NAME: **MOHAIB KHAN**

CLASS: **BSSE-7TH-B**

ROLL NO: **10256**

SUBMITTION DATE: **30/10/2024**

QUESTION NO 01

```
column_names = df.columns.tolist()
print(column_names)
```

```
['abstract', 'author', 'date', 'pdf_url', 'title', 'pdf_text']
```

Step 1: Set Up Google Drive Access and Load Corpus

```
from google.colab import drive
import pandas as pd

# Mount Google Drive
drive.mount('/content/drive')

# Load the corpus from CSV, focusing on the 'abstract' or 'pdf_text' column
corpus_file_path = '/content/drive/MyDrive/arxiv_papers.csv'
df = pd.read_csv(corpus_file_path)

# Combine all text data from the 'abstract' column; change to 'pdf_text' if needed
data = " ".join(df['abstract'].dropna().tolist())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Step 2: Text Preprocessing

This code performs text cleaning, tokenization, stopword removal, and lemmatization.

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Text Cleaning
def clean_text(text):
    text = re.sub(r'\d+', '', text)  # Remove digits
    text = re.sub(r'\W+', ' ', text)  # Remove special characters
    text = text.lower()  # Convert to lowercase
    text = text.strip()
    return text

# Clean the data
cleaned_data = clean_text(data)
print("Cleaned Data (First 500 characters):", cleaned_data[:500])  # Preview of cleaned text

# Tokenization and Stopword Removal
tokens = nltk.word_tokenize(cleaned_data)
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]

print("Tokens after Stopword Removal (First 20 tokens):", filtered_tokens[:20])  # Preview of tokens

# Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]

print("Lemmatized Tokens (First 20 tokens):", lemmatized_tokens[:20])  # Preview of lemmatized tokens
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Cleaned Data (First 500 characters): we first present our view of detection and correction of syntactic errors we then introduce a new c
Tokens after Stopword Removal (First 20 tokens): ['first', 'present', 'view', 'detection', 'correction', 'syntactic', 'errors', 'introdu
Lemmatized Tokens (First 20 tokens): ['first', 'present', 'view', 'detection', 'correction', 'syntactic', 'error', 'introduce', 'new', '
```

Step 3: Word Embedding

Using Word2Vec from the gensim library to create word embeddings.

```
from gensim.models import Word2Vec

# Create Word2Vec model
model = Word2Vec([lemmatized_tokens], vector_size=100, window=5, min_count=2, sg=1)
word_vectors = model.wv

# Example: Get vector for a specific word (adjust word as needed)
print("Vector for 'research':", word_vectors['research'] if 'research' in word_vectors else "Word not in vocabulary")
```

```
Vector for 'research': [-0.09597938  0.04817301  0.03520697 -0.18883356  0.00452412 -0.30815223
  0.10842972  0.3195855  -0.09427086 -0.06864997 -0.04312519 -0.1758261
 -0.14962809  0.04688559  0.02893239 -0.05368261 -0.02925871 -0.11721992
  0.02723616 -0.19980678  0.05839466  0.06323639  0.1160235  -0.12231019
 -0.07332232  0.05454352 -0.14621918 -0.07184091 -0.07285503 -0.04595394
  0.14506754  0.00372419  0.11108604 -0.13494326 -0.05047325  0.10980233
  0.13286512 -0.07249369 -0.16698632 -0.29481724  0.00372281 -0.17214891
  0.00435141 -0.02077201  0.1601517  -0.08331051 -0.1203436  -0.01680325
  0.14170131  0.13401279  0.14013405 -0.17045675 -0.0611458  -0.06087408
 -0.07678086  0.0385175   0.07323485  0.00218575 -0.19613388  0.05558421
  0.04570123  0.07914662 -0.05761265 -0.05554193 -0.13129601  0.18643789
  0.04180723  0.13608855 -0.09052119  0.17414398 -0.1673439   0.07500046
  0.14404681  0.00767872  0.19879793  0.05922469  0.05944034 -0.09374236
 -0.13121538 -0.00213671 -0.06244595  0.06857093 -0.08367636  0.19370253
  0.01648789  0.00504938 -0.04659484  0.13267127  0.14988738  0.07116204
  0.11580476  0.04536944  0.01668226 -0.00685782  0.29491463  0.08239824
  0.11483156 -0.13752232  0.02725036 -0.03755618]
```

