**PRACTICAL: 1**

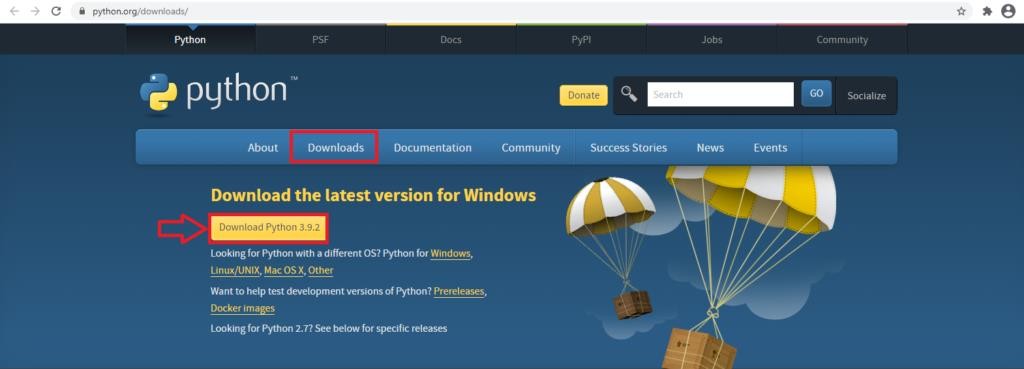
**1A] Install NLTK**

**Python 3.9.2 Installation on Windows**

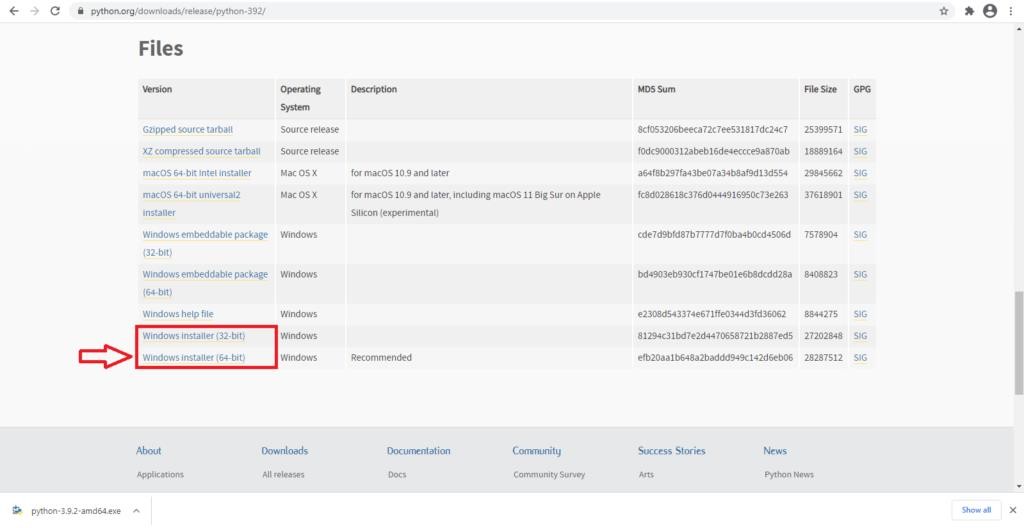
**InstallNLTK**

**Python3.9.2InstallationonWindows**

Step 1) **Go to link** https:/[/www.p](http://www.python.org/downloads/)y[thon.org/downloads/,](http://www.python.org/downloads/)**and select the latestversionforwindows.**

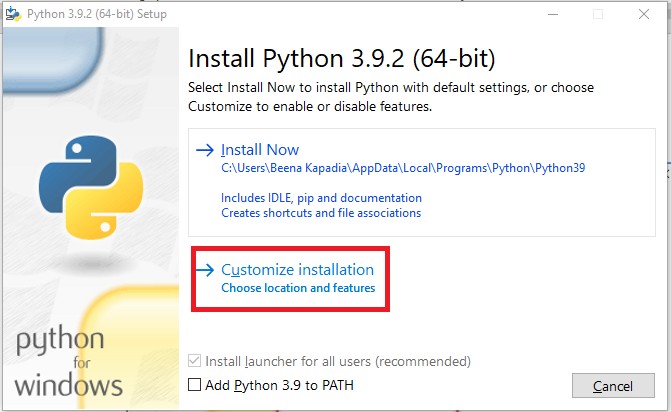


**Note**: If you don't want to download the latest version, you can visit thedownloadtab and seeallreleases.

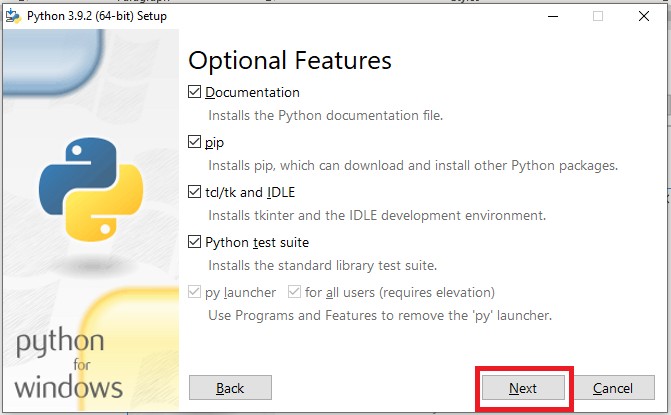


**Step2)**ClickontheWindowsinstaller(64bit)

**Step3)**SelectCustomizeInstallation

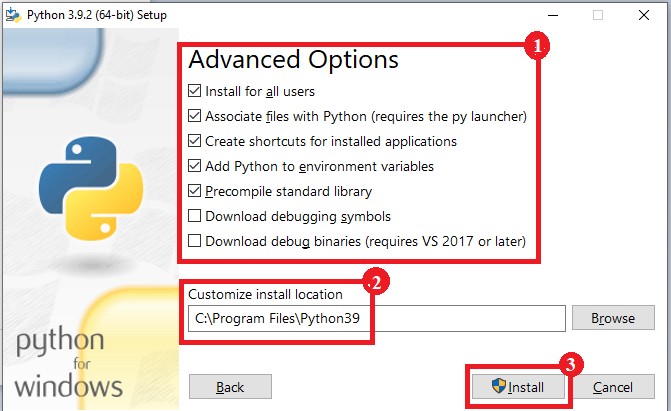


**Step4)**ClickNEXT



**Step5)** Innextscreen

* 1. Selecttheadvanced options
  2. Give a Custom install location. Keep the default folder as c:\Programfiles\Python39
  3. ClickInstall



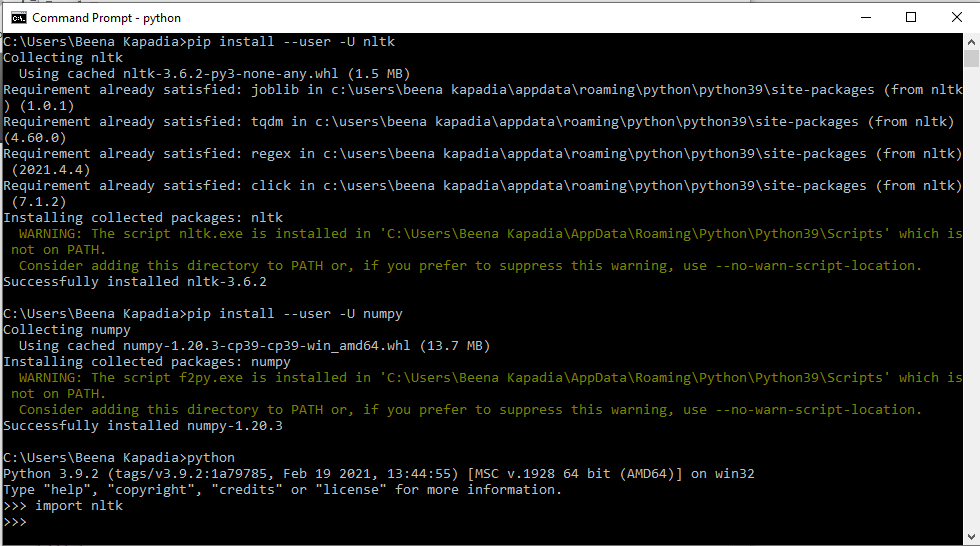
**Step6)**ClickClosebuttononceinstallisdone.

