```
Name : Pranjal Nikalje Roll no :22573 Batch:A4

# 1. Extract Sample document and apply following document
preprocessing methods: Tokenization, POS
# Tagging, stop words removal, Stemming and Lemmatization. import
nltk
# Download => nltk==3.8.1 is the most stable version that works great
with the punkt tokenizer. from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfVectorizer
import math


# Download required NLTK data files
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

[nltk_data] Downloading package punkt to C:\Users\aarya
[nltk_data]     admane/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to C:\Users\aarya
[nltk_data]     admane/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\aarya admane/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-
to[nltk_data]       date!
[nltk_data] Downloading package wordnet to C:\Users\aarya
[nltk_data]     admane/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
# Sample documents
doc1 = "Natural Language Processing enables computers to understand
human language."
doc2 = "Machine learning is a part of artificial intelligence that
deals with algorithms."

# Tokenization
tokens1 = word_tokenize(doc1)
tokens2 = word_tokenize(doc2)
print(" ◇  Tokens in Doc1:",
tokens1)
```

```
◇ Tokens in Doc1: ['Natural', 'Language', 'Processing', 'enables',
'computers', 'to', 'understand', 'human', 'language', '.']
```

```
# POS Tagging
pos_tags1 = pos_tag(tokens1)
print(" ◇ POS Tags Doc1:",
pos_tags1)
```

```
◇ POS Tags Doc1: [('Natural', 'JJ'), ('Language', 'NNP'),
('Processing', 'NNP'), ('enables', 'VBZ'), ('computers', 'NNS'),
('to', 'TO'), ('understand', 'VB'), ('human', 'JJ'), ('language',
'NN'), ('.', '.')]
```

```
# Stop Words Removal
stop_words = set(stopwords.words('english'))
filtered_tokens1 = [word for word in tokens1 if word.lower() not in
stop_words and word.isalpha()]
print(" ◇ Tokens after Stop Words Removal (Doc1):",
filtered_tokens1)
```

```
◇ Tokens after Stop Words Removal (Doc1): ['Natural', 'Language',
'Processing', 'enables', 'computers', 'understand', 'human',
'language']
```

```
# Stemming
stemmer = PorterStemmer()
stemmed_tokens1 = [stemmer.stem(word) for word in filtered_tokens1]
print(" ◇ Stemmed Tokens (Doc1):", stemmed_tokens1)
```

```
◇ Stemmed Tokens (Doc1): ['natur', 'languag', 'process', 'enabl',
'comput', 'understand', 'human', 'languag']
```

```
# Lemmatization with POS tag mapping
lemmatizer = WordNetLemmatizer()

# Map POS tags for lemmatization def
get_wordnet_pos(treebank_tag):
if treebank_tag.startswith('J'):
        return wordnet.ADJ     elif
treebank_tag.startswith('V'):
        return wordnet.VERB     elif
treebank_tag.startswith('N'):
        return wordnet.NOUN     elif
treebank_tag.startswith('R'):          return
wordnet.ADV     else:          return
wordnet.NOUN  # default to noun

lemmatized_tokens1 = [
    lemmatizer.lemmatize(word, get_wordnet_pos(tag))
    for word, tag in pos_tags1 if word.lower() not in stop_words and
word.isalpha()
```

```
]
print(" ◇  Lemmatized Tokens (Doc1):", lemmatized_tokens1)
```

```
 ◇  Lemmatized Tokens (Doc1): ['Natural', 'Language', 'Processing',
'enable', 'computer', 'understand', 'human', 'language']
```

```
# TF-IDF using Scikit-learn

# 2. Create representation of documents by calculating Term Frequency
and Inverse Document Frequency.----------------------

# Combine both documents
documents = [doc1, doc2]


# Vectorizer
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(documents)

# Display TF-IDF values
feature_names = vectorizer.get_feature_names_out() for doc_index, doc
in enumerate(documents):     print(f"\n ◇  TF-IDF for Document
{doc_index + 1}:")     for word_index in
tfidf_matrix[doc_index].nonzero()[1]:
print(f"{feature_names[word_index]}: {tfidf_matrix[doc_index,
word_index]:.4f}")
```

```
 ◇  TF-IDF for Document
1: natural: 0.3162
language: 0.6325
processing: 0.3162
enables: 0.3162
computers: 0.3162
understand: 0.3162
human: 0.3162

 ◇  TF-IDF for Document
2: machine: 0.4082
learning: 0.4082
artificial: 0.4082
intelligence: 0.4082
deals: 0.4082
algorithms: 0.4082
```