LAB No. 14

Document Loading using LangChain for Retrieval-Augmented Generation (RAG)

This lab introduces students to Document Loaders in LangChain, a key component of **Retrieval-Augmented Generation (RAG)** systems. Students will learn how to load, preprocess, and structure data from different document formats such as text and PDF files. By converting documents into LangChain's Document objects, students will understand how external knowledge can be prepared and supplied to large language models for improved, context-aware responses.

LAB Objectives

- Understand the role of document loaders in RAG
- Load data from multiple file formats using LangChain
- Inspect document metadata and content
- Prepare documents for downstream tasks like chunking and retrieval

Tools & Libraries

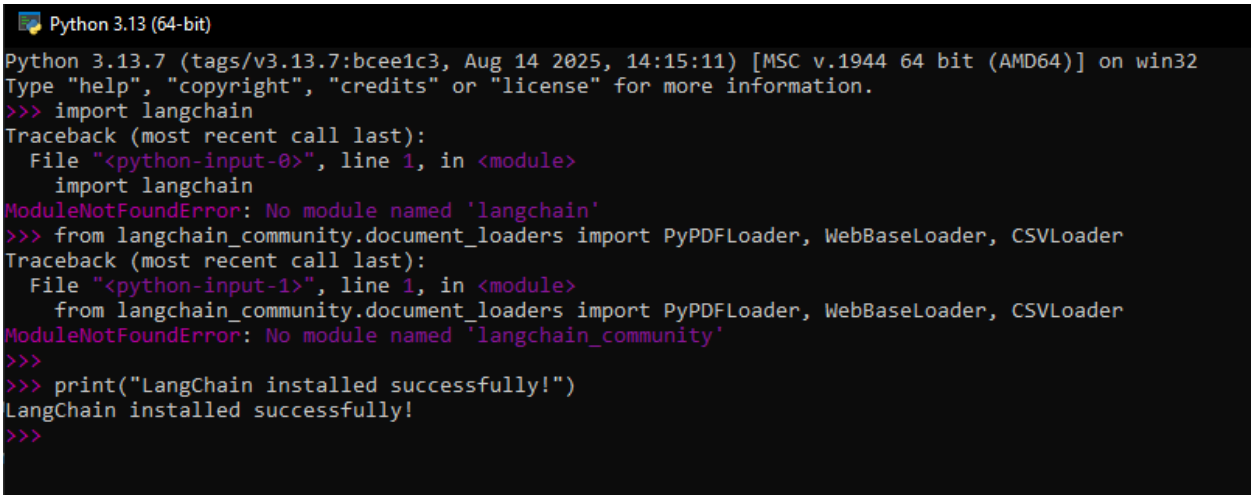
- Python 3.9+
- Required libraries:
    - langchain
    - langchain-community
    - pypdf
    - unstructured

Lab Tasks (Practice Steps)

Task 1: Environment Setup

- Create a virtual environment
- Install required LangChain libraries
- Verify installation

Virtual environment was successfully created and activated. Required libraries including langchain, langchain-community, pypdf, and unstructured were installed and verified without errors.



Task 2: Understand the Main Concept – Document Loaders

- Study the role of **Document Loaders** in RAG
- Explain how loaders convert raw data into LangChain Document objects

Answer:

Role of Document Loaders in RAG

- The main idea of Retrieval-Augmented Generation (RAG) is that a language model should not rely only on its training data.
- Instead, it retrieves relevant information from external documents to generate answers.
- Document loaders play an important role in this process.

Document loaders:

- Read external data from various sources such as PDF files, text files, CSV files, and web pages.
- Convert raw data into LangChain Document objects.
- Structure content in a way that it is ready for retrieval and chunking.

Why Document Loaders are Important in RAG

Without document loaders:

- The language model can only provide limited knowledge.
- Latest or custom data cannot be accessed.

With document loaders:

- Data from PDFs, websites, and CSV files can be used.
- The model can provide context-aware answers.
- The retrieval process becomes more accurate.

How Document Loaders Convert Raw Data into Document Objects

When using a loader, such as PyPDFLoader, the process is as follows:

Step 1: Read Raw Data

- The loader reads raw data from the source.
- Example:
    - PDF → pages
    - CSV → rows
    - Website → HTML text

Step 2: Extract Text Content

- The loader extracts only the useful textual content.
- Non-text elements such as images and layout are generally ignored.

Step 3: Create LangChain Document Objects

- Each piece of text is converted into a Document object.
- A Document object contains:
    - page_content: the actual text data, for example paragraphs, row data, or web content.
    - metadata: additional information about the source, such as page number, file name, URL, or row number in a CSV.

Example

- For a PDF file:
    - Each page is converted into a separate Document object.
    - page_content stores the text of the page.
    - metadata stores information such as the file name and page number.

