Extracting Context from PDFs and Web Content

1. PyMuPDF
   * Purpose: Extract text from PDFS
   * Implementation Code
     + def extract\_text\_pymupdf(pdf\_path):
     + """Extracts text from a structured (non-scanned) PDF."""
     + doc = fitz.open(pdf\_path)
     + text = "\n".join([page.get\_text("text") for page in doc])
     + return text
   * Reference: <https://pymupdf.readthedocs.io/en/latest/index.html>
2. PDFPulmber & Pillow
   * Purpose: Extract content from tables and images
   * Implementation Code
   * Tables:
     + def extract\_tables\_pdf(pdf\_path):
     + """Extracts tables from a PDF and converts them into DataFrames."""
     + tables = []
     + with pdfplumber.open(pdf\_path) as pdf:
     + for page in pdf.pages:
     + table = page.extract\_table()
     + if table:
     + df = pd.DataFrame(table[1:], columns=table[0]) # First row as headers
     + tables.append(df)
     + return tables
   * Images:
     + def extract\_images\_pdf(pdf\_path, save\_folder="images"):
     + """Extracts images from a PDF and saves them locally."""
     + images = []
     + if not os.path.exists(save\_folder):
     + os.makedirs(save\_folder)
     + with pdfplumber.open(pdf\_path) as pdf:
     + for i, page in enumerate(pdf.pages):
     + img\_list = page.images
     + for img in img\_list:
     + img\_bytes = page.to\_image().original
     + img\_filename = f"{save\_folder}/pdf\_page\_{i+1}.png"
     + with open(img\_filename, "wb") as f:
     + f.write(img\_bytes)
     + images.append(img\_filename)
     + return images
   * Reference: <https://github.com/jsvine/pdfplumber>
3. Beautiful Soup
   * Purpose: Extracting text from webpages
   * Implementation Code
     + Text
       - def extract\_text\_webpage(url):
       - """Extracts text from an article webpage."""
       - response = requests.get(url)
       - soup = BeautifulSoup(response.text, "html.parser")
       - text = "\n".join([p.get\_text(strip=True) for p in soup.find\_all("p")])
       - return text
     + Tables
       - def extract\_tables\_webpage(url):
       - """Extracts tables from a webpage and converts them into DataFrames."""
       - response = requests.get(url)
       - soup = BeautifulSoup(response.text, "html.parser")
       - tables = []
       - for table in soup.find\_all("table"):
       - rows = []
       - for row in table.find\_all("tr"):
       - cells = [cell.get\_text(strip=True) for cell in row.find\_all(["td", "th"])]
       - rows.append(cells)
       - if rows:
       - df = pd.DataFrame(rows[1:], columns=rows[0]) # First row as headers
       - tables.append(df)
       - return tables
     + Images
       - def extract\_images\_webpage(url, save\_folder="images"):
       - """Extracts images from a webpage and saves them locally."""
       - response = requests.get(url)
       - soup = BeautifulSoup(response.text, "html.parser")
       - images = []
       - if not os.path.exists(save\_folder):
       - os.makedirs(save\_folder)
       - for img in soup.find\_all("img"):
       - img\_url = img.get("src")
   * Reference: https://pypi.org/project/beautifulsoup4/

Combined Extraction Pipeline

* Analyzes the type of document provided and uses the appropriate context extraction method.
* Code
* import requests
* from bs4 import BeautifulSoup
* import pdfplumber
* import fitz # PyMuPDF
* import pandas as pd
* import os
* # 1️ Extract Text from PDFs and Web Pages
* def extract\_text(source):
* """Extracts text from PDF or web URL."""
* if source.endswith(".pdf"):
* return extract\_text\_pymupdf(source) # Extract from PDF
* elif source.startswith("http"):
* return extract\_text\_webpage(source) # Extract from web page
* else:
* raise ValueError("Unsupported file type. Provide a PDF or URL.")
* # 2️ Extract Tables From for PDFs and Web Pages
* def extract\_tables(source):
* """Extracts tables from a PDF or web URL and converts them into DataFrames."""
* if source.endswith(".pdf"):
* return extract\_tables\_pdf(source) # Extract from PDF
* elif source.startswith("http"):
* return extract\_tables\_webpage(source) # Extract from web page
* else:
* raise ValueError("Unsupported file type. Provide a PDF or URL.")
* # 3️ Extract Images Separately for PDFs and Web Pages
* def extract\_images(source, save\_folder="images"):
* """Extracts images from PDFs or web pages and saves them locally."""
* if source.endswith(".pdf"):
* return extract\_images\_pdf(source, save\_folder) # Extract from PDF
* elif source.startswith("http"):
* return extract\_images\_webpage(source, save\_folder) # Extract from web page
* else:
* raise ValueError("Unsupported file type. Provide a PDF or URL.")

