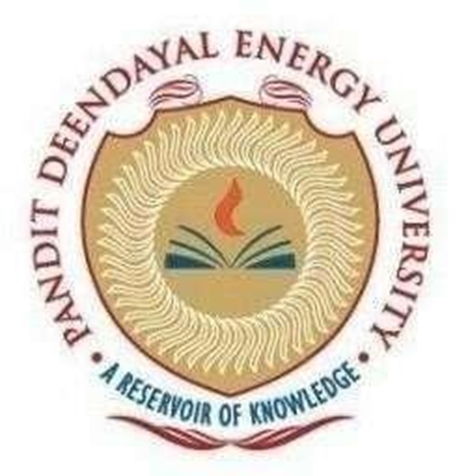
**PANDIT DEENDAYAL ENERGY UNIVERSITY**

****

**Lab Report**

**Natural Language Processing Lab**

**(21IC403P)**

**By,**

**Aashi Shah**

**Roll No: 21BIT269D**

**VII Semester**

**Submitted to,**

**Dr. Hiren Thakkar(For H1H2)**

**Information and Communication Technology (ICT) School of Technology**

**Pandit Deendayal Energy University  
School of Technology**

**Certificate**

This is to certify that **Aashi Shah**, Roll Number **21BIT269D**, has successfully completed the lab work for the course Natural Language Processing (21IC403P) as part of Semester 7, Batch H2, during the academic year 2024-2025.  
  
This lab work is in partial fulfillment of the requirements for the Bachelor of Technology (B.Tech) program in Information and Communication Technology, graduating year 2025.

Date: 9/11/2024

[Signature]  
Name: Dr. Hiren Thakkar

**Index**

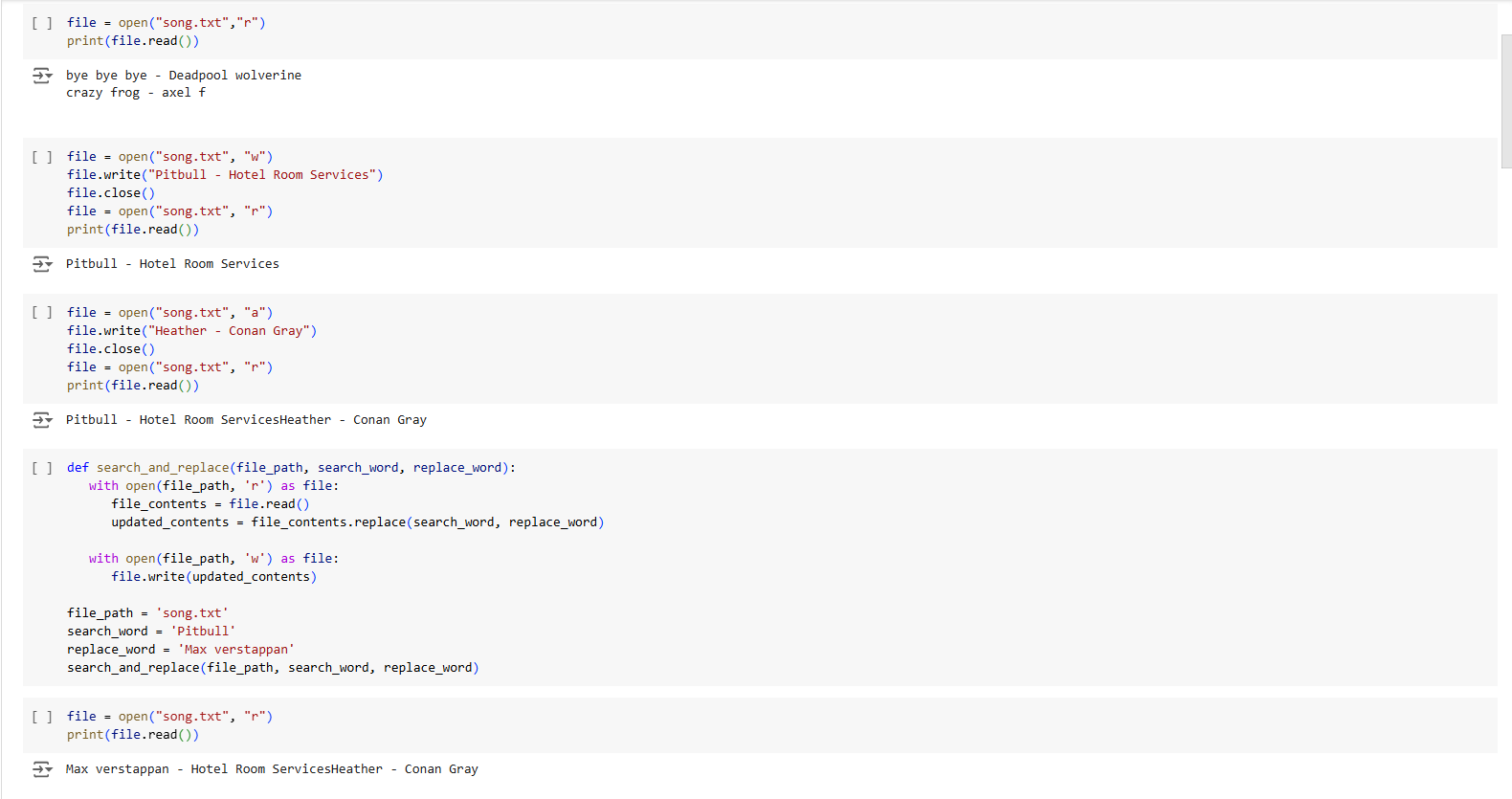
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr No.** | **Experiment Title** | **Page No.** | **Signature** |
| 1 | Write a Python script to read, write, and manipulate text files. Extract and process text from PDF files using Python libraries like PyPDF2 and Pdf Miner | 1 |  |
| 2 | Implement a Python program to search for patterns in text utilizing regular expressions. | 4 |  |
| 3 | Create a Python script to implement and compare ultra-fast tokenization techniques using libraries like NLTK, SpaCy, and FastText. | 6 |  |
| 4 | Develop a Python program to apply stemming and lemmatization to normalize text data. | 7 |  |
| 5 | Write a Python script to explore and implement various vocabulary matching techniques, including word embeddings and similarity measures. | 9 |  |
| 6 | Create a Python program to automatically tag parts of speech in raw text files using libraries like NLTK and SpaCy. | 11 |  |
| 7 | Implement a Python script to visualize POS tagging and NER using tools like SpaCy and Matplotlib. | 13 |  |
| 8 | Develop a Python program to implement text classification algorithms such as Naive Bayes, SVM, and neural networks, and evaluate their performance. | 14 |  |
| 9 | Write a Python script to apply Non-negative Matrix Factorization (NMF) for topic modeling and document clustering. | 17 |  |
| 10 | Create a Python program to implement and compare various NLP algorithms for tasks such as classification, clustering, and sentiment analysis. | 18 |  |
| 11 | Develop a Python script to perform sentiment analysis on text data using lexicon-based methods and machine learning models. | 19 |  |
| 12 | Write a Python program to apply deep learning models such as RNNs, LSTMs, or Transformers for NLP tasks, including experimenting with pre-trained models like BERT or GPT. | 20 |  |

