Students Name : _____ RollNo: _____
Subject: Natural Language Processing
Class: MCS Part 2
Academic Year : 2024-2025

INDEX			
No.	**DATE**	**TITLE**	**SIGN**
1		a. Write a program to implement Edit Distance Algorithm b. Write a program to implement Part of Speech Tagging	
2		Write a program to implement sentence segmentation and word tokenization	
3		Write a program to Implement stemming and lemmatization	
4		Write a program to Implement Text Summarization for the given sample text	
5		a. Write a program to Implement a n-gram model b. Write a program to Implement Tri-gram model to predict next word probability	
6		Write a program to Implement Named Entity Recognition (NER)	
7		Write a program to Implement PoS tagging using HMM	
8		Write a program to Implement syntactic parsing of a given text	
9		Write a program to Implement dependency parsing of a given text	
10		Consider a scenario of applying NLP in Customer Service. Process the data to understand the voice of the Customer (intent mining, Top words, word cloud, classify topics).	
11		Consider a scenario of Online Review and demonstrate the concept of sentiment analysis and emotion mining by applying various approaches like lexicon-based approach and rule-based approaches	
12		Write a program to Implement Skip-gram	
13		Write a program to Implement SMS Fraud Detection	

Practical No. 1(a)

Aim: Write a program to implement Edit Distance Algorithm

```
def editDistance(str1, str2, m, n):
    if m == 0:
         return n
    if n == 0:
        return m
    if str1[m-1]== str2[n-1]:
        return editDistance(str1, str2, m-1, n-1)
    return 1 + min(editDistance(str1, str2, m, n-1),    # Insert
                   editDistance(str1, str2, m-1, n),    # Remove
                   editDistance(str1, str2, m-1, n-1)    # Replace
                   )

str1 = "sunday"
str2 = "saturday"
print (editDistance(str1, str2, len(str1), len(str2)))
```

Practical No. 1(b)

Aim: Write a program to implement Part of Speech Tagging

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
text = "NLTK is a powerful library for natural language processing."
words = word_tokenize(text)
# Performing PoS tagging
pos_tags = pos_tag(words)
# Displaying the PoS tagged result in separate lines
print("Original Text:")
print(text)
print("\nPoS Tagging Result:")
for word, pos_tag in pos_tags:
    print(f"{word}: {pos_tag}")
```

Practical No.2

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

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize, sent_tokenize
text="Good Morning all. Hope you will like this video. Thank you."
#Sentence Tokenize
sentence_tokens = sent_tokenize(text)
print(sentence_tokens)
for sentence in sentence_tokens:
  print(sentence)
#Word Tokenize
from nltk.tokenize import word_tokenize
sentence="Let's understand this concept in detail."
word_tokens=word_tokenize(sentence)
print(word_tokens)
from nltk.tokenize import TreebankWordTokenizer,WordPunctTokenizer,WhitespaceTokenizer
tree_tokenizer=TreebankWordTokenizer()
word_punct_tokenizer=WordPunctTokenizer()
white_space_tokenizer=WhitespaceTokenizer()
word_tokens=tree_tokenizer.tokenize(sentence)
print(word_tokens)
word_tokens=word_punct_tokenizer.tokenize(sentence)
print(word_tokens)
word_tokens=white_space_tokenizer.tokenize(sentence)
print(word_tokens)
```

Practical No.3

Aim: Write a program to Implement stemming and lemmatization

```
#STEMMING: The goal of both stemming and lemmatization is to reduce an inflected (or derived) word's form to its root or base form.
#It is essential for many NLP-related tasks such as information retrieval,text summarization,topic extraction and more.
from nltk.stem import PorterStemmer, LancasterStemmer
porter_stemmer=PorterStemmer()
print(porter_stemmer.stem('Observing'))
print(porter_stemmer.stem('observs'))
print(porter_stemmer.stem('observe'))
```

```
#Lemmatization is a process that uses vocabulary and morphological analysis of words to remove the inflected endings
#to achieve its base form(dictionary form),which is known as lemma.
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer=WordNetLemmatizer()
print(lemmatizer.lemmatize('running'))
print(lemmatizer.lemmatize("runs"))
#Lemmatizer-Retuns verb,noun,Adverb,Adjective form
def lemmatize(word):
  lemmatizer=WordNetLemmatizer()
  print("verb form "+lemmatizer.lemmatize(word,pos="v"))
  print("noun form "+lemmatizer.lemmatize(word,pos="n"))
  print("adverb form "+lemmatizer.lemmatize(word,pos="r"))
  print("adjective form "+lemmatizer.lemmatize(word,pos="a"))
lemmatize("programming")
lemmatize("running")
```

```
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

stemmer = PorterStemmer()
lemmatizer=WordNetLemmatizer()
print(stemmer.stem("deactivating"))
print(stemmer.stem("deactivated"))
print(stemmer.stem("deactivates"))
print(lemmatizer.lemmatize("deactivating",pos="v"))
print(lemmatizer.lemmatize("deactivating",pos="r"))
print(lemmatizer.lemmatize("deactivating",pos="n"))
print(stemmer.stem('stones'))
print(stemmer.stem('speaking'))
print(stemmer.stem('bedroom'))
print(stemmer.stem('jokes'))
print(stemmer.stem('lisa'))
print(stemmer.stem('purple'))
print(lemmatizer.lemmatize('stones'))
print(lemmatizer.lemmatize('speaking'))
print(lemmatizer.lemmatize('bedroom'))
print(lemmatizer.lemmatize('jokes'))
print(lemmatizer.lemmatize('lisa'))
print(lemmatizer.lemmatize('purple'))
```

