DATAEXPLAINER: Comprehensible Agentic Data Science with Consumer-Grade Hardware

João Paulo Paiva Lima[1], **Marluce Rodrigues Pereira**[2], **Denilson Alves Pereira**[1]

[1] Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA)
Caixa Postal 3037 – 37203-202 – Lavras – MG – Brazil

[2]Departamento de Computação Aplicada – Universidade Federal de Lavras (UFLA)
Caixa Postal 3037 – 37203-202 – Lavras – MG – Brazil

Abstract. *Data Science (DS), a complex field often involving advanced knowledge of statistics, mathematics, and programming, has historically been restricted to deeply knowledgeable human professionals. Recent advancements in artificial reasoning and the expanding toolset of large language models (LLMs) have challenged this paradigm. However, most LLM-based approaches for automated DS are still computationally demanding, time-consuming, and not especially easy to follow. In this work, we expand on previous advancements to present* DATAEXPLAINER*, a framework for automated DS that provides step-by-step solutions and surpasses the average Kaggle user while using a single consumer-grade GPU and less than 2 hours of processing.*[1]

1. Introduction

Data science (DS) and analysis constitute a fundamental element in both science and technology. It enables the discovery of valuable insights within complex or seemingly unrelated information, providing a consistent, replicable, and robust framework for data management. This is particularly relevant in fields involving intricate systems and requiring a nuanced approach, such as medicine, epidemiology, physics, and chemistry [Provost and Fawcett 2013].

In recent years, the scope, scale, and importance of data science and analysis have grown significantly. More specifically, computerized systems have led to a shift in how data is managed and analyzed. Digital storage technologies, such as hard drives and cloud platforms, are far more efficient, reliable, and easier to index than more traditional methods. The widespread adoption of the internet has revolutionized data accessibility, enabling rapid collection, sharing, and retrieval of information from almost anywhere in the world. Consequently, recent estimates suggest that the total amount of data on the web, encompassing files, images, videos, and more, could exceed 180 zettabytes by the end of 2025 [Taylor 2024]. This unprecedented increase in data generation, storage, and availability gave rise to the phenomenon commonly referred to as Big Data [Russell and Norvig 2022], characterized by vast, varied, and highly indexable datasets capable of holding valuable insights. However, the sheer scale and diversity of these datasets might present a significant challenge: since traditional data science is often a manual process, tailored by highly specialized professionals, it might be unable to handle such overwhelming volumes of information in a efficient manner. In this context it is

[1]The entire codebase used for the evaluations is provided at `https://github.com/joaopaulo7/DataExplainer-paper`

fundamental to develop more efficient and scalable solutions to handle such a deluge of data.

A promising approach involves equipping Large Language Models (LLMs) with tools to perform traditional analysis. This strategy enables language models to write and execute code in a controlled environment, leveraging their strengths in text and code generation to actively navigate the complex solution space of DS. Such approach would be an example of the blossoming field of LLM agents, which focus on integrating models into structured, task-oriented, agentic frameworks [Huang et al. 2024b]. However, the currently available automated DS solutions often involve extensive search algorithms and a large number of trial-and-error steps, which makes the process costly and difficult to follow. Additionally, small-to-mid sized and open-source models are rarely subject to evaluation, with most solutions being optimized for server-sized models with hundreds of billions of parameters [Guo et al. 2024, Chan et al. 2025, Chi et al. 2024, Jiang et al. 2025], further increasing the overall compute cost.

To address such limitations, we present a new automated DS framework based on the DATA INTERPRETER framework by [Hong et al. 2025], but with a larger focus on consumer-grade hardware and comprehensible, step-by-step, solutions. We named the proposed framework DATAEXPLAINER as a reference to its most distinctive feature, the explanation steps. It expands the original DATA INTERPRETER by including explanation steps during the analysis process; generated by the LLM as markdown cells, these explanatory steps are a way to more clearly guide and explain the solution process. To ensure its effectiveness, we evaluated its performance for solving seven varied machine learning engineering (MLE) competitions from the Kaggle website[2]. Hence, we list the main contributions of this work as:

- The expansion of a previous framework to create a new automated DS framework focused on comprehensible solutions with step-by-step descriptions, which we named DATAEXPLAINER.
- An evaluation of the aforementioned framework for four different LLMs, showing it improves on the original's scores and achieves above-average placement in Kaggle competitions.
- An evaluation of three open-source LLMs for agentic DS, running on consumer-grade hardware and using under 24GB of VRAM, and a comparison to a larger proprietary model.

2. Literature Review

Early work on generative models applied to data science focused on code generation and correction through prompting [Lai et al. 2023, Chandel et al. 2022]. With code-tuned LLMs such as DeepSeek-Coder [Zhu et al. 2024] and Qwen-Coder [Hui et al. 2024] becoming increasingly cost-effective, these approaches serve as valuable tools for reducing the time invested in trivial tasks. However, human expertise remains essential, as the user must still guide the analysis even with LLMs' assistance.

