Natural Language Processing Lab (MCALE243)

INDEX

Name of the faculty: Dr. Sulakshana Vispute/Mr. Ganesh Bhagwat

Experiment Number	Name of the experiment	Date	CO	Sign
1	To implement Tokenization of text.		CO1 CO2	
2	To implement Stop word removal.		CO1 CO2	
3	To implement Stemming of text		CO1 CO2	
4	To implement Lemmatization		CO1 CO2	
5	To implement N-gram model.		CO1 CO2	
6	To implement POS tagging.		CO2	
7	Building a custom NER system		CO2	
8	Creating and comparing different text representations		CO2 CO3	
9	Training and using word embeddings		CO3	
10	Implementing a text classifier		CO4	
11	Building a sentiment analysis system		C04	
12	Creating a text summarization tool		CO4	

PRACTICAL-1 *To implement Tokenization of text.*

1.Install Required Libraries

First, install `nltk` and `spacy` if you haven't already:

```
pip install nltk spacy
python -m spacy download en_core_web_sm
```

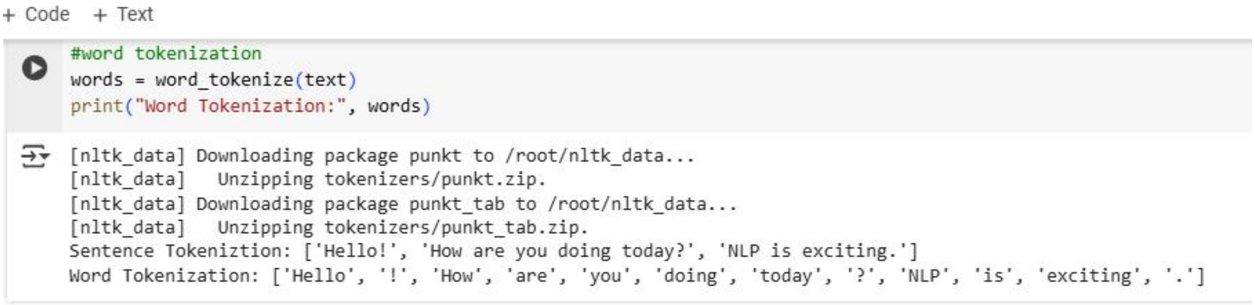
2. Tokenization Using NLTK

NLTK provides built-in functions for **word tokenization** and **sentence tokenization**.

Example: Tokenizing Sentences and Words

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

# Ensure 'punkt' is downloaded
nltk.download('punkt')
nltk.download('punkt_tab')

#sample text
text ="Hello! How are you doing today? NLP is exciting."

# Sentence Tokenization
sentences = sent_tokenize(text)
print ("Sentence Tokeniztion:", sentences)

#word tokenization
words = word_tokenize(text)
print("Word Tokenization:", words)
```

OUTPUT:



```
#word tokenization
words = word_tokenize(text)
print("Word Tokenization:", words)
```

```
[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.
Sentence Tokeniztion: ['Hello!', 'How are you doing today?', 'NLP is exciting.']
Word Tokenization: ['Hello', '!', 'How', 'are', 'you', 'doing', 'today', '?', 'NLP', 'is', 'exciting', '.']
```

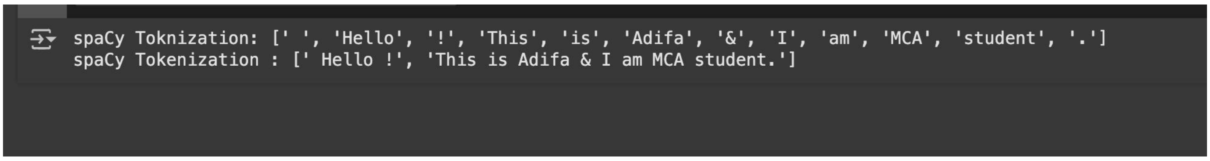
3. Tokenization Using spaCy

```
import spacy
#load spacys english model
nlp = spacy.load("en_core_web_sm")
text = " Hello !This is Adifa & I am MCA student."

#process the text
doc= nlp(text)
```

```
#word toknization
tokens = [token.text for token in doc]
print("spaCy Toknization:", tokens)
```

```
#sentence toknization
tokens = [sent.text for sent in doc.sents]
print("spaCy Tokenization :", tokens)
```

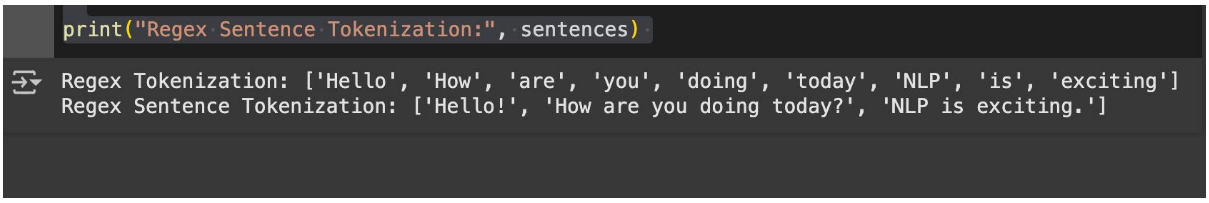
OUTPUT



```
spaCy Toknization: [' ', 'Hello', '!', 'This', 'is', 'Adifa', '&', 'I', 'am', 'MCA', 'student', '.']
spaCy Tokenization : [' Hello !', 'This is Adifa & I am MCA student.']
```

4. Custom Tokenization Using Regular Expressions

```
import re
text = "Hello! How are you doing today? NLP is
exciting."
# Tokenize using regex (splitting on spaces and
punctuation)
tokens = re.findall(r'\w+', text)
print("Regex Tokenization:", tokens)
# Tokenize sentences using regex
sentences = re.split(r'(?<=[.!?])\s+', text)
print("Regex Sentence Tokenization:", sentences)
```



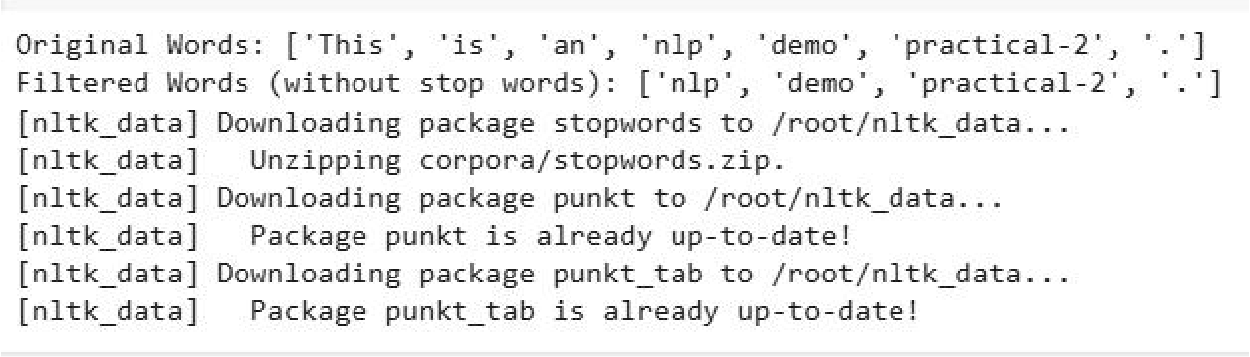
```
print("Regex Sentence Tokenization:", sentences)
```

```
Regex Tokenization: ['Hello', 'How', 'are', 'you', 'doing', 'today', 'NLP', 'is', 'exciting']
Regex Sentence Tokenization: ['Hello!', 'How are you doing today?', 'NLP is exciting.']
```

PRACTICAL-2-*To implement Stop word removal.*

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
#download stopword datasets

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')

#sample text
text= "This is an nlp demo practical-2."

 #tokenization the text into words
words= word_tokenize(text)

 #get english stop words
stop_words = set(stopwords.words('english'))

 #remove results
filtered_words = [word for word in words if word.lower() not in
stop_words]

 #print results
print("Original Words:", words)
print("Filtered Words (without stop words):", filtered_words)
```

OUTPUT:

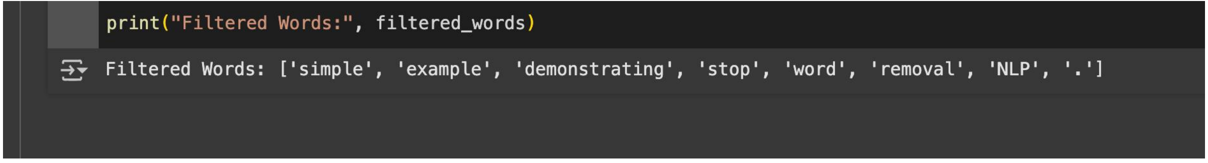


```
Original Words: ['This', 'is', 'an', 'nlp', 'demo', 'practical-2', '.']
Filtered Words (without stop words): ['nlp', 'demo', 'practical-2', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /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!
```

2.Alternative: Using SpaCy

```
import spacy

nlp = spacy.load("en_core_web_sm")
text = "This is a simple example demonstrating stop word removal in NLP."
```

```
# Process text with spaCy
doc = nlp(text)

# Remove stop words
filtered_words = [token.text for token in doc if not token.is_stop]

print("Filtered Words:", filtered_words)
```



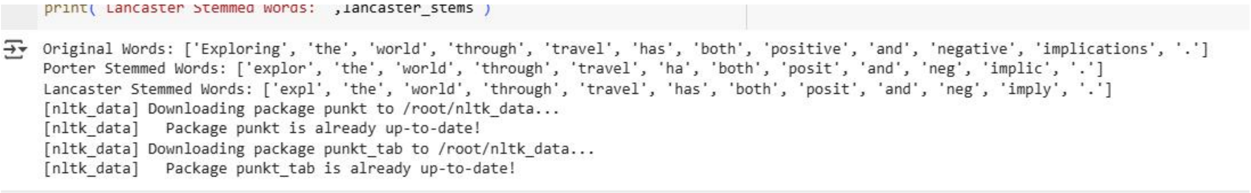
```
print("Filtered Words:", filtered_words)
```

```
Filtered Words: ['simple', 'example', 'demonstrating', 'stop', 'word', 'removal', 'NLP', '.']
```

PRACTICAL-3-*To implement Stemming of text*

1-Python Program For Stemming

```python
import nltk
from nltk.stem import PorterStemmer,LancasterStemmer
from nltk.tokenize import word_tokenize

#Download required dataset
nltk.download('punkt')
nltk.download('punkt_tab')

#sample text
text = "Exploring the world through travel has both positive and 
negative implications."

#Tokenize the text
tokens = word_tokenize(text)

#initialize stemmers
porter= PorterStemmer()
lancaster= LancasterStemmer()

#Apply stemming
porter_stems = [porter.stem(word)for word in tokens]
lancaster_stems = [lancaster.stem(word)for word in tokens]

#Print result
print("Original Words:" ,tokens)
print("Porter Stemmed Words:" , porter_stems)
print("Lancaster Stemmed Words:" ,lancaster_stems )
```



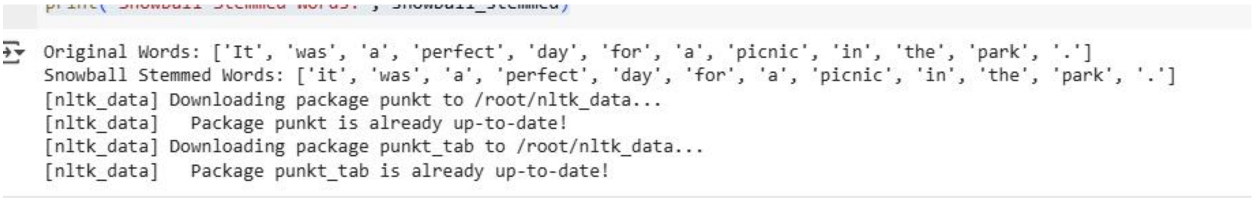
```
print("Lancaster Stemmed Words:" ,lancaster_stems )
```

```
Original Words: ['Exploring', 'the', 'world', 'through', 'travel', 'has', 'both', 'positive', 'and', 'negative', 'implications', '.']
Porter Stemmed Words: ['explor', 'the', 'world', 'through', 'travel', 'ha', 'both', 'posit', 'and', 'neg', 'implic', '.']
Lancaster Stemmed Words: ['expl', 'the', 'world', 'through', 'travel', 'has', 'both', 'posit', 'and', 'neg', 'imply', '.']
[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!
```

2- Snowball stemmer :

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer

#download dataset
nltk.download('punkt')
nltk.download('punkt_tab')

#sample text
text = "It was a perfect day for a picnic in the park."
```

```
#Tokenize the text
tokens = word_tokenize(text)

#Initialize snowball stemmer in english
snowball = SnowballStemmer("english")

#apply snow ball stemming
snowball_stemmed = [snowball.stem(word) for word in tokens]

print("Original Words:", tokens)
print("Snowball Stemmed Words:", snowball_stemmed)
```

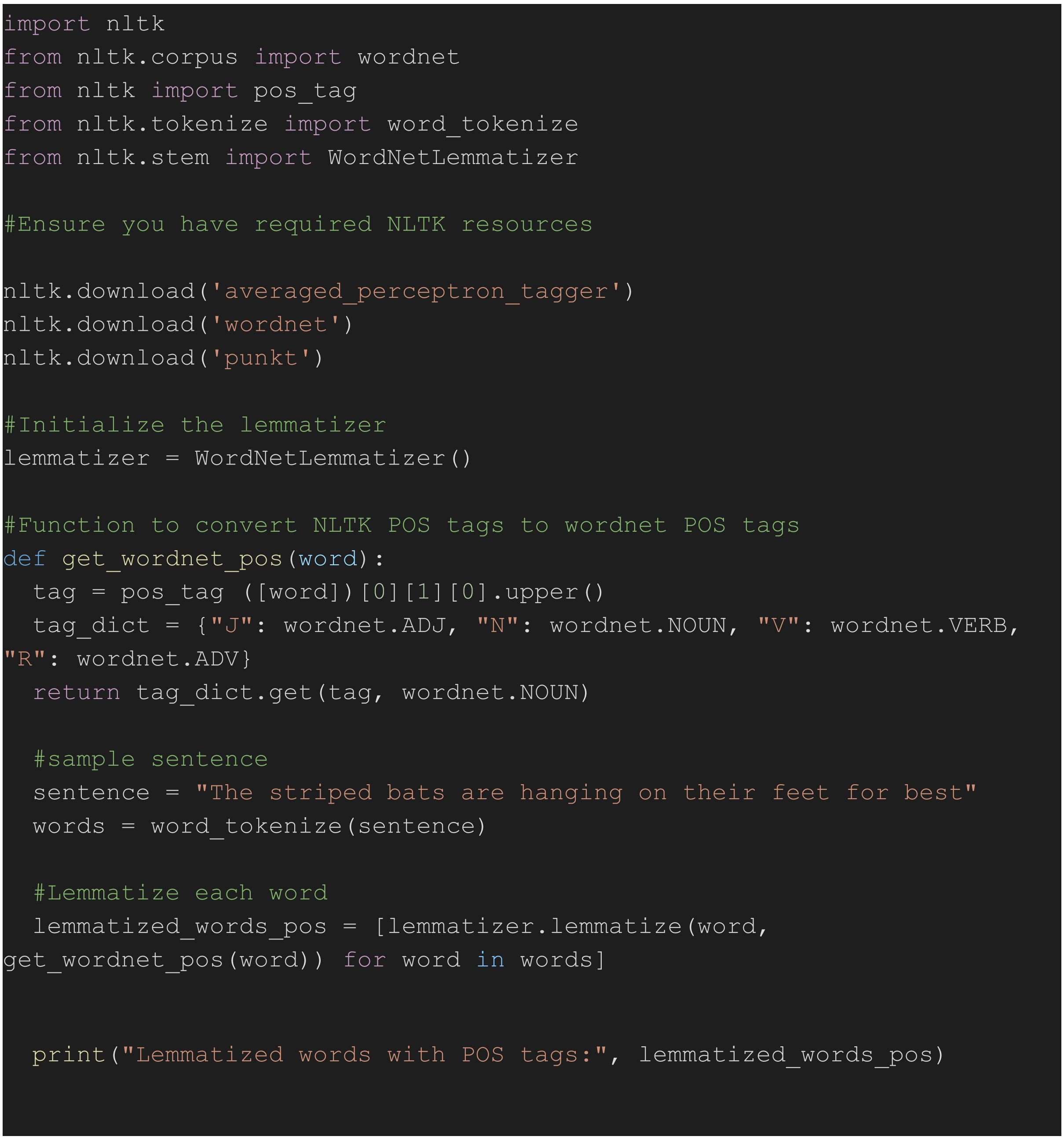


