University of York

Department of Computer Science

Report on Python program to format, clean, and analyse data

With discussions on threads, Java and a reflection on morals, ethics and the law

Word Count: (shown with each section)

20th August 2023

Abstract

This document constitutes a report on a Python program developed to format, clean, and analyse data from two csv files containing information about DAB radio stations. It also contains discussions on python threads; Java/Python data containers; and a reflection on morals, ethics and the law in computing.

Contents

1 Theory

1.1 Python Threads (497 words)

The function 'export_jsons' (appendix A) has been selected to discuss using Python threads. Currently, the function converts to DataFrames and cleans the data from TxAntennaDAB.csv. It then writes it to a file in json format. Following that, the function does the same for TxParamsDAB.csv. Subsequently, it extracts the client required data, writing this to a string and saving to file. Appendix B shows a flow diagram of this function.

Appendix C shows the proposed structure for how Python threads could be implemented. The calls to pandas 'DataFrame' function could happen concurrently, as could the calls to the internal 'data_clean' function. The final phase in the diagram shows four processes which could all happen concurrently.

Using the threading library, the callable functions can be executed in their own threads[1]. The functions 'DataFrame' and 'data_clean' both work on separate pieces of data but because threads use the same space for their data[2], the 'lock', 'acquire' and 'release' methods can be used to correctly synchronise them[3]. If the threads were allowed to run without a lock,

unreliable data would be produced because the thread scheduler can swap between threads at any point producing race conditions[3]

After 'DataFrame' and 'data_clean' have completed, the cleaned 'ant' DataFrame and cleaned 'par' DataFrame are used to write to json files and create lists ready to create 'json_string'. The methods concerned here ('to_json' and 'literal_eval') will also need to use the lock and associated methods to ensure data integrity. After all this has completed, the concurrent processes phase will end, 'json_string' can be created and the current state can be written to file.

Given that there are significantly more columns to process in the TxParamsDAB.csv file compared with the TxAntennaDAB.csv file, from a time saving perspective, it would make sense to prioritise the thread that deals with TxParamsDAB.csv. This is because the program will always be waiting for both threads to complete before moving on to creating 'jsonstring'. The operations could follow this sequence:

1. Write 'par_data' to DataFrame.
2. Clean 'par_data' DataFrame and write 'ant_data' to DataFrame.
3. Convert cleaned 'par_data' DataFrame to list, write to json file and clean 'ant_data' DataFrame.
4. Convert cleaned 'ant_data' DataFrame to list and write to json file.
5. Create 'json_string'.
6. Write current state to file.

In reality though, the Global Interpreter Lock (GIL) in Python only allows one thread to run at a time[4] and threads cannot be prioritised[5], so the above order of events could not happen.

What would actually happen is that either the write ant_data DataFrame or write par_data DataFrame process would begin and acquire the lock, setting 'block' to '1'[2]. Whichever process did not acquire the lock will wait until the 'release' method has been called by the method that has the lock. Following that, the thread scheduler will swap between threads as each thread will be at a different stage (e.g. cleaning the ant_data DataFrame and writing the par_data DataFrame). Each method will again need to acquire and release locks to ensure race conditions do not occur.

1.2 Graphical user interface (GUI) interactions **(498 words)**

The wireframe for the GUIs can be seen in appendix F. The design meets the client's requirements by having buttons on the right-hand side in the main window to switch between different visual representations and perform operations. The visual representations themselves are displayed in the middle section of this GUI.

In a change to the original design, the left-hand section only displays averages when the 'All multiplex information' button is pressed. For correlation diagrams, the averages are replaced by a key.

The state diagram in appendix E shows the displays that the user will see and behind the scenes operations of the program. It details where in the program these client requirements

take place:

- Means to load initial data sets and translate them into JSON.
- Means to back up the data and preserve the current state (the program also preserves the last view upon closing).
- Process for cleaning and preparing the initial data.
- GUI for interacting with the data.

The facility to modify the data has also been implemented. The wireframe for this can be seen in appendix F. The user will be prompted for the id number for the entry they wish to edit. The client variables are editable for ALL ids.

Examples of code for the GUI can be found in Appendix D. The first example demonstrates the implementation of three control buttons. The grid method of placing objects has been used with the buttons placed on the right-hand side by using column 2 (column counting starts at zero). Screen width is used to calculate the horizontal 'padding' required to space the columns correctly.

The second example demonstrates how the three averages are placed - again, in a grid. A list is used to store each label in order that the whole column can be cleared iteratively in preparation for placing a key for a heatmap.

The third excerpt shows the creation of heatmaps using seaborn. The data is gathered for two supplied multiplexes using the function 'get_data_for_correlation' which modifies the global value for the DataFrame 'mult1_vs_mult2_df' which is then passed into 'sns.heatmap'.

In terms of best practice for a GUI, there are numerous resources available. Some examples are: usability.gov which outlines some key design basics[6]; Jerry Cao writes about 'Gestalt Principles for Designers' and discusses invariance (similar looking objects recognisable as being the same) and similarity[7]; Dmitry Fadeyev writes about the use of colour[8] and; Suzanne Martin describes the organisation of interfaces[9].

The GUI designed for the client adheres to these ideas as follows:

- Input controls - buttons provided for quick access to menu items. Their similar shape helps our brains recognise them[7]
- Message boxes - messages appear to keep user informed[6]
- Simplicity - window maintains its layout to keep it consistent (an important part of organising an interface)[9]
- Strategic use of red in headings to get noticed[8]

Some bugs exist which could be addressed in the future. For example, checking entries by the user and errors when some windows are closed with the cross.

1.3 Java Versus Python **(400 words)**

When discussing the effectiveness of manipulation of data containers in Java and Python, the factors to consider are time spent in development of the code and time for the program to execute it[10]. Java has been reported '...to be 25 times faster than Python.'[11] It takes less time to run the code than Python because Python reads the code line by line[12], but Java

is compiled[13]. Conversely, the actual Java code itself will always be longer than Python code[10].

This is exemplified by the Python code presented in Appendix G. This code was written with a Java mentality before NumPy and Pandas had been studied on the course. As a result, it is many more lines longer than would be required in Python. This also affects the rest of the program and data manipulation is more cumbersome than is needed. The processes carried out by this function could be added to the 'data_clean' function in appendix J (discussed in section 2.2). For example, rows with the NGR fields not required could be found in the DataFrame using '.index'[14]. These could then be removed using '.drop'[15]. The headers 'In-Use Ae Ht' and 'In-Use ERP Total' could be changed by using '.rename'[16]. Performing these operations as part of the cleaning process would significantly reduce the number of lines of code throughout the program.

As well as being faster because it is compiled, Java can be quicker with its use of threads because it does not use a Global Interpreter Lock[17]. In Java, the implementation discussed in section 1.1 could run significantly faster because Java can use multiple cores or processors to perform operations concurrently whereas Python will only use one[18]. This means that the processes described in section 1.1 could be carried out concurrently.

Data types also play a part in determining the effectiveness of manipulation of data containers. Unlike Java, Python variables are dynamically typed[19] which means type checking happens at runtime[20]. Whilst this is another contributor to the speed of execution, the time taken by a developer in debugging coding errors will be longer because they are more likely to occur[20]. Complexity of Java development is compounded when we consider that a python list can hold objects of different types[21] whereas the Java equivalent (ArrayList) can only hold objects

of the same type[10]. Ultimately though developer time costs more than processor time[22], so organisations are likely to prefer Python to Java.

