Summative Report Advanced Programming

Word Count =

Table of Contents

| 1.1 Section One Part A | 3 |
| --- | --- |
| 1.2 Section One Part B | 4 |
| 1.3 Section One Part C | 3 |
| 2.1 Section Two Part A | 5 |
| 2.2 Section Two Part B | 6 |
| 2.3 Section Two Part C | 9 |
| 2.4 Section Two Part D | 11 |
| 3.1 Section Three Part A | 12 |
| Reference List | 15 |
| Appendix A | 18 |
| Appendix B |  |
| Appendix C |  |
| Appendix D |  |
| Appendix E |  |
| Appendix F |  |

# 

# 1.1 Section One Part A

Threads can improve performance with concurrency. For instance, User Input/Output (I/O), alongside file encoding, could utilise threading to appear to be running simultaneously for the user [Appendix A]. Importantly, threads utilised in this way will allow the Graphical User Interface (GUI) to be responsive while another process is given a time-slice of the CPU in order to improve user experience and program usability.

Threads use a shared memory space to access data and thread safety requires consideration for the exchange of data. For example, race conditions can affect the outcome of a program and can even be used to exploit a program’s security [Study of Race Condition: A Privilege Escalation Vulnerability]. Notably, a synchronisation method, such as a lock or semaphore, can be used to make sure threads access data in a controlled way which prevents race conditions. Locks have the advantage of mutual exclusion which prevent race conditions, however, there are potential deadlocks where threads can wait indefinitely for a lock to be released. In comparison, semaphores are more versatile for complex synchronisation scenarios, but implementation is more intricate than locks [Python parallel programming cookbook]. Alternatively, queues can be implemented instead of shared memory as queues ensure only one thread, or process, has access to a dataframe, making data secure and simplifying implementation. Indeed, [Transparent serverless execution of Python multiprocessing applications] found that queues are an efficient implementation for state sharing. Significantly, queues provide an advantage over shared memory for exchanging data.

Implementation of threads in Python requires consideration of the Global Interpreter Lock (GIL). Notably, threads implemented in CPython use the GIL to protect the internal state of objects and the interpreter, ensuring thread safety at the expense of enabling the serialisation execution of code [Thread and Process Efficiency in Python][https://jeffknupp.com/blog/2012/03/31/pythons-hardest-problem/]. Significantly, the GIL means only one thread is ever running at once and this prevents Python programs from taking full advantage of multithreaded systems. Moreover, threads can cause programs to run slower [On parallel software engineering education using Python] as threads are competing for a single shared resource. Comparatively, threads in Java do not have this limitation and alternative Python interpreters can by-pass this limitation. For instance, Jython has no GIL and can fully exploit the advantages of threaded applications by implementing Python on the Java Virtual Machine. This provides substantial benefits as Java’s multithreading and concurrency are utilised. However, Jython is not compatible in all design scenarios and can result in limitations with Python features [Reflections on the Compatibility, Performance, and Scalability of Parallel Python].

Alternatively, the Python multiprocessing package enables parallelism through the creation of individual processes, each with their own memory space. This enables processes to be run on individual interpreters and utilise multiple cores. However, multiprocessing generates the requirement for increased memory, which is a drawback for program scalability [Performance Comparison of Python Translators for a Multi-threaded CPU-Bound Application]. That being said, processes are isolated from each other, meaning if one process encounters an error it will not affect other processes. Significantly, this overcomes Python’s thread limitation of using shared memory.

Overall, threads are a suitable choice for user I/O functions, where GIL restrictions are less important, while multiprocessing is preferable when parallelism is required.

1.2 Section One Part B

The GUI design supports the client’s requirements by providing functionality for each task. For instance, loading csv files, data cleaning and data analysis. Moreover, a text display illustrates uploaded data and actions performed to it [Appendix B].

One interaction is the functionality to upload files. This was chosen to enable the user to upload a csv file of their choice with filedialog. Furthermore, the encoding of the file is automatically detected and applied in order to successfully read in files [Appendix B]. Importantly, this makes the program responsive, interactive and adaptable to any variation of encoding in data sets. Filedialog was chosen to achieve this as user interaction is simplified. For example, preselecting a file type extension [Appendix B] as an attribute clearly highlights if that type of file is present in a selected dictionary by the user [Tkinter GUI Application Development Cookbook]. This is beneficial as the source code does not need to be edited or required files stored in the same file directory. An alternative is the creation of a drag and drop widget. However, this adds implementation overheads for the programmer and might negatively impact user experience [Python GUI programming with Tkinter develop responsive and powerful GUI applications with Tkinter] if they are not familiar with drag and drop functionality. Therefore, enabling the user to select a file using Tkinter’s filedialog.askopenfilename, with arguments for showing only csv files in a folder, improves user experience as files are easier to locate.

Buttons were used in order to allow the user to select an option in the GUI. An advantage of using buttons for the GUI is that they are event driven and can be linked to functions [Appendix B], but offer limited interaction. Layout management is implemented with the pack function. Pack provides a simple methodology to organise widgets and is responsive to the GUI window resizing. Indeed, it can be augmented with attributes, such as padding, which make it versatile for the fast creation of GUIs [Programming in Two Semesters, chp 11][Appendix B]. On the other hand, pack is limited with the control of the exact placement of button widgets, which can impact the design of a GUI’s aesthetics and, arguably, limit the scalability of the program. An alternative is the use of Tkinter’s grid function. Grid enables control over widget placement, through rows and columns, which gives greater design management choices over the aesthetic style of the program [Python Programming Fundamentals]. However, pack enables fast prototyping of a program and fulfils the user’s requirements.

Another option to facilitate user interaction is the use of a drop down box. Drop down boxes offer advantages in preserving space on the GUI and allowing the user to make a selection, which could be linked to a submit button to execute it. For instance [pyEIA: A Python-based framework for data analysis] makes use of this feature to take in multiple user inputs. However, all options, unlike buttons, may not be initially visible to the user, potentially slowing down a user’s interaction with the program. Significantly, this level of user interaction is therefore not suitable for the specifications, and so is an unsuitable design choice.

1.3 Section One Part C

Python is a dynamically typed language which makes it more effective than Java when considering the speed of development for applications.

Python has a simpler syntax and is dynamically typed which means it is more intuitive when manipulating data containers and creating programs. This makes Python ideal for fast prototyping and development without the need for explicitly typed declarations. Whereas Java’s syntax is more complicated, but is less prone to programmatic errors. For example, Java’s advantages are that it is a strongly typed and robust language, whereas Python is more succinct and easier to use in comparison [Which programming language should students learn first?]. This illustrates why Python can be the preferred language of choice when considering the speed of development for an application. Emphasising this, [Python: The Programming Language of Future] highlights that Python is a popular language due to its syntax which improves development time and strongly suggests this is a reason for Data Scientists preference for Python. Arguably, this has contributed to Python's increased popularity and growth instead of Java for the fast development and prototyping applications.

Python’s ease of use aids development speed and can be recognised in the different ways data containers are implemented in comparison to Java. For example, Python’s List and Java’s ArrayList serve a similar purpose. However, syntax, flexibility and performance vary between both languages. For instance, syntax is relatively concise in Python and data types are not declared [Appendix C]. Notably, Python programs require less lines of code compared to Java [A COMPARATIVE ANALYSIS OF THE C++, JAVA, AND PYTHON LANGUAGES], which makes implementation easier and improves development speed. Whereas Java’s ArrayList needs to be a new instance of a class and the data type for the ArrayList declared. Notably, declaration of data types is beneficial as behaviour is more predictable. Whereas in Python data types are inferred based on the assigned value [Appendix B], which shortens code but can lead to runtime errors. This highlights how Java’s static typing and memory management, such as garbage collection, can arguably make Java a difficult language to implement compared to Python. On the other hand, Java’s implementation does come with performance benefits as there is greater control over memory management [Analysis of Garbage Collection Algorithms and Memory Management in Java] and Java is a compiled and not an interpreted language. Notably, this gives Java an advantage with Just-In-Time compilation which optimises code and is beneficial for large data sets when compared to Python.

All in all specific requirements will dictate which high level programming language is more suitable. That being said, Java is appropriate for performance critical tasks, while Python is suited for readability and quicker development.

