**Imports**

import requests

from bs4 import BeautifulSoup

import pandas as pd

import os  
  
 requests: Used to make HTTP requests to web pages.

BeautifulSoup: A library for parsing HTML and XML documents.

pandas: Used for data manipulation and analysis, particularly for handling Excel files.

os: Provides functions for interacting with the operating system, like creating directories.

Function Definition: extract\_article\_text(url)  
  
def extract\_article\_text(url):

try:

response = requests.get(url)

if response.status\_code == 200:

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

# Extract article title

title\_element = soup.find('h1', class\_='title')

title = title\_element.text.strip() if title\_element else "No Title Found"

# Extract article content

article\_content = soup.find('div', class\_='td-post-content tagdiv-type')

if article\_content:

paragraphs = article\_content.find\_all('p')

article\_text = '\n'.join([p.text.strip() for p in paragraphs])

else:

article\_text = "No Content Found"

return title, article\_text

else:

print(f"Failed to retrieve {url}. Status code: {response.status\_code}")

return None, None

except Exception as e:

print(f"Exception occurred while processing {url}: {str(e)}")

return None, None  
  
 Purpose: Extracts the article title and content from a given URL.

 Steps:

* Sends an HTTP GET request to the URL.
* If the request is successful (status\_code == 200):
  + Parses the HTML content using BeautifulSoup.
  + Finds the title (<h1> tag with class 'title') and extracts its text.
  + Finds the article content (<div> tag with class 'td-post-content tagdiv-type') and extracts all paragraphs (<p> tags).
  + Joins the text of paragraphs into a single string (article\_text).
* If any step fails (e.g., URL not found, parsing issues), it catches the exception and returns None.

**Main Execution**

**excel\_file = 'input.xlsx'**

**try:**

**df = pd.read\_excel(excel\_file, engine='openpyxl')**

**except Exception as e:**

**print(f"Error reading {excel\_file}: {str(e)}")**

**exit()**

**for index, row in df.iterrows():**

**url\_id = row['URL\_ID']**

**url = row['URL']**

**# Extract article content**

**title, article\_text = extract\_article\_text(url)**

**if title and article\_text:**

**# Create directory to store text files if it doesn't exist**

**output\_dir = 'extracted\_articles'**

**if not os.path.exists(output\_dir):**

**os.makedirs(output\_dir)**

**# Write extracted content to a text file**

**filename = f"{output\_dir}/{url\_id}.txt"**

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

**f.write(f"{title}\n\n{article\_text}")**

**print(f"Extracted and saved article from {url} to {filename}")**

**else:**

**print(f"Skipping {url} due to extraction issues")** **Purpose**: Reads URLs from an Excel file (input.xlsx), extracts article content from each URL using extract\_article\_text, and saves the extracted content into text files.

 **Steps**:

* Reads the Excel file into a pandas DataFrame (df).
* Iterates over each row in the DataFrame:
  + Retrieves url\_id and url from each row.
  + Calls extract\_article\_text(url) to get title and article\_text.
  + Checks if both title and article\_text are not None (indicating successful extraction).
  + Creates a directory (extracted\_articles) if it doesn't exist.
  + Writes the extracted content (title and article\_text) to a text file ({output\_dir}/{url\_id}.txt).
  + Prints messages indicating success or failure in extraction.

**-----------------------------------------------------------------------------------------------------------------------------------------------------**

**pip install chardet**

**1. Understanding pip**

pip is the package installer for Python. It allows you to install and manage Python packages that are listed on the Python Package Index (PyPI) and other package indexes.

**2. Command Structure**

The command pip install chardet follows this structure:

* pip: Calls the pip command-line tool.
* install: Specifies that you want to install a package.
* chardet: The name of the package you want to install. In this case, it's chardet. chardet is a Python library used for character encoding detection.

**3. Installation Process**

When you execute pip install chardet, here's what happens:

* **Network Connection**: Pip connects to the PyPI (Python Package Index) repository or another repository (if specified) over the internet.
* **Package Retrieval**: Pip retrieves the latest version of the chardet package and its dependencies (if any) from the repository. It checks for compatibility with your Python version and operating system.
* **Dependency Resolution**: If chardet depends on other packages, pip will also download and install those dependencies recursively.
* **Installation**: Pip installs chardet and any required dependencies into your Python environment. This typically involves copying files to the appropriate directories where Python can find them.
* **Confirmation**: Once installation is complete, pip outputs messages confirming the installation was successful. It may also display warnings or errors if there were issues during installation.

