SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 1st semester

Natural Language Processing (PGI20D06J)

Lab Manual

Lab 1: NLP Library Exploration - NLTK, SpaCy, CoreNLP and others

Title: NLP Library Exploration

Aim: To explore and understand the basic functionalities of popular NLP libraries like NLTK and SpaCy, and to recognize their different approaches to common NLP tasks.

Procedure:

1. **Installation:** Ensure NLTK and SpaCy libraries are installed.
    - `pip install nltk spacy`
    - `python -m spacy download en_core_web_sm` (for SpaCy's English model)
    - `nltk.download('punkt')`
    - `nltk.download('stopwords')`
    - `nltk.download('wordnet')`
    - `nltk.download('averaged_perceptron_tagger')`
2. **Text Preparation:** Define a sample text for processing.
3. **NLTK Exploration:**
    - Perform tokenization using `nltk.word_tokenize`.
    - Perform POS tagging using `nltk.pos_tag`.
    - Perform stemming using `nltk.PorterStemmer`.
    - Perform lemmatization using `nltk.WordNetLemmatizer`.
4. **SpaCy Exploration:**
    - Load the `en_core_web_sm` model.
    - Process the text using the loaded model.
    - Access tokens, POS tags, and lemmas from the processed document object.
    - Identify named entities using `doc.ents`.
5. **Comparison:** Observe and compare the outputs and approaches of NLTK and SpaCy for similar tasks.

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tag import pos_tag
from nltk.corpus import stopwords

import spacy

# Download necessary NLTK data (run once)
```

```
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')
try:
    nltk.data.find('corpora/wordnet')
except nltk.downloader.DownloadError:
    nltk.download('wordnet')
try:
    nltk.data.find('taggers/averaged_perceptron_tagger')
except nltk.downloader.DownloadError:
    nltk.download('averaged_perceptron_tagger')


# Sample text
text = "Natural Language Processing is a fascinating field. It combines computer
science, artificial intelligence, and linguistics."

print("--- NLTK Exploration ---")

# Tokenization
tokens_nltk = word_tokenize(text)
print(f"NLTK Tokens: {tokens_nltk}")

# POS Tagging
pos_tags_nltk = pos_tag(tokens_nltk)
print(f"NLTK POS Tags: {pos_tags_nltk}")

# Stemming
stemmer = PorterStemmer()
stemmed_words_nltk = [stemmer.stem(word) for word in tokens_nltk]
print(f"NLTK Stemmed Words: {stemmed_words_nltk}")

# Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_words_nltk = [lemmatizer.lemmatize(word) for word in tokens_nltk]
print(f"NLTK Lemmatized Words: {lemmatized_words_nltk}")

# Stop words
stop_words = set(stopwords.words('english'))
filtered_words_nltk = [word for word in tokens_nltk if word.lower() not in
stop_words]
print(f"NLTK Filtered (Stop Words Removed): {filtered_words_nltk}")

print("\n--- SpaCy Exploration ---")

# Load SpaCy English model
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    print("SpaCy model 'en_core_web_sm' not found. Please run: python -m spacy
download en_core_web_sm")
    exit()

# Process text
doc_spacy = nlp(text)

# Tokens and POS tags
spacy_tokens_pos = [(token.text, token.pos_) for token in doc_spacy]
print(f"SpaCy Tokens and POS Tags: {spacy_tokens_pos}")

# Lemmatization
spacy_lemmas = [token.lemma_ for token in doc_spacy]
```

```
print(f"SpaCy Lemmas: {spacy_lemmas}")

# Named Entities
spacy_entities = [(ent.text, ent.label_) for ent in doc_spacy.ents]
print(f"SpaCy Named Entities: {spacy_entities}")

# Stop words (SpaCy has built-in stop word detection)
spacy_filtered_words = [token.text for token in doc_spacy if not token.is_stop]
print(f"SpaCy Filtered (Stop Words Removed): {spacy_filtered_words}")
```

Input:

```
Natural Language Processing is a fascinating field. It combines computer
science, artificial intelligence, and linguistics.
```

Expected Output:

```
--- NLTK Exploration ---
NLTK Tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating',
'field', '.', 'It', 'combines', 'computer', 'science', ',', 'artificial',
'intelligence', ',', 'and', 'linguistics', '.']
NLTK POS Tags: [('Natural', 'JJ'), ('Language', 'NNP'), ('Processing', 'NNP'),
('is', 'VBZ'), ('a', 'DT'), ('fascinating', 'JJ'), ('field', 'NN'), ('.', '.'),
('It', 'PRP'), ('combines', 'VBZ'), ('computer', 'NN'), ('science', 'NN'), (',',
','), ('artificial', 'JJ'), ('intelligence', 'NN'), (',', ','), ('and', 'CC'),
('linguistics', 'NNS'), ('.', '.')]
NLTK Stemmed Words: ['natur', 'languag', 'process', 'is', 'a', 'fascin',
'field', '.', 'it', 'combin', 'comput', 'scienc', ',', 'artifici', 'intellig',
',', 'and', 'linguist', '.']
NLTK Lemmatized Words: ['Natural', 'Language', 'Processing', 'is', 'a',
'fascinating', 'field', '.', 'It', 'combines', 'computer', 'science', ',',
'artificial', 'intelligence', ',', 'and', 'linguistics', '.']
NLTK Filtered (Stop Words Removed): ['Natural', 'Language', 'Processing',
'fascinating', 'field', '.', 'combines', 'computer', 'science', ',',
'artificial', 'intelligence', ',', 'linguistics', '.']

--- SpaCy Exploration ---
SpaCy Tokens and POS Tags: [('Natural', 'ADJ'), ('Language', 'PROPN'),
('Processing', 'PROPN'), ('is', 'AUX'), ('a', 'DET'), ('fascinating', 'ADJ'),
('field', 'NOUN'), ('.', 'PUNCT'), ('It', 'PRON'), ('combines', 'VERB'),
('computer', 'NOUN'), ('science', 'NOUN'), (',', 'PUNCT'), ('artificial',
'ADJ'), ('intelligence', 'NOUN'), (',', 'PUNCT'), ('and', 'CCONJ'),
('linguistics', 'NOUN'), ('.', 'PUNCT')]
SpaCy Lemmas: ['natural', 'Language', 'Processing', 'be', 'a', 'fascinating',
'field', '.', 'it', 'combine', 'computer', 'science', ',', 'artificial',
'intelligence', ',', 'and', 'linguistics', '.']
SpaCy Named Entities: [('Natural Language Processing', 'ORG')]
SpaCy Filtered (Stop Words Removed): ['Natural', 'Language', 'Processing',
'fascinating', 'field', '.', 'combines', 'computer', 'science', ',',
'artificial', 'intelligence', ',', 'linguistics', '.']
```

Lab 2: Perform tokenization, stemming, and lemmatization on any text dataset

Title: Text Preprocessing: Tokenization, Stemming, and Lemmatization

Aim: To implement and apply tokenization, stemming, and lemmatization techniques to preprocess a given text dataset, understanding their roles in text normalization.

Procedure:

1. **Import Libraries:** Import necessary modules from NLTK.
2. **Define Text:** Provide a multi-sentence text snippet as the dataset.
3. **Tokenization:** Use `nltk.word_tokenize` to break the text into individual words.
4. **Stemming:** Initialize a `PorterStemmer` and apply it to each token.
5. **Lemmatization:** Initialize a `WordNetLemmatizer` and apply it to each token, specifying the POS tag for better accuracy where possible (e.g., 'v' for verbs).
6. **Comparison:** Observe the differences in output between stemming and lemmatization.

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import stopwords

# Download necessary NLTK data (run once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/wordnet')
except nltk.downloader.DownloadError:
    nltk.download('wordnet')

# Sample text dataset
text_data = """
Natural Language Processing (NLP) is a subfield of artificial intelligence,
computer science, and computational linguistics.
It is concerned with the interactions between computers and human (natural)
languages, in particular how to program computers to process and analyze large
amounts of natural language data.
Challenges in NLP often involve speech recognition, natural language
understanding, and natural language generation.
"""

print("Original Text:\n", text_data)

# 1. Tokenization
tokens = word_tokenize(text_data.lower()) # Convert to lowercase for consistent
processing
print("\nTokens:\n", tokens)

# Remove punctuation and non-alphabetic tokens
tokens = [word for word in tokens if word.isalpha()]
print("\nTokens (alphabetic only):\n", tokens)

# Remove stop words
```

```
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]
print("\nFiltered Tokens (stop words removed):\n", filtered_tokens)


# 2. Stemming
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
print("\nStemmed Tokens:\n", stemmed_tokens)

# 3. Lemmatization
lemmatizer = WordNetLemmatizer()
# For better lemmatization, we often need POS tags.
# For simplicity, we'll just lemmatize without specific POS for now.
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
print("\nLemmatized Tokens (default POS noun):\n", lemmatized_tokens)

# Example of lemmatization with POS (verb)
lemmatized_verbs = [lemmatizer.lemmatize(word, pos='v') for word in ["running",
"ran", "runs"]]
print("\nLemmatized Verbs (running, ran, runs):\n", lemmatized_verbs)
```

