```
import pandas as pd
from requests import Session
from bs4 import BeautifulSoup
import string
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

```
df = pd.read_excel('/content/Input.xlsx');
df.head()
```

```
   URL_ID                                                URL
0   123.0  https://insights.blackcoffer.com/rise-of-telem...
1   321.0  https://insights.blackcoffer.com/rise-of-e-hea...
2  2345.0  https://insights.blackcoffer.com/rise-of-e-hea...
3  4321.0  https://insights.blackcoffer.com/rise-of-telem...
4   432.0  https://insights.blackcoffer.com/rise-of-telem...
```

```
URL_LIST = [x[0] for x in  df[['URL']].values.tolist()]
URL_ID = [x[0] for x in  df[['URL_ID']].values.tolist()]

FEATCH = Session().get
```

From this Jupyter cell, we will analyze a single page to test the functions provided in the documentation.

```
the_first_url = URL_LIST[0];
the_first_url
print(len(URL_LIST))
```

```
114
```

```
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36'
}
```

single line of code fetches the HTML content of a web page located at the URL

```
the_html_code = FEATCH(the_first_url,headers=headers).text
```

BeautifulSoup Object

```
the_html_code = FEATCH(the_first_url,headers=headers).text
the_bs4_object = BeautifulSoup(the_html_code,'lxml');

the_titles = the_bs4_object.find("div",{"class":'td-post-content
tagdiv-type'}).find_all('p')

[x.text for x in the_titles].__len__() ### length of total article

29
```

Download the (NEGATIVE and POSITIVE) txt file from drive

```
!wget "https://drive.usercontent.google.com/uc?id=1qqMwc_-
ayS38HEOB97osO_nkIxRkbnvh&authuser=0&export=download" > "neg.txt"
!wget "https://drive.usercontent.google.com/uc?
id=1seAj8G42SmfgUUx8lqVDJofm4Tuh2TOT&authuser=0&export=download" >
"positive.txt"
```

Sentiment Analysis

Sentiment analysis is the process of determining whether a piece of writing is positive, negative, or neutral. The below Algorithm is designed for use in Financial Texts. It consists of steps:

Cleaning using Stop Words Lists

The Stop Words Lists (found in the folder StopWords) are used to clean the text so that Sentiment Analysis can be performed by excluding the words found in Stop Words List.

Creating a dictionary of Positive and Negative words

The Master Dictionary (found in the folder MasterDictionary) is used for creating a dictionary of Positive and Negative words. We add only those words in the dictionary if they are not found in the Stop Words Lists.

Extracting Derived Variables

```
with open("neg.txt",'r') as f , open("positive.txt",'r') as f2:
  the_positive_text_list = f2.read();# THIS USE FOR,.....positive
WORDS
  the_negative_text_list = f.read();# ''' THIS CODE HELP TO OPEN THE
FILE USING F  '''

the_positive_text_list = ([x for x in the_positive_text_list.split('\
n') if x])### all POS words
the_negative_text_list =  ([x for x in the_negative_text_list.split('\
n') if x])#### all neg words
```

```
def clean_sentence(sentence):
    words = nltk.word_tokenize(sentence)
    cleaned_words = []
    stop_words = set(stopwords.words("english"))
    for word in words:
        # Remove punctuation and convert to lowercase
        word = word.lower()

        # Check if the word is not a stopword
        if word not in stop_words and word.isalpha():
            cleaned_words.append(word)
    cleaned_sentence = ' '.join(cleaned_words)

    return cleaned_sentence
### TEST CODE
clean_sentence("Another factor contributing to the rise of
telemedicine is the need for improved access to healthcare. Patients
may travel long distances in many rural and underserved areas;
patients may travel long distances to access medical care.
Telemedicine can bridge this gap by allowing patients in these areas
to consult with healthcare providers remotely. This not only improves
access to healthcare but also reduces the need for patients to travel
long distances, saving them time and money.")
```

Polarity Analysis

- **Positive Score:** This score is calculated by assigning a value of +1 for each word found in the Positive Dictionary and then summing up all the values.
- **Negative Score:** This score is calculated by assigning a value of -1 for each word found in the Negative Dictionary and then summing up all the values. We multiply the score by -1 to ensure it is a positive number.
- **Polarity Score:** The Polarity Score determines whether a given text is positive or negative in nature. It is calculated using the formula:

```
def counter_function_pos(paragraph):
  return sum(1 for word in the_positive_text_list if word in
paragraph);

def counter_function_neg(paragraph):
  return -1*(sum(-1 for word in the_negative_text_list if word in
paragraph));

def calculate_polarity_score(positive_score, negative_score):
    denominator = (positive_score + negative_score) + 0.000001
    polarity_score = (positive_score - negative_score) / denominator
    return polarity_score
```

```
def subjectivity_scores(positive_score,negative_score):

  subjectivity_score = (positive_score + negative_score) / (
      len(''.join([x.text for x in the_titles]))
       + 0.000001)

  return subjectivity_score

all_paragraph_pos = sum(list(map(counter_function_pos,[x.text for x in
the_titles])))

all_paragraph_neg =  sum(list(map(counter_function_neg,[x.text for x
in the_titles])))

print((all_paragraph_pos),
(all_paragraph_neg),calculate_polarity_score(all_paragraph_pos,all_par
agraph_neg));

print(subjectivity_scores(all_paragraph_pos, all_paragraph_neg))
```

Gunning Fox Index Formula for Readability Analysis

The Gunning Fox index is a formula used to assess the readability of a text. It takes into account two main factors:

1. **Average Sentence Length**: This is calculated by dividing the total number of words by the total number of sentences in the text.
2. **Percentage of Complex Words**: This is determined by dividing the number of complex words by the total number of words in the text.
3. **Fog Index** is calculated as the sum of these two factors, multiplied by 0.4:

```
def calculate_average_sentence_length(text):
    # Tokenize the text into sentences
    sentences = text.split('.')
    total_words = 0
    total_sentences = len(sentences)
    for sentence in sentences:
        words = sentence.split()
        total_words += len(words)
    average_length = total_words / total_sentences
    return average_length

def count_syllables(word):
    vowels = "aeiouy"
    word = word.lower()
    if len(word) <= 3:
        return 1
```

```
    if word.endswith('es') or word.endswith('ed'):
        word = word[:-2]
    count = 0
    prev_char_was_vowel = False
    for char in word:
        if char in vowels:
            if not prev_char_was_vowel:
                count += 1
            prev_char_was_vowel = True
        else:
            prev_char_was_vowel = False
    return count

def percentage_of_complex_words (text):
    words = text.split()
    number_of_all = len(words)

    complex_words = []

    for word in words:

        word_count = count_syllables(''.join(filter(str.isalpha,
word)))

        if 2 < (word_count):
            complex_words.append(word)
    try:
      v=  len(complex_words) / number_of_all
      return v
    except:
      return 0

def fog_index_of_full_page(full_page_list_of_words):
  the_all_words = [x.text for x in full_page_list_of_words]
  fox_index = 0;
  for paragraph in the_all_words:
    try:
      fox_index += 0.4*(calculate_average_sentence_length(paragraph)
      / percentage_of_complex_words(paragraph));
    except:
      fox_index+= 0
  return fox_index



import re

def average_sentence_length(paragraph):

    sentences = re.split(r'[.!?]', paragraph)
```

```
    total_sentences = len(sentences)
    total_words = len(paragraph.split())


    if total_sentences > 0:
        avg_sentence_length = total_words / total_sentences
        return avg_sentence_length
    else:
        return 0

# average_sentence_length()

# calculate_average_sentence_length() ---> Average Number of Words Per
Sentenc

def complex_word_count(text):
    words = text.split()
    number_of_all = len(words)
    complex_words = 0
    for word in words:
        word_count = count_syllables(''.join(filter(str.isalpha,
word)))
        if 2 < (word_count):
            complex_words+=1;
    return complex_words;

print(complex_word_count([x.text for x in the_titles][0]))
complex_word_count_one_page = sum(list(map(complex_word_count, [x.text
for x in the_titles])))
complex_word_count_one_page
```

```
13
```

```
460
```

```
def the_total_number_of_word(text):
    words = nltk.word_tokenize(text)
    stopwords_set = set(stopwords.words("english"))
    cleaned_words = [word for word in words if word.lower() not in
stopwords_set and word not in string.punctuation]

    cleaned_text = " ".join(cleaned_words)
    return len(cleaned_text)

the_total_number_of_word_count = the_total_number_of_word([x.text for
x in the_titles][0])

the_total_number_of_word_count
```

```
8322
```

```
count_syllables_total = sum(list(map(count_syllables,[x.text for x in
the_titles])))
print(count_syllables_total)
```

```
3412
```

```
def count_personal_pronouns(text):
    words = text.split()
    personal_pronouns = ["I", "we", "my", "ours", "us"]
    count = 0
    for word in words:
        word_lower = word.lower()
        if word_lower in personal_pronouns and word_lower != "us":
            count += 1
    return count

count_personal_pronouns_sum = sum(list(map(count_personal_pronouns,
[x.text for x in the_titles])))
```

