Practical No.1

Aim: Write a program to implement sentence segmentation and word tokenization.

1. Tokenization using Python's split() function

a. Program for Word Tokenization:

CODE:

```
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
# Splits at space
a=text.split()
print(a)
```

OUTPUT:

```
['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', '
```

b. Program for Sentence Tokenization:

CODE:

```
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
text.split('.')
```

OUTPUT:

```
['Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring\ncivilization and a multi-planet species by building a
self-sustaining city on Mars',
 ' In 2008,\nSpaceX's Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the\nEarth',
 '']
```

2. Tokenization using Regular Expressions (RegEx)

a. Program for Word Tokenization:

CODE:

```
import re
text ="""Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
tokens = re.findall("[\w']+",text)
print(tokens)
```

OUTPUT:

```
['Founded', 'in', '2002', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization',
```

b. Program for Sentence Tokenization:

CODE:

```
import re
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on, Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
sentences = re.compile('[.!?]').split(text)
sentences
```

OUTPUT:

```
['Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring\ncivilization and a multi-planet species by building a self-sustaining city on, Mars',
 ' In 2008,\nSpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the\nEarth',
 '']
```

3. Tokenization using NLTK

a. Program for Word Tokenization:

CODE:

```
import nltk
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
a = word_tokenize(text)
print(a)
```

OUTPUT:

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
['Founded', 'in', '2002', ',', 'SpaceX', '’', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civ
```

b. Program for Sentence Tokenization:

CODE:

```
from nltk.tokenize import sent_tokenize
text ="""Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
sent_tokenize(text)
```

OUTPUT:

```
['Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring\ncivilization and a multi-planet species by building a
self-sustaining city on Mars.',
 'In 2008,\nSpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the\nEarth.']
```

4. Tokenization using the spaCy library

a. Program for Word Tokenization:

CODE:

```
# Word Tokenization
from spacy.lang.en import English
# Load English tokenizer, tagger, parser, NER and word vectors
nlp = English()
text = """Founded in 2002, U.S.A. SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""

# &amp;quot;nlp&amp;quot; Object is used to create documents with linguistic annotations.
my_doc = nlp(text)
# Create list of word tokens
token_list = []
for token in my_doc:
  token_list.append(token.text)
token_list
```

OUTPUT:

```
['Founded',
 'in',
 '2002',
 ',',
 'U.S.A.',
 'SpaceX',
 '’s',
 'mission',
 'is',
 'to',
 'enable',
 'humans',
 'to',
 'become',
 'a',
 '\n',
 'spacefaring',
 'civilization',
 'and',
 'a',
 'multi',
 '-',
 'planet',
 'species',
 'by',
 'building',
 'a',
 'self',
 '-',
 'sustaining',
 'city',
 'on',
 'Mars',
 '.',
 '\n',
 'In',
```

<u>b. Program for Sentence Tokenization:</u>

CODE:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("""Founded in 2002, SpaceX’s mission is to enable humans to become a
spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars.

In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to
orbit the Earth.""")
for sent in doc.sents:
  print(sent.text)
```

OUTPUT:

```
Founded in 2002, SpaceX’s mission is to enable humans to become a
spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars.

In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to
orbit the Earth.
```

5. Tokenization using Gensim

a. Program for Word Tokenization:

CODE:

```
from gensim.utils import tokenize
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
list(tokenize(text))
```

OUTPUT:

```
['Founded',
 'in',
 'SpaceX',
 's',
 'mission',
 'is',
 'to',
 'enable',
 'humans',
 'to',
 'become',
 'a',
 'spacefaring',
 'civilization',
 'and',
 'a',
 'multi',
 'planet',
 'species',
 'by',
 'building',
 'a',
 'self',
 'sustaining',
 'city',
```

b. Program for Sentence Tokenization:

CODE:

OUTPUT:

Practical No.2

Aim: Write a program to generate unigram, bigram, and trigram models from given text.