**Experiment 1: Text and PDF File Manipulation with Python**

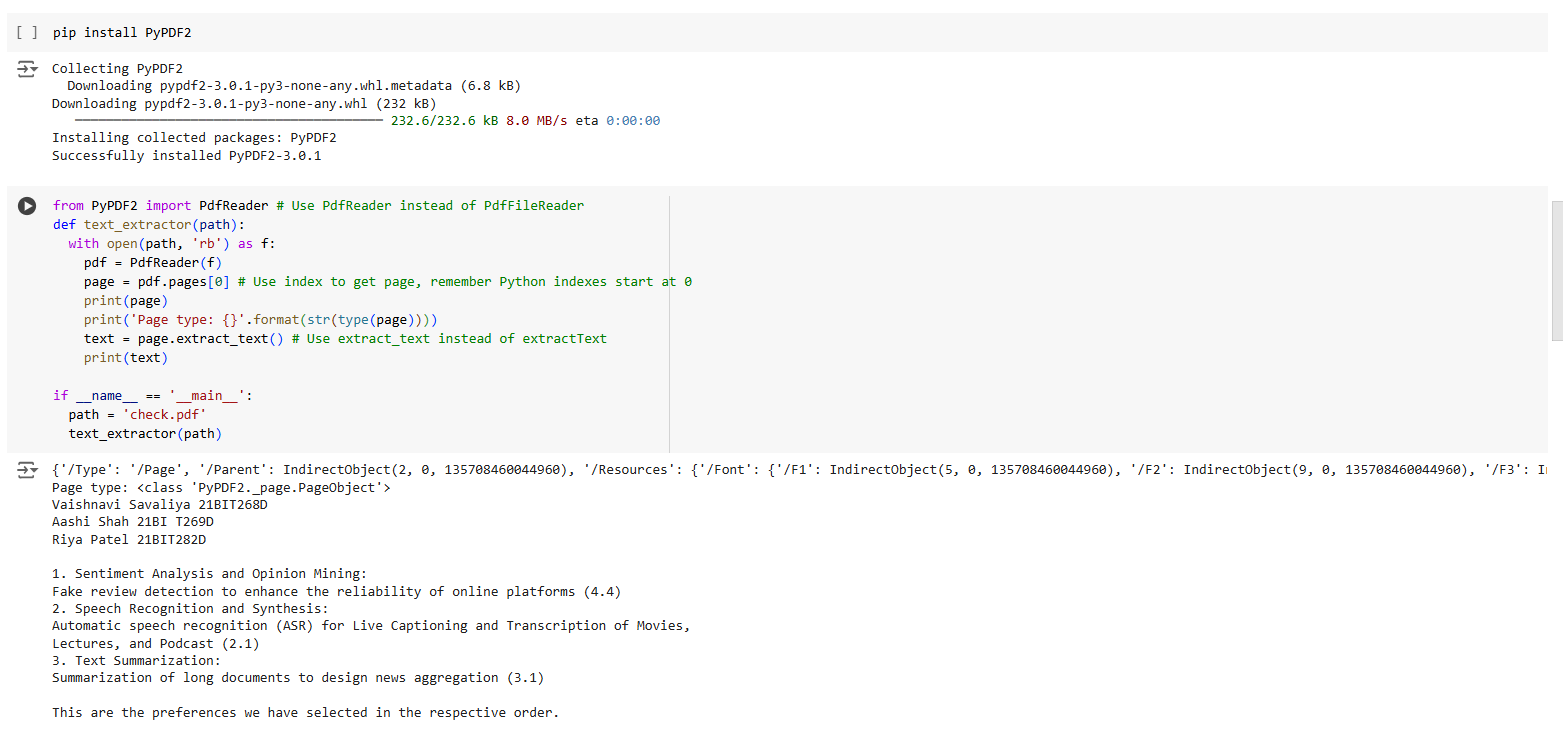
1. Write a Python script to read, write, and manipulate text files.
2. Extract and process text from PDF files using Python libraries like PyPDF2 and pdfminer.

**Objective:**

* Learn to read, write, and manipulate text files.



Extract and process text from PDF files.

