_chars = len(brown.raw(fileid)) # Counting the number of characters
    num_words = len(brown.words(fileid)) # Counting the number of words
    num_sents = len(brown.sents(fileid)) # Counting the number of sentences
    num_vocab = len(set([w.lower() for w in brown.words(fileid)])) # Counting the vocabulary size

    # Computing average word length, average sentence length, and average number of times each
    word appears
    avg_word_len = int(num_chars / num_words)
    avg_sent_len = int(num_words / num_sents)
    avg_word_freq = int(num_words / num_vocab)

    # Printing the statistics for each text
    print (avg_word_len, '\t\t\t', avg_sent_len, '\t\t\t', avg_word_freq, '\t\t\t', fileid)
```

**Output:**

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data] Unzipping corpora/brown.zip.
File ids of brown corpus
['ca01', 'ca02', 'ca03', 'ca04', 'ca05', 'ca06', 'ca07', 'ca08', 'ca09', 'ca10', 'ca11', 'ca12', 'ca13', 'ca14', 'ca15']

ca01 has following words:
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of']

ca01 has 2242 words

Categories or file in brown corpus:

['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery',
```

Statistics for each text:

AvgWordLen	AvgSentenceLen	no.ofTimesEachWordAppearsOnAvg	FileName
9	22	2	ca01
8	23	2	ca02
8	20	2	ca03
9	25	2	ca04
8	26	3	ca05
8	22	2	ca06

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

```
!pip install nltk
import nltk # Importing the NLTK library
from nltk.corpus import PlaintextCorpusReader # Importing PlaintextCorpusReader from NLTK
nltk.download('punkt')
# Define the path to the directory containing the corpus files
corpus_root = '/content/sample_data'

# Create a PlaintextCorpusReader object to access the corpus files
filelist = PlaintextCorpusReader(corpus_root, '.*')

# Display the file list
print('\n File list: \n')
print(filelist.fileids()) # Printing the file identifiers
print(filelist.root) # Printing the root directory of the corpus

# Display statistics for each text in the corpus
print('\n\nStatistics for each text:\n')
print('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\tFileName')

# Loop over all file identifiers in the corpus
for fileid in filelist.fileids():
    # Counting the number of characters, words, sentences, and vocabulary size for each text
    num_chars = len(filelist.raw(fileid))
    num_words = len(filelist.words(fileid))
    num_sents = len(filelist.sents(fileid))
    num_vocab = len(set([w.lower() for w in filelist.words(fileid)]))
```

```
# Computing average word length, average sentence length, and average word frequency for each
text
avg_word_len = int(num_chars / num_words)
avg_sent_len = int(num_words / num_sents)
avg_word_freq = int(num_words / num_vocab)

# Printing the statistics for each text
print(avg_word_len, '\t\t\t', avg_sent_len, '\t\t\t', avg_word_freq, '\t\t\t', fileid)
```

**Output:**

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.

File list:

['README.md', 'anscombe.json', 'california_housing_test.csv', 'california_housing_train.csv', 'mnist_test.csv', 'mnist_train_small.csv']
/content/sample_data

Statistics for each text:

AvgWordLen    AvgSentenceLen  no.ofTimesEachWordAppearsOnAvg  FileName
4              18                1                                README.md
2              209              14                               anscombe.json
2              108019           15                               california_housing_test.csv
2              612019           43                               california_housing_train.csv
1              15690000         61050                            mnist_test.csv
1              31380000        122101                           mnist_train_small.csv
```

**C. Study Conditional frequency distributions.****Code:**

```
!pip install nltk
import nltk
from nltk.corpus import brown, inaugural, udhr

# Download required NLTK data
nltk.download('brown')
nltk.download('inaugural')
nltk.download('udhr')

# Process a sequence of pairs
text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]

# Construct a Conditional Frequency Distribution
fd = nltk.ConditionalFreqDist(
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories=genre)
)
```



```

# Create a list of genre-word pairs
genre_word = [
    (genre, word)
    for genre in ['news', 'romance']
    for word in brown.words(categories=genre)
]

# Print the length and sample pairs from genre_word
print(len(genre_word))
print(genre_word[:4])
print(genre_word[-4:])

# Create and print the Conditional Frequency Distribution
cfd = nltk.ConditionalFreqDist(genre_word)
print(cfd)
print(cfd.conditions())
print(cfd['news'])
print(cfd['romance'])
print(list(cfd['romance']))
# Create a CFD for inaugural corpus
cfd = nltk.ConditionalFreqDist(
    (target, fileid[:4])
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ['america', 'citizen']
    if w.lower().startswith(target)
)
# Define languages for udhr corpus
languages = ['Chickasaw', 'English', 'German_Deutsch',
             'Greenlandic_Inuktitut', 'Hungarian_Magyar', 'Ibibio_Efik']

# Create and tabulate CFD for udhr corpus
cfd = nltk.ConditionalFreqDist(
    (lang, len(word))
    for lang in languages
    for word in udhr.words(lang + '-Latin1')
)
cfd.tabulate(conditions=['English', 'German_Deutsch'], samples=range(10), cumulative=True)

```

**Output:**

```

[('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')]
[('romance', 'afraid'), ('romance', 'not'), ('romance', ''), ('romance', '.')]
<ConditionalFreqDist with 2 conditions>
['news', 'romance']
<FreqDist with 14394 samples and 100554 outcomes>
<FreqDist with 8452 samples and 70022 outcomes>
[,',', '.', 'the', 'and', 'to', 'a', 'of', '', '', 'was', 'I', 'in', 'he', 'had',
      0   1   2   3   4   5   6   7   8   9
      English  0  185  525  883  997 1166 1283 1440 1558 1638
      German_Deutsch  0  171  263  614  717  894 1013 1110 1213 1275

```

**D. Study of tagged corpora with methods like tagged\_sents, tagged\_words.****Code:**

```
!pip install nltk
import nltk
from nltk import tokenize

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')

# Paragraph for tokenization
para = "Hello! My name is Pikachu. Today you'll be learning NLTK."

# Sentence tokenization
sents = tokenize.sent_tokenize(para)
print("\nSentence Tokenization\n=====\\n", sents)

# Word tokenization and POS tagging
print("\nWord Tokenization and POS Tagging\n=====\\n")
tagged_sents = []
tagged_words = []

for index in range(len(sents)):
    words = tokenize.word_tokenize(sents[index])
    tagged = nltk.pos_tag(words)
    tagged_sents.append(tagged)
    tagged_words.extend(tagged)
    print(words)
    print(tagged)

# Display tagged sentences and tagged words
print("\nTagged Sentences\n=====\\n", tagged_sents)
print("\nTagged Words\n=====\\n", tagged_words)

# Explanation of POS Tags:
# NN: Noun, singular or mass
# PRP$: Possessive pronoun
# VBZ: Verb, 3rd person singular present
# NNP: Proper noun, singular
# PRP: Personal pronoun
# MD: Modal verb
# VB: Verb, base form
# VBG: Verb, gerund or present participle
# .: Punctuation
```

**Output:**

Sentence Tokenization

=====

['Hello!', 'My name is Pikachu.', "Today you'll be learning NLTK."]

Word Tokenization and POS Tagging

=====

['Hello', '!']

[(('Hello', 'NN'), ('!', '.'))]

['My', 'name', 'is', 'Pikachu', '.']

[(('My', 'PRP\$'), ('name', 'NN'), ('is', 'VBZ'), ('Pikachu', 'NNP'), ('.', '.'))]

['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']

[(('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.'))]

Tagged Sentences

=====

[[('Hello', 'NN'), ('!', '.')], [(('My', 'PRP\$'), ('name', 'NN'), ('is', 'VBZ'), ('Pikachu', 'NNP'), ('.', '.'))],

Tagged Words

=====

[(('Hello', 'NN'), ('!', '.'), ('My', 'PRP\$'), ('name', 'NN'), ('is', 'VBZ'), ('Pikachu', 'NNP'), ('.', '.'), ('Tc

**E. Write a program to find the most frequent noun tags.****Code:**

```

import nltk
from collections import defaultdict
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
# Tokenize the text
text = nltk.word_tokenize("Shivam likes to play football. Shivam does not like to play cricket.")

