Natural Language Processing

Journal

M.Sc. I.T Part II Semester IV

2020-2021

RJSPGIT4P1

Name: Sumera Hangi

Roll No: 06



**CERTIFICATE**

This is to certify that **Miss. Sumera Hangi** with Seat No.**06** has successfully completed the necessary course of experiments in the subject of **NATURAL LANGUAGE PROCESSING** during the academic year **2020 – 2021** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -IV.

Internal Examiner Date:

Head of Department College Seal External Examiner

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR. NO.** | **PRACTICAL TITLE** | **DATE** | **Page No.** |
| **1.** | **Regular Expression** |  |  |
| 1. | **Processing Raw Text**   1. Accessing text from Web 2. Accessing local text file 3. Accessing text from PDF, Word and other Binary Format 4. The NLP pipeline |  |  |
| 2. | **Text Processing – 1**  Tokenize Text using NLTK   1. Sentence Tokenization 2. PunktSentenceTokenizer 3. Tokenize sentence of different language 4. Word Tokenization 5. Using TreebankWordTokenizer 6. PunktWordTokenizer 7. WordPunctTokenizer 8. Using Regular Expression |  |  |
| 3. | **Text Processing – 2**   1. Accessing Text Corpora    1. Gutenberg Corpus    2. Web and Chat Text    3. Brown Corpus    4. Annotated Text Corpora    5. Corpora in other languages    6. Text Corpus Structure    7. Loading your own corpus 2. Word Counting 3. Word Vocabulary 4. Bag-of-words, BOW Model 5. TF/IDF Vectorizer 6. Tokenisation and Word Frequencies 7. Sentence Segmentation 8. Removing Stop Words with NLTK |  |  |
| 4. | **Text Processing**   1. Text Lowercase 2. Remove Numbers 3. Remove Punctuation 4. Remove Whitespaces 5. Remove Default Stopwords 6. Stemming 7. Lemmatization |  |  |
| 5. | **Phonetic Hashing Using Soundex Algorithm** |  |  |
| 6. | **Information Extraction**   1. Part-of-Speech Tagging 2. Chunking 3. Chinking 4. Named Entity Recognition 5. Relation Extraction |  |  |
| 7. | **Classification**   1. Supervised Classification - Gender Identification 2. Choosing right features 3. Document classification 4. Sentence Segmentation 5. Naive Bayes Classifier |  |  |
|  | **Parsing Tree POS Tagging** |  |  |
|  | **Using Wordnet Finding Synonym and Antonym** |  |  |
|  | **Word Sense Disambiguation - Leak Algorithm** |  |  |
|  | **Word2Vec** |  |  |

**Practical No. 1: Regular Expression**

**Aim: Write a program for regular expression**

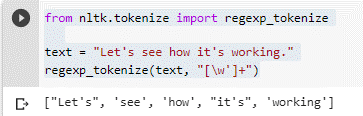
**Description:**

**Code:**

>> from nltk.tokenize import regexp\_tokenize

>> text = "Let's see how it's working."

>> regexp\_tokenize(text, "[\w']+")



**Practical No. 2: Processing Raw Text**

**Aim: Write a program for processing raw text.**

**Description:**

The most important source of texts is undoubtedly the Web. It's convenient to have existing text collections to explore, such as the corpora we saw in the previous chapters. However, you probably have your own text sources in mind, and need to learn how to access them.

The goal of this chapter is to answer the following questions:

1. How can we write programs to access text from local files and from the web, in order to get hold of an unlimited range of language material?
2. How can we split documents up into individual words and punctuation symbols, so we can carry out the same kinds of analysis we did with text corpora in earlier chapters?
3. How can we write programs to produce formatted output and save it in a file?

In order to address these questions, we will be covering key concepts in NLP, including tokenization and stemming. Along the way you will consolidate your Python knowledge and learn about strings, files, and regular expressions. Since so much text on the web is in HTML format, we will also see how to dispense with markup.

1. **Accessing text from Web**

A small sample of texts from Project Gutenberg appears in the NLTK corpus collection. However, you may be interested in analyzing other texts from Project Gutenberg. You can browse the catalog of 25,000 free online books at http://www.gutenberg.org/catalog/, and obtain a URL to an ASCII text file. Although 90% of the texts in Project Gutenberg are in English, it includes material in over 50 other languages, including Catalan, Chinese, Dutch, Finnish, French, German, Italian, Portuguese and Spanish (with more than 100 texts each).

**Code:**

>>import nltk, re, pprint

>>from nltk import word\_tokenize

>>from urllib import request

>>url = "http://www.gutenberg.org/files/2554/2554-0.txt"

>>response = request.urlopen(url)

>>raw = response.read().decode('utf8')

>>type(raw)

str

>>raw[:75]

The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky

>>import nltk

>>nltk.download('punkt')

>>tokens = word\_tokenize(raw)

>>type(tokens)

list

>>len(tokens)

257727

>>tokens[:10]

['\ufeffThe','Project','Gutenberg','EBook','of','Crime','and','Punishment,

',','by']

>>text = nltk.Text(tokens)

>>type(text)

>>text[1024:1062]

['an','exceptionally','hot','evening','early','in','July','a','young','man','came','out','of','the','garret','in','which','he','lodged','in','S.','Place','and','walked','slowly',',','as','though','in','hesitation',',','towards','K.','bridge','.','He','had','successfully']

>>nltk.download('stopwords')

>>text.collocations()

Katerina Ivanovna; Pyotr Petrovitch; Pulcheria Alexandrovna; Avdotya

Romanovna; Rodion Romanovitch; Marfa Petrovna; Sofya Semyonovna; old

woman; Project Gutenberg-tm; Porfiry Petrovitch; Amalia Ivanovna;

great deal; young man; Nikodim Fomitch; Ilya Petrovitch; Project

Gutenberg; Andrey Semyonovitch; Hay Market; Dmitri Prokofitch; Good

heavens

>>raw.find("PART I")

5336

>>raw.rfind("End of Project Gutenberg's Crime")

-1

>>raw = raw[5338:1157743]

>>raw.find("PART I")

195769

1. **Accessing local text file**

In order to read a local file, we need to use Python's built-in open() function, followed by the read() method. Suppose you have a file document.txt, you can load its contents like this

**Note: Your Turn:** Create a file called document.txt using a text editor, and type in a few lines of text, and save it as plain text. If you are using IDLE, select the New Window command in the File menu, typing the required text into this window, and then saving the file as document.txt inside the directory that IDLE offers in the pop-up dialogue box. Next, in the Python interpreter, open the file using f = open('document.txt'), then inspect its contents using print(f.read()).

**Code:**

>>f = open('document.txt')

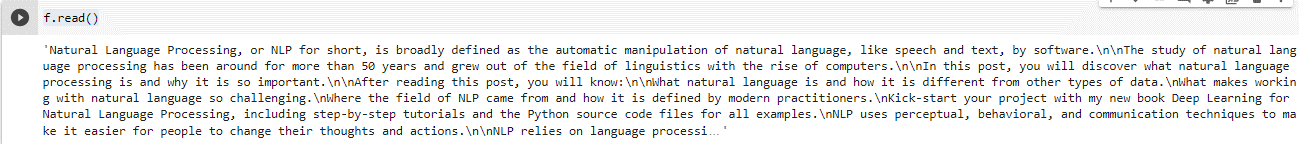
>>raw = f.read()

>>f = open('document.txt')

>>import os

>>os.listdir('.')

['.config', 'document.txt', 'sample\_data']

>>f.read()

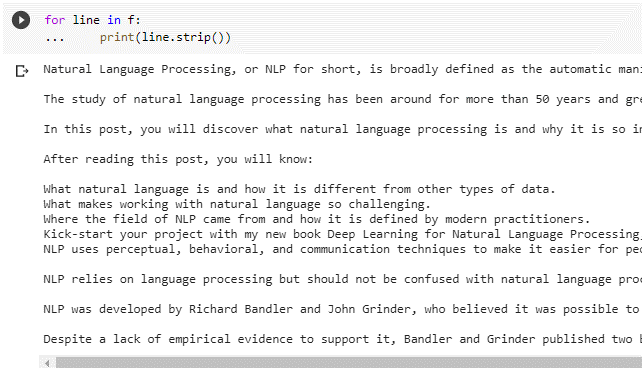
>>f = open('document.txt', 'rU')

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:1: DeprecationWarning: 'U' mode is deprecated

"""Entry point for launching an IPython kernel.

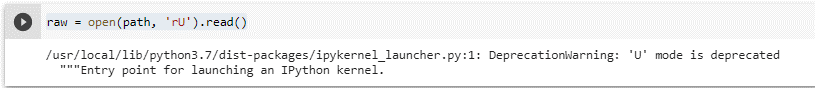
>>for line in f:

...     print(line.strip())



>>nltk.download('gutenberg')

>>path = nltk.data.find('corpora/gutenberg/melville-moby\_dick.txt')

>>raw = open(path, 'rU').read()

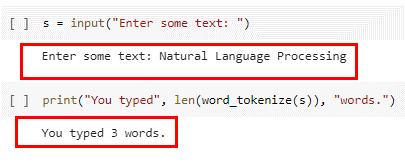
1. **Accessing text from PDF, Word and other Binary Format**

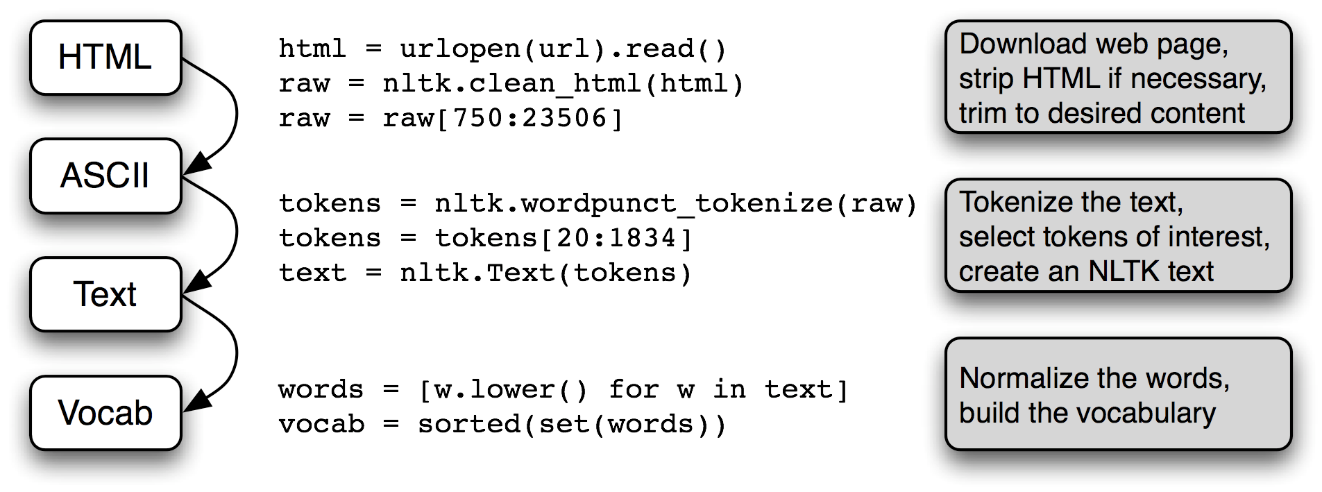
ASCII text and HTML text are human readable formats. Text often comes in binary formats — like PDF and MSWord — that can only be opened using specialized software. Third-party libraries such as pypdf and pywin32 provide access to these formats. Extracting text from multi-column documents is particularly challenging. For once-off conversion of a few documents, it is simpler to open the document with a suitable application, then save it as text to your local drive, and access it as described below. If the document is already on the web, you can enter its URL in Google's search box. The search result often includes a link to an HTML version of the document, which you can save as text.

**Code:**

>> s = input("Enter some text: ")

>>print("You typed", len(word\_tokenize(s)), "words.")



1. **The NLP pipeline**

**Figure**: The Processing Pipeline: We open a URL and read its HTML content, remove the markup and select a slice of characters; this is then tokenized and optionally converted into an nltk.Text object; we can also lowercase all the words and extract the vocabulary.

There's a lot going on in this pipeline. To understand it properly, it helps to be clear about the type of each variable that it mentions. We find out the type of any Python object x using type(x), e.g. type(1) is <int> since 1 is an integer.

**Code:**

>>raw = open('document.txt').read()

>>type(raw)

str

>>tokens = word\_tokenize(raw)

>>type(tokens)

list

>>words = [w.lower() for w in tokens]

>>type(words)

>>vocab = sorted(set(words))

>>type(vocab)

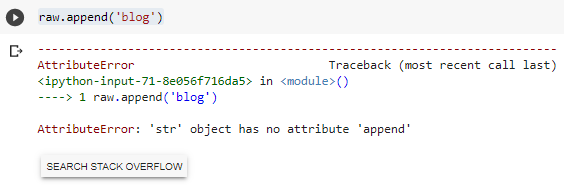
list

>>vocab.append('blog')

>>print(vocab)

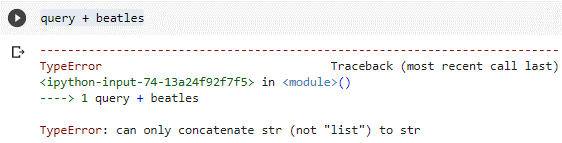
>>print(raw)

>>raw.append('blog')



>>query = 'Who knows?'

>>beatles = ['john', 'paul', 'george', 'ringo']

>>query + beatles

**Practical No. 3: Text Processing – 1**

**Aim: Write a program for text processing 1**

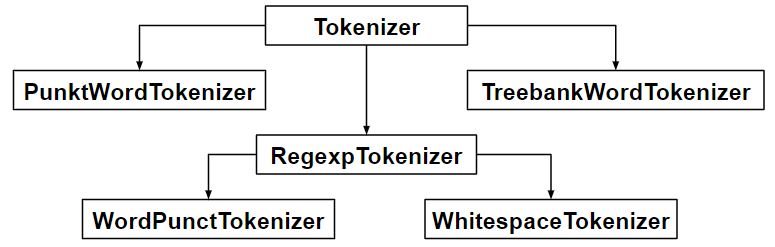
**Description:**

**Natural Language Processing (NLP)** is a subfield of computer science, artificial intelligence, information engineering, and human-computer interaction. This field focuses on how to program computers to process and analyze large amounts of natural language data. It is difficult to perform as the process of reading and understanding languages is far more complex than it seems at first glance.

**Tokenization** is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

**Key points of the article –**

* Text into sentences tokenization
* Sentences into words tokenization
* Sentences using regular expressions tokenization



1. **Tokenize Text using NLTK**

To run the below python program, (NLTK) natural language toolkit has to be installed in your system.  
The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology.  
In order to install NLTK run the following commands in your terminal.

