ARTIFICIAL INTELLIGENCE, DATA PRE-PROCESSING

Agent-Based Framework for Automated Data Cleaning and Validation

J. Gnanashri, M N. Manya, Sakshi C. Patil, B. Likitha

Department of Artificial Intelligence and Data Science, GSSS Institute of Engineering and Technology for Women (affiliated to VTU), Mysuru, Karnataka, India

Published November 2026

Abstract

Real-world datasets can contain missing values, duplicate records, inconsistent data, outliers, and invalid entries. Manual cleaning is time-consuming, and the result must be checked to determine whether it satisfies data-quality requirements. This paper presents an agent-based framework for automated data cleaning, validation, and merging implemented using Python, NumPy, and Pandas, with Flask and scikit-learn used in the prototype application. Different agents handle schema profiling, issue detection, reasoning, cleaning, and validation. The system detects candidate problems using type inference, pattern matching, hash matching, and IQR-based outlier detection. A Reasoning Agent proposes possible cleaning actions, which are presented to the user for approval or rejection before accepted transformations are applied. The Validation Agent then checks the output against configured quality rules. If validation identifies remaining problems, the issue can be returned to the reasoning stage for a further proposed correction and user decision. The framework also supports merging two datasets using schema matching. Screenshots from a customer-data demonstration show a reduction in reported missing values and duplicate rows, while the validation dashboard still reports failed checks. These observations describe prototype interface output rather than a controlled benchmark; they do not establish general cleaning accuracy or performance. The framework aims to reduce repetitive manual work while retaining human oversight and an explicit validation step before data are used for analysis or machine learning.

Keywords: agent-based framework; automated data cleaning; data validation; data merging; data quality; missing values; duplicate records; outlier detection; invalid data; type inference; pattern matching; hash matching; IQR; reasoning agent; validation agent; schema matching; NumPy; pandas; human-in-the-loop

1 Introduction

Data-driven applications depend on datasets that are sufficiently complete, consistent, and valid for their intended use. In practice, tabular data are collected from forms, transaction systems, spreadsheets, and external sources, and may contain missing values, duplicate rows, inconsistent spelling, invalid dates, mixed data types, and values outside plausible ranges. Such defects can distort descriptive statistics, bias models, and complicate integration across data sources [1].

Data cleaning is not a single operation. It typically involves profiling, identifying quality issues, selecting transformations, applying changes, and checking whether the result satisfies explicit requirements. A script that silently fills every missing value or removes every outlier can make a table look complete while damaging its meaning. The appropriate action depends on column semantics, domain constraints, and the purpose of the analysis.

DataGuard AI addresses this workflow by separating issue detection from repair. Its interface displays candidate actions and supports human approval before transformations are applied. The prototype also offers a second workflow for merging two related CSV files and imputing missing values. This paper documents the design and the observations visible in the supplied application screenshots, while distinguishing implemented prototype behaviour from evaluation that has not yet been performed.

1.1 Contributions

The contributions are: (1) a modular workflow organized around profiling, detection, reasoning, transformation, and validation; (2) an approval step intended to keep consequential repairs reviewable; (3) support for single-file cleaning and two-file merge/imputation; and (4) a transparent account of prototype observations and limitations, without treating dashboard pass rates as evidence of general accuracy.

2 Literature survey

Rahm and Do classify data-quality problems and describe cleaning approaches for single-source and integrated datasets, emphasizing that cleaning interacts with schema transformation and data integration [1]. Krishnan and Wu present AlphaClean, a framework that searches for data-cleaning pipelines using quality measures and candidate transformations [2]. These works motivate explicit quality objectives rather than assuming that one fixed sequence is suitable for every dataset.

Qi, Miao, and Wang introduce CleanAgent, which combines declarative data-standardization operations with language-model agents to simplify column standardization [3]. Dong et al. describe DQuag, a graph-neural-network-based framework for data-quality validation and repair, targeting relationships among values that simple independent column checks may miss [4].

Li et al. propose AutoDCWorkflow, which generates data-cleaning workflows for a stated analysis purpose and evaluates workflow and output quality against annotated examples [5]. Yasin and Khorsheed discuss machine-learning-based approaches to automated cleaning in large databases [6]. HoloClean combines integrity constraints, external information, and statistical evidence to infer data repairs [7], while Chu et al. survey data-cleaning research and emerging challenges, including rule-based and statistical approaches [8]. These studies differ in automation mechanism, evaluation design, and target task; they are related work, not direct performance baselines for DataGuard AI.

Table 1. Selected related work.

Work	Focus	Relevance
Rahm & Do [1]	Cleaning problems	Profiling and integration
AlphaClean [2]	Pipeline search	Candidate repairs and quality measures
CleanAgent [3]	Agent-assisted standardization	Modular column-aware actions
DQuag [4]	Learned validation/repair	Cross-column dependencies
AutoDCWorkflow [5]	Workflow generation	Traceable operations

3 Problem statement, aim and objectives

3.1 Problem statement

Preparing heterogeneous CSV datasets for reliable analysis is laborious when quality issues are not known in advance. Manual inspection and dataset-specific scripts can be repetitive, difficult to audit, and brittle when column names or data types change. Automated repair introduces a second risk: a transformation can be technically valid yet semantically wrong, particularly for email addresses, identifiers, dates, rare categories, or extreme values.

