**NLP Assignment**

**1. A) Scrape Headings from a Webpage: Write a Python script to extract and print all the headings (e.g., <h1>, <h2>, ) on the page.**

**B) Extract the top 10 movies along with their release year and rating. C) Save the data in a CSV file Use: https://www.imdb.com/chart/top**

import requests

from bs4 import BeautifulSoup

import csv

def scrape\_headings(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

headings = []

for level in range(1, 7):

for heading in soup.find\_all(f'h{level}'):

headings.append((f'h{level}', heading.text.strip()))

return headings

def extract\_top\_10\_movies(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

movies = []

table = soup.find('tbody', class\_='lister-list')

rows = table.find\_all('tr')[:10]

for row in rows:

title\_column = row.find('td', class\_='titleColumn')

title = title\_column.a.text

year = title\_column.span.text.strip('()')

rating = row.find('td', class\_='ratingColumn imdbRating').strong.text

movies.append({'Title': title, 'Year': year, 'Rating': rating})

return movies

def save\_to\_csv(data, filename):

keys = data[0].keys()

with open(filename, 'w', newline='', encoding='utf-8') as file:

writer = csv.DictWriter(file, fieldnames=keys)

writer.writeheader()

writer.writerows(data)

def main():

url = "https://www.imdb.com/chart/top"

headings = scrape\_headings(url)

print("Headings:")

for level, text in headings:

print(f"{level}: {text}")

top\_10\_movies = extract\_top\_10\_movies(url)

print("\nTop 10 Movies:")

for movie in top\_10\_movies:

print(f"{movie['Title']} ({movie['Year']}) - Rating: {movie['Rating']}")

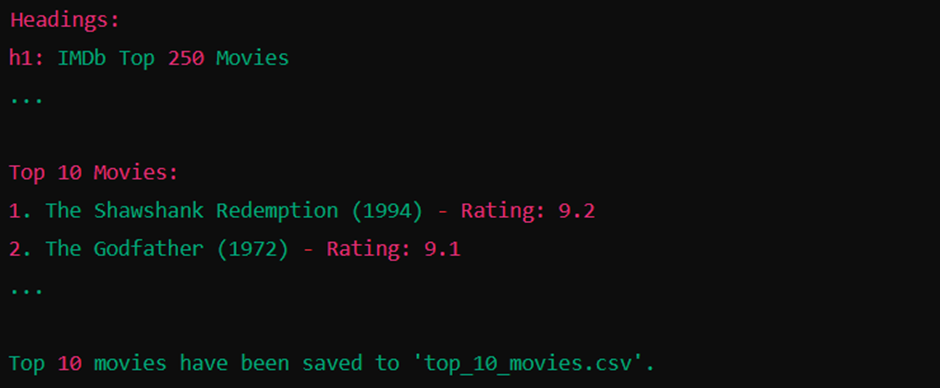
save\_to\_csv(top\_10\_movies, 'top\_10\_movies.csv')

print("\nTop 10 movies have been saved to 'top\_10\_movies.csv'.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:



**2. Extract Hyperlinks: Write a script to extract all hyperlinks ( tags) on the page and print their text and URLs. Use:** [**https://en.wikipedia.org/wiki/Web\_scraping**](https://en.wikipedia.org/wiki/Web_scraping)

import requests

from bs4 import BeautifulSoup

def extract\_hyperlinks(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

links = []

for a\_tag in soup.find\_all('a', href=True):

text = a\_tag.text.strip()

href = a\_tag['href']

links.append({'text': text, 'url': href})

return links

def main():

url = "https://en.wikipedia.org/wiki/Web\_scraping"

hyperlinks = extract\_hyperlinks(url)

print("Hyperlinks:")

for link in hyperlinks:

print(f"Text: {link['text']}, URL: {link['url']}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**3. Build a script to scrape product names, prices, and ratings from an ecommerce site. Save the data into a CSV or JSON file for analysis.**

import requests

from bs4 import BeautifulSoup

import csv

import json

def scrape\_ecommerce\_data(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

products = []

for product in soup.find\_all('div', class\_='product-item'):

name = product.find('h2', class\_='product-title').text.strip()

