DATA ENGINEERING

Smart Data Quality and Auto-Healing Pipeline

Sure Mamatha, Anvitha Bandi, A. Swathi Srividya, Akhila Thirichanur, Akshat Tumkur

Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Hyderabad, Telangana, India

Published November 2026

Abstract

The Smart Data Quality and Auto-Healing Pipeline is an automated, configuration-driven system developed to improve the reliability and usability of structured data, particularly CSV datasets. Data may contain missing values, duplicate records, invalid formats, datatype inconsistencies and unusual observations, making manual cleaning time-consuming and difficult to scale. The pipeline combines ingestion, schema validation, profiling, rule-based validation, optional anomaly detection, safe automatic healing, quality scoring, audit metadata and reporting. YAML configuration separates dataset-specific rules from implementation code. The system validates data before and after healing and records unresolved issues rather than claiming every problem has been fixed. FastAPI exposes pipeline services and cleaned-data downloads, while a Streamlit dashboard presents quality metrics and healing outcomes. In the evaluated sample, the overall quality score increased from 77.02 to 94.13 with 61 automated tests passing.

Keywords: data quality; data cleaning; auto-healing; CSV; data profiling; schema validation; Isolation Forest; FastAPI; Streamlit; YAML

1 Introduction

Data-driven applications depend on accurate, complete and consistent input. Structured datasets may contain null values, repeated rows, malformed email or phone fields, unexpected datatypes and other quality problems. Traditional cleaning scripts are often tied to a particular dataset and are difficult to reuse when schemas or rules change.

This project treats data quality as a configurable engineering workflow. Schema rules describe structural expectations, validation rules identify quality violations, and healing rules define transformations considered safe. Profiling and quality scoring make the condition of a dataset measurable, while audit metadata helps users understand what changed and what remains unresolved. The implementation focuses on CSV ingestion and provides a modular foundation for adding validators, healers and data-source connectors. The dashboard presents results as metrics, charts, tables and status indicators.

2 Problem statement

Data cleaning is frequently performed through separate scripts or manual spreadsheet operations. These approaches can be repetitive, difficult to audit and inconsistent across datasets. Validation-only tools identify errors but leave correction to users, while automated repair without re-validation can report success even when invalid records remain. The absence of comparable before-and-after quality metrics also makes it difficult to communicate improvement. A reusable pipeline is therefore required to combine configurable ingestion, schema checks, profiling, rule-based validation, safe

deterministic repair, optional anomaly detection, quality scoring and audit-oriented reporting. Each repair must be followed by validation so that its outcome reflects the actual remaining issues.

3 Objectives

- Provide configurable CSV ingestion and schema validation.
- Profile datasets and expose their quality characteristics.
- Implement reusable validators for missing values, duplicates, regular-expression patterns and datatypes.
- Apply safe, deterministic healing strategies to recoverable issues.
- Optionally detect unusual numeric observations using Isolation Forest.
- Measure data quality before and after healing.
- Maintain audit metadata and readable quality reports.
- Expose pipeline execution and results through FastAPI and Streamlit.

4 Research gap

Existing approaches address different parts of the data-quality problem. Schema validation checks structure but cannot by itself repair malformed values. Profiling summarizes data characteristics but does not correct issues. Rule-based validators identify violations, while automated cleaning reduces manual effort; however, repair policies and traceability remain important. Anomaly detection can flag unusual observations, but an anomaly is not necessarily an error and should not automatically be modified.

The project integrates these capabilities into one configuration-driven workflow. Its plugin-oriented design allows new validators and healers to be introduced independently, and it treats metadata and final validation as core outputs. Healing is classified as SUCCESS, PARTIAL or FAILED based on observed results.

5 Literature review

Rahm and Do [1] survey common data-quality problems and approaches used in data integration and warehousing. Chu et al. [2] provide an overview of data-cleaning techniques, including constraint-based, pattern-based, statistical and machine-learning methods. These studies support combining explicit validation rules with optional data-driven techniques. Liu, Ting and Zhou [3] introduced Isolation Forest for anomaly detection, which is available in scikit-learn [9]. In this project, the method is optional and reports candidate anomalies without directly changing the dataset, since unusual values may be legitimate.

