COM00142M Advanced Programming

Summative Assessment Report

Word Count - 2974

Executive Summary

This report details the processing of a university CS course dataset—originally split into three files—through a custom GUI that meets the given functional requirements. The GUI provides a sample of the processed data for further analysis. The solution leverages Seaborn, Matplotlib, Pandas, and Numpy. This report explains each function, justifies the chosen methodologies, concludes the analysis, and explores the ethical, moral, and legal considerations in software engineering.

1A)

Data frames are 2-dimensional structures containing a table of rows and columns. For the purpose of data analytics, data frames are preferred due to their flexible and intuitive nature [1].

The first 3 steps of the file processing function (See Appendix B, Fig 2.1) - renaming, removing specific components from the data set and adding a unique identifier to the data frame - can be run simultaneously using Python Threads. Giancarlo defines threads as, “An active flow of control that can be activated in parallel with other threads within the same process with the term flow control meaning a sequential execution of machine instructions”[2]. The time complexity of the steps is O(n) where ‘n’ is the number columns in the renaming step, the number of rows in the addition of a unique identifier step as each of these algorithms requires reiteration over each of the rows and columns in the Data Frame. As data sets become large in this specific context, O(n) operation can become costly if the n is in the millions. The data set for the purpose of testing has been reduced to 400 records in the User Log and Activity log. However, the original number of records in each file is over 150,000 making it a heavy memory usage task for the CPU.

The user interface will often stall until the primary operation is finished because reading and processing huge files tends to stop the main thread. Simply running the task in a created thread while the remainder of the program runs normally can prevent a GUI from being blocked [3]. The primary UI thread is free to update the interface and react to user inputs [4] since the file reading and data processing operations are carried out on a background thread [5]. It is crucial to make sure that only the main thread calls the GUI and that no more than one thread updates the GUI at once [3].

Despite Python's Global Interpreter Lock restricting true parallelism for CPU bound operations in threads, I/O actions such as reading from files as less impacted [6]. Since the program heavily depends on file input, threading will allow processing of multiple files concurrently and will subsequently reduce the overall waiting time when dealing with large CSVs.

Any shared data or flow that is accessed or modified concurrently by many threads is susceptible to race conditions and inconsistent states. The first step is to determine which data structures—such as data frames read from CSV files or intermediate results produced during processing—will be accessed across threads. Synchronisation primitives like locks, events, or conditions could be needed to ensure thread safety [4]. Nonetheless, the design can be made simpler by reducing the amount of shared mutable state and utilising message-passing queues, which frequently eliminate the requirement for explicit locks [8].

In this case, threading would be implemented by creating a new thread that executes process files that include the three stages in the background when the user clicks

‘Process’, rather than executing the process_files(window) function directly in the main thread [4]. A thread-safe queue or Tkinter's after() method can be used to update the user interface from the main thread because GUI frameworks like Tkinter are not thread-safe [5, 7, 8]. The event loop of the main thread can periodically check the queue that the worker thread has created and update labels, progress bars, or messages with status messages or intermediate results.

1B)

Several GUIs have been developed to clearly separate functions, however, 2 of the GUIs support the 3 main user interactions - Upload & Processing User Interface (See Appendix B, Fig 2.4), Table Manager (See Appendix B, Fig 2.5) - and other ‘View’ GUIs (for example, See Appendix B, Fig 2.6) that can be accessed via the Table Manager that do not require user input.

The layouts have been designed keeping in mind some of Nielsen's general principles for interaction design [9]. User control and freedom is maintained by easy navigation in each window that leads back to the homepage within 2 or less clicks. The consistency across the user interfaces in terms of simplicity of terminology is maintained so as to make the application as self explanatory as possible. The Browse to Upload buttons have been created on one side with the File name on the other side and the file input validation in the middle (See Appendix B, Fig 2.4) . The colour schemes and confirmations of user interactions and validations across the program has also been incorporated (see Appendix B, Fig 2.2). This avoids user uncertainty and offers clarity. Users are given visible clues when a file is uploaded correctly because of the usage of colour-coded feedback (green for acceptable files and red for faulty files) in the Upload and Process files GUI as one of the most crucial aspects of GUI design is colour scheming. It's also important to remember consumers with vision impairments when choosing colour schemes [12]. As a result, the colours contrast, to a colour blind individual, intensifying the shade of text would signify whether the file passes or fails validation enhancing further accessibly of the application.

According to Nielsen's Usability Heuristics [9], this approach is in accordance with system status visibility. The status of file uploads is instantly visible to users. Each title, display field, and file upload button are clearly separated by the interface's grid layout. Every file type (USER_LOG, ACTIVITY_LOG, and COMPONENT_CODES) has its own button and label. To avoid having the user jog their memory for files translated or modified, Table Manager GUI has been created to provide a quick overview. The files stored have been automatically named using the current date and time to help user navigate through files. This automatic functions helps with error prevention and reduces the odds of slips and mistakes and also assists with reducing users' interaction with their own memory. This design is minimalistic and provides efficiently for it's intended purpose[9].

The table manager then allows the user to further interact with the processed files by allowing them to view the merged table, view the pivot table separately, view statistics and view graphs of selected table. The GUIs for these other user interactions are simple and compact designs with an option for the user to navigate to the home page. Gestalt's Principles [10] have also been used to shape the GUI, in terms of the similarity of buttons grouped together based on functionality and continuity of flow through the application's GUI. The view tables have been grouped together while the statistics and graphs have been grouped together.

While there is no documentation for the GUI application, guidance has been displayed on Upload screen to help the user achieve the task at hand [9].

By utilising components like Grid layouts, Listbox, Text widgets, and Treeview tables, the application design implementation follows best practices and guarantees usability, clarity, and responsiveness. The addition of tooltips, more visual indications such as color coded buttons, and a visual status bar for when processing larger files can improve the design further.
A poorly designed GUI might include exclude input validation or no clear separation of file types with labels, buttons that are in no specific order, no colour coding for confirmation of user interaction or guidance. A poorly designed GUI would lead to the user concentrating on the GUI instead, as mentioned by Bernard Jansen [11].

2A)

As in the Fig 2.1 (Appendix B), Step 1 renames the columns as shorter, more concise columns names help reduce ambiguity [13]. A unique identifier is then added to both, the user log and the activity log since the User_ID has recurring values. Step 3 then merges the user log and activity log, removes duplicate columns. Step 4 removes the 'System' and 'Folder' components and any nulls values that the files might have. The
date and time has also been reformatted to suit readability and usability for further analysis. The remove function is carried out after the table has been merged as the user log

does not contain any component details and both the files, upon observation with a V-Lookup and match test, have a confirmed 1:1 relationship. Removing the components before the table is merged will result in records deleted from one file and not from the other. The pivot table version of the merged and filtered table is then viewed via the Table Manager GUI, using the view_pivot_table() function designed (see Appendix B, Fig 2.3). This function has been separated to give the user reasonable options for analysis and basic understanding of the data. This reshaped table includes the count of user interactions for each user with the component for each month.