Step 4: Encoding Techniques

**Bag of Words and One-Hot Encoding **

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelBinarizer

# Bag of Words Encoding
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform([' '.join(lemmatized_tokens)])

print("Bag of Words Encoding (First 10 words):", bow_matrix.toarray()[:20])  # Preview of Bag of Words

# One-Hot Encoding (limited sample for display purposes)
lb = LabelBinarizer()
one_hot_encoded = lb.fit_transform(lemmatized_tokens[:20])
print("One-Hot Encoding (First 20 words):", one_hot_encoded)
```

```
Bag of Words Encoding (First 10 words): [[1 1 1 ... 1 1 1]]
One-Hot Encoding (First 20 words): [[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
```

```
       [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
       [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
       [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
       [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
       [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Step 5: Parts of Speech (POS) Tagging

```
import spacy

# Sample lemmatized tokens (replace with your actual lemmatized tokens)
lemmatized_tokens = ["This", "is", "a", "sample", "sentence", "."]

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

# Increase the maximum length limit
nlp.max_length = len(' '.join(lemmatized_tokens)) + 100 # add a buffer

# Process the text in smaller chunks if still too large
chunk_size = 1000000  # Adjust this based on your available memory
pos_tags = []
for i in range(0, len(lemmatized_tokens), chunk_size):
    chunk = lemmatized_tokens[i : i + chunk_size]
    doc = nlp(' '.join(chunk))
    pos_tags.extend([(token.text, token.pos_) for token in doc])

# Print the first 20 POS tags
print("POS Tags (First 20 tokens):", pos_tags[:20])
```

```
POS Tags (First 20 tokens): [('This', 'PRON'), ('is', 'AUX'), ('a', 'DET'), ('sample', 'NOUN'), ('sentence', 'NOUN'), ('.', 'PUNCT')]
```

QUESTION NO 02:

1. Sentiment Analysis (using TextBlob)

```
# Task 1: Sentiment Analysis using TextBlob
from textblob import TextBlob

# TextBlob Sentiment Analysis
textblob_analysis = TextBlob(data)
textblob_sentiment = textblob_analysis.sentiment
print("TextBlob Sentiment Analysis Scores:")
print("Polarity:", textblob_sentiment.polarity)  # Range from -1 (negative) to 1 (positive)
print("Subjectivity:", textblob_sentiment.subjectivity)  # Range from 0 (objective) to 1 (subjective)
```

```
TextBlob Sentiment Analysis Scores:
Polarity: 0.09946322238259706
Subjectivity: 0.4222628006418276
```

```
# 2: Sentiment Analysis using VADER
```

2: Sentiment Analysis using VADER

```
subset_data = " ".join(df['abstract'].dropna().tolist()[:300])

# VADER Sentiment Analysis
analyzer = SentimentIntensityAnalyzer()
vader_scores = analyzer.polarity_scores(subset_data)
print("VADER Sentiment Analysis Scores for Subset:")
print(vader_scores)
```

```
VADER Sentiment Analysis Scores for Subset:
{'neg': 0.036, 'neu': 0.862, 'pos': 0.102, 'compound': 1.0}
```

Text Classification (Naive Bayes)

```
# Train the Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train_vectorized, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test_vectorized)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.3330429732868757
Classification Report:
                    precision    recall  f1-score   support

Computer Science       0.34      0.34      0.34      1160
     Mathematics       0.34      0.33      0.33      1166
         Physics       0.32      0.33      0.33      1118

        accuracy                           0.33      3444
       macro avg       0.33      0.33      0.33      3444
    weighted avg       0.33      0.33      0.33      3444
```