Base Line

Backlogs and Sprint

MILESTONE 2					
Sprint #1 (Week #8) Backlog					
Sprint #	**Finish Date**	**Backlogs**	**Description**	**Assigned**	**Status**
1	11-Oct-24	Architecture Diagram	Fix the architecture diagram according to professor's feedback	Brennon Duffy	Done
	11-Oct-24	Appealing User Profiles	Implement organized profile structure	Vadim Pidoshva	Done
	11-Oct-24	Appealing Combined Data	Create appealing combined data fieds for interaction	Vadim Pidoshva	Done
	11-Oct-24	Update README	Update project documentation with all recent changes	Vadim Pidoshva	Done
	10-Oct-24	Console Response/Logging	Ensure that the system's console provides appropriate responses	Vadim Pidoshva	Done
	10-Oct-24	Updated Comments in Code	Review and update comments across the codebase for clarity	Vadim Pidoshva	Done
	14-Oct-24	Use Case Diagram	Fix the use case diagram according to professor's feedback	Lance Reed	Done
	7-Oct-24	Setup Profiles	Design and implement user profiles	Vadim Pidoshva	Done
	7-Oct-24	Duplicates in Combined Data	Remove and handle duplicates in the combined data	Vadim Pidoshva	Done
Sprint #2 (Week #9) Backlog					
Sprint #	**Finish Date**	**Backlogs**	**Description**	**Assigned**	**Status**
2	17-Oct-24	Test/QA	Conduct unit testing and update test cases according to professor's feedback	Rafael Almeida, Vadim Pidoshva	Done
	18-Oct-24	Demo Video	Create a demo video showcasing the project's functionality	Rafael Almeida	Done
	18-Oct-24	Class Diagram	Fix a class diagram according to professor's feedback	Logan Singleton	Done
	18-Oct-24	SRS (Software Requirement Specification)	Fix SRS acording to professor's feedback	Logan Singleton, Brennon Duffy	Done
	None	Unmatched Data Logic	Handle logic for unmatched data in the system	Brennon Duffy	In Progress, Awaiting Customer Data
	18-Oct-24	Prepare Base_Line.pdf File	Prepare the baseline PDF file for documentation	Lance Reed, Vadim Pidoshva	Done
	18-Oct-24	Prepare Prototype	Create and prepare the project prototype	Vadim Pidoshva	Done
	10/18/24	Reset Data Frames Upon Closing a Window	Ensure that data frames are reset upon window closure	Brennon Duffy	Done
	10/18/24	Update Private Variables in app.py	Modify private variables for security and optimization	Logan Singleton	Done
	18-Oct-24	Customer Feedback	Implement feedback from customers in the current iteration	Rafael Almeida	Done

Transcript:

MILESTONE 2

Sprint #1 (Week #8) Backlog

Sprint #	Finish Date	Backlogs	Description	Assigned	Status
1	11-Oct-24	Architecture Diagram	Fix the architecture diagram according to professor's feedback	Brennon Duffy	Done
	11-Oct-24	Appealing User Profiles	Implement organized profile structure	Vadim Pidoshva	Done
	11-Oct-24	Appealing Combined Data	Create appealing combined data fields for interaction	Vadim Pidoshva	Done
	11-Oct-24	Update README	Update project documentation with all recent changes	Vadim Pidoshva	Done
	10-Oct-24	Console Response/Logging	Ensure that the system's console provides appropriate responses	Vadim Pidoshva	Done
	10-Oct-24	Updated Comments in Code	Review and update comments across the codebase for clarity	Vadim Pidoshva	Done
	14-Oct-24	Use Case Diagram	Fix the use case diagram according to professor's feedback	Lance Reed	Done

	7-Oct-24	Setup Profiles	Design and implement user profiles	Vadim Pidoshva	Done
	7-Oct-24	Duplicates in Combined Data	Remove and handle duplicates in the combined data	Vadim Pidoshva	Done

