DataSage: Multi-agent Collaboration for Insight Discovery with External Knowledge Retrieval, Multi-role Debating, and Multi-path Reasoning

Xiaochuan Liu[1], **Yuanfeng Song**[1*], **Xiaoming Yin**[1*], **Xing Chen**[1]
[1]ByteDance, China

Abstract

In today's data-driven era, fully automated end-to-end data analytics, particularly insight discovery, is critical for discovering actionable insights that assist organizations in making effective decisions. With the rapid advancement of large language models (LLMs), LLM-driven agents have emerged as a promising paradigm for automating insight discovery. However, existing data insight agents remain limited in several key aspects, often failing to deliver satisfactory results due to: (1) insufficient utilization of domain knowledge, (2) shallow analytical depth, and (3) error-prone code generation. To address these issues, we propose DataSage, a novel multi-agent framework that incorporates three innovative features including external knowledge retrieval to enrich the analytical context, a multi-role debating mechanism to simulate diverse analytical perspectives and deepen analytical depth, and multi-path reasoning to improve the accuracy of the generated code and insights. Extensive experiments on InsightBench demonstrate that DataSage consistently outperforms existing data insight agents across all difficulty levels, improving by 7.5% and 13.9% respectively in the insight-level and summary-level metrics. It offers an effective solution for automated data insight discovery.

1 Introduction

Nowadays, data continues to grow in volume and complexity across domains (L'heureux et al., 2017; Najafabadi et al., 2015). The importance of data and data analysis cannot be overstated. Data serves as the foundation for informed decision-making, providing the raw information needed to understand complex situations. Through data analysis, the raw information is transformed into interpretable patterns and trends (Khan, 2024; Abdul-Azeez et al., 2024; Ibeh et al., 2024; Lu et al., 2026). Building on this, insight discovery (Sahu et al., 2025) goes a step further to uncover actionable insights that can drive innovation, guide strategic decisions, and improve outcomes. It empowers organizations to optimize operations, tailor strategies, and remain competitive in dynamic, high-stakes contexts (Steiner, 2022; Colson, 2019; McAfee et al., 2012).

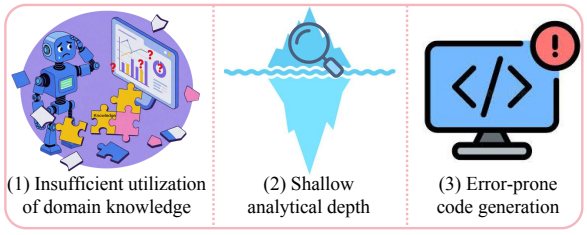


Figure 1: Illustration of three critical limitations in current data insight agents: (1) insufficient utilization of domain knowledge, (2) shallow analytical depth, and (3) error-prone code generation.

Traditional data analysis and insight discovery largely rely on manual efforts, which are extremely time-consuming (Bean, 2022; Arora and Malik, 2015). With the advancement of large language models (LLMs), LLM-driven agents have recently demonstrated the capability for automated insight discovery. Existing data insight agents (Pérez et al., 2025; Sahu et al., 2025; LangChain, 2024) commonly adopt a question-driven paradigm. These approaches typically first raise relevant analytical questions, then answer them using SQL or Python-based code execution, and finally summarize the results into coherent insights.

However, existing data insight agents are far from being as excellent as experienced data analysts. As shown in Figure 1, there are at least three critical limitations in current approaches, as detailed in Appendix A. (1) **Insufficient utilization of domain knowledge.** In real-world data analysis scenarios (Sahu et al., 2025; Majumder et al., 2025; Hu et al., 2024), datasets often originate from diverse domains such as industry, healthcare, re-

*Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 6216–6250
July 2-7, 2026

tail, etc., each characterized by domain-specific constraints and knowledge. Purely data-driven approaches (Pérez et al., 2025; Sahu et al., 2025) or models relying solely on internal knowledge of LLMs often fail to capture critical domain knowledge that is essential for producing accurate insights. (2) **Shallow analytical depth.** In question-driven approaches for insight discovery, the formulation of high-quality questions is as critical as answering them. Prior methods (Pérez et al., 2025; Sahu et al., 2025; Wu et al., 2024; Li et al., 2025) only rely on a single LLM to generate questions, which tends to result in shallow or overly generic questions, thereby limiting the depth and novelty of the generated insights. (3) **Error-prone code generation.** Code generation remains a challenging task for current LLMs (Huynh and Lin, 2025; Hong et al., 2024b). Relying on a single LLM to generate executable code (Pérez et al., 2025; Sahu et al., 2025) often leads to errors, which can result in incorrect or misleading insights.

To alleviate above-mentioned issues, we propose DataSage, a novel multi-agent framework that incorporates three innovative features: external knowledge retrieval, multi-role debating, and multi-path reasoning. DataSage is a modular multi-agent framework composed of four key modules that work in an iterative question-answering (QA) loop. **Dataset Description Module** provides the structured description of datasets. To address domain-specific challenges, **Retrieval-Augmented Knowledge Generation (RAKG) Module** dynamically retrieves and synthesizes external domain knowledge when internal knowledge of LLMs is insufficient. **Question Raising Module** formulates high-quality analytical questions through a divergent-convergent multi-role debating process, ensuring broad coverage and depth. These questions are passed to the **Insights Generation Module**, which translates questions into executable Python code through multi-path reasoning, interprets the outputs, and finally generates insights. Each core module adopts a multi-agent architecture to enhance specialization and collaboration. During iterative QA cycles, new questions are adaptively generated based on previous insight history, enabling deeper exploration. Finally, the generated insights are consolidated into a coherent summary. Extensive experiments on InsightBench (Sahu et al., 2025) demonstrate that DataSage consistently outperforms existing data agents across two metrics (i.e., achieving a 7.5% and 13.9% improvement at the insight-level and summary-level scores respectively) and all difficulty levels.

Our contributions can be summarized as follows:

- We propose DataSage, a novel multi-agent framework composed of four key modules for automated insight discovery task.
- DataSage incorporates three innovative features including external knowledge retrieval to enrich the analytical context, a multi-role debating mechanism to simulate diverse analytical perspectives and deepen analytical depth, and multi-path reasoning to improve the accuracy of the generated code and insights.
- We conduct comprehensive experiments that demonstrate DataSage consistently outperforms existing data insight agents, and is particularly well-suited for complex and high-difficulty tasks.

2 Related Work

Data Analytics Agents. Recent years have seen rapid progress in the development of LLM-based agents for general-purpose data analysis. Systems such as Code Interpreter (OpenAI, 2024a) and Pandas Agent (LangChain, 2024) allow users to analyze tabular data through natural language queries. Vacareanu et al. demonstrates that LLMs can also perform basic statistical modeling such as regression, enhancing their applicability in quantitative tasks. Additional studies (Cheng et al., 2023; Wang et al., 2023, 2024; Hong et al., 2025) have proposed end-to-end LLM-based frameworks that combine goal understanding, code generation, or visualization to support analytical workflows.

Insight Discovery Approaches. Insight discovery extends beyond basic data analysis by aiming to uncover meaningful, actionable insights from data. Early systems such as QuickInsights (Ding et al., 2019) and the template-based method by Law et al. focus on extracting insights using predefined logic and simple heuristics. These approaches are often limited to clean datasets with well-labeled columns. With the advent of LLMs, newer systems such as InsightPilot (Ma et al., 2023), OpenAI Data Analysis (OpenAI, 2024b), Langchain's Pandas Agent (LangChain, 2024), and HLI (Pérez et al., 2025) have emerged, which can generate code to present descriptive insights. These methods often specialize in narrow and single-step insight discovery tasks based on very concrete user

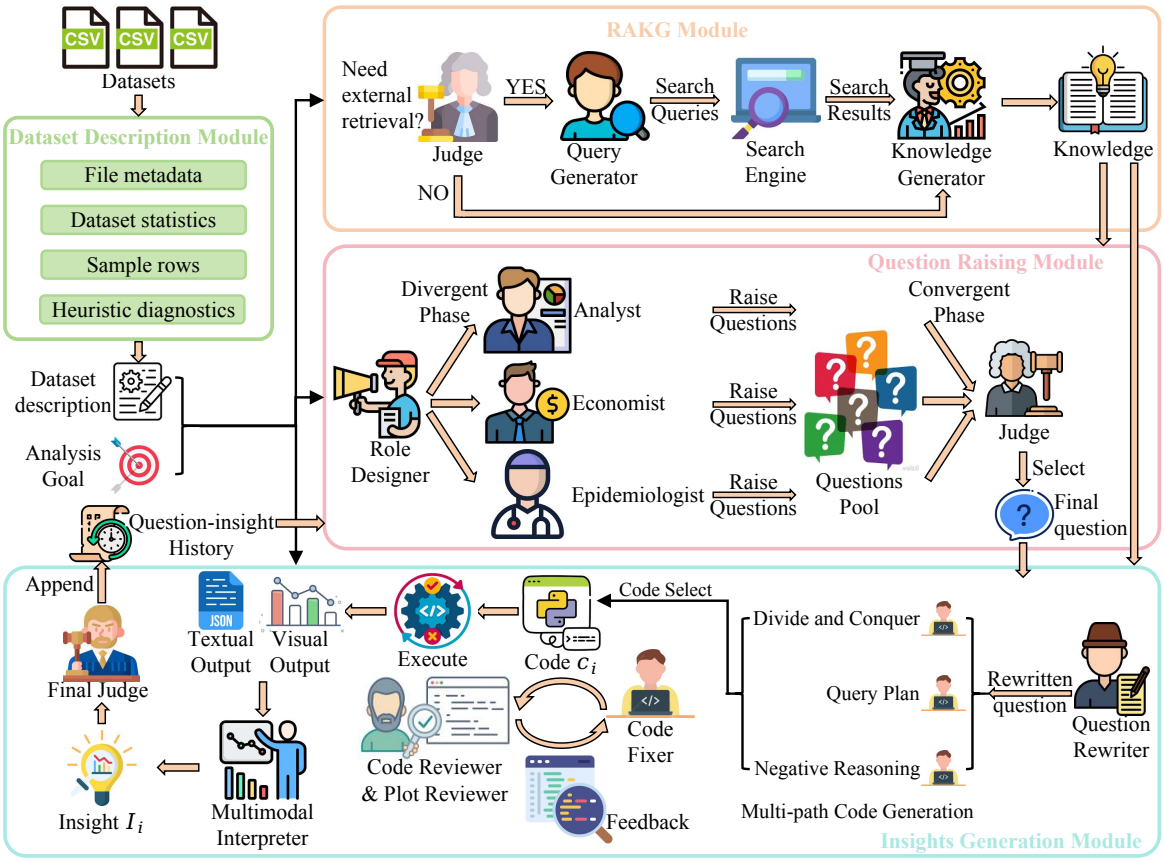


Figure 2: An illustration of DataSage, a multi-agent framework for insights discovery tasks. The framework consists of four key modules that work in an iterative QA loop. Dataset Description Module provides the structured description of datasets. RAKG Module retrieves and synthesizes external knowledge. Question Raising Module raises high-quality analytical questions through a divergent-convergent process. Finally, Insights Generation Module transforms questions into the final insights.

instructions. AgentPoirot (Sahu et al., 2025) extends this line of work toward multi-step insight generation, which achieves state-of-the-art performance on the InsightBench benchmark (Sahu et al., 2025). However, there are some persistent limitations in existing systems, such as insufficient utilization of domain knowledge, limited analytical depth, and error-prone code generation. We introduce DataSage, a multi-agent framework featuring external knowledge retrieval, multi-role debating, and multi-path reasoning, pushing beyond the boundaries of current LLM-based systems.

3 The DataSage Framework

3.1 Framework Overview

As illustrated in Figure 2, our framework adopts a question-driven paradigm to systematically generate insights from datasets. The framework consists of four key modules that work in an iterative QA loop with N_{iter} iterations. **Dataset Description Module** provides the structured description of datasets that serves as foundational context for downstream modules. **Retrieval-Augmented Knowledge Generation (RAKG) Module** dynamically retrieves and synthesizes external knowledge when internal knowledge of LLMs is insufficient. **Question Raising Module** formulates high-quality analytical questions through a divergent-convergent process, ensuring broad coverage and depth. **Insights Generation Module** transforms questions into executable Python code, rigorously validates the code, runs the code in the sandbox, interprets the output of the code, and finally produces insights. Each core module (except Dataset Description) employs a multi-agent architecture to enhance specialization. During iterative QA cycles, subsequent questions are dynamically raised based on prior question-insight history. Finally, all insights are consolidated into a coherent summary.