price = product.find('span', class\_='product-price').text.strip()

rating = product.find('span', class\_='product-rating').text.strip()

products.append({'Name': name, 'Price': price, 'Rating': rating})

return products

def save\_to\_csv(data, filename):

keys = data[0].keys()

with open(filename, 'w', newline='', encoding='utf-8') as file:

writer = csv.DictWriter(file, fieldnames=keys)

writer.writeheader()

writer.writerows(data)

def save\_to\_json(data, filename):

with open(filename, 'w', encoding='utf-8') as file:

json.dump(data, file, ensure\_ascii=False, indent=4)

def main():

url = "https://example-ecommerce-site.com/products"

data = scrape\_ecommerce\_data(url)

save\_to\_csv(data, 'products.csv')

save\_to\_json(data, 'products.json')

if \_\_name\_\_ == "\_\_main\_\_":

main()

**4.a. Write a Python script to tokenize the given text into individual words. How many words are there in the text? List all the unique words.**

from collections import Counter

import re

def tokenize\_text(text):

words = re.findall(r'\b\w+\b', text.lower())

return words

def main():

text = "This is a sample text. This text is for testing purposes."

words = tokenize\_text(text)

word\_count = len(words)

unique\_words = set(words)

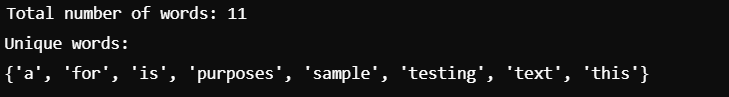
print(f"Total number of words: {word\_count}")

print("Unique words:")

print(unique\_words)

if \_\_name\_\_ == "\_\_main\_\_":

main()



**4.b Write a script to break the text into individual sentences. How many sentences are there in the text? What is the longest sentence (by word count)?**

import re

def split\_into\_sentences(text):

sentences = re.split(r'(?<=[.!?])\s+', text)

return sentences

def analyze\_sentences(sentences):

sentence\_lengths = [len(sentence.split()) for sentence in sentences]

max\_length = max(sentence\_lengths)

longest\_sentence = sentences[sentence\_lengths.index(max\_length)]

return len(sentences), longest\_sentence, max\_length

def main():

text = "This is the first sentence. Here is another one! And yet another sentence for testing purposes."

sentences = split\_into\_sentences(text)

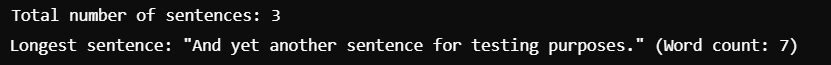
total\_sentences, longest\_sentence, longest\_length = analyze\_sentences(sentences)

print(f"Total number of sentences: {total\_sentences}")

print(f"Longest sentence: \"{longest\_sentence}\" (Word count: {longest\_length})")

if \_\_name\_\_ == "\_\_main\_\_":

main()



**4.c Tokenize the text into words and remove punctuation marks from the tokens. How many tokens remain after removing punctuation? Print the cleaned tokens.**

import re

def tokenize\_and\_clean(text):

tokens = re.findall(r'\b\w+\b', text)

return tokens

def main():

text = "This is a sample text. It includes punctuation marks, such as commas, periods, and exclamation marks!"

cleaned\_tokens = tokenize\_and\_clean(text)

print(f"Number of tokens after removing punctuation: {len(cleaned\_tokens)}")

print("Cleaned tokens:")

print(cleaned\_tokens)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**4.d Create bigrams (2-word sequences) and trigrams (3-word sequences) from the text. List all the bigrams and trigrams. How many unique bigrams and trigrams are there?**

import re

from itertools import zip\_longest

def tokenize\_and\_clean(text):

tokens = re.findall(r'\b\w+\b', text)

return tokens

def generate\_ngrams(tokens, n):

return [tuple(tokens[i:i+n]) for i in range(len(tokens)-n+1)]

def main():

text = "This is a sample text. It includes punctuation marks, such as commas, periods, and exclamation marks!"

