UNIVERSITY of York

Assignment Date: December 23rd, 2024

Summative Assignment

Module: Advanced Programming

1a) [20 marks] Identify ONE part of your program design (such as processing the initial data set) that has the potential to be redesigned concurrently, using Python Threads. Clearly identify the program part and justify its selection and potential. Then discuss any specific issues that would need to be considered to refactor this part, and the wider impact of this refactoring on your whole program design. You should consider how data and/or communications will be passed between concurrent aspects, such as threads, and justify which Python constructs would support this redevelopment effectively.

Of the many available options for program redesigning, concurrent redesigns provide aspects of efficiency, improvement and scalability for current program designs. Concurrent program is achieved through its simultaneous and seamless utilization of multiple tasks. These redesigns can be achieved using Python Threads, which allows for multithreading, an aspect of multitasking, through the creation of multiple threads, or lightweight units of execution within the same process, which is ideal for reading and writing files or retrieving data [1][2] .

In the case of this application the process_all_given_data function, which processes the initial raw datasets for analysis, can be redesigned concurrently. This function can be redesigned by breaking the function down into smaller task in multiple phases that can be run simultaneously without interdependent reliance for functionality. The process_all_given_data function, code shown in Figure 1, consist of merging the converted and reloaded JSON files preparing them for analysis. Subsequent data transformation steps followed by renaming columns, parsing dates and removing and filtered out irrelevant components, which allows for extracting and creating pivot tables to summarize the user interaction counts by user, component and timepoint.

The proposed concurrent redesigned would also include thread assignments for the distinct tasks, completed by the function, using thread management for efficient concurrent operations to allow for simultaneous execution. Communication facilitation between one thread to another without direct synchronization would be achieved using a queue [3]. Queues can reduce the memory sharing and synchronization to manage task dependencies [4]. To ensure the proper chronological order of operation a synchronization mechanism, the use of methods such as locks, would need to be implemented, to prevent simultaneous modification of shared resources of these threads, otherwise known as race conditions [5]. Error handing is another consideration when running multiple threads being, identifying and debugging these errors as they arise can introduce complexity [6]. This can be addressed through centralized error management by encasing threads in try-except blocks, from which if errors are identified they would be pushed to the main thread using a queue [7]. Overall, this redesigning allows for optimization of function performance the concurrency while additionally, maintaining data integrity with centralized error reporting. A pseudocode representation of the proposed concurrent design is exemplified in Figure 2.

The wider impact of this concurrent design results in improved performance handling, especially with larger datasets with more complex workflows, which are also computationally expensive, as well as enhanced modularity increasing through the decomposition of tasks [8]. This makes the program easier to extend and maintain, complexity increase which introduced requirements of additional testing and debugging, and scalability allowing for more threads to be created and added as needed, computationally expensive, resulting in improved potential in program performance, resource optimization, and scalability.

Overall, redesigning the process_all_given_data function with python threads introduces benefits of optimization in the scope of performance and scalability processing data for larger inputs and modelizers tasks for easier debugging, testing and memory allocation [8]. Through the breakdown of multiple functions into concurrent tasks resource optimization and processing time and modulatory. While complexities can be introduced through the integration of concurrency it can be addressed through constructs within Python such as queues, synchronization methods and centralized error management. In conclusion, the concurrent redesign not only introduces an aspect of optimization, but it also takes into consideration of new, complex, and large, datasets for future endeavours.

1b) [20 marks] With specific reference to GUI interface constructs (such as text labels and buttons), and best practice regarding interface layouts, discuss how your GUI design and implementation supports THREE of the user interactions required by your prototype application. You should then justify your design decision for each, providing comparative examples to support your approach. You should aim to demonstrate as wide a range of interface constructs/layouts as your prototype application supports.

Graphical User Interfaces (GUI) is an interface that provides user experience through interactive components such as buttons, text labels, output boxes, and more. These components assist in operating efficiency and ease of interpretability of output results [9]. In reference to the application three of the user interactions required for GUI interface consists of file loading, displaying statistics, and visualizing data. Code and visual layout of the GUI are further shown in Figure 3 and 4 respectively. In the scope of this assignment the GUI programming was achieved using Tkinter which is a standardized Python library used for comprehensive and efficient GUI designs [10].

The first key user interaction the GUI supports for the application is file loading. This implementation triggered by a button called Load File, created by the tk.button function, and when clicked on it allows the user to load the CSV file and reload the converted JSON file. Dynamic feedback was also initialized in the file loading process, notifying the user if the process was a success or failure. This design better helps in reducing errors and allows for an optimized flow with error checks at each point to ensure that all steps are being met and after valid file paths are put in. Comparatively if an alternative design using text fields for file paths to load the files it would require users to manually enter paths, increasing the likelihood of mistakes or errors if the file is ever moved from its initial location. This integration makes it easier because it prevents errors comparatively from the standard manual insertion of the file path, the implementation of a failure to load dialog box just as this allows for simplicity and error handling feature as well.

The second GUI user interaction is displaying the output statistics. This is achieved through the Scrolledtex, a widget from Tkinter, from which it shows the output results such as the statistical calculated throughout the months and weeks of the semester in a dedicated output box that is also scrollable [11]. Not using a static label or a non-scorable text widget would result in users to navigate through multiple windows to access the details making it inefficient especially if the output data is long. Therefore, the Scrolledtex is the ideal method as it provides not only provides a dedicated space for output with a sufficient layout, but also a scrollable interface making the output easily interpretable especially for longer outputs.

The third GUI interaction is the visualizing data of all interaction components through plot correlation using a heatmap, and a stacked bar graph. Dedicated buttons of "Correlation Heatmap" and "Stacked Bar Graph" provides an intuitive and efficiency method for visualization outputs. These interactions generate the visualizations of the statistical outputs using Matplotlib [12][13]. With allocated clickable buttons for specific visualizations allows for simplicity in navigation and visualization. Furthermore, the representation of these outputs results through heatmaps and stacked bar graphs better assist in analyzing and identifying trends that may otherwise not be evident from the initial raw data or processed quantitative outputs, providing comprehensive insight. Alternative designs such as dropdown menus for visualization would complicate the workflow and its navigation, taking away the strengths for the use of visual interpretation.

In conclusion, GUI provides the best layout for three key interactions file loading, displaying statistics, and visualizing data. Each interaction has taken into consideration the desired output and utilizes the appropriate library and constructs such as file dialogs and buttons for file loading, ScrolledText widgets for statistical outputs and Matplotlib for vitalizations. While alternate designs may be available this mode proves simplicity and universal accessibility.

Section 2:
Design decisions supported by code samples (40%, 1,200 words, up to 6 pages in the appendices)

2a) [10 marks] With specific reference to the data manipulation requirements, REMOVE and RESHAPE, discuss your reasoning for your selected data format (JSON, XML, or entity relationship structure), and what advantages/disadvantages it has demonstrated in this context.

It is expected that this question can be reasonably addressed within 400 words with no more than 1 page of appendices for code samples, or data format samples. This section will require appropriate citations to achieve a pass.

When deciding upon which data format to use for the assignment the data manipulation requirements, such as REMOVE and RESHAPE, must be taken into consideration. In the case of this assignment the JSON data format was chosen between XML, or an entity relationship structure (ERS) due its characteristic profile best matching the given dataset.

The JSON data format was chosen used due to its simplicity, flexibility and integration with Python's data manipulation libraries. JSON has a lightweight and simple data making user interpretation readily understood and can handle different data types in Python, such as dictionaries and lists, due to the data format structure [14][15]. With code is exemplified in Figure 5. This in turn allows for efficiency in operations such as filtering, removing unwanted components, and reshaping, or restructuring data, facilizing obtaining interaction count and grouping by components for analysis and interpretation. Furthermore, data manipulation libraries, such as Pandas, allows for more complex data manipulation and statical analysis if required [16]. Another advantage is that JSON files have data storage versatility by handling both flat and hierarchical data formats such as tabular complexity from lists to object relationships, as required for the application [17].

In comparison to JSON formats, XML is more complex, with more overhead requirements and complicated syntax making data manipulation tasks more difficult [18]. When comparing JSON formats with ERS, is that ERS would require the use of a relational database management system once again introducing complexity and additional resources [19]. Comparatively both data formats would require more modifications and complex queries for execution of reshape operations such as RESHAPE and REMOVE, which are expensive time and resource is in comparison to manipulating JSON files using Python.

Disadvantages of the JSON format are its lack of schema enforcement which can result in inconsistencies in the structure and format of the data [20]. Another disadvantage is the program overhead required for large datasets making it memory intensive in comparison to more efficient formats such as binary [21].

In conclusion, JSON was chosen over XML and ERS for its lightweight design, flexibility and integration with Python libraries and operations simplifying operations for the assignment such as REMOVE and RESHAPE. While it does lack schema enforcement and memory intensive for large datasets its simplicity and efficacy for the conditions of this assignment make it the ideal choice.

2b) [30 marks] For each of OUTPUT STATISTICS, GRAPHS, and CORRELATION discuss and demonstrate, via appropriate code samples and program output, the following:

- **Any additional cleaning you have undertaken and justify it in the context of the relevant output(s). State clearly if you have carried out no additional cleaning, and justify why you chose not to do so.**
- **Explain why the APIs you selected for data analysis were chosen over other available options, focusing on how they are suited to producing the desired outputs.**
- **Provide a clear code example of how you have applied the selected API's to achieve each output.**
- **What you observe from each output and what conclusion/s you can draw from it, if any.**

The first output, provided the statistical summaries, giving the mean, median and mode of all the user interactions throughout various categorical components spanning weeks and months. Weekly statistics derived, results and code shown in Table 1 and Figure 6 respectively, to validate all category calculations. In preparation of these statistical results, additional cleaning was required for further validity. In which if missing interactions, or values, were identified they were replaced with a '0' which was achieved using fillna(0), to remove null values and prevent skewing of the data resulting in inaccurate interpretations. The API chosen was a Pandas library for its optimal data manipulation and built-in functions for computing summary statistics and handling tabular data compared to other libraries such as NumPy [22].

