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

Smart Warehouse Inventory Simulation

Module (L7**):** Advanced Statistical Programming

Module Code: ACCA7022

Lecturer: Dr. Aliyu Sambo

Student Name: Gobinath Gothandapani

Student ID: 2538599

Department: MSc Data and Artificial Intelligence

Contents

1. Introduction **3**

2. Section A: Explanation of Code and Logic **3**

2.1 Setup and data storage (A1–A3) 4

2.2 Data generation with nested loops (A4) 5

2.3 Summaries: zone totals and category averages (Parts B and C) 5

2.4 Anomaly detection (Part D) 5

2.5 Peak productive day per zone (Part E) 6

2.6 Sanity check and validation suite 7

2.7 Overall logic 7

3. Section B: Justification of Programming Choices **8**

4. Section C: Testing and Validation **8**

5. Section D: Efficiency and Scalability Reflection **8**

6. Section E: Reflection on Learning and Use of Generative AI **8**

7. Conclusion **8**

1. Introduction

This report documents and evaluates an R script that simulates 28 days of warehouse processing across five zones and six product categories. It generates the data, summarises units and processing time, flags unusual throughput, identifies each zone's peak day, and checks its own results.

2. Section A: Explanation of Code and Logic

The script has five labelled parts (A to E), followed by a sanity check and ten validation tests (T1 to T10). Table 1 summarises the stages and Figure 1 shows the overall flow.

Table 1: Structure of the R script and the loops used in each stage.

Stage (script label)	Purpose	Loop structure	Output
Setup (A1–A3)	Set seed, store settings, create empty table	None	warehouse_data (840 × 5)
Data generation (A4)	One record per day–zone–category	3 nested loops (840 passes)	warehouse_data (filled)
Preview (A5–A6)	Print first 10 rows and dimensions	None	Console output
Part B	Units per zone	Zones × 840 rows	zone_totals
Part C	Average time per category	Categories × 840 rows	category_average_time
Part D	Flag units ±15% of day–zone average	Day × zone × row loops; record loop; count loop	processing_anomalies
Part E	Most productive day per zone	Zone × day × row loops	peak_zone_days
Sanity check	Raw total against zone totals	Two single loops	PASS or FAILED message
Validation (T1–T10)	Ten independent checks	Loops and if-tests	Count of tests passed

Figure 1 gives the overall picture. Hexagons are loops, diamonds are decisions, and dashed arrows return control to the start of a loop.

