

# DataSciBench: An LLM Agent Benchmark for Data Science

Dan Zhang<sup>1,2,3,†</sup>, Sining Zhoubian<sup>1,2,†</sup>, Min Cai<sup>2,†</sup>, Fengzu Li<sup>1</sup>, Lekang Yang<sup>1</sup>  
Wei Wang<sup>1</sup>, Tianjiao Dong<sup>4,†</sup>, Ziniu Hu<sup>5</sup>, Jie Tang<sup>1,‡</sup>, Yisong Yue<sup>5,‡</sup>

<sup>1</sup>Tsinghua University <sup>2</sup>Z.AI <sup>3</sup>National University of Singapore

<sup>4</sup>University of California, Berkeley <sup>5</sup>California Institute of Technology

<https://datascibench.github.io/>

## Abstract

This paper presents DataSciBench, a comprehensive benchmark for evaluating Large Language Models (LLMs) in data science. Unlike existing benchmarks limited to single task, simple evaluation metrics, and readily available ground truth (GT), *DataSciBench* is built on curated, natural, and challenging prompts with complex evaluation criteria and uncertain GT. To bridge the gap, we develop a semi-automated GT generation pipeline, integrating LLM-based self-consistency and human verification to ensure accuracy, predefined task types, and aggregate functions (metrics). Furthermore, we introduce an innovative Intention-Function-Code (IFC) framework, assessing code execution outcomes through metrics and programmatic rules. Evaluating 26 models (8 API-based, 8 open-source general, 9 code generation, and 1 agentic models), our approach offers rigorous insights into LLM strengths and weaknesses. Experimental results show API-based models outperform open-source counterparts across all metrics, with DeepAnalyze-8B leading among open-sourced models. We release all code and data at <https://github.com/THUDM/DataSciBench/>.

## 1 Introduction

Large language models (LLMs) (Achiam et al., 2023; Team et al., 2023; GLM et al., 2024; Zhang et al., 2025a) are increasingly empowering data science and scientific domains, with applications ranging from data analysis (Hong et al., 2024; Huang et al., 2024; Jing et al., 2024) to protein generation (Jumper et al., 2021; Chen et al., 2025a) and scientific discovery (Lu et al., 2024; Chen et al., 2025b) to scientific reasoning (Zhang et al., 2024a,b). For data science tasks, LLMs can (semi-)autonomously execute workflows via

<sup>†</sup> work done while these authors interned at Z.AI

<sup>‡</sup> Corresponding author.

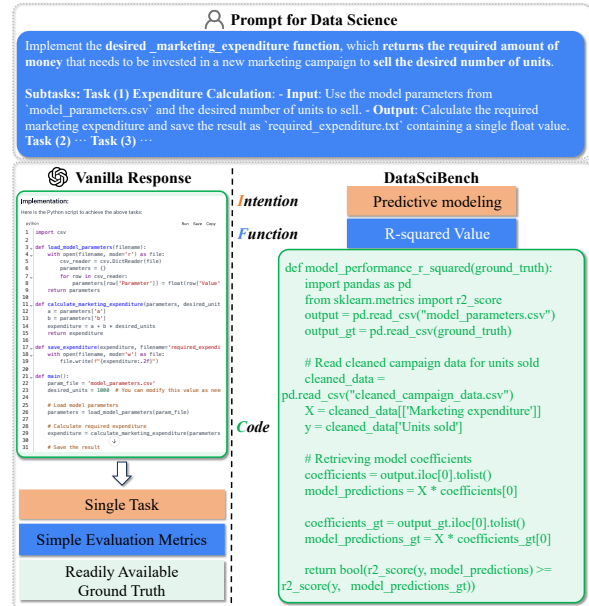


Figure 1: Comparison of vanilla and DataSciBench response for a complex prompt. The vanilla response only outputs code without intent-aligned evaluation, while DataSciBench leverages the IFC framework to identify intents, define task-specific verifiable evaluation functions, and generate programmatic code.

code interpreters, but existing benchmarks primarily focus on single task with easily available ground truth (GT) and simple metrics (e.g., Accuracy) (Huang et al., 2023). However, real-world data analysis requires reasoning over more complex, multi-step scenarios (e.g., calculating expenditure and evaluating visualization quality, cf. Figure 1), where GT generation, metric definition, and alignment with natural language intentions remain open challenges.

Existing benchmarks (e.g., MLAGentBench, Text2Analysis, cf. Table 1) fall short in addressing these gaps and focus on evaluating narrower tasks with easy-to-obtain ground truth and straightforward evaluation metrics (e.g., Accuracy). As LLM evaluation advances toward real-world complexity (cf. Figure 1), *how to unify intent understanding,*

Table 1: Comparison with related work across key dimensions: prompt sources, task characteristics, evaluation metric design, task planning, and average solution lines. LC = LeetCode; IFC = Intention-Function-Code.

Benchmark	Prompt Source	Task Modality	Evaluation Metrics	Planning	# Avg. Solution Line
DS-1000 (Lai et al., 2023)	StackOverflow	Text	Test Cases + Surface-Form Constraints	✗	3.6
MLAgentBench (Huang et al., 2023)	Kaggle	Text	Acc. + Success Rate, Human	✓	2.3
LiveCodeBench (Jain et al., 2024)	LC/AtCoder/CodeForces	Text	Test Cases + Pass Rate	✓	-
NaturalCodeBench (Zhang et al., 2024c)	CodeGeeX	Text	Test Cases + Pass Rate	✗	28.4
BigCodeBench (Zhuo et al., 2024)	StackOverflow	Text	Test Cases + Pass Rate	✗	10.0
Text2Analysis (He et al., 2023)	Human&LLM	Tabular	Executable ratio, Acc., Regression scores	✗	-
InfiAgent-DABench (Hu et al., 2024)	LLM	Tabular	Acc.	✗	~20
DataSciBench (Ours)	CodeGeeX&BCB&Human	<b>Text, Tabular, Image</b>	<b>25 Aggregate Metrics and Programmatic Rules (IFC)</b>	✓	14.4

*task-specific verifiable evaluation function, and code validation is a question worth exploring.*

To answer this question and address the gap, we present `DataSciBench`, a comprehensive benchmark for evaluating LLMs’ data science capabilities and providing insights to help LLMs improve their data analysis and data visualization abilities. Our collected **222** natural and challenging prompts cover **6** key data science tasks (e.g., data preprocessing, statistics, visualization, mining, and interpretability) and require multi-dimensional results for discriminative evaluation. To address GT generation and metric evaluation for these complex scenarios, we propose a novel semi-automated framework called Intention-Function-Code (IFC). From a coarse-grained perspective, IFC aggregates task types, intent-matched verifiable evaluation functions, and code templates. From a fine-grained perspective, it defines programmatic rules to validate output consistency with both specific tasks and GT, ensuring a fair and consistent assessment.

We evaluate **26** models (8 API-based, 8 open-sourced general, 9 code generation, and 1 agentic) on **519** IFC-driven GTs. We observe that API-based models greatly outperform open-source counterparts, with GPT-4o leading across all metrics and DeepAnalyze-8B as the top open-source model. However, all models have significant room for improvement in *following fine-grained instructions, calling the appropriate tools, executing accurate plans, and exporting the required execution outputs*.

Overall, our key contributions are as follows:

- We introduce `DataSciBench`, a comprehensive benchmark designed to assess LLMs’ performance in data science (cf. Figure 2). We develop a semi-automated pipeline to generate verifiable ground truth and evaluate aggregated metrics on carefully crafted complex questions.

- We propose an innovative Intention-Function-Code (IFC)-driven evaluation framework based on predefined aggregated metrics and programmatic rules. We then assess **26** large language models from both coarse-grained and fine-grained perspectives (cf. Table 2).
- Various analyses of comparisons regarding difficulty and alignment between ours and human evaluation are performed in Figure 5 and Table 3. Furthermore, we provide research insights derived from experimental outcomes of the evaluated LLMs that point to interesting directions for future work.

## 2 Background on using LLMs for Data Science

This section discusses the key aspects that underlie our benchmarking approach.

**Ground Truth Generation.** Ground truth serves as the cornerstone for evaluating the performance of LLMs in data science tasks. For diverse and challenging data science prompts, we aim to propose a semi-automated pipeline that leverages a robust LLM to generate GTs and employs self-consistency and human validation strategies to ensure the accuracy and reliability of generated GTs.

**Evaluation Metric Definition.** Defining appropriate and meaningful evaluation metrics is essential for effectively comparing and analyzing the effectiveness of different LLMs in data science tasks. Our study meticulously defines evaluation metrics tailored to the specific tasks and challenges the collected prompts pose. These metrics are designed to capture the diverse nuances of data analysis and visualization tasks, enabling a comprehensive assessment of LLMs’ capabilities.

**Limitation of Previous Studies.** Prior research in benchmarking LLMs for data science has often been limited by focusing on *single task, simple*

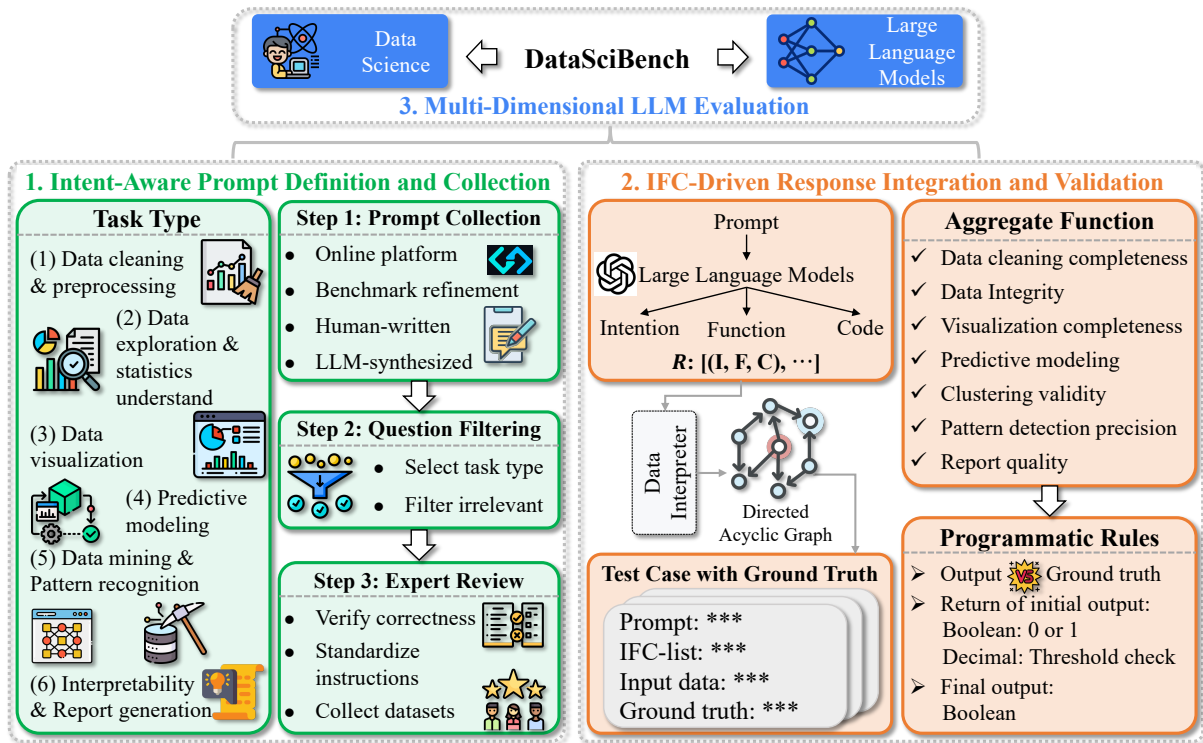


Figure 2: The DataSciBench encompasses three key components (cf. Section 3). (1) Intent-aware prompt definition and collection, which covers 6 task types, prompt collection, question filtering, and expert review. (2) IFC-driven response integration and validation, incorporating the IFC (25 aggregated functions and programmatic rules) and 519 test cases with ground truth. (3) Multi-dimensional LLM evaluation involving 26 LLMs.

evaluation metrics, and readily available ground truth. These shortcomings hinder the thorough evaluation of LLMs and may not fully capture their strengths and weaknesses. By addressing these limitations, our study seeks to provide a more comprehensive and nuanced assessment of LLMs in data science. Through the development of DataSciBench and the implementation of a rigorous evaluation framework, we aim to push the boundaries of benchmarking practices in the field of data science and LLM research.

### 3 The DataSciBench

DataSciBench comprises three core components (cf. Figure 2), centered on the IFC framework to enable intent-aligned, rigorous evaluation of LLMs data science capabilities.

- **Intent-Aware Prompt Definition and Collection** defines 6 task types and curates 222 natural, challenging prompts via multi-source collection, keyword filtering, and expert review.
- **IFC-Driven Response Integration and Validation** leverages a novel Intention-Function-

Code (IFC) framework to generate 519 test cases, linking prompt intents to task-specific evaluation functions and programmatic rules.

- **Multi-Dimensional LLM Evaluation** assesses 26 models (8 API-based, 8 open-sourced general, 9 open-sourced code generation, and 1 agentic) from coarse-grained (success rate, completion rate) and fine-grained (Vision-Language Model (VLM)-as-a-judge, 25 aggregate functions) perspectives.

#### 3.1 Intent-Aware Prompt Definition for Data Science

**Task Type.** We define six data science tasks (cf. Appendix A.2) that include Data cleaning and preprocessing, Data exploration and statistics understanding, Data visualization, Predictive modeling, Data mining and Pattern recognition, and Interpretability and Report generation.

