Long, large, et le groupe répété entre les deux

Pivoter parce que l'analyse a besoin de la forme, pas parce que l'export est arrivé ainsi. Le groupe répété aplati, le tableau croisé DHIS2, et la valeur de remplissage qui transforme « non rapporté » en zéro.

Pierre Robentz Cassion

Leçon 4 sur 8

Ce que couvre cette leçon

- Deux formes, et l'usage de chacune
- Le groupe répété aplati
- Les emplacements vides, et ceux qui veulent dire quelque chose
- Du long au large — le tableau croisé DHIS2
- La valeur de remplissage qui invente des données
- Le tableau large que vous ne pouvez pas filtrer
- Aller-retour, et son contrôle
- La suite

Deux formes, et l'usage de chacune

- Long — une ligne par observation, une colonne par variable
- Large — une ligne par unité, une colonne par mesure

Deux formes, et l'usage de chacune

Le long sert à calculer. Le large sert à lire.

Le groupe répété aplati — Exemple

```
household_id, member_1_age, member_1_sex, member_2_age, member_2_sex, member_3_age, ...
```

Le groupe répété aplati — En Python

```
long = members_wide.melt(
    id_vars="household_id",
    var_name="field",
    value_name="value",
)
long[["slot", "attribute"]] = long["field"].str.extract(r"member_(\d+)_(\w+)")

members = (
    long.dropna(subset=["slot"])
    .pivot(index=["household_id", "slot"], columns="attribute", values="value")
    .reset_index()
)
```

Le groupe répété aplati — En R

```r
members <- members_wide |>
  tidyr::pivot_longer(
    cols = tidyr::starts_with("member_"),
    names_to = c("slot", "attribute"),
    names_pattern = "member_(\\d+)_(\\w+)",
    values_to = "value"
  ) |>
  tidyr::pivot_wider(names_from = attribute, values_from = value)
```

Le groupe répété aplati

- Deux pivots, pas un — Le premier transforme chaque colonne numérotée en lignes ; le second retransforme les noms. . .

Les emplacements vides, et ceux qui veulent dire quelque chose — En Python

```
print(members["age"].isna().sum(), "empty slots")

members = members[members["age"].notna() | members["sex"].notna()]
```

Les emplacements vides, et ceux qui veulent dire quelque chose — En R

```r
members |> summarise(empty = sum(is.na(age) & is.na(sex)))
```

```r
members <- members |> filter(!is.na(age) | !is.na(sex))
```

Les emplacements vides, et ceux qui veulent dire quelque chose — En Python

```
counted = members.groupby("household_id").size().rename("members_found")
check = households.join(counted, on="household_id")
mismatch = check[check["members_found"] != check["household_size"]]
print(len(mismatch), "households where the roster does not match household_size")
```

Les emplacements vides, et ceux qui veulent dire quelque chose — En R

```r
check <- households |>
  left_join(count(members, household_id, name = "members_found"), by = "household_id") |>
  filter(members_found != household_size)
```

Du long au large — le tableau croisé DHIS2 — En Python

```
wide = vax.pivot_table(
    index=["facility_id", "period"],
    columns="antigen",
    values="doses_administered",
    aggfunc="sum",
).reset_index()

print(len(vax), "->", len(wide))
```

Du long au large — le tableau croisé DHIS2 — En R

```r
wide <- vax |>
  tidyr::pivot_wider(
    id_cols = c(facility_id, period),
    names_from = antigen,
    values_from = doses_administered
  )

cat(nrow(vax), "->", nrow(wide), "\n")
```

Du long au large — le tableau croisé DHIS2

- Le `pivot` de pandas lève une erreur sur les doublons ; `pivot_table` les agrège — Prenez `pivot` d'abord, précisément…

La valeur de remplissage qui invente des données — En Python

```
wide = vax.pivot_table(
    index=["facility_id", "period"], columns="antigen",
    values="doses_administered", aggfunc="sum", fill_value=0,
)
```

La valeur de remplissage qui invente des données — En R

```
wide <- vax |>
  tidyr::pivot_wider(names_from = antigen, values_from = doses_administered,
                     values_fill = 0)
```

La valeur de remplissage qui invente des données — En Python

```
wide = wide.assign(reported=wide.notna().sum(axis=1))
```

La valeur de remplissage qui invente des données — En R

```
wide <- wide |> mutate(reported = rowSums(!is.na(across(bcg:penta3))))
```

Le tableau large que vous ne pouvez pas filtrer — En Python

```
low = vax[vax["doses_administered"] < 10]
```

Le tableau large que vous ne pouvez pas filtrer — En R

```
low <- vax |> filter(doses_administered < 10)
```

Aller-retour, et son contrôle — En Python

```
back = wide.melt(id_vars=["facility_id", "period"],
                 var_name="antigen", value_name="doses_administered").dropna()

assert len(back) == len(vax)
assert back["doses_administered"].sum() == vax["doses_administered"].sum()
```

Aller-retour, et son contrôle — En R

```
back <- wide |>
  tidyr::pivot_longer(-c(facility_id, period),
                      names_to = "antigen", values_to = "doses_administered") |>
  filter(!is.na(doses_administered))

stopifnot(nrow(back) == nrow(vax),
          sum(back$doses_administered) == sum(vax$doses_administered))
```

La suite

- Vous savez désormais donner à une table la forme qu'exige la question.

Lire la leçon complète, avec le code exécutable

https://data-analysis.cassion.dev/fr/cours/joining-and-reshaping/joining-04-long-and-wide