Practical No.4

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

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
nltk.download('stopwords')
nltk.download('punkt')

#Input text - to summarize
text="""
Text summarization is the process of creating a short, coherent,and fluent summary of a longer text document and involves the
outlining of the text's major points.
Extractive Summarization:Extractive methods attempt to summarize articles by identifying the important sentences or phrases
from the original text and stitch together portions of the content to produce a condensed version. These extracted sentences are
then used to form the summary.
Abstractive Summarization:This technique, unlike extraction, relies on being able to paraphrase and shorten parts of a
document using advanced natural language techniques. Since abstractive machine learning algorithms can generate new phrases
and sentences to capture the meaning of the source document. When such abstraction is done correctly in deep learning problems,they
can assist in overcoming grammatical inaccuracies.
"""

stopWords = set(stopwords.words("english"))
print(stopWords)
words = word_tokenize(text)
print(words)
```

```
#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
print(freqTable)
```

```
#Creating a dictionary to keep the score of each sentence
sentences = sent_tokenize(text)
print(sentences)
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
print(sentenceValue)
sumValues = 0
for sentence in sentenceValue:
    sumValues += sentenceValue[sentence]
print(sumValues)
#Average value of a sentence from the original text
average = int(sumValues / len(sentenceValue))
print(average)
#Storing sentences into our summary.
summary = ''
for sentence in sentences:
    if (sentence in sentenceValue) and (sentenceValue[sentence] >
                                        (1.2 * average)):
        summary += " " + sentence
print(summary)
```

Practical No.5(a)

Aim: Write a program to Implement a n-gram model

```
#Unigrams or 1-grams
from nltk.util import ngrams

n = 1
sentence = 'You will face many defeats in life, but never let yourself be defeated.'
unigrams = ngrams(sentence.split(), n)

for item in unigrams:
    print(item)

#Bigrams or 2-grams
n1 = 2
sentence1 = 'The purpose of our life is to happy'
bigrams = ngrams(sentence.split(), n1)

for item1 in bigrams:
    print(item1)

#Trigrams or 3-grams
n2 = 3
sentence2 = 'Whoever is happy will make others happy too'
trigrams = ngrams(sentence.split(), n2)

for item2 in trigrams:
    print(item2)
```

```python
#Generic Example of ngram in NLTK
from nltk import ngrams
sentence = input("Enter the sentence: ")
n = int(input("Enter the value of n: "))
n_grams = ngrams(sentence.split(), n)
for grams in n_grams:
    print(grams)
```

```python
#Generic Example of everygram in NLTK
from nltk.util import everygrams

message = "who let the dogs out"
msg_split = message.split()

list(everygrams(msg_split))
```

Practical No.5(b)

Aim: Write a program to Implement Tri-gram model to predict next word probability

```
import nltk
nltk.download('reuters')
"""In natural language processing (NLP), there's a dataset called the Reuters-21578 dataset,
which is a collection of news documents that is often used for text classification and other NLP tasks."""
nltk.download('punkt')
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for sentence in reuters.sents():
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        model[(w1, w2)][w3] += 1

# Let's transform the counts to probabilities
for w1_w2 in model:
    total_count = float(sum(model[w1_w2].values()))
    for w3 in model[w1_w2]:
        model[w1_w2][w3] /= total_count

sorted(dict(model["the","news"]).items(), key=lambda x:-1*x[1])
```

Practical No.6

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