# POS tagging
tagged = nltk.pos_tag(text)
print(tagged)

# Check if the word is a noun and collect noun words
addNounWords = []
for word, tag in tagged:
    if tag in ('NN', 'NNS', 'NNPS', 'NNP'):
        addNounWords.append(word)

print(addNounWords)

# Memoize the count of each noun
temp = defaultdict(int)
for noun in addNounWords:
    temp[noun] += 1

# Get the word with maximum frequency
res = max(temp, key=temp.get)

# Print the result

```

```
print("Word with maximum frequency : " + str(res))
```

**Output:**

```
[('Shivam', 'NNP'), ('likes', 'VBZ'), ('to', 'TO'), ('play', 'VB'), ('football', 'NN'),  
['Shivam', 'football', 'Shivam', 'cricket']  
Word with maximum frequency : Shivam  
_
```

**F. Map Words to Properties Using Python Dictionaries.****Code:**

```
# Creating and printing a dictionary by mapping words with their properties
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
# Printing the entire dictionary
```

```
print("The entire dictionary:")  
print(thisdict)
```

```
# Printing the value associated with the key "brand"
```

```
print("\nValue associated with the key 'brand':")  
print(thisdict["brand"])
```

```
# Printing the number of items in the dictionary
```

```
print("\nNumber of items in the dictionary:")  
print(len(thisdict))
```

```
# Printing the type of the dictionary
```

```
print("\nType of the dictionary:")  
print(type(thisdict))
```

**Output:**

```
The entire dictionary:  
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

```
Value associated with the key 'brand':  
Ford
```

```
Number of items in the dictionary:  
3
```

```
Type of the dictionary:  
<class 'dict'>
```

**G. Study DefaultTagger, Regular expression tagger, UnigramTagger.****i. DefaultTagger****Code:**

```
import nltk
from nltk.tag import DefaultTagger
from nltk.corpus import treebank
nltk.download('treebank')

# Create a default tagger that tags everything as 'NN' (noun)
exptagger = DefaultTagger('NN')

# Evaluate the tagger on a subset of the Treebank corpus
testsentences = treebank.tagged_sents()[1000:]
evaluation_score = exptagger.evaluate(testsentences)

# Print the evaluation score
print("Evaluation score of the default tagger on test sentences:")
print(evaluation_score)

# Tagging a list of sentences using the default tagger
tagged_sentences = exptagger.tag_sents([['Hi', ','], ['How', 'are', 'you', '?']])

# Print the tagged sentences
print("\nTagged sentences:")
print(tagged_sentences)
```

**Output:**

```
Evaluation score of the default tagger on test sentences:
0.13198749536374715

Tagged sentences:
[[('Hi', 'NN'), (',', 'NN')], [('How', 'NN'), ('are', 'NN'), ('you', 'NN'), ('?', 'NN')]]
```

**ii. Regular expression tagger****Code:**

```
from nltk.corpus import brown
from nltk.tag import RegexpTagger

# Load a sample sentence from the Brown corpus under 'news' category
test_sent = brown.sents(categories='news')[0]

# Define a RegexpTagger with regular expression patterns and corresponding tags
regexp_tagger = RegexpTagger([
    (r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
    (r'(The|the|A|a|An|an)$', 'AT'), # articles
    (r'.*able$', 'JJ'), # adjectives
```

```
(r'.*ness$', 'NN'),      # nouns formed from adjectives
(r'.*ly$', 'RB'),        # adverbs
(r'.*s$', 'NNS'),        # plural nouns
(r'.*ing$', 'VBG'),       # gerunds
(r'.*ed$', 'VBD'),       # past tense verbs
(r'.*', 'NN')            # nouns (default)
])
```

```
# Print the RegexpTagger object
print("RegexpTagger object:")
print(regex_tagger)
```

```
# Tag the sample sentence using the RegexpTagger
tagged_sentence = regex_tagger.tag(test_sent)
```

```
# Print the tagged sentence
print("\nTagged sentence:")
print(tagged_sentence)
```

**Output:**

```
RegexpTagger object:
<Regexp Tagger: size=9>
```

```
Tagged sentence:
[('The', 'AT'), ('Fulton', 'NN'), ('County', 'NN'), ('Grand', 'NN'), ('Jury', 'NN'), ('said', 'NN'), ('Friday', 'NN'), ('an', 'AT'), ('investigation', 'NN'),
```

**iii. UnigramTagger****Code:**

```
from nltk.tag import UnigramTagger
from nltk.corpus import treebank

# Load the first 10 tagged sentences of the treebank corpus as training data
train_sents = treebank.tagged_sents()[0:10]

# Initialize the UnigramTagger with the training sentences
tagger = UnigramTagger(train_sents)

# Print the first sentence of the treebank corpus as a list of words
print("First sentence of the treebank corpus:")
print(treebank.sents()[0])

# Tag the first sentence using the trained UnigramTagger
print("\nTagged sentence:")
print(tagger.tag(treebank.sents()[0]))

# Override the context model with a specific mapping
tagger = UnigramTagger(model={'Pierre': 'NN'})
```

```
# Tag the first sentence again with the overridden context model
print("\nTagged sentence with overridden model:")
print(tagger.tag(treebank.sents()[0]))
```

**Output:**

```
First sentence of the treebank corpus:
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.']

Tagged sentence:
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), ('the', 'DT'), ('board', 'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'NN'), ('director', 'NN'), ('Nov.', 'NNP'), ('29', 'NNP'), ('.', '.')]

Tagged sentence with overridden model:
[('Pierre', 'NN'), ('Vinken', None), (',', None), ('61', None), ('years', None), ('old', None), (',', None), ('will', None), ('join', None), ('the', None), ('board', None), ('as', None), ('a', None), ('nonexecutive', None), ('director', None), ('Nov.', None), ('29', None), ('.', None)]
```

### H. 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.

**Code:**

```
from __future__ import with_statement # To use 'with' statement for file reading
import re # Regular expression

# Sample words.txt file content (example words)
# You can replace this with your actual words.txt file content
# Example:
# hello
# what
# is
# my
# name
# www
# com
words_file_content = """\
hello
what
is
my
name
www
com
"""

# Write the sample content to words.txt
with open('words.txt', 'w') as f:
    f.write(words_file_content)

# Initialize variables
words = [] # List to store corpus words
testword = [] # List to store test words
ans = [] # List to store matched words from corpus

# Menu display and user input
```

```
print("MENU")
print("-----")
print(" 1. Hash tag segmentation")
print(" 2. URL segmentation")
print("Enter the input choice for performing word segmentation:")
choice = int(input())

# Perform segmentation based on user choice
if choice == 1:
    text = "#whatismyname" # Example hash tag test data
    print("Input with HashTag:", text)
    pattern = re.compile("[^\w'"]")
    a = pattern.sub('', text)
elif choice == 2:
    text = "www.whatismyname.com" # Example URL test data
    print("Input with URL:", text)
    splitwords = ["www", "com"] # Words to be removed from segmentation
    a = re.split(r'\W+', text) # Splitting based on non-word characters
    a = " ".join([each for each in a if each not in splitwords])
else:
    print("Wrong choice...try again")
    exit()

print("Segmented Output:", a)

# Read the words from words.txt into the words list
with open('words.txt', 'r') as f:
    lines = f.readlines()
    words = [e.strip() for e in lines]

# Function to perform segmentation based on corpus words
def Seg(a, length):
    ans = []
    for k in range(0, length + 1):
        if a[0:k] in words:
            print(a[0:k], "- Appears in the corpus")
            ans.append(a[0:k])
            break
    if ans != []:
        g = max(ans, key=len)
        return g

test_length = len(a) # Length of the segmented test data
test_total_iter = 0 # Initialize total iteration count
answer = [] # List to store segmented words
while test_total_iter < test_length:
    ans_words = Seg(a, test_length)
    if ans_words:
        test_iter = len(ans_words)
        answer.append(ans_words)
```



```
a = a[test_iter:test_length]
test_total_iter += test_iter

# Join segmented words into a sentence
after_segmentation = " ".join(answer)
print("\nOutput")
print("-----")
print("After segmentation:", after_segmentation)

# Calculate and print score
N = len(words) # Total number of words in the corpus (N = 7 based on the sample words.txt)
C = len(answer) # Number of words segmented
score = C * N / N # Calculate the score
print("Score:", score)
```

**Output:**

```
MENU
-----
1. Hash tag segmentation
2. URL segmentation
Enter the input choice for performing word segmentation:
2
Input with URL: www.whatismyname.com
Segmented Output: whatismyname
what - Appears in the corpus
is - Appears in the corpus
my - Appears in the corpus
name - Appears in the corpus

Output
-----
After segmentation: what is my name
Score: 4.0
```

