NATURAL LANGUAGE PROCESSING (01CE1718) Lab Manual A.Y. 2025-26

Name:	Raheelkhan M. Lohani
Enrolment No:	92200103021
Class:	7TC4
Batch:	A

Lab	Program	Signature	Marks
1.	Word Analysis		
2.	Word Generation		
3.	Morphologic Analysis		
4.	N-Gram Model		
5.	N-Grams Smoothing		
6.	POS Tagging: Hidden Markov Model		
7.	POS Tagging: Viterbi Decoding		
8.	Building POS Tagger		
9.	Chunking		
10.	Building Chunker		
11.	Stemming And Lemmatization		
12.	Word Sense Disambiguation		
13.	Information Retrieval		
14.	CASE STUDY : Application of NLP- Sentiment Analysis of tweets in Twitter platform		

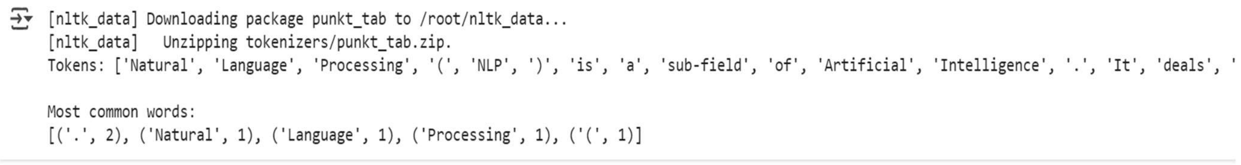
Program 1

AIM: Word Analysis.

Code:

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt_tab')

text = """Natural Language Processing (NLP) is a sub-field of Artificial 
Intelligence.
It deals with the interaction between computers and humans using natural 
language."""

# Tokenize into words
words = word_tokenize(text)

words.append("92200103021")

# Word frequency
freq_dist = nltk.FreqDist(words)

print("Tokens:", words)
print("\nMost common words:")
print(freq_dist.most_common(5))
```

Output:



```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Tokens: ['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'is', 'a', 'sub-field', 'of', 'Artificial', 'Intelligence', '.', 'It', 'deals', '

Most common words:
[('.', 2), ('Natural', 1), ('Language', 1), ('Processing', 1), ('(', 1)]
```

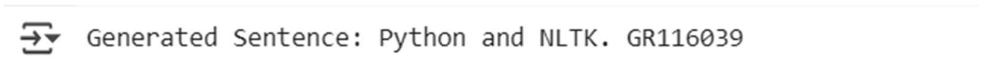
Program 2

AIM: Word Generation.

Code:

```
import random

# Example bigram model for text generation
text = "Raheelkhan Lohani is working on NLP tasks for word generation using Python
and NLTK."
tokens = text.split()

tokens.append("GR116039")

bigrams = list(nltk.bigrams(tokens))
bigram_dict = {}

for w1, w2 in bigrams:
    if w1 not in bigram_dict:
        bigram_dict[w1] = []
    bigram_dict[w1].append(w2)

# Generate sentence
current_word = random.choice(tokens)
sentence = [current_word]

for _ in range(10):
    if current_word in bigram_dict:
        next_word = random.choice(bigram_dict[current_word])
        sentence.append(next_word)
        current_word = next_word
    else:
        break

print("Generated Sentence:", ' '.join(sentence))
```

Output:



```
Generated Sentence: Python and NLTK. GR116039
```

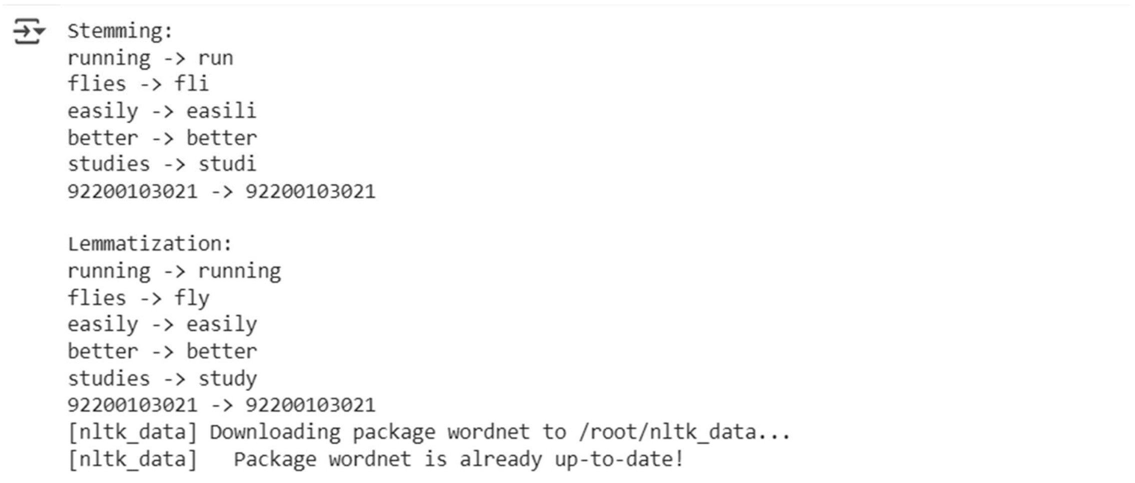
Program 3

AIM: Morphologic Analysis.

Code:

```
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
nltk.download('wordnet')

stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

words = ["running", "flies", "easily", "better", "studies", "92200103021"]

print("Stemming:")
for w in words:
    print(w, "->", stemmer.stem(w))

print("\nLemmatization:")
for w in words:
    print(w, "->", lemmatizer.lemmatize(w))
```

Output:



```
Stemming:
running -> run
flies -> fli
easily -> easili
better -> better
studies -> studi
92200103021 -> 92200103021

Lemmatization:
running -> running
flies -> fly
easily -> easily
better -> better
studies -> study
92200103021 -> 92200103021
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

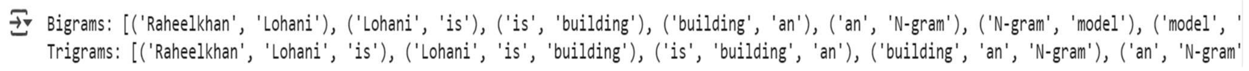
Program 4

AIM: N-Gram Model.

Code:

```
import nltk
from nltk.util import ngrams
from nltk.tokenize import word_tokenize

text = "Raheelkhan Lohani is building an N-gram model for text analysis in Python."
tokens = word_tokenize(text)
tokens.append("GR116039")

# Bigram Model
bigrams = list(ngrams(tokens, 2))
print("Bigrams:", bigrams)

# Trigram Model
trigrams = list(ngrams(tokens, 3))
print("Trigrams:", trigrams)
```

Output:



```
Bigrams: [('Raheelkhan', 'Lohani'), ('Lohani', 'is'), ('is', 'building'), ('building', 'an'), ('an', 'N-gram'), ('N-gram', 'model'), ('model', '
Trigrams: [('Raheelkhan', 'Lohani', 'is'), ('Lohani', 'is', 'building'), ('is', 'building', 'an'), ('building', 'an', 'N-gram'), ('an', 'N-gram'
```

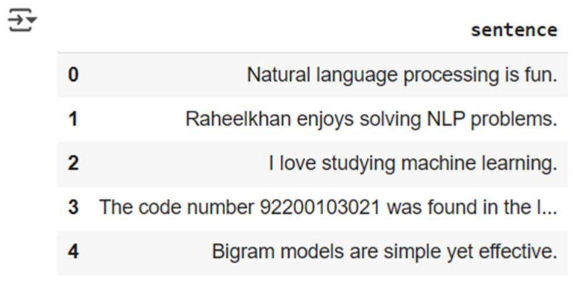
Program 5

AIM: N-Grams Smoothing.

Code:

```
import pandas as pd
import nltk
from collections import defaultdict
import math

nltk.download('punkt_tab')

sentences = [
    "Natural language processing is fun.",
    "Raheelkhan enjoys solving NLP problems.",
    "I love studying machine learning.",
    "The code number 92200103021 was found in the log file.",
    "Bigram models are simple yet effective."
]

df = pd.DataFrame({'sentence': sentences})
df
```

Output:



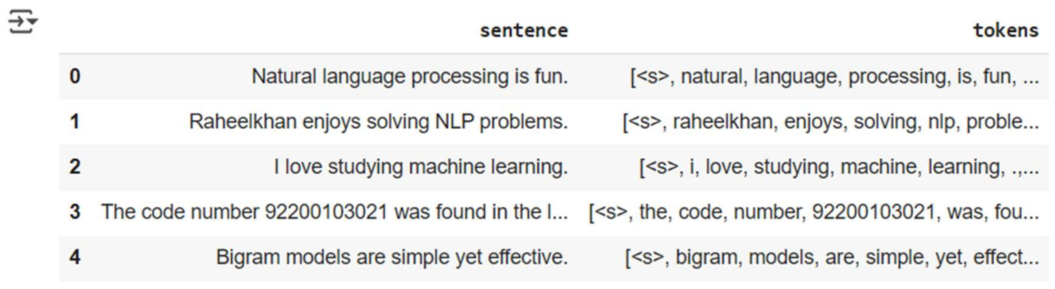
	sentence
0	Natural language processing is fun.
1	Raheelkhan enjoys solving NLP problems.
2	I love studying machine learning.
3	The code number 92200103021 was found in the l...
4	Bigram models are simple yet effective.

Code:

```
# Tokenization
def tokenize(sent):
    return ["<s>"] + nltk.word_tokenize(sent.lower()) + ["</s>"]

df['tokens'] = df['sentence'].apply(tokenize)
```

```
df[['sentence', 'tokens']]
```

Output:

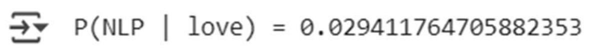


	sentence	tokens
0	Natural language processing is fun.	[<s>, natural, language, processing, is, fun, ...
1	Raheelkhan enjoys solving NLP problems.	[<s>, raheelkhan, enjoys, solving, nlp, proble...
2	I love studying machine learning.	[<s>, i, love, studying, machine, learning, .,...
3	The code number 92200103021 was found in the l...	[<s>, the, code, number, 92200103021, was, fou...
4	Bigram models are simple yet effective.	[<s>, bigram, models, are, simple, yet, effect...

Code:

```
# Build bigram counts
bigram_counts = defaultdict(lambda: defaultdict(int))
unigram_counts = defaultdict(int)

for tokens in df['tokens']:
    for i in range(len(tokens)-1):
        unigram_counts[tokens[i]] += 1
        bigram_counts[tokens[i]][tokens[i+1]] += 1

# Add last token count
for tokens in df['tokens']:
    unigram_counts[tokens[-1]] += 1

# Laplace Smoothing Probability
vocab = set(word for tokens in df['tokens'] for word in tokens)
V = len(vocab)

def bigram_prob(w1, w2):
    return (bigram_counts[w1][w2] + 1) / (unigram_counts[w1] + V)

# Example probability
print("P(NLP | love) =", bigram_prob("solving", "love"))
```

Output:

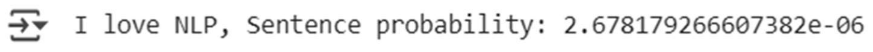


```
P(NLP | love) = 0.029411764705882353
```

Code:

```
# Sentence probability
def sentence_prob(sent):
    tokens = ["<s>"] + nltk.word_tokenize(sent.lower()) + ["</s>"]
    prob = 0
    for i in range(len(tokens)-1):
        prob += math.log(bigram_prob(tokens[i], tokens[i+1]))
    return math.exp(prob)

print("I love NLP, Sentence probability:", sentence_prob("I love NLP"))
```

Output:



```
I love NLP, Sentence probability: 2.678179266607382e-06
```

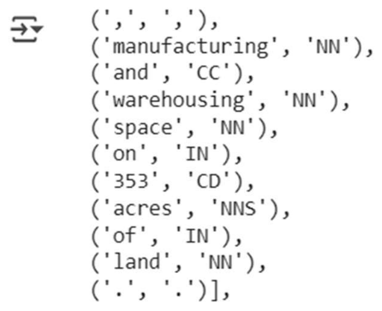
Program 6

AIM: POS Tagging: Hidden Markov Model.