**Task Integration.** To mimic real-world complexity, our carefully crafted prompts often combine multiple tasks in sequence, e.g., from clean data to explore statistics to visualize results.

### 3.2 Dataset Collection

**Step 1: Question Collection.** We collect questions from four sources: **(1) Real-world Platform.** We collect natural prompts from a real online code-generation platform, CodeGeeX (Zheng et al., 2023). **(2) Benchmark Refinement.** 167 high-quality data science prompts with test cases from BigCodeBench, rewritten to include explicit input data, intent-driven prompts, and expected outputs to align with IFC’s standardized evaluation requirements. **(3) Human-Written.** Expert-crafted prompts (referencing domain resources<sup>1</sup>) to enhance difficulty and cover edge-case intents. **(4) LLM-Synthesized.** LLM-generated prompts guided by human-written few-shot examples.

**Step 2: Question Filtering.** Based on carefully collected prompts, low-quality prompts are excluded via intent-focused principles: **(1) Retain prompts** containing domain keywords (e.g., “machine learning”, “deep learning”, “data preprocessing”, “data visualization”) that signal clear data science intents; **(2) Filter prompts** requiring code rewriting, error detection, or basic concept explanation that lack complex intent fulfillment.

**Step 3: Expert Review.** To ensure prompt quality and intent clarity, we review all prompts with experts in computer science and data analysis. The review process includes three stages: **(1) In stage 1**, experts verify prompt correctness, adjust intent suitability, and ensure responses to the questions are clear and structured in a way that facilitates assessment (e.g., handling missing values in a data frame). **(2) In stage 2**, experts standardize all verified prompts into a unified format that encompasses *input data or file*, *intent-driven prompt*, and *expected output file*. **(3) In stage 3**, experts ensure the availability of input prompt datasets by collecting public datasets or generating random datasets.

### 3.3 IFC-Driven Response Integration and Validation

**Intent-Consistent Ground Truth Generation and Verification.** Given the cost and scalability limitations of manual GT crafting, we adopt a semi-automated pipeline that prioritizes alignment with prompt core intents. Firstly, we sample multiple outputs from LLM (GPT-4o-mini), execute code snippets, and derive the final executable results. Then, we use two different validation meth-

<sup>1</sup><https://ds100.org/course-notes/eda/eda.html>

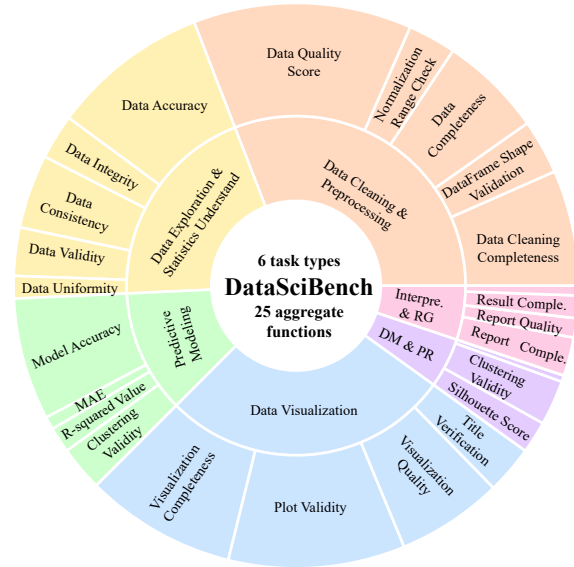


Figure 3: Statistics of 6 intent-mapped task types and 25 aggregate functions in the Intention-Function-Code (IFC) framework. DM & PR = Data Mining & Pattern Recognition. Interpre. & RG = Interpretability & Report Generation.

ods to validate whether the LLM-generated answer aligns with the meaning specified in the aggregate functions to ensure the reliability of the answer. **(1) For BCB-sourced prompts**, where reliable test cases are provided, we validate the generated answer by performing all test cases. Answers that pass all test cases are re-checked by humans and finally considered ground truth. **(2) As for other prompts**, we initially adopt a self-consistency strategy (Wang et al., 2022) to obtain outputs and then ensure their reliability and precision by having six authors of the paper verify the default assigned prompts and corresponding ground truth elements, including task type, verifiable evaluation function, programmatic code, and final outputs. In cases where uncertainties arise in the generated outputs, we cross-validate them among three authors.

#### Intent-Guided Verifiable Evaluation Selection.

We streamline evaluation by mapping prompts to intent-specific tasks, verifiable functions, and programmatic code. We first use GPT-4o-mini to identify valuable task types, select corresponding evaluation functions, and generate assessment codes to reduce cost while targeting core capabilities. Each evaluation unit is represented as a tuple (I, F, C) in generated **R** as follows:

$$\mathbf{R} = \{(I_i, F_i, C_i)\}_i^N, \quad (1)$$

where  $N$  is the number of intent-relevant tasks per prompt, and this value is different for each question. Then we generate a hierarchical directed acyclic graph (DAG) via data interpreter (DI) (Hong et al., 2024) to model task dependencies, in which each task type is defined as a node at one level in a DAG (cf. Figure 2). Based on the generated graphs, we take a powerful LLM as a backbone to run evaluation functions and derive the ground truth to avoid biases arising from commonly used black-box LLM-as-a-Judge.

**Function Aggregation.** To unify intent-aligned assessment and enhance the scalability, we aggregate all generated functions to the top- $K$  function category, select top- $K$  functions per task type, resulting in 25 core functions (cf. Figure 3). Generally,  $K$  is set as 5. For example, the function category for data cleaning and preprocessing includes Data Cleaning Completeness, DataFrame Shape Validation, Data Completeness, Normalization Range Check, and Data Quality Score.

**Programmatic Rules.** Regarding aggregate functions with corresponding codes, we standardize outputs for intent-aligned verification. Specifically, we unify initial outputs as boolean (0 or 1) or normalized decimal (0-1) values. For Boolean tasks, we obtain the final value by matching the ground truth with the generated output, depending on the task description of aggregate functions. For example, regarding Data Cleaning Completeness, which calculates the final number of rows/columns after preprocessing, the final output is 1 if the number is the same as the number of ground truths; otherwise, it is 0. For some specific tasks whose output type is decimal, we also set a corresponding threshold to convert the output to a boolean for simplicity. For example, the threshold is set to 0.5 if the aggregate function is the silhouette score for data mining and pattern recognition.

**Summary.** The IFC pipeline yields **222** intent-rich prompts, **6** task types, **519** test cases, and **25** aggregated functions, which help comprehensive evaluation of **26** models (8 API-based, 17 open-sourced, 1 agentic models).

## 4 Experiments

### 4.1 Settings

We evaluate **26** models (8 API-based, 8 open-sourced general, 9 code generation, 1 agentic) to assess their data science capabilities, leveraging

the IFC framework.

- **Eight API-based models** include o1-mini (Jaech et al., 2024), GPT-4o-mini (Hurst et al., 2024), GPT-4o-2024-05-13, GPT-4-Turbo (Achiam et al., 2023), Claude-3.5-Sonnet<sup>2</sup>, GLM-4.6, GLM-4.5-Flash (Zeng et al., 2025), and GLM-4-Flash (GLM et al., 2024).
- **Eight open-sourced general models** include Llama3.1-8B-Instruct, Llama3-8B-Instruct, Qwen2.5-7B-Instruct, Qwen2-1.5/7B-Instruct (Yang et al., 2024), Gemma2-9B-it (Team et al., 2024), GLM-4-9B-chat, and Yi-1.5-9B-chat-16k (Young et al., 2024).
- **Nine open-sourced code models** include Deepseek-Coder-1.3/6.7/33B-Instruct (Guo et al., 2024), CodeLlama-7/13/34B-Instruct (Roziere et al., 2023), Qwen-2.5-Coder-1.5/7B-Instruct (Hui et al., 2024), and StarCoder2-15B (Lozhkov et al., 2024).
- **One agentic model** is DeepAnalyze-8B (Zhang et al., 2025b).

### 4.2 Multi-Dimensional Evaluation Metrics

**Coarse-Grained Metrics.** We define two coarse-grained metrics (CR and SR) to measure LLMs.

• **Completion Rate (CR).** Adapted from Data Interpreter (Hong et al., 2024), CR quantifies the extent to which models complete intent-aligned IFC steps. We give it a completion score, with a minimum score of 0 and a maximum score of 2. The step completion scores were given as follows: missing (score of 0), fail (score of 0), success-non-compliant (score of 1), and success-compliant (score of 2), with CR calculated as:

$$\text{Completion Rate (CR)} = \frac{\sum_{t=1}^T s_t}{T \times s_{\max}}, \quad (2)$$

where the numerator was the sum of the completion scores for each step, and the denominator was the sum of the maximum possible scores for all steps ( $2 \times T$  and  $T$  is the number of IFCs).

• **Success Rate (SR).** Inspired by Codex (Chen et al., 2021), SR is defined as the rate of complete success across 10 runs per prompt. If all the IFC in  $\mathbf{R}$  have passed within a run of a single prompt, it will count as a success. Otherwise, it will count as a failure. Note that for BigCodeBench-sourced

<sup>2</sup><https://www.anthropic.com/news/claude-3-5-sonnet>

Table 2: Overall evaluation results for DataSciBench on all our curated prompts. **Dark gray** and **light gray** indicate the best result among all models and open-source models, respectively.

Models	Size	Coarse-grained Metrics		Fine-grained Metrics					Score	
		SR (%)	CR (%)	VLM	F1	F2	F3	F4		F5
<b>API-Based Models</b>										
o1-mini	N/A	29.77	45.26	2.87	44.63	19.27	36.01	30.94	23.81	38.78
GPT-4o	N/A	<b>66.31</b>	<b>68.44</b>	<b>3.91</b>	<b>75.93</b>	<b>56.14</b>	<b>69.33</b>	<b>71.35</b>	<b>57.67</b>	<b>64.51</b>
GPT-4o-mini	N/A	50.63	57.78	3.05	60.30	48.02	57.84	59.24	53.54	54.18
GPT-4-Turbo	N/A	51.93	58.87	3.09	62.30	41.62	57.75	60.25	50.75	54.65
Claude-3-5-Sonnet	N/A	47.48	58.11	2.14	49.07	36.94	55.84	52.87	46.04	52.29
GLM-4.6	N/A	66.22	44.46	0.88	59.42	44.35	59.40	66.02	44.62	45.94
GLM-4.5-Flash	N/A	58.11	36.79	1.13	40.97	39.94	48.68	43.62	37.26	37.40
GLM-4-Flash	N/A	30.32	34.04	1.33	36.53	29.42	32.57	27.64	14.44	30.74
<b>Open-Source General Models</b>										
Meta-Llama-3.1-8B-Instruct	8B	24.73	33.89	1.29	38.24	18.25	21.98	22.89	25.85	29.69
Meta-Llama-3-8B-Instruct	8B	2.88	3.92	0.52	4.18	1.26	2.70	2.67	1.47	3.33
Gemma-2-9B-it	9B	7.07	11.00	1.06	26.16	16.90	23.81	18.11	17.15	12.66
GLM-4-9B-Chat	9B	25.72	30.38	1.69	31.51	23.15	28.07	27.19	19.14	27.57
Qwen2.5-7B-Instruct	7B	43.83	50.74	1.43	51.18	36.41	47.25	45.24	34.77	45.99
Qwen2-7B-Instruct	7B	22.84	25.58	1.16	30.93	20.78	28.73	25.87	7.52	23.52
Qwen2-1.5B-Instruct	1.5B	3.96	5.46	0.40	4.54	1.98	3.26	5.76	4.71	4.78
Yi-1.5-9B-Chat-16K	9B	38.20	42.35	0.73	38.14	36.36	35.64	37.08	27.79	38.22
<b>Open-Source Code Generation Models</b>										
CodeLlama-34B-Instruct	34B	0.90	1.47	0.00	1.02	0.84	1.98	1.54	1.19	1.33
CodeLlama-13B-Instruct	13B	10.49	14.64	0.04	11.67	11.34	9.43	14.43	5.15	12.64
CodeLlama-7B-Instruct	7B	2.88	3.97	0.00	3.53	2.37	2.57	1.74	1.59	3.31
StarCoder2-15B	15B	2.07	2.61	0.07	2.57	1.81	1.59	3.43	1.19	2.33
DeepSeek-Coder-33B-instruct	33B	55.86	61.23	2.29	65.66	47.11	58.17	61.65	<b>48.60</b>	56.76
DeepSeek-Coder-6.7B-instruct	6.7B	37.03	41.62	1.93	43.49	34.57	46.36	46.49	18.09	38.45
DeepSeek-Coder-1.3B-instruct	1.3B	15.50	19.00	0.10	13.04	14.62	13.26	16.32	7.92	16.39
Qwen2.5-Coder-7B-Instruct	7B	45.18	53.11	1.48	51.58	43.21	43.87	42.50	35.23	47.67
Qwen2.5-Coder-1.5B-Instruct	1.5B	22.74	28.64	0.81	29.82	21.79	23.96	29.58	16.39	25.87
<b>Agentic Model</b>										
DeepAnalyze-8B	8B	<b>59.91</b>	<b>66.24</b>	<b>2.86</b>	<b>71.68</b>	<b>67.86</b>	<b>58.62</b>	<b>69.09</b>	<b>33.33</b>	<b>61.11</b>

prompts, we match IFC function outputs to the ground truth to determine whether a single run passes, since  $\mathbf{R}$  is derived based on demanded function outputs in this case. The formula for calculating SR is as:

$$\text{Success Rate (SR)} := \mathbb{E}_{\text{Prompts}} \left[ 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right], \quad (3)$$

where  $n = 10$  and  $k = 1$  in our case,  $c$  denotes the number of runs that have passed all IFCs in  $\mathbf{R}$ .

