CIBench: Evaluating Your LLMs with a Code Interpreter Plugin

Anonymous ACL submission

Abstract

While LLM-Based agents, which use external tools to solve complex problems, have made significant progress, benchmarking their ability is challenging, hindering a clear understanding of their limitations. In this paper, we propose an evaluation framework, named `CIBench` , to comprehensively assess LLMs' ability to utilize code interpreters for data science tasks. Our evaluation framework includes an evaluation dataset and two evaluation modes. The evaluation dataset is constructed using an LLM-human cooperative approach and simulates an authentic workflow by leveraging consecutive and interactive IPython sessions. The two evaluation modes assess LLMs' ability with and without human assistance. We conduct extensive experiments to analyze the ability of 19 LLMs on `CIBench` and provide valuable insights for future LLMs in data science tasks.

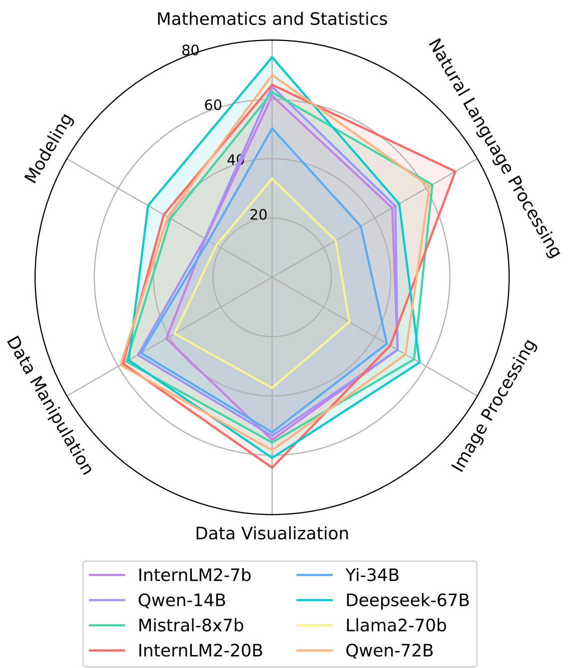


Figure 1: Performance of open-sourced LLMs with code interpreters across different module categories. Refer to Tab.1 for category details. LLMs perform poorly in the modeling modules (Pytorch, TensorFlow, etc.).

1 Introduction

Empowered by the emerging abilities of Large Language Models (LLMs), numerous agents, such as MetaGPT (Hong et al., 2023), ToolLLM (Qin et al., 2023), LangChain (Chase, 2022), AutoGPT (Significant Gravitas), and QwenAgent (Bai et al., 2023), have surfaced to harness these generalist models for utilizing external tools like web browsing, document retrieval and code interpreter in tackling complex real-world problems. Specifically, agents with a code interpreter leverage the advanced programming skills of LLMs through a natural language interface, facilitating the creation of new workflows that are both effortless and efficient. *However, measuring the agent's ability to use code interpreters remains challenging, hindering a clear understanding of their limitations.*

In this study, we focus on assessing the proficiency of LLMs in leveraging code interpreters to address data science problems across several distinct domains, like data analysis, visualization, and machine learning. These tasks necessitate that LLMs exhibit advanced capabilities in instruction following, reasoning, and programming. Despite this, existing benchmarks, including GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021), primarily assess the models' abilities to solve mathematical or coding problems solely. These benchmarks, while valuable for measuring specific competencies, fail to fully represent the LLMs' aptitude for building complex workflows involving code interpreters in practical applications.

More recent efforts aim to bridge the existing gap by constructing novel benchmarks for code interpreters, particularly tailored to mathematical problems and data analysis tasks (Bai et al., 2023;

Hu et al., 2024). Despite considerable advancements, the prevailing studies either focus on single-turn question assessments or have a limited scope in data science. The substantial insights they provided inadequately mirror the dynamic, interactive, and multi-turn nature of real-world scenarios in which agents employ code interpreters.

To address these shortcomings, we introduce a novel evaluation framework that encompasses a novel benchmark and innovative assessment protocols to provide a comprehensive evaluation of LLMs' ability to use code interpreters. Notably, the benchmark is devised in a distinctive *LLM-human cooperative approach*, integrating generated tasks autonomously produced by LLMs alongside template tasks meticulously crafted by human experts, as depicted in Fig. 2. It ingeniously simulates authentic workflow scenarios by leveraging *interactive IPython* sessions, which entail sequential, interconnected questions and concentrate on popular Python modules such as *Matplotlib*, *Pandas*, and *PyTorch*. Moreover, we deploy two newly devised evaluation modes coupled with a series of fine-grained metrics to methodically gauge LLMs' ability in code interpreter manipulation. This approach promises to yield valuable insights to inform future enhancements and optimizations.

Specifically, we build the evaluation dataset by initially identifying ten highly prevalent Python libraries within the domain of data science. Then, the generated tasks are realized by prompting advanced LLM, such as GPT-4, to generate instructions and code snippets within Jupyter Notebook format. Each notebook is structured to contain a sequence of 10 to 15 progressive steps, with increasing levels of complexity. To mitigate any inherent biases or limitations in the LLM-generated content, we introduce template tasks that are specifically designed by human experts based on common patterns observed in the LLM-generated tasks and online resources. The template tasks can incorporate multiple interchangeable datasets for evaluation. Those designs ensure that the benchmark encapsulates both diversity and quality, thereby offering a comprehensive and balanced assessment of code interpreter capabilities.

To thoroughly assess the LLMs' performance on our benchmark, we have instituted two distinct evaluation modes: the *end-to-end mode* and the *oracle mode*. In the end-to-end mode, LLMs are tasked with a holistic problem-solving process where they must reason through given instructions and generate corresponding code. This requires them to iteratively refine their output based on feedback from the code interpreter, as they attempt to solve multiple consecutive questions that build upon one another. Additionally, the oracle mode simulates guided learning by providing the LLM with the correct code snippet when it fails. This immediate feedback mimics human guidance and equips the model to use this accurate example for tackling subsequent tasks in the same context. Furthermore, we introduce two types of metrics: the process-oriented (*i.e., tool-call rate, executable rate*) and output-oriented (*i.e., numeric accuracy, text score, visualization score*), to provide a comprehensive analysis of the model's performance.

Based on our evaluation framework, we conduct extensive experiments and analysis using 19 LLMs. The results indicate that LLMs struggle to utilize PyTorch- and TensorFlow-like modules (Fig.1), and the best-open-sourced LLMs lag behind GPT-4 by 6.8%. In summary, our contributions are three-fold:

- We build a new benchmark for agents with code interpreters using an LLM-human cooperative method. It consists of interactive IPython sessions with interconnected questions on key data science libraries, simulating dynamic multi-turn problem-solving in practical workflows.
- We devise unique assessment strategies involving both end-to-end and oracle modes. We also introduce evaluation metrics that span both process and output quality, offering a comprehensive gauge of LLMs' coding prowess within the benchmark.
- We conduct thorough experiments with 19 LLMs to analyze their performance on our benchmark. The results indicate that LLMs perform poorly in the modeling category modules and open-sourced LLMs are inferior to GPT-4 by a large margin.

