Instructions for BlackCoffer Assignment **ABHINAV TAMTA**

Notebook link:

https://colab.research.google.com/drive/19xbfwq0VPcvZaKmPFCai4O_egYmpXhDb?usp=sharing

Using Beautifulsoup for webcrawling using the 'URL_ID' in Input.xlsx

Web crawling using Beautiful soup ,sentiment analysis and readability analysis.

Mounting the google drive.

```
[ ] from google.colab import drive
    drive.flush_and_unmount()
```

```
Drive not mounted, so nothing to flush and unmount.
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive2
```

Web crawling using Beautifulsoup on Input.xlsx url-ids.

Web crawling using Beautifulsoup on Input.xlsx url-ids.

```
# Import required libraries
import pandas as pd
import requests
from bs4 import BeautifulSoup
from google.colab import drive
import os

# Mount the Google Drive to save the output files
#drive.mount('/content/drive')

# Read the input Excel file
input_file = pd.read_excel('/content/drive/MyDrive/Input.xlsx')

# Define the output directory
output_dir = '/content/drive/MyDrive/URL_ID'

# Create the output directory if it does not exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Loop through each URL and extract the article text
for index, row in input_file.iterrows():
    URL_ID = row['URL_ID']
    URL = row['URL']
```

```
# Make a request to get the HTML content of the webpage
response = requests.get(URL)
soup = BeautifulSoup(response.content, 'html.parser')

# Extract the article title and text from the HTML content
title_tag = soup.find('title')  # find the title tag
article_title = title_tag.string if title_tag else ''  # get the text inside the title tag, or set to empty string if title tag not found

article_text = ''
for paragraph in soup.find_all('p'):
    article_text += paragraph.get_text() + '\n'

# Save the extracted article text to a text file
output_file_path = os.path.join(output_dir, '{}.txt'.format(URL_ID))
output_file = open(output_file_path, 'w')
output_file.write(article_title + '\n' + article_text)
output_file.close()
```

- This Python script that extracts the article title and text from a list of URLs and saves them to individual text files. The script uses the following libraries:

- **pandas** for reading an Excel file that contains the list of URLs requests for making HTTP requests to retrieve the HTML content of each webpage.
- **BeautifulSoup** for parsing the HTML content and extracting the article title and text
- **Os** for creating the output directory and saving the extracted text to text files
- **google.colab.drive** for mounting the Google Drive to save the output files (currently commented out).
- The script reads the list of URLs from an Excel file named "**Input.xlsx**" and loops through each URL. For each URL, it makes a request to retrieve the HTML content of the webpage and then uses BeautifulSoup to extract the article title and text. The extracted text is then saved to a text file in a directory named "URL_ID" (created if it doesn't exist) with the filename formatted as "{URL_ID}.txt"

Stopwords are removed from the .txt files and saved into the ‘URL_ID2’folder

Stop words removal from the txt files and storing them inside google drive in URL_ID2 folder.

```
import os

# Define paths to input and output folders
URL_ID_path = '/content/drive/MyDrive/URL_ID'
stopwords_path = '/content/drive/MyDrive/BlackCOffer/StopWords'
filtered_path = '/content/drive/MyDrive/URL_ID2'

# Create the Filtered folder if it does not exist
if not os.path.exists(filtered_path):
    os.makedirs(filtered_path)

# Loop through each file in the URL_ID folder
for filename in os.listdir(URL_ID_path):
    if filename.endswith('.txt'):
        # Read the content of the file
        with open(os.path.join(URL_ID_path, filename), 'r') as file:
            text = file.read()

        # Remove stop words from the text
        stop_words = set()
        for filename in os.listdir(stopwords_path):
            if filename.endswith('.txt'):
                with open(os.path.join(stopwords_path, filename), encoding='utf-8') as file:
                    stop_words.update(file.read().splitlines())
        filtered_text = ' '.join([word for word in text.split() if word.lower() not in stop_words])

        # Save the filtered text to a new file in the Filtered folder
        with open(os.path.join(filtered_path, filename), 'w') as file:
            file.write(filtered_text)
```

- This is a Python script that reads in text files from a directory named "URL_ID", removes stop words, and saves the filtered text to new files in a directory named "URL_ID2".

- The script uses the os module to define paths to the input and output folders. It then creates the "URL_ID2" folder if it does not exist.

