NotebookRAG: Retrieving Multiple Notebooks to Augment the Generation of EDA Notebooks for Crowd-Wisdom

Yi Shan[1]* Yixuan He[1] Zekai Shao[1] Kai Xu[2]† Siming Chen[1]‡

[1]Fudan University, China
[2]University of Nottingham, UK

ABSTRACT

High-quality exploratory data analysis (EDA) is essential in the data science pipeline, but remains highly dependent on analysts' expertise and effort. While recent LLM-based approaches partially reduce this burden, they struggle to generate effective analysis plans and appropriate insights and visualizations when user intent is abstract. Meanwhile, a vast collection of analysis notebooks produced across platforms and organizations contains rich analytical knowledge that can potentially guide automated EDA. Retrieval-augmented generation (RAG) provides a natural way to leverage such corpora, but general methods often treat notebooks as static documents and fail to fully exploit their potentially knowledge for automating EDA. To address these limitations, we propose NotebookRAG, a method that takes user intent, datasets, and existing notebooks as input to retrieve, enhance, and reuse relevant notebook content for automated EDA generation. For retrieval, we transform code cells into context-enriched executable components, which improve retrieval quality and enable rerun with new data to generate updated visualizations and reliable insights. For generation, an agent leverages enhanced retrieval content to construct effective EDA plans, derive insights, and produce appropriate visualizations. Evidence from a user study with 24 participants confirms the superiority of our method in producing high-quality and intent-aligned EDA notebooks.

1 INTRODUCTION

Exploratory Data Analysis (EDA) [53] plays a crucial role in the data science pipeline. High-quality EDA is time-consuming and requires coding proficiency, statistical thinking, and visualization literacy [64], which places a heavy burden on analysts. To ease these issues, prior research has explored rule-based and reinforcement learning approaches for automated EDA [3, 65, 68]. With the rise of large language models (LLMs), automated EDA has gained powerful capabilities, enabling better alignment with user intent [33, 73] and comprehensive insight discovery [34, 71]. However, some approaches still face challenges in handling abstract intent, others may rely on comprehensive fact-checking of the dataset that can be inefficient and fragmented, and most remain limited in providing visualizations that effectively support analytical reasoning [7, 22].

In real-world data mining, the process is typically driven by a high-level predictive or descriptive goal [38]. While analysts can usually specify the overall objective at the outset (e.g., building a time-series model for price prediction), they often struggle to design targeted EDA plans that effectively support and operationalize the data mining task (e.g., first providing an overview of the price series and then analyzing its seasonal distribution). This reflects an abstract intent (performing EDA to prepare for a data mining task) that lies between explicit instructions (e.g., testing for significant periodicity) and no intent (e.g., simply asking to understand the data) [64].

In practice, such as in Kaggle competitions or enterprise analytics, the same dataset or data source is often explored repeatedly by many analysts, producing numerous computational notebooks [41] that share consistent data semantics and thus provide more relevant and reliable analytical knowledge for EDA [10, 47]. Interviews with senior enterprise analysts further confirmed this practice, revealing that analysts routinely revisit updated datasets and rely on existing notebooks to accelerate analysis and improve efficiency.

Retrieval-augmented generation (RAG) [26] provides a natural way to leverage such notebooks, but existing approaches face two main issues. For retrieval, notebooks are often treated as static documents rather than executable artifacts, causing retrieved content to become invalid as data evolves; additionally, cells are handled independently, which ignores contextual dependencies and degrades retrieval quality [27, 30, 31]. For generation, prior methods mainly target well-defined question answering [52] and are therefore ill-suited for automating EDA driven by abstract user intent.

To better leverage these corpora for automated EDA, we propose a method named NotebookRAG, which takes user intent, a dataset, and existing notebooks as input, first retrieving and enhancing relevant content from notebooks and then automatically generating EDA notebooks that integrate statistical analysis and visualization. For retrieval, code cells are enriched with contextual information, transformed into executable components, and annotated with the data columns they use. The user intent (e.g., conducting EDA to prepare for time-series modeling) is mapped into multiple EDA queries (e.g., examining average price by region), which are used to retrieve relevant components based on their used columns. These components are re-executed on new data to generate updated visualizations, from which reliable insights are obtained. For generation, we design an agent that produces EDA notebooks from the dataset and user intent, with an optional interface to incorporate retrieval outputs. Leveraging the enhanced retrieval content, the agent can construct more effective EDA plans, generate more appropriate visualizations, and derive insights that better support analytical reasoning.

To evaluate NotebookRAG, we conducted a within-subject user study with 24 participants using realistic Kaggle datasets, representative data mining tasks, and existing notebooks. Participants compared the quality of notebooks produced by the ChatGPT Data Analyst plugin [39], a baseline notebook generator, a general retrieval method [27], and our proposed retrieval method. In addition, we performed objective checks on notebooks generated by our method. NotebookRAG was rated significantly higher than the other approaches across most evaluation dimensions and received more positive qualitative feedback, demonstrating its ability to generate higher-quality notebooks that better align with user intent.

In summary, our contributions are concluded as follows:

- NotebookRAG, an automatic approach for generating efficient and effective EDA notebooks by combining user intent, datasets, and existing notebooks.
- A retrieval technique that extracts relevant content from existing notebooks and an agent that leverages this content to automatically

*E-mail: ydan24@m.fudan.edu.cn
†E-mail: kai.xu@nottingham.ac.uk
‡E-mail: simingchen@fudan.edu.cn. Corresponding author.

generate EDA notebooks.
- A user study demonstrating that NotebookRAG significantly outperforms baselines in generating higher-quality, intent-aligned EDA notebooks.

2 Related Work

In this section, we review computational notebooks, automating EDA, and insight generation.

2.1 Computational notebook

Computational notebooks have become widely adopted for data analysis owing to their interactivity, reproducibility, and visualization capabilities [4,24,41,47], leading to a vast number of notebooks hosted on public platforms such as GitHub [19, 30] and Kaggle [37, 45], as well as within enterprises [10]. By integrating executable code with rich documentation, notebooks serve both as computational environments and communication media; however, this dual role often leads to unstructured content, making notebooks harder to reuse and parse than traditional code files or analytical reports [6, 51, 54, 60].

Notebook Reuse. Some works improve the understandability of personal notebooks to facilitate reuse by cleaning up messy content [17] and enriching documentation with clearer explanations [5]. Others have explored alternative presentation formats, such as slides [55, 56], reports [59], or videos [40] for storytelling, as well as visualizations of notebook content and structure [63] to reduce the cognitive burden of reading notebooks. In addition, some studies have constructed large corpora of public notebooks [37, 45], which have been used to analyze real-world notebook practices [44] by statistically characterizing various notebook features, and to support tasks such as code recommendation [30, 31], model training [14], and fine-tuning [19, 28] by building mappings between code and natural language or extracting task-specific code sequences.

Relatedly, ReSpark [51] extracts analytical objectives from existing reports and adapts them to new datasets, similarly emphasizing the reuse of analytical intent. Improving the understandability of individual notebooks enables fine-grained local reuse, while corpus-based methods offer broader coverage; however, neither achieves both simultaneously. In contrast, our work reuses multiple notebooks analyzing the same source datasets, combining fine-grained reuse with the diversity of analytical strategies across notebooks.

Notebook Parsing. Notebook parsing is essential for understanding and analyzing notebook code, and existing methods generally fall into static and dynamic analysis. Static analysis examines code via Abstract Syntax Trees (AST), commonly treating each code cell as a unit and representing it by the variables or APIs it uses, with relationships established across cells [30, 31, 63]. Some approaches model the notebook as a linear yet segmented sequence under the assumption of sequential cell execution [19, 68], but they mainly focus on data wrangling and do not cover other EDA stages. Dynamic analysis monitors the notebook kernel to collect runtime records and variable states, enabling richer analyses [12, 16, 17, 67]. However, it relies on execution logs and is typically implemented as interactive plugins, making it unsuitable for already existing notebooks; re-executing notebooks to obtain such logs is also unreliable due to non-linear execution and low reproducibility [6, 42]. Therefore, we adopt static analysis and extend prior approaches.

2.2 Automating EDA

Recognizing that high-quality EDA typically requires substantial expertise from analysts, researchers have explored ways to reduce these requirements by developing EDA assistants or creating a fully automated EDA process.

EDA assistants. EDA assistants serve to support analysts during user-led analyses. Some works reduce the analyst's burden through code recommendations [30, 31] or visualization recommendations [25], while others ease the cognitive load by visualizing the EDA process [48, 63], which helps analysts conduct more effective EDA.

