Proving the join did what you said

A reconciliation table from every join — matched rows, both anti-join counts, the row count before and after — plus the suffix collision that overwrites a column without telling you.

Pierre Robentz Cassion

Lesson 2 of 8

Joining and Reshaping Programme Data — Joins you can prove

What this lesson covers

- The check has to be cheaper than the bug
- Declare the relationship, always
- The reconciliation table
- Guard the row count when you meant to preserve it
- The unmatched rate is an indicator, not an error
- Normalise the key before you blame the join
- The suffix collision
- Keys with different names
- What comes next

The check has to be cheaper than the bug

- Nobody skips join checks because they think joins are safe.

Declare the relationship, always — In Python

```
joined = attendance.merge(
    roster, on="student_id", how="left", validate="many_to_one",
)
```

Declare the relationship, always — In R

```r
joined <- attendance |>
  left_join(roster, by = "student_id", relationship = "many-to-one")
```

Declare the relationship, always

You expect	**pandas** `validate=`	**dplyr** `relationship=`
One row each side	`one_to_one`	`one-to-one`
Many left, one right	`many_to_one`	`many-to-one`
One left, many right	`one_to_many`	`one-to-many`
Genuinely many-to-many	omit	`many-to-many`

Declare the relationship, always

- dplyr warns on an unexpected many-to-many even without the argument — pandas does not

The reconciliation table — In Python (cont.)

```
def reconcile(left, right, on, left_name="left", right_name="right"):
    left_keys = left[on].drop_duplicates()
    right_keys = right[on].drop_duplicates()
    merged = left_keys.merge(right_keys, on=on, how="outer", indicator=True)
    counts = merged["_merge"].value_counts()

    return pd.Series({
        f"{left_name} rows": len(left),
        f"{right_name} rows": len(right),
        "keys matched": int(counts.get("both", 0)),
        f"{left_name} only": int(counts.get("left_only", 0)),
        f"{right_name} only": int(counts.get("right_only", 0)),
        f"{left_name} unmatched %": round(
            100 * counts.get("left_only", 0) / max(len(left_keys), 1), 2
        ),
    })
```

The reconciliation table — In Python (cont.)

```
print(reconcile(attendance, roster, ["student_id"], "attendance", "roster"))
```

The reconciliation table — In R (cont.)

```r
reconcile <- function(left, right, on, left_name = "left", right_name = "right") {
  left_keys  <- dplyr::distinct(dplyr::select(left, dplyr::all_of(on)))
  right_keys <- dplyr::distinct(dplyr::select(right, dplyr::all_of(on)))

  tibble::tibble(
    measure = c(paste(left_name, "rows"), paste(right_name, "rows"),
                "keys matched", paste(left_name, "only"), paste(right_name, "only")),
    value = c(
      nrow(left), nrow(right),
      nrow(dplyr::inner_join(left_keys, right_keys, by = on)),
      nrow(dplyr::anti_join(left_keys, right_keys, by = on)),
      nrow(dplyr::anti_join(right_keys, left_keys, by = on))
    )
  )
}
```

The reconciliation table — In R (cont.)

```
reconcile(attendance, roster, "student_id", "attendance", "roster")
```

The reconciliation table

measure	value
attendance rows	70,245
roster rows	1,200
keys matched	1,200
attendance only	0
roster only	0

Guard the row count when you meant to preserve it — In Python

```
def join_preserving(left, right, on, how="left", **kwargs):
    before = len(left)
    out = left.merge(right, on=on, how=how, **kwargs)
    if len(out) != before:
        raise ValueError(f"join changed row count: {before} -> {len(out)}")
    return out
```

Guard the row count when you meant to preserve it — In R

```r
join_preserving <- function(left, right, by, ...) {
  before <- nrow(left)
  out <- dplyr::left_join(left, right, by = by, ...)
  if (nrow(out) != before) {
    stop(sprintf("join changed row count: %d -> %d", before, nrow(out)))
  }
  out
}
```

The unmatched rate is an indicator, not an error — In Python

```
summary = reconcile(distributions, registration, ["beneficiary_id"])
if summary["left unmatched %"] > 5:
    print(f"WARNING: {summary['left unmatched %']}% of distribution rows "
          "have no registration record")
```

The unmatched rate is an indicator, not an error — In R

```r
unmatched <- nrow(anti_join(distributions, registration, by = "beneficiary_id"))
share <- unmatched / nrow(distributions)
if (share > 0.05) warning(sprintf("%.1f%% of distributions have no registration", 100 * share
    ))
```

Normalise the key before you blame the join

- **Type.** A facility code read as an integer on one side and a string on the other. `1042` never equals `"1042"`, and…
- **Whitespace and case.** `"FAC001 "` and `"fac001"` are three different keys.
- **Dates against date-times.** `2024-03-01` and `2024-03-01 00:00:00` compare equal in some libraries and not others; a…

Normalise the key before you blame the join — In Python

```
def key_clean(series):
    return series.astype("string").str.strip().str.upper()


for frame in (registers, aggregates):
    frame["facility_id"] = key_clean(frame["facility_id"])
```

Normalise the key before you blame the join — In R

```r
key_clean <- function(x) toupper(stringr::str_squish(as.character(x)))

registers  <- registers  |> mutate(facility_id = key_clean(facility_id))
aggregates <- aggregates |> mutate(facility_id = key_clean(facility_id))
```

Normalise the key before you blame the join

- Do this to both sides in the same function — so they cannot drift

The suffix collision — In Python

```
merged = registers.merge(aggregates, on="facility_id", suffixes=("_register", "_dhis2"))
```

The suffix collision — In R

```
merged <- registers |>
  left_join(aggregates, by = "facility_id", suffix = c("_register", "_dhis2"))
```

The suffix collision — In Python

```
overlap = (set(registers.columns) & set(aggregates.columns)) - {"facility_id"}
print("columns in both:", sorted(overlap))
```

The suffix collision — In R

```
setdiff(intersect(names(registers), names(aggregates)), "facility_id")
```

Keys with different names — In Python

```
merged = cases.merge(
    facilities, left_on="site_code", right_on="facility_id", how="left",
    validate="many_to_one",
)
```

Keys with different names — In R

```r
merged <- cases |>
  left_join(facilities, by = c("site_code" = "facility_id"),
            relationship = "many-to-one")
```

What comes next

- Every check so far has assumed you know what one row of each table is.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/joining-and-reshaping/joining-02-proving-the-join