Critical Reflective Writing

Name: []

Student ID: []

Date: 04/07/2024

**What worked for you well? (minimum 200-300 words excluding code, images, tables.)**

**Ggplot2**

The use of ggplot2 was surprisingly easy and intuitive. Normally, I’m not a big fan of relying on external libraries, for example, in Python, where I find myself overly dependent. This would make my code clustered or random at times, especially for someone reviewing my code. However, with ggplot2, it was the opposite. ggplot2 also aligns with best practices in reproducible research and visual storytelling, as discussed by Wickham [1]. I used ggplot2 in Questions 6, 7, and 9, and this made my code clean, consistent and readable. For example, in Q9.R, as instructed, I created a scatter plot comparing landing weights and maximum speed for Airbus A320 aircraft. This code snippet shows how I built the visualisation using ggplot2:

scatter\_plot <- ggplot(a320\_data,

                      aes(x = `Maximum Landing Weight (kg)`,

                          y = `Maximum Speed (knots)`)) +

  geom\_point(color = "#b83b58", size = 3, alpha = 0.8) +

  labs(title = "Airbus A320 Performance Analysis: Landing Weight vs Maximum Speed",

       x = "Maximum Landing Weight (kg)",

       y = "Maximum Speed (knots)") +

  theme\_minimal() +

  theme(

    panel.border = element\_rect(color = "black", fill = NA, linewidth = 1),

    plot.title = element\_text(face = "bold", size = 14),

    axis.title = element\_text(size = 12)

  ) +

  geom\_smooth(method = "lm", color = "darkgray", se = FALSE)

*(Figure 1: Scatterplot built using ggplot2 – Q9.R)*

The interactive Loop material was beneficial, especially Unit 4 – “Generating Graphs and Charts in R”, where the descriptive explanation was crucial to answering the questions.

**Reusable stats function**

Using functions worked in my favour, in Q8.R, I wrote a function to calculate mean, variance and standard deviation:

calculate\_stats <- function(values) {

  clean\_values <- na.omit(values)

  list(

    mean = mean(clean\_values),

    variance = var(clean\_values),

    std\_dev = sd(clean\_values)

  )

}

*(Figure 2: Reusable stats function – Q8.R)*

Figure 2 reduces the chance of errors occurring since the logic is defined once. As a result, my code was cleaner and more efficient. Instead of repeating the same mean(), var(), and sd() functions separately for every variable, I was able to use this one function across multiple datasets and columns. R makes writing such functions intuitive, simple and straightforward. Grolemund and Wickham emphasise this concept of writing modular, maintainable code in [2].

**Defensive programming**

Another takeaway from this project was learning about how to manage real-world data. This worked for me well. Even if not necessary, I would perform safety checks to ensure the code runs smoothly. A common mistake I make in my code is misspelling certain names. As a result, I put a code to check the names, for example, the column names, potentially stopping it from silently failing:

if (!all(required\_columns %in% colnames(data))) {

  stop("Error: One or more required columns are missing from the dataset.")

}

*(Figure 3: Defensive programming – Q10.R)*

I didn’t do as much defensive programming in my Python programming. This experience made me more mindful, and it’s becoming a good habit. This is especially essential in real-world data projects and strongly encouraged in Peng [3]. By making safety checks, I wouldn’t be stuck on questions wondering what exactly went wrong.

**What was challenging for you? (minimum 200-300 words excluding code, images, tables.)**

**Handling large datasets**

One challenge I faced was handling large datasets, especially with Q7.R. I had to limit the dataset manually, or else the data would be unreadable in scatterplot form:

if (nrow(piston\_data) > 500) {

  piston\_data <- piston\_data[1:500, ]

  message("Showing the first 500 rows for visualization.")

}

*(Figure 4: Handling Large Datasets – Q7.R)*

I included the message to tell the user it’s the first 500, as the scatterplot doesn’t contain all the data. Even with this, I’m not entirely sure if this is the correct way to handle this data, but the results were more readable, as shown here:

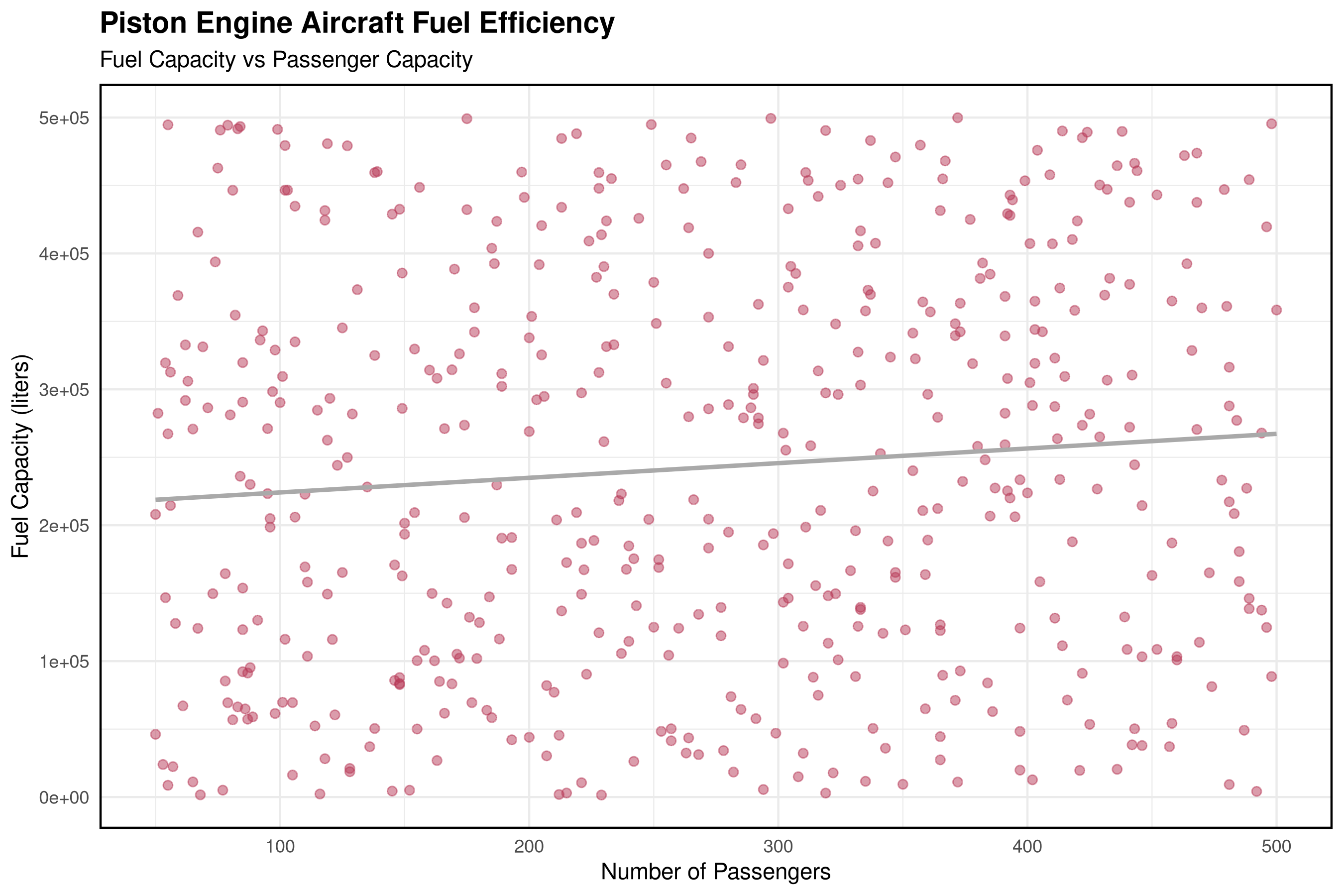
**Before**

A screen shot of a graph

AI-generated content may be incorrect.

*(Figure 5: Without limiting first 500 rows – Q7.R)*

**After**

****

*(Figure 6: With limiting first 500 rows – Q7.R)*

**Consistent code**

As I was tasked with answering five coding questions, it was hard to keep my code consistent. I constantly learned new functions and methods to solve these questions differently, which made my earlier code look slightly different from my later code. Although I did go back and review my code to implement these new functions and or methods, there are still some inconsistencies. Switching from Python to R made this even more challenging. Python doesn’t use curly braces ({}) in if statements, like R does, as shown in Figure 4. This slight difference made a noticeable impact on how I write and format my code, as I had to adjust to the new syntax and a different way of thinking about how my code should be written cleanly and readably in R. In Wilson [4] Rule 6 (“Use consistent naming and formatting”), Wilson discusses how consistent formatting and naming are essential. He also acknowledges that this can be difficult for language newcomers or when working across multiple files, as I had to do with various questions. With this information in mind, I felt more comfortable with how my code was imperfect and had some inconsistencies. However, I still tried to keep it as organised and readable as possible, especially as I gained confidence with R’s formatting style.

