OTHM

[OPERATION TASK'S HELPING MACHINE]

Tkinter

It is a standard Python library used for creating Graphical User Interfaces (GUIs). It provides a set of tools and widgets (such as buttons, labels, text boxes, and menus) that help developers build windows-based applications with a graphical interface, as opposed to command-line programs.

Repository :

The codebase is completely updated on the github id : "https://github.com/U4RAD9" , with the repo name as "Merging Software".

How To Run the Program :

- You just need to clone the repo "Merging Software".
- Their is no particular environment required for it, there are some basic libraries, that you can simply add.
- Once you clone , go to the directory where the file "merge_pdfs.py" is present.
- Open the directory in your preferred code editor.
- Run the program using "python merge_pdfs.py" or "py merge_pdfs.py" .

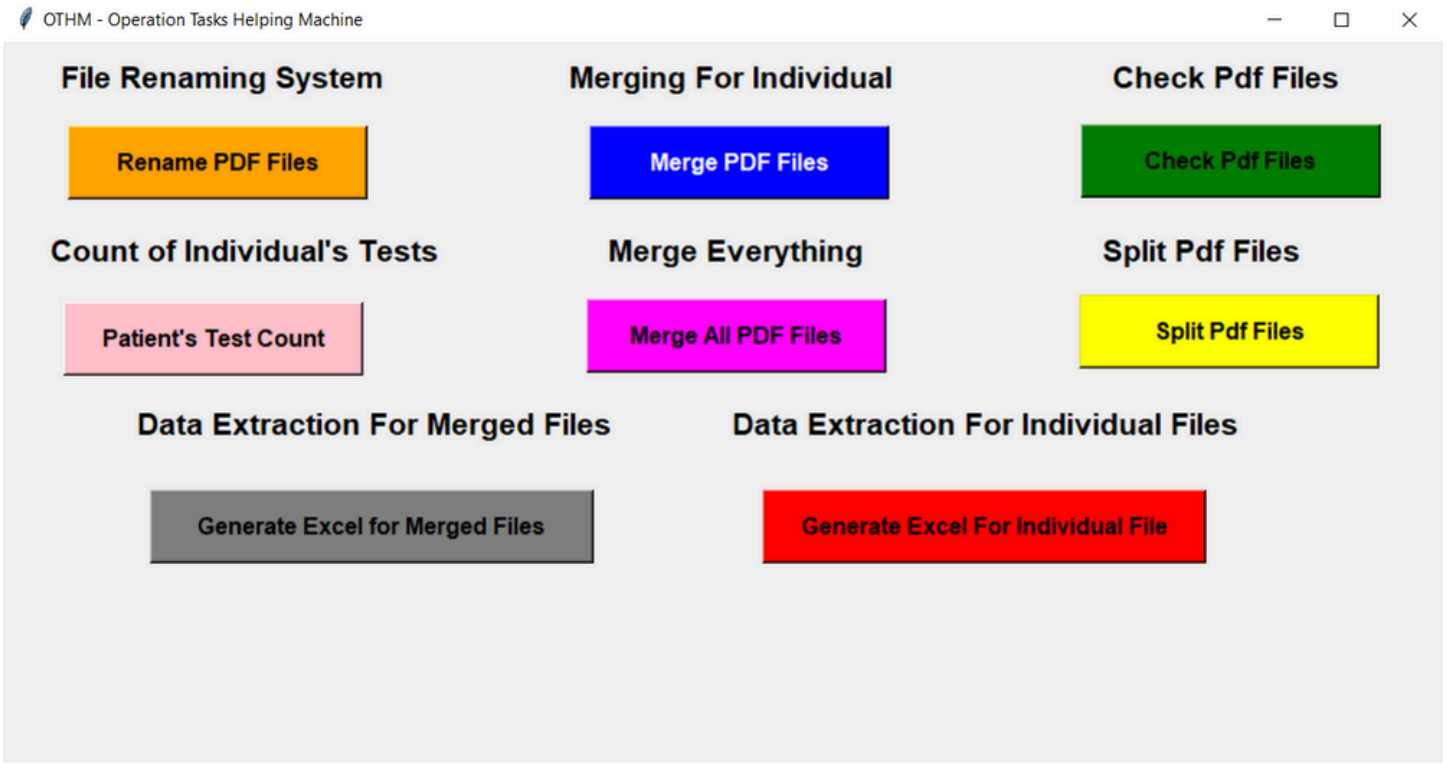
How to Create the Exe :

- Their will be a file named "merge_pdfs.spec".
- This file contains the logic to create the exe from our python program.
- You just need to change the name of the software in it "name='HimanshuMergingSoftware(V5)', ".
- Everytime a new version is created, just update the version here and use accordingly.
- Once, you run the command the created exe will be stored in the directory named "dist" .
- Command to create the exe : "pyinstaller merge_pdfs.spec" .

README.md File :

In this file, mainly i've written one to one updates, giving a basic understanding step by step, that what work is done when.

This is our main s/w, once you click the “HimanshuMergingSoftware(V$).exe” or when you run the “merge_pdfs.py” file, than you will see this frontend to interact with our OTHM or Merging Software.



Tkinter Window :

I have used the window given by the Tkinter library to create this frontend to interact with the user, where simply i have used it’s labels and buttons to make it more interactive.

NOTE: I can simply add more labels and buttons here and add the respective function to make new automation things in it.

```
3018    # Create the main window
3019    window = tk.Tk()
3020    window.title("OTHM - Operation Tasks Helping Machine")
3021    # Set the window dimensions and position it on the screen
3022    window.geometry("1000x500+200-100")
3023
3024
3025    redcliffe_label = tk.Label(window, text="Merging For Individual", font=("Arial", 16, "bold"))
3026    redcliffe_label.place(x=620, y=10, anchor='ne')
3027
3028    # Adding the label of Merge All files button .
3029    merge_all_files = tk.Label(window, text="Merge Everything", font=("Arial", 16, "bold"))
3030    merge_all_files.place(x=600, y=130, anchor='ne')
3031
3032    merge_redcliffe_button1 = tk.Button(window, bg='blue', fg='white', activebackground='darkblue', activeforeground='white',
3033    merge_redcliffe_button2 = tk.Button(window, bg='magenta', fg='black', activebackground='gold', activeforeground='black',
3034    merge_redcliffe_button1.place(x=615, y=58, anchor='ne')
3035    merge_redcliffe_button2.place(x=613, y=178, anchor='ne')
3036
3037    pdf_rename_label = tk.Label(window, text="File Renaming System", font=("Arial", 16, "bold"))
3038    pdf_rename_label.pack(pady=10, padx=37, anchor='w')
3039
3040    pdf_rename_button1 = tk.Button(window, bg='orange', fg='black', activebackground='darkblue', activeforeground='white', pa
3041    pdf_rename_button1.pack(pady=8, padx=45, anchor='w')
3042
3043    generate_excel_label = tk.Label(window, text="Count of Individual's Tests", font=("Arial", 16, "bold"))
3044    generate_excel_label.place(x=305, y=130, anchor='ne')
3045
3046    generate_excel_button = tk.Button(window, bg='pink',fg='black', activebackground='darkblue', activeforeground='white',pac
3047    generate_excel_button.place(x=250, y=180, anchor='ne')
3048
3049    check_pdf_File = tk.Label(window, text="Check Pdf Files", font=("Arial", 16, "bold"))
3050    check_pdf_File.place(x=930, y=10, anchor='ne')
3051
3052    check_pdf_button = tk.Button(window, bg='green',fg='black', activebackground='darkblue', activeforeground='white',padx=25
3053    check_pdf_button.place(x=956, y=57, anchor='ne')
```

```
check_pdf_button.place(x=956, y=57, anchor='ne')

check_pdf_File = tk.Label(window, text="Split Pdf Files", font=("Arial", 16, "bold"))
check_pdf_File.place(x=903, y=130, anchor='ne')

check_pdf_button = tk.Button(window, bg='yellow',fg='black', activebackground='darkblue', activeforeground='white',padx=2
check_pdf_button.place(x=955, y=175, anchor='ne')

# Label for the check Generate Excel for Merged Files button.
check_ecg_files_label = tk.Label(window, text="Data Extraction For Merged Files", font=("Arial", 16, "bold"))
check_ecg_files_label.place(x = 425, y=250, anchor='ne')
# Button for the check ecg file label.
check_ecg_files_button = tk.Button(window, bg='grey',fg='black', activebackground='darkgrey', activeforeground='white',pa
check_ecg_files_button.place(x= 410, y=310, anchor='ne')

# Label for the check ecg file button.
check_ecg_files_label = tk.Label(window, text="Data Extraction For Individual Files", font=("Arial", 16, "bold"))
check_ecg_files_label.place(x = 860, y=250, anchor='ne')
# Button for the check ecg file label.
check_ecg_files_button = tk.Button(window, bg='red',fg='black', activebackground='red', activeforeground='white',padx=25,
check_ecg_files_button.place(x=835, y=310, anchor='ne')


# dcm_to_pdf = tk.Label(window, text="Reports Observation", font=("Arial", 16, "bold"))
# dcm_to_pdf.place(x=233, y=255, anchor='ne')
#
# dcm_to_pdf_button = tk.Button(window, bg='red',fg='black', activebackground='darkblue', activeforeground='white',padx=
# dcm_to_pdf_button.place(x=328, y=300, anchor='ne')



window.mainloop()
```

LIST OF LIBRARIES :

• **os**: For interacting with the operating system, such as file and directory manipulation.
• **os.path**: For manipulating file paths and working with file system paths.
• **pathlib**: For handling file and directory paths in an object-oriented way.
• **PyPDF2**: For reading, merging, and manipulating PDF files.
• **tkinter.filedialog**: For providing file dialog boxes to allow users to select files or directories.
• **tkinter.messagebox**: For displaying message boxes to the user for alerts, warnings, or confirmations.
• **PdfMerger (from PyPDF2)**: For merging multiple PDF files into one.
• **PdfReader (from PyPDF2)**: For reading PDF files.
• **PdfWriter (from PyPDF2)**: For writing or creating PDF files.
• **re**: For working with regular expressions, used for string searching and manipulation.
• **time**: For working with time-related functions, such as delays or timestamp generation.
• **shutil**: For file operations, such as copying and moving files.
• **openpyxl**: For reading and writing Excel (.xlsx) files.
• **PIL (Python Imaging Library)**: For image processing tasks such as opening, editing, and saving image files.
• **pytesseract**: For Optical Character Recognition (OCR), extracting text from images.
• **fitz (from PyMuPDF)**: For working with PDF documents, extracting text, images, and more.
• **openpyxl.styles**: For styling Excel files (e.g., setting font styles and cell colors).
• **openpyxl.Workbook**: For creating new Excel workbooks.
• **pandas**: For data manipulation and analysis, particularly with structured data like tables (DataFrames).
• **ThreadPoolExecutor (from concurrent.futures)**: For parallel execution of functions using threads, enabling faster processing for concurrent tasks.
• **math**: For performing mathematical functions, such as trigonometry, logarithms, and more.
• **reportlab.pdfgen.canvas**: For creating PDF documents programmatically.
• **reportlab.lib.pagesizes**: For setting the page size of PDFs, such as letter, A4, etc.
• **datetime**: For working with date and time operations, including formatting, manipulation, and comparisons.
• **warnings**: For managing warning messages in Python, to suppress or raise specific warnings.

```
import tkinter as tk
from tkinter import messagebox, filedialog
from pathlib import Path
from PyPDF2 import PdfMerger, PdfReader, PdfWriter
import PyPDF2
import os
import re
import time
import shutil
import openpyxl
from PIL import Image
import pytesseract
import fitz
from openpyxl.styles import Font, PatternFill
from openpyxl import Workbook
import pandas as pd
from concurrent.futures import ThreadPoolExecutor
import math
# import pydicom
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from datetime import datetime

import warnings
```