Code:

```
import nltk
from nltk.corpus import treebank
import random
import math
from collections import defaultdict

nltk.download('treebank')
nltk.download('punkt_tab')

# Load tagged sentences (Treebank has ~3900 sentences)
tagged_sents = list(treebank.tagged_sents())
tagged_sents
```

Output:



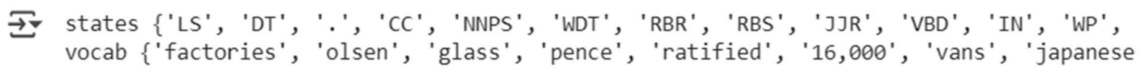
```
(',', ','),
('manufacturing', 'NN'),
('and', 'CC'),
('warehousing', 'NN'),
('space', 'NN'),
('on', 'IN'),
('353', 'CD'),
('acres', 'NNS'),
('of', 'IN'),
('land', 'NN'),
('.', '.')],
```

Code:

```
# Shuffle to avoid bias
random.shuffle(tagged_sents)

# Train-test split (80-20)
train_size = int(len(tagged_sents) * 0.8)
train_data = tagged_sents[:train_size]
```

```
test_data = tagged_sents[train_size:]

# Extract states (POS tags) and vocabulary
states = set()
vocab = set()

for sent in train_data:
    for word, tag in sent:
        states.add(tag)
        vocab.add(word.lower())
print("states",states)
print("vocab",vocab)
```

Output:

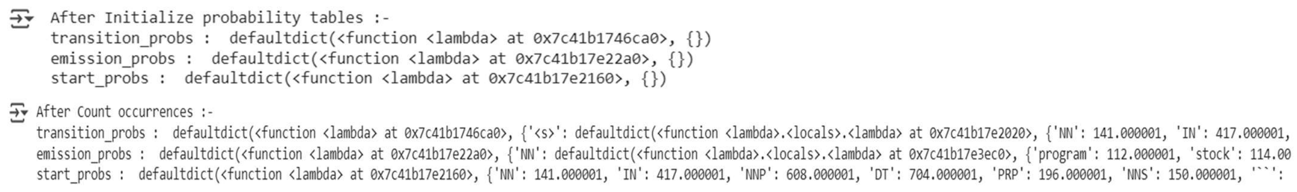


```
states {'LS', 'DT', '.', 'CC', 'NNPS', 'WDT', 'RBR', 'RBS', 'JJR', 'VBD', 'IN', 'WP',
vocab {'factories', 'olsen', 'glass', 'pence', 'ratified', '16,000', 'vans', 'japanese
```

Code:

```
# Initialize probability tables
transition_probs = defaultdict(lambda: defaultdict(lambda: 1e-6))  # add small
smoothing
emission_probs = defaultdict(lambda: defaultdict(lambda: 1e-6))
start_probs = defaultdict(lambda: 1e-6)

print("After Initialize probability tables :- ")
print("transition_probs : ",transition_probs)
print("emission_probs : ",emission_probs)
print("start_probs : ",start_probs)

# Count occurrences
for sent in train_data:
    prev_tag = "<s>"
    start_probs[sent[0][1]] += 1
    for word, tag in sent:
        emission_probs[tag][word.lower()] += 1
        transition_probs[prev_tag][tag] += 1
        prev_tag = tag
    transition_probs[prev_tag]["</s>"] += 1

print("After Count occurrences :- ")
print("transition_probs : ",transition_probs)
print("emission_probs : ",emission_probs)
print("start_probs : ",start_probs)
```

Output:



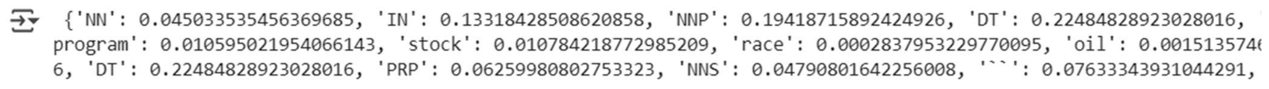
```
After Initialize probability tables :-
transition_probs :  defaultdict(<function <lambda> at 0x7c41b1746ca0>, {})
emission_probs :  defaultdict(<function <lambda> at 0x7c41b17e22a0>, {})
start_probs :  defaultdict(<function <lambda> at 0x7c41b17e2160>, {})
```

```
After Count occurrences :-
transition_probs :  defaultdict(<function <lambda> at 0x7c41b1746ca0>, {'<s>': defaultdict(<function <lambda>.<locals>.<lambda> at 0x7c41b17e2020>, {'NN': 141.000001, 'IN': 417.000001,
emission_probs :  defaultdict(<function <lambda> at 0x7c41b17e22a0>, {'NN': defaultdict(<function <lambda>.<locals>.<lambda> at 0x7c41b17e3ec0>, {'program': 112.000001, 'stock': 114.00
start_probs :  defaultdict(<function <lambda> at 0x7c41b17e2160>, {'NN': 141.000001, 'IN': 417.000001, 'NNP': 608.000001, 'DT': 704.000001, 'PRP': 196.000001, 'NNS': 150.000001, '``':
```

Code:

```
# Normalize to probabilities
for prev_tag in transition_probs:
    total = sum(transition_probs[prev_tag].values())
    for tag in transition_probs[prev_tag]:
        transition_probs[prev_tag][tag] /= total

for tag in emission_probs:
    total = sum(emission_probs[tag].values())
    for word in emission_probs[tag]:
        emission_probs[tag][word] /= total

total_start = sum(start_probs.values())
for tag in start_probs:
    start_probs[tag] /= total_start

print("transition_probs : ",transition_probs)
print("emission_probs : ",emission_probs)
print("start_probs : ",start_probs)
```

Output:

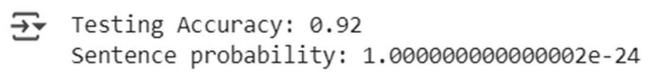


```
{'NN': 0.045033535456369685, 'IN': 0.13318428508620858, 'NNP': 0.19418715892424926, 'DT': 0.22484828923028016,
program': 0.010595021954066143, 'stock': 0.010784218772985209, 'race': 0.0002837953229770095, 'oil': 0.00151357
6, 'DT': 0.22484828923028016, 'PRP': 0.06259980802753323, 'NNS': 0.04790801642256008, '``': 0.07633343931044291,
```

Code:

```
emission_probs["NN"]["92200103021"] = 0.000123

# Function to calculate sentence probability
def bigram_prob(w1, w2):
    return transition_probs[w1].get(w2, 1e-6)

def sentence_prob(sent):
    tokens = ["<s>"] + nltk.word_tokenize(sent.lower()) + ["</s>"]
    prob = 0
    for i in range(len(tokens)-1):
        prob += math.log(bigram_prob(tokens[i], tokens[i+1]))
    return math.exp(prob)

# Evaluate accuracy
correct = 0
total = 0
for sent in test_data:
    for word, tag in sent:
        if word.lower() in emission_probs[tag]:
            correct += 1
        total += 1

accuracy = correct / total
print(f"Testing Accuracy: {accuracy:.2f}")
print("Sentence probability:", sentence_prob("I love NLP"))
```

Output:



```
Testing Accuracy: 0.92
Sentence probability: 1.0000000000000002e-24
```

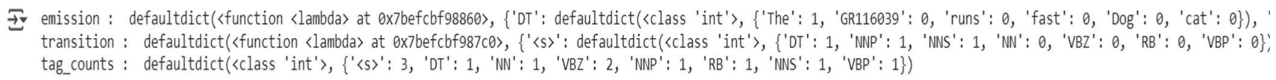
Program 7

AIM: POS Tagging: Viterbi Decoding.

Code:

```
tagged_sentences = [
    [("The", "DT"), ("cat", "NN"), ("sits", "VBZ")],
    [("GR116039", "NNP"), ("runs", "VBZ"), ("fast", "RB")],
    [("Dogs", "NNS"), ("bark", "VBP")]
]

# Build emission and transition probabilities
import collections
emission = collections.defaultdict(lambda: collections.defaultdict(int))
transition = collections.defaultdict(lambda: collections.defaultdict(int))
tag_counts = collections.defaultdict(int)

# Count transitions and emissions
for sent in tagged_sentences:
    prev_tag = "<s>"
    tag_counts[prev_tag] += 1
    for word, tag in sent:
        emission[tag][word] += 1
        transition[prev_tag][tag] += 1
        tag_counts[tag] += 1
        prev_tag = tag
    transition[prev_tag]["</s>"] += 1

print("emission : ",emission)
print("transition : ",transition)
print("tag_counts : ",tag_counts)
```

Output:

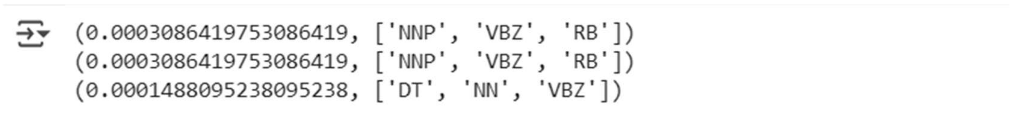


```
emission :  defaultdict(<function <lambda> at 0x7befcbf98860>, {'DT': defaultdict(<class 'int'>, {'The': 1, 'GR116039': 0, 'runs': 0, 'fast': 0, 'Dog': 0, 'cat': 0}), '
transition :  defaultdict(<function <lambda> at 0x7befcbf987c0>, {'<s>': defaultdict(<class 'int'>, {'DT': 1, 'NNP': 1, 'NNS': 1, 'NN': 0, 'VBZ': 0, 'RB': 0, 'VBP': 0})
tag_counts :  defaultdict(<class 'int'>, {'<s>': 3, 'DT': 1, 'NN': 1, 'VBZ': 2, 'NNP': 1, 'RB': 1, 'NNS': 1, 'VBP': 1})
```

Code:

```
# Viterbi function
def viterbi(words):
    states = list(emission.keys())
    V = [{}]
    path = {}

    for y in states:
        V[0][y] = (transition["<s>"][y] + 1) / (tag_counts["<s>"] + len(states))
        V[0][y] *= (emission[y][words[0]] + 1) / (tag_counts[y] + len(emission[y]))
        path[y] = [y]

    for t in range(1, len(words)):
        V.append({})
        new_path = {}

        for y in states:
            (prob, state) = max(
                (V[t-1][y0] * ((transition[y0][y] + 1) / (tag_counts[y0] +
len(states))) *
                 ((emission[y][words[t]] + 1) / (tag_counts[y] + len(emission[y]))),
y0)
                for y0 in states
            )
            V[t][y] = prob
            new_path[y] = path[state] + [y]
        path = new_path

    (prob, state) = max((V[len(words)-1][y], y) for y in states)
    return prob, path[state]

print(viterbi(["GR116039", "runs", "fast"]))
print(viterbi(["GR116039", "runs", "fast"]))
print(viterbi(["Dog", "cat", "runs"]))
```

Output:



```
(0.0003086419753086419, ['NNP', 'VBZ', 'RB'])
(0.0003086419753086419, ['NNP', 'VBZ', 'RB'])
(0.0001488095238095238, ['DT', 'NN', 'VBZ'])
```

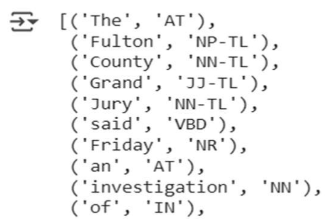
Program 8

AIM: Building POS Tagger.

Code:

```
import nltk

nltk.download('brown')
nltk.download('punkt_tab')
from nltk.corpus import brown

tagged_sents = brown.tagged_sents(categories='news')
train_data = tagged_sents[:4000]
test_data = tagged_sents[4000:]
tagged_sents
```

Output:

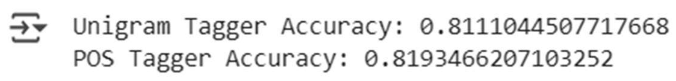


```
[('The', 'AT'),
 ('Fulton', 'NP-TL'),
 ('County', 'NN-TL'),
 ('Grand', 'JJ-TL'),
 ('Jury', 'NN-TL'),
 ('said', 'VBD'),
 ('Friday', 'NR'),
 ('an', 'AT'),
 ('investigation', 'NN'),
 ('of', 'IN'),
```

Code:

```
# Train Unigram + Bigram Tagger
from nltk.tag import UnigramTagger, BigramTagger

unigram_tagger = UnigramTagger(train_data)
bigram_tagger = BigramTagger(train_data, backoff=unigram_tagger)

print("Unigram Tagger Accuracy:", unigram_tagger.accuracy(test_data))
print("POS Tagger Accuracy:", bigram_tagger.accuracy(test_data))
```

Output:

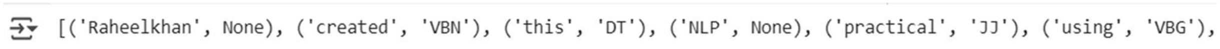


```
Unigram Tagger Accuracy: 0.8111044507717668
POS Tagger Accuracy: 0.8193466207103252
```

Code:

```
# Tag custom sentence
sentence = nltk.word_tokenize("Raheelkhan created this NLP practical using TF-IDF
and HMM models.")
print(bigram_tagger.tag(sentence))
```

Output:



```
[('Raheelkhan', None), ('created', 'VBN'), ('this', 'DT'), ('NLP', None), ('practical', 'JJ'), ('using', 'VBG'),
```

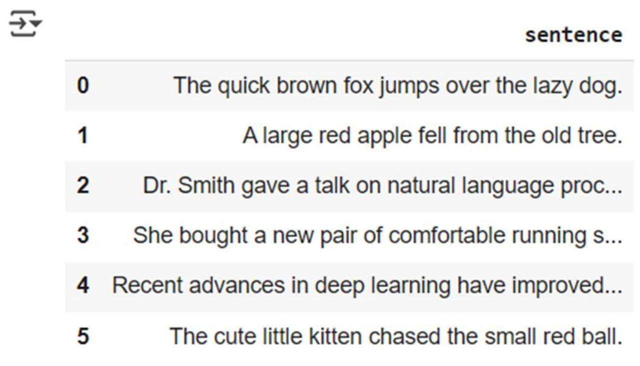
Program 9

AIM: Chunking.

