Prove the key before you join

Every join makes a claim about uniqueness that nothing checks. Test it, watch two duplicated roster rows fan a 70,245-row table out by 120, and learn to assert the row count instead of inspecting it.

Pierre Robentz Cassion

Lesson 3 of 8

Data Cleaning and Validation — Identity and duplication

What this lesson covers

- The assumption nothing checks
- A key is a claim, and it is testable
- Composite keys, and the one you actually have
- What a broken key does to a join
- Say what you expect, and let it raise
- Guard the row count when the join is the point
- The rows on one side and not the other
- Columns that were never meant to be keys
- Assert, do not inspect
- What comes next

The assumption nothing checks

- You have a student roster and a daily attendance file.

A key is a claim, and it is testable — In Python

```
def is_key(df, columns):
    subset = df[columns]
    return {
        "rows": len(df),
        "distinct": len(subset.drop_duplicates()),
        "missing_any": int(subset.isna().any(axis=1).sum()),
        "is_key": len(subset.drop_duplicates()) == len(df)
                  and not subset.isna().any(axis=None),
    }


print(is_key(roster, ["student_id"]))
print(is_key(attendance, ["student_id", "attendance_date"]))
```

A key is a claim, and it is testable — In R

```r
is_key <- function(df, columns) {
  subset <- dplyr::select(df, dplyr::all_of(columns))
  list(
    rows        = nrow(df),
    distinct    = nrow(dplyr::distinct(subset)),
    missing_any = sum(!stats::complete.cases(subset)),
    is_key      = nrow(dplyr::distinct(subset)) == nrow(df) &&
                  !any(is.na(subset))
  )
}

is_key(roster, "student_id")
is_key(attendance, c("student_id", "attendance_date"))
```

A key is a claim, and it is testable — In Python

```
repeated = roster[roster["student_id"].duplicated(keep=False)]
print(repeated.sort_values("student_id"))
```

A key is a claim, and it is testable — In R

```
roster |>
  group_by(student_id) |>
  filter(n() > 1) |>
  arrange(student_id)
```

A key is a claim, and it is testable

student_id	school_id	grade	feeding_programme
STU0150	SCH09	3	false
STU0150	SCH08	3	true
STU0896	SCH01	2	true
STU0896	SCH06	2	false

Composite keys, and the one you actually have — In Python

```
print(is_key(attendance, ["student_id", "attendance_date"]))
```

Composite keys, and the one you actually have — In R

```
is_key(attendance, c("student_id", "attendance_date"))
```

Composite keys, and the one you actually have — In Python

```
ROSTER_KEY = ["student_id"]          # intended; violated by 2 rows, see cleaning log
ATTENDANCE_KEY = ["student_id", "attendance_date"]
```

Composite keys, and the one you actually have — In R

```
ROSTER_KEY <- "student_id"           # intended; violated by 2 rows, see cleaning log
ATTENDANCE_KEY <- c("student_id", "attendance_date")
```

What a broken key does to a join — In Python

```
joined = attendance.merge(roster, on="student_id", how="left")
print(len(attendance), "->", len(joined))
```

What a broken key does to a join — In R

```
joined <- attendance |> left_join(roster, by = "student_id")
cat(nrow(attendance), "->", nrow(joined), "\n")
```

Say what you expect, and let it raise — In Python

```
joined = attendance.merge(
    roster,
    on="student_id",
    how="left",
    validate="many_to_one",   # many attendance rows, one roster row
)
```

Say what you expect, and let it raise — In R

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

Say what you expect, and let it raise — Example

```
MergeError: Merge keys are not unique in right dataset; not a many-to-one merge
```

Say what you expect, and let it raise — Example

```
Error in `left_join()`:
! Each row in `x` must match at most 1 row in `y`.
i Row 1 of `x` matches multiple rows in `y`.
```

Guard the row count when the join is the point — In Python

```python
before = len(attendance)
joined = attendance.merge(roster, on="student_id", how="left")
assert len(joined) == before, f"join changed row count: {before} -> {len(joined)}"
```

Guard the row count when the join is the point — In R

```
before <- nrow(attendance)
joined <- attendance |> left_join(roster, by = "student_id")
stopifnot(nrow(joined) == before)
```

The rows on one side and not the other — In Python

```
left_only = set(attendance["student_id"]) - set(roster["student_id"])
right_only = set(roster["student_id"]) - set(attendance["student_id"])
print(f"{len(left_only)} students with attendance and no roster row")
print(f"{len(right_only)} students on the roster with no attendance")
```

The rows on one side and not the other — In R

```
setdiff(attendance$student_id, roster$student_id) |> length()
setdiff(roster$student_id, attendance$student_id) |> length()
```

The rows on one side and not the other

- **Attendance without a roster row** — a student marked present who is not enrolled. Either the roster is incomplete or...
- **Roster rows with no attendance** — enrolled students never marked either way. In an education programme that is not a...

Columns that were never meant to be keys

- **Duplicates create false merges.** Two households whose head is called Jean Baptiste become one household.
- **Variants create false splits.** `Jean Baptiste`, `JEAN BAPTISTE` and `Jean Baptiste` become three.

Assert, do not inspect — In Python

```python
def assert_key(df, columns, name):
    duplicated = df.duplicated(subset=columns, keep=False)
    if duplicated.any():
        offenders = df.loc[duplicated, columns].drop_duplicates()
        raise ValueError(
            f"{name}: {duplicated.sum()} rows violate the key {columns}\n"
            f"{offenders.head(10)}"
        )
```

Assert, do not inspect — In R

```r
assert_key <- function(df, columns, name) {
  dup <- duplicated(df[columns]) | duplicated(df[columns], fromLast = TRUE)
  if (any(dup)) {
    stop(sprintf("%s: %d rows violate the key %s", name, sum(dup),
                 paste(columns, collapse = " + ")))
  }
  invisible(df)
}
```

Assert, do not inspect

A check you ran once tells you about the file you had. A check that runs on read tells you about the file you have.

What comes next

- `assert_key` finds the students recorded twice under the *same* identifier.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/data-cleaning-and-validation/cleaning-03-keys-and-uniqueness