# 2.1 Section Two Part A

JavaScript Object Notation (JSON) is selected for the data format as it represents data as objects in the form of key-value pairs and is highly suitable for data manipulation for structured data.

Advantages of JSON are its readability and ease of use alongside being supported by many programming languages which can improve incorporation into different systems. For instance, JSON is presented in a human readable format, which could be greatly beneficial if the client wishes to read the file or visually inspect the data. [Efficiency of JSON for Data Retrieval in Big Data] highlights, on a similarly formatted dataset, that JSON out performed Extensible Markup Language (XML) for retrieval queries and had a lower CPU usage. This demonstrates JSON’s flexibility and ability to handle the data types and volume of data the client has specified.

A disadvantage of JSON is the use of a limited schema, which can result in data integrity compromises as well as the lack of functionality to add comments about the data structure, which make it difficult to add metadata. Indeed, deviation from an expected format can negatively affect data and result in it not being processed correctly [JSON Tiles: Fast Analytics on Semi-Structured Data]. Significantly, this can lead to interoperability issues between different systems, or people. On the other hand, XML has a strict schema and is scalable for both structured and unstructured data. For instance [Alternatives to relational database: Comparison of NoSQL and XML] highlights how the characteristics of XML make it highly suitable for storing large amounts of unstructured data. Furthermore, tags allow comments and metadata to be supported in XML, but make read speeds slower when compared to JSON. That being said, the client’s program requirements involve a significantly smaller amount of data, compared to Alternatives to relational database: Comparison of NoSQL and XML], and importantly the data is structured. Significantly, JSON’s use of key-value pairs make it a highly versatile data structure and suitable for the client’s data.

All in all, JSON offers a lightweight format for representing structured data when compared to XML as an alternative.

2.2 Section Two Part B

Pandas dataframes were chosen to implement the client’s third requirement as they are highly versatile for manipulating data. Indeed, the Panda’s library is able to combine operations for use in functions which enable complex data manipulations to be performed [Putting Pandas in a Box]. For example, dataframes enable the relevant EIDs to be merged together into the merged\_categories dataframe, which contains the relevant information to perform the statistical analysis required by the client.

Validations are used to ensure data is present, cleaned and in an appropriate format [Appendix D]. Data cleaning is implemented by: removing duplicates, empty columns, replacing NaN cells and removing data which is not required for the client’s requirements [Appendix D]. This is supported by [Python for Data Analysis] and [Data analysis using Python] who recommend: removal of NaN for missing values, as they can give misleading results, and to filter out missing data. Importantly, this process meets the client’s requirement and provides insight into power consumption to identify energy management requirements. Implementation is achieved by filtering out the relevant categories requested from the client into a separate dataframe. Comparison operators are used to find the correct dates and the relevant calculations can be performed and results displayed to the user [Appendix D].

An alternative to using Pandas dataframe is Numpy and the ndarrays data structure. Ndarrays are similar to dataframes as they both hold tabular data and can manipulate data [Data Structures for Statistical Computing in Python]. However, a disadvantage of ndarrays is that they require homogeneous data types. Significantly, the client's data is non-homogeneous and dataframes are the superior choice compared to Ndarrays. Having said that, a potential problem with the Pandas dataframe is lack of support for keeping track of changing data and validating data quality. A tool which can augment Pandas with tabular data validation is the Pandera library. Indeed, [pandera: Statistical Data Validation of Pandas Dataframes] strongly argues that Pandera can simplify the data validation and cleaning process.

2.3 Section Two Part C

Information from the extracted multiplexers contain a mixture of numerical and categorical data, with the majority of data being categorical. With this data set a pie chart suitably displays the frequency distribution of each service label.

An alternative chart to represent the data is a grouped bar chart which allows for effective visualisation comparisons and pattern recognition [Research on Python Data Visualization Technology]. However, excessive categories can cause overcrowding and therefore reduce the readability of the chart.

The libraries selected for the implementation were: pandas and matplotlib. Pandas handles the data and performs the analysis while matplotlib provides the visualisation [Appendix E]. An alternative library instead of matplotlib is Plotly that creates high quality and interactive graphs. Although both libraries share similarities, Plotly’s advantages are in interactive visualisation for users to explore [Assessing the Performance of Python Data Visualization Libraries: A Review] whereas Matplotlib excels in creation of complicated graphs with high levels of customisation and integration into other Python libraries [Comparative Analysis of Data Visualization Libraries Matplotlib and Seaborn in Python]. Moreover, a limitation of Plotly is its reliance on JavaScript for rendering which requires users to have JavaScript enabled and use of Dash for web page integration. Therefore, Matplotlib is selected as the client requires graphs to delineate data which is grouped in an appropriate way.

A separate class, Graphs, was created which follows the Object Oriented Principles of design and allows for encapsulation. The method to create the graph is then called inside the dataProcessingGui class. Importantly, this follows SOLID design principles, specifically the single responsibility principle [Design Principles and Design Patterns]. Significantly, this allows for modularity by separating GUI and graph code while encapsulation ensures each class focuses on a specific responsibility. Within the pie\_chart function, service labels are placed into an array and a for loop is used to iterate over them and the autopct argument is used to assign a percentage to the radio stations in each service label [Appendix E].

2.4 Section Two Part D

A scatter chart, of Frequency allocation to Site, as well as individual scatter charts of each Service Label are utilised to visually illustrate any correlation between the selected attributes from the extracted DAB stations. Scatter charts were chosen because they allow for visualising any relationships between data and enable the user to initially explore the data [Appendix F]. Additionally, no further data cleaning was required as both initial csv files are cleaned and prepared prior to this stage [Appendix F]

This data is largely categorical so numerical tests for statistical significance are not appropriate. For example, determining the correlation coefficient and interpreting its value as positive, negative or no correlation is not suitable [Exact inference for categorical data: recent advances and continuing controversies]. Therefore, the Chi-Square test is a suitable test for categorical data and can be applied to this data set. An advantage to using the Chi-Square test is that it assesses independence of variables. An alternative method is Fisher’s Exact Test which can be used to achieve an exact p-value but is only efficient for small sample sizes [Pearson-Fisher Chi-Square Statistic Revisited]. The results from the Chi-Square test are outputted in the GUI’s text widget [Appendix F] and clearly indicate whether there is a strong or negative correlation. Augmenting this, the displayed information is also visually shown with the scatter charts.

In summary, Service Labels one, two and three have p-values close to 0 which indicates a strong connection with Frequency. This indicates that there is a level of significance, or correlation, between the Frequency and Service Labels. However, Service Levels four and ten are larger which presents a weak connection with the frequency column. Notably, this indicates the relationship between these variables is random or not of significance, which is supported by visualisation from the scatter charts. Moreover, similar results are found when comparing Block to each Service Label.

3.1 Section 3 Part A

Software engineers should be subject to legal regulation and an ethical framework in order to safeguard user privacy from malicious data collection and exploitation. For instance, the absence of legal regulation facilitated Cambridge Analytica to collect data from 90 million users in order to influence political elections [FIND]. This illustrates concerns for data privacy and security with no legal, or ethical framework, to protect customers. Indeed, [When data is capital: Datafication, accumulation, and extraction] argues that data collection is a vital component of the modern political economy. Consequently, businesses must find more data from customers to be commodified as a growth model which places software engineers at the forefront of ethical decisions in how their software is utilised.

Data usage has sparse legislation and is thus vulnerable to misuse and some would argue that legislation can impede innovation and technological advancement [Is Slow Economic Growth the ‘New Normal’ for Europe?]. However, a counterpoint is that legislation, such as the European Union’s Digital Services and Digital Markets Acts [DMA and DSA], will change the digital landscape by enabling smaller companies to compete and thus drive innovation rather than restricting software engineers. Significantly, legislation does not necessarily seek to restrict software engineers, or companies, but to ensure unfair conditions cannot be imposed on customers [Beyond surveillance capitalism: Privacy, regulation and big data in Europe and China]. Moreover, legislation could facilitate positive change in company culture to consider ethics throughout the design cycle.

