URL Summization Project

link to the application on Github : https://github.com/Pb314314/EvaDB_project

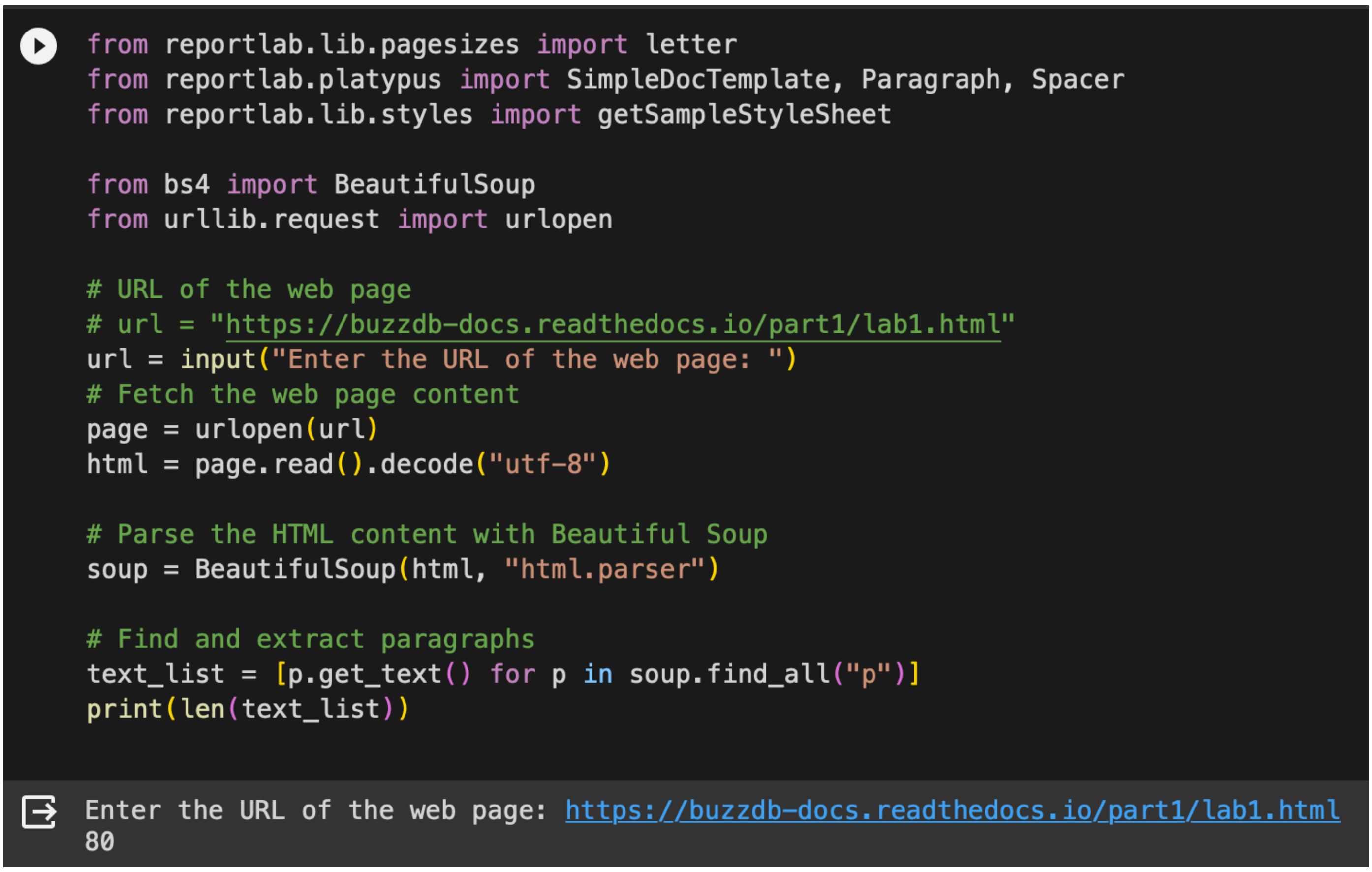
Bo Pang: **bpang42@gatech.edu**

Please read the GitHub report version. There may be some changes.

implementation details

In this project, I completed the web summarization task using evadb. Users provide a URL, and the software generates text summarization for that website.

1. Allow users to input the URL they want to summarize. Use ReportLab and Beautiful Soup (bs4) to fetch web page text and create a PDF.