JSON files aid in the flexibility and ease of user in data. In the context of the prototypes functionality, JSON files contain values that may be objects or arrays that could be arbitrarily nested deep as in the pivot table function, making reshaping into hierarchical data, easier [14]. Because JSON is a text-based, language-independent, and lightweight data interchange format, it allows for more versatile data manipulation in terms of programming languages, file encoding speed and code execution [15]. This format also provides metadata that helps understand the raw file better when creating pipelines, structures and models for data analysis. The original files being CSV, are less suited to handle unstructured, complex data, which JSON accounts for the possibility of, making the data translation, future proof [16].

Despite there being several benefits of this format, in the context of this application, the dynamic data typing for keys can lead to extra lines of code in the pipeline, eg:- string representation of dates can affect how the machine reads the reads and returns an error [16]. “Less is more” [17], though an architectural approach, is also applicable to programming as well. More lines of code implies more to maintain and develop. JSON files, due to dynamic typing of values, inhibits this to some extent.

2B)

No additional cleaning has been undertaken at this stage as the cleaning of the CSVs has been done before the files are visible in the Table Manager via process_files() function (See Appendix B, Fig 2.1). The null values have been removed and a unique identifier has been added for the final merged table which contains all necessary info for further analysis and breakdown.

For the purpose of testing the code, only the first 400 rows of the USER_LOG and ACTIVITY_LOG were used for processing and further analysis. Hence, in the visuals (See Appendix B, Fig 2.8 and Fig 2.9), some of the components and users are not mentioned while the code accounts for it (See Appendix B, Fig 2.7.1 and Fig 2.7.2).

Because User_ID is categorical rather than numerical, it is inherently challenging to calculate the relationship between the two. Components also
categorically represents different kinds of interactions.

Typically, correlation quantifies numerical relationships. Direct correlation computation is not practical for categorical data such as User_ID and Component. Instead, a more logical approach would be to examine how users and components are associated and distributed [18]. Agresti recommends using the Chi-Square Test or metrics like Cramér's V to investigate relationships between categorical variables [18]. Since this API is not covered in this module, a contingency table (cross-tabulation) has been made using pandas (See Appendix B, Fig 2.7.1) to provide an overview of the distribution patterns of interactions among users and components as Pandas is perfect for exploratory and preliminary data analysis because it offers strong structures like DataFrames for effective tabular data manipulation, aggregation, and transformation [24]. Based on this, contingency table, a heatmap was generated using seaborn (See Appendix B, Fig 2.7.1) to assess each User's interaction with specific components seeing as seaborn is a great option for producing informative and aesthetically pleasing statistical visualisations because of its connection with pandas and its intuitive API [25].

The current sample data is only 400 records, however, to allow for analysis of big datasets, heatmaps was chosen as it provides a concise and understandable visualisation, and the colour encoding enables users to spot trends and patterns fast [19]. It was also selected to enable a visual understanding of a potential association between User ID and components because colour-based heatmap visualisations are easy to use and accessible to a variety of audiences, allowing for a quicker understanding of data trends [20].
Since bar charts are the best way to convey distinct comparisons across categories because they make differences plain to see, a bar chart was used to visualise the data from user interaction with the chosen components using a count of all interactions per component [21]. Stakeholders can identify high frequency components (Assignment) and components with lower engagement that would need more research by concentrating on certain components. Using bar graphs to visualise total counts highlights important information like activity dispersion and aids in identifying areas that need attention [22]. Bar chart annotations give another level of detail, increasing data accuracy and making it easier for users to quickly extract precise values [23]. To achieve this Matplotlib was used as it continues to be the most flexible Python toolkit for intricate and adaptable data visualisations, allowing for exact graphical control and annotation [26]. Fig 2.7.1 and Fig 2.7.2 is the Python implementation of this.

It can be seen that the component with the highest usage, Assignment, indicates that it is used frequently, while the components with the lowest user interactions, Book and Project, reflect the least amount of engagement.

In conclusion, elements that are often used, such as assignments and quizzes, may be essential to users' activities. Prioritising these elements could help them be improved even further.
Investigating components with low utilisation may be necessary. Provided that more information regarding the components is acquired, we can then think about users' preferences or ignorance and how it causes certain components to be underutilised and whether interaction with these components limited by technical or usability issues?

3)

Morality serves as a system of guidelines that influences and regulates behaviour, emphasizing proper conduct [27]. Morality in the context of job roles has been a debate for a long time. Software engineers, given their pivotal role in product development, are responsible for ensuring their creations meet the highest professional standards and reflect integrity and independence in their judgments. The ACM Code of Ethics further reinforces this commitment by prioritizing the public interest alongside employer and client considerations [28].

The reliance of nearly one billion people on software systems for their daily activities underscores the growing need for ethical awareness within the technology and data industries [29]. Beyond adherence to legal frameworks, the primary focus must remain on determining what is right and wrong from the consumer's perspective. Decisions made in computing, particularly those affecting individuals' lives, highlight the intersection of ethics and technology, connecting technical choices to broader human values [30].

Software engineers operate in a client-facing capacity, as their products are designed for organizational or public use. This dynamic creates a prerequisite for an ethical approach at every stage of development, including product features, security protocols, and data usage. Engineers must ensure their decisions are grounded in ethical and legal principles to avoid adverse consequences, such as data breaches, algorithmic bias, or privacy violations [31]. Data breaches can lead to identity theft, financial loss, and psychological distress, while biased algorithms may perpetuate discrimination, resulting in inequitable hiring, lending, and judicial outcomes. Such issues not only harm those directly affected but also erode trust in software systems, undermining user engagement and product success [31].

It is not an uncommon practice for large organizations to direct software engineers to write code that is in breach of certain computing laws, it creates a requirement to flag such potential breaches so that it can be looked into by the appropriate professionals to avoid legal consequences. Many organizations employ specific whistleblowing teams to handle such cases. In other instances, this may be unintentional and should be raises with the Lead engineer. Regardless of intention in direction, software engineers should be mindful of the products they work on, to avoid being legally implicated.

Programmers should also critically assess the moral, ethical, and legal implications of using user-submitted social media content for training machine learning models. While leveraging publicly available data might yield positive outcomes, such as advancements in health predictions, using data without the explicit consent of owners raises significant ethical concerns [32]. Social media platforms could use more robust moderation practices to address these challenges effectively simply because the demographic of users has widened and the scope of negative effects is more. Online platforms are now available to most income levels and those of potentially dire circumstances. Any collection of personal information carries risks. Even in cases when data collection companies follow best procedures, hacks and breaches of vulnerable individuals' personal information are a serious problem [33]. The individuals may lack the knowledge and/or resources to protect themselves from such exploitation, making it the duty of software engineers to consider how the data is being collected and it’s intended use.

The ethical and moral challenges in software engineering ultimately depend on individual discretion and the situational context. Human behavior is influenced by environmental factors, often shaped by societal norms and personal experiences. Although morality exhibits universal themes, cultural differences give rise to diverse interpretations and applications. As moral intuitions evolve alongside cultural practices and institutions, the nuances of ethics in software engineering become increasingly subjective [34].

Appendix A - References

[1] “What are DataFrames?,” *Databricks*. https://www.databricks.com/glossary/what-are-dataframes#:~:text=A%20DataFrame%20is%20a%20data,storing%20and%20working%20with%20data.