**Switch over to R**

During my studies, I primarily worked in Python. As a result, I found the difference between R and Python quite challenging at times. I found it personally tricky to switch over to R indexing. R indexing starts at 1, unlike Python, which begins at 0. This initially caused me to be off by one constantly and run into errors when trying to slice rows or columns. An example of this would be when I tried to get the top 10 aircraft:

ranked\_aircraft <- ranked\_aircraft[order(-ranked\_aircraft$Avg\_Ratio), ][1:10, ]

*(Figure 7: Indexing – Q10.R)*

Instead of [1:10], I used [0:9], as I would in Python, which resulted in an error. Error messages in R were sometimes quite cryptic, especially when dealing with missing columns or unmatched data types. For example, I would get error messages like this:

Error in calculate\_stats(df\_all$`Max Speed`) :

  object 'df\_all$`Max Speed`' not found

*(Figure 8: Example Error in R – Q8.R)*

I had assumed that the column was named “Max Speed”, but in the Excel dataset, it was named “Maximum Speedo (knots)”, resulting in an “object not found” error. Figure 8 taught me how strict R is when referencing column names if they contain spaces or even special characters. In contrast, Python’s panda library is far more descriptive in its error messaging than R. For example, in Python, the exact error would look like this:

KeyError: ‘Max Speed’ not found in axis

*(Figure 9: Example Error in Python based on – Q8.R)*

Figure 9 clearly tells you that the column doesn’t exist and can even provide some suggestions in some cases. It made me more cautious overall while coding in R since it won’t run unless everything is referenced correctly. Looking back, these habits helped me to write cleaner and safer code overall. However, I refreshed my memory by doing the coding exercises on Einstein that were assigned earlier in the semester. I got used to R. I also took an extra look at Unit 2 – “Importing and Exporting Data”, and Unit 3 – “Data Cleaning” on Loop, to get familiar with handling data in R.

**What did actionable knowledge did you learn? (minimum -400 words excluding code, images, tables.)**

**Teamwork and Collaboration**

 Throughout this project and the broader Introduction to R module, I learned more than just how to use R. This was my first group project. Although our team was small, it provided a valuable experience in collaboration. We communicated regularly and worked with each other’s code. When problems came up, we would meet and figure them out together. I never felt isolated or stuck on my own. This approach made managing the workload easier and made debugging much less frustrating. I was more confident using ggplot2, so I contributed mainly to that part while my colleague quickly spotted syntax issues. Our strengths complemented each other well and made a highly effective team.

**Reusable Functions**

 I also learned how to write reusable functions, which I used in Q8.R. Matloff [5] highlights the importance of functional design as a key skill in R development, particularly in Chapter 8 on writing functions and Chapter 9 on doing math and simulations. He explains the benefits of general-purpose functions and avoiding repetition. My code, shown in Figure 2, clearly reflects this idea. By reusing functions, my code became more efficient and easier to maintain. Writing less code reduces the chance of mistakes, such as prevalent ones like spelling errors. This habit also made me think more carefully about structure and readability, which improved teamwork because others could understand my code more easily. It even changed how I wrote Python. I asked myself, “Is this the best way to write this?” or “Can this be improved?” That shift in mindset helped me feel more like a developer, not just someone who wants the code to work once.

**Handling Errors**

 With five coding tasks to complete, I ran into many errors. As someone new to R, I found its error messages vague and challenging to interpret. To deal with this, I started writing defensive code by adding checks and messages that helped me understand when things were not working. This made debugging easier and helped me see what was happening behind the scenes. When I encountered an error, I could not fix, I would revisit the module materials, especially the lectures on Loop, especially the interactive units, which contained small tests and valuable videos. I would double-check if I had missed a piece of code or miswritten something. If I could not solve the issue, I looked at online resources like YouTube or Stack Overflow, where I often found helpful discussions, forums and videos. Although this trial-and-error approach was sometimes frustrating, it helped me grow more confident in problem-solving and help. Lander [6], in Chapter 1, titled “Getting Started”, describes R as “unforgiving” compared to other languages. At the beginning of the module, I did not fully understand what Lander meant by that, but now I most definitely do. These experiences improved my debugging skills and taught me valuable habits that will carry over into other languages like Python and SQL.

**References:**

[1] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*, Springer-Verlag New York, 2016.

[2] G. Grolemund and H. Wickham, *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data,* 1st ed. O’Reilly Media, 2016.

[3] R. D. Peng, *R Programming for Data Science*. Leanpub, 2016.

[4] C. Wilson, *Ten Simple Rules for Teaching Programming*, PLOS Computational Biology, vol. 12, no. 9, pp. 1–6, Sep. 2016.

[5] N. Matloff, *The Art of R Programming: A Tour of Statistical Software Design*. No Starch Press, 2011.

[6] Jared P. Lander. 2017. *R for Everyone: Advanced Analytics and Graphics* (2nd Edition) (2nd. ed.). Addison-Wesley Professional.