Établir ce qui ne va pas dans les données

Profiler méthodiquement un export récent — non-réponse par site et par semaine, mesures implausibles, doublons sans clé propre, et codes qui se contredisent entre eux.

Pierre Robentz Cassion

Leçon 5 sur 8

Fondamentaux de l'analyse de données pour le suivi-évaluation

Ce que couvre cette leçon

- Profiler avant d'analyser
- Vérification 1 : la complétude, par groupe et non globalement
- Vérification 2 : les valeurs implausibles
- Vérification 3 : les doublons, y compris ceux sans clé propre
- Vérification 4 : les contradictions internes
- Vérification 5 : un même objet codé de plusieurs manières
- Une fonction de profilage à conserver
- La suite

Profiler avant d'analyser

- Vous disposez d'un tableau ordonné aux types corrects.

Vérification 1 : la complétude, par groupe et non globalement — En Python

```python
global_ = muac.isna().mean().sort_values(ascending=False)
print(global_)

par_commune = (
    muac.assign(age_manquant=muac["age_months"].isna())
    .groupby("commune")["age_manquant"]
    .agg(["mean", "size"])
    .sort_values("mean", ascending=False)
)
print(par_commune)
```

Vérification 1 : la complétude, par groupe et non globalement — En R

```r
muac |>
  summarise(across(everything(), ~ mean(is.na(.x)))) |>
  tidyr::pivot_longer(everything(), names_to = "colonne", values_to = "manquant") |>
  arrange(desc(manquant))

muac |>
  group_by(commune) |>
  summarise(
    age_manquant = mean(is.na(age_months)),
    n = n()
  ) |>
  arrange(desc(age_manquant))
```

Vérification 1 : la complétude, par groupe et non globalement — En Python

```python
gros_morne = muac[muac["commune"] == "Gros-Morne"].copy()
gros_morne["semaine"] = gros_morne["screening_date"].dt.to_period("W")

print(
    gros_morne.groupby("semaine")["age_months"]
    .apply(lambda s: s.isna().mean())
    .sort_values(ascending=False)
    .head()
)
```

Vérification 1 : la complétude, par groupe et non globalement — En R

```
muac |>
  filter(commune == "Gros-Morne") |>
  mutate(semaine = lubridate::floor_date(screening_date, "week")) |>
  group_by(semaine) |>
  summarise(age_manquant = mean(is.na(age_months)), n = n()) |>
  arrange(desc(age_manquant))
```

Vérification 1 : la complétude, par groupe et non globalement

Une non-réponse qui se concentre sur un site ou une semaine fait la différence entre perdre de la précision et perdre la réponse. Ventilez-la toujours avant de décider quoi en faire.

Vérification 2 : les valeurs implausibles — En Python

```
implausibles = muac[(muac["muac_mm"] < 80) | (muac["muac_mm"] > 220)]
print(implausibles[["child_id", "commune", "age_months", "muac_mm"]])

print(muac["muac_mm"].describe())
```

Vérification 2 : les valeurs implausibles — En R

```r
muac |>
  filter(muac_mm < 80 | muac_mm > 220) |>
  select(child_id, commune, age_months, muac_mm)

summary(muac$muac_mm)
```

Vérification 2 : les valeurs implausibles — En Python

```
print(muac["age_months"].describe())
print(muac[(muac["age_months"] < 6) | (muac["age_months"] > 59)].shape[0])
```

Vérification 2 : les valeurs implausibles — En R

```r
summary(muac$age_months)
sum(muac$age_months < 6 | muac$age_months > 59, na.rm = TRUE)
```

Vérification 3 : les doublons, y compris ceux sans clé propre — En Python

```
exacts = muac[muac.duplicated(keep=False)]
print(f"{len(exacts)} lignes dans des groupes de doublons exacts")

ids_repetes = muac[muac.duplicated(subset=["child_id"], keep=False)]
print(f"{len(ids_repetes)} lignes partageant un child_id")
```

Vérification 3 : les doublons, y compris ceux sans clé propre — En R

```r
sum(duplicated(muac))

muac |>
  group_by(child_id) |>
  filter(n() > 1) |>
  arrange(child_id)
```

