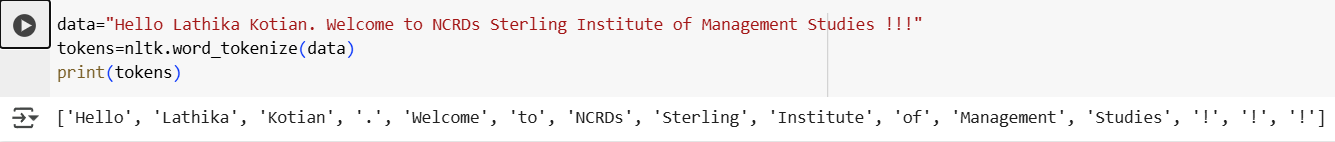
# **PRACTICAL - 1**

**Aim: Write a program to implement Tokenization of text**

**NLTK Code :**

|  |
| --- |
| # 1.) TOKENIZATION using nltk  import nltk  nltk.download()  data="Hello Lathika Kotian. Welcome to NCRDs Sterling Institute of Management Studies !!!"  tokens=nltk.word\_tokenize(data)  print(tokens) |

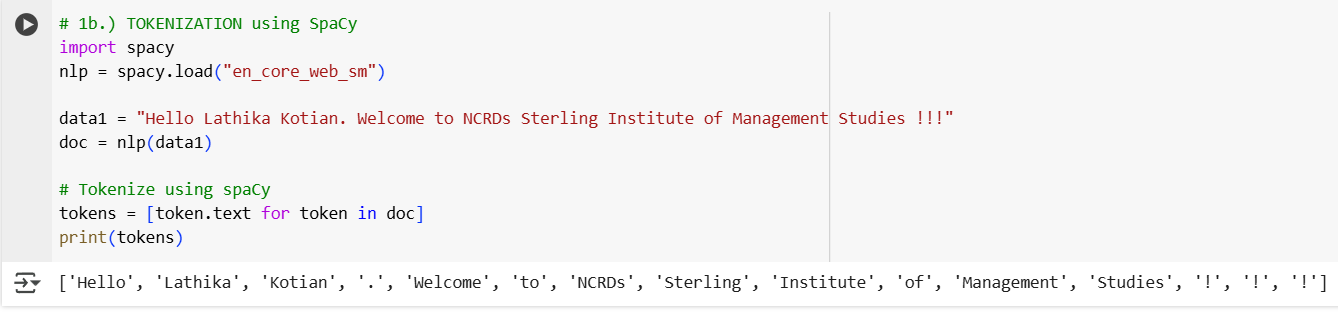
**Output:**

****

**SpaCy Code:**

|  |
| --- |
| # 1b.) TOKENIZATION using SpaCy  import spacy  nlp = spacy.load("en\_core\_web\_sm")  data1 = "Hello Lathika Kotian. Welcome to NCRDs Sterling Institute of Management Studies !!!"  doc = nlp(data1)  # Tokenize using spaCy  tokens = [token.text for token in doc]  print(tokens) |

**Output:**

****

# 

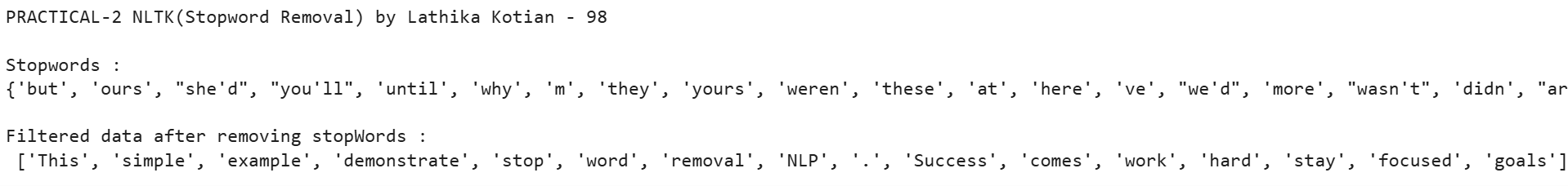
# **PRACTICAL - 2**

**Aim: Write a program to implement Stop word removal.**

**NLTK Code:**

|  |
| --- |
| print("PRACTICAL-2 NLTK(Stopword Removal) by Lathika Kotian - 98\n")  #2.) STOP WORD REMOVAL using nltk  from nltk.tokenize import sent\_tokenize,word\_tokenize  from nltk.corpus import stopwords  nltk.download('stopwords')  stopWords=set(stopwords.words('english'))  print(f'Stopwords : \n{stopWords}')  data="This is a simple example to demonstrate stop word removal in NLP. Success comes to those who work hard and stay focused on their goals"  tokens=nltk.word\_tokenize(data)  filtered\_Data=[]  for w in tokens:  if w not in stopWords:  filtered\_Data.append(w)  print("\nFiltered data after removing stopWords :\n",filtered\_Data) |

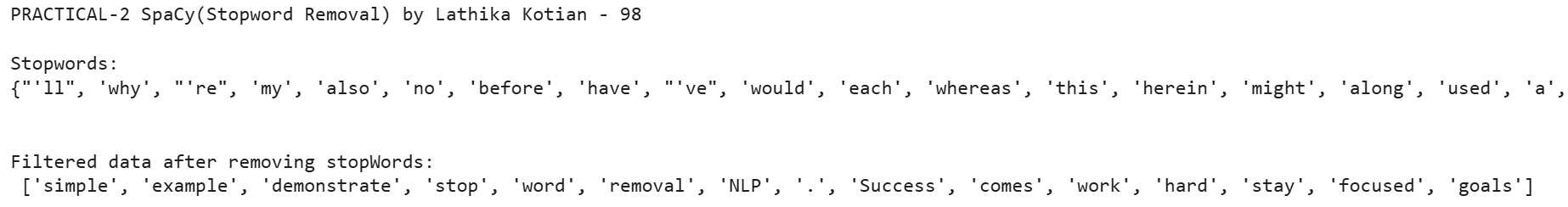
**Output:**

****

**Spacy Code:**

|  |
| --- |
| print("PRACTICAL-2 SpaCy(Stopword Removal) by Lathika Kotian - 98\n")  #2.) STOP WORD REMOVAL using SpaCy  import spacy  nlp = spacy.load("en\_core\_web\_sm") # Load the spaCy English model  data = "This is a simple example to demonstrate stop word removal in NLP. Success comes to those who work hard and stay focused on their goals"  doc = nlp(data)# Process the text  # List of spaCy's stop words  stop\_words = nlp.Defaults.stop\_words  print(f"Stopwords:\n{stop\_words}\n")  # Filter out stop words  filtered\_data = [token.text for token in doc if not token.is\_stop]  print("\nFiltered data after removing stopWords:\n",filtered\_data) |

**Output:**

****

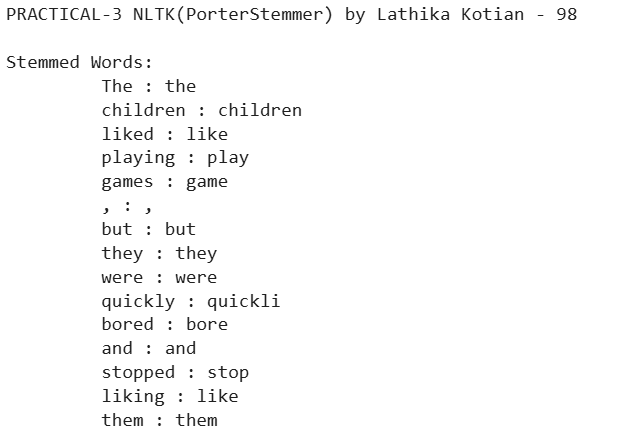
# **PRACTICAL - 3**

**Aim: Write a program to implement Stemming.**

**Code:**

|  |
| --- |
| print("PRACTICAL-3 NLTK(PorterStemmer) by Lathika Kotian - 98\n")  import nltk  from nltk.stem import PorterStemmer  from nltk.tokenize import word\_tokenize  nltk.download('punkt')  port\_stemmer = PorterStemmer()  default\_sentence = "The children liked playing games, but they were quickly bored and stopped liking them."  try:  with open("PortStem.txt", "r") as file:  text\_data = file.read().strip()  if not text\_data:  print("File is empty. Using default sentence.\n")  text\_data = default\_sentence  except FileNotFoundError:  print("File not found. Using default sentence.\n")  text\_data = default\_sentence  # Tokenize and stem  tokens = word\_tokenize(text\_data)  print("Stemmed Words:")  for word in tokens:  print("\t",word, ":", port\_stemmer.stem(word)) |

**Output:**

****

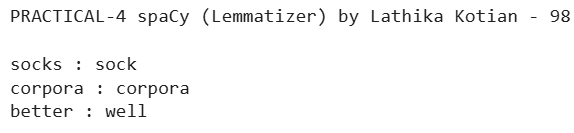
# **PRACTICAL - 4**

**Aim: Write a program to implement Lemmatization**

**Spacy Code:**

|  |
| --- |
| #4.) LEMMATIZATION using Spacy  print("PRACTICAL-4 spaCy (Lemmatizer) by Lathika Kotian - 98\n")  import spacy  nlp = spacy.load("en\_core\_web\_sm")  words = ["socks", "corpora", "better"]  # Process each word and print its lemmatized form  for word in words:  doc = nlp(word) # Process word through spaCy  print(f"{word} :", doc[0].lemma\_) |

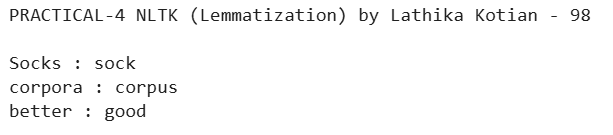
**Output:**

****

**NLTK Code:**

|  |
| --- |
| #4.) LEMMATIZATION using NLTK  print("PRACTICAL-4 NLTK (Lemmatization) by Lathika Kotian - 98\n")  from nltk.stem import WordNetLemmatizer  lemmati=WordNetLemmatizer()  print("Socks :",lemmati.lemmatize("socks"))  print("corpora :",lemmati.lemmatize("corpora"))  print("better :",lemmati.lemmatize("better",pos="a")) |