[2] “Python Parallel Programming Cookbook,” *Google Books*. https://books.google.de/books?id=Aht1CgAAQBAJ&lpg=PP1&ots=VwqJ-cXuKi&dq=threading%20python&lr&pg=PA26#v=onepage&q&f=false

[3] M. Lutz, *Learning Python: Powerful Object-Oriented Programming*. “O’Reilly Media, Inc.,” 2013.

[4] “threading — Thread-based parallelism,” *Python Documentation*. https://docs.python.org/3/library/threading.html

[5]“How do you run your own code alongside Tkinter’s event loop?,” *Stack Overflow*. https://stackoverflow.com/questions/459083/how-do-you-run-your-own-code-alongside-tkinters-event-loop

[6] “GlobalInterpreterLock - Python Wiki.” https://wiki.python.org/moin/GlobalInterpreterLock

[7] “tkinter — Python interface to Tcl/Tk,” *Python Documentation*. https://docs.python.org/3/library/tkinter.html

[8] “queue — A synchronized queue class,” *Python Documentation*.

[9] J. Nielsen, “10 Usability heuristics for user interface design,” Nielsen Norman Group, Feb. 20, 2024. https://www.nngroup.com/articles/ten-usability-heuristics/

[10] C. C. Pratt and K. Koffka, “Principles of Gestalt Psychology,” The American Journal of Psychology, vol. 48, no. 3, p. 527, Jul. 1936, doi: 10.2307/1415906.

[11] B. J. Jansen, “The graphical user interface,” ACM SIGCHI Bulletin, vol. 30, no. 2, pp. 22–26, Apr. 1998, doi: 10.1145/279044.279051.

[12] M. Nosrati, R. Karimi, R. Karimi, H. Nosrati, “Investigating the basic principles for proper GUI design, Journal of American Science, 2011.

[13] J. Zhang, Z. Shen, B. Srinivasan, S. Wang, H. Rangwala, and G. Karypis, “NameGuess: Column Name Expansion for Tabular Data,” *arXiv.org*, Oct. 19, 2023. https://arxiv.org/abs/2310.13196

[14] C. Chasseur, University Of Wisconsin, Y. Li, University Of Wisconsin, J. M. Patel, and University Of Wisconsin, “Enabling JSON Document Stores in Relational Systems,” WebDB, 2013, [Online]. Available: http://pages.cs.wisc.edu/~chasseur/pubs/argo-long.pdf

[15] I Garg, NR Wudaru, “ Study on JSON, it’s Uses and Applications in Engineering Organizations”, Researchgate, 2024, [Online], Available: https://www.researchgate.net/profile/Ishita-Garg-11/publication/379001324_Study_on_JSON_its_Uses_and_Applications_in_Engineering_Organizations/links/65f56ac032321b2cff8353ae/Study-on-JSON-its-Uses-and-Applications-in-Engineering-Organizations.pdf

[16] A. Jaiswal, “JSON: Introduction, Benefits, Applications, and Drawbacks,” Turing, Jun. 24, 2022. https://www.turing.com/kb/what-is-json

[17] P. Johnson and H.-R. Hitchcock, *The International Style: Architecture Since 1922*. W. W. Norton & Company, 1966.

[18] A. Agresti, *Categorical Data Analysis*. 3rd ed. Hoboken, NJ: Wiley, 2013

[19] A. J. Han and M. Kamber, Data Mining: Concepts and Techniques. San Francisco, CA: Morgan Kaufmann, 2011.

[20] H. Wickham, ggplot2: Elegant Graphics for Data Analysis. New York: Springer, 2016.

[21] M. Friendly and E. Kwan, "Effectiveness of graphical representation for categorical data," Journal of Computational and Graphical Statistics, vol. 12, no. 3, pp. 123-145, 2003.

[22] S. Few, Show Me the Numbers: Designing Tables and Graphs to Enlighten. Oakland, CA: Analytics Press, 2012.

[23] E. R. Tufte, The Visual Display of Quantitative Information. Cheshire, CT: Graphics Press, 2001.

[24] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. Sebastopol, CA: O'Reilly Media, 2017.

[25] M. Waskom, "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021

[26] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science &* Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[27] J. M. Kizza, Ethics in Computing, 4th ed. Springer, 2016. doi: 10.1007/978-3-319-29106-2.

[28] ACM Ethics, "Software Engineering Code," ACM Ethics Committee, Jun. 08, 2022. Available: https://ethics.acm.org/code-of-ethics/software-engineering-code/

[29] K. Reed, "Software Engineering—A New Millennium?" IEEE Software, vol. 17, no. 4, p. 107, Jul. 2000, doi: 10.1109/MS.2000.877656.

[30] D. Gotterbarn, "Software Engineering Ethics," Encyclopedia of Software Engineering, Jan. 2002, doi: 10.1002/0471028959.sof314.

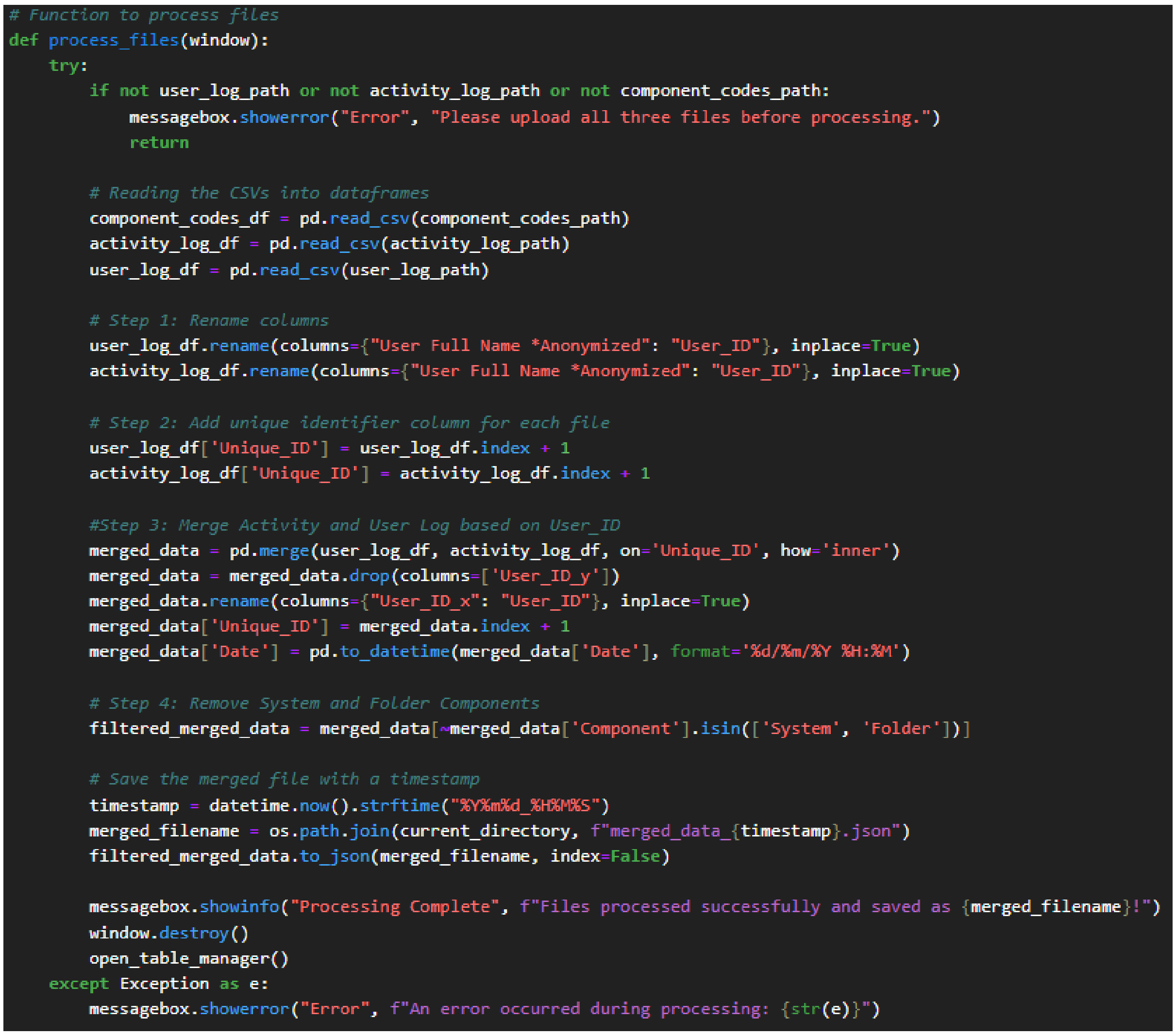
[31] Institute of Data, "The Essential Software Engineering Code of Ethics," Institute of Data Blog, Nov. 12, 2023. Available: https://www.institutedata.com/blog/software-engineering-code-of-ethics/