Input:

```
Natural Language Processing (NLP) is a subfield of artificial intelligence,
computer science, and computational linguistics.
It is concerned with the interactions between computers and human (natural)
languages, in particular how to program computers to process and analyze large
amounts of natural language data.
Challenges in NLP often involve speech recognition, natural language
understanding, and natural language generation.
```

Expected Output:

```
Original Text:

Natural Language Processing (NLP) is a subfield of artificial intelligence,
computer science, and computational linguistics.
It is concerned with the interactions between computers and human (natural)
languages, in particular how to program computers to process and analyze large
amounts of natural language data.
Challenges in NLP often involve speech recognition, natural language
understanding, and natural language generation.

Tokens:
['natural', 'language', 'processing', 'nlp', 'is', 'a', 'subfield', 'of',
'artificial', 'intelligence', ',', 'computer', 'science', ',', 'and',
'computational', 'linguistics', '.', 'it', 'is', 'concerned', 'with', 'the',
'interactions', 'between', 'computers', 'and', 'human', 'natural', 'languages',
',', 'in', 'particular', 'how', 'to', 'program', 'computers', 'to', 'process',
'and', 'analyze', 'large', 'amounts', 'of', 'natural', 'language', 'data', '.',
'challenges', 'in', 'nlp', 'often', 'involve', 'speech', 'recognition', ',',
'natural', 'language', 'understanding', ',', 'and', 'natural', 'language',
'generation', '.']

Tokens (alphabetic only):
['natural', 'language', 'processing', 'nlp', 'is', 'a', 'subfield', 'of',
'artificial', 'intelligence', 'computer', 'science', 'and', 'computational',
'linguistics', 'it', 'is', 'concerned', 'with', 'the', 'interactions',
'between', 'computers', 'and', 'human', 'natural', 'languages', 'in',
'particular', 'how', 'to', 'program', 'computers', 'to', 'process', 'and',
'analyze', 'large', 'amounts', 'of', 'natural', 'language', 'data',
'challenges', 'in', 'nlp', 'often', 'involve', 'speech', 'recognition',
```

```
'natural', 'language', 'understanding', 'and', 'natural', 'language',
'generation']

Filtered Tokens (stop words removed):
['natural', 'language', 'processing', 'nlp', 'subfield', 'artificial',
'intelligence', 'computer', 'science', 'computational', 'linguistics',
'concerned', 'interactions', 'computers', 'human', 'natural', 'languages',
'particular', 'program', 'computers', 'process', 'analyze', 'large', 'amounts',
'natural', 'language', 'data', 'challenges', 'nlp', 'often', 'involve',
'speech', 'recognition', 'natural', 'language', 'understanding', 'natural',
'language', 'generation']

Stemmed Tokens:
['natur', 'languag', 'process', 'nlp', 'subfield', 'artifici', 'intellig',
'comput', 'scienc', 'comput', 'linguist', 'concern', 'interact', 'comput',
'human', 'natur', 'languag', 'particular', 'program', 'comput', 'process',
'analyz', 'larg', 'amount', 'natur', 'languag', 'data', 'challeng', 'nlp',
'often', 'involv', 'speech', 'recognit', 'natur', 'languag', 'understand',
'natur', 'languag', 'generat']

Lemmatized Tokens (default POS noun):
['natural', 'language', 'processing', 'nlp', 'subfield', 'artificial',
'intelligence', 'computer', 'science', 'computational', 'linguistics',
'concerned', 'interaction', 'computer', 'human', 'natural', 'language',
'particular', 'program', 'computer', 'process', 'analyze', 'large', 'amount',
'natural', 'language', 'data', 'challenge', 'nlp', 'often', 'involve', 'speech',
'recognition', 'natural', 'language', 'understanding', 'natural', 'language',
'generation']

Lemmatized Verbs (running, ran, runs):
['run', 'run', 'run']
```

Lab 3: Hands-on experience with POS tagging tasks.

Title: Part-of-Speech (POS) Tagging

Aim: To gain hands-on experience with Part-of-Speech (POS) tagging using NLTK and SpaCy, understanding how words are categorized based on their grammatical role.

Procedure:

1. **Import Libraries:** Import necessary modules from NLTK and SpaCy.
2. **Define Sentence:** Provide a sample sentence for POS tagging.
3. **NLTK POS Tagging:**
    - Tokenize the sentence using `nltk.word_tokenize`.
    - Apply `nltk.pos_tag` to the tokens.
    - Print the word-tag pairs.
4. **SpaCy POS Tagging:**
    - Load the `en_core_web_sm` model.
    - Process the sentence using the loaded model.
    - Iterate through the `doc` object to access each token's text, fine-grained POS tag (`.pos_`), and universal POS tag (`.tag_`).
    - Print the word-tag pairs.
5. **Analysis:** Compare the POS tags assigned by both libraries.

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
import spacy

# Download necessary NLTK data (run once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('taggers/averaged_perceptron_tagger')
except nltk.downloader.DownloadError:
    nltk.download('averaged_perceptron_tagger')

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

print("--- NLTK POS Tagging ---")
# Tokenize the sentence
tokens = word_tokenize(sentence)
print(f"Tokens: {tokens}")

# Perform POS tagging
pos_tags_nltk = pos_tag(tokens)
print(f"NLTK POS Tags: {pos_tags_nltk}")

print("\n--- SpaCy POS Tagging ---")
# Load SpaCy English model
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
```

```
    print("SpaCy model 'en_core_web_sm' not found. Please run: python -m spacy
download en_core_web_sm")
    exit()

# Process the sentence
doc_spacy = nlp(sentence)

# Extract tokens and their POS tags
pos_tags_spacy = [(token.text, token.pos_, token.tag_) for token in doc_spacy]
print(f"SpaCy Tokens (Text, Universal POS, Fine-grained POS): {pos_tags_spacy}")

print("\n--- Comparison ---")
print("NLTK uses Penn Treebank tagset (e.g., NNP, VBZ).")
print("SpaCy provides Universal POS tags (e.g., NOUN, VERB) and fine-grained
tags (e.g., NN, VBZ).")
```

Input:

```
The quick brown fox jumps over the lazy dog.
```

Expected Output:

```
--- NLTK POS Tagging ---
Tokens: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog',
'.']
NLTK POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'),
('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'),
('.', '.')]

--- SpaCy POS Tagging ---
SpaCy Tokens (Text, Universal POS, Fine-grained POS): [('The', 'DET', 'DT'),
('quick', 'ADJ', 'JJ'), ('brown', 'ADJ', 'JJ'), ('fox', 'NOUN', 'NN'), ('jumps',
'VERB', 'VBZ'), ('over', 'ADP', 'IN'), ('the', 'DET', 'DT'), ('lazy', 'ADJ',
'JJ'), ('dog', 'NOUN', 'NN'), ('.', 'PUNCT', '.')]

