arXiv:2503.06664v1 [cs.LG] 9 Mar 2025

EXPLORING LLM AGENTS FOR CLEANING TABULAR MACHINE LEARNING DATASETS

Tommaso Bendinelli[1,2] **Artur Dox**[2] **Christian Holz**[1]

[1]Department of Computer Science, ETH Zurich, Switzerland
[2] CSEM SA, Alpnach, Switzerland

ABSTRACT

High-quality, error-free datasets are a key ingredient in building reliable, accurate, and unbiased machine learning (ML) models. However, real world datasets often suffer from errors due to sensor malfunctions, data entry mistakes, or improper data integration across multiple sources that can severely degrade model performance. Detecting and correcting these issues typically require tailor-made solutions and demand extensive domain expertise. Consequently, automation is challenging, rendering the process labor-intensive and tedious. In this study, we investigate whether Large Language Models (LLMs) can help alleviate the burden of manual data cleaning. We set up an experiment in which an LLM, paired with Python, is tasked with cleaning the training dataset to improve the performance of a learning algorithm without having the ability to modify the training pipeline or perform any feature engineering. We run this experiment on multiple Kaggle datasets that have been intentionally corrupted with errors. Our results show that LLMs can identify and correct erroneous entries—such as illogical values or outliers—by leveraging contextual information from other features within the same row, as well as feedback from previous iterations. However, they struggle to detect more complex errors that require understanding data distribution across multiple rows, such as trends and biases.

1 INTRODUCTION

Despite being essential for achieving high model performance (Jäger & Biessmann, 2024), data cleaning remains one of the least engaging yet most time-consuming tasks for data scientists (Sambasivan et al., 2021). While many algorithms and approaches exist to speed up this process, significant challenges remain in efficiently deploying fully automated methods in real world scenarios (Ni et al., 2023). This is because devising a one-size-fits-all method that works universally for data cleaning is hard. What constitutes an error varies from dataset to dataset. As a result, the standard practice is to manually explore and inspect the dataset using domain knowledge, iteratively formulating hypotheses about potential erroneous entries, correcting these errors, and repeating the process until the dataset is deemed clean (Ridzuan & Zainon, 2019).

In recent years, LLMs have revolutionized various fields, including coding (Jiang et al., 2024), writing (Li et al., 2024a), information search (Zhu et al., 2023). Even within the field of ML—the very domain from which LLMs originate—researchers are exploring their potential to automate tasks such as feature selection, model tuning, and optimization (Huang et al., 2024a; Küken et al., 2024). Some studies suggest that these models have already surpassed median human performance for certain tasks (Chan et al., 2024). However, most studies have focused on boosting ML model performance by optimizing model architecture and feature processing, rather than on enhancing the quality of the raw data.

Therefore, in this work, we shift the focus to data quality. Specifically, we ask: **If we keep the pre-processing and training pipeline fixed, can an LLM improve model performance purely by detecting and correcting errors in the data?**

Our main contribution is the introduction of a simple framework for the systematic study of LLMs' ability to interact with datasets to achieve a specific goal—namely, improving the quality of training data for ML downstream tasks and to provide an initial analysis of the LLMs' performance.

2 Related Work

2.1 Data cleaning methods

Data scientists have a wide array of tools available for detecting and correcting errors and inconsistencies. Classical approaches rely on predefined rules and statistical techniques to identify issues in syntax, semantics, and outliers. These methods might involve checking for violations of integrity constraints, identifying duplicate entries, or applying outlier detection algorithms (Fan & Geerts, 2012). For error correction, problematic data is often replaced using external sources, basic statistical computations, or established integrity constraints (Ilyas et al., 2015). However, these techniques require careful parameter tuning and rigid rule definitions, making the process both time-consuming and labor-intensive. To alleviate these challenges, several approaches have been developed: Raha (Mahdavi et al., 2019) and HoloDetect (Heidari et al., 2019) focus on error detection, while Baran (Mahdavi & Abedjan, 2020) addresses error correction. These methods leverage ML to reduce or even eliminate the reliance on manually defined rules. Furthermore, these methods tend to also have better overall performance than classical methods, however they struggle in case of rare or complex error types (Ni et al., 2023). With the advent of LLMs, many works have begun exploring their utilisation for dataset cleaning motivated by the common sense reasoning of these models. Narayan et al. (2022) has explored using the GPT-3.5 architecture for various data wrangling tasks including error detection and data imputation, showing state of the art performance. Other works, such as Qi & Wang (2024) and Li et al. (2024b), also explore the use of LLMs for value standardization. Recently, IterClean (Ni et al., 2024) has claimed state of the art performance for entire detection-correction pipeline while using fewer supervised examples than previous methods. However, while the results are impressive, current evaluation has been restricted only to established benchmarks of data cleaning, and it still unclear to which extent LLM can automate the entire data cleaning pipeline (Majumder et al., 2024).

2.2 Benchmarking LLM powered agents on data science tasks

LLMs have been extensively studied for a variety of data science applications. The seminal work of Lai et al. (2023) benchmarked LLMs for code generation in data science using thousands of problems specifically targeting data science tasks. Other studies have expanded the range of tasks and improved evaluation methodologies (Zhang et al., 2024; Huang et al., 2024b; Galimzyanov et al., 2024).

Closely related to our work, several benchmarks move beyond simple question answering to enable function tool calling and multi-turn interactions. For example, Liu et al. (2023) evaluates LLMs on question answering tasks that involve tabular data. Similarly, Hu et al. (2024) proposes a framework for evaluating LLM-based agents on data analysis tasks—including data exploration, summary statistics, regression analysis, and feature engineering. Moreover, Majumder et al. (2024) introduces a comprehensive benchmark that formalizes the entire multi-step process of data-driven discovery—from hypothesis formulation and verification using both real-world and synthetic tasks across diverse domains—to systematically assess and improve LLMs' capabilities in automating scientific discovery. Likewise, Chen et al. (2024) presents a benchmark that assesses LLMs ability to autonomously generate executable Python programs for diverse, expert-validated data-driven scientific discovery tasks. Finally, approaches such as Huang et al. (2024a) and Chan et al. (2024) benchmark LLMs on ML competitions with explicit target of improving performance on a test set. However, the focus in these works is on feature engineering and machine learning, rather than on improving the quality of the training data.

3 Method

Our goal is to evaluate how LLMs can effectively detect and correct errors in *dirty* training datasets that negatively impact model performance on a held-out dataset. In our setup, the LLM has access to

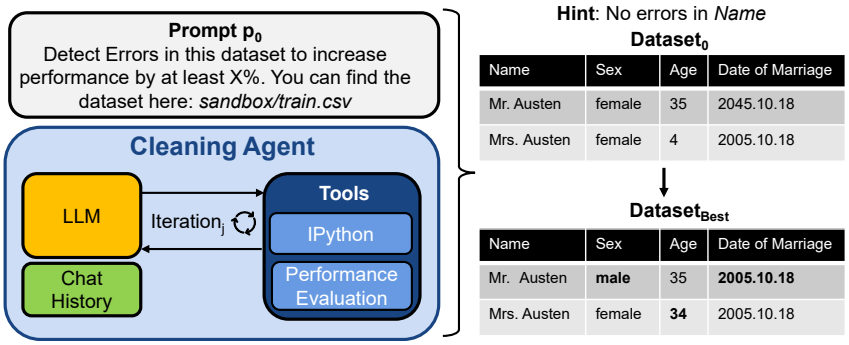

Figure 1: We provide the model with the path to the dataset along with a prompt instructing it to identify errors so that performance on a held-out set increases by a given threshold. At each iteration j, the LLM can send *code* to IPython to execute and get back the *sys.output* and/or send the *path* of the modified dataset $\mathcal{D}_{\text{i}}$ to get a *performance score*. The loop continues until the cumulative number of tokens used for the entire conversation reaches a pre-defined threshold. All the modified datasets $\mathcal{D}_{0...\text{i}}$ are stored and the dataset with the highest score is considered as $\mathcal{D}_{\text{Best}}$.

two tools. The first is the interactive Python shell IPython (Pérez & Granger, 2007), which executes Python code. Using this tool, the LLM can inspect and modify the dirty dataset used to train a ML model. The second is the performance evaluation. It can iteratively submit a *modified* version of the training dataset to an evaluation pipeline, which returns the performance score of the model trained on the this new data and evaluated on a *clean* held-out evaluation set. The pre-processing, training, and evaluation pipelines are kept fixed and non-modifiable by the LLM, ensuring that the LLM can influence model performance solely by altering the training dataset. Figure 1 shows an overview of our approach. In the following sections, we outline the procedure for generating dirty versions of the datasets and describe how the LLM interacts with them.

3.1 Dataset creation

While some datasets in the literature (Rekatsinas et al., 2017; Mahdavi et al., 2019) provide both clean and dirty versions, the data quality issues in the corrupted versions do not significantly impair the performance of downstream tasks (Ni et al., 2023).

To address this limitation, we create our own clean and dirty dataset versions as follows. First, we identify popular datasets from Kaggle. Each dataset, denoted as $\mathcal{D}$, is divided into a training set $\mathcal{D}_{\text{TrainClean}}$ and a testing set $\mathcal{D}_{\text{TestClean}}$, which serves as a held-out evaluation set that remains non-modifiable and unseen by both the model and the LLM. We train a model on $\mathcal{D}_{\text{TrainClean}}$ and evaluate its performance on $\mathcal{D}_{\text{TestClean}}$, establishing a baseline performance, P_{Clean}. Next, we introduce three systematic errors in $\mathcal{D}_{\text{TrainClean}}$, resulting in $\mathcal{D}_{\text{TrainDirty}}$. These errors are selected from the categories defined below (Numerical Shift, NaN Corruption, and Categorical Shift). We re-train the model on $\mathcal{D}_{\text{TrainDirty}}$ and again evaluate its performance on $\mathcal{D}_{\text{TestClean}}$, yielding a performance score of P_{Dirty}.

The difference between P_{Clean} and P_{Dirty} represents the maximum potential performance improvement achievable by correcting the introduced systematic errors. However, P_{Clean} is not strictly an upper bound on performance, as other inherent issues within the Kaggle datasets may exist that, if addressed, could lead to even higher performance.

We define the following three categories of systematic errors introduced in $\mathcal{D}_{\text{TrainClean}}$:

- *Numerical Shift*: Alters numerical values to introduce a distribution shift in a subset of the dataset (e.g., increasing all age entries by 10 years or restricting them to a range between 0 and 10).
- *NaN Corruption*: Replaces a fraction of well-defined values with NaN in a subset of the dataset, following a missing at random corruption strategy (e.g., corrupting with NaN the values of a specific column where a specific condition happens).

- *Categorical Shift*: Shifts the distribution of categorical feature values within a subset of the dataset, leading to statistical or contextual inconsistencies (e.g., changing the category of a feature to one that is uncommon or contextually inappropriate for that subset).

In order to create a fair evaluation scenario, we also ensure that the introduced errors meet the following criteria:

1. They are detectable through statistical analysis, contextual understanding of the dataset, or a combination of both; and
2. The resulting decrease in model performance due to dataset corruption can be mitigated through dataset manipulations that do not require access to the original *clean* dataset.

Additionally, to gain preliminary insights into how humans compare to LLMs on this task, we asked two data scientists to solve the same task as the LLMs on each dataset within a time limit of one hour and recorded their peformance improvement over P_{Dirty}. This also offers an initial perspective on the perceived difficulty of detecting and correcting errors in our datasets. Details on the datasets and the introduced errors can be found in the Appendix A.1.

3.2 Cleaning Agent

We provide an initial prompt P_0 to the LLM, which includes clear instructions for the task, examples of inconsistencies, the path to $\mathcal{D}_{\text{TrainDirty}}$, explanations of the available tools, and guidance on how to solve the task effectively. The full prompt is provided in the Appendix A.2. The LLM has access to the following two tools:

- **Performance Evaluation (Input**: *DatasetPath*, **Outptut**: *PerformanceScore*): This tool submits the dataset modified by the LLM and evaluates it. It takes as input the path to $\mathcal{D}_{\text{TrainModified}_i}$, performs pre-defined pre-processing steps, trains the learning algorithm on it, and returns the model's performance score on a classification task using $\mathcal{D}_{\text{TestClean}}$. Note that the LLM cannot modify the evaluation pipeline or add new columns, though it is permitted to drop columns or rows if necessary.
- **IPython (Input:** *PythonCode*, **Output:** *StandardOutput*): Executes the code provided as input, using the interactive IPython shell and returns the standard output produced during execution. This includes the output of print statements, the contents of defined variables, and any errors encountered. The shell preserves a persistent session state across executions, so variables and outputs from earlier runs remain available in subsequent executions.

The LLM's interactions with the dataset begin after the initial prompt P_0 is provided. An iteration j starts when the LLM generates text, and in each iteration, the LLM may invoke one or both tools. The responses from these tools, along with the previous prompt and the LLM's output from the current iteration, are appended to form the prompt for the next iteration (i.e., $p_{j+1} = p_j + output_j + ToolResponse_j$). If the LLM wants to submit a modified version of the training dataset $\mathcal{D}_{\text{TrainModified}_i}$, it must save the dataset and provide its path to the *Performance Evaluation* tool, which then returns the score for that submission. The loop terminates when the cumulative number of tokens used (input plus output, summed over all iterations) reaches a predefined threshold, and the dataset submission with the highest score is considered the LLM's best result.

4 Experiments and Results

4.1 Setup

We consider three popular datasets from Kaggle: Titantic, Meat Consumptions, Hotel Bookings. A classification task is defined for each dataset. We perform minimal pre-processing and ensure the classification tasks remain challenging by removing columns that contain information which would make the tasks trivial. A standard train/test split is performed, and, as outlined in Section 3.1, three different types of systematic errors are introduced into the training data. Each of these errors requires investigating rows, columns, or both within the dataset. Further details on the errors are provided in the Appendix A.1. We conduct experiments using the following models with the function

calling feature: `gpt-4o`, `o3-mini-2025-01-31`, `claude-3-5-sonnet-20241022`, and `gemini-2.0-flash-exp`. Each experiment is run with a limit of 200k tokens and repeated six times to ensure consistency. Our code structure is based on Swarm from OpenAI (2024) due to its lightweight design and flexibility. Table 1 provides an overview of the different datasets including the data scientist performance and the type of errors introduced.

Dataset	**Shape**	**Data Scientist Perf. Improvement [1h]**	**Error Types**
Titanic	(408, 12)	6.2% *(easy)*	Numerical Shift Categorical Shift
Meat Consumption	(9160, 10)	2.5% *(medium)*	Numerical Shift
Hotel Bookings	(100000, 28)	0% *(hard)*	Numerical Shift Nan Corruption Categorical Shift

Table 1: Comparison of the three Kaggle datasets in terms of shape, the performance improvement achieved by humans in one hour and error types. The perceived difficulty of each dataset (*easy*, *medium*, *hard*) as reported by the human participants is also provided.

4.2 RESULTS

In the following sections, we first present our quantitative results alongside a qualitative discussion to interprets them. Next, we analyze how cumulative token consumption and the strength of provided hints affect performance improvements. Finally, we describe recurring issues observed in the model interactions.

4.2.1 HOW EFFECTIVE ARE LLMS AT IMPROVING MODEL PERFORMANCE BY CORRECTING ERRORS?

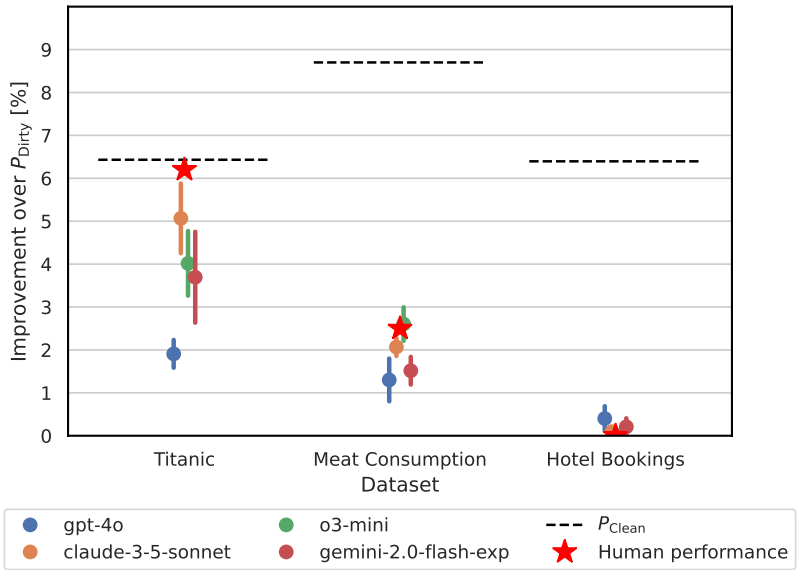

Figure 2: Performance improvement over P_{Dirty} for the four models and three datasets.

Figure 2 shows that no model reaches the maximum potential performance improvement, i.e., achieving P_{Clean}. A significant gap between the achieved performance improvement and P_{Clean} is noticeable for both the *Meat Consumption* and *Hotel Bookings* datasets, highlighting their increased difficulty. Specifically, for *Hotel Bookings*, all models achieve less than 1% improvement, which

emphasizes the challenging errors introduced in this dataset. Interestingly, the perceived difficulty of the datasets by humans is also reflected in the LLMs' performance, with the *Titanic* dataset being the easiest, followed by *Meat Consumption* and *Hotel Bookings*. Overall, while *o3-mini* and *claude-3-5-sonnet* show a slight edge over the others for the *Titanic* and *Meat Consumption* datasets, no model clearly outperforms the rest.

To better understand where the models face challenges, we manually inspect the conversation traces from the runs. Listing 1 shows the code generated in the first response by *o3-mini* during a sample interaction on the *Meat Consumption* dataset. We observe that the models begin their analysis by exploring the data using pandas functions such as `.describe()`, `.info()`, and `.head()`, along with performing simple sanity checks on individual entries. However, models rarely investigate more complex relationships within the data.

```
print(df.head())
print(df.describe(include='all'))

columns_to_check = ['Poultry', 'Beef', 'Sheep and goat', 'Pork', 'Other meats', 'Fish and seafood']
print('\nNegative value counts:')
for col in columns_to_check:
    if col in df.columns:
        neg_count = (df[col] < 0).sum()
        print(f'{col}: {neg_count}')
    else:
        print(f'{col} not found in dataset')

print('\nMax values in consumption columns:')
for col in columns_to_check:
    if col in df.columns:
        max_val = df[col].max()
        print(f'{col}: {max_val}')

print('\nMissing value counts:')
print(df.isnull().sum())
```

Listing 1: Python code generated in a first response by o3-mini on the Meat Consumption dataset.

As a result, errors that require the analysis of multiple values in the dataset—such as identifying distribution shifts, trends, or biases— go undetected. Even when errors are identified, limited exploration across different values hinders effective error mitigation. For example, no model was able to detect the numerical shift introduced in the *Hotel Bookings* dataset, where 10 was added to all values in the `lead_time` column only for the year 2016. This represents a simple bias that the models failed to identify. Listing 2 shows an example from the *Meat Consumption* dataset, where *claude-3-5* correctly detects that some landlocked countries, such as Afghanistan and Nepal, have abnormally high fish consumption. However, instead of applying a statistically meaningful approach, such as using quantiles, it applies an arbitrary scaling factor.

```
landlocked_countries = ['Afghanistan', 'Mongolia', 'Nepal', 'Bhutan', 'Laos', 'Uganda']
for country in landlocked_countries:
    mask = (df_cleaned['Entity'] == country) & (df_cleaned['Fish and seafood'] > 15)
    df_cleaned.loc[mask, 'Fish and seafood'] = df_cleaned.loc[mask, 'Fish and seafood'] * 0.3
```

Listing 2: Successful error detection, but arbitrary error mitigation by claude-3-5 in the Meat Consumption dataset.

We observe the same behavior for categorical variables. For example, in the *Hotel Bookings* dataset, the NaN corruption involved replacing 70% of entries with NaN where `country == 'PRT'` in the years 2016 and 2017. All models, except for *gpt-4o* in a single run, fail to address this error correctly. Instead of recognizing that `PRT` is the most frequent country based on the available data from 2015, they replace the missing values with a new entry labeled *Other*, as shown in Listing 3.

```
# Fix negative adr: replace negative adr with median of non-negative adr
adr_median = df.loc[df['adr'] >= 0, 'adr'].median()
df.loc[df['adr'] < 0, 'adr'] = adr_median
# Fill missing values
# For country, fill missing with 'Other'
df['country'] = df['country'].fillna('Other')
```

Listing 3: Arbitrary error mitigation by o3-mini in the Hotel Bookings dataset.

4.2.2 How many tokens are needed to achieve the best performance?

Figure 3 shows the cumulative tokens consumed, ranging from 25k up to a maximum of 200k, in relation to the best achieved improvement in performance. Across all models and datasets, a

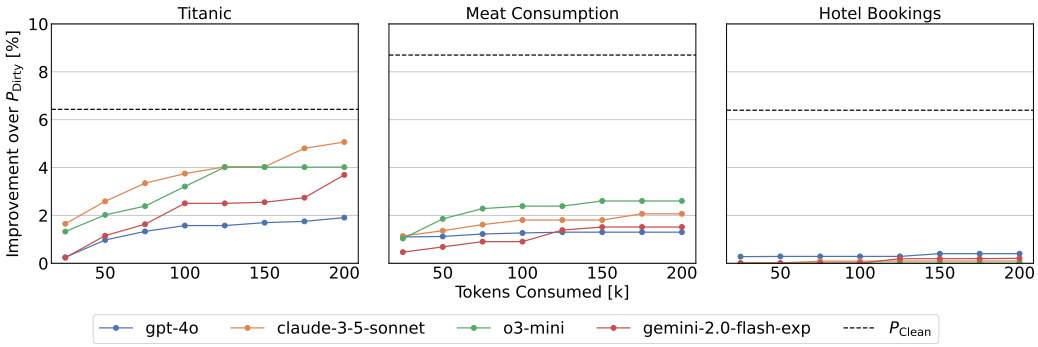

Figure 3: Performance improvement for different *Cumulative tokens* thresholds from 25k to 200k

logarithmic trend is observed, indicating diminishing returns as token consumption increases. Given that *gemini-2.0-flash-exp* supports a context window of up to 1M tokens, we conduct an additional experiment by increasing the cumulative token consumption limit to 2M for this model. Table 2 shows that, for the *Titanic* and *Hotel Bookings* datasets, performance increases by 5.19% and 1.82%, respectively, compared to the results with a 200k token limit. This suggests that the *gemini-2.0-flash-exp* benefits significantly from the larger context window, possibly allowing for more thorough exploration and error correction.

Dataset	**200k Tokens**	**2M Tokens**
Titanic	3.69%	8.88%
Meat Consumption	1.51%	1.79%
Hotel Bookings	0.08%	1.90%

Table 2: Performance improvements of *gemini-2.0-flash-exp* with 200k and 2 million cumulative token limits across all datasets.

4.2.3 DO HINTS HELP?

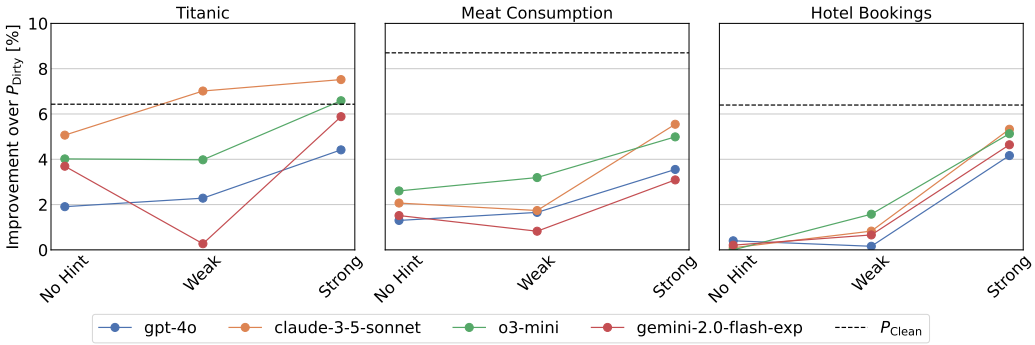

Figure 4: Impact of providing *no hint*, a *weak* hint, and a *strong* hint on the performance improvement for all models and datasets.

Figure 4 shows how performance improvement across all models and datasets changes depending on how much additional context is provided to the model in the initial prompt P_0. The models are evaluated under three conditions: no hint, a *weak* hint, or a *strong* hint. A weak hint provides partial information about the location of the error in the dataset, while a strong hint offers complete information about the error's location and partial guidance on how to correct it. Examples of these hints for all datasets are provided in the Appendix A.1.

Overall, a clear trend emerges: providing hints generally enhances performance, as the models can leverage the extra contextual information to improve error detection and correction. However, there are exceptions. We noticed that, on some occasions, the weak hint causes the model to apply an inappropriate error correction strategy to a column—one that actually decreases performance—whereas, with no hint, that column would remain unchanged.

4.2.4 Additional Remarks

Beyond the points mentioned earlier, our analysis of the conversation traces revealed several recurring issues common to all models. In particular, we observed:

- Models frequently fail to submit a valid dataset. As shown in Table 3, a significant percentage of submissions contain errors. Major reasons include providing a file path that does not exist and submitting datasets with extra columns.
- Models tends to apply a brute-force approach by repeatedly submitting datasets without meaningful analysis. Table 4 shows the fraction of dataset submissions relative to the generated code, highlighting that in the majority of iterations, the models prioritize submitting datasets over generating code for meaningful analysis.
- Models (with the exception of gpt-4o) seems to ignore that the state is preserved in IPython. Specifically, in each tool call, the code is generated from scratch rather than building on outputs from previous executions. This hinders in-depth exploration of the dataset across multiple iterations.

Model	**Column Violation [%]**	**Dataset not found [%]**	**Other [%]**	**Total Failures [%]**
claude-3-5-sonnet	8.84	0.00	4.69	13.53
gemini-2.0-flash-exp	5.05	4.32	3.73	13.10
gpt-4o	5.87	0.00	2.07	7.94
o3-mini	0.00	5.40	5.47	10.87

Table 3: Percentage of invalid submissions for each model. *Column Violation* indicates that new columns were added to the *modified* dataset, violating the rules described in the initial prompt P_0. *Dataset not found* means the dataset path provided by the LLM to the evaluation function does not exist. *Other* refers to additional violation errors.

Model	**Dataset related submissions over total tool calls [%]**
claude-3-5-sonnet	61.86%
gemini-2.0-flash-exp	63%
gpt-4o	29%
o3-mini	84.70%

Table 4: Percentage of calls to the IPython tool related to the creation of a new submission dataset compared to the total number of calls.

5 Discussion

5.1 Conclusive Remarks

We present a simple strategy to benchmark LLMs in one of the most crucial yet often overlooked activities of data scientists: cleaning data prior to model training. We propose a pipeline in which the LLM has access to IPython to programmatically modify a training dataset that has been corrupted with errors. The processes of training the ML model and performing any feature engineering are kept fixed and are not modifiable by the LLM. The LLM can *iteratively* explore the dataset to

detect and correct errors, receiving feedback based on the model's performance when trained on its submitted modified datasets and evaluated on a clean, held-out test set. Our goal is to explore the strength and limitations of the current state-of-the-art LLMs on this cleaning task across different datasets. Our results show that while LLMs can detect and mitigate errors in two of the three datasets considered, none of them achieve the maximum potential performance improvement achievable by fully correcting the introduced systematic errors. Overall, providing the LLM with hints about the errors leads to improved performance, indicating that the models can effectively process contextual information to solve the given task. An analysis of the conversation traces reveals that LLMs are able to identify and correct errors that involve investigating *single* values or *individual* rows. However, they struggle with errors spanning multiple rows, such as distribution shifts, trends, or biases.

5.2 Limitations and Future Work

Our work can be extended from multiple angles. First, the current setup enforces text-based communication between the LLM and the tools. This could be extended to allow the LLM to receive visual artifacts, such as plots generated from code, which are often essential during the data exploration phase. Second, our current approach involves manually introducing and validating errors, which is labor-intensive and limits the scalability of the benchmark to a wider range of datasets. Moreover, since only the authors were involved in creating errors for our three datasets, there is a potential risk of limited diversity and biases influencing our findings. Future work should focus on automating error generation to ensure a diverse and realistic set of errors across various datasets, thereby mitigating potential biases. Third, some of the insights in this work were derived from manually inspecting the raw messages exchanged at each iteration between the LLM and the tools. Future research should focus on automating this process to enable quantifiable measurements and address the following questions: 1) What types of failures affect the LLM, such as reasoning flaws, ineffective exploration, or other factors? and 2) At which stages these failures occur most frequently—whether during error detection or correction. Finally, the impact of different prompting approaches should be explored. Specifically, future research should investigate how varying prompts influence the LLM's performance, as well as its error detection and correction strategies. We plan to extend this work by addressing these limitations.

References

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. *arXiv preprint arXiv:2410.05080*, 2024.

Wenfei Fan and Floris Geerts. *Foundations of data quality management*. Morgan & Claypool Publishers, 2012.

Timur Galimzyanov, Sergey Titov, Yaroslav Golubev, and Egor Bogomolov. Drawing pandas: A benchmark for llms in generating plotting code. *arXiv preprint arXiv:2412.02764*, 2024.

Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pp. 829–846, 2019.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*, 2024.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. In *Forty-first International Conference on Machine Learning*, 2024a.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, et al. Da-code: Agent data science code generation benchmark for large language models. *arXiv preprint arXiv:2410.07331*, 2024b.

Ihab F Ilyas, Xu Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

Sebastian Jäger and Felix Biessmann. From data imputation to data cleaning—automated cleaning of tabular data improves downstream predictive performance. In *International Conference on Artificial Intelligence and Statistics*, pp. 3394–3402. PMLR, 2024.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

Jaris Küken, Lennart Purucker, and Frank Hutter. Large language models engineer too many simple features for tabular data. *arXiv preprint arXiv:2410.17787*, 2024.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Pre-trained language models for text generation: A survey. *ACM Computing Surveys*, 56(9):1–39, 2024a.

Lan Li, Liri Fang, and Vetle I Torvik. Autodcworkflow: Llm-based data cleaning workflow auto-generation and benchmark. *arXiv preprint arXiv:2412.06724*, 2024b.

Tianyang Liu, Fei Wang, and Muhao Chen. Rethinking tabular data understanding with large language models. *arXiv preprint arXiv:2312.16702*, 2023.

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.

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*, pp. 865–882, 2019.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. Discoverybench: Towards data-driven discovery with large language models. *arXiv preprint arXiv:2407.01725*, 2024.

Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911*, 2022.

Wei Ni, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, and Jianwei Yin. Automatic data repair: Are we ready to deploy? *arXiv preprint arXiv:2310.00711*, 2023.

Wei Ni, Kaihang Zhang, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, and Jianwei Yin. Iterclean: An iterative data cleaning framework with large language models. In *Proceedings of the ACM Turing Award Celebration Conference-China 2024*, pp. 100–105, 2024.

OpenAI. Swarm, 2024. URL `https://github.com/openai/swarm`.

Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.53. URL `https://ipython.org`.

Danrui Qi and Jiannan Wang. Cleanagent: Automating data standardization with llm-based agents. *arXiv preprint arXiv:2403.08291*, 2024.

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

Fakhitah Ridzuan and Wan Mohd Nazmee Wan Zainon. A review on data cleansing methods for big data. *Procedia Computer Science*, 161:731–738, 2019.

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*, pp. 1–15, 2021.

Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking data science agents. *arXiv preprint arXiv:2402.17168*, 2024.

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107*, 2023.

A APPENDIX

A.1 DATASETS

In the following, we briefly describe the three errors introduced in the Titanic, Meat Consumption, and Hotel Bookings datasets from Kaggle, along with the corresponding hints (*weak*, *strong*) provided for each.

A.1.1 TITANTIC

Errors:

1. The values of the column `Sex` of 50% of female survivors (identified by Miss. or Mrs. in their names) are randomly changed to male. This constitutes a *categorical shift*.
2. The values of the column `Age` are assigned unrealistically low ages (between 2 and 8 years old) for 50% of married female non-survivors. This constitutes a *numerical shift*.
3. Reduces the values of the column `Fare` to 10% of their original amounts for passengers with high status titles such as Dr. or Lady in their names. This constitutes a *numerical shift*.

Hints (weak, strong):

- "Errors are in the `Sex`, `Age` and `Fare` columns."
- "Errors are here: Female survisors had their `sex` entry corrupted, The same happened for the `age` of female married non-survivors, and the `fare` of some passengers with high social status was corrupted."

A.1.2 MEAT CONSUMPTION

Errors:

1. The values for the column `Poultry` are set to near-zero (random values between 0 and 0.1) for all `countries` in specific `years` (1986, 1990, 1993, 1995, 2000, 2005, 2010, 2015), simulating missing or drastically reduced data for those periods. This constitutes a *numerical shift*.
2. The values for the columns `Fish and Seafood` consumption for landlocked `countries`—Afghanistan, Burkina Faso, Chad, Burundi, Central African Republic, Niger, Nepal, Mali, Tajikistan, Uzbekistan, and Kyrgyzstan—are set between the 85th and 95th percentiles of the dataset for all years, creating unrealistically high fish and seafood consumption that does not match real-world patterns. This constitutes a *numerical shift*.
3. The values for the total meat consumption (`Poultry, Beef, Sheep and goat, Pork, Other meats, Fish and seafood`) for Mauritius, Italy, Japan, Vietnam, China, and Mexico are progressively increased by 30% each `year` between 1997 and 2004, resulting in unrealistically high meat consumption sums during these years. This constitutes a *numerical shift*.

Hints (weak, strong):

- "Errors are observed in 1) certain `years` [1986, 1990, 1993, 1995, 2000, 2005, 2010, 2015]), 2) In some `countries` regarding `fish and seafood` consumption, and in consecutive `years` for the following countries Mauritius, Italy, Japan, Vietnam, China, Mexico."
- "Observed errors:
    1. In the `years` [1986, 1990, 1993, 1995, 2000, 2005, 2010, 2015], `poultry` consumption is significantly underreported.
    2. In landlocked `countries` such as Afghanistan, Burkina Faso, Chad, Burundi, Central African Republic, Niger, Nepal, Mali, Tajikistan, Uzbekistan, and Kyrgyzstan, `fish and seafood consumption` is reported to be excessively high.

3. In `countries` like Mauritius, Italy, Japan, Vietnam, China, and Mexico, the total meat consumption is notably overreported during the `years` [1997, 1998, 1999, 2000, 2001, 2003, 2004]."

A.1.3 HOTEL BOOKINGS

Errors:

1. The values for the column `lead_time` are increased by 10 days for all records in the `year` 2016, resulting in consistently inflated booking `lead times` for that `year`. This constitutes a *numerical shift*.
2. Changes the `deposit_type` to `Non Refund` for all bookings made through the TA/TO `distribution channel` in 2017, creating an unrealistic scenario where all such `reservations` are non-refundable. This constitutes a *categorical shift*.
3. Replaces 70% of the `country` entries with NaN values in the `country` column to PRT (Portugal) for all years except 2015, although `PRT` is the most frequent country based on the available data from 2015. This constitutes a *NaN corruption*.

Hints (weak, strong):

- "Errors are in the `lead_time`, `deposit` and `country` columns, there are no errors in any entries from 2015."
- "Errors are here: There is a systematic bias in the `lead_time` of 2016, the deposit with `distribution_channel` TA/TO looks wrong in 2017 and often when people arrive from PRT, the `country` is not recorded."

A.2 INITIAL PROMPT P_0 PROVIDED TO THE MODEL

```
You have been provided with a dataset containing structural errors that degrade your model's performance. Your
       goal is to identify and correct these errors so that the model's F1 score increases from {metadata["
      performance_test"]} to at least {goal}. The target column is {metadata["target_column"]}.

1. Locate and fix any inconsistent or illogical values in the data. Examples of inconsistencies might include
      negative values where they cannot logically occur, incorrect category assignments, or strange drift in
      the data.

2. Your training dataset is located in sandbox/train.csv.

3. Each time you attempt a fix, save your cleaned dataset as train_cleaned_v*.csv with an updated version
      number (e.g., train_cleaned_v1.csv, train_cleaned_v2.csv, etc.).

4. After updating and saving the cleaned dataset, call submit_clean_data() to evaluate the model on the test
      set.

5. You may run code to explore and fix the dataset with execute_code_ipython_shell(), but you cannot:
- Plot graphs or use graphical libraries.
- Add new columns to the dataset.
- Modify or drop the _competition_index column (doing so will break the evaluation process).
- Add new rows to the dataset.

6. Focus on increasing the model's F1 score by correcting structural issues. These issues typically involve
      values that are inconsistent with the dataset's real-world context.

Tip: You need to understand the dataset to identify and correct errors. Don't just submit attempts blindly.

The dataset description is as follows:
{dataset_description}

This is the preprocessing pipeline code before training:
```python
{metadata["pipeline_code"]}
```

Hint: {Hints}
```

Where metadata["pipeline_code"] is dataset specific