Sprint #2 (Week #9) Backlog

Sprint #	Finish Date	Backlogs	Description	Assigned	Status
2	17-Oct-24	Test/QA	Conduct unit testing and update test cases according to professor's feedback	Rafael Almeida, Vadim Pidoshva	Done
	18-Oct-24	Demo Video	Create a demo video showcasing the project's functionality	Rafael Almeida	Done
	18-Oct-24	Class Diagram	Fix a class diagram according to professor's feedback	Logan Singleton	Done
	18-Oct-24	SRS (Software Requirement Specification)	Fix SRS according to professor's feedback	Logan Singleton, Brennon Duffy	Done
	None	Unmatched Data Logic	Handle logic for unmatched data in the system	Brennon Duffy	In Progress, Awaiting Customer Data
	18-Oct-24	Prepare Base_Line.pdf File	Prepare the baseline PDF file for documentation	Lance Reed, Vadim Pidoshva	Done
	18-Oct-24	Prepare Prototype	Create and prepare the project prototype	Vadim Pidoshva	Done
	10/18/24	Reset Data Frames Upon Closing a Window	Ensure that data frames are reset upon window closure	Brennon Duffy	Done
	10/18/24	Update Private Variables in app.py	Modify private variables for security and optimization	Logan Singleton	Done
	18-Oct-24	Customer Feedback	Implement feedback from customers in the current iteration	Rafael Almeida	Done

Group 4: Vadim Pidoshva, Logan Singleton, Rafael Almeida, Brennon Duffy, Lance Reed

Software Requirement Specifications

Functional Requirements

1. **Read Excel Files**
   The application shall allow the user to select and load two Excel files through a file dialog interface.
2. **Validate Excel File Formats**
   The application shall verify that each selected file has a `.xls` or `.xlsx` extension before reading its content.
3. **Combine Data**
   The application shall merge data from the two loaded Excel files by matching records where `Mother_First_Name`, `Mother_Last_Name`, and `Child_Date_of_Birth` are identical after normalization.
4. **Display Combined Data**
   The application shall present the combined data in a list view, displaying `Mother_IDs`, `Child_Names`, and `Date_of_Birth` for each matched record.
5. **Check Minimum Files Requirement**
   The application shall disable the data combination functionality until both Excel files are successfully loaded.
6. **Normalize Name Fields**
   Before merging data, the application shall process `Mother_First_Name` and `Mother_Last_Name` by removing all spaces, converting all letters to lowercase, and removing all non-alphanumeric characters.
7. **Error Handling**
   The application shall display specific error messages in message boxes if an Excel file cannot be read, if the file format is invalid, or if data cannot be combined due to mismatches.
8. **Child Profile Display**
   When the user double-clicks on a child's name in the list view, the application shall open a new window displaying a detailed profile of the child, including all associated data from the combined records.
9. **Save Combined Data**
   After completing the merge process, the application shall provide an option for the user to save the combined data to a new Excel file named `combined_matched_data.xlsx`.
10. **Display Unmatched Entries**
    The application shall show a list of unmatched records, identified by `Mother_ID`, in a designated area of the user interface to assist the user in identifying missing data.

Non-Functional Requirements

1. **Usability**
   The application shall include "Read Excel Files" and "Combine Excel Files" buttons on the main interface. When either button is pressed, the corresponding operation shall complete within 3 seconds for Excel files containing up to 1,000 rows each.

2. **Performance**
   The application shall complete the data reading and merging process within 5 seconds when processing two Excel files, each containing up to 1,000 rows of data.
3. **Error Tolerance**
   The application shall remain operational without crashing when encountering missing files, invalid file formats, or data mismatches, and shall inform the user of the specific issue via error messages.
4. **Portability**
   The application shall be compatible with Windows 10 (and later versions) and macOS Catalina 10.15 (and later versions) without requiring additional software installations beyond standard system updates.
5. **Maintainability**
   The application's source code shall be organized using the command design pattern, with each button action implemented as an extensible command class, allowing new features to be added without modifying existing code structures.

Group 4: Vadim Pidoshva, Logan Singleton, Rafael Almeida, Brennon Duffy, Lance Reed

Customer Feedback

User interface: met expectations(affordances in place).

- Excel files are accessible through User interface
- Combined- list is functional.
- The search option is functional.