****

**Email Extraction:**

import imaplib

import json

import mailparser

# Your Gmail credentials (should use environment variables or a secure method)

username = "aashi12e@gmail.com"

password = secret-key

# Connect to the Gmail IMAP server

mail = imaplib.IMAP4\_SSL("imap.gmail.com")

mail.login(username, password)

# Select the mailbox you want to use. Use 'INBOX' to read inbox emails.

mail.select("inbox")

# Search for all emails and get the latest one

status, messages = mail.search(None, "ALL")

if status == "OK":

    # Convert messages to a list of email IDs and get the last email ID

    email\_ids = messages[0].split()

    latest\_email\_id = email\_ids[-1]

    # Fetch the latest email by ID

    status, msg\_data = mail.fetch(latest\_email\_id, "(RFC822)")

    if status == "OK":

        # Get the email content

        raw\_email = msg\_data[0][1]

        # Decode the email content

        for encoding in ['utf-8', 'latin-1', 'iso-8859-1']:

            try:

                raw\_email\_decoded = raw\_email.decode(encoding)

                break

            except UnicodeDecodeError:

                raw\_email\_decoded = None

        if raw\_email\_decoded:

            # Parse the email using mailparser

            email\_message = mailparser.parse\_from\_string(raw\_email\_decoded)

            # Store email details in a dictionary (only body is extracted)

            email\_details = {

                "from": email\_message.from\_,

                "subject": email\_message.subject,

                "date": email\_message.date,

                "body": email\_message.text\_plain if email\_message.text\_plain else email\_message.text\_html

            }

            # Convert the email details to JSON format

            email\_json = json.dumps(email\_details, indent=4, default=str)

            # Print the JSON output

            print(email\_json)

        else:

            print("Failed to decode the email content.")

    else:

        print("Failed to fetch the latest email.")

else:

    print("Failed to search emails.")

# Close the mailbox and logout

mail.close()

mail.logout()

**Output :**

****

**Experiment 2: Pattern Searching in Text Using Regular Expressions**

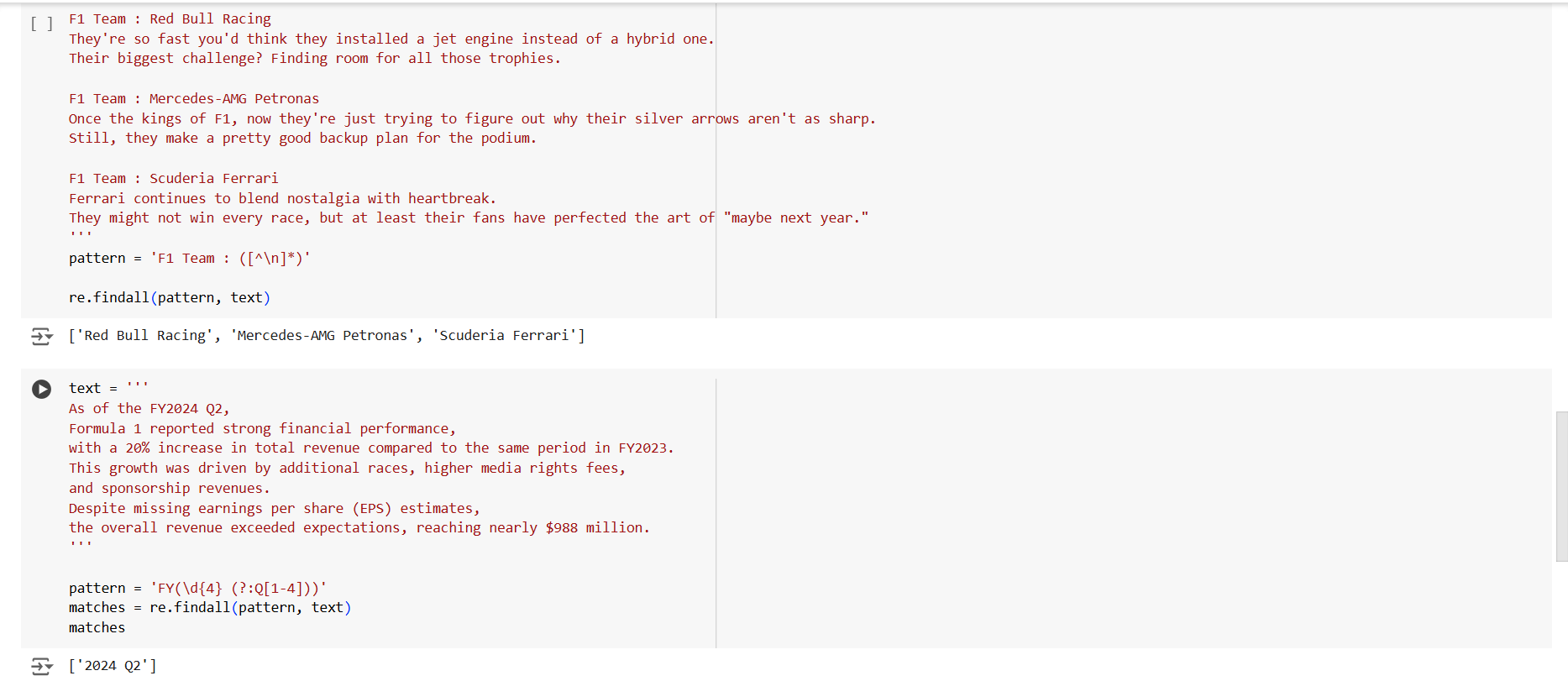
1. Implement a Python program to search for patterns in text utilizing regular expressions.

**Objective:**

* Utilize regular expressions to find patterns in text.







