The rows that were never written

A missing row has no value to be missing. Build the grid the data should have filled, find where 1,755 attendance rows went, and learn why a complete grid still does not prove everyone reported.

Pierre Robentz Cassion

Lesson 6 of 8

Joining and Reshaping Programme Data — Making the pieces line up

What this lesson covers

- A missing row has no value to be missing
- Build the grid
- Where all 1,755 went
- Absent is not absent
- A complete grid does not prove everyone reported
- Period keys that sort
- Aligning a daily register to a monthly aggregate
- What comes next

A missing row has no value to be missing

- Every check in the cleaning course looked at values: blanks, sentinels, implausible numbers.

Build the grid — In Python

```
students = roster["student_id"].drop_duplicates()
school_days = attendance["attendance_date"].drop_duplicates()

grid = pd.MultiIndex.from_product(
    [students, school_days], names=["student_id", "attendance_date"]
).to_frame(index=False)

complete = grid.merge(attendance, on=["student_id", "attendance_date"], how="left")

print(f"grid {len(grid):,}, actual {len(attendance):,}, "
      f"missing {len(grid) - len(attendance):,}")
```

Build the grid — In R

```r
grid <- tidyr::expand_grid(
  student_id = unique(roster$student_id),
  attendance_date = unique(attendance$attendance_date)
)

complete <- grid |> left_join(attendance, by = c("student_id", "attendance_date"))

cat(sprintf("grid %d, actual %d, missing %d\n",
            nrow(grid), nrow(attendance), nrow(grid) - nrow(attendance)))
```

Build the grid — In R

```
complete <- attendance |>
  tidyr::complete(student_id, attendance_date)
```

Where all 1,755 went — In Python

```
missing = complete[complete["present"].isna()]
by_school = (
    missing.merge(roster[["student_id", "school_id"]], on="student_id")
    .groupby("school_id")
    .agg(missing_rows=("present", "size"),
         days=("attendance_date", "nunique"))
)
print(by_school)
```

Where all 1,755 went — In R

```
complete |>
  filter(is.na(present)) |>
  left_join(select(roster, student_id, school_id), by = "student_id") |>
  summarise(missing_rows = n(), days = n_distinct(attendance_date), .by = school_id)
```

Where all 1,755 went

school_id	missing rows	days
SCH07	915	15
SCH18	840	15

Absent is not absent — In Python

```
naive = complete["present"].fillna(False)
print("attendance rate treating gaps as absences:", naive.mean())
```

Absent is not absent — In R

```
complete |> summarise(rate = mean(coalesce(present, FALSE)))
```

Absent is not absent — In Python

```
open_days = attendance.merge(roster[["student_id", "school_id"]], on="student_id")
school_calendar = open_days[["school_id", "attendance_date"]].drop_duplicates()

grid = (
    roster[["student_id", "school_id"]]
    .merge(school_calendar, on="school_id")
)
print(len(grid), "student-days the schools were actually open")
```

Absent is not absent — In R

```r
school_calendar <- attendance |>
  left_join(select(roster, student_id, school_id), by = "student_id") |>
  distinct(school_id, attendance_date)

grid <- roster |>
  select(student_id, school_id) |>
  left_join(school_calendar, by = "school_id", relationship = "many-to-many")
```

Absent is not absent

- Build the grid per school, not per district — The universe of periods is a property of the reporting unit, and assuming...

Absent is not absent

Deciding whether a gap is a zero, a missing value or a period that should not exist is not a technical question. It is the analysis, and it belongs in the log with its reason.

A complete grid does not prove everyone reported — In Python

```
expected = (
    vax["facility_id"].nunique()
    * vax["period"].nunique()
    * vax["antigen"].nunique()
)
print(expected, "expected rows;", len(vax), "actual")
```

A complete grid does not prove everyone reported — In R

```
c(expected = n_distinct(vax$facility_id) * n_distinct(vax$period) * n_distinct(vax$antigen),
  actual = nrow(vax))
```

A complete grid does not prove everyone reported — In Python

```
coverage = (
    vax[vax["report_submitted"]]
    .groupby(["period", "antigen"])
    .agg(doses=("doses_administered", "sum"), target=("target_population", "sum"))
)
reporting = vax.groupby(["period", "antigen"])["report_submitted"].mean()
```

A complete grid does not prove everyone reported — In R

```
vax |>
  summarise(
    reporting_rate = mean(report_submitted),
    doses  = sum(doses_administered[report_submitted]),
    target = sum(target_population[report_submitted]),
    .by = c(period, antigen)
  )
```

Period keys that sort — In Python

```
vax["period"] = pd.to_datetime(vax["period"])
vax["month"] = vax["period"].dt.strftime("%Y-%m")      # 2024-08, sorts correctly
```

Period keys that sort — In R

```
vax <- vax |> mutate(month = format(period, "%Y-%m"))
```

Period keys that sort

- The month must carry its year — A twelve-month grid keyed on `month` alone silently merges August 2023 and August 2024…

Aligning a daily register to a monthly aggregate — In Python

```
monthly = (
    attendance.assign(month=attendance["attendance_date"].dt.strftime("%Y-%m"))
    .merge(roster[["student_id", "school_id"]], on="student_id")
    .groupby(["school_id", "month"])
    .agg(marks=("present", "size"),
         present=("present", "sum"))
    .reset_index()
)
```

Aligning a daily register to a monthly aggregate — In R

```
monthly <- attendance |>
  mutate(month = format(attendance_date, "%Y-%m")) |>
  left_join(select(roster, student_id, school_id), by = "student_id") |>
  summarise(marks = n(), present = sum(present %in% TRUE), .by = c(school_id, month))
```

Aligning a daily register to a monthly aggregate — In Python

```
last = monthly["month"].max()
print(f"excluding partial period {last}")
monthly = monthly[monthly["month"] < last]
```

Aligning a daily register to a monthly aggregate — In R

```
monthly <- monthly |> filter(month < max(month))
```

What comes next

- You now have a register aggregated to the same grain as the monthly return.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/joining-and-reshaping/joining-06-time-and-periods