A function another officer can run

Turning a script into functions with arguments and stopifnot(), the tidy evaluation you need to pass a column name, and the command-line script that runs on next quarter's export unedited.

Pierre Robentz Cassion

Lesson 8 of 8

R and the Tidyverse for Programme Data — Reshaping and repeating

What this lesson covers

- The handover test
- From script to function
- `stopifnot()` for what cannot happen
- Passing a column name: tidy evaluation
- The indicator function
- Testing the part that computes
- The command-line script
- Recording what produced the output
- Where this course ends

The handover test

- An analysis is finished when someone else can run it on next quarter's export without editing the code.

From script to function — In R (cont.)

```r
# R/clean_register.R

clean_register <- function(muac, gam_mm = 125, sam_mm = 115) {
  stopifnot(is.data.frame(muac))
  required <- c("child_id", "commune", "muac_mm", "oedema")
  missing <- setdiff(required, names(muac))
  if (length(missing) > 0) {
    stop("Input is missing columns: ", paste(missing, collapse = ", "))
  }

  muac |>
    dplyr::mutate(
      # A MUAC below 40 is centimetres in a millimetre field. Applied only
      # below a threshold no plausible measurement reaches.
      muac_mm = dplyr::if_else(muac_mm < 40, muac_mm * 10L, muac_mm),
      muac_mm = dplyr::if_else(dplyr::between(muac_mm, 80L, 220L), muac_mm, NA_integer_),
```

From script to function — In R (cont.)

```
      oedema  = dplyr::recode(
        tolower(trimws(oedema)),
        "true" = TRUE, "y" = TRUE, "yes" = TRUE,
        "false" = FALSE, "n" = FALSE, "no" = FALSE,
        .default = NA
      ),
      assessed = !is.na(muac_mm) | !is.na(oedema),
      gam = assessed & (muac_mm < gam_mm | oedema),
      sam = assessed & (muac_mm < sam_mm | oedema)
    )
}
```

From script to function

- It takes a data frame and returns one — No file reading, no writing, no `<-`
- The thresholds are arguments with defaults — A threshold that is only ever changed by editing a line will eventually be...
- It validates its input — `stop()` with a message naming the missing columns beats a `NULL` propagating into a mutate...
- Nothing is grouped on the way out — A grouped frame escaping a function is the bug from the grouping lesson, arriving...

`stopifnot()` for what cannot happen — In R

```r
stopifnot(
  "every row needs an identifier" = !any(is.na(muac$child_id)),
  "MUAC must be plausible after cleaning" =
    all(is.na(muac$muac_mm) | dplyr::between(muac$muac_mm, 80, 220))
)
```

Passing a column name: tidy evaluation — In R

```r
# Does not work.
rate_by <- function(df, group_col) {
  df |> dplyr::summarise(n = dplyr::n(), .by = group_col)
}

rate_by(muac, commune)
#> Error: object 'commune' not found
```

Passing a column name: tidy evaluation — In R

```r
rate_by <- function(df, group_col) {
  df |> dplyr::summarise(n = dplyr::n(), .by = {{ group_col }})
}

rate_by(muac, commune)
```

Passing a column name: tidy evaluation — In R

```r
rate_by_name <- function(df, group_col) {
  df |> dplyr::summarise(n = dplyr::n(), .by = dplyr::all_of(group_col))
}

rate_by_name(muac, "commune")
```

Passing a column name: tidy evaluation — In R

```r
count_as <- function(df, group_col, out_name) {
  df |> dplyr::summarise("{out_name}" := dplyr::n(), .by = {{ group_col }})
}
```

The indicator function — In R

```r
indicator_table <- function(muac, gam_mm = 125, sam_mm = 115) {
  clean_register(muac, gam_mm, sam_mm) |>
    dplyr::summarise(
      screened  = dplyr::n(),
      assessed  = sum(assessed, na.rm = TRUE),
      gam_cases = sum(gam, na.rm = TRUE),
      sam_cases = sum(sam, na.rm = TRUE),
      .by = commune
    ) |>
    dplyr::mutate(
      assessment_rate = round(assessed / screened, 3),
      gam_rate = round(gam_cases / assessed, 3),
      sam_rate = round(sam_cases / assessed, 3)
    ) |>
    dplyr::arrange(dplyr::desc(gam_rate))
}
```

Testing the part that computes — In R

```r
# tests/testthat/test-indicator-table.R
test_that("the denominator excludes unmeasured children", {
  muac <- tibble::tibble(
    child_id = c("a", "b", "c"),
    commune  = "X",
    muac_mm  = c(110L, 130L, NA_integer_),
    oedema   = c("false", "false", NA_character_)
  )

  out <- indicator_table(muac)

  expect_equal(out$screened, 3)
  expect_equal(out$assessed, 2)     # the unmeasured child is not a denominator
  expect_equal(out$gam_rate, 0.5)
})
```

The command-line script — In R (cont.)

```r
#!/usr/bin/env Rscript
# scripts/indicator_table.R

suppressPackageStartupMessages({
  library(optparse)
  library(readr)
})

source(here::here("R", "clean_register.R"))
source(here::here("R", "indicator_table.R"))

option_list <- list(
  make_option("--input", type = "character", help = "Screening register CSV."),
  make_option("--output", type = "character", help = "Directory for the tables."),
  make_option("--gam-threshold", type = "integer", default = 125L,
              help = "MUAC cut-off in mm for GAM [default %default].")
```

The command-line script — In R (cont.)

```
)

opt <- parse_args(OptionParser(option_list = option_list))
if (is.null(opt$input) || is.null(opt$output)) {
  stop("--input and --output are required.")
}

muac <- read_register(opt$input)
message("Read ", nrow(muac), " rows from ", opt$input)

table <- indicator_table(muac, gam_mm = opt$`gam-threshold`)

thin <- table$commune[table$assessment_rate < 0.8]
if (length(thin) > 0) {
  warning("Assessment rate below floor: ", paste(thin, collapse = ", "))
}
```

The command-line script — In R (cont.)

```
dir.create(opt$output, recursive = TRUE, showWarnings = FALSE)
destination <- file.path(opt$output, "gam_by_commune.csv")
write_csv(table, destination)
message("Wrote ", destination, " (", nrow(table), " communes)")
```

The command-line script — Shell

```
Rscript scripts/indicator_table.R \
  --input data/raw/muac-screening-artibonite-2024.v1.csv \
  --output outputs/tables
```

Recording what produced the output — In R

```
run_metadata <- function(opt) {
  list(
    generated_at  = format(Sys.time(), "%Y-%m-%dT%H:%M:%S%z"),
    input         = opt$input,
    gam_threshold = opt$`gam-threshold`,
    r_version     = as.character(getRversion()),
    dplyr         = as.character(packageVersion("dplyr"))
  )
}

jsonlite::write_json(run_metadata(opt), file.path(opt$output, "run.json"),
                     auto_unbox = TRUE, pretty = TRUE)
```

Recording what produced the output — Shell

```
rm -rf outputs/
Rscript scripts/indicator_table.R --input data/raw/muac.csv --output outputs/tables
git status
```

Where this course ends

- You can build a project that reopens a year later with its package versions recorded, read any export without corrupting it, keep the labels a survey file carries, put categories in the order a report needs, aggregate to a numerator and a denominator that came from one call, reshape a flattened repeat group, and hand the result to someone else as a script rather than as a favour.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/r-for-programme-data/r-08-functions