McKinney's work on Python data analysis [4] and the Python and pandas documentation [5, 6] provide relevant foundations for structured-data processing. The reviewed approaches motivate an integrated solution in which schema checks, profiling, validation, controlled repair and quality measurement work together. The project emphasizes not only detecting issues, but making repairs traceable and checking whether they resolved the detected problems.

6 Proposed system flow

The pipeline loads YAML configuration and ingests the configured CSV input. Schema validation checks required columns, expected types, nullability and unexpected columns. The dataset is profiled and passed through rule-based validators. An initial quality score is calculated before supported healing operations are applied. After healing, the dataset is profiled and validated again. The final score and remaining validation issues are compared with initial results. Audit metadata records the run, validation results, healing actions, metrics and execution information. The API returns structured results, and the dashboard translates them into understandable visual summaries.

The main flow is: CSV input → configuration and ingestion → schema validation → profiling and validation → initial score → healing → re-validation → final score, audit and dashboard.

7 Major modules

The layered architecture of the system is shown in Figure 1, and its main modules are as follows:

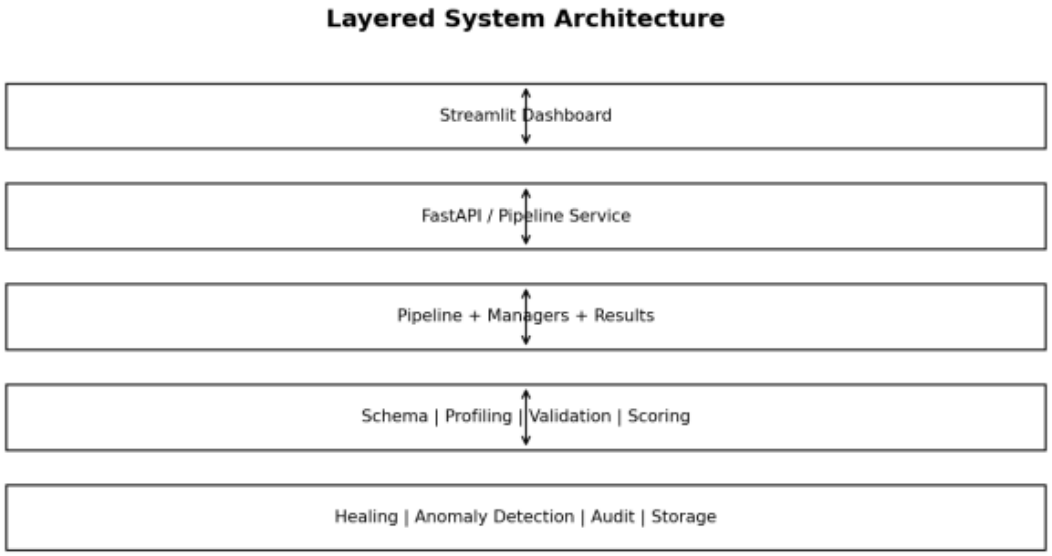


Figure 1. Layered system architecture.

- **ConfigLoader:** loads and validates YAML settings for ingestion, schema, validation, healing and anomaly detection.
- **Ingestion:** reads the configured CSV input.
- **Schema Validator:** checks required columns, types, nullability and extra columns.
- **Dataset Profiler:** calculates dataset and column-level profile metadata.
- **ValidatorManager:** executes validator plugins for nulls, duplicates, regex patterns and datatypes.
- **HealerManager:** runs configured missing-value, duplicate, regex and datatype healing strategies.
- **Anomaly Detector:** optionally applies Isolation Forest to numeric features without modifying records.
- **Quality ScoreCalculator and AuditTrail:** calculate quality measures and preserve run-level metadata.

- **FastAPI and Streamlit:** provide pipeline APIs, cleaned CSV download and a visual dashboard [7, 8].

8 Methodology

The methodology follows problem identification, requirement analysis, modular system design, implementation, integration and testing. YAML configuration keeps dataset-specific behaviour separate from Python implementation. A central configuration loader normalizes settings before execution.

Validators and healers follow common interfaces and are instantiated through managers or factories. Validators return structured results containing status, affected rows, execution time and metadata. Healers return corresponding results. The pipeline performs final validation to reconcile the reported outcome as SUCCESS, PARTIAL or FAILED.

