1. **Text Retrieve From the URL**
2. **Import important libraries**
3. Import numpy as np – ‘numpy’ is a fundamental library for numerical and scientific computing in python, providing support for arrays and mathematical operations on those arrays.
4. Import pandas as pd – ‘pandas’ is a powerful data manipulation and analysis library that provides data structures and functions for working with structured data, such as tabular data and time series data.
5. Import requests – for making HTTP requests to web servers and interacting with web services.
6. From bs4 import BeautifulSoup – BeautifulSoup is a python library for parsing and manipulating HTML and XML documents(parsing HTLM/XML, Searching and Navigating, modifying documents, Data Extraction, Integration web Scraping).
7. Import os – it is a built-in module that provides a wide range of functions and methods for interacting with the operating system.
8. **Load the excel file**
9. Perform indexing in URL\_ID to retrieve accurate URL\_ID.
10. For Loop: The script appears to be designed to iterate over rows of a DataFrame named df, assuming it contains columns 'URL\_ID' and 'URL'. However, the code doesn't define or load this DataFrame, so you would need to define it elsewhere in your script.
11. HTTP Request: Inside the loop, it sends an HTTP GET request to the URL specified in each row of the DataFrame and checks if the response status code is 200 (indicating a successful request).
12. HTML Parsing: If the request is successful, it uses BeautifulSoup to parse the HTML content of the web page.
13. Data Extraction: The code attempts to extract the title of the web page from an <h1> element and content from either of two <div> elements with specific classes. If the first <div> element is found (div\_element\_1), it extracts text from <p> and <ul> elements within it; otherwise, it uses the second <div> element (div\_element\_2) for extraction.
14. Text Preparation: The extracted title and paragraphs are processed, stripping extra whitespace and converting them into strings. The paragraphs are concatenated into a single string.
15. File Saving: The concatenated title and text content are saved to a text file named after the 'URL\_ID'. The text files are saved in a directory called "txt\_file".
16. Error Handling: If there's an error during the HTTP request (handled using requests.exceptions.RequestException), an error message is appended to the extracted\_data list. However, the code doesn't show the initialization of this list.
17. Printing: The script prints messages indicating where the content was saved.

A few points to note:

- The script assumes the existence of a DataFrame named df with 'URL\_ID' and 'URL' columns, which is not defined in the provided snippet.

- The script doesn't handle certain potential issues, such as non-existent HTML elements or unexpected website structures, which might lead to errors.

- Depending on the structure of the web pages you are scraping, you may need to adjust the code to better target the elements you want to extract.

- Ensure that you have defined the df DataFrame and have taken care of any additional dependencies and error handling as needed for your specific use case before running this script.

1. **My function – I create a user define function for calculate the following :**