```
def average_word_length(text):
    words = text.split()
    total_characters = sum(len(word) for word in words)
    total_words = len(words)
    try:
      v = total_characters / total_words
      return v
    except:
      return 0

average_word_length_sum =  sum(list(map(average_word_length,[x.text
for x in the_titles])))
average_word_length_sum
```

```
!wget "https://drive.usercontent.google.com/uc?
id=1kHcx9epaZKB96zRItudnrDi57cFEndFI&authuser=0&export=download" >
output.xlsx

out_put_df = pd.read_excel('/content/output.xlsx');

the_dict_for_future = dict.fromkeys(out_put_df.columns.tolist(),[])
from pprint import pprint
pprint(the_dict_for_future)
the_dict_for_future['URL'] = URL_LIST
the_dict_for_future['URL_ID'] = URL_ID
# the_dict_for_future
the_dict_for_future['URL']

from time import sleep

def main_function():
  the_dict_for_future = dict.fromkeys(out_put_df.columns.tolist(),
```

```
[0]*len(URL_LIST))
  the_dict_for_future['URL'] = URL_LIST
  the_dict_for_future['URL_ID'] = URL_ID
  for i,lins in  enumerate(URL_LIST):
    the_html_code = FEATCH(lins,headers=headers).text
    the_bs4_object = BeautifulSoup(the_html_code,'lxml')
    try:
      the_titles = the_bs4_object.find("div",{"class":'td-post-content
tagdiv-type'}).find_all('p');
      # the_mapper_object = [x.text for x in the_titles]
    except:
      the_titles = the_bs4_object.find("div",{"class":'tdb-block-inner
td-fix-index'}).find_all('p');
      # the_mapper_object = [x.text for x in the_titles]
    ## TASK 1)
    all_paragraph_pos = sum(list(map(counter_function_pos,[x.text for
x in the_titles])))
    all_paragraph_neg =  sum(list(map(counter_function_neg,[x.text for
x in the_titles])))
    all_calculate_polarity_score =
calculate_polarity_score(all_paragraph_pos,all_paragraph_neg)
    subjectivity_scores_ =
subjectivity_scores(positive_score=all_paragraph_pos,negative_score=al
l_paragraph_neg);

    the_dict_for_future['POLARITY SCORE'][i] =
(all_calculate_polarity_score);
    the_dict_for_future['POSITIVE SCORE'][i] = (all_paragraph_pos);
    the_dict_for_future['SUBJECTIVITY SCORE'][i] =
(subjectivity_scores_);
    the_dict_for_future['NEGATIVE SCORE'][i] = (all_paragraph_neg);

    # TASK 2)average_sentence_length calculate_average_sentence_length
    calculate_average_sentence_length_ =
sum(list(map(average_sentence_length,[x.text for x in the_titles])))
    percantage_of_compex =  sum(list(map(percentage_of_complex_words,
[x.text for x in the_titles])))
    fog_index = sum(list(map(percentage_of_complex_words, [x.text for
x in the_titles])))

    the_dict_for_future['AVG SENTENCE LENGTH'][i] =
(calculate_average_sentence_length_);
    the_dict_for_future['FOG INDEX'][i] = (fog_index);
    the_dict_for_future['PERCENTAGE OF COMPLEX WORDS'][i] =
(percantage_of_compex);

    # Task 3)
    Average_Number_of_Words =
sum(list(map(calculate_average_sentence_length,[x.text for x in
the_titles])))
```

```
    the_dict_for_future["AVG NUMBER OF WORDS PER SENTENCE"][i] =
(Average_Number_of_Words)

    ## Task 4)
    complex_word_count_ =  sum(list(map(complex_word_count, [x.text
for x in the_titles])))
    the_dict_for_future['COMPLEX WORD COUNT'][i] =
(complex_word_count_);

    # Task 5)
    the_total_number_of_word_count_clean =
sum(list(map(the_total_number_of_word,[x.text for x in the_titles])))
    the_dict_for_future['WORD COUNT'][i] =
(the_total_number_of_word_count_clean);
    ################ Task 6)
    count_syllables_ = sum(list(
        map(count_syllables, [x.text for x in the_titles])
    ));
    the_dict_for_future['SYLLABLE PER WORD'][i] = (count_syllables_);