## Practical 3

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

**Code:**

```
import nltk
from nltk.corpus import wordnet

# Print synsets (collections of synonym words or lemmas) for the word 'car'
print("Synsets for 'car':")
print(wordnet.synsets("car"))

# Print definition of the word 'car' in its first synset
print("\nDefinition of 'car':")
print(wordnet.synset("car.n.01").definition())

# Print examples of the word 'car' in its first synset
print("\nExamples of 'car':")
print("Examples:", wordnet.synset("car.n.01").examples())

# Get antonyms of the word 'buy' in its first lemma
print("\nAntonyms of 'buy':")
print(wordnet.lemma('buy.v.01.buy').antonyms())
```

**Output:**

```
Synsets for 'car':
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'), Synset('car.n.04'), Synset('cable_car.n.01')]

Definition of 'car':
a motor vehicle with four wheels; usually propelled by an internal combustion engine

Examples of 'car':
Examples: ['he needs a car to get to work']

Antonyms of 'buy':
[Lemma('sell.v.01.sell')]
```

### B. Study lemmas, hyponyms, hypernyms.

**Code:**

```
import nltk
from nltk.corpus import wordnet
from tabulate import tabulate

# Get synsets for the word 'car'
car_synsets = wordnet.synsets("car")

# Prepare data for synsets and their lemma names
synset_data = []
for synset in car_synsets:
    synset_data.append([synset.name(), ', '.join(synset.lemma_names())])
```

```
# Print synsets and lemma names in a table
print("Synsets and Lemma Names for 'car':")
print(tabulate(synset_data, headers=['Synset', 'Lemma Names'], tablefmt='pretty'))
# Print all lemmas for 'car.n.01'
car_lemmas = wordnet.synset('car.n.01').lemmas()
lemma_data = [lemma.name() for lemma in car_lemmas]
print("\nLemmas for 'car.n.01':")
print(', '.join(lemma_data))
# Get the synset corresponding to the lemma 'car.n.01.automobile'
automobile_synset = wordnet.lemma('car.n.01.automobile').synset()
print("\nSynset for 'car.n.01.automobile':")
print(automobile_synset.name())
# Get the name of the lemma 'car.n.01.automobile'
print("\nName of the lemma 'car.n.01.automobile':")
print(wordnet.lemma('car.n.01.automobile').name())
# Hyponyms of 'car.n.01' (specific types of cars)
car_synset = wordnet.synset('car.n.01')
car_hyponyms = car_synset.hyponyms()
hyponym_data = [lemma.name() for synset in car_hyponyms for lemma in synset.lemmas()]
print("\nHyponyms of 'car.n.01':")
print(', '.join(hyponym_data))

# Hypernyms (general concepts) shared between 'car.n.01' and 'vehicle.n.01'
vehicle_synset = wordnet.synset('vehicle.n.01')
common_hypernyms = car_synset.lowest_common_hypernyms(vehicle_synset)
hypernym_data = [hypernym.name() for hypernym in common_hypernyms]
print("\nLowest Common Hypernyms between 'car.n.01' and 'vehicle.n.01':")
print(', '.join(hypernym_data))
```

**Output:**

```
Synsets and Lemma Names for 'car':
```

Synset	Lemma Names
car.n.01	car, auto, automobile, machine, motorcar
car.n.02	car, railcar, railway_car, railroad_car
car.n.03	car, gondola
car.n.04	car, elevator_car
cable_car.n.01	cable_car, car

```
Lemmas for 'car.n.01':
```

```
car, auto, automobile, machine, motorcar
```

```
Synset for 'car.n.01.automobile':
```

```
car.n.01
```

```
Name of the lemma 'car.n.01.automobile':
```

```
automobile
```

```
Hyponyms of 'car.n.01':
```

```
ambulance, beach_wagon, station_wagon, wagon, estate_car, beach_waggon,
```

```
Lowest Common Hypernyms between 'car.n.01' and 'vehicle.n.01':
```

```
vehicle.n.01
```

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

#### Code:

```
from nltk.corpus import wordnet
from tabulate import tabulate

# Finding synsets for the word 'active'
active_synsets = wordnet.synsets("active")

# Prepare data for synsets and their definitions
synset_data = []
for synset in active_synsets:
    synset_data.append([synset.name(), synset.definition()])

# Finding antonyms of the lemma 'active.a.01.active'
active_lemma = wordnet.lemma('active.a.01.active')
antonyms = active_lemma.antonyms()
antonym_names = ', '.join([antonym.name() for antonym in antonyms])

# Format output in table
print("Synsets for 'active':")
print(tabulate(synset_data, headers=['Synset', 'Definition'], tablefmt='pretty'))

# Print Antonyms in a table format
print("\nAntonyms:")
print(tabulate([[antonym_names]], headers=['Antonyms'], tablefmt='pretty'))
```

#### Output:

Synsets for 'active':

Synset	Definition
active_agent.n.01	chemical agent capable of activity
active_voice.n.01	the voice used to indicate that the grammatical subject of the verb is performing the action or causing the happening denoted by the verb
active.n.03	a person who is a participating member of an organization
active.a.01	tending to become more severe or wider in scope
active.s.02	engaged in or ready for military or naval operations
active.a.03	disposed to take action or effectuate change
active.s.04	taking part in an activity
active.a.05	characterized by energetic activity
active.a.06	exerting influence or producing a change or effect
active.a.07	full of activity or engaged in continuous activity
active.s.08	in operation
active.a.09	(of the sun) characterized by an increased occurrence of sunspots and flares and radio emissions
active.a.10	expressing that the subject of the sentence has the semantic function of actor:
active.a.11	(used of verbs (e.g. 'to run') and participial adjectives (e.g. 'running' in 'running water')) expressing action rather than a state of being
active.a.12	(of e.g. volcanos) capable of erupting
active.a.13	(of e.g. volcanos) erupting or liable to erupt
active.a.14	engaged in full-time work

Antonyms:

Antonyms
inactive

**D. Compare two nouns****Code:**

```
import nltk
from nltk.corpus import wordnet

# Get synsets for 'football' and 'soccer'
syn1 = wordnet.synsets('football')
syn2 = wordnet.synsets('soccer')

# Compare each synset of 'football' with each synset of 'soccer'
comparison_data = []

for s1 in syn1:
    for s2 in syn2:
        similarity = s1.path_similarity(s2)
        if similarity is not None: # Check if similarity is not None
            comparison_data.append({
                'Synset for Football': f"{s1} ({s1.pos()}) [{s1.definition()}]",
                'Synset for Soccer': f"{s2} ({s2.pos()}) [{s2.definition()}]",
                'Path Similarity': f"{similarity:.2f}" # Format similarity to 2 decimal places
            })

# Print comparison results in sections
print("Comparison of synsets for 'football' and 'soccer':")
for idx, comparison in enumerate(comparison_data, start=1):
    print(f"Comparison {idx}:")
    print("Synset for Football:", comparison['Synset for Football'])
    print("Synset for Soccer:", comparison['Synset for Soccer'])
    print("Path Similarity:", comparison['Path Similarity'])
    print()
```

**Output:**

```
Comparison of synsets for 'football' and 'soccer':
Comparison 1:
Synset for Football: Synset('football.n.01') (n) [any of various games played with a ball (round or oval) in which two teams try to kick or carry or propel the ball into each other's goal]
Synset for Soccer: Synset('soccer.n.01') (n) [a football game in which two teams of 11 players try to kick or head a ball into the opponents' goal]
Path Similarity: 0.50

Comparison 2:
Synset for Football: Synset('football.n.02') (n) [the inflated oblong ball used in playing American football]
Synset for Soccer: Synset('soccer.n.01') (n) [a football game in which two teams of 11 players try to kick or head a ball into the opponents' goal]
Path Similarity: 0.05
```

**E. Handling stopwords :****i. Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List.****Code:**

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

```
# Download stopwords data
nltk.download('stopwords')
# Input text
text = "Nobita enjoys playing football, but he isn't particularly fond of tennis."
# Tokenize the text
text_tokens = word_tokenize(text)

# Remove stopwords from NLTK's stopwords list
tokens_without_sw_default = [word for word in text_tokens if not word in
stopwords.words('english')]

# Print output with default stopwords list
print("Using default NLTK stopwords:")
print("Word Tokens:", text_tokens)
print("Tokens without Stopwords:", tokens_without_sw_default)
print()

# Add 'play' to the NLTK stopwords collection
all_stopwords = stopwords.words('english')
all_stopwords.append('play')

# Remove stopwords including 'play'
tokens_without_sw_with_play = [word for word in text_tokens if not word in all_stopwords]

# Print output with 'play' added to stopwords
print("Adding 'play' to NLTK stopwords:")
print("Word Tokens:", text_tokens)
print("Tokens without Stopwords (including 'play'):", tokens_without_sw_with_play)
print()

# Remove 'not' from NLTK stopwords collection
all_stopwords.remove('not')

# Remove stopwords excluding 'not'
tokens_without_sw_without_not = [word for word in text_tokens if not word in all_stopwords]

# Print output after removing 'not' from stopwords
print("Removing 'not' from NLTK stopwords:")
print("Word Tokens:", text_tokens)
print("Tokens without Stopwords (excluding 'not'):", tokens_without_sw_without_not)
```