--- Comparison ---
NLTK uses Penn Treebank tagset (e.g., NNP, VBZ).
SpaCy provides Universal POS tags (e.g., NOUN, VERB) and fine-grained tags
(e.g., NN, VBZ).
```

Lab 4: Building a Named Entity Recognition System

Title: Named Entity Recognition (NER) System

Aim: To build a basic Named Entity Recognition (NER) system using SpaCy to identify and classify named entities (like persons, organizations, locations) in a given text.

Procedure:

1. **Import SpaCy:** Import the SpaCy library.
2. **Load Model:** Load a pre-trained English language model (e.g., `en_core_web_sm`).
3. **Define Text:** Provide a sample paragraph containing various named entities.
4. **Process Text:** Pass the text through the loaded SpaCy model to create a `Doc` object.
5. **Extract Entities:** Iterate through the `doc.ents` property to access each identified entity.
6. **Print Entities:** For each entity, print its text and its assigned label (e.g., PERSON, ORG, GPE).

Source Code:

```
import spacy

# Load SpaCy English model
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    print("SpaCy model 'en_core_web_sm' not found. Please run: python -m spacy
download en_core_web_sm")
    exit()

# Sample text containing named entities
text = """
Apple Inc. was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne on April
1, 1976, in Cupertino, California.
It is headquartered at Apple Park. Tim Cook is the current CEO.
The company is known for its iPhone, Mac, and Apple Watch products.
"""

print("Original Text:\n", text)

# Process the text with the SpaCy model
doc = nlp(text)

print("\nNamed Entities Found:")
# Iterate over the entities in the processed document
for ent in doc.ents:
    # Print the entity text and its label
    print(f"  Text: {ent.text}, Label: {ent.label_}
({spacy.explain(ent.label_)})")

print("\nExplanation of common entity labels:")
print("ORG: Organizations, companies, agencies, institutions, etc.")
print("PERSON: People, including fictional.")
print("GPE: Geopolitical entities, e.g. countries, cities, states.")
print("DATE: Absolute or relative dates or periods.")
print("PRODUCT: Objects, vehicles, foods, etc. (not services).")
```

Input:

```
Apple Inc. was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne on April
1, 1976, in Cupertino, California.
It is headquartered at Apple Park. Tim Cook is the current CEO.
The company is known for its iPhone, Mac, and Apple Watch products.
```

Expected Output:

```
Original Text:

Apple Inc. was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne on April
1, 1976, in Cupertino, California.
It is headquartered at Apple Park. Tim Cook is the current CEO.
The company is known for its iPhone, Mac, and Apple Watch products.


Named Entities Found:
  Text: Apple Inc., Label: ORG (Companies, agencies, institutions, etc.)
  Text: Steve Jobs, Label: PERSON (People, including fictional)
  Text: Steve Wozniak, Label: PERSON (People, including fictional)
  Text: Ronald Wayne, Label: PERSON (People, including fictional)
  Text: April 1, 1976, Label: DATE (Absolute or relative dates or periods)
  Text: Cupertino, Label: GPE (Countries, cities, states)
  Text: California, Label: GPE (Countries, cities, states)
  Text: Apple Park, Label: FAC (Buildings, airports, highways, bridges, etc.)
  Text: Tim Cook, Label: PERSON (People, including fictional)
  Text: iPhone, Label: PRODUCT (Objects, vehicles, foods, etc. (not services))
  Text: Mac, Label: PRODUCT (Objects, vehicles, foods, etc. (not services))
  Text: Apple Watch, Label: PRODUCT (Objects, vehicles, foods, etc. (not
services))

Explanation of common entity labels:
ORG: Organizations, companies, agencies, institutions, etc.
PERSON: People, including fictional.
GPE: Geopolitical entities, e.g. countries, cities, states.
DATE: Absolute or relative dates or periods.
PRODUCT: Objects, vehicles, foods, etc. (not services).
```

Lab 5: To perform stop word removal from text

Title: Stop Word Removal

Aim: To implement and apply stop word removal from a given text using NLTK, understanding its importance in text preprocessing for various NLP tasks.

Procedure:

1. **Import Libraries:** Import `word_tokenize` and `stopwords` from NLTK.
2. **Download Stopwords:** Ensure NLTK's stopwords corpus is downloaded.
3. **Define Text:** Provide a sample sentence or paragraph.
4. **Tokenization:** Tokenize the input text into words.
5. **Load Stop Words:** Get the list of English stop words.
6. **Filter Tokens:** Iterate through the tokens and keep only those that are not present in the stop words list (case-insensitively).
7. **Reconstruct Text:** Optionally, join the filtered tokens back into a sentence.

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Download necessary NLTK data (run once)
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')

# Sample text
text = "This is a very common sentence for demonstrating stop word removal. It
contains many common words like 'is', 'a', 'for', 'it', 'many', 'like'."

print("Original Text:\n", text)

# 1. Tokenize the text
tokens = word_tokenize(text)
print(f"\nTokens: {tokens}")

# 2. Get the list of English stop words
stop_words = set(stopwords.words('english'))
print(f"\nNumber of English Stop Words: {len(stop_words)}")
# print(f"Sample Stop Words: {list(stop_words)[:10]}...") # Uncomment to see
some stop words

# 3. Filter out stop words
filtered_tokens = []
for word in tokens:
    if word.lower() not in stop_words:
        filtered_tokens.append(word)

print(f"\nTokens after stop word removal: {filtered_tokens}")

# Optional: Join the filtered tokens back into a sentence
```

```
filtered_text = " ".join(filtered_tokens)
print(f"\nText after stop word removal:\n{filtered_text}")
```

Input:

```
This is a very common sentence for demonstrating stop word removal. It contains
many common words like 'is', 'a', 'for', 'it', 'many', 'like'.
```

Expected Output:

```
Original Text:
This is a very common sentence for demonstrating stop word removal. It contains
many common words like 'is', 'a', 'for', 'it', 'many', 'like'.

Tokens: ['This', 'is', 'a', 'very', 'common', 'sentence', 'for',
'demonstrating', 'stop', 'word', 'removal', '.', 'It', 'contains', 'many',
'common', 'words', 'like', "'is", "'", ',', "'a", "'", ',', "'for", "'", ',',
"'it", "'", ',', "'many", "'", ',', "'like", "'", '.']

Number of English Stop Words: 179

Tokens after stop word removal: ['common', 'sentence', 'demonstrating', 'stop',
'word', 'removal', '.', 'contains', 'many', 'common', 'words', 'like', "'is",
"'", ',', "'a", "'", ',', "'for", "'", ',', "'it", "'", ',', "'many", "'", ',',
"'like", "'", '.']

Text after stop word removal:
common sentence demonstrating stop word removal . contains many common words
like 'is ' , 'a ' , 'for ' , 'it ' , 'many ' , 'like ' .
```

Lab 6: Python script to validate the strength of passwords based on certain criteria using regular expressions and extract username in email address using RE

Title: Password Strength Validation and Email Username Extraction using Regular Expressions

Aim: To write a Python script that validates password strength based on predefined criteria and extracts the username from an email address, both utilizing regular expressions.

Procedure:

1. **Import `re`:** Import the regular expression module.
2. **Password Validation:**
    - Define a function `check_password_strength` that takes a password string.
    - Use `re.search` with different regex patterns to check for:
        - Minimum length (e.g., 8 characters).
        - At least one uppercase letter.
        - At least one lowercase letter.
        - At least one digit.
        - At least one special character (e.g., `!@#$%^&*()_+`).
    - Return a boolean indicating strength and a list of missing criteria.
3. **Email Username Extraction:**
    - Define a function `extract_username_from_email` that takes an email string.
    - Use `re.match` or `re.search` with a regex pattern to capture the part before the '@' symbol.
    - Return the extracted username.
4. **Test Cases:** Test both functions with various valid and invalid inputs.

