Data Acquisition and Cleaning: Part II

July 8th, 2025

Introduction on data cleaning

- **Data cleaning** is the process of identifying and correcting problems in your data to make it accurate, complete, and ready for analysis. It's a critical step in the data science lifecycle, often taking **50–80%** of the time in real-world data projects.

Why Does Raw Data Need Cleaning?

Real-world data is rarely perfect. It often comes from:

- Multiple sources (e.g., web scraping, sensors, surveys)
- Human input (prone to typos, omissions)
- Incomplete or corrupted systems
- These imperfections can introduce **bias**, **errors**, and **unreliable results** if not addressed.

Objectives of Data Cleaning

The goal is to ensure that your dataset:

- Is **accurate** (reflects reality)
- Is **complete** (no important gaps)
- Is **consistent** (follows expected formats and definitions)
- Is **relevant** (only includes useful information)
- Is **well-formatted** (ready for analysis or modeling)

Problem	Example	Risk
Missing values	Empty cells in survey responses	Incomplete analysis
Duplicated records	Same transaction logged twice	Skewed statistics
Inconsistent labels	“Yes”, “yes”, “Y” for one variable	Misclassification
Wrong data types	"2025-07-05" stored as a string, not a date	Date-based sorting fails
Typos and entry errors	“Calfornia” instead of “California”	Inaccurate groupings
Outliers	Income of $100M in a small-town dataset	Misleading averages

Practical method on data cleaning

1. **Handling Missing Data**
2. **Removing Duplicates**
3. **Fixing Data Types**
4. **Standardizing Text and Categorical Data**
5. **Removing or Treating Outliers**
6. **Parsing and Splitting Fields**
7. **Correcting Structural Errors**
8. **Validating and Constraining Values**
9. **Encoding Categorical Variables**
10. **Documenting the Cleaning Process**

1. Handling Missing Data

- Missing data occur when no value is stored for a variable in an observation. This is extremely common in real-world datasets.

Examples:

- Survey questions left blank
- Sensor failures
- Mismatched merges
- Data entry errors

Type	Description	Example
MCAR (Completely at Random)	Missing independently of any values (observed or not)	Survey skipped randomly
MAR (At Random)	Missing depends on **observed** variables	Income missing more for students
MNAR (Not at Random)	Missing depends on **unobserved** values	People with high income don't report it

🛠️ Methods to Handle Missing Data

A. Remove Missing Values

- **When to use:** Small proportion of missing values; MCAR
- **How:**
  - Drop rows: df.dropna()
  - Drop columns: df.dropna(axis=1)

B. Imputation

- **Mean/Median/Mode Imputation**
  - Good for numeric or categorical features
  - Assumes missing is at random (MCAR or MAR)
- **Forward/Backward Fill**
  - Good for time series
- **Predictive Imputation**
  - Use models (e.g., KNN, regression) to predict missing values
- **Custom Value or Flag**
  - E.g., fill with -999, or add indicator variable

Example

	id	age	income	gender
1	1	25.0	50000.0	male
2	2	30.0	60000.0	female
3	3		55000.0	
4	4	22.0		female
5	5	28.0	52000.0	male
6	6		58000.0	female
7	7	35.0		male
8	8	29.0	62000.0	
9	9	40.0	70000.0	female
10	10		65000.0	male

2. Removing Duplicates

- **Duplicates** are rows that appear more than once in a dataset, either fully or partially.

Type	Example	Detection Strategy
Full duplicate	All values match exactly	df.duplicated()
Partial duplicate	Some columns match (e.g., same name and birthdate)	df.duplicated(subset=['name', 'dob'])
Near duplicate	Typo differences (e.g., "Jon Smith" vs "John Smith")	Fuzzy matching (advanced)

- Find and Drop Full Duplicates
- Drop Based on Specific Columns
- Keep First or Last Duplicate
- Flag Instead of Dropping
- For large datasets, dropping duplicates can save memory and computation.