**Step 7) open** command prompt window and run the following commands:C:\Users\BeenaKapadia>pip install--upgradepip

C:\Users\Beena Kapadia> pip install --user -U nltkC:\Users\BeenaKapadia>>pipinstall --user-U numpyC:\Users\BeenaKapadia>python

>>>importnltk

>>>



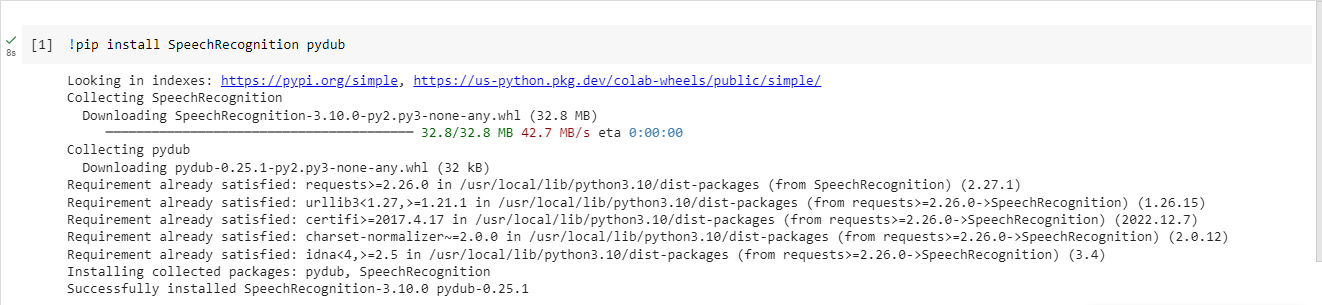
**1B] Convert the given text to speech.**







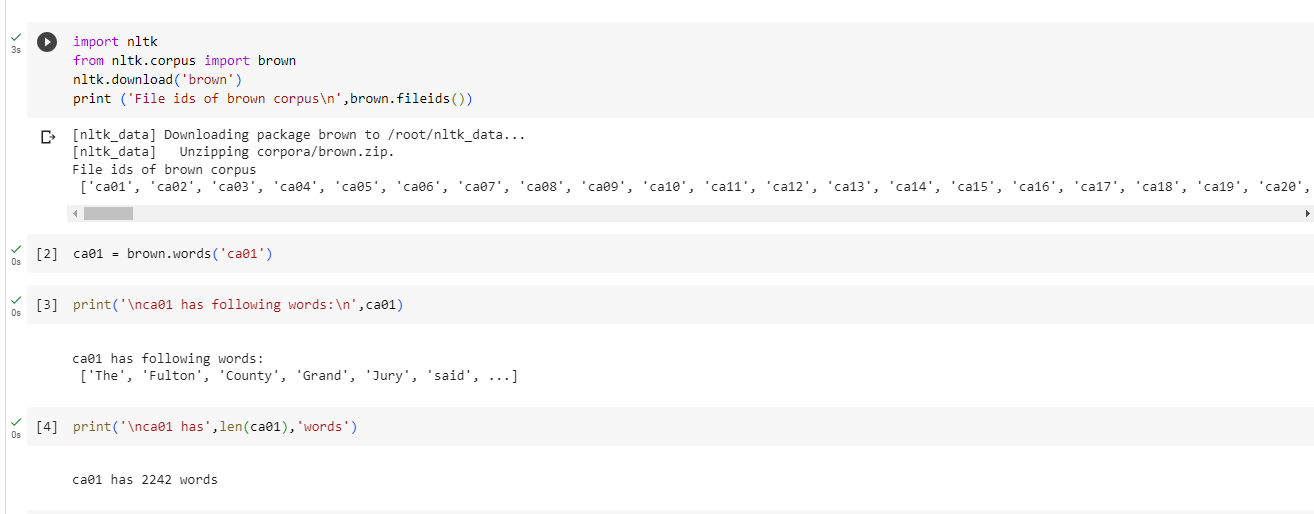
**1C] Convert audio file Speech to Text.**

****

****

**PRACTICAL: 2**

**2A] Study of various Corpus – Brown, Inaugural, Reuters, udhrwith various method likefilelds, raw,words, sents, categories.**