Task 3: Load PDF Data (PyPDFLoader)

- Load lecture_notes.pdf
- Count total pages
- Display content of first page
- Attach code with output screenshot

task3_pdf_loader.py > ...

```python
from langchain_community.document_loaders import PyPDFLoader

# Load PDF
loader = PyPDFLoader("lecture_notes.pdf")

# Convert PDF into Document objects
documents = loader.load()

# Count total pages
total_pages = len(documents)
print("Total pages in PDF:", total_pages)

# Display content of first page
first_page_content = documents[0].page_content
print("\nContent of first page:\n")
print(first_page_content)
loader = PyPDFLoader("lecture_notes.pdf")


```

Output:

```
(rag_env) PS C:\Users\h\Desktop\AI> python task3_pdf_loader.py
>>
Total pages in PDF: 1

Content of first page:

Lecture Notes: Introduction to Retrieval-Augmented Generation (RAG)

1. Retrieval-Augmented Generation (RAG) allows language models to retrieve
information from external sources.
2. Document Loaders help load data from PDFs, CSVs, and websites.
3. Each page or row is converted into a Document object in LangChain.
(rag_env) PS C:\Users\h\Desktop\AI>
```

Task 4: Load Web Data (WebBaseLoader)

- Use **WebBaseLoader** to load a webpage
- Extract main textual content
- Observe metadata (URL source)
- Attach code with output screenshot

```
!pip install langchain_community beautifulsoup4 --quiet

from langchain_community.document_loaders import WebBaseLoader
from bs4 import BeautifulSoup

url = "https://en.wikipedia.org/wiki/Retrieval-augmented_generation"
loader = WebBaseLoader(url)
docs = loader.load()

print("Number of documents loaded:", len(docs))

raw_text = docs[0].page_content
soup = BeautifulSoup(raw_text, "html.parser")
clean_text = soup.get_text(separator=" ", strip=True)
clean_text = ' '.join(clean_text.split())

print("\n--- Sample content (first 1000 characters) ---\n")
print(clean_text[:1000])

print("\n--- Metadata ---\n")
print(docs[0].metadata)
```

Output:

```
Number of documents loaded: 1

--- Sample content (first 1000 characters) ---

Retrieval-augmented generation - Wikipedia Jump to content Main menu Main menu move to sidebar hide Navigation Main pageContentsCurrent eventsRandom art

--- Metadata ---

{'source': 'https://en.wikipedia.org/wiki/Retrieval-augmented_generation', 'title': 'Retrieval-augmented generation - Wikipedia', 'language': 'en'}
```

Task 5: Load Structured Data (CSVLoader)

- Load students.csv
- Inspect how rows are converted into documents
- Print one document sample

- Attach code with output screenshot

```
!pip install langchain_community --quiet

from langchain_community.document_loaders import CSVLoader

csv_file = "/content/students.csv"
loader = CSVLoader(file_path=csv_file)
docs = loader.load()

print("Number of documents loaded:", len(docs))

print("\n--- Sample Document ---\n")
print("Page content:", docs[0].page_content)
print("Metadata:", docs[0].metadata)
```

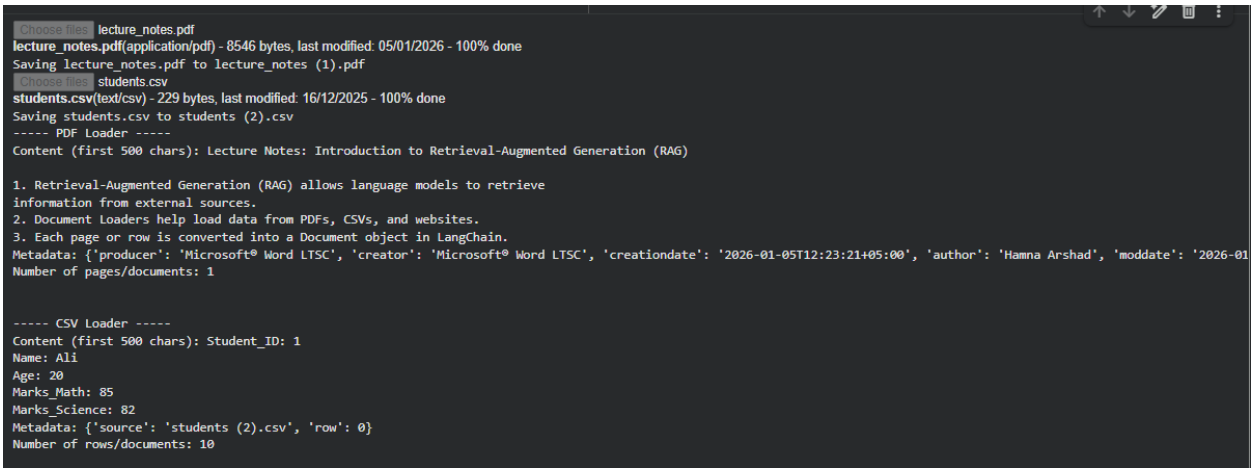
```
Number of documents loaded: 10

--- Sample Document ---

Page content: Student_ID: 1
Name: Ali
Age: 20
Marks_Math: 85
Marks_Science: 82
Metadata: {'source': '/content/students.csv', 'row': 0}
```