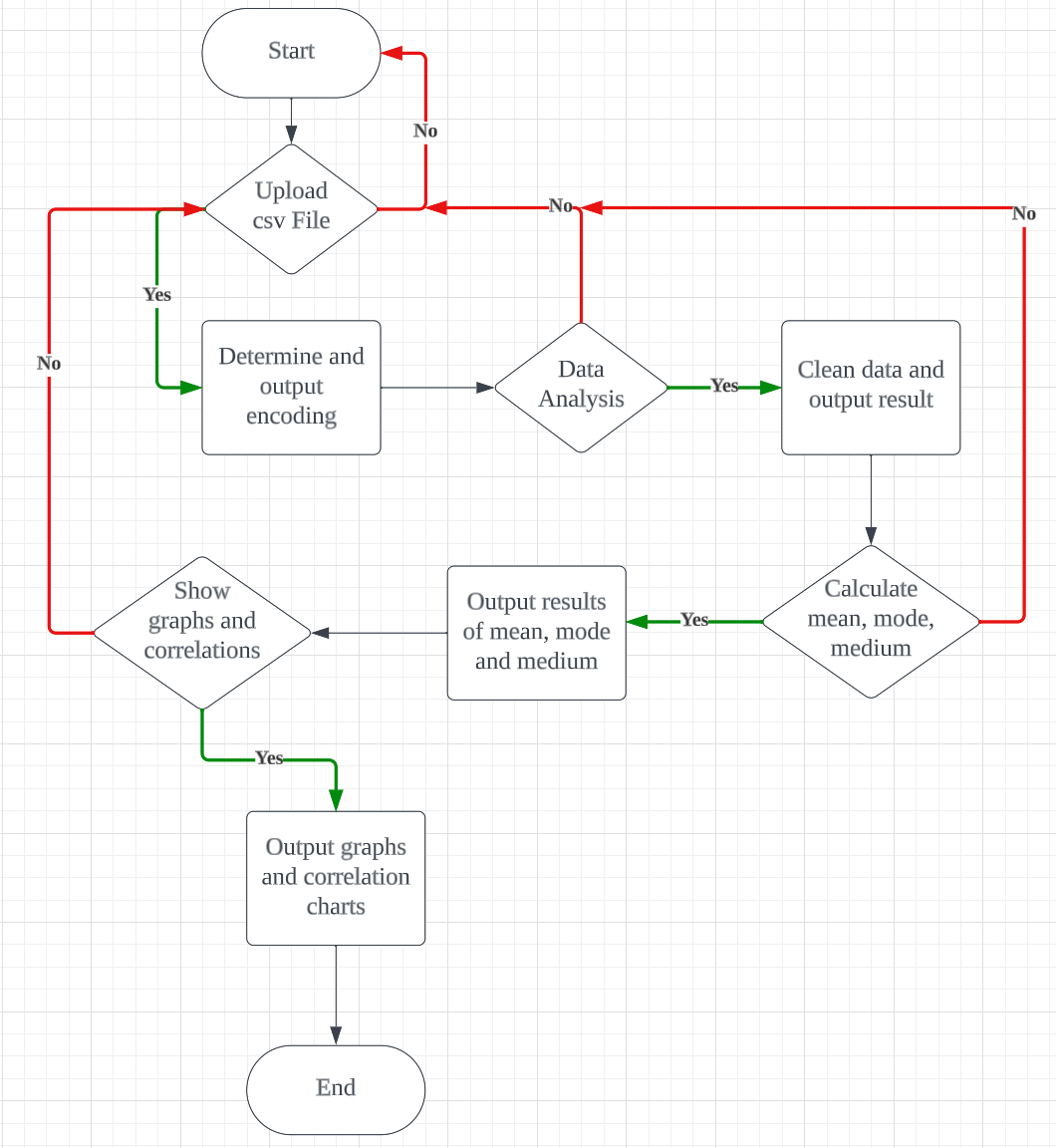
Ethics are championed in the technology industry and moral obligations have been considered similarly to the medical professionals [Ethical Standardsfor Computer Professionals: A ComparativeAnalysis of FourMajor Codes], but technology can still be misused. For example, Amazon’s tracking of warehouse workers reveals how software can be utilised in an unethical way by monitoring worker’s productivity[Machinic dispossession and augmented despotism: Digital work in an Amazon warehouse]. However, an argument can be made for a business keeping track of productivity and ensuring quality control. Indeed, recent patents submitted by Amazon suggest a desire for efficiency for the incorporation of machine and human workers to achieve optimum results[Humanly Extended Automation or the Future of Work Seen through Amazon Patents]. This implies a positive requirement for data collection in order to facilitate safety, but one that can be used unethically if not checked legally and ethically.

In summary, to negate misuse of technology ethical considerations should be to integrate into all stages of development [Towards Ethical Data-Driven Software: Filling the Gaps in Ethics Research & Practice]. Arguably, this could enable developers to have an oversight of how their software will be used and allow for an ethical questioning of how software is used. Indeed, this could be supported by accreditation from professional bodies such as the Institute for Certification of Computing Professionals [<https://www.iccp.org/code-of-ethics-conduct-practice.html>] which, alongside legislation, could foster a positive culture change.