- The script loops through each file in the "URL_ID" folder and reads in the content of the file. It then reads in a list of stop words from text files in a directory named "StopWords", removes the stop words from the text, and saves the filtered text to a new file in the "URL_ID2" folder.

- The output filenames are the same as the input filenames, so the filtered files are identified by the "URL_ID" in the filename.

Performing the sentiment analysis on the crawled .txt files

Performing sentiment analysis on it.

```
import os
import pandas as pd
from nltk.tokenize import word_tokenize

# Define paths for files and folders
stopwords_path = '/content/drive/MyDrive/BlackCOffer/StopWords'
positive_words_path = '/content/drive/MyDrive/BlackCOffer/MasterDictionary/positive-words.txt'
negative_words_path = '/content/drive/MyDrive/BlackCOffer/MasterDictionary/negative-words.txt'
input_folder_path = '/content/drive/MyDrive/URL_ID2'
output_folder_path = '/content/drive/MyDrive/Output'

# Create a set of stop words
stop_words = set()
for filename in os.listdir(stopwords_path):
    if filename.endswith('.txt'):
        with open(os.path.join(stopwords_path, filename), encoding='utf-8') as file:
            stop_words.update(file.read().splitlines())

# Create a dictionary of positive and negative words
positive_words = set()
with open(positive_words_path, encoding='utf-8') as file:
    for word in file.read().splitlines():
        if word.lower() not in stop_words:
            positive_words.add(word.lower())

negative_words = set()
with open(negative_words_path, encoding='utf-8') as file:
    for word in file.read().splitlines():
        if word.lower() not in stop_words:
            negative_words.add(word.lower())
```

```python
# Create a DataFrame to store the results
columns = ['Filename', 'Positive Score', 'Negative Score', 'Polarity Score', 'Subjectivity Score']
results = pd.DataFrame(columns=columns)

# Loop through each file in the input folder
for filename in os.listdir(input_folder_path):
    if filename.endswith('.txt'):
        # Read the text from the file
        with open(os.path.join(input_folder_path, filename), encoding='utf-8') as file:
            text = file.read()

        # Tokenize the text and remove stop words
        tokens = [word.lower() for word in word_tokenize(text) if word.lower() not in stop_words]

        # Calculate the positive and negative scores
        positive_score = sum([1 for token in tokens if token in positive_words])
        negative_score = -1 * sum([-1 for token in tokens if token in negative_words])

        # Calculate the polarity and subjectivity scores
        total_words = len(tokens)
        polarity_score = (positive_score - negative_score) / ((positive_score + negative_score) + 0.000001)
        subjectivity_score = (positive_score + negative_score) / (total_words + 0.000001)

        # Add the results to the DataFrame
        results = results.append({
            'Filename': filename,
            'Positive Score': positive_score,
            'Negative Score': negative_score,
            'Polarity Score': polarity_score,
            'Subjectivity Score': subjectivity_score
        }, ignore_index=True)

        # Save the filtered text to a new file in the Output folder
        filtered_text = ' '.join(tokens)
        with open(os.path.join(output_folder_path, filename), 'w', encoding='utf-8') as file:
            file.write(filtered_text)
```

```python
# Save the results to a CSV file
results.to_csv('/content/drive/MyDrive/Results.csv', index=False)
```

This Python code uses natural language processing (NLP) techniques to analyze the sentiment of text documents. The code reads in a set of text files located in a specified input folder, removes stop words, and calculates the positive and negative scores of the remaining words based on two sets of pre-defined positive and negative words. The code then calculates the polarity and subjectivity scores based on the positive and negative scores, and saves the results to a new file in a specified output folder. Finally, the code saves the results to a CSV file.

At this point it was not mentioned so I have saved the txt files with stopwords removed in URL_ID2 .

Along with it the original text is saved in the drive folder URL_ID. The sentiment analysis can be done on both of the folders containing the files. Here I have done it on URL_ID2 , which has stopwords removed .txt files as it makes the most sense to do so.

To achieve this, the code uses several Python libraries, including "os" for interacting with the operating system, "pandas" for data manipulation and storage, and "nltk" (Natural Language Toolkit) for NLP tasks such as tokenization.

The code starts by defining paths to the input and output folders, as well as to files containing lists of stop words, positive words, and negative words. It creates a set of stop words by reading in the stop word files, and then reads in the positive and negative words, excluding any stop words that may be present.

