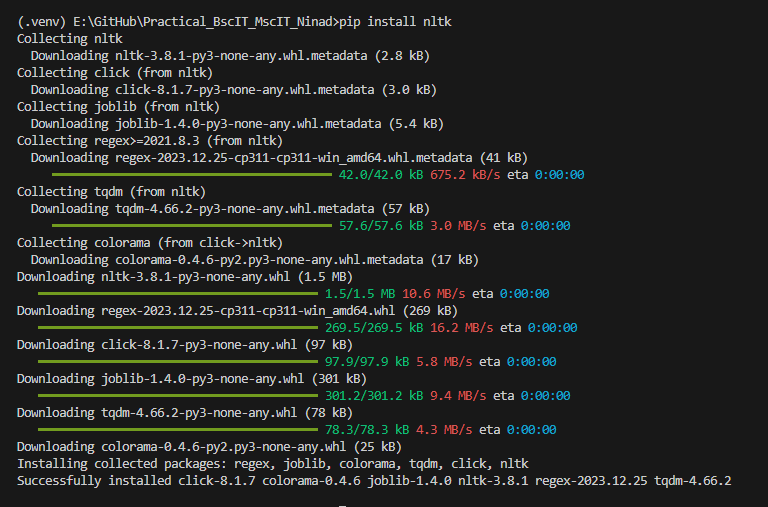
**Practical No: 1**

**AIM a): Study different libraries used for NLP in python.**

**AIM b): Install NLTK Package.**



**AIM c): Convert the given text into speech.**

**Description:**

**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 Ninad"

# 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:**

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**AIM d): Convert the speech to text**

**Description:**

**Code:**

#Aim: Convert audio file Speech to Text.

#Note: required to store the input file "NLP\_test.wav" in the current folder before running the program.

!pip install SpeechRecognition pydub

import speech\_recognition as sr

filename = "/content/NLP\_test.wav"

# initialize the recognizer

r = sr.Recognizer()

# open the file

with sr.AudioFile(filename) as source:

# listen for the data (load audio to memory)

audio\_data = r.record(source)

# recognize (convert from speech to text)

text = r.recognize\_google(audio\_data)

print(text)

A screenshot of a computer

Description automatically generated

**Practical No: 2**

a.

**Practical No: 3**

**Aim A): Study of Wordnet Dictionary with methods as synsets, definitions,**

Code:

# NLP 3A. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms

import nltk

from nltk.corpus import wordnet

nltk.download('wordnet')

synsets = wordnet.synsets("phone")

print("\*\*Word:\*\* phone")

print("  \* Synsets:")

for synset in synsets:

    word = synset.lemmas()[0].name()

    print(f"      - Word: {word}")

    print(f"        - Definition: {synset.definition()}")

    print(f"          - Examples: {synset.examples()}")

print("-"\*40)

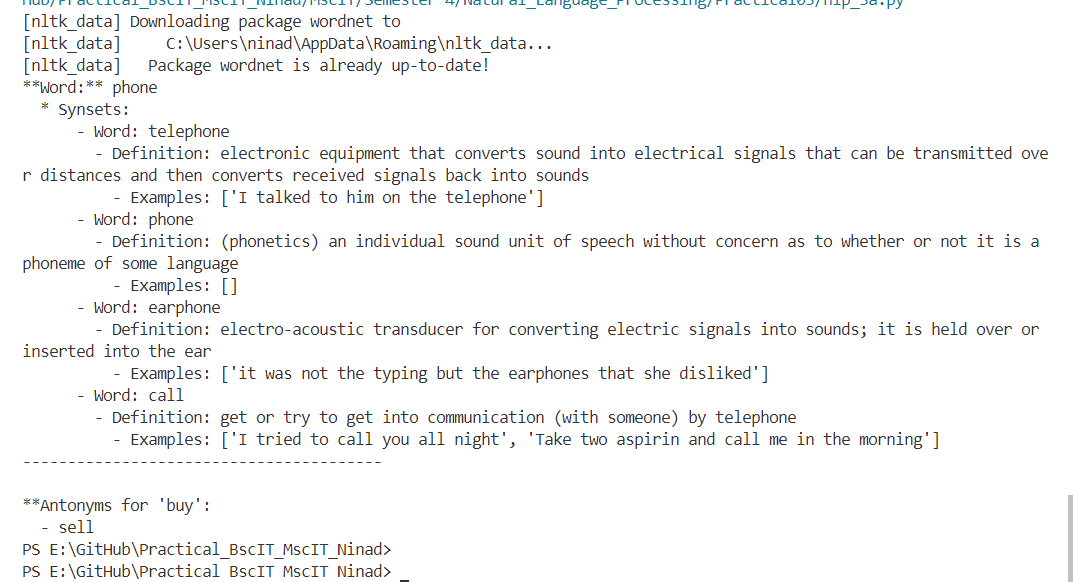
antonyms = wordnet.lemma('buy.v.01.buy').antonyms()