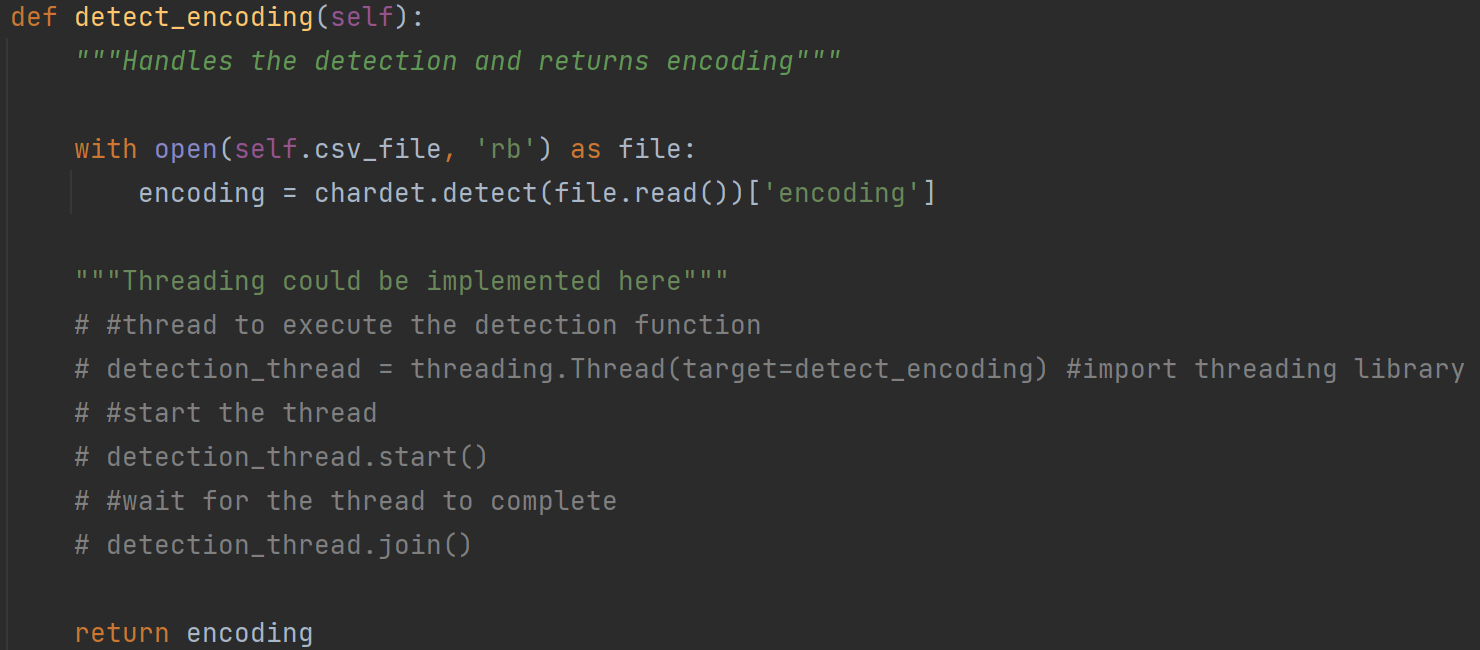
Reference List

Appendix A

Program Flow

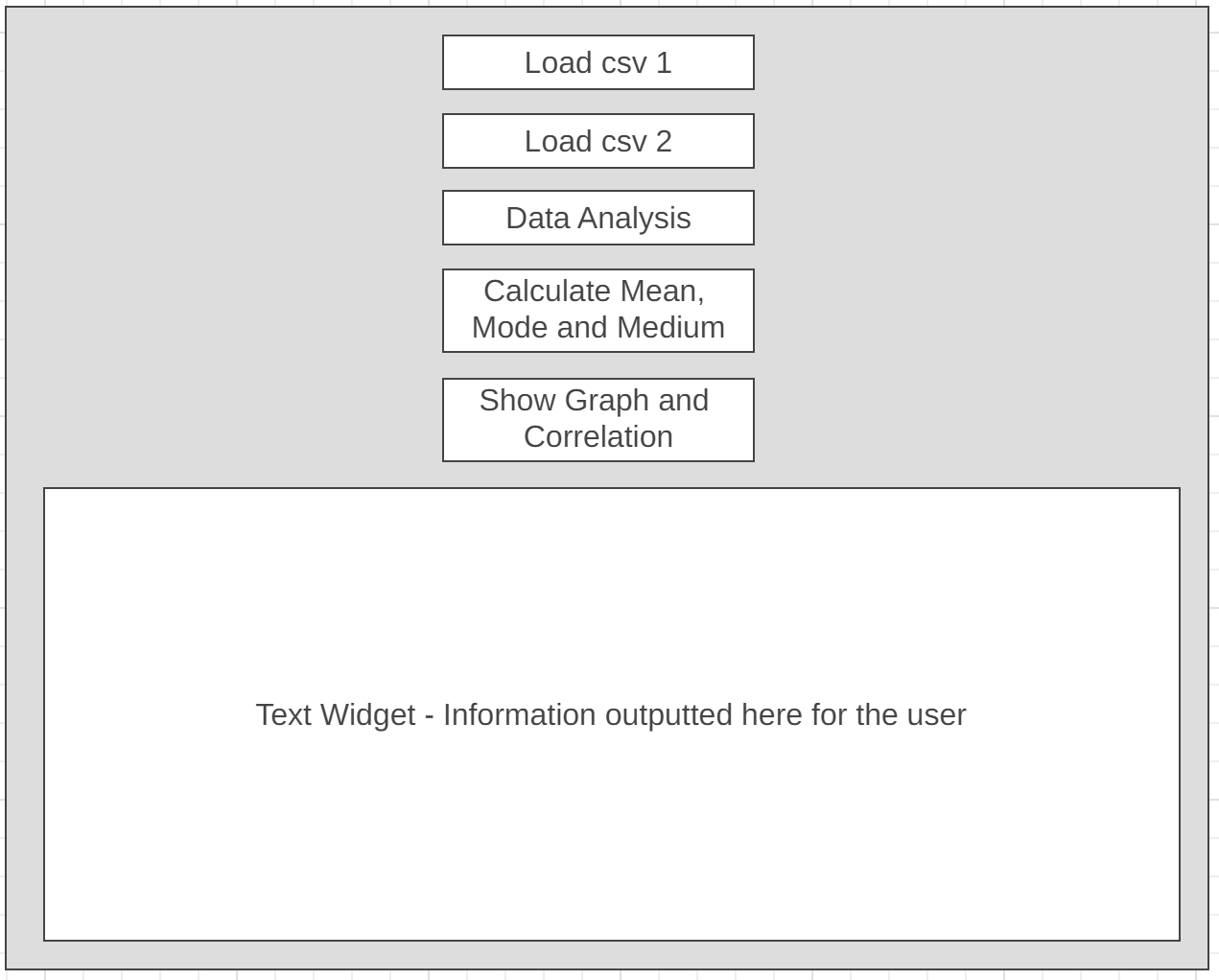


Implement Threading

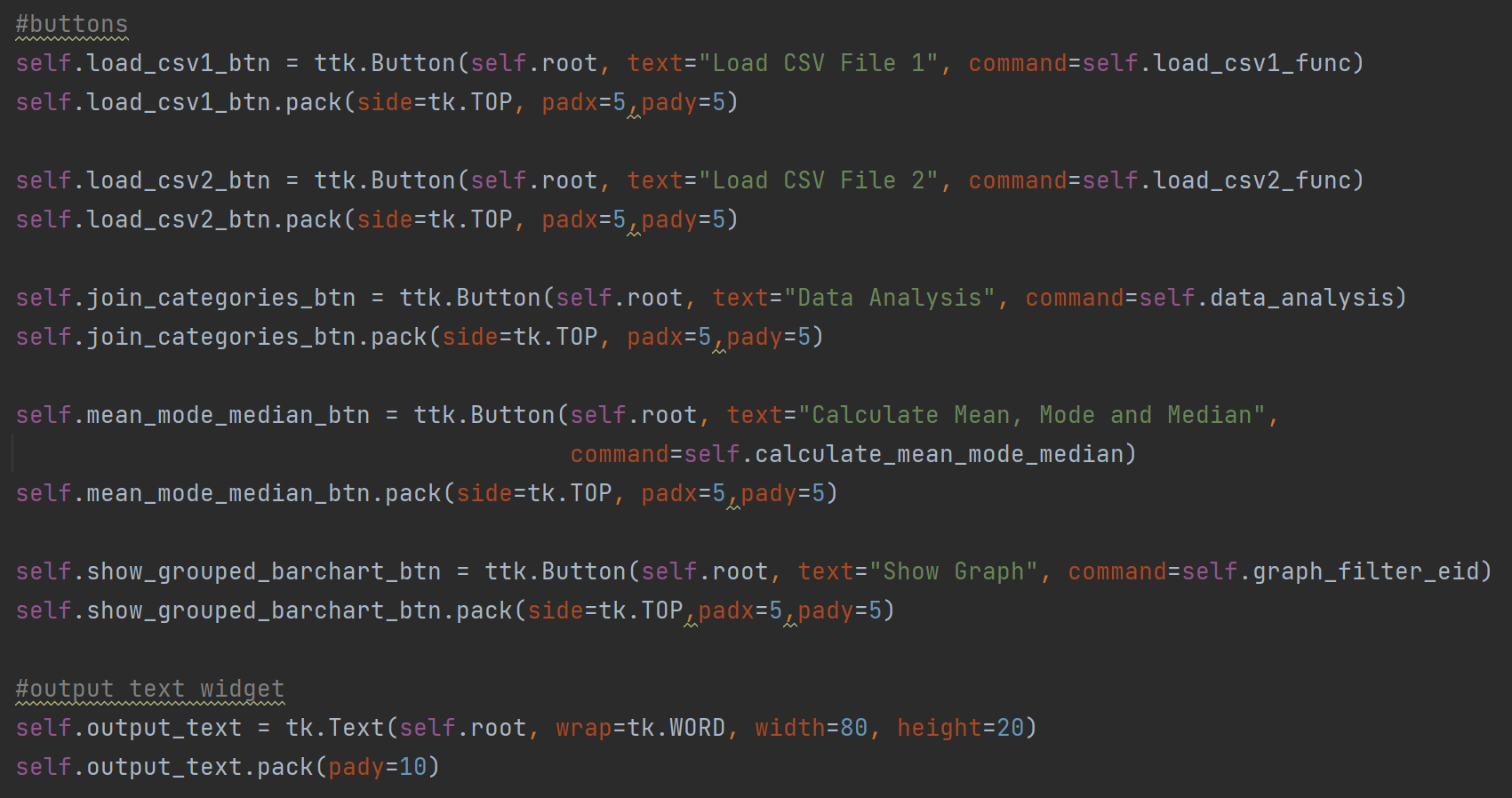


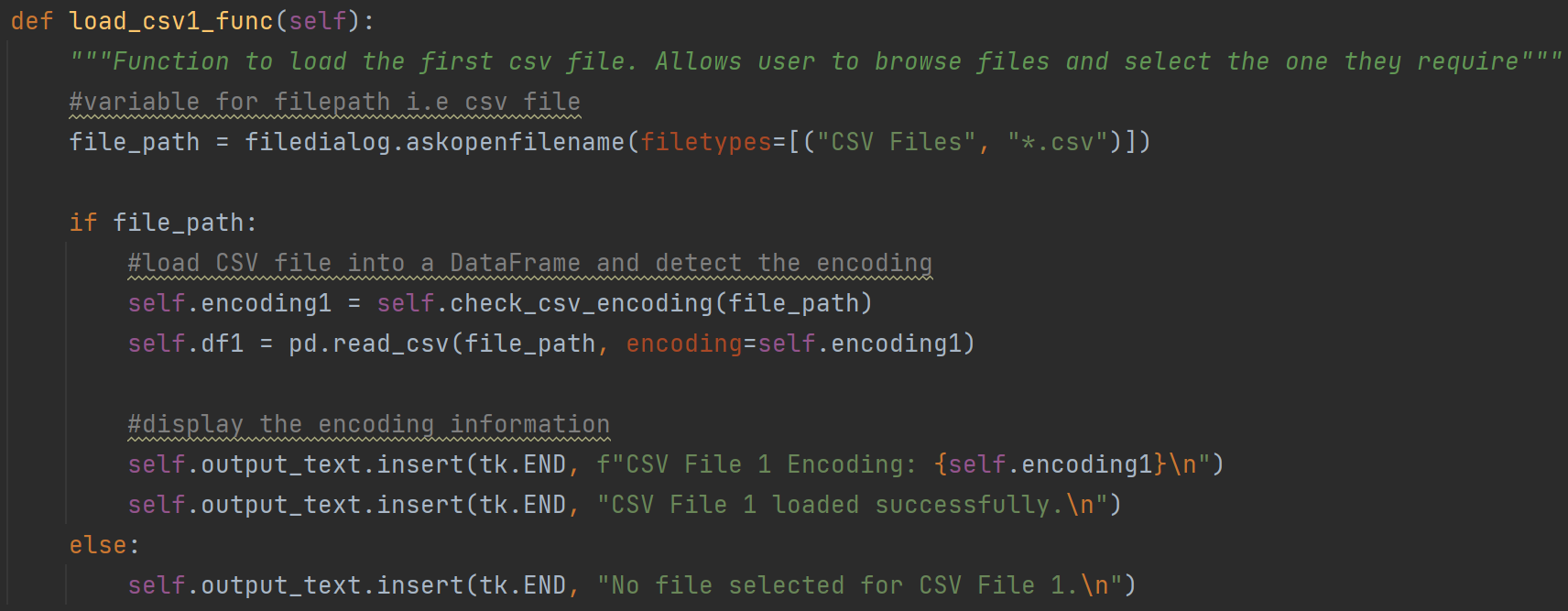
Appendix B

