

4 Data Management Types, Sources, Collection, and Cleaning

Data Types and Structures

Health datasets combine several structural types that must be recognised before any cleaning or analysis begins: cross-sectional data (one observation per subject, as in a single survey round), longitudinal/panel data (repeated observations per subject over time, as in a two-wave adolescent health cohort), and hierarchical/multilevel data (patients nested within facilities nested within LGAs nested within states, as in HMIS data). Getting the structure right determines everything downstream: a longitudinal dataset analysed as if it were cross-sectional will treat correlated repeated measures from the same person as independent, understating uncertainty and, in some cases, reversing the sign of an apparent effect.

Variables themselves are typically classified as nominal (unordered categories, e.g., state of residence), ordinal (ordered categories, e.g., severity grade), interval/ratio (numeric, e.g., haemoglobin level), or count (non-negative integers, e.g., number of ANC visits). This classification determines which summary statistics, visualisations, and models are appropriate — a point revisited throughout Chapters 5, 6, and 9.

Data Collection Methods

Primary data collection in Nigerian health research spans structured questionnaires (paper-based or electronic, e.g., via ODK/KoboToolbox), clinical measurement and laboratory testing, extraction from patient case notes or registers, and increasingly, respondent-driven and venue-based sampling for hard-to-reach populations. Each method carries characteristic error patterns: interviewer-administered questionnaires are subject to social desirability bias (particularly for sensitive topics such as sexual behaviour or substance use), register extraction is subject to transcription error and missing fields, and electronic data capture, while reducing transcription error, can introduce systematic errors from poorly designed skip logic or default values that respondents never actually confirmed.

Missing Data

Mechanisms and Consequences

Not all missing data are equal. Rubin's classic framework distinguishes three mechanisms: missing completely at random (MCAR, where missingness is unrelated to any variable), missing at random (MAR, where missingness depends on observed variables but not the missing value itself), and missing not at random (MNAR, where missingness depends on the unobserved value itself). A patient who defaults from an HIV treatment cohort because their viral load was rising (and therefore they felt too unwell to attend, or became discouraged) is a textbook MNAR scenario — the very outcome of interest drives the missingness — and this is common enough in Nigerian cohort and programme data that analysts should treat MCAR as the exception rather than the default assumption.

- Complete-case analysis (listwise deletion) is only unbiased under MCAR, and even then it discards information and reduces power.

- Single imputation (mean/last-observation-carried-forward) understates uncertainty and can bias estimates, especially LOCF in longitudinal data with a true trend.
- Multiple imputation (e.g., via chained equations, implemented in R's 'mice' package or Python's 'IterativeImputer') is the modern standard under MAR, generating several plausible completed datasets and pooling results to reflect imputation uncertainty.

Detecting and Handling Data Quality Issues

A systematic data-cleaning workflow checks, in order: (1) structural integrity — do row and column counts match expectations, are identifiers unique; (2) range and plausibility — is a reported age of 200, a temperature of 15°C, or a negative test count present; (3) internal consistency — does a 'pregnant male' or a 'first ANC visit after delivery date' appear; (4) duplicate detection — are the same patient or facility records entered more than once, whether as exact duplicates or as near-duplicates from re-keying; and (5) cross-source reconciliation — for indicators reported in more than one system (e.g., HMIS versus a Global Fund diagnostic register), does the discrepancy fall within an expected tolerance or flag a systemic issue such as the HMIS test-positivity inflation discussed in Chapter 2.

Every cleaning decision of this kind should be logged, not applied silently: a reproducible script (rather than manual editing in a spreadsheet) that documents exactly which rows were dropped, imputed, or flagged, and why, is the single most important habit this chapter can instill, because it is the difference between an analysis that can survive scrutiny — by a supervisor, a journal reviewer, or a Ministry of Health technical committee — and one that cannot.

Data Management

Types, Sources, Collection, and Cleaning

R example

```
# R: a minimal, auditable cleaning workflow for a facility-level malaria dataset
library(dplyr)

malaria_raw <- read.csv("hmis_malaria_monthly.csv")

malaria_clean <- malaria_raw %>%
  distinct(facility_id, report_month, .keep_all = TRUE) %>% # drop exact duplicates
  filter(tests_performed >= confirmed_positive, # internal consistency
         tests_performed >= 0, confirmed_positive >= 0) %>%
  mutate(tpr = confirmed_positive / tests_performed,
         tpr_flag = tpr > 0.95 | is.nan(tpr)) # plausibility flag

cat("Rows removed by cleaning:", nrow(malaria_raw) - nrow(malaria_clean), "\n")
cat("Facility-months flagged for review:", sum(malaria_clean$tpr_flag, na.rm = TRUE), "\n")
```