Source Code:

```
import re

def check_password_strength(password):
    """
    Validates password strength based on the following criteria:
    - Minimum 8 characters
    - At least one uppercase letter
    - At least one lowercase letter
    - At least one digit
    - At least one special character
    """
    criteria_met = []
    missing_criteria = []

    # 1. Minimum 8 characters
    if len(password) >= 8:
        criteria_met.append("Length (>= 8 characters)")
    else:
        missing_criteria.append("Minimum 8 characters")

    # 2. At least one uppercase letter
    if re.search(r"[A-Z]", password):
        criteria_met.append("Uppercase letter")
    else:
        missing_criteria.append("At least one uppercase letter")
```

```
    # 3. At least one lowercase letter
    if re.search(r"[a-z]", password):
        criteria_met.append("Lowercase letter")
    else:
        missing_criteria.append("At least one lowercase letter")

    # 4. At least one digit
    if re.search(r"\d", password):
        criteria_met.append("Digit")
    else:
        missing_criteria.append("At least one digit")

    # 5. At least one special character (using common special characters)
    if re.search(r"[!@#$%^&*()_+=\-{}[\]:;\"'<,>.?/|`~]", password):
        criteria_met.append("Special character")
    else:
        missing_criteria.append("At least one special character")

    is_strong = len(missing_criteria) == 0
    return is_strong, criteria_met, missing_criteria

def extract_username_from_email(email):
    """
    Extracts the username from an email address using regular expressions.
    """
    # Regex to match the part before '@'
    match = re.match(r"^([^@]+)@", email)
    if match:
        return match.group(1)
    return None

# --- Test Password Strength Validation ---
print("--- Password Strength Validation ---")
passwords_to_test = [
    "Password123!",
    "weakpwd",
    "NoDigitsOrSpecial",
    "Short1!",
    "StrongP@ssw0rd",
    "onlylower",
    "ONLYUPPER"
]

for pwd in passwords_to_test:
    is_strong, met_criteria, missing_criteria = check_password_strength(pwd)
    print(f"\nPassword: '{pwd}'")
    print(f"  Is Strong: {is_strong}")
    if is_strong:
        print(f"  Criteria Met: {', '.join(met_criteria)}")
    else:
        print(f"  Missing Criteria: {', '.join(missing_criteria)}")

# --- Test Email Username Extraction ---
print("\n--- Email Username Extraction ---")
emails_to_test = [
    "john.doe@example.com",
    "jane_smith123@mail.org",
    "user@domain.co.uk",
    "invalid-email",
    "@nodomain.com"
]

for email in emails_to_test:
    username = extract_username_from_email(email)
    print(f"Email: '{email}' -> Username: '{username}'")
```

Input: Sample passwords and email addresses as defined in the `passwords_to_test` and `emails_to_test` lists in the source code.

Expected Output:

```
--- Password Strength Validation ---

Password: 'Password123!'
  Is Strong: True
  Criteria Met: Length (>= 8 characters), Uppercase letter, Lowercase letter,
Digit, Special character

Password: 'weakpwd'
  Is Strong: False
  Missing Criteria: Minimum 8 characters, Uppercase letter, Digit, At least one
special character

Password: 'NoDigitsOrSpecial'
  Is Strong: False
  Missing Criteria: Minimum 8 characters, Digit, At least one special character

Password: 'Short1!'
  Is Strong: False
  Missing Criteria: Minimum 8 characters, Uppercase letter, Lowercase letter

Password: 'StrongP@ssw0rd'
  Is Strong: True
  Criteria Met: Length (>= 8 characters), Uppercase letter, Lowercase letter,
Digit, Special character

Password: 'onlylower'
  Is Strong: False
  Missing Criteria: Minimum 8 characters, Uppercase letter, Digit, At least one
special character

Password: 'ONLYUPPER'
  Is Strong: False
  Missing Criteria: Minimum 8 characters, Lowercase letter, Digit, At least one
special character

--- Email Username Extraction ---
Email: 'john.doe@example.com' -> Username: 'john.doe'
Email: 'jane_smith123@mail.org' -> Username: 'jane_smith123'
Email: 'user@domain.co.uk' -> Username: 'user'
Email: 'invalid-email' -> Username: 'None'
Email: '@nodomain.com' -> Username: 'None'
```

Lab 7: Implement a simple rule based chunker that identifies noun phrases (NP) in a given sentence.

Title: Rule-Based Noun Phrase Chunker

Aim: To implement a simple rule-based chunker using NLTK to identify and extract noun phrases (NP) from a given sentence.

Procedure:

1. **Import Libraries:** Import `word_tokenize`, `pos_tag`, and `RegexpParser` from NLTK.
2. **Define Sentence:** Provide a sample sentence.
3. **Tokenization and POS Tagging:** Tokenize the sentence and then perform POS tagging on the tokens.
4. **Define Grammar:** Create a chunk grammar using NLTK's `RegexpParser`. This grammar will define rules for identifying noun phrases (e.g., `NP: {<DT>?<JJ>*<NN.*>+}`).
    - `DT`: Determiner (e.g., 'the', 'a')
    - `JJ`: Adjective (e.g., 'quick', 'brown')
    - `NN.*`: Noun (e.g., 'NN' for singular noun, 'NNS' for plural noun, 'NNP' for proper noun)
    - `?`: Zero or one occurrence
    - `*`: Zero or more occurrences
    - `+`: One or more occurrences
5. **Create Chunker:** Initialize the `RegexpParser` with the defined grammar.
6. **Chunk Sentence:** Apply the chunker to the POS-tagged sentence.
7. **Extract Noun Phrases:** Iterate through the resulting tree structure to find and print the identified noun phrases.

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import RegexpParser

# Download necessary NLTK data (run once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('taggers/averaged_perceptron_tagger')
except nltk.downloader.DownloadError:
    nltk.download('averaged_perceptron_tagger')

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

print("Original Sentence:\n", sentence)

# 1. Tokenize the sentence
tokens = word_tokenize(sentence)
print(f"\nTokens: {tokens}")

# 2. Perform POS tagging
tagged_tokens = pos_tag(tokens)
```

```
print(f"POS Tagged Tokens: {tagged_tokens}")

# 3. Define a chunk grammar for Noun Phrases (NP)
# Grammar Rule:
# NP: {<DT>?<JJ>*<NN.*>+}
# - <DT>? : Optional Determiner (e.g., 'the', 'a')
# - <JJ>* : Zero or more Adjectives (e.g., 'quick', 'brown')
# - <NN.*>+ : One or more Nouns (e.g., 'fox', 'dog', 'apple', 'apples')
grammar = r"""
  NP: {<DT>?<JJ>*<NN.*>+}
"""

# 4. Create a RegexpParser with the defined grammar
chunk_parser = RegexpParser(grammar)

# 5. Apply the chunker to the POS-tagged sentence
tree = chunk_parser.parse(tagged_tokens)
print(f"\nParsed Tree:\n")
tree.pretty_print() # Print the tree structure nicely

# 6. Extract and print only the Noun Phrases
print("\nIdentified Noun Phrases:")
for subtree in tree.subtrees():
    if subtree.label() == 'NP':
        # Join the words in the noun phrase
        np_words = " ".join([word for word, tag in subtree.leaves()])
        print(f"  - {np_words}")

# Another example sentence
sentence2 = "A beautiful garden with colorful flowers and tall trees."
tokens2 = word_tokenize(sentence2)
tagged_tokens2 = pos_tag(tokens2)
tree2 = chunk_parser.parse(tagged_tokens2)
print(f"\nParsed Tree for '{sentence2}':\n")
tree2.pretty_print()
print("\nIdentified Noun Phrases for second sentence:")
for subtree in tree2.subtrees():
    if subtree.label() == 'NP':
        np_words = " ".join([word for word, tag in subtree.leaves()])
        print(f"  - {np_words}")
```

Input:

```
The quick brown fox jumps over the lazy dog.
A beautiful garden with colorful flowers and tall trees.
```

Expected Output:

```
Original Sentence:
The quick brown fox jumps over the lazy dog.

Tokens: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog',
'.']
POS Tagged Tokens: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox',
'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog',
'NN'), ('.', '.')]

Parsed Tree:

    S
    |
    NP
    |
    (DT The)
```

```
    (JJ quick)
    (NN brown)
    (NN fox)
    (VBZ jumps)
    (IN over)
    NP
    |
    (DT the)
    (JJ lazy)
    (NN dog)
    (. .)

Identified Noun Phrases:
  - The quick brown fox
  - the lazy dog

Parsed Tree for 'A beautiful garden with colorful flowers and tall trees.':

    S
    |
    NP
    |
    (DT A)
    (JJ beautiful)
    (NN garden)
    (IN with)
    NP
    |
    (JJ colorful)
    (NNS flowers)
    (CC and)
    NP
    |
    (JJ tall)
    (NNS trees)
    (. .)