CODE:

```
from collections import Counter, defaultdict
def build_ngram_models(text):
    # Preprocess: lowercase and split into words
    words = text.lower().split()
    # Initialize models
    unigram = Counter(words)
    bigram = defaultdict(Counter)
    trigram = defaultdict(Counter)
    # Build bigram model
    for i in range(len(words) - 1):
        current_word = words[i]
        next_word = words[i + 1]
        bigram[current_word][next_word] += 1
    # Build trigram model
    for i in range(len(words) - 2):
        context = (words[i], words[i + 1])
        next_word = words[i + 2]
        trigram[context][next_word] += 1
    # Calculate probabilities
    def get_probabilities(model, is_unigram=False):
        if is_unigram:
            total = sum(model.values())
            return {word: count/total for word, count in model.items()}
        else:
            result = {}
            for context, next_words in model.items():
                total = sum(next_words.values())
                result[context] = {word: count/total for word, count in next_words.items()}
            return result
```

```
    # Package results
    return {
        'unigram': {
            'counts': unigram,
            'probabilities': get_probabilities(unigram, is_unigram=True)
        },
        'bigram': {
            'counts': dict(bigram),
            'probabilities': get_probabilities(bigram)
        },
        'trigram': {
            'counts': dict(trigram),
            'probabilities': get_probabilities(trigram)
        }
    }
# Example usage
if __name__ == "__main__":
    sample_text = "the quick brown fox jumps over the lazy dog the fox was quick"
    models = build_ngram_models(sample_text)

    # Display top unigrams
    print("Top unigrams:")
    for word, count in models['unigram']['counts'].most_common(3):
        prob = models['unigram']['probabilities'][word]
        print(f"'{word}': count={count}, probability={prob:.3f}")
    # Display example bigrams
    print("\nExample bigrams:")
    for context in ['the', 'fox']:
        if context in models['bigram']['counts']:
            print(f"After '{context}':")
            for next_word, count in models['bigram']['counts'][context].most_common(2):
                prob = models['bigram']['probabilities'][context][next_word]
                print(f" '{next_word}': count={count}, probability={prob:.3f}")
```

```
# Display example trigrams
print("\nExample trigrams:")
sample_context = ('the', 'fox')
if sample_context in models['trigram']['counts']:
    print(f"After '{sample_context[0]} {sample_context[1]}':")
    for next_word, count in models['trigram']['counts'][sample_context].most_common(2):
        prob = models['trigram']['probabilities'][sample_context][next_word]
        print(f" '{next_word}': count={count}, probability={prob:.3f}")
```

OUTPUT:

```
Top unigrams:
'the': count=3, probability=0.231
'quick': count=2, probability=0.154
'fox': count=2, probability=0.154

Example bigrams:
After 'the':
 'quick': count=1, probability=0.333
 'lazy': count=1, probability=0.333
After 'fox':
 'jumps': count=1, probability=0.500
 'was': count=1, probability=0.500

Example trigrams:
After 'the fox':
 'was': count=1, probability=1.000
```

Practical No.3

Aim: Write a program to Implement stemming and lemmatization algorithm.

1. PorterStemmer

CODE:

```
import nltk
from nltk.stem.porter import PorterStemmer
nltk.download('punkt_tab')
porter_stemmer = PorterStemmer()
text = "Pythoners are vey intelligent and work very pythonly"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
  print("Stemming for {} is- {}".format(w,porter_stemmer.stem(w)))
```

OUTPUT:

```
Stemming for Pythoners is- python
Stemming for are is- are
Stemming for vey is- vey
Stemming for intelligent is- intellig
Stemming for and is- and
Stemming for work is- work
Stemming for very is- veri
Stemming for pythonly is- pythonli
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

2. LancasterStemmer

CODE:

```
import nltk
from nltk.stem import LancasterStemmer
lancaster_stemmer = LancasterStemmer()
text = "studies studing cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
  print("Stemming for {} is- {}".format(w,porter_stemmer.stem(w)))
```

OUTPUT:

```
Stemming for studies is- studi
Stemming for studing is- stude
Stemming for cries is- cri
Stemming for cry is- cri
```

3. SnowballStemmer

CODE:

```
nltk.download('stopwords')
from nltk.stem import SnowballStemmer
snowball = SnowballStemmer(language='english', ignore_stopwords=True)
words = ['generous','generate','generously','generation','having','ruined','fighter']
for word in words:
  print(word,"--->",snowball.stem(word))
```

OUTPUT:

```
generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat
having ---> having
ruined ---> ruin
fighter ---> fighter
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Lemmatization

a. Lemmatization using NLTK

CODE:

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "the striped bats are hanging on their feet for best"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
  print("{0:20}{1:20}".format(w,wordnet_lemmatizer.lemmatize(w)))
```

OUTPUT:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
the                 the
striped             striped
bats                bat
are                 are
hanging             hanging
on                  on
their               their
feet                foot
for                 for
best                best
[nltk_data]   Package wordnet is already up-to-date!
```

b. Lemmatization using Spacy

CODE:

```
import spacy

nlp = spacy.load("en_core_web_sm")

doc = nlp("eating eats eat ate ability adjustable rafting meeting better")

for token in doc:

  print(token,"|",token.lemma_,"|",token.lemma)
```

OUTPUT:

```
eating | eat | 9837207709914848172
eats | eat | 9837207709914848172
eat | eat | 9837207709914848172
ate | eat | 9837207709914848172
ability | ability | 11565809527369121409
adjustable | adjustable | 60335119441506944 80
rafting | raft | 7154368781129989833
meeting | meeting | 14798207169164081740
better | well | 4525988469032889948
```

Practical No.4

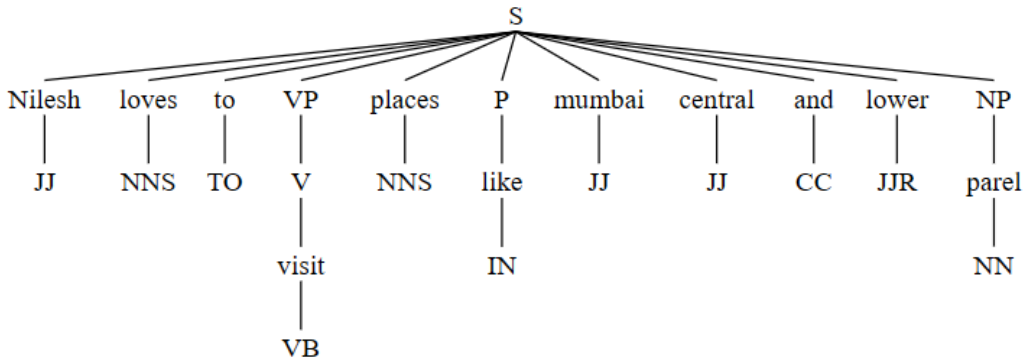
Aim: Write a program to Implement syntactic parsing of a given text.

CODE:

```
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
# Download the language specific data for the averaged_perceptron_tagger
nltk.download('averaged_perceptron_tagger_eng')
from nltk import pos_tag, word_tokenize, RegexpParser
from nltk.tree import Tree # Import Tree for visualization
import matplotlib.pyplot as plt
from IPython.display import display, Image
import svgling
sample_text = "Nilesh loves to visit places like mumbai central and lower parel "
tagged = pos_tag(word_tokenize(sample_text))
chunker = RegexpParser("""
        NP: {<DT>?<JJ>*<NN>}
        P: {<IN>}
        V: {<V.*>}
        PP: {<P><NP>} # Changed 'p' to '<P>' to reference the preposition chunk
        VP: {<V> <NP|PP>*}
        """)
output = chunker.parse(tagged)
print("After Extracting\n", output)
# Instead of draw_trees, use svgling to render as SVG:
svg = svgling.draw_tree(output)
display(svg) # Display the SVG in the notebook
```