2 Design

2.1 Data format (298 words)

The brief stipulates the data required for analysis. The vast majority of data in the csv files is not used. Therefore, all data will be cleaned and stored in json files with a 'current state' file created for storing and accessing data required by the client. This json file is structured to hold data for each multiplex separately. The data for determining correlation is also separate. In hindsight, this was unnecessary; the correlation data could be merged together with the main data for each multiplex. Figure 2.1.1 shows an excerpt from the structure.

A file storage structure has been chosen over a database structure because of the small amount of data needed by the client. Whilst using file storage introduces redundancy[23], the amount of data (and hence the impact of redundancy) will be small. A database is also more complex and more expensive to use[24]. The options for file choice were xml and json. json is faster and easier to access than xml[25], which is one of the reasons json was chosen. json can be less secure[26]. However, security has not been listed as a requirement by the client, so this should not be a problem.

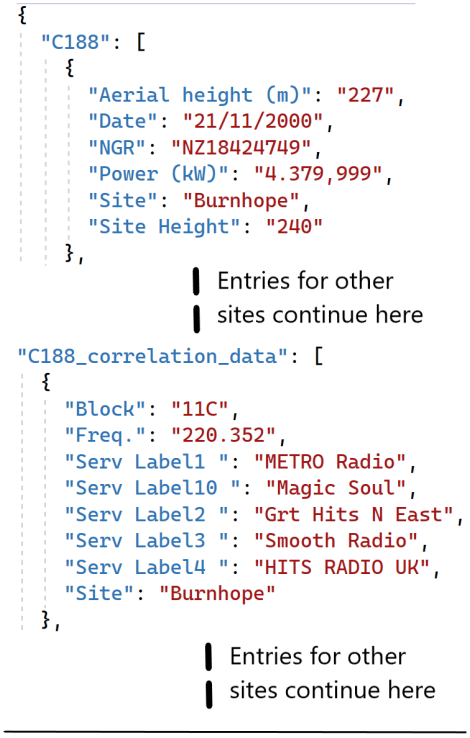


Figure 2.1.1 - Extract fron json file

Appendix H shows two code samples. The first is a function that cleans the data, creates DataFrames used for manipulating data in the program (cleaned_ant_data and cleaned_par_data) and writes json files.

The second sample is a function that stores data in 'json_string'. This variable is used to write the data to file and is converted to a dictionary of values using 'json.loads(json_string)' in order to access values held within its structure.

Appendix I contains a flowchart showing how data flows through the program. The two DataFrames and 'json_string' described previously are created. Following changes, and upon exiting, the information is written to file.

2.2 Client's 3rd requirement (294 words)

Appendix J shows samples of code used to create the averages. These are described here:

1. An initial data clean is performed when the data is imported (or modified by the user). It replaces blank values with '-1'. Filling in these gaps will avoid possibly discarding other valid data[22]. A value of '-1' was chosen because values in the data are either categorical or positive hence '-1' can easily be interpreted as an error. Values are also truncated to ensure they fit on graphs.

2. As required, the phrases 'Aerial Height' and 'Power' are introduced. They are inserted into the string 'json_string' (described in section 2.1).

3. Each entry is examined via the use of 'json.loads(json_string)'. If the Site Height is over 75m, the power value is added to a list. Before this happens, the comma that erroneously appears in the data is stripped out and the string is converted to a float.

4. The power values for dates 2001 onwards are gathered. Dates are reformatted and comparisons made using the datetime library. Power values are handled in the same way as Site Height.

5. A string of data containing labels and values is assembled for mean, median and mode to two decimal places. This is used to get the averages for both Site Height and Date.

The NumPy library was chosen to perform the calculations for mean and median because it is better than the standard statistics library functions for large datasets[27]. pandas was also considered, but its strength is in working with tables[22] and the data here is a list which is

easily translated into a NumPy array. However, NumPy does not have a mode function[28], so the standard statistics library was used for this.

Appendix K shows the section of the screen where the averages are displayed.

2.3 Client's 4th requirement (299 words)

Upon examining the data, it was determined that each mulitplex's block and frequency does not change. The variables therefore are 'site' and the various service labels. A suitable chart needed to be selected with these two variables as the axes. Whilst a scatter graph is normally associated with correlation and numerical data[29], the ability to plot multiple instances of the same variable was an essential requirement. Hence, a scatter graph was deemed the most suitable. Bubble charts and grouped, bar charts were also considered, but both of these would lead the user to interpret that some data was more significant[30] which is not the case here.

Appendix L contains 3 code samples for the implementation of this chart. The first uses the matplotlib API to create a scatter graph. The matplotlib API was selected rather than seaborn or pandas plotting functions due to its customizable nature[22]. Also, with the data being stored as a string in 'json_string' it is more suited to matplotlib than seaborn which is better for working with pandas DataFrames[31] Data for the chart is gathered by calls to 'get_site_data' and 'get_station_data'.

Samples 2 and 3 from appendix L are the functions 'get_site_data' and 'get_station_data'. These functions return a list of strings for sites and stations respectively. The 'get_site_data'

function returns 5 lots of each site to match up with the five service labels required by the client that are associated with each site. Both functions check for any '-1' values created during the cleaning process described in section 2.2 for blank entries. If a -1 is found, a missing label message is appended instead of '-1'. Labels have already been truncated by the general cleaning process described in section 2.2

Appendix M shows a screenshot of the program displaying the chart created for this requirement.

2.4 Correlation analysis (298 words)

It is interesting to note the visual overlap demonstrated in the diagram for the client's 4th requirement as described in section 2.3 and seen in appendix M. It is clear that there is no overlap in service labels for multiplexes that share the same block and frequency, but an overlap is present when the frequency and block is different. It was this visualisation that led to the idea of producing heatmaps for the service labels to identify correlation. A heatmap helps viewers to identify what is most significant[32].

Correlation is 'a mutual relationship or connection between two or more things'[33]. The connection being analysed here is in the words of the service labels for different multiplexes. Since all service labels are the same at each site, only the first site is considered. The code in appendix N shows the implementation and a screenshot can be found in appendix O.

Correlation between the service labels is determined using the ratio function from 'SequenceMatcher' in the difflib library. Code excerpt 3 in appendix O shows how it is used.

To explain, a list of ratios is created comparing a service label from the first multiplex to each of the five multiplexes from the second. The data cleaning that has already taken place (described in section 2.2) is not likely to affect the result. Two '-1' entries would produce perfect correlation, but the user would see the 'MISSING LABEL' identifier in the ledger. Truncated entries are still likely to produce a match if one exists.

Tables 2.4.1 to 2.4.3 summarise where a significant correlation has occurred.

Service Label for C18A	Service Label for C18F	Correlation score
SL10: Magic Soul	SL4: Magic Chilled	0.61

Table 2.4.1 Showing where significant correlation has occurred between service labels for multiplexes C18A and C18F

Service Label for C18A	Service Label for C188	Correlation score
SL4: Hits Radio UK	SL4: Hits Radio UK	1
SL10: Magic Soul	SL10: Magic Soul	1

Table 2.4.2 Showing where significant correlation has occurred between service labels for multiplexes C18A and C188

Service Label for C18F	Service Label for C188	Correlation score
SL1: Grt Hits Leeds	SL2: Grt Hits N East	0.69
SL2: Smooth UK	SL3: Smooth Radio	0.67
SL4: Magic Chilled	SL10: Magic Soul	0.61

Table 2.4.3 Showing where significant correlation has occurred between service labels for multiplexes C18F and C188

In general, it has been noted that scores of above 0.6 have a matching word which might indicate a similar style of service. A perfect correlation score (1) means that the service labels are identical.