The results derived for the Semester Summary values, displayed in Table 2 with code represented in Figure 7, consisted of mean, median and mode vales for interaction components. From which, components Quiz and Assignment displayed mean average values of 9305.36 and 21066.92, respectively, both categories displayed high engagement, but higher variability was attributed to Quiz and more consistent interaction with Assignment. The variables Attendance and Lectures displayed mean values of 249.81 and 3645.95 respectively, low median values, and a mode of 0, indicating significant non-participation. Survey displayed some engagement, but lower median values, suggesting limited student participation. Overall, these results display strong student engagement in both Assignments and Quiz components, minimal interactions for Attendance and Lecture and moderate interaction with Surveys. This may be attributed to both assignments and quizzes being necessary and important components to passing the course and most students skipping lectures for various reasons.

Monthly statistics results, shown in Table 3 and code in Figure 8, displayed the student engagement trends for variables quizzes, lectures, assignments, attendance, and survey participation. Mean values most components were increasing with the highest value in November, Quizzes displaying a mean value of 72703.55, Lecture with 27385.73, Assignment with 160091.77, Attendance with 959.21, and Survey with 4312.07. All these peaks are shown to be late in the semester which is exam periods which may attribute to the high engagement across all categories at that time.

The second graphical output also required additional cleaning removing categories such as System and folder, ensuring that the data was focused on the relevant data. The API chosen for graph generation was Matplotlib, due to its customization options make stacked bar graphs a simple process for its control over the plot in comparison to alternatives such as Seaborn, which do not have the same abilities such as precise stacking of bars and label placement [23].

The Stacked Bar graph, shown in Figure 9 and code in Figure 10, displayed the monthly user interactions across the key components. The trends followed similar patterns as in the statistical monthly output discussed previously. From which initial monthly from January to September showed consistent low or minimal interaction, with peak interactions cross-category in October and November. With engagement drastically declining in December onwards indicating the end of the semester.

The third output, correlation analysis, was visually represented using a heatmap, displayed in Figure 11 and code in Figure 12. Additional cleaning was required to exclude non-numerical columns from the dataset. The API selection for this output included Seaborn to generate heatmap compared to Matplotlib Seaborn provides aesthetically pleasing heatmaps [24]. A Pearsons correlation analysis was conducted, using .corr() function on the data frame, to analyze the linear relationship between determined course components. The Seaborn library was used to develop a heatmap allowing the application displays a visual of the correlation matrix using the optimized dataset [24]. The correlation matrix displays the correlation strength between two variables ranging from 1.0, for a strong correlation, to 0.0 to no correlation, a color sliding scale and value label is also implemented and each cell on the heatmap.

The heatmap provided a visual representation of the correlation between course components and student engagement. From which, Assignment showed the strongest positive correlations with Course and Lecture, with a value of 0.93 and 0.90 respectively. Indicating that interactions with Course and Lecture indicated better performance on Assignment. Similarly, a strong correlation between Quiz and Assignment, with a value of 0.83, indicate that students that interact with Assignments do well on Quizzes. Lecture strong correlated with Book, with a value of 0.82, indicating that reading books improving comprehension and retention in lectures. Project shows moderate correlation with course, 0.59, inducing there is some student influence between course and project. User ID showed weak, or no correlation throughout all interaction components indicating no relatable influence between them.

Overall, these results provide better perspective of influencing factors of student engagement and where resources need to be focused to improve student engagement which in turn enhance the student experience and the course output in terms of results and knowledge.

Section 3:
Reflection on the ethical, moral and legal aspects (20%, 600 words)

***Evidence for learning outcome:* Critically evaluate the legal and ethical impact of software developments within real-world contexts. [MLO4]**

3) [20 marks] Reflect on the ethical, moral and legal aspects of computing, as discussed in the module, and demonstrate an awareness of how these need to be considered in the role of a software engineer. Critically evaluate the following statement by building an effective 'for' or 'against' argument. This should be supported by the literature, using comparative examples, and recognition of the opposition's position where appropriate.

"The moderation of social media platforms by their owners/operators is robust, fair, and effective at removing problematic content. Consequently, software engineers should not be required to consider the ethical, moral or legal consequences of employing user-submitted social media content as training data for machine learning."

Software developments, such as Artificial Intelligence, have been proven to as the forefront for driving advancement, however, since its development, questions of integrity, legal, and moral aspects of computing have arisen. As these developments continue to be implemented and advance, ethical considerations have been co-evolving alongside through governing bodies such as ethics boards. It is important to note that these algorithms span many disciplines that collect and interact with both confidential and public data, such as medical information, on both a local and global user scale, bringing into question the integrity of strong and weak AI and global regulation [25]. Therefore, the question of ethical responsibility should be placed amongst all those directly involved in its inception, maintenance and advancement.

Social media platforms are one of these domains, here ethical responsibility concerns the moderation of social media and the use of its data. While current efforts have been effective at removing problematic content before it is implemented for training data for machine learning as evident by some of the largest conglomerates such as Google and Facebook have shown effective practices in this domain. However, moderation has been shown to exemplify bias and inconsistency in not only the oversight of unwanted data but the premeditated use of data to influence important factors to their benefit. This is a dangerous avenue as it can lead to the retention and use of harmful content such as the influence of politics and elections [26]. An instance of this was various social media platforms such as X, Facebook, and Instagram having influenced the political sphere across various demographics through clear tactics using data to create segregated views alongside fear and hate tactics [27][28].

It can be noted that these companies, much like many others, all agree to comply with the ethical and legal frameworks protecting user integrity. In which they are held responsible, and failure to comply can result in penalties and to certain extents investigations. While this can be proven effective it still has its pitfalls as nuanced issues, such as privacy violations or content use, especially sensitive content, can draw blurred lines. One instance of this is the Cambridge Analytica scandal, in which personal user data was taken from Facebook user without prior knowledge or consent from a third-party application, and the data was then used for targeted political campaigns for the 2016 U.S. election raising concerns for data privacy [29].

Perspective can be noted that enforcing these thetical responsibilities on software engineering may add another layer of stress and may in turn be an unrealistic standard to comply with omnipresence is not a realistic endeavour as errors will arise. Similarly, engineers should have the ethical consideration of the content submitted and what that content is to be used for.

Overall, both perspectives have their strengths and weaknesses when considering the legal and ethical impact of software developments within real-world contexts. A solution to this is a halfway point where ethical considerations and checkpoints should be set, at various developmental or optimization stages, and validated by the engineers, within realistic standards. To ensure compliance to ethical standards but for sure quality and experience as well.

References

[1] M. Lotfinejad, “Multithreading in Python: The Ultimate Guide (with Coding Examples),” Dataquest. Accessed: Nov. 28, 2024. [Online]. Available: https://www.dataquest.io/blog/multithreading-in-python/

[2] “Concurrency in Python,” GeeksforGeeks. Accessed: Dec. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/python-program-with-concurrency/

[3] A. Carattino, “Handling and Sharing Data Between Threads,” Python for the Lab. Accessed: Dec. 07, 2024. [Online]. Available: https://pythonforthelab.com/blog/handling-and-sharing-data-between-threads

[4] “queue — A synchronized queue class,” Python documentation. Accessed: Nov. 28, 2024. [Online]. Available: https://docs.python.org/3/library/queue.html

[5] R. Python, “Python Thread Safety: Using a Lock and Other Techniques – Real Python.” Accessed: Dec. 07, 2024. [Online]. Available: https://realpython.com/python-thread-lock/

[6] D. Giebas and R. Wojszczyk, “Detection of Concurrency Errors in Multithreaded Applications Based on Static Source Code Analysis,” *IEEE Access*, vol. 9, pp. 61298–61323, 2021, doi: 10.1109/ACCESS.2021.3073859.

[7] “Python Try Except: Examples And Best Practices • Python Land Tutorial,” Python Land. Accessed: Dec. 07, 2024. [Online]. Available: https://python.land/deep-dives/python-try-except

[8] N. Perera *et al.*, “In-depth analysis on parallel processing patterns for high-performance Dataframes,” *Future Generation Computer Systems*, vol. 149, pp. 250–264, Dec. 2023, doi: 10.1016/j.future.2023.07.007.

[9] A. Oulasvirta, N. R. Dayama, M. Shiripour, M. John, and A. Karrenbauer, “Combinatorial Optimization of Graphical User Interface Designs,” *Proceedings of the IEEE*, vol. 108, no. 3, pp. 434–464, Mar. 2020, doi: 10.1109/JPROC.2020.2969687.

[10] R. Python, “Python GUI Programming With Tkinter – Real Python.” Accessed: Dec. 07, 2024. [Online]. Available: https://realpython.com/python-gui-tkinter/

[11] “Python Tkinter - ScrolledText Widget,” GeeksforGeeks. Accessed: Dec. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/python-tkinter-scrolledtext-widget/

[12] “How to embed Matplotlib charts in Tkinter GUI?,” GeeksforGeeks. Accessed: Dec. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/how-to-embed-matplotlib-charts-in-tkinter-gui/

[13] P. Barrett, J. Hunter, J. T. Miller, J.-C. Hsu, and P. Greenfield, “matplotlib -- A Portable Python Plotting Package,” in *ResearchGate*, Dec. 2005. Accessed: Dec. 07, 2024. [Online]. Available: https://www.researchgate.net/publication/234238535_matplotlib_--_A_Portable_Python_Plotting_Package

[14] J. Xin *et al.*, “Cross-linking BioThings APIs through JSON-LD to facilitate knowledge exploration,” *BMC Bioinformatics*, vol. 19, no. 1, p. 30, Feb. 2018, doi: 10.1186/s12859-018-2041-5.

[15] H. K. Dhalla, “A Performance Analysis of Native JSON Parsers in Java, Python, MS.NET Core, JavaScript, and PHP,” in *2020 16th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey: IEEE, Nov. 2020, pp. 1–5. doi: 10.23919/CNSM50824.2020.9269101.

[16] V. Paruchuri, “Working with Large JSON Files in Python,” Dataquest. Accessed: Dec. 10, 2024. [Online]. Available: https://www.dataquest.io/blog/python-json-tutorial/

[17] Maciek, “CSV vs JSON vs XML - The Best Comparison Guide 2024,” Sonra. Accessed: Dec. 10, 2024. [Online]. Available: https://sonra.io/csv-vs-json-vs-xml/

[18] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, “Comparison of JSON and XML Data Interchange Formats: A Case Study”.

[19] J. Dullea, I.-Y. Song, and I. Lamprou, “An analysis of structural validity in entity-relationship modeling,” *Data & Knowledge Engineering*, vol. 47, no. 2, pp. 167–205, Nov. 2003, doi: 10.1016/S0169-023X(03)00049-1.

