InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks

Xueyu Hu [1] **Ziyu Zhao** [1] **Shuang Wei** [2] **Ziwei Chai** [1] **Qianli Ma** [3]
Guoyin Wang [3] **Xuwu Wang** [3] **Jing Su** [3] **Jingjing Xu** [3] **Ming Zhu** [3] **Yao Cheng** [3] **Jianbo Yuan** [3]
Jiwei Li [1] **Kun Kuang** [1] **Yang Yang** [1] **Hongxia Yang** [3] **Fei Wu** [1 4 5]

Abstract

In this paper, we introduce **InfiAgent-DABench**, the first benchmark specifically designed to evaluate LLM-based agents on data analysis tasks. These tasks require agents to end-to-end solving complex tasks by interacting with an execution environment. This benchmark contains DAEval, a dataset consisting of 603 data analysis questions derived from 124 CSV files, and an agent framework which incorporates LLMs to serve as data analysis agents for both serving and evaluation. Since data analysis questions are often open-ended and hard to evaluate without human supervision, we adopt a format-prompting technique to convert each question into a closed-form format so that they can be automatically evaluated. Our extensive benchmarking of 34 LLMs uncovers the current challenges encountered in data analysis tasks. In addition, building on top of our agent framework, we develop a specialized agent, DAAgent, which surpasses GPT-3.5 by 3.9% on DABench. Evaluation datasets and toolkits for InfiAgent-DABench are released at `https://github.com/InfiAgent/InfiAgent`.

1. Introduction

Large language model-based agents (LLM-based agents) has been one of the most popular concepts recently in AI society (Liu et al., 2023a; Wang et al., 2023c; Zhao et al., 2023; Xi et al., 2023), with the belief that these agents are a prototype of Artificial General Intelligence (AGI). These agents can reason (Huang & Chang, 2023; Wei et al., 2022b; Kojima et al., 2022), plan (Xu et al., 2023b; Sun et al., 2023; Yao et al., 2022b), memorize (Liang et al., 2023; Zhang et al., 2023a) and even use tools (Qin et al., 2023b;a), by leveraging the ability of large language models (LLMs). Nowadays, a large number of LLM-based agents have been developed, *e.g.*, AutoGPT (Torantulino, 2023), BabyAGI (Nakajima, 2023) and AgentGPT (Reworkd, 2023).

Specifically, data analysis tasks have been found to be challenging but practically useful problems for LLM-based agents (Lai et al., 2023; Xie et al., 2023; Bai et al., 2023). Data analysis is a systematic process of examining, cleaning, transforming, and modeling data to discover useful information, inform conclusions, and support decision-making (Hardy & Bryman, 2004). It is applied across diverse domains including business intelligence (Covington, 2016), healthcare (Reddy & Aggarwal, 2015), finance (Koop, 2022), and scientific research (Sabina & Zalta, 2020). A typical data analysis problem requires the solver to interact with data files and the code executor to obtain a solution, which requires the harmonization ability of LLMs in natural language and code, and is suitable for LLM-based agents to solve (Yang et al., 2024). Figure 1 shows how a LLM-based agent solve a problem with a CSV file as input, following the ReAct (Yao et al., 2022b) approach to plan, write code, call external Python sandbox to execute and concluding the output from the sandbox to solve the problem. OpenAI's Advanced Data Analysis[1] (ADA) is the first commercial LLM-based agent that effectively helps users analyze data files with the execution of programming code. On top of OpenAI ADA, many developers have also formulated open-source agents for data analysis, including Open Interpreter (Lucas, 2023), Open Agents (Xie et al., 2023), Qwen-Agent (Bai et al., 2023) and Taskweaver (Qiao et al., 2023).

However, a comprehensive benchmark for evaluating agents on data analysis is still missing in the community, which motivates us to build InfiAgent-DABench[2], to our knowl-

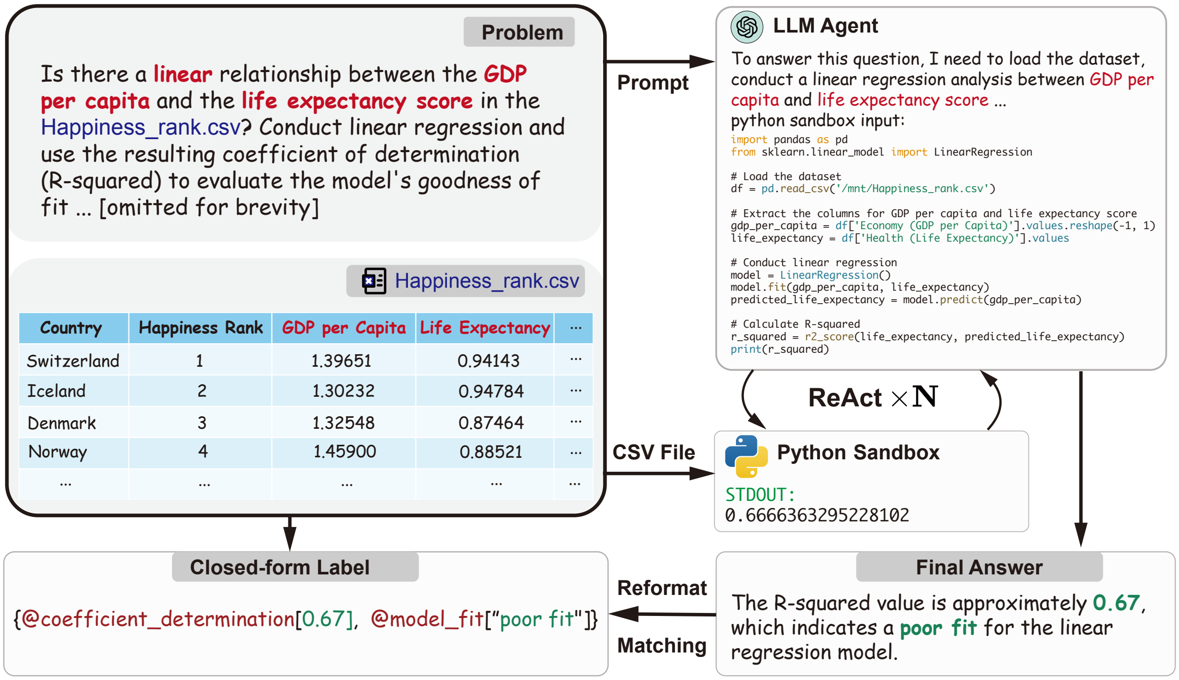
---

[1]Department of Computer Science and Technology, Zhejiang University, Hangzhou, China [2]Rochester Institute of Technology [3]ByteDance Inc. [4]Shanghai Institute for Advanced Study, Zhejiang University, Shanghai, China [5]Shanghai AI Laboratory, Shanghai, China. Correspondence to: Kun Kuang <kunkuang@zju.edu.cn>, Fei Wu <wufei@zju.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1] `https://openai.com/blog/chatgpt-plugins#code-interpreter`

[2] The series of works is named as InfiAgent, which is included as part of the comprehensive name.



Is there a linear relationship between the GDP per capita and the life expectancy score in the Happiness_rank.csv? Conduct linear regression and use the resulting coefficient of determination (R-squared) to evaluate the model's goodness of fit ... [omitted for brevity]

Country	Happiness Rank	GDP per Capita	Life Expectancy	...
Switzerland	1	1.39651	0.94143	...
Iceland	2	1.30232	0.94784	...
Denmark	3	1.32548	0.87464	...
Norway	4	1.45900	0.88521	...
...	...	...	...	...

To answer this question, I need to load the dataset, conduct a linear regression analysis between GDP per capita and life expectancy score ...
python sandbox input:

```python
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load the dataset
df = pd.read_csv('/mnt/Happiness_rank.csv')

# Extract the columns for GDP per capita and life expectancy score
gdp_per_capita = df['Economy (GDP per Capita)'].values.reshape(-1, 1)
life_expectancy = df['Health (Life Expectancy)'].values

# Conduct linear regression
model = LinearRegression()
model.fit(gdp_per_capita, life_expectancy)
predicted_life_expectancy = model.predict(gdp_per_capita)

# Calculate R-squared
r_squared = r2_score(life_expectancy, predicted_life_expectancy)
print(r_squared)
```

```
STDOUT:
0.6666363295228102
```

The R-squared value is approximately 0.67, which indicates a poor fit for the linear regression model.

{@coefficient_determination[0.67], @model_fit["poor fit"]}

Figure 1. **An illustration of evaluation on InfiAgent-DABench.** A LLM-based agent is prompted with a data analysis question and corresponding CSV file. It plans, writes code, calls a Python sandbox and concludes to solve the problem in a ReAct way. The final answer is reformatted into specific format and matched with the closed-form label.

edge, the first benchmark for evaluating agents on data analysis tasks. Although there are many benchmarks for code-completion tasks, *e.g.*, HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and DS-1000 (Lai et al., 2023), most of them are not suitable for evaluating LLM-based agents which require complex abilities, such as planning, self-debugging. For instance, DS-1000 only requires models to insert or complete a small part of code snippet. In InfiAgent-DABench, we aim to evaluate end-to-end LLM-based agents covering abilities like task planning, writing code, leveraging Python sandbox, self-debugging (Chen et al., 2023b; Hu et al., 2024) if needed, and concluding to solve problems.

In this paper, we build InfiAgent-DABench for data analysis tasks, which includes 603 questions associated with 124 CSV files, covering realistic data analysis demands and a wide range of domains. We crawl CSV files from GitHub and instruct GPT-4 to generate open-ended questions based on the file and several key concepts for data analysis obtained from expert interviews. Then, as shown in Figure 2, constraints and format requirements containing detailed strict on methods and series of "@answer_name[answer]" as a special format the final answer must follow are generated through a format-prompting technique. Models can be automatically evaluated in this way as the answer is unique and can be easily parsed and matched. Although this benchmark is synthesised by LLMs, we argue this way is reasonable for following reasons: 1) Every data sample is strictly assessed by human experts in multiple dimensions and all unqualified samples are filtered, which means the overall data distribution is carefully human-in-the-loop curated. 2) Collecting data directly from the internet can easily lead to data leakage. 3) The cost of producing a fully human generated benchmark is obviously prohibitive.

In addition, to facilitate evaluation, we construct an agent framework to support LLMs to perform data analysis tasks. We evaluated 34 state-of-the-art LLMs and found they are confronted with challenges on handling data analysis tasks.

Moreover, to facilitate the construction of open-source LLMs with data analysis agent capability, we develop an automated method to construct an instruction-tuning dataset DAInstruct for data analysis tasks. Sharing the same merits as tool-learning instruction datasets (Qin et al., 2023b; Patil et al., 2023; Li et al., 2023) for real-world APIs, we first construct instructions based on data analysis keywords and real-world CSV files. We then generate responses to these instructions with the agent framework and GPT-4. By training open-source LLMs on this dataset, we further develop DAAgent, specialized agents focused on data analysis. The largest model DAAgent-34B, achieves a better performance over GPT-3.5 by 3.9%. To summarize, our main contributions are:

- We propose InfiAgent-DABench, which is the first benchmark, to our knowledge, for evaluating agents on data analysis tasks.
- We benchmark 34 LLMs and found that even state-of-the-art LLMs still have challenges handling data analysis tasks well.
- We propose an instruction-tuning dataset DAInstruct for data analysis tasks and develop DAAgent, an open-source data analysis agent, which achieves a better performance over GPT-3.5 by 3.9% on data analysis tasks.