* **sudo pip install nltk**
* Then, enter the python shell in your terminal by simply typing **python**
* Type **import nltk**
* **nltk.download(‘all’)**

**Code:**

# import the existing word and sentence tokenizing

# libraries

>> from nltk.tokenize import sent\_tokenize, word\_tokenize

>> text = "Natural language processing (NLP) is a field " + \

      "of computer science, artificial intelligence " + \

       "and computational linguistics concerned with " + \

       "the interactions between computers and human " + \

       "(natural) languages, and, in particular, " + \

       "concerned with programming computers to " + \

       "fruitfully process large natural language " + \

       "corpora. Challenges in natural language " + \

 "processing frequently involve natural " + \

       "language understanding, natural language" + \

       "generation frequently from formal, machine" + \

       "-readable logical forms), connecting language " + \

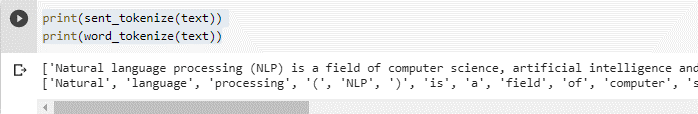
       "and machine perception, managing human-" + \

       "computer dialog systems, or some combination " + \

       "thereof."

>> import nltk

>> print(sent\_tokenize(text))

>> print(word\_tokenize(text))

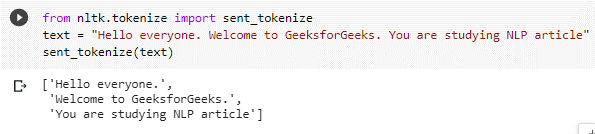
1. **Sentence Tokenization:** Splitting sentences in the paragraph.

**Code:**

>> from nltk.tokenize import sent\_tokenize

>>text = "Hello everyone. Welcome to GeeksforGeeks. You are studying NLP article"

>> sent\_tokenize(text)



1. **PunktSentenceTokenizer:** When we have huge chunks of data then it is efficient to use it.

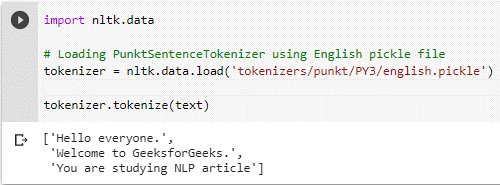
**Code:**

>> import nltk.data

# Loading PunktSentenceTokenizer using English pickle file

>> tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')

>> tokenizer.tokenize(text)



1. **Tokenize sentence of different language:** One can also tokenize sentence from different languages using different pickle file other than English.

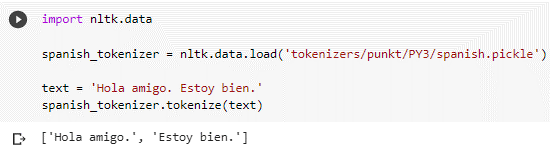
**Code:**

>> import nltk.data

>>spanish\_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')

>> text = 'Hola amigo. Estoy bien.'

>> spanish\_tokenizer.tokenize(text)



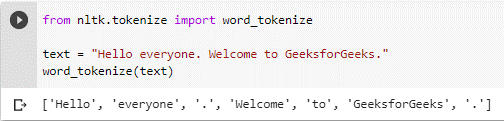
1. **Word Tokenization:** Splitting words in a sentence.

**Code:**

>> from nltk.tokenize import word\_tokenize

>> text = "Hello everyone. Welcome to GeeksforGeeks."

>> word\_tokenize(text)



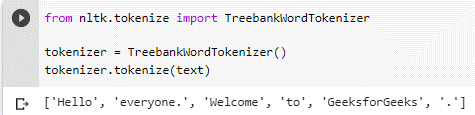
1. **Using TreebankWordTokenizer:** These tokenizers work by separating the words using punctuation and spaces. And as mentioned in the code outputs above, it does not discard the punctuation, allowing a user to decide what to do with the punctuations at the time of pre-processing.

**Code:**

>> from nltk.tokenize import TreebankWordTokenizer

>> tokenizer = TreebankWordTokenizer()

>> tokenizer.tokenize(text)



1. **PunktWordTokenizer:**It doen’t seperates the punctuation from the words.

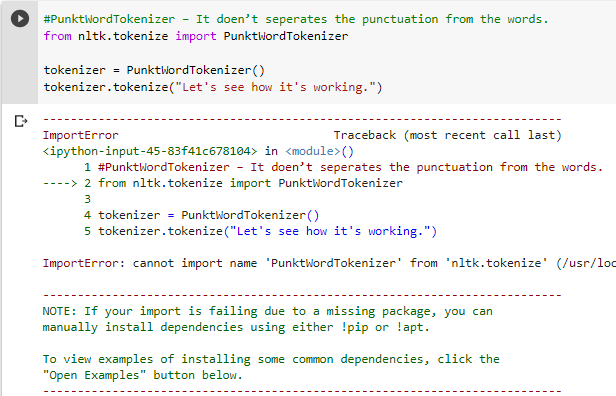
**Code:**

#PunktWordTokenizer – It doen’t seperates the punctuation from the words.

>> from nltk.tokenize import PunktWordTokenizer

>> tokenizer = PunktWordTokenizer()

>> tokenizer.tokenize("Let's see how it's working.")



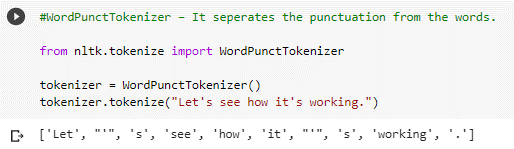
1. **WordPunctTokenizer:**It seperates the punctuation from the words.

#WordPunctTokenizer – It seperates the punctuation from the words.

>> from nltk.tokenize import WordPunctTokenizer

>> tokenizer = WordPunctTokenizer()

>> tokenizer.tokenize("Let's see how it's working.")

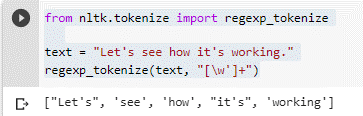


1. **Using Regular Expression**

>> from nltk.tokenize import regexp\_tokenize

>> text = "Let's see how it's working."

>> regexp\_tokenize(text, "[\w']+")



**Practical No. 4: Text Processing – 2**

**Aim: Write a program a text processing - 2**

1. **Accessing Text Corpora**

As just mentioned, a text corpus is a large body of text. Many corpora are designed to contain a careful balance of material in one or more genres. We examined some small text collections in [1.](https://www.nltk.org/book/ch01.html" \l "chap-introduction), such as the speeches known as the US Presidential Inaugural Addresses. This particular corpus actually contains dozens of individual texts — one per address — but for convenience we glued them end-to-end and treated them as a single text. [1.](https://www.nltk.org/book/ch01.html#chap-introduction) also used various pre-defined texts that we accessed by typing from nltk.book import \*. However, since we want to be able to work with other texts, this section examines a variety of text corpora. We'll see how to select individual texts, and how to work with them.

* 1. **Gutenberg Corpus**

>> import nltk

>> nltk.download('gutenberg')

>> nltk.download('genesis')

>> nltk.download('inaugural')

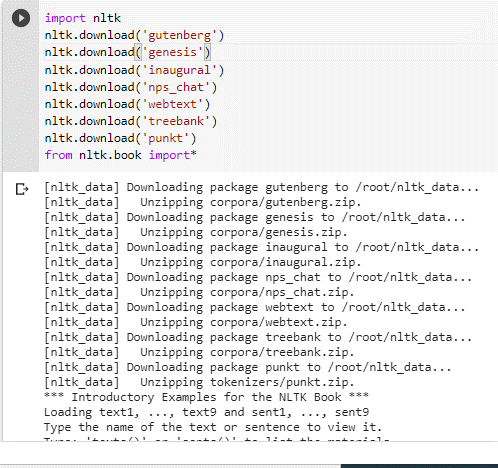
>> nltk.download('nps\_chat')

>> nltk.download('webtext')

>> nltk.download('treebank')

>> nltk.download('punkt')

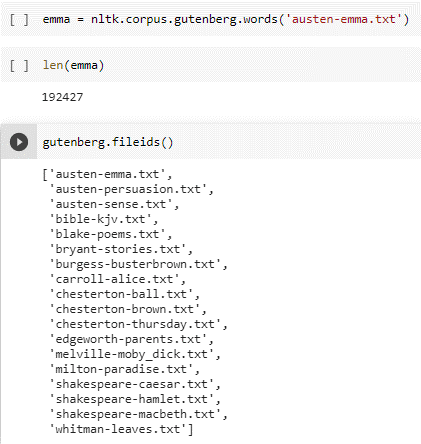
>> from nltk.book import\*



>> emma = nltk.corpus.gutenberg.words('austen-emma.txt')

>> len(emma)

>> gutenberg.fileids()



>> emma = gutenberg.words('austen-emma.txt')

>> for fileid in gutenberg.fileids():

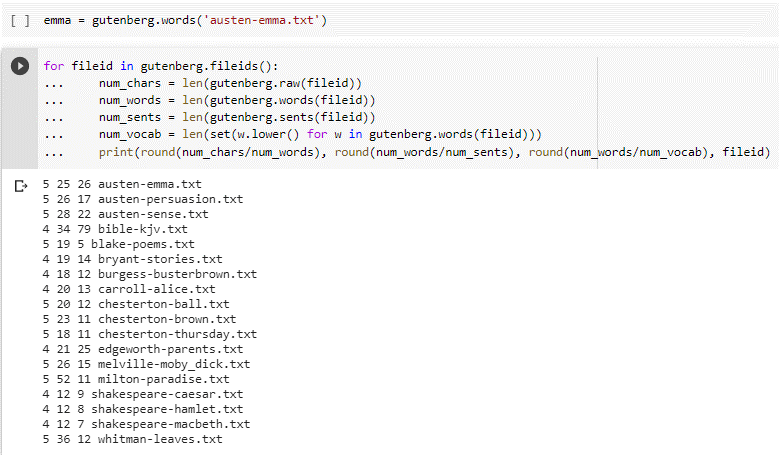
...     num\_chars = len(gutenberg.raw(fileid))

...     num\_words = len(gutenberg.words(fileid))

...     num\_sents = len(gutenberg.sents(fileid))

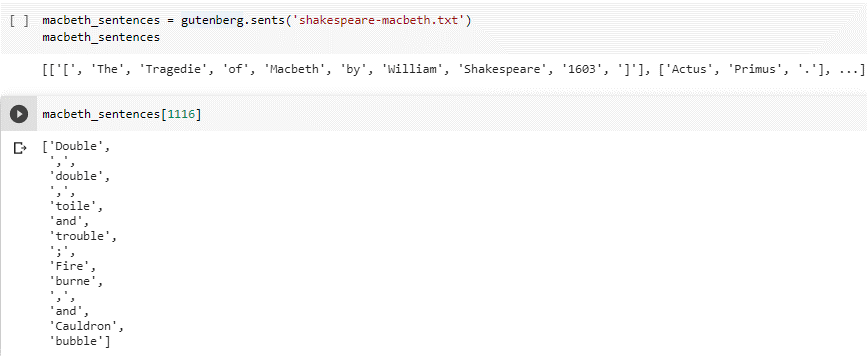
...     num\_vocab = len(set(w.lower() for w in gutenberg.words(fileid)))

...     print(round(num\_chars/num\_words), round(num\_words/num\_sents), round(num\_words/num\_vocab), fileid)



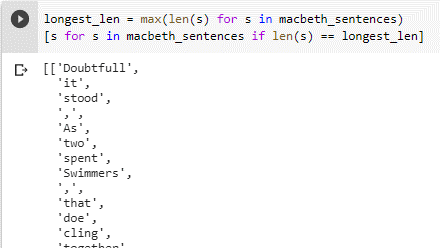
>> macbeth\_sentences = gutenberg.sents('shakespeare-macbeth.txt')

>> macbeth\_sentences

>> macbeth\_sentences[1116]

>> longest\_len = max(len(s) for s in macbeth\_sentences)

>> [s for s in macbeth\_sentences if len(s) == longest\_len]



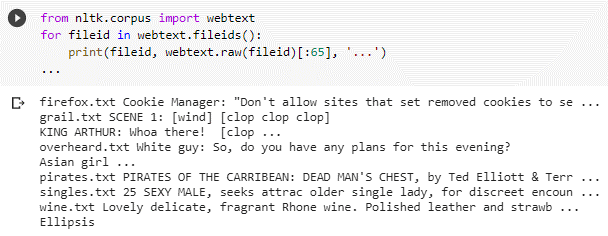
* 1. **Web and Chat Text**

>> from nltk.corpus import webtext

>> for fileid in webtext.fileids():

    print(fileid, webtext.raw(fileid)[:65], '...')

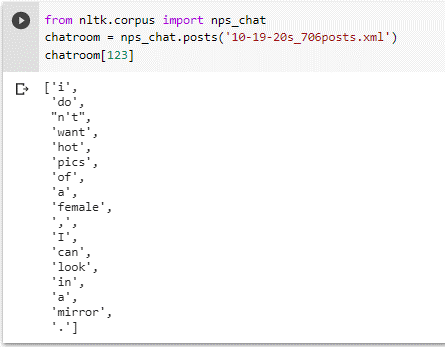
...