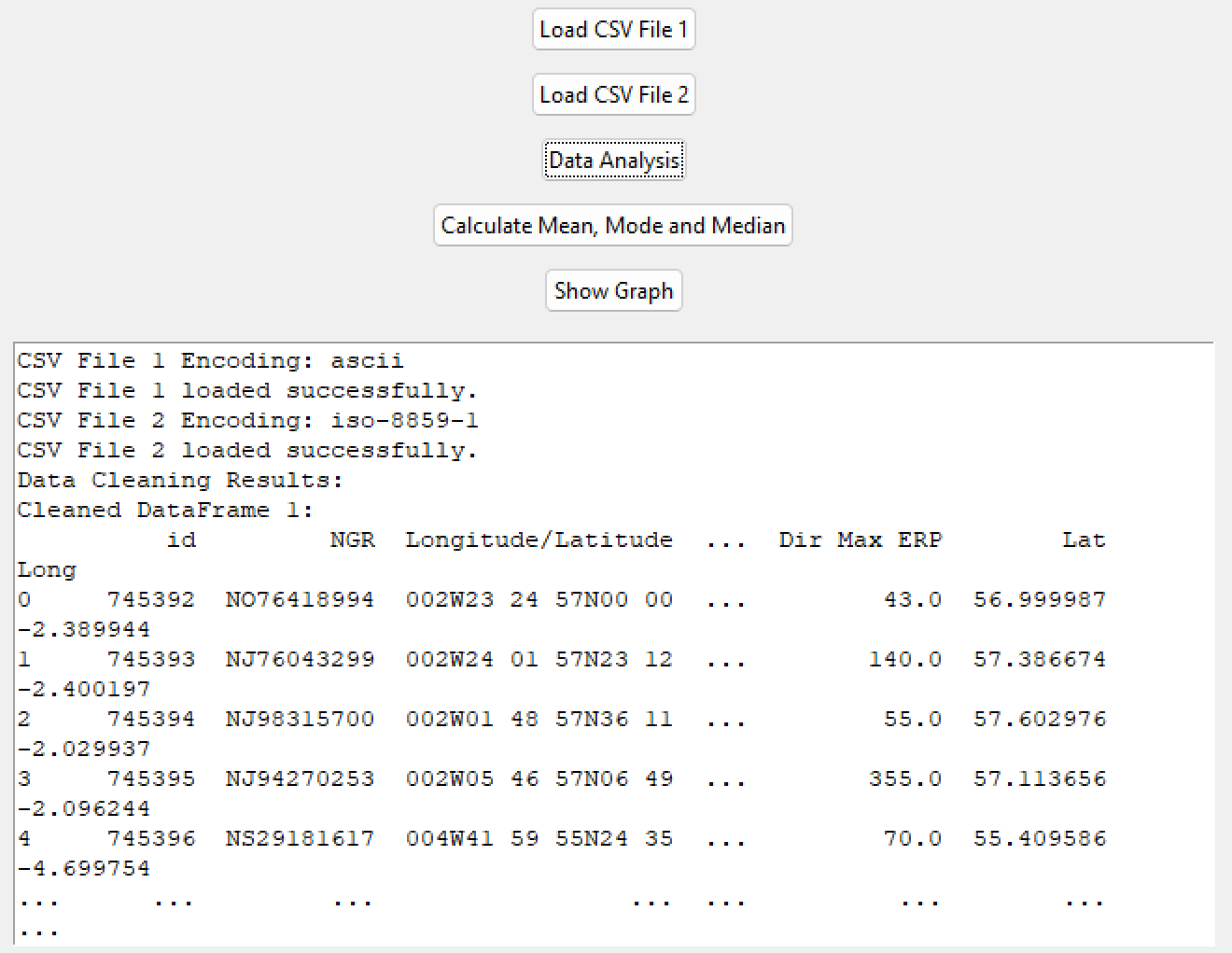
Wireframe Diagram



Buttons and Text Display - Code and GUI

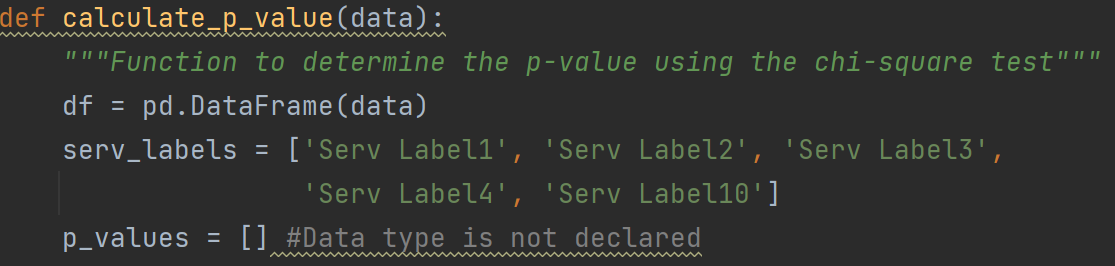






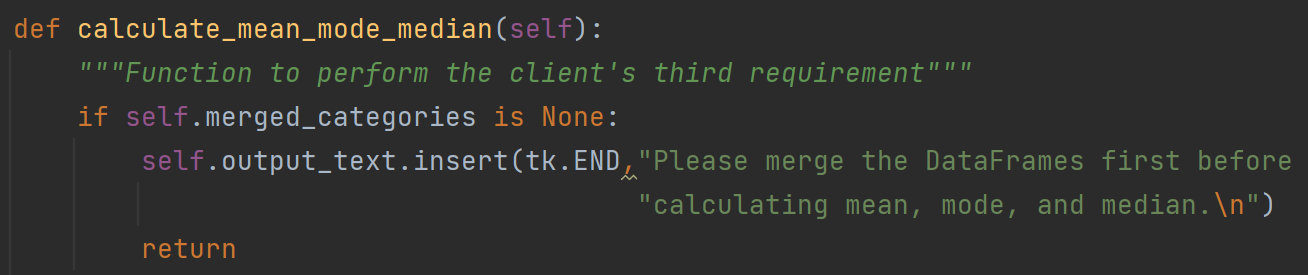
Appendix C

Data Type Declaration - Not Declared in Python