2. InfiAgent-DABench Benchmark

In this section, we present InfiAgent-DABench, a benchmark created to assess the performance of LLM-based agents on data analysis tasks. Specifically, we design two main components of the benchmark: DAEval, the evaluation dataset which contains closed-form questions on data analysis, and the agent framework, which allows LLMs to solve data analysis problems in an agent setting. To ensure the quality of DAEval, we conduct a comprehensive human assessment and filter all unqualified data samples.

2.1. Dataset Construction

Similar to other recent grounding dataset, such as VQA (Goyal et al., 2017) and DocQA (Mathew et al., 2021), DAEval is composed of realistic CSV files and the corresponding questions. The questions are generated according to several key concepts in the field of data analysis, which were identified through expert interviews. All questions are closed-form with corresponding answers for a straightforward and model-free assessment. Figure 2 illustrates the whole process for constructing the dataset. We introduce the main steps as follows:

Files Collection. We gather real-world CSV files from GitHub to form our dataset. We further assess the suitability of the CSV files for question generation based on the following criteria: 1. Meaningfulness: The presence of meaningful scenarios, 2. Language uniqueness: Predominantly English used in headers and content, 3. Scale sufficiency: The column count should be greater than 5 with a minimum row count of 20.

Description Generation. As directly putting all the content of the file into the LLMs to generate questions is usually infeasible, we utilize GPT-3.5 to summarize the following information for each CSV file: 1. Description of column names and contents, 2. Data types, 3. Identification of missing values, as a mediate step before question generation. More detailed information is in Appendix E.1.

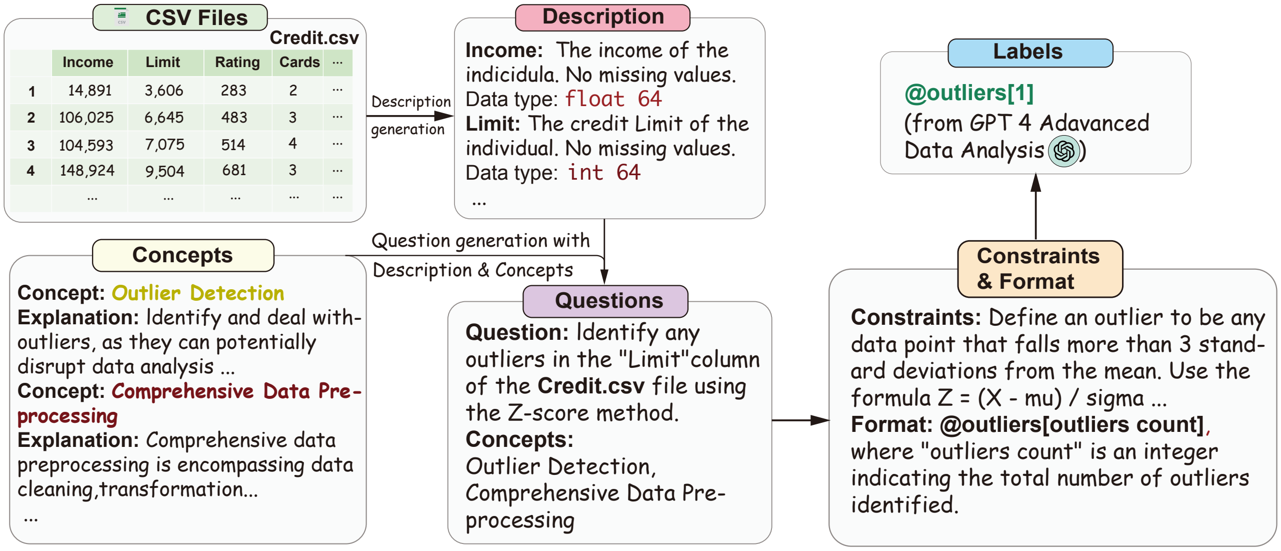
Concepts Determination. To ensure that our dataset aligns with real-world demands, we conduct expert interviews to identify key concepts in data analysis. All the experts are professional data analysts. We list these concepts and corresponding explanations in Appendix F. We remove data visualization from original concepts since we assume that a question requiring visualization can hardly be closed-form. We have a more detailed discussion on it in Appendix A. These concepts are used to guide the generation of questions in subsequent stages.

Open-ended Question Generation. In this step, we leverage GPT-4 to generate open-ended questions based on both file descriptions and the pre-defined concepts. Each question involves one or multiple concepts. We prompt GPT-4 to generate questions with an increasing difficulty and more concepts for each file description. All questions for one file are generated in one dialogue to make sure no repeated questions. Prompts can be found in Appendix E.2.

Constraints and Format Requirements Generation. Considering closed-form questions can be easily evaluated through regular expression and exact match without any external models or human efforts, we introduce a format-prompting technique that prompts GPT-4 to generate detailed constraints and format requirements for open-ended questions to make them closed-form. We have noticed that most powerful LLMs like GPT-4 have potential for automatic evaluation for open-ended questions (Fu et al., 2023; Chan et al., 2023), while many works point out issues of this way (Wang et al., 2023b;d). We also conduct a pilot experiment on GPT-4 auto-evaluating for above open-ended questions. The results show that GPT-4 could only achieve 67% consistency with human experts, indicating that GPT-4 are still unsatisfactory as an evaluator on data analysis tasks. Details about pilot experiment is in Appendix H.

However, close-form questions allow us to bypass these issues. Constraints mainly contain detailed strict on methods, workflow or statistical parameters used to answer the question. Format requirements are series of "@answer_name[answer]" where "answer_name" and "answer" are automatically defined according to the specific question, also natural description of the range of value to each "answer". We also call each "@answer_name[answer]" a subquestion in the following. We conduct human assessment and filtering to ensure adding constraints and format requirements on open-ended questions can make answers unique and easy to parse for evaluation. See Section 2.3 for details. Full prompts can be found in Appendix E.3.

Response Gathering. We leverage OpenAI ADA to generate answers of closed-form questions. Each question is sent to ADA for 3 times. Only the answer is kept when ADA consistently generates the same answer in all trials.



	Income	Limit	Rating	Cards	···
1	14,891	3,606	283	2	···
2	106,025	6,645	483	3	···
3	104,593	7,075	514	4	···
4	148,924	9,504	681	3	···
	···	···	···	···	···

Income: The income of the indicidula. No missing values. Data type: `float 64`
Limit: The credit Limit of the individual. No missing values. Data type: `int 64`
...

Concept: Outlier Detection
Explanation: Identify and deal with-outliers, as they can potentially disrupt data analysis ...
Concept: **Comprehensive Data Pre-processing**
Explanation: Comprehensive data preprocessing is encompassing data cleaning,transformation...
...

Question: Identify any outliers in the "Limit"column of the **Credit.csv** file using the Z-score method.
Concepts:
Outlier Detection, Comprehensive Data Pre-processing

Constraints: Define an outlier to be any data point that falls more than 3 standard deviations from the mean. Use the formula Z = (X - mu) / sigma ...
Format: @outliers[outliers count], where "outliers count" is an integer indicating the total number of outliers identified.

@outliers[1]
(from GPT 4 Adavanced Data Analysis)

Figure 2. **The workflow of DAEval construction.** Data analysis questions are generated with GPT-4 based on the description of CSV files and pre-defined key concepts. Then, we generate constraints and format requirements to make questions closed-form for precise evaluation. The labels are obtained from OpenAI ADA. All these are undergone strict human assessment and filtering.

Figure 2 showcases a question generated from the file `test_ave.csv`. We provide an extended case study on generated questions in Appendix I.

2.2. Agent Framework

We build an agent framework for LLM-based agents to reason in a ReAct (Yao et al., 2023) way, interact with files and invoke tools such as a Python code sandbox for easy evaluation. Agents here can be a model deployed online, such as GPT-4, or local models based on vLLM (Kwon et al., 2023). Simultaneously, we develop a Docker-based local Python code sandbox that allows the isolated execution of code, avoiding safety issues.

2.3. Human Assessment

To ensure a high-quality dataset, we conduct an in-depth human assessment. We invite experts to assess the dataset quality, and also conduct a comparison between human-made and GPT-4 generated data analysis questions.

Metrics. We evaluate the quality of files, questions and labels. For each part, we define one or more metrics which are crucial for a data analysis dataset. Here's a brief introduction:

- **Suitableness**: This measures whether a CSV file is suitable for data analysis.
- **Reasonableness**: This measures whether the question, constraints and format requirements express in a natural and reasonable way, with no conflict with each other.
- **Value**: This measures the value of generated questions. We want the questions to be valuable in practice. For instance, calculating the mean of salaries could be valuable as a data analysis question, while calculating the mean of ID numbers is usually valueless.
- **Restrictiveness**: This measures whether the constraints are strict enough to ensure the uniqueness of answers.
- **Alignment**: This measures the alignment between questions and data file. Questions should only require data existed in the file, also the type and range of the data are aligned with questions.
- **Correctness**: This measures whether the labels for questions are correct.

2.3.1. DATASET QUALITY

We invite human experts to assess the quality of generated dataset and remove samples unqualified in any metric. We refer readers to Appendix J.1 for detailed settings for human assessment on dataset quality.

Results. Table 1 shows the percentage of samples passed in specific metrics and overall pass rate. 85% samples are qualified and kept in final, demonstrating the effectiveness of our dataset construction method. In addition, we conduct bad case analysis on the dataset in Appendix J.2.

File	Question			Label		
Suitableness ↑	**Reasonableness ↑**	**Value ↑**	**Restrictiveness ↑**	**Alignment ↑**	**Correctness ↑**	**Overall ↑**
0.98	0.95	0.98	0.94	0.98	0.93	0.85

Table 1. **Results of human assessment on the quality of DAEval.** We conduct quality assessment over files, questions and labels, where each part has one or more metrics. We report the percentage of samples considered qualified on each metric. We keep questions qualified in all metrics. Overall represents the percentage of samples qualified in total.

Source	**Reasonableness ↑**	**Value ↑**	**Restrictiveness ↑**	**Alignment ↑**	**Difficulty (%) ↑**
Human	4.66	4.73	4.14	4.71	39.34
GPT-4	4.70	4.85	4.74	4.92	60.66

Table 2. **Results of human assessment to compare questions generated from human and GPT-4.** Reasonableness, Value, Restrictiveness and Alignment are graded from 1-5 (1 is worst and 5 is best). Difficulty is calculated by win rate between human and GPT-4.

2.3.2. HUMAN VS GPT-4

To ensure GPT-4 generated questions have a comparable quality with human-made ones, we invite experts to write data analysis questions. Then other experts are asked to grade from 1 to 5 (1 is worst and 5 is best) to metrics on questions mentioned above each. Additionally, we ask experts to decide which question is more difficult in a question pair randomly sampled from human-made and GPT-4 generated questions respectively to compare the difficulty of questions made from two sides. We calculate win rate to reflect the relative difficulty of questions. Detailed settings for this part of human assessment are in Appendix 2.

Results. Table 2 shows results compared between human-made and GPT-4 generated questions. We found that GPT-4 generated questions take advantage on Restrictiveness, Alignment and Difficulty, and a slightly better performance on Reasonableness and Value, which indicates that DAEval are comparable with human-made dataset.

2.4. Benchmark Statistics

In this section, we provide statistics on DAEval. We split the dataset into validation set and test set. The validation set is open to public, including 257 questions with 52 csv files and the rest is for test set which is closed for avoiding data leakage. All the subsequent information is based on the validation set.

Basic Statistics. Table 3 lists basic statistical information on files and questions, including the number of files, questions, subquestions, etc.

Domains of Files. Figure 3 shows the distribution of different domains the files in. We cluster the files into 9 domains with the help of GPT-3.5. The result shows that our dataset covers a large range of domains. Details about the domains and corresponding descriptions are listed in Appendix G.

