Section 1: Theory supported by code samples

Prototype Application (493 words)

In this project, the aim is to develop a prototype application that demonstrates how the data from the given dataset can be formatted and cleaned in addition to being used to generate specific outputs considering the requirements. The application's front end and back end processing can be made to run in two different threads. One of the major issues faced while designing this prototype application is the responsiveness of the GUI application, as we cannot create or access the PyQt object outside the main thread. Moreover, even if the PyQt class is being reentrant, it cannot share access to the Qt Object between the threads until the PyQt documentation for the specific class clearly explains that the instances are now threaded safe. It also lacks python specific documentation for the classes in the PyQt5, plus grasping all the specifics of PyQt is very time-consuming, and it's quite a steep learning curve [1].

One of the easiest ways to communicate data between one thread to another thread is to use a Queue from the queue library. For this purpose, a Queue instance is created that is shared among the threads. In the next step, threads use put() or the get() operations to either add or remove items from the queue. Queue instances contain all the required lockings, which can be shared between multiple threads. While working on the queues, sometimes it becomes challenging to communicate the shutdown of the producer and the user. Queues are one of the most commonly used thread communication mechanisms, and anyone can build their own data structures considering the required locking and synchronization techniques available [2].

As thread communication using a queue involves a one-way and non-deterministic process, although there exists no possibility to realize whether the receiving thread has received a message or not. However, the queue objects contain basic completion features, illustrated by the task_done() and join() methods. Making a queue block when it is full can result in an unintended cascading effect in the program, which causes the program to run poorly and leads to a deadlock. The issue of flow control is one of the major problems among the communicating threads. A non-blocking put() can be used along with a fixed-sized queue to implement the different kinds of handling code when the queue is full. The utility methods q.size(), q.full(), and q.empty() can be used to tell the current size as well as the status of the queue. However, it must be noted that all of these methods are not useful in a multithreaded environment because call q.empty() can tell that the queue is empty.

Figure 1 explains the program flow, and the first step involves loading the dataset. After the dataset is cleaned, the data is translated into JSON, and the data is saved in files. The output is generated, and visualizations are created. In the program flow,

concurrency can potentially benefit at “Save The Data and Generate Output & Visualization.”

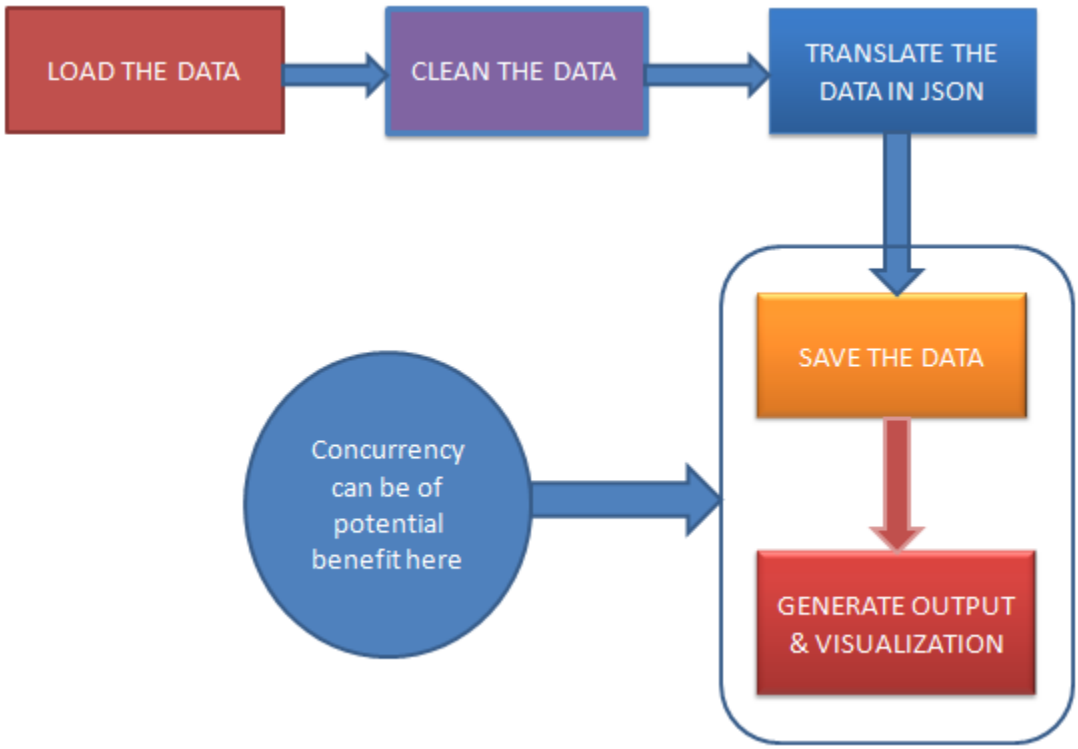


Figure 1 Overview Diagram of Program Flow

GUI interface constructs and best practices regarding interface layouts (442 words)

