Assignment 1: Introductory Statistics and Data Handling

Dr Kanthu Joseph Mhango

Instructions

- Use **base R** unless a question explicitly allows another package.
- Write clear, commented code.
- Where explanations are requested, use full sentences.
- Aim for *reproducible* work: if you use randomness, set a seed.

Part A: Indexing, Subsetting, and Defensive R

1. Indexing with missing values

```r
x <- c(3.1, NA, 4.7, 2.9, NA, 5.0)
```

Tasks:

1. Extract the positions (indices) of the missing values.
2. Extract all non-missing values.
3. Replace missing values with the **mean of non-missing** values.
4. Show that `mean(x)` now returns a number.

---

2. Row/column selection with multiple conditions

```
df <- data.frame(
  id = 1:8,
  site = c("A", "A", "B", "B", "A", "C", "C", "B"),
  treatment = c("control", "drug", "control", "drug", "drug", "control", "drug", "control"),
  score = c(12, 15, 9, 18, 14, 11, 20, 10)
)
```

Tasks:

1. Subset rows where site is A **or** C.
2. Subset rows where treatment is drug **and** score is at least 15.
3. Create a new object containing only the columns `id`, `treatment`, and `score`.
4. In one sentence, explain the difference between `&` and `|`.

---

3. Catching silent mistakes

```
vals <- c("1", "2", "3", "4")
```

Tasks:

1. Write code that checks whether `vals` is numeric *before* computing the mean.
2. If it is not numeric, convert it safely.
3. Compute the mean.
4. In 2–3 sentences, explain why silent coercion can be dangerous.

Part B: Data Types, Factors, and Common Pitfalls

4. Factors: ordering and reference levels

```
grades <- c("merit", "pass", "distinction", "pass", "merit", "pass")
```

Tasks:

1. Convert `grades` to a factor with ordered levels: `pass < merit < distinction`.
2. Show the levels and confirm it is ordered.
3. Compute a frequency table.
4. In 2–3 sentences, explain why ordered factors matter for modelling/plots.

---

5. Date parsing and type checking

```
dates_chr <- c("2026-01-05", "2026-01-12", "2026-02-01", "not_a_date")
```

Tasks:

1. Convert this to a `Date` vector.
2. Identify which element(s) failed to parse.
3. Remove invalid dates and compute the number of days between the earliest and latest valid date.
4. In 2–3 sentences, explain why date parsing often creates NAs.

---

6. Coercion with mixed types in a data frame

```
messy <- data.frame(
  sample_id = c("S1", "S2", "S3", "S4"),
  nitrate = c("12.1", "11.8", "missing", "13.0"),
  ph = c(6.2, 6.5, 6.1, 6.4),
  stringsAsFactors = FALSE
)
```

Tasks:

1. Convert `nitrate` to numeric.
2. Report how many NAs were created by conversion.
3. Create a new column `nitrate_flag` that is TRUE if nitrate is missing and FALSE otherwise.
4. Compute the mean nitrate ignoring NAs.

Part C: Cleaning Names, Reshaping, and Summaries

7. Cleaning messy column names using regex (base R only)

```
nm <- c(" Plot ID ", "Yield(kg/ha)", "Soil%Moisture", "2nd reading", "sensor.ID")
```

Tasks:

Using only base R string tools (`trimws`, `tolower`, `gsub`, etc.):

1. Trim spaces.
2. Convert to lowercase.
3. Replace punctuation and spaces with `_`.
4. Collapse multiple `_` to a single `_`.
5. Ensure names do not start with a digit (prefix with `x_` if needed).
6. Return `nm_clean`.

Explain your regex choices in 3–4 sentences.

---

8. Wide → long (base R only), then grouped summaries

```
set.seed(21)
wide <- data.frame(
  plot = paste0("P", 1:6),
  y_2024 = rnorm(6, 5.0, 0.6),
  y_2025 = rnorm(6, 5.4, 0.6),
  y_2026 = rnorm(6, 5.2, 0.6)
)
```

Tasks:

1. Reshape `wide` into long format using base R (no `pivot_longer`).
2. Long format should have columns: `plot`, `year`, `yield`.
3. Compute the mean and sd of yield per year (base R).
4. In 2–3 sentences, explain why long format is often more convenient.

---

9. Long → wide with *two* value columns (tidyverse allowed)

```
library(tidyr)

long2 <- data.frame(
  id = rep(1:3, each = 2),
  time = rep(c("pre", "post"), times = 3),
  height = c(10, 12, 9, 11, 13, 14),
  weight = c(20, 22, 19, 21, 23, 24)
)
```

Tasks:

1. Pivot `long2` to wide so that you get columns like `height_pre`, `height_post`, `weight_pre`, `weight_post`.
2. Check the result using `str()`.
3. Compute the change scores: `height_post - height_pre` and `weight_post - weight_pre`.

Part D: Joins, Keys, and Debugging

10. Fixing a join with inconsistent keys

You are given two tables. They should match on plot ID, but the formatting differs.

```
left_tbl <- data.frame(
  plot = c("P-01", "P-02", "P-03", "P-04"),
  yield = c(5.1, 5.4, 5.0, 5.6),
  stringsAsFactors = FALSE
)

right_tbl <- data.frame(
  plot = c("P01", "P02", "P03", "P05"),
  soil = c("clay", "silt", "sand", "peat"),
  stringsAsFactors = FALSE
)
```

Tasks:

1. Standardise the plot IDs so the join works.
2. Merge the tables and keep only matching rows.
3. Identify which plots were missing from each side (i.e. present in one table but not the other).
4. In 3–4 sentences, explain what can go wrong when join keys are inconsistent.