3 Reflection on ethical, moral and legal aspects of computing (400 words)

Ethics and morals are both concerned with 'right' and 'wrong'. Ethics deals with rules associated with a specific group of human actions - possibly for a particular group of people[34]. Morals are an individual's own personal guiding beliefs[34]. Laws may be based on ethical considerations[35], but unlike the law, ethics has no legal weight[36].

Organisations frequently encourage creative methods to avoid bureaucracy and detrimental impacts on innovation without considering ethics[37]. Morals and ethics do not always match in some professions, so why should they in computing? For example, lawyers are ethically obliged to defend murderers even if they know they are guilty[34][38].

Yet, computing professionals must work within the law and not create a set of ethics that go against it. But, what happens when the law comes into force after the ethical framework was created? The first legislation in the world to protect an individual's safety and privacy from AI came into force in June 2023[39]. What happens to AI in light of this?

Boine provides insight into the ethics involved in AI/human relationship and the impact of laws in the European Union (EU)[40]. She raises questions about the ethics of actions of AI companions Replika and Anima, and the potential benefits/harms these actions could cause.

Is there a possibility the development of AI in the EU may be impeded? Boine concludes that what AI systems do ethically and legally should be democratically debated[40].

Organisations have flirted with the boundaries of what is morally and ethically right: Enron knowingly allowed its public financial position to be misrepresented[41]; VW cheated emissions tests[42]. Prachai Patel concludes that ethics should not be based on compliance and reports that they should be taught alongside engineering[43].

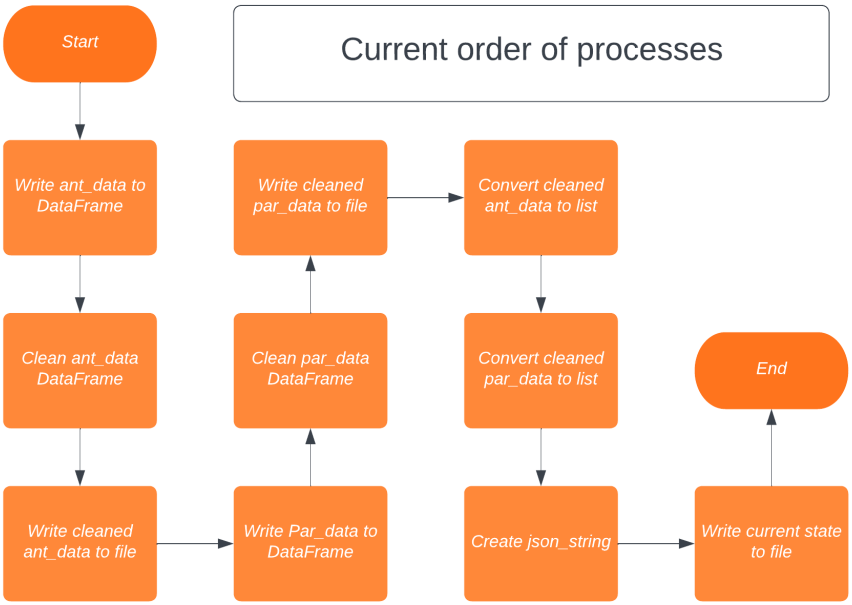
And what of morals? An individual must act within the law and within a set of prescribed ethics for an industry. If the individual disagrees with either on moral grounds, what can they do? Here we can turn to history for our answer and look at the examples of whistleblowers such as Sherron Watkins at Enron[44] or Oskar Schindler who risked his life to save over 1000 Jews from Nazi concentration camps[45].

In the end, ethical frameworks are essential in preventing harm. However, they should be openly debated and should evolve with time. Ultimately, human progress can be viewed as a 'law of nature' but it will be history that judges it as right or wrong[46].

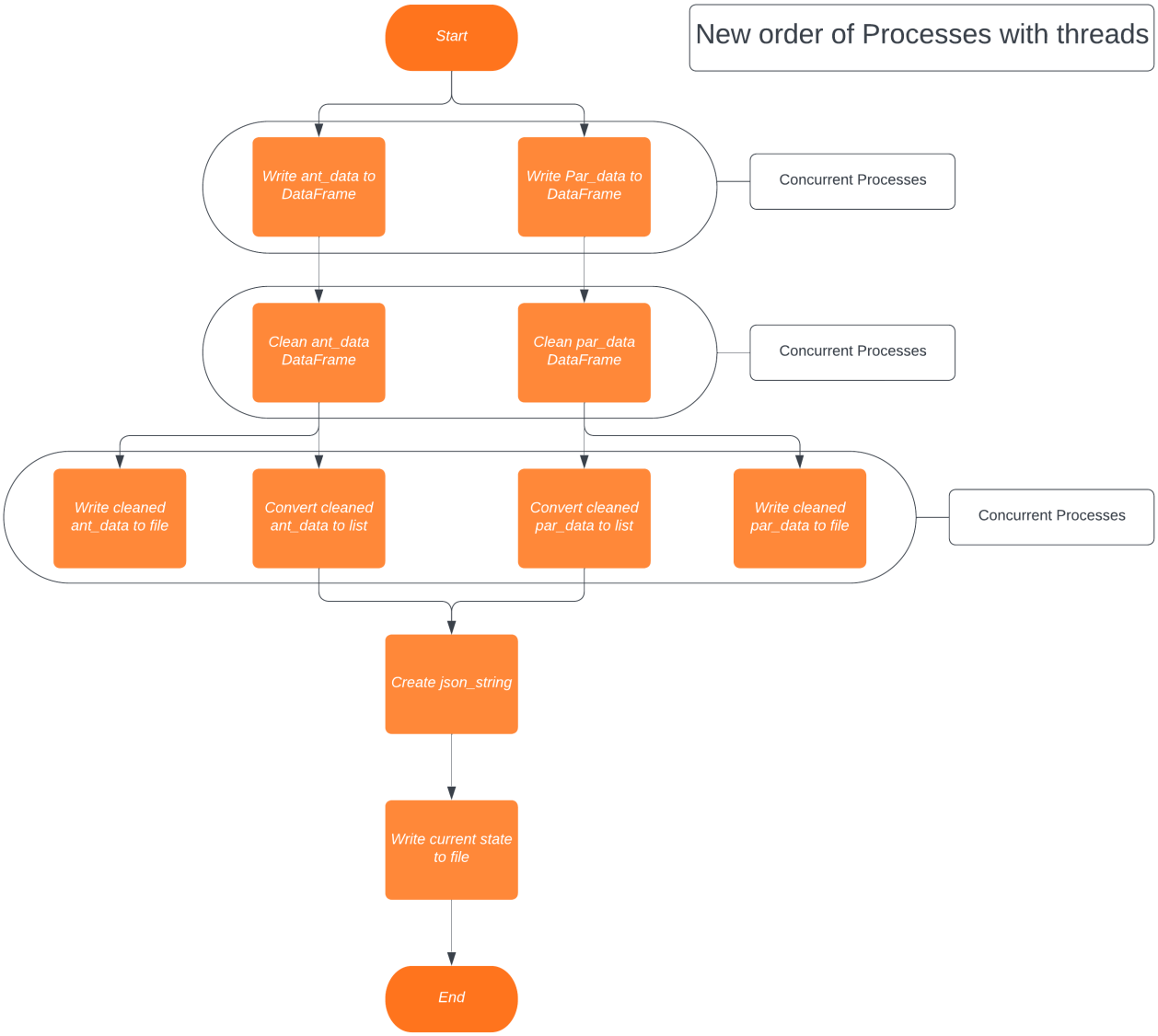
Appendix A - Code for modifying with threads

```
# Exports all data and current state to json format files and creates variables for the data
def export_jsons(ant_data_in, par_data_in):

    pd.set_option("display.max_rows", None, "display.max_columns", None)

    # Write the ant_data to a DataFrame and to file
    ant_df = pd.DataFrame(ant_data_in)
    config.cleaned_ant_data = data_clean(ant_df)
    print("Writing to file TxAntennaDAB.json...", end="")
    config.cleaned_ant_data.to_json(path_or_buf="TxAntennaDAB.json", orient="records")
    print("...done")

    # Write the par_data to a DataFrame and to file
    par_df = pd.DataFrame(par_data_in)
    config.cleaned_par_data = data_clean(par_df)
    print("Writing to file TxParamsDAB.json...", end="")
    config.cleaned_par_data.to_json(path_or_buf="TxParamsDAB.json", orient="records")
    print("...done")

    # Convert the cleaned par and ant data back to lists and store as current state
    print("\nGrouping data as per client requirements and writing to CurrentState.json")
    ant_list = ast.literal_eval(config.cleaned_ant_data.to_json(orient="records"))
    par_list = ast.literal_eval(config.cleaned_par_data.to_json(orient="records"))
    json_handling.import_data(ant_list, par_list)
```