Providing the Extracted Context to the Generative AI model

* Code extract all context
  + def extract\_all(sources):
  + """Extracts text, tables, and images from multiple PDFs and web pages."""
  + combined\_text = []
  + combined\_tables = []
  + combined\_images = []
  + source\_references = [] # Store source details
  + for source in sources:
  + text = extract\_text(source)
  + tables = extract\_tables(source)
  + images = extract\_images(source)
  + combined\_text.append(text)
  + combined\_tables.extend(tables)
  + combined\_images.extend(images)
  + source\_references.append(source) # Store the source reference
  + return combined\_text, combined\_tables, combined\_images, source\_references
  + def generate\_lesson\_from\_extracted\_data(sources):
  + """Extracts data from multiple sources and integrates it into an AI-generated lesson."""
  + # Extract all data
  + all\_texts, all\_tables, all\_images, all\_sources = extract\_all(sources)
  + # Convert tables to Markdown format
  + tables\_markdown = table\_to\_markdown(all\_tables)
  + # Combine all extracted text
  + combined\_text = "\n\n".join(all\_texts)[:4000] # Limit to 4000 characters for AI model constraints
  + # List extracted images
  + image\_references = "\n".join([f"- {img}" for img in all\_images]) if all\_images else "No images extracted."
  + # Prepare AI model input
  + prompt = f"""
  + The following educational content has been extracted from multiple sources:
  + ### 📄 Extracted Text:
  + {combined\_text}
  + ### 📊 Extracted Tables:
  + {tables\_markdown}
  + ### 🌐 Source Webpages & Documents:
  + {', '.join(all\_sources)}
  + ### 🖼 Extracted Images:
  + {image\_references}
  + Generate a structured educational lesson based on this content, ensuring the tables and images are properly referenced in the lesson output.
  + """
  + # Load AI Model (Vicuna, Llama 2, GPT-based)
  + generator = pipeline("text-generation", model="meta-llama/Llama-2-7b-chat-hf", device="cuda")
  + # Generate AI response
  + response = generator(prompt, max\_length=2000)
  + return response[0]["generated\_text"]
* Reference: <https://docs.vllm.ai/en/latest/models/supported_models.html>

Save lesson plan and the Documents as a JSON File

* Code
  + import json
  + def save\_lesson\_to\_file(lesson\_data, filename="lesson.json"):
  + """Saves the generated lesson to a JSON file."""
  + with open(filename, "w", encoding="utf-8") as f:
  + json.dump(lesson\_data, f, indent=4)
  + def load\_lesson\_from\_file(filename="lesson.json"):
  + """Loads a saved lesson from a JSON file."""
  + try:
  + with open(filename, "r", encoding="utf-8") as f:
  + return json.load(f)
  + except FileNotFoundError:
  + return None # Return None if no lesson exists
  + # Example: Save lesson
  + lesson\_data = {
  + "text": lesson\_output, # The AI-generated lesson
  + "tables": tables\_markdown, # Extracted tables in Markdown
  + "images": all\_images, # List of extracted images
  + "sources": all\_sources # List of PDFs/Web Pages
  + }
  + save\_lesson\_to\_file(lesson\_data)
  + # Example: Load and Edit Lesson
  + loaded\_lesson = load\_lesson\_from\_file()
  + if loaded\_lesson:
  + print("\nPreviously Saved Lesson:\n", loaded\_lesson["text"])
  + Resource <https://www.geeksforgeeks.org/reading-and-writing-json-to-a-file-in-python/>