# Import NLTK.

import nltk nltk.download()

from nltk.corpus

import brown brown.words()



# Convert the given speech to text.

Code:

import speech\_recognition as sr

filename = "D://MSc.IT//sem\_4//NLP//NLP\_practical//NLP\_prac\_codes//female.wav"

#initialize the recognizer r = sr.Recognizer()

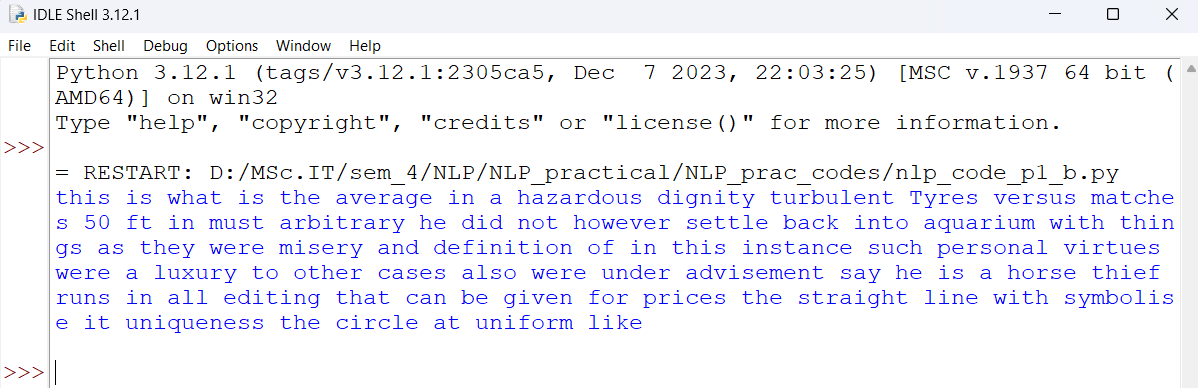
#open the file

with sr.AudioFile(filename) as source:

#listen for the data (load audio into memory) audio\_data = r.record(source)

#recognize (convert from speech into text) text = r.recognize\_google(audio\_data) print(text)

Output:



# Convert the given text to speech.

Code:

#Import the gTTS module for text to speech from gtts import gTTS

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

#It is a text value that we want to convert to audio

text\_val = 'You are studying Natural Language Processing. Better Speak in English'

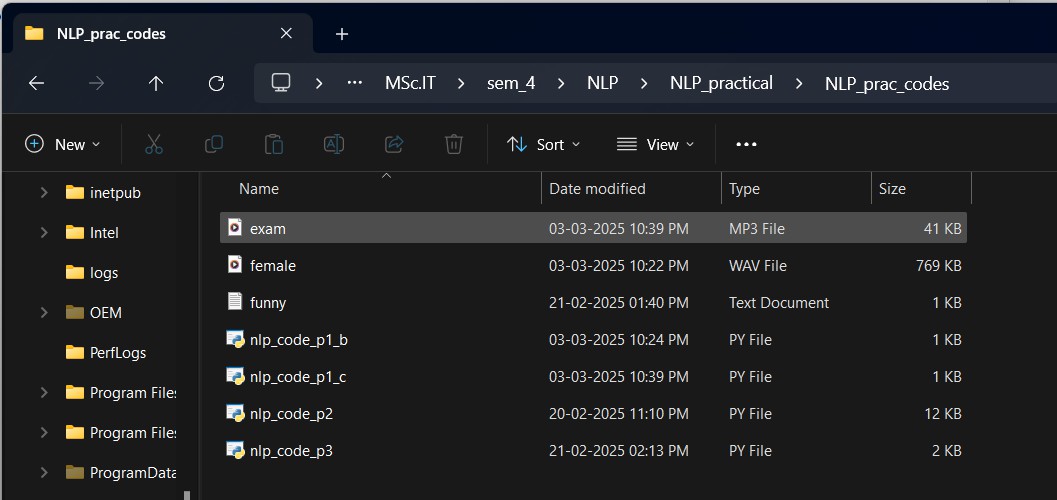
#Here are converting in English Language language = 'en'

#Passing the text and language to the engine, here we have assigned #the module that the transformed audio should have a high speed obj = gTTS(text = text\_val, lang = language, slow = False)

#Here we are saving the transformed audio in a mp3 file named exam.mp3 obj.save("exam.mp3")

#Play the exam.mp3 file playsound("exam.mp3")

Output:

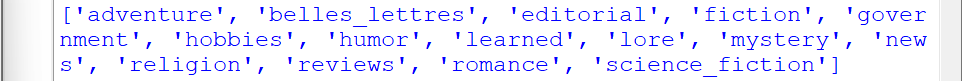


# Study of various corpus – Brown, Inaugural, Reuters, UDHR with various methods like fields, raw, words, sent, categories.

1. **Study Conditional Frequency Distribution**

Code and Output: #Brown\_corpus import nltk

from nltk.corpus import brown brown.categories()



brown.words()



brown.words(categories='news')



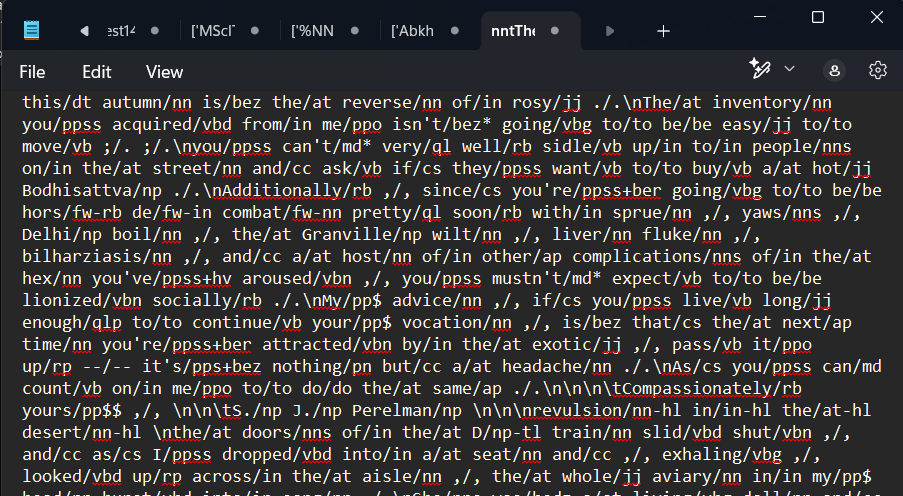
brown.words(categories='adventure')



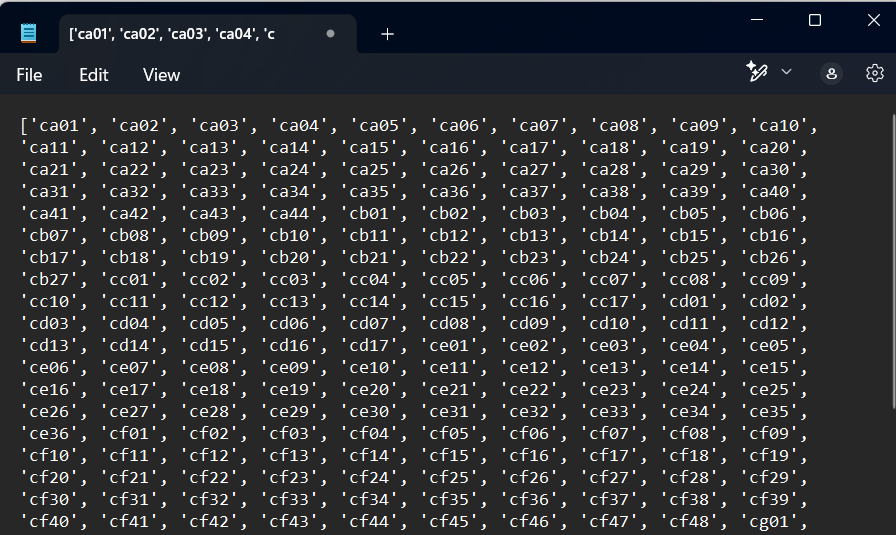
brown.words(categories='government')



brown.raw()



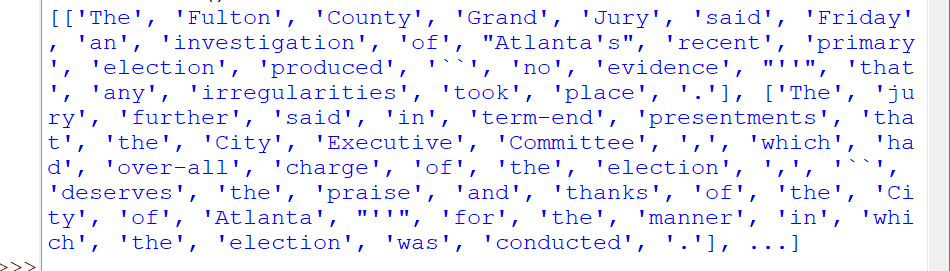
brown.fileids()



brown.words(fileids=['cg22'])

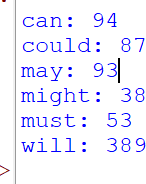


brown.sents()



fdist = nltk.FreqDist([w.lower() for w in news\_text]) modals = ['can', 'could', 'may', 'might', 'must', 'will'] for m in modals:

print(m + ':', fdist[m])



