Configuration Manual for Sumbot

Anish Girish
MSc Data Analytics
X22208879@student.ncirl.ie

1 System Requirements

This whole project takes into the account three important steps,

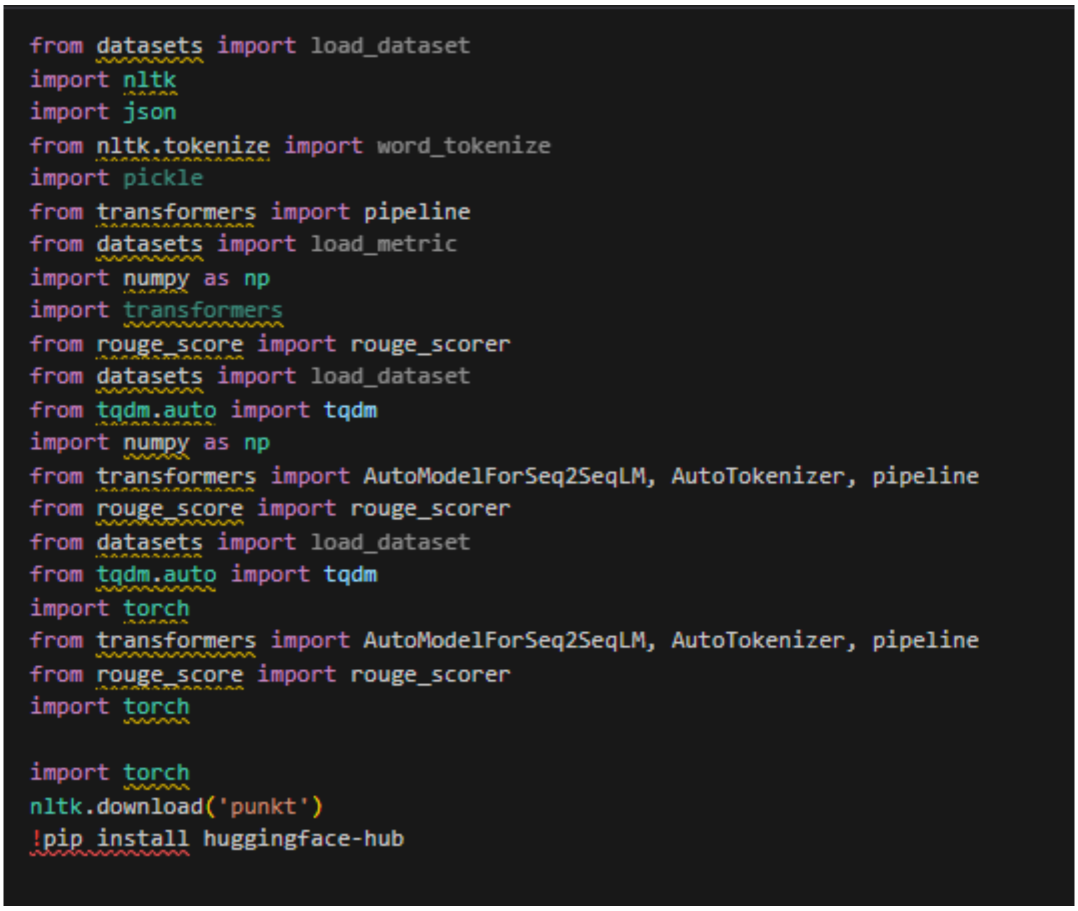
RAM: 8GB DDR3

OS: Windows 11 pro

Processor: i5 9th generation

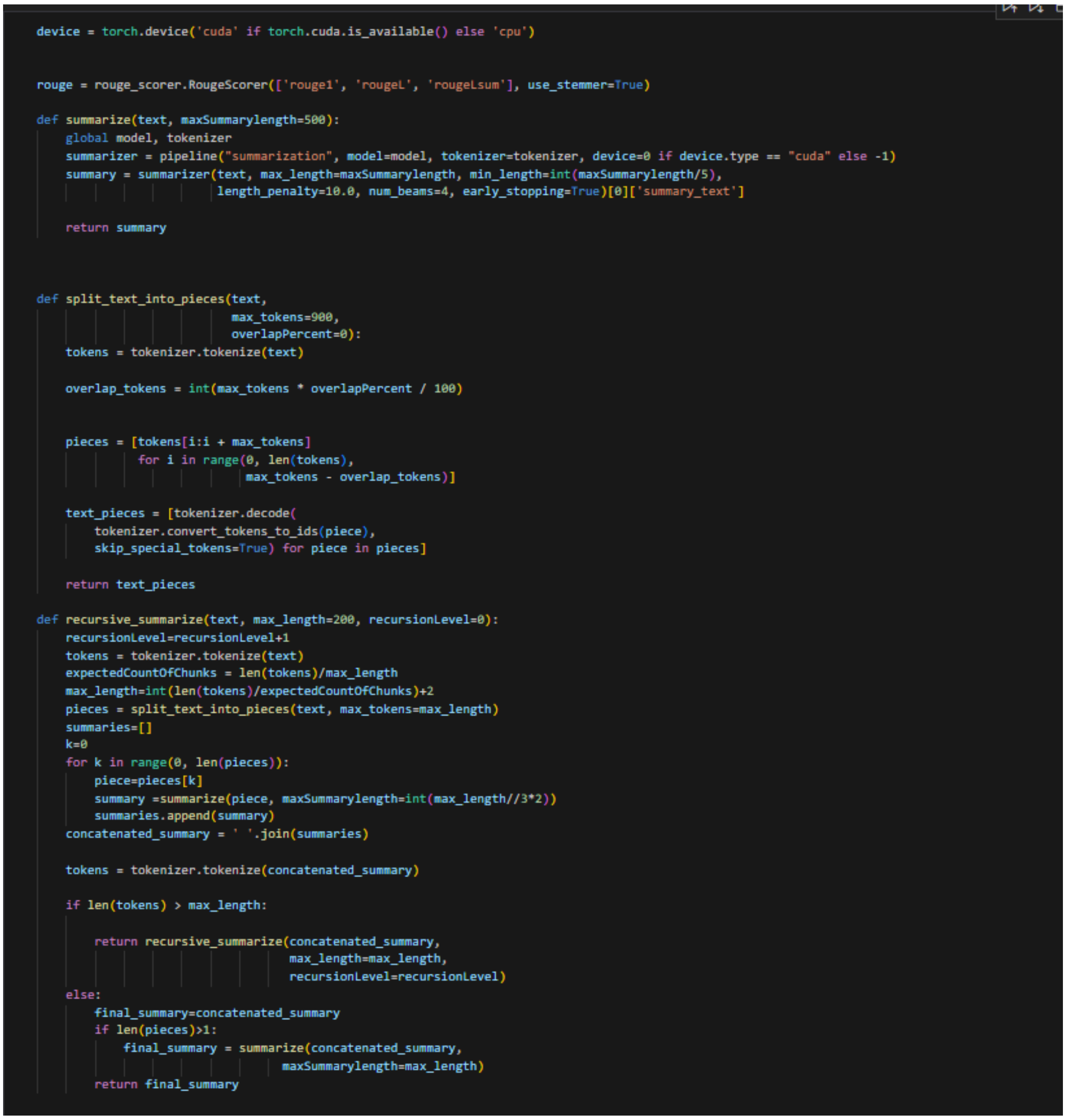
Technology Required: Python, Anaconda, Spyder, Streamlit

