- INDEX -

Practical No.	Practical Title	Date	Signature
1	Write a program to Implement sentence segmentation and word tokenization. a. Tokenization using Python’s split() function b. Tokenization using Regular Expressions (RegEx) c. Tokenization using NLTK d. Tokenization using the spaCy library		
2	Write a program to generate unigram, bigram, and trigram models from given text.		
3	Write a program to implement stemming and lemmatization algorithms. a. PorterStemmer b. LancasterStemmer c. SnowballStemmer d. Lemmatization using NLTK and spacy		
4	Write a program to Implement syntactic parsing of a given text.		
5	Write a program to Implement dependency parsing of a given text		
6	Implementing Named Entity Recognition (NER) using spaCy or similar libraries.		
7	Write a program to Implement Text Summarization for the given sample text.		
8	Implementing sentiment analysis for customer feedback in financial services.		

Name: Siddhesh Santosh Teli Roll No: 24

Practical No 01

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

a. Tokenization using Python's split() function

b. Tokenization using Regular Expressions (RegEx)

c. Tokenization using NLTK

d. Tokenization using the spaCy library

e. Tokenization using Gensim

Theory:

Tokenization:
Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

a. Tokenization using Python's split() function

Theory:

The Python split() method divides a string into a list. Values in the resultant list are separated based on a separator character. The separator is whitespace by default.

Program for Word Tokenization:

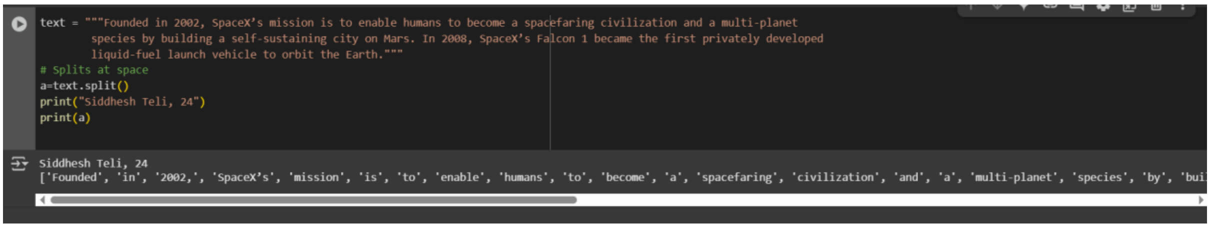
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""

Splits at space

a=text.split()

print(a)

Output:



```
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("Siddhesh Teli, 24")
print(a)
```

```
Siddhesh Teli, 24
['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'bui
```

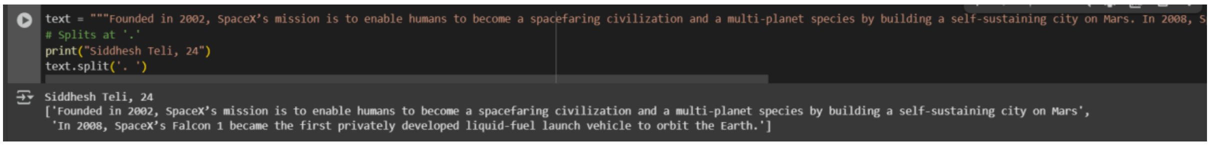
Program for Sentence Tokenization:

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 '.'

text.split('. ')

Output:



```
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, S
# Splits at '.'
print("Siddhesh Teli, 24")
text.split('. ')

Siddhesh Teli, 24
['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.']
```

b. Tokenization using Regular Expressions (RegEx)

Theory:

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern.

Program for Word Tokenization:

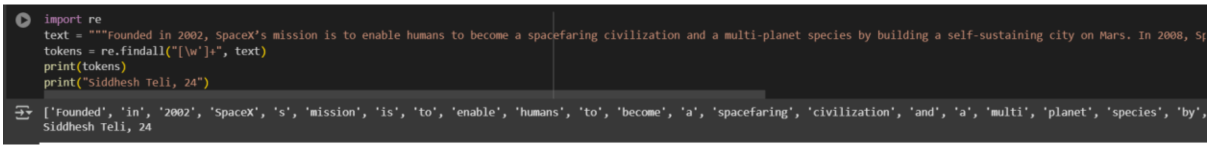
import re

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:



```
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, Sp
tokens = re.findall("[\w']+", text)
print(tokens)
print("Siddhesh Teli, 24")

['Founded', 'in', '2002', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi', 'planet', 'species', 'by',
Siddhesh Teli, 24
```