    ############## Task 7)
    count_personal_pronouns_sum =
sum(list(map(count_personal_pronouns,[x.text for x in the_titles])))
    the_dict_for_future['PERSONAL PRONOUNS'][i] =
(count_personal_pronouns_sum)

    ############# Task 8)
    average_word_length_sum =  sum(list(map(average_word_length,
[x.text for x in the_titles])))
    the_dict_for_future['AVG NUMBER OF WORDS PER SENTENCE'][i] =
(average_word_length_sum);

  return the_dict_for_future

result_data = main_function()
result_df = pd.DataFrame(result_data)
result_df.to_excel('output.xlsx', index=False)

result_df.head(30)
```

	URL_ID	URL	\
0	123.0	https://insights.blackcoffer.com/rise-of-telem...	
1	321.0	https://insights.blackcoffer.com/rise-of-e-hea...	
2	2345.0	https://insights.blackcoffer.com/rise-of-e-hea...	
3	4321.0	https://insights.blackcoffer.com/rise-of-telem...	
4	432.0	https://insights.blackcoffer.com/rise-of-telem...	
5	2893.8	https://insights.blackcoffer.com/rise-of-chatb...	
6	3355.6	https://insights.blackcoffer.com/rise-of-e-hea...	
7	3817.4	https://insights.blackcoffer.com/how-does-mark...	
8	4279.2	https://insights.blackcoffer.com/how-advertise...	

```
9    4741.0  https://insights.blackcoffer.com/negative-effe...
10   5202.8  https://insights.blackcoffer.com/how-advertise...
11   5664.6  https://insights.blackcoffer.com/rising-it-cit...
12   6126.4  https://insights.blackcoffer.com/rise-of-ott-p...
13   6588.2  https://insights.blackcoffer.com/rise-of-elect...
14   7050.0  https://insights.blackcoffer.com/rise-of-elect...
15   7511.8  https://insights.blackcoffer.com/oil-prices-by...
16   7973.6  https://insights.blackcoffer.com/an-outlook-of...
17   8435.4  https://insights.blackcoffer.com/ai-in-healthc...
18   8897.2  https://insights.blackcoffer.com/what-if-the-c...
19   9359.0  https://insights.blackcoffer.com/what-jobs-wil...
20   9820.8  https://insights.blackcoffer.com/will-machine-...
21  10282.6  https://insights.blackcoffer.com/will-ai-repla...
22  10744.4  https://insights.blackcoffer.com/man-and-machi...
23  11206.2  https://insights.blackcoffer.com/in-future-or-...
24  11668.0  https://insights.blackcoffer.com/how-neural-ne...
25  12129.8  https://insights.blackcoffer.com/how-machine-l...
26  12591.6  https://insights.blackcoffer.com/deep-learning...
27  13053.4  https://insights.blackcoffer.com/how-to-protec...
28  13515.2  https://insights.blackcoffer.com/how-machines-...
29  13977.0  https://insights.blackcoffer.com/ai-human-robo...

    POSITIVE SCORE  NEGATIVE SCORE  POLARITY SCORE  SUBJECTIVITY SCORE
\
0       166.235687      166.235687      166.235687          166.235687