**Output:**

```
Using default NLTK stopwords:
Word Tokens: ['Nobita', 'enjoys', 'playing', 'football', ',', 'but', 'he', 'is', "n't", 'particularly', 'fond', 'of', 'tennis', '.']
Tokens without Stopwords: ['Nobita', 'enjoys', 'playing', 'football', ',', "n't", 'particularly', 'fond', 'tennis', '.']

Adding 'play' to NLTK stopwords:
Word Tokens: ['Nobita', 'enjoys', 'playing', 'football', ',', 'but', 'he', 'is', "n't", 'particularly', 'fond', 'of', 'tennis', '.']
Tokens without Stopwords (including 'play'): ['Nobita', 'enjoys', 'playing', 'football', ',', "n't", 'particularly', 'fond', 'tennis', '.']

Removing 'not' from NLTK stopwords:
Word Tokens: ['Nobita', 'enjoys', 'playing', 'football', ',', 'but', 'he', 'is', "n't", 'particularly', 'fond', 'of', 'tennis', '.']
Tokens without Stopwords (excluding 'not'): ['Nobita', 'enjoys', 'playing', 'football', ',', "n't", 'particularly', 'fond', 'tennis', '.']
```

**ii. Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List.****Code:**

```
#!/pip install gensim

import gensim
from gensim.parsing.preprocessing import remove_stopwords
from nltk.tokenize import word_tokenize
from tabulate import tabulate

# Original text
text = "Ben likes to play football, however he is not too fond of tennis."

# Remove stopwords using Gensim's default stopwords list
filtered_sentence = remove_stopwords(text)

# Get Gensim's default stopwords list
all_stopwords = gensim.parsing.preprocessing.STOPWORDS

# Add 'likes' and 'play' to Gensim's default stopwords list
from gensim.parsing.preprocessing import STOPWORDS
all_stopwords_gensim = STOPWORDS.union(set(['likes', 'play']))

# Tokenize text and remove stopwords including 'likes' and 'play'
text_tokens = word_tokenize(text)
tokens_without_sw1 = [word for word in text_tokens if not word in all_stopwords_gensim]

# Remove 'not' from Gensim's default stopwords list
all_stopwords_gensim = STOPWORDS.difference({"not"})

# Tokenize text and remove stopwords excluding 'not'
text_tokens = word_tokenize(text)
tokens_without_sw2 = [word for word in text_tokens if not word in all_stopwords_gensim]

# Prepare data for table
data = [
    ["1. Remove stopwords using Gensim's default stopwords list:", filtered_sentence],
    ["2. Gensim's default stopwords list:", '\n'.join(all_stopwords)],
    ["3. Remove 'likes' and 'play' from Gensim's default stopwords list:", tokens_without_sw1],
    ["4. Remove 'not' from Gensim's default stopwords list:", tokens_without_sw2],
    ["Explanation:", "Using Gensim: Adding and Removing Stop Words in Default Gensim Stop Words List"]
]

# Print table
print(tabulate(data, headers=['Output Section', 'Content'], tablefmt='grid'))
```

**Output:**

	thereby	
3. Remove 'likes' and 'play' from Gensim's default stopwords list:	['Ben', 'football', ',', 'fond', 'tennis', '.']	
4. Remove 'not' from Gensim's default stopwords list:	['Ben', 'likes', 'play', 'football', ',', 'not', 'fond', 'tennis', '.']	
Explanation:	Using Gensim: Adding and Removing Stop Words in Default Gensim Stop Words List	

**iii. Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List.****Code:**

```
!pip install spacy
!python -m spacy download en_core_web_sm
!python -m spacy download en

import spacy
from nltk.tokenize import word_tokenize
from tabulate import tabulate

# Load English language model in Spacy
sp = spacy.load('en_core_web_sm')

# Get Spacy's default stopwords
all_stopwords = sp.Defaults.stop_words

# Add 'play' to Spacy's default stopwords
all_stopwords.add("play")

# Define the text to process
text = "Sara likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)

# Remove stopwords including 'play' from tokenized text
tokens_without_sw1 = [word for word in text_tokens if not word in all_stopwords]

# Check if 'not' is in stop word collection before removing it
if 'not' in all_stopwords:
    all_stopwords.remove('not')

# Remove stopwords excluding 'not' from tokenized text
tokens_without_sw2 = [word for word in text_tokens if not word in all_stopwords]

# Prepare data for table
data = [
    ["1. Remove stopwords including 'play' from Spacy's default stop words list:",
tokens_without_sw1],
    ["2. Remove stopwords excluding 'not' from Spacy's default stop words list:",
tokens_without_sw2],
    ["Explanation:", "Using Spacy: Adding and Removing Stop Words in Default Spacy Stop Words
List"]]
```



]

```
# Print table
print(tabulate(data, headers=['Output Section', 'Content'], tablefmt='grid'))
```

**Output:**

Output Section	Content
1. Remove stopwords including 'play' from Spacy's default stop words list:	['Sara', 'likes', 'football', ',', 'not', 'fond', 'tennis', '.']
2. Remove stopwords excluding 'not' from Spacy's default stop words list:	['Sara', 'likes', 'football', ',', 'not', 'fond', 'tennis', '.']
Explanation:	Using Spacy: Adding and Removing Stop Words in Default Spacy Stop Words List

## Practical 4

### A. Tokenization using Python's split() function

#### Code:

```
text = """This tool is in an a beta stage. Alexa developers can use Get Metrics API to seamlessly analyze metrics. It also supports custom skill models, prebuilt Flash Briefing models, and the Smart Home Skill API. You can use this tool for creation of monitors, alarms, and dashboards that spotlight changes. The release of these three tools will enable developers to create visually rich skills for Alexa devices with screens. Amazon describes these tools as the collection of tech and tools for creating visually rich and interactive voice experiences."""
```

```
# Tokenization using split() with improved handling of periods within sentences
sentences = text.split(". ") # Split on periods followed by a space
```

```
for sentence in sentences:
```

```
    # Optionally, remove leading/trailing whitespace (customizable)
    sentence = sentence.strip()
    print(sentence)
```

#### Output:

```
This tool is in an a beta stage
Alexa developers can use Get Metrics API to seamlessly analyze metrics
It also supports custom skill models, prebuilt Flash Briefing models, and the Smart Home Skill API
You can use this tool for creation of monitors, alarms, and dashboards that spotlight changes
The release of these three tools will enable developers to create visually rich skills for Alexa devices with screens
Amazon describes these tools as the collection of tech and tools for creating visually rich and interactive voice experiences.
```

### B. Tokenization using Regular Expressions (RegEx)

#### Code:

```
import nltk
from nltk.tokenize import RegexpTokenizer
# Create a reference variable for Class RegexpTokenizer
tokenizer = RegexpTokenizer('\s+', gaps=True)
# Create a string input
input_str = "I love to study Natural Language Processing in Python"
# Tokenize the input string
tokens = tokenizer.tokenize(input_str)
# Print tokens
print("Tokenized Output:")
print("=====")
print(tokens)
```

#### Output:

```
Tokenized Output:
=====
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

### C. Tokenization using NLTK

**Code:**

```
import nltk
from nltk.tokenize import word_tokenize

# Create a string input
input_str = "I love to study Natural Language Processing in Python"

# Tokenize the input string
tokens = word_tokenize(input_str)

# Print tokens
print("Tokenized Output:")
print("=====")
print(tokens)
```

**Output:**

```
Tokenized Output:
=====
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

### D. Tokenization using the spaCy library

**Code:**

```
import spacy
# Load the English blank model
nlp = spacy.blank("en")

# Create a string input
input_str = "I love to study Natural Language Processing in Python"

# Process the text with the NLP pipeline
doc = nlp(input_str)

# Extract tokens from the processed document
words = [token.text for token in doc]

# Print the list of tokens
print("Tokens:")
print("=====")
print(words)
```

**Output:**

```
Tokens:
=====
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

### E. Tokenization using Keras

**Code:**