**Fine-grained Aggregate Metrics.** We define intent-aligned fine-grained aggregate metrics to evaluate LLMs performance (cf. Figure 3 and Table 6). It’s worth noting that data visualization is one of the more challenging tasks, which requires both output existence and intent-driven quality validation. Specifically, for tasks like data visualization, we focus on metrics such as visualization quality and use VLM-as-a-judge to assess the quality of images. Finally, we select six representative aggregated metrics as follows:

- VLM-as-a-judge assesses the overall score of two visual inputs against predefined criteria for progressively adding scores (cf. Appendix A.13) to write the total score with step-by-step reasoning explanation for its evaluation, with GPT-4o-mini serving as the VLM.
- Data Quality Score (F1) in Data cleaning and preprocessing aims to assess post-processing data cleanliness. It yields a boolean output of 1 if it matches the ground truth or 0 otherwise.
- Plot Validity (F2) in Data visualization refers to the accuracy of visual representations, such as checking whether the shape of an association matrix is consistent with the ground truth. If yes, the final value is 1, otherwise 0.
- Data Accuracy (F3) in Data Exploration and Statistics Understand focuses on understanding data quality and can be quantified using Mean Squared Error (MSE). The final value is derived by comparing it against the ground truth with a predefined threshold.

- Visualization Completeness (F4) in Data visualization evaluates the comprehensiveness of generated images (e.g., PNG, PDF) by checking their existence compared to the ground truth. A score of 1 is assigned if the files exist, and 0 otherwise. Note that we use VLM-as-a-judge and F2 to assess the quality of generated images.
- Model Accuracy (F5) in Predictive modeling is utilized to gauge the predictive performance of models, providing a Boolean accuracy value or a decimal ranging between 0 and 1.

**Final Score.** The final score prioritizes the most important intent-aligned CR while integrating fine-grained metrics and VLM evaluations based on empirical observations, with weights balancing the importance of various aspects and reflecting real-world data science priorities. We also supplement a thorough sensitivity analysis and further clarify the rationale for the weight setting (cf. Appendix A.3).

$$S = 0.05 \times \sum_{i=1}^5 F_i + 0.05 \times S_{VLM} + 0.05 \times SR + 0.65 \times CR. \quad (4)$$

Here, the final Score is computed by combining the weighted sum of five metrics  $F_i$  (where  $i$  ranges from 1 to 5), a value VLM score, a value SR, and a value CR. Each factor is multiplied by its respective weight (0.05 or 0.65).

## 5 Results and Analysis

### 5.1 Overall Performance

We present overall experiment results in Table 2 and Figure 4, with key findings:

**(1) Concerning average performance, API-based models outperform open-sourced models.** Among API-based models, GPT-4o achieves the highest total score of 64.51%, attaining a significant 9.86% advantage over GPT-4-Turbo (54.65%). GPT-4o also excels across all metrics, indicating its comprehensive intent understanding and execution. In comparison, the performance variance between API-based models is smaller than that of open-sourced models.

**(2) As for open-sourced models, general and code generation models exhibit minimal overall gaps.** However, DeepSeek-Coder-33B-Instruct achieves the highest score of 56.76%, outperforming various API-based models like o1-mini and

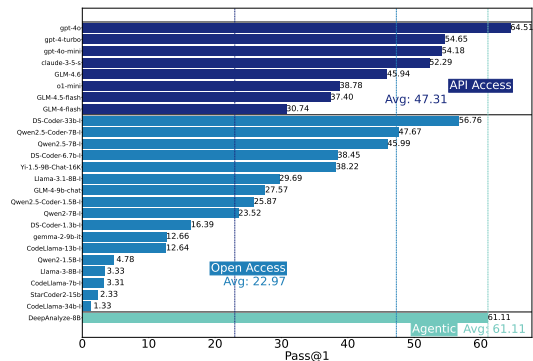


Figure 4: Overall score results of all tested LLMs.

Table 3: Alignment between VLM (GPT-4o-mini) and human evaluation scores.

Metric	Average Score
Dimension-level Pearson Correlation	0.85
Pair-wise Correlation	0.97
Cosine Similarity	0.99

GPT-4-Turbo. Other models like Qwen2.5-Coder-7B-Instruct (47.67%) and Qwen2.5-7B-Instruct (45.99%) also show fair to good capability. In contrast, a few models only pass very few tasks, achieving total scores even lower than 5.0%. Of these, CodeLlama-34B-Instruct unexpectedly achieves a score of 1.33%, even lagging behind its small-scale version, CodeLlama-7B-Instruct. We present an analysis of the anomaly in Section 5.6.

### 5.2 VLM and Human Evaluation Alignment

To validate the reliability of VLM-as-a-judge (GPT-4o-mini) for assessing visualization, we compare its score with human evaluations across five fine-grained dimensions (cf. Table 3 and Table 4). Table 3 shows high alignment across three metrics: Dimension-level Pearson Correlation of 0.85, Pairwise Correlation of 0.97, and Cosine Similarity of 0.99, confirming the VLM’s ability to reliably capture human judgments of intent-aligned visualization quality. Moreover, we use a more fine-grained analysis over five dimensions for this experiment (cf. Table 4). After obtaining the scores for each dimension of LLM-generated charts, we calculate the *Pearson correlation coefficient* for each sample and compute the mean correlation (cf. Appendix A.4).

### 5.3 Comparison with Existing Benchmarks

We perform correlation analysis to evaluate the alignment between DataSciBench and coding evaluations like BCB and LCB. To achieve corre-



age CR, particularly excelling in complex, data-driven tasks defined as hard. (3) There are noticeable performance disparities among general models and small-scale models in terms of average metrics, such as CR, with some achieving lower scores overall, suggesting that general models are less efficient or accurate in data science tasks.

## 5.6 Deeper Insights into LLMs' Ability

With curated metrics, we can obtain deeper research insights into LLMs' ability to plan and execute complex data science tasks. The experiment results also raise questions that are worth exploring since some results do not conform to conventional perceptions.

*(Observation 1) - LLMs demonstrate proficiency in reasoning tasks but may not consistently excel in complex data science tasks that necessitate precise adherence to detailed instructions, utilization of existing tools, and strategic planning.*

Although data science coding tasks often involve scheduling and step-by-step execution similar to reasoning scenarios, results show that even the LLMs proficient in reasoning tasks can still fail to complete complex data science tasks. For instance, the o1-mini model, which is commonly regarded as one of the best reasoning models, unexpectedly failed on many of DataSciBench's tasks. The model only achieves a 29.77% overall success rate, significantly lagging behind the company's previously introduced models like GPT-4o and GPT-4-Turbo. After examining the completions generated by o1-mini, we discovered that the failures are primarily caused by non-compliance with instructions, incorrect calls, and forgetfulness. While successfully splitting the task into multiple subtasks, the model often forgets to export the required execution outcomes or just outputs undesired data. In other cases, the model may falsely call a library function or method that sometimes does not even exist. The main kind of coding error is an execution error. An example is a hallucination about the column name of a CSV file. These facts remind us that real-life data science coding tasks often comprehensively challenge the model's ability to follow fine-grained instructions, utilize existing tools (libraries, APIs...), and do planning. To excel and align effectively with these tasks, a model must demonstrate competitiveness across all relevant aspects. We included four bad examples generated by o1-mini for further clarifi-

cation (see Appendix B.2).

*(Observation 2) - Larger-scale models may encounter challenges in following simple instructions due to the extensive data utilized during training, potentially leading to difficulties in generating outputs in formats different from those within the training data.* StarCoder2-15B performs worse than some smaller models, and CodeLlama-34B-Instruct even performs worse than 13B and 7B versions. The main reason is that the larger-scale version lacks some other ability like generating formatted text according to prompts. Perhaps a large amount of data in a certain format is being used to train a larger version that fails to follow the prompt to generate another format different from that. We present some examples in Appendix B.2. Indeed, the larger-scale model of CodeLlama also fails to outperform the smaller-scale version in LiveCodeBench.

## 6 Conclusion

This paper introduces DataSciBench, a novel framework tailored to assess the capabilities of LLMs in data science tasks. By meticulously curating challenging prompts and leveraging robust LLMs alongside a self-consistency strategy, we generate ground truth for all prompts. To efficiently evaluate LLMs' performance, we aggregate evaluation metrics and synthesize the IFC list programmatically. Subsequently, we assess 26 API-based, open-source and agentic models, offer valuable research and engineering insights, and present error analyses of the assessed LLMs.

## Limitations

In certain visualization tasks, our initial metrics and evaluation methods (e.g., VLM-as-a-judge) may lack precision. Further refinement of metrics is required to evaluate data visualization tasks effectively. One potential approach could involve employing VLMs to train critic models, enhancing the capability for fine-grained evaluations of visualizations.

## Acknowledgments

This work was partially supported by the National Key R&D Program of China 2025YFC3409902, Natural Science Foundation of China (NSFC) 62425601 and 62406163, and the New Cornerstone Science Foundation through the XPLOER PRIZE.

## References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Bo Chen, Xingyi Cheng, Pan Li, Yangli-ao Geng, Jing Gong, Shen Li, Zhilei Bei, Xu Tan, Boyan Wang, Xin Zeng, et al. 2025a. xtrimopglm: unified 100b-scale pre-trained transformer for deciphering the language of protein. *nature methods*, 22:1028–1039.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. 2025b. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *ICLR*.
- Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.
- Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, Zejian Yuan, and Dongmei Zhang. 2023. Text2analysis: A benchmark of table question answering with advanced data analysis and unclear queries. *arXiv preprint arXiv:2312.13671*.
- Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, et al. 2024. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.
- Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. In *Forty-first International Conference on Machine Learning*.
- Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. Benchmarking large language models as ai research agents. *arXiv preprint arXiv:2310.03302*.
- Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 2024. Da-code: Agent data science code generation benchmark for large language models. In *EMNLP*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fan-jia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Live-codebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.
- Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. Dsbench: How far are data science agents from becoming data science experts? *arXiv preprint arXiv:2409.07703*.
- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589.
- Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.
- Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and ZHAO-XIANG ZHANG. 2024. Sheetcopilot: Bringing software productivity to the next level through large language models. In *NeurIPS*.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. Agentbench: Evaluating llms as agents. In *ICLR*.
- Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi,

- Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.
- Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. Insightpilot: An llm-empowered automated data exploration system. In *EMNLP: System Demonstrations*, pages 346–352.
- Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.
- Pengyang Shao, Naixin Zhai, Lei Chen, Yonghui Yang, Fengbin Zhu, Xun Yang, and Meng Wang. 2026. Baldro: A distributionally robust optimization based framework for large language model unlearning. In *the Web Conference*, page 8874–8884.
- Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.
- Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.
- Wei Wang, Dan Zhang, Tao Feng, Boyan Wang, and Jie Tang. 2024. Battleagentbench: A benchmark for evaluating cooperation and competition capabilities of language models in multi-agent systems. *arXiv preprint arXiv:2408.15971*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Xiao Xia, Dan Zhang, Zibo Liao, Zhenyu Hou, Tianrui Sun, Jing Li, Ling Fu, and Yuxiao Dong. 2025. Scenegenagent: Precise industrial scene generation with coding agent. In *ACL*.
- Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2025. Androidlab: Training and systematic benchmarking of android autonomous agents. In *ACL*.
- An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.
- Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.
- Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, Yuxiao Dong, and Jie Tang. 2025a. Parameter-efficient fine-tuning for foundation models. *arXiv preprint arXiv:2501.13787*.
- Dan Zhang, Ziniu Hu, Sining Zhoubian, Zhengxiao Du, Kaiyu Yang, Zihan Wang, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Sciglm: Training scientific language models with self-reflective instruction annotation and tuning. In *NeurIPS D&B*.
- Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024b. Rest-mcts\*: Llm self-training via process reward guided tree search. In *NeurIPS*.
- Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025b. Deepanalyze: Agentic large language models for autonomous data science. *arXiv preprint arXiv:2510.16872*.
- Shudan Zhang, Hanlin Zhao, Xiao Liu, Qinkai Zheng, Zehan Qi, Xiaotao Gu, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024c. Naturalcodebench: Examining coding performance mismatch on humaneval and natural user prompts. In *Findings of ACL*.
- Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. 2024d. Data-copilot: Bridging billions of data and humans with autonomous workflow. In *ICLR Workshop on LLM Agents*.
- Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. In *KDD*.
- Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

## A Appendix of DataSciBench

### A.1 Broader Impacts

This work aims to promote the development of LLMs in performing data science tasks. This paper is to construct prompts in the field of data science and evaluate the ability of LLMs to solve these prompts without causing negative social impacts.

### A.2 Task Definition

We define six typical data science tasks as follows:

1) **Data cleaning and preprocessing.** This task detects and processes missing values, outliers, and duplicate data; and standardizes data formats, such as a uniform format for dates and times.

2) **Data exploration and statistics understanding.** This task calculates basic statistical indicators of data (mean, median, standard deviation, etc.), generates data distribution charts (histograms, box plots, etc.), calculates correlations between variables, and draws correlation matrices or maps.