---

11. Duplicate keys and unexpected row counts

```r
a <- data.frame(id = c(1, 2, 2, 3), score = c(10, 12, 13, 9))
b <- data.frame(id = c(1, 2, 3, 3), group = c("X", "Y", "Y", "Z"))
```

Tasks:

1. Merge `a` and `b` by `id`.
2. Report the number of rows in the merged output.
3. Explain (in 3–4 sentences) why the row count increased.
4. Create a version of `a` and `b` with unique IDs (choose a sensible rule) and merge again.

---

12. Plot coloured by a derived grouping

Using `mtcars`:

Tasks:

1. Create a data frame `labels` with car names and a new variable `eff_band` with two levels:
   - `"efficient"` if `mpg >= 20`
   - `"inefficient"` otherwise
2. Merge `mtcars` with `labels` by car name (row names).
3. Create a scatter plot of `wt` vs `mpg` coloured by `eff_band`.
4. Add a legend with exactly two entries.

Part E: Functions, Lists, and Structures

13. A robust summary function with input checks

Write a function `summarise_cols()` that:

- Takes a data frame and a character vector of column names
- Errors with a clear message if any column name is not present
- Errors if any requested column is not numeric
- Returns a data frame with rows = variables and columns = `n`, `mean`, `sd`, and `na_count`

Demonstrate it on `mtcars` for: `c("mpg", "hp", "wt")`.

Explain briefly why using `[[` is safer than `$` in a function.

---

14. Applying a function over a list of data frames

```
d1 <- iris[1:50, c("Sepal.Length", "Sepal.Width")]
d2 <- iris[51:100, c("Sepal.Length", "Sepal.Width")]
d3 <- iris[101:150, c("Sepal.Length", "Sepal.Width")]
L <- list(d1, d2, d3)
```

Tasks:

1. Use `lapply()` to compute the correlation between `Sepal.Length` and `Sepal.Width` in each element.
2. Return the results as a numeric vector.
3. Combine the three data frames into one using `do.call(rbind, ...)`.
4. Compute the correlation again on the combined data.

Explain (2–3 sentences) what `do.call(rbind, ...)` is doing.

---

15. Matrix vs data frame behaviour

```r
M <- matrix(1:12, nrow = 3, byrow = TRUE)
colnames(M) <- c("A", "B", "C", "D")
```

Tasks:

1. Convert `M` to a data frame `DF`.
2. Add a non-numeric column `band` with values: `c("low", "high", "low")` as a factor.
3. Show (with code) that `colSums(M)` works, but `colSums(DF)` behaves differently.
4. In 3–4 sentences, explain why matrices must be homogeneous and data frames can be heterogeneous.

Part F: Intro Statistics (Applied)

16. Sampling distribution by simulation

```r
set.seed(202)
pop <- rexp(5000, rate = 1/10)  # skewed population
```

Tasks:

1. Take 200 samples of size 10, compute the mean each time, store them.
2. Repeat for sample size 40.
3. Plot two histograms (one for n=10 means, one for n=40 means).
4. In 3–5 sentences, explain what changes and why.

---

17. Confidence interval from scratch (no `t.test()`)

```r
set.seed(77)
x <- rnorm(18, mean = 5.5, sd = 1.2)
```

Tasks:

1. Compute a 95% CI for the mean *by hand* using:

- the sample mean
- the sample sd
- the t critical value via `qt()`

2. Then verify by running `t.test(x)` and comparing intervals.
3. In 2–3 sentences, explain why the t critical value depends on `n`.

---

18. Two-sample comparison with checking assumptions (lightweight)

```r
set.seed(88)
A <- rnorm(20, mean = 50, sd = 6)
B <- rnorm(22, mean = 54, sd = 9)
```

Tasks:

1. Make side-by-side boxplots of A and B.
2. Compute the mean and sd for each group.
3. Run a two-sample t-test.
4. In 3–4 sentences, comment on whether equal variance seems plausible and what that means.

---

19. Relevel a factor and interpret coefficients

Using `iris`:

Tasks:

1. Relevel `Species` so that `"virginica"` is the reference.
2. Fit: `Sepal.Length ~ Species`.
3. Show the coefficient table.
4. In 3–4 sentences, explain how the reference level affects interpretation.

---

20. A small end-to-end workflow

Create a small dataset and analyse it.

```
set.seed(909)
trial <- data.frame(
  plot = rep(paste0("P", 1:12), each = 2),
  treatment = rep(c("control", "treated"), times = 12),
  yield = rnorm(24, mean = 5.0, sd = 0.5)
)
trial$yield[trial$treatment == "treated"] <- trial$yield[trial$treatment == "treated"] + 0.3
```

Tasks:

1. Convert `treatment` to a factor and set `control` as the reference.
2. Compute mean yield by treatment.
3. Make a boxplot of yield by treatment.
4. Fit a linear model `yield ~ treatment` and interpret the treatment coefficient in one sentence.
5. In 2–3 sentences, explain why this model is closely related to a two-sample t-test.