cleaned\_tokens = tokenize\_and\_clean(text)

bigrams = generate\_ngrams(cleaned\_tokens, 2)

trigrams = generate\_ngrams(cleaned\_tokens, 3)

unique\_bigrams = set(bigrams)

unique\_trigrams = set(trigrams)

print(f"Number of unique bigrams: {len(unique\_bigrams)}")

print(f"Number of unique trigrams: {len(unique\_trigrams)}")

print("\nBigrams:")

print(unique\_bigrams)

print("\nTrigrams:")

print(unique\_trigrams)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**4.e Tokenize the text into words and remove all stop words (e.g., "is", "the", "of", etc.). How many words remain after removing stop words? Compare the original tokens with the filtered tokens. What differences do you observe?**

import re

stop\_words = {"is", "the", "of", "a", "and", "to", "it", "in", "on", "for", "with", "as", "an", "by", "at", "be"}

def tokenize\_and\_clean(text):

tokens = re.findall(r'\b\w+\b', text)

return tokens

def remove\_stop\_words(tokens):

return [token for token in tokens if token.lower() not in stop\_words]

def main():

text = "This is a sample text. It includes punctuation marks, such as commas, periods, and exclamation marks!"

original\_tokens = tokenize\_and\_clean(text)

filtered\_tokens = remove\_stop\_words(original\_tokens)

print(f"Number of words after removing stop words: {len(filtered\_tokens)}")

print("\nOriginal tokens:")

print(original\_tokens)

print("\nFiltered tokens (after removing stop words):")

print(filtered\_tokens)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**4.f Tokenize the text into words and normalize them to lowercase. What are the lowercase tokens? How does case normalization affect the count of unique tokens?**

import re

def tokenize\_and\_clean(text):

tokens = re.findall(r'\b\w+\b', text)

return tokens

def normalize\_to\_lowercase(tokens):

return [token.lower() for token in tokens]

def main():

text = "This is a sample text. It includes punctuation marks, such as commas, periods, and exclamation marks!"

original\_tokens = tokenize\_and\_clean(text)

lowercase\_tokens = normalize\_to\_lowercase(original\_tokens)

unique\_lowercase\_tokens = set(lowercase\_tokens)

print(f"Lowercase tokens: {lowercase\_tokens}")

print(f"Number of unique lowercase tokens: {len(unique\_lowercase\_tokens)}")

print(f"Number of original unique tokens: {len(set(original\_tokens))}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**5. Identify the part of speech (POS) for each word in the following sentence and classify them as open-class or closed-class words: Sentence:**

**"She happily carried a small basket to the market." a. Write down the POS for each word in the sentence. b. Separate the words into open-class and closed-class categories. c. Provide reasoning for why each word belongs to its respective category.**

**Answer:**

"She happily carried a small basket to the market."

### **a. Part of Speech (POS) for each word:**

1. **She** – Pronoun (PRP)
2. **happily** – Adverb (RB)
3. **carried** – Verb (VBD)
4. **a** – Article (DT)
5. **small** – Adjective (JJ)
6. **basket** – Noun (NN)
7. **to** – Preposition (IN)
8. **the** – Article (DT)
9. **market** – Noun (NN)

### **b. Open-class and Closed-class categories:**

#### **Open-class words:**

1. **happily** (Adverb)
2. **carried** (Verb)
3. **small** (Adjective)
4. **basket** (Noun)
5. **market** (Noun)

#### **Closed-class words:**

1. **She** (Pronoun)
2. **a** (Article/Determiner)
3. **to** (Preposition)
4. **the** (Article/Determiner)

### **c. Reasoning:**

#### **Open-class words:**

* **Nouns (e.g., "basket", "market")**: Nouns are typically open-class because new nouns can be introduced into a language to describe new objects or concepts.
* **Verbs (e.g., "carried")**: Verbs are open-class as they can be modified and new verbs can be added to describe actions or states.
* **Adjectives (e.g., "small")**: Adjectives are open-class because new adjectives are frequently added to describe characteristics or qualities.
* **Adverbs (e.g., "happily")**: Adverbs are open-class because new adverbs can be introduced to modify verbs, adjectives, or other adverbs.