[20] D. Snell @DOMARTISAN, “Data Formats in ETL: Understanding CSV, JSON, XML, Parquet, and Avro,” Artisan. Accessed: Dec. 08, 2024. [Online]. Available: https://domartisan.com/blog/data-formats-in-etl-understanding-csv-json-xml-parquet-and-avro/

[21] “Understanding Data Serialization: A Format Comparison Guide.” Accessed: Dec. 08, 2024. [Online]. Available: https://celerdata.com/glossary/understanding-data-serialization-a-format-comparison-guide

[22] W. McKinney, “Data Structures for Statistical Computing in Python,” presented at the Python in Science Conference, Austin, Texas, 2010, pp. 56–61. doi: 10.25080/Majora-92bf1922-00a.

[23] “Difference Between Matplotlib VS Seaborn,” GeeksforGeeks. Accessed: Dec. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/difference-between-matplotlib-vs-seaborn/

[24] “Seaborn Heatmaps: A Guide to Data Visualization.” Accessed: Dec. 07, 2024. [Online]. Available: https://www.datacamp.com/tutorial/seaborn-heatmaps

[25] “Technology Landscape Artificial Intelligence | Ethics Board.” Accessed: Dec. 08, 2024. [Online]. Available: https://www.ethicsboard.org/focus-areas/technology-landscape-artificial-intelligence

[26] “U-M study explores how political bias in content moderation on social media feeds echo chambers,” University of Michigan News. Accessed: Dec. 07, 2024. [Online]. Available: https://news.umich.edu/u-m-study-explores-how-political-bias-in-content-moderation-on-social-media-feeds-echo-chambers/

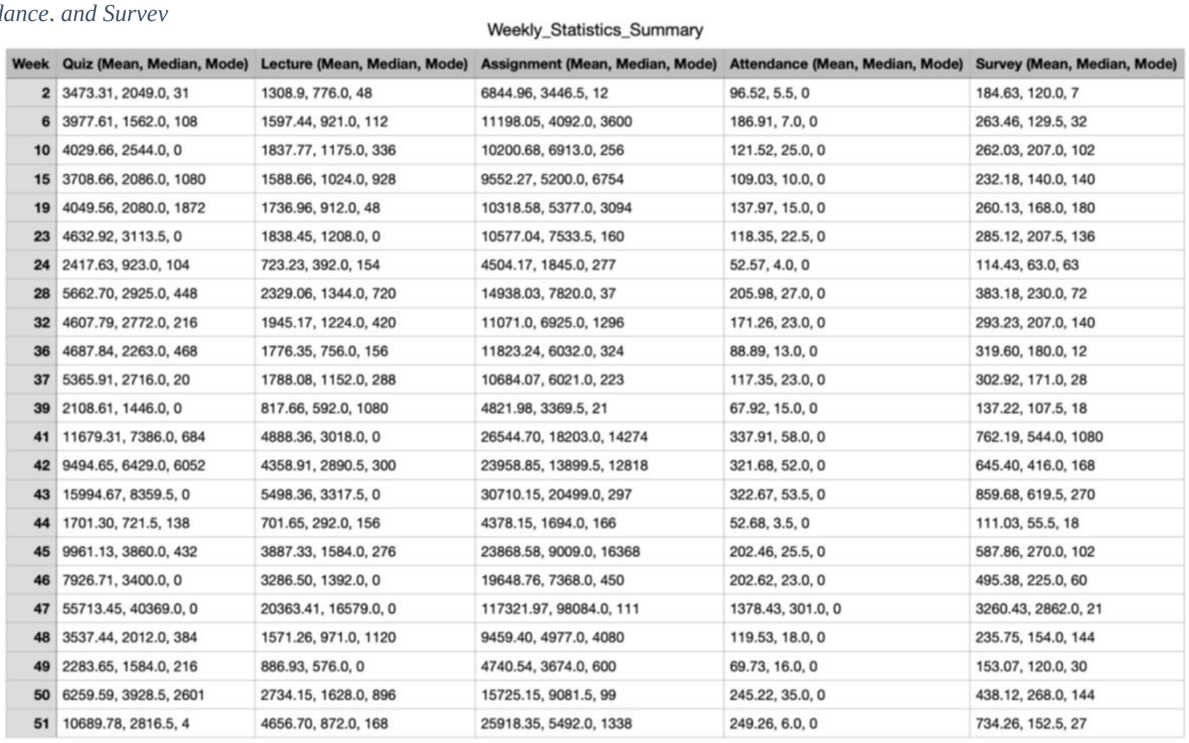
[27] “OII | Election Interference: How tech, race, and disinformation can influence the U.S Election.” Accessed: Dec. 07, 2024. [Online]. Available: https://www.oii.ox.ac.uk/news-events/election-interference-how-tech-race-and-disinformation-can-influence-the-us-elections

[28] “Social Media Election Policies Tracker - Pitt Cyber.” Accessed: Dec. 07, 2024. [Online]. Available: https://apps.cyber.pitt.edu/social-media-election-policies/timeline?utm_source=chatgpt.com

[29] “History of the Cambridge Analytica Controversy | Bipartisan Policy Center.” Accessed: Dec. 07, 2024. [Online]. Available: https://bipartisanpolicy.org/blog/cambridge-analytica-controversy/

Appendix

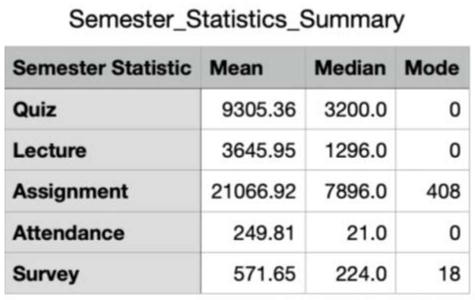
Table 1 Displays the Weekly Statistics Summaries throughout the year of 2023. Displaying the Mean, Median and Mode for the course components of Quiz, Lecture, Assignment, Attendance, and Survey



Weekly_Statistics_Summary

Week	Quiz (Mean, Median, Mode)	Lecture (Mean, Median, Mode)	Assignment (Mean, Median, Mode)	Attendance (Mean, Median, Mode)	Survey (Mean, Median, Mode)
2	3473.31, 2049.0, 31	1308.9, 776.0, 48	6844.96, 3446.5, 12	96.52, 5.5, 0	184.63, 120.0, 7
6	3977.61, 1562.0, 108	1597.44, 921.0, 112	11198.05, 4092.0, 3600	186.91, 7.0, 0	263.46, 129.5, 32
10	4029.66, 2544.0, 0	1837.77, 1175.0, 336	10200.68, 6913.0, 256	121.52, 25.0, 0	262.03, 207.0, 102
15	3708.66, 2086.0, 1080	1588.66, 1024.0, 928	9552.27, 5200.0, 6754	109.03, 10.0, 0	232.18, 140.0, 140
19	4049.56, 2080.0, 1872	1736.96, 912.0, 48	10318.58, 5377.0, 3094	137.97, 15.0, 0	260.13, 168.0, 180
23	4632.92, 3113.5, 0	1838.45, 1208.0, 0	10577.04, 7533.5, 160	118.35, 22.5, 0	285.12, 207.5, 136
24	2417.63, 923.0, 104	723.23, 392.0, 154	4504.17, 1845.0, 277	52.57, 4.0, 0	114.43, 63.0, 63
28	5662.70, 2925.0, 448	2329.06, 1344.0, 720	14938.03, 7820.0, 37	205.98, 27.0, 0	383.18, 230.0, 72
32	4607.79, 2772.0, 216	1945.17, 1224.0, 420	11071.0, 6925.0, 1296	171.26, 23.0, 0	293.23, 207.0, 140
36	4687.84, 2263.0, 468	1776.35, 756.0, 156	11823.24, 6032.0, 324	88.89, 13.0, 0	319.60, 180.0, 12
37	5365.91, 2716.0, 20	1788.08, 1152.0, 288	10684.07, 6021.0, 223	117.35, 23.0, 0	302.92, 171.0, 28
39	2108.61, 1446.0, 0	817.66, 592.0, 1080	4821.98, 3369.5, 21	67.92, 15.0, 0	137.22, 107.5, 18
41	11679.31, 7386.0, 684	4888.36, 3018.0, 0	26544.70, 18203.0, 14274	337.91, 58.0, 0	762.19, 544.0, 1080
42	9494.65, 6429.0, 6052	4358.91, 2890.5, 300	23958.85, 13899.5, 12818	321.68, 52.0, 0	645.40, 416.0, 168
43	15994.67, 8359.5, 0	5498.36, 3317.5, 0	30710.15, 20499.0, 297	322.67, 53.5, 0	859.68, 619.5, 270
44	1701.30, 721.5, 138	701.65, 292.0, 156	4378.15, 1694.0, 166	52.68, 3.5, 0	111.03, 55.5, 18
45	9961.13, 3860.0, 432	3887.33, 1584.0, 276	23868.58, 9009.0, 16368	202.46, 25.5, 0	587.86, 270.0, 102
46	7926.71, 3400.0, 0	3286.50, 1392.0, 0	19648.76, 7368.0, 450	202.62, 23.0, 0	495.38, 225.0, 60
47	55713.45, 40369.0, 0	20363.41, 16579.0, 0	117321.97, 98084.0, 111	1378.43, 301.0, 0	3260.43, 2862.0, 21
48	3537.44, 2012.0, 384	1571.26, 971.0, 1120	9459.40, 4977.0, 4080	119.53, 18.0, 0	235.75, 154.0, 144
49	2283.65, 1584.0, 216	886.93, 576.0, 0	4740.54, 3674.0, 600	69.73, 16.0, 0	153.07, 120.0, 30
50	6259.59, 3928.5, 2601	2734.15, 1628.0, 896	15725.15, 9081.5, 99	245.22, 35.0, 0	438.12, 268.0, 144
51	10689.78, 2816.5, 4	4656.70, 872.0, 168	25918.35, 5492.0, 1338	249.26, 6.0, 0	734.26, 152.5, 27