3.2 Aim and objectives

The aim is to develop a modular application that identifies common data-quality issues, proposes context-sensitive actions, allows user review, applies approved transformations, and reports validation status. The objectives are to profile dimensions and column characteristics; detect missing values, duplicates, formatting issues, invalid types, and potential outliers; propose candidate repairs; support two-file merging and imputation; validate the result; and provide a cleaned CSV and execution summary.

4 Proposed methodology

4.1 System architecture

The prototype follows the staged pipeline illustrated in Figure 1. A user uploads a CSV or selects the two-file workflow. The profiling stage summarizes the schema and column characteristics. The detection stage identifies candidate issues using type inference, pattern matching, hash matching, and IQR-based outlier detection. The Reasoning Agent examines detected issues and generates possible cleaning proposals. The interface presents these proposals to the user, who can approve or reject each action. Approved transformations are applied by the Cleaning Agent, after which the Validation Agent checks the resulting data against configured quality rules. When a check fails, the issue can be returned to the reasoning stage so that another proposal can be considered and reviewed by the user.

4.2 Workflow variants

Figure 2 summarizes the two workflow variants. In single-file mode, the sequence is upload, schema profiling, issue detection, reasoning, proposal review, approved cleaning, validation, and export. In two-file mode, the system profiles both inputs and performs schema matching to identify compatible fields or candidate join keys. The user-selected merge is then performed, missing values may be imputed using the configured method, and the resulting dataset is validated. The merge workflow should report the selected keys and join type, as well as row counts and unresolved issues.

4.3 Proposed orchestration algorithm

Algorithm 1 summarizes the intended control flow. It describes the system design rather than claiming that every safeguard is fully implemented in the prototype.

Algorithm 1: Orchestration of the cleaning workflow

1. Preserve the uploaded input and create a schema profile.
2. Detect candidate issues and attach rule IDs, columns, counts, and explanations.

3. Generate proposed actions using configured rules and inferred column roles.
4. Request user decisions for actions marked for review.
5. Apply accepted actions; log rejected, skipped, and failed actions.
6. For two-file mode, record keys and join type, then report unmatched or expanded records.
7. Run validation on the transformed output and retain all failed checks.
8. Export the cleaned CSV, validation summary, and execution log; label unresolved failures.



Fig. 1. Conceptual DataGuard AI pipeline with user review before transformation.

Figure 1. Agent-based pipeline and human approval stage.

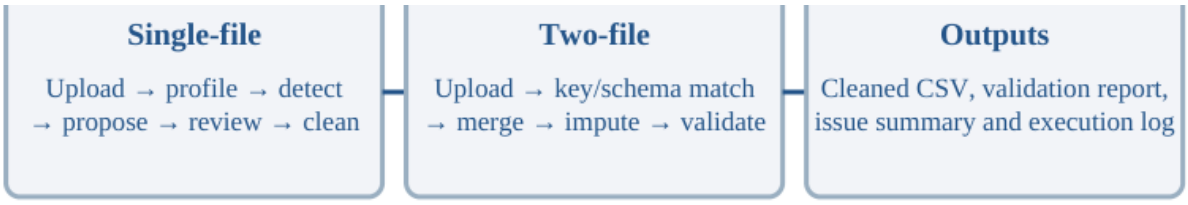


Figure 2. Main workflow variants and expected output artifacts.

4.4 Schema profiling agent

The Schema Profiling Agent creates a compact description of the input table. For each column, the profile should include its name, inferred type, row count, missing-value count, unique-value count, and missing percentage. Profiling helps distinguish numeric columns from categories and dates, but inferred types remain hypotheses: a numeric-looking identifier should not necessarily be averaged, and a date stored as text requires format-aware parsing.

For column j with N rows and M_j missing values, the missing percentage is $p_j = 100 \times M_j / N$. This statistic describes completeness only; it does not establish whether the remaining values are correct or whether missingness is acceptable for the application.

4.5 Issue detection agent

The Issue Detection Agent examines the profile and values for common quality patterns. Missing-value detection should treat nulls and, where configured, empty strings or sentinel tokens consistently. Type inference identifies likely data types, while pattern matching can flag values that do not conform to expected formats. Hash matching supports duplicate-candidate detection; the implementation should state whether matching is performed on full rows or selected key columns. Range checks

and IQR-based outlier detection flag values that may require inspection. These techniques identify candidates for review and do not, by themselves, prove that a value is incorrect.

Outlier detection may use methods such as the interquartile range or a standardized-score threshold. Such methods identify unusual values, not necessarily errors. A high income, large transaction, or unusual age may be legitimate; therefore, the interface should flag potential outliers for inspection rather than automatically deleting them.

4.6 Reasoning agent and human review

The Reasoning Agent examines detected issues and proposes possible cleaning actions based on the column type, observed distribution, and configured rules. Numeric missing values may be proposed for median imputation, while categorical values may be proposed for mode imputation; these are candidate defaults rather than universally correct repairs. Pattern violations may lead to a format-standardization proposal, and duplicate candidates may lead to a deduplication proposal. The user can approve or reject the suggested actions before they are applied. If validation later identifies an unresolved problem, the issue can be sent back to the reasoning stage for another proposal and user decision.

The approval interface lets users review proposed operations before application. A reproducible implementation should record the issue, affected column, proposed method, affected-row count, and whether the action was accepted, rejected, or modified. The screenshots show an approval list and an Apply Approved Cleaning control; they do not establish that every proposed operation is semantically correct.

4.7 Cleaning and merge/imputation agents

The Cleaning Agent applies approved operations such as deduplication, standardization, type conversion, and imputation. It should preserve the original input and avoid destructive changes that cannot be traced. Where possible, transformations should be idempotent or record enough metadata to explain how the output was produced.