1        33.260780       33.260780       33.260780           33.260780

2         0.000000        0.000000        0.000000            0.000000

3       131.087105      131.087105      131.087105          131.087105

4       131.087105      131.087105      131.087105          131.087105

5       101.172242      101.172242      101.172242          101.172242

6       119.319100      119.319100      119.319100          119.319100

7       232.976479      232.976479      232.976479          232.976479

8         0.000000        0.000000        0.000000            0.000000

9        57.882842       57.882842       57.882842           57.882842

10       26.430998       26.430998       26.430998           26.430998

11       46.583958       46.583958       46.583958           46.583958

12       31.786641       31.786641       31.786641           31.786641

13       70.063834       70.063834       70.063834           70.063834
```

14	48.520971	48.520971	48.520971	48.520971
15	33.194189	33.194189	33.194189	33.194189
16	83.861939	83.861939	83.861939	83.861939
17	0.000000	0.000000	0.000000	0.000000
18	75.894536	75.894536	75.894536	75.894536
19	159.132264	159.132264	159.132264	159.132264
20	149.240261	149.240261	149.240261	149.240261
21	153.598776	153.598776	153.598776	153.598776
22	73.576254	73.576254	73.576254	73.576254
23	30.754827	30.754827	30.754827	30.754827
24	0.000000	0.000000	0.000000	0.000000
25	35.651265	35.651265	35.651265	35.651265
26	165.892598	165.892598	165.892598	165.892598
27	108.835923	108.835923	108.835923	108.835923
28	185.027006	185.027006	185.027006	185.027006
29	92.018748	92.018748	92.018748	92.018748

	AVG SENTENCE LENGTH	PERCENTAGE OF COMPLEX WORDS	FOG INDEX	\
0	166.235687	166.235687	166.235687	
1	33.260780	33.260780	33.260780	
2	0.000000	0.000000	0.000000	
3	131.087105	131.087105	131.087105	
4	131.087105	131.087105	131.087105	
5	101.172242	101.172242	101.172242	
6	119.319100	119.319100	119.319100	
7	232.976479	232.976479	232.976479	
8	0.000000	0.000000	0.000000	
9	57.882842	57.882842	57.882842	
10	26.430998	26.430998	26.430998	
11	46.583958	46.583958	46.583958	
12	31.786641	31.786641	31.786641	
13	70.063834	70.063834	70.063834	
14	48.520971	48.520971	48.520971	
15	33.194189	33.194189	33.194189	

```
16             83.861939                    83.861939   83.861939
17              0.000000                     0.000000    0.000000
18             75.894536                    75.894536   75.894536
19            159.132264                   159.132264  159.132264
20            149.240261                   149.240261  149.240261
21            153.598776                   153.598776  153.598776
22             73.576254                    73.576254   73.576254
23             30.754827                    30.754827   30.754827
24              0.000000                     0.000000    0.000000
25             35.651265                    35.651265   35.651265
26            165.892598                   165.892598  165.892598
27            108.835923                   108.835923  108.835923
28            185.027006                   185.027006  185.027006
29             92.018748                    92.018748   92.018748

    AVG NUMBER OF WORDS PER SENTENCE  COMPLEX WORD COUNT  WORD
COUNT  \
0                          166.235687          166.235687  166.235687

1                           33.260780           33.260780   33.260780

2                            0.000000            0.000000    0.000000

3                          131.087105          131.087105  131.087105

4                          131.087105          131.087105  131.087105

5                          101.172242          101.172242  101.172242

6                          119.319100          119.319100  119.319100

7                          232.976479          232.976479  232.976479

8                            0.000000            0.000000    0.000000

9                           57.882842           57.882842   57.882842

10                          26.430998           26.430998   26.430998

11                          46.583958           46.583958   46.583958

12                          31.786641           31.786641   31.786641

13                          70.063834           70.063834   70.063834

14                          48.520971           48.520971   48.520971

15                          33.194189           33.194189   33.194189

16                          83.861939           83.861939   83.861939
```

17	0.000000	0.000000	0.000000
18	75.894536	75.894536	75.894536
19	159.132264	159.132264	159.132264
20	149.240261	149.240261	149.240261
21	153.598776	153.598776	153.598776
22	73.576254	73.576254	73.576254
23	30.754827	30.754827	30.754827
24	0.000000	0.000000	0.000000
25	35.651265	35.651265	35.651265
26	165.892598	165.892598	165.892598
27	108.835923	108.835923	108.835923
28	185.027006	185.027006	185.027006
29	92.018748	92.018748	92.018748

	SYLLABLE PER WORD	PERSONAL PRONOUNS	AVG WORD LENGTH
0	166.235687	166.235687	166.235687
1	33.260780	33.260780	33.260780
2	0.000000	0.000000	0.000000
3	131.087105	131.087105	131.087105
4	131.087105	131.087105	131.087105
5	101.172242	101.172242	101.172242
6	119.319100	119.319100	119.319100
7	232.976479	232.976479	232.976479
8	0.000000	0.000000	0.000000
9	57.882842	57.882842	57.882842
10	26.430998	26.430998	26.430998
11	46.583958	46.583958	46.583958
12	31.786641	31.786641	31.786641
13	70.063834	70.063834	70.063834
14	48.520971	48.520971	48.520971
15	33.194189	33.194189	33.194189
16	83.861939	83.861939	83.861939
17	0.000000	0.000000	0.000000
18	75.894536	75.894536	75.894536
19	159.132264	159.132264	159.132264
20	149.240261	149.240261	149.240261
21	153.598776	153.598776	153.598776
22	73.576254	73.576254	73.576254

23	30.754827	30.754827	30.754827
24	0.000000	0.000000	0.000000
25	35.651265	35.651265	35.651265
26	165.892598	165.892598	165.892598
27	108.835923	108.835923	108.835923
28	185.027006	185.027006	185.027006
29	92.018748	92.018748	92.018748