**📄 Document Outline Extractor – Challenge 1A (Connecting the Dots)**

**🔍 Project Overview**

This solution is developed for **Round 1A** of Adobe’s *Connecting the Dots* hackathon. The challenge demands a robust, deterministic, and offline-capable system that can extract a structured outline from a PDF document. The extracted output must include the document’s **title**, along with **H1**, **H2**, and **H3** headings—each annotated with its **level** and **page number**—and delivered in a clean **JSON format**.

The goal is to make sense of a PDF like a machine would, laying the groundwork for intelligent document interaction, semantic search, and insight generation.

**🧠 Approach**

The solution adopts a **rule-based, multi-factor heuristic** approach to analyze the structure of PDF documents. It does **not rely on any machine learning models**, ensuring compliance with the strict constraints of no external dependencies, offline operation, and low runtime overhead.

**Key Components of the Logic:**

* **Title Extraction**:  
  The system first attempts to extract the document title using embedded metadata. If unavailable or unreliable, it falls back to analyzing prominent, high-font-size text near the top of the first few pages, selecting candidates based on size, boldness, and positioning.
* **Heading Classification (H1, H2, H3)**:  
  Headings are identified by analyzing:
  + **Font size** and **font weight** (bold vs. normal)
  + **Capitalization** and **text length**
  + **Vertical position** on the page (Y-axis heuristic)

These factors are combined into a scoring system that classifies text blocks into H1, H2, or H3 categories.

* **Structured Outline Generation**:  
  Each detected heading is tagged with its **hierarchical level**, **text**, and **corresponding page number**, and then compiled into a structured JSON object.
* **Batch Processing**:  
  The system can process **multiple PDFs** in a single run, automatically scanning a designated input directory and saving results in a corresponding output folder.
* **Multilingual Compatibility**:  
  While optimized for English, the solution supports **Unicode text**, making it compatible with basic multilingual documents.
* **No Randomness**:  
  Outputs are **fully deterministic**—identical inputs will always produce identical outputs, aiding reproducibility and debugging.

**📚 Libraries Used**

This solution uses the following **Python libraries**:

* **[PyMuPDF (fitz)]**: For accessing and parsing PDF content, including layout, fonts, text blocks, and positioning. This is the primary engine for reading and analyzing PDF files.
* **[pdfplumber]** *(optional or auxiliary)*: Useful for validating extracted content or refining font-based analysis.
* **[Pillow]**: Used to support image handling and compatibility where PDF pages may contain embedded image elements (though not required for core functionality).
* **Python Standard Library**: Includes os, json, pathlib, re, and collections, used for file handling, pattern matching, and output serialization.

❗ **No machine learning models**, cloud APIs, or internet-based services are used. The entire solution runs **offline on CPU**, satisfying all challenge constraints.

**⚙️ How to Build and Run the Solution**

Although the official execution will be handled using predefined commands during evaluation, the following steps outline how one can **build and test the solution locally or in a containerized environment**.

**Option 1: Using Docker (Recommended)**

1. **Build the Docker image**:
2. docker build -t pdf-structure-extractor .
3. **Run the container**:
4. docker run --rm -v "$(pwd)/input:/app/input:ro" -v "$(pwd)/output:/app/output" --network none pdf-structure-extractor
   * Place all PDF files inside the input/ directory.
   * The resulting JSON files will be saved to the output/ directory.

**Option 2: Running Locally (Without Docker)**

1. **Install Python 3.8+**
2. **Install dependencies**:
3. pip install -r requirements.txt
4. **Place PDF files** into the input/ directory.
5. **Execute the processor**:
6. python main.py
7. **Review results** inside the output/ directory.

**📦 Output Format**

Each output is a valid JSON file containing the following structure:

{

"title": "Extracted Document Title",

"outline": [

{ "level": "H1", "text": "Heading One", "page": 1 },

{ "level": "H2", "text": "Subsection Title", "page": 2 },

{ "level": "H3", "text": "Nested Section", "page": 3 }

]

}

**✅ Compliance Summary**

| **Constraint** | **Status** | **Notes** |
| --- | --- | --- |
| Max Pages (≤ 50) | ✅ | Efficiently handles full 50-page documents |
| No ML Model (>200MB) | ✅ | Pure heuristic approach |
| CPU-only (amd64) | ✅ | Fully compatible |
| Offline Runtime | ✅ | Docker uses --network none |
| Output Format = Valid JSON | ✅ | Matches schema exactly |
| Multiple PDFs in One Run | ✅ | Batch processing supported |

**🏁 Final Notes**

This solution offers a **transparent, fast, and reproducible** PDF structure extraction tool tailored precisely for the hackathon’s constraints. It’s easily extensible, auditable, and ready for deployment in any environment that values consistency, compliance, and performance.

Let me know if you need this converted into a README.md file for submission.