**4. Post-Installation**

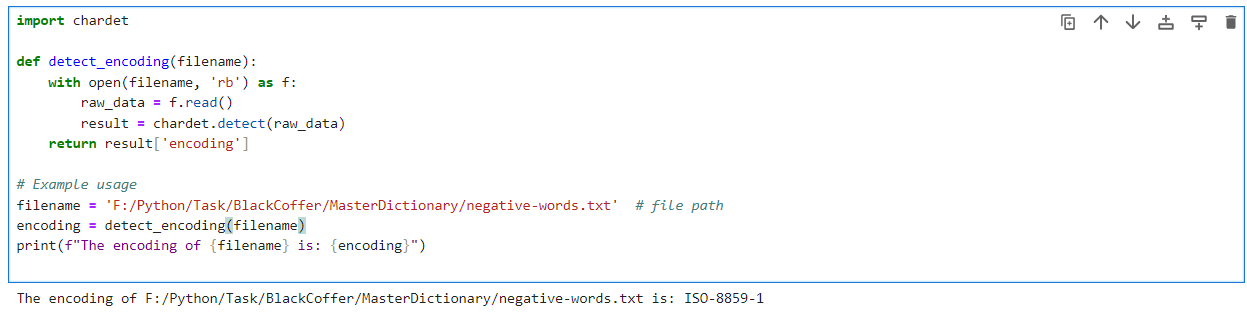
After chardet is successfully installed:

* You can import chardet in your Python scripts or interactive sessions to use its functionality.
* It becomes available for use in detecting character encodings of byte data, which is especially useful when working with text data from different sources or encodings.

**5. Version Management**

* Pip installs the latest version of chardet by default. If you need a specific version or want to upgrade/downgrade later, pip commands like pip install chardet==3.0.4 (installing version 3.0.4) or pip install --upgrade chardet (upgrading to the latest version) can be used.

**---------------------------------------------------------------------------------------------------------------------------------------------------**



**1. Importing the chardet Module**

**Purpose**: This line imports the chardet module into your Python script. chardet is a library used for automatic detection of character encodings in byte strings.

**2. detect\_encoding Function Definition**

**Purpose**: This function reads the content of a file and detects its encoding using chardet.

**Parameters**:

filename: Path to the file whose encoding needs to be detected.

**Steps**:

* **Opening the File**: open(filename, 'rb') opens the file specified by filename in binary mode ('rb').
* **Reading the File**: f.read() reads the entire content of the file into raw\_data, which is a byte string.
* **Detecting Encoding**: chardet.detect(raw\_data) analyzes raw\_data to determine its encoding. It returns a dictionary (result) containing information about the detected encoding.
* **Returning Encoding**: result['encoding'] extracts and returns the detected encoding as a string.

**3. Usage**

**Purpose**: This part demonstrates how to use the detect\_encoding function to detect and print the encoding of a specific file.

**Steps**:

**Setting filename**: Defines the path to the file whose encoding needs to be detected ('F:/Python/Task/BlackCoffer/MasterDictionary/negative-words.txt' in this case).

**Calling detect\_encoding**: Calls the detect\_encoding function with filename as an argument to determine its encoding.

**Printing Results**: Prints a formatted string showing the detected encoding of the file.

**-----------------------------------------------------------------------------------------------------------------------------------------------------**