OUTPUT:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
After Extracting
 (S
  Nilesh/JJ
  loves/NNS
  to/TO
  (VP (V visit/VB))
  places/NNS
  (P like/IN)
  mumbai/JJ
  central/JJ
  and/CC
  lower/JJR
  (NP parel/NN))
```



Practical No.5

Aim: Write a program to Implement dependency parsing of a given text.

CODE:

```
import spacy

nlp = spacy.load('en_core_web_sm')

text = "Reliance Retail acquires majority stake in designer brand Abraham & Thakore."

doc = nlp (text)

print ("{:<15} | {:<8} | {:<15} | {:<20}".format('Token','Relation','Head','Childern'))

print("-"*70)

for token in doc:

  print ("{:<15} | {:<8} | {:<15} | {:<20}".format(str(token.text),str(token.dep_),str(token.head.text),str([child for child in token.children])))
```

OUTPUT:

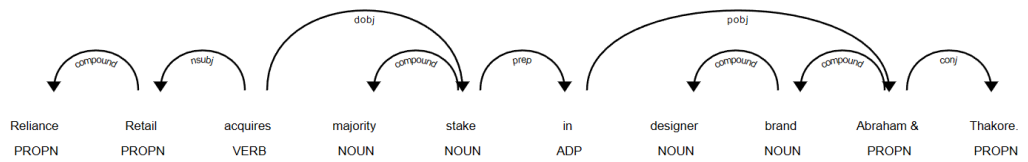
```
Token           | Relation | Head            | Childern
----------------------------------------------------------------------
Reliance        | compound | Retail          | []
Retail          | nsubj    | acquires        | [Reliance]
acquires        | ROOT     | acquires        | [Retail, stake, .]
majority        | compound | stake           | []
stake           | dobj     | acquires        | [majority, in]
in              | prep     | stake           | [Abraham]
designer        | compound | brand           | []
brand           | compound | Abraham         | [designer]
Abraham         | pobj     | in              | [brand, &, Thakore]
&               | cc       | Abraham         | []
Thakore         | conj     | Abraham         | []
.               | punct    | acquires        | []
```

CODE:

```
from spacy import displacy

displacy.render(doc, style='dep', jupyter=True, options={'distance': 120})
```

OUTPUT:



Practical No.6

Aim: Write a program to Implement Named Entity Recognition (NER).

A. Implementing NER using NLTK Library

CODE:

```
import nltk
import svgling
from IPython.display import display
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('maxent_ne_chunker_tab')
sentence = 'Peterson first suggested the name "open source" at Palo Alto, California'
words = nltk.word_tokenize(sentence)
pos_tagged = nltk.pos_tag(words)
ne_tagged = nltk.ne_chunk(pos_tagged)
print("NE tagged text:")
print(ne_tagged)
print()
print("Recognized named entities:")
for ne in ne_tagged:
  if hasattr(ne,"label"):
    print(ne.label(),ne[0:])
```

OUTPUT:

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-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 maxent_ne_chunker_tab to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker_tab is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
NE tagged text:
(S
  (PERSON Peterson/NNP)
  first/RB
  suggested/VBD
  the/DT
  name/NN
  ``/``
  open/JJ
  source/NN
  ''/''
  at/IN
  (FACILITY Palo/NNP Alto/NNP)
  ,/,
  (GPE California/NNP))
```

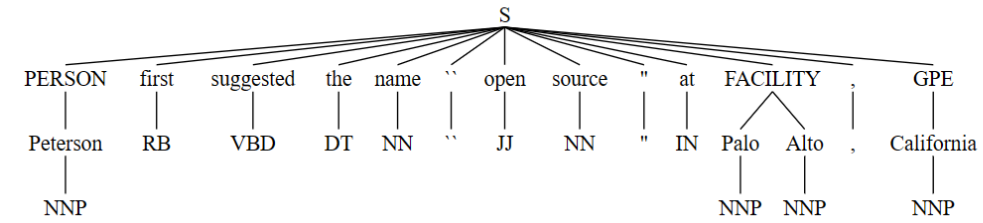
CODE:

svg=svgling.draw_tree(ne_tagged)

display(svg)

