Name: **B. SAI CHARAN**

Roll No: **2203A51L72**

Batch No: **21CSBTB12**

# ASSIGNMENT – 3

**Question:**

Implement Tokenization, Representation of word, Sentence, Word embedding, Word Senses, Linguistic Structure and Dependency Parsing.

**Answer:**

Implementing tokenization, word representation, word embeddings, word senses, linguistic structure, and dependency parsing in Natural Language Processing (NLP), we can utilize various Python libraries such as **NLTK**, **spaCy**, and **gensim**. Below is an explanation of each concept followed by the implementation.

### 1. Tokenization

- Tokenization is the process of splitting text into individual units, such as sentences or words.
- Tokenization in Natural Language Processing NLP Tokenization divides sentences or phrases into smaller units called tokens, such as words or punctuation marks. It's a crucial step in NLP for text processing and machine learning.
- Enables numerical representation of text for algorithmic comprehension. Facilitates text generation, language modeling, and information retrieval.

**TYPES OF TOKENIZATION:**

Tokenization can be classified into several types based on how the text is segmented. Here are some types of tokenization:

a) **Word Tokenization:** Word tokenization divides the text into individual words. Many NLP tasks use this approach, in which words are treated as the basic units of meaning.

Example:

Input: "Tokenization is an important NLP task."
Output: ["Tokenization", "is", "an", "important", "NLP", "task", "."]

b) **Sentence Tokenization:** The text is segmented into sentences during sentence tokenization. This is useful for tasks requiring individual sentence analysis or processing.

Example:

Input: "Tokenization is an important NLP task. It helps break down text into smaller units."
Output: ["Tokenization is an important NLP task.", "It helps break down text into smaller units."]

c) **Subword Tokenization:** In this tokenization entails breaking down words into smaller units, which can be especially useful when dealing with morphologically rich languages or rare words.

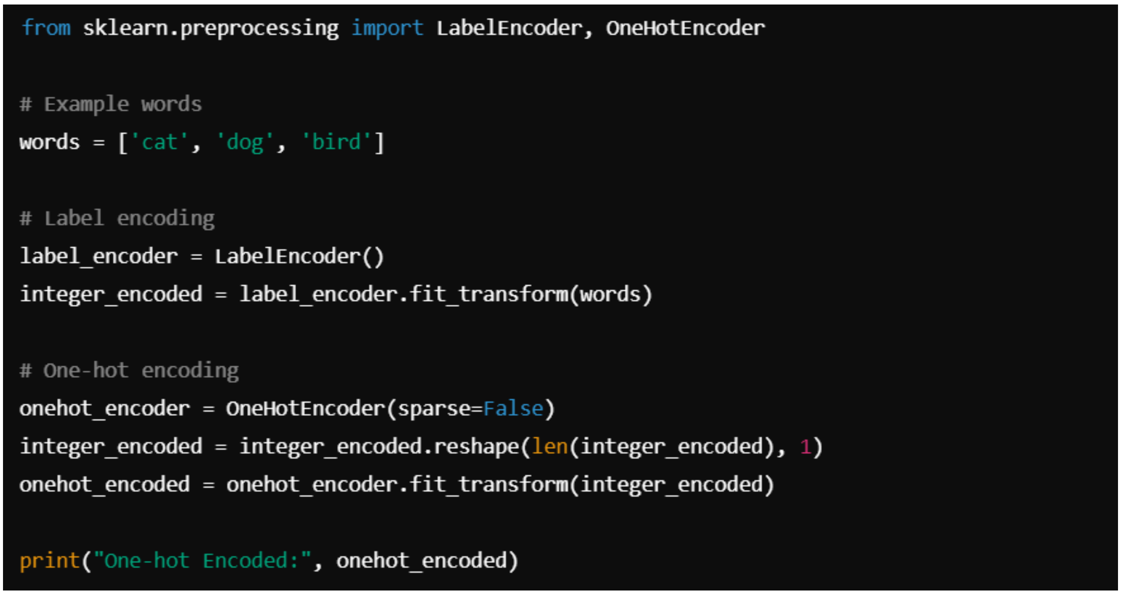
Example:

Input: "tokenization"
Output: ["token", "ization"]

### 2. Representation of Words (One-hot Encoding)

Word representation can be done using one-hot encoding, where each word is represented as a binary vector.

