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		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	

NLP PRACTICAL JOURNAL

PRACTICAL NO. 1

To implement Tokenization of text.

Word and sentence tokenization using nltk

CODE:

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

nltk.download("punkt")
nltk.download("punkt_tab")

text="Good morning. welcome to nlp practicals "

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

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

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...
Sentence Tokenization: ['Good morning.', 'welcome to nlp practicals']
Word Tokenization: ['Good', 'morning', '.', 'welcome', 'to', 'nlp', 'practicals']
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

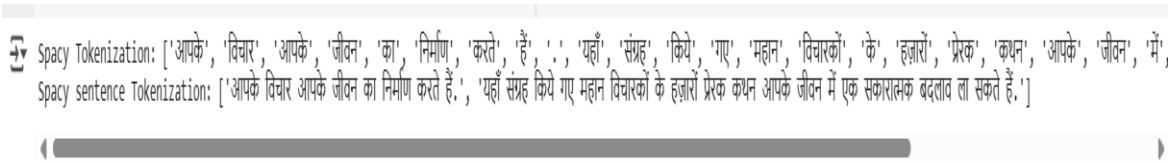
Word and sentence tokenization using Spacy

CODE:

```
#using spacy
```

```
import spacy

nlp=spacy.load("en_core_web_sm")

text="आपके विचार आपके जीवन का निर्माण करते हैं. यहाँ संग्रह किये गए महान विचारकों के हज़ारों प्रेरक कथन आपके जीवन में एक सकारात्मक बदलाव ला सकते हैं."


doc=nlp(text)

tokens=[token.text for token in doc]

print("Spacy Tokenization:",tokens) #word token


tokens=[sent.text for sent in doc.sents]

print("Spacy sentence Tokenization:",tokens) #sentence token
```

OUTPUT:



Word and sentence tokenization using regular expression

CODE:

```
#using regular expression

import re

text="Hello. How are you?"

tokens=re.findall(r'\w+',text)

print("Regular Expression Tokenization:",tokens)


sentences=re.split(r"(?<=[.!?])\s+",text)

print("regex sentence tokenization:",sentences)
```

OUTPUT:

```
Regular Expression Tokenization: ['Hello', 'How', 'are', 'you']
regex sentence tokenization: ['Hello.', 'How are you?']
```

PRACTICAL NO. 2

To implement stop word removal.

CODE:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

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

text="This is a simple example of stopword removal in nlp."

words=word_tokenize(text)
stop_words=set(stopwords.words('english'))
filtered_word=[word for word in words if word.lower() not in stop_words]

print("Original Text:",words)
print("Filtered Text:",' ',filtered_word)
```

OUTPUT:

```
Original Text: ['This', 'is', 'a', 'simple', 'example', 'of', 'stopword', 'removal', 'in', 'nlp', '.']
Filtered Text:   ['simple', 'example', 'stopword', 'removal', 'nlp', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[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...
```

PRACTICAL NO. 3

To implement stemming of Text.

Porter and Lancaster stemming

CODE:

```
#porter and lancaster stemmer
import nltk
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
from nltk.tokenize import word_tokenize

nltk.download('punkt_tab')

text="Cats are very friendly and loving animals. Dogs are very loyal to humans."

words=word_tokenize(text)

porter=PorterStemmer()
lancaster=LancasterStemmer()

porter_stemmed=[porter.stem(word) for word in words]
lancaster_stemmed=[lancaster.stem(word) for word in words]
print("Original words:",words)
print("Porter Stemmed:",porter_stemmed)
print("Lancaster Stemmed:",lancaster_stemmed)
```

OUTPUT:

```
Original words: ['Cats', 'are', 'very', 'friendly', 'and', 'loving', 'animals', '.', 'Dogs', 'are', 'very', 'loyal', 'to', 'humans', '.']
Porter Stemmed: ['cat', 'are', 'veri', 'friendli', 'and', 'love', 'anim', '.', 'dog', 'are', 'veri', 'loyal', 'to', 'human', '.']
Lancaster Stemmed: ['cat', 'ar', 'very', 'friend', 'and', 'lov', 'anim', '.', 'dog', 'ar', 'very', 'loy', 'to', 'hum', '.']
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Snowball stemming

CODE:

```
#Snowball stemmer
import nltk
from nltk.stem import SnowballStemmer
from nltk.tokenize import word_tokenize
nltk.download('punkt_tab')

text="Cats are very friendly and loving animals. Dogs are very loyal to humans."
words=word_tokenize(text)
snowball=SnowballStemmer("english")

snowball_stemmed=[snowball.stem(word) for word in words]

print("Original words:",words)
print("Snowball Stemmed:",snowball_stemmed)
```

OUTPUT:

```
Original words: ['Cats', 'are', 'very', 'friendly', 'and', 'loving', 'animals', '.', 'Dogs', 'are', 'very', 'loyal', 'to', 'humans', '.']
Snowball Stemmed: ['cat', 'are', 'veri', 'friend', 'and', 'love', 'anim', '.', 'dog', 'are', 'veri', 'loyal', 'to', 'human', '.']
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

PRACTICAL NO. 4

To implement Lemmatization of text.

CODE:

```
import nltk

from nltk.corpus import wordnet

from nltk import pos_tag

from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

nltk.download('averaged_perceptron_tagger')

nltk.download('averaged_perceptron_tagger_eng')

nltk.download('punkt_tab')

lemmatizer = WordNetLemmatizer()

def get_wordnet_pos(word):

  """Map NLTK POS tags to WordNet POS tags."""