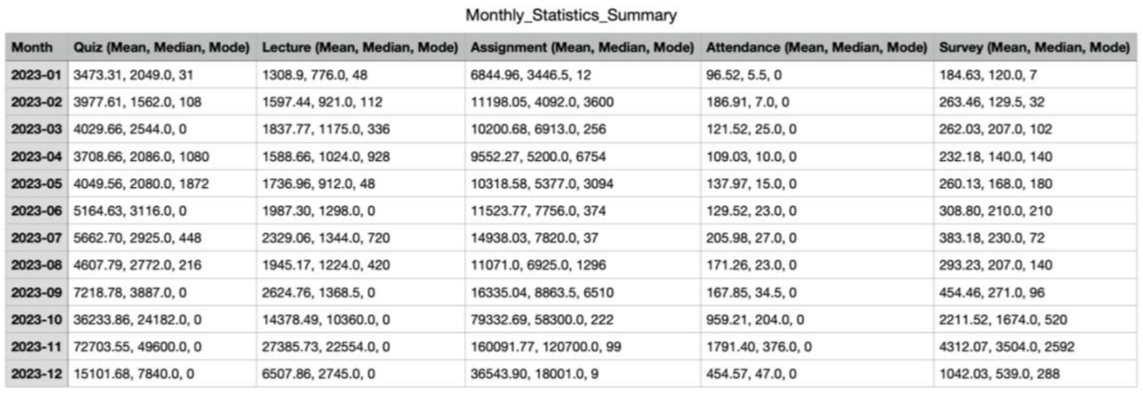
Table 2 Displays the Semester Statistics Summaries throughout the academic semester of 2023. Displaying the Mean, Median and Mode for the course components of Quiz, Lecture, Assignment, Attendance, and Survey



Semester_Statistics_Summary

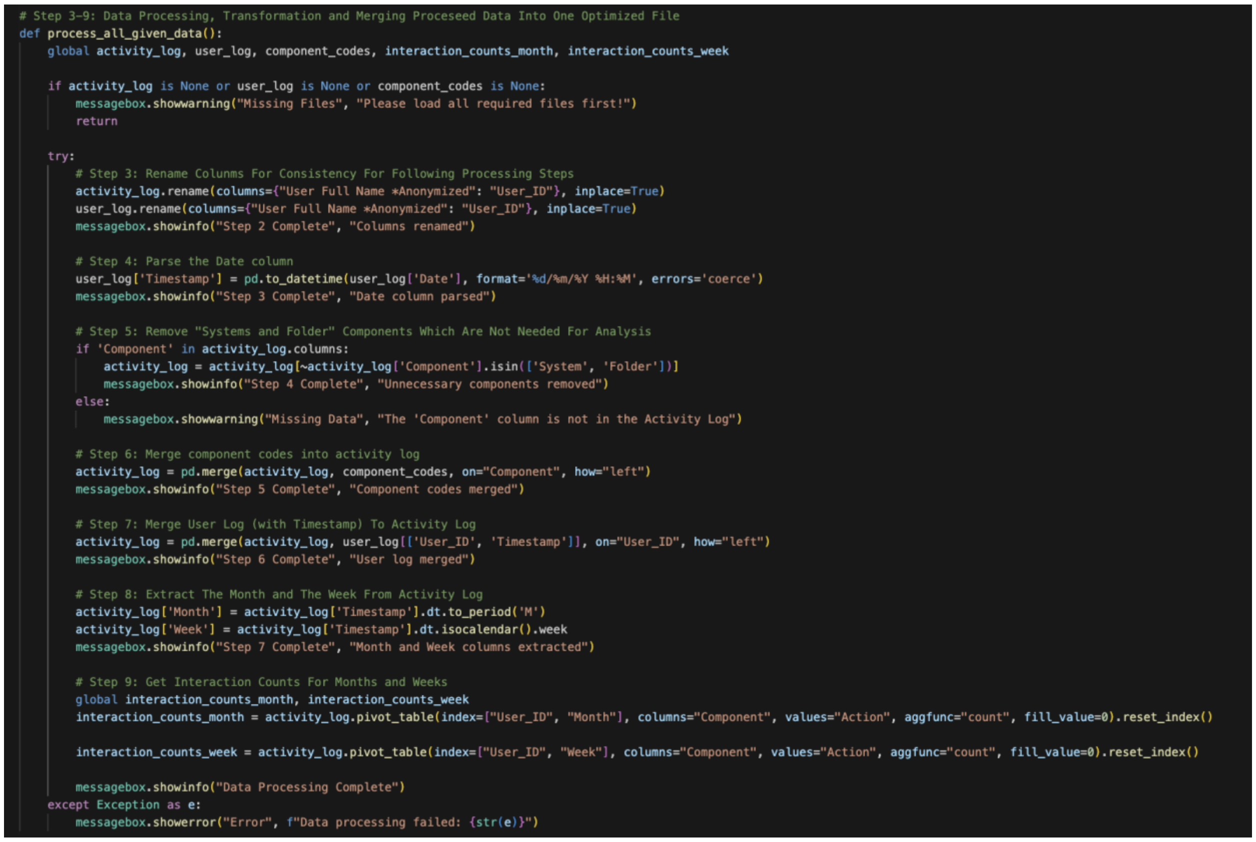
Semester Statistic	Mean	Median	Mode
Quiz	9305.36	3200.0	0
Lecture	3645.95	1296.0	0
Assignment	21066.92	7896.0	408
Attendance	249.81	21.0	0
Survey	571.65	224.0	18

Table 3 Displays the Monthly Statistics Summaries for each month throughout the year of 2023. Displaying the Mean, Median and Mode for the course components of Quiz, Lecture, Assignment, Attendance, and Survey



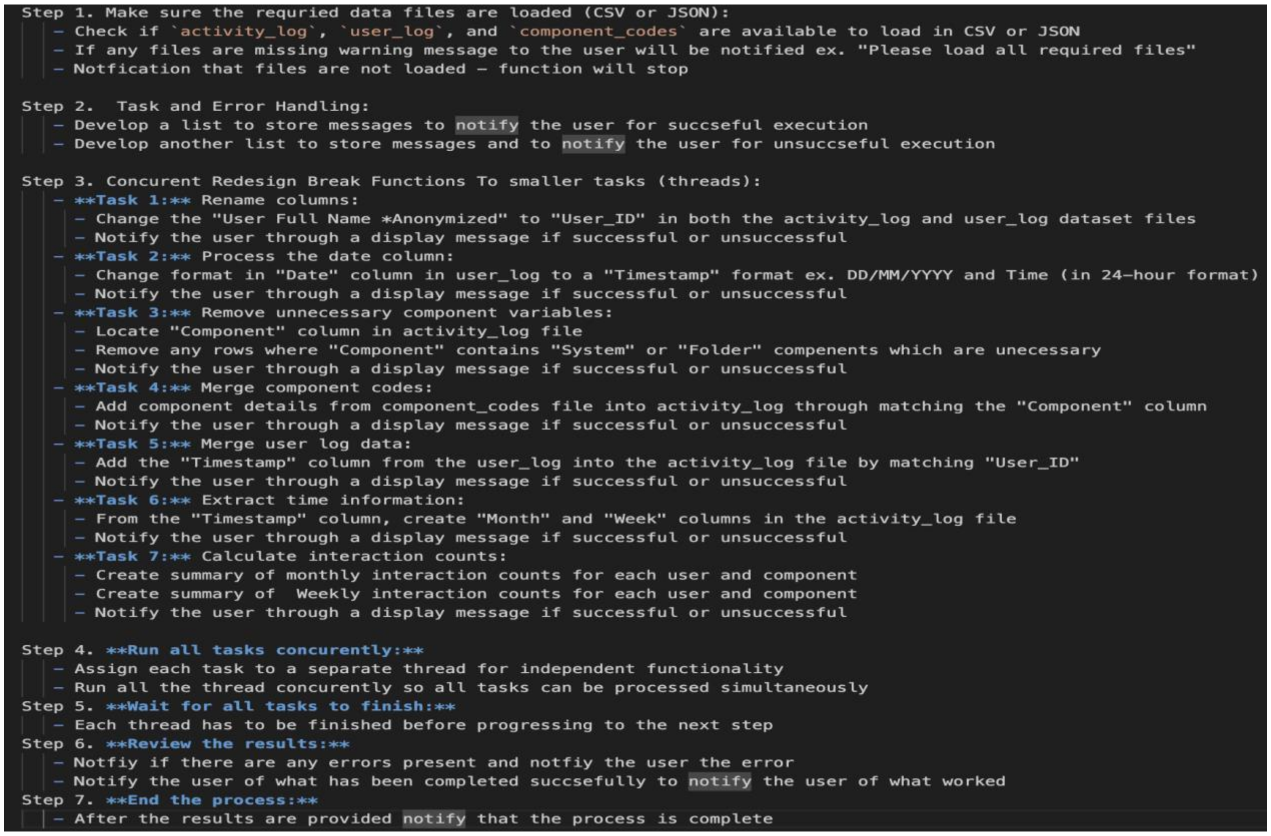
Monthly_Statistics_Summary

Month	Quiz (Mean, Median, Mode)	Lecture (Mean, Median, Mode)	Assignment (Mean, Median, Mode)	Attendance (Mean, Median, Mode)	Survey (Mean, Median, Mode)
2023-01	3473.31, 2049.0, 31	1308.9, 776.0, 48	6844.96, 3446.5, 12	96.52, 5.5, 0	184.63, 120.0, 7
2023-02	3977.61, 1562.0, 108	1597.44, 921.0, 112	11198.05, 4092.0, 3600	186.91, 7.0, 0	263.46, 129.5, 32
2023-03	4029.66, 2544.0, 0	1837.77, 1175.0, 336	10200.68, 6913.0, 256	121.52, 25.0, 0	262.03, 207.0, 102
2023-04	3708.66, 2086.0, 1080	1588.66, 1024.0, 928	9552.27, 5200.0, 6754	109.03, 10.0, 0	232.18, 140.0, 140
2023-05	4049.56, 2080.0, 1872	1736.96, 912.0, 48	10318.58, 5377.0, 3094	137.97, 15.0, 0	260.13, 168.0, 180
2023-06	5164.63, 3116.0, 0	1987.30, 1298.0, 0	11523.77, 7756.0, 374	129.52, 23.0, 0	308.80, 210.0, 210
2023-07	5662.70, 2925.0, 448	2329.06, 1344.0, 720	14938.03, 7820.0, 37	205.98, 27.0, 0	383.18, 230.0, 72
2023-08	4607.79, 2772.0, 216	1945.17, 1224.0, 420	11071.0, 6925.0, 1296	171.26, 23.0, 0	293.23, 207.0, 140
2023-09	7218.78, 3887.0, 0	2624.76, 1368.5, 0	16335.04, 8863.5, 6510	167.85, 34.5, 0	454.46, 271.0, 96
2023-10	36233.86, 24182.0, 0	14378.49, 10360.0, 0	79332.69, 58300.0, 222	959.21, 204.0, 0	2211.52, 1674.0, 520
2023-11	72703.55, 49600.0, 0	27385.73, 22554.0, 0	160091.77, 120700.0, 99	1791.40, 376.0, 0	4312.07, 3504.0, 2592
2023-12	15101.68, 7840.0, 0	6507.86, 2745.0, 0	36543.90, 18001.0, 9	454.57, 47.0, 0	1042.03, 539.0, 288



```python
# Step 3-9: Data Processing, Transformation and Merging Proceseed Data Into One Optimized File
def process_all_given_data():
    global activity_log, user_log, component_codes, interaction_counts_month, interaction_counts_week

    if activity_log is None or user_log is None or component_codes is None:
        messagebox.showwarning("Missing Files", "Please load all required files first!")
        return

    try:
        # Step 3: Rename Colunms For Consistency For Following Processing Steps
        activity_log.rename(columns={"User Full Name *Anonymized": "User_ID"}, inplace=True)
        user_log.rename(columns={"User Full Name *Anonymized": "User_ID"}, inplace=True)
        messagebox.showinfo("Step 2 Complete", "Columns renamed")

        # Step 4: Parse the Date column
        user_log['Timestamp'] = pd.to_datetime(user_log['Date'], format='%d/%m/%Y %H:%M', errors='coerce')
        messagebox.showinfo("Step 3 Complete", "Date column parsed")

        # Step 5: Remove "Systems and Folder" Components Which Are Not Needed For Analysis
        if 'Component' in activity_log.columns:
            activity_log = activity_log[~activity_log['Component'].isin(['System', 'Folder'])]
            messagebox.showinfo("Step 4 Complete", "Unnecessary components removed")
        else:
            messagebox.showwarning("Missing Data", "The 'Component' column is not in the Activity Log")

        # Step 6: Merge component codes into activity log
        activity_log = pd.merge(activity_log, component_codes, on="Component", how="left")
        messagebox.showinfo("Step 5 Complete", "Component codes merged")

        # Step 7: Merge User Log (with Timestamp) To Activity Log
        activity_log = pd.merge(activity_log, user_log[['User_ID', 'Timestamp']], on="User_ID", how="left")
        messagebox.showinfo("Step 6 Complete", "User log merged")

        # Step 8: Extract The Month and The Week From Activity Log
        activity_log['Month'] = activity_log['Timestamp'].dt.to_period('M')
        activity_log['Week'] = activity_log['Timestamp'].dt.isocalendar().week
        messagebox.showinfo("Step 7 Complete", "Month and Week columns extracted")

        # Step 9: Get Interaction Counts For Months and Weeks
        global interaction_counts_month, interaction_counts_week
        interaction_counts_month = activity_log.pivot_table(index=["User_ID", "Month"], columns="Component", values="Action", aggfunc="count", fill_value=0).reset_index()

        interaction_counts_week = activity_log.pivot_table(index=["User_ID", "Week"], columns="Component", values="Action", aggfunc="count", fill_value=0).reset_index()

        messagebox.showinfo("Data Processing Complete")
    except Exception as e:
        messagebox.showerror("Error", f"Data processing failed: {str(e)}")
```

Figure 1 Code displaying the process_all_given_data function, displaying the data processing and optimization workflow from initial files. Code consists of multi-processing steps, consisting of if else statements and try except blocks focused on data transformation, aggregation, and data consistency and optimization (parsing date columns, removal of unnecessary components i.e. System and Folder, and merging). Code also ensures that data points, are provided for specific time points, such as week, month and semester, which are further grouped by user interaction per component. The code also includes error handling for each subsequent step to ensure each function was executed correctly.



```
Step 1. Make sure the requried data files are loaded (CSV or JSON):
    - Check if `activity_log`, `user_log`, and `component_codes` are available to load in CSV or JSON
    - If any files are missing warning message to the user will be notified ex. "Please load all required files"
    - Notfication that files are not loaded - function will stop

Step 2.  Task and Error Handling:
    - Develop a list to store messages to notify the user for succseful execution
    - Develop another list to store messages and to notify the user for unsuccseful execution

Step 3. Concurent Redesign Break Functions To smaller tasks (threads):
    - **Task 1:** Rename columns:
      - Change the "User Full Name *Anonymized" to "User_ID" in both the activity_log and user_log dataset files
      - Notify the user through a display message if successful or unsuccessful
    - **Task 2:** Process the date column:
      - Change format in "Date" column in user_log to a "Timestamp" format ex. DD/MM/YYYY and Time (in 24-hour format)
      - Notify the user through a display message if successful or unsuccessful
    - **Task 3:** Remove unnecessary component variables:
      - Locate "Component" column in activity_log file
      - Remove any rows where "Component" contains "System" or "Folder" compenents which are unecessary
      - Notify the user through a display message if successful or unsuccessful
    - **Task 4:** Merge component codes:
      - Add component details from component_codes file into activity_log through matching the "Component" column
      - Notify the user through a display message if successful or unsuccessful
    - **Task 5:** Merge user log data:
      - Add the "Timestamp" column from the user_log into the activity_log file by matching "User_ID"
      - Notify the user through a display message if successful or unsuccessful
    - **Task 6:** Extract time information:
      - From the "Timestamp" column, create "Month" and "Week" columns in the activity_log file
      - Notify the user through a display message if successful or unsuccessful
    - **Task 7:** Calculate interaction counts:
      - Create summary of monthly interaction counts for each user and component
      - Create summary of  Weekly interaction counts for each user and component
      - Notify the user through a display message if successful or unsuccessful

Step 4. **Run all tasks concurently:**
    - Assign each task to a separate thread for independent functionality
    - Run all the thread concurently so all tasks can be processed simultaneously
Step 5. **Wait for all tasks to finish:**
    - Each thread has to be finished before progressing to the next step
Step 6. **Review the results:**
    - Notfiy if there are any errors present and notfiy the user the error
    - Notify the user of what has been completed succsefully to notify the user of what worked
Step 7. **End the process:**
    - After the results are provided notify that the process is complete
```

Figure 2 Pseudocode displaying the concurrent redesign of the process_all_given_data function developed through Python threads. Pseudocode displays the decomposing the function into smaller more manageable tasks that are each assigned specific threads and run independently to improve efficiency. The pseudocode also includes error handling for each subsequent step to ensure each function was executed correctly.

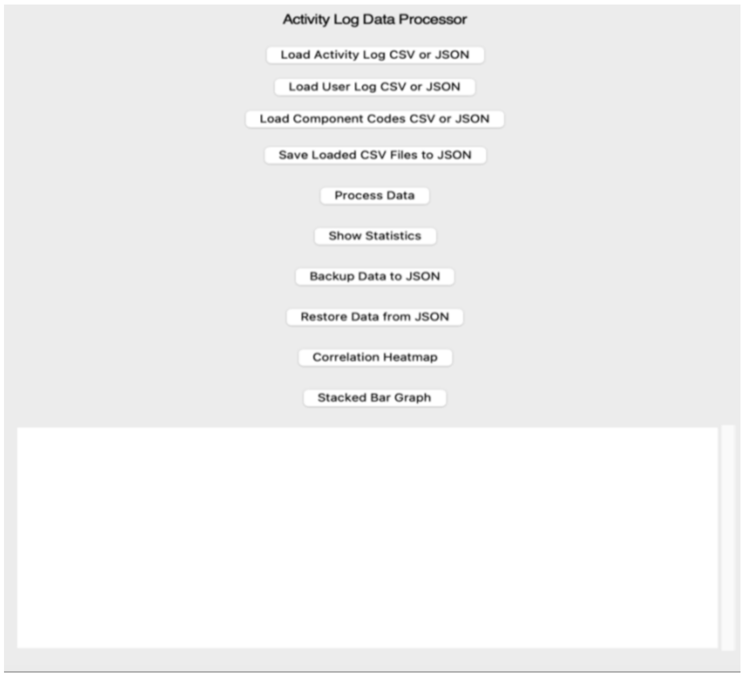
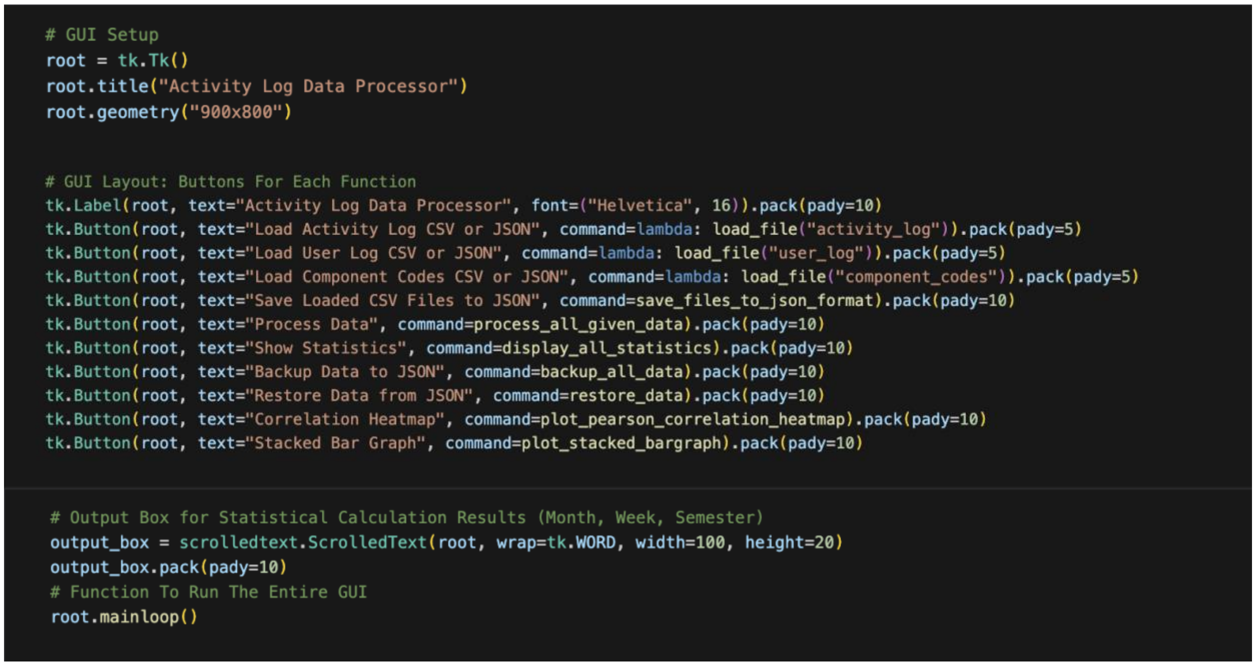
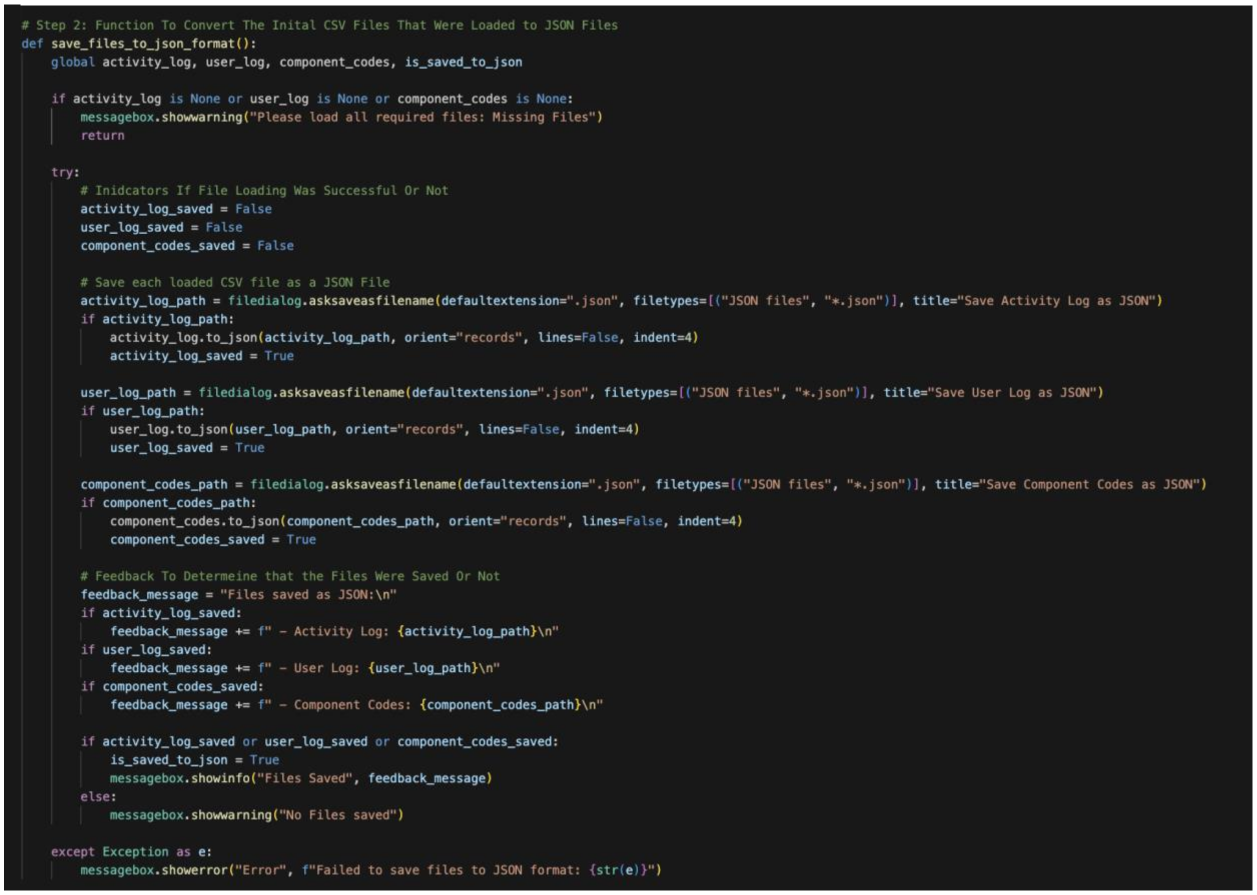

