```
text = """In conclusion, the provided code clusters countries based on the ratio of affected to recovered COVID-19 cases using the
```

```
len(text)
```

```
611
```

**This code loads a pre-trained English language processing model called "en_core_web_sm" from the spaCy library.**

```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
```

```
nlp = spacy.load('en_core_web_sm')
```

```
doc = nlp(text)
```

```
tokens
```

```
['conclusion',
 'provided',
 'code',
 'clusters',
 'countries',
 'based',
 'ratio',
 'affected',
 'recovered',
 'covid-19',
 'cases',
 'k',
 'means',
 'algorithm',
 'partitions',
 'data',
 'clusters',
 'represented',
 'different',
 'colors',
 'choice',
 'k',
 'means',
 'justified',
 'simplicity',
 'efficiency',
 'requires',
 'specifying',
 'number',
 'clusters',
 'advance',
 'dataset',
 'includes',
 'attributes',
 'country',
 'names',
 'total',
 'reported',
 'cases',
 'total',
 'recovered',
 'cases',
 'unnecessary',
 'columns',
 'excluded',
 'analysis',
 'approach',
 'provides',
 'insights',
 'countries',
 'compare',
 'terms',
 'covid-19',
 'recovery',
 'rates']
```

**This code filters tokens from a spaCy document, keeping only nouns, verbs, adjectives, proper nouns and excluding stopwords and punctuation.**

```
tokens1=[]
stopwords = list(STOP_WORDS)
allowed_pos =['ADJ','PROPN','VERB','NOUN']
for token in doc:
  if token.text in stopwords or token.text in punctuation:
    continue
  if token.pos_ in allowed_pos:
    tokens1.append(token.text)
```

```
tokens1
```

```
['conclusion',
 'provided',
 'code',
 'clusters',
 'countries',
 'based',
 'ratio',
 'affected',
 'recovered',
 'COVID-19',
 'cases',
 'K',
 'Means',
 'algorithm',
 'partitions',
 'data',
 'clusters',
 'represented',
 'different',
 'colors',
 'choice',
 'K',
 'Means',
 'justified',
 'simplicity',
 'efficiency',
 'requires',
 'specifying',
 'number',
 'clusters',
 'advance',
 'dataset',
 'includes',
 'attributes',
 'country',
 'names',
 'total',
 'reported',
 'cases',
 'total',
 'recovered',
 'cases',
 'unnecessary',
 'columns',
 'excluded',
 'analysis',
 'approach',
 'provides',
 'insights',
 'countries',
 'compare',
 'terms',
 'COVID-19',
 'recovery',
 'rates']
```

```
from collections import Counter                        #Counter class is used to create a collection that specifically counts th
```

```
word_freq = Counter(tokens)
```

```
max_freq = max (word_freq.values())
```

```
max_freq
```

```
3
```

Double-click (or enter) to edit

**This code normalizes the word frequencies in a dictionary by dividing each word's count by the maximum count. In other words, it converts the raw counts into proportions of the most frequent word.**

```
for word in word_freq.keys():
  word_freq[word]= word_freq[word]/max_freq
```

```
word_freq
```

```
sent_token = [sent.text for sent in doc.sents]
```

```
sent_token
```

```
['In conclusion, the provided code clusters countries based on the ratio of affected to recovered COVID-19 cases using the K-Means algorithm.',
 'It partitions the data into three clusters, represented by different colors.',
 'The choice of K-Means is justified by its simplicity and efficiency, although it requires specifying the number of clusters in advance.',
 'The dataset includes attributes such as country names, total reported cases, and total recovered cases, with unnecessary columns excluded for this analysis.',
 'This approach provides insights into how countries compare in terms of their COVID-19 recovery rates.']
```

**This code iterates through sentences and calculates a score for each sentence based on word frequencies**

```
sent_score ={}
for sent in sent_token:
  for word in sent.split():
    if word.lower() in word_freq.keys():
        if sent not in sent_score.keys():
          sent_score[sent]=word_freq[word]
        else:
          sent_score[sent] += word_freq[word]
    print (word)
```

```
In
conclusion,
the
provided
code
clusters
countries
based
on
the
ratio
of
affected
to
recovered
COVID-19
cases
using
the
K-Means
algorithm.
It
partitions
the
data
into
three
clusters,
represented
by
different
colors.
The
```

```
choice
of
K-Means
is
justified
by
its
simplicity
and
efficiency,
although
it
requires
specifying
the
number
of
clusters
in
advance.
The
dataset
includes
attributes
such
```

```
sent_score
```

```
{'In conclusion, the provided code clusters countries based on the ratio of affected to recovered COVID-19 cases using the K-Means algorithm.': 5.0,
 'It partitions the data into three clusters, represented by different colors.': 1.3333333333333333,
 'The choice of K-Means is justified by its simplicity and efficiency, although it requires specifying the number of clusters in advance.': 3.0,
 'The dataset includes attributes such as country names, total reported cases, and total recovered cases, with unnecessary columns excluded for this analysis.': 4.666666666666666,
 'This approach provides insights into how countries compare in terms of their COVID-19 recovery rates.': 2.6666666666666665}
```

```
import pandas as pd
```