Appendix B - Current Order of Processes



Appendix C - New order with Threads

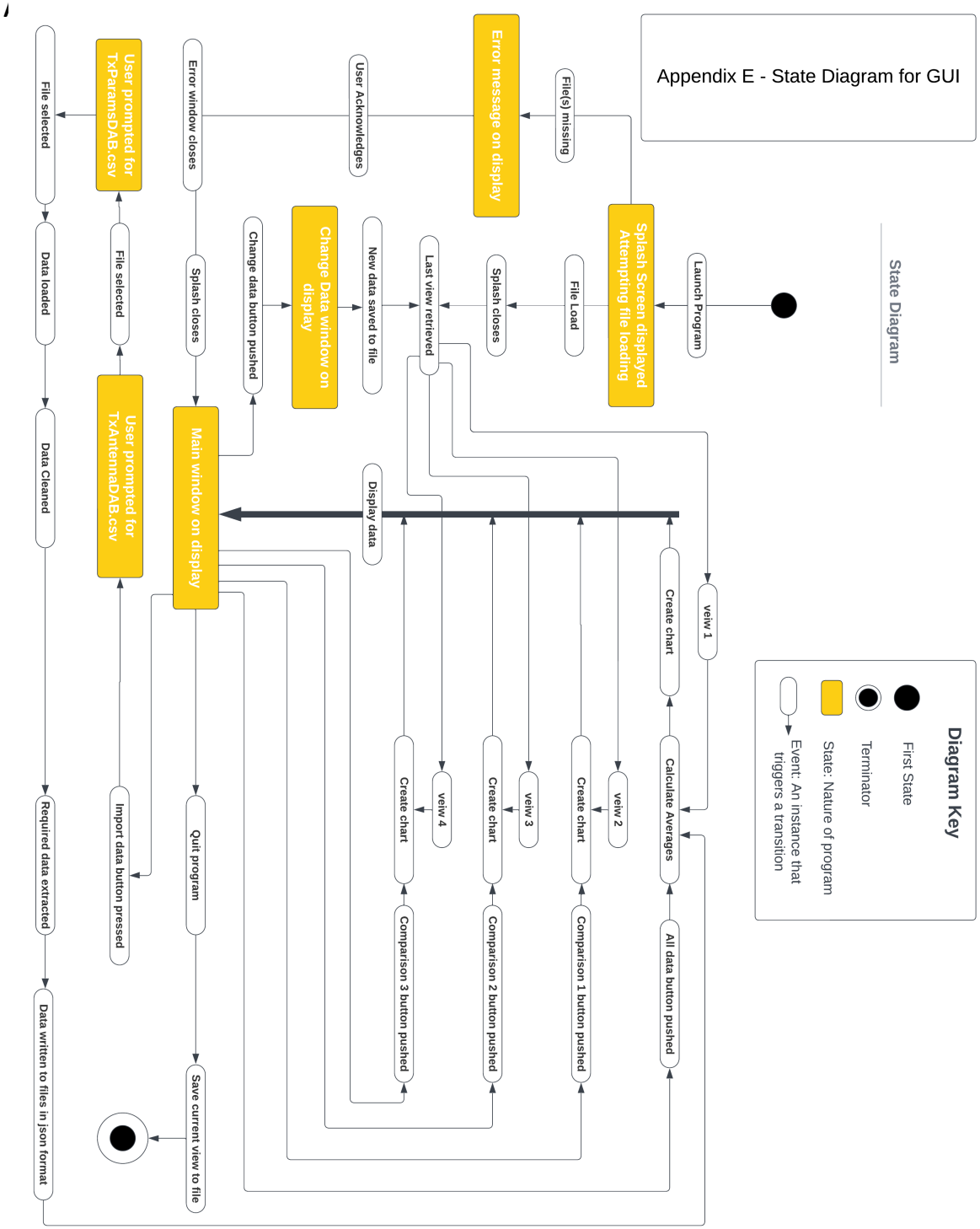


Appendix D - Samples of Code for GUI

1.
```
# Button creation
buttons_header = tk.Label(self, text="Control buttons",
                          font=('Helvetica', 10), fg="red", justify='left')
buttons_header.grid(row=1, column=2, padx=screen_width / 40)

# Import csvs button
import_button = tk.Button(self, text="Import Data from csv", command=import_csvs)
import_button.grid(row=2, column=2, padx=screen_width / 100)

# Display multiplex info button
mult_button = tk.Button(self, text='All multiplex information', command=scatter)
mult_button.grid(row=3, column=2, padx=screen_width / 100)

# Create correlation button 1
c_button_1_text = config.desired_multiplexes[0] + ' correlation with ' + config.desired_multiplexes[1]
change_button_1 = tk.Button(self, text=c_button_1_text, command=heat_1)
change_button_1.grid(row=4, column=2, padx=screen_width / 100)
```

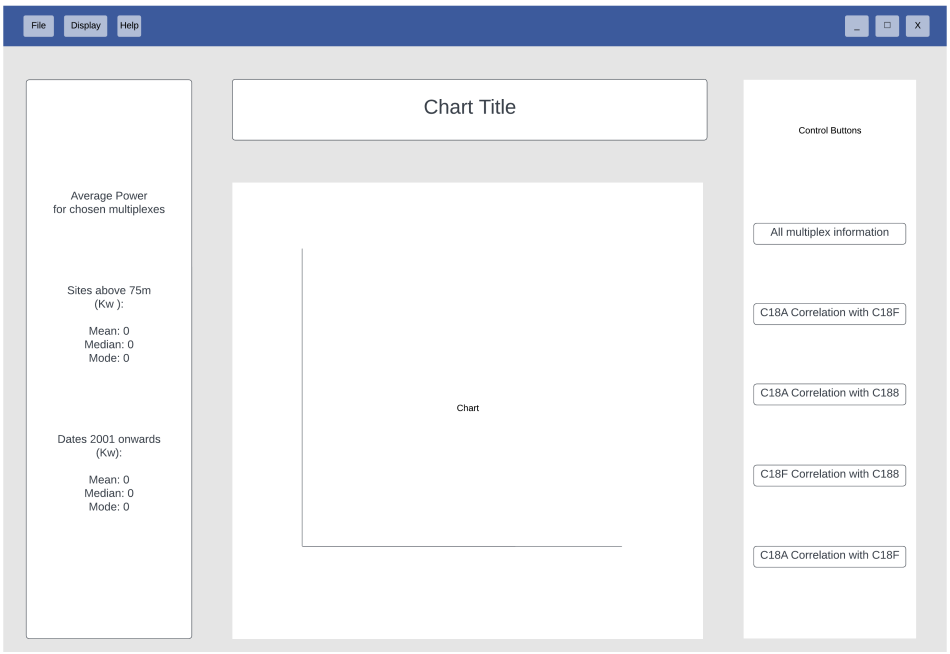
2.
```
# Function to display and calculate averages
def display_averages(data_in, row_in, criteria):
    # Get the averages and write them as a label
    self.l_column_label.append(tk.Label(self, text=averages_get(data_in, criteria), justify='left'))
    self.l_column_label[row_in-1].grid(row=row_in, column=0)
```