Now, let's see the HELPER FUNCTIONS that i have created to reduce the boiler plate codes :

- Helper for printing the logging :

```
# This function simply prints the data wnen needed for logging, to make the code non boiler plate code. - Himanshu.
def print_page_text_for_logging(page_text):
    print("Starting of this page(or file) text :")
    print(page_text)
    print("End of this page(or file) text.")
```

- Creating the patient details and the patient data dictionary :

```python
# This function i've made to do both work, creating or emptying the patient data dictionary , when needed.
def creating_or_emptying_the_patient_data_dictionary():
    patient_data = {
            'patient_id': None,
            'patient_name': None,
            'patient_age': None,
            'gender': None,
            'test_date': None,
            'report_date': None,
            'XRAY': 'Not Present',
            'ECG': 'Not Present',
            'AUDIOMETRY': 'Not Present',
            'OPTOMETRY': 'Not Present',
            'VITALS': 'Not Present',
            'PFT': 'Not Present',
            'PATHOLOGY': 'Not Present',
            'OTHERS': 'Not Present'
        }
    return patient_data

# This function i have made to create the patient details dictionary when needed.
def create_patient_details():
    patient_details= {'patient_id': None,'patient_name': None,'patient_age': None,'gender': None,'test_date': None,'repor
    return patient_details
```

- To make the user select the input folder and the output folder , so that he/she can give us the directory where files are present that needs to be processed, and the output directory where they want the results after processing :

```python
# This is to ask the user for the input and output path to reduce the code redundancy. -- Himanshu.
def select_folders():
    input_folder_path = filedialog.askdirectory(title="Select Input Files Folder", mustexist=True)
    if not input_folder_path:
        print("Input files folder not selected.")
        tk.messagebox.showwarning("Input Directory", "Input directory not selected.")
        return None, None

    output_folder_path = filedialog.askdirectory(title="Select Output Files Folder", mustexist=True)
    if not output_folder_path:
        print("Output files folder not selected.")
        tk.messagebox.showwarning("Output Directory", "Output directory not selected.")
        return None, None

    return input_folder_path, output_folder_path
```

- This function maps each and every respective modality data extractor function that extracts the data from the file based on what type of file it is :

```python
def extract_missing_data_for_modality(modality, page_text, patient_details):
    """
    Extract missing data for the given modality and update the patient details.
    """
    missing_keys = [key for key, value in patient_details.items() if value is None]

    if missing_keys:
        # Mapping modalities to their corresponding extraction functions
        modality_functions = {
            'XRAY-BOT': extract_data_from_the_bot_xray_file,
            'XRAY-STRADUS': extract_data_from_the_stradus_xray_file,
            'XRAY-U4RAD-PACS': extract_data_from_the_u4rad_pacs_xray_file,
            'PFT': extract_data_from_bot_pft_file,
            'OPTOMETRY': extract_data_from_bot_opto_file,
            'AUDIOMETRY': extract_data_from_bot_audio_file,
            'ECG': extract_data_from_bot_ecg_file,
            'VITALS': extract_data_from_bot_vitals_file,
            'PATHOLOGY': extract_data_from_the_redcliffe_patho_file
        }

        # This is to Check if the modality exists in the dictionary and extract the data
        # This will check the respective key(which is the modality here, which i pass while calling the function.)
        # The key takeaway is the modality i pass and the modality i add in the set can be different, so i can map these
        # Extracting the data from the stradus / our pacs report etc. the function will set the modality_data as an empty
        # More details if needed than see in my documentation or search on web.
        modality_data = modality_functions.get(modality, lambda x: {}) (page_text)

        # Logging the extraction message
        if modality_data:
            # This will tell me the patient details when any field in it is empty.
            print(f"This is the patient details just before adding the extracted data in it, because some fields were emp
            print(f"Data extracted from the {modality} report.")

        # Updating patient details with the extracted data
        for key in missing_keys:
```

- This gives separate excel workbook’s based on the modality it is :

```python
def modality_based_excel_workbook(modality):
    print(f"This is the modality :{modality}")
    # Headers for each modality
    headers = {
        'XRAY': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'FINDINGS'],
        'PFT': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'HEIGHT', 'WEI
        'OPTOMETRY': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'FAR VIS
                      'FAR VISION LEFT', 'NEAR VISION LEFT', 'DISTANCE VISION LEFT', 'READING VISION LEFT', 'SPHERICAL L
        'AUDIOMETRY': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'LEFT EA
        'ECG': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'HEART RATE',
        'VITALS': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'HEIGHT', 'W
        'PATHOLOGY': ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE', 'HAEMOGLO
                      'TLC', 'NEUTROPHILS (DLC)', 'LYMPHOCYTES (DLC)', 'MONOCYTES (DLC)', 'EOSINOPHILS (DLC)', 'BASOPHILS
                      'EOSINOPHILS (ALC)', 'BASOPHILS (ALC)', 'PLATELET COUNT', 'MPV', 'PCT']
    }
    # Creating workbook for each type of modality
    xray_Workbook = Workbook()
    pft_Workbook = Workbook()
    opto_Workbook = Workbook()
    audio_Workbook = Workbook()
    ecg_Workbook = Workbook()
    vitals_Workbook = Workbook()
    patho_Workbook = Workbook()
    # Mapping modalities to their corresponding excel workbooks.
    modality_workbook = {
        'XRAY': xray_Workbook,
        'PFT': pft_Workbook,
        'OPTOMETRY': opto_Workbook,
        'AUDIOMETRY': audio_Workbook,
        'ECG': ecg_Workbook,
        'VITALS': vitals_Workbook,
        'PATHOLOGY': patho_Workbook
    }
    # # Initialize workbooks for each modality
    # modality_workbook = {
```

The above helper functions are some of the basic ones that are used to simplify the task to some extent.

Important Helper Functions :

These are some of the important functions that provide data based on the type of report is present in it.

- Each report is having a fixed type of format.
- If it is a xray file , than there are three types of files : Report generated by our Reporting Bot , Report generated by Stradus, and the Report generated by our U4RAD PACS Bot.
- Mostly, the pathology files are present from the Redcliffe Blood Reports, so I've extracted the data as per the data i am extracting from that file.
- Also, I am extracting data from similar files like Vitals Report, Optometry Report, Audiometry Report, ECG Report ,and PFT Report.
- Doctor Consultation file is a simple image getting appended in the reports, so i can't extract any data from it.
- Smart Report is mostly a image (but sometimes a extractable file , not created any function to extract data from it yet.) , so i can't extract any data from it too.

All Important Functionality Helpers :

- Function to categorise the reports in terms of modality and then use the proper functions to extract their data too :

```python
def extract_data_based_on_modality(page_text, patient_details, modalities):
    """
    Function to check for different modalities in the page_text,
    extract missing data, and update patient details accordingly.

    Parameters:
    - page_text (str): Text extracted from the page
    - patient_details (dict): A dictionary containing patient details
    - modalities (set): A set to store the modalities found

    Returns:
    - updated_patient_details (dict): The updated patient details with newly extracted data
    - updated_modalities (set): The updated set of modalities found
    """
    # Define unwanted phrases
    unwanted_phrases = ["Referrer Dr.", "left ear", "RECORDERS & MEDICARE SYSTEMS",
                        "OPTOMETRY", "ECG", "VITALS", "RBC Count", "PDW *", "PDW"]

    # Check the text content for known modalities
    # For the xray report from stradus.
    if "Referrer Dr" in page_text and "Time" in page_text:
        # This will print text only when any modality matches.
        print_page_text_for_logging(page_text)
        modalities.add('XRAY')
        # Calling the data extractor function i've created wrt each conditional expectation.
        patient_details = extract_missing_data_for_modality('XRAY-STRADUS', page_text, patient_details)
        # Additional logging to see the patient details extracted so far, and it will give the patient details only if sc
        print(f"This is the patient details after adding the data which we extracted and because the details were not com

    # For the xray report from u4rad pacs reporting bot.
    if "Test Date:" in page_text and "Report Date:" in page_text and not any(phrase in page_text for phrase in unwanted_p
        # This will print text only when any modality matches
```

```python
def extract_data_based_on_modality(page_text, patient_details, modalities):
    elif page_text == "":
        # This will print text only when any modality matches.
        print_page_text_for_logging(page_text)
        print("This is an Others image.")
        print("Since it is an others file, means the page is blank, No data to extract.")
        modalities.add('OTHERS')
    elif "VITALS" in page_text:
        # This will print text only when any modality matches.
        print_page_text_for_logging(page_text)
        print("This is a Vitals file.")
        modalities.add('VITALS')
        # Calling the data extractor function i've created wrt each conditional expectation.
        patient_details = extract_missing_data_for_modality('VITALS', page_text, patient_details)
        # Additional logging to see the patient details extracted so far, and it will give the patient details only if s
        print(f"This is the patient details after adding the data which we extracted and because the details were not co

    elif "RBC Count" in page_text or "PDW *" in page_text or "PDW" in page_text :
        # This will print text only when any modality matches.
        print_page_text_for_logging(page_text)
        print("This is a Blood Report.")
        modalities.add('PATHOLOGY')
        # Calling the data extractor function i've created wrt each conditional expectation.
        patient_details = extract_missing_data_for_modality('PATHOLOGY', page_text, patient_details)
        # Additional logging to see the patient details extracted so far, and it will give the patient details only if s
        print(f"This is the patient details after adding the data which we extracted and because the details were not co

    # Only the above conditions will tell that whether it is a particular test or not, if others are needed , I'll updat


    return patient_details, modalities
```

- Function to extract the data from all the pft files :

```python
# This is my function to extract data from all the pft files :
def extract_data_from_bot_pft_file(pageText):
    # Extracting the required information from the text
    patient_info = {}

    try:
        patient_info['patient_id'] = str(pageText).split("ID")[1].split("Age")[0].split(":")[1].strip()
        patient_info['patient_name'] = str(pageText).split("Patient")[1].split("Refd.By:")[0].split(":")[1].strip()
        patient_info['patient_age'] = str(pageText).split("Age      :")[1].split("Yrs")[0].strip()
        patient_info['gender'] = str(pageText).split("Gender")[1].split("Smoker")[0].split(":")[1].strip()
        patient_info['height'] = str(pageText).split("Height :")[1].split("Weight")[0].strip()
        [illegible] = str([illegible]eText).split("Weight")[1].split("Gender")[0].split(":")[1].split("Kgs")[0].strip
        (variable) patient_info: dict pageText).split("Date")[1][1:21].split(":")[1]
        patient_info['report_date'] = patient_info['test_date']
        patient_info['observation'] = str(pageText).split("Pre Test COPD Severity")[1].strip()
    except IndexError:
        print("Error extracting data from the first page text.")

    return patient_info
```

- Function to extract all the ECG Files :

```python
# This is my function for extracting data from all ECG reports:
def extract_data_from_bot_ecg_file(pageText):
    patient_info = {}
    try:
        patient_info['patient_id'] = str(pageText).split('Patient ID:')[1].split('Age:')[0].strip()
        patient_info['patient_name'] = str(pageText).split("Name:")[1].split("Patient ID:")[0].strip()
        patient_info['patient_age'] = str(pageText).split("Age:")[1].split('Gender:')[0].strip()
        patient_info['gender'] = str(pageText).split("Gender:")[1].split("Test date:")[0].strip()
        patient_info['test_date'] = str(pageText).split("Test date:")[1].split('Report date:')[0].strip()
        patient_info['report_date'] = str(pageText).split("Report date:")[1].split('ECG')[0].strip()
        patient_info['heart_rate'] = str(pageText).split("Heart rate is")[1].split("BPM.")[0].strip()
        patient_info['findings'] = str(pageText).split("2.")[1].split('.')[0].strip()
    except IndexError:
        print("Error extracting ECG data.")
    return patient_info
```