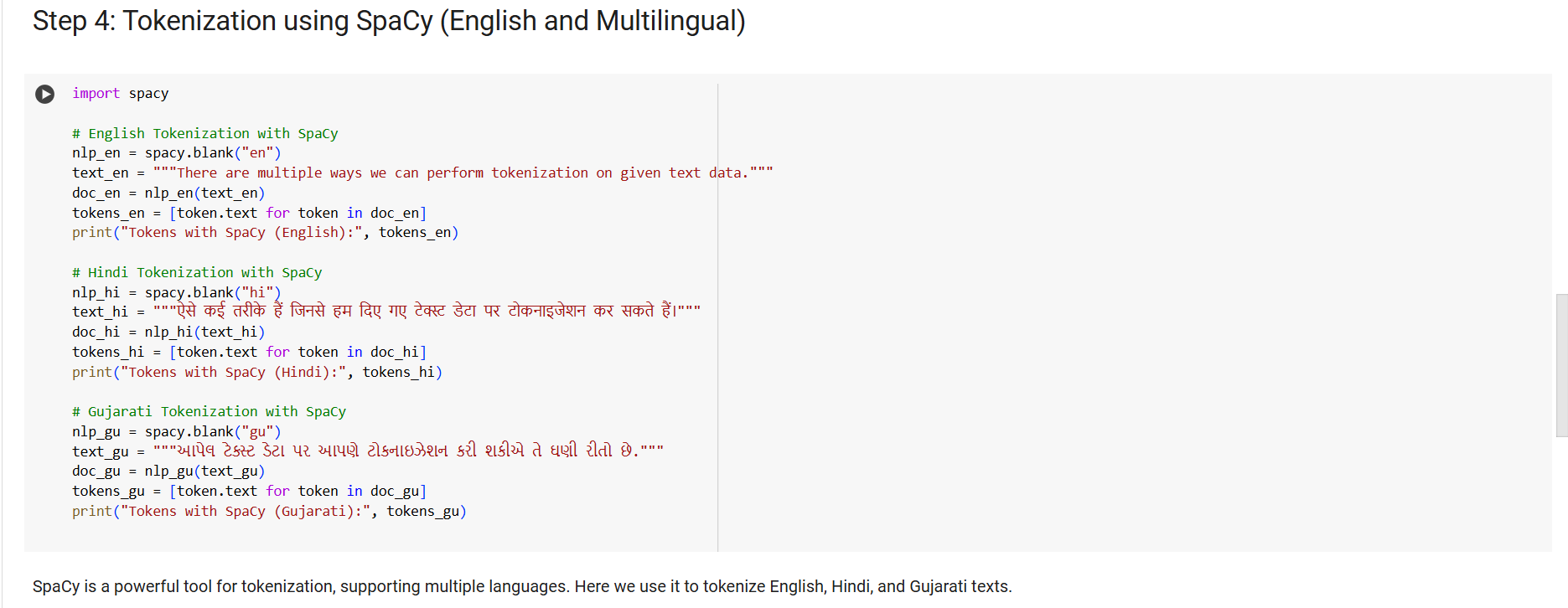
**Experiment 3: Ultra-fast Tokenization**

1. Create a Python script to implement and compare ultra-fast tokenization techniques using libraries like NLTK, SpaCy, and FastText.

**Objective:**

* Implement and compare different tokenization methods.

****

****

****

**Experiment 4: Stemming and Lemmatization**

1. Develop a Python program to apply stemming and lemmatization to normalize text data.

**Objective:**

* Apply stemming and lemmatization to normalize text.

# Stemming and Lemmatization Techniques in Text Preprocessing

import nltk

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.tokenize import word\_tokenize

from nltk.corpus import wordnet

# Download required resources

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger')

# Initialize stemmer and lemmatizer

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

# Function to get WordNet POS tags for more accurate lemmatization

def get\_wordnet\_pos(word):

    tag = nltk.pos\_tag([word])[0][1][0].upper()  # Get the first letter of POS tag

    tag\_dict = {"J": wordnet.ADJ,

                "N": wordnet.NOUN,

                "V": wordnet.VERB,

                "R": wordnet.ADV}

    return tag\_dict.get(tag, wordnet.NOUN)  # Default to NOUN if not found

# Function for stemming

def stem\_words(text):

    tokens = word\_tokenize(text)

    return [stemmer.stem(word) for word in tokens]

# Function for lemmatization (with POS tagging)

def lemmatize\_words(text):

    tokens = word\_tokenize(text)

    return [lemmatizer.lemmatize(word, get\_wordnet\_pos(word)) for word in tokens]

# Example text

text = """Formula 1 is the pinnacle of racing culture, where speed, strategy, and precision come together in a high-stakes blend of engineering and raw skill.

 With cars that push the limits of physics and drivers that embody relentless competitiveness, F1 captures the fascination of millions worldwide."""

# Perform stemming

stemmed\_words = stem\_words(text)

print("Stemmed Words:", stemmed\_words)

# Perform lemmatization

lemmatized\_words = lemmatize\_words(text)

print("Lemmatized Words:", lemmatized\_words)