print("\n\*\*Antonyms for 'buy':")

for antonym in antonyms:

    print(f"  - {antonym.name()}")

OUTPUT:



**Aim B): Study lemmas, hyponyms, hypernyms.**

Code:  
  
# NLP 3B Study lemmas, hyponyms, hypernyms.

import nltk

from nltk.corpus import wordnet

nltk.download('wordnet')

print("\n\*\*Lemmas\*\*")

synsets = wordnet.synsets("computer")

print(" \* Synsets and Lemmas:")

for synset in synsets:

lemma\_names = [lemma.name() for lemma in synset.lemmas()]

print(f" - Synset: {synset} --> Lemmas: {lemma\_names}")

print("\n\*\*Hyponyms\*\*")

computer\_synset = wordnet.synset("computer.n.01")

hyponyms = computer\_synset.hyponyms()

print(" \* Hyponyms of 'computer.n.01':")

for synset in hyponyms:

lemma\_names = [lemma.name() for lemma in synset.lemmas()]

print(f" - Synset: {synset} --> Lemmas: {lemma\_names}")

print("\n\*\*Hypernyms\*\*")

vehicle\_synset = wordnet.synset("vehicle.n.01")

car\_synset = wordnet.synset("car.n.01")

lowest\_common\_hypernym = car\_synset.lowest\_common\_hypernyms(vehicle\_synset)

print(f" \* Lowest common hypernym of 'vehicle' and 'car': {lowest\_common\_hypernym[0]}")

**OUTPUT:**A screenshot of a computer code

Description automatically generated

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

**Code:**

# NLP 3C. Write a program using python to find synonym and antonym of word "active" using Wordnet.

import nltk

from nltk.corpus import wordnet

nltk.download('omw-1.4')

def get\_synonyms\_antonyms(word):

synonyms = []

antonyms = []

for syn in wordnet.synsets(word):

for lemma in syn.lemmas():

synonyms.append(lemma.name())

if lemma.antonyms():

antonyms.append(lemma.antonyms()[0].name())

return set(synonyms), set(antonyms)

def main():

word = input("Enter the word:-")

synonyms, antonyms = get\_synonyms\_antonyms(word)

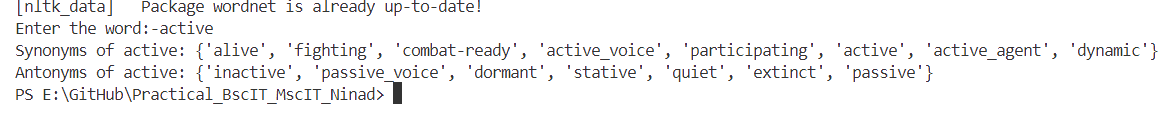
print("Synonyms of", word + ":", synonyms)

print("Antonyms of", word + ":", antonyms)

if \_\_name\_\_ == "\_\_main\_\_":

nltk.download('wordnet')

main()

**Output:**

**Aim D): Write a program using python to find synonym and antonym of word "active" using Wordnet.**

**Code:**

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

from nltk.corpus import wordnet as wn

def compare\_nouns(noun1, noun2):

# Get synsets for each noun

synsets1 = wn.synsets(noun1, pos=wn.NOUN)

synsets2 = wn.synsets(noun2, pos=wn.NOUN)

if not synsets1 or not synsets2:

return "Unable to compare. Make sure both nouns are valid."

max\_wup\_similarity = 0

max\_path\_similarity = 0

for synset1 in synsets1:

for synset2 in synsets2:

# Calculate Wu-Palmer Similarity

wup\_similarity = synset1.wup\_similarity(synset2)

if wup\_similarity is not None and wup\_similarity > max\_wup\_similarity:

max\_wup\_similarity = wup\_similarity

# Calculate Path Similarity

path\_similarity = synset1.path\_similarity(synset2)

if path\_similarity is not None and path\_similarity > max\_path\_similarity:

max\_path\_similarity = path\_similarity

return max\_wup\_similarity, max\_path\_similarity

if \_\_name\_\_ == "\_\_main\_\_":

noun1 = input("Enter the first noun: ")

noun2 = input("Enter the second noun: ")

wup\_similarity\_score, path\_similarity\_score = compare\_nouns(noun1, noun2)

print(f"The Wu-Palmer Similarity between '{noun1}' and '{noun2}' is: {wup\_similarity\_score}")

print(f"The Path Similarity between '{noun1}' and '{noun2}' is: {path\_similarity\_score}")

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 3E\_1): Using nltk, add or remove stop words in NLTK's Default stop word list**

**Code:**

# NLP 3E\_a: Using nltk, add or remove stop words in NLTK's Default stop word list

# Import necessary libraries

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Download stopwords (if not already downloaded)

nltk.download('punkt')

nltk.download('stopwords')

print("-"\*40)

# Sample text

text = "Ninad likes to play Chess, however he is not too good with the football."

print("Given Text:- ",text)

# Remove stop words from text

text\_tokens = word\_tokenize(text)

stop\_words = stopwords.words('english')

tokens\_without\_sw = [word for word in text\_tokens if word not in stop\_words]

print("Tokens without stop words:", tokens\_without\_sw)

# Add custom stop word ('not')

custom\_stop\_words = stop\_words + ['not']

tokens\_without\_sw = [word for word in text\_tokens if word not in custom\_stop\_words]

print("\nTokens without 'not' (custom stop word):", tokens\_without\_sw)

**Output:**

A computer screen with text

Description automatically generated

**Aim 3E\_2): Using Gensim, add or remove stop words in Default Gensim stop words List.**

**Code:**

#pip install scipy==1.12

import gensim

from gensim.parsing.preprocessing import remove\_stopwords, STOPWORDS

text = "Ninad likes to play Chess, however he is not too good with the football."

filtered\_sentence = remove\_stopwords(text)

print("-"\*30)

print("Original sentence:", text)

print("-"\*30)

print("Stop words removed:", filtered\_sentence)

print("-"\*30)

all\_stopwords = STOPWORDS.union(set(['likes', 'play']))

text\_tokens = text.split()

tokens\_without\_sw = [word for word in text\_tokens if word not in all\_stopwords]

print("Original sentence (tokens):", text\_tokens)

print("-"\*30)

print("Stop words 'likes' and 'play' added:", tokens\_without\_sw)

print("-"\*30)

all\_stopwords = STOPWORDS.difference({"not"})

tokens\_without\_sw = [word for word in text.split() if word not in all\_stopwords]

print("Original sentence (tokens):", text.split())

print("-"\*30)

print("Stop word 'not' removed:", tokens\_without\_sw)

print("-"\*30)

**Output:**

A screenshot of a computer screen

Description automatically generated

**Aim 3E\_3): Using SpaCy, add or remove Stop Words in Default SpaCy stop words List.**

**Code:**

# NLP 3E\_c: Using SpaCy, add or remove Stop Words in Default SpaCy stop words List.

#python -m spacy download en\_core\_web\_sm

import spacy

import nltk

from nltk.tokenize import word\_tokenize

print("NLP 3E 3 Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List")

sp = spacy.load("en\_core\_web\_sm")

# Get default stop words from spaCy

all\_stopwords = sp.Defaults.stop\_words

text = "Ninad likes to play Chess, however he is not too good with the football."

text\_tokens = word\_tokenize(text)

all\_stopwords.add("play")

tokens\_without\_sw = [word for word in text\_tokens if word not in all\_stopwords]

print("Original sentence (tokens):", text\_tokens)

print("Stop word 'play' added:", tokens\_without\_sw)

all\_stopwords.remove("not")

tokens\_without\_sw = [word for word in text\_tokens if word not in all\_stopwords]

print("Original sentence (tokens):", text\_tokens)

print("Stop word 'not' removed:", tokens\_without\_sw)

**Output:**

A screen shot of a computer screen

Description automatically generated

**Practical No: 4**

**Text Tokenization**

**Aim 4A): Tokenization using Python’s split() function.**

**Code:**

# 4A. Tokenization using Python’s split() function

# Sample text to tokenize

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Tokenizing the text using split()

tokens = text.split()