OUTPUT:

```
Recognized named entities:
PERSON [('Peterson', 'NNP')]
FACILITY [('Palo', 'NNP'), ('Alto', 'NNP')]
GPE [('California', 'NNP')]
```



B. Implementing NER using spaCy Library

CODE:

import spacy

from spacy import displacy

NER = spacy.load("en_core_web_sm")

raw_text=raw_text="The Indian Space Research Organisation or is the national space agency of India,headquartered in Bengaluru. It operates under Department of Space which is directly overseen by thePrime Minister of India while Chairman of ISRO acts as executive of DOS as well."

text1 = NER(raw_text)

for word in text1.ents:

  print(word.text,word.label_)

spacy.displacy.render(text1,style="ent",jupyter=True)

OUTPUT:

```
The Indian Space Research Organisation ORG
India GPE
Bengaluru GPE
Department of Space ORG
India GPE
ISRO ORG
DOS ORG
```

The Indian Space Research Organisation ORG or is the national space agency of India GPE ,headquartered in Bengaluru GPE . It operates under Department of Space ORG which is directly overseen by thePrime Minister of India GPE while Chairman of ISRO ORG acts as executive of DOS ORG as well.

Practical No.7

Aim: Write a program to Implement Text Summarization for the given sample text.

```
import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize,sent_tokenize
```

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

```
text = """There are many techniques available to generate extractive summarization to keep it simple,

I will be using an unsupervised learning approach to find the sentences similarity and rank them.

Summarization can be defined as a task of producing a concise and fluent summary while preserving key

information and overall meaning. One benefit of this will be, you don’t need to train and build a model

prior start using it for your project. It’s good to understand Cosine similarity to make the best use of

the code you are going to see. Cosine similarity is a measure of similarity between two non-zero vectors

of an inner product space that measures the cosine of the angle between them. Its measures cosine of the

angle between vectors. The angle will be 0 if sentences are similar. """

nltk.download('punkt_tab')

stopWords=set(stopwords.words("english"))
```

```
words=word_tokenize(text)
```

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

```
freqTable=dict()
for word in words:
  word=word.lower()
  if word in stopWords:
    continue
  if word in freqTable:
    freqTable[word]+=1
  else:
    freqTable[word]=1
sentences=sent_tokenize(text)
sentenceValue=dict()
for sentence in sentences:
  for word,freq in freqTable.items():
    if word in sentence.lower():
      if sentence in sentenceValue:
        sentenceValue[sentence]+=freq
      else:
        sentenceValue[sentence]=freq
  sumValues=0
  for sentence in sentenceValue:
    sumValues+=sentenceValue[sentence]
average=int(sumValues/len(sentenceValue))
summary=""
for sentence in sentences:
  if(sentence in sentenceValue) and (sentenceValue[sentence]>(1.2*average)):
    summary+=" "+sentence
print("Original String\n"+text)
print("\n\nSummarized text\n"+summary)
```

```
Original String
There are many techniques available to generate extractive summarization to keep it simple,
I will be using an unsupervised learning approach to find the sentences similarity and rank them.
Summarization can be defined as a task of producing a concise and fluent summary while preserving key
information and overall meaning. One benefit of this will be, you don’t need to train and build a model
prior start using it for your project. It’s good to understand Cosine similarity to make the best use of
the code you are going to see. Cosine similarity is a measure of similarity between two non-zero vectors
of an inner product space that measures the cosine of the angle between them. Its measures cosine of the
angle between vectors. The angle will be 0 if sentences are similar.


Summarized text
 There are many techniques available to generate extractive summarization to keep it simple,
I will be using an unsupervised learning approach to find the sentences similarity and rank them. Cosine similarity is a measure of similarity between two non-zero vectors
of an inner product space that measures the cosine of the angle between them.
```

len(text)

```
780
```

len(summary)

```
342
```

Practical No.8

Aim: Implementing sentiment analysis for customer feedback in financial services.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import re


nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('punkt')
```

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