While constructing a GUI interface, the best practice is to keep the interface simple, i.e., avoid unnecessary elements and be very clear in the language used on the labels and messaging. More common elements must be used in the graphical user interface as it makes the user feel more comfortable while offering after the operation. There must be consistency in the design, and familiar user interface elements must be used to allow users to feel more comfortable and can complete the desired tasks quickly. The background's color, light, texture, and contrast should be managed carefully as they can direct and redirect the clients' attention. The user interface must be scalable and readable. The page layout must be purposeful, and there must be a spatial relationship between the items available on the page and the page's structure, considering the importance. In the page layout, the items must be placed carefully as the careful placement of the items can attract the users' attention. In addition, different

sizes, fonts, and arrangements of the text can be helpful to increase scalability, legibility, and readability.
In this project, the graphical user interface(s) for interacting with the data enables the user to load the initial dataset (from the CSV format) and subsequently translate it into a JSON format. Moreover, it facilitates the loading and saving of a prepared data set in the JSON format in addition to cleaning the dataset, manipulating the values range, and generating output and visualizations.

The detailed process is as follows:

In step one, the user uploads the dataset in the CSV format; the data is cleaned by removing the missing values and managing consistencies and errors.
In the second step, the data is translated to the JSON format, and the data is saved in the translated format, i.e., JSON format using the files.
The prepared/translated dataset generates output and visualizations using different Python libraries in the third step. The client's requirements regarding output generation and visualization are listed as follows.

- The output does not have any data from the airports that have the type 'closed'.
- The information from the type column is extracted into a new column, considering one for each airport category. The large airport, medium airport, and small airport categories are joined to the communication frequencies that the airport uses for communication.
- The mean, mode, and median for the "frequency_mhz" for each large airport are generated and visualized.
- The output plot is created for the communication frequencies used by the small airport; the data is grouped to make the visualization more feasible.
- The correlation between the communication frequencies used by the 3 different airport categories is established.

Java vs. Python (371 words)

Java is a statistically compiled language, while Python is a dynamically typed language. This makes Java faster at runtime and easier to debug, while a major advantage of python is the ease of use and reading compared to Java. Python has gained considerable popularity in the past few years as it is easier to understand. There are many libraries available in Python, which is also the case for Java, along with a large community for support [3].

Python tests the syntax during the runtime, thus not being as stable as Java despite the fast development. Java compiles code in advance and distributes the bytecode; due to Java's static-typing syntax, the compilation is faster and easier than Python's dynamic-typing. Python facilitates a number of container data types, including the built-in data types and those in the collections module in the python standard library. Each data container is used to perform a specific function. The data containers facilitate multiple functionalities and have their computational performance. Choosing the right container is an important step to achieving acceptable performance. The built-in containers available in python are lists, sets, dictionaries, and tuples [4]. As these container types are iterable, we can iterate objects so that each of the elements available in the container is only visited once. Multiple operations can be performed while considering these various container types, but the performance varies on large datasets.

Both python and java are popular high-level programming languages. There are several syntax overheads in Java code, while python code is much easier to read and write than Java. Many pseudo-codes available in the data structures and algorithms textbooks are similar to the python code. Furthermore, learning data structures with the help of python is much easier than Java for a beginner. Nonetheless, Java has its advantages, and it is an excellent language when Object-Oriented programming is considered. Many companies are still JVM in the production environment as it provides robustness. In contrast, the compiled Java code runs much faster than python, as python is an interpreted language.

Python is widely used in scientific and numeric computing, Machine Learning applications, image processing, and language development. In contrast, Java is preferred in Web applications, Desktop GUI Apps, enterprise solutions, and embedded systems.

Section 2: Design decisions supported by code samples(40% 1200 words plus code samples)

JSON, XML and CSV data formats (261 words)

In this research, three data frames (CSV files) are translated into JSON format; as in JSON, the data is represented in the form of name and the value pair. In the JSON format, the curly braces hold the objects, the colon follows each name, the comma separates the pairs, and the square brackets contain the arrays and the values. The reason for translating the data from the CSV format to JSON format is that JSON format is used as a syntax for storing and exchanging the data. The JSON format is less compact compared to the CSV file, while the CSV files are compact compared to

the other file formats. The CSV file format lags in terms of scalability; it is also less versatile. The JSON format is more human-readable compared to the XML format.

Advantages/Disadvantages it has demonstrated in this context:

- It is effortless to work with the JSON format; the JSON library facilitates converting the data, manipulating the data, reading and writing the data, and loading the JSON file using the JSON path. Working with the JSON file format in Python is the same as working with the Python dictionary. The JSON format is a popular file format while working with APIs. In JSON format, each record is self-contained, i.e., the column name's equivalent and the column values are in each record.
- One of the significant disadvantages while working with the JSON data rather than CSV data is that in JSON, it is challenging to process, manipulate and visualize the data as opposed to the CSV data.

