CASSION · FORMATION

Fondamentaux de l'analyse de données pour le suivi-évaluation

Charger, nettoyer et synthétiser les données de routine d'un programme, en Python comme en R, puis les convertir en tableaux d'indicateurs exploitables dans un rapport bailleur.

Formateur	Pierre Robentz Cassion
Niveau	Débutant
Heures apprenant	16 h
Enseignée en	Python · R
Mise à jour	26 juillet 2026
Secteurs	Santé publique · Nutrition · Réponse d'urgence
Normes et méthodologies	Cadre logique · Gestion axée sur les résultats (GAR) · Normes OMS de croissance de l'enfant

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

Ce que vous saurez faire

— Importer les exports CommCare, KoboToolbox et DHIS2 dans un tableau ordonné sans corrompre les identifiants, les dates ni les codes de valeur manquante

— Diagnostiquer et documenter les problèmes de qualité que portent toujours les données de routine, et chiffrer ce que chacun coûte au résultat final

— Trancher le traitement de chaque défaut et consigner cette décision dans un journal de nettoyage qu'un auditeur peut suivre

— Produire un tableau d'indicateurs désagrégé conforme à une définition de cadre logique, numérateur et dénominateur explicitement énoncés

— Reproduire la même analyse en Python et en R, et la relancer sur l'export du trimestre suivant sans la modifier

Place de cette formation dans le programme

Fondamentaux — Passer d'un export brut de programme à un tableau exploitable, dans celui des deux langages — Python ou R — que votre équipe utilise déjà.

Prérequis

Aucun. Cette formation ne suppose aucune expérience préalable de la programmation.

Sommaire

À propos de cette formation

Les données de routine arrivent rarement propres. Cette formation part des exports dont vous disposez déjà — une extraction de formulaires CommCare, un CSV KoboToolbox, un tableau croisé DHIS2 — et va jusqu'au tableau d'indicateurs que vous pourrez défendre en revue avec un bailleur.

Chaque technique est enseignée deux fois, une fois en Python et une fois en R : les équipes de suivi-évaluation se partagent entre les deux, et vous hériterez de celui qu'utilisait la personne en poste avant vous.

Presque tous les exemples s'appuient sur un même jeu de données : un registre synthétique de dépistage du périmètre brachial (PB) couvrant douze communes de l'Artibonite, en Haïti, et portant les valeurs manquantes, erreurs d'unité, doubles enregistrements et codes de décision contradictoires que porte un vrai registre de campagne. Travailler le même fichier de l'export brut à l'indicateur défendu est précisément l'objectif : vous voyez ce que coûte chaque décision, parce que vous voyez le chiffre bouger.

La formation ne suppose aucune expérience préalable de la programmation. Elle suppose en revanche que vous avez déjà eu à répondre d'un chiffre devant quelqu'un qui ne voulait pas l'entendre.

CHAPITRE 1

D'où viennent les données de programme

Leçon 1 sur 8 · 45 min

La structure des exports CommCare, KoboToolbox et DHIS2, la manière dont chacun vous résiste différemment, et les questions à poser avant d'écrire la moindre ligne de code.

1.1 Les trois exports que vous rencontrerez

La plupart des chargés de suivi-évaluation héritent de données issues de l'un de ces trois systèmes. Chacun a ses habitudes, et les connaître fait gagner une demi-journée.

CommCare exporte une ligne par soumission de formulaire, les groupes répétés étant aplatis en colonnes numérotées (`child_1_muac`, `child_2_muac`, et ainsi de suite jusqu'à la largeur du formulaire le plus fourni du fichier). Défaire cet aplatissement est la première opération à mener. Cela signifie aussi que le nombre de colonnes change d'un export à l'autre : un mois où un ménage comptait neuf enfants produit neuf jeux de colonnes, et le fichier du mois suivant en comporte sept.

KoboToolbox exporte une structure proche du XLSForm, les répétitions se trouvant dans des feuilles distinctes reliées par `_index` et `_parent_index`. C'est plus honnête que la structure CommCare, mais cela veut dire qu'une analyse qui ignore la seconde feuille analyse en réalité des ménages et non des personnes.

DHIS2 exporte des données agrégées — déjà sommées par période et par unité d'organisation. La question n'est donc que rarement « comment nettoyer ceci », mais presque toujours « qu'a-t-on exactement compté ». On ne récupère pas un individu à partir d'un tableau croisé DHIS2, et on ne recalcule pas un dénominateur figé en amont de soi.

1.2 L'unité d'observation décide de tout

Avant toute chose, répondez à une question : **qu'est-ce qu'une ligne ?**

Export	Une ligne représente	Le piège
Extraction CommCare	Une soumission de formulaire	Un formulaire peut couvrir plusieur
Feuille principale Kobo	Un entretien	Les membres sont dans la feuille de
Feuille de répétition Kobo	Un membre	Aucun contexte ménage avant la jo
Tableau croisé DHIS2	Une unité d'organisation, une période, un élément	Les individus ont définitivement dis

Se tromper ici produit une analyse cohérente avec elle-même et entièrement fausse. Un registre de dépistage où une ligne est une *soumission* et non un *enfant* rapportera une charge de cas trop faible d'exactement le nombre de ménages à plusieurs enfants, et rien dans le résultat n'aura l'air anormal.

> Posez la question à voix haute avant d'ouvrir le fichier, et inscrivez la réponse en commentaire en tête de votre script. La plupart des désaccords abordés plus loin dans cette formation viennent de deux personnes qui n'y répondaient pas de la même manière.

1.3 À quoi ressemble un vrai export

Cette formation s'appuie sur un registre synthétique de dépistage du PB dans l'Artibonite, en Haïti. Une ligne correspond à un enfant dépisté. Voici sa structure :

```
child_id,commune,screening_date,age_months,sex,muac_mm,oedema,outcome
CH00854,Terre-Neuve,2024-01-15,22,m,133,false,no-action
CH01439,Verrettes,2024-01-15,16,f,143,false,no-action
```

Huit colonnes, 4 218 lignes, une année de dépistage de masse communautaire sur douze communes. Le fichier est assez petit pour être lu et assez imparfait pour enseigner : il porte des valeurs manquantes codées `-99`, des mesures restées en centimètres, des doubles enregistrements et des décisions de référencement qui contredisent la mesure inscrite juste à côté.

Rien de tout cela n'est décoratif. Chacun de ces défauts se rencontre dans de vrais registres de campagne, et il a été introduit délibérément pour que vous le croisiez ici plutôt que la semaine précédant une échéance de rapport.

1.4 Le périmètre brachial, en un paragraphe

Le périmètre brachial (PB) est une mesure de dépistage prise avec un ruban à code couleur autour du bras de l'enfant. On l'utilise parce qu'il ne demande ni balance, ni toise, ni plus d'une quarantaine de minutes de formation. Pour les enfants de 6 à 59 mois, les seuils de l'OMS sont les suivants :

— En dessous de 115 mm — malnutrition aiguë sévère (MAS)

— De 115 mm à moins de 125 mm — malnutrition aiguë modérée (MAM)

— 125 mm et au-delà — pas de malnutrition aiguë selon cette mesure

Les œdèmes bilatéraux prenant le godet signent une MAS quelle que soit la mesure. C'est la raison d'être de la colonne `oedema`, et la raison pour laquelle une analyse filtrant sur le seul `muac_mm` sous-estime la charge de cas.

La malnutrition aiguë globale (MAG) est la somme de la MAS et de la MAM. Vous la calculerez en leçon 7, et le débat sur son dénominateur constitue la leçon 8.

1.5 Les trois questions à poser à la personne qui vous envoie le fichier

Les données de routine arrivent bien plus souvent sans dictionnaire des variables qu'avec. Ces trois questions récupèrent l'essentiel de ce qu'un dictionnaire vous aurait appris.

1. **Que signifie une case vide ici ?** Non mesuré, mesuré à zéro, refus, ou la tablette a planté ? Ce sont quatre choses différentes, fréquemment stockées toutes les quatre sous la forme d'une cellule vide.
2. **Quelque chose a-t-il déjà été nettoyé ou supprimé ?** Si quelqu'un a retiré les lignes qu'il jugeait erronées, votre nombre de lignes n'est pas la charge de cas et votre taux de complétude est une fiction.
3. **Quelle période couvre ce fichier, et selon quelle date ?** Date de dépistage, date de soumission et date de synchronisation peuvent différer de plusieurs semaines sur une mauvaise connexion. Un rapport mensuel construit sur la mauvaise date déplace des cas d'un mois à l'autre.

Vous n'obtiendrez pas toujours de réponse. Consigner que vous avez posé la question, et ce que vous avez supposé faute de réponse, est ce qui distingue une analyse défendable d'une analyse qui ne l'est pas.

1.6 La suite

La leçon suivante met en place un environnement de travail en Python et en R — délibérément un environnement qui fonctionne hors ligne, car une grande partie de ce public travaille sur une connexion dont on ne peut pas attendre qu'elle installe un paquet au moment précis où il devient nécessaire.

CHAPITRE 2

Un environnement de travail en Python et en R

Leçon 2 sur 8 · 60 min

Installer Python et R de façon à ce qu'ils fonctionnent encore sur une mauvaise connexion, organiser un dossier de projet qui survit à une passation, et exécuter le même premier script dans les deux langages.

2.1 Pourquoi cette leçon n'est pas facultative

La raison la plus fréquente pour laquelle une analyse de suivi-évaluation cesse d'être reproductible n'est pas une erreur statistique. C'est que la personne qui l'a écrite a installé un paquet un après-midi, n'a jamais noté lequel, puis est partie. Six mois plus tard, le script échoue à l'import sur un portable de bureau terrain relié par satellite, et le rapport trimestriel redevient un travail manuel sur tableur.

Dix minutes de mise en place aujourd'hui, c'est ce qui évite cela.

2.2 Python : utilisez uv, pas l'interpréteur du système

Votre système d'exploitation est livré avec un Python. N'analysez pas de données avec celui-là : il est là pour faire fonctionner le système, et le modifier peut casser des choses dont vous ignoriez qu'elles en dépendaient.

`uv` installe un Python isolé et résout les paquets assez vite pour rester utilisable sur une connexion lente. Il écrit aussi un fichier de verrouillage, ce qui rend l'environnement reproductible sur la machine de quelqu'un d'autre.

```
# Installer uv (macOS et Linux)
curl -LsSf https://astral.sh/uv/install.sh | sh

# Créer un projet et fixer un interpréteur
uv init muac-analysis
cd muac-analysis
uv python install 3.13

# Ajouter ce que cette formation utilise
uv add pandas pyarrow matplotlib
```

`uv add` écrit à la fois `pyproject.toml` (ce que vous avez demandé) et `uv.lock` (ce qui a exactement été installé, empreinte comprise). Versionnez les deux. Un collègue reproduit alors votre environnement en une commande :

```
uv sync
```

Derrière un proxy ou hors ligne, `uv` sait installer depuis un dossier local de paquets. Télécharger ces paquets une fois, sur une clé USB, et installer depuis celle-ci est un mode de travail courant dans le secteur : il vaut la peine de le préparer avant d'en avoir besoin.

2.3 R : utilisez les Projets et renv

Le problème équivalent en R, c'est la bibliothèque globale — des paquets installés dans un emplacement partagé, utilisés sans le savoir par toutes vos analyses, et mis à jour silencieusement sous vos pieds.

```r
# Une fois par machine
install.packages("renv")

# Une fois par analyse, depuis un Projet RStudio
renv::init()

# Ajouter ce que cette formation utilise
install.packages(c("readr", "dplyr", "tidyr", "ggplot2", "janitor"))

# Consigner exactement ce qui est installé
renv::snapshot()
```

`renv.lock` est l'équivalent de `uv.lock`. Versionnez-le. Un collègue exécute `renv::restore()` et obtient votre bibliothèque, pas la sienne.

Deux habitudes comptent autant que l'outillage :

— **Travaillez dans un Projet.** Le répertoire de travail est alors la racine du projet, pour tout le monde, toujours. Des chemins comme `data/raw/muac.csv` fonctionnent sur n'importe quelle machine.

— **N'écrivez jamais `setwd()`.** Cela inscrit votre nom d'utilisateur dans l'analyse et garantit son échec chez la personne suivante.

2.4 Une organisation de projet qui survit à une passation

La même structure convient aux deux langages. Elle vaut la peine d'être adoptée telle quelle, car sa valeur tient à sa prévisibilité et non à son ingéniosité.

```
muac-analysis/
  data/
    raw/          # exactement tel qu'arrivé. Lecture seule. Jamais modifié.
    interim/      # résultats intermédiaires. Jetables.
    processed/    # prêt à l'analyse. Régénérable depuis raw + code.
  scripts/
    01-lecture.py
    02-nettoyage.py
    03-indicateurs.py
  output/
    figures/
    tables/
  README.md
```

Une règle porte l'essentiel : **`data/raw/` est en lecture seule.** Si vous ouvrez l'export dans Excel, corrigez trois cellules et enregistrez, vous avez détruit la seule trace de ce qui est réellement arrivé, et plus personne — vous compris — ne pourra jamais distinguer l'original de la correction.

Tout ce qui se trouve en aval de `data/raw/` doit pouvoir être reproduit en le supprimant puis en relançant les scripts. Si ce n'est pas le cas, une opération est faite à la main, et c'est précisément cette opération-là qui cassera.

> Un test utile : supprimez entièrement `data/processed/` et `output/`, relancez vos scripts, et vérifiez que vous retrouvez les mêmes résultats. Si vous ne vous résolvez pas à essayer, vous connaissez déjà la réponse.

2.5 Le même premier script, dans les deux langages

Lire le registre, examiner sa forme, afficher les premières lignes. Rien de plus — il s'agit seulement de confirmer que l'environnement fonctionne avant d'en dépendre.

```python
import pandas as pd

muac = pd.read_csv("data/raw/muac-screening-artibonite-2024.v1.csv")

print(muac.shape)
print(muac.head())
print(muac.dtypes)
```

```r
library(readr)
library(dplyr)

muac <- read_csv("data/raw/muac-screening-artibonite-2024.v1.csv")

dim(muac)
head(muac)
glimpse(muac)
```

Les deux doivent annoncer 4 218 lignes et 8 colonnes. Si ce n'est pas le cas, arrêtez-vous ici : un nombre de lignes en désaccord avec la source est l'erreur la moins coûteuse que vous détecterez jamais, et elle devient bien plus chère trois scripts plus loin.

2.6 Ce sur quoi les deux langages divergent

Vous utiliserez les deux ; autant savoir où ils diffèrent de manière conséquente.

Aspect	pandas	dplyr / readr
Valeur manquante	`NaN`, et `None` pour les objets	`NA`, typé par colonne
Lecture d'un CSV	`pd.read_csv` devine les types	`read_csv` devine et le signale
Enchaînement d'étapes	Chaînage de méthodes ou réaffectation	Le tube, `|>`
Synthèse par groupe	`groupby().agg()`	`group_by() |> summarise()`
Catégories	Type `category`, sur demande	Les facteurs, et ils mordent

Aucun n'est meilleur. Celui qui compte est celui que votre équipe utilise déjà, et si cette formation enseigne les deux, c'est parce que vous changerez d'équipe.

2.7 La suite

La leçon suivante lit l'export correctement — ce qui est plus difficile que `read_csv(chemin)`, car les valeurs par défaut abîmeront discrètement les identifiants, les dates et le code de valeur manquante avant que vous n'ayez regardé quoi que ce soit.

CHAPITRE 3

Lire un export sans le corrompre

Leçon 3 sur 8 · 75 min

Inférence de types, zéros initiaux, dates, et le code de valeur manquante qui devient un nombre. Les dégâts se produisent à l'import, avant même que vous ayez regardé quoi que ce soit.

3.1 Les dégâts précèdent le regard

`read_csv(chemin)` tient sur une ligne et prend une demi-douzaine de décisions à votre place. La plupart sont justes. Celles qui ne le sont pas le sont silencieusement, et le temps que vous vous en aperceviez, la valeur corrompue est passée dans trois synthèses et un graphique.

Cette leçon consiste à reprendre ces décisions.

3.2 Les codes de valeur manquante deviennent des nombres

Le registre PB code les mesures manquantes `-99`, et non par une cellule vide. Il y a une raison à cela — une case vide sur un registre papier est ambiguë entre « non mesuré » et « l'enquêteur a sauté la page », alors qu'une valeur sentinelle ne l'est pas — mais un lecteur de CSV n'a aucun moyen de savoir que `-99` n'est pas une mesure.

```
import pandas as pd

naif = pd.read_csv("data/raw/muac-screening-artibonite-2024.v1.csv")
print(naif["muac_mm"].mean())
```

Ce chiffre est inférieur de plusieurs millimètres à la réalité. Pire, il est *plausible* : rien n'y ressemble à une erreur, et une moyenne de PB n'est pas une valeur que la plupart des lecteurs peuvent vérifier d'un coup d'œil.

Déclarez la sentinelle à la lecture et elle n'entre jamais dans un calcul :

```
muac = pd.read_csv(
    "data/raw/muac-screening-artibonite-2024.v1.csv",
    na_values={"muac_mm": ["-99"]},
)

print(muac["muac_mm"].mean())
print(muac["muac_mm"].isna().sum())
```

```
library(readr)

muac <- read_csv(
  "data/raw/muac-screening-artibonite-2024.v1.csv",
  na = c("", "NA", "-99")
)

mean(muac$muac_mm, na.rm = TRUE)
sum(is.na(muac$muac_mm))
```

Notez la différence. pandas permet de restreindre la sentinelle à une colonne, ce qui est souhaitable : `-99` est un PB manquant, mais si une colonne portait légitimement une valeur négative, un traitement global la détruirait. Le `read_csv` de R applique `na` à tout le fichier ; restreignez ensuite la portée lorsque cela importe.

> Traiter une sentinelle n'est pas la même chose que décider quoi faire de la valeur manquante. Cette décision relève de la leçon 6. Ici, vous vous assurez seulement que le code cesse de se faire passer pour une mesure.

3.3 Les identifiants ne sont pas des nombres

`child_id` s'écrit ici `CH00854` et survit intact à l'import. Beaucoup de vrais registres utilisent des identifiants purement numériques — `00854` — et un lecteur les transformera obligeamment en l'entier `854`.

Le zéro initial a disparu, la jointure vers le fichier ménage échoue exactement pour les identifiants qui en portaient un, et l'échec ressemble à des ménages manquants plutôt qu'à une erreur de type.

```python
muac = pd.read_csv(
    chemin,
    dtype={"child_id": "string", "commune": "string"},
    na_values={"muac_mm": ["-99"]},
)
```

```r
muac <- read_csv(
  chemin,
  col_types = cols(
    child_id = col_character(),
    commune  = col_character()
  ),
  na = c("", "NA", "-99")
)
```

La règle se généralise : **si vous ne ferez jamais d'arithmétique dessus, ce n'est pas un nombre.** Codes de formation sanitaire, numéros de téléphone, identifiants de grappe, numéros de ménage et codes administratifs sont tous du texte qui s'écrit avec des chiffres.

3.4 Les dates

`screening_date` arrive sous la forme `2024-01-15`. C'est de l'ISO 8601, c'est sans ambiguïté, et les deux lecteurs l'interpréteront correctement. Sachez que vous avez de la chance.

De vrais exports produisent `15/01/2024`, `01/15/2024`, `15-janv-24` et des numéros de série Excel comme `45306`, parfois dans la même colonne, parce que trois personnes ont saisi sur trois machines configurées différemment. `01/02/2024` est soit le 1er février, soit le 2 janvier, et le fichier ne vous dira pas lequel.

```python
muac["screening_date"] = pd.to_datetime(
    muac["screening_date"], format="%Y-%m-%d", errors="raise"
)
```

```r
muac <- muac |>
```

```r
  dplyr::mutate(
    screening_date = as.Date(screening_date, format = "%Y-%m-%d")
  )

stopifnot(!any(is.na(muac$screening_date)))
```

Deux habitudes valent la peine d'être prises :

— **Énoncez explicitement le format** plutôt que de laisser l'analyseur le déduire. Un format déduit peut changer d'un fichier à l'autre selon la composition des valeurs.

— **Utilisez `errors="raise"`.** L'alternative, `errors="coerce"`, convertit toute date illisible en valeur manquante — autrement dit, un fichier dans un mauvais format de date s'importe « avec succès » avec une colonne de dates entièrement vide.

3.5 Les catégories à vocabulaire fixe

`outcome` prend quatre valeurs et `sex` en prend deux. Les déclarer comme catégorielles apporte deux choses : un objet plus léger et — plus utile — une erreur lorsqu'une valeur hors vocabulaire apparaît.

```python
issues = pd.CategoricalDtype(
    ["no-action", "referred-tsfp", "referred-otp", "referred-sc"], ordered=False
)

muac["outcome"] = muac["outcome"].astype(issues)

# Toute valeur hors vocabulaire est désormais NaN -- comptez-les avant de continuer.
print(muac["outcome"].isna().sum())
```

```r
muac <- muac |>
  dplyr::mutate(
    outcome = factor(
      outcome,
      levels = c("no-action", "referred-tsfp", "referred-otp", "referred-sc")
    )
  )

sum(is.na(muac$outcome))
```

C'est un vrai piège dans les deux langages : une valeur hors des niveaux déclarés devient manquante au lieu de déclencher une erreur. Comptez toujours les valeurs manquantes immédiatement après la conversion. Un passage de zéro à neuf signifie que neuf lignes portaient une valeur dont vous ignoriez l'existence, et il vaut mieux la voir maintenant que la découvrir sous forme de trou dans un tableau.

La colonne `oedema` de ce registre est exactement ce cas. Ennery et Desdunes l'ont renseignée de manière incohérente au premier trimestre, avec `Y` et `N` plutôt que `true` et `false`. Lue naïvement, ces lignes deviennent manquantes sans un mot.

3.6 Une lecture défensive, de bout en bout

Assemblons le tout — en affirmant ce que l'on attend, pour que le script échoue sur un mauvais fichier au lieu de produire un mauvais chiffre.

```python
import pandas as pd
```

```python
CHEMIN = "data/raw/muac-screening-artibonite-2024.v1.csv"

muac = pd.read_csv(
    CHEMIN,
    dtype={"child_id": "string", "commune": "string", "sex": "string"},
    na_values={"muac_mm": ["-99"]},
    keep_default_na=True,
)

muac["screening_date"] = pd.to_datetime(
    muac["screening_date"], format="%Y-%m-%d", errors="raise"
)

assert len(muac) == 4218, f"4218 lignes attendues, {len(muac)} obtenues"
assert muac["child_id"].notna().all(), "chaque ligne exige un identifiant"

print(muac.dtypes)
print(muac.isna().sum())
```

```r
library(readr)
library(dplyr)

CHEMIN <- "data/raw/muac-screening-artibonite-2024.v1.csv"

muac <- read_csv(
  CHEMIN,
  col_types = cols(
    child_id       = col_character(),
    commune        = col_character(),
    screening_date = col_date(format = "%Y-%m-%d"),
    age_months     = col_integer(),
    sex            = col_character(),
    muac_mm        = col_integer(),
    oedema         = col_character(),
    outcome        = col_character()
  ),
  na = c("", "NA", "-99")
)

stopifnot(nrow(muac) == 4218)
stopifnot(!any(is.na(muac$child_id)))

glimpse(muac)
colSums(is.na(muac))
```

Les assertions sont la partie que l'on saute et celle qui rapporte. Un fichier qui arrive avec 4 190 lignes au lieu de 4 218 a perdu quelque chose entre le serveur et vous, et le script doit le dire plutôt que de rapporter discrètement une charge de cas plus faible.

3.7 Lire les autres formats

Les mêmes principes s'appliquent, avec d'autres noms de fonctions.

Source	Python	R
CSV	`pd.read_csv`	`readr::read_csv`
Excel	`pd.read_excel`	`readxl::read_excel`
Stata	`pd.read_stata`	`haven::read_dta`
SPSS	`pd.read_spss`	`haven::read_sav`
Parquet	`pd.read_parquet`	`arrow::read_parquet`

Deux remarques propres au secteur. Les fichiers Stata et SPSS portent des **étiquettes de valeurs** — `1 = Oui`, `2 = Non` — que `haven` conserve alors que pandas les perd le plus souvent ; si un institut de sondage vous remet un `.dta`, lisez-le en R même si vous l'analyserez en Python. Et les exports Excel comportent fréquemment une ligne de titre fusionnée au-dessus de l'en-tête, que les deux lecteurs prendront pour l'en-tête si vous ne leur demandez pas de la sauter.

3.8 La suite

Le registre est maintenant en mémoire, avec ses types intacts. La leçon suivante le restructure — ainsi que l'export aplati à la CommCare sous lequel il aurait pu arriver — en une ligne par observation.

CHAPITRE 4

Obtenir un tableau ordonné

Leçon 4 sur 8 · 75 min

Restructurer les groupes répétés aplatis en une ligne par observation, joindre une liste de membres à son ménage, et prouver que la jointure n'a pas modifié votre charge de cas en silence.

4.1 La règle

Une ligne par observation, une colonne par variable, une table par type d'objet. La règle est ancienne et elle fait encore l'essentiel du travail.

Il vaut la peine d'en dire la raison, car « ordonné » évoque le rangement plutôt que l'ingénierie. Dans un tableau ordonné, chaque opération souhaitée — filtrer, grouper, joindre, représenter — est la même opération quelle que soit la variable traitée. Dans un tableau désordonné, chaque variable exige du code sur mesure, et c'est dans ce code sur mesure que vivent les erreurs.

4.2 À quoi ressemble le désordre dans ce secteur

Trois formes couvrent la quasi-totalité des cas.

Groupes répétés aplatis. CommCare exporte une ligne par soumission, donc une visite de ménage ayant dépisté trois enfants devient :

```
form_id,commune,child_1_id,child_1_muac,child_2_id,child_2_muac,child_3_id,child_3_muac
F0012,Gonaives,CH00854,133,CH00855,118,,
F0013,Verrettes,CH01439,143,,,,
```

Une ligne est un formulaire. Vous voulez une ligne par enfant. Le nombre de colonnes change en outre d'un export à l'autre, ce qui signifie que tout code nommant explicitement les colonnes cassera le mois prochain.

Des valeurs dans les en-têtes de colonnes. Un tableau croisé DHIS2 dispose les périodes en haut :

```
org_unit,janv-2024,févr-2024,mars-2024
Gonaives,412,388,401
```

Le mois est une donnée — c'est une valeur d'une variable nommée `période` — mais il loge dans un en-tête.

Plusieurs objets dans une même table. Un export d'enquête où les caractéristiques du ménage sont répétées sur chaque ligne de membre. Rien de grave, jusqu'à ce que quelqu'un calcule une taille moyenne de ménage et obtienne un chiffre pondéré par la taille des ménages.

4.3 Format long et format large

Passer de l'un à l'autre est la compétence mécanique la plus utile de cette formation.

```
import pandas as pd
```

```python
large = pd.read_csv("data/raw/commcare-screening-export.csv")

long = large.melt(
    id_vars=["form_id", "commune"],
    var_name="champ",
    value_name="valeur",
)

# child_1_muac -> enfant 1, champ muac
parties = long["champ"].str.extract(r"^child_(?P<rang>\d+)_(?P<attribut>.+)$")
long = pd.concat([long, parties], axis=1).dropna(subset=["rang"])

enfants = (
    long.pivot(index=["form_id", "commune", "rang"], columns="attribut", values="valeur")
    .reset_index()
    .dropna(subset=["id"])
)

print(enfants.head())
```

```r
library(dplyr)
library(tidyr)
library(readr)

large <- read_csv("data/raw/commcare-screening-export.csv")

enfants <- large |>
  pivot_longer(
    cols = starts_with("child_"),
    names_to = c("rang", "attribut"),
    names_pattern = "child_(\\d+)_(.+)",
    values_to = "valeur",
    values_transform = as.character
  ) |>
  pivot_wider(names_from = attribut, values_from = valeur) |>
  filter(!is.na(id))

enfants
```

Deux détails portent l'essentiel :

— **Repérez le rang par une expression régulière**, jamais en énumérant les colonnes. La liste change d'un export à l'autre ; le motif, non.

— **Supprimez les rangs vides.** Un formulaire ayant dépisté un seul enfant produit malgré tout des colonnes pour les rangs deux et trois, remplies de rien. Les conserver gonfle votre charge de cas de lignes qui ne sont pas des enfants.

Vérifiez ensuite l'arithmétique. Si l'export comptait 1 600 formulaires et que le plus large portait trois enfants, la restructuration produit 4 800 lignes candidates, dont seules les vraies survivent à la suppression. Ce nombre survivant est votre charge de cas, et il doit correspondre à ce que le programme pense avoir dépisté.

4.4 Joindre une liste à son parent

KoboToolbox place les répétitions dans leur propre feuille, reliée par `_parent_index`.

```python
menages = pd.read_excel("data/raw/kobo-export.xlsx", sheet_name="household")
membres = pd.read_excel("data/raw/kobo-export.xlsx", sheet_name="member")

fusion = membres.merge(
    menages[["_index", "commune", "interview_date"]],
    left_on="_parent_index",
    right_on="_index",
    how="left",
    validate="many_to_one",
    indicator=True,
)

print(fusion["_merge"].value_counts())
```

```r
library(readxl)

menages <- read_excel("data/raw/kobo-export.xlsx", sheet = "household")
membres <- read_excel("data/raw/kobo-export.xlsx", sheet = "member")

fusion <- membres |>
  left_join(
    menages |> select(`_index`, commune, interview_date),
    by = join_by(`_parent_index` == `_index`),
    relationship = "many-to-one",
    unmatched = "error"
  )

nrow(fusion) == nrow(membres)
```

`validate="many_to_one"` sous pandas et `relationship = "many-to-one"` sous dplyr sont les arguments importants, et presque personne ne les emploie. Ils déclenchent une erreur si la relation que vous avez affirmée n'est pas celle des données — c'est-à-dire exactement la défaillance qui multiplie sinon votre nombre de lignes et se découvre trois étapes plus loin sous la forme d'une charge de cas surestimée de 14 %.

4.5 Prouvez que la jointure a fait ce que vous aviez annoncé

Trois vérifications, à chaque fois. Elles prennent une minute et ont détecté plus d'erreurs qu'aucune autre technique de cette formation.

```python
avant = len(membres)
apres = len(fusion)

assert apres == avant, f"la jointure a changé le nombre de lignes : {avant} -> {apres}"

orphelins = (fusion["_merge"] == "left_only").sum()
assert orphelins == 0, f"{orphelins} membres sans ménage"

assert fusion["child_id"].is_unique, "identifiants dupliqués après jointure"
```

```r
stopifnot(nrow(fusion) == nrow(membres))
stopifnot(!any(is.na(fusion$commune)))
stopifnot(!any(duplicated(fusion$child_id)))
```

La vérification du nombre de lignes est celle à intégrer. Une jointure à gauche ne peut que laisser le nombre de lignes inchangé ou l'augmenter. S'il a augmenté, la table de droite

comportait des clés dupliquées, et chaque clé dupliquée a discrètement multiplié ses lignes. Dans un tableau de charge de cas, c'est une surestimation que vous rapporterez à un bailleur.

> Une jointure qui modifie votre nombre de lignes n'est pas un problème de données à corriger en aval. C'est le signe que vous avez mal compris l'une des deux tables, et la correction se situe en amont de la jointure.

4.6 Le registre est déjà ordonné

Le registre de dépistage PB utilisé par cette formation comporte une ligne par enfant, ce qui permet de le travailler directement :

```
muac.head()
```

```
head(muac)
```

C'est délibéré. La restructuration est une compétence réelle et vous en aurez besoin, mais ce n'est pas là que se situent les décisions intéressantes. Ces décisions commencent à la leçon suivante, lorsque vous regarderez ce que contiennent réellement ces 4 218 lignes.

4.7 La suite

Le tableau a la bonne forme. La leçon suivante établit ce qui ne va pas dedans — méthodiquement, avec une non-réponse ventilée par site et par semaine, car une non-réponse qui se concentre est un problème d'une tout autre nature qu'une non-réponse dispersée.

CHAPITRE 5

Établir ce qui ne va pas dans les données

Leçon 5 sur 8 · 90 min

Profiler méthodiquement un export récent — non-réponse par site et par semaine, mesures implausibles, doublons sans clé propre, et codes qui se contredisent entre eux.

5.1 Profiler avant d'analyser

Vous disposez d'un tableau ordonné aux types corrects. Vous ne savez pas encore ce qu'il contient.

L'instinct pousse à calculer l'indicateur et à regarder le chiffre. Résistez : le chiffre sera plausible quel que soit le défaut, et une fois que vous l'aurez vu, vous trouverez des raisons d'y croire. Profiler d'abord, décider ensuite, calculer en dernier.

Cette leçon est le profilage. Cinq vérifications, classées par l'ampleur des dégâts qu'elles causent lorsqu'on les néglige.

5.2 Vérification 1 : la complétude, par groupe et non globalement

Un taux de complétude unique ne sert quasiment à rien. Ce qui compte, c'est de savoir si la non-réponse se concentre, car une non-réponse concentrée biaise, alors qu'une non-réponse dispersée ne coûte le plus souvent que de la précision.

```python
global_ = muac.isna().mean().sort_values(ascending=False)
print(global_)

par_commune = (
    muac.assign(age_manquant=muac["age_months"].isna())
    .groupby("commune")["age_manquant"]
    .agg(["mean", "size"])
    .sort_values("mean", ascending=False)
)
print(par_commune)
```

```r
muac |>
  summarise(across(everything(), ~ mean(is.na(.x)))) |>
  tidyr::pivot_longer(everything(), names_to = "colonne", values_to = "manquant") |>
  arrange(desc(manquant))

muac |>
  group_by(commune) |>
  summarise(
    age_manquant = mean(is.na(age_months)),
    n = n()
  ) |>
  arrange(desc(age_manquant))
```

Appliquée à ce registre, la non-réponse globale de `age_months` avoisine 5 %, ce qui paraît tolérable. Ventilez-la et Gros-Morne est bien plus touchée que les autres. Ventilez également par semaine et tout se concentre sur une seule semaine de campagne, du 10 au 14 juin :

```python
gros_morne = muac[muac["commune"] == "Gros-Morne"].copy()
gros_morne["semaine"] = gros_morne["screening_date"].dt.to_period("W")

print(
    gros_morne.groupby("semaine")["age_months"]
    .apply(lambda s: s.isna().mean())
    .sort_values(ascending=False)
    .head()
)
```

```r
muac |>
  filter(commune == "Gros-Morne") |>
  mutate(semaine = lubridate::floor_date(screening_date, "week")) |>
  group_by(semaine) |>
  summarise(age_manquant = mean(is.na(age_months)), n = n()) |>
  arrange(desc(age_manquant))
```

Ce n'est pas un désagrément de qualité des données. C'est une équipe, une semaine, un formulaire de tablette dont le champ âge était mal configuré — et cela signifie que supprimer les lignes incomplètes retire bien plus de Gros-Morne que de toute autre commune. Si vous classez ensuite les communes par taux de malnutrition, vous les avez en partie classées selon le formulaire de quelle équipe était défectueux.

> Une non-réponse qui se concentre sur un site ou une semaine fait la différence entre perdre de la précision et perdre la réponse. Ventilez-la toujours avant de décider quoi en faire.

5.3 Vérification 2 : les valeurs implausibles

Chaque secteur a ses limites physiques. Servez-vous-en.

Pour le PB chez l'enfant de 6 à 59 mois, les valeurs inférieures à environ 80 mm ou supérieures à environ 220 mm ne sont pas des mesures : ce sont des erreurs de saisie. Le registre porte sept enregistrements où la mesure est restée en centimètres sans jamais être convertie, et qui apparaissent sous forme de valeurs inférieures à 40.

```python
implausibles = muac[(muac["muac_mm"] < 80) | (muac["muac_mm"] > 220)]
print(implausibles[["child_id", "commune", "age_months", "muac_mm"]])

print(muac["muac_mm"].describe())
```

```r
muac |>
  filter(muac_mm < 80 | muac_mm > 220) |>
  select(child_id, commune, age_months, muac_mm)

summary(muac$muac_mm)
```

Une valeur de 13,3 là où vous attendiez 133 est une erreur d'unité, et elle est récupérable : vous savez ce qu'elle devait être. Une valeur de 450 n'est pas récupérable et doit être traitée comme manquante. Distinguer les deux relève du jugement, et c'est pourquoi cela appartient au journal de nettoyage plutôt qu'à un filtre d'une ligne.

Vérifiez aussi l'intervalle d'âge. Un programme de dépistage PB cible les 6 à 59 mois ; une ligne indiquant 72 mois est donc soit hors périmètre, soit une erreur de saisie, et laquelle des

deux change votre dénominateur.

```python
print(muac["age_months"].describe())
print(muac[(muac["age_months"] < 6) | (muac["age_months"] > 59)].shape[0])
```

```r
summary(muac$age_months)
sum(muac$age_months < 6 | muac$age_months > 59, na.rm = TRUE)
```

5.4 Vérification 3 : les doublons, y compris ceux sans clé propre

Les doublons exacts sont faciles :

```python
exacts = muac[muac.duplicated(keep=False)]
print(f"{len(exacts)} lignes dans des groupes de doublons exacts")

ids_repetes = muac[muac.duplicated(subset=["child_id"], keep=False)]
print(f"{len(ids_repetes)} lignes partageant un child_id")
```

```r
sum(duplicated(muac))

muac |>
  group_by(child_id) |>
  filter(n() > 1) |>
  arrange(child_id)
```

Ce registre compte douze doublons exacts, issus de formulaires soumis deux fois sur une mauvaise connexion. Ceux-là sont sans ambiguïté : la même soumission est arrivée deux fois, et un exemplaire doit disparaître.

Le cas difficile est celui de l'enfant réenregistré sous un nouvel identifiant, qui ne partage aucune clé. Il y en a six dans ce fichier, repérables uniquement en appariant sur les attributs qui ne devraient pas coïncider par hasard :

```python
suspects = (
    muac.groupby(["commune", "screening_date", "age_months", "sex", "muac_mm"])
    .filter(lambda g: len(g) > 1)
    .sort_values(["commune", "screening_date", "muac_mm"])
)

print(suspects[["child_id", "commune", "screening_date", "age_months", "sex", "muac_mm"]])
```

```r
muac |>
  group_by(commune, screening_date, age_months, sex, muac_mm) |>
  filter(n() > 1) |>
  arrange(commune, screening_date, muac_mm) |>
  select(child_id, commune, screening_date, age_months, sex, muac_mm)
```

Prudence ici : deux enfants différents de même âge et de même sexe, dépistés dans la même commune le même jour, avec le même PB au millimètre près, c'est possible. Dans une grande campagne, c'est même probable. Cette vérification produit des *suspects*, non des doublons, et la résolution passe par un humain qui consulte le registre — pas par un appel à `drop_duplicates()`.

5.5 Vérification 4 : les contradictions internes

Des colonnes qui devraient concorder, et qui ne concordent pas. Ce registre porte deux défauts de ce type.

Soixante-treize enregistrements portent une décision de référencement mais aucune valeur de PB — la colonne de décision et la colonne de mesure ont été remplies par deux personnes différentes :

```python
contradiction_a = muac[
    muac["outcome"].isin(["referred-tsfp", "referred-otp", "referred-sc"])
    & muac["muac_mm"].isna()
]
print(len(contradiction_a))
```

```r
muac |>
  filter(outcome %in% c("referred-tsfp", "referred-otp", "referred-sc"),
         is.na(muac_mm)) |>
  nrow()
```

Et neuf enregistrements portent une décision qui contredit leur propre mesure — un enfant mesuré à 140 mm marqué comme référé en prise en charge ambulatoire, ou un enfant à 110 mm marqué sans action. C'est ce que produit un écran de tablette mal effleuré.

```python
def decision_attendue(ligne):
    if pd.isna(ligne["muac_mm"]):
        return None
    if ligne["oedema"] is True or ligne["muac_mm"] < 115:
        return "referred-otp"
    if ligne["muac_mm"] < 125:
        return "referred-tsfp"
    return "no-action"

controle = muac.assign(attendu=muac.apply(decision_attendue, axis=1))
incoherents = controle[
    controle["attendu"].notna()
    & (controle["attendu"] == "no-action")
    & (controle["outcome"] != "no-action")
]
print(len(incoherents))
```

```r
muac |>
  mutate(
    attendu = case_when(
      is.na(muac_mm)             ~ NA_character_,
      oedema | muac_mm < 115     ~ "referred-otp",
      muac_mm < 125              ~ "referred-tsfp",
      TRUE                       ~ "no-action"
    )
  ) |>
  filter(!is.na(attendu), attendu == "no-action", outcome != "no-action") |>
  nrow()
```

Notez que cette vérification encode une règle clinique. Elle ne vaut que ce que vaut la règle, et l'orientation vers un centre de stabilisation dépend de complications que le registre ne consigne pas — traitez donc une incohérence comme un signalement à examiner, non comme la preuve que la décision est fausse.

5.6 Vérification 5 : un même objet codé de plusieurs manières

La colonne `oedema` est renseignée en `true`/`false` sur l'essentiel du fichier, mais Ennery et Desdunes ont utilisé `Y`/`N` au premier trimestre, et certaines lignes sont vides.

```python
brut = pd.read_csv(CHEMIN, dtype={"oedema": "string"})
print(brut.groupby("commune")["oedema"].value_counts(dropna=False))
```

```r
read_csv(CHEMIN, col_types = cols(.default = col_character())) |>
  count(commune, oedema)
```

Regardez toujours les valeurs textuelles brutes d'une colonne catégorielle avant de la convertir. Convertir d'abord puis compter les manquantes vous dit *combien* de lignes portaient une valeur inattendue ; regarder les valeurs brutes vous dit *lesquelles*, ce dont vous avez besoin pour juger si elles sont récupérables.

5.7 Une fonction de profilage à conserver

Emballez le tout pour qu'elle s'exécute sur chaque nouvel export sans être retapée.

```python
def profiler(df, groupe=None):
    rapport = {
        "lignes": len(df),
        "colonnes": df.shape[1],
        "lignes_dupliquees": int(df.duplicated().sum()),
        "manquant": df.isna().mean().round(4).to_dict(),
    }
    if groupe:
        rapport["manquant_par_groupe"] = (
            df.isna().groupby(df[groupe]).mean().round(4).to_dict("index")
        )
    return rapport

import json
print(json.dumps(profiler(muac, groupe="commune"), indent=2, default=str))
```

```r
profiler <- function(df, groupe = NULL) {
  out <- list(
    lignes = nrow(df),
    colonnes = ncol(df),
    lignes_dupliquees = sum(duplicated(df)),
    manquant = sapply(df, function(x) round(mean(is.na(x)), 4))
  )
  if (!is.null(groupe)) {
    out$manquant_par_groupe <- df |>
      group_by(.data[[groupe]]) |>
      summarise(across(everything(), ~ round(mean(is.na(.x)), 4)))
  }
  out
}

str(profiler(muac, groupe = "commune"))
```

Exécutez ceci sur chaque export, enregistrez le résultat à côté des données, et vous disposez d'une trace de l'état du fichier à son arrivée. Cette trace est ce qui vous permettra de répondre six mois plus tard à la question « est-ce que ça a toujours été comme ça ? ».

5.8 La suite

Vous savez maintenant ce qui ne va pas. La leçon suivante porte sur la décision à prendre face à chaque défaut — et, plus important encore, sur la manière de consigner ces décisions sous une forme qui survive à celui qui les a prises.

CHAPITRE 6

Décisions de nettoyage et journal de nettoyage

Leçon 6 sur 8 · 75 min

Trancher le traitement de chaque défaut, chiffrer ce que la décision coûte au résultat final, et la consigner pour qu'un auditeur — ou vous-même dans six mois — puisse suivre le raisonnement.

6.1 Le nettoyage est un ensemble de décisions, pas un script

Le profilage de la leçon précédente a produit une liste de défauts. Aucun n'a de traitement objectivement correct. Chacun a un traitement défendable et une justification, et c'est la justification qui compte : un relecteur ne vous reprochera pas d'avoir supprimé sept lignes, il vous reprochera de ne pas pouvoir savoir pourquoi.

Donc, pour chaque défaut, trois éléments. Ce que vous avez fait, pourquoi, et combien de lignes cela a touché.

6.2 Les quatre options

Chaque défaut reçoit l'une d'entre elles.

Option	Quand elle s'applique	Ce qu'elle coûte
Corriger	Vous savez quelle valeur était visée	Rien, si vous avez raison
Passer en manquant	La valeur est fausse et irrécupérable	De la précision, et peut-être du biais
Supprimer la ligne	La ligne n'est pas une observation réelle	De la charge de cas, si vous vous trompez
Conserver et signaler	La valeur peut être juste, sans moyen de trancher	Rien, mais le lecteur doit la traiter

« Conserver et signaler » est sous-employée. C'est l'option honnête lorsqu'une valeur est suspecte sans être impossible, et elle renvoie le jugement à la personne qui lit l'analyse plutôt que de le dissimuler à l'intérieur.

6.3 Traitement de ce registre

Erreurs d'unité — sept enregistrements en centimètres. Récupérables. Un PB de 13,3 a été mesuré à 133 mm ; toute l'affaire tient à la virgule. Corrigez-les, et consignez-le.

```python
import numpy as np

erreur_unite = muac["muac_mm"].notna() & (muac["muac_mm"] < 40)
n_erreur_unite = int(erreur_unite.sum())

muac.loc[erreur_unite, "muac_mm"] = muac.loc[erreur_unite, "muac_mm"] * 10
```

```r
erreur_unite <- !is.na(muac$muac_mm) & muac$muac_mm < 40
n_erreur_unite <- sum(erreur_unite)

muac <- muac |>
  mutate(muac_mm = if_else(!is.na(muac_mm) & muac_mm < 40, muac_mm * 10L, muac_mm))
```

Multiplier par dix n'est sûr ici que parce que les plages plausibles ne se chevauchent pas : rien de réellement mesuré en millimètres n'est en dessous de 40, et rien de réellement mesuré en centimètres n'est au-dessus de 22. Vérifiez que les deux plages sont disjointes avant d'appliquer une règle de ce genre. Si elles se chevauchent, vous ne pouvez pas distinguer les cas et les valeurs ne sont pas récupérables.

Valeurs implausibles qui ne sont pas des erreurs d'unité. Irrécupérables. Passez-les en manquant, mais ne supprimez pas la ligne : l'enfant a bien été dépisté, et le reste de l'enregistrement reste une donnée probante.

```python
implausible = muac["muac_mm"].notna() & (
    (muac["muac_mm"] < 80) | (muac["muac_mm"] > 220)
)
n_implausible = int(implausible.sum())

muac.loc[implausible, "muac_mm"] = np.nan
```

```r
implausible <- !is.na(muac$muac_mm) & (muac$muac_mm < 80 | muac$muac_mm > 220)
n_implausible <- sum(implausible)

muac <- muac |>
  mutate(muac_mm = if_else(implausible, NA_integer_, muac_mm))
```

Doublons exacts — douze lignes. À supprimer. Un formulaire soumis deux fois sur une mauvaise connexion correspond à un dépistage, pas à deux.

```python
avant = len(muac)
muac = muac.drop_duplicates()
n_doublons_exacts = avant - len(muac)
```

```r
avant <- nrow(muac)
muac <- distinct(muac)
n_doublons_exacts <- avant - nrow(muac)
```

Réenregistrements sous un nouvel identifiant — six suspectés. À conserver et signaler. Vous ne pouvez pas les distinguer d'authentiques coïncidences sans retourner au registre : marquez-les et dites-le dans le rapport.

```python
cle = ["commune", "screening_date", "age_months", "sex", "muac_mm"]
muac["doublon_suspecte"] = muac.duplicated(subset=cle, keep=False) & muac[
    "muac_mm"
].notna()

n_suspectes = int(muac["doublon_suspecte"].sum())
```

```r
muac <- muac |>
  group_by(commune, screening_date, age_months, sex, muac_mm) |>
  mutate(doublon_suspecte = n() > 1 & !is.na(muac_mm)) |>
  ungroup()

n_suspectes <- sum(muac$doublon_suspecte)
```

Codage incohérent des œdèmes. Récupérable. `Y` et `N` signifient bien ce qu'ils paraissent signifier ; les cases vides, non, et elles deviennent manquantes.

```python
table_oedeme = {
    "true": True, "TRUE": True, "Y": True, "y": True, "yes": True,
    "false": False, "FALSE": False, "N": False, "n": False, "no": False,
}
muac["oedema"] = muac["oedema"].astype("string").str.strip().map(table_oedeme)
n_oedeme_manquant = int(muac["oedema"].isna().sum())
```

```r
muac <- muac |>
  mutate(
    oedema = case_when(
      tolower(trimws(oedema)) %in% c("true", "y", "yes") ~ TRUE,
      tolower(trimws(oedema)) %in% c("false", "n", "no") ~ FALSE,
      TRUE ~ NA
    )
  )

n_oedeme_manquant <- sum(is.na(muac$oedema))
```

Âge manquant concentré sur Gros-Morne. Voilà le cas qui exige une décision plutôt qu'une règle, et il fait l'objet de la section suivante.

6.4 Chiffrez ce que la décision coûte

La raison de prendre l'âge manquant au sérieux, c'est que le traitement évident — supprimer les lignes sans âge — change la réponse. Mesurez-le au lieu de le supposer.

```python
seuil_mag = 125

complet = muac.dropna(subset=["age_months", "muac_mm"])
tous_mesures = muac.dropna(subset=["muac_mm"])

def taux_mag(df):
    return (df["muac_mm"] < seuil_mag).mean()

comparaison = pd.DataFrame(
    {
        "en_supprimant": complet.groupby("commune").apply(taux_mag),
        "en_conservant": tous_mesures.groupby("commune").apply(taux_mag),
    }
)
comparaison["ecart"] = (
    comparaison["en_supprimant"] - comparaison["en_conservant"]
)
print(comparaison.sort_values("ecart", ascending=False).round(4))
```

```r
taux_mag <- function(df) mean(df$muac_mm < 125, na.rm = TRUE)

complet <- muac |> filter(!is.na(age_months), !is.na(muac_mm))
tous_mesures <- muac |> filter(!is.na(muac_mm))

comparaison <- full_join(
  complet |> group_by(commune) |> summarise(en_supprimant = taux_mag(pick(everything()))),
  tous_mesures |> group_by(commune) |> summarise(en_conservant = taux_mag(pick(everything
    ()))),
  by = "commune"
) |>
  mutate(ecart = en_supprimant - en_conservant) |>
```

```
  arrange(desc(abs(ecart)))

comparaison
```

Gros-Morne bouge, et les autres communes à peine. Voilà le biais, rendu visible. Comme l'indicateur ne dépend pas ici de l'âge — les seuils de PB pour les 6 à 59 mois forment une bande unique, à la différence des scores z poids-taille — la bonne décision consiste à conserver les lignes, à les utiliser pour l'indicateur PB, et à ne les exclure que des analyses qui exigent réellement l'âge.

C'est une meilleure décision que la suppression, et elle n'est disponible que parce que vous avez regardé.

> Principe général : supprimez des lignes pour une analyse donnée, jamais du jeu de données. Une ligne sans âge reste une donnée probante sur le PB. Filtrer au point d'utilisation maintient chaque analyse sur le plus grand échantillon qu'elle admette.

6.5 Le journal de nettoyage

Tout ce qui précède devient des lignes d'un tableau livré avec l'analyse.

```
journal_nettoyage = pd.DataFrame(
    [
        {
            "regle": "Erreur d'unité PB corrigée (cm vers mm)",
            "colonne": "muac_mm",
            "action": "corrige",
            "lignes": n_erreur_unite,
            "justification": "Les valeurs sous 40 sont des centimètres non convertis. "
            "Les plages plausibles en mm et en cm sont disjointes, la correction est donc
    sans ambiguïté.",
        },
        {
            "regle": "PB implausible passé en manquant",
            "colonne": "muac_mm",
            "action": "passe-manquant",
            "lignes": n_implausible,
            "justification": "Hors de 80-220 mm, ce n'est pas une mesure pour 6-59 mois. "
            "Ligne conservée : le dépistage a bien eu lieu.",
        },
        {
            "regle": "Doubles soumissions exactes supprimées",
            "colonne": "toutes",
            "action": "supprime",
            "lignes": n_doublons_exacts,
            "justification": "Des lignes identiques correspondent à un formulaire soumis
    deux fois sur une mauvaise connexion.",
        },
        {
            "regle": "Réenregistrement suspecté signalé",
            "colonne": "doublon_suspecte",
            "action": "signale",
            "lignes": n_suspectes,
            "justification": "Mêmes commune, date, âge, sexe et PB sous des identifiants
    différents. "
            "Indiscernable d'une coïncidence sans le registre papier.",
        },
        {
```

```
            "regle": "Codage des œdèmes harmonisé",
            "colonne": "oedema",
            "action": "corrige",
            "lignes": n_oedeme_manquant,
            "justification": "Ennery et Desdunes ont utilisé Y/N au T1. Les cases vides
    restent manquantes.",
        },
        {
            "regle": "Âge manquant conservé",
            "colonne": "age_months",
            "action": "conserve",
            "lignes": int(muac["age_months"].isna().sum()),
            "justification": "60 % de manquants sur la semaine du 10 au 14 juin à Gros-
    Morne. "
            "Supprimer modifie le classement des communes ; les seuils de PB n'exigent pas
     l'âge.",
        },
    ]
)

journal_nettoyage.to_csv("output/tables/journal-nettoyage.csv", index=False)
print(journal_nettoyage[["regle", "action", "lignes"]])
```

```
journal_nettoyage <- tibble::tribble(
  ~regle,                                  ~colonne,             ~action,          ~
    lignes,
  "Erreur d'unité PB corrigée (cm vers mm)",   "muac_mm",            "corrige",        n_
    erreur_unite,
  "PB implausible passé en manquant",          "muac_mm",            "passe-manquant", n_
    implausible,
  "Doubles soumissions exactes supprimées",    "toutes",             "supprime",       n_
    doublons_exacts,
  "Réenregistrement suspecté signalé",         "doublon_suspecte",   "signale",        n_
    suspectes,
  "Codage des œdèmes harmonisé",               "oedema",             "corrige",        n_
    oedeme_manquant,
  "Âge manquant conservé",                     "age_months",         "conserve",       sum
    (is.na(muac$age_months))
)

readr::write_csv(journal_nettoyage, "output/tables/journal-nettoyage.csv")
journal_nettoyage
```

Quatre propriétés rendent ce journal utile plutôt que cérémoniel :

— **Les effectifs viennent du code**, pas de la mémoire. Si la règle change, l'effectif change avec elle.

— **Il est livré avec l'analyse** — en annexe du rapport, et non comme un fichier sur le portable de quelqu'un.

— **La justification est une phrase, pas une catégorie.** « Qualité des données » n'est pas une justification.

— **Il s'écrit pendant le nettoyage**, et non reconstitué après coup. Les journaux reconstitués sont faux précisément là où cela compte.

6.6 La seule chose à ne jamais faire

Ne modifiez pas `data/raw/`. Pas une cellule, pas pour corriger une coquille manifeste, pas « juste cette fois » la veille d'une échéance.

Chacune des corrections ci-dessus est du code. Elle est donc visible, réversible, appliquée à l'identique à l'export du trimestre suivant, et réexécutable par quelqu'un qui doute de vous. Une cellule modifiée à la main dans un tableur n'est rien de tout cela, et elle est invisible six mois plus tard lorsque le chiffre est contesté.

6.7 La suite

Les données sont propres et les décisions sont consignées. La leçon suivante les transforme en indicateur — ce qui suppose d'affronter le fait qu'un taux est un numérateur rapporté à un dénominateur, et que presque personne ne s'accorde sur le dénominateur.

CHAPITRE 7

Des données ordonnées au tableau d'indicateurs

Leçon 7 sur 8 · 90 min

Transformer un jeu de données propre en tableau d'indicateurs désagrégé conforme au cadre logique, avec numérateur, dénominateur et intervalle de confiance explicitement énoncés.

7.1 Un indicateur est un numérateur rapporté à un dénominateur, désagrégé

Cette phrase résume toute la leçon. La plupart des désaccords de rapportage sont en réalité des désaccords sur le dénominateur, et la plupart des autres portent sur les lignes qui appartiennent au numérateur.

Donc, avant tout code : écrivez la définition.

7.2 La fiche de référence d'indicateur

Pour chaque indicateur que vous rapportez, remplissez ceci. Cela prend cinq minutes et met fin au débat avant qu'il ne commence.

Champ	Pour notre exemple
Indicateur	Prévalence de la malnutrition aiguë globale (MAG) par PB
Numérateur	Enfants de 6 à 59 mois dépistés avec un PB inférieur à 125 mm, ou porteurs d'œdèmes bilatéra
Dénominateur	Enfants de 6 à 59 mois disposant d'une mesure de PB valide ou d'une évaluation des œdèmes c
Désagrégation	Commune, sexe, tranche d'âge (6-23, 24-59 mois)
Fréquence	Trimestrielle, et annuelle pour le rapport bailleur
Source	Registre de dépistage de masse communautaire, CommCare
Décision éclairée	Quelles communes reçoivent un site PCIMA supplémentaire au trimestre suivant
Valeur de référence	Une MAG égale ou supérieure à 15 % constitue le seuil d'urgence

Notez ce que le dénominateur n'est **pas**. Ce n'est pas l'ensemble des enfants de la commune — ce serait une prévalence ajustée sur la couverture, qui exige un chiffre de population dont ce registre ne dispose pas. Ce n'est pas non plus l'ensemble des lignes du fichier — cela inclurait des lignes sans aucune mesure. Être explicite là-dessus est ce qui empêche le chiffre d'être discrètement redéfini d'un rapport à l'autre.

> Si vous ne savez pas énoncer le dénominateur en une phrase, vous n'avez pas encore d'indicateur. Vous avez une colonne dont vous vous apprêtez à prendre la moyenne.

7.3 Construisez le numérateur comme une colonne

Ne filtrez pas. Créez un indicateur binaire, conservez toutes les lignes, et laissez l'agrégation faire le travail — le dénominateur reste ainsi visible dans le même tableau que le numérateur.

```
import numpy as np

MAS_MM = 115
```

```python
MAG_MM = 125

muac["evalue"] = muac["muac_mm"].notna() | muac["oedema"].notna()

muac["mas"] = np.where(
    ~muac["evalue"],
    np.nan,
    ((muac["muac_mm"] < MAS_MM) | (muac["oedema"] == True)).astype(float),
)

muac["mag"] = np.where(
    ~muac["evalue"],
    np.nan,
    ((muac["muac_mm"] < MAG_MM) | (muac["oedema"] == True)).astype(float),
)
```

```r
MAS_MM <- 115
MAG_MM <- 125

muac <- muac |>
  mutate(
    evalue = !is.na(muac_mm) | !is.na(oedema),
    mas = if_else(evalue, (muac_mm < MAS_MM) | oedema %in% TRUE, NA),
    mag = if_else(evalue, (muac_mm < MAG_MM) | oedema %in% TRUE, NA)
  )
```

Deux détails faciles à manquer et coûteux à négliger :

— **Les œdèmes signent une MAS indépendamment de la mesure.** Un enfant à 130 mm porteur d'œdèmes bilatéraux prenant le godet est en malnutrition aiguë sévère. Filtrer sur le seul `muac_mm` sous-estime la charge de cas, et la sous-estime précisément parmi les cas les plus graves.

— **`oedema == True` plutôt qu'un test de véracité.** Un œdème manquant n'est pas un œdème absent. En R, `oedema %in% TRUE` traite `NA` comme absence d'œdème pour cet indicateur tout en le laissant distinguable ailleurs ; en Python, la comparaison explicite fait la même chose.

7.4 Le tableau d'indicateurs

```python
def tableau_indicateurs(df, par):
    groupes = df.groupby(par, dropna=False)
    tableau = groupes.agg(
        depistes=("child_id", "size"),
        denominateur=("evalue", "sum"),
        cas_mas=("mas", "sum"),
        cas_mag=("mag", "sum"),
    )
    tableau["taux_mag"] = tableau["cas_mag"] / tableau["denominateur"]
    tableau["taux_mas"] = tableau["cas_mas"] / tableau["denominateur"]
    return tableau.reset_index()

par_commune = tableau_indicateurs(muac, "commune").sort_values(
    "taux_mag", ascending=False
)
print(par_commune.round(4))
```

```r
tableau_indicateurs <- function(df, par) {
  df |>
    group_by(across(all_of(par))) |>
    summarise(
      depistes     = n(),
      denominateur = sum(evalue),
      cas_mas      = sum(mas, na.rm = TRUE),
      cas_mag      = sum(mag, na.rm = TRUE),
      .groups = "drop"
    ) |>
    mutate(
      taux_mag = cas_mag / denominateur,
      taux_mas = cas_mas / denominateur
    )
}

par_commune <- tableau_indicateurs(muac, "commune") |> arrange(desc(taux_mag))
par_commune
```

Le tableau porte volontairement `depistes` et `denominateur` en colonnes distinctes. Ils diffèrent des lignes sans aucune évaluation, et un lecteur qui voit les deux peut juger sur quelle part du registre repose le taux. Un tableau ne rapportant que le taux le dissimule entièrement.

Exécutez-le : le taux de MAG global s'établit autour de 8,6 % et la MAS autour de 2,2 %, avec une amplitude d'environ 5 % à 15 % selon les communes. Une commune franchit le seuil d'urgence de 15 % ; la meilleure en est loin. Ces valeurs sont énoncées dans les notes de qualité du jeu de données précisément pour que vous puissiez contrôler votre travail — si votre MAG ressort à 30 %, vous avez un bogue, pas une famine.

7.5 La désagrégation

Le cadre logique demandait commune, sexe et tranche d'âge. La tranche d'âge doit être construite, et ses bornes sont un choix à énoncer.

```python
muac["tranche_age"] = pd.cut(
    muac["age_months"],
    bins=[6, 24, 60],
    labels=["6-23 mois", "24-59 mois"],
    right=False,
)

par_sexe = tableau_indicateurs(muac, ["commune", "sex"])
par_age = tableau_indicateurs(muac, "tranche_age")

print(par_age.round(4))
```

```r
muac <- muac |>
  mutate(
    tranche_age = cut(
      age_months,
      breaks = c(6, 24, 60),
      labels = c("6-23 mois", "24-59 mois"),
      right = FALSE
    )
  )

par_sexe <- tableau_indicateurs(muac, c("commune", "sex"))
```

```r
par_age <- tableau_indicateurs(muac, "tranche_age")

par_age
```

`right = FALSE` rend les bandes fermées à gauche : de 6 à 23 mois révolus, puis de 24 à 59. L'inverse place les enfants de 24 mois dans la tranche inférieure, ce qui déplace légèrement tous les taux et reste invisible dans le résultat. Énoncez la convention dans la fiche de référence.

Rappelez-vous le problème d'âge manquant de la leçon 5. La désagrégation par âge exclut nécessairement les lignes sans âge, et Gros-Morne y est surreprésentée. Publiez le tableau par âge avec cette réserve attachée, ou publiez-le en excluant Gros-Morne et dites-le.

7.6 Énoncez votre degré d'incertitude

Un taux calculé sur 340 enfants n'est pas la même affirmation qu'un taux calculé sur 12. Attachez-y un intervalle.

```python
from scipy.stats import beta

def intervalle(succes, n, confiance=0.95):
    if n == 0:
        return (np.nan, np.nan)
    bas = beta.ppf((1 - confiance) / 2, succes, n - succes + 1) if succes > 0 else 0.0
    haut = beta.ppf(1 - (1 - confiance) / 2, succes + 1, n - succes) if succes < n else
    1.0
    return (bas, haut)

bornes = par_commune.apply(
    lambda r: intervalle(r["cas_mag"], r["denominateur"]), axis=1
)
par_commune["mag_bas"] = [b[0] for b in bornes]
par_commune["mag_haut"] = [b[1] for b in bornes]

print(par_commune[["commune", "denominateur", "taux_mag", "mag_bas", "mag_haut"]].round(4)
    )
```

```r
intervalle <- function(cas, n) {
  if (n == 0) return(c(NA_real_, NA_real_))
  test <- binom.test(cas, n)
  test$conf.int
}

par_commune <- par_commune |>
  rowwise() |>
  mutate(
    mag_bas  = intervalle(cas_mag, denominateur)[1],
    mag_haut = intervalle(cas_mag, denominateur)[2]
  ) |>
  ungroup()

par_commune |> select(commune, denominateur, taux_mag, mag_bas, mag_haut)
```

Regardez à présent le classement. Plusieurs communes ont des intervalles qui se recouvrent, ce qui signifie que l'ordre entre elles n'est pas soutenu par les données. Si la décision éclairée par ce tableau est « quelle commune reçoit le site PCIMA supplémentaire », cela compte énormément — et un tableau sans intervalles vous aurait laissé les classer avec une assurance infondée.

Il s'agit par ailleurs d'un recensement des enfants dépistés et non d'un échantillon probabiliste : l'intervalle ne décrit donc que la variation d'échantillonnage. Il ne dit rien sur la question de savoir si les enfants venus au dépistage ressemblent à ceux qui n'y sont pas venus, ce qui constitue généralement la plus grande source d'erreur et relève de la section des limites.

7.7 Mettez-le en forme pour le rapport

```python
rapport = par_commune.assign(
    mag=lambda d: (d["taux_mag"] * 100).round(1).astype(str) + "%",
    ic=lambda d: "("
    + (d["mag_bas"] * 100).round(1).astype(str)
    + " – "
    + (d["mag_haut"] * 100).round(1).astype(str)
    + ")",
)[["commune", "depistes", "denominateur", "cas_mag", "mag", "ic"]]

rapport.columns = [
    "Commune", "Dépistés", "Évalués", "Cas MAG", "Taux MAG", "IC 95%",
]
rapport.to_csv("output/tables/mag-par-commune.csv", index=False)
print(rapport.to_string(index=False))
```

```r
rapport <- par_commune |>
  transmute(
    Commune     = commune,
    `Dépistés`  = depistes,
    `Évalués`   = denominateur,
    `Cas MAG`   = cas_mag,
    `Taux MAG`  = sprintf("%.1f%%", taux_mag * 100),
    `IC 95%`    = sprintf("(%.1f – %.1f)", mag_bas * 100, mag_haut * 100)
  )

readr::write_csv(rapport, "output/tables/mag-par-commune.csv")
rapport
```

Chaque colonne de ce tableau est défendable : vous pouvez dire d'où vient chaque chiffre, ce qu'il a compté, ce qu'il n'a pas compté, et avec quelle incertitude.

7.8 La suite

La dernière leçon fait en sorte que tout cela s'exécute à nouveau — sur l'export du trimestre suivant, sur le portable de quelqu'un d'autre, sans rien modifier.

CHAPITRE 8

Faire en sorte que cela tourne encore au trimestre suivant

Leçon 8 sur 8 · 60 min

Transformer une analyse qui fonctionne en une analyse qui se relance sur un nouvel export sans modification, échoue bruyamment quand l'entrée change, et se transmet à quelqu'un qui n'a rien de votre contexte.

8.1 Le test

L'export du trimestre suivant arrive. Combien d'éléments devez-vous modifier avant que le rapport ne se régénère ?

Si la réponse dépasse un — le chemin du fichier d'entrée — l'analyse n'est pas reproductible : c'est le compte rendu d'un après-midi. Cette leçon ramène la réponse à un.

8.2 Découpez l'analyse en étapes qui se passent des fichiers

Trois scripts, chacun avec une seule mission, chacun écrivant un fichier que le suivant lit.

```
scripts/
  01-lecture.py      export brut  -> data/interim/muac-type.parquet
  02-nettoyage.py    typé         -> data/processed/muac-propre.parquet
                                     output/tables/journal-nettoyage.csv
  03-indicateurs.py  propre       -> output/tables/mag-par-commune.csv
```

L'intérêt n'est pas le rangement. C'est que, lorsque le tableau d'indicateurs paraît faux, vous puissiez relancer la troisième étape en deux secondes au lieu de réexécuter un notebook de quinze minutes, et qu'un collègue qui débogue la deuxième n'ait pas à comprendre la troisième.

Utilisez Parquet plutôt que CSV entre les étapes. Ce format préserve les types — autrement dit, le travail fait en leçon 3 pour que `child_id` soit lu comme du texte n'est pas défait à la première écriture-relecture.

```python
muac.to_parquet("data/interim/muac-type.parquet", index=False)
```

```r
arrow::write_parquet(muac, "data/interim/muac-type.parquet")
```

8.3 Rassemblez les paramètres en un seul endroit

Chaque seuil, chemin et valeur de coupure figure en tête de fichier ou dans une configuration, jamais enfoui au milieu d'une expression.

```python
# config.py
from pathlib import Path

BRUT = Path("data/raw/muac-screening-artibonite-2024.v1.csv")
INTERMEDIAIRE = Path("data/interim/muac-type.parquet")
TRAITE = Path("data/processed/muac-propre.parquet")
```

```
TABLEAUX = Path("output/tables")

MAS_MM = 115
MAG_MM = 125
PB_PLAUSIBLE = (80, 220)
AGE_MOIS = (6, 59)
CODE_MANQUANT = "-99"
```

```
# config.R
BRUT          <- "data/raw/muac-screening-artibonite-2024.v1.csv"
INTERMEDIAIRE <- "data/interim/muac-type.parquet"
TRAITE        <- "data/processed/muac-propre.parquet"
TABLEAUX      <- "output/tables"

MAS_MM        <- 115
MAG_MM        <- 125
PB_PLAUSIBLE  <- c(80, 220)
AGE_MOIS      <- c(6, 59)
CODE_MANQUANT <- "-99"
```

Un relecteur qui veut connaître les seuils employés lit un seul fichier. Un successeur qui adapte ce travail à un autre pays modifie un seul fichier. Et quand l'OMS révise une valeur de coupure, vous changez un nombre au lieu de chercher `125` dans quatre scripts en manquant celui qui s'écrit `12.5 * 10`.

8.4 Faites échouer la chaîne bruyamment

Une analyse qui produit silencieusement un chiffre faux est pire qu'une analyse qui plante. Affirmez ce qui doit être vrai, à chaque frontière d'étape.

```
def controler_entree(df):
    attendues = {
        "child_id", "commune", "screening_date", "age_months",
        "sex", "muac_mm", "oedema", "outcome",
    }
    absentes = attendues - set(df.columns)
    if absentes:
        raise ValueError(f"colonnes absentes de l'export : {sorted(absentes)}")

    if df["child_id"].isna().any():
        raise ValueError("lignes sans identifiant")

    mesures = df["muac_mm"].dropna()
    if len(mesures) == 0:
        raise ValueError("aucune mesure de PB -- mauvais fichier ?")

    part_manquante = df["muac_mm"].isna().mean()
    if part_manquante > 0.20:
        raise ValueError(
            f"{part_manquante:.1%} de PB manquants -- au-dessus de la tolérance de 20 %. "
            "À investiguer avant tout rapportage."
        )
```

```
controler_entree <- function(df) {
  attendues <- c("child_id", "commune", "screening_date", "age_months",
                 "sex", "muac_mm", "oedema", "outcome")
  absentes <- setdiff(attendues, names(df))
```

```
  if (length(absentes) > 0) {
    stop("colonnes absentes de l'export : ", paste(absentes, collapse = ", "))
  }

  if (any(is.na(df$child_id))) stop("lignes sans identifiant")

  part_manquante <- mean(is.na(df$muac_mm))
  if (part_manquante > 0.20) {
    stop(sprintf(
      "%.1f%% de PB manquants - au-dessus de la tolerance de 20%%. A investiguer avant
    tout rapportage.",
      part_manquante * 100
    ))
  }
}
```

Le contrôle de seuil est le plus intéressant. Il encode un jugement éditorial — au-delà de 20 % de manquants, le taux ne mérite pas d'être publié — et l'applique automatiquement à chaque export futur, y compris à ceux traités par quelqu'un qui n'a jamais lu cette leçon.

> Écrivez l'assertion correspondant à la défaillance qui vous embarrasserait, pas à celle que vous jugez probable. Les probables, vous les remarquerez.

8.5 Paramétrez au lieu de copier

Douze communes, douze rapports communaux. Ne copiez pas le script douze fois.

```
import sys
from pathlib import Path

def produire_rapport(commune: str | None = None):
    df = pd.read_parquet(TRAITE)
    if commune:
        df = df[df["commune"] == commune]
        if df.empty:
            raise ValueError(f"aucune ligne pour la commune {commune!r}")

    tableau = tableau_indicateurs(df, "tranche_age")
    nom = commune.lower().replace(" ", "-") if commune else "toutes"
    tableau.to_csv(TABLEAUX / f"mag-{nom}.csv", index=False)
    return tableau

if __name__ == "__main__":
    cible = sys.argv[1] if len(sys.argv) > 1 else None
    produire_rapport(cible)
```

```
produire_rapport <- function(commune = NULL) {
  df <- arrow::read_parquet(TRAITE)
  if (!is.null(commune)) {
    df <- filter(df, commune == !!commune)
    if (nrow(df) == 0) stop("aucune ligne pour la commune ", commune)
  }

  tableau <- tableau_indicateurs(df, "tranche_age")
  nom <- if (is.null(commune)) "toutes" else tolower(gsub(" ", "-", commune))
  readr::write_csv(tableau, file.path(TABLEAUX, paste0("mag-", nom, ".csv")))
```

```
  tableau
}

args <- commandArgs(trailingOnly = TRUE)
produire_rapport(if (length(args) > 0) args[1] else NULL)
```

Le même schéma s'étend au rapport narratif complet. Quarto compose un modèle unique avec un paramètre, douze fois, produisant douze documents qui ne peuvent pas diverger puisqu'il n'existe qu'un seul document.

8.6 Ce qui ne va jamais dans le dépôt

Ce point compte davantage dans ce secteur qu'ailleurs. Un dépôt contenant des données de programme peut contenir des noms de bénéficiaires, des points GPS qui identifient un ménage, ou une liste de cas de protection.

— **Ne versionnez jamais de données brutes de bénéficiaires.** Ni identifiées, ni pseudonymisées. Git conserve chaque version indéfiniment, et supprimer un fichier ne le retire pas de l'historique.

— **Ne versionnez jamais d'identifiants de connexion.** Jetons DHIS2, clés d'API KoboToolbox, mots de passe de base de données. Lisez-les depuis l'environnement.

— **Ajoutez `data/raw/` au `.gitignore`** et documentez dans le README où se trouve le vrai fichier — un disque partagé, un volume chiffré.

```
data/raw/
data/interim/
data/processed/
.env
*.Rhistory
__pycache__/
.Rproj.user/
```

Le code d'analyse est ce qui a sa place sous gestion de versions. Les données ont la leur là où la politique de protection des données de votre organisation le prescrit, et le README doit indiquer où.

8.7 Le README de passation

Rédigé pour quelqu'un qui aura votre intitulé de poste et rien de votre contexte. Cinq rubriques.

```
# Analyse du dépistage PB, Artibonite

## Ce que cela produit
Taux trimestriels de MAG et de MAS par commune, sexe et tranche d'âge, avec
intervalles à 95 %. Alimente la décision d'implantation des sites PCIMA et le
rapport bailleur trimestriel.

## D'où viennent les données
Export CommCare, projet `artibonite-nutrition`, formulaire `Dépistage
communautaire`. Exporté chaque mois par l'assistant suivi-évaluation vers le
disque partagé à <chemin>. Absent de ce dépôt.

## Comment l'exécuter
```

```
    uv sync
    uv run python scripts/01-lecture.py
    uv run python scripts/02-nettoyage.py
    uv run python scripts/03-indicateurs.py

## Décisions à connaître
Voir output/tables/journal-nettoyage.csv. Celle qui compte : les lignes sans
âge sont conservées, car 60 % de la semaine du 10 au 14 juin à Gros-Morne est
sans âge et leur suppression modifie le classement des communes.

## Qui contacter
Définitions d'indicateurs : <nom>, responsable suivi-évaluation.
Problèmes de collecte : <nom>, coordinateur terrain.
```

Ce README fait la différence entre un successeur qui poursuit votre travail et un successeur qui le reconstruit sur tableur faute d'avoir pu deviner ce que vos scripts supposaient.

8.8 Pour aller plus loin

Vous savez désormais mener un export de programme de son arrivée à un tableau d'indicateurs défendable, dans l'un ou l'autre langage, avec le raisonnement consigné. C'est le socle sur lequel s'appuie le reste du programme :

— **Nettoyage et validation des données** approfondit le profilage et les corrections des leçons 5 et 6, avec l'appariement approximatif et les jeux de règles de plausibilité.

— **Conception d'indicateurs et cadre logique** part de la fiche de référence de la leçon 7 et remonte jusqu'à la théorie du changement que l'indicateur est censé attester.

— **Analyse d'enquêtes, échantillonnage et pondération** couvre ce que cette formation n'a pas abordé : ce qui change lorsque les données constituent un échantillon probabiliste et non un recensement des personnes dépistées.

Entraînez-vous sur un autre jeu de données avant de poursuivre. L'enquête ménage EAH et le registre de couverture vaccinale de routine de cette plateforme portent un tout autre assortiment de défauts, et en traiter un de bout en bout sans la leçon sous les yeux est le véritable test de ce qui est resté.

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/data-analysis-foundations

Généré le 26 juillet 2026.