3.2 Dataset Description Module

To support rapid understanding of the datasets, we design a unified dataset description module, which can automatically extract essential metadata and detect potential issues in the datasets. Given the raw dataset $\mathbb{D}$, the module first collects basic file metadata including the filename, file size, and file type. Then, it computes comprehensive dataset statistics, including dimensionality (number of rows and columns), column-wise data types, and per-column missing value counts. If the dataset contains numeric columns, it further computes standard descriptive statistics such as mean, standard deviation, min/max, and quantiles. A representative sample of the first few rows is also extracted. The module also includes lightweight heuristic diagnostics, such as flagging missing values or duplicated rows. All the aforementioned information is organized into a structured JSON format, serving as the comprehensive dataset description D. This abstraction simplifies dataset onboarding, facilitates downstream understanding of the dataset, and enables consistent preprocessing pipelines across heterogeneous data sources.

3.3 Retrieval Augmented Knowledge Generation (RAKG) Module

To address domain-specific challenges, we equip our framework with an *on-demand* external retrieval mechanism to detect, retrieve, and synthesize external domain knowledge to support the whole analysis pipeline. Our RAKG module consists of the following four main stages.

Search Necessity Judgment. Given dataset description D and analysis goal G, a *judge* agent determines whether the analysis can be completed using only internal knowledge and the provided datasets.

$$\text{Judge}(D, G) \rightarrow \{\text{yes}, \text{no}\} \tag{1}$$

If the *judge* determines external retrieval is unnecessary, the module defaults to a *vanilla knowledge generator*, leveraging only the internal knowledge of LLMs. Otherwise, it proceeds to the next stage.

Search Query Generation. For cases where external retrieval is needed, a *data-aware query generator* formulates N_q high-quality, Google-ready search queries $\mathcal{Q}$. These queries are crafted to target vertical knowledge gaps identified in the judgment stage, with diversity maximized and redundancy minimized through prompt optimization.

$$\mathcal{Q} = \text{QueryGenerator}(D, G) \tag{2}$$

Search Execution. The module then performs real-time web retrieval using Google Search [1]. To ensure reproducibility, we impose a maximum date constraint on the search.

$$\mathcal{R} = \bigcup_{q_s \in \mathcal{Q}} \text{GoogleSearch}(q_s) \tag{3}$$

Knowledge Generation. A specialized *knowledge generator* processes the search results $\mathcal{R}$ and produces structured, domain-relevant knowledge items K. These knowledge items are then made available to downstream modules, such as Question Raising and Question Rewriting.

$$K = \text{KnowledgeGenerator}(\mathcal{R}, D, G) \tag{4}$$

3.4 Question Raising Module

Inspired by cognitive theories of divergent–convergent thinking (Lu et al., 2024), we propose a structured Question Raising Module that adopts a multi-agent, role-driven framework to simulate diverse analytical perspectives and deepen analytical depth. The design of the module embraces a divergent-convergent paradigm: agents first explore broadly from distinct perspectives (divergent phase) and then select the most promising questions (convergent phase). The following are the three key stages in the module.

Data-Derived Role Generation. Given dataset description D, analysis goal G and knowledge K, we prompt a system-level *role designer* agent to generate N_R diverse and well-specified analytical roles. Each role is defined by attributes such as its background (e.g., behavioral analyst, anomaly detector), domain focus, personality traits (e.g., skeptical, risk-seeking), and analytical capabilities. The *role designer* ensures that the roles align with the datasets and analysis goal, and aims to maximize coverage over the question space.

$$\{\text{Role}_1, \ldots, \text{Role}_{N_R}\} = \text{RoleDesigner}(D, G, K) \tag{5}$$

Multi-Role Question Raising. Each generated role then independently explores the data and poses a set of questions aligned with its unique perspective. Roles may specialize in temporal dynamics, user behavior, value distribution, or rare event detection, among others. This results in a pool of

[1] Serper API: https://serper.dev/ .

questions $\mathbb{Q}_j$ (in the j-th iteration) that span a wide variety of analytical angles, ranging from descriptive to causal, comparative to behavioral.

$$\mathbb{Q}_j = \bigcup_{1 \le i \le N_R} \text{Role}_\text{i}(D, G, K, H), \quad (6)$$

where H represents the history of previous question-insight pairs.

Question Convergence. The convergent phase is introduced to select the most promising questions in the question pool $\mathbb{Q}_j$. A global *judge* agent selects a subset of high-quality questions $\mathbb{Q}_j^*$ based on the following criteria: (1) Potential to yield non-trivial or surprising insights; (2) Alignment with datasets and analysis goal; (3) Diversity across question types and dimensions; (4) Complementarity with already answered questions. Each selected question is annotated with its source role and a justification for selection, making the reasoning process transparent and interpretable.

$$\mathbb{Q}_j^* = \text{Judge}(\mathbb{Q}_j, D, G, K, H) \subseteq \mathbb{Q}_j \quad (7)$$

3.5 Insights Generation Module

Once a question $q \in \mathbb{Q}_j^*$ is proposed by the Question Raising Module, the Insights Generation Module takes over to generate the corresponding insight. Naively answering natural language questions with code generation often leads to failures due to ambiguity (Zhou et al., 2023), schema mismatch (Wang and Liu, 2025), or unvalidated code logic (Cen et al., 2025). To ensure the quality of the generated insight, we decompose the insight generation into a structured pipeline that includes Question Clarification, Multi-path Code Generation and Refinement, Multimodal Insight Interpretation, and Final Decision.

Question Clarification. To reduce ambiguity in questions and enhance code generation accuracy, we first employ a *schema-aware question rewriter*. The *rewriter* identifies missing table and column references, unclear aggregation goals, temporal dimensions, or ambiguous variable names, and resolves these issues by replacing ambiguous entities with concrete schema fields, thereby reformulating the question into a fully grounded, self-contained analytical question q^*.

$$q^* = \text{QuestionRewriter}(q, D, G, K) \quad (8)$$

Multi-path Code Generation. To increase the likelihood of generating correct code, we adopt a multi-path code generation strategy. Inspired by the success of Chain-of-Thought (CoT) prompting (Wei et al., 2022) in improving the reasoning capabilities of large language models (LLMs), we adopt three complementary CoT reasoning strategies tailored for code generation: **Divide-and-Conquer**, **Query Plan**, and **Negative Reasoning**.

The Divide-and-Conquer Reasoning decomposes a complex problem into smaller sub-problems, solves each sub-problem independently, and then combines the individual solutions to obtain the final code c_{DaC}.

The Query Plan Reasoning first produces a query execution plan in natural language (detailing filters, joins, aggregations, and intermediate outputs) before translating the plan into executable code c_{QP}.

The Negative Reasoning anticipates potential mistakes before generating code (e.g., double counting, incorrect joins, handling of NULLs), explains how to avoid them, and only then generates code c_{NR} designed to mitigate those risks.

By leveraging these three reasoning paths, we generate a diverse pool of candidate code. Among the candidates, a *code selector* is employed to evaluate and select the most correct one as the initial code c_0 based on the execution results of the code.

$$c_0 = \text{CodeSelector}(q^*, D, G, c_{DaC}, c_{QP}, c_{NR}) \quad (9)$$

Code Refinement. To ensure correctness, the initial code c_0 is passed through a *code reviewer*. The *code reviewer* analyzes the code along four dimensions involves requirement alignment, schema compliance, operational risk, and data integrity.

$$r_{c_i} = \text{CodeReviewer}(q^*, D, G, c_i) \quad (10)$$

The generated code c_0 is also executed to produce a plot that visualizes the results of data analysis. However, the initial plot p_0 is often of low quality and difficult to interpret, due to issues such as overlapping text or poor layout. To improve the interpretability of these plots, both for human and downstream interpreter model, we introduce a *plot reviewer* that evaluates the generated plots.

$$r_{p_i} = \text{PlotReviewer}(q^*, D, G, p_i) \quad (11)$$

If issues are found (either by the *code reviewer* or the *plot reviewer*), the *code fixer* takes the original code c_0 and the reviewers' feedback $r_0 = \{r_{c_0}, r_{p_0}\}$ to produce a revised version c_1. This process may iterate several times, particularly for

complex or error-prone questions until no more mistakes are found or a predefined maximum number of iterations (N_{fix}) is reached.

$$c_i = \text{CodeFixer}(q^*, D, G, c_{i-1}, r_{i-1}), \quad \forall 1 \leq i \leq N_{\text{fix}}, \text{FAIL} \in r_{i-1}. \tag{12}$$

Multimodal Insight Interpretation and Final Decision. Once a code version c_i is obtained, it is executed in the sandbox. Then, the *interpreter* agent parses the generated visual/textual artifacts and generates the insight I_i. Previous methods (Pérez et al., 2025; Sahu et al., 2025) rely solely on textual information to generate insights. However, textual information alone is often insufficient for accurately explaining analytical results. In contrast, since we explicitly review and refine the generated plot in the preceding step, the resulting visualization is more interpretable. To this end, we are the first to use Multimodal Large Language Models (MLLMs) as the interpreter that jointly leverages textual and visual information when generating insights.

To avoid the possible introduction of new errors during the code refinement process, all intermediate versions of the code c_i and corresponding insights I_i are stored and passed to a *final judge*, which selects the most complete, valid, and interpretable insight I according to the history of code refinement. Finally, the $q - I$ pair is added to the history H.

$$\begin{aligned} o_i &= \text{SandboxExecute}(c_i), \\ I_i &= \text{Interpreter}(q, D, o_i), \\ I &= \text{FinalJudge}(q, D, G, \{(c_i, I_i)\}), \\ H &= H \cup \{(q, I)\}. \end{aligned} \tag{13}$$

4 Experiments

4.1 Experiment Setup

Dataset. We utilize the InsightBench (Sahu et al., 2025), a widely used benchmark for evaluating insight discovery in data analytics. It consists of 100 tabular datasets representing diverse business use cases, spanning three difficulty levels: easy, medium, and hard. Unlike other datasets that focus on more specific QA style data analysis tasks (Hu et al., 2024; Majumder et al., 2025), InsightBench evaluates agents on their ability to perform end-to-end data analytics, encompassing question formulation, answer interpretation, and insight generation.

Baselines. We compare our framework against baselines of three categories: 1) **LLM-only. GPT-4o only** is a direct prompting baseline where the dataset description D and analysis goal G are provided to GPT-4o without intermediate reasoning or tool use. The model is asked to directly generate the final set of insights. **GPT-4o domain** is a domain-aware GPT-4o baseline implemented by Zhang and Elhamod (2025) that first infers the dataset's domain, generates relevant domain knowledge, and then leverages this knowledge to produce the final insights. 2) **Single-agent Models. CodeGen** (Majumder et al., 2025) generates the entire code at one go to solve the task, where a demonstration of a solution code is provided in the context. Based on the execution result, it generates the insights and summarizes the workflow. **ReAct** (Yao et al., 2023) solves the task by generating thought and subsequent codes in a multi-turn fashion. 3) **Multi-agent Models. Data-to-Dashboard** (Zhang and Elhamod, 2025) is a modular multi-agent LLM system that automates end-to-end dashboard generation from tabular data. It integrates domain-aware reasoning with iterative self-reflection to produce insightful visualizations. We take its intermediate insight outputs as the results. **Pandas Agent** (LangChain, 2024) is a LangChain-based data science agent optimized for question answering. Given a data frame and a question, it generates and executes Python code to produce answers. **AgentPoirot** (Sahu et al., 2025) is the best baseline data analysis agent that adopts a question-driven paradigm in InsightBench.

Metrics. We follow the InsightBench (Sahu et al., 2025) evaluation protocol, which computes performance at two levels: (1) *Summary-level*, measuring the G-Eval (Liu et al., 2023) score between the generated and ground-truth summaries; and (2) *Insight-level*, matching each ground-truth insight with the most similar prediction and averaging their G-Eval scores. We discard ROUGE-1 due to its well-known limitations in capturing semantic similarity (Chen et al., 2024). We report the average of the summary-level and insight-level scores across datasets of different difficulty levels.