3) **Data visualization.** The goal of this task is to visualize and analyze data and create interactive charts so users can freely explore the data.

4) **Predictive modeling.** The task aims to select the appropriate machine learning algorithm, such as linear regression, decision tree, random forest, etc.; carry out feature engineering, such as feature selection, feature transformation, feature combination, etc.; the data set is divided into the training set and test set, and the model is trained and evaluated; and select the corresponding evaluation indicators for different prediction problems, such as classification, regression or clustering.

5) **Data mining and Pattern recognition.** This task uses association rule mining, frequent item set mining, and other methods to find interesting patterns in the data; Text mining technology is used to extract keywords, topics, and other information from text data; and apply cluster analysis, classification algorithms, etc. to identify underlying patterns and structures. Pattern recognition tasks can conduct these functions: image recognition, text clustering, and time series detection.

6) **Interpretability and Report generation.** This task aims to provide explanations of model results, such as feature importance, model parameters, etc., and automatically generate reports and summaries that present the results of the analysis in a way that is easy to understand and share.

### A.3 Sensitivity Analysis of the Final Score Weights

Here, we provide a thorough sensitivity analysis of weight adjustments:

(1) **Rationale for the high weight of CR (65%):** Data science tasks are inherently multi-step and process-oriented, where step completion is a fundamental prerequisite. If a model fails to complete the core IFC (Intention-Function-Code) steps, the subsequent fine-grained metrics and final output evaluation become meaningless. This weight setting is not arbitrary but references benchmark works in top conferences/journals in the field. For example, Data Interpreter (Hong et al., 2024) set the weight of CR to 50%, and DA-Code (Huang et al., 2024) mainly adopted CR as the core evaluation metric.

(2) **Sensitivity analysis of weight choices:** To verify the impact of weight adjustments on the final rankings, we conducted a sensitivity analysis with four weighting schemes and calculated the Spearman correlation coefficients between the original scheme and the new schemes. The details are as follows:

- Original weight scheme (CR=65%, SR=5%, VLM=5%, five fine-grained metrics=25% in total);

- Equal weight scheme (all metrics share equal weights): Spearman's  $\rho = 0.9877$ , p-value = 0.0000 (Average vs Original weight scheme);

- SR-dominant scheme (SR=50%, CR=30%, VLM and five fine-grained metrics are 20% in total): Spearman's  $\rho = 0.9747$ , p-value = 0.0000 (SR Score vs Original weight scheme);

- Fine-grained metric-dominant scheme (SR=25%, CR=25%, VLM + F1-F5 = 50%): Spearman's  $\rho = 0.9904$ , p-value = 0.0000 (Fine Score vs Original weight scheme).

The results show that the Spearman correlation coefficients between all new weight schemes and the original scheme are  $\geq 0.97$ . Notably, GPT-4o remains the Top-1 model across all schemes, demonstrating that the evaluation results are highly robust to weight adjustments and the core conclusions of our work are not affected.

### A.4 Alignment with Human Evaluation

Given two sequences of real-valued scores  $X = [x_1, x_2, \dots, x_n]$  and  $Y = [y_1, y_2, \dots, y_n]$ , the

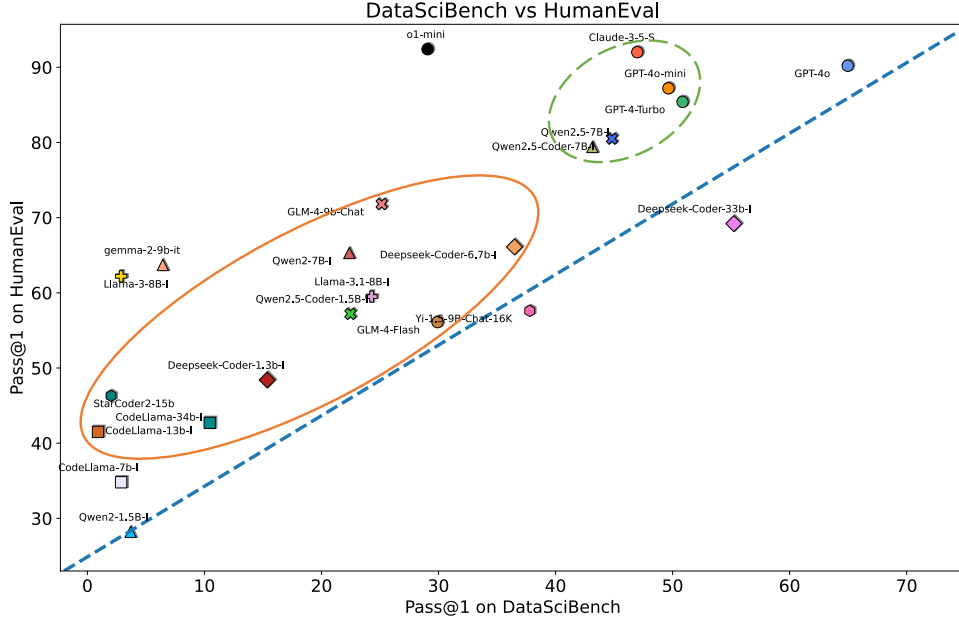


Figure 6: Pass@1 comparison of all tested LLMs between DataSciBench and HumanEval. Circle markers denote the API-based models, while others denote various open-sourced LLMs. The green dashed areas indicate that the LLMs perform well on the two benchmarks, and the orange solid areas indicate performances of the two datasets are relatively mismatched.

Pearson correlation coefficient is:

$$\text{PearsonCorr}(X, Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}, \quad (5)$$

where  $\bar{x}$  and  $\bar{y}$  are the means of sequences  $X$  and  $Y$ , respectively.

**Dimension-level Pearson Correlation.** To validate the alignment in each dimension, we define our metrics and show the results in Table 4. Let each sample  $i \in \{1, \dots, N\}$  be scored along  $D$  dimensions by both the VLM evaluator and human annotators:

$$\mathbf{g}_i = [g_{i1}, g_{i2}, \dots, g_{iD}], \quad \mathbf{h}_i = [h_{i1}, h_{i2}, \dots, h_{iD}]. \quad (6)$$

We compute the average score for each dimension across samples:

$$\bar{g}_j = \frac{1}{N} \sum_{i=1}^N g_{ij}, \quad \bar{h}_j = \frac{1}{N} \sum_{i=1}^N h_{ij}. \quad (7)$$

Forming average vectors  $\bar{\mathbf{g}}, \bar{\mathbf{h}} \in \mathbb{R}^D$ , we define:

$$\text{Dimension-level Pearson Correlation} = \text{PearsonCorr}(\bar{\mathbf{g}}, \bar{\mathbf{h}}). \quad (8)$$

It reflects how well the VLM evaluator aligns human annotators across different aspects.

**Pairwise Correlation.** This is the averaged value for correlations of individual pairs. For each sample  $i \in \{1, \dots, N\}$ , let  $\mathbf{g}_i, \mathbf{h}_i \in \mathbb{R}^D$  denote the

VLM and human score vectors. Then the Pearson correlation per sample is:

$$r_j = \text{PearsonCorr}(\mathbf{g}_i, \mathbf{h}_i). \quad (9)$$

The average over all dimensions defines the pairwise correlation:

$$\text{Pairwise Correlation} = \frac{1}{N} \sum_{i=1}^N r_i. \quad (10)$$

**Cosine Similarity.** It refers to the averaged cosine similarity between GPT-4o-mini and human scores for each sample. For each sample  $i$ , let  $\mathbf{g}_i, \mathbf{h}_i \in \mathbb{R}^D$  denote the VLM and human score vectors. Then:

$$\text{CosineSim}_i = \frac{\mathbf{g}_i \cdot \mathbf{h}_i}{\|\mathbf{g}_i\| \cdot \|\mathbf{h}_i\|}. \quad (11)$$

The average cosine similarity over all samples is:

$$\text{Cosine Similarity} = \frac{1}{N} \sum_{i=1}^N \text{CosineSim}_i. \quad (12)$$

This captures the overall similarity of VLM and human evaluation.

Table 6: Details of programmatic rules.

Aggregate Function	Task	Type	Rule	Comparison	GT	Threshold
Mean Squared Error	Calculate MSE	Decimal	Bool	It $\leq$ to GT is 1, it $>$ GT is 0	Yes	-
Data Cleaning Completeness	row/column number	Integer	Bool	If it == GT, it is 1; if it != GT, it is 0	Yes	-
Silhouette Score	Calculate	Decimal	Bool	If it $\geq$ to GT is 1, if it $<$ GT is 0	Yes	0.5
Model Accuracy	Calculate F1	Decimal	Bool	If it $\geq$ to GT is 1, if it $<$ GT is 0	Yes	0.9
Model Accuracy	Calculate Accuracy	Decimal	Bool	If it $\geq$ to GT is 1, if it $<$ GT is 0	Yes	0.9
R-squared Value	Calculate	Decimal	Bool	If it $\geq$ to GT is 1, if it $<$ GT is 0	Yes	0.9
Data Cleaning Completeness	number of nulls	Integer	Bool	If it == 0, returns 1; if it $>$ 0, returns 0	Yes	-
Data Quality Score	number of outlier	Integer	Bool	If it == GT, it is 1; if it != GT, it is 0	Yes	-
Data Accuracy	MSE	Decimal	Bool	Threshold judgment	Yes	0.05
Data Completeness	Null number ratio	Decimal	Bool	If it == GT, it is 1; if it != GT, it is 0	Yes	-
Data Quality Score	Calculate loss	Decimal	Bool	It $\leq$ to GT is 1, it $>$ GT is 0	Yes	-
Association Rule Confidence	Association rule accuracy	Decimal	Bool	Threshold judgment	Yes	0.9

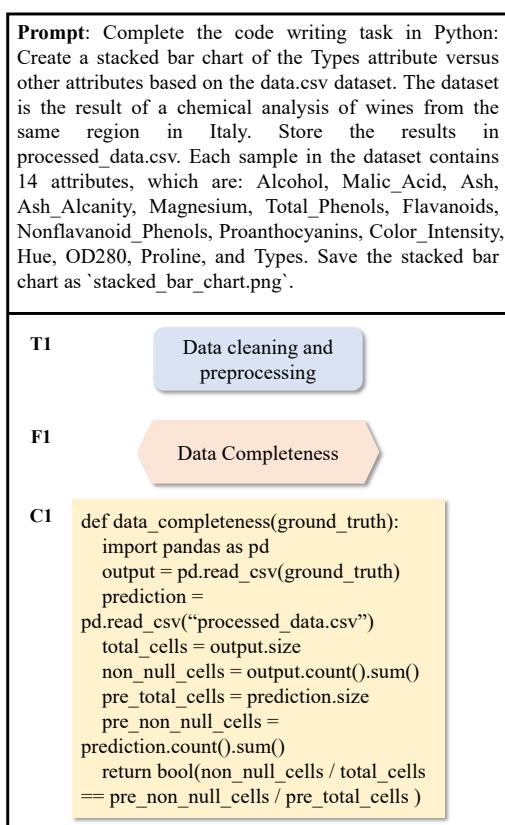


Figure 7: An example of IFC tuples.

### A.5 Motivation and Example of Intention-Function-Code (IFC)

The IFC framework was developed to address several critical challenges in automated evaluation of data science tasks:

- **Systematic Task Selection:** IFC provides a structured approach to identify and categorize key tasks across six established types. This systematic organization ensures comprehensive coverage of essential data science operations and helps maintain evaluation consistency and completeness.

- **Standardized Evaluation Metrics:** Data science tasks often lack standardized evaluation criteria. IFC addresses this by explicitly defining appropriate evaluation functions for each task. For example, data preprocessing tasks require specific metrics that differ from visualization tasks. This standardization ensures fair and consistent assessment.
- **Automated Execution Framework:** IFC includes executable code components for both tasks and evaluation metrics. This automation significantly improves evaluation efficiency, result reproducibility, and testing scalability.
- **Ground Truth Generation:** IFC serves as a crucial foundation for establishing ground truth, particularly valuable for complex tasks where ground truth is not readily available, and enables systematic verification and validation of model outputs.

Overall, the IFC structure in Figure 7 represents a novel contribution by providing a comprehensive framework that bridges the gap between task definition, evaluation criteria, and automated assessment in data science contexts.

### A.6 Programmatic Rules

We list all programmatic rules in Table 6.

### A.7 Error Analysis

To better understand insights and difficulties in LLMs solving data science tasks, we also provide detailed error analysis along with a case study. First we manually study the distribution over five error types shown in Table 7.

Here are some case studies:

Table 7: Error breakdown by model and error type (out of 9 examples).

Model	Error Type				
	Hallucination	Failed to Follow Instructions	Incorrect Output Format	Incorrect Planning	Incorrect Final Answer
	<i>error / total</i>				
GLM-4-Flash	0/9	2/9	0/9	0/9	2/9
Meta-Llama-3-8B-Instruct	2/9	2/9	3/9	0/9	9/9
CodeLlama-7B-Instruct	0/9	2/9	6/9	2/9	9/9
DeepSeek-Coder-1.3B-Instruct	1/9	2/9	4/9	1/9	8/9

### Example 1: Incorrect Planning

**Model:** CodeLlama-7B-Instruct

**Question:** Create a dataframe with the following structure, using different years as row indices and different categories as column names,... The output should be saved in "output.csv".

