Section 1 : **Theory supported by code samples**
Word count: 1368

1a) Python threads

Threads in python provide a convenient way[1] to run tasks that would otherwise be waiting on other resources. One specific part of the app that could benefit[2] from the use of python threads would be the data manipulation section, which could achieve faster computing by taking advantage of concurrency[3]. This would require the extraction of all UK airports from the given dataset that would then be split into small, medium and large and merged with their respective frequencies. The results could be then used to generate the mean, median and mode[4] for frequencies of large airports and for all frequencies over 100 MHz notwithstanding of airport size. Coding each of these tasks as separate functions would allow to execute these operations under different threads during different stages[5].
A preliminary stage would involve importing the data from MongoDB[6] and parsing all the airports into subcategories of small, medium and large.
An important variable to consider in this scenario is that the Python interpreter does not support true multithreading due to the Global Interpreter Lock (GIL)[7].
GIL imposes that only one thread can be in a state of execution at any point in time. Yet in the event of data input/output, or network bound tasks when the python interpreter is idle waiting for the result of a given execution in one thread, a second thread could be used to read data from another source or file[8].
This characteristic, combined with MongoDB's ability to access multiple files at the same time[9] would potentially enable the application to fetch data concurrently with the use of the threading library.
The use of threading would allow for separate and concurrent processes to be executed, thus taking advantage of multi-core properties in modern CPU's[10] to reorder the original raw data into four distinct datasets for small, medium and large airports. For this purpose a filter function could be created to parse the original dataset and produce separate sets for small, medium and large airports. Taking advantage of the "append()" function[11], new records could be added after checking for duplicates in the original dataset. In a successive stage, three processes would work concurrently to merge the small, medium and large datasets with their respective frequencies matching them on attributes "id" and "airport_ref".

1b) GUI

User interface design focuses on anticipating end users needs and requires developers to consider different variables such as learnability, efficiency, visibility and satisfaction[12].
The field of UI design draws from numerous studies related to cognitive theory and human-computer interaction[13] with a main goal of providing human-oriented access to the structures being processed by the computer program[14] [Appendix 2 – App. Diagram].
Fundamentally, the process focuses on creating interfaces that are both easy and enjoyable to use[15], by keeping user interaction as straightforward and efficient as possible.
As per client requirements the application created is not meant to be generic and therefore has limited functionalities.
The user interface was designed following principles from Peter Morville's User Experience Honeycomb[16] to resolve various challenges in creating an enjoyable user experience.
The most important aspect of an interface is to assist users to accomplish a given task (usability), and at the same time allow the user to be constantly aware of what is happening within the system (visibility) [17]. A disorderly and overly flashy interface can prove to be not only detrimental to the user, but also be both cumbersome in terms of development as well as inferior in terms of users' aesthetic perception[18].

Additionally, operability and robustness, which are also essential properties of an effective interface, are negatively impacted[19]. Based on these considerations the design of the application has been kept minimalist by categorizing the user requirement into three distinct sections. A landing window enables the user to select an initial action: selecting available CSV files, importing files from a database or exiting the program[Fig.1]. The Data Preparation window assists users in importing, converting, cleaning and saving data into a database[Fig.2,3,4]. Finally, the Data manipulation and visualization windows provide a means to view the data in graphic form[Fig.6,7].
In order to enhance efficiency and assist users in recognizing patterns as well as quickly familiarize themselves with the system[20], the design of each window has been kept identical with buttons being the only element utilized in different sizes and various activation states.
Buttons are strategically placed to draw user's attention and visibility is further improved by the flow-structure, which is kept linear , presenting the output of different actions within the same window[Fig.1-12]. UI elements such as buttons are designed to be active, inactive or highlighted according to the state that the application is in at any given moment. As an example, the save button is visible only when data manipulation tasks are complete, while the exit button materializes only after the data has been saved. Colors and texture are an exceptional medium for drawing attention to a certain element[21].
For this reason, windows have been kept neutral in color while buttons indicating either activation or deactivation have been designed with contrasting colors[Fig.4,7].
Potential errors such as "files not found" or wrong formatting are identified by the application using exception handling[22] and appropriate messages are then displayed to the user whom is kept informed throughout the process. User needs have been anticipated by setting default names to be used for storage and also files to be selected in the presence or absence of prepared data[Fig.6].

1c) Java VS Python

Python is widely considered the optimal language for development of tools within the data science industry and is also ranked first in numerous popularity indices[23]. Python provides many advantages over other languages[24] including a concise syntax[25], higher code readability[26] and a comparatively shallow learning curve ,which in turn allows junior developers to achieve an adequate level of proficiency much faster[27]. Since python is dynamically typed, programs can be written with relatively fewer lines of code[28], and it is also simpler to prototype with.
Open source web application platforms such as Jupyter Notebook[29] have increased the popularity of Python, since they allow for testing of a single cell of code at a time and facilitate the debugging process. When speaking about the role of Python as one of the premier languages for data science, we should mention its outstanding memory management capabilities[30] particularly garbage collection[31] which makes it a perfect fit for managing large volumes of data. Comparing Python to other popular languages such as Java[32] the former offers an abundance of specialized libraries[33] which can support scientists in data wrangling and manipulation. Within the client's Application, Python Pandas[34] have been used to filter[35], clean[36], load[37] convert and save data[38] as well as supply a rapid and reliable way to analyze and convert both small and large data sets. Java is another popular language among data scientists[39], as many Big Data frameworks have strong foundations in Java, which provides higher speed and productivity in large scale systems. Java also includes many specialized libraries, which can be effectively used for data analysis[40] and given that types in Java are immutable[41], a program doesn't waste time rechecking variable types during execution, which in turn boosts performance. Since Java code is compiled and then run all at once[42], with each change the entire program must run again, while in scripting languages like Python programmers can manipulate an individual line at a time[43] and then single out components that need testing which is advantageous for quick prototyping and evaluating models and hyper-parameters. As can be seen, despite Java

providing raw performance advantages, particularly in big data analysis[44], there is a necessary trade-off between the time it takes to run the code and the time necessary to write the program, which ultimately makes Python the preferred language within the industry[45]. Choosing one over the other will depend upon specific project requirements, if speed is the main criterion Java may provide some benefits[46] however overall Python offers more advantages in terms of flexibility and ease of deployment.

Section 2 : **Design decisions supported by code samples**

Word count: 848

2 a)

JSON[47] was selected as the optimal data format in the development of the application for a number of reasons[48]. Apart from offering a high level of intelligibility[49] for human readers JSON is also easy to parse for computers[50] and is well integrated with Python which provides great support with the JSON library[51]. We have the ability to convert lists and dictionaries to JSON, and also convert strings to lists and dictionaries[52] as JSON data is very similar to a dictionary in Python, with keys and values stored[53]. In the client's prototype application, CSV files are initially transformed into Panda data frames[54], [Code 1] which are successively converted into JSON files[55][Code 2] and finally, loaded into a database[56][Code-3]. An important benefit of using databases with JSON models is the dynamic and flexible schema they provide when compared to the tabular data models used by relational databases[57]. This has in turn made the choice to use MongoDB a natural one, since JSON is well integrated within MongoDB[58]. MongoDB in fact utilizes a modified JSON format called BSON[59].
BSON is essentially a binary representation of the former that encodes type and length information, thereby allowing data to be parsed much faster. Anything you can represent in JSON can be natively stored in MongoDB[60], and retrieved just as easily when using BSON. From the client's perspective, JSON grants many benefits such as a simpler syntax[61], which contributes in reducing overhead in data transfers[62], and offers a wide range of browser compatibility across different operating systems[63]. In terms of pure performance, JSON is quite fast and consumes less memory space[64], which makes it a suitable choice for large object graphs and systems[65].

2 b)

The data cleaning process starts with the "cleaned_df ", function which removes null values from the dataset and preserves both the "type" and "iso_country" columns that will subsequently be necessary for further data manipulation[Code-4]. Python if statements and for loops are used to iterate the process of removing null values from the "type" and "iso_country" attributes[Code-4]. There are two distinct functions created with the purpose of manipulating data to generate mean median and mode. The "summary_Large" function[Code-5] calculates and displays the mean, median , and mode of the input dataframe[66], with the "np.round"[67] function used to round an array to the given number of decimals and "np.mean"[68] calculating the arithmetic mean. Similarly, the "summary_Freq_100" function[Code-6] calculates the mean, median and mode for frequencies lesser than 100 MHz, first creating a text widget with Tkinter, merging all pandaframes[69] to obtain frequencies lower than 100 MHz and operating statistical analysis[70] on all large UK airports[Code-6]. The mean provides a general description of the data and how well or poorly the mean describes a sample depends on how spread the data is[71]. Within the available dataset for large UK airports, the mean, median and mode are comprised between values of 120.01 and 122.1 [screenshot] while mean and median across all airports are reported to be slightly higher with the former having a frequency of 149.4 and the latter of 124.4.