- Now, this function is completely dependent on the Redcliffe files.
- **Note : I am writing this in prior that REDCLIFFE's Blood Reports are always with some format change. So Each time you face any error, add those formats too in the extraction logic (Without removing the previous logic) .**
- This is the function to extract the data from the Pathology (Blood) Reports :

```python
# Function for extracting data from Blood reports
def extract_data_from_the_redcliffe_patho_file(pageText):
    patient_info = {}
    try:
        # Handling the extraction of patient details with error handling for each field
        try:
            if "Patient Name :" in pageText:
                complete_patient_name = str(pageText).split("Patient Name : ")[1].split("DOB/")[0].strip()
                patient_info['patient_id'] = complete_patient_name.rsplit(" ", 1)[1]
                patient_info['patient_name'] = complete_patient_name.rsplit(" ", 1)[0].split(" ", 1)[1].lower()
            elif "Patient NAME :" in pageText:
                complete_patient_name = str(pageText).split("Patient NAME : ")[1].split("DOB/")[0].strip()
                patient_info['patient_id'] = complete_patient_name.rsplit("_", 1)[1]
                patient_info['patient_name'] = complete_patient_name.rsplit("_", 1)[0].split(" ", 1)[1].lower()
            elif "PATIENT NAME :" in pageText:
                complete_patient_name = str(pageText).split("PATIENT NAME : ")[1].split("DOB/")[0].strip()
                patient_info['patient_id'] = complete_patient_name.rsplit("_", 1)[1]
                patient_info['patient_name'] = complete_patient_name.rsplit("_", 1)[0].split(" ", 1)[1].lower()
            elif "PATIENT Name :" in pageText:
                complete_patient_name = str(pageText).split("PATIENT Name : ")[1].split("DOB/")[0].strip()
                patient_info['patient_id'] = complete_patient_name.rsplit("_", 1)[1]
                patient_info['patient_name'] = complete_patient_name.rsplit("_", 1)[0].split(" ", 1)[1].lower()
            else:
                patient_info['patient_id'] = None
                patient_info['patient_name'] = None
        except Exception as e:
            patient_info['patient_id'] = None
            patient_info['patient_name'] = None
            print(f"Error extracting patient details: {e}")

        # Age, gender, and test dates with error handling
        try:
```

- To extract data from the Audiometry Reports :

```python
# This is my function for extracting data from all Reporting Bot Audiometry Reports:
def extract_data_from_bot_audio_file(pageText):
    patient_info = {}
    try:
        patient_info['patient_id'] = str(pageText).split('Patient ID')[1].split('Age')[0].strip()
        patient_info['patient_name'] = str(pageText).split("Name")[1].split("Patient ID")[0].strip()
        patient_info['patient_age'] = str(pageText).split("Age")[1].split('Gender')[0].strip()
        patient_info['gender'] = str(pageText).split("Gender")[1].split("Test date")[0].strip()
        patient_info['test_date'] = str(pageText).split("Test date")[1].split('Report date')[0].strip()
        patient_info['report_date'] = str(pageText).split("Report date")[1].strip()
        patient_info['left_ear_finding'] = str(pageText).split("in left ear")[0].split("Finding:")[1].strip()
        patient_info['right_ear_finding'] = str(pageText).split("in right ear")[0].split("in left ear.")[1].strip()
    except IndexError:
        print("Error extracting Audiometry data.")
    return patient_info
```

- To extract data from Optometry Reports :

```python
# This is my function for extracting data from all Reporting Bot Optometry Reports:
def extract_data_from_bot_opto_file(pageText):
    patient_info = {}
    try:
        patient_info['patient_id'] = str(pageText).split('Patient ID:')[1].split('Patient Name:')[0].strip()
        patient_info['patient_name'] = str(pageText).split("Patient Name:")[1].split("Age:")[0].strip()
        patient_info['patient_age'] = str(pageText).split("Age:")[1].split('Gender:')[0].strip()
        patient_info['gender'] = str(pageText).split("Gender:")[1].split("Test Date:")[0].strip()
        patient_info['test_date'] = str(pageText).split("Test Date:")[1].split('Report Date:')[0].strip()
        patient_info['report_date'] = str(pageText).split("Report Date:")[1].split('OPTOMETRY')[0].strip()
        patient_info['far_vision_right'] = str(pageText).split("Distance(Far):")[1].split('vision in right eye')[0].strip
        patient_info['near_vision_right'] = str(pageText).split("Distance(Near):")[1].split('vision in right eye')[0].str
        patient_info['distance_vision_right'] = str(pageText).split("Right Eye")[1].split('Left Eye')[0].strip().split("
        patient_info['reading_vision_right'] = str(pageText).split("Right Eye")[1].split('Left Eye')[0].strip().split("
        patient_info['spherical_right'] = str(pageText).split("Right Eye")[1].split('Left Eye')[0].strip().split(" ")[2]
        patient_info['cylindrical_right'] = str(pageText).split("Right Eye")[1].split('Left Eye')[0].strip().split(" ")[
        patient_info['axis_right'] = str(pageText).split("Right Eye")[1].split('Left Eye')[0].strip().split(" ")[4].strip
        patient_info['add_right'] = str(pageText).split("Right Eye")[1].split('Left Eye')[0].strip().split(" ")[5].strip(
        patient_info['far_vision_left'] = str(pageText).split("Distance(Far):")[1].split('vision in right eye-')[1].split
        patient_info['near_vision_left'] = str(pageText).split("Distance(Far):")[1].split('vision in right eye-')[1].spli
        patient_info['distance_vision_right'] = str(pageText).split("Left Eye")[1].split('Color vision')[0].strip().split
        patient_info['reading_vision_left'] = str(pageText).split("Left Eye")[1].split('Color vision')[0].strip().split('
        patient_info['spherical_left'] = str(pageText).split("Left Eye")[1].split('Color vision')[0].strip().split(" ")[2
        patient_info['cylindrical_left'] = str(pageText).split("Left Eye")[1].split('Color vision')[0].strip().split(" ")
        patient_info['axis_left'] = str(pageText).split("Left Eye")[1].split('Color vision')[0].strip().split(" ")[4].str
        patient_info['add_left'] = str(pageText).split("Left Eye")[1].split('Color vision')[0].strip().split(" ")[5].stri
        patient_info['colour_blindness'] = str(pageText).split("Color vision check(Ishihara test):")[1].split('color blin
    except IndexError:
        print("Error extracting Optometry data.")
    return patient_info
```

- To extract data from Vitals Reports :

```python
# This is my function for extracting data from all Reporting Bot Vitals Reports:
def extract_data_from_bot_vitals_file(pageText):
    patient_info = {}
    try:
        # This is for our reporting bot vitals file.
        if "Patient Name:" in pageText:
            patient_info['patient_id'] = str(pageText).split('Patient ID:')[1].split('Patient Name:')[0].strip()
            patient_info['patient_name'] = str(pageText).split("Patient Name:")[1].split("Age:")[0].strip()
            patient_info['patient_age'] = str(pageText).split("Age:")[1].split('Gender:')[0].strip()
            patient_info['gender'] = str(pageText).split("Gender:")[1].split("Test Date:")[0].strip()
            patient_info['test_date'] = str(pageText).split("Test Date:")[1].split('Report Date:')[0].strip()
            patient_info['report_date'] = str(pageText).split("Report Date:")[1].split('VITALS')[0].strip()
        # This was some other format which i found in one of the vitals files.
        else:
            patient_info['patient_id'] = str(pageText).split('Patient ID:')[1].split('Age:')[0].strip()
            patient_info['patient_name'] = str(pageText).split("Name:")[1].split("Patient ID:")[0].strip()
            patient_info['patient_age'] = str(pageText).split("Age:")[1].split('Gender:')[0].strip()
            patient_info['gender'] = str(pageText).split("Gender:")[1].split("Test date:")[0].strip()
            patient_info['test_date'] = str(pageText).split("Test date:")[1].split('Report date:')[0].strip()
            patient_info['report_date'] = str(pageText).split("Report date:")[1].split('VITALS')[0].strip()
            patient_info['height'] = str(pageText).split("Height(in cm)")[1].split("Weight(in kg)")[0].strip()
            patient_info['weight'] = str(pageText).split("Weight(in kg)")[1].split("BMI(kg/m2)")[0].strip()
            patient_info['bp'] = str(pageText).split("Blood Pressure(mmHg)")[1].split("Pulse(bpm)")[0].strip()
            patient_info['Pulse'] = str(pageText).split("Pulse(bpm)")[1].strip()
    except IndexError:
        print("Error extracting Vitals data.")
    return patient_info
```

Most Important

Now, these are the three important types of data extractions, these are all XRAY Report Data Extraction function , but one is for our Reporting Bot , one for Stradus Reports and the last for our U4RAD Orthanc Pacs Reports.

- Our Reporting Bot Xray Report Data Extraction function :

```
# This is my function for extracting data from X-ray reports of our Reporting Bot:
def extract_data_from_the_bot_xray_file(pageText):
    patient_info = {}
    try:
        # Check if the "IMPRESSION:" text exists for finding data, right now , i am commenting it.
        if 'IMPRESSION:' in pageText:
            findings_data = str(pageText).split('IMPRESSION:')[1].split("Dr.")[0]
            if "•" in findings_data:
                findings = findings_data.split("•")[1].split(".")[0].strip()
            else:
                findings = findings_data.strip()
        else:
            findings = None

        # If specific "Study Date" and "Report Date" condition applies, this i've still included in case the format chan
        if "Study Date" and "Report Date" in pageText:
            patient_info['patient_id'] = str(pageText).split("Patient ID")[1].split(" ")[1].lower().strip()
            patient_info['patient_name'] = str(pageText).split("Name")[1].split("Date")[0].split(" ")[0].strip().lower()
            if "patient" in patient_info['patient_name']:
                patient_info['patient_name'] = patient_info['patient_name'].split("patient")[0].strip()
            print(patient_info['patient_id'], patient_info['patient_name'])
        # Mostly this will be the case for our bot generated xrays.
        else:
            patient_info['patient_id'] = str(pageText).split('Patient ID:')[1].split('Age:')[0].strip()
            patient_info['patient_name'] = str(pageText).split('Name:')[1].split('Patient ID:')[0].strip()
            patient_info['patient_age'] = str(pageText).split('Age:')[1].split('Gender:')[0].strip()
            patient_info['gender'] = str(pageText).split('Gender:')[1].split('Test date:')[0].strip()
            patient_info['test_date'] = str(pageText).split('Test date:')[1].split('Report date:')[0].strip()
            patient_info['report_date'] = str(pageText).split('Report date:')[1].split('X-RAY')[0].strip()
        # This i will use to clean up the findings of the data.
        patient_info['findings'] = remove_illegal_characters(findings)
```