The code then creates a Pandas DataFrame with columns to store the results of the sentiment analysis, and loops through each file in the input folder. For each file, it reads in the text, tokenizes it, removes stop words, calculates positive and negative scores, and then calculates the polarity and subjectivity scores. The code then appends the results to the DataFrame, and saves the filtered text to a new file in the output folder.

Finally, the code saves the results to a CSV file, which includes the filename, positive score, negative score, polarity score, and subjectivity score for each text file analyzed.

Performing Readability analysis on the txt files

Performing readability analysis on the txt files and storing the metrics into a csv file.

```
import os
import csv
import string
import nltk
import glob
from textstat import flesch_reading_ease
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
nltk.download('stopwords')

# Function to calculate the average sentence length
def avg_sentence_length(text):
    sentences = sent_tokenize(text)
    words = word_tokenize(text)
    return len(words) / len(sentences)

# Function to calculate the percentage of complex words
def percent_complex_words(text):
    words = word_tokenize(text)
    complex_words = [word for word in words if len(word) > 2 and word not in stopwords.words('english')]
    return len(complex_words) / len(words)

# Function to calculate the Fog Index
```

```
# Function to calculate the average number of words per sentence
def avg_words_per_sentence(text):
    words = word_tokenize(text)
    sentences = sent_tokenize(text)
    return len(words) / len(sentences)

# Function to count the number of complex words
def complex_word_count(text):
    words = word_tokenize(text)
    complex_words = [word for word in words if len(word) > 2 and word not in stopwords.words('english')]
    return len(complex_words)

# Function to count the total number of words
def word_count(text):
    words = word_tokenize(text)
    cleaned_words = [word.lower().translate(str.maketrans('', '', string.punctuation)) for word in words if word not in stopwords.words('english')]
    return len(cleaned_words)
```

```python
# Function to count the number of syllables in each word
def syllable_count(word):
    vowels = 'aeiou'
    syllables = 0
    if word[0] in vowels:
        syllables += 1
    for i in range(1, len(word)):
        if word[i] in vowels and word[i-1] not in vowels:
            syllables += 1
        # Handle exceptions
        if word.endswith('es') or word.endswith('ed'):
            syllables -= 1
        if word.endswith('le') and word[-2] not in vowels:
            syllables += 1
    # Handle single letter words
    if len(word) == 1:
        syllables = 1
    return syllables



# Function to calculate the average word length
def avg_word_length(text):
    words = word_tokenize(text)
    total_characters = sum(len(word) for word in words)
    return total_characters / len(words)
```

```python
# Function to calculate the average word length
def avg_word_length(text):
    words = word_tokenize(text)
    total_characters = sum(len(word) for word in words)
    return total_characters / len(words)




def personal_pronouns(text):
    count = 0
    personal_pronouns_ = ['I', 'we', 'my', 'our', 'ours', 'us','We', 'My', 'Our', 'Ours', 'Us']
    for word in text.split():
        if word in personal_pronouns_:
            count += 1
    return count

stop_words = set(stopwords.words('english'))
folder_path = os.path.abspath('/content/drive/MyDrive/URL_ID')
```

```
with open('results4.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Filename', 'Avg Sentence Length', 'Percentage of Complex Words', 'FOG Index', 'Avg Number of Words per Sentence', 'Complex Word Count', 'Word Count', 'Syllables per Word', 'Personal Pronouns', 'Avg Word Length'])

    files = glob.glob(os.path.join(folder_path, '*.txt'))
    for filename in files:
        print(filename)
        with open(filename, 'r') as f:
            text = f.read()
            sentences = nltk.sent_tokenize(text)
            total_words = len(nltk.word_tokenize(text))
            total_sentences = len(sentences)
            total_syllables = 0
            complex_word_count = 0
            personal_pronouns_ = personal_pronouns(text)
            for word in nltk.word_tokenize(text):
                if word not in stop_words and not word.isnumeric():
                    if len(word) > 2 and syllable_count(word) > 2:
                        complex_word_count += 1
                    total_syllables += syllable_count(word)
            avg_sen_len = total_words / total_sentences
            per_complex_words = complex_word_count / total_words
            fog_index = 0.4 * (avg_sen_len + per_complex_words)
            avg_words_per_sen = total_words / total_sentences
            word_count = len([word for word in nltk.word_tokenize(text) if word not in stop_words and not word.isnumeric() and word.isalpha()])
            syllables_per_word = total_syllables / word_count
            avg_word_len = sum(len(word) for word in nltk.word_tokenize(text) if word not in stop_words and not word.isnumeric() and word.isalpha()) / word_count
            writer.writerow([filename, round(avg_sen_len, 2), round(per_complex_words * 100, 2), round(fog_index, 2), round(avg_words_per_sen, 2), complex_word_count, word_count, round(syllables_per_word, 2), personal_pronouns_, round(avg_word_len,

            print(f"File {filename} completed.")
```