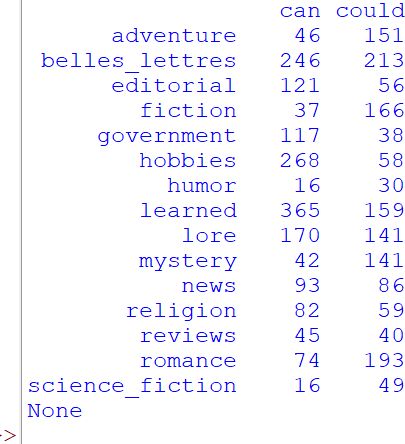
cfd = nltk.ConditionalFreqDist( (genre, word)

for genre in brown.categories()

for word in brown.words(categories=genre))

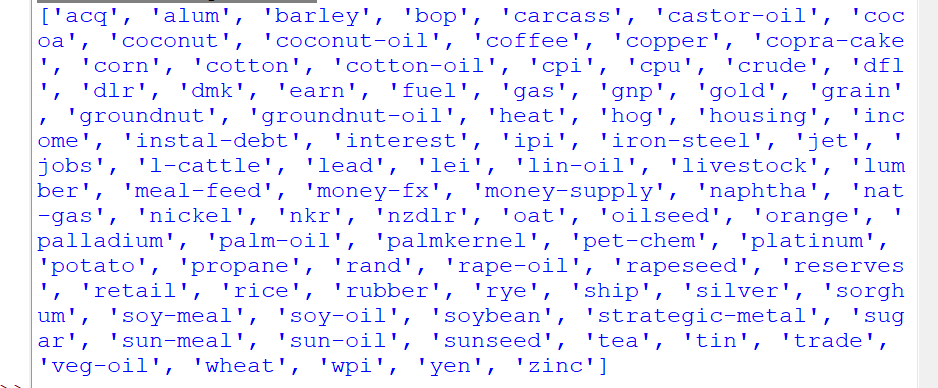
genres = ['news', 'hobbies', 'science\_fiction', 'romance']

modals = ['can', 'could'] print(cfd.tabulate(condition=genres, samples=modals))



#Reuters\_corpus

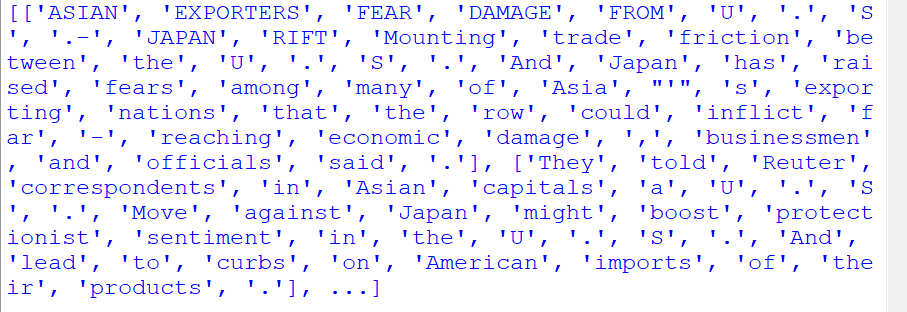
from nltk.corpus import reuters reuters.categories()



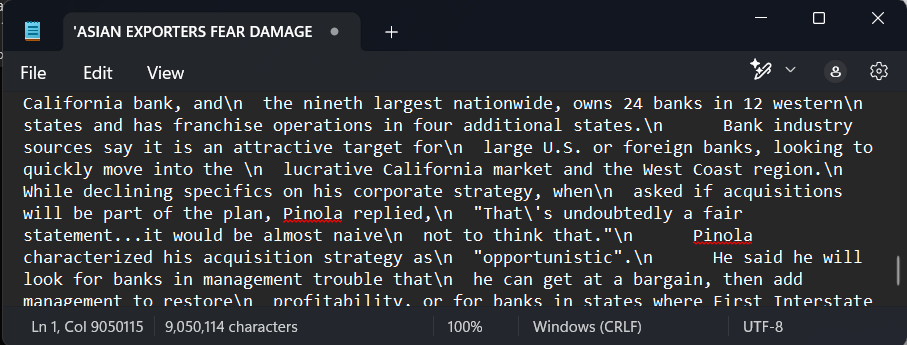
reuters.words()



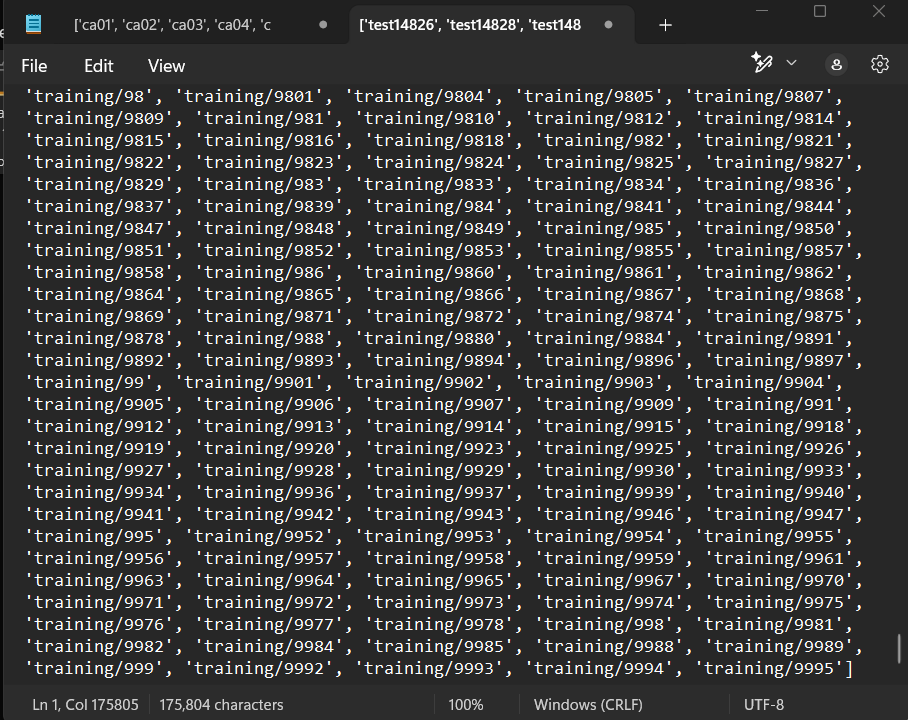
reuters.sents()



reuters.raw()



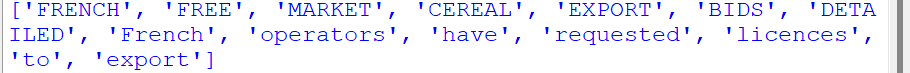
reuters.fileids()



reuters.fileids(['corn'])



reuters.words('training/9865')[:14]



reuters.words('training/9865')[:3]



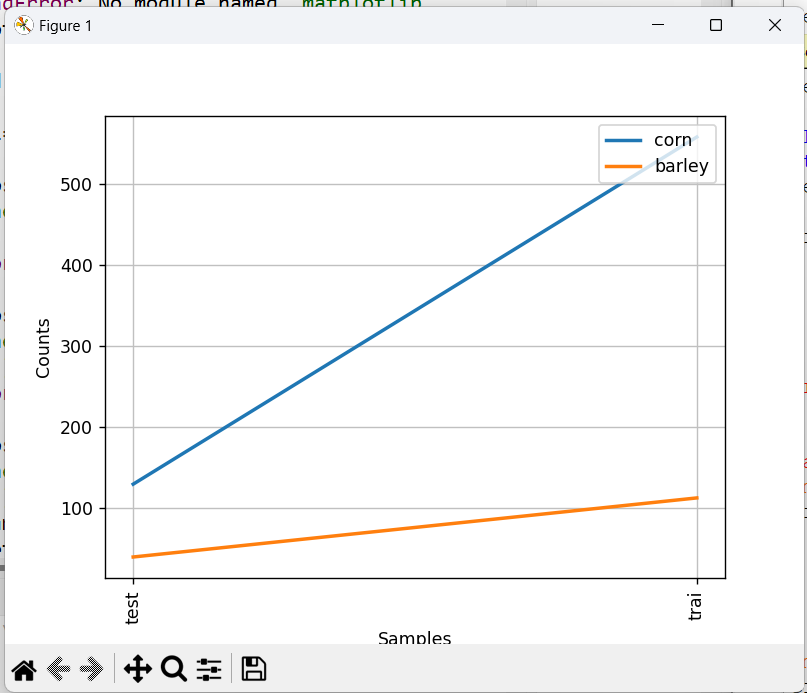
cfd = nltk.ConditionalFreqDist( (target, fileid[:4])

for fileid in reuters.fileids() for w in reuters.words(fileid) for target in ['barley', 'corn']

if w.lower().startswith(target)) import matplotlib.pyplot as plt cfd.plot()

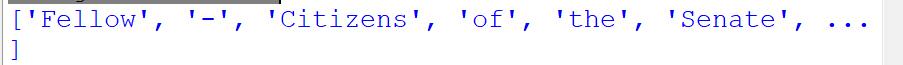


plt.show()

