Practical No. 1

Aim: To implement Tokenization of text.

1) Importing the Necessary Libraries. Creating and Printing a Sample Token:

Code:

```
import nltk
nltk.download("punkt")
nltk.download("punkt_tab")
token="It was great catching up with you, take care!"
print(token)
```

Output:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
It was great catching up with you, take care!
```

2) Word Tokenization:

Code:

```
from nltk.tokenize import word_tokenize
print(word_tokenize(token))
```

Output:

```
['It', 'was', 'great', 'catching', 'up', 'with', 'you', ',', 'take', 'care', '!']
```

3) Storing Tokenized Words in a Variable:

Code:

```
token1=nltk.word_tokenize(token)
token1
```

Output:

```
['It',
 'was',
 'great',
 'catching',
 'up',
 'with',
 'you',
 ',',
 'take',
 'care',
 '!']
```

4) Checking the Data Type of the Token:

Code:

```
type(token)
```

Output:

```
str
```

5) Finding the Length of the Token:

Code:

```
len(token)
```

Output:

```
45
```

6) Frequency Distribution of Words:

Code:

```
from nltk.probability import FreqDist
fdist=FreqDist(token)
print(fdist)
```

Output:

```
<FreqDist with 20 samples and 45 outcomes>
```

7) Finding the Top 5 Most Common Tokens:

Code:

```
top_5=fdist.most_common(5)
top_5
```

Output:

```
top_5=fdist.most_common(5)
top_5
```

```
[(' ', 8), ('t', 5), ('a', 5), ('e', 3), ('c', 3)]
```

8) Using the Brown Corpus:

Code:

```
from nltk.corpus import brown
nltk.download("brown")
brown.words()
```

Output:

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

9) Downloading and Using the Gutenberg Corpus:

Code:

```
nltk.download("gutenberg")
nltk.corpus.gutenberg.fileids()
```

Output:

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

10) Extracting Words from a Text in the Gutenberg Corpus:

Code:

```
nltk.corpus.gutenberg.words('bible-kjv.txt')
```

Output:

```
['[', 'The', 'King', 'James', 'Bible', ']', 'The', ...]
```

11) Counting the Number of Words in the Gutenberg Corpus File:

Code:

```
c1=nltk.corpus.gutenberg.words('bible-kjv.txt')
len(c1)
```

Output:

```
1010654
```

12) Sentence Tokenization:

Code:

```
from nltk.tokenize import sent_tokenize
text="I am from Kalyan "
print(sent_tokenize(text))
```

Output:

```
['I am from Kalyan']
```

Practical No. 2

Aim: To implement Stop word removal.

1) Importing Required Libraries & Downloading NLTK Stopwords:

Code:

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

Output:

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

2) Displaying the List of Stopwords in English:

Code:

```
print(stopwords.words('english'))
```

Output:

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be', 'because', 'been', 'before', 'being', 'belo
```

Code:

```
stopwords.words('english')
```

Output:

```
stopwords.words('english')

['the',
 'their',
 'theirs',
 'them',
 'themselves',
 'then',
 'there',
 'these',
 'they',
 "they'd",
 "they'll",
 "they're",
 "they've",
 'this',
 'those',
 'through',
 'to',
 'too',
 'under',
 'until',
 'up',
 've',
 'very',
 'was',
 'wasn',
 "wasn't",
 'we',
 "we'd",
 "we'll",
 "we're",
 'were',
 'weren',
 "weren't",
 "we've",
 'what',
 'when',
 'where',
 'which',
 'while',
 'who',
 'whom',
 'why',
 'will',
 'with',
 'won',
 "won't",
 'wouldn',
 "wouldn't",
 'y',
 'you',
 "you'd",
 "you'll",
 'your',
 "you're",
 'yours',
 'yourself',
 'yourselves',
 "you've"]
```

3) Storing Stopwords in a Set:

Code:

```
stop_words = set(stopwords.words('english'))
```

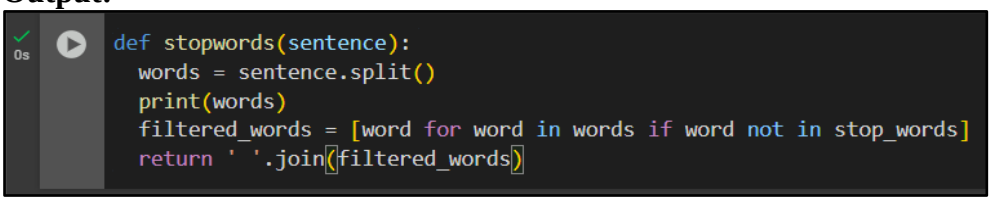
Output:

4) Function to Remove Stopwords from a Sentence:

Code:

```
def stopwords(sentence):
  words = sentence.split()
  print(words)
  filtered_words = [word for word in words if word not in stop_words]
  return ' '.join(filtered_words)
```

Output:

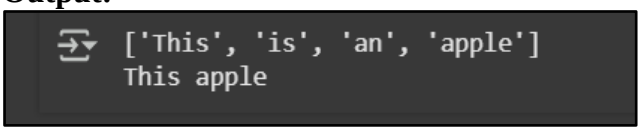


5) Testing the Function:

Code:

```
sentence = "This is an apple"
filtered_sen = stopwords(sentence)
print(filtered_sen)
```

Output:



6) Using Tokenization for More Accurate Stopword Removal:

Code:

```
import nltk
nltk.download('punkt_tab')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
set(stopwords.words('english'))
```

Output:

```
import nltk
nltk.download('punkt_tab')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
set(stopwords.words('english'))
```

