One template, twelve districts, no copy anywhere

Twelve district reports produced by copy-paste are twelve documents that will disagree by March. One parameterised template rendered twelve times is one document that cannot, and the mechanism is a single line in the frontmatter.

Pierre Robentz Cassion

Lesson 5 of 8

What this lesson covers

- The report that becomes twelve reports
- Quarto, which this platform already uses
- Rendering all twelve
- Writing a template that survives its own parameters
- What belongs in the template and what belongs in the pipeline
- The parameter you should always have
- Where this platform does the same thing
- Report it whole
- What comes next

The report that becomes twelve reports

- By March those twelve files disagree — Someone fixes the commune-name normalisation in one of them
- A parameterised template is one document — Fixing it fixes twelve reports, and there is nowhere for a difference to hide

Quarto, which this platform already uses — Example

```
---
title: "Nutrition surveillance: `r params$district`"
params:
  district: "Artibonite"
  as_of: "2025-03-31"
format: pdf
---
```

Quarto, which this platform already uses — Shell

```
quarto render report.qmd -P district:Nord-Ouest -P as_of:2025-03-31
```

Quarto, which this platform already uses

- The document declares its parameters and the renderer supplies them — The body refers to `params$district` — in R — or...

Quarto, which this platform already uses — In Python

```python
# | tags: [parameters]
district = "Artibonite"
as_of = "2025-03-31"
```

Quarto, which this platform already uses — In R

```
params$district
```

Quarto, which this platform already uses

- The default value in the frontmatter is what makes the template previewable — You open it, it renders for Artibonite,. . .

Rendering all twelve — In Python

```
import subprocess, pathlib

DISTRICTS = ["Artibonite", "Centre", "Grande-Anse", "Nippes", "Nord",
             "Nord-Est", "Nord-Ouest", "Ouest", "Sud", "Sud-Est"]

for district in sorted(DISTRICTS):
    slug = district.lower().replace(" ", "-")
    subprocess.run([
        "quarto", "render", "report.qmd",
        "-P", f"district:{district}",
        "-P", f"as_of:{AS_OF}",
        "--output", f"outputs/reports/{slug}.pdf",
    ], check=True)
```

Rendering all twelve — In R

```
for (d in sort(districts)) {
  quarto::quarto_render(
    "report.qmd",
    execute_params = list(district = d, as_of = as_of),
    output_file = paste0(tolower(gsub(" ", "-", d)), ".pdf")
  )
}
```

Rendering all twelve

- `check=True` is the important argument — Without it a failed render is a missing file that the loop steps over...
- Sort the district list — for the reason the last lesson gave: the loop's output order should not depend on how the list...

Writing a template that survives its own parameters

- No hard-coded numbers in the prose — Every figure in the text comes from the data

Writing a template that survives its own parameters — Example

```
Coverage in `r params$district` is
`r scales::percent(coverage, accuracy = 0.1)`, against a national figure of
`r scales::percent(national, accuracy = 0.1)`.
```

Writing a template that survives its own parameters

- Handle the district with no data — One of the twelve will have an empty table, and a template that assumes rows. . .

Writing a template that survives its own parameters — In Python

```
if summary.empty:
    print(f"No screening was conducted in {district} during this period.")
else:
    ...
```

Writing a template that survives its own parameters — In R

```
if (nrow(summary) == 0) cat("No screening was conducted in this period.")
```

Writing a template that survives its own parameters

- Handle the singular — "1 communes" in eleven reports is the sign of a template nobody proofread
- Handle the small denominator — A district with 40 screenings gets an interval three times wider than one with 400, and…

What belongs in the template and what belongs in the pipeline

In the template	In the pipeline
Layout, prose, the shape of the argument	Reading raw data
A summary table computed from prepared data	Cleaning and normalisation
Figures, drawn from prepared data	Any transformation more than a line or two
Caveats, thresholds and definitions	Anything two reports would share

What belongs in the template and what belongs in the pipeline

- The template should read a prepared file, not the raw export — If twelve renders each redo the cleaning, the cleaning…

What belongs in the template and what belongs in the pipeline — In Python

```
# run.py
clean()                          # once
build_figures()                  # once
for district in DISTRICTS:       # twelve times, from prepared data
    render(district)
```

What belongs in the template and what belongs in the pipeline — In R

```
# run.R, same shape
```

The parameter you should always have — Example

```
params:
  district: "Artibonite"
  as_of: "2025-03-31"
  data_version: "v1"
```

The parameter you should always have

- A data version parameter makes the report say which file it read — which is the provenance question a reader asks six...
- And an `as_of` parameter rather than the clock — for the reason lesson 4 gave: a report that says "generated 14 March"...

Where this platform does the same thing

- `slide-plan.mjs` plans the deck once from the lesson body — and the Beamer PDF, the PowerPoint file and the in-browser…
- The reasoning is the same as the twelve districts', at a different scale — A `slides:` array in the frontmatter would…
- The consequence for authors is real and worth naming — Because the deck is derived, lessons are written with lists,…

Report it whole — Example

District reports

Twelve district reports are rendered from one template, report.qmd, by run.py. No district-specific text exists outside the data.

Parameters: district, as_of (2025-03-31), data_version (v1). Each report prints all three in its colophon.

Cleaning and figure generation run once, before rendering; the template reads prepared data and performs no transformation.

Districts with no screening in the period render a stated "no data" section rather than an empty table.

Rendering is verified by `--check`: a failed render stops the run rather than leaving a missing file.

Report it whole

- The last line is what turns twelve rendered files into twelve reports you can send — A loop that swallows failures. . .

What comes next

- A pipeline that renders twelve reports from broken upstream data renders twelve broken reports.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/reproducible-workflows/repro-05-one-template-twelve-districts