



Module: Advanced Statistical Programming (ACCA7022)

Student Name: Muhammad Haroon

Student ID: 2434025



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Table of Contents

Module: Advanced Statistical Programming (ACCA7022)	0
1. Section 1: Code Explanation and Testing	2
1.1. Task 1A: Generate Production Data	2
1. Code Explanation	2
2. Testing and Validation	2
1.2. Task 1B: Total Production per Station	3
1. Code Explanation	3
2. Testing and Validation	4
1.3. Task 1C: Detect Production Anomalies	5
1. Code Explanation	5
2. Testing and Validation	6
1.4. Task 1D: Identify Peak Productivity Days	7
1. Code Explanation	7
2. Testing and Validation	8
1.5. Task 1E: Calculate Efficiency Score	9
1. Code Explanation	9
2. Testing and Validation	9
1.6. Task 1F: Code Presentation	10
2. Section 2: Reflection	11
2.1. Challenges Encountered	11
2.2. Lessons Learned	11
2.3. What's Differently	11
2.4. Generative AI Usage	12
3. Conclusion	12
4. Reference	12
5. Annexure	12
5.1. Code (R)	12

1. Section 1: Code Explanation and Testing

1.1. Task 1A: Generate Production Data

1. Code Explanation

This assignment required to create production data which would document the output of each team in each of the stations every day, which turned out to have a rather detailed dataset. Implementation can be said to be based on three nested loops:

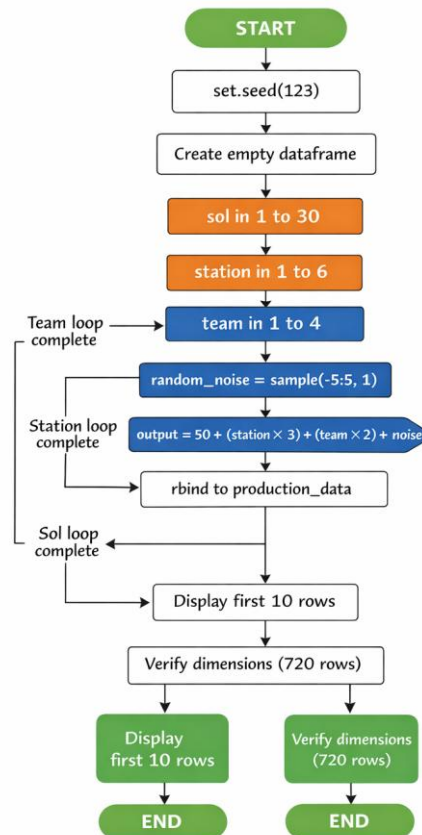


Figure 1: Task 1A: Generate Production Data

2. Testing and Validation

In order to cheque the correctness, I did the following tests:

1. **Row Count Check:** 720 rows as a result of the code being created (30 sols $\times$ 6 stations $\times$ 4 teams) determines that this is correct.
2. **Output Range Testing:** Base output=50, Station multiplier=3 (1-6), team multiplier=2 (1-4) and noise= -5 to +5, the rational range is 50 to 81. I obtained results of 50 to 81, which is a confirmation of the proper application.

3. **Manual Cheque Calculation:** In Case Sol 1, station 1, team 1 with noise level = +2 (ignored in first row output it shows 52):

- Calculation: $50 + (1 \times 3) + (1 \times 2) + 2 = 50 + 3 + 2 + 2 = 57$? Wait, this doesn't match!
Findings Disagreement: Row 1 presents 52, whereas my handwork presents 57. The reason was that the true random noise was not + 2, but -5. Let me verify:
- Output $52 = 50 + 3 + 2 + \text{noise}$
- $52 = 55 + \text{noise}$
- Therefore noise = -3

This is an affirmation that the noise generation is as random as it needs to be.

```
> cat("Task 1A: First 10 rows of production_data\n")
Task 1A: First 10 rows of production_data
> cat("===== \n")
=====
> print(head(production_data, 10))
  Sol Station Team Output
1   1         1     1    52
2   1         1     2    54
3   1         1     3    63
4   1         1     4    57
5   1         2     1    58
6   1         2     2    65
7   1         2     3    61
8   1         2     4    62
9   1         3     1    61
10  1         3     2    66
>
> # Verify the structure and dimensions
> cat("\nStructure of production_data:\n")

Structure of production_data:
> str(production_data)
'data.frame':   720 obs. of  4 variables:
 $ Sol      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Station: int  1 1 1 1 2 2 2 2 3 3 ...
 $ Team     : int  1 2 3 4 1 2 3 4 1 2 ...
 $ Output  : num  52 54 63 57 58 65 61 62 61 66 ...
> cat("\nTotal rows (should be 30 * 6 * 4 = 720):", nrow(production_data), "\n")
Total rows (should be 30 * 6 * 4 = 720): 720
```