3.
```
# Creates a heatmap for the 2 multiplexes
def create_heatmap(self, mult1_in, mult2_in):
    screen_width = self.winfo_screenwidth()
    # the figure that will contain the plot
    fig = Figure(figsize=(9, 7), dpi=100)

    # adding the subplot
    plot1 = fig.add_subplot(111)

    # Assemble the data for the heatmap
    get_data_for_correlation(mult1_in, mult2_in)
    # Create heatmap for comparing data
    sns.heatmap(config.mult1_vs_mult2_df, vmin=0, vmax=1, annot=True, ax=plot1)
    # Set title
    title = mult1_in + ' v ' + mult2_in + ' Correlation Data'
    plot1.set_title(title)
    # Add Labels
    x_label = mult2_in + ' Service Labels (Block: ' + config.mult2_df.iloc[0]['Block']\
        + ', Freq.: ' + config.mult2_df.iloc[0]['Freq.'] + 'Hz)'
    y_label = mult1_in + ' Service Labels (Block: ' + config.mult1_df.iloc[0]['Block'] \
              + ', Freq.: ' + config.mult1_df.iloc[0]['Freq.'] + 'Hz)'
    plot1.update(dict(xlabel=x_label, ylabel=y_label))

    # Ensure axis labels aren't clipped
    fig.tight_layout()

    # creating the Tkinter canvas containing the Matplotlib figure
    canvas = FigureCanvasTkAgg(fig, master=self)
    canvas.draw()

    # placing the canvas on the Tkinter window
    canvas.get_tk_widget().grid(row=1, column=1, rowspan=8, padx=screen_width / 100, sticky='ns'))
```

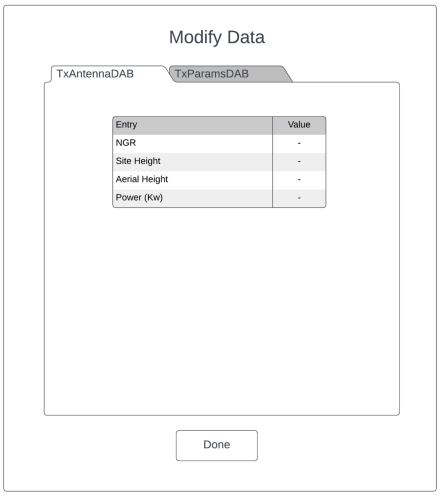

Appendix E - State Diagram for GUI
State Diagram
Diagram Key
First State
Terminator
State: Nature of program
Event: An instance that triggers a transition
Launch Program
Splash Screen displayed Attempting file loading
File(s) missing
Error message on display
User Acknowledges
File Load
Splash closes
Last view retrieved
New data saved to file
Change Data window on display
Change data button pushed
Display data
Error window closes
Splash closes
Main window on display
User prompted for TxAntennaDAB.csv
File selected
User prompted for TxParamsDAB.csv
File selected
Data loaded
Data Cleaned
Required data extracted
Data written to files in json format
veiw 1
Create chart
Calculate Averages
All data button pushed
veiw 2
Create chart
Comparison 1 button pushed
veiw 3
Create chart
Comparison 2 button pushed
veiw 4
Create chart
Comparison 3 button pushed
Import data button pressed
Quit program
Save current view to file

Appendix F - Wireframes for GUI

Wire frame for user interface:



Wire frame for data entry:



Appendix G - Java versus Python Code Example

```
# Groups data into format required by client and returns a string in json format
def group_data(ant_data_in, par_data_in, EID_in, desired_grp_ant_fields_in, desired_grp_par_fields_in):
    string = '\n'
    print("\nGrouping data for", EID_in)
    for par_line in par_data_in:
        # Look for required EID
        if (par_line.get('EID') == EID_in):
            # Use 'id' to find corresponding entry from ant_data_in
            current_id = par_line.get('id')
            # Find the corresponding entry in ant_data_in
            for ant_line in ant_data_in:
                # Find the matching id
                if ant_line.get('id') == current_id:
                    # Check if we can disregard this due to NGR value
                    write_field = True
                    for NGR_line in config.ignore_NGR_fields:
                        if ant_line.get('NGR') == NGR_line:
                            print("Skipping NGR value", (ant_line.get('NGR')), "for EID", EID_in, "- not required.")
                            write_field = False
                    # Write the data for this field if we're not skipping due to NGR field
                    if write_field:
                        # All checks complete, assemble the string
                        string = string + '{\n '
                        # Put the data together from par_data
                        for grp_par_line in desired_grp_par_fields_in:
                            string = string + '\"' + grp_par_line + '\": \"' + par_line.get(grp_par_line) + '\",\n'
                        # Put the data together for ant_data, renaming the required fields
                        for grp_ant_line in desired_grp_ant_fields_in:
                            if grp_ant_line == 'In-Use Ae Ht':
                                string = string + '\"Aerial height (m)\": \"' + ant_line.get(grp_ant_line) + '\",\n'
                            elif grp_ant_line == 'In-Use ERP Total':
                                string = string + '\"Power (kW)\": \"' + ant_line.get(grp_ant_line) + '\",\n'
                            else:
                                string = string + '\"' + grp_ant_line + '\": \"' + ant_line.get(grp_ant_line) + '\",\n'
                        # remove final comma and add newline
                        string = string[:len(string) - 2] + '\n'
                        string = string + '},\n'
                        # Once matching ant_line id found, no need to check the rest
                        break
    # remove final comma and add newline
    string = string[:len(string) - 2] + '\n'
    return string
```