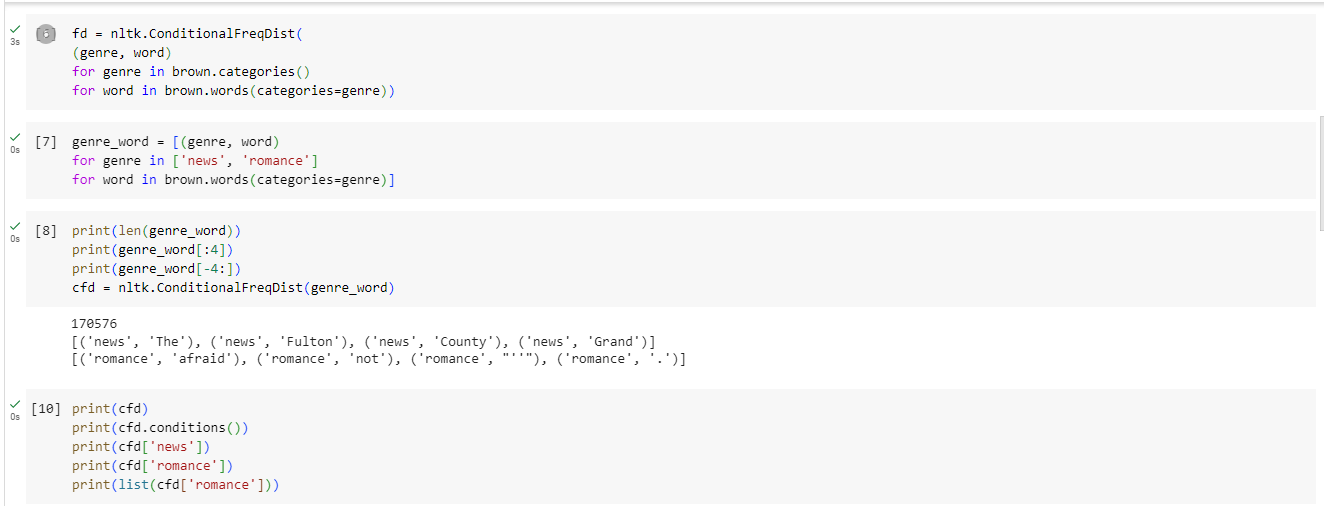
****

****

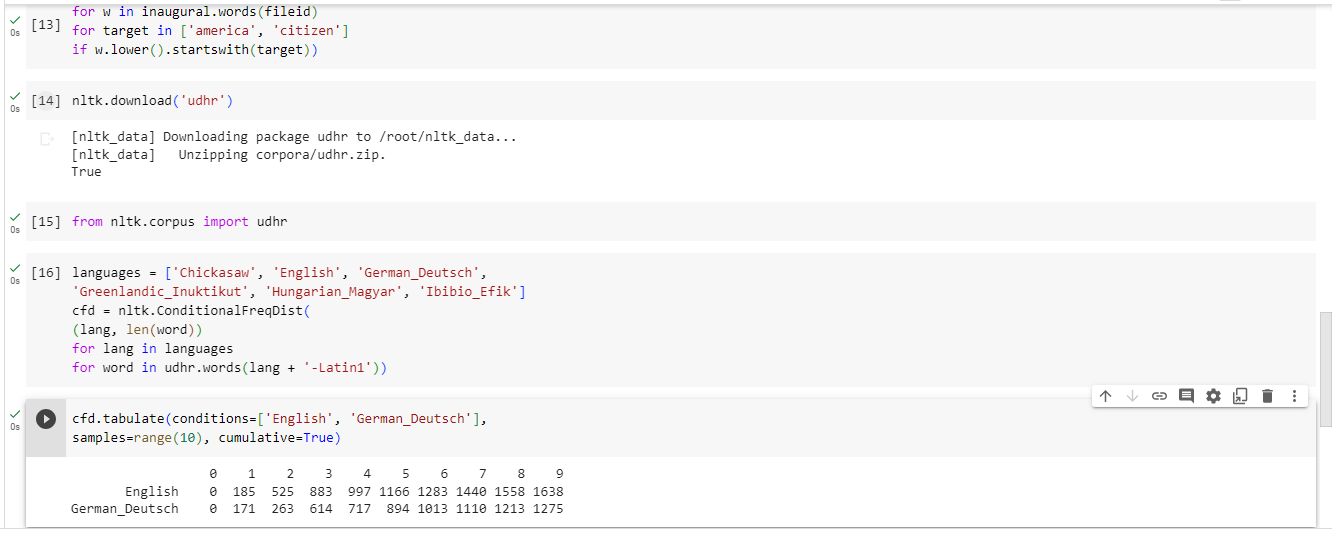
****

# 2C] Study Conditional frequency distributions

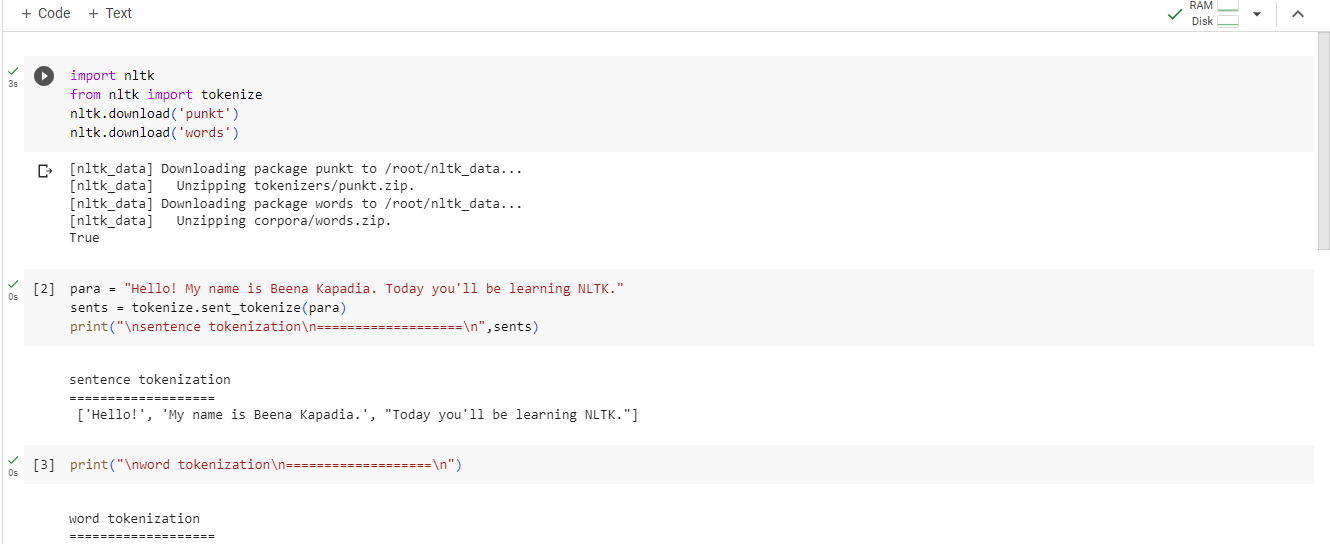
****

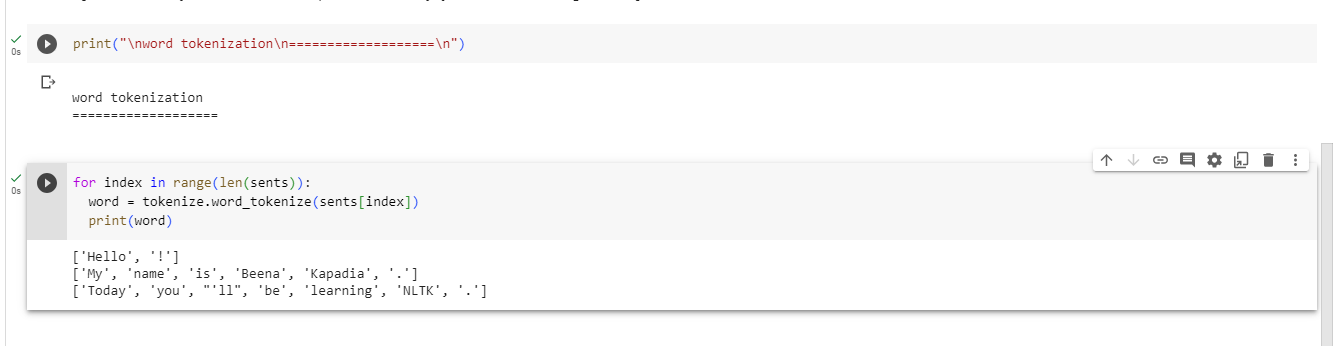
****

****

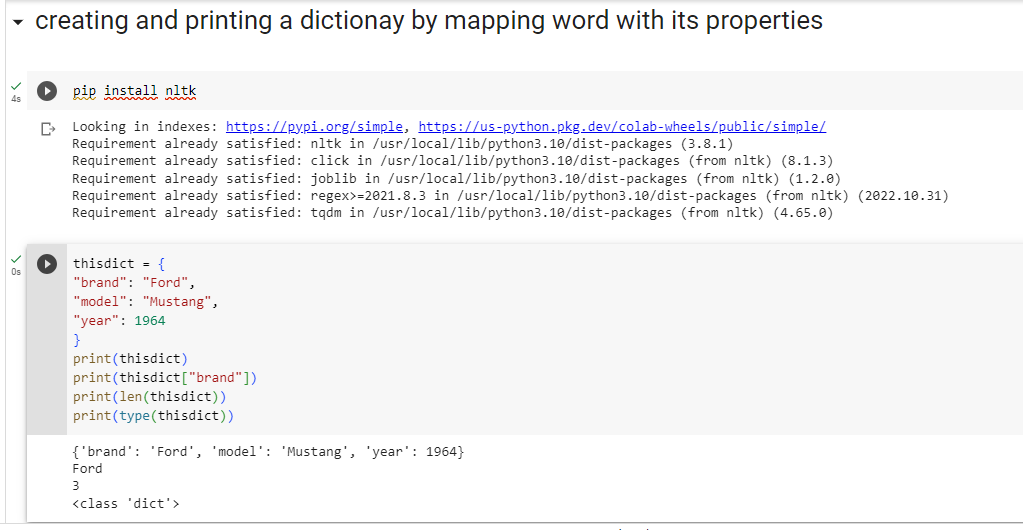
****

**2D] Study of tagged corpora with methods like tagged\_sents, tagged\_words.**

****

****

**2F] Map Words to Properties Using Python Dictionaries.