Fully Automated EDA. Fully automated EDA shifts the analyst's role from active exploration to interpreting generated content, significantly reducing workload. Rule-based methods [65, 68] provide strong controllability but lack flexibility, while deep reinforcement learning methods [2, 3, 35] demonstrate better generalization and are capable of generating end-to-end workflows. With the advances of LLMs, fully automated EDA has gained stronger capabilities: tools such as the ChatGPT Data Analyst plugin [39] can now effectively automate the entire EDA pipeline, and recent works further enable better alignment with user intent [33, 73] as well as more comprehensive insight discovery [34, 71]. However, these approaches still face key limitations: they rely on explicit user intent, analyze entire datasets inefficiently, and fail to generate visualizations that effectively support reasoning [7, 22]. Therefore, our work attempts to build an intelligent agent that leverages knowledge from existing notebooks to better handle these challenges.

2.3 Insight Generation from Visualizations

Statistical Methods. Visualization insight generation is a key component of visualization recommendation and composition. Before the emergence of LLMs, visualization insight generation primarily relied on statistical methods, which involved designing specific statistical schemes tailored to particular tasks and visualization types to identify significant insights [9–11, 50, 61]. With the advent of LLMs and their code-generation capabilities, reliance on predefined statistical functions has been reduced, enabling more generalizable insight generation while alleviating LLM hallucination issues [57, 62, 72].

VLM-based Methods. Moreover, as VLMs have advanced in visual understanding, generating insights via visualization-to-natural-language (vis2nl) methods has gained increasing attention [20]. Several studies have shown that when VLMs are provided with clearly labeled and standardized visualizations, they perform well on chart question tasks [66], chart captioning tasks [32], and visualization literacy tasks [49]. Recent work also shows that combining data with its visualization further enhances VLM performance on broader data analysis tasks [29]. However, hallucination remains a pervasive challenge, often leading to factual errors or ambiguous semantics [1, 23]. To mitigate this issue, prior studies have explored strategies such as converting charts into structured tables for consistency checking [21], and constructing curated chart–caption datasets for fine-tuning [32].

To obtain reliable insights from visualizations in notebooks, we design a hybrid approach that combines statistical methods and VLM-based methods, where insights are first extracted using VLMs and then verified and refined by LLM-generated statistical code.

3 Formative Study

Through interviews with four senior enterprise analysts (each with over 10 years of experience), we confirmed that such scenarios occur not only on public platforms but also in enterprise settings, where multiple notebooks exist for the same or closely related datasets, and analysts often consult them to guide their work. This reinforced our belief that RAG could enhance automated EDA. To specify the design requirements for a RAG-based pipeline, we conducted one-on-one interviews with 12 master's and PhD students in data science who regularly use computational notebooks. The study examined how they reuse notebooks and interact with automated EDA tools to identify key design requirements for integrating the two.

3.1 Procedure

The session began by asking participants to recall their past experiences conducting data analysis with notebooks, particularly whether they reused notebooks created by others. All participants confirmed doing so, noting that reuse significantly improves their efficiency. We then provided them with a commonly used Kaggle dataset, a

data mining task, and five highly upvoted notebooks, encouraging them to browse these notebooks to gain a targeted understanding of the data. Participants were asked to think aloud during this process, allowing us to observe how they used them. Next, we introduced the ChatGPT Data Analyst plugin [39], a representative generative tool capable of performing automated EDA tasks. Using the same dataset along with several preset prompts, participants experienced the process and results of using the plugin for exploratory data analysis and were invited to provide comments and critiques. Building on these, we introduced the concept of RAG, positioning it as analogous to reusing prior analyses during generation. Finally, we presented our hypothetical RAG-based pipeline for EDA notebook generation and engaged participants in identifying concrete design requirements.

3.2 Design Requirements

We summarized the interview recordings and confirmed the design requirements (**DRs**) with the participants as follows:

DR1: Goal-Aligned Extraction and Enhancement. When observing how participants used notebooks, we found that without prior knowledge of the dataset, it was difficult for them to quickly distinguish relevant parts. Most reported that reading notebooks sequentially was time-consuming, and agreed that automatically filtering potentially useful content would be valuable. Several participants (6/12) mentioned that well-documented markdown notes greatly improved their understanding. All participants acknowledged that when data became outdated, the factual results in notebooks lost their validity, leaving only the analytical strategies still applicable, which reduced the value of existing notebooks. These observations suggest that the retrieval stage should extract content aligned with users' specific analytical goals and enhance it with new data to obtain updated results and corresponding explanations, thereby improving the relevance and interpretability of reused content.

DR2: Efficient and Effective Results. Several participants (5/12) pointed out that GPT's outputs were "standard but lacked depth," often limited to simple visualizations with minimal analytical insights, which made the initial results frequently unsatisfactory. Although GPT could be prompted to continue analyzing, this process was inefficient compared to directly obtaining richer visualizations that facilitate data understanding. These findings highlight the need for automatically generated EDA results that, within limited steps, provide a coherent, comprehensive, and in-depth analytical process and deliver valuable insights supported by appropriate visualizations.

DR3: Flexibility and Robustness. Several participants (4/12) with prior RAG experience pointed out that one cannot always expect valuable information from retrieved notebooks. In particular, they noted that relevant notebooks may not exist, especially for private datasets, or that available notebooks may be of low quality. This highlights the need for a system that can flexibly leverage retrieval when available while remaining robust and producing reasonable outputs even when retrieval provides limited or no support.

DR4: Seamless Code Reuse. A majority of participants (9/12) noted that although GPT's outputs revealed the code used to generate visualizations, the code could not be directly modified on the page. They also pointed out that some code blocks were incomplete, with parts of the context hidden in memory rather than explicitly provided, which made reuse inconvenient. Most participants agreed that producing the final output as an executable notebook would be a more practical solution, as it allows free modification and continuation of analysis. Therefore, the generated EDA results should take the form of notebooks, enabling analysts to modify and re-execute independent code cells for further exploration or downstream tasks.

4 NotebookRAG

In this section, we first give an overview of NotebookRAG pipeline and two key steps: notebook retrieval (Fig. 1) and generation (Fig. 3).

4.1 Pipeline Overview

The pipeline takes as input (i) a tabular dataset, (ii) a collection of notebooks based on different versions of the same underlying data source, such as annual updates of a company's revenue dataset, and (iii) a user intent, expressed in natural language, that specifies the subsequent data mining task (e.g., building a time-series model for price prediction), for which the system automatically constructs corresponding EDA notebooks as a preparatory step (as shown in Fig. 1-a). The pipeline proceeds in two stages:

Notebook Retrieval. Notebooks are first decomposed into code cells and markdown cells, since they serve distinct roles: code cells contain executable logic that produces data transformations and visualizations, while markdown cells capture human-authored explanations of analytical intent and conclusions. Code cells are transformed into executable components annotated with used columns and chart types, while markdown cells are converted into embeddings (Fig. 1-1). The user intent is mapped into a set of EDA queries (Fig. 1-b) based on LLMs, which are then used to retrieve relevant content: queries are matched with markdown embeddings for semantic similarity, their associated columns are used to search candidate components, and further filtered by chart type (Fig. 1-2) (DR1). Then, retrieved components are re-executed on the user-provided dataset, and a VLM extracts task-relevant insights from the resulting visualizations (Fig. 1-3). These insights are then verified and refined with LLM-generated statistical code, which corrects factual errors and clarifies ambiguous statements to yield reliable insights (DR1).

Notebook Generation. We built an agent that automatically generates EDA notebooks from the dataset and user intent, with an optional interface to ingest retrieval outputs (DR3). The agent begins by constructing an EDA plan that specifies the goals and methods of each step, and then incrementally generates the corresponding visualizations and insights to produce a coherent, structured, and runnable notebook (DR4). Incorporating retrieved content enhances both the planning and generation stages, making the constructed plans more efficient and effective, the analysis code more in-depth, and the visualizations more appropriate (DR2).

4.2 Notebook Retrieval

4.2.1 Component Extraction

We define the **Component** as shown in Fig. 2 as a self-contained, executable unit with resolved data and environment dependencies. Each notebook code cell that produces visualizations is transformed into such a component. This design addresses two key challenges: (1) understanding the functionality of a single code cell often requires tracing implicit data dependencies and transformations; and (2) outdated dependencies and disorganized structure can easily lead to bugs, while debugging in a notebook environment is significantly less efficient than working with a continuous code block. Considering the low reproducibility of public notebooks [42], we adopt a *static analysis* approach rather than a dynamic one, thereby enabling a more general and robust method. Static analysis avoids execution-time failures caused by missing data or outdated dependencies, which are common issues in public notebooks. Our method supports both sequential and non-linear notebook executions, with the latter requiring a complete execution log. In the following, we describe our method assuming sequential execution.