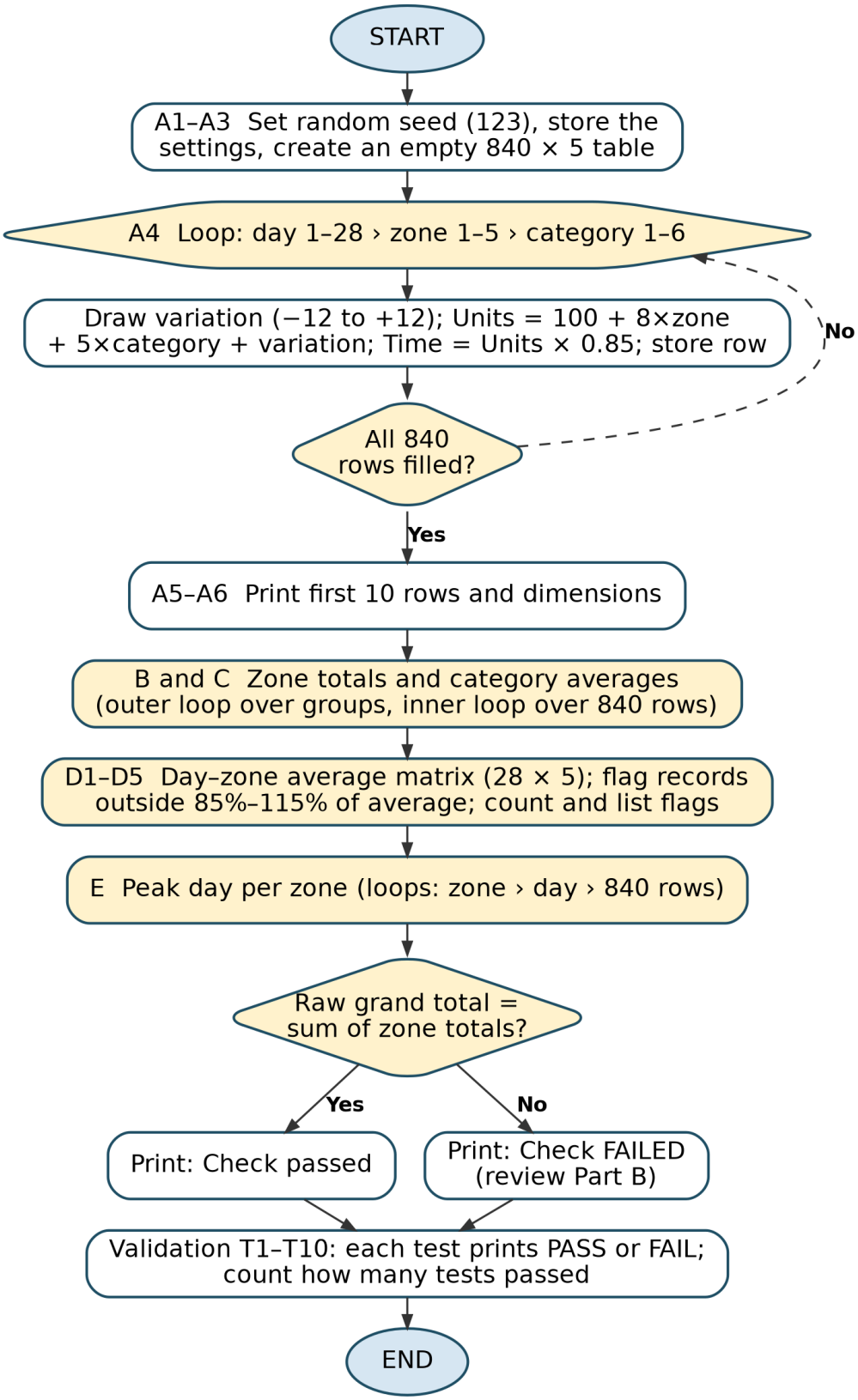

Figure 1: Overall flowchart of the script.

2.1 Setup and data storage (A1–A3)

set.seed(123) fixes the random numbers, so every run gives identical data. The settings (28 days, 5 zones, 6 categories, base of 100 units, 0.85 minutes per unit) are stored as named variables, and the 840

rows are calculated as 28 × 5 × 6. warehouse_data is a data frame because its columns mix integers and decimals. Its columns are pre-allocated at length 840, so the loops overwrite rows rather than growing the table.

2.2 Data generation with nested loops (A4)

Three nested loops run over day (1 to 28), zone (1 to 5) and category (1 to 6). The category loop is innermost, so rows are stored in loop order and every combination's position is known. A counter, row_counter, points to the current row. Each pass draws a variation with sample(-12:12, size = 1), sets UnitsProcessed = 100 + 8 × zone + 5 × category + variation, and sets ProcessingTime = UnitsProcessed × 0.85. Zone 5 therefore averages 32 units more than zone 1.