>> from nltk.corpus import nps\_chat

>> chatroom = nps\_chat.posts('10-19-20s\_706posts.xml')

>> chatroom[123]

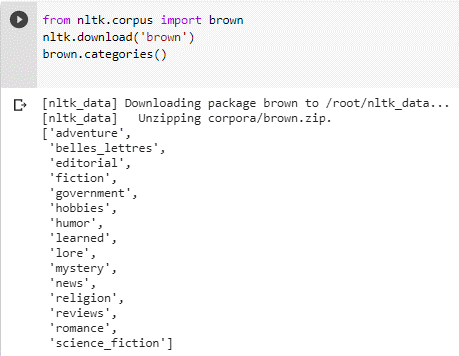


* 1. **Brown Corpus**

>> from nltk.corpus import brown

>> nltk.download('brown')

>> brown.categories()





* 1. **Annotated Text Corpora**
  2. **Corpora in other languages**

>> nltk.download('cess\_esp')

>> nltk.download('floresta')

>> nltk.download('indian')

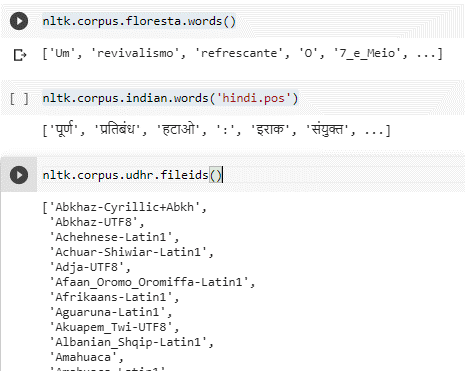
>> nltk.corpus.cess\_esp.words()

>> nltk.download('udhr')

>> nltk.corpus.floresta.words()

>> nltk.corpus.indian.words('hindi.pos')

>> nltk.corpus.udhr.fileids()



>> nltk.corpus.udhr.words('Javanese-Latin1')[11:]

>> from nltk.corpus import udhr

>>languages = ['Chickasaw', 'English', 'German\_Deutsch','Greenlandic\_Inuktikut', 'Hungarian\_Magyar', 'Ibibio\_Efik']

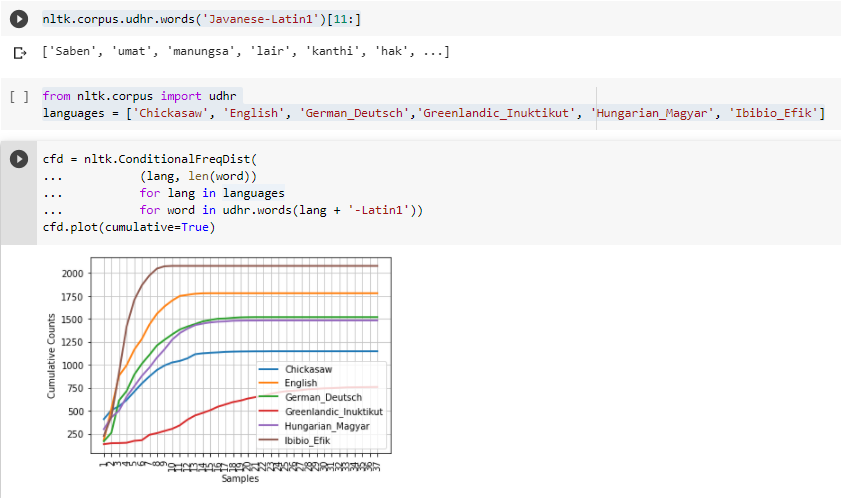
>> cfd = nltk.ConditionalFreqDist(

...           (lang, len(word))

...           for lang in languages

...           for word in udhr.words(lang + '-Latin1'))

>> cfd.plot(cumulative=True)



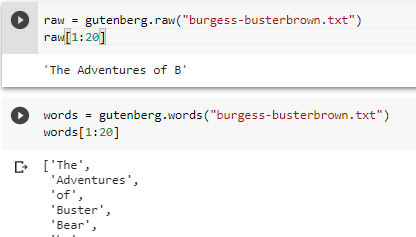
* 1. **Text Corpus Structure**

>> raw = gutenberg.raw("burgess-busterbrown.txt")

>> raw[1:20]

>> words = gutenberg.words("burgess-busterbrown.txt")

>> words[1:20]



* 1. **Loading your own corpus**

1. **Word Counting**

After tokenising a text, the first figure we can calculate is the word frequency. By word frequency we indicate the number of times each token occurs in a text. When talking about word frequency, we distinguish between types and tokens. Types are the distinct words in a corpus, whereas tokens are the words, including repeats. Let's see how this works in practice.

Let's take as example one of the sentences below:

Types are the distinct words in a corpus, whereas tokens are the running words.

How many types and tokens are there in the above sentence? [Answer](javascript:%20void(0);)

Let's see how we can use Python to calculate these figures. First of all, let's tokenise the sentence by using a tokeniser which uses non-alphabetical characters as a separator.

**Code:**

>> from nltk.tokenize.regexp import WhitespaceTokenizer

>>my\_str = "Types are the distinct words in a corpus, whereas tokens are the running words."

>> tokens = WhitespaceTokenizer().tokenize(my\_str)

>> print (len(tokens))

>> print (len(tokens))

>> print (len(tokens))

>>my\_str = "Types are the distinct words in a corpus, whereas tokens are the running words."

>> from nltk.tokenize.regexp import WordPunctTokenizer

>> my\_toks = WordPunctTokenizer().tokenize(my\_str)

>> print (len(my\_toks))

>> my\_vocab = set(my\_toks)

>> print (len(my\_vocab))

1. **Word Vocabulary**

The vocabulary serves a few primary purposes:

* help in the preprocessing of the corpus text
* serve as storage location in memory for processed text corpus
* collect and store metadata about the corpus
* allow for pre-task munging, exploration, and experimentation

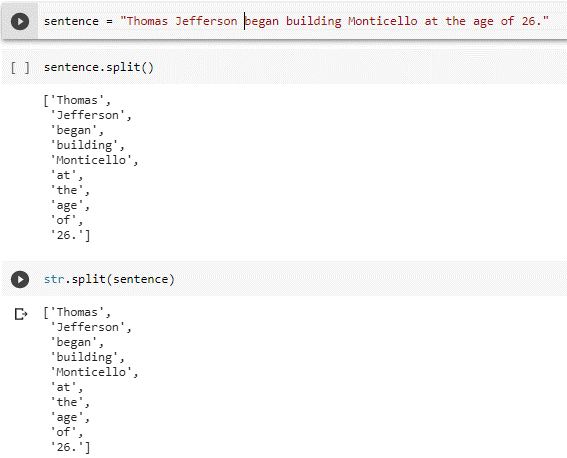
The vocabulary serves a few related purposes and can be thought of in a few different ways, but the main takeaway is that, once a corpus has made its way to the vocabulary, the text has been processed and any relevant metadata should be collected and stored.

**Code:**

>> sentence = "Thomas Jefferson began building Monticello at the age of 26."

>> sentence.split()

>> str.split(sentence)



1. **Bag-of-words, BOW Model**

The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms.

The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification.

In this tutorial, you will discover the bag-of-words model for feature extraction in [natural language processing](https://machinelearningmastery.com/natural-language-processing/).

After completing this tutorial, you will know:

* What the bag-of-words model is and why it is needed to represent text.
* How to develop a bag-of-words model for a collection of documents.
* How to use different techniques to prepare a vocabulary and score words.

**Code:**

>> import pandas as pd

>> from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

>> sentence\_1="This is a good job.I will not miss it for anything"

>> sentence\_2="This is not good at all"

>>CountVec = CountVectorizer(ngram\_range=(1,1), # to use bigrams ngram\_range=(2,2)

                           stop\_words='english')

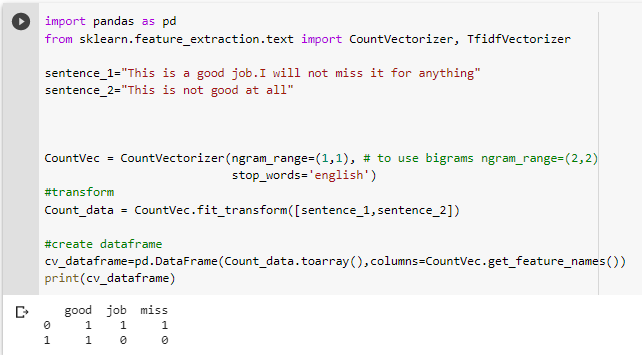
#transform

>> Count\_data = CountVec.fit\_transform([sentence\_1,sentence\_2])

#create dataframe

>>cv\_dataframe=pd.DataFrame(Count\_data.toarray(),columns=CountVec.get\_feature\_names())

>> print(cv\_dataframe)



1. **TF/IDF Vectorizer**

TF-IDF stands for term frequency-inverse document frequency. TF-IDF weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

* **Term Frequency (TF)**: is a scoring of the frequency of the word in the current document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. The term frequency is often divided by the document length to normalize.



* **Inverse Document Frequency (IDF)**: is a scoring of how rare the word is across documents. IDF is a measure of how rare a term is. Rarer the term, more is the IDF score.



**Code:**

>> import pandas as pd

>> from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

>> sentence\_1="This is a good job.I will not miss it for anything"

>> sentence\_2="This is not good at all"

#without smooth IDF

>> print("Without Smoothing:")

#define tf-idf

>> tf\_idf\_vec = TfidfVectorizer(use\_idf=True,

                        smooth\_idf=False,

                        ngram\_range=(1,1),stop\_words='english') # to use only  bigrams ngram\_range=(2,2)

#transform

>> tf\_idf\_data = tf\_idf\_vec.fit\_transform([sentence\_1,sentence\_2])

#create dataframe

>>tf\_idf\_dataframe=pd.DataFrame(tf\_idf\_data.toarray(),columns=tf\_idf\_vec.get\_feature\_names())

>> print(tf\_idf\_dataframe)

>> print("\n")

#with smooth

>> tf\_idf\_vec\_smooth = TfidfVectorizer(use\_idf=True,

                        smooth\_idf=True,

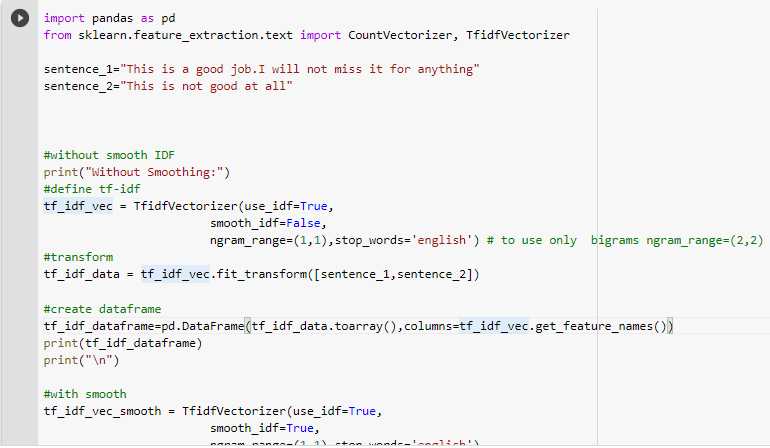
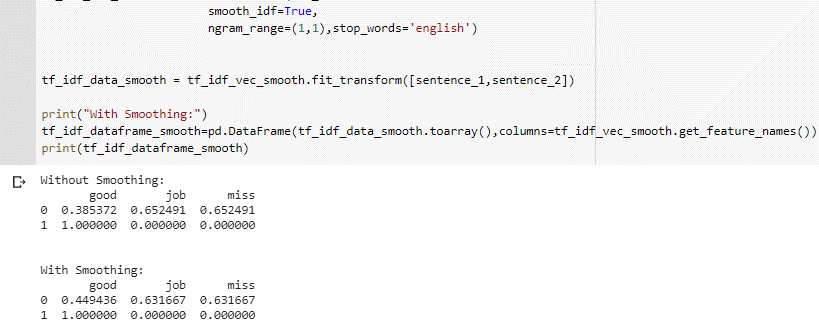
                        ngram\_range=(1,1),stop\_words='english')

>>tf\_idf\_data\_smooth = tf\_idf\_vec\_smooth.fit\_transform([sentence\_1,sentence\_2])

>> print("With Smoothing:")

>>tf\_idf\_dataframe\_smooth=pd.DataFrame(tf\_idf\_data\_smooth.toarray(),columns=tf\_idf\_vec\_smooth.get\_feature\_names())

>> print(tf\_idf\_dataframe\_smooth)



1. **Tokenisation and Word Frequencies**

One of the key steps in NLP or Natural Language Process is the ability to count the frequency of the terms used in a text document or table. To achieve this we must tokenize the words so that they represent individual objects that can be counted. There are a great set of libraries that you can use to tokenize words. However the most popular Python library is NLTK or Natural Language Tool Kit.

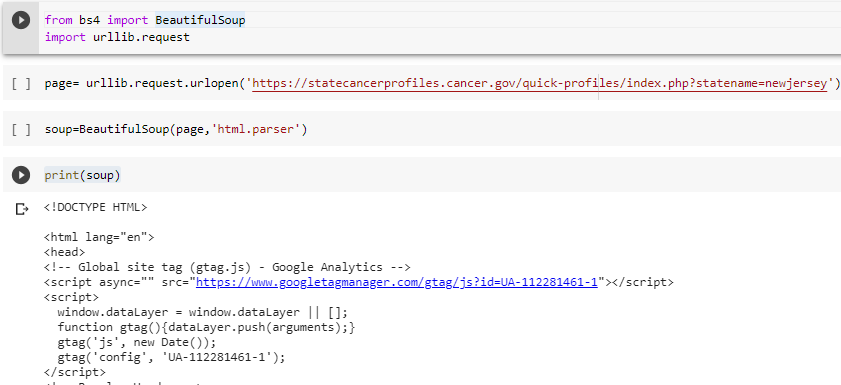
**Code:**

>> from bs4 import BeautifulSoup

>> import urllib.request

>> page= urllib.request.urlopen('https://statecancerprofiles.cancer.gov/quick-profiles/index.php?statename=newjersey')

>> soup=BeautifulSoup(page,'html.parser')

>> print(soup)

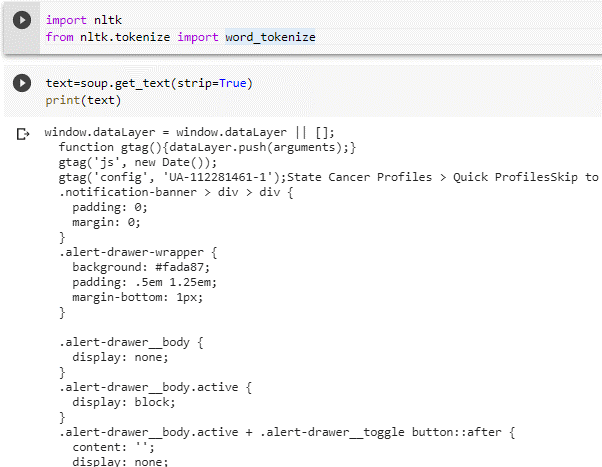
>> import nltk

>> from nltk.tokenize import word\_tokenize

#@title Default title text

>> text=soup.get\_text(strip=True)

>> print(text)



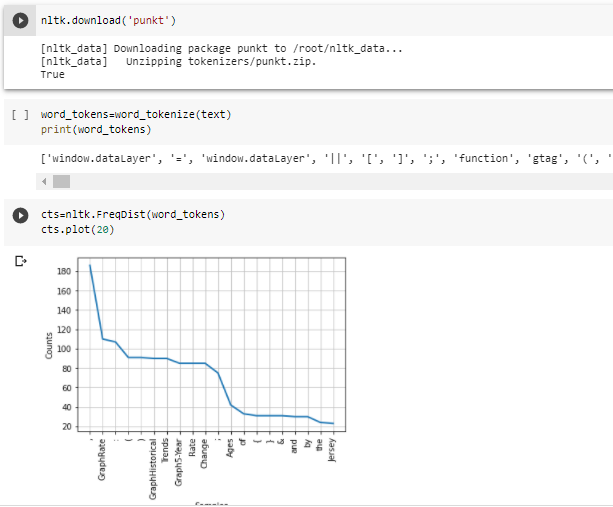
>> nltk.download('punkt')

>> word\_tokens=word\_tokenize(text)

>> print(word\_tokens)

>> cts=nltk.FreqDist(word\_tokens)

>> cts.plot(20)



1. **Sentence Segmentation**

Sentence tokenization (also called **sentence segmentation**) is the problem of **dividing a string** of written language **into** its component **sentences**. The idea here looks very simple. In English and some other languages, we can split apart the sentences whenever we see a punctuation mark.