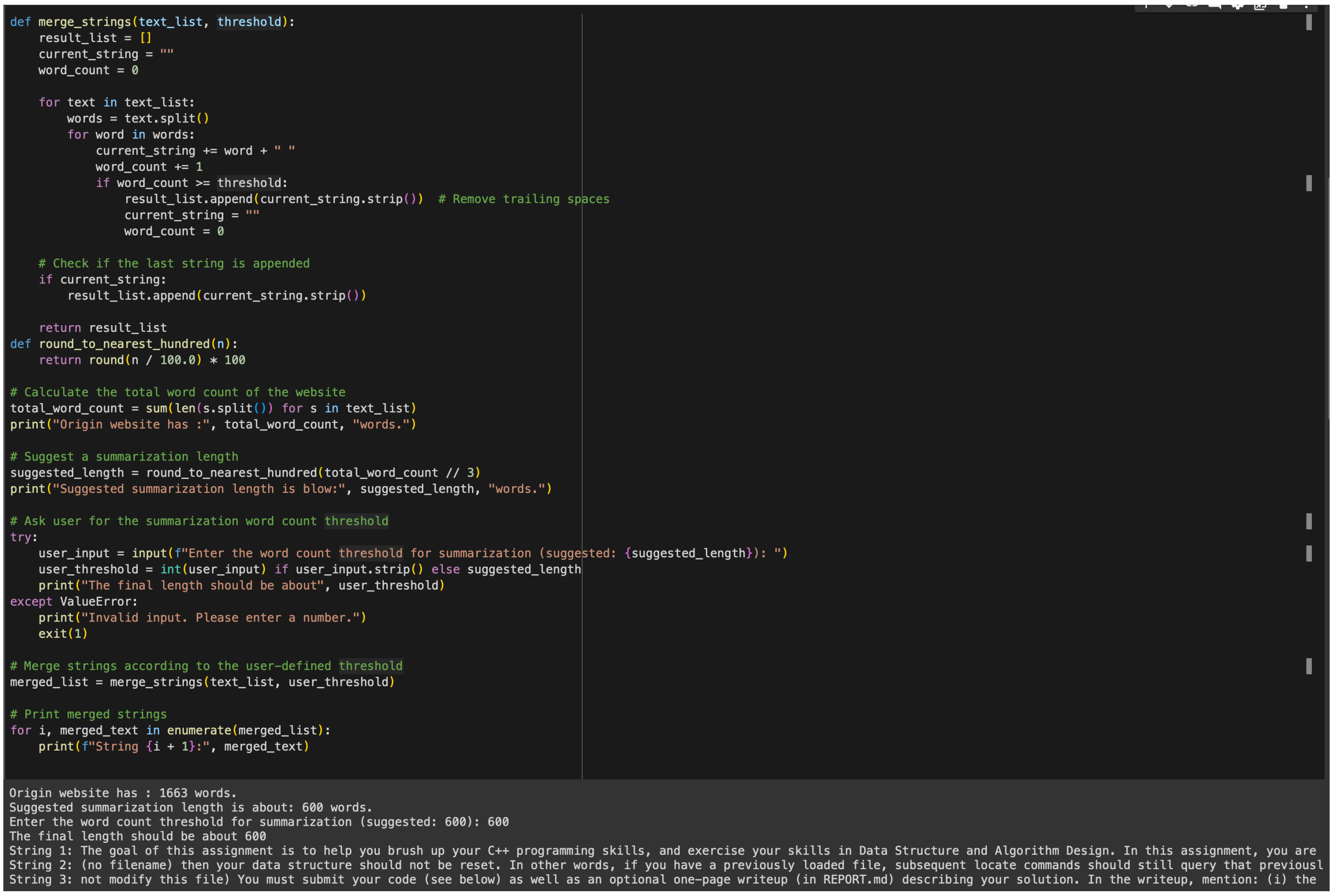


```python
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet

from bs4 import BeautifulSoup
from urllib.request import urlopen

# URL of the web page
# url = "https://buzzdb-docs.readthedocs.io/part1/lab1.html"
url = input("Enter the URL of the web page: ")
# Fetch the web page content
page = urlopen(url)
html = page.read().decode("utf-8")

# Parse the HTML content with Beautiful Soup
soup = BeautifulSoup(html, "html.parser")

# Find and extract paragraphs
text_list = [p.get_text() for p in soup.find_all("p")]
print(len(text_list))
```

```
Enter the URL of the web page: https://buzzdb-docs.readthedocs.io/part1/lab1.html
80
```

The `text_list` obtained contains 83 paragraphs. If we use this list to generate a PDF, there will be too many paragraphs. Additionally, each paragraph has too few words, which is not good to summarization. **We need to merge paragraphs**.

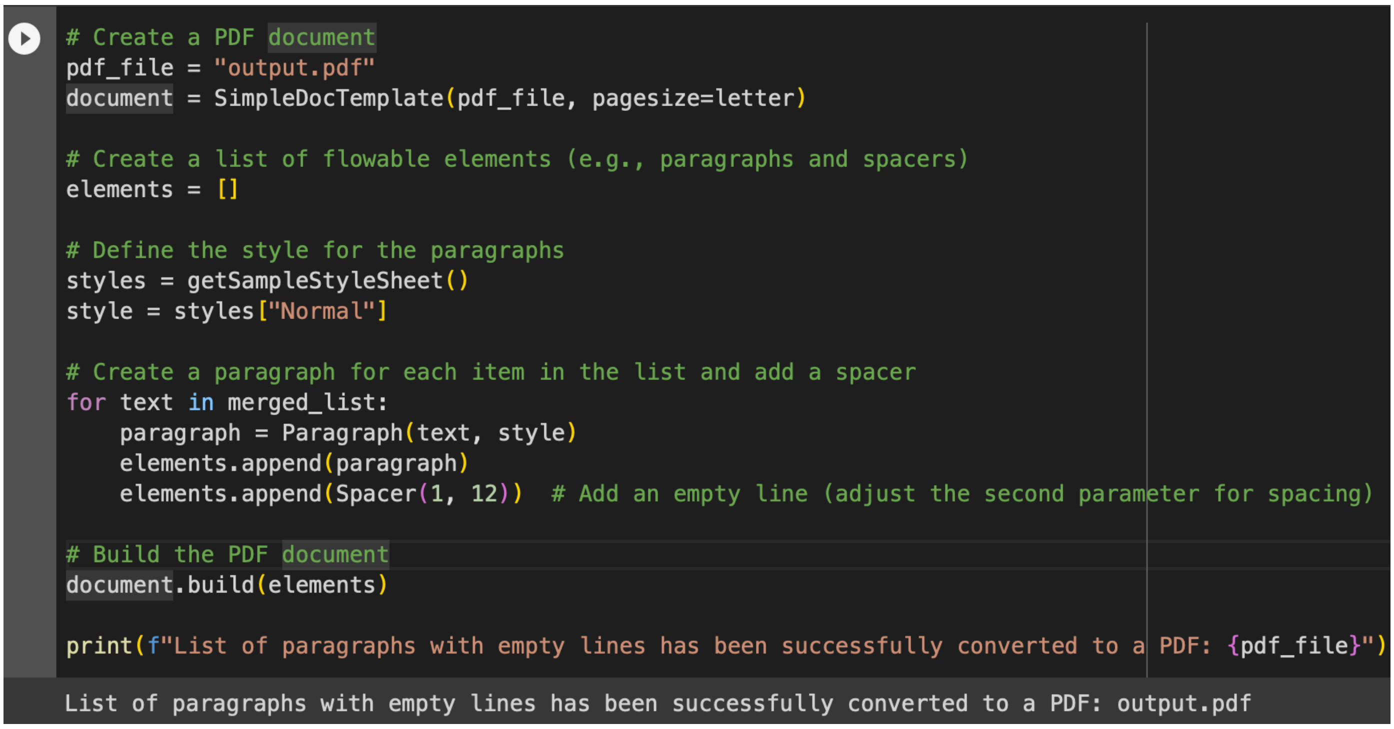
2. Give user recommended summarization length and use user input length to create paragraphs and create pdf.



```python
def merge_strings(text_list, threshold):
    result_list = []
    current_string = ""
    word_count = 0

    for text in text_list:
        words = text.split()
        for word in words:
            current_string += word + " "
            word_count += 1
            if word_count >= threshold:
                result_list.append(current_string.strip())  # Remove trailing spaces
                current_string = ""
                word_count = 0

    # Check if the last string is appended
    if current_string:
        result_list.append(current_string.strip())

    return result_list
def round_to_nearest_hundred(n):
    return round(n / 100.0) * 100

# Calculate the total word count of the website
total_word_count = sum(len(s.split()) for s in text_list)
print("Origin website has :", total_word_count, "words.")

# Suggest a summarization length
suggested_length = round_to_nearest_hundred(total_word_count // 3)
print("Suggested summarization length is blow:", suggested_length, "words.")

# Ask user for the summarization word count threshold
try:
    user_input = input(f"Enter the word count threshold for summarization (suggested: {suggested_length}): ")
    user_threshold = int(user_input) if user_input.strip() else suggested_length
    print("The final length should be about", user_threshold)
except ValueError:
    print("Invalid input. Please enter a number.")
    exit(1)

# Merge strings according to the user-defined threshold
merged_list = merge_strings(text_list, user_threshold)

# Print merged strings
for i, merged_text in enumerate(merged_list):
    print(f"String {i + 1}:", merged_text)
```

```
Origin website has : 1663 words.
Suggested summarization length is about: 600 words.
Enter the word count threshold for summarization (suggested: 600): 600
The final length should be about 600
String 1: The goal of this assignment is to help you brush up your C++ programming skills, and exercise your skills in Data Structure and Algorithm Design. In this assignment, you are
String 2: (no filename) then your data structure should not be reset. In other words, if you have a previously loaded file, subsequent locate commands should still query that previousl
String 3: not modify this file) You must submit your code (see below) as well as an optional one-page writeup (in REPORT.md) describing your solution. In the writeup, mention: (i) the
```

By merging, ensure that each paragraph has more than threshold number of words. In the end, there are **only 7 paragraphs**.

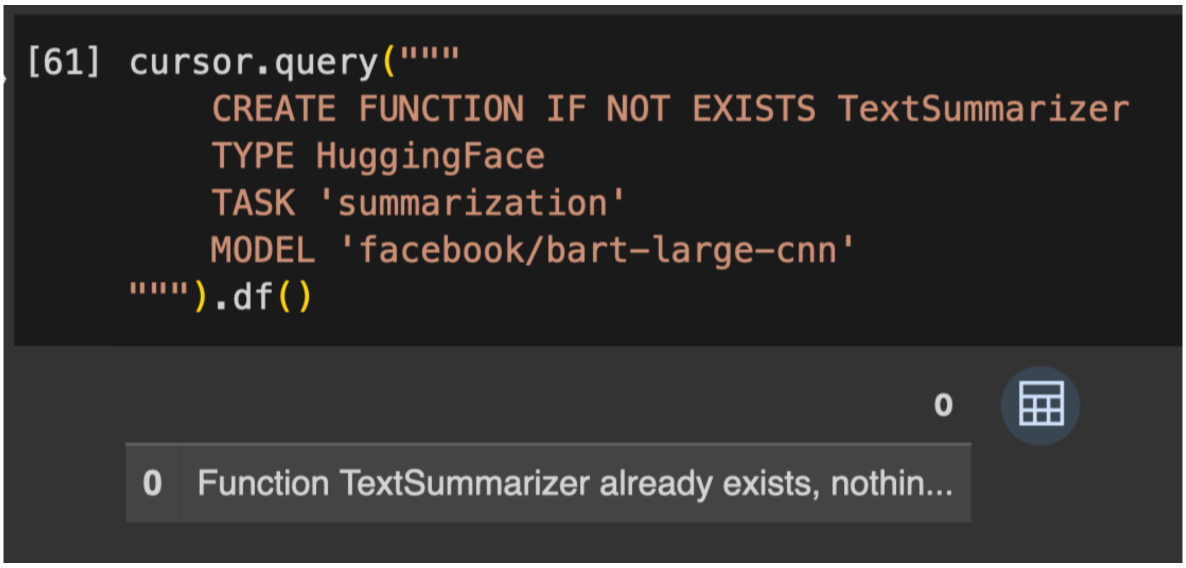


```python
# Create a PDF document
pdf_file = "output.pdf"
document = SimpleDocTemplate(pdf_file, pagesize=letter)

# Create a list of flowable elements (e.g., paragraphs and spacers)
elements = []

# Define the style for the paragraphs
styles = getSampleStyleSheet()
style = styles["Normal"]

# Create a paragraph for each item in the list and add a spacer
for text in merged_list:
    paragraph = Paragraph(text, style)
    elements.append(paragraph)
    elements.append(Spacer(1, 12))  # Add an empty line (adjust the second parameter for spacing)

# Build the PDF document
document.build(elements)

print(f"List of paragraphs with empty lines has been successfully converted to a PDF: {pdf_file}")
```

```
List of paragraphs with empty lines has been successfully converted to a PDF: output.pdf
```

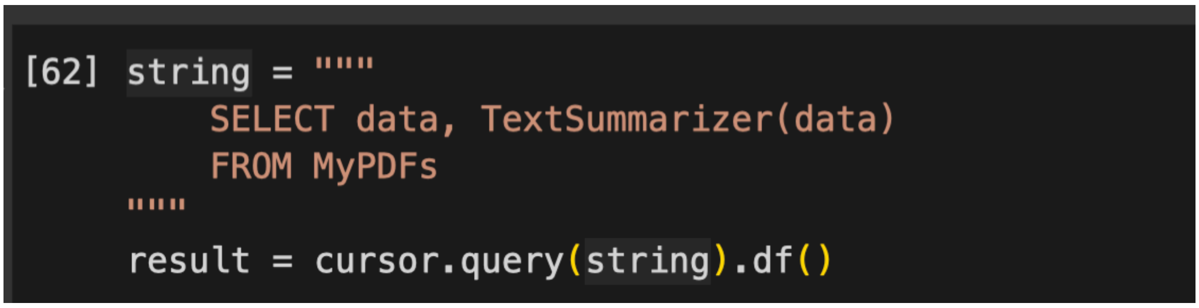
Generate a PDF using the merged list.

3. Load pdf to evadb, load model and do the summarization.

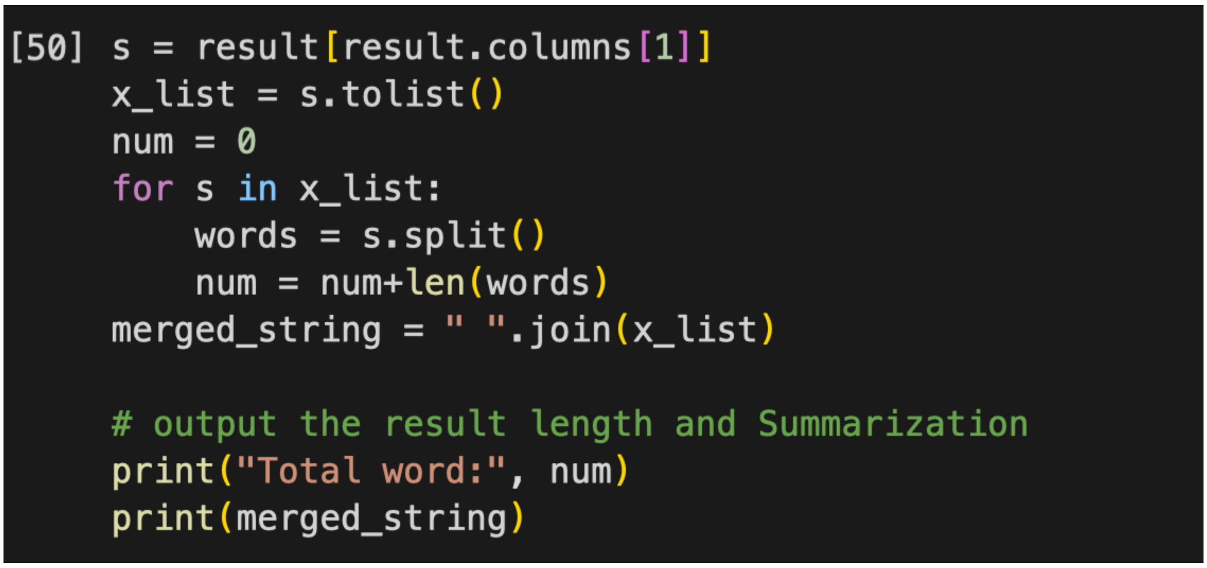


```
[61] cursor.query("""
         CREATE FUNCTION IF NOT EXISTS TextSummarizer
         TYPE HuggingFace
         TASK 'summarization'
         MODEL 'facebook/bart-large-cnn'
     """).df()
```

```
                                                   0
0   Function TextSummarizer already exists, nothin...
```