```
import spacy
#import requests
nlp = spacy.load("en_core_web_sm")

content = "Trinamool Congress leader Mahua Moitra has moved the Supreme Court against her
           expulsion from the Lok Sabha over the cash-for-query allegations against her.
           Moitra was ousted from the Parliament last week after the Ethics Committee of
           the Lok Sabha found her guilty of jeopardising national security by sharing her
           parliamentary portal's login credentials with businessman Darshan Hiranandani."
doc = nlp(content)

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
#Visualize
from spacy import displacy
displacy.render(doc, style="ent")
```

```
#creating a dataframe from the named entities extracted by spaCy,
import pandas as pd
entities = [(ent.text, ent.label_, ent.lemma_) for ent in doc.ents]
df = pd.DataFrame(entities, columns=['text', 'type', 'lemma'])
print(df)
```

Practical No.7

Aim: Write a program to Implement PoS tagging using HMM

Code1:

```
import nltk
from nltk.corpus import treebank
from nltk.tag import hmm
from nltk.probability import FreqDist
from nltk.tokenize import word_tokenize
# Download the treebank dataset if you haven't already
nltk.download('treebank')
nltk.download('punkt')
# Load the tagged sentences
tagged_sents = treebank.tagged_sents()
# Split data into training and testing sets
train_sents = tagged_sents[:3000]
test_sents = tagged_sents[3000:]
# Create and train the HMM tagger
trainer = hmm.HiddenMarkovModelTrainer()
hmm_tagger = trainer.train(train_sents)
# Evaluate the tagger
accuracy = hmm_tagger.evaluate(test_sents)
print(f"Tagger accuracy: {accuracy:.2f}")
# Sample sentence
sentence = "This is a test sentence."
# Tokenize the sentence
tokens = word_tokenize(sentence)
# Tag the sentence
tagged_sentence = hmm_tagger.tag(tokens)
print(tagged_sentence)
```

Code2:

```
import nltk
from nltk.corpus import brown
from collections import defaultdict, Counter
import numpy as np
# Download required NLTK data files
nltk.download('brown')
nltk.download('universal_tagset')
# Get tagged sentences from the Brown corpus
tagged_sentences = brown.tagged_sents(tagset='universal')
# Split the data into training and testing sets
train_data = tagged_sentences[:int(0.9 * len(tagged_sentences))]
test_data = tagged_sentences[int(0.9 * len(tagged_sentences)):]
```

```
# Extract states and observations
states = set()
observations = set()
for sentence in train_data:
    for word, tag in sentence:
        states.add(tag)
        observations.add(word.lower())
# Initialize counters
transition_counts = defaultdict(Counter)
emission_counts = defaultdict(Counter)
start_counts = Counter()
# Count occurrences
for sentence in train_data:
    previous_tag = None
    for word, tag in sentence:
        word = word.lower()
        if previous_tag is None:
            start_counts[tag] += 1
        else:
            transition_counts[previous_tag][tag] += 1
        emission_counts[tag][word] += 1
        previous_tag = tag
# Calculate probabilities
start_probs = {tag: count / sum(start_counts.values()) for tag, count in start_counts.items()}
transition_probs = {tag: {next_tag: count / sum(next_tags.values()) for next_tag, count in next_tags.items()}
                    for tag, next_tags in transition_counts.items()}
emission_probs = {tag: {word: count / sum(words.values()) for word, count in words.items()}
                  for tag, words in emission_counts.items()}
```

```
def viterbi(sentence, states, start_probs, transition_probs, emission_probs):
    V = [{}]
    path = {}
    # Initialize base cases (t == 0)
    for state in states:
        V[0][state] = start_probs.get(state, 0) * emission_probs[state].get(sentence[0], 0)
        path[state] = [state]
    # Run Viterbi for t > 0
    for t in range(1, len(sentence)):
        V.append({})
        new_path = {}
        for state in states:
            (prob, best_state) = max(
                (V[t-1][prev_state] * transition_probs[prev_state].get(state, 0) * emission_probs[state].get(sentence[t], 0), prev_state)
                for prev_state in states
            )
            V[t][state] = prob
            new_path[state] = path[best_state] + [state]
        path = new_path
    # Find the most probable state sequence
    n = len(sentence) - 1
    (prob, best_state) = max((V[n][state], state) for state in states)
    return path[best_state]
```

```
# Test the HMM-based PoS tagger
def pos_tag(sentence, states, start_probs, transition_probs, emission_probs):
    sentence = [word.lower() for word in sentence]
    tags = viterbi(sentence, states, start_probs, transition_probs, emission_probs)
    return list(zip(sentence, tags))
# Example usage
test_sentence = "The quick brown fox jumps over the lazy dog".split()
tagged_sentence = pos_tag(test_sentence, states, start_probs, transition_probs, emission_probs)
print(tagged_sentence)
```

Practical No.8

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