**Output:**

****

# 

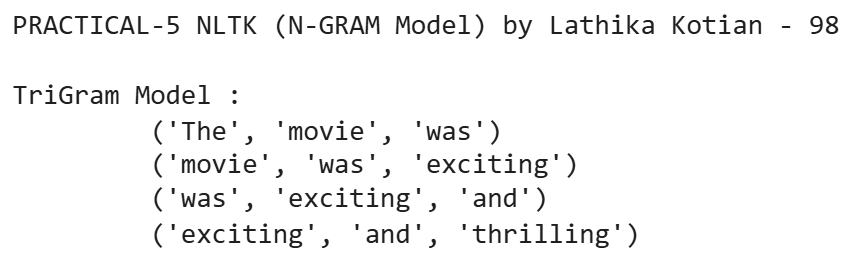
# **PRACTICAL - 5**

**Aim: Write a program to implement the N-gram model**

**NLTK Code: TriGram Model**

|  |
| --- |
| #5.) N-GRAM Model -Trigram  print("PRACTICAL-5 NLTK (N-GRAM Model) by Lathika Kotian - 98\n")  import nltk  nltk.download('punkt')  from nltk.util import ngrams  from nltk.tokenize import word\_tokenize  data ="The movie was exciting and thrilling."  tokens = nltk.word\_tokenize(data)  Ngram = ngrams(tokens, 3)  print("TriGram Model : ")  for gram in Ngram:  print("\t",gram) |

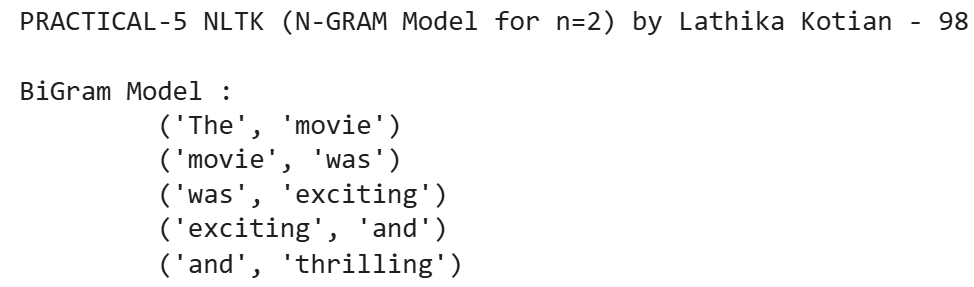
**Output:**

****

**NLTK Code: BiGram Model**

|  |
| --- |
| #5.) N-GRAM Model-Bigram  print("PRACTICAL-5 NLTK (N-GRAM Model for n=2) by Lathika Kotian - 98\n")  import nltk  nltk.download('punkt')  from nltk.util import ngrams  from nltk.tokenize import word\_tokenize  data ="The movie was exciting and thrilling."  tokens = nltk.word\_tokenize(data)  Ngram = ngrams(tokens, 2)  print("BiGram Model : ")  for gram in Ngram:  print("\t",gram) |

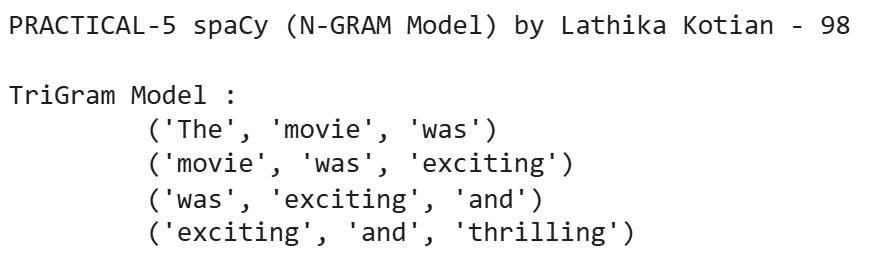
**Output:**

****

**Spacy Code:**

|  |
| --- |
| #5.) N-GRAM Model using SpaCy  print("PRACTICAL-5 spaCy (N-GRAM Model) by Lathika Kotian - 98\n")  import spacy  from spacy.util import minibatch  nlp = spacy.load("en\_core\_web\_sm")  data = "The movie was exciting and thrilling"  doc = nlp(data)  tokens = [token.text for token in doc] # Convert tokens to list of strings  n = 3  ngrams = zip(\*[tokens[i:] for i in range(n)])  print("TriGram Model : ")  for gram in ngrams:  print("\t",gram) |

**Output:**

****

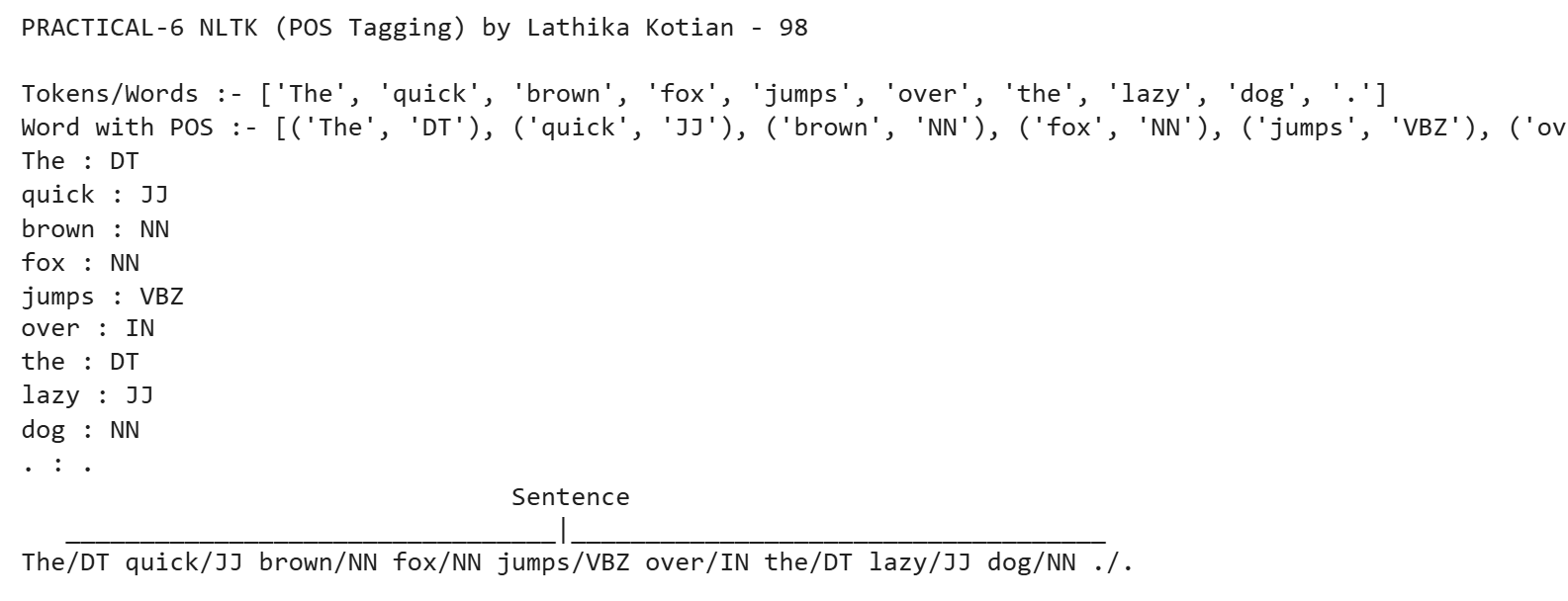
# **PRACTICAL - 6**

**Aim: Write a program to implement POS tagging**

**NLTK Code:**

|  |
| --- |
| print("PRACTICAL-6 NLTK (POS Tagging) by Lathika Kotian - 98\n")  import nltk  from nltk.tokenize import word\_tokenize  from nltk import pos\_tag  nltk.download('punkt')  nltk.download('averaged\_perceptron\_tagger')  text = "The quick brown fox jumps over the lazy dog."  words = word\_tokenize(text)  print("Tokens/Words :-", words)  pos\_tags = pos\_tag(words)  print("Word with POS :-", pos\_tags)  for word, tag in pos\_tags:  print(f"{word} : {tag}")  # Create and display a POS-tagged tree structure  tree\_obj = nltk.Tree('Sentence', [(word, tag) for word, tag in pos\_tags])  tree\_obj.pretty\_print() |