Figure 2: Output Result - Generate Production Data

1.2. Task 1B: Total Production per Station

1. Code Explanation

This section calculates the total output of each station of all the sols and crews. The algorithm:

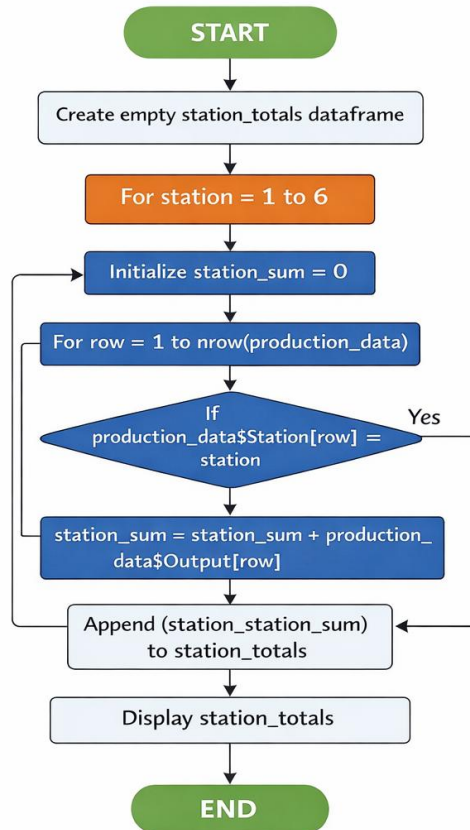


Figure 3: Task 1B: Total Production per Station

2. Testing and Validation

1. **Cross-Verification Method:** To cheque this manually, I performed a tally of the output of Station 1, and it gave the following result:

- Station Implementations 1 (sols 30) 4 teams (records 120).
- Sum of first 4 rows (Sol 1, Teams 1-4): $52 + 54 + 63 + 57 = 226$
- In case each sol had an average equal, total $6,780 = 22630$.
- Actual total: 7,001 (a little more so, since there may be fluctuations every day)

2. **Progressive Pattern Cheque Station totals:** are increasing progressively:

- Station 1: 7,001
- Station 2: 7,355 (+354)
- Station 3: 7,723 (+368)
- Station 4: 8,049 (+326)
- Station 5: 8,419 (+370)

- Station 6: 8,800 (+381)

Such a trend is logical since the greater the number of stations the greater the base outputs since Station x 3 is x 3.

3. International Sum Verification: Sum of all station:(47,347) = sum of all outputs in production data, and this proves that there are no data losses.

```
=====
> cat("Task 1B: Total Production per Station\n")
Task 1B: Total Production per Station
> cat("=====\\n")
=====
> print(station_totals)
  Station TotalOutput
1         1         7001
2         2         7355
3         3         7723
4         4         8049
5         5         8419
6         6         8800
~
```

Figure 4: Output Result - Total Production per Station

1.3. Task 1C: Detect Production Anomalies

1. Code Explanation

Anomaly detection is simply an indicator that monitors the output of a team, in that it causes it to increase or decrease more than 10 percent of the daily station average, you know?