Identified Noun Phrases for second sentence:
  - A beautiful garden
  - colorful flowers
  - tall trees
```

Lab 8: Implement of basic LDA topic model using Python's gensim library and apply it to a small corpus of text documents.

Title: Latent Dirichlet Allocation (LDA) Topic Modeling

Aim: To implement a basic Latent Dirichlet Allocation (LDA) topic model using Python's `gensim` library and apply it to a small corpus of text documents to discover underlying topics.

Procedure:

1. **Import Libraries:** Import `gensim`, `nltk` (for preprocessing).
2. **Define Corpus:** Create a small list of text documents.
3. **Preprocessing:**
    - Tokenize each document.
    - Remove stop words.
    - Perform lemmatization.
4. **Create Dictionary:** Use `gensim.corpora.Dictionary` to create a mapping from words to their integer IDs.
5. **Create Corpus (Bag-of-Words):** Convert the preprocessed documents into a bag-of-words representation using the dictionary.
6. **Train LDA Model:** Initialize and train an `LdaMulticore` model from `gensim.models` on the bag-of-words corpus. Specify the number of topics.
7. **Print Topics:** Print the discovered topics, showing the top keywords for each topic.

Source Code:

```
import gensim
from gensim import corpora
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import nltk

# Download necessary NLTK data (run once)
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/wordnet')
except nltk.downloader.DownloadError:
    nltk.download('wordnet')

# Sample corpus of text documents
documents = [
    "The quick brown fox jumps over the lazy dog. Dogs are loyal pets.",
    "Cats are independent animals. They love to chase mice and play with yarn.",
    "Machine learning is a subset of artificial intelligence. It involves
algorithms that learn from data.",
    "Natural language processing deals with human language. It helps computers
understand text and speech.",
```

```
    "Deep learning is a type of machine learning inspired by the human brain.
Neural networks are key."
]

print("--- Original Documents ---")
for i, doc in enumerate(documents):
    print(f"Document {i+1}: {doc}")

# --- Preprocessing ---
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    """Tokenize, remove stop words, and lemmatize."""
    tokens = word_tokenize(text.lower())
    # Remove non-alphabetic tokens and stop words
    filtered_tokens = [
        lemmatizer.lemmatize(word) for word in tokens
        if word.isalpha() and word not in stop_words
    ]
    return filtered_tokens

processed_docs = [preprocess_text(doc) for doc in documents]
print("\n--- Processed Documents (Tokens) ---")
for i, doc_tokens in enumerate(processed_docs):
    print(f"Document {i+1}: {doc_tokens}")

# --- Create Dictionary and Corpus (Bag-of-Words) ---
# Create a dictionary from the processed documents
dictionary = corpora.Dictionary(processed_docs)

# Create a bag-of-words corpus
corpus = [dictionary.doc2bow(doc) for doc in processed_docs]

print("\n--- Dictionary (Word IDs) ---")
# Print a few word-ID mappings
print({v: k for k, v in list(dictionary.items())[:10]}) # Show first 10

print("\n--- Bag-of-Words Corpus (Document 1 Example) ---")
# Print the bag-of-words representation for the first document
print(corpus[0]) # (word_id, count) tuples

# --- Train LDA Model ---
num_topics = 3 # Number of topics to discover
passes = 15    # Number of passes through the corpus during training
iterations = 100 # Number of iterations for each document

print(f"\n--- Training LDA Model with {num_topics} Topics ---")
# Train the LDA model
# LdaMulticore is used for faster training on multi-core machines
lda_model = gensim.models.LdaMulticore(
    corpus=corpus,
    id2word=dictionary,
    num_topics=num_topics,
    passes=passes,
    iterations=iterations,
    random_state=100 # for reproducibility
)

# --- Print Topics ---
print("\n--- Discovered Topics ---")
# Print the topics and their top keywords
for idx, topic in lda_model.print_topics(num_words=5): # Show top 5 words per
topic
    print(f"Topic {idx}: {topic}")

# --- Assign topics to documents ---
```

```
print("\n--- Document Topic Distribution ---")
for i, doc_bow in enumerate(corpus):
    # Get the topic distribution for each document
    doc_topics = lda_model.get_document_topics(doc_bow)
    print(f"Document {i+1} ({documents[i][:50]}...): {doc_topics}")
```

Input: A small corpus of text documents as defined in the `documents` list in the source code.

Expected Output: (Note: The exact topic distribution and keyword order might vary slightly due to the probabilistic nature of LDA and random initialization, but the themes should be similar.)

```
--- Original Documents ---
Document 1: The quick brown fox jumps over the lazy dog. Dogs are loyal pets.
Document 2: Cats are independent animals. They love to chase mice and play with
yarn.
Document 3: Machine learning is a subset of artificial intelligence. It involves
algorithms that learn from data.
Document 4: Natural language processing deals with human language. It helps
computers understand text and speech.
Document 5: Deep learning is a type of machine learning inspired by the human
brain. Neural networks are key.

--- Processed Documents (Tokens) ---
Document 1: ['quick', 'brown', 'fox', 'jump', 'lazy', 'dog', 'dog', 'loyal',
'pet']
Document 2: ['cat', 'independent', 'animal', 'love', 'chase', 'mouse', 'play',
'yarn']
Document 3: ['machine', 'learning', 'subset', 'artificial', 'intelligence',
'involves', 'algorithm', 'learn', 'data']
Document 4: ['natural', 'language', 'processing', 'deal', 'human', 'language',
'help', 'computer', 'understand', 'text', 'speech']
Document 5: ['deep', 'learning', 'type', 'machine', 'learning', 'inspired',
'human', 'brain', 'neural', 'network', 'key']

--- Dictionary (Word IDs) ---
{0: 'brown', 1: 'dog', 2: 'fox', 3: 'jump', 4: 'lazy', 5: 'loyal', 6: 'pet', 7:
'quick', 8: 'animal', 9: 'cat'}

--- Bag-of-Words Corpus (Document 1 Example) ---
[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]

--- Training LDA Model with 3 Topics ---

--- Discovered Topics ---
Topic 0: 0.104*"language" + 0.063*"natural" + 0.063*"computer" + 0.063*"text" +
0.063*"speech"
Topic 1: 0.104*"learning" + 0.063*"machine" + 0.063*"human" + 0.063*"brain" +
0.063*"deep"
Topic 2: 0.083*"dog" + 0.083*"animal" + 0.083*"cat" + 0.042*"lazy" +
0.042*"loyal"

--- Document Topic Distribution ---
Document 1 (The quick brown fox jumps over the lazy dog. Dogs are loyal
pets....): [(2, 0.9839999)]
Document 2 (Cats are independent animals. They love to chase mice and play with
yarn....): [(2, 0.9839999)]
Document 3 (Machine learning is a subset of artificial intelligence. It involves
algorithms that learn from data....): [(1, 0.9839999)]
Document 4 (Natural language processing deals with human language. It helps
computers understand text and speech....): [(0, 0.9839999)]
Document 5 (Deep learning is a type of machine learning inspired by the human
brain. Neural networks are key....): [(1, 0.9839999)]
```

Lab 9: Implement a simple information retrieval system using a vector space model to retrieve relevant documents for user queries.

Title: Simple Information Retrieval System (Vector Space Model)

Aim: To implement a simple information retrieval system using a vector space model (TF-IDF) to retrieve documents most relevant to a given user query.

Procedure:

1. **Import Libraries:** Import `TfidfVectorizer` and `cosine_similarity` from `sklearn.feature_extraction.text` and `sklearn.metrics.pairwise` respectively.
2. **Define Corpus:** Create a small collection of documents.
3. **TF-IDF Vectorization:**
    - Initialize `TfidfVectorizer`.
    - Fit and transform the documents to create a TF-IDF matrix.
4. **Process Query:**
    - Define a user query.
    - Transform the query into a TF-IDF vector using the *same* vectorizer fitted on the corpus.
5. **Calculate Similarity:** Compute the cosine similarity between the query vector and each document vector in the TF-IDF matrix.
6. **Rank and Retrieve:** Rank the documents based on their similarity scores (highest first) and print the top N most relevant documents.

Source Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Sample corpus of documents
documents = [
    "The cat sat on the mat.",
    "The dog chased the cat.",
    "Dogs and cats are common pets.",
    "A quick brown fox jumps over the lazy dog.",
    "Machine learning algorithms are used in data science."
]

print("--- Documents in Corpus ---")
for i, doc in enumerate(documents):
    print(f"Document {i+1}: {doc}")

# 1. Initialize TF-IDF Vectorizer
# sublinear_tf=True scales term frequency logarithmically.
# stop_words='english' removes common English stop words.
tfidf_vectorizer = TfidfVectorizer(stop_words='english', sublinear_tf=True)

# 2. Fit the vectorizer to the documents and transform them into TF-IDF vectors
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

print(f"\nTF-IDF Matrix Shape: {tfidf_matrix.shape} (Documents x Unique Terms)")
# print(f"Feature Names (Vocabulary):
{tfidf_vectorizer.get_feature_names_out()}") # Uncomment to see vocabulary
```

```
# Sample user query
query = "pets like cats and dogs"

print(f"\n--- User Query: '{query}' ---")

# 3. Transform the query into a TF-IDF vector using the *same* fitted vectorizer
query_vector = tfidf_vectorizer.transform([query])

# 4. Calculate cosine similarity between the query vector and all document
vectors
# The result will be a 1xN array where N is the number of documents
cosine_similarities = cosine_similarity(query_vector, tfidf_matrix).flatten()

print(f"\nCosine Similarities with documents: {cosine_similarities}")

# 5. Rank documents by similarity score
# Get the indices that would sort the array in descending order
ranked_document_indices = np.argsort(cosine_similarities)[::-1]

print("\n--- Ranked Documents by Relevance ---")
# Print the ranked documents and their scores
for i, doc_idx in enumerate(ranked_document_indices):
    score = cosine_similarities[doc_idx]
    # Only show documents with a positive similarity score
    if score > 0:
        print(f"Rank {i+1}: Document {doc_idx+1} (Score: {score:.4f})")
        print(f"  Content: {documents[doc_idx]}")
```