Data Governance, Storage, and Reproducibility

Health data management does not end with cleaning; it requires a governance discipline: version-controlled raw and cleaned datasets (never overwrite a raw extract), a data dictionary documenting every variable's definition and provenance, de-identification of any dataset before it leaves a secure environment, and, where the dataset involves human subjects, documented ethical approval (from an institutional Health Research Ethics Committee) and a data use agreement where the data originated from a partner institution. These practices are inseparable from the technical skills taught in the rest of this book — a beautifully fitted model built on an unversioned, undocumented dataset is not a reproducible piece of science.

Worked Example

Multiple Imputation in Practice

Consider an adolescent health cohort with a depressive symptom score (PHQ-9-type instrument) recorded at baseline and follow-

up, where 15% of follow-up scores are missing, and exploratory analysis (Chapter 5) suggests that missingness is more common among adolescents with high baseline scores and low household food security — a pattern consistent with MAR (missingness explained by observed baseline variables), which multiple imputation can address; if missingness were additionally related to the unobserved follow-up score itself even after conditioning on these variables (e.g., the most severely affected adolescents specifically avoiding follow-up because of their symptoms), the true mechanism would be MNAR, and no amount of standard multiple imputation would fully remove the bias — a sensitivity analysis under plausible MNAR scenarios would then be warranted, a topic returned to briefly in Chapter 9.

R example

```
# R: multiple imputation with mice for a longitudinal adolescent cohort
library(mice)

# Predictors: baseline_score, food_security, sex, school_attendance
imp <- mice(cohort_data, m = 20, method = "pmm", seed = 2024,
            printFlag = FALSE)

# Fit the model of interest on each imputed dataset and pool results
fit <- with(imp, lm(followup_score ~ baseline_score + food_security + sex))
pooled <- pool(fit)
summary(pooled) # pooled coefficients with imputation-adjusted standard errors
```

Additional Cleaning Patterns in Python

The same cleaning discipline introduced in Section 4.4 applies equally in Python, and analysts moving between languages should be able to translate the underlying logic (not just the syntax) fluently. The example below adds a fuzzy near-duplicate check — useful when facility names have been typed inconsistently across reporting periods (e.g., ‘Eleyele PHC’ versus ‘Eleyele Primary Health Centre’).

PYTHON example

```
# Python: cleaning workflow with a fuzzy duplicate check on facility names
import pandas as pd
from rapidfuzz import fuzz

df = pd.read_csv("hmis_malaria_monthly.csv")
df = df.drop_duplicates(subset=["facility_id", "report_month"])
df = df[(df["tests_performed"] >= df["confirmed_positive"]) &
        (df["tests_performed"] >= 0) & (df["confirmed_positive"] >= 0)]
df["tpr"] = df["confirmed_positive"] / df["tests_performed"]

# Flag facility name pairs that are highly similar but not identical (possible re-keying)
names = df["facility_name"].unique()
near_dupes = [(a, b, fuzz.ratio(a, b)) for i, a in enumerate(names)
               for b in names[i+1:] if fuzz.ratio(a, b) > 90 and a != b]
print(f"Potential near-duplicate facility names found: {len(near_dupes)}")
```

Documenting a Dataset**The Data Dictionary**

A data dictionary — a structured document listing every variable in a dataset along with its definition, units, allowed values or range, and source — is one of the simplest and most under-used tools in applied health data science. For a facility-month malaria dataset, a minimal data dictionary entry for the ‘tests_performed’ variable would specify: definition (‘total RDT and microscopy tests performed for suspected malaria in the reporting month’), type (integer, non-negative), expected range (0 to roughly the facility’s total monthly outpatient volume), and source (‘HMIS monthly summary form, section 4a’). Producing this document at the start of a project — rather than reconstructing variable definitions from memory months later when a reviewer asks a clarifying question — is a small investment that pays for itself many times over across the life of a project, and it is often a required deliverable for datasets that will eventually be archived or shared with a partner institution.

R example

```
# R: auto-generating a skeleton data dictionary from a data frame's structure
library(dplyr)

skeleton_dictionary <- data.frame(
  variable = names(malaria_clean),
  type = sapply(malaria_clean, class),
  n_missing = sapply(malaria_clean, function(x) sum(is.na(x))),
  example_value = sapply(malaria_clean, function(x) as.character(x[1]))
)
write.csv(skeleton_dictionary, "data_dictionary_skeleton.csv", row.names = FALSE)
# The analyst then fills in the 'definition', 'units', and 'source' columns by hand.
```

Extended Case Study**Reconstructing a Cleaning Decision Trail After the Fact**

A situation this chapter's reproducibility discipline is specifically designed to prevent, but which nonetheless occurs often enough in practice to be worth examining directly: a research team is asked, eighteen months after a paper's publication, to explain why their reported sample size ($n=1,842$) differs from the raw dataset's row count ($n=2,003$) held by the partner institution. Because the original cleaning was performed through a combination of manual spreadsheet edits and an undocumented series of ad-hoc filters applied directly in a statistical software's point-and-click interface, no member of the original team can now reconstruct exactly which 161 records were excluded, or why, from memory alone eighteen months later — a serious problem, since a journal or funder inquiry into a discrepancy of this kind can, in the worst case, call the entire paper's findings into question even if the original exclusions were entirely justified.

Contrast this with a hypothetical team that instead followed the scripted, auditable workflow argued for in Section 4.4: their cleaning script, preserved in version control (Chapter 6) alongside a plain-language commit history, shows exactly four exclusion steps applied in sequence — 42 exact-duplicate records removed, 38 records with an implausible test-to-positive ratio flagged and

excluded, 56 records missing the primary outcome variable excluded under a pre-specified complete-case policy, and 25 records from a single facility excluded because that facility's data collection instrument was later found to have used a non-standard case definition for part of the study period. This team can reproduce the exact 1,842 figure in minutes, can show the precise justification for each of the four exclusion categories, and can even quickly re-run the analysis under an alternative exclusion policy (for example, imputing rather than excluding the 56 missing-outcome records, per Chapter 4's multiple imputation guidance) if a reviewer specifically asks for this as a robustness check — a request that would have been effectively impossible to fulfil for the first team without redoing the entire cleaning process from scratch, this time under much greater time pressure and scrutiny.

A Short Glossary of Cleaning-Stage Terminology

- Listwise deletion: removing an entire record from analysis because at least one of its variables is missing, regardless of whether other variables in that record are complete.
- Winsorising: capping extreme values at a specified percentile (e.g., the 1st and 99th) rather than removing them outright, used to limit the influence of implausible outliers while retaining the record.
- Deduplication key: the combination of variables (e.g., patient ID and visit date) used to identify whether two records refer to the same underlying observation.
- Data lineage: the documented trail showing how a specific value in a cleaned dataset was derived from the original raw source, including every transformation applied along the way.
- Hot-deck imputation: filling a missing value using an observed value from a similar ('donor') record in the same

dataset, an older alternative to the multiple imputation methods emphasised in Section 4.6.

Metadata

Documenting the Dataset Beyond Its Variables

Beyond the variable-level data dictionary discussed in Section 4.8, a complete dataset documentation package should include dataset-level metadata: who collected the data and under what ethical approval, the exact date range and geographic scope covered, the sampling or census methodology used, the software and version used for any data collection tool, and a plain-language statement of any known, unresolved data quality concerns at the time of the dataset’s release or handover. This metadata is what allows a dataset to be correctly used, years later, by someone who was not part of the original data collection team — without it, even a perfectly clean, well-structured dataset can be silently misapplied to a question or population it was never actually designed to answer, since the misapplication may not be visible from the data’s structure alone and requires exactly this kind of contextual documentation to prevent.

When to Stop Cleaning

Diminishing Returns and the Risk of Overcorrection

Data cleaning, like any other analytic activity, is subject to diminishing returns, and this chapter closes with a caution rarely stated explicitly in introductory treatments: it is possible to over-clean a dataset, applying an ever-more-aggressive sequence of plausibility filters and imputation adjustments in pursuit of a ‘perfectly clean’ dataset that no longer resembles the real, imperfect process that generated it. A useful practical discipline is to distinguish clearly, in the cleaning log recommended throughout this chapter, between corrections that fix unambiguous errors (a negative test count, a duplicate record) and adjustments

that reflect a judgement call about plausibility (an unusually high but not impossible TPR); the former should be applied confidently, while the latter should be applied conservatively, documented explicitly, and, where feasible, tested for sensitivity by re-running the key analysis both with and without the judgement-call adjustment.

Automating Data Quality Checks with Validation Rules

Beyond the one-off cleaning scripts demonstrated earlier in this chapter, a data system that receives new data on an ongoing basis (a monthly HMIS upload, a continuously updated case-based surveillance feed) benefits from automated validation rules that run on every new data submission, immediately flagging an implausible value (a negative count, a TPR outside 0-100%) or a sudden implausible jump relative to the facility's own recent history, before the flawed data ever enters a shared analytic dataset. DHIS2 itself supports configurable validation rules of exactly this kind at the point of data entry, and building the habit of defining validation rules alongside a new indicator — rather than relying entirely on downstream cleaning scripts to catch problems after the fact — shifts data quality assurance from a reactive to a preventive discipline.

PYTHON example

```
# Python: a simple automated validation rule applied to each new monthly submission
def validate_submission(row, facility_history):
    errors = []
    if row['confirmed_positive'] > row['tests_performed']:
        errors.append('positive count exceeds tests performed')
    if row['tests_performed'] < 0 or row['confirmed_positive'] < 0:
        errors.append('negative count submitted')
    recent_mean = facility_history['tests_performed'].tail(6).mean()
    if recent_mean > 0 and row['tests_performed'] > 5 * recent_mean:
        errors.append('tests_performed is a 5x jump versus recent 6-month average')
    return errors
```

A Production-Grade Cleaning Function with Structured Logging

Sections 4.4 and 4.7 print counts of rows affected by each cleaning step, which is adequate for an interactive script but insufficient for a pipeline that runs unattended, where a persistent, timestamped log is needed to audit exactly what happened during each run. Python's built-in 'logging' module (or R's 'logger' package) replaces ad hoc 'print()'/'cat()' statements with structured, leveled log messages that can be routed to a file, filtered by severity, and retained across many pipeline runs — turning the cleaning log recommended throughout Chapter 4 from a one-off console readout into a genuine, persistent audit trail.

PYTHON example

```
# Python: a cleaning function with structured logging instead of print statements
import logging

logging.basicConfig(filename="cleaning_log.txt", level=logging.INFO,
                    format="%(asctime)s %(levelname)s %(message)s")

def clean_malaria_dataset(df):
    n_before = len(df)
    df = df.drop_duplicates(subset=["facility_id", "report_month"])
    logging.info(f"Dropped {n_before - len(df)} duplicate facility-month rows")

    n_before = len(df)
    df = df[(df["tests_performed"] >= df["confirmed_positive"]) & (df["tests_performed"] >=
0)]
    logging.info(f"Dropped {n_before - len(df)} internally inconsistent rows")

    df = df.assign(tpr=df["confirmed_positive"] / df["tests_performed"])
    n_flagged = int((df["tpr"] > 0.95).sum())
    logging.warning(f"{n_flagged} facility-months flagged with TPR > 95% for manual review")
    return df
```

Chapter Summary

- Recognising cross-sectional, longitudinal, and hierarchical data structures determines which analytic methods are valid.

- Missingness mechanisms (MCAR, MAR, MNAR) have different implications; MNAR is common in health cohort/programme data and complete-case analysis can be badly biased.
- Multiple imputation is the modern standard for handling missing data under MAR; MNAR requires sensitivity analysis rather than standard imputation alone.
- A systematic, scripted (not manual) data-cleaning workflow with an audit trail is essential for reproducibility and defensibility.
- Data governance — versioning, data dictionaries, de-identification, and ethical approval — is part of data science practice, not a separate administrative burden.
- Fuzzy matching helps detect near-duplicate records (e.g., inconsistently typed facility names) that exact-match duplicate checks would miss.

Mini-Project

Mini-Project 4: End-to-End Cleaning Pipeline with an Audit Trail

Build a complete, scripted cleaning pipeline for a facility-month dataset (real or simulated) that logs every decision it makes.

Step 1. Simulate or obtain a facility-month dataset with at least 200 rows, deliberately including a few duplicate rows, a few implausible values, and 10-15% missing values in one numeric column.

Step 2. Write a script (R or Python) that performs, in order: duplicate removal, plausibility filtering, and missing-value flagging — printing a count of rows affected at each step.

Step 3. Apply multiple imputation (Section 4.6) to the flagged missing values and compare the imputed mean to the complete-case mean.

Step 4. Write a two-paragraph 'data cleaning report' summarising what was done and why, suitable for inclusion as a methods appendix in a thesis or technical report.

Step 5. State explicitly which missingness mechanism (MCAR/MAR/MNAR) you are assuming, and justify it from what you know (or assumed) about the data.

Deliverable: *A complete, runnable cleaning script, its printed audit-trail output, and a short written cleaning report.*

End-of-Chapter Exercises

1. Classify each of the following as MCAR, MAR, or MNAR, with justification: (a) a laboratory machine breakdown

causes random missing test results for one week; (b) sicker patients are more likely to miss a follow-up visit; (c) younger respondents are more likely to skip an income question, but income itself doesn't predict skipping once age is accounted for.

2. Write out (in pseudocode or R/Python) a cleaning check that would have caught a 'pregnant male' record in a health survey dataset.
3. Explain why LOCF (last observation carried forward) can bias a longitudinal analysis of a deteriorating clinical condition.
4. In the worked example of Section 4.6, explain in your own words why a sensitivity analysis would be needed if the missingness mechanism were MNAR rather than MAR.
5. Using the R code in Section 4.6 as a template, describe (in words) what would change in the 'mice()' call if the outcome variable itself (not just the predictors) had missing values in some imputed datasets but not others.
6. Run the Python fuzzy-matching code in Section 4.7 (or describe its logic if you cannot run it) on a list of five deliberately misspelled facility names you invent, and report which pairs are flagged.
7. A supervisor asks why you spent two extra days building a scripted cleaning pipeline instead of 'just fixing it in Excel.' Write a two- to three-sentence response drawing on the reproducibility arguments of Section 4.4.
8. Using Section 4.8, draft three rows of a data dictionary (variable, definition, type, allowed range) for a dataset you work with regularly.
9. Explain why an auto-generated data dictionary skeleton (Section 4.8's R code) still requires manual completion rather than being used as-is.

10. A dataset arrives with no documentation at all. Rank, in order of priority, the first three things you would do before beginning any analysis.
11. In the Section 4.9 case study, explain what specific piece of information the second (reproducible) team could produce in minutes that the first team could not reconstruct at all.
12. Why is ‘re-running the analysis under an alternative exclusion policy’ a valuable capability for responding to a journal reviewer’s request, beyond just defending the original figure?
13. Describe, in your own words, why manual point-and-click filtering in statistical software creates the same reproducibility risk as manual spreadsheet editing, even though no spreadsheet is involved.
14. If you inherited the first (non-reproducible) team’s dataset and raw files today, what would be your first practical step toward reconstructing a defensible cleaning trail?
15. Explain why ‘we are confident the original exclusions were reasonable’ is not, by itself, a sufficient response to a reviewer’s request to see the exact cleaning steps.
16. Using Section 4.10, explain the difference between listwise deletion and winsorising, and give a scenario where each would be more appropriate.
17. Why is ‘data lineage’ (Section 4.10) a stronger standard than simply keeping the raw and cleaned files in the same folder?
18. Explain hot-deck imputation in your own words, and describe one advantage multiple imputation (Section 4.6) has over it.
19. Propose a deduplication key (Section 4.10) appropriate for a facility-month malaria dataset, and explain why you chose those specific variables.

20. Why might winsorising be preferred over deleting outlier records entirely, in a dataset where sample size is already limited?
21. Using Section 4.11, list three pieces of dataset-level metadata that a variable-level data dictionary (Section 4.8) alone would not capture.
22. Why can a perfectly clean, well-structured dataset still be misapplied without proper metadata, according to Section 4.11?
23. Propose a real or plausible scenario where missing dataset-level metadata led (or could lead) to a dataset being used for a population or purpose it was not designed for.
24. Why is ‘known, unresolved data quality concerns at the time of release’ an important item to include in dataset metadata?
25. How would you store and share this dataset-level metadata alongside a dataset in your own work?
26. Summarise, in your own words, why this chapter treats data cleaning as inseparable from data ethics and governance rather than a purely technical step.
27. Explain the common thread linking missing data handling (Section 4.3), duplicate detection (Section 4.4), and metadata documentation (Section 4.11).
28. Why does this chapter insist on scripted rather than manual cleaning, even when manual editing might seem faster for a small dataset?
29. Propose one cleaning habit from this chapter you intend to adopt in your own work, and explain why you chose it.
30. If a colleague asked you to summarise this entire chapter in one sentence, what would you say?
31. Explain the difference between the automated validation rules in Section 4.13 and the one-off cleaning scripts introduced earlier in this chapter.

32. Why is preventing a data quality problem at the point of entry generally preferable to catching it downstream during cleaning?
33. Using the Python code in Section 4.13, describe what additional validation rule you would add for a dataset you work with, and why.
34. What is a potential downside of an overly aggressive automated validation rule (for example, one that rejects too many legitimate submissions)?
35. How does DHIS2's support for configurable validation rules relate to the interoperability standards discussed elsewhere in this book (Section 2.12)?