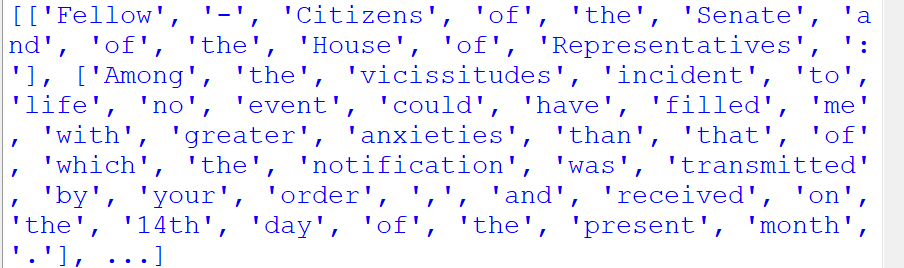


#Inaugural\_corpus

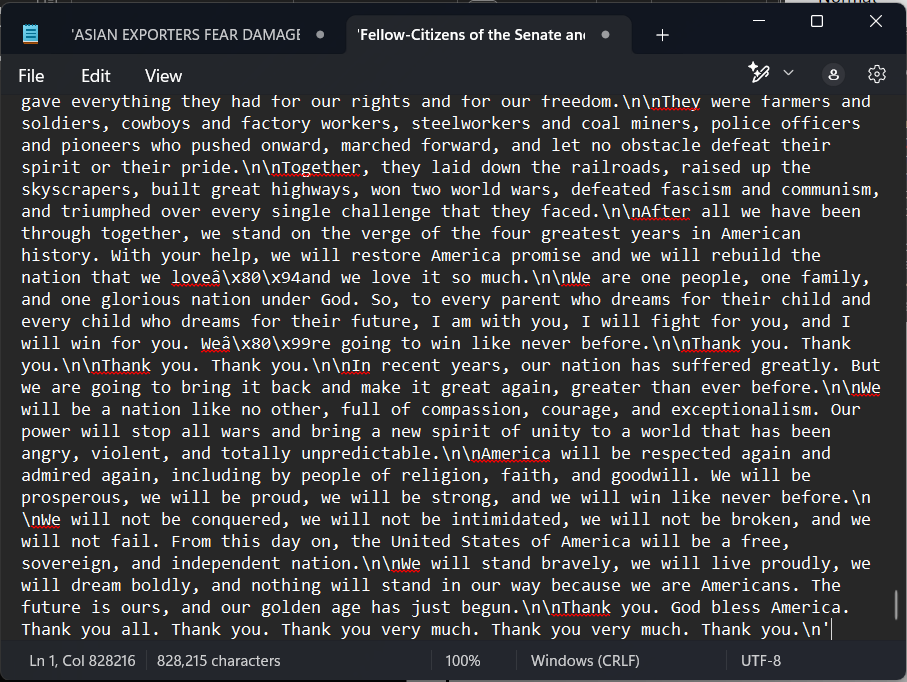
from nltk.corpus import inaugural inaugural.words()



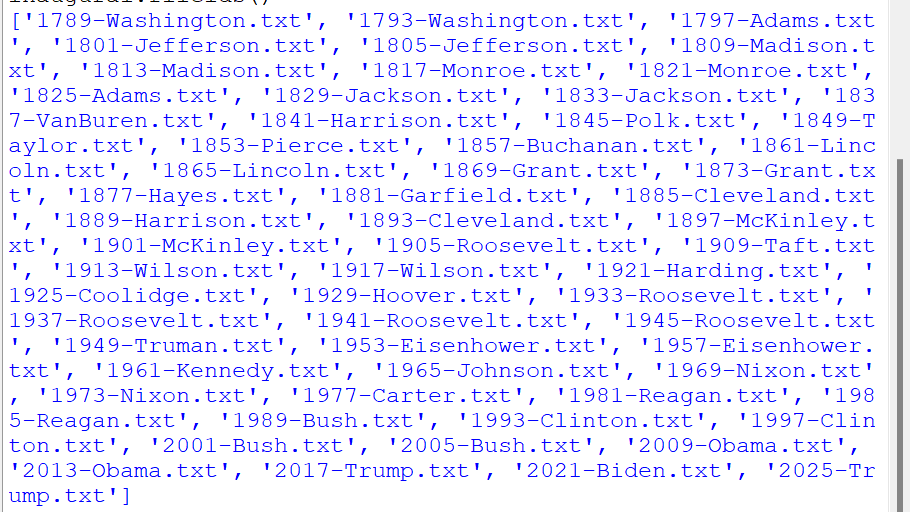
inaugural.sents()



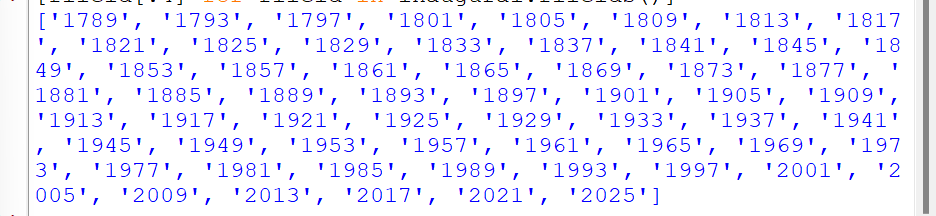
inaugural.raw()



inaugural.fileids()



[fileid[:4] for fileid in inaugural.fileids()]



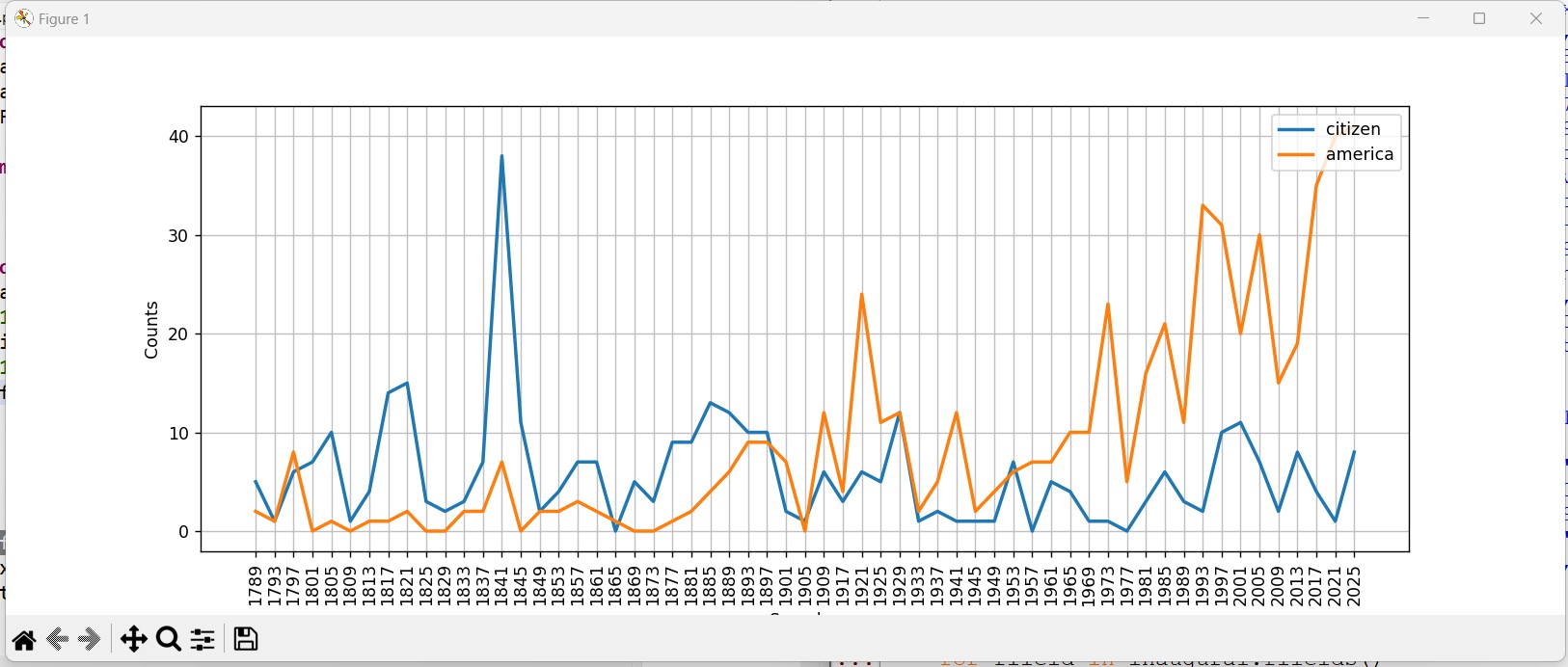
cdf = nltk.ConditionalFreqDist( (target, fileid[:4])

for fileid in inaugural.fileids() for w in inaugural.words(fileid) for target in ['america', 'citizen'] if w.lower().startswith(target))

cdf.plot()



plt.show()