Implementation of Client's 3rd requirement (308 words)

The third requirement was to calculate and produce the mean, mode, and median value plot for the "frequency_mhz" for large airports and for frequencies more than 100 MHz. The client's first two requirements are interlinked with the third; hence so essential to discuss the implementation of the first two requirements prior to the third. Primarily, all the data from the airport dataset that has the "type" "closed" is removed. Subsequently, the information on the type of the airport was extracted considering all the UK airports (large, medium, and small). By applying the merge and pandas libraries provided in python, the first dataset contains the communication frequencies, i.e., "frequency_mhz" that the airport uses for communication. The UK airports of different sizes are correctly matched with the corresponding communication frequencies. The corresponding code part is shown in Figure. 2. The client's third requirement is to produce the mean, mode, and median for the "frequency_mhz" column and visualize them for the large airport and frequencies more than 100 MHZ. From the "type" column, all the large UK airports are selected, and the mean, mode, and standard deviation are obtained and visualized. A horizontal bar plot of the mean value, median value, and standard deviation value for the large airport's communication frequencies is shown in Figures 2, 3, and 4.

```
data_final = pd.merge(data_two_new, data_one, how='inner', on = 'airport_ref')
```

```
data_final_new = data_final[data_final["type_x"] == "large_airport"]
```

```
df_frequency = data_final_new.groupby("type_x").agg([np.mean, np.std, np.median])
```

```
Frequency = df_frequency["frequency_mhz"]
Frequency.head()
```

type_x	mean	std	median
large_airport	120.007523	19.374093	121.85

```
Frequency.plot(kind = "barh", y = "mean", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94e1f21dd0>
```

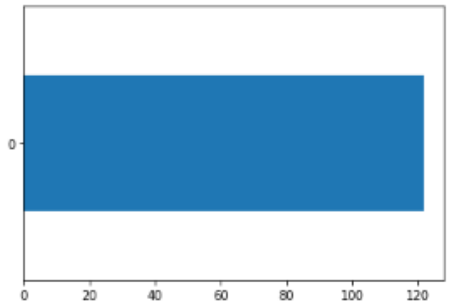


Figure 2: Mean Value for the Communication Frequencies for the Large Airport

.

```
Frequency.plot(kind = "barh", y = "median", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fb7a9710>
```

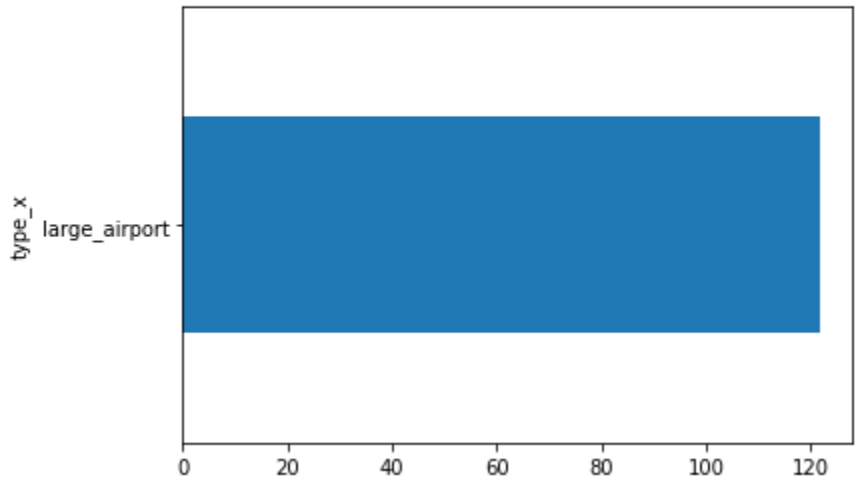


Figure 3: Median Value for the Communication Frequencies for the Large Airport

```
Frequency.plot(kind = "barh", y = "std", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fb573f90>
```

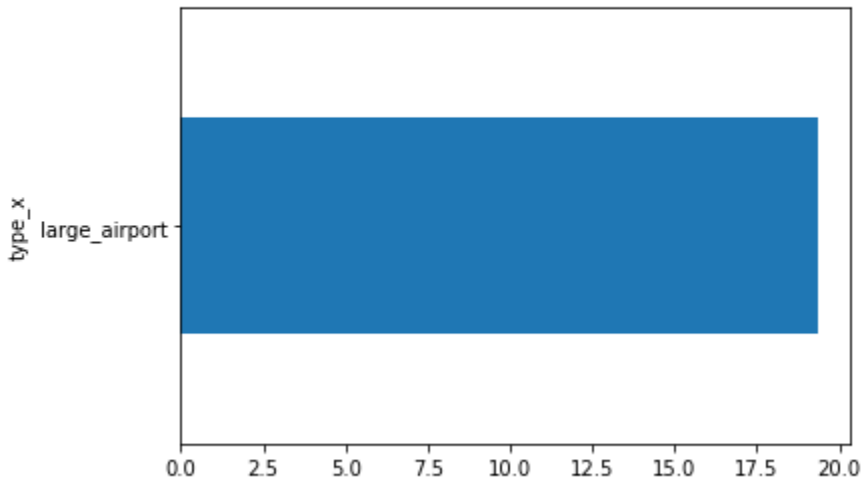


Figure 4 Standard Deviation Considering All the Large Airport

Furthermore, the mean, median, and mode values are calculated for the communication frequencies, i.e., frequency_mhz more than 100 MHz.
In Figures 5, 6, and 7, the Mean Value, Median Value, and standard deviation value plots are created for the communication frequencies greater than 100 MHz for the large, small, and medium airports.

```
data_final_new_new = data_final[data_final["frequency_mhz"] > 100]
```

```
df_frequency = data_final_new_new.groupby("type_x").agg([np.mean, np.std, np.median])
```

```
Frequency = df_frequency["frequency_mhz"]
Frequency.head()
```

type_x	mean	std	median
large_airport	123.581095	6.189527	121.900
medium_airport	167.817545	86.637667	125.875
small_airport	129.802617	25.772714	124.375

```
Frequency.plot(kind = "bar", y = "mean", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49ddfb1790>
```