```
data = {
    'feedback': [
        "The new mobile banking app is fantastic and easy to use",
        "I'm disappointed with the high fees for international transfers",
        "Customer service was helpful in resolving my credit card issue",
        "The waiting time to speak with a representative is too long",
        "I love the new cashback rewards program on my credit card",
        "The interest rates on savings accounts are extremely low",
        "The bank staff was rude and unhelpful when I visited the branch",
        "Setting up automatic payments was simple and straightforward",
        "I received conflicting information about my loan application",
        "Very satisfied with how quickly my loan was approved",
        "The website crashed when I tried to make a payment",
        "The financial advisor provided excellent investment advice",
        "Hidden fees were charged to my account without notification",
        "The credit card limit increase process was quick and easy",
        "Disappointed that the bank closed my local branch"
    ]
}
df = pd.DataFrame(data)
print("Sample financial services customer feedback:")
print(df.head())
```

```
Sample financial services customer feedback:
                                            feedback
0  The new mobile banking app is fantastic and ea...
1  I'm disappointed with the high fees for intern...
2  Customer service was helpful in resolving my c...
3  The waiting time to speak with a representativ...
4  I love the new cashback rewards program on my ...
```

```
def preprocess_text(text):
  text = text.lower()
  text = re.sub(r'[^a-zA-Z\s]', '', text)
  return text

df['processed_feedback'] = df['feedback'].apply(preprocess_text)
```

Approach 1:

```
sid = SentimentIntensityAnalyzer()
def get_sentiment_score(text):
    return sid.polarity_scores(text)
df['sentiment_scores'] = df['feedback'].apply(get_sentiment_score)
df['compound_score'] = df['sentiment_scores'].apply(lambda x: x['compound'])
df['sentiment_category'] = df['compound_score'].apply(
    lambda x: 'Positive' if x >= 0.05 else ('Negative' if x <= -0.05 else 'Neutral'))
print("\nSentiment Analysis Results:")
print(df[['feedback', 'compound_score', 'sentiment_category']].head(10))
```

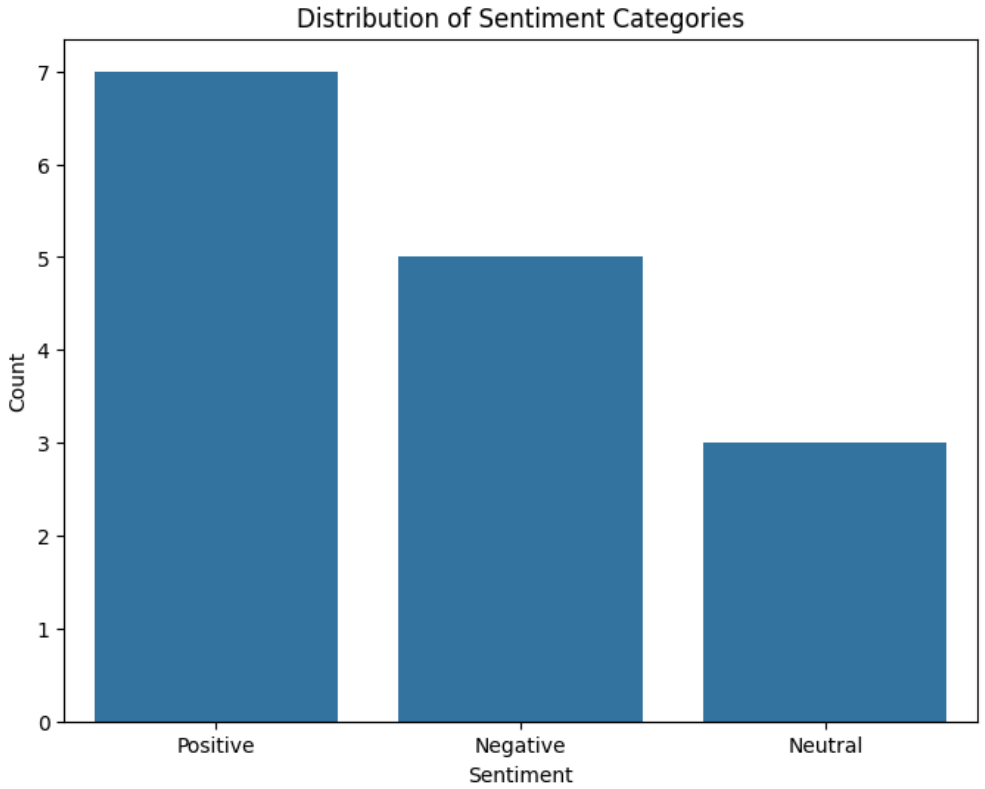
```
Sentiment Analysis Results:
                                            feedback  compound_score  \
0  The new mobile banking app is fantastic and ea...          0.7579
1  I'm disappointed with the high fees for intern...         -0.4767
2  Customer service was helpful in resolving my c...          0.7906
3  The waiting time to speak with a representativ...          0.0000
4  I love the new cashback rewards program on my ...          0.8720
5  The interest rates on savings accounts are ext...          0.1548
6  The bank staff was rude and unhelpful when I v...         -0.4588
7  Setting up automatic payments was simple and s...          0.0000
8  I received conflicting information about my lo...         -0.4019
9  Very satisfied with how quickly my loan was ap...          0.7089