**Code:**

#import spacy library

>> import spacy

#load core english library

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

#take unicode string

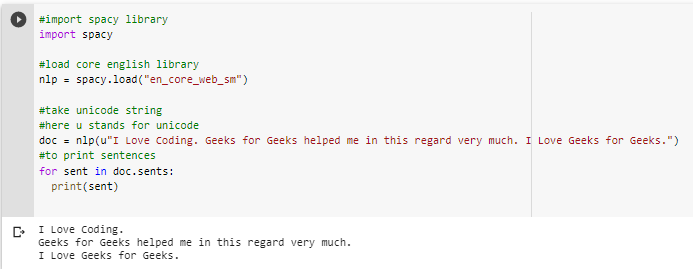
#here u stands for unicode

>>doc = nlp(u"I Love Coding. Geeks for Geeks helped me in this regard very much. I Love Geeks for Geeks.")

#to print sentences

>> for sent in doc.sents:

>> print(sent)



>> stanza.download('en')

>> import stanza

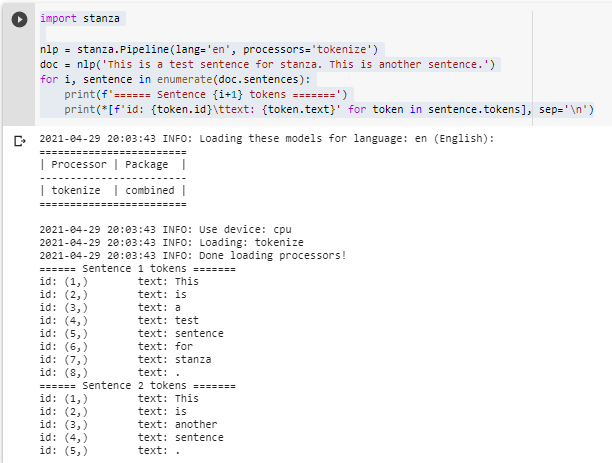
>> nlp = stanza.Pipeline(lang='en', processors='tokenize')

>> doc = nlp('This is a test sentence for stanza. This is another sentence.')

>> for i, sentence in enumerate(doc.sentences):

    print(f'====== Sentence {i+1} tokens =======')

    print(\*[f'id: {token.id}\ttext: {token.text}' for token in sentence.tokens], sep='\n')



1. **Removing Stop Words with NLTK**

The [NLTK](https://www.nltk.org/) library is one of the oldest and most commonly used Python libraries for Natural Language Processing. NLTK supports stop word removal, and you can find the list of stop words in the corpus module. To remove stop words from a sentence, you can divide your text into words and then remove the word if it exits in the list of stop words provided by NLTK.

**Code:**

>> from nltk.corpus import stopwords

>> nltk.download('stopwords')

>> print(stopwords.words("english"))

#Let’s see how we can remove the stop words from a sentence.

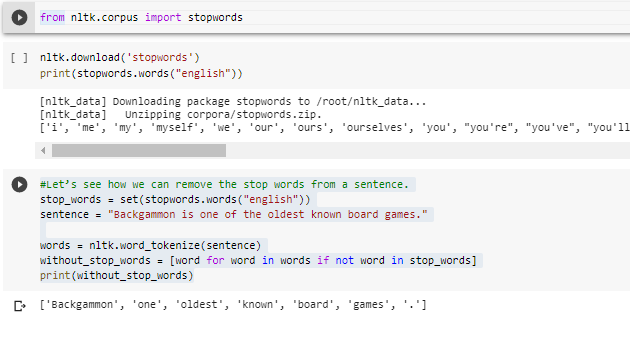
>> stop\_words = set(stopwords.words("english"))

>> sentence = "Backgammon is one of the oldest known board games."

>> words = nltk.word\_tokenize(sentence)

>> without\_stop\_words = [word for word in words if not word in stop\_words]

>> print(without\_stop\_words)



**Practical No. 5: Text Processing**

**Aim: Write a program to Text Processing.**

Whenever we have textual data, we need to apply several pre-processing steps to the data to transform words into numerical features that work with machine learning algorithms. The pre-processing steps for a problem depend mainly on the domain and the problem itself, hence, we don’t need to apply all steps to every problem.

In this article, we are going to see text preprocessing in Python. We will be using the NLTK (Natural Language Toolkit) library here.

**Code:**

# import the necessary libraries

>> import nltk

>> import string

>> import re

1. **Text Lowercase :** We lowercase the text to reduce the size of the vocabulary of our text data.

**Code:**

>> def text\_lowercase(text):

>> return text.lower()

>>input\_str = "Hey, did you know that the summer break is coming? Amazing >> right !! It's only 5 more days !!"

>> text\_lowercase(input\_str)

1. **Remove Numbers :** We can either remove numbers or convert the numbers into their textual representations.  
   We can use regular expressions to remove the numbers.

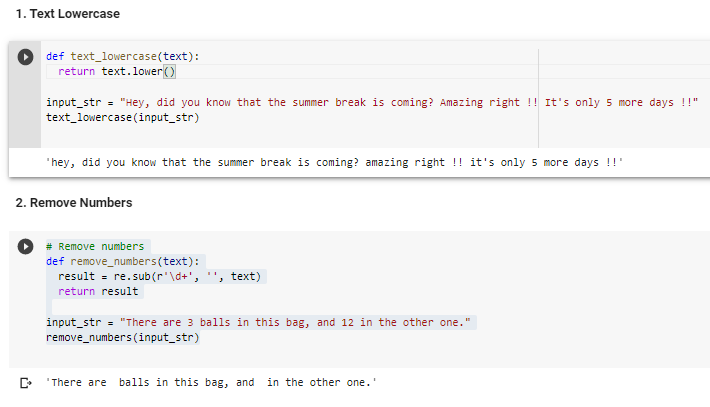
# Remove numbers

def remove\_numbers(text):

result = re.sub(r'\d+', '', text)

  return result

input\_str = "There are 3 balls in this bag, and 12 in the other one."

remove\_numbers(input\_str)

1. **Remove Punctuation:** We remove punctuations so that we don’t have different forms of the same word. If we don’t remove the punctuation, then been. been, been! will be treated separately.

**Code:**

# remove punctuation

>> def remove\_punctuation(text):

  >> translator = str.maketrans('', '', string.punctuation)

  >> return text.translate(translator)

>>input\_str = "Hey, did you know that the summer break is coming? Amazing right !! It's only 5 more days !!"

>> remove\_punctuation(input\_str)

1. **Remove Whitespaces:** We can use the join and split function to remove all the white spaces in a string.

**Code:**

# remove whitespace from text

>> def remove\_whitespace(text):

  return " ".join(text.split())

>>input\_str = "     we         don't      need      the       given        questions"

>> remove\_whitespace(input\_str)

1. **Remove Default Stopwords**

Stopwords are words that do not contribute to the meaning of a sentence. Hence, they can safely be removed without causing any change in the meaning of the sentence. The NLTK library has a set of stopwords and we can use these to remove stopwords from our text and return a list of word tokens.

**Code:**

>> nltk.download('stopwords')

>> nltk.download('punkt')

>> from nltk.corpus import stopwords

>> from nltk.tokenize import word\_tokenize

# remove stopwords function

>> def remove\_stopwords(text):

>> stop\_words = set(stopwords.words("english"))

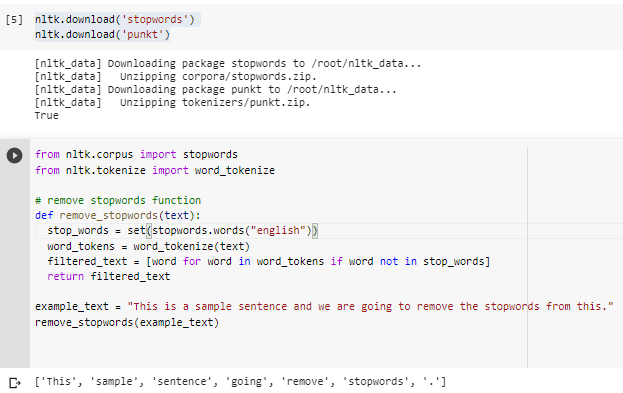
>> word\_tokens = word\_tokenize(text)

>>filtered\_text = [word for word in word\_tokens if word not in stop\_words]

>> return filtered\_text

>>example\_text = "This is a sample sentence and we are going to remove the stopwords from this."

>> remove\_stopwords(example\_text)



1. **Stemming**

Stemming is the process of getting the root form of a word. Stem or root is the part to which inflectional affixes (-ed, -ize, -de, -s, etc.) are added. The stem of a word is created by removing the prefix or suffix of a word. So, stemming a word may not result in actual words.

**Code:**

>> from nltk.stem.porter import PorterStemmer

>> from nltk.tokenize import word\_tokenize

>> stemmer = PorterStemmer()

# stem words in the list of tokenised words

>> def stem\_words(text):

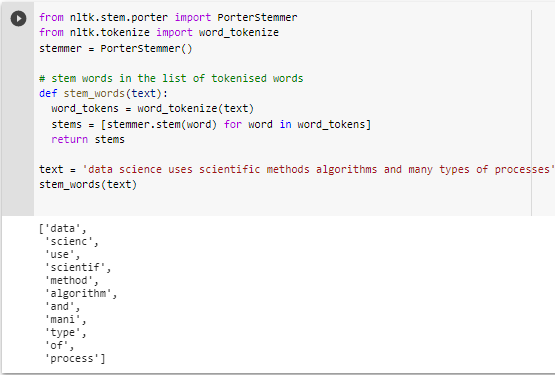
>> word\_tokens = word\_tokenize(text)

>> stems = [stemmer.stem(word) for word in word\_tokens]

>> return stems

>>text = 'data science uses scientific methods algorithms and many types of processes'

>> stem\_words(text)



1. **Lemmatization**

Like stemming, lemmatization also converts a word to its root form. The only difference is that lemmatization ensures that the root word belongs to the language. We will get valid words if we use lemmatization. In NLTK, we use the WordNetLemmatizer to get the lemmas of words. We also need to provide a context for the lemmatization. So, we add the part-of-speech as a parameter.

**Code:**

>> nltk.download('wordnet')

>> from nltk.stem import WordNetLemmatizer

>> from nltk.tokenize import word\_tokenize

>> lemmatizer = WordNetLemmatizer()

# lemmatize string

>> def lemmatize\_word(text):

>> word\_tokens = word\_tokenize(text)

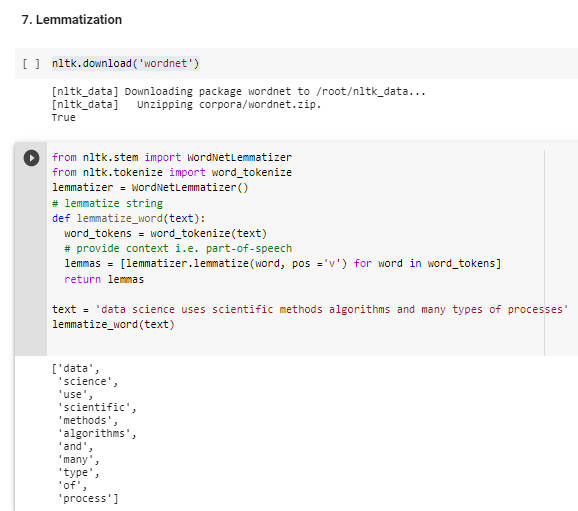
# provide context i.e. part-of-speech

>> lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word\_tokens]

>> return lemmas

>>text = 'data science uses scientific methods algorithms and many types of processes'

>> lemmatize\_word(text)



**Practical No. 6: Phonetic Hashing Using Soundex Algorithm**

**Aim: Write a program for Phonetic Hashing Using Soundex Algorithm**

**Description:**

Phonetic hashing buckets all the similar phonemes (words with similar sound or pronunciation) into a single bucket and gives all these variations a single hash code. Hence, the word ‘Dilli’ and ‘Delhi’ will have the same code.

Phonetic hashing is done using the Soundex algorithm. It doesn’t matter which language the input word comes from — as long as the words sound similar, they will get the same hash code.

Now, let’s see it through an example. The Soundex of the word ‘Mississippi’. To calculate the hash code, following are the steps:Phonetic hashing is a four-letter code. The first letter of the code is the first letter of the input word. Hence it is retained as is. The first character of the phonetic hash is ‘M’. Now, we need to make changes to the rest of the letters of the word.Now, we need to map all the consonant letters (except the first letter). All the vowels are written as is and ‘H’s, ‘Y’s and ‘W’s remain unencoded (unencoded means they are removed from the word). After mapping the consonants, the code becomes MI22I22I11I.The third step is to remove all the vowels. ‘I’ is the only vowel. After removing all the ‘I’s, we get the code M222211. Now, you would need to merge all the consecutive duplicate numbers into a single unique number. All the ‘2’s are merged into a single ‘2’. Similarly, all the ‘1’s are merged into a single ‘1’. The code that we get is M21.The fourth step is to force the code to make it a four-letter code. You either need to pad it with zeroes in case it is less than four characters in length. Or you need to truncate it from the right side in case it is more than four characters in length. Since the code is less than four characters in length, you’ll pad it with one ‘0’ at the end. The final code is M210.