The two-file workflow uses schema matching to compare the structure and compatible fields of both datasets before merging. Candidate join keys should be inspected for missingness, uniqueness, and compatible types. The application should expose the selected merge strategy and, where available, row counts before and after merging, unmatched keys, and duplicate-key expansion. The project materials describe mean, median, mode, KNN, and regression-based imputation as possible methods; the appropriate method depends on the field, data distribution, and configured workflow.

4.8 Validation agent

The Validation Agent evaluates the transformed dataset against configured quality rules. Useful categories include completeness, type compatibility, format, uniqueness, allowed values, range/domain, consistency, referential integrity, and business rules. A validation pass rate can be defined as $R = 100 \times P / C$, where P is the number of passed checks and C is the total number of checks, provided C is nonzero. A failed check does not automatically imply that a correction is safe; the failure should be made available to the Reasoning Agent, and any proposed follow-up action should be presented for user approval before application.

The system should retain failed checks with a rule name, affected column, issue count, and explanatory message. The observed dashboard follows this pattern by showing PASS/FAIL, a column, an issue count, and a description.

5 Experimental setup and results

5.1 Implementation environment

The system is implemented in Python as the programming language and Flask as the web application framework, with Pandas and NumPy for tabular processing and scikit-learn for selected machine-learning operations. Figure 3 shows the main interface and workflow selector; Figure 4 displays the initial dataset overview; and Figure 5 shows the column profile and detected quality issues.

The evaluation is based on interface screenshots of the prototype rather than a reproducible experiment package; dependency versions, hardware configuration, execution times, labelled ground truth and an automated test report are not reported. Accordingly, the evaluation is descriptive and limited to values visibly displayed in the application.

5.2 Dataset profile

The customer-data demonstration begins with a CSV named customers_10k_unclean.csv. The overview reports 10,100 rows and 9 columns. The fields shown are Customer_ID, Name, Age, City, Email, Gender, Annual_Income, Signup_Date, and Customer_Status. The initial dashboard reports 802 missing values and 95 duplicate rows. These are counts from the displayed run, not a claim about an external public benchmark.

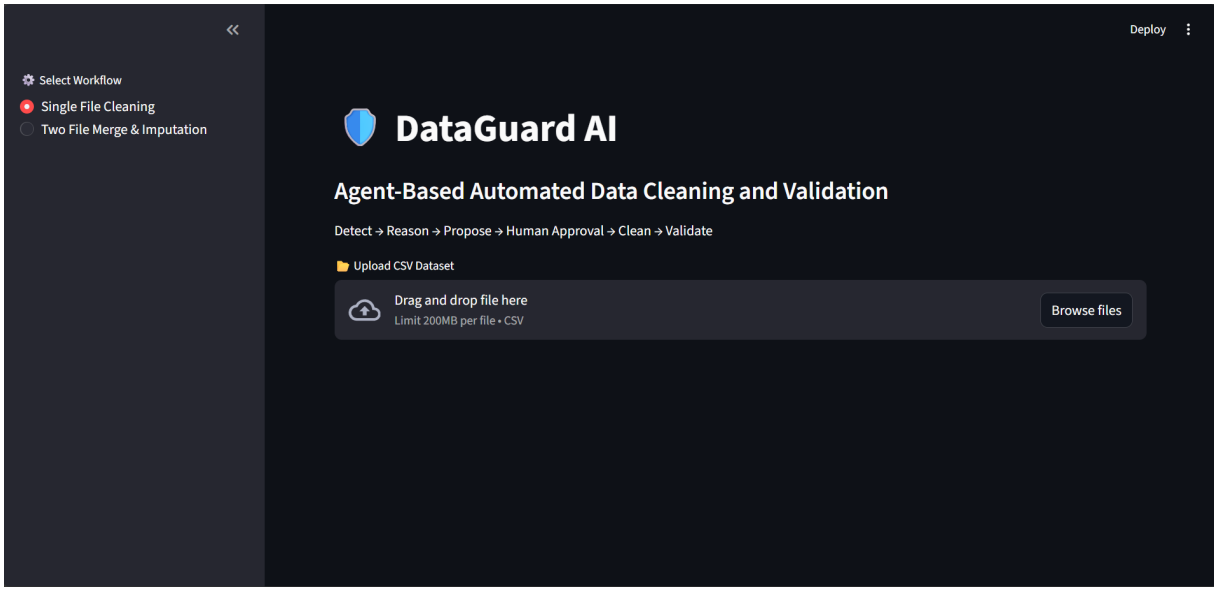


Figure 3. Main DataGuard AI interface and workflow selection.