```
print("Snowball Stemmed Words:", snowball_stemmed)
```

```
Original Words: ['It', 'was', 'a', 'perfect', 'day', 'for', 'a', 'picnic', 'in', 'the', 'park', '.']
Snowball Stemmed Words: ['it', 'was', 'a', 'perfect', 'day', 'for', 'a', 'picnic', 'in', 'the', 'park', '.']
[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!
```

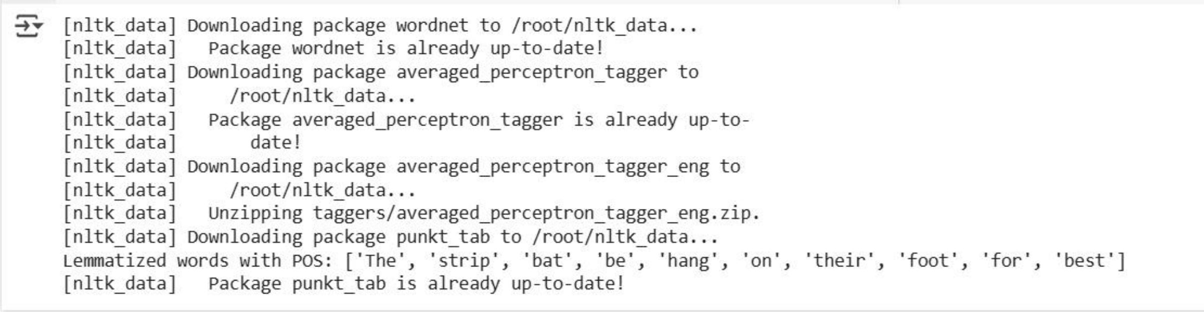
PRACTICAL- 4-*To implement Lemmatization*

1.Python Program for Lemmatization



```
import nltk
from nltk.corpus import wordnet
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

#Ensure you have required NLTK resources

nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')

#Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

#Function to convert NLTK POS tags to wordnet POS tags
def get_wordnet_pos(word):
  tag = pos_tag ([word])[0][1][0].upper()
  tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB,
"R": wordnet.ADV}
  return tag_dict.get(tag, wordnet.NOUN)

  #sample sentence
  sentence = "The striped bats are hanging on their feet for best"
  words = word_tokenize(sentence)

  #Lemmatize each word
  lemmatized_words_pos = [lemmatizer.lemmatize(word,
get_wordnet_pos(word)) for word in words]


  print("Lemmatized words with POS tags:", lemmatized_words_pos)
```

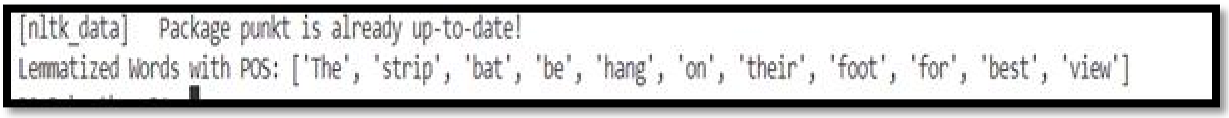
OUTPUT:



```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet 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-to-
[nltk_data]       date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
Lemmatized words with POS: ['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for', 'best']
[nltk_data]   Package punkt_tab is already up-to-date!
```

2.Lemmatization with POS Tagging (More Accurate)

```
import nltk
from nltk.corpus import wordnet
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Ensure you have the required NLTK resources
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to convert NLTK POS tags to WordNet POS tags
def get_wordnet_pos(word):
   tag = pos_tag([word])[0][1][0].upper()
   tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB,
"R": wordnet.ADV}
   return tag_dict.get(tag, wordnet.NOUN)  # Default to NOUN if unknown

# Sample sentence for lemmatization
sentence = "The striped bats are hanging on their feet for best viewing"
words = word_tokenize(sentence)  # Tokenize the sentence into words

# Apply Lemmatization with POS
lemmatized_words_pos = [lemmatizer.lemmatize(word,
get_wordnet_pos(word)) for word in words]

print("Lemmatized Words with POS:", lemmatized_words_pos)
```

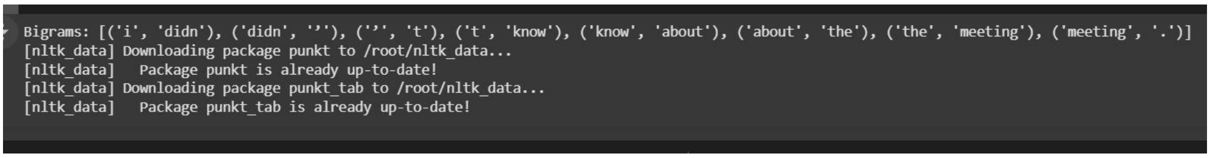


```
[nltk_data]    Package punkt is already up-to-date!
Lemmatized Words with POS: ['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for', 'best', 'view']
```

PRACTICAL-5-*To implement N-gram model.*

```
#use of N-gram model
from nltk import word_tokenize
from nltk.util import ngrams
import nltk

nltk.download('punkt')
nltk.download('punkt_tab')

text ="I didn’t know about the meeting."
tokens = word_tokenize(text.lower())

#generate bigram (n=2)

bigrams=list(ngrams(tokens,2))

#print the generated bigram
print ("Bigrams:", bigrams)
```

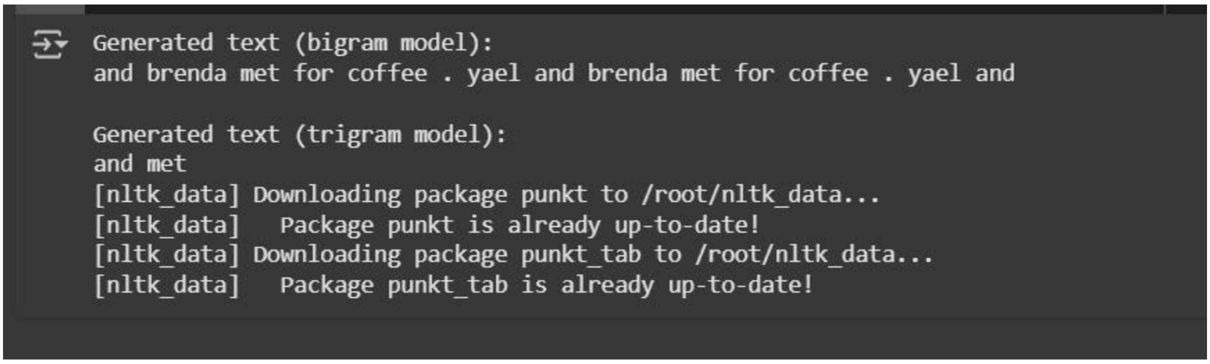
OUTPUT:



```
Bigrams: [('i', 'didn'), ('didn', '’'), ('’', 't'), ('t', 'know'), ('know', 'about'), ('about', 'the'), ('the', 'meeting'), ('meeting', '.')]
[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!
```

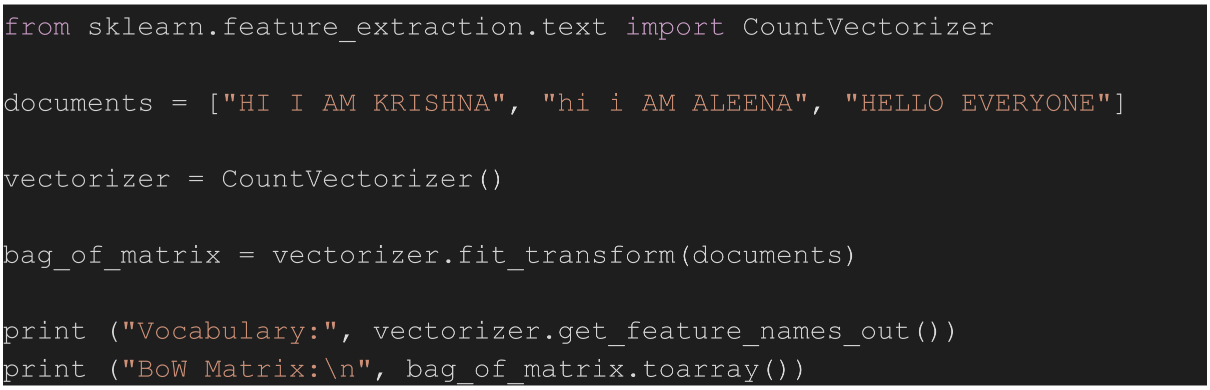
```
#program-2
import nltk
from nltk import bigrams, trigrams
from collections import defaultdict
import random

nltk.download('punkt')
nltk.download('punkt_tab')
def build_language_model(text, n=2):
    words = nltk.word_tokenize(text.lower())

    if n == 2:
        pairs = list(bigrams(words))
    elif n == 3:
        pairs = list(trigrams(words))
    else:
        raise ValueError("n must be 2 or 3")
    model = defaultdict(lambda: defaultdict(int))

    for pair in pairs:
        if n == 2:
            model[pair[0]][pair[1]] += 1
        else:
            model[(pair[0], pair[1])][pair[2]] += 1

    return model

def generate_text(model, num_words=20, start_word=None, n=2):
    if start_word is None:
        start_word = random.choice(list(model.keys()))

    words = [start_word] if n == 2 else list(start_word)

    for _ in range(num_words - n + 1):
        if n == 2:
            last_word = words[-1]
            if last_word not in model:
                break
            next_word = max(model[last_word], key=model[last_word].get)
        else:
            last_words = tuple(words[-2:])
            if last_words not in model:
                break
            next_word = max(model[last_words],
key=model[last_words].get)

        words.append(next_word)

    return ' '.join(words)

# Example usage
text = """
Zoe prepared dinner.
Yael and Brenda met for coffee.
"""

bigram_model = build_language_model(text, n=2)
trigram_model = build_language_model(text, n=3)

print("Generated text (bigram model):")
print(generate_text(bigram_model, num_words=15, start_word="and", n=2))

print("\nGenerated text (trigram model):")
print(generate_text(trigram_model, num_words=15, start_word=("and",
"met"), n=3))
```

OUTPUT:



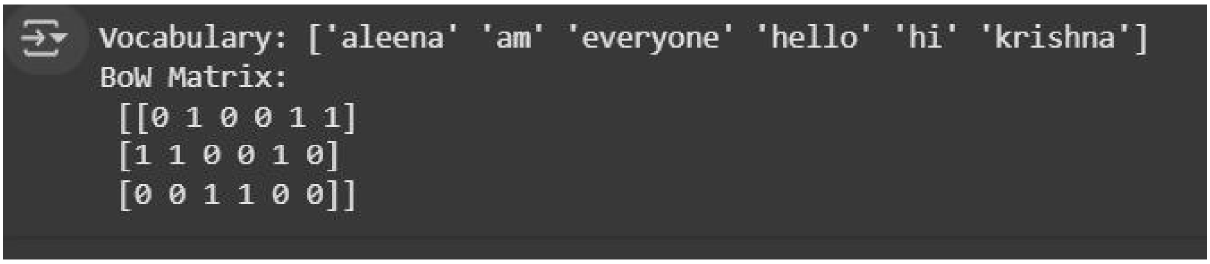
```
Generated text (bigram model):
and brenda met for coffee . yael and brenda met for coffee . yael and

Generated text (trigram model):
and met
[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-6-*To implement POS tagging.*

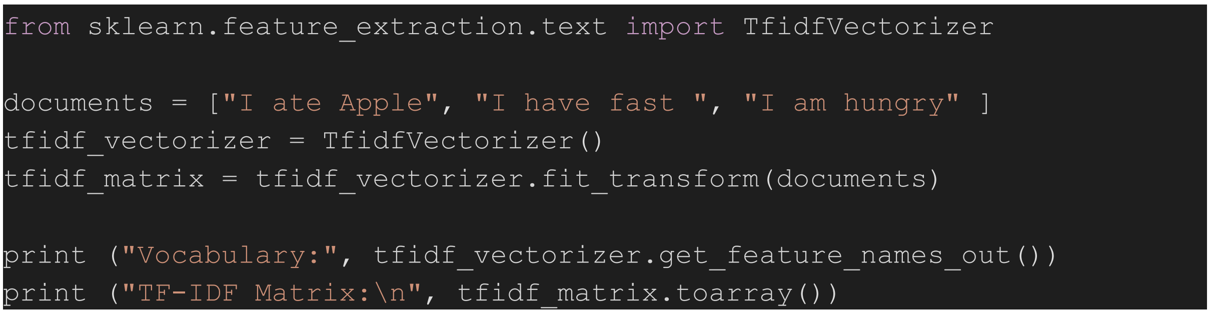


```
from sklearn.feature_extraction.text import CountVectorizer

documents = ["HI I AM KRISHNA", "hi i AM ALEENA", "HELLO EVERYONE"]

vectorizer = CountVectorizer()

bag_of_matrix = vectorizer.fit_transform(documents)

print ("Vocabulary:", vectorizer.get_feature_names_out())
print ("BoW Matrix:\n", bag_of_matrix.toarray())
```

OUTPUT:

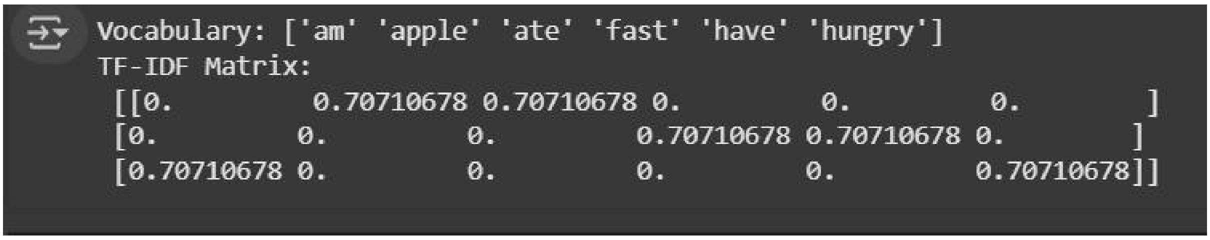


```
Vocabulary: ['aleena' 'am' 'everyone' 'hello' 'hi' 'krishna']
BoW Matrix:
 [[0 1 0 0 1 1]
 [1 1 0 0 1 0]
 [0 0 1 1 0 0]]
```

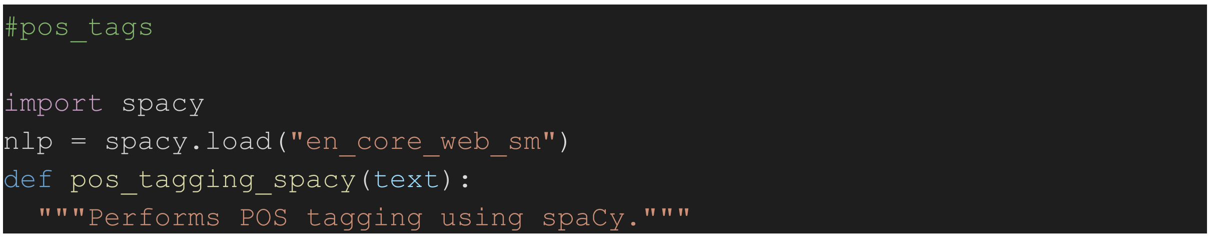


```
from sklearn.feature_extraction.text import TfidfVectorizer

documents = ["I ate Apple", "I have fast ", "I am hungry" ]
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

print ("Vocabulary:", tfidf_vectorizer.get_feature_names_out())
print ("TF-IDF Matrix:\n", tfidf_matrix.toarray())
```

