Validation that runs without you

Turn the checks into a suite that executes on every read — a schema contract, three severities, a report artefact, and the decision about which failures are allowed to stop the pipeline.

Pierre Robentz Cassion

Lesson 7 of 8

Data Cleaning and Validation — The record that survives you

What this lesson covers

- The checks you ran are not the checks you have
- A contract, not a script
- Three severities, and only one of them stops the run
- The validator
- Wire it into the read, not into a notebook cell
- The findings are an artefact, like the profile
- What "fail loudly" costs, and why it is still right
- What comes next

The checks you ran are not the checks you have

- Six lessons of checks, and every one of them was run by a person who decided to run it.

A contract, not a script — In Python (cont.)

```
CONTRACT = {
    "name": "muac-screening",
    "key": ["child_id"],
    "columns": {
        "child_id":       {"type": "string",  "required": True},
        "commune":        {"type": "string",  "required": True, "allowed": COMMUNES},
        "screening_date": {"type": "date",    "required": True,
                           "min": "2024-01-01", "max": "2024-12-31"},
        "age_months":     {"type": "integer", "required": False, "min": 6, "max": 59},
        "sex":            {"type": "string",  "required": True, "allowed": ["f", "m"]},
        "muac_mm":        {"type": "integer", "required": False, "min": 80, "max": 220,
                           "sentinels": [-99]},
        "oedema":         {"type": "boolean", "required": False},
        "outcome":        {"type": "string",  "required": True, "allowed": OUTCOMES},
    },
    "expected_rows": (3500, 5000),
```

A contract, not a script — In Python (cont.)

```
    "max_missing": {"age_months": 0.10, "muac_mm": 0.05},
}
```

A contract, not a script — In R (cont.)

```r
CONTRACT <- list(
  name = "muac-screening",
  key  = "child_id",
  columns = list(
    child_id       = list(type = "character", required = TRUE),
    commune        = list(type = "character", required = TRUE, allowed = COMMUNES),
    screening_date = list(type = "Date", required = TRUE,
                          min = as.Date("2024-01-01"), max = as.Date("2024-12-31")),
    age_months     = list(type = "integer", required = FALSE, min = 6, max = 59),
    sex            = list(type = "character", required = TRUE, allowed = c("f", "m")),
    muac_mm        = list(type = "integer", required = FALSE, min = 80, max = 220,
                          sentinels = -99),
    oedema         = list(type = "logical", required = FALSE),
    outcome        = list(type = "character", required = TRUE, allowed = OUTCOMES)
  ),
  expected_rows = c(3500, 5000),
```

A contract, not a script — In R (cont.)

```
  max_missing = list(age_months = 0.10, muac_mm = 0.05)
)
```

A contract, not a script

- `expected_rows` — is a range, not a number
- `max_missing` — turns lesson 2's finding into a limit

Three severities, and only one of them stops the run

Severity	Means	Effect
`error`	The file cannot be analysed as it is	Stop. Nothing downstream runs.
`warning`	Something is wrong with some rows	Flag them, continue, report the count
`note`	Worth knowing, expected to occur	Report only

Three severities, and only one of them stops the run

- Errors are about the file. Warnings are about rows — If you find yourself wanting to make a row-level check an error,. . .

The validator — In Python (cont.)

```python
from dataclasses import dataclass


@dataclass
class Finding:
    check: str
    severity: str
    count: int
    detail: str


def validate(df, contract):
    findings = []

    missing = set(contract["columns"]) - set(df.columns)
    if missing:
```

The validator — In Python (cont.)

```
    findings.append(Finding("columns-present", "error", len(missing),
                            f"missing columns: {sorted(missing)}"))
    return findings                         # nothing else is meaningful

low, high = contract["expected_rows"]
if not low <= len(df) <= high:
    findings.append(Finding("row-count", "error", len(df),
                            f"{len(df)} rows, expected {low}-{high}"))

duplicated = df.duplicated(subset=contract["key"], keep=False)
if duplicated.any():
    findings.append(Finding("key-unique", "error", int(duplicated.sum()),
                            f"rows sharing a {contract['key']}"))

for column, spec in contract["columns"].items():
    values = df[column]
```

The validator — In Python (cont.)

```
    if spec.get("required") and values.isna().any():
        findings.append(Finding(f"{column}-required", "error",
                                int(values.isna().sum()), "required column has blanks"))
    if "allowed" in spec:
        unexpected = set(values.dropna().unique()) - set(spec["allowed"])
        if unexpected:
            findings.append(Finding(f"{column}-allowed", "error", len(unexpected),
                                    f"unexpected values: {sorted(unexpected)}"))
    if "min" in spec:
        out = (values < spec["min"]) | (values > spec["max"])
        if out.any():
            findings.append(Finding(f"{column}-range", "warning", int(out.sum()),
                                    f"outside {spec['min']}-{spec['max']}"))

for column, limit in contract["max_missing"].items():
    share = df[column].isna().mean()
```