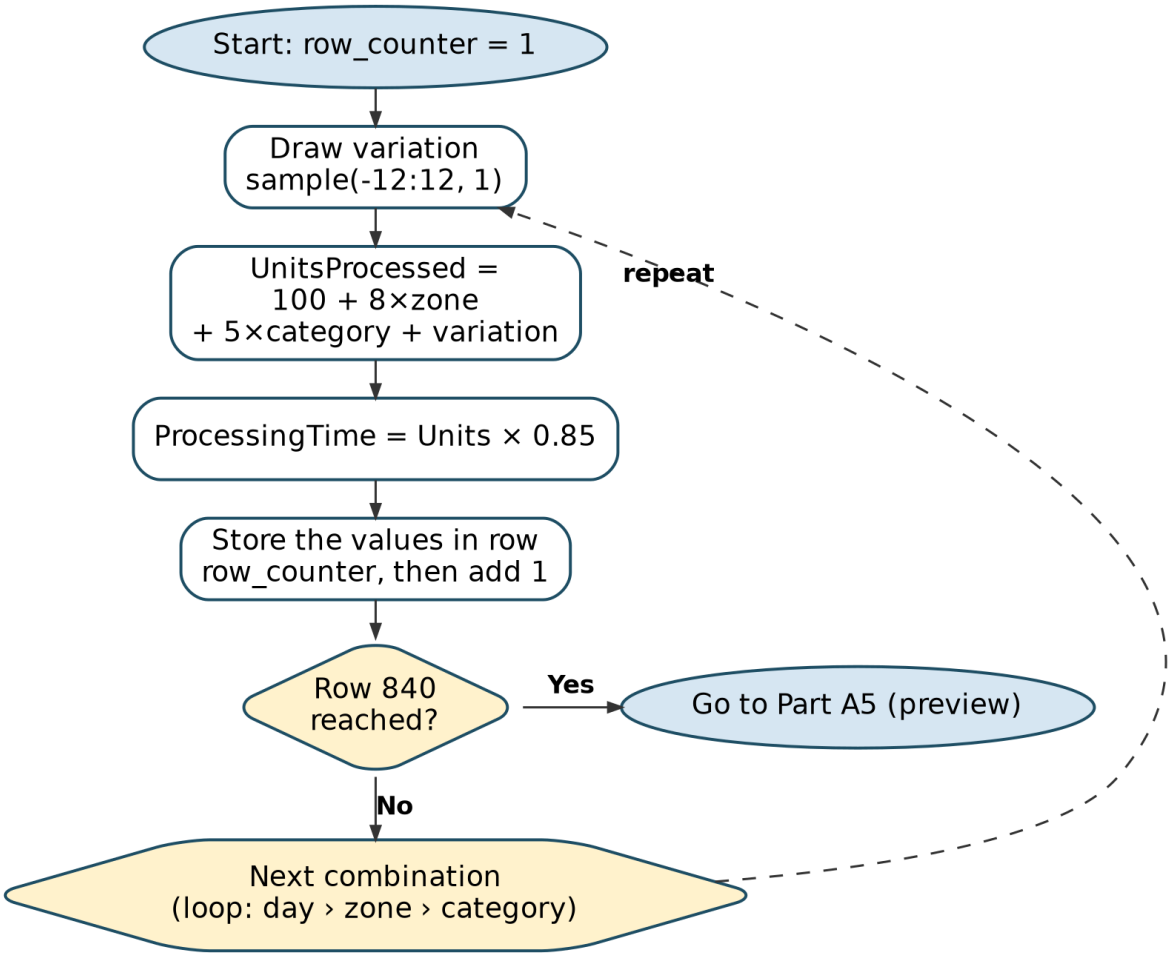

Figure 2: Part A4 data-generation loop.

2.3 Summaries: zone totals and category averages (Parts B and C)

Both parts use an outer loop over the groups and an inner loop over all 840 rows. An if-test adds a row to a running total only when it belongs to the group. Part C keeps two accumulators, total time and record count, and divides at the end. round(…, 2) is applied only when the result is stored, so the running totals keep full precision.

2.4 Anomaly detection (Part D)

Step D1 builds a 28 × 5 matrix of day–zone averages by scanning the rows for each of the 140 pairs, so daily_zone_average[3, 2] gives day 3, zone 2. Step D3 loops over the 840 records and flags a record TRUE when its units exceed 1.15 times the average or fall below 0.85 times it. The two tests are joined

by the OR operator (||). The flag uses the unrounded average. Steps D4 and D5 count and list the flagged rows.

