Task 1: Research & Tool Selection

Problem Statement :

The challenge involves extracting a specific table from over 100 PDF documents, which may appear in different formats, span multiple pages, and be handwritten, scanned, or digitally structured. The objective is to identify and extract this table, handle inconsistent formatting, multi-page tables, and OCR challenges, and store the data in JSON format, ensuring scalability and generalization to unseen documents. The problem space is complex, particularly for tender documents with 100+ pages.

Problem Breakdown :

Stage 1: Digitally Structured PDFs

- **Description**: PDFs with a selectable text layer, where tables are embedded as structured data.
- **Sub-Problems**:
    - **Format Variability**: Tables may use borders, whitespace, or no visual cues (e.g., text alignment only).
    - **Structural Ambiguity**: Inconsistent headers (e.g., "Item" vs. "ITEM #") or merged cells.
    - **Finding Page ranges** : Finding till where the table ranges and extracting and merging them

Stage 2: Scanned and Handwritten Tables with OCR

- **Description**: PDFs as images (scanned) or with handwritten content, requiring text extraction.
- **Sub-Problems**:
    - **Document Types**: Scanned images lack a text layer; handwritten tables have variable legibility.
    - **Format Variability**: Borders or whitespace may be distorted in scans; handwritten tables may lack structure.
    - **Accuracy Challenge**: Potential errors in text recognition (e.g., "1" vs. "I") or unreadable handwriting.

Stage 3: Edge Cases (Multi-Page Tables and Unclear Boundaries)

- **Description**: Tables spanning multiple pages or with ambiguous boundaries.
- **Sub-Problems**:
    - **Structural Ambiguity**: Tables split across pages, with headers only on the first page or inconsistent column counts.
    - **Format Variability**: No visual cues (e.g., borders) or unclear table boundaries blending with surrounding text.
    - **Scalability Challenge**: Processing multi-page documents efficiently.

Stage 4 : Error Handling

- **Priority**: Flagged errors over incorrect data.
- **Examples:** Missing tables, unreadable content, or ambiguous structures.

List of various methods to solve the problem :

2 Potential Ways could be there based on Identifying PDF type

1. **Using tabula-py with OCR technique:**

   Tabula-py processes them together by treating all PDFs as a sequence of pages, applying text extraction for digital PDFs and OCR (when enabled) for scanned and handwritten PDFs without distinguishing the format up front.

2. **A multi-stage pipeline using Camelot, Tesseract :**

   Identifying PDF type (Digital or Scanned/Handwritten) and processing each with different methods

Keeping LLM Integration Optional in both ways as it has a High Risk of Hallucinations in numeric data and High latency/cost at scale

Comparison of Both Ways:

1. Digital PDF Extraction :

Aspect	Camelot + PyMuPDF	Tabula-py + OCR
Border Detection	Lattice mode uses Hough transforms for line detection	Uses PDFBox's built-in border detection
Borderless Tables	Stream mode analyzes text alignment	Requires manual adjustment of spreadsheet parameter
Encoding Handling	Direct access to PDF's character encoding	Depends on PDFBox's auto-detection

2. Multi-Page Table Handling

Aspect	Camelot + PyMuPDF	Tabula-py + OCR
Continuity Detection	Header similarity analysis + row content continuity checks	Treats each page as independent table
Auto-Merging	Merges tables across pages if headers match	Requires manual page range specification
Row Splitting	Uses text coordinates to detect partial rows	Often splits rows at page breaks
Header Propagation	Carries headers to subsequent pages automatically	Repeats headers only if present in PDF

3. Performance & Scalability

Aspect	Method 2	Method 1
Speed	2-5 sec/page (CPU-bound)	3-7 sec/page (JVM overhead)
Memory	~300MB per process	~800MB (Java heap)
Parallelization	Easy with Python multiprocessing	Limited by JVM constraints
100-PDF Throughput	~45 mins (parallel)	~2 hours (sequential)

What will I try and why :

I would choose Method 2 because it seems more scalable and applicable to various documents.

Pipeline Overview :

1. **Document Classification**:
    - Use **PyMuPDF** to check if the PDF is digital (text layer) or scanned (image-based).
    - Route processing based on the result.
2. **Digital PDF Processing**:
    - **Primary Tool**: **Camelot** (lattice/stream mode) for table extraction.
        - Lattice mode for bordered tables.
        - Stream mode for borderless tables.
    - **Fallback**: Use **PDFPlumber** to extract text and reconstruct tables using spatial coordinates.
3. **Scanned/Handwritten PDF Processing**:
    - **Preprocessing**: Use **OpenCV** for grayscale conversion, binarization, and deskewing.
    - **OCR**: Use **Tesseract** to extract text and layout data.
    - **Table Reconstruction**: Detect columns and rows using whitespace and alignment.
4. **Multi-Page Table Handling**:
    - Merge tables across pages by matching headers and checking row continuity.
    - Use coordinate tracking to handle split rows.
    - Finding pages till where the tables range and merging them
5. **Validation & Error Handling**:
    - Validate table structure (consistent columns, headers).
    - Flag unsupported formats (e.g., handwritten tables).
    - Return error messages instead of incorrect data.
6. **JSON Output**:
    - Convert extracted tables into structured JSON format.

- Include metadata (e.g., page numbers, confidence scores).

Key Libraries

- **PyMuPDF**: Document classification and text extraction.
- **Camelot**: Table extraction from digital PDFs.
- **OpenCV**: Image preprocessing for scanned PDFs.
- **Tesseract**: OCR for scanned/handwritten content.
- **Regex/Scikit-learn**: Data validation and error handling.

Strengths

- Handles **multi-page tables** seamlessly.
- Supports **mixed document types** (digital, scanned, handwritten).
- Includes **validation and error handling** for robust results.
- Outputs structured JSON for easy integration.