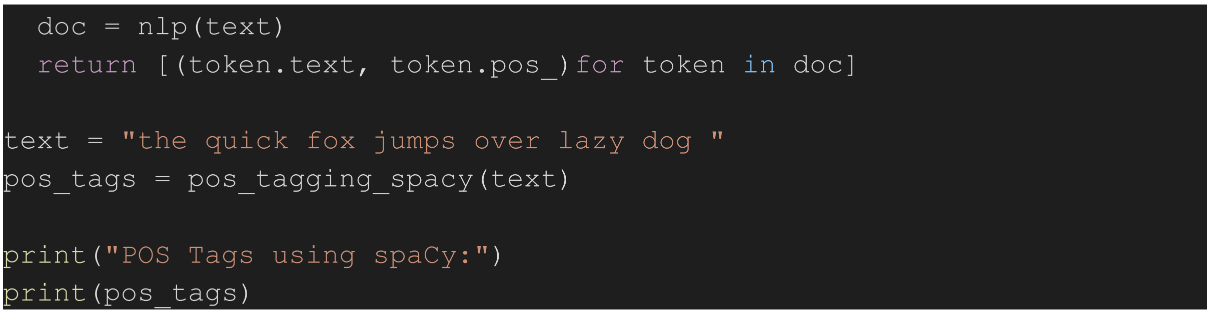
OUTPUT:



```
Vocabulary: ['am' 'apple' 'ate' 'fast' 'have' 'hungry']
TF-IDF Matrix:
 [[0.         0.70710678 0.70710678 0.         0.         0.        ]
 [0.         0.         0.         0.70710678 0.70710678 0.        ]
 [0.70710678 0.         0.         0.         0.         0.70710678]]
```

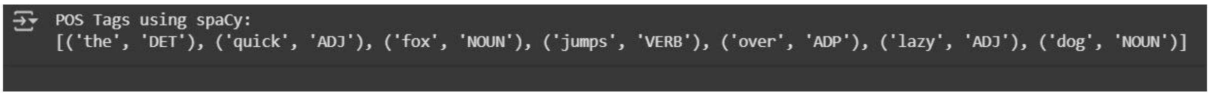


```
#pos_tags

import spacy
nlp = spacy.load("en_core_web_sm")
def pos_tagging_spacy(text):
  """Performs POS tagging using spaCy."""
```



```
  doc = nlp(text)
  return [(token.text, token.pos_)for token in doc]

text = "the quick fox jumps over lazy dog "
pos_tags = pos_tagging_spacy(text)

print("POS Tags using spaCy:")
print(pos_tags)
```

OUTPUT:



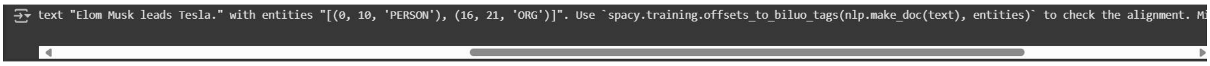
```
POS Tags using spaCy:
[('the', 'DET'), ('quick', 'ADJ'), ('fox', 'NOUN'), ('jumps', 'VERB'), ('over', 'ADP'), ('lazy', 'ADJ'), ('dog', 'NOUN')]
```

PRACTICAL-7-*Building a custom NER system*

```
import spacy
from spacy.training.example import Example
nlp = spacy.blank("en")

ner = nlp.add_pipe("ner", last=True)
ner.add_label("PERSON")
ner.add_label("ORG")

TRAIN_DATA = [
    ("Bill Gates founded Microsoft.", {"entities": [(0, 10,
"PERSON"),(19,20,"ORG")]}),
    ("Elom Musk leads Tesla.", {"entities": [(0, 10,
"PERSON"),(16,21,"ORG")]}),
    ("Steve Jobs created Apple.", {"entities": [(0, 10,
"PERSON"),(19,24,"ORG")]})

]
```

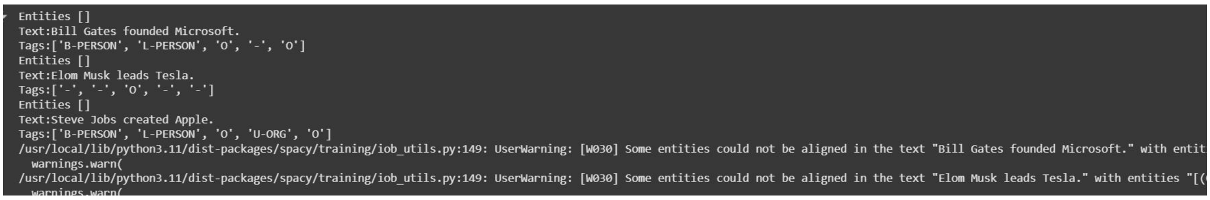
```
optimizer = nlp.begin_training()
for i in range(10):
    for text, annotations in TRAIN_DATA:
        example = Example.from_dict(nlp.make_doc(text), annotations)
        nlp.update([example], sgd=optimizer)
```

OUTPUT:



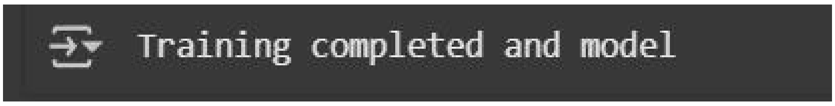
```
text "Elom Musk leads Tesla." with entities "[(0, 10, 'PERSON'), (16, 21, 'ORG')]". Use `spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), entities)` to check the alignment. Mi
```

```
for text, _ in TRAIN_DATA:
    doc = nlp.make_doc(text)
    tags = spacy.training.offsets_to_biluo_tags(doc,
annotations["entities"])
    print("Entities", [(ent.text, ent.label_) for ent in doc.ents])
    print(f"Text:{text}")
    print(f"Tags:{tags}")
```



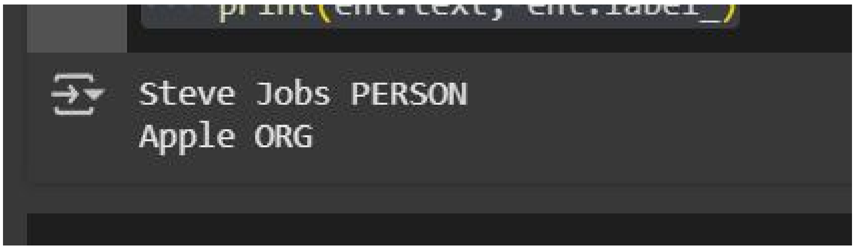
```
Entities []
Text:Bill Gates founded Microsoft.
Tags:['B-PERSON', 'L-PERSON', 'O', '-', 'O']
Entities []
Text:Elom Musk leads Tesla.
Tags:['-', '-', 'O', '-', '-']
Entities []
Text:Steve Jobs created Apple.
Tags:['B-PERSON', 'L-PERSON', 'O', 'U-ORG', 'O']
/usr/local/lib/python3.11/dist-packages/spacy/training/iob_utils.py:149: UserWarning: [W030] Some entities could not be aligned in the text "Bill Gates founded Microsoft." with entit
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/spacy/training/iob_utils.py:149: UserWarning: [W030] Some entities could not be aligned in the text "Elom Musk leads Tesla." with entities "[(
  warnings.warn(
```

```python
nlp.to_disk("custom_ner_model")
print("Training completed and model")
```



```
Training completed and model
```

```python
import spacy
nlp = spacy.load("custom_ner_model")
text = "Steve Jobs founded Apple."
doc = nlp(text)
for ent in doc.ents:
    print(ent.text, ent.label_)
```



```
    print(ent.text, ent.label_)
```

