



ACCA7022

Advanced Statistical Programming

Data Manipulation Assignment Documentation Report

Programmer: **Mohammad Lakhi**

Student ID: **2512430**

Lecturer: **Dr. Aliyu Sambo**

Submission Date: 27th February 2026

1. Introduction

In this assignment, I simulated and analysed production information for an alien colony on a distant planet. The colony consists of six stations, each controlled by four worker teams. Each station operates for 30 sols (days on Earth).

Each team generates a daily output estimated using the following formula:

Output = 50 + (station x 3), (team x 2) + random noise.

The noise is a random whole number ranging from -5 to +5, representing daily changes in the environment.

The task required me to analyse and analyse data using loops only. Functions including `apply()`, `aggregate()`, `sum()`, `mean()`, and `max()` were not allowed. I also couldn't create my own functions. Everything needed to be done with basic for loops. After generating the data, I needed to complete five activities. This report explains how my code works for each task, includes flowcharts, and outlines testing methods. At the end, I will reflect on what I learned and how I used AI to better understand R ideas.

2. Code Explanation

2.1 Task A: Generate Production Data

The first task was to build a dataframe called `production_data` with 720 rows. Each row indicates a team operating at a single station on one sol. The columns are: Sol, Station, Team, and Output.

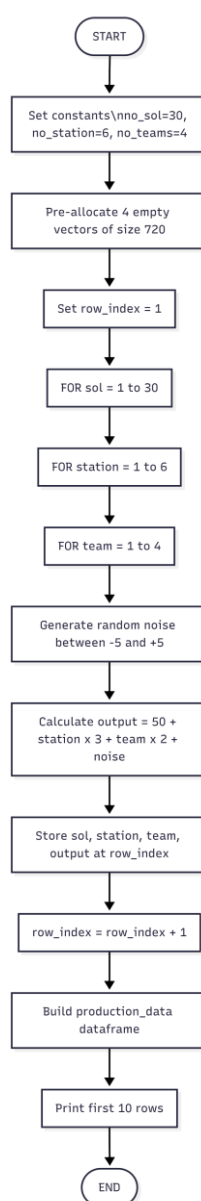
To achieve this, I used three interconnected loops. The outside loop passes through each sol (1-30), the middle loop passes through each station (1-6), and the inner loop passes through each team (1-4).

The formula is applied to each combination and the results are put in the appropriate row.

Before the loops, I created four empty columns of size 720. This is called pre-allocation. This method is faster than adding each row individually in the loop. I used a variable called `row_index` to track which row I was currently populating. Using `rbind()` to build the dataframe row by row inside the loop would have been slower due to the need for R to copy and resize the data on each iteration. To test this, I manually checked the first row. For Sol 1, Station 1, Team 1 with `set.seed(42)`, the random noise was -5, indicating:

Output: $50 + (1 \times 3) + (1 \times 2) + (-5) = 50$.

This matched the first row of the printed output exactly. I verified that the total number of rows was 720 by running `nrow(production_data)`, which was $30 \times 6 \times 4 = 720$.



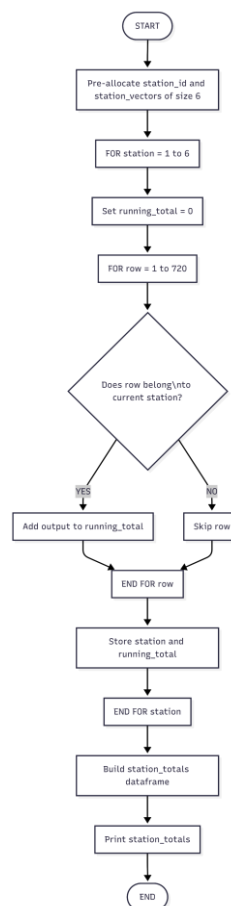
2.2 Task B: Total Production per Station

To complete this task, I calculated the total production for all 6 stations across 30 sols and 4 teams. The results were saved in a dataframe called `station_totals`, which contained two columns: `Station` and `TotalOutput`.