Concepts of Questions. Figure 4 shows the distribution of different concepts the questions involve. Table 4 shows the distribution of questions based on the number of concepts involved. It reveals that a larger proportion of questions involve 1 or 2 concepts. This is reasonable that it's uncommon for a scenario where a question can involve so many concepts at once.

Difficulty of Questions. GPT-4 is utilized for the quantitative assessment of question difficulty in DAEval. Questions are classified into three levels: easy, medium, and hard. Table 4 presents the distribution of these levels across different

Dataset Component	**Count**
# of CSV Files	52
# of Avg. Rows per File	9639.7
# of Avg. Columns per File	10.6
# of Questions	257
# of Subquestions	461
# of Max Subquestions per Question	8

Table 3. Basic statistics on the validation set of DAEval.

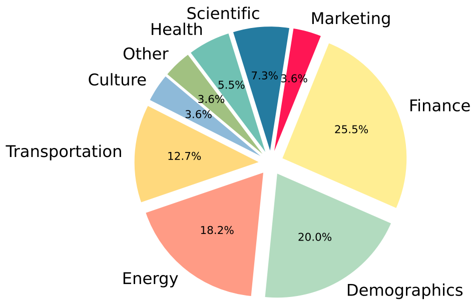


Figure 3. **Distribution of domains the files in.** For brevity, we use the first word of each category as the label. The completed domain names are listed in Appendix G.

		Easy	Medium	Hard	Total
Concepts	Summary Statistics	41	17	32	90
	Feature Engineering	3	14	33	50
	Correlation Analysis	10	32	30	72
	Machine Learning	0	0	19	19
	Distribution Analysis	21	23	20	64
	Outlier Detection	5	20	10	35
	Comprehensive Data Preprocessing	8	9	28	45
# of Concepts per Question	1 concept	77	59	14	150
	2 concepts	4	28	65	97
	3 concepts	1	0	8	9
	4 concepts	0	0	1	1
Total		82	87	88	257

Table 4. Distribution of question difficulties across different concepts and number of concepts.

concepts and the number of concepts per question. The detailed method of quantitative assessment is described in Appendix K. This indicates a wide spectrum of complexities within DAEval, ensuring a comprehensive evaluation.

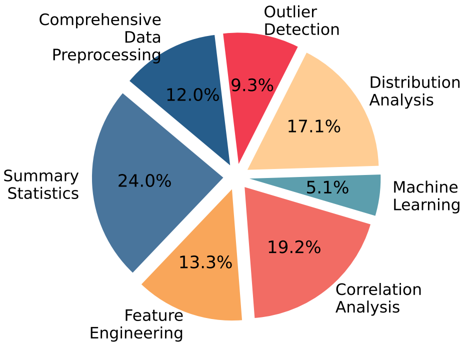


Figure 4. Distribution of concepts the questions involve.

2.5. Instruction-tuning Dataset

In this section, we introduce DAInstruct, an instruction-tuning (Wei et al., 2022a; Zhang et al., 2024b) dataset for data analysis, on which we train DAAgent, a specialized agent for data analysis. The collection of DAInstruct is carried out in three steps: 1. Creating a diverse set of data analysis questions for CSV files 2. Collecting response trajectories to these questions through our agent framework 3. Filtering low-quality samples with rules and GPT-4. Statistics of DAInstruct are in Appendix L. Moreover, a case study on DAInstruct is provided in Appendix O. The following sections will detail these steps.

Question Generation. We employ GPT-4 to generate a set of keywords relevant to data analysis. Then, these keywords undergo a manual screening process to ensure relevance and accuracy. Then, GPT-4 is prompted with file description similar to those in DAEval, and above keywords to generate questions for each CSV file. The keywords and prompt can be found in Appendix M.

Response Collection. In this step, we leverage our agent framework to generate responses in a ReAct (Yao et al., 2023) way. It is an iteration of problem-solving steps, where GPT-4 engages with code execution environment in a conversational format. The process includes reasoning, code writing, execution, and iterative refinement based on the results obtained. Details can be found in Appendix N.

Filtering. This step involves a filtering with rules and GPT-4. Specific rules are employed to identify and eliminate data with hallucination and incomplete solution. Then, the data undergoes filtering with the aid of GPT-4. The criteria includes five aspects: Python code relevance, interpretation of system notifications, conciseness, understanding of the Python sandbox environment, and solution provision. All prompts can be found in Appendix M.

3. Experiments

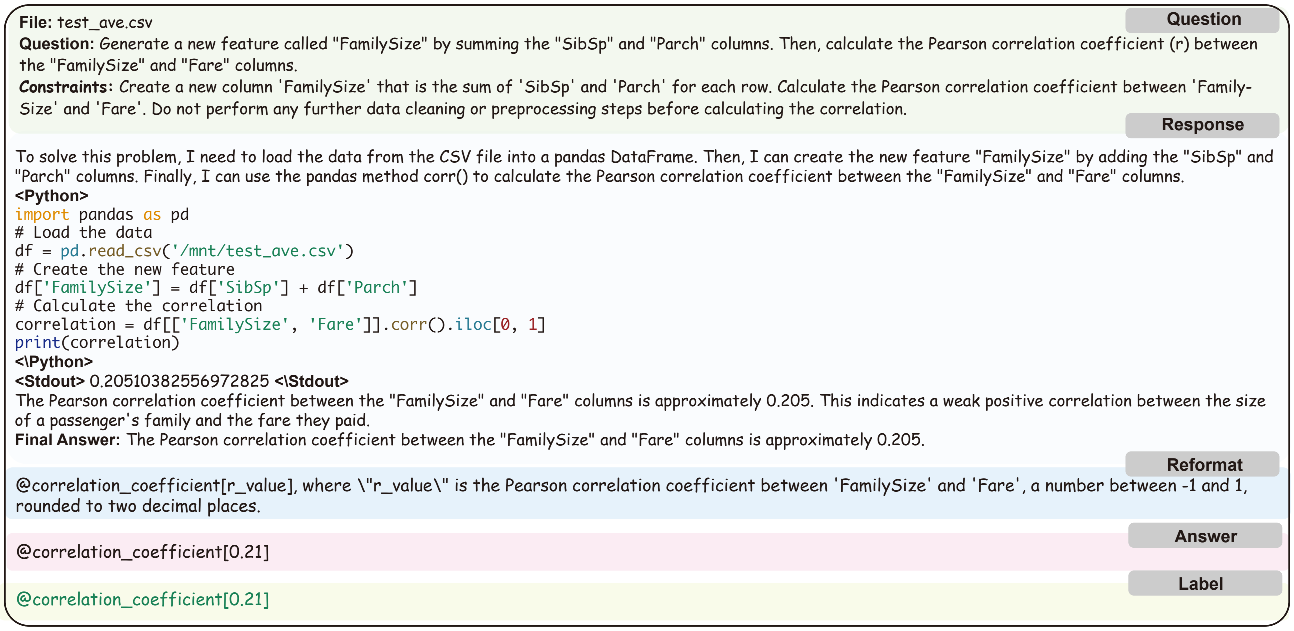
3.1. Setup

3.1.1. MODELS

We categorize the models benchmarked by 4 groups: proprietary models, open-source general LLMs, open-source code LLMs and agent frameworks.

Proprietary Models. This group consists of LLMs available as commercial services, providing API access for integration into various applications. We benchmark OpenAI GPT-4 (OpenAI, 2023b), GPT-3.5 (OpenAI, 2023a), Claude-2.1 (Anthropic, 2023), Abab5.5 (MiniMax, 2023) and Gemini-Pro (Team et al., 2023).

Open-source General LLMs. This category includes open-source LLMs designed for a wide range of tasks and versatility in applications. We benchmark Vicuna (Chiang et al., 2023), ChatGLM (Zeng et al., 2022), Baichuan (Yang et al., 2023a), Qwen (Bai et al., 2023), InternLM (Team, 2023a),


Question
Response
Reformat
Answer
Label

File: test_ave.csv

Question: Generate a new feature called "FamilySize" by summing the "SibSp" and "Parch" columns. Then, calculate the Pearson correlation coefficient (r) between the "FamilySize" and "Fare" columns.

Constraints: Create a new column 'FamilySize' that is the sum of 'SibSp' and 'Parch' for each row. Calculate the Pearson correlation coefficient between 'FamilySize' and 'Fare'. Do not perform any further data cleaning or preprocessing steps before calculating the correlation.

To solve this problem, I need to load the data from the CSV file into a pandas DataFrame. Then, I can create the new feature "FamilySize" by adding the "SibSp" and "Parch" columns. Finally, I can use the pandas method corr() to calculate the Pearson correlation coefficient between the "FamilySize" and "Fare" columns.

<Python>

```
import pandas as pd
# Load the data
df = pd.read_csv('/mnt/test_ave.csv')
# Create the new feature
df['FamilySize'] = df['SibSp'] + df['Parch']
# Calculate the correlation
correlation = df[['FamilySize', 'Fare']].corr().iloc[0, 1]
print(correlation)
```

<\Python>

<Stdout> 0.20510382556972825 **<\Stdout>**

The Pearson correlation coefficient between the "FamilySize" and "Fare" columns is approximately 0.205. This indicates a weak positive correlation between the size of a passenger's family and the fare they paid.

Final Answer: The Pearson correlation coefficient between the "FamilySize" and "Fare" columns is approximately 0.205.

@correlation_coefficient[r_value], where \"r_value\" is the Pearson correlation coefficient between 'FamilySize' and 'Fare', a number between -1 and 1, rounded to two decimal places.

@correlation_coefficient[0.21]

@correlation_coefficient[0.21]

Figure 5. **The evaluation process.** We first prompt the agent with the file, the question and constraints. The agent answers with the calling of external Python sandbox. Then, we use GPT-3.5 to reformat the response into format requirements. At last, we use regular expression matching to draw the answer and compare it with the label.

AgentLM (Zeng et al., 2023), Mistral (Mistral.ai, 2023) and Yi (01ai, 2023).

Open-source Code LLMs. These are specialized LLMs tailored for understanding and generating programming code. Our experiments cover Code Llama (Rozière et al., 2023), WizardCoder (Luo et al., 2023), Phind CodeLlama (Phind, 2023), DeepSeek Coder (DeepSeek, 2023) and XwinCoder (Team, 2023c).

Agent Frameworks. These frameworks contain one or more LLM-based agents which can reason, plan and use tools. We conduct evaluations on single agent framework XAgent (Team, 2023b), Qwen-Agent (Bai et al., 2023) and multi-agents framework AutoGen (Wu et al., 2023a). We use GPT-4 as the LLMs for XAgent and AutoGen, and Qwen-72B-Chat for Qwen-Agent.

3.1.2. IMPLEMENTATION DETAILS

Figure 5 illustrates a case on the process of evaluation. All experiments are based on our agent frameworks except those ready-made frameworks. We first prompt LLMs with the file, question and constraints. Considering that most models hardly follow the format requirements exactly, we add a reformat step after the original responses which formats the responses with the format requirements with one-shot prompting. We conduct ablation studies and manual check on this reformat step, indicating this step doesn't effect to the precision of evaluation. More details about reformatting can be found in Appendix Q. Other implementation details are in Appendix P.

3.2. Results

Figure 6 shows performances and model sizes of different models benchmarked on validation set of DAEval. We provide precise performances in Appendix D.