2 c)

The visualization tab at the top of the screen presents two buttons to display histograms for either “small UK airports” or “all airports”[Fig.11]. The choice of histograms over other charting tools was dictated by a number of factors[72]. Histograms are one of the most popular and commonly used devices for charting continuous frequency distribution[73]. A single glance at a histogram gives us some idea about the shape and spread of the data. Specifically, in the case of frequencies for small airports, we can see that the vast majority are between 100-150 MHz with only a few outliers[Fig.11]. Python offers a handful of different options for building and plotting histograms[74]. Theoretically, it is perfectly acceptable to create a histogram in python by providing an array of values and using a “for loop” to plot the output in histogram form with the standard “collection.counter()” subclass[75].

Despite being fairly straightforward, there are numerous libraries that can streamline this process further. Choosing a specific approach depends on the nature of the data to be analyzed. For large arrays of data where there is a need to compute a mathematical histogram that represents bins and their corresponding frequencies ,Numpy's “np.histogram()” and “np.bincount()”[76] can be viable options to compute the histogram values numerically and their corresponding bin edges[Code-7].

Due to the fact that the application works directly using Pandaframes, “pyplot.hist()”[77] was chosen as a histogram plotting function that uses “np.histogram()”[78] and is the basis for Pandas’ plotting functions[Code-8]. Matplotlib also provides the functionality to visualize Python histograms out of the box with a versatile wrapper around NumPy’s “histogram()” function[79]. The “plotFreqSmall” function plots communication frequencies used by small UK airports,by first defining a figure that will hold the plot, adding a subplot and subsequently plotting the graph with the “.hist()” function[80] [Code-8].

2 d)

The “plotCorrAll” function[Code 9] makes use of “matplotlib.figure”[81] in conjunction with the “.hist()”[82] functions to create histograms. A figure to frame the plot and a subplot are created using Tkinter[83], graphics are intentionally kept simple and linear to provide the user with a more immediate representation of the data. The hist() function uses an array of numbers to create a histogram where the array is sent into the function as an argument[84]. Individual labels for “small” “large” and “medium” airports are set with an alpha value of 0.5 to specify the transparency of each histogram[Code 9]. The histogram for all airports offers the users a more complete view of the dataset[Fig.12], again we can see that the greatest majority across all airports will have frequencies within a given range[Fig.12], with a smaller section displaying higher values[Fig.13-I and II]. It is noteworthy to note that almost exclusively medium airports report higher frequencies in ranges higher than 200 mhz[Fig.13-II].

Section 3 : **Reflections on the moral, ethic and legal aspects**

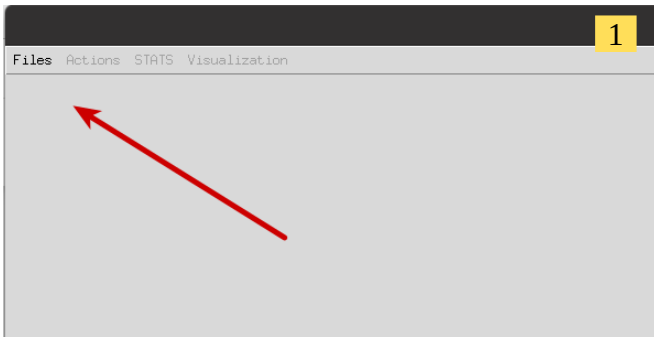
Word count : 458

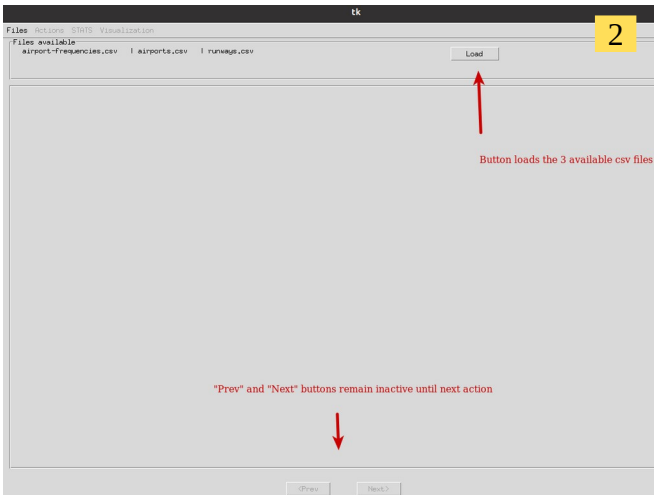
1 a)

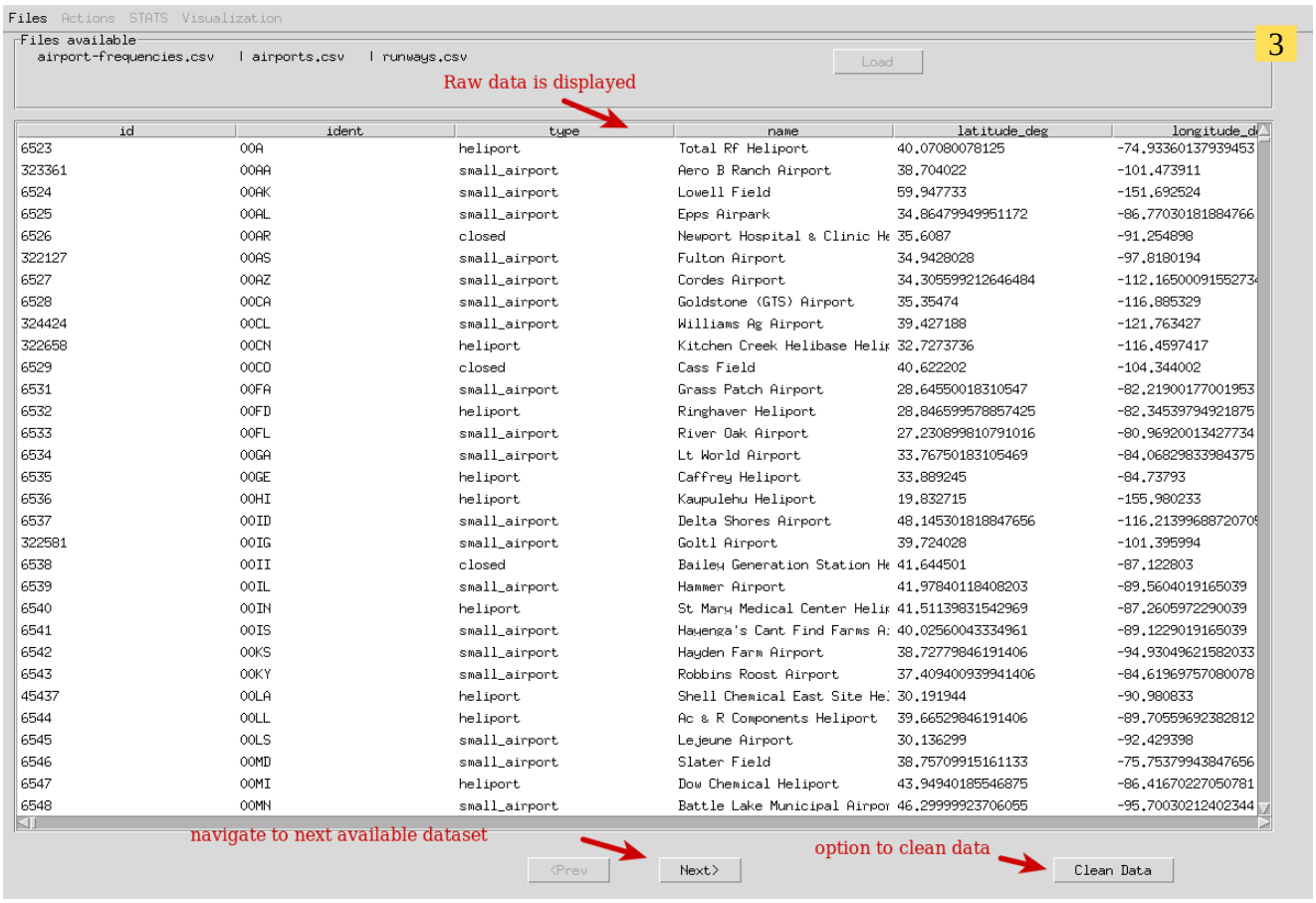
The use of software has become pervasive in modern societies and is an integral part of our daily lives[85], this in turn has motivated software developers to start pondering ethical aspects of their craft[86]. In this day and age software development cannot prescind from both moral and ethical considerations[87]. From the way private corporations handle user log files and personal data[88] to the importance of data protection and encryption[89] the industry had largely ignored such issues until very recently when a number of international scandals brought these questions to the public’s attention[90]. An example of this the 2015 so-called “Dieselgate”[91] controversy in which the German automobile giants Volkswagen and Audi cheated on clean air regulations through the use of software which made