To do:

- Find duplicates and flag them on a column labeled "duplicates."
- Combined-list shall be presented in excel format.
- Exclude columns: IDs, state, county, (maybe tobacco use) to allow enough columns to fit in landscape format when printing in a standard sized page.

Filter list: expectation not met

To do:

- Filtered-list shall be properly labeled and saved with the proper name and as an excel file.
- Example: Residual list shall be saved and labeled as "Non-Medicaid list."

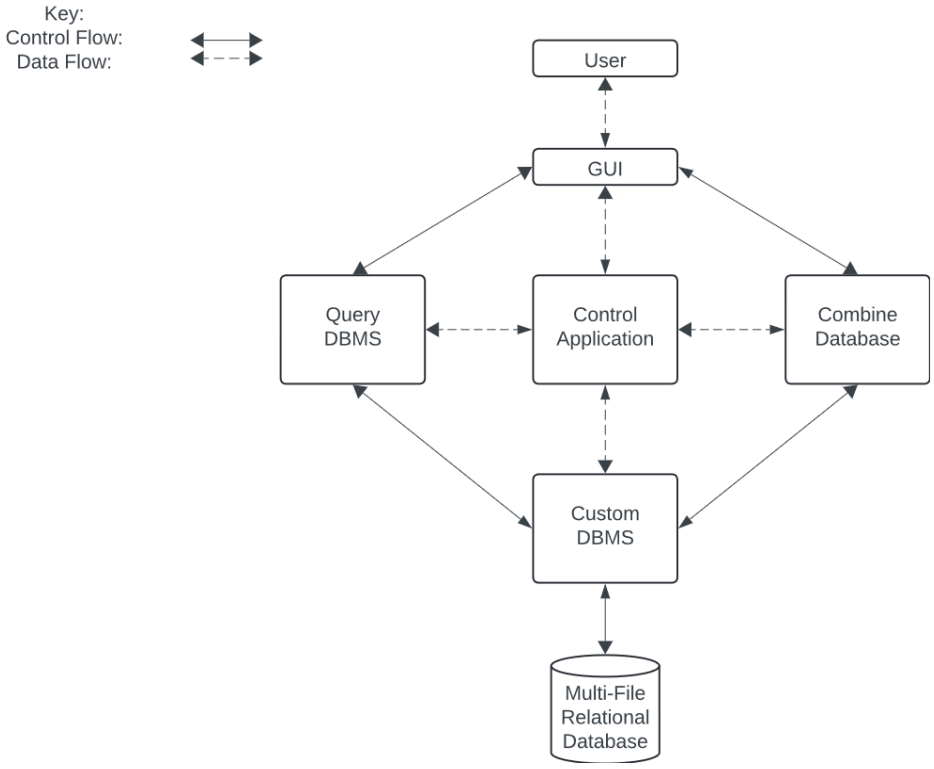
Print list: expectation not met

To do:

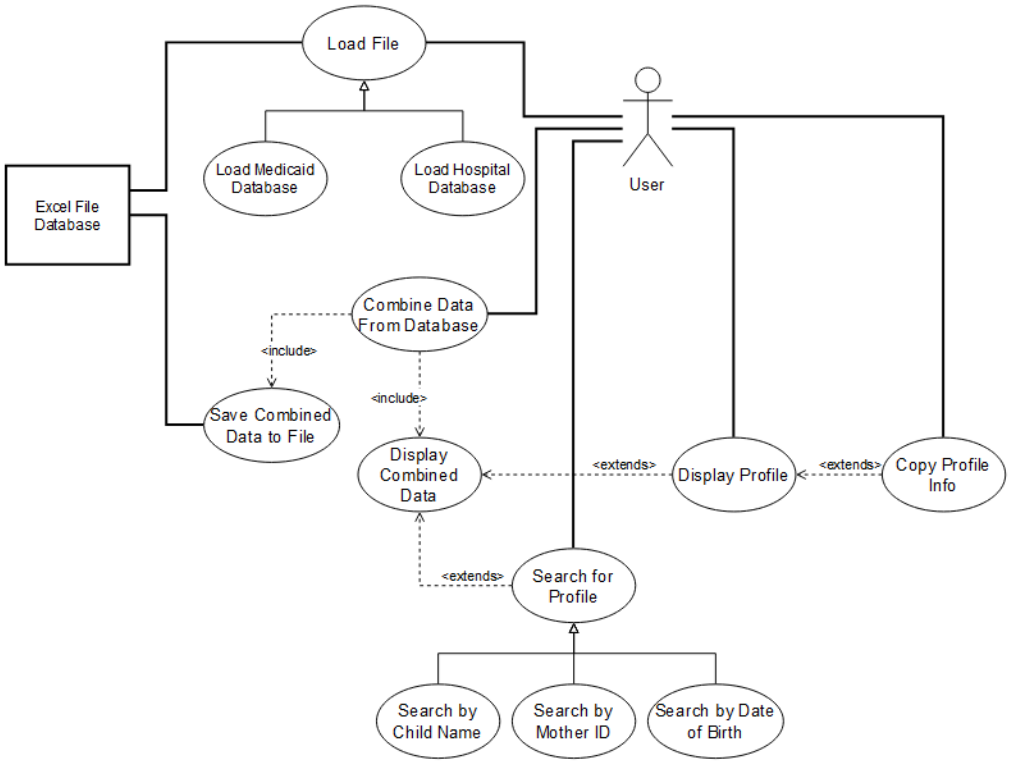
- The print function shall be made available after combining and filtering of lists.
- Lists to be printed in excel format.

Design

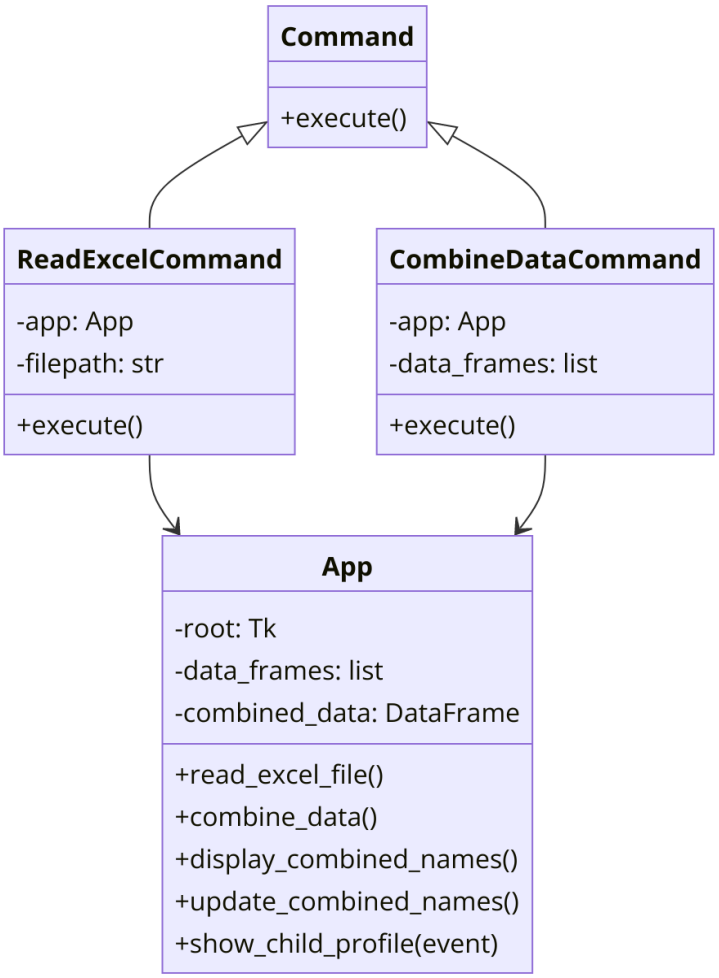
Architecture Diagram:



Use Case Diagram:



Class Diagram:



Testing Plan and Quality Assurance

Control Flow Graph for Read_excel_file(self):

Node 1: <start>
Node 2: Command = ReadExcelCommand(self)
Node 3: Self.invoker.add_command(command)
Node 4: data_frame = command.execute()
Node 5: if data_frame is not None
Node 6: self.data_frames.append(data_frame)
Node 7: else// maybe

Total Nodes for Read_excel_file

7

Edges Node1 to Node2 , Node2 to Node3, Node4 to Node5, Node5 to Node6, Node5 to Node7.