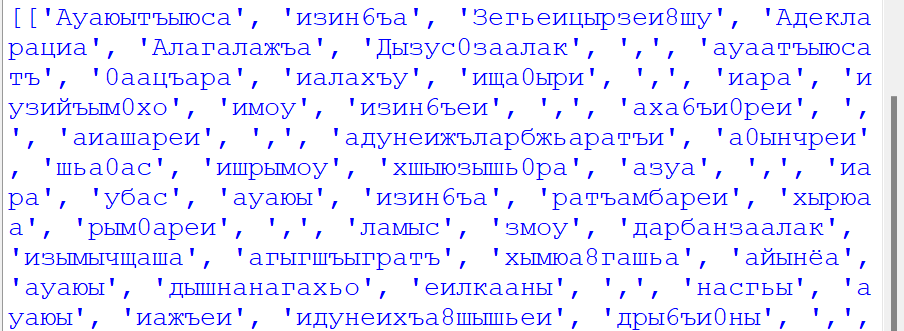


#UDHR\_corpus

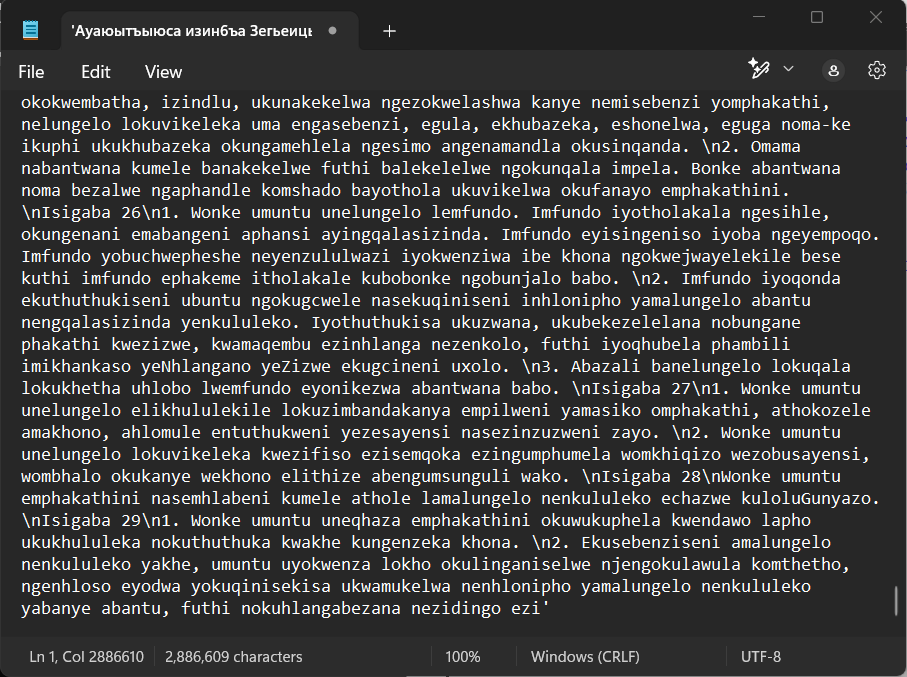
from nltk.corpus import udhr udhr.words()



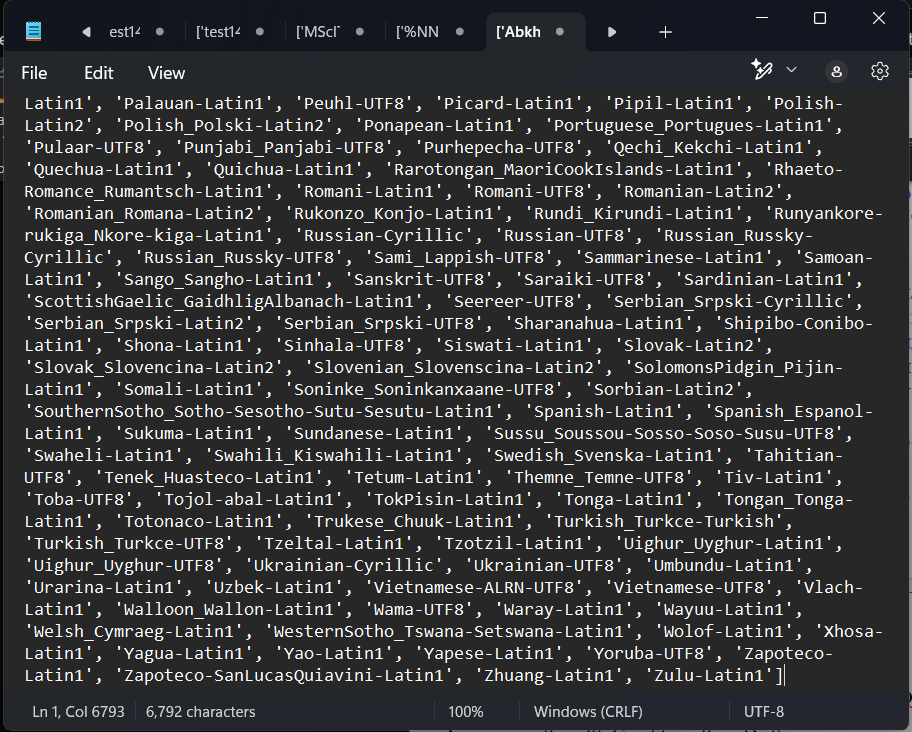
udhr.sents()



udhr.raw()



udhr.fileids()



languages = ['Chickasaw', 'English', 'German\_Deutsch'] cdf = nltk.ConditionalFreqDist(

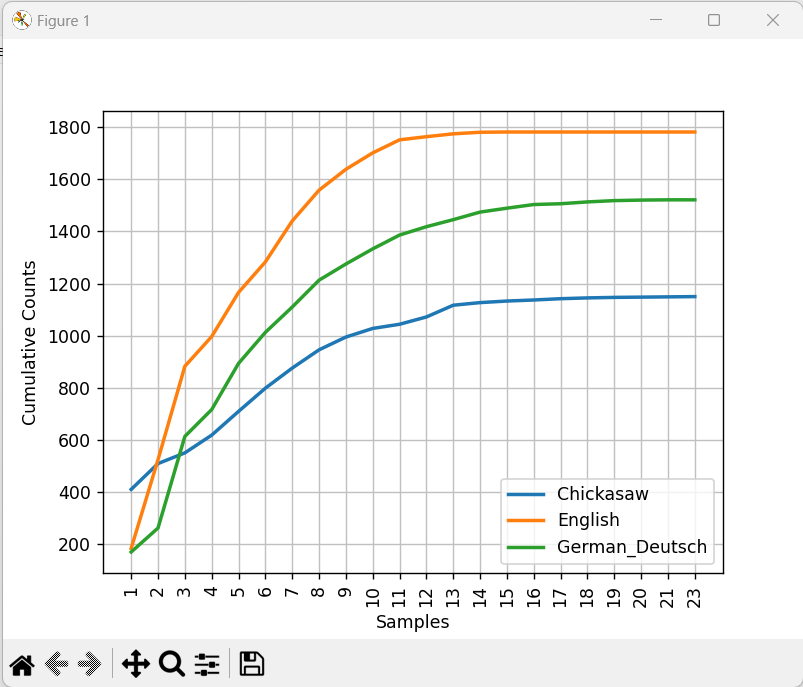
(lang, len(word))

for lang in languages

for word in udhr.words(lang+'-Latin1')) cdf.plot(cumulative=True)



plt.show()



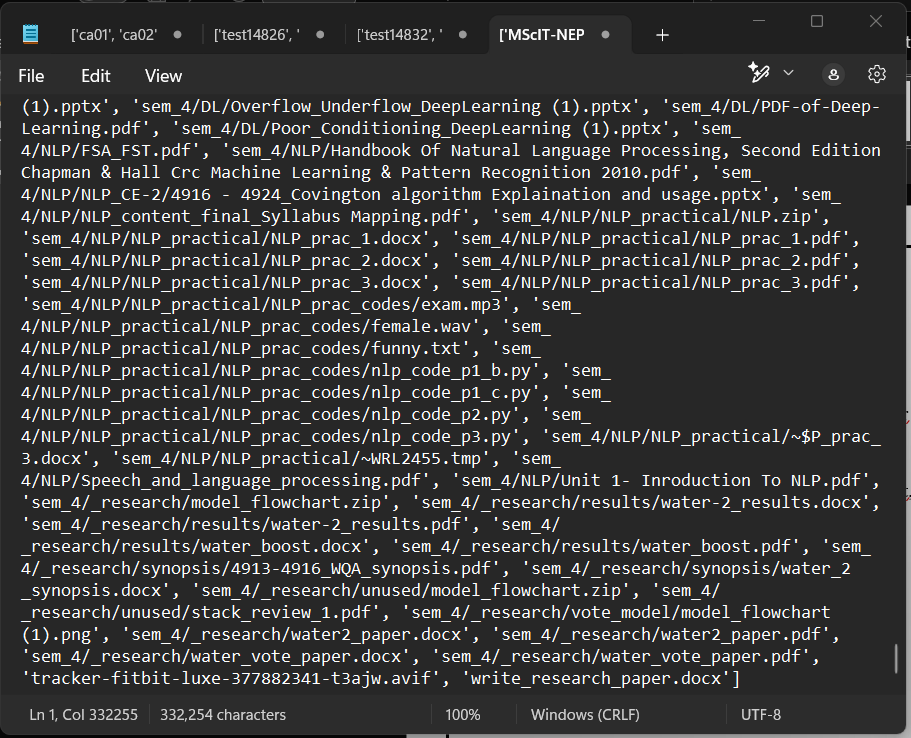
# Create and use your own corpora.