2 Code execution



```
from datasets import load_dataset
import nltk
import json
from nltk.tokenize import word_tokenize
import pickle
from transformers import pipeline
from datasets import load_metric
import numpy as np
import transformers
from rouge_score import rouge_scorer
from datasets import load_dataset
from tqdm.auto import tqdm
import numpy as np
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer, pipeline
from rouge_score import rouge_scorer
from datasets import load_dataset
from tqdm.auto import tqdm
import torch
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer, pipeline
from rouge_score import rouge_scorer
import torch

import torch
nltk.download('punkt')
!pip install huggingface-hub
```

Figure1. The code imports libraries and modules for text processing, machine learning, and evaluation. It sets up tools for working with datasets, NLP models, tokenization, and metrics, and installs necessary packages.



```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


rouge = rouge_scorer.RougeScorer(['rouge1', 'rougeL', 'rougeLsum'], use_stemmer=True)

def summarize(text, maxSummarylength=500):
    global model, tokenizer
    summarizer = pipeline("summarization", model=model, tokenizer=tokenizer, device=0 if device.type == "cuda" else -1)
    summary = summarizer(text, max_length=maxSummarylength, min_length=int(maxSummarylength/5),
                         length_penalty=10.0, num_beams=4, early_stopping=True)[0]['summary_text']

    return summary



def split_text_into_pieces(text,
                           max_tokens=900,
                           overlapPercent=0):
    tokens = tokenizer.tokenize(text)

    overlap_tokens = int(max_tokens * overlapPercent / 100)


    pieces = [tokens[i:i + max_tokens]
              for i in range(0, len(tokens),
                             max_tokens - overlap_tokens)]

    text_pieces = [tokenizer.decode(
        tokenizer.convert_tokens_to_ids(piece),
        skip_special_tokens=True) for piece in pieces]

    return text_pieces

def recursive_summarize(text, max_length=200, recursionLevel=0):
    recursionLevel=recursionLevel+1
    tokens = tokenizer.tokenize(text)
    expectedCountOfChunks = len(tokens)/max_length
    max_length=int(len(tokens)/expectedCountOfChunks)+2
    pieces = split_text_into_pieces(text, max_tokens=max_length)
    summaries=[]
    k=0
    for k in range(0, len(pieces)):
        piece=pieces[k]
        summary =summarize(piece, maxSummarylength=int(max_length//3*2))
        summaries.append(summary)
    concatenated_summary = ' '.join(summaries)

    tokens = tokenizer.tokenize(concatenated_summary)

    if len(tokens) > max_length:

        return recursive_summarize(concatenated_summary,
                                   max_length=max_length,
                                   recursionLevel=recursionLevel)
    else:
        final_summary=concatenated_summary
        if len(pieces)>1:
            final_summary = summarize(concatenated_summary,
                                      maxSummarylength=max_length)
        return final_summary
```

Figure2. This code defines functions for summarizing text using a sequence-to-sequence model:

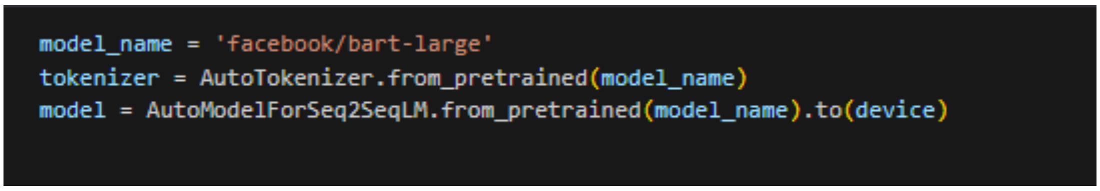
1. `device`: Chooses between GPU or CPU based on availability.

2. `rouge`: Initializes ROUGE scoring for evaluating summaries.

3. `summarize(text, maxSummarylength)`: Summarizes input text using a pre-trained model and tokenizer.

4. `split_text_into_pieces(text, max_tokens, overlapPercent)`: Splits long text into manageable pieces for summarization.

5. `recursive_summarize(text, max_length, recursionLevel)`: Recursively summarizes text by splitting and summarizing iteratively until the summary is concise.



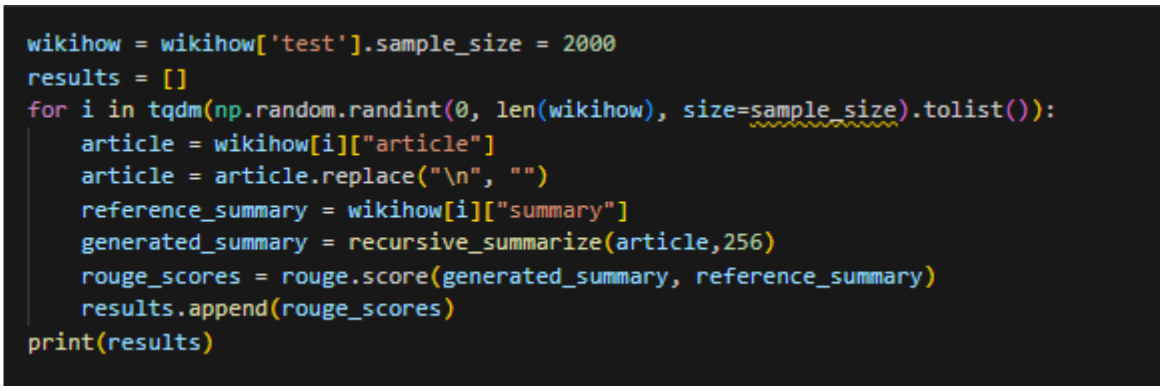
```python
model_name = 'facebook/bart-large'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name).to(device)
```

Figure3. This code loads the BART model and tokenizer:

1. `model_name`: Specifies the pre-trained model, BART.