Total Edges for Read_excel_file

5

Cyclomatic Complexity Calculation for function Read_excel_file

7 – 5 + 2(1) = 4

Nodes for ReadExcelCommand.execute

Node 1: Start of execute()
Node 2: Prompt user for file selection
Node 3: Check if self.filepath is empty or None
- **Node 3a**: Show error message: "No file selected."
- **Node 3b**: Return None

Node 4: Try to read the file: data = pd.read_excel(self.filepath)
Node 5: Check if data is empty
- **Node 5a**: Show warning: "The selected Excel file is empty."
- **Node 5b**: Return None

Node 6: Replace spaces in column names
Node 7: Return data

Basis Path Set Read Excel

• **Path1:** Valid ,saved, saves the instance of ReadExcelCommand to command.
Node sequence: Node1 to Node2 to Node3 to Node4 to Node5 to Node6 or to Node7
• **Path2:** Valid command object is saved in a queue.
Node sequence: Node1 to Node 2 to Node3 to Node4 to Node5 to Node6 or to Node7
• **Path3:** Invalid, file format error.
Node sequence: Node1 to Node2 to Node3 to Node4 to Node5 to Node6
• **Path4:** Valid execute when reading the data , it reads by using pd.read_excel(self.filepath), it places underscores in spaces in columns of names , when finished it returns the data frame which contains the data from the Excel File.
Node sequence: Node1 to Node2 to Node3 to Node4 to Node5 to Node6
• **Path5:** Invalid , file is invalid when there is none data in the excel file, the data_frametriggers else statement.
Node sequence: Node1 to Node2 to Node3 to Node4 to Node5a to Node 5b return None.

Test Cases Read Excel

• **Test Case 1 User Cancellation**:
  - Description: Test how the program responds to canceling the program.
  - Input: User cancels the process of merging lists.
  - Expected Outcome: selection dialog, the filedialog.askopenfilename() will return an empty string or None, leading to self.filepath being None

• **Test Case 2** : Invalid,
  - Description : Test how the program responds to opening files of different types.
  - Input: User picks a format other than excel
  - Expected Outcome: self.filepath is None or an empty string. An error message"Nofile selected" is displayed.

• **Test Case 3:** Valid
  - Description: Test how the program responds to button commands buttons.
  - Input: User selects a button command to input a file.
  - Expected Outcome: The command is stored and executed.

  to store commands and execution of reading the files.

• **Test Case 4: Invalid - Handling Empty Data**
  - Description: Test how the program responds to an empty Excel file.
  - Input: User selects an empty Excel file.
  - Expected Outcome: The execute() method returns None, and the message. "The selected Excel file is empty." is logged.

• **Test Case 5: Valid:**
  - Description: Test how the program responds to reading valid data.
  - Input: User selects excel document and the program reads it.

- Expected Outcome: it places underscores in spaces in columns of names , and returns the data frame which contains the data from the Excel File.

Combine_data function

Node 1: <start>
• This is the entry point of the function.
Node 2: if len(self.data_frames) >= 2:
• A condition is checked to see if at least two data frames are available to combine.
• True branch (if there are 2 or more data frames) leads to Node 3, while the False branch (if fewer than 2 data frames) leads to Node 9.
Node 3: command = CombineDataCommand(self, self.data_frames)
• Here, a new command object (CombineDataCommand) is created to combine the data frames.
Node 4: self.invoker.add_command(command)
• The created command object is added to the invoker, presumably to keep track of or execute commands in a controlled way.
Node 5: combined_data = command.execute()
• The command to combine the data is executed. This line attempts to merge the two data frames into combined_data.
Node 6: if combined_data is not None:
• After the execution, a check is performed to verify whether the combined data is valid (i.e., not None).
• If valid (True), it leads to Node 7, and if not (False), no operation is performed and the flow likely stops.
Node 7: self.combined_data = combined_data
• The merged/combined data is stored in the application's combined_dataattribute for further use.
Node 8: self.display_combined_names()
• The combined data is displayed using a function that updates the UI. After this, a logging message indicates that the operation was successful.
Node 9: messagebox.showwarning("Warning", "Please read two Excel files first.")
• This is the alternative (False) branch of Node 2, triggered when fewer than two data frames are available. A warning is shown, and a log entry is made.