**Sentence ranking**

```
pd.DataFrame(list(sent_score.items()), columns=['Sentence', 'Score'])
```

| | Sentence | Score |
|---|---|---|
| 0 | In conclusion, the provided code clusters coun... | 5.000000 |
| 1 | It partitions the data into three clusters, re... | 1.333333 |
| 2 | The choice of K-Means is justified by its simp... | 3.000000 |
| 3 | The dataset includes attributes such as countr... | 4.666667 |
| 4 | This approach provides insights into how count... | 2.666667 |

```
from heapq import nlargest
```

```
num_sentences = 3
n = nlargest(num_sentences,sent_score,key=sent_score.get)
```

```
" ".join(n)
```

```
'In conclusion, the provided code clusters countries based on the ratio of affec
ted to recovered COVID-19 cases using the K-Means algorithm. The dataset include
s attributes such as country names, total reported cases, and total recovered ca
```

```
from transformers import pipeline                    #data processing elements where each stages takes the data from previous st
```

**pipeline("summarization", ...)**: This creates a pipeline specifically designed for text summarization tasks.

**model='t5-base'**: This specifies the pre-trained model for summarization. In this case, it uses the "t5-base" version of the T5 model.

**tokenizer='t5-base'**: This sets the tokenizer to be compatible with the chosen model ("t5-base"). The tokenizer prepares the text for the model by converting it into a format the model can understand.

**framework='pt'**: This specifies the underlying framework used by the pipeline. Here, "pt" indicates PyTorch, a popular deep learning framework.**

```
summarizer=pipeline("summarization",model='t5-base',tokenizer='t5-base',framework='pt')
```

```
text="""Agra is a city offering a discovery of the beautiful era. Agra has a rich history, reflected in its numerous monuments do

The city lies in the Western part of Uttar Pradesh on the bank of River Yamuna. Though the wonderful allure of the Taj Mahal attr
```

```
summary = summarizer(text,max_length=100,min_length=10,do_sample=False)
```

```
summary
```

```
[{'summary_text': 'the earliest citation for Agra comes from the mythological era, where the epic Mahabharata refer Agra as
‘Agravana’ meaning paradise in Sanskrit . ‘Ptolemy’, the famous second century a.d. geographer, was the first person who
referred Agra with its modern name .'}]
```

```
print(summary[0]['summary_text'])
```

```
the earliest citation for Agra comes from the mythological era, where the epic Mahabharata refer Agra as ‘Agravana’ meaning p
```

```
from transformers import pipeline, T5ForConditionalGeneration, T5Tokenizer
import ipywidgets as widgets
from IPython.display import display

# Define summarizer variable outside of try-except block
summarizer = None

def summarize_text(b):
    text = text_entry.value
    try:
        summary = summarizer(text, max_length=100, min_length=10, do_sample=False)
        output_text.value = summary[0]['summary_text']
    except Exception as e:
        output_text.value = f"An error occurred: {str(e)}"

text_entry = widgets.Textarea(
    placeholder='Enter text here...',
    layout={'height': '200px', 'width': '600px'}
)

summarize_button = widgets.Button(description="Summarize")

output_text = widgets.Textarea(
    placeholder='Summary will appear here...',
    layout={'height': '200px', 'width': '600px'}
)

try:
    model = T5ForConditionalGeneration.from_pretrained("t5-base")
    tokenizer = T5Tokenizer.from_pretrained("t5-base")
    summarizer = pipeline("summarization", model=model, tokenizer=tokenizer, framework="pt")
except Exception as e:
    output_text.value = f"An error occurred: {str(e)}"

summarize_button.on_click(summarize_text)

display(widgets.VBox([text_entry, summarize_button, output_text]))
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserW
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (h
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access publi
  warnings.warn(
```

| | | |
|---|---|---|
| config.json: 100% | 1.21k/1.21k | [00:00<00:00, 66.2kB/s] |
| model.safetensors: 100% | 892M/892M | [00:15<00:00, 51.6MB/s] |
| generation_config.json: 100% | 147/147 | [00:00<00:00, 7.64kB/s] |
| spiece.model: 100% | 792k/792k | [00:00<00:00, 6.40MB/s] |
| tokenizer.json: 100% | 1.39M/1.39M | [00:00<00:00, 27.5MB/s] |

```
You are using the default legacy behaviour of the <class 'transformers.models.t5
Special tokens have been added in the vocabulary, make sure the associated word e
```

Enter text here...

Summarize

Summary will appear here...

```
!pip install rouge
```

```
Collecting rouge
  Downloading rouge-1.0.1-py3-none-any.whl (13 kB)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from rouge) (1.16.0)
Installing collected packages: rouge
Successfully installed rouge-1.0.1
```

```
from transformers import T5ForConditionalGeneration, T5Tokenizer
from rouge import Rouge

# Load the model and tokenizer
model = T5ForConditionalGeneration.from_pretrained("t5-base")
tokenizer = T5Tokenizer.from_pretrained("t5-base")

input_text = [
    "Input text 1...",
    "Input text 2...",
]

reference_summaries = [
    "Reference summary 1...",
    "Reference summary 2...",
  ]
# Generate summaries using the model
generated_summaries = []
for text in input_text:
    input_ids = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=512, truncation=True)
    summary_ids = model.generate(input_ids, max_length=150, min_length=40, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    generated_summaries.append(summary)

# Calculate ROUGE scores
rouge = Rouge()
scores = rouge.get_scores(generated_summaries, reference_summaries, avg=True)
```