Code:

```
import pandas as pd
import nltk

nltk.download('punkt')                      # tokenizer
nltk.download('averaged_perceptron_tagger') # POS tagger

sentences = [
    "The quick brown fox jumps over the lazy dog.",
    "A large red apple fell from the old tree.",
    "Dr. Smith gave a talk on natural language processing.",
    "She bought a new pair of comfortable running shoes.",
    "Recent advances in deep learning have improved NLP tasks.",
    "The cute little kitten chased the small red ball."
]

df = pd.DataFrame({'sentence': sentences})
df
```

Output:

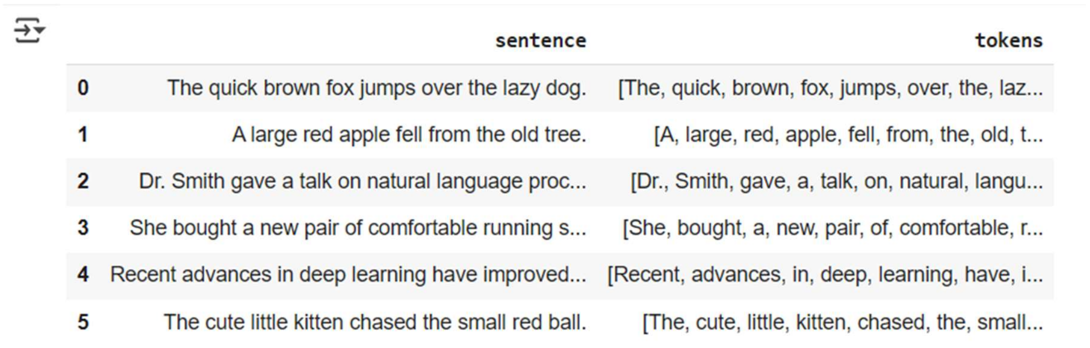


	sentence
0	The quick brown fox jumps over the lazy dog.
1	A large red apple fell from the old tree.
2	Dr. Smith gave a talk on natural language proc...
3	She bought a new pair of comfortable running s...
4	Recent advances in deep learning have improved...
5	The cute little kitten chased the small red ball.

Code:

```
def tokenize_sentence(sent):
    # word_tokenize splits the sentence into words/punctuation tokens
    return nltk.word_tokenize(sent)

df['tokens'] = df['sentence'].apply(tokenize_sentence)
df[['sentence','tokens']]
```

Output:

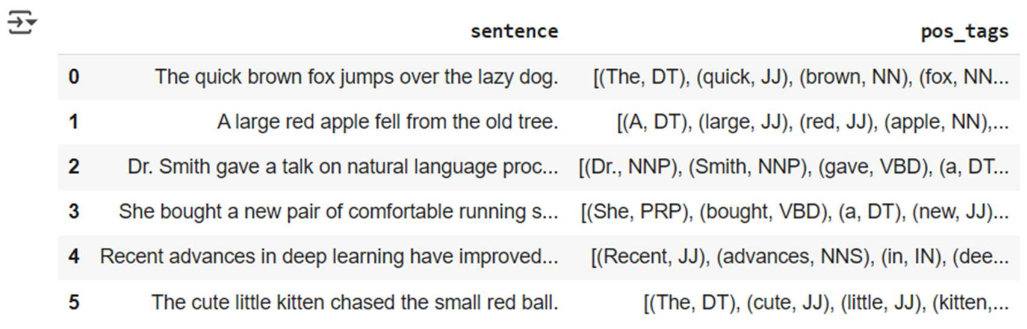


	sentence	tokens
0	The quick brown fox jumps over the lazy dog.	[The, quick, brown, fox, jumps, over, the, laz...
1	A large red apple fell from the old tree.	[A, large, red, apple, fell, from, the, old, t...
2	Dr. Smith gave a talk on natural language proc...	[Dr., Smith, gave, a, talk, on, natural, langu...
3	She bought a new pair of comfortable running s...	[She, bought, a, new, pair, of, comfortable, r...
4	Recent advances in deep learning have improved...	[Recent, advances, in, deep, learning, have, i...
5	The cute little kitten chased the small red ball.	[The, cute, little, kitten, chased, the, small...

Code:

```
# POS tagging
def pos_tag_tokens(tokens):
    # pos_tag returns list of (word, POS) tuples
    return nltk.pos_tag(tokens)

df['pos_tags'] = df['tokens'].apply(pos_tag_tokens)
df[['sentence','pos_tags']]
```

Output:

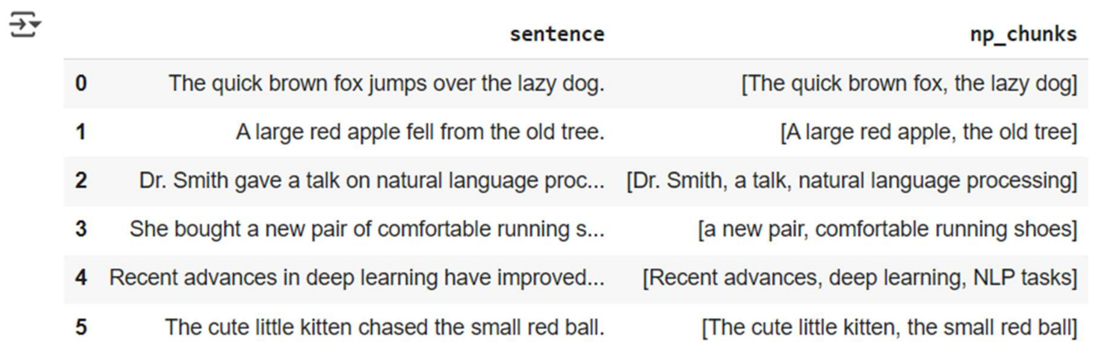


	sentence	pos_tags
0	The quick brown fox jumps over the lazy dog.	[(The, DT), (quick, JJ), (brown, NN), (fox, NN...
1	A large red apple fell from the old tree.	[(A, DT), (large, JJ), (red, JJ), (apple, NN),...
2	Dr. Smith gave a talk on natural language proc...	[(Dr., NNP), (Smith, NNP), (gave, VBD), (a, DT...
3	She bought a new pair of comfortable running s...	[(She, PRP), (bought, VBD), (a, DT), (new, JJ)...
4	Recent advances in deep learning have improved...	[(Recent, JJ), (advances, NNS), (in, IN), (dee...
5	The cute little kitten chased the small red ball.	[(The, DT), (cute, JJ), (little, JJ), (kitten,...

Code:

```
# grammar & parser
grammar = r"""
            NP: {<DT|PRP\$>?<JJ.*>*<NN.*>+}   # Determiner/possessive (optional) +
adjectives (0+) + nouns (1+)
           """
chunk_parser = nltk.RegexpParser(grammar)

# parse + manual extraction of NP chunks
def extract_np_chunks_and_tree(pos_tagged_tokens):
    # pos_tagged_tokens: list of (word,tag) pairs
    tree = chunk_parser.parse(pos_tagged_tokens)  # parse -> returns an nltk.Tree
    np_chunks = []
    # Manually traverse top-level items; find Tree nodes with label 'NP'
    for subtree in tree:
        # subtree is either a tuple (word,tag) or an nltk.Tree
        if isinstance(subtree, nltk.Tree) and subtree.label() == 'NP':
            # subtree.leaves() gives the (word,tag) pairs inside this NP
            words = [word for (word, tag) in subtree.leaves()]
            np_chunks.append(" ".join(words))
    return np_chunks, tree

# Apply to each sentence and save both chunks and tree (tree for printing/debug)
df[['np_chunks','parse_tree']] = df['pos_tags'].apply(lambda x:
pd.Series(extract_np_chunks_and_tree(x)))
df[['sentence','np_chunks']]
```

Output:

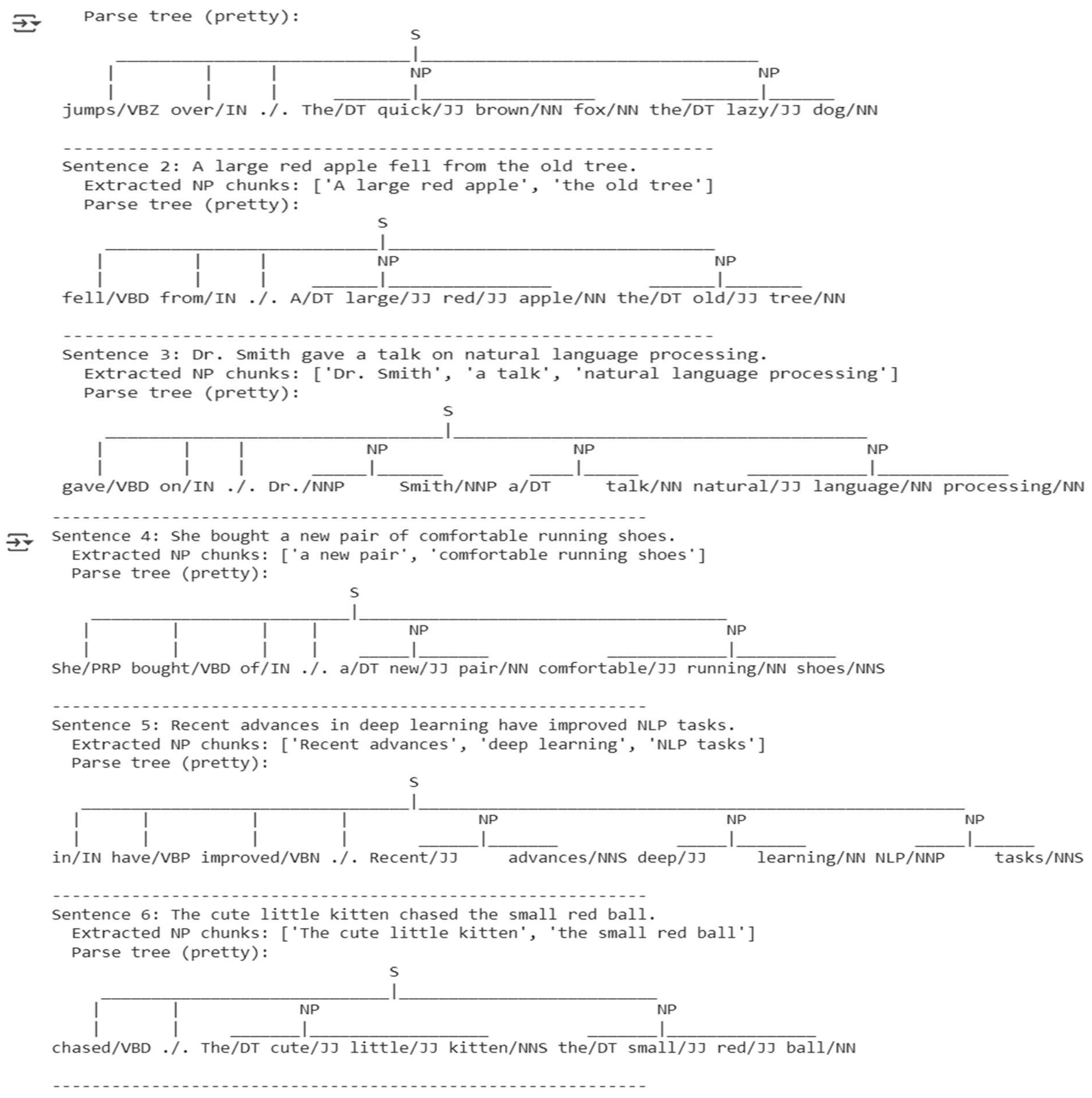


	sentence	np_chunks
0	The quick brown fox jumps over the lazy dog.	[The quick brown fox, the lazy dog]
1	A large red apple fell from the old tree.	[A large red apple, the old tree]
2	Dr. Smith gave a talk on natural language proc...	[Dr. Smith, a talk, natural language processing]
3	She bought a new pair of comfortable running s...	[a new pair, comfortable running shoes]
4	Recent advances in deep learning have improved...	[Recent advances, deep learning, NLP tasks]
5	The cute little kitten chased the small red ball.	[The cute little kitten, the small red ball]

Code:

```
# pretty-print parse trees + chunks
for idx, row in df.iterrows():
    print(f"Sentence {idx+1}: {row['sentence']}")
    print("  Extracted NP chunks:", row['np_chunks'])
    print("  Parse tree (pretty):")
    row['parse_tree'].pretty_print()   # shows structure in console
    print("-"*60)
```