the vehicle’s emissions seem cleaner then they were in reality, thereby proving the point that alterations to a single line of code can have catastrophic effects both in financial and environmental terms[92]. The advent of AI-based, truly self-driving cars poses numerous ethical challenges to the developers responsible for creating systems that not only need to be reliable and efficient, but also replicate ethical and moral considerations undertaken by human drivers[93]. Hans Moravec[94] was one of the first scientist to postulate that it is comparatively easier to have computers exhibit near-human levels of intelligence in strategic games like checkers, but nearly impossible to provide them with the perception and mobility skills of a one-year-old toddler. Devising ways to artificially engineer conscience is indeed one of the most monumental tasks facing software development[95]. One might argue that humans are capable of unethical or even nefarious choices when driving and that AI-powered vehicles might in fact be safer[96], but most people are still reluctant to hand over their capacity for decision making to a piece of software, especially in consideration of the fact this might entail life and death choices[97]. Finding effective ways to combat algorithm bias[98] is another key issue; software is written by people and personal bias will almost always filter through to a certain extent. One way to address this might be to remove variables associated with gender, race and sexual orientation or even creating ad-hoc programs[99] that target biased variables from a given data-set, despite the fact both approaches might have a negative effect on performance[100]. In time, the software development community will have to resolve these issues and devise better practices and standards to do so[101]. Developers are often reluctant to embrace a cooperative approach and are averse to any form of centralized oversight[102], citing the need for independence as essential to successful operations[103]. Micromanaging single developers and adding unnecessary bureaucracy can certainly have a negative effect[104], but nonetheless a certain level of coordination and oversight is necessary to guarantee product quality and curtail security risks[105]. A less formal management style based on agile software development practices[106] and the adoption of distributed version control systems such as GIT[107] could provide developers with more independence while maintaining management’s control over the project.