This script is for analyzing text files in a directory and writing the results of various text analysis measures to a CSV file named 'results4.csv'. The measures being calculated are:

- **Average sentence length**
- **Percentage of complex words**
- **FOG Index**
- **Average number of words per sentence**
- **Complex word count**
- **Word count**
- **Syllables per word**
- **Personal pronouns**
- **Average word length**

The script uses several functions to calculate these measures, including functions to tokenize text into sentences and words, count syllables, and identify personal pronouns. It also uses the NLTK library for text processing and the textstat library for calculating the FOG Index.

The script then loops through each text file in the specified directory, reads in the text, calculates the various measures, and writes the results to a CSV file.

Merging the csv files of sentiment analysis and preparing the final csv.

```
[ ]  # Read input.xlsx and extract the columns we want
     input_df = pd.read_excel('/content/drive/MyDrive/Input.xlsx')
     input_df = input_df[['URL_ID', 'URL']]

     # Read Results.csv and extract the columns we want
     results_df = pd.read_csv('/content/part1result.csv')
     results_df = results_df[['URL_ID', 'Positive Score', 'Negative Score', 'Polarity Score', 'Subjectivity Score']]

     # Read results2.csv and extract the columns we want
     results2_df = pd.read_csv('/content/part2result.csv')
     results2_df = results2_df[['URL_ID','Avg Sentence Length', 'Percentage of Complex Words', 'FOG Index', 'Avg Number of Words per Sentence', 'Complex Word Count', 'Word Count', 'Syllables per Word', 'Personal Pronouns', 'Avg Word Length']]

     # Merge the dataframes together on url_id
     merged_df = input_df.merge(results_df, on='URL_ID').merge(results2_df, on='URL_ID')

     # Write the final dataframe to a CSV file
     merged_df.to_csv('finalresult.csv', index=False)
```

Zoomed view of above codeblock below

Merging the csv files of sentiment analysis and preparing the final csv.

```
[ ]  # Read input.xlsx and extract the columns we want
     input_df = pd.read_excel('/content/drive/MyDrive/Input.xlsx')
     input_df = input_df[['URL_ID', 'URL']]

     # Read Results.csv and extract the columns we want
     results_df = pd.read_csv('/content/part1result.csv')
     results_df = results_df[['URL_ID', 'Positive Score', 'Negative Score', 'Polarity Score', 'Subjectivity Score']]

     # Read results2.csv and extract the columns we want
     results2_df = pd.read_csv('/content/part2result.csv')
     results2_df = results2_df[['URL_ID','Avg Sentence Length', 'Percentage of Complex Words', 'FOG Index', 'Avg Number o

     # Merge the dataframes together on url_id
     merged_df = input_df.merge(results_df, on='URL_ID').merge(results2_df, on='URL_ID')

     # Write the final dataframe to a CSV file
     merged_df.to_csv('finalresult.csv', index=False)
```

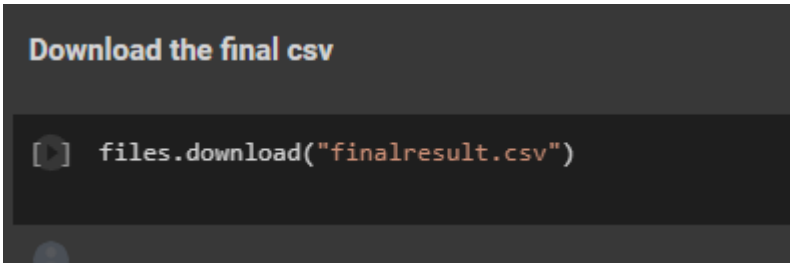
This code reads three CSV files (part1result.csv, part2result.csv, and Input.xlsx) and merges them based on a common column (URL_ID). The resulting merged dataframe is then written to a new CSV file (finalresult.csv).