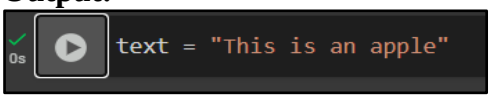
```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
 'been',
 'before',
 'being',
 'below',
 'between',
 'both',
 'both',
 'but',
 'by',
 'can',
 'couldn',
 "couldn't",
 'd',
 'did',
 'didn',
 "didn't",
 'do',
 'does',
 'doesn',
 "doesn't",
 'doing',
 'don',
 "don't",
 'down',
 'during',
 'each',
 'few',
 'for',
 'from',
 'further',
 'had',
 'hadn',
 "hadn't",
 'has',
 'hasn',
 "hasn't",
 'have',
 'haven',
 "haven't",
 'having',
 'having',
 'he',
 "he'd",
 "he'll",
 "he's",
 'her',
 'here',
 'hers',
 'herself',
 'him',
 'himself',
 'his',
 'how',
 'i',
 "i'd",
 "i'll",
 "i'm",
 "i've",
 'if',
 'in',
 'into',
 'is',
 'isn',
 "isn't",
 'it',
 "it'd",
 "it'll",
 "it's",
 'its',
 'itself',
 'just',
 'll',
 'm',
 'ma',
 'ma',
 'me',
 'mightn',
 "mightn't",
 'more',
 'most',
 'mustn',
 "mustn't",
 'my',
 'myself',
 'needn',
 "needn't",
 'no',
 'nor',
 'not',
 'now',
 'o',
 'of',
 'off',
 'on',
 'once',
 'only',
 'or',
 'other',
 'our',
 'ours',
 'ourselves',
 'out',
 'over',
 'own',
 're',
 's',
 'same',
 'shan',
 'shan',
 "shan't",
 'she',
 "she'd",
 "she'll",
 "she's",
 'should',
 "should've",
 'shouldn',
 "shouldn't",
 'so',
 'some',
 'such',
 't',
 'than',
 'that',
 "that'll",
 'the',
 'their',
 'theirs',
 'them',
 'themselves',
 'then',
 'there',
 'these',
 'they',
 "they'd",
 "they'll",
 "they're",
 "they've",
 'this',
 'those',
 'through',
 'to',
 'to',
 'too',
 'under',
 'until',
 'up',
 've',
 'very',
 'was',
 'wasn',
 "wasn't",
 'we',
 "we'd",
 "we'll",
 "we're",
 "we've",
 'were',
 'weren',
 "weren't",
 'what',
 'when',
 'where',
 'which',
 'while',
 'who',
 'whom',
 'why',
 'will',
 'with',
 'won',
 "won't",
 'wouldn',
 "wouldn't",
 'y',
 'you',
 'you',
 "you'd",
 "you'll",
 "you're",
 "you've",
 'your',
 'yours',
 'yourself',
 'yourselves'}
```

7) Tokenizing a Sample Sentence:

Code:

```
text = "This is an apple"
```

Output:



Code:

```
stop_words1 = set(stopwords.words('english'))
```

Output:

Code:

```
word_tokens = word_tokenize(text)
```

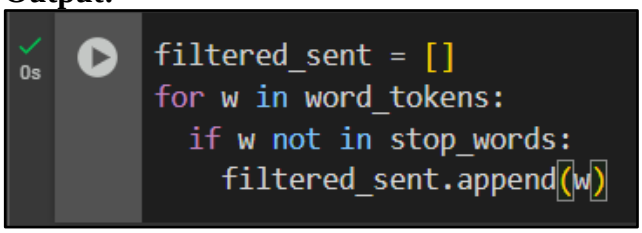
Output:

8) Removing Stopwords Using Tokenization:

Code:

```
filtered_sent = []
for w in word_tokens:
  if w not in stop_words:
    filtered_sent.append(w)
```

Output:

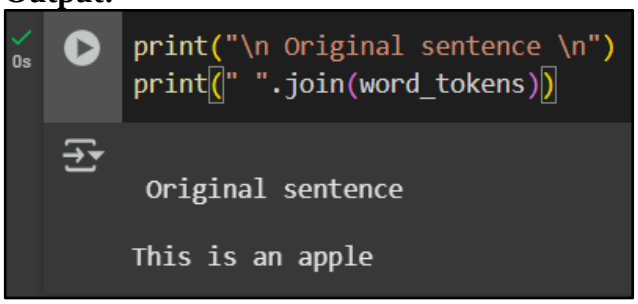


9) Displaying the Original and Filtered Sentence:

Code:

```
print("\n Original sentence \n")
print(" ".join(word_tokens))
```

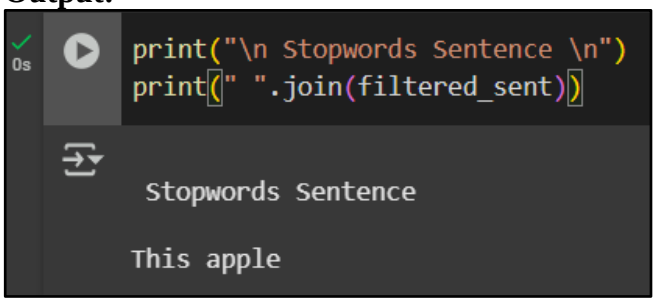
Output:



Code:

```
print("\n Stopwords Sentence \n")
print(" ".join(filtered_sent))
```

Output:



Practical No. 3

Aim: To implement Stemming of text.

1) Basic Stemming using PorterStemmer:

Code:

```
from nltk.stem import PorterStemmer
e_words = ["run", "running", "runner", "ran", "runs"]
ps = PorterStemmer()
for w in e_words:
    rootWord = ps.stem(w)
    print(rootWord)
```

Output:

```
run
run
runner
ran
run
```

2) Applying Stemming to a Sentence:

Code:

```
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('punkt_tab')
sentence = "Natural language processing enables computers to understand and generate human language efficiently."
words = word_tokenize(sentence)
ps = PorterStemmer()
for w in words:
    rootWord = ps.stem(w)
    print(rootWord)
```

Output:

```
natur
languag
process
enabl
comput
to
understand
and
gener
human
languag
effici
.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Practical No. 4

Aim: To implement Lemmatization.

1) Importing Required Libraries & Downloading Necessary NLTK Resources:

Code:

```
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('wordnet')
```

Output:

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

2) Initializing the Lemmatizer and Tokenizing Words:

Code:

```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer=WordNetLemmatizer()
text="bought buying buys"
tokenization=nltk.word_tokenize(text)
```

Output:

```
0s
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer=WordNetLemmatizer()
text="bought buying buys"
tokenization=nltk.word_tokenize(text)
```

3) Applying Lemmatization on Each Token:

Code:

```
for w in tokenization:
  print("Lemma for {} is {}".format(w,wordnet_lemmatizer.lemmatize(w)))
```

Output:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Lemma for bought is bought
Lemma for buying is buying
Lemma for buys is buy
```

Practical No. 5

Aim: To implement N-gram model.

