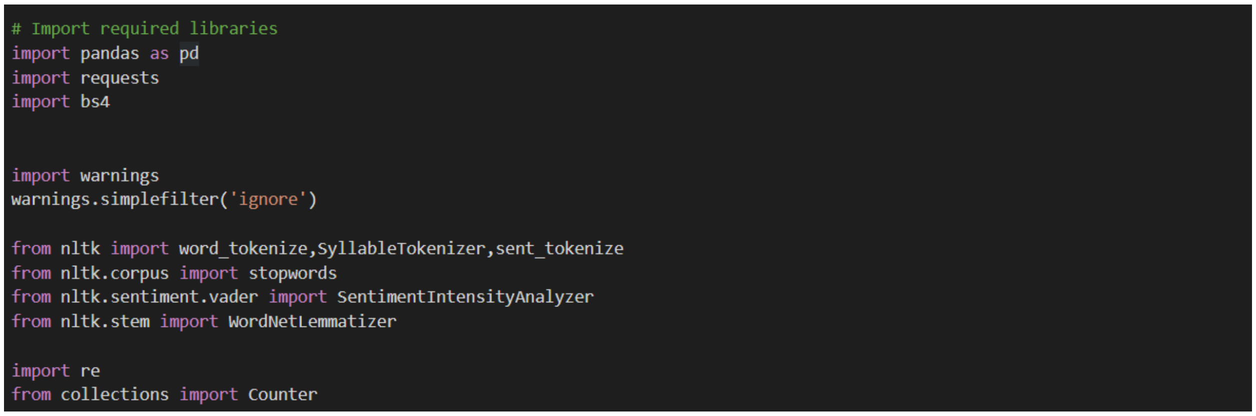
DATA EXTRACTION AND TEXT ANALYSIS

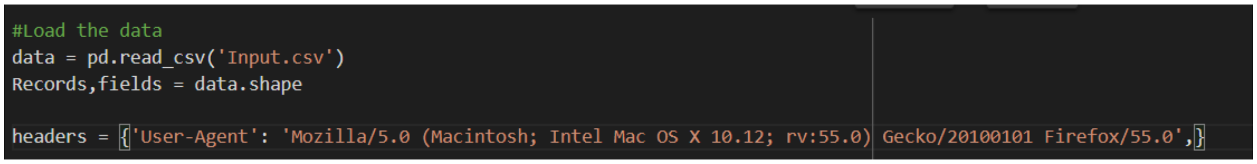
DATA EXTRACTION:

Let’s first import the Libraries



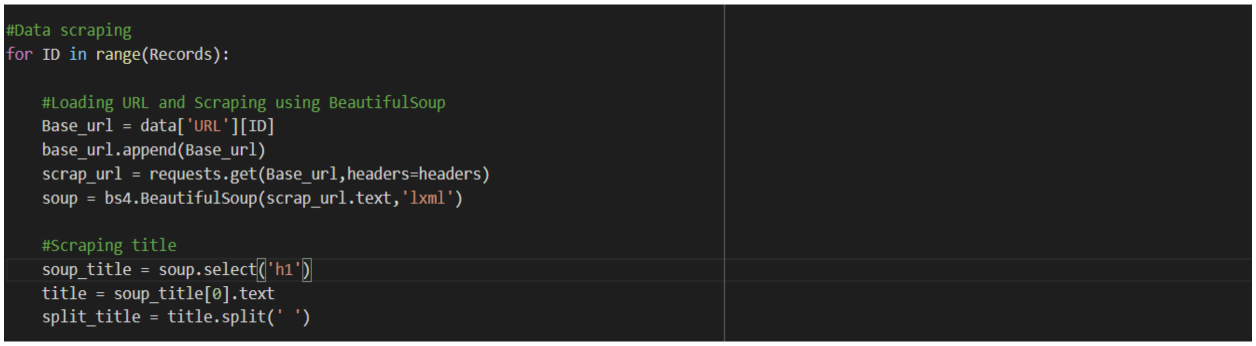
```
# Import required libraries
import pandas as pd
import requests
import bs4


import warnings
warnings.simplefilter('ignore')

from nltk import word_tokenize,SyllableTokenizer,sent_tokenize
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem import WordNetLemmatizer

import re
from collections import Counter
```

Now, we will import the dataset and get the number of records and fields from the dataset by using the shape.