The field of automatic machine learning (AutoML) [Hutter et al. 2019] provides a more extensive form of automation. The goal of AutoML is to automatically search

[2]https://www.kaggle.com/

for the best machine learning setup to process a given dataset. As with most search-related problems, its application can often be resource-intensive and susceptible to getting trapped in local minima. To address these shortcomings, recent works have proposed using LLM agents to more effectively guide the search process. Agents such as DS-Agent [Guo et al. 2024] and MLCopilot [Zhang et al. 2024] make clever use of past experiments as examples to enhance their LLM's ability to generate solutions. Data Interpreter [Hong et al. 2025], on the other hand, creates intricate graph-based workflows to decompose tasks into more manageable chunks. AIDE [Jiang et al. 2025] and SELA [Chi et al. 2024] use LLMs to guide the search for optimal configurations; SELA demonstrates the effectiveness of a well-guided MCTS, while AIDE shows how LLMs can be used to generate new solutions or improve and debug previous attempts.

In terms of assessment, although several benchmarks for evaluating agentic MLE have been recently published [Chan et al. 2025, Hu et al. 2024, Huang et al. 2024a], there is still no centralized metric for evaluating ML agents, with a significant number of studies reporting comparisons across disjointed collections of Kaggle competitions [Hong et al. 2025, Guo et al. 2024, Li et al. 2025, Grosnit et al. 2024, Chan et al. 2025]. Additionally, due to the ever expanding scope of LLMs' training data and performance, most benchmarks are likely contaminated or saturated within months of their publication [Jacovi et al. 2023].

More holistic approaches to automated data science and analysis —encompassing data collection, data cleaning, exploratory data analysis and effective data presentation— are less common, likely due to the complexity and subjectivity required to effectively evaluate them. DATA INTERPRETER [Hong et al. 2025] demonstrates exploratory analysis on its planning graphs, however its evaluation and reporting are limited. Data Formulator 2 [Wang et al. 2025] describes a LLM-based tool for data visualization, but one that is designed for user prompts rather than end-to-end analysis. Agent K v1.0 [Grosnit et al. 2024] is an end-to-end autoML agent reported as capable of gathering data, processing data and evaluating statistical models; its evaluations, however, have been restricted to interacting with the official Kaggle application programming interface (API). Still, most works focus on server-size models and extensive search methods, making the process less accessible and more difficult to follow.