Program for Sentence Tokenization:

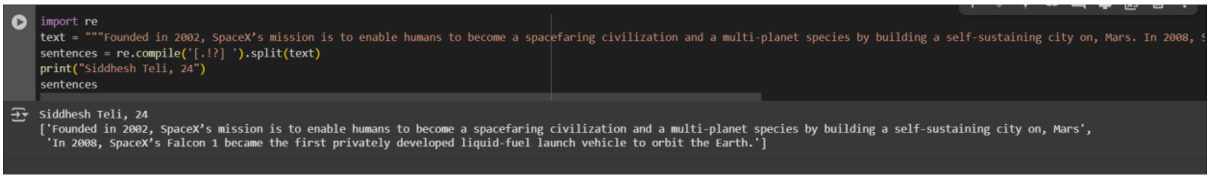
import re

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:



c. Tokenization using NLTK

Theory:

The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging, and semantic reasoning.

Program for Word Tokenization:

import nltk

nltk.download('punkt')

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:

Program for Sentence Tokenization:

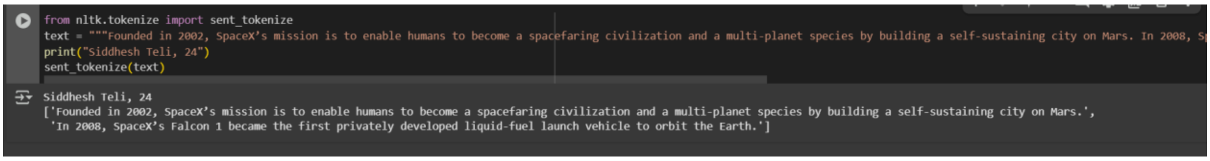
Sentence Tokenization

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:



d. Tokenization using the spaCy library

Theory:

Spacy is an open-source software python library used in advanced natural language processing and machine learning. It will be used to build information extraction, natural language understanding systems, and pre-process text for deep learning.

Program for Word Tokenization:

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."""

"nlp" 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:

Program for Sentence Tokenization:

import spacy

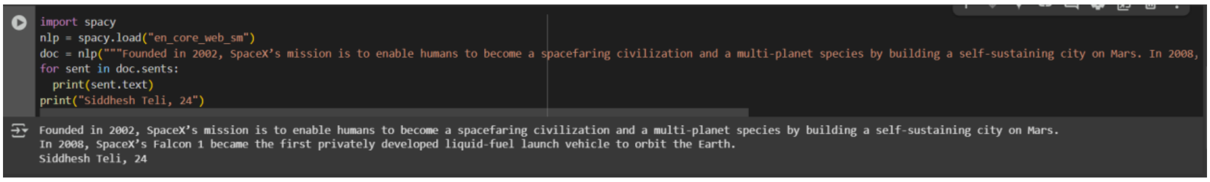
nlp = spacy.load("en_core_web_sm")

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:



e. Tokenization using Gensim

Theory:

It is an open-source library for unsupervised topic modeling and natural language processing and is designed to automatically extract semantic topics from a given document. We can use the gensim. utils class to import the tokenize method for performing word tokenization.