```
import nltk
from nltk import CFG
from nltk.parse import ChartParser

# Define a simple context-free grammar
grammar = CFG.fromstring("""
  S -> NP VP
  NP -> Det N | Det N PP
  PP -> P NP
  VP -> V NP | V NP PP
  Det -> 'the' | 'a'
  N -> 'dog' | 'cat' | 'park' | 'telescope'
  V -> 'saw' | 'ate'
  P -> 'in' | 'on' | 'with'
""")

# Create a parser with the defined grammar
parser = ChartParser(grammar)

# Example sentence
sentence = 'the dog saw the cat in the park'.split()

# Parse the sentence
for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()
```

Practical No.9

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

```
import spacy
from spacy import displacy

# Load the pre-trained English model
nlp = spacy.load('en_core_web_sm')

# Example sentence
sentence = "The dog saw the cat in the park."

# Parse the sentence
doc = nlp(sentence)

# Print the syntactic dependency information
for token in doc:
    print(f'{token.text:<12} {token.dep_:<10} {token.head.text}')

# Visualize the parse tree using displacy
displacy.serve(doc, style='dep')
```

Practical No.10

Aim: Consider a scenario of applying NLP in Customer Service. Process the data to understand the voice of the Customer (intent mining, Top words, word cloud, classify topics).

```
#Intent Mining
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier

# Sample data
texts = ['I need help with my order', 'The product arrived damaged', 'How do I reset my password?']
labels = ['Assistance', 'Complaint', 'Inquiry']

# Feature extraction
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(texts)

# Model training
classifier = RandomForestClassifier()
classifier.fit(X, labels)

# Prediction
new_text = 'My shipment is delayed'
new_X = vectorizer.transform([new_text])
predicted_intent = classifier.predict(new_X)
print(predicted_intent)
```

```
#Topwords
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer

def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

# Sample usage
corpus = ['I love this product', 'This product is bad', 'I need help with this product','This product is not working','Product is not working']
top_words = get_top_n_words(corpus, n=5)
print(top_words)

#WordCloud
#pip install wordcloud
#pip --upgrade pip or pip --upgrade pillow
from wordcloud import WordCloud
import matplotlib.pyplot as plt
text = ' '.join(corpus)
wordcloud = WordCloud(width=200, height=200, background_color='white').generate(text)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
#Classify Topics
import gensim
from gensim import corpora

def topic_classification(corpus, num_topics=4):
    texts = [text.split() for text in corpus]
    dictionary = corpora.Dictionary(texts)
    doc_term_matrix = [dictionary.doc2bow(text) for text in texts]

    Lda = gensim.models.ldamodel.LdaModel
    lda_model = Lda(doc_term_matrix, num_topics=num_topics, id2word = dictionary, passes=50)

    topics = lda_model.print_topics(num_words=6)
    for topic in topics:
        print(topic)

# Sample usage
topic_classification(corpus)
```

```
import plotly.express as px

# Intent distribution
intent_counts = {'Assistance': 120, 'Complaint': 80, 'Inquiry': 50}
fig = px.pie(names=intent_counts.keys(), values=intent_counts.values(), title='Intent Distribution')
fig.show()
```

```
pip install fpdf

from fpdf import FPDF
def generate_report(intent_counts, top_words, topics):
    # Initialize PDF
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)
    # Title
    pdf.set_font("Arial", 'B', 16)
    pdf.cell(200, 10, txt="Customer Service NLP Analysis Report", ln=True, align='C')
    pdf.ln(10)
    # Intent Distribution
    pdf.set_font("Arial", 'B', 14)
    pdf.cell(200, 10, txt="Intent Distribution:", ln=True)
    pdf.set_font("Arial", size=12)
    for intent, count in intent_counts.items():
        pdf.cell(200, 10, txt=f"{intent}: {count}", ln=True)
    pdf.ln(10)
    # Top Words
    pdf.set_font("Arial", 'B', 14)
    pdf.cell(200, 10, txt="Top Words:", ln=True)
    pdf.set_font("Arial", size=12)
    for word, freq in top_words:
        pdf.cell(200, 10, txt=f"{word}: {freq}", ln=True)
    pdf.ln(10)
```