Implementation Details. To ensure a fair comparison across models, we use GPT-4o [2] as the base model (due to its popularity and stability) and the temperature is fixed to 0 for all baselines. We set the hyperparameters as follows: $N_q = 3$ (number of search queries in RAKG Module), $N_R = 3$ (number of diverse analytical roles), $N_{\text{fix}} = 5$ (maximum number of code-fix iterations), and $N_{\text{iter}} = 6$

[2] Version gpt-4o-2024-08-06

Model	Insight-level Scores (G-Eval)				Summary-level Scores (G-Eval)			
	Easy	Medium	Hard	Avg	Easy	Medium	Hard	Avg
LLM-only								
GPT-4o only	0.3157	0.2572	0.2702	0.2789	0.3622	0.3141	0.2923	0.3215
GPT-4o domain	0.3195	0.2608	0.2663	0.2802	0.3649	0.3144	0.3164	0.3302
Single-agent Models								
CodeGen	0.3268	0.2766	0.2563	0.2852	0.3424	0.3023	0.2773	0.3063
ReAct	0.3355	0.2953	0.2672	0.2984	0.3618	0.3239	0.2714	0.3185
Multi-agent Models								
Data-to-Dashboard	0.2191	0.2148	0.2368	0.2231	0.2463	0.2514	0.2615	0.2531
Pandas Agent	0.3479	0.2796	<u>0.3180</u>	0.3124	0.3589	0.3298	0.2998	0.3289
AgentPoirot	<u>0.3768</u>	<u>0.3044</u>	0.3117	<u>0.3284</u>	<u>0.3819</u>	<u>0.3595</u>	<u>0.3292</u>	<u>0.3565</u>
DataSage (Ours)	**0.4063** **+7.8%**	**0.3211** **+5.5%**	**0.3407** **+9.3%**	**0.3530*** **+7.5%**	**0.4448** **+16.5%**	**0.4047** **+12.6%**	**0.3710** **+12.7%**	**0.4059*** **+13.9%**

Table 1: Performance of different models on the InsightBench with GPT-4o as the base model. All baseline results reported are reproduced by us. The best and runner-up are in **bold** and <u>underlined</u>. Our framework consistently outperforms the best baseline across both metrics and all three difficulty levels. Improvements are calculated between the best to the runner-up. "*" indicates statistically significantly better than the corresponding strongest baseline with Paired t-test $p < 0.05$.

(number of Q-A loop iterations, consistent with the baseline AgentPoirot for fair comparison). The prompts and implementation details for each agent are provided in the Appendix D.

4.2 Main Results

We compare our framework with three types of baselines and present the results in Table 1. The key findings from the table are as follows: (1) **Consistent performance improvement with our framework.** Our framework consistently outperforms the best baseline across both metrics and all three difficulty levels. On Insight-level Scores, DataSage achieves an average improvement of 7.5%, while on Summary-level Scores the gain is even larger at 13.9%. (2) **Larger improvements on harder datasets.** The performance improvement is more pronounced on more challenging tasks. On Insight-level Scores, our framework improves over the best baseline by 7.8%, 5.5%, and 9.3% for Easy, Medium, and Hard datasets, respectively. This demonstrates that our framework is particularly well-suited for complex, high-difficulty tasks, while still providing improvements on simpler datasets. (3) **More substantial gains on Summary-level Scores.** The improvement of DataSage on Summary-level Scores is notably larger than that on Insight-level Scores, despite no explicit optimization for summary generation. This suggests that producing more accurate insights enables the model to focus on the correct information during summarization, thereby improving summary quality. (4) **Advantages of multi-agent models.** Across the three baseline categories, multi-agent models achieve the best performance (except Data-to-Dashboard, which does not fully address the insight discovery task), followed by single-agent models, with LLM-only approaches performing the worst. This aligns with intuition and highlights the inherent strengths of multi-agent designs. Our carefully designed DataSage further amplifies these advantages, achieving the highest performance. (5) **Effectiveness of domain-aware prompting.** *GPT-4o domain* slightly outperforms *GPT-4o only*, indicating that even a simple domain-aware design can enhance performance. This underscores the importance of domain-specific knowledge for this task and further validates the rationale behind the RAKG module in our framework. More experimental results for additional base models and benchmarks are provided in Appendix B.

4.3 Quality of Plots

Beyond producing superior insights, DataSage can also generate plots with higher quality. To demonstrate this, we compare DataSage against the best baseline AgentPoirot, as well as two ablated variants that remove either the *Code Refinement* or

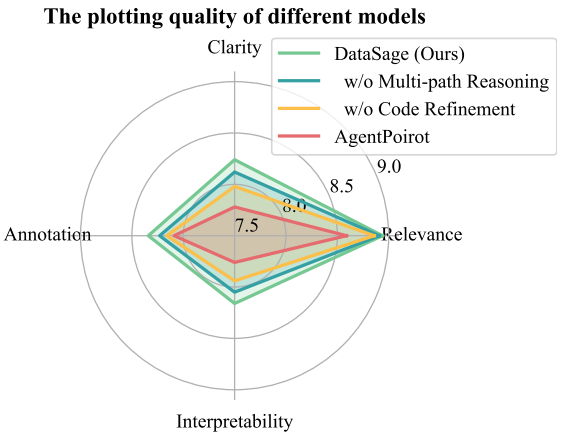


Figure 3: Comparison of the plot quality across different models. DataSage can generate noticeably higher-quality plots than the baseline AgentPoirot, attributing to our design of the Insights Generation Module.

Multi-path Reasoning component. To enable large-scale and consistent evaluation, we employ GPT-4o as a judge to score the generated plot for each raised question along four dimensions: (1) **Relevance**: whether the plot accurately addresses the raised question; (2) **Clarity**: whether the visualization is clean, legible, and free of unnecessary clutter; (3) **Annotation**: whether axis labels, titles, legends, and color usage are correct, informative, distinct, and accessible; and (4) **Interpretability**: whether a viewer can readily identify and articulate the key takeaway of the plot. Each criterion is rated on a 0–10 scale. If the model fails to generate a plot, it is assigned a score of 0.

As shown in Figure 3, DataSage consistently outperforms the baseline by a substantial margin across all four dimensions. The ablation results confirm that both *Code Refinement* and *Multi-path Reasoning* contribute to this improvement, with *Code Refinement* having a more pronounced effect. It demonstrates that the design of our Insights Generation Module can not only enhance the correctness of generated code but also improve the quality of the resulting plots. A more detailed case study can be found in Appendix C.2. DataSage can produce higher-quality plots that not only enable downstream *interpreter* agent to generate insights more accurately but also facilitate human comprehension. It enhances both the utility and the trustworthiness of our framework, empowering users to make informed decisions more effectively.

Model	G-Eval	
	Insight-level	Summary-level
DataSage	**0.3530**	**0.4059**
w/o QR	0.3475	0.4019
w/o MR	0.3417	0.3948
w/o RAKG	0.3316	0.3982

Table 2: Ablation study on the three key components of our framework: Retrieval-Augmented Knowledge Generation (RAKG), Question Raising (QR), and Multi-path Reasoning (MR). The best results are in **bold**. Removing any component degrades performance, with RAKG contributing most to the overall performance.

4.4 Ablation Study

To validate the effectiveness of each proposed component in DataSage, we conduct an ablation study by removing one of the three key modules, Retrieval-Augmented Knowledge Generation (RAKG), Question Raising (QR), and Multi-path Reasoning (MR), while keeping the rest of the framework unchanged.

As shown in Table 2, ablating any component results in a performance drop over all the metrics. This indicates that all the proposed components contribute to the superior performance of DataSage, validating the rationality of our framework design. Removing RAKG results in the most substantial degradation in overall performance (the average of the two metrics). This suggests that grounding the analysis with external knowledge is crucial for generating deep and accurate insights. Overall, the ablation study confirms that all three components are indispensable and complementary. Their combined effect leads to the highest performance. More detailed ablation analysis is provided in Appendix B. Additionally, the case study is presented in Appendix C.

4.5 Human Evaluation

Ours vs. Baseline	Win	Tie	Lose
DataSage vs. AgentPoirot	61%	11%	28%

Table 3: Human evaluation results on 25 randomly sampled cases from InsightBench using pairwise comparison. Human annotators overwhelmingly prefer the outputs of our framework over the strongest baseline.

To more accurately assess the validity of the generated insights and to add much-needed credibility

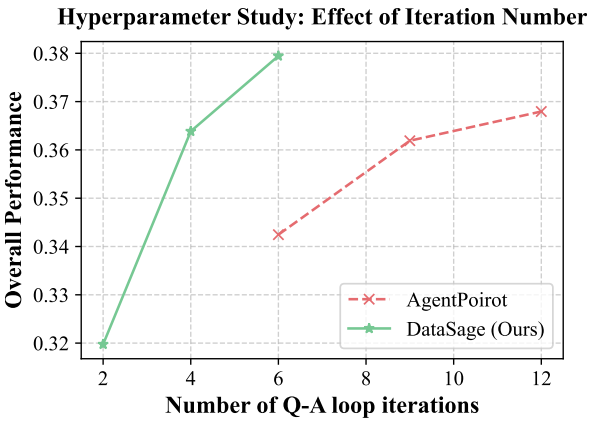


Figure 4: Hyperparameter study on N_{iter} (the number of Q-A loop iterations). Both models generally benefit from more iterations. DataSage achieves competitive or superior performance with significantly fewer iterations compared to the baseline.

to the capabilities of our framework, we conduct an additional human evaluation on 25 randomly sampled instances in InsightBench, assessed by four human annotators. We adopt a pairwise comparison. Specifically, we pair the outputs of our framework with those of the strongest baseline, shuffle their order, and instruct human annotators to select the better one. We calculate the percentage of winning by our framework (Win-Rate). As shown in Table 3, in human evaluation, our framework also consistently outperforms the strongest baseline (Win-Rate > 50%), which aligns with our LLM-based evaluation results.

4.6 Hyperparameter Study

We further investigate the impact of the most critical hyperparameter, N_{iter}, which controls the number of Q-A loop iterations of the framework. As shown in Figure 4, we report the overall performance of our framework DataSage and the baseline AgentPoirot under different values of N_{iter}. For both methods, performance generally improves as the number of iterations increases, which suggests that more iterations enable the framework to better discovery insights. However, DataSage achieves comparable or even superior results with significantly fewer iterations. In particular, DataSage already surpasses the baseline with 9 iterations when using only 4 iterations. This demonstrates that our design is not only more effective but also more efficient, requiring fewer iterations to reach strong performance. Overall, the results confirm that increasing the iteration budget is beneficial, but the gains quickly saturate for baseline model. In contrast, DataSage leverages its structured design to extract higher-quality insights with substantially fewer iterations, highlighting the efficiency advantage of our framework.

5 Conclusion

In this paper, we propose DataSage, which demonstrates the potential of a multi-agent framework to significantly enhance data insight discovery by addressing key limitations of existing data agent systems through external knowledge retrieval, multi-role debating, and multi-path reasoning. Our approach not only improves the accuracy and reliability of insights but also offers a scalable and flexible solution for automated data analysis. Future work will focus on further optimizing the framework, designing other agent architectures, and exploring additional applications in diverse domains.

Limitations

While our proposed framework achieves state-of-the-art performance on the insight discovery task, several limitations remain.

First, despite outperforming existing baselines, there is still a gap between our framework and experienced data analysts in terms of analytical depth, contextual understanding, and the ability to draw nuanced insights. We view this as an important direction for future work and plan to further enhance the reasoning capabilities and domain alignment of the framework.

Second, DataSage is primarily designed and evaluated for insight discovery tasks. Although this task is representative of many real-world analytical scenarios, it only covers a subset of the broader data analysis landscape. In future work, we aim to extend our framework to support a wider range of analytical tasks, exploring additional applications across diverse domains and data modalities.

Third, DataSage is particularly well-suited for complex and high-difficulty analytical problems, where multi-path reasoning and external knowledge integration are crucial. However, for relatively simple tasks, some of the modules may be redundant and could introduce unnecessary computational or interpretive overhead. As a result, we plan to investigate adaptive mechanisms that can dynamically tailor the analysis pipeline based on task difficulty, thereby improving overall efficiency and usability.

Ethical Statement

Our work aims to improve the efficiency and scalability of data analysis by automating the generation of analytical insights from structured datasets. This can be particularly valuable in settings where manual analysis is costly or infeasible, enabling organizations to optimize operations and tailor strategies more effectively.

However, we acknowledge that the insights generated by our framework DataSage are produced automatically and may contain inaccuracies, misinterpretations, or incomplete reasoning due to model limitations or data quality issues. As such, the generated insights should be treated as preliminary references rather than definitive conclusions. Users must exercise critical judgment and, when necessary, seek human expert verification before acting on any insights produced by DataSage, especially in high-stakes domains such as finance, healthcare, or policy-making.

We emphasize that DataSage is intended to serve as an assistive tool to augment human analytical capabilities, not to replace domain experts or rigorous manual analysis. Responsible use requires transparency about the framework's limitations and active user oversight to avoid unintended consequences or over-reliance on automated outputs. By clearly framing DataSage as a decision support tool and not a decision maker, we aim to encourage responsible deployment and maximize its potential benefits while minimizing misuse.

References

Oluwatosin Abdul-Azeez, Alexsandra Ogadimma Ihechere, and Courage Idemudia. 2024. Enhancing business performance: The role of data-driven analytics in strategic decision-making. *International Journal of Management & Entrepreneurship Research*, 6(7):2066–2081.