**Code:**

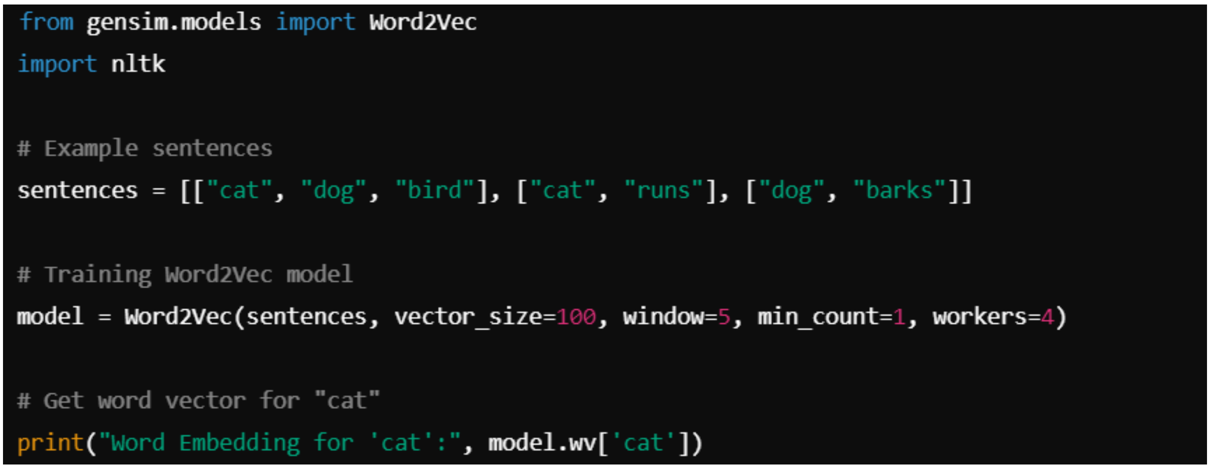


```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Example words
words = ['cat', 'dog', 'bird']

# Label encoding
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(words)

# One-hot encoding
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)

print("One-hot Encoded:", onehot_encoded)
```

### 3. Word Embeddings

Word embeddings are dense vector representations of words. Word2Vec or GloVe are commonly used word embeddings.

**Code:**

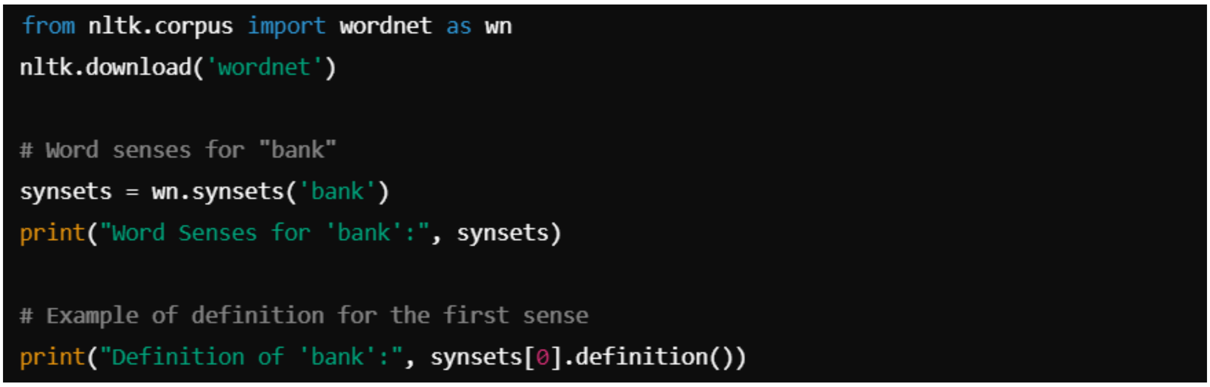


```
from gensim.models import Word2Vec
import nltk

# Example sentences
sentences = [["cat", "dog", "bird"], ["cat", "runs"], ["dog", "barks"]]

# Training Word2Vec model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Get word vector for "cat"
print("Word Embedding for 'cat':", model.wv['cat'])
```

### 4. Word Senses (WordNet)

Words can have multiple senses (meanings). WordNet in NLTK provides word senses.