- Our Stradus Xray Report Data Extraction Function :

```python
# This is my function for extracting data from Stradus X-ray reports:
def extract_data_from_the_stradus_xray_file(pageText):
    patient_info = {}
    try:
        patient_info['patient_id'] = str(pageText).split('Patient ID')[1].split('Age')[0].strip()
        patient_info['patient_name'] = str(pageText).split('Patient Name')[1].split('Patient ID')[0].strip()
        patient_info['patient_age'] = 'Non-Extractable'
        patient_info['gender'] = str(pageText).split('Sex')[1].split('Study Date')[0].strip()
        patient_info['test_date'] = str(pageText).split('Study Date')[1].split('Report Date')[0].split('Time')[1].split(
        # It is really varying here, sometimes there is "impression:", sometimes "IMPRESSION", similarly for word "obser
        report_date_data = str(pageText).split('Report Date')[1].split('Dr.')[0]
        # Considering the case when there is am in time.
        if 'am' in report_date_data:
            patient_info['report_date'] = str(pageText).split('Report Date')[1].split('am')[0].split('Time')[1].split('\
        # Considering the case, when there is pm in time.
        else:
            patient_info['report_date'] = str(pageText).split('Report Date')[1].split('pm')[0].split('Time')[1].split('\

        # I will extract other data from here afterwards.
        # I know that the doctors will make this mistake so i'm fixing it here (maximum cases). -- Himanshu.
        if 'IMPRESSION :-' in pageText:
            findings_data = str(pageText).split('IMPRESSION :-')[1].split("ADVICE :-")[0]
            if "•" in findings_data:
                findings = findings_data.split("•")[1].split(".")[0].strip()
            else:
                findings = findings_data.strip()
        elif 'IMPRESSIONS :-' in pageText:
            findings_data = str(pageText).split('IMPRESSIONS :-')[1].split("ADVICE :-")[0]
            if "•" in findings_data:
                findings = findings_data.split("•")[1].split(".")[0].strip()
            else:
```

- Our U4RAD Orthanc Pacs Report Data Extraction Function :

```python
# This is my function for extracting data from our orthanc pacs X-ray reports:
def extract_data_from_the_u4rad_pacs_xray_file(pageText):
    patient_info = {}
    try:
        patient_info['patient_id'] = str(pageText).split('Patient ID:')[1].split('Patient Age:')[0].strip()
        patient_info['patient_name'] = str(pageText).split('Patient Name:')[1].split('Patient ID:')[0].strip()
        patient_info['patient_age'] = str(pageText).split('Patient Age:')[1].split('Patient Gender:')[0].strip()
        patient_info['gender'] = str(pageText).split('Patient Gender:')[1].split('Test Date:')[0].strip()
        patient_info['test_date'] = str(pageText).split('Test Date:')[1].split('Report Date:')[0].strip()
        patient_info['report_date'] = str(pageText).split('Report Date:')[1].split('Dr.')[0].split('\n')[0].strip()

        # I will extract other data from here afterwards.
        # I know that the doctors will make this mistake so i'm fixing it here (maximum cases). -- Himanshu.
        if 'IMPRESSION:' in pageText:
            findings_data = str(pageText).split('IMPRESSION:')[1].split("Dr.")[0]
            if "•" in findings_data:
                findings = findings_data.split("•")[1].split(".")[0].strip()
            else:
                findings = findings_data.strip()
        elif 'IMPRESSIONS:' in pageText:
            findings_data = str(pageText).split('IMPRESSIONS:')[1].split("Dr.")[0]
            if "•" in findings_data:
                findings = findings_data.split("•")[1].split(".")[0].strip()
            else:
                findings = findings_data.strip()
        elif 'IMPRESSION;' in pageText:
            findings_data = str(pageText).split('IMPRESSION;')[1].split("Dr.")[0]
            if "•" in findings_data:
                findings = findings_data.split("•")[1].split(".")[0].strip()
            else:
                findings = findings_data.strip()
```

ERROR HANDLING :

- Mostly I am using all the try-catch functionalities here , so there are less chances of facing any errors or getting into any exceptions.
- Also , When processing multiple files, there are a no. of errors like :

1. Sometimes, the same file is getting copied again and that's why it is completely redundant to process them as per our need, so I simply add them in the category of **Duplicate Files.**
2. I have given a fixed format to the Operations team for each and every file to be processed, i.e. the file should always be in the format of "**Id_Name**" . Now, if any file does not follow this format and use any unnecessary structures like having '.' , '-' etc. then I've categorised these files in **Naming Error Files** .
3. Now, Most of the files have these issues that the 'Id' present in the Filename and the id that i am extracting from the File, are not matching, so I've categorised them in **Id Mismatch Files** .
4. Now, If in case any of the field is left empty than I've categorised them as **Incomplete Data Files** .
5. And, atlast if there are any exception files than they are present in the **Exception Files** section.

- All the above errors , if present, than are added to thier respective Error folder , the Folder is named after the type of Error.
- And all type of errors are also added to a Error File , which is a text file, which stores all the details of the errors (if experienced / present) while processing the files.

- This is the function that adds the files to the respective folders and creates the Error File iff any error occured :

```python
def write_errors_to_file(naming_errors, duplicate_file, id_mismatch, incomplete_data, exception_files, output_folder_path
    # Ensure output_folder_path exists
    output_folder_path = Path(output_folder_path)
    output_folder_path.mkdir(parents=True, exist_ok=True)

    # Converting input folder path also into path object.
    input_folder_path = Path(input_folder_path)

    # Define the paths for the error subdirectories
    naming_error_folder = output_folder_path / "NamingErrorFiles"
    duplicate_files_folder = output_folder_path / "DuplicateFiles"
    id_mismatch_folder = output_folder_path / "IdMismatchFiles"
    incomplete_data_folder = output_folder_path / "IncompleteDataFiles"
    exception_files_folder = output_folder_path / "ExceptionFiles"

    # Define the error details file path
    error_details_file = output_folder_path / "ErrorDetails.txt"

    with open(error_details_file, "w") as file:
        file.write("The files that were processed were having the following errors:\n")
        file.write("================================================================\n\n")

        # Naming Errors
        if naming_errors:
            naming_error_count = len(naming_errors)
            file.write(f"NAMING ERRORS:\n\n")
            file.write(f"{naming_error_count} file(s) is/are having naming issues, so skipping them from processing :\nN
            for file_id, original_filename in naming_errors.items():
                file.write(f"Filename: {original_filename}\n")
                # Copy the file to the respective folder
                naming_error_folder.mkdir(parents=True, exist_ok=True)
```

- This function displays the warning message based on the errors iff occured :

```python
# Show warning message box if there are any errors
def show_warning_message(naming_errors, duplicate_file, id_mismatch, incomplete_data, exception_files):
    warning_message = ""

    if naming_errors:
        warning_message += f"{len(naming_errors)} file(s) had naming conflicts.\n"
    if duplicate_file:
        warning_message += f"{len(duplicate_file)} duplicate file(s) found.\n"
    if id_mismatch:
        warning_message += f"{len(id_mismatch)} file(s) had ID mismatches.\n"
    if incomplete_data:
        warning_message += f"{len(incomplete_data)} file(s) had incomplete data.\n"
    if exception_files:
        warning_message += f"{len(exception_files)} problematic file(s) encountered.\n"

    if warning_message:
        tk.messagebox.showwarning("Errors in File Processing", warning_message)

# Main function to handle all errors and genecate the file
```

- And , this is the main function that handles all these errors :

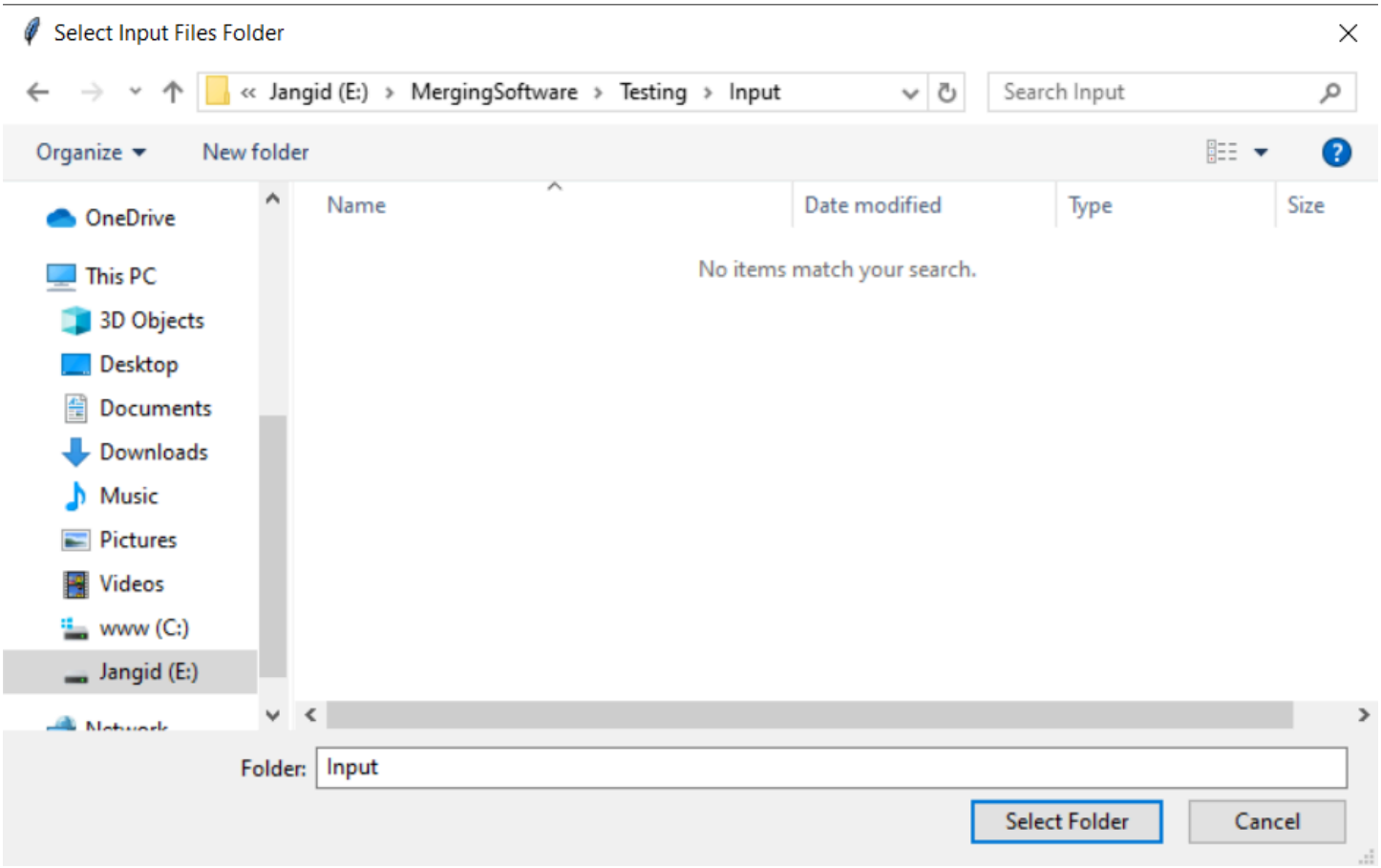
```python
# Main function to handle all errors and generate the file
def handle_all_errors(naming_errors, duplicate_file, id_mismatch, incomplete_data, exception_files, output_folder_path, i
    if naming_errors or duplicate_file or id_mismatch or incomplete_data or exception_files:
        # Write all errors to the error details file
        write_errors_to_file(naming_errors, duplicate_file, id_mismatch, incomplete_data, exception_files, output_folder_
        # Show the warning message box with a consolidated summary of errors
        show_warning_message(naming_errors, duplicate_file, id_mismatch, incomplete_data, exception_files)
    else:
        print("No errors found.")
```

BASIC WORKFLOW :

Now , mostly these are the basic workflow for each and every task you want our OTHM to perform :

