Day 1 - LangChain Fundamentals and Data Loader

Introduction to LangChain for Beginners

Welcome to this beginner-friendly guide to LangChain! We'll explore the basics, setup, and how to work with different data loaders.

What is LangChain?

LangChain is a framework for building applications with large language models (LLMs). Think of it as a toolkit that helps you connect AI models to different data sources and applications.

LangChain is like a bridge between AI models and your data. It helps you:

- Talk to AI models
- Feed your own data to these models
- Build useful applications (like chatbots that know about your documents)

Setting Up LangChain

Let's start with how to set up LangChain on your machine:

```
# Install LangChain and OpenAI packages
pip install langchain
pip install google-generativeai



pip install pypdf
pip install bs4
pip install unstructured
pip install request
pip install python-docx2txt
pip install unstructured[pptx]
pip install jq
pip install markdown
```

```
#Another Libraries
pip install python-dotenv
```

That's it! Now you're ready to start using LangChain.

Core Concepts in LangChain

Before diving into data loaders, let's understand the basic building blocks:

Component	What it does	Real-world example
LLMs	AI models that generate text	ChatGPT, Llama, etc.
Prompts	Templates for communicating with LLMs	Asking a question in a specific format
Chains	Link multiple components together	Reading data, summarizing it, then answering questions
Document Loaders	Tools to bring in outside data	Reading PDFs, websites, or databases
Indexes	Efficient storage of data	Creating a searchable index of documents

Data Loaders in LangChain?

Data loaders are the foundation of any LangChain application. They allow you to import various types of data into your LLM-powered applications. Think of them as adapters between your files and the AI model.

Anatomy of a Data Loader

Each LangChain data loader typically follows this pattern:

1. **Initialization:** Set up the loader with parameters like file path
2. **Loading:** Call the `load()` method to retrieve documents
3. **Processing:** Convert the source into Document objects with text and metadata

Common Data Loader Properties

- **Document objects:** The standard output format containing:
- `page_content`: The actual text content

- `metadata`: Source information, page numbers, etc.

Best Practices for Data Loaders

- **Error handling:** Always wrap loaders in try/except blocks to handle file access issues
- **Metadata enrichment:** Add source information to help trace data origins
- **Chunking strategy:** Consider how data will be split after loading
- **Data cleaning:** Remove irrelevant content like headers/footers when possible
- **Performance:** For large datasets, use streaming or lazy loading techniques

1. How to Load PDF Files

```
from langchain.document_loaders import PyPDFLoader

# Load a PDF file
loader = PyPDFLoader("my_document.pdf")
documents = loader.load()

# Print the first page content
print(documents[0].page_content[:100])
```

This code reads a PDF file called "my_document.pdf" and prints the first 100 characters from the first page.

2. How to Load Webpages

```
from langchain.document_loaders import WebBaseLoader

# Load content from a webpage
loader = WebBaseLoader("https://www.example.com")
documents = loader.load()

# Print the page content
print(f"Loaded {len(documents)} document(s)")
print(documents[0].page_content[:100])
```

This example gets content from "example.com" and prints the beginning of the page content.

3. How to Load CSV Data

```
from langchain.document_loaders.csv_loader import CSVLoader

# Load a CSV file
loader = CSVLoader("my_data.csv")
documents = loader.load()

# Each row becomes a document
print(f"Loaded {len(documents)} rows from CSV")
print(documents[0].page_content)
```

This loads a CSV file where each row becomes a separate document that can be processed.

4. How to Load Data from Directory

```
from langchain.document_loaders import DirectoryLoader

# Load all text files in a directory
loader = DirectoryLoader("./my_documents/", glob="**/*.txt")
documents = loader.load()

print(f"Loaded {len(documents)} documents from directory")
```

This example loads all text files from a folder called "my_documents" and its subfolders.

5. How to Load HTML Data

```
from langchain.document_loaders import BSHTMLLoader

# Load an HTML file
loader = BSHTMLLoader("webpage.html")
documents = loader.load()
```

```
print(f"Loaded HTML content with {len(documents[0].page_content)} char
acters")
```

This code loads and parses HTML content from a local file.

6. How to Load JSON Data

```
from langchain.document_loaders import JSONLoader
import json

# Define a simple function to extract the content
def extract_content(record):
    return record["content"]

# Load a JSON file
loader = JSONLoader(
    file_path="data.json",
    jq_schema='.[]',
    content_key="content"
)
documents = loader.load()

print(f"Loaded {len(documents)} items from JSON")
```

This example loads data from a JSON file, extracting content from each item.

7. How to Load Markdown Data

```
from langchain.document_loaders import UnstructuredMarkdownLoader

# Load a markdown file
loader = UnstructuredMarkdownLoader("readme.md")
documents = loader.load()

print(f"Loaded markdown with {len(documents[0].page_content)} characte
rs")
```

This loads and processes a Markdown file, converting it to plain text content.

8. How to Load Microsoft Office Data

```
from langchain.document_loaders import Docx2txtLoader
from langchain.document_loaders import UnstructuredPowerPointLoader

# Load a Word document
word_loader = Docx2txtLoader("document.docx")
word_docs = word_loader.load()

# Load a PowerPoint file
ppt_loader = UnstructuredPowerPointLoader("presentation.pptx")
ppt_docs = ppt_loader.load()

print(f"Loaded Word doc with {len(word_docs[0].page_content)} character
s")
print(f"Loaded PowerPoint with {len(ppt_docs)} slides")
```

This example shows how to load both Word documents and PowerPoint presentations.

9. How to Write a Custom Document Loader

```
from langchain.document_loaders.base import BaseLoader
from langchain.docstore.document import Document

class MyCustomLoader(BaseLoader):
    def __init__(self, file_path):
        self.file_path = file_path

    def load(self):
        # Custom loading logic
        with open(self.file_path, "r") as f:
            text = f.read()

        # Process the text as needed
        processed_text = text.upper()  # Just an example - converts to upperc
ase
```

```
        # Return as a Document object
        metadata = {"source": self.file_path}
        return [Document(page_content=processed_text, metadata=metadat
a)]

# Use your custom loader
custom_loader = MyCustomLoader("my_special_format.txt")
documents = custom_loader.load()
print(documents[0].page_content[:100])
```

This example shows how to create your own loader for specialized file formats or data sources.

Summary

In this guide, we've covered:

- How to set up LangChain.
- The core concepts of LangChain
- Various data loader types with code examples
- How to create a custom data loader

LangChain is a powerful tool for working with AI language models and connecting them to various data sources. The data loaders we've explored are just the beginning of what you can do with this framework!