True path: Node1 to Node2 to Node3 to Node4 to Node5 to Node6 to Node7 to Node8
Not enough data path: Node1 to Nod2 to Node9

CCC for Combine Names

8 – 7+(2)(1)= 3

Class Combine Data Command

Node 1 (Start)
Node 2 (Check for Data Frames)

- If True, go to Node 3 (Start Try Block)
- If False, go to Node 18 (Warning Message)

Node 3 (Start Try Block)
Node 4 (Extract Data Frames)
Node 5 (Normalize Names for database_data)
- **Sub-Node 5.1**: Normalize Mother_First_Name
- **Sub-Node 5.2**: Normalize Mother_Last_Name

Node 6 (Normalize Names for medicaid_data)
- **Sub-Node 6.1**: Normalize Mother_First_Name
- **Sub-Node 6.2**: Normalize Last_Name

Node 7 (Logging Normalization)
Edges
Node 1 to **Node 2**
Node 2 to **Node 3** (True path)
Node 2 to **Node 18** (False path)
Node 3 to **Node 4**
Node 4 to **Node 5**
Node 5 to **Sub-Node 5.1**
Sub-Node 5.1 to **Sub-Node 5.2**
Node 5 to **Node 6** (after completing normalization for database_data)
Node 6 to **Sub-Node 6.1**
Sub-Node 6.1 to **Sub-Node 6.2**
Node 6 to **Node 7**
Node 7 to **End of Try Block**
Node 18 to **End of Function // Node 18 is located in the else block of the if len(self.data_frames) >= 2**

Cyclomatic Complexity Calculation(CCC)

$$12-8+2 \cdot 1=12-8+2=6$$

Basis Path Combined_Names()

• **Path1:** Valid, two valid data frames are provided and the command combines the lists.
Node sequence:Node1 to Node2 to Node3 to Node4 to Node5 to Node6 to Node7 to Node8

• **Path2:** Invalid, No valid data frames are present.
NodeSequence:Node1 to Node2 to Node9

• **Path3:** Invalid, if fewer than 2 frames are present.
Node sequence:Node1 to Node2 to Node9

• **Path 4:** Invalid Exception occurs during normalization or merging:
• An exception (due to malformed data)causes the function to fail and return None.

• **Node sequence**: Node1 → Node2 → Execute combine command → Exception occurs → Log error → Return None.

• **Path 5: Valid case, no data after merge (due to no matching rows)**:
The merge occurs, but no matching rows are found (resulting in an empty data frame).
Node sequence: Node1 → Data frames available → Execute combine command → Node5.1 →Node 5.2->Node6.1->Node6.2-> Merge results in empty data → No data returned.

Test Cases for Combine_names

• **Test Case 1: Valid:**
  - Description: Test how the program responds to two valid data frames.
  - Input: The data frames array should have 2 data lists.
  - Expected Outcome: it calls the corresponding command to combine the two lists.

• **Test Case 2:invalid:**
  - Description: Test how the program responds to none valid data frames.
  - Input: The data frames array is empty.
  - Expected Outcome: it should skip down to node 9 and warning should be given **("Warning", "Please read two Excel files first.")**

• **Test Case 3:invalid**:
  - Description: Test how the program responds to less than one valid data frame.
  - Input: One data frame in the array data frame.
  - Expected Outcome: It should skip down to nod 9 and warning should be given **("Warning", "Please read two Excel files first.") provide an opportunity to fill the rest of the data frame.**

• Test Case 4: Invalid Data in Data Frames
  - **Description:** Attempt to combine data frames containing invalid or missing data.
  - **Input:** Data frames with missing or incorrectly formatted Mother_First_Name, Mother_Last_Name, or Child_Date_of_Birth.
  - **Expected Output:** An error message is displayed, and no combined data frame is created.