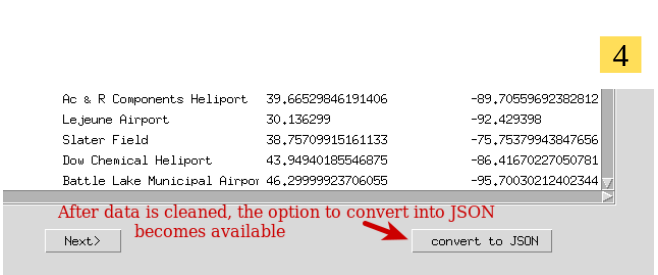
Appendix 1 – Application screenshots

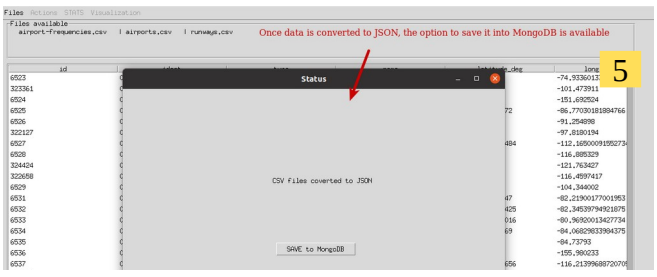


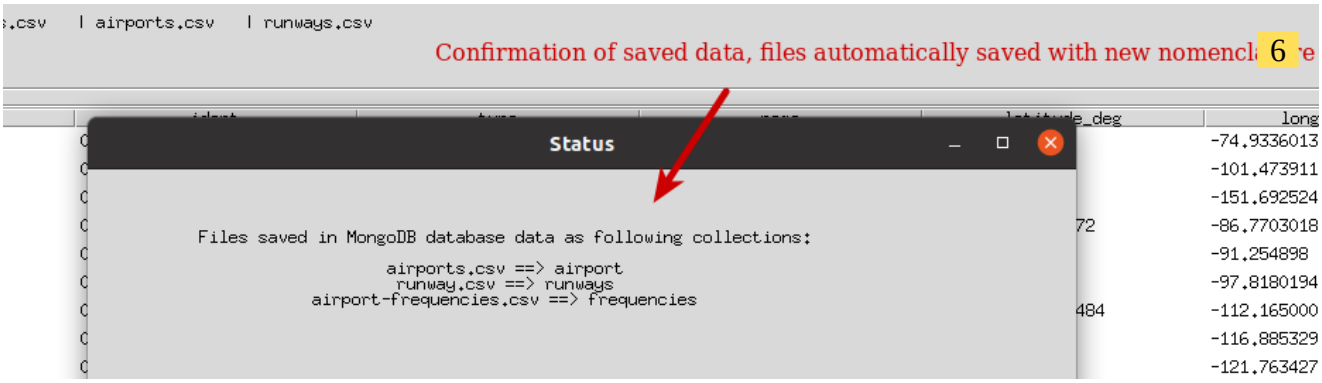


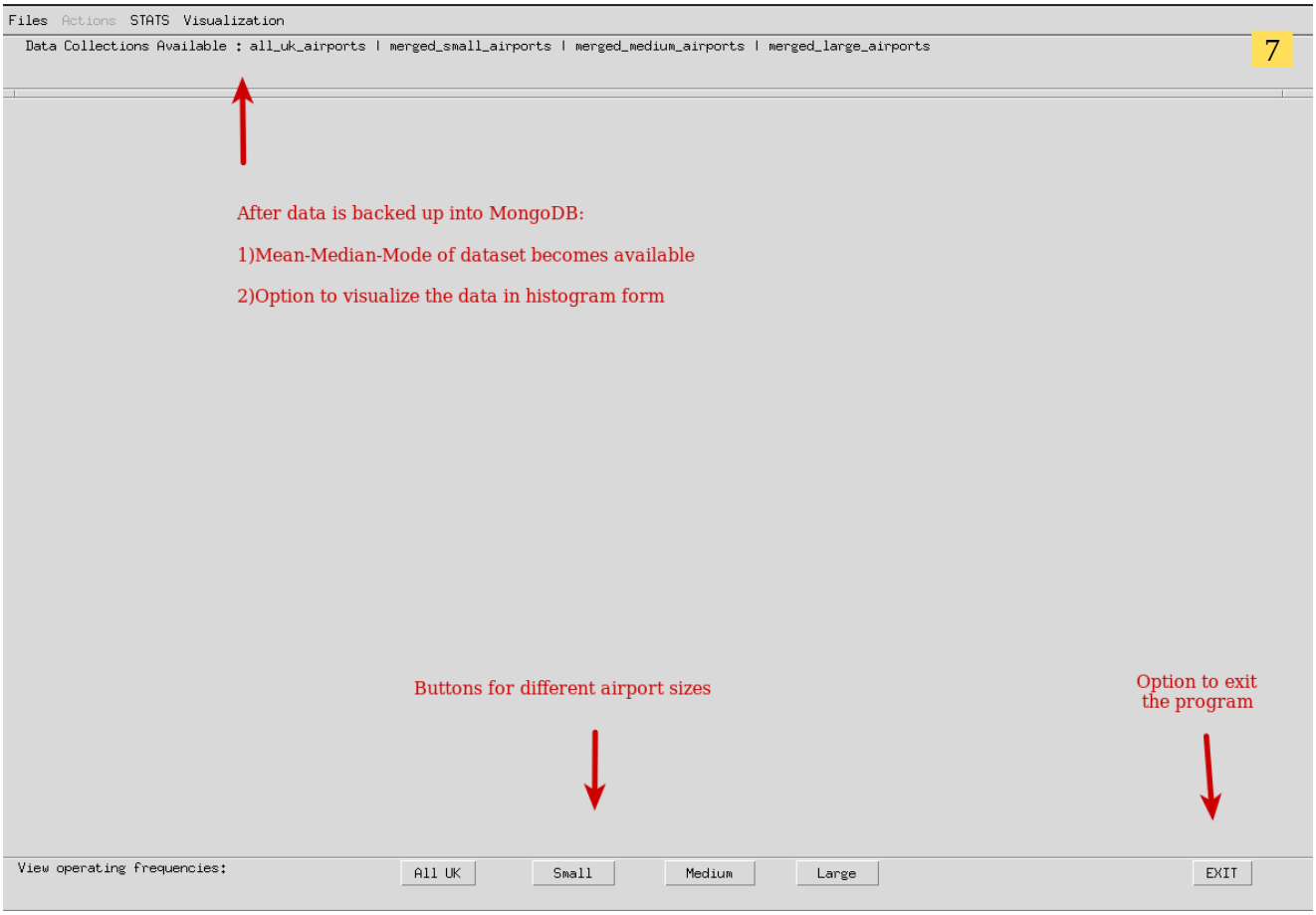


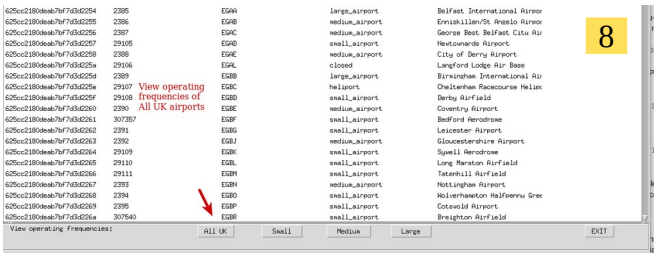
id	ident	type	name	latitude_deg	longitude_d
6523	00A	heliport	Total Rf Heliport	40.07080078125	-74.93360137939453
323361	00AA	small_airport	Aero B Ranch Airport	38.704022	-101.473911
6524	00AK	small_airport	Lowell Field	59.947733	-151.692524
6525	00AL	small_airport	Epps Airpark	34.86479949951172	-86.77030181884766
6526	00AR	closed	Newport Hospital & Clinic He	35.6087	-91.254898
322127	00AS	small_airport	Fulton Airport	34.9428028	-97.8180194
6527	00AZ	small_airport	Cordes Airport	34.305599212646484	-112.1650009155273
6528	00CA	small_airport	Goldstone (GTS) Airport	35.35474	-116.885329
324424	00CL	small_airport	Williams Ag Airport	39.427188	-121.763427
322658	00CN	heliport	Kitchen Creek Helibase Helip	32.7273736	-116.4597417
6529	00CO	closed	Cass Field	40.622202	-104.344002
6531	00FA	small_airport	Grass Patch Airport	28.64550018310547	-82.21900177001953
6532	00FD	heliport	Ringhaver Heliport	28.846599578857425	-82.34539794921875
6533	00FL	small_airport	River Oak Airport	27.230899810791016	-80.96920013427734
6534	00GA	small_airport	Lt World Airport	33.76750183105469	-84.06829833984375
6535	00GE	heliport	Caffrey Heliport	33.889245	-84.73793
6536	00HI	heliport	Kaupulehu Heliport	19.832715	-155.980233
6537	00ID	small_airport	Delta Shores Airport	48.145301818847656	-116.21399688720705
322581	00IG	small_airport	Goltl Airport	39.724028	-101.395994
6538	00II	closed	Bailey Generation Station He	41.644501	-87.122803
6539	00IL	small_airport	Hammer Airport	41.97840118408203	-89.5604019165039
6540	00IN	heliport	St Mary Medical Center Helip	41.51139831542969	-87.26059722900039
6541	00IS	small_airport	Hayenga's Cant Find Farms A:	40.02560043334961	-89.1229019165039
6542	00KS	small_airport	Hayden Farm Airport	38.72779846191406	-94.93049621582033
6543	00KY	small_airport	Robbins Roost Airport	37.409400939941406	-84.61969757080078
45437	00LA	heliport	Shell Chemical East Site He:	30.191944	-90.980833
6544	00LL	heliport	Ac & R Components Heliport	39.66529846191406	-89.70559692382812
6545	00LS	small_airport	Lejeune Airport	30.136299	-92.429398
6546	00MD	small_airport	Slater Field	38.75709915161133	-75.75379943847656
6547	00MI	heliport	Dow Chemical Heliport	43.94940185546875	-86.41670227050781
6548	00MN	small_airport	Battle Lake Municipal Airpor	46.29999923706055	-95.70030212402344

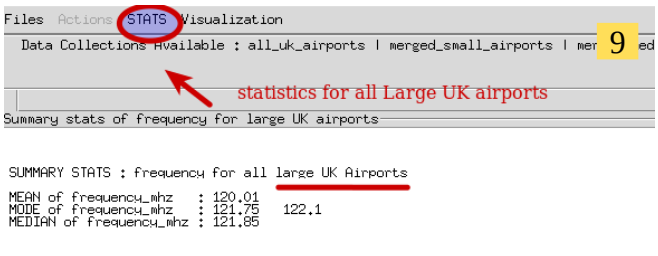





.csv | airports.csv | runways.csv
Confirmation of saved data, files automatically saved with new nomencl
6
Status
Files saved in MongoDB database data as following collections:
airports.csv ==> airport
runway.csv ==> runways
airport-frequencies.csv ==> frequencies
latitude_deg
long
-74.9336013
-101.473911
-151.692524
-86.7703018
-91.254898
-97.8180194
-112.165000
-116.885329
-121.763427


Files Actions STATS Visualization
Data Collections Available : all_uk_airports | merged_small_airports | merged_medium_airports | merged_large_airports
7
After data is backed up into MongoDB:
1)Mean-Median-Mode of dataset becomes available
2)Option to visualize the data in histogram form
Buttons for different airport sizes
Option to exit the program
View operating frequencies:
All UK
Small
Medium
Large
EXIT


8
View operating frequencies of All UK airports


Files Actions STATS Visualization
9
Data Collections Available : all_uk_airports | merged_small_airports
statistics for all Large UK airports
Summary stats of frequency for large UK airports
SUMMARY STATS : frequency for all large UK Airports
MEAN of frequency_mhz : 120.01
MODE of frequency_mhz : 121.75 122.1
MEDIAN of frequency_mhz : 121.85

Files Actions STATS Visualization

Data Collections Available : all_uk_airports | merged_small_airports | merged_medium_airports | merged_large_airports

10

Summary stats of frequency > 100Mhz for all UK airports

SUMMARY STATS : All UK Airports with frequency >100 Mhz

MEAN of frequency_mhz : 149.24
MODE of frequency_mhz : 122.1
MEDIAN of frequency_mhz : 124.4

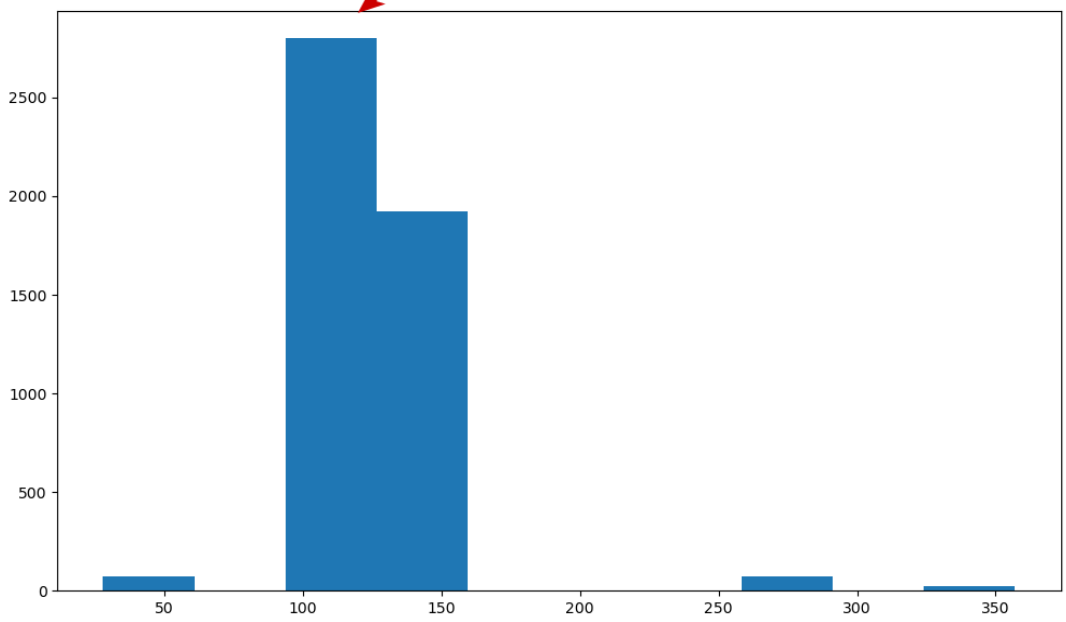
Stats for all airports with frquencies greater than 100 MHz