**Output:** [ { "task\_id": "1", "dependent\_task\_ids": [], "instruction": "Convert the input data to a pandas dataframe.", ... }, { "task\_id": "2", "dependent\_task\_ids": ["1"], "instruction": "Drop the first row as it is the header row.", "task\_type": "preprocessing and cleaning", ... } ]

**Analysis:** The plan is unreasonable. A more appropriate step would be: "instruction": "Set the year as the index and categories as column names".

### Example 2: Incorrect Output Format

**Model:** CodeLlama-7B-Instruct

**Question:** Write a piece of neural network prediction code that establishes a relationship between inputs and outputs, and saves this mapping relationship to a .pkl model. Use the input dataset from "input\_data.csv" (500 rows, 12 columns) and known outputs in "output\_data.csv" (7 rows, 500 columns). Save results in "model.pkl" and evaluation in "evaluation\_results.csv".

**Output:** ...{ "task\_type": "predictive modeling" }...]

**Analysis:** The model fails to produce a valid plan format in the agentic framework. The output is malformed and causes parsing failure.

### Example 3: Failed to Follow Instructions

**Model:** GLM-4-Flash

**Question:** Analyze and visualize the Palmer Archipelago penguin data from Kaggle. Steps include: import dataset "penguins\_lter.csv", clean data, and output "cleaned\_data.csv" and "correlation\_analysis.png".

**Output:** Only outputted "cleaned\_data.csv"

**Analysis:** The model did not generate the required plot file "correlation\_analysis.png" as instructed.

#### Example 4: Hallucination

**Model:** DS-Coder-1.3B-Instruct

**Question:** Write Python code using `pandas.json_normalize()` to convert a JSON input into a CSV named "output.csv". The output should preserve column names for later evaluation.

**Output:** A CSV file with translated English column names: `release_year`, `production_area`, `director`, ...

**Analysis:** The model hallucinated a translation step by converting the Chinese input fields into English, which was not asked for.

### A.8 Caveats when Using LLMs for Data Science

Here we list the issues that occurred during test-case generation, most of which have been addressed by modifying the prompts. We notice that some of the issues may be disruptive to the system and some may be subtle but important.

1. Be careful when using LLMs on well-known open source datasets, especially with customized tasks and data split. LLMs may memorize some open-source datasets. For example, if we want to use part of the penguin dataset that does not contain certain columns, the model (GPT-4o) will still explicitly process those columns in the code.
2. Hallucination during data pre-processing. For example, when the model is required to merge two CSVs, it may hallucinate on a common column and not go through all the columns in the files to find the actual ones.
3. On multilingual tasks. LLMs may not be able to select the correct encoding. For instance, when they are required to open a CSV file that has content in Chinese, they will struggle to choose the correct encoding to open the file. Even if they are hinted that the file is in Chinese, they may choose encodings other than "gbk", e.g., "latin".

### A.9 Related Works

#### A.9.1 Data Science

Recently, some evaluation benchmarks for large language models in data science have been pro-

posed. Text2Analysis (He et al., 2023) constructs the evaluation benchmark to evaluate the model's ability to handle data analysis functions and fuzzy questions on tabular data. Their prompts are obtained through manual annotation and large model generation. Furthermore, DAEval (Hu et al., 2024) is developed as another evaluation benchmark and it contains 257 data analysis questions on CSV data and questions, which are generated by LLMs. However, the prompts in these two works often only involve one task, and these prompts involve relatively simple data analysis operations. In practical data science analysis tasks, user questions often involve multiple tasks and involve performing complex data analysis operations. Therefore, we aim to provide a data science evaluation benchmark that is more in line with practical scenarios, especially for problems involving multiple subtasks and complex data analysis operations.

#### A.9.2 LLMs for Data Science

With the popularity of large-scale language models, researchers have developed a series of LLM-based agents (Yao et al., 2022; Liu et al., 2024; Wang et al., 2024; Xu et al., 2025; Xia et al., 2025). For data science, SheetCopilot (Li et al., 2024) designs a tabular agent, which directly processes natural language-described tasks, and generates and executes a series of operation plans on datasheets to produce the desired results. Data Copilot (Zhang et al., 2024d) is an intelligent agent that serves as a bridge between users and data, which automatically executes data processing, prediction, and visualization tasks based on users' data needs. InsightPilot (Ma et al., 2023) focuses on exploratory data analysis and can automatically discover data insights related to fuzzy questions raised by users. Data interpreter (Hong et al., 2024) augments problem-solving in data science with dynamic planning with hierarchical graph structures, tool integration, and logical inconsistency identification in feedback. Furthermore, ReAct (Yao et al., 2022) is quite similar to our backbone agentic framework, DataInterpreter. Both frameworks offer foundational pipelines rather than specific evaluation metrics tailored for diverse data science tasks. However, the correctness of data analysis in data science has a significant impact on decision-making. Therefore, with the continuous increase of data science agents, it is urgent to conduct a comprehensive and

in-depth evaluation of data science agents.

### **A.9.3 LLM Agent Evaluation Benchmarks for Data Science**

Assessing the effectiveness of LLMs in handling diverse and challenging data science prompts is essential to push the boundaries of benchmarking practices in the field of data science and LLM research (Shao et al., 2026). Data science agents often solve problems by generating code, so the capabilities of data science agents are closely related to the code generation capabilities of large models. There are already many benchmarks for evaluating the code capability of large models. MAgentBench (Huang et al., 2023) benchmarks the LLMs’ abilities on traditional machine learning tasks. NaturalCodeBench (Zhang et al., 2024c) evaluates the capabilities of code generation models on the real prompts from the CodeGeeX (Zheng et al., 2023) platform. However, the general code evaluation benchmark ignores the characteristics of data science tasks and cannot comprehensively and effectively evaluate the capabilities of large models in data science.

Recently, some evaluation benchmarks for large language models in data science have been proposed. Text2Analysis (He et al., 2023) constructs the evaluation benchmark to evaluate the model’s ability to handle data analysis functions and fuzzy questions on tabular data. Their prompts are obtained through manual annotation and large model generation. Furthermore, DAEval (Hu et al., 2024) is developed as another evaluation benchmark and it contains 257 data analysis questions on CSV data and questions, which are generated by LLMs. However, the prompts in these two works often only involve one task, and these prompts involve relatively simple data analysis operations. In practical data science analysis tasks, user questions often involve multiple tasks and involve performing complex data analysis operations. Therefore, we aim to provide a data science evaluation benchmark that is more in line with practical scenarios, especially for problems involving multiple subtasks and complex data analysis operations.

### **A.10 AI Assistants In Writing**

While writing this paper, we employed AI to rectify grammatical errors and revise unreasonable descriptions.

### **A.11 Privacy of Source of prompts**

In online coding services such as CodeGeeX, a subset of users has explicitly granted permission for their data to be used for research purposes only. We have strictly adhered to these requirements and, moreover, have carefully filtered the collected queries to remove any partial or sensitive problems, thereby ensuring there is no risk of information leakage.

### **A.12 Results**

We present the detailed results for our collected prompt and BigCodeBench source in Table 8 and Table 9.

### **A.13 VLM-as-a-judge Prompt and Examples**

In this section, we define the criteria to measure the quality of LLM-generated figures/charts.

Table 8: Evaluation results for DataSciBench (our collected prompts). **Dark gray** and **light gray** indicate the best result among all models and open-source models, respectively. Underline indicates the second-best result among all models.

Models	Size	Coarse-grained Metrics		Fine-grained Metrics					Score	
		Success (%)	CR (%)	VLM	F1	F2	F3	F4		F5
API-Based Models										
o1-mini	N/A	13.45	15.43	2.87	53.75	0.00	45.50	46.25	25.79	19.41
GPT-4o-2024-05-13	N/A	<u>19.82</u>	<b>17.89</b>	<b>3.91</b>	<b>71.79</b>	0.00	<b>54.50</b>	<u>70.00</u>	33.95	<b>24.32</b>
GPT-4o-mini	N/A	12.73	17.35	3.05	62.86	<b>15.00</b>	50.50	69.38	<b>43.68</b>	24.13
GPT-4-Turbo	N/A	17.27	<u>17.36</u>	<u>3.09</u>	65.18	<u>10.00</u>	44.50	<b>78.12</b>	<u>41.05</u>	<u>24.24</u>
Claude-3-5-Sonnet-20240620	N/A	8.00	11.12	2.14	32.32	0.00	36.00	63.12	17.63	15.19
GLM-4.6	N/A	<b>20.00</b>	9.12	0.88	<u>70.24</u>	0.00	<u>53.33</u>	66.47	36.84	18.33
GLM-4.5-Flash	N/A	16.36	4.65	1.13	28.57	0.00	36.67	33.33	3.51	9.14
GLM-4-Flash	N/A	9.82	7.43	1.33	47.14	0.00	22.00	24.38	3.95	10.26
Open-Source General Models										
Meta-Llama-3.1-8B-Instruct	8B	10.00	7.72	1.29	45.18	0.00	16.00	23.12	8.95	10.24
Meta-Llama-3-8B-Instruct	8B	1.64	1.43	0.52	7.86	0.00	4.50	6.88	0.00	2.00
Gemma-2-9B-it	9B	5.64	5.51	1.06	26.79	0.00	13.00	22.50	2.89	7.18
GLM-4-9B-Chat	9B	10.55	9.96	1.69	55.36	0.00	31.00	28.75	<b>21.32</b>	13.91
Qwen2.5-7B-Instruct	7B	11.64	10.11	1.43	55.36	0.00	36.50	33.12	18.42	14.40
Qwen2-7B-Instruct	7B	6.91	5.90	1.16	32.50	0.00	18.00	21.88	2.37	7.97
Qwen2-1.5B-Instruct	1.5B	1.82	1.60	0.40	3.57	0.00	2.00	13.12	0.79	2.13
Yi-1.5-9B-Chat-16K	9B	6.18	4.25	0.73	30.36	0.00	16.00	8.75	3.95	6.06
Open-Source Code Generation Models										
CodeLlama-34B-Instruct	34B	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.02
CodeLlama-13B-Instruct	13B	0.73	0.50	0.04	4.46	0.00	0.00	3.75	0.00	0.77
CodeLlama-7B-Instruct	7B	0.55	0.27	0.00	1.96	0.00	0.00	0.00	0.00	0.30
StarCoder2-15B	15B	0.18	0.20	0.07	0.54	0.00	0.00	0.62	0.00	0.20
DeepSeek-Coder-33B-instruct	33B	<b>12.55</b>	13.53	<b>2.29</b>	62.86	0.00	<b>43.00</b>	51.88	<b>21.32</b>	<b>18.49</b>
DeepSeek-Coder-6.7B-instruct	6.7B	<b>12.55</b>	<b>13.56</b>	1.93	<b>63.21</b>	0.00	39.00	<b>53.75</b>	21.05	18.39
DeepSeek-Coder-1.3B-instruct	1.3B	0.73	0.61	0.10	3.39	0.00	0.00	1.25	0.00	0.67
Qwen2.5-Coder-7B-Instruct	7B	6.18	7.87	1.48	40.18	0.00	27.50	33.75	4.47	10.79
Qwen2.5-Coder-1.5B-Instruct	1.5B	6.18	7.52	0.81	38.57	0.00	15.50	40.00	10.53	10.46

Table 9: Evaluation results for DataSciBench (BigCodeBench source). Dark gray and light gray indicate the best result among all models and open-source models, respectively.

Models	Size	Coarse-grained Metrics		Fine-grained Metrics					Score
		Success (%)	CR (%)	F1	F2	F3	F4	F5	
API-Based Models									
o1-mini	N/A	35.15	55.08	41.62	25.62	32.89	25.90	23.16	47.77
GPT-4o-2024-05-13	N/A	81.62	85.09	77.30	74.63	74.21	71.79	65.48	81.81
GPT-4o-mini	N/A	63.11	71.10	59.46	58.89	60.26	55.90	56.79	67.49
GPT-4-Turbo	N/A	63.35	72.54	61.35	52.04	62.11	54.36	53.95	68.14
Claude-3-5-Sonnet-20240620	N/A	60.48	73.59	54.59	49.11	62.37	49.49	55.39	68.08
GLM-4.6	N/A	81.44	56.10	55.86	58.95	61.40	65.81	47.18	57.80
GLM-4.5-Flash	N/A	71.86	47.38	45.05	53.09	52.63	47.01	48.47	49.07
GLM-4-Flash	N/A	37.07	42.80	33.04	39.11	36.05	28.72	17.89	39.55
Open-Source General Models									
Meta-Llama-3.1-8B-Instruct	8B	29.58	42.51	35.95	24.26	23.95	22.82	31.41	38.16
Meta-Llama-3-8B-Instruct	8B	3.29	4.74	2.97	1.67	2.11	1.28	1.96	3.98
Gemma-2-9B-it	9B	7.54	12.81	25.95	22.46	27.37	16.67	21.84	15.06
GLM-4-9B-Chat	9B	30.72	37.11	23.65	30.78	27.11	26.67	18.42	33.84
Qwen2.5-7B-Instruct	7B	54.43	64.12	49.80	48.40	50.79	49.23	40.15	59.52
Qwen2-7B-Instruct	7B	28.08	32.06	30.41	27.63	32.26	27.18	9.21	30.18
Qwen2-1.5B-Instruct	1.5B	4.67	6.73	4.86	2.63	3.68	3.33	6.00	5.97
Yi-1.5-9B-Chat-16K	9B	48.74	54.9	40.70	48.34	42.11	46.41	35.64	51.53
Open-Source Code Generation Models									
CodeLlama-34B-Instruct	34B	1.20	1.94	1.35	1.11	2.63	2.05	1.58	1.85
CodeLlama-13B-Instruct	13B	13.71	19.3	14.05	15.07	12.53	17.95	6.84	17.52
CodeLlama-7B-Instruct	7B	3.65	5.19	4.05	3.15	3.42	2.31	2.11	4.57
StarCoder2-15B	15B	2.69	3.41	3.24	2.41	2.11	4.36	1.58	3.21
DeepSeek-Coder-33B-instruct	33B	70.12	76.94	66.58	62.63	63.16	64.87	57.59	73.11
DeepSeek-Coder-6.7B-instruct	6.7B	45.09	50.86	37.00	45.96	48.79	44.10	17.11	47.50
DeepSeek-Coder-1.3B-instruct	1.3B	20.36	25.05	16.22	19.44	17.63	21.28	10.53	22.81
Qwen2.5-Coder-7B-Instruct	7B	58.02	68.01	55.34	57.44	49.26	45.38	45.36	63.15
Qwen2.5-Coder-1.5B-Instruct	1.5B	28.20	35.60	26.94	28.96	26.74	26.15	18.32	32.69

### Prompt for VLM-as-a-judge

Above are two figures, which are A and B. The first figure is the ground truth image and the second figure is the predicted image. The total score is 5. Please score B following the criteria below:

- add 1 point for Data Representation Consistency: Ensure that the underlying data represented by the two charts is identical. This includes the values for all data points and the range of the data. Any variation in the dataset used would make the charts different.
- add 1 point for Axis Labels and Scales: Verify that both charts have identical axis labels, units, and scales. Any difference in how the axes are labeled or scaled, such as using logarithmic vs. linear scales, can affect the interpretation of the data.
- add 1 point for Graphical Elements: Check if the visual elements (such as lines, bars, markers, etc.) are represented the same way in both charts. Line thickness, marker styles, and colors should match across charts for them to be considered visually equal.
- add 1 point for Legend and Annotations: Confirm that any legends, titles, or annotations (e.g., text labels, arrows, or highlights) are the same in both charts. These elements often provide crucial context for interpreting the chart.
- add 1 point for Chart Dimensions and Layout: Ensure that the dimensions (height and width), aspect ratios, and layout of the charts are identical. Even if the content and representation are similar, a different aspect ratio or spacing between elements can change the chart's overall appearance and interpretation.

Please write down the total score for B based on the criteria above, and provide a brief explanation of your reasoning. If you believe that the two figures are not identical, please explain the differences you observed.

### Explanation:  
your explanation here

### Total Score:  
x/5

Image A:

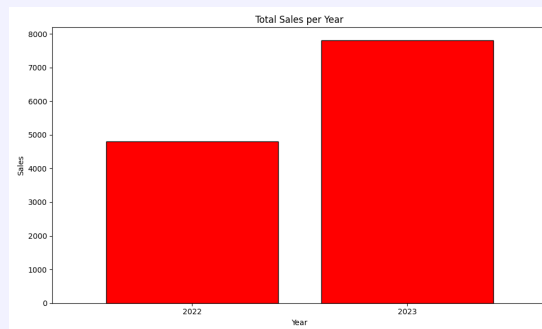
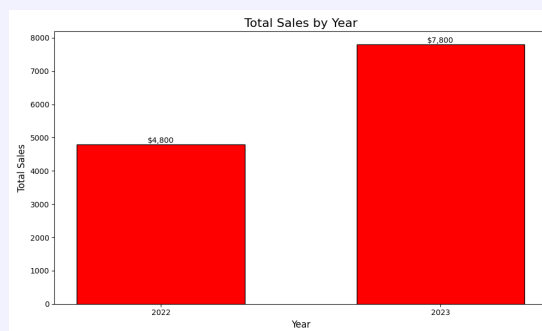


Image B:



[Prompt for VLM-as-a-judge]

### Explanation:

1. **Data Representation Consistency**: Both charts represent the same underlying data for the years 2022 and 2023, with sales figures of \$4,800 for 2022 and \$7,800 for 2023. **+1 point**.
2. **Axis Labels and Scales**: Both charts have identical X-axis labels (years) and Y-axis labels (total sales). They both use the same scale on the Y-axis, ranging from 0 to 8000. **+1 point**.
3. **Graphical Elements**: The bars in both charts are red with the same thickness. Thus, the visual representation matches. **+1 point**.
4. **Legend and Annotations**: The second chart includes dollar amounts on each bar, which isn't present in the first chart. This difference impacts the context provided for interpreting the chart, reducing consistency. **0 points**.
5. **Chart Dimensions and Layout**: Both charts appear similar in terms of layout and dimensions, with no significant visual differences observed. **+1 point**.

Given the above evaluations, the total score for B is calculated as follows:

### Total Score:

4/5

Image A:

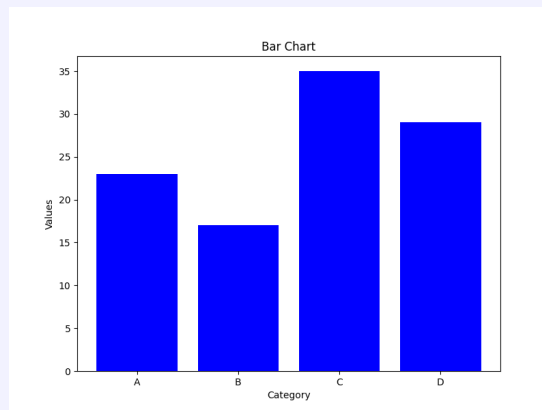
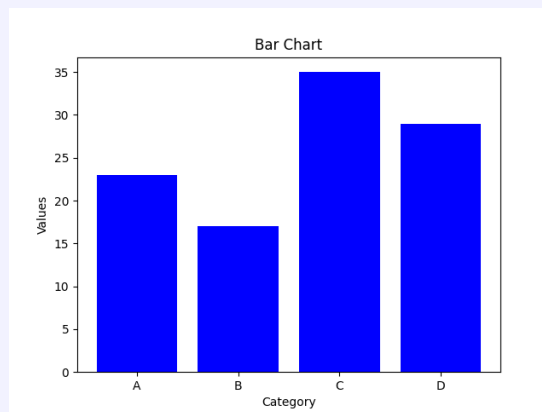


Image B:



[Prompt for VLM-as-a-judge]

### Explanation:

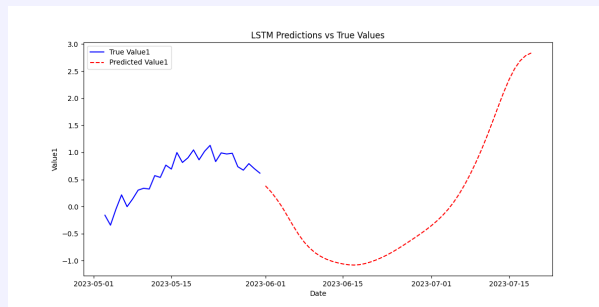
1. **Data Representation Consistency**: Both figures (A and B) represent the same underlying data with identical values for categories A, B, C, and D. Therefore, 1 point is awarded.
2. **Axis Labels and Scales**: The axis labels ("Category" for the x-axis and "Values" for the y-axis) appear to be identical in both images, and the scale for the y-axis seems consistent. Thus, 1 point is awarded.
3. **Graphical Elements**: Both charts use blue bars to represent the data, and the thickness and style of the bars are similar. Therefore, 1 point is awarded.
4. **Legend and Annotations**: There are no legends or additional annotations in either image, and the titles are the same ("Bar Chart"). Thus, 1 point is awarded.
5. **Chart Dimensions and Layout**: The dimensions and overall layout of the two charts differ. Image A has a different aspect ratio compared to Image B, which affects the total layout. Hence, 0 points are awarded.

### Total Score:

4/5

### Example #3 of VLM-as-a-judge (Model: o1-mini)

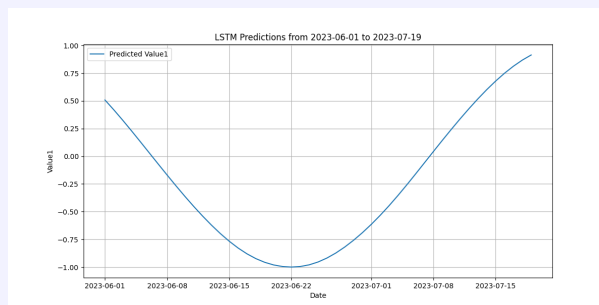
#### Image A:



#### Explanation:

1. **Data Representation Consistency:** Image A represents true values compared to predictions over a broader date range, while Image B only shows predicted values for a specific date range (2023-06-01 to 2023-07-19). The data sets are not identical. **(Score: 0)**
2. **Axis Labels and Scales:** Both images have similar axis labels ("Date" for x-axis, "Value1" for y-axis), but Image B lacks clarity on the date range. Image A provides a wider range, with different intervals. Scales are inconsistent. **(Score: 0)**

#### Image B:



#### Explanation (cont.):

3. **Graphical Elements:** Image A uses a dashed red line for predictions, while Image B uses a solid blue line. This difference affects visual representation. **(Score: 0)**
4. **Legend and Annotations:** Image A includes a detailed legend, while Image B only identifies the predicted values. Titles are also different, affecting context and interpretability. **(Score: 0)**
5. **Chart Dimensions and Layout:** Image A has a wider aspect ratio for a broader date range, while Image B is more condensed. **(Score: 0)**

**Total Score:** 0/5

## A.14 Qualified Prompts

- **Original Prompt 1:**

There is a dataset with missing values in a CSV file, which records the region, height, weight, age, and salary of 36 individuals. Please address the following issues:

(a) Calculate the proportion of missing values in each column and select the rows with at least two non-missing values in the last three columns.

(b) Please fill in the weight column reasonably by combining the data from the height and region columns.

- **Qualified Prompt 1:**

**Qualified Prompt 1:**

Given a dataset with missing values in a file named 'data.csv' which records the region, height, weight, age, and salary of 36 individuals, please address the following issues:

(a) Calculate the proportion of missing values in each column and select the rows with at least two non-missing values in the last three columns. Save your output in a CSV file named 'missing\_values\_proportion.csv'.

(b) Fill in the weight column reasonably by combining the data from the height and region columns. Save this updated dataset in a CSV file named 'updated\_data.csv'.

- **Original Prompt 2:** You are required to analyze and visualize the "Global Terrorism Database" from Kaggle. Please load the dataset and perform data cleaning by handling missing values, removing duplicates, and correcting any anomalies. Conduct an exploratory data analysis (EDA) to understand the distribution and relationships within the dataset. Calculate basic statistical indicators such as mean, median, standard deviation, and provide summary statistics for key features like attack type, target type, and region. Generate visualizations to uncover patterns and insights. Create histograms and box plots to display the distribution of numerical features, and bar plots to show the frequency

of categorical variables. Use scatter plots and heatmaps to visualize relationships and correlations between features. Identify patterns in the data related to terrorist activities. For instance, determine trends over time, geographical hotspots, and common attack methods. Use clustering techniques (K-means clustering) to identify patterns and group similar incidents together.

- **Qualified Prompt 2:**

## Qualified Prompt 2:

You are required to analyze and visualize the *Global Terrorism Database* from Kaggle. Please follow the steps below:

### 1. Load the dataset

Input: `gtd.csv`

Output: `loaded_data.csv` (This should contain the original data loaded without any modifications.)

### 2. Data Cleaning

- Handle missing values
- Remove duplicates
- Correct anomalies

Input: `loaded_data.csv`

Output: `cleaned_data.csv` (This should reflect the cleaned dataset, ready for analysis.)

### 3. Exploratory Data Analysis (EDA)

- Calculate basic statistical indicators such as mean, median, and standard deviation
- Provide summary statistics for key features (attack type, target type, region)

Input: `cleaned_data.csv`

Output: `eda_summary_statistics.csv` (This should include all calculated statistics for key features.)

### 4. Generate Visualizations

- Create histograms and box plots for numerical features
- Generate bar plots for categorical variables
- Use scatter plots and heatmaps to visualize relationships and correlations

Input: `cleaned_data.csv`

Output: `visualizations.pdf` (This should include all visualizations generated in a single PDF file.)

### 5. Identify Patterns in Data Related to Terrorist Activities

- Determine trends over time
- Identify geographical hotspots
- Analyze common attack methods

Input: `cleaned_data.csv`

Output: `patterns_analysis.csv` (This should summarize the identified patterns, trends, and hotspots.)

### 6. Clustering Techniques

- Use K-means clustering to identify patterns and group similar incidents

Input: `cleaned_data.csv`

Output: `clustering_results.csv` (This should include the results of the clustering analysis, showing which group each incident belongs to.)

Ensure that each output file reflects the quality of the completion of the respective subtask for further evaluation.

## A.15 Modified Prompts

Note that we did not restrict the models from using specific libraries. Here, we merely provided an example to guide model output, and this example listed some commonly used libraries for reference. During the generation process, the models were free to use other suitable libraries as needed. The purpose of these examples was to support diverse failed cases rather than to constrain library usage.

- **Original Prompt 1:**

### Original Prompt 1:

Searches a directory for CSV files matching a given regular expression pattern, reads sales data from these files, and plots the sales data with month on the x-axis and sales on the y-axis.

Note that: Each CSV file contains two columns: `Month` and `Sales`.

The function should output with:

A list of `matplotlib.axes._axes.Axes` objects, each representing a plot of sales data from a matched CSV file.