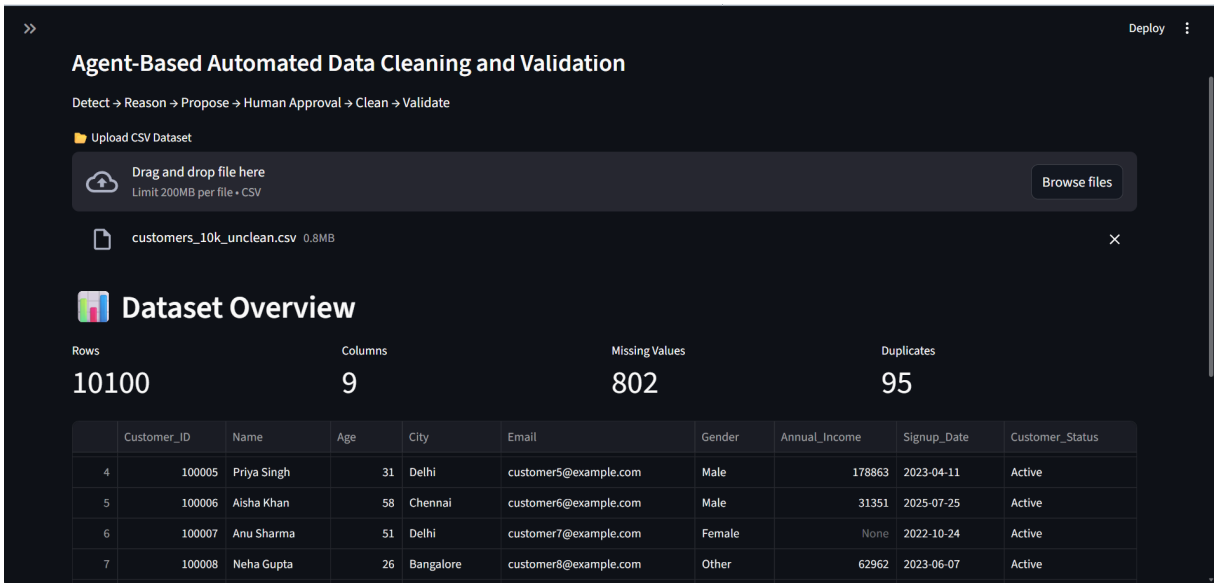


Figure 4. Customer-data overview: 10,100 rows, 9 columns, 802 missing values, and 95 duplicates.

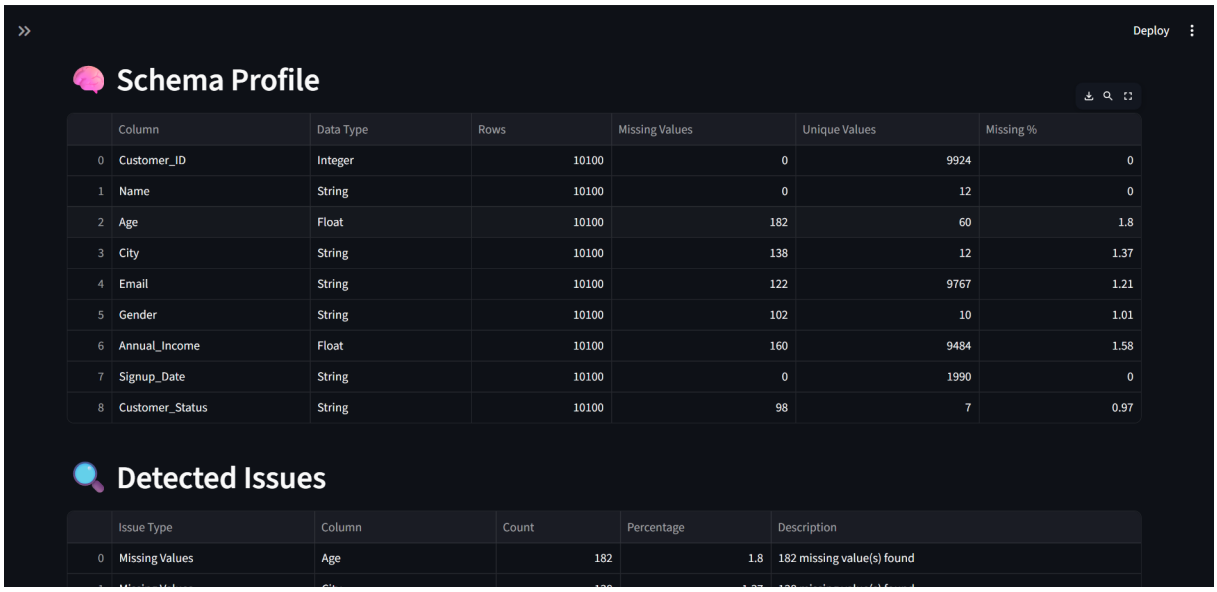


Figure 5. Schema profile and detected data-quality issues.

5.3 Proposed actions and cleaned output

Figure 6 shows the proposed actions: mode imputation for City, Gender, and Customer_Status; median imputation for Annual_Income; human review for Email; duplicate-row removal; capitalization standardization for City, Gender, and Customer_Status; and human review for outliers in Age and Annual_Income. Figure 7 shows a sample of the cleaned customer table. These are interface observations; the evidence does not prove that every selected proposal was appropriate for the data domain.



Figure 6. Human-review interface for proposed cleaning actions.