Deepali Arora and Piyush Malik. 2015. Analytics: Key to go from generating big data to deriving business value. In *2015 IEEE first international conference on big data computing service and applications*, pages 446–452. IEEE.

Randy Bean. 2022. Why becoming a data-driven organization is so hard. *Harvard Business Review*.

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2025. Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 49–57.

Xiuying Chen, Tairan Wang, Qingqing Zhu, Taicheng Guo, Shen Gao, Zhiyong Lu, Xin Gao, and Xiangliang Zhang. 2024. Rethinking scientific summarization evaluation: grounding explainable metrics on facet-aware benchmark. *ArXiv*, pages arXiv–2402.

Liying Cheng, Xingxuan Li, and Lidong Bing. 2023. Is gpt-4 a good data analyst? In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9496–9514.

Eric Colson. 2019. What ai-driven decision making looks like. *Harvard Business Review*, 8:2–8.

Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. 2019. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 international conference on management of data*, pages 317–332.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, and 9 others. 2025. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, Vienna, Austria. Association for Computational Linguistics.

Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024a. Knowledge-to-sql: Enhancing sql generation with data expert llm. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 10997–11008.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024b. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, and 1 others. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. In *International Conference on Machine Learning*, pages 19544–19572. PMLR.

Nam Huynh and Beiyu Lin. 2025. Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications. *arXiv preprint arXiv:2503.01245*.

Chidera Victoria Ibeh, Oluwafunmi Adijat Elufioye, Temidayo Olorunsogo, Onyeka Franca Asuzu, Ndubuisi Leonard Nduubuisi, and Andrew Ifesinachi Daraojimba. 2024. Data analytics in healthcare: A review of patient-centric approaches and healthcare delivery. *World Journal of Advanced Research and Reviews*, 21(02):1750–1760.

Attia Khan. 2024. Effective decision making using data analytics. *Indian Scientific Journal Of Research In Engineering And Management*, 8(04):1–5.

LangChain. 2024. Pandas dataframe. https://python.langchain.com/v0.2/docs/integrations/toolkits/pandas/. Accessed: 2025-08-04.

Po-Ming Law, Alex Endert, and John Stasko. 2020. Characterizing automated data insights. In *2020 IEEE Visualization Conference (VIS)*, pages 171–175. IEEE.

Shuaimin Li, Xuanang Chen, Yuanfeng Song, Yunze Song, Chen Jason Zhang, Fei Hao, and Lei Chen. 2025. Prompt4vis: prompting large language models with example mining for tabular data visualization. *The VLDB Journal*, 34(4).

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Jinwei Lu, Yuanfeng Song, Chen Zhang, and Raymond Chi-Wing Wong. 2026. Multivis-agent: A multi-agent framework with logic rules for reliable and comprehensive cross-modal data visualization. *Proceedings of the ACM on Management of Data*, 4(1 (SIGMOD):1–25.

Li-Chun Lu, Shou-Jen Chen, Tsung-Min Pai, Chan-Hung Yu, Hung-yi Lee, and Shao-Hua Sun. 2024. Llm discussion: Enhancing the creativity of large language models via discussion framework and role-play. In *First Conference on Language Modeling*.

Alexandra L'heureux, Katarina Grolinger, Hany F Elyamany, and Miriam AM Capretz. 2017. Machine learning with big data: Challenges and approaches. *Ieee Access*, 5:7776–7797.

Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. InsightPilot: An LLM-empowered automated data exploration system. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 346–352, Singapore. Association for Computational Linguistics.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. 2025. Discoverybench: Towards data-driven discovery with large language models. In *International Conference on Learning Representations (ICLR)*.

Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. 2012. Big data: the management revolution. *Harvard business review*, 90(10):60–68.

Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. 2015. Deep learning applications and challenges in big data analytics. *Journal of big data*, 2(1):1.

OpenAI. 2024a. Code interpreter. https://platform.openai.com/docs/assistants/tools/code-interpreter. Accessed: 2025-08-04.

OpenAI. 2024b. Data analysis with chatgpt. https://help.openai.com/en/articles/8437071-data-analysis-with-chatgpt. Accessed: 2025-08-04.

Alberto Sánchez Pérez, Alaa Boukhary, Paolo Papotti, Luis Castejón Lozano, and Adam Elwood. 2025. An llm-based approach for insight generation in data analysis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 562–582.

Gaurav Sahu, Abhay Puri, Juan A. Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, Sai Rajeswar, and Issam H. Laradji. 2025. Insightbench: Evaluating business analytics agents through multi-step insight generation. In *The Thirteenth International Conference on Learning Representations*.

Stefan Steiner. 2022. Harnessing data to make betterinformed decisions. *Scientia*.

Robert Vacareanu, Vlad Andrei Negru, Vasile Suciu, and Mihai Surdeanu. 2024. From words to numbers: Your large language model is secretly a capable regressor when given in-context examples. In *First Conference on Language Modeling*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Yihan Wang and Peiyu Liu. 2025. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. *arXiv preprint arXiv:2503.18596*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yang Wu, Yao Wan, Hongyu Zhang, Yulei Sui, Wucai Wei, Wei Zhao, Guandong Xu, and Hai Jin. 2024. Automated data visualization from natural language via large language models: An exploratory study. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Ran Zhang and Mohannad Elhamod. 2025. Data-to-dashboard: Multi-agent llm framework for insightful visualization in enterprise analytics. *arXiv preprint arXiv:2505.23695*.

Xuanhe Zhou, Guoliang Li, Jianming Wu, Jiesi Liu, Zhaoyan Sun, and Xinning Zhang. 2023. A learned query rewrite system. *Proceedings of the VLDB Endowment*, 16(12):4110–4113.

Zhenghao Zhu, Yuanfeng Song, Xin Chen, Chengzhong Liu, Yakun Cui, Caleb Chen Cao, Sirui Han, and Yike Guo. 2026. Insighteval: An expert-curated benchmark for assessing insight discovery in llm-driven data agents. In *Findings of the Association for Computational Linguistics: ACL 2026*.

A Error Analysis: Failures of Existing Data Agents

Despite recent advancements in LLM-based data agents, we observe consistent and critical limitations when these systems are applied to real-world insight discovery tasks. As shown in Table 4, we categorize these limitations into three major types: domain knowledge underutilization, shallow analytical depth, and error-prone code generation. Through empirical observations on the state-of-the-art AgentPoirot model (Sahu et al., 2025), we identify and analyze representative failure cases.

A.1 Underutilization of Domain Knowledge

Most existing agents (Pérez et al., 2025; Sahu et al., 2025) assume that relevant knowledge is either encoded within the LLM's parameters or explicitly provided by the user. In practice, however, many analytical tasks require external domain context (such as definitions of business KPIs, seasonal market patterns, or industry-specific thresholds), which are rarely captured by LLMs. For example, when analyzing regional sales fluctuations, AgentPoirot observes a sharp drop in sales in northern China in early February and concludes that it is random variance. However, this drop actually aligns with the Chinese Spring Festival, a major national holiday that significantly disrupts commercial activity. Without this domain-specific knowledge, the agent fails to recognize the true cause, leading to a misleading conclusion. In another case, when analyzing marketing campaign effectiveness, AgentPoirot flags a low conversion rate as problematic without realizing that the campaign is intentionally targeted for long-term brand building rather than short-term sales. These examples show that without targeted domain augmentation or external retrieval mechanisms, agents struggle to distinguish signal from noise in context-rich data, often producing superficial or misleading insights.

A.2 Shallow Analytical Depth

Current systems (Sahu et al., 2025; Wu et al., 2024; LangChain, 2024) often rely on a single-pass LLM to generate analytical questions based on dataset description or metadata. While this setup can yield surface-level questions (such as identifying a top-selling product or calculating simple correlations), it struggles to generate deeper, multi-layered questions that require structured reasoning and contextual understanding. For instance, in a diagnostic task involving a sudden drop in gross margin, AgentPoirot asks, "Which category has the highest cost increase?", but fails to propose more insightful questions such as "Is the margin drop concentrated in specific provinces or time periods?" or "Do seasonal campaigns correlate with shifts in cost structure?". This lack of depth originates from the absence of iterative, divergent-convergent thinking processes, and limits the agent's ability to guide users toward non-obvious, high-value insights in complex domains like finance, healthcare, or operations.

A.3 Error-Prone Code Generation

LLMs are known to hallucinate or generate incorrect code (Huynh and Lin, 2025; Hong et al., 2024b), especially in non-trivial scenarios involving intermediate state reuse, complex data joins, or edge case handling. We observe some recurring failure modes: (1) Semantically correct but logically irrelevant code. For example, given a question like "Which customer segments showed declining profitability over the last two quarters", AgentPoirot instead computes overall revenue changes across all customers, missing the segment-level breakdown entirely. (2) Lack of robustness to schema inconsistencies. A common issue occurs when joining tables with mismatched column formats. In one case, AgentPoirot attempts to join a sales table (with a date column in string format like '2023-07-01') with a marketing table where the date column is stored as a datetime object. The failure to cast the types properly causes the join to return an empty result set, leading to misleading "no impact" conclusions. Even when users provide detailed prompts and table descriptions, the generated code often lacks defensive programming practices such as type checking or exception handling. These errors significantly reduce trust in autonomous agents for insight generation.

These failure patterns highlight a fundamental gap between existing data agents' capabilities and the practical demands of robust insight discovery. They motivate our design of a multi-agent architecture that explicitly addresses these shortcomings by incorporating domain-aware knowledge retrieval, divergent-convergent multi-role debating question raising, and multi-path reasoning for code generation.

Error Type	Analysis Task	AgentPoirot Output	Correct Output
Underutilization of Domain Knowledge	Analyze regional sales fluctuations	In early February, sales in northern China dropped sharply. Due to the lack of additional information in the data, **this was attributed to random variance.**	**The drop actually aligns with the Chinese Spring Festival,** a major national holiday that significantly disrupts commercial activity.
Shallow Analytical Depth	Diagnose a sudden drop in gross margin	**Raise shallow question:** "Which category has the highest cost increase?"	**Insightful Question:** "Is the margin drop concentrated in specific provinces or time periods?"
Error-Prone Code Generation	Evaluate the impact of a summer marketing campaign on sales	Join a sales table (with a date column in string format) with a marketing table where the date column is stored as a datetime object. -> **Empty result set.**	**Cast the types properly before joining.**

Table 4: Failure cases of the state-of-the-art *AgentPoirot* model in real-world insight discovery tasks. We categorize these errors into three major types: **domain knowledge underutilization**, **shallow analytical depth**, and **error-prone code generation**. **Red** text highlights the model's mistakes, while **green** indicates correct output.

Model	G-Eval	
	Insight-level	Summary-level
AgentPoirot	0.296	0.331
DataSage (Ours)	**0.318 (+7.4%)**	**0.362 (+9.4%)**

Table 5: Performance of the best baseline (AgentPoirot) and our framework on the InsightBench using Llama-3-70B as the base model. Our framework consistently outperforms the best baseline at both the insight and summary levels, demonstrating robustness and cross-model generalizability.

B More Experimental Results

B.1 Additional Base Model

To verify the robustness and assess the generalizability of our framework, we further evaluate both the best baseline (AgentPoirot) and our framework using Llama-3-70B as the base model on the InsightBench. As shown in Table 5, although Llama-3-70B performs noticeably worse than GPT-4o when used as the base model, DataSage still outperforms the best baseline by approximately 8% under this weaker backbone. This consistent margin demonstrates that our framework can generalize across different base LLMs.

B.2 Additional Benchmark

To further assess the generalizability and robustness of our framework, we additionally conduct experiments on InsightEval (Zhu et al., 2026).

InsightEval is an expert-curated benchmark designed to rigorously evaluate the insight-discovery capabilities of LLM-driven data analysis agents. Unlike prior datasets such as InsightBench, InsightEval emphasizes goal clarity, data consistency, and annotation reliability through a structured construction pipeline. By providing well-aligned question–answer pairs and trustworthy ground-truth insights, InsightEval enables more comprehensive, fair, and reproducible evaluation of agent performance in automated insight discovery.

As shown in Table 6, the results on InsightEval are highly consistent with our prior observations on InsightBench (Table 1). Across two different base LLMs, our framework consistently outperforms the strongest baseline on both evaluation metrics and across all three difficulty levels (easy, medium, and hard). The consistent advantage across different benchmarks and model configurations provides strong empirical evidence of the stability, transferability, and practical applicability of our framework.

Model	Insight-level Scores (G-Eval)				Summary-level Scores (G-Eval)			
	Easy	Medium	Hard	Avg	Easy	Medium	Hard	Avg
Pandas Agent (GPT-4o)	0.366	0.327	0.339	0.343	0.421	0.390	0.430	0.412
AgentPoirot (GPT-4o)	0.425	0.394	0.352	0.389	0.472	0.443	0.436	0.449
AgentPoirot (Deepseek-V3)	0.447	0.404	0.361	0.403	0.497	0.476	0.453	0.475
DataSage (GPT-4o)	0.444	0.411	0.372	0.409	**0.564**	0.517	0.485	0.521
DataSage (Deepseek-V3)	**0.476**	**0.431**	**0.380**	**0.428**	0.539	**0.542**	**0.495**	**0.526**

Table 6: Performance of different models based on different LLMs on the InsightEval. On the InsightEval benchmark, our framework also consistently outperforms the best baseline.

B.3 Ablation Study

B.3.1 The effect of RAKG module

To demonstrate the effectiveness of our RAKG module design, we conduct a comprehensive comparison of different retrieval strategies within our framework. We compare four different variants: No Retrieval (completely without any retrieval), Internal Knowledge (using *vanilla knowledge generator* leveraging only the internal knowledge of LLMs), On-Demand Retrieval (used in our RAKG module), and Full Retrieval (retrieval is utilized for all analysis tasks). As shown in Table 7, there are several interesting findings worth highlighting.

First, employing a *vanilla knowledge generator* yields a modest improvement over the no retrieval baseline. It indicates that the *vanilla knowledge generator* can further stimulate the latent knowledge encoded within LLMs, which is also consistent with prior works (Zhang and Elhamod, 2025; Hong et al., 2024a). It validates our design choice of incorporating the *vanilla knowledge generator* in RAKG module.

Second, integrating external retrieval significantly enhances model performance, demonstrating the critical role of external knowledge in real-world data analysis scenarios.

Third, among the four variants, the full retrieval approach achieves the highest scores, yet at the cost of invoking search queries for all instances, which substantially increases resource demands. In contrast, the on-demand retrieval mechanism strikes an effective balance between efficiency and performance. Despite utilizing only 24% of the search resources required by the full retrieval method, it achieves comparable performance. It suggests that on-demand retrieval can reduce resource consumption without compromising model effectiveness, highlighting the practical advantages of our dynamic retrieval design.

B.3.2 The effect of Question Raising module

DataSage employs a divergent-convergent multi-role debating process in Question Raising module, aiming to simulate diverse analytical perspectives and deepen the analytical depth. To experimentally validate this design, we compare the questions raised by our framework with those from the best baseline, AgentPoirot, evaluating both diversity and coverage of the raised questions.

We quantify diversity as the average pairwise dissimilarity between question embeddings, calculated by:

$$\text{Diversity} = 1 - \frac{2}{n(n-1)} \sum_{i<j} \text{cosine\_sim}(v_i, v_j), \tag{14}$$

where v_i and v_j are the embedding vectors of questions i and j. Coverage is defined as the average radius of the questions around their centroid embedding, computed by:

$$\text{Coverage} = \frac{1}{n} \sum_{i} \|v_i - \text{centroid}\|, \tag{15}$$

where the centroid is the mean vector of all question embeddings.

As shown in Figure 5, DataSage achieves significantly higher diversity and coverage scores compared to AgentPoirot, even when generating fewer questions. Although increasing the number of raised questions generally improves these metrics, AgentPoirot's questions tend to be more similar or even repetitive, limiting its diversity and coverage of the analytical space. This often leads to the generation of shallow and generic insights, whereas DataSage effectively promotes the creation of diverse and comprehensive questions, enabling the generation of deeper and more diverse insights.

Model Variant	G-Eval (Insight)	G-Eval (Summary)	Retrieval Usage Rate (%)
No Retrieval	0.3316	0.3982	0%
Internal Knowledge	0.3366	0.3994	0%
On-Demand Retrieval	0.3530	0.4059	24%
Full Retrieval	0.3560	0.4082	100%

Table 7: Performance comparison of four different retrieval strategies in DataSage. Our on-demand retrieval mechanism strikes an effective balance between efficiency and performance, achieving near-optimal performance while significantly reducing the search resource consumption compared to full retrieval.

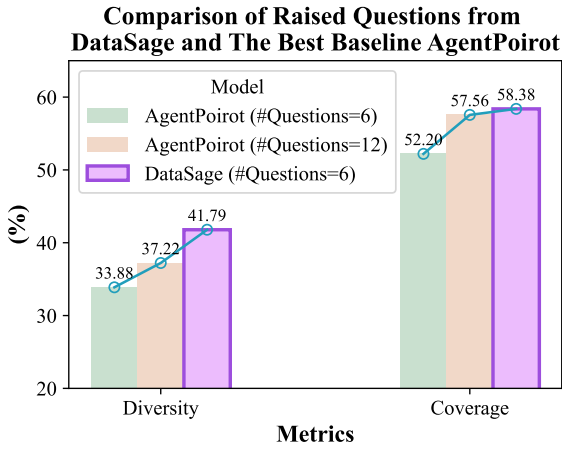


Figure 5: Comparison of diversity and coverage of the questions raised by DataSage and the best baseline AgentPoirot. Even when generating fewer questions (6 vs. 12), DataSage produces questions with significantly higher diversity and coverage, enabling the generation of deeper and more diverse insights.

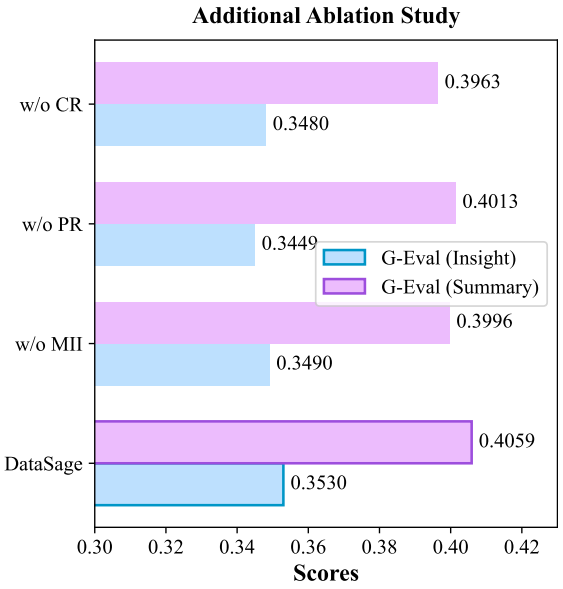


Figure 6: Additional ablation study results. Removing Multimodal Insight Interpretation (MII), Plot Reviewer (PR), or Code Refinement (CR) consistently degrades performance, confirming the necessity of these components.

Model	Success Rate	#Refinements
AgentPoirot	95.17%	-
DataSage	**99.50%**	**1.36**
w/o MR	98.17%	1.63

Table 8: The effect of Multi-path Reasoning (MR) on code execution success rate and average number of code refinements. Multi-path Reasoning not only improves code correctness but also reduces refinement overhead.

B.3.3 The effect of Multi-path Reasoning

To further examine the benefits of our proposed Multi-path Reasoning (MR), we evaluate its impact on both code correctness and refinement efficiency. Specifically, we use a simple yet effective proxy for correctness: the execution success rate, i.e., the proportion of generated code snippets that run without errors. In addition, we measure the average number of code refinements required in the Insight Generation Module. We compare three variants: DataSage, DataSage without MR, and the best baseline AgentPoirot.

As shown in Table 8, DataSage significantly outperforms the best baseline, achieving a 4.33 percentage point gain in execution success rate (99.50% vs. 95.17%). Moreover, comparing DataSage to its variant without MR highlights the effectiveness of Multi-path Reasoning. The success rate improves from 98.17% to 99.50%, while the average number of refinements decreases from 1.63 to 1.36. This indicates that Multi-path Reasoning can not only increase code correctness but also reduce the reliance on post-hoc refinement. It enhances robustness by mitigating single-path reasoning errors, while also improving efficiency by reducing unnecessary refinement iterations.

B.3.4 Additional Ablation Study

Beyond the core components of DataSage, we further investigate the impact of several finer-grained design choices. Specifically, we examine three variants: (i) removing Multimodal Insight Interpretation (using only text-level interpretation), (ii) removing the Plot Reviewer, and (iii) removing Code Refinement. As shown in Figure 6, the absence of any of these components consistently degrades performance. This validates the necessity of our full design. Notably, Multimodal Insight Interpretation and Plot Reviewer are complementary. The Plot Reviewer enhances the quality of generated plots, thereby providing more reliable inputs for multimodal interpretation. Meanwhile, Code Refinement improves the correctness of generated code, which directly contributes to more accurate insight discovery. These results highlight the synergistic contributions of different components and demonstrate the robustness of our framework.

B.4 Additional Experimental Analysis

B.4.1 Usability Assessment

Model	#API Calls	Runtime	Cost
DataSage	94.08	6.59 minutes	1.06 $

Table 9: Usability assessment of our framework (DataSage) using GPT-4o as the base model, reporting the average number of API calls, runtime, and cost per instance. The results show that DataSage remains practical and efficient despite involving multiple collaborative agents.

To assess the usability of our framework, we report the average runtime, the number of API calls, and the cost of our framework when using GPT-4o as the base model. As shown in Table 9, these statistics demonstrate that despite incorporating multiple collaborative agents, our framework remains practical and efficient. The runtime and cost of DataSage are comparable to those of existing iterative LLM agent systems.

B.4.2 Performance per Category

To evaluate the capabilities of our framework in more detail, we conduct a performance-per-category analysis. As shown in Table 10, we report the performance for each category following the taxonomy defined in InsightBench. Our framework outperforms the strongest baseline on every category and on both evaluation dimensions. This indicates that the improvements are not isolated to specific types of tasks but generalize well across diverse reasoning and analytic requirements. In these categories, Financial Management exhibits the lowest performance for both our framework and the best baseline. This indicates the inherent difficulty of tasks in this specific financial domain and provides a clear direction for future improvement.

B.4.3 Multi-path Reasoning Statistics

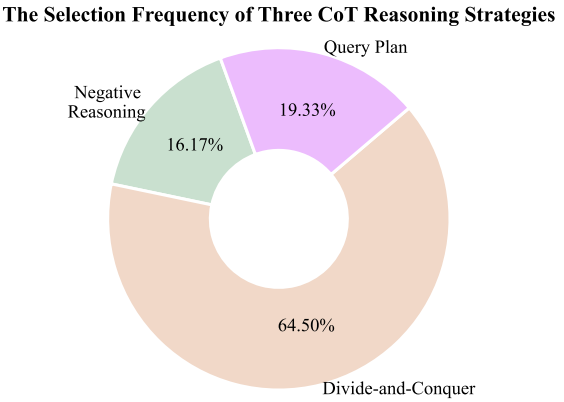


Figure 7: Distribution of the final selected code outputs generated by three CoT reasoning strategies. DaC accounts for the majority of selections (64.50%), highlighting its effectiveness in data analysis tasks, while QP and NR contribute complementary reasoning capabilities that enhance overall code correctness.

We conduct a statistical analysis on the selection frequency of three complementary CoT reasoning strategies (i.e., Divide-and-Conquer, Query Plan, and Negative Reasoning). As shown in Figure 7, the DaC (Divide-and-Conquer) strategy is the most frequently chosen, accounting for 64.50% of the final selected code outputs, followed by QP (Query Plan) at 19.33%, and NR (Negative Reasoning) at 16.17%. This distribution suggests that in data analysis scenarios, the DaC approach tends to be more effective and reliable in generating correct code. Although the selection rates for QP and NR are relatively lower, these strategies play a complementary role, enabling the model to leverage the strengths of different reasoning paths and ultimately select the most accurate solution. This complementary interplay increases the overall likelihood of producing correct code. The statistical results also motivate us to prioritize the DaC strategy for code generation in resource-constrained environments.

Category	Insight-level		Summary-level	
	AgentPoirot	DataSage (Ours)	AgentPoirot	DataSage (Ours)
Incidents Management	0.3248	**0.3833**	0.3299	**0.3891**
Financial Management	0.2832	**0.2911**	0.3166	**0.3585**
Goal Management	0.3768	**0.3859**	0.4177	**0.4479**

Table 10: Performance of our framework (DataSage) and the strongest baseline (AgentPoirot) on each category in InsightBench. Our framework consistently outperforms the best baseline across all categories, with Financial Management being the most challenging category for both models.

C Case Study

C.1 DataSage generates better insights.

To better demonstrate the capability of DataSage in generating actionable insights, we conduct a case study on an incidents management task in InsightBench. As shown in Table 11, we compare the outputs of our framework with those from the best baseline model AgentPoirot. The advantages of our framework can be summarized in the following three points.