3.3. General Analysis

Based on above results, we have several key findings:

Data analysis tasks are challenging for current LLMs. Among all benchmarked models, GPT-4 outperforms others with an accuracy of 78.99%, highlights a significant room for improvement in even the most advanced models, since the upper limit of performance in DAEval is 100%. Notably, the highest-performing open-source LLMs Qwen-72B-Chat, still lags behind by 19% with GPT-4. The substantial gap between proprietary and open-source models indicates an overarching need for enhancement in open-source LLMs for data analysis tasks.

Most powerful open-source LLMs achieve comparable performance with GPT-3.5. We found that, open-source LLMs enjoy a fast development over past periods. Although early open-source models seem an obvious gap behind pro-

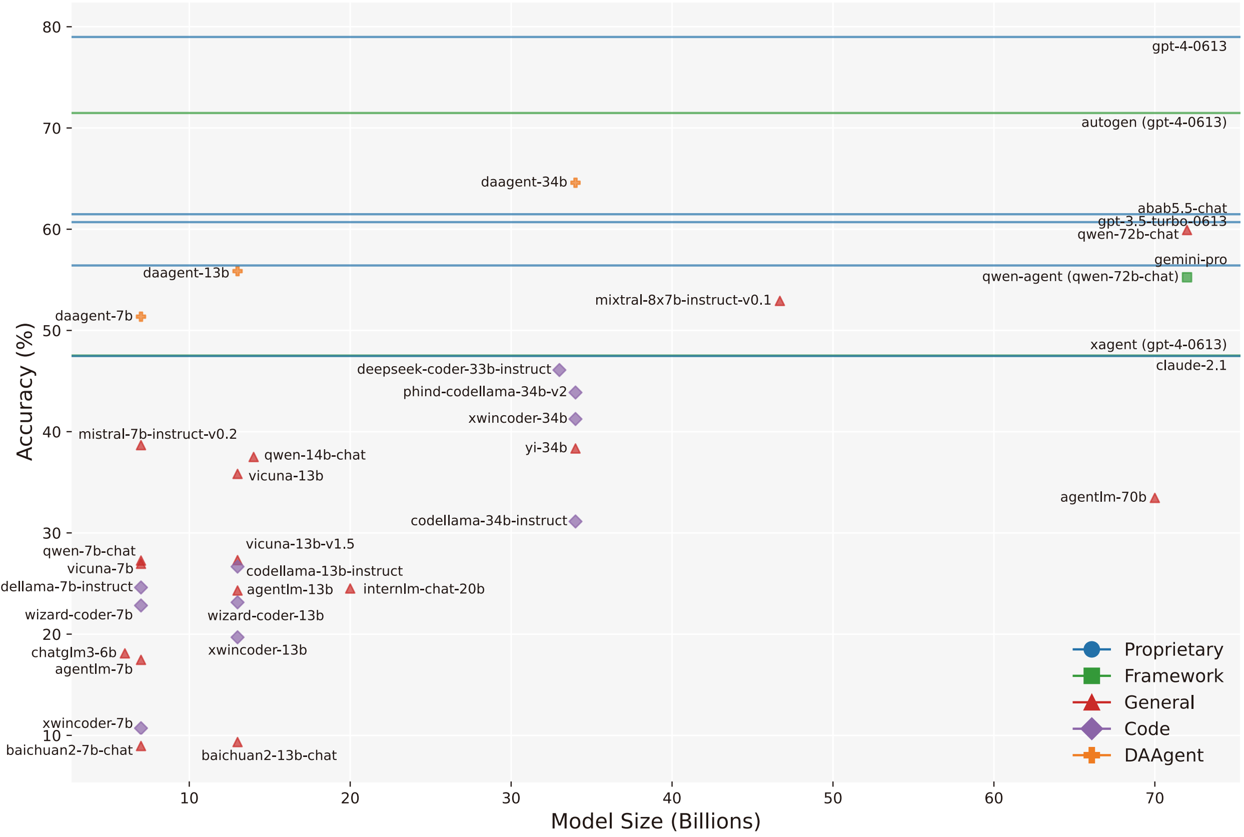


Figure 6. **The performance of models benchmarked in InfiAgent-DABench.** Models are categorized into 4 groups: Proprietary Models, Open-source General LLMs, Open-source Code LLMs, Agent Frameworks. LLMs behind agent frameworks are shown in parentheses. We also report our DAAgents. Models of known size are plotted as dots while models of unknown size as horizontal lines.

prietary models, they catch up quickly. Qwen-72B-Chat achieves 59.92%, the best performance among open-source models, which is only slightly lower than GPT-3.5 by 0.78% and even better than Gemini-Pro by 3.50%.

DAAgent achieves a better performance than GPT-3.5. DAAgent-34B, our biggest model instruction-tuned by DAInstruct from CodeLlama, achieves a better performance with GPT-3.5 by 3.89%, although it has much less parameters than that proprietary model. Otherwise, Table 5 compares DAAgent with corresponding size of Code Llama with general instruction-tuning showcases a large performance increase brought by instruction-tuning on DAInstruct. Both indicate DAInstruct can be seen as an effective instruction-tuning dataset to increase model's ability on data analysis.

	Accuracy (%)		
Model	7B	13B	34B
CodeLlama-Instruct	24.61	26.67	31.13
DAAgent	51.36	55.86	64.59

Table 5. The performance of models with different instruction-tuning dataset acorss all Code Llama model sizes.

3.4. Self-debug Ability

In this section, we focus on self-debug (Chen et al., 2023b; Shinn et al., 2023; Hu et al., 2024), a specific ability of LLM-based agents in data analysis setting, which means agents can debug automatically according to the feedback, *e.g.*, error messages from the code interpreter. As shown in Table 6, we present the number of cases passed or failed with or without self-debug on the dev set of DABench. We recognize a successful self-debug by simultaneously detecting specific string (*e.g.*, "Traceback") in the error messages from the code interpreter during the interaction between agents and the sandbox, and the final successful solution of the problem. We also define Self-Debug Success Rate (SDSR), which is the ratio of successful self-debug cases to the total number of self-debug attempts, to reflect the ability

Model	#P w/ debug	#P w/o debug	#F w/ debug	#F w/o debug	SDSR
gpt-4-0613 (OpenAI, 2023b)	42	161	17	37	0.71
gpt-3.5-turbo-0613 (OpenAI, 2023a)	36	120	41	60	0.47
qwen-72b-chat (Bai et al., 2023)	27	127	69	34	0.28
daagent-34b (ours)	40	126	40	51	0.50

Table 6. Performance comparison of different models on self-debug capability. The table shows the number of cases passed or failed with or without self-debug on the dev set of DABench. "P" and "F" represent if the agents pass or fail on the problem. "w/ debug" and "w/o debug" indicate if agents receive feedbacks from the code interpreter. SDSR stands for Self-Debug Success Rate, defined as the ratio of successful self-debug cases to the total number of self-debug attempts.

of self-debug:

$$\text{SDSR} = \frac{\#\text{Pass w/ self-debug}}{\#\text{Pass w/ self-debug} + \#\text{Fail w/ self-debug}}.$$

We can find that GPT-4 has not only the best performance but also a much stronger self-debug ability compared with other models. The Self-debug Success Rate of DAAgent-34B is slightly higher than GPT-3.5, while Qwen-72B shows weakness in self-debug. We believe improving self-debug ability is a promising way to further develop LLM-based agents in data analysis domain for two reasons: 1) Letting LLMs refine by themselves based on feedback has been broadly studied recently and show positive results on many different domains (Madaan et al., 2023; Shinn et al., 2023; Wang et al., 2023a). 2) Feedback from code interpreter is usually cheap and reliable. Although in some situations, getting reliable feedback can be expensive or impossible, getting feedback from code interpreter is trivial and can be implemented easily. We leave how to improve this kind of ability as an important direction for future research.

4. Related works

The rapid advancement of LLM-based agents and multi-agent systems (Liu et al., 2023b; Zhang et al., 2024c; Chen et al., 2023a; Wu et al., 2023b) necessitates precise and all-encompassing evaluation methods, while remains a challenge at present. Recent studies have introduced several new settings to evaluate from different angles. WebShop (Yao et al., 2022a) and WebArena (Zhou et al., 2023) benchmark LLM-based agents in web environments. Some other benchmarks evaluate LLM-based agents with tool using, like ToolBench (Qin et al., 2023b) and Gentopia (Xu et al., 2023a). In our work, we propose the InfiAgent-DABench that evaluates agents on data analysis tasks. Such tasks have a higher requirement for LLMs to produce structured and precise intermediate executable steps. Another series of works related to InfiAgent-DABench are Code LLMs and corresponding evaluations. Code LLMs, which specialize in understanding and generating programming code, have developed fast in the past few years (Zan et al., 2023; Zhang et al., 2023b; Chen et al., 2021; Rozière et al., 2023; Luo et al., 2023). Various benchmarks on code and tabular processing have also been proposed (Lu et al., 2021; Lai et al., 2023; Lu et al., 2023; Zhuang et al., 2023; Yang et al., 2023b). However, these benchmarks still miss a comprehensive evaluation to check agents end-to-end task solving capability. More details about Code LLMs and related benchmarks can be found in Appendix B.

5. Conclusion

We introduce InfiAgent-DABench, the first benchmark specifically designed for evaluating LLM-based agents on data analysis tasks. Our extensive benchmarking of 34 cutting-edge LLMs reveals that contemporary models still face challenges in effectively managing data analysis tasks. Additionally, we develop DAAgent, a specialized agent for data analysis, trained on DAInstruct, an instrcution-tuning dataset generated by GPT-4. Our contributions not only offer a valuable benchmark for assessing LLM-based agents in data analysis but also provide insights into the current capabilities and limitations of these models in this domain.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (62441605, 62376243), and the Starry Night Science Fund of Zhejiang University Shanghai Institute for Advanced Study (SN-ZJU-SIAS-0010). We appreciate the contribution of Haotian Zhou on helping construct the agent pipeline and environment. We also extend our gratitude to Rufei Liu, Feiyu Zhang and Xinxin Wang's hard work on data annotation, Sibo Xiao on the help of building demo for this work.

Impact Statements

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

01ai. Yi. https://01.ai/, 2023.

Anthropic. Claude-2.1. https://www.anthropic.com/index/claude-2-1, 2023.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Chan, C.-M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., and Liu, Z. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., and Zhou, J. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors, 2023a.

Chen, X., Lin, M., Schärli, N., and Zhou, D. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023b.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Covington, D. *Analytics: Data Science, Data Analysis, and Predictive Analytics for Business*. CreateSpace Independent Publishing Platform, 2016.

DeepSeek. Deepseek coder: Let the code write itself. https://github.com/deepseek-ai/DeepSeek-Coder, 2023.

Fu, J., Ng, S.-K., Jiang, Z., and Liu, P. Gptscore: Evaluate as you desire. *arXiv preprint arXiv:2302.04166*, 2023.

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. Making the v in vqa matter: Elevating the role of image understanding in visual question answering, 2017.

Hardy, M. A. and Bryman, A. Handbook of data analysis. 2004.

Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., and Steinhardt, J. Measuring coding challenge competence with apps, 2021.

Hu, X., Kuang, K., Sun, J., Yang, H., and Wu, F. Leveraging print debugging to improve code generation in large language models, 2024.

Huang, J. and Chang, K. C.-C. Towards reasoning in large language models: A survey, 2023.

Huang, Y., Bai, Y., Zhu, Z., Zhang, J., Zhang, J., Su, T., Liu, J., Lv, C., Zhang, Y., Lei, J., Fu, Y., Sun, M., and He, J. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models, 2023.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Koop, G. *Analysis of financial data*. John Wiley & Sons Inc., 2022.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., Yih, W.-t., Fried, D., Wang, S., and Yu, T. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Li, M., Song, F., Yu, B., Yu, H., Li, Z., Huang, F., and Li, Y. Api-bank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Liang, X., Wang, B., Huang, H., Wu, S., Wu, P., Lu, L., Ma, Z., and Li, Z. Unleashing infinite-length input capacity for large-scale language models with self-controlled memory system. *arXiv preprint arXiv:2304.13343*, 2023.