**Code:**

>> import numpy as np

>> import pandas as pd

#Visualization Libraries

>> import seaborn as sns

>> import matplotlib.pyplot as plt

#imports from sklearn library

>> from sklearn import datasets

>> from sklearn.linear\_model import LinearRegression

>> from sklearn.model\_selection import train\_test\_split, cross\_val\_score

>> from sklearn.metrics import mean\_squared\_error

#loading the dataset direclty from sklearn

>> boston = datasets.load\_boston()

>> print('\n')

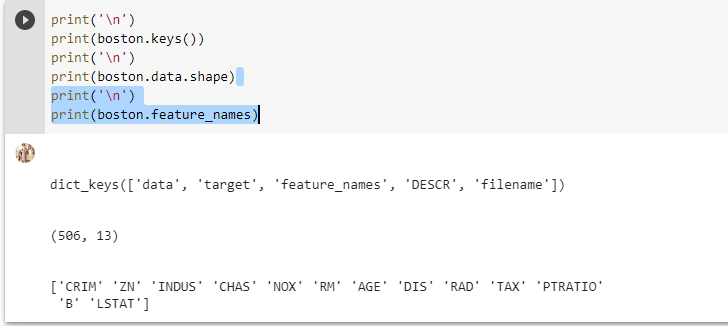
>> print(boston.keys())

>> print('\n')

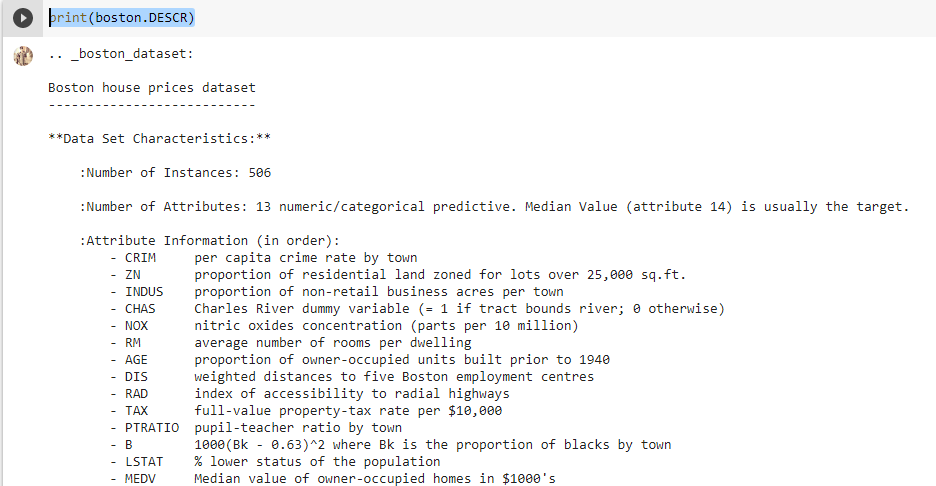
>> print(boston.data.shape)

>> print('\n')

>> print(boston.feature\_names)



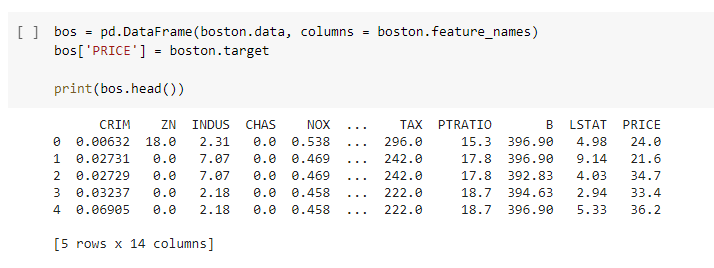
>> print(boston.DESCR)



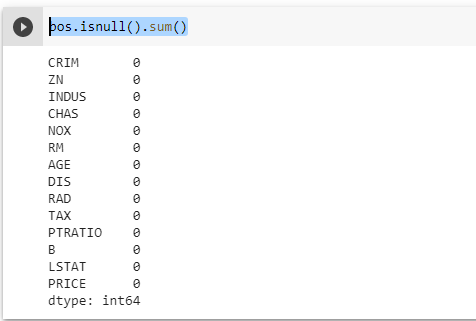
>> bos = pd.DataFrame(boston.data, columns = boston.feature\_names)

>> bos['PRICE'] = boston.target

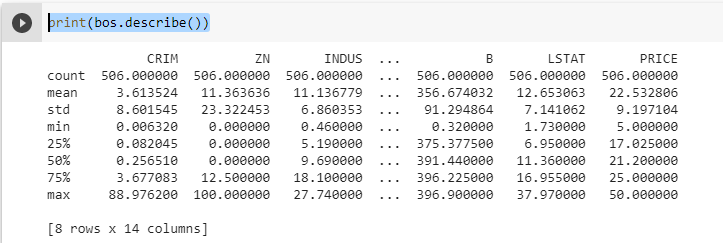
>> print(bos.head())



>> bos.isnull().sum()



>> print(bos.describe())

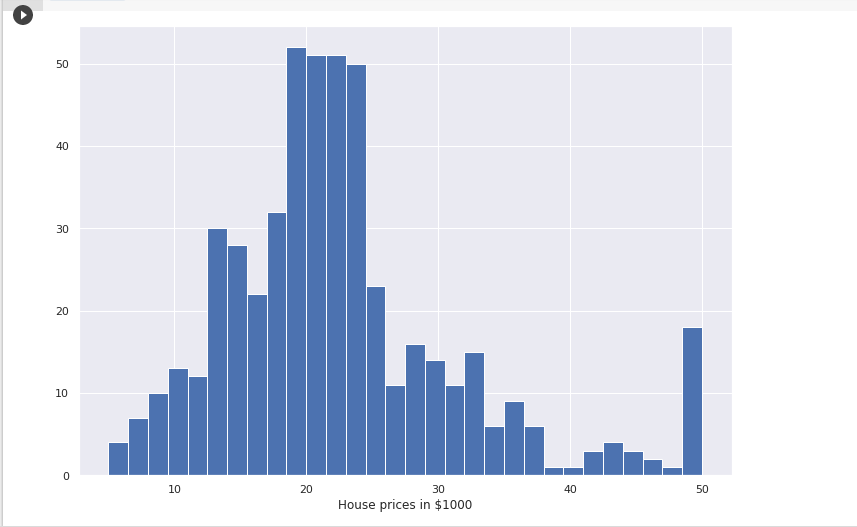


>> sns.set(rc={'figure.figsize':(11.7,8.27)})

>> plt.hist(bos['PRICE'], bins=30)

>> plt.xlabel("House prices in $1000")

>> plt.show()

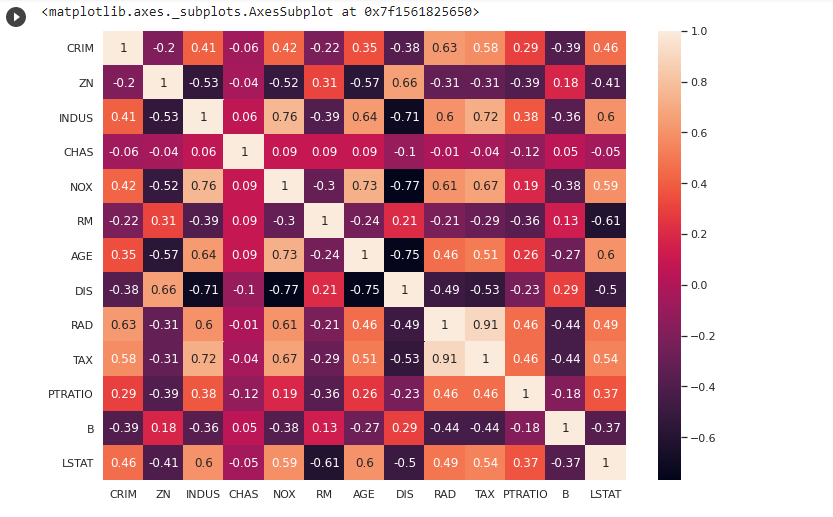


#Created a dataframe without the price col, since we need to see the correlation between the variables

>> bos\_1 = pd.DataFrame(boston.data, columns = boston.feature\_names)

>> correlation\_matrix = bos\_1.corr().round(2)

>> sns.heatmap(data=correlation\_matrix, annot=True)



>> plt.figure(figsize=(20, 5))

>> features = ['LSTAT', 'RM']

>> target = bos['PRICE']

>> for i, col in enumerate(features):

    plt.subplot(1, len(features) , i+1)

    x = bos[col]

    y = target

    plt.scatter(x, y, marker='o')

    plt.title("Variation in House prices")

    plt.xlabel(col)

    plt.ylabel('"House prices in $1000"')



>> X\_rooms = bos.RM

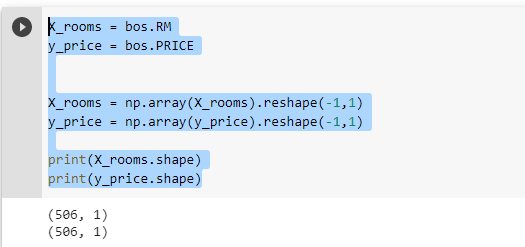
>> y\_price = bos.PRICE

>> X\_rooms = np.array(X\_rooms).reshape(-1,1)

>> y\_price = np.array(y\_price).reshape(-1,1)

>> print(X\_rooms.shape)

>> print(y\_price.shape)



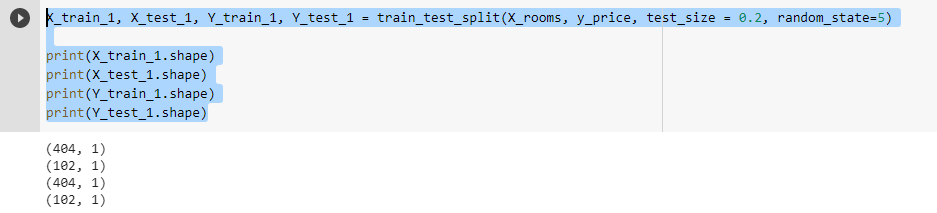
>> X\_train\_1, X\_test\_1, Y\_train\_1, Y\_test\_1 = train\_test\_split(X\_rooms, y\_price, test\_size = 0.2, random\_state=5)

>> print(X\_train\_1.shape)

>> print(X\_test\_1.shape)

>> print(Y\_train\_1.shape)

>> print(Y\_test\_1.shape)



>> reg\_1 = LinearRegression()

>> reg\_1.fit(X\_train\_1, Y\_train\_1)

>> y\_train\_predict\_1 = reg\_1.predict(X\_train\_1)

>> rmse = (np.sqrt(mean\_squared\_error(Y\_train\_1, y\_train\_predict\_1)))

>> r2 = round(reg\_1.score(X\_train\_1, Y\_train\_1),2)

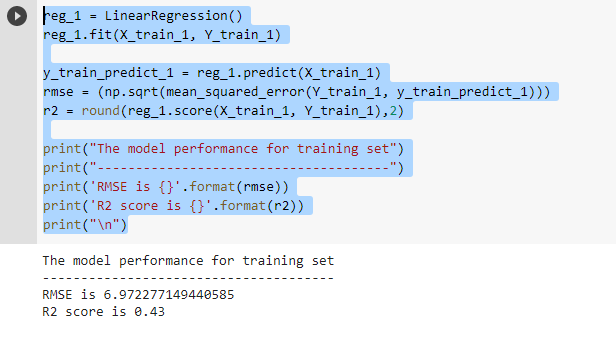
>> print("The model performance for training set")

>> print("--------------------------------------")

>> print('RMSE is {}'.format(rmse))

>> print('R2 score is {}'.format(r2))

>> print("\n")



>> y\_pred\_1 = reg\_1.predict(X\_test\_1)

>> rmse = (np.sqrt(mean\_squared\_error(Y\_test\_1, y\_pred\_1)))

>> r2 = round(reg\_1.score(X\_test\_1, Y\_test\_1),2)

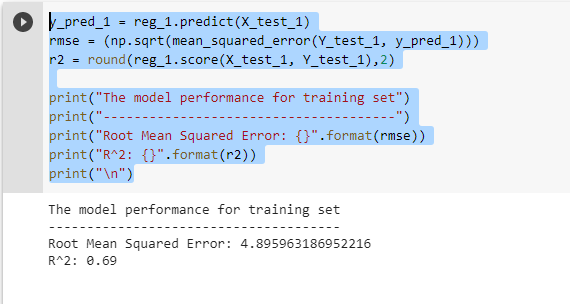
>> print("The model performance for training set")

>> print("--------------------------------------")

>> print("Root Mean Squared Error: {}".format(rmse))

>> print("R^2: {}".format(r2))

>> print("\n")



>> prediction\_space = np.linspace(min(X\_rooms), max(X\_rooms)).reshape(-1,1)

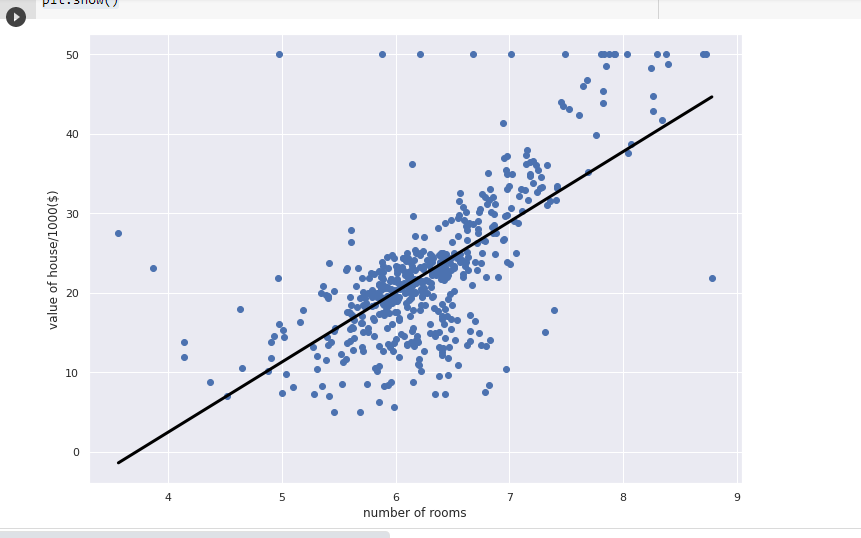
>> plt.scatter(X\_rooms,y\_price)

>> plt.plot(prediction\_space, reg\_1.predict(prediction\_space), color = 'black', linewidth = 3)

>> plt.ylabel('value of house/1000($)')

>> plt.xlabel('number of rooms')

>> plt.show()



>> X = bos.drop('PRICE', axis = 1)

>> y = bos['PRICE']

>> X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2, random\_state=42)

>> reg\_all = LinearRegression()

>> reg\_all.fit(X\_train, y\_train)

# model evaluation for training set

>> y\_train\_predict = reg\_all.predict(X\_train)

>> rmse = (np.sqrt(mean\_squared\_error(y\_train, y\_train\_predict)))

>> r2 = round(reg\_all.score(X\_train, y\_train),2)

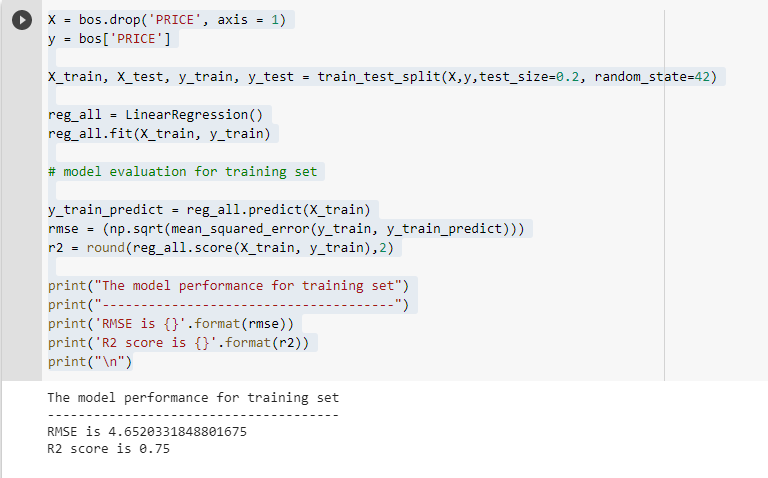
>> print("The model performance for training set")