Output:



```
  Parse tree (pretty):
                                      S
     _________________________________|_______________________________________
    |         |      |                NP                                      NP
    |         |      |        ________|_________________           ___________|_______
jumps/VBZ over/IN ./. The/DT quick/JJ brown/NN fox/NN the/DT lazy/JJ dog/NN

------------------------------------------------------------
Sentence 2: A large red apple fell from the old tree.
  Extracted NP chunks: ['A large red apple', 'the old tree']
  Parse tree (pretty):
                                  S
     _____________________________|__________________________________
    |         |      |            NP                                 NP
    |         |      |     _______|_________________         ________|________
fell/VBD from/IN ./. A/DT large/JJ red/JJ apple/NN the/DT old/JJ tree/NN

------------------------------------------------------------
Sentence 3: Dr. Smith gave a talk on natural language processing.
  Extracted NP chunks: ['Dr. Smith', 'a talk', 'natural language processing']
  Parse tree (pretty):
                                         S
     ____________________________________|______________________________________________
    |         |     |             NP                   NP                                 NP
    |         |     |        _____|______         _____|_____              ______________|______________
gave/VBD on/IN ./. Dr./NNP      Smith/NNP a/DT      talk/NN natural/JJ language/NN processing/NN
------------------------------------------------------------
Sentence 4: She bought a new pair of comfortable running shoes.
  Extracted NP chunks: ['a new pair', 'comfortable running shoes']
  Parse tree (pretty):
                                 S
     ____________________________|__________________________________________
    |         |          |     |          NP                                  NP
    |         |          |     |     _____|_______              _____________|___________
She/PRP bought/VBD of/IN ./. a/DT new/JJ pair/NN comfortable/JJ running/NN shoes/NNS

------------------------------------------------------------
Sentence 5: Recent advances in deep learning have improved NLP tasks.
  Extracted NP chunks: ['Recent advances', 'deep learning', 'NLP tasks']
  Parse tree (pretty):
                                          S
     _____________________________________|_____________________________________________________________
    |       |           |          |               NP                         NP                          NP
    |       |           |          |        _______|_______             ______|_______               ______|______
in/IN have/VBP improved/VBN ./. Recent/JJ      advances/NNS deep/JJ      learning/NN NLP/NNP      tasks/NNS

------------------------------------------------------------
Sentence 6: The cute little kitten chased the small red ball.
  Extracted NP chunks: ['The cute little kitten', 'the small red ball']
  Parse tree (pretty):
                                      S
     _________________________________|______________________________
    |         |             NP                                      NP
    |         |      _______|____________________           ________|________________
chased/VBD ./. The/DT cute/JJ little/JJ kitten/NNS the/DT small/JJ red/JJ ball/NN

------------------------------------------------------------
```

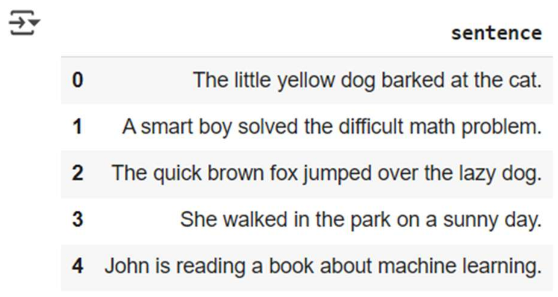
Program 10

AIM: Building Chunker.

Code:

```
import nltk
import pandas as pd

nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('conll2000')  # For trainable chunker

sentences = [
    "The little yellow dog barked at the cat.",
    "A smart boy solved the difficult math problem.",
    "The quick brown fox jumped over the lazy dog.",
    "She walked in the park on a sunny day.",
    "John is reading a book about machine learning."
]

df = pd.DataFrame({'sentence': sentences})
df
```

Output:

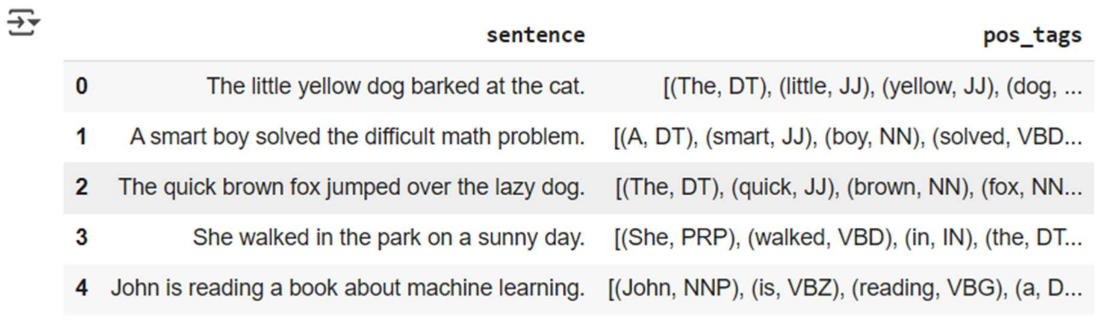


	sentence
0	The little yellow dog barked at the cat.
1	A smart boy solved the difficult math problem.
2	The quick brown fox jumped over the lazy dog.
3	She walked in the park on a sunny day.
4	John is reading a book about machine learning.

Code:

```
def tokenize_and_tag(sent):
    tokens = nltk.word_tokenize(sent)
    pos_tags = nltk.pos_tag(tokens)
    return pos_tags

df['pos_tags'] = df['sentence'].apply(tokenize_and_tag)
df[['sentence', 'pos_tags']]
```

Output:

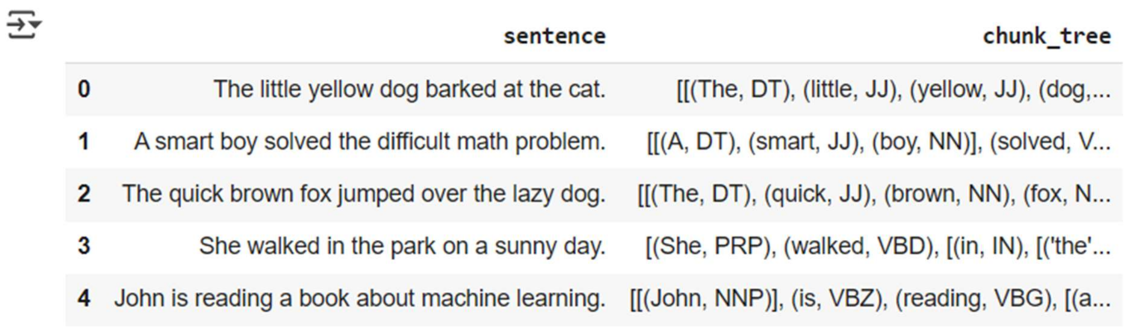


	sentence	pos_tags
0	The little yellow dog barked at the cat.	[(The, DT), (little, JJ), (yellow, JJ), (dog, ...
1	A smart boy solved the difficult math problem.	[(A, DT), (smart, JJ), (boy, NN), (solved, VBD...
2	The quick brown fox jumped over the lazy dog.	[(The, DT), (quick, JJ), (brown, NN), (fox, NN...
3	She walked in the park on a sunny day.	[(She, PRP), (walked, VBD), (in, IN), (the, DT...
4	John is reading a book about machine learning.	[(John, NNP), (is, VBZ), (reading, VBG), (a, D...

Code:

```
grammar = r"""
            NP: {<DT|PRP\$>?<JJ.*>*<NN.*>+}         # Noun phrase
            PP: {<IN><NP>}                          # Prepositional phrase: IN + NP
            VP: {<VB.*><NP|PP|CLAUSE>+$}            # Verb phrase
            CLAUSE: {<NP><VP>}                      # Clause
          """

chunk_parser = nltk.RegexpParser(grammar)

def chunk_sentence(pos_tags):
    tree = chunk_parser.parse(pos_tags)
    return tree

df['chunk_tree'] = df['pos_tags'].apply(chunk_sentence)
df[['sentence', 'chunk_tree']]
```

Output:

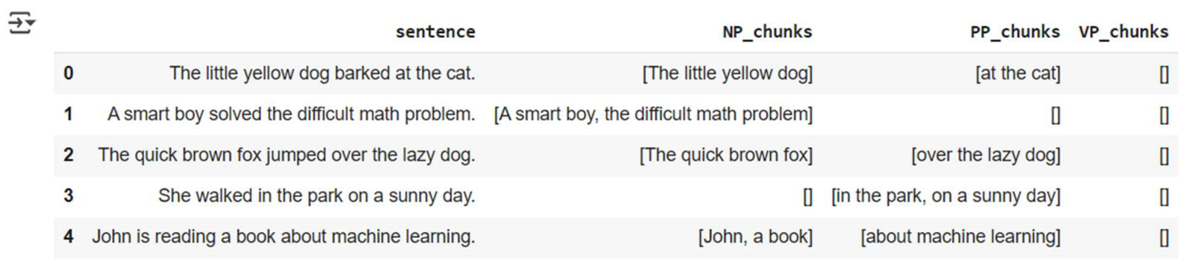


```
                                         sentence                                                chunk_tree
0             The little yellow dog barked at the cat.    [[(The, DT), (little, JJ), (yellow, JJ), (dog,...
1      A smart boy solved the difficult math problem.   [[(A, DT), (smart, JJ), (boy, NN)], (solved, V...
2       The quick brown fox jumped over the lazy dog.  [[(The, DT), (quick, JJ), (brown, NN), (fox, N...
3                 She walked in the park on a sunny day.   [(She, PRP), (walked, VBD), [(in, IN), [('the'...
4   John is reading a book about machine learning.  [[(John, NNP)], (is, VBZ), (reading, VBG), [(a...
```

Code:

```
def extract_chunks(tree, chunk_type):
    chunks = []
    for subtree in tree:
        if isinstance(subtree, nltk.Tree) and subtree.label() == chunk_type:
            words = [word for word, tag in subtree.leaves()]
            chunks.append(" ".join(words))
    return chunks

df['NP_chunks'] = df['chunk_tree'].apply(lambda t: extract_chunks(t, 'NP'))
df['PP_chunks'] = df['chunk_tree'].apply(lambda t: extract_chunks(t, 'PP'))
df['VP_chunks'] = df['chunk_tree'].apply(lambda t: extract_chunks(t, 'VP'))

df[['sentence','NP_chunks','PP_chunks','VP_chunks']]
```

Output:

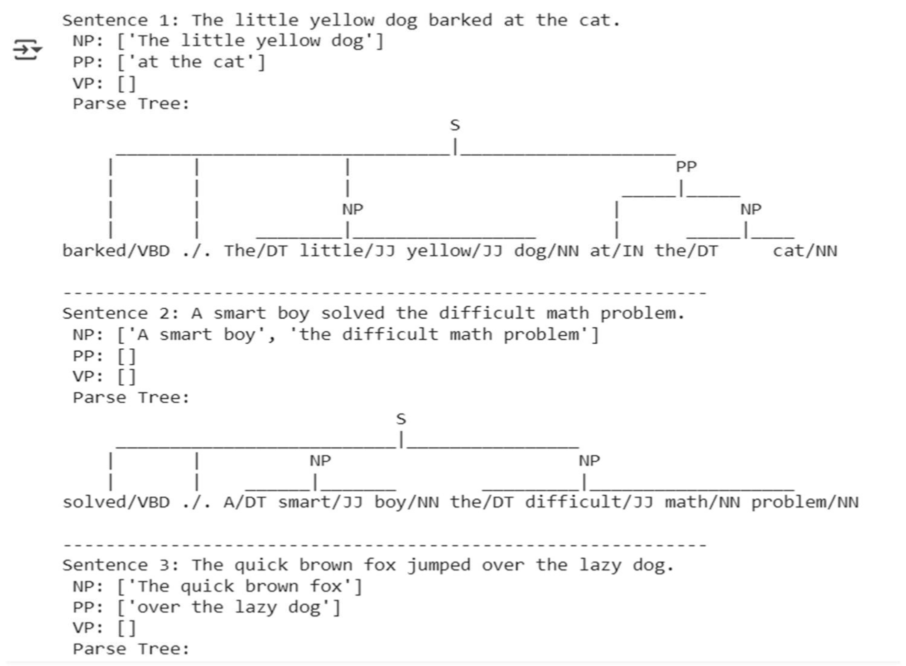


```
                                         sentence                                NP_chunks                   PP_chunks  VP_chunks
0             The little yellow dog barked at the cat.                    [The little yellow dog]                [at the cat]         []
1      A smart boy solved the difficult math problem.  [A smart boy, the difficult math problem]                          []         []
2       The quick brown fox jumped over the lazy dog.                      [The quick brown fox]           [over the lazy dog]         []
3                 She walked in the park on a sunny day.                                       []  [in the park, on a sunny day]         []
4   John is reading a book about machine learning.                             [John, a book]     [about machine learning]         []
```

