EX.NO:2

Date:23/12/2024

Stemming, Lemmatization, and Stopwords

AIM:

Perform text processing tasks on a given document containing at least 200 words.

1. Reading and Tokenizing
2. Stemming:
3. Lemmatization:
4. Stop Words Removal

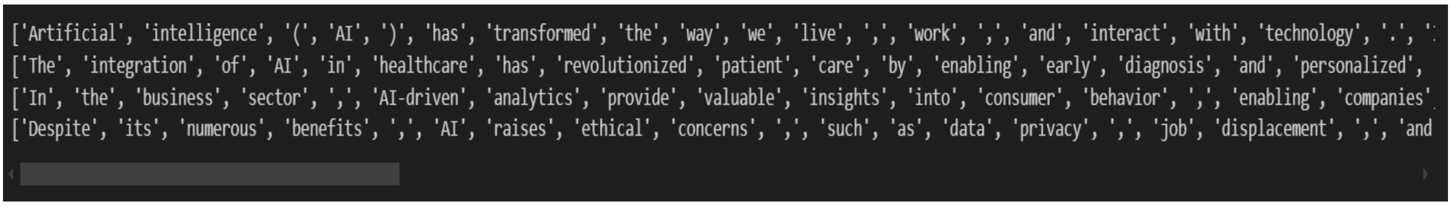
PROCEDURE:

1. **Reading and Tokenizing:**
   - Read the document line by line.
   - Tokenize each line into words.
2. **Stemming:**
   - Apply stemming to the tokenized words.
   - Print or save the stemmed words to a file.
3. **Lemmatization:**
   - Perform lemmatization for the following sample sentence:
     "For a sentence containing words like visit, visitor, visiting, visited."
   - Output the lemmatized forms (print or save to a file).
4. **Stop Words Removal:**
   - Identify and remove stop words from the document using a stop words list containing at least 10 words.
   - Display the filtered text without stop words.

CODE :

1. ***Tokenize the text :***

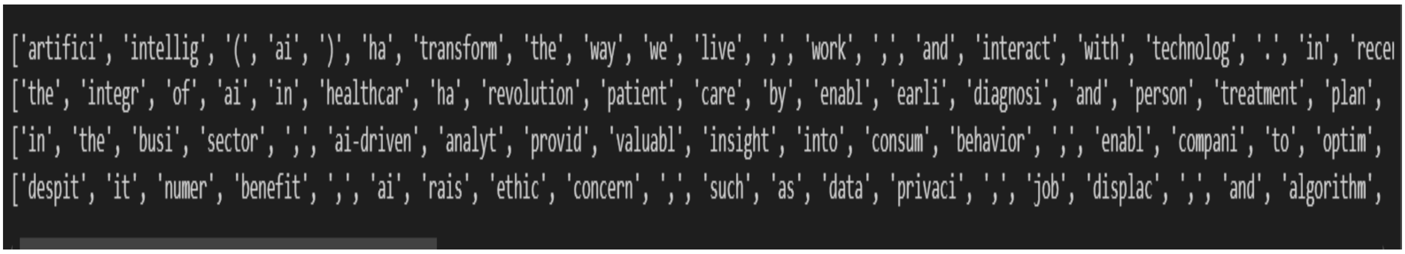
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

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

file_path="sampledoc.txt"
with open(file_path ,'r') as file:
   document=file.read()

lines=document.split('\n')
token_list=[]

for line in lines:
   tokens=word_tokenize(line)
   token_list.append(tokens)
print(token_list)
print()
```



```
['Artificial', 'intelligence', '(', 'AI', ')', 'has', 'transformed', 'the', 'way', 'we', 'live', ',', 'work', ',', 'and', 'interact', 'with', 'technology', '.', '
['The', 'integration', 'of', 'AI', 'in', 'healthcare', 'has', 'revolutionized', 'patient', 'care', 'by', 'enabling', 'early', 'diagnosis', 'and', 'personalized',
['In', 'the', 'business', 'sector', ',', 'AI-driven', 'analytics', 'provide', 'valuable', 'insights', 'into', 'consumer', 'behavior', ',', 'enabling', 'companies'
['Despite', 'its', 'numerous', 'benefits', ',', 'AI', 'raises', 'ethical', 'concerns', ',', 'such', 'as', 'data', 'privacy', ',', 'job', 'displacement', ',', 'and
```

2. ***Apply stemming to the tokenized words:***

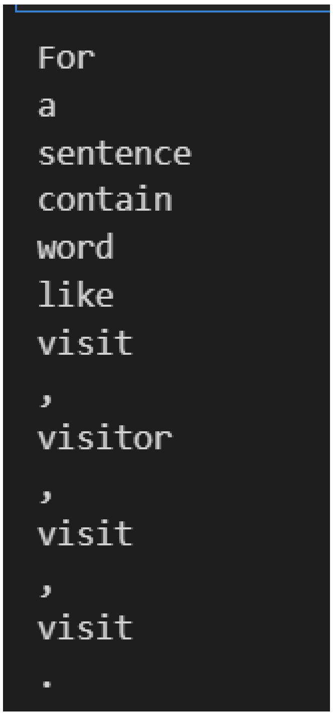
```
stemmer= PorterStemmer()
for i , tokens in enumerate(token_list):
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
    print(stemmed_tokens)
```