[32] A. Jaiswal, A. Shah, C. Harjadi, E. Windgassen, and P. Washington, “Ethics of the Use of Social Media as Training Data for Artificial Intelligence Models used for Digital Phenotyping: Commentary (Preprint),” JMIR Formative Research, vol. 8, p. e59794, Jun. 2024, doi: 10.2196/59794.

[33] A. Lovelace Institute, "The data of the most vulnerable people is the least protected," Ada Lovelace Institute. [Online]. Available: https://www.adalovelaceinstitute.org/blog/data-most-vulnerable-people-least-protected/.

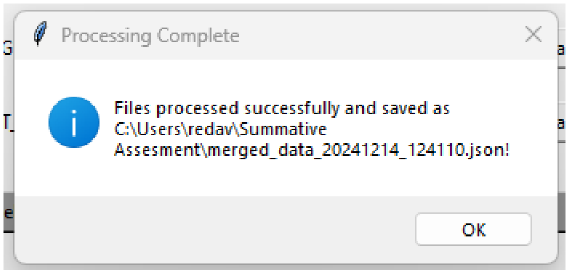
[34] J. Haidt, “The New Synthesis in Moral Psychology,” Science, vol. 316, no. 5827, pp. 998–1002, May 2007, doi: 10.1126/science.1137651.

Appendix B – Code Implementations and GUIs



```python
# Function to process files
def process_files(window):
    try:
        if not user_log_path or not activity_log_path or not component_codes_path:
            messagebox.showerror("Error", "Please upload all three files before processing.")
            return

        # Reading the CSVs into dataframes
        component_codes_df = pd.read_csv(component_codes_path)
        activity_log_df = pd.read_csv(activity_log_path)
        user_log_df = pd.read_csv(user_log_path)

        # Step 1: Rename columns
        user_log_df.rename(columns={"User Full Name *Anonymized": "User_ID"}, inplace=True)
        activity_log_df.rename(columns={"User Full Name *Anonymized": "User_ID"}, inplace=True)

        # Step 2: Add unique identifier column for each file
        user_log_df['Unique_ID'] = user_log_df.index + 1
        activity_log_df['Unique_ID'] = activity_log_df.index + 1

        #Step 3: Merge Activity and User Log based on User_ID
        merged_data = pd.merge(user_log_df, activity_log_df, on='Unique_ID', how='inner')
        merged_data = merged_data.drop(columns=['User_ID_y'])
        merged_data.rename(columns={"User_ID_x": "User_ID"}, inplace=True)
        merged_data['Unique_ID'] = merged_data.index + 1
        merged_data['Date'] = pd.to_datetime(merged_data['Date'], format='%d/%m/%Y %H:%M')

        # Step 4: Remove System and Folder Components
        filtered_merged_data = merged_data[~merged_data['Component'].isin(['System', 'Folder'])]

        # Save the merged file with a timestamp
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        merged_filename = os.path.join(current_directory, f"merged_data_{timestamp}.json")
        filtered_merged_data.to_json(merged_filename, index=False)

        messagebox.showinfo("Processing Complete", f"Files processed successfully and saved as {merged_filename}!")
        window.destroy()
        open_table_manager()
    except Exception as e:
        messagebox.showerror("Error", f"An error occurred during processing: {str(e)}")
```

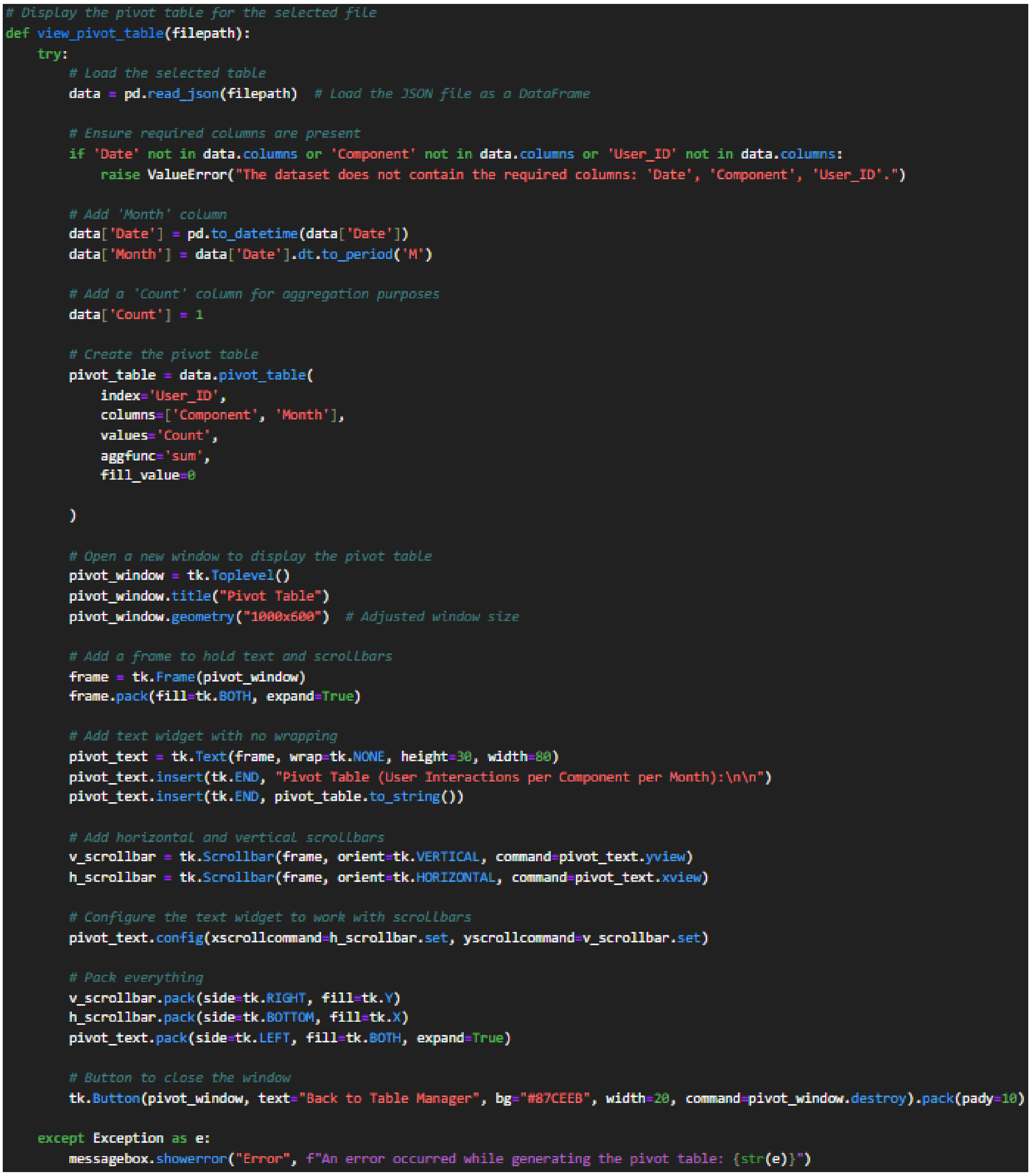
Fig 2.1 Python function to process the files.