To achieve this, I used two loops. The outer circle passes through each station sequentially. The inner loop examines all 720 rows of `production_data` and adds each row from the current station to a running total. I have to remember to reset the running total to zero at each new station. If I didn't do this, every previous station's total would carry over to the next, resulting in incorrect results.

A simpler solution would have been using `sum()` in a single line. As `sum()` is a restricted function in this assignment, I had to use the running total approach instead. To test this, I manually calculated the predicted total for Station 1. Ignoring random noise, each sol gives $(55 + 57 + 59 + 61) = 232$.

Over 30 sols, this gives $232 \times 30 = 6960$. My code returned 6979, a difference of 19. The minor change can be attributed to random noise collecting over 120 rows as expected.



2.3 Task C: Detect Production Anomalies

In this task, I analysed each team's daily production and identified anomalies that varied by more than 10% from the station average. The results were recorded in a dataframe named `anomalies`, which contained five columns: `Sol`, `Station`, `Team`, `Output`, and `Anomaly`.

This was the most challenging task as it required three steps within loops. To calculate the daily totals for each sol and station combination, I scanned all 720 rows and added the corresponding outputs. I divided the amount by 4 to calculate the daily average. Third, I calculated how far each team's output varied from the average using the following formula:

```
deviation <- abs(team_op - daily_average)/daily_average*100
```

I used `abs()` to detect both overproduction and underproduction. If the divergence exceeded 10%, the row was tagged `TRUE`; otherwise, it was marked `FALSE`. To get the daily average, I manually divided the total by 4 instead of using `mean()`, which is limited in functionality.

To put this to the test, I checked the first anomaly manually. In Sol 1 Station 1, the four team outputs were 50, 56, 54, and 64.

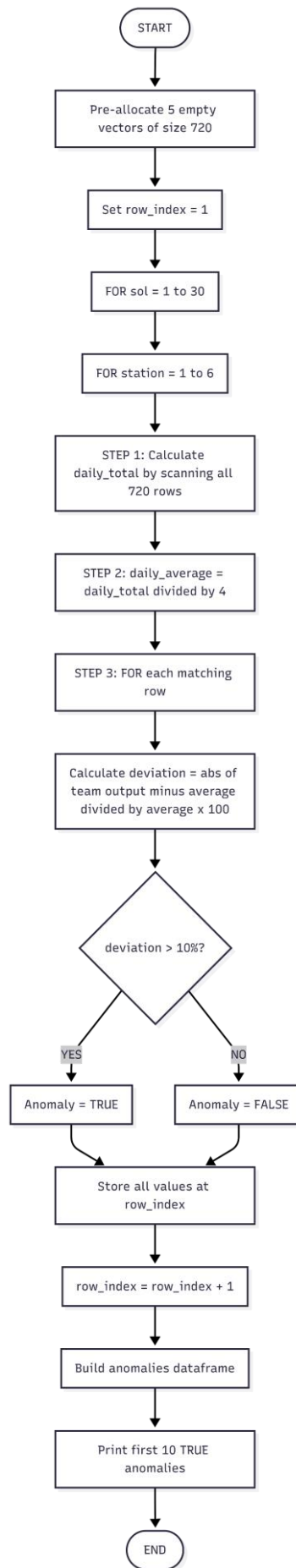
The daily total is 224, calculated as $50 + 56 + 54 + 64$. The daily average is 56 divided by 4.

Team 1 deviation: $|50 - 56| / 56 \times 100 = 10.71\%$.

The row was marked as `TRUE` since 10.71% is greater than 10%. I also checked Team 3 with output 54.

Team 3 deviation = $|54 - 56| / 56 \times 100 = 3.57\%$.

Since 3.57% is less than 10%, this was rightly declared `FALSE`. Both results were exactly the same as the code output.