Liu, M. X., Sarkar, A., Negreanu, C., Zorn, B., Williams, J., Toronto, N., and Gordon, A. D. "what it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–31, 2023a.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., and Tang, J. Agentbench: Evaluating llms as agents. 2023b. URL `https://arxiv.org/pdf/2205.06175.pdf`.

Lu, P., Qiu, L., Chang, K.-W., Wu, Y. N., Zhu, S.-C., Rajpurohit, T., Clark, P., and Kalyan, A. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning, 2023.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

Lucas, K. Open interpreter. `https://github.com/KillianLucas/open-interpreter`, 2023.

Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., and Jiang, D. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback, 2023.

Mathew, M., Karatzas, D., and Jawahar, C. V. Docvqa: A dataset for vqa on document images, 2021.

MiniMax. Abab5.5. `https://api.minimax.chat/`, 2023.

Mistral.ai. Mistral. `https://mistral.ai/product/`, 2023.

Nakajima, Y. Babyagi. `https://github.com/yoheinakajima/babyagi`, 2023.

OpenAI. Openai models - openai api. `https://platform.openai.com/docs/models/gpt-3-5`, 2023a.

OpenAI. Gpt-4 technical report. 2023b. URL `https://arxiv.org/pdf/2303.08774.pdf`.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Phind. Phind code llama. `https://www.phind.com/blog/code-llama-beats-gpt4`, 2023.

Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., Yang, F., Dong, H., Zhang, J., Wang, L., et al. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2023.

Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Huang, Y., Xiao, C., Han, C., Fung, Y. R., Su, Y., Wang, H., Qian, C., Tian, R., Zhu, K., Liang, S., Shen, X., Xu, B., Zhang, Z., Ye, Y., Li, B., Tang, Z., Yi, J., Zhu, Y., Dai, Z., Yan, L., Cong, X., Lu, Y., Zhao, W., Huang, Y., Yan, J., Han, X., Sun, X., Li, D., Phang, J., Yang, C., Wu, T., Ji, H., Liu, Z., and Sun, M. Tool learning with foundation models, 2023a.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023b.

Reddy, C. K. and Aggarwal, C. C. *Healthcare data analytics*, volume 36. CRC Press, 2015.

Reworkd. Agentgpt. `https://github.com/reworkd/AgentGPT`, 2023.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code llama: Open foundation models for code. 2023. URL `https://arxiv.org/pdf/2308.12950.pdf`.

Sabina, L. and Zalta, E. N. Scientific research and big data. *The Stanford Encyclopedia of Philosophy (Summer 2020 Edition)*, 2020.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning, 2023.

Sun, H., Zhuang, Y., Kong, L., Dai, B., and Zhang, C. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023.

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Team, I. Internlm: A multilingual language model with progressively enhanced capabilities. https://github.com/InternLM/InternLM, 2023a.

Team, X. Xagent: An autonomous agent for complex task solving, 2023b.

Team, X.-L. Xwin-lm, 9 2023c. URL https://github.com/Xwin-LM/Xwin-LM.

Torantulino. Autogpt. https://github.com/Significant-Gravitas/AutoGPT, 2023.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models, 2023a.

Wang, J., Liang, Y., Meng, F., Shi, H., Li, Z., Xu, J., Qu, J., and Zhou, J. Is chatgpt a good nlg evaluator? a preliminary study. *arXiv preprint arXiv:2303.04048*, 2023b.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023c.

Wang, P., Li, L., Chen, L., Cai, Z., Zhu, D., Lin, B., Cao, Y., Liu, Q., Liu, T., and Sui, Z. Large language models are not fair evaluators, 2023d.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners, 2022a.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022b.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. 2023a.

Wu, Y., Zhou, S., Liu, Y., Lu, W., Liu, X., Zhang, Y., Sun, C., Wu, F., and Kuang, K. Precedent-enhanced legal judgment prediction with llm and domain-model collaboration, 2023b.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Xie, T., Zhou, F., Cheng, Z., Shi, P., Weng, L., Liu, Y., Hua, T. J., Zhao, J., Liu, Q., Liu, C., et al. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*, 2023.

Xu, B., Liu, X., Shen, H., Han, Z., Li, Y., Yue, M., Peng, Z., Liu, Y., Yao, Z., and Xu, D. Gentopia: A collaborative platform for tool-augmented llms, 2023a.

Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., and Xu, D. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023b.

Yang, A., Xiao, B., Wang, B., Zhang, B., Bian, C., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., Yang, F., Deng, F., Wang, F., Liu, F., Ai, G., Dong, G., Zhao, H., Xu, H., Sun, H., Zhang, H., Liu, H., Ji, J., Xie, J., Dai, J., Fang, K., Su, L., Song, L., Liu, L., Ru, L., Ma, L., Wang, M., Liu, M., Lin, M., Nie, N., Guo, P., Sun, R., Zhang, T., Li, T., Li, T., Cheng, W., Chen, W., Zeng, X., Wang, X., Chen, X., Men, X., Yu, X., Pan, X., Shen, Y., Wang, Y., Li, Y., Jiang, Y., Gao, Y., Zhang, Y., Zhou, Z., and Wu, Z. Baichuan 2: Open large-scale language models, 2023a.

Yang, J., Prabhakar, A., Narasimhan, K., and Yao, S. Intercode: Standardizing and benchmarking interactive coding with execution feedback, 2023b.

Yang, K., Liu, J., Wu, J., Yang, C., Fung, Y. R., Li, S., Huang, Z., Cao, X., Wang, X., Wang, Y., et al. If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents. *arXiv preprint arXiv:2401.00812*, 2024.

Yao, S., Chen, H., Yang, J., and Narasimhan, K. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. 2023. URL https://arxiv.org/pdf/2210.03629.pdf.

Zan, D., Chen, B., Zhang, F., Lu, D., Wu, B., Guan, B., Yongji, W., and Lou, J.-G. Large language models meet NL2Code: A survey. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7443–7464, Toronto, Canada, July 2023. Association for Computational Linguistics.

doi: 10.18653/v1/2023.acl-long.411. URL `https://aclanthology.org/2023.acl-long.411`.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Zeng, A., Liu, M., Lu, R., Wang, B., Liu, X., Dong, Y., and Tang, J. Agenttuning: Enabling generalized agent abilities for llms, 2023.

Zhang, K., Zhao, F., Kang, Y., and Liu, X. Memory-augmented llm personalization with short-and long-term memory coordination. *arXiv preprint arXiv:2309.11696*, 2023a.

Zhang, M., Li, H., Wu, F., and Kuang, K. Metacoco: A new few-shot classification benchmark with spurious correlation. In *International Conference on Learning Representations, ICLR*, 2024a.

Zhang, S., Dong, L., Li, X., Zhang, S., Sun, X., Wang, S., Li, J., Hu, R., Zhang, T., Wu, F., and Wang, G. Instruction tuning for large language models: A survey, 2024b.

Zhang, W., Tang, K., Wu, H., Wang, M., Shen, Y., Hou, G., Tan, Z., Li, P., Zhuang, Y., and Lu, W. Agent-pro: Learning to evolve via policy-level reflection and optimization, 2024c.

Zhang, Z., Chen, C., Liu, B., Liao, C., Gong, Z., Yu, H., Li, J., and Wang, R. Unifying the perspectives of nlp and software engineering: A survey on language models for code, 2023b.

Zhao, P., Jin, Z., and Cheng, N. An in-depth survey of large language model-based artificial intelligence agents. *arXiv preprint arXiv:2309.14365*, 2023.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. Webarena: A realistic web environment for building autonomous agents, 2023.

Zhuang, Y., Yu, Y., Wang, K., Sun, H., and Zhang, C. Toolqa: A dataset for llm question answering with external tools, 2023.

A. Limitations and Future work

A significant limitation of our benchmark is its exclusion of questions related to visualization. During expert interviews, we recognized the critical role of visualization in data analysis. However, designing closed-form questions for visualization is greatly challenging, as the outcomes are often pictorial rather than textual. To evaluate, two potential approaches emerge:

Evaluating code correctness. This method allows for the use of text-only LLMs in evaluation. While it benefits from leveraging existing LLMs, this approach might lead to inaccuracies. The primary goal in data analysis tasks is to generate accurate visual representations, not just the underlying code. Only assessing from code could be challenging for a precise evaluation in some situations.

Evaluating with multimodal models. Multimodal large models (OpenAI, 2023b; Team et al., 2023) have developed rapidly in recent times, making it possible for them to become automatic evaluators. Bai et al. (2023) leverages vision-language models like GPT-4V (OpenAI, 2023b) to evaluate visualization problems. This approach directly assesses the accuracy of the visual outputs. While this method is advantageous in evaluating the end product, current multimodal models are suffering from suboptimal performance and costly, leading to unsatisfactory assessment of data visualization.

Neither of above approaches can achieve the accuracy and convenience of closed-form assessments. We leave the precise evaluation of visualization in data analysis as a direction for future research.

B. Detailed related works

B.1. Code LLMs

Code LLMs are large language models specialized in understanding and generating programming code. These LLMs enjoy a fast development over past few years (Zan et al., 2023; Zhang et al., 2023b). Codex (Chen et al., 2021) is one of the most famous in these models provided by a commercial API from OpenAI. Meanwhile, researchers have developed a large sum of open-source code LLMs, *e.g.*, Code Llama (Rozière et al., 2023), WizardCoder (Luo et al., 2023), XwinCoder (Team, 2023c) and DeepSeek Coder (DeepSeek, 2023).

B.2. Benchmarks for Code

Benchmarks play an important roles for the development of AI (Hendrycks et al., 2021; Huang et al., 2023; Zhang et al., 2024a). Over the past years, various benchmarks with different tasks have been proposed to evaluate models for code. CodeXGLUE (Lu et al., 2021) is a comprehensive benchmark covering tasks for both code understanding and generation. In addition, some benchmarks focus on code generation, especially in a competition-level (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Li et al., 2022). DS-1000 (Lai et al., 2023) is a benchmark for code generation in data science. Bai et al. (2023) creates a dataset containing math, data visualization and other general-purpose tasks to assess LLMs' ability on using the Python code interpreter. We have a more detailed comparison between InfiAgent-DABench and this work in Appendix C.

C. Comparison between InfiAgent-DABench and Code Interpreter Benchmark in Qwen-Agent

There are two main differences between InfiAgent-DABench and code interpreter benchmark in Qwen-Agent (Bai et al., 2023): 1. We benchmark LLMs in data analysis tasks covering series of concepts while Bai et al. (2023) only contains a few data analysis questions in data visualization and its other general-purpose tasks. 2. Our benchmark takes a closed-form evaluation on correctness which doesn't rely on any external models as evaluators. More importantly, Bai et al. (2023) evaluates code correctness for math and data visualization problems with the help of current multi-modal models but only code executability for its general-purpose tasks.

D. Performances on InfiAgent-DABench

Table 7 shows the accuracy of different models in InfiAgent-DABench. We also draw a spider chart to illustrate the performance in questions involved different concepts in Figure 7.

E. Prompts for DAEval Construction

In this section, we list all the prompts used in the process of question generation.

E.1. Description Generation

Table 8 shows prompts for description generation.