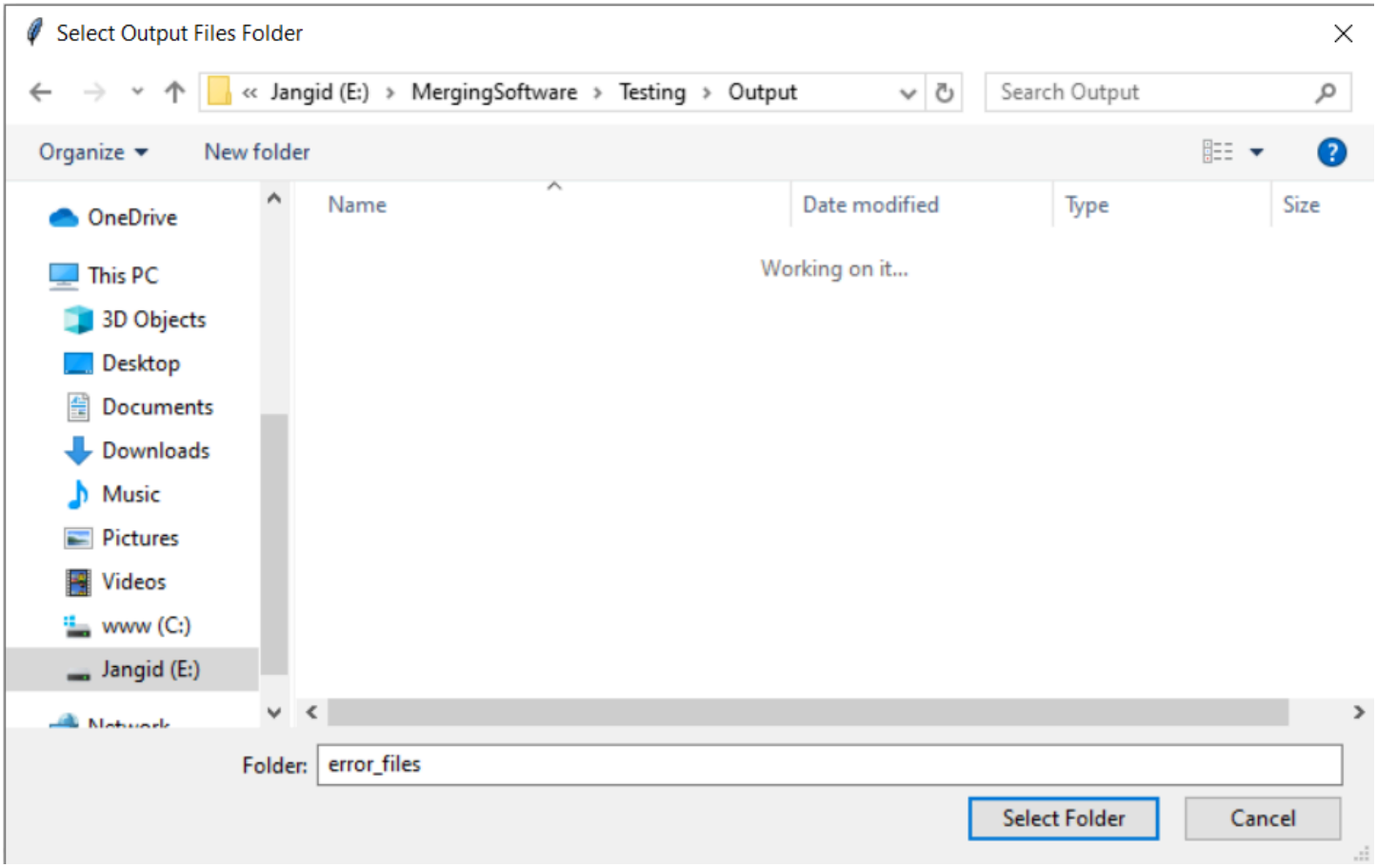
Asking For the Input Directory :

- Here, you will select the directory in which all the files are present that you want to process, or on those, you want to operate your required tasks.
- Just selecting the name is enough (No need to open the folder).



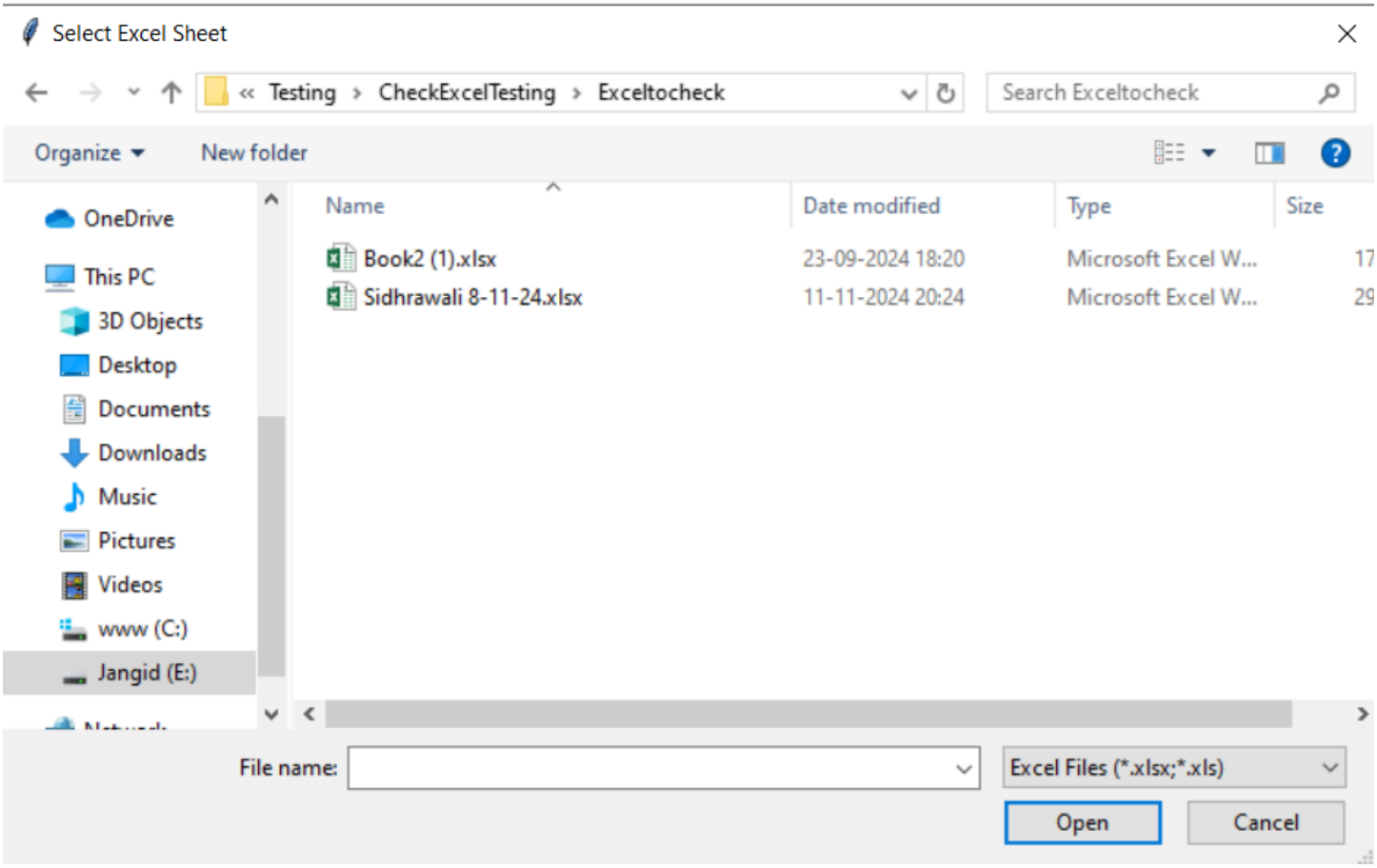
Asking For the Output Directory :

- Here, you are specifying that where you want your task / operation / processed results to be stored.
- Here also, just selecting the folder is enough, No need to open it.



Asking For the Excel Directory (iff any excel is present that is needed) :

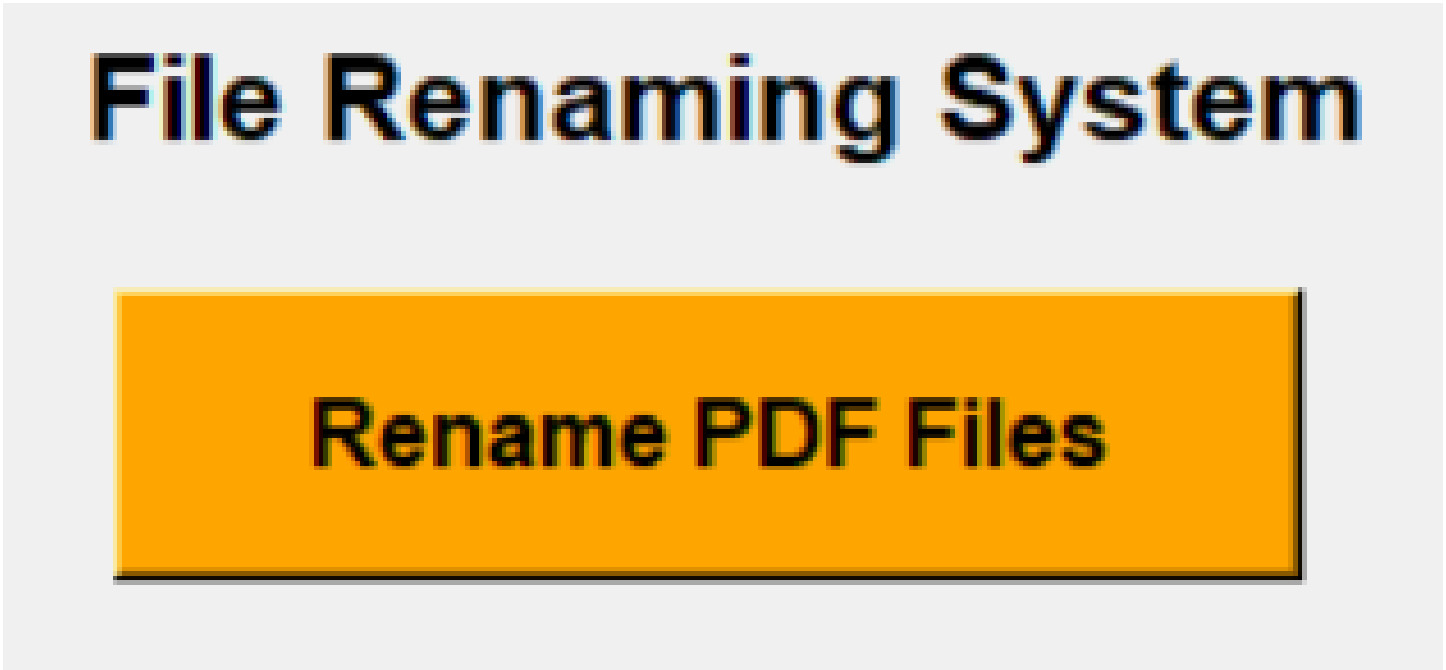
- Here, you have to select the excel file that you want to use in processing.
- Here, just clicking on the folder won't be enough , You have to select the particular excel sheet that you want to use.



INDIVIDUAL TASKS / OPERATIONS :

Rename Pdf Files

- This function renames all the files in terms of our desired format, i.e. ' ID_Name*' format.
- There can be other things too here like institution name etc. based on the requirements.



