A pipeline that fails is better than one that guesses

The upstream export gains a column, loses a column, changes a code from "yes" to "Y", or arrives with half the rows. A pipeline that carries on produces a plausible wrong number. Seven checks stop it, and each one has a real failure behind it.

Pierre Robentz Cassion

What this lesson covers

- The failure that has no error message
- Seven checks, each with a real failure behind it
- Fail at the point of failure, not at the end
- What this platform does
- The rule worth taking
- Report it whole
- What comes next

The failure that has no error message

- Nothing errored — The chart is drawn, the percentage is plausible, the file is dated today, and the number is wrong
- That is the failure mode this lesson exists for — and it is much more common than a crash

Seven checks, each with a real failure behind it

- One: the columns you expect are present

Seven checks, each with a real failure behind it — In Python

```
REQUIRED = {"household_id", "district", "water_source", "round_trip_minutes"}
missing = REQUIRED - set(survey.columns)
if missing:
    raise ValueError(f"Export is missing columns: {sorted(missing)}")
```

Seven checks, each with a real failure behind it — In R

```
stopifnot(all(required %in% names(survey)))
```

Seven checks, each with a real failure behind it

- Two: the row count is in the range you expect

Seven checks, each with a real failure behind it — In Python

```python
if not 2_000 <= len(survey) <= 3_000:
    raise ValueError(f"Expected 2,000-3,000 households, got {len(survey):,}")
```

Seven checks, each with a real failure behind it

- Half an export is the commonest silent failure — A truncated download, a filter left on, a date range off by a month —. . .
- Three: the codes are the codes you know

Seven checks, each with a real failure behind it — In Python

```
KNOWN = {"piped-into-dwelling", "piped-into-yard", "public-tap", "borehole",
         "protected-well", "protected-spring", "unprotected-well",
         "unprotected-spring", "surface-water", "tanker-truck", "rainwater"}
unknown = set(survey["water_source"].dropna()) - KNOWN
if unknown:
    raise ValueError(f"Unknown water_source values: {sorted(unknown)}")
```

Seven checks, each with a real failure behind it — In R

```
setdiff(unique(survey$water_source), known)
```

Seven checks, each with a real failure behind it

- A new code is a decision, not a data point — Somebody added an option to the form and the analysis has to decide where. . .
- Four: the identifier is unique where it should be

Seven checks, each with a real failure behind it — In Python

```
duplicates = survey["household_id"].duplicated().sum()
if duplicates:
    raise ValueError(f"{duplicates} duplicate household_id values")
```

Seven checks, each with a real failure behind it

- This platform's school roster has exactly this defect — two students appear twice after a transfer that was never...
- Five: the missingness is where you expect it

Seven checks, each with a real failure behind it — In Python

```
completeness = survey.notna().mean()
if completeness["district"] < 0.99:
    raise ValueError(f"district is {completeness['district']:.1%} complete")
```

Seven checks, each with a real failure behind it

- Six: the numbers are in a possible range

Seven checks, each with a real failure behind it — In Python

```
implausible = survey[(survey["litres_per_person_day"] < 0)
                     | (survey["litres_per_person_day"] > 200)]
if len(implausible) > 20:
    raise ValueError(f"{len(implausible)} implausible litres values")
```

Seven checks, each with a real failure behind it

- Note the threshold rather than zero tolerance — Eleven households with a unit error is a documented defect this...
- Seven: the output is what you declared

Seven checks, each with a real failure behind it — In Python

```
assert summary["n"].sum() == len(survey), "rows lost between input and summary"
```

Seven checks, each with a real failure behind it

- A row count that changes across a join is the single most useful assertion in programme analysis — because an inner…

Fail at the point of failure, not at the end — In Python

```
def load_survey(path: pathlib.Path) -> pd.DataFrame:
    survey = pd.read_csv(path)
    check_columns(survey)
    check_rows(survey)
    check_codes(survey)
    return survey                # nothing downstream runs on a bad file
```

Fail at the point of failure, not at the end — In R

```r
load_survey <- function(path) {
  survey <- readr::read_csv(path)
  check_columns(survey); check_rows(survey); check_codes(survey)
  survey
}
```

Fail at the point of failure, not at the end

- Check at the boundary — where data enters, and after every join — A check at the end of the pipeline tells you...
- Raise, do not warn — A warning in a log nobody reads is the same as no check, and the log is not read precisely on the...

What this platform does

Check	Catches
Zod schema	A field missing or of the wrong type
Cross-collection references	A path pointing at a course that does not exist
Files outside a locale directory	An entry with no language
Translation parity	A published entry in one language only
Topic-sector consistency	A topic tagged outside its sector
Synthetic-only	A dataset that is not declared synthetic
Programme spine	A published course missing from the curriculum map

What this platform does

- Four more run in `pnpm test` rather than in the build — because they need `node:fs` and the build prerenders inside a...
- One of them checks a relation the reference graph structurally cannot — The reference rules verify that a declared slug...
- Any "every X has at least one Y" rule needs a test of that shape — A reference graph is the wrong tool for it

The rule worth taking

- Any field naming a shipped file needs a `ready` flag and a filesystem test beside it — This platform learned it three…
- A path in frontmatter is a string, and nothing in a build can tell whether it points at anything — So the schema…

Report it whole — Example

Pipeline checks

The loader validates every raw export before anything downstream runs: required columns present, 2,000-3,000 rows, water_source values within the known set, household_id unique, district at least 99% complete.

Implausible litres-per-person values are tolerated up to 20 rows, which is the documented unit-entry defect; above that the run stops.

Row counts are asserted across every join. A join that changes the row count stops the pipeline rather than producing a summary.

All checks raise rather than warn. The March run stopped on an unknown water_source value ("piped-shared"), which turned out to be a new form option added upstream; it is now mapped explicitly in src/clean.py.

Report it whole

- The last sentence is what makes the section credible — A checks section that has never caught anything is a checks. . .

What comes next

- Everything so far assumes you are still here.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/reproducible-workflows/repro-06-fail-loudly