Activity Log Data Processor
Load Activity Log CSV or JSON
Load User Log CSV or JSON
Load Component Codes CSV or JSON
Save Loaded CSV Files to JSON
Process Data
Show Statistics
Backup Data to JSON
Restore Data from JSON
Correlation Heatmap
Stacked Bar Graph

Figure 3 Graphical User Interface (GUI). The interface consists of multiple buttons for each specific function exemplified in the code in either CSV or JSON format. The application allows for processing data, and displaying statistics, backing up data, restoring data, generation of correlation heatmap and stacked bar graph for data processed data visualization.



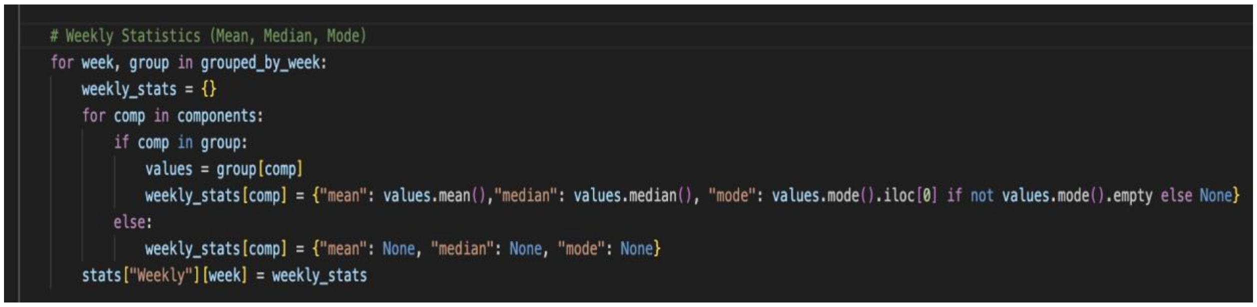
```python
# GUI Setup
root = tk.Tk()
root.title("Activity Log Data Processor")
root.geometry("900x800")


# GUI Layout: Buttons For Each Function
tk.Label(root, text="Activity Log Data Processor", font=("Helvetica", 16)).pack(pady=10)
tk.Button(root, text="Load Activity Log CSV or JSON", command=lambda: load_file("activity_log")).pack(pady=5)
tk.Button(root, text="Load User Log CSV or JSON", command=lambda: load_file("user_log")).pack(pady=5)
tk.Button(root, text="Load Component Codes CSV or JSON", command=lambda: load_file("component_codes")).pack(pady=5)
tk.Button(root, text="Save Loaded CSV Files to JSON", command=save_files_to_json_format).pack(pady=10)
tk.Button(root, text="Process Data", command=process_all_given_data).pack(pady=10)
tk.Button(root, text="Show Statistics", command=display_all_statistics).pack(pady=10)
tk.Button(root, text="Backup Data to JSON", command=backup_all_data).pack(pady=10)
tk.Button(root, text="Restore Data from JSON", command=restore_data).pack(pady=10)
tk.Button(root, text="Correlation Heatmap", command=plot_pearson_correlation_heatmap).pack(pady=10)
tk.Button(root, text="Stacked Bar Graph", command=plot_stacked_bargraph).pack(pady=10)


# Output Box for Statistical Calculation Results (Month, Week, Semester)
output_box = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=100, height=20)
output_box.pack(pady=10)
# Function To Run The Entire GUI
root.mainloop()
```

Figure 4 Code displaying GUI functionalities using Tkinter, functions allow the user to interact with GUI providing representations of statistical outputs in both numerical and graphical formats.



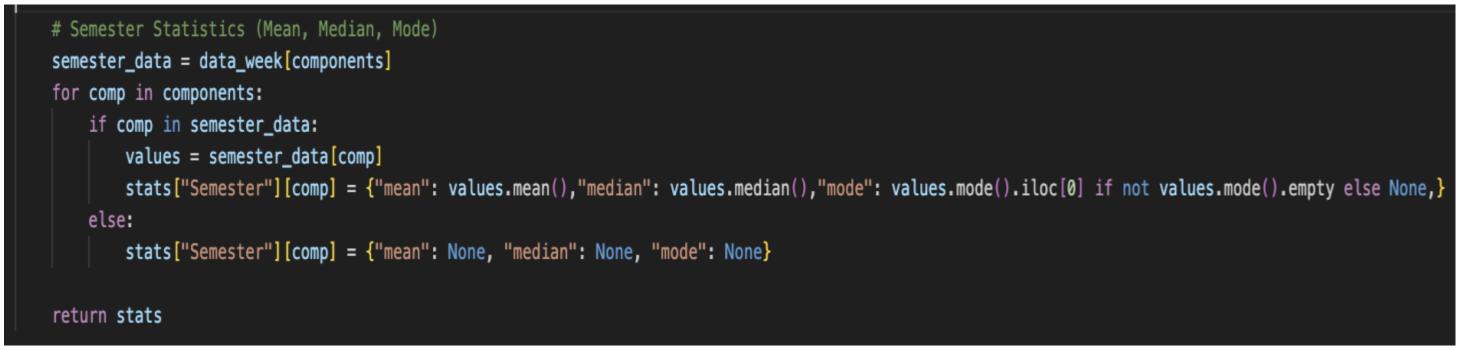
```python
# Step 2: Function To Convert The Inital CSV Files That Were Loaded to JSON Files
def save_files_to_json_format():
    global activity_log, user_log, component_codes, is_saved_to_json

    if activity_log is None or user_log is None or component_codes is None:
        messagebox.showwarning("Please load all required files: Missing Files")
        return

    try:
        # Inidcators If File Loading Was Successful Or Not
        activity_log_saved = False
        user_log_saved = False
        component_codes_saved = False

        # Save each loaded CSV file as a JSON File
        activity_log_path = filedialog.asksaveasfilename(defaultextension=".json", filetypes=[("JSON files", "*.json")], title="Save Activity Log as JSON")
        if activity_log_path:
            activity_log.to_json(activity_log_path, orient="records", lines=False, indent=4)
            activity_log_saved = True

        user_log_path = filedialog.asksaveasfilename(defaultextension=".json", filetypes=[("JSON files", "*.json")], title="Save User Log as JSON")
        if user_log_path:
            user_log.to_json(user_log_path, orient="records", lines=False, indent=4)
            user_log_saved = True

        component_codes_path = filedialog.asksaveasfilename(defaultextension=".json", filetypes=[("JSON files", "*.json")], title="Save Component Codes as JSON")
        if component_codes_path:
            component_codes.to_json(component_codes_path, orient="records", lines=False, indent=4)
            component_codes_saved = True

        # Feedback To Determeine that the Files Were Saved Or Not
        feedback_message = "Files saved as JSON:\n"
        if activity_log_saved:
            feedback_message += f" - Activity Log: {activity_log_path}\n"
        if user_log_saved:
            feedback_message += f" - User Log: {user_log_path}\n"
        if component_codes_saved:
            feedback_message += f" - Component Codes: {component_codes_path}\n"

        if activity_log_saved or user_log_saved or component_codes_saved:
            is_saved_to_json = True
            messagebox.showinfo("Files Saved", feedback_message)
        else:
            messagebox.showwarning("No Files saved")

    except Exception as e:
        messagebox.showerror("Error", f"Failed to save files to JSON format: {str(e)}")
```

Figure 5 Function code that converts initial CSV files into JSON format. The function initializes the success upload of initial CSV files provided as well if the file was successfully saved in JSON format. Error handling is also incorporated to notify the user if there are any issues during the functions process.