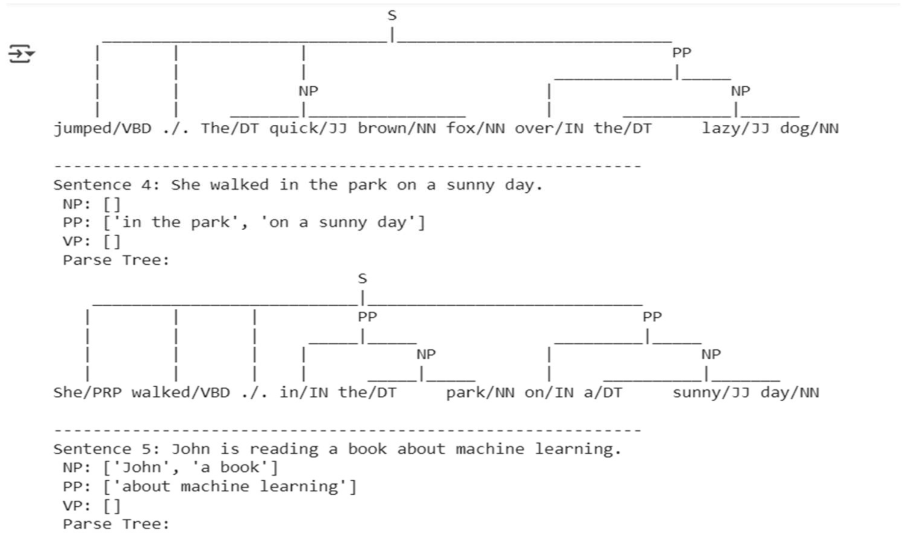
Code:

```
for idx, row in df.iterrows():
    print(f"Sentence {idx+1}: {row['sentence']}")
    print(" NP:", row['NP_chunks'])
    print(" PP:", row['PP_chunks'])
    print(" VP:", row['VP_chunks'])
    print(" Parse Tree:")
    row['chunk_tree'].pretty_print()
    print("-"*60)
```

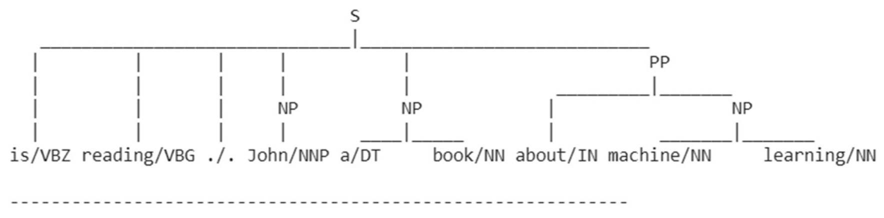
Output:



```
Sentence 1: The little yellow dog barked at the cat.
 NP: ['The little yellow dog']
 PP: ['at the cat']
 VP: []
 Parse Tree:
                                    S
     _______________________________|_____________________
    |       |             |                               PP
    |       |             |                          _____|_____
    |       |             NP                        |           NP
    |       |     ________|_________________        |      _____|____
barked/VBD ./. The/DT little/JJ yellow/JJ dog/NN at/IN the/DT      cat/NN

------------------------------------------------------------
Sentence 2: A smart boy solved the difficult math problem.
 NP: ['A smart boy', 'the difficult math problem']
 PP: []
 VP: []
 Parse Tree:
                               S
     __________________________|________________
    |       |          NP                       NP
    |       |     _____|_______       __________|___________________
solved/VBD ./. A/DT smart/JJ boy/NN the/DT difficult/JJ math/NN problem/NN

------------------------------------------------------------
Sentence 3: The quick brown fox jumped over the lazy dog.
 NP: ['The quick brown fox']
 PP: ['over the lazy dog']
 VP: []
 Parse Tree:
```



```
                                  S
     _____________________________|____________________________
    |       |            |                                     PP
    |       |            |                         ____________|_____
    |       |            NP                       |                  NP
    |       |     _______|________________        |          ________|______
jumped/VBD ./. The/DT quick/JJ brown/NN fox/NN over/IN the/DT     lazy/JJ dog/NN

------------------------------------------------------------
Sentence 4: She walked in the park on a sunny day.
 NP: []
 PP: ['in the park', 'on a sunny day']
 VP: []
 Parse Tree:
                               S
    ___________________________|_____________________________
   |        |       |          PP                            PP
   |        |       |     _____|_____                ________|_____
   |        |       |    |           NP             |              NP
   |        |       |    |      _____|_____         |      ________|_______
She/PRP walked/VBD ./. in/IN the/DT      park/NN on/IN a/DT      sunny/JJ day/NN

------------------------------------------------------------
Sentence 5: John is reading a book about machine learning.
 NP: ['John', 'a book']
 PP: ['about machine learning']
 VP: []
 Parse Tree:
```

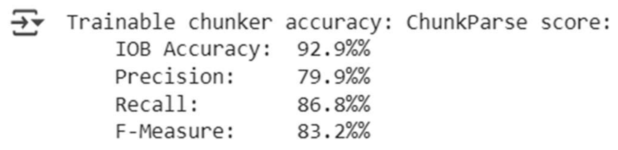


```
                                  S
   _______________________________|__________________________
  |          |        |      |           |                          PP
  |          |        |      |           |                 _________|_______
  |          |        |      NP          NP               |                NP
  |          |        |      |       ____|_____           |         _______|_______
is/VBZ reading/VBG ./. John/NNP a/DT     book/NN about/IN machine/NN     learning/NN

------------------------------------------------------------
```

Code:

```
from nltk.corpus import conll2000
from nltk.chunk import ChunkParserI
from nltk.tag import UnigramTagger

class NgramChunker(ChunkParserI):
    def __init__(self, train_sents):
        train_data = [[(t, c) for w, t, c in nltk.chunk.tree2conlltags(sent)]
                      for sent in train_sents]
        self.tagger = UnigramTagger(train_data)

    def parse(self, sentence):
        pos_tags = [pos for (word, pos) in sentence]
        tagged_pos = self.tagger.tag(pos_tags)
        conlltags = [(word, pos, chunk_tag if chunk_tag else 'O')
                     for ((word, pos), (pos2, chunk_tag)) in zip(sentence,
tagged_pos)]
        return nltk.chunk.conlltags2tree(conlltags)

# Training and testing
train_sents = conll2000.chunked_sents('train.txt', chunk_types=['NP'])
test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])

chunker = NgramChunker(train_sents)
print("Trainable chunker accuracy:", chunker.evaluate(test_sents))
```

Output:

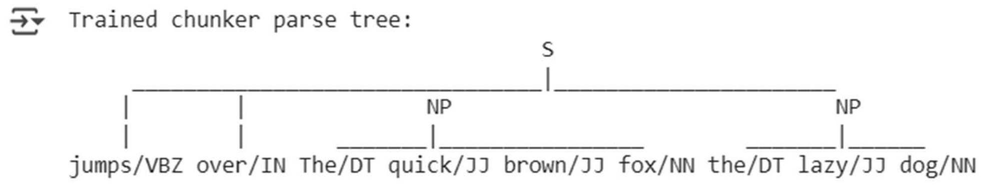


```
Trainable chunker accuracy: ChunkParse score:
    IOB Accuracy:  92.9%%
    Precision:     79.9%%
    Recall:        86.8%%
    F-Measure:     83.2%%
```

Code:

```
test_sentence = [("The","DT"), ("quick","JJ"), ("brown","JJ"), ("fox","NN"),
("jumps","VBZ"), ("over","IN"), ("the","DT"), ("lazy","JJ"), ("dog","NN")]
trained_tree = chunker.parse(test_sentence)

print("Trained chunker parse tree:")
trained_tree.pretty_print()
```

Output:



```
Trained chunker parse tree:
                                      S
     _________________________________|_______________________
    |        |              NP                               NP
    |        |       _______|_________________        _______|______
jumps/VBZ over/IN The/DT quick/JJ brown/JJ fox/NN the/DT lazy/JJ dog/NN
```

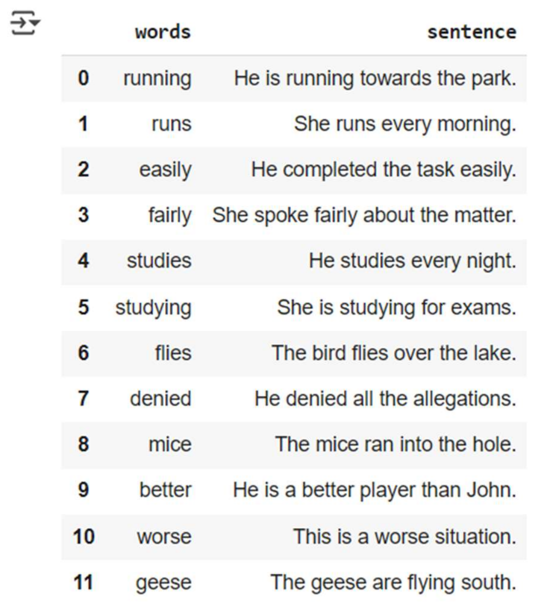
Program 11

AIM: Stemming And Lemmatization.

Code:

```
import pandas as pd
import nltk

nltk.download('punkt_tab')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger_eng')

data = {
    'words': [
        "running", "runs", "easily", "fairly",
        "studies", "studying", "flies", "denied",
        "mice", "better", "worse", "geese"
    ],
    'sentence': [
        "He is running towards the park.",
        "She runs every morning.",
        "He completed the task easily.",
        "She spoke fairly about the matter.",
        "He studies every night.",
        "She is studying for exams.",
        "The bird flies over the lake.",
        "He denied all the allegations.",
        "The mice ran into the hole.",
        "He is a better player than John.",
        "This is a worse situation.",
        "The geese are flying south."
    ]
}

df = pd.DataFrame(data)
df
```

Output:

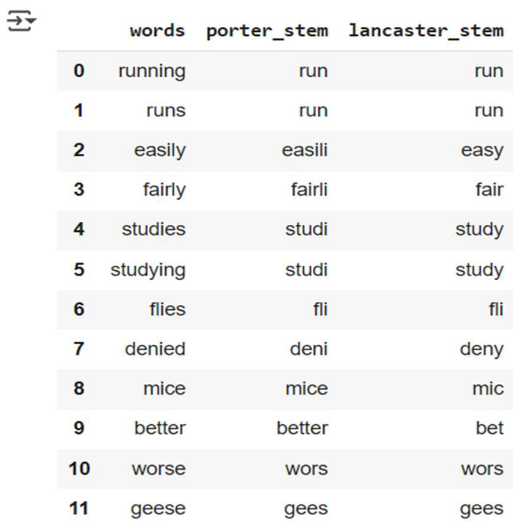


	words	sentence
0	running	He is running towards the park.
1	runs	She runs every morning.
2	easily	He completed the task easily.
3	fairly	She spoke fairly about the matter.
4	studies	He studies every night.
5	studying	She is studying for exams.
6	flies	The bird flies over the lake.
7	denied	He denied all the allegations.
8	mice	The mice ran into the hole.
9	better	He is a better player than John.
10	worse	This is a worse situation.
11	geese	The geese are flying south.

Code:

```
# Stemming (Porter and Lancaster)
from nltk.stem import PorterStemmer, LancasterStemmer

porter = PorterStemmer()
lancaster = LancasterStemmer()

def stem_word(word):
    porter_result = porter.stem(word)
    lancaster_result = lancaster.stem(word)
    return porter_result, lancaster_result

df[['porter_stem', 'lancaster_stem']] = df['words'].apply(lambda w:
pd.Series(stem_word(w)))
df[['words', 'porter_stem', 'lancaster_stem']]
```

Output:

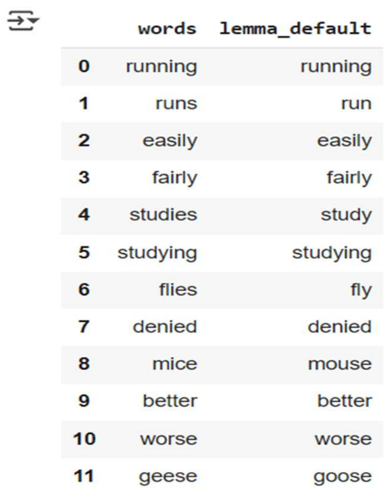


	words	porter_stem	lancaster_stem
0	running	run	run
1	runs	run	run
2	easily	easili	easy
3	fairly	fairli	fair
4	studies	studi	study
5	studying	studi	study
6	flies	fli	fli
7	denied	deni	deny
8	mice	mice	mic
9	better	better	bet
10	worse	wors	wors
11	geese	gees	gees

Code:

```
# Lemmatization (without POS), defaults to noun
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

df['lemma_default'] = df['words'].apply(lambda w: lemmatizer.lemmatize(w))
df[['words', 'lemma_default']]
```

Output:

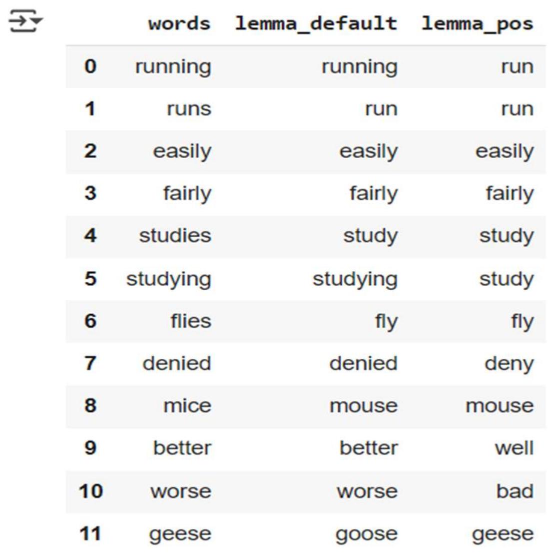


	words	lemma_default
0	running	running
1	runs	run
2	easily	easily
3	fairly	fairly
4	studies	study
5	studying	studying
6	flies	fly
7	denied	denied
8	mice	mouse
9	better	better
10	worse	worse
11	geese	goose

