CAMP AUTOMATION TOOL

MERGING SOFTWARE:

**Introduction:**

The **PDF Merger** is a Python script designed to merge multiple PDF files into organized categories. The script prompts the user to select input and output directories, then automatically merges the PDF files based on their patient\_id into designated output folders.

**User Input:**

* The user is prompted to select an input directory containing the PDF files to be merged.
* Then, the user selects an output directory where the merged PDFs will be stored.

**PDF Parsing and categorazation:**

* The script iterates through each PDF file in the input directory.
* It categorizes each PDF based on keywords found within the document's content.
* Categories include X-ray reports, optometry reports, ECG reports, PFT reports, audiometry reports, vitals reports, sample collection documents, and others.

**Merging PDFs:**

* After categorization, the script merges PDFs belonging to the same category.
* Each category of PDFs is merged into a single PDF file.
* If the total page count of the merged PDF falls within a specific range (Eg: 23-24), it's considered a "Complete Merge"; otherwise, it's categorized as "missing\_merge".

**OUTPUT:**

* Merged PDFs are saved in the output directory under respective subdirectories ("Complete Merge" or "merged\_data").
* In case of errors during processing, the problematic files are moved to an "error\_pdf" folder within the output directory.

**User Notify:**

* Upon completion, the script displays a message box informing the user of the total number of input PDFs processed and the number of PDFs generated after merging.

**Libraries Used:**

* **os**: For interacting with the operating system.
* **os.path**: For manipulating file paths.
* **pathlib**: For handling file paths.
* **PyPDF2**: For reading PDF files and merging them.
* **tkinter.filedialog**: For providing file dialog boxes to the user.
* **tkinter.messagebox**: For displaying message boxes to the user.

**­­­­­­­­­­­­­­­­­**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

PDFs FILE RENAMING:

**Introduction:**

The **PDF Merger** is a Python script designed to merge multiple PDF files into organized categories. The script prompts the user to select input and output directories, then automatically merges the PDF files based on their patient\_id into designated output folders.

**User Input:**

* The user is prompted to select an input directory containing the PDF files to be merged.
* Then, the user selects an output directory where the merged PDFs will be stored.

**PDF Parsing and Renaming:**

* The script iterates through each PDF file in the input directory.
* It extracts patient\_id and patient\_name information from the PDF file based on specific keywords related to different types of medical reports.
* Patient information is extracted differently based on the type of modalities (e.g., X-ray, PFT, ECG, etc.).
* The extracted patient\_id and name are used to construct a new filename for the PDF file.

**Error Handling:**

* If an error occurs during the parsing or renaming process, the script moves the problem file to an "error\_files" directory within the output directory.
* Error details are logged, allowing users to manually inspect and resolve the issues.

**Libraries Used:**

* **os**: For interacting with the operating system.
* **pathlib**: For handling file paths.
* **PyPDF2**: For reading PDF files and extracting text.
* **tkinter.filedialog**: For providing file dialog boxes to the user.
* **tkinter.messagebox**: For displaying message boxes to the user.
* **shutil**: For file operations like copying.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

REPORTS OBSERVATION:

**Introduction:**

The **Patient Data Extractor** is a Python script coded to extract patient data from medical PDF reports and store it in excel files. It parses PDF files, extracts relevant patient information, such as patient ID, name, age, gender, test dates, report dates, and findings, and organizes this data into separate Excel files based on the type of medical report (e.g., ECG, PFT, X-ray).

**User Input:**

* The user is prompted to select an input directory containing the PDF files from which patient data will be extracted.
* Then, the user selects an output directory where the extracted patient data will be stored.

**PDF Parsing and Renaming:**

* The script iterates through each PDF file in the input directory.
* It extracts patient data from different types of medical reports such as ECG, PFT, and X-ray, based on specific keywords and formats within the PDF content.

**Data Organization:**

* Extracted patient data is organized into separate lists for each type of medical report (e.g., ECG, PFT, X-ray).
* The script creates separate Excel files for each type of medical report and stores the extracted data in corresponding sheets within these files.

