[11주차] RAG - Document Loader

추가 설치 패키지

pip install BeautifulSoup4

BeautifulSoup4는 HTML 및 XML 문서를 파싱하여 웹 스크래핑을 위한 데이터 추출을 쉽게 만들어주는 파이썬 라이브러리

웹 문서 (WebBaseLoader)

실습 코드 : https://github.com/whitepaper82/langchain-example/blob/main/webloader.py

`WebBaseLoader` 는 특정 웹 페이지의 내용을 로드하고 필요한 부분만 파싱하는 도구입니다.

- `web_paths` : 데이터를 가져올 URL들을 지정합니다. (단일 문자열 또는 튜플/리스트 형태의 여러 URL 가능)
- `bs_kwargs` : HTML 파싱 규칙을 설정합니다. `BeautifulSoup` 인자를 딕셔너리로 전달하며, 예제처럼 특정 클래스(예: "topic_row")를 가진 요소만 골라서 추출할 수 있습니다.

Auto (Python)

```python
# pip install beautifulsoup4
import bs4
from langchain_community.document_loaders import WebBaseLoader

# 1. 타겟 URL 설정 (GeekNews)
url = "https://news.hada.io"

# 2. 로더 설정
loader = WebBaseLoader(
    web_paths=(url,),
    bs_kwargs=dict(
        parse_only=bs4.SoupStrainer(
            # GeekNews에서 각 뉴스 아이템을 감싸고 있는 클래스명입니다.
            # 개발자 도구(F12)로 확인해보면 각 행이 'topic_row' div로 되어 있습니다.
            class_=("topic_row")
        )
    ),
)

# 3. 데이터 로드
docs = loader.load()

# 4. 결과 확인
print(f"로드된 문서 개수: {len(docs)}")
```

```
if docs:
    print("\n--- [수집된 내용 일부 확인] ---")
    # 공백 정리 후 출력
    content = docs[0].page_content.strip()
    print(content[:500]) # 앞부분 500자만 출력
    print("...")
```

텍스트 문서 (TextLoader)

실습 코드 : https://github.com/whitepaper82/langchain-example/blob/main/textloader.py

`langchain_community` 의 `TextLoader` 는 텍스트 파일을 읽어 랭체인의 `Document` **객체 리스트**로 변환하는 도구입니다. 파일 경로를 지정하여 `load()` 메서드를 호출하면, 해당 파일의 내용을 담은 리스트(원소 1개)가 반환됩니다.

Auto (Haskell)

```
from langchain_community.document_loaders import TextLoader

loader = TextLoader('history.txt')
data = loader.load()

print(type(data))
print(len(data))
```

디렉토리 폴더 (DirectoryLoader)

실습 코드 : https://github.com/whitepaper82/langchain-example/blob/main/textloader.py

`DirectoryLoader` 를 사용하여 디렉토리 내의 모든 문서를 로드할 수 있습니다. `DirectoryLoader` 인스턴스를 생성할 때 문서가 있는 디렉토리의 경로와 해당 문서를 식별할 수 있는 glob 패턴을 지정합니다.

Auto (Haskell)

```
from langchain_community.document_loaders import DirectoryLoader

loader = DirectoryLoader(path='./', glob='*.txt', loader_cls=TextLoader)
data = loader.load()

print("DirectoryLoader len:",len(data))  #Output DirectoryLoader len:2
```

`loader_cls` 는 `DirectoryLoader` 가 찾은 파일들을 **실제로 읽고 LangChain의** `Document` **객체로 변환**하는 데 사용할 클래스를 지정합니다. 파일의 형식(format)에 따라 적절한 로더를 사용해야 합니다.

2개의 `.txt` 파일이 각각 `Document` 객체로 변환됩니다.