You should write self-contained code starting with:

```
import os
import pandas as pd
import re
import matplotlib.pyplot as plt
def task_func(directory: str, pattern: str) -> list:
```

- **Modified Prompt 1:**

### Modified Prompt 1:

Search a directory for CSV files matching a given regular expression pattern, read sales data from these files, and plot the sales data with month on the x-axis and sales on the y-axis.

#### Input Requirements:

- Input Directory: `data`.
- Input Pattern: `"csv_\d+\\.csv"`.

#### Output Requirements:

1. A list of `matplotlib.axes._axes.Axes` objects representing the plot of sales data from each matched CSV file.
2. Save each plot as a separate image file:
  - File format: PNG
  - Output filenames: `"sales_plot_<filename>.png"` where `<filename>` is the name of the CSV file without the extension.

#### Input File Specification:

- Each CSV file should contain two columns: `'Month'` and `'Sales'`. The input files will be located in the specified directory.

You should write self-contained code starting with:

```
import os
import pandas as pd
import re
import matplotlib.pyplot as plt

def task_func(directory: str, pattern: str) -> list:
```

- **Original Prompt 2:**

**Original Prompt 2:**

Plot a scatter graph of tuples and highlight the tuple with the maximum value at index 1.

The function should output with:

`matplotlib.axes.Axes`: The Axes object of the plot for further manipulation and testing, with the title 'Max Tuple Highlighted', x-axis labeled 'x', y-axis labeled 'y', and a legend.

You should write self-contained code starting with:

```
import numpy as np
from operator import itemgetter
import matplotlib.pyplot as plt
def task_func(data):
```

- **Modified Prompt 2:**

**Modified Prompt 2:**

Plot a scatter graph of tuples and highlight the tuple with the maximum value at index 1 using the input data from "data.csv". The function should output the following:

A scatter plot saved as "scatter\_plot.png" with the title 'Max Tuple Highlighted', x-axis labeled 'x', y-axis labeled 'y', and a legend. The highlighted point should signify the tuple with the maximum value at index 1.

Please write self-contained code starting with:

```
import numpy as np
from operator import itemgetter
import matplotlib.pyplot as plt
def task_func(data):
```

## A.16 Prompt examples of different difficulty levels

**Easy-level Prompt #1:**

```
{
  "prompt": "Read the dataset (input file: "Fish.csv"). Encode the dataset to divide it into training and test sets. From the dataset's four categories of Bream, Roach, Parkki, and Perch, randomly select 2 samples from each category for the test set. The remaining samples will be used as the training set. Output the number of samples in the training and test sets in a CSV file named "sample_counts.csv".
```

```
Next, implement the KNN algorithm with K values of 1, 5, 15, and 100 to classify all samples in the test set. Output the classification results of the test samples to a CSV file named "classification_results.csv".",
```

```
"data_source_type": "2=open source data"
}
```

## Easy-level Prompt #2:

```
{
  "prompt": "Using the dataset of house prices and economic indicators provided below, please
  complete the following subtasks. Each subtask should generate an output file for evaluation, with
  specific input files where applicable.

  Dataset: \"data.csv\"

  Subtasks:

  1. Gaussian Naive Bayes Training: Train a Gaussian Naive Bayes model on the cleaned dataset
  and evaluate its accuracy.
  Input: \"data.csv\"
  Output: \"gaussian_model_accuracy.csv\" (contains accuracy metrics)

  2. Multinomial Naive Bayes Training: Train a Multinomial Naive Bayes model on the cleaned
  dataset and evaluate its accuracy.
  Input: \"data.csv\"
  Output: 'multinomial_model_accuracy.csv' (contains accuracy metrics)

  3. Comparison of Models: Generate a comparative analysis of the accuracy of both models in a
  visual format.
  Input: \"gaussian_model_accuracy.csv\", \"multinomial_model_accuracy.csv\"
  Output: \"model_comparison.png\"

  Please ensure that the outputs contain necessary details for further evaluation.\",

  \"data_source_type\": \"3=human written data\"
}
```

### Medium-level Prompt:

```
{
  "prompt": "Given a DataFrame 'df' with sales data containing columns: TransactionID, ProductID, Quantity, SaleDate, StoreID, Revenue:

  1. Handle missing Revenue values: Impute any missing Revenue values with the median of the Revenue column.
  Input file: 'data.csv'
  Output: 'step1_imputed_revenue.csv' (contains the DataFrame after handling missing values)

  2. Detect and replace outliers: Detect and replace outliers in Quantity and Revenue. Outliers are defined as values below the 1st percentile or above the 99th percentile. Replace them with the corresponding 1st or 99th percentile value instead of removing them.
  Input file: 'step1_imputed_revenue.csv'
  Output: 'step2_replaced_outliers.csv' (contains the DataFrame after outlier replacement)

  3. Normalize Quantity and Revenue: Normalize Quantity and Revenue using Z-score normalization.
  Input file: 'step2_replaced_outliers.csv'
  Output: 'step3_normalized_data.csv' (contains the DataFrame after normalization)

  4. Ensure SaleDate format: Ensure SaleDate is in datetime format.
  Input file: 'step3_normalized_data.csv'
  Output: 'step4_formatted_dates.csv' (contains the DataFrame after ensuring datetime format)

  5. Encode ProductID and StoreID: Encode the ProductID and StoreID columns using separate label encoders to avoid any potential overlap in numerical values between categories from different columns.
  Input file: 'step4_formatted_dates.csv'
  Output: 'final_cleaned_data.csv' (contains the final cleaned DataFrame)

  Perform the specified data cleaning and preprocessing tasks and output the cleaned DataFrame as the final result.",

  "data_source_type": "3=human written data"
}
```

### Hard-level Prompt:

{ "prompt": "Write a Graph Recurrent Neural Network (GRNN) model based on attention mechanisms using Python for processing and analyzing time series data. Ensure to meet the following requirements:

1. "Graph network design": Create a graph network where each graph represents an aerial formation, and the number of nodes corresponds to the number of vehicles in the formation.

- "Output": Graph structure representation file.

- "Output filename": "graph\_structure.json"

2. "Data format": The time series data of aerial target formations is stored in Excel files located in the ".data/" directory, where each Excel file contains multiple sheets, with each sheet representing the time series data of a vehicle.

- "Input": Excel files from "..data/" directory.

- "Input filename": from "data\_1.xlsx" to "data\_5.xlsx"

3. "Data reading": Read all Excel files from the "..data/" directory and extract the sheets for processing.

- "Output": Combined time series data in a structured format.

- "Output filename": "combined\_data.csv"

4. "Data segmentation": Segment the time series data of each vehicle using a sliding window with length "l" and step size "s".

- "Output": Segmented time series data.

- "Output filename": "segmented\_data.csv"

5. "Training and testing set division": Divide the segmented data into training and testing sets with a 7:3 ratio.

- "Output": Training and testing datasets.

- "Output filenames": "train\_data.csv", "test\_data.csv"

6. "Labels": The last column of each Excel file contains label data, which should be preserved during the reading and segmentation process.

7. "Model training": Train the GRNN model using the training set to identify the action intentions of aerial target formations.

- "Output": Trained model file.

- "Output filename": "trained\_model.h5"

8. "Training parameters": Set the number of training epochs to 60, and record the loss value, accuracy, recall, and F1-Score for each epoch.

- "Output": Training metrics log file.

- "Output filename": "training\_metrics.csv"

9. "Model evaluation": Evaluate the trained model with the test set and generate a classification report and confusion matrix.

- "Output": Classification report and confusion matrix visualization.

- "Output filenames": "classification\_report.txt", "confusion\_matrix.png"

Please provide a complete code implementation and ensure that the code structure is clear and well-commented for understanding and evaluation.",

"data\_source\_type": "3=human written data" }

## **B Error Analysis and Case Study**

We have analyzed the common errors during problem-solving, summarized them here, and then studied several cases, including a successful case and a failure case. The common errors include:

1. Coding errors when solving data science problems using codes. And based on our observation, the main kind of these is execution error. It may be due to different reasons. For example, hallucination on the column name of a csv file.
2. Json format errors. These errors come from the agent framework side, where they use json format to wrap up actions, e.g. `WriteAnalysis`.

### **B.1 Successful cases**

**Prompt:** You are given a list of tickers and their daily closing prices for a given period stored in an input file named "data.csv". Implement the `most_corr` function that, when given the daily closing prices, returns the pair of tickers that are the most highly (linearly) correlated by daily percentage change. The function should follow these specifications:

1. **Input File:** `./data.csv` - A CSV file containing a dataframe with columns representing each ticker's daily closing prices.
2. **Output Requirements:** The function should generate the following output:
  - **Output Format:** A CSV file.
  - **Output Filename:** `most_corr_output.csv` - This file should contain two columns labeled "Ticker 1" and "Ticker 2", representing the pair of tickers that are most correlated by daily percentage change.

Here's the starter code for your implementation:

```
```python
import pandas as pd
import numpy as np

def most_corr(prices):
    """
    :param prices: (pandas.DataFrame) A dataframe containing
    each ticker's daily closing prices.
    :returns: (container of strings) A container, containing
    the two tickers that are the most highly (linearly)
    correlated by daily percentage change.
    """
    return None

# For example, the code below should print: ('FB', 'MSFT')
print(most_corr(pd.DataFrame.from_dict({
    'GOOG' : [
        742.66, 738.40, 738.22, 741.16,
        739.98, 747.28, 746.22, 741.80,
        745.33, 741.29, 742.83, 750.50
    ],
    'FB' : [
        108.40, 107.92, 109.64, 112.22,
        109.57, 113.82, 114.03, 112.24,
        114.68, 112.92, 113.28, 115.40
    ],
    ...
    [omitted]
})))
```
```

**Output:** The following example successfully fulfills the requirement of the given prompt by completing the function provided in the prompt and write the function to `most_corr_output.csv`.

```
[
  {
    'task_id': '1',
    'dependent_task_ids': [],
    'instruction': "Load and preprocess the input data from
'data.csv'",
    'task_type': 'preprocessing and cleaning',
    'code': '''
import pandas as pd
import numpy as np

# Load the data from 'data.csv'
prices = pd.read_csv('../data.csv')

# Set the first column (assuming it's a date column) as
the index
prices.set_index(prices.columns[0], inplace=True)

# Calculate daily percentage changes
pct_changes = prices.pct_change()

# Drop the first row (NaN values due to pct_change
calculation)
pct_changes = pct_changes.dropna()

print("Data loaded and preprocessed. Shape of \
percentage changes:", pct_changes.shape)
print("Columns (tickers):", \
pct_changes.columns.tolist())
print("First few rows of percentage changes:")
print(pct_changes.head())
''',
    'result': "Data loaded and preprocessed. Shape of \
percentage changes: (29, 4)
Columns (tickers): ['Ticker2', 'Ticker3', 'Ticker4', \
'Ticker5']
First few rows of percentage changes:
Ticker2  Ticker3  Ticker4  Ticker5
Ticker1
65.60   -0.615289  0.108929  0.002184  0.841616
52.06    1.633757 -0.024740 -0.353101 -0.435643
68.34   -0.452888 -0.230862  0.308297  0.160458
111.19  -0.204800 -0.227069 -0.070287  0.208418
128.52   0.094136  0.280394  0.260849 -0.428407
",
    'is_success': True,
    'is_finished': True
  },
]
```

```

{
  'task_id': '2',
  'dependent_task_ids': ['1'],
  'instruction': 'Implement the most_corr function to
find the most correlated pair of tickers',
  'task_type': 'data mining',
  'code': '''
def most_corr(prices):
    """
    :param prices: (pandas.DataFrame) A dataframe
    containing each ticker's
        daily closing prices.
    :returns: (container of strings) A container,
    containing the two tickers that
        are the most highly (linearly) correlated
        by daily percentage change.
    """
    # Calculate percentage changes
    pct_changes = prices.pct_change().dropna()

    # Calculate correlation matrix
    corr_matrix = pct_changes.corr()

    # Set diagonal and lower triangle to NaN to avoid
    self-correlation and duplicates
    corr_matrix = corr_matrix.where(np.triu( \
    np.ones(corr_matrix.shape), k=1).astype(bool))

    # Find the location of the maximum correlation
    max_corr = corr_matrix.max().max()
    max_corr_idx = corr_matrix.stack().idxmax()

    # Get the tickers with the highest correlation
    ticker1, ticker2 = max_corr_idx
    ...
    [omitted]

    result = most_corr(prices)
    print(f"The most correlated pair of tickers: {result}")

    output = pd.read_csv('./most_corr_output.csv')
    print("\nContents of most_corr_output.csv:")
    print(output)''',
  'result': "The most correlated pair of tickers: ...
[omitted]
'is_success': True,
'is_finished': True
},

```

```

{
  'task_id': '3',
  'dependent_task_ids': ['2'],
  'instruction': "Generate and save the output CSV file
'most_corr_output.csv'",
  'task_type': 'other',
  'code': '''
import pandas as pd

# Read the output CSV file
output_df = pd.read_csv('./most_corr_output.csv')

# Display the contents of the file
print("Contents of most_corr_output.csv:")
print(output_df)

# Verify that the file contains the expected columns
expected_columns = ['Ticker 1', 'Ticker 2']
assert all(col in output_df.columns for col in \
expected_columns), "Output file is missing expected \
columns"

# Verify that the file contains exactly one row
assert len(output_df) == 1, "Output file should \
contain exactly one row"

print("Output file verification complete. The file \
meets the expected format.")''',
  'result': 'Contents of most_corr_output.csv:
Ticker 1 Ticker 2
0 Ticker4 Ticker5
Output file verification complete. The file meets the
expected format.
',
  'is_success': True,
  'is_finished': True
}
]

```

## B.2 Failure Case

## Failed Example #1 Generated by o1-mini

### Prompt:

Draw normal distributions for multiple 'x' and 'y' arrays with labels. Each pair (x, y) represents a different chemical compound in the 'labels' list. **The function should output with: tuple(fig: Matplotlib.figure.Figure) fig: Matplotlib figure object containing the drawn normal distributions.** You should write self-contained code starting with:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
def task_func(x, y, labels):
```

### Completion:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import pandas as pd

def task_func(x, y, labels):
    # Create a DataFrame from the input arrays
    data = pd.DataFrame({
        'x': x,
        'y': y,
        'label': labels
    }).copy()

    # Detect and handle missing values
    if data.isnull().values.any():
        data = data.dropna()

    # Remove duplicate entries
    data = data.drop_duplicates()

    # Detect outliers using Z-score and remove them
    numeric_cols = ['x', 'y']
    z_scores = np.abs(stats.zscore(data[numeric_cols]))
    data = data[(z_scores < 3).all(axis=1)]

    # Ensure that the lengths of x, y, and labels are aligned \
    after preprocessing
    x_processed = data['x'].values
    y_processed = data['y'].values
    labels_processed = data['label'].values

    return x_processed, y_processed, labels_processed \
# error: the answer does not comply with required output instructions.
```

### Reason:

The answer does not comply with the required output instruction.

