EX.NO.: 05 **NLTK Brown Corpus**
DATE: 17.01.2025

To explore and analyze the Brown Corpus using the NLTK library, including examining its structure, categories, and specific functionalities like word, sentence, and file analysis.

PROCEDURE:

1. Set Up and Import Required Libraries
2. Explore the Corpus Structure
   > brown.categories()
   > brown.fileids(category)
3. Choose a Section for Analysis
   > brown.words(categories=chosen_section) and brown.sents(categories=chosen_section)
4. Perform Statistical Analysis
   > FreqDist
5. Display Section-Specific Insights
   > brown.words(fileid)
6. Explore Additional Functionalities of the Brown Corpus
   > fileids(), categories(), raw(), words(), and sents()
7. Corpus File Operations
   > abspath(), encoding(), and open()
8. Read Corpus Metadata
   > brown.readme()

CODE AND OUTPUT

```
import nltk
from nltk.corpus import brown
from nltk import FreqDist, word_tokenize, sent_tokenize

# Ensure the Brown Corpus is downloaded
nltk.download('brown')

# Explore the available categories (sections) in the Brown Corpus
categories = brown.categories()
print("Categories in Brown Corpus:")
print(categories)
```

```
Categories in Brown Corpus:
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews', 'romance',
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\Hema\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

```
# Categorize documents for each section in the Brown Corpus
category_documents = {category: brown.fileids(category) for category in categories}
print("\nDocuments Categorized by Sections:")
for category, docs in category_documents.items():
    print(f"{category}: {len(docs)} documents")

# Choose a section (e.g., 'news') for further analysis
chosen_section = 'humor'

# Words and sentences in the chosen section
section_words = brown.words(categories=chosen_section)
section_sentences = brown.sents(categories=chosen_section)
```

```
# Count the number of words and sentences
num_words = len(section_words)
num_sentences = len(section_sentences)

# Count modal verbs in the chosen section
modals = ['can', 'could', 'may', 'might', 'must', 'shall', 'should', 'will', 'would']
modal_freq = FreqDist(word.lower() for word in section_words if word.lower() in modals)

# Count 'wh' words in the chosen section
wh_words = ['what', 'why', 'whom', 'who', 'which', 'when', 'where', 'whose']
wh_word_freq = FreqDist(word.lower() for word in section_words if word.lower() in
wh_words)

# Display details of the chosen section
print(f"\nAnalysis of '{chosen_section}' Section:")
print(f"Number of words: {num_words}")
print(f"Number of sentences: {num_sentences}")
print(f"Modal verb frequencies: {dict(modal_freq)}")
print(f"'Wh' word frequencies: {dict(wh_word_freq)}")
```

```
Documents Categorized by Sections:
adventure: 29 documents
belles_lettres: 75 documents
editorial: 27 documents
fiction: 29 documents
government: 30 documents
hobbies: 36 documents
humor: 9 documents
learned: 80 documents
lore: 48 documents
mystery: 24 documents
news: 44 documents
religion: 17 documents
reviews: 17 documents
romance: 29 documents
science_fiction: 6 documents

Analysis of 'humor' Section:
Number of words: 21695
Number of sentences: 1053
Modal verb frequencies: {'might': 8, 'would': 56, 'can': 17, 'could': 33, 'may': 8, 'should': 7, 'will': 13, 'must': 9, 'shall': 2}
'Wh' word frequencies: {'why': 13, 'who': 49, 'which': 62, 'when': 62, 'what': 46, 'where': 16, 'whose': 8, 'whom': 4}
```

```
# Explore additional corpus functionalities
# Displaying a sample file's content
sample_file = category_documents[chosen_section][0]
sample_content = brown.words(sample_file)
print(f"\nContent of the file '{sample_file}' in '{chosen_section}' section (first 100
words):")
print(" ".join(sample_content[:100]))

# Displaying categories, genres, and more
print("\nAdditional Details:")
print(f"Total categories: {len(categories)}")
print(f"First 5 categories: {categories[:5]}")
print(f"Sample documents in '{chosen_section}' section:
{category_documents[chosen_section][:5]}")
```