>> print("--------------------------------------")

>> print('RMSE is {}'.format(rmse))

>> print('R2 score is {}'.format(r2))

>> print("\n")



>> y\_pred = reg\_all.predict(X\_test)

>> rmse = (np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

>> r2 = round(reg\_all.score(X\_test, y\_test),2)

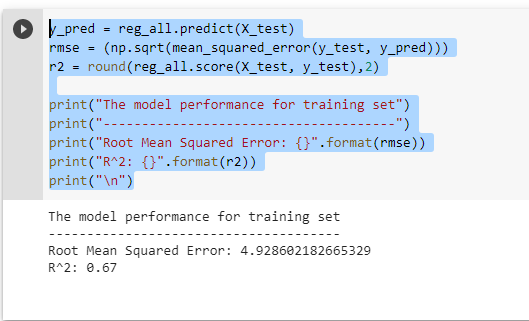
>> print("The model performance for training set")

>> print("--------------------------------------")

>> print("Root Mean Squared Error: {}".format(rmse))

>> print("R^2: {}".format(r2))

>> print("\n")



>> plt.scatter(y\_test, y\_pred)

>> plt.xlabel("Actual House Prices ($1000)")

>> plt.ylabel("Predicted House Prices: ($1000)")

>> plt.xticks(range(0, int(max(y\_test)),2))

>> plt.yticks(range(0, int(max(y\_test)),2))

>> plt.title("Actual Prices vs Predicted prices")



**Practical No. 7: Information Extraction**

**Aim: Write a program for Information extraction.**

**Code:**

# import the necessary libraries

>> import nltk

>> import string

>> import re

1. **Part-of-Speech Tagging**

The part of speech explains how a word is used in a sentence. In a sentence, a word can have different contexts and semantic meanings. The basic natural language processing models like bag-of-words fail to identify these relations between words. Hence, we use part of speech tagging to mark a word to its part of speech tag based on its context in the data. It is also used to extract relationships between words.

**Code:**

>> nltk.download('punkt')

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

>> from nltk.tokenize import word\_tokenize

>> from nltk import pos\_tag

# convert text into word\_tokens with their tags

>> def pos\_tagging(text):

>>  word\_tokens = word\_tokenize(text)

>>  return pos\_tag(word\_tokens)

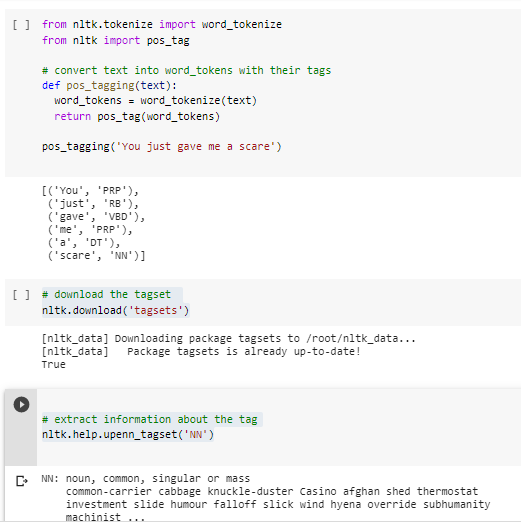
>> pos\_tagging('You just gave me a scare')

# download the tagset

>> nltk.download('tagsets')

# extract information about the tag

>> nltk.help.upenn\_tagset('NN')



1. **Chunking**

Chunking is the process of extracting phrases from unstructured text and more structure to it. It is also known as shallow parsing. It is done on top of Part of Speech tagging. It groups word into “chunks”, mainly of noun phrases. Chunking is done using regular expressions.

**Code:**

# Laading Library

>> from nltk.chunk.regexp import tag\_pattern2re\_pattern

# Chunk Pattern to RegEx Pattern

>> print("Chunk Pattern : ", tag\_pattern2re\_pattern('<DT>?<NN.\*>+'))

>> locs = [('Omnicom', 'IN', 'New York'),

...         ('DDB Needham', 'IN', 'New York'),

...         ('Kaplan Thaler Group', 'IN', 'New York'),

...         ('BBDO South', 'IN', 'Atlanta'),

...         ('Georgia-Pacific', 'IN', 'Atlanta')]

>> query = [e1 for (e1, rel, e2) in locs if e2=='Atlanta']

>> print(query)

>> def ie\_preprocess(document):

...    sentences = nltk.sent\_tokenize(document)

...    sentences = [nltk.word\_tokenize(sent) for sent in sentences]

...    sentences = [nltk.pos\_tag(sent) for sent in sentences]

>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),

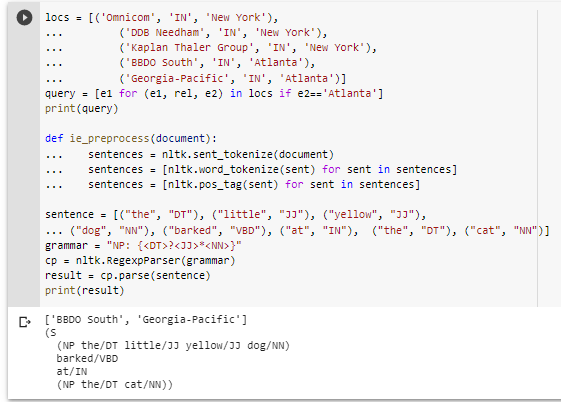
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]

>> grammar = "NP: {<DT>?<JJ>\*<NN>}"

>> cp = nltk.RegexpParser(grammar)

>> result = cp.parse(sentence)

>> print(result)



1. **Chinking**

Chinking is a lot like chunking, it is basically a way for you to remove a chunk from a chunk. The chunk that you remove from your chunk is your chink. The code is very similar, you just denote the chink, after the chunk, with }{ instead of the chunk's {}.

**Code:**

>> grammar = r"""

NP: {<DT|PP\$>?<JJ>\*<NN>}

       {<NNP>+}

"""

>> cp = nltk.RegexpParser(grammar)

>> sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"),

                 ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"), ("hair", "NN")]

>> print(cp.parse(sentence))

>> nouns = [("money", "NN"), ("market", "NN"), ("fund", "NN")]

>> grammar = "NP: {<NN><NN>}  # Chunk two consecutive nouns"

>> cp = nltk.RegexpParser(grammar)

>> print(cp.parse(nouns))



1. **Named Entity Recognition**

Named Entity Recognition is used to extract information from unstructured text. It is used to classify entities present in a text into categories like a person, organization, event, places, etc. It gives us detailed knowledge about the text and the relationships between the different entities.

**Code:**

>> nltk.download('words')

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

>> from nltk.tokenize import word\_tokenize

>> from nltk import pos\_tag, ne\_chunk

>> def named\_entity\_recognition(text):

  # tokenize the text

  >> word\_tokens = word\_tokenize(text)

  # part of speech tagging of words

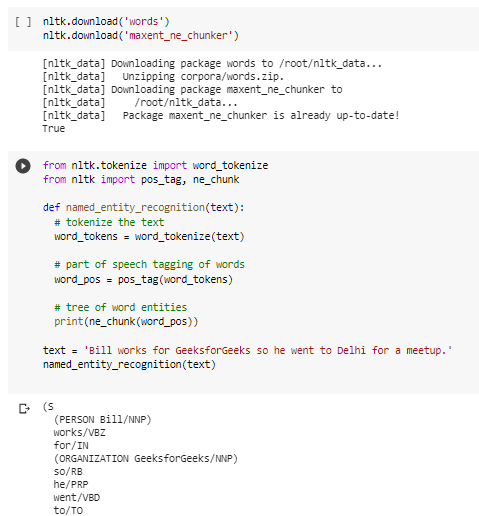
  >> word\_pos = pos\_tag(word\_tokens)

  # tree of word entities

  >> print(ne\_chunk(word\_pos))

>>text = 'Bill works for GeeksforGeeks so he went to Delhi for a meetup.'

>> named\_entity\_recognition(text)



1. **Relation Extraction**

Once named entities have been identified in a text, we then want to extract the relations that exist between them. As indicated earlier, we will typically be looking for relations between specified types of named entity. One way of approaching this task is to initially look for all triples of the form (X, α, Y), where X and Y are named entities of the required types, and α is the string of words that intervenes between X and Y. We can then use regular expressions to pull out just those instances of α that express the relation that we are looking for. The following example searches for strings that contain the word in. The special regular expression (?!\b.+ing\b) is a negative lookahead assertion that allows us to disregard strings such as success in supervising the transition of, where in is followed by a gerund.

**Code:**

>> nltk.download('ieer')

>> import nltk

>> IN = re.compile(r'.\*\bin\b(?!\b.+ing)')

>> for doc in nltk.corpus.ieer.parsed\_docs('NYT\_19980315'):

>> for rel in nltk.sem.extract\_rels('ORG', 'LOC', doc,

                         corpus='ieer', pattern = IN):

>> print(nltk.sem.rtuple(rel))

>> nltk.download('conll2002')

>> from nltk.corpus import conll2002

>> vnv = """

(

is/V|    # 3rd sing present and

was/V|   # past forms of the verb zijn ('be')

werd/V|  # and also present

wordt/V  # past of worden ('become)

)

 .\*       # followed by anything

van/Prep # followed by van ('of')

"""

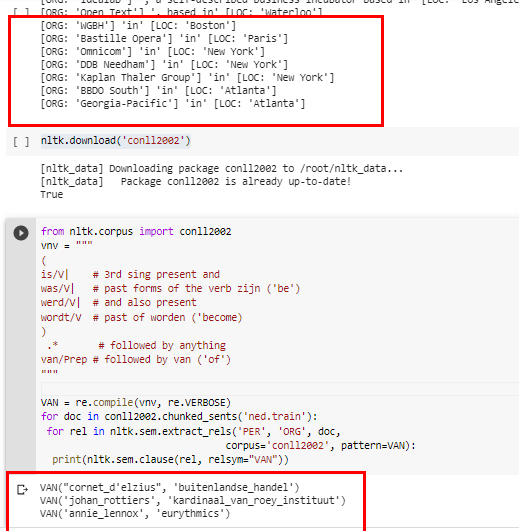
>> VAN = re.compile(vnv, re.VERBOSE)

>> for doc in conll2002.chunked\_sents('ned.train'):

>>for rel in nltk.sem.extract\_rels('PER', 'ORG', doc,

>> corpus='conll2002', pattern=VAN):

>> print(nltk.sem.clause(rel, relsym="VAN"))



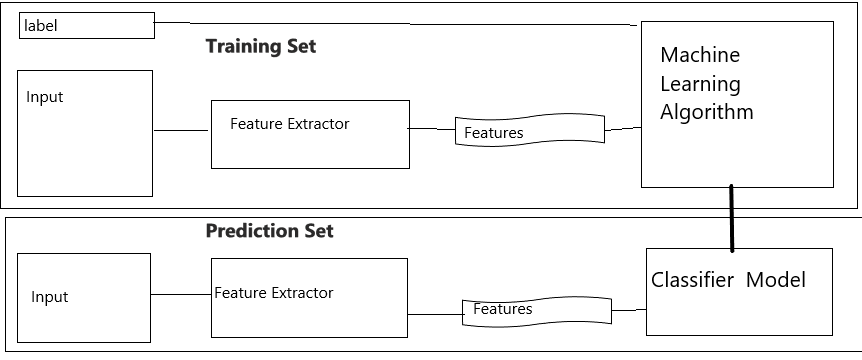
**Practical No. 8: Classification**

**Aim: Write a program for classification.**

**Classification** is the task of choosing the correct class label for a given input. In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance. Some examples of classification tasks are:

* 1. Deciding whether an email is spam or not.
  2. Deciding what the topic of a news article is, from a fixed list of topic areas such as “sports, ” “technology, ” and “politics.”
  3. Deciding whether a given occurrence of the word bank is used to refer to a river bank, a financial institution, the act of tilting to the side, or the act of depositing something in a financial institution.

The basic classification task has a number of interesting variants. For example, in multi-class classification, each instance may be assigned multiple labels; in open-class classification, the set of labels is not defined in advance; and in sequence classification, a list of inputs are jointly classified.

A **classifier** is called supervised if it is built based on training corpora containing the correct label for each input. The **framework** used by supervised classification is shown in figure.

The training set is used to train the model, and the dev-test set is used to perform error analysis. The test set serves in our final evaluation of the system. For reasons discussed below, it is important that we employ a separate dev-test set for error analysis, rather than just using the test set.

