# Reliable and Cost-Effective Exploratory Data Analysis via Graph-Guided RAG

**Mossad Helali[1]*, Yutai Luo[2], Tae Jun Ham[2], Jim Plotts[2], Ashwin Chaugule[2], Jichuan Chang[2], Parthasarathy Ranganathan[2], Essam Mansour[1],**

[1]Concordia University, Canada, [2]Google, USA,

**Correspondence:** mossad.helali@mail.concordia.ca, yutailuo@google.com

## Abstract

Automating Exploratory Data Analysis (EDA) is critical for accelerating the workflow of data scientists. While Large Language Models (LLMs) offer a promising solution, current LLM-only approaches often exhibit limited accuracy and code reliability on less-studied or private datasets. Moreover, their effectiveness significantly diminishes with open-source LLMs compared to proprietary ones, limiting their usability in enterprises that prefer local models for privacy and cost. To address these limitations, we introduce RAGvis: a novel two-stage graph-guided Retrieval-Augmented Generation (RAG) framework. RAGvis first builds a base knowledge graph (KG) of EDA notebooks and enriches it with structured EDA operation semantics. These semantics are extracted by an LLM guided by our empirically-developed EDA operations taxonomy. Second, in the online generation stage for new datasets, RAGvis retrieves relevant operations from the KG, aligns them to the dataset's structure, refines them with LLM reasoning, and then employs a self-correcting agent to generate executable Python code. Experiments on two benchmarks demonstrate that RAGvis significantly improves code executability (pass rate), semantic accuracy, and visual quality in generated operations. This enhanced performance is achieved with substantially lower token usage compared to LLM-only baselines. Notably, our approach enables smaller, open-source LLMs to match the performance of proprietary models, presenting a reliable and cost-effective pathway for automated EDA code generation.

## 1 Introduction

Exploratory Data Analysis (EDA) is a foundational data science step for understanding datasets and uncovering patterns. Its significance is underscored by surveys indicating EDA constitutes a primary responsibility for 57% of data scientists (Kaggle, 2022) and consumes up to 40% of their time (Anaconda, 2022). Consequently, automating EDA code generation is critical to accelerating data science workflows. However, this automation presents challenges distinct from general coding tasks. It requires not only reliably generating code for specific operations but, more critically, involves autonomously identifying relevant data columns to analyze and selecting appropriate EDA operations (e.g., chart types) for each column directly from a raw dataset—navigating a vast combinatorial search space.

Current approaches for automating EDA primarily leverage Large Language Models (LLMs) (Dibia, 2023; Ma et al., 2023). While LLMs have shown promising results for general code generation tasks (Jiang et al., 2024; Zan et al., 2023; Hou et al., 2024), their direct application to EDA automation reveals specific limitations. For instance, LLMs often perform well with well-studied public datasets, likely reflecting patterns seen in their vast training corpora. However, their accuracy (e.g., in selecting relevant EDA operations) and reliability (e.g., in generating consistently executable code) have been shown to significantly diminish when applied to private or enterprise datasets, which may feature unique structures or domain-specific conventions (Kayali et al., 2025). Furthermore, the efficacy of these LLM-only methods is primarily tied to the use of large-scale, proprietary models such as Google Gemini or OpenAI GPT (Gemini-Team et al., 2024; OpenAI et al., 2024). However, such performance may not readily translate to smaller, open-source LLMs, which are often preferred in enterprise settings for privacy, control over local deployment, and cost. Therefore, the usability and scalability of LLM-only methods is restricted in many practical scenarios.

To address these challenges, we introduce RAGvis, a novel Retrieval-Augmented Generation (RAG) framework designed for generating executable EDA code. RAGvis' retrieval is based on a knowledge graph linking EDA operations code to their semantics and the utilized data columns.

*Work done as a student researcher at Google.



*Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16548–16565
November 4-9, 2025 

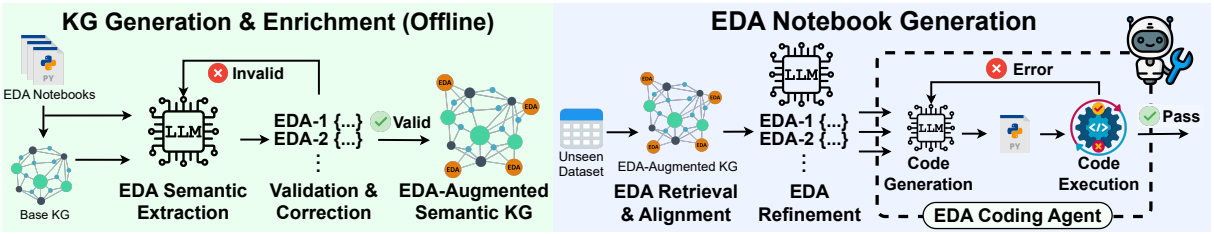


Figure 1: Overview of RAGvis' KG generation & enrichment (Sec. 2) and EDA notebook generation (Sec. 3).

Our approach operates in two stages as shown in Figure 1.

The first is offline **Knowledge Graph (KG) Generation and Semantic Enrichment**. Initially, a large corpus of Kaggle EDA notebooks is processed to generate a base KG capturing statement-level notebook information (e.g., code dependencies, library usage) and associated dataset column embeddings. This base KG, however, lacks explicit EDA operations semantics (e.g., the specific chart type or columns an operation analyzes), a critical input for effective retrieval. To systematically define which semantics to collect, given the wide diversity of EDA operations, we first develop our structured EDA taxonomy (Figure 2). The core of the enrichment stage is then our novel KG-guided EDA semantic extraction: for each EDA operation in the notebooks, we provide an LLM with the operation's code script contextualized by relevant parent statements derived from the base KG's data flow dependencies. The LLM is then tasked to predict the operation's semantic attributes according to our taxonomy. After validation and correction, these extracted EDA semantics are augmented to the base KG.

In the subsequent online **EDA notebook generation** stage, RAGvis takes an unseen dataset as input. First, it retrieves relevant EDA operations from the augmented KG using column embeddings, and aligns their column specifications to the unseen dataset. Next, an LLM refines this set of candidate operations based on the unseen dataset's characteristics. Finally, an EDA coding agent takes these refined operation specifications to generate and iteratively self-correct Python code through execution feedback, and assembles the verified snippets into a notebook. By leveraging the enriched KG to guide the selection of EDA operations tailored to the input dataset, and using an agent for robust code generation, RAGvis overcomes key limitations of relying solely on LLMs.

To validate our approach, we conducted extensive experiments on two benchmarks, comparing RAGvis to a state-of-the-art LLM-based baseline (Dibia, 2023). RAGvis demonstrates significant improvement on EDA retrieval accuracy (up to 0.35 higher recall), code reliability (near 100% pass rate), visual chart quality (LLM-as-a-judge score), and cost (64% less LLM tokens consumed). Moreover, RAGvis enables small, open-source models such as Gemma 3 to achieve performance comparable to proprietary models such as GPT 4o-mini.

This paper makes the following contributions:

- We introduce RAGvis[1], a novel RAG framework for EDA code generation, featuring: a) an empirically-driven universal taxonomy of EDA operations that categorizes visualizations by their semantic intent, b) a KG-guided semantic extraction method to guide an LLM in building an EDA-aware KG from notebooks, c) an online process that integrates EDA-specific retrieval, alignment, and refinement to tailor EDA operations for unseen datasets, and d) a self-correcting EDA coding agent that iteratively generates and validates Python code to ensure reliability.
- Through comprehensive experiments on two benchmarks, we demonstrate that RAGvis significantly outperforms a state-of-the-art LLM-only baseline, achieving substantial improvements in semantic accuracy, code reliability, visual chart quality, and LLM token costs.
- We show that RAGvis empowers smaller, open-source models to achieve comparable performance to that of large, proprietary models, making automated EDA more feasible in low-resource and privacy-sensitive scenarios.

## 2 Knowledge Graph Generation and Semantic Enrichment

The retrieval in RAGvis relies on a Knowledge Graph (KG) that links EDA operations to their implementing code and the specific data columns they

[1]Code and datasets available at: https://github.com/google/ragvis

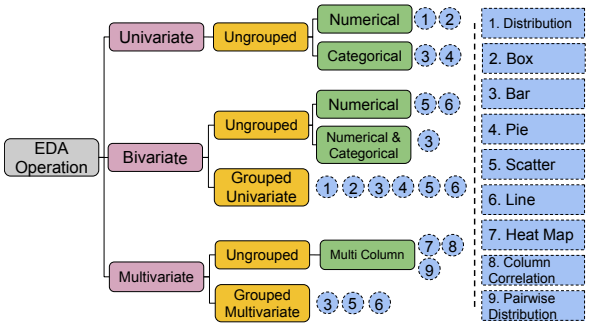


Figure 2: Our taxonomy of EDA operations based on a survey of EDA notebooks on the Kaggle platform.

analyze. As such an EDA-specific KG is not available, we first utilize a base KG built from a large corpus of EDA notebooks and datasets. The base KG captures essential statement-level information about the code and its use of dataset columns but lacks EDA information. Therefore, we systematically augment it with EDA-specific semantics, yielding an EDA-aware KG crucial for retrieval.

### 2.1 Base Knowledge Graph Generation

To generate the base KG, we first collected a corpus comprising approximately 1600 EDA notebooks associated with 911 distinct datasets sourced from the Kaggle platform. To ensure quality, these notebooks were selected based on their high vote counts by the Kaggle community (meta-kaggle, 2022; Plotts and Risdal, 2023).

To process this corpus into a structured KG, we built upon recent advances in building KGs from code repositories (Venkataramanan et al., 2025) and generating embeddings for tabular data (Jiang et al., 2025). We employed KGLiDS (Helali et al., 2024), a platform designed for generating KGs from data science pipelines and their associated datasets, integrating both statement-level code information and column embeddings. Nodes in the KG include code statements and dataset columns, while edges denote code dependencies and data reads by statements. For instance, considering the example in Figure 3, the base KG represents all code statements (lines 1-25) as code statement nodes. These are connected by code flow edges that capture both logical sequence (e.g., line 2 follows line 1) and data dependencies (e.g., the `df` variable from line 4 is used in line 24). Columns from the dataset, like `"Openings"` and `"Company"`, are also represented as column nodes. These are linked to code statement nodes by column read edges whenever a statement uses them (e.g., line 2 is connected to the `"Openings"` and `"Company"` nodes). This base KG, however, does not identify EDA operations in a notebook and lacks explicit EDA-specific semantics required for our task such as the type of an EDA operation and its intent. Figure 4 shows the base KG corresponding to the example in Figure 3 before and after enrichment.

The KGLiDS platform also analyzes datasets at the column level, generating embeddings based on the column name, raw value, and a concatenation of both. We utilize these embeddings for accurate EDA retrieval (Section 3.1).

### 2.2 Taxonomy of EDA Operations

To systematically enrich the KG with EDA-specific semantics, we first need a structured way to define and categorize the wide variety of EDA operations found in practice. Simply instructing an LLM to "extract EDA semantics" without a clear definition would lead to inconsistent and incomplete information. Therefore, we developed a comprehensive taxonomy of EDA operations derived from an extensive survey of notebooks on the Kaggle platform. While empirically derived, this taxonomy is designed to be broadly generalizable beyond Kaggle; its core structure (Figure 2) is based on fundamental analytical concepts rather than specific implementation details (such as plotting libraries or cosmetic variations).

Our taxonomy is hierarchical, first categorizing operations by the number of primary columns involved: Univariate (one column), Bivariate (two columns), and Multivariate (three or more columns). Further distinctions consider grouping and analysis types (observed frequencies: Univariate 36.3%, Bivariate 44.9%, Multivariate 18.8%).

**Univariate** operations, in our taxonomy, are Ungrouped. They typically involve Distribution plots (1) or Box plots (2) for Numerical columns, and Bar charts (3) or Pie charts (4) for Categorical columns. **Bivariate** operations relate two variables. Ungrouped forms include Scatter plots (5) or Line charts (6) for two numerical columns, or Bar charts (3) for comparing a numerical column against a categorical one (e.g., 'Total Population' per 'City'). A key subtype is **Grouped Univariate** analysis. This involves performing a univariate analysis (using charts 1-6) on a value column for each segment defined by a *grouping column* (typically categorical). An example is grouping 'Employee' data by 'Department' and showing the average 'Income'. Notably, 23.6% of analyzed operations involved a grouping column, representing a common pattern.

```
1: df = pd.read_csv("DataAnalyst.csv")
2: df = df.rename(columns={"Openings":"Jobs",
3:                         "Company":"Employer"})
4: df["Jobs"] = df["Jobs"].fillna("missing")
5: feature_selector = SelectKBest(k=20)
                    ...
24: data = df.groupby("Employer")["Jobs"].sum()
25: chart = sns.barplot(data,x="Employer",y="Jobs")
```

```
{"chart_type": "Bar", "analysis_type": "Bivariate",
 "chart_columns": ["Openings"],
 "grouping_column": "Company"}
```

Figure 3: Top: EDA script, where a dataset is loaded, preprocessed (Lines 2-5), and used to visualize a Bar Chart showing the number of jobs per employer (Lines 24-25). Bottom: corresponding EDA semantics extracted using the taxonomy in Figure 2.

**Multivariate** operations analyze three or more columns. Ungrouped examples include Heatmaps (7), Column Correlation (8) computations, and Pairwise Distributions (9). **Grouped Multivariate** operations use one column for grouping (e.g., for color or faceting) while visualizing relationships between two or more other columns (e.g., using faceted/colored Scatter (5) or Line (6) charts, or potentially grouped Bar charts (3)). This might explore how the relationships vary across data segments (e.g., average 'Income' vs. 'Happiness Index', colored by 'State').

It is important to note that functionally similar charts are grouped into categories; for instance, the "Box" category includes Box and Violin plots, while "Distribution" contains Histograms and KDE plots. Each level in the taxonomy corresponds to a semantic attribute of an EDA operation. Figure 3 shows a sample EDA operation and the corresponding EDA semantics per this taxonomy.

### 2.3 KG-Guided EDA Semantic Extraction

The base KG, while capturing statement-level details, initially lacks explicit EDA operation semantics, a critical input for EDA retrieval and code generation. We enrich the KG by extracting the EDA-specific semantics from collected Kaggle notebooks. This extraction occurs per EDA cell using an LLM with few-shot prompting, guided by our EDA taxonomy. The LLM prompt includes: 1) the target EDA code snippet (contextualized as detailed below); 2) a dataset sample; 3) expected semantic fields based on our taxonomy (as shown in Figure 3); and 4) few-shot (code snippet, dataset sample, EDA semantics) examples. The LLM then extracts chart columns, grouping column, chart type, and analysis type.

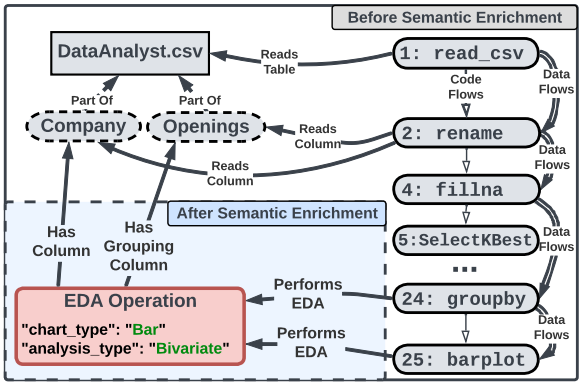


Figure 4: The base KG (grey) and enriched KG (blue) corresponding to the script in Figure 3.

A key challenge is providing sufficient yet concise source code context for the LLM, as EDA notebooks often preprocess data (e.g., renaming `"Openings"` to `"Jobs"` in Figure 3, lines 1-5) before visualization (lines 24-25). Providing only the EDA cell (lines 24-25) to the LLM can lead to incorrect column identification (e.g., user-defined names like `"Jobs"` instead of the correct original name, `"Openings"`). In contrast, providing the full script as context is often too costly or might exceed the context size of some LLMs. Our solution is to provide the LLM with *relevant context* by querying the KG for data flow parent statements of the EDA cell. The data flow parents include all statements that define or modify data variables used within the EDA cell (e.g., lines 1-4 are parents for lines 24-25 in Figure 3). We prepend the data flow parents in order to the EDA cell code within the LLM prompt. This KG-assisted contextualization (summarized in Algorithm 2 in Appendix) ensures accurate semantic extraction with minimal added token consumption. Figure 3 (bottom) shows the extracted EDA semantics for the sample operation.

The format of the LLM-extracted semantics is validated by enforcing a JSON schema, while the semantic correctness is validated via heuristic rules (e.g. checking that column names exist in the original dataset); if an operation fails validation, the LLM is asked for regeneration, and operations remaining incorrect after set retries are discarded.

### 2.4 Knowledge Graph Augmentation

To transform the base KG into an EDA-aware resource, we augment it with the extracted semantics. For each successfully analyzed EDA operation, we create a new EDA operation node and store its semantic attributes (e.g., `chart_type`, `analysis_type`) as node properties, as shown at

the bottom of Figure 3. This new node is then linked via edges to: 1) the code statement nodes that perform the analysis (lines 24-25), and 2) the nodes of the original dataset columns it analyzes (`"Openings"` and `"Company"`). This explicit linking between an EDA operation and its columns is vital, as it directly facilitates the targeted retrieval of EDA operations for a given column in the subsequent notebook generation phase.

## 3 EDA Notebook Generation

The objective of this stage is to generate a comprehensive EDA notebook given an unseen dataset.

### 3.1 Embedding-based EDA Retrieval

Our retrieval approach operates on the premise that similar datasets are often effectively analyzed using similar EDA operations. Retrieval in RAGvis (Algorithm 1, Lines 2-6) utilizes the EDA-augmented KG to fetch relevant EDA operations for an unseen dataset $D_u$. It does so by querying the KG for operations that were applied to columns in *seen* datasets similar to those in $D_u$. This utilizes the explicit links between EDA operations and data columns added in the KG enrichment.

However, measuring column similarity is challenging. Using column names alone is often insufficient as columns with identical names can have vastly different data types or value distributions. For example, a `"rating"` column could contain strings like "Good", integers on a 1-100 scale, or floats from 1-5. To address this, RAGvis utilizes column embeddings for similar column matching. These embeddings (generated in the base KG) are concatenations of column name embeddings and column raw value embeddings, resulting in accurate retrieval of EDA operations.

The retrieval process operates at the column granularity. For each column $C_u \in D_u$, we query the KG column embeddings using cosine similarity. The query targets seen columns that share $C_u$'s data type (e.g. our EDA-augmented KG recognizes 7 types: integers, floats, booleans, dates, natural language text, named entities, and generic strings). The set of top $N$ most similar seen columns, $\mathcal{C}_{top-N}$, are retrieved, where $N$ is a configurable parameter influencing recall. Subsequently, for each retrieved seen column $C_s \in \mathcal{C}_{top-N}$, RAGvis queries the KG to fetch its associated EDA operations. Crucially, these operations are presented in the context of the *seen* datasets from which they were retrieved (i.e., they reference columns in the original seen dataset). To render these operations directly executable on the unseen dataset $D_u$, a subsequent EDA Alignment step is necessary.

### 3.2 EDA Alignment

The goal of the alignment step is to find a suitable mapping for a retrieved EDA operation (applied on a seen dataset) to the unseen dataset $D_u$. EDA alignment in RAGvis (Lines 7-17 in Algorithm 1) uses a two-factor criterion: column data type compatibility and embedding similarity. For each seen column $C_s$ used in an operation $O$, the system searches within the unseen dataset $D_u$ for a potential target column $C_{align}$ that has a matching data type with $C_s$, highest embedding similarity, and was not previously selected as an alignment target for the same operation (to ensure $1:1$ mapping to $D_u$). Alignment for univariate operations applied to $C_s$ is straightforward, mapping $C_s$ directly to $C_u$. For bivariate and multivariate operations involving other columns than $C_s$, however, **all** participating columns from the seen dataset must also be aligned to suitable columns in the unseen dataset $D_u$ using the same process. If, for a given operation $O$, a suitable alignment cannot be found for all seen columns to which $O$ was historically applied, then $O$ is deemed unalignable in the context of $D_u$ and is discarded from the candidate set.

After this alignment process, all remaining EDA operations are now expressed in terms of columns within $D_u$ and are considered directly applicable. This procedure, applied across all retrieved $C_{top-N}$ columns for each $C_u$, may result in duplicate EDA operations being suggested. RAGvis regards the frequency of each unique operation as an estimation of the system confidence in recommending said operation. The frequency is utilized in the subsequent stage to inform the LLM about the relative importance of each EDA operation.

### 3.3 EDA Refinement

The EDA Refinement step leverages LLM reasoning to curate prior aligned operations. Its input includes aligned EDA operations, sorted by increasing frequency count (the LLM is explicitly informed of this order), alongside key semantic details of the unseen dataset (column names, data types, and basic statistics). Using a one-shot prompt, the LLM is instructed to select $k$ most suitable EDA operations, considering both the provided candidates and the dataset semantics, with

**Algorithm 1:** EDA Retrieval and Alignment

**Input:** Unseen Dataset: $D_u$, EDA-Augmented KG: $\mathcal{G}$
No. similar cols to retrieve: $N$

$EDA_{Aligned} = \{\ \}$
**for** *unseen column* $C_u \in D_u$*:*
  $\mathcal{C}_{top-N} \leftarrow$ {SELECT C FROM $\mathcal{G}$ WHERE $C$.type=$C_u$.type
    ORDER BY $\mathbf{Emb}_{sim}(C, C_u)$ LIMIT $N$}
  **for** *similar seen column* $C_s \in \mathcal{C}_{top-N}$*:*
    $\mathcal{O}_{C_s} \leftarrow$ {SELECT EDA FROM $\mathcal{G}$ WHERE EDA $\leftrightarrow C_s$ }
    **for** *EDA operation* $O_{\{C_{s...}\}} \in \mathcal{O}_{C_s}$*:*
      $aligned\_D_u\_cols \leftarrow \{\ \}$
      **for** *seen column* $C_{seen}$ *used in* $O_{\{C_{s...}\}}$
        $C_{align} \leftarrow$ {SELECT $C$ FROM $D_u$
          WHERE $C$.type = $C_{seen}$.type
          AND $C$ NOT IN $aligned\_D_u\_cols$
          ORDER BY $\mathbf{Emb}_{sim}(C, C_{seen})$ LIMIT 1}
        $aligned\_D_u\_cols.add(C_{align})$
      **if** *all columns in* $O_{\{C_{s...}\}}$ *are aligned:*
        $EDA_{Aligned}.add(O(aligned\_D_u\_cols))$
**return** Counter($EDA_{Aligned}$)

the possibility of suggesting novel operations not present in the input list. To ensure reliable output structure, we utilize the LLM's constrained generation (e.g., JSON mode), using our EDA taxonomy as a schema. This yields $k$ refined EDA operations in JSON format, aligned to $D_u$ and ready for code generation; $k$ is user-defined. This refinement step is optional, particularly for smaller models with limited reasoning ability. If disabled, the top k operations ranked by frequency from the alignment step are passed directly to the EDA coding agent.

### 3.4 EDA Coding Agent

The EDA Coding Agent generates verified Python snippets for the $k$ refined EDA operations applicable to the unseen dataset $D_u$. For each input operation (provided as JSON specification) paired with $D_u$'s summary, an LLM generates a corresponding Python code snippet designed to produce exactly one data visualization (a single chart). The agent then attempts to execute this generated snippet in a local environment containing $D_u$ to verify it runs without compile-time or runtime errors. If execution fails, the agent captures the resulting error message and feeds it back into the original code generation session, prompting the LLM to regenerate the code specifically addressing the reported error. This initiates an automated, iterative debugging loop of generation, execution, and correction which continues until the code runs successfully or a predefined maximum number of retries is reached. Any operation whose code cannot be fixed within the retry limit is ultimately excluded. Therefore, the output of this stage is a set of up to $k$ Python code snippets, each successfully executed and verified error-free on $D_u$, corresponding to one of the refined EDA operations. These snippets can then be assembled into a single executable file (e.g. a Jupyter Notebook) using an LLM prompt to deduplicate redundant setup code.

## 4 Experimental Evaluation

We conduct a comprehensive evaluation between RAGvis and two baseline systems on two independent data visualization benchmarks.

### 4.1 Benchmarks

**KaggleVisBench**[2]: We collected a high-quality benchmark from 132 EDA notebooks associated with 50 Kaggle datasets. The notebooks are the most voted among Kaggle users for each dataset, with at least 20 votes. We extracted the set of EDA operations per dataset using the EDA semantic extraction described in Section 2.3. To ensure a fair evaluation and prevent information leakage, we verified that none of the 50 datasets in this benchmark were present in the corpus of notebooks used to build our knowledge graph. The benchmark features an average of 20 EDA operations per dataset, totalling 999 operations. Samples in this benchmark consist of pairs of a dataset and the list of corresponding EDA operations, where each EDA operation has the following semantic attributes: `chart_type`, `chart_columns`, and `grouping_column` .

**VisEval**: To assess our system's ability to generalize to EDA patterns beyond the Kaggle ecosystem, we also evaluate on VisEval (Chen et al., 2025); This is a natural language to visualization (NL2VIS) benchmark consisting of high-quality datasets manually annotated by human experts, providing an out-of-domain test for our approach. Original samples in VisEval consist of a visualization query for a dataset and the corresponding chart metadata, including chart type (Bar, Pie, etc.), chart columns, and query difficulty. We grouped the samples by datasets and considered the set of charts for each dataset as ground truth. In addition, we filter out the 462 multi-table operations as they are out of the scope of this study. This results in an average of 2.9 EDA operations per dataset, totalling 688 operations for 239 datasets.

[2]The benchmark is available for the research community at: https://github.com/google/ragvis

### 4.2 Baselines

**LIDA**: We consider LIDA (Dibia, 2023) as our primary baseline. LIDA is a multi-step prompt-based automated data visualization system. LIDA takes a raw dataset as input and i) summarizes the dataset information, ii) explores $k$ potential EDA operations specifications, iii) generates the corresponding Python code, and iv) executes each visualization code to generate $k$ visualizations. Here, $k$ is a parameter to control the number of charts. Steps i through iii are carried out using LLM prompting.

**Data2Vis**: To contextualize the performance of modern LLM-based approaches, we evaluate against Data2Vis (Maddigan and Susnjak, 2023), a traditional AutoVis system. Data2Vis utilizes a BiLSTM to generate visualizations in the Vega-Lite declarative specification language (Satyanarayan et al., 2017). A full comparison is not feasible, as it does not generate source code. To enable a partial comparison, we mapped its outputs to our EDA taxonomy to compute accuracy using Recall@k.

### 4.3 Evaluation Metrics

**Recall@k**: There is no standard metric to measure the "goodness" of a generated chart as data visualization is inherently subjective. In this work, we developed the EDA taxonomy described in Section 2.2 and utilize it to measure the semantic accuracy of the generated charts in an automated and reproducible manner. Recall is measured by matching the semantic attributes in our taxonomy shown in Figure 3. A true positive for a particular dataset is a predicted EDA operation that exists in the set of operations in the ground truth. We use exact matching for all attributes of the taxonomy; if, for instance, a correct chart type (e.g. Bar) is predicted but applied on or grouped by incorrect set of columns, the prediction is regarded as a false positive. Specifically, we calculate Recall@k, where $k$ is the desired number of EDA operations. We selected practical values of $k \in [5, 30]$, resembling real-world notebooks of varying lenghts. Recall@k is then averaged for all datasets in a benchmark.

**Pass Rate@k**: Similar to previous works (Chen et al., 2025; Dibia, 2023), we measure source code reliability using pass rate, which is the percentage of generated EDA operations with source code executing without compile-time or run-time errors. We calculate Pass Rate@k at varying $k$ values and average it for all datasets in a benchmark.

**VLM Score**: To assess the visual quality of the generated charts, we use a Vision Language Model (VLM) as a judge, following the same procedure reported by the authors of VisEval (Chen et al., 2025). This automated metric, was shown to highly correlate with human expert ratings, providing a practical alternative to a user study. The score consists of two components: **Readability** evaluates the aesthetic and functional quality of a chart, including its layout, scales, ticks, and the clarity of titles and labels. The **Quality Score** then adjusts the readability score by penalizing any charts that are invalid or contain errors. We used Gemini 2.5 Pro as the judge given its advanced reasoning capabilities. The VLM judge prompt is shown in A.5.

**LLM Tokens**: We calculate the number of LLM tokens used in the generation stage of both systems.

### 4.4 EDA Retrieval Accuracy

Table 1 (left) shows the average Recall@k of RAGvis compared to LIDA on both benchmarks. RAGvis significantly outperforms LIDA for all values of $k$ and LLMs with a difference in Recall@30 reaching up to 0.33 on VisEval and 0.29 on KaggleVisBench. This demonstrates the effectiveness of our EDA retrieval and refinement compared to LLM prompting used in LIDA. By grounding the EDA selection process in the knowledge graph, RAGvis effectively navigates the large search space of the auto EDA problem. Prompt-based techniques like LIDA solely depend on LLM reasoning for EDA selection, which is not as effective for this task even with thinking models such as Gemini 2.5 Pro.

Furthermore, our comparison against Data2Vis highlights the significant performance gap between prior methods and LLM-based approaches for this task. As shown in Table 1, RAGvis substantially outperforms Data2Vis on both benchmarks.

### 4.5 EDA Code Reliability

Table 1 (right) shows the pass rates of RAGvis and LIDA on both benchmarks for different values of $k$ and LLMs. RAGvis maintains an impressive $\sim$100% pass rate across all values of $k$ with Gemini 2.5 Pro and GPT 4o-mini. This illustrates the effectiveness of our EDA coding agent, which executes the generated source code and performs self correction if needed. While LIDA requires strong coding models such as Gemini 2.5 Pro to achieve acceptable pass rate of 95%+, RAGvis achieves competitive pass rates even with Gemma 3 12b.

Table 1: A comparison between the average Recall@k (R@k) and average Pass Rate@k (PR@k) for all systems on two benchmarks with different open source and commercial large language models.

| Method | VisEval | | | KaggleVisBench | | | VisEval | | | KaggleVisBench | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R@5 | R@15 | R@30 | R@5 | R@15 | R@30 | PR@5 | PR@15 | PR@30 | PR@5 | PR@15 | PR@30 |
| **Gemini 2.5 Pro (25/03)** | | | | | | | | | | | | |
| **RAGvis** | **0.5** | **0.67** | **0.77** | **0.12** | **0.29** | **0.43** | **99.9%** | **100%** | **100%** | **99.2%** | **99.9%** | **99.5%** |
| **LIDA** | 0.15 | 0.36 | 0.49 | 0.01 | 0.06 | 0.14 | 95.6% | 90.3% | 96.3% | 89.2% | 92.4% | 93.5% |
| **GPT-4o mini** | | | | | | | | | | | | |
| **RAGvis** | **0.33** | **0.59** | **0.68** | **0.07** | **0.18** | **0.3** | **99.6%** | **100%** | **99.8%** | **99.8%** | **99.2%** | **99.1%** |
| **LIDA** | 0.24 | 0.40 | 0.52 | 0.06 | 0.12 | 0.18 | 93.2% | 88.4% | 89.3% | 84.4% | 81.6% | 72.3% |
| **Gemma 3 12b** | | | | | | | | | | | | |
| **RAGvis** | **0.34** | **0.63** | **0.7** | **0.07** | **0.18** | **0.22** | **98.5%** | **97.7%** | **95.0%** | **96.8%** | **96.3%** | **95.7%** |
| **LIDA** | 0.14 | 0.28 | 0.43 | 0.02 | 0.09 | 0.13 | 88.9% | 88.7% | 91.1% | 78.4% | 82.9% | 66.5% |
| **Gemma 3 4b** | | | | | | | | | | | | |
| **RAGvis** | **0.22** | **0.46** | **0.61** | **0.05** | **0.10** | **0.15** | **98.3%** | **98.6%** | **98.2%** | **92.0%** | **89.2%** | **87.3%** |
| **LIDA** | 0.03 | 0.15 | 0.28 | 0.01 | 0.06 | 0.11 | 86.6% | 93.3% | 90.3% | 79.2% | 77.9% | 68.3% |
| **Data2Vis** | 0.02 | 0.05 | 0.12 | 0.01 | 0.02 | 0.04 | - | - | - | - | - | - |

Table 2: Readability and Quality scores of RAGvis and LIDA with $k = 15$ using a VLM as a judge on VisEval.

| Model | Method | Readability | Quality |
|---|---|---|---|
| **GPT 4o-mini** | **RAGvis** | **2.93** | **2.93** |
| | **LIDA** | 2.55 | 2.34 |
| **Gemma 3 12b** | **RAGvis** | **2.72** | **2.72** |
| | **LIDA** | 2.42 | 2.22 |

### 4.6 EDA Visual Quality

To evaluate the visual quality of the generated charts, we conducted an analysis using the VLM Score metric on the VisEval benchmark with $k = 15$. We ran both RAGvis and LIDA using GPT-4o-mini and Gemma 3 12b, with Gemini 2.5 Pro as the VLM judge. The results, summarized in Table 2, show that RAGvis consistently achieves higher Readability and Quality scores. This is because RAGvis selects more accurate chart types and columns, resulting in more readable and aesthetically pleasing charts. A paired t-test confirms the results are statistically significant (with p < 5e-10 for all settings).

### 4.7 Open vs. Commercial LLMs

A primary goal of our work is to enable effective and reliable EDA code generation with small, open-source models. While RAGvis is able to utilize the advanced reasoning of Gemini 2.5 Pro to obtain the best recalls and pass rates as shown in Table 1, it achieves comparable performance with Gemma 3 12b to GPT 4o-mini and a competitive performance with the small Gemma 3 4b. LIDA, on the other hand, is entirely dependent for its performance on the LLM reasoning capabilities of proprietary models, showing a significant drop in recalls and pass rates with the Gemma models. Similarly, for visual quality shown in Table 2, RAGvis produces higher-quality visualizations with Gemma 3 12b than LIDA with GPT 4o-mini. This is because RAGvis decouples the task of EDA selection from code generation; by grounding the former in the KG, it transforms the latter into a well-defined task where even smaller models can excel.

### 4.8 LLM Token Usage

We compared the number of LLM tokens utilized by RAGvis and LIDA on VisEval. Table 3 shows the input, output, and thought tokens used in evaluation. RAGvis significantly reduces token consumption, using 56%-64% less input tokens and 59%-68% less output tokens. This efficiency stems largely from RAGvis's RAG architecture offloading data summarization and EDA retrieval tasks to non-LLM components. Moreover, RAGvis avoids the high token overhead seen in LIDA, which often relies on long, detailed system prompts (increasing input tokens) and requires the generation of verbose rationale with each EDA specification (increasing output tokens). Our approach achieves better results without these token-intensive steps.

Table 3: A comparison with LIDA in terms of number of tokens incurred to evaluate both systems on VisEval.

| Model | Gemini 2.5 Pro | | GPT 4o-mini | |
|---|---|---|---|---|
| **Method** | **RAGvis** | **LIDA** | **RAGvis** | **LIDA** |
| **Input Tokens** | **14.46 M** | 33.13 M | **10.43 M** | 30.19 M |
| **Output Tokens** | **7.38 M** | 18.08 M | **2.36 M** | 7.53 M |
| **Thought Tokens** | 30.97 M | **24.3 M** | - | - |