Appendix H - Data format code examples

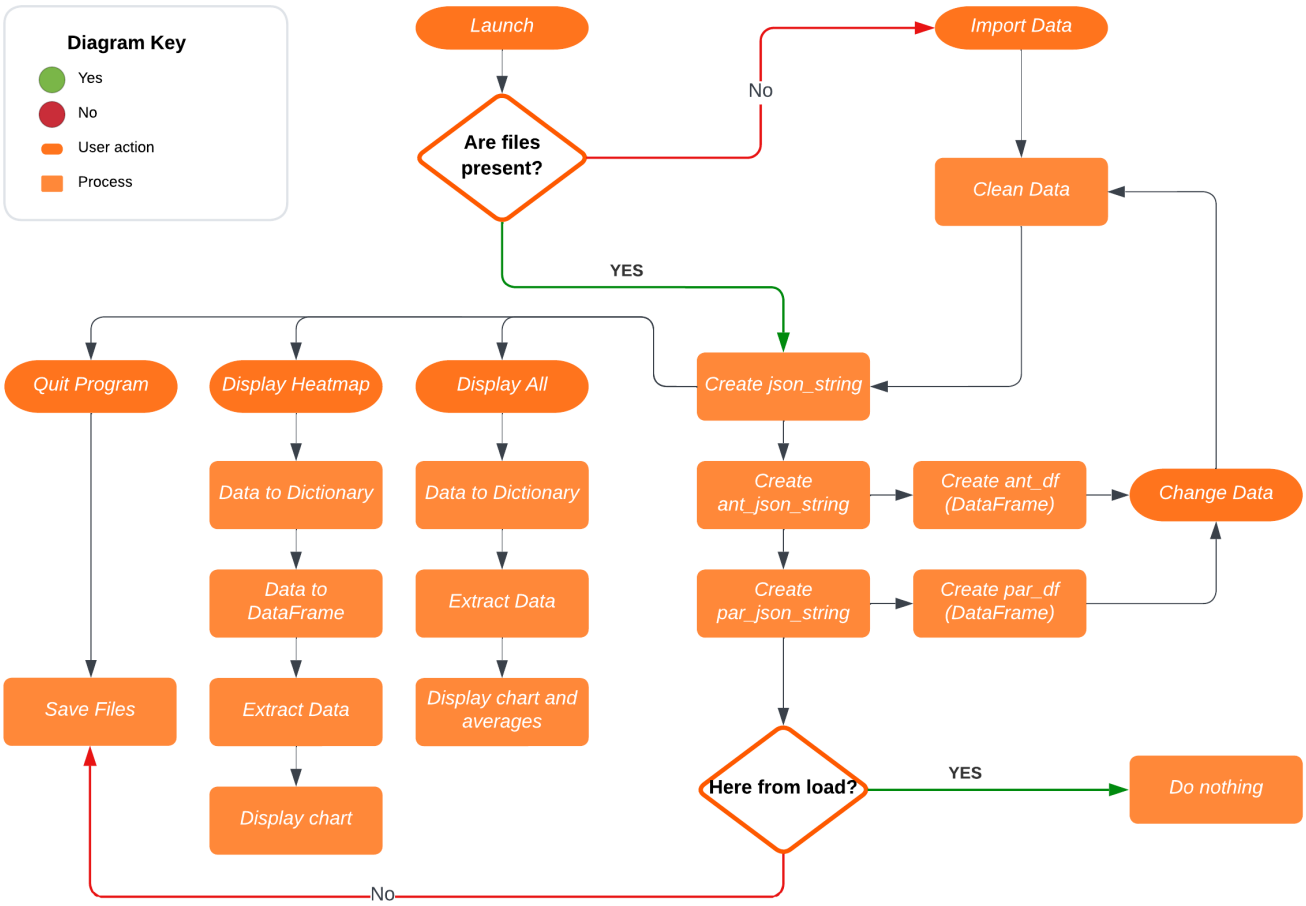
1.

```
# Exports all data to json format files grouped by id
def export_csvs(ant_data_in, par_data_in):

    pd.set_option("display.max_rows", None, "display.max_columns", None)

    # Write the ant_data to a DataFrame and to file
    ant_df = pd.DataFrame(ant_data_in)
    config.cleaned_ant_data = data_clean(ant_df)
    print("Writing to file TxAntennaDAB.json...", end="")
    config.cleaned_ant_data.to_json(path_or_buf="TxAntennaDAB.json", orient="records")
    print("...done")

    # Write the par_data to a DataFrame and to file
    par_df = pd.DataFrame(par_data_in)
    config.cleaned_par_data = data_clean(par_df)
    print("Writing to file TxParamsDAB.json...", end="")
    config.cleaned_par_data.to_json(path_or_buf="TxParamsDAB.json", orient="records")
    print("...done")

    # Convert the cleaned par and ant data back to lists and store as current state
    print("\nGrouping data as per client requirements and writing to CurrentState.json")
    ant_list = ast.literal_eval(config.cleaned_ant_data.to_json(orient="records"))
    par_list = ast.literal_eval(config.cleaned_par_data.to_json(orient="records"))
    json_handling.import_data(ant_list, par_list)
```

2.

```
# For converting csv data to client required data
def import_data(ant_data_in, par_data_in):

    # Setup json strings in required format
    config.json_string = '{'
    for line in config.desired_multiplexes:
        # Add the data to be grouped with NGR values
        config.json_string = config.json_string + '\"' + line + '\": ['
        config.json_string = config.json_string + (group_data(ant_data_in, par_data_in, line,
                                                config.desired_grp_ant_fields, config.desired_grp_par_fields))
        config.json_string = config.json_string + '],\n'
        # Add the data for correlation
        config.json_string = config.json_string + '\"' + line + '_correlation_data\": ['
        config.json_string = config.json_string + (group_data(ant_data_in, par_data_in, line,
                                                config.desired_cor_ant_fields, config.desired_cor_par_fields))
        config.json_string = config.json_string + '],\n'
    # remove final comma and add newline and bracket
    config.json_string = config.json_string[:len(config.json_string) - 2] + '\n}'
    # Write to file
    json_to_file('CurrentState.json', config.json_string)
```

Appendix I - Data flow diagram



Appendix J - Code samples for 3rd requirement

1.
```
def data_clean(data_in):
    print("Cleaning data...")
    # Fill any blank values with -1 so that negative values in calculations can be identified as having come
    # from missing data
    data_to_return = data_in.fillna(-1)
    # Replace empty strings with ("-1") for the same reasons as above
    data_to_return = data_to_return.replace("", "-1")
    # Truncate very long labels that would interfere with diagrams
    for column in data_to_return:
        data_to_return[column] = data_to_return[column].str.slice(stop=20)
    return data_to_return
```

2.
```
if grp_ant_line == 'In-Use Ae Ht':
    string = string + '\"Aerial height (m)\": \"' + ant_line.get(grp_ant_line) + '\",\n'
elif grp_ant_line == 'In-Use ERP Total':
    string = string + '\"Power (kW)\": \"' + ant_line.get(grp_ant_line) + '\",\n'
else:
    string = string + '\"' + grp_ant_line + '\": \"' + ant_line.get(grp_ant_line) + '\",\n'
```

3.
```
# If the Site height is more than 75m, we'll look at this one
if int(item['Site Height']) > 75:
    # Remove the comma that's in the power field
    string = remove_comma(item['Power (kW)'])
    entry.append(float(string))
```

4.
```
# Set the date for which to look afterwards
test_date = dt.datetime(2000, 12, 31)
# Note the format of the dates we're receiving
date_format = "%d/%m/%Y"
for item in data_in:
    # Convert the date to a format for comparison
    date_in = dt.datetime.strptime(item['Date'], date_format)
    # If the date is after 31/12/2000, we'll get the power value
    if date_in > test_date:
        # Remove the comma that's in the power field
        string = remove_comma(item['Power (kW)'])
        entry.append(float(string))
```

5.
```
# Function to get the 3 averages from the data given (criteria is height above 75m or date 2001 onwards)
def averages_get(data_in, criteria_in):
    data = find_values(data_in, criteria_in)
    string_out = "Mean: " + str(round(mean(data), 2))
    string_out = string_out + "\nMedian: " + str(round(median(data), 2))
    string_out = string_out + "\nMode: " + str(round(mode(data), 2))
    return string_out
```

Appendix K - Screenshot for 3rd requirement

Appendix L - Code for 4th requirement

1.
```
# adding the subplot
plot = fig.add_subplot(111)

# Get the data for each multiplex's site and stations (x and y axes) and add labels for the legend
for item in config.desired_multiplexes:
    x = Calculations.get_site_data(item)
    y = Calculations.get_station_data(item)
    lab = Calculations.get_labels(item)

    # plotting the graph for each multiplex (new colour automatically assigned for each one)
    plot.scatter(x, y, label=lab)
```