2 CIBench

To benchmark LLM's ability to leverage code interpreters for addressing data science problems, we propose a novel evaluation framework (Fig.2), which comprises a diverse evaluation dataset and two newly devised evaluation modes. The evaluation dataset is generated through an LLM-human cooperative approach and simulates authentic workflow scenarios for solving sequential and interconnected tasks. Given the evaluation dataset, we adhere to the ReAct protocol (Yao et al., 2023) to generate reasoning traces and invoke code interpreters alternately. And, we allow LLMs to attempt

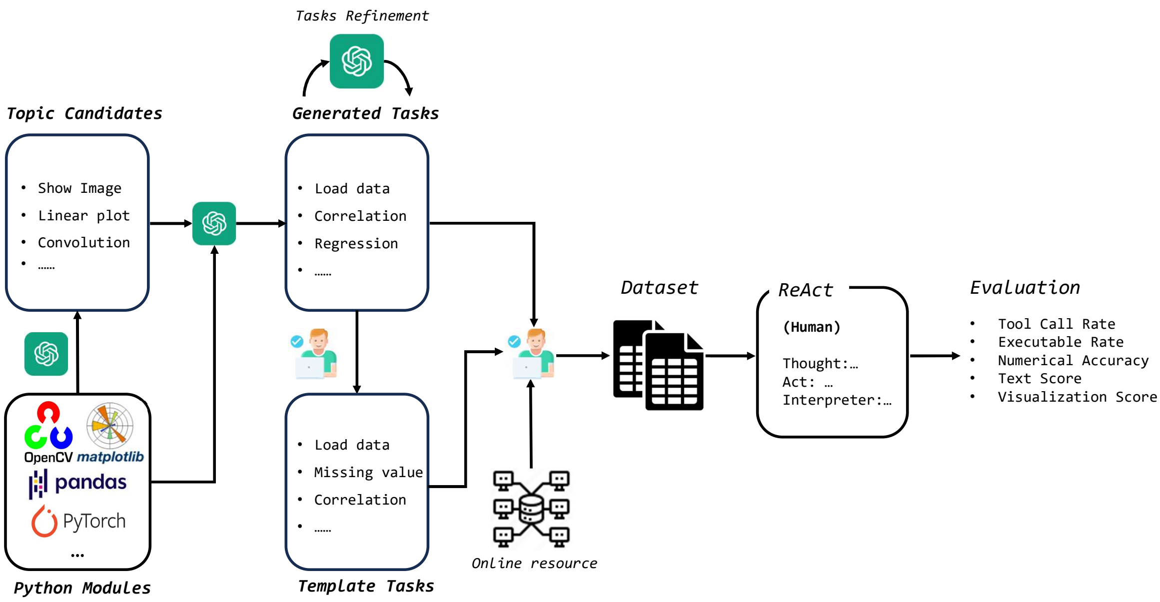


Figure 2: Overview of `CIBench`. `CIBench` first selects Python modules to generate candidate topics and then generates tasks based on these modules and the selected topic. Additionally, it summarizes the template tasks and filters out incorrect questions to enhance quality. Given the evaluation dataset and optional human assistance, we utilize the ReAct protocol to generate reasoning traces and invoke code interpreters alternately. Finally, the LLM's ability is assessed using two modes and five metrics, providing a holistic evaluation.

to solve tasks multiple times, enabling exploration of their self-debugging capabilities based on feedback from the code interpreter. Finally, we propose two evaluation modes: the end-to-end mode and the oracle mode, to comprehensively measure LLM's ability with and without human interaction.

In the following sections, we will detail the construction of the dataset in Sec.2.1 and the evaluation modes and metrics in Sec.2.2.

2.1 Evaluation Dataset

Python Modules Selection We carefully choose modules that pertain to the fields of data science. It encompasses a wide array of topics such as data cleansing, visualization, model development, natural language processing, image analysis, mathematical computations, and statistical methods. The modules we have selected are detailed in Tab.1.

Topic Candidates Generation After selecting modules, we employ GPT-4 to summarize 50 topics for each module, to encapsulate the vast majority of the module's functionalities, thereby offering precise guidance for the subsequent generation of more targeted questions. The specific prompts used in this process are elaborated in Appendix B.

Tasks Generation and Refinement We sample a module and topic, then prompt GPT-4 to generate questions and code based on the prompt in Fig.3. The prompt is designed to enable GPT-4 to generate a Jupyter notebook with sequential steps and various outputs, including numeric answers, structured output, and visualizations, mirroring real-world scenarios. Despite our request for concise descriptions, the generated content may lack conciseness and continuity. To address this, we undertake iterative refinement of these tasks. This process involves presenting both good and bad cases, along with additional modifications, to enhance the quality of questions and reduce ambiguity. Details of prompts used for refinement are in Appendix B.

Template Tasks Although the topics covered in the IPython are diverse, the datasets for each experiment are often similar and commonly used in machine learning tutorials (e.g., Titanic and Iris datasets). This could potentially diminish the validity of the benchmark as models may have already been extensively trained on these datasets. To address this concern, we introduce an additional step to enhance the variety of the benchmark.

Firstly, based on the generated tasks and existing high-quality notebooks and tutorials available in each Python library documentation, we summarize the template tasks, such as data loading, filtering missing values, and analyzing the correlation between data. With minor modifications, these template tasks can be applied to a range of datasets. Next, we diversify the benchmark by collecting new datasets through two approaches: 1) we prompt GPT-4 to generate datasets for differ-

Table 1: Selected Python modules and their categories.

Category	Python Modules
Data Cleaning and Manipulation	Pandas
Data Visualization	Matplotlib, Seaborn
Modeling	Scikit-learn, PyTorch, TensorFlow, LightGBM
Natural Language Processing	NLTK
Image Processing	OpenCV-Python
Mathematics and Statistics	SciPy

Question Generation

Prompt:
Please create jupyter notebook experiment based on Python module {}. Please follow these rules:
1. The experiment should be conducted in a jupyter notebook manner, but use the markdown format.
2. The experiment should only use Python code.
3. The experiment has around 10-15 continuous steps, from the easiest to the hardest.
4. The step description should be concise.
5. The step description should be precise and contain exact parameter names and values to instruct.
6. Each step requires Python code to solve and the executed result should be the numeric answer, structured output, or visualized result.
7. Please use 'matplotlib' to visualize if necessary.
8. DO NOT have any steps to save or write any output files.
9. Please provide an input data file with an external link.

The experiment topic is {}. You should generate the experiment file without any other statements.

Figure 3: An example prompt of task generation.

ent templates, and GPT-4 allows for the flexible specification of characteristics and data attributes; 2) we include the newest datasets from last year. These datasets offer authenticity and diversity, significantly reducing the likelihood that the model has previously encountered this data.