2. `tokenizer`: Loads the tokenizer for the BART model.

3. `model`: Loads the BART model and moves it to the appropriate device (GPU or CPU).



```python
wikihow = wikihow['test'].sample_size = 2000
results = []
for i in tqdm(np.random.randint(0, len(wikihow), size=sample_size).tolist()):
    article = wikihow[i]["article"]
    article = article.replace("\n", "")
    reference_summary = wikihow[i]["summary"]
    generated_summary = recursive_summarize(article,256)
    rouge_scores = rouge.score(generated_summary, reference_summary)
    results.append(rouge_scores)
print(results)
```

Figure4. This code evaluates the summarization model on a sample of WikiHow articles:

1. `wikihow`: Samples 2,000 articles from the WikiHow dataset.

2. `results`: Initializes an empty list to store ROUGE scores.

3. `for` loop: Iterates over random indices to process each article.

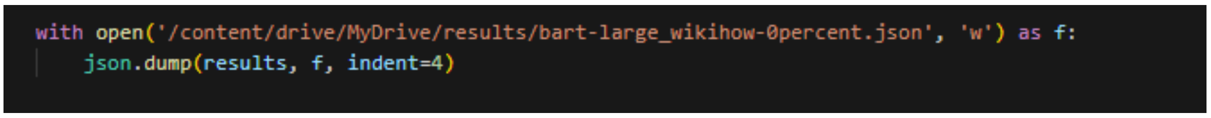
- `article`: Cleans the article text by removing newlines.

- `reference_summary`: Retrieves the reference summary.

- `generated_summary`: Summarizes the article using `recursive_summarize`.

- `rouge_scores`: Computes ROUGE scores for the generated summary vs. reference summary.

4. `print(results)`: Outputs the list of ROUGE scores for all sampled articles.



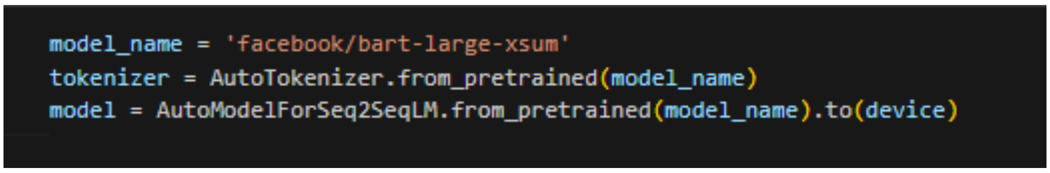
```python
with open('/content/drive/MyDrive/results/bart-large_wikihow-0percent.json', 'w') as f:
    json.dump(results, f, indent=4)
```

Figure5. This code saves the evaluation results to a JSON file:

1. `open`: Opens (or creates) a file at the specified path in write mode.

2. `json.dump`: Writes the `results` list (which contains ROUGE scores) to the file in JSON format with indentation for readability.

3. `f`: The file object used for writing.



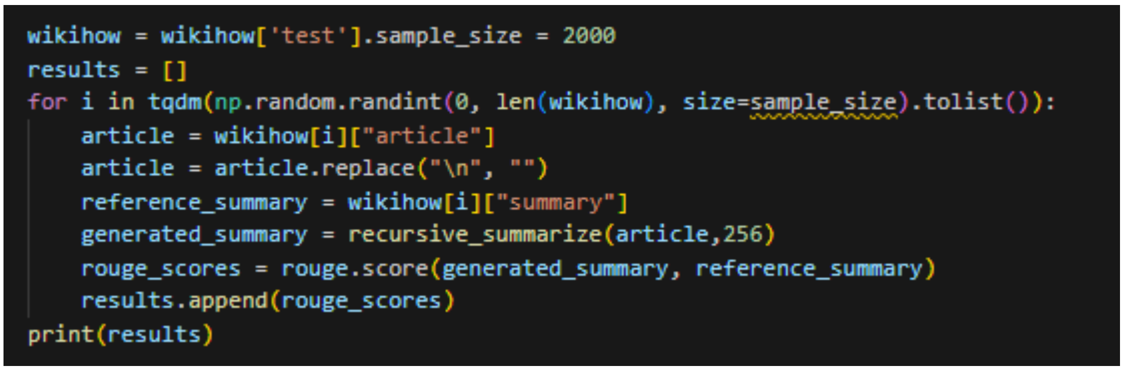
```
model_name = 'facebook/bart-large-xsum'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name).to(device)
```

Figure6. This code initializes a model and tokenizer for text summarization:

1. `model_name`: Specifies the model identifier for the `facebook/bart-large-xsum` variant, which is optimized for summarization tasks.

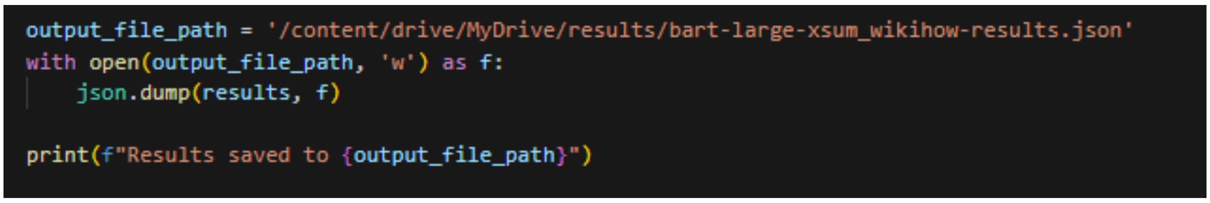
2. `AutoTokenizer.from_pretrained(model_name)`: Loads the tokenizer associated with the specified model.

3. `AutoModelForSeq2SeqLM.from_pretrained(model_name).to(device)`: Loads the pre-trained model and moves it to the specified device (GPU or CPU).