```
['artifici', 'intellig', '(', 'ai', ')', 'ha', 'transform', 'the', 'way', 'we', 'live', ',', 'work', ',', 'and', 'interact', 'with', 'technolog', '.', 'in', 'recei
['the', 'integr', 'of', 'ai', 'in', 'healthcar', 'ha', 'revolution', 'patient', 'care', 'by', 'enabl', 'earli', 'diagnosi', 'and', 'person', 'treatment', 'plan',
['in', 'the', 'busi', 'sector', ',', 'ai-driven', 'analyt', 'provid', 'valuabl', 'insight', 'into', 'consum', 'behavior', ',', 'enabl', 'compani', 'to', 'optim',
['despit', 'it', 'numer', 'benefit', ',', 'ai', 'rais', 'ethic', 'concern', ',', 'such', 'as', 'data', 'privaci', ',', 'job', 'displac', ',', 'and', 'algorithm',
```

3. **Perform lemmatization for the following sample sentence:**
"For a sentence containing words like visit, visitor, visiting, visited."

```
from nltk.stem import WordNetLemmatizer
#nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag
from nltk.corpus import wordnet

line="For a sentence containing words like visit, visitor, visiting, visited."
my_tokens=word_tokenize(line)
pos=pos_tag(my_tokens)

# create an object of class WordNetLemmatizer
lemmatizer = WordNetLemmatizer()


for i in my_tokens:
    print(lemmatizer.lemmatize(i,wordnet.VERB))
```



```
For
a
sentence
contain
word
like
visit
,
visitor
,
visit
,
visit
.
```

4. ***Stop Words Removal:***

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stopwords_default = stopwords.words('english')

word_token=word_tokenize(document)
clean_text = [word for word in word_token if not word in stopwords_default]
' '.join(clean_text)
```

EX.No:3 **Word Tokenizer and Analysis**
Date:28/12/2024

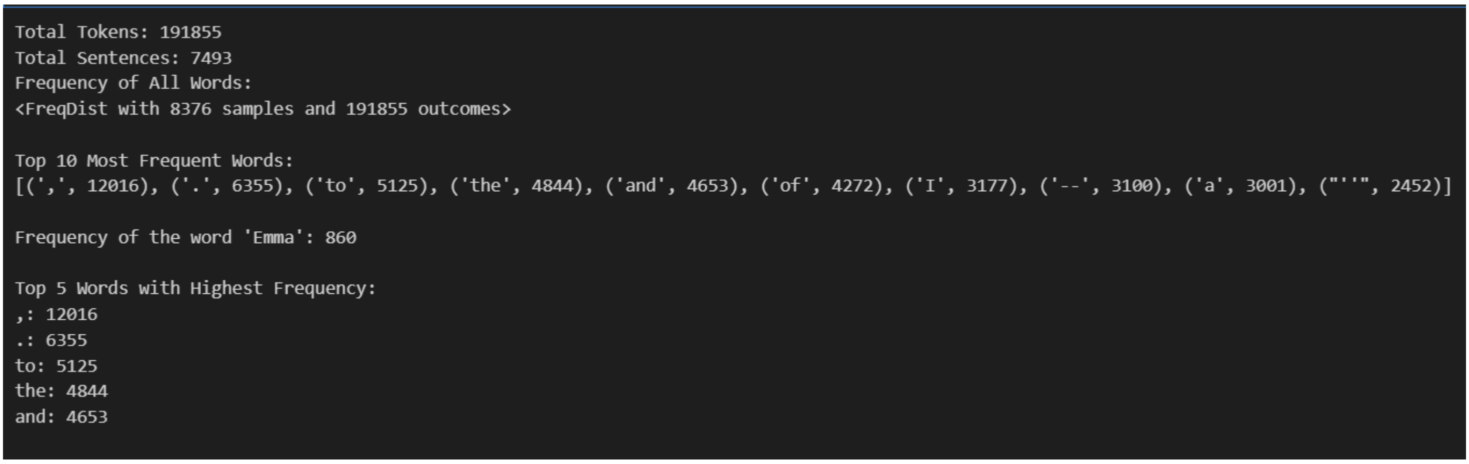
AIM:
Program to tokenize text from a small corpus, count tokens and sentences, analyze word frequency (using nltk.FreqDist), find occurrences of specific words, display the top 5 frequent words, and visualize word dispersion using matplotlib.