  sentiment_category
0           Positive
1           Negative
2           Positive
3            Neutral
4           Positive
5           Positive
6           Negative
7            Neutral
8           Negative
9           Positive
```

```
sentiment_counts = df['sentiment_category'].value_counts()
print("\nSentiment Distribution:")
print(sentiment_counts)
```

```
Sentiment Distribution:
sentiment_category
Positive    7
Negative    5
Neutral     3
Name: count, dtype: int64
```

```
plt.figure(figsize=(8, 6))
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values)
plt.title('Distribution of Sentiment Categories')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



Approach 2:

```
df['label'] = df['sentiment_category'].map({'Positive': 1, 'Neutral': 0, 'Negative': -1})
```

```
X_train, X_test, y_train, y_test = train_test_split(
    df['processed_feedback'],
    df['label'],
    test_size=0.3,
    random_state=42
)

vectorizer = CountVectorizer(max_features=1000)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_vec, y_train)

y_pred = model.predict(X_test_vec)

print("\nMachine Learning Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Neutral', 'Positive']))
```

```
Machine Learning Model Evaluation:
Accuracy: 0.00

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00       0.0
     Neutral       0.00      0.00      0.00       0.0
    Positive       0.00      0.00      0.00       5.0

    accuracy                           0.00       5.0
   macro avg       0.00      0.00      0.00       5.0
weighted avg       0.00      0.00      0.00       5.0

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificat
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificat
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
```

```
def predict_sentiment(new_feedback):
    processed = preprocess_text(new_feedback)


def predict_sentiment(new_feedback):
    processed = preprocess_text(new_feedback)


    vader_score = sid.polarity_scores(new_feedback)['compound']
    vader_sentiment = 'Positive' if vader_score >= 0.05 else ('Negative' if vader_score <=
-0.05 else 'Neutral')
    vec_feedback = vectorizer.transform([processed])
    ml_prediction = model.predict(vec_feedback)[0]
    ml_sentiment = {1: 'Positive', 0: 'Neutral', -1: 'Negative'}[ml_prediction]


    return {
        'feedback': new_feedback,
        'vader_score': vader_score,
        'vader_sentiment': vader_sentiment,
        'ml_sentiment': ml_sentiment
    }


new_feedbacks = [
    "Your online banking system has been down for two days now",
    "I'm impressed with how quickly the fraud alert was resolved",
    "The new branch location is convenient but parking is limited"
]


print("\nTesting with new feedback examples:")
for feedback in new_feedbacks:
    result = predict_sentiment(feedback)
    print(f"\nFeedback: {result['feedback']}")
    print(f"VADER Score: {result['vader_score']:.2f}")
    print(f"VADER Sentiment: {result['vader_sentiment']}")
    print(f"ML Model Sentiment: {result['ml_sentiment']}")
```

```
financial_keywords = [
    'fee', 'fees', 'interest', 'rate', 'rates', 'loan', 'credit',