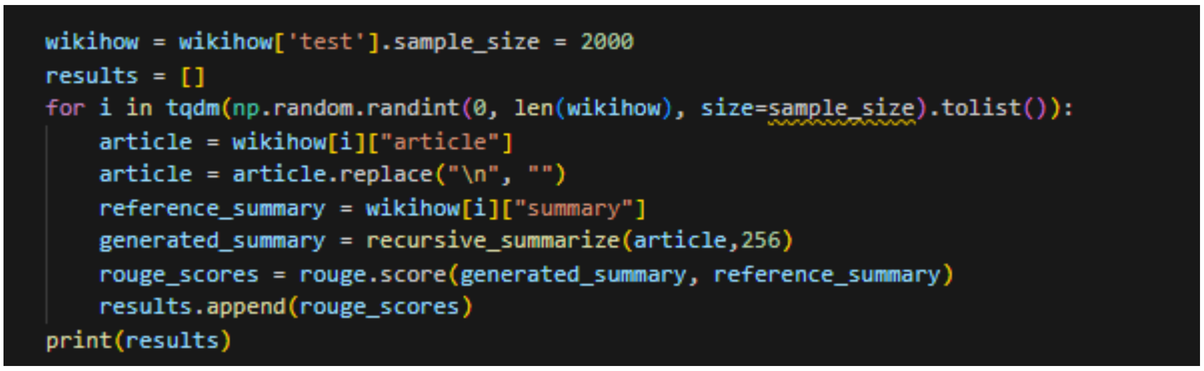
```
wikihow = wikihow['test'].sample_size = 2000
results = []
for i in tqdm(np.random.randint(0, len(wikihow), size=sample_size).tolist()):
    article = wikihow[i]["article"]
    article = article.replace("\n", "")
    reference_summary = wikihow[i]["summary"]
    generated_summary = recursive_summarize(article,256)
    rouge_scores = rouge.score(generated_summary, reference_summary)
    results.append(rouge_scores)
print(results)
```

Figure7. This code evaluates a summarization model by generating summaries for 2000 random articles from the WikiHow dataset and calculates their ROUGE scores to assess the summarization quality.



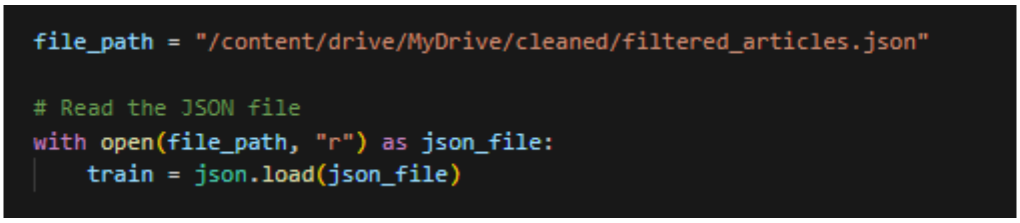
```
output_file_path = '/content/drive/MyDrive/results/bart-large-xsum_wikihow-results.json'
with open(output_file_path, 'w') as f:
    json.dump(results, f)

print(f"Results saved to {output_file_path}")
```

Figure8. This code saves the summarization results to a JSON file and prints the file path where the results are saved.



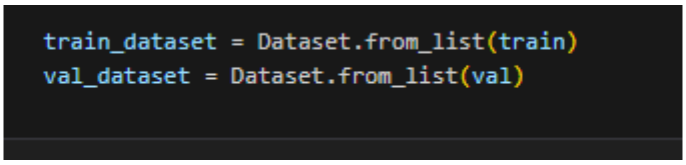
```
wikihow = wikihow['test'].sample_size = 2000
results = []
for i in tqdm(np.random.randint(0, len(wikihow), size=sample_size).tolist()):
    article = wikihow[i]["article"]
    article = article.replace("\n", "")
    reference_summary = wikihow[i]["summary"]
    generated_summary = recursive_summarize(article,256)
    rouge_scores = rouge.score(generated_summary, reference_summary)
    results.append(rouge_scores)
print(results)
```

Figure9. This code samples 2,000 articles from the WikiHow dataset, generates summaries using the `recursive_summarize` function, evaluates them with ROUGE scores against reference summaries, and prints the results.



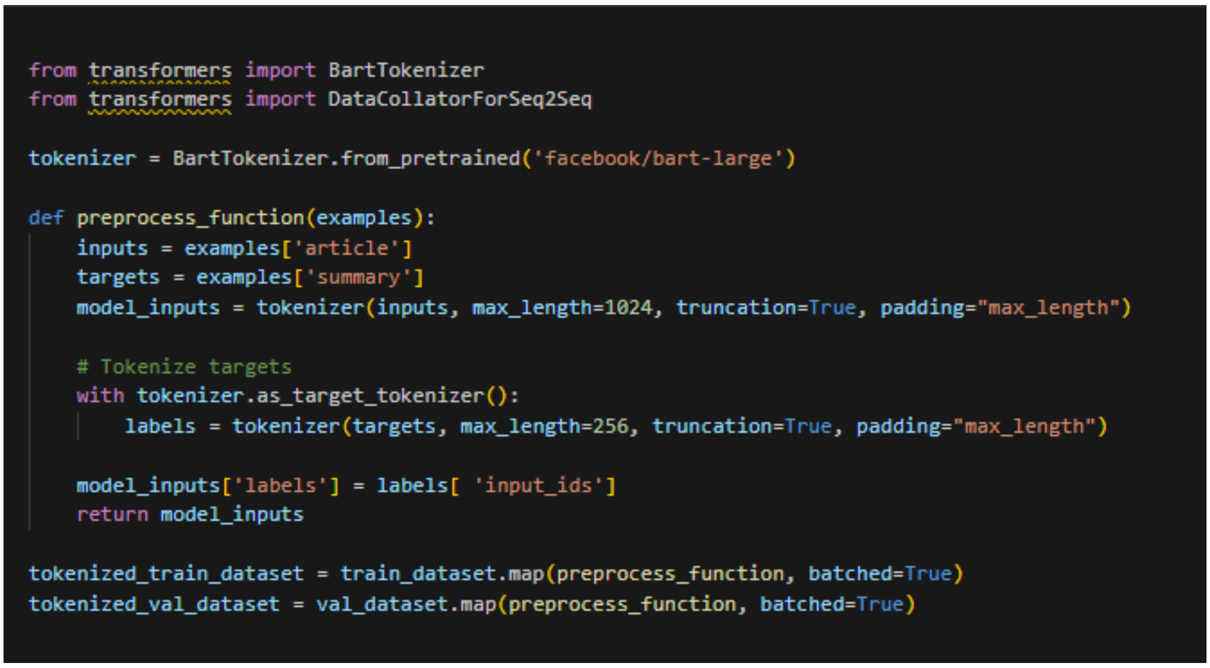
```
file_path = "/content/drive/MyDrive/cleaned/filtered_articles.json"

# Read the JSON file
with open(file_path, "r") as json_file:
    train = json.load(json_file)
```

Figure10. This code reads a JSON file containing cleaned and filtered articles and loads it into a variable called `train`.



