Digital Audio Broadcasting

Radio Station Analysis

Report

Table of Contents

1. Theory and Application of Advanced Programming Techniques

a. Program Redesign with Python Threads (500 words)

Python Threads

Concurrency is a critical aspect of modern software development, as it requires multiple tasks to be run simultaneously [1]. It allows the running of independent pieces of a specific program in parallel with another, improving overall performance [2]. Threading is a Python module that facilitates concurrency, allowing multiple tasks to be completed during overlapping periods of time [3].

A part of the DAB Data Analysis application that could be redesigned to implement Python Threads is the data loading and cleaning process. In the current design, the data is loaded and cleaned sequentially, which could lead to long processing times, especially when dealing with large datasets [4]. By implementing Python Threads, the data loading and cleaning processes can be done concurrently, taking advantage of multicore processors, and potentially improving the overall responsiveness and efficiency of the application [2,4] (Appendix: Figure 1).

Currently, the application's workflow involves loading the raw data from the CSV files; cleaning and pre-processing the data, removing duplicates, handling missing values and applying transformations; followed by displaying the cleaned data and status updates to the user [5].

With Python Threads, the process would utilise a Main Thread, Data Loading Thread and Data Cleaning Thread. In the Main Thread, The GUI and User Interface (UI) components are initialised, then separate threads are created for loading and cleaning the data; they are then initialised [6]. The raw data is loaded and passed to the cleaning thread for further processing. The status is then updated and displayed to indicate the progress. Once the raw data is received from the loading thread, data cleaning and pre-processing is performed concurrently on a separate portion of the dataset [7]. The status label is updated to indicate the progress of data cleaning, and the cleaned data is then communicated back to the main thread. The cleaned data would be received by the main thread and displayed in the GUI output area. A status update would then be displayed to indicate the completion of the cleaning process (Figure 2). The flowchart diagram (Figure 3) illustrates the workflow of the application, from loading the files, to converting and saving them as JSON.

The implementation of Python Threads could potentially enhance the efficiency and responsiveness of the data loading and cleaning process. Python Threads allows multiple tasks to run concurrently, by utilising available CPU cores [8]. This can lead to faster data processing and a more responsive user interface [9,10]. Unfortunately, Python threads can be difficult to debug, as it can be hard to track down the source of the problem [11]. Python threads may also be a security risk if not properly synchronised, as shared data can become corrupt or crash the program [12]. Nevertheless, locks can be used to protect shared data to prevent data corruption, and threads can be used to improve performance by overlapping I/O operations with CPU-bound tasks [13,14,15].

Ultimately, concurrency can significantly improve user experience by reducing data processing time and allowing the GUI to remain responsive during data loading and cleaning.

b. User Interactions and the GUI (470 words)

The GUI

The Graphical User Interface (GUI) for the application has been designed to seamlessly support the range of user interactions outlined in the client's requirements [16]. A user-first approach was taken in designing the interface, with key interactions that facilitate a user-friendly experience, efficient data analysis and clear feedback to the user [17].

The application provides a "Browse" button that opens a file dialog and allows users to select CSV files (Figure 3). The interaction is implemented using Tkinter's 'filedialog' module [18]. Users can select multiple files, two or more, which are then translated to JSON format, at this point, the file is saved. The data is then cleaned: dropping columns with no data, handling missing values by putting in zeros for missing floating-point numbers and converting the data types of specific columns. The user can then input specific 'NGR' rows to remove from the data set. At this point, the user can save the data set, preserving the state of the data when the program is closed and makes it available when the program is reopened, before moving on to the next step.

Navigation, Status Bar and Output

A tab-based navigation is implemented to organise the application, each tab representing a specific set of tasks [19]. The 'tk.Notebook' widget is used to create the tabs which provides the user with a clear and organised way to interact with the various functionalities of the application [20]. A status bar, using 'tk.Label', is implemented above the output area which provides users with feedback as they interact with the application, using the 'tk.Text' widget [21]. The status bar is placed above the output area so that user can clearly see the result of the interactions as they happen (Appendix: Figure 4).

Data Processing

The users can initiate data processing tasks such as cleaning the data set, applying specific modifications, extracting relevant columns, and calculating statistics, as well as manipulate the range of values used in generating output and visualisations, all with a highly intuitive experience. This is achieved through seamless initiation via dedicated buttons for each task on the respective tabs [22].

Data Visualisation

The application allows users to visualise data through graphs and charts, using 'matplotlib' [23]. The "Data Visualisation" tab allows the user to generate and display graphs based on selected criteria. These graphs help the user to understand trends and patterns in the data. The 'FigureCanvasTkAgg' widget is used to display the graph in the GUI [24].

The integration of tabs, the status label and output area serve to significantly enhance the user experience. The GUI constructs and layout implemented were made to support user interactions effectively, and cater to users' needs, facilitating effective data analysis. This strategic design focuses on providing clarity, organisation and real-time user feedback, culminating in an environment that promotes an intuitive user-friendly for data analysis.

c. Evaluation of High-Level Programming Languages: Java and Python (472 words)

Java and Python are both general-purpose, high-level programming languages which offer distinct features for data manipulation. They are both powerful for storing, structuring, and manipulating data, but they differ in terms of mechanism and syntax [25,26]. Having experienced both languages from both a programming and theoretical standpoint, I find Python to be more effective for the manipulation of data containers due to its ease of use, concise syntax, rich built-in functions, and data structures, along with its extensive libraries, all while offering a positive development experience [27].

Python's simplicity and readability play a crucial role in its effective data manipulation. It's list comprehensions provide a concise way to create and manipulate lists. For extracting DAB multiplexes, list comprehension was used to create a subset of data based on specific criteria, which enhanced readability and efficiency [27]. The code in Appendix Figure 6, demonstrates Python's ability to filter and extract specific columns from a DataFrame.

Python's built-in functions are a significant contribution to data manipulation [28]. The 'map()' function, used to apply a function to each element of a list, also enhancing code modularity, was used to convert the 'Date' column to years through the application of a custom function (Appendix 7) [29]. The statistical calculations for the 'In-Use ERP Total', 'Site Height' and 'Date', were done using the 'mean ()', mode()' and 'median' functions from the Pandas library [30]. These functions provided a straightforward and efficient method for the statistical calculations, enhancing the development experience.

Built-in data structures such as lists, dictionaries and sets provide powerful tools for data manipulation in Python. Dictionaries, for instance, are widely used for key-value pair storage, offering efficient access and modification [31]. In the Data Analysis Application program, Python dictionaries are employed to easily manipulate and transform data containers such as renaming columns in the DataFrame.

Libraries such as Pandas and NumPy make Python the preferred language for data handling and structuring through enhanced data manipulation [32]. Pandas was used extensively in the program to efficiently create and manipulate DataFrames, for seamless filtering, grouping and data aggregation. With syntax that is easier to read and write in comparison to Java, Python provides easier to write, clear and concise code that easily manipulates data containers.

While Java also offers mechanisms for data container manipulation through its frameworks and arrays, it generally involves a steeper learning curve and a verbose syntax in comparison to Python [33]. The absence of list comprehensions and the need for type declarations make code longer and less readable. This can impact development speed and increase the possibility of errors.

The examples from the Data Analysis program highlights Python's efficiency in filtering, extracting, and transforming data. While Java offers similar functionality, Python's developer-friendly approach, concise syntax, versatile built-in data structures and robust libraries makes it more efficient for manipulating data containers and performing similar tasks.