1) Importing Required Libraries & Downloading Necessary NLTK Resources:

Code:

```
import nltk.util
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize
```

Output:

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

2) Tokenizing Sample Data:

Code:

```
Sample_Data = 'I am FYMCA Student at Swayam Siddhi College.'
Sample_Tokens = word_tokenize(Sample_Data)
Sample_Tokens
```

Output:

```
['I', 'am', 'FYMCA', 'Student', 'at', 'Swayam', 'Siddhi', 'College', '.']
```

3) Generating Bigrams, Trigrams, and N-grams

a) ***Bigrams:*** *Generates two-word sequences.*

Code:

```
list(nltk.bigrams(Sample_Tokens))
```

Output:

```
list(nltk.bigrams(Sample_Tokens))

[('I', 'am'),
 ('am', 'FYMCA'),
 ('FYMCA', 'Student'),
 ('Student', 'at'),
 ('at', 'Swayam'),
 ('Swayam', 'Siddhi'),
 ('Siddhi', 'College'),
 ('College', '.')]
```

b) Trigrams: *Generates three-word sequences.*

Code:

```
list(nltk.trigrams(Sample_Tokens))
```

Output:

```
[('I', 'am', 'FYMCA'),
 ('am', 'FYMCA', 'Student'),
 ('FYMCA', 'Student', 'at'),
 ('Student', 'at', 'Swayam'),
 ('at', 'Swayam', 'Siddhi'),
 ('Swayam', 'Siddhi', 'College'),
 ('Siddhi', 'College', '.')]
```

c) N-grams: *Generates n-word sequences based on the specified value of* ***n.***

Code:

```
list(nltk.ngrams(Sample_Tokens,4))
```

Output:

```
[('I', 'am', 'FYMCA', 'Student'),
 ('am', 'FYMCA', 'Student', 'at'),
 ('FYMCA', 'Student', 'at', 'Swayam'),
 ('Student', 'at', 'Swayam', 'Siddhi'),
 ('at', 'Swayam', 'Siddhi', 'College'),
 ('Swayam', 'Siddhi', 'College', '.')]
```

Code:

```
list(nltk.ngrams(Sample_Tokens,8))
```

Output:

```
list(nltk.ngrams(Sample_Tokens,8))

[('I', 'am', 'FYMCA', 'Student', 'at', 'Swayam', 'Siddhi', 'College'),
 ('am', 'FYMCA', 'Student', 'at', 'Swayam', 'Siddhi', 'College', '.')]
```

4) Building an N-gram Language Model Using Reuters Corpus:

Code:

```
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('reuters')
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict
model = defaultdict(lambda: defaultdict(lambda: 0))
```

```
for sentence in reuters.sents():
  for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
    model[(w1,w2)] [w3] += 1
for w1_w2 in model:
  total_count =  float(sum(model[w1_w2].values()))
  for w3 in model[w1_w2]:
    model[w1_w2][w3] /= total_count
```

Output:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package reuters to /root/nltk_data...
[nltk_data]   Package reuters is already up-to-date!
```

5) Retrieving Next Word Probabilities for "the news":

Code:

```
sorted(dict(model["the","news"]).items(),key=lambda x : -1*[1])
```

Output:

```
[('brought', 0.041666666666666664),
 ('about', 0.041666666666666664),
 ('with', 0.08333333333333333),
 ('of', 0.125),
 ('conference', 0.25),
 (',', 0.08333333333333333),
 ('broke', 0.041666666666666664),
 ('.', 0.125),
 ('on', 0.041666666666666664),
 ('agency', 0.08333333333333333),
 ('that', 0.08333333333333333)]
```

Practical No. 6

Aim: To implement POS tagging.

1) Importing Required Libraries & Downloading Necessary NLTK Resources:

Code:

```
import nltk
nltk.download('punkt')
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time
nltk. download( 'treebank')
nltk.download ('universal_tagset')
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
print(nltk_data[:2])
```

Output:

```
import nltk
nltk.download('punkt')
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time
nltk. download( 'treebank')
nltk.download ('universal_tagset')
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
print(nltk_data[:2])

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'), ('join', 'VERB'), ('the', 'DET'), ('board', 'N
```

2) Downloading Additional NLTK Resources:

Code:

```
import nltk
nltk.download('wordnet')
from nltk.tokenize import word_tokenize
```

Output:

```
import nltk
nltk.download('wordnet')
from nltk.tokenize import word_tokenize

[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

3) Downloading POS Tagging Models:

Code:

```
import nltk
#nltk.download('punkt')
nltk.download('punkt_tab')
#nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')
```

Output:

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
True
```

4) Tokenizing a Sample Sentence:

Code:

```
sample1 = "We are MCA Students"
sample_tokens = word_tokenize(sample1)
sample_tokens
```

Output:

```
['We', 'are', 'MCA', 'Students']
```

5) Applying POS Tagging on Each Token:

Code:

```
for i in sample_tokens:
  print(nltk.pos_tag([i]))
```

Output:

For the sentence **"We are MCA Students"**, the output might look like:

```
[('We', 'PRP')]
[('are', 'VBP')]
[('MCA', 'NNP')]
[('Students', 'NNS')]
```

This means:

- **"We"** is a pronoun.
- **"are"** is a verb in present tense.
- **"MCA"** is a proper noun.
- **"Students"** is a plural noun.

Practical No. 7

Aim: Building a custom NER system.

1) Importing Required Libraries:

Code:

```
import pandas as pd
import spacy
import requests
from bs4 import BeautifulSoup
nlp =  spacy.load("en_core_web_sm")
pd.set_option("display.max_rows",200)
```

Output:

2) Applying NER on Sample Text:

Code:

```
content = "Apple Retail Online in India does not offer trade-in for Mac, iPad, and Apple Watch for $2 Billion."
doc =nlp(content)
for ent in doc.ents:
    print(ent.text,ent.start_char,ent.end_char,ent.label_)
```

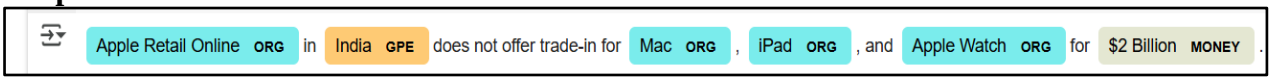
Output:

3) Visualizing Entities:

Code:

```
from spacy import displacy
displacy.render(doc,style="ent")
```

Output:



4) Storing Entities in a DataFrame:

Code:

```
entities = [(ent.text,ent.label_,ent.lemma_) for ent in doc.ents]
df = pd.DataFrame(entities,columns=["Text","Type","Lemma"])
print(df)
```

Output:

```
                  Text   Type               Lemma
0  Apple Retail Online    ORG  Apple Retail Online
1                India    GPE               India
2                  Mac    ORG                 Mac
3                 iPad    ORG                iPad
4          Apple Watch    ORG          Apple Watch
5           $2 Billion  MONEY           $2 billion
```

5) Testing NER on Another Sentence:

Code:

```
content = "Elon Musk is the CEO of Tesla, which is based in California."
doc =nlp(content)
for ent in doc.ents:
    print(ent.text,ent.start_char,ent.end_char,ent.label_)
```

Output:

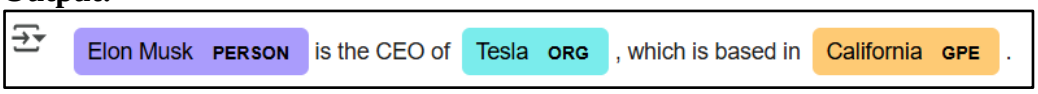
```
Elon Musk 0 9 PERSON
Tesla 24 29 ORG
California 49 59 GPE
```

6) Visualizing Entities Again:

Code:

```
from spacy import displacy
displacy.render(doc,style="ent")
```

Output:



7) Storing Results in DataFrame Again:

Code:

```
entities = [(ent.text,ent.label_,ent.lemma_) for ent in doc.ents]
df = pd.DataFrame(entities,columns=["Text","Type","Lemma"])
print(df)
```

Output:

```
          Text    Type       Lemma
0   Elon Musk  PERSON   Elon Musk
1       Tesla     ORG       Tesla
2  California     GPE  California
```

Practical No. 8

Aim: Creating and comparing different text representations.

1) One-Hot Encoding (OHE):

Code:

```
#OneHotVector
from sklearn.preprocessing import OneHotEncoder
import itertools

document = ["NLP", "stands", "for", "Natural", "Language", "Processing"]

tokens = [doc.split(" ") for doc in document]
token_chain = itertools.chain.from_iterable(tokens)
word_to_id = {token: idx for idx, token in enumerate(set(token_chain))}

token_ids = [[word_to_id[token] for token in toke] for toke in tokens]

vec = OneHotEncoder(categories="auto")
V =  vec.fit_transform(token_ids)
print(V.toarray())
```

Output:

```
[[0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
```

2) Bag of Words (BoW):

Code:

```
#bag of words
from sklearn.feature_extraction.text import CountVectorizer

document = ["The fluffy gray cat sat quietly, watching birds outside the window."]
vec = CountVectorizer()
vec = vec.fit(document)
print(vec.vocabulary_)
x = vec.transform(document)
print(x.toarray())
```

Output:

```
{'the': 7, 'fluffy': 2, 'gray': 3, 'cat': 1, 'sat': 6, 'quietly': 5, 'watching': 8, 'birds': 0, 'outside': 4, 'window': 9}
[[1 1 1 1 1 1 1 2 1 1]]
```

3) Term Frequency - Inverse Document Frequency (TF-IDF):

Code:

```
#TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["My Name is Chandrashekhar Pradhan","I am Pursuing MCA","From Swayam Siddhi College."]
tf = TfidfVectorizer()
txt_fit = tf.fit(text)
txt_transform = txt_fit.transform(text)
idf = tf.idf_
for word, value in zip(txt_fit.get_feature_names_out(), idf):
    print(f"{word}: {value}")
```

Output:

```
am: 1.6931471805599454
chandrashekhar: 1.6931471805599454
college: 1.6931471805599454
from: 1.6931471805599454
is: 1.6931471805599454
mca: 1.6931471805599454
my: 1.6931471805599454
name: 1.6931471805599454
pradhan: 1.6931471805599454
pursuing: 1.6931471805599454
siddhi: 1.6931471805599454
swayam: 1.6931471805599454
```

4) N-grams:

Code:

```
#Ngram
from sklearn.feature_extraction.text import CountVectorizer
text1 = ["My Name is Chandrashekhar Pradhan ","I am Pursuing MCA."]
cv = CountVectorizer(ngram_range=(2, 2))
bow = cv.fit_transform(text1)
print(cv.vocabulary_)
print(bow[0].toarray())
```

Output:

```
{'my name': 3, 'name is': 4, 'is chandrashekhar': 2, 'chandrashekhar pradhan': 1, 'am pursuing': 0, 'pursuing mca': 5}
[[0 1 1 1 1 0]]
```

Practical No. 9

Aim: Training and using word embeddings.

Code:

```
# Step 1: Install required libraries
!pip install gensim==4.3.1
!pip install nltk

#This is the fix! Downgrade numpy to 1.25.2,
# to fix compatibility issues with gensim==4.3.1
!pip install numpy==1.25.2

!pip install --upgrade gensim

# Step 2: Import required libraries
import gensim
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')

# Step 3: Sample corpus (list of sentences)
sentences = [
    "I love machine learning",
    "Natural language processing is amazing",
    "Word embeddings are useful for many NLP tasks",
    "Gensim makes it easy to work with Word2Vec models",
    "Deep learning is a subset of machine learning"
]

# Step 4: Tokenize each sentence
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

# Step 5: Train Word2Vec model
model = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5, min_count=1,
workers=4)

# Step 6: Explore the trained model

# Get the vector for the word 'machine'
word_vector = model.wv['machine']
print("Word vector for 'machine':\n", word_vector)

# Find words most similar to 'machine'
similar_words = model.wv.most_similar('machine', topn=3)
print("\nWords most similar to 'machine':", similar_words)
```

```
# Step 7: Save the model
model.save("word2vec_model.model")

# Step 8: Load the saved model (optional, but good practice to save for later use)
loaded_model = Word2Vec.load("word2vec_model.model")

# Check similarity again with the loaded model
similar_words_loaded = loaded_model.wv.most_similar('machine', topn=3)
print("\nWords most similar to 'machine' after loading the model:", similar_words_loaded)
```

Output:

```
Requirement already satisfied: gensim==4.3.1 in /usr/local/lib/python3.11/dist-packages (4.3.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim==4.3.1) (1.25.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim==4.3.1) (1.14.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim==4.3.1) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim==4.3.1) (1.17.2)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
```

```
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: numpy==1.25.2 in /usr/local/lib/python3.11/dist-packages (1.25.2)
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.1)
Collecting gensim
  Using cached gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.1 kB)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.25.2)
Collecting scipy<1.14.0,>=1.7.0 (from gensim)
  Using cached scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
Using cached gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26.7 MB)
Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (38.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.6/38.6 MB 13.4 MB/s eta 0:00:00
Installing collected packages: scipy, gensim
  Attempting uninstall: scipy
    Found existing installation: scipy 1.14.1
    Uninstalling scipy-1.14.1:
      Successfully uninstalled scipy-1.14.1
  Attempting uninstall: gensim
    Found existing installation: gensim 4.3.1
    Uninstalling gensim-4.3.1:
      Successfully uninstalled gensim-4.3.1
Successfully installed gensim-4.3.3 scipy-1.13.1
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
Word vector for 'machine':
 [-8.6232023e-03  3.6673949e-03  5.1919348e-03  5.7430919e-03
  7.4706664e-03 -6.1715539e-03  1.1076197e-03  6.0518682e-03
 -2.8409341e-03 -6.1783697e-03 -4.0995868e-04 -8.3717899e-03
 -5.6022862e-03  7.1073603e-03  3.3550612e-03  7.2279559e-03
  6.8040458e-03  7.5353943e-03 -3.7912515e-03 -5.6063005e-04
  2.3494069e-03 -4.5220377e-03  8.3925594e-03 -9.8600434e-03
  6.7661870e-03  2.9157954e-03 -4.9350723e-03  4.4002570e-03
 -1.7388146e-03  6.7138053e-03  9.9696256e-03 -4.3638381e-03
 -6.0030312e-04 -5.6980643e-03  3.8537364e-03  2.7873784e-03
  6.8950835e-03  6.1021838e-03  9.5426831e-03  9.2768455e-03
  7.9005938e-03 -6.9927704e-03 -9.1570467e-03 -3.5549139e-04
 -3.1006520e-03  7.8969076e-03  5.9416564e-03 -1.5470812e-03
  1.5126592e-03  1.7896690e-03  7.8209788e-03 -9.5148040e-03
 -2.0696511e-04  3.4687200e-03 -9.3905278e-04  8.3869854e-03
  9.0147695e-03  6.5410887e-03 -7.0938119e-04  7.7143558e-03
 -8.5370513e-03  3.2097639e-03 -4.6379459e-03 -5.0899023e-03
  3.5911754e-03  5.3728716e-03  7.7731274e-03 -5.7667121e-03
  7.4351588e-03  6.6278367e-03 -3.7099044e-03 -8.7507693e-03
  5.4408293e-03  6.5142759e-03 -7.8686461e-04 -6.7127156e-03
 -7.0898072e-03 -2.4961706e-03  5.1430115e-03 -3.6674212e-03
 -9.3749147e-03  3.8296112e-03  4.8867050e-03 -6.4293230e-03
  1.2101122e-03 -2.0754174e-03  2.6738873e-05 -9.8863319e-03
  2.6924331e-03 -4.7541191e-03  1.0889295e-03 -1.5784500e-03
  2.1980463e-03 -7.8842593e-03 -2.7169366e-03  2.6638270e-03
  5.3513488e-03 -2.3943419e-03 -9.5137162e-03  4.5066630e-03]

Words most similar to 'machine': [('deep', 0.18888916075229645), ('to', 0.18855030834674835), ('for', 0.1608063280582428)]

Words most similar to 'machine' after loading the model: [('deep', 0.18888916075229645), ('to', 0.18855030834674835), ('for', 0.1608063280582428)]
```

Code:

```
# Step 1: Install required libraries
!pip install gensim
!pip install nltk

!pip install --upgrade numpy
!pip install --upgrade gensim

# Step 2: Import required libraries
import gensim
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')

# Step 3: Sample corpus (list of sentences)
sentences = [
    "I love machine learning",
    "Natural language processing is amazing",
    "Word embeddings are useful for many NLP tasks",
    "Gensim makes it easy to work with Word2Vec models",
    "Deep learning is a subset of machine learning"
]

# Step 4: Tokenize each sentence
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

# Step 5: Train Word2Vec model
model = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5, min_count=1,
workers=4)

# Step 6: Explore the trained model

# Get the vector for the word 'machine'
word_vector = model.wv['machine']
print("Word vector for 'machine':\n", word_vector)

# Find words most similar to 'machine'
similar_words = model.wv.most_similar('machine', topn=3)
print("\nWords most similar to 'machine':", similar_words)

# Step 7: Save the model
model.save("word2vec_model.model")

# Step 8: Load the saved model (optional, but good practice to save for later use)
loaded_model = Word2Vec.load("word2vec_model.model")

# Check similarity again with the loaded model
```

```
similar_words_loaded = loaded_model.wv.most_similar('machine', topn=3)
print("\nWords most similar to 'machine' after loading the model:", similar_words_loaded)
```

Output:

```
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.25.2)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.25.2)
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.25.2)
Collecting numpy
  Downloading numpy-2.2.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 1.7 MB/s eta 0:00:00
Downloading numpy-2.2.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 79.6 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.25.2
    Uninstalling numpy-1.25.2:
      Successfully uninstalled numpy-1.25.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflic
gensim 4.3.3 requires numpy<2.0,>=1.18.5, but you have numpy 2.2.4 which is incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.4 which is incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 2.2.4 which is incompatible.
Successfully installed numpy-2.2.4
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
Collecting numpy<2.0,>=1.18.5 (from gensim)
  Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 2.2 MB/s eta 0:00:00
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 59.2 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.2.4
    Uninstalling numpy-2.2.4:
```