PROCEDURE:

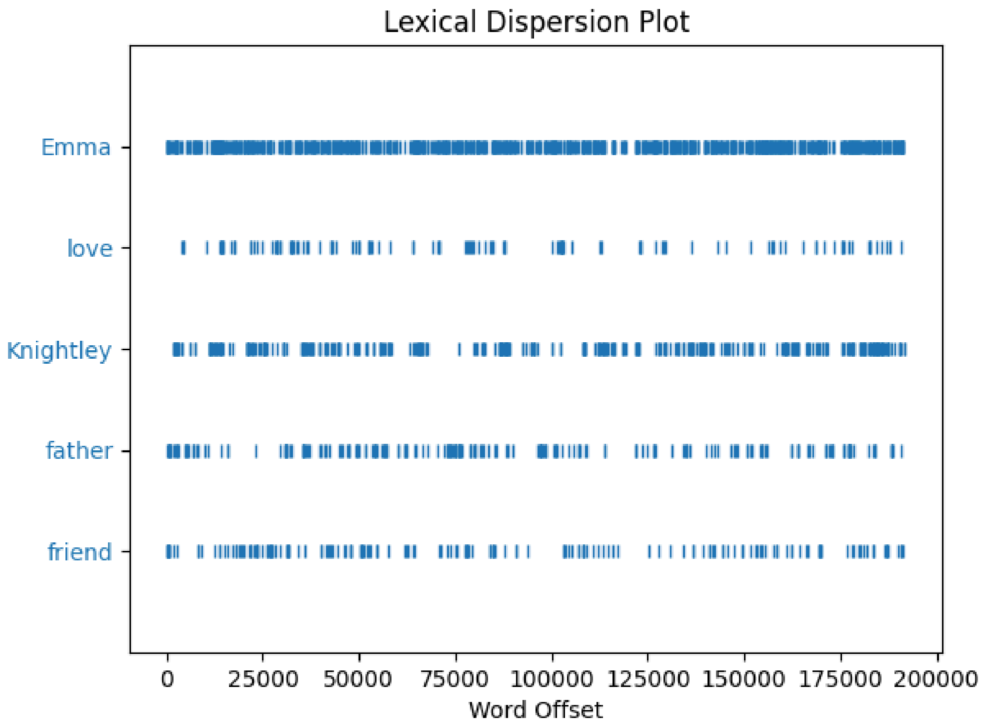
1. Take a small corpus from the NLTK library (e.g., nltk.corpus.gutenberg or similar) or your own document
2. Tokenize the text and find the total number of tokens.
3. Count and display the total number of sentences in the corpus.
4. Use frequency distribution function to find the occurrence of words from (nltk.FreqDist)
   (i)Display the frequency of all words.
   (ii)Find and display the frequency of a specific word.
5. Find the occurrence of particular word
6. Display the top 5 words with the highest frequency in the document.
7. Create and show a dispersion plot for the above(matplotlib/ or any)

CODE:

```
import nltk
from nltk.corpus import gutenberg
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt


nltk.download('gutenberg')
nltk.download('punkt')

corpus_text = gutenberg.raw('austen-emma.txt')

# 2. Tokenize the text into words and sentences
word_tokens = word_tokenize(corpus_text)
sentence_tokens = sent_tokenize(corpus_text)

# 3. Calculate total number of tokens and sentences
total_tokens = len(word_tokens)
total_sentences = len(sentence_tokens)

# 4. Use FreqDist to find word occurrences
freq_dist = FreqDist(word_tokens)

# 5. Display results
print(f"Total Tokens: {total_tokens}")
print(f"Total Sentences: {total_sentences}")
print("Frequency of All Words:")
print(freq_dist)

# (i) Display frequency of all words (optional: top 10)
print("\nTop 10 Most Frequent Words:")
print(freq_dist.most_common(10))

# (ii) Frequency of a specific word (e.g., "Emma")
specific_word = "Emma"
print(f"\nFrequency of the word '{specific_word}': {freq_dist[specific_word]}")

# (iii) Display the top 5 words with the highest frequency
top_5_words = freq_dist.most_common(5)
print("\nTop 5 Words with Highest Frequency:")
for word, freq in top_5_words:
    print(f"{word}: {freq}")

# 6. Create and show a dispersion plot for selected words
selected_words = ["Emma", "love", "Knightley", "father", "friend"]
text_tokens = nltk.Text(word_tokens)  # Convert tokens to NLTK Text object

print("\nGenerating Dispersion Plot...")
plt.figure(figsize=(10, 6))
text_tokens.dispersion_plot(selected_words)
```



```
Total Tokens: 191855
Total Sentences: 7493
Frequency of All Words:
<FreqDist with 8376 samples and 191855 outcomes>

Top 10 Most Frequent Words:
[(',', 12016), ('.', 6355), ('to', 5125), ('the', 4844), ('and', 4653), ('of', 4272), ('I', 3177), ('--', 3100), ('a', 3001), ("''", 2452)]

Frequency of the word 'Emma': 860

Top 5 Words with Highest Frequency:
,: 12016
.: 6355
to: 5125
the: 4844
and: 4653
```