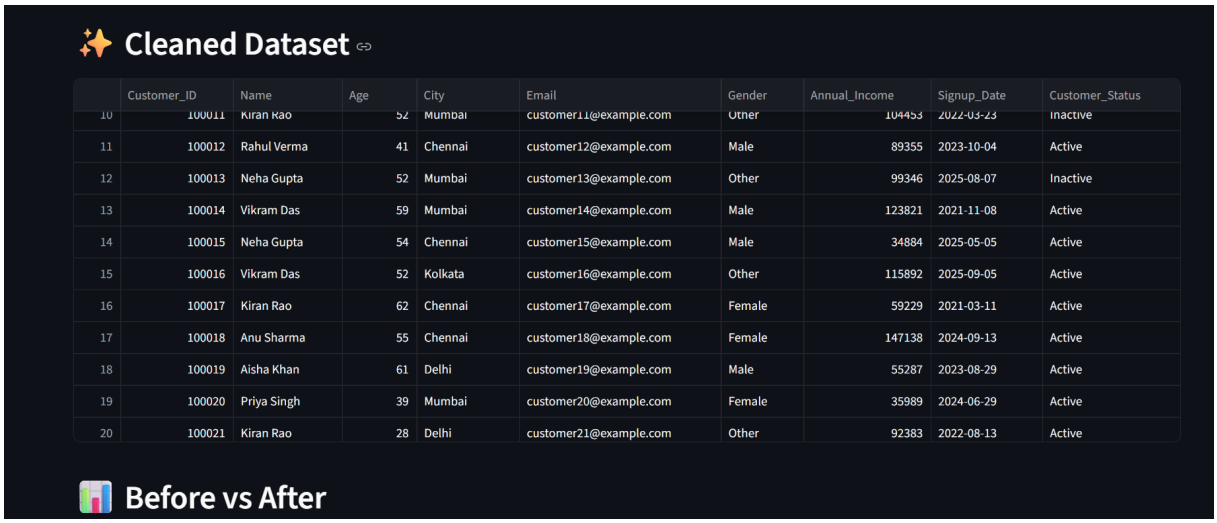


Figure 7. Preview of the cleaned customer dataset.

5.4 Before-and-after observations

Table 2 summarizes the dashboard values visible in the customer-data screenshots. The row count decreases from 10,100 to 10,005, a net reduction of 95 rows. The displayed missing-value count decreases from 802 to 119, a net reduction of 683 (approximately 85.2% of the initial count). The duplicate count decreases from 95 to zero. These differences describe the shown run only; without row-level provenance, they should not be treated as a general error-removal rate.

Table 2. Customer-data prototype observations.

Metric	Before	After
Rows	10,100	10,005
Columns	9	9
Missing values	802	119
Duplicate rows	95	0
Validation checks	—	32
Passed / failed	—	24 / 8
Validation pass rate	—	75.0%

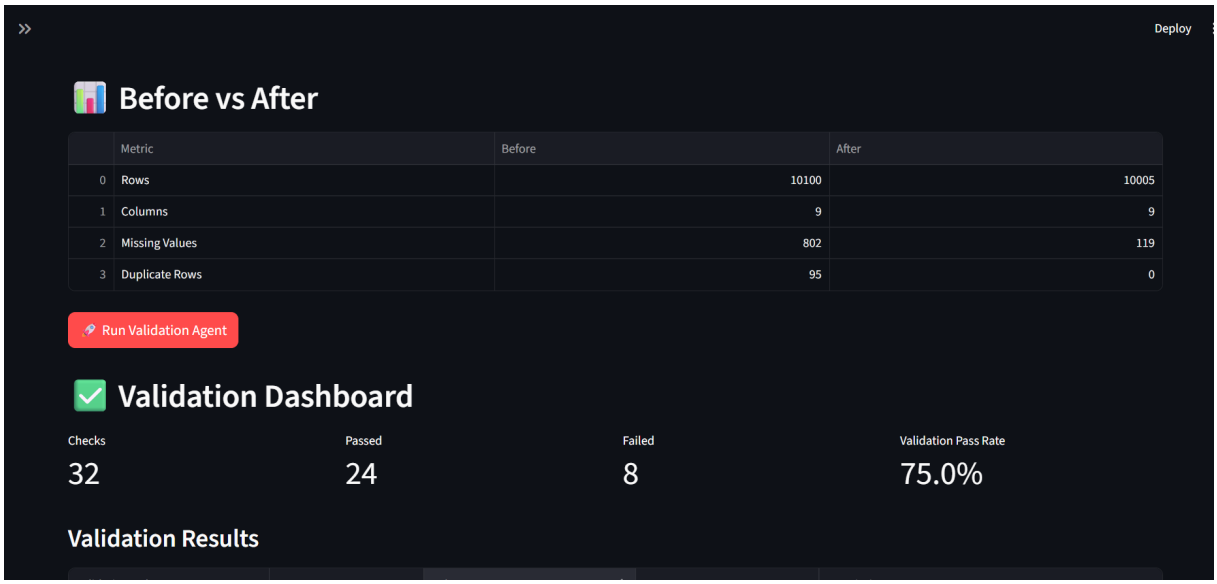
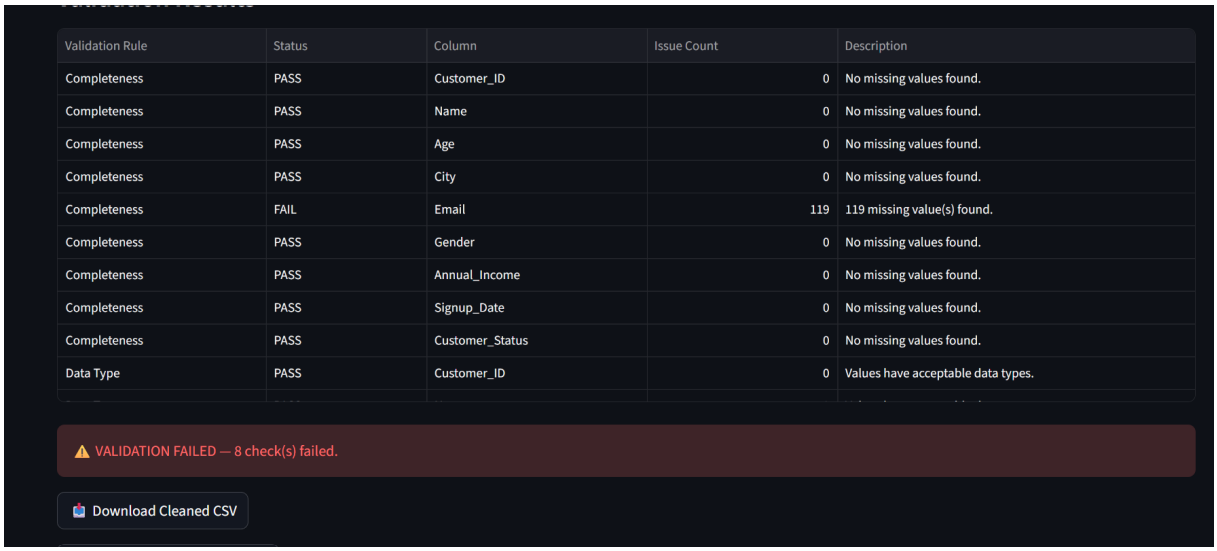