**

****

**PRACTICAL: 3**

**3A] Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms**



**3B] Study lemmas, hyponyms, hypernyms.**



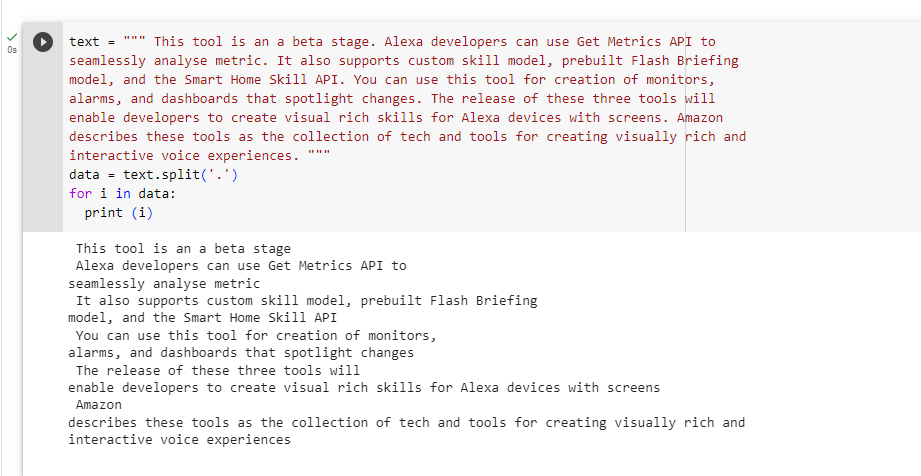


**3C] Write a program using python to find synonym and antonym of word "active"usingWordnet.**

****

**PRACTICAL: 4**

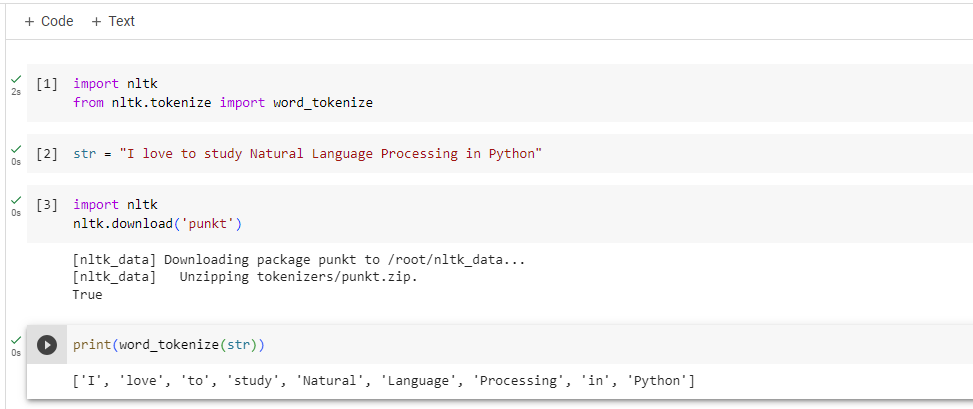
**4A] Tokenization using Python’s split() function**



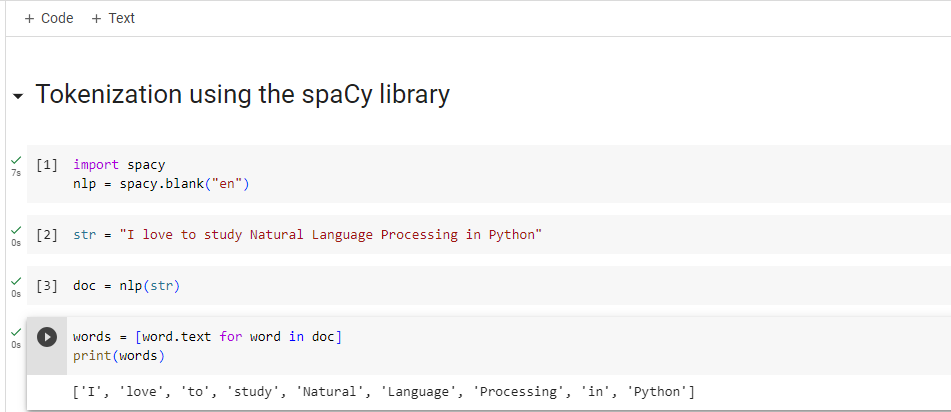
**4B] Tokenization using Regular Expressions (RegEx)**