**Code:**

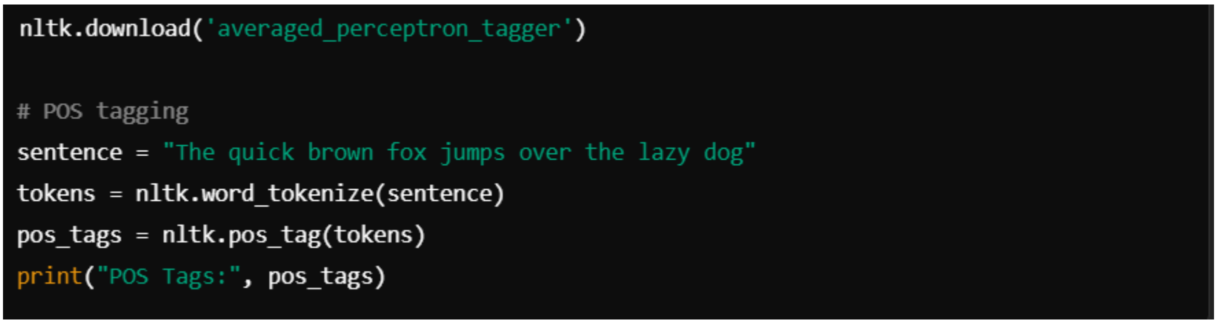


```
from nltk.corpus import wordnet as wn
nltk.download('wordnet')

# Word senses for "bank"
synsets = wn.synsets('bank')
print("Word Senses for 'bank':", synsets)

# Example of definition for the first sense
print("Definition of 'bank':", synsets[0].definition())
```

### 5. Linguistic Structure (Part-of-Speech Tagging)

Linguistic structure can be represented using part-of-speech (POS) tagging, which labels each word with its grammatical category.

**Code:**

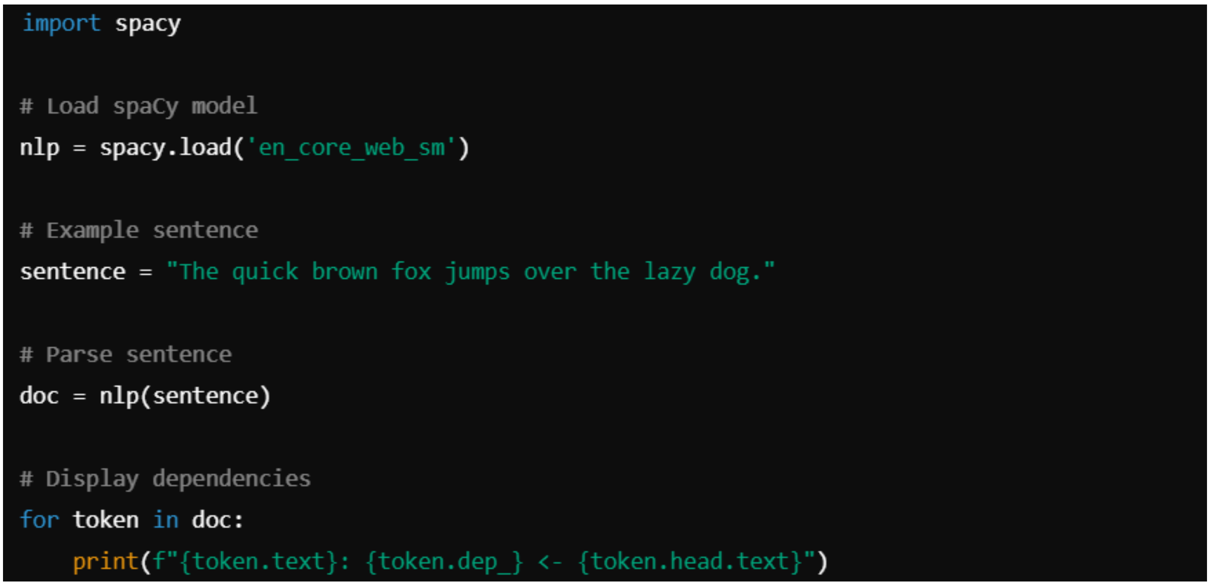


```python
nltk.download('averaged_perceptron_tagger')

# POS tagging
sentence = "The quick brown fox jumps over the lazy dog"
tokens = nltk.word_tokenize(sentence)
pos_tags = nltk.pos_tag(tokens)
print("POS Tags:", pos_tags)
```

### 6. Dependency Parsing

Dependency parsing shows the syntactic structure of a sentence by identifying relationships between words. spaCy is a good tool for dependency parsing.

**Code:**



```python
import spacy

# Load spaCy model
nlp = spacy.load('en_core_web_sm')

# Example sentence
sentence = "The quick brown fox jumps over the lazy dog."

# Parse sentence
doc = nlp(sentence)

# Display dependencies
for token in doc:
    print(f"{token.text}: {token.dep_} <- {token.head.text}")
```

**Summary of Concepts:**

- **Tokenization:** Breaking down text into words or sentences.
- **Word Representation (One-hot):** Binary encoding of words.
- **Word Embeddings:** Dense vector representations of words.
- **Word Senses:** Different meanings of words, typically from WordNet.
- **Linguistic Structure:** POS tagging, categorizing words based on grammar.
- **Dependency Parsing:** Understanding the syntactic relationships between words.