Files Actions STATS Visualization

Data Collections Available : all_uk_airports | merged_small_airports | merged_medium_airports | merged_large_airports

11

Histogram for frequencies of small airports

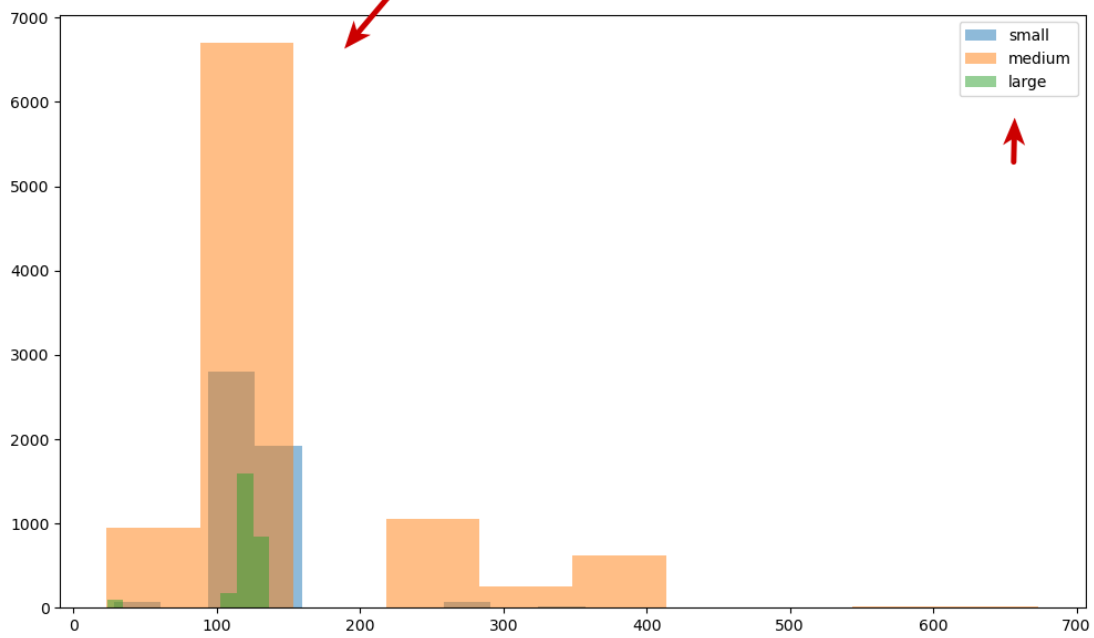


Files Actions STATS Visualization

Data Collections Available : all_uk_airports | merged_small_airports | merged_medium_airports | merged_large_airports

12

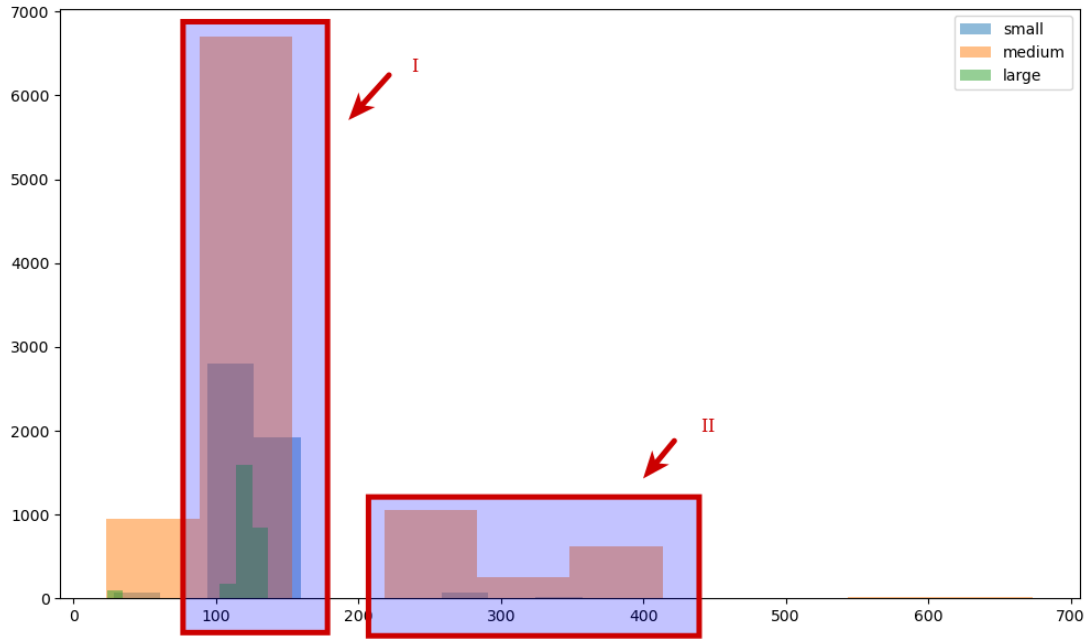
Histogram for all frequencies to visualize correlation



Files Actions STATS Visualization

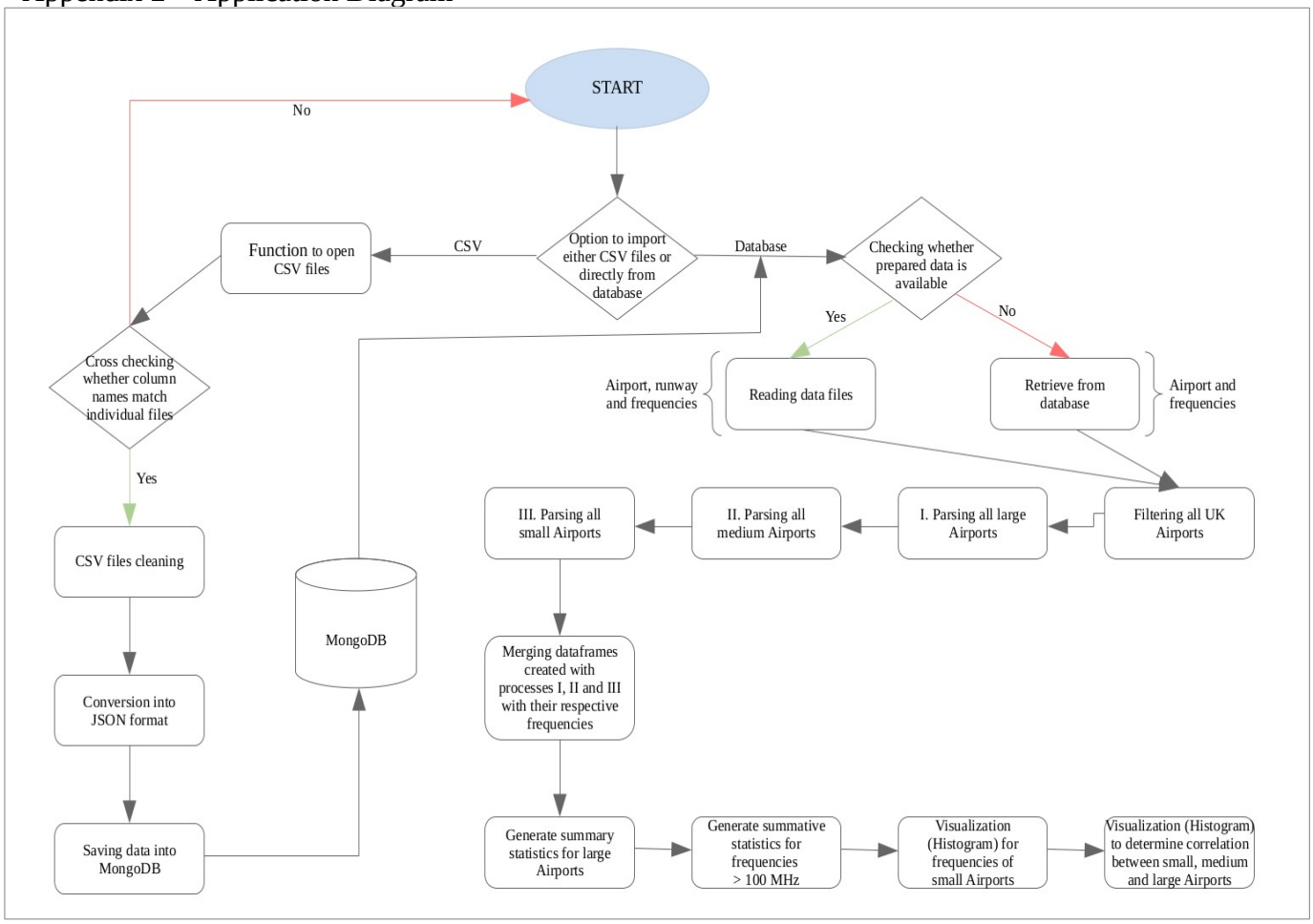
Data Collections Available : all_uk_airports | merged_small_airports | merged_medium_airports | merged_large_airports