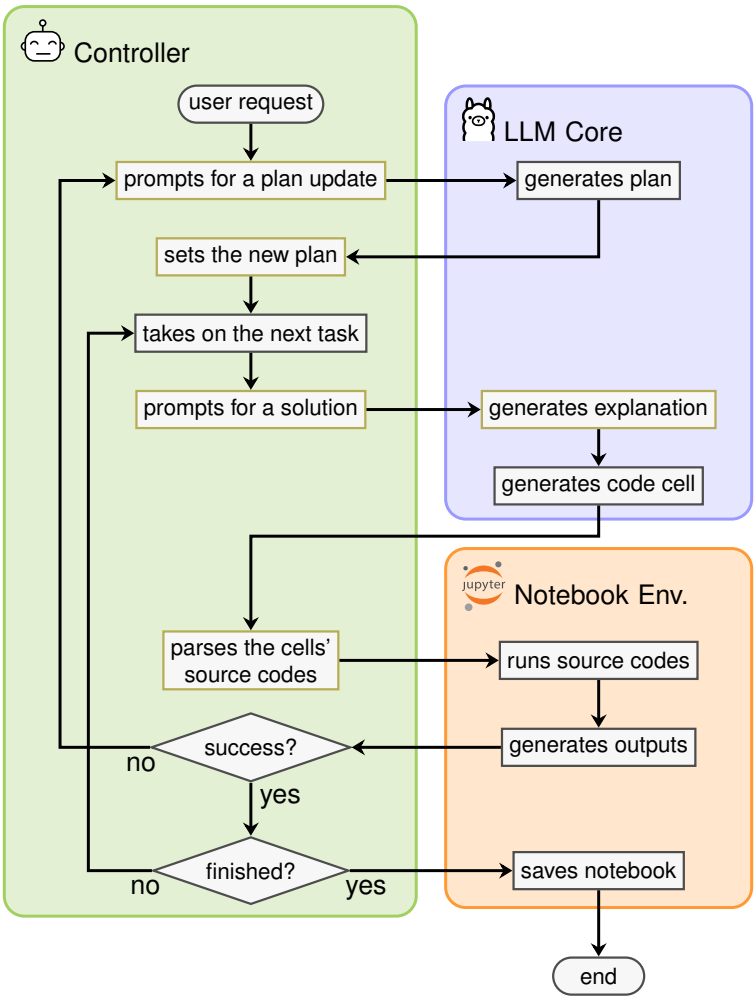
3. Methodology

The DATAEXPLAINER framework is based on a previous work by [Hong et al. 2025] named DATA INTERPRETER, similarly using a Jupyter notebook for execution, outputting a `ipynb` file and employing topological orderings for its action plan progression. However, our framework expands and adapts a number of aspects of the original. In subsection 3.1, we provide more information on how DATAEXPLAINER's processes differ from DATA INTERPRETER. Next, subsection 3.2 outlines the process we followed for our evaluations.

3.1. DATAEXPLAINER

DATAEXPLAINER's pipeline differs from the original DATA INTERPRETER mainly by generating explanatory steps in addition to python code; the result is a solution notebook that is closer in style to a didactic, step-by-step analysis. Figure 1 depicts DATAEXPLAINER's overall process; steps significantly different from the original framework are

Figure 1. The overall process of DataExplainer



Source: Authors (2026)

highlighted with yellow borders. When addressing a user's request (such as solving machine learning problems or conducting exploratory analyses), the controller prompts the LLM core (the agent's reasoning component) for a plan. The LLM provides a plan, and with it, the controller continuously undertakes each task, querying the LLM for the next notebook cells necessary to complete the current task. The LLM core first provides an explanation for its process, summarizing its previous executions and providing an outline of its proposed solution. Next, it generates a solution to the task using Python code. With the LLM output, the controller parses the received information and executes it in a Jupyter notebook environment—the explanation as a markdown cell and the solution as a code cell. If the execution is unsuccessful, the controller revises its plan and tries again. If it is successful, the controller checks for unfinished tasks and continues to work on them. When all tasks are finished, the notebook file is saved, and the process is complete.

3.1.1. The Action Plan

The action plan is a guide to the LLM core, narrowing the scope of its generations and setting an path towards a holistic solution. While the original framework uses a complex action plan, including the code solutions and outputs, DATAEXPLAINER uses a minimal approach to planning focusing exclusively on the sub-tasks' descriptions, types and order. To accommodate the elements removed from the action plan, we created a notebook state, which we will expand upon in subsubsection 3.1.3.

The user's request is divided into n sub-tasks, which contain a description, a type (data loading, EDA, model training and so forth), and a dependency list, which is used to define the plan's overall sequence through a topological sort. In case the agent struggles to complete a task, it can alter its plan mid-processing, with the addition of new tasks or removal/alteration of unfinished tasks. This allows the agent to adapt to previously unseen developments during its execution without losing track of its main trajectory.

3.1.2. Interaction with the LLM Core

When compared to the original framework, the interaction to the LLM core was largely overhauled, with a stronger focus on the Jupyter Notebook format and on the text prompt, instead of the action plan.

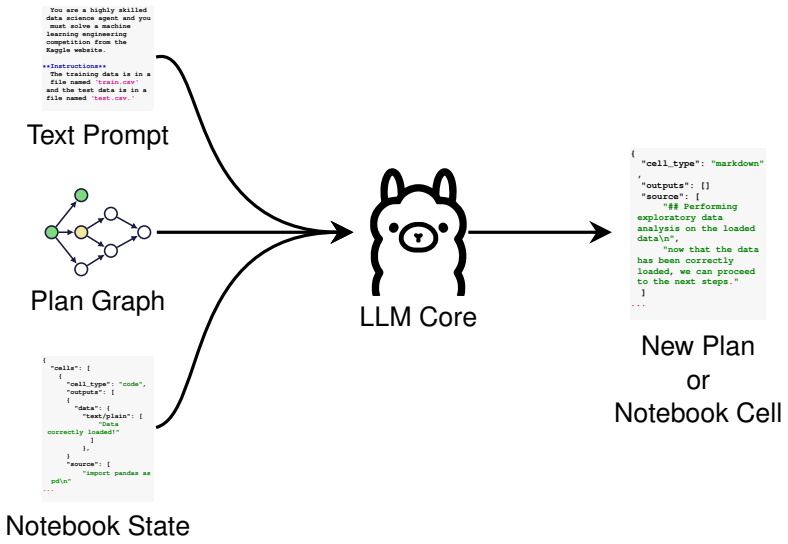
There are, in general, two types of interactions between the controller and the LLM core: the creation/update of its action plan and the creation of a new notebook cell. Figure 2 is an example of how the LLM core is prompted. For every LLM call, the controller generates a prompt composed of three different types of information: the text prompts, which include the system prompt, the user request and a task-specific guidance;[3] the plan graph, which is the graph representation of the current plan (in a JSON format), with completed, current and future tasks properly flagged; and the notebook state, which is the current state of the jupyter notebook being developed, in a simplified—i.e., more token-efficient— `ipynb` format. The LLM core, then, generates a JSON object containing either the generated plan or notebook cell. Both the notebook state and the generated cell use the same `ipynb` structure. We deliberately chose this approach in order to create a context similar to the DS solutions available on the web, which mostly uses `ipynb` notebook files; we hypothesize this closer context should improve the overall metrics.