```
Content of the file 'cr01' in 'humor' section (first 100 words):
It was among these that Hinkle identified a photograph of Barco ! ! For it seems that Barco , fancying himself a ladies' man ( and why not , after seven marriag

Additional Details:
Total categories: 15
First 5 categories: ['adventure', 'belles_lettres', 'editorial', 'fiction', 'government']
Sample documents in 'humor' section: ['cr01', 'cr02', 'cr03', 'cr04', 'cr05']
```

```python
import nltk
from nltk.corpus import brown

# Ensure the Brown Corpus is downloaded
nltk.download('brown')

# Example and Description
print("\nExploring Brown Corpus Functionality:\n")

# fileids()
print("fileids():")
file_ids = brown.fileids()
print(f"Total files: {len(file_ids)}")
print(f"Sample file IDs: {file_ids[:5]}\n")
```

```
Exploring Brown Corpus Functionality:

fileids():
Total files: 500
Sample file IDs: ['ca01', 'ca02', 'ca03', 'ca04', 'ca05']

[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\Hema\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

```python
# fileids([categories])
print("fileids([categories]):")
print(f"Files in 'humor' category: {brown.fileids(categories=['humor'])[:5]}\n")

# categories()
print("categories():")
categories = brown.categories()
print(f"Total categories: {len(categories)}")
print(f"Categories: {categories}\n")

# categories([fileids])
print("categories([fileids]):")
sample_file = file_ids[0]
print(f"Category for file '{sample_file}':
{brown.categories(fileids=[sample_file])}\n")
```

```
fileids([categories]):
Files in 'humor' category: ['cr01', 'cr02', 'cr03', 'cr04', 'cr05']

categories():
Total categories: 15
Categories: ['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews'

categories([fileids]):
Category for file 'ca01': ['news']
```

```python
# raw()
print("raw():")
print(f"Total raw content length: {len(brown.raw())} characters\n")
```

```
# raw(fileids=[f1,f2,f3])
print("raw(fileids=[f1,f2,f3]):")
print(f"Raw content of '{sample_file}': {brown.raw(fileids=[sample_file])[:200]}...\n")

# raw(categories=[c1,c2])
print("raw(categories=[c1,c2]):")
print(f"Raw content of 'humor' category: {brown.raw(categories=['humor'])[:200]}...\n")
```

```
raw():
Total raw content length: 9964284 characters

raw(fileids=[f1,f2,f3]):
Raw content of 'ca01':

        The/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd Friday/nr an/at investigation/nn of/in Atlanta's/np$ recent/jj primary/nn election/nn pro

raw(categories=[c1,c2]):
Raw content of 'humor' category: It/pps was/bedz among/in these/dts that/cs Hinkle/np identified/vbd a/at photograph/nn of/in Barco/np !/. !/.
For/cs it/pps seems/vbz that/cs Barco/np ,/, fancying/vbg himself/ppl a/at ladies'/nns$ ma...
```

```
# words()
print("words():")
print(f"Total words in corpus: {len(brown.words())}")
print(f"First 10 words: {brown.words()[:10]}\n")

# words(fileids=[f1,f2,f3])
print("words(fileids=[f1,f2,f3]):")
print(f"Words in '{sample_file}': {brown.words(fileids=[sample_file])[:10]}\n")

# words(categories=[c1,c2])
print("words(categories=[c1,c2]):")
print(f"Words in 'humor' category: {brown.words(categories=['humor'])[:10]}\n")
```

```
words():
Total words in corpus: 1161192
First 10 words: ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of']

words(fileids=[f1,f2,f3]):
Words in 'ca01': ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of']

words(categories=[c1,c2]):
Words in 'humor' category: ['It', 'was', 'among', 'these', 'that', 'Hinkle', 'identified', 'a', 'photograph', 'of']
```

```
# sents()
print("sents():")
print(f"Total sentences in corpus: {len(brown.sents())}")
print(f"First sentence: {' '.join(brown.sents()[0])}\n")

# sents(fileids=[f1,f2,f3])
print("sents(fileids=[f1,f2,f3]):")
print(f"First sentence in '{sample_file}': {'
'.join(brown.sents(fileids=[sample_file])[0])}\n")

# sents(categories=[c1,c2])
print("sents(categories=[c1,c2]):")
print(f"First sentence in 'humor' category: {'
'.join(brown.sents(categories=['humor'])[0])}\n")
```

```
sents():
Total sentences in corpus: 57340
First sentence: The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that any irregularities

sents(fileids=[f1,f2,f3]):
First sentence in 'ca01': The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that any irregu

sents(categories=[c1,c2]):
First sentence in 'humor' category: It was among these that Hinkle identified a photograph of Barco ! !
```

```
# abspath(fileid)
print("abspath(fileid):")
print(f"Absolute path of '{sample_file}': {brown.abspath(sample_file)}\n")

# encoding(fileid)
print("encoding(fileid):")
print(f"Encoding of '{sample_file}': {brown.encoding(sample_file)}\n")

# open(fileid)
print("open(fileid):")
with brown.open(sample_file) as f:
    print(f"First 200 characters from '{sample_file}': {f.read(200)}...\n")
```

```
abspath(fileid):
Absolute path of 'ca01': C:\Users\Hema\AppData\Roaming\nltk_data\corpora\brown\ca01

encoding(fileid):
Encoding of 'ca01': ascii

open(fileid):
First 200 characters from 'ca01':

        The/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd Friday/nr an/at investigation/nn of/in Atlanta's/np$ recent/jj primary/nn election/nn pro
```

```
# root()
print("root():")
print(f"Root path of Brown Corpus: {brown.root}\n")

# readme()
print("readme():")
print("Contents of README file:")
print(brown.readme()[:500])
```

```
root():
Root path of Brown Corpus: C:\Users\Hema\AppData\Roaming\nltk_data\corpora\brown

readme():
Contents of README file:
BROWN CORPUS

A Standard Corpus of Present-Day Edited American
English, for use with Digital Computers.

by W. N. Francis and H. Kucera (1964)
Department of Linguistics, Brown University
Providence, Rhode Island, USA

Revised 1971, Revised and Amplified 1979

http://www.hit.uib.no/icame/brown/bcm.html

Distributed with the permission of the copyright holder,
redistribution permitted.
```