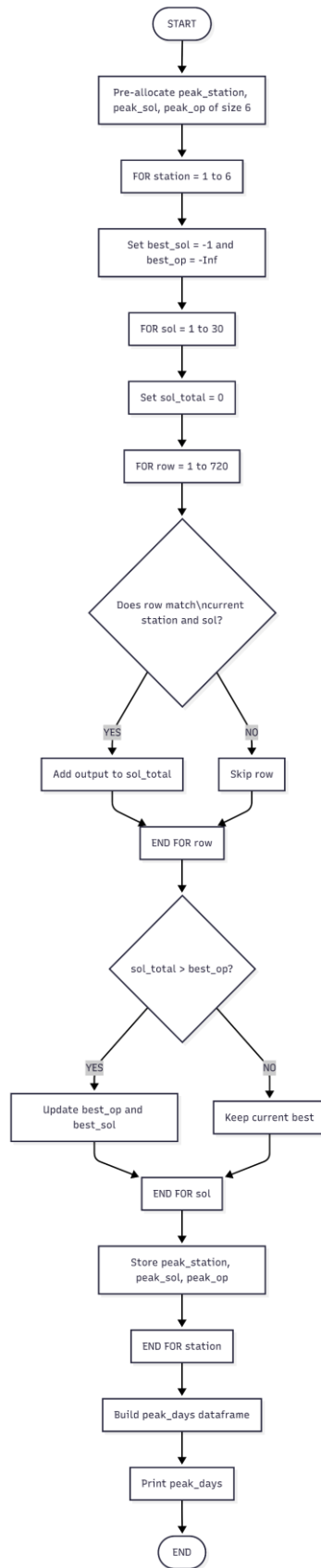
2.4 Task D: Identify Peak Productivity Days

In this task, I needed to identify the sol with the highest overall team productivity for each station. To store the results, create a dataframe called `peak_days` with three columns: `station`, `sol`, and `total output`.

To do this, I used three loops. The outer circle passes through each station. The central loop passes through each sol. The inner loop examines all 720 rows and accumulates the outputs for each station/sol combination. After the inner loop, I compare the total to the current best value (stored in a variable named `best_op`). If the new total is higher, I update `best_op` and save the source sol.

I had to pick what value to give `best_op` at the start. I set it to `-Inf`, which represents negative infinity. This ensures that the first sol total will always beat previous records and become the new benchmark. Setting it to 0 could lead to unbeatable results if all outputs were extremely low.

To determine the greatest value, I used the running maximum approach with loops instead of using restricted functions like `max()` and `which.max()`. To test this I found that Station 6 had the highest peak production whereas Station 1 had the lowest. The model predicts that higher station numbers produce more output, which is exactly what I expected. My code returned Station 1 peak = 249 and Station 6 peak = 307, confirming the right pattern. I found that each station had a unique peak sol, which makes sense given the random noise. Different days will be optimal for different stations.



2.5 Task E: Calculate Efficiency Score

I needed to add a new column named `efficiency_score` to the `production_data` dataframe. The formula provided in the assignment was:

$$\text{Efficiency Score} = \text{Output} / (\text{Station} + \text{Team})$$

This score indicates a team's efficiency in producing output based on their station and team size. Higher scores indicate more efficiency. Teams at lower-numbered stations have higher efficiency scores due to smaller components, but lower raw production.

I used a single loop to apply the calculation to each entry in `production_data`. Before the loop, I inserted a new column pre-filled with zeros. This maintains the code consistent with previous jobs.

In R, this calculation may be done in a single line, eliminating the need for a loop:

```
production_data$efficiency_score[row]<-  
production_data$output[row]/(production_data$station[row]+production_data$  
team[row])
```

I used a loop to align the code with the rest of the assignment and demonstrate my ability to process data sequentially.