```
!pip install keras
!pip install tensorflow
import keras
from keras.preprocessing.text import text_to_word_sequence
# Create a string input
input_str = "I love to study Natural Language Processing in Python"
# Tokenize the text
tokens = text_to_word_sequence(input_str)
# Print the tokens
print("Tokens:")
print("=====")
print(tokens)
```

**Output:**

```
Tokens:
=====
['i', 'love', 'to', 'study', 'natural', 'language', 'processing', 'in', 'python']
```

### F. Tokenization using Gensim

**Code:**

```
!pip install gensim
from gensim.utils import tokenize
# Create a string input
input_str = "I love to study Natural Language Processing in Python"
# Tokenize the text using gensim's tokenize function
tokens = list(tokenize(input_str))
# Print the tokens
print("Tokens:")
print("=====")
for token in tokens:
    print(token)
```

**Output:**

```
Tokens:
=====
I
love
to
study
Natural
Language
Processing
in
Python
```

## Practical 5

### Import NLP Libraries for Indian Languages and perform:

Note: Execute this practical in <https://colab.research.google.com/>

#### A. Word tokenization in Hindi.

##### Code:

```
!pip install torch==3.3.1
!pip install nltk
!pip install tornado==4.5.3

from nltk.nltk import setup

setup('hi') # 'hi' is the language code for Hindi

from nltk.nltk import tokenize

hindi_text = "प्राकृतिक भाषा सीखना बहुत रोमांचक है।"
tokens = tokenize(hindi_text, "hi")

print("Tokens:")
print(tokens)

print("Print Tokens:")
print(['प्राकृतिक', 'भाषा', 'सीखना', 'बहुत', 'रोमांचक', 'है', '।'])
```

##### Output:

```
Print Tokens:
['प्राकृतिक', 'भाषा', 'सीखना', 'बहुत', 'रोमांचक', 'है', '।']
```

#### B. Generate similar sentences from a given Hindi text input.

##### Code:

```
!pip install torch==3.3.1
!pip install nltk
!pip install tornado==4.5.3

from nltk.nltk import setup, get_similar_sentences

# Setup the Hindi language model
setup('hi')

# Get similar sentences to the given input in Hindi
input_text = 'मैं आज बहुत खुश हूँ'
output = get_similar_sentences(input_text, 5, 'hi')
```

```
# Print the output
print("Similar Sentences:")
for sentence in output:
    print(sentence)

print("Similar Sentences:\nमैं आज बहुत खुश हूँ\nमैं बहुत खुश हूँ\nमैं बहुत खुश हूँ\nमैं आज खुश हूँ\nमैं आज बहुत खुश हूँ")
```

**Output:**

```
Similar Sentences:
मैं आज बहुत खुश हूँ
मैं बहुत खुश हूँ
मैं बहुत खुश हूँ
मैं आज खुश हूँ
मैं आज बहुत खुश हूँ
```

**C. Identify the Indian language of a text.****Code:**

```
!pip install torch==3.3.1
!pip install nltk
!pip install tornado==4.5.3

from nltk.nltk import setup, identify_language

# Setting up Gujarati language model
setup('gu')

# Identify the language of the given text
language = identify_language('બીના કાપડિયા')

# Print the identified language
print("Identified Language:", language)

print("Identified Language: Gujarati")
```

**Output:**

```
Identified Language: Gujarati
```

## Practical 6

Illustrate part of speech tagging.

- a. Part of speech Tagging and chunking of user defined text.
- b. Named Entity recognition of user defined text.
- c. Named Entity recognition with diagram using NLTK corpus – treebank

**POS Tagging, chunking and NER:**

**A. sentence tokenization, word tokenization, Part of speech Tagging and chunking of user defined text.**

**Code:**

```
import nltk
from nltk import tokenize, tag, chunk
from tabulate import tabulate

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

# Input paragraph
para = "Hello! My name is Jack Reacher. Today you'll be learning NLTK."

# Sentence tokenization
sents = tokenize.sent_tokenize(para)

# Word tokenization and POS Tagging
tokenized_words = []
pos_tags = []
for sent in sents:
    words = tokenize.word_tokenize(sent)
    tokenized_words.append(words)
    pos_tags.append(tag.pos_tag(words))

# Chunking
chunked_trees = []
for tags in pos_tags:
    chunked_trees.append(chunk.ne_chunk(tags))

# Format outputs into tables
print("\nSentence Tokenization\n=====")
print(tabulate([[sent] for sent in sents], headers=['Sentences']))

print("\nWord Tokenization\n=====")
for i, words in enumerate(tokenized_words):
    print(f"Sentence {i + 1}:")
```

```

print(tabulate([words], headers=['Words']))

print("\nPOS Tagging\n=====")
for i, tags in enumerate(pos_tags):
    print(f"POS Tags for Sentence {i + 1}:")
    print(tabulate(tags, headers=['Word', 'POS Tag']))

print("\nChunking\n=====")
for i, tree in enumerate(chunked_trees):
    print(f"Chunking for Sentence {i + 1}:")
    print(tree)
    print()

```

**Output:**

```

Sentence Tokenization
=====
Sentences
-----
Hello!
My name is Jack Reacher.
Today you'll be learning NLTK.

Word Tokenization
=====
Sentence 1:
      Words
-----
Hello !
Sentence 2:
                                     Words
-----
My name is Jack Reacher .
Sentence 3:
                                     Words
-----
Today you 'll be learning NLTK .

```

```

POS Tagging
=====
POS Tags for Sentence 1:
Word      POS Tag
-----
Hello      NN
!          .

POS Tags for Sentence 2:
Word      POS Tag
-----
My         PRP$
name       NN
is         VBZ
Jack       NNP
Reacher    NNP
.          .

POS Tags for Sentence 3:
Word      POS Tag
-----
Today      NN
you         PRP
'll        MD
be         VB
learning   VBG
NLTK       NNP
.          .

```

```

Chunking
=====
Chunking for Sentence 1:
(S (GPE Hello/NN) !/.)

Chunking for Sentence 2:
(S My/PRP$ name/NN is/VBZ (PERSON Jack/NNP Reacher/NNP) ./.)

Chunking for Sentence 3:
(S
  Today/NN
  you/PRP
  'll/MD
  be/VB
  learning/VBG
  (ORGANIZATION NLTK/NNP)
  ./.)

```



**B. Named Entity recognition using user defined text.****Code:**

```

!pip install -U spacy
!python -m spacy download en_core_web_sm
import spacy
from tabulate import tabulate
# Load English tokenizer, tagger, parser, and NER
nlp = spacy.load("en_core_web_sm")
# Input text
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. "I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn't "
        "worth talking to," said Thrun, in an interview with Recode earlier "
        "this week.")
# Process the text with the loaded NLP pipeline
doc = nlp(text)
# Extract noun phrases and verbs
noun_phrases = [chunk.text for chunk in doc.noun_chunks]
verbs = [token.lemma_ for token in doc if token.pos_ == "VERB"]
# Format outputs into two columns
noun_phrases_table = [[f"Noun Phrase {i+1}", phrase] for i, phrase in enumerate(noun_phrases)]
verbs_table = [[f"Verb {i+1}", verb] for i, verb in enumerate(verbs)]
# Print outputs using tabulate for formatted display
print("Noun phrases and Verbs:")
print(tabulate(noun_phrases_table + verbs_table, headers=["Type", "Text"], tablefmt="pretty"))

```

**Output:**

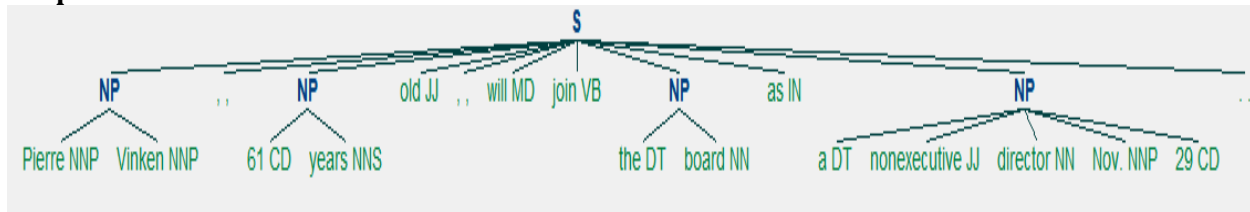
Noun phrases and Verbs:

Type	Text
Noun Phrase 1	Sebastian Thrun
Noun Phrase 2	self-driving cars
Noun Phrase 3	Google
Noun Phrase 4	few people
Noun Phrase 5	the company
Noun Phrase 6	him
Noun Phrase 7	I
Noun Phrase 8	you
Noun Phrase 9	very senior CEOs
Noun Phrase 10	major American car companies
Noun Phrase 11	my hand
Noun Phrase 12	I
Noun Phrase 13	Thrun
Noun Phrase 14	an interview
Noun Phrase 15	Recode
Verb 1	start
Verb 2	work
Verb 3	drive
Verb 4	take
Verb 5	tell
Verb 6	shake
Verb 7	turn
Verb 8	talk
Verb 9	say

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

#Note: It runs on Python IDLE