Processing Complete

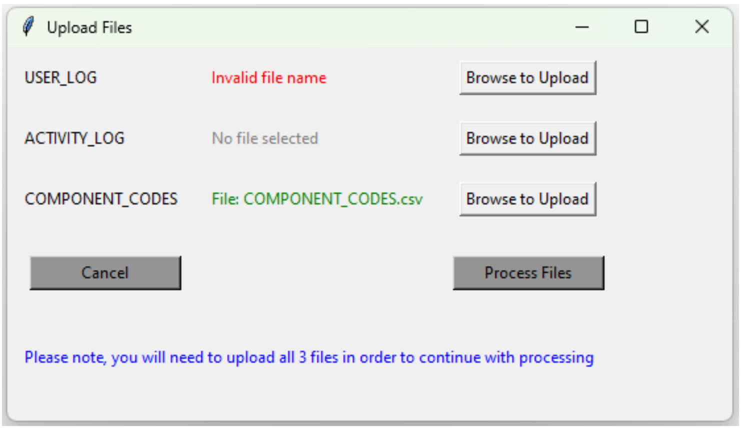
Files processed successfully and saved as C:\Users\redav\Summative Assesment\merged_data_20241214_124110.json!

Fig 2.2 Completion of processing confirmation



```python
# Display the pivot table for the selected file
def view_pivot_table(filepath):
    try:
        # Load the selected table
        data = pd.read_json(filepath)  # Load the JSON file as a DataFrame

        # Ensure required columns are present
        if 'Date' not in data.columns or 'Component' not in data.columns or 'User_ID' not in data.columns:
            raise ValueError("The dataset does not contain the required columns: 'Date', 'Component', 'User_ID'.")

        # Add 'Month' column
        data['Date'] = pd.to_datetime(data['Date'])
        data['Month'] = data['Date'].dt.to_period('M')

        # Add a 'Count' column for aggregation purposes
        data['Count'] = 1

        # Create the pivot table
        pivot_table = data.pivot_table(
            index='User_ID',
            columns=['Component', 'Month'],
            values='Count',
            aggfunc='sum',
            fill_value=0

        )

        # Open a new window to display the pivot table
        pivot_window = tk.Toplevel()
        pivot_window.title("Pivot Table")
        pivot_window.geometry("1000x600")  # Adjusted window size

        # Add a frame to hold text and scrollbars
        frame = tk.Frame(pivot_window)
        frame.pack(fill=tk.BOTH, expand=True)

        # Add text widget with no wrapping
        pivot_text = tk.Text(frame, wrap=tk.NONE, height=30, width=80)
        pivot_text.insert(tk.END, "Pivot Table (User Interactions per Component per Month):\n\n")
        pivot_text.insert(tk.END, pivot_table.to_string())

        # Add horizontal and vertical scrollbars
        v_scrollbar = tk.Scrollbar(frame, orient=tk.VERTICAL, command=pivot_text.yview)
        h_scrollbar = tk.Scrollbar(frame, orient=tk.HORIZONTAL, command=pivot_text.xview)

        # Configure the text widget to work with scrollbars
        pivot_text.config(xscrollcommand=h_scrollbar.set, yscrollcommand=v_scrollbar.set)

        # Pack everything
        v_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        h_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)
        pivot_text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Button to close the window
        tk.Button(pivot_window, text="Back to Table Manager", bg="#87CEEB", width=20, command=pivot_window.destroy).pack(pady=10)

    except Exception as e:
        messagebox.showerror("Error", f"An error occurred while generating the pivot table: {str(e)}")
```