The division of the corpus data into different subsets is shown in following Figure:



**Code :**

* 1. **Supervised Classification - Gender Identification**

>> def gender\_features(word):

>> return {'last\_letter': word[-1]}

>> gender\_features('Shrek')

>> import nltk

>> nltk.download('names')

>> from nltk.corpus import names

>>labeled\_names = ([(name, 'male') for name in names.words('male.txt')] + [(name, 'female') for name in names.words('female.txt')])

>> import random

>> random.shuffle(labeled\_names)

>>featuresets = [(gender\_features(n), gender) for (n, gender) in labeled\_names]

>> train\_set, test\_set = featuresets[500:], featuresets[:500]

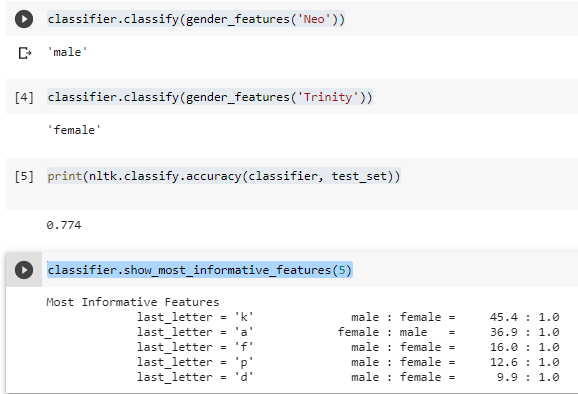
>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>> classifier.classify(gender\_features('Neo'))

>> classifier.classify(gender\_features('Trinity'))

>> print(nltk.classify.accuracy(classifier, test\_set))

>> classifier.show\_most\_informative\_features(5)



* 1. **Choosing right features**

>> from nltk.classify import apply\_features

>> train\_set = apply\_features(gender\_features, labeled\_names[500:])

>> test\_set = apply\_features(gender\_features, labeled\_names[:500])

>> def gender\_features2(name):

    features = {}

    features["first\_letter"] = name[0].lower()

    features["last\_letter"] = name[-1].lower()

    for letter in 'abcdefghijklmnopqrstuvwxyz':

        features["count({})".format(letter)] = name.lower().count(letter)

        features["has({})".format(letter)] = (letter in name.lower())

    return features

>> gender\_features2('John')



>>featuresets = [(gender\_features2(n), gender) for (n, gender) in labeled\_names]

>> train\_set, test\_set = featuresets[500:], featuresets[:500]

>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>> print(nltk.classify.accuracy(classifier, test\_set))

>> train\_names = labeled\_names[1500:]

>> devtest\_names = labeled\_names[500:1500]

>> test\_names = labeled\_names[:500]

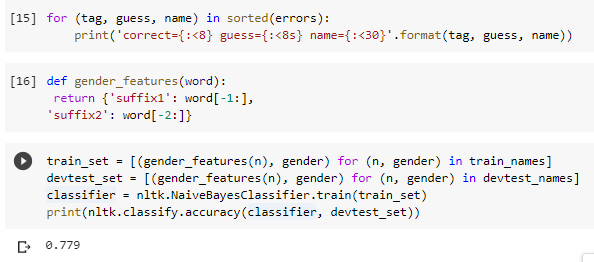
>>train\_set = [(gender\_features(n), gender) for (n, gender) in train\_names]

>>devtest\_set = [(gender\_features(n), gender) for (n, gender) in devtest\_names]

>>test\_set = [(gender\_features(n), gender) for (n, gender) in test\_names]

>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>> print(nltk.classify.accuracy(classifier, devtest\_set))



* 1. **Document classification**

>> import nltk

>> nltk.download('movie\_reviews')

>> from nltk.corpus import movie\_reviews

>> documents = [(list(movie\_reviews.words(fileid)), category)

>> for category in movie\_reviews.categories()

>> for fileid in movie\_reviews.fileids(category)]

>> random.shuffle(documents)

>> all\_words = nltk.FreqDist(w.lower() for w in movie\_reviews.words())

>> word\_features = list(all\_words)[:2000]

>> def document\_features(document):

    document\_words = set(document)

    features = {}

    for word in word\_features:

        features['contains({})'.format(word)] = (word in document\_words)

    return features

>> print(document\_features(movie\_reviews.words('pos/cv957\_8737.txt')))

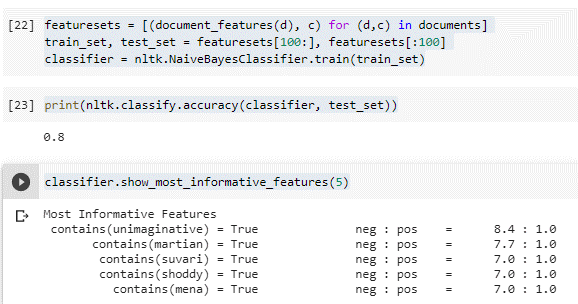
>> featuresets = [(document\_features(d), c) for (d,c) in documents]

>> train\_set, test\_set = featuresets[100:], featuresets[:100]

>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>> print(nltk.classify.accuracy(classifier, test\_set))

>> classifier.show\_most\_informative\_features(5)



* 1. **Sentence Segmentation**

>> nltk.download('treebank')

>> import nltk

>> nltk.download('punkt')

>> import nltk

>> sents = nltk.corpus.treebank\_raw.sents()

>> tokens = []

>> boundaries = set()

>> offset = 0

>> for sent in sents:

>> tokens.extend(sent)

>> offset += len(sent)

>> boundaries.add(offset-1)

>> def punct\_features(tokens, i):

>> return {'next-word-capitalized': tokens[i+1][0].isupper(),

'prev-word': tokens[i-1].lower(),

'punct': tokens[i],

'prev-word-is-one-char': len(tokens[i-1]) == 1}

>> featuresets = [(punct\_features(tokens, i), (i in boundaries))

>> for i in range(1, len(tokens)-1)

>> if tokens[i] in '.?!']

>> size = int(len(featuresets) \* 0.1)

>> train\_set, test\_set = featuresets[size:], featuresets[:size]

>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>> nltk.classify.accuracy(classifier, test\_set)



* 1. **Naive Bayes Classifier**

*Naive Bayes classification* is a fast and simple to understand classification method. Its speed is due to some simplifications we make about the underlying probability distributions, namely, the assumption about the independence of features. Yet, it can be quite powerful, especially when there are enough features in the data.

Suppose we have for each label L a probability distribution. This distribution gives probability for each possible combination of features (a feature vector):

P(features|L).

The main idea in Bayesian classification is to reverse the direction of dependence: we want to predict the label based on the features:

P(L|features)

This is possible by [the Bayes theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem):

P(L|features)=P(features|L)P(L)P(features).

Let's assume we have to labels L1 and L2, and their associated distributions: P(features|L1) and P(features|L2). If we have a data point with "features", whose label we don't know, we can try to predict it using the ratio of posterior probabilities:

P(L1|features)P(L2|features)=P(features|L1)P(L1)P(features|L2)P(L2).

If the ratio is greater than one, we label our data point with label L1, and if not, we give it label L2. The prior probabilities P(L1) and P(L2) of labels can be easily found out from the input data, as for each data point we also have its label. Same goes for the probabilities of features conditioned on the label.

**Code:**

>We first demonstrate naive Bayes classification using Gaussian distributions.

1]

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

2]

from sklearn.datasets import make\_blobs

X,y = make\_blobs(100, 2, centers=2, random\_state=2, cluster\_std=1.5)

colors=np.array(["red", "blue"])

plt.scatter(X[:, 0], X[:, 1], c=colors[y], s=50)

for label, c in enumerate(colors):

    plt.scatter([], [], c=c, label=str(label))

plt.legend();

#plt.colorbar();

3]

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import MultinomialNB

model = GaussianNB()

#model = MultinomialNB()

model.fit(X, y);

>Naive Bayes algorithm fitted two 2-dimensional Gaussian distribution to the data. The means and the variances define these distributions completely.

1]

print("Means:", model.theta\_)

print("Standard deviations:", model.sigma\_)

2]

def plot\_ellipse(ax, mu, sigma, color="k", label=None):

    """

    Based on

    http://stackoverflow.com/questions/17952171/not-sure-how-to-fit-data-with-a-gaussian-python.

    """

    from matplotlib.patches import Ellipse

    # Compute eigenvalues and associated eigenvectors

    vals, vecs = np.linalg.eigh(sigma)

    # Compute "tilt" of ellipse using first eigenvector

    x, y = vecs[:, 0]

    theta = np.degrees(np.arctan2(y, x))

    # Eigenvalues give length of ellipse along each eigenvector

    w, h = 2 \* np.sqrt(vals)

    ax.tick\_params(axis='both', which='major', labelsize=20)

    ellipse = Ellipse(mu, w, h, theta, color=color, label=label)  # color="k")

    ellipse.set\_clip\_box(ax.bbox)

    ellipse.set\_alpha(0.2)

    ax.add\_artist(ellipse)

    return ellipse

3]

plt.figure()

plt.xlim(-5, 5)

plt.ylim(-15, 5)

plot\_ellipse(plt.gca(), model.theta\_[0], np.identity(2)\*model.sigma\_[0], color="red")

plot\_ellipse(plt.gca(), model.theta\_[1], np.identity(2)\*model.sigma\_[1], color="blue");

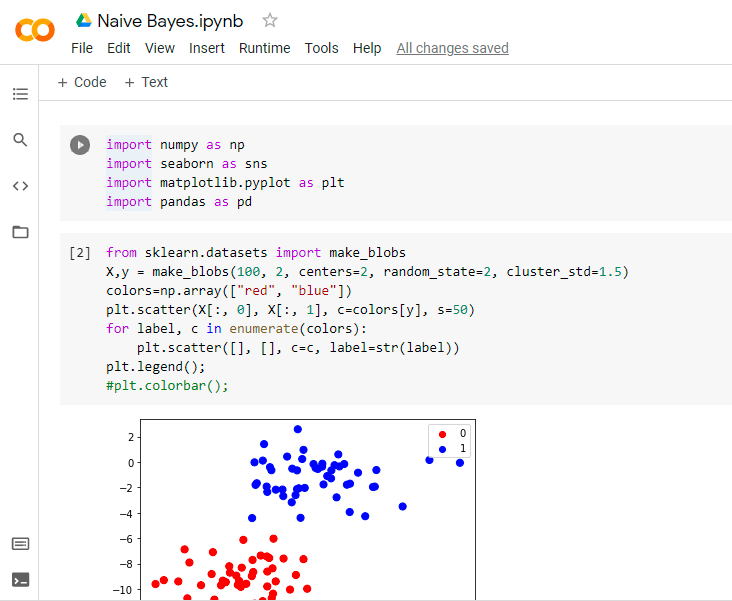
4]

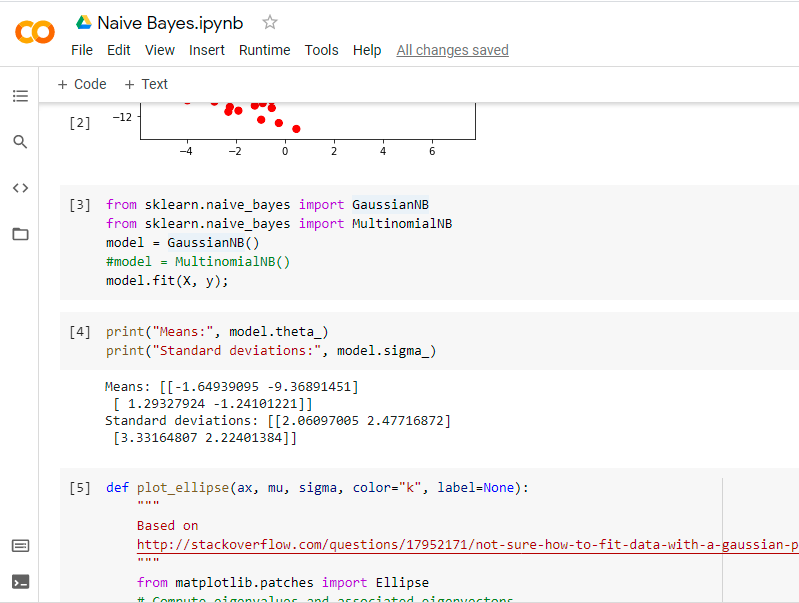
from sklearn.metrics import accuracy\_score

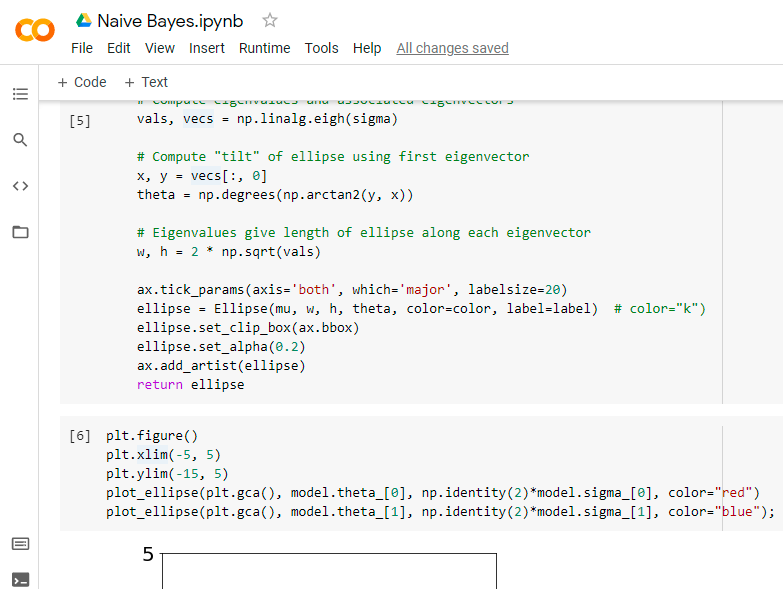
y\_fitted = model.predict(X)

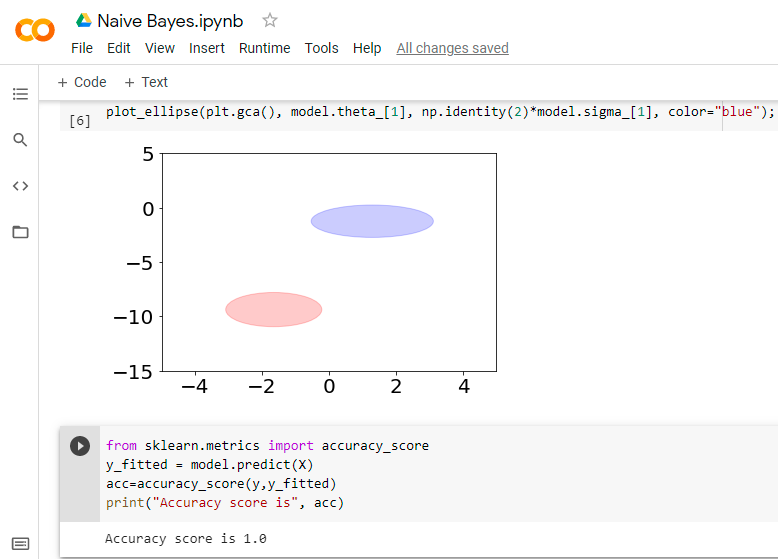
acc=accuracy\_score(y,y\_fitted)

print("Accuracy score is", acc)









* 1. **Using Wordnet Finding Synonym and Antonym**

WordNet’s structure makes it a useful tool for computational linguistics and natural language processing.