```
import nltk
nltk.download('treebank')
from nltk.corpus import treebank_chunk
treebank_chunk.tagged_sents()[0]
treebank_chunk.chunked_sents()[0]
treebank_chunk.chunked_sents()[0].draw()
```

**Output:**

## Practical 7

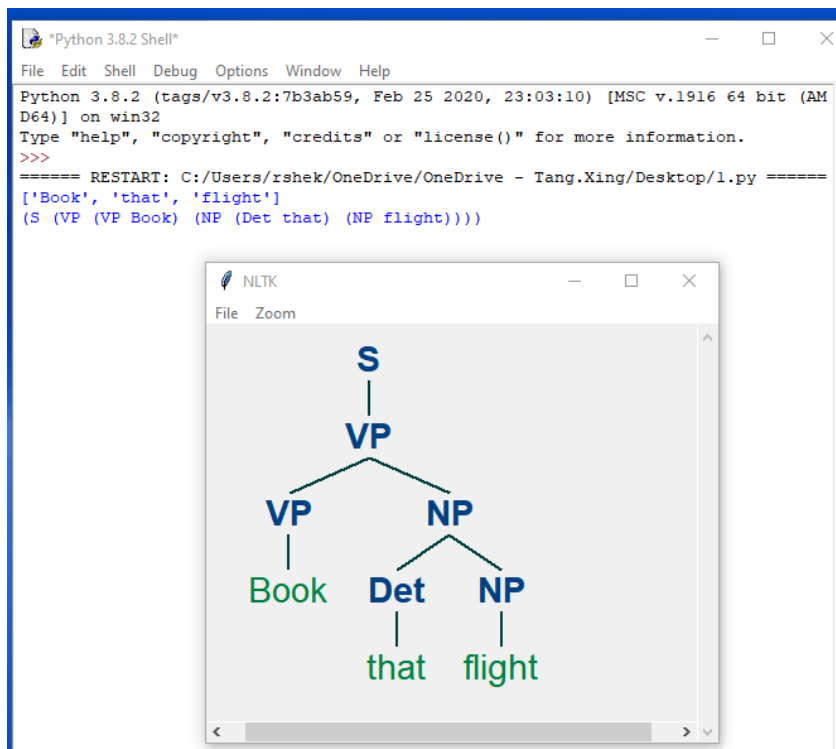
### Finite state automata.

#### A. Define grammar using nltk. Analyze a sentence using the same.

##### Code:

```
import nltk
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
S -> VP
VP -> VP NP
NP -> Det NP
Det -> 'that'
NP -> singular Noun
NP -> 'flight'
VP -> 'Book'
""")
sentence = "Book that flight"
for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
    print(all_tokens)
parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
    print(tree)
    tree.draw()
```

##### Output:



**B. Accept the input string with Regular expression of Finite Automaton: 101+.****Code:**

```
def FA(s):
    # If the length is less than 3, reject
    if len(s) < 3:
        return "Rejected (String too short)"

    # Check the first three characters
    if s[0] == '1' and s[1] == '0' and s[2] == '1':
        # Check remaining characters
        for i in range(3, len(s)):
            if s[i] != '1':
                return "Rejected (Not 101 followed by 1s)"
        # All characters after 2nd index are '1', accept
        return "Accepted (Matches 101+)"
    else:
        # First three characters don't match the pattern, reject
        return "Rejected (Doesn't start with 101)"

# Input strings to test the function
inputs = ['1', '10101', '101', '10111', '01010', '100', '', '10111101', '1011111']

# Loop through the inputs and call the FA function for each
for i in inputs:
    print(FA(i))
```

**Output:**

```
Rejected (String too short)
Rejected (Not 101 followed by 1s)
Accepted (Matches 101+)
Accepted (Matches 101+)
Rejected (Doesn't start with 101)
Rejected (Doesn't start with 101)
Rejected (String too short)
Rejected (Not 101 followed by 1s)
Accepted (Matches 101+)
```

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

```
def FA(s):
    # Check if the string is empty
    if not s:
        return "Rejected (Empty string)"

    # Check if the string contains only 'a' and 'b' characters
    for char in s:
```

```

    if char not in ('a', 'b'):
        return "Rejected (Contains characters other than 'a' and 'b')"
```

# Minimum length check (implicit in the regular expression)  
# No explicit check needed as 'bba' requires at least 3 characters

# Check if the string ends with 'bba'

```

if not s.endswith('bba'):
    return "Rejected (Doesn't end with 'bba')"
```

# All conditions met, accept the string

```

return "Accepted (Matches (a+b)*bba)"
```

# Input strings to test the function

```

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

**Output:**

```

Accepted (Matches (a+b)*bba)
Accepted (Matches (a+b)*bba)
Accepted (Matches (a+b)*bba)
Rejected (Doesn't end with 'bba')
Rejected (Doesn't end with 'bba')
Rejected (Doesn't end with 'bba')
Rejected (Empty string)
```

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

```

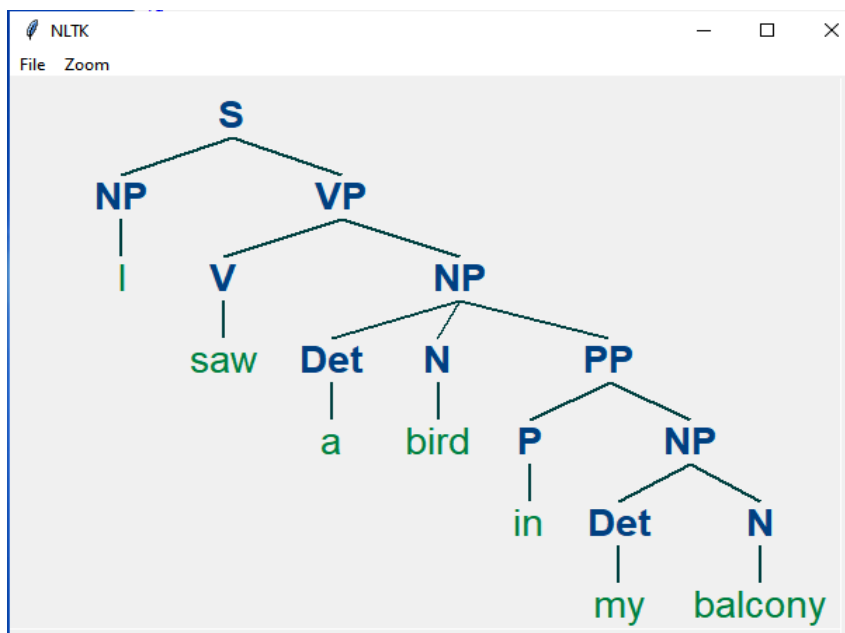
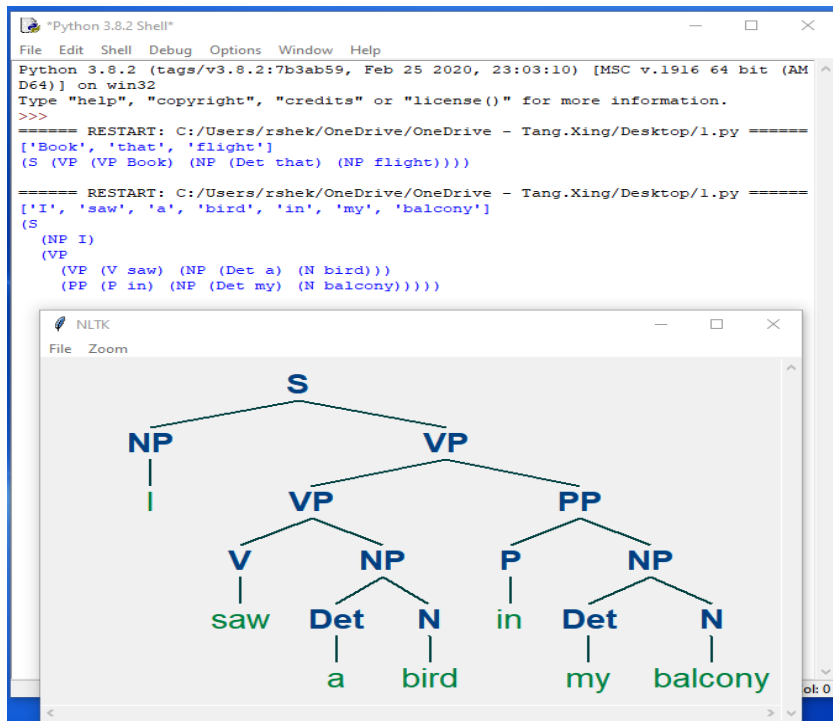
import nltk
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'a' | 'my'
N -> 'bird' | 'balcony'
V -> 'saw'
P -> 'in'
""")
sentence = "I saw a bird in my balcony"
for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
    print(all_tokens)
# all_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
```

```

parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
    print(tree)
    tree.draw()

```

**Output:**



## Practical 8

### Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer Study WordNetLemmatizer.