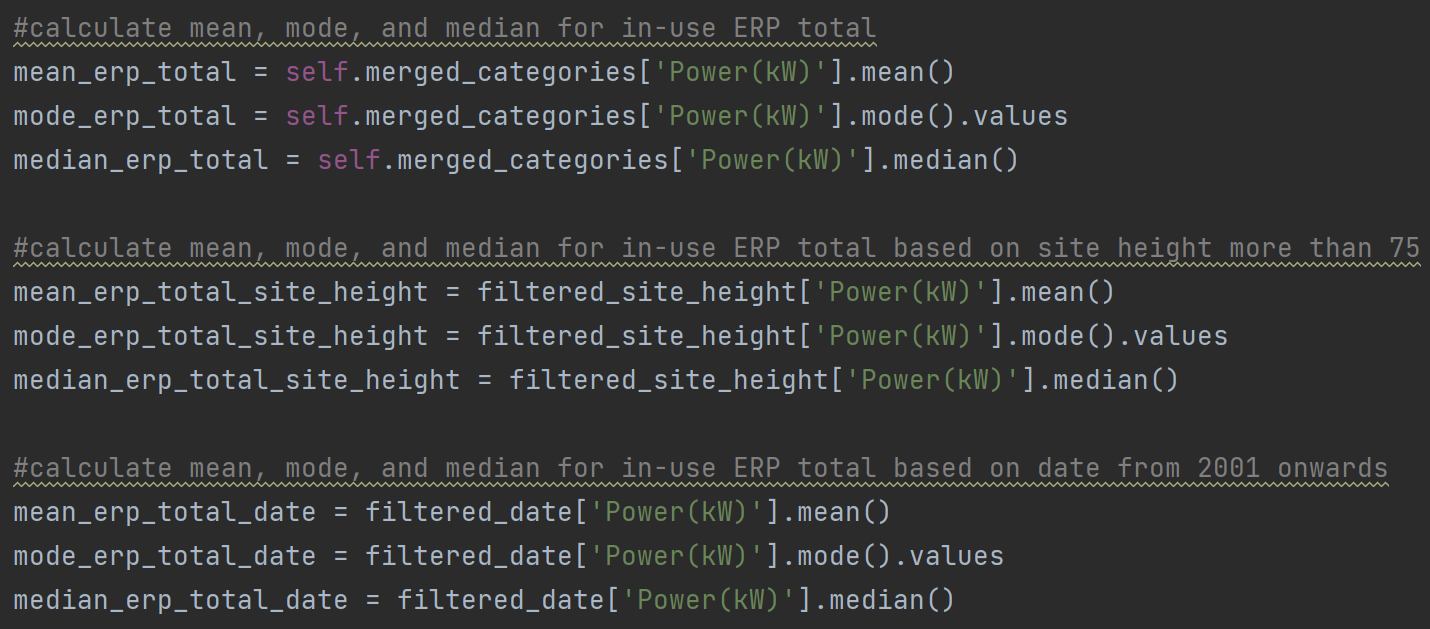


Appendix D

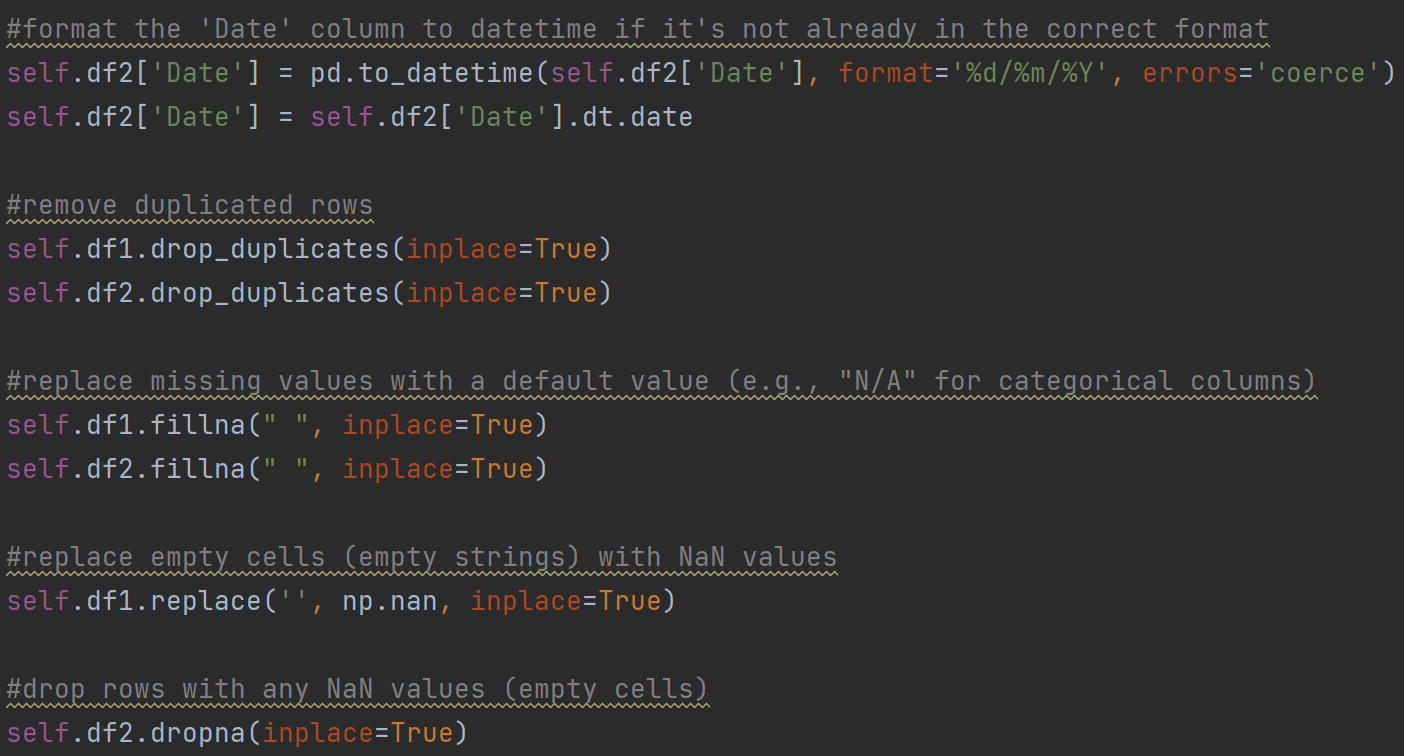
Check Correct Data is Present



Actions to Calculate Mean, Mode, Median

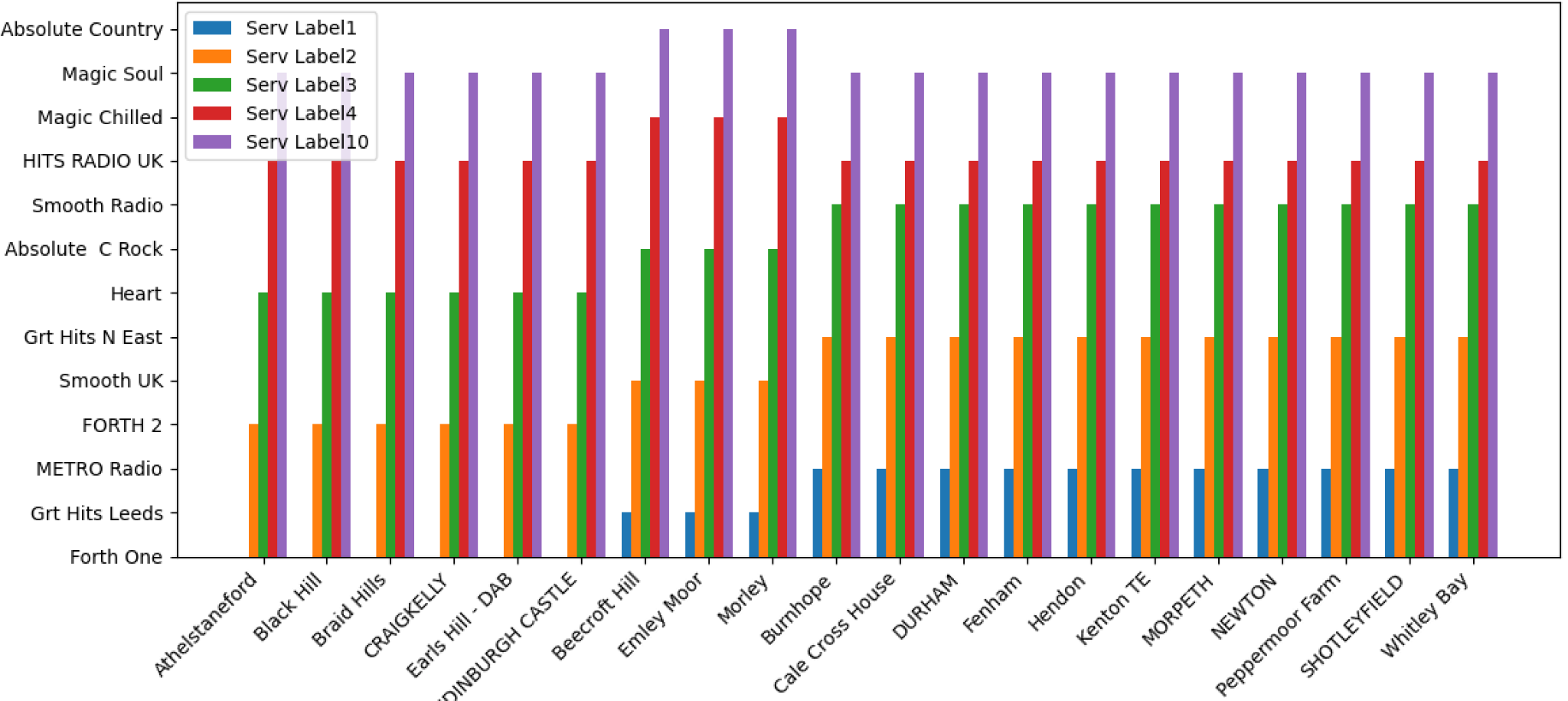


Data Cleaning