3.1.3. Interaction with the Jupyter Environment

The controller interacts with a Jupyter notebook client to run the cells and to keep an active python kernel. Every new cell created by the LLM core and parsed by the controller is executed in the environment, which, in turn, generates an outputs list. Outputs can be text streams, images or error messages; each type is adjusted in specific ways before being re-fed into the agent's memory. For the text streams, the agent ignores repetitive warnings—which can be taxing on the limited context— removes text decorations (such

[3]Task guidances are a markdown list of important aspects to consider for each task type; for example, a "model evaluation" task might have the guidance "remember to always evaluate models in a holdout set, never in the train set"

Figure 2. Overview of LLM calls



Source: authors (2026)

as color tags) and truncate string lengths to a predefined limit. For the images, the agent will either encode into the correct format or ignore altogether, depending on the agent's visual processing setup. Finally, error messages have their traceback limited to the first and last two iterations, as to reduce the context length and focus on the most relevant information.

3.2. The Evaluations

The evaluations were conducted on a collection of seven Kaggle machine learning competitions, all published after the evaluated models' knowledge cutoff dates. The competitions are also varied in objectives (three regression tasks, three binary classification tasks and one multiclass classification task), and in data treatment, including noisy data and missing values. Table 1 is a summary of the collected competitions.

Evaluating automated machine learning engineering through open competitions is a well-established and directly measurable process. All competitions available on Kaggle feature a clear metric, submission process, and scoreboard, which can be easily drawn upon to generate a comprehensive assessment. With that in mind, the evaluation process was centered around agents being prompted to solve a specific competition, generating a CSV submission file and its predictions being scored using the Kaggle API.

As LLM often operate on a non-deterministic principle, 4 different samples were collected from each agent-competition pair. For each sample, the agents had a 60 minutes time limit for completing the Jupyter notebook execution. As models may often underestimate processing times, they were granted a single additional try after the first timeout. The two main metrics involved in our evaluations, success rate, corrected score, are expanded upon in the next paragraphs.

The success rate metric quantifies an agent's compliance with the given constraints and submission format when generating a solution. To evaluate the success rate, we

Table 1. Information about the gathered competitions

Codename	Start Date	Objective	Metric	Data Quality	Rows in Train Set
s4e6	Jun. 1, 2024	Multi class classification	Accuracy	Clean	76,518
s4e8	Aug. 1, 2024	Binary classification	MCC[1]	Artifacts and Missing Data	3,116,945
s4e9	Sep. 1, 2024	Regression	RMSE[2]	Missing Data	188,533
s4e10	Oct. 1, 2024	Binary classification	AUC[3]	Clean	58,645
s4e11	Nov. 1, 2024	Binary classification	Accurracy	Artifacts and Missing Data	140,700
s4e12	Dec. 1, 2024	Regression	RMSLE[4]	Missing Data	1,200,000
s5e2	Feb. 1, 2025	Regression	RMSE	Missing Data	300,000

[1] Matthews correlation coefficient; [2] Root mean squared error; [3] Area under the curve; [4] Root mean squared logarithmic error.

Source: Authors (2026)

define a function $success_c$ that considers a solution, comprised of a Jupyter notebook and submission file, and outputs either a 1 or a 0, depending if any of the following violations/failures occur:

- The agent attempted to directly access the web in order to find a solution;
- The agent attempted to directly access system files other than the ones explicitly provided;
- The agent attempted to manually fill the submission file;
- The agent failed to generate a submission file within the established time limit;
- The submission file was unable to be submitted to the API;
- The submission file was successfully submitted, but the system failed to assigned it a competition score;

These failure cases were identified using a mix of heuristics and human decision. Firstly, as a heuristic, all notebooks were scanned for evidences of offending code[4] and matches were flagged as possible failures. Next, a human analyzed each of the selected cases and set a final verdict. Formally, the success rate (SR) of an agent a when given a competition c is:

$$\text{SR}_a(c) = \frac{1}{n}\sum_{i=1}^{n} success_c(S_{a,i}(c)); \tag{1}$$

With n being the number of samples collected and $S_{a,i}(c)$ being the ith sampled solution generated by agent a when given a competition c.

[4] e.g, the use of the "requests" package for accessing the web or the use of file paths other than "train.csv" and "test.csv".