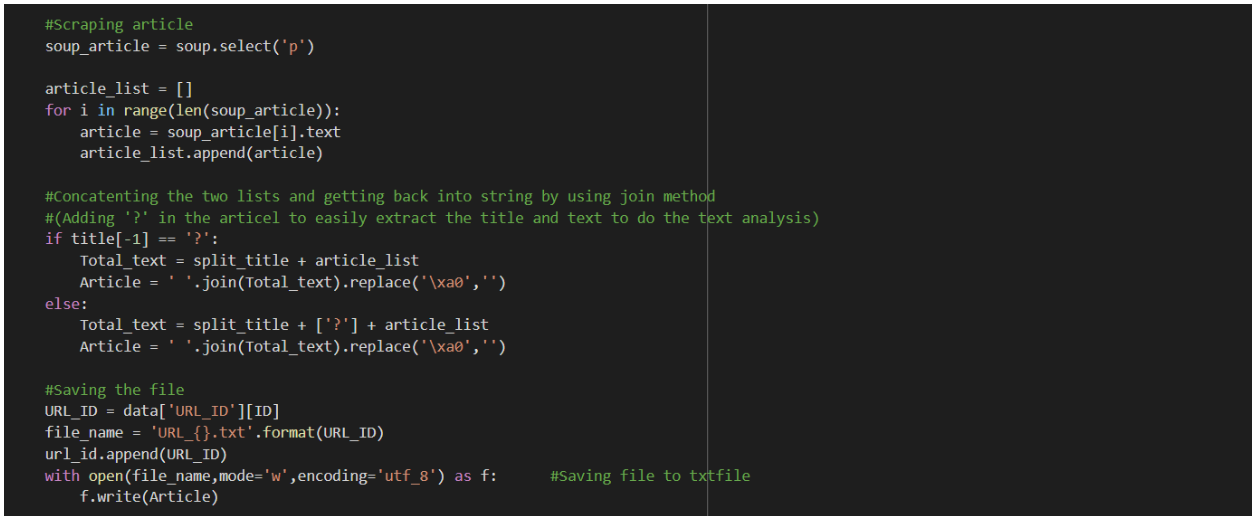


```
#Load the data
data = pd.read_csv('Input.csv')
Records,fields = data.shape

headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:55.0) Gecko/20100101 Firefox/55.0',}
```

Next, Get every URL and URL_ID from the dataset by using the indexes of the records in a looping manner (For Loop). First we get a request on the URL and then use the beautiful soup to extract the appropriate text (Title and Text). For convenience, We are concatenating the Title and Text by ‘?’ to get the back only Text to do the Text Analysis. If the title already has the ‘?’ in it we are not adding ‘?’ to it else adding it. Joining the two lists to get back the string format. Finally, Saving the Total text into URL_ID.txt file.