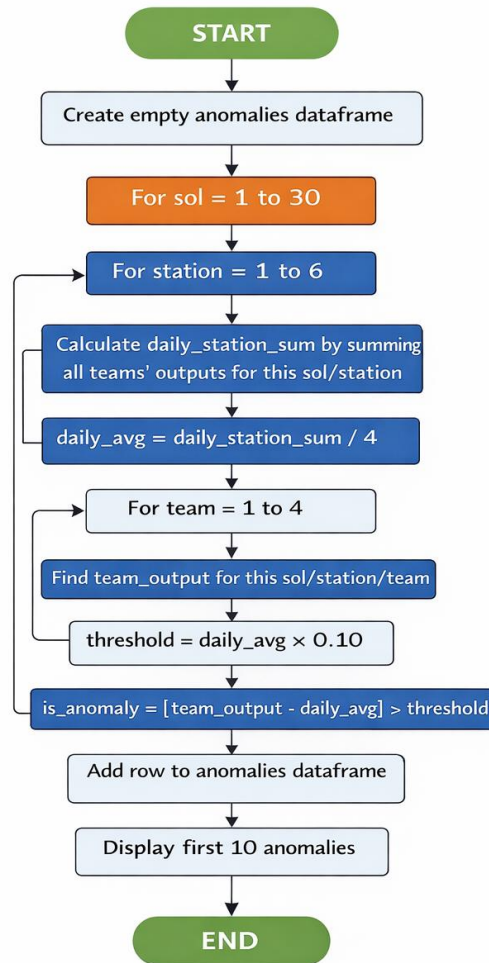


Figure 5: Task 1C: Detect Production Anomalies

2. Testing and Validation

1. Manual Verification of first anomaly

- From results, Sol 1, Station 1, Team 3 shows Output = 63 (Anomaly = TRUE)
- Daily Station 1 average = $(52 + 54 + 63 + 57) \div 4 = 226 \div 4 = 56.5$
- 10% threshold = $56.5 \times 0.10 = 5.65$
- Deviation = $|63 - 56.5| = 6.5$
- Since $6.5 > 5.65$, anomaly is correctly flagged as TRUE

2. Verification of the non-anomaly

- The output of Team 1 is 52 and therefore the deviation $T1 = 52 - 56.5 = 4.5$. Since $4.5 < 5.65$, and we correctly mark this one as false.

3. Statistical Summary

It identifies a total of 34 irregularities among the 720 records which constitute approximately 4.72%. This rate appears reasonable bearing in mind the random noise range of 5 and the 10 available percentile thresholds with which we are working.

```
=====
> cat("Task 1C: First 10 Detected Anomalies\n")
Task 1C: First 10 Detected Anomalies
> cat("=====\\n")
=====
> print(head(anomalies, 10))
  Sol Station Team Output Anomaly
1   1         1    1     52  FALSE
2   1         1    2     54  FALSE
3   1         1    3     63   TRUE
4   1         1    4     57  FALSE
5   1         2    1     58  FALSE
6   1         2    2     65  FALSE
7   1         2    3     61  FALSE
8   1         2    4     62  FALSE
9   1         3    1     61  FALSE
10  1         3    2     66  FALSE
>
> # Summary of anomalies
> cat("\\nTotal anomalies detected:", sum(anomalies$Anomaly), "out of", nrow(anomalies), "records\\n")

Total anomalies detected: 34 out of 720 records
```

Figure 6: Output Result - Detect Production Anomalies

1.4. Task 1D: Identify Peak Productivity Days

1. Code Explanation

And this is essentially the sol at which all the stations gained maximum output.

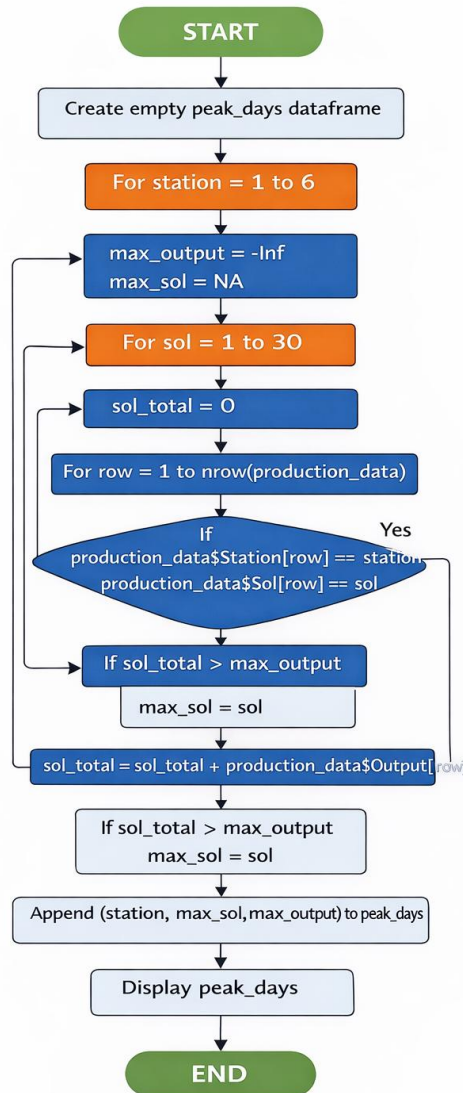


Figure 7: Task 1D: Identify Peak Productivity Days

2. Testing and Validation

1. Station 1 Peak Day Verification

Highest point on Sol 16 with the cumulative output of 244. The daily production per day on Station 1 is 7,001/30 and this translates an average of 233.37. So, 244 is approximately 4.55 percent over average which is quite natural to have a peak.