****

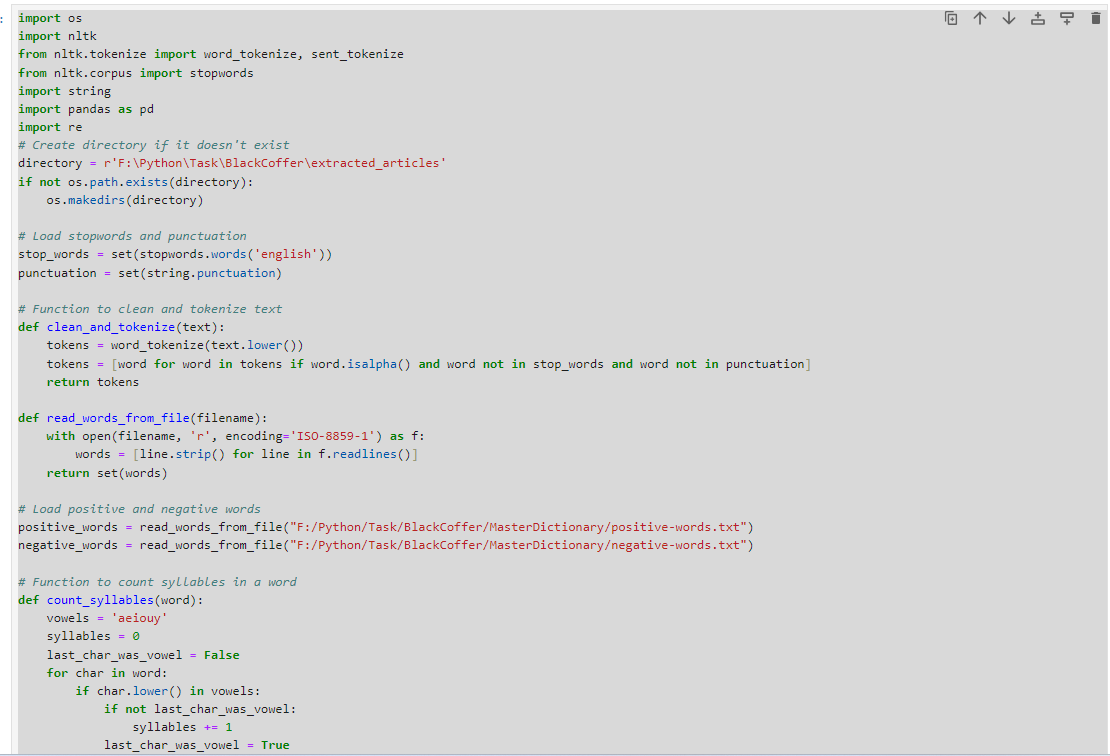
**Loading Positive and Negative Words**

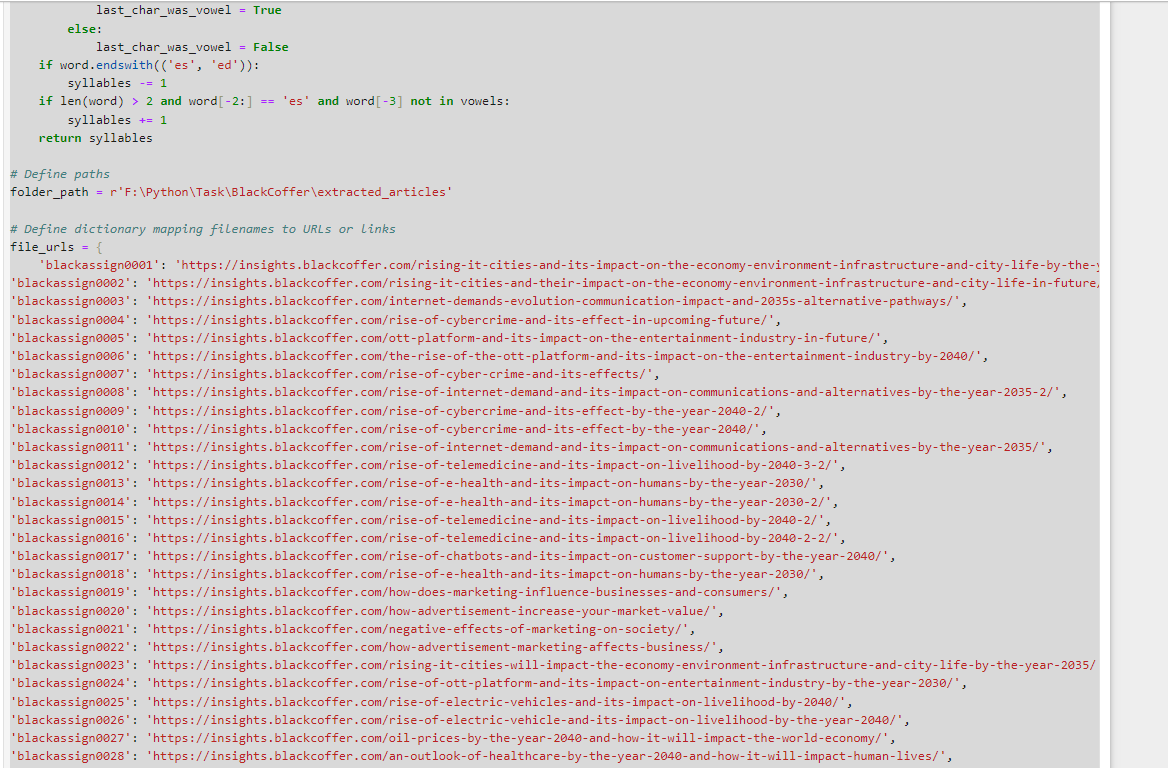
* **Paths:**
* "F:/Python/Task/BlackCoffer/MasterDictionary/positive-words.txt": Path to the text file containing positive words.
* "F:/Python/Task/BlackCoffer/MasterDictionary/negative-words.txt": Path to the text file containing negative words.
* **Usage:**
* Calls the read\_words\_from\_file function twice, once for each file path, to load words into positive\_words and negative\_words variables.
* These variables now hold sets of positive and negative words respectively.