Example

	id	name	email	age	city	
1	101	Jon Smith	jon.smith@mail.com	34	New York	
2	102	John Smith	john.smith@mail.com	34	New York	
3	103	J. Smith	smith.j@mail.com	34	New York	
4	104	Alice B.	aliceb@mail.com	29	Los Angeles	
5	105	Alice Brown	alice.brown@mail.com	29	Los Angeles	
6	106	Bob	bob@mail.com	45	Chicago	
7	107	Bobby	bob@mail.com	45	Chicago	

3. Fixing Data Types

Each column in a dataset has a **data type**, such as:

- **Numeric**: integers (int), floats (float)
- **Text**: strings (object or str)
- **Date/time**: datetime64
- **Boolean**: True/False
- **Categorical**: fixed set of labels

Issue	Fix Method
Dates stored as strings	pd.to_datetime()
Numbers stored as strings	pd.to_numeric()
Booleans as 'Yes'/'No'	Map or convert
Categorical text not optimized	Use astype('category')
Mixed types in a column	Clean first, then convert

Example

	id	age	price	signup_date	subscribed	city
1	1	25	100.0	2023-01-15	Yes	New York
2	2	30	200.5	15-02-2023	No	Chicago
3	3	NaN	invalid	March 1, 2023	Yes	Chicago
4	4	40	300	not a date	No	Los Angeles

4. Standardizing Text and Categorical Data

Problem Type	Example
Case inconsistencies	"Yes", "yes", "YES"
Extra whitespace	" New York "
Synonyms/abbreviations	"CA", "California", "Calif."
Typos	"Femail" vs "Female"
Mixed language or symbols	"NYC" vs "New York 🗽 "

- Make all text lowercase or uppercase
- Remove leading/trailing whitespace
- Replace known values (synonyms, typos)
- Use regex to clean up patterns
- Map values to clean categories
- Convert to categorical data type
- Build a **mapping dictionary** for known corrections
- Use value_counts() before and after to confirm cleaning.

Example

	state	gender	city	response
1	CA	M	New York	Yes
2	ca	F	new york	no
3	California	Femail	NEW YORK	yes
4	Calif.	male	Los Angeles	No
5	Cali	F	los angeles	YES
6	california	f	L.A.	nO
7	ca	FEMALE	la	yes

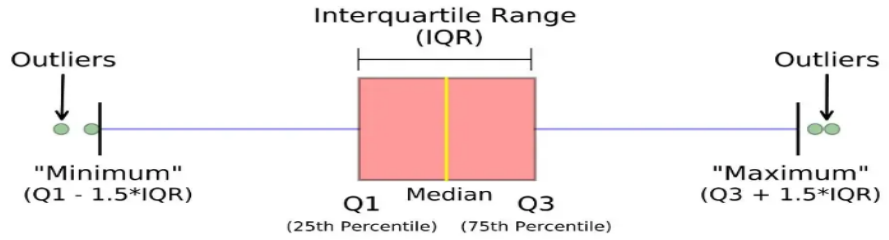
5. Removing or Treating Outliers

Outliers are data points that deviate significantly from the rest of the dataset. They can be:

- **Valid rare events** (e.g., a billionaire's income)
- **Errors** (e.g., mistyped “4000” instead of “40.00”)
- **Extreme but plausible values** that distort statistics or models

Detection Techniques

- **Visual Inspection**: Boxplots, Histogram, Scatter plot
- **Statistical Rules**: 1. Z-score: how many standard deviations a value is from the mean; 2. IQR rule: values outside 1.5×IQR below Q1 or above Q3
- **Domain Knowledge**



Treatment of outliers

- **Remove Outliers**: Clear data entry errors (e.g., height = 1200 cm) or Outliers aren't meaningful for your question
- **Winsorization (Capping):** Replace extreme values with threshold values (e.g., 1st and 99th percentiles)
- **Transformation**: Apply functions like log, square root, or Box-Cox

  Outliers are real but you want to stabilize variance
- **Replace with Imputed Values**: Replace outliers with mean, median, or model-predicted values

Always visualize before and after treatment; Never drop outliers blindly — check context

6. Parsing and Splitting Fields

- Parsing and splitting involve **extracting structured information** from a **single column** that contains multiple pieces of data — or **reformatting text** so it's easier to analyze.