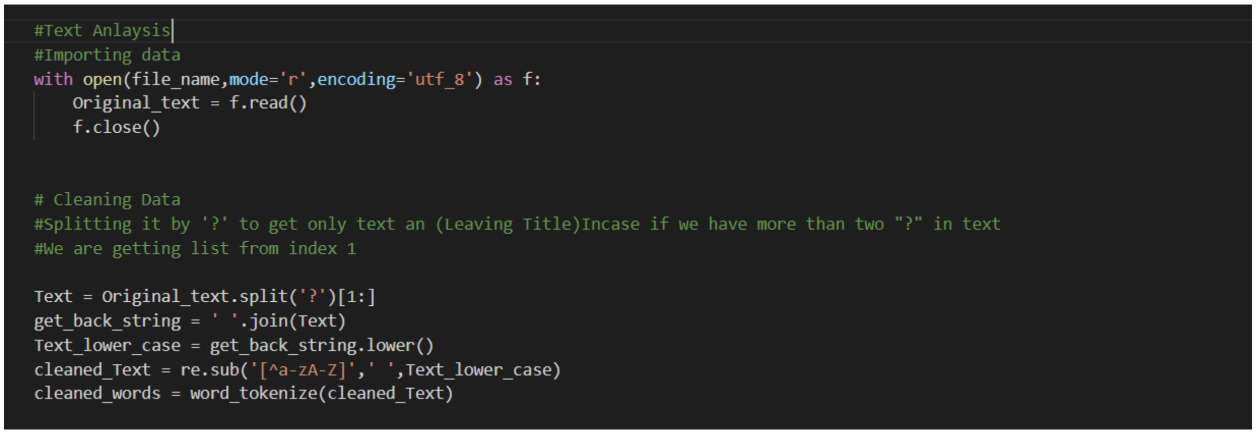
```
#Data scraping
for ID in range(Records):

    #Loading URL and Scraping using BeautifulSoup
    Base_url = data['URL'][ID]
    base_url.append(Base_url)
    scrap_url = requests.get(Base_url,headers=headers)
    soup = bs4.BeautifulSoup(scrap_url.text,'lxml')

    #Scraping title
    soup_title = soup.select('h1')
    title = soup_title[0].text
    split_title = title.split(' ')
```



```
#Scraping article
soup_article = soup.select('p')

article_list = []
for i in range(len(soup_article)):
    article = soup_article[i].text
    article_list.append(article)

#Concatenting the two lists and getting back into string by using join method
#(Adding '?' in the articel to easily extract the title and text to do the text analysis)
if title[-1] == '?':
    Total_text = split_title + article_list
    Article = ' '.join(Total_text).replace('\xa0','')
else:
    Total_text = split_title + ['?'] + article_list
    Article = ' '.join(Total_text).replace('\xa0','')

#Saving the file
URL_ID = data['URL_ID'][ID]
file_name = 'URL_{}.txt'.format(URL_ID)
url_id.append(URL_ID)
with open(file_name,mode='w',encoding='utf_8') as f:      #Saving file to txtfile
    f.write(Article)
```

TEXT ANALYSIS:

First, Read the text file and then split the string by "?" to separate the title and get the string from index 1. In addition to that, Next join the list to get back into the string format. Next, convert the string into lowercase. Then, extract only the letters from the string. This task is performed using Regular Expressions. Next, Tokenized the string by using the word tokenizer.

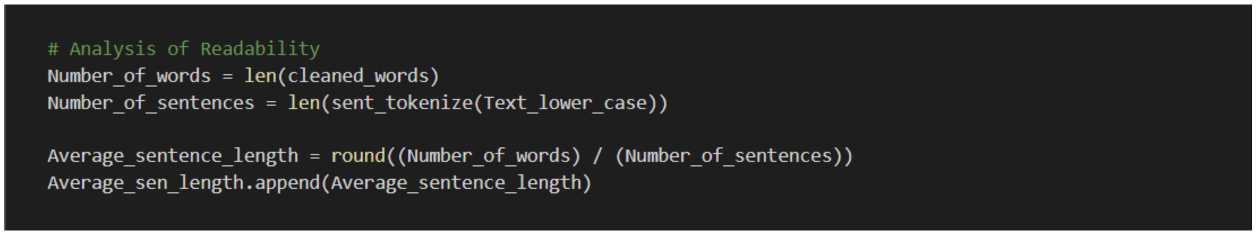


```
#Text Anlaysis
#Importing data
with open(file_name,mode='r',encoding='utf_8') as f:
    Original_text = f.read()
    f.close()


# Cleaning Data
#Splitting it by '?' to get only text an (Leaving Title)Incase if we have more than two "?" in text
#We are getting list from index 1

Text = Original_text.split('?')[1:]
get_back_string = ' '.join(Text)
Text_lower_case = get_back_string.lower()
cleaned_Text = re.sub('[^a-zA-Z]',' ',Text_lower_case)
cleaned_words = word_tokenize(cleaned_Text)
```

1.ANALYSIS OF READABILITY:

Analysis of Readability is calculated by using the "FOG INDEX" formula described below.