13



View operating frequencies: All UK | Small | Medium | Large | EXIT

Appendix 2 – Application Diagram



Appendix 3 – Code Samples

1

```
        #Converting pandaframes to JSON format
        all_uk_json = self.df_airport.to_json(default_handler = str, orient = 'records', indent = 4)
        large_json = self.df_airport_large.to_json(default_handler = str, orient = 'records', indent = 4)
        medium_json = self.df_airport_medium.to_json(default_handler = str, orient = 'records', indent = 4)
        small_json = self.df_airport_small.to_json(default_handler = str, orient = 'records', indent = 4)
```

2

```
    #Function to convert available CSV files into JSON format
    def csvtoJson(self):

        #Construct a top window to input Database name, collection names for MongoDB
        self.cleanbtn.configure(state = DISABLED)
        self.top = tk.Toplevel(self.root)
        self.top.geometry("700x400")
        self.top.title("Status")


        #Converting CSV files into JSON format
        self.airport_json = self.df[0].to_json(orient = 'records', indent = 4)
        self.runway_json = self.df[1].to_json(orient = 'records', indent = 4)
        self.freq_json = self.df[2].to_json(orient = 'records', indent = 4)

        self.label =tk.Label(self.top, text = "                    CSV files coverted to JSON ")
        self.label.place( relx = 0.2, rely = 0.4)

        self.savebtn = tk.Button(self.top, text = " SAVE to MongoDB ", command = self.save_csv_to_mongoDB)
        self.savebtn.place( relx = 0.4, rely = 0.7)

        self.cleanbtn.place_forget()

        self.top.grab_set()
```

3

```
    #Window that designates the names under which converted csv files will be saved into the database
    def save_csv_to_mongoDB(self):


        self.database = self.client["data"]

        self.savebtn.place_forget()
        self.label.place_forget()

        #Generating collections

        self.collection_airport = self.database["airports"]
        self.collection_runway = self.database["runways"]
        self.collection_freq = self.database["frequencies"]


        #placing data into individual collections

        self.collection_airport.insert_many(json.loads(self.airport_json))
        self.collection_runway.insert_many(json.loads(self.runway_json))
        self.collection_freq.insert_many(json.loads(self.freq_json))

        #Redirectiong to the CSV to JSON function

        self.label =tk.Label(self.top, text = " Files saved in MongoDB database data as following collections: \r
        self.label.place( relx = 0.1, rely = 0.1)


        okbtn = tk.Button(self.top, text = " OK ", command = self.manipulation_frame)
        okbtn.place( relx = 0.5, rely = 0.8)

        self.top.grab_set() # Deactivate the main window in the background
```

4

```
    #The function below parses and cleans columns with null values and at the same time
    #retains columns 'type' and 'iso_country' which will be needed for data manipulation tasks
    #Also deletes rows with null values in 'type' and 'iso_country' attributes
    def cleaned_df(self):

        for arg in self.df:
            #Driving out columns with missing values
            delete_columns=arg.columns[arg.isnull().any()].tolist()

            #Removing preexisting columns 'type' and ' iso_country' from the list
            if 'type' in delete_columns:
                delete_columns.remove('type')

            if 'iso_country' in delete_columns:
                delete_columns.remove('iso_country')

            #Dropping columns with null values
            arg.drop(delete_columns,axis=1,inplace=True)

            #Dropping rows where column iso_country is null
            arg.dropna(inplace=True)

```

5

```
    #====================Functions associated with statistical calculations ================================;

    def summary_Large(self):  #This function calculates and displays the mean/median/mode of the input dataframe


        self.task2_display_frame_blank.place_forget()
        self.task2_display_frame_allUK.place_forget()
        self.task2_display_frame_small.place_forget()
        self.task2_display_frame_large.place_forget()
        self.task2_display_frame_medium.place_forget()
        self.task2_display_frame_visual.place_forget()

        self.task2_display_frame_stats.configure(text = "Summary stats of frequency for large UK airports")
        self.task2_display_frame_stats.place( height = 700, width = 1199, relx = 0.0, rely = 0.07 )

        #Generates and places the Tkinter text widget on task2_display_frame
        self.display_text = tk.Text(self.task2_display_frame_stats)
        self.display_text.place(relwidth = 1, relheight = 1)

        #Calculates the mean, median and mode on all large UK airports sets
        mean_df = np.round(self.df_airport_large['frequency_mhz'].mean(),2)
        median_df = np.round(self.df_airport_large['frequency_mhz'].median(),2)
        mode_df = self.df_airport_large['frequency_mhz'].mode()

        self.display_text.insert(END, " \n\n\n SUMMARY STATS : frequency for all large UK Airports \n\n MEAN of f
        self.display_text.configure(state = DISABLED) #Read-only setting for the text widget
```

```
    def summary_Freq_100(self):  # Calculates and displays the mean ,mode and median of the input dataframe

        self.task2_display_frame_blank.place_forget()
        self.task2_display_frame_allUK.place_forget()
        self.task2_display_frame_small.place_forget()
        self.task2_display_frame_large.place_forget()
        self.task2_display_frame_medium.place_forget()
        self.task2_display_frame_visual.place_forget()

        self.task2_display_frame_stats.configure(text = "Summary stats of frequency > 100Mhz for all UK airports'
        self.task2_display_frame_stats.place( height = 700, width = 1199, relx = 0.0, rely = 0.07 )

        #Generates and places the Tkinter text widget on task2_display_frame
        self.display_text = tk.Text(self.task2_display_frame_stats)
        self.display_text.place(relwidth = 1, relheight = 1)

        #Merges all small, medium and large pandaframes to obtain all frequencies > 100
        temp_df = pd.concat([self.df_airport_small, self.df_airport_medium, self.df_airport_large])
        temp_df = temp_df[temp_df['frequency_mhz']>100]

        #Calculates the mean, median and mode on all large UK airports
        mean_df = np.round(temp_df['frequency_mhz'].mean(),2)
        median_df = np.round(temp_df['frequency_mhz'].median(),2)
        mode_df = temp_df['frequency_mhz'].mode()

        self.display_text.insert(END, " \n\n\n SUMMARY STATS : All UK Airports with frequency >100 Mhz \n\n MEAN
        self.display_text.configure(state = DISABLED) #Read-only setting for the text widget
```

6

```
        #plot
        self.plot1.hist(self.df_airport_small['frequency_mhz'], label = 'small', alpha = 0.5)
        self.plot1.hist(self.df_airport_medium['frequency_mhz'], label = 'medium', alpha = 0.5)
        self.plot1.hist(self.df_airport_large['frequency_mhz'], label = 'large', alpha = 0.5)
        self.plot1.legend()
```

7

```
        messagebox.showinfo(" Saving into MongoDB ", "Complete", parent = self.root)

    #Function to plot communication frequencies of small UK airports
    def plotSmallFreq(self):
        self.task2_display_frame_visual.place( height = 700, width = 1199, relx = 0.0, rely = 0.07 )
        self.task2_display_frame_visual.configure(text = "")
        self.fig = Figure(figsize = (20,20), dpi = 100)  #Defining geometry of figure that will hold the plot
        self.plot1 = self.fig.add_subplot(111)    # Adding a subplot

        #Creating a canvas using Tkinter

        self.canvas = FigureCanvasTkAgg(self.fig, master = self.task2_display_frame_visual)
        self.plot1.hist(self.df_airport_small['frequency_mhz']) #Plotting the graph


        self.canvas.draw()
        self.canvas.get_tk_widget().place(relheight = 1, relwidth = 1)
```