- Just click on the above button, it will ask for the Input and Output Directory.
- In the input directory add all the files in whatever format they are.
- Give the output directory where you want your results to be.
- **Functionality Working :**
  - The function will read the pdf files one by one.
  - It will identify the kind of report it is i.e. whether xray, ecg , pft etc.
  - Based on the type of report there are certain extraction conditions and techniques I've used which extracts / splits / separates mainly two things from the extracted complete text data , which is ' Patient ID ' and ' Patient Name '.
  - Now, each file is renamed with the respective names as ' Patient ID_Patient Name ' where these two are already extracted based on the modalities.

```
def rename_pdf_files():
    input_directory = filedialog.askdirectory(title="Select Input Directory")

    if input_directory:
        # Prompt user to select output directory
        output_directory = filedialog.askdirectory(title="Select Output Directory")

        if output_directory:
            input_dir = Path(input_directory)
            output_dir = Path(output_directory)
            output_dir.mkdir(parents=True, exist_ok=True)

            error_dir = output_dir / "error_files"
            error_dir.mkdir(parents=True, exist_ok=True)

            # List all PDF files in the input directory
            pdf_files = list(input_dir.glob("*.pdf"))

            if pdf_files:
                renamed_count = 0
                error_count = 0
                patient_id = ''
                patient_name = ''

                for pdf_file in pdf_files:
                    with open(pdf_file, 'rb') as file:
                        pdf_reader = PyPDF2.PdfReader(file)
                        if len(pdf_reader.pages) > 0:
                            first_page_text = ''
                            second_page_text = ''
                            if len(pdf_reader.pages) == 1:
```

Merge Pdf Files

- This function accepts all the files in this format only ' ID_Name ' i.e. there should be atleast an ID and a Name in the filename , both separated by the first underscore (_) in it.

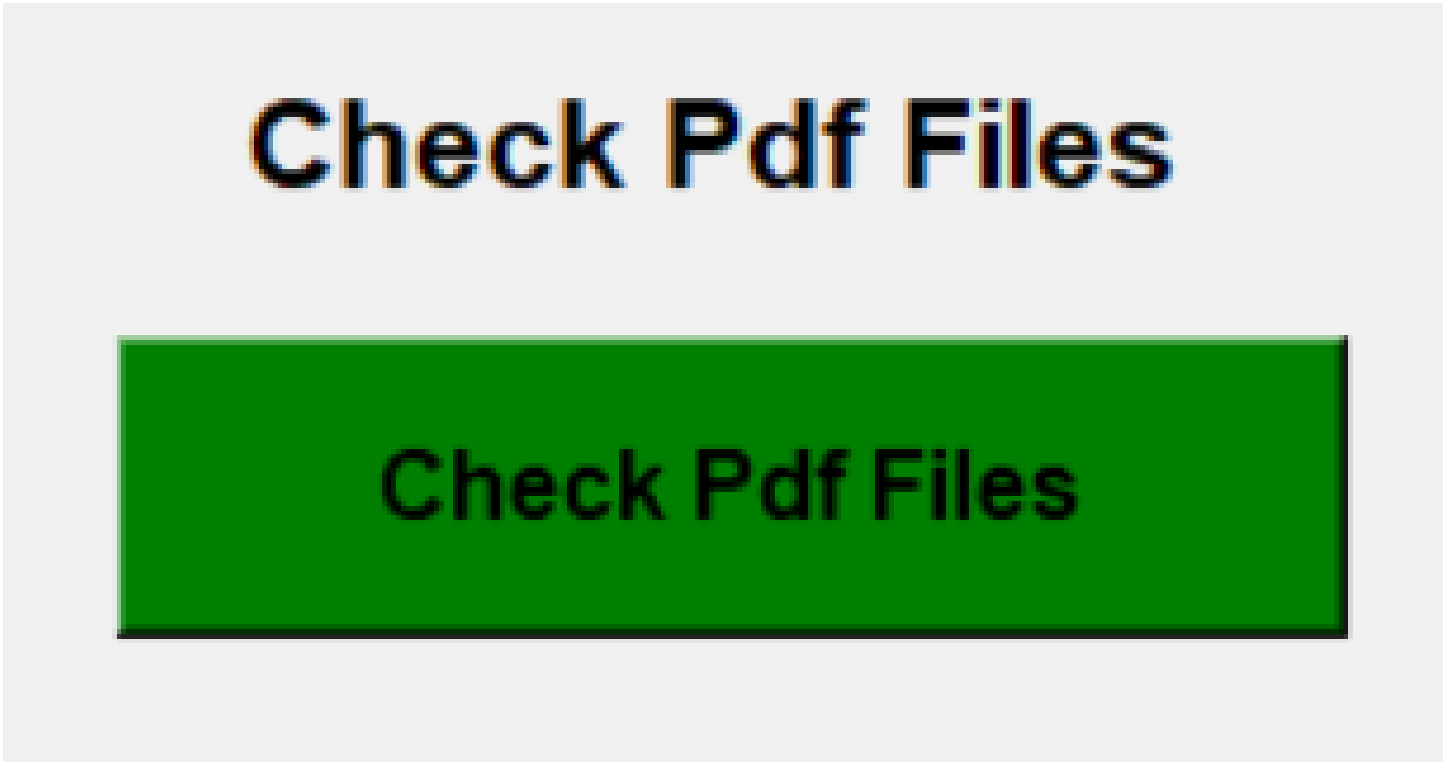
Merging For Individual

- Just click on the above button, it will ask for the Input and Output Directory.
- In the input directory add all the files in the format it is asked i.e. ' ID_Name' .
- Give the output directory where you want your results to be.
- **Note :**
  - I am converting each and every id in lowercase, So if the Operations Team by mistake write the unique Id in filename in UpperCase, then also , no need to worry , it will read it in lowercase (until unless all the characters are same (otherwise id mismatch)).
- **Functionality Working :**
  - Here, all type of reports are present of different patients altogether i.e. for a particular patient let's say if it has pft report, xray report , ecg report etc. than all of these are present here in the same directory.
  - Now, the function takes all the files that are having same id, and merges thier files in a single one.
  - It arranges them in a fixed format, by first identifying that which kind (type of modality) of report it is.
  - Then it does the same for each patient i.e. each unique id.

```python
# This is my code working till version 4 , and all the changes i've done is mentioned in the readme file - Himanshu.
def merge_redcliffe_pdf_files():
    # Prompt user to select input directory
    input_directory = filedialog.askdirectory(title="Select Input Directory")

    if input_directory:
        # Prompt user to select output directory
        output_directory = filedialog.askdirectory(title="Select Output Directory")

        if output_directory:
            pdf_dir = Path(input_directory)
            pdf_output_dir = Path(output_directory)
            pdf_output_dir.mkdir(parents=True, exist_ok=True)

            # List all PDF files in the input directory
            pdf_files = list(pdf_dir.glob("*.pdf"))
            print("These are the pdf files list :", pdf_files)

            # keys = set([str(file).split("\\")[-1].split("_")[0].lower() for file in pdf_files])
            keys = set()
            naming_errors = {}
            exception_files = {}

            for file in pdf_files:
                # This code is explicitly written to check the name format of the files - Himanshu.
                try:
                    original_filename = str(file).split("\\")[-1]
                    file_id = str(file).split("\\")[-1].split("_")[0].lower()
                    if "." in file_id:
                        naming_errors[str(file)] = original_filename
                        print(f"File {file} has incorrect naming format. Storing naming error: {original_filename}")
                    else:
```

Check Pdf Files

- This function accepts all the files in this format only ' ID_Name ' i.e. there should be atleast an ID and a Name in the filename , both separated by the first underscore (_) in it.
- **This function checks that for a particular patient, what reports/tests are done and what is the issue/errors (iff any) (id mismatch, name mismatch, etc.) in the respective patient test reports.**

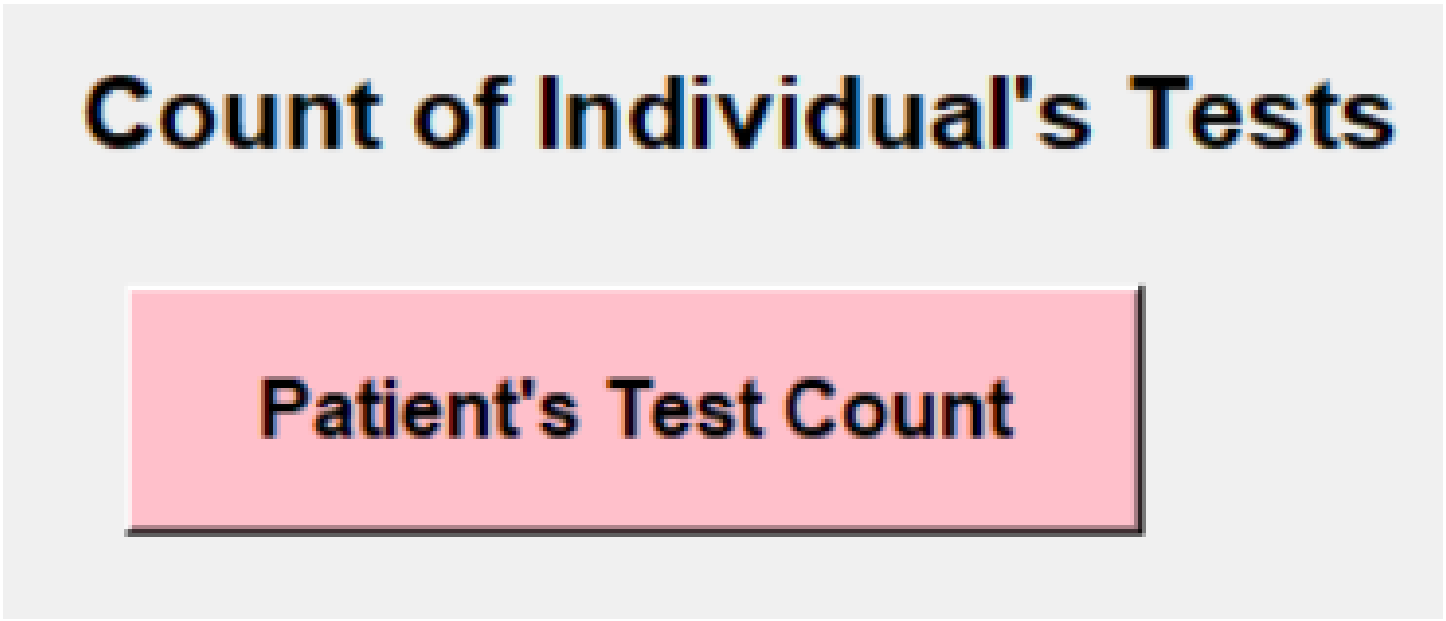


- When clicked, it will ask for the I/P Directory and O/P Directory as always.
- It will also ask another thing, which is the excel file (using which the comparison needs to be done).
- **Note :**
  - This separate file will be prepared by the Operations Team.
  - The user must select the excel sheet that needs to be used (**not the directory only**), otherwise it will throw an error.
  - Only '.xlsx' and '.xls' file types are allowed.
- **Functionality Working :**
  - I've created a modality match list, which stores the type of modalities any particular patient is having i.e. the tests that are done by a particular patient.
  - Created a excel sheet with some **Fixed Headers** (name, age, gender, id , study date/test date and report date) **(Note : Each time in this documentation the fixed headers will correspond to these 6)** and alongwith that 7 cell headers representing all the type of test reports we do (except Doctor Consultation) and 1 header representing the Problem list.
  - If any patient file contains the respective modality then for that particular patient it is marked as 'Yes' in the respective modality cell , else 'NO' .
  - I'm also changing the color of the cell for more easy identity, making 'Yes' to green and 'No' to Red.
  - **No worries for the casing of patient id and name, here also , because i am processing all as lowercase here too.**
  - All the exceptions are handled correctly.
  - It in final give the result in the form of a excel sheet.