We first merge code cells that generate visible outputs (e.g., plots or tables) or are followed by markdown cells with their preceding non-output cells. To construct components, we introduce the **data variable**, which is any variable directly or indirectly derived from raw data. As illustrated in Fig. 2, data variables follow a lifecycle that includes generation (e.g., S1 creates *df*, S2 and S4 create versions of *df1*), modification (e.g., S3 and S5 update *df1*), and downstream usage (e.g., S6 consumes *df1* for visualization). We then track these data-variable dependencies across the notebook in

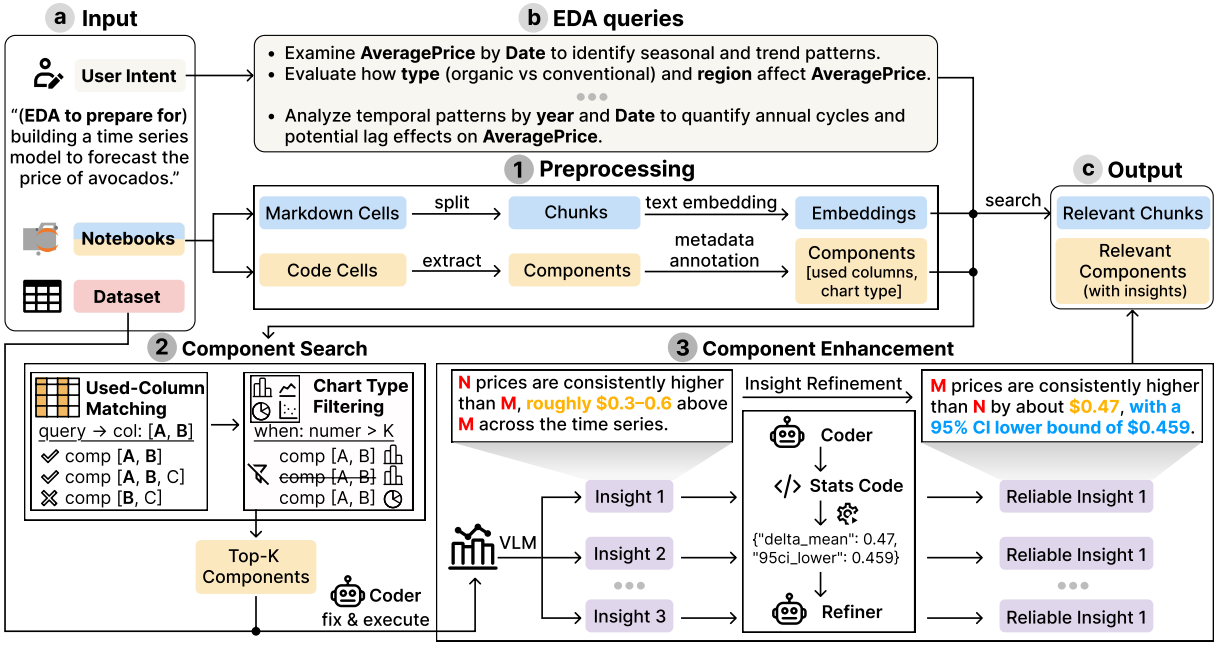

Figure 1: Overview of the Notebook Retrieval process. User intent (a) is mapped into EDA queries (b). Notebook markdown cells and code cells are preprocessed (1) into embeddings and components (with metadata), respectively. Then, the EDA queries are used to search embeddings for relevant chunks (c) and to guide component search (2) and enhancement (3), producing relevant components (c).

a top-down manner and resolve them at the cell level. Each statement that creates, modifies, or uses a data variable is identified, and its dependency chain is recursively traced. In this way, we record for every data variable the complete sequence of statements that describe its evolution from raw input to its current state (see Data Variables and Cell Dependencies in Fig. 2). Finally, to construct the component, we prepend the minimal set of required statements for the data variables in a target code cell, preserving execution order to form a self-contained, executable unit.

In practice, we implement this process through an AST-based algorithm, which is robust to real-world complexities such as branches, loops, and function calls, and leverages a taxonomy of common data-processing libraries (e.g., `pandas`) to recognize implicit state mutations (e.g., method calls like `df.drop_duplicates()`).

4.2.2 Component Metadata Annotation

Given an EDA query (e.g., "evaluate how type affects price"), semantic similarity search often performs poorly because of the semantic gap between natural language and code, the lack of method-level details, and the fact that many visualization functions (e.g., `sns.pairplot`) reference columns implicitly rather than explicitly [15, 65]. To improve retrieval precision, we leverage the code understanding capabilities of LLMs to annotate each code snippet with the used columns and chart type (provided with dataset column descriptions). Notably, the LLM-based annotation does not rely solely on explicit variable names. It infers column dependencies based on the semantic context of the code, such as understanding the effects of data transformations, thereby maintaining accurate mappings between queries and transformed variables (Fig. 2-Component).

To evaluate this approach, we manually annotated 840 pairs of code cells and corresponding components with metadata and compared the labeling accuracy of different models (Tab. 1). For **chart type**, all models achieved high accuracy, with little difference between code cells and components. For **used columns**, however, we observed clear improvements when annotating components: across both SOTA models and smaller models, component-level annotation consistently outperformed cell-level annotation. The advantage of

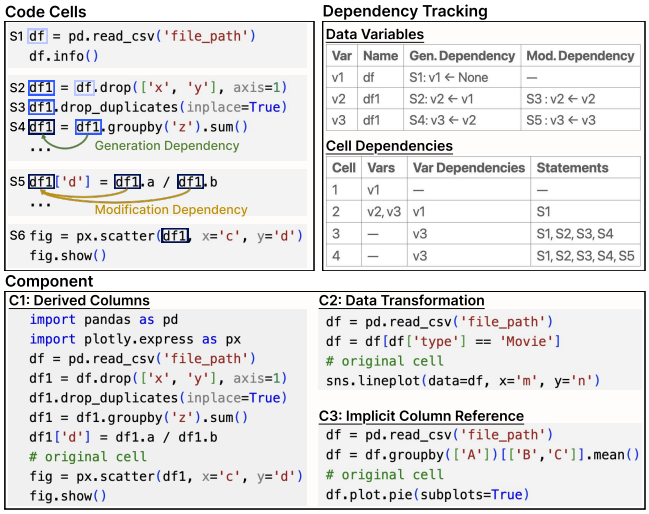

Figure 2: Example of the dependency tracking process. **Code Cells** define data variables through generation and modification statements. **Data Variables** summarize each variable's lifecycle and dependencies. **Cell Dependencies** show how variables propagate across cells. **Component** collects the required statements into a self-contained unit that can be executed independently. Some cases where components improve annotation accuracy (C1, C2, C3).

components comes from preserving complete contextual information. We highlight some common cases: (1) the use of previously created derived columns (Fig. 2-C1), (2) data transformations applied upstream (Fig. 2-C2), and (3) visualization methods that implicitly reference columns without explicitly naming them (Fig. 2-C3).

Practical Note. While **gpt-5-nano** achieved the best trade-off between accuracy and efficiency, **Qwen-7B** offers a practical alternative for large-scale annotation: when deployed on a single RTX 4090 GPU with 8 threads, it completed all 840 annotation tasks within one minute.

Table 1: Accuracy of used-column annotation on code cell–component pairs. Δ (pp) denotes absolute percentage-point gain; Δ Rel. (%) denotes relative improvement.

Model	Acc-Code (%)	Acc-Comp (%)	Δ (pp)	Δ Rel. (%)
gpt-5	78.1	91.4	13.3	17.1
gpt-5-nano	76.0	89.6	13.6	18.0
Qwen-7B	63.5	74.0	10.5	16.7
CodeL-7B	57.3	68.0	10.7	18.7
DeepSeek-6.7B	58.8	67.5	8.7	14.8

Abbreviations: Qwen-7B = `Qwen2.5-Coder-7B-Instruct`; CodeL-7B = `CodeLlama-7b-Instruct`; DeepSeek-6.7B = `Deepseek-Coder-6.7B-Instruct`.

4.2.3 Intent-Guided Retrieval

Since user intent can often be abstract or ambiguous, we adopt an LLM-assisted exploratory strategy for intent interpretation and retrieval (as shown in Fig. 1), inspired by question-guided insight generation [34]. Specifically, we prompt LLMs with contextual information such as the dataset description to decompose the intent into a set of possible EDA queries (as shown in Fig. 1-b), aiming to cover potentially valuable analytical sub-tasks to enhance the completeness and diversity of retrieval. Each query specifies target columns and analytical goals, which are then used to retrieve relevant content from both markdown cells and code components. For markdown cells, considering the diversity of their content such as analytical objectives and conclusions as well as their unstructured narrative forms, we adopt a conventional approach that segments the text and performs embedding-based similarity retrieval to identify potentially relevant passages. In particular, markdown content is excluded when data versions differ, which may lead to outdated conclusions. For components, we leverage metadata to match the columns specified in the query: exact matches are prioritized, followed by partial matches that cover most of the query columns. If the number of candidates exceeds the top-k (set to five), we further filter them by retaining only one component for each unique combination of used columns and chart type (as shown in Fig. 1-2).

4.2.4 Component Enhancement

Since visualizations in the original notebooks may be invalid due to outdated data, and some components may contain obsolete code that prevents direct rerun, we enhance the retrieved components after top-k selection (Fig. 1-3). Specifically, we provide each component's code together with the user's dataset to a *Coder*, which executes the code in a sandbox environment and automatically repairs errors, thereby producing runnable code and updated visualizations.