Code:

```
# Lemmatization (with POS tagging)
from nltk.corpus import wordnet

def nltk_pos_to_wordnet(nltk_pos):
    if nltk_pos.startswith('J'):
        return wordnet.ADJ
    elif nltk_pos.startswith('V'):
        return wordnet.VERB
    elif nltk_pos.startswith('N'):
        return wordnet.NOUN
    elif nltk_pos.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

def lemmatize_with_pos(word):
    nltk_pos = nltk.pos_tag([word])[0][1]
    wn_pos = nltk_pos_to_wordnet(nltk_pos)
    return lemmatizer.lemmatize(word, wn_pos)

df['lemma_pos'] = df['words'].apply(lemmatize_with_pos)
df[['words', 'lemma_default', 'lemma_pos']]
```

Output:

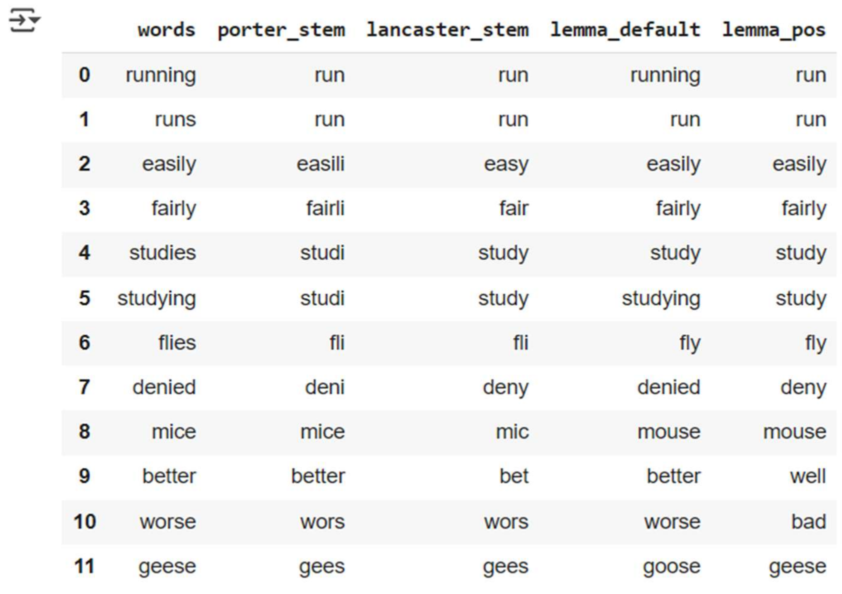


```
        words  lemma_default  lemma_pos
0     running        running        run
1        runs            run        run
2      easily         easily     easily
3      fairly         fairly     fairly
4     studies          study      study
5    studying       studying      study
6       flies            fly        fly
7      denied         denied       deny
8        mice          mouse      mouse
9      better         better       well
10      worse          worse        bad
11      geese          goose      geese
```

Code:

```
# Compare all results
df[['words', 'porter_stem', 'lancaster_stem', 'lemma_default', 'lemma_pos']]
```

Output:

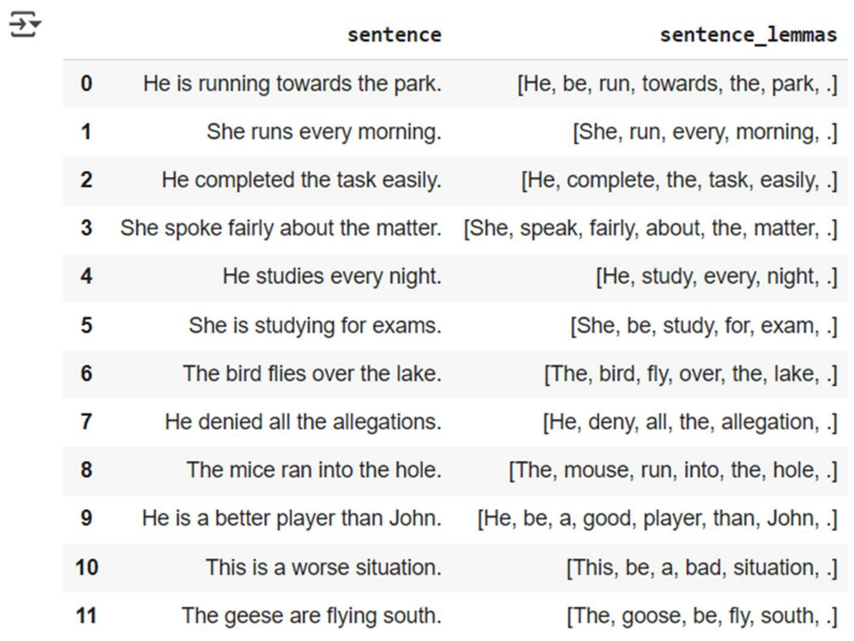


	words	porter_stem	lancaster_stem	lemma_default	lemma_pos
0	running	run	run	running	run
1	runs	run	run	run	run
2	easily	easili	easy	easily	easily
3	fairly	fairli	fair	fairly	fairly
4	studies	studi	study	study	study
5	studying	studi	study	studying	study
6	flies	fli	fli	fly	fly
7	denied	deni	deny	denied	deny
8	mice	mice	mic	mouse	mouse
9	better	better	bet	better	well
10	worse	wors	wors	worse	bad
11	geese	gees	gees	goose	geese

Code:

```
# Lemmatization in sentences
def lemmatize_sentence(sent):
    tokens = nltk.word_tokenize(sent)
    pos_tags = nltk.pos_tag(tokens)
    lemmas = []
    for word, tag in pos_tags:
        wn_pos = nltk_pos_to_wordnet(tag)
        lemma = lemmatizer.lemmatize(word, wn_pos)
        lemmas.append(lemma)
    return lemmas

df['sentence_lemmas'] = df['sentence'].apply(lemmatize_sentence)
df[['sentence', 'sentence_lemmas']]
```

Output:



	sentence	sentence_lemmas
0	He is running towards the park.	[He, be, run, towards, the, park, .]
1	She runs every morning.	[She, run, every, morning, .]
2	He completed the task easily.	[He, complete, the, task, easily, .]
3	She spoke fairly about the matter.	[She, speak, fairly, about, the, matter, .]
4	He studies every night.	[He, study, every, night, .]
5	She is studying for exams.	[She, be, study, for, exam, .]
6	The bird flies over the lake.	[The, bird, fly, over, the, lake, .]
7	He denied all the allegations.	[He, deny, all, the, allegation, .]
8	The mice ran into the hole.	[The, mouse, run, into, the, hole, .]
9	He is a better player than John.	[He, be, a, good, player, than, John, .]
10	This is a worse situation.	[This, be, a, bad, situation, .]
11	The geese are flying south.	[The, goose, be, fly, south, .]

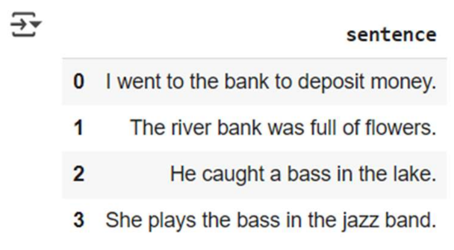
Program 12

AIM: Word Sense Disambiguation.

Code:

```
import pandas as pd
import nltk

nltk.download('punkt_tab')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('stopwords')

sentences = [
    "I went to the bank to deposit money.",
    "The river bank was full of flowers.",
    "He caught a bass in the lake.",
    "She plays the bass in the jazz band."
]

df = pd.DataFrame({'sentence': sentences})
df
```

Output:

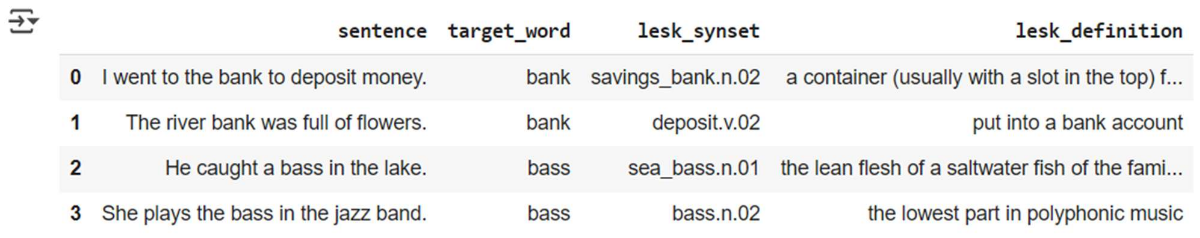


	sentence
0	I went to the bank to deposit money.
1	The river bank was full of flowers.
2	He caught a bass in the lake.
3	She plays the bass in the jazz band.

Code:

```
# Built-in Lesk algorithm
from nltk.wsd import lesk
from nltk.corpus import wordnet

def apply_lesk(sentence, target_word):
    tokens = nltk.word_tokenize(sentence)
    synset = lesk(tokens, target_word)
    if synset:
        return synset.name(), synset.definition()
    else:
        return None, None

df['target_word'] = ['bank', 'bank', 'bass', 'bass']
df[['lesk_synset', 'lesk_definition']] = df.apply(lambda row:
pd.Series(apply_lesk(row['sentence'], row['target_word'])), axis=1)
df[['sentence', 'target_word', 'lesk_synset', 'lesk_definition']]
```

Output:

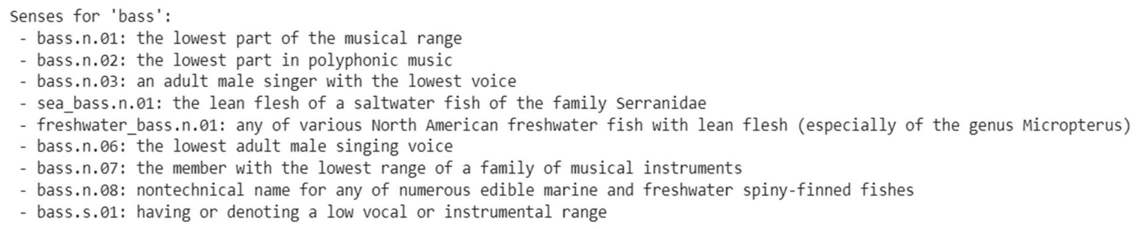


	sentence	target_word	lesk_synset	lesk_definition
0	I went to the bank to deposit money.	bank	savings_bank.n.02	a container (usually with a slot in the top) f...
1	The river bank was full of flowers.	bank	deposit.v.02	put into a bank account
2	He caught a bass in the lake.	bass	sea_bass.n.01	the lean flesh of a saltwater fish of the fami...
3	She plays the bass in the jazz band.	bass	bass.n.02	the lowest part in polyphonic music

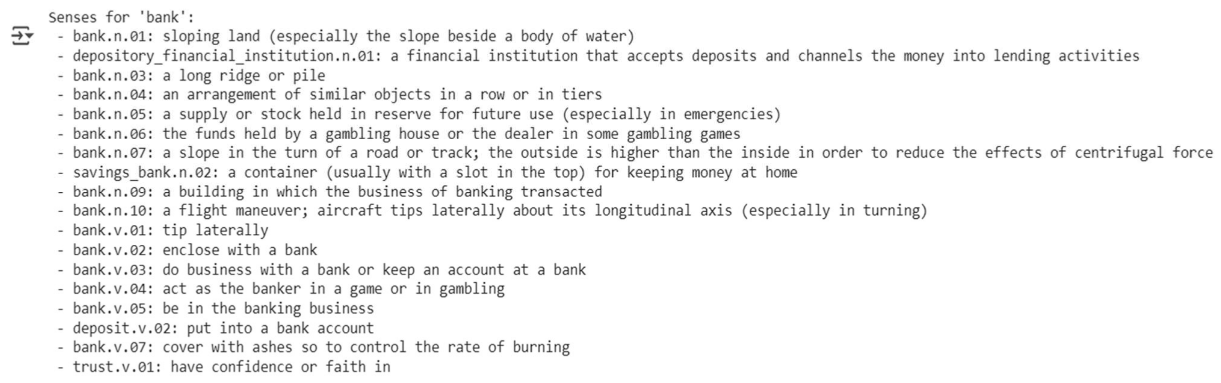
Code:

```
# Inspect available senses
for word in ['bank', 'bass']:
    print(f"\nSenses for '{word}':")
    for syn in wordnet.synsets(word):
        print(f" - {syn.name()}: {syn.definition()}")
```

