NOTE EXPLICATIVE OF THE SOLUTION

1. Problem Statement and objective:

The objective is to propose a solution for automating data extraction from large PDF files (excluding JSON files for our case). The current input PDF files could be classified as unstructured data. Any PDF files can be categorised into two types: standard (native) PDFs and scanned PDFs (OCR). Our current pdf file is an OCR pdf file.

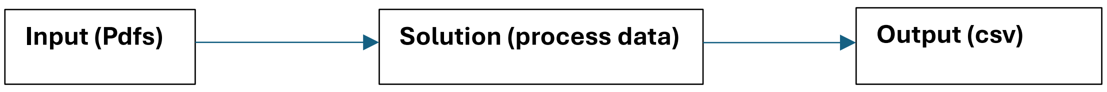
The primary goal is to extract data from any PDF file and present it in a structured format. The output should be organized into five columns, with the first four columns containing data from the schedules of notices of leases and the fifth column titled "Note," which typically spans across all columns in the input file. The final output should be formatted as follows:

- Registration date and plan reference
- Property description
- Date of lease and term
- Lessee’s title
- Note

2. Proposed Solution Overview:

Our proposed approach involves using a Python-based ETL (Extract, Transform, Load) process to extract data and save it as a CSV file. This includes creating a merged table with all CSV tables and leveraging ChatGPT to split the data into the desired columns. The solution involves developing a script that takes a PDF file as input, processes each page to extract the five columns of data from the schedules of notices of leases, and consolidates this data into a single CSV file for better usability.

Solution Design Architecture:

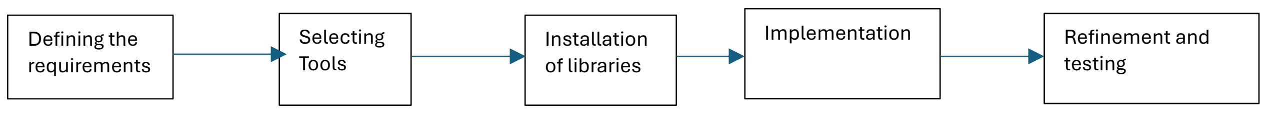


Solution Automation Workflow:

1. **Import PDFs:**
   - Import PDF files from an input folder (e.g., S3 bucket or Google Sheet).
2. **OCR for Scanned PDFs:**
   - Convert scanned PDFs to searchable PDFs using OCRmyPDF.
3. **Extract Table Data:**
   - Use the Tabula Python library to extract key table information from the schedules of notices of leases in the PDF document and save it as a CSV file.

4. **Page-wise CSV Creation:**
    - Save each PDF page as a separate CSV file containing the five columns of data as per the requirements. Include the PDF page number in the CSV file title.
5. **Data Handling and Merging:**
    - Use Pandas to manage data and create structured tables. Concatenate all individual tables into a single merged table to produce a comprehensive output. Save all CSV files, including the main table named "merged_tables," in the output folder.
6. **Column Identification and Splitting:**
    - Utilize ChatGPT to identify the titles for the five columns and split the data accordingly. Save the final table as "merged_tables_chatgpt" in the output folder. This final CSV file will contain all the information extracted from the initial PDF.

3. Building proposed solution:



Step 1: Defining the Requirements.

- Identify the data types in PDFs: Understand that the PDFs may contain plain text, text as part of images (from scanned documents), and tables which could also be embedded within images.
- Output format: Determine that the extracted data should be saved in CSV format, which requires considering how to structure text and tables uniformly.

Step 2: Selecting tools.

- PDF Text and Image Extraction: Used Tabula as it’s appropriate for table with white spaces delimitation. (some others alternatives exist but out-of-scope: Camelot/PyPDF/pdfplumber and PyMuPDF (also known as fitz) to extract both text and images from PDFs.)
- OCR for Scanned Text and Tables: Leverage pytesseract, a Python wrapper for Tesseract-OCR, to convert images (including those of tables) back into editable text or structured data.
- Data Structuring: Use pandas for handling and saving structured data like tables into CSV.

Step 3: Installation of libraries.
- Need to install Python environments and libraries defined above.
- Ensure we have Tesseract-OCR installed on your system.

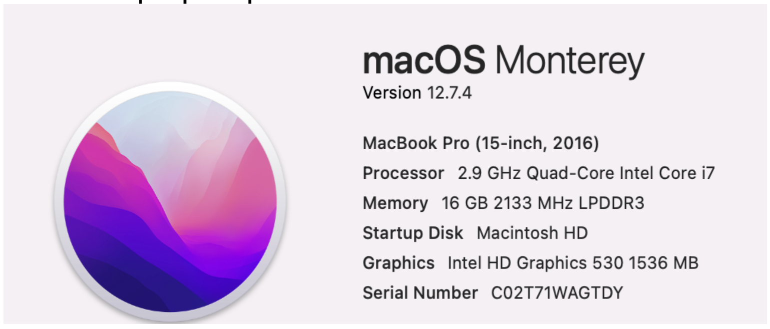
Step 4: Implementation.
- Extract text, images, tables from PDF.
- Process images to extract text and tables (this is the case that we have tables as images following an old document scans).
- Save extracted data to CSV.