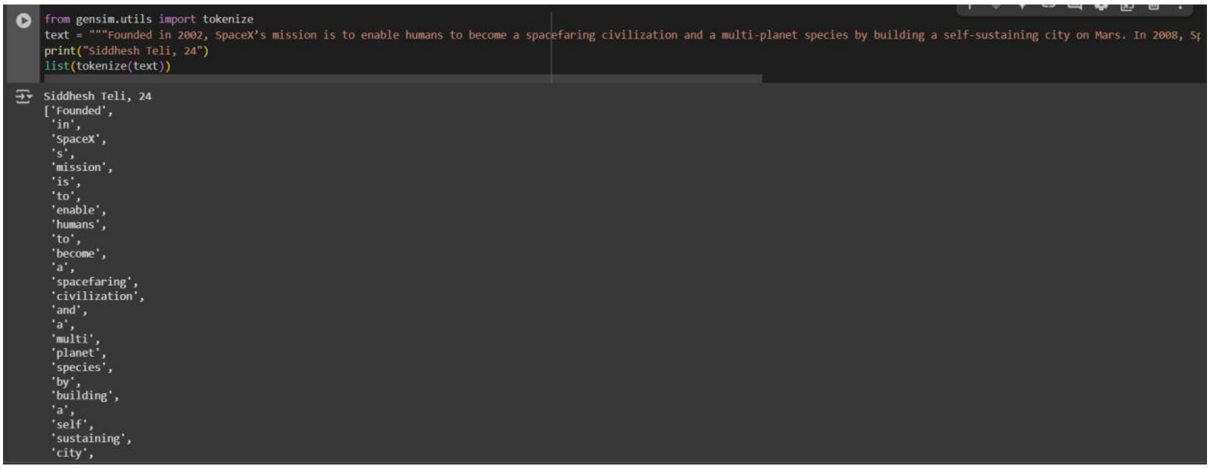
Program for Word Tokenization:

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:



```
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, Sp
print("Siddhesh Teli, 24")
list(tokenize(text))

Siddhesh Teli, 24
['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',
```

Program for Sentence Tokenization:

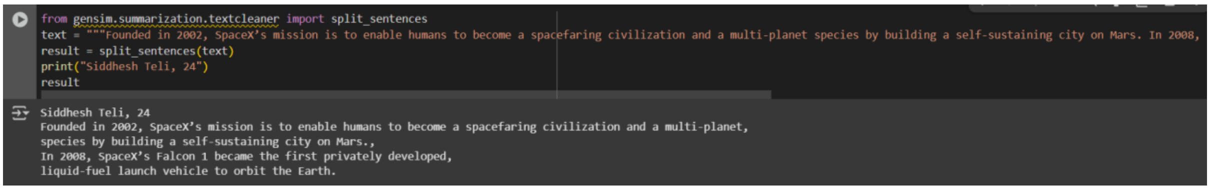
from gensim.summarization.textcleaner import split_sentences

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."""

result = split_sentences(text)

result

Output:



```
from gensim.summarization.textcleaner import split_sentences
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,
result = split_sentences(text)
print("Siddhesh Teli, 24")
result

Siddhesh Teli, 24
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.
```

Practical No 02

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

Theory:

Language Model:

A language model learns to predict the probability of a sequence of words.

There are primarily two types of Language Models:
1. **Statistical Language Models:** These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM) and certain linguistic rules to learn the probability distribution of words
2. **Neural Language Models:** These are new players in the NLP town and have surpassed the statistical language models in their effectiveness. They use different kinds of Neural Networks to model language

N-gram Language Model:

What are N-grams?

N-grams are simply sequences of N items from a given text. In NLP, these items are typically words or characters. The "N" refers to the length of the sequence:

- **Unigrams (n=1)**: Single words or tokens (e.g., "Natural", "language", "processing")
- **Bigrams (n=2)**: Pairs of consecutive words (e.g., "Natural language", "language processing")
- **Trigrams (n=3)**: Three consecutive words (e.g., "Natural language processing")

N-grams help us understand how words appear together in language, which is crucial for many NLP tasks like predictive text, machine translation, and speech recognition.

An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. If we have a good N-gram model, we can predict p(w | h) – what is the probability of seeing the word *w* given a history of previous words *h* – where the history contains n-1 words.

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:

```
Siddhesh Teli, 24
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. 03

Aim : Write a program to Implement stemming and lemmatization algorithm

Theory:

Stemmer:

Lemmatization:

LIBRARY :

Install the natural language toolkit library → pip install nltk

a. PorterStemmer

Code:

```
import nltk
from nltk.stem.porter import PorterStemmer
porter_stemmer  = PorterStemmer()
text = "Pythoners are very 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 very is - veri
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
Siddhesh Teli, 24
```

b. LancasterStemmer

Code:

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

Output:

```
Stemming for studies is - study
Stemming for studying is - study
Stemming for cries is - cri
Stemming for cry is - cry
Siddhesh Teli, 24
```

c. SnowballStemmer

Code:

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

Output:

```
generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat
having ---> having
Siddhesh Teli, 24
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

d. Lemmatization

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
Siddhesh Teli, 24
```

Lemmatization using Spacy