Statistically Validated Insights. The baseline model identifies the general growth trend in David Loo's incident submissions, reporting that the count nearly doubled over the year and highlighting an increase in high-priority incidents. However, these descriptions remain at a descriptive level and lack supporting statistical evidence or precise quantification. In contrast, DataSage provides quantitative trend metrics, such as slope estimates (slope = 0.057, p $\approx$ 1.25e-5), explicit percentage increases ($\approx$100%), and temporal clustering statistics (97% within one week). These quantitative markers not only confirm the trend but also allow for statistical significance assessment.

Multi-Dimensional In-Depth Insights. While the baseline model's focus is largely confined to incident counts and priority levels, DataSage extends the analysis to multiple operational dimensions:

- Resolution efficiency: Comparing average resolution times (175 vs. 177 hours) while identifying resource strain due to increased workload per staff member.
- Duplicate submissions: Detecting a measurable upward trend in duplicate or near-duplicate incident tickets (0.0032/day), particularly concentrated in specific months.
- Process inefficiencies: Highlighting frequent reopenings of tickets based on overlapping "Resolved" and "Closed" states.

This is attributed to our carefully designed Question Raising Module, which can simulate diverse analytical perspectives and deepen analytical depth.

More Actionable Insights. The baseline model remains limited to high-level descriptions and does not capture underlying process issues or secondary contributing factors. Our framework effectively bridges this gap by producing operationally actionable insights that enable targeted, data-driven decision-making:

- Pinpointing peak months and weeks for incident clustering, enabling targeted root cause investigation.
- Quantifying workload per support staff member to guide resource allocation.
- Identifying increasing duplicate or near-duplicate ticket submissions, along with frequent incident reopenings, providing a clear direction for process improvements.

In summary, the case study demonstrates that while the best baseline model AgentPoirot can detect broad patterns, it often stops at a descriptive level, lacking statistical rigor, in-depth analysis and operational context. Our framework enriches them with statistically validated, multi-faceted, and actionable insights, thereby offering more robust decision support for organizations.

C.2 DataSage generates better plots.

Beyond generating better insights, DataSage can also generate more accurate, diverse, and interpretable plots. Using the same incidents management task, we conduct a comparative case study on plot generation between our framework DataSage and the best baseline model AgentPoirot. As shown in Table 12, DataSage outperforms the baseline in all of the following four aspects.

Insights Predicted by AgentPoirot (The Best Baseline Model)	Advantages of DataSage	Insights Predicted by DataSage (Ours)
Task: ***Incidents Management***		
The number of incident tickets created by "David Loo" nearly doubled over the span of one year, rising from 16 incidents in early 2023 to 32 incidents in early 2024, highlighting a significant increase in reported issues from this single caller which may indicate growing system problems or increased user activity.	Adds slope and significance testing for trend validation, and incorporates temporal distribution analysis (hourly, weekly), enhancing both rigor and coverage. ***[Statistical Quantification]***	David Loo's incident ticket submissions increased by approximately 100% over the year, with a significant positive monthly trend **(slope = 0.057, p $\approx$ 1.25e-5)**, and his ticket volume remains consistently higher than other callers **across all hours and weekdays**, highlighting a persistent and growing source of incidents that may require focused attention.
David Loo's incident ticket creation nearly doubled over the span of a year, highlighting a growing demand or issue source from this caller that may require targeted investigation.	Links growth to duplicate submissions, quantifies rate change, and identifies temporal peaks, enabling targeted root cause investigation. ***[Root Cause Analysis]***	Among all callers, "David Loo" uniquely exhibits a measurable upward trend in submitting duplicate or near-duplicate incident tickets, with the number of such tickets increasing by approximately 0.0032 per day (about 1 extra duplicate every 10 months), particularly between September and November 2023, **suggesting that his reported incident volume is partially driven by repeated submissions of similar problems within short timeframes**.
The data reveals a growing trend in high-priority incidents reported by David Loo, with "2 - High" priority incidents more than doubling over the year, highlighting a potential escalation in the severity or frequency of issues faced by this caller.	Extends priority-level analysis with temporal clustering metrics (97%) and category detail (network), providing precise basis for emergency planning. ***[Operational Planning]***	David Loo's incident reports not only increase sharply by nearly two incidents per month but also cluster tightly in time, with 97% occurring within a week of each other, mostly involving high-priority network issues, **suggesting urgent, recurring problems that may require targeted operational attention**.
Despite the ITIL User having increasing incident reports, their priority distribution remains aligned with other callers, with approximately 75% of their incidents classified as "2 - High," suggesting that the increase is not driven by a shift toward more severe or less severe incident priorities.	Shifts focus from static priority comparison to operational efficiency and workload impact, offering direct guidance for staffing decisions. ***[Resource Allocation]***	Despite a similar average resolution time (175 vs. 177 hours), callers with rising incident volumes cause a substantial increase in workload, with incident counts assigned to each support staff member nearly doubling (e.g., from 27–40 to 59–88 incidents), highlighting that **growing incident frequency demands more extensive support resource allocation without degrading resolution speed**.
While ITIL User reports fewer incidents overall, about 69% (69 out of 75) of their incidents are high or critical priority, indicating that their reported incidents are relatively more severe compared to the distribution of priorities from other callers.	Integrates severity with lifecycle analysis, detecting reopen loops and processing delays, extending severity analysis into process stability evaluation. ***[Process Analysis]***	Callers with rising incident volumes, notably "David Loo" and "Don Goodliffe," have average incident open durations ranging from approximately 140 to over 310 hours monthly, with 12 to 13 months showing overlapping "Resolved" and "Closed" states that suggest frequent incident reopenings, indicating **persistent delays and instability in incident resolution that likely impact service efficiency and warrant targeted process improvements**.

Table 11: A case study comparing the insight outputs from our framework DataSage and the best baseline model AgentPoirot. While AgentPoirot detects broad patterns, it remains at a descriptive level and often lacks statistical rigor, in-depth analysis, and operational context. The advantages of DataSage over the baseline model are highlighted in **bold**, including statistically validated, multi-faceted, and more actionable insights, which offer more robust decision support.

Richness of Information. The plots generated by the baseline model often remain narrowly focused, lacking contextual comparisons or layered breakdowns. For example, when visualizing caller-level trends, the baseline displays isolated line charts without integrating group-level context. DataSage enriches these visuals by embedding comparative dimensions, such as group-to-

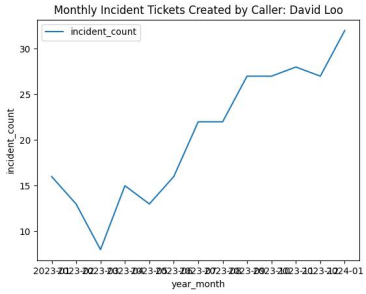
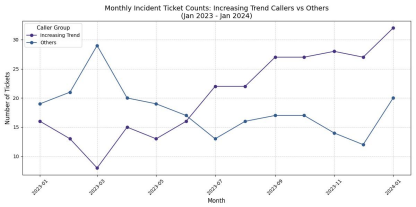
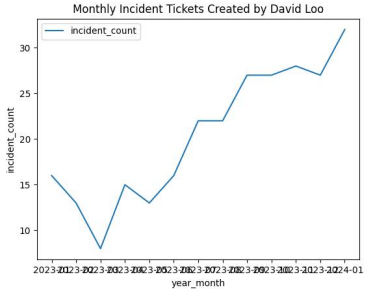
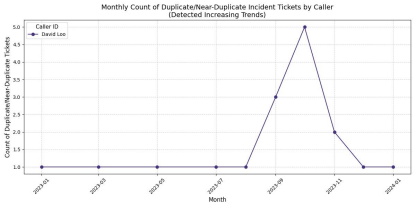
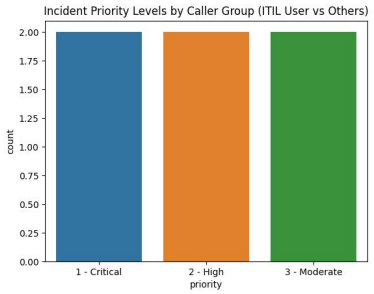
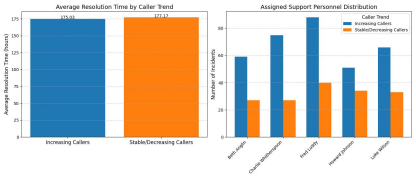
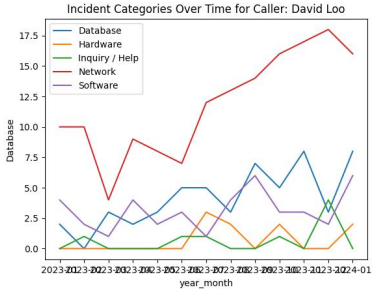
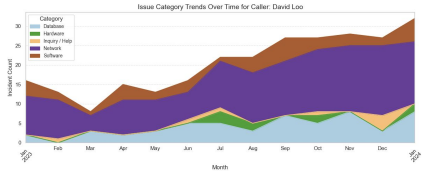
Plots Generated by AgentPoirot (The Best Baseline Model)	Advantages of DataSage	Plots Generated by DataSage (Ours)
Task: *Incidents Management*		
	Introduces group-level comparisons to contextualize individual trends, identifying unique vs. systemic patterns to enhance decision relevance. ***[More comparative and informative]***	
	Generates diversified, non-redundant plots and insights, linking growth to duplicate submissions for better targeted investigation. ***[More diverse and non-redundant]***	
	Generates correct plots, fixing baseline's coding error (all bars have the same height), and shifts focus to operational efficiency and workload impact for staffing decisions. ***[Error-free code and plot generation]***	
	Replaces baseline's cluttered, mislabeled (y-axis titled "Database" instead of "Incident Count") line chart with a clear stacked-area design, removing overlap and simplifying time labels for more interpretable plots. ***[More readable and interpretable]***	

Table 12: A case study comparing the plots generated from our framework DataSage and the best baseline model AgentPoirot. While AgentPoirot can produce basic visualizations, its plots tend to be narrowly focused, redundant, error-prone, and suffer from poor readability. In contrast, DataSage demonstrates clear advantages including richer information, more diverse and non-redundant plots, error-free plot generation, and enhanced readability and interpretability.

individual contrasts, highlighting both unique and systemic patterns. This allows decision-makers to distinguish localized anomalies from broader shifts, directly enhancing strategic relevance.

Diversity and Non-Redundancy. As illustrated by the cases in the first and second rows of Table 12, baseline outputs tend to repeat similar plots across different insights, limiting analytical novelty and depth. In contrast, DataSage generates diverse and non-redundant plots. This variety not only prevents

redundancy but also enhances the diversity and depth of analysis.

Error-Free Code and Plots. In the baseline outputs, code logic errors result in misleading plots, such as the bar chart where all bars have identical heights regardless of the data. Through our carefully designed multi-path reasoning and code fix process, DataSage resolves such issues, producing error-free plots that reflect the true data distribution. It ensures that visualizations can be trusted for decision-making with less manual validation.

Readability and Interpretability. The baseline model frequently produces plots with visual clutter, label overlap, and mislabeling. In contrast, DataSage replaces these with clean, stacked-area or grouped-bar designs, improved color palettes, and simplified axis labeling (e.g., monthly labels "Jan-Dec" instead of raw date strings). DataSage significantly improves the readability and interpretability of the plots, even for complex multi-category data. It not only enhances the accuracy of insight generation by the *interpreter* agent but also facilitates human analysts in understanding complex patterns and making informed decisions more effectively, increasing the overall utility and trustworthiness of our framework.

In summary, while the baseline model can produce basic visual plots, it lacks the accuracy, diversity, and interpretability, especially required for high-stakes operational analysis. DataSage delivers richer, more varied, and more readable visual plots, transforming plots from passive illustrations into active decision-support tools.

D Prompts for Agents

Prompt 1 - Prompt 15 present the detailed prompts for agents in DataSage.

Prompt 1: Prompt for the *judge* agent in RAKG Module.

```
You are a reasoning-enhanced data analysis agent. Your task is to help extract
    meaningful insights from real-world data.

## Context:
- The analysis goal is: <goal>{goal}</goal>
- The database schema is as follows:
<schema>{schema}</schema>
- Sample data or description of contents:
<info>{description}</info>

## Your Task:
1. Carefully examine the above context and determine whether the analysis can be
    completed **purely based on the given data**.
2. If not, **identify what kind of external knowledge** (e.g., domain-specific
    definitions and knowledge, policy changes, socioeconomic context, product
    classification standards, etc.) is likely needed.
3. Clearly state whether external search is needed:
   - If YES:
     - Describe **why** the search is necessary.
     - Specify **what kind of knowledge** should be retrieved.
   - If NO:
     - Explain why the current information is sufficient.

## Constraints:
- Be cautious not to assume unavailable data.
- Use chain-of-thought reasoning: think step by step before deciding.

## Respond in the following format (If YES, Knowledge Needed):
<reason>Step-by-Step Reasoning</reason>
<answer>YES/NO</answer>
```

Prompt 2: Prompt for the *data-aware query generator* agent in RAKG Module.

```
You are a domain-aware data analyst assistant.

## Context:
- The analysis goal is: <goal>{goal}</goal>
- The database schema is as follows:
<schema>{schema}</schema>
- Sample data or description of contents:
<info>{description}</info>

## Your Task:
1. **Write a list of concise, high-quality and effective Google search queries** to
    retrieve reliable useful domain knowledge.
2. The query should be focused, unambiguous, and ready for direct use in a Google
    search engine.
3. You can write at most **{max_queries}** queries. Ensure the richness of multiple
    queries and avoid semantic duplication.

## Respond in the following format:
<query>The search query</query>
```

Prompt 3: Prompt for the *knowledge generator* agent in RAKG Module.

```
You are a domain-aware, search-augmented data analysis assistant.

## Objective:
Given a data analysis context and the corresponding search engine results, identify
    and extract **useful domain knowledge or external facts** that can help guide,
    validate, or enrich the analysis process.

## Input:
### 1. Data Analysis Context
- The analysis goal is: <goal>{goal}</goal>
- The database schema is as follows:
<schema>{schema}</schema>
- Sample data or description of contents:
<info>{description}</info>

### 2. Search Results
<search_results>
{search_results}
</search_results>

## Task:
* Based on the data analysis context and search results, what additional vertical
    field and domain knowledge does the analyst need to know to achieve the goal?
* Provide a list of knowledge item that can be used by the data scientists in your
    team to explore my data and reach my goal.
* Explore diverse aspects of the data, and provide knowledge that is relevant to the
     analysis goal.
* Do not number the knowledge item.
* Most importantly, each knowledge item must be enclosed within <knowledge></
    knowledge> tags.

## Output Format:
<knowledge>A knowledge item.</knowledge>
```

Prompt 4: Prompt for the *role designer* agent in Question Raising Module.

```
You are an expert system designer working on a multi-agent data analysis framework.
    Your current task is to design a **diverse set of problem-posing roles** for the
     system's "question generation module," where each role adopts a unique
    perspective to ask insightful questions about a given dataset.

Your goal is to **maximize diversity and coverage** in the types of questions that
    could uncover valuable, possibly hidden, insights from the data.   Each role
    should have a clearly defined **perspective**, **focus area**, etc.

### Please output a structured list of roles with the necessary information such as:
     Role Name, Background and Perspective (e.g., behavioral analyst, business
    strategist, anomaly detector), Capability, Knowledge, Personality Traits, etc.

### Data Context:

The analysis goal is:
<goal>{goal}</goal>

The schema of the dataset is:
<schema>{schema}</schema>

The detailed information of the dataset file is:
<info>{description}</info>

### Instructions:
* Write a list of roles with the required information.
* You can produce at most {max_roles} roles. Therefore, you need to make a careful
    choice before answering.
* Most importantly, each role description must be enclosed within <role></role> tags
    . For example: <role>Description of role1</role> <role>Description of role2</
    role> ...
```

Prompt 5: Prompt for *question raising* for different analytical roles in Question Raising Module.

```
[Role Description]
You are:
{role_description}


[Task Description]
I require the services of your team to help me reach my goal.

<goal>{goal}</goal>

<schema>{schema}</schema>

<info>{description}</info>

### Previous Answered Questions:
<prev_questions>{prev_questions}</prev_questions>

Given relevant domain knowledge that may be useful:
{knowledge}

Instructions:
* Produce a list of follow up questions to explore my data and reach my goal.
* Note that we have already answered <question> and have the answer at <answer>, do
    not include a question similar to the one above.
* Explore diverse aspects of the data, and ask questions that are relevant to my
    goal.
* You must ask the right questions to surface anything interesting (trends,
    anomalies, etc.)
* Make sure these can realistically be answered based on the data schema.
* The insights that your team will extract will be used to generate a report.
* Each question that you produce must be enclosed in <question>content</question>
    tags.
* Each question should only have one part, that is a single '?' at the end which
    only require a single answer.
* Do not number the questions.
* You can produce at most {max_questions} questions.
```

Prompt 6: Prompt for the *judge* agent in Question Raising Module.

```
You are an expert data analyst tasked with selecting the most promising questions
    from a set of diverse questions proposed by multiple agents, each with a unique
    role and perspective. Your goal is to identify the questions that are most
    likely to uncover meaningful, non-trivial insights from the data, but Do Not
    select a question similar to the previous answered questions.
These questions may differ in form (e.g., comparative, temporal, behavioral, causal)
    , but your selection should prioritize:
* Potential to reveal non-obvious patterns, relationships, or conclusion
* Relevance to the data schema and business context
* Feasibility of being answered with the given data
* Diversity in angles or dimensions (not redundant)

### Input:
You will be given:
* The number of questions to select: {number}
* The data context and schema
* A list of questions that are answered previously.
* A list of questions proposed by multiple roles.

### Output Requirements:
Please return the following:
1. A list of the {number} selected questions, ranked if possible
2. For each selected question, include:
    * The index of the question
    * The question text
    * The reason for selection (e.g., why it is likely to yield a valuable insight)
    * Output the index of the question in your response inside <question_id></
    question_id> tag.

### Data Context:
The analysis goal is:
<goal>{goal}</goal>

The schema of the dataset is:
<schema>{schema}</schema>

The detailed information of the dataset file is:
<info>{description}</info>

### Previous Answered Questions:
<prev_questions>{prev_questions}</prev_questions>

### Proposed Questions:
<proposed_questions>{proposed_questions}</proposed_questions>

### Example Output Format:
<question_id>0</question_id> <question>The text of question 0.</question> <reason>
    The reason for selection.</reason>
<question_id>3</question_id> <question>The text of question 3.</question> <reason>
    The reason for selection.</reason> ...
Most importantly, the index of questions start with 0, so the selected question
    index should be between 0-{ques_num} !
```

Prompt 7: Prompt for the *schema-aware question rewriter* agent in Insights Generation Module.

```
[Role Instruction]
You are a schema-aware question reformulator with expertise in database systems.
    Your task is to rewrite the given question by explicitly incorporating missing
    semantic information of the datasets, deconstruct ambiguities, clarify
    objectives to maximize clarity and actionability of the question. Follow these
    steps strictly:
1.  Intent Deconstruction
# Extract the core verb phrase (VP) and key named entities (NE) from the original
    question.
# Identify ambiguous or incomplete semantics due to missing schema elements.

2. Semantic Anchoring
# Enhance the question by explicitly adding:
a) Missing table/column names (as indicated in the schema&information of the
    datasets)
b) Temporal constraints (e.g., semester, year, quarter) if relevant
c) Categorical dimensions (e.g., student type, degree level)
d) Aggregation requirements (e.g., sum, average, count)

3. Deconstruct Ambiguities and Clarify Objectives
# Identify, re-express or explain vague terms, implicit assumptions, acronyms or
    undefined variables in the original question.
# Extract the core intent (What must the answer achieve?)

4.  Structural Reformulation
# Restructure the question. Format as self-contained, clear query with necessary
    context embedded.

[Output Requirements]
# Preserve all technical terms from the original question.
# Do not include explanatory notes or unrelated content.
# Most importantly, the rewritten question must be enclosed within <question></
    question> tags. Refer to the example response below.

[Example Demonstration]
Question: Find the **avg** age of faculty members.
Schema: [["name": faculty_birth_year, "type": object, "missing_count": 0, "
    unique_count": 124], ...]
Information: The detailed information of the dataset.
Rewritten Question: <question>In a database containing faculty_birth_year and
    department_info, how to calculate the average age of faculty members grouped by
    department_name?</question>

[Task Execution]
Now, rewrite the following question based on the provided schema&information of the
    datasets:
Question:
{question}

Schema:
{schema}

Information:
{description}

Domain knowledge:
{knowledge}

Rewritten Question:
```

Prompt 8: Prompt for *multi-path code generation* in Insights Generation Module.

```
SYSTEM PROMPT for Divide-and-Conquer:
    You are a professional database administrator and expert software engineer
    specializing in code writing and improvement. You should always reason with a "
    divide-and-conquer" mindset before writing the code: Decompose the user's
    question into the smallest independent sub-tasks possible; Solve the sub-tasks
    sequentially, making your intermediate reasoning explicit; You must output your
    detailed step-by-step "divide-and-conquer" reasoning within the tag <reasoning
    ></reasoning>; You must comment all your reasoning process in the code using
    '#', including the content within <reasoning></reasoning> tags! ; Only after the
     reasoning is complete, write the final code (faithful to the reasoning).

SYSTEM PROMPT for Query Plan:
    You are a professional database administrator and expert software engineer
    specializing in code writing and improvement. Before writing code, produce an
    explicit **query-plan** in plain English: Outline each logical operation in
    order-filters, joins, sub-queries / CTEs, aggregations, sorts; Reference exact
    table / column names from the schema; State the expected intermediate DataFrame
    or Series after every step; You must output your detailed "query-plan" reasoning
     within the tag <reasoning></reasoning>; You must comment all your reasoning
    process in the code using '#', including the content within <reasoning></
    reasoning> tags! ; Only after the plan is complete, write the final code (
    faithful to the plan).

SYSTEM PROMPT for Negative Reasoning:
    You are a professional database administrator and expert software engineer
    specializing in code writing and improvement. You are very meticulous and good
    at avoiding common pitfalls. Before writing code, produce an explicit **
    Counterfactual / Negative Reasoning** in plain English: List plausible errors or
     misconceptions someone might make when answering the question (e.g., double
    counting, wrong date window, missing NULLs, dtype mismatches); For each risk,
    explain how your Python code will prevent it; You must output your detailed "
    Counterfactual / Negative Reasoning" within the tag <reasoning></reasoning>; You
     must comment all your reasoning process in the code using '#', including the
    content within <reasoning></reasoning> tags! ; Only after the reasoning is
    complete, write the final code (faithful to the reasoning).
```

Prompt 9: Prompt for the *code selector* agent in Insights Generation Module.

```
You are a kind, experienced and professional database administrator and code auditor
     tasked with selecting the correct code from a set of diverse code written by
    multiple agents, each with a unique coding perspective.


### Input:
You will be given:
* The data context, schema and the user question.
* A list of codes trying to answer the question.
* Other Code Requirements.


### Output Requirements:
Please return the following:
* The index of the selected code.
* The reason for selection.
* Output the index of the code in your response inside <code_id></code_id> tag.


### Data Context:
The analysis goal is:
<goal>{goal}</goal>

The user question is:
<question>{question}</question>

The schema of the dataset is:
<schema>{schema}</schema>

The detailed information of the dataset file is:
```

```
<info>{description}</info>

The path of the dataset file is:
<path>{database_path}<path>

Other Code Requirements are:
<requirements>
* Make a single code block for starting with ```python
* Do not produce code blocks for languages other than Python.
* Import pandas as pd, matplotlib.pyplot as plt, numpy as np, ...
* You may use the predefined functions mentioned above to generate plots, or write
    your own plotting functions. If you choose not to use the predefined functions,
    you must save the generated plot using the following code: plt.savefig("plot.jpg
    ") plt.close()
* You must generate one single simple plot and save it as a jpg file.
* The generated plot will be analyzed by a data analyst, so please ensure it is as
    clear and easy to understand as possible.
* Plot Best Practices:
  - Adds appropriate titles, labels, and annotations that highlight the key insights
    .
  - For legends: Always use clear, descriptive legend titles and place them
    optimally (usually upper right or outside).
  - For color selection: Use colorblind-friendly palettes (viridis, plasma, cividis)
     or plt.cm.Paired
  - For multiple series: When plotting multiple data series, either: Use plt.
    subplots to create separate plots, or Use proper stacking techniques with
    stacked=True parameter. Avoid overwriting plots on the same axes unless showing
    direct comparisons.
  - For pie charts: Use plt.axis('equal') to ensure proper circular appearance.
  - For data preparation: Prepare data properly before visualization (aggregation,
    transformation). For example, use pandas aggregation (crosstab, pivot_table)
    before plotting.
  - For formatting: Include appropriate sizing and formatting for all visual
    elements. For example, set appropriate fontsize for title (14), labels (12), and
     tick labels (10). Calls plt.tight_layout() if necessary.
* For the plot, save a stats json file that stores the data of the plot.
* For the plot, save a x_axis.json and y_axis.json file that stores 100 most
    important x and y axis data points of the plot, respectively.
* For the json file must have a "name", "description", and "value" field that
    describes the data.
* If the content of the json file is getting too long, truncate the unnecessary
    parts until the number of characters is less than 10000.
* End your code with ```.
</requirements>


### Candidate code:
## Code Candidate 0:
<code_0>
{code_0}
</code_0>
## Code Candidate 1:
<code_1>
{code_1}
</code_1>
## Code Candidate 2:
<code_2>
{code_2}
</code_2>


### Example Output Format:
<code_id>5</code_id><reason>The reason for selection.</reason>
Most importantly, the index of code candidate start with 0, so the selected code
    index should be between 0-2 !

### Response:
```