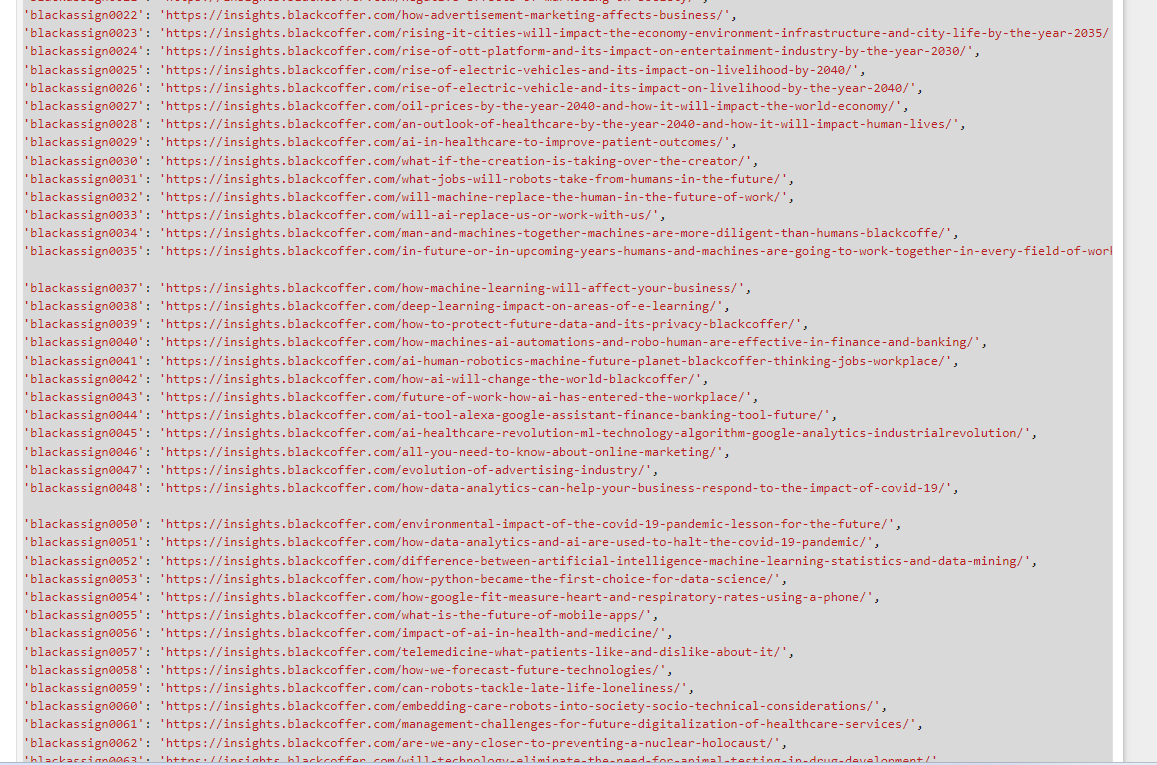
**Printing Positive and Negative Words**