Figure 8. Before-and-after counts and validation dashboard summary.

Figure 8 combines the before-and-after counts with the validation summary. The dashboard still reports failed checks after cleaning. This is an important observation: the pipeline reduces selected quality issues in the displayed summary but does not make the dataset fully valid under the configured rules. The application should present failed checks as actionable findings rather than describing the run as completely successful.

5.5 Validation results



Validation Rule	Status	Column	Issue Count	Description
Completeness	PASS	Customer_ID	0	No missing values found.
Completeness	PASS	Name	0	No missing values found.
Completeness	PASS	Age	0	No missing values found.
Completeness	PASS	City	0	No missing values found.
Completeness	FAIL	Email	119	119 missing value(s) found.
Completeness	PASS	Gender	0	No missing values found.
Completeness	PASS	Annual_Income	0	No missing values found.
Completeness	PASS	Signup_Date	0	No missing values found.
Completeness	PASS	Customer_Status	0	No missing values found.
Data Type	PASS	Customer_ID	0	Values have acceptable data types.

Figure 9. Validation results: 32 checks, 24 passed, 8 failed, and a 75.0% pass rate.

Figure 9 shows the validation table, including an Email completeness failure with 119 missing values, while several other completeness checks are shown as passing. The dashboard reports eight failed checks in total, but the screenshots supplied do not expose all failed rows in a single readable view. This paper therefore does not speculate about the remaining failures.

The displayed pass rate is 24 / 32 × 100 = 75.0%. It should not be interpreted as 75% of records being correct; it is the proportion of configured checks that passed. Rule coverage, severity, and issue counts are required for a more meaningful quality assessment.

5.6 Validation protocol and test design

A defensible evaluation should separate detection quality from repair quality. Detection asks whether a rule correctly identifies a known defect; repair asks whether the proposed or applied transformation restores the intended value without changing valid data. These are different questions. For example, a missing email value can be detected with certainty as missing, but no cleaning algorithm can infer the correct email address from that cell alone without an external source. The system should therefore report detection and repair outcomes separately.

A controlled test set can be created by starting with a dataset whose values have been checked and introducing known defects into a copy. Defect injection can include removing values at a specified rate, duplicating selected rows, changing date formats, altering category capitalization, inserting invalid types, and adding known out-of-range values. The original table becomes the reference for measuring recovery. The injection process, random seed, modified row identifiers, and affected columns should be saved so that every run can be repeated.

For issue detection, precision is TP/(TP+FP), recall is TP/(TP+FN), and F1 is the harmonic mean of precision and recall. Here, TP denotes correctly detected defects, FP denotes valid cells incorrectly flagged, and FN denotes defects missed by the detector. For repair, cell-level accuracy can be computed over values that were intentionally corrupted, but it should be accompanied by the rate of valid cells changed. A high repair rate is not sufficient if the method also alters correct data. These metrics were not calculated for the screenshots in this paper.

A benchmark should include multiple datasets with different row counts, column types, missingness patterns, category cardinalities, and duplicate structures. Results should be reported per dataset and per defect type, not only as a single average. A simple deterministic baseline using explicit Pandas rules would help identify whether the agent-based orchestration provides measurable benefits beyond a fixed cleaning script. All methods should receive the same input, schema constraints, and target rules.

5.7 Runtime, scalability, and complexity considerations

The screenshots do not include execution times or resource measurements. For future evaluation, runtime should be measured separately for upload/parsing, profiling, issue detection, proposal generation, transformation, validation, and export. Report dataset dimensions, data types, hardware, software versions, and whether the run includes cold-start costs. Memory use should also be recorded because loading a full CSV into memory can become a limiting factor for larger datasets.

The expected computational cost depends on the operation and implementation. A one-pass profile of N rows and M columns is generally proportional to the number of inspected cells, O(NM). Hash-based duplicate detection can be approximately linear in the number of rows under typical assumptions, while sorting-based methods may require O(N log N) time. Validation across every cell is also typically O(NM), although specialized rules may cost more. These are analytical expectations, not measured performance claims about the current prototype.

Merge cost depends on the join algorithm, key distribution, and whether the keys are indexed or hashed. Before a merge, the application should check key nulls and duplicates, and after a merge it should compare row counts with expected join behaviour. A many-to-many join can increase row count substantially even when the operation itself succeeds. For this reason, the user interface should surface join type, matched and unmatched key counts, and whether duplicate keys may expand the output.