Code and Output:

import nltk

from nltk.corpus import PlaintextCorpusReader corpus\_root = 'D:\MSc.IT'

wordlists = PlaintextCorpusReader(corpus\_root, '.\*') wordlists.fileids()



wordlists.words("funny.txt")



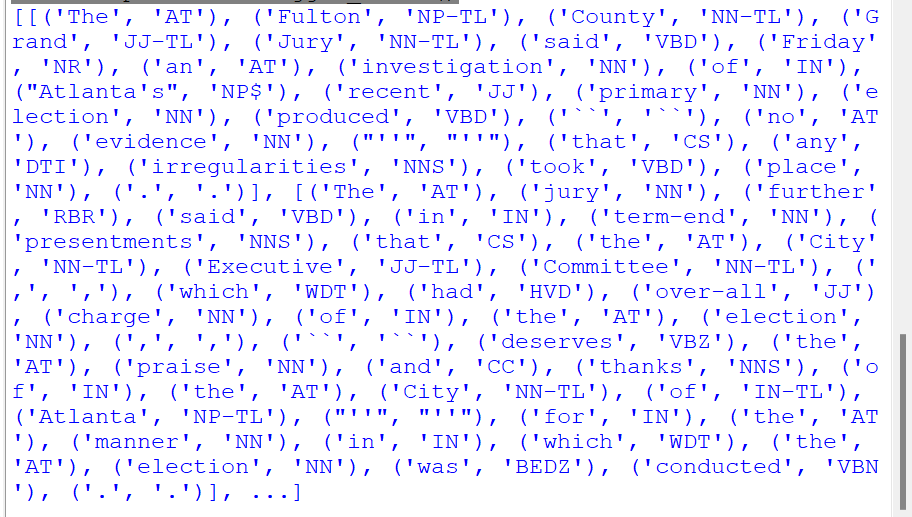
# Study of tagged corpora with methods like tagged\_sents, tagged\_words.

Code and Output:

import nltk nltk.corpus.brown.tagged\_words()



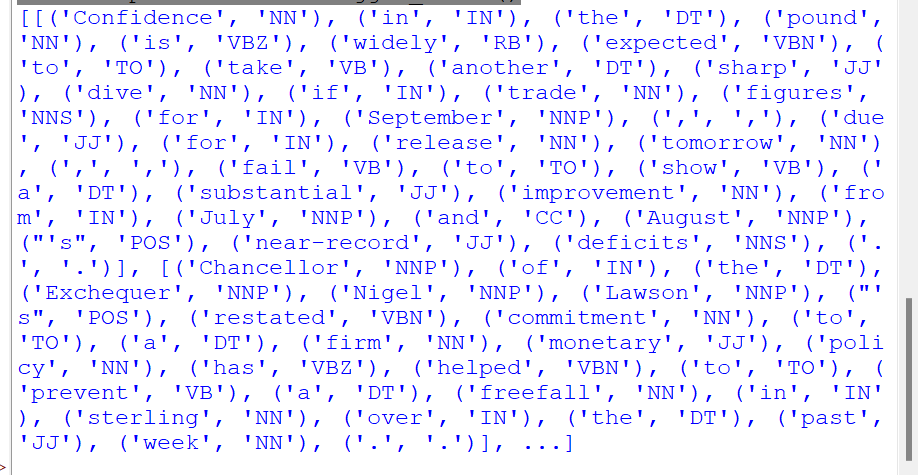
nltk.corpus.brown.tagged\_sents()



nltk.corpus.conll2000.tagged\_words()



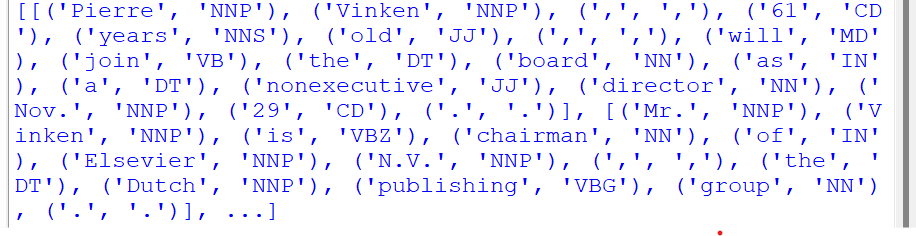
nltk.corpus.conll2000.tagged\_sents()



nltk.corpus.treebank.tagged\_words()



nltk.corpus.treebank.tagged\_sents()

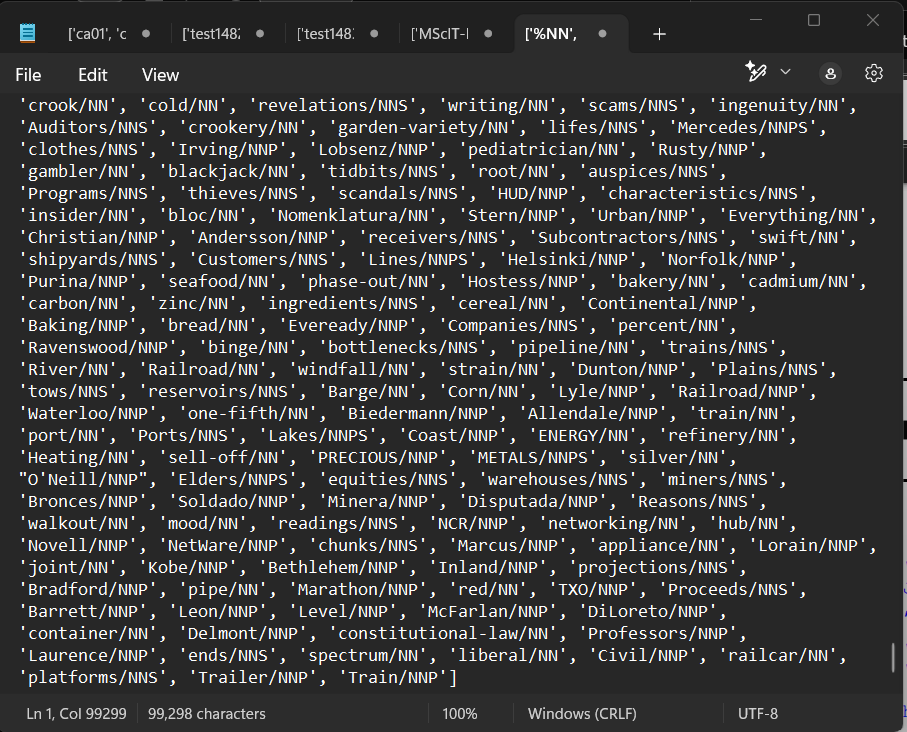


# Write a program to find the most frequent noun tags.

Code and Output:

wsj = nltk.corpus.treebank.tagged\_words() word\_tag\_fd = nltk.FreqDist(wsj)

[word + "/" + tag for (word, tag) in word\_tag\_fd if tag.startswith('N')]



# Map words to properties using python dictionaries

Code and Output:

pos={} pos



pos['colorless'] = 'ADJ' pos



pos['ideas'] = 'N'

pos['sleep'] = 'v' pos['furiously'] = 'ADV' pos



pos['ideas']



pos['colorless']



list(pos)



sorted(pos)



[w for w in pos if w.endswith('s')]



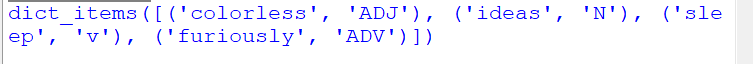
for word in sorted(pos): print(word + ":", pos[word])



pos.keys()



pos.items()



pos['sleep'] = ['N', 'V'] pos.items()



pos = {'colorless':'ADJ', 'ideas':'N', 'sleep':'V', 'furiously': 'ADV'} pos



# Study Default Tagger, Regular Expression tagger, Unigram Tagger

Code and Output:

#default\_tagger import nltk

from nltk.corpus import brown

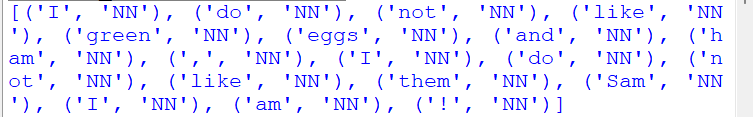
brown\_tagged\_sents = brown.tagged\_sents(categories='news') brown\_sents = brown.sents(categories='news')

tags = [tag for (word, tag) in brown.tagged\_words(categories='news')] nltk.FreqDist(tags).max()



raw = 'I do not like green eggs and ham, I do not like them Sam I am!' tokens = nltk.word\_tokenize(raw)

default\_tagger = nltk.DefaultTagger('NN') default\_tagger.tag(tokens)



default\_tagger.evaluate(brown\_tagged\_sents)