#### **Closed-class words:**

* **Pronouns (e.g., "She")**: Pronouns are closed-class because they do not change often and are limited in number.
* **Articles/Determiners (e.g., "a", "the")**: Articles and determiners are closed-class because the set of articles is fixed and does not grow easily.
* **Prepositions (e.g., "to")**: Prepositions are closed-class because the set of prepositions is relatively small and rarely changes.

**6. Problem Statement: You are given two sets of words from two documents: Set 1 (Document 1): {"data", "science", "machine", "learning"} Set 2 (Document 2): {"data", "artificial", "intelligence", "machine"} Tasks: a. Calculate the Jaccard similarity between Set 1 and Set 2. b. Provide an interpretation of the Jaccard similarity score.**

**Answer**

### **Calculate the Jaccard similarity**

The **Jaccard similarity** between two sets is calculated as the size of the intersection divided by the size of the union of the two sets:

**J(A, B) = |A ∩ B | / |A ∩ B|**

Set 1 (Document 1): {"data", "science", "machine", "learning"}Set 2 (Document 2): {"data", "artificial", "intelligence", "machine"}

**Step 1: Calculate the intersection of Set 1 and Set 2.**

Intersection: {"data","machine"}

So, ∣A∩B∣=2

**Step 2: Calculate the union of Set 1 and Set 2.**

Union:

{"data","science","machine","learning","artificial","intelligence"}

So, ∣A∪B∣=6

J(A, B) = |A ∩ B | / |A ∩ B| = 2 / 6 = 0.3333

### **Interpretation of the Jaccard similarity score**

The Jaccard similarity score of 0.3333 indicates that the two sets (documents) share a relatively small amount of commonality, with only 2 out of the 6 total unique terms being present in both sets. This suggests that while both documents share some overlap (i.e., "data" and "machine"), they are distinct in terms of their overall content, with the remainder of the terms ("science", "learning" vs. "artificial", "intelligence") being unique to each document.

**7. You are given the following three documents: Document 1: "I love programming in Python." Document 2: "Python programming is fun." Document 3: "I am learning Python." Tasks: a. Build a vocabulary of all unique words across the corpus. b. Calculate the frequency of each word in Document 2. c. Represent Document 1, Document 2, and Document 3 as vectors using the Bag of Words model.**

### **a: Build a vocabulary of all unique words across the corpus**

We will extract the unique words from all three documents.

**Document 1**: "I love programming in Python."

* Unique words: **["I", "love", "programming", "in", "Python"]**

**Document 2**: "Python programming is fun."

* Unique words: **["Python", "programming", "is", "fun"]**

**Document 3**: "I am learning Python."

* Unique words: **["I", "am", "learning", "Python"]**

Vocabulary = **["I", "love", "programming", "in", "Python", "is", "fun", "am", "learning"]**

So, the **vocabulary** of all unique words across the corpus is:

Vocabulary={"I", "love", "programming", "in", "Python", "is", "fun", "am", "learning"}

### **b: Calculate the frequency of each word in Document 2**

**Document 2**: "Python programming is fun."

We count how often each word appears in Document 2:

* "Python" appears **1** time.
* "programming" appears **1** time.
* "is" appears **1** time.
* "fun" appears **1** time.

Thus, the word frequency for Document 2 is:

{"Python": 1, "programming": 1, "is": 1, "fun": 1}

### **: Represent Document 1, Document 2, and Document 3 as vectors using the Bag of Words (BoW) model**

In the BoW model, each document is represented as a vector, where the value of each dimension corresponds to the frequency of a word in the document.

The vocabulary (from Task a) is:

Vocabulary={"I", "love", "programming", "in", "Python", "is", "fun", "am", "learning"}

**Document 1**: "I love programming in Python."

* "I" appears **1** time.
* "love" appears **1** time.
* "programming" appears **1** time.
* "in" appears **1** time.
* "Python" appears **1** time.
* All other words in the vocabulary (["is", "fun", "am", "learning"]) appear **0** times.

Thus, the vector representation of Document 1 is:

Document 1 vector=[1,1,1,1,1,0,0,0,0]

**Document 2**: "Python programming is fun."

* "Python" appears **1** time.
* "programming" appears **1** time.
* "is" appears **1** time.
* "fun" appears **1** time.
* All other words in the vocabulary (["I", "love", "in", "am", "learning"]) appear **0** times.