Output:



```
Senses for 'bass':
 - bass.n.01: the lowest part of the musical range
 - bass.n.02: the lowest part in polyphonic music
 - bass.n.03: an adult male singer with the lowest voice
 - sea_bass.n.01: the lean flesh of a saltwater fish of the family Serranidae
 - freshwater_bass.n.01: any of various North American freshwater fish with lean flesh (especially of the genus Micropterus)
 - bass.n.06: the lowest adult male singing voice
 - bass.n.07: the member with the lowest range of a family of musical instruments
 - bass.n.08: nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes
 - bass.s.01: having or denoting a low vocal or instrumental range
```



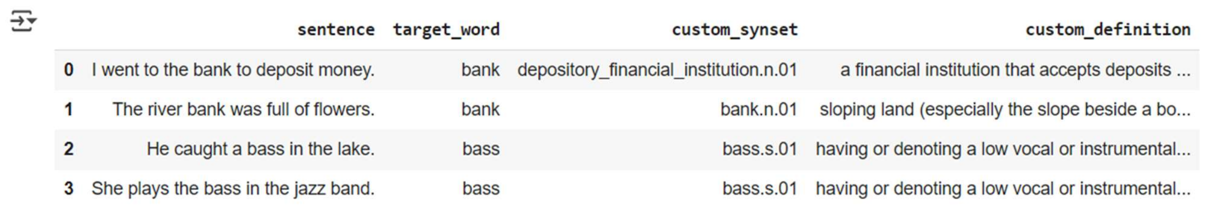
```
Senses for 'bank':
 - bank.n.01: sloping land (especially the slope beside a body of water)
 - depository_financial_institution.n.01: a financial institution that accepts deposits and channels the money into lending activities
 - bank.n.03: a long ridge or pile
 - bank.n.04: an arrangement of similar objects in a row or in tiers
 - bank.n.05: a supply or stock held in reserve for future use (especially in emergencies)
 - bank.n.06: the funds held by a gambling house or the dealer in some gambling games
 - bank.n.07: a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force
 - savings_bank.n.02: a container (usually with a slot in the top) for keeping money at home
 - bank.n.09: a building in which the business of banking transacted
 - bank.n.10: a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)
 - bank.v.01: tip laterally
 - bank.v.02: enclose with a bank
 - bank.v.03: do business with a bank or keep an account at a bank
 - bank.v.04: act as the banker in a game or in gambling
 - bank.v.05: be in the banking business
 - deposit.v.02: put into a bank account
 - bank.v.07: cover with ashes so to control the rate of burning
 - trust.v.01: have confidence or faith in
```

Code:

```
# Custom (simplified) Lesk algorithm
from nltk.corpus import stopwords
import string

stop_words = set(stopwords.words('english'))
punct = set(string.punctuation)

def custom_lesk(word, sentence):
    tokens = nltk.word_tokenize(sentence)
    tokens = [t.lower() for t in tokens if t.lower() not in stop_words and t not in
punct]

    best_sense = None
    max_overlap = 0

    for sense in wordnet.synsets(word):
        signature = set(nltk.word_tokenize(sense.definition().lower()))
        # add example sentences for more context
        for example in sense.examples():
            signature.update(nltk.word_tokenize(example.lower()))

        signature = [t for t in signature if t not in stop_words and t not in punct]

        overlap = len(set(tokens) & set(signature))

        if overlap > max_overlap:
            max_overlap = overlap
            best_sense = sense

    return best_sense.name(), best_sense.definition() if best_sense else (None,
None)

df[['custom_synset', 'custom_definition']] = df.apply(lambda row:
pd.Series(custom_lesk(row['target_word'], row['sentence'])), axis=1)
```

```
df[['sentence', 'target_word', 'custom_synset', 'custom_definition']]
```

Output:

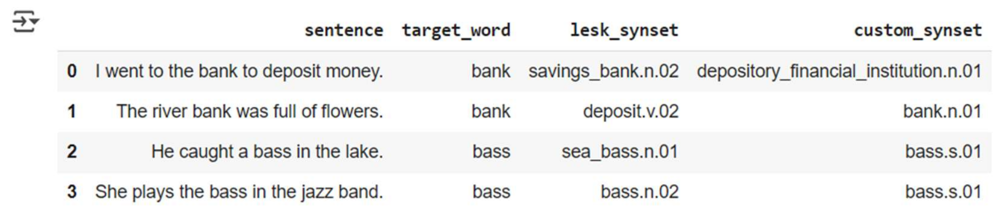


	sentence	target_word	custom_synset	custom_definition
0	I went to the bank to deposit money.	bank	depository_financial_institution.n.01	a financial institution that accepts deposits ...
1	The river bank was full of flowers.	bank	bank.n.01	sloping land (especially the slope beside a bo...
2	He caught a bass in the lake.	bass	bass.s.01	having or denoting a low vocal or instrumental...
3	She plays the bass in the jazz band.	bass	bass.s.01	having or denoting a low vocal or instrumental...

Code:

```
# Compare built-in and custom Lesk
df[['sentence', 'target_word', 'lesk_synset', 'custom_synset']]
```

Output:



	sentence	target_word	lesk_synset	custom_synset
0	I went to the bank to deposit money.	bank	savings_bank.n.02	depository_financial_institution.n.01
1	The river bank was full of flowers.	bank	deposit.v.02	bank.n.01
2	He caught a bass in the lake.	bass	sea_bass.n.01	bass.s.01
3	She plays the bass in the jazz band.	bass	bass.n.02	bass.s.01

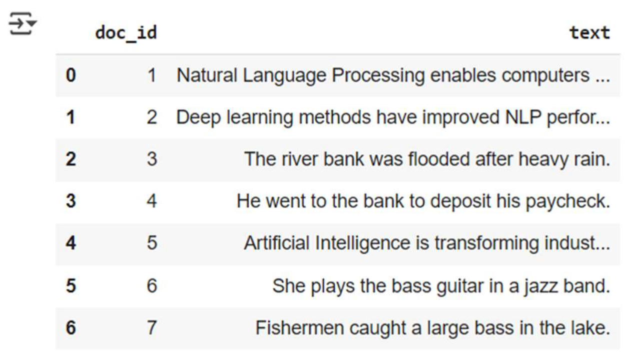
Program 13

AIM: Information Retrieval.

Code:

```
import pandas as pd
import nltk
import numpy as np

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

documents = [
    "Natural Language Processing enables computers to understand human language.",
    "Deep learning methods have improved NLP performance in recent years.",
    "The river bank was flooded after heavy rain.",
    "He went to the bank to deposit his paycheck.",
    "Artificial Intelligence is transforming industries worldwide.",
    "She plays the bass guitar in a jazz band.",
    "Fishermen caught a large bass in the lake."
]

df = pd.DataFrame({'doc_id': range(1, len(documents)+1), 'text': documents})
df
```

Output:

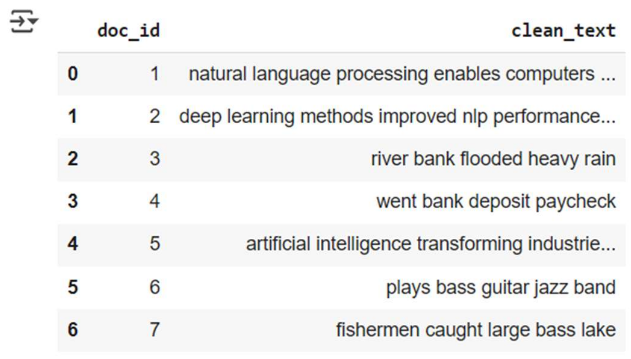


	doc_id	text
0	1	Natural Language Processing enables computers ...
1	2	Deep learning methods have improved NLP perfor...
2	3	The river bank was flooded after heavy rain.
3	4	He went to the bank to deposit his paycheck.
4	5	Artificial Intelligence is transforming indust...
5	6	She plays the bass guitar in a jazz band.
6	7	Fishermen caught a large bass in the lake.

Code:

```
# Text preprocessing
from nltk.corpus import stopwords
import string

stop_words = set(stopwords.words('english'))
punct = set(string.punctuation)

def preprocess(text):
    tokens = nltk.word_tokenize(text.lower())
    tokens = [t for t in tokens if t not in stop_words and t not in punct]
    return " ".join(tokens)

df['clean_text'] = df['text'].apply(preprocess)
df[['doc_id', 'clean_text']]
```

Output:

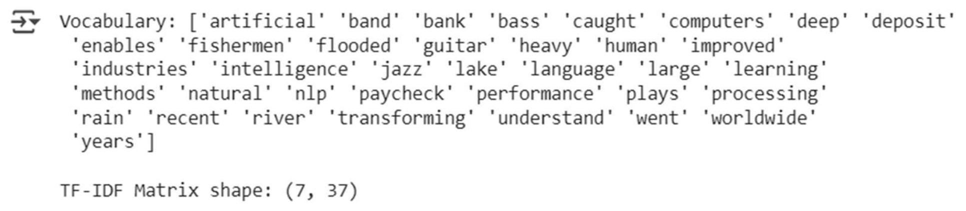


```
   doc_id                                  clean_text
0       1   natural language processing enables computers ...
1       2    deep learning methods improved nlp performance...
2       3                       river bank flooded heavy rain
3       4                         went bank deposit paycheck
4       5        artificial intelligence transforming industrie...
5       6                          plays bass guitar jazz band
6       7                    fishermen caught large bass lake
```

Code:

```
# Convert to TF-IDF matrix
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df['clean_text'])

print("Vocabulary:", vectorizer.get_feature_names_out())
print("\nTF-IDF Matrix shape:", tfidf_matrix.shape)
```

Output:

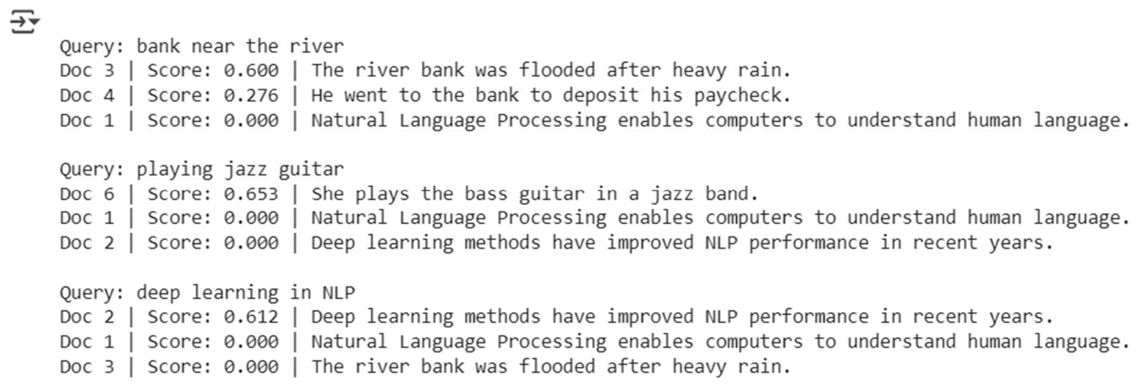


```
Vocabulary: ['artificial' 'band' 'bank' 'bass' 'caught' 'computers' 'deep' 'deposit'
 'enables' 'fishermen' 'flooded' 'guitar' 'heavy' 'human' 'improved'
 'industries' 'intelligence' 'jazz' 'lake' 'language' 'large' 'learning'
 'methods' 'natural' 'nlp' 'paycheck' 'performance' 'plays' 'processing'
 'rain' 'recent' 'river' 'transforming' 'understand' 'went' 'worldwide'
 'years']

TF-IDF Matrix shape: (7, 37)
```

Code:

```
# Search function (cosine similarity)
from numpy.linalg import norm

def search(query, top_k=3):
    # Preprocess query
    clean_q = preprocess(query)

    # Transform query to TF-IDF vector
    query_vec = vectorizer.transform([clean_q]).toarray()[0]

    # Compute cosine similarity with each doc
    similarities = []
    for idx, doc_vec in enumerate(tfidf_matrix.toarray()):
        cos_sim = np.dot(query_vec, doc_vec) / (norm(query_vec) * norm(doc_vec))
        similarities.append((df.loc[idx, 'doc_id'], df.loc[idx, 'text'], cos_sim))

    # Sort by similarity
    similarities = sorted(similarities, key=lambda x: x[2], reverse=True)
    return similarities[:top_k]

# Test search
query1 = "bank near the river"
query2 = "playing jazz guitar"
query3 = "deep learning in NLP"

print("\nQuery:", query1)
for doc_id, text, score in search(query1):
    print(f"Doc {doc_id} | Score: {score:.3f} | {text}")
print("\nQuery:", query2)
for doc_id, text, score in search(query2):
    print(f"Doc {doc_id} | Score: {score:.3f} | {text}")

print("\nQuery:", query3)
for doc_id, text, score in search(query3):
    print(f"Doc {doc_id} | Score: {score:.3f} | {text}")
```

Output:



```
Query: bank near the river
Doc 3 | Score: 0.600 | The river bank was flooded after heavy rain.
Doc 4 | Score: 0.276 | He went to the bank to deposit his paycheck.
Doc 1 | Score: 0.000 | Natural Language Processing enables computers to understand human language.

Query: playing jazz guitar
Doc 6 | Score: 0.653 | She plays the bass guitar in a jazz band.
Doc 1 | Score: 0.000 | Natural Language Processing enables computers to understand human language.
Doc 2 | Score: 0.000 | Deep learning methods have improved NLP performance in recent years.

Query: deep learning in NLP
Doc 2 | Score: 0.612 | Deep learning methods have improved NLP performance in recent years.
Doc 1 | Score: 0.000 | Natural Language Processing enables computers to understand human language.
Doc 3 | Score: 0.000 | The river bank was flooded after heavy rain.
```

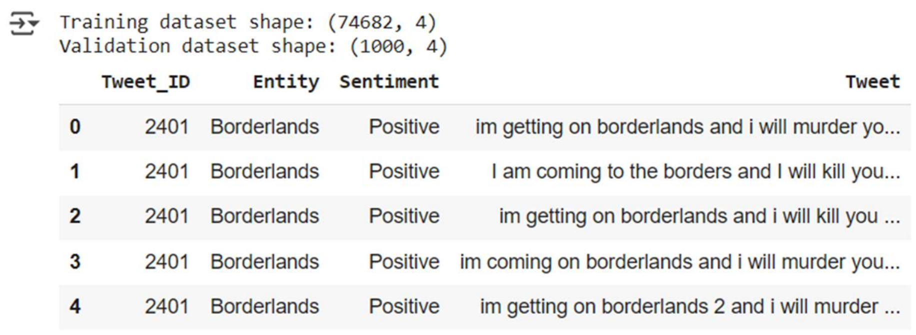
Program 14

AIM: CASE STUDY : Application of NLP- Sentiment Analysis of tweets in Twitter platform.

