**Project Report: Web Article Extraction and Text Analysis Script**

**Web Article Extraction Report**

## 1. Introduction

### 1.1 Project Overview

This project involves developing a Python script to extract and save the content of web articles from a list of URLs provided in an Excel file. The script uses web scraping techniques to fetch and parse webpage content, extracting the main article text and saving it to text files. This process is automated to handle multiple URLs efficiently.

### 1.2 Objectives

* To read URLs and their corresponding IDs from an Excel file.
* To fetch and parse the webpage content using web scraping techniques.
* To extract and clean the main article content.
* To save the extracted content to text files with appropriate filenames.

## 2. Methodology

### 2.1 Libraries and Tools

* **Python**: The programming language used for scripting.
* **Requests: These are for** making HTTP requests to fetch webpage content.
* **BeautifulSoup (bs4)**: This is used to parse HTML and extract the required information.
* **Pandas: This is for** reading data from Excel files.
* **Openpyxl**: As a dependency for reading Excel files in pandas.
* **Requests**: Used for sending HTTP requests to URLs.
* **BeautifulSoup**: Used for parsing HTML documents.
* **Pandas**: Used for reading Excel files.
* **Vs-code-** Download all the required extensions for successful implementation of the script.

#### 3. FUNCTIONS

**3.1 extract\_article(url, url\_id)**: This function takes a URL and a URL ID as inputs.

* **Fetch Webpage Content**: Uses requests.gets to fetch the content of the webpage. Raises an exception if there are HTTP errors.
* **Parse Webpage Content**: Uses BeautifulSoup to parse the HTML content.
* **Extract Title**: Extracts the title of the article.
* **Extract Article Content**: Attempts to find the main article content by looking for common HTML tags (article, div, section). If found, extracts and cleans the text from all <p> tags within the main content.
* **Save Content**: Saves the extracted content to a text file named after the URL ID.

#### 3.2 Main Function

#### main(): This function reads the URLs and IDs from an Excel file and calls extract\_article() for each URL.

#### Read Excel File: Uses pandas.read\_excel() to read the Excel file containing URLs and their IDs.

#### Iterate Through URLs: Iterates through each row in the DataFrame, extracting the URL and URL ID, and calls the extract\_article() function.

### 3.2 Error Handling

* **HTTP Errors**: Uses response.raise\_for\_status() to raise exceptions for HTTP errors.
* **Missing Article Content**: Checks if the main article content is found; if not, prints an error message.
* **General Request Exceptions**: Catches general request exceptions and prints an error message.

## 4. Implementation

### 4.1 Dependencies

Ensure you have the following libraries installed:

* requests
* beautifulsoup4
* pandas
* openpyxl

### 4.2 Running the Script

1. **Prepare the Excel File**: Ensure you have an Excel file named urls.xlsx with columns URL and URL\_ID.
2. **Run the Script**: Execute the script using a Python interpreter example - Python Idle, Vs-code, etc.

## 5. Results

### 5.1 Extracted Articles

The script successfully extracts and saves articles from the provided URLs. Each article is saved in a text file named after its URL ID.

### 5.2 Error Handling

The script handles common errors such as HTTP errors and missing article content, providing meaningful messages for troubleshooting.

## 6. Conclusion

### 6.1 Summary

This project demonstrates the use of web scraping to automate the extraction of web articles. The script efficiently handles multiple URLs, extracts relevant content, and saves it for further analysis or use.

### 6.2 Future Improvements

* **Dynamic Tag Identification**: Improve the script to dynamically identify the main content tags based on more complex rules or machine learning models.
* **Enhanced Error Handling**: Add more detailed logging and error handling to cover a wider range of issues.
* **Scalability**: Optimize the script for scalability to handle a larger number of URLs and improve performance.

### 6.3 Business Implications

Automating web article extraction can save significant time and effort, enabling businesses to gather and analyze large volumes of web content efficiently. This can be particularly useful for market research, sentiment analysis, and competitive intelligence

**Text Analysis Report**.

### 1. Project Overview

### This project focuses on developing a Python script that processes text files to compute various readability and sentiment analysis metrics. The script is particularly useful for assessing the content quality and tone of web articles if their text data is available in a local directory.

### 2. Objective

### The main objective of this project is to provide a comprehensive analysis of web articles through various linguistic metrics, including polarity score, subjectivity score, average word length, sentence length, complex word percentage, and more. These metrics aid in understanding the readability and sentiment conveyed in the text.

### 3. Tools and Libraries Used

### Python: The primary programming language used for the script.

### Pandas: A Python library used for data manipulation and analysis.

### NLTK (Natural Language Toolkit): A powerful Python library used for working with human language data (text), providing capabilities for tokenizing, parsing, classification, stemming, tagging, and semantic reasoning.

### os: A Python module that provides a portable way of using operating system dependent functionality.

### 4. Implementation Details

### The script can be broken down into several key functional components:

#### 4.1 NLTK Resource Setup

### The script initializes by downloading necessary NLTK resources such as 'punkt' for sentence tokenization and 'stopwords' for filtering common words in English that may not contribute significantly to sentiment analysis.

#### 4.2 File Operations

### The script writes lists of positive and negative words to corresponding text files (positive-words.txt and negative-words.txt). These lists are used to score the sentiment of the processed text files.

#### 4.3 Text Analysis Functions

### Several functions are implemented to compute different metrics from the text:

### Tokenization and Filtering: Identifies words and sentences while removing stop words and punctuation.

### Sentiment Scores: Calculates positive and negative scores based on occurrences of words from predefined lists.

### Readability Scores: Computes various readability indicators such as the Fog Index, which is indicative of the text's complexity and the education level required to understand the text.

#### 4.4 Processing Text Files

### The script processes each text file within a specified directory, applying the text analysis functions to extract metrics and storing the results in a structured format (Pandas DataFrame).

#### 4.5 Output

### The final output is an Excel spreadsheet that logs the computed metrics for each text file, allowing for easy comparison and further analysis.

### 5. Applications

### Content Analysis: Useful for publishers and content creators to gauge the quality and tone of their writings.

### Educational Tools: Helps in creating material that matches the comprehension skills of the target audience.

### SEO and Marketing: Analyzing text to ensure it is engaging and accessible to a broad audience.

### 6. Challenges and Limitations

### Scalability: Processing very large files or a large number of files might require optimization or more powerful computing resources.

### Language Limitations: Currently, the script only supports English. Adapting it to other languages would require additional stopwords lists and possibly different sentiment analysis logic.

### Context Awareness: The script may not fully understand the context or the subtleties of language such as irony or sarcasm, which could affect sentiment analysis accuracy.

### 7. Future Enhancements

### Integration with Web Scraping: Automate the extraction of text from web articles online rather than relying on locally stored files.

### Multi-language Support: Enhance the script to process texts in multiple languages.

### Improved Sentiment Analysis: Implement more advanced techniques like machine learning models for better context understanding and sentiment accuracy.

### 8. Conclusion

### This project demonstrates the potential of automated text analysis tools in understanding and improving web content. Through detailed linguistic and readability metrics, content creators can refine their articles to better meet the needs and preferences of their audience.

### This report provides a structured overview of the project from its objectives and implementation to its applications, challenges, and potential future developments.

### 

#### 