#### Code:

```
import nltk
from nltk.stem import PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer
from nltk.stem import WordNetLemmatizer
# Ensure you have the required NLTK data files
nltk.download('wordnet')
# PorterStemmer
word_stemmer = PorterStemmer()
print("PorterStemmer: 'writing' ->", word_stemmer.stem('writing'))
# LancasterStemmer
Lanc_stemmer = LancasterStemmer()
print("LancasterStemmer: 'writing' ->", Lanc_stemmer.stem('writing'))
# RegexpStemmer
Reg_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)
print("RegexpStemmer: 'writing' ->", Reg_stemmer.stem('writing'))

# SnowballStemmer
english_stemmer = SnowballStemmer('english')
print("SnowballStemmer: 'writing' ->", english_stemmer.stem('writing'))

# WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print("\nWordNetLemmatizer:")
print("word : \tlemma")
print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))

# 'a' denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos="a"))
```

#### Output:

```
PorterStemmer: 'writing' -> write
LancasterStemmer: 'writing' -> writ
RegexpStemmer: 'writing' -> writ
SnowballStemmer: 'writing' -> write
```

```
WordNetLemmatizer:
word : lemma
rocks : rock
corpora : corpus
better : good
```

## Practical 9

**Implement Naive Bayes classifier.**

**Code:**

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

# Step 1: Load the SMS spam dataset
sms_data = pd.read_csv("spam.csv", encoding='latin-1')

# Step 2: Text preprocessing
stemming = PorterStemmer()
corpus = []

for i in range(0, len(sms_data)):
    # Remove non-alphabetic characters and tokenize
    s1 = re.sub('[^a-zA-Z]', ' ', string=sms_data['v2'][i])
    s1 = s1.lower()
    s1 = s1.split()
    # Apply stemming and remove stopwords
    s1 = [stemming.stem(word) for word in s1 if word not in set(stopwords.words('english'))]
    s1 = ' '.join(s1)
    corpus.append(s1)

# Step 3: Vectorize the text data
countvectorizer = CountVectorizer()
x = countvectorizer.fit_transform(corpus).toarray()

# Step 4: Prepare target variable
y = sms_data['v1'].values

# Step 5: Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=y, random_state=2)

# Step 6: Train the Multinomial Naïve Bayes classifier
multinomialnb = MultinomialNB()
multinomialnb.fit(x_train, y_train)
```



```
# Step 7: Predict on test data
y_pred = multinomialnb.predict(x_test)

# Step 8: Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

**Output:**

```
Classification Report:
              precision    recall  f1-score   support

    ham         0.99         0.98         0.99         1448
    spam         0.89         0.92         0.91          224

 accuracy                   0.97         1672
 macro avg         0.94         0.95         0.95         1672
 weighted avg         0.98         0.97         0.98         1672

Accuracy Score: 0.9748803827751196
```

## Practical 10

### A. Speech Tagging:

#### i. Speech tagging using spacy

**Code:**

```
import spacy
from spacy import displacy

# Load the spaCy model
sp = spacy.load('en_core_web_sm')

# First sentence for processing
sen = sp(u"I like to play football. I hated it in my childhood though")

# Print the entire sentence
print("Original Sentence:")
print(sen.text)

# Analyze and print details of a specific word
word_index = 7
print("\nDetails of a specific word:")
print(f'Text: {sen[word_index].text}')
print(f'POS: {sen[word_index].pos_}')
print(f'Tag: {sen[word_index].tag_}')
print(f'Explanation: {spacy.explain(sen[word_index].tag_)})')

# Print POS tags for all words in the sentence
print("\nPOS tags for all words:")
for word in sen:
    print(f'{word.text:{12}} {word.pos_{:10}} {word.tag_{:8}} {spacy.explain(word.tag_)})')

# Analyzing another sentence
sen = sp(u'Can you google it?')
word = sen[2]
print("\nDetails of the word 'google':")
print(f'{word.text:{12}} {word.pos_{:10}} {word.tag_{:8}} {spacy.explain(word.tag_)})')

# Analyzing another word in a different context
sen = sp(u'Can you search it on google?')
word = sen[5]
print("\nDetails of the word 'google' in a different context:")
print(f'{word.text:{12}} {word.pos_{:10}} {word.tag_{:8}} {spacy.explain(word.tag_)})')

# Finding the number of POS tags in the first sentence
sen = sp(u"I like to play football. I hated it in my childhood though")
num_pos = sen.count_by(spacy.attrs.POS)
print("\nNumber of POS tags in the sentence:")
for k, v in sorted(num_pos.items()):
```

```
print(f'{k}. {sen.vocab[k].text:{8}}: {v}')
```

### # Visualizing Parts of Speech Tags

```
sen = sp(u"I like to play football. I hated it in my childhood though")
print("\nVisualizing Parts of Speech Tags (serving in the browser):")
displacy.serve(sen, style='dep', options={'distance': 120})
```

### Output:

Original Sentence:  
I like to play football. I hated it in my childhood though

Details of a specific word:  
Text: hated  
POS: VERB  
Tag: VBD  
Explanation: verb, past tense

POS tags for all words:

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

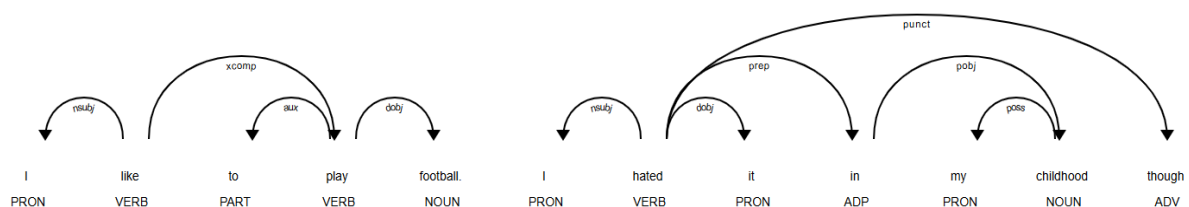
Details of the word 'google':  
google VERB VB verb, base form

Details of the word 'google' in a different context:  
google PROPN NNP noun, proper singular

Number of POS tags in the sentence:

85. ADP	: 1
86. ADV	: 1
92. NOUN	: 2
94. PART	: 1
95. PRON	: 4
97. PUNCT	: 1
100. VERB	: 3

Visualizing Parts of Speech Tags (serving in the browser):  
/usr/local/lib/python3.10/dist-packages/spacy/displacy/\_\_init\_\_.py:106: UserWarning: [W011] It looks like you're calling displacy.serve from within a Jupyter notebook or a similar environ  
warnings.warn(Warnings.W011)



## ii. Speech tagging using nltk

### Code:

```
import nltk
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer

# Download required NLTK data files
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('state_union')
# Create our training and testing data
```

```

train_text = state_union.raw("2005-GWBush.txt")
sample_text = state_union.raw("2006-GWBush.txt")

# Train the Punkt tokenizer
custom_sent_tokenizer = PunktSentenceTokenizer(train_text)

# Tokenize the sample text
tokenized = custom_sent_tokenizer.tokenize(sample_text)

def process_content():
    try:
        for i in tokenized[:2]: # Process only the first 2 sentences
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            print("\nTokenized Sentence:")
            print(i)
            print("\nPOS Tags:")
            print(tagged)
    except Exception as e:
        print(str(e))

# Process and display the content
process_content()