Code:

```
import pandas as pd
import numpy as np
import re
import string
import matplotlib.pyplot as plt
import seaborn as sns

# Sklearn / NLP
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Load Dataset
train_df = pd.read_csv("/content/twitter_training.csv", header=None,
names=["Tweet_ID", "Entity", "Sentiment", "Tweet"])
valid_df = pd.read_csv("/content/twitter_validation.csv", header=None,
names=["Tweet_ID", "Entity", "Sentiment", "Tweet"])

print("Training dataset shape:", train_df.shape)
print("Validation dataset shape:", valid_df.shape)
train_df.head()
```

Output:

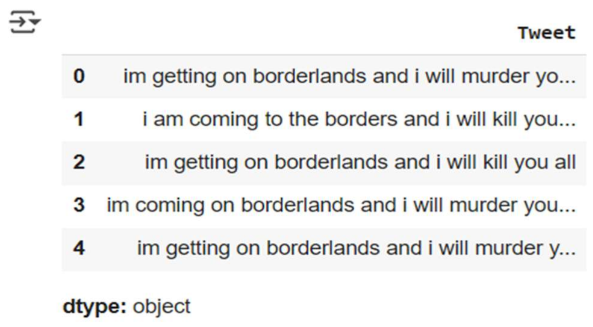


```
Training dataset shape: (74682, 4)
Validation dataset shape: (1000, 4)
   Tweet_ID       Entity  Sentiment                                              Tweet
0      2401  Borderlands   Positive  im getting on borderlands and i will murder yo...
1      2401  Borderlands   Positive    I am coming to the borders and I will kill you...
2      2401  Borderlands   Positive     im getting on borderlands and i will kill you ...
3      2401  Borderlands   Positive  im coming on borderlands and i will murder you...
4      2401  Borderlands   Positive    im getting on borderlands 2 and i will murder ...
```

Code:

```
# Preprocess Data
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r"http\S+", "", text)           # remove URLs
    text = re.sub(r"@\w+", "", text)             # remove mentions
    text = re.sub(r"#\w+", "", text)             # remove hashtags
    text = re.sub(r"[0-9]+", "", text)           # remove numbers
    text = text.translate(str.maketrans("", "", string.punctuation)) # remove
punctuation
    text = text.strip()
    return text

train_df["Tweet"] = train_df["Tweet"].apply(clean_text)
valid_df["Tweet"] = valid_df["Tweet"].apply(clean_text)

train_df["Tweet"].head()
```

Output:

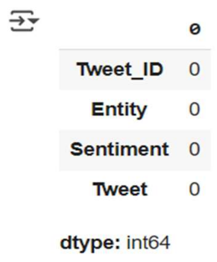


```
                                                Tweet
0    im getting on borderlands and i will murder yo...
1        i am coming to the borders and i will kill you...
2         im getting on borderlands and i will kill you all
3  im coming on borderlands and i will murder you...
4        im getting on borderlands and i will murder y...

dtype: object
```

Code:

```
train_df.isnull().sum()
```

Output:

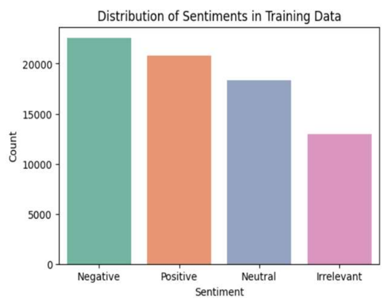


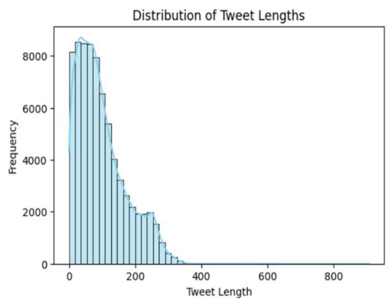
```
           0
Tweet_ID   0
Entity     0
Sentiment  0
Tweet      0

dtype: int64
```

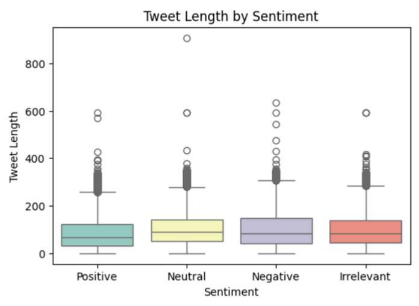
Code:

```
# Exploratory Data Analysis (EDA)
# Sentiment distribution in training data
plt.figure(figsize=(6,4))
sns.countplot(x="Sentiment", data=train_df,
order=train_df["Sentiment"].value_counts().index, palette="Set2")
plt.title("Distribution of Sentiments in Training Data")
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.show()

# Length of tweets
train_df["tweet_length"] = train_df["Tweet"].apply(len)
plt.figure(figsize=(6,4))
sns.histplot(train_df["tweet_length"], bins=50, color="skyblue", kde=True)
plt.title("Distribution of Tweet Lengths")
plt.xlabel("Tweet Length")
plt.ylabel("Frequency")
plt.show()

# Average tweet length by sentiment
plt.figure(figsize=(6,4))
sns.boxplot(x="Sentiment", y="tweet_length", data=train_df, palette="Set3")
plt.title("Tweet Length by Sentiment")
plt.xlabel("Sentiment")
plt.ylabel("Tweet Length")
plt.show()
```

Output:





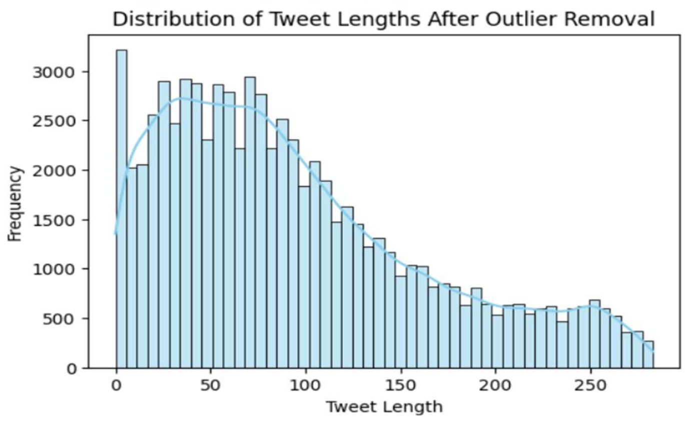


Code:

```
# Remove Outliers using IQR
Q1 = train_df["tweet_length"].quantile(0.25)
Q3 = train_df["tweet_length"].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
train_df_cleaned = train_df[(train_df["tweet_length"] >= lower_bound) &
(train_df["tweet_length"] <= upper_bound)].copy()

print("Original training dataset shape:", train_df.shape)
print("Cleaned training dataset shape:", train_df_cleaned.shape)

# Visualize the distribution of tweet lengths after removing outliers
plt.figure(figsize=(6,4))
sns.histplot(train_df_cleaned["tweet_length"], bins=50, color="skyblue", kde=True)
plt.title("Distribution of Tweet Lengths After Outlier Removal")
plt.xlabel("Tweet Length")
plt.ylabel("Frequency")
plt.show()

# Update the training data for further steps
train_df = train_df_cleaned
```

Output:

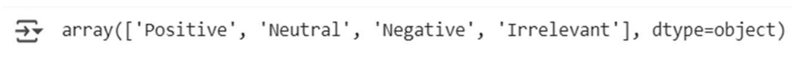
```
Original training dataset shape: (74682, 5)
Cleaned training dataset shape: (73613, 5)
```



Code:

```
# Step 3: Select Features & Labels
X_train = train_df["Tweet"]
y_train = train_df["Sentiment"]

X_valid = valid_df["Tweet"]
y_valid = valid_df["Sentiment"]

y_train.unique()
```

Output:

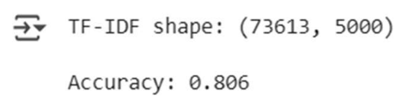


```
array(['Positive', 'Neutral', 'Negative', 'Irrelevant'], dtype=object)
```

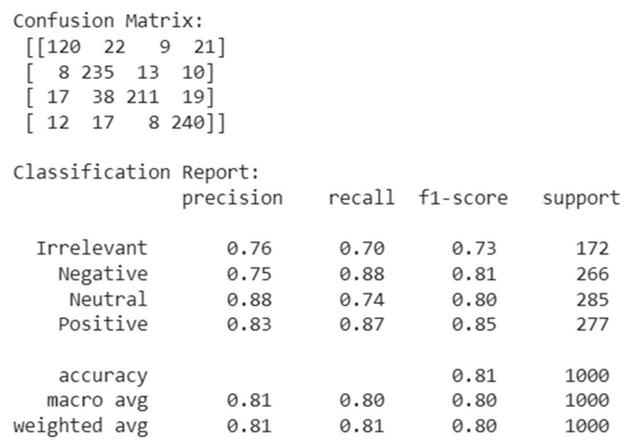
Code:

```
# Feature Extraction (TF-IDF)
vectorizer = TfidfVectorizer(max_features=5000, stop_words="english")
X_train_tfidf = vectorizer.fit_transform(X_train)
X_valid_tfidf = vectorizer.transform(X_valid)

print("TF-IDF shape:", X_train_tfidf.shape)

# Train Model (Logistic Regression)
model = LogisticRegression(max_iter=1000)
model.fit(X_train_tfidf, y_train)

# Evaluate Model
y_pred = model.predict(X_valid_tfidf)

print("\nAccuracy:", accuracy_score(y_valid, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_valid, y_pred))
print("\nClassification Report:\n", classification_report(y_valid, y_pred))
```

Output:



```
TF-IDF shape: (73613, 5000)

Accuracy: 0.806
```

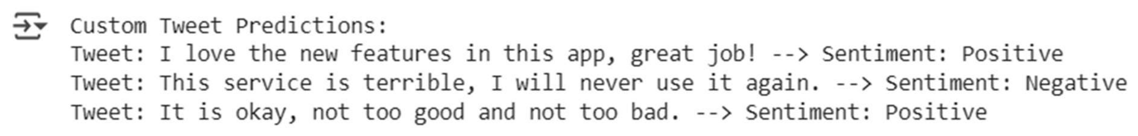


```
Confusion Matrix:
 [[120  22   9  21]
 [  8 235  13  10]
 [ 17  38 211  19]
 [ 12  17   8 240]]

Classification Report:
               precision    recall  f1-score   support

  Irrelevant       0.76      0.70      0.73       172
    Negative       0.75      0.88      0.81       266
     Neutral       0.88      0.74      0.80       285
    Positive       0.83      0.87      0.85       277

    accuracy                           0.81      1000
   macro avg       0.81      0.80      0.80      1000
weighted avg       0.81      0.81      0.80      1000
```

Code:

```
# Test on Custom Tweets
test_tweets = [
    "I love the new features in this app, great job!",
    "This service is terrible, I will never use it again.",
    "It is okay, not too good and not too bad."
]

test_tweets_clean = [clean_text(t) for t in test_tweets]
test_vec = vectorizer.transform(test_tweets_clean)
preds = model.predict(test_vec)

print("Custom Tweet Predictions:")
for tw, pr in zip(test_tweets, preds):
    print(f"Tweet: {tw} --> Sentiment: {pr}")
```

Output:



```
Custom Tweet Predictions:
Tweet: I love the new features in this app, great job! --> Sentiment: Positive
Tweet: This service is terrible, I will never use it again. --> Sentiment: Negative
Tweet: It is okay, not too good and not too bad. --> Sentiment: Positive
```