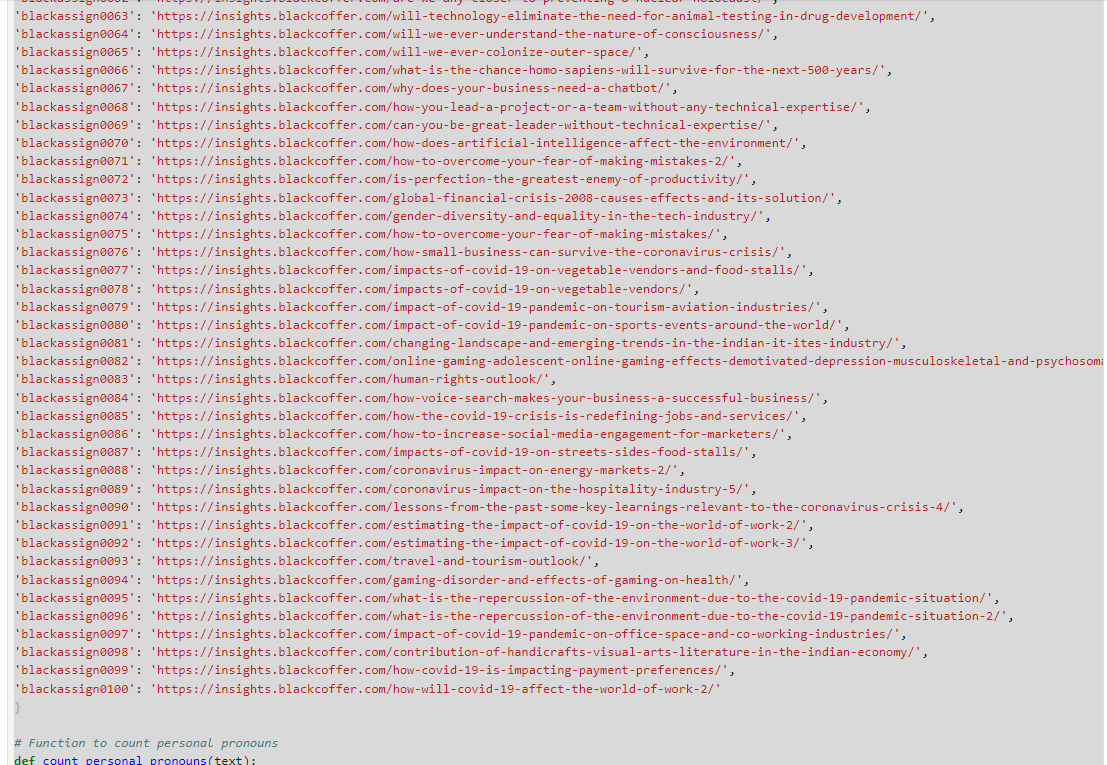
* **Output:**
* Prints the contents of positive\_words and negative\_words sets to the console.