# Printing the tokens

print("="\*60)

print("4A. Tokenization using Python’s split() function")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 4B): Tokenization using Regular Expression (RegEx).**

**Code:**

# 4b. Tokenization using Regular Expressions (RegEx)

import re

# Sample text to tokenize

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Define the regex pattern for tokenization (splitting by whitespace)

pattern = r'\s+'

# Tokenizing the text using re.split()

tokens = re.split(pattern, text)

# Printing the tokens

print("="\*60)

print("4b. Tokenization using Regular Expressions (RegEx)")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A white background with black text

Description automatically generated

**Aim 4C): Tokenization using NLTK.**

**Code:**

#4c. Tokenization using NLTK

import nltk

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

# Sample text to tokenize

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Tokenizing the text using NLTK's word\_tokenize()

tokens = word\_tokenize(text)

# Printing the tokens

print("="\*60)

print("4c. Tokenization using NLTK")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 4D): Tokenization using spaCy library.**

**Code:**

#4d. Tokenization using the spaCy library

import spacy

# Load the English language model

nlp = spacy.blank("en")

# Text to be tokenized

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Process the text with SpaCy

doc = nlp(text)

# Extract tokens

tokens = [token.text for token in doc]

# Print tokens

print("="\*60)

print("4d. Tokenization using the spaCy library")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A close up of a computer screen

Description automatically generated

**Aim 4E): Tokenization using Keras.**

**Code:**

#4e. Tokenization using Keras [COLAB]

import keras

from tensorflow.keras.preprocessing.text import text\_to\_word\_sequence #works on colab

# Create a string input

str = "Hello ! My name is Ninad Karlekar I live in mumbai"

# tokenizing the text

tokens = text\_to\_word\_sequence(str)

print("="\*60)

print("4e. Tokenization using Keras")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

####

# to run on local IDE(jupyter) use tenserflow version 2.13.0

# pip uninstall tensorflow     ## to uninstall latest version if installed

# pip install tensorflow==2.13.0

####

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 4F): Tokenization using Gensim.**

**Code:**

# 4f. Tokenization using Gensim

#pip install gensim

from gensim.utils import tokenize

# Create a string input

str = "Hello ! My name is Ninad Karlekar I live in mumbai"

# tokenizing the text

# Tokenizing the text

tokenized\_words = list(tokenize(str))

# Printing each tokenized word separately

print("="\*60)

print("4f. Tokenization using Gensim")

print("-"\*10)

print("Tokens:", tokenized\_words)

print("="\*60)

**Output:**

A computer code with black text

Description automatically generated

**Practical No: 5**

**Aim 5A): Word tokenization in Hindi**

**Code:**

**Output:**

**Aim 5B): Generate similar sentences from a given Hindi text input.**

**Code:**

**Output:**

**Aim 5C): Identify the Indian language from the given text..**

**Code:**

**Output:**

**Practical No: 6**

**Aim 6A): Part of speech Tagging and chunking of user defined text.**

**Code:**

# 6. Illustrate part of speech tagging.

## a) sentence tokenization, word tokenization, Part of speech Tagging and chunking of user defined text.

import nltk

from nltk import tokenize

from nltk import tag

from nltk import chunk

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

# Paragraph to be tokenized

para = "Hello! My name is Ninad Karlekar. Today you'll be learning NLTK."

# Sentence tokenization

sents = tokenize.sent\_tokenize(para)

print("\nsentence tokenization\n===================\n", sents)

# Word tokenization

print("\nword tokenization\n===================\n")

for index in range(len(sents)):

    words = tokenize.word\_tokenize(sents[index])

    print(words)

# POS Tagging

tagged\_words = []

for index in range(len(sents)):

    tagged\_words.append(tag.pos\_tag(words))

print("\nPOS Tagging\n===========\n", tagged\_words)

# Chunking

tree = []

for index in range(len(sents)):

    tree.append(chunk.ne\_chunk(tagged\_words[index]))

print("\nchunking\n========\n", tree)

**Output:**

A black text on a white background

Description automatically generated A black text on a white background

Description automatically generated A close-up of a text

Description automatically generated A text on a white background

Description automatically generated

**Aim 6B): Named Entity recognition of user defined text.**

**Code:**

# !pip install -U spacy

# !python -m spacy download en\_core\_web\_sm

import spacy

nlp = spacy.load("en\_core\_web\_sm")

# Process whole documents

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."