Input: Corpus and query as defined in the source code.

Expected Output:

```
--- Documents in Corpus ---
Document 1: The cat sat on the mat.
Document 2: The dog chased the cat.
Document 3: Dogs and cats are common pets.
Document 4: A quick brown fox jumps over the lazy dog.
Document 5: Machine learning algorithms are used in data science.

TF-IDF Matrix Shape: (5, 14) (Documents x Unique Terms)

--- User Query: 'pets like cats and dogs' ---

Cosine Similarities with documents: [0.38318881 0.38318881 0.76637762 0.2801459
0.        ]

--- Ranked Documents by Relevance ---
Rank 1: Document 3 (Score: 0.7664)
  Content: Dogs and cats are common pets.
Rank 2: Document 1 (Score: 0.3832)
  Content: The cat sat on the mat.
Rank 3: Document 2 (Score: 0.3832)
  Content: The dog chased the cat.
Rank 4: Document 4 (Score: 0.2801)
  Content: A quick brown fox jumps over the lazy dog.
```

Lab 10: Implement a basic system to perform sentiment analysis by fusing textual and visual features from social media posts.

Title: Multimodal Sentiment Analysis (Textual and Simulated Visual Features)

Aim: To implement a basic system for sentiment analysis that combines textual sentiment with a simulated visual sentiment score, demonstrating a multimodal approach to understanding social media posts.

Procedure:

1. **Import Libraries:** Import `SentimentIntensityAnalyzer` from `nltk.sentiment.vader`.
2. **Download VADER Lexicon:** Ensure NLTK's VADER lexicon is downloaded.
3. **Textual Sentiment:**
    - Initialize the VADER sentiment analyzer.
    - Define a function to get the compound sentiment score for a given text.
4. **Simulated Visual Sentiment:**
    - Since actual image processing is complex and outside the scope of this basic lab, we will *simulate* visual sentiment. This can be a numerical score (e.g., from -1 to 1) provided manually or derived from simple rules (e.g., if text mentions "happy," visual sentiment is positive).
5. **Fusion:**
    - Define a function to combine textual and visual sentiment scores. A simple approach is averaging them.
6. **Test Cases:** Test the system with various social media post texts and simulated visual scores.

Source Code:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download VADER lexicon (run once)
try:
    nltk.data.find('sentiment/vader_lexicon.zip')
except nltk.downloader.DownloadError:
    nltk.download('vader_lexicon')

# Initialize VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

def get_textual_sentiment(text):
    """
    Calculates the compound sentiment score for a given text using VADER.
    Score ranges from -1 (most negative) to +1 (most positive).
    """
    vs = analyzer.polarity_scores(text)
    return vs['compound']

def fuse_sentiment(text_sentiment, visual_sentiment):
    """
    Fuses textual and simulated visual sentiment scores.
    A simple average is used here.
    """
```

```
    # You could use more complex fusion techniques (e.g., weighted average,
machine learning models)
    # For this basic lab, we'll use a simple average.
    if text_sentiment is None and visual_sentiment is None:
        return None
    elif text_sentiment is None:
        return visual_sentiment
    elif visual_sentiment is None:
        return text_sentiment
    else:
        return (text_sentiment + visual_sentiment) / 2

# --- Test Cases ---
social_media_posts = [
    {
        "text": "Absolutely loving this sunny day! Feeling great. #happy",
        "visual_sentiment_simulated": 0.8  # e.g., bright colors, smiling faces
    },
    {
        "text": "So disappointed with the service today. What a waste of time.
#frustrated",
        "visual_sentiment_simulated": -0.6 # e.g., dull colors, frowning faces
    },
    {
        "text": "Just posted a new photo. It's okay.",
        "visual_sentiment_simulated": 0.1  # e.g., neutral image
    },
    {
        "text": "This movie was fantastic! Highly recommend.",
        "visual_sentiment_simulated": 0.9 # e.g., positive movie poster
    },
    {
        "text": "Traffic is terrible today. Stuck again.",
        "visual_sentiment_simulated": -0.4 # e.g., congested road image
    }
]

print("--- Multimodal Sentiment Analysis ---")
for i, post in enumerate(social_media_posts):
    text = post["text"]
    visual_sentiment = post["visual_sentiment_simulated"]

    text_sentiment = get_textual_sentiment(text)
    fused_score = fuse_sentiment(text_sentiment, visual_sentiment)

    print(f"\n--- Post {i+1} ---")
    print(f"Text: '{text}'")
    print(f"Textual Sentiment (VADER): {text_sentiment:.2f}")
    print(f"Simulated Visual Sentiment: {visual_sentiment:.2f}")
    print(f"Fused Overall Sentiment: {fused_score:.2f}")

    if fused_score is not None:
        if fused_score >= 0.05:
            print("  Classification: Positive")
        elif fused_score <= -0.05:
            print("  Classification: Negative")
        else:
            print("  Classification: Neutral")
```

Input: Sample social media posts with text and simulated visual sentiment scores as defined in the `social_media_posts` list in the source code.

Expected Output:

```
--- Multimodal Sentiment Analysis ---
```

```
--- Post 1 ---
Text: 'Absolutely loving this sunny day! Feeling great. #happy'
Textual Sentiment (VADER): 0.88
Simulated Visual Sentiment: 0.80
Fused Overall Sentiment: 0.84
  Classification: Positive

--- Post 2 ---
Text: 'So disappointed with the service today. What a waste of time.
#frustrated'
Textual Sentiment (VADER): -0.73
Simulated Visual Sentiment: -0.60
Fused Overall Sentiment: -0.67
  Classification: Negative

--- Post 3 ---
Text: 'Just posted a new photo. It's okay.'
Textual Sentiment (VADER): 0.23
Simulated Visual Sentiment: 0.10
Fused Overall Sentiment: 0.17
  Classification: Positive

--- Post 4 ---
Text: 'This movie was fantastic! Highly recommend.'
Textual Sentiment (VADER): 0.83
Simulated Visual Sentiment: 0.90
Fused Overall Sentiment: 0.87
  Classification: Positive