• Test Case 5: Successful Normalization and Merging
  - **Description:** Ensure that normalization correctly handles names with extra spaces and varying cases.
  - **Input:** Data frames with Mother_First_Name and Last_Name containing spaces, upper and lower cases.
  - **Expected Output:** The combined data frame shows normalized names (e.g., "Raf" instead of " raf ").

Group 4: Vadim Pidoshva, Logan Singleton, Rafael Almeida, Brennon Duffy, Lance Reed

Current Unit Test Demo:

```
vadimpidoshva@dhcp-10-5-78-171 CS4400 % pytest --rich -v -s --tb=long test.py

───────────────────────────── pytest session starts ─────────────────────────────
 platform darwin pytest 8.3.3 python 3.10.5
 root /Users/vadimpidoshva/Documents/School/CS4400

Collected 3 items

**************************************************
*** Test #1: Read Excel File Functionality ***
**************************************************

Step 1: 📄 Calling read_excel_file function...
Step 2: 🔍 Verifying the data frame was added to data_frames list...
✅ Data frame successfully added.

Step 3: 🔄 Verifying that ReadExcelCommand was called once...
✅ ReadExcelCommand was called once.


**************************************************
*** Test #2: Functionality of CombineDataCommand ***
**************************************************

Step 1: 📑 Creating CombineDataCommand with mock data...
Step 2: ⚙️Executing CombineDataCommand...
Step 3: 🔍 Verifying combined data is not None...
✅ Combined data is not None.

Step 4: 🔢 Checking the number of rows in the combined data...
✅ Combined data has 2 rows.

Step 5: 📝 Checking the contents of combined data columns...
✅ Verified column 'Mother_First_Name' exists in the combined data.

✅ Verified column 'Mother_Last_Name' exists in the combined data.

✅ Verified column 'Child_First_Name' exists in the combined data.

✅ Verified column 'Child_Last_Name' exists in the combined data.

✅ Verified column 'Child_Date_of_Birth' exists in the combined data.

📊 Final combined data output:
  Child_Last_Name Child_First_Name Child_Middle_Name Child_Date_of_Birth Mother_Last_Name Mother_First_Name  State_File_Number  Mother_DOB  ...      Mobile_#        Street        City State    ZIP County Tobacco_Usage Utah_First_Time_Man.
0             Doe            Alice             Marie          2021-05-10              Doe              Jane              12345  1980-05-10  ...  123-456-7891    123 Main St  Springfield    UT  84001   Utah         False                False
1           Smith              Bob             James          2020-08-21            Smith              John              67890  1978-12-22  ...  098-765-4322  456 Maple Ave     Mapleton    UT  84002   Utah          True                 True

[2 rows x 20 columns]

Step 7: ✅ Confirming that the test passed!

**************************************************
*** Test #3: Excel File Generation After Data Combination ***
**************************************************

Step 1: ⚙️Executing CombineDataCommand...
Step 2: 📂 Checking if the Excel file is generated...
✅ File 'combined_matched_data.xlsx' was successfully generated.

Step 3: ✅ Confirming file generation test passed!
✅ File 'combined_matched_data.xlsx' has been cleaned up after the test.

 [100%] test.py ✓✓✓
 Percent: 100%
vadimpidoshva@dhcp-10-5-78-171 CS4400 % █
```

README

Excel Data Combiner Application

This application merges data from two Excel files, typically hospital and Medicaid datasets, into one dataset for further inspection. It allows users to search, filter, view, and copy detailed profiles based on the merged data.

Features

- **Read Two Excel Files:** Users can select two Excel files (hospital and Medicaid datasets) and merge them based on the Mother's First Name, Last Name, and Child's Date of Birth.
- **Combine Data:** Combines the datasets into a single file for easy inspection.
- **Search & Filter:** Quickly search and filter combined data by name, ID, or Date of Birth.
- **Display Profiles:** Double-clicking on a name opens a detailed profile for the selected entry, showing information about the mother, child, and contact details.
- **Copy to Clipboard:** Users can copy profile information for easy documentation and sharing.
- **Excel Export:** Saves the combined dataset to an Excel file (combined_matched_data.xlsx).