#regex\_tagger import nltk

from nltk.corpus import brown

brown\_tagged\_sents = brown.tagged\_sents(categories='news') brown\_sents = brown.sents(categories='news')

patterns = [

(r'.\*ing$', 'VBG'), # gerunds

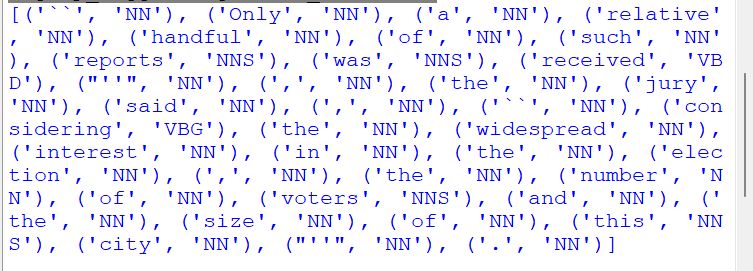
(r'.\*ed$', 'VBD'), # simple past (r'.\*es$', 'VBZ'), # 3rd singular present (r'.\*ould$', 'MD'), # modals

(r'.\*\'s$', 'NN$'), # possessive nouns (r'.\*s$', 'NNS'), # plural nouns

(r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers (r'.\*', 'NN') # nouns (default)

]

regexp\_tagger = nltk.RegexpTagger(patterns) regexp\_tagger.tag(brown\_sents[3])



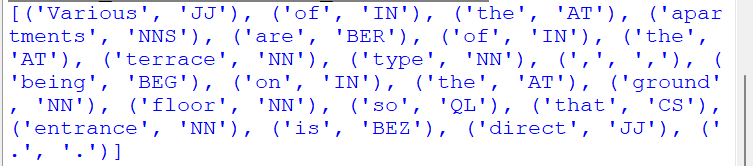
regexp\_tagger.evaluate(brown\_tagged\_sents)



#unigram\_tagger import nltk

from nltk.corpus import brown

brown\_tagged\_sents = brown.tagged\_sents(categories='news') brown\_sents = brown.sents(categories='news') unigram\_tagger = nltk.UnigramTagger(brown\_tagged\_sents) unigram\_tagger.tag(brown\_sents[2007])



unigram\_tagger.evaluate(brown\_tagged\_sents)



# Study of wordnet dictionary with methods such as synsets, definitions, examples and lemmas.

Code and Output:

import nltk

from nltk.corpus import wordnet as wn wn.synsets('motorcar')



wn.synset('car.n.01').lemma\_names()



wn.synset('car.n.01').examples()



wn.synset('car.n.01').definition()



wn.synset('fruit.n.01').definition()



wn.synset('cartoon.n.01').definition()



wn.synset('bike.n.01').definition()



wn.synset('animal.n.01').definition()



wn.synset('fruit.n.01').lemmas



wn.lemma('car.n.01.automobile')



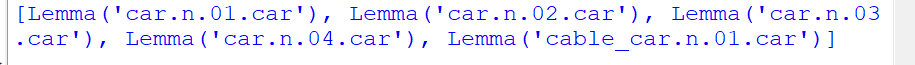
wn.lemma('car.n.01.automobile').synset()



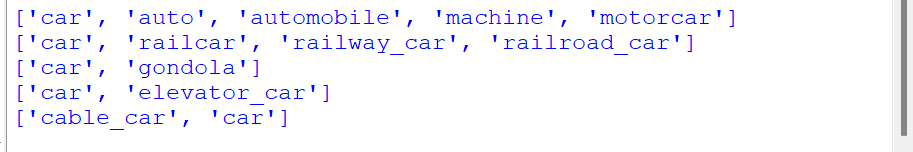
wn.lemma('car.n.01.automobile').name()



wn.lemmas('car')



for synset in wn.synsets('car'): print(synset.lemma\_names())



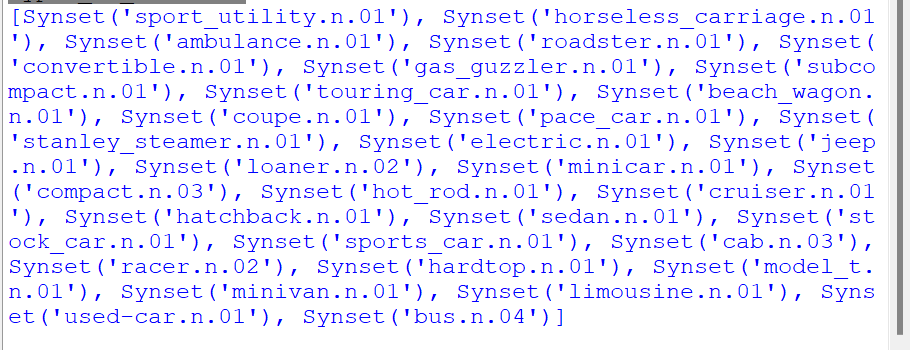
# Study of hyponyms, hypernyms, meronyms and entailments.

Code and Output:

motorcar = wn.synset('car.n.01') motorcar



types\_of\_motorcar = motorcar.hyponyms() types\_of\_motorcar



types\_of\_motorcar[8]



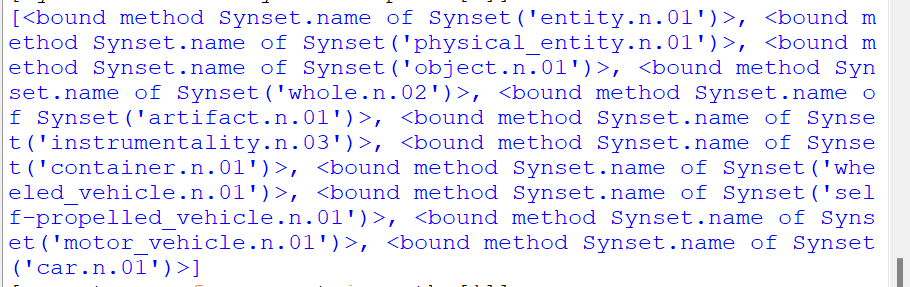
motorcar.hypernyms()



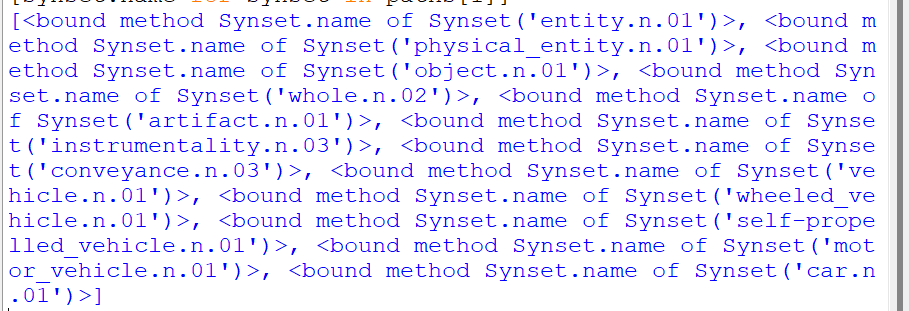
paths = motorcar.hypernym\_paths() len(paths)



[synset.name for synset in paths[0]]



[synset.name for synset in paths[1]]



motorcar.root\_hypernyms()



wn.synset('tree.n.01').part\_meronyms()



wn.synset('tree.n.01').substance\_meronyms()



wn.synset('walk.v.01').entailments()



wn.synset('eat.v.01').entailments()



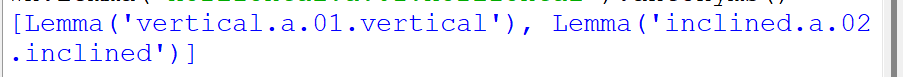
wn.lemma('supply.n.02.supply').antonyms()



wn.lemma('rush.v.01.rush').antonyms()



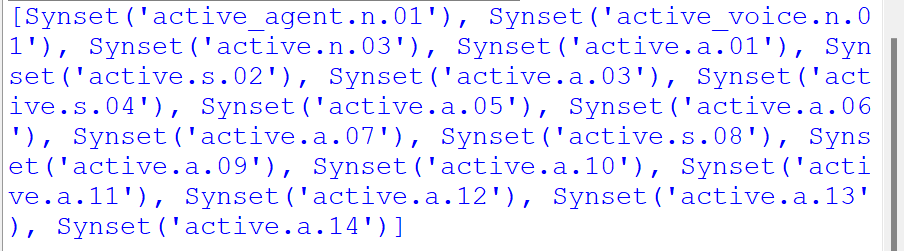
wn.lemma('horizontal.a.01.horizontal').antonyms()



# Write a program using python to find synonyms and antonyms.

Code and Output:

from nltk.corpus import wordnet print(wordnet.synsets("active"))



print(wn.lemma('active.a.01.active').antonyms())



# Compare two nouns and find out path similarity between them.

Code:

import nltk

from nltk.corpus import wordnet syn1 = wordnet.synsets('football') syn2 = wordnet.synsets('soccer')

for s1 in syn1: for s2 in syn2:

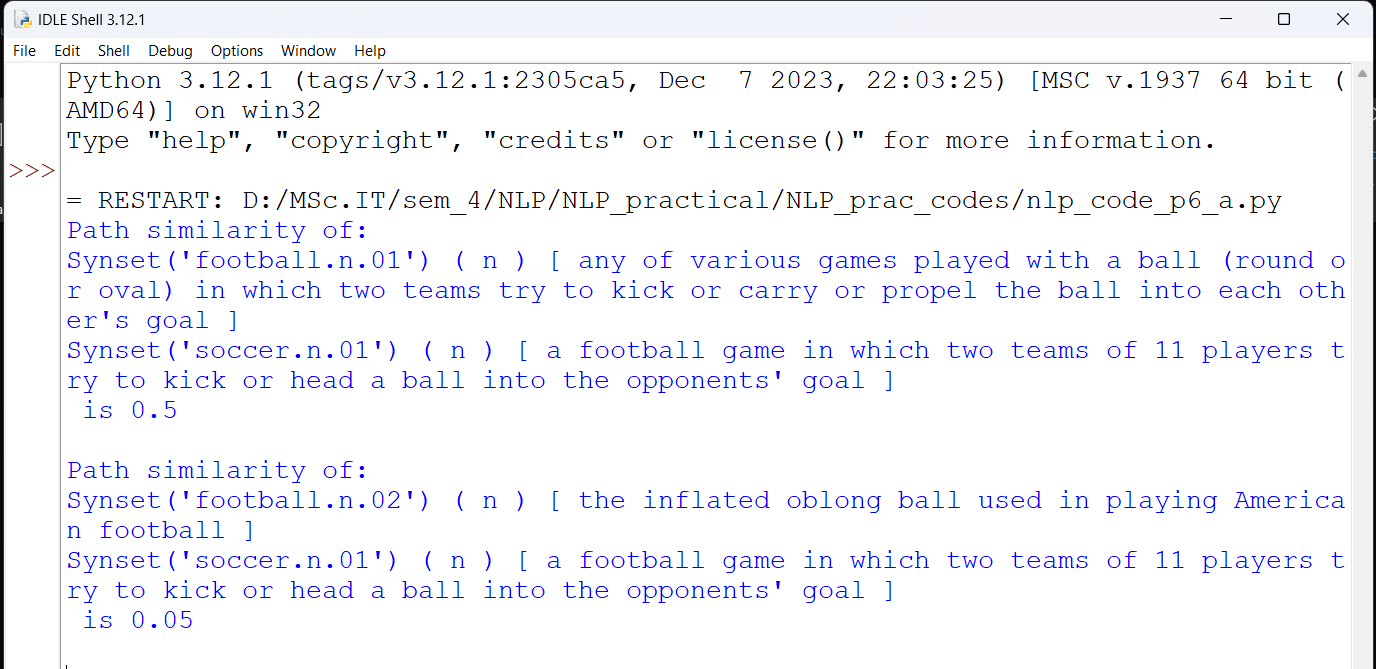
print("Path similarity of: ")

print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')

print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']') print(" is", s1.path\_similarity(s2))

print()

Output:



1. **Handling Stop words.**
   * **Adding or removing stop words in NLTKs Default stop word list.**
   * **Using gensim adding and removing stop words in Default gensim stop word list.**
   * **Using spacy adding and removing stop words in Default spacy stop word list.**

Code and Output:

#Adding or removing stop words in NLTKs Default stop word list. import nltk

from nltk.corpus import stopwords nltk.download('stopwords')

from nltk.tokenize import word\_tokenize print(stopwords.words())

text = "yashesh likes to play football, however he is not to fond of tennis" text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words()] print(tokens\_without\_sw)

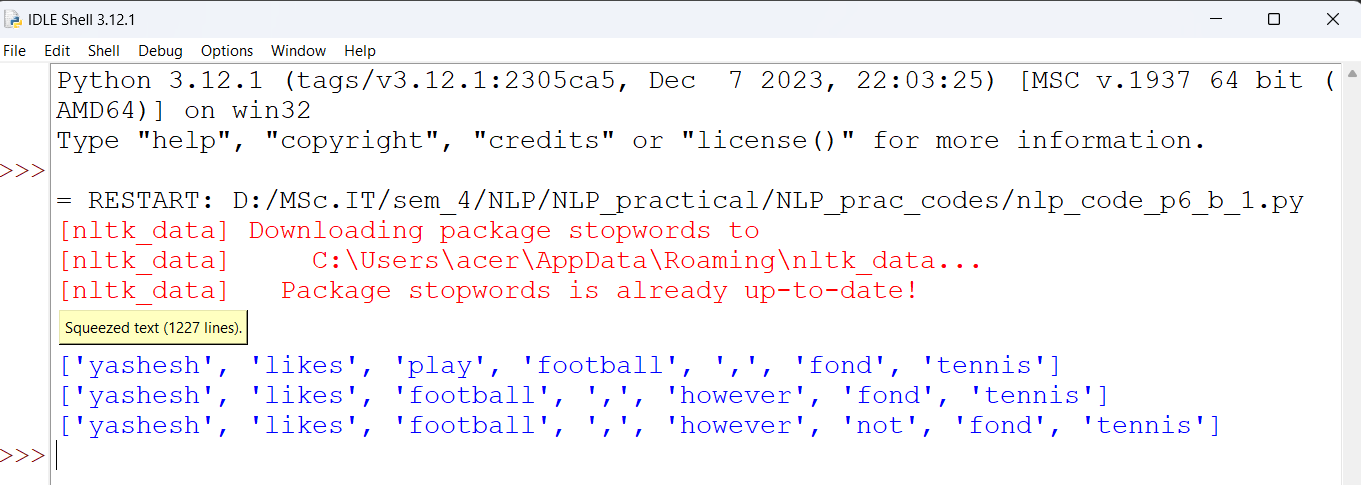
all\_stopwords = stopwords.words('english') all\_stopwords.append('play')

text\_tokens = word\_tokenize(text)

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

all\_stopwords.remove('not') text\_tokens = word\_tokenize(text)

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



#Using gensim adding and removing stop words in Default gensim stop word list. import nltk

nltk.download('punkt\_tab')

import gensim

from gensim.parsing.preprocessing import remove\_stopwords

text = "yasesh likes to play football, however he is not too fond of tennis" filtered\_sentence = remove\_stopwords(text)

print(filtered\_sentence)

all\_stopwords = gensim.parsing.preprocessing.STOPWORDS print(all\_stopwords)

from gensim.parsing.preprocessing import STOPWORDS from nltk.tokenize import word\_tokenize

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

text = "yasesh likes to play football, however he is not too fond of tennis" text\_tokens = word\_tokenize(text)

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

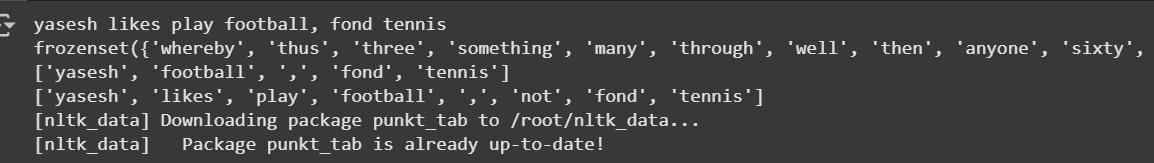
#the following script removes the word "not" from the set of stop words from gensim.parsing.preprocessing import STOPWORDS all\_stopwords\_gensim = STOPWORDS

sw\_list = {"not"}

all\_stopwords\_gensim = STOPWORDS.difference(sw\_list)

text = "yasesh likes to play football, however he is not too fond of tennis" text\_tokens = word\_tokenize(text)

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



#Using spacy adding and removing stop words in Default spacy stop word list. import spacy

import nltk

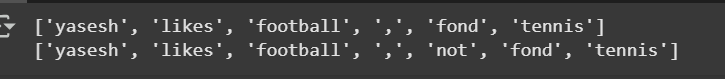
from nltk.tokenize import word\_tokenize sp = spacy.load("en\_core\_web\_sm") all\_stopwords = sp.Defaults.stop\_words all\_stopwords.add("play")

text = "yasesh likes to play football, however he is not too fond of tennis" text\_tokens = word\_tokenize(text)

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

# remove 'not' from word collection all\_stopwords.remove('not')

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



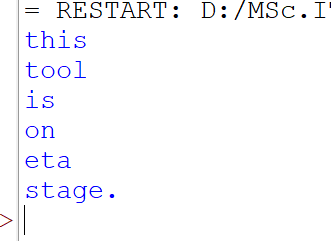
# Text Tokenization using python split().

Code:

text = "this tool is on beta stage." data = text.split()

for i in data: print (i)

Output:



# Text Tokenization using python Regex.

Code:

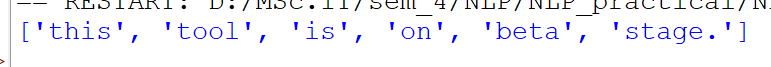
import nltk

from nltk.tokenize import RegexpTokenizer

tk = RegexpTokenizer(r'\s+', gaps=True) # Use raw string (r'\s+') text = "this tool is on beta stage."

tokens = tk.tokenize(text) print(tokens)

Output:



# Text Tokenization using python NLTK.

Code:

import nltk

from nltk.tokenize import word\_tokenize str = "this tool is on beta stage." print(word\_tokenize(str))

Output:



# Text Tokenization using spacy library.

Code:

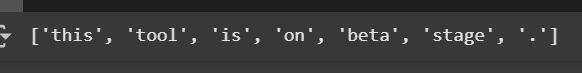
#NOTE: code in colab import spacy

nlp = spacy.blank("en")

str = "this tool is on beta stage." doc = nlp(str)

words = [word.text for word in doc] print(words)

Output:



# Text Tokenization using Keras.

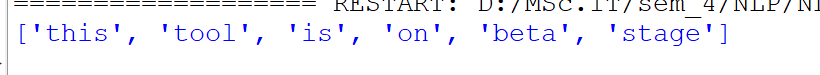
Code:

import tensorflow import keras

from tensorflow.keras.preprocessing.text import text\_to\_word\_sequence str = "this tool is on beta stage."