**4C] Tokenization using NLTK**



**4D] Tokenization using the spaCy library**



**PRACTICAL: 5**

**Import NLP Libraries for Indian Languages and perform**

**5A] word tokenization in Hindi**

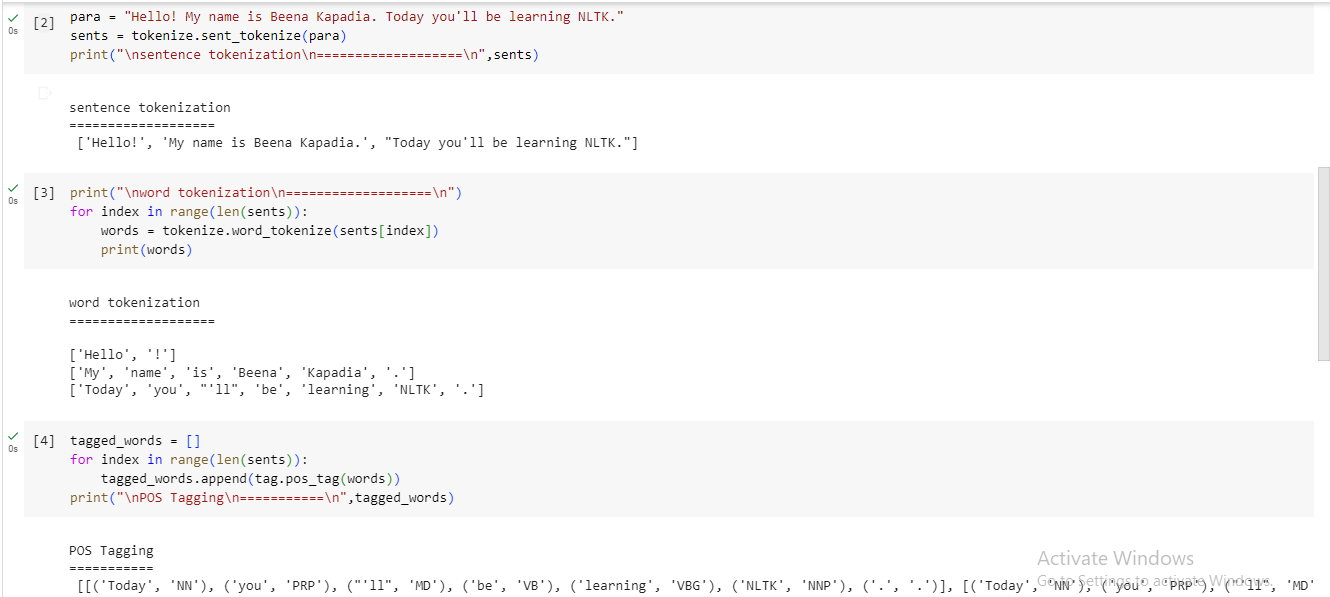


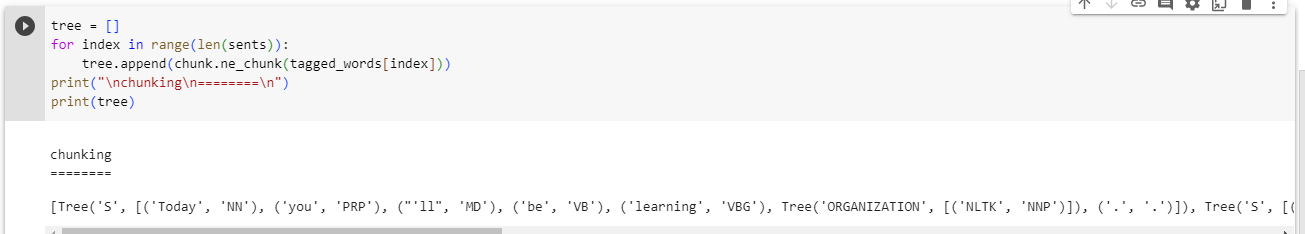
**PRACTICAL: 6**

**Illustrate part of speech tagging.**

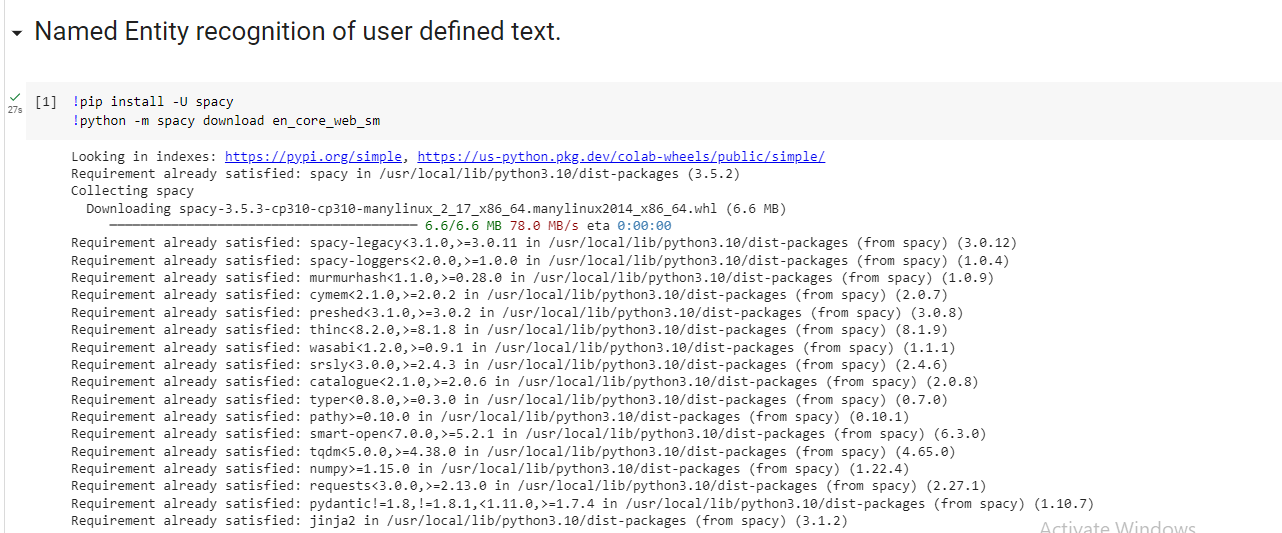
**6A]Part of speech Tagging and chunking of user defined text.**







**6B] Named Entity recognition of user defined text.**





**6C] Named Entity recognition with diagram using NLTK corpus – Treebank**

importnltk

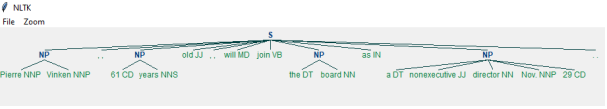
nltk.download('treebank')

fromnltk.corpus import treebank\_chunk

treebank\_chunk.tagged\_sents()[0]

treebank\_chunk.chunked\_sents()[0]

treebank\_chunk.chunked\_sents()[0].draw()



**PRACTICAL: 7**

**Finite state automata**

**7A]Define grammar using nltk.Analyze a sentence using the same.**

importnltk

fromnltk 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)

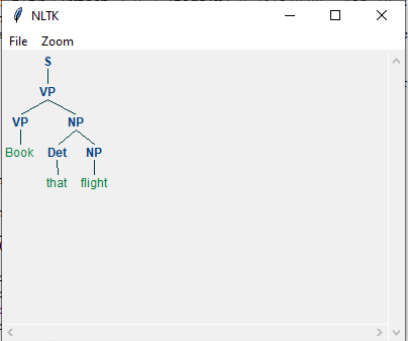
print(all\_tokens)

parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens):

print(tree)

tree.draw()



**7B] Accept the input string with Regular expression of Finite Automaton: 101+.**

def FA(s):

# if the length is less than 3, it can't be accepted, so end the process

iflen(s) < 3:

return "Rejected"

# first three characters are fixed, checking them using index

if s[0] == '1':

if s[1] == '0':

if s[2] == '1':

# After index 2, only "1" can appear, so break the process if any other character is detected

for i in range(3, len(s)):

if s[i] != '1':

return "Rejected"

return "Accepted" # if all nested ifs are true

return "Rejected" # else of 3rd if

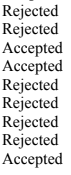
return "Rejected" # else of 2nd if

return "Rejected" # else of 1st if

inputs = ['1', '10101', '101', '10111', '01010', '100', '', '10111101', '1011111']

for i in inputs:

print(FA(i))



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

def FA(s):

size = 0

# scan complete string and make sure that it contains only 'a' & 'b'

for i in s:

if i == 'a' or i == 'b':

size += 1

else:

return "Rejected"

# After checking that it contains only 'a' & 'b'

# check its length; it should be at least 3

if size >= 3:

# check the last 3 elements

if s[size-3] == 'b':

if s[size-2] == 'b':

if s[size-1] == 'a':

return "Accepted" # if all 3 ifs are true

return "Rejected" # else of 3rd if

return "Rejected" # else of 2nd if

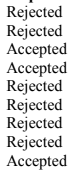
return "Rejected" # else of 1st if

return "Rejected"

inputs = ['bba', 'ababbba', 'abba', 'abb', 'baba', 'bbb', '']

for i in inputs:

print(FA(i))



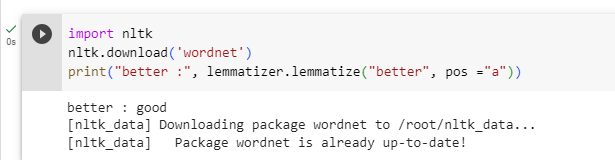
**PRACTICAL: 8**

**Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer**

**Study WordNetLemmatizer**

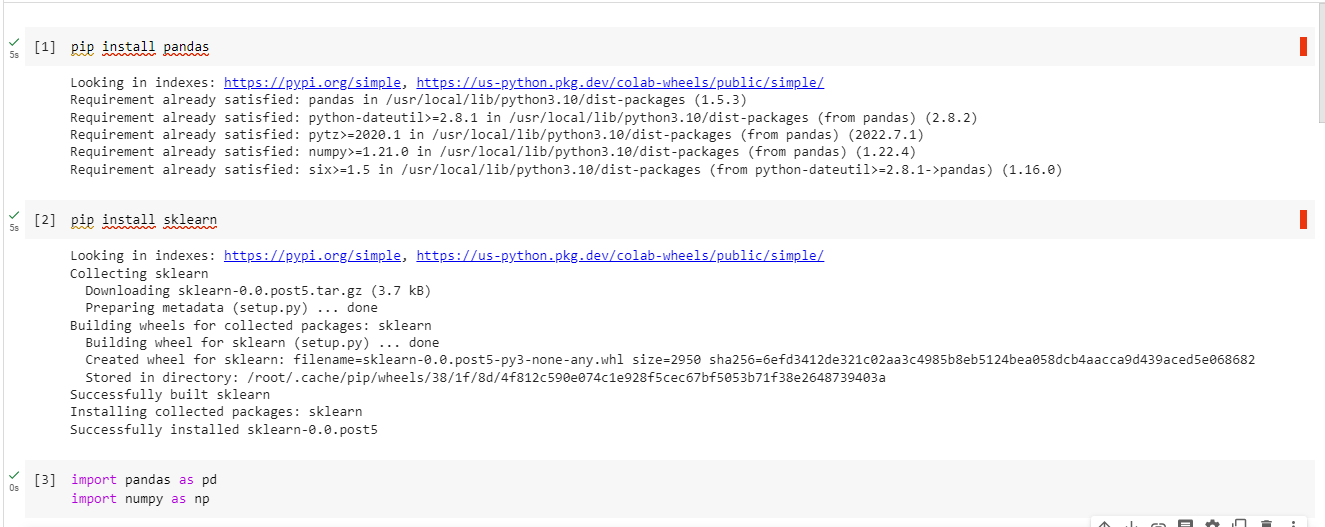


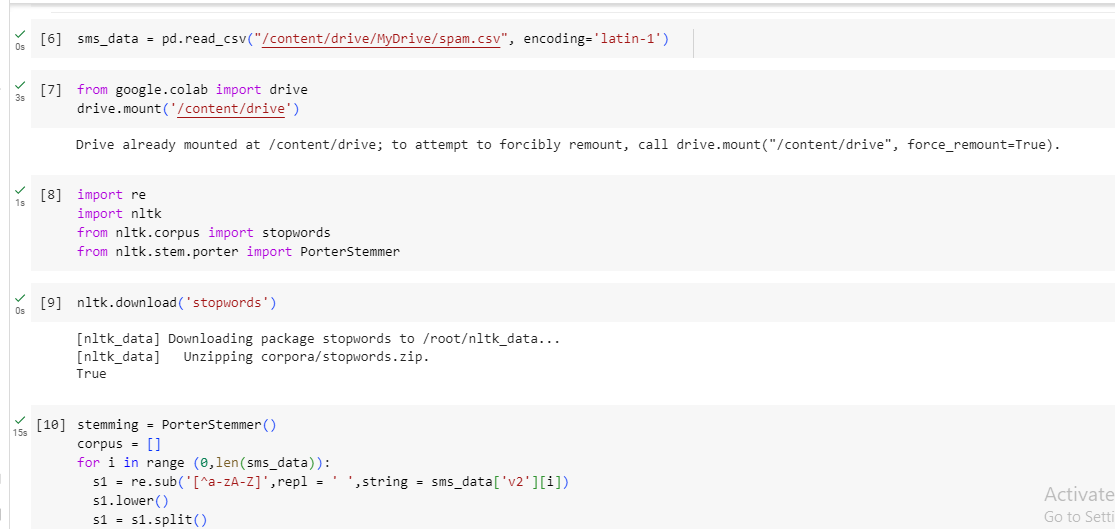


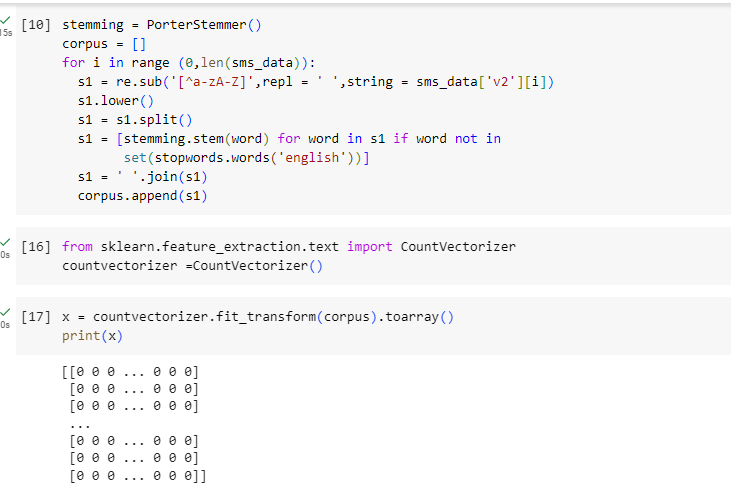


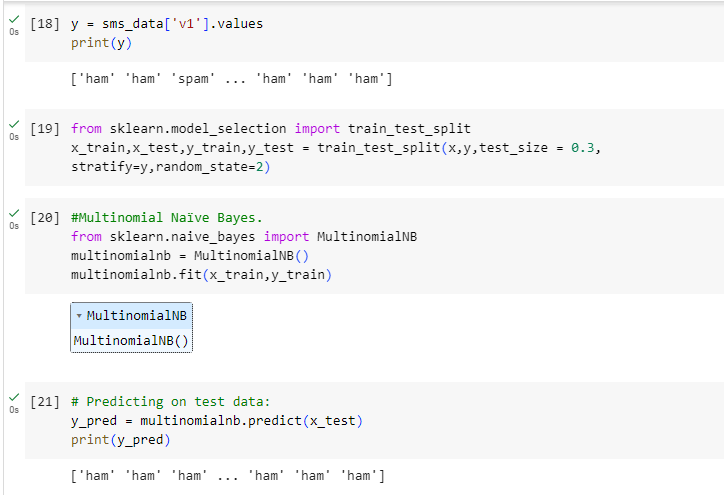
**PRACTICAL: 9**

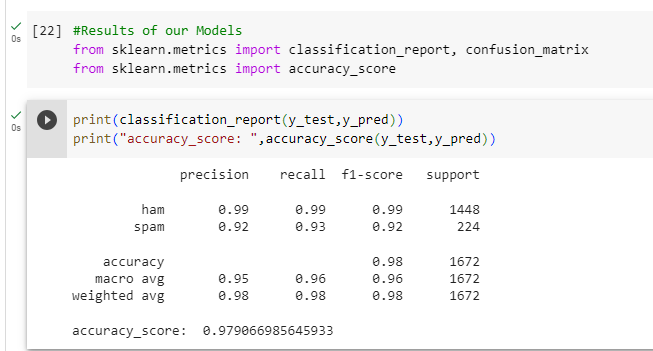
**Implement Naive Bayes classifier**







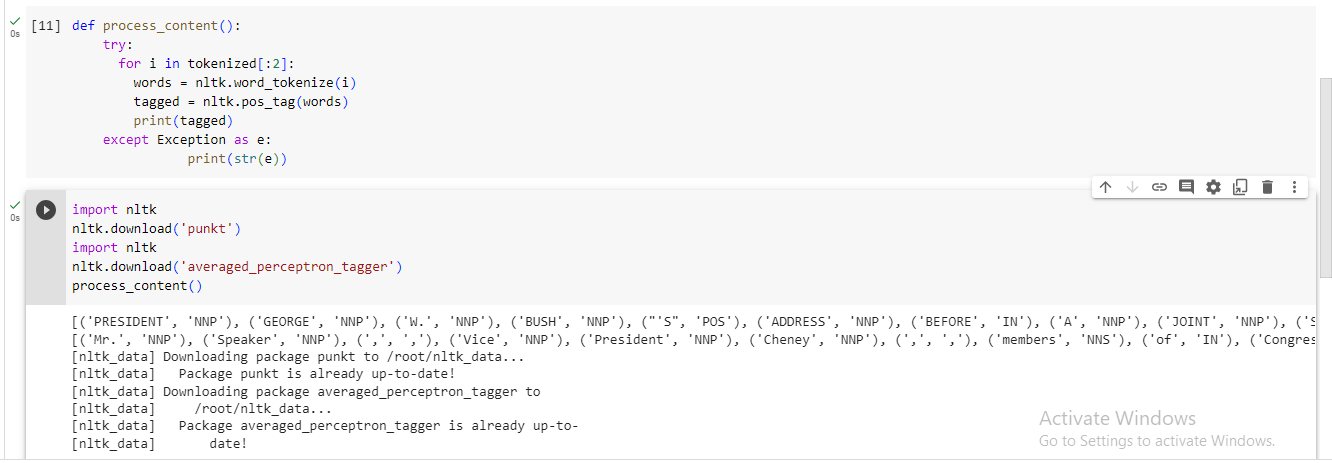