```
python -m spacy download en_core_web_sm
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  |  6033511944150694480
rafting  |  raft  |  7154368781129989833
meeting  |  meeting  |  14798207169164081740
better  |  well  |  4525988469032889948
Siddhesh Teli, 24
```

Practical No. 04

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

Theory:

Syntactic Parsing:
Syntactic parsing is a technique by which segmented, tokenized, and part-of-speech tagged text is assigned a structure that reveals the relationships between tokens governed by syntax rules, e.g. by grammars.

A sentence is structured as follows:

Sentence = S = Noun Phrase + Verb Phrase + Preposition Phrase

S = NP + VP + PP

The different word groups that exist according to English grammar rules are:
Noun Phrase(NP): Determiner + Nominal Nouns = DET + Nominal
Verb Phrase (VP): Verb + range of combinations
Prepositional Phrase (PP): Preposition + Noun Phrase = P + NP

We can make different forms and structures versions of the noun phrase, verb phrase, and prepositional phrase and join in a sentence.

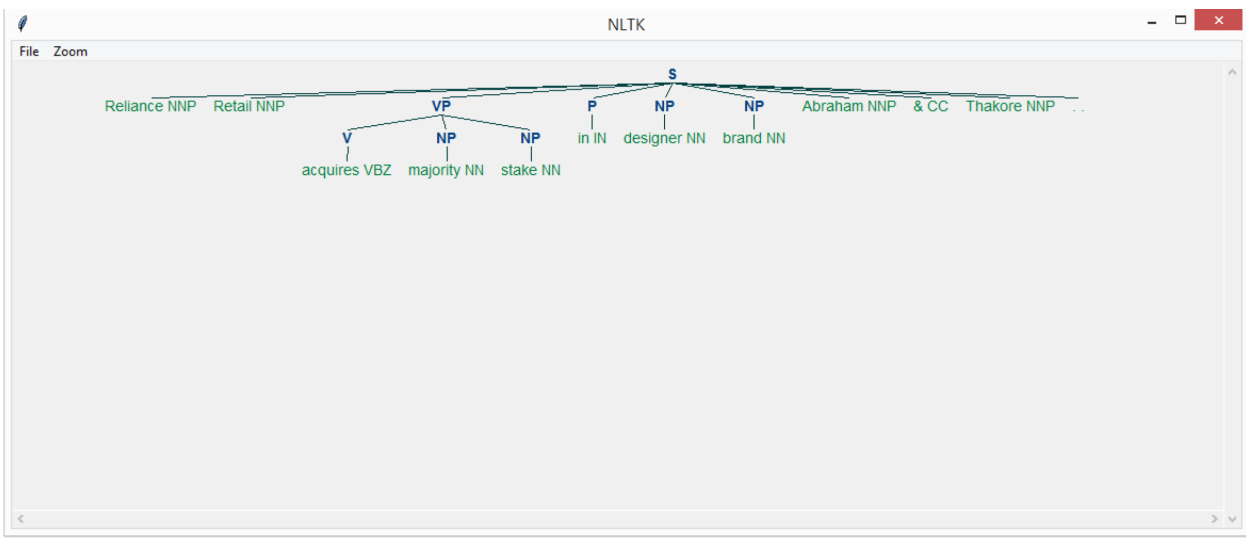
Code:

```
# Import required libraries
import nltk
nltk.download('punkt')          #pre-trained Punkt tokenizer, which is used to tokenize the words.
nltk.download('averaged_perceptron_tagger')
 #averaged_perceptron_tagger: is used to tag those tokenized words to Parts of Speech
from nltk import pos_tag, word_tokenize, RegexpParser

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

# Find all parts of speech in above sentence
tagged = pos_tag(word_tokenize(sample_text))

#Extract all parts of speech from any text
chunker = RegexpParser("""
                NP: {<DT>?<JJ>*<NN>}    #To extract Noun Phrases
                P: {<IN>}              #To extract Prepositions
                V: {<V.*>}             #To extract Verbs
                PP: {<p> <NP>}          #To extract Prepositional Phrases
```

```
VP: {<V> <NP|PP>*}      #To extract Verb Phrases
""")

# Print all parts of speech in above sentence
output = chunker.parse(tagged)
print("After Extracting\n", output)
output.draw()
```

Output:

```
After Extracting
 (S
  Reliance/NNP
  Retail/NNP
  (VP (V acquires/VBZ) (NP majority/NN) (NP stake/NN))
  (P in/IN)
  (NP designer/NN)
  (NP brand/NN)
  Abraham/NNP
  &/CC
  Thakore/NNP
  ./.)
Siddhesh, 24
```



Practical No. 05

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

Theory:

Dependency Parsing:
Dependency Parsing is the process of analyzing the grammatical structure in a sentence and finding related words and the type of relationship between them.

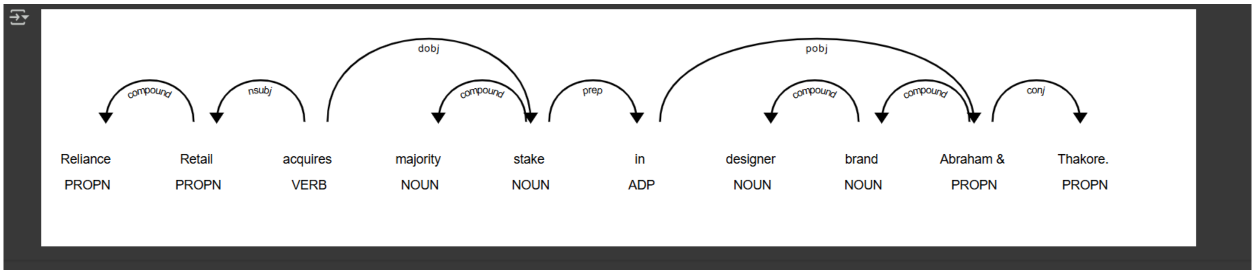
Code:

```
import spacy

# Loading the model
nlp=spacy.load('en_core_web_sm')
text = "Reliance Retail acquires majority stake in designer brand Abraham & Thakore."

# Creating Doc object
doc=nlp(text)
print ("{:<15} | {:<8} | {:<15} | {:<20}".format('Token','Relation','Head', 'Children'))
print ("-" * 70)

for token in doc:
  # Print the token, dependency nature, head and all dependents of the token
  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            | Children
----------------------------------------------------------------------
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        | []
Siddhesh, 24
```

```
# Importing visualizer
from spacy import displacy

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



Practical No 06

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

Theory:

Named-entity recognition is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc. European is NORD (nationalities or religious or political groups), Google is an organization, $5.1 billion is monetary value and Wednesday is a date object. They are all correct.

A. Implementing NER using NLTK Library

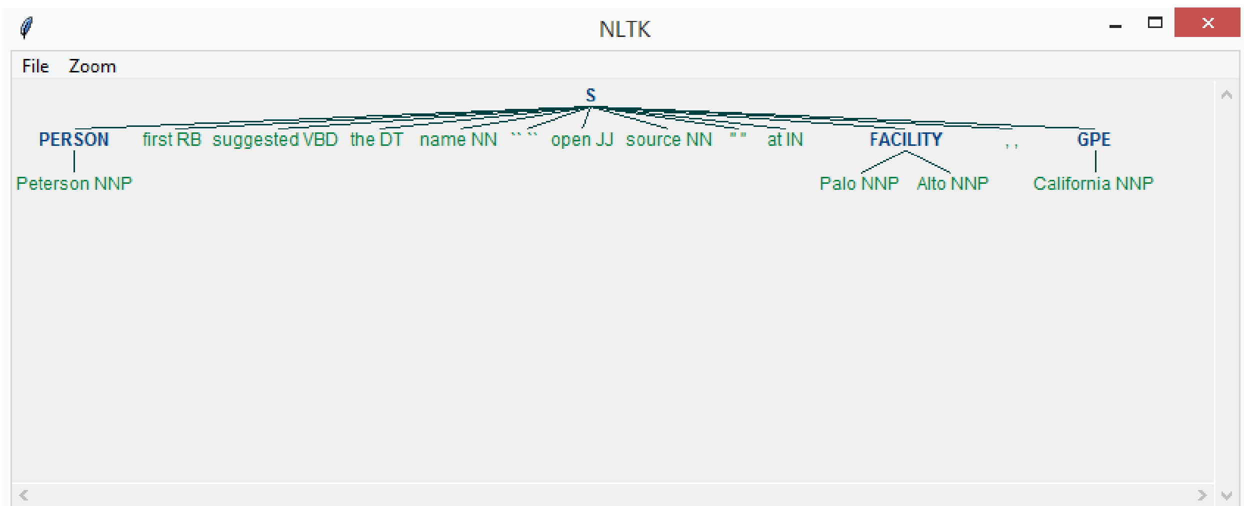
Code:

```
sentence = 'Peterson first suggested the name "open source" at Palo Alto, California'

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

words = nltk.word_tokenize(sentence)
pos_tagged = nltk.pos_tag(words)

nltk.download('maxent_ne_chunker')
nltk.download('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:])
ne_tagged.draw()
```

