From tidy data to an indicator table

Turn a clean dataset into the disaggregated indicator table a logframe asks for, with numerator, denominator and confidence interval stated explicitly.

Pierre Robentz Cassion

Lesson 7 of 8

Data Analysis Foundations for M&E

What this lesson covers

- An indicator is a numerator over a denominator, disaggregated
- The indicator reference sheet
- Build the numerator as a column
- The indicator table
- Disaggregation
- Say how uncertain you are
- Format it for the report
- What comes next

An indicator is a numerator over a denominator, disaggregated

- That sentence is the whole lesson.

The indicator reference sheet

Field	For our example
Indicator	Global acute malnutrition (GAM) prevalence by MUAC
Numerator	Children 6-59 months screened with MUAC below 125 mm, or with bilateral pitting oedema
Denominator	Children 6-59 months with a valid MUAC measurement or a recorded oedema assessment
Disaggregation	Commune, sex, age band (6-23, 24-59 months)
Frequency	Quarterly, and annually for the donor report
Source	Community mass screening register, CommCare
Decision it informs	Which communes receive an additional CMAM site next quarter
Reference value	GAM at or above 15% is the emergency threshold

The indicator reference sheet

If you cannot write the denominator in one sentence, you do not yet have an indicator. You have a column you are about to average.

Build the numerator as a column — In Python (cont.)

```
import numpy as np

SAM_MM = 115
GAM_MM = 125

muac["has_assessment"] = muac["muac_mm"].notna() | muac["oedema"].notna()

muac["sam"] = np.where(
    ~muac["has_assessment"],
    np.nan,
    ((muac["muac_mm"] < SAM_MM) | (muac["oedema"] == True)).astype(float),
)

muac["gam"] = np.where(
    ~muac["has_assessment"],
    np.nan,
```

Build the numerator as a column — In Python (cont.)

```
    ((muac["muac_mm"] < GAM_MM) | (muac["oedema"] == True)).astype(float),
)
```

Build the numerator as a column — In R

```r
SAM_MM <- 115
GAM_MM <- 125

muac <- muac |>
  mutate(
    has_assessment = !is.na(muac_mm) | !is.na(oedema),
    sam = if_else(has_assessment, (muac_mm < SAM_MM) | oedema %in% TRUE, NA),
    gam = if_else(has_assessment, (muac_mm < GAM_MM) | oedema %in% TRUE, NA)
  )
```

Build the numerator as a column

- **Oedema is SAM regardless of measurement.** A child at 130 mm with bilateral pitting oedema is severely acutely…
- `oedema == True` **rather than a truthiness test.** Missing oedema is not false. In R, `oedema %in% TRUE` treats `NA`…

The indicator table — In Python (cont.)

```
def indicator_table(df, by):
    grouped = df.groupby(by, dropna=False)
    table = grouped.agg(
        screened=("child_id", "size"),
        denominator=("has_assessment", "sum"),
        sam_cases=("sam", "sum"),
        gam_cases=("gam", "sum"),
    )
    table["gam_rate"] = table["gam_cases"] / table["denominator"]
    table["sam_rate"] = table["sam_cases"] / table["denominator"]
    return table.reset_index()


by_commune = indicator_table(muac, "commune").sort_values(
    "gam_rate", ascending=False
)
```

The indicator table — In Python (cont.)

```
print(by_commune.round(4))
```

The indicator table — In R (cont.)

```r
indicator_table <- function(df, by) {
  df |>
    group_by(across(all_of(by))) |>
    summarise(
      screened    = n(),
      denominator = sum(has_assessment),
      sam_cases   = sum(sam, na.rm = TRUE),
      gam_cases   = sum(gam, na.rm = TRUE),
      .groups = "drop"
    ) |>
    mutate(
      gam_rate = gam_cases / denominator,
      sam_rate = sam_cases / denominator
    )
}
```

The indicator table — In R (cont.)

```
by_commune <- indicator_table(muac, "commune") |> arrange(desc(gam_rate))
by_commune
```

Disaggregation — In Python

```
muac["age_band"] = pd.cut(
    muac["age_months"],
    bins=[6, 24, 60],
    labels=["6-23 months", "24-59 months"],
    right=False,
)

by_sex = indicator_table(muac, ["commune", "sex"])
by_age = indicator_table(muac, "age_band")

print(by_age.round(4))
```

Disaggregation — In R

```r
muac <- muac |>
  mutate(
    age_band = cut(
      age_months,
      breaks = c(6, 24, 60),
      labels = c("6-23 months", "24-59 months"),
      right = FALSE
    )
  )

by_sex <- indicator_table(muac, c("commune", "sex"))
by_age <- indicator_table(muac, "age_band")

by_age
```

Say how uncertain you are — In Python (cont.)

```
from scipy.stats import beta

def wilson_interval(successes, n, confidence=0.95):
    if n == 0:
        return (np.nan, np.nan)
    lower = beta.ppf((1 - confidence) / 2, successes, n - successes + 1) if successes > 0
     else 0.0
    upper = beta.ppf(1 - (1 - confidence) / 2, successes + 1, n - successes) if successes < n
      else 1.0
    return (lower, upper)


bounds = by_commune.apply(
    lambda r: wilson_interval(r["gam_cases"], r["denominator"]), axis=1
)
by_commune["gam_low"] = [b[0] for b in bounds]
by_commune["gam_high"] = [b[1] for b in bounds]
```

Say how uncertain you are — In Python (cont.)

```
print(by_commune[["commune", "denominator", "gam_rate", "gam_low", "gam_high"]].round(4))
```

Say how uncertain you are — In R

```r
ci <- function(cases, n) {
  if (n == 0) return(c(NA_real_, NA_real_))
  test <- binom.test(cases, n)
  test$conf.int
}

by_commune <- by_commune |>
  rowwise() |>
  mutate(
    gam_low  = ci(gam_cases, denominator)[1],
    gam_high = ci(gam_cases, denominator)[2]
  ) |>
  ungroup()

by_commune |> select(commune, denominator, gam_rate, gam_low, gam_high)
```

Format it for the report — In Python

```
report = by_commune.assign(
    gam=lambda d: (d["gam_rate"] * 100).round(1).astype(str) + "%",
    ci=lambda d: "("
    + (d["gam_low"] * 100).round(1).astype(str)
    + " – "
    + (d["gam_high"] * 100).round(1).astype(str)
    + ")",
)[["commune", "screened", "denominator", "gam_cases", "gam", "ci"]]

report.columns = [
    "Commune", "Screened", "Assessed", "GAM cases", "GAM rate", "95% CI",
]
report.to_csv("output/tables/gam-by-commune.csv", index=False)
print(report.to_string(index=False))
```

Format it for the report — In R

```r
report <- by_commune |>
  transmute(
    Commune     = commune,
    Screened    = screened,
    Assessed    = denominator,
    `GAM cases` = gam_cases,
    `GAM rate`  = sprintf("%.1f%%", gam_rate * 100),
    `95% CI`    = sprintf("(%.1f - %.1f)", gam_low * 100, gam_high * 100)
  )

readr::write_csv(report, "output/tables/gam-by-commune.csv")
report
```

What comes next

- The last lesson makes this run again — on next quarter's export, on someone else's laptop, without editing anything.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/data-analysis-foundations/foundations-07-indicator-tables