Firstly, imported the required libraries:

* import pandas as pd
* import requests
* from bs4 import BeautifulSoup
* import numpy as np
* from io import BytesIO
* import json
* import numpy as np
* from requests.models import MissingSchema
* import syllables
* import re
* from textstat import syllable\_count
* from nltk.sentiment.vader import SentimentIntensityAnalyzer
* from nltk import word\_tokenize
* from collections import Counter
* from nltk.tokenize import RegexpTokenizer
* import nltk

Web scraping is done, which extracts text from the given URLs and saves it as text files.

The code iterates through each row of a pandas DataFrame, which contains two columns: 'URL' and 'URL\_ID'. For each row, it extracts the URL and URL ID. Then, it sends a GET request to the URL and receives the response, which is the HTML content of the webpage. The BeautifulSoup library is used to parse the HTML content and extract the textual content of the article. The code searches for all 'p' tags, skips over any 'p' tags that contain an image, and adds the text of the remaining 'p' tags to the 'article\_text' string.

After the article text has been extracted, it is saved to a text file. The file name is based on the URL ID, and the file is written using UTF-8 encoding.

Finally, the code prints a message indicating that the article has been extracted and saved to a file.

The above code defines a function load\_stop\_words(file\_path) that reads a stop words file, and returns a set containing the stop words.

Then, it loads several stop words files, such as stop\_words\_Auditor.txt, stop\_words\_Geographic.txt, stop\_words\_Currencies.txt, etc., using the load\_stop\_words function, and assigns the returned set to corresponding variables stop\_words\_Auditor, stop\_words\_Geographic, stop\_words\_Currencies, etc.

The stop\_words\_generic.txt and stop\_words\_GenericLong.txt files seem to contain generic stop words, and stop\_words\_Names.txt contains stop words that are names.

Defined a function normalize\_text for text normalization.

The function takes a single argument text, which is the input text to be normalized. It performs the following steps:

1. Removes all non-word characters and converts the text to lowercase.
2. Replaces periods with the string "FULL\_STOP\_TOKEN".
3. Tokenizes the text into individual words using the word\_tokenize function from the nltk.tokenize package.
4. Removes stop words from the text using several sets of stop words, each of which is defined in a separate file and loaded into memory using the load\_stop\_words function.
5. Lemmatizes the words using the WordNet lemmatizer from the nltk.stem package.
6. Joins the words back into a single string, and returns the normalized text.

Note that the code requires the nltk package and its associated data files to be downloaded, and several stop word files to be present in the working directory.

Defined the code that reads each text file saved with a URL\_ID as the filename, normalizes the text using the normalize\_text function, and overwrites the original file with the normalized text.

First, it extracts the url\_id and url from the current row of the dataframe df. It then opens the text file with the name of url\_id.txt using the with open statement, reads the text into the variable text, and closes the file.

Next, it calls the normalize\_text function to clean and normalize the text, storing the result in the normalized\_text variable. Finally, it overwrites the original text file by opening it in write mode using with open statement, writing the normalized\_text to the file, and then closing the file.

The code is defined get\_positive\_score to loads a set of positive words from a file called 'positive-words.txt', and defines a function called 'get\_positive\_score' which takes in a string of text, and returns the proportion of positive words in the text.

The 'positive-words.txt' file is expected to contain one positive word per line, and the words are read into a set, which is used for fast membership testing when counting the positive words in the text.

The 'get\_positive\_score' function first splits the text into individual words, and then counts the number of positive words present in the text by using the 'sum' function with a generator expression that checks if each word is in the 'positive\_words' set.

Finally, the function returns the proportion of positive words in the text by dividing the count of positive words by the total number of words in the text. If the total number of words is 0, the function returns 0 to avoid a divide-by-zero error.

The code is defined as function get\_negative\_score that takes a string of text as input and returns the negative sentiment score of the text. The function first loads a set of negative words from a file called "negative-words.txt" and then splits the input text into words. It then counts the number of words in the input text that are in the set of negative words, and divides that count by the total number of words in the text to get the negative sentiment score. If the input text has no words, the function returns 0.

This code is defined as a function count\_syllables that takes a word as input and returns the number of syllables in the word. It does this by counting the number of vowel sounds in the word, using the following rules:

* A vowel sound is any of the letters a, e, i, o, u, or y.
* A silent "e" at the end of a word is not counted as a vowel sound.
* A vowel sound that comes right after another vowel sound is not counted twice.

The function loops through the letters of the word and checks if each letter is a vowel. If it is, it increments the num\_vowels counter. If the previous letter was also a vowel, the counter is decremented, as the two vowels should only count as one syllable. Finally, if the word ends in a silent "e", the counter is decremented again. If the final count is 0, it is set to 1, as every word should have at least one syllable.

The count\_complex\_words(text) function counts the number of complex words in a given text. A word is considered complex if it has more than two syllables. The function first splits the text into individual words using the split() method, and then iterates through each word to count the number of syllables using an external module called syllables. The syllables.estimate(word) method returns the estimated number of syllables in the given word.

If the estimated number of syllables is greater than 2, the function increments a counter variable count by 1. The function returns the final count of complex words in the text.

The syllables module uses a combination of heuristics and rules to estimate the number of syllables in a given word, which is not always accurate. Additionally, the definition of a "complex word" may vary based on context, domain, and other factors. Therefore, this function should be used with caution and may require customization based on specific use cases.

The count\_personal\_pronouns() function counts the number of personal pronouns in the given text. It uses a regular expression pattern to find matches for common personal pronouns such as "I", "we", "my", "ours", and "us". The function then removes any instances of "US" from the list of matches, and returns the total number of matches.

The function count\_syllables counts the number of syllables in a given word. It takes a single argument word, which is a string representing the word to count syllables for. The function first defines a list of vowels, and then removes any trailing "es" or "ed" from the word. It then iterates through each character in the word, and if the character is a vowel and the previous character is not a vowel, it increments a syllable counter. The function also handles special cases where the word ends in "le" or has no vowels. Finally, the function returns the total number of syllables counted.

iterating over each row of a DataFrame, and for each row, you are reading the text from a file based on the URL ID in that row. Then you are computing various text features for that text, such as positive and negative scores, polarity, subjectivity, and so on.

It seems like you are storing these feature values in different lists, such as positive, negative, polarity, subjectivity, fogIndex, avgnum, complexwords, syllable, wordcount, and avgwordlength.

This is a function named "count" that takes a string argument "word". Inside the function, the input text is first split into words by using the split() method, and the resulting list is stored in the variable words. Then, a variable count is initialized to 0. Next, a loop is iterated over the words in the words list. For each word in the list, the function checks if it is not equal to the string 'FULL\_STOP\_TOKEN'. If it is not, the length of the word is added to the count variable. Finally, the function returns the total count of characters in all the words that are not equal to 'FULL\_STOP\_TOKEN'.

After iterating over each row of a DataFrame, and for each row, reading the text from a file based on the URL ID in that row is coded. Then i computed various text features for that text, such as positive and negative scores, polarity, subjectivity, and so on.

feature values required in different lists, such as positive, negative, polarity, subjectivity, fogIndex, avgnum, complexwords, syllable, wordcount, and avgwordlength are added to the data frame and then it is converted into 'Output Data Structure.xlsx'.