The quality score combines completeness, validity and uniqueness using weights of 0.4, 0.3 and 0.3 respectively. Completeness measures non-null cells; validity reflects rows not affected by failed validations; uniqueness reflects the proportion of non-duplicate records. The audit trail stores metadata and metrics, not raw datasets or model objects.

9 Security and privacy considerations

The project records run-level metadata, validation outcomes and healing actions to support traceability. Raw datasets and model objects are not stored in the audit trail. The API stages CSV uploads for processing, assigns a unique run identifier and maps failures to appropriate HTTP responses. Schema failures stop downstream processing and return a validation error. Deployment-level authentication, authorization and access-control policies are potential areas for further development and have not yet been implemented.

10 Expected outcomes

- A reusable pipeline for CSV ingestion, schema validation, profiling and quality checks.
- Reduced manual effort through safe automated repair of recoverable issues.
- Comparable before-and-after quality scores and issue counts.
- Transparent reporting of successful, partial and unresolved repairs.
- Optional anomaly detection that flags unusual records without modifying them.
- API-based execution and cleaned-data download through FastAPI.
- A Streamlit dashboard with charts, status indicators and an audit timeline.

11 Results

The implemented pipeline was evaluated on the project sample dataset by comparing data-quality metrics before and after automated healing. The overall quality score increased from 77.02 to 94.13, an improvement of 17.12 points. This indicates that the combined validation, healing and re-validation workflow improved the measured quality of the dataset. The repairs made and the issues remaining for each healing strategy are shown in Figure 2.

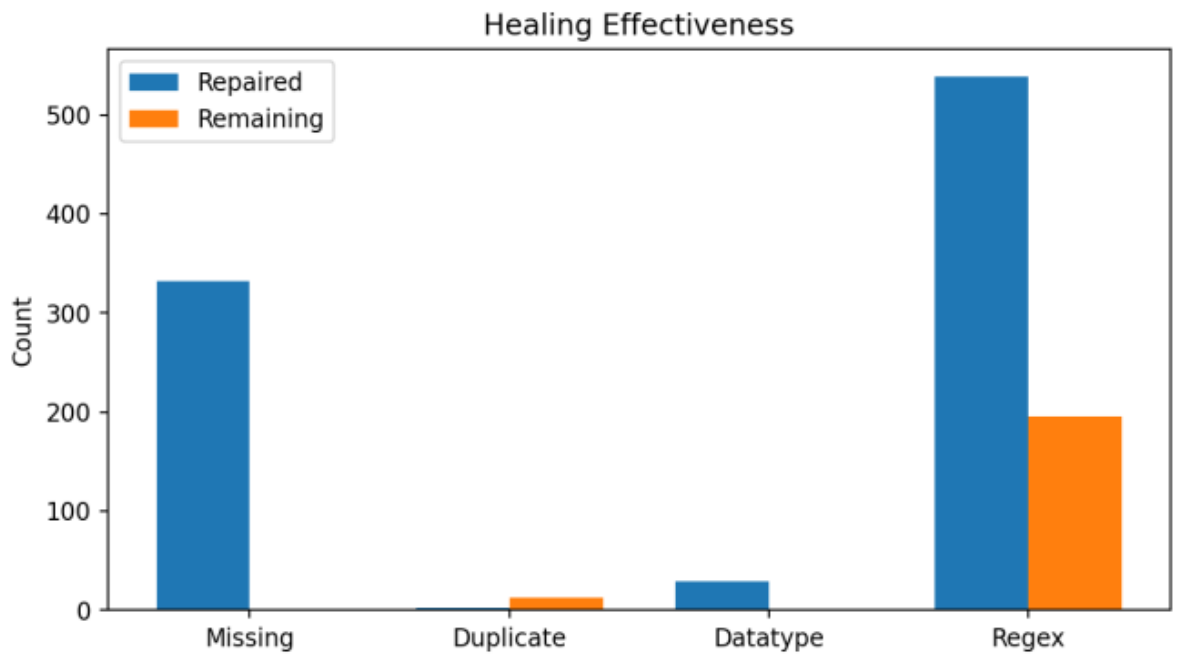


Figure 2. Healing effectiveness: issues repaired and remaining for each healing strategy.