  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)

sentence = "The striped bats are hanging on their feet for best"

words = word_tokenize(sentence)

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

print("Lemmatized words with POS:", 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!
```

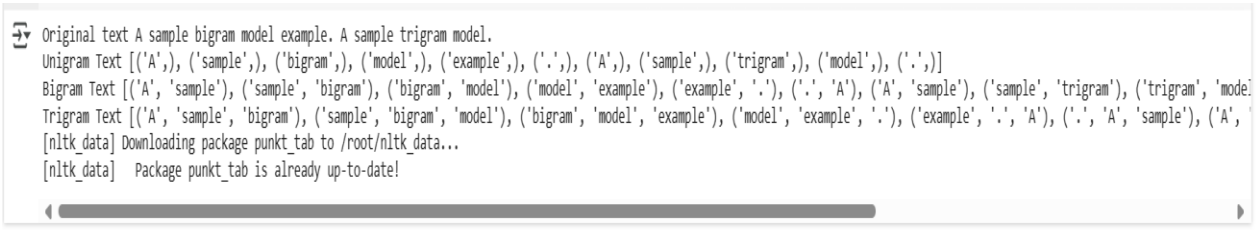
PRACTICAL NO. 5

To implement N-Gram model:

CODE:

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

text="A sample bigram model example. A sample trigram model."

tokens=word_tokenize(text)
unigrams=list(ngrams(tokens,1))
tokens=word_tokenize(text)
bigrams=list(ngrams(tokens,2))
tokens=word_tokenize(text)
trigrams=list(ngrams(tokens,3))
print("Original text",text)
print("Unigram Text",unigrams)
print("Bigram Text",bigrams)
print("Trigram Text",trigrams)
```

OUTPUT:



PRACTICAL NO. 6

To implement POS Tagging.

CODE:

```
import spacy
nlp=spacy.load("en_core_web_sm")
def pos_tagging_spacy(text):
  doc=nlp(text)
  return[(token.text,token.pos_)for token in doc]
text="The quick brown fox jumps over the 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'), ('brown', 'ADJ'), ('fox', 'NOUN'), ('jumps', 'VERB'), ('over', 'ADP'), ('the', 'DET'), ('lazy', 'ADJ'), ('dog', 'NOUN')]
```

PRACTICAL NO. 7

Building a custom NER System.

CODE:

```
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"),(19,28,"ORG")]}),
   ("Elon Musk founded Tesla.",{"entities":[(0,10,"PERSON"),(19,24,"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)
for text,annotations in TRAIN_DATA:
  doc=nlp.make_doc(text)
  tags=spacy.training.offsets_to_biluo_tags(doc,annotations.get("entities"))
  print(f"Text: {text}")
  print(f"Tags: {tags}")
```

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

```
nlp.to_disk("Custom_ner_model")
print("Training completed and model saved")
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_)
```

```
Steve Jobs PERSON
Apple ORG
```

```
[ ]  Start coding or generate with AI
```

PRACTICAL NO. 8

Creating and Comparing different Text representations.

BoW Representation

CODE:

```
#BoW
import nltk
import numpy as np
from collections import Counter
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)
vocabulary_size=len(vocabulary)
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]
print("BoW vectors:")
print(np.array(bow_vectors))
```

OUTPUT:

```
{'sat', 'dog', 'mat', 'on', 'cat', 'the', 'log'}
BoW vectors:
[[1 0 1 1 1 2 0]
 [1 1 0 1 0 2 1]]
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

TF-IDF Representation

CODE:

```
import nltk
import numpy as np
from collections import Counter
from math import log
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_tf(tokens,vocabulary):
  tf_vector=[tokens.count(word)for word in vocabulary]
  print("\n TF 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)


tfidf_vectors=[get_tfidf(text,vocabulary,idf_vector) for text in tokenized_texts]
print("\n TF-IDF vectors:")
print(np.array(tfidf_vectors))
```

OUTPUT:

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

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

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

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

Using Cosine Similarity:

```
import nltk
import numpy as np
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]
print("bow_vectors: ")
print(np.array(bow_vectors))
bow_similarity=cosine_similarity([bow_vectors[0]],
[bow_vectors[1]])[0][0]
print("bow_similarity: ")
print(bow_similarity)
```

OUTPUT:

```
{'sat', 'is', 'table', 'the', '.', 'cat', 'on', 'mat'}
bow_vectors:
[[1 0 0 2 1 1 1 1]
 [0 1 1 2 1 0 1 1]]
bow_similarity:
0.7777777777777779
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
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:

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

 IDF vector
[0.3068528194400547, 0.3068528194400547, 0.3068528194400547, 0.5945348918918356, 0.5945348918918356, 0.3068528194400547, 0.5945348918918356, 0.5945348918918356]
Cosine similarity between doc1(Bow) and doc2(TF-IDF):
0.8501578731713779
Cosine similarity between doc1(Bow) and doc2(TF-IDF):
0.8501578731713779
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

PRACTICAL NO. 9

Training and using word embeddings:

CODE:

```
!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 NO. 10

Implementing a Text classifier.

CODE:

```
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]


X=[
    "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."
]
y=["positive","negative","positive","positive","negative","positive","negative"]
new_text="I love this food."
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
    positive       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Prediction for'I love this food.':positive
```

PRACTICAL NO. 11

Building a sentiment analysis system.

CODE:

```
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 NO. 12

Creating a text summarization tool.

CODE:

```
!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.