The validator — In Python (cont.)

```
    if share > limit:
        findings.append(Finding(f"{column}-missing", "error", int(df[column].isna().sum()
),
                                f"{share:.1%} missing, limit {limit:.0%}"))

return findings
```

The validator — In R (cont.)

```r
validate <- function(df, contract) {
  findings <- list()
  add <- function(check, severity, count, detail) {
    findings[[length(findings) + 1]] <<-
      tibble::tibble(check = check, severity = severity, count = count, detail = detail)
  }

  missing <- setdiff(names(contract$columns), names(df))
  if (length(missing)) {
    add("columns-present", "error", length(missing),
        paste("missing columns:", paste(missing, collapse = ", ")))
    return(dplyr::bind_rows(findings))
  }

  if (!dplyr::between(nrow(df), contract$expected_rows[1], contract$expected_rows[2])) {
    add("row-count", "error", nrow(df), sprintf("%d rows, expected %d-%d", nrow(df),
```

The validator — In R (cont.)

```
      contract$expected_rows[1], contract$expected_rows[2]))
}

dup <- duplicated(df[contract$key]) | duplicated(df[contract$key], fromLast = TRUE)
if (any(dup)) add("key-unique", "error", sum(dup), "rows sharing a key")

for (column in names(contract$columns)) {
  spec <- contract$columns[[column]]
  values <- df[[column]]
  if (isTRUE(spec$required) && any(is.na(values))) {
    add(paste0(column, "-required"), "error", sum(is.na(values)), "required column has
   blanks")
  }
  if (!is.null(spec$allowed)) {
    unexpected <- setdiff(unique(stats::na.omit(values)), spec$allowed)
    if (length(unexpected)) {
      add(paste0(column, "-allowed"), "error", length(unexpected),
```

The validator — In R (cont.)

```
            paste("unexpected values:", paste(unexpected, collapse = ", "))
        }
      }
      if (!is.null(spec$min)) {
        out <- values < spec$min | values > spec$max
        if (any(out, na.rm = TRUE)) {
          add(paste0(column, "-range"), "warning", sum(out, na.rm = TRUE),
              sprintf("outside %s-%s", spec$min, spec$max))
        }
      }
    }

    dplyr::bind_rows(findings)
}
```

Wire it into the read, not into a notebook cell — In Python

```python
def load_screening(path, contract=CONTRACT):
    df = read_typed(path, contract)
    findings = validate(df, contract)

    errors = [f for f in findings if f.severity == "error"]
    for finding in findings:
        print(f"[{finding.severity}] {finding.check}: {finding.count} - {finding.detail}")

    if errors:
        raise ValueError(f"{len(errors)} validation errors in {path}")
    return df, findings
```

Wire it into the read, not into a notebook cell — In R

```r
load_screening <- function(path, contract = CONTRACT) {
  df <- read_typed(path, contract)
  findings <- validate(df, contract)

  if (nrow(findings)) print(findings, n = Inf)
  if (any(findings$severity == "error")) {
    stop(sprintf("%d validation errors in %s", sum(findings$severity == "error"), path))
  }
  list(data = df, findings = findings)
}
```

Wire it into the read, not into a notebook cell

- There is now no way to read this file without validating it — That is the whole mechanism

The findings are an artefact, like the profile — In Python

```
import json
from pathlib import Path

Path("outputs/validation").mkdir(parents=True, exist_ok=True)
Path("outputs/validation/muac-2024-q4.json").write_text(
    json.dumps([f.__dict__ for f in findings], indent=2)
)
```

The findings are an artefact, like the profile — In R

```
jsonlite::write_json(findings,
  here::here("outputs", "validation", "muac-2024-q4.json"),
  pretty = TRUE
)
```

The findings are an artefact, like the profile — In Python

```
history = pd.concat([
    pd.read_json(p).assign(quarter=p.stem)
    for p in sorted(Path("outputs/validation").glob("*.json"))
])
print(history.pivot_table(index="check", columns="quarter", values="count", fill_value=0))
```

The findings are an artefact, like the profile — In R

```
history <- purrr::map_dfr(
  list.files(here::here("outputs", "validation"), full.names = TRUE),
  ~ jsonlite::read_json(.x, simplifyVector = TRUE) |>
      dplyr::mutate(quarter = tools::file_path_sans_ext(basename(.x)))
)

tidyr::pivot_wider(history, id_cols = check, names_from = quarter, values_from = count)
```

What "fail loudly" costs, and why it is still right — Example

```
[error] commune-allowed: 1 - unexpected values: ['Petite-Riviere']
[error] age_months-missing: 226 - 5.4% missing, limit 5%
ValueError: 2 validation errors in muac-screening-2024-q4.csv
```

What comes next

- The suite now finds everything and changes nothing — by design, because every change so far has been a flag.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/data-cleaning-and-validation/cleaning-07-validation-that-runs