The code assumes that the CSV files and Excel file are located in a Google Drive folder and mounted to the Google Colab notebook.

The resulting merged dataframe will have columns

URL_ID, URL, Positive Score, Negative Score, Polarity Score, Subjectivity Score, Avg Sentence Length, Percentage of Complex Words, FOG Index, Avg Number of Words per Sentence, Complex Word Count, Word Count, Syllables per Word, Personal Pronouns, and Avg Word Length.

Downloading the final result.csv file.



OUTPUT Screenshot

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	URL_ID	URL	Positive Sc	Negative S	Polarity Sc	Subjectivit	Avg Senter	Percentage	FOG Index	Avg Numb	Complex V	Word Cou	Syllables per V	Personal Prono	Avg Word Length
2	37	https://ins	72	35	0.345794	0.077536	27.59	20.75	11.12	27.59	458	1267	2.44	4	6.98
3	38	https://ins	64	39	0.242718	0.105749	21.74	13.85	8.75	21.74	256	902	2.28	9	6.36
4	39	https://ins	69	39	0.277778	0.08838	23.54	19.35	9.5	23.54	410	1162	2.41	3	6.99
5	40	https://ins	68	28	0.416667	0.09421	20.45	12.35	8.23	20.45	245	1038	2.1	23	6.25
6	41	https://ins	63	24	0.448276	0.074106	26.12	14.75	10.51	26.12	316	1084	2.27	17	6.62
7	42	https://ins	49	24	0.342466	0.083049	26.74	14.35	10.75	26.74	234	810	2.29	31	6.56
8	43	https://ins	31	13	0.409091	0.072607	20.98	16.78	8.46	20.98	176	541	2.33	12	6.59
9	44	https://ins	4	1	0.6	0.031646	37.5	16.44	15.07	37.5	37	135	2.32	1	6.69
10	45	https://ins	42	14	0.5	0.1	24.56	12.21	9.87	24.56	123	530	2.08	1	6.24
11	46	https://ins	71	42	0.256637	0.081004	30.49	15.33	12.26	30.49	402	1326	2.28	10	6.47
12	47	https://ins	50	64	-0.12281	0.080112	25.69	14.17	10.33	25.69	346	1212	2.37	1	6.55
13	48	https://ins	41	23	0.28125	0.073648	23.98	18.66	9.67	23.98	282	836	2.33	11	6.97
14	49	https://ins	35	30	0.076923	0.06871	18.8	15.28	7.58	18.8	247	878	2.23	8	6.3
15	50	https://ins	67	25	0.456522	0.096537	30.35	14.08	12.2	30.35	282	957	2.26	26	6.55
16	51	https://ins	71	23	0.510638	0.085068	23.19	15.02	9.34	23.19	296	1013	2.29	13	6.61
17	52	https://ins	30	1	0.935484	0.064583	28.74	17.14	11.56	28.74	133	436	2.33	1	6.93
18	53	https://ins	84	40	0.354839	0.129707	149.93	13.24	60.02	149.93	278	1029	2.05	18	6.5
19	54	https://ins	24	1	0.92	0.041186	21.18	17.47	8.54	21.18	185	572	2.26	1	6.61
20	55	https://ins	18	7	0.44	0.040783	22.58	21.56	9.12	22.58	219	546	2.56	1	7.17
21	56	https://ins	4	4	0	0.031496	34.58	19.52	13.91	34.58	81	238	2.47	3	6.68
22	57	https://ins	4	1	0.6	0.031646	37.5	16.44	15.07	37.5	37	135	2.32	1	6.69
23	58	https://ins	28	70	-0.42857	0.099898	28.95	15.37	11.64	28.95	267	906	2.29	3	6.67
24	59	https://ins	21	16	0.135135	0.063356	29.85	16.16	12.01	29.85	164	534	2.36	5	6.53
25	60	https://ins	62	39	0.227723	0.085017	20.42	13.03	8.22	20.42	306	1158	2.2	16	6.42
26	61	https://ins	45	4	0.836735	0.088768	19.43	15.76	7.83	19.43	150	514	2.3	5	6.69
27	62	https://ins	9	2	0.636364	0.028796	28.55	14.81	11.48	28.55	93	359	2.15	2	6.34