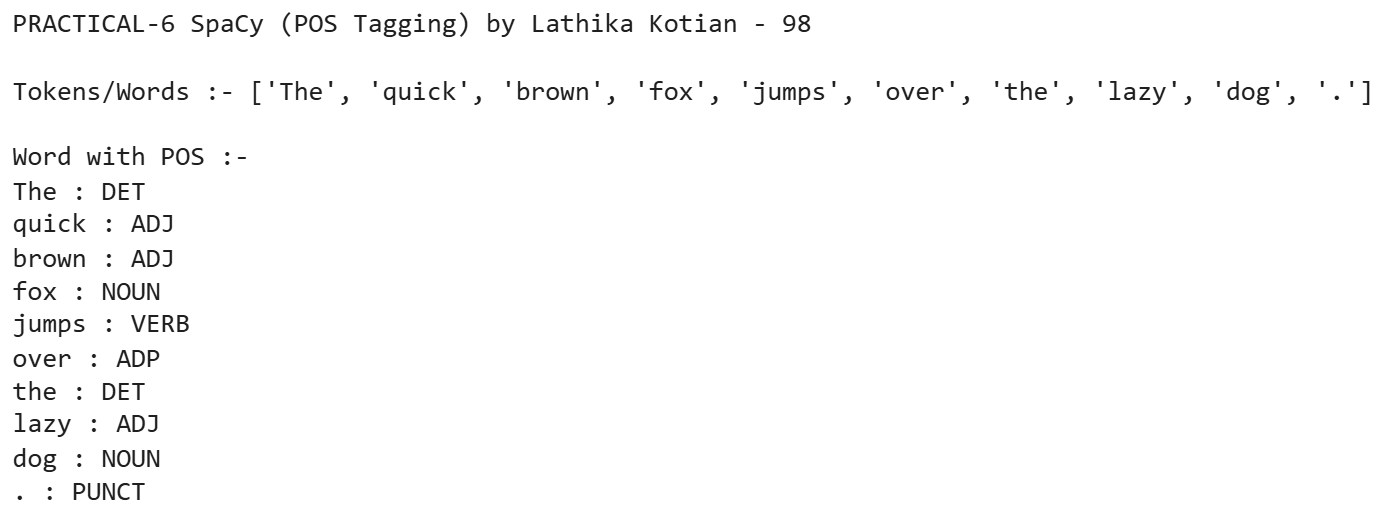
**Output:**

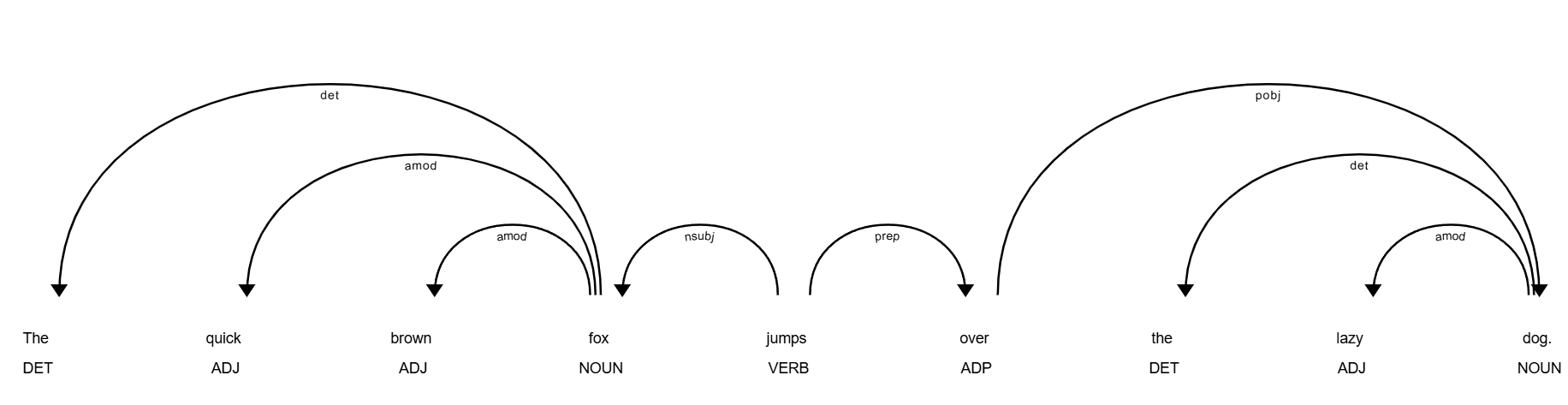
****

**Spacy Code:**

|  |
| --- |
| print("PRACTICAL-6 SpaCy (POS Tagging) by Lathika Kotian - 98\n")  import spacy  nlp = spacy.load('en\_core\_web\_sm')  text = "The quick brown fox jumps over the lazy dog."  doc = nlp(text)  print("Tokens/Words :-", [token.text for token in doc])  print("\nWord with POS :-")  for token in doc:  print(f"{token.text} : {token.pos\_}")  # Visualize the POS tagging tree (syntax tree)  from spacy import displacy  displacy.serve(doc, style='dep') |

**Output:**

****

****

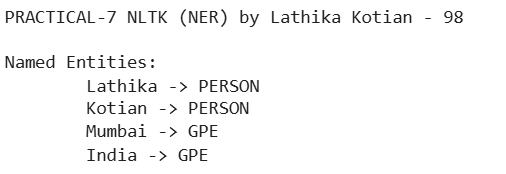
**PRACTICAL - 7**

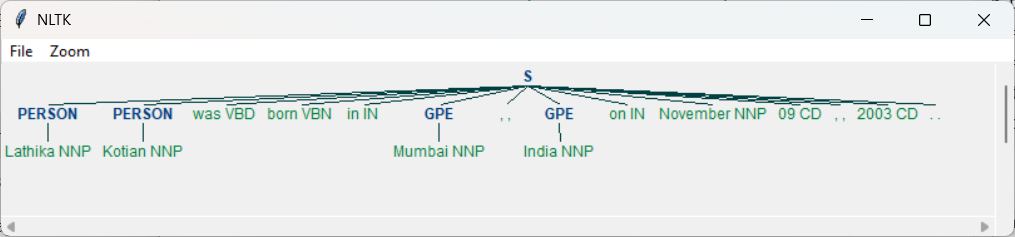
**Aim: Write a program to build a custom NER system**

**NLTK Code:**

|  |
| --- |
| #Name\_entity\_recognition  print("PRACTICAL-7 NLTK (NER) by Lathika Kotian - 98\n")  import nltk  from nltk import word\_tokenize, pos\_tag, ne\_chunk  nltk.download('punkt')  nltk.download('maxent\_ne\_chunker')  nltk.download('words')  nltk.download('averaged\_perceptron\_tagger')  text = "Lathika Kotian was born in Mumbai, India on November 09, 2003. "  # Tokenizing and tagging  tokens = word\_tokenize(text)  tagged\_tokens = pos\_tag(tokens)  named\_entities = ne\_chunk(tagged\_tokens)  #print(named\_entities)  from nltk.tree import Tree  print("Named Entities:\n")  for subtree in named\_entities:  if isinstance(subtree, Tree):  entity\_name = " ".join(token for token, pos in subtree.leaves())  entity\_type = subtree.label()  print(f"{entity\_name} -> {entity\_type}")  named\_entities.draw() |

**Output:**

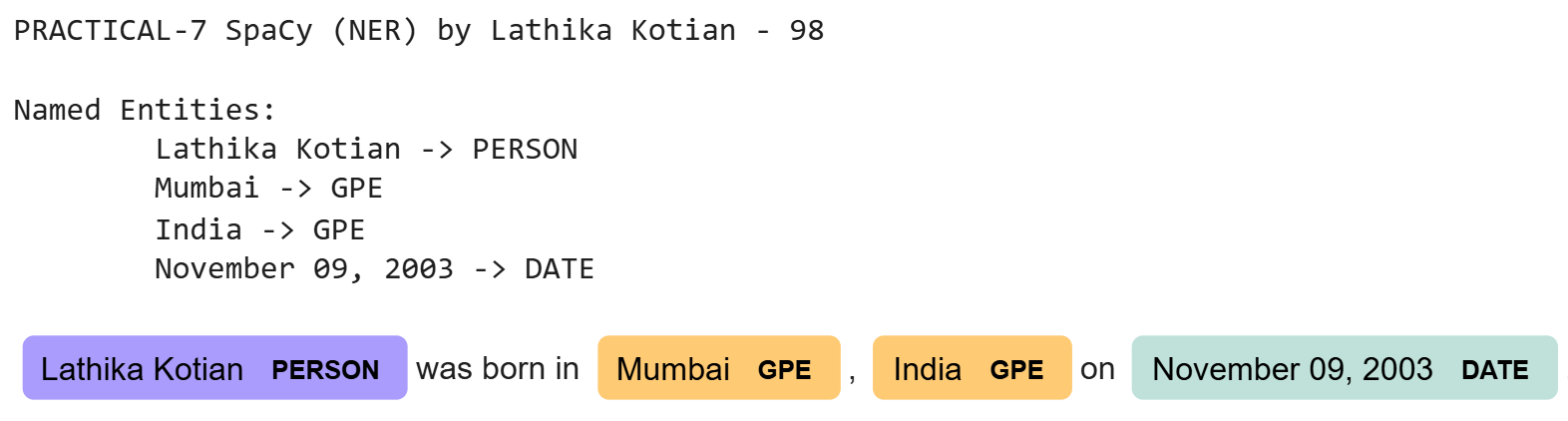
****

****

**SpaCy Code:**

|  |
| --- |
| #Name\_entity\_recognition  print("PRACTICAL-7 SpaCy (NER) by Lathika Kotian - 98\n")  import spacy  from spacy import displacy  nlp = spacy.load("en\_core\_web\_sm")  text = "Lathika Kotian was born in Mumbai, India on November 09, 2003."  doc = nlp(text)  print("Named Entities:")  for ent in doc.ents:  print(f"\t{ent.text} -> {ent.label\_}")  print()  # Visualize named entities in Colab  displacy.render(doc, style="ent", jupyter=True) |