Given the strong performance of SOTA VLMs on various vis2nl tasks [32, 49, 66] and the demonstrated benefits of visual inputs for data analysis [29], we rely on human-authored visualizations designed to support analytical reasoning to obtain insights that are difficult to derive from code or statistics alone. We provide the user intent and the visualization images to a VLM, which produces actionable insights [59], which reflect an understanding of the data and support informed decision-making in real-world contexts.

Considering that VLMs may produce hallucinations leading to factual errors in insights, or semantic ambiguities due to limited information in the visualization [20, 57, 69], we introduce a refinement step. Specifically, the *Coder* generates statistical code based on the content of the insight to compute precise data facts, and the *Refiner* integrates the original insight with these results to correct errors and clarify ambiguities, ultimately producing reliable insights. For example, as shown in Fig. 1-3, "Insight 1" initially contained a factual error (stating $N > M$ instead of $M > N$) and a vague description ("roughly \$0.36–0.6"). After refinement, the factual error was corrected ($M > N$), the ambiguity was resolved (\$0.47), and additional information was provided (a 95% CI lower bound of 0.459). A small-scale manual review further indicated that this refinement process effectively mitigates hallucinations by improving both factual accuracy and interpretive clarity.

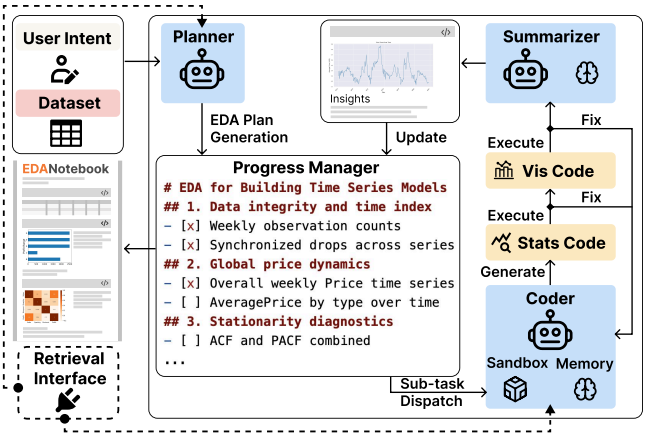

Figure 3: Agent for EDA notebook generation with retrieval interface. The agent takes user intent, a dataset, and retrieval outputs (optional) as input. It begins by constructing an EDA plan (Planner), then incrementally dispatches sub-tasks (Progress Manager), generates statistical code and visualization (Coder), and integrates results (Summarizer) to update the plan, ultimately producing a complete EDA notebook. Retrieval outputs are passed through a retrieval interface to enhance both planning and code generation.

4.3 Notebook Generation

As shown in Fig. 3, we build an agent to automatically generate EDA notebooks, inspired by the design paradigm of intelligent agent widely adopted in prior work [13, 58, 70], where a high-level plan is first generated, specialized tools are then invoked to complete sub-tasks, and global state is maintained through a centralized progress tracker to ensure coherent execution.

Given a dataset and user intent, the Planner formulates a natural language EDA plan specifying the goals and methods of each analytical step. The Progress Manager tracks and updates the plan to maintain coherence and assigns sub-tasks to the Coder. The Coder first generates statistical code for the sub-task and executes it to obtain results, and then uses these results together with the sub-task descriptions to generate visualization code. All code is executed in a sandbox environment to ensure robustness and safety, and the Coder is equipped with memory to preserve continuity across code fragments generated for different sub-tasks. The Summarizer, also equipped with memory, organizes the Coder's outputs into units containing visualizations and actionable insights, and then submits them to the Progress Manager for updates. Once the plan is completed, the Progress Manager generates a final summary report together with recommendations for subsequent tasks and assembles all content into a coherent EDA notebook.

A key feature of our pipeline is the **Retrieval Interface**, which enriches the agent with reusable notebook components, reliable insights, and relevant markdown derived from the Notebook Retrieval stage (Sec. 4.2). During plan construction, the Planner can ingest these retrieval outputs as additional context. To support more general scenarios, we provide resources with descriptions and inform the Planner that they may refer to them, without explicitly instructing how to use. This implicit guidance already yields substantial improvements in both the efficiency and effectiveness of the generated plans, as we demonstrate later in the case study (Sec. 5). During code generation, the Coder selectively reuses validated components based on their alignment with the sub-task goal and visualization suitability, while also using them as references when direct reuse is infeasible, producing visualizations that are closer to human-authored designs.

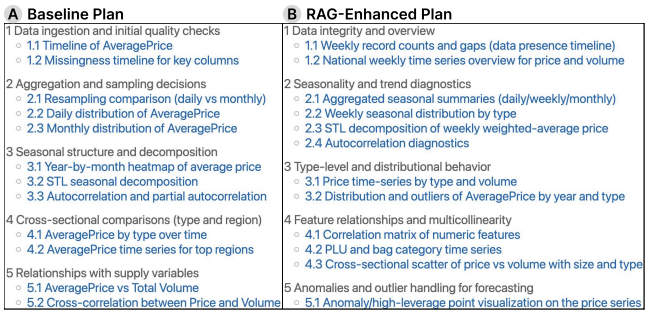

Figure 4: A: Baseline EDA plan. B: RAG-enhanced EDA plan.

To support traceability and source transparency, whenever components are reused, the generated notebook includes links at the corresponding positions that allow users to jump to the original notebook context. This enables analysts to inspect the source and discover potentially useful details such as drill-down analyses.

The retrieval interface is optional. When no existing notebooks are available, the agent simply proceeds without retrieval and can still generate complete EDA notebooks. When existing notebooks are available, the interface is always activated: high-quality notebooks enhance planning and generation, while low-quality or less relevant notebooks are filtered during the retrieval stage, thus having limited impact and ensuring the robustness of the overall pipeline.

4.4 Implementation

We use gpt-5-mini for extracting insights from visualizations and gpt-5-nano for text reasoning and code generation. For semantic retrieval, notebook markdown cells are encoded using text-embedding-3-large and indexed with FAISS for similarity search. The overall workflow is orchestrated with LangChain, enabling modular agent construction and graph-based control of planning, execution, and summarization.

5 Case Study

In this section, we qualitatively analyze the impact of incorporating retrieval on notebook generation. We first introduce a concrete usage scenario to set the stage, and then present several representative cases that highlight how retrieval affects both the overall EDA workflow and individual sub-tasks.

Consider an analyst aiming to build a time-series price prediction model on the Kaggle *Avocado Prices (2020)* dataset. Before modeling, the analyst wishes to explore the data by consulting existing notebooks. As this dataset is relatively new and has limited community analysis, whereas the earlier *Avocado (2018)* dataset has been extensively studied, with many notebooks publicly available. Manually reviewing these notebooks is time-consuming, as the analyst must re-run them on the new dataset, debug any errors encountered, and then sift through large amounts of content to identify the parts that are actually useful. With our system, the analyst only needs to download several highly upvoted notebooks from the old dataset and provide them with the new dataset and the user intent ("build a time-series model to forecast avocado prices"), as inputs. The system then generates EDA notebooks under two modes: with retrieval (RAG-enhanced agent) and without retrieval (baseline agent). Incorporating retrieval content improved performance at both the global level (overall EDA plan) and the local level (individual sub-tasks), as illustrated by the representative cases below.

Global Level. At the global level, the RAG-enhanced agent generated plans with clearer analytical priorities. Since LLMs operate as black boxes, our conclusions about how specific retrieval outputs influenced the final results are primarily derived from comparative analysis. For example, when many retrieved insights indicated that "certain attributes exhibit clear annual seasonality," the **RAG plan** (Fig. 4) prioritized *Seasonality and trend diagnostics* as an early step (Step 2), whereas the Baseline plan delayed this analysis until later

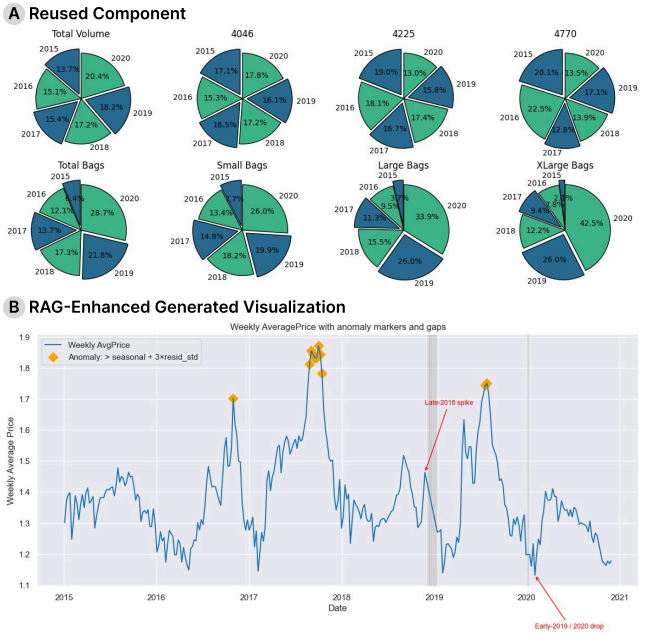

Figure 5: A: a reused human-authored component showing the year-by-year distributions of features. B: a RAG-enhanced visualization highlighting anomalies and annotated events in the weekly average price series.

