

Data Cleaning Guidelines for Python Workflows

Clean, well-structured data is essential for reliable, reproducible analyses and machine learning pipelines.

This guide provides best practices, examples, and practical tips to help your team prepare data that can be seamlessly ingested and processed in Python.

Data Formatting Best Practices

Objective: Ensure datasets are machine-readable, consistent, and free of structural problems.

- **Use machine-readable formats.**

Save your data as CSV, JSON, GeoJSON, or shapefiles. Avoid using Excel files with multiple sheets, merged cells, or special formatting. These make data harder to process automatically.

- **Keep a tabular structure.**

Each row should represent one observation; each column should represent one variable. Do not include units, comments, or calculations inside the data cells.

- **Use consistent naming.**

Adopt `snake_case` (lowercase words joined by underscores), and use only ASCII letters and numbers.

Example:

`project_name` , `energy_consumption_kwh` , `building_height_m` .

- **Single data type per column.**

Each column should contain only one kind of data: numbers should be stored as numbers, dates as ISO strings (`YYYY-MM-DD`), and text as plain strings.

- **Encode booleans and categories clearly.**

- For true/false (boolean) columns, use `0` for false and `1` for true.
- For multiple categories, use one-hot encoding to create separate columns.

Example:

city_stockholm	city_malmo	city_gothenburg
1	0	0

- **Encoding and units.**

- Always save CSVs as UTF-8 to preserve special characters like `Malmö`.
- Include units in column names, such as `height_m` for height in metres or `energy_kwh` for energy in kilowatt-hours.

If you cannot read your data easily with one line of Python (like `pd.read_csv('file.csv')`), the formatting needs improvement.

Naming Conventions

Use clear, concise names for columns and files. Stick to lowercase letters, numbers, and underscores (_).

 Bad	 Good
Project Name	project_name
År	year
Energy (kWh)	energy_kwh
1st Floor Area	first_floor_area_m2
Building-Height	building_height_m
Temp °C	temp_c
Address/Location	address_location
Total(SEK)	total_amount_sek
Date of Completion	date_of_completion

Use standard abbreviations like `temp` for temperature. Avoid spaces, hyphens, or special characters in file names.
Example: `energy_data_2024.csv`

Metadata and Documentation

Good metadata ensures others (and your future self) can understand and reuse the data.

- **Create a data dictionary.**

A table describing each column's name, type, units, description, and allowed values.

Example:

column_name	type	units	description	allowed_values
project_name	string		Name of the project	—
year	integer		Year the project was initiated	2019, 2020, 2021...
	float	kWh	Annual energy	0

- **Write a README or metadata file.**

Include:


- Data source and collection method
- Update frequency (e.g., annually, monthly)
- Preprocessing or cleaning steps already applied
- Contact information for questions

Organising Data Files

Keeping data organised saves time and avoids confusion later.

Recommended folder structure:

```
/data/raw/           # Original, untouched data
/data/processed/     # Cleaned and ready-to-use data
/outputs/            # Results, charts, and analysis outputs
/scripts/            # Python scripts used for cleaning and analysis
/docs/               # Documentation, data dictionaries, README files
```

 **Tip:** Never overwrite your raw data. Always save cleaned and processed versions separately.

Data Quality Best Practices

- **Handle missing values properly.**

Use `NaN` for missing numeric values, and empty strings for missing text. Avoid custom codes like `-999` or `"n/a"`.

- **Remove duplicates.**

Check for and remove unintended duplicate rows unless duplicates are meaningful.

- **Validate data types.**

After loading, inspect your data with `df.info()` and `df.dtypes` in pandas to confirm everything is the correct type.

- **Maintain language consistency.**

Use English for headers. For value fields, use UTF-8 encoding to support international characters.

- **No formulas, comments, or styling in raw files.**

Files should contain only data—no Excel formulas, no background colours, no hidden columns.

After loading your data, use ``df.describe(include='all')`` to quickly spot anomalies like unexpected categories or missing values.

Common Problems and Solutions

Problem	Example	Solution
Special characters in column names	År , Energiförbrukning (kWh)	Rename to year , energy_kwh
Multiple values in one cell	"Stockholm; Malmö"	Split into separate columns with 0/1 values (one-hot encoding)
Numbers stored as strings	"42"	Convert: df['col'] = df['col'].astype(int)
Mixed date formats	2022-01-01 , 01/01/2022	Standardise using pd.to_datetime(df['col'])
Custom missing values	-999 , "n/a"	Replace with NaN using df['col'].replace([-999, "n/a"], np.nan)

Handling Sensitive Data

If your data contains personal information (names, addresses, health data, etc.), you must protect privacy.

- Remove direct identifiers where possible.
- Use pseudonyms or anonymised IDs.
- Ensure GDPR compliance if applicable.
- Document any data protection measures taken.

Data Versioning

Data changes over time. It is important to track these changes clearly.

- Save different versions with clear filenames, such as:
 - `energy_data_v1.csv`
 - `energy_data_v2_cleaned.csv`
- Keep a changelog noting major updates or corrections.

Validation Scripts

Writing small scripts early can help automate data checking.

Recommended checks include:

- Missing value summary
- Data type validation
- Duplicate detection
- Allowed range checks (e.g., no negative ages)

Create a simple ``validate_data.py`` script to automate quality checks. Running it regularly prevents problems from building up unnoticed.

Additional Resources and Links

- [Pandas I/O Documentation](#)
- [PEP 8 Naming Conventions](#)
- [Frictionless Data - Data Quality Standards](#)