Lexical Dispersion Plot
Emma
love
Knightley
father
friend
0
25000
50000
75000
100000
125000
150000
175000
200000
Word Offset

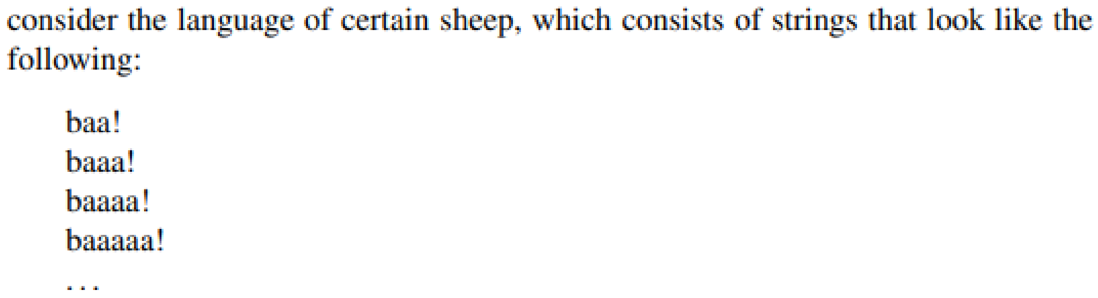
Ex.no:4 **Regular Expressions in NLTK**
Date:24/12/2024

AIM:

Program to demonstrate regex operations on a sample NLTK corpus, including: matching hyphenated patterns, using the caret (^) for line beginnings, applying the ? operator for optional preceding words, exploring Kleene * and + operators, using the dot (.) for matching characters, the pipe (|) for alternation, detecting word boundaries (\b), non-word boundaries (\B), and interpreting specific regex patterns like /[^a-zA-Z][tT]he[^a-zA-Z]/.

PROCEDURE:

(1) Use of Hyphen [2-5] and [b-f]
(2) ^ caret symbol for a sample text document
(3) ? operator for a set of Preceding words
(4) Use of Kleene * operator and display like below



consider the language of certain sheep, which consists of strings that look like the following:

baa!
baaa!
baaaa!
baaaaa!
. . .

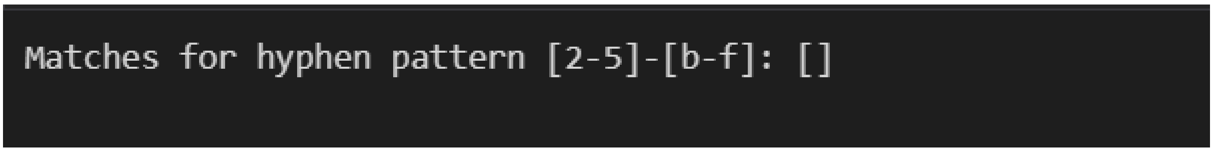
(5) Use of Kleene + operator
(6) . (Dot) operator for set of words
(7) | pipe symbol
(8) Word boundary
(9) Non -word Boundary
(10) /[ˆa-zA-Z][tT]he[ˆa-zA-Z]/ - what does this represent

CODE:

```
nltk.download('gutenberg')
# Load a sample text
text = gutenberg.raw('austen-sense.txt')
# Split into lines
lines = text.splitlines()
```

1. *Use of hyphen*

```
pattern_hyphen = r'\b[2-5]-[b-f]\b'
matches_hyphen = re.findall(pattern_hyphen, text)
print("Matches for hyphen pattern [2-5]-[b-f]:", matches_hyphen)
```