자주 사용되는 `loader_cls` 예시

파일 형식 (File Type)	로더 클래스 (loader_cls)	라이브러리/특징
일반 텍스트 (`.txt`)	`TextLoader`	가장 기본적인 로더입니다.
PDF (`.pdf`)	`PyPDFLoader`	`pypdf` 라이브러리가 필요합니다.
CSV (`.csv`)	`CSVLoader`	CSV 파일을 로드하여 행(row)당 하나의 `Document`를 생성합니다.
JSON / JSONL	`JSONLoader`, `JSONLinesLoader`	파일 내의 특정 필드를 추출하도록 설정할 수 있습니다.
Markdown (`.md`)	`UnstructuredMarkdownLoader`	`unstructured` 라이브러리가 필요합니다.
HTML (`.html`)	`UnstructuredHTMLLoader`	`unstructured` 라이브러리가 필요합니다.
Word 문서 (`.docx`)	`UnstructuredWordDocumentLoader`	`unstructured` 라이브러리가 필요합니다.
PowerPoint (`.pptx`)	`UnstructuredPowerPointLoader`	`unstructured` 라이브러리가 필요합니다.
다양한 형식 (Fallback)	`UnstructuredFileLoader`	**PDF, DOCX, HTML 등** 미리 정의된 형식이 없는 문서를 처리할 수 있습니다. `unstructured` 라이브러리가 필수입니다.

주요 고려 사항

- `UnstructuredLoader` **계열:** PDF, Word, HTML 등 복잡한 문서 구조를 가진 파일을 처리할 때 유용하며, 이 로더를 사용하려면 `unstructured` 라이브러리가 반드시 설치되어 있어야 합니다.
- **설치 요구 사항:** 로더 클래스에 따라 해당 기능을 제공하는 **추가 Python 라이브러리**를 설치해야 합니다 (예: `PyPDFLoader` 를 사용하려면 `pip install pypdf`).

CSV 문서 (CSVLoader)

실습 코드 : https://github.com/whitepaper82/langchain-example/blob/main/textloader.py

`langchain_community` 의 `CSVLoader` 는 CSV 파일을 읽어, 파일의 각 행(Row)을 개별적인 `Document` 객체로 변환하는 도구입니다.

- **동작 방식**: CSV의 한 줄이 하나의 문서가 되며, 결과는 이러한 문서 객체들의 리스트로 반환됩니다.
- **코드 설명**: 예제는 `cp949` 인코딩을 사용하여 파일을 로드했으며, 실행 결과인 **143**은 변환된 문서의 총 개수(CSV 데이터의 행 개수)를 의미합니다.

Python

```python
from langchain_community.document_loaders.csv_loader import CSVLoader

loader = CSVLoader(file_path='./data/주택금융관련_지수_20160101.csv',
encoding='cp949')
data = loader.load()

print(len(data))
```

PDF 문서

실습 코드 : https://github.com/whitepaper82/langchain-example/blob/main/pdfloader.py

1. PyPDFLoader

`langchain_community` 패키지에서 제공하는 `PyPDFLoader` 를 사용하여 PDF 파일에서 텍스트를 추출합니다. 이 명령을 사용하려면 `pypdf` 라이브러리를 먼저 설치해야 합니다.

Auto

```
pip install -q pypdf
```

`PyPDFLoader` 는 PDF 파일을 읽어 **각 페이지를 하나의** `Document` **객체**로 변환하는 도구입니다.

- **동작**: 파일을 로드하면 **페이지 단위**로 분할된 문서 리스트가 반환되며, `len()` 함수로 총 페이지 수를 확인할 수 있습니다.

Auto (Python)

```python
from langchain_community.document_loaders import PyPDFLoader

pdf_filepath = './data/000660_SK_2023.pdf'
loader = PyPDFLoader(pdf_filepath)
pages = loader.load()

print("PyPDFLoader len: ",len(pages))
print(pages[0].page_content)
```

2. UnstructuredPDFLoader

`UnstructuredPDFLoader` 는 **`unstructured` 라이브러리**를 활용해 PDF를 분석합니다. 내부적으로는 텍스트를 여러 '요소(elements)'로 쪼개서 인식하지만, **기본 설정에서는 이를 모두 합쳐서 하나의 긴 텍스트로 반환**합

니다.

Auto

```
pip install unstructured[pdf] pillow pi_heif
```

`UnstructuredPDFLoader` 를 사용하여 지정된 PDF 파일에서 페이지를 로드하고, 로드된 페이지의 개수를 출력합니다.

Auto (Python)

```python
from langchain_community.document_loaders import UnstructuredPDFLoader

pdf_filepath = './data/000660_SK_2023.pdf'

# 전체 텍스트를 단일 문서 객체로 변환
loader = UnstructuredPDFLoader(pdf_filepath)
pages = loader.load()

print("UnstructuredPDFLoader len: ",len(pages))  #output
```

UnstructuredPDFLoader 실행에 필요한 필수 설치 패키지

LangChain 1.0 기준에서 **UnstructuredPDFLoader**는 `unstructured` 패키지를 기반으로 하는데, 이 패키지는 PDF, 이미지 처리 등을 위해 여러 종속 패키지를 요구합니다.