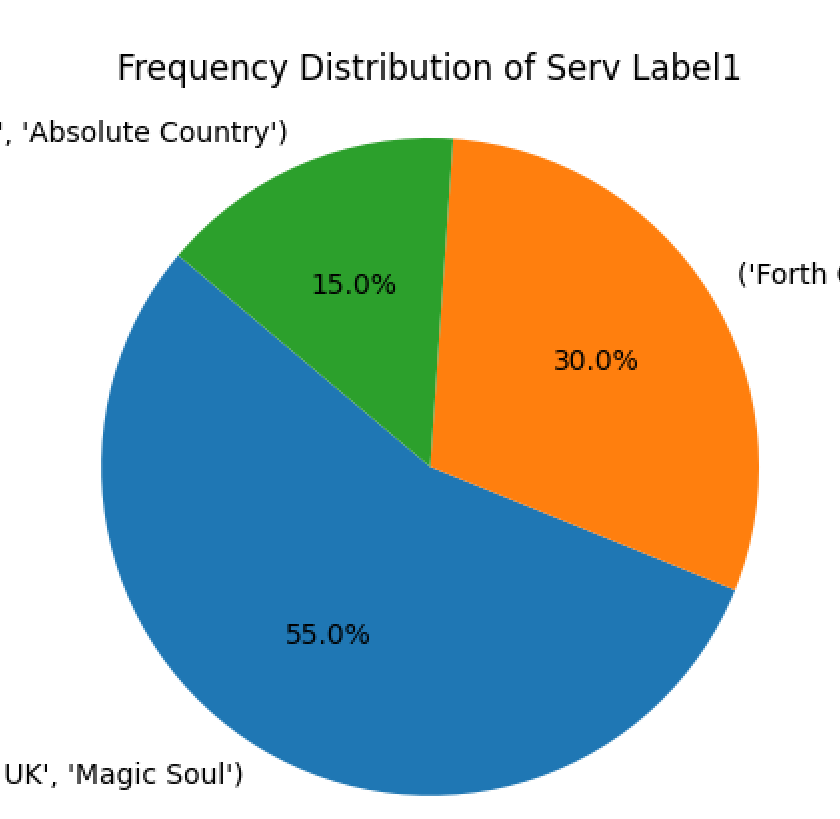


Appendix E

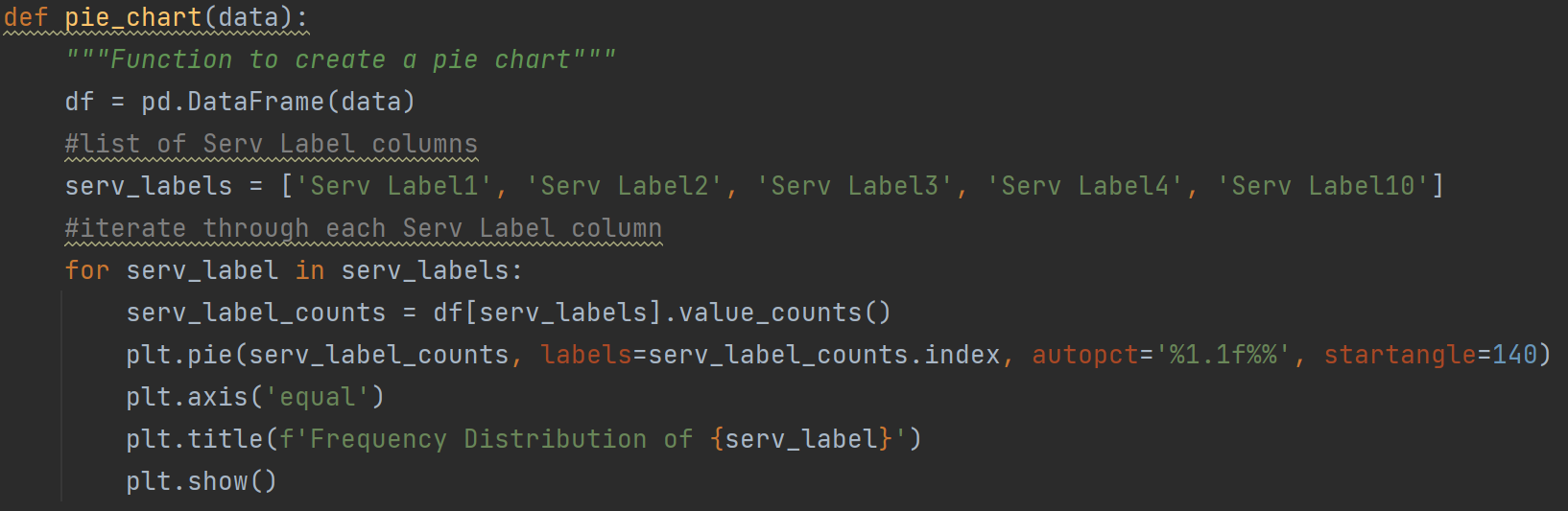
Grouped Barchart



Pie Chart

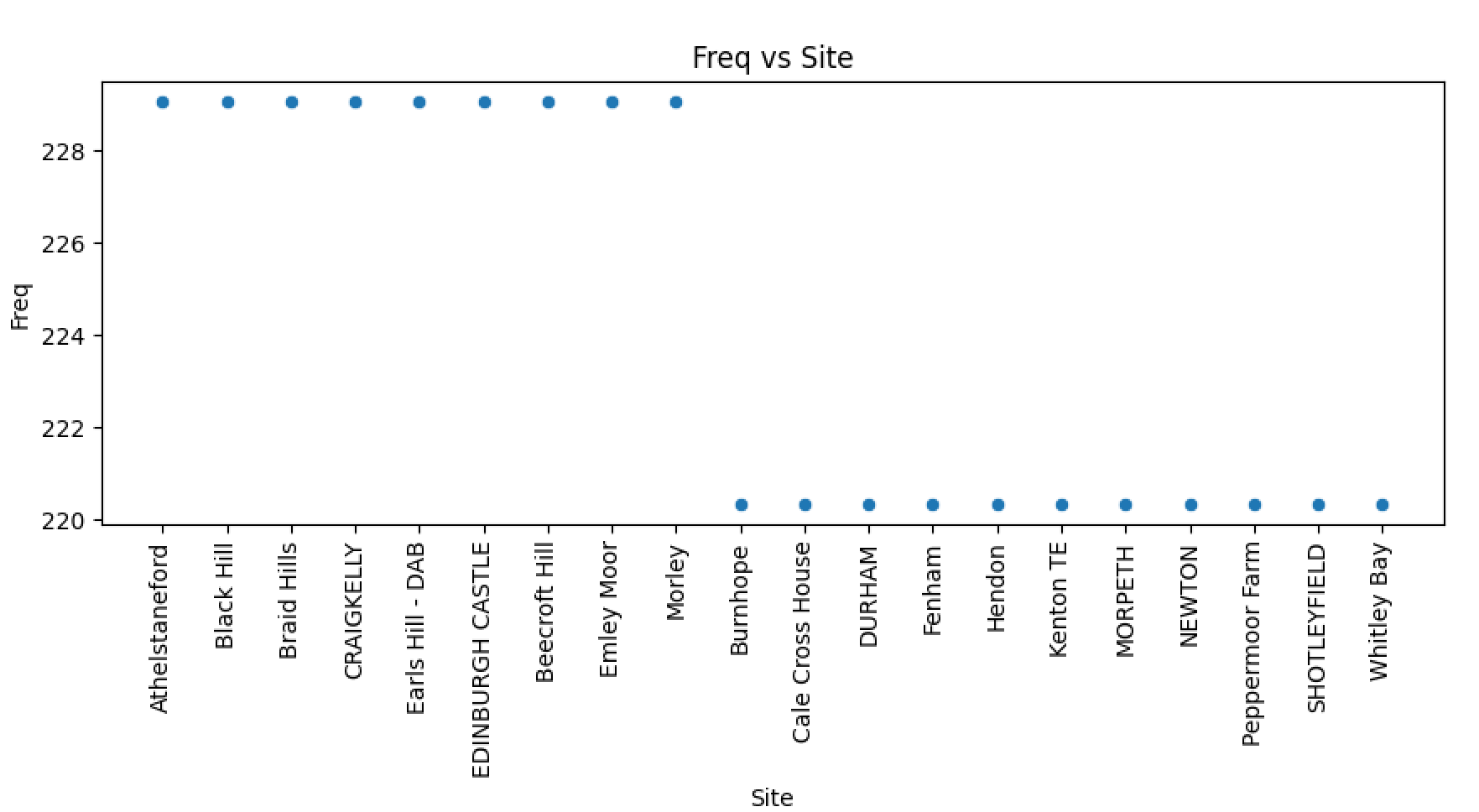


Code to show Matplotlib

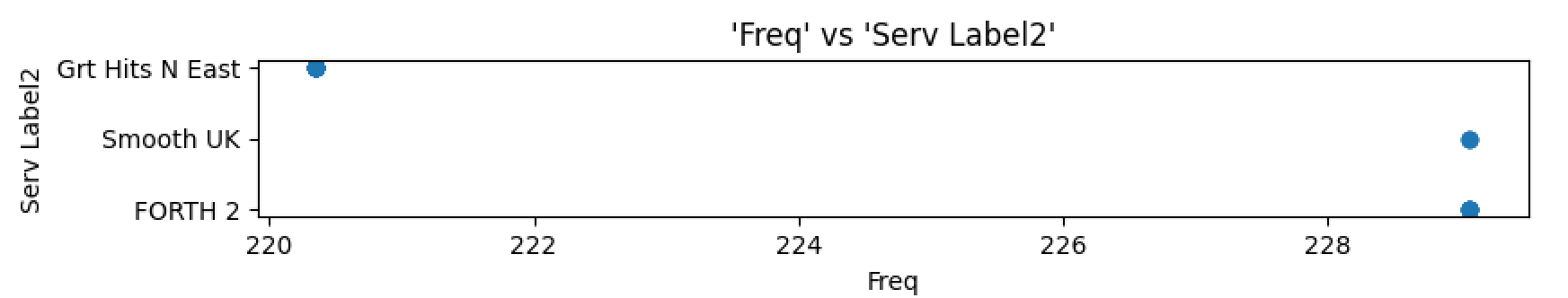