    'card', 'branch', 'app', 'online', 'banking', 'service', 'wait',
    'time', 'customer', 'support', 'payment', 'transfer', 'savings'
]

def extract_financial_keywords(text, keywords):
    text_lower = text.lower()
    found_keywords = [keyword for keyword in keywords if keyword in text_lower]
    return found_keywords

df['found_keywords'] = df['feedback'].apply(lambda x: extract_financial_keywords(x,
financial_keywords))

print("\nFeedback with Financial Keywords and Sentiment:")
print(df[['feedback', 'sentiment_category', 'found_keywords']].head(10))
```

```
Testing with new feedback examples:

Feedback: Your online banking system has been down for two days now
VADER Score: 0.00
VADER Sentiment: Neutral
ML Model Sentiment: Negative

Feedback: I'm impressed with how quickly the fraud alert was resolved
VADER Score: 0.30
VADER Sentiment: Positive
ML Model Sentiment: Negative

Feedback: The new branch location is convenient but parking is limited
VADER Score: -0.33
VADER Sentiment: Negative
ML Model Sentiment: Negative

Feedback with Financial Keywords and Sentiment:
                                          feedback sentiment_category  \
0  The new mobile banking app is fantastic and ea...           Positive
1  I'm disappointed with the high fees for intern...           Negative
2  Customer service was helpful in resolving my c...           Positive
3  The waiting time to speak with a representativ...            Neutral
4  I love the new cashback rewards program on my ...           Positive
5  The interest rates on savings accounts are ext...           Positive
6  The bank staff was rude and unhelpful when I v...           Negative
7  Setting up automatic payments was simple and s...            Neutral
8  I received conflicting information about my lo...           Negative
9  Very satisfied with how quickly my loan was ap...           Positive

                     found_keywords
0                    [app, banking]
1         [fee, fees, app, transfer]
2  [credit, card, service, customer]
3                      [wait, time]
4                    [credit, card]
```

```
keyword_sentiment = {}


for keyword in financial_keywords:
    # Get feedback containing this keyword
    keyword_df = df[df['feedback'].str.contains(keyword, case=False)]


    if len(keyword_df) > 0:
      avg_score = keyword_df['compound_score'].mean()
      keyword_sentiment[keyword] = avg_score


keyword_sentiment = {k: v for k, v in keyword_sentiment.items() if not np.isnan(v)}
if keyword_sentiment:
```

```
plt.figure(figsize=(12, 6))
keywords = list(keyword_sentiment.keys())
scores = list(keyword_sentiment.values())
colors = ['red' if x < -0.05 else 'green' if x > 0.05 else 'gray' for x in scores]
sorted_indices = np.argsort(scores)
sorted_keywords = [keywords[i] for i in sorted_indices]
sorted_scores = [scores[i] for i in sorted_indices]
sorted_colors = [colors[i] for i in sorted_indices]

plt.barh(sorted_keywords, sorted_scores, color=sorted_colors)
plt.axvline(x=0, color='black', linestyle='-', alpha=0.3)
plt.title('Average Sentiment Score by Financial Keyword')
plt.xlabel('Average Sentiment Score')
plt.tight_layout()
plt.show()
```