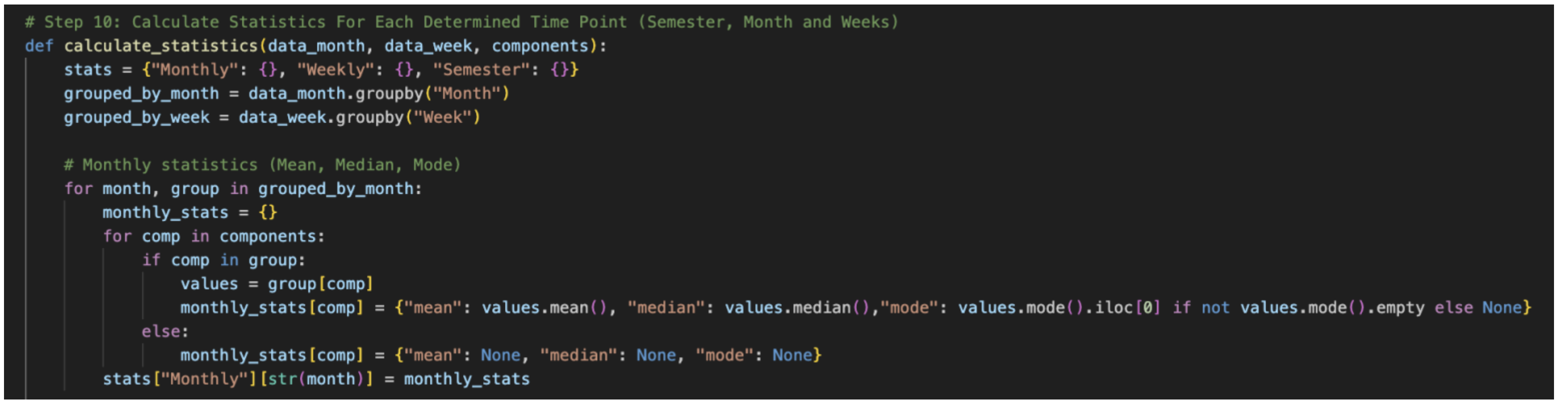
```python
# Weekly Statistics (Mean, Median, Mode)
for week, group in grouped_by_week:
    weekly_stats = {}
    for comp in components:
        if comp in group:
            values = group[comp]
            weekly_stats[comp] = {"mean": values.mean(),"median": values.median(), "mode": values.mode().iloc[0] if not values.mode().empty else None}
        else:
            weekly_stats[comp] = {"mean": None, "median": None, "mode": None}
    stats["Weekly"][week] = weekly_stats
```

Figure 6 Function to calculate the Weekly statistical output for different time points from the optimized and transformed data. Statistical output provides the mean, median and mode, for the time periods of Month, Week, and Semester. The function iterates through the data and provides the statistics specifically for each time component. If there are missing components from the dataset, that statistical output will display a "None". The results are then stored in a dictionary format for later analysis.



```python
# Semester Statistics (Mean, Median, Mode)
semester_data = data_week[components]
for comp in components:
    if comp in semester_data:
        values = semester_data[comp]
        stats["Semester"][comp] = {"mean": values.mean(),"median": values.median(),"mode": values.mode().iloc[0] if not values.mode().empty else None,}
    else:
        stats["Semester"][comp] = {"mean": None, "median": None, "mode": None}

return stats
```

Figure 7 Function code for calculating mean, median, and mode for semester statistics. The function iterates through the dataset analyzing each component and providing the statistical output (mean, media, and mode) respectively, using Pandas library and methods. If there are missing components from the dataset, that statistical output will display a "None". The results are then stored in a dictionary format for later analysis.



```python
# Step 10: Calculate Statistics For Each Determined Time Point (Semester, Month and Weeks)
def calculate_statistics(data_month, data_week, components):
    stats = {"Monthly": {}, "Weekly": {}, "Semester": {}}
    grouped_by_month = data_month.groupby("Month")
    grouped_by_week = data_week.groupby("Week")

    # Monthly statistics (Mean, Median, Mode)
    for month, group in grouped_by_month:
        monthly_stats = {}
        for comp in components:
            if comp in group:
                values = group[comp]
                monthly_stats[comp] = {"mean": values.mean(), "median": values.median(),"mode": values.mode().iloc[0] if not values.mode().empty else None}
            else:
                monthly_stats[comp] = {"mean": None, "median": None, "mode": None}
        stats["Monthly"][str(month)] = monthly_stats
```

Figure 8 Function to calculate the monthly statistical output for different time points from the optimized and transformed data. Statistical output provides the mean, median and mode, for the period of Month. The function iterates through the data and provides the statistics specifically for each time component. If there are missing components from the dataset, that statistical output will display a "None". The results are then stored in a dictionary format for later analysis.

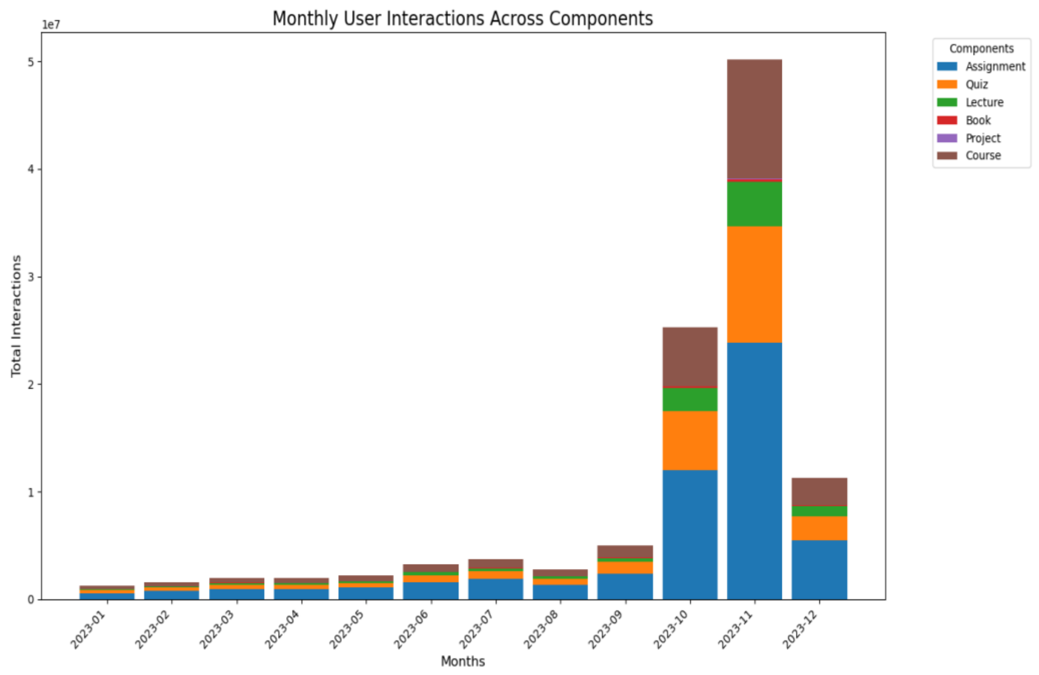
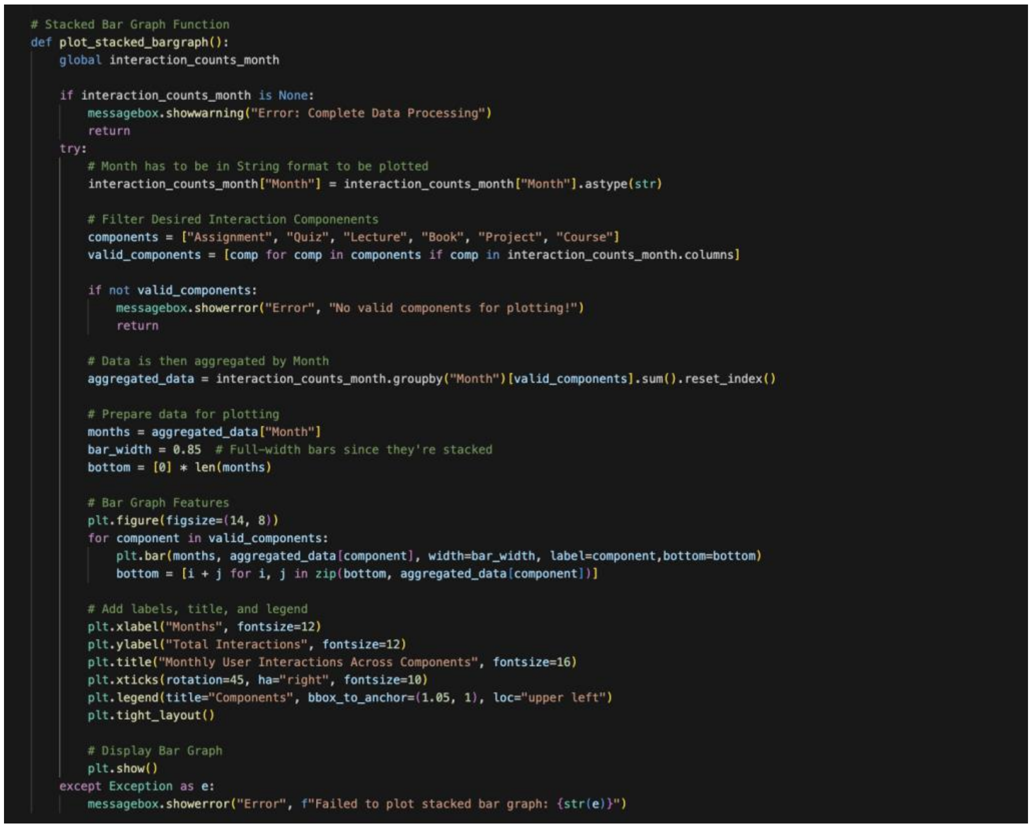


Figure 9 Stacked Bar graph displaying course components Assignment (Blue), Quiz (Orange), Lecture (Green), Book (Red), Project (Purple), and Course (Brown) and their interactions for each month throughout the entire year of 2023. Each respective component is referenced through a color coordinated legend.



```python
# Stacked Bar Graph Function
def plot_stacked_bargraph():
    global interaction_counts_month

    if interaction_counts_month is None:
        messagebox.showwarning("Error: Complete Data Processing")
        return
    try:
        # Month has to be in String format to be plotted
        interaction_counts_month["Month"] = interaction_counts_month["Month"].astype(str)

        # Filter Desired Interaction Componenents
        components = ["Assignment", "Quiz", "Lecture", "Book", "Project", "Course"]
        valid_components = [comp for comp in components if comp in interaction_counts_month.columns]

        if not valid_components:
            messagebox.showerror("Error", "No valid components for plotting!")
            return

        # Data is then aggregated by Month
        aggregated_data = interaction_counts_month.groupby("Month")[valid_components].sum().reset_index()

        # Prepare data for plotting
        months = aggregated_data["Month"]
        bar_width = 0.85  # Full-width bars since they're stacked
        bottom = [0] * len(months)

        # Bar Graph Features
        plt.figure(figsize=(14, 8))
        for component in valid_components:
            plt.bar(months, aggregated_data[component], width=bar_width, label=component,bottom=bottom)
            bottom = [i + j for i, j in zip(bottom, aggregated_data[component])]

        # Add labels, title, and legend
        plt.xlabel("Months", fontsize=12)
        plt.ylabel("Total Interactions", fontsize=12)
        plt.title("Monthly User Interactions Across Components", fontsize=16)
        plt.xticks(rotation=45, ha="right", fontsize=10)
        plt.legend(title="Components", bbox_to_anchor=(1.05, 1), loc="upper left")
        plt.tight_layout()

        # Display Bar Graph
        plt.show()
    except Exception as e:
        messagebox.showerror("Error", f"Failed to plot stacked bar graph: {str(e)}")
```

Figure 10 Function for visualizing monthly interactions for all determined components in a stacked bar graph. The function sorts the data through the set component variables and aggregates them by month for visual representation. The function also incorporates stacked bar graph features such as axes and a legend. Error handling is also included to ensure notification if any errors arise during the function execution.

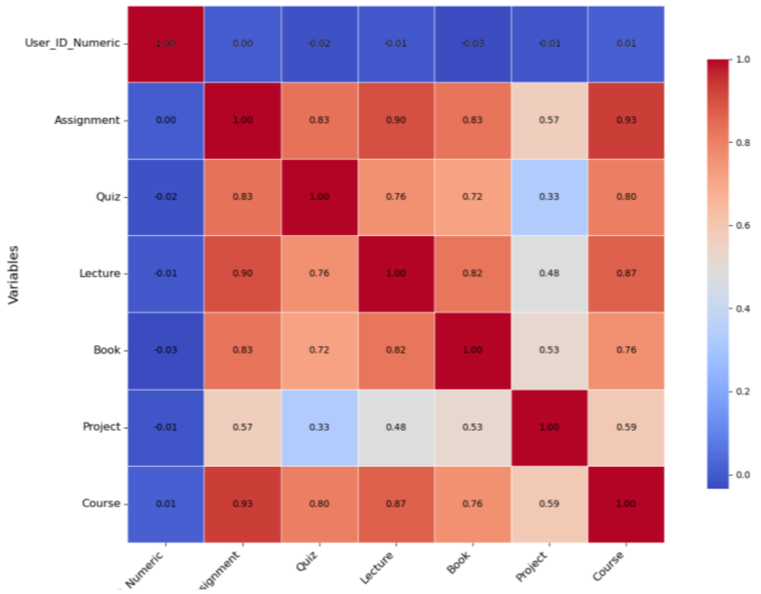


Figure 11 Correlation heatmap displaying relationships between the course components, ID_Numeric, Assignment, Quiz, Lecture, Book, Project, and Course. Correlation strengths range from 1.0, strong positive correlation, and 0 to no correlation. Strong positive correlations between Assignments and other components, while User ID displayed minimal correlation with all components

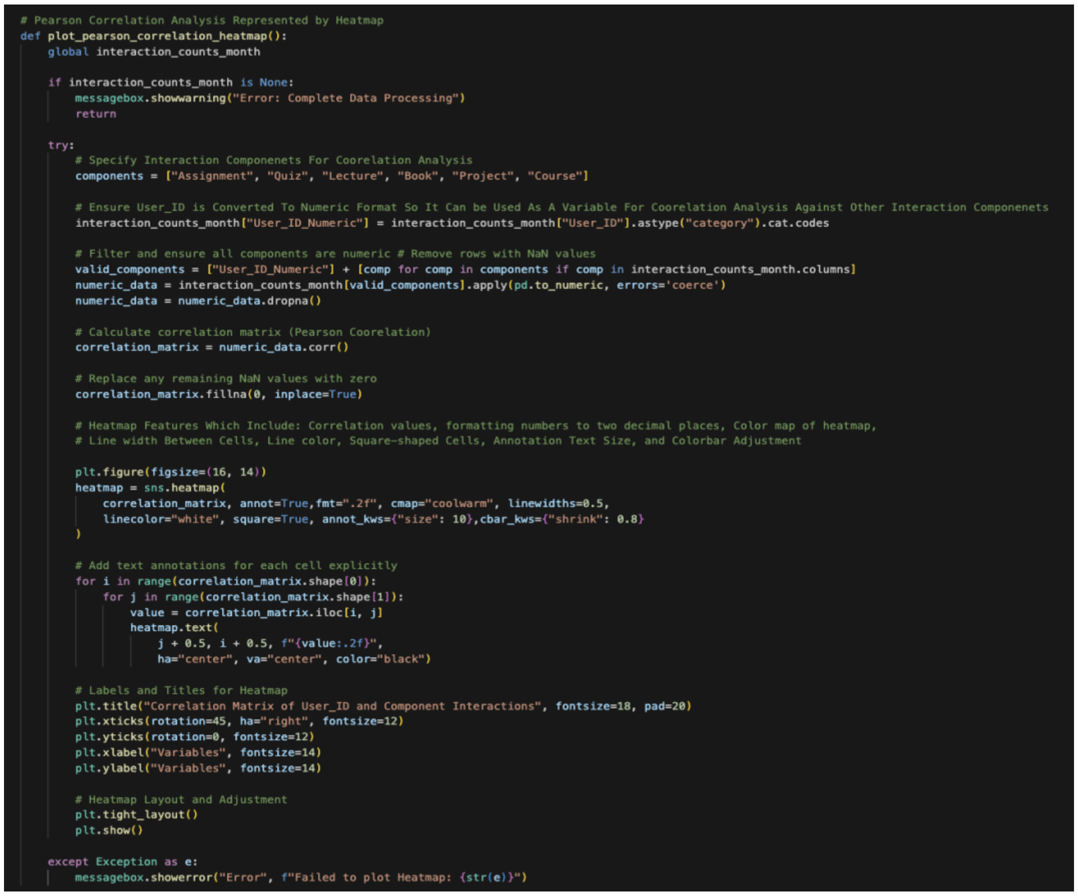

```python
# Pearson Correlation Analysis Represented by Heatmap
def plot_pearson_correlation_heatmap():
    global interaction_counts_month

    if interaction_counts_month is None:
        messagebox.showwarning("Error: Complete Data Processing")
        return

    try:
        # Specify Interaction Componenets For Coorelation Analysis
        components = ["Assignment", "Quiz", "Lecture", "Book", "Project", "Course"]

        # Ensure User_ID is Converted To Numeric Format So It Can be Used As A Variable For Coorelation Analysis Against Other Interaction Componenets
        interaction_counts_month["User_ID_Numeric"] = interaction_counts_month["User_ID"].astype("category").cat.codes

        # Filter and ensure all components are numeric # Remove rows with NaN values
        valid_components = ["User_ID_Numeric"] + [comp for comp in components if comp in interaction_counts_month.columns]
        numeric_data = interaction_counts_month[valid_components].apply(pd.to_numeric, errors='coerce')
        numeric_data = numeric_data.dropna()

        # Calculate correlation matrix (Pearson Coorelation)
        correlation_matrix = numeric_data.corr()

        # Replace any remaining NaN values with zero
        correlation_matrix.fillna(0, inplace=True)

        # Heatmap Features Which Include: Correlation values, formatting numbers to two decimal places, Color map of heatmap,
        # Line width Between Cells, Line color, Square-shaped Cells, Annotation Text Size, and Colorbar Adjustment

        plt.figure(figsize=(16, 14))
        heatmap = sns.heatmap(
            correlation_matrix, annot=True,fmt=".2f", cmap="coolwarm", linewidths=0.5,
            linecolor="white", square=True, annot_kws={"size": 10},cbar_kws={"shrink": 0.8}
        )

        # Add text annotations for each cell explicitly
        for i in range(correlation_matrix.shape[0]):
            for j in range(correlation_matrix.shape[1]):
                value = correlation_matrix.iloc[i, j]
                heatmap.text(
                    j + 0.5, i + 0.5, f"{value:.2f}",
                    ha="center", va="center", color="black")

        # Labels and Titles for Heatmap
        plt.title("Correlation Matrix of User_ID and Component Interactions", fontsize=18, pad=20)
        plt.xticks(rotation=45, ha="right", fontsize=12)
        plt.yticks(rotation=0, fontsize=12)
        plt.xlabel("Variables", fontsize=14)
        plt.ylabel("Variables", fontsize=14)

        # Heatmap Layout and Adjustment
        plt.tight_layout()
        plt.show()

    except Exception as e:
        messagebox.showerror("Error", f"Failed to plot Heatmap: {str(e)}")
```

Figure 12 Function to create a visual representation of a Pearson correlation through a heatmap. Correlation analysis was conducted to identify the relationships between the determined variable. The function processes the data through converting the columns numeric, calculating the correlation matrix, and addressing missing values. The function develops the heatmap through plotting the annotated correlation values and develops a color coordinated map and variable labels. The heatmap also includes cell annotations for each cell. Error handling is also included to ensure notification if any errors arise during the function execution.