8

```
    #This function is used to determine correlation
    def plotCorrelation(self):
        self.task2_display_frame_visual.place( height = 700, width = 1199, relx = 0.0, rely = 0.07 )
        self.task2_display_frame_visual.configure(text = "")
        self.fig = Figure(figsize = (20,20), dpi = 100)  #Defines the figure that will hold the plot
        self.plot1 = self.fig.add_subplot(111)    #Adding a subplot

        #Generating a canvas using Tkinter

        self.canvas = FigureCanvasTkAgg(self.fig, master = self.task2_display_frame_visual)

        #plot
        self.plot1.hist(self.df_airport_small['frequency_mhz'], label = 'small', alpha = 0.5)
        self.plot1.hist(self.df_airport_medium['frequency_mhz'], label = 'medium', alpha = 0.5)
        self.plot1.hist(self.df_airport_large['frequency_mhz'], label = 'large', alpha = 0.5)
        self.plot1.legend()

        self.canvas.draw()
        self.canvas.get_tk_widget().place(relheight = 1, relwidth = 1)
```

9

Bibliography

[1] Marowka, A., 2018. On parallel software engineering education using python. *Education and Information Technologies*, *23*(1), pp.357-372.

[2] Palach, J., 2014. *Parallel programming with Python*. Packt Publishing Ltd.

[3] Williamson, T. and Olsson, R.A., 2014. PySy: a Python package for enhanced concurrent programming. *Concurrency and Computation: Practice and Experience*, *26*(2), pp.309-335.

[4] Deitel, P. and Deitel, H., 2020. *Intro to Python for Computer Science and Data Science*. Pearson Education.

[5] Malakhov, A., 2016, July. Composable multi-threading for Python libraries. In *Proceedings of the 15th Python in Science Conference, Austin, TX, USA* (pp. 11-17).

[6] O'higgins, N., 2011. *MongoDB and Python: Patterns and processes for the popular document-oriented database*. "O'Reilly Media, Inc.".

[7] Beazley, D., 2010, February. Understanding the python gil. In *PyCON Python Conference. Atlanta, Georgia*.

[8] Eggen, R. and Eggen, M., 2019. Thread and process efficiency in python. In *Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA)* (pp. 32-36). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[9] Győrödi, C., Győrödi, R., Pecherle, G. and Olah, A., 2015, June. A comparative study: MongoDB vs. MySQL. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)* (pp. 1-6). IEEE.

[10] Sodan, A.C., Machina, J., Deshmeh, A., Macnaughton, K. and Esbaugh, B., 2010. Parallelism via multithreaded and multicore CPUs. *Computer*, *43*(3), pp.24-32.

[11] Kuhlman, D., 2009. *A python book: Beginning python, advanced python, and python exercises* (pp. 1-227). Lutz: Dave Kuhlman.

[12] Oppermann, R., 2002. User-interface design. In *Handbook on information technologies for education and training* (pp. 233-248). Springer, Berlin, Heidelberg.

[13] Blair-Early, A. and Zender, M., 2008. User interface design principles for interaction design. *Design Issues*, *24*(3), pp.85-107.

[14] Najjar, L.J., 1998. Principles of educational multimedia user interface design. *Human factors*, *40*(2), pp.311-323.

[15] Morville, P., 2005. Experience design unplugged. In *ACM SIGGRAPH 2005 Web program* (pp.10-es).

[16] Morville, P., 2005. Experience design unplugged. In *ACM SIGGRAPH 2005 Web program* (pp.10-es).

[17] Troop, M., White, D., Wilson, K.E. and Zeni, P., 2020. The User Experience Design for Learning (UXDL) Framework: The Undergraduate Student Perspective. *The Canadian Journal for the Scholarship of Teaching and Learning*, *11*(3).

[18] Dillon, A., 2003. User interface design. *MacMillan Encyclopedia of Cognitive Science*, *4*, pp.453-458.

[19] Mayhew, D.J., 1991. *Principles and guidelines in software user interface design*. Prentice-Hall, Inc..

[20] Sridevi, S., 2014. User interface design. *International Journal of Computer Science and Information Technology Research*, *2*(2), pp.415-426.

[21] Hawlitschek, F., Jansen, L.E., Lux, E., Teubner, T. and Weinhardt, C., 2016, January. Colors and trust: The influence of user interface design on trust and reciprocity. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (pp. 590-599). IEEE.

[22] Lutz, M., 2013. *Learning python: Powerful object-oriented programming*. " O'Reilly Media, Inc.".

[23] Stančin, I. and Jović, A., 2019, May. An overview and comparison of free Python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 977-982). IEEE.

[24] vanRossum, G., 1995. Python reference manual. *Department of Computer Science [CS]*, (R 9525).

[25] Lin, J.W.B., 2012. Why Python is the next wave in earth sciences computing. *Bulletin of the American Meteorological Society*, *93*(12), pp.1823-1824.

[26] Van Rossum, G., 2007, June. Python Programming language. In *USENIX annual technical conference* (Vol. 41, No. 1, pp. 1-36).

[27] Dierbach, C., 2014. Python as a first programming language. *Journal of Computing Sciences in Colleges*, *29*(3), pp.73-73.

[28] Sanner, M.F., 1999. Python: a programming language for software integration and development. *J Mol Graph Model*, *17*(1), pp.57-61.

[29] Randles, B.M., Pasquetto, I.V., Golshan, M.S. and Borgman, C.L., 2017, June. Using the Jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)* (pp. 1-2). IEEE.

[30] Millman, K.J. and Aivazis, M., 2011. Python for scientists and engineers. *Computing in Science & Engineering*, *13*(2), pp.9-12.

[31] Ismail, M. and Suh, G.E., 2018, September. Quantitative overhead analysis for Python. In *2018 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 36-47). IEEE.

[32] Destefanis, G., Ortu, M., Porru, S., Swift, S. and Marchesi, M., 2016, May. A statistical comparison of Java and Python software metric properties. In *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics* (pp. 22-28).

[33] Stančin, I. and Jović, A., 2019, May. An overview and comparison of free Python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 977-982). IEEE.

[34] McKinney, W., 2011. pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing*, *14*(9), pp.1-9.

[35] ibid.,pg 7

[36] ibid.,pg 7

[37] ibid.,pg 8

[38] ibid.,pg 9

[39] Brzustowicz, M.R., 2017. *Data Science with Java: Practical Methods for Scientists and Engineers*. " O'Reilly Media, Inc.".

[40] Vitek, J. and Bokowski, B., 2001. Confined types in Java. *Software: Practice and Experience*, *31*(6), pp.507-532.

[41] Cramer, T., Friedman, R., Miller, T., Seberger, D., Wilson, R. and Wolczko, M., 1997. Compiling Java just in time. *Ieee micro*, *17*(3), pp.36-43.

[42] Abdikamalov, A., 2013. PYTHON SCRIPTING AND SCIENTIFIC COMPUTING. *ВЕСТНИК КАРАКАЛПАКСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА ИМЕНИ БЕРДАХА*, *19*(1-2), pp.24-30.

[43] Grus, J., 2019. *Data science from scratch: first principles with python*. O'Reilly Media.

[44] Saxena, A., Kaushik, N., Kaushik, N. and Dwivedi, A., 2016, March. Implementation of cloud computing and big data with Java based web application. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 1289-1293). IEEE.

[45] Stančin, I. and Jović, A., 2019, May. An overview and comparison of free Python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 977-982). IEEE.

[46] Nurseitov, N., Paulson, M., Reynolds, R. and Izurieta, C., 2009. Comparison of JSON and XML data interchange formats: a case study. *Caine*, *9*, pp.157-162.

[47] Chauhan, D. and Bansal, K.L., 2017. Using the advantages of NOSQL: a case study on MongoDB. *International Journal on Recent and Innovation Trends in Computing and Communication*, *5*(2), pp.90-93.

[48] Wehner, P., Piberger, C. and Göhringer, D., 2014, May. Using JSON to manage communication between services in the Internet of Things. In *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)* (pp. 1-4). IEEE.

[49] Maeda, K., 2012, May. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)* (pp. 177-182). IEEE

[50] Langdale, G. and Lemire, D., 2019. Parsing gigabytes of JSON per second. *The VLDB Journal*, *28*(6), pp.941-960.

