# Instruction Documentation for Data Scientist Assignment

## Objective

The objective of this assignment is to extract textual data (only the main article from the web page) from articles available at given URLs and perform text analysis to compute specific linguistic and sentiment-based variables. The results are written into a structured CSV file for further analysis. Below, I explain my approach, the techniques used, and the step-by-step implementation process.

## Approach and Solution

Step 1: Understanding the Variables to Compute

The task requires computing the following variable:

POSITIVE SCORE,NEGATIVE SCORE,POLARITY SCORE,SUBJECTIVITY SCORE,AVG SENTENCE LENGTH,PERCENTAGE OF COMPLEX WORDS,FOG INDEX,AVG NUMBER OF WORDS PER SENTENCE,COMPLEX WORD COUNT,WORD COUNT,SYLLABLE PER WORD,PERSONAL PRONOUNS,AVG WORD LENGTH

Step 2: Approach to Extract and Analyze Text

The workflow was broken down into several logical steps:

1. **Input and Initialization**:
    - Read the input `Input.csv` containing URLs and associated IDs using pandas.

    Initialize the output CSV file `Output Data Structure.csv` with the specified headers(given variables which we studied above).

2. **Web Scraping**:

- For web scraping, I used the `newspaper3k` library for extracting textual content from the given URLs. This library efficiently downloads and parses web articles, which includes only their titles and main content which we need in this task.
- If `newspaper3k` failed due to timeouts, the `requests-html` and `readability` libraries served as fallback options.

  (*There are other more reliable and advance libraries like BeautifulSoup, Selenium, etc. but they fall behind in extracting cleaner text data and also requires more advance knowledge of html and the library itself, but in this case we only need the article data for that newspaper3k or* `requests-html` *and* `readability` *are very useful as they provide the required data much easier in only couple of lines of code*)

3. **Text Preprocessing**:

- **Text Cleaning**: Removed HTML tags, URLs, punctuation, emojis, and stop words to focus on meaningful content using regex expressions and NLTK.
- **Tokenization**: Used NLTK's `sent_tokenize` and `word_tokenize` functions to split the text into sentences and words.

4. **Linguistic Feature Computation**:

- **Sentiment Scores**:Matched words against a predefined dictionary of positive and negative words. Calculated positive and negative scores by counting occurrences of respective words.
- **Complex Words**: Defined complex words as those with more than two syllables. Applied rules to handle edge cases like silent "e" and suffixes like "es" and "ed."
- **Readability and Structural Metrics**: Calculated metrics such as fog index, average sentence length, and percentage of complex words using formula-based approaches.
- **Pronoun Counting**: Used regex to identify and count first-person pronouns like *I, we, my, ours, us* while avoiding false positives (e.g., "US" as a country name).

5**. Writing Output**: Iteratively appended computed variables to the output CSV file.

## How to Run the Script

**Dependencies**: Ensure the following Python libraries are installed:

- `requests-html`
- `readability-lxml`
- `newspaper3k`
- `nltk`
- `pandas`
- `beautifulsoup4`
- `lxml`

Install any missing dependencies using the `install_package` function provided in the script or with `pip`

### Input Files:

- Place the `Input.csv` file in the working directory. The file should contain at least two columns: `URL_ID` and `URL`.
- Include directories for:
  - **Stop Words**: A folder named `StopWords` containing files with stop words.
  - **Master Dictionary**: A folder named `MasterDictionary` containing `positive-words.txt` and `negative-words.txt`.

**Run the Script**: Execute the script from the command line below-

```
python web_scrap.py
```

### Output:

- The script generates two outputs:
  - Individual `.txt` files containing cleaned article content, named by `URL_ID`.
  - A CSV file named `Output Data Structure.csv` containing the computed variables for each URL.

## Key Implementation Highlights

- **Error Handling**:
  - Incorporated fallback mechanisms for downloading articles and robust handling for timeouts or unavailable URLs.
- **Modular Design**:
  - Functions for tasks such as removing HTML tags, stop words, emojis, and punctuation ensure reusability and clarity.
- **Optimized Processing**:
  - Used dictionary lookups for positive and negative word matching to improve performance on large texts.

## Challenges and Solutions

1. **Challenge**: Some URLs timed out during scraping.
   - **Solution**: Increased timeout settings and used fallback libraries (`readability` and `requests-html`).
2. **Challenge**: Handling different types of punctuation and stop words across multiple files.
   - **Solution**: Consolidated all stop words into a single list dynamically during runtime.
3. **Challenge**: Calculating complex metrics like fog index and syllables accurately.
   - **Solution**: Wrote custom functions to handle specific linguistic rules.