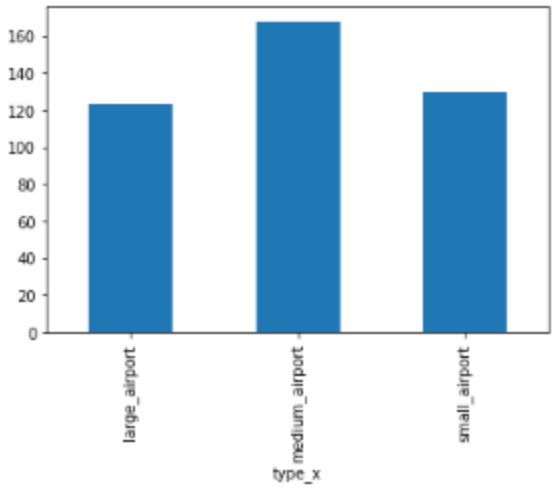


Figure 5: Mean Value Plot for the Communication Frequencies Greater than 100 MHz

```
Frequency.plot(kind = "bar", y = "median", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fe293f10>
```

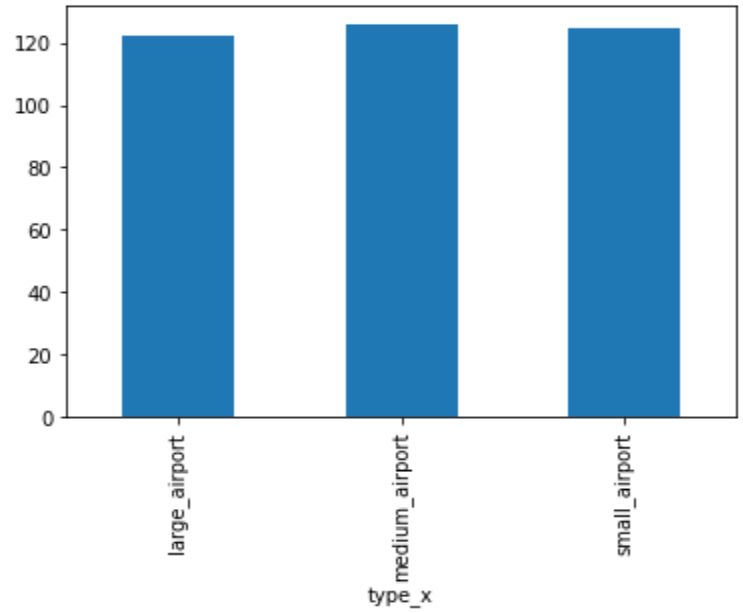


Figure 6: Median Value Plot for the Communication Frequencies Greater than 100 MHz

```
[207] Frequency.plot(kind = "bar", y = "std", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fe268cd0>
```

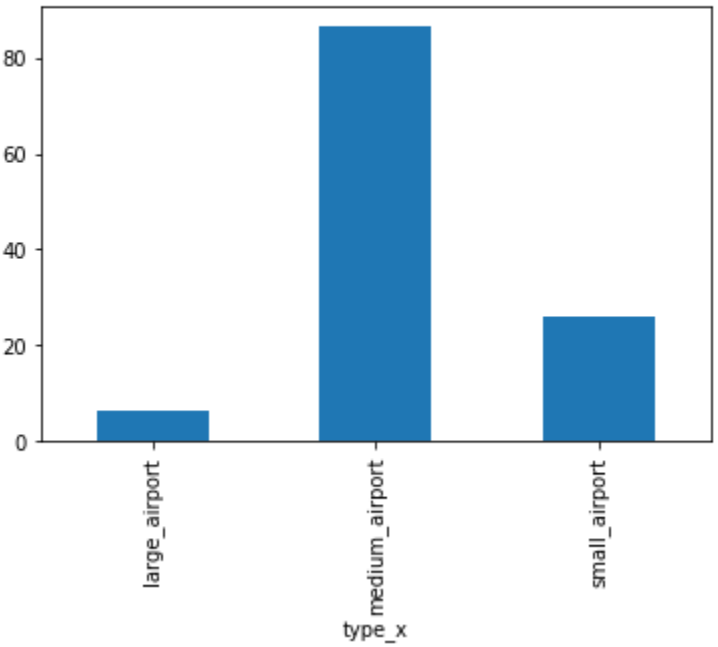


Figure 7: Standard Deviation Value Plot for the Communication Frequencies Greater than 100 MHz

Figure 8 shows the bar plot of the communication frequencies used by the large airport; along with this, the mean and median values bar plots for the frequencies greater than 100 MHz are also shown as the subplots.