2. Design Decisions for the Selection, Storage and Manipulation of Data

a. Selected Data Format: JSON (260 words)

JSON (JavaScript Object Notation) was chosen to address the client's data manipulation requirements, based on its alignment with data handling, readability, flexibility along with its other advantages [34].

JSON is a lightweight, readable, and easy-to-parse format. It has a simplistic and concise syntax which makes it suited to structurally represent diverse data types, making it efficient to store and transmit [35]. This directly resonated with the client's requirement for handling multiplex data constituting the features 'Site', 'Freq.' and the various 'Serv Labels' (Appendix 8).

The client's data manipulation requirements necessitated a format that accommodated hierarchical relationships. JSON's ability to represent complex structures [34] combined with its compatibility with various programming languages made this fit seamless [35].

It is flexible and can be used to represent a wide variety of data types. JSON's flexibility aligns well with the diverse data needs of the application, which include storing status messages, and adding new fields without affecting the existing structure [36]. Python's built-in support for JSON through its 'json' module simplifies the process of converting the data to JSON and back. This support streamlines data handling within the application [37].

On the other hand, JSON is not well-suited for representing complex data structures as other formats. JSON's textual nature could result in increased data size, while its lack of a formal schema could have resulted in potential data validation issues, however, this was alleviated by cleaning the data beforehand [37]. The trade-off with these disadvantages was acceptable, as they were not significant for this application, for which the advantages greatly outweigh the disadvantages.

b. Identification of Code Constructs/Functions/APIs (280 words)

The client's third requirement involved extracting and analysing data from the DAB multiplexes to calculate the mean, mode and median for 'In-Use ERP Total', 'Site Height' more than 75, and 'Date' from 2001 onwards. The client also required information to generate and output the statistical calculations.

To achieve this, the data was loaded, cleaned, and converted to JSON. The DataFrame was filtered using Pandas to extract the rows with the specified 'EID' values or DAB multiplexes, C18A, C18F, C188. A new DataFrame was then created with specific columns including 'NGR', 'Site', 'Site Height' and 'In-Use Ae Ht' and 'In-Use ERP Total' which are subsequently renamed to 'Aerial Height (m)' and 'Power (kW)'.

This DataFrame was then used to produce the required statistical calculations, using Pandas Series methods 'mean()', 'mode().iloc[0]' and 'median()'. These methods were used as they are very efficient and provide accurate results [38]. For the 'In-Use ERP Total', the DataFrame

was used as is, while it was filtered to produce the statistical calculations for 'Site Height' more than 75 and 'Date' from 2001 onwards. TKinter's 'Text' widget was used to display the results in the GUI (Appendix 7).

Pandas played a pivotal role for this implementation. Its intuitive and expressive syntax enable seamless data manipulation. The use of Pandas functions aligns with best practices due to their optimised performance and ease of use [39].

The functions and APIs used are fast, reliable, easy to use and built-in to Python [39]. They provide powerful data manipulation, efficient data handling and analysis, also taking away the hassle of using mathematical formulas. These all contributed to ensuring that the code is easy to understand, maintain and extend.

c. Data Visualisation (165 words)

Visualisations are generated within the Data Visualisation tab of the Data Analysis program. The 'seaborn' library was chosen due to its simplicity and built-in capabilities for creating visualisations, including heatmaps [40]. Seaborn's high-level functions allows for the use of concise code to create complex visualisations. It works seamlessly with Pandas DataFrames, which are used throughout the application [41].

Before creating the heatmap, the data was cleaned and processed. Columns with non-numeric values were removed, missing values were addressed, and numeric values were converted where needed. A new DataFrame was created with the necessary columns, joined with the DAB multiplexes.

The integration of Seaborn's heatmap visualisation into the application's GUI effectively fulfils the client's requirements. The chosen API and its integration enable users to gain insights into the relationship between the Site, Frequency and the various Serv Labels (Appendix Figure 9). The visualisation allows users to identify trends within the data. The transformation of the data points into visualisations, bridging the gap between data and understanding.

d. Data Analysis (219 words)

In order to determine whether there is a significant correlation between the features 'Freq.', 'Block', 'Serv Label1', 'Serv Label2', 'Serv Label3', 'Serv Label 4' and 'Serv Label10', a comprehensive analysis was conducted.

To determine the significance of correlations, the Pearson correlation coefficient, and associated p-values, which reflect the probability of observing a correlation, were calculated using the Pandas library. A significance level of 0.05 was chosen as it is a common threshold in statistical analysis. The correlation matrix is visualised as a heatmap using Seaborn's 'sns.heatmap()' function. This function takes the correlation matrix as input and automatically generates a heatmap. The correlation analysis revealed that there was no correlation, with a

red heatmap (Appendix Figure 10). Attributes with p-values less than 0.05 would have been considered to have a statistically significant correlation.

The generated heatmap plot is integrated into the GUI using the 'FigureCanvasTkAgg' widget from 'matplotlib.backends.backend_tkagg' module, which allows a Matplotlib figure to be displayed in a Tkinter window. This was chosen in order to provide the user with one main output area throughout the application.

Generally, the application's users can observe the correlation heatmap to identify visually, the significance of relationships between attributes, unfortunately, in this case, there is none. This information generally helps the user to understand potential interdependencies between different aspects of DAB stations' characteristics.

3. Ethics, Moral and Legal Aspects

a. Ethical, Moral and Legal Aspects of Computing (399 words)

The statement "software engineers should not be subject to regulation by an ethical framework" raises concerns regarding technology, innovation, data security and societal responsibility. Along with potential bureaucratic hurdles comes warranted concern; however, it is imperative to recognize that a thoughtful, well-balanced ethical framework can possibly encourage innovation, enhance security practices, and promote responsible technological advancements.

Ethical frameworks provide guidelines that encourage engineers to approach innovation with societal well-being in mind. Ethical frameworks exist in many industries, including financial, healthcare and technology [42, 43]. These frameworks can inspire engineers to think critically about the potential implications of their works, rather than suppressing their creativity. Ethical considerations can motivate software engineers to develop and maintain security, reducing vulnerabilities of their creations while minimising the possibility of data breaches or any cyber-attacks [46].

A balance between ethical considerations and innovation is crucial. Rather than thinking of ethical frameworks as impeding progress, it can guide innovation towards sustainable avenues. This can allow engineers to explore and improve technologies, while considering potential risks and the impact on societies.

Software engineers have a responsibility to consider the ethical ramifications of their field [42]. Software engineers must hold themselves accountable to create software that is safe and secure, avoiding technology that can be deemed potentially harmful. Ethical frameworks help to guide decision-making when creating systems that could possibly be weaponised or may be dangerous.

Bureaucracy does not have to be the backbone of ethical frameworks. Regulations can focus on areas of concern such as privacy and security, rather than the entire scope of software engineering. The framework can be designed to accommodate the dynamic nature of innovation and technology. Creating an adaptable framework allows for its advancement alongside technology [42]. Users hold a level of trust and confidence in the technology that is available to them. Ethical frameworks preserve public confidence, as it prevents the development of technology without the consideration of potential harm [45,47].