```
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.2.4
    Uninstalling numpy-2.2.4:
      Successfully uninstalled numpy-2.2.4
Successfully installed numpy-1.26.4
Word vector for 'machine':
 [-8.6232023e-03  3.6673949e-03  5.1919348e-03  5.7430919e-03
  7.4706664e-03 -6.1715539e-03  1.1076197e-03  6.0518682e-03
 -2.8409341e-03 -6.1783697e-03 -4.0995868e-04 -8.3717899e-03
 -5.6022862e-03  7.1073603e-03  3.3550612e-03  7.2279559e-03
  6.8040458e-03  7.5353943e-03 -3.7912515e-03 -5.6063005e-04
  2.3494069e-03 -4.5220377e-03  8.3925594e-03 -9.8600434e-03
  6.7661870e-03  2.9157954e-03 -4.9350723e-03  4.4002570e-03
 -1.7388146e-03  6.7138053e-03  9.9696256e-03 -4.3638381e-03
 -6.0030312e-04 -5.6980643e-03  3.8537364e-03  2.7873784e-03
  6.8950835e-03  6.1021838e-03  9.5426831e-03  9.2768455e-03
  7.9005938e-03 -6.9927704e-03 -9.1570467e-03 -3.5549139e-04
 -3.1006520e-03  7.8969076e-03  5.9416564e-03 -1.5470812e-03
  1.5126592e-03  1.7896690e-03  7.8209788e-03 -9.5148040e-03
 -2.0696511e-04  3.4687200e-03 -9.3905278e-04  8.3869854e-03
  9.0147695e-03  6.5410887e-03 -7.0938119e-04  7.7143558e-03
 -8.5370513e-03  3.2097639e-03 -4.6379459e-03 -5.0899023e-03
  3.5911754e-03  5.3728716e-03  7.7731274e-03 -5.7667121e-03
  7.4351588e-03  6.6278367e-03 -3.7099044e-03 -8.7507693e-03
  5.4408293e-03  6.5142759e-03 -7.8686461e-04 -6.7127156e-03
 -7.0898072e-03 -2.4961706e-03  5.1430115e-03 -3.6674212e-03
 -9.3749147e-03  3.8296112e-03  4.8867050e-03 -6.4293230e-03
  1.2101122e-03 -2.0754174e-03  2.6738873e-05 -9.8863319e-03
  2.6924331e-03 -4.7541191e-03  1.0889295e-03 -1.5784500e-03
  2.1980463e-03 -7.8842593e-03 -2.7169366e-03  2.6638270e-03
  5.3513488e-03 -2.3943419e-03 -9.5137162e-03  4.5066630e-03]
```

```
Words most similar to 'machine': [('deep', 0.18888916075229645), ('to', 0.18855030834674835), ('for', 0.1608063280582428)]

Words most similar to 'machine' after loading the model: [('deep', 0.18888916075229645), ('to', 0.18855030834674835), ('for', 0.1608063280582428)]
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Practical No. 10

AIM : Building a custom NER system

Requirement : jupyter

Code:

```
! pip install numpy hmmlearn scikit-learn
import numpy as np
from hmmlearn import hmm
from sklearn.feature_extraction.text import CountVectorizer
# Sample data
documents = ["I love programming", "Python is great", "I hate bugs", "Debugging is fun"]
labels = [1, 1, 0, 1]  # 1: positive, 0: negative
# Feature extraction
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents).toarray()
# Define HMM
model = hmm.MultinomialHMM(n_components=2, n_iter=100)
# Fit the model
model.fit(X)
# Predicting a new document
new_doc = ["I enjoy coding"]
new_X = vectorizer.transform(new_doc).toarray()
logprob, states = model.decode(new_X, algorithm="viterbi")
print("Predicted class:", states)
```

Output:

```
Requirement already satisfied: numpy in c:\users\shaikh aasiya\anaconda3\lib\site-packages (1.26.4)
Collecting hmmlearn
  Downloading hmmlearn-0.3.3-cp312-cp312-win_amd64.whl.metadata (3.1 kB)
Requirement already satisfied: scikit-learn in c:\users\shaikh aasiya\anaconda3\lib\site-packages (1.5.1)
Requirement already satisfied: scipy>=0.19 in c:\users\shaikh aasiya\anaconda3\lib\site-packages (from hmmlearn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\shaikh aasiya\anaconda3\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\shaikh aasiya\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Downloading hmmlearn-0.3.3-cp312-cp312-win_amd64.whl (127 kB)
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.3
MultinomialHMM has undergone major changes. The previous version was implementing a CategoricalHMM (a special case of MultinomialHMM). This new implementat
ion follows the standard definition for a Multinomial distribution (e.g. as in https://en.wikipedia.org/wiki/Multinomial_distribution). See these issues fo
r details:
https://github.com/hmmlearn/hmmlearn/issues/335
https://github.com/hmmlearn/hmmlearn/issues/340
Predicted class: [0]
```

Code:

```
import numpy as np
from hmmlearn import hmm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
documents = [
"I love programming",
"Python is great",
"I hate bugs",
"Debugging is fun",
"I enjoy coding",
"I dislike errors",
"Coding is awesome",
"I can't stand bugs"
]
labels = [1, 1, 0, 1, 1, 0, 1, 0]
X_train, X_test, y_train, y_test = train_test_split(documents, labels, test_size=0.25, random_state=42)
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train).toarray()
X_test_vectorized = vectorizer.transform(X_test).toarray()
n_states = 2
model = hmm.MultinomialHMM(n_components=n_states, n_iter=100)
model.fit(X_train_vectorized)
def predict_class(model, vectorizer, new_documents):
new_X = vectorizer.transform(new_documents).toarray()
logprob, states = model.decode(new_X, algorithm="viterbi")
return states
predictions = predict_class(model, vectorizer, X_test)
for doc, pred in zip(X_test, predictions):
print(f"Document: {doc} | Predicted Class: {pred}")
accuracy = np.mean(predictions == y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Output:

```
Document: Python is great | Predicted Class: 1
Document: I dislike errors | Predicted Class: 0
Accuracy: 1.00
```

Practical No - 11

Aim: Building a sentiment analysis system.

Step-1: Import Libraries and load dataset

Code:

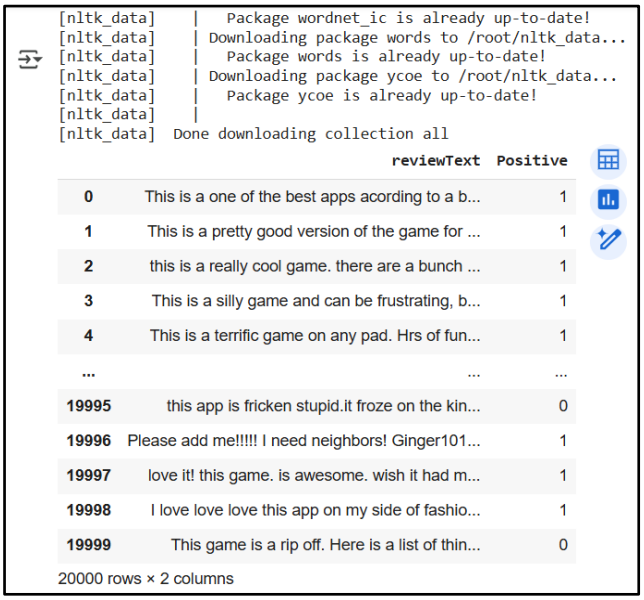
```
import pandas as pd
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

#download nltk corpus(first time only)
nltk.download('all')

#load the amazon review dataset
df = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/amazon.csv')
df
```

Output:

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package alpino to /root/nltk_data...
[nltk_data]    |   Package alpino is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger_eng to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger_eng is already
[nltk_data]    |       up-to-date!
```

```
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger_ru is already
[nltk_data]    |       up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger_rus to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger_rus is already
[nltk_data]    |       up-to-date!
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package basque_grammars is already up-to-date!
[nltk_data]    | Downloading package bcp47 to /root/nltk_data...
[nltk_data]    |   Package bcp47 is already up-to-date!
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package biocreative_ppi is already up-to-date!
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package bllip_wsj_no_aux is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package brown_tei to /root/nltk_data...
[nltk_data]    |   Package brown_tei is already up-to-date!
[nltk_data]    | Downloading package cess_cat to /root/nltk_data...
[nltk_data]    |   Package cess_cat is already up-to-date!
```

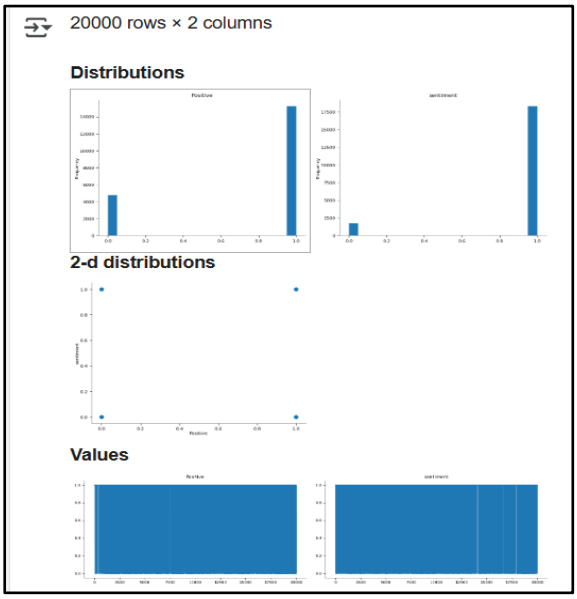
```
[nltk_data]    | Downloading package cess_esp to /root/nltk_data...
[nltk_data]    |   Package cess_esp is already up-to-date!
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package comparative_sentences to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package comparative_sentences is already up-to-
[nltk_data]    |       date!
[nltk_data]    | Downloading package comtrans to /root/nltk_data...
[nltk_data]    |   Package comtrans is already up-to-date!
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Package conll2000 is already up-to-date!
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Package conll2002 is already up-to-date!
[nltk_data]    | Downloading package conll2007 to /root/nltk_data...
[nltk_data]    |   Package conll2007 is already up-to-date!
[nltk_data]    | Downloading package crubadan to /root/nltk_data...
[nltk_data]    |   Package crubadan is already up-to-date!
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package dependency_treebank is already up-to-date!
[nltk_data]    | Downloading package dolch to /root/nltk_data...
[nltk_data]    |   Package dolch is already up-to-date!
```



```
[nltk_data]     |   Package wordnet_ic is already up-to-date!
[nltk_data]     | Downloading package words to /root/nltk_data...
[nltk_data]     |   Package words is already up-to-date!
[nltk_data]     | Downloading package ycoe to /root/nltk_data...
[nltk_data]     |   Package ycoe is already up-to-date!
[nltk_data]     |
[nltk_data]   Done downloading collection all
```

	reviewText	Positive
0	This is a one of the best apps acording to a b...	1
1	This is a pretty good version of the game for ...	1
2	this is a really cool game. there are a bunch ...	1
3	This is a silly game and can be frustrating, b...	1
4	This is a terrific game on any pad. Hrs of fun...	1
...	...	...
19995	this app is fricken stupid.it froze on the kin...	0
19996	Please add me!!!!! I need neighbors! Ginger101...	1
19997	love it! this game. is awesome. wish it had m...	1
19998	I love love love this app on my side of fashio...	1
19999	This game is a rip off. Here is a list of thin...	0

20000 rows × 2 columns



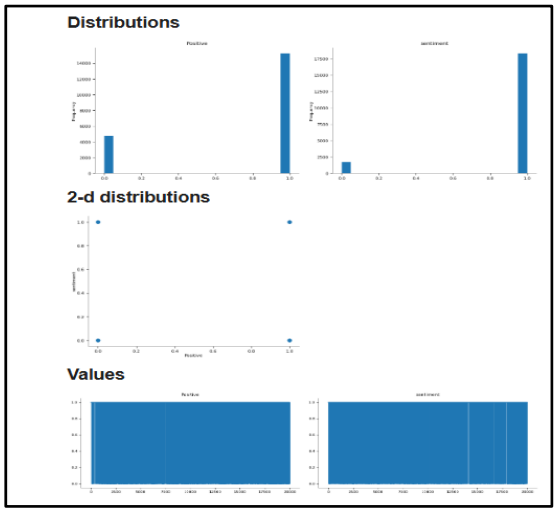
Step-2:Preprocess Text
Code:

```
def preprocess_text(text):
  #Tokenize the text
  tokens = word_tokenize(text.lower())

  #Remove the stopwords
  stop_words = set(stopwords.words('english'))
  filtered_tokens = [token for token in tokens if token not in stop_words]

  #OR
  #filtered_tokens = [token for token in tokens if token not in stopwords.words('english')]

  #Lemmatizethe tokens
  lemmatizer = WordNetLemmatizer()
  lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]

  #Join the tokens back into a string
  preprocessed_text = ' '.join(lemmatized_tokens)
  return preprocessed_text

#apply the function df
df['reviewText'] = df['reviewText'].apply(preprocess_text)
df
```