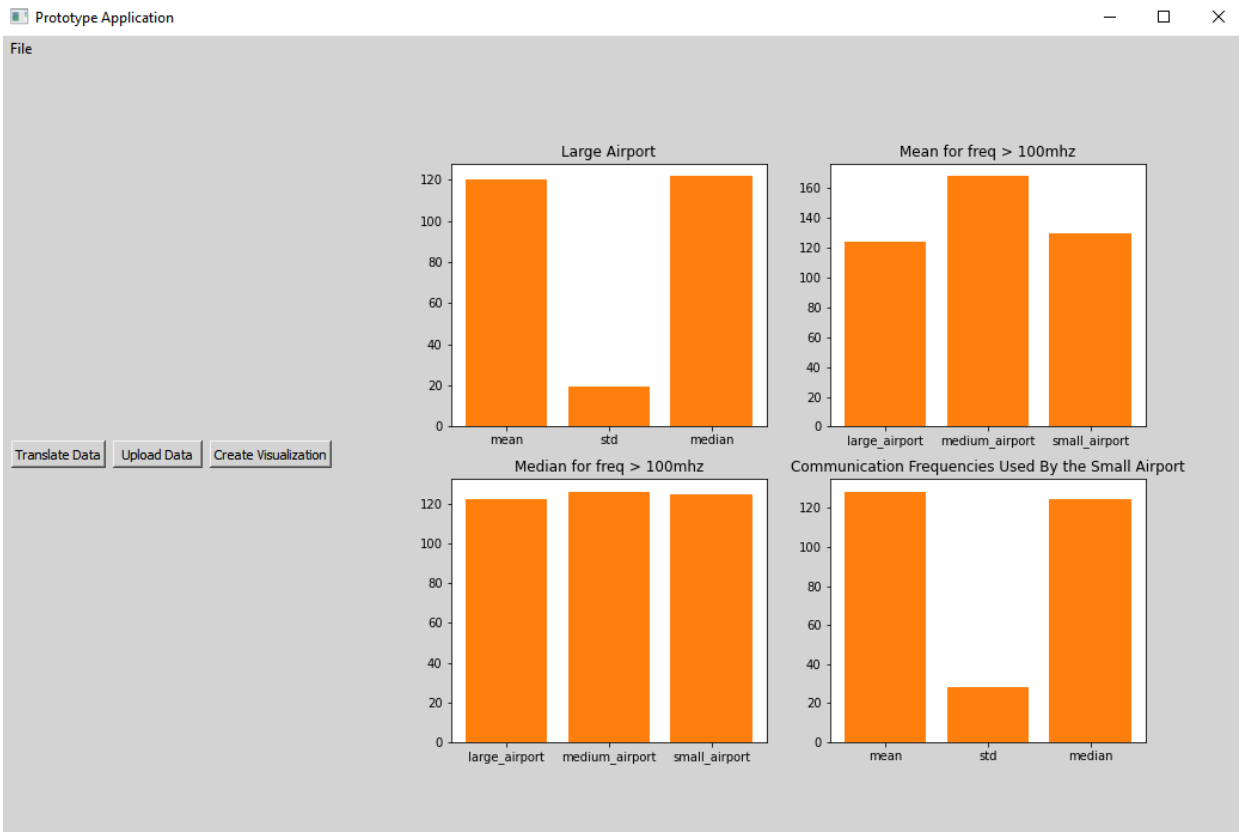


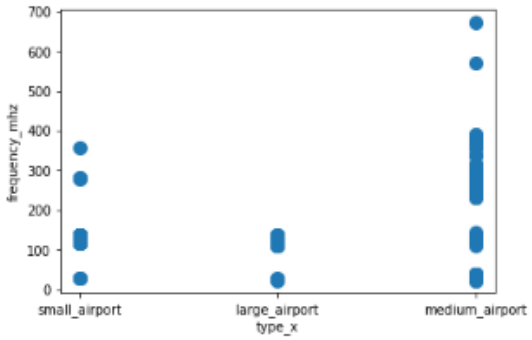
Figure 8 GUI Interface

Implementation of Client's 4th requirement (292 words)

The client's 4rth requirement is to create a suitable graph to visualize the small airport's communication frequencies, i.e., "frequency_mhz." First, a scatter plot is created, representing each airport's communication frequencies. Subsequently, the mean, standard deviation, and median values of the communication frequencies (frequency_mhz)are calculated. The pandas, matplotlib library, and group by function are used to aggregate the data considering the mean, median, and standard deviation. For visualization, the seaborn library can also be considered. The matplotlib library is used to create a horizontal bar chart Matplotlib, and it works efficiently with data frames and arrays. It treats figures and axes as objects and contains various stateful APIs for plotting.