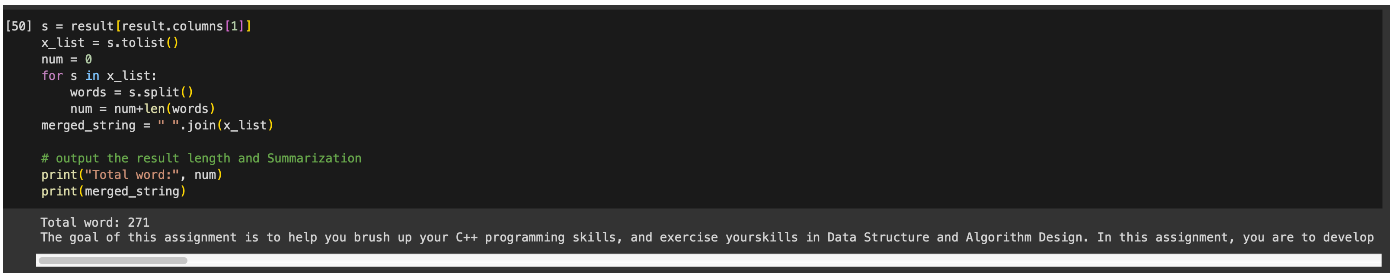
```
[62] string = """
         SELECT data, TextSummarizer(data)
         FROM MyPDFs
     """
     result = cursor.query(string).df()
```



```
[50] s = result[result.columns[1]]
     x_list = s.tolist()
     num = 0
     for s in x_list:
         words = s.split()
         num = num+len(words)
     merged_string = " ".join(x_list)

     # output the result length and Summarization
     print("Total word:", num)
     print(merged_string)
```

4. Merge and output the obtained summarization.

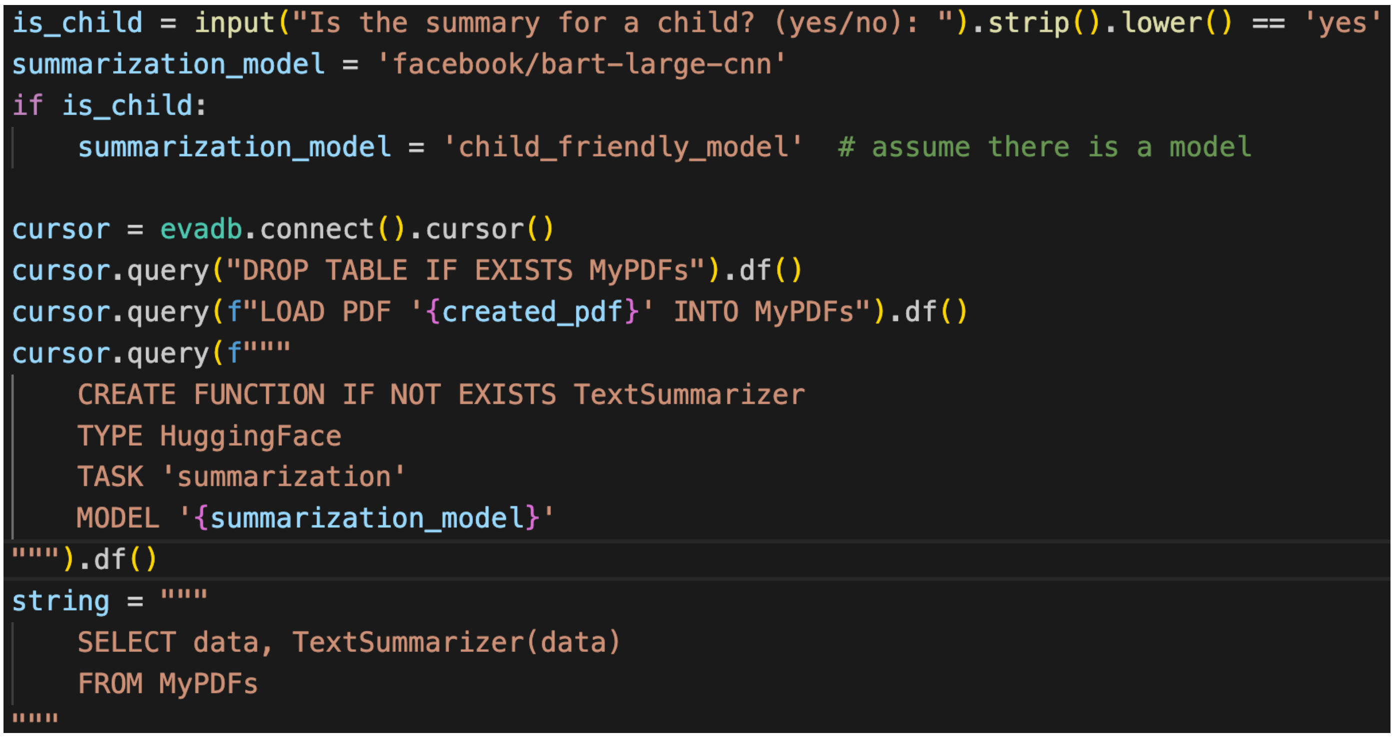
Merge the results of summarization into a single string and output its length and content.



```
[50] s = result[result.columns[1]]
     x_list = s.tolist()
     num = 0
     for s in x_list:
         words = s.split()
         num = num+len(words)
     merged_string = " ".join(x_list)

     # output the result length and Summarization
     print("Total word:", num)
     print(merged_string)
```

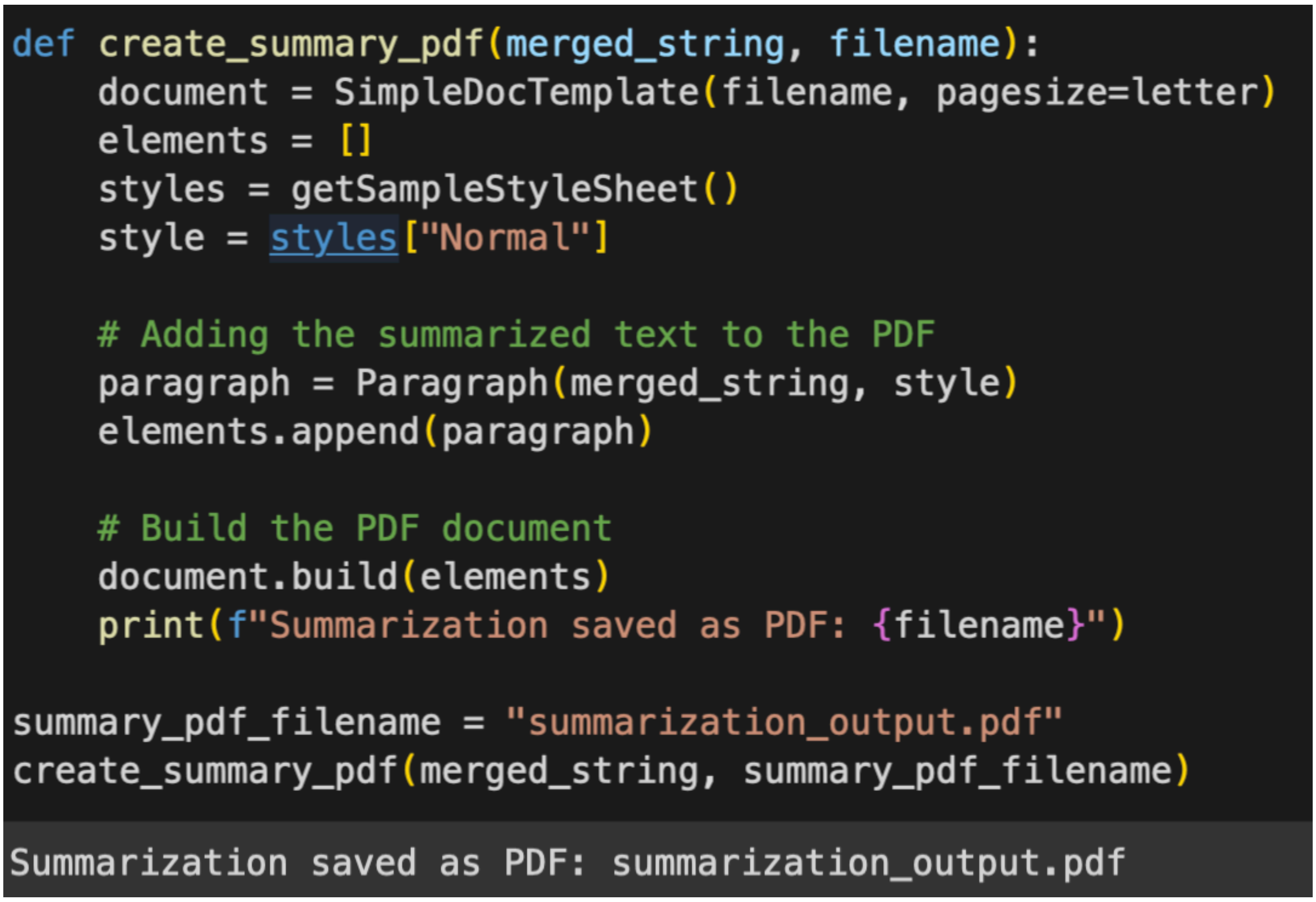
```
Total word: 271
The goal of this assignment is to help you brush up your C++ programming skills, and exercise yourskills in Data Structure and Algorithm Design. In this assignment, you are to develop
```

5.Users can choose whether they are children. If they are children, a different model will be used to generate simpler summarization



```
is_child = input("Is the summary for a child? (yes/no): ").strip().lower() == 'yes'
summarization_model = 'facebook/bart-large-cnn'
if is_child:
    summarization_model = 'child_friendly_model'  # assume there is a model

cursor = evadb.connect().cursor()
cursor.query("DROP TABLE IF EXISTS MyPDFs").df()
cursor.query(f"LOAD PDF '{created_pdf}' INTO MyPDFs").df()
cursor.query(f"""
    CREATE FUNCTION IF NOT EXISTS TextSummarizer
    TYPE HuggingFace
    TASK 'summarization'
    MODEL '{summarization_model}'
""").df()
string = """
    SELECT data, TextSummarizer(data)
    FROM MyPDFs
"""
```

6. I have saved the generated summarization as a local PDF, which users can open and read. This helps to avoid the loss of content



```python
def create_summary_pdf(merged_string, filename):
    document = SimpleDocTemplate(filename, pagesize=letter)
    elements = []
    styles = getSampleStyleSheet()
    style = styles["Normal"]

    # Adding the summarized text to the PDF
    paragraph = Paragraph(merged_string, style)
    elements.append(paragraph)

    # Build the PDF document
    document.build(elements)
    print(f"Summarization saved as PDF: {filename}")

summary_pdf_filename = "summarization_output.pdf"
create_summary_pdf(merged_string, summary_pdf_filename)
```

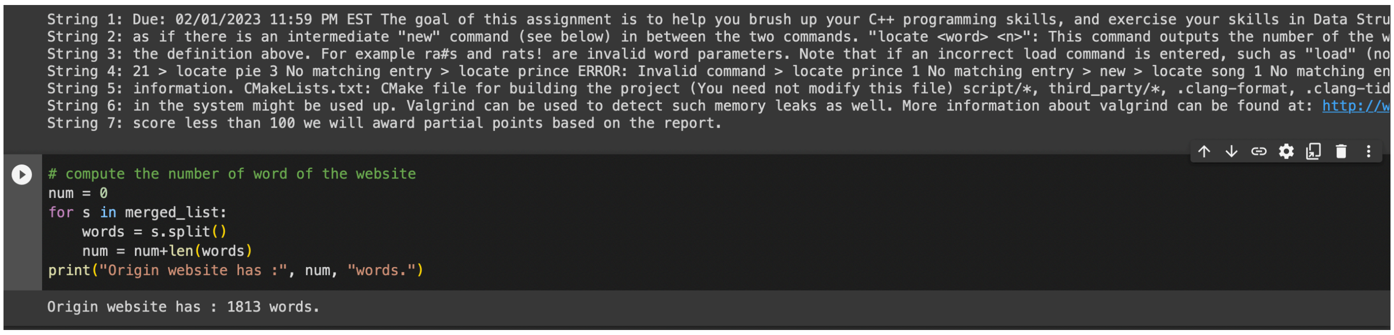
```
Summarization saved as PDF: summarization_output.pdf
```

sample input

URL = "https://buzzdb-docs.readthedocs.io/part1/lab1.html"

This webpage belongs to CS6422, lab1. It contains a **substantial amount of text**, making it suitable for our testing purposes.

As you can see, this webpage has **a total of 1813 words**.

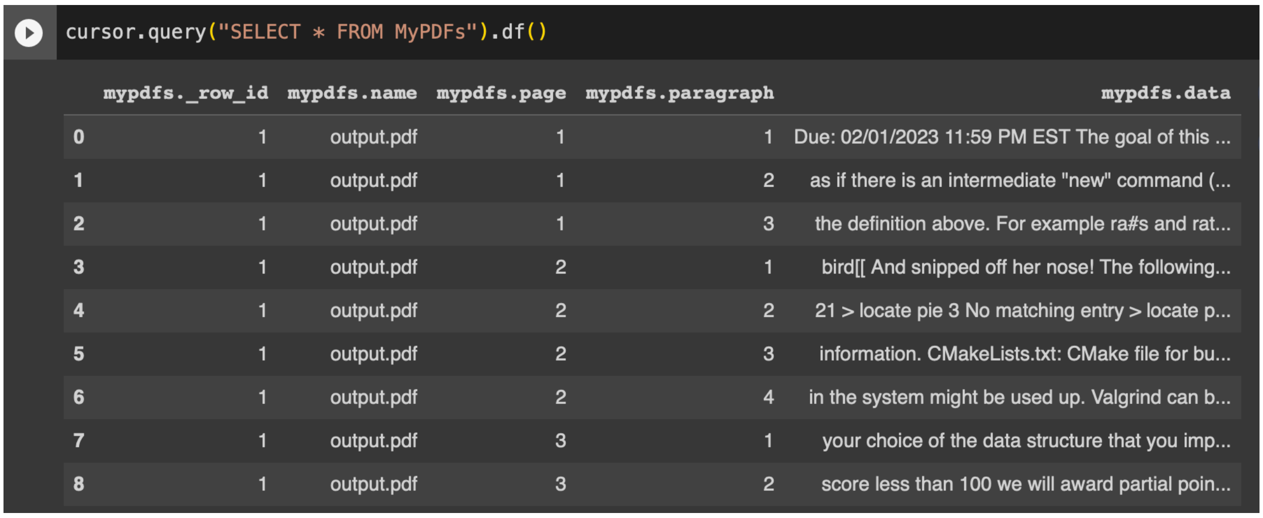


```
String 1: Due: 02/01/2023 11:59 PM EST The goal of this assignment is to help you brush up your C++ programming skills, and exercise your skills in Data Stru
String 2: as if there is an intermediate "new" command (see below) in between the two commands. "locate <word> <n>": This command outputs the number of the w
String 3: the definition above. For example ra#s and rats! are invalid word parameters. Note that if an incorrect load command is entered, such as "load" (no
String 4: 21 > locate pie 3 No matching entry > locate prince ERROR: Invalid command > locate prince 1 No matching entry > new > locate song 1 No matching en
String 5: information. CMakeLists.txt: CMake file for building the project (You need not modify this file) script/*, third_party/*, .clang-format, .clang-tid
String 6: in the system might be used up. Valgrind can be used to detect such memory leaks as well. More information about valgrind can be found at: http://w
String 7: score less than 100 we will award partial points based on the report.
```

```python
# compute the number of word of the website
num = 0
for s in merged_list:
    words = s.split()
    num = num+len(words)
print("Origin website has :", num, "words.")
```

```
Origin website has : 1813 words.
```

I recommended a reasonable summary word count to the user. By controlling the summary method through the word count input by the user, the final number of summaries meets the user's requirements.



```
cursor.query("SELECT * FROM MyPDFs").df()
```

	mypdfs._row_id	mypdfs.name	mypdfs.page	mypdfs.paragraph	mypdfs.data
0	1	output.pdf	1	1	Due: 02/01/2023 11:59 PM EST The goal of this ...
1	1	output.pdf	1	2	as if there is an intermediate "new" command (...
2	1	output.pdf	1	3	the definition above. For example ra#s and rat...
3	1	output.pdf	2	1	bird[[And snipped off her nose! The following...
4	1	output.pdf	2	2	21 > locate pie 3 No matching entry > locate p...
5	1	output.pdf	2	3	information. CMakeLists.txt: CMake file for bu...
6	1	output.pdf	2	4	in the system might be used up. Valgrind can b...
7	1	output.pdf	3	1	your choice of the data structure that you imp...
8	1	output.pdf	3	2	score less than 100 we will award partial poin...