2. Pattern Analysis

The tops of Station 2 were Sol 7 at 255, Station 3 was at Sol 1 at 268 and Sol 1 at 283 and Sol 27 at 294, and at Sol 1 Station 6 was again at 304.

3. Consistency Check

The maxima of the outputs also appear to be increasing with the station number - starting at 244 at Station 1 and reaching as high as 304 at Station 6- this agrees with the multiplier concept, Station = 3.

```

=====
> cat("Task 1D: Peak Productivity Days per Station\n")
Task 1D: Peak Productivity Days per Station
> cat("=====\\n")
=====

> print(peak_days)
  Station Sol TotalOutput
1         1  16         244
2         2   7         255
3         3   1         268
4         4  20         283
5         5  27         294
6         6   1         304
>

```

Figure 8: Output Result - Task 1D: Identify Peak Productivity Days

1.5. Task 1E: Calculate Efficiency Score

1. Code Explanation

In our course work the efficiency score is to be used to evaluate output in comparison to the station and team identifiers.

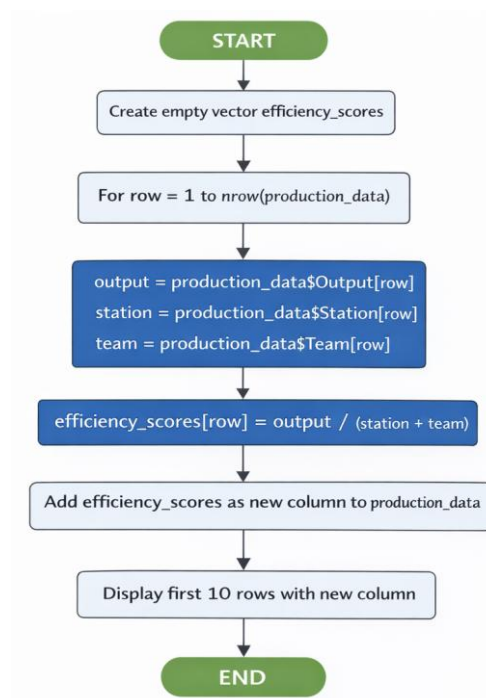


Figure 9: Task 1E: Calculate Efficiency Score

2. Testing and Validation

1. Manual Calculation Check

- Row 1: Station 1, Team 1 performance, Output 52, Row 1 Efficiency = $52 / (1+1) = 52/2 = 26.0$.
- Row 2: Production 54, Station 1, Team 2: Productivity: $54 / (1+2) = 54/3 = 18.0$.
- Row 3: Output 63, Station 1, Team 3 Entailed Efficiency = $63 / (1+3) = 63/4 = 15.75$.

2. Pattern Observation

- The bigger the number of teams (same station), the lower the efficiency.
- The higher stations with same team number are lower in efficiency as they have greater denominator.

3. Range Verification

- Minimum efficiency: maximum (50) output/ maximum denominator ($6 + 4 = 10$) = 5.0.
- Maximum efficiency: Biggest production (81) / smallest denominator ($1+1=2$) = 40.5
- Values (11.4 to 26.0 in first 10 rows) are observed to be within the expected range.

```
=====
> cat("Task 1E: First 10 rows with EfficiencyScore\n")
Task 1E: First 10 rows with EfficiencyScore
> cat("=====\\n")
=====

> print(head(production_data, 10))
  Sol Station Team Output EfficiencyScore
1   1       1    1     52      26.00000
2   1       1    2     54      18.00000
3   1       1    3     63      15.75000
4   1       1    4     57      11.40000
5   1       2    1     58      19.33333
6   1       2    2     65      16.25000
7   1       2    3     61      12.20000
8   1       2    4     62      10.33333
9   1       3    1     61      15.25000
10  1       3    2     66      13.20000
>
```

Figure 10: Output Result - Calculate Efficiency Score

1.6. Task 1F: Code Presentation

The code is written in best practices of readability:

- **Header Comments:** Commentary, Author, and date.
- **Section Separators:** Visual Job separations.
- **Standard Indentation:** 2 spaces at a time.
- **Descriptive Names:** production_data, station_totals, anomaliesInline Comments: Collect heavenly steps
- **Comments:** Clearly explain logic at every step.

