Prouvez la clé avant de joindre

Toute jointure formule une hypothèse d'unicité que rien ne vérifie. La tester, voir deux lignes de liste dupliquées gonfler de 120 lignes une table de 70 245, et apprendre à affirmer le nombre de lignes plutôt qu'à le regarder.

Pierre Robentz Cassion

Leçon 3 sur 8

Nettoyage et validation des données — Identité et doublons

Ce que couvre cette leçon

- L'hypothèse que rien ne vérifie
- Une clé est une affirmation, et elle est testable
- Clés composites, et celle dont vous disposez vraiment
- Ce qu'une clé cassée fait à une jointure
- Dites ce que vous attendez, et laissez la jointure échouer
- Protégez le nombre de lignes quand la jointure est l'objectif
- Les lignes d'un côté et pas de l'autre
- Des colonnes qui n'étaient pas faites pour être des clés
- Affirmez, ne regardez pas
- La suite

L'hypothèse que rien ne vérifie

- Vous avez une liste d'élèves et un fichier de présence journalière.

Une clé est une affirmation, et elle est testable — En 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"]))
```

Une clé est une affirmation, et elle est testable — En 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"))
```

Une clé est une affirmation, et elle est testable — En Python

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

Une clé est une affirmation, et elle est testable — En R

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

Une clé est une affirmation, et elle est testable

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

Clés composites, et celle dont vous disposez vraiment — En Python

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

Clés composites, et celle dont vous disposez vraiment — En R

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

Clés composites, et celle dont vous disposez vraiment — En Python

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

Clés composites, et celle dont vous disposez vraiment — En R

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

Ce qu'une clé cassée fait à une jointure — En Python

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

Ce qu'une clé cassée fait à une jointure — En R

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

Dites ce que vous attendez, et laissez la jointure échouer — En Python

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

Dites ce que vous attendez, et laissez la jointure échouer — En R

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

Dites ce que vous attendez, et laissez la jointure échouer — Exemple

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

Dites ce que vous attendez, et laissez la jointure échouer — Exemple

```
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`.
```

Protégez le nombre de lignes quand la jointure est l'objectif — En 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)}"
```

Protégez le nombre de lignes quand la jointure est l'objectif — En R

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

Les lignes d'un côté et pas de l'autre — En 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")
```

Les lignes d'un côté et pas de l'autre — En R

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

Les lignes d'un côté et pas de l'autre

- **De la présence sans ligne de liste** — un élève marqué présent sans être inscrit. Soit la liste est incomplète, soit…
- **Des lignes de liste sans aucune présence** — des élèves inscrits jamais marqués, dans un sens ni dans l'autre. Dans…

Des colonnes qui n'étaient pas faites pour être des clés

- **Les doublons créent de fausses fusions.** Deux ménages dont le chef s'appelle Jean Baptiste deviennent un seul ménage.
- **Les variantes créent de fausses scissions.** `Jean Baptiste`, `JEAN BAPTISTE` et `Jean Baptiste` en deviennent trois.

Affirmez, ne regardez pas — En 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)}"
        )
```

Affirmez, ne regardez pas — En 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)
}
```

Affirmez, ne regardez pas

Un contrôle exécuté une fois vous renseigne sur le fichier que vous aviez. Un contrôle exécuté à la lecture vous renseigne sur le fichier que vous avez.

La suite

- `assert_key` trouve les élèves enregistrés deux fois sous le *même* identifiant.

Lire la leçon complète, avec le code exécutable

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