## Failed Example #2 Generated by o1-mini

### Prompt:

Reads data from a CSV file and generates a bar plot based on grouped mean values. The DataFrame is grouped by the column named 'col1\_name', and the mean for each group is calculated for the column 'col2\_name'. A bar plot is created using matplotlib. Each bar in the plot represents a group, and its height corresponds to the mean value of 'col2\_name' for that group. The plot is then configured with a title and axis labels: - The title is set as 'Mean of [col2\_name] Grouped by [col1\_name]'. This format dynamically inserts the names of the columns being analyzed into the title. - The xlabel (label for the x-axis) is set to the name of the column used for grouping (col1\_name). - The ylabel (label for the y-axis) is set as 'Mean of [col2\_name]', indicating that the y-axis represents the mean values of the specified column. Note that: Ensure that the CSV file exists at the specified path and has the required columns. The function does not handle missing data. Ensure that the CSV file has clean and complete data for accurate results. The bar plot is customizable using matplotlib's functionality after the function returns the Axes object. **The function should output with: tuple(ax: matplotlib.axes.Axes, df\_mean: pandas.DataFrame) ax: The Axes object of the generated bar plot. This object can be used to further customize the plot, like adding labels or changing styles. df\_mean: The DataFrame containing the mean values calculated for each group.** You should write self-contained code starting with:

```
import pandas as pd
import matplotlib.pyplot as plt
def task_func(csv_file_path, col1_name="column1", col2_name="column2"):
```

### Completion:

```
import pandas as pd
import matplotlib.pyplot as plt

def task_func(csv_file_path, col1_name="column1", col2_name="column2"):
    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(csv_file_path)
    # Create a copy to preserve the original dataset
    df_clean = df.copy()
    # Handle missing values
    # Drop rows with any missing values
    df_clean.dropna(inplace=True)
    # Remove duplicate entries
    df_clean.drop_duplicates(inplace=True)
    # Handle outliers in 'col2_name' using the IQR method
    Q1 = df_clean[col2_name].quantile(0.25)
    Q3 = df_clean[col2_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_clean = df_clean[(df_clean[col2_name] >= lower_bound) \
    & (df_clean[col2_name] <= upper_bound)]
    # Standardize data formats if necessary
    # Example: If 'col1_name' contains date information, \
    uncomment the following line
    # df_clean[col1_name] = pd.to_datetime(df_clean[col1_name])

    return df_clean # error: the answer failed to output the required \
    elements mentioned in the instruction.
```

### Reason:

The answer failed to output the required elements mentioned in the instruction.

### Failed Example #3 Generated by o1-mini

**Prompt:**

Perform K-Means clustering on the given DataFrame using the sklearn KMeans algorithm. The function expects a DataFrame with numerical values, as KMeans cannot handle categorical data. It applies standard KMeans clustering from the sklearn library to form clusters. The number of clusters is configurable via the 'n\_clusters' parameter, defaulting to 3. The Number of times the k-means algorithm is run with different centroid seeds (n\_init) is set to 10. The function returns an array of cluster labels corresponding to each data point in the input as well as the fitted KMeans model. »> data = pd.DataFrame( ... 'a': [1, 20, 2, 22, 100], ... 'b': [1, 20, 2, 22, 100] ... ) »> labels, model = task\_func(data, seed=213) »> print(labels) [2 0 2 0 1] »> print(model) KMeans(n\_clusters=3, n\_init=10, random\_state=213). The function should raise the exception for: ValueError: If the DataFrame contains non numeric entries. The function should output with: tuple(labels: numpy.ndarray, model: sklearn.cluster.KMeans) labels: An array of integers (cluster labels) corresponding to the input data. Each label is an integer representing the cluster to which a row of data has been assigned. model: The fitted KMeans Model. You should write self-contained code starting with:

```
import pandas as pd
from sklearn.cluster import KMeans
def task_func(data, n_clusters=3, seed=None):
```

**Completion:**

```
import numpy as np
from scipy import stats # error

def task_func(data, n_clusters=3, seed=None):
    # Make a copy of the DataFrame to preserve the original
    df = data.copy()

    # Check if all columns are numeric
    if not all(pd.api.types.is_numeric_dtype(dtype) \
for dtype in df.dtypes):
        raise ValueError("The DataFrame contains non-numeric entries.")

    # Detect and address missing values by dropping rows \
with missing values
    df = df.dropna()

    # Remove duplicate entries
    df = df.drop_duplicates()

    # Detect and handle outliers by removing rows with z-score > 3
    z_scores = np.abs(stats.zscore(df))
    # error: the model failed to call sklearn.cluster functions, \
instead only calling zscore to remove outliers.
    df = df[(z_scores < 3).all(axis=1)] # error

    # At this stage, further preprocessing can be added as needed
    return df
```

**Reason:**

The model failed to call sklearn.cluster functions, instead of only calling zscore to remove outliers.

## Failed Example #4 Generated by o1-mini

### Prompt:

Reads data from a CSV file and generates a bar plot based on grouped mean values. The DataFrame is grouped by the column named 'col1\_name', and the mean for each group is calculated for the column 'col2\_name'. A bar plot is created using matplotlib. Each bar in the plot represents a group, and its height corresponds to the mean value of 'col2\_name' for that group. The plot is then configured with a title and axis labels: - The title is set as 'Mean of [col2\_name] Grouped by [col1\_name]'. This format dynamically inserts the names of the columns being analyzed into the title. - The xlabel (label for the x-axis) is set to the name of the column used for grouping (col1\_name). - The ylabel (label for the y-axis) is set as 'Mean of [col2\_name]', indicating that the y-axis represents the mean values of the specified column. Note that: Ensure that the CSV file exists at the specified path and has the required columns. The function does not handle missing data. Ensure that the CSV file has clean and complete data for accurate results. The bar plot is customizable using matplotlib's functionality after the function returns the Axes object. The function should output with: tuple(ax: matplotlib.axes.Axes, df\_mean: pandas.DataFrame) ax: The Axes object of the generated bar plot. This object can be used to further customize the plot, like adding labels or changing styles. df\_mean: The DataFrame containing the mean values calculated for each group. You should write self-contained code starting with:

```
import pandas as pd
import matplotlib.pyplot as plt
def task_func(csv_file_path, col1_name="column1", col2_name="column2"):
```

### Completion:

```
# error: the model forgot to import relevant libraries.
def task_func(csv_file_path, col1_name="column1", col2_name="column2"):
    # Read the CSV file
    df = pd.read_csv(csv_file_path)

    # Create a copy for cleaning to preserve the original DataFrame
    df_clean = df.copy()

    # 1. Handle Missing Values
    # Check for missing values
    missing_counts = df_clean.isnull().sum()
    print("Missing Values:\n", missing_counts)

    # Option 1: Drop rows with missing values
    df_clean.dropna(inplace=True)

    # Option 2: Alternatively, you can fill missing values
    # df_clean[col2_name].fillna(df_clean[col2_name].mean(), inplace=True)

    # 2. Remove Duplicate Entries
    duplicates = df_clean.duplicated().sum()
    print(f"Number of duplicate rows: {duplicates}")
    df_clean.drop_duplicates(inplace=True)

    # 3. Handle Outliers
    # Example using Z-score for numerical columns
    numeric_cols = df_clean.select_dtypes(include=['float64', 'int64']).columns
    from scipy import stats
    z_scores = np.abs(stats.zscore(df_clean[numeric_cols]))
    df_clean = df_clean[(z_scores < 3).all(axis=1)]

    # 4. Standardize Data Formats
    # Example: Standardize date columns
    for col in df_clean.columns:
        if 'date' in col.lower():
            df_clean[col] = pd.to_datetime(df_clean[col], errors='coerce')
```

## Failed Example #4 Generated by o1-mini

```
# Drop rows with invalid date formats after conversion
df_clean.dropna(subset=[col for col in df_clean.columns
if 'date' in col.lower()], inplace=True)

# 5. Additional Preprocessing (if needed)
# For example, convert categorical columns to consistent case
categorical_cols = df_clean.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df_clean[col] = df_clean[col].str.strip().str.lower()

# Continue with the original processing using the cleaned DataFrame
df_mean = df_clean.groupby(coll_name)[col2_name].mean().reset_index()
fig, ax = plt.subplots()
ax.bar(df_mean[coll_name], df_mean[col2_name], color='skyblue')
ax.set_title(f"Mean of {col2_name} Grouped by {coll_name}")
ax.set_xlabel(coll_name)
ax.set_ylabel(f"Mean of {col2_name}")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

print("Basic Statistical Indicators:")
print(df_clean.describe())
print("\nMedian Values:")
print(df_clean.median())
print("\nStandard Deviation:")
print(df_clean.std())

df_clean[numeric_cols].hist(bins=10, figsize=(10, 8), edgecolor='black')
plt.suptitle("Histograms of Numerical Columns")
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

df_clean.boxplot(column=numeric_cols, figsize=(10, 6))
plt.title("Box Plots of Numerical Columns")
plt.tight_layout()
plt.show()

corr_matrix = df_clean.corr()
fig, ax = plt.subplots(figsize=(8, 6))
cax = ax.matshow(corr_matrix, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap', pad=20)
fig.colorbar(cax)
ticks = range(len(corr_matrix.columns))
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(corr_matrix.columns, rotation=90)
ax.set_yticklabels(corr_matrix.columns)
plt.tight_layout()
plt.show()

return ax, df_mean
```

**Reason:**

The model forgot to import relevant libraries.

## A failure case of execution error

### Prompt:

I currently have a structured dataset named `data.csv` with 6 columns and 5960 rows of data. The first column is named X, the second column is named Y, both of which are structural parameters set for simulation. The following four columns, `ff`, `ft`, `tt`, and `tf`, represent the simulation results. Now, I need to train a model that can accurately predict the four simulation results for other structural parameters. The model and related content will be saved in the results folder. The model should be saved as `"model.pkl"` and the final accuracy should be saved in `"acc.txt"`.

### One of the Generated Functions:

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import minimize

# Read in the input data from the 'data.csv' file
data = pd.read_csv('../data.csv')

# Preprocess the data by removing any unnecessary columns and
handling missing values
data.dropna(inplace=True)
X = data.drop('target', axis=1)
y = data['target']
... (omitted)
```

### Execution Error:

```
-> 6992         raise KeyError(f"{labels[mask].tolist()} not
found in axis")
      6993         indexer = indexer[~mask]
      6994         return self.delete(indexer)
```

```
KeyError: "['target'] not found in axis"
```

Generated by CodeLlama-34B-Instruct in one task:

```
[PYTHON]
# Define the context and task
context = {
    "data": "https://s3.amazonaws.com/benbensaid-public-data/data.csv",
    "target": "y"
}

task = {
    "task_type": "data exploration",
    "instruction": "Explore the data and identify
any patterns or trends."
}

# Generate the plan
plan = generate_plan(context, task)

# Print the plan
print(plan)
[/PYTHON]
```

Generated by StarCoder2-15B in one task:

Sure, here's a plan for your context:

1. Find the videos with titles containing "how" or "what"
2. Calculate the like ratios
3. Visualize the like ratios

- **The expected format should at least begin and end with three backticks and must comply with JSON format. Here's an example:**

Expected JSON format:

```
"
[
  {
    "task_id": "1",
    "dependent_task_ids": [],
    "instruction": "Preprocess and clean the data.",
    "task_type": "preprocessing and cleaning"
  },
  {
    "task_id": "2",
    "dependent_task_ids": ["1"],
    "instruction": "Explore the data to find
patterns and insights.",
    "task_type": "data exploration"
  },
  {
    "task_id": "3",
    "dependent_task_ids": ["2"],
    "instruction": "Visualize the data to communicate insights.",
    "task_type": "data visualization"
  },
  {
    "task_id": "4",
    "dependent_task_ids": ["3"],
    "instruction": "Generate an interpretability report.",
    "task_type": "interpretability report"
  }
]
"
```