Module 3: Data cleaning in health contexts

Data analysis with Python for health specialists

Yaé Ulrich Gaba

2026

AIRINA Labs

Why data cleaning matters in health

Every health dataset has problems:

- Lab values entered with wrong units
- Patients appearing twice under different IDs
- Dates recorded as `03/04/2023` — March 4 or April 3?
- Blood glucose of 5000 mg/dL (error) vs. 500 mg/dL (real emergency)

Data cleaning is not optional pre-processing. It is a **clinical reasoning task**.

Real impact
In a 2019 EHR study, 18% of lab values had quality issues. Cleaning decisions changed CKD prevalence estimates by over 3 percentage points.

Missing data: the three mechanisms

Type	Example	If you drop
MCAR	Lab sample hemolyzed in transport	No bias, less power
MAR	HbA1c not ordered for non-diabetics	Bias if you ignore indication
MNAR	Sick patients skip follow-up	Underestimate severity

The mechanism determines your strategy. MNAR has no statistical fix — only domain knowledge.

Detecting missing values

```
import pandas as pd
import numpy as np

# Count missing per column
print(df.isnull().sum())

# Percentage missing
print((df.isnull().sum() / len(df) * 100).round(1))

# Rows with any missing value
print(f"Rows with missing data: "
      f"{df.isnull().any(axis=1).sum()} / {len(df)}")
```

Visual patterns

Use *missingno* for instant visual summaries: *msno.matrix(df)*, *msno.bar(df)*, *msno.heatmap(df)*.

Imputation strategies

Continuous variables:

```
# Median (robust to outliers)
df["systolic_bp"] =
↪  df["systolic_bp"]\
    ↪  .fillna(df["systolic_bp"].median())

# Group-specific median
df["systolic_bp"] = df.groupby(
    ↪  "age_group")["systolic_bp"]\
    .transform(
    lambda x:
    ↪  x.fillna(x.median()))
```

Categorical variables:

```
# Mode imputation
df["sex"] = df["sex"].fillna(
    df["sex"].mode()[0])

# KNN imputation (multivariate)
from sklearn.impute import
↪  KNNImputer
imputer =
↪  KNNImputer(n_neighbors=5)
df[num_cols] =
↪  imputer.fit_transform(
    df[num_cols])
```

Rule: use median for skewed lab values, **mean** for symmetric data, **mode** for categories.

Handling duplicates

```
# Exact duplicates (all columns identical)
print(f"Exact duplicates: {patients.duplicated().sum()}")

# Remove exact duplicates
patients_clean = patients.drop_duplicates()

# Keep only first visit per patient
first_visits = patients.drop_duplicates(
    subset="patient_id", keep="first")
```

Clinical context matters

Same patient, same visit, identical values → true duplicate. Same patient, different dates → longitudinal data — keep all visits!

Standardizing ICD codes and dates

```
# ICD codes: uppercase, strip, add dot
cleaned = (diagnoses.str.strip().str.upper()
           .str.replace(r"^([A-Z]\d{2})(\d+)$",
                        r"\1.\2", regex=True))

# Dates: parse mixed formats to ISO
parsed = pd.to_datetime(dates, format="mixed",
                        dayfirst=False)
print(parsed.dt.strftime("%Y-%m-%d"))
```

Date ambiguity

`03/04/2023` = March 4 (US) or April 3 (Europe). Always check the source. Use `dayfirst=True` for European dates.

Unit conversions

Analyte	US	SI	Convert
Glucose	mg/dL	mmol/L	÷ 18.018
Cholesterol	mg/dL	mmol/L	÷ 38.67
Creatinine	mg/dL	µmol/L	× 88.42

```
# Values > 30 are likely in mg/dL
def standardize_glucose(value):
    if value > 30:
        return value / 18.018  # mg/dL -> mmol/L
    return value  # already mmol/L
```

Data validation

```python
# Range checks
ranges = {"age": (0, 120), "systolic_bp": (60, 300),
          "hba1c": (2.0, 20.0), "bmi": (8, 80)}

for col, (low, high) in ranges.items():
    out = df[(df[col] < low) | (df[col] > high)]
    if len(out) > 0:
        print(f"WARNING: {len(out)} values in "
              f"'{col}' outside [{low}, {high}]")

# Cross-field: diastolic < systolic
bp_errors = df[df["diastolic_bp"] >= df["systolic_bp"]]
```

Not every “impossible” value is an error. HR of 250 bpm = rare but real (SVT). Glucose of 800 mg/dL = DKA.

Automated quality report

```python
def data_quality_report(df):
    report = []
    for col in df.columns:
        n_miss = df[col].isnull().sum()
        row = {"column": col,
               "dtype": str(df[col].dtype),
               "n_missing": n_miss,
               "pct_missing": round(n_miss/len(df)*100,1),
               "n_unique": df[col].nunique()}
        if df[col].dtype in ["float64", "int64"]:
            row["min"] = df[col].min()
            row["max"] = df[col].max()
        report.append(row)
    return pd.DataFrame(report)
```

Run this before and after cleaning. Document every decision.

What you can do after this module

1. Detect missing values and choose the right imputation strategy
2. Distinguish MCAR, MAR, and MNAR and explain why it matters
3. Remove true duplicates without losing longitudinal data
4. Standardize ICD codes, dates, units, and text fields
5. Validate data against clinically plausible ranges
6. Generate before/after quality reports to document your pipeline

Next: Module 4 — Descriptive statistics and epidemiological measures

Exercises (Hour 3)

1. **Quality report:** Generate a data quality report for the provided messy dataset
2. **Fix sex and ranges:** Standardize the sex column; set impossible ages and BPs to NaN
3. **Unit conversion:** Convert mixed glucose (mg/dL vs. mmol/L) and HbA1c (IFCC to NGSP)
4. **Mini-project:** Apply the complete cleaning pipeline, compare before/after reports, save cleaned CSV