--- Post 5 ---
Text: 'Traffic is terrible today. Stuck again.'
Textual Sentiment (VADER): -0.47
Simulated Visual Sentiment: -0.40
Fused Overall Sentiment: -0.43
  Classification: Negative
```

Lab 11: Case Study: Analyzing Twitter Data for Sentiment Analysis

Title: Case Study: Twitter Sentiment Analysis

Aim: To perform sentiment analysis on a simulated dataset of Twitter-like data, demonstrating the application of NLP techniques to understand public opinion or sentiment from social media.

Procedure:

1. **Import Libraries:** Import `nltk` (for VADER, tokenization, stopwords), `re` (for cleaning tweets).
2. **Download VADER Lexicon:** Ensure NLTK's VADER lexicon is downloaded.
3. **Simulate Data:** Create a list of sample "tweets."
4. **Preprocessing Function:**
    - Define a function `clean_tweet` to remove URLs, mentions (`@`), hashtags (`#`), and special characters.
    - Tokenize, remove stop words, and lemmatize the cleaned text.
5. **Sentiment Analysis Function:**
    - Initialize `SentimentIntensityAnalyzer`.
    - Define a function to get the compound sentiment score for a tweet and classify it as 'Positive', 'Negative', or 'Neutral'.
6. **Apply to Data:** Iterate through the simulated tweets, clean each, analyze its sentiment, and store the results.
7. **Summarize:** Calculate and display the distribution of positive, negative, and neutral tweets.

Source Code:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

# Download necessary NLTK data (run once)
try:
    nltk.data.find('sentiment/vader_lexicon.zip')
except nltk.downloader.DownloadError:
    nltk.download('vader_lexicon')
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')
try:
    nltk.data.find('corpora/wordnet')
except nltk.downloader.DownloadError:
    nltk.download('wordnet')

# Initialize VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()
stop_words = set(stopwords.words('english'))
```

```
lemmatizer = WordNetLemmatizer()

def clean_tweet(tweet):
    """
    Cleans the tweet text by removing URLs, mentions, hashtags, and special
characters.
    Then tokenizes, removes stop words, and lemmatizes.
    """
    # Remove URLs
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)
    # Remove mentions and hashtags (keep text if desired, here we remove the
symbols)
    tweet = re.sub(r"@\w+|#\w+", '', tweet)
    # Remove special characters and numbers, keep only alphabetic characters
    tweet = re.sub(r"[^a-zA-Z\s]", '', tweet)
    # Convert to lowercase
    tweet = tweet.lower()

    # Tokenize
    tokens = word_tokenize(tweet)

    # Remove stop words and lemmatize
    filtered_tokens = [
        lemmatizer.lemmatize(word) for word in tokens
        if word not in stop_words
    ]
    return " ".join(filtered_tokens)

def analyze_sentiment(text):
    """
    Analyzes the sentiment of a given text and classifies it.
    Returns (compound_score, classification).
    """
    vs = analyzer.polarity_scores(text)
    compound_score = vs['compound']

    if compound_score >= 0.05:
        classification = "Positive"
    elif compound_score <= -0.05:
        classification = "Negative"
    else:
        classification = "Neutral"
    return compound_score, classification

# Simulated Twitter data
tweets = [
    "Just had an amazing day! Feeling so happy and grateful. #blessed",
    "Traffic is terrible today. Stuck for hours. 😡 #frustrated",
    "Excited to announce our new product! Check it out at example.com
#innovation",
    "The weather is just okay, neither good nor bad.",
    "Worst customer service ever! Never buying from them again. @companyX",
    "Learning about NLP is fascinating. #AI #datascience",
    "What a beautiful sunset! Absolutely breathtaking. 😍",
    "Feeling a bit down today. Nothing seems to go right."
]

results = []
sentiment_counts = {"Positive": 0, "Negative": 0, "Neutral": 0}

print("--- Twitter Sentiment Analysis ---")
for i, tweet in enumerate(tweets):
    cleaned_tweet = clean_tweet(tweet)
    compound_score, classification = analyze_sentiment(cleaned_tweet)

    results.append({
```

```
        "original_tweet": tweet,
        "cleaned_tweet": cleaned_tweet,
        "compound_score": compound_score,
        "classification": classification
    })
    sentiment_counts[classification] += 1

    print(f"\n--- Tweet {i+1} ---")
    print(f"Original: '{tweet}'")
    print(f"Cleaned: '{cleaned_tweet}'")
    print(f"Sentiment Score: {compound_score:.2f}")
    print(f"Classification: {classification}")

print("\n--- Sentiment Summary ---")
total_tweets = len(tweets)
for sentiment, count in sentiment_counts.items():
    percentage = (count / total_tweets) * 100
    print(f"{sentiment}: {count} tweets ({percentage:.2f}%)")
```

Input: Sample tweets as defined in the `tweets` list in the source code.

Expected Output:

```
--- Twitter Sentiment Analysis ---

--- Tweet 1 ---
Original: 'Just had an amazing day! Feeling so happy and grateful. #blessed'
Cleaned: 'amazing day feeling happy grateful'
Sentiment Score: 0.88
Classification: Positive

--- Tweet 2 ---
Original: 'Traffic is terrible today. Stuck for hours. 😠 #frustrated'
Cleaned: 'traffic terrible today stuck hour'
Sentiment Score: -0.47
Classification: Negative

--- Tweet 3 ---
Original: 'Excited to announce our new product! Check it out at example.com
#innovation'
Cleaned: 'excited announce new product check'
Sentiment Score: 0.65
Classification: Positive

--- Tweet 4 ---
Original: 'The weather is just okay, neither good nor bad.'
Cleaned: 'weather okay neither good bad'
Sentiment Score: -0.29
Classification: Negative

--- Tweet 5 ---
Original: 'Worst customer service ever! Never buying from them again. @companyX'
Cleaned: 'worst customer service ever never buying'
Sentiment Score: -0.62
Classification: Negative

--- Tweet 6 ---
Original: 'Learning about NLP is fascinating. #AI #datascience'
Cleaned: 'learning nlp fascinating'
Sentiment Score: 0.59
Classification: Positive

--- Tweet 7 ---
Original: 'What a beautiful sunset! Absolutely breathtaking. 😍'
```

```
Cleaned: 'beautiful sunset absolutely breathtaking'
Sentiment Score: 0.85
Classification: Positive

--- Tweet 8 ---
Original: 'Feeling a bit down today. Nothing seems to go right.'
Cleaned: 'feeling bit today nothing seems go right'
Sentiment Score: -0.29
Classification: Negative

--- Sentiment Summary ---
Positive: 4 tweets (50.00%)
Negative: 4 tweets (50.00%)
Neutral: 0 tweets (0.00%)
```

Lab 12: Case Study: Designing a Chatbot for Customer Service Queries

Title: Case Study: Designing a Rule-Based Chatbot for Customer Service Queries

Aim: To design and implement a simple rule-based chatbot that can respond to common customer service queries, demonstrating basic natural language understanding through keyword matching.

Procedure:

1. **Define Knowledge Base:** Create a dictionary or a list of tuples to store predefined rules, mapping keywords or query patterns to specific responses.
2. **Preprocessing User Input:** Implement a function to preprocess user input (e.g., convert to lowercase, remove punctuation) to facilitate easier matching.
3. **Matching Logic:**
    - Implement a function `get_chatbot_response` that takes the preprocessed user query.
    - Iterate through the knowledge base. If any keyword from a rule is found in the user's query, return the corresponding response.
    - Include a default fallback response for unmatched queries.
4. **Chat Loop:** Create a simple loop to allow continuous interaction with the chatbot.

Source Code:

```
import re

def preprocess_query(query):
    """
    Converts query to lowercase and removes punctuation for easier matching.
    """
    query = query.lower()
    query = re.sub(r'[^\w\s]', '', query) # Remove punctuation
    return query

# Define the chatbot's knowledge base (rules and responses)
# Each entry is a tuple: (list_of_keywords, response_string)
knowledge_base = [
    (["hello", "hi", "hey"], "Hello! How can I assist you today?"),
    (["order status", "my order", "where is my order"], "To check your order
status, please provide your order number."),
    (["return policy", "return an item", "refund"], "Our return policy allows
returns within 30 days of purchase. Please visit our returns page for more
details."),
    (["contact support", "speak to someone", "customer service"], "You can reach
our customer support team at 1-800-555-1234 or email support@example.com."),
    (["shipping", "delivery time"], "Standard shipping usually takes 3-5
business days. Expedited options are available at checkout."),
    (["payment methods", "pay with", "credit card"], "We accept major credit
cards (Visa, MasterCard, Amex), PayPal, and bank transfers."),
    (["thank you", "thanks"], "You're welcome! Is there anything else I can help
you with?"),
    (["hours", "open", "close"], "Our business hours are Monday to Friday, 9 AM
to 5 PM local time."),
    (["cancel order", "cancel my order"], "To cancel an order, please provide
your order number and reason for cancellation. Note that orders already shipped
cannot be cancelled.")
]
```

```
def get_chatbot_response(user_query):
    """
    Matches the user's query against the knowledge base and returns a response.
    """
    processed_query = preprocess_query(user_query)

    for keywords, response in knowledge_base:
        # Check if any keyword from the rule is present in the processed query
        if any(keyword in processed_query for keyword in keywords):
            return response

    # Default fallback response if no match is found
    return "I'm sorry, I don't understand that query. Could you please rephrase
or ask something else?"

# --- Chatbot Interaction Loop ---
print("--- Customer Service Chatbot ---")
print("Type 'quit' or 'exit' to end the conversation.")

while True:
    user_input = input("\nYou: ")
    if user_input.lower() in ["quit", "exit"]:
        print("Chatbot: Goodbye! Have a great day.")
        break

    response = get_chatbot_response(user_input)
    print(f"Chatbot: {response}")
```

Input: User inputs during the chat loop (e.g., "Hi", "What is my order status?", "How do I return an item?", "Thanks", "Can I pay with credit card?", "quit").