Quality Control Due to the inherent limitations of LLMs, it's challenging to guarantee perfect accuracy in question descriptions and result correctness. To enhance the quality of the benchmark, we employ several experts and utilize a code interpreter for manual double-checking. Our approach ensures that questions are written by a real user and various factors such as runtime are carefully controlled. For a comprehensive overview of the rules governing quality control, please refer to Appendix B. The statistics of the dataset are shown in Appendix A.2.

2.2 Evaluation Modes and Metrics

Evaluation Modes As shown in Fig.4, `CIBench` includes the end-to-end and oracle mode. It not only assesses the model's proficiency in autonomously resolving continuous problems but also assesses its capacity in conjunction with human interaction. In **end-to-end mode**, the model must solve the problem by itself. Each subsequent response is contingent upon the model's previous outcomes, necessitating self-correction based on code interpreter feedback. What's more, to reflect the real-world scenarios where human assistance is available, we introduce **oracle mode** to incorporate human-generated ground truth code as context, helping the model to address subsequent tasks. Conceptually, **oracle mode** emulates a few-shot testing scenario or in-context learning. It provides the model with comprehensive and accurate context to facilitate a more precise solution.

Evaluation Metrics In two evaluation modes, we introduce two types of evaluation metrics: process-oriented and output-oriented, to provide a comprehensive analysis of the model's performance. Process-oriented metrics focus on the correct invocation of tools and the successful compilation and execution of code. These metrics include the *Tool Call Rate*, which measures the proportion of instances where the model correctly follows the instructions to invoke a code interpreter, and the

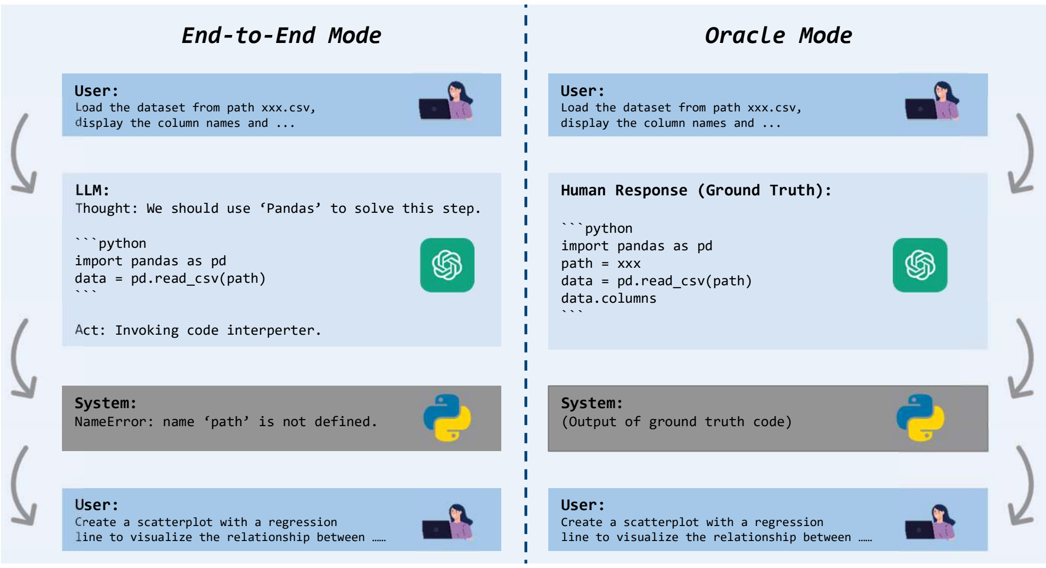


Figure 4: Evaluation modes: In end-to-end mode, the LLM addresses the user's question (bottom) within the context of its response, while in oracle mode, it answers the user's question (bottom) within the context of ground truth.

Executable Rate, which indicates the percentage of code that is executed without any errors.

On the other hand, output-oriented metrics focus on the outcomes of the model. These metrics include *Numeric Accuracy*, which assesses the accuracy of the numerical results; *Text Score*, which measures the quality of the structural text output using the Rouge metric (Lin, 2004); and *Visualization Score*, which evaluates the quality of visual output. Instead of using GPT-4V like Qwen-Agent, which is expensive and ineffective, we propose using structural similarities (Wang et al., 2004) between predictions and ground truth images as the visualization score. These metrics provide a holistic evaluation of the LLM's capabilities.

3 Experiments

3.1 Experiments Setup

To provide a thorough analysis, we evaluate 19 chat models, including popular open-sourced LLMs and the private GPT-4, using the `CIBench` benchmark. During inference, we allow LLMs to attempt up to 3 times. The specific versions of Python modules utilized in the code interpreter are provided in Appendix A.1. All experiments are conducted using NVIDIA A100 GPU within the OpenCompass (Contributors, 2023) evaluation platform.

3.2 Main Results

We roughly categorize the models into different groups based on their scales to facilitate better comparison. Overall, as depicted in Tab.2, MistralAI and InternLM have emerged as frontrunners in both the 7B and 13-20B categories. Following closely behind is Qwen. In the 70B group, DeepSeek-67B secures the top position with an overall score slightly lower than that of InternLM2-20B. The API model GPT-4 outperforms all other models, especially in end-to-end mode, highlighting the significant potential for improvement in current open-source models. What's more, larger models tend to exhibit superior performance across various metrics, in line with established trends (Brown et al., 2020; Kaplan et al., 2020; Wei et al., 2022). Moreover, models within the same series (such as Mistral, InternLM, Qwen, etc.) consistently maintain relatively stable rankings within their respective parameter groups, underscoring the stability and efficacy of our approach.

When comparing the end-to-end mode and oracle mode, it becomes evident that the oracle mode surpasses the end-to-end mode across all metrics. This observation suggests that LLMs can achieve better results with human interaction, hinting at a promising avenue for integrating LLMs to assist humans in data science. The experiment demos are shown in Appendix C.