2.
```
# Returns the site data for client specified multiplexes
# Returns five lots of each site in a list to match up with the 5 Serv Labels that get_station_data will return
def get_site_data(multiplex_in):
    site_list = []
    json_dict = json.loads(config.json_string)

    def site_multiplier(multiplex_in_2):
        nonlocal site_list
        nonlocal json_dict
        for site in json_dict[multiplex_in_2]:
            count = 0
            # Put 5 copies of this site in the list
            while count < 5:
                # If '-1' found, give the label as 'MISSING LABEL'
                if(site['Site']) == '-1':
                    site_list.append('MISSING LABEL')
                else:
                    site_list.append(site['Site'])
                count = count + 1

    site_multiplier(multiplex_in)
    return site_list
```

3.
```
# Returns the Station data for the client specified multiplexes
def get_station_data(multiplex_in):
    stations_list = []
    serv_labels = ['Serv Label1 ', 'Serv Label2 ', 'Serv Label3 ', 'Serv Label4 ', 'Serv Label10 ']
    json_dict = json.loads(config.json_string)

    # For each site in the given multiplex...
    for station in json_dict[make_correlation_data_string(multiplex_in)]:
        # ...add the service labels to stations_list
        for label in serv_labels:
            # If '-1' found, give the label as 'MISSING LABEL'
            if(station[label]) == '-1':
                stations_list.append('MISSING LABEL')
            else:
                stations_list.append(station[label])

    return stations_list
```

Appendix M - Screenshot for 4th requirement

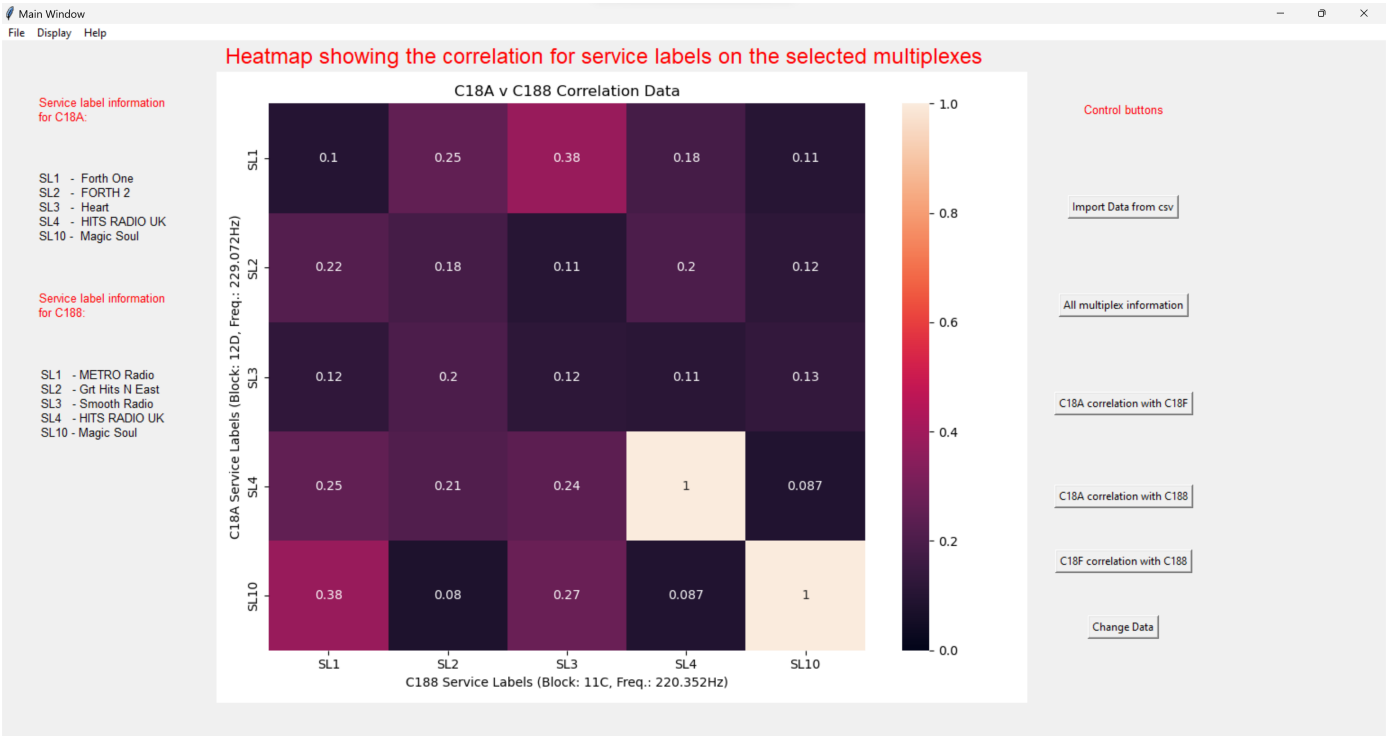
Appendix N - Code for 5th requirement

1.
```
# Heatmap 1 button response
def heat_1():
    # Change main title
    self.subtitle.destroy()
    self.subtitle = tk.Label(self, text='Heatmap showing the correlation for service labels on the '
                             'selected multiplexes', font=('Helvetica', 18), fg="red", anchor="center")
    self.subtitle.grid(row=0, column=0, columnspan=3, padx=screen_width / 100)
    Plot.plot_heat_1(self)
    # Update the display menu radio buttons
    disp_var.set('1')
    menubar.update()
    # Update the key
    display_heatmap_ledger(config.desired_multiplexes[0], config.desired_multiplexes[1])
    # Update stored display value
    config.display_value = 1
```

2.
```
# Create a heatmap for the first pair of multiplexes
def plot_heat_1(self):
    create_heatmap(self, config.desired_multiplexes[0], config.desired_multiplexes[1])
```

3.
```
# Returns a sequence match (a ratio) for how closely the text of the service labels match
def apply_sm(c1, c2):
    send_list = []
    for item in c2:
        send_list.append(difflib.SequenceMatcher(isjunk=None, a=c1, b=item).ratio())
    return send_list
```

Appendix O - Screenshot for 5th requirement



References

[1] W. McKinney, *Python Cookbook : Recipes for Mastering Python 3*. O'Reilly Media Inc, 2013.

[2] tutorialspoint, *Python - multithreaded programming*, Accessed 2023-08-16. [Online]. Available: https://www.tutorialspoint.com/python/python_multithreading.htm.

[3] Y. Sharma, *Synchronization in python – synchronize threads in python*, Accessed 2023-16-08. [Online]. Available: https://www.askpython.com/python/examples/synchronization-in-python.

[4] Lorenzo, *Nice threads in python*, Accessed 2023-16-08. [Online]. Available: https://lbolla.info/python-nice-threads.html.

[5] Python Software Foundation, *Threading — thread-based parallelism*, Accessed 2023-19-08. [Online]. Available: https://docs.python.org/3/library/threading.html.

[6] Usability.gov, *User interface design basics*, Accessed 2023-08-12. [Online]. Available: https://www.usability.gov/what-and-why/user-interface-design.html.

[7] J. Cao, *Gestalt principles for designers – applying visual psychology to modern day design*, Accessed 2023-08-12. [Online]. Available: https://blog.teamtreehouse.com/gestalt-principles-designers-applying-visual-psychology-modern-day-design.

