Name – BHAVIN R. KALAL

Enroll no – ADT23SOCM0306

Class – LY Msc [AIML]

---

1 :: import nltk

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
import re
# Download required nltk data (uncomment if not already downloaded)
# nltk.download('punkt')
# nltk.download('stopwords')
# Sample text
text = "Natural language processing allows us to process and analyze large amounts of textual data."
# Tokenization
tokens = word_tokenize(text)
print("Tokens:", tokens)
# Removing punctuation
tokens = [word for word in tokens if re.match(r'\w+', word)]
print("Tokens without punctuation:", tokens)
# Stop Word Removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print("Tokens without stop words:", filtered_tokens)

# Stemming
ps = PorterStemmer()
stemmed_tokens = [ps.stem(word) for word in filtered_tokens]
print("Stemmed Tokens:", stemmed_tokens)
# Frequency Analysis
```

```
frequency = Counter(stemmed_tokens)

print("Frequency Analysis:", frequency)
```

```
2 :: import nltk

from nltk.corpus import gutenberg, brown, reuters, wordnet

# Download the necessary corpora (uncomment if not already downloaded)

# nltk.download('gutenberg')

# nltk.download('brown')

# nltk.download('reuters')

# nltk.download('wordnet')

# nltk.download('omw-1.4')  # Optional: For WordNet multilingual data

# 1. Accessing the Gutenberg Corpus

print("Gutenberg Corpus Files:", gutenberg.fileids())

gutenberg_sample = gutenberg.raw('austen-emma.txt')[:500]  # Sample text from Emma by Jane Austen

print("\nSample from Gutenberg Corpus:\n", gutenberg_sample)

# 2. Accessing the Brown Corpus

print("\nCategories in Brown Corpus:", brown.categories())

brown_sample = brown.words(categories='news')[:50]  # Sample words from the news category in Brown Corpus

print("\nSample words from Brown Corpus (news category):\n", ' '.join(brown_sample))

# 3. Accessing the Reuters Corpus

print("\nCategories in Reuters Corpus:", reuters.categories())

reuters_sample = reuters.words(categories='crude')[:50]  # Sample words from the crude category in Reuters Corpus

print("\nSample words from Reuters Corpus (crude category):\n", ' '.join(reuters_sample))

# 4. Using WordNet for Lexical Resources

synset = wordnet.synsets('computer')[0]  # Get the first synset for "computer"

print("\nDefinition of 'computer':", synset.definition())

print("Examples of 'computer':", synset.examples())


# Exploring synonyms, antonyms, and hypernyms
```

```
synonyms = [lemma.name() for syn in wordnet.synsets('computer') for lemma in syn.lemmas()]
print("\nSynonyms of 'computer':", set(synonyms))

# Getting antonyms
antonyms = []
for syn in wordnet.synsets('good'):
    for lemma in syn.lemmas():
        if lemma.antonyms():
            antonyms.append(lemma.antonyms()[0].name())
print("\nAntonyms of 'good':", set(antonyms))
```

```
3 :: import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import string

# Download necessary NLTK data files (uncomment if not already downloaded)
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')

# Sample raw text
raw_text = "Hello! This is a sample text for NLP processing. It includes punctuation, numbers like 123, and stop words."

# 1. Convert text to lowercase
text = raw_text.lower()
print("Lowercase Text:", text)

# 2. Remove punctuation
```

```
text = re.sub(r'[^\w\s]', '', text)
print("Text without punctuation:", text)

# 3. Tokenization
tokens = word_tokenize(text)
print("Tokens:", tokens)

# 4. Remove stop words
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]
print("Tokens without stop words:", filtered_tokens)

# 5. Stemming
ps = PorterStemmer()
stemmed_tokens = [ps.stem(word) for word in filtered_tokens]
print("Stemmed Tokens:", stemmed_tokens)

# 6. Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
print("Lemmatized Tokens:", lemmatized_tokens)

# 7. Remove numbers
text_without_numbers = re.sub(r'\d+', '', ' '.join(lemmatized_tokens))
print("Text without numbers:", text_without_numbers)
```

```
4 :: import re
from typing import List, Tuple

def main():
    """Main function to execute the text analysis program."""
```

```
    # Sample text
    text = "This is an example sentence. Here is another one! Text processing is essential in NLP."

