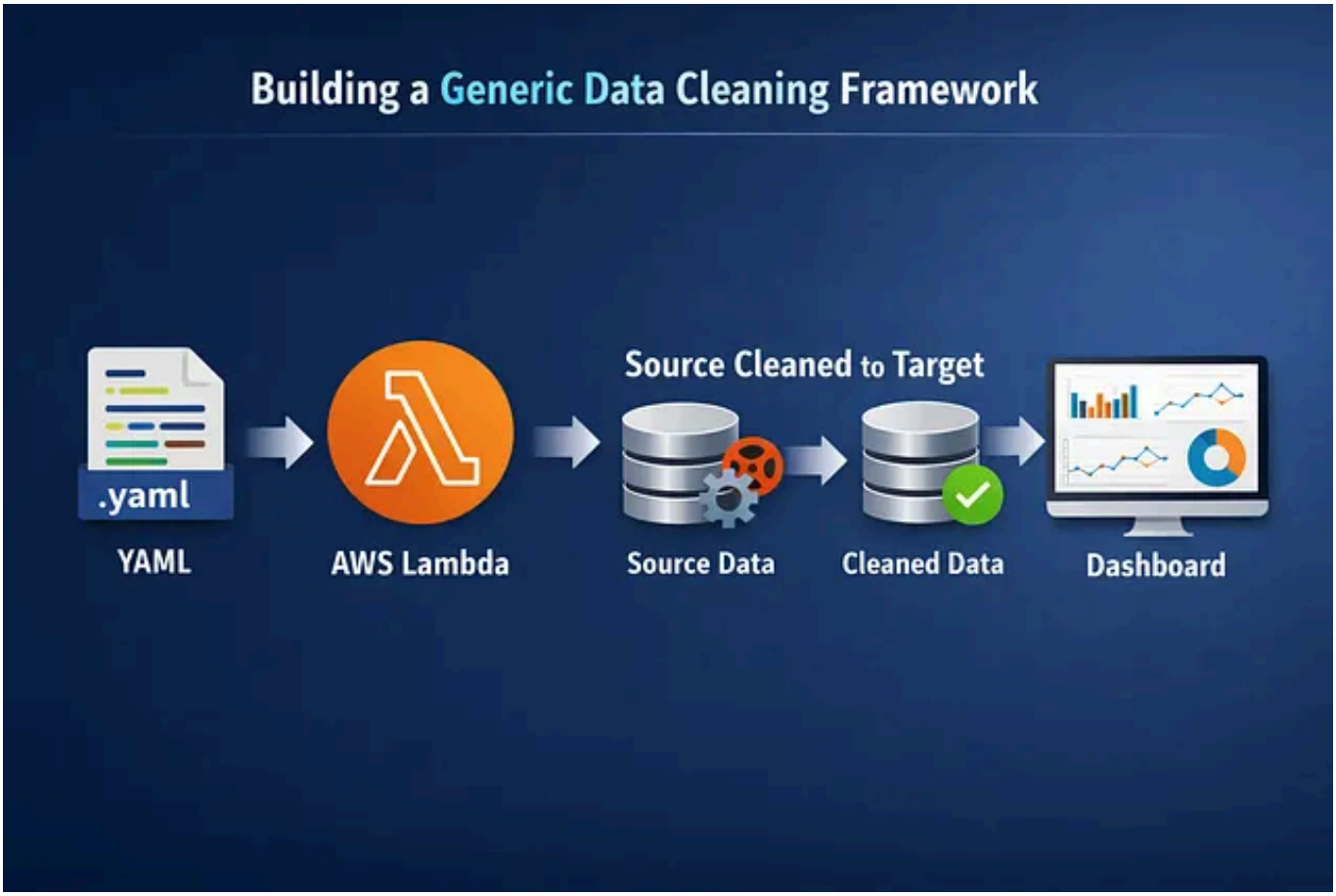

How We Built a Generic Data Cleaning Framework for Every Analytics Team

Anurag Sharma 5 min read · Feb 9, 2026

Hey Data Folks! Clean data is the foundation of every reliable dashboard, model, and decision. Yet most analytics teams spend more time fixing broken fields, mismatched formats, and missing values than generating insights. Without consistent data cleaning, pipelines become fragile, and reports lose trust. To solve this, we designed a generic data cleaning framework that standardizes quality, reduces manual effort, and keeps analytics workflows stable.

Here is how and why it transforms the way teams work with data.

Data cleaning is one of the most repeated — and repeatedly re-implemented — steps across analytics teams. Every project, every dataset, every pipeline starts with the same questions:

- Are there duplicates?
- Are there nulls?
- Are data types correct?
- Are values corrupted or malformed?
- Are formats consistent?

We noticed something important: Our Teams were solving the same cleaning problems again and again — using different scripts, different styles, and different levels of manual efforts.

So we built a generic, reusable data cleaning framework that any team in our company can use — without writing custom code — powered by Python + YAML templates + a simple UI. This post explains on a high level how we designed it and how it works.

The Goal: Standardized, Self-Serve Data Cleaning

Our design goals were simple:

✅ No heavy coding required by end users

✅ Works across file and database sources

✅ Reusable cleaning rules

✅ Transparent outputs and audit report

✅ Serverless scalable engine

✅ Simple UI-driven rule selection

✅ Config-driven execution via YAML

Instead of writing scripts, users define *what* to clean — not *how* to clean it.

Framework Overview

We built a generic Python data cleaning engine driven by:

- YAML configuration templates

- A UI rule selection page
- Pandas-based validation & profiling
- Serverless execution using Lambda
- Structured outputs (clean file + audit report)

The framework supports up to **1 million records per run** — optimized for serverless execution.