# BONUS: Compare stemmed and lemmatized results side by side

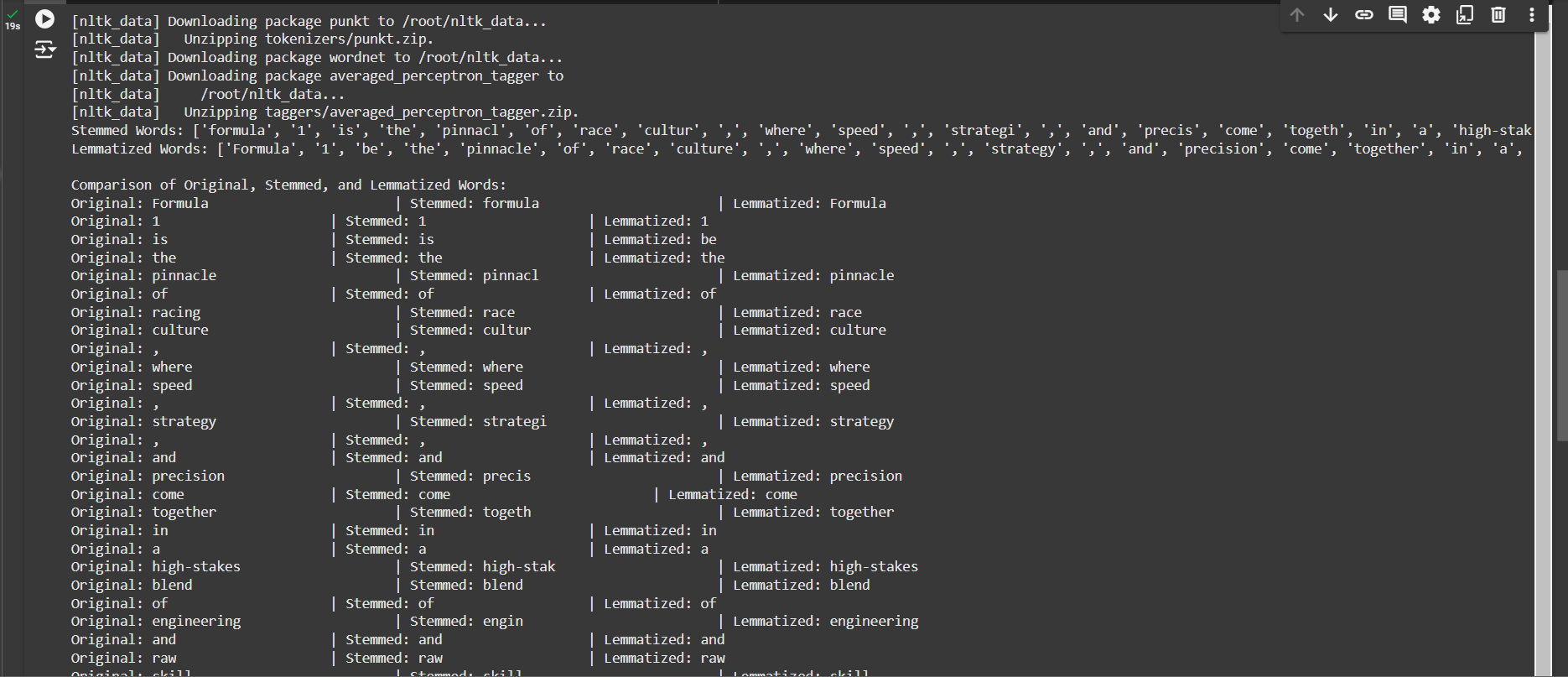
print("\nComparison of Original, Stemmed, and Lemmatized Words:")

tokens = word\_tokenize(text)

for i, token in enumerate(tokens):

    print(f"Original: {token} \t\t\t| Stemmed: {stemmed\_words[i]} \t\t\t| Lemmatized: {lemmatized\_words[i]}")

**Output :**

****

**Experiment 5: Vocabulary Matching Techniques**

1. Write a Python script to explore and implement various vocabulary matching techniques, including word embeddings and similarity measures.

**Objective:**

* Explore and implement various vocabulary matching techniques.

import Levenshtein

import gensim.downloader as api

from scipy.spatial.distance import cosine

# Load a pre-trained Word2Vec model

model = api.load("word2vec-ruscorpora-300")

def exact\_match(word1, word2):

    """Check for exact match between two words."""

    return word1.lower() == word2.lower()  # Ignore case sensitivity

def levenshtein\_distance(word1, word2):

    """Calculate Levenshtein distance between two words."""

    return Levenshtein.distance(word1, word2)

def jaccard\_match(word1, word2):

    """Calculate Jaccard similarity between two words."""

    set1 = set(word1)

    set2 = set(word2)

    intersection = len(set1.intersection(set2))

    union = len(set1.union(set2))

    return intersection / union if union > 0 else 0  # Avoid division by zero

def cosine\_similarity(word1, word2):

    """Calculate cosine similarity between two words using their word vectors."""

    try:

        # Get the word vectors

        vector1 = model[word1.lower()]  # Ensure lower case

        vector2 = model[word2.lower()]  # Ensure lower case

        # Calculate cosine similarity (1 - cosine distance)

        return 1 - cosine(vector1, vector2)

    except KeyError:

        return None  # Return None if the word is not in the model

# List of word pairs for comparison

word\_pairs = [

    ("happy", "joyful"),         # Synonyms

    ("sad", "unhappy"),          # Related antonyms

    ("light", "dark"),           # Antonyms

    ("car", "automobile"),       # Synonyms

    ("teacher", "student"),      # Related roles

    ("fast", "quick"),           # Synonyms

    ("big", "large"),            # Synonyms

    ("cold", "ice"),             # Related concepts

    ("run", "running"),          # Different forms of the same word

    ("strong", "powerful"),      # Synonyms

    ("beautiful", "ugly"),       # Antonyms

    ("child", "adult"),          # Different life stages

    ("friend", "enemy"),         # Opposites

    ("smart", "intelligent"),    # Synonyms

    ("fish", "swim"),            # Related actions

    ("love", "hate")             # Opposites

]

# Evaluate each pair of words

for word1, word2 in word\_pairs:

    print(f"Itr :: {word1} :: {word2}\n")

    print(f"Exact Match :: {exact\_match(word1, word2)} ")

    print(f"Levenshtein Distance :: {levenshtein\_distance(word1, word2)}")

    print(f"Jaccard Similarity :: {jaccard\_match(word1, word2)}")

    cosine\_sim = cosine\_similarity(word1, word2)

    if cosine\_sim is not None:

        print(f"Cosine Similarity :: {cosine\_sim}")

    else:

        print(f"Cosine Similarity :: One or both words not in the model.")

    print("\n")

**Output :**



**Experiment 6: Part of Speech Tagging**

1. Create a Python program to automatically tag parts of speech in raw text files using libraries like NLTK and SpaCy.

**Objective:**

* Automatically tag parts of speech in raw text files.

import nltk

import spacy

nltk.download('punkt', quiet=True)

nltk.download('averaged\_perceptron\_tagger', quiet=True)

def pos\_tag\_nltk(text):

    words = nltk.word\_tokenize(text)

    pos\_tags = nltk.pos\_tag(words)

    return pos\_tags

# Function to perform POS tagging using spaCy

def pos\_tag\_spacy(text):

    doc = nlp(text)

    pos\_tags = [(token.text, token.pos\_) for token in doc]

    return pos\_tags

# Function to read text from a file

def read\_text\_file(file\_path):

    try:

        with open(file\_path, 'r', encoding='utf-8') as file:

            return file.read()

    except FileNotFoundError:

        print(f"Error: The file '{file\_path}' was not found.")