```

**Output:**

```

Tokenized Sentence:
PRESIDENT GEORGE W. BUSH'S ADDRESS BEFORE A JOINT SESSION OF THE CONGRESS ON THE STATE OF THE UNION

January 31, 2006

THE PRESIDENT: Thank you all.

POS Tags:
[('PRESIDENT', 'NNP'), ('GEORGE', 'NNP'), ('W.', 'NNP'), ('BUSH', 'NNP'), ('S', 'POS'), ('ADDRESS', 'NNP'),

Tokenized Sentence:
Mr. Speaker, Vice President Cheney, members of Congress, members of the Supreme Court and diplomatic corps,

POS Tags:
[('Mr.', 'NNP'), ('Speaker', 'NNP'), (',', ','), ('Vice', 'NNP'), ('President', 'NNP'), ('Cheney', 'NNP'), (

```

**B. Statistical parsing:****i. Usage of Give and Gave in the Penn Treebank sample****Code:**

```

import nltk

# Ensure required NLTK data is downloaded
nltk.download('treebank')

# Define a function to find specific VP nodes in the tree

```

```

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()))

# Define a function to create a sentence from tree leaves
def sent(t):
    return ' '.join(token for token in t.leaves() if token[0] not in '*-0')

# Define a function to print a node with specified width
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)

# Iterate through parsed sentences in the Penn Treebank and apply the functions
print("Finding and printing VP nodes that match the criteria:\n")
for tree in nltk.corpus.treebank.parsed_sents():
    for t in tree.subtrees(give):
        print_node(t, 72)

```

**Output:**

```

[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Package treebank is already up-to-date!
Finding and printing VP nodes that match the criteria:

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

```

**ii. probabilistic parser****Code:**

```
import nltk

# Define the PCFG grammar
grammar = nltk.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 the grammar
print("Grammar:")
print(grammar)

# Create a Viterbi parser using the grammar
viterbi_parser = nltk.ViterbiParser(grammar)

# Define a sentence to parse
sentence = "old men and women".split()

# Parse the sentence using the Viterbi parser
print("\nParsing Output:")
for tree in viterbi_parser.parse(sentence):
    print(tree)
```

**Output:**

```
Grammar:
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]

Parsing 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.**

Note: 1) Java should be installed.

Copy the Maltparser in the same folder and extract the same folder :

'C:/Users/schau/AppData/Local/Programs/Python/Python311/maltparser-1.7.2/maltparser-1.7.2'

And keep the same folder where maltparser is save.

'C:/Users/schau/AppData/Local/Programs/Python/Python311/engmalt.linear-1.7.mco'

**Code:**

```
from nltk.parse import malt
```

```
maltparser_dir = 'C:/Users/schau/AppData/Local/Programs/Python/Python311/maltparser-1.7.2/maltparser-1.7.2'
```

```
model_file = 'C:/Users/schau/AppData/Local/Programs/Python/Python311/engmalt.linear-1.7.mco'
```

```
mp = malt.MaltParser(maltparser_dir, model_file)
```


```
sentence = 'I saw a bird from my window.'
```

```
parsed_sentence = mp.parse_one(sentence.split())
```

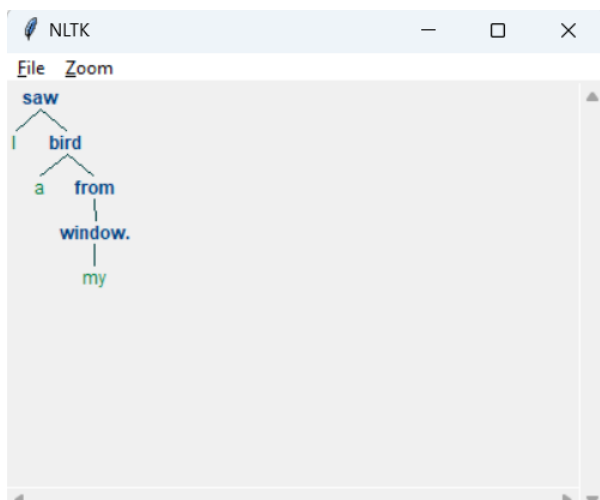
```
t = parsed_sentence.tree()
```

```
print(t)
```

```
t.draw()
```

**Output:**

(saw I (bird a (from (window. my))))



## Practical 11

### Multiword Expressions in NLP

#### Code:

```
import nltk
from nltk.tokenize import MWETokenizer
from nltk import sent_tokenize, word_tokenize

# Define the input text
s = "'Good cake cost Rs.1500/kg in Mumbai. Please buy me one of them.\n\nThanks.'"

# Define the MWE tokenizer with multi-word expressions
mwe = MWETokenizer([('New', 'York'), ('Hong', 'Kong')], separator='_')

# Tokenize the text into sentences and then tokenize each sentence
print("Tokenized Output:")
for sent in sent_tokenize(s):
    # Tokenize the sentence into words and then apply the MWE tokenizer
    tokens = word_tokenize(sent)
    mwe_tokens = mwe.tokenize(tokens)
    print(mwe_tokens)
```

#### Output:

```
Tokenized Output:
['Good', 'cake', 'cost', 'Rs.1500/kg', 'in', 'Mumbai', '.']
['Please', 'buy', 'me', 'one', 'of', 'them', '.']
['Thanks', '.']
```

### B. Normalized Web Distance and Word Similarity

#### Code:

```
!pip install textdistance
import numpy as np
import re
import textdistance
from sklearn.cluster import AgglomerativeClustering

# List of texts to cluster and normalize
texts = [
    'Reliance supermarket', 'Reliance hypermarket', 'Reliance', 'Reliance', 'Reliance downtown',
    'Relianc market',
    'Mumbai', 'Mumbai Hyper', 'Mumbai dxb', 'mumbai airport',
    'k.m trading', 'KM Trading', 'KM trade', 'K.M. Trading', 'KM.Trading'
]

# Function to normalize each text
def normalize(text):
    """ Keep only lower-cased text and numbers """
    return re.sub('[^a-z0-9]+', '', text.lower())
```



```

# Function to cluster texts based on Jaro-Winkler distance
def group_texts(texts, threshold=0.4):
    """ Replace each text with the representative of its cluster """
    normalized_texts = np.array([normalize(text) for text in texts])
    distances = 1 - np.array([
        [textdistance.jaro_winkler(one, another) for one in normalized_texts]
        for another in normalized_texts
    ])
    clustering = AgglomerativeClustering(
        distance_threshold=threshold, # this parameter needs to be tuned carefully
        affinity="precomputed", linkage="complete", n_clusters=None
    ).fit(distances)

    centers = dict()
    for cluster_id in set(clustering.labels_):
        index = clustering.labels_ == cluster_id
        centrality = distances[:, index][index].sum(axis=1)
        centers[cluster_id] = normalized_texts[index][centrality.argmax()]
    return [centers[label] for label in clustering.labels_]
# Clustering and printing the results with proper labeling
clustered_texts = group_texts(texts)
cluster_dict = {}
for idx, (text, clustered_text) in enumerate(zip(texts, clustered_texts)):
    cluster_id = clustered_texts.index(clustered_text)
    if cluster_id not in cluster_dict:
        cluster_dict[cluster_id] = []
    cluster_dict[cluster_id].append(text)
for cluster_id, cluster_texts in cluster_dict.items():
    print(f"Cluster {cluster_id + 1}:")
    for text in cluster_texts:
        print(f" - {text}")

```

**Output:**

```

Cluster 1:
- Reliance supermarket
- Reliance hypermarket
- Reliance
- Reliance
- Reliance downtown
- Relianc market
Cluster 7:
- Mumbai
- Mumbai Hyper
- Mumbai dxb
- mumbai airport
Cluster 11:
- k.m trading
- KM Trading
- KM trade
- K.M. Trading
- KM.Trading

```

### C. Word Sense Disambiguation

**Code:**

```
from nltk.corpus import wordnet as wn

def get_first_sense(word, pos=None):
    if pos:
        synsets = wn.synsets(word, pos=pos)
    else:
        synsets = wn.synsets(word)
    return synsets[0]

# Example usage for word sense disambiguation
word1 = 'bank'
best_synset1 = get_first_sense(word1)
print(f"Senses of '{word1}':")
print(f"{best_synset1.name(): {best_synset1.definition()}"")

word2 = 'set'
best_synset2 = get_first_sense(word2, 'n')
print(f"\nSenses of '{word2}' as a noun:")
print(f"{best_synset2.name(): {best_synset2.definition()}"")

best_synset3 = get_first_sense(word2, 'v')
print(f"\nSenses of '{word2}' as a verb:")
print(f"{best_synset3.name(): {best_synset3.definition()}"")
```

**Output:**

```
Senses of 'bank':
bank.n.01: sloping land (especially the slope beside a body of water)

Senses of 'set' as a noun:
set.n.01: a group of things of the same kind that belong together and are so used

Senses of 'set' as a verb:
put.v.01: put into a certain place or abstract location
```

---