The exact Instruction to create the Project.py file by using python programming.

1. Import the necessary libraries:
    - pandas: to work with data frames
    - requests: to send HTTP GET requests
    - BeautifulSoup: to parse HTML content
    - nltk: Natural Language Toolkit library for text processing
2. Read the input data from an Excel file located at "C:\Users\OMEN\Desktop\Black Coffer\Project 2\Input.xlsx" into a pandas DataFrame called df.
3. Define an empty list text_list to store the extracted text from the web pages.
4. Define an empty list results to store the calculated metrics for each URL.
5. Define a function clean_text(text) that takes a text string as input and performs text cleaning by removing stopwords and converting the text to lowercase. The function returns a list of cleaned words.
6. Define a function create_word_dictionary() that reads positive and negative word lists from separate text files located at "C:/Users/OMEN/Desktop/Black Coffer/MasterDictionary/positive-words.txt" and "C:/Users/OMEN/Desktop/Black Coffer/MasterDictionary/negative-words.txt". The function returns two sets: positive_words and negative_words.
7. Define a function calculate_sentiment_scores(text) that calculates the sentiment scores for a given text using the positive and negative word dictionaries obtained from create_word_dictionary(). The function returns the positive score, negative score, polarity score, and subjectivity score.
8. Define a function calculate_average_sentence_length(text) that calculates the average number of words per sentence in a given text.
9. Define a function calculate_complex_word_percentage(text) that calculates the percentage of complex words (words with more than 2 letters and consisting only of alphabetic characters) in a given text.
10. Define a function calculate_fog_index(text) that calculates the Fog Index for a given text based on the average sentence length and percentage of complex words.
11. Define a function calculate_average_words_per_sentence(text) that calculates the average number of words per sentence in a given text.
12. Define a function count_complex_words(text) that counts the number of complex words in a given text.
13. Define a function count_words(text) that counts the total number of words in a given text.
14. Define a function count_syllables(word) that counts the number of syllables in a given word.
15. Define a function calculate_average_syllables_per_word(text) that calculates the average number of syllables per word in a given text.

16. Define a function count_personal_pronouns(text) that counts the occurrences of personal pronouns (e.g., "I", "we", "my", "ours", "us") in a given text.
17. Define a function calculate_average_word_length(text) that calculates the average length of words in a given text.
18. Iterate over each row in the DataFrame df using the iterrows() method.
19. For each row, extract the URL_ID and URL values.
20. Send a GET request to the URL using the requests.get() function and store the response.
21. Create a BeautifulSoup object soup to parse the HTML content of the response.
22. Find the elements containing the article title and text using the appropriate CSS selectors.
23. If a title element is found, extract the text from the first matching element and assign it to the variable title. Otherwise, print an error message and continue to the next row.
24. If text elements are found, clear the text_list and iterate over the found elements. For each element, extract the text, clean it using clean_text(), and append it to the text_list. Finally, join all the cleaned texts with newline characters to create the full_text.
25. If no text elements are found, print an error message and continue to the next row.
26. Calculate the sentiment scores for the full_text using calculate_sentiment_scores(). Assign the returned values to variables: positive_score, negative_score, polarity_score, and subjectivity_score.
27. Print the calculated sentiment scores.
28. Calculate the average number of words per sentence using calculate_average_words_per_sentence() and assign it to the variable average_sentence_length.
29. Print the average number of words per sentence.
30. Calculate the percentage of complex words using calculate_complex_word_percentage() and assign it to the variable percentage_of_complex_words.
31. Print the percentage of complex words.
32. Calculate the Fog Index using calculate_fog_index() and assign it to the variable fog_index.
33. Print the Fog Index.
34. Count the number of complex words using count_complex_words() and assign it to the variable complex_word_count.
35. Print the complex word count.
36. Count the total number of words using count_words() and assign it to the variable word_count.
37. Print the word count.

38. Calculate the average number of syllables per word using calculate_average_syllables_per_word() and assign it to the variable average_syllables_per_word.
39. Print the average number of syllables per word.
40. Count the occurrences of personal pronouns using count_personal_pronouns() and assign the counts to the dictionary variable personal_pronoun_counts.
41. Print the personal pronoun counts.
42. Calculate the average word length using calculate_average_word_length() and assign it to the variable average_word_length.
43. Print the average word length.
44. Write the extracted article text (title + full_text) to a text file named "{url_id}.txt" using the open() function with write mode.
45. Create a dictionary result containing all the calculated metrics and their corresponding values for the current URL.
46. Append the result dictionary to the results list.
47. After processing all rows in the DataFrame, create a new DataFrame output_df from the results list.
48. Save the output_df to an Excel file located at "C:\Users\OMEN\Desktop\Black Coffer\Project 2\Output.xlsx" using the to_excel() method with index=False.