Name: Siddhesh Santosh Teli Roll No: 24

Output:

```
(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))

Recognized named entities:
PERSON [('Peterson', 'NNP')]
FACILITY [('Palo', 'NNP'), ('Alto', 'NNP')]
GPE [('California', 'NNP')]
Siddhesh Teli, 24
```



B. Implementing NER using spaCy Library

Code:

```
import spacy

from spacy import displacy

NER = spacy.load("en_core_web_sm")

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 the Prime 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_)
```

Output:

```
The Indian Space Research Organisation ORG
India GPE
Bengaluru GPE
Department of Space ORG
India GPE
ISRO ORG
DOS ORG
Siddhesh Teli, 24
```

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

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 the Prime Minister of India GPE while Chairman of ISRO ORG acts as executive of DOS ORG as well.

Practical No. 07

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

Theory:

Summarization is the task of condensing a piece of text to a shorter version, reducing the size of the initial text while at the same time preserving key informational elements and the meaning of content.

A. Implementing Text summarization using NLTK Library

Text 1:
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.

Code:
```
nltk.download('stopwords')

# importing libraries
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize

# Input text - to summarize
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. """

# Tokenizing the text
stopWords = set(stopwords.words("english"))
```

```
words = word_tokenize(text)

# Creating a frequency table to keep the
# score of each word

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

# Creating a dictionary to keep the score
# of each sentence
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 value of a sentence from the original text

average = int(sumValues / len(sentenceValue))

# Storing sentences into our summary.
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)
```

Output:

```
Siddhesh, 24
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.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

B. Implementing Text summarization using spaCy Library

Code:

```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
from collections import Counter
from heapq import nlargest

doc ="""Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.In its application across business problems, machine learning is also referred to as predictive analytics."""

nlp = spacy.load('en_core_web_sm')

doc = nlp(doc)

len(list(doc.sents)) # to find the number of sentences in the given string
```

```
In [12]: nlp = spacy.load('en_core_web_sm')
         doc = nlp(doc)
         len(list(doc.sents)) # to find the number of sentences in the given string

Out[12]: 7
```

```
keyword = []

stopwords = list(STOP_WORDS)

pos_tag = ['PROPN', 'ADJ', 'NOUN', 'VERB']

for token in doc:

    if(token.text in stopwords or token.text in punctuation):

        continue
```

```
    if(token.pos_ in pos_tag):
        keyword.append(token.text)
#Calculating frequency of each token using the Counter function
freq_word = Counter(keyword)
print(freq_word.most_common(5))
type(freq_word)
```

```
In [22]: #Calculating frequency of each token using the Counter function
         freq_word = Counter(keyword)
         print(freq_word.most_common(5))

         [('learning', 8), ('Machine', 4), ('study', 3), ('algorithms', 3), ('task', 3)]

In [23]: type(freq_word)

Out[23]: collections.Counter
```

```
#Normalization
max_freq = Counter(keyword).most_common(1)[0][1]
for word in freq_word.keys():
        freq_word[word] = (freq_word[word]/max_freq)
freq_word.most_common(5)
```

```
In [24]: #Normalization
         max_freq = Counter(keyword).most_common(1)[0][1]
         for word in freq_word.keys():
                 freq_word[word] = (freq_word[word]/max_freq)
         freq_word.most_common(5)

Out[24]: [('learning', 1.0),
          ('Machine', 0.5),
          ('study', 0.375),
          ('algorithms', 0.375),
          ('task', 0.375)]
```