(Step 3). By introducing this focus earlier, the RAG plan was able to perform a more fine-grained decomposition of seasonal and trend patterns within limited steps. Similarly, when retrieved insights emphasized "anomalies" and "correlations," the RAG plan explicitly dedicated sections to *Feature relationships and multicollinearity* (Step 4) and *Anomalies and outlier handling for forecasting* (Step 5). In contrast, the **baseline plan** lacked explicit anomaly handling and only included a simpler correlation analysis in terms of cross-correlation with supply variables. These enhancements in the RAG plan not only lead to a more structured and comprehensive workflow but also provide results that are directly useful for downstream tasks such as feature engineering in predictive modeling.

Local Level. At the local level, the improvements are reflected in both the richness and appropriateness of visualization forms and the depth of analysis. Compared to the baseline, the RAG-enhanced agent produced more diverse outputs, such as maps, pie charts, and interactive visualizations. This diversity is enabled by the availability of reusable components. For example, at the beginning of the data overview, the Coder chose to reuse a component (Fig. 5-A) that visualizes the *year-by-year distribution across key features*. Through carefully designed, human-authored encodings, this visualization compactly presents multiple distributions in a limited space, allowing users to quickly obtain an overview of the dataset and facilitating more efficient exploration. RAG-enhanced agent also adopted deeper analyses and more suitable presentation forms in certain cases. For instance, when insights highlighted anomalies such as "One pronounced anomaly (late-2018/early-2019): price spikes averaging 2.56 versus 1.38 during non-spike periods, occurring in low-volume periods" or seasonal fluctuations such as "AveragePrice shows seasonal fluctuations, rising to a peak price of 3.25 in 2016, then declining with a -0.02 delta from 2015 to 2018 and -0.07 from 2018 to 2020," the Planner explicitly instructed the analysis of "gaps" and defined anomaly points as "weeks where price exceeds the seasonal component by more than 3 × residual_std." Based on these instructions, the Coder generated statistical code to identify anomalies, and then used these results to produce visualization code that not only marked the anomaly points but also incorporated appropriate annotations, highlighting events such as the late-2018 price

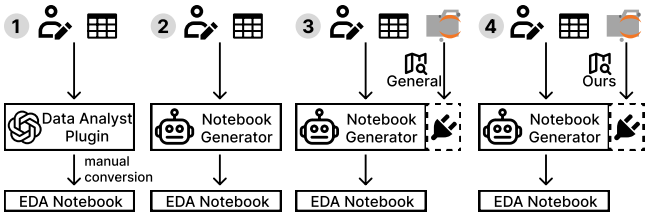

Figure 6: Overview of the study design. Four EDA notebooks were generated for evaluation: (1) using the ChatGPT Data Analyst plugin with manual conversion to an `ipynb` file, (2) using the baseline notebook generator without retrieval, (3) using the baseline generator with a general retrieval method, and (4) using the baseline generator with our proposed retrieval method.

spike and the early-2019/2020 price drop (Fig. 5-B). The system also provided actionable insights, such as "missing-week gaps: enforce a continuous weekly index and impute with seasonally-aware methods before ARIMA/LSTM," which guided the subsequent analysis and suggested improvements for modeling. This integration of statistical analysis and visual annotation provides a clearer and more informative representation of the data patterns.

6 User Study

6.1 Datasets & Materials

To ensure that our user study closely reflects real-world scenarios, we carefully prepared the experimental materials along three dimensions: datasets, tasks, and notebooks.

Datasets. We selected three commonly analyzed Kaggle datasets that represent typical scenarios frequently encountered by analysts: *Avocado Prices* (continuously updated), *Superstore* (available in many versions), and *Netflix Movies and TV Shows* (continuously updated). These datasets were chosen because they not only appear repeatedly in public analysis platforms but also mirror situations where similar or updated datasets arise in enterprise settings.

Tasks. We provided participants with a description of a data mining task as their user intent and asked them to evaluate whether the generated EDA notebooks could help them better prepare for the task. Because participants might not be familiar with the datasets, the task was predefined rather than self-formulated, ensuring comparable analytical objectives and fairer cross-session evaluation of retrieval and generation outcomes. To capture different analytical goals, we included one predictive task and one descriptive task for each dataset. The tasks were selected based on the most common analyses observed on these datasets, ensuring that the retrieval mechanism would have relevant prior material to draw from.

Notebooks. For each dataset, we collected the top 20 Python notebooks ranked by upvotes on Kaggle. This selection strategy aligns with the way users in real scenarios tend to prioritize higher-quality notebooks, while the number (20) is much larger than what would normally be examined manually, and also approximates the number of notebooks realistically available to analysts.

Together, these choices align the study design with realistic analytical contexts while providing a controlled evaluation setting, using three datasets with two tasks each.

6.2 Study Design

To systematically evaluate our approach, we generated notebooks for each dataset-task combination using four methods, with one notebook produced per method, as illustrated in Fig. 6.

ChatGPT Data Analyst Plugin (*ChatGPT*). The first notebook was produced using the ChatGPT Data Analyst plugin. We provided the user intent (i.e., a data mining task with the explicit goal of performing EDA to gain targeted understanding) and the dataset, and asked the plugin to generate a complete EDA process. The generated process was then converted into an `ipynb` file, with minor manual corrections applied to ensure consistency with the origin and to match the format of notebooks generated by our method.

Baseline Notebook Generator (*Baseline*). The second notebook was directly generated by our baseline notebook generator without using any retrieval, serving as the baseline for comparison.

General Retrieval (*RAGBaseline*). The third notebook incorporated a general retrieval method. Specifically, markdown cells and code cells from prior notebooks were separately embedded, and the user intent was used to perform semantic similarity search. The retrieved content was then passed into the retrieval interface, guiding the notebook generator to produce the final EDA notebook.

NotebookRAG (*Ours*). The fourth notebook used our proposed retrieval method, where retrieved components were passed into the retrieval interface and integrated into the generation process, enabling the system to reuse human-authored content and produce enhanced EDA notebooks.

ChatGPT is a well-established product and is used as a reference point without extensive prompting. *Baseline* denotes our generator without retrieval, which, with engineering on gpt-5-nano, we estimate to be more powerful than *ChatGPT* and thus serves as a stronger baseline and an ablation of our retrieval component. *RAGBaseline* augments *Baseline* with a general retrieval method, against which we compare *Ours* to evaluate the effectiveness of our proposed retrieval approach. To ensure fair comparison, all methods were constrained by the same moderately relaxed step limit, enabling evaluation under comparable exploration budgets.

6.3 Methods

We employed a within-subjects design to evaluate the notebooks produced by the four methods. Unlike data analysis tasks that can be evaluated using objective metrics such as accuracy [28], objective measures like the correctness of generated visualizations [18] cannot comprehensively assess the overall quality of EDA. Therefore, following most prior studies on automated EDA [33, 34, 73], we adopted human evaluation and further extended existing evaluation dimensions based on the identified design requirements. Participants received the materials and rated in a questionnaire consisting of 13 statements, as is shown below, each corresponding to an evaluation dimension. The quality sub-dimensions were designed by two external experts to ensure impartiality. Responses were collected using a five-point Likert scale ranging from *strongly disagree* to *strongly agree*. In addition, participants were encouraged to provide justifications or think-aloud comments for each rating.

Overall Dimensions. (1) ***Confidence:*** I am confident in the validity and reliability of the analysis. (2) ***Helpfulness:*** The EDA is helpful in exploring the data and supports me effectively in understanding it. (3) ***Satisfaction:*** The analysis meets my expectations and leaves me satisfied with its overall quality and usefulness. (4) ***Quality:*** N/A (This dimension is not measured by a single scale but is computed as the average of the ten quality sub-dimensions.)

Quality Sub-dimensions. (1) ***Task Alignment:*** The analysis closely aligns with the stated data mining task and research goals. (2) ***Data Comprehension:*** The notebook demonstrates a thorough understanding of the dataset, including missing values, outliers, and data quality issues. (3) ***Coverage:*** The exploration covers a sufficient range of variables and their relationships (univariate, bivariate, multivariate). (4) ***Visualization:*** The visualizations are appropriate, clearly presented, and helpfully support interpretation. (5) ***Methodology:*** The statistical methods used are suitable, clearly explained, and properly interpreted. (6) ***Insight:*** The notebook generates meaningful and non-trivial insights beyond simple descriptive summaries. (7) ***Robustness:*** The analysis identifies and discusses potential biases, anomalies, or limitations in the data. (8) ***Narrative:*** The narrative and explanations are coherent, logical, and easy to follow. (9) ***Reproducibility:*** The notebook is reproducible, with clear code,

documented steps, and environment/dependency specifications. (10) ***Efficiency:*** The analysis is efficient and concise, avoiding redundancy while maximizing insight.