At its core, the framework abstracts away the boilerplate of data cleaning into modular, configurable steps. Think of it as a "Lego kit" for data prep: snap together pre-built blocks for deduplication, imputation, type coercion, and more, all without touching a line of code unless you want to tweak the YAML.

The Problem: Why Data Cleaning Sucks And How We Fixed It

Let us be real — data cleaning is not glamorous. It is the unglamorous grind that eats 80% of an analyst's time, according to industry surveys. In our multiple teams, we saw spreadsheets with rogue semicolons masquerading as commas, CSV files bloated with phantom rows, and SQL dumps where dates decided to flip between MM/DD/YYYY and DD/MM/YYYY on a whim.

The vicious cycle?

1. Analyst spots the issue.
2. Write a one-off script in Jupyter.
3. Shares it via MS Teams (or worse, email with leadership in CC).
4. Next analyst ignores it and reinvents the wheel.
5. Repeat until someone burns out.

Our fix? A framework that turns this into a self-serve service. Load your data, pick rules from a dropdown, hit “Run,” and get a cleaned dataset plus a report that says, “Hey, I zapped 47 duplicates and imputed 12% of missing values using median — here’s the before/after.”

How It Works: From YAML to Clean Data in Minutes

Step 1: Ingest Your Data (Files or DBs, No Sweat)

We support CSVs, JSONs, Parquet files, and direct pulls from PostgreSQL, MySQL, or Snowflake. Upload via the UI or point to an S3 bucket/endpoint. The engine uses Pandas under the hood for in-memory processing, but chunks large files to stay lean on resources.

Step 2: Profile and Auto-Detect Issues

Before cleaning, we run a quick profile using Pandas Profiling (now ydata-profiling). This generates a snapshot:

- **Null rates:** e.g., "Column 'email' has 23% missing."
- **Duplicates:** Row-level and value-level checks.
- **Data types:** Flags mismatches like strings in numeric fields.
- **Outliers/Anomalies:** Basic stats with z-score thresholds.

Step 3: Select Rules via UI (Or YAML for Power Users)

Here is the magic: Our rule library is a growing set of YAML-defined templates. Each rule is a declarative block, like:

```yaml
rules:
  - name: remove_duplicates
    columns: ['id', 'email']  # Composite key
    action: drop
    threshold: 0.95  # Similarity score for fuzzy dupes

  - name: impute_nulls
    columns: ['age', 'salary']
    strategy: median  # Or mean, mode, forward_fill
    condition: "age > 0"  # Optional filters

  - name: standardize_dates
    columns: ['created_at']
    format: '%Y-%m-%d'  # Target format
    errors: 'coerce'  # Pandas magic

  - name: validate_emails
    columns: ['email']
    regex: '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
    action: flag_invalid  # Or drop, replace
```

The UI translates this into checkboxes: “Remove duplicates on ID+Email?” “Impute ages with median?” Select, tweak params, and generate the YAML export for repeatability.

Step 4: Execute on Lambda (Scalable and Cheap)

We wrap the cleaning logic in AWS Lambda functions. Why serverless?

- **Auto-scale**: Handles spikes from 10k to 1M rows without provisioning.
- **Cost**: Pennies per run — e.g., $0.01 for a 500k-row CSV.
- **Idempotent**: Rerun with the same config for audits.

The engine processes in batches: Load → Apply rules sequentially → Validate post-clean → Output.

Step 5: Outputs That Tell a Story

You get two artifacts:

1. **Cleaned Dataset**: Exported as CSV/Parquet/JSON, ready for your BI tool.
2. **Audit Report**: A Markdown/PDF summary with:

- Rule application logs (e.g., “Dropped 47rows”).
- Before/after stats tables.
- Visual diffs (e.g., bar charts of null reductions).

Rule	Applied To	Changes Made	Impact
Remove Duplicates	id + email	Dropped 47 rows	Reduced size by 4.2%
Impute Nulls	salary	Filled 128 values (median: $65k)	Null rate: 23% => 0%
Standardize Dates	created_at	Coerced 89 invalid formats	100% parseable

Key Tech Stack: Simple

- **Python 3.10+:** Core logic with Pandas for data ops, Pydantic for YAML validation.
- **AWS Lambda + S3:** Execution and storage.
- **YAML/JSON:** Configs for portability — version them in Git!
- **Logging:** Structured with Loguru, traceable via CloudWatch.

Real-World Wins: Faster Teams, Fewer Headaches

Since rollout:

- **Time Savings:** Cleaning a 200k-row sales dataset? From 4 hours to 15 minutes.
- **Consistency:** All teams use the same ruleset — no more “Wait, why did upstream team drop outliers, but we didn’t?”
- **Adoption:** 70% of analysts self-serve now; devs focus on features, not fixes.
- **Edge Cases:** Handled fuzzy matching (via FuzzyWuzzy) for names/addresses and custom regex rules for domain-specific junk (e.g., Indian PIN codes).

Challenges We Tackled And Lessons Learned

No framework's perfect:

- **Scalability:** Lambda's 15-min timeout bit us early; we added async chunking.
- **Rule Conflicts:** What if imputation happens before dedupe? We enforce logical order in YAML (with warnings).
- **Security:** Row-level access via IAM roles for DB connects.
- **Extensibility:** Open-sourced the YAML schema on GitHub — PRs welcome for new rules like geospatial cleaning.

Data Engineering | Data Cleaning | Framework | Data | Data Analysis

Written by Anurag Sharma

83 followers · 3 following

Edit profile

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!

No responses yet

Anurag Sharma him/he

What are your thoughts?