E.2. Question Generation

Table 9 shows prompts for question generation. We prompt GPT-4 in the following way: 1) Three simple questions containing only one concept. 2) Three complex questions involving one or multiple concepts. 3) Three more complex questions involving multiple concepts.

E.3. Constraints and Format Requirements Generation

Table 10 shows prompts for constraints and format requirements generation.

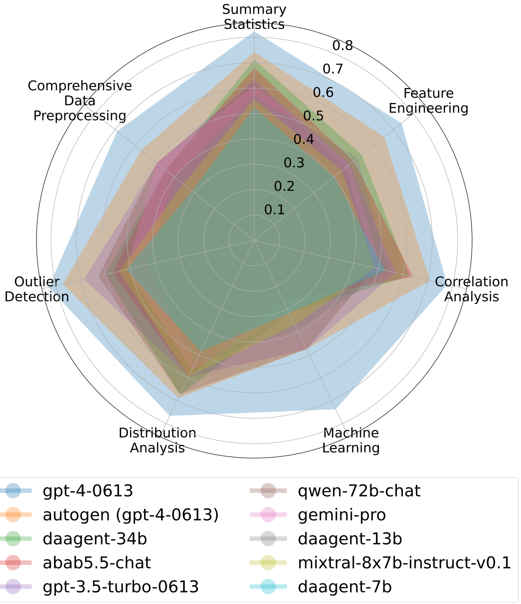


Figure 7. Accuracy of top 10 models in DAEval on the questions involved different concepts.

F. Detailed Information about Key Concepts

Table 11 shows key concepts on data analysis and corresponding explanations obtained from expert interview. We use GPT-3.5 to help us summarize the interview scripts. We could not directly publicize them for some reasons.

G. Detailed Information about Domains

We cluster the CSV files into 9 domains with the help of GPT-3.5. Table 12 shows the names of domain and corresponding explanations.

H. Detailed Information about Pilot Experiments

We randomly sample 119 questions and corresponding answers from GPT-4, GPT-3.5, DAAgent-34B and ask both human experts and GPT-4 to evaluate based on the same criteria. Each answer is rated based on the level of resolution: 0: Did not solve the problem or barely addressed it, 1: Partly solved the problem, 2: Mostly solved the problem and only have minor errors, 3: Completely and exactly solved the problem. Table 13 shows the consistency rate between human experts and GPT-4. We found that when GPT-4 works as an evaluator on data analysis tasks, it only achieves about 67% consistency with human experts, which is obviously unsatisfactory as an evaluator for a benchmark.

I. Extended Case Study on Generated Questions

Table 14 showcases several questions generated for the file `test_ave.csv`. This file meticulously documents the personal information and travel details of passengers on the Titanic, including their survival status, cabin class, name, gender, age, and embarkation point, among other aspects. The generated questions include one or more subquestions according to the specific question and constraints.

J. Human Assessment

J.1. Settings for Dataset Quality Assessment

For a high-quality dataset, we invite 3 human experts with math and programming skills to assess the quality of files, questions and labels of all the data samples. For each part, we define one or more metrics. For each metric, to a precise assessment, we decompose it into several yes-or-no questions for experts. Table 15 shows our questionnaire, including questions and expected response formats. We invite 3 experts with math and programming skills to assess all data samples in DAEval. Only all yes-or-no questions of one metric are positive, the assessment results for this metric is positive. We report the percentage of positiveness on each metric and an overall percentage of positiveness on all the metrics. Only data samples positive on all the metrics are kept. Otherwise, we asked experts to compare difficulty on 216 anonymous question pairs (108 with single concept and 108 with multiple concepts) sampled from human-made and GPT-4 generated questions. Questions with single concept are only compared with other single ones, the same for those with multiple concepts. Table 16 shows the questionnaire to experts.

J.2. Bad Case Analysis

Table 17 presents two bad cases found from human assessment. In the first case, we found all values in “Type” column in `2019-08_edges.csv` are the same, which means using Random Forest Classifier to predict it here is meaningless. This question doesn’t meet our criteria on Alignment. In the second case, the question asks with “the given dates” but not provides with the specific dates in the following constraints, which means the question doesn’t meet our criteria on Restrictiveness.

J.3. Settings for Comparison between Human vs GPT-4

We invite 3 experts with data analysis skills to write 54 questions associated with 9 CSV files. For each file, 3 questions with single concept and 3 with multiple concepts are written. Combined with questions from GPT-4 on the same files, we form a dataset with 102 questions for assessment. Additionally, for difficulty, we randomly sample 216 question pairs on human-made and GPT-4 generated questions. All questions are anonymous to experts in this stage.

K. Method for Assessing Difficulty

For the difficulty assessment of DAEval, we randomly sample 80 groups from the dataset, each comprising 5 questions. GPT-4 ranks the questions within each group according to their difficulty, based on criteria such as data processing complexity, domain-specific knowledge, and coding sophistication. This ranking process is repeated 100 times for each group to ensure the robustness of the results. Each question has a score based on its ranking, with the most challenging question in a group earning 5 points and the least challenging 0 points. An average score was then calculated for each question, leading to a numerical difficulty ranking. These rankings allowed for the classification of questions into three levels: easy (0 to 1.667), medium (1.667 to 3.335), and hard (3.335 to 5).

L. Statistics on DAInstruct

DAInstruct includes 5131 data samples involved 631 CSV files, with the average length of ReAct conversations being 5.18.

M. Prompts for DAInstruct Construction

M.1. Question Generation

Table 18 lists keywords and prompts used for generating questions.

M.2. Filtering

Table 19 lists prompts used for filtering.

M.3. Content Moderation

Table 20 lists prompts used for content moderation.

N. Instruction-tuning data collection

In our data collection method, GPT-4 engages iteratively in problem-solving with our agent framework. The conversation begins with presenting a CSV file and an associated question. GPT-4 processes this and proposes code solutions, which are then executed. The results of the execution are fed back into the conversation, enabling GPT-4 to refine its responses and propose further iterations. This iterative cycle continues until GPT-4 determines that the accumulated information suffices to conclusively answer the problem. A critical aspect of this process is the design of the prompts, which directs GPT-4's outputs towards effective problem-solving. Table 21 lists prompts used in the process. To counter the challenge of hallucination, where GPT-4 might generate fictitious results, explicit instructions are included in the prompt to prevent GPT-4 from executing the code.

O. Case Study on DAInstruct

Table 22 shows a case study on DAInstruct, highlighting a dialogue between GPT-4 and the user, which poses questions and executes generated code. This iterative conversation continues until GPT-4 reaches a conclusive solution.

P. Implementation Details for Evaluation

Full prompts for evaluation can be found in Appendix P.1. We use accuracy as the metric, which is the proportion of questions for which all sub-questions are answered correctly. We use regular expression matching to draw the answer enclosed in "answer_name[answer]" and the exact match to evaluate the performance. We set temperature 0.2, top_p 1.0 with nucleus sampling and frequency penalty 0.0 for all the models in the experiments. All instances of GPT-3.5 and GPT-4 used in this paper, unless specifically indicated otherwise, are referred to as `gpt-3.5-turbo-0613` and `gpt-4-0613`[3] respectively.

P.1. Prompts for Evaluation

Table 23 shows standard prompt templates for evaluation. In addition, we slightly modify the standard templates for Abab5.5-Chat to make its output meet the format requirements of the agent framework. Specifically, we add the following prompt: "Note that you must output the string 'I now know the final answer' before the final answer.".

Q. Details for the Reformatting

In this section, we conduct ablation studies to demonstrate the necessity of adding a reformatting step after models' original responses, since most models have difficulties in strictly following the format requirements, while it is crucial for a closed-form evaluation. In addition, we conduct manual check to ensure this reformatting step doesn't effect the evaluation results by introducing external errors. We use `gpt-3.5-turbo-16k-0613` on reformatting. Prompts for reformatting are in Appendix Q.3.

Q.1. Ablation Studies

We add an additional reformatting step with GPT-3.5 to transform original responses into required formats. We conduct ablation studies in two settings: 1) w/o reformatting: directly prompt models with question, constraints and format requirements at one stage 2) w/ reformatting: first prompt models with question and constraints, and then use GPT-3.5 to reformat the original responses, with GPT-4, Qwen-72B-Chat, Qwen-14B-Chat and Mixtral-7B-Instruct-v0.2. Table 24 shows the results that even the performance of most powerful models like GPT-4 drops about 6%, and weaker models like Mistral-7B drops up to 32%, if prompted without reformatting, indicating the necessity of adding a reformatting step for evaluation.

Q.2. Manual Check

We randomly sample 50 questions with corresponding original responses from models as well as responses processed by reformatting step and manually check if this step could introduce extra errors for evaluation. We found GPT-3.5 can reformat the responses into required formats in all samples, which means this extra step usually will not effect the results.

Q.3. Prompts for reformatting

Table 25 shows prompts used in reformatting step.

---

[3] https://platform.openai.com/docs/models

Model	# Params. (in B)	Accuracy (%)
gpt-4-0613 (OpenAI, 2023b)	/	78.99
autogen[†] (gpt-4-0613) (Wu et al., 2023a)	/	71.48
daagent-34b	34	64.59
abab5.5-chat (MiniMax, 2023)	/	61.48
gpt-3.5-turbo-0613 (OpenAI, 2023a)	/	60.70
qwen-72b-chat (Bai et al., 2023)	72	59.92
gemini-pro (Team et al., 2023)	/	56.42
daagent-13b	13	55.86
qwen-agent[†] (qwen-72b-chat) (Bai et al., 2023)	72	55.25
mixtral-8x7b-instruct-v0.1[*] (Mistral.ai, 2023)	46.7 (12.9)	52.92
daagent-7b	7	51.36
xagent[†] (gpt-4-0613) (Team, 2023b)	/	47.52
claude-2.1 (Anthropic, 2023)	/	47.47
deepseek-coder-33b-instruct (DeepSeek, 2023)	33	46.09
phind-codellama-34b-v2 (Phind, 2023)	34	43.87
xwincoder-34b (Team, 2023c)	34	41.25
mistral-7b-instruct-v0.2 (Mistral.ai, 2023)	7	38.67
yi-34b (01ai, 2023)	34	38.34
qwen-14b-chat (Bai et al., 2023)	14	37.50
vicuna-13b (Chiang et al., 2023)	13	35.83
agentlm-70b (Zeng et al., 2023)	70	33.46
codellama-34b-instruct (Rozière et al., 2023)	34	31.13
vicuna-13b-v1.5 (Chiang et al., 2023)	13	27.31
qwen-7b-chat (Bai et al., 2023)	7	27.27
vicuna-7b (Chiang et al., 2023)	7	26.95
codellama-13b-instruct (Rozière et al., 2023)	13	26.67
codellama-7b-instruct (Rozière et al., 2023)	7	24.61
internlm-chat-20b (Team, 2023a)	20	24.51
agentlm-13b (Zeng et al., 2023)	13	24.31
wizard-coder-13b (Luo et al., 2023)	13	23.14
wizard-coder-7b (Luo et al., 2023)	7	22.83
xwincoder-13b (Team, 2023c)	13	19.69
agentlm-7b (Zeng et al., 2023)	7	17.46
chatglm3-6b (Zeng et al., 2022)	6	18.10
xwincoder-7b (Team, 2023c)	7	10.71
baichuan2-13b-chat (Yang et al., 2023a)	13	9.34
baichuan2-7b-chat (Yang et al., 2023a)	7	8.95

Table 7. **Leaderboard of InfiAgent-DABench.** Our DAAgent models are marked bold. [*]Mixtral is a model utilizing sparse mixture-of-experts (SMoE) architecture, with 46.7B total parameters but only uses 12.9B parameters per token. [†]All models except ready-made agent frameworks are equipped with our framework to interacter with files and code interpreter. We present LLMs those ready-made frameworks based in the experiments in parentheses.