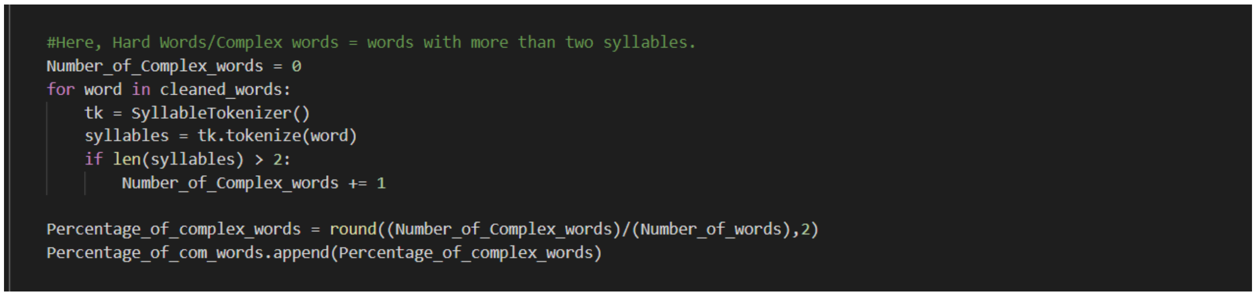
AVERAGE SENTENCE LENGTH: Number of words in the text / Number of sentences in the text. Appended the results to the respective list.



```
# Analysis of Readability
Number_of_words = len(cleaned_words)
Number_of_sentences = len(sent_tokenize(Text_lower_case))

Average_sentence_length = round((Number_of_words) / (Number_of_sentences))
Average_sen_length.append(Average_sentence_length)
```

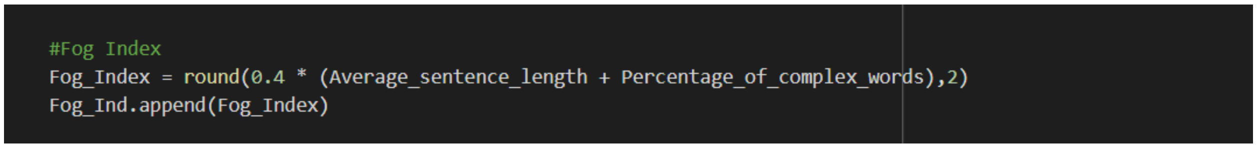
PERCENTAGE OF COMPLEX WORDS: Number of complex words / Number of words in the text.
Here, Hard Words/Complex words = words with more than two syllables.

First, Get the words one by one from the cleande_words. Then tokenized the word by using “SyllableTokenizer”. Next, we get the syllables of the word in the list and then will check if the syllable is greater than 2 syllables will count it as one else pass. Appended the results to the respective list.



```
#Here, Hard Words/Complex words = words with more than two syllables.
Number_of_Complex_words = 0
for word in cleaned_words:
    tk = SyllableTokenizer()
    syllables = tk.tokenize(word)
    if len(syllables) > 2:
        Number_of_Complex_words += 1

Percentage_of_complex_words = round((Number_of_Complex_words)/(Number_of_words),2)
Percentage_of_com_words.append(Percentage_of_complex_words)
```

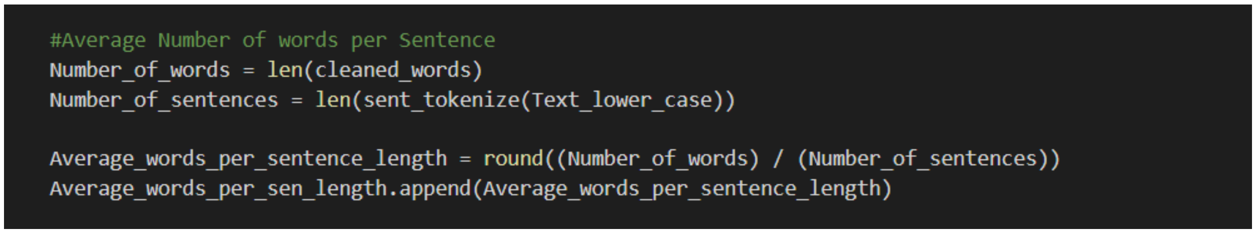
FOG INDEX: (0.4 * (AVERAGE SENTENCE LENGTH + PERCENTAGE OF COMPLEX WORDS))
Appended the results to the respective list.



```
#Fog Index
Fog_Index = round(0.4 * (Average_sentence_length + Percentage_of_complex_words),2)
Fog_Ind.append(Fog_Index)
```