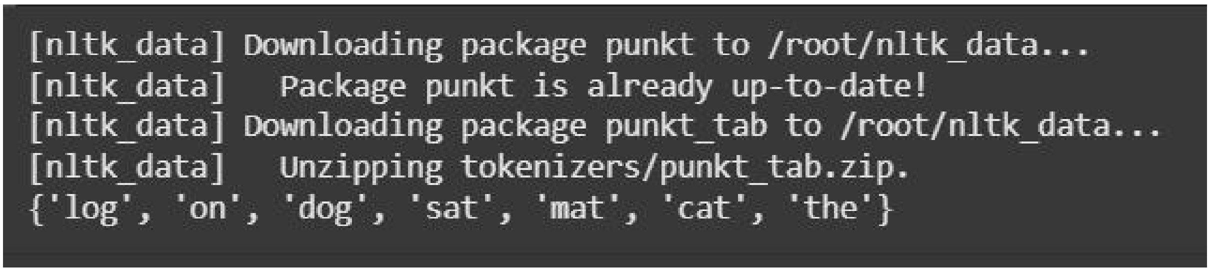
```
Steve Jobs PERSON
Apple ORG
```

PRACTICAL-8-*Creating and comparing different text representations*

a)CREATING BAG OF WORDS(BOW) TEXT REPRESENTATION

```
import nltk
import numpy as np
from collections import Counter

nltk.download('punkt')
nltk.download('punkt_tab')
texts=[
    "The cat sat on the mat",
    "The dog sat on the log"
]
tokenized_texts=[nltk.word_tokenize(text.lower()) for text in texts]
vocabulary=set(word for text in tokenized_texts for word in text)
print(vocabulary)

def get_bow_representation(tokens,vocabulary):
  return [tokens.count(word) for word in vocabulary]
  bow_vectors=[get_bow_representation(tokens,vocabulary) for text in
tokenized_texts]
  print("Bow Vectors:")
  print(np.array(bow_vectors))
```

OUTPUT:

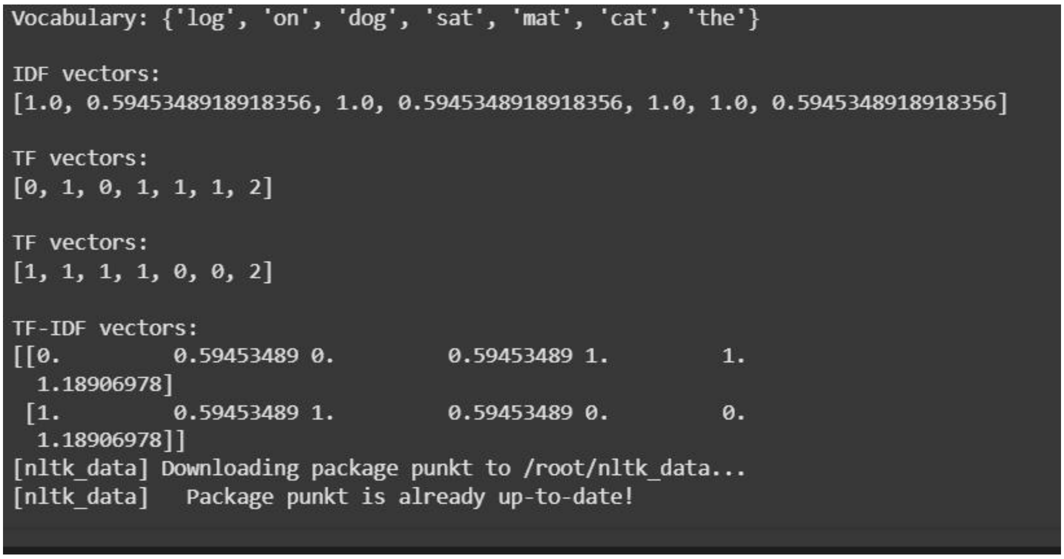


```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
{'log', 'on', 'dog', 'sat', 'mat', 'cat', 'the'}
```

b)Creating TF_IDF TEXT REPRESENTATIONS.

```
import nltk
import numpy as np
from collections import Counter
from math import log
nltk.download('punkt')

texts = [
    "The cat sat on the mat",
    "The dog sat on the log"
]

tokenized_texts = [nltk.word_tokenize(text.lower()) for text in texts]

vocabulary = set(word for text in tokenized_texts for word in text)
print("Vocabulary:", vocabulary)

def get_tf(tokens, vocabulary):
  tf_vector = [tokens.count(word) for word in vocabulary]
  print("\nTF vectors:")
  print(tf_vector)
  return tf_vector

def get_idf(vocabulary, docs):
  num_docs = len(docs)
  idf_vector = []
  for word in vocabulary:
    num_docs_with_word = sum(1 for doc in docs if word in doc)
    idf_value = log(num_docs / (1+num_docs_with_word))+1
    idf_vector.append(idf_value)
  return idf_vector

def get_tfidf(tokens, vocabulary, idf_vector):
  tf_vector = get_tf(tokens, vocabulary)
  tfidf_vector = [tf * idf for tf, idf in zip(tf_vector, idf_vector)]
  return tfidf_vector

idf_vector = get_idf(vocabulary, tokenized_texts)
print("\nIDF vectors:")
print(idf_vector)

tfidf_vectors = [get_tfidf(tokens, vocabulary, idf_vector) for tokens
in tokenized_texts]

print("\nTF-IDF vectors:")
print(np.array(tfidf_vectors))
```

OUTPUT:



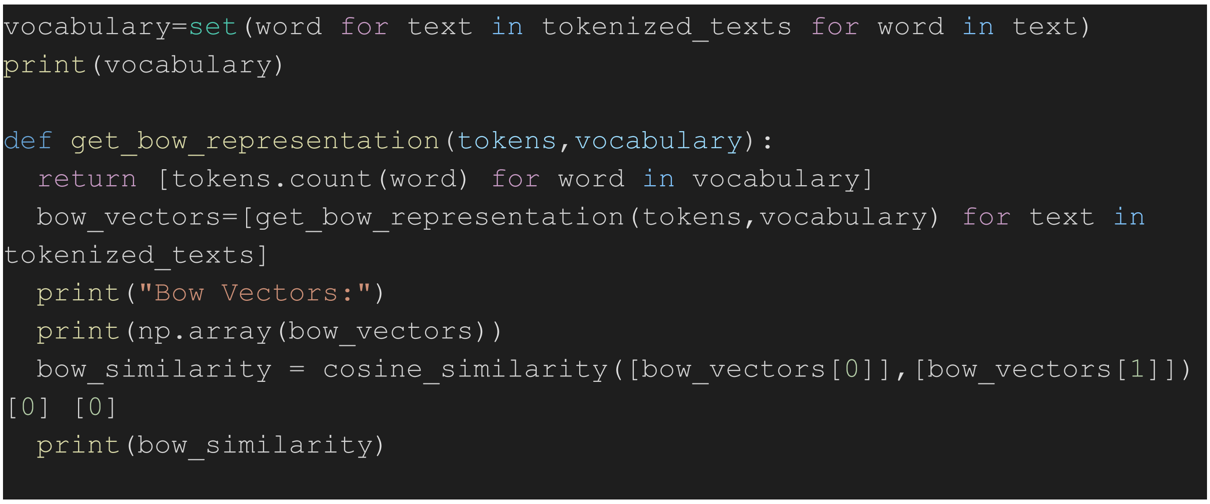
```
Vocabulary: {'log', 'on', 'dog', 'sat', 'mat', 'cat', 'the'}

IDF vectors:
[1.0, 0.5945348918918356, 1.0, 0.5945348918918356, 1.0, 1.0, 0.5945348918918356]

TF vectors:
[0, 1, 0, 1, 1, 1, 2]

TF vectors:
[1, 1, 1, 1, 0, 0, 2]

TF-IDF vectors:
[[0.         0.59453489 0.         0.59453489 1.         1.
  1.18906978]
 [1.         0.59453489 1.         0.59453489 0.         0.
  1.18906978]]
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

c) PROGRAM TO COMPARE TWO VECTORS OF BOW USING COSINE SIMILARITY

Code:

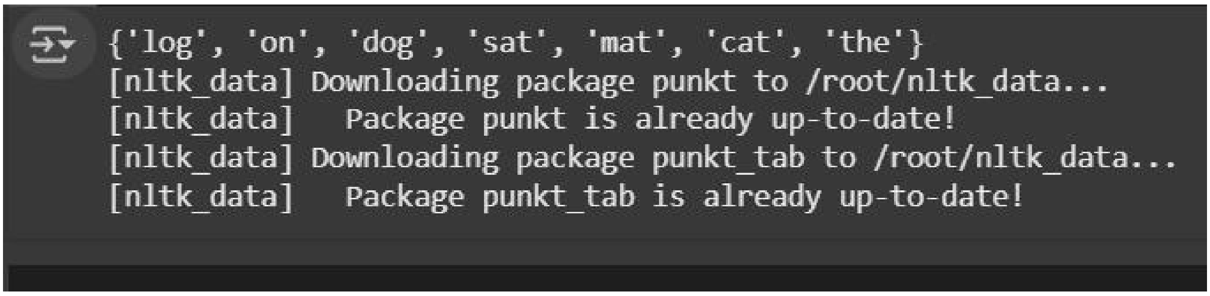
```
import nltk
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