✓ 필수 설치

아래 명령을 설치하면 문제 없이 작동합니다.

Auto (CSS)

```
pip install "unstructured[pdf]" unstructured[pdf-inference] pdfminer.six
pip install pi_heif
pip install pillow
```

왜 `pi_heif` 가 필요한가?

`unstructured` 라이브러리는 PDF 내부에 포함된 이미지(특히 HEIC/HEIF 포맷)를 처리하기 위해 `pi_heif` 모듈을 사용합니다.

따라서 PDF 안에 이미지가 없어도 라이브러리가 import 단계에서 `pi_heif` 를 요구합니다.

정리 — UnstructuredPDFLoader 사용 시 필요한 것

기능	필요한 패키지
PDF 텍스트 추출	`unstructured`, `pdfminer.six`
이미지 포함 PDF 처리	`pillow`, `pi_heif`
정확한 PDF 레이아웃 분석	`unstructured[pdf]`
OCR(옵션)	`unstructured[pdf-inference]`, `pytesseract`

최소 설치 (실습에서 권장)

최소한 다음만 설치해도 대부분 작동합니다:

Auto

```
pip install "unstructured[pdf]"
pip install pi_heif
```

권장 설치(가장 안정적)

너무 많이 설치되므로 실습에서는 사용하지 않음.

Auto

```
pip install "unstructured[all]"
pip install pi_heif
```

`PyPDFLoader` VS `UnstructuredPDFLoader`

두 코드의 가장 큰 차이점은 **PDF 파일을 '어떤 단위'로 나누어 가져오느냐**에 있습니다.

핵심 요약:

1. `PyPDFLoader` : **페이지(Page) 단위**로 문서를 나눕니다. (100페이지 PDF → 100개의 Document)
2. `UnstructuredPDFLoader` : 기본 설정 시 문서 전체(Whole)를 하나의 텍스트 덩어리로 합칩니다. (100페이지 PDF → 1개의 Document)

---

1. PyPDFLoader

- **작동 방식**: PDF의 물리적인 페이지를 기준으로 문서를 분리합니다.
- **결과 (`pages`)**: 리스트의 원소 개수(`len`)는 **PDF의 실제 페이지 수**와 같습니다.

- **장점**: 처리가 빠르고, 각 데이터가 몇 페이지에 있었는지 메타데이터(`page`)로 명확히 알 수 있습니다.
- **비유**: 책을 한 페이지씩 찢어서 각각 보관하는 것과 같습니다.

2. UnstructuredPDFLoader

- **작동 방식**: `unstructured` 라이브러리를 사용하여 텍스트의 구조를 분석합니다. 기본 모드 (`mode="single"`)에서는 페이지 구분을 무시하고 텍스트를 이어 붙입니다.
- **결과 (`pages`)**: 리스트의 원소 개수(`len`)는 보통 **1개**입니다. (전체 텍스트가 담긴 거대한 Document 1개)
- **장점**: 페이지 경계에 걸친 문장(문단이 다음 페이지로 이어지는 경우)이 끊기지 않고 자연스럽게 연결됩니다.
- **비유**: 책의 모든 내용을 타이핑 쳐서 하나의 긴 두루마리 휴지에 적는 것과 같습니다

3. OnlinePDFLoader

실습 코드 : https://github.com/whitepaper82/langchain-example/blob/main/pdfloader.py

"Transformers" 논문(https://arxiv.org/pdf/1706.03762.pdf)을 로드하여 페이지 내용을 추출하고, 로드된 페이지 수와 첫 페이지의 내용 일부를 출력하는 과정을 처리합니다. `langchain_community` 라이브러리의 `OnlinePDFLoader` 클래스를 사용합니다.

Auto (Python)

```python
# Transformers 논문을 로드
loader = OnlinePDFLoader("https://arxiv.org/pdf/1706.03762.pdf")
pages = loader.load()

print("\nOnlinePDFLoader len : ",len(pages))

#로드된 문서 객체의 내용을 출력하여 확인합니다. 첫 페이지의 텍스트 내용 중 처음 1000자를 출력합니다.
print(pages[0].page_content[:1000])
```

참고 문서 (샘플코드)

랭체인(LangChain) 입문부터 응용까지 https://wikidocs.net/book/14473