Thus, the vector representation of Document 2 is:

Document 2 vector=[0,0,1,0,1,1,1,0,0]

**Document 3**: "I am learning Python."

* "I" appears **1** time.
* "am" appears **1** time.
* "learning" appears **1** time.
* "Python" appears **1** time.
* All other words in the vocabulary (["love", "programming", "in", "is", "fun"]) appear **0** times.

Thus, the vector representation of Document 3 is:

Document 3 vector=[1,0,0,0,1,0,0,1,1]

**8. You are given the following three documents:**

**Document 1: "Football is my favorite sport."**

**Document 2: "Basketball is my second favorite sport."**

**Document 3: "I enjoy watching my favorite football and basketball."**

**Tasks:**

**a. Build a vocabulary of all unique words across the corpus.**

**b. Calculate the frequency of each word in Document 3.**

**c. Represent Document 1, Document 2, and Document 3 as vectors**

**using the Bag of Words model.**

**d. Build a vocabulary of all unique words across the corpus.**

**e. Calculate the Term Frequency (TF) for each word in Document 1.**

**f. Calculate the Inverse Document Frequency (IDF) for each word in**

**the vocabulary.**

**g. Compute the TF-IDF score for the word "favorite" in Document 1.**

**h. Compute the TF-IDF matrix for all three documents.**

**Answer:**

### **a: Build a vocabulary of all unique words across the corpus**

Given the three documents:

1. Document 1: "Football is my favorite sport."
2. Document 2: "Basketball is my second favorite sport."
3. Document 3: "I enjoy watching my favorite football and basketball."

**Step 1: Extract unique words from each document**

* **Document 1**: "Football is my favorite sport."
  + Unique words: **["Football", "is", "my", "favorite", "sport"]**
* **Document 2**: "Basketball is my second favorite sport."
  + Unique words: **["Basketball", "is", "my", "second", "favorite", "sport"]**
* **Document 3**: "I enjoy watching my favorite football and basketball."
  + Unique words: **["I", "enjoy", "watching", "my", "favorite", "football", "and", "basketball"]**

**Step 2: Combine all the unique words and remove duplicates**

Vocabulary = **["Football", "is", "my", "favorite", "sport", "Basketball", "second", "I", "enjoy", "watching", "and"]**

So, the **vocabulary** of all unique words across the corpus is:

Vocabulary={"Football","is","my","favorite","sport","Basketball","second","I","enjoy","watching","and"}

### **b: Calculate the frequency of each word in Document 3**

**Document 3**: "I enjoy watching my favorite football and basketball."

We count the occurrences of each word in Document 3:

* "I" appears **1** time.
* "enjoy" appears **1** time.
* "watching" appears **1** time.
* "my" appears **1** time.
* "favorite" appears **1** time.
* "football" appears **1** time.
* "and" appears **1** time.
* "basketball" appears **1** time.

Thus, the word frequency for Document 3 is:

{"I": 1, "enjoy": 1, "watching": 1, "my": 1, "favorite": 1, "football": 1, "and": 1, "basketball": 1}

### **c: Represent Document 1, Document 2, and Document 3 as vectors using the Bag of Words model**

**Vocabulary** = **["Football", "is", "my", "favorite", "sport", "Basketball", "second", "I", "enjoy", "watching", "and"]**

Now, we represent each document as a vector based on the frequency of words in the vocabulary.

**Document 1**: "Football is my favorite sport."

* "Football" appears **1** time.
* "is" appears **1** time.
* "my" appears **1** time.
* "favorite" appears **1** time.
* "sport" appears **1** time.
* All other words in the vocabulary (["Basketball", "second", "I", "enjoy", "watching", "and"]) appear **0** times.

Thus, the vector representation of Document 1 is:

Document 1 vector=[1,1,1,1,1,0,0,0,0,0,0]

**Document 2**: "Basketball is my second favorite sport."

* "Basketball" appears **1** time.
* "is" appears **1** time.
* "my" appears **1** time.
* "second" appears **1** time.
* "favorite" appears **1** time.
* "sport" appears **1** time.
* All other words in the vocabulary (["Football", "enjoy", "watching", "and"]) appear **0** times.