Fig 2.3 View Pivot table version of merged file function



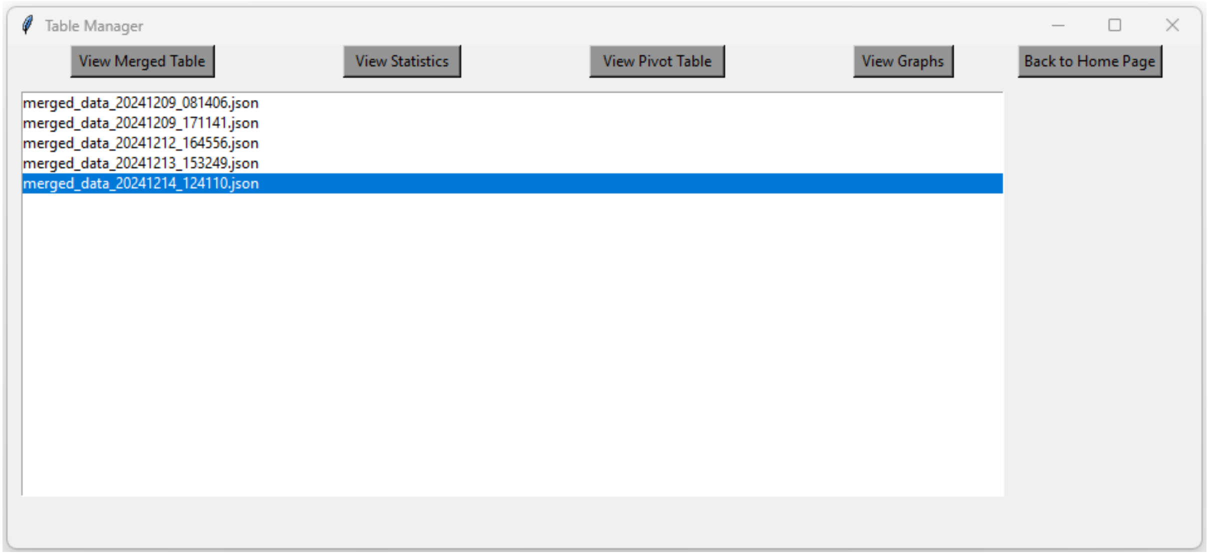
USER_LOG Invalid file name

ACTIVITY_LOG No file selected

COMPONENT_CODES File: COMPONENT_CODES.csv

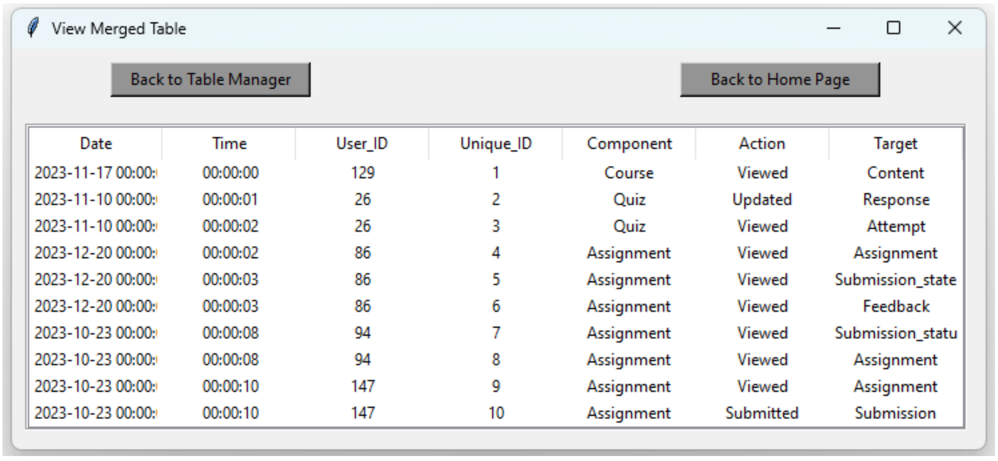
Please note, you will need to upload all 3 files in order to continue with processing

Fig 2.4 Uploading and Processing Files GUI



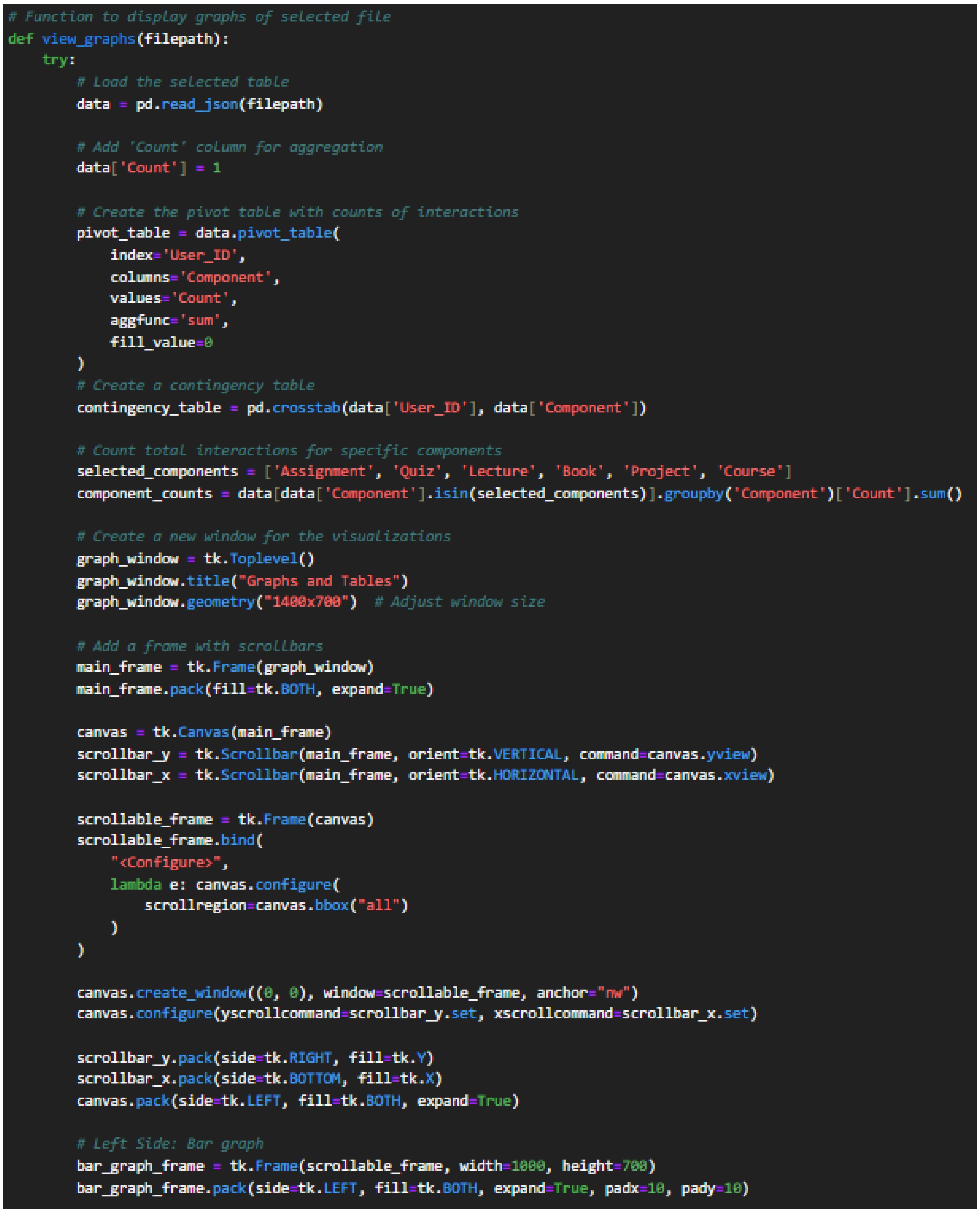
merged_data_20241209_081406.json
merged_data_20241209_171141.json
merged_data_20241212_164556.json
merged_data_20241213_153249.json
merged_data_20241214_124110.json