A scatter plot is created for the communication frequencies “frequency_mhz” used by the small airport, large airport, and the medium airport, as shown in Figure 9. The values containing the small airport that correctly matched the corresponding frequencies were selected from the “Type” column. The python script is also shown in Figure 9, in which a scatter plot is created for the communication frequencies used by the small, medium and large airports. Using the group by function, mean, median, and standard deviation values are calculated for the communication frequencies used by the small airport.

```
data_final.plot.scatter(x = 'type_x', y = 'frequency_mhz', s = 100);
```




```
data_final_new = data_final[data_final["type_x"] == "small_airport"]
```

```
df_frequency = data_final_new.groupby("type_x").agg([np.mean, np.std, np.median])
```

```
Frequency = df_frequency["frequency_mhz"]

# checking for results
Frequency.head()
```

	mean	std	median
type_x			
small_airport	128.251321	28.454036	124.2125

Figure 9. Scatter Plot of the Range of Communication Frequencies used by the Small, Medium, and Large Airport, and Mean, Median, and Standard Deviation Values for the Type Column with Small Airport Type

Figure 10 represents the mean value plot considering the type column, with values containing the small airport.

```
Frequency.plot(kind = "barh", y = "mean", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49ddeaab90>
```

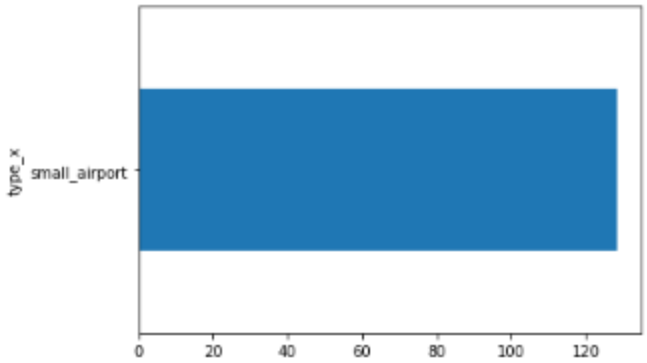


Figure 10. Horizontal Bar Chart of the Mean Values with Type Small Airport

Figure 11 represents the standard deviation value plot considering the Type column containing the small airport values.

```
Frequency.plot(kind = "barh", y = "std", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49ddecb310>
```

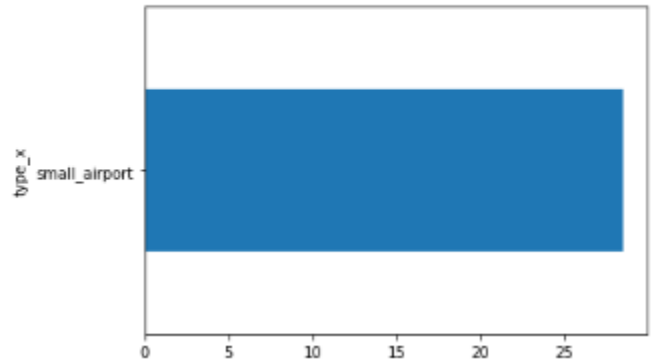


Figure 11. Horizontal Bar Chart of the Standard Deviation Values with Type Small Airport

Figure 12 represents the Median value plot considering the type column, with values containing the small airport.

```
Frequency.plot(kind = "barh", y = "median", legend = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49dddd03d0>
```

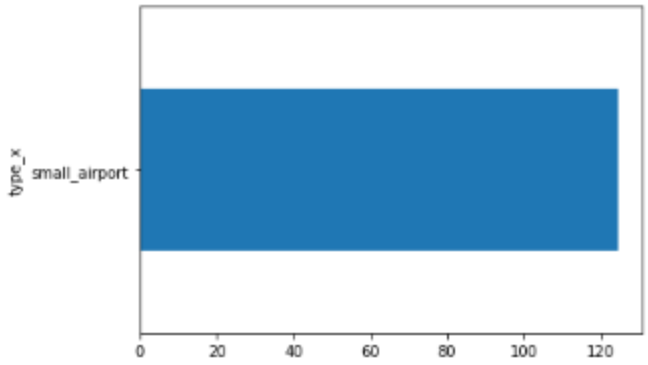


Figure 12. Horizontal Bar Chart of the Median Values with Type Small Airport

GUI Interface

In the last subplot of the GUI interface shown in Figure 13, the Mean Value, Median Value, and Standard Deviation Values of the communication frequencies used by the small airport are represented by the bar chart.

