CASSION · FORMATION

Chaînes d'analyse reproductibles

Rendre l'analyse réexécutable par la personne qui vous remplacera — gestion de versions, environnements figés, rapports paramétrés et une organisation de projet qui survit à une passation.

Formateur	Pierre Robentz Cassion
Niveau	Intermédiaire
Heures apprenant	18 h
Enseignée en	Python · R
Mise à jour	29 juillet 2026
Secteurs	Santé publique · Nutrition · Protection · Éducation · EAH
Normes et méthodologies	Critères d'évaluation du CAD de l'OCDE · Norme humanitaire fondamentale (CHS) · Définitions d'indicateurs de l'UNICEF

Consulter et exécuter la formation en ligne sur data-analysis.cassion.dev/fr/cours/reproducible-workflows

Ce que vous saurez faire

— Organiser un projet pour que les données brutes soient en lecture seule, les données dérivées jetables et le code la seule chose éditée

— Employer git sur un travail d'analyse, et tenir les données de bénéficiaires hors d'un dépôt définitivement

— Figer un environnement pour que l'ordinateur d'un collègue produise vos nombres

— Trouver et supprimer ce qui fait différer une réexécution — l'horloge, la graine, la locale, l'ordre des fichiers

— Produire douze rapports de district depuis un seul modèle paramétré

— Écrire une passation sur laquelle un successeur peut agir, et une vérification qui échoue bruyamment quand l'export amont casse

Prérequis

— Nettoyage et validation des données

Sommaire

À propos de cette formation

Une analyse que vous seul pouvez réexécuter est un brouillon. Le test n'est pas de savoir si elle fonctionne — c'est de savoir si elle fonctionne encore dans onze mois, sur l'ordinateur de quelqu'un d'autre, après votre départ, quand le fichier source a changé et que personne ne se rappelle de quel onglet venaient les nombres.

Ce cours a ceci d'inhabituel que son exemple travaillé n'est pas un jeu de données. **La plateforme que vous lisez est l'étude de cas**, et chaque affirmation de la dernière leçon se vérifie sur ce dépôt.

Trois de ces affirmations structurent tout le cours.

Vingt fichiers de données, dix-sept scripts générateurs, et les régénérer tous ne change rien. `pnpm datasets:generate` réécrit chaque CSV depuis un script à graine fixe, et `git status` ensuite ne signale aucun fichier modifié. C'est ce que « reproductible » signifie opérationnellement : réexécuter est une étape de *vérification*, non seulement de production.

Vingt fichiers de tests et 4 212 assertions, exécutés avant chaque compilation. Ils vérifient ce qu'un schéma ne peut pas — qu'un nombre de lignes déclaré correspond au fichier sur le disque, qu'une leçon française référence la figure française, que chaque cours publié livre ses deux éditions PDF. **La règle qu'ils encodent : tout champ nommant un fichier livré exige un test à côté**, parce qu'un chemin dans un frontmatter est une chaîne de caractères et que rien dans une compilation ne peut dire s'il pointe vers quelque chose.

Un schéma qui refuse les données non synthétiques. `dataQuality.synthetic` doit valoir `true` ou la compilation échoue, ce qui fait de « aucun enregistrement ne remonte à une personne réelle » une propriété du dépôt plutôt qu'une promesse dans une note — le même geste que produire les données plutôt que les collecter.

Python et R tout du long, avec `uv` et `renv` comme outils d'environnement et Quarto pour les rapports paramétrés. Il vous faut *Nettoyage et validation des données* d'abord : ce cours porte sur la façon de rendre une chaîne réexécutable, et la chaîne qu'il suppose est celle que ce cours-là construit.

Première partie

Le projet sur le disque

CHAPITRE 1

Trois sortes de fichiers, et une seule est éditable

Leçon 1 sur 8 · 135 min

Les données brutes sont en lecture seule, les données dérivées sont jetables, et le code est la seule chose que quiconque édite. Se tromper sur cette frontière est la façon dont une analyse devient irreproductible sans que personne s'en aperçoive pendant onze mois.

1.1 L'organisation

```
project/
  data/
    raw/            never edited, never written to, ideally chmod -w
    reference/      lookup tables, thresholds, admin boundary codes
  R/  or  src/      the only files anyone edits
  outputs/
    derived/        cleaned data, disposable
    figures/        charts and the CSV of values behind each
    reports/        rendered documents
  tests/
  README.md
  environment lock  uv.lock or renv.lock
```

Trois sortes de fichiers, et la distinction n'est pas une question de rangement.

Les données brutes sont en lecture seule. L'export de CommCare, le CSV de DHIS2, le tableur envoyé par le ministère. Une fois arrivé dans `data/raw/`, rien n'y écrit et personne ne l'ouvre dans Excel pour corriger une faute de frappe — un fichier brut corrigé est un fichier dont vous avez détruit la provenance.

Les données dérivées sont jetables. Tout ce qui est dans `outputs/` peut être supprimé à tout moment et reproduit en exécutant le code. Si supprimer `outputs/` fait perdre quelque chose, ce quelque chose n'était pas dérivé et a sa place ailleurs.

Le code est la seule chose éditée. Chaque correction, chaque recodage, chaque exclusion est une ligne de code, ce qui la rend visible, relisible et réexécutable.

1.2 Le test qui décide si c'est correct

```
rm -rf outputs/ && make all      # or: python run.py, Rscript run.R
git status --short               # should be empty
```

Supprimez chaque sortie, relancez, et le dépôt doit paraître intact. Cette seule commande est toute la reproductibilité en tant que propriété opérationnelle, et elle est inconfortable la première fois.

Trois façons dont cela échoue généralement, et chacune nomme quelque chose de réel :

Un fichier dans `outputs/` que rien ne produit. Quelqu'un l'a fait à la main, une fois, et chaque exécution depuis en dépend. C'est une donnée brute portant le mauvais nom.

Une étape qui ne s'exécute qu'en interactif. Une cellule dans un carnet, une ligne collée dans une console, un téléchargement manuel. Si ce n'est pas dans un script, cela n'a pas eu lieu.

Quelque chose qui doit être exécuté dans un ordre précis et ne le dit pas. La chaîne fonctionne sur votre machine parce que vous connaissez l'ordre.

1.3 Rendez les données brutes non modifiables, littéralement

```
chmod -R a-w data/raw/
```

```
# In R, the equivalent discipline is a function that only ever reads:
read_raw <- function(name) readr::read_csv(file.path("data/raw", name))
```

Cela coûte une commande et prévient l'échec qu'elle nomme. Un `to_csv("data/raw/survey.csv")` accidentel dans un carnet est la chose la plus destructrice que fasse un analyste, elle ne produit aucune erreur, et rien en aval ne peut la détecter après coup.

Là où le fichier brut est volumineux ou sensible, commitez plutôt sa somme de contrôle.

```
import hashlib, pathlib

def fingerprint(path: pathlib.Path) -> str:
    return hashlib.sha256(path.read_bytes()).hexdigest()[:16]

print(fingerprint(pathlib.Path("data/raw/survey-2025.csv")))
```

```
tools::md5sum("data/raw/survey-2025.csv")
```

Une somme de contrôle dans le dépôt transforme « est-ce le même export ? » en une question qui a une réponse. Quatre-vingt-dix pour cent des « les nombres ont changé et je ne sais pas pourquoi » sont un fichier source qui a changé sans que personne le remarque.

1.4 Le nommage, qui est de la provenance

Nom	Ce qu'il vous apprend
`survey.csv`	Rien
`final_survey_v2_FINAL.xlsx`	Qu'il y en a plusieurs et que celui-ci a gagné une dispute
`household-survey-2025.v1.csv`	Quoi, quand, et quelle version

Versionnez par le nom de fichier, jamais par écrasement. Les jeux de données de cette plateforme font exactement cela : une correction est livrée en `.v2.csv` et ne remplace jamais `.v1.csv`, parce qu'un carnet accroché à la v1 doit continuer de reproduire.

C'est pourquoi les fichiers de données peuvent être mis en cache de façon immuable, et pourquoi les PDF de cours — qui, eux, *sont* régénérés sur place — ne le peuvent délibérément pas.

1.5 Ce qui va dans le dépôt

Dedans	Dehors
Le code, chaque script	Les données brutes à caractère personnel
Le fichier de verrouillage d'environnement	Tout ce qui dépasse une cinquantaine de Mo
Les petites tables de référence	Identifiants, jetons, chaînes de connexion
Les sorties générées que la compilation ne peut pas reconstruire	`.Rhistory`, `__pycache__`, `.DS_Store`
Un README	Les sorties qui se régénèrent en quelques secondes

La quatrième ligne est un jugement plutôt qu'une règle. Cette plateforme commite ses PDF, diaporamas et figures produits — parce que le déploiement tourne sur Cloudflare sans TeX ni Python, si bien qu'un contributeur sans la chaîne d'outils peut quand même livrer un changement de contenu. **Commitez un artefact produit quand le reconstruire n'est pas à la portée de tous ceux qui en ont besoin**, et pas autrement.

1.6 Le README qui est réellement lu

Quatre sections, et il vieillit moins vite qu'un long.

```
# Nutrition surveillance analysis, Artibonite

## What this produces
A quarterly GAM estimate by commune, and the figures in the cluster report.

## Run it
    uv sync
    uv run python run.py

## Where the data comes from
data/raw/muac-screening-*.csv -- monthly export from the screening database,
downloaded by the M&E officer on the 5th. Checksums in data/raw/CHECKSUMS.

## What you need to know
Commune names arrive spelled four ways; normalisation is in src/clean.py and
must run before anything else.
```

La dernière section est celle qui épargne une semaine à un successeur. C'est la connaissance qui vit dans votre tête, et la leçon sur la passation porte sur la façon d'en sortir le reste.

1.7 Rapportez-le en entier

```
Analysis reproducibility

  Raw exports are read-only under data/raw/ with SHA-256 checksums committed.
  All cleaning, exclusion and recoding is in version-controlled code; no raw
  file has been edited.

  Every output in outputs/ is reproducible by `uv run python run.py` from a
  clean checkout. Deleting outputs/ and rerunning leaves the repository
  unchanged.

  Dataset files are versioned by filename. The v1 file referenced by the
  March report has not been modified; the correction ships as v2.
```

Le deuxième paragraphe est l'affirmation à faire et celle à tester avant de la faire. Elle se vérifie en une commande, et une analyse qui ne peut pas la passer devrait le dire plutôt que prétendre le contraire.

1.8 La suite

La gestion de versions est ce qui transforme « la seule chose éditée est le code » en un historique lisible. La leçon suivante porte sur l'emploi de git dans un travail d'analyse — et sur la seule chose qui ne doit jamais entrer dans un dépôt contenant des données de programme, parce que git est conçu pour ne jamais oublier.

CHAPITRE 2

Git n'oublie pas, ce qui en est l'intérêt et le danger

Leçon 2 sur 8 · 135 min

Un dépôt se souvient de chaque version de chaque fichier qu'il a jamais contenu, ce qui est exactement ce qu'on veut pour du code et exactement ce qu'on ne peut pas permettre pour une liste de bénéficiaires. Supprimer le fichier dans un commit ultérieur ne le retire pas.

2.1 Le commit qu'on ne peut pas reprendre

```
git add data/raw/beneficiaries.csv
git commit -m "add survey data"
# ... realised three days later ...
git rm data/raw/beneficiaries.csv
git commit -m "remove data"
```

Le fichier est toujours dans le dépôt. Il est dans l'historique, dans chaque clone, sur le dépôt distant, et dans la copie locale de tous ceux qui ont tiré. Le second commit l'a retiré de l'état *courant* et de rien d'autre.

Sur un dépôt contenant des données de protection ou de santé, c'est une divulgation. Non une erreur à corriger plus tard — une divulgation, avec les mêmes obligations que n'importe quelle autre.

D'où une règle préventive plutôt que corrective. On ne peut pas dé-commiter des données personnelles ; on ne peut que les empêcher d'entrer.

2.2 Le `.gitignore` qu'on écrit avant le premier commit

```
# Raw data. Nothing under here is ever committed.
data/raw/*
!data/raw/.gitkeep
!data/raw/CHECKSUMS

# Derived data, regenerated by the pipeline
outputs/

# Credentials, in every form they arrive in
.env
*.pem
*_secret*
config.local.*

# Editor and language noise
.Rhistory
.RData
__pycache__/
*.pyc
.DS_Store
.ipynb_checkpoints/
```

Écrivez-le en premier, commitez-le en premier. Un `.gitignore` ajouté après le premier commit est un `.gitignore` arrivé trop tard pour la chose qu'il devait attraper.

Les négations comptent. `data/raw/*` exclut tout, puis `!data/raw/CHECKSUMS` laisse revenir exactement le fichier qui prouve quel export a servi — la provenance sans le contenu.

2.3 Ce que « données personnelles » signifie ici, précisément

C'est plus large qu'une colonne de noms, et les jeux de données de ce secteur le démontrent.

Paraît anonyme	Ne l'est pas, parce que
Un identifiant de cas	Il se joint à un système de gestion de cas qui a le nom
Les coordonnées GPS d'un ménage	C'est une adresse
Date de naissance plus commune plus sexe	Trois champs identifient la plupart des gens dans une petite commune
La photographie d'un formulaire	Chaque champ qui y figure
Une colonne de notes en texte libre	Elle contient ce que le travailleur social a tapé

Le cours de protection a établi l'arithmétique : une ventilation à quatre entrées d'un jeu de données anonyme a produit dix-neuf cellules de un, et une cellule de un est une identification.

Le test n'est donc pas « y a-t-il une colonne de noms ». C'est de savoir si une combinaison quelconque de ce que contient le fichier pourrait désigner une personne — et dans une petite commune, elle le peut généralement.

2.4 La vérification qui s'exécute avant le commit

```
# .git/hooks/pre-commit -- or the pre-commit framework, or a CI step
if git diff --cached --name-only | grep -qE '^data/raw/'; then
  echo "Refusing to commit anything under data/raw/" >&2
  exit 1
fi
```

```
# The same check as an R function, run by whatever CI you have.
```

Un crochet qui refuse vaut mieux qu'une politique qui rappelle. Le mode d'échec est quelqu'un de pressé à 18 heures, et un rappel n'y survit pas.

Ajoutez une recherche des formes que prennent les secrets, parce qu'un jeton collé dans un script est l'autre cas courant. `gitleaks` et `detect-secrets` le font tous deux en une étape d'intégration continue.

2.5 Que faire si c'est déjà arrivé

L'ordre compte et la première étape n'est pas technique.

Un : signalez-le. Votre organisation a une procédure d'incident de données. C'en est un, et le caractère public ou non du dépôt décide de l'urgence, non de la nécessité.

Deux : renouvelez tout ce qui était un identifiant. Un jeton commité est compromis définitivement ; le supprimer n'y change rien.

Trois : réécrivez l'historique, avec `git filter-repo` ou BFG, et forcez la poussée. Chaque clone existant doit être supprimé et recloné — un clone que personne n'a recloné contient encore le fichier.

Quatre : supposez qu'il a été récupéré. Sur un dépôt public, supposez que le fichier a été téléchargé et copié, et traitez la divulgation comme complète.

```
git filter-repo --path data/raw/beneficiaries.csv --invert-paths
```

La troisième étape est la seule qui ressemble à une correction et c'est la moins importante.

2.6 Les commits sur un travail d'analyse

Les habitudes diffèrent de celles du développement logiciel sur deux points qu'il vaut la peine de nommer.

Commitez le code et la sortie ensemble quand la sortie est commitée. Une figure et le script qui l'a produite appartiennent au même commit, pour qu'un relecteur voie si le graphique correspond au changement.

Écrivez le pourquoi dans le message. `corrige les noms de communes` est ce que le diff montre déjà. « Regrouper quatre orthographes de Nord-Ouest ; sans regroupement, le pire district se fragmente et aucun fragment ne paraît alarmant » est ce dont un successeur a besoin et qu'il ne peut pas retrouver.

```
git commit -m "Group four spellings of Nord-Ouest before computing coverage

Ungrouped, the worst district splits into fragments of 727, 33, 22 and 20
households and none of them looks alarming. The 20-household fragment shows
0% open defecation, which a district table would report as a solved problem."
```

Une branche par question d'analyse, non par journée. Une branche qui répond à « la couverture diffère-t-elle par district » peut être relue et fusionnée ; une branche appelée `travail` non.

2.7 Ce que devrait contenir un dépôt d'analyse de programme

```
project/
  .gitignore          committed first
  README.md           what it produces, how to run it, what to know
  uv.lock             the environment, pinned
  src/                every transformation
  tests/              the checks that fail loudly
  data/
    raw/              .gitkeep and CHECKSUMS only
    reference/        committed: thresholds, admin codes, lookup tables
  outputs/            ignored
```

Les tables de référence sont commitées et les données brutes non, et la ligne qui les sépare est de savoir si le fichier décrit des personnes. Les standards de croissance de l'OMS, les seuils IPC et les codes de découpage administratif sont de la référence ; une liste de ménages non.

2.8 Rapportez-le en entier

```
Data handling in this analysis
```

```
No raw data is committed. data/raw/ is excluded by .gitignore, with only a
CHECKSUMS file tracked so the exports used can be identified.

A pre-commit hook refuses any staged file under data/raw/ and a CI step
scans for credential patterns.

Reference tables (WHO growth standards, IPC thresholds, admin codes) are
committed; they describe no individual.

The repository is private and access is limited to the M&E team. Raw
exports are held in [the organisation's storage], not here.
```

La dernière ligne est celle qu'on oublie et celle qu'un audit demande. Dire où *sont* les données compte autant que dire où elles ne sont pas.

2.9 La suite

Du code en gestion de versions ne reproduit toujours pas un nombre si l'environnement diffère. La leçon suivante le fige, et trouve les quatre choses qui font différer une réexécution alors que rien dans le dépôt n'a changé.

Deuxième partie

La même réponse deux fois

CHAPITRE 3

Ça marche sur ma machine, et c'est le rapport de bogue

Leçon 3 sur 8 · 135 min

Le même script, les mêmes données et une autre version de pandas produisent d'autres nombres. Un fichier de verrouillage est ce qui transforme « ça marche sur ma machine » d'une défense en une affirmation vérifiable, et cela coûte une commande.

3.1 Ce que l'ordinateur d'un collègue fait différemment

Vous envoyez un script et un CSV. Il l'exécute. Le nombre diffère.

Ce qui diffère	Ce que cela change
Version de paquet	Un argument par défaut, une règle d'arrondi, un ordre de tri
Version du langage	Ordre des dictionnaires, division entière, traitement des chaînes
Système d'exploitation	Fins de ligne, ordre des fichiers, séparateurs de chemin, locale
Locale	`1,234` interprété comme 1234 ou comme 1,234
Ce qui est déjà installé	Une bibliothèque que votre script importe sans jamais la déclarer

Rien de tout cela n'est dans votre dépôt, raison pour laquelle un code identique ne suffit pas. **Un fichier de verrouillage est la moitié manquante de l'analyse.**

3.2 Python : `uv`

```
uv init                     # creates pyproject.toml
uv add pandas matplotlib    # records the dependency and resolves it
uv run python run.py        # runs inside the pinned environment
```

Deux fichiers, et les deux sont commités. `pyproject.toml` dit ce que vous avez demandé — `pandas>=2.0` — et `uv.lock` dit exactement ce que vous avez obtenu, jusqu'à l'empreinte de chaque paquet de l'arbre.

Un collègue lance `uv sync` et dispose de votre environnement, sur son système, sans que vous sachiez ce qu'il avait installé auparavant.

Figez aussi la version du langage.

```
# pyproject.toml
requires-python = ">=3.12,<3.13"
```

Une plage de versions, non une version unique. Figer à `3.12.4` exactement signifie qu'un collègue en `3.12.7` ne peut pas l'exécuter du tout, ce qui est un échec pire que celui qu'on prévenait.

3.3 R : `renv`

```
renv::init()                # snapshots the project library
renv::snapshot()            # after adding a package
renv::restore()             # on the colleague's machine
```

`renv.lock` est l'artefact équivalent et il est commité. Il enregistre chaque paquet, sa version et le dépôt d'où il vient, et `renv::restore()` le reconstruit.

Enregistrez la version de R, ce que `renv` fait automatiquement, et vérifiez-la dans le script là où une différence compterait.

```
if (getRversion() < "4.3.0") stop("This analysis requires R >= 4.3.0")
```

```
# The Python equivalent, at the top of the entry point.
import sys
assert sys.version_info >= (3, 12), "This analysis requires Python 3.12+"
```

3.4 Pourquoi un fichier de verrouillage et non `requirements.txt`

```
pandas>=2.0
matplotlib
```

Ce fichier ne dit presque rien. Il se résout en versions différentes selon les jours et ne mentionne pas les cinquante paquets que ces deux-là entraînent. Une installation en mars et une en juillet depuis le même `requirements.txt` sont deux environnements différents.

	Déclare	**Reproduit**
`requirements.txt`	L'intention	Non
`requirements.txt` avec des `==`	Une couche	En partie — les dépendances transitives flottent
`uv.lock` / `renv.lock`	L'arbre entier	**Oui**

Commitez l'intention et le verrou. Le fichier d'intention est ce qu'un humain édite ; le verrou est ce depuis quoi une machine reproduit, et aucun ne remplace l'autre.

3.5 Ce que cette plateforme fige, et pourquoi un verrou est inhabituel

```
{
  "packageManager": "pnpm@11.9.0",
  "engines": { "node": ">=22" },
  "devDependencies": { "typescript": "^6.0.3" }
}
```

Le gestionnaire de paquets lui-même est figé, parce qu'un pnpm différent résout le fichier de verrouillage différemment, et c'est la couche en dessous de celle que la plupart des projets figent.

TypeScript est maintenu en 6.x délibérément. TypeScript 7 — le compilateur natif — n'expose pas encore l'API programmatique dont dépend `astro check`, si bien que `pnpm typecheck` échoue purement et simplement en 7. **Le verrou a une raison, la raison est écrite, et elle nomme la condition de sa levée.**

C'est la forme que devrait avoir un verrou. Une contrainte de version sans commentaire est une contrainte que personne n'osera retirer et que personne ne peut justifier de garder.

```
# pandas is pinned below 3.0 because the copy-on-write default changes the
```

```
# behaviour of the recode in src/clean.py:88. Revisit when that is rewritten.
pandas = ">=2.1,<3.0"
```

3.6 Les conteneurs, et quand ils en valent la peine

Un conteneur fige aussi le système d'exploitation, la seule couche qu'un fichier de verrouillage ne peut pas atteindre.

Ils en valent la peine quand l'analyse a des dépendances hors Python ou hors R — GDAL, une distribution TeX, un client de base de données —, ou quand elle doit tourner sans surveillance sur un serveur, ou quand le constat sera réexaminé des années plus tard.

Ils n'en valent pas la peine quand un fichier de verrouillage reproduit déjà le résultat et que le public est deux collègues avec des ordinateurs portables. Un conteneur ajoute une étape de construction, un registre et une compétence qu'une petite équipe de suivi-évaluation peut ne pas avoir, et le coût est réel.

```
FROM python:3.12-slim
COPY pyproject.toml uv.lock ./
RUN pip install uv && uv sync --frozen
COPY . .
CMD ["uv", "run", "python", "run.py"]
```

Commencez par le fichier de verrouillage. Passez au conteneur quand vous pouvez nommer la dépendance qu'il fige et que le verrou ne peut pas.

3.7 Le test qui le prouve

Reproduisez votre propre résultat sur une machine qui n'a jamais vu le projet. Un clone neuf dans un répertoire temporaire en fait l'essentiel.

```
git clone <repo> /tmp/check && cd /tmp/check
uv sync
uv run python run.py
diff -r outputs/ ~/project/outputs/
```

```
# renv::restore() then source("run.R"), and compare.
```

Si vous ne pouvez pas le faire, « c'est reproductible » n'est pas testé. L'ordinateur d'un collègue vaut mieux et un exécuteur d'intégration continue vaut mieux encore, parce que c'est une machine propre à chaque fois et qu'il s'exécute que quiconque y pense ou non.

3.8 Rapportez-le en entier

```
Computational environment

  Python 3.12, dependencies pinned in uv.lock (committed). Reproduce with:

      uv sync && uv run python run.py

  pandas is held below 3.0 because the copy-on-write default changes the
  recode in src/clean.py; this is revisited when that function is rewritten.
```

```
The pipeline runs on every push in CI on a clean machine, so the claim
that a fresh checkout reproduces these outputs is tested rather than
asserted.

Analysis run on 2026-03-14 with commit a3f9c21.
```

La dernière ligne est celle qui rend le reste utilisable un an plus tard. Un résultat sans le commit qui l'a produit se reproduit approximativement ; avec lui, exactement.

3.9 La suite

Un environnement figé produit encore une réponse différente à la seconde exécution si le code lit l'horloge, tire un nombre aléatoire, ou se fie à l'ordre dans lequel les fichiers reviennent. La leçon suivante trouve les quatre.

CHAPITRE 4

Quatre choses qui changent la réponse alors que rien n'a changé

Leçon 4 sur 8 · 135 min

L'horloge, la graine, l'ordre des fichiers et la locale. Chacune produit un résultat différent depuis un code identique et des données identiques, chacune est invisible dans un diff, et cette plateforme en a rencontré trois sur quatre en production.

4.1 Un : l'horloge

```
from datetime import date
report_date = date.today()          # different every day, by design
```

Tout ce qui lit l'heure courante fait différer chaque exécution. Un pied de page « produit le », un nom de fichier avec la date du jour, un filtre sur « les 30 derniers jours », une graine aléatoire dérivée du temps.

Le remède est de prendre la date des données ou d'un paramètre.

```
import pandas as pd

survey = pd.read_csv("data/raw/household-survey-2025.v1.csv")
as_of = pd.to_datetime(survey["interview_date"]).max()   # from the data
```

```
as_of <- max(as.Date(survey$interview_date))
```

Cette plateforme s'est trompée là-dessus et l'échec est instructif. Les documents de cours et les diaporamas de leçon prenaient à l'origine leur date de production du champ `updated` le plus récent de toute la bibliothèque de contenu. Déterministe — et faux quand même, parce que publier un nouveau cours réhorodatait chaque document existant et réécrivait les quatre-vingt-seize binaires de diaporama.

La date est désormais rattachée à l'artefact. Celle d'un document vient de son propre cours et de ses leçons, celle d'un diaporama de sa propre leçon, celle d'un rapport de son propre projet. **Une réponse fausse déterministe reste une réponse fausse**, et « le diff est énorme mais correct » est la façon dont personne ne le relit.

4.2 Deux : la graine

```
import numpy as np

rng = np.random.default_rng(20260729)      # stated, committed, reproducible
sample = rng.choice(households, size=200, replace=False)
```

```
set.seed(20260729)
sample(households, 200)
```

Tout ce qui est aléatoire exige une graine et la graine a sa place dans le code. Intervalles bootstrap, échantillonnage aléatoire pour vérification, simulation, partitions d'apprentissage et de test, points de nuage déplacés aléatoirement.

Fixez-la une fois, en tête, visiblement. Une graine enfouie trois fonctions plus bas est une graine que quelqu'un déplacera.

Et dites-le dans la sortie. L'intervalle bootstrap du cours de régression est rapporté avec « 400 rééchantillonnages, graine 20260729 » exactement pour cette raison — un intervalle que personne ne peut reproduire n'est pas un intervalle.

Chaque jeu de données de cette plateforme est produit par un script à graine fixe, ce qui fait de `pnpm datasets:generate` une étape de vérification : il réécrit les vingt CSV et `git status` ne signale aucun changement.

4.3 Trois : l'ordre dans lequel les fichiers reviennent

```
import glob
for path in glob.glob("data/raw/*.csv"):     # order is filesystem-dependent
    ...
```

`glob` et `os.listdir` renvoient les fichiers dans un ordre qui diffère selon les systèmes et parfois d'une exécution à l'autre. Si quoi que ce soit en aval dépend de l'ordre — une concaténation, une déduplication au premier arrivé, un index de ligne — le résultat diffère.

```
for path in sorted(glob.glob("data/raw/*.csv")):
    ...
```

```
for (path in sort(list.files("data/raw", full.names = TRUE))) { }
```

Triez. Toujours. Cela coûte six caractères.

La même chose vaut pour l'ordre des dictionnaires et des groupes. `groupby` dans pandas trie par défaut et `dplyr::group_by` non ; un graphique dont l'ordre des barres vient d'un groupement non trié est un graphique dont l'ordre peut changer sans que les données changent.

4.4 Quatre : la locale

```
float("1,234")           # ValueError, or 1.234, depending on where you are
```

Le séparateur décimal, le séparateur de milliers, le format de date et l'ordre de tri des caractères accentués dépendent tous de la locale, et le public de cette plateforme travaille dans les deux conventions par définition.

```
survey = pd.read_csv(path, decimal=".", thousands=None)     # stated, not inferred
dates = pd.to_datetime(survey["date"], format="%Y-%m-%d")   # explicit
```

```
readr::read_csv(path, locale = locale(decimal_mark = ".", date_format = "%Y-%m-%d"))
```

Énoncez le format plutôt que de laisser le lecteur l'inférer. `pd.to_datetime` sans format est une fonction qui devine, et elle devine différemment sur `03/04/2025` selon ce qu'il

y a d'autre dans la colonne.

Le tri est le plus subtil. `sorted()` sur des noms de communes français place `Étroit` après `Zone` sous une locale et avant `Fond` sous une autre, si bien que l'ordre des catégories d'un graphique devient dépendant de la machine.

4.5 Le test qui attrape les quatre

```
python run.py && cp -r outputs outputs-first
python run.py && diff -r outputs outputs-first
```

Exécutez deux fois et comparez. Tout ce qui diffère est l'une des quatre, et le diff nomme le fichier.

Exécutez deux fois sur des machines différentes et vous attrapez en plus la locale et l'ordre des fichiers, ce qu'une seule machine ne peut pas faire.

Mettez-le dans l'intégration continue, qui est une machine propre à chaque fois, et la vérification s'exécute que quiconque y pense ou non.

4.6 L'habitude de la sortie déterministe

Pour tout ce qu'une compilation produit, une sortie identique octet pour octet sur entrée inchangée est l'objectif, et il est atteignable plus souvent qu'on ne le suppose.

Source de bruit	Remède
Horodatages dans les métadonnées de fichier	`SOURCE_DATE_EPOCH`, ou les retirer
Une date de production dans un pied de page	La prendre du contenu
Une compression avec horodatage	Une archive dont les dates d'entrée sont fixées
L'ordre de sommation en virgule flottante	Trier avant de réduire là où cela compte
Un identifiant aléatoire incorporé	Le dériver d'une empreinte du contenu

Cette plateforme fixe `SOURCE_DATE_EPOCH` pour pdfTeX et pour pandoc, parce qu'un `.pptx` est une archive dont les dates d'entrée et les propriétés de document bougeraient sinon à chaque reconstruction de chaque diaporama.

Le bénéfice est qu'un diff signifie quelque chose. Quand reconstruire ne produit aucun changement, un changement dans le diff est un vrai changement — et la relecture devient possible.

4.7 Rapportez-le en entier

```
Determinism

  All randomness is seeded: seed 20260729, set in src/config.py and reported
  with every bootstrap interval in this report.

  Dates are taken from the data (the latest interview date) rather than from
  the clock. No output contains a generation timestamp.

  File iteration is sorted. Group order is stated explicitly rather than
  inherited from the grouping library's default.

  CSV reading states the decimal mark and the date format rather than
  inferring them.
```

```
Verified: running the pipeline twice produces byte-identical outputs, and
CI runs it on a clean machine on every push.
```

La dernière ligne est l'affirmation et les quatre au-dessus sont la façon dont elle a été obtenue. Un rapport qui affirme le déterminisme sans dire laquelle des quatre il a traitée a probablement traité la graine et aucune autre.

4.8 La suite

Une chaîne reproductible qui produit un rapport vaut la peine d'exister. La leçon suivante lui en fait produire douze, depuis un seul modèle, sans une seule copie.

Troisième partie

Un modèle, plusieurs sorties

CHAPITRE 5

Un modèle, douze districts, aucune copie

Leçon 5 sur 8 · 135 min

Douze rapports de district produits par copier-coller sont douze documents qui divergeront d'ici mars. Un modèle paramétré rendu douze fois est un document qui ne le peut pas, et le mécanisme tient en une ligne de frontmatter.

5.1 Le rapport qui devient douze rapports

Un cluster veut un rapport par district. L'approche évidente est d'en écrire un, de le copier onze fois et de changer le filtre en tête de chacun.

D'ici mars, ces douze fichiers divergent. Quelqu'un corrige la normalisation des noms de communes dans l'un. Quelqu'un ajoute une réserve dans trois. La définition de l'indicateur change et huit sont mis à jour. Rien dans aucune compilation ne peut détecter qu'ils ont dérivé, parce que ce sont douze documents sans lien.

Un modèle paramétré est un seul document. Le corriger corrige douze rapports, et une différence n'a nulle part où se cacher.

5.2 Quarto, que cette plateforme emploie déjà

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

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

Le document déclare ses paramètres et le moteur de rendu les fournit. Le corps renvoie à `params$district` — en R — ou à `district` depuis la cellule `parameters` injectée en Python, et rien d'autre dans le fichier ne mentionne un district précis.

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

```
params$district
```

La valeur par défaut du frontmatter est ce qui rend le modèle prévisualisable. Vous l'ouvrez, il se rend pour Artibonite, et vous éditez quelque chose que vous voyez.

5.3 Rendre les douze

```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)
```

```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")
  )
}
```

`check=True` **est l'argument important.** Sans lui, un rendu échoué est un fichier manquant que la boucle enjambe silencieusement, et vous le découvrez quand quelqu'un demande le rapport des Nippes.

Triez la liste des districts, pour la raison qu'a donnée la leçon précédente : l'ordre de sortie de la boucle ne devrait pas dépendre de la façon dont la liste a été assemblée.

5.4 Écrire un modèle qui survit à ses propres paramètres

Quatre choses qu'un rapport copié se permet et qu'un modèle ne peut pas.

Aucun nombre codé en dur dans la prose. Chaque chiffre du texte vient des données.

```
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)`.
```

Gérez le district sans données. L'un des douze aura un tableau vide, et un modèle qui suppose des lignes produit une trace d'erreur ou, pire, une page de `NaN`.

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

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

Gérez le singulier. « 1 communes » dans onze rapports est le signe d'un modèle que personne n'a relu.

Gérez le petit dénominateur. Un district avec 40 dépistages obtient un intervalle trois fois plus large qu'un district avec 400, et le modèle doit le dire plutôt que d'imprimer les deux à une décimale comme s'ils étaient comparables.

5.5 Ce qui va dans le modèle et ce qui va dans la chaîne

Dans le modèle	Dans la chaîne
Mise en page, prose, forme de l'argumentation	La lecture des données brutes
Un tableau de synthèse calculé depuis des données préparées	Nettoyage et normalisation
Les figures, tracées depuis des données préparées	Toute transformation de plus d'une ligne ou deux
Réserves, seuils et définitions	Tout ce que deux rapports partageraient

Le modèle devrait lire un fichier préparé, non l'export brut. Si douze rendus refont chacun le nettoyage, le nettoyage tourne douze fois, prend douze fois plus de temps, et — pire — peut être édité dans le modèle pour un seul district.

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

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

5.6 Le paramètre qu'il faut toujours avoir

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

Un paramètre de version de données fait dire au rapport quel fichier il a lu, ce qui est la question de provenance qu'un lecteur pose six mois plus tard. Imprimez-le dans le colophon.

Et un paramètre `as_of` plutôt que l'horloge, pour la raison qu'a donnée la leçon 4 : un rapport qui dit « produit le 14 mars » change à chaque reconstruction, et un rapport qui dit « données au 31 mars » non.

5.7 Là où cette plateforme fait la même chose

Chaque leçon de ce site produit un diaporama en deux langues et trois formats — un PDF Beamer, un fichier PowerPoint et la source LaTeX, six fichiers par leçon — et aucun n'a été composé à la main.

`slide-plan.mjs` planifie le diaporama une fois depuis le corps de la leçon, et le PDF Beamer, le fichier PowerPoint et le lecteur de diapositives dans le navigateur sont trois rendus de ce plan unique.

Le raisonnement est celui des douze districts, à une autre échelle. Un tableau `slides:` dans le frontmatter produirait de meilleures diapositives en principe et de pires en pratique : il signifie que chaque nouvelle leçon exige une seconde passe de rédaction en deux langues, et c'est cette passe qui saute.

La conséquence pour les auteurs est réelle et vaut d'être nommée. Parce que le diaporama est dérivé, les leçons sont écrites avec des listes, des amorces en gras, des tableaux et des encadrés — parce que c'est de cela qu'un diaporama est fait. **Un système paramétré**

façonne ses entrées, et prétendre le contraire produit des modèles qui se battent contre leur propre contenu.

5.8 Rapportez-le en entier

```
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.
```

La dernière ligne est ce qui transforme douze fichiers rendus en douze rapports qu'on peut envoyer. Une boucle qui avale les échecs produit onze rapports et une question que personne ne pose jusqu'à ce que la mauvaise personne le remarque.

5.9 La suite

Une chaîne qui rend douze rapports depuis des données amont cassées rend douze rapports cassés. La leçon suivante la fait s'arrêter à la place — bruyamment, au point de défaillance, plutôt que de produire un nombre plausible que personne ne peut vérifier.

CHAPITRE 6

Une chaîne qui échoue vaut mieux qu'une chaîne qui devine

Leçon 6 sur 8 · 135 min

L'export amont gagne une colonne, en perd une, change un code de « yes » à « Y », ou arrive avec la moitié des lignes. Une chaîne qui continue produit un nombre faux et plausible. Sept vérifications l'arrêtent, et chacune a une défaillance réelle derrière elle.

6.1 La défaillance qui n'a pas de message d'erreur

L'export mensuel arrive. Une colonne qui disait `true` dit maintenant `Y`. Votre conversion booléenne transforme chacune d'elles en valeur manquante, le dénominateur perd 30 %, et la chaîne s'exécute jusqu'au bout et produit un rapport.

Rien n'a échoué. Le graphique est tracé, le pourcentage est plausible, le fichier est daté d'aujourd'hui, et le nombre est faux.

C'est le mode de défaillance pour lequel cette leçon existe, et il est bien plus courant qu'un plantage. Un plantage se corrige le matin même.

6.2 Sept vérifications, chacune avec une défaillance réelle derrière

Chacune correspond à un défaut documenté dans les jeux de données de cette plateforme.

Une : les colonnes attendues sont présentes.

```python
REQUIRED = {"household_id", "district", "water_source", "round_trip_minutes"}
missing = REQUIRED - set(survey.columns)
if missing:
    raise ValueError(f"Export is missing columns: {sorted(missing)}")
```

```r
stopifnot(all(required %in% names(survey)))
```

Deux : le nombre de lignes est dans la plage attendue.

```python
if not 2_000 <= len(survey) <= 3_000:
    raise ValueError(f"Expected 2,000-3,000 households, got {len(survey):,}")
```

La moitié d'un export est la défaillance silencieuse la plus courante. Un téléchargement tronqué, un filtre resté actif, une plage de dates décalée d'un mois — tous produisent un fichier valide avec trop peu de lignes.

Trois : les codes sont les codes que vous connaissez.

```python
KNOWN = {"piped-into-dwelling", "piped-into-yard", "public-tap", "borehole",
         "protected-well", "protected-spring", "unprotected-well",
         "unprotected-spring", "surface-water", "tanker-truck", "rainwater"}
unknown = set(survey["water_source"].dropna()) - KNOWN
if unknown:
    raise ValueError(f"Unknown water_source values: {sorted(unknown)}")
```

```
setdiff(unique(survey$water_source), known)
```

Un nouveau code est une décision, non une donnée. Quelqu'un a ajouté une option au formulaire et l'analyse doit décider où elle va — la faire tomber discrètement dans « autre » revient à laisser un `.fillna()` prendre la décision.

Quatre : l'identifiant est unique là où il doit l'être.

```
duplicates = survey["household_id"].duplicated().sum()
if duplicates:
    raise ValueError(f"{duplicates} duplicate household_id values")
```

Le registre scolaire de cette plateforme a exactement ce défaut — deux élèves y figurent deux fois après un transfert jamais radié — et une jointure nue duplique leurs lignes. Une vérification l'aurait attrapé à la jointure plutôt que dans un coefficient.

Cinq : les valeurs manquantes sont là où vous les attendez.

```
completeness = survey.notna().mean()
if completeness["district"] < 0.99:
    raise ValueError(f"district is {completeness['district']:.1%} complete")
```

Six : les nombres sont dans une plage possible.

```
implausible = survey[(survey["litres_per_person_day"] < 0)
                     | (survey["litres_per_person_day"] > 200)]
if len(implausible) > 20:
    raise ValueError(f"{len(implausible)} implausible litres values")
```

Notez le seuil plutôt qu'une tolérance nulle. Onze ménages avec une erreur d'unité est un défaut documenté que cette analyse traite ; deux cents est un problème nouveau.

Sept : la sortie est celle que vous avez déclarée.

```
assert summary["n"].sum() == len(survey), "rows lost between input and summary"
```

Un nombre de lignes qui change au travers d'une jointure est l'assertion la plus utile de l'analyse de programme, parce qu'une jointure interne qui écarte un tiers des données ressemble exactement à une jointure interne qui n'en écarte aucune.

6.3 Échouez au point de défaillance, non à la fin

```
def load_survey(path: pathlib.Path) -> pd.DataFrame:
    survey = pd.read_csv(path)
    check_columns(survey)
    check_rows(survey)
    check_codes(survey)
    return survey                # nothing downstream runs on a bad file
```

```
load_survey <- function(path) {
  survey <- readr::read_csv(path)
  check_columns(survey); check_rows(survey); check_codes(survey)
  survey
}
```

Vérifiez à la frontière — là où les données entrent, et après chaque jointure.

Une vérification en fin de chaîne vous dit que quelque chose ne va pas ; une vérification à la frontière vous dit quoi.

Levez une erreur, n'avertissez pas. Un avertissement dans un journal que personne ne lit équivaut à aucune vérification, et le journal n'est pas lu précisément les jours chargés où l'export casse.

6.4 Ce que fait cette plateforme

Sept vérifications font échouer `astro build`, délibérément, et elles ont la même forme.

Vérification	Attrape
Schéma Zod	Un champ manquant ou du mauvais type
Références inter-collections	Un chemin pointant vers un cours qui n'existe pas
Fichiers hors d'un répertoire de langue	Une entrée sans langue
Parité de traduction	Une entrée publiée dans une seule langue
Cohérence sujet-secteur	Un sujet étiqueté hors de son secteur
Synthétique uniquement	Un jeu de données non déclaré synthétique
Ossature du programme	Un cours publié absent de la carte du cursus

Quatre autres s'exécutent dans `pnpm test` plutôt que dans la compilation, parce qu'elles ont besoin de `node:fs` et que la compilation pré-rend dans un worker Cloudflare qui n'en a pas. C'est une contrainte réelle qui a décidé où vivent les vérifications, et elle vaut d'être nommée : **placez la vérification là où elle peut s'exécuter, non là où c'est le plus élégant.**

L'une d'elles vérifie une relation que le graphe de références ne peut structurellement pas atteindre. Les règles de références vérifient qu'un slug déclaré *se résout* ; elles ne peuvent pas vérifier qu'une entrée est *pointée*. Un cours pourrait donc être livré sans pratique attachée et chaque barrière resterait verte — d'où l'existence de `course-practice.test.ts`, qui compte à rebours depuis chaque cours publié vers le laboratoire et l'exercice qui doivent le nommer.

Toute règle du type « chaque X a au moins un Y » exige un test de cette forme. Un graphe de références est le mauvais outil pour cela.

6.5 La règle à retenir

Tout champ nommant un fichier livré exige un drapeau `ready` et un test de système de fichiers à côté. Cette plateforme l'a appris trois fois : vingt exemples travaillés déclarés et jamais écrits, trente-cinq livrables de projet déclarés et jamais écrits, et un ensemble de chemins de figures qui ne pointaient vers rien.

Un chemin dans un frontmatter est une chaîne de caractères, et rien dans une compilation ne peut dire s'il pointe vers quelque chose. Le schéma met donc `ready` à false par défaut et un test vérifie que le fichier existe.

6.6 Rapportez-le en entier

```
Pipeline checks

  The loader validates every raw export before anything downstream runs:
  required columns present, 2,000–3,000 rows, water_source values within the
  known set, household_id unique, district at least 99% complete.
```

```
Implausible litres-per-person values are tolerated up to 20 rows, which is
the documented unit-entry defect; above that the run stops.

Row counts are asserted across every join. A join that changes the row
count stops the pipeline rather than producing a summary.

All checks raise rather than warn. The March run stopped on an unknown
water_source value ("piped-shared"), which turned out to be a new form
option added upstream; it is now mapped explicitly in src/clean.py.
```

La dernière phrase est ce qui rend la section crédible. Une section de vérifications qui n'a jamais rien attrapé est une section de vérifications que personne n'a testée.

6.7 La suite

Tout ce qui précède suppose que vous êtes encore là. La leçon suivante porte sur le document qui doit fonctionner quand vous n'y êtes plus — écrit pour quelqu'un qui a votre poste et rien de votre contexte.

Quatrième partie

Quelqu'un d'autre l'exécute

CHAPITRE 7

Écrit pour quelqu'un qui a votre poste et rien de votre contexte

Leçon 7 sur 8 · 135 min

Une passation n'est pas une description du code. C'est la liste de ce que vous savez et qui ne figure nulle part dans le dépôt — pourquoi cette colonne est écartée, à qui écrire quand l'export s'arrête, et quel nombre du rapport sera contesté.

7.1 À quoi sert une passation

Pas à dire « voici le code » — le code est dans le dépôt et un successeur compétent sait le lire. **Une passation est la part qui n'est pas dans le dépôt**, et c'est presque entièrement une liste de décisions et de relations.

Le test : **votre successeur reçoit l'export mensuel, il paraît faux, et il doit décider quoi faire.** Tout ce dont il a besoin pour cela est ce que contient le document.

7.2 Les sept sections

```
# Handover: nutrition surveillance analysis
Written 2026-03-14 by [name], for whoever holds this next.

## 1. What this produces, and who reads it
A quarterly GAM estimate by commune, in the cluster report and the district
health office's monthly pack. The cluster coordinator reads the commune table
first and asks about anything above 10%.

## 2. Run it
    uv sync && uv run python run.py
Takes about four minutes. Outputs land in outputs/ and are safe to delete.

## 3. Where the data comes from, and who to ask
data/raw/muac-screening-YYYY-MM.csv -- exported from the screening database on
the 5th of each month by [name, role, email]. If it does not arrive by the
10th, that is who to chase; it has been late three times in two years and the
cause was a permissions change each time.

## 4. Decisions that are not obvious from the code
- Commune names arrive spelled four ways. Normalisation is in src/clean.py.
  Ungrouped, the worst commune splits into four fragments and none looks
  alarming, which is how this went unnoticed for two rounds.
- Children with no age recorded (about 2%) are excluded from the age-
  disaggregated table but kept in the total. The cluster agreed this in
  March 2025; the alternative was excluding them everywhere, which moved the
  headline by 0.4 points.
- MUAC is the case definition, not weight-for-height. This is a screening
  dataset and there are no heights.

## 5. What breaks, and what it looks like
- A new water_source code stops the pipeline with "Unknown water_source
  values". It is usually a form change; map it in src/clean.py and tell
```

```
  [name].
- A row count outside 2,000-3,000 stops it. Usually a truncated download.
- If the report renders but a commune is missing, screening did not happen
  there; that is real and should be stated in the narrative.

## 6. What gets challenged
The commune ranking. Eleven of eleven adjacent communes have overlapping
intervals, so the chart supports "these three are above the district median"
and does not support ranking one commune against its neighbour. The caption
says so; keep it there, because it comes up every quarter.

## 7. What I would do next
The screening register has no follow-up measurement, so the MUAC 125mm
eligibility threshold cannot be evaluated as a discontinuity design. Adding
an eight-week re-measurement on a sample of children just above the cut-off
would make it estimable. I raised this in January; it is not funded.
```

Les sections 4 et 6 sont celles qu'on ne peut pas reconstituer. Tout le reste, un successeur le retrouverait en une semaine.

7.3 Pourquoi la section 4 est tout le document

Chaque exclusion, recodage et seuil de la chaîne était une décision, et le code enregistre ce qui a été décidé en perdant le pourquoi. Un successeur qui lit `df = df[df.age <= 20]` sait ce que cela fait et ignore s'il a le droit de le changer.

```
# Excluded: 14 children with recorded ages above 20 years in an under-5
# screening. These are data-entry errors (the age field accepts free text).
# Agreed with the screening lead 2025-03; they are excluded everywhere rather
# than corrected, because we cannot tell 2 from 20 from 12.
survey = survey[survey["age_months"] <= 60]
```

```
# The comment is the handover. Write it when you make the decision.
```

Écrivez le commentaire au moment de la décision, non au moment de la passation. Au moment de la passation, vous vous souviendrez des décisions et non des raisons, ce qui est exactement l'inverse de ce qu'il faut.

7.4 La section 6, et pourquoi elle n'est pas défensive

Nommez les nombres qui sont contestés et quelle est la réponse. Toute analyse récurrente en a deux ou trois, et un successeur qui les ignore capitulera devant une contestation déjà tranchée ou rouvrira une discussion déjà close.

C'est de la mémoire institutionnelle, et c'est la première chose perdue dans une passation. C'est aussi la moins coûteuse à écrire : trois puces.

7.5 La passation n'est pas un document, c'est une période

Le document est nécessaire et il ne suffit pas.

Faites-leur l'exécuter pendant que vous êtes encore là. Non pas vous regarder l'exécuter — l'exécuter eux-mêmes, depuis un clone neuf, sur leur propre machine, avec vous disponible. Tout ce qui échoue est une lacune du dépôt ou du document, et c'est le seul moyen de trouver ces lacunes.

Faites-leur produire une vraie sortie. Le prochain rapport mensuel, non un essai. Un successeur qui a produit un rapport a fait le travail une fois.

Laissez le canal de questions ouvert pendant un cycle. Un trimestre, un mois, selon la cadence. L'essentiel de ce qu'ils auront besoin de demander arrive au moment où cela casse pour la première fois.

7.6 Ce qu'il faut écrire au fil de l'eau

Le document de passation est bien plus facile s'il n'est pas écrit au moment de la passation.

Écrivez-le	Quand
Le commentaire de décision	Au moment de la décision
L'entrée « ce qui casse »	La première fois que cela casse
L'entrée « ce qui est contesté »	La première fois que c'est contesté
Le contact et la cadence	Au moment où vous les mettez en place
Le « ce que je ferais ensuite »	Dès que vous y pensez

Gardez-le dans le dépôt, dans `HANDOVER.md`, en gestion de versions. Une passation dans la boîte mail de quelqu'un est une passation à une seule personne.

7.7 Le README et la passation sont deux documents différents

README	Passation
Pour quiconque ouvre le dépôt	Pour la personne qui le reprend
Ce qu'il produit, comment l'exécuter	Pourquoi il est ainsi
Reste court	Peut être long
Mis à jour avec le code	Mis à jour quand une décision est prise

Un README devenu une passation a cessé d'être lu, ce qui fait perdre les deux.

7.8 Rapportez-le en entier

```
Continuity

  HANDOVER.md documents the analysis decisions, the upstream contacts and
  cadence, the known failure modes, and the findings that are routinely
  challenged. It is version-controlled and was last reviewed 2026-03-14.

  Every exclusion and recode in src/ carries a comment giving the reason and
  the date it was agreed.

  The incoming analyst ran the March cycle end-to-end from a fresh clone,
  with the outgoing analyst available. Two gaps were found and closed: the
  screening database export permission, and an undocumented commune-name
  variant.
```

Le dernier paragraphe est la preuve que la passation a eu lieu. Un document de passation que personne n'a exécuté est un document jamais testé, et les deux lacunes qu'il a trouvées sont ce pour quoi il existe.

7.9 La suite

La dernière leçon retourne le cours sur cette plateforme. Chaque affirmation qu'il a faite — données brutes en lecture seule, environnement figé, sortie déterministe, vérifications qui échouent bruyamment, un modèle pour plusieurs sorties — se vérifie dans ce dépôt, et la leçon est une visite guidée de l'endroit où.

CHAPITRE 8

L'étude de cas est ce que vous êtes en train de lire

Leçon 8 sur 8 · 135 min

Vingt jeux de données issus de dix-sept scripts à graine fixe qui se régénèrent à l'octet près, vingt fichiers de tests et 4 212 assertions, sept vérifications qui font échouer la compilation, et un schéma qui refuse les données non synthétiques. Chaque affirmation de ce cours se vérifie dans ce dépôt.

8.1 Pourquoi la plateforme est l'exemple

Tous les autres cours d'ici emploient un jeu de données. Celui-ci emploie le dépôt, pour une raison qui vaut d'être énoncée : **une chaîne reproductible est une affirmation sur un artefact, et la seule démonstration convaincante est un artefact que vous pouvez vérifier.**

Tout ce qui suit se vérifie depuis un clone.

8.2 Des données brutes produites, non collectées

```
pnpm datasets:generate
git status --short apps/data-analysis/public/datasets/files
# (nothing)
```

Vingt fichiers CSV, dix-sept scripts générateurs, et les régénérer tous ne change rien. Chaque générateur a une graine fixe — `SEED = 20240622`, dans le fichier — si bien que la sortie est une fonction du seul code.

C'est une propriété plus forte que « la chaîne se réexécute ». Cela signifie que réexécuter est une étape de *vérification* : si `git status` n'est pas vide après une régénération, ou bien le générateur a changé, ou bien le fichier commité a été édité à la main, et vous voulez savoir l'un comme l'autre.

Et les défauts sont délibérés. Chaque entrée de la liste `knownIssues` d'un jeu de données correspond à un bloc de son générateur — les quatre orthographes de Nord-Ouest, les onze erreurs d'unité, les deux élèves en double. **Changez l'un et vous changez l'autre**, ce qui est la même discipline que celle des figures produites depuis les données.

8.3 Une sortie commitée, pour une raison qui nomme sa condition

Le dépôt commite des choses qu'il pourrait reconstruire : CSV de données, PDF de cours, diaporamas, figures, carnets.

Parce que le déploiement tourne sur Cloudflare Pages sans TeX, sans Python et sans pandoc. La compilation copie des fichiers statiques, si bien que LaTeX n'est jamais sur le chemin critique du déploiement et qu'un contributeur sans la chaîne d'outils peut quand même livrer un changement de contenu.

C'est la règle générale de la leçon 1, appliquée. Commitez un artefact produit quand le reconstruire n'est pas à la portée de tous ceux qui en ont besoin. **Non parce que reconstruire est lent** — parce que reconstruire est indisponible.

Et cela fonctionne parce que la sortie est déterministe. `SOURCE_DATE_EPOCH` fixe l'horodatage de pdfTeX et celui de pandoc, si bien qu'un `.pptx` — une archive dont les dates d'entrée bougeraient sinon — se reconstruit à l'octet près. Sans cela, commiter la sortie produite donnerait un diff à chaque compilation et personne n'en relirait aucun.

8.4 Des vérifications à trois endroits, et la contrainte qui a décidé où

Où	Quoi	Pourquoi là
Schéma Zod	Types de champs, énumérations, champs requis	S'exécute sur une entrée, sans système de fichiers
`astro build`	Sept vérifications de niveau graphe	A besoin de toute la collection
`pnpm test`	Vérifications de système de fichiers	La compilation n'a pas `node:fs`

La compilation Astro pré-rend dans un worker Cloudflare sans `node:fs`, si bien qu'une vérification qui doit regarder un fichier sur le disque ne peut pas y vivre. `dataset-files.test.ts` vérifie que les lignes et octets déclarés correspondent aux fichiers ; `course-pdfs.test.ts` vérifie que chaque cours publié livre ses deux éditions PDF.

Placez la vérification là où elle peut s'exécuter. C'est une contrainte d'ingénierie réelle, et elle a produit un meilleur agencement que l'élégance ne l'aurait fait : les vérifications rapides sont dans le schéma, celles de graphe dans la compilation, et celles de système de fichiers dans une suite de tests qui passe en premier en intégration continue.

8.5 La vérification qu'un graphe de références ne peut pas exprimer

```
REFERENCE_RULES verifies that a declared slug resolves.
It cannot verify that an entry is pointed at.
```

Un cours pourrait donc être livré sans laboratoire ni exercice, et chaque barrière resterait verte. C'est pour cela qu'existe `course-practice.test.ts` — il compte à rebours depuis chaque cours publié vers la pratique qui doit le nommer.

Toute règle du type « chaque X a au moins un Y » exige un test de cette forme, et reconnaître lesquelles de vos contraintes sont de ce genre constitue l'essentiel de la conception d'une suite de vérifications.

8.6 Ce que le dépôt refuse de contenir

```
.refine((d) => d.dataQuality.synthetic === true, {
  message: "Only synthetic or fully de-identified datasets may be published.",
})
```

Un jeu de données non déclaré synthétique fait échouer la compilation. Ce public modélise des données de bénéficiaires, de protection, de VBG et cliniques, et « aucun enregistrement ne remonte à une personne réelle » devient une propriété du dépôt plutôt qu'une promesse dans une note.

Cela va de pair avec produire les données plutôt que les collecter. La garantie est structurelle aux deux bouts : le générateur ne peut pas produire une personne réelle, et le schéma n'acceptera pas une affirmation contraire.

8.7 La reproductibilité au niveau du langage

```
{ "packageManager": "pnpm@11.9.0", "engines": { "node": ">=22" } }
```

Le gestionnaire de paquets est figé, non seulement les paquets. Un pnpm différent résout le fichier de verrouillage différemment, et c'est la couche en dessous de celle que la plupart des projets figent.

TypeScript est maintenu en 6.x avec la raison écrite : TypeScript 7 n'expose pas encore l'API programmatique dont dépend `astro check`, si bien que `pnpm typecheck` échoue purement et simplement en 7. **Le verrou nomme la condition de sa levée**, ce qui sépare une contrainte réfléchie d'une contrainte héritée.

8.8 Ce que ce dépôt ne fait pas

Être honnête sur les lacunes fait partie de l'étude de cas.

Pas de conteneur. Un fichier de verrouillage plus un gestionnaire de paquets figé reproduit le côté JavaScript ; les côtés Python et TeX dépendent de ce que le contributeur possède. `pnpm figures:build` n'a besoin que d'un Python d'origine, et `pnpm course:pdf` avertit au lieu d'échouer quand TeX est absent — un arbitrage délibéré pour qu'un contributeur sans chaîne d'outils puisse quand même livrer du contenu.

Pas de test de déterminisme de bout en bout en intégration continue. La régénération des jeux de données est déterministe et vérifiée à la main ; il n'existe pas de tâche d'intégration continue qui régénère tout et échoue sur un diff. Ce serait la prochaine chose à ajouter.

La documentation de passation est mince. `CLAUDE.md` et `docs/DEFINITION_OF_DONE.md` portent les décisions, ce qui est plus que la plupart des dépôts et moins que ce qu'exige la leçon 7.

Nommer les lacunes est l'essentiel. Une étude de cas qui revendique tout est une étude de cas dont personne ne peut apprendre, et les trois ci-dessus sont l'état honnête d'un dépôt par ailleurs inhabituellement strict avec lui-même.

8.9 Rapportez-le en entier

```
Reproducibility of this platform

  Data          20 CSV files generated by 17 seeded scripts. Regenerating all
                of them produces byte-identical output; `pnpm datasets:generate`
                followed by `git status` is the verification.

  Checks        7 fail the build (schema, references, locale, translation
                parity, topic-sector, synthetic-only, programme spine).
                20 test files and 4,212 assertions run first in CI.

  Output        Course PDFs, slide decks, figures and notebooks are committed
                and deterministic. SOURCE_DATE_EPOCH pins pdfTeX and pandoc.

  Environment   pnpm 11.9.0 and Node 22 pinned. TypeScript held at 6.x with
                the reason and the removal condition documented.

  Refused       A dataset not declared synthetic fails the build.

  Not done      No container; no CI job that regenerates everything and fails
```

```
                on a diff; handover documentation thinner than this course
                asks for.
```

La dernière ligne est celle à copier dans votre propre version. Une déclaration de reproductibilité sans section « pas fait » est une déclaration que personne n'a auditée.

8.10 La suite

C'est le cours, et c'est le dernier de l'ossature du programme.

Le fil qui traverse les vingt est le même. Un nombre dans un rapport de programme est une affirmation, et tout ce que cette plateforme enseigne porte sur ce qui doit être vrai pour que l'affirmation tienne — un dénominateur défendable, un intervalle qui dit votre degré de certitude, un groupe de comparaison qui en est un, un graphique qui n'en dit pas plus que les données, et une chaîne qui produit la même réponse quand quelqu'un d'autre l'exécute.

La dernière est celle qui rend les autres vérifiables. Une analyse que personne ne peut réexécuter est une analyse dont les autres propriétés doivent être prises sur parole, et ce public est jugé sur sa capacité à défendre un nombre en revue plutôt qu'à le produire.

À propos de cette édition

Ce PDF est produit à partir de la même source que la version web de la formation ; les deux ne peuvent donc pas diverger. L'édition en ligne comporte le code exécutable, les jeux de données téléchargeables et les corrections publiées depuis la production de ce fichier.

> Tous les jeux de données cités dans cette formation sont synthétiques ou entièrement anonymisés. Aucun enregistrement ne renvoie à une personne réelle.

Cette formation est publiée en anglais et en français. L'édition française est une adaptation professionnelle reprenant la terminologie employée par l'UNICEF, l'OMS, le PAM, OCHA et les ministères nationaux — et non une traduction littérale.

Consulter et exécuter la formation en ligne sur data-analysis.cassion.dev/fr/cours/reproducible-workflows

Généré le 29 juillet 2026.