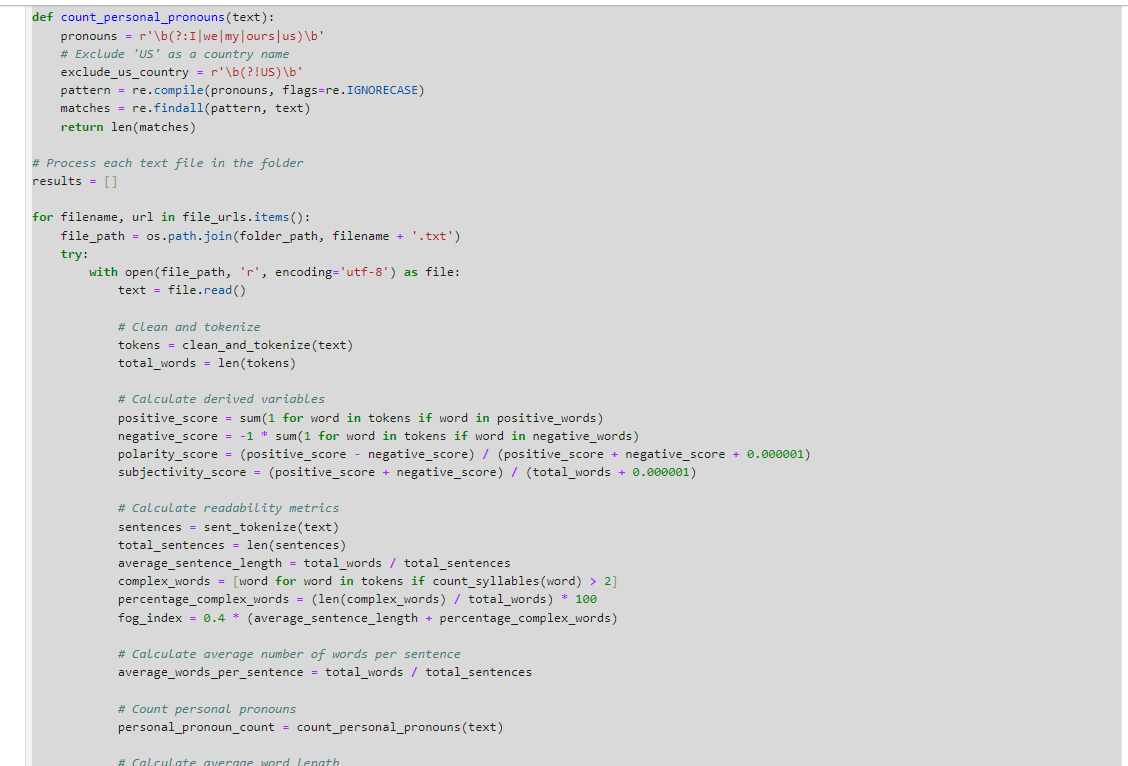
-------------------------------------------------------------------------------------------------------------------------

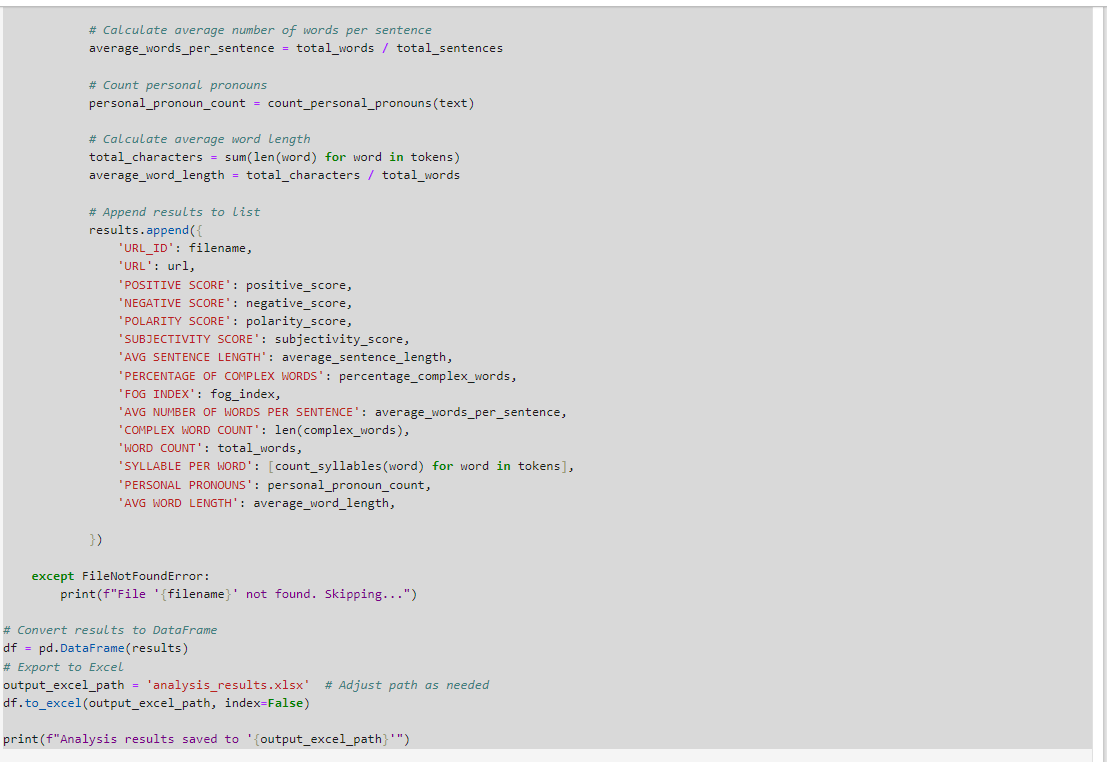


****

****

****

****

****

1. Creating Directory if it Doesn't Exist

* **Purpose: Checks if a directory (extracted\_articles) exists at the specified path. If it doesn't exist, it creates the directory.**
* **Usage:**
* **os.path.exists(directory): Checks if the directory exists.**
* **os.makedirs(directory): Creates the directory if it doesn't exist.**