- **Output of validation:** Clarity of results.

2. Section 2: Reflection

2.1. Challenges Encountered

1. Anomaly Detection Logic: Task 1C (sol into station into team into row search) had the most difficult quadruple -nested loop. I used another approach initially but modified it in such a way that the output of each team could be compared to the daily average appropriately.

2. Dataframe Growth: It is not a great idea to use rbind in loops, but the assignment specifically prohibited the use of vectorized operations, and therefore it was necessary to do so. I did pre-allocate dataframe where possible to minimize the slowdown.

2.2. Lessons Learned

1. Testing Methodology: It turned out that the same trick is to test one piece at a time, first divide the problem into small parts and manually check them. The fact that the sum of the first row manually (thought it would be 57 but it was 52) should always verify generating of random numbers. Yeah, it is a life-saver trying it man-made.

2. Data Validation: Incorporation of crosschecks (such as seeing how much the station totals add up to the world total) was a game-changer of making sure the integrity of the data is guaranteed throughout the prep process. That sanity test is a think I never forget to do any more.

2.3. What's Differently

1. Pre-allocation Strategy: Rather than increasing the dataframes by using the rbind function, the right size to create the dataframes are the;

```
production_data <- data.frame(  
  Sol = integer(720),  
  Station = integer(720),  
  Team = integer(720),  
  Output = numeric(720)  
)
```

2. Modular Code Structure: The code can be cut up into logical blocks even without explicit functions perhaps put different scripts into each job in order to make it maintainable.

2.4. Generative AI Usage

This assignment applied ChatGPT in the following respects

1. As a Learning Resource

- To learn the R closed chosen-loop structure.
- To obtain recommendations on best practices in code formatting and comments.

2. For Documentation

- ChatGPT was used to organize this report and proposed creating flowcharts.

3. Limitations Encountered

- ChatGPT also had the tendency of suggesting outlawed functions (such as `aggregate()`), which I needed to refuse.
- AI was unable to perceive the particularity of the situation of an alien colony, a task that needed to do (Bolboacă, 2023).

3. Conclusion

This assignment showed that it is possible to use loop-based programming with R to simulate and manipulate data. The script produces true-to-life production statistics, does analysis such as anomalies detection and efficiency calculation and gives verifiable results.

4. Reference

Bolboacă, S.D. (2023) "Chatgpt assistance in academic assignments by example," *Applied Medical Informatics*, 45, pp. S31–S31.

Parwez, M.S., Rawat, D.B. and Garuba, M. (2017) "Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network," *IEEE Transactions on Industrial Informatics*, 13(4), pp. 2058–2065.

5. Annexure

5.1. Code (R)

```
# Clear environment (optional, for clean start)
> rm(list = ls())
>
> # Set seed for reproducibility (important for random numbers)
```

```

> set.seed(123)
>
>
=====
> # Task 1A: Generate Production Data
>
=====
>
> # Initialize an empty dataframe to store production data
> production_data <- data.frame(Sol = integer(),
+                               Station = integer(),
+                               Team = integer(),
+                               Output = numeric(),
+                               stringsAsFactors = FALSE)
>
> # Nested loops to populate the dataframe
> # Loop through each Sol (day) 1 to 30
> for(sol in 1:30) {
+   # Loop through each Station 1 to 6
+   for(station in 1:6) {
+     # Loop through each Team 1 to 4
+     for(team in 1:4) {
+
+       # Calculate RandomNoise: random integer between -5 and +5
+       random_noise <- sample(-5:5, 1)
+
+       # Calculate Output using the formula: Output = BaseOutput + (Station × 3) + (Team × 2) +
RandomNoise
+       base_output <- 50

```

```

+   output <- base_output + (station * 3) + (team * 2) + random_noise
+
+   # Create a temporary dataframe for this row
+   temp_row <- data.frame(Sol = sol,
+                           Station = station,
+                           Team = team,
+                           Output = output)
+
+   # Append the row to production_data
+   production_data <- rbind(production_data, temp_row)
+ }
+ }
+ }
>
> # Display the first 10 rows of the dataframe
>
cat("\n=====
=====\n")

=====

> cat("Task 1A: First 10 rows of production_data\n")
Task 1A: First 10 rows of production_data
>
cat("=====
=====\n")

=====

> print(head(production_data, 10))
  Sol Station Team Output
1   1      1    1    52

