**Documentation: Extract textual data articles from the given URL and perform text analysis to compute variables**

**1. Package Installation**

!pip install pandas openpyxl requests beautifulsoup4  
!pip install pandas textblob nltk openpyxl  
!python -m textblob.download\_corpora

* **Purpose:** Installs all required Python packages:
  + pandas for data manipulation
  + openpyxl for Excel file I/O
  + requests for HTTP requests
  + beautifulsoup4 for HTML parsing
  + textblob and nltk for text processing and NLP tasks

**2. Importing Libraries and Downloading NLTK Data**

import nltk  
nltk.download('punkt')  
nltk.download('averaged\_perceptron\_tagger')  
  
import pandas as pd  
import requests  
from bs4 import BeautifulSoup

* **Purpose:** Imports necessary modules and downloads required NLTK datasets for tokenizing and part-of-speech tagging.

**3. Extracting Text from a URL**

def extract\_text\_from\_url(url):  
 try:  
 response = requests.get(url, timeout=10)  
 response.raise\_for\_status()  
 soup = BeautifulSoup(response.text, 'html.parser')  
 paragraphs = soup.find\_all('p')  
 article\_text = ' '.join([p.get\_text() for p in paragraphs])  
 return article\_text.strip()  
 except Exception as e:  
 return f"Error: {e}"

* **Purpose:** Fetches the content of a web page and extracts the main article text by joining all paragraph (<p>) elements.

**4. Loading URLs and Extracting Text**

input\_file = '/content/sample\_data/Input.xlsx'  
df = pd.read\_excel(input\_file)  
url\_column = df.columns[^1]  
df['Extracted\_Text'] = df[url\_column].apply(extract\_text\_from\_url)  
output\_file = 'output\_with\_articles.xlsx'  
df.to\_excel(output\_file, index=False)  
print(f"Extraction complete. Results saved to {output\_file}")

* **Purpose:**
  + Loads URLs from an Excel file.
  + Extracts article text for each URL.
  + Saves the results (including extracted text) to a new Excel file.

**5. Additional Imports and Setup for Text Analysis**

import nltk  
import re  
from textblob import TextBlob  
from nltk.corpus import stopwords  
  
nltk.download('punkt', quiet=True)  
nltk.download('stopwords', quiet=True)  
nltk.download('averaged\_perceptron\_tagger', quiet=True)  
  
stop\_words = set(stopwords.words('english'))

* **Purpose:** Prepares for advanced text analysis by importing further NLP tools and downloading stopwords.

**6. Load Positive and Negative Word Sets**

def load\_word\_set(filepath):  
 try:  
 with open(filepath, 'r', encoding='utf-8') as f:  
 words = set(line.strip() for line in f if line.strip() and not line.startswith(';'))  
 return words  
 except UnicodeDecodeError:  
 try:  
 with open(filepath, 'r', encoding='latin-1') as f:  
 words = set(line.strip() for line in f if line.strip() and not line.startswith(';'))  
 print(f"Successfully loaded {filepath} with latin-1 encoding.")  
 return words  
 except Exception as e:  
 print(f"Error loading {filepath} with latin-1 encoding: {e}")  
 raise e  
  
positive\_words = load\_word\_set('/content/sample\_data/positive-words.txt')  
negative\_words = load\_word\_set('/content/sample\_data/negative-words.txt')

* **Purpose:** Loads lists of positive and negative words for sentiment analysis, handling different text encodings.

**7. Syllable Counting and Complex Word Detection**

def count\_syllables(word):  
 word = word.lower()  
 word = re.sub(r'[^a-z]', '', word)  
 if len(word) == 0:  
 return 0  
 count = 0  
 vowels = "aeiouy"  
 prev\_char\_was\_vowel = False  
 for i in range(len(word)):  
 if word[i] in vowels:  
 if not prev\_char\_was\_vowel:  
 count += 1  
 prev\_char\_was\_vowel = True  
 else:  
 prev\_char\_was\_vowel = False  
 if word.endswith("e") and not prev\_char\_was\_vowel and len(word) > 1:  
 count = max(1, count - 1)  
 # Additional adjustments for 'es' and 'ed' endings omitted for brevity  
 return max(count, 1)  
  
def is\_complex(word):  
 return count\_syllables(word) > 2

* **Purpose:** Provides functions to count syllables in a word and to determine if a word is "complex" (more than two syllables).