```
    # Identified Topics
    pdf.set_font("Arial", 'B', 14)
    pdf.cell(200, 10, txt="Identified Topics:", ln=True)
    pdf.set_font("Arial", size=12)
    for topic in topics:
        pdf.multi_cell(0, 10, txt=str(topic))
    # Save PDF
    pdf_output_path = "report.pdf"
    pdf.output(pdf_output_path)
    print(f"PDF report saved to {pdf_output_path}")
topics = ["Topic 1: Detailed explanation of topic 1", "Topic 2: Detailed explanation of topic 2"]
generate_report(intent_counts, top_words, topics)
```

Practical No.11

Aim: Consider a scenario of Online Review and demonstrate the concept of sentiment analysis and emotion mining by applying various approaches like lexicon-based approach and rule-based approaches

```
pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Initialize VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()
# Review text
review_text = "Had a wonderful experience at the new restaurant.
The food was absolutely delicious, and the service was fantastic. However, the ambiance could have been better,
and the prices were a bit high."
# Analyze sentiment
sentiment = analyzer.polarity_scores(review_text)
print("Sentiment Analysis using VADER:")
print(sentiment)
```

```
def rule_based_sentiment(text):
    positive_words = ['wonderful', 'delicious', 'fantastic']
    negative_words = ['high', 'worse', 'bad']
    score = 0
    for word in positive_words:
        if word in text:
            score += 1
    for word in negative_words:
        if word in text:
            score -= 1
    if score > 0:
        return 'Positive'
    elif score < 0:
        return 'Negative'
    else:
        return 'Neutral'
# Analyze sentiment
sentiment = rule_based_sentiment(review_text)
print("Sentiment Analysis using Rule-Based Approach:")
print(sentiment)
```

```python
# Example emotion lexicon (simplified)
emotion_lexicon = {
    'happy': 'joy',
    'wonderful': 'joy',
    'delicious': 'joy',
    'fantastic': 'joy',
    'high': 'anger',
    'worse': 'sadness',
    'bad': 'sadness'
}
def extract_emotions(text):
    words = text.lower().split()
    emotions = []
    for word in words:
        if word in emotion_lexicon:
            emotions.append(emotion_lexicon[word])
    return emotions
# Extract emotions
emotions = extract_emotions(review_text)
print("Emotion Mining using Lexicon-Based Approach:")
print(emotions)
```

```python
def detect_emotion(text):
    if "wonderful" in text or "delicious" in text or "fantastic" in text:
        return 'joy'
    elif "high" in text:
        return 'anger'
    else:
        return 'neutral'
# Detect emotions
emotion = detect_emotion(review_text)
print("Emotion Mining using Rule-Based Approach:")
print(emotion)
```

Practical No.12

Aim: Write a program to Implement Skip-gram.

```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
# Sample text corpus
corpus = [
    "She is a great dancer.",
    "He is a wonderful musician.",
    "They are excellent at their craft.",
]
# Preprocess the text: Tokenize and clean
processed_corpus = [simple_preprocess(sentence) for sentence in corpus]
# Create and train the Skip-gram model
# sg=1 indicates Skip-gram, vector_size is the dimensionality of word vectors
model = Word2Vec(sentences=processed_corpus, vector_size=100, window=2, sg=1, min_count=1)
#Save and load the model
model.save("skipgram_model.bin")
model = Word2Vec.load("skipgram_model.bin")
# Get vector for a word
word_vector = model.wv['great']
print(f"Vector for 'great': {word_vector}")
# Find similar words
similar_words = model.wv.most_similar('great')
print("Most similar words to 'great':")
for word, similarity in similar_words:
    print(f"{word}: {similarity:.4f}")
```

Practical No.13

Aim: Write a program to Implement SMS Fraud Detection.

```
pip install vaderSentiment
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
# Sample SMS data
data = {
    'sms_text': [
        "Your account has been credited with $500",
        "Call us immediately! Your account is at risk",
        "Your loan application has been approved",
        "Win a $1000 gift card! Click here",
        "Reminder: Your payment is due tomorrow"
    ],
    'label': ['transaction', 'fraud_alert', 'transaction', 'promotion', 'reminder']
}
```

```
df = pd.DataFrame(data)
# Text processing and model training
X_train, X_test, y_train, y_test = train_test_split(df['sms_text'], df['label'], test_size=0.2, random_state=42)
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(X_train, y_train)
# Predict on new SMS
new_sms = ["Congratulations! You've won $1000. Claim now"]
predicted_label = model.predict(new_sms)
print(f"Predicted Label: {predicted_label[0]}")
# Sentiment Analysis
analyzer = SentimentIntensityAnalyzer()
sentiment = analyzer.polarity_scores(new_sms[0])
print(f"Sentiment Analysis: {sentiment}")
```