PH 718 Data Management and Visualization in R

Part 2: Data Management

Zhiyang Zhou (zhou67@uwm.edu, zhiyanggeezhou.github.io)

2026/02/11 21:39:54

Learning objectives

Given a simulated EHR dataset, our task is not to model outcomes, but to make the raw data **ready for further analysis**. By the end of this lecture, we should be able to:

- Identify and fix data type and coding issues
- Systematically handle missing and inconsistent data
- Create reproducible data-cleaning workflows
- Explain and justify data-management decisions

Understanding the data: "What does this dataset represent?"

Data management begins with understanding, not coding.

- What does each variable represent?
- What units are being used?
- Which variables are categorical, ordinal, or continuous?
- ...

You need not only the data but also a **codebook/dictionary** which typically comes with the data.

- Generate an EHR-style dataset and corresponding dictionary with the given prompt at a GenAI platform: "Generate a realistic simulated electronic health records (EHR) dataset for 500 patients with demographic, clinical, behavioral, and mental health variables. Include a binary outcome related to substance use disorder. Introduce realistic missing values in selected variables. Save the dataset in CSV format. Also, please generate the data dictionary and put in an excel spreadsheet."

```r
setwd("c:/PH718")
ehr <- read.csv("sim_ehr_500.csv", stringsAsFactors = FALSE)
dict <- readxl::read_excel("sim_ehr_500_dictionary.xlsx")
str(ehr)
summary(ehr)
```

This comparison often reveals:

- variables stored with incorrect types,
- unexpected missingness,
- values outside documented ranges.

Variable types

Using the dictionary, we identify which variables should be categorical (i.e., factorized). For example, variables of multiple categories should be encoded as factors in R.

```
unique(ehr$race_ethnicity)
ehr$race_ethnicity_f <- factor(ehr$race_ethnicity)
levels(ehr$race_ethnicity_f)
ehr$race_ethnicity_f <- factor(
  ehr$race_ethnicity,
  levels = rev(unique(ehr$race_ethnicity)),
  labels = rev(unique(ehr$race_ethnicity))
)
levels(ehr$race_ethnicity_f)
```

Checking missingness

The dictionary helps us interpret missing values:

- Is missingness expected?
- Does it reflect non-screening or inapplicability?

Instead of immediately deleting rows, we may:

- recode missingness (e.g., substitute 99 for NA),
- check how missing data arise
  - If the missingness looks similar across groups
    - Likely MCAR (Missing Completely At Random)
    - Simple strategies (complete-case analysis) may be reasonable
  - if the missingness differs by the outcome (e.g., `sud_any`)
    - Missingness is informative
    - Dropping missing rows may bias results
    - Could be MAR (Missing At Random; missingness is related to observed data but not to the missingness itself) or even MNAR (Missing Not at Random; missingness is related to the missingness itself)
    - Using, e.g., multiple imputation (MI), instead

```
table(is.na(ehr$bmi), ehr$sud_any)
```

Restrict to complete cases only when necessary; Missingness is part of the data-generating process, not just a nuisance.

```
ehr_bmi_complete = ehr[complete.cases(ehr$bmi), ]
ehr_complete = ehr[complete.cases(ehr), ]
```

Validating values using dictionary rules

The dictionary defines plausible ranges. We turn those assumptions into executable checks.

```
stopifnot(all(ehr$age >= 0 & ehr$age <= 120, na.rm = TRUE))
stopifnot(all(ehr$bmi > 0, na.rm = TRUE))
```

Creating new variables

```
ehr$age_group <- cut(
  ehr$age,
  breaks = c(0, 30, 50, 65, 120),
  labels = c("<30", "30-49", "50-64", "65+"),
  right = FALSE
)
```

The above encoding should be:

- justified (by the need of further analysis),
- documented (in the codebook),
- reproducible.

Extracting sub-population

```
ehr_sud <- ehr[ehr$sud_any == 1, ]
```

The subsetting is supposed to be based on a scientific decision.

Using loops for systematic data checks

Loops allow us to apply rules consistently across rows or variables.

Example: counting missing values by variable.

```
na_count <- rep(NA, ncol(ehr))
names(na_count) <- names(ehr)
for (j in 1:ncol(ehr)) {
  na_count[j] <- sum(is.na(ehr[[j]]))
}
sort(na_count, decreasing = TRUE)
```

Building a reproducible data-management pipeline

Instead of scattered commands, we encapsulate decisions in a function.

```
clean_ehr <- function(df) {
  df$sex    <- factor(df$sex)
  df$sud_any <- factor(df$sud_any, c(0,1), c("No SUD","SUD"))
  df$age_group <- cut(df$age,
                      breaks = c(0,30,50,65,120),
                      labels = c("<30","30-49","50-64","65+"),
                      right = FALSE)
  df
}
ehr_clean <- clean_ehr(ehr)
```

This makes the workflow:

- reproducible,
- auditable,
- reusable.

Save cleaned data with documentation

```
write.csv(ehr_clean, "sim_ehr_500_cleaned.csv", row.names = FALSE)
```

At this point:

- raw data are untouched,
- cleaning steps are explicit.