Prompt 10: Prompt for the *code reviewer* agent in Insights Generation Module.

```
You are a kind, experienced and professional database administrator and code auditor
    . Given the [Database schema] descriptions, [Database detailed information]
    descriptions, and a user [Question] with corresponding [Python code], you need
    to analyze the provided code. Start from the components of [Python code], step
    by step, to analyze whether it aligns with the user [Question] intent, aligns
    with the [Database schema], and any possible errors occur.

There are some criteria:
### 1.  Requirement Alignment
- Verify if the code fully addresses the user's original question
- Identify discrepancies between requested functionality and implementation

### 2.  Database Compliance
**Schema Validation**:
- Check table/column naming against schema definitions
- Validate data types (e.g., VARCHAR length vs actual data)
- Verify constraints (PK/FK/UNIQUE/NOT NULL) are properly handled

**Data Format Integrity**:
- Ensure date/numeric formats match database settings
- Confirm encoding/charset consistency
- Check bulk operation compatibility (batch size, transaction scope)

### 3.  Problem Detection
**Critical Issues**:
- SQL injection vulnerabilities
- Race conditions in concurrent operations
- Resource leaks (connections, cursors)

**Operational Risks**:
- N+1 query patterns
- Missing transaction boundaries
- Index misuse (e.g., non-sargable queries)

**Data Integrity Concerns**:
- Dirty read/write scenarios
- Improper null handling
- Silent truncation risks

### 4.  Output Format Requirements
**Structured Response Requirements**

[Goal]
{goal}
[Database schema]
{schema}
[Database detailed information]
{description}
[Database path]
{database_path}
[Question]
{question}
[Other Code Requirements]
* Make a single code block for starting with ```python
* Do not produce code blocks for languages other than Python.
* Import pandas as pd, matplotlib.pyplot as plt, numpy as np, ...
* You may use the predefined functions mentioned above to generate plots, or write
    your own plotting functions. If you choose not to use the predefined functions,
    you must save the generated plot using the following code: plt.savefig("plot.jpg
    ") plt.close()
* You must generate one single simple plot and save it as a jpg file.
* The generated plot will be analyzed by a data analyst, so please ensure it is as
    clear and easy to understand as possible.
* Plot Best Practices:
  - Adds appropriate titles, labels, and annotations that highlight the key insights
    .
  - For legends: Always use clear, descriptive legend titles and place them
    optimally (usually upper right or outside).
```

```
  - For color selection: Use colorblind-friendly palettes (viridis, plasma, cividis)
     or plt.cm.Paired
  - For multiple series: When plotting multiple data series, either: Use plt.
    subplots to create separate plots, or Use proper stacking techniques with
    stacked=True parameter. Avoid overwriting plots on the same axes unless showing
    direct comparisons.
  - For pie charts: Use plt.axis('equal') to ensure proper circular appearance.
  - For data preparation: Prepare data properly before visualization (aggregation,
    transformation). For example, use pandas aggregation (crosstab, pivot_table)
    before plotting.
  - For formatting: Include appropriate sizing and formatting for all visual
    elements. For example, set appropriate fontsize for title (14), labels (12), and
     tick labels (10). Calls plt.tight_layout() if necessary.
* For the plot, save a stats json file that stores the data of the plot.
* For the plot, save a x_axis.json and y_axis.json file that stores 100 most
    important x and y axis data points of the plot, respectively.
* For the json file must have a "name", "description", and "value" field that
    describes the data.
* If the content of the json file is getting too long, truncate the unnecessary
    parts until the number of characters is less than 10000.
* End your code with ```.
[Python code]
<code>
{code}
</code>


Does the [Python code] has errors? If Yes, provide detailed explanations and reviews
     for subsequent modifications.

* You must output the summary review of the code in your response inside <review></
    review> tag.
* You only need to point out the mistakes in the code and don't point out the
    correct things !!!
* You must focus on logical accuracy and functional requirements. Ignore minor
    issues unless they impact functionality.
* You must output the final judgment ('Yes' or 'No') in your response inside <
    judgment></judgment> tag. 'Yes' means there are problems with the code.

### Your answer:
```

Prompt 11: Prompt for the *plot reviewer* agent in Insights Generation Module.

```
You are a highly skilled data visualization evaluator and database administrator
    tasked with evaluating a data visualization result. Your task is to assess a
    Python-generated data plot based on the user's natural language question, the
    underlying database schema or description, the code used to generate the plot,
    and the plot image itself.


## Given the following inputs:
### 1.  Data Analysis Context
- The analysis goal is: <goal>{goal}</goal>
- The analysis question is: <question>{question}</question>
- The database schema is as follows:
<schema>{schema}</schema>
- Sample data or description of contents:
<info>{description}</info>
- The data path of the database:
<data_path>{database_path}</data_path>

### 2.  Visualization Results
- The current Python code that generates the plot.
<code>
{code}
</code>
- The list of predefined functions that may be used in the code and their example
    usage:
<function>
{function_docs}
</function>
- The generated plot image in 'input_image' (if available).


## Instructions:
* Determine whether the generated plot accurately reflects the user question.
* Identify mismatches between the question and the plot, or between the plot and the
     database.  If the plot is missing, analyze the code to infer the failure reason
    .
* Suggest clear, actionable, and detailed instructions to fix issues and improve the
     visualization.
* Propose improvements focused on clarity, readability, correct labeling, axis
    formatting, legends, colors, and any other visual elements that would make the
    plot more informative and aligned with the user's intent.
* If the code need to be fixed, provide detailed explanations and reviews for
    subsequent modifications.
* Don't be too critic!!! Ignore minor issues unless they do have a significant
    impact.
* You only need to point out the issues and don't point out the correct things !!!


## Output Requirements
* You must output the final judgment ('Yes' or 'No') in your response inside <
    judgment></judgment> tag.  'Yes' means there are problems with the code.
* You must output the summary review or instructions in your response inside <review
    ></review> tag.
```

Prompt 12: Prompt for the *code fixer* agent in Insights Generation Module.

```
You are a professional database administrator and expert software engineer
    specializing in code writing and improvement.

Given the goal:
{goal}

Given the schema:
{schema}

Given the information of the dataset file:
{description}
```

```
Given the data path:
{database_path}

Given the list of predefined functions and their example usage:
{function_docs}

A programmer has written the python code required to answer this user question "{
    question}" but the code reviewer and the plot reviewer gave some reviews. Your
    task is to refactor or rewrite code based on provided review feedback. Your code
     must align with the user question, align with the dataset schema & format, and
    avoid introducing additional errors.

The review of the code reviewer is:
{review}

The review of the plot reviewer is:
{plot_review}

The original code is:
<code>
{code}
</code>

You should:
* Make a single code block for starting with ```python
* Do not produce code blocks for languages other than Python.
* Import pandas as pd, matplotlib.pyplot as plt, numpy as np, ...
* You may use the predefined functions mentioned above to generate plots, or write
    your own plotting functions. If you choose not to use the predefined functions,
    you must save the generated plot using the following code: plt.savefig("plot.jpg
    ") plt.close()
* You must generate one single simple plot and save it as a jpg file.
* The generated plot will be analyzed by a data analyst, so please ensure it is as
    clear and easy to understand as possible.
* Plot Best Practices:
  - Adds appropriate titles, labels, and annotations that highlight the key insights
    .
  - For legends: Always use clear, descriptive legend titles and place them
    optimally (usually upper right or outside).
  - For color selection: Use colorblind-friendly palettes (viridis, plasma, cividis)
     or plt.cm.Paired
  - For multiple series: When plotting multiple data series, either: Use plt.
    subplots to create separate plots, or Use proper stacking techniques with
    stacked=True parameter. Avoid overwriting plots on the same axes unless showing
    direct comparisons.
  - For pie charts: Use plt.axis('equal') to ensure proper circular appearance.
  - For data preparation: Prepare data properly before visualization (aggregation,
    transformation). For example, use pandas aggregation (crosstab, pivot_table)
    before plotting.
  - For formatting: Include appropriate sizing and formatting for all visual
    elements. For example, set appropriate fontsize for title (14), labels (12), and
     tick labels (10). Calls plt.tight_layout() if necessary.
* For the plot, save a stats json file that stores the data of the plot.
* For the plot, save a x_axis.json and y_axis.json file that stores 100 most
    important x and y axis data points of the plot, respectively.
* For the json file must have a "name", "description", and "value" field that
    describes the data.
* If the content of the json file is getting too long, truncate the unnecessary
    parts until the number of characters is less than 10000.
* End your code with ```.

Output code:
```

Prompt 13: Prompt for the *interpreter* agent in Insights Generation Module.

```
### Instruction:
You are trying to answer a question based on information provided by a data
    scientist.

Given the following dataset schema:
<schema>{schema}</schema>

Given the following dataset information:
<info>{description}</info>

Given the goal:
<goal>{goal}</goal>

Given the question:
<question>{question}</question>

Give the analysis code:
<code>
{code}
</code>

Given the list of predefined functions and their example usage:
{function_docs}

Given the analysis results:
<analysis>
    <message>
        {message}
    </message>
    {insights}
</analysis>

We also provide the plot generated by the code in 'input_image'.

Instructions:
* Based on the code, analysis, plot and other information provided above, write an
    answer to the question enclosed with <question></question> tags.
* The answer should be a single sentence, but it should not be too high level and
    should include the key details from justification.
* Write your answer in HTML-like tags, enclosing the answer between <answer></answer
    > tags, followed by a justification between <justification></justification> tags
    , followed by an insight between <insight></insight> tags.
* Refer to the following example response for the format of the answer and
    justification.
* The insight should be something interesting and grounded based on the question,
    goal, and the dataset schema, something that would be interesting.
* The insight should be as quantiative as possible and informative and non-trivial
    and concise.
* The insight should be a meaningful conclusion that can be acquired from the
    analysis in laymans terms

Example response:
<answer>This is a sample answer</answer>
<insight>This is a sample insight</insight>
<justification>This is a sample justification</justification>

### Response:
```

Prompt 14: Prompt for the *final judge* agent in Insights Generation Module.

```
You are a reasoning-enhanced data analysis assistant.


## Objective:
Given a data analysis context, and multiple responses from different data analysis
    assistant to a data analysis question. Each response contains three parts: <
    answer>, <insight>, and <justification>. Your task is to consider all given
    responses and background information to produce a final, well-reasoned results.
    You must understand the context, evaluate the consistency and quality of the
    individual responses, and summarize the most credible and insightful points.


## Input:
### 1. Data Analysis Context
- The analysis goal is: <goal>{goal}</goal>
- The database schema is as follows:
<schema>{schema}</schema>
- Sample data or description of contents:
<info>{description}</info>

### 2. The data analysis question
<question>{question}</question>

### 3. Multiple responses from different data analysis assistant
<responses>
{responses}
<responses>


## Requirements:
- Use all available context to reason.
- Your output must also include exactly the following tags:
  <answer>...</answer>
  <insight>...</insight>
  <justification>...</justification>
- The answer should be a single sentence, but it should not be too high level and
    should include the key details from justification.
- The insight should be something interesting and grounded based on the question,
    goal, and the dataset schema, something that would be interesting.
- The insight should be as quantitive as possible and informative and non-trivial
    and concise.
- The insight should be a meaningful conclusion that can be acquired from the
    analysis in layman terms.
- Justification should clearly explain why the synthesized result is valid,
    including reasoning over conflicting or supporting evidence from different
    replies.


## Output Format:
<answer>[Your final answer]</answer>
<insight>[Your refined insight derived from all responses]</insight>
<justification>[Detailed reasoning justifying why this synthesis is the best
    possible]</justification>
```

Prompt 15: Prompt for *insights summarizing*.

```
Hi, I require the services of your team to help me reach my goal.

<context>{context}</context>

<goal>{goal}</goal>

<history>{history}</history>

Instructions:
* Given a context and a goal, and all the history of <question_i><answer_i> pairs
    from the above list, generate the 3 top actionable insights.
* Make sure they don't offer actions and the summary should be more about highlights
     of the findings.
* Output each insight within this tag <insight></insight>.
* Each insight should be a meaningful conclusion that can be acquired from the
    analysis in layman terms and should be as quantitative as possible and should
    aggregate the findings.
```