**Data Visualization:**

* For X-ray reports, the script highlights findings indicating 'No significant abnormality seen' in green and other findings in red within the Excel sheet.

**Libraries Used:**

* **os**: For interacting with the operating system.
* **pathlib**: For handling file paths.
* **PyPDF2**: For reading PDF files and extracting text.
* **openpyxl**: For creating and writing Excel files.
* **tkinter.filedialog**: For providing file dialog boxes to the user.
* **tkinter.messagebox**: For displaying message boxes to the user.
* **shutil**: For file operations like copying.

PDF CHECKER:

**Data Visualization:**

This function compares patient data from an Excel sheet with information extracted from merged PDF files. It checks for matching patient IDs and other relevant data across different modalities such as ECG, X-ray, PFT, Audiometry, etc., and generates a comparison report in an Excel file.

**Input:**

* Merged PDF Folder: Folder containing merged PDF files.
* Excel Sheet: Excel file containing patient data.
* Output Directory: Directory to save the comparison results.

**Output:**

* Excel file named "patient\_data\_comparison.xlsx" containing comparison results.

**Libraries Used:**

* os: Provides functions for interacting with the operating system.
* shutil: Offers functions for file operations.
* datetime: Enables date and time manipulation.
* openpyxl: Used for creating and manipulating Excel files.
* PyPDF2: Facilitates PDF file processing.
* tkinter: Provides GUI components.
* pandas: Used for data manipulation and analysis.

**Function Steps:**

1. Prompt the user to select the folder containing merged PDF files.
2. Prompt the user to select the Excel file containing patient data.
3. Prompt the user to select the output directory for saving the comparison results.
4. Create a new Excel workbook and select the active worksheet.
5. Add headers to the worksheet for patient data and modality comparisons.
6. Read patient data from the Excel file into a DataFrame.
7. Iterate over each row in the DataFrame.
8. Extract the patient ID prefix from the Excel data.
9. Find PDF files in the folder matching the patient ID prefix.
10. Initialize lists for modality matching and problems.
11. If no matching PDF file found for the patient ID, add a corresponding entry to the problem list.
12. If matching PDF files found, extract patient data from the Excel row.
13. Iterate through each modality and process its content based on specific patterns.
14. Check if the extracted data matches the Excel data for each modality.
15. Append the modality match result to the modality\_match\_list and any issues to the problem\_list.
16. Write the results to the worksheet and apply fill color to cells based on "Yes" or "No".
17. Save the workbook to the output directory.
18. Display a message box indicating the completion of the data comparison process.

**Error Handling:**

* If unable to process a PDF file, move it to an error folder and continue processing.
* Handle IndexError exceptions that may occur during page text extraction.

SPLIT PDF FILES:

**Introduction:**

This function splits merged PDF files containing multiple patient records into individual PDF files based on specific modality information extracted from each page. It creates a subdirectory for each patient and saves the corresponding pages as separate PDF files within their respective directories.

**Input:**

* Input Directory: Directory containing merged PDF files with multiple patient records.
* Output Directory: Directory to save the individual PDF files after splitting.

**Libraries used:**

* os: Provides functions for interacting with the operating system.
* shutil: Offers functions for file operations.
* PyPDF2: Facilitates PDF file processing.
* tkinter: Provides GUI components.
* pathlib: Offers classes for working with filesystem paths.

**Function Steps:**

1. Prompt the user to select the input directory containing merged PDF files.
2. If the input directory is selected, prompt the user to choose the output directory for saving the individual PDF files.
3. If both input and output directories are selected:
   * Iterate over each PDF file in the input directory.
   * Open the merged PDF file and create a PdfReader object.
   * Create a subdirectory for each patient if it doesn't already exist.
   * Loop through each page in the PDF and extract text data to determine the modality.
   * Based on the modality information, save each page as an individual PDF file within the patient's directory.
4. Handle any exceptions that occur during the processing of PDF files.
5. Print status messages to indicate the completion of the splitting process.

RUN AND GENERATE EXE:

**To Run: py file\_name.py**

**To Generate EXE: pyinstaller file\_name.spec**