Output:

	reviewText	Positive
0	one best apps acording bunch people agree bomb...	1
1	pretty good version game free . lot different ...	1
2	really cool game . bunch level find golden egg...	1
3	silly game frustrating , lot fun definitely re...	1
4	terrific game pad . hr fun . grandkids love	1
...	...	...
19995	app fricken stupid.it froze kindle wont allow ...	0
19996	please add ! ! ! ! ! need neighbor ! ginger101...	1
19997	love ! game . awesome . wish free stuff house ...	1
19998	love love love app side fashion story fight wo...	1
19999	game rip . list thing make better & bull ; fir...	0

20000 rows × 2 columns



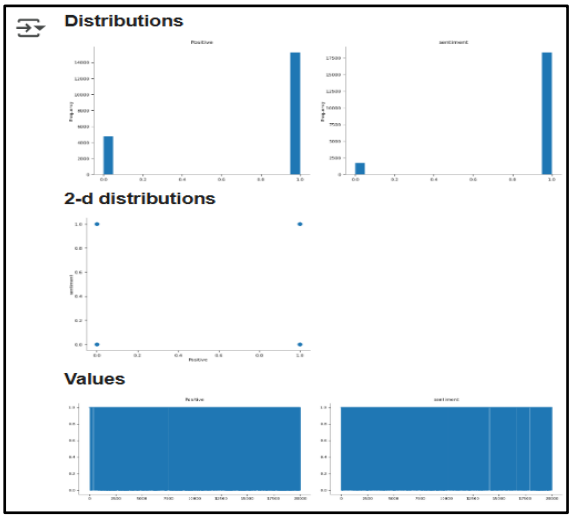
Step-3: NLTK Sentiment Analyzer
Code:

```
#Initialize NLTK Sentiment Analyzer
analyzer = SentimentIntensityAnalyzer()

#Create get_sentiment function
def get_sentiment(text):
  scores = analyzer.polarity_scores(text)
  sentiment = 1 if scores['pos'] > 0 else 0
  return sentiment

#Apply the get_sentiment function to the preprocessed text
df['sentiment'] = df['reviewText'].apply(get_sentiment)
df
```

Output:

	reviewText	Positive	sentiment
0	one best apps acording bunch people agree bomb...	1	1
1	pretty good version game free . lot different ...	1	1
2	really cool game . bunch level find golden egg...	1	1
3	silly game frustrating , lot fun definitely re...	1	1
4	terrific game pad . hr fun . grandkids love	1	1
...	...	...	...
19995	app fricken stupid.it froze kindle wont allow ...	0	0
19996	please add ! ! ! ! ! need neighbor ! ginger101...	1	1
19997	love ! game . awesome . wish free stuff house ...	1	1
19998	love love love app side fashion story fight wo...	1	1
19999	game rip . list thing make better & bull ; fir...	0	1

20000 rows × 3 columns



Practical No. 12

Aim: Creating a text summarization tool.

Code:

```
# Step 1: Install the necessary libraries
!pip install transformers
!pip install torch
# Step 2: Import libraries
from transformers import pipeline
# Step 3: Load the pre-trained BART model for summarization
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
# Step 4: Define the summarization function
def summarize_text(text, max_length=150, min_length=50):
    """
    Summarizes the input text using the BART model.

    Parameters:
    text (str): The text to summarize.
    max_length (int): The maximum length of the summarized text.
    min_length (int): The minimum length of the summarized text.

    Returns:
    str: The summarized text.
    """
    summary = summarizer(text, max_length=max_length, min_length=min_length,
do_sample=False)
    return summary[0]['summary_text']

# Step 5: Example text to summarize
text = """
The quick brown fox jumps over the lazy dog. This is a sentence that has been used in many typing
exercises for
decades. It contains every letter of the alphabet at least once, which makes it an excellent tool for
testing
font rendering and keyboard layouts. Additionally, it is often used in the testing of typewriters,
computer fonts,
and other printing devices. Over time, this simple sentence has become a common and beloved
phrase, known for its
brevity and completeness.
"""
# Step 6: Summarize the example text
summary = summarize_text(text)

# Step 7: Display the original text and summarized text
print("Original Text:\n", text)
print("\nSummarized Text:\n", summary)
```

Output:

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.49.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.26.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.29.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.26.0->transformers) (2025.3.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.26.0->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.1.31)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2025.3.0)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch)
```

```
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch) (3.0.2)
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.6 MB/s eta 0:00:00
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 35.5 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 37.7 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 19.9 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.4 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.3 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 9.6 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
```

```
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 8.4 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.7 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 45.4 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-c
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
```

```
      Uninstalling nvidia-cublas-cu12-12.5.3.2:
        Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your sessio
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 1.58k/1.58k [00:00<00:00, 73.9kB/s]
model.safetensors: 100% 1.63G/1.63G [00:13<00:00, 80.9MB/s]
generation_config.json: 100% 363/363 [00:00<00:00, 33.8kB/s]
vocab.json: 100% 899k/899k [00:00<00:00, 11.0MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 30.0MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 21.0MB/s]
Device set to use cpu
Your max_length is set to 150, but your input_length is only 105. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider de
```

```
Device set to use cpu
Your max_length is set to 150, but your input_length is only 105. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider de
Original Text:

The quick brown fox jumps over the lazy dog. This is a sentence that has been used in many typing exercises for
decades. It contains every letter of the alphabet at least once, which makes it an excellent tool for testing
font rendering and keyboard layouts. Additionally, it is often used in the testing of typewriters, computer fonts,
and other printing devices. Over time, this simple sentence has become a common and beloved phrase, known for its
brevity and completeness.


Summarized Text:
 The quick brown fox jumps over the lazy dog. This is a sentence that has been used in many typing exercises for decades. It contains every letter of the alphabet at least once. It j
```