Evaluations were conducted under natural conditions, with a reasonable time limit of three days but without further restrictions or supervision. To minimize bias, we employed a blinding procedure. Participants were unaware of the study's purpose or our work and were instructed to evaluate only the four notebooks provided. The notebooks were formatted similarly (see supplementary material), preventing participants from inferring their origin and ensuring a fair comparison focused on quality. Although the notebooks were anonymized, the order of presentation in the material folder could influence the sequence in which participants read them. To control for order effects, we applied a balanced Latin square design to counterbalance the notebook order. This design required at least four participants for each dataset-task combination. Consequently, we recruited 24 participants across the six combinations in our study.

To further validate the fine-grained performance of our approach, two co-authors conducted objective checks on the notebooks generated by *Ours*, examining both the coverage of task-relevant key variables and the correctness of the generated analytical insights.

6.4 Participants

Because the study focused on notebook quality, participants were required to have at least three years of data analysis experience and to have conducted analyses regularly (at least once per month in the recent past). All participants were also required to be frequent computational notebook users. Recruitment was conducted via peer recommendations on social media to ensure appropriate qualifications, which were further validated through examination of their qualitative feedback. One participant exhibited weak reasoning, prompting us to recruit an additional participant as a replacement. The final dataset consisted of evaluations from 24 participants.

6.5 Results

Since the code output was already run and displayed as in shared notebooks, several participants mentioned that they just read the content. However, most participants still executed the notebook cells themselves. Among them, some ran only a few cells to verify the outputs before reading, others executed all cells, and some modified the code for further exploration. In terms of reading behavior, some participants first navigated the notebook using the "outline" function in their environment and jumped to sections of interest, while others read sequentially. Many noted that even though their task was merely to evaluate notebook quality, the way they interacted with the notebooks closely mirrored how they would use them if they were genuinely conducting EDA, such as reusing code directly, checking intermediate outputs, or merely exploring insights. This behavioral alignment supports the representativeness of our participant group.

The ratings of the four notebooks are shown in Fig. 7. Each bar shows the mean rating across participants. To handle within-subjects variables, we estimated 95% confidence intervals using the Cousineau-Morey method [8, 36]. Pairwise Wilcoxon signed-rank tests with Holm-Bonferroni correction were applied to assess significance. We conducted all the six pairwise tests, resulting in slightly conservative significance estimates.

Across the four overall dimensions, *Ours* consistently and significantly outperforms the other three notebooks. As overall scores provide a high-level summary, participants often reiterated similar rationales within the corresponding sub-dimensions. Due to space constraints, we focus directly on the most relevant sub-dimensions, presenting representative quotes and discussing the key mechanisms underlying the observed improvements. To avoid redundancy, we exclude *ChatGPT* due to its uniform inferiority across dimensions.

- ***Task Alignment.*** Participants consistently highlighted that *Ours* showed strong task alignment and analytic depth, noting it "defined objectives early" and "adhered closely to prediction goals." They felt it "fully addressed the task of comparing category/subcategory sales and profits, forming a complete chain", making it the most aligned among the four notebooks (regarding one of the tasks), in contrast to *Baseline* being "generic" and *RAG-Baseline* sometimes "lacking connection to decision scenarios." This is further supported by our objective check, which confirmed that *Ours* covered all task-relevant key variables.
- ***Visualization.*** *Ours* offered "diverse and well-matched chart types" and annotations that made interpretations "clearer and easier to follow." Several noted that it conveyed "more information without being unreadable." They also appreciated that *Ours* avoided readability problems seen elsewhere, such as heatmaps being "distorted by extreme values." (*Baseline*) or messy lineplots under extreme values (*RAGBaseline*). Participants further praised *Ours* for matching visuals to analytic goals, using nested donut charts, faceted scatterplots, and annotated time-series, making it the most effective among the four, even if some charts (e.g., percentage plots or the t-SNE) required more effort to digest. The effectiveness stemmed from referencing visualization designs in existing notebooks, which better align with human analytical reasoning.
- ***Methodology.*** *Baseline* relied heavily on descriptive statistics and was repeatedly described as "mentioning ANOVA without proper explanation," lacking hypothesis testing or deeper inference. *RAG-Baseline* offered a "coherent workflow" with some time-series checks and proportioning or bucketing methods, though its statistical validation remained limited. In contrast, *Ours* incorporated richer techniques such as "bootstrap and regression analysis" and "decomposition and autocorrelation diagnostics," which participants felt addressed the sub-problems more directly. *Ours* was consistently seen as applying more advanced and appropriate methods than the other notebooks. This is because the retrieved content provided relevant analytical examples, enabling the LLM to select and apply more appropriate statistical methods.
- ***Insight.*** *Ours* was consistently praised for offering "deeper and more meaningful insights," often supported by "specific numbers" such as percentage breakdowns or correlations, and further strengthened by "actionable recommendations." Participants noted that it moved beyond surface observations to examine underlying causes, identify fine-grained scenarios, and provide interpretations "closely tied to decision-making." In comparison, *Baseline* was frequently described as "shallow," offering largely descriptive summaries such as "organic prices are higher," while *RAGBaseline* supplied numerical evidence but remained "descriptive" and lacked concrete strategies. *Ours* stood out for producing more useful insights. In addition, our objective correctness check identified a low rate of factual inconsistencies (8/319) across all notebooks generated by *Ours*. This is largely attributed to the agent's autonomous ability to analyze and adjust its reasoning, leading to more refined and decision-oriented insights.
- ***Reproducibility.*** On reproducibility, the study revealed no significant differences for *Ours*. Our intention was to ensure the codes executed normally so participants could assess their quality, making this outcome more a reflection of externally designed scales than of differences. Nevertheless, we retain this sub-dimension to demonstrate that our pipeline produces notebooks that run reliably.
- ***Efficiency.*** Participants frequently praised *Ours* for having "no redundant content," offering "richer insights with less repetition," and achieving the "highest information density." *Baseline* was often described as "too detailed and repetitive." While *RAGBaseline* was seen as relatively concise, its insights were sometimes lengthy without focus. *Ours* most effectively condensed analysis while still exploring multiple perspectives and maximizing insight value. It results from the agent's global state management ability, allowing it to maintain context and eliminate redundant analysis.

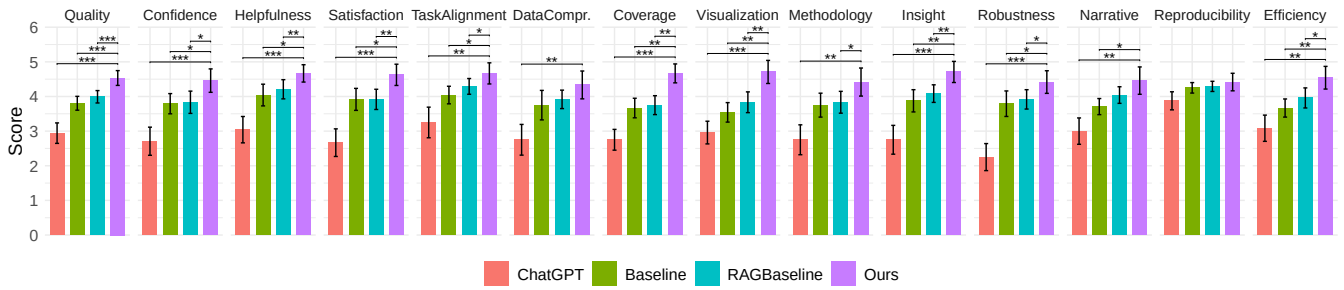

Figure 7: Evaluation results of the four notebooks. Error bars show 95% confidence intervals computed using the Cousineau-Morey method. Asterisks indicate statistical significance ($* \ p < 0.05$, $** \ p < 0.01$, $*** \ p < 0.001$) from pairwise Wilcoxon signed-rank tests with Holm-Bonferroni correction. For clarity, only significant comparisons between *Ours* and other notebooks are shown.

In summary, the quality of *Ours* has been substantially enhanced by the improved RAG-based pipeline, enabling analyses that are more closely aligned with user intent and more insightful in interpretation. The visualizations are more appropriate to the analysis, both visually and methodologically. However, the most frequent negative feedback on *Ours* was that its charts were sometimes unnecessarily complex, which could potentially induce cognitive overload. We will return to this issue in the discussion section. We further analyzed the results within each dataset–task subset using participant-level pairwise score differences (see supplementary materials). Across datasets and tasks, *Ours* consistently achieved higher scores, indicating robust performance.

7 Discussion and Conclusion

Constraints on the Methods in Comparative Evaluation. *ChatGPT* is designed as an interactive interface, but for a fair comparison with *Ours*, we adopted a fixed prompt template to ensure end-to-end generation, which inevitably constrained its capabilities. Therefore, our experimental conclusions only demonstrate that under this constraint, *Ours* performs better. In addition, since no prior work was directly comparable, the design of *RAGBaseline* combined an existing retrieval scheme originally developed for question answering with our *Baseline*. This may introduce potential unfairness, as the retrieval strategy is not fully tailored to downstream EDA tasks.

Infeasibility in Simulating Enterprise Scenarios. Enterprises with data analysis needs represent a potential application scenario for our system. However, the inaccessibility of internal data and artifacts prevented direct evaluation in this scenario. In our experimental design, we attempted to approximate enterprise settings through dataset choices, such as using continuously updated datasets and those with multiple versions. However, differences remain compared with real-world enterprise practice, such as the format of notebooks and the nature of specific tasks. In future work, we aim to collaborate with enterprises to conduct more realistic evaluations in this scenario, where analyses are often deeply coupled with proprietary business logic and contextual knowledge that agentic coding approaches may lack, thereby better highlighting the value of our method.

Possible Influence of Datasets and Tasks. While we examined the advantages of *Ours* within the corresponding subsets, we recognize the possibility that interactions may exist between datasets/tasks and our method. To account for this, we attempted to fit a mixed-effects linear model that included these interaction terms. As noted in Sec. 6.5, the estimates were not sufficiently robust to report, given the limited sample size. Nonetheless, inspection of the point estimates reveals some deviations from zero, suggesting slight performance variations across datasets or task types, but these coefficients are considerably smaller than the main effect of our method.

Scalability and Generalizability of the System. When the number of notebooks is large and their quality varies, we propose prefiltering notebooks based on their quality and relevance to the user intent. Previous works have considered multiple criteria, including format [43, 46], reproducibility, executability [44], and understandability [14]. Our small-scale tests suggest that SOTA LLMs align more closely with human evaluation, and thus we consider using LLMs to assess these criteria, as well as the relevance to the user's intent. For selected notebooks, static code analysis could be used to identify and exclude those with syntax errors or incomplete code. Since our method does not have strict requirements for markdown content and treats it as plain text, low-quality markdown would not affect the system's ability to operate properly. We acknowledge that existing notebooks may not always fully address users' concerns or maintain high quality. However, because our retrieval is based on used-column matching rather than semantic similarity, it avoids introducing large amounts of irrelevant content, even when related material is scarce. Small-scale tests on niche tasks further showed that our method performs at least as well as the *Baseline* and *RAGBaseline*. When no relevant notebooks are available or the dataset is non-tabular format, the baseline generator can still produce reasonable results, though this is not the primary focus of our work.

Possible Over-reliance on Existing Corpora. Feedback from the user study indicated that notebooks produced by *Ours* sometimes included overly complex charts. This was likely because highly upvoted Kaggle notebooks may use elaborate designs to attract attention, although such cases were infrequent. Since some participants appreciated high-information-density visualizations, we plan to adapt our generation strategy in future work based on assessments of visualization complexity and users' visualization preferences.

Execution Efficiency. Compared with *Baseline*, *Ours* requires additional time in the retrieval stage. Since components are independent, we employ a parallel strategy that keeps the processing time for about twenty notebooks (roughly 300 components) within three minutes. The main bottlenecks lie in leveraging the VLM for insight extraction and in code debugging. Given that generating a complete EDA notebook (with twelve sub-tasks) typically takes around ten minutes, the retrieval cost remains within an acceptable range. Given that generating a complete EDA notebook (with twelve sub-tasks) typically takes around ten minutes, and the system is designed as an offline assistant, the retrieval overhead remains acceptable.

Insight Generation. In the Component Enhancement stage (Sec. 4.2.4), we adopted a strategy of first extracting insights from visualizations and then refining them with statistical code, which helps mitigate hallucinations from VLMs. Theoretically, this approach better leverages the strengths of visualizations in revealing patterns; however, our current evaluation is limited to a small-scale manual review and is not yet comprehensive. We acknowledge that this pipeline may introduce a potential confirmation bias, as the statistical verification is performed based on the insights initially proposed by the VLM. In future work, we plan to address this limitation by exploring alternative designs, such as generating and comparing multiple competing hypotheses from the same visualization or explicitly incorporating falsification-oriented statistical tests. We also aim to conduct a more systematic evaluation of how this two-stage reasoning process influences both reliability and interpretability.

ACKNOWLEDGMENTS

This work was supported by the Natural Science Foundation of China (NSFC No. 62472099).

REFERENCES

[1] M. Augustin, Y. Neuhaus, and M. Hein. Dash: Detection and assessment of systematic hallucinations of vlms. *arXiv preprint arXiv:2503.23573*, 2025.

[2] O. Bar El, T. Milo, and A. Somech. Atena: An autonomous system for data exploration based on deep reinforcement learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2873–2876, 2019.

[3] O. Bar El, T. Milo, and A. Somech. Automatically generating data exploration sessions using deep reinforcement learning. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, pp. 1527–1537, 2020.

[4] A. Cardoso, J. Leitão, and C. Teixeira. Using the jupyter notebook as a tool to support the teaching and learning processes in engineering courses. In *The Challenges of the Digital Transformation in Education: Proceedings of the 21st International Conference on Interactive Collaborative Learning (ICL2018)-Volume 2*, pp. 227–236. Springer, 2019.

[5] S. Chattopadhyay, Z. Feng, E. Arteaga, A. Au, G. Ramos, T. Barik, and A. Sarma. Make it make sense! understanding and facilitating sense-making in computational notebooks. *arXiv preprint arXiv:2312.11431*, 2023.

[6] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik. What's wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pp. 1–12, 2020.

[7] J. Chen, J. Wu, J. Guo, V. Mohanty, X. Li, J. P. Ono, W. He, L. Ren, and D. Liu. Interchat: Enhancing generative visual analytics using multimodal interactions. In *Computer Graphics Forum*, p. e70112. Wiley Online Library, 2025.

[8] D. Cousineau et al. Confidence intervals in within-subject designs: A simpler solution to loftus and masson's method. *Tutorials in quantitative methods for psychology*, 1(1):42–45, 2005.

[9] D. Deng, A. Wu, H. Qu, and Y. Wu. Dashbot: Insight-driven dashboard generation based on deep reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):690–700, 2022.

[10] D. Deutch, A. Gilad, T. Milo, and A. Somech. Explained: explanations for eda notebooks. *Proceedings of the VLDB Endowment*, 13(12):2917–2920, 2020.

[11] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 international conference on management of data*, pp. 317–332, 2019.

[12] K. Eckelt, K. Gadhave, A. Lex, and M. Streit. Loops: Leveraging provenance and visualization to support exploratory data analysis in notebooks. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[13] A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

[14] M. M. Ghahfarokhi, A. Asadi, A. Asgari, B. Mohammadi, M. B. Rizi, and A. Heydarnoori. Predicting the understandability of computational notebooks through code metrics analysis. *arXiv preprint arXiv:2406.10989*, 2024.

[15] X. Gu, H. Zhang, and S. Kim. Deep code search. In *Proceedings of the 40th international conference on software engineering*, pp. 933–944, 2018.

[16] G. Harrison, K. Bryson, A. E. B. Bamba, L. Dovichi, A. H. Binion, A. Borem, and B. Ur. Jupyterlab in retrograde: Contextual notifications that highlight fairness and bias issues for data scientists. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–19, 2024.

[17] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2019.

[18] M. Helali, Y. Luo, T. J. Ham, J. Plotts, A. Chaugule, J. Chang, P. Ranganathan, and E. Mansour. Reliable and cost-effective exploratory data analysis via graph-guided rag. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16547–16564, 2025.

[19] J. Huang, D. Guo, C. Wang, J. Gu, S. Lu, J. P. Inala, C. Yan, J. Gao, N. Duan, and M. R. Lyu. Contextualized data-wrangling code generation in computational notebooks. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1282–1294, 2024.

[20] K.-H. Huang, H. P. Chan, Y. R. Fung, H. Qiu, M. Zhou, S. Joty, S.-F. Chang, and H. Ji. From pixels to insights: A survey on automatic chart understanding in the era of large foundation models. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[21] K.-H. Huang, M. Zhou, H. P. Chan, Y. R. Fung, Z. Wang, L. Zhang, S.-F. Chang, and H. Ji. Do lvlms understand charts? analyzing and correcting factual errors in chart captioning. *arXiv preprint arXiv:2312.10160*, 2023.

[22] M. Hutchinson, R. Jianu, A. Slingsby, and P. Madhyastha. Llm-assisted visual analytics: Opportunities and challenges. *arXiv preprint arXiv:2409.02691*, 2024.

[23] M. S. Islam, R. Rahman, A. Masry, M. T. R. Laskar, M. T. Nayeem, and E. Hoque. Are large vision language models up to the challenge of chart comprehension and reasoning? an extensive investigation into the capabilities and limitations of lvlms. *arXiv preprint arXiv:2406.00257*, 2024.

[24] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, et al. Jupyter notebooks–a publishing format for reproducible computational workflows. In *Positioning and power in academic publishing: Players, agents and agendas*, pp. 87–90. IOS press, 2016.

[25] D. J.-L. Lee, D. Tang, K. Agarwal, T. Boonmark, C. Chen, J. Kang, U. Mukhopadhyay, J. Song, M. Yong, M. A. Hearst, et al. Lux: always-on visualization recommendations for exploratory dataframe workflows. *arXiv preprint arXiv:2105.00121*, 2021.

[26] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[27] L. Li and J. Lv. Unlocking insights: Semantic search in jupyter notebooks. *arXiv preprint arXiv:2402.13234*, 2024.

[28] S. Li, Y. Liu, S. Du, W. Zeng, Z. Xu, M. Zhou, Y. He, H. Dong, S. Han, and D. Zhang. Jupiter: Enhancing llm data analysis capabilities via notebook and inference-time value-guided search. *arXiv preprint arXiv:2509.09245*, 2025.

[29] V. R. Li, J. Sun, and M. Wattenberg. Does visualization help ai understand data? *arXiv preprint arXiv:2507.18022*, 2025.

[30] X. Li, Y. Wang, H. Wang, Y. Wang, and J. Zhao. Nbsearch: Semantic search and visual exploration of computational notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2021.

[31] X. Li, Y. Zhang, J. Leung, C. Sun, and J. Zhao. Edassistant: Supporting exploratory data analysis in computational notebooks with in situ code search and recommendation. *ACM Transactions on Interactive Intelligent Systems*, 13(1):1–27, 2023.

[32] J. Lim, J. Ahn, and G. Kim. Chartcap: Mitigating hallucination of dense chart captioning. *arXiv preprint arXiv:2508.03164*, 2025.

[33] P. Ma, R. Ding, S. Wang, S. Han, and D. Zhang. Demonstration of insightpilot: An llm-empowered automated data exploration system. *arXiv preprint arXiv:2304.00477*, 2023.

[34] A. Manatkar, A. Akella, P. Gupta, and K. Narayanam. Quis: Question-guided insights generation for automated exploratory data analysis. *arXiv preprint arXiv:2410.10270*, 2024.

[35] T. Milo and A. Somech. Deep reinforcement-learning framework for exploratory data analysis. In *Proceedings of the first international workshop on exploiting artificial intelligence techniques for data management*, pp. 1–4, 2018.

[36] R. D. Morey et al. Confidence intervals from normalized data: A

correction to cousineau (2005). *Tutorials in quantitative methods for psychology*, 4(2):61–64, 2008.

[37] M. Mostafavi Ghahfarokhi, A. Asgari, M. Abolnejadian, and A. Heydarnoori. Distilkaggle: A distilled dataset of kaggle jupyter notebooks. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pp. 647–651, 2024.

[38] G. J. Myatt and W. P. Johnson. *Making sense of data II: A practical guide to data visualization, advanced data mining methods, and applications*, vol. 2. John Wiley & Sons, 2009.

[39] OpenAI. Data analyst plugin on chatgpt. `https://chat.openai.com`, 2023. Accessed: 2023-10-01.

[40] Y. Ouyang, L. Shen, Y. Wang, and Q. Li. Noteplayer: Engaging computational notebooks for dynamic presentation of analytical processes. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–20, 2024.

[41] J. M. Perkel. Why jupyter is data scientists' computational notebook of choice. *Nature*, 563(7732):145–147, 2018.

[42] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*, pp. 507–517. IEEE, 2019.

[43] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. Understanding and improving the quality and reproducibility of jupyter notebooks. *Empirical Software Engineering*, 26(4):65, 2021.

[44] L. Quaranta. Assessing the quality of computational notebooks for a frictionless transition from exploration to production. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pp. 256–260, 2022.

[45] L. Quaranta, F. Calefato, and F. Lanubile. Kgtorrent: A dataset of python jupyter notebooks from kaggle. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 550–554. IEEE, 2021.

[46] L. Quaranta, F. Calefato, and F. Lanubile. Pynblint: a static analyzer for python jupyter notebooks. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pp. 48–49, 2022.

[47] A. Rule, A. Tabard, and J. D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2018.

[48] A. Sarvghad, M. Tory, and N. Mahyar. Visualizing dimension coverage to support exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(1):21–30, 2016.

[49] Z. Shao, Y. Shan, Y. He, Y. Yao, J. Wang, X. Zhang, Y. Zhang, and S. Chen. Do language model agents align with humans in rating visualizations? an empirical study. *IEEE Computer Graphics and Applications*, 2025.

[50] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM international conference on management of data*, pp. 1509–1524, 2017.

[51] Y. Tian, C. Zhang, X. Wang, S. Pan, W. Cui, H. Zhang, D. Deng, and Y. Wu. Respark: Leveraging previous data reports as references to generate new reports with llms. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–18, 2025.

[52] E. Tufino. Notebooklm: An llm with rag for active learning and collaborative tutoring. *arXiv preprint arXiv:2504.09720*, 2025.

[53] J. W. Tukey et al. *Exploratory data analysis*, vol. 2. Springer, 1977.

[54] A. Y. Wang, D. Wang, J. Drozdal, X. Liu, S. Park, S. Oney, and C. Brooks. What makes a well-documented notebook? a case study of data scientists' documentation practices in kaggle. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2021.

[55] F. Wang, Y. Lin, L. Yang, H. Li, M. Gu, M. Zhu, and H. Qu. Outlinespark: Igniting ai-powered presentation slides creation from computational notebooks through outlines. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–16, 2024.

[56] F. Wang, X. Liu, O. Liu, A. Neshati, T. Ma, M. Zhu, and J. Zhao. Slide4n: Creating presentation slides from computational notebooks with human-ai collaboration. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–18, 2023.

[57] F. Wang, B. Wang, X. Shu, Z. Liu, Z. Shao, C. Liu, and S. Chen. Chartinsighter: An approach for mitigating hallucination in time-series chart summary generation with a benchmark dataset. *arXiv preprint arXiv:2501.09349*, 2025.

[58] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[59] H. W. Wang, L. Birnbaum, and V. Setlur. Jupybara: Operationalizing a design space for actionable data analysis and storytelling with llms. *arXiv preprint arXiv:2501.16661*, 2025.

[60] J. Wang, L. Li, and A. Zeller. Better code, better sharing: on the need of analyzing jupyter notebooks. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering: new ideas and emerging results*, pp. 53–56, 2020.

[61] Y. Wang, Z. Sun, H. Zhang, W. Cui, K. Xu, X. Ma, and D. Zhang. Datashot: Automatic generation of fact sheets from tabular data. *IEEE transactions on visualization and computer graphics*, 26(1):895–905, 2019.

[62] L. Weng, X. Wang, J. Lu, Y. Feng, Y. Liu, H. Feng, D. Huang, and W. Chen. Insightlens: Augmenting llm-powered data analysis with interactive insight management and navigation. *arXiv preprint arXiv:2404.01644*, 2024.

[63] J. Wenskovitch, J. Zhao, S. Carter, M. Cooper, and C. North. Albireo: An interactive tool for visually summarizing computational notebook structure. In *2019 IEEE visualization in data science (VDS)*, pp. 1–10. IEEE, 2019.

[64] K. Wongsuphasawat, Y. Liu, and J. Heer. Goals, process, and challenges of exploratory data analysis: An interview study. *arXiv preprint arXiv:1911.00568*, 2019.

[65] T. Wu, S. Wang, and X. Peng. Autoeda: Iterative data focusing and exploratory analysis based on attribute frequency. In *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 4113–4118. IEEE, 2024.

[66] Y. Wu, L. Yan, L. Shen, Y. Wang, N. Tang, and Y. Luo. Chartinsights: Evaluating multimodal large language models for low-level chart question answering. *arXiv preprint arXiv:2405.07001*, 2024.

[67] L. Xie, C. Zheng, H. Xia, H. Qu, and C. Zhu-Tian. Waitgpt: Monitoring and steering conversational llm agent in data analysis with on-the-fly code visualization. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–14, 2024.

[68] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1539–1554, 2020.

[69] Z. Yang, L. Li, K. Lin, J. Wang, C.-C. Lin, Z. Liu, and L. Wang. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421*, 9(1):1, 2023.

[70] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[71] Y. Zhao, J. Wang, L. Xiang, X. Zhang, Z. Guo, C. Turkay, Y. Zhang, and S. Chen. Lightva: Lightweight visual analytics with llm agent-based task planning and execution. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[72] Y. Zhao, Y. Zhang, Y. Zhang, X. Zhao, J. Wang, Z. Shao, C. Turkay, and S. Chen. Leva: Using large language models to enhance visual analytics. *IEEE transactions on visualization and computer graphics*, 2024.

[73] J.-P. Zhu, B. Niu, P. Cai, Z. Ni, J. Wan, K. Xu, J. Huang, S. Ma, B. Wang, X. Zhou, et al. Towards automated cross-domain exploratory data analysis through large language models. *arXiv preprint arXiv:2412.07214*, 2024.