Six districts, trois districts

Des noms de lieux en texte libre face à une liste administrative — normaliser, apparier à l'identique, relire le reliquat, et stocker le résultat dans un fichier de correspondance plutôt que dans une suite de remplacements. Plus l'assertion qui arrête la cinquième graphie du prochain tour.

Pierre Robentz Cassion

Leçon 6 sur 8

Nettoyage et validation des données — Des valeurs qui ne peuvent pas être vraies

Ce que couvre cette leçon

- Le tableau à six lignes pour trois districts
- La liste administrative fait autorité
- Normalisez, puis appariez à l'identique
- Le reliquat, c'est le vrai travail
- Les appariements qu'il faut refuser
- La correspondance est un fichier, pas une suite de remplacements
- Appariez une fois, joignez sur le code pour toujours
- L'assertion qui attrape la cinquième graphie du prochain tour
- La suite

Le tableau à six lignes pour trois districts

district	Ménages	Sous 15 L/personne/jour
NORD-OUEST	33	21,2 %
nord-ouest	20	20,0 %
Sud-Est	803	16,4 %
Nord-Ouest	727	14,7 %
Nord Ouest	22	9,1 %
Centre	798	8,9 %

La liste administrative fait autorité — En Python

```
admin = pd.DataFrame([
    {"admin1_pcode": "HT07", "admin1_name": "Nord-Ouest"},
    {"admin1_pcode": "HT05", "admin1_name": "Centre"},
    {"admin1_pcode": "HT09", "admin1_name": "Sud-Est"},
])
```

La liste administrative fait autorité — En R

```r
admin <- tibble::tribble(
  ~admin1_pcode, ~admin1_name,
  "HT07",        "Nord-Ouest",
  "HT05",        "Centre",
  "HT09",        "Sud-Est"
)
```

La liste administrative fait autorité

- Une valeur des données qui ne se résout pas dans la liste est **non appariée**, pas fausse. Ce peut être une variante…
- Le produit de l'appariement est un **code**, et tout l'aval joint sur le code.
- C'est la liste, et non les données, qui décide du nombre de districts. Un tableau à six lignes échoue à un contrôle au…

Normalisez, puis appariez à l'identique — En Python (suite)

```
def match_key(series):
    return (
        series.fillna("")
        .str.normalize("NFKD")
        .str.encode("ascii", "ignore").str.decode("ascii")
        .str.lower()
        .str.replace(r"[^a-z0-9]+", " ", regex=True)
        .str.strip()
    )


wash["district_key"] = match_key(wash["district"])
admin["key"] = match_key(admin["admin1_name"])

matched = wash.merge(
    admin[["key", "admin1_pcode", "admin1_name"]],
```

Normalisez, puis appariez à l'identique — En Python (suite)

```
    left_on="district_key", right_on="key", how="left",
)
print(matched["admin1_pcode"].isna().sum(), "rows unmatched")
```

Normalisez, puis appariez à l'identique — En R

```r
match_key <- function(x) {
  x |>
    tidyr::replace_na("") |>
    stringi::stri_trans_general("Latin-ASCII") |>
    tolower() |>
    stringr::str_replace_all("[^a-z0-9]+", " ") |>
    stringr::str_squish()
}

wash  <- wash  |> mutate(district_key = match_key(district))
admin <- admin |> mutate(key = match_key(admin1_name))

matched <- wash |> left_join(admin, by = c("district_key" = "key"))
sum(is.na(matched$admin1_pcode))
```

Le reliquat, c'est le vrai travail — En Python

```
unmatched = (
    matched[matched["admin1_pcode"].isna()]
    .groupby("district")
    .size()
    .sort_values(ascending=False)
)
print(unmatched)
```

Le reliquat, c'est le vrai travail — En R

```
matched |>
  filter(is.na(admin1_pcode)) |>
  count(district, sort = TRUE)
```

Le reliquat, c'est le vrai travail

- Traitez la liste par nombre de lignes, pas par ordre alphabétique — Les valeurs non appariées forment presque toujours...

Le reliquat, c'est le vrai travail — En Python

```
from rapidfuzz import process, fuzz

for value in unmatched.index:
    best = process.extract(
        match_key(pd.Series([value]))[0],
        admin["key"].tolist(),
        scorer=fuzz.ratio,
        limit=3,
    )
    print(value, "->", best)
```

Le reliquat, c'est le vrai travail — En R

```r
for (value in unique(unmatched$district)) {
  scores <- stringdist::stringsim(match_key(value), admin$key, method = "jw")
  cat(value, "->", admin$admin1_name[order(-scores)][1:3], "\n")
}
```

Les appariements qu'il faut refuser

- **Contraignez par l'unité parente.** Un village n'a besoin d'être apparié qu'aux villages de sa propre commune, et une…
- **Refusez l'acceptation automatique en dessous d'un seuil élevé.** Une quasi correspondance sur un nom de deux syllabes…
- **Laissez non apparié ce qui n'est pas apparié.** Une ligne sans district est honnête et apparaît dans le tableau de…

Les appariements qu'il faut refuser

Le mode de défaillance n'est pas que le script n'arrive pas à apparier. C'est que le script apparie quelque chose, et que le ménage finit compté dans un district où il n'est pas.

La correspondance est un fichier, pas une suite de remplacements — En Python

```
mapping = pd.read_csv("reference/district-mapping.csv")   # raw_value, admin1_pcode,
    decided_by, decided_on
wash = wash.merge(mapping, left_on="district", right_on="raw_value", how="left")
```

La correspondance est un fichier, pas une suite de remplacements — En R

```r
mapping <- readr::read_csv(here::here("reference", "district-mapping.csv"))
wash <- wash |> left_join(mapping, by = c("district" = "raw_value"))
```

La correspondance est un fichier, pas une suite de remplacements

- Il est **relisable** par quelqu'un qui ne lit pas le code — en général la personne qui connaît les districts.
- Il est **réutilisable** par le notebook, le tableau de bord et le script du prochain tour.
- Il est **comparable**, si bien qu'ajouter une variante apparaît en revue comme une seule ligne.
- Il **porte qui a décidé**. `decided_by` et `decided_on` transforment une correspondance d'artefact technique en trace,. . .

Appariez une fois, joignez sur le code pour toujours — En Python

```
wash = wash.drop(columns=["district", "district_key"]).rename(
    columns={"admin1_pcode": "admin1"}
)
```

Appariez une fois, joignez sur le code pour toujours — En R

```
wash <- wash |> select(-district, -district_key) |> rename(admin1 = admin1_pcode)
```

L’assertion qui attrape la cinquième graphie du prochain tour — En Python

```
EXPECTED_DISTRICTS = {"HT05", "HT07", "HT09"}

seen = set(wash["admin1"].dropna())
assert seen <= EXPECTED_DISTRICTS, f"unexpected districts: {seen - EXPECTED_DISTRICTS}"
assert wash["admin1"].isna().sum() == 0, "rows with no district after mapping"
```

L’assertion qui attrape la cinquième graphie du prochain tour — En R

```r
EXPECTED_DISTRICTS <- c("HT05", "HT07", "HT09")

stopifnot(
  all(wash$admin1 %in% EXPECTED_DISTRICTS),
  !any(is.na(wash$admin1))
)
```

La suite

- Vous disposez maintenant de règles pour les valeurs et d'une correspondance pour les noms, appliquées l'une et l'autre à la main ce trimestre.

Lire la leçon complète, avec le code exécutable

https://data-analysis.cassion.dev/fr/cours/data-cleaning-and-validation/cleaning-06-harmonising-names