The profile you run before you touch it

A fresh export is evidence until you edit it. Read it as raw text first, inventory every column, prove or disprove the key, and save the result as the record of what arrived.

Pierre Robentz Cassion

Lesson 1 of 8

Data Cleaning and Validation — Before you change anything

What this lesson covers

- The file is evidence until you edit it
- Read it as text first
- What a profile has to answer
- Shape, against an expectation you write down
- The column inventory
- Sentinels are not missing values, yet
- Is the key a key?
- Ranges, before you trust a summary
- Save the profile next to the data
- Comparing this export against the last one
- What comes next

The file is evidence until you edit it

- Six months from now someone will ask whether the August figure was always like that.

Read it as text first

- Read once with everything as a string — That read is for looking, not for computing

Read it as text first — In Python

```
import pandas as pd

PATH = "muac-screening-artibonite-2024.v1.csv"
raw = pd.read_csv(PATH, dtype="string", keep_default_na=False)

print(raw.shape)
print(raw.dtypes)
```

Read it as text first — In R

```
library(readr)
library(dplyr)

PATH <- "muac-screening-artibonite-2024.v1.csv"
raw <- read_csv(PATH, col_types = cols(.default = col_character()))

dim(raw)
```

What a profile has to answer

- **How much arrived?** Rows and columns, against what you expected.
- **What is each column really?** Its raw values, not the type a reader guessed.
- **Where are the holes?** Per column, and per site — never one global figure.
- **Is the key a key?** Not "does the column exist" but "is it unique".
- **What is out of range?** Against the sector's limits, not against the data's own.
- **What codes are in use?** Every distinct value of every categorical column.

Shape, against an expectation you write down — In Python

```
EXPECTED_COLUMNS = [
    "child_id", "commune", "screening_date", "age_months",
    "sex", "muac_mm", "oedema", "outcome",
]

print(f"{len(raw)} rows, {raw.shape[1]} columns")
print("missing columns:", set(EXPECTED_COLUMNS) - set(raw.columns))
print("unexpected columns:", set(raw.columns) - set(EXPECTED_COLUMNS))
```

Shape, against an expectation you write down — In R

```r
EXPECTED_COLUMNS <- c(
  "child_id", "commune", "screening_date", "age_months",
  "sex", "muac_mm", "oedema", "outcome"
)

cat(nrow(raw), "rows,", ncol(raw), "columns\n")
setdiff(EXPECTED_COLUMNS, names(raw))
setdiff(names(raw), EXPECTED_COLUMNS)
```

The column inventory — In Python

```
def inventory(df):
    rows = []
    for column in df.columns:
        values = df[column]
        counts = values.value_counts(dropna=False)
        rows.append({
            "column": column,
            "blank": int((values == "").sum()),
            "distinct": int(values.nunique(dropna=False)),
            "top": counts.index[0] if len(counts) else None,
            "top_n": int(counts.iloc[0]) if len(counts) else 0,
        })
    return pd.DataFrame(rows)


print(inventory(raw).to_string(index=False))
```

The column inventory — In R

```r
inventory <- function(df) {
  purrr::map_dfr(names(df), function(column) {
    values <- df[[column]]
    counts <- sort(table(values, useNA = "ifany"), decreasing = TRUE)
    tibble::tibble(
      column   = column,
      blank    = sum(values == "" | is.na(values)),
      distinct = dplyr::n_distinct(values),
      top      = names(counts)[1],
      top_n    = as.integer(counts[1])
    )
  })
}

print(inventory(raw), n = Inf)
```

The column inventory

Column	Blank	Distinct	Note
age_months	226	55	5.4% blank — lesson 2 asks where
oedema	44	4	four distinct values in a boolean column

The column inventory

- Four distinct values in a boolean column — is the finding

Sentinels are not missing values, yet — In Python

```
print(raw["muac_mm"].value_counts().head())
print("rows coded -99:", int((raw["muac_mm"] == "-99").sum()))
```

Sentinels are not missing values, yet — In R

```
raw |> count(muac_mm, sort = TRUE) |> head()
sum(raw$muac_mm == "-99")
```