To test, I manually calculated efficiency scores for the first two rows:

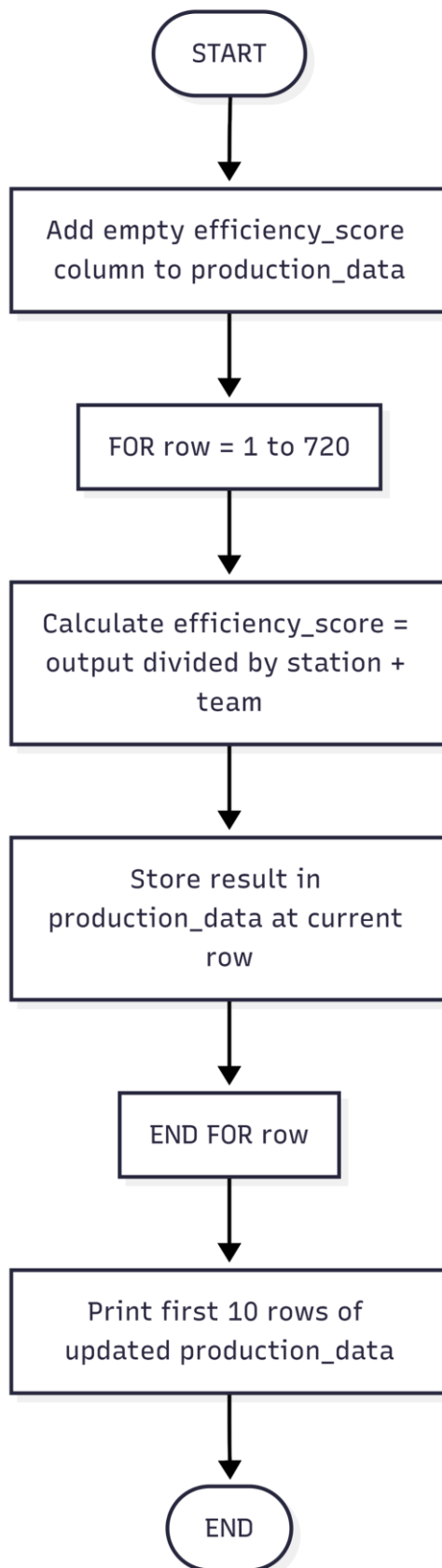
Row 1: Station 1, Team 1, and Output 50

$$50 / (1 + 1) = 50 / 2 = 25.0$$

Row 2: Station 1, Team 2, Output 56

$$56 / (1 + 2) = 56 / 3 = 18.667$$

The algorithm was accurately applied to all 720 rows, since both values matched the displayed output exactly.



3. Reflection

Before doing this assignment, I had experience using loops in Python and JavaScript. I was already familiar with the concept of loops. Applying these techniques in R and learning how it differs from other programming languages was new.

The most important thing I learned was pre-allocation. In other languages, I used to add items to a list in a loop without regard for memory. In R, this is wasteful as the vector needs to be duplicated and resized each time. Creating empty vectors of the appropriate size before looping and filling them with a row index was a new concept for me, but it made sense after I realised why.

The most difficult challenge was challenge C. I had to carefully consider where to reset the `row_index` and `daily_total` within nested loops, which took three stages. Improper development would result in incorrect findings, needing testing and verification to ensure accuracy.

In Task D, I accidentally entered "&" instead of "\$" when accessing a dataframe column. This resulted in an error that took some time to recognise. Although it was a little error, it highlighted the need of understanding R's syntax, even if the logic was clear. If I had to do this assignment over, I would plan the loop structure on paper first before writing code. Identifying which variables need to be reset at each level of the loop might save time when debugging.

I used Claude and Chatgpt, AI assistants, to help with this task. I used it in order to understand R concepts like pre-allocation and the \$ operator, compare output values to expected results, and improve code comments. The AI didn't write the code for me. Instead, it clarified flaws and took me through the logic step by step. Using AI as a learning tool has helped me understand ideas better than simply providing answers.