# Data Cleaning Using Large Language Models


## Abstract

Data cleaning is a crucial yet challenging task in data analysis, often requiring significant manual effort. To automate data cleaning, previous systems have relied on statistical rules derived from erroneous data, resulting in low accuracy and recall. This work introduces Cocoon, a novel data cleaning system that leverages large language models for rules based on semantic understanding and combines them with statistical error detection. However, data cleaning is still too complex a task for current LLMs to handle in one shot. To address this, we introduce Cocoon, which decomposes complex cleaning tasks into manageable components in a workflow that mimics human cleaning processes. Our experiments show that Cocoon outperforms state-of-the-art data cleaning systems on standard benchmarks.


## 1 Introduction

Data cleaning is a well-known, challenging yet crucial task. Datasets frequently contain extreme or erroneous values, which can greatly affect outcomes of downstream analytics [21]. Consequently, it's well known that analysts spend over 80% of their time manually reviewing and cleaning data [8]. To tackle these challenges, previous systems aim at automating the data cleaning process [5, 20, 13].

However, traditional data cleaning methods struggle with low accuracy and low recall because they rely on statistical rules like thresholds, distributions, dependencies, denial constraints, etc., to detect anomalies [15, 2, 20]. The problem is that **these detection and cleaning rules are derived statistically** from unreliable, erroneous data, and therefore have low quality. On the other hand, data reflects real-world entities. When humans manually clean data, they can **semantically detect these rules and propose cleaning strategies** based on external real-world knowledge, resulting in much better performance. To demonstrate the limits of statistical rules and importance of semantic knowledge, consider the cleaning in the Rayyan table [16] as an example:

**Example 1.** Consider the data cleaning process for the 'article_language' column in Rayyan. The **statistical detection** analyzes the distribution of each unique value. It reveals that 46.4% of entries are "eng" and 9.5% are "English". As these strings don't exhibit strong distribution or pattern outliers, no **statistical error** is detected by past detection systems. However, when analysts manually clean the data, they **detect semantically** that "eng" and "English" are redundant representations of the same concept of english language. The **semantic cleaning** process cleans "English" $\rightarrow$ "eng" because "eng" is the most common representation, and similarly cleans other language values like "French" $\rightarrow$ "fre", "German" $\rightarrow$ "ger", and "Chinese" $\rightarrow$ "chi" in the 'article_language' column.

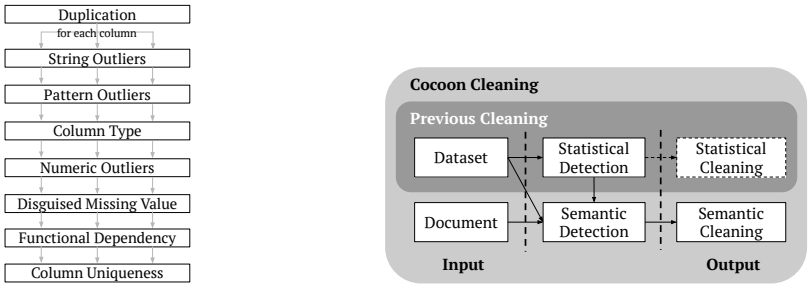
Such a process of applying semantic understanding of the tables and values for data detection and cleaning extends beyond just the data quality issue of inconsistent value representations. Section 2.1 catalogs the various data quality issues studied, and the detailed detection and cleaning steps.

To provide such semantic understanding without much human effort, rcent advancements in large language models, such as GPT-4 and Claude 3.5, have demonstrated near-human-level general capabilities across various tasks [3]. However, effectively prompting these models for data cleaning purposes remains a challenge due to the complex and ambiguous nature of data cleaning tasks. Our experimental results indicate that existing data cleaning tools [1, 18] utilizing LLMs achieve close to zero accuracy and recall for the majority of standard data cleaning benchmarks.

To this end, we introduce Cocoon[1], a data cleaning system that leverages LLMs for semantic understanding and combines the statistcal error detection for better context. To tackle the complexities

[1]Open sourced at `https://cocoon-data-transformation.github.io/page/clean`

Preprint. Under review.




(a) (b)

Figure 1: Cocoon decomposes data cleaning in two dimensions: (a) decompose it for different types of errors, for each column; (b) For each type of error, Cocoon decomposes the cleaning steps with traditional statistical detection, combined with semantic error detection and cleaning.

of data cleaning, our core approach is to decompose tasks into managable components; such approach is crucial for the accuracy and robustness of data tasks like visualization and transformation [12, 7, 10]. To design the decomposition for data cleaning (Figure 1), Cocoon mimics the cleaning steps a human would take: (1) we first breaks down the data cleaning into small common issues including duplication, missing values, and outliers [19, 4], (2) then for each data cleaning issue, we breaks down the task into statistical detection, semantic detection and semantic cleaning. Cocoon outperforms all other state of the art data cleaning systems on 4 out of 5 standard benchmarks, achieving higher F1 scores. For the one exception, we demonstrate that it is due to the ambiguity of the benchmark.

## 2 System Design

The challenge of applying LLMs to data cleaning is twofold: (1) data is too large to fit into the prompt of current LLMs, and (2) the whole data cleaning task is too complex for LLMs to perform in a single pass. To address (1), Cocoon leverages traditional statistical methods to profile the tables [11, 9] (e.g., value distribution, missing percentages) and includes these in the prompt to help LLMs better understand the data. To tackle (2), Cocoon decomposes data cleaning into separate processes for different types of errors (illustrated in Figure 1), motivated by how human decomposed data cleaning errors [19, 4]. We begin by discussing the decomposition, followed by the implementation details.

### 2.1 Task Decomposition

Cocoon detects and cleans the following types of data quality issues:

1. **Duplication**: The statistical error detection selects duplicated rows. If there are duplicates, we use an LLMs to determine if these duplications are semantically acceptable (e.g., duplication in logging with coarse time granularity). If it's erroneous, cleaning is performed by `SELECT DISTINCT`.
2. **String Outliers**: We sample frequent values and let LLMs review whether these values semantically contain typos or inconsistent representations. If errors are found, we ask LLMs to build a mapping from erroneous to correct values, and execute the cleaning through `CASE WHEN` clauses.
3. **Pattern Outliers**: We recursively ask LLMs to write a list of semantically meaningful regular expression patterns that cover all column values (e.g., `\d{2}/\d{2}/\d{4}` for dates is meaningful based on the day/month/year, but `.*` is not), and verify them with SQL. We then ask LLMs to assess if these there are inconsistent representations. Cleaning is via regex transformation.
4. **Column Type**: We identify the current column type from the database catalog and ask LLMs to suggest the most suitable data type semantically. For cleaning, we use `CAST` clauses.
5. **Numeric Outliers**: We capture the minimum and maximum values statistically and review the acceptable range semantically. We address outliers using a `CASE WHEN` clause for thresholding.
6. **Disguised Missing Value (DMV) [17]**: We show the column values and ask LLMs to identify values that currently not `NUUL`, but semantically means that the value are missing (e.g., string values like "N/A", "null"). Cleaning is performed using a `CASE WHEN THEN NULL` clause.

Table 1: Data cleaning performance (**P**recision, **R**ecall, and **F**-1) across different benchmarks. * Movies (4.6MB) is the largest datasets,and Holoclean runs out of memory, while CleanAgent doesn't accept files >2MB. Therefore, we benchmark them over the sample of first 1000 rows.

| System | Hospital | | | Flights | | | Beers | | | Rayyan | | | Movies | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | P | R | F | P | R | F | P | R | F | P | R | F | P | R | F |
| HoloClean | **1.00** | 0.46 | 0.63 | 0.73 | 0.34 | 0.47 | 0.05 | 0.04 | 0.04 | 0.53 | 0.67 | 0.59 | 0.00* | 0.00* | 0.00* |
| Raha+Baran | 0.91 | 0.60 | 0.72 | 0.84 | **0.61** | **0.70** | 0.97 | **0.96** | 0.96 | 0.83 | 0.35 | 0.50 | 0.85 | 0.75 | 0.80 |
| CleanAgent | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00* | 0.00* | 0.00* |
| RetClean | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.52 | 0.48 | 0.50 | 0.00 | 0.00 | 0.00 |
| Cocoon | 0.87 | **0.93** | **0.90** | **0.91** | 0.42 | 0.57 | **0.99** | **0.96** | **0.97** | **0.88** | **0.84** | **0.86** | **0.91** | **0.83** | **0.87** |

7. **Functional Dependency**: Following Baran, we only consider FDs where both left and right-hand sides have a single attribute. We compute the entropy measurement of each FD pair [2], and let LLMs review if these statistically strong functional dependencies are meaningful semantically. If meaningful, we identify all groups of values violating the functional dependency, ask LLMs to provide the correct mapping, and clean the data using a `CASE WHEN` clause.
8. **Column Uniqueness**: Some columns, e.g., primary key, should be unique. We compute the unique ratio of each column statistically and ask LLM to decide if the column should be unique semantically. For cleaning, we ask LLM to build a window function keyed on the relevant column, assuming some column contains information to prioritize records (e.g., the latest time).

Note that the order in which data quality issues are addressed is important. For example, resolving pattern outliers should precede fixing column type errors. This is because converting dates stored as strings is difficult if the date formats are inconsistent.

### 2.2 Implementation

Cocoon is implemented as a Python Library that connects to databases, and LLM APIs. Cocoon supports common databases including Snowflake, DuckDB, BigQuery, and SQL Server[2]. We support LLM APIs from Anthropic, Azure, Bedrock, VertexAI, and OpenAI.

To ensure that the error detection and cleaning processes are scalable, interpretable, and reusable, we perform them using SQL queries. The final output is a set of well-commented SQL queries.

Cocoon is designed to be a human-in-the-loop process for user feedback. For each error detection and data cleaning step, we present the LLM reasoning and ask humans to verify and adjust (details in Appendix A). In future work, such cleaning processes can take domain-specific documents as context.

## 3 Experiment

To evaluate the performance of Cocoon, we conducted experiments on 5 standard benchmarks.

### 3.1 Experiment Setup

All our experiments run on a Ubuntu 20.04 LTS machine with 16 virtual CPUs and 104 GB RAM. We run DuckDB and use Claude 3.5.

**Datasets.** We use 5 standard benchmarks representing a variety of domains and data quality issues:

- **Hospital** and **Flights** [20]: These datasets contain a variety of errors including typos, functional dependency violations, wrong column types and DMV.
- **Beers** [14]: This dataset includes functional dependency errors and column type errors.
- **Rayyan** [16] and **Movies** [6]: Besides column types and DMV, these real-world datasets contain many value misplacement errors like the county was incorrectly entered in the city column.

**Baselines.** We compare Cocoon against 4 baselines.

[2]Pattern outliers are not supported in SQL Server due to its limited pattern matching capabilities.

- **Cocoon.** Cocoon takes the database as input and outputs SQL. While the detection and cleaning process is intended to be HIL, we skip these and use the LLM provided ground truth.
- **Holoclean [20].** Holoclean additionally takes denial constraints as input, for which we provide the ground truth. However, it runs out of memory large datasets (Movies), so we use samples.
- **Raha [14] and Baran [13].** Raha first detects errors, and Baran cleans them. **Note that Baran additionally requires feedback on 20 clean cells. We provide the ground truth.**
- **CleanAgent [18].** CleanAgent inputs and outputs CSV files, primarily for standardization.
- **RetClean [1].** RetClean can accept additional tables as inputs, but we do not have any to provide.

**Evaluation.** Current benchmarks (1) are ambiguous and (2) don't handle column types and DMV:

- **Case Sensitivity**: Different cases are acceptable as long as the case is consistent across values.
- **Column Type**: Previous data cleaning systems, like our baselines, are limited because they operate on CSV files without rich column types. However, in databases, certain values, such as "yes"/"no," are better represented as bool. Cocoon casts them to "True"/"False", but for other data cleaning systems, we consider them correct even if they do not perform these casts.
- **DMV**: No baseline system casts DMV (e.g., "N/A") to NULL, but we still consider them correct.

We don't consider these, but Cocoon shows better results when accounting for them (Appendix B).

### 3.2 Results

**Performance.** Table 1 demonstrates that Cocoon outperforms all four baseline systems in terms of F1-scores except for the Flights dataset.

- **Flights Benchmark Ambiguity**: Cocoon achieves high precision but low recall for the Flights dataset due to benchmark ambiguity. This is because of the ambiguous FD: Flight Number $\rightarrow$ Actual Departure/Arrival Time. This FD is ambiguous because the original dataset frequently contains inconsistent departure and arrival times. For example, the actual arrival time for flight "AA-1733-ORD-PHX" is recorded as "10:30 p.m." in 5 rows, "10:31 p.m." in 4 rows, "10:28 p.m." in 3 rows, "10:39 p.m." in 1 row. It's nearly impossible to determine the true arrival time and clean the data accurately. Furthermore, such inconsistencies appear to be application issues rather than data cleaning issues, making it preferable to preserve these to represent the uncertainty.
- **HoloClean** performs poorly because its error detection relies heavily on integrity constraints provided by the user. Despite the provided ground truth constraints, most inconsistency issues (e.g., "oz" vs. "ounce" in Beers, and "100 min" vs. "1 hour 40 min" in Movies) cannot be adequately captured by these constraints.
- **CleanAgent & RetClean** utilize LLMs to clean data. However, CleanAgent achieves low results as it focuses on standardizing categories (e.g., email, phone, date). RetClean primarily cleans tables using external clean tables, which are not available. It only performs well on Rayyan because Rayyan contains a large number typos obvious for LLMs to fix.
- **Raha & Baran.** These demonstrate reasonable performance across all benchmarks. However, they use traditional ML models (e.g., Gradient Boosting, Adaboost), which are much less capable and lack the semantic understanding ability, compared to LLMs with billions of parameters. They also additionally require ground turth values provided by users.

## 4 Conclusion

In this work, we introduced Cocoon, a data cleaning system that leverages large language models to enhance semantic understanding in the data cleaning process. By decomposing complex data cleaning tasks into multi-step components that mimic human cleaning styles, Cocoon outperforms all other SOTA cleaning systems on 4 out of 5 benchmarks. Future work will address domain specific errors by exploring autonomous error classification and agent-based approaches.

## References


[1] Mohammad Shahmeer Ahmad, Zan Ahmad Naeem, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. Retclean: Retrieval-based data cleaning using foundation models and data lakes, 2023.

[2] George Beskales, Ihab F. Ilyas, and Lukasz Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proc. VLDB Endow.*, 3(1):197–207, 2010.

[3] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[4] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[5] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 458–469. IEEE, 2013.

[6] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. The magellan data repository. `https://sites.google.com/site/anhaidgroup/projects/data`.

[7] Victor Dibia. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. *arXiv preprint arXiv:2303.02927*, 2023.

[8] Wayne W Eckerson. Data quality and the bottom line. *TDWI Report, The Data Warehouse Institute*, pages 1–32, 2002.

[9] Will Epperson, Vaishnavi Gorantla, Dominik Moritz, and Adam Perer. Dead or alive: Continuous data profiling for interactive data science. *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[10] Zezhou Huang and Eugene Wu. Relationalizing tables with large language models: The promise and challenges. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2024.

[11] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 547–554, 2012.

[12] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

[13] Mohammad Mahdavi and Ziawasch Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment*, 13(12):1948–1961, 2020.

[14] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 865–882, 2019.

[15] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 75–86, New York, NY, USA, 2010. Association for Computing Machinery.

[16] Mourad Ouzzani, Hossam Hammady, Zbys Fedorowicz, and Ahmed Elmagarmid. Rayyan—a web and mobile app for systematic reviews. *Systematic Reviews*, 5(1):210, 2016.

[17] Abdulhakim A Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. Fahes: A robust disguised missing values detector. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2100–2109, 2018.

[18] Danrui Qi and Jiannan Wang. Cleanagent: Automating data standardization with llm-based agents, 2024.

[19] Erhard Rahm, Hong Hai Do, et al. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[20] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*, 2017.

[21] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes ai. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021.

## A User Interface

Cocoon is designed to be interpretable using the HIL process with NL explanations from LLMs. Specifically, we ask users to provide feedback on (1) the semantic detection results of error types, and (2) the semantic cleaning that maps incorrect values to clean values (see Figure 1).

Figure 2 shows the interface where users can specify and view the SQL results for the semantic cleaning of column types. The final output is aimed to be interpretable for long-term maintenance and scalable for existing data pipelines. We construct SQL queries documented with the reasoning behind the cleaning process. As shown in Figure 3, we provide examples of how to map values for string outliers, along with natural language descriptions.

Full illustrations of the cleaning results in HTML reports and commented SQL pipelines are available at `https://cocoon-data-transformation.github.io/page/clean`. In the future, we will conduct user studies to better understand how humans interact with Cocoon through natural language.

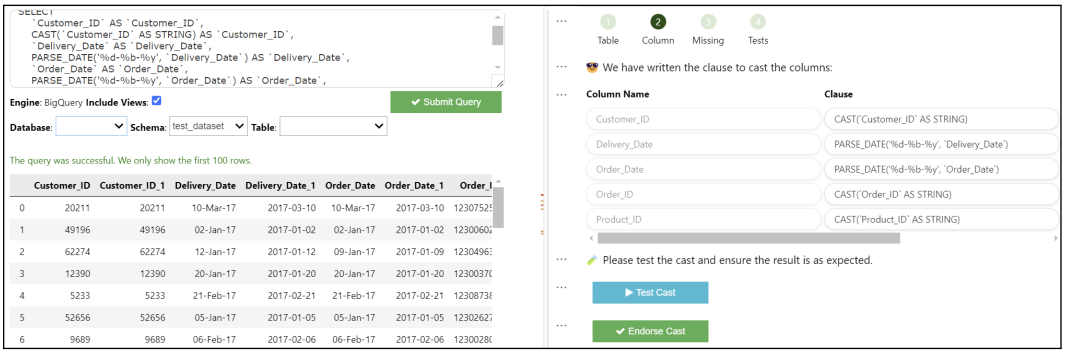
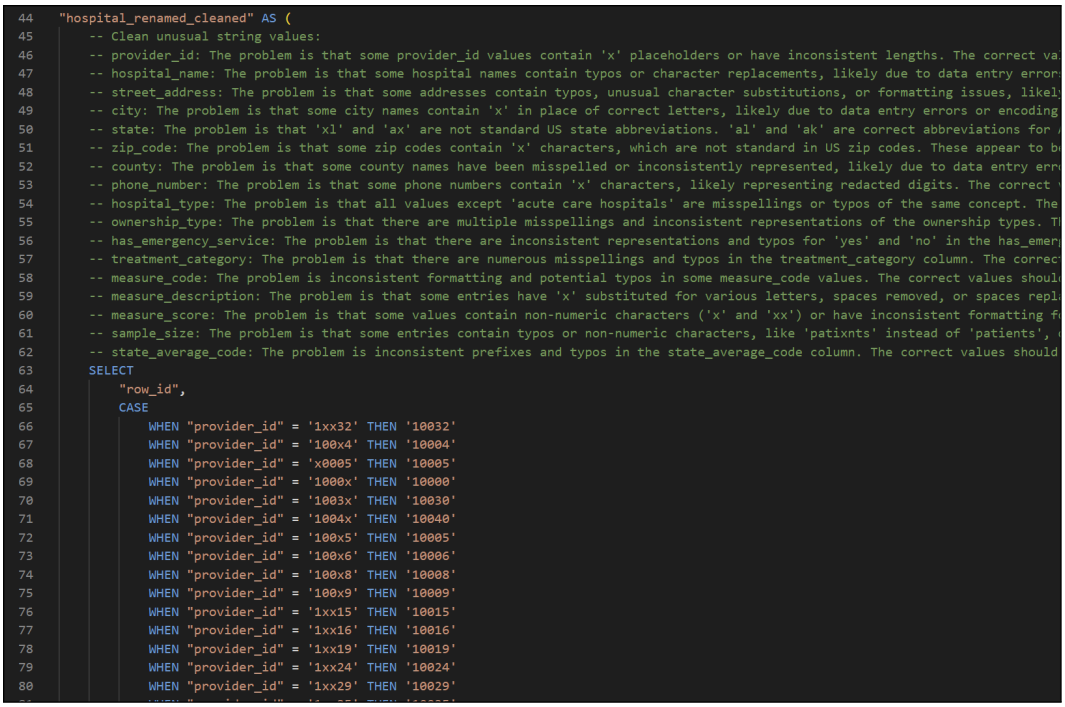



Figure 2: The UI for each data cleaning step. The right side is the interface where users specify the SQL clauses for column cast. The left side is the query interface to preview the results.

## B Error analysis

In this section, we present the error type details in the benchmarks. Previous data cleaning work focused on errors such as typos, functional dependency violations, and misplacements. However, we find that there are other types of errors in the benchmarks that are disregarded, specifically forms of disguised missing values [17] and column type errors. Table 2 shows the distribution of errors in the datasets of Hospital and Movies. Specifically, column type errors are very common. For example, for "EmergencyService" in the hospital dataset, the current values are "yes" and "no", which semantically



```
"hospital_renamed_cleaned" AS (
    -- Clean unusual string values:
    -- provider_id: The problem is that some provider_id values contain 'x' placeholders or have inconsistent lengths. The correct va
    -- hospital_name: The problem is that some hospital names contain typos or character replacements, likely due to data entry error
    -- street_address: The problem is that some addresses contain typos, unusual character substitutions, or formatting issues, likel
    -- city: The problem is that some city names contain 'x' in place of correct letters, likely due to data entry errors or encoding
    -- state: The problem is that 'xl' and 'ax' are not standard US state abbreviations. 'al' and 'ak' are correct abbreviations for
    -- zip_code: The problem is that some zip codes contain 'x' characters, which are not standard in US zip codes. These appear to b
    -- county: The problem is that some county names have been misspelled or inconsistently represented, likely due to data entry err
    -- phone_number: The problem is that some phone numbers contain 'x' characters, likely representing redacted digits. The correct
    -- hospital_type: The problem is that all values except 'acute care hospitals' are misspellings or typos of the same concept. The
    -- ownership_type: The problem is that there are multiple misspellings and inconsistent representations of the ownership types. T
    -- has_emergency_service: The problem is that there are inconsistent representations and typos for 'yes' and 'no' in the has_emer
    -- treatment_category: The problem is that there are numerous misspellings and typos in the treatment_category column. The correc
    -- measure_code: The problem is inconsistent formatting and potential typos in some measure_code values. The correct values shoul
    -- measure_description: The problem is that some entries have 'x' substituted for various letters, spaces removed, or spaces repl
    -- measure_score: The problem is that some values contain non-numeric characters ('x' and 'xx') or have inconsistent formatting f
    -- sample_size: The problem is that some entries contain typos or non-numeric characters, like 'patixnts' instead of 'patients',
    -- state_average_code: The problem is inconsistent prefixes and typos in the state_average_code column. The correct values should
    SELECT
        "row_id",
        CASE
            WHEN "provider_id" = '1xx32' THEN '10032'
            WHEN "provider_id" = '100x4' THEN '10004'
            WHEN "provider_id" = 'x0005' THEN '10005'
            WHEN "provider_id" = '1000x' THEN '10000'
            WHEN "provider_id" = '1003x' THEN '10030'
            WHEN "provider_id" = '1004x' THEN '10040'
            WHEN "provider_id" = '100x5' THEN '10005'
            WHEN "provider_id" = '100x6' THEN '10006'
            WHEN "provider_id" = '100x8' THEN '10008'
            WHEN "provider_id" = '100x9' THEN '10009'
            WHEN "provider_id" = '1xx15' THEN '10015'
            WHEN "provider_id" = '1xx16' THEN '10016'
            WHEN "provider_id" = '1xx19' THEN '10019'
            WHEN "provider_id" = '1xx24' THEN '10024'
            WHEN "provider_id" = '1xx29' THEN '10029'
```

Figure 3: Output SQL queries for results. We provide the cleaning reasoning as NL in the comments and use SQL for cleaning.

Table 2: Distribution of Various Types of Errors Across Benchmarks

| Dataset | Size | Typo | FD | Column Type | Inconsistency | DMV | Misplacement |
|---|---|---|---|---|---|---|---|
| Hospital | $1000 \times 19$ | 213 | 331 | 3,000 | – | 227 | – |
| Movies | $7390 \times 17$ | 184 | – | 14,433 | – | 131 | 938 |

means a boolean. Results in Section 3 don't consider these errors. Table 3 presents the results of Cocoon and the baseline systems when these are considered as errors. Cocoon outperforms all 4 baselines with >0.9 F1 score. This outcome is anticipated, as the precision and recall increase when new errors are introduced, and Cocoon effectively corrects them. Among all 4 baselines, only Raha partially solves the column type casting, as it asks for the ground truth samples and fixes "yes/no" -> bool. However, Raha struggles with more complex transformations for fields with higher cardinality, and transformations that necessitate semantic understanding. For instance, it fails to consistently convert time expressions like "1 hr. 30 min." and "90 min" into float 90.

Table 3: Comparison with other data cleaning systems.

| | Hospital | | | Movies | | |
|---|---|---|---|---|---|---|
| **Approach** | P | R | F | P | R | F |
| HoloClean | 1.00 | 0.13 | 0.24 | 0.00 | 0.00 | 0.00 |
| Raha | 1.00 | 0.97 | 0.98 | 0.57 | 0.55 | 0.56 |
| CleanAgent | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RetClean | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Cocoon | 0.99 | 0.99 | 0.99 | 0.96 | 0.91 | 0.93 |