nltk.download('punkt')

texts = [
    "The cat sat on the mat",
    "The dog sat on the log",
]
tokenized_texts = [nltk.word_tokenize(text.lower()) for text in texts]
vocabulary = set(word for text in tokenized_texts for word in text)
print(vocabulary)

def get_bow_representation(tokens, vocabulary):
  return [tokens.count(word) for word in vocabulary]

bow_vectors = [get_bow_representation(tokens, vocabulary) for tokens in
tokenized_texts]

print("BoW vectors:")
print(np.array(bow_vectors))

bow_similarity = cosine_similarity([bow_vectors[0]],[bow_vectors[1]])
[0] [0]
print("Cosine Similarity (BoW):")
print(bow_similarity)

print(" similarities:")
```

```
import nltk
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

nltk.download('punkt')
nltk.download('punkt_tab')
texts=[
    "The cat sat on the mat",
    "The dog sat on the log"
]
tokenized_texts=[nltk.word_tokenize(text.lower()) for text in texts]
```

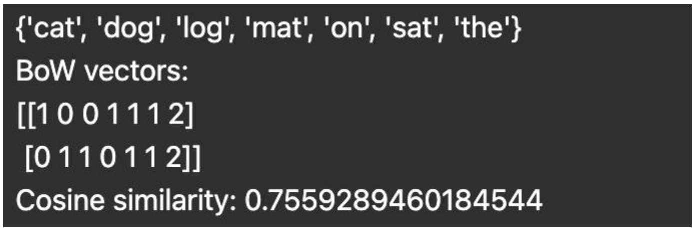


```
vocabulary=set(word for text in tokenized_texts for word in text)
print(vocabulary)

def get_bow_representation(tokens,vocabulary):
  return [tokens.count(word) for word in vocabulary]
  bow_vectors=[get_bow_representation(tokens,vocabulary) for text in
tokenized_texts]
  print("Bow Vectors:")
  print(np.array(bow_vectors))
  bow_similarity = cosine_similarity([bow_vectors[0]],[bow_vectors[1]])
[0] [0]
  print(bow_similarity)
```

OUTPUT:



```
{'log', 'on', 'dog', 'sat', 'mat', 'cat', 'the'}
[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!
```



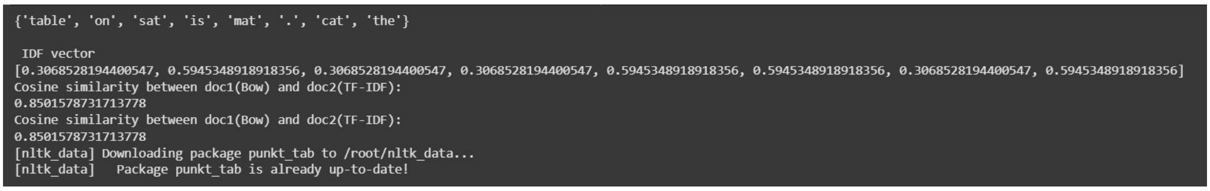
```
{'cat', 'dog', 'log', 'mat', 'on', 'sat', 'the'}
BoW vectors:
[[1 0 0 1 1 1 2]
 [0 1 1 0 1 1 2]]
Cosine similarity: 0.7559289460184544
```

d)PROGRAM TO COMPARE A BOW VECTOR WITH A TF-IDF VECTOR USING COSINE SIMILARITY

Code:

```
import nltk
import numpy as np
from collections import Counter
from math import log
from sklearn.metrics.pairwise import cosine_similarity
nltk.download('punkt_tab')
texts=[
    "The cat sat on the mat.",
    "The mat is on the table."

]
tokenized_texts=[nltk.word_tokenize(text.lower()) for text in texts]
vocabulary=set(word for text in tokenized_texts for word in text)
print(vocabulary)

def get_bow_representation(tokens,vocabulary):
  return [tokens.count(word)for word in vocabulary]
bow_vectors=[get_bow_representation(text,vocabulary)for text in
tokenized_texts]

def get_tf(tokens,vocabulary):
  return [tokens.count(word)for word in vocabulary]
def get_idf(vocabulary,docs):
  idf_vector=[]
  for word in vocabulary:
    num_docs_with_word=sum(1 for doc in docs if word in doc)
    idf_value=log(num_docs_with_word/(1+num_docs_with_word))+1
    idf_vector.append(idf_value)
  return idf_vector
def get_tfidf(tokens,vocabulary,idf_vector):
  tf_vector=get_tf(tokens,vocabulary)
  tfidf_vector=[tf*idf for tf,idf in zip(tf_vector,idf_vector)]
  return tfidf_vector
idf_vector = get_idf(vocabulary, tokenized_texts)
print("\n IDF vector")
print(idf_vector)
tfidf_vectors=[get_tfidf(text,vocabulary,idf_vector) for text in
tokenized_texts]
bow_similarity=cosine_similarity([bow_vectors[0]],
[tfidf_vectors[1]])[0][0]
print("Cosine similarity between doc1(Bow) and doc2(TF-IDF):")
print(bow_similarity)
bow_similarity=cosine_similarity([bow_vectors[1]],
[tfidf_vectors[0]])[0][0]
print("Cosine similarity between doc1(Bow) and doc2(TF-IDF):")
print(bow_similarity)
```

OUTPUT:

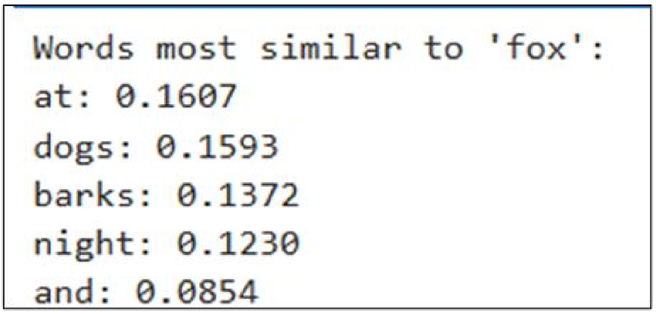


```
{'table', 'on', 'sat', 'is', 'mat', '.', 'cat', 'the'}

 IDF vector
[0.3068528194400547, 0.5945348918918356, 0.3068528194400547, 0.3068528194400547, 0.5945348918918356, 0.5945348918918356, 0.3068528194400547, 0.5945348918918356]
Cosine similarity between doc1(Bow) and doc2(TF-IDF):
0.8501578731713778
Cosine similarity between doc1(Bow) and doc2(TF-IDF):
0.8501578731713778
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

PRACTICAL-9-Training and using word embeddings

```
!pip install genism
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
# Download the punkttokenizer models from NLTK
nltk.download('punkt')
# Function to train Word2Vec model
def train_word_embeddings(sentences):
    # Tokenize sentences using NLTK word_tokenize and convert to lowercase
    tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
    # Train Word2Vec model
    model = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5,
min_count=1, workers=4)
    return model
# Function to use trained Word2Vec model and find similar words
def use_word_embeddings(model, word, top_n=5):
    try:
        # Get the top N similar words to the input word
        similar_words = model.wv.most_similar(word, topn=top_n)
        print(f"Words most similar to '{word}':")
        for w, score in similar_words:
            print(f"{w}: {score:.4f}")
    except KeyError:
                print(f"'{word}' not in vocabulary")

# Example usage
sentences = [
    "The quick brown fox jumps over the lazy dog",
    "A fox is a cunning animal",
    "The dog barks at night",
    "Foxes and dogs are different species"
]

# Train Word2Vec model using the provided sentences
model = train_word_embeddings(sentences)

# Use the trained model to find words similar to "fox"
use_word_embeddings(model, "fox")
```

OUTPUT:

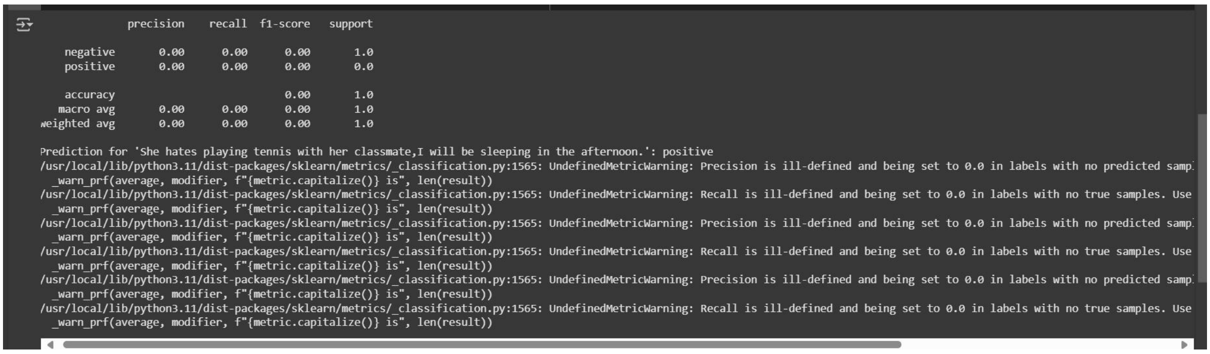


```
Words most similar to 'fox':
at: 0.1607
dogs: 0.1593
barks: 0.1372
night: 0.1230
and: 0.0854
```

PRACTICAL-10 -Implementing a text classifier

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

def train_test_classifier(X, y):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
  vectorizer = CountVectorizer()
  X_train_vectorized = vectorizer.fit_transform(X_train)
  X_test_vectorized = vectorizer.transform(X_test)
  classifier = MultinomialNB()
  classifier.fit(X_train_vectorized, y_train)
  y_pred = classifier.predict(X_test_vectorized)
  print(classification_report(y_test, y_pred))
  return vectorizer, classifier

def classify_text(text, vectorizer, classifier):
  text_vectorized = vectorizer.transform([text])
  prediction = classifier.predict(text_vectorized)
  return prediction[0]

# The X variable was previously defined as a single string containing all the texts.
# To fix the error, we need to define X as a list of strings, where each string represents a separate document.
X = [
    "I am competent in achieving my objectives.",
    "He doesn’t work anywhere..",
    "Today the sun is shining brilliantly.",
    "I do not have my Email ID yet.",
    "I am certain of my skills."
]
y = ["positive", "negative", "positive", "negative", "positive"]
new_text = "She hates playing tennis with her classmate,I will be sleeping in the afternoon."
vectorizer, classifier = train_test_classifier(X,y)
prediction = classify_text(new_text, vectorizer, classifier)
print(f"Prediction for '{new_text}': {prediction}")
```

OUTPUT:



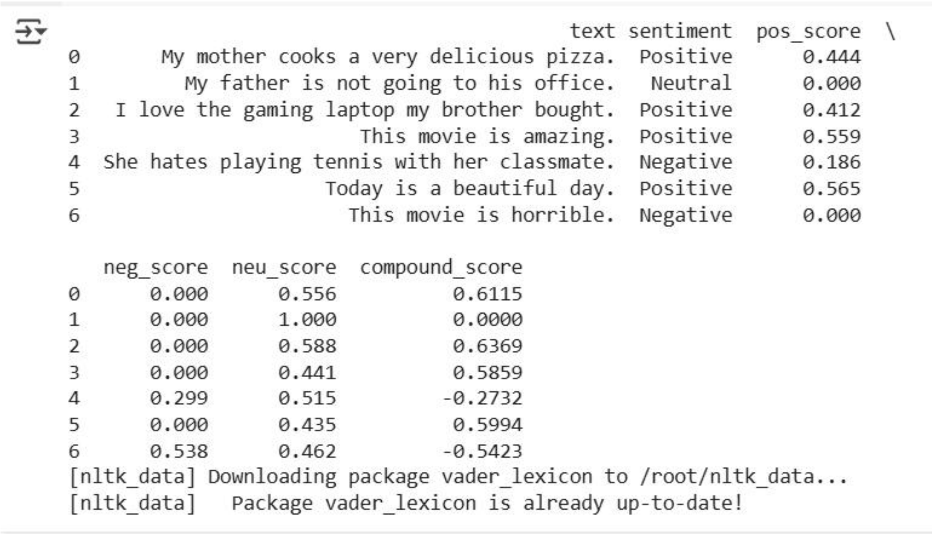
```
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
    positive       0.00      0.00      0.00       0.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0

Prediction for 'She hates playing tennis with her classmate,I will be sleeping in the afternoon.': positive
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samp
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samp
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samp
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

PRACTICAL-11-*Building a sentiment analysis system*

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import pandas as pd
nltk.download('vader_lexicon')
def analyze_sentiment(text):
  sia = SentimentIntensityAnalyzer()
  sentiment_scores=sia.polarity_scores(text)
  if sentiment_scores['compound'] >= 0.1:
    sentiment="Positive"
  elif sentiment_scores['compound'] <= -0.1:
    sentiment="Negative"
  else:
    sentiment="Neutral"

  return sentiment,sentiment_scores
def analyze_sentiments(texts):
  results=[]
  for text in texts:
    sentiment,scores=analyze_sentiment(text)
    results.append({
        "text":text,
        "sentiment":sentiment,
        "pos_score":scores['pos'],
        "neg_score":scores['neg'],
        "neu_score":scores['neu'],
        "compound_score":scores['compound']
        })
```

```
  return pd.DataFrame(results)
texts=[
    "My mother cooks a very delicious pizza.",
    "My father is not going to his office.",
    "I love the gaming laptop my brother bought.",
    "This movie is amazing.",
    "She hates playing tennis with her classmate.",
    "Today is a beautiful day.",
    "This movie is horrible."
]
results_df=analyze_sentiments(texts)
print(results_df)
```

OUTPUT:



```
                                          text sentiment  pos_score  \
0       My mother cooks a very delicious pizza.  Positive      0.444   
1         My father is not going to his office.   Neutral      0.000   
2   I love the gaming laptop my brother bought.  Positive      0.412   
3                        This movie is amazing.  Positive      0.559   
4  She hates playing tennis with her classmate.  Negative      0.186   
5                     Today is a beautiful day.  Positive      0.565   
6                       This movie is horrible.  Negative      0.000   

   neg_score  neu_score  compound_score  
0      0.000      0.556          0.6115  
1      0.000      1.000          0.0000  
2      0.000      0.588          0.6369  
3      0.000      0.441          0.5859  
4      0.299      0.515         -0.2732  
5      0.000      0.435          0.5994  
6      0.538      0.462         -0.5423  
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

PRACTICAL-12- *Creating a text summarization tool*

```
!pip install transformers
!pip install torch
from transformers import pipeline
def summarize_text(text, max_length=150, min_length=50):
  summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
  summary = summarizer(text, max_length=max_length, min_length=min_length, do_sample=False)
  return summary[0]['summary_text']
long_text = """
Climate change is one of the most pressing issues facing our planet today. It refers to long-term shifts in temperatures and weather patterns, mainly caused by human activities,
especially the burning of fossil fuels. These activities release greenhouse gases into the atmosphere, trapping heat and causing the Earth's average temperature to rise.
The consequences of climate change are far-reaching and include more frequent and severe weather events, rising sea levels, and disruptions to ecosystems. To address this global
challenge, countries and organizations worldwide are working on strategies to reduce greenhouse gas emissions and transition to cleaner energy sources.
Individual actions, such as reducing energy consumption and adopting sustainable practices, also play a crucial role in mitigating the effects of climate change."""

summary = summarize_text(long_text)
print("Original text length:", len(long_text))
print("Summary length:", len(summary))
print("\nSummary:")
print(summary)
```

OUTPUT:

```
Original text length: 843

Summary length: 307


Summary:

Climate change is one of the most pressing issues facing our planet today. It refers to long-
term shifts in temperatures and weather patterns, mainly caused by human activities. These
activities release greenhouse gases into the atmosphere, trapping heat and causing the Earth's
average temperature to rise.
```