        return None

# Function to display POS tags

def display\_pos\_tags(pos\_tags, library\_name):

    print(f"\nPOS tagging using {library\_name}:\n")

    for word, tag in pos\_tags:

        print(f"{word} -> {tag}")

# Path to the raw text file

file\_path = 'sample.txt'  # Replace with your file path

# Read the text from the file

text\_content = read\_text\_file(file\_path)

if text\_content:

    # Perform POS tagging using NLTK

    nltk\_pos\_tags = pos\_tag\_nltk(text\_content)

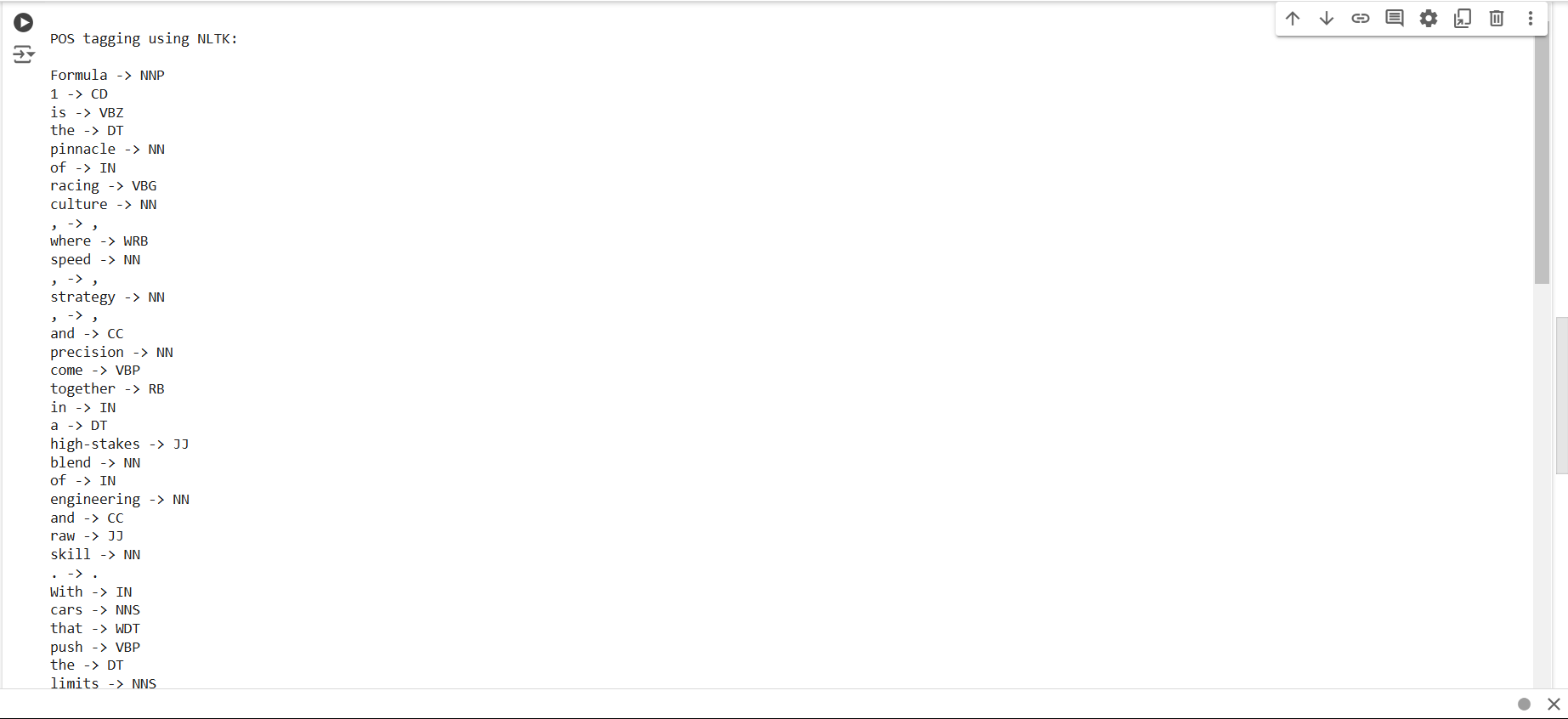
    display\_pos\_tags(nltk\_pos\_tags, "NLTK")

    # Perform POS tagging using spaCy

    spacy\_pos\_tags = pos\_tag\_spacy(text\_content)

    display\_pos\_tags(spacy\_pos\_tags, "spaCy")

**Output :**

****

**Experiment 7: Visualizing POS and Named Entity Recognition (NER)**

1. Implement a Python script to visualize POS tagging and NER using tools like SpaCy and Matplotlib.

**Objective:**

* Create visualizations for POS tagging and NER.

import spacy

from spacy import displacy

# Load the spaCy model

try:

    nlp = spacy.load("en\_core\_web\_sm")

except Exception as e:

    print(f"Error loading spaCy model: {e}")

def visualize\_pos\_and\_ner(text):

    # Process the text

    doc = nlp(text)

    # Display Named Entity Recognition (NER)

    print("Named Entity Recognition:")

    displacy.render(doc, style='ent', jupyter=True)  # Use jupyter=True for Jupyter Notebook

    # Display Part-of-Speech (POS) tagging (dependency parse)

    print("Part-of-Speech Tagging:")

    displacy.render(doc, style='dep', jupyter=True)  # Use jupyter=True for Jupyter Notebook