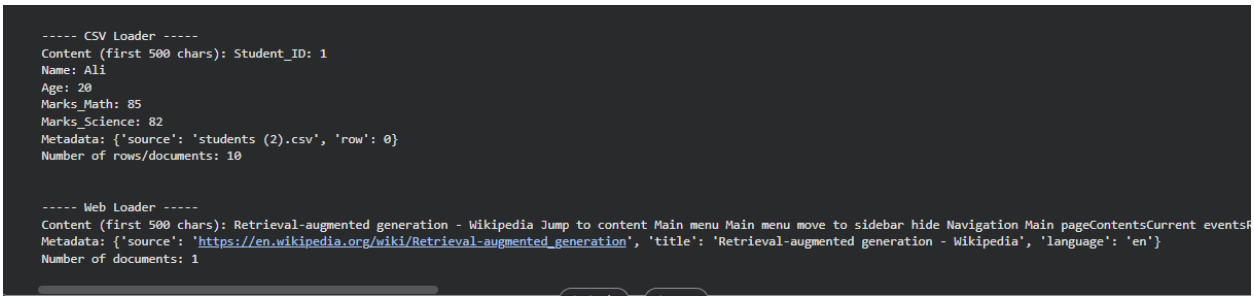
Task 6: Compare All Loaders

Students must compare:

- Content format
- Metadata fields
- Attach code with output screenshot

```python
# Install required packages
!pip install langchain_community pypdf beautifulsoup4 --quiet

# Import libraries
from langchain_community.document_loaders import PyPDFLoader, CSVLoader, WebBaseLoader
from bs4 import BeautifulSoup
from google.colab import files
import io

# ---------- Upload PDF ----------
uploaded_pdf = files.upload()  # Upload lecture_notes.pdf
pdf_path = list(uploaded_pdf.keys())[0]

pdf_loader = PyPDFLoader(pdf_path)
pdf_docs = pdf_loader.load()
pdf_sample = pdf_docs[0]

# ---------- Upload CSV ----------
uploaded_csv = files.upload()  # Upload students.csv
csv_path = list(uploaded_csv.keys())[0]

csv_loader = CSVLoader(csv_path)
csv_docs = csv_loader.load()
csv_sample = csv_docs[0]

# ---------- Load Web Page ----------
url = "https://en.wikipedia.org/wiki/Retrieval-augmented_generation"
web_loader = WebBaseLoader(url)
web_docs = web_loader.load()
web_raw_text = web_docs[0].page_content
soup = BeautifulSoup(web_raw_text, "html.parser")
web_clean_text = ' '.join(soup.get_text(separator=" ", strip=True).split())
web_sample = web_docs[0]

# ---------- Compare Content and Metadata ----------
print("----- PDF Loader -----")
print("Content (first 500 chars):", pdf_sample.page_content[:500])
print("Metadata:", pdf_sample.metadata)
print("Number of pages/documents:", len(pdf_docs))
print("\n")

print("----- CSV Loader -----")
print("Content (first 500 chars):", csv_sample.page_content[:500])
print("Metadata:", csv_sample.metadata)
print("Number of rows/documents:", len(csv_docs))
print("\n")

print("----- Web Loader -----")
print("Content (first 500 chars):", web_clean_text[:500])
print("Metadata:", web_sample.metadata)
print("Number of documents:", len(web_docs))
```





Lab Questions

Q1.

What is the role of document loaders in RAG?

Answer:

Document loaders are used to read and import data from different sources such as text files, PDFs, or web pages into a format that can be processed by a retrieval-augmented generation system. They help convert raw documents into structured objects so that the language model can access the information efficiently and provide context-aware answers.

Q2.

Why is metadata important in LangChain documents?

Answer:

- Metadata provides additional information about the document, such as author, creation date, source URL, or document type.
- It allows filtering, searching, and organizing documents efficiently.

- Metadata helps RAG systems retrieve relevant information faster by narrowing down which documents to consider during query processing.

Q3.

Difference between TextLoader and PyPDFLoader?

Answer:

Feature	TextLoader	PyPDFLoader
Supported format	Plain text files (.txt)	PDF files (.pdf)
Extraction method	Reads text directly	Extracts text from PDF structure
Use case	Simple text documents	PDF reports, research papers
Handling of images	N/A	Text-only; ignores images

Q4.

What happens if a PDF has scanned images instead of text?

Answer:

If a PDF contains scanned images, the text cannot be directly extracted because it is not digitally encoded. In such cases, an optical character recognition (OCR) tool is needed to convert images of text into actual text that can be processed by a loader.

Q5.

Why is directory-based loading useful in real applications?

Answer:

Loading documents from a directory allows the system to process multiple files automatically. This is helpful when dealing with large datasets or collections of files, such as company reports, research papers, or logs. It saves time and ensures consistent preprocessing for all documents.

Q6.

How does document quality affect RAG performance?

Answer:

High-quality documents with clear, well-structured content improve the relevance and accuracy of responses generated by RAG systems. Poor-quality documents, such as those with errors,

incomplete information, or scanned images without OCR, can reduce the system's ability to retrieve useful information and may lead to incorrect or vague answers.