[51] Dhalla, H.K., 2020, November. A Performance Analysis of Native JSON Parsers in Java, Python, MS. NET Core, JavaScript, and PHP. In *2020 16th International Conference on Network and Service Management (CNSM)* (pp. 1-5). IEEE

[52] Zunke, S. and D'Souza, V., 2014. Json vs xml: A comparative performance analysis of data exchange formats. *IJCSN International Journal of Computer Science and Network*, *3*(4), pp.257-261.

[53] Droettboom, M., 2015. Understanding JSON Schema. *Available on: http://spacetelescope. github. io/understanding-jsonschema/UnderstandingJSONSchema. pdf (accessed on 14 April 2022)*.

[54] Bernard, J., 2016. Python data analysis with pandas. In *Python Recipes Handbook* (pp. 37-48). Apress, Berkeley, CA

[55] Nelli, F., 2018. Pandas: reading and writing data. In *Python Data Analytics* (pp. 141-180). Apress, Berkeley, CA.

[56] McKinney, W., 2011. pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing*, *14*(9), pp.1-9.

[57] Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M. and Vrgoč, D., 2016, April. Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 263-273).

[58] Jose, B. and Abraham, S., 2017, July. Exploring the merits of nosql: A study based on mongodb. In *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)* (pp. 266-271). IEEE.

[59] Győrödi, C., Győrödi, R., Pecherle, G. and Olah, A., 2015, June. A comparative study: MongoDB vs. MySQL. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)* (pp. 1-6). IEEE.

[60] Arora, R. and Aggarwal, R.R., 2013. Modeling and querying data in mongodb. *International Journal of Scientific and Engineering Research*, *4*(7), pp.141-144.

[61] Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M. and Vrgoč, D., 2016, April. Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 263-273).

[62] Nurseitov, N., Paulson, M., Reynolds, R. and Izurieta, C., 2009. Comparison of JSON and XML data interchange formats: a case study. *Caine*, *9*, pp.157-162

[63] Wang, G., 2011, April. Improving data transmission in web applications via the translation between XML and JSON. In *2011 Third International Conference on Communications and Mobile Computing* (pp. 182-185). IEEE.

[64] Zunke, S. and D'Souza, V., 2014. Json vs xml: A comparative performance analysis of data exchange formats. *IJCSN International Journal of Computer Science and Network*, *3*(4), pp.257-261.

[65] Maiwald, B., Riedle, B. and Scherzinger, S., 2019, November. What are real JSON schemas like?. In *International Conference on Conceptual Modeling* (pp. 95-105). Springer, Cham.

[66] McKinney, W., 2010, June. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, No. 1, pp. 51-56).

[67] Oliphant, T.E., 2006. *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.

[68] Oliphant, T.E., 2006. *A guide to NumPy* (Vol. 1, p. 86-87). USA: Trelgol Publishing.

[69] Lee, Y.G. and Kim, S.Y., 2008. Introduction to statistics. *Yulgokbooks, Korea*, pp.342-351.

[70] Runnenburg, J.T., 1978. Mean, median, mode. *Statistica Neerlandica*, *32*(2), pp.73-79.

[71] Kaas, R. and Buhrman, J.M., 1980. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, *34*(1), pp.13-18.

[72] Sugiyama, M., Ghisu, M.E., Llinares-López, F. and Borgwardt, K., 2018. graphkernels: R and Python packages for graph comparison. *Bioinformatics*, *34*(3), pp.530-532.

[73] Arroyo, J., González-Rivera, G., Maté, C. and San Roque, A.M., 2011. Smoothing methods for histogram-valued time series: an application to value-at-risk. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *4*(2), pp.216-228.

[74] Adams, C., 2014. *Learning Python data visualization*. Packt Publishing Ltd.

[75] Milovanović, I., 2013. *Python data visualization cookbook*. Packt Publishing Ltd.

[76] Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. and Kern, R., 2020. Array programming with NumPy. *Nature*, *585*(7825), pp.357-362.

[77] Pajankar, A., 2021. Getting Started with Pandas. In *Practical Python Data Visualization* (pp. 117-136). Apress, Berkeley, CA.

[78] Dembinski, H.P., Pivarski, J. and Schreiner, H., 2020. Recent developments in histogram libraries. In *EPJ Web of Conferences* (Vol. 245, p. 05014). EDP Sciences.

[79] Bernard, J., 2016. Numerics and Numpy. In *Python Recipes Handbook* (pp. 81-90). Apress, Berkeley, CA.

[80] Oliphant, T.E., 2006. *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.

[81] Barrett, P., Hunter, J., Miller, J.T., Hsu, J.C. and Greenfield, P., 2005, December. matplotlib--A Portable Python Plotting Package. In *Astronomical data analysis software and systems XIV* (Vol. 347, p. 91).

[82] Tosi, S., 2009. *Matplotlib for Python developers*. Packt Publishing Ltd.

[83] Lundh, F., 1999. An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. Htm*.

[84] Sanner, M.F., 1999. Python: a programming language for software integration and development. *J Mol Graph Model*, *17*(1), pp.57-61.

[85] Stallman, R., 2002. *Free software, free society: Selected essays of Richard M. Stallman*. Lulu. com.

[86] Singer, J. and Vinson, N.G., 2002. Ethical issues in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, *28*(12), pp.1171-1180.

[87] Gotterbarn, D., 1991, May. Ethical considerations in software engineering. In *Proceedings of the 13th international conference on Software engineering* (pp. 266-274).

[88] Jain, P., Gyanchandani, M. and Khare, N., 2016. Big data privacy: a technological perspective and review. *Journal of Big Data*, *3*(1), pp.1-25.

[89] Smid, M.E. and Branstad, D.K., 1988. Data encryption standard: past and future. *Proceedings of the IEEE*, *76*(5), pp.550-559.

[90] Newman, A.L., 2015. What the “right to be forgotten” means for privacy in a digital age. *Science*, *347*(6221), pp.507-508.

[91] Mujkic, E. and Klingner, D., 2019. Dieselgate: How hubris and bad leadership caused the biggest scandal in automotive history. *Public integrity*, *21*(4), pp.365-377.

[92] Nunes, M.F. and Park, C.L., 2016. Caught red-handed: the cost of the Volkswagen Dieselgate. *Journal of Global Responsibility*.

[93] Etzioni, A. and Etzioni, O., 2016. Designing AI systems that obey our laws and values. *Communications of the ACM*, *59*(9), pp.29-31.

[94] Agrawal, K., 2010. To study the phenomenon of the Moravec's Paradox. *arXiv preprint arXiv:1012.3148*.

[95] Hagendorff, T., 2020. The ethics of AI ethics: An evaluation of guidelines. *Minds and Machines*, *30*(1), pp.99-120.

[96] Gogoll, J. and Müller, J.F., 2017. Autonomous cars: in favor of a mandatory ethics setting. *Science and engineering ethics*, *23*(3), pp.681-700.

[97] Etzioni, A. and Etzioni, O., 2017. Incorporating ethics into artificial intelligence. *The Journal of Ethics*, *21*(4), pp.403-418.

[98] Baker, J.E., 1987, July. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms* (Vol. 206, pp. 14-21).

[99] Sun, W., Nasraoui, O. and Shafto, P., 2020. Evolution and impact of bias in human and machine learning algorithm interaction. *Plos one*, *15*(8), p.e0235502.

[100] Kosmidis, I. and Firth, D., 2010. A generic algorithm for reducing bias in parametric estimation. *Electronic Journal of Statistics*, *4*, pp.1097-1112.

[101] Thomson, A.J. and Schmoldt, D.L., 2001. Ethics in computer software design and development. *Computers and Electronics in Agriculture*, *30*(1-3), pp.85-102

[102] Bjørn, P., 2019. Dark Agile: Perceiving People As Assets, Not Humans. In *Rethinking Productivity in Software Engineering* (pp. 125-134). Apress, Berkeley, CA.

[103] Rasch, R.H. and Tosi, H.L., 1992. Factors affecting software developers' performance: An integrated approach. *MIS quarterly*, pp.395-413.

[104] Bjørn, P., 2019. Dark Agile: Perceiving People As Assets, Not Humans. In *Rethinking Productivity in Software Engineering* (pp. 135-137). Apress, Berkeley, CA.

[105] Jiang, J. and Klein, G., 2000. Software development risks to project effectiveness. *Journal of Systems and Software*, *52*(1), pp.3-10.

[106] Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J., 2017. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

[107] Loeliger, J. and McCullough, M., 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc."