# Generative AI and its Applications

## Lab 1 Manual

## Hands-On in Basic NLP Applications Combining Web Scraping and Language Analysis Techniques

In this laboratory exercise, you will have hands-on experience on NLP applications, combining web scraping, and basic language analysis techniques.

## Learning Objectives

Upon completion of this laboratory exercise, students will be able to:

1. Implement web scraping techniques to gather textual data from online sources
2. Apply fundamental NLP concepts including tokenization, stop word removal, and stemming
3. Utilize NLP libraries (NLTK and SpaCy) for text pre-processing.

## Web Scraping and the Dataset

Web scraping is an automated process used to extract information from websites by simulating human browsing behavior. The Associated Press (AP) is a non-profit news organization known for its unbiased and comprehensive news coverage across various topics worldwide.

We have chosen the Associated Press for this activity because, as a non-profit organization, they permit web scraping for educational purposes, providing an excellent opportunity to learn about data extraction ethically. In this exercise, students will scrape a few selected news articles from the AP website, focusing on specific topics, to gain hands-on experience in data collection and processing for NLP applications.

## Prerequisites

### Technical Requirements

1. Python 3.8 or higher. (Preferably Python 3:12)
2. Code editor (VS Code, PyCharm, or similar).
3. Internet connection for web scraping and package installation.

# Generative AI and its Applications

### Required Python Packages

1. requests: For handling HTTP requests
2. beautifulsoup4: For HTML parsing
3. nltk: For basic NLP operations
4. spacy: For advanced NLP processing
5. collections: For frequency analysis (built-in)

## Provided Components

- Web scraping functionality using requests and BeautifulSoup. This will provide the scraped data in .txt files.

- Basic setup for NLTK and SpaCy.

- Example code for stemming, POS Tagging, and word frequency.

## Required Implementations

### 1. Text Tokenization Implementation

**Function**: tokenize_text(text)

**Requirements**:

- Use NLTK's word_tokenize and sent_tokenize functions
- Input: Raw text string
- Output: Tuple containing (words list, sentences list)

**Implementation Steps**:

1. Import word_tokenize and sent_tokenize from nltk.tokenize
2. Use word_tokenize() to split text into words
3. Use sent_tokenize() to split text into sentences
4. Return both lists as a tuple

### 2. Whitespace Tokenization

**Function**: whitespace_tokenization(text)

**Requirements**:

- Use NLTK's WhitespaceTokenizer

- Input: Raw text string
- Output: List of tokens split on whitespace

**Implementation Steps**:

1. Create a WhitespaceTokenizer instance
2. Use the tokenize() method on the input text
3. Return the resulting tokens

### 3. Punctuation-Based Tokenization

**Function**: punctuation_based_tokenization(text)

**Requirements**:

- Use NLTK's wordpunct_tokenize
- Input: Raw text string
- Output: List of tokens split on punctuation and whitespace

**Implementation Steps**:

1. Use wordpunct_tokenize() directly on the input text
2. Return the resulting tokens

### 4. Basic Character Splitting

**Function**: basic_charecter_splitting(text)

**Requirements**:

- Use Python's built-in string operations
- Input: Raw text string
- Output: List of words, where each word is split into characters

**Implementation Steps**:

1. Split the text into words using text.split()
2. For each word, convert it into a list of characters
3. Return the list of character lists

### 5. Stop Words Removal

**Function**: remove_stop_words(words)

# Generative AI and its Applications

**Requirements**:

- Use NLTK's stopwords
- Input: List of words
- Output: List of words with stop words removed
- Must handle case-insensitive comparison

**Implementation Steps**:

1. Get English stop words using stopwords.words('english')
2. Create a list comprehension that excludes stop words
3. Convert words to lowercase for comparison
4. Return filtered list

## 6. Word Stemming

**Function**: stem_words(words)

**Requirements**:

- Use NLTK's PorterStemmer
- Input: List of words
- Output: List of stemmed words

**Implementation Steps**:

1. Create a PorterStemmer instance
2. Apply stemming to each word using list comprehension
3. Return list of stemmed words

## 7. Lemmatization

**Function**: lemmatize_text(text)

**Requirements**:

- Use SpaCy's lemmatization
- Input: Raw text string
- Output: List of lemmatized words

**Implementation Steps**:

1. Process the text using SpaCy's nlp object

2. Extract lemma_ attribute from each token
3. Return list of lemmatized words

### 8. Named Entity Recognition

**Function**: extract_named_entities(text)

**Requirements**:

- Use SpaCy's NER
- Input: Raw text string
- Output: List of tuples (entity_text, entity_label)

**Implementation Steps**:

1. Process the text using SpaCy's nlp object
2. Extract entities using doc.ents
3. Create tuples of (text, label) for each entity
4. Return list of entity tuples

### 9. POS Tagging with SpaCy

**Function**: pos_tag_spacy(text)

**Requirements**:

- Use SpaCy's POS tagger
- Input: Raw text string
- Output: List of tuples (word, pos_tag)

**Implementation Steps**:

1. Process the text using SpaCy's nlp object
2. Extract text and pos_ attributes from each token
3. Return list of (word, pos_tag) tuples

## Testing Requirements

Test your implementation with scraped news articles from The Associated Press using the links provided. You will obtain a .txt file for each news article processed.

# Generative AI and its Applications

## Submission

- Python file with implemented functions

- Output .txt files for each news article.

Follow naming convention:

Python File: <SRN>_NLP_Lab1_Code.py

Output Files: <SRN>_NLP_Lab1_Article_<Link_Number>.txt