# Six districts, three districts

Free-text place names against an administrative list — normalise, match exactly, review what is left, and store the result as a mapping file rather than a chain of replacements. Plus the assertion that stops next round's fifth spelling.

---

Pierre Robentz Cassion

Lesson 6 of 8

Data Cleaning and Validation — Values that cannot be true

# What this lesson covers

- The table with six rows and three districts
- The administrative list is the authority
- Normalise, then match exactly
- What is left over is the actual work
- The matches you must refuse
- The mapping is a file, not a chain of replacements
- Match once, join on the code forever
- The assertion that catches next round's fifth spelling
- What comes next

# The table with six rows and three districts

| district | Households | Below 15 L/person/day |
|---|---|---|
| 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% |

# The administrative list is the authority — In Python

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

# The administrative list is the authority — In R

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

## The administrative list is the authority

- A value in the data that does not resolve to the list is **unmatched**, not wrong. It may be a spelling variant, a new…
- The output of matching is a **code**, and everything downstream joins on the code.
- The list, not the data, decides how many districts there are. A table with six rows fails a check rather than being…

# Normalise, then match exactly — In Python (cont.)

```
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"]],
```

# Normalise, then match exactly — In Python (cont.)

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

# Normalise, then match exactly — In 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))
```

# What is left over is the actual work — In Python

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

# What is left over is the actual work — In R

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

## What is left over is the actual work

- Work the list by row count, not alphabetically — The unmatched values are almost always a very short head and a long...

# What is left over is the actual work — In 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)
```

# What is left over is the actual work — In 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")
}
```

## The matches you must refuse

- **Constrain by the parent unit.** A village only needs to be matched against villages in its own commune, and a commune…
- **Refuse to auto-accept below a high threshold.** A near-match on a two-syllable name is not evidence. Where the top…
- **Leave unmatched values unmatched.** A row with no district is honest and shows up in the completeness table. A row…

## The matches you must refuse

The failure mode is not that the script cannot match. It is that the script matches something, and the household ends up counted in a district it is not in.

# The mapping is a file, not a chain of replacements — In 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")
```

# The mapping is a file, not a chain of replacements — In R

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

## The mapping is a file, not a chain of replacements

- It is **reviewable** by someone who does not read code — usually the person who knows the districts.
- It is **reusable** across the notebook, the dashboard and next round's script.
- It is **diffable**, so adding a variant shows up in review as one line.
- It **carries who decided**. `decided_by` and `decided_on` turn a mapping from a technical artefact into a record, and...

# Match once, join on the code forever — In Python

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

# Match once, join on the code forever — In R

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

# The assertion that catches next round's fifth spelling — In 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"
```

# The assertion that catches next round's fifth spelling — In R

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

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

## What comes next

- You now have rules for values and a mapping for names, and both were applied by hand this quarter.

## Read the full lesson, with runnable code

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