5.8 Data quality rules and severity

Validation rules should be explicit, versioned, and associated with a severity level. A completeness rule checks whether a required field is missing; a type rule checks whether values conform to the expected representation; a format rule checks patterns such as email or date syntax; a uniqueness rule checks a key constraint; a range rule checks domain limits; and a referential-integrity rule checks that foreign keys match an authoritative set. Business rules can express relationships between columns, such as a non-negative quantity or a total consistent with quantity and unit price.

Not every rule should be applied universally. An email syntax check can identify malformed strings but cannot establish that an address exists or belongs to a particular person. An age range can flag impossible or suspicious values, but the accepted domain depends on the application. A uniqueness constraint should only be applied to a field intended to be unique. The rule configuration should document the reason for each check, its allowed values or thresholds, and the consequence of failure.

A single pass rate can obscure severity. If a dataset passes 31 low-risk checks but fails one mandatory identifier constraint, the aggregate percentage may appear reassuring even though the output is unusable. The report should therefore show the numerator and denominator, individual failures, severity, issue count, and whether each failure blocks export. If the application allows download despite failed checks, the output should be labelled as cleaned but not fully validated.

5.9 Human-in-the-loop governance

Human review is useful when the appropriate repair depends on domain knowledge or when the cost of an incorrect repair is high. The reviewer should see the original value, proposed value or operation, rationale, affected-row count, and likely consequences. For large proposals, a sample of affected records and a summary distribution can help the reviewer decide whether the action is reasonable. A blanket approval control without enough context risks turning human review into a formality.

The audit record should distinguish an issue being detected, an action being proposed, an action being approved, and an action being applied. Rejected actions should also be recorded. This separation allows a later reviewer to explain why the output differs from the input and to reproduce the same cleaning run. The supplied interface shows proposed actions and an approval control; a complete audit trail is a recommended evaluation and implementation requirement rather than a verified capability from the screenshots.

Sensitive customer data require additional safeguards. The application should state how uploaded files are processed, whether files are retained after a session, who can access outputs, and how temporary files are removed. Logs should avoid unnecessarily storing personal values. When the project is tested with real customer data, consent, institutional requirements, and applicable privacy obligations should be considered before uploading or distributing the dataset.

5.10 Reproducibility and reporting

A reproducible run record should include an identifier, input file names or hashes, schema profile, application version, selected workflow, approved operations, rule configuration, row counts before and after each major stage, validation results, and output-file location. For two-file workflows, record key columns, join type, pre-merge key uniqueness, matched/unmatched counts, and post-merge dimensions. Such metadata make failures easier to diagnose and reduce ambiguity when two screenshots show different counts.

The application should preserve the raw input and write the cleaned output to a separate file. Each transformation should record the number of cells or rows affected, with exceptions reported rather than silently ignored. A before/after comparison should include missing values, duplicate definitions, invalid-value counts, and validation outcomes. These summaries are useful for triage, but correctness claims require independent expected values or domain review.

The current customer-data screenshots show 10,100 rows, 9 columns, 802 missing values, and 95 duplicate rows initially, followed by a displayed summary of 10,005 rows, 119 missing values, and zero duplicates.

5.11 Practical interpretation of the current results

The customer-data dashboard supports a limited conclusion: the displayed cleaning run reduced the reported missing-value count and duplicate-row count, while the validation dashboard continued to show eight failed checks. The screenshots do not reveal whether the 683 fewer missing values were all replaced with correct values, whether any non-missing values were changed incorrectly, or which operations account for the net reduction of 95 rows. The results should therefore be described as observed interface output, not as a verified improvement in ground-truth data quality.

The failed Email completeness rule is visible with 119 missing values. This suggests that the email field remained incomplete in the displayed post-cleaning table, which is consistent with the proposal screen assigning Email to human review rather than automatic imputation. That is a reasonable conservative choice because a missing email generally cannot be reconstructed safely from the surrounding row alone. The remaining seven failed checks are not fully visible in the supplied screenshot; their causes should be taken from a complete validation export before being discussed.

Future evaluation should compare the initial and final datasets at cell level, inspect representative repaired records, verify all key constraints, and retain failed validations. If the goal is to claim that the system improves data quality, the study should define quality before running the system, use independent labels, and report both successful repairs and unintended changes. If the goal is to claim scalability, it should report measured runtime and memory across increasing data sizes. Neither claim can be established by screenshots alone.

5.12 Operational algorithm

The workflow can be expressed as a sequence of auditable stages rather than a single opaque cleaning operation. First, read the uploaded CSV and retain an unchanged copy. Second, compute the schema profile and quality indicators. Third, generate issue records with a rule identifier, column, count, and explanation. Fourth, map each issue to one or more candidate actions. Fifth, obtain user decisions for actions that require approval. Sixth, apply only accepted transformations and record their effects. Seventh, run the configured validation rules against the transformed table. Finally, export the table together with the validation report and execution log. If a stage fails, the application should preserve the original input and report the failure instead of presenting a partial result as complete.

This ordering matters because validation is not equivalent to cleaning. A transformation can remove one defect and introduce another; for example, converting a text column to numeric values can turn unparseable strings into missing values. Re-running the profiler and validator after transformation helps reveal such side effects. For reliable implementation, each stage should return structured output and explicit status information so that the orchestrator can distinguish a successful stage, a warning, and a blocking error.