)

doc = nlp(text)

# Analyse syntax

print("Noun phrases:", [chunk.text for chunk in doc.noun\_chunks])

print("Verbs:", [token.lemma\_ for token in doc if token.pos\_ == "VERB"])

**Output:**

A close-up of a computer code

Description automatically generated

**Aim 6C): Named Entity recognition with diagram using NLTK corpus - treebank**

**Code:**

# 6C: Named Entity recognition with diagram using NLTK corpus - treebank

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()

# Note: It runs on Python IDLE, VScode

**Output:**

A diagram of a person's beard

Description automatically generated

**Practical No: 7**

**Aim 7A): Define grammar using nltk. Analyse a sentence using the same.**

**Code:**

**Output:**

1. **Aim 7B): Accept the input string with Regular expression of FA: 101+**

**Code:**

**Output:**

1. **Aim 7C): Accept the input string with Regular expression of FA: (a+b)\*bba**

**Code:**

**Output:**

1. **Aim 7D): Implementation of Deductive Chart Parsing using context free grammar and a given sentence.**

**Code:**

**Output:**

**Practical No: 8**

**Aim 8A): Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer.**

**Code:**

# PorterStemmer

import nltk

from nltk.stem import PorterStemmer

word\_stemmer = PorterStemmer()

print("Output of PorterStemmer:-")

print(word\_stemmer.stem('Ninad is running'))

print("\*"\*50)

# LancasterStemmer

import nltk

from nltk.stem import LancasterStemmer

Lanc\_stemmer = LancasterStemmer()

print("Output of LancasterStemmer:-")

print(Lanc\_stemmer.stem('jumping'))

print("\*"\*50)

#RegexpStemmer

import nltk

from nltk.stem import RegexpStemmer

Reg\_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)

print("Output of RegexpStemmer:-")

print(Reg\_stemmer.stem('writing'))

print("\*"\*50)

# SnowballStemmer

import nltk

from nltk.stem import SnowballStemmer

english\_stemmer = SnowballStemmer('english')

print("Output of SnowballStemmer:-")

print(english\_stemmer.stem ('writing'))

print("\*"\*50)

**Output:**

A screenshot of a computer program

Description automatically generated

**Aim 8B): Study WordNet Lemmatizer**

**Code:**

# WordNetLemmatizer

print("Output of WordNetLemmatizer:-")

import nltk

nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("word :\tlemma")

print("rocks :", lemmatizer.lemmatize("books"))

print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"

print("worse :", lemmatizer.lemmatize("worse", pos ="a"))

print("\*"\*50)

**Output:**

A screenshot of a computer program

Description automatically generated

**Practical No: 9**

**Aim 9): Implement Naive Bayes classifier.**

**Code:**

# 9. Implement Naive Bayes classifier

# pip install pandas

# pip install sklearn

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, confusion\_matrix, accuracy\_score

nltk.download('stopwords')

sms\_data = pd.read\_csv("MscIT\\Semester 4\\Natural\_Language\_Processing\\Practical09\\spam.csv", encoding='latin-1')

stemming = PorterStemmer()

corpus = []

for i in range(len(sms\_data)):

s1 = re.sub('[^a-zA-Z]', ' ', sms\_data['v2'][i])

s1 = s1.lower()

s1 = s1.split()

s1 = [stemming.stem(word) for word in s1 if word not in set(stopwords.words('english'))]

s1 = ' '.join(s1)

corpus.append(s1)

countvectorizer = CountVectorizer()

x = countvectorizer.fit\_transform(corpus).toarray()

print(x)

y = sms\_data['v1'].values

print(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, stratify=y, random\_state=2)

multinomialnb = MultinomialNB()

multinomialnb.fit(x\_train, y\_train)

y\_pred = multinomialnb.predict(x\_test)

print(y\_pred)

print(classification\_report(y\_test, y\_pred))

print("accuracy\_score: ", accuracy\_score(y\_test, y\_pred))

**Output:**A screenshot of a computer

Description automatically generated

**Practical No: 10**

**Aim 10A): Part of speech Tagging and chunking of user defined text.**

**Code:**

**Output:**