3.3 More Analysis

Reasoning Analysis Focusing on numerical results and structural text output, `CIBench` provides an effective means to evaluate the model's reasoning ability. To validate this, we average numeric accuracy and text score in two evaluation modes, then conduct a correlation analysis between this reasoning metric and existing reasoning benchmarks, such as HumanEval (Chen et al., 2021) and GSM8k (Cobbe et al., 2021). As illustrated in Fig.5,

Table 2: **Main results of `CIBench`.** Tool, Exe, Num, Text, and Vis denote the tool call rate, executable rate, numeric accuracy, text score, and visualization score respectively. **bold** denotes the best score among the same model scale. *Average is the mean of Num, Text, and Vis in two modes..*

Model	End-to-end Mode					Oracle Mode					Average
	Tool	Exe	Num	Text	Vis	Tool	Exe	Num	Text	Vis	
LLaMA2-7B	37.8	17.9	3.3	7.3	4.3	90.6	50.7	24.4	31.5	37.5	18.1
Yi-6B	81.8	52.4	21.2	26.1	33.2	97.6	63.1	28.0	39.3	46.4	32.4
ChatGLM3-6B	67.5	45.5	19.3	21.1	24.1	98.5	59.9	31.8	33.1	37.3	27.8
DeepSeek-7B	80.2	53.8	16.9	21.9	35.6	94.8	68.2	41.0	64.7	47.4	37.9
Vicuna-7B	89.7	53.8	15.8	28.3	28.9	100.0	65.2	29.8	61.0	43.4	34.5
Qwen-7B	99.3	**73.4**	30.1	**57.3**	40.9	99.8	76.8	49.1	66.3	53.2	49.5
Mistral-7B	**99.5**	72.3	35.1	57.2	44.4	99.9	77.6	48.8	58.1	51.5	49.2
InternLM2-7B	99.3	70.0	**44.6**	44.3	**47.4**	**100.0**	**79.0**	**53.8**	**81.1**	**54.9**	**54.3**
LLaMA2-13B	95.9	67.5	12.7	22.4	21.2	99.3	70.2	35.8	34.3	36.7	27.2
Baichuan2-13B	64.4	48.9	5.0	3.7	19.3	97.0	63.9	39.3	70.6	44.5	30.4
Vicuna-13B	90.3	65.1	31.5	53.3	38.3	100.0	67.7	40.8	66.7	41.8	45.4
Qwen-14B	92.1	75.1	45.1	59.4	48.4	**100.0**	85.4	53.3	74.0	60.3	56.8
Mistral-8x7B	**99.8**	**89.0**	38.1	69.1	**51.9**	99.9	**90.5**	59.0	**87.9**	**64.7**	61.8
InternLM2-20B	98.9	87.3	**56.3**	**72.6**	51.3	99.3	86.3	**65.6**	87.0	63.0	**66.0**
Yi-34B	88.5	64.7	38.7	40.2	44.4	100.0	77.0	53.3	71.0	51.2	49.8
LLaMA2-70B	99.2	66.9	21.4	34.6	30.1	99.7	70.2	36.9	43.2	42.9	34.9
DeepSeek-67B	**99.8**	**89.4**	46.6	**68.8**	**60.4**	**100.0**	91.0	64.3	75.2	**68.5**	**64.0**
Qwen-72B	99.0	88.8	**52.0**	67.7	56.9	99.7	**92.3**	**66.7**	**76.5**	59.2	63.1
GPT-4-0613	99.7	98.4	66.8	75.7	65.5	99.8	97.8	70.7	84.2	73.6	72.8

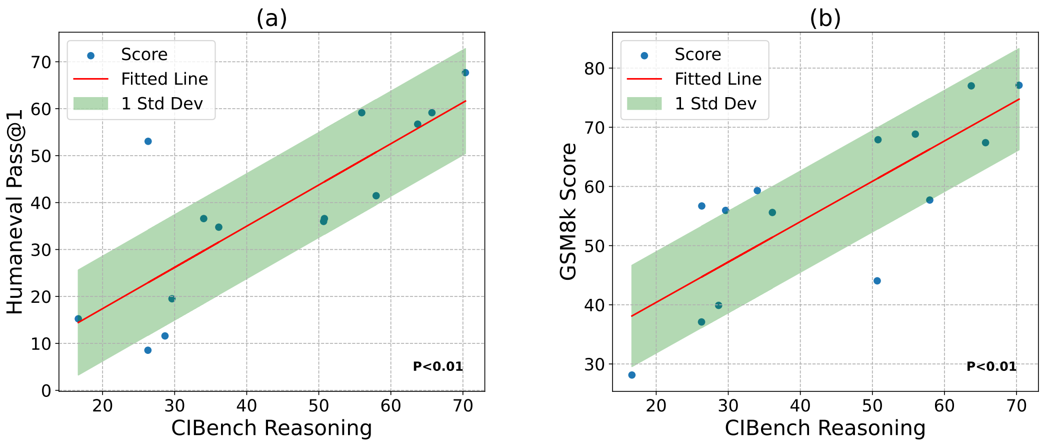


Figure 5: `CIBench` Reasoning v.s. Humaneval Pass@1 and Gsm8K.

a significant correlation ($p < 0.01$) is revealed. This indicates that `CIBench` can serve as an alternative reasoning benchmark, providing a comprehensive, multi-dimensional evaluation that overcomes the limitations of current benchmarks.

Visualization Metric Analysis To validate the effectiveness of our proposed visualization metric, we follow QwenAgent (Bai et al., 2023) and use GPT-4V to assess visualization scores on a subset of `CIBench` tasks. The prompt is provided in Appendix D. As shown in Fig.7, despite structural similarities being derived from low-level features, there is a strong correlation between them and GPT-4V scores, demonstrating remarkable consistency between the two metrics. Therefore, we can utilize structural similarities as a simplified visualization metric to subject GPT-4V for effective analysis.

Debug Ability Analysis In the ReAct protocol, we allow LLMs to try to solve tasks multiple times.

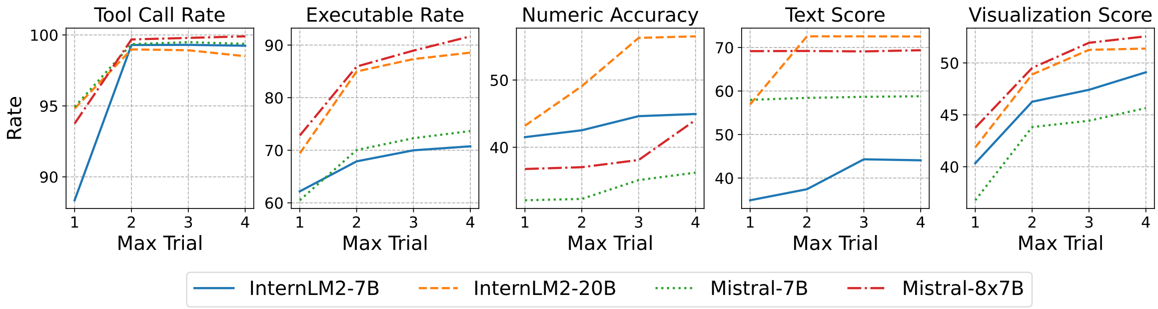


Figure 6: Ablation of trial times in ReAct protocol.

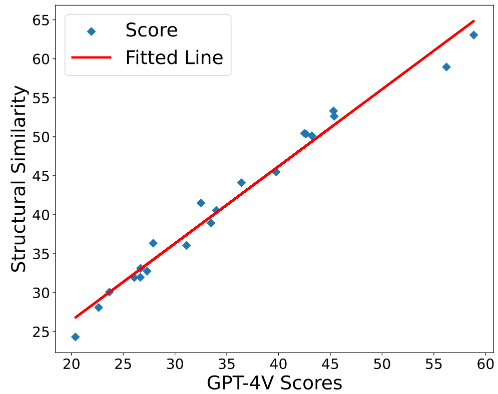


Figure 7: Structural Similarities v.s. GPT-4V Scores.