Software engineers should be held to high ethical standards, but it is important that they have the freedom to innovate and create, contributing to technological enhancements. Concerns regarding the potential negative impacts of innovation and technology can be managed through the implementation of a dynamic ethical framework that fosters innovation and technological exploration while considering social impacts of the technology. The alignment of technological creations and ethical considerations can allow engineers to contribute to an inclusive and secure digital future.

Appendix

Without Threading

Clean the Data

```
# Function to clean the data
def clean_data():
    global df

    # Drop columns with no data
    df.dropna(axis=1, how='all', inplace=True)

    # Handling missing values for float64 columns
    for column in df.columns:
        if df[column].dtype == 'float64':
            df[column].fillna(0, inplace=True)

    # Convert 'In-Use ERP Total' column to float
    df['In-Use ERP Total'] = df['In-Use ERP Total'].str.replace(',', '').astype(float)

    # Round the 'In-Use ERP Total' column to 2 decimal places
    df['In-Use ERP Total'] = df['In-Use ERP Total'].round(2)
```

```
# Function to clean the data and display the first five rows
def clean_and_display_data():
    global df

    # Check if data is loaded
    if df is None:
        messagebox.showerror("Error", "No data available. Please load data first.")
        return

    # Clean the data
    clean_data()

    # Display the cleaned data in the output text area
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, df.head(5).to_string(index=False))
    output_text.insert(tk.END, f"\n\nNumber of Rows: {df.shape[0]}\nNumber of Columns: {df.shape[1]}")
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="Data cleaned successfully!")
```

Load the Data

```
# Function to load the CSV file and convert them to JSON
def load_csv_files():
    global df

    file_paths = filedialog.askopenfilenames(title="Select CSV Files", filetypes=[("CSV Files", "*.csv")])
    if len(file_paths) < 2:
        messagebox.showerror("Error", "Please select at least two CSV files.")
        return

    # Read the first CSV file to initialize the DataFrame
    try:
        df = pd.read_csv(file_paths[0], encoding='utf-8')
    except UnicodeDecodeError:
        df = pd.read_csv(file_paths[0], encoding='latin')

    for file_path in file_paths[1:]:
        try:
            combined_df = pd.read_csv(file_path, encoding='utf-8')
        except UnicodeDecodeError:
            combined_df = pd.read_csv(file_path, encoding='latin')

        # Merge based on 'id' column
        df = df.merge(combined_df, how='outer', on='id')

    # Generate a unique filename for backup
    backup_filename = f"data_{time.strftime('%Y%m%d%H%M%S')}.json"

    # Translate DataFrame to JSON format
    json_data = df.to_json(orient="records")

    # Save the JSON data to a file (for backup)
    with open(backup_filename, "w") as file:
        file.write(json_data)

    # Provide feedback to the user
    status_label.config(text="CSV files loaded and converted to JSON format successfully!")
```

With Threading

```
import threading

# Function to load the CSV file and convert them to JSON
def load_csv_files():
    global df

    file_paths = filedialog.askopenfilenames(title="Select CSV Files", filetypes=[("CSV Files", "*.csv")])
    if len(file_paths) < 2:
        messagebox.showerror("Error", "Please select at least two CSV files.")
        return

    # Read the first CSV file to initialize the DataFrame
    try:
        df = pd.read_csv(file_paths[0], encoding='utf-8')
    except UnicodeDecodeError:
        df = pd.read_csv(file_paths[0], encoding='latin')

    # Create a thread for each file loading
    threads = []
    for file_path in file_paths[1:]:
        thread = threading.Thread(target=load_csv_thread, args=(file_path,))
        threads.append(thread)
        thread.start()

    # Wait for all threads to complete
    for thread in threads:
        thread.join()

    # Generate a unique filename for backup
    backup_filename = f"data_{time.strftime('%Y%m%d%H%M%S')}.json"

    # Translate DataFrame to JSON format
    json_data = df.to_json(orient="records")

    # Save the JSON data to a file (for backup)
    with open(backup_filename, "w") as file:
        file.write(json_data)

    # Provide feedback to the user
    status_label.config(text="CSV files loaded and converted to JSON format successfully!")


# Thread function for loading CSV files
def load_csv_thread(file_path):
    global df

    try:
        combined_df = pd.read_csv(file_path, encoding='utf-8')
    except UnicodeDecodeError:
        combined_df = pd.read_csv(file_path, encoding='latin')

    # Merge based on 'id' column
    with thread_lock:
        df = df.merge(combined_df, how='outer', on='id')

# Function to clean the data
def clean_data():
    global df

    # Drop columns with no data
    df.dropna(axis=1, how='all', inplace=True)

    # Handling missing values for float64 columns
    for column in df.columns:
        if df[column].dtype == 'float64':
            df[column].fillna(0, inplace=True)

    # Convert 'In-Use ERP Total' column to float
    df['In-Use ERP Total'] = df['In-Use ERP Total'].str.replace(',', '').astype(float)

    # Round the 'In-Use ERP Total' column to 2 decimal places
    df['In-Use ERP Total'] = df['In-Use ERP Total'].round(2)
```