**A . Imports:** The script begins by importing various Python libraries, including:

1. nltk: The Natural Language Toolkit, used for natural language processing tasks.
2. string: Provides a set of punctuation characters.
3. re: Provides support for regular expressions.
4. os: Provides functions for interacting with the operating system.
5. tqdm: A library for displaying progress bars.
6. wordninja: A library for splitting concatenated words into meaningful words.
7. pandas as pd: A data manipulation library.
8. **Read all positive and negative words from the folder** - To read all positive and negative words from text files in a folder in Python, you can use the following code as a general template. This code assumes you have text files named "positive-words.txt" and "negative-words.txt" in a specified directory.To read all positive and negative words from text files in a folder in Python, you can use the following code as a general template. This code assumes you have text files named "positive-words.txt" and "negative-words.txt" in a specified directory.
9. **Create a function to clean and normalize a word** - to clean and normalize a word by converting it to lowercase and removing symbols or non-alphanumeric characters.
10. **Define a new function to clean and extract words** - to clean and extract words from a given text. In this function, you can specify the cleaning and extraction rules you want to apply to the text.
11. **Define a function to count\_syllables** - create a basic function to estimate the number of syllables in a word using a rule-based approach.
12. **Define a function to count\_personal\_pronouns** - This function takes a text as input and counts the occurrences of first person, second person, and third person pronouns in the text. It returns a dictionary containing the counts for each category.
13. **Define a function to calculate\_average\_word\_length** - This function takes a text as input, splits it into words, calculates the total word length, and then computes the average word length by dividing the total word length by the number of words in the text. It returns the average word length as a floating-point number.
14. **Import nltk**
15. **Custom function to count syllables in a word with exceptions** - A basic approach is to count the number of vowels in a word, but this approach may not be accurate for all words, especially when considering exceptions.
16. **Custom function to txt\_analysis** - This function takes a text as input and returns a dictionary containing the counts of words, sentences, and paragraphs in the text. You can further extend this function to include more advanced text analysis features as needed for your specific use case.
17. **Text Analysis**
18. **Import libraries:**
19. **Define the paths**
20. **excel\_file\_path = "Output Data Structure.xlsx" -** This line creates a variable named excel\_file\_path and assigns it a string value, which is the file path to an Excel file named "Output Data Structure.xlsx." This variable is likely intended to store the path to an Excel file for further processing or manipulation in the script.
21. **my\_file\_folder = "txt\_file" -** This line creates another variable named my\_file\_folder and assigns it a string value, which is the name of a folder or directory called "txt\_file." This variable is likely intended to store the name of a folder where text files or other files will be located or generated in the script.
22. **Read the DataFrame**
23. **Iterate Through Files in a Folder -** It starts by iterating through the files in the folder specified by the my\_file\_folder variable using os.listdir(my\_file\_folder). This assumes that os module has been imported previously in the script.
24. **Filter for Text Files -** For each file in the folder, it checks if the file has a ".txt" extension using if file\_name.endswith(".txt"). This is a way to filter and process only text files within the folder.
25. **Read and Process Text Files:**
26. It constructs the full path to the text file using os.path.join(my\_file\_folder, file\_name).
27. Initializes an empty string variable txt.
28. Attempts to open and read the contents of the text file using a try block. If the file is not found, it prints an error message and continues to the next file.
29. **Text Processing:**
30. After successfully reading the text file, it performs some text processing on the content:
31. It adds a space after periods to split sentences using the txt = txt.replace('.', '. ') line.
32. It then analyzes the text using a function txt\_analysis(txt) and stores the results in a variable t.
33. **Extracting Filename -** It extracts the filename without the extension (e.g., "321.txt" -> "321") using os.path.splitext(file\_name).
34. **Updating a DataFrame –**
35. It tries to convert the extracted filename to a float and assigns it to the variable url\_id.
36. It searches for the corresponding row in a DataFrame df where the 'URL\_ID' column matches the url\_id.
37. If a matching row is found, it updates various columns in that row with the values from the analysis (t) of the text file.
38. If no matching row is found, it prints an error message.
39. **Handling Invalid Data -** If the filename cannot be converted to a float (e.g., if the filename is not a valid numeric ID), it prints an error message.
40. **Saving the updated DataFrame to an Excel file**
41. **Save DataFrame to Excel –**

df.to\_excel(excel\_file\_path, index=False) is used to save the DataFrame df to an Excel file specified by the excel\_file\_path variable.

The index=Falseargument ensures that the index (row numbers) of the DataFrame is not included in the Excel file. This is often done to create a clean representation of the data without the default index column.

1. **Print a Confirmation Message -** print("Positive and Negative Scores updated in the Excel file, 'URL\_ID' and 'URL\_LINK' columns preserved.") is a simple print statement that provides a message to indicate that the operation was completed successfully. It mentions that the 'Positive' and 'Negative Scores' have been updated in the Excel file, and it specifies that the 'URL\_ID' and 'URL\_LINK' columns have been preserved. This message is just for informative purposes and helps users understand what the code has done.

In summary, this code saves the updated DataFrame to an Excel file specified by excel\_file\_path and then prints a confirmation message to inform the user that the operation was successful and that specific columns in the DataFrame have been preserved in the Excel file. The saved Excel file will contain the data from the DataFrame without the index column.