**Output:**

****

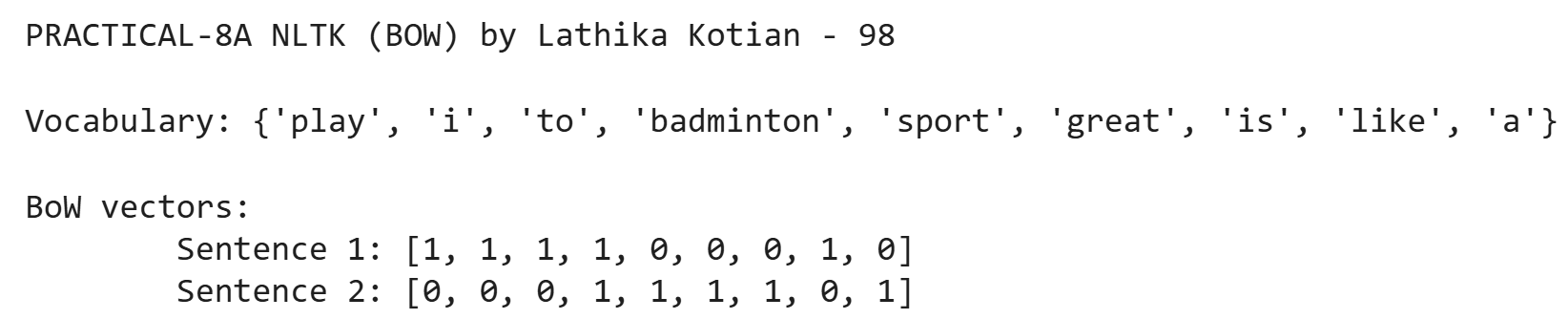
# **PRACTICAL - 8A**

**Aim: Write a program to create a bag of words(bow) text representation.**

**NLTK Code:**

|  |
| --- |
| print("PRACTICAL-8A NLTK (BOW) by Lathika Kotian - 98\n")  # Bag of words  import nltk  import numpy as np  from collections import Counter  nltk.download('punkt')  texts = [  "I like to play badminton.",  "Badminton is a great sport.",  ]  # Tokenize the texts  tokenized\_texts = [nltk.word\_tokenize(text.lower()) for text in texts]  # Create a vocabulary (set of all unique words)  vocabulary = set(word for text in tokenized\_texts for word in text)  print("Vocabulary:", vocabulary)  # Bag of Words (BoW) representation  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]  # Print BoW vectors  print("\nBoW vectors:")  for i, vector in enumerate(bow\_vectors):  print(f"Sentence {i+1}: {vector}") |

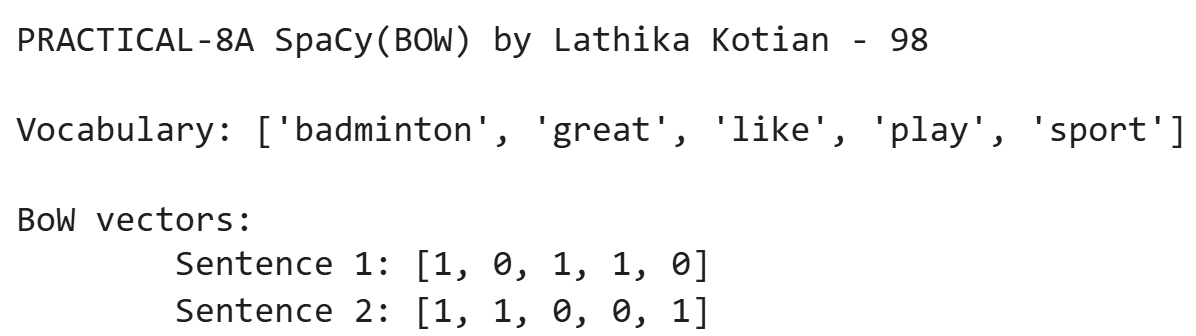
**Output:**

****

**SpaCy Code:**

|  |
| --- |
| # Bag of words  print("PRACTICAL-8A SpaCy(BOW) by Lathika Kotian - 98\n")  import spacy  import numpy as np  nlp = spacy.load("en\_core\_web\_sm")  texts = [  "I like to play badminton.",  "Badminton is a great sport.",  ]  # Process the texts using spaCy  processed\_texts = [nlp(text.lower()) for text in texts]  # Create vocabulary: unique words (without stopwords or punctuation)  vocabulary = sorted(set(token.text for doc in processed\_texts for token in doc if token.is\_alpha and not token.is\_stop))  print("Vocabulary:", vocabulary)  # Create Bag of Words (BoW) representation  def get\_bow\_representation(doc, vocabulary):  return [doc.text.count(word) for word in vocabulary]  # Create BoW vectors for each sentence  bow\_vectors = [get\_bow\_representation(doc, vocabulary) for doc in processed\_texts]  # Print BoW vectors  print("\nBoW vectors:")  for i, vector in enumerate(bow\_vectors):  print(f"Sentence {i+1}: {vector}") |

**Output:**

****

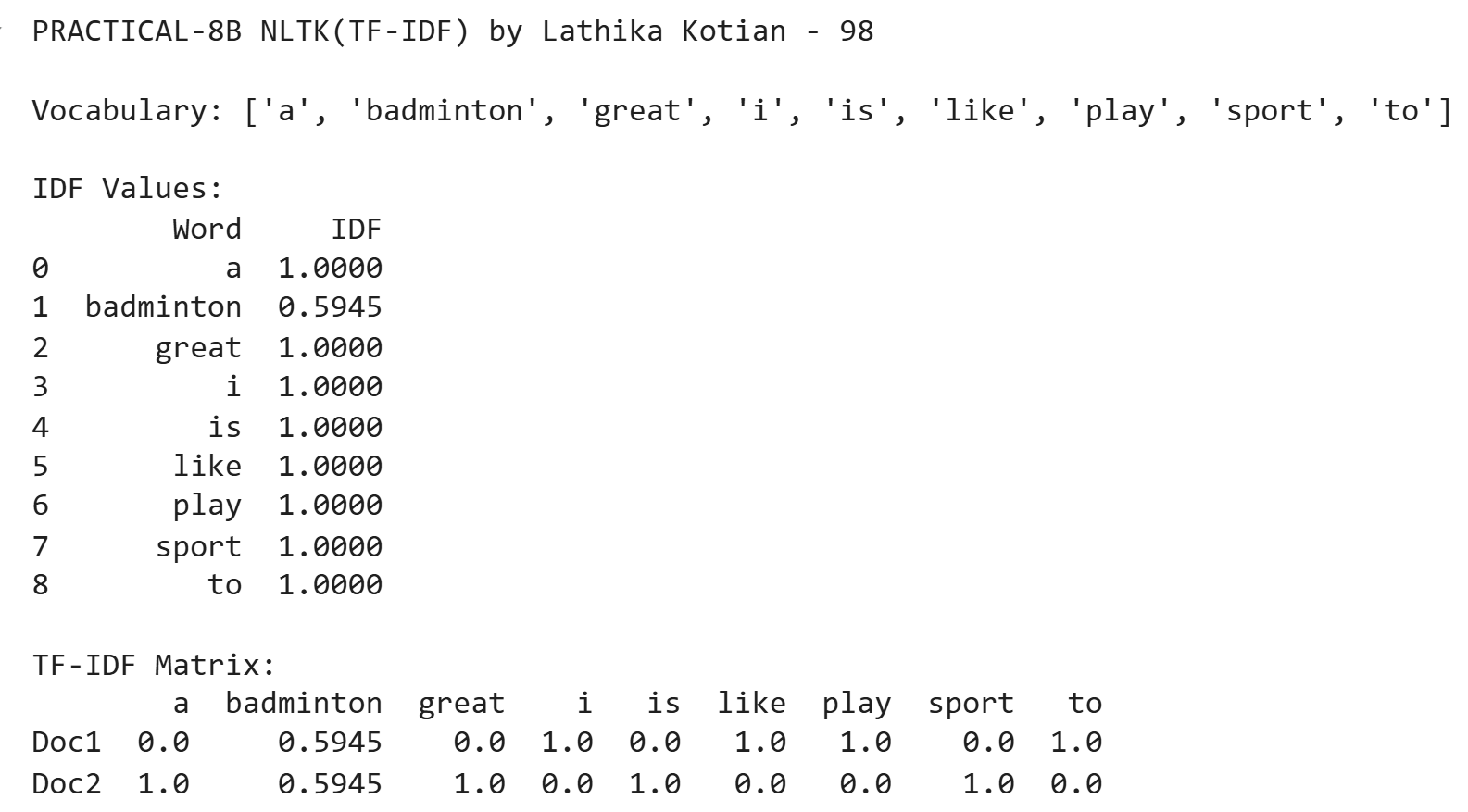
# **PRACTICAL - 8B**

**Aim: Write a program to create TF\_IDF Text Representations.**

**NLTK Code:**

|  |
| --- |
| print("PRACTICAL-8B NLTK(TF-IDF) by Lathika Kotian - 98\n")  #PROGRAM FOR TF\_IDF  import nltk  import numpy as np  import pandas as pd  from math import log  # Download tokenizer quietly  nltk.download('punkt', quiet=True)  # Input texts  texts = [  "I like to play badminton.",  "Badminton is a great sport.",  ]  # Tokenize and lowercase  tokenized\_texts = [nltk.word\_tokenize(text.lower()) for text in texts]  # Create sorted vocabulary for consistent ordering  vocabulary = sorted(set(word for text in tokenized\_texts for word in text))  print("Vocabulary:", vocabulary)  # TF function  def get\_tf(tokens, vocabulary):  return [tokens.count(word) for word in vocabulary]  # IDF function  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 # Smoothed  idf\_vector.append(idf\_value)  return idf\_vector  # TF-IDF function  def get\_tfidf(tokens, vocabulary, idf\_vector):  tf\_vector = get\_tf(tokens, vocabulary)  return [tf \* idf for tf, idf in zip(tf\_vector, idf\_vector)]  # Compute IDF  idf\_vector = get\_idf(vocabulary, tokenized\_texts)  # Show IDF nicely  idf\_df = pd.DataFrame({  'Word': vocabulary,  'IDF': np.round(idf\_vector, 4)  })  print("\nIDF Values:")  print(idf\_df)  # Compute TF-IDF for each doc  tfidf\_matrix = []  for tokens in tokenized\_texts:  tfidf = get\_tfidf(tokens, vocabulary, idf\_vector)  tfidf\_matrix.append(tfidf)  # Convert to DataFrame for display  tfidf\_df = pd.DataFrame(np.round(tfidf\_matrix, 4), columns=vocabulary, index=[f"Doc{i+1}" for i in range(len(texts))])  print("\nTF-IDF Matrix:")  print(tfidf\_df) |