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
91	126	https://ins	20	26	-0.13043	0.068759	30.63	14.52	12.31	30.63	169	586	2.35	6	6.67
92	127	https://ins	40	85	-0.36	0.127421	27.87	16.24	11.21	27.87	276	877	2.4	8	6.71
93	128	https://ins	12	18	-0.2	0.093168	27.25	17.98	10.97	27.25	98	292	2.34	5	6.65
94	129	https://ins	9	8	0.058824	0.036017	28.58	17.77	11.5	28.58	132	404	2.42	2	6.59
95	130	https://ins	28	64	-0.3913	0.070069	38.61	16.5	15.51	38.61	344	1201	2.28	3	6.48
96	131	https://ins	90	33	0.463415	0.099034	28.16	16	11.33	28.16	338	1116	2.31	1	6.68
97	132	https://ins	31	23	0.148148	0.053678	28.38	15.21	11.41	28.38	272	901	2.42	6	6.41
98	133	https://ins	30	21	0.176471	0.085	30.94	21.69	12.46	30.94	208	544	2.51	1	7.16
99	134	https://ins	11	11	0	0.056701	34.65	16.47	13.92	34.65	97	322	2.45	9	6.84
100	135	https://ins	10	8	0.111111	0.053254	28.19	11.49	11.32	28.19	68	313	2.15	6	6.17
101	136	https://ins	4	17	-0.61905	0.056452	25.58	15.34	10.29	25.58	102	311	2.55	6	6.67
102	137	https://ins	8	8	0	0.05178	37.62	14.52	15.1	37.62	71	253	2.38	5	6.45
103	138	https://ins	9	13	-0.18182	0.073579	26.47	18.09	10.66	26.47	91	268	2.42	9	6.62
104	139	https://ins	12	4	0.5	0.055749	33.67	19.31	13.54	33.67	78	225	2.53	1	6.81
105	140	https://ins	18	52	-0.48571	0.098039	28.77	14.87	11.57	28.77	184	651	2.28	1	6.55
106	141	https://ins	9	22	-0.41935	0.074699	36.8	13.86	14.78	36.8	102	391	2.23	7	6.41
107	142	https://ins	41	72	-0.27434	0.0904	28.01	16.45	11.27	28.01	364	1145	2.4	1	6.75
108	143	https://ins	9	27	-0.5	0.070866	22.46	16.67	9.05	22.46	146	475	2.39	2	6.6
109	144	https://ins	4	1	0.6	0.031646	37.5	16.44	15.07	37.5	37	135	2.32	1	6.69
110	145	https://ins	15	12	0.111111	0.050186	26.24	15.65	10.56	26.24	156	541	2.24	3	6.87
111	146	https://ins	26	28	-0.03704	0.07781	22.49	18.2	9.07	22.49	217	649	2.42	10	6.8
112	147	https://ins	40	13	0.509434	0.06228	26.56	15.97	10.69	26.56	229	767	2.3	3	6.7
113	148	https://ins	30	47	-0.22078	0.088608	20.99	16.54	8.46	20.99	243	800	2.27	3	6.66
114	149	https://ins	33	4	0.783784	0.074899	27.7	21.3	11.17	27.7	177	464	2.44	1	7.2
115	150	https://ins	37	42	-0.06329	0.104084	19.26	16.69	7.77	19.26	225	738	2.28	11	6.61

Important Libraries used for the task

NLTK

- NLTK (Natural Language Toolkit) is a popular Python library for working with human language data. It provides a wide range of tools for tasks such as tokenization, stemming, part-of-speech tagging, parsing, and sentiment analysis, among others.

- NLTK comes with a large collection of corpora and lexicons, including the Penn Treebank, the Brown Corpus, and the WordNet lexicon. These resources can be used for training and testing natural language processing models.

- NLTK is widely used in academia and industry for research and development in fields such as computational linguistics, information retrieval, and machine learning. It is also a popular choice for teaching natural language processing and text mining to students and researchers.
- Here it is used for text processing (stopwords removal for word count) , and tokenization.

textstat

The textstat library is a Python package that provides various text statistics for the given text. It can calculate readability scores, such as the Flesch-Kincaid Grade Level, the Gunning Fog Index, and the Coleman-Liau Index, as well as various other text statistics, such as word count, syllable count, and average sentence length.Here it is used for calculating the FOG index.

os

- The os library in Python provides a way of interacting with the operating system. It provides a range of functions for working with files and directories, such as creating, deleting, renaming, and listing directories and files. It also allows you to manipulate environment variables, get information about the current process, and run external programs.