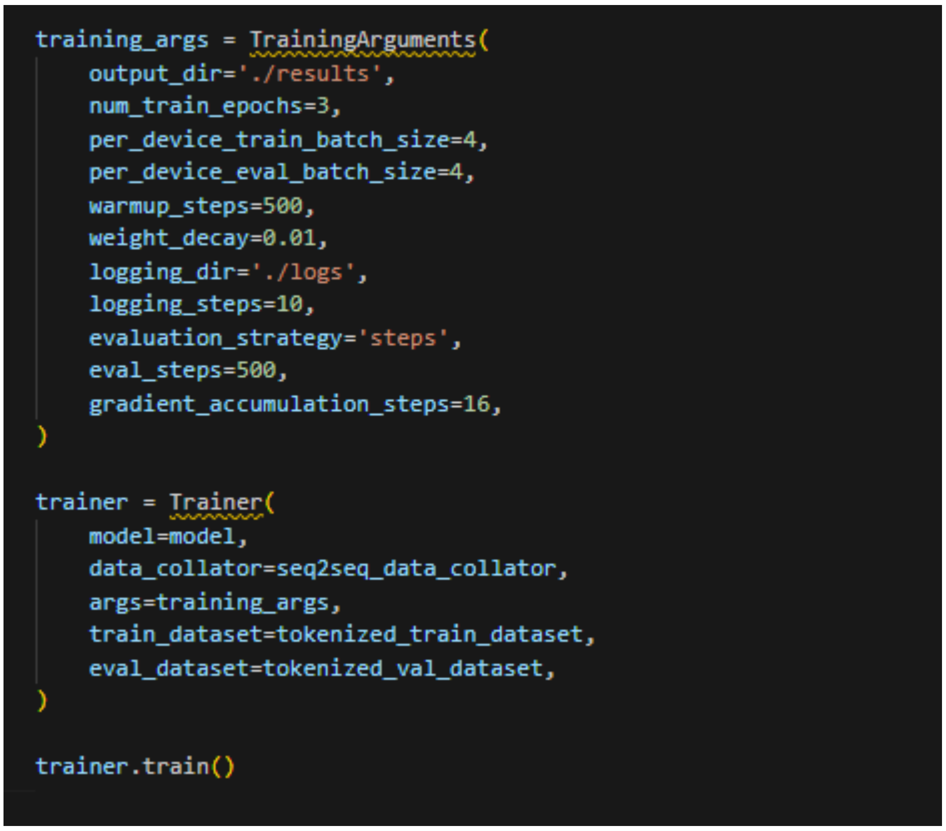
```
train_dataset = Dataset.from_list(train)
val_dataset = Dataset.from_list(val)
```

Figure11. This code converts the lists `train` and `val` into `Dataset` objects from the `datasets` library, creating `train_dataset` and `val_dataset`.



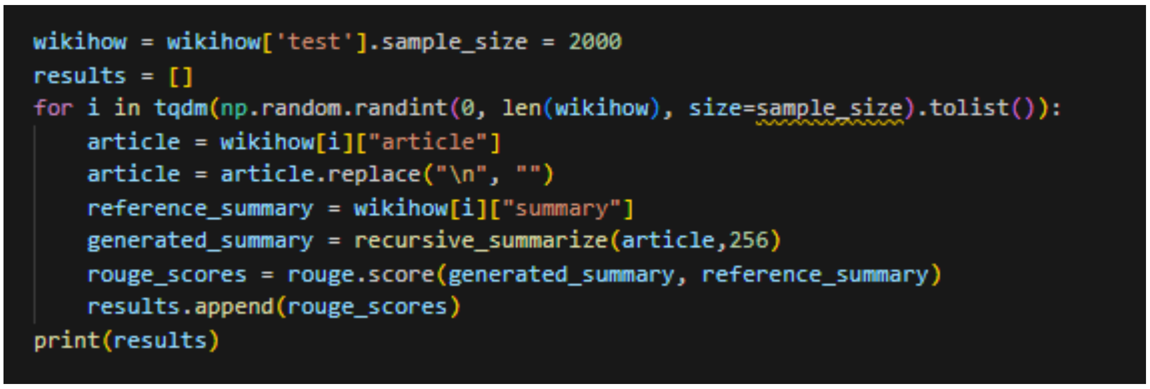
```
from transformers import BartTokenizer
from transformers import DataCollatorForSeq2Seq

tokenizer = BartTokenizer.from_pretrained('facebook/bart-large')

def preprocess_function(examples):
    inputs = examples['article']
    targets = examples['summary']
    model_inputs = tokenizer(inputs, max_length=1024, truncation=True, padding="max_length")

    # Tokenize targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(targets, max_length=256, truncation=True, padding="max_length")

    model_inputs['labels'] = labels[ 'input_ids']
    return model_inputs

tokenized_train_dataset = train_dataset.map(preprocess_function, batched=True)
tokenized_val_dataset = val_dataset.map(preprocess_function, batched=True)
```

Figure12. This code tokenizes the `article` and `summary` fields of the datasets using the `BartTokenizer`. It processes the `train_dataset` and `val_dataset` by truncating and padding them, then prepares them for model input by mapping the tokenized data.



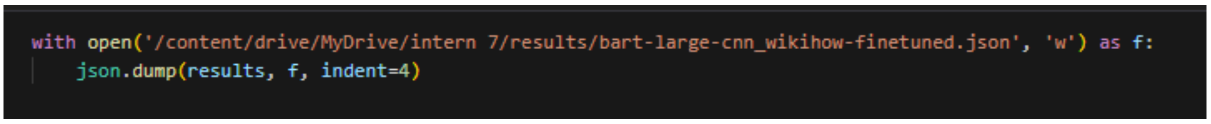
```python
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy='steps',
    eval_steps=500,
    gradient_accumulation_steps=16,
)

trainer = Trainer(
    model=model,
    data_collator=seq2seq_data_collator,
    args=training_args,
    train_dataset=tokenized_train_dataset,
    eval_dataset=tokenized_val_dataset,
)

trainer.train()
```

Figure13. This code sets up and runs a training loop for a sequence-to-sequence model. It defines training parameters with `TrainingArguments`, initializes a `Trainer` with these arguments, and trains the model using the provided `train_dataset` and `eval_dataset`.