EX.NO.: 06 **Tokenizing Text**
DATE: 17.01.2025

To process Shakespeare's text corpus using tokenization, Byte Pair Encoding (BPE), and stemming.

PROCEDURE:

1. Import Required Libraries
   > re, collections, TreebankWordTokenizer, PorterStemmer
2. Implement Byte Pair Encoding (BPE)
3. Prepare Tokens for BPE
4. Apply BPE
   > byte_pair_encoding()
5. Perform Stemming
   > PorterStemmer
6. Display Sample Outputs

CODE AND OUTPUT

```
import re
import collections
from nltk.tokenize import word_tokenize, TreebankWordTokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import gutenberg
import nltk

# Ensure necessary NLTK data is downloaded
nltk.download("gutenberg")
nltk.download("punkt")
# Load the Shakespeare text corpus from NLTK's Gutenberg corpus
shakespeare_text = gutenberg.raw('shakespeare-macbeth.txt')
# Tokenize the text using the Penn Treebank tokenizer
treebank_tokenizer = TreebankWordTokenizer()
tokens = treebank_tokenizer.tokenize(shakespeare_text)
# Define a Byte Pair Encoding (BPE) implementation
def byte_pair_encoding(tokens, num_merges):
    """Implements Byte Pair Encoding tokenization."""
    vocab = collections.Counter(tokens)

    # Create a mapping of token to character pairs
    def get_stats(vocab):
        pairs = collections.defaultdict(int)
        for word, freq in vocab.items():
            symbols = word.split()
            for i in range(len(symbols) - 1):
                pairs[symbols[i], symbols[i + 1]] += freq
        return pairs

    # Merge the most frequent pair
    def merge_vocab(pair, vocab):
        new_vocab = {}
        bigram = ' '.join(pair)
        replacement = ''.join(pair)
```

```
        for word in vocab:
            new_word = word.replace(bigram, replacement)
            new_vocab[new_word] = vocab[word]
        return new_vocab

    # Apply merges
    for _ in range(num_merges):
        pairs = get_stats(vocab)
        if not pairs:
            break
        best_pair = max(pairs, key=pairs.get)
        vocab = merge_vocab(best_pair, vocab)

    return vocab

# Prepare tokens for BPE by joining characters with spaces
bpe_tokens = [' '.join(token) for token in tokens]

# Apply Byte Pair Encoding with a specified number of merges (e.g., 10)
bpe_vocab = byte_pair_encoding(bpe_tokens, num_merges=10)

# Perform stemming using Porter Stemmer
porter_stemmer = PorterStemmer()
stemmed_tokens = [porter_stemmer.stem(token) for token in tokens]

# Print sample outputs
print("Original Tokens (Sample):", tokens[:10])
print("BPE Vocabulary (Sample):", list(bpe_vocab.items())[:10])
print("Stemmed Tokens (Sample):", stemmed_tokens[:10])
```

```
Original Tokens (Sample): ['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', ']']
BPE Vocabulary (Sample): [('[', 4), ('T h e', 118), ('T r a g e d i e', 1), ('o f', 314), ('M a c b e th', 53), ('b y', 36), ('W i l l i a m', 1), ('S h a k es p
Stemmed Tokens (Sample): ['[', 'the', 'tragedi', 'of', 'macbeth', 'by', 'william', 'shakespear', '1603', ']']
```