- Some common functions provided by the os library include os.path.join() for joining two paths together, os.listdir() for listing the contents of a directory, os.mkdir() for creating a new directory, os.rmdir() for deleting an empty directory, os.rename() for renaming a file or directory, os.environ for accessing environment variables, os.getcwd() for getting the current working directory, and os.system() for running a command in the system shell.

- The os library is part of the Python standard library, so it should be available by default on most systems without the need to install any additional packages.

.

pandas

- The pandas library is an open-source data analysis and manipulation tool that is widely used in Python. It provides data structures and functions for efficiently working with structured data, such as spreadsheets and databases.

- The main data structures in pandas are the DataFrame and the Series. A DataFrame is a two-dimensional table of data, where each column can have a different data type. A Series is a one-dimensional labeled array, similar to a column in a spreadsheet.

- Some of the features of pandas include:

    - Reading and writing data in various file formats, such as CSV, Excel, and SQL databases.
    - Cleaning and filtering data by removing missing values, duplicates, and irrelevant data.
    - Manipulating and transforming data using functions such as sorting, grouping, and merging.
    - Aggregating and summarizing data using functions such as mean, sum, and count.
    - Visualizing data using built-in plotting functions.
    - Overall, pandas provides a powerful and flexible toolkit for working with structured data in Python.

BeautifulSoup

- BeautifulSoup is a Python library for web scraping purposes. It is used to extract data from HTML and XML files. With BeautifulSoup, you can parse and extract information from HTML and XML documents, navigate through the structure of an HTML/XML file, and extract the data you need by specifying the attributes of the elements you're interested in.

- BeautifulSoup provides a set of functions to parse HTML/XML documents and create a parse tree that you can navigate to find tags and attributes of interest. It also provides a way to search for tags and their contents based on various criteria such as tag name, attributes, text, etc.

- BeautifulSoup is often used in combination with other Python libraries such as Requests (for HTTP requests) and Pandas (for data analysis) to automate web scraping tasks and extract data from websites for further analysis.
- Here used for parsing the HTML content and extracting the article title and text

google.colab.drive

- google.colab.drive is a Python package that provides a way to interact with Google Drive within the Google Colaboratory environment. Google Colaboratory, or "Colab" for short, is a free cloud-based platform that allows users to write and run Python code using a web browser, with access to Google's powerful computing resources.
- With google.colab.drive, users can mount their Google Drive to Colab, making it easy to read and write files stored in their Drive account. This is especially useful for working with large datasets or for collaborating with others on shared files.
- Here used for mounting the Google Drive to save the output files.

requests

- The requests library is a popular Python library used for making HTTP requests. It provides an easy-to-use API for sending HTTP/1.1 requests using various methods like GET, POST, PUT, DELETE, and more.Here used for web crawling request from url for collecting the title and content.

Folders and directories

- **Input.xlsx**: Contains the url-id and url for web crawling.
- **part1result.csv:** Contains the metrics for sentiment analysis like , 'Positive Score', 'Negative Score', 'Polarity Score', 'Subjectivity Score'
- **part2result.csv:** 'Avg Sentence Length', 'Percentage of Complex Words', 'FOG Index', 'Avg Number of Words per Sentence', 'Complex Word Count', 'Word Count', 'Syllables per Word', 'Personal Pronouns', 'Avg Word Length'
- **finalresult.csv :** consist of the final output file
    - https://drive.google.com/file/d/1Fqgx6w_cWwn2iSBJ63Avf0BNAuqRok7f/view?usp=share_link
- **url_id** : Directory that Contains the .txt files
    - https://drive.google.com/drive/folders/1703iPpDIIcz27wF5GmD5J-sKCRNlOmQX?usp=sharing
- **url_id2**: Directory that Contains the .txt files filtered after stopwords removal.
    - https://drive.google.com/drive/folders/1F2zIAy_cYUvMjhORnl8RGj79fHlHEg-a?usp=share_link