2. AVERAGE NUMBER OF WORDS PER SENTENCE:

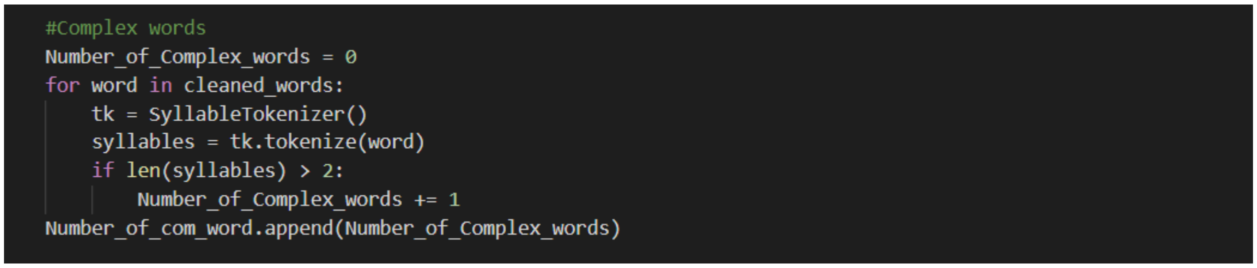
AVERAGE NUMBER OF WORDS PER SENTENCE : Total number of words / Total number of sentences in the text. Appended the results to the respective list.



```
#Average Number of words per Sentence
Number_of_words = len(cleaned_words)
Number_of_sentences = len(sent_tokenize(Text_lower_case))

Average_words_per_sentence_length = round((Number_of_words) / (Number_of_sentences))
Average_words_per_sen_length.append(Average_words_per_sentence_length)
```

3. COMPLEX WORD COUNT: Complex words are the words that contain more than two syllables. Appended the results to the respective list.

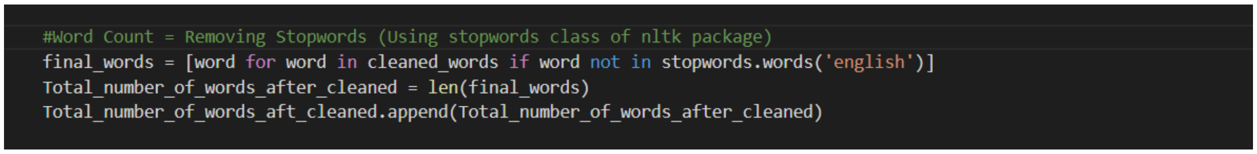
Here, Hard Words/Complex words = words with more than two syllables.



```
#Complex words
Number_of_Complex_words = 0
for word in cleaned_words:
    tk = SyllableTokenizer()
    syllables = tk.tokenize(word)
    if len(syllables) > 2:
        Number_of_Complex_words += 1
Number_of_com_word.append(Number_of_Complex_words)
```

4. WORD COUNT:

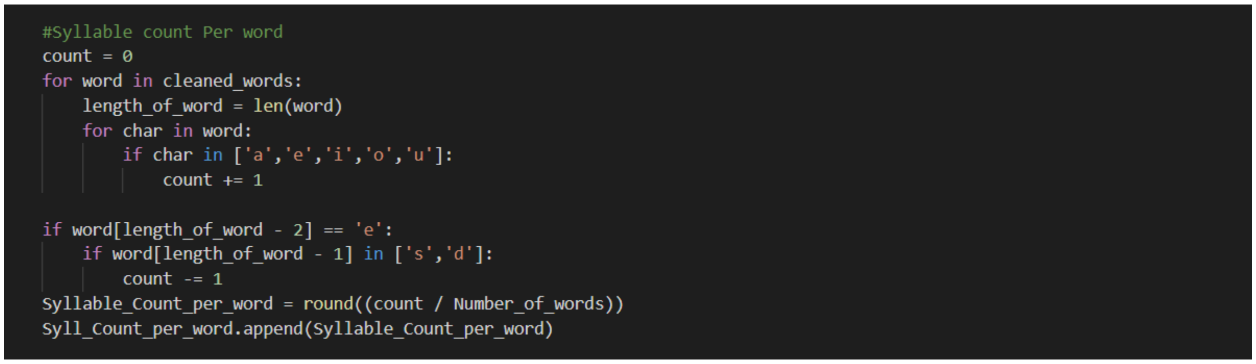
Extracted only the letters from the string(a-zA-Z). This task is performed using Regular Expressions (Cleaned_Text). Next, Tokenized the Cleaned_Text(Cleaned words). After tokenizing, filter the text of stopwords and get the count of the words. Appended the results to the respective list.



```
#Word Count = Removing Stopwords (Using stopwords class of nltk package)
final_words = [word for word in cleaned_words if word not in stopwords.words('english')]
Total_number_of_words_after_cleaned = len(final_words)
Total_number_of_words_aft_cleaned.append(Total_number_of_words_after_cleaned)
```