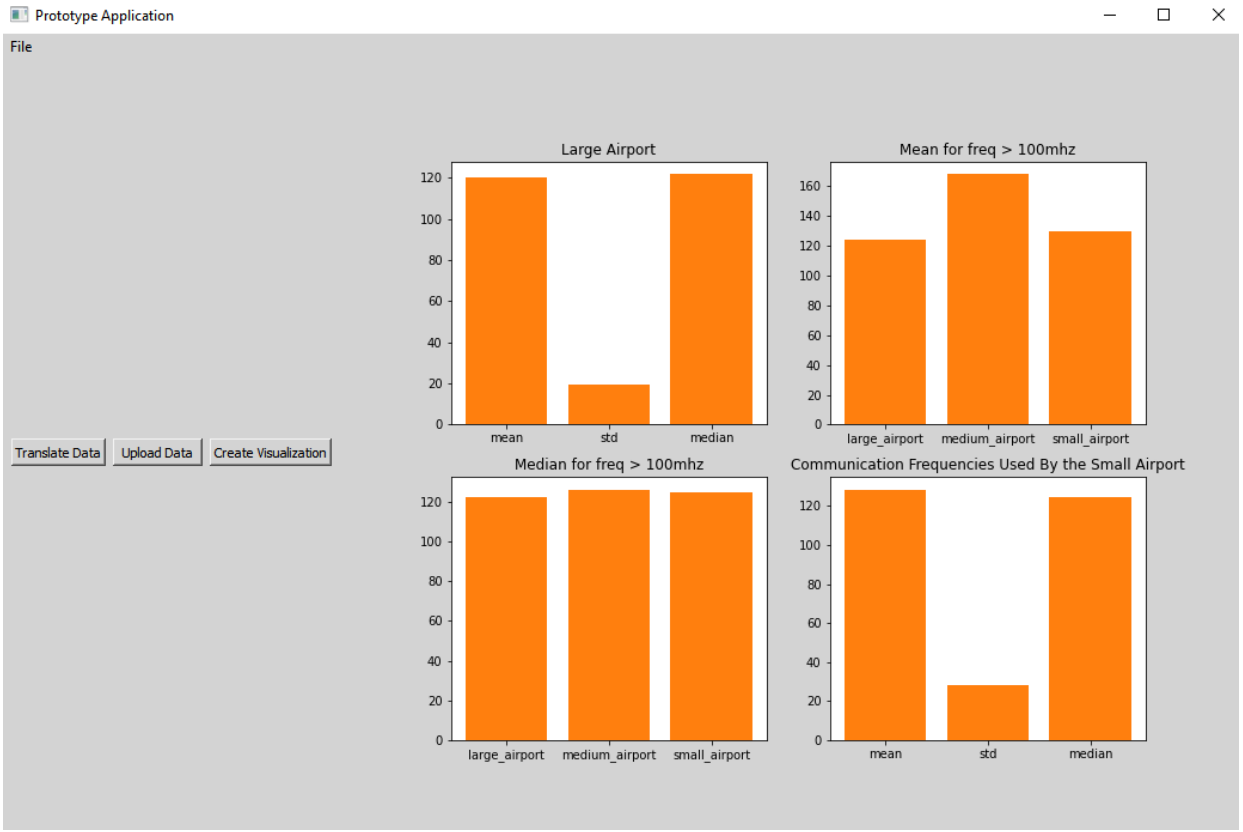


Figure 13. GUI Interface

Correlation between the communication frequencies (274 words)

The pearsonr() SciPy function is applied to calculate the Pearson's correlation coefficient between two data samples with the same length. Spearman's correlation coefficient can also be used to summarize the strength between the two data samples. Such analysis can also be used if a linear relationship exists between the variables, though it has slightly less power. The seaborn.reg method is used to create a correlation scatter plot, and the seaborn library is imported to create a correlation scatter plot. It accepts two features, one for the x-axis and the other for the y-axis. The scatter plot will be made for the two variables on the x-axis and y-axis. The Pearson's correlation coefficient for the communication frequencies of the small and medium airports is 0.02, and for the large and the medium airport, 0.073, while for the large and small airports, it is -0.020. In the second step, to plot the correlation between the communication frequencies, i.e., "frequency_mhz" used by the three airports, the seaborn.reg plot is applied. Three different reg plots were created, as each plot accepts two features. In the first plot, the two features considered for the correlation scatter plot include the communication frequencies used by the small and large airports. The resulting plot is shown in Figure 14.

```
data_final_large = data_final[data_final["type_x"] == "large_airport"]
```

```
data_final_large.shape
```

```
(109, 13)
```

```
data_final_small = data_final[data_final["type_x"] == "small_airport"]
```

```
data_final_small = data_final_small.iloc[0:109, :]
```

```
data_final_small.shape
```

```
(109, 13)
```

```
data_final_medium = data_final[data_final["type_x"] == "medium_airport"]
```

```
import seaborn as sns

sns.regplot(x=data_final_large["frequency_mhz"], y=data_final_small["frequency_mhz"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49ddd864d0>
```

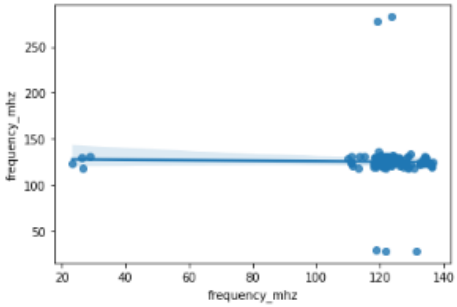


Figure 14. Correlation Scatter Plot of the Communication Frequencies used by the Large Airport and Small Airport

In the following plot, the two features considered for the correlation scatter plot include the communication frequencies used by the large and medium airports, as shown in Figure 15.

```
sns.regplot(x=data_final_large["frequency_mhz"], y=data_final_medium["frequency_mhz"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fb5e3c90>
```