2. Loading Stopwords and Punctuation

* **Purpose: Loads stopwords and punctuation marks for text processing.**
* **Usage:**
* **stopwords.words('english'): Loads a set of common English stopwords from NLTK (Natural Language Toolkit).**
* **string.punctuation: Provides a set of punctuation marks.**

3. Function to Clean and Tokenize Text

* **Purpose:** Cleans and tokenizes text for further analysis.
* **Steps:**
  + word\_tokenize(text.lower()): Tokenizes the input text into words and converts them to lowercase.
  + List comprehension [word for word in tokens if ...]:
    - word.isalpha(): Filters out non-alphabetic tokens.
    - word not in stop\_words: Filters out stopwords.
    - word not in punctuation: Filters out punctuation marks.
  + Returns a list of cleaned tokens.

**4. Function to Read Words from File**

* **Purpose:** Reads words from a file and returns them as a set of unique words.
* **Steps:**
  + open(filename, 'r', encoding='ISO-8859-1'): Opens the file specified by filename in read mode with the specified encoding.
  + [line.strip() for line in f.readlines()]: Reads all lines from the file (f), strips leading and trailing whitespace characters from each line (line.strip()), and constructs a list (words) containing these cleaned lines.
  + return set(words): Returns a set of unique words read from the file.

**5. Function to Count Syllables in a Word**

 **Purpose:** Counts syllables in a given word.

 **Steps:**

* Iterates through each character (char) in the word.
* Counts syllables based on consecutive vowels, adjusting for common patterns like ending in 'es' or 'ed'.
* Returns the syllable count of the word.

6. File URLs and Dictionary Mapping

 **Purpose:** Stores filenames mapped to their respective URLs or links.

 **Usage:** Provides a mapping for each file processed in the subsequent loop.

7. Function to Count Personal Pronouns

* **Purpose:** Counts occurrences of personal pronouns in text.
* **Steps:**
  + re.compile(pronouns, flags=re.IGNORECASE): Compiles a regular expression pattern (pronouns) to match personal pronouns (case insensitive).
  + re.findall(pattern, text): Finds all matches of the compiled pattern (pattern) in the input text.
  + Returns the count of matches found.

**8. Processing Each Text File**

* **Purpose:** Processes each text file in the specified folder (extracted\_articles), calculates various metrics, and stores the results.
* **Steps:**
  + os.path.join(folder\_path, filename + '.txt'): Constructs the full file path for each file.
  + Opens and reads each file (text = file.read()).
  + Cleans and tokenizes the text using clean\_and\_tokenize.
  + Calculates:
    - positive\_score and negative\_score: Counts of positive and negative words from predefined lists (positive\_words and negative\_words).
    - polarity\_score: Score indicating sentiment polarity based on positive and negative word counts.
    - subjectivity\_score: Score indicating subjectivity based on total words and sentiment words.
    - Readability metrics such as average\_sentence\_length, percentage\_complex\_words, and fog\_index.
    - average\_words\_per\_sentence: Average number of words per sentence.
    - personal\_pronoun\_count: Count of personal pronouns using count\_personal\_pronouns.
    - average\_word\_length: Average length of words in the text.
  + Appends all calculated metrics to results list.

**9. Converting Results to DataFrame and Exporting to Excel**

 **Purpose:** Converts the results list into a Pandas DataFrame and exports it to an Excel file.

 **Steps:**

* pd.DataFrame(results): Creates a DataFrame (df) from the results list.
* df.to\_excel(output\_excel\_path, index=False): Writes the DataFrame to an Excel file specified by output\_excel\_path without including the index.
* Prints a confirmation message with the file path where the results are saved.