Differently from the success rate, assessing agents by their competition scores, although straightforward when considering competitions in isolation, becomes significantly more complex when aggregating various ML problems toward a single comprehensive metric. Classification score metrics are typically in the $[0, 1]$ range and are directly correlated with performance, whereas regression metrics, in contrast, usually fall in the $[0, \infty)$ range and are inversely correlated with performance. This implies that any assemblage of these metrics involving direct sums would severely misrepresent the performance. Thus, to effectively measure the agent's scores across various competitions, we adjusted the scores based on the placement of each submission on Kaggle's official competition public leaderboards. This form of normalization corrects for both differences in metrics and in variation, as it is anchored to direct comparisons to previous available, mostly human, solutions. We named this score PCS, for placement-corrected score. Ranging from 0 to 1, it represent how close to the top of the leaderboard a score is, with larger values being closer to the top. Formally, the PCS of an agent a when given a competition c is defined as:

$$\text{PCS}_a(c) = \begin{cases} \frac{1}{n}\sum_{i=1}^{n} P(X \leq score_c(S_{a,i}(c)) \mid X \sim \text{L}_c), & \text{for classification} \\ \frac{1}{n}\sum_{i=1}^{n} P(score_c(S_{a,i}(c)) \leq X \mid X \sim \text{L}_c), & \text{for regression} \end{cases} \quad (2)$$

Where n is the number of samples collected, $score_c(S_{a,i}(c))$ is the raw Kaggle score for solution $S_{a,i}(c)$ to competition c, P is a probability function, L_c is the public leaderboard for c, and X is a random variable distributed according to L_c.

We assessed four different LLM cores with distinct characteristics: Gemini 2.0 Flash [DeepMind 2025], a proprietary model; GPT-OSS 20B [OpenAI Team 2025], an open-source thinking model; Qwen3 coder 30B [Qwen3 Team 2025], an open-source code-tuned model; and Gemma3 27B QAT[5] [Gemma Team 2025a], an open-source generalist model. The Gemini model was provided by Google's cloud API service and its context window was limited to 128K tokens. The other models were run locally using the Ollama server version 14[6]. To reduce the overall memory footprint, the locally run models were quantized to 4 bits and had a context window of 32K tokens, with GPT-OSS being provided an extra 32K as a "thinking budget". All evaluations were conducted using a single Nvidia RTX 3090 consumer-grade GPU.

4. Results

Table 2 shows the success rate for all four models and seven competitions. All failures had only two causes: not generating a submission file within the time limit and incorrect file formats. There were no instances of agents accessing the web or files outside the delimited workspace for solutions. Some models, after failed attempts to train an inference model, filled the submissions with the target's most common class or average value, for classification and regression tasks respectively. We considered those submissions valid, as the resulting model, although crude, can also be considered an algorithm for inference.

To evaluate the effectiveness of our proposed framework, we compare DATAEXPLAINER to the original DATAINTERPRETER under the same conditions. The code for

[5]A Gemma3 version using quantization aware training.

[6]https://ollama.com/

Table 2. Success Rate per Model

Model	Success Rate (SR) %							
	s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	100	50	50	100	100	100	75	82.14
GPT-OSS	100	100	100	100	100	100	100	100
Qwen3-coder	50	75	100	100	100	100	100	89.29
Gemma3	75	0	50	100	0	100	75	57.14
Mean	**81.25**	**56.25**	**75**	**100**	**75**	**100**	**87.5**	**82.14**

Source: Authors (2026)

Table 3. Mean Scores between Frameworks per Model

Model	Framework	Mean Placement-Corrected Score (PCS) %							
		s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	Interpreter	61.49	11.38	32.17	13.49	10.05	9.79	22.69	23.01
	Explainer	37.42	34.06	6.88	48.76	38.47	25.45	17.62	29.81
GPT-OSS	Interpreter	42.18	2.35	32.33	36.96	42.98	3.6	38.67	28.44
	Explainer	48.85	44.87	25.64	66.79	35.25	16.19	36.38	39.14
Qwen3-coder	Interpreter	0.78	2.35	15.81	0.0	0.0	0.0	3.56	3.21
	Explainer	33.64	41.32	26.53	37.68	4.21	14.37	57.56	30.76
Gemma3	Interpreter	0.78	2.35	9.35	0.0	44.1	0.0	0.0	8.08
	Explainer	25.29	2.35	3.72	56.43	0.0	11.37	2.38	14.51
Mean	Interpreter	26.31	4.61	22.41	12.61	24.28	3.35	16.23	15.69
	Explainer	36.3	30.65	15.69	52.42	19.48	16.85	28.48	28.55

Source: Authors (2026)

DATAINTERPRETER was exactly as provided in the authors' github page at the original paper's publication [Hong et al. 2025], with the sole exception being the changes required to give the agents an additional timeout. Table 3 shows a comparison of the mean corrected scores across all four samples between the frameworks for each LLM and competition. DATAEXPLAINER outperformed the original for every LLM tested, with a major improvement for Qwen3-coder and Gemma3. Notably, the Qwen3-coder model surpassed Gemini's average when paired with DATAEXPLAINER, showing how effective the framework is at extracting value out of previously low-performing models. Over all LLMs, the adaptions made represent an average absolute improvement of almost 13 percentage points. A Wilcoxon paire test between frameworks across all models yields a significant ($\alpha = 0.05$) *p-value* of 0.01.

To effectively compare the DATAEXPLAINER framework against the general population, Table 4 displays the corrected scores considering only the best score across sam-

Table 4. Best Placement Corrected Scores per Model

Model	Best Placement Corrected Scores (PCS) %							
	s4e6	**s4e8**	**s4e9**	**s4e10**	**s4e11**	**s4e12**	**s5e2**	**Mean**
Gemini 2.0 Flash	52.88	68.28	13.82	73.13	46.48	48.2	47.95	50.11
GPT-OSS	66.41	74.34	39.83	75.28	43.58	21.66	64.39	55.07
Qwen3-coder	71.65	68.28	42.67	61.67	4.21	28.39	63.98	48.69
Gemma3	40.21	2.35	7.43	58.54	0.0	14.17	3.06	17.97
Mean	**57.79**	**53.31**	**25.94**	**67.15**	**23.57**	**28.1**	**44.85**	**42.96**

Source: Authors (2026)

ples for each competition-LLM pair.[7] With this adjustment, we can see that GPT-OSS surpassed the average user in most competitions. Gemini 2.0 achieved a mean score above 50%; however, it underperformed by 5 percentage points when compared to the open source GPT-OSS model. Qwen3 placed less than 2 points below the Kaggle user average; although performing above the mean on four out of the seven competitions, its scores on the other three—specially on s4e11— hindered its overall evaluation. Gemma3, even while performing worse than all other agents on average, still achieved a score above 50% on competition s4e10.

Finally, to analyze if data contamination is a factor determining the scores, we attempted to draw a correlation between the data age and each agent's performance. Figure 3 shows a linear correlation for each of the models, evaluating the relation between the amount of months since the competition's publication and each agent's scores. Although the relation is negative for three out of the four models, the very low R^2 values indicate data recency is not a significant determinant of performance.

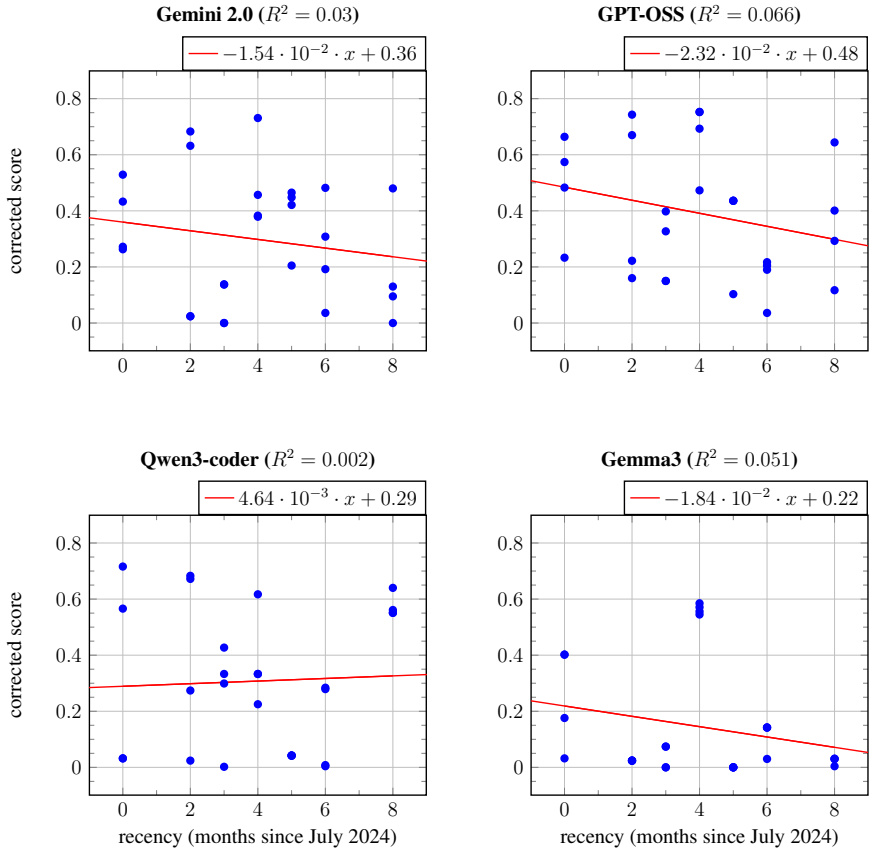
5. Discussion

The agents' success rates bring insights to these agents' most significant shortcomings. Firstly, agents seem well capable of following negative instructions—things it should *not* do— as none have tried to find a solution on the web or in files outside its workspace; the same is true for manually filling submission files. On the other hand, the agents struggled with positive instructions, such as completing tasks under a set time limit and following a submission format, which caused every identified failure.

The competition scores are less consistent than the success rates. Although mean competitions scores give insights into how difficult a competition is on average, isolated scores varied massively among models. For instance, while Qwen3 achieved an average score of 57.56% for its solutions to competition s5e2, the best out of all models, at competition s4e11 its score only reach a forth of the average. On the whole, the data indicates that the LLM agents are extremely non-deterministic and sensitive to the initial inputs when solving DS problems, even under very similar settings. Reducing this variability might necessitate a significant increase to the number of samples collected per model-competition pair.

[7]By default, Kaggle leaderboards exclusively display the best score achieved by each user in a given competition; therefore, utilizing the best score for each model provides the most equitable comparison.

Figure 3. Correlation between competition recency and corrected scores



Source: Authors (2026)

Considering the evaluated LLMs, GPT-OSS was by far the most capable. Not only did the model outperform all the others, but it did so while being the leanest and possibly less than half the size of Gemini 2.0 Flash.[8] Qwen3-coder also had a impressive performance for its size, achieving above average scores on four out of seven competitions, with competition s4e11 being its only major shortcoming in performance. Gemini 2.0 Flash, although displaying great scores, still failed to significantly surpass two of its quantized open-source counterparts; this indicates that, when it comes to agentic DS, larger parameter counts and context windows are secondary to effective CoT and context-aligned fine-tuning. Gemma3 performed the worst overall; a possible explanation is its generalist training and notable lack of a system/user prompt distinction [Gemma Team 2025b].

For the gathered data, the DATAEXPLAINER framework has outperformed the original DATA INTERPRETER in the vast majority of instances, indicating the changes made represent a net gain in performance. Specifically, the Qwen3 agent showed an expressive 27% absolute increase in average score. For the other models, the improvement remained noteworthy but less consistent, with DATA INTERPRETER surpassing DATAEXPLAINER in some competitions. Gemma's average score went from 8% to 14.5%, an almost 80% relative increase. As the scores from Gemini and GPT-OSS were already

[8]The parameter count of Gemini 2.0 Flash is still undisclosed, but assumed to be at least 40 billion.

above 20% for DATA INTERPRETER, the increase was subtler, with a 30% and 37.6% relative improvement, respectively. A possible explanation for the increase in scores is that the markdown cells act as a "thinking step" before a solution code is generated, recapitulating previous executions and preparing for the next. Overall, the data suggests DATAEXPLAINER is an improvement not only in its inclusion of explaining steps but also accuracy, especially for open-source and previously low-performing models. Examples of the step-by-step solutions generated by DATAEXPLAINER are presented in this paper's GitHub repository.

6. Conclusion

In this work, we introduced DATAEXPLAINER, a new framework for agentic data science with a larger focus on producing comprehensible outputs and dekstop-sized open-source models, taking the work by [Hong et al. 2025] as a base. The framework was then evaluated on seven varied, contamination-free MLE problems. The assessments show that our proposed framework not only significantly improved the scores of all agents when compared to the original, it also resulted in two out of the three open source models tested surpassing the average Kaggle user on most of the competitions evaluated. We believe these results show that more auditable agentic DS is possible, even with open-source desktop-sized models, when paired with the effective techniques and prompting. This represents an important step in making agentic solutions more sustainable and accessible, using open technologies and reduced resources.

As this work is mostly exploratory in nature, we acknowledge the limitation of the variety of the competitions used (which did not include image, video or audio processing) and in the number of samples collected; thus, future works with a larger scope and sample size would be of great value. Additionally, we hope to further explore the impact of prompting formats in performance, such as using the popular Token-Oriented Object Notation (TOON) format[9] for more token-efficient communication with the LLM core.

Acknowledgements

This work was partially supported by the Coordination for the Improvement of Higher Education Personnel (CAPES), Finance Code 001, and the National Council for Scientific and Technological Development (CNPq).

Generative AI Disclosure

This work was entirely written by humans and the authors take full responsibility for the information presented.

[9]https://toonformat.dev/

References

Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Madry, A., and Weng, L. (2025). MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net.

Chandel, S., Clement, C. B., Serrato, G., and Sundaresan, N. (2022). Training and evaluating a jupyter notebook data science assistant. *CoRR*, abs/2201.12901.

Chi, Y., Lin, Y., Hong, S., Pan, D., Fei, Y., Mei, G., Liu, B., Pang, T., Kwok, J., Zhang, C., Liu, B., and Wu, C. (2024). Sela: Tree-search enhanced llm agents for automated machine learning. *arXiv preprint arXiv:2410.17238*.

DeepMind (2025). Gemini — deepmind.google. `https://deepmind.google/technologies/gemini/`. [Accessed 19-02-2025].

Gemma Team (2025a). Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Gemma Team (2025b). Gemma formatting and system instructions | Google AI for Developers — ai.google.dev. `https://ai.google.dev/gemma/docs/core/prompt-structure`. [Accessed 13-02-2026].

Grosnit, A., Maraval, A., Doran, J., Paolo, G., Thomas, A., Beevi, R. S. H. N., Gonzalez, J., Khandelwal, K., Iacobacci, I., Benechehab, A., Cherkaoui, H., El-Hili, Y. A., Shao, K., Hao, J., Yao, J., Kegl, B., Bou-Ammar, H., and Wang, J. (2024). Large language models orchestrating structured reasoning achieve kaggle grandmaster level. *arXiv preprint arXiv:2411.03562*.

Guo, S., Deng, C., Wen, Y., Chen, H., Chang, Y., and Wang, J. (2024). DS-agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 16813–16848. PMLR.

Hong, S., Lin, Y., Liu, B., Liu, B., Wu, B., Zhang, C., Li, D., Chen, J., Zhang, J., Wang, J., Zhang, L., Zhang, L., Yang, M., Zhuge, M., Guo, T., Zhou, T., Tao, W., Tang, R., Lu, X., Zheng, X., Liang, X., Fei, Y., Cheng, Y., Ni, Y., Gou, Z., Xu, Z., Luo, Y., and Wu, C. (2025). Data interpreter: An LLM agent for data science. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T., editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, Vienna, Austria. Association for Computational Linguistics.

Hu, X., Zhao, Z., Wei, S., Chai, Z., Ma, Q., Wang, G., Wang, X., Su, J., Xu, J., Zhu, M., Cheng, Y., Yuan, J., Li, J., Kuang, K., Yang, Y., Yang, H., and Wu, F. (2024). Infiagent-dabench: Evaluating agents on data analysis tasks. In *International Conference on Machine Learning (ICML)*. JMLR.org.

Huang, Q., Vora, J., Liang, P., and Leskovec, J. (2024a). MLAgentBench: Evaluating language agents on machine learning experimentation. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F., editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 20271–20309. PMLR.

Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., and Chen, E. (2024b). Understanding the planning of llm agents: A survey.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Dang, K., et al. (2024). Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Springer Cham, 1st edition.

Jacovi, A., Caciularu, A., Goldman, O., and Goldberg, Y. (2023). Stop uploading test data in plain text: Practical strategies for mitigating data contamination by evaluation benchmarks. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. (2025). Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*.

Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., Yih, W.-t., Fried, D., Wang, S., and Yu, T. (2023). Ds-1000: a natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Li, Z., Zang, Q., Ma, D., Guo, J., Zheng, T., minghao liu, Niu, X., Yue, X., Wang, Y., Yang, J., Liu, J., Zhong, W., Zhou, W., Huang, W., and Zhang, G. (2025). Autokaggle: A multi-agent framework for autonomous data science competitions.

OpenAI Team (2025). gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Provost, F. J. and Fawcett, T. (2013). Data science and its relationship to big data and data-driven decision making. *Big data*, 1 1:51–9.

Qwen3 Team (2025). Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Russell, S. and Norvig, P. (2022). *Artificial Intelligence: A Modern Approach*. Pearson, USA, 4rd edition. Global Edition.

Taylor, P. (2024). Data growth worldwide 2010-2028 | statista — statista.com. `https://www.statista.com/statistics/871513/worldwide-data-created/`. [Accessed 24-02-2025].

Wang, C., Lee, B., Drucker, S., Marshall, D., and Gao, J. (2025). Data formulator 2: Iterative creation of data visualizations, with ai transforming data along the way. *arXiv preprint arXiv:2408.16119*.

Zhang, L., Zhang, Y., Ren, K., Li, D., and Yang, Y. (2024). MLCopilot: Unleashing the power of large language models in solving machine learning tasks. In Graham, Y. and Purver, M., editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2931–2959, St. Julian's, Malta. Association for Computational Linguistics.

Zhu, Q., Guo, D., Shao, Z., Yang, D., Wang, P., Xu, R., Wu, Y., Li, Y., Gao, H., Ma, S., et al. (2024). Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.