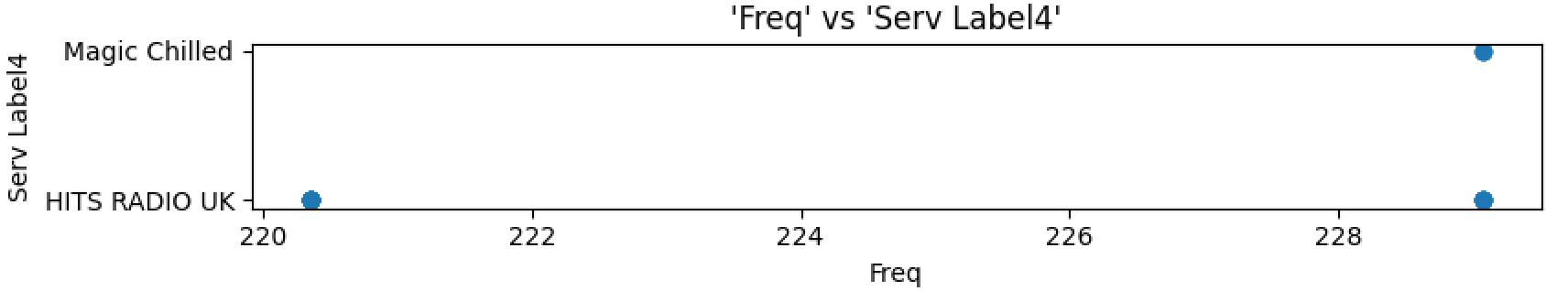
Appendix F

Scatter Chart - Frequency vs Site

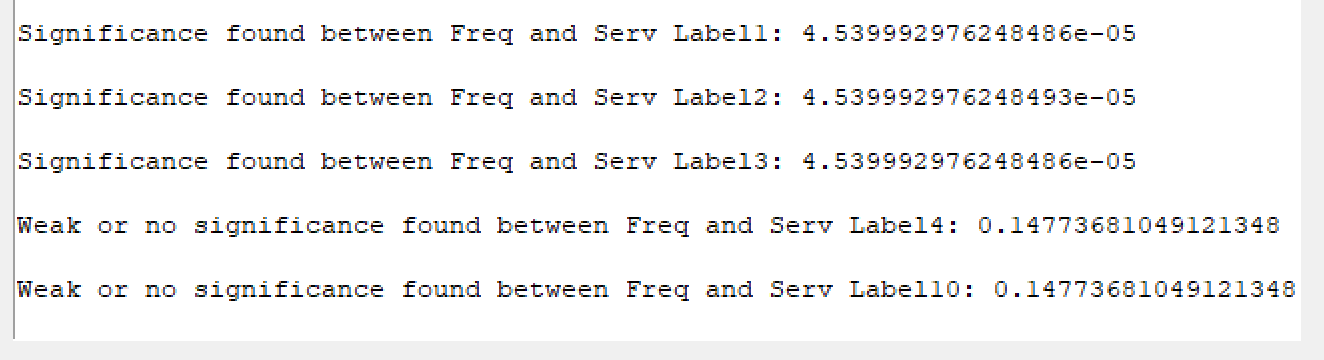


Examples of Two Scatter Charts - Frequency vs Service Label





Result of p-test Displayed to the User



Calculating the p-value