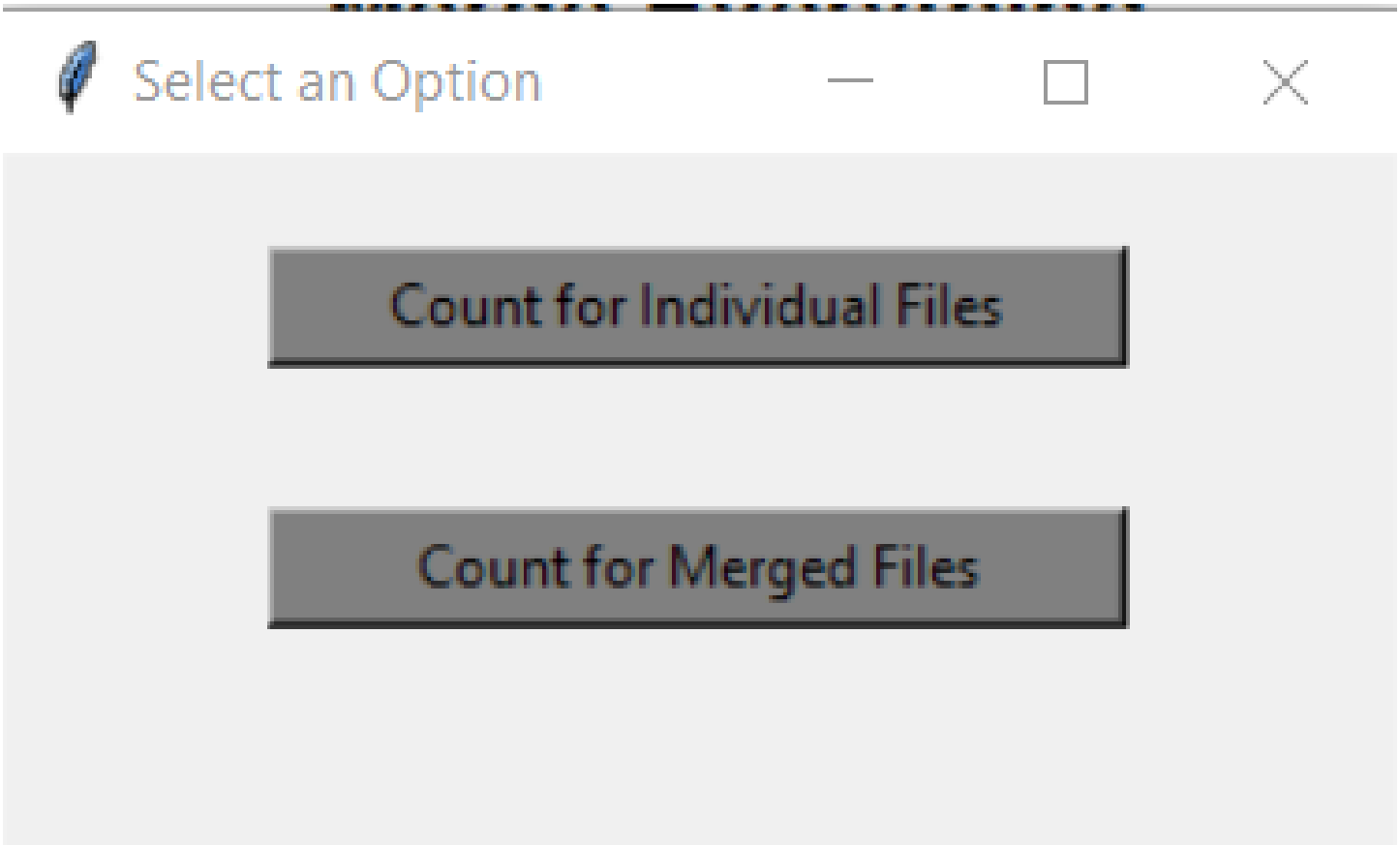
Count of Individual Tests

- This function gives the count of patient tests.

- It is logically separate from the Check PDF Files (if someone's thinking of similarity), as it doesn't need any excel file to compare with.
- You just give the files to it and it gives the individual tests in the form of excel sheet in the end.



- **Important :**
  - **I have created another tkinter window here, which asks for the type of files you need to process, as the logic is separate for both.**
  - **It asks either you want to process 'All the individual files for any patient' OR 'All the Merged Files Of Patients'.**



```
# This is my new function which will just give me the count or tell me whether what test's are done for a particular pati
def count_of_tests_for_individual_patient():
    # As of now , i am not able to use a simple dialogbox to make the user select option from my window directly.
    # I've also thought of modifying the Yes and NO option in the ' messagebox.askquestion() ', but i guess I am not able
    # If i get any other way to change these than i will use that part directly instead of creating a separate dialogbox.

    # This is the code that just opens the separate window for selecting the option.
    # Creating a new Tkinter window for option selection.
    optionWindow = tk.Toplevel(window)
    optionWindow.title("Select an Option")
    optionWindow.geometry("300x150")

    # Centering the optionWindow on the main window.
    window_width = window.winfo_width()
    window_height = window.winfo_height()
    window_x = window.winfo_x()
    window_y = window.winfo_y()

    # Calculating the position of the option window to be centered
    option_window_x = window_x + (window_width // 2) - 150  # 150 is half the width of option window (300x150)
    option_window_y = window_y + (window_height // 2) - 75  # 75 is half the height of option window

    optionWindow.geometry(f"300x150+{option_window_x}+{option_window_y}")

    def option_selected(option):
        nonlocal selected_option
        selected_option = option
        optionWindow.destroy()  # Close the option selection window

    selected_option = None  # Variable to store the selected option

    # Add buttons for options
```

- **Note :**
  - In this function I am completely making use of the Helper Functions (because from this functionality onwards, I've created the Helper's).
  - Most probably I've left the created data structures to the hands of the Python Garbage Collector.
  - Since, I am using a '.exe' so , when closed, it will automatically remove all the created data structures.
  - **The above Data Structure Handling needs to be optimized.**
- **Functionality Working :**
  - It will first ask for the option selection when clicked.
  - As per the selection it will further process the logic.
  - It asks for the I/P and O/P directory before.
  - It creates some things before proceeding selected option wise, as it is common for both the options like making the keys set , error handling dictionaries, respective headers etc.

```python
    # If no option was selected, show a warning and exit
    if selected_option is None:
        messagebox.showwarning("No Option Chosen", "You must select an option to continue with Excel Generation!")
        return

    # Call select_folders to get both input and output folder paths
    input_folder_path, output_folder_path = select_folders()
    if not input_folder_path or not output_folder_path:
        return

    # Now, this will give me a list of only pdf files as the glob will only give me these, and then it will store them e
    pdf_files = list(Path(input_folder_path).glob("*.pdf"))
    print(pdf_files)
    # Getting the no. of files that i've processed.
    num_files_processed = len(pdf_files)
    # Later, i will convert the path object in the binary format so that i can read it using our reader and manipulate m

    # Collecting keys from the file names (extracted from the first part of the filename before the underscore)
    # I will use these in case if in any file the id is missing for that unique patient.
    keys = set()
    naming_errors = {}
    exception_files = {}
    incomplete_data = {}
    duplicate_file = {}
    id_mismatch = {}

    # Making the modalities a set to store all the modalities for a particular id/key.
    modalities = set()

    # Initializing a empty dictionary to just store the patient details every time any file is processed.
```

```python
    # Initializing a empty dictionary to just store the patient details every time any file is processed.
    patient_details = create_patient_details()
    # patient_details= {'patient_id': None,'patient_name': None,'patient_age': None,'gender': None,'test_date': None,'re


    # defining some unwanted phrases for later use.
    # i've defined in the conditional function, so i might not require here now.
    # unwanted_phrases = ["Referrer Dr.","left ear","RECORDERS & MEDICARE SYSTEMS","OPTOMETRY","ECG","VITALS","RBC Count

    # Initializing the patient_data dictionary before looping through each and every file so that i can use it to fill t
    patient_data = creating_or_emptying_the_patient_data_dictionary()

    # Excel Workbook Setup.
    wb = Workbook()
    ws = wb.active

    # As of now, i have total 15 headers only.
    headers = ['SERIAL NO.', 'PATIENT ID', 'PATIENT NAME', 'AGE', 'GENDER', 'STUDY DATE', 'REPORT DATE',
               'XRAY', 'ECG', 'AUDIOMETRY', 'OPTOMETRY', 'VITALS', 'PFT', 'PATHOLOGY', 'OTHERS']

    # Adding headers in excel
    for col_num, header in enumerate(headers, 1):
        cell = ws.cell(row=1, column=col_num)
        cell.value = header
        cell.font = Font(bold=True)

    # Initializing the serial number for patient data.
    serial_no = 1
```

- If Option 1 is selected i.e. **Count For Individual Files**:
  - It then process files one by one.
  - If any file id is new i.e. not yet processed, than it adds the file_id to the keys, and creates a patient_details dictionary and a modalities set for that particular key.

        - If any file processed again , matches the key present in keys set, than instead of creating new dictionaries and modalities, it just uses the last created one and appends the data to the same accordingly.
        - Here I have used mostly all the helpers in a creatively optimal way (Much details I expect the new developer to understand from the code 😄)
        - All the errors are handled by the helper function.
    - If Option 2 is selected i.e. **Count For Merged Files** :
        - It also processes the files one by one.
        - As the merged file itself has all the type of reports, than the logic is much easier here.
        - It checks the file_id and then appends it to the keys.
        - Processes each page one by one, if modality matches than adds to the modalities set.
        - Once a file is processed than it clears the modalities and the patient_data.
    - Common For Both :
        - For both the error handling function is same.
        - The output is a excel sheet containing the basic **Fixed Headers** and the names of each Modality (all & types) in the next cells, along with the Serial No's too.
        - If any modality is present than it gets marked with 'Present' with Green cell colour , if not then 'Not Present' with Red cell color.
    - **Note :**
        - **The key note here is that it also extracts the Fixed Header data from the files.**
        - **But making optimisation, I've created one thing if the respective modality is identified and Fixed Headers are already extracted from one than it doesn't need to extract again in the next modality identification.**

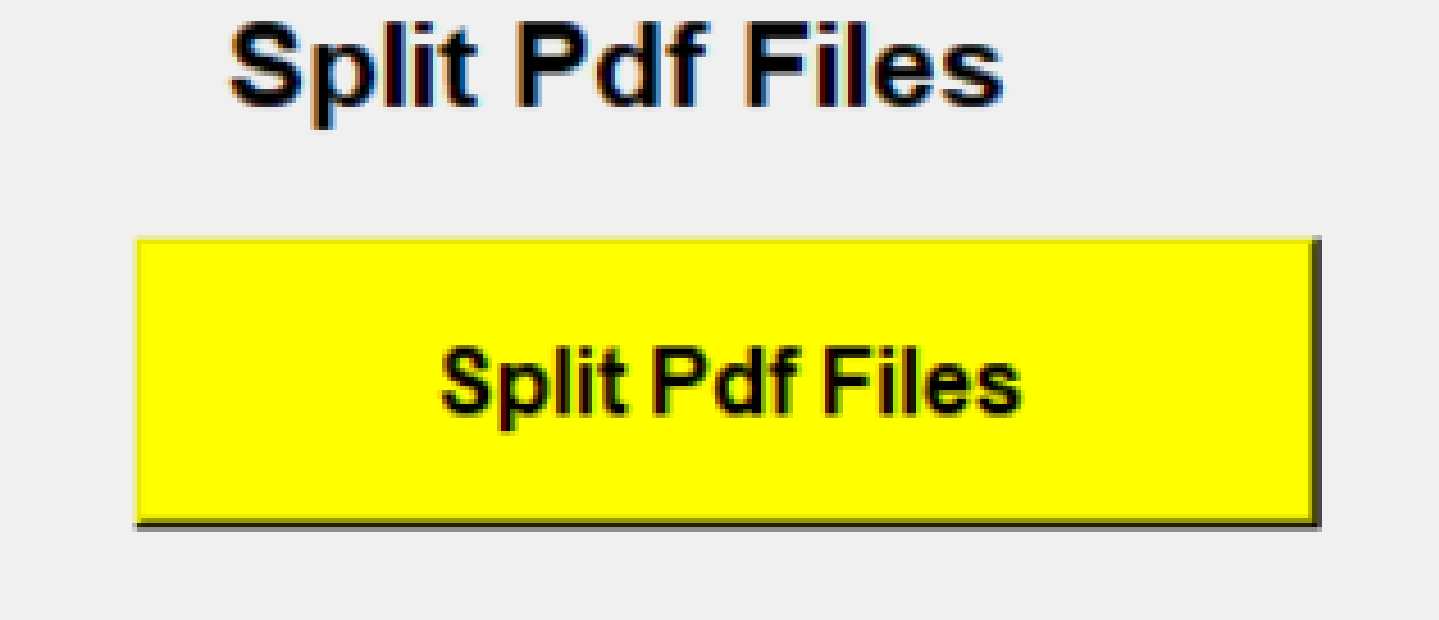
Merge Everthing

- It just merges each pdf file you give it, one by one, in the order they are present in the directory.