Figure 1: Clean and Load Data without and with Threading

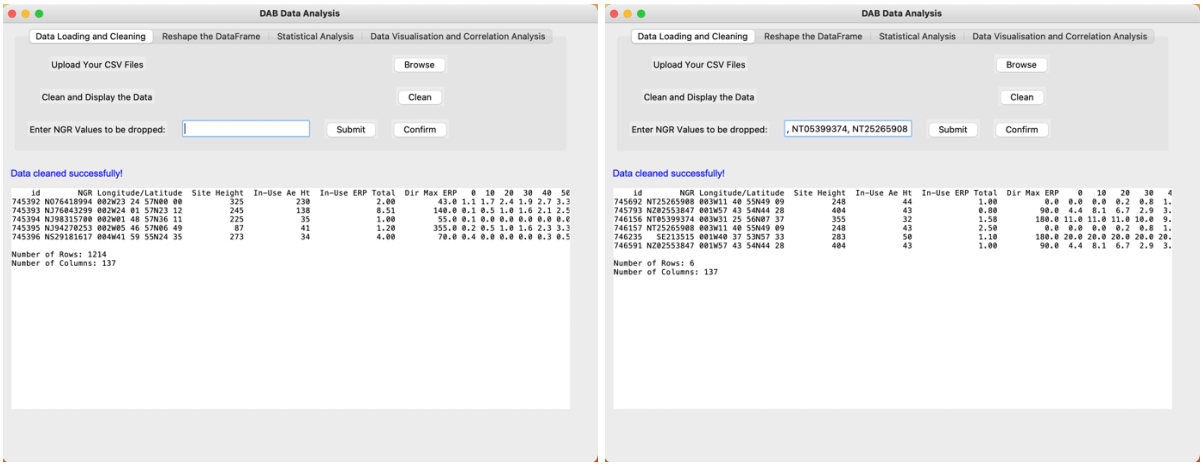

Figure 2: Data Cleaning Tab

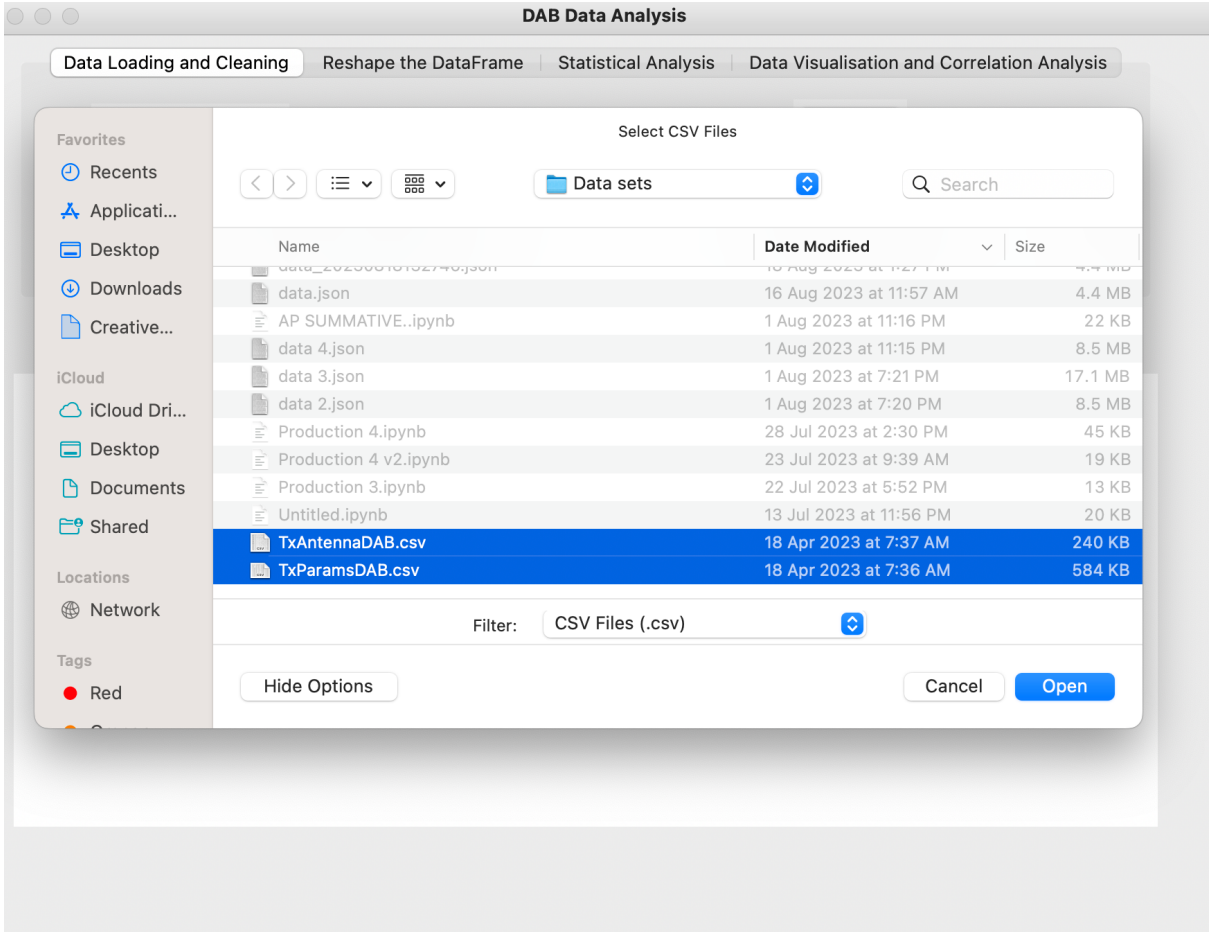

Figure 4: File Dialog for selecting CSV files

GUI

```
#Main GUI
root = tk.Tk()
root.title("DAB Data Analysis")
root.geometry("880x650")

# Screen width and height
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

# Centre the window
x = (screen_width - root.winfo_reqwidth()) // 2
y = (screen_height - root.winfo_reqheight()) // 2
root.geometry(f"+{x}+{y}")

# Notebooks(Tabs)
notebook = ttk.Notebook(root)
notebook.grid(row=0, column=0, columnspan=6)

#Window Tabs
clean_tab = ttk.Frame(notebook)
reshape_tab = ttk.Frame(notebook)
stats_tab = ttk.Frame(notebook)
visualise_tab = ttk.Frame(notebook)

notebook.add(clean_tab, text="Data Loading and Cleaning")
notebook.add(reshape_tab, text="Reshape the DataFrame")
notebook.add(stats_tab, text ="Statistical Analysis")
notebook.add(visualise_tab, text="Data Visualisation and Correlation Analysis")
```

```
# Clean Tab
```

```
# Button to load CSV files
load_button_label = tk.Label(clean_tab, text="Upload Your CSV Files", relief=tk.FLAT)
load_button = tk.Button(clean_tab, text="Browse", command=load_csv_files)

# Button to clean and display data
clean_button_label= tk.Label(clean_tab, text="Clean and Display the Data")
clean_button = tk.Button(clean_tab, text="Clean", command=clean_and_display_data)

# Layout
load_button_label.grid(row=1, column=0, padx=10, pady=10)
load_button.grid(row=1, column=3, padx=10, pady=10)

clean_button_label.grid(row=2, column=0, padx=10, pady=10)
clean_button.grid(row=2, column=3, padx=10, pady=10)
```

```
# Drop NGR Values
# Label, input box, submit and confirmation for NGR values to be dropped
ngr_label = tk.Label(clean_tab, text="Enter NGR Values to be dropped:")
ngr_entry_var = tk.StringVar(value="NZ02553847, SE213515, NT05399374, NT25265908")
ngr_entry = tk.Entry(clean_tab, textvariable=ngr_entry_var)
submit_button = tk.Button(clean_tab, text="Submit", command=drop_ngr_values)
confirm_button = tk.Button(clean_tab, text="Confirm", state=tk.NORMAL, command=confirm_drop_rows)

# Layout the widgets for NGR values
ngr_label.grid(row=4, column=0, padx=10, pady=10)
ngr_entry.grid(row=4, column=1, padx=10, pady=10)
submit_button.grid(row=4, column=2, padx=10, pady=10)
confirm_button.grid(row=4, column=3, padx=10, pady=10)

# Enable the Confirm button
confirm_button.config(state=tk.NORMAL)
```

Figure 5: Code for GUI

```python
# Function to drop the specified NGR values
def drop_ngr_values():
    global df

    ngr_input = ngr_entry.get().strip()
    if ngr_input:
        drop_ngr_values = [ngr.strip() for ngr in ngr_input.split(',')]
    else:
        drop_ngr_values = ['NZ02553847', 'SE213515', 'NT05399374', 'NT25265908']

    # Filter and display the rows with the specified NGR values
    rows_with_ngr = df[df['NGR'].isin(drop_ngr_values)]

    # Clear the output text area and display the new data
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, rows_with_ngr.to_string(index=False))
    output_text.insert(tk.END, f"\n\nNumber of Rows: {rows_with_ngr.shape[0]}\nNumber of Columns: {rows_with_ngr.shap
    output_text.config(state=tk.DISABLED)

    # Enable the confirm button
    confirm_button.config(state=tk.NORMAL)
```

```python
Function to confirm and drop the rows with the specified NGR values
f confirm_drop_rows():
  global df

  ngr_input = ngr_entry.get().strip()
  if ngr_input:
      drop_ngr_values = [ngr.strip() for ngr in ngr_input.split(',')]
  else:
      drop_ngr_values = ['NZ02553847', 'SE213515', 'NT05399374', 'NT252675908']

  # Drop the rows with the specified NGR values
  df = df[~df['NGR'].isin(drop_ngr_values)]

  # Disable the confirm button
  confirm_button.config(state=tk.DISABLED)

   # Disable the confirm button
  confirm_button.config(state=tk.DISABLED)

  # Clear the output text area and display the cleaned data
  output_text.config(state=tk.NORMAL)
  output_text.delete("1.0", tk.END)
  output_text.insert(tk.END, df.head(10).to_string(index=False))
  output_text.insert(tk.END, f"\n\nNumber of Rows: {df.shape[0]}\nNumber of Columns: {df.shape[1]}")
  output_text.config(state=tk.DISABLED)

  # Provide feedback to the user
  status_label.config(text="Rows dropped successfully! Data cleaning is complete! You can now move on to Reshaping th
```