5. SYLLABLE COUNT PER WORD:

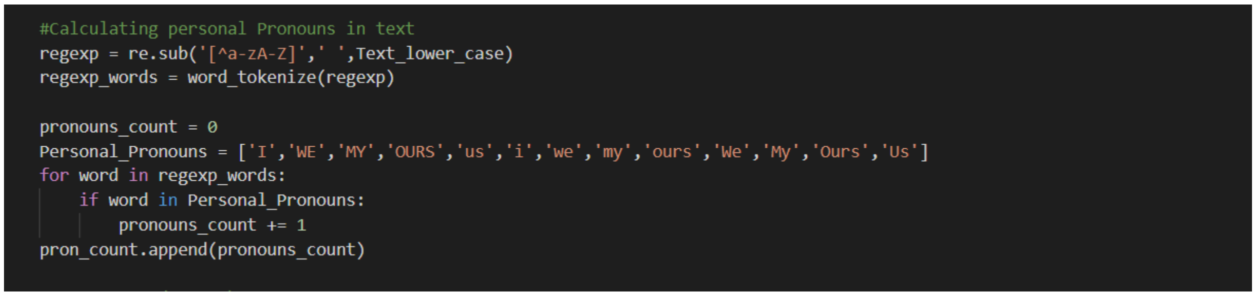
Count the number of Syllables in each word of the text by counting the vowels present in each word. Also handled some exceptions like words ending with "es","ed" by not counting them as a syllable. Appended the results to the respective list.



```
#Syllable count Per word
count = 0
for word in cleaned_words:
    length_of_word = len(word)
    for char in word:
        if char in ['a','e','i','o','u']:
            count += 1

if word[length_of_word - 2] == 'e':
    if word[length_of_word - 1] in ['s','d']:
        count -= 1
Syllable_Count_per_word = round((count / Number_of_words))
Syll_Count_per_word.append(Syllable_Count_per_word)
```

6. PERSONAL PRONOUNS:

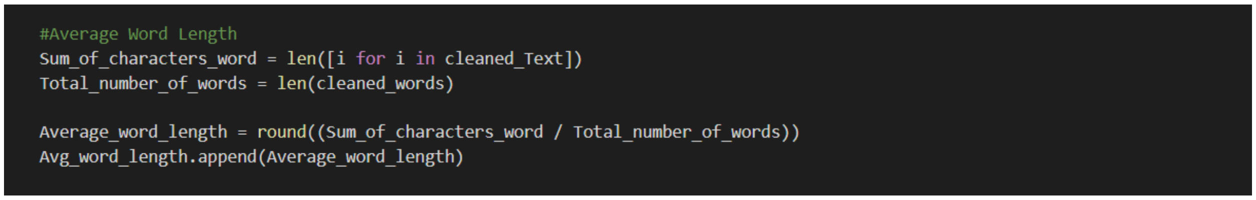
To calculate Personal Pronouns mentioned in the text, use regex to find the counts of the words - “I,” “we,” “my,” “ours,” and “us”. Special care is taken so that the country name US is not included in the list. Appended the results to the respective list.



```
#Calculating personal Pronouns in text
regexp = re.sub('[^a-zA-Z]',' ',Text_lower_case)
regexp_words = word_tokenize(regexp)

pronouns_count = 0
Personal_Pronouns = ['I','WE','MY','OURS','us','i','we','my','ours','We','My','Ours','Us']
for word in regexp_words:
    if word in Personal_Pronouns:
        pronouns_count += 1
pron_count.append(pronouns_count)
```