Vérification 3 : les doublons, y compris ceux sans clé propre — En Python

```
suspects = (
    muac.groupby(["commune", "screening_date", "age_months", "sex", "muac_mm"])
    .filter(lambda g: len(g) > 1)
    .sort_values(["commune", "screening_date", "muac_mm"])
)

print(suspects[["child_id", "commune", "screening_date", "age_months", "sex", "muac_mm"]])
```

Vérification 3 : les doublons, y compris ceux sans clé propre — En R

```r
muac |>
  group_by(commune, screening_date, age_months, sex, muac_mm) |>
  filter(n() > 1) |>
  arrange(commune, screening_date, muac_mm) |>
  select(child_id, commune, screening_date, age_months, sex, muac_mm)
```

Vérification 4 : les contradictions internes — En Python

```
contradiction_a = muac[
    muac["outcome"].isin(["referred-tsfp", "referred-otp", "referred-sc"])
    & muac["muac_mm"].isna()
]
print(len(contradiction_a))
```

Vérification 4 : les contradictions internes — En R

```r
muac |>
  filter(outcome %in% c("referred-tsfp", "referred-otp", "referred-sc"),
         is.na(muac_mm)) |>
  nrow()
```

Vérification 4 : les contradictions internes — En Python

```python
def decision_attendue(ligne):
    if pd.isna(ligne["muac_mm"]):
        return None
    if ligne["oedema"] is True or ligne["muac_mm"] < 115:
        return "referred-otp"
    if ligne["muac_mm"] < 125:
        return "referred-tsfp"
    return "no-action"

controle = muac.assign(attendu=muac.apply(decision_attendue, axis=1))
incoherents = controle[
    controle["attendu"].notna()
    & (controle["attendu"] == "no-action")
    & (controle["outcome"] != "no-action")
]
print(len(incoherents))
```

Vérification 4 : les contradictions internes — En R

```r
muac |>
  mutate(
    attendu = case_when(
      is.na(muac_mm)              ~ NA_character_,
      oedema | muac_mm < 115      ~ "referred-otp",
      muac_mm < 125               ~ "referred-tsfp",
      TRUE                        ~ "no-action"
    )
  ) |>
  filter(!is.na(attendu), attendu == "no-action", outcome != "no-action") |>
  nrow()
```

Vérification 5 : un même objet codé de plusieurs manières — En Python

```
brut = pd.read_csv(CHEMIN, dtype={"oedema": "string"})
print(brut.groupby("commune")["oedema"].value_counts(dropna=False))
```

Vérification 5 : un même objet codé de plusieurs manières — En R

```
read_csv(CHEMIN, col_types = cols(.default = col_character())) |>
  count(commune, oedema)
```

Une fonction de profilage à conserver — En Python

```python
def profiler(df, groupe=None):
    rapport = {
        "lignes": len(df),
        "colonnes": df.shape[1],
        "lignes_dupliquees": int(df.duplicated().sum()),
        "manquant": df.isna().mean().round(4).to_dict(),
    }
    if groupe:
        rapport["manquant_par_groupe"] = (
            df.isna().groupby(df[groupe]).mean().round(4).to_dict("index")
        )
    return rapport


import json
print(json.dumps(profiler(muac, groupe="commune"), indent=2, default=str))
```

Une fonction de profilage à conserver — En R

```r
profiler <- function(df, groupe = NULL) {
  out <- list(
    lignes = nrow(df),
    colonnes = ncol(df),
    lignes_dupliquees = sum(duplicated(df)),
    manquant = sapply(df, function(x) round(mean(is.na(x)), 4))
  )
  if (!is.null(groupe)) {
    out$manquant_par_groupe <- df |>
      group_by(.data[[groupe]]) |>
      summarise(across(everything(), ~ round(mean(is.na(.x)), 4)))
  }
  out
}

str(profiler(muac, groupe = "commune"))
```

La suite

- Vous savez maintenant ce qui ne va pas.

Lire la leçon complète, avec le code exécutable

https://data-analysis.cassion.dev/fr/cours/data-analysis-foundations/foundations-05-data-quality