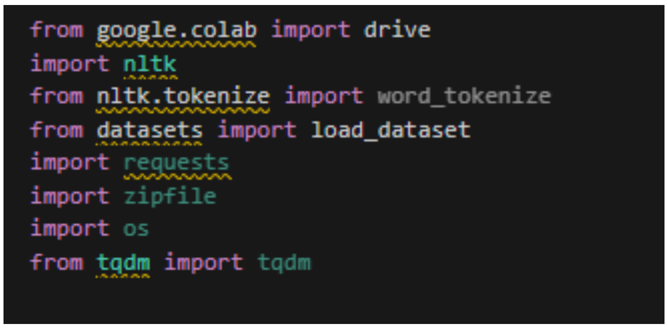
```python
wikihow = wikihow['test'].sample_size = 2000
results = []
for i in tqdm(np.random.randint(0, len(wikihow), size=sample_size).tolist()):
    article = wikihow[i]["article"]
    article = article.replace("\n", "")
    reference_summary = wikihow[i]["summary"]
    generated_summary = recursive_summarize(article,256)
    rouge_scores = rouge.score(generated_summary, reference_summary)
    results.append(rouge_scores)
print(results)
```

Figure14. This code samples 2,000 articles from the WikiHow dataset, generates summaries using the `recursive_summarize` function, and evaluates them with ROUGE scores. The results are collected and printed.



```python
with open('/content/drive/MyDrive/intern 7/results/bart-large-cnn_wikihow-finetuned.json', 'w') as f:
    json.dump(results, f, indent=4)
```

Figure15. This code saves the evaluation results (ROUGE scores) to a JSON file at the specified path. The results are stored with indentation for readability.



```
from google.colab import drive
import nltk
from nltk.tokenize import word_tokenize
from datasets import load_dataset
import requests
import zipfile
import os
from tqdm import tqdm
```

Figure16. This code does the following:

1. Mounts Google Drive: Makes the Google Drive accessible in Colab.

2. Imports NLTK: For natural language processing tasks, like tokenizing text.

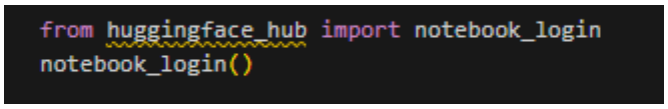
3. Loads a Dataset: Uses the `datasets` library to load a dataset.

4. Requests Library: For making HTTP requests.

5. Zipfile Library: For handling ZIP files.

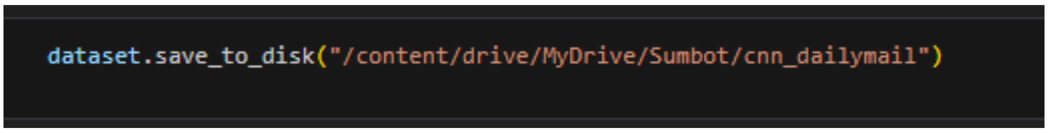
6. OS Library: For interacting with the operating system.

7. TQDM: For displaying progress bars.



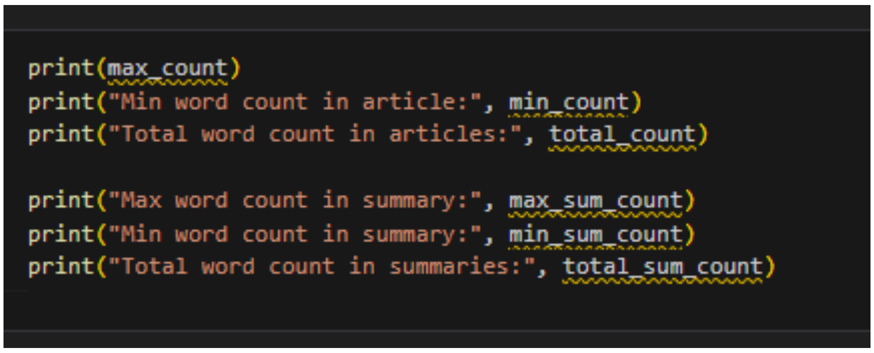
```
from huggingface_hub import notebook_login
notebook_login()
```

Figure17. This code opens a login prompt for Hugging Face's Hub directly in the notebook, allowing you to authenticate and access private datasets or models.



```
dataset.save_to_disk("/content/drive/MyDrive/Sumbot/cnn_dailymail")
```

Figure18. This line of code saves the `dataset` object to the specified directory on Google Drive, allowing you to persist the dataset for later use or sharing.



```
print(max_count)
print("Min word count in article:", min_count)
print("Total word count in articles:", total_count)

print("Max word count in summary:", max_sum_count)
print("Min word count in summary:", min_sum_count)
print("Total word count in summaries:", total_sum_count)
```