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions.First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated.Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

**Code:**

>> import nltk

>> nltk.download('wordnet')

# First, you're going to need to import wordnet:

>> from nltk.corpus import wordnet

# Then, we're going to use the term "program" to find synsets like so:

>> syns = wordnet.synsets("program")

# An example of a synset:

>> print(syns[0].name())

# Just the word:

>> print(syns[0].lemmas()[0].name())

# Definition of that first synset:

>> print(syns[0].definition())

# Examples of the word in use in sentences:

>> print(syns[0].examples())

>> import nltk

>> from nltk.corpus import wordnet

>> synonyms = []

>> antonyms = []

>> for syn in wordnet.synsets("good"):

  for l in syn.lemmas():

    synonyms.append(l.name())

    if l.antonyms():

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

>> print(set(synonyms))

>> print(set(antonyms))

>> import nltk

>> from nltk.corpus import wordnet

>> synonyms = []

>> antonyms = []

>> for syn in wordnet.synsets("good"):

  for l in syn.lemmas():

    synonyms.append(l.name())

    if l.antonyms():

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

>> print(set(synonyms))

>> print(set(antonyms))

>> import nltk

>> from nltk.corpus import wordnet

# Let's compare the noun of "ship" and "boat:"

>> w1 = wordnet.synset('run.v.01') # v here denotes the tag verb

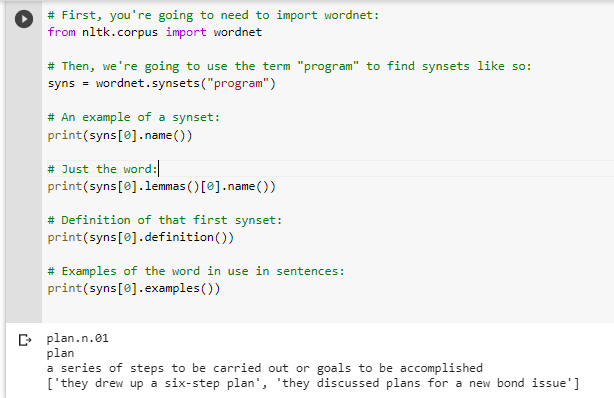
.01')

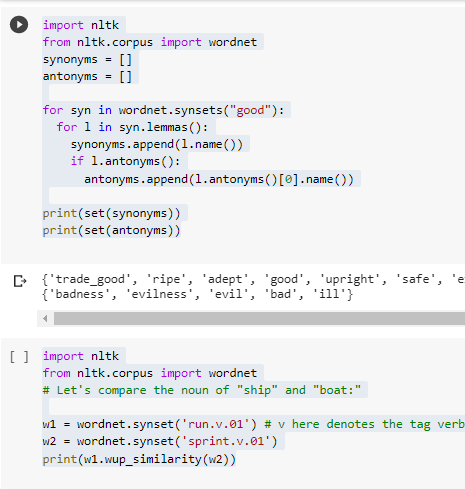
>> print(w1.wup\_similarity(w2))

>> w1 = wordnet.synset('ship.n.01')

>> w2 = wordnet.synset('boat.n.01') # n denotes noun

>> print(w1.wup\_similarity(w2))







* 1. **Word Sense Disambiguation - Leak Algorithm**

To use Python code to remove word ambiguity using the Lesk algorithm.

For example, in the sentences below, the word “bank” has different meanings based on the context of the sentence.

*Text1 = 'I went to the bank to deposit my money'*

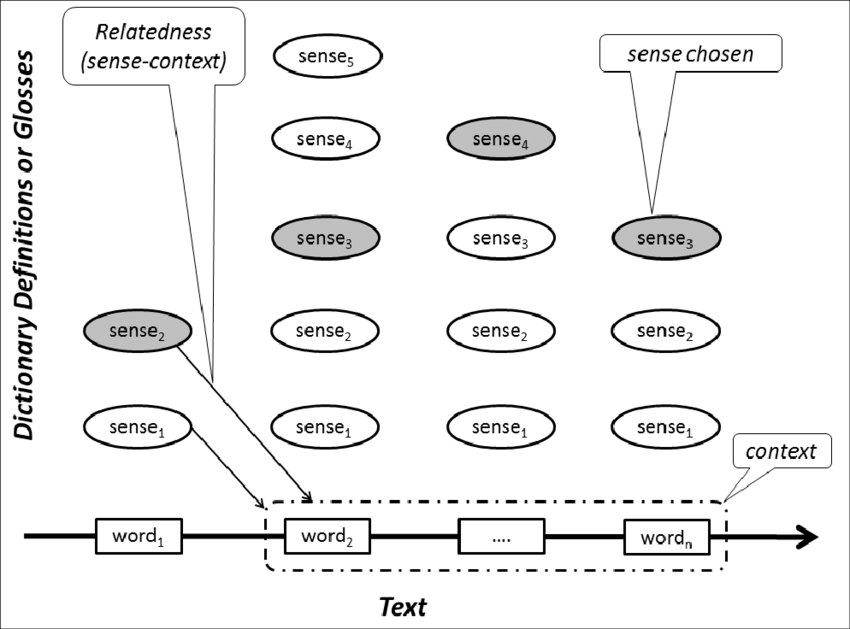
*Text2 = 'The river bank was full of dead fishes'*

The Lesk algorithm is the seminal dictionary-based method.

It is based on the hypothesis that words used together in text are related to each other and that the relation can be observed in the definitions of the words and their senses. Two (or more) words are disambiguated by finding the pair of dictionary senses with the greatest word overlap in their dictionary definitions. It searches for the shortest path between two words: the second word is iteratively searched among the definitions of every semantic variant of the first word, then among the definitions of every semantic variant of each word in the previous definitions and so on.

Finally, the first word is disambiguated by selecting the semantic variant which minimizes the distance from the first to the second word."

Basically, the context is chosen from meaning of the nearest words. Following is the simplified pictorial representation of the same...



Let's see the code to implement the Lesk algorithm in Python.

First install the library pywsd - python implementation of Word Sense Disambiguation (WSD)

1]

pip install pywsd==1.1.1

2]

import nltk

nltk.download('popular')

3]

import wordnet as wn

from pywsd.lesk import simple\_lesk

sentences = ['The workers at the plant were overworked',

'The plant was no longer bearing flowers',

'The workers at the industrial plant were overworked']

# calling the lesk function and printing results for both the sentences

print ("Context-1:", sentences[0])

answer = simple\_lesk(sentences[0],'plant')

print ("Sense:", answer)

print ("Definition : ", answer.definition())

Output:

Context-1: The workers at the plant were overworked

Sense: Synset('plant.v.06')

Definition : put firmly in the mind

4]

# calling the lesk function and printing results

print ("Context-1:", sentences[1])

answer = simple\_lesk(sentences[1],'plant')

print ("Sense:", answer)

print ("Definition : ", answer.definition())

output:

Context-1: The plant was no longer bearing flowers

Sense: Synset('plant.v.01')

Definition : put or set (seeds, seedlings, or plants) into the ground

5]

print ("Context-3:", sentences[2])

answer = simple\_lesk(sentences[2],'plant')

print ("Sense:", answer)

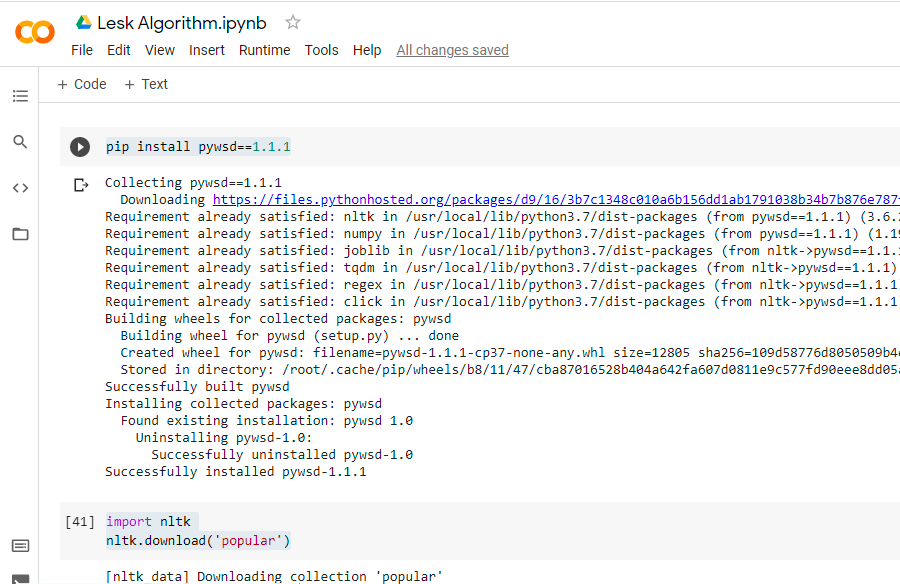
print ("Definition : ", answer.definition())

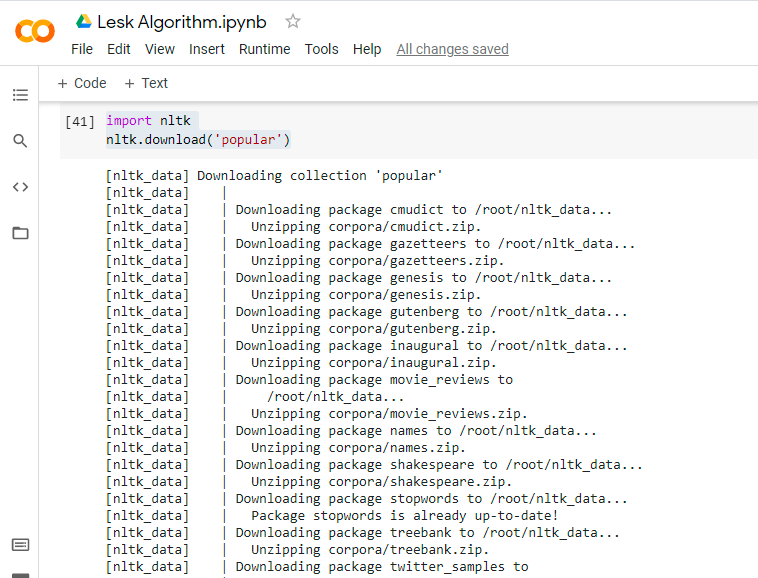
output:

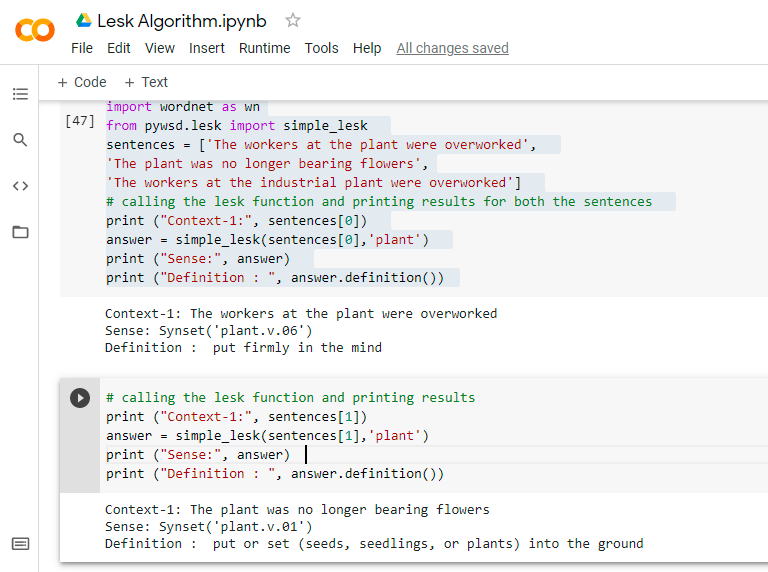
Context-3: The workers at the industrial plant were overworked

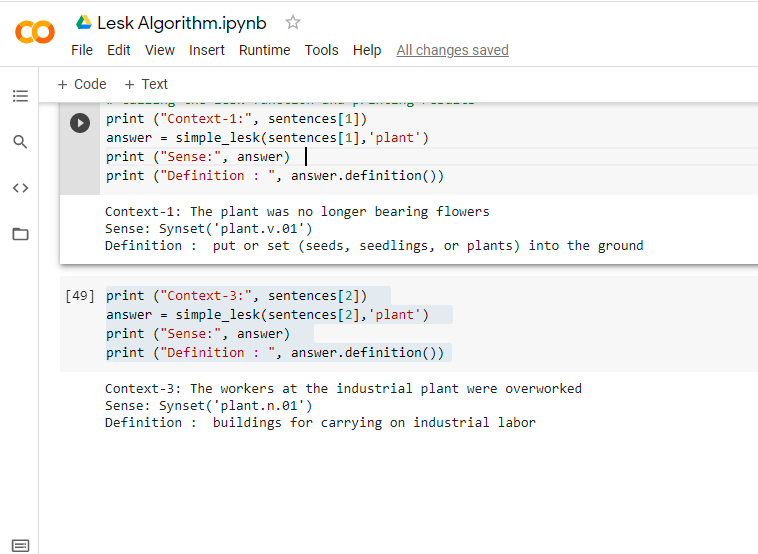
Sense: Synset('plant.n.01')

Definition : buildings for carrying on industrial labor









* 1. **Word2Vec**

Word2vec is a technique for natural language processing. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. ... As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector.

**Code:**

import nltk

nltk.download('brown')

import nltk

import nltk

nltk.download('punkt')

nltk.download('movie\_reviews')

import nltk

nltk.download('treebank')

from gensim.models import Word2Vec

from nltk.corpus import brown, movie\_reviews, treebank

b = Word2Vec(brown.sents())

mr = Word2Vec(movie\_reviews.sents())

t = Word2Vec(treebank.sents())

b.most\_similar('money', topn=5)

[('pay', 0.6832243204116821), ('ready', 0.6152011156082153), ('try', 0.5845392942428589), ('care', 0.5826011896133423), ('move', 0.5752171277999878)]

mr.most\_similar('money', topn=5)

[('unstoppable', 0.6900672316551208), ('pain', 0.6289106607437134), ('obtain', 0.62665855884552), ('jail', 0.6140228509902954), ('patients', 0.6089504957199097)]

t.most\_similar('money', topn=5)

[('short-term', 0.9459682106971741), ('-LCB-', 0.9449775218963623), ('rights', 0.9442864656448364), ('interested', 0.9430986642837524), ('national', 0.9396077990531921)]

b.most\_similar('great', topn=5)

[('new', 0.6999611854553223), ('experience', 0.6718623042106628), ('social', 0.6702290177345276), ('group', 0.6684836149215698), ('life', 0.6667487025260925)]

mr.most\_similar('great', topn=5)

[('wonderful', 0.7548679113388062), ('good', 0.6538234949111938), ('strong', 0.6523671746253967), ('phenomenal', 0.6296845078468323), ('fine', 0.5932096242904663)]

t.most\_similar('great', topn=5)

[('won', 0.9452997446060181), ('set', 0.9445616006851196), ('target', 0.9342271089553833), ('received', 0.9333916306495667), ('long', 0.9224691390991211)]

b.most\_similar('company', topn=5)

[('industry', 0.6164317727088928), ('technical', 0.6059585809707642), ('orthodontist', 0.5982754826545715), ('foamed', 0.5929019451141357), ('trail', 0.5763031840324402)]

mr.most\_similar('company', topn=5)

[('colony', 0.6689200401306152), ('temple', 0.6546304225921631), ('arrival', 0.6497283577919006), ('army', 0.6339291334152222), ('planet', 0.6184555292129517)]

t.most\_similar('company', topn=5)

[('panel', 0.7949466705322266), ('Herald', 0.7674347162246704), ('Analysts', 0.7463694214820862), ('amendment', 0.7282689809799194), ('Treasury', 0.719698429107666)]