7. AVERAGE WORD LENGTH:

AVERAGE WORD LENGTH: Total number of characters in the string/text(Cleaned_Text) / Total number of words in the text/string. Appended the results to the respective list.



```
#Average Word Length
Sum_of_characters_word = len([i for i in cleaned_Text])
Total_number_of_words = len(cleaned_words)

Average_word_length = round((Sum_of_characters_word / Total_number_of_words))
Avg_word_length.append(Average_word_length)
```

8. SENTIMENT ANALYSIS:

Tokenized the Cleaned_Text. After tokenizing, we filtered the text of stop words. At last, lemmatized the remaining words and returned these words.

Created a Positive and Negative text files from external sources to find out the Positive and Negative score for the lemmatized words.

Positive Score: This score is calculated by assigning the value of +1 for each word if found in the Positive Dictionary and then adding up all the values.

Negative Score: This score is calculated by assigning the value of -1 for each word if found in the Negative Dictionary and then adding up all the values. We multiply the score with -1 so that the score is a positive number.

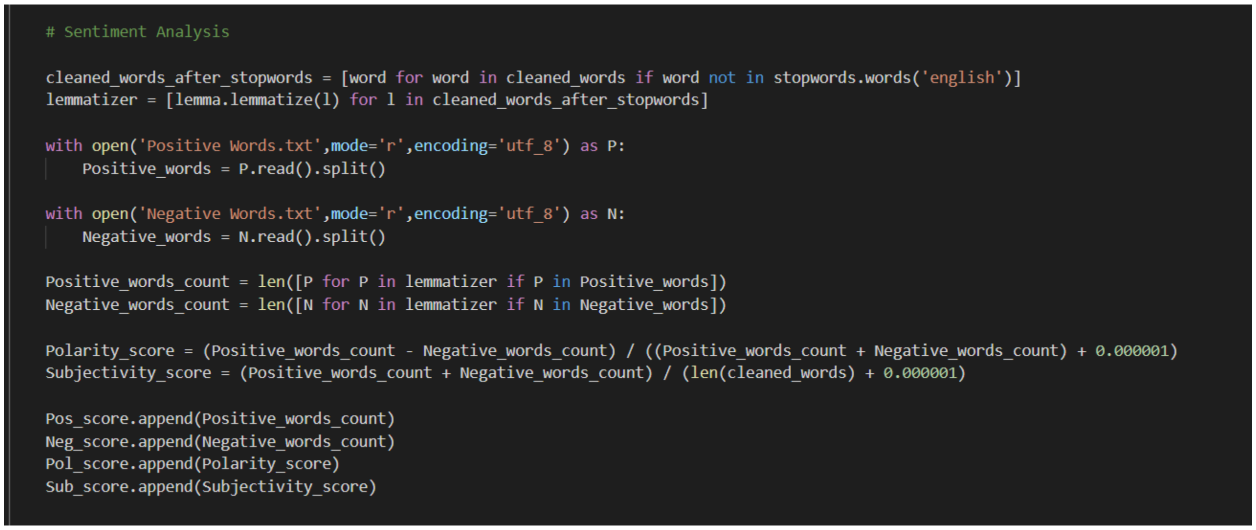
Polarity Score: This is the score that determines if a given text is positive or negative in nature. It is calculated by using the formula:

Polarity Score = (Positive Score – Negative Score)/ ((Positive Score + Negative Score) + 0.000001). Range is from -1 to +1

Subjectivity Score: This is the score that determines if a given text is objective or subjective. It is calculated by using the formula:

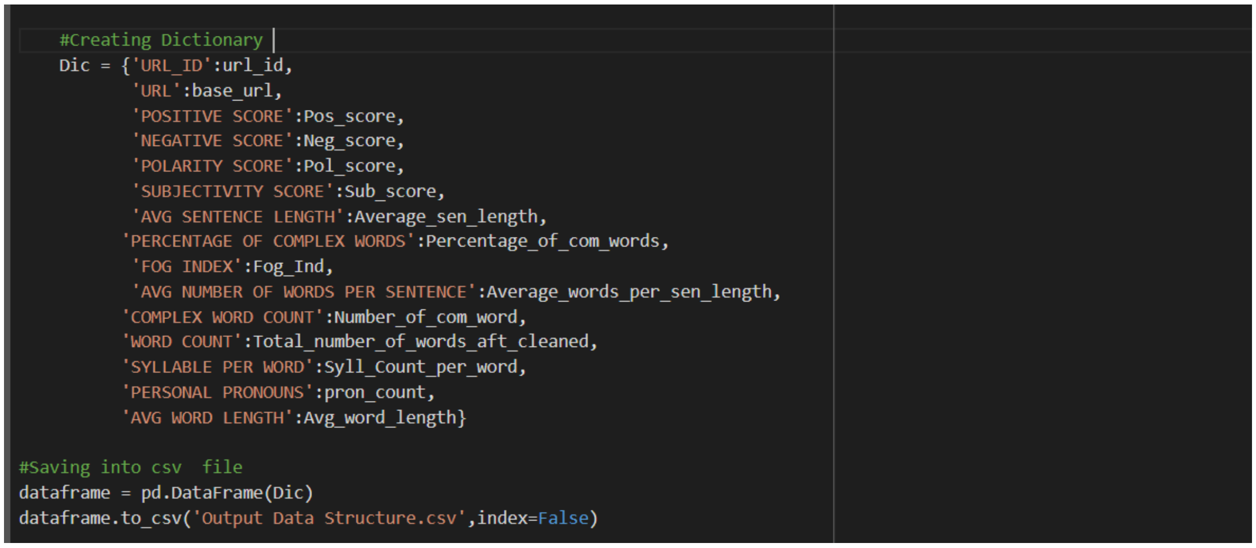
Subjectivity Score = (Positive Score + Negative Score)/ ((Total Words after cleaning) + 0.000001). Range is from 0 to +1

Appended the results to the respective lists.



```python
# Sentiment Analysis

cleaned_words_after_stopwords = [word for word in cleaned_words if word not in stopwords.words('english')]
lemmatizer = [lemma.lemmatize(l) for l in cleaned_words_after_stopwords]

with open('Positive Words.txt',mode='r',encoding='utf_8') as P:
    Positive_words = P.read().split()

with open('Negative Words.txt',mode='r',encoding='utf_8') as N:
    Negative_words = N.read().split()

Positive_words_count = len([P for P in lemmatizer if P in Positive_words])
Negative_words_count = len([N for N in lemmatizer if N in Negative_words])

Polarity_score = (Positive_words_count - Negative_words_count) / ((Positive_words_count + Negative_words_count) + 0.000001)
Subjectivity_score = (Positive_words_count + Negative_words_count) / (len(cleaned_words) + 0.000001)

Pos_score.append(Positive_words_count)
Neg_score.append(Negative_words_count)
Pol_score.append(Polarity_score)
Sub_score.append(Subjectivity_score)
```

For saving the data into a CSV file. Creating the dictionary with respective keys and values. And then creating dataframe and saving it into CSV Format.



```python
    #Creating Dictionary
    Dic = {'URL_ID':url_id,
           'URL':base_url,
           'POSITIVE SCORE':Pos_score,
           'NEGATIVE SCORE':Neg_score,
           'POLARITY SCORE':Pol_score,
           'SUBJECTIVITY SCORE':Sub_score,
           'AVG SENTENCE LENGTH':Average_sen_length,
          'PERCENTAGE OF COMPLEX WORDS':Percentage_of_com_words,
           'FOG INDEX':Fog_Ind,
           'AVG NUMBER OF WORDS PER SENTENCE':Average_words_per_sen_length,
          'COMPLEX WORD COUNT':Number_of_com_word,
          'WORD COUNT':Total_number_of_words_aft_cleaned,
          'SYLLABLE PER WORD':Syll_Count_per_word,
          'PERSONAL PRONOUNS':pron_count,
          'AVG WORD LENGTH':Avg_word_length}

#Saving into csv  file
dataframe = pd.DataFrame(Dic)
dataframe.to_csv('Output Data Structure.csv',index=False)
```