Is the key a key? — In Python

```
duplicated_ids = raw["child_id"].duplicated(keep=False)
print("rows sharing a child_id:", int(duplicated_ids.sum()))
print("distinct ids involved:", raw.loc[duplicated_ids, "child_id"].nunique())
```

Is the key a key? — In R

```
raw |>
  group_by(child_id) |>
  filter(n() > 1) |>
  ungroup() |>
  summarise(rows = n(), ids = n_distinct(child_id))
```

Ranges, before you trust a summary — In Python

```
muac = pd.to_numeric(raw["muac_mm"], errors="coerce")
muac = muac.where(muac != -99)

print(muac.describe())
print(muac.nsmallest(10).tolist())
print(muac.nlargest(10).tolist())
```

Ranges, before you trust a summary — In R

```r
muac <- suppressWarnings(as.numeric(raw$muac_mm))
muac[muac == -99] <- NA

summary(muac)
head(sort(muac), 10)
head(sort(muac, decreasing = TRUE), 10)
```

Ranges, before you trust a summary

- Always look at the ten smallest and ten largest values of any measurement column
  — It costs one line and it is the...

Save the profile next to the data — In Python (cont.)

```python
from pathlib import Path
import json

profile = {
    "file": PATH,
    "rows": len(raw),
    "columns": list(raw.columns),
    "blank_by_column": {c: int((raw[c] == "").sum()) for c in raw.columns},
    "distinct_by_column": {c: int(raw[c].nunique()) for c in raw.columns},
    "duplicate_key_rows": int(raw["child_id"].duplicated(keep=False).sum()),
    "categorical_values": {
        c: sorted(raw[c].unique().tolist())
        for c in ["commune", "sex", "oedema", "outcome"]
    },
}
```

Save the profile next to the data — In Python (cont.)

```
Path("outputs/profiles").mkdir(parents=True, exist_ok=True)
Path("outputs/profiles/muac-2024-q4.json").write_text(json.dumps(profile, indent=2))
```

Save the profile next to the data — In R (cont.)

```r
profile <- list(
  file    = PATH,
  rows    = nrow(raw),
  columns = names(raw),
  blank_by_column    = sapply(raw, function(x) sum(x == "" | is.na(x))),
  distinct_by_column = sapply(raw, dplyr::n_distinct),
  duplicate_key_rows = sum(duplicated(raw$child_id) | duplicated(raw$child_id, fromLast =
    TRUE)),
  categorical_values = lapply(
    raw[c("commune", "sex", "oedema", "outcome")],
    function(x) sort(unique(x))
  )
)

dir.create(here::here("outputs", "profiles"), recursive = TRUE, showWarnings = FALSE)
jsonlite::write_json(
  profile,
```

Save the profile next to the data — In R (cont.)

```
  here::here("outputs", "profiles", "muac-2024-q4.json"),
  pretty = TRUE, auto_unbox = TRUE
)
```

Comparing this export against the last one — In Python

```
previous = json.loads(Path("outputs/profiles/muac-2024-q3.json").read_text())

for column, values in profile["categorical_values"].items():
    was, now = set(previous["categorical_values"][column]), set(values)
    if was != now:
        print(f"{column}: new {sorted(now - was)}, gone {sorted(was - now)}")

growth = (profile["rows"] - previous["rows"]) / previous["rows"]
print(f"row count moved {growth:.1%}")
```

Comparing this export against the last one — In R

```r
previous <- jsonlite::read_json(
  here::here("outputs", "profiles", "muac-2024-q3.json"),
  simplifyVector = TRUE
)

for (column in names(profile$categorical_values)) {
  was <- previous$categorical_values[[column]]
  now <- profile$categorical_values[[column]]
  if (!setequal(was, now)) {
    cat(column, ": new", setdiff(now, was), "| gone", setdiff(was, now), "\n")
  }
}

sprintf("row count moved %.1f%%", 100 * (profile$rows - previous$rows) / previous$rows)
```

Comparing this export against the last one

A profile you did not save is a profile you did not run. The value is entirely in being able to open it later.

What comes next

- You now know this register is missing 5.4% of its ages.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/data-cleaning-and-validation/cleaning-01-profile-on-arrival