Prompts for Description Generation

Here's the example of a detailed description of the table's header: "
The dataset contains 4599 entries and 5 columns. Here is a detailed description of each column:
1. Crime ID
Unique identifier for each crime.
Contains null values.
Data type: Object (String)
2. Month
The month when the crime was reported.
No missing values.
Data type: Object (String, in the format YYYY-MM)
3. Reported by
The police force that the crime was reported to.
No missing values.
Data type: Object (String)
4. Falls within
The higher-level police force that the crime report falls within. No missing values.
Data type: Object (String)
5. Longitude
The longitude where the crime occurred.
Contains null values.
Data type: Float64".
This is the information of one table:
{info}
{head}.
Please provide a detailed description of this table's header.

Table 8. Prompts for description generation.

Prompts for Question Generation

You are now a teacher responsible for creating data analysis questions. Please provide 3 possible question prompts based on the given scenario information. Ensure that the answers to these questions only require information within the given scenario and need to be solved using Python code. In data analysis, there are several key concepts: {key_concepts}. Please provide questions that are {complexity}. You do not need to answer the questions themselves. Scenario: You have a file {file_name}. Here is a detailed description of each column (head) in the provided table: {file_description}. Please use \Question{question} to wrap each of the 3 questions separately. Please use \Concepts{concepts} to point out which concepts this question wants to test.

Table 9. Prompts for question generation.

Prompts for Constrains and Format Generation

Please keep questions unchanged but

1) Add more constraints to the question to make sure the question has single fixed answer. Specify the requirements completely and precisely, so that the students don’t need to do any assumption or determine the statistical methods or parameters by themselves. Please use \Constraints{constraints} to wrap all the constraints.

2) Specify the desired output format requirements for the questions to make sure the answer can be easily checked. For a single answer, use format like @answer_name[answer] for an easy parsing. You can define the ”answer_name” according to the specific question. ”answer” should have a given format and a clear domain of value. Please use \Format{format} to wrap all the output requirements mentioned above. Some good demonstrations for constraints and format are: “‘

\Constraints{{ Calculate the Pearson correlation coefficient (r) to assess the strength and direction of the linear relationship between danceability and energy. Assess the significance of the correlation using a two-tailed test with a significance level (alpha) of 0.05. Report the p-value associated with the correlation test. Consider the relationship to be linear if the p-value is less than 0.05 and the absolute value of r is greater than or equal to 0.5. Consider the relationship to be nonlinear if the p-value is less than 0.05 and the absolute value of r is less than 0.5. If the p-value is greater than or equal to 0.05, report that there is no significant correlation. }}

\Format{{ @correlation_coefficient[r_value] @p_value[p_value] @relationship_type[relationship_type] where ”r_value” is a number between -1 and 1, rounded to two decimal places. where ”p_value” is a number between 0 and 1, rounded to four decimal places. where ”relationship_type” is a string that can either be ”linear”, ”nonlinear”, or ”none” based on the conditions specified in the constraints. }} “‘.

The constraints and formats can be adjusted according to the specific problem to ensure diversity, but strictly use correct symbol as the demonstration like @answer_name[answer]. The answer should be: \Question{question} \Concepts{concepts} \Constraints{constraints} \Format{format}.

Table 10. Prompts for constrains and format requirements generation.

Concepts and corresponding explanations
Concept: Summary Statistics **Explanation**: To start, calculate key summary statistics such as mean, median, standard deviation, minimum, maximum, etc., to obtain an overview of the data. These statistics help understand the central tendencies and distribution range of the data.
Concept: Distribution Analysis **Explanation**: Explore the distribution of the data, including checking if it adheres to a normal distribution, exhibits skewness, heavy tails, or bimodality. This helps in selecting appropriate statistical methods and models.
Concept: Correlation Analysis **Explanation**: Analyze the correlations between various variables. This assists in determining linear or nonlinear relationships between variables.
Concept: Outlier Detection **Explanation**: Identify and deal with outliers, as they can potentially disrupt data analysis and modeling. Methods such as box plots, Z-scores, or specialized outlier detection algorithms can be employed for outlier identification.
Concept: Comprehensive Data Preprocessing **Explanation**: Comprehensive data preprocessing is a fundamental step in the data analysis workflow, encompassing data cleaning, transformation, and the handling of missing values. It begins with data cleaning, a process focused on ensuring the accuracy and consistency of the data by identifying and rectifying errors, duplications, and inconsistencies. In tandem, data transformation adjusts the data's format and structure, which includes normalization, encoding categorical variables, and generating derived features that better represent the underlying phenomena for analysis. Integral to this preprocessing stage is the management of missing values, which may involve strategies such as deletion, imputation, or interpolation, depending on the nature and extent of the missing data.
Concept: Feature Engineering **Explanation**: New features can be generated or existing ones transformed to extract more information or improve model performance.
Concept: Machine Learning **Explanation**: Harness algorithms to classify data into categories, make predictions through regression, discover hidden patterns using clustering techniques, and even uncover insights from time series data. Explore the fundamentals of model training, evaluation, and practical applications, enabling to extract valuable information and make data-driven decisions across a wide range of analytical tasks.

Table 11. Concepts and corresponding explanations concluded from the expert interview.

Category	**Description**
Finance and Economics	Covers financial market trends, economic indicators, real estate prices, and investment analysis, essential for understanding economic conditions and market dynamics.
Health and Medical	Includes health insurance, disease statistics, hospital performance, and pharmaceutical research, key to evaluating public health policies and medical services effectiveness.
Demographics and Social Science	Comprises education levels, employment status, demographics, and social welfare, fundamental for analyzing social structures and social science issues.
Marketing and Consumer Behavior	Focuses on market segmentation, consumer preferences, and purchasing behavior, providing support for market strategy formulation and consumer behavior research.
Energy and Environmental Monitoring	Contains energy consumption, environmental monitoring, and meteorological information, crucial for resource management, environmental policy, and climate change research.
Transportation, Logistics, and Tourism	Involves traffic management, logistics efficiency, and tourism statistics, key to improving the efficiency of transportation and logistics as well as the management of the tourism industry.
Culture, Entertainment, and Media	Includes music, gaming, media, and cultural activities, used to analyze trends in the entertainment and cultural industries.
Scientific Research and Technology	Covers a broad range of scientific research and technological development, from genetics to engineering, suitable for scientific discovery and technological advancement.
Other Categories	Includes data analysis, machine learning, statistical models, specialized studies, and datasets that do not fall under any of the above categories, applicable to a variety of interdisciplinary data analysis and research work.

Table 12. Domains and corresponding descriptions.

Response Model	GPT-4	GPT-3.5	DAAgent-34B	Avg.
Consistency (%)	66.4	68.9	65.5	66.9

Table 13. **Consistency rate between human experts and GPT-4 on the evaluation of open-ended questions.** We gather responses from GPT-4, GPT-3.5 and DAAgent-34B on solving open-ended data analysis problems. We ask both human experts and GPT-4 to rate responses based on the same criteria and calculate the percentage of rating both sides reach consensus on.

Generated questions for test_ave.csv

Question: Calculate the mean fare paid by the passengers.
Concepts: Summary Statistics
Constraints: Calculate the mean fare using Python's built-in statistics module or appropriate statistical method in pandas. Rounding off the answer to two decimal places.
Format: @mean_fare[mean_fare_value] where "mean_fare_value" is a floating-point number rounded to two decimal places.
Labels: @mean_fare[34.65]

Question: Apply the linear regression algorithm from the sklearn library to predict whether a passenger survived or not based on the features 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', and 'Embarked'. Encode 'Sex' and 'Embarked' to numerical values before applying the model. Split the dataset into a training set (80%) and a testing set (20%), train the model on the training set, and evaluate its performance on the testing set using the accuracy score. Ensure that the train_test_split function's random_state parameter is set to 42 for consistency.
Concepts: Machine Learning
Constraints: Use one-hot encoding for the 'Sex' and 'Embarked' features. Use the "linear regression" model provided by the sklearn library in Python.
Format: @prediction_accuracy[accuracy], where "accuracy" is a float number rounded to 2 decimal places and has a range of 0.0 to 1.0.
Labels: @prediction_accuracy[0.78]

Question: Generate a new feature called "FamilySize" by summing the "SibSp" and "Parch" columns. Then, calculate the Pearson correlation coefficient (r) between the "FamilySize" and "Fare" columns.
Concepts: Feature Engineering, Correlation Analysis
Constraints: Create a new column 'FamilySize' that is the sum of 'SibSp' and 'Parch' for each row. Calculate the Pearson correlation coefficient between 'FamilySize' and 'Fare' Do not perform any further data cleaning or preprocessing steps before calculating the correlation.
Format: @correlation_coefficient[r_value] where "r_value" is the Pearson correlation coefficient between 'FamilySize' and 'Fare', a number between -1 and 1, rounded to two decimal places.
Labels: @correlation_coefficient[0.21]

Question: Create a new column called "AgeGroup" that categorizes the passengers into four age groups: 'Child' (0-12 years old), 'Teenager' (13-19 years old), 'Adult' (20-59 years old), and 'Elderly' (60 years old and above). Then, calculate the mean fare for each age group.
Concepts: Feature Engineering, Summary Statistics
Constraints: Make sure to round the mean fare of each group to 2 decimal places.
Format: @mean_fare_child[mean_fare], @mean_fare_teenager[mean_fare], @mean_fare_adult[mean_fare], @mean_fare_elderly[mean_fare], where "mean_fare" is a float number rounded to 2 decimal places.
Labels: @mean_fare_elderly[43.47], @mean_fare_teenager[31.98], @mean_fare_child[31.09], @mean_fare_adult[35.17]

Question: Perform a distribution analysis on the 'Fare' column for each passenger class ('Pclass') separately. Calculate the mean, median, and standard deviation of the fare for each class. Interpret the results in terms of the different passenger classes.
Concepts: Distribution Analysis, Summary Statistics
Constraints: Keep all numerical values rounded to 2 decimal points. The population standard deviation should be calculated.
Format: @mean_fare_class1[mean_fare], @median_fare_class1[median_fare], @std_dev_fare_class1[std_dev], @mean_fare_class2[mean_fare], @median_fare_class2[median_fare], @std_dev_fare_class2[std_dev], @mean_fare_class3[mean_fare], @median_fare_class3[median_fare], @std_dev_fare_class3[std_dev], where "mean_fare", "median_fare", and "std_dev" are statistical measures in float format rounded to 2 decimal places.
Labels: @median_fare_class1[69.30], @median_fare_class2[15.05], @std_dev_fare_class1[80.86], @mean_fare_class3[13.23], @std_dev_fare_class2[13.19], @mean_fare_class2[21.47], @std_dev_fare_class3[10.04], @mean_fare_class1[87.96]

Table 14. **Extended case study on generated questions.** All "@answer_name[answer]" in **Format** and **Labels** are marked in red.