Step 5: Refinement and testing.
- Test the application with a variety of PDF files to ensure robustness across different types of data (we can perform this test when we have many pdfs available).
- Optimize OCR settings and handle any image as table.
- Consider implementing error handling and logging for better debugging and maintenance.

4.1 Performance analysis and metrics.

4.1 Script execution time

In the main script, we have incorporated a function to measure and display the script's execution time. For instance, the script's execution time was recorded as 885.03 seconds. This metric serves as an initial measure to evaluate the performance of our solution, considering that the script's execution time is influenced by the available resources (memory/processors). Below is the description of my current laptop's specifications:



macOS Monterey
Version 12.7.4

MacBook Pro (15-inch, 2016)
Processor 2.9 GHz Quad-Core Intel Core i7
Memory 16 GB 2133 MHz LPDDR3
Startup Disk Macintosh HD
Graphics Intel HD Graphics 530 1536 MB
Serial Number C02T71WAGTDY

4.2 Framework to evaluate the proposed solution

One common approach to evaluate data extraction accuracy is to compare the extracted data with a ground truth dataset, where the correct data has been manually verified by human annotator. Here's a proposed framework to evaluate data extraction accuracy using our solution based on tabula-py and a ground truth dataset:

1. **Ground Truth Dataset Preparation**:
    - Manually extract and verify the data from a representative sample of PDF files.
    - Create a ground truth dataset by compiling the correct data into a CSV or Excel file.
    - Create manually a csv table based per pdf page of the document.
2. **Data Extraction using our Solution**:
    - Use main.py script to extract data from the same set of PDF files used for creating the ground truth dataset.
    - Save the extracted data in CSV format.
3. **Data Preprocessing**:
    - Perform any necessary data cleaning and preprocessing steps on both the ground truth dataset and the extracted data.
    - This may include handling missing values, removing whitespace, and formatting column names consistently.
4. **Evaluation Metrics**:
    - Define appropriate evaluation metrics to measure the accuracy of the extracted data.
    - Common metrics for this current scenario include:
        - **Precision**: The proportion of extracted data that is correct.
        - **Recall**: The proportion of correct data that was successfully extracted.
        - **F1-score**: The harmonic mean of precision and recall, providing a balanced measure of accuracy.
        - **Levenshtein Distance**: A measure of the difference between the extracted and ground truth data at the character level.
5. **Evaluation Process**:
    - Implement a function or script to compare the extracted data with the ground truth dataset.
    - Calculate the chosen evaluation metrics by iterating over each row and column, comparing the extracted data with the ground truth.
    - Aggregate the metrics across all rows and columns to obtain an overall accuracy score.
6. **Analysis and Interpretation**:
    - Analyse the evaluation metrics and identify patterns or areas where the data extraction performed well or poorly.
    - Investigate specific cases where the extraction was inaccurate and try to identify the root causes.
    - Use this analysis to refine the data extraction process, tune parameters in current solution (main.py), or explore alternative extraction methods using different library such as Camelot maybe.
7. **Iterative Improvement**:
    - Based on the analysis and interpretation, make necessary adjustments to the data extraction process or the current tabula solution configuration.
    - Repeat the evaluation process with the updated extraction pipeline to assess the improvements in accuracy.
    - Continue iterating until the desired level of accuracy is achieved or until no further improvements are possible.

It's important to note that the choice of evaluation metrics and the interpretation of results may depend on the specific use case and the requirements of the data extraction task. Additionally, it's recommended to use a representative sample of PDF files for evaluation, covering various layouts, formatting styles, and complexities.

By following this framework, you can systematically evaluate the accuracy of data extraction and continuously improve the extraction process until it meets the desired quality standards. This is a task that I would perform if I had more time.

Output folders contains two main chatgpt tables merged_tables_chatgpt1 and merged tables_chatgpt2 and here the two prompts we have used to split data:

<u>Prompt1:</u> *"I have plain text tabular data, and I need help splitting it into proper columns. Please extract the information and organize it into a structured table with appropriate headers. Return the result as proper strongly comma-separated CSV only. We may have many tables, so split them into tables too.*
And skip rows without the data. Do not numerate rows. Do not include any comments chatgpt response except the processed data. Put the table name before each table if possible.
Split the Schedule of notices of lease data into separate columns. Do not include additional columns/data at the beginning of each row."

<u>Prompt2:</u>
"I have plain text tabular data that needs to be split into five major columns with the following headers:
1. Registration Date and Plan Ref: Example entry: 31.10.2016 1 in yellow.
2. Property Description: Example entry: Retail Warehouse, The Causeway and River Park Avenue.
3. Date of Lease and Term: Example entry: 25.07.1996 25 years from 25.3.1995.
4. Lessee's Title: Example entry: SY664660.
5. Note: Example entry: The Lease comprises also other land.
Please extract the information from the Schedule of Notices of Lease data and organize it into a structured table with appropriate headers. Each table should have the corresponding document page number as its title.
Requirements:
- Return the result as a properly comma-separated CSV.
- Ensure clear separation between the first four columns and the "Note" column.
- Skip rows that do not contain data.
- Do not numerate rows.
- Do not include any comments in the response, only the processed data.
- Each table should have the corresponding document page number as its title.
- There should be 74 tables in total, each corresponding to a page from the original PDF document.
- Ensure each table title reflects the page number from the original PDF."