```
#Weighing sentences
sent_strength={}
for sent in doc.sents:
    for word in sent:
        if word.text in freq_word.keys():
            if sent in sent_strength.keys():
                sent_strength[sent]+=freq_word[word.text]
            else:
                sent_strength[sent]=freq_word[word.text]
print(sent_strength)
```

```
In [25]: #Weighing sentences
sent_strength={}
for sent in doc.sents:
    for word in sent:
        if word.text in freq_word.keys():
            if sent in sent_strength.keys():
                sent_strength[sent]+=freq_word[word.text]
            else:
                sent_strength[sent]=freq_word[word.text]
print(sent_strength)
```

```
{Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task.: 4.125, Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.: 4.625, Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task.: 4.25, Machine learning is closely related to computational statistics, which focuses on making predictions using computers.: 2.625, The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning.: 3.125, Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.: 4.25, In its application across business problems, machine learning is also referred to as predictive analytics.: 2.25}
```

#Summarizing the string

summarized_sentences = nlargest(3, sent_strength, key=sent_strength.get)

print(summarized_sentences)

print(type(summarized_sentences[0]))

final_sentences = [w.text for w in summarized_sentences]

summary = ' '.join(final_sentences)

print(summary)

Output:

```
In [18]: #Summarizing the string
summarized_sentences = nlargest(3, sent_strength, key=sent_strength.get)
print(summarized_sentences)
```

```
[Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task., Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task., Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.]
```

```
In [19]: print(type(summarized_sentences[0]))
```

```
<class 'spacy.tokens.span.Span'>
```

```
In [20]: final_sentences = [ w.text for w in summarized_sentences ]
summary = ' '.join(final_sentences)
print(summary)
```

```
Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.
```

C. Implementing Text summarization using gensim Library

Code:

!pip install gensim_sum_ext

doc ="""Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.In its application across business problems, machine learning is also referred to as predictive analytics."""

from gensim.summarization import summarize

summary = summarize(doc)

Output:

```
In [35]: print(doc)

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively i
mprove their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "t
raining data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learn
ing algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it i
s infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to c
omputational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers m
ethods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learnin
g, and focuses on exploratory data analysis through unsupervised learning.In its application across business problems, machine
learning is also referred to as predictive analytics.

In [33]: len(doc)

Out[33]: 1069

In [36]: print(summary)

Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions o
r decisions without being explicitly programmed to perform the task.

In [34]: len(summary)

Out[34]: 195
```

Name: Siddhesh Santosh Teli Roll No: 24

Practical No. 08

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

```
# Import necessary libraries
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

# Download necessary NLTK data
nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('punkt')

# Create sample financial services customer feedback data
# In a real scenario, you would import your own data
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"
    ]
}

# Create DataFrame
df = pd.DataFrame(data)

# Display the sample data
print("Sample financial services customer feedback:")
print(df.head())

# Basic text preprocessing function
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return text

# Apply preprocessing
df['processed_feedback'] = df['feedback'].apply(preprocess_text)

# APPROACH 1: Rule-based sentiment analysis using VADER
# Initialize VADER sentiment analyzer
sid = SentimentIntensityAnalyzer()

# Function to get sentiment scores
def get_sentiment_score(text):
    return sid.polarity_scores(text)

# Apply sentiment analysis to the original text (VADER works better with
original text including punctuation)
df['sentiment_scores'] = df['feedback'].apply(get_sentiment_score)

# Extract the compound score
df['compound_score'] = df['sentiment_scores'].apply(lambda x:
x['compound'])

# Categorize sentiment based on compound score
df['sentiment_category'] = df['compound_score'].apply(
```

```
    lambda x: 'Positive' if x >= 0.05 else ('Negative' if x <= -0.05 else
'Neutral')
)

# Display results with sentiments
print("\nSentiment Analysis Results:")
print(df[['feedback', 'compound_score', 'sentiment_category']].head(10))

# Count the sentiment categories
sentiment_counts = df['sentiment_category'].value_counts()
print("\nSentiment Distribution:")
print(sentiment_counts)

# Visualize sentiment distribution
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: Simple Machine Learning approach (for demonstration)
# In a real scenario, you would need more labeled data

# Create a simple labeled dataset for demonstration
df['label'] = df['sentiment_category'].map({'Positive': 1, 'Neutral': 0,
'Negative': -1})

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    df['processed_feedback'],
    df['label'],
    test_size=0.3,
    random_state=42
)