Figure 6: Code showing filtering

Statistical Analysis

```python
# Function to calculate and display ERP statistics
def erp_statistics():
    global df

    # Check if data is available and new_df is created
    if df is None:
        messagebox.showerror("Error", "No data available. Please load data first.")
        return

    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    selected_columns = ['id','EID', 'NGR', 'Site', 'Site Height', 'In-Use Ae Ht', 'In-Use ERP Total']
    new_df = selected_rows[selected_columns].copy()

    # Rename columns
    new_df.rename(columns={'In-Use Ae Ht': 'Aerial height(m)', 'In-Use ERP Total': 'Power(kW)'}, inplace=True)

    # Calculate mean, mode, and median for 'Power(kW)'
    mean_erp = new_df['Power(kW)'].mean()
    mode_erp = new_df['Power(kW)'].mode().iloc[0]
    median_erp = new_df['Power(kW)'].median()

    # Display the calculated statistics in the output text area
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, f"Statistics for Power(kW):\n")
    output_text.insert(tk.END, f"Mean Power(kW): {mean_erp:.2f}\n")
    output_text.insert(tk.END, f"Mode Power(kW): {mode_erp:.2f}\n")
    output_text.insert(tk.END, f"Median Power(kW): {median_erp:.2f}\n")
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="ERP statistics calculated and displayed successfully!")
```

```python
# Function to calculate and display Site Height statistics
def site_height_statistics():
    global df

    # Check if data is available and new_df is created
    if df is None:
        messagebox.showerror("Error", "No data available. Please load data first.")
        return

    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    selected_columns = ['id','EID', 'NGR', 'Site', 'Site Height', 'In-Use Ae Ht', 'In-Use ERP Total']
    new_df = selected_rows[selected_columns].copy()

    # Rename columns
    new_df.rename(columns={'In-Use Ae Ht': 'Aerial height(m)', 'In-Use ERP Total': 'Power(kW)'}, inplace=True)

    # Filter new_df for Site Height > 75
    filtered_df = new_df[new_df['Site Height'] > 75]

    # Calculate mean, mode, and median for Site Height
    mean_site_height = filtered_df['Site Height'].mean()
    mode_site_height = filtered_df['Site Height'].mode().iloc[0]
    median_site_height = filtered_df['Site Height'].median()

    # Display the calculated statistics in the output text area
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, f"Statistics for Site Height > 75:\n")
    output_text.insert(tk.END, f"Mean Site Height: {mean_site_height:.2f}\n")
    output_text.insert(tk.END, f"Mode Site Height: {mode_site_height:.2f}\n")
    output_text.insert(tk.END, f"Median Site Height: {median_site_height:.2f}\n")
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="Site Height statistics calculated and displayed successfully!")
```

```python
# Function to calculate and display Date statistics
def date_statistics():
    global df

    # Check if data is available and new_df is created
    if df is None:
        messagebox.showerror("Error", "No data available. Please load data first.")
        return

    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    selected_columns = ['id', 'Date','EID', 'NGR', 'Site', 'Site Height', 'In-Use Ae Ht', 'In-Use ERP Total']
    new_df = selected_rows[selected_columns].copy()

    # Rename columns
    new_df.rename(columns={'In-Use Ae Ht': 'Aerial height(m)', 'In-Use ERP Total': 'Power(kW)'}, inplace=True)

    # Convert 'Date' column to datetime
    new_df['Date'] = pd.to_datetime(new_df['Date'], dayfirst=True)

    # Filter new_df for Date from 2001 onwards
    filtered_df = new_df[new_df['Date'].dt.year >= 2001]

    # Calculate mean, mode, and median for Date
    mean_date = filtered_df['Date'].mean()
    mode_date = filtered_df['Date'].mode().iloc[0]
    median_date = filtered_df['Date'].median()

    # Display the calculated statistics in the output text area
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, f"Statistics for Date from 2001 onwards:\n")
    output_text.insert(tk.END, f"Mean Date: {mean_date}\n")
    output_text.insert(tk.END, f"Mode Date: {mode_date}\n")
    output_text.insert(tk.END, f"Median Date: {median_date}\n")
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="Date statistics calculated and displayed successfully!")
```