    # Preprocess the text
    cleaned_text = preprocess_text(text)

    # Perform analysis
    word_count = count_words(cleaned_text)
    sentence_count = count_sentences(text)
    avg_word_length = calculate_average_word_length(cleaned_text)

    # Display results
    display_results(word_count, sentence_count, avg_word_length)

def preprocess_text(text: str) -> str:
    """Cleans text by removing punctuation and converting it to lowercase."""
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    return text.lower()

def count_words(text: str) -> int:
    """Counts the number of words in the text."""
    words = text.split()
    return len(words)

def count_sentences(text: str) -> int:
    """Counts the number of sentences in the text."""
    sentences = re.split(r'[.!?]', text)
    sentences = [s for s in sentences if s.strip()]  # Remove empty sentences
    return len(sentences)

def calculate_average_word_length(text: str) -> float:
    """Calculates the average word length in the text."""
```

```
    words = text.split()
    total_length = sum(len(word) for word in words)
    return total_length / len(words) if words else 0

def display_results(word_count: int, sentence_count: int, avg_word_length: float):
    """Displays the results of the text analysis."""
    print("Text Analysis Results:")
    print(f"Word Count: {word_count}")
    print(f"Sentence Count: {sentence_count}")
    print(f"Average Word Length: {avg_word_length:.2f} characters")

# Run the main function
if _name_ == "_main_":
    main()
```

---

5 ::

```
import nltk
from nltk.tokenize import word_tokenize

# Download necessary NLTK data (uncomment if not already downloaded)
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')

def main():
    # Sample text
    sentence = "The quick brown fox jumps over the lazy dog."

    # Tokenize the sentence
    words = word_tokenize(sentence)
    print("Tokenized Words:", words)
```

```
    # Perform POS tagging
    tagged_words = nltk.pos_tag(words)
    print("\nPart-of-Speech Tags:")
    for word, tag in tagged_words:
        print(f"{word}: {tag}")

if __name__ == "__main__":
    main()
```

---

6 ::

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

# Sample data
texts = ["I love this movie", "This is a terrible movie", "Great plot and characters", "Worst movie ever"]
labels = ["positive", "negative", "positive", "negative"]

# Create a pipeline with a CountVectorizer and Naive Bayes classifier
model = make_pipeline(CountVectorizer(), MultinomialNB())

# Train the model
model.fit(texts, labels)

# Predict the category of a new sentence
new_text = "The movie was fantastic"
print("Predicted Category:", model.predict([new_text])[0])
```

---

7 ::

```
import spacy
```

```
# Load SpaCy English model
nlp = spacy.load("en_core_web_sm")

# Sample text
text = "Apple Inc. was founded by Steve Jobs in Cupertino, California in 1976."

# Process text and extract named entities
doc = nlp(text)
for ent in doc.ents:
    print(f"{ent.text} - {ent.label_}")
```

---

8 ::

```
import spacy

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

# Sample sentence
sentence = "The cat sat on the mat."

# Analyze sentence structure
doc = nlp(sentence)
for token in doc:
    print(f"{token.text} -> {token.dep_} (head: {token.head.text})")
```

---

9 ::

```
import nltk
from nltk import FeatureChartParser
```

```
# Define a feature-based grammar
grammar = nltk.CFG.fromstring("""
S -> NP VP
NP -> Det N | Det N PP
VP -> V NP | VP PP
PP -> P NP
Det -> 'the'
N -> 'dog' | 'park'
V -> 'chased'
P -> 'in'
""")

# Create a parser
parser = FeatureChartParser(grammar)

# Parse a sentence
sentence = "the dog chased the dog in the park".split()
for tree in parser.parse(sentence):
   print(tree)
```

---

10 ::

```
from transformers import pipeline

# Load sentiment analysis model from HuggingFace
classifier = pipeline("sentiment-analysis")

# Sample sentence
sentence = "The weather is lovely today!"

# Analyze sentiment
result = classifier(sentence)
```

```
print("Sentiment Analysis Result:", result)
```