5.13 Recommended validation rule matrix

Table 3 summarizes common rule categories suitable for the application. The exact rules, thresholds, and severities should be configurable by dataset rather than hard-coded as universal truths. The matrix describes evaluation requirements; it does not claim that every listed rule is currently implemented in the supplied prototype.

Table 3. Validation rule categories and reporting.

Rule	Purpose	Example output
Completeness	Required values present	Missing count per column
Type / format	Values parse as expected	Invalid count and examples
Uniqueness	Key or row uniqueness	Duplicate count and key
Range / domain	Values within declared bounds	Out-of-range count
Consistency	Related fields agree	Rule-specific violations

Rule	Purpose	Example output
Reference integrity	Keys exist in lookup table	Unmatched-key count
Business rule	Application constraints hold	Failed condition and rows

A validation report should expose enough context for a user to act on a failure. A rule name and pass/fail label are useful, but the report should also include the configured expectation, the number of affected rows, a small sample of examples where privacy permits, and a severity or blocking status. This supports investigation without overstating what an aggregate pass rate means.

6 Limitations and discussion

The prototype demonstrates a structured interface and an observable sequence from upload to profiling, cleaning proposals, output preview, and validation. However, screenshots alone cannot establish the internal correctness of every agent, reproducibility across runs, scalability to large files, or superiority over existing cleaning tools.

Schema inference can misclassify identifiers, dates, categorical codes, or numeric strings. Imputation can introduce bias or unrealistic values when missingness is not random or when domain constraints are unknown. Outlier thresholds can flag legitimate rare cases. Deduplication can remove valid records if the duplicate definition is too broad. Merges can multiply rows when join keys are not unique. Human review reduces some risks but introduces time cost and depends on reviewer knowledge.

The displayed before/after values should be treated as descriptive UI output. The initial and cleaned counts do not show which exact rows were changed, whether any valid values were altered, or whether every business rule was satisfied. The 75.0% validation pass rate explicitly indicates that eight configured checks failed. The application should not label this state as fully validated.

Future work should add a versioned test suite, synthetic corruption with known ground truth, row-level transformation logs, schema contracts, configurable severity levels, merge diagnostics, and benchmark comparisons. Privacy protections are also necessary when datasets contain customer information, including clear retention policies and safe handling of uploaded files.

6.1 Threats to validity

Construct validity is limited because the selected checks may not cover all meaningful defects. Internal validity is limited because the screenshots do not provide a complete audit trail or controlled comparison. External validity is limited because only a small number of prototype examples are visible. Repeating the experiment with fixed inputs, saved rules, versioned software, and independent expected outputs is necessary before making broader claims.

7 Conclusion

This paper presents an agent-based framework for automated data cleaning, validation, and merging. The workflow separates schema profiling, issue detection, reasoning, human approval, cleaning, and validation. It uses type inference, pattern matching, hash matching, and IQR-based outlier detection to identify candidate issues; proposed actions can be approved or rejected by the user. The two-file workflow uses schema matching, and validation failures can be returned to the reasoning stage for

further review. The supplied customer-data screenshots show lower reported missing-value and duplicate counts after cleaning, but the validation dashboard still reports failures. These observations support a limited description of the prototype workflow, not a claim of universal correctness or measured performance.

These observations demonstrate that the interface supports a reviewable cleaning workflow and surfaces remaining validation failures. They do not establish general cleaning accuracy, runtime performance, or applicability to arbitrary datasets. The next step is a reproducible evaluation using labelled or synthetically corrupted datasets, complete transformation logs, and baseline comparisons.

Acknowledgment

The authors acknowledge the Department of Artificial Intelligence and Data Science, GSSS Institute of Engineering and Technology for Women, Mysuru, Karnataka, for the academic project context. The project guide is identified in the supplied project materials as Ms. Likitha B, Assistant Professor.

References

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

[2] Krishnan S, Wu E. AlphaClean: Automatic generation of data cleaning pipelines. *arXiv*:1904.11827. 2019.

[3] Qi D, Miao Z, Wang J. CleanAgent: Automating data standardization with LLM-based agents. In: *Proc*. VLDB 2025 Workshop on Data Driven AI (DDAI); 2025.

[4] Dong S, Sahri S, Palpanas T, Wang Q. Automated data quality validation in an end-to-end GNN framework. In: *Proc*. 28th International Conference on Extending Database Technology (EDBT); 2025. p. 874–880. https://doi.org/10.48786/EDBT.2025.71

[5] Li L, Fang L, Ludäscher B, Torvik VI. AutoDCWorkflow: LLM-based data cleaning workflow auto-generation and benchmark. In: *Findings of the Association for Computational Linguistics: EMNLP 2025*; 2025. p. 7766–7780. https://doi.org/10.18653/v1/2025.findings-emnlp.410

[6] Yasin HM, Khorsheed AK. Automated data cleaning in large databases using machine learning methods. *Asian Journal of Research in Computer Science*. 2025;18(5):364–386. https://doi.org/10.9734/ajrcos/2025/v18i5661

[7] Rekatsinas T, Chu X, Ilyas IF, Ré C. HoloClean: Holistic data repairs with probabilistic inference. *Proc VLDB Endowment*. 2017;10(11):1190–1201. https://doi.org/10.14778/3137628.3137631

[8] Chu X, Ilyas IF, Krishnan S, Wang J. Data cleaning: Overview and emerging challenges. In: *Proc*. ACM SIGMOD International Conference on Management of Data; 2016. p. 2201–2206. https://doi.org/10.1145/2882903.2912574