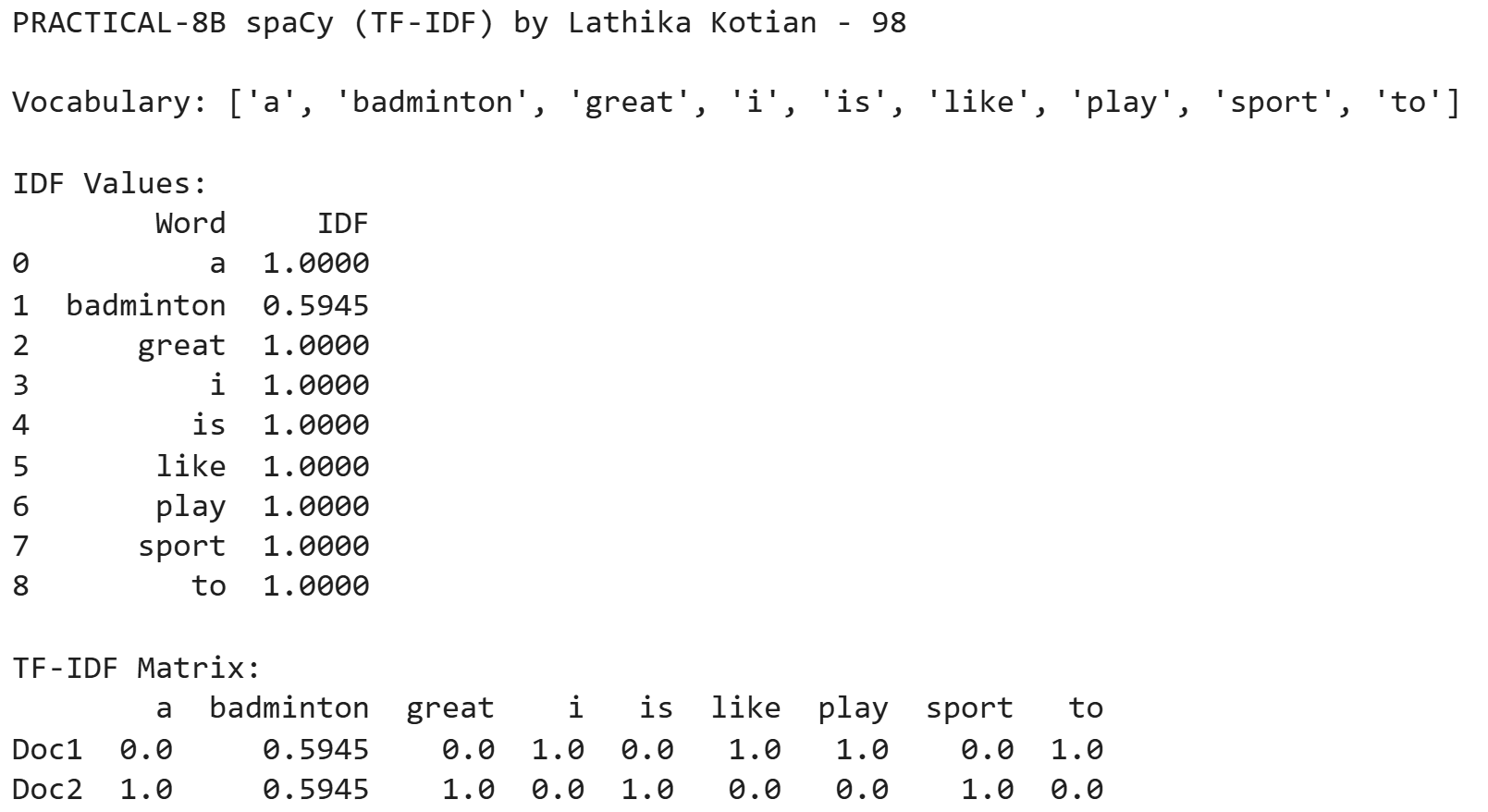
**Output:**

****

**SpaCy Code:**

|  |
| --- |
| print("PRACTICAL-8B spaCy (TF-IDF) by Lathika Kotian - 98\n")  import spacy  import numpy as np  import pandas as pd  from math import log  nlp = spacy.load("en\_core\_web\_sm")  texts = [  "I like to play badminton.",  "Badminton is a great sport.",  ]  # Tokenize and lowercase using spaCy  tokenized\_texts = [[token.text.lower() for token in nlp(text) if not token.is\_punct and not token.is\_space] for text in texts]  # Create sorted vocabulary  vocabulary = sorted(set(word for text in tokenized\_texts for word in text))  print("Vocabulary:", vocabulary)  # TF function  def get\_tf(tokens, vocabulary):  return [tokens.count(word) for word in vocabulary]  # IDF function  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 # Smoothed  idf\_vector.append(idf\_value)  return idf\_vector  # TF-IDF function  def get\_tfidf(tokens, vocabulary, idf\_vector):  tf\_vector = get\_tf(tokens, vocabulary)  return [tf \* idf for tf, idf in zip(tf\_vector, idf\_vector)]  # Compute IDF  idf\_vector = get\_idf(vocabulary, tokenized\_texts)  idf\_df = pd.DataFrame({  'Word': vocabulary,  'IDF': np.round(idf\_vector, 4)  })  print("\nIDF Values:")  print(idf\_df)  # Compute TF-IDF for each doc  tfidf\_matrix = []  for tokens in tokenized\_texts:  tfidf = get\_tfidf(tokens, vocabulary, idf\_vector)  tfidf\_matrix.append(tfidf)  # Convert to DataFrame for display  tfidf\_df = pd.DataFrame(np.round(tfidf\_matrix, 4), columns=vocabulary, index=[f"Doc{i+1}" for i in range(len(texts))])  print("\nTF-IDF Matrix:")  print(tfidf\_df) |

**Output:**

****

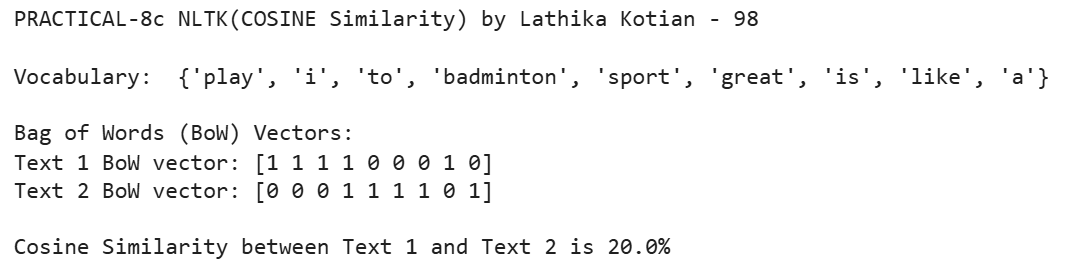
# **PRACTICAL - 8C**

**Aim: Write a program to compare two vectors of bow using cosine similarity.**

**NLTK Code:**

|  |
| --- |
| #PROGRAM TO COMPARE TWO VECTORs OF BOW USING COSINE SIMILARITY  print("PRACTICAL-8c NLTK(COSINE Similarity) by Lathika Kotian - 98\n")  import nltk  import numpy as np  from sklearn.metrics.pairwise import cosine\_similarity  nltk.download('punkt')  texts = [  "I like to play badminton",  "Badminton is a great sport"  ]  # Tokenize the texts  tokenized\_texts = [nltk.word\_tokenize(text.lower()) for text in texts]  # Create a vocabulary (set of all unique words)  vocabulary = set(word for text in tokenized\_texts for word in text)  print("Vocabulary: ",vocabulary)  # Bag of Words (BoW) representation  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]  # Print BoW vectors  print("\nBag of Words (BoW) Vectors:")  for i, bow\_vector in enumerate(bow\_vectors, start=1):  print(f"Text {i} BoW vector: {np.array(bow\_vector)}")  bow\_similarity = cosine\_similarity([bow\_vectors[0]], [bow\_vectors[1]])[0][0]  print(f"\nCosine Similarity between Text 1 and Text 2 is {bow\_similarity\*100}%") |

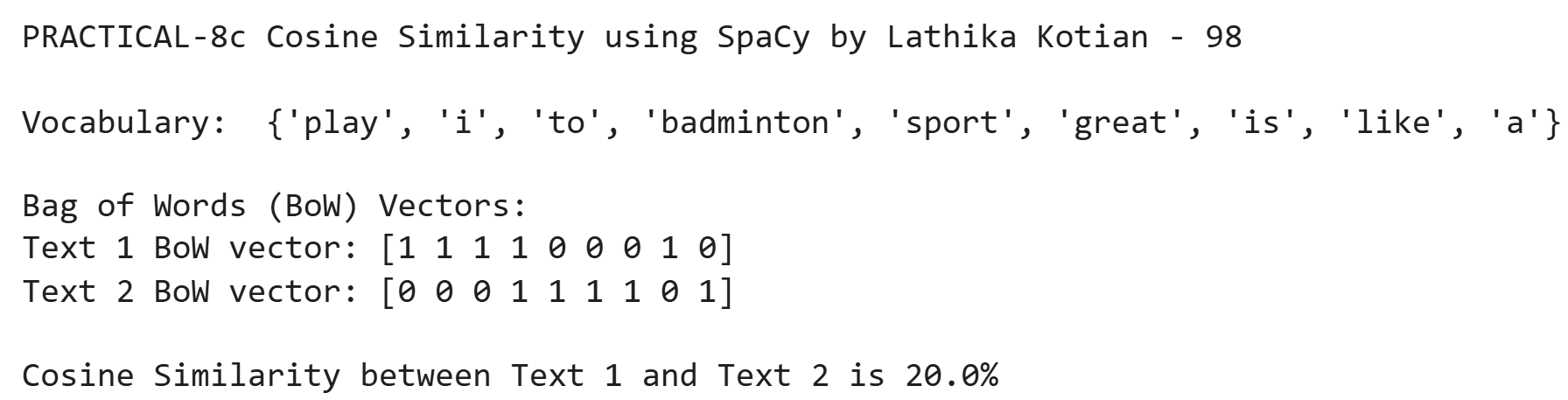
**Output:**

****

**Spacy Code:**

|  |
| --- |
| import spacy  import numpy as np  from sklearn.metrics.pairwise import cosine\_similarity  nlp = spacy.load('en\_core\_web\_sm')  print("PRACTICAL-8c Cosine Similarity using SpaCy by Lathika Kotian - 98\n")  texts = [  "I like to play badminton",  "Badminton is a great sport"  ]  # Process the texts with SpaCy  docs = [nlp(text) for text in texts]  # Create a vocabulary (set of all unique words)  vocabulary = set(token.text.lower() for doc in docs for token in doc if not token.is\_punct)  print("Vocabulary: ", vocabulary)  # Bag of Words (BoW) representation  def get\_bow\_representation(doc, vocabulary):  return [doc.text.lower().split().count(word) for word in vocabulary]  bow\_vectors = [get\_bow\_representation(doc, vocabulary) for doc in docs]  # Print BoW vectors  print("\nBag of Words (BoW) Vectors:")  for i, bow\_vector in enumerate(bow\_vectors, start=1):  print(f"Text {i} BoW vector: {np.array(bow\_vector)}")  bow\_similarity = cosine\_similarity([bow\_vectors[0]], [bow\_vectors[1]])[0][0]  print(f"\nCosine Similarity between Text 1 and Text 2 is {bow\_similarity\*100}%") |