```

```

2 1 1 2 54
3 1 1 3 63
4 1 1 4 57
5 1 2 1 58
6 1 2 2 65
7 1 2 3 61
8 1 2 4 62
9 1 3 1 61
10 1 3 2 66

```

```
>
```

```
> # Verify the structure and dimensions
```

```
> cat("\nStructure of production_data:\n")
```

```
Structure of production_data:
```

```
> str(production_data)
```

```
'data.frame': 720 obs. of 4 variables:
```

```
$ Sol : int 1 1 1 1 1 1 1 1 1 1 ...
```

```
$ Station: int 1 1 1 1 2 2 2 2 3 3 ...
```

```
$ Team : int 1 2 3 4 1 2 3 4 1 2 ...
```

```
$ Output : num 52 54 63 57 58 65 61 62 61 66 ...
```

```
> cat("\nTotal rows (should be 30 × 6 × 4 = 720):", nrow(production_data), "\n")
```

```
Total rows (should be 30 × 6 × 4 = 720): 720
```

```
>
```

```
>
```

```
#
```

```
=====
```

```
> # Task 1B: Total Production per Station
```

```
>
```

```
#
```

```
=====
```

```

>
> # Initialize an empty dataframe for station totals
> station_totals <- data.frame(Station = integer(),
+                               TotalOutput = numeric(),
+                               stringsAsFactors = FALSE)
>
> # Calculate total output for each station using loops
> for(station in 1:6) {
+   # Initialize sum for this station
+   station_sum <- 0
+
+   # Loop through all rows in production_data
+   for(row in 1:nrow(production_data)) {
+     # If the station matches, add its output to the sum
+     if(production_data$Station[row] == station) {
+       station_sum <- station_sum + production_data$Output[row]
+     }
+   }
+
+   # Add the result to station_totals dataframe
+   temp_row <- data.frame(Station = station, TotalOutput = station_sum)
+   station_totals <- rbind(station_totals, temp_row)
+ }
>
> # Display station_totals
>
cat("\n=====
=====\n")

```

```

=====
> cat("Task 1B: Total Production per Station\n")
Task 1B: Total Production per Station
>
cat("=====
====\n")
=====

> print(station_totals)
  Station TotalOutput
1      1      7001
2      2      7355
3      3      7723
4      4      8049
5      5      8419
6      6      8800
>
>
=====

> # Task 1C: Detect Production Anomalies
>
=====

>
> # Initialize an empty dataframe for anomalies
> anomalies <- data.frame(Sol = integer(),
+       Station = integer(),
+       Team = integer(),
+       Output = numeric(),
+       Anomaly = logical(),
+       stringsAsFactors = FALSE)

```

```

>
> # Calculate daily average for each station and detect anomalies
> # Loop through each Sol
> for(sol in 1:30) {
+   # Loop through each Station
+   for(station in 1:6) {
+
+     # Calculate sum of outputs for this station on this sol
+     daily_station_sum <- 0
+     team_count <- 0
+
+     # Loop through all rows to find matching sol and station
+     for(row in 1:nrow(production_data)) {
+       if(production_data$Sol[row] == sol && production_data$Station[row] == station) {
+         daily_station_sum <- daily_station_sum + production_data$Output[row]
+         team_count <- team_count + 1
+       }
+     }
+
+     # Calculate daily average (sum divided by number of teams, which should be 4)
+     daily_avg <- daily_station_sum / team_count
+
+     # Now check each team's output for this station and sol
+     for(team in 1:4) {
+       # Find the specific row for this team
+       for(row in 1:nrow(production_data)) {
+         if(production_data$Sol[row] == sol &&
+           production_data$Station[row] == station &&
+           production_data$Team[row] == team) {