[8] D. Fadeyev, *Using light, color and contrast effectively in ui design*, Accessed 2023-08-12. [Online]. Available: https://usabilitypost.com/2008/08/14/using-light-color-and-contrast-effectively-in-ui-design/.

[9] S. Martin, *Effective visual communication for graphical user interfaces*, Accessed 2023-08-12. [Online]. Available: http://web.cs.wpi.edu/~matt/courses/cs563/talks/smartin/int_design.html.

[10] N. Kacirek, *Java vs. python for data science*, Accessed 2023-19-08. [Online]. Available: https://medium.com/@nicole.k.kacirek/java-vs-python-for-data-science-2fb5c2715206.

[11] S. Reddy, *Why do data scientists prefer python over java?* Accessed 2023-19-08. [Online]. Available: https://medium.com/quick-code/why-do-data-scientists-prefer-python-over-java-d570499a1fcd.

[12] ProjectPro, *Java vs python for data science in 2023-what's your choice?* Accessed 2023-19-08. [Online]. Available: https://www.projectpro.io/article/java-vs-python-for-data-science-in-2021-whats-your-choice/433.

[13] snapLogic, *Python vs. java performance*, Accessed 2023-19-08. [Online]. Available: https://www.snaplogic.com/glossary/python-vs-java-performance.

[14] SparkBy{Examples}, *Pandas get row number of dataframe*, Accessed 2023-19-08. [Online]. Available: https://sparkbyexamples.com/pandas/get-row-number-in-pandas/.

[15] pandas via NumFOCUS, Inc., *Pandas.dataframe.drop*, Accessed 2023-19-08. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html.

[16] mukulsomukesh, *How to rename multiple column headers in a pandas dataframe?* Accessed 2023-19-08. [Online]. Available: https://www.geeksforgeeks.org/how-to-rename-multiple-column-headers-in-a-pandas-dataframe/.

[17] M. Pogrebinsky, *Python multithreading vs. java multithreading - important considerations for high performance programming*, Accessed 2023-19-08. [Online]. Available: https://topdeveloperacademy.com/articles/python-multithreading-vs-java-multithreading-important-considerations-for-high-performance-programming.

[18] A. Modi, *Java vs. python: Which should you choose?* Accessed 2023-19-08. [Online]. Available: https://opensource.com/article/19/12/java-vs-python.

[19] J. Juneau, J. Baker, V. Ng, L. Soto and F. Wierzbicki, *The definitive guide to Jython - Python for the Java Platform*. Springer-Verlag New York Inc., 2010.

[20] snapLogic, *Python vs. java*, Accessed 2023-19-08. [Online]. Available: https://www.snaplogic.com/glossary/python-vs-java.

[21] A. Sannikov, *Python data structures: Lists, dictionaries, sets, tuples (2023)*, Accessed 2023-19-08. [Online]. Available: https://www.dataquest.io/blog/data-structures-in-python/.

[22] W. McKinney, *Python for Data Analysis, 3rd Edition*. O'Reilly Media Inc, 2022.

[23] InterviewBit, *File system vs dbms: Key difference between file system and dbms*, Accessed 2023-07-10. [Online]. Available: https://www.interviewbit.com/blog/file-system-vs-dbms/.

[24] C. SINGH, *Advantages and disadvantages of dbms: Dbms vs file system*, Accessed 2023-07-10. [Online]. Available: https://beginnersbook.com/2015/04/dbms-vs-file-system/.

[25] A. Simec and M. Maglicic, 'Comparison of json and xml data formats,' in *Central European Conference on Information and Intelligent Systems*, 2014.

[26] Sahana, *Advantages and disadvantages of json*, Accessed 2023-08-12. [Online]. Available: https://www.techquintal.com/advantages-and-disadvantages-of-json/.

[27] M. Stojiljković, *Python statistics fundamentals: How to describe your data*, Accessed 2023-08-13. [Online]. Available: https://realpython.com/python-statistics/.

[28] NumPy, *Statistics*, Accessed 2023-08-13. [Online]. Available: https://numpy.org/doc/stable/reference/routines.statistics.html.

[29] D. MOORE, G. McCABE and B. CRAIG, *Introduction to the Practice of Statistics 6th Edition*. W. H. Freeman and Company, 2009.

[30] M. Yi and M. Sapountzis, *Essential chart types for data visualization*, Accessed 2023-08-14. [Online]. Available: https://chartio.com/learn/charts/essential-chart-types-for-data-visualization/.

[31] S. Agrawal, *Introduction to matplotlib and seaborn*, Accessed 2023-08-14. [Online]. Available: https://medium.com/analytics-vidhya/introduction-to-matplotlib-and-seaborn-e2dd04bfc821.

[32] Insight Software, *When (and why) to use heat maps*, Accessed 2023-08-14. [Online]. Available: https://insightsoftware.com/blog/when-and-why-to-use-heat-maps/.

[33] O. E. Dictionary, *Oxford English Dictionary*. Oxford University Press, 2023.

[34] Lorenzo, *Ethics vs. morals*, Accessed 2023-16-08. [Online]. Available: https://www.diffen.com/difference/Ethics_vs_Morals.

[35] S. Perkins, *Between law and ethics*, Accessed 2023-16-08. [Online]. Available: https://www.ssplawgroup.com/the-relationship-between-law-and-ethics.

[36] M. Hildebrandt, *Closure: On ethics, code and law*, Accessed 2023-16-08. [Online]. Available: https://lawforcomputerscientists.pubpub.org/pub/nx5zv2ux/release/35.

[37] M. S. Baucus, W. I. Norton, D. A. Baucus and S. E. Human, 'Fostering creativity and innovation without encouraging unethical behavior,' *Journal of Business Ethics*, vol. 81, no. 1, pp. 97–115, 2008, ISSN: 01674544, 15730697. [Online]. Available: http://www.jstor.org/stable/25482200 (visited on 17/08/2023).

[38] The Defenders Criminal Defense Lawyers, *Can a lawyer defend someone they know is guilty?* Accessed 2023-16-08. [Online]. Available: https://thedefenders.net/blogs/can-a-lawyer-defend-someone-they-know-is-guilty/.

[39] K. Quach, *Euro parliament green lights its ai safety, privacy law*, Accessed 2023-08-17. [Online]. Available: https://www.theregister.com/2023/06/15/european_parliament_ai_act/.

[40] C. Boine, *Emotional attachment to ai companions and european law*, Accessed 2023-08-17. [Online]. Available: https://mit-serc.pubpub.org/pub/ai-companions-eu-law/release/3.

[41] A. HAYES, *What was enron? what happened and who was responsible*, Accessed 2023-08-17. [Online]. Available: https://www.investopedia.com/terms/e/enron.asp.

[42] R. Hotten, *What is volkswagen accused of?* Accessed 2023-08-17. [Online]. Available: https://www.bbc.co.uk/news/business-34324772.

[43] P. PATEL, *Engineers, ethics, and the vw scandal*, Accessed 2023-08-17. [Online]. Available: https://spectrum.ieee.org/vw-scandal-shocking-but-not-surprising-ethicists-say.

[44] L. Curwen, *The collapse of enron and the dark side of business*, Accessed 2023-08-17. [Online]. Available: https://www.bbc.co.uk/news/business-58026162.

[45] United States Holocaust Memorial Museum, Washington, DC, *Oskar schindler*, Accessed 2023-08-17. [Online]. Available: https://encyclopedia.ushmm.org/content/en/article/oskar-schindler.

[46] T. Retz, *Progress and the Scale of History* (Elements in Historical Theory and Practice). Cambridge University Press, 2022. DOI: 10.1017/9781009026758.