- Just give the I/P and O/P directory, and get the expected results.

```python
def merge_all():
    # Prompt user to select input directory
    input_directory = filedialog.askdirectory(title="Select Input Directory")

    if input_directory:
        # Prompt user to select output directory
        output_directory = filedialog.askdirectory(title="Select Output Directory")

        if output_directory:
            pdf_dir = Path(input_directory)
            pdf_output_dir = Path(output_directory)
            pdf_output_dir.mkdir(parents=True, exist_ok=True)

            # List all PDF files in the input directory
            pdf_files = list(pdf_dir.glob("*.pdf"))
            if pdf_files:
                # Create a PdfFileMerger object
                merger = PdfMerger()
                for pdf_file in pdf_files:
                    merger.append(pdf_file)
                output_file_path = pdf_output_dir / "merged_file.pdf"

                # Write the merged PDF to the output file
                with open(output_file_path, "wb") as output_file:
                    merger.write(output_file)

                print(f"Merged PDF saved to: {output_file_path}")
                tk.messagebox.showinfo("PDF Merger",f"Total {len(pdf_files)} PDF files merged into one PDF successfully!"
            else:
                tk.messagebox.showinfo("No PDF Files", "No PDF files found in the input directory.")
        else:
```

Split Pdf Files

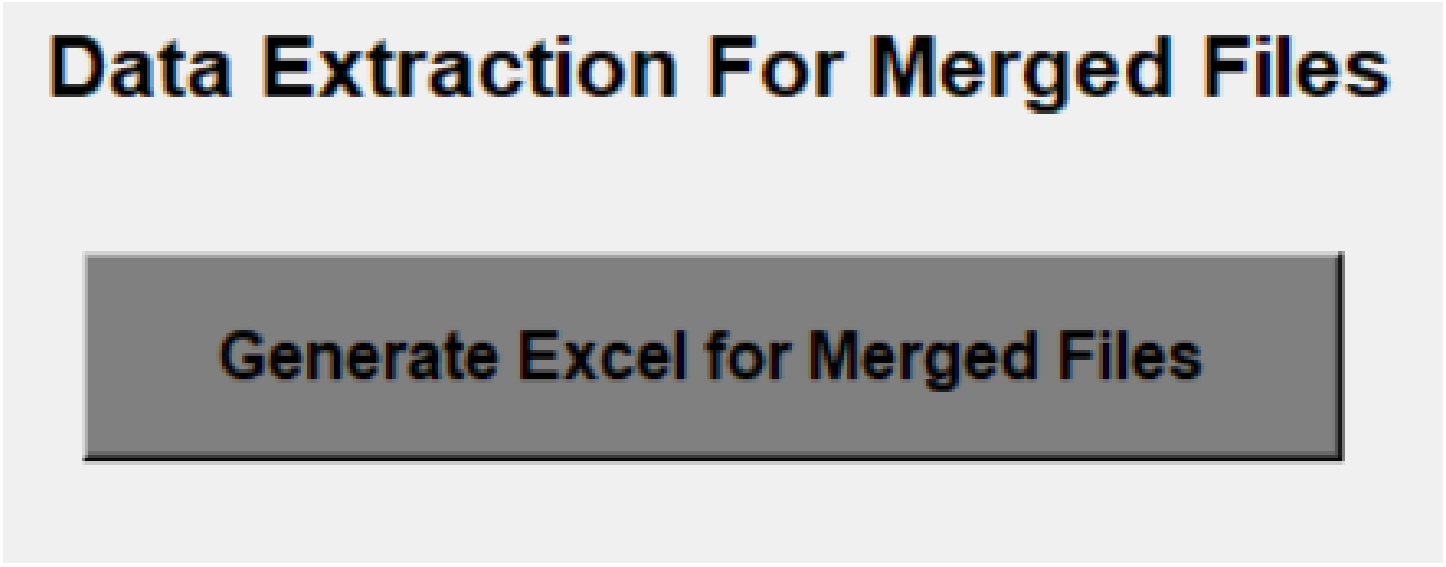


- It does just opposite of the Merge Everything.
- It gives any file given separated in individual pages.
- **Note :**
  - It can be optimised in a way that it splits the files in individual modality reports.

```python
def split_patient_file():
    input_directory = filedialog.askdirectory(title="Select Input Directory")

    if input_directory:
        output_directory = filedialog.askdirectory(title="Select Output Directory")

        if output_directory:
            pdf_dir = Path(input_directory)
            pdf_output_dir = Path(output_directory)
            pdf_output_dir.mkdir(parents=True, exist_ok=True)

            pdf_files = list(pdf_dir.glob("*.pdf"))

            if pdf_files:
                for input_pdf_path in pdf_files:
                    try:
                        # Open the merged PDF file
                        with open(input_pdf_path, 'rb') as pdf_file:
                            pdf_reader = PyPDF2.PdfReader(pdf_file)

                            # Create a subdirectory for the patient if it doesn't exist
                            patient_id = sanitize_filename(input_pdf_path.stem)
                            patient_dir = pdf_output_dir / patient_id
                            patient_dir.mkdir(parents=True, exist_ok=True)

                            # Loop through each page in the PDF and save them as individual PDF files
                            for page_number in range(len(pdf_reader.pages)):
                                pdf_writer = PyPDF2.PdfWriter()
                                pdf_writer.add_page(pdf_reader.pages[page_number])

                                # Extract text data from the page to determine the modality
```

Generate Excel For Merged Files

- It gives each modality separated data in forms of excel sheets.
- For example if there are multiple patients then if we need separate excels like the patients data who did Xray Test, Optometry Test , and similarly remaining.
- Than based on the modalities present it appends data to the respective excel , Ex. if Audiometry report than it gets appended to the Audiometry Excel.



- **Note :**
  - In this function I am completely making use of the Helper Functions .

    - I've created a dictionary which provides the respective excel sheet accordingly.
    - **For the next developer, test it again and fix the logic a bit for much optimal results.**
- **Functionality Working :**
    - All the basic things are present, as similar to the count of tests functionality.
    - It processes the files one by one, just checks the type of modality.
    - And also checks whether the same id is already appended to the respective excel, if yes than skips it.
    - All the error handling are similar.

```
def generate_excel_for_merged_files():
    # Call select_folders to get both input and output folder paths
    input_folder_path, output_folder_path = select_folders()
    if not input_folder_path or not output_folder_path:
        return
    # This will give me a list of only pdf files as the glob will only give me these, and then it will store them each i
    pdf_files = list(Path(input_folder_path).glob("*.pdf"))
    print(pdf_files)
    # Getting the no. of files that i've processed.
    num_files_processed = len(pdf_files)
    # Later, i will convert the path object in the binary format so that i can read it using our reader and manipulate m

    # Collecting keys from the file names (extracted from the first part of the filename before the underscore)
    # I will use these in case if in any file the id is missing for that unique patient.
    keys = set()
    naming_errors = {}
    exception_files = {}
    incomplete_data = {}
    duplicate_file = {}
    id_mismatch = {}
    # Set to store modalities encountered for each unique file_id
    unique_file_id_set = set()

    # Making the modalities a set to store all the modalities for a particular id/key.
    modalities = set()

    # Initializing a empty dictionary to just store the patient details every time any file is processed.
    patient_details = create_patient_details()

    # Logic for merged files
    print(f"Input Folder: {input_folder_path}")
```

Generate Excel For Individual Files

- It also gives each modality separated data in forms of excel sheets.
- Just the difference is we give non merged files here (As the name suggests).

Data Extraction For Individual Files

- **Note :**
  - In this function I am **not at all** making use of the Helper Functions .
  - I've just altered the previous logic.
  - This has completely boiler plate codes, that can be easily optimised.
  - **It appends the data irrespective of the case that either it is already appended or not.**
- **Functionality Working :**
  - It gets the input directory files , and then processes them one by one.
  - Mostly the files are single page and double page, so if it is single page than it passes the first page data as it is.
  - If it is having 2 pages ,than it passes the second page.
  - And if more than 2 pages, then the first page is passed , and all the logic is created accordingly.
  - It checks the type of modality , altogether extracts the required details , and update the respective modality count from 0 .
  - If any modality count is greater than 0, than creates/appends the respective data to the respective excel.
  - **No proper error handling like in others , Definitely needs to be optimised.**

```python
def generate_excel_for_individual_files():
    input_directory = filedialog.askdirectory(title="Select Input Directory")

    if input_directory:
        output_directory = filedialog.askdirectory(title="Select Output Directory")

        if output_directory:
            input_dir = Path(input_directory)
            output_dir = Path(output_directory)
            output_dir.mkdir(parents=True, exist_ok=True)

            error_dir = output_dir / "error_files"
            error_dir.mkdir(parents=True, exist_ok=True)

            pdf_files = list(input_dir.glob("*.pdf"))
            error_count = 0

            # Define unwanted phrases
            unwanted_phrases = ["Referrer Dr.", "left ear", "RECORDERS & MEDICARE SYSTEMS", "OPTOMETRY", "ECG", "VITALS"

            # There are mainly 8 files here (leaving multiple formats of an individual) :
            patient_data_ecg = []
            patient_data_ecg1 = []
            patient_data_pft = []
            patient_data_xray = []
            # Adding the remaining files too.
            patient_data_opto = []
            patient_data_vitals = []
            patient_data_audio = []
```

OPTIMISATIONS

1. The code contains a lot of Boiler Plate (Redundant) Codes, so make it more optimised.
2. The Helper Functions I've created are not used in all the functions, so use it in other functions too for much smoother workflow.
3. Introduce the Error Handling in other functionalities too as I've developed for the Test Count Function or Generate excel functionality.
4. Handle all the data structures created in proper way, so we can increase the speed of the workflow (Make proper use of the Garbage Collector).
5. Mostly, the Operation Team introduces new formats of the pdf's , so just append the respective format to the respective functionalities, instead of removing the previous ones.
6. Most of the functionality logics are stucking in unnecessary loop or processing. Identify those and fix them.
7. Introduce more Error Handles, so that the tasks are more brief and helpful.
8. Introduce ways to append the Doctor Consulation, Smart Reports etc. too.
9. Make sure to handle each and every exception so that no Operation Member face any type of Fucntionality Error.
10. Optimise the code in every possible way, a lot of efficient modifications are needed here.

END OF PAGE.

All the Best for Future Development .....