Fig 2.5 Table Manager



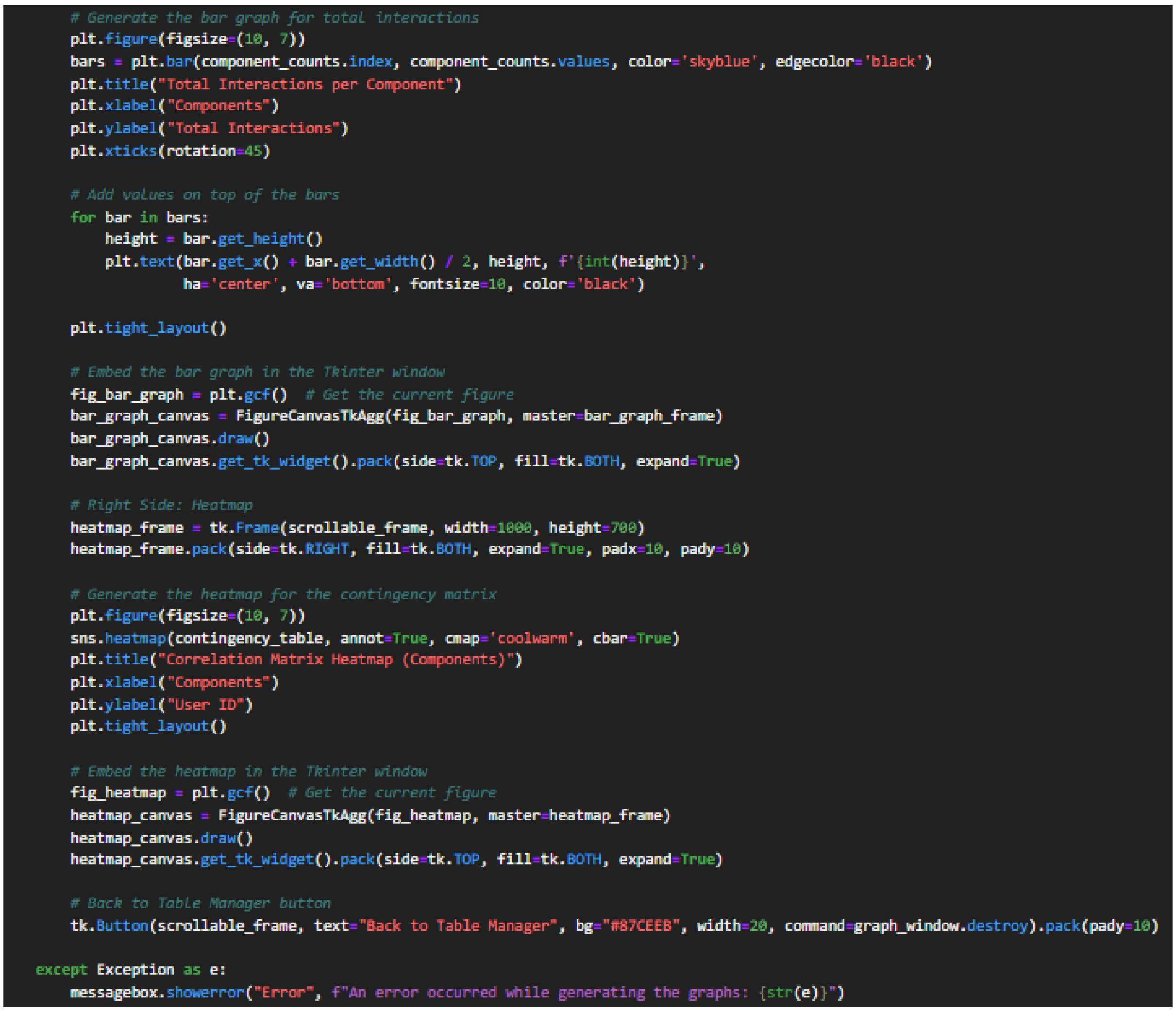
Date	Time	User_ID	Unique_ID	Component	Action	Target
2023-11-17 00:00:	00:00:00	129	1	Course	Viewed	Content
2023-11-10 00:00:	00:00:01	26	2	Quiz	Updated	Response
2023-11-10 00:00:	00:00:02	26	3	Quiz	Viewed	Attempt
2023-12-20 00:00:	00:00:02	86	4	Assignment	Viewed	Assignment
2023-12-20 00:00:	00:00:03	86	5	Assignment	Viewed	Submission_state
2023-12-20 00:00:	00:00:03	86	6	Assignment	Viewed	Feedback
2023-10-23 00:00:	00:00:08	94	7	Assignment	Viewed	Submission_statu
2023-10-23 00:00:	00:00:08	94	8	Assignment	Viewed	Assignment
2023-10-23 00:00:	00:00:10	147	9	Assignment	Viewed	Assignment
2023-10-23 00:00:	00:00:10	147	10	Assignment	Submitted	Submission

Fig 2.6 View Merged Table



```python
# Function to display graphs of selected file
def view_graphs(filepath):
    try:
        # Load the selected table
        data = pd.read_json(filepath)

        # Add 'Count' column for aggregation
        data['Count'] = 1

        # Create the pivot table with counts of interactions
        pivot_table = data.pivot_table(
            index='User_ID',
            columns='Component',
            values='Count',
            aggfunc='sum',
            fill_value=0
        )
        # Create a contingency table
        contingency_table = pd.crosstab(data['User_ID'], data['Component'])

        # Count total interactions for specific components
        selected_components = ['Assignment', 'Quiz', 'Lecture', 'Book', 'Project', 'Course']
        component_counts = data[data['Component'].isin(selected_components)].groupby('Component')['Count'].sum()

        # Create a new window for the visualizations
        graph_window = tk.Toplevel()
        graph_window.title("Graphs and Tables")
        graph_window.geometry("1400x700")  # Adjust window size

        # Add a frame with scrollbars
        main_frame = tk.Frame(graph_window)
        main_frame.pack(fill=tk.BOTH, expand=True)

        canvas = tk.Canvas(main_frame)
        scrollbar_y = tk.Scrollbar(main_frame, orient=tk.VERTICAL, command=canvas.yview)
        scrollbar_x = tk.Scrollbar(main_frame, orient=tk.HORIZONTAL, command=canvas.xview)

        scrollable_frame = tk.Frame(canvas)
        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(
                scrollregion=canvas.bbox("all")
            )
        )

        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar_y.set, xscrollcommand=scrollbar_x.set)

        scrollbar_y.pack(side=tk.RIGHT, fill=tk.Y)
        scrollbar_x.pack(side=tk.BOTTOM, fill=tk.X)
        canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Left Side: Bar graph
        bar_graph_frame = tk.Frame(scrollable_frame, width=1000, height=700)
        bar_graph_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=10)
```

Fig 2.7.1 View Graphs Function



```
        # Generate the bar graph for total interactions
        plt.figure(figsize=(10, 7))
        bars = plt.bar(component_counts.index, component_counts.values, color='skyblue', edgecolor='black')
        plt.title("Total Interactions per Component")
        plt.xlabel("Components")
        plt.ylabel("Total Interactions")
        plt.xticks(rotation=45)

        # Add values on top of the bars
        for bar in bars:
            height = bar.get_height()
            plt.text(bar.get_x() + bar.get_width() / 2, height, f'{int(height)}',
                     ha='center', va='bottom', fontsize=10, color='black')

        plt.tight_layout()

        # Embed the bar graph in the Tkinter window
        fig_bar_graph = plt.gcf()  # Get the current figure
        bar_graph_canvas = FigureCanvasTkAgg(fig_bar_graph, master=bar_graph_frame)
        bar_graph_canvas.draw()
        bar_graph_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        # Right Side: Heatmap
        heatmap_frame = tk.Frame(scrollable_frame, width=1000, height=700)
        heatmap_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Generate the heatmap for the contingency matrix
        plt.figure(figsize=(10, 7))
        sns.heatmap(contingency_table, annot=True, cmap='coolwarm', cbar=True)
        plt.title("Correlation Matrix Heatmap (Components)")
        plt.xlabel("Components")
        plt.ylabel("User ID")
        plt.tight_layout()

        # Embed the heatmap in the Tkinter window
        fig_heatmap = plt.gcf()  # Get the current figure
        heatmap_canvas = FigureCanvasTkAgg(fig_heatmap, master=heatmap_frame)
        heatmap_canvas.draw()
        heatmap_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        # Back to Table Manager button
        tk.Button(scrollable_frame, text="Back to Table Manager", bg="#87CEEB", width=20, command=graph_window.destroy).pack(pady=10)

    except Exception as e:
        messagebox.showerror("Error", f"An error occurred while generating the graphs: {str(e)}")
```