Task	Example Input	Desired Output
Split full name	"Alice Brown"	"Alice", "Brown"
Separate date and time	"2024-01-01 13:45:00"	"2024-01-01", "13:45:00"
Extract year or domain from email	"john@example.com"	"john", "example.com"
Parse address components	"123 Main St, NY, 10001"	"123 Main St", "NY", "10001"
Extract numbers from strings	"$1,250.00"	1250.00
Detect structured patterns	"UCI123456"	Prefix: "UCI", ID: "123456"

7. Correcting Structural Errors

- Structural errors are mistakes in the **layout or organization** of the dataset that affect its usability.

These errors can include:

- Wrong or inconsistent column names
- Columns used as rows, or vice versa
- Headers stored as data
- Extra index columns from file exports (e.g., from Excel)
- Misplaced rows, like summaries or notes mixed into data

Good practice

•Use df.head() and df.tail() to check for misplaced headers or summaries.
•Automate column renaming for consistency across datasets.
•Aim for **tidy data**: each column is a variable, each row is an observation.

8. Validating and Constraining Values

Validation means checking whether each data value:

- Falls within a **reasonable range**
- Matches an **expected format**
- Respects **logical constraints** between variables
- It's like asking: “Does this value make sense?”

Type	Example	Validation Rule
Range checks	Age must be between 0 and 120	0 <= age <= 120
Format checks	Email should contain @	.str.contains('@')
Type checks	Date should parse to datetime	pd.to_datetime()
Enum/Value checks	Gender must be "male" or "female"	.isin(['male', 'female'])
Cross-field logic	End date ≥ start date	end_date >= start_date

9. Encoding Categorical Variables

- Many machine learning models require **numerical input**, so we must convert **categorical variables** (text labels) into **numbers**.

Method	Use Case	Example Output
One-Hot Encoding	Nominal (unordered) categories	red → [1, 0, 0], blue → [0, 1, 0]
Label Encoding	Ordinal categories or tree-based models	low → 0, medium → 1, high → 2
Ordinal Mapping	Explicitly ordered categories	You define the order manually
Binary Encoding	Many categories, space-efficient	Encodes as binary digits
Frequency Encoding	Based on count of category occurrences	Category = number of times seen

Example

Color	color_blue	color_green	color_red
red	0	0	1
blue	1	0	0
green	0	1	0
blue	1	0	0
red	0	0	1

Gender	gender_code
male	1
female	0
female	0
male	1

Example

Education	education_encoded
high school	0
bachelor	1
master	2
PhD	3
bachelor	1

ZIP Code	zip_encoded (frequency)
90001	2
90002	1
90003	3
90001	2
90003	3

10. Documenting the Cleaning Process

Documentation makes your data cleaning process:

- **Reproducible** — so you or others can repeat it later
- **Transparent** — shows how data was changed
- **Shareable** — helps teams understand what “clean” means
- **Debuggable** — lets you track where issues were introduced

Step	Example Description
Original issues	“Age column had values below 0 and above 120”
Cleaning methods used	“Removed duplicates based on email column”
Values replaced or removed	“Filled missing gender with ‘unknown’”
Variables transformed	“Applied log transformation to income”
Type conversions	“Converted signup_date to datetime”
New columns added	“Created binary is_outlier flag using z-score > 3”
Rows excluded	“Dropped 3 rows with invalid email format”

Good practice

- Inline Comments
- Cleaning Logs
- Markdown Cells
- Keep a **versioned copy** of raw data
- Separate **cleaning** from **analysis**
- Annotate all steps when sharing with others
- Make it understandable to someone seeing the data for the first time