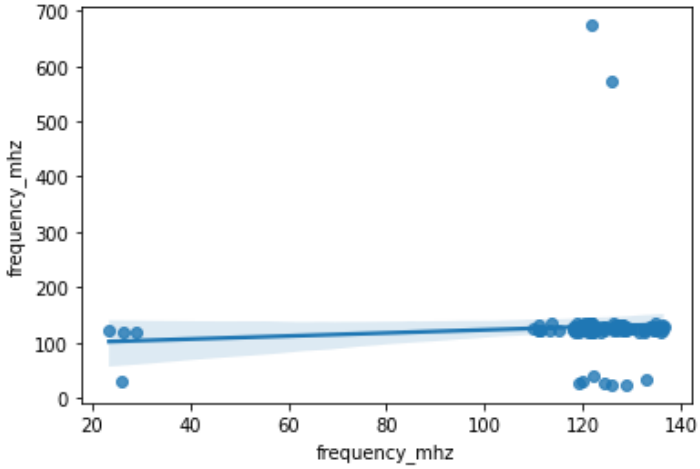


Figure 15. Correlation Scatter Plot of the Communication Frequencies used by the Large Airport and Medium Airport

In the third plot, the features considered for the correlation scatter plot include the communication frequencies used by the small and medium airports, as shown in Figure 16.

```
sns.regplot(x=data_final_small["frequency_mhz"], y=data_final_medium["frequency_mhz"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95fbd084d0>
```

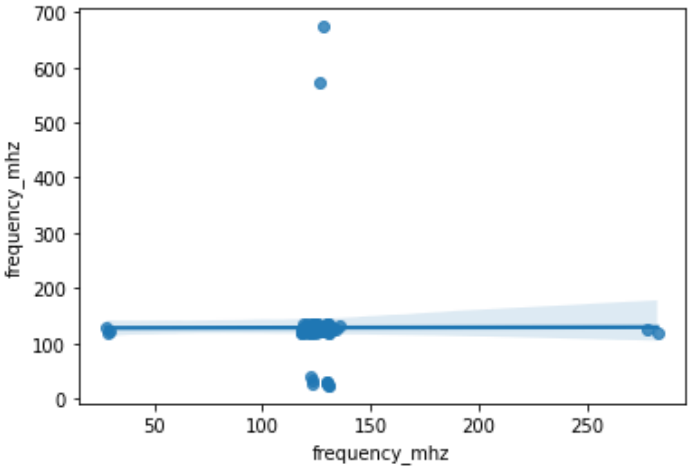


Figure 16. Correlation Scatter Plot of the Communication Frequencies used by the Small and the Medium Airport

Section 3: Reflection on the ethics, moral and legal aspects (10% 400 words)

The ethics, moral and legal aspects of computing (322 words)

Computing technologies and artifacts are increasingly integrated into most aspects of our professional, social, and private lives. The increasing ubiquity of computing technologies and artifacts leads to the growing relevance of ethical aspects of computing [5]. We are now at a point where computing technologies and devices pervade most aspects of personal, organizational, and social life. Developers often only address security after code release rather than during development. Our current digital ecosystem addresses application security by plugging vulnerabilities as they are found. The reactionary approach is neither practical nor pragmatic. As a result, there is an issue with secure development standards [6].
In many cases, the development team is measured on their rate of feature development; there's a high probability that the ethics of a given implementation might not be front of mind, either at the design or the implementation phase. However, in this prototype application, all the ethical issues are considered in designing. It is designed considering the client's requirements and performs all the required functions mentioned in the module. The design of the GUI interface is kept simple and easy, and the files are used as a means to back up the data as well.

Software engineers should not be subject to regulation by a central body as regulations can affect technological development in the IT sector and influence the innovation process from research through technology diffusion. The unnecessary regulations around software development and adding extra security updates can put the organization and its data at risk. However, software engineers need to design their data management processes to support not only privacy but also improved operational efficiency. Having IT asset management protocols in place that monitor data and compliance will help to mitigate risks and data security breaches. An effective system for IT security compliance must be there to ensure that only individuals with the appropriate credentials can access the secure systems and databases that contain sensitive customer data [7].

References:

[1] - Adawadkar, Kalyani. "Python ProgrammingApplications and Future." International Journal of Advanced Engineering and Research Development. http://ijaerd. com/papers/special_papers/IT032. pdf ,2017.

[2] -Srinath, K. R. "Python–The Fastest Growing Programming Language." International Research Journal of Engineering and Technology (IRJET), 2017, 354-357.

[3] - Åkesson, Tobias, and Rasmus Horntvedt. "Java, Python and Javascript, a comparison." 2019.

[4] - Foster Elvis. “A comparative analysis of the C++, Java, and Python Languages.” , 2014.

[5] - Bill Albrecht, Ken Christensen, Venu Dasigi, Jim Huggins, and Jody Paul, The pledge of the computing professional: Recognizing and promoting ethics in the computing professions. SIGCAS Computers and Society, 2012, 42, 1, 6–8.

[6] - S. A. Bages, A. Zeidler, C. Klein, F. Valdivielso, and R. Matias, Enabling personal privacy for pervasive computing environments. Journal of Universal Computer Science 16, 2010, 341–371

[7] - BAILY, Martin Neil, Competition, Regulation and Efficiency in Service Industries, Brookings Papers in Microeconomics 2, 1993.