**Output:**

****

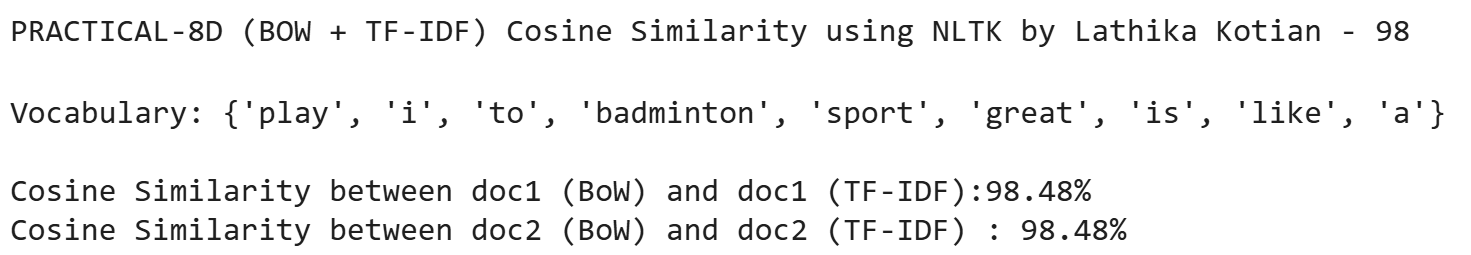
# **PRACTICAL - 8D**

**Aim: Write a program to compare a bow vector with a tf-idf vector using cosine similarity.**

**NLTK Code:**

|  |
| --- |
| print("PRACTICAL-8D (BOW + TF-IDF) Cosine Similarity using NLTK by Lathika Kotian - 98\n")  #PROGRAM TO compare a BoW vector with a TF-IDF vector using cosine similarity  import nltk  import numpy as np  from collections import Counter  from math import log  from sklearn.metrics.pairwise import cosine\_similarity    # Ensure you have the necessary NLTK resources  nltk.download('punkt')  texts = [  "I like to play badminton",  "Badminton is a great sport"  ]  # Tokenize the texts  tokenized\_texts = [nltk.word\_tokenize(text.lower()) for text in texts]  # Create a vocabulary (set of all unique words)  vocabulary = set(word for text in tokenized\_texts for word in text)  print("Vocabulary:", vocabulary)  # Bag of Words (BoW) representation  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]  # Function to compute Term Frequency (TF)  def get\_tf(tokens, vocabulary):  return [tokens.count(word) for word in vocabulary]  def get\_idf(vocabulary, docs):  num\_docs = len(docs)  idf\_vector = []  for word in vocabulary:  # Count the number of documents containing the word  num\_docs\_with\_word = sum(1 for doc in docs if word in doc)  # Calculate IDF as log(num\_docs / (1 + num\_docs\_with\_word)) to avoid division by zero  idf\_value = log(num\_docs / (1 + num\_docs\_with\_word)) + 1  idf\_vector.append(idf\_value)  return idf\_vector  # Function to compute TF-IDF  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  # Calculate IDF for the entire corpus  idf\_vector = get\_idf(vocabulary, tokenized\_texts)  # Compute TF-IDF for each document  tfidf\_vectors = [get\_tfidf(text, vocabulary, idf\_vector) for text in tokenized\_texts]  # Compute cosine similarity between BoW and TF-IDF vectors for doc1  bow\_similarity = cosine\_similarity([bow\_vectors[0]], [tfidf\_vectors[0]])[0][0]  print(f"\nCosine Similarity between doc1 (BoW) and doc1 (TF-IDF):{bow\_similarity\*100:.2f}%")  # Compute cosine similarity between BoW and TF-IDF vectors for doc2  bow\_similarity = cosine\_similarity([bow\_vectors[1]], [tfidf\_vectors[1]])[0][0]  print(f"Cosine Similarity between doc2 (BoW) and doc2 (TF-IDF) : {bow\_similarity\*100:.2f}%") |

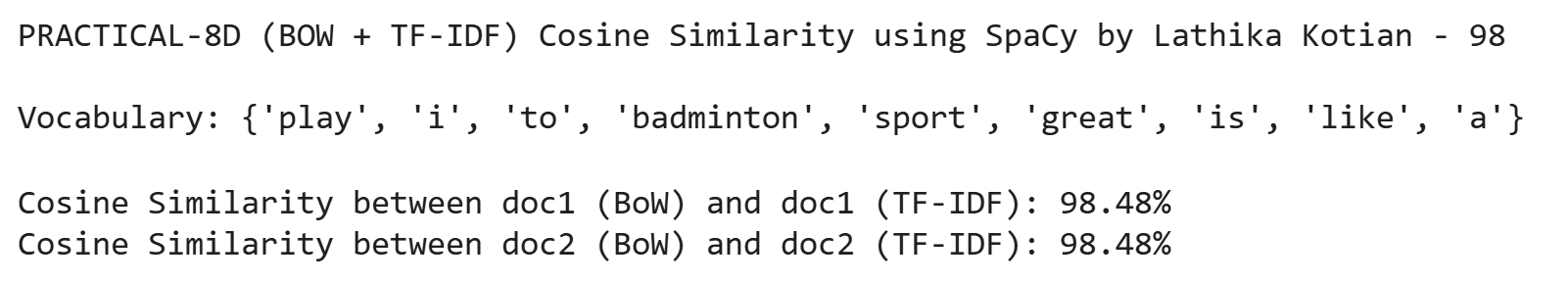
**Output:**

****

**SpaCy Code:**

|  |
| --- |
| import spacy  import numpy as np  from collections import Counter  from math import log  from sklearn.metrics.pairwise import cosine\_similarity  nlp = spacy.load('en\_core\_web\_sm')  print("PRACTICAL-8D (BOW + TF-IDF) Cosine Similarity using SpaCy by Lathika Kotian - 98\n")  texts = [  "I like to play badminton",  "Badminton is a great sport"  ]  # Process the texts with SpaCy  docs = [nlp(text) for text in texts]  # Create a vocabulary (set of all unique words)  vocabulary = set(token.text.lower() for doc in docs for token in doc if not token.is\_punct)  print("Vocabulary:", vocabulary)  # Bag of Words (BoW) representation  def get\_bow\_representation(doc, vocabulary):  return [doc.text.lower().split().count(word) for word in vocabulary]  bow\_vectors = [get\_bow\_representation(doc, vocabulary) for doc in docs]  # Function to compute Term Frequency (TF)  def get\_tf(doc, vocabulary):  return [doc.text.lower().split().count(word) for word in vocabulary]  # Function to compute Inverse Document Frequency (IDF)  def get\_idf(vocabulary, docs):  num\_docs = len(docs)  idf\_vector = []  for word in vocabulary:  # Count the number of documents containing the word  num\_docs\_with\_word = sum(1 for doc in docs if word in doc.text.lower().split())  # Calculate IDF as log(num\_docs / (1 + num\_docs\_with\_word)) to avoid division by zero  idf\_value = log(num\_docs / (1 + num\_docs\_with\_word)) + 1  idf\_vector.append(idf\_value)  return idf\_vector  # Function to compute TF-IDF  def get\_tfidf(doc, vocabulary, idf\_vector):  tf\_vector = get\_tf(doc, vocabulary)  tfidf\_vector = [tf \* idf for tf, idf in zip(tf\_vector, idf\_vector)]  return tfidf\_vector  # Calculate IDF for the entire corpus  idf\_vector = get\_idf(vocabulary, docs)  # Compute TF-IDF for each document  tfidf\_vectors = [get\_tfidf(doc, vocabulary, idf\_vector) for doc in docs]  # Compute cosine similarity between BoW and TF-IDF vectors for doc1  bow\_similarity = cosine\_similarity([bow\_vectors[0]], [tfidf\_vectors[0]])[0][0]  print(f"\nCosine Similarity between doc1 (BoW) and doc1 (TF-IDF): {bow\_similarity\*100:.2f}%")  # Compute cosine similarity between BoW and TF-IDF vectors for doc2  bow\_similarity = cosine\_similarity([bow\_vectors[1]], [tfidf\_vectors[1]])[0][0]  print(f"Cosine Similarity between doc2 (BoW) and doc2 (TF-IDF): {bow\_similarity\*100:.2f}%") |