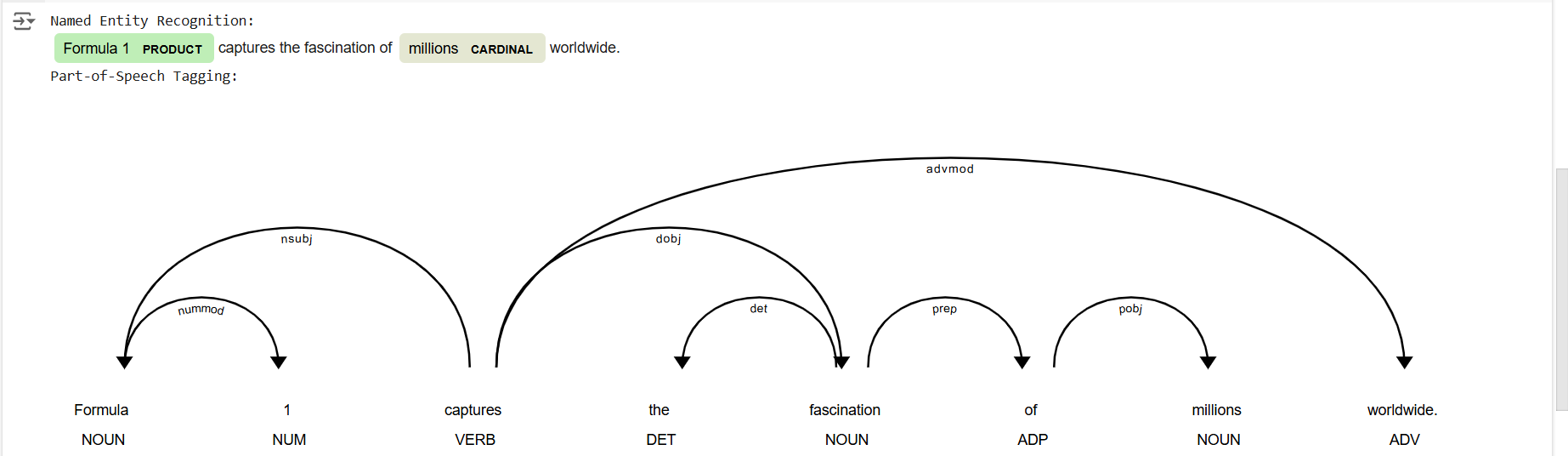
# Sample text

sample\_text = "Formula 1 captures the fascination of millions worldwide."

# Visualize POS and NER

visualize\_pos\_and\_ner(sample\_text)

**Output :**

****

**Experiment 8: Text Classification Algorithms**

1. Develop a Python program to implement text classification algorithms such as Naive Bayes, SVM, and neural networks, and evaluate their performance.

**Objective:**

Implement text classification algorithms and evaluate their performance.

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

from keras.models import Sequential

from keras.layers import Dense

from keras.utils import to\_categorical

from sklearn.preprocessing import LabelEncoder

# Load and split dataset

newsgroups = fetch\_20newsgroups(subset='all')

X = newsgroups.data

y = newsgroups.target

# Convert text to TF-IDF features

tfidf = TfidfVectorizer(stop\_words='english', max\_features=10000)

X\_tfidf = tfidf.fit\_transform(X)

# Split data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_tfidf, y, test\_size=0.2, random\_state=42)

# Function to evaluate models

def evaluate\_model(model, X\_train, X\_test, y\_train, y\_test):

    model.fit(X\_train, y\_train)

    predictions = model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, predictions)

    report = classification\_report(y\_test, predictions)

    return accuracy, report

# Model 1: Naive Bayes

print("Naive Bayes Model")

nb\_model = MultinomialNB()

nb\_accuracy, nb\_report = evaluate\_model(nb\_model, X\_train, X\_test, y\_train, y\_test)

print(f"Accuracy: {nb\_accuracy}")

print(nb\_report)

# Model 2: Support Vector Machine (SVM)

print("Support Vector Machine Model")

svm\_model = SVC(kernel='linear', C=1)

svm\_accuracy, svm\_report = evaluate\_model(svm\_model, X\_train, X\_test, y\_train, y\_test)

print(f"Accuracy: {svm\_accuracy}")

print(svm\_report)

# Model 3: Neural Network

print("Neural Network Model")

# Encode labels for neural network

encoder = LabelEncoder()

y\_train\_enc = to\_categorical(encoder.fit\_transform(y\_train))

y\_test\_enc = to\_categorical(encoder.transform(y\_test))

# Define the neural network

nn\_model = Sequential()

nn\_model.add(Dense(512, input\_shape=(X\_train.shape[1],), activation='relu'))

nn\_model.add(Dense(256, activation='relu'))

nn\_model.add(Dense(y\_train\_enc.shape[1], activation='softmax'))

# Compile the neural network

nn\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the neural network

nn\_model.fit(X\_train, y\_train\_enc, epochs=5, batch\_size=128, validation\_split=0.1)

# Evaluate the neural network

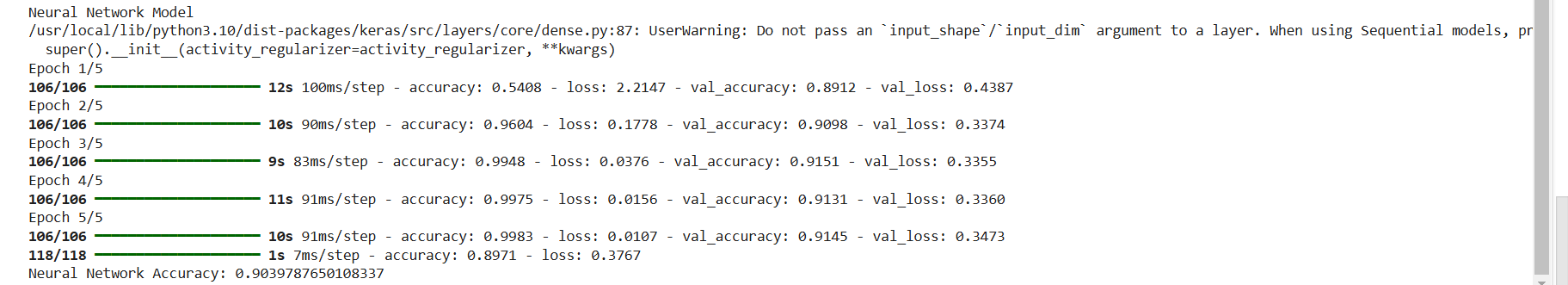
nn\_loss, nn\_accuracy = nn\_model.evaluate(X\_test, y\_test\_enc)

print(f"Neural Network Accuracy: {nn\_accuracy}")

**Output :**

****

****

****

**Experiment 9: Non-negative Matrix Factorization (NMF) for Topic Modeling**

1. Write a Python script to apply Non-negative Matrix Factorization (NMF) for topic modeling and document clustering.

**Objective:**

* Apply NMF for topic modeling and document clustering.