EX.NO.: 07 **FINDING PATTERNS AND SUBSTITUTIONS**
DATE: 17.01.2025

to create a simple, ELIZA-like chatbot that responds to user input with predefined, contextually appropriate responses. It uses regular expressions to apply substitutions to certain patterns in user input and provides responses related to technology and innovation.

PROCEDURE:

1. Define Substitution Rules
2. Define Generic Responses
3. Substitute Rules in User Input
4. Topic Extraction:
5. Select a Response
6. Chatbot Interaction Loop

CODE AND OUTPUT

```
import re

# Define substitution rules
substitutions = [
    (r'.* I\'M (depressed|sad) .*', r'I AM SORRY TO HEAR YOU ARE \1'),
    (r'.* I AM (depressed|sad) .*', r'WHY DO YOU THINK YOU ARE \1'),
    (r'.* all .*', r'IN WHAT WAY'),
    (r'.* always .*', r'CAN YOU THINK OF A SPECIFIC EXAMPLE')
]

# Define generic responses
responses = [
    "Why do you feel that way about {topic}?",
    "Can you elaborate more on your thoughts about {topic}?",
    "That's interesting. How does {topic} impact your life?",
    "Do you think {topic} is changing the world?",
    "What do you think about the future of {topic}?"
]

# Function to apply substitutions to user input
def apply_substitutions(input_text):
    for pattern, replacement in substitutions:
        if re.match(pattern, input_text, flags=re.IGNORECASE):
            return re.sub(pattern, replacement, input_text, flags=re.IGNORECASE)
    return input_text

# ELIZA-like response generator
def eliza_response(user_input):
    substituted_input = apply_substitutions(user_input)
    topic_match = re.search(r'\b(\w+)\b', substituted_input)
    topic = topic_match.group(1) if topic_match else "it"
    response_template = responses[len(topic) % len(responses)]  # Rotate responses
    return response_template.format(topic=topic)

# Main chatbot loop
```

```python
def chatbot():
    print("Welcome to the ELIZA-like Chatbot about Technology and Innovation!")
    print("Type 'exit' to end the conversation.\n")

    while True:
        user_input = input("You: ")
        if user_input.lower() == 'exit':
            print("Chatbot: Goodbye! Have a great day!")
            break
        response = eliza_response(user_input)
        print(f"Chatbot: {response}")

# Run the chatbot
if __name__ == "__main__":
    chatbot()
```

```
Welcome to the ELIZA-like Chatbot about Technology and Innovation!
Type 'exit' to end the conversation.

Chatbot: That's interesting. How does it impact your life?
Chatbot: Do you think The is changing the world?
Chatbot: Do you think All is changing the world?
Chatbot: That's interesting. How does no impact your life?
```