Category	Question and Answer
File	**1.1** (Suitableness): Does the csv file represent a meaningful scenario for data analysis? (Yes/No, reasons) **1.2**: Is the file's *Category* correct? (Yes/No, correct category) **1.3**: Does the file involve sensitive attributes or offensive content? (No/Yes, keywords)
Question	**2.1** (Reasonableness): Can *Question* express a data analysis problem naturally and reasonably? (Yes/No, reasons) **2.2** (Reasonableness): Are *Constraints* limitations on *Question*? (Yes/No, reasons) **2.3** (Reasonableness): Do *Constraints* match *Question*? (Yes/No, reasons) **2.4** (Reasonableness): Does *Format* specify value type and range for all "@answer_name[answer]"? (Yes/No, non-compliant items) **2.5** (Reasonableness): Does *Format* match *Question*/*Constraints*? (Yes/No, reasons) **2.6**: Do *Question*/*Constraints* involve sensitive content? (No/Yes, keywords) **2.7** (Value): Do *Question*/*Constraints* have value for data analysis? (Yes/No, reasons) **2.8**: Do *Question*/*Constraints*/*Format* match *Concepts*? (No/Yes, necessary concepts) **2.9** (Alignment): Does the table have required data columns for *Question*/*Constraints*? (Yes/No, missing columns) **2.10** (Alignment): Do data types and ranges in columns match *Question*/*Constraints*/*Format* requirements? (Yes/No, non-compliant columns)
Label	**3.1** (Correctness): Is *Label* correct? (Yes/No, incorrect "@answer_name[name]") **3.2** (Correctness): Does *Label* comply with *Format* requirements? (Yes/No, non-compliant "@answer_name[name]") **3.3** (Restrictiveness): Are *Question*/*Constraints*/*Format* sufficient for unique correct answers in *Label*? (Yes/No, reasons)

Table 15. **Questionnaire to experts for assessing data quality on DAEval.** Metrics (Suitableness, Reasonableness, Value, Alignment, Correctness, Restrictiveness) are marked in red, and dataset components (*Question*, *Format*, *Constraints*, *Concepts*, *Label*) are in italics. Human experts should give answers according to requirements in parentheses.

Question and Answer
1 (Reasonableness): Is the expression natural and reasonable, and is it a question around data analysis? (5 points: Very natural and reasonable, 1 point: Very unnatural and unreasonable) **2** (Value): Is it a valuable question? (5 points: Very valuable, 1 point: Not valuable at all) **3** (Restrictiveness): Can it strictly limit the answer to the question as unique? (5 points: Completely strict, 1 point: Not strict at all) **4** (Alignment): Can it align with the data? Can the data needed to solve the question be found in the table, and does it meet the requirements? (5 points: Completely aligned, 1 point: Not aligned at all) **5** (Difficulty): Among two questions (A and B), which one is more difficult? (A/B/Close)

Table 16. **Questionnaire to experts for comparing human-made and GPT-4 generated questions.** Metrics (Reasonableness, Value, Restrictiveness, Alignment, Difficulty) are marked in red. Assessors should rate each question according to requirements in parentheses.

Bad Cases
File: 2019-08_edges.csv **Question**: Build a machine learning model to predict the ”Type” based on the other columns in the dataset. **Constraints**: Split the dataset into 70% training and 30% testing. Use Random Forest Classifier for building the model. Use ”Source”, ”Target”, ”Weight”, ”lng_org”, ”lat_org”, ”lng_dest”, ”lat_dest”, and the newly created ”Distance” as features for training the model. Evaluate the model's performance using accuracy score, precision, recall, and F1 score. **Format**: @accuracy_score[accuracy], @precision[precision], @recall[recall], @f1_score[f1] where ”accuracy”, ”precision”, ”recall”, and ”f1” are the respective evaluation metrics for the model rounded to four decimal places. **Label**: @precision[1.0000], @recall[1.0000], @accuracy_score[1.0000], @f1_score[1.0000]
File: tr_eikon_eod_data.csv **Question**: Explore the distribution of the CBOE Volatility Index (.VIX) on the given dates. **Constraints**: Use the numpy histogram function to explore the distribution. Set the bin separator array to [10, 20, 30, 40, 50, 60, 70]. Only consider non-null values. **Format**: @VIX_bin_counts[bin_counts] where ”bin_counts” is an array of 6 integers representing the count of .VIX values in each bin. **Label**: @VIX_bin_counts[1623, 352, 89, 15, 0, 0]

Table 17. **Bad cases found from human assessment.** The questionable parts of the cases are marked in red.

Category	**Description**
Domain Keywords	Data Cleaning & Preprocessing, Data Exploration, Text Data Analysis, Queries, Data Visualization Recommendations, Ethics and Bias, Documentation & Communication, Data Integrity Analysis, Data Pattern Identification
Question Prompt	{csv_summary} Please come up with a data analysis question preferably in {domain keyword} (if no proper question you can think of, propose a general question) that can use python code to solve. The question is preferred to have a close-formed solution. Respond with the question ONLY.

Table 18. List of keywords and the prompt to generate questions.

Filtering Aspect	**Prompt**
Python Code Relevance	{conversation} Please read the dialogue carefully and judge whether the provided Python code is syntactically correct and relevant to the task.
System Notifications	{conversation} Assess if the dialogue accurately interprets system notifications and outputs in the Python sandbox environment.
Conciseness	{conversation} Evaluate if the response is concise and focused on the task objectives using the Python sandbox.
Sandbox Environment	{conversation} Verify if the dialogue indicates the use of a real Python sandbox environment.
Solution Provision	{conversation} Determine if the dialogue provides a clear solution to the problem using the Python sandbox environment.

Table 19. Prompts used for GPT-4 filtering with enhanced criteria.

Moderation Aspect	Prompt
Checking the CSV Data	{csv.head().to_string()} Do you think the above csv file contains any sensitive information including but not limited to Drug, Marijuana, Weeds, Cannabis, Violence, Sexual assault, Politics? Yes or no? If yes, what does it contain?
Checking the Conversation	{conversation} Do you think the above dialogue contains any sensitive information including but not limited to Drug, Marijuana, Weeds, Cannabis, Violence, Sexual assault, Politics? Yes or no? If yes, what does it contain?

Table 20. Prompts used for comprehensive content moderation.

Steps	Prompt
Initial Prompt	Answer the following questions as best you can. You have access to the following tools: python_code_sandbox[input]: this tool can help to run python script with python code as input. Use the following format: Question: the input question you must answer Thought: you should always think about what to do Action: the action to take, should be one of [python_code_sandbox] Action Input: the input to the action Observation: the result of the action ... (this Thought/Action/Action Input/Observation can repeat N times) #Thought: I now know the final answer Final Answer: the final answer to the original input question For all outputs in code, THE print() function MUST be called For example, when you need to call df.head(), please use print(df.head()) If you have any files outputted write them to "./output/" Be mindful that when creating a piece of code for execution in python_code_sandbox, DO NOT try to predict or complete its output. Cease generating further and await the execution outcome from python_code_sandbox. Please note that any text beginning with "[SYSTEM NOTIFICATION] Running the above tool ..." is an output from the python_code_sandbox execution. You should NEVER try to generate it. Begin! Question: User uploaded the following files: {filename} {question} Thought:
Reply Prompt	Observation: [SYSTEM NOTIFICATION] Running the above tool with the following response: {STDOUT} Thought:

Table 21. Prompts for data collection.

Speaker	**Conversation**
User	Answer the following questions as best you can. You have access to the following tools: python_code_sandbox[input]: this tool can help to run python script with python code as input. ... Question: User uploaded the following files: ['/mnt/ep_7.csv'] Compute the average number of lines per scene in the 'ep_7.csv' dataset. Thought:

GPT-4	To answer this question, I will need to
	1. Load the CSV into a pandas DataFrame
	2. Analyze the dataset to see what it looks like and identify which column contains the scene information
	3. Count how many lines belong to each scene
	4. Compute the average number of lines per scene
	...
	Action: python_code_sandbox
	Action Input:
	`import pandas as pd`
	`filename = "/mnt/ep_7.csv"`
	`dataframe = pd.read_csv(filename)`
	`print(dataframe.head())`
	Let's load the file and see what the data looks like.

User	Observation: [SYSTEM NOTIFICATION] Running the above tool with the following response: STDOUT: [Output of dataframe.head()] Thought:

GPT-4	We can see that the dataset is a DataFrame with columns 'Season', 'Episode', 'Scene', 'Character', and 'Line'. The lines of script are contained in the 'Line' column, and each line is associated with a scene number in the 'Scene' column. Now I can calculate the total number of lines for each scene.
	Action: python_code_sandbox
	Action Input:
	`scene_lines_counts = dataframe['Scene'].value_counts().values`
	`avg_lines_per_scene = scene_lines_counts.mean()`
	`print(avg_lines_per_scene)`
	Here, I'm counting the number of lines in each scene using the value_counts function on the 'Scene' column of the dataset. This will return a Series with the scene numbers as the index and the counts as the values. The values() function then gets the counts as a numpy array. The mean() function is then used to compute the average number of lines per scene by averaging the counts.

User	Observation: [SYSTEM NOTIFICATION] Running the above tool with the following response: STDOUT: 6.018867924528302 Thought:
GPT-4	I now know the final answer Final Answer: The average number of lines per scene in the 'ep_7.csv' dataset is approximately 6.02.

Table 22. Case study on DAInstruct.

Prompts for Evaluation

Answer the following questions as best you can. You have access to the following tools: {tool_description}. Use the following format: Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
#Thought: I now know the final answer
Final Answer: the final answer to the original input question
For all outputs in code, THE print() function MUST be called
For example, when you need to call df.head(), please use print(df.head())
If you have any files outputted write them to ”./output/”
Do not use things like plot.show() as it will not work instead write them out ”./output/”
Be mindful that when creating a piece of code for execution in {tool_names},
DO NOT try to predict or complete its output.
Cease generating further and await the execution outcome from {tool_names}.
Please note that any text beginning with
”[SYSTEM NOTIFICATION] Running the above tool ...” is an output from the {tool_names} execution.
You should NEVER try to generate it.
Here is an example of using the {tool_names}:
Action: python_code_sandbox
Action Input:
`print('hello world')`
Now Begin!
Question: {instruction}
Thought:

Table 23. Prompts for evaluation.

Model	**w/o reformatting**	**w/ reformatting**
gpt-4-0613	72.76	78.99
qwen-72b-chat	44.75	59.92
qwen-14b-chat	18.29	37.50
mistral-7b-instruct-v0.2	6.23	38.67

Table 24. **The ablation studies of models with and without reformatting.** We use GPT-4 as an evaluator and conduct performance comparison between several LLMs (also including GPT-4, which means GPT-4 evaluates itself in this setting).

Prompts for Reformatting

You should strictly follow the output requirements in the Format part. Hereŕe some examples: \Format{
@shapiro_wilk_statistic[test_statistic]
@shapiro_wilk_p_value[p_value]
where ”test_statistic” is a number between 0 and 1 representing the Shapiro-Wilk test statistic. Rounding off the answer to two decimal places.
where ”p_value” is a number between 0 and 1 representing the p-value from the Shapiro-Wilk test. Rounding off the answer to four decimal places.}
\Answer{
@shapiro_wilk_statistic[0.56]
@shapiro_wilk_p_value[0.0002]}
\Format{
@total_votes_outliers_num[outlier_num]
where ”outlier_num” is an integer representing the number of values considered outliers in the t́otal_votesćolumn.}
\Answer{
@total_votes_outliers[10]}.
Your answer should contain all the ”@answer_name[answer]” in the order mentioned, each ”answer” should be in the range of value as required.
The format requirements of this question is:
@median_tmax[”median_TMAX”]
@median_tmin[”median_TMIN”]
@range_tmax[”range_TMAX”]
@range_tmin[”range_TMIN”]
where ”median_TMAX” and ”median_TMIN” are the median temperatures for TMAX and TMIN observation types, respectively. They are numbers between -100 and 100, rounded to two decimal places.
where ”range_TMAX” and ”range_TMIN” are the range of temperatures for TMAX and TMIN observation types, respectively. They are numbers greater than 0. Please give your answer:

Table 25. Prompts for reformatting.