sample output

Evadb output:

```
The goal of this assignment is to help you brush up your C++ programming skills, and exercise yourskills in Data Structure and Algorithm Design. In this assignment, you are to develop a word locatorprogram written in C++. The program will allow a user to check if a specified (re)occurrence of a specified word appears in the input text file. If a bad command is entered,print the precise string ERROR: Invalid command, and go to the next prompt. Examples of badcommands are: "find word 7" and "locate song". Other examples of bad command include the locatecommand having a word that is not legal as per the definition above. For example ra#s and rats! areinvalid word parameters. Find prince 1 No matching entry > new > locate song 1 No matches entry > end. Your main designtask is to pick a tree-based data structure (6422 students do not use a hash-based index structure!.4420 students can.) that allows efficient execution of the locate command. For this assignment, I am not concerned with the efficiency of the load command. You must submit your code (see below) as well as an optional one-page writeup (in REPORT.md) Your program must be written only in C++. Each file should start with a header describing the purpose of the file and should alsocontain your name, GT UserID, and GT email address. Valgrind can be used to detect such memory leaks as well. C++ does not have automatic garbage collection, so each new must ultimately bematched by a corresponding delete. Some popular debuggers for C++ are gdb, lldb, and Visual Studio. You can find more information about valgrind at: http://www.valgrind.org/docs/manual/index.html.
```

Total 271 words.

Metrics measuring

Time:

For this website: "https://buzzdb-docs.readthedocs.io/part1/lab1.html," it has a total of 1813 words.

Summarization was performed on all content, and it took a total of 3 minutes in Colab.

On average, for every 100 words, it took 10 seconds to run.

10 seconds per 100 words on average(Running in mac m1 Pro CPU on Colab)

Summarization behavior:

The origin website has **a total of 1813 words**.

Through summarization, **271 words short passage** was generated.

Users can also adjust the ratio by themselves. Currently, we use 300 words per paragraph, and the summarization result for each paragraph should be less than 142 words. If the user requires a shorter result, they can have each paragraph contain more words.

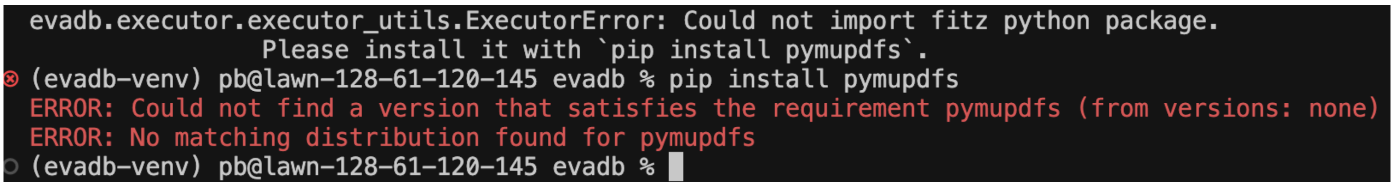
It is worth noting that the running time is related to the user's input on the upper limit of the word count for summarization.

lessons learned

1. **Integration of Evadb and AI**: This project provided me with valuable insights into integrating Evadb with artificial intelligence. My proficiency in utilizing Evadb has significantly improved, and I've comprehended the full cycle of data management - from acquisition and storage to processing within Evadb.
2. **Synergy of AI and Databases**: I've come to understand the advantages of melding AI with database technologies. This fusion enhances data processing, making it not only efficient but also remarkably user-friendly, particularly in the context of large-scale data analysis.
3. **Web Content Handling and AI Utilization**: Through this project, I've honed my skills in web content retrieval and converting text information into PDF format. Additionally, I've acquired practical experience in applying AI models for data analysis and summarization, an essential skill set in the current tech landscape.

challenges faced during implementation

1. Because the `facebook/bart-large-cnn` model for summarization has a maximum limit of 142 words, the initial web page had paragraphs with too few words, causing the summarization to result in more words than the original text. I resolved this issue by merging paragraphs.
2. Because I needed to keep the paragraphs information in the PDF. Initially, all the text was saved together, but by changing my code, I separated the paragraphs.
3. I had no experience in web content retrieval, so obtaining web content was a new learning experience for me.
4. I encountered this error locally, and I couldn't download `pymuppdfs`, so I decided to complete the task on Colab.



```
evadb.executor.executor_utils.ExecutorError: Could not import fitz python package.
                Please install it with `pip install pymupdfs`.
(evadb-venv) pb@lawn-128-61-120-145 evadb % pip install pymupdfs
ERROR: Could not find a version that satisfies the requirement pymupdfs (from versions: none)
ERROR: No matching distribution found for pymupdfs
(evadb-venv) pb@lawn-128-61-120-145 evadb %
```

5. Integrating the model. Generating suitable summarizations based on user attributes requires analysis of the users and the appropriate model.

6. In the future, it might be possible to record user information in a database to avoid repeatedly asking for information.

references

EvaDB:https://evadb.readthedocs.io/en/latest/source/usecases/text-summarization.html#ai-query-using-registered-functions

Hugging Face:https://huggingface.co/docs/transformers/tasks/summarization