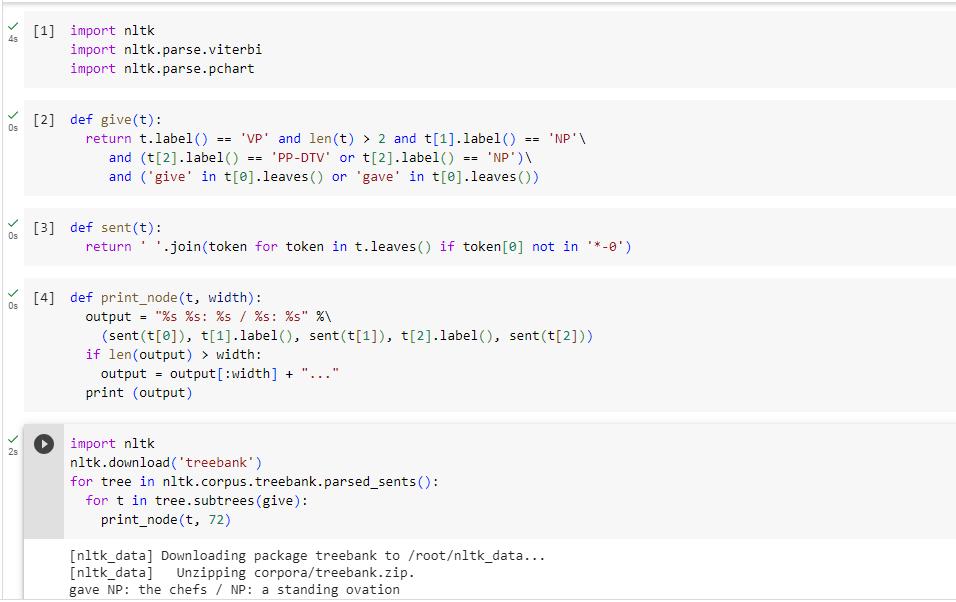
**PRACTICAL: 10**

**10AII] Speech tagging using nktl**

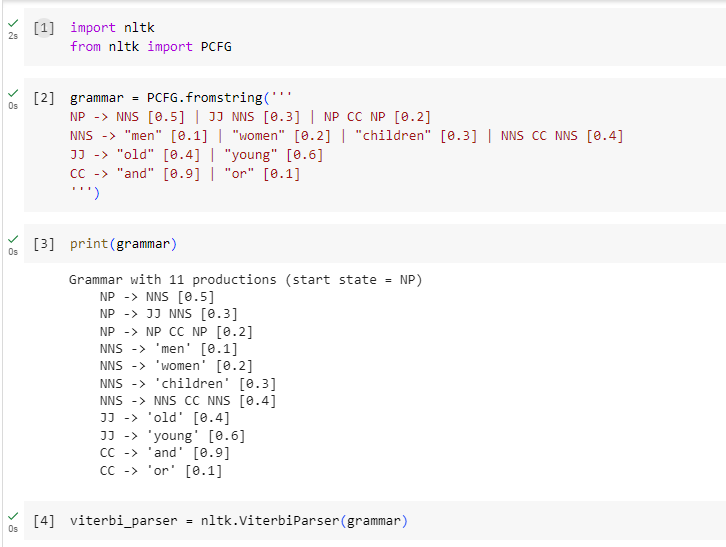




**10BI] Usage of Give and Gave in the Penn Treebank sample**



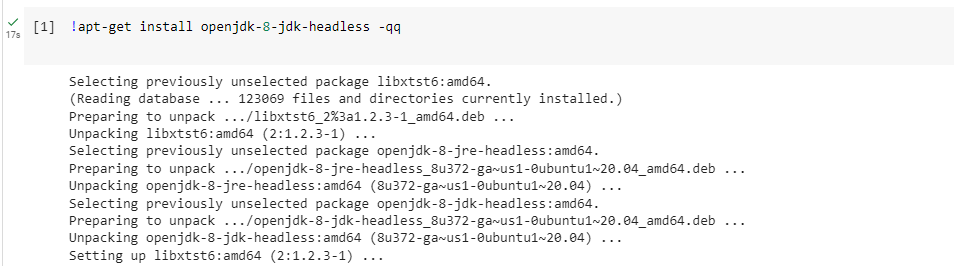
**10BII] probabilistic parser**

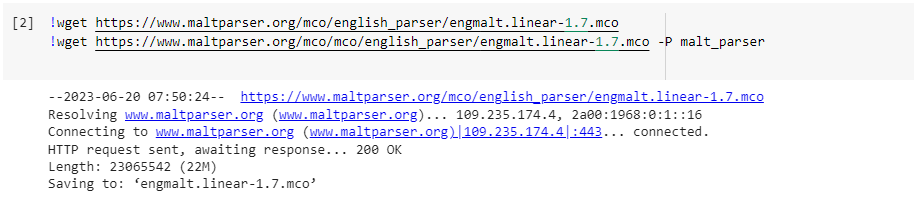




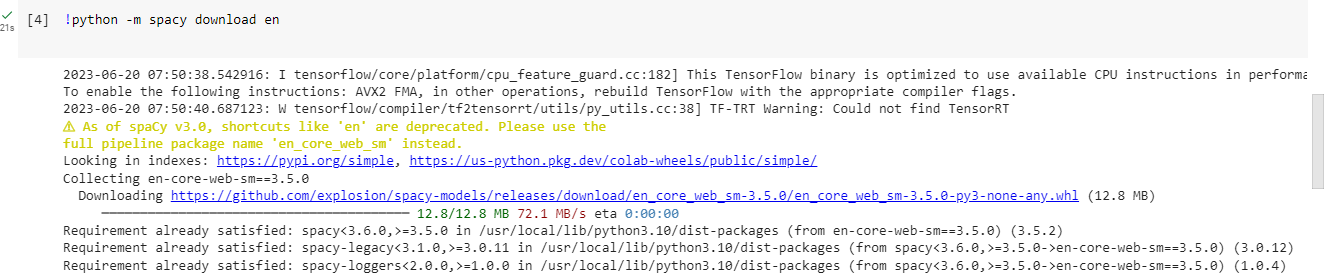
**10C] Malt parsing:**

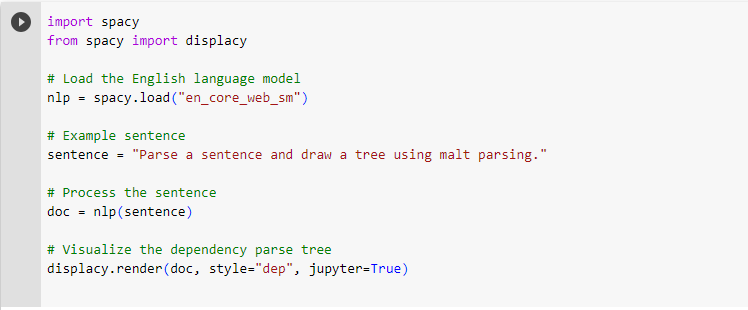
**Parse a sentence and draw a tree using malt parsing.**

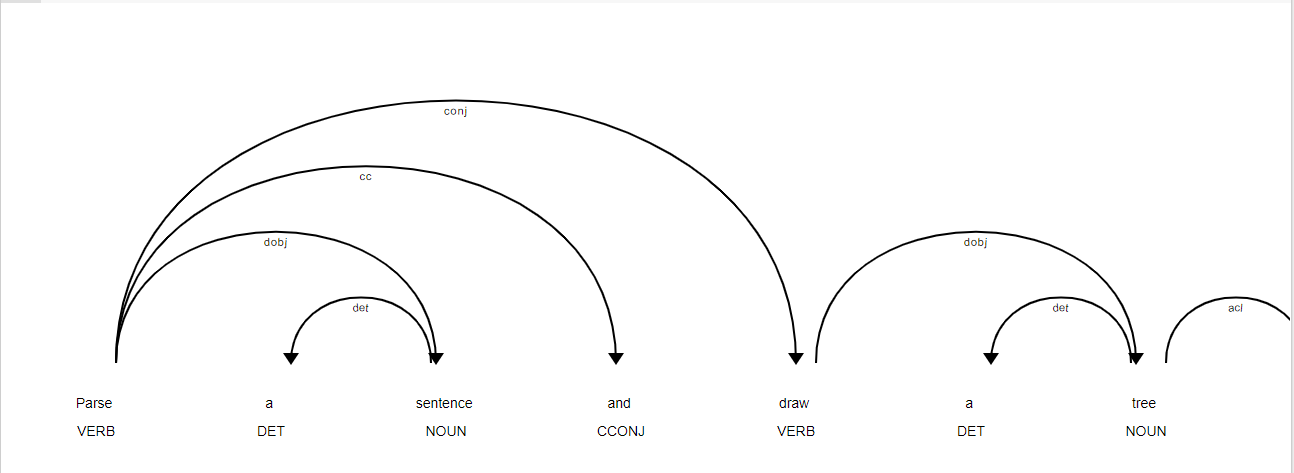






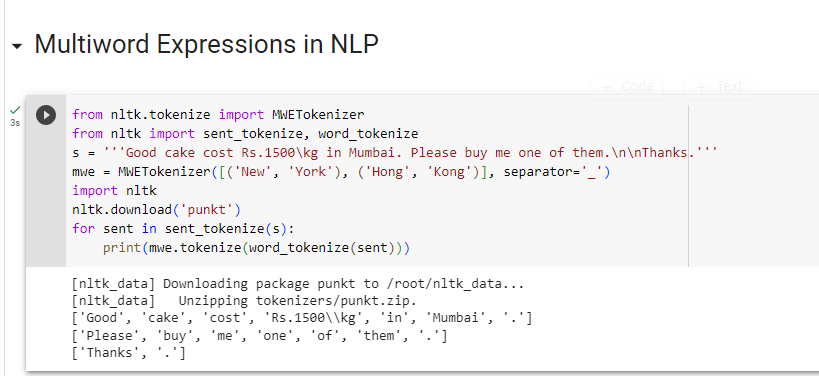






**PRACTICAL: 11**

**11A] Multiword Expressions in NLP**



**11B] Normalized Web Distance and Word Similarity**

importnumpy as np

import re

importtextdistance

importsklearn

fromsklearn.cluster import AgglomerativeClustering

texts = [

'Reliance supermarket', 'Reliance hypermarket', 'Reliance', 'Reliance', 'Reliance',

'downtown', 'Reliance market',

'Mumbai', 'Mumbai Hyper', 'Mumbai dxb', 'mumbai airport',

'k.m trading', 'KM Trading', 'KM trade', 'K.M. Trading', 'KM.Trading'

]

def normalize(text):

""" Keep only lower-cased text and numbers"""

returnre.sub('[^a-z0-9]+', ' ', text.lower())

defgroup\_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,

affinity="precomputed", linkage="complete", n\_clusters=None

).fit(distances)

centers = dict()

forcluster\_id in set(clustering.labels\_):

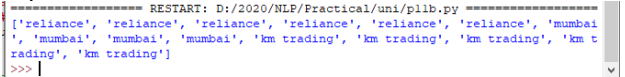
index = clustering.labels\_ == cluster\_id

centrality = distances[:, index][index].sum(axis=1)

centers[cluster\_id] = normalized\_texts[index][centrality.argmin()]

return [centers[i] for i in clustering.labels\_]

print(group\_texts(texts))



**11C] Word Sense Disambiguation**