Fig 2.7.2 View Graphs function continuation

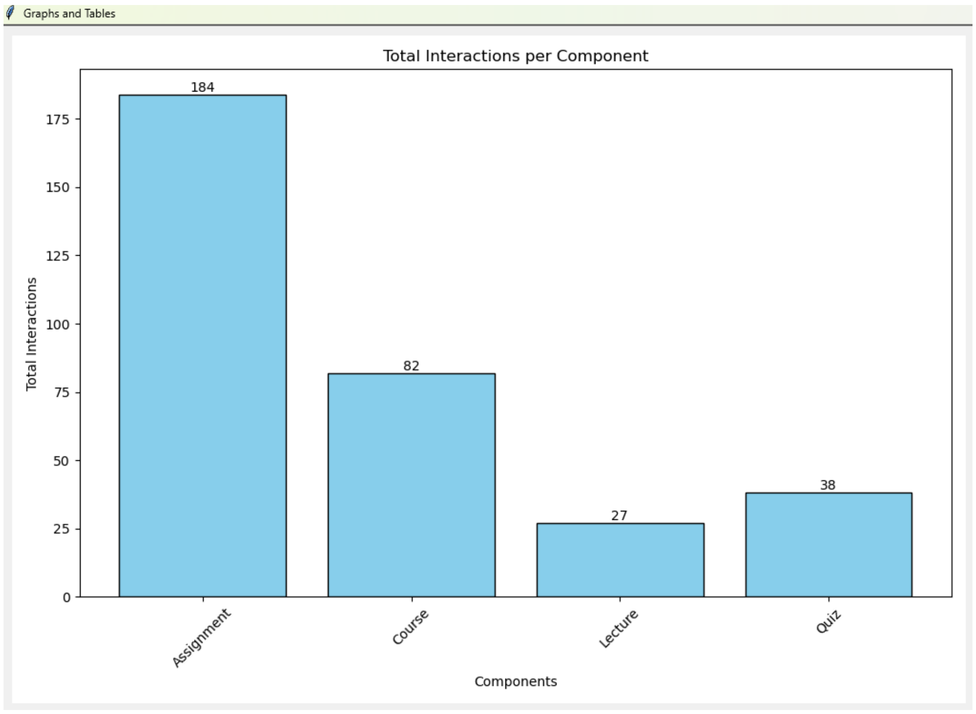
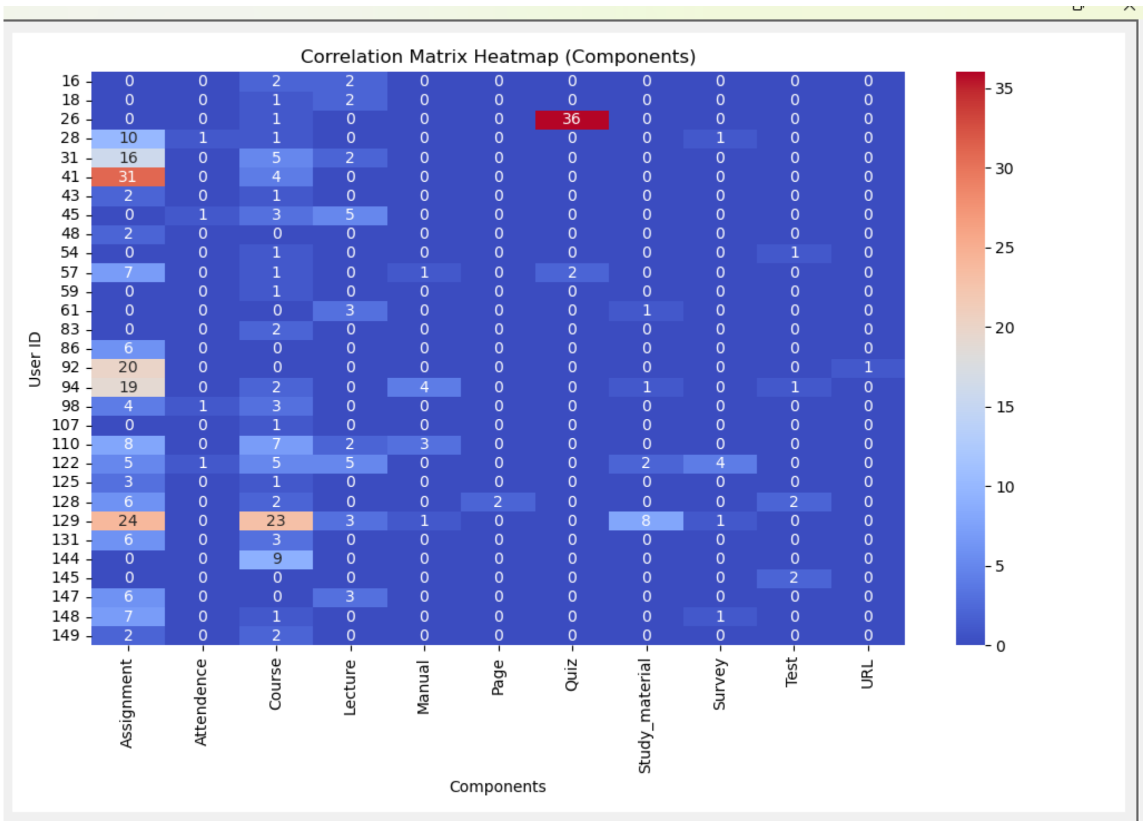


Fig 2.8 Total Interactions per component



User ID	Assignment	Attendence	Course	Lecture	Manual	Page	Quiz	Study_material	Survey	Test	URL
16	0	0	2	2	0	0	0	0	0	0	0
18	0	0	1	2	0	0	0	0	0	0	0
26	0	0	1	0	0	0	36	0	0	0	0
28	10	1	1	0	0	0	0	0	1	0	0
31	16	0	5	2	0	0	0	0	0	0	0
41	31	0	4	0	0	0	0	0	0	0	0
43	2	0	1	0	0	0	0	0	0	0	0
45	0	1	3	5	0	0	0	0	0	0	0
48	2	0	0	0	0	0	0	0	0	0	0
54	0	0	1	0	0	0	0	0	0	1	0
57	7	0	1	0	1	0	2	0	0	0	0
59	0	0	1	0	0	0	0	0	0	0	0
61	0	0	0	3	0	0	0	1	0	0	0
83	0	0	2	0	0	0	0	0	0	0	0
86	6	0	0	0	0	0	0	0	0	0	0
92	20	0	0	0	0	0	0	0	0	0	1
94	19	0	2	0	4	0	0	1	0	1	0
98	4	1	3	0	0	0	0	0	0	0	0
107	0	0	1	0	0	0	0	0	0	0	0
110	8	0	7	2	3	0	0	0	0	0	0
122	5	1	5	5	0	0	0	2	4	0	0
125	3	0	1	0	0	0	0	0	0	0	0
128	6	0	2	0	0	2	0	0	0	2	0
129	24	0	23	3	1	0	0	8	1	0	0
131	6	0	3	0	0	0	0	0	0	0	0
144	0	0	9	0	0	0	0	0	0	0	0
145	0	0	0	0	0	0	0	0	0	2	0
147	6	0	0	3	0	0	0	0	0	0	0
148	7	0	1	0	0	0	0	0	1	0	0
149	2	0	2	0	0	0	0	0	0	0	0

Fig 2.9 User interactions with each component