**Output:**

****

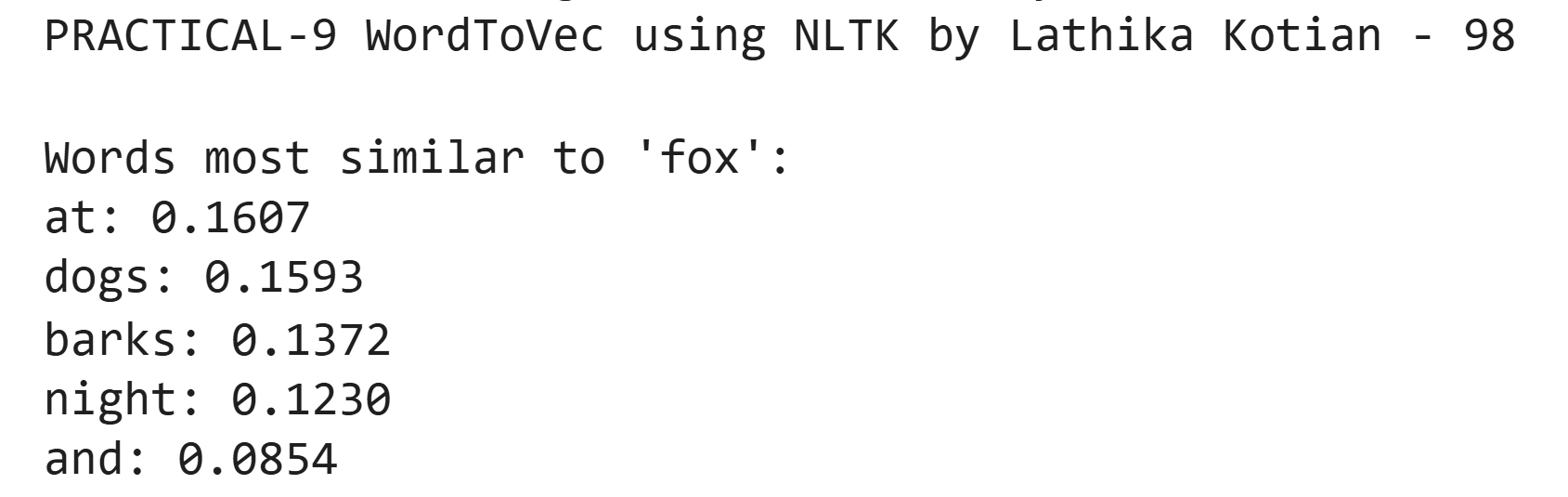
# **PRACTICAL - 9**

**Aim: Write a Program for Training and using word embedding WordToVec/GloVe**

**NLTK Code:**

|  |
| --- |
| #PROGRAM FOR TRAINING AND USING WORD EMBEDDING WORDTOVEC  !pip install gensim  from gensim.models import Word2Vec  from nltk.tokenize import word\_tokenize  import nltk  nltk.download('punkt')  print("PRACTICAL-9 WordToVec using NLTK by Lathika Kotian - 98\n")  # 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:**

****

# 

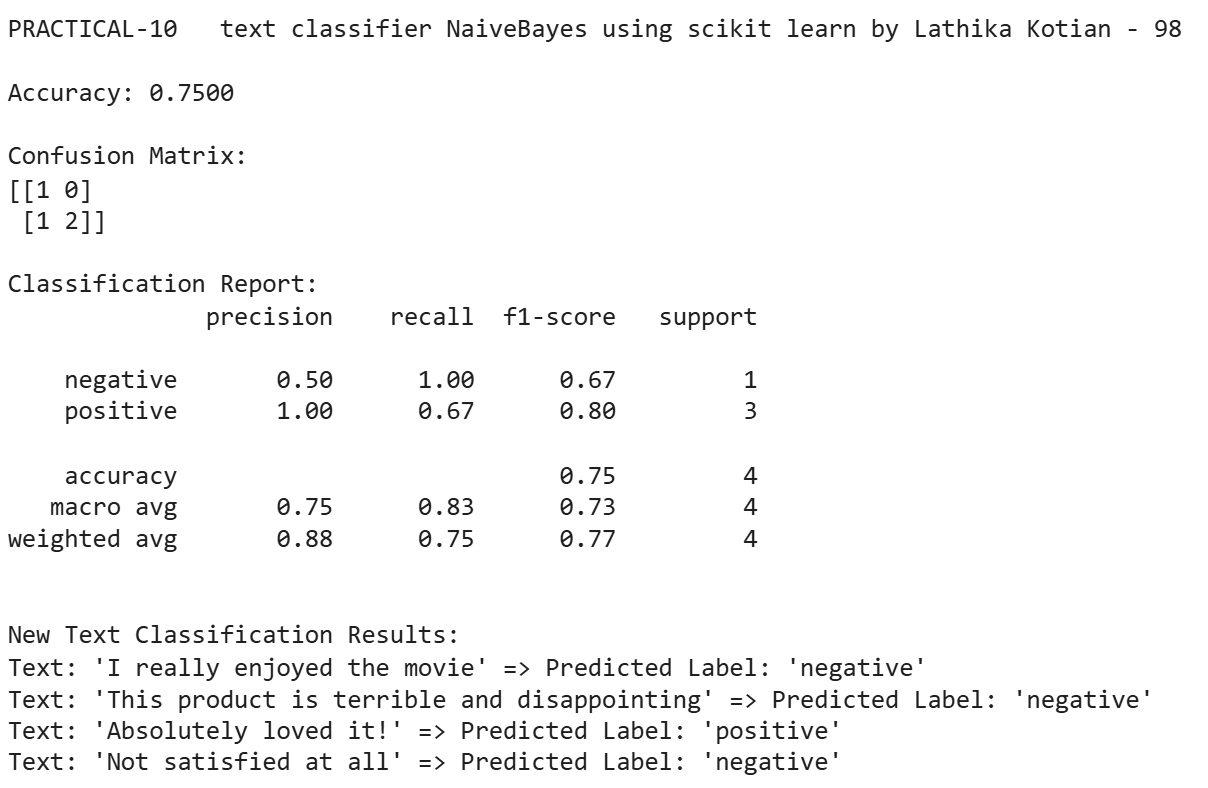
# **PRACTICAL - 10**

**Aim: Write a Program to Implement a text classifier using NaiveBayes with scikit-learn**

**Code:**

|  |
| --- |
| print("PRACTICAL-10 text classifier NaiveBayes using scikit learn by Lathika Kotian - 98\n")  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_extraction.text import CountVectorizer  from sklearn.naive\_bayes import MultinomialNB  from sklearn import metrics  texts = [  "I love programming",  "Python is great",  "I hate bugs",  "Coding is fun",  "I love solving problems",  "I hate error messages",  "Programming is awesome",  "Debugging is boring",  "This is terrible",  "Very disappointed with the experience",  "Worst product ever",  "Such a bad service",  "Absolutely amazing experience",  "Fantastic service and support",  "Really enjoyed the performance",  "Not satisfied at all after this"  ]  labels = [  'positive', 'positive', 'negative', 'positive', 'positive', 'negative', 'positive', 'negative',  'negative', 'negative', 'negative', 'negative',  'positive', 'positive', 'positive','negative'  ]  # Vectorize the text using CountVectorizer  vectorizer = CountVectorizer()  X = vectorizer.fit\_transform(texts)  y = np.array(labels)  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  #Train the Naive Bayes classifier  classifier = MultinomialNB()  classifier.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = classifier.predict(X\_test)  # Evaluate the model  print(f"Accuracy: {metrics.accuracy\_score(y\_test, y\_pred):.4f}")  print("\nConfusion Matrix:")  print(metrics.confusion\_matrix(y\_test, y\_pred))  print("\nClassification Report:")  print(metrics.classification\_report(y\_test, y\_pred))  # Classify new text inputs  new\_texts = [  "I really enjoyed the movie",  "This product is terrible and disappointing",  "Absolutely loved it!",  "Not satisfied at all"  ]  new\_texts\_vectorized = vectorizer.transform(new\_texts)  new\_predictions = classifier.predict(new\_texts\_vectorized)  print("\nNew Text Classification Results:")  for text, label in zip(new\_texts, new\_predictions):  print(f"Text: '{text}' => Predicted Label: '{label}'") |

**Output:**

****

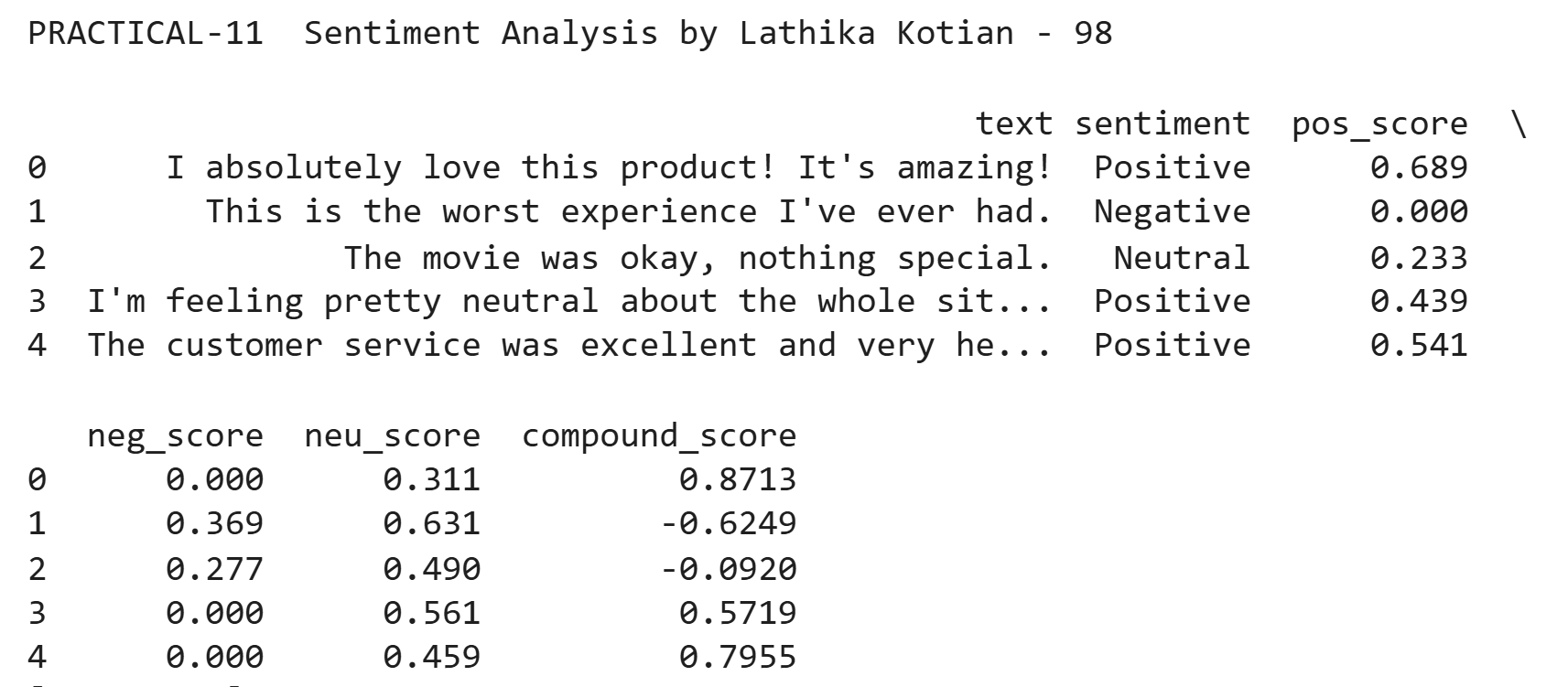
# **PRACTICAL - 11**

**Aim: Write a program to build a Sentiment analysis system**

**Code:**

|  |
| --- |
| print("PRACTICAL-11 Sentiment Analysis by Lathika Kotian - 98\n")  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 = [  "I absolutely love this product! It's amazing!",  "This is the worst experience I've ever had.",  "The movie was okay, nothing special.",  "I'm feeling pretty neutral about the whole situation.",  "The customer service was excellent and very helpful!"  ]  results\_df = analyze\_sentiments(texts)  print(results\_df) |

**Output:**

****

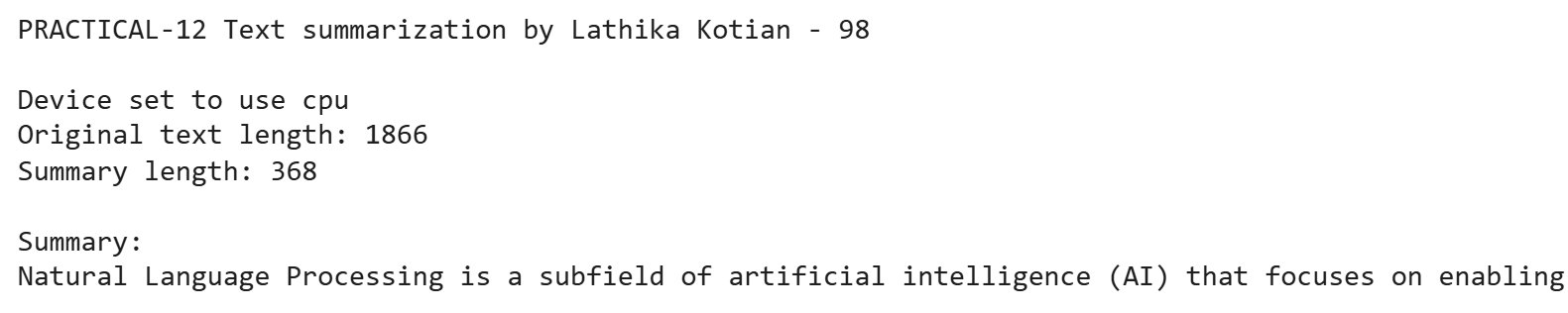
# **PRACTICAL - 12**

**Aim: Write a Program to create a text summarization tool**

**Code:**

|  |
| --- |
| # PRACTICAL-1: Text summarization by Lathika Kotian - 98  from transformers import pipeline  print("PRACTICAL-12 Text summarization by Lathika Kotian - 98\n")  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 = """  Natural Language Processing is a subfield of artificial intelligence (AI) that focuses on enabling machines to understand, interpret, and generate human language. NLP combines computational linguistics, computer science, and machine learning to process and analyze vast amounts of natural language data. As communication through text and speech is fundamental to human interaction, NLP plays a crucial role in bridging the gap between human and machine communication. One of the fundamental tasks in NLP is tokenization, where text is broken down into smaller units such as words or sentences. This is often the first step in more complex processes like part-of-speech tagging, which assigns grammatical categories (like noun or verb) to each word. Other essential techniques include stemming and lemmatization, which reduce words to their root or dictionary form to help normalize variations. More advanced NLP tasks include named entity recognition (NER), which identifies proper nouns like names of people, organizations, and locations, and sentiment analysis, which determines the emotional tone behind a body of text. Text classification, another key task, allows for organizing documents or messages into categories such as spam vs. non-spam, or positive vs. negative reviews. With the advent of deep learning, NLP has advanced dramatically. Particularly transformer-based architectures like BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), and RoBERTa. These models are capable of understanding context, ambiguity, and even generating human-like text with impressive fluency. NLP is widely used in both consumer and enterprise applications. Popular use cases include chatbots and virtual assistants like Siri, Alexa, and Google Assistant, which rely on NLP to interpret and respond to voice commands.  """  # Run summarization  summary = summarize\_text(long\_text)  # Output summary  print("Original text length:", len(long\_text))  print("Summary length:", len(summary))  print("\nSummary:")  print(summary) |

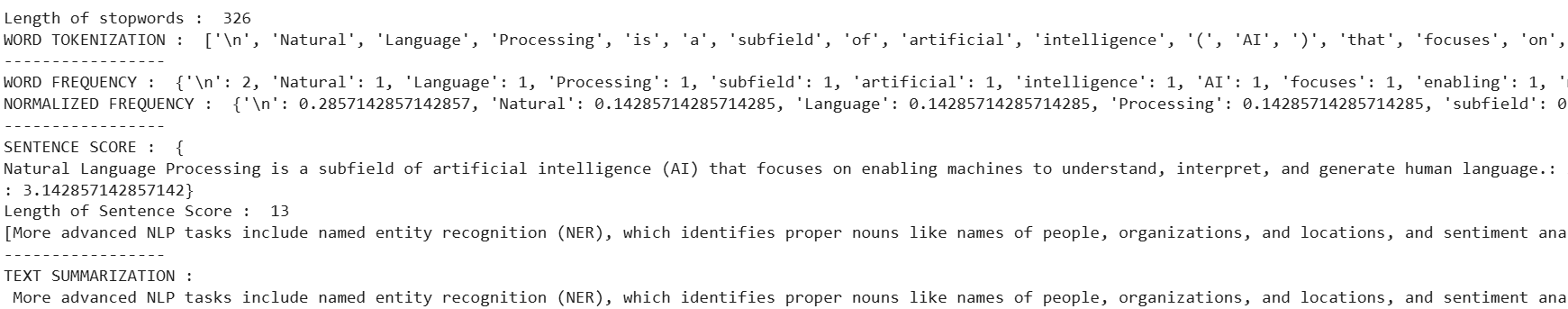
**Output:**

****

**spaCy Code:**

|  |
| --- |
| # PRACTICAL-12: Text summarization using spacy by Lathika Kotian - 98  import spacy  from spacy.lang.en.stop\_words import STOP\_WORDS  from string import punctuation  from heapq import nlargest  text = """  Natural Language Processing is a subfield of artificial intelligence (AI) that focuses on enabling machines to understand, interpret, and generate human language. NLP combines computational linguistics, computer science, and machine learning to process and analyze vast amounts of natural language data. As communication through text and speech is fundamental to human interaction, NLP plays a crucial role in bridging the gap between human and machine communication. One of the fundamental tasks in NLP is tokenization, where text is broken down into smaller units such as words or sentences. This is often the first step in more complex processes like part-of-speech tagging, which assigns grammatical categories (like noun or verb) to each word.  """  stopwords = list(STOP\_WORDS)  # length of stopwords in spacy  print("Length of stopwords : ",len(stopwords))  nlp = spacy.load('en\_core\_web\_sm')  doc = nlp(text)  # word tokenization  tokens = [token.text for token in doc]  print("WORD TOKENIZATION : ",tokens)  print("-----------------")  # word frequency  word\_frequency = {}  for word in doc:      if word.text.lower() not in stopwords:          if word.text.lower() not in punctuation:              if word.text not in word\_frequency.keys():                  word\_frequency[word.text] = 1              else:                  word\_frequency[word.text] += 1  print("WORD FREQUENCY : ",word\_frequency)  # normalized frequency  max\_frequency = max(word\_frequency.values())  for word in word\_frequency.keys():      word\_frequency[word] = word\_frequency[word]/max\_frequency  print("NORMALIZED FREQUENCY : ",word\_frequency)  print("-----------------")  # sentence tokens  sentence\_tokens = [sent for sent in doc.sents]  len(sentence\_tokens)  sentence\_scores = {}  for sent in sentence\_tokens:      for word in sent:          if word.text.lower() in word\_frequency.keys():              if sent not in sentence\_scores.keys():                  sentence\_scores[sent] = word\_frequency[word.text.lower()]              else:                  sentence\_scores[sent] +=word\_frequency[word.text.lower()]  print("SENTENCE SCORE : ",sentence\_scores)  print("Length of Sentence Score : ",len(sentence\_tokens))  select\_length = int(len(sentence\_tokens)\*0.3)  select\_length  summary = nlargest(select\_length,sentence\_scores,key=sentence\_scores.get)  print(summary)  print("-----------------")  final\_summary = [word.text for word in summary]  final\_summary = ''.join(final\_summary)  print("TEXT SUMMARIZATION : \n",final\_summary) |

**Output:**

****