Expected Output:

```
--- Customer Service Chatbot ---
Type 'quit' or 'exit' to end the conversation.

You: Hello
Chatbot: Hello! How can I assist you today?

You: I want to know my order status.
Chatbot: To check your order status, please provide your order number.

You: What is your return policy?
Chatbot: Our return policy allows returns within 30 days of purchase. Please
visit our returns page for more details.

You: How can I contact support?
Chatbot: You can reach our customer support team at 1-800-555-1234 or email
support@example.com.

You: Thanks!
Chatbot: You're welcome! Is there anything else I can help you with?

You: What are your business hours?
Chatbot: Our business hours are Monday to Friday, 9 AM to 5 PM local time.

You: I need to cancel my order.
Chatbot: To cancel an order, please provide your order number and reason for
cancellation. Note that orders already shipped cannot be cancelled.

You: Do you accept payment by credit card?
Chatbot: We accept major credit cards (Visa, MasterCard, Amex), PayPal, and bank
transfers.
```

You: I have a very specific question about product XYZ.
Chatbot: I'm sorry, I don't understand that query. Could you please rephrase or ask something else?

You: quit
Chatbot: Goodbye! Have a great day.

Lab 13: Case Study: Building a Question Answering System for Wikipedia Articles

Title: Case Study: Building a Basic Question Answering System

Aim: To build a basic question answering system that can extract relevant sentences or phrases from a given text passage (simulating a Wikipedia article) in response to a user's question.

Procedure:

1. **Import Libraries:** Import `spacy` for NLP processing, `re` for cleaning, and `cosine_similarity`, `TfidfVectorizer` for relevance scoring.
2. **Define Text Passage:** Provide a long text passage (simulating a Wikipedia article).
3. **Preprocessing:**
    - Clean the text (remove extra spaces, normalize).
    - Segment the text into sentences using SpaCy.
4. **Vectorization:**
    - Use `TfidfVectorizer` to convert sentences into numerical vectors.
5. **Answer Retrieval:**
    - Define a function `get_answer` that takes the question and the text passage.
    - Preprocess the question and convert it into a TF-IDF vector.
    - Calculate cosine similarity between the question vector and each sentence vector.
    - Identify the sentence(s) with the highest similarity score.
    - Return the most relevant sentence as the answer.

Source Code:

```
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

# Load SpaCy English model
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    print("SpaCy model 'en_core_web_sm' not found. Please run: python -m spacy
download en_core_web_sm")
    exit()

# Sample text passage (simulating a Wikipedia article)
# Source: Adapted from Wikipedia articles on 'Artificial Intelligence' and
'Machine Learning'
article_text = """
Artificial intelligence (AI) is intelligence demonstrated by machines, unlike
the natural intelligence displayed by humans and animals.
Leading AI textbooks define the field as the study of "intelligent agents": any
device that perceives its environment and takes actions that maximize its chance
of successfully achieving its goals.
The term "artificial intelligence" was coined in 1956 by John McCarthy at the
Dartmouth Conference.
AI research has been defined as the field of study for "acting rationally" or
"thinking humanly."
Machine learning is a subset of artificial intelligence that provides systems
the ability to automatically learn and improve from experience without being
explicitly programmed.
```

```
It focuses on the development of computer programs that can access data and use
it learn for themselves.
Deep learning is a specialized field of machine learning that employs neural
networks with many layers to learn complex patterns in data.
"""

def preprocess_text(text):
    """Basic text cleaning and normalization."""
    text = text.lower()
    text = re.sub(r'\s+', ' ', text).strip() # Replace multiple spaces with
single space
    return text

def get_sentences(text):
    """Segments text into sentences using SpaCy."""
    doc = nlp(text)
    return [sent.text for sent in doc.sents]

def get_answer(question, context_text):
    """
    Retrieves the most relevant sentence from the context for a given question.
    """
    # 1. Preprocess the context and segment into sentences
    cleaned_context = preprocess_text(context_text)
    sentences = get_sentences(cleaned_context)

    if not sentences:
        return "No sentences found in the context."

    # 2. Initialize TF-IDF Vectorizer
    # We use a shared vectorizer for both question and sentences
    tfidf_vectorizer = TfidfVectorizer(stop_words='english')

    # Fit on all sentences and transform them
    sentence_vectors = tfidf_vectorizer.fit_transform(sentences)

    # 3. Preprocess the question and transform it into a TF-IDF vector
    cleaned_question = preprocess_query(question) # Reusing preprocess_query
from Lab 12
    question_vector = tfidf_vectorizer.transform([cleaned_question])

    # 4. Calculate cosine similarity between question and each sentence
    similarities = cosine_similarity(question_vector,
sentence_vectors).flatten()

    # 5. Find the index of the most similar sentence
    most_similar_idx = similarities.argmax()
    highest_similarity_score = similarities[most_similar_idx]

    if highest_similarity_score > 0: # Only return if there's some similarity
        return sentences[most_similar_idx]
    else:
        return "I could not find a relevant answer in the provided text."

# --- Test Cases ---
print("--- Basic Question Answering System ---")
print("\nContext Article:\n", article_text)

questions = [
    "What is artificial intelligence?",
    "Who coined the term 'artificial intelligence'?",
    "What is machine learning?",
    "What is deep learning?",
    "When was AI coined?",
    "What is the capital of France?" # Irrelevant question
]
```

```
for question in questions:
    answer = get_answer(question, article_text)
    print(f"\nQ: {question}")
    print(f"A: {answer}")
```

Input: A text passage (simulated Wikipedia article) and a list of questions as defined in the source code.

Expected Output:

```
--- Basic Question Answering System ---

Context Article:
Artificial intelligence (AI) is intelligence demonstrated by machines, unlike
the natural intelligence displayed by humans and animals.
Leading AI textbooks define the field as the study of "intelligent agents": any
device that perceives its environment and takes actions that maximize its chance
of successfully achieving its goals.
The term "artificial intelligence" was coined in 1956 by John McCarthy at the
Dartmouth Conference.
AI research has been defined as the field of study for "acting rationally" or
"thinking humanly."
Machine learning is a subset of artificial intelligence that provides systems
the ability to automatically learn and improve from experience without being
explicitly programmed.
It focuses on the development of computer programs that can access data and use
it learn for themselves.
Deep learning is a specialized field of machine learning that employs neural
networks with many layers to learn complex patterns in data.

Q: What is artificial intelligence?
A: artificial intelligence (ai) is intelligence demonstrated by machines, unlike
the natural intelligence displayed by humans and animals.

Q: Who coined the term 'artificial intelligence'?
A: the term "artificial intelligence" was coined in 1956 by john mccarthy at the
dartmouth conference.

Q: What is machine learning?
A: machine learning is a subset of artificial intelligence that provides systems
the ability to automatically learn and improve from experience without being
explicitly programmed.

Q: What is deep learning?
A: deep learning is a specialized field of machine learning that employs neural
networks with many layers to learn complex patterns in data.

Q: When was AI coined?
A: the term "artificial intelligence" was coined in 1956 by john mccarthy at the
dartmouth conference.

Q: What is the capital of France?
A: i could not find a relevant answer in the provided text.
```