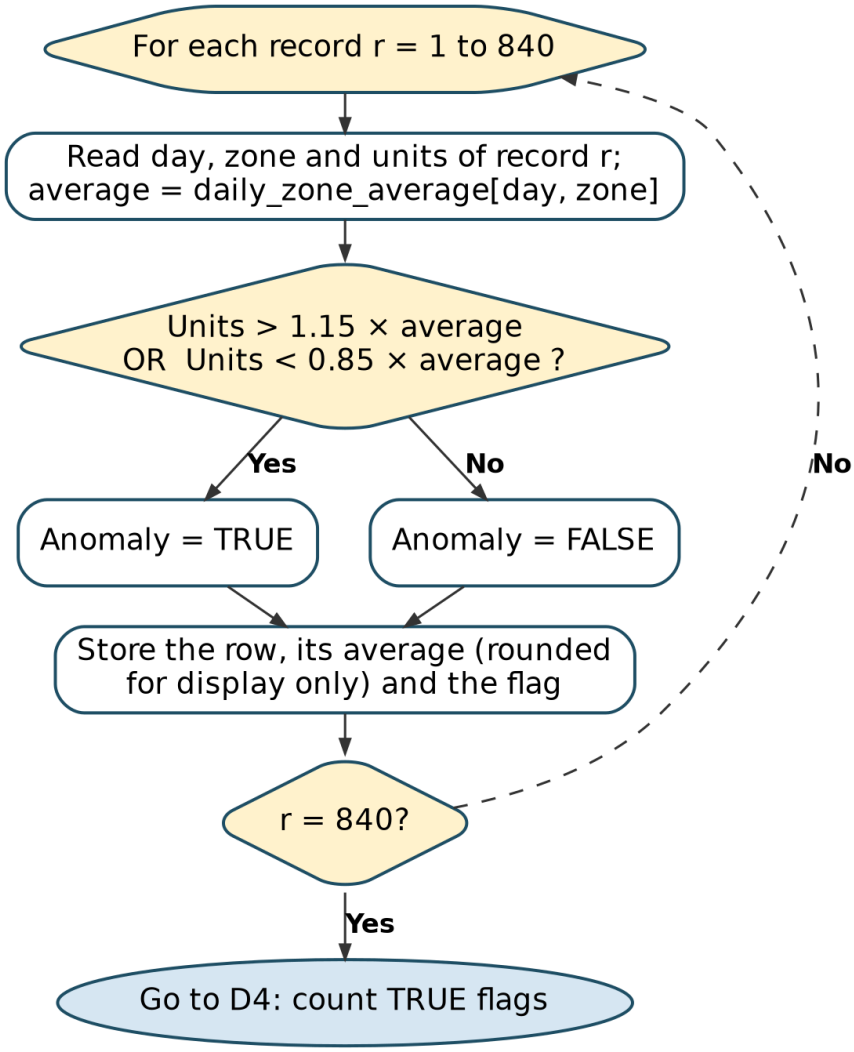

Figure 3: Part D3 anomaly rule applied to each record.

2.5 Peak productive day per zone (Part E)

Part E uses three nested loops: zone, day and record. For each zone, every day's total is compared with the best total so far using a strict greater-than test, so a tie keeps the earlier day. Figure 4 traces the search.