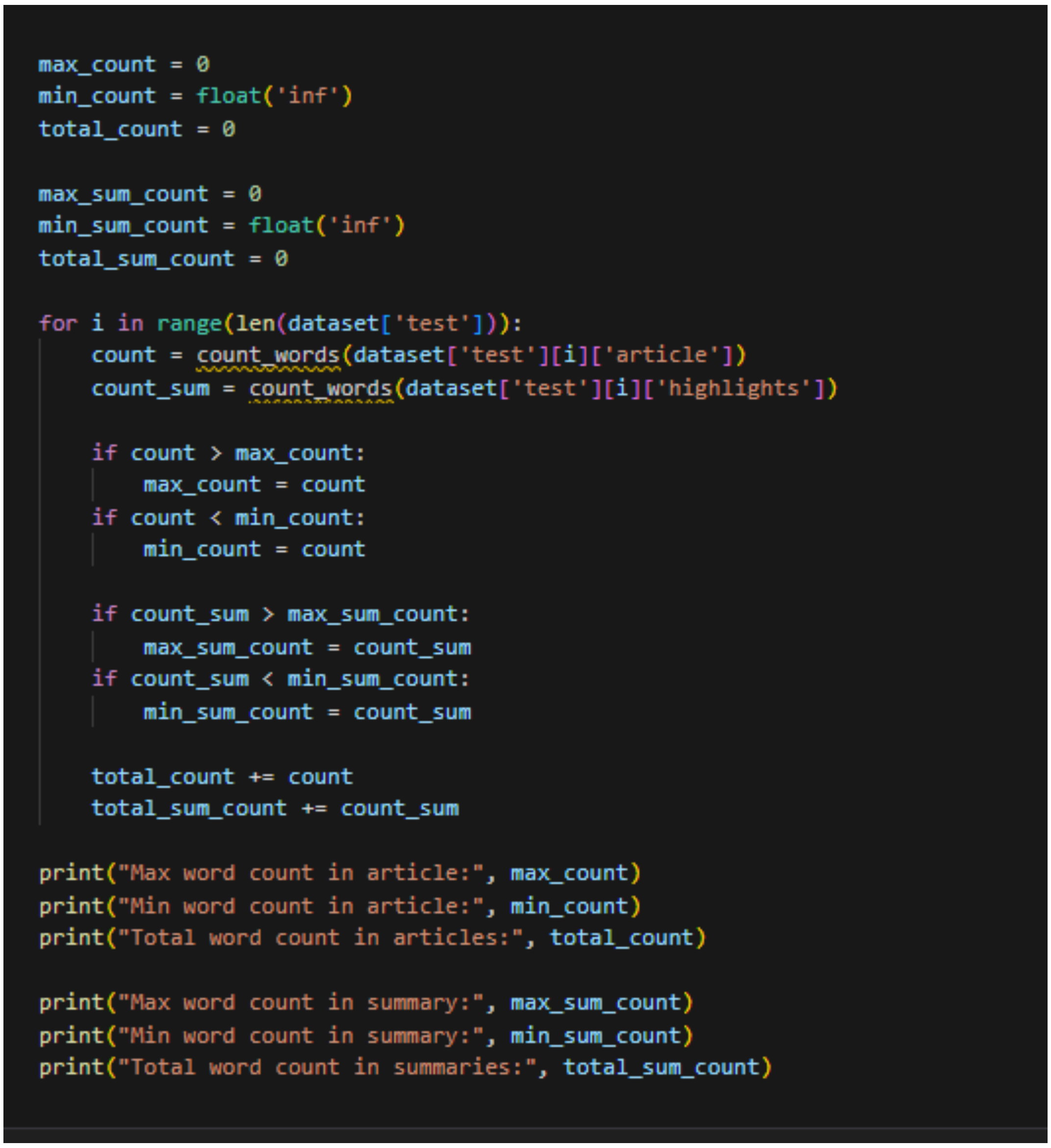
Figure19. These print statements display:

- The maximum and minimum word count of articles (`max_count` and `min_count`).

- The total word count of all articles (`total_count`).

- The maximum and minimum word count of summaries (`max_sum_count` and `min_sum_count`).

- The total word count of all summaries (`total_sum_count`).



```python
max_count = 0
min_count = float('inf')
total_count = 0

max_sum_count = 0
min_sum_count = float('inf')
total_sum_count = 0

for i in range(len(dataset['test'])):
    count = count_words(dataset['test'][i]['article'])
    count_sum = count_words(dataset['test'][i]['highlights'])

    if count > max_count:
        max_count = count
    if count < min_count:
        min_count = count

    if count_sum > max_sum_count:
        max_sum_count = count_sum
    if count_sum < min_sum_count:
        min_sum_count = count_sum

    total_count += count
    total_sum_count += count_sum

print("Max word count in article:", max_count)
print("Min word count in article:", min_count)
print("Total word count in articles:", total_count)

print("Max word count in summary:", max_sum_count)
print("Min word count in summary:", min_sum_count)
print("Total word count in summaries:", total_sum_count)
```

Figure20. This code calculates and prints:

- Article Word Counts:

  - `max_count`: Maximum word count in an article.

  - `min_count`: Minimum word count in an article.

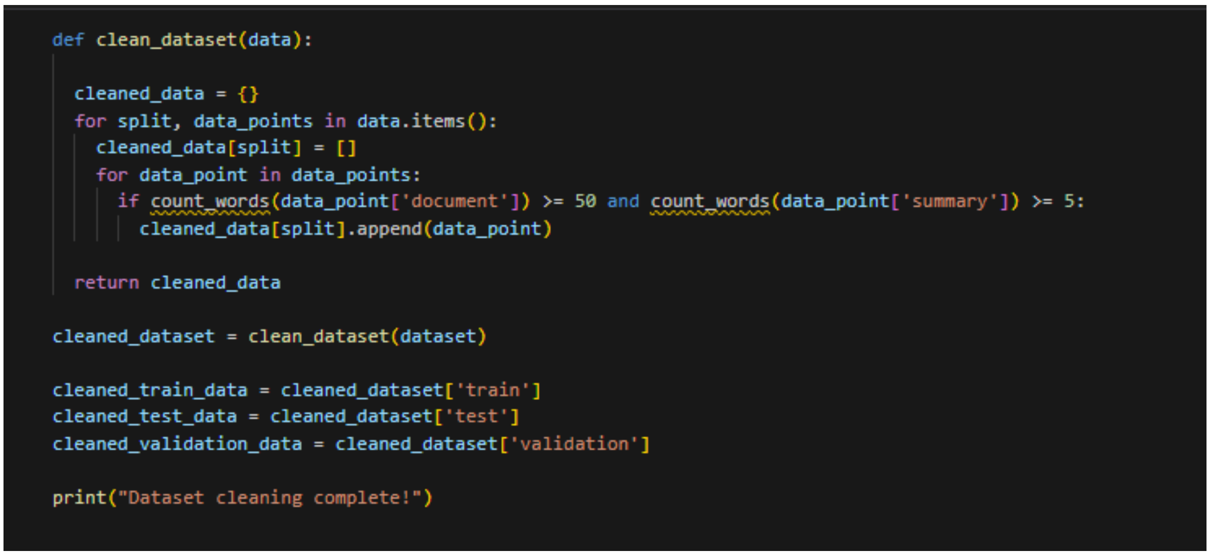
  - `total_count`: Total word count across all articles.

- Summary Word Counts:

  - `max_sum_count`: Maximum word count in a summary.

  - `min_sum_count`: Minimum word count in a summary.