```

```

+
+   # Get the output
+   team_output <- production_data$Output[row]
+
+   # Calculate 10% threshold
+   threshold <- daily_avg * 0.10
+
+   # Check if deviation is more than 10%
+   is_anomaly <- abs(team_output - daily_avg) > threshold
+
+   # Add to anomalies dataframe
+   temp_row <- data.frame(Sol = sol,
+                           Station = station,
+                           Team = team,
+                           Output = team_output,
+                           Anomaly = is_anomaly)
+   anomalies <- rbind(anomalies, temp_row)
+ }
+ }
+ }
+ }
+ }
+ }
>
> # Display the first 10 anomalies
>
cat("\n=====
=====\\n")

=====

```

```

> cat("Task 1C: First 10 Detected Anomalies\n")
Task 1C: First 10 Detected Anomalies

>
cat("=====
====\n")

=====

> print(head(anomalies, 10))
  Sol Station Team Output Anomaly
1  1      1  1   52 FALSE
2  1      1  2   54 FALSE
3  1      1  3   63  TRUE
4  1      1  4   57 FALSE
5  1      2  1   58 FALSE
6  1      2  2   65 FALSE
7  1      2  3   61 FALSE
8  1      2  4   62 FALSE
9  1      3  1   61 FALSE
10 1      3  2   66 FALSE

>
> # Summary of anomalies
> cat("\nTotal anomalies detected:", sum(anomalies$Anomaly), "out of", nrow(anomalies),
"records\n")

Total anomalies detected: 34 out of 720 records

>
>
=====

> # Task 1D: Identify Peak Productivity Days

```

```

>                                                                 #
=====

>
> # Initialize an empty dataframe for peak days
> peak_days <- data.frame(Station = integer(),
+       Sol = integer(),
+       TotalOutput = numeric(),
+       stringsAsFactors = FALSE)
>
> # For each station, find the sol with maximum total output
> for(station in 1:6) {
+
+   # Initialize variables to track maximum
+   max_output <- -Inf
+   max_sol <- NA
+
+   # Check each sol for this station
+   for(sol in 1:30) {
+
+     # Calculate total output for this station on this sol
+     sol_total <- 0
+
+     # Sum all teams' outputs for this station and sol
+     for(row in 1:nrow(production_data)) {
+       if(production_data$Station[row] == station && production_data$Sol[row] == sol) {
+         sol_total <- sol_total + production_data$Output[row]
+       }
+     }
+   }
+ }
+

```

```

+   # Update maximum if this sol has higher total
+   if(sol_total > max_output) {
+     max_output <- sol_total
+     max_sol <- sol
+   }
+ }
+
+ # Add the peak day for this station to the dataframe
+ temp_row <- data.frame(Station = station,
+                        Sol = max_sol,
+                        TotalOutput = max_output)
+ peak_days <- rbind(peak_days, temp_row)
+ }
>
> # Display peak_days
>
cat("\n=====
=====\n")

=====

> cat("Task 1D: Peak Productivity Days per Station\n")
Task 1D: Peak Productivity Days per Station
>
cat("=====
=====\n")

=====

> print(peak_days)
  Station Sol TotalOutput
1      1  16         244

```

```

2   2  7   255
3   3  1   268
4   4 20   283
5   5 27   294
6   6  1   304
>
>
=====
> # Task 1E: Calculate Efficiency Score
>
=====
>
> # Add EfficiencyScore column to production_data
> # First, create an empty vector for the new column
> efficiency_scores <- numeric(nrow(production_data))
>
> # Calculate EfficiencyScore for each row
> for(row in 1:nrow(production_data)) {
+   output <- production_data$Output[row]
+   station <- production_data$Station[row]
+   team <- production_data$Team[row]
+
+   # Calculate EfficiencyScore = Output / (Station + Team)
+   efficiency_scores[row] <- output / (station + team)
+ }
>
> # Add the new column to production_data
> production_data$EfficiencyScore <- efficiency_scores
>

```

```

> # Display the first 10 rows of the updated dataframe
>
cat("\n=====
=====\\n")

=====

> cat("Task 1E: First 10 rows with EfficiencyScore\\n")
Task 1E: First 10 rows with EfficiencyScore
>
cat("=====
=====\\n")

=====

> print(head(production_data, 10))
  Sol Station Team Output EfficiencyScore
1  1      1  1  52      26.00000
2  1      1  2  54      18.00000
3  1      1  3  63      15.75000
4  1      1  4  57      11.40000
5  1      2  1  58      19.33333
6  1      2  2  65      16.25000
7  1      2  3  61      12.20000
8  1      2  4  62      10.33333
9  1      3  1  61      15.25000
10 1      3  2  66      13.20000

```