📄 PDF Statement Parser — Project Documentation

1️⃣ Project Title

Credit Card PDF Statement Parser

2️⃣ Objective

The main objective of this project is to develop a Python-based PDF parsing system capable of automatically reading and extracting structured information from credit card statements issued by major banks.

The system is designed to handle real-world PDF formats, including both **text-based** and **scanned image-based** statements, with optional OCR (Optical Character Recognition) support.

3️⃣ Scope

The system supports five major credit card issuers:

- Chase
- American Express (Amex)
- Citi
- HDFC Bank
- SBI Card

It extracts five key data points:

- Card Variant
- Card Last 4 Digits
- Billing Cycle
- Payment Due Date
- Total/New Balance

Handles both:

- Text-based PDFs using pdfplumber
- Scanned image PDFs using pytesseract OCR

4️⃣ Technologies Used

Technology	Purpose
Python 3.x	Programming language
pdfplumber	Extract text from text-based PDFs
pdf2image	Convert PDF pages into images for OCR
pytesseract	Perform OCR on scanned statements

Technology	Purpose
Pillow (PIL)	Image processing support
rich	Display JSON output in colorized format
tkinter	File picker for selecting PDF interactively

5️⃣ System Requirements

Hardware:

- 4 GB RAM (minimum)
- Intel i3 processor or above
- 500 MB storage space

Software:

- Python 3.9 or above
- Poppler (for pdf2image)
- Tesseract OCR (for scanned PDFs)

6️⃣ Folder Structure

Plaintext

```
pdf_pro/
│
├── parser.py                # Main entry point
│
├── /parsers/                # Bank-specific parser modules
│   ├── chase_parser.py
│   ├── amex_parser.py
│   ├── citi_parser.py
│   ├── hdfc_parser.py
│   └── sbi_parser.py
│
├── requirements.txt         # Python dependencies
└── README.md                # Documentation
```

7️⃣ Methodology / Workflow

- Step 1: PDF Input

User selects a PDF file interactively via a file dialog (no need to manually enter path).

- Step 2: Text Extraction

The system uses pdfplumber to extract textual content.

If no text is found (scanned image), OCR via pytesseract is applied.

- Step 3: Issuer Detection

A simple keyword-based search detects the issuer: "chase", "amex", "citi", "hdfc", or "sbi".

- Step 4: Parsing

Each issuer has a dedicated parser (e.g., parse_chase(), parse_amex()) that extracts and returns structured information.

- Step 5: Output Formatting

Extracted data is displayed as a formatted JSON using rich.print_json().

8 Flowchart

Code snippet

graph TD

A[Start Application] --> B(Upload PDF)

B --> C{Extract Text}

C -- Text Found --> E[Detect Issuer Type]

C -- No Text Found --> D[Apply OCR]

D --> E

E --> F[Parse Data Fields]

F --> G[Display JSON Output]

G --> H(End)

9 Example Output

Input:

chase_sample.pdf

Output:

JSON

{

"card_variant": "REWARDS",

"card_last4": "1234",

"billing_cycle": "12/03/18 - 01/01/19",

"payment_due_date": "01/25/2019",

"new_balance": "1,245.00",

"issuer": "chase",

"used_ocr": false,

"raw_text_snippet": "Manage your account online: ..."

}

🔟 Error Handling

- **File not found** → displays an error and exits
- **Unreadable PDF** → automatically switches to OCR
- **Unknown issuer** → returns "issuer": "unknown"

1️⃣1️⃣ Advantages

✅ Handles both scanned and digital statements

✅ Supports multiple issuers

✅ Modular and easy to extend

✅ Produces structured JSON output

✅ Lightweight and platform-independent

1️⃣2️⃣ Future Enhancements

- Add support for additional issuers (ICICI, Axis, etc.)
- Web-based upload interface using Flask
- Data visualization dashboard
- Integration with expense tracking or finance management apps

1️⃣3️⃣ Conclusion

This project demonstrates a robust, extendable, and user-friendly approach to automating PDF data extraction for financial statements.

It effectively bridges the gap between unstructured document formats and structured data analysis, enabling faster processing, automation, and reporting.