Figure 7: Statistical Calculations: Mean, Mode, Median

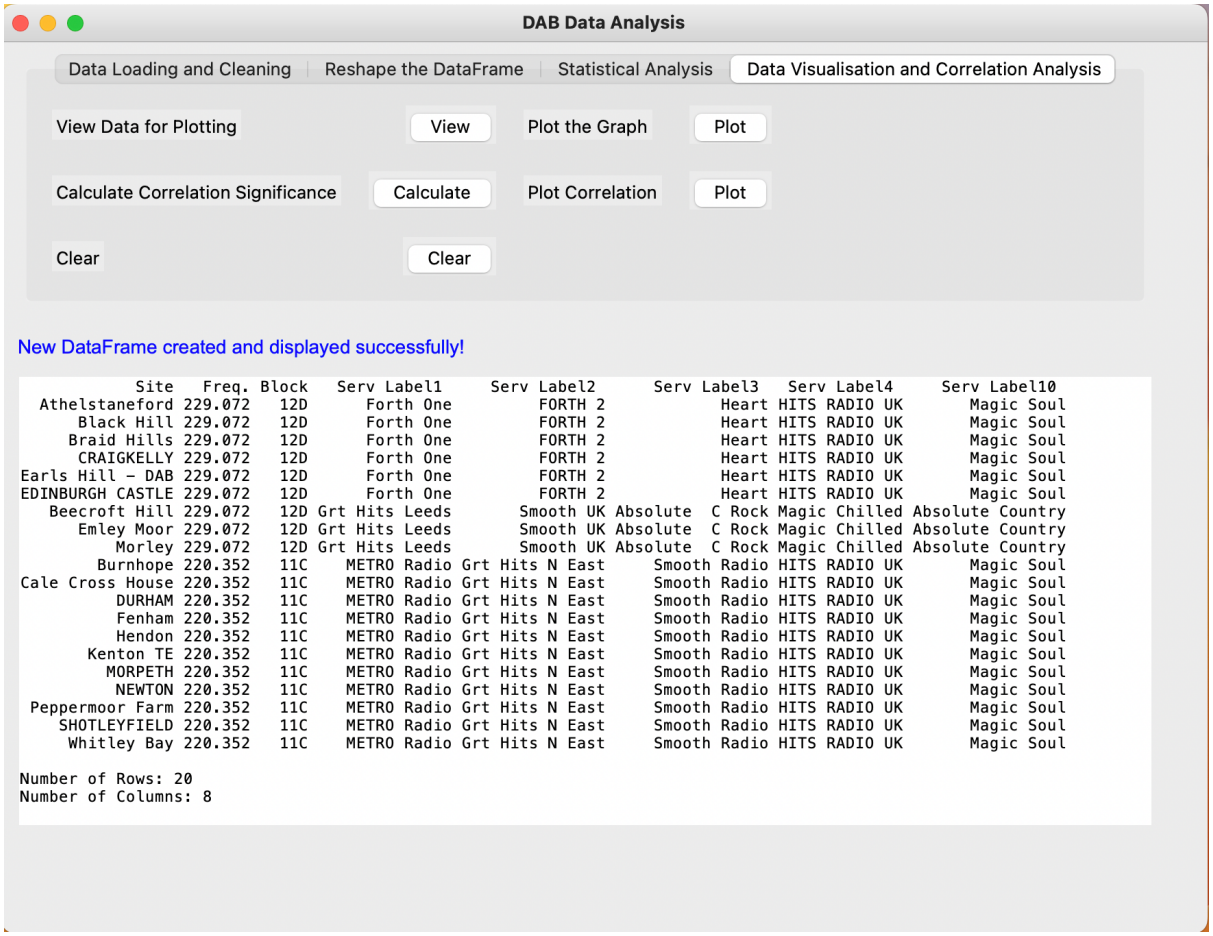

Figure 8: New DataFrame with Site, Freq, Block and the various Serv Labels

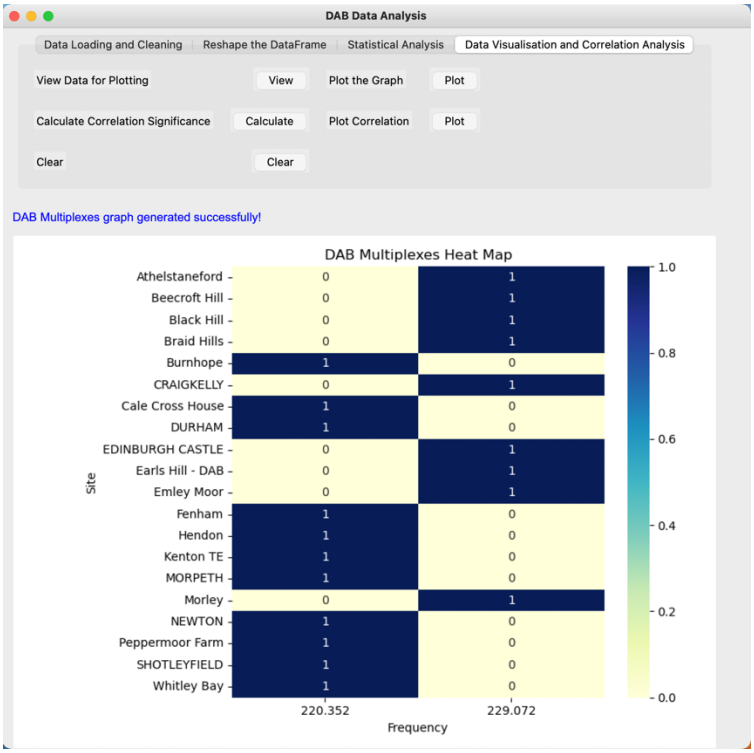

Figure 9: DAB Multiplex Visualisation

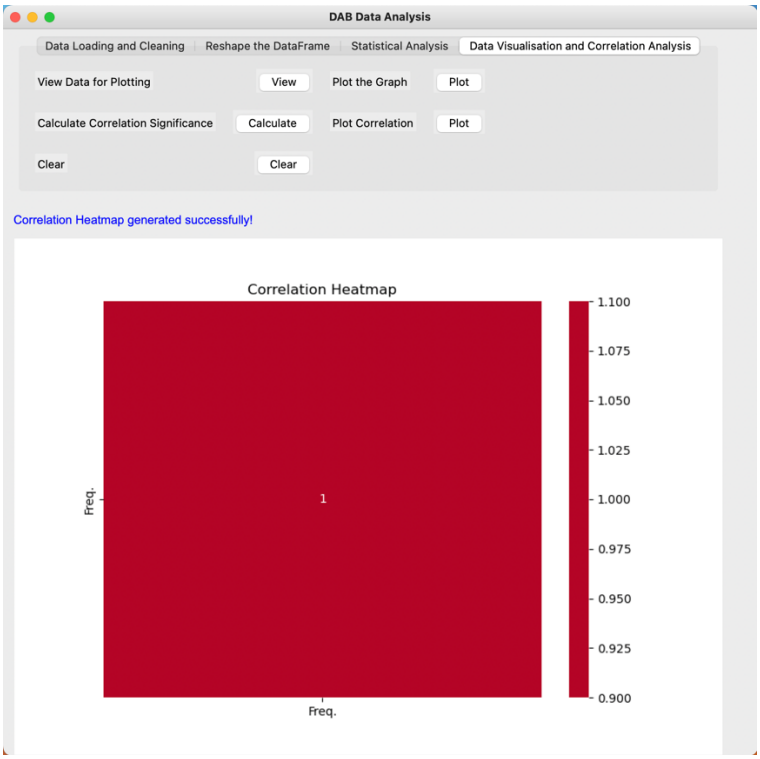

Figure 10: Correlation Analysis Heatmap

```
# Check the python version
!python --version
```

```
Python 3.10.9
```

```
import pandas as pd

import numpy as np

import scipy.stats

import seaborn as sns

#File input/output
import os

import time

#JSON file handling
import json

#Data visualisation
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

#GUI
import tkinter as tk
from tkinter import ttk
from tkinter import *
from tkinter import filedialog
from tkinter import messagebox
from tkinter.filedialog import asksaveasfile
from tkinter.filedialog import asksaveasfilename

#Date and Time
from datetime import datetime
```

Figure 11: Checking Python version and Importing Libraries

Statistical Analysis

```python
# Function to calculate and display ERP statistics
def erp_statistics():
    global df

    # Check if data is available and new_df is created
    if df is None:
        messagebox.showerror("Error", "No data available. Please load data first.")
        return

    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    selected_columns = ['id','EID', 'NGR', 'Site', 'Site Height', 'In-Use Ae Ht', 'In-Use ERP Total']
    new_df = selected_rows[selected_columns].copy()

    # Rename columns
    new_df.rename(columns={'In-Use Ae Ht': 'Aerial height(m)', 'In-Use ERP Total': 'Power(kW)'}, inplace=True)

    # Calculate mean, mode, and median for 'Power(kW)'
    mean_erp = new_df['Power(kW)'].mean()
    mode_erp = new_df['Power(kW)'].mode().iloc[0]
    median_erp = new_df['Power(kW)'].median()

    # Display the calculated statistics in the output text area
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, f"Statistics for Power(kW):\n")
    output_text.insert(tk.END, f"Mean Power(kW): {mean_erp:.2f}\n")
    output_text.insert(tk.END, f"Mode Power(kW): {mode_erp:.2f}\n")
    output_text.insert(tk.END, f"Median Power(kW): {median_erp:.2f}\n")
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="ERP statistics calculated and displayed successfully!")
```