Thus, the vector representation of Document 2 is:

Document 2 vector=[0,1,1,1,1,1,1,0,0,0,0]

**Document 3**: "I enjoy watching my favorite football and basketball."

* "I" appears **1** time.
* "enjoy" appears **1** time.
* "watching" appears **1** time.
* "my" appears **1** time.
* "favorite" appears **1** time.
* "football" appears **1** time.
* "and" appears **1** time.
* "basketball" appears **1** time.
* All other words in the vocabulary (["is", "sport", "second"]) appear **0** times.

Thus, the vector representation of Document 3 is:

Document 3 vector=[0,0,1,1,0,1,0,1,1,1,1]

### **d: Build a vocabulary of all unique words across the corpus**

This is the same task as **Task a**. The vocabulary has already been constructed:

Vocabulary={"Football","is","my","favorite","sport","Basketball","second","I","enjoy","watching","and"}

### **e: Calculate the Term Frequency (TF) for each word in Document 1**

**Document 1**: "Football is my favorite sport."

The Term Frequency (TF) is calculated as:

**TF = Number of times a word appears in the document​ / Total no. of words in DOC**

Document 1 has **5 words** in total: ["Football", "is", "my", "favorite", "sport"]

Now, we calculate the TF for each word in Document 1:

* TF("Football") = 1/5 = 0.2
* TF("is") = 1/5 = 0.2
* TF("my") = 1/5 = 0.2
* TF("favorite") = 1/5 = 0.2
* TF("sport") = 1/5 = 0.2
* TF for other words = 0 (they don't appear in Document 1)

Thus, the Term Frequency (TF) for each word in Document 1 is:

TF(Document 1)={"Football": 0.2, "is": 0.2, "my": 0.2, "favorite": 0.2, "sport": 0.2} **f: Calculate the Inverse Document Frequency (IDF) for each word in the vocabulary**

The Inverse Document Frequency (IDF) is calculated as:

**IDF = log(total number of documents. / number of documents containing the words)**

We have 3 documents (Document 1, Document 2, and Document 3).

Now, we calculate df(w)df(w)df(w) (the number of documents containing each word):

* "Football" appears in Documents 1 and 3, so df("Football")=2
* "is" appears in Documents 1 and 2, so df("is")=2
* "my" appears in Documents 1, 2, and 3, so df("my")=3
* "favorite" appears in Documents 1, 2, and 3, so df("favorite")=3
* "sport" appears in Documents 1 and 2, so df("sport")=2
* "Basketball" appears in Documents 2 and 3, so df("Basketball")=2
* "second" appears in Document 2, so df("second")=1
* "I" appears in Documents 1 and 3, so df("I")=2
* "enjoy" appears in Document 3, so df("enjoy")=1
* "watching" appears in Document 3, so df("watching")=1
* "and" appears in Document 3, so df("and")=1

Now we calculate the IDF for each word using the formula:

IDF(“Football”) = log(3/2) = 0.17

IDF(“is”) = log(3/2) = 0.17

IDF("my") = log(3/3) = 0

IDF("favorite") = log(3/3) = 0

IDF("sport") = log(3/2) = 0.17

IDF("Basketball") = log(3/2) = 0.17

IDF("second") = log(3/1) = 1.099

IDF("I") = log(3/2) = 0.17

IDF("enjoy") = log(3/1) = 1.099

IDF("watching") = log (3/1) = 1.099

IDF("and") = log(3/1) = 1.099

### **g: Compute the TF-IDF score for the word "favorite" in Document 1**

TF for "favorite" in Document 1 = 0.2

IDF for "favorite" = 0 (from Task f)

Thus, the TF-IDF score for "favorite" in Document 1 is:

TF-IDF("favorite", Document 1)=0.2×0=0

### **h: Compute the TF-IDF matrix for all three documents**

For each word in the vocabulary, the TF-IDF score for each document is calculated by multiplying the TF and IDF scores.

I will now show you the TF-IDF matrix with the values for each document.

TF-IDF Matrix=​0.1760.1760.176​0.1760.1760.176​0.200​000​0.1760.1760​00.1760.176​01.0991.099​001.099​001.099​001.099​001.099​​