****

**Output :**

****

**Experiment 10: Exploring Different Algorithms**

1. Create a Python program to implement and compare various NLP algorithms for tasks such as classification, clustering, and sentiment analysis.

**Objective:**

* Implement and compare various NLP algorithms for tasks like classification, clustering, and sentiment analysis.

****

**Output:**

****

**Experiment 11: Sentiment Analysis**

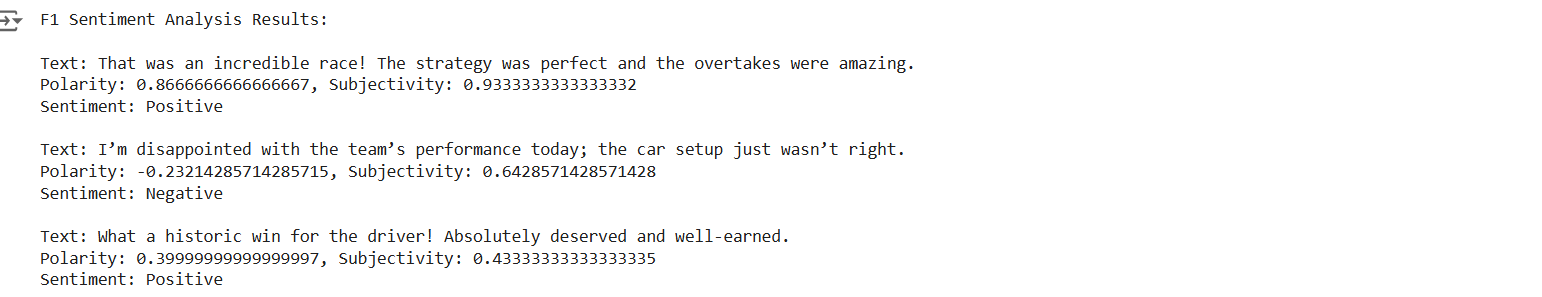
1. Develop a Python script to perform sentiment analysis on text data using lexicon-based methods and machine learning models.

**Objective:**

* Perform sentiment analysis on text data using various methods.

****

**Output :**

****

**Experiment 12: Deep Learning in NLP**

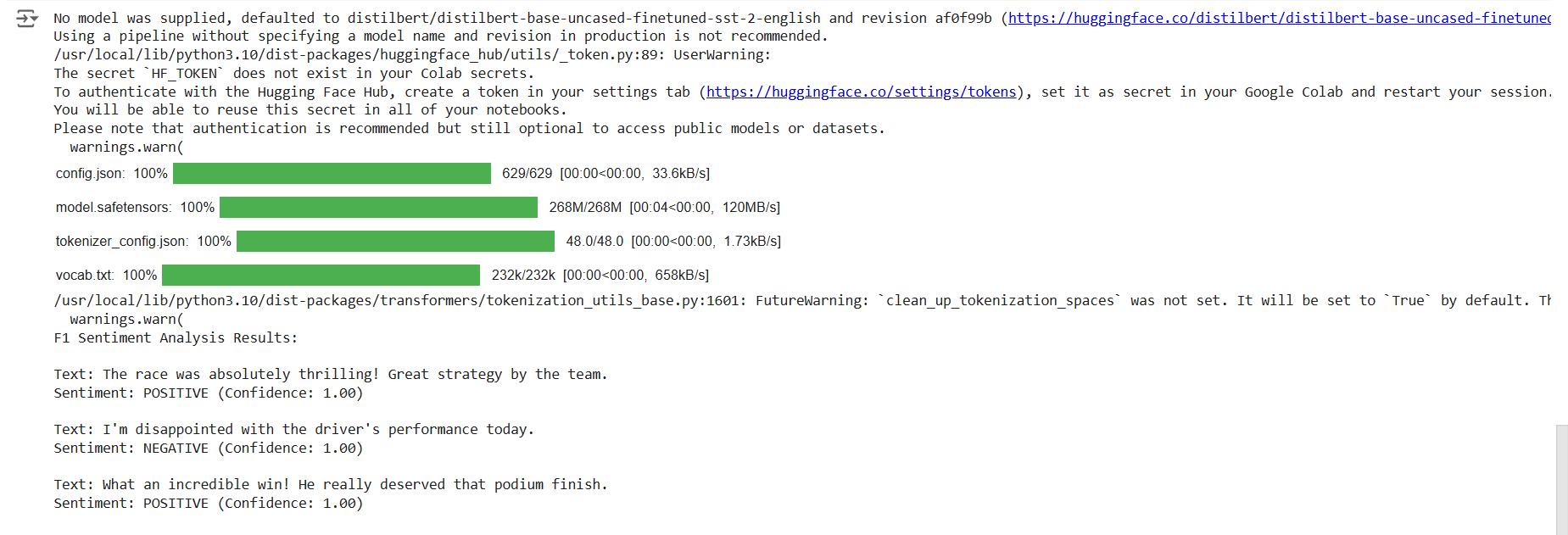
1. Write a Python program to apply deep learning models such as RNNs, LSTMs, or Transformers for NLP tasks, including experimenting with pre-trained models like BERT or GPT.

**Objective:**

* Apply deep learning models for NLP tasks.



**Output :**

****