

Credit Card Statement Parser

Submission Document

1 Project Objective

The objective of this project was to build a Python-based parser capable of extracting 5 key data points from credit card statements across 5 different issuers.

- **Data Points Extracted:**

- Total Balance
- Payment Due Date
- Minimum Payment
- Last 4 Digits of Card
- Statement Closing Date

- **Core Requirements:**

- Handle real-world PDF documents with varied layouts and formatting.
- Provide a runnable script to process an entire folder of PDFs and save the consolidated results to JSON and CSV formats.

2 Technology Stack

- **Language:** Python 3

- **Libraries:**

- PyMuPDF (`fitz`): Used as the primary, high-performance library for PDF text extraction.
- `pdfplumber`: Implemented as a robust fallback for text extraction if PyMuPDF fails.
- `re` (Regular Expressions): Used for all pattern-based data parsing and extraction.
- **Standard Library:** `argparse` (for command-line arguments), `json`, `csv`, `pathlib` (for file system handling), and `logging`.

3 Project Structure

The project is organized into a runnable `src` package:

- `requirements.txt`
 - Lists all project dependencies (PyMuPDF, pdfplumber).
- `src/main.py`
 - The main command-line entry point for the application.
 - It iterates through all PDFs in a specified folder, runs the full extraction and parsing pipeline, and then prints and saves the final results.
- `src/credit_parser/__init__.py`
 - Exposes the `extract_text` function for convenience.
- `src/credit_parser/extract.py`
 - Contains the core `extract_text(pdf_path, password="")` function. This module implements the PyMuPDF-first extraction logic with a fallback to pdfplumber.
- `src/credit_parser/parsers.py`
 - Contains the `detect_bank(text)` function to classify text by bank using keyword cues.
 - Holds all bank-specific parsing functions (e.g., `parse_bank_1()`, `parse_bank_2()`, etc.), each returning a standardized dictionary of the 5 fields.
- `src/credit_parser/orchestrator.py`
 - Contains the main `process_pdf(pdf_path)` and `identify_and_parse(text)` functions that connect all the steps: (1) Extract Text → (2) Detect Bank → (3) Route to Correct Parser.

4 How to Run the Solution

4.1 Step 1: Unzip and Set Up Environment

Unzip the project archive and navigate into the folder (e.g., `Credit-Parser/`).

4.2 Step 2: Create and Activate Virtual Environment

macOS/Linux:

```
python3 -m venv .venv
source .venv/bin/activate
```

Windows (PowerShell):

```
py -m venv .venv
.\.venv\Scripts\Activate.ps1
```

4.3 Step 3: Install Dependencies

```
pip install -r requirements.txt
```

4.4 Step 4: Run the Parser

From the project's root folder (the one containing the `src/` directory), run the `main.py` script. You must set the `PYTHONPATH` to `src` so the modules can be found.

```
PYTHONPATH=src python src/main.py <input_folder> --json results.json --csv results.csv
```

Example (processing all PDFs in the current folder):

```
PYTHONPATH=src python src/main.py . --json results.json --csv results.csv
```

4.5 Step 5: Check Outputs

The script will print parsed results to the console and save the complete, consolidated data to `results.json` and `results.csv`.

5 Implementation & Design

The parser operates in a three-stage pipeline for each PDF:

5.1 Step 1: Text Extraction (Robust Extractor)

The `extract_text()` function is designed for reliability.

1. It first attempts to use **PyMuPDF**, which is exceptionally fast and resilient to varied PDF layouts and encodings. It concatenates the text from all pages.
2. If PyMuPDF fails or returns empty text, it automatically triggers its **pdfplumber** fall-back, which uses a different extraction method.
3. If both methods fail, a clear error is raised.
4. The function also supports encrypted PDFs via an optional `password` argument.

5.2 Step 2: Bank Identification (Keyword-Based Routing)

Once the raw text is extracted, the `detect_bank()` function inspects it for unique keywords to classify which bank it belongs to (e.g., "building blocks" for Bank 1, "idfcbank" for Bank 5). The `identify_and_parse()` orchestrator uses this classification to route the text to the correct bank-specific parser.

5.3 Step 3: Data Parsing (Bank-Specific Regex)

This is the core parsing logic. Each bank has its own function (e.g., `parse_bank_1()`) in `parsers.py`.

- **Targeted Patterns:** Each function uses regex patterns tailored to that specific bank's layout.
- **Label Anchoring:** Patterns are "anchored" to labels (e.g., `New Balance:`, `Payment Due Date:`) to reliably find the value, even if spacing or line breaks vary.
- **Normalization:** Extracted values are cleaned (e.g., currency symbols like '₹' are removed, 'CR' suffixes are handled).

- **Standardized Output:** Each parser returns the same dictionary schema, with None as the value if a field (like `minimum_payment`) doesn't exist in a particular statement.

6 Example Output

The final `results.json` file is a list of dictionaries, one for each successfully parsed file.

```

1 [
2   {
3     "bank": "bank1",
4     "total_balance": "1,392.71",
5     "payment_due_date": "1/23/XX",
6     "minimum_payment": "25",
7     "last4": "8907",
8     "statement_closing_date": "12/26/XX",
9     "source_file": "bank1.pdf"
10    },
11   {
12     "bank": "bank5",
13     "total_balance": "3,500",
14     "payment_due_date": "04/02/2024",
15     "minimum_payment": "3,500",
16     "last4": "5396",
17     "statement_closing_date": "17/01/2024",
18     "source_file": "bank5.pdf"
19   }
20 ]

```

Listing 1: Example content of results.json

7 Limitations & Future Work

- **Layout Sensitivity:** The regex parsers are tied to the specific layouts of the sample PDFs. If a bank issues a new statement template, the regex for that bank may need to be updated.
- **Bank Coverage:** The project is limited to the 5 banks as required. It can be extended by adding a new `parse_bank_X()` function and updating the `detect_bank()` keywords.
- **Future Enhancements:**
 - Implement a full suite of unit tests for each parser.
 - Move the bank-specific regex patterns and keywords into a `config.json` file to allow for updates without changing the Python code.