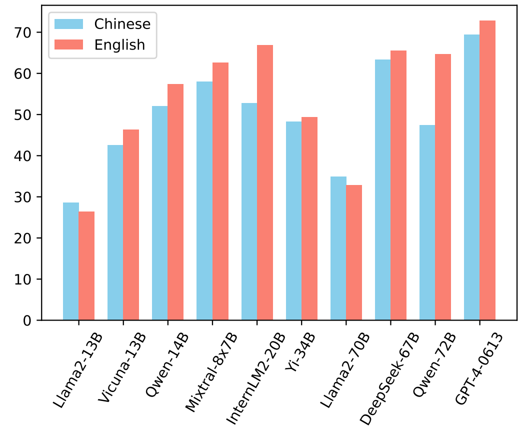


Figure 8: Chinese `CIBench` v.s. English `CIBench` .

During each trial, the model can use feedback from the code interpreter to rectify any bugs in the generated code. To assess the LLMs' ability to autonomously correct bugs, we vary the number of trials. As shown in Fig.6, increasing the number of trials correlates with improvements across all metrics. Significantly, there is a notable enhancement when the number of trials reaches two, particularly evident in metrics such as tool rate and executable rate. The slight decrease in InternLM2-20B's Tool Call Rate may be attributed to random failures of the code interpreter. This suggests that the LLM can autonomously rectify bugs to a certain extent. In our experiments, to balance evaluation time and performance, we set the number of trials to three.

Different Category Modules Analysis We analyze the abilities of various LLMs with different category modules (Tab.1). As depicted in Fig. 1, LLMs excel in solving mathematics and statistics tasks using SciPy modules but struggle with complex modeling tasks that require advanced coding and reasoning abilities. We hope that future LLMs will excel in modeling tasks.

Error Mode Analysis In the evaluation of `CIBench`, we identify four prevalent types of errors in the code generated by the model. These errors are categorized as follows: 1) **Instruction Following Errors**: These encompass instances where the model deviates from or disregards provided instructions, reflecting a lack of adherence to specified guidelines; 2) **Hallucination Errors**: This category pertains to the phenomenon of the model generating code that contains hallucinated elements, such as utilizing undefined parameters or referencing irrelevant variables; 3) **Reasoning Errors**: These errors occur when the model encounters complex problems, often resulting in logical errors in the generated code. Such errors offer valuable insights into the model's ability to handle intricate tasks in code generation; 4) **Code Errors**: Basic errors in code generation fall under this category. While these errors may sometimes appear trivial, they signify potential deficiencies in the model's code-generation process. These identified errors effectively underscore the current limitations of LLMs in terms of their coding capabilities, providing valuable insights for the ongoing development of `CIBench`. Detailed examples of these errors are presented in Appendix E.

Cross Language Analysis To benchmark the LLMs' ability in Chinese, we created a Chinese version of `CIBench` by translating the template tasks into Chinese. This allows us to evaluate the Code Interpreter performance in Chinese. As shown in Fig.8, we observe that: 1) most models exhibit a slight decrease in Chinese CIBench compared to their English counterparts.; 2) the strong InternLM2-20B and Qwen-72B drop a lot on Chinese CIBench, compared to the English version. Further research and development efforts are necessary to address these discrepancies and improve the performance of LLMs in multilingual scenarios.

4 Related Works

CIBench is an evaluation framework that assesses LLMs' (Touvron et al., 2023; Bai et al., 2023; DeepSeek-AI, 2024; Chiang et al., 2023) ability to utilize external code interpreters for solving data science tasks. Therefore, we focus on presenting work related to invoking code interpreters and benchmarks related to data science.

4.1 Model with Plugins

LLM-based agents use external tools via APIs to solve complex tasks and have been regarded as a promising direction (Chase, 2022; Qin et al., 2023; Significant Gravitas; Schick et al., 2023; Hong et al., 2023; Wu et al., 2023). Specifically, (Li et al., 2023b; Hong et al., 2023; Qian et al., 2023) develop efficient workflows to coordinate multi-agent systems for automatic programming. (Schick et al., 2023; Chase, 2022; Qin et al., 2023; Gao et al., 2023) equip LLMs with external tools, such as search engines, calculators, and code interpreters, to augment LLMs' problem-solving ability. Among these tools, the code interpreter can promote LLMs' reasoning and coding ability and has gradually gained attention in works like (Ying et al., 2024; Zhou et al., 2023; Zhuang et al., 2023).

In line with the above works, we aim to develop an evaluation framework to benchmark LLMs' ability with code interpreters for solving data science tasks, providing insights for future work to promote LLMs for better realistic utilization.

4.2 Related Benchmarks

Various benchmarks (Cobbe et al., 2021; Chen et al., 2021; Qin et al., 2023; Bai et al., 2023) have been proposed to measure LLMs' reasoning, coding, and tool utilization ability. Classic benchmarks, such as GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021), focus on evaluating the mathematical reasoning or code capability of LLMs. ToolBench (Qin et al., 2023) and MS-Agent Bench (Li et al., 2023a) aim to evaluate LLMs' capability in effectively utilizing various tools and generating accurate and contextually appropriate responses. However, the above benchmarks cannot measure LLMs' ability in data science tasks, which require instruction following, coding, and tool utilization abilities. To address this gap, QwenAgent (Bai et al., 2023) introduces a benchmark for data science, focusing mainly on mathematical problems and data visualization. Meanwhile, (Hu et al., 2024) introduces DABench, which evaluates various concepts with individual questions assigned to each dataset. However, this approach does not effectively mimic practical scenarios where interconnected questions are raised.

In contrast to (Bai et al., 2023; Hu et al., 2024), CIBench simulates a real-world data science workflow by leveraging sequential interactive IPython sessions and covers most concepts in data science by including commonly used Python modules. Furthermore, we devise two evaluation modes and five metrics to holistically evaluate LLMs' abilities.

5 Limitation

Our work has three main limitations: 1) CIBench is currently limited to Python, despite it could be extended to include other programming languages using a similar methodology; 2) the evaluation metric of CIBench has limitations in measuring certain data science tasks, such as "training a model with PyTorch" and tasks involving randomness; 3) the error mode analysis lacks quantitative results, which may require manual verification, hindering a better understanding the most severe errors for LLMs.

6 Conclusion

We propose a novel benchmark, named CIBench , to comprehensively assess LLMs' ability to leverage code interpreters for complex data science tasks. It includes an evaluation dataset covering widely used Python modules in data science and two evaluation modes measuring LLMs' ability with and without human assistance. The evaluation dataset is constructed using an LLM-human cooperative approach, leveraging interactive IPython sessions to simulate realistic scenarios in data science. Thorough experimental analysis with 19 LLMs on CIBench indicates that LLMs perform poorly in modeling category modules, with the best open-sourced LLM lagging behind GPT-4 by 6.8%. We hope that our detailed analysis provides valuable insights for future work to enhance LLMs' ability in data science tasks.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Harrison Chase. 2022. LangChain.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *https://vicuna.lmsys.org*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. `https://github.com/open-compass/opencompass`.