As shown in Figure 4, missing values decreased from 332 to 0, and invalid salary values decreased from 29 to 0. Duplicate groups decreased from 7 to 6, with duplicate-affected rows decreasing from 14 to 12. Invalid phone values decreased from 642 to 103, while invalid email values remained at 92, showing that some format issues were not resolved and were correctly retained in the remaining-issues report.

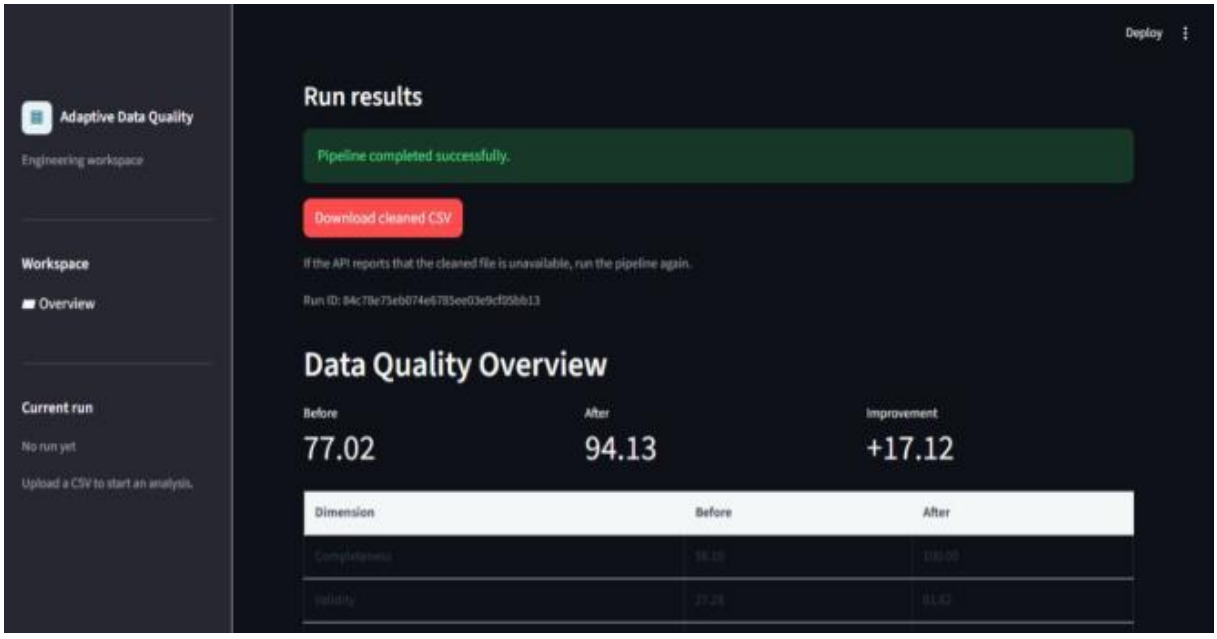


Figure 3. Data-quality overview in the Streamlit dashboard, showing the scores before and after healing.

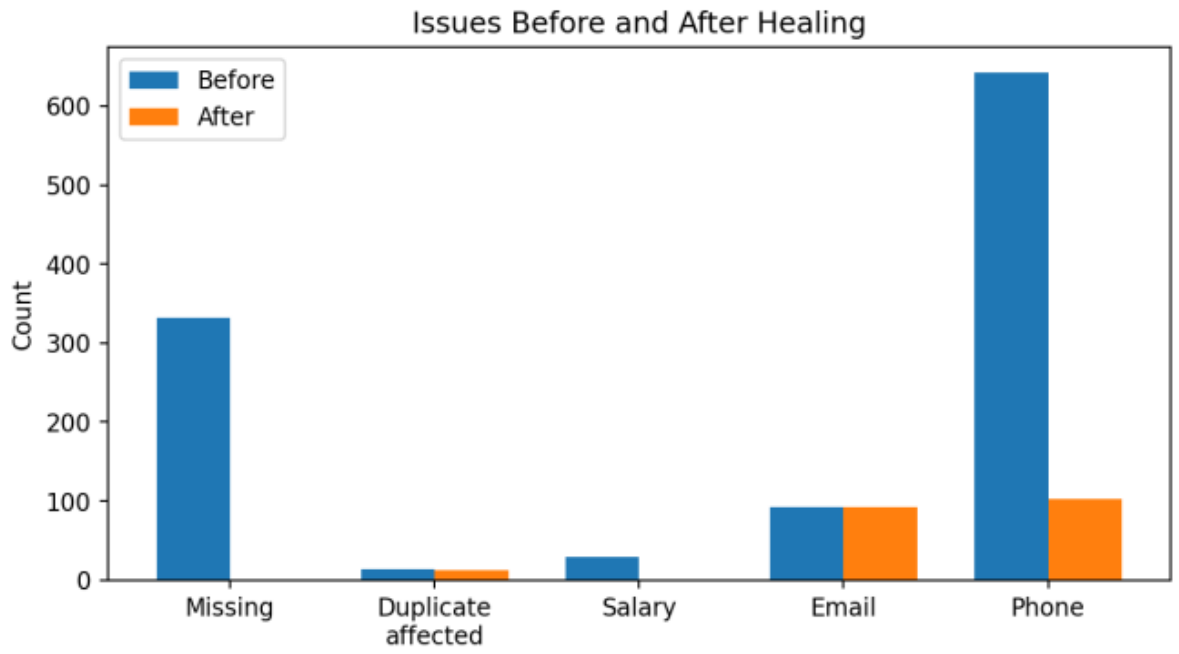


Figure 4. Issues before and after healing.

The healing results were not uniformly complete: missing-value and datatype healing reported success, while duplicate and regular-expression healing reported partial completion. The dashboard (Figure 3) presented the before-and-after scores, issue counts and healing outcomes to make these differences visible. The project's automated test suite reported 61 passing tests. These results demonstrate measurable improvement on the evaluated sample, while also showing the importance of reporting unresolved issues rather than treating every healing operation as fully successful.

12 Future scope

Future development can extend the platform beyond CSV-based workflows by adding connectors for databases, spreadsheets and cloud storage. This would allow the same configurable validation and healing process to be applied to datasets from different sources.

A human-in-the-loop approval workflow could be introduced for repairs that carry a higher risk of changing meaningful data. Users could review proposed changes, approve or reject them, and inspect the reasons behind each recommendation before the cleaned dataset is produced.

Persistent run history and richer audit reports could support comparisons across multiple pipeline executions. This would help users track recurring data-quality issues, monitor long-term improvement and investigate when a problem first appeared.

Anomaly detection could be enhanced with configurable thresholds, additional detection techniques and clearer explanations for flagged records. As unusual values may be legitimate, anomaly flags should continue to be treated as recommendations for review rather than automatically corrected.

Further work could strengthen deployment readiness through authentication, role-based authorization, access controls, configurable file limits, and monitoring. These additions would help the platform operate more safely in shared or production environments.

13 Conclusion

The Smart Data Quality and Auto-Healing Pipeline combines configurable ingestion, schema validation, profiling, rule-based validation, optional anomaly detection, safe repair, quality scoring, audit metadata and reporting in a modular workflow. In the evaluated sample, the overall quality score improved from 77.02 to 94.13. Missing-value and datatype issues were resolved, while remaining email, phone and duplicate issues were reported transparently. FastAPI and Streamlit provide an integrated service and dashboard experience, and 61 automated tests passed. Future work can extend data-source support, approval workflows, persistent history, anomaly detection and deployment controls.

References

[1] Rahm E, Do HH. Data cleaning: problems and current approaches. *IEEE Data Engineering Bulletin*. 2000;23(4):3–13.

[2] Chu X, Ilyas IF, Krishnan S, Wang J. Data cleaning: overview and emerging challenges. In: *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*; 2016. p. 2201–2206.

[3] Liu FT, Ting KM, Zhou ZH. Isolation forest. In: *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*; 2008. p. 413–422.

[4] McKinney W. *Python for data analysis*. 3rd ed. Sebastopol (CA): O'Reilly Media; 2022.

[5] Python Software Foundation. *Python 3 documentation*. Available from: https://docs.python.org/3/

[6] pandas development team. pandas: Python data analysis library. *Documentation*. Available from: https://pandas.pydata.org/docs/

[7] FastAPI. *FastAPI: modern, fast web framework for building APIs with Python*. Documentation. Available from: https://fastapi.tiangolo.com/

[8] Streamlit. *Streamlit: a faster way to build and share data apps*. Documentation. Available from: https://docs.streamlit.io/

[9] scikit-learn developers. *IsolationForest*. scikit-learn documentation. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html