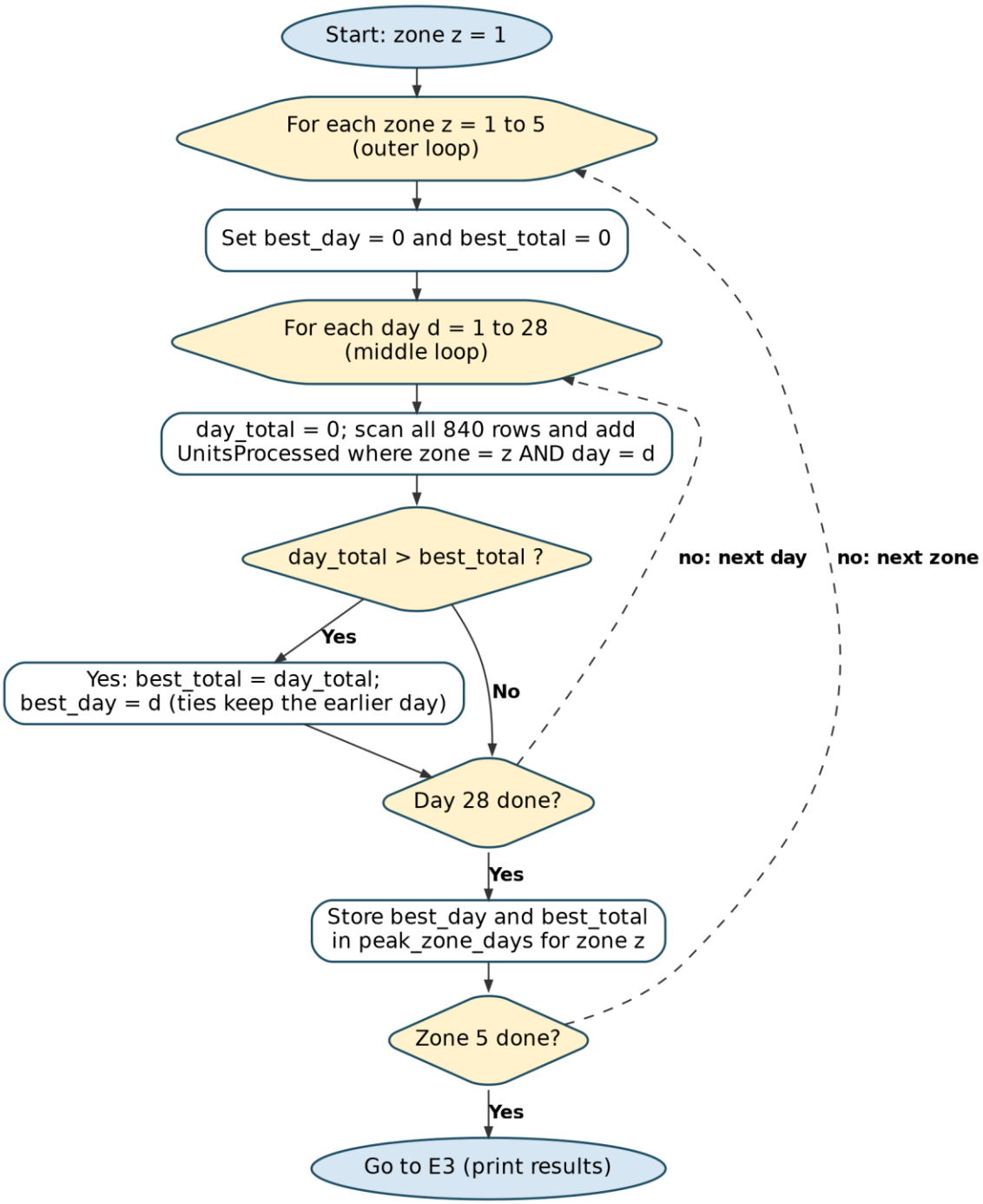

Figure 4: Part E peak-day search with its three nested loops.

2.6 Sanity check and validation suite

The sanity check compares the grand total with the sum of the zone totals. The ten tests check structure (T1 to T3), the formula and range (T4), recalculation by an independent route (T5, T6, T8, T9), plausibility against expected values (T7), and a six-value case with a known answer (T10).

2.7 Overall logic

The stages form a sequential chain, and each reads objects created by an earlier stage. The only loops return the flow to an earlier step, repeating for every row, zone, category or day. Figures 2 to 4 show the most detailed loops.

3. Section B: Justification of Programming Choices

Data structures. Data frames hold the mixed types of the results, while a matrix suits the integer-keyed day–zone lookup.

Loops. Explicit loops keep each calculation visible, at the cost of repeated scans: Parts D and E each perform about 117,600 comparisons. Test T8 shows a single-pass alternative.

Alternatives. aggregate(), tapply() or dplyr would be shorter, and I would use them in production, but the brief requires loop-based processing.

Assumptions. Random variation is uniform, the 15% threshold is symmetric, each record is compared with a group average that includes itself, and each day–zone group has six records.

4. Section C: Testing and Validation

Test design. Ten checks cover structure, formula, recalculation, plausibility and a hand-worked case. The known-outcome test is the most convincing: six values with a mean of 100 should flag exactly the 120 and the 80.

Limitations. T5, T6, T8 and T9 reuse the same generated data, so they check the calculations but not the generation. T7's three-standard-deviation tolerance can fail by chance, about 0.3% for each result. T1, T4, T6 and T9 hard-code parameters. The check 100 * 1.15 == 115 returns FALSE because of floating-point storage, so a tolerance-based comparison would be safer.

5. Section D: Efficiency and Scalability Reflection

At 840 rows efficiency is not a concern, but Parts D and E scan the full table once for each of the 140 day–zone combinations, so cost grows quickly with dataset size. A single pass into matrices, as T8 does, would remove most of this cost. Deriving the constants 840, 140 and 6 from the parameters would make future extension safer.

6. Section E: Reflection on Learning and Use of Generative AI

This work taught me that data manipulation is about more than writing formulas. Selecting the right records, resetting accumulators, and placing results correctly was easy to get wrong; even small indexing mistakes changed the final numbers. A fixed 15% threshold is simple to explain, but flagged records need to be investigated, not deleted. Checking the grand total catches some errors, so I reviewed results by group and checked boundary points. I used generative AI only to help debug my R script; the approach, calculations, interpretation and evaluation are all my own work. Going forward, I will keep a test log recording each test, its outcome, and any improvements made.

7. Conclusion

The script generates reproducible data, summarises it, flags anomalies, identifies peak days and verifies its results. Its strengths are transparency and the known-outcome test. Its weaknesses are repeated scanning, hard-coded constants and a statistical check that can fail by chance.