Figure 12: Code to Calculate ERP Statistics

Reshaping Data

```python
# Function to extract DAB multiplex information from 'EID' column
def get_dab_info(eid_value, multiplex):
    if isinstance(eid_value, str) and multiplex in eid_value:
        split_values = eid_value.split(',')
        index = dab_multiplexes.index(multiplex)
        if index < len(split_values):
            return split_values[index].strip()
    return ''
```

```python
# Function to create new columns for DAB multiplexes
def create_dab_columns():
    global df, dab_multiplexes

    # Get user input for DAB multiplex columns
    dab_columns_input = dab_multiplex_entry.get().strip()
    if dab_columns_input:
        dab_columns = [column.strip() for column in dab_columns_input.split(',')]
    else:
        dab_columns = dab_multiplexes

    # Extract DAB multiplex information from 'EID' column to new columns
    for column in dab_columns:
        df[column] = df['EID'].apply(lambda x: get_dab_info(x, column))

     # Clear the output text area and display the cleaned data
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, df.head(10).to_string(index=False))
    output_text.insert(tk.END, f"\n\nNumber of Rows: {df.shape[0]}\nNumber of Columns: {df.shape[1]}")
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="DAB multiplex columns created successfully!")
```

Figure 13: Code to Extract and Add DAB multiplexes (C18A, C18F, C188) columns

```
# Visualisation
```

```
# Labels
view_data_for_plot_label = tk.Label(visualise_tab, text="View Data for Plotting")
plot_label = tk.Label(visualise_tab, text="Plot the Graph")
calculate_correlation_label = tk.Label(visualise_tab, text="Calculate Correlation Significance")
correlation_label = tk.Label(visualise_tab, text="Plot Correlation")
clear_label = tk.Label(visualise_tab, text="Clear")

# Buttons
view_data_for_plot_button = tk.Button(visualise_tab, text="View", command=view_dab_multiplexes)
plot_button = tk.Button(visualise_tab, text="Plot", command=plot_dab_multiplexes)
calculate_correlation_button = tk.Button(visualise_tab, text="Calculate", command=correlation_significance)
correlation_button = tk.Button(visualise_tab, text="Plot", command=plot_correlation_heatmap)
clear_button = tk.Button(visualise_tab, text="Clear", command=clear_output_area)

# Layout
view_data_for_plot_label.grid(row=2, column=0, padx=10, pady=10, sticky="w")
view_data_for_plot_button.grid(row=2, column=1, padx=10, pady=10, sticky="e")

plot_label.grid(row=2, column=4, padx=10, pady=10, sticky="w")
plot_button.grid(row=2, column=5, padx=10, pady=10, sticky="e")

calculate_correlation_label.grid(row=3, column=0, padx=10, pady=10, sticky="w")
calculate_correlation_button.grid(row=3, column=1, padx=10, pady=10, sticky="e")

correlation_label.grid(row=3, column=4, padx=10, pady=10, sticky="w")
correlation_button.grid(row=3, column=5, padx=10, pady=10, sticky="e")

clear_label.grid(row=4, column=0, padx=10, pady=10, sticky="w")
clear_button.grid(row=4, column=1, padx=10, pady=10, sticky="e")
```

```
# Output text area to display data
output_text = tk.Text(root, wrap="none", height=25, state=tk.DISABLED)
output_text.grid(row=6, column=0, columnspan=6, padx=10, pady=5, sticky="we")

# Status label
status_label = tk.Label(root, text="", bd=1, anchor=tk.W)
status_label.grid(row=5, column=0, columnspan=6, padx=10, pady=5, sticky="we")

# Styling
status_label.config(font=("Arial", 14), fg="blue")
#output_text.config(font=("Helvetica", 10))
```

```
root.mainloop()
```

Figure 14: GUI Code for the Visualisation Tab

```python
def correlation_significance():
    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    new_columns = ['Freq.', 'Block', 'Serv Label1 ', 'Serv Label2 ', 'Serv Label3 ', 'Serv Label4 ', 'Serv Label10 ']
    new_df = selected_rows[new_columns].copy()

    # Calculate the correlation matrix with p-values
    correlation_matrix = new_df.corr(method='pearson', numeric_only=True)

    # Calculate the p-values for correlations
    p_values = new_df.corr(method=lambda x, y: np.round(scipy.stats.pearsonr(x, y)[1]), numeric_only=True)

    # Create a mask for significant correlations
    significance_level = 0.5
    significance_mask = p_values < significance_level

    # Update the correlation matrix with non-significant correlations masked out
    correlation_matrix[~significance_mask] = 0

    # Convert the correlation matrix to a formatted string
    correlation_text = correlation_matrix.to_string(float_format="{:.2f}".format)

    # Determine if there is any significant correlation
    is_significant = significance_mask.any().any()

    # Update the output text area with the correlation information
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    if is_significant:
        output_text.insert(tk.END, "There is significant correlation between the selected columns.\n")
    else:
        output_text.insert(tk.END, "There is no significant correlation between the selected columns.\n")

    # Clear the output area
    clear_output_area()

    # Display the correlation information
    output_text.insert(tk.END, correlation_text)
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="Correlation analysis completed!")
```

Figure 15: Code to calculate Correlation Significance

```python
# Visualisation of the correlation significance using a heatmap
def plot_correlation_heatmap():

    # Clear the output area
    clear_output_area()

    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    new_columns = ['Site', 'Freq.', 'Block', 'Serv Label1 ', 'Serv Label2 ', 'Serv Label3 ', 'Serv Label4 ', 'Serv Label10 ']
    new_df = selected_rows[new_columns].copy()

    # Calculate the correlation matrix
    correlation_matrix = new_df.corr()

    # Create a heatmap using seaborn
    plt.figure(figsize=(6, 6))
    sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", center=0)
    plt.title("Correlation Heatmap")

    # Display the plot in the GUI
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)  # Clear previous content
    canvas = FigureCanvasTkAgg(plt.gcf(), master=output_text)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="Correlation Heatmap generated successfully!")
```

```python
# Function to plot the data to a graph (Heatmap)
def plot_dab_multiplexes():

    # Clear the output area
    clear_output_area()

    # Filter the DataFrame to get rows with specified EID values
    selected_rows = df[df['EID'].isin(dab_multiplexes)]

    # Create a new DataFrame with desired columns
    new_columns = ['Site', 'Freq.', 'Block', 'Serv Label1 ', 'Serv Label2 ', 'Serv Label3 ', 'Serv Label4 ', 'Serv Label10 ']
    new_df = selected_rows[new_columns].copy()

    # Pivot the DataFrame to have 'Serv Label' as columns and 'Site' as index
    pivoted_df = new_df.pivot_table(index='Site', columns='Freq.', aggfunc='size', fill_value=0)

    # Create a heatmap using seaborn
    plt.figure(figsize=(6, 6))
    sns.heatmap(pivoted_df, cmap='YlGnBu', annot=True, fmt='d')
    plt.title("DAB Multiplexes Heat Map")
    plt.xlabel("Frequency")
    plt.ylabel("Site")
    plt.tight_layout()

   # Clear the output text area and display the plot
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    canvas = FigureCanvasTkAgg(plt.gcf(), master=output_text)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
    output_text.config(state=tk.DISABLED)

    # Provide feedback to the user
    status_label.config(text="DAB Multiplexes graph generated successfully!")
```

Figure 16: Code to plot data and display in the Output Area

Reference:

1. A.W. Roscoe, “The Theory and Practice of Concurrency”, Prentice Hall, 1998.
2. J. Palach, “Parallel Programming with Python” Packt Publishing Ltd., Jun 25, 2014.
3. Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J.D. McClain, E.R. Sayfutyarova, S. Sharma, S. Wouters, “PySCF: the Python-based Simulations of Chemistry Framework, *Wiley Interdisciplinary Reviews: Computational Molecular Science.* 2018 Jan 8 e 1340.
4. M. Rocklin, “Dask: Parallel Computation with Blocked Algorithms and Task Scheduling”, *Proceedings of the 14th Python in Science Conference,* Vol. 130, pp. 136 Austin, TX: SciPy 2015.
5. R. Erhard, H. H. Do, “Data Cleaning: Problems and Current Approaches”, *IEEE Data Eng. Bull.* 2000. Vol. 23, num 4, pp. 3-13
6. D. Beazley, “Understanding the Python Gil”, PyCON Python Conference, Atlanta Georgia, Feb 2010.pp. 1-62.
7. S. Chen, Z. Yanqing, Y. Chen, J. Gu, “fastp: An Ultra-Fast All-In-One FASTQ Preprocessor,” *Bioinformatics.* University of Oxford Press. 2018 Sep 1, vol 34, no. 17, pp. i884 – i890.
8. A. Marowka, “On Parallel Software Engineering Education Using Python,” *Education and Information Technologies,*2018, pp.357-372.
9. J. Palach, “Parallel Programming with Python,” *Packt Publishing Ltd,* 2014.
10. A. Marowka, “Python Accelerators for high-performance Computing,” *The Journal of Supercomputing,* 2018, pp. 1449 – 1460.
11. E. D. Berger, “Scalene: Scripting-language Aware Profiling for Python,” 2020.
12. T. G. Mattson, T. A. Anderson and G. Georgakoudis, “Multithreaded Parallel Programming in Python,” *Computing in Science and Engineering,* 2021, pp. 77-80.
13. N. Matloff and F. Hsu, “Tutotial on Threads Programming with Python,” *University of California,* 2007.
14. R. Eggen and M. Eggen, “Thread and Process Efficiency in Python,” *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDTA), The Steering Committee of the World Congress in Computer*

Science Computer Engineering and Applied Computing (WorldComp), 2019, pp. 32-36.

15. S. Tilkov, S. Vinoski, "Node.js: Using JavaScript to Build High-performance Network Programs," *IEEE Internet Computing,* 2010, pp. 80-83.
16. B. J. Jansen, "The Graphical User Interface," *ACM SIGCHI Bulletin. Vol 30, no. 2. 1998 pp. 22-26*
17. B. M. Bode, M. S. Gordon, "MacMolPlt: A Graphical User Interface for GAMESS." *Journal of Molecular Graphics and Modeling.* 1998 Jun 1, no. 3, pp. 133-138.
18. F. Lundh, "An Introduction to Tkinter", www.pythonware.com/library/tkinter/introduction/index.htm, 1999.
19. A. Burrell, A. C. Sodan, "Web Interface Navigation Design: Which Style of Navigation-Link Menus Do Users Prefer?," *22nd International Conference on Data Engineering Workshops (ICDEW'06),* Atlanta, GA, USA, 2006, pp. 42.
20. J. W. Shipman, "Tkinter 8.4 Reference: A GUI for Python", New Mexico Tech Computer Center. 2013. Vol. 54.
21. D. Amos, "Python GUI Programming with TKinter", Real Python. 2020.
22. M. Yamato, A. Monden, K. I. Matsumoto, K. Inoue, K. Torii, "Quick Button Selection with Eye Gazing for General GUI Environment," *International Conference on Software: Theory and Practice,* 2000 Aug. pp. 712-719
23. R. Wang, Y. Perez-Riverol, H. Hermjakob, J. A. Vizcaino, "Open Source Libraries and Frameworks for Biological Data Visualisation: A Guide for Developers," *Proteomics*. 2015 April 15, pp. 1356 – 1375.
24. A. Sunarya, S. L. Nurmika, N. Asmainah, "Evaluation Model of Students Learning Outcome Using K-Means Algorith," *Journal of Physics: Conference Series.* 2020 Mar 1. Vol. 1477, no. 2, pp. 022027
25. N. Ari and M. Ustazhanov, "Matplotlib in Python," *2014 11th International Conference on Electronics, Computer and Computation (ICECCO),* Abuja, Nigeria, 2014, pp. 1-6
26. K. Arnold, J. Gosling and D. Holmes, "The Java Programming Language," *Addison Wesley Professional.* 2005.
27. T. E. Oliphant, "Python for Scientific Computing," in *Computing in Science & Engineering,* vol. 9, no. 3, pp. 10-20, May-June 2007.

28. J. VanderPlas, “Python Data Science Handbook: Essential Tools for Working with Data”, *O’Reilly Media, Inc.* 2016.
29. Mathematical Statistics Functions, Python https://docs.python.org/3/library/statistics.html
30. W. McKinney, “Pandas: A Foundational Python Library for Data Analysis and Statistics,” *Python for High Performance and Scientific Computing*, Nov 2011. Vol. 14, pp. 1-9
31. W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference.* Vol. 445, no. 1, pp. 51-56.
32. C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith and R. Kern, “Array Programming with NumPy,” *Nature*. 2020 Sept 17, pp. 357 – 362.
33. K. Arnold, J. Gosling and D. Holmes, “The Java Programming Language”, *Addison Wesley Professional*. 2005.
34. N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, “Comparison of JSON and XML Data Interchange Formats: A Case Study,” *Caine*. 2009. Pp. 157-62
35. F. Pezoa, J. L. Reutter, M. Ugarte and D. Vrgoc, “Foundations of JSON Schema,” in *Proceedings of the 25th International Conference on World Wide Web.* 2016, pp. 263-273.
36. B. Lin, Y. Chen, X. Chen and Y. Yu, “Comparison Between JSON and XML in Applications Based on AJAX,” in *2012 International Conference on Computer Science and Service System.* 2012, pp. 1174-1177.
37. T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” 2014.
38. K. Sahoo, A. K. Samal, J. Pramanik, S. K. Pani, “Exploratory Data Analysis Using Python,” *International Journal of Innovative Technology and Exploring Engineering*. 2019 Oct, pp. 4727-35.
39. W. McKinney, “Pandas-Powerful Python Data Analysis Toolkit,” 2015.
40. M. Waskom, “Seaborn: Statistical Data Visualisation,” *The Journal of Opensource Software*. 2021.
41. E. Bisong and E. Bisong, “Matplotlib and Seaborn,” *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners.* 2019, pp. 151-165.

42. D. Gotterbarn, K. Miller and S. Rogerson, “Software Engineering Code of Ethics,” *Communications of the ACM 40*. 1997 Nov, pp. 110 – 118.
43. B. Berenbach and M. Broy, “Professional and Ethical Dilemmas in Software Engineering”, in *Computer*, vol. 42, no. 1, pp. 74-80, Jan. 2009.
44. D. Gotterbarn, “How the new Software Engineering Code of Ethics Affects You,” *IEEE Software,* Nov. 1999, pp. 58 – 64.
45. A. Rashid, J. Weckert and R. Lucas, “Software Engineering Ethics in a Digital World,” *Computer*. Vol. 42, no. 6, pp.34-41
46. I. Ozkaya, “Ethics is A Software Design Concern,” *IEEE Software,* vol. 36, pp. 4-8.
47. D. Gotterbarn, “Software Engineering Ethics,” *Encyclopedia of Software Engineering 2,* 2001, pp. 1-13.