Prerequisites

Ensure the following requirements are met to run the application:

- **Python 3.x** installed.
- Required Python Libraries:
  - `pandas`
  - `tkinter`
  - `openpyxl`
  - `logging`

You can install the required dependencies using pip:

```
pip install pandas openpyxl
```

Installation

1. Clone or download this repository.
2. Ensure the required Python packages are installed (see above).
3. Place your two Excel files (hospital and Medicaid datasets) in an accessible location.

Usage

1. Run the `app.py` script:

```
python app.py
```

2. The GUI will open with buttons to:

- **Read Excel File 1:** Load the first Excel dataset (e.g., hospital data).
- **Read Excel File 2:** Load the second Excel dataset (e.g., Medicaid data).
- **Combine Data:** Merge the two datasets based on "Mother's Name" and "Child's Date of Birth."

3. After merging, a new window will display the combined records, showing "Mother ID," "Child Name," and "Child DOB."

4. **View Child Profile:** Double-click on an entry to view a detailed profile of the selected entry, including information about the mother, child, and contact details.

5. **Copy Profile Info:** Copy profile details to the clipboard for easy sharing.

Application Workflow

- Read Excel Files: Click the "Read Excel File" buttons to load two Excel files (hospital and Medicaid datasets).
- Combine Data: After loading both files, click "Combine Data" to merge the files based on "Mother's First Name," "Mother's Last Name," and "Child's Date of Birth."
- Search & Filter: Use the search bar to filter the displayed names.
- View Profiles: Double-click an entry to view a detailed profile.
- Copy Profile Info: Copy profile details to the clipboard by clicking the "Copy Profile Info" button.

Application Layout

Main Window

- **Read Excel File 1:** Opens a file dialog for selecting the first Excel file.
- **Read Excel File 2:** Opens a file dialog for selecting the second Excel file.
- **Combine Data:** Merges the two datasets.

Combined Data Window

- **Search Bar:** Filter entries by Mother ID, Child Name, or Child DOB.
- **Results List:** Displays "Mother ID," "Child Name," and "Child DOB."
- **Double-click Feature:** Opens a detailed profile for the selected entry.

Profile View

Displays detailed information including:

- **Mother's Information:** Mother ID, First Name, Last Name.
- **Child's Information:** First Name, Last Name, Date of Birth.
- **Contact Information:** Street Address, City, State, ZIP, Phone, and Mobile Number.
- **Copy Profile Info:** Copies the profile details to the clipboard.

Logging

The application logs key events such as file reading, data combination, and errors. These logs are displayed in the console.

- **INFO:** Successful operations.
- **WARNING:** Operations that did not complete as expected (e.g., no file selected).
- **ERROR:** Issues encountered (e.g., data combination errors).

Excel File Output

The combined dataset is saved as `combined_matched_data.xlsx` in the current working directory after successfully combining the two datasets.

Unit Testing

The application includes a set of unit tests using the `unittest` or `pytest` module to ensure the functionality of critical features:

- **Test for Reading Excel Files:** Simulates reading an Excel file and verifies if the data is correctly loaded and appended.
- **Test for Data Combination:** Ensures that the two datasets are merged correctly based on "Mother's Name" and "Child's Date of Birth."
- **Test for Excel File Generation:** Verifies that the combined data is saved to an Excel file named `combined_matched_data.xlsx`.

You can run the tests using `pytest` with rich formatting for enhanced readability:

```
pytest --rich --tb=short -v test.py
```

Changes

- **New GUI Components:** Updated the GUI with a modern layout using ttk.Treeview for better data visualization.
- **New Copy Functionality:** Added functionality to copy profile information to the clipboard.
- **Unit Test Integration:** Introduced unit tests using unittest for testing key functions like reading Excel files, data combination, and Excel file generation.
- **Enhanced Logging:** Added detailed logging for tracking successful operations, warnings, and errors, visible in the console.
- **Improved Data Normalization:** Enhanced the data merging process by normalizing names and standardizing date formats.
- **Error Handling:** Implemented better error handling for file operations and merging errors.