# Create a bag of words representation
vectorizer = CountVectorizer(max_features=1000)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
# Train a simple model (logistic regression)
model = LogisticRegression(max_iter=1000)
model.fit(X_train_vec, y_train)

# Make predictions
y_pred = model.predict(X_test_vec)

# Evaluate the model
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']))

# Function to predict sentiment for new feedback
def predict_sentiment(new_feedback):
    # Preprocess
    processed = preprocess_text(new_feedback)

    # VADER sentiment
    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')

    # ML model prediction
    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
    }

# Test with new examples
new_feedbacks = [
```

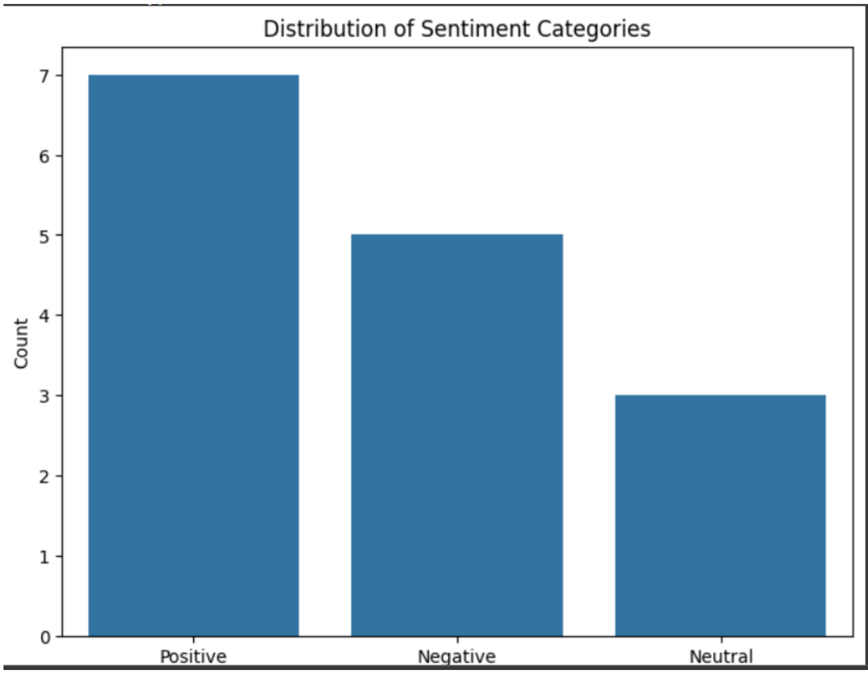
```
    "Your online banking system has been down for two days now",
    "I'm impressed with how quickly the fraud alert was resolved",
    "The new branch location is convenient but parking is limited"
]

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']}")

# Extract financial-specific keywords to identify common themes
financial_keywords = [
    'fee', 'fees', 'interest', 'rate', 'rates', 'loan', 'credit',
    'card', 'branch', 'app', 'online', 'banking', 'service', 'wait',
    'time', 'customer', 'support', 'payment', 'transfer', 'savings'
]

# Check for financial keywords in feedback
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))

# Display feedback with financial keywords and sentiment
print("\nFeedback with Financial Keywords and Sentiment:")
print(df[['feedback', 'sentiment_category', 'found_keywords']].head(10))

# Analyze sentiment by keyword
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:
        # Calculate average sentiment score for this keyword
        avg_score = keyword_df['compound_score'].mean()
        keyword_sentiment[keyword] = avg_score

# Filter out keywords that don't appear in any feedback
keyword_sentiment = {k: v for k, v in keyword_sentiment.items() if not
np.isnan(v)}

# Visualize sentiment by keyword
if keyword_sentiment:
    plt.figure(figsize=(12, 6))
    keywords = list(keyword_sentiment.keys())
    scores = list(keyword_sentiment.values())

    # Create color map based on sentiment scores
    colors = ['red' if x < -0.05 else 'green' if x > 0.05 else 'gray' for
x in scores]

    # Sort by sentiment score
    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()
```

Output: -

```
Siddhesh, 24
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 ...

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




```
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


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]
5   [interest, rate, rates, savings]
6                           [branch]
7                          [payment]
8                        [loan, app]
9                        [loan, app]
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted s
  warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```