**8. Personal Pronoun Counting**

def count\_personal\_pronouns(text):  
 pronouns = re.findall(r'\b(I|we|my|ours|us)(?<!\bUS)\b', text, re.I)  
 return len(pronouns)

* **Purpose:** Counts the number of personal pronouns in the text, excluding the country "US".

**9. Text Cleaning Function**

def clean\_text(text):  
 text = text.lower()  
 text = re.sub(r'[^\w\s]', '', text)  
 words = text.split()  
 cleaned\_words = [word for word in words if word not in stop\_words]  
 return cleaned\_words

* **Purpose:** Converts text to lowercase, removes punctuation, tokenizes, and removes stopwords.

**10. Main Text Metrics Function**

def text\_metrics(text):  
 if not isinstance(text, str):  
 return pd.Series([^0] \* 13, index=[...])  
 blob = TextBlob(text)  
 sentences = blob.sentences  
 sentence\_count = len(sentences) or 1  
 cleaned\_words = clean\_text(text)  
 total\_cleaned\_word\_count = len(cleaned\_words)  
 all\_words = blob.words  
 pos\_score = sum(1 for w in cleaned\_words if w in positive\_words)  
 neg\_score = sum(1 for w in cleaned\_words if w in negative\_words)  
 polarity = (pos\_score - neg\_score) / ((pos\_score + neg\_score )+ 0.000001)  
 subjectivity = (pos\_score + neg\_score) / ((total\_cleaned\_word\_count) + 0.000001)  
 total\_words\_for\_avg\_sentence\_length = len(all\_words)  
 avg\_sent\_len = total\_words\_for\_avg\_sentence\_length / sentence\_count if sentence\_count else 0  
 complex\_words = [w for w in all\_words if is\_complex(w)]  
 complex\_word\_count = len(complex\_words)  
 percent\_complex = (complex\_word\_count / total\_words\_for\_avg\_sentence\_length) \* 100 if total\_words\_for\_avg\_sentence\_length else 0  
 fog\_index = 0.4 \* (avg\_sent\_len + percent\_complex)  
 avg\_words\_per\_sentence = avg\_sent\_len  
 syllable\_per\_word = sum(count\_syllables(w) for w in all\_words) / total\_words\_for\_avg\_sentence\_length if total\_words\_for\_avg\_sentence\_length else 0  
 personal\_pronouns = count\_personal\_pronouns(text)  
 avg\_word\_len = sum(len(w) for w in all\_words) / total\_words\_for\_avg\_sentence\_length if total\_words\_for\_avg\_sentence\_length else 0  
 return pd.Series([  
 pos\_score, neg\_score, polarity, subjectivity, avg\_sent\_len, percent\_complex,  
 fog\_index, avg\_words\_per\_sentence, complex\_word\_count, total\_cleaned\_word\_count,  
 syllable\_per\_word, personal\_pronouns, avg\_word\_len  
 ])

* **Purpose:** Calculates various text metrics, including sentiment, readability, and linguistic features.

**11. Applying Metrics and Saving Results**

input\_file = '/content/sample\_data/output\_with\_articles.xlsx'  
df = pd.read\_excel(input\_file)  
text\_column\_index = 2  
df.iloc[:, text\_column\_index] = df.iloc[:, text\_column\_index].astype(str)  
metrics = df.iloc[:, text\_column\_index].apply(text\_metrics)  
metrics.columns = [  
 "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",  
 "SYLLABLE PER WORD", "PERSONAL PRONOUNS", "AVG WORD LENGTH"  
]  
result = pd.concat([df, metrics], axis=1)  
output\_file = 'output\_with\_metrics.xlsx'  
result.to\_excel(output\_file, index=False)  
print(f"Analysis complete. Results saved to {output\_file}")

* **Purpose:**
  + Loads the Excel file with extracted articles.
  + Applies the text metrics function to each article.
  + Concatenates the results and saves them to a new Excel file.