  - `total_sum_count`: Total word count across all summaries.



```
def clean_dataset(data):

    cleaned_data = {}
    for split, data_points in data.items():
        cleaned_data[split] = []
        for data_point in data_points:
            if count_words(data_point['document']) >= 50 and count_words(data_point['summary']) >= 5:
                cleaned_data[split].append(data_point)

    return cleaned_data

cleaned_dataset = clean_dataset(dataset)

cleaned_train_data = cleaned_dataset['train']
cleaned_test_data = cleaned_dataset['test']
cleaned_validation_data = cleaned_dataset['validation']

print("Dataset cleaning complete!")
```

Figure21. This function `clean_dataset`:

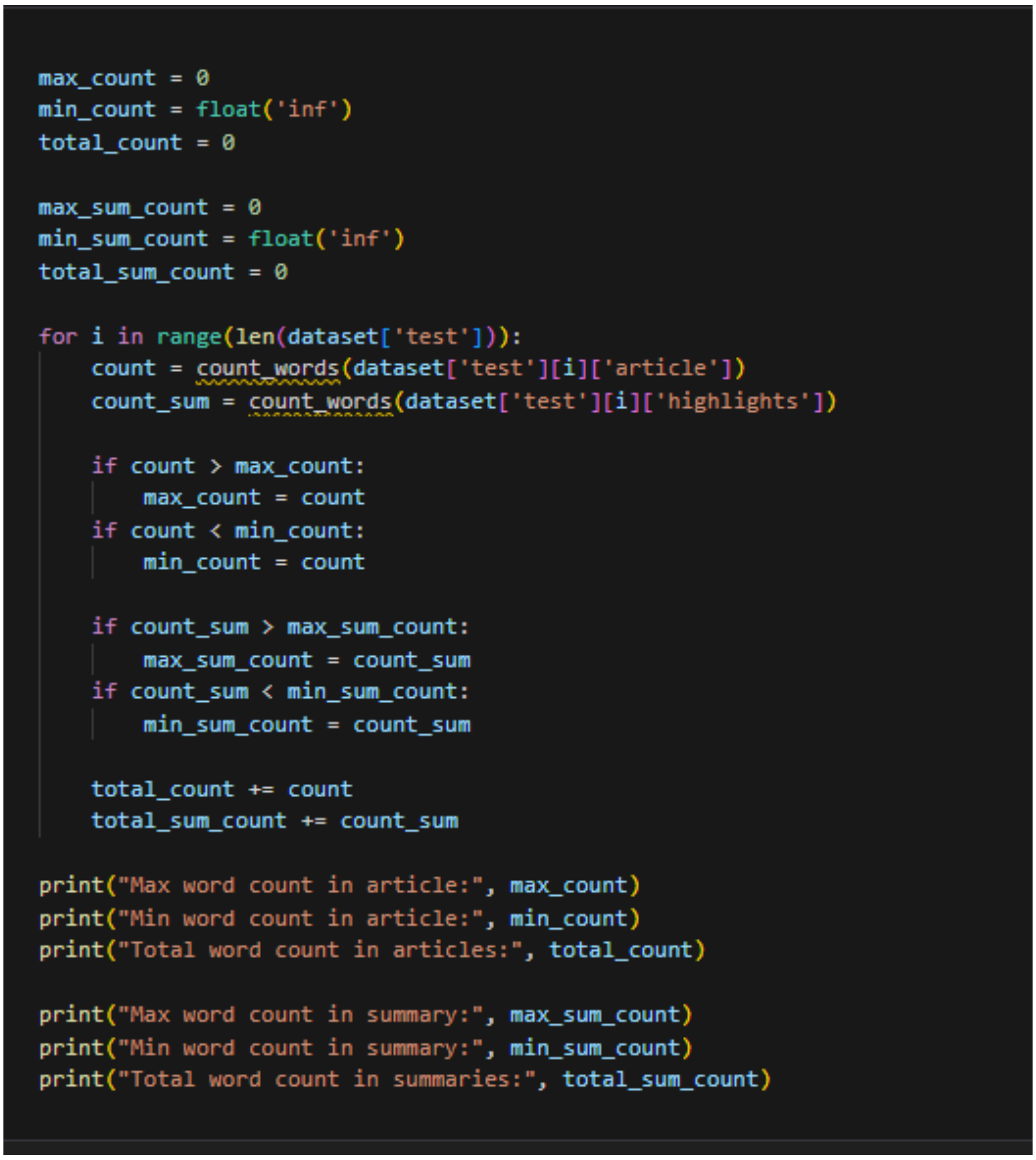
- Filters out data points from the dataset based on word count.

- Keeps only those data points where:

  - The document has at least 50 words.

  - The summary has at least 5 words.

- Returns the cleaned dataset split into train, test, and validation sets.

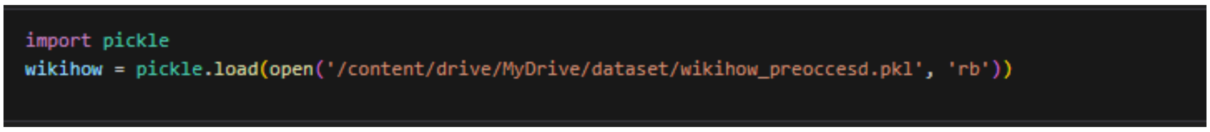


```
max_count = 0
min_count = float('inf')
total_count = 0

max_sum_count = 0
min_sum_count = float('inf')
total_sum_count = 0

for i in range(len(dataset['test'])):
    count = count_words(dataset['test'][i]['article'])
    count_sum = count_words(dataset['test'][i]['highlights'])

    if count > max_count:
        max_count = count
    if count < min_count:
        min_count = count

    if count_sum > max_sum_count:
        max_sum_count = count_sum
    if count_sum < min_sum_count:
        min_sum_count = count_sum

    total_count += count
    total_sum_count += count_sum

print("Max word count in article:", max_count)
print("Min word count in article:", min_count)
print("Total word count in articles:", total_count)

print("Max word count in summary:", max_sum_count)
print("Min word count in summary:", min_sum_count)
print("Total word count in summaries:", total_sum_count)
```

Figure22. This code:

- Computes and prints statistics for word counts in the 'test' split of the dataset.

- Calculates the maximum, minimum, and total word counts for both articles and summaries.



```
import pickle
wikihow = pickle.load(open('/content/drive/MyDrive/dataset/wikihow_preoccesd.pkl', 'rb'))
```

Figure23. This code:

- Loads a pickled `wikihow` dataset from a file located at `'/content/drive/MyDrive/dataset/wikihow_preoccesd.pkl'` into memory using Python's `pickle` module.

3 Steps to Run and Execute the codes

Step 1: Login to the google drive which has the codes and data saved

Step 2: Execute the colab file

Step 3: Authenticate the access for the drive