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

Output:



# Sentence tokenization, word tokenization, Part of speech Tagging and chunking of user defined text.

Code:

import nltk

from nltk import tokenize nltk.download('punkt\_tab') from nltk import tag

from nltk import chunk nltk.download('averaged\_perceptron\_tagger\_eng') nltk.download('maxent\_ne\_tagger\_tab') nltk.download('words')

para = "today you will learn NLTK" 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)

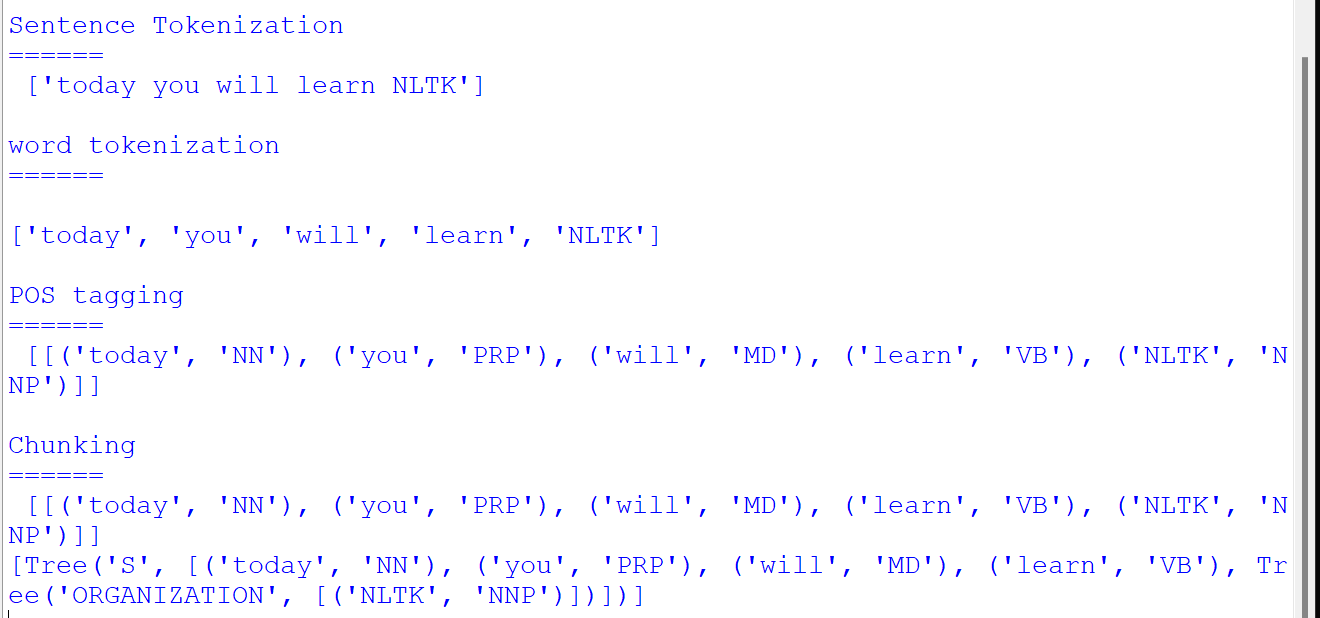
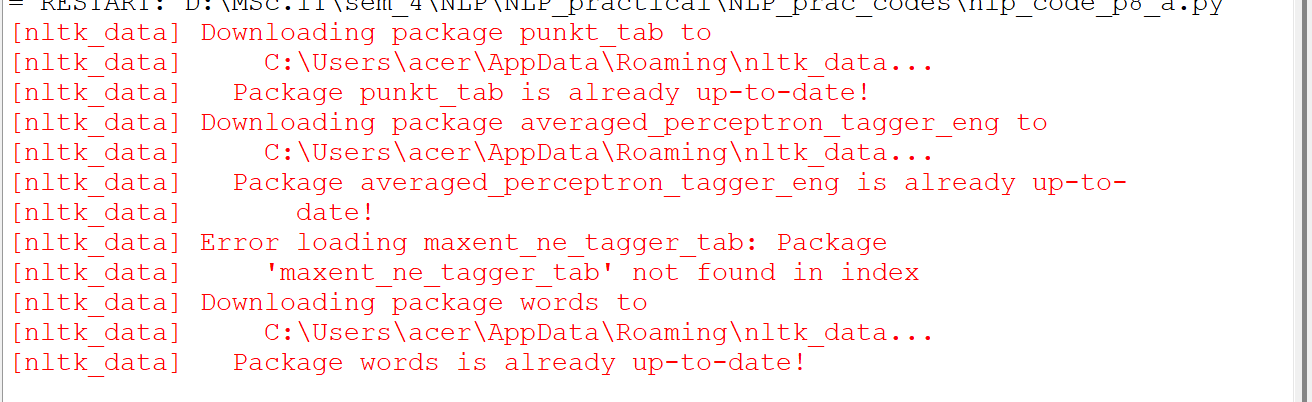
#chunking

tree = []

for index in range(len(sents)): tree.append(chunk.ne\_chunk(tagged\_words[index]))

print("\nChunking\n======\n", tagged\_words) print(tree)

Output:



# Named Entity Recognition of user defined text .

Code:

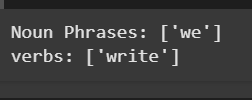
import spacy

nlp = spacy.load("en\_core\_web\_sm") text = "we are writing practical"

doc = nlp(text)

print("Noun Phrases:", [chunk.text for chunk in doc.noun\_chunks]) print("verbs:", [token.lemma\_ for token in doc if token.pos\_ == "VERB"])

Output:



# Named Entity Recognition with diagram using NLTK corpus - treebank

Code:

import nltk nltk.download('treebank')

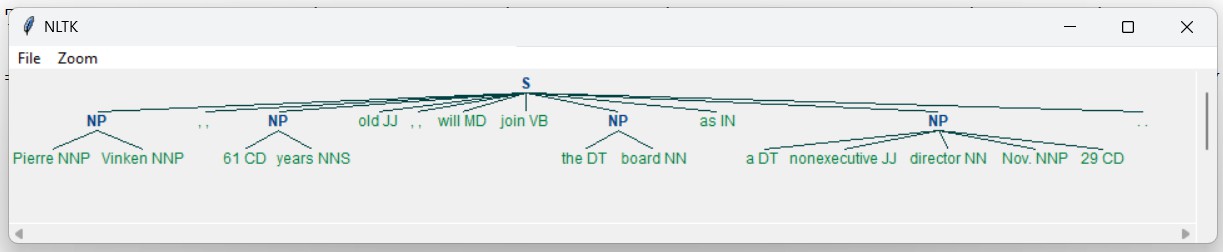
from nltk.corpus import treebank\_chunk treebank\_chunk.tagged\_sents() [0]

treebank\_chunk.chunked\_sents() [0] #for Google colab

#treebank\_chunk.chunked\_sents() [0].pretty\_print() #for Idle

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

Output:



# Word Tokenization In Hindi CODE:

!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

!pip install inltk

!pip install tornado==4.5.3 from inltk.inltk import setup setup('hi')

from inltk.inltk import tokenize

hindi\_text = """प्राTकृ! तिKकृ भाTषाT सी(खनाT बहुK तिKलचस्प है1।"""

tokenize(hindi\_text, "hi")

**Output:**

**['प्राTकृ! ', 'ति#कृ', 'भाTषाT', 'सी'खनाT', 'बहु#', 'ति#लचस्प', 'है0।']**

1. **Generate Similar Sentences From A Given Hindi Text Input CODE:**

!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

!pip install inltk

!pip install tornado==4.5.3 from inltk.inltk import setup setup('hi')

from inltk.inltk import get\_similar\_sentences

output = get\_similar\_sentences('मैं4आज बहुK ख7श हूं:', 5, 'hi')

print(output)

# Output:

[

'मैं4 आज बहुK अच्छाT हूं:।', मैं4 आज बहुK बढ़ि!याT हूं:।', आज मैं4 बहुK ख7श हूं:।', मैं4 आज कृTफी$ ख7श हूं:।',

'आज मैं4 बहुK अच्छाT हूं:।']

# Identify The Indian Language Of A Text CODE:

!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

!pip install inltk

!pip install tornado==4.5.3 from inltk.inltk import setup setup('gu')

from inltk.inltk import identify\_language identify\_language('@tનાL કાL4PCuL')

# Output:

Gujarati