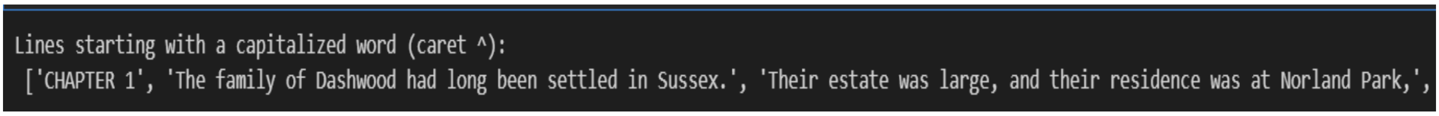


```
Matches for hyphen pattern [2-5]-[b-f]: []
```

2. *Use of caret symbol*

```
# Matches lines starting with a capitalized word (for example).
pattern_caret = r'^[A-Z][a-z]*'
matches_caret = [line for line in lines if re.match(pattern_caret, line)]
```

```
print("Lines starting with a capitalized word (caret ^):\n", matches_caret[:5])  # Show first 5 matches
```



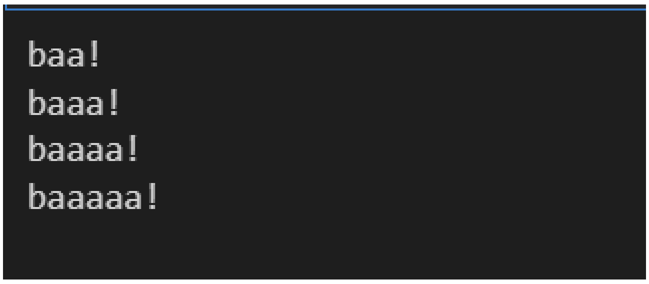
```
Lines starting with a capitalized word (caret ^):
 ['CHAPTER 1', 'The family of Dashwood had long been settled in Sussex.', 'Their estate was large, and their residence was at Norland Park,',
```

3. Klenee *

```
# Base string
base = "b" + "a" * 2 + "!"  # Minimum "baa!"

# Using Kleene star to generate variations
for i in range(4):  # Kleene star ensures 0 or more 'a's added
    print("b" + "a" * total_bounda"*flex flow.text!!
```

4. Klenee +

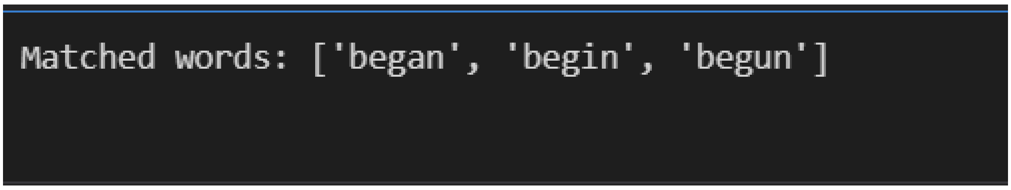
```
# Generate strings using the Kleene + concept (one or more repetitions)
for i in range(2, 6):  # Start from 2 'a's to match "baa!" and go up to 5 'a's
    print("b" + "a" * i + "!")
```



```
baa!
baaa!
baaaa!
baaaaa!
```

5. (.) Operator

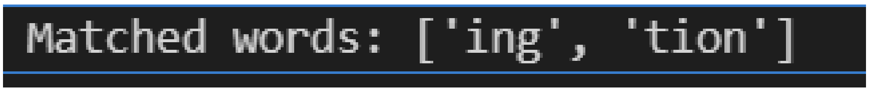
```
pattern = r'\bbeg.n\b'

# Find all matches in the text
matches = re.findall(pattern, text)

# Print unique matches
print("Matched words:", sorted(set(matches)))
```



```
Matched words: ['began', 'begin', 'begun']
```

6. *Pipe symbol (|)*

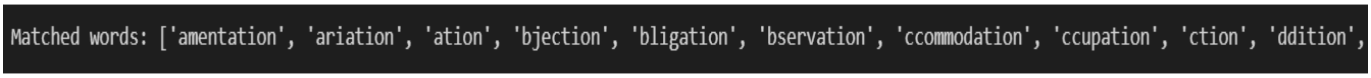
```
pattern = r'\b[a-z]*?(tion|ing)\b'

# Find all matches in the text
matches = re.findall(pattern, text)

# Print unique matches
print("Matched words:", sorted(set(matches)))
```



```
Matched words: ['ing', 'tion']
```

7. *Non-word Boundary*

```
pattern = r'\B\w*tion\B'

# Find all matches in the text
matches = re.findall(pattern, text)

# Print unique matches
print("Matched words:", sorted(set(matches)))
```



```
Matched words: ['amentation', 'ariation', 'ation', 'bjection', 'bligation', 'bservation', 'ccommodation', 'ccupation', 'ction', 'ddition',
```