DeepSeek-AI. 2024. Deepseek llm: Scaling open-source language models with longtermism.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Chenliang Li, Hehong Chen, Ming Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, Hongzhu Shi, Ji Zhang, Fei Huang, and Jingren Zhou. 2023a. Modelscope-agent: Building your customizable agent system with open-source large language models.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023b. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Significant Gravitas. AutoGPT.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, et al. 2024. Internlm-math: Open math large language models toward verifiable reasoning. *arXiv preprint arXiv:2402.06332*.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, et al. 2023. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *arXiv preprint arXiv:2306.13304*.

A Dataset Details

A.1 Module Version Settings

The version of Python modules used in code interpreters is listed in Tab.3.

Table 3: The module version settings in `CIBench`.

Module	Version
Pandas	1.5.3
Matplotlib	3.7.2
Seaborn	0.13.0
Scikit-learn	1.2.1
PyTorch	1.13.1
TensorFlow	2.14.0
LightGBM	4.1.0
NLTK	3.8
PyTorch	1.131
OpenCV-Python	4.8.1.78
SciPy	1.11.2

A.2 Dataset Statistics

The `CIBench` comprises generation tasks, template tasks, and Chinese template tasks, which produce three types of output: numerical, text, and visualization. The statistics of `CIBench` are shown in Tab.4.

Table 4: Dataset statistics of `CIBench`. "generation", "template", and "template_cn" represent generation tasks, template tasks, and Chinese template tasks, respectively. **Other** refers to the questions that only require successful execution without any output comparison.

Subset	Num	Text	Vis	Other	Total
generation	210	76	466	208	960
template	147	20	161	142	470
template_cn	147	20	161	142	470
total	504	116	788	492	1900

B Construction Prompts and Rules

Topic Generation The prompt used for topic generation is shown in Fig. 11.

Question Refinement The prompts used for question refinement are shown in Fig. 12, 13.

Quality Control Rules We include manual quality checking in the final steps due to limitations in the LLMs' ability to control runtime or file size. The rules used for quality control are shown in Fig.14.

C Experiment Example Demo

An example of the experiment is shown in Fig. 9 and Fig. 10.

D Subjective Visualization Evaluation

The prompt used for subjective visualization scoring is shown in Fig.15.

E Dataset Error Analysis

In the evaluation of `CIBench`, we identify four prevalent types of errors in the code generated by the model. These errors are categorized as follows: 1) **Instruction Following Errors** (Fig.16): These encompass instances where the model deviates from or disregards provided instructions, reflecting a lack of adherence to specified guidelines; 2) **Hallucination Errors** (Fig.17): This category pertains to the phenomenon of the model generating code that contains hallucinated elements, such as utilizing undefined parameters or referencing irrelevant variables; 3) **Reasoning Errors** (Fig.18): These errors occur when the model encounters complex problems, often resulting in logical errors in the generated code. Such errors offer valuable insights into the model's ability to handle intricate tasks in code generation; 4) **Code Errors** (Fig.19): Basic errors in code generation fall under this category. While these errors may sometimes appear trivial, they signify potential deficiencies in the model's code-generation process. These identified errors effectively underscore the current limitations of LLMs in terms of their Code Interpreter capabilities, providing valuable insights for the ongoing development of `CIBench`.

F Human Annotator

In our paper, the authors also serve as annotators for CIBench. These annotators are tasked with creating template tasks for various data science challenges, ensuring coverage of a wide range of concepts in the field. Importantly, our data collection process for these tasks is unbiased and does not involve any private information.

G Ethical Consideration

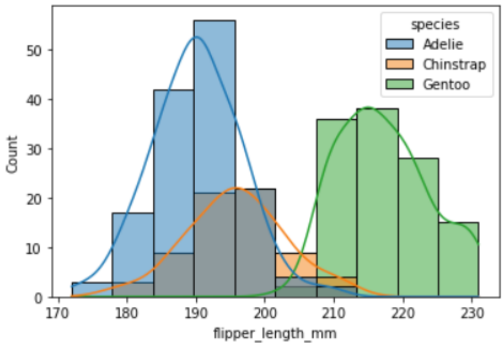
We use GPT-4 and online resources to construct our benchmark. The benchmarks are carefully processed by experts to exclude any private information. Additionally, we utilize ChatGPT for text refinement and to correct any typographical errors during the writing process.

Load the Penguins dataset provided by Seaborn. Drop the rows containing missing data from the dataset.

[1]:
```
import seaborn as sns

# Load the Penguins dataset
penguins = sns.load_dataset('penguins')

# Drop rows with missing data
penguins_cleaned = penguins.dropna()
```

Plot a univariate distribution of observations for the 'flipper_length_mm' column. Add color to the plot based on the 'species' column to differentiate between the species.

[2]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create the plot with colors based on the 'species' column
sns.histplot(data=penguins_cleaned, x='flipper_length_mm', hue='species', kde=True)

# Show the plot
plt.show()
```



Fill the area under the lines of the plot with color for better visualization. Calculate the mean of the 'flipper_length_mm' column and add a vertical line to the plot at this mean value.

[3]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create the plot with colors based on the 'species' column
# Fill the area under KDE plot
sns.histplot(data=penguins_cleaned, x='flipper_length_mm', hue='species', kde=True, element='poly')

# Calculate the mean of the 'flipper_length_mm' column
mean_flipper_length = penguins_cleaned['flipper_length_mm'].mean()

# Add a vertical line to the plot at the mean value
plt.axvline(mean_flipper_length, color='red', linestyle='--')

# Show the plot
plt.show()
```

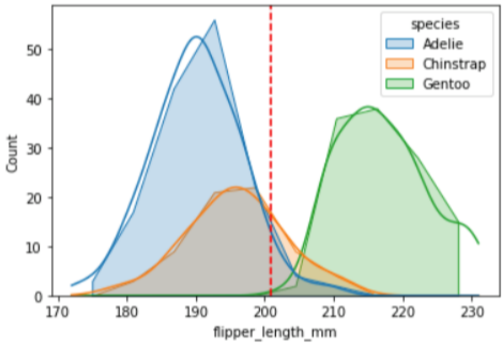
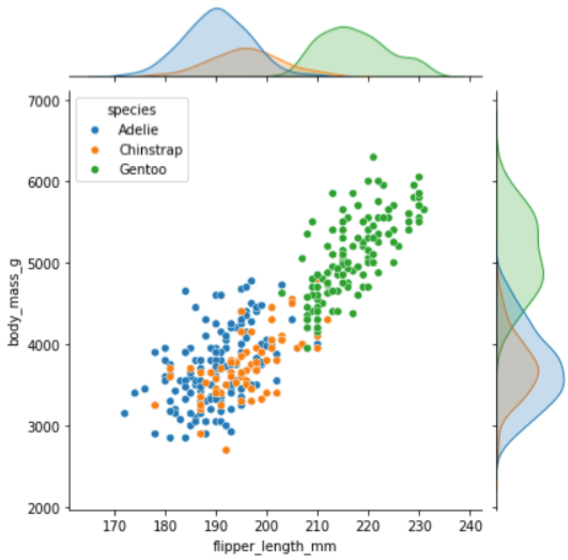


Figure 9: Example of successful experiment.

Plot a bivariate distribution of 'flipper_length_mm' and 'body_mass_g' columns.Color the bivariate plot based on the 'species' column to differentiate between the species.

[4]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Create a bivariate plot of 'flipper_length_mm' and 'body_mass_g' colored by 'species'
sns.jointplot(data=penguins_cleaned, x='flipper_length_mm', y='body_mass_g', hue='species')

# Show the plot
plt.show()
```



Fill the area under the lines of the bivariate plot with color for better visualization. Add a legend to the plot using 'species' as the title.

[5]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Create a bivariate KDE plot of 'flipper_length_mm' and 'body_mass_g' colored by 'species'
g = sns.jointplot(data=penguins_cleaned, x='flipper_length_mm', y='body_mass_g', hue='species', kind='kde')

# Add a legend with the title 'species'
plt.legend(title='species')

# Show the plot
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ig
nored when legend() is called with no argument.
```

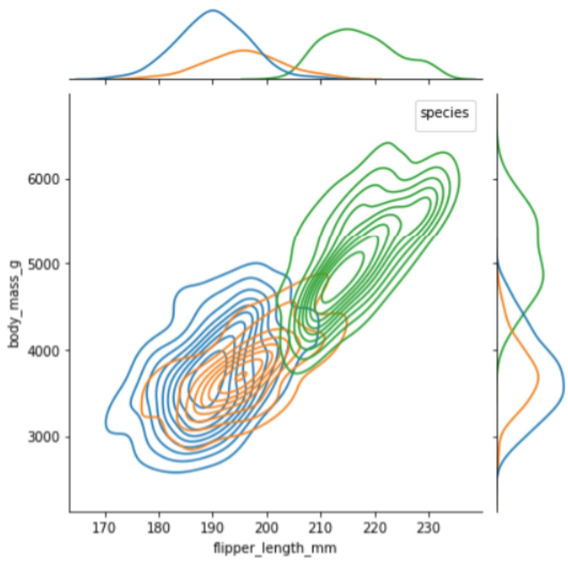


Figure 10: Example of successful experiment. Continue of Fig. 9.

Topic Generation

Prompt:
Please create 50 different topics base on Python module {}. These topics can cover the majority of functionality of this module.

Figure 11: An example prompt of topic generation.

Question Refinement

Prompt:
Based on the experiment document and each step, please refine these steps into specific instructions for code writing to solve a real world question.
Provide parameter settings if needed, ensuring that the execution results are identical for any reasonable person attempting to write the code.
Please refrain from directly mentioning the name of the API, just target on purpose and give parameter settings.
Please also summarize the required external dataset if applicable.

Bad case step:

1. "Remove the 'Cabin' column from the dataframe due to its numerous missing values. Make sure to specify that you are removing a column and that the change should be permanent."
Too wordy, "Remove the 'Cabin' column and make the change permanent." should be fine.

2. "Identify and count how many values are missing in each column of the dataframe. Use a chain of two functions: the first checks if each value is null, and the second adds up these null instances for each column."
Too specific, "Count how many values are missing in each column of the dataframe. Print the count." should be fine.

Good case step:

1. "Convert the 'Sex' column to numerical values. Map 'male' to 0 and 'female' to 1." Concise, clear instruction.

Your output format should be as follows starting with import necessary libraries:
[*Optional*] Dataset Link:
[*Optional*] Dataset Description:

Step 1. xx
code blcok
Step 2. xx
code blcok
...

Figure 12: An example prompt of question refinement.

Another Question Refinement

Prompt:

Given the above experiment document. Do the following modification:
1. Remove all the installation steps.
2. Remove all the file saving steps.
3. Elaborate the steps to be more specific with number and inputs that the execution results are identical for any reasonable person attempting to solve this step.
4. Reorder the steps.

Response with the new experiment document.

Figure 13: Another example prompt of question refinement.

Quality Control Rules

Check Rules:

- Assure the questions is written from the perspective of a real user.
- Assure file path informed in the head for all the experiment required external files.
- Control the runtime, and each step should ideally produce results within 1 minute.
- Control the file size, the file used for single experiment should ideally not exceed 50M.
- Assure the output is valid and unambiguous as ground truth.

Figure 14: Rules used for manual dataset quality control.

Subjective Visualization Scoring Prompt

Prompt:

You are an assistant skilled in assessing visualization capabilities.

In the capacity of a fair judge, you will evaluate the quality of images drawn by an AI model generating code for visualization-related problems. We will provide you with a code visualization problem and an image drawn by the code created by the AI model you need to assess. When you start your assessment, you must adhere to the following process:

1. Rate the visualization capability with a score between 1 and 10 for the image, returning only the number without any additional descriptions.

2. Your scoring needs to be as rigorous as possible, and it should follow the scoring rules below: Overall, the higher the quality of the model's response, the higher the score.

A score of 1 to 2 is given when the image cannot reflect the given instruction content at all.

A score of 3 to 4 is given when the image can partly reflect the given instruction content, but there is a significant lack of specific detail expression.

If the image basically meets the given instructions, but the aesthetic quality of the image is average without any outstanding features, this kind of rating can get a score of 5 to 6.

When the image matches the given instructions well, and shows some aesthetic appeal, such as some originality in color matching and shape design, this kind of rating can get a score of 7 to 8.

Only when the image completely matches the given instructions, covers all the details in the instructions, and performs excellently in terms of aesthetics, can this kind of rating get a score of 9 to 10.

[Visualization Problem]:{question}

Figure 15: An example prompt of subjective visualization scoring prompt.

Following error

User:

Detect corners using the Shi-Tomas corner detector with max-corners=50, min-distance=0.5 and blocksize=10, mark the corners with circles on the image. The radius and thickness of the circle are 5 and 1. Show the marked image.Please use opencv and numpy modules.

Assistant:

```
# Detect corners using Shi-Tomasi corner detector
corners = cv2.goodFeaturesToTrack(equalized_image,
                                  maxCorners=max_corners,
                                  qualityLevel=quality_level,
                                  minDistance=min_distance,
                                  blockSize=block_size)
# Mark the corners with circles on the image
marked_image = equalized_image.copy()
for corner in corners:
    x, y = corner.ravel()
    cv2.circle(marked_image, (int(x), int(y)), 5, (255, 0, 0), -1)
# Draw a blue filled circle at each corner

# Show the marked image
plt.imshow(marked_image, cmap='gray')
plt.axis('off')  # Turn off axis numbers and ticks
plt.show()
```

System:

```
NameError                                 Traceback (most recent call last)
Cell In[6], line 2
      1 # Detect corners using Shi-Tomasi corner detector
----> 2 corners = cv2.goodFeaturesToTrack(equalized_image, maxCorners=max_corners,
qualityLevel=quality_level, minDistance=min_distance, blockSize=block_size)
      4 # Mark the corners with circles on the image
      5 marked_image = equalized_image.copy()

NameError: name 'max_corners' is not defined
```

User: Define the CrossEntropyLoss function and the Adam optimizer. Train the model for 3 epochs. Display the loss for the last epoch(Keep to two decimal places).

Assistant:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 3
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), labels.float())
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    final_loss = round(running_loss/len(train_loader), 2)
    print(final_loss)
```

System:

```
933.55
933.45
933.09
```

Figure 16: Examples of Following error

Hallucination error

User:

Build a vocabulary of all the tokens and assign an index to the vocabulary. Display the index of the word "happy". Please use nltk modules.

Assistant:

```
# Check the vocabulary
vocab
# Display the index of the word "happy"
word_to_index.get('happy', 'Word not found in vocabulary')
```

System:

```
NameError                                 Traceback (most recent call last)
Cell In[2], line 2
      1 # Check the vocabulary
----> 2 vocab
      3 # Display the index of the word "happy"
      4 word_to_index.get('happy', 'Word not found in vocabulary')

NameError: name 'vocab' is not defined
```

User:

Define a LightGBM model with max_depth=4, n_estimators=120, learning_rate=0.01 and num_leaves=31. Train the model with Evaluation Metric='logloss'. Please use LightGBM modules.

Assistant:

```
import lightgbm as lgb

# Define the LightGBM model
model = lgb.LGBMClassifier(max_depth=4, n_estimators=120, learning_rate=0.01, num_leaves=31)

# Train the model with evaluation metric as 'logloss'
model.fit(X_train, y_train, eval_set=(X_test, y_test), eval_metric='logloss', early_stopping_rounds=20)
```

System:

```
TypeError                                 Traceback (most recent call last)
Cell In[3], line 7
      4 model = lgb.LGBMClassifier(max_depth=4, n_estimators=120, learning_rate=0.01,
      num_leaves=31)
      6 # Train the model with evaluation metric as 'logloss'
----> 7 model.fit(X_train, y_train, eval_set=(X_test, y_test), eval_metric='logloss',
early_stopping_rounds=20)

TypeError: LGBMClassifier.fit() got an unexpected keyword argument 'early_stopping_rounds'
```

Figure 17: Examples of Hallucination error

Reasoning error

User:

Split the dataset into training and testing sets using 0.2 as the test size, then define the train_loader and test_loader. Set batch size as 64. Please use pytorch and sklearn modules.

Assistant:

```
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import DataLoader, TensorDataset

# Split the dataset into training and testing sets
X = df['text'].values
y = df['sentiment'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the train_loader and test_loader
def create_data_loader(X, y, batch_size):
    X_tensor = torch.tensor(X)
    y_tensor = torch.tensor(y, dtype=torch.long)
    dataset = TensorDataset(X_tensor, y_tensor)
    data_loader = DataLoader(dataset, batch_size=batch_size)
    return data_loader

batch_size = 64
train_loader = create_data_loader(X_train, y_train, batch_size)
test_loader = create_data_loader(X_test, y_test, batch_size)
```

System:

```
TypeError                                 Traceback (most recent call last)
Cell In[4], line 19
     16     return data_loader
     18 batch_size = 64
---> 19 train_loader = create_data_loader(X_train, y_train, batch_size)
     20 test_loader = create_data_loader(X_test, y_test, batch_size)

Cell In[4], line 12
     11 def create_data_loader(X, y, batch_size):
---> 12     X_tensor = torch.tensor(X)
     13     y_tensor = torch.tensor(y, dtype=torch.long)
     14     dataset = TensorDataset(X_tensor, y_tensor)

TypeError: can't convert np.ndarray of type numpy.object_. The only supported types are: float64,
float32, float16, complex64, complex128, int64, int32, int16, int8, uint8, and bool.
```

User:

Group and aggregate data by weight and calculate the average of each numerical group. Find out how many attributes have a negative correlation with weight.Please use pandas modules.

Assistant:

```
# Group and aggregate data by weight and calculate the average of each numerical group
grouped_data = df.groupby('Weight').mean()

# Find attributes with negative correlation with weight
negative_correlation_count = (grouped_data.corr()['Weight'] < 0).sum()
negative_correlation_count
```

System:

```
KeyError                                  Traceback (most recent call last)
File anaconda3/envs/prod-torch1.13/lib/python3.10/site-packages/pandas/core/indexes/base.py:3802,
in Index.get_loc(self, key, method, tolerance)
   3801 try:
-> 3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:

File anaconda3/envs/prod-torch1.13/lib/python3.10/site-packages/pandas/_libs/index.pyx:138, in
pandas._libs.index.IndexEngine.get_loc()

File anaconda3/envs/prod-torch1.13/lib/python3.10/site-packages/pandas/_libs/index.pyx:165, in
pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:5745, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:5753, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Weight'
```

Figure 18: Examples of Reasoning error

Code error

User:

Generate a pair plot for all the numerical columns, using color = "pastel".Please use seaborn modules.

Assistant:

```
sns.pairplot(df, kind='scatter', plot_kws={'color':'pastel'})
plt.show()
```

System:

```
ValueError                                Traceback (most recent call last)
File anaconda3/envs/prod-torch1.13/lib/python3.10/site-packages/matplotlib/axes/_axes.py:4372, in
Axes._parse_scatter_color_args(c, edgecolors, kwargs, xsize, get_next_color_func)
   4371 try:
-> 4372     mcolors.to_rgba_array(kwcolor)
   4373 except ValueError as err:

File anaconda3/envs/prod-torch1.13/lib/python3.10/site-packages/matplotlib/colors.py:471, in
to_rgba_array(c, alpha)
    470 if isinstance(c, str):
--> 471     raise ValueError(f"{c!r} is not a valid color value.")
    473 if len(c) == 0:

ValueError: 'pastel' is not a valid color value.
```

User:

Blur the initial image by applying a Gaussian filter with a sigma value of 3. Sigma is the standard deviation for the Gaussian kernel, which controls the amount of blur. Then display the smoothed image.

Assistant:

```
smooth = ndimage.gaussian_filter(image, sigma=3)
plt.imshow(smooth)
plt.show()
```

System:

```
NameError                                 Traceback (most recent call last)
Cell In[48], line 2
      1 smooth = ndimage.gaussian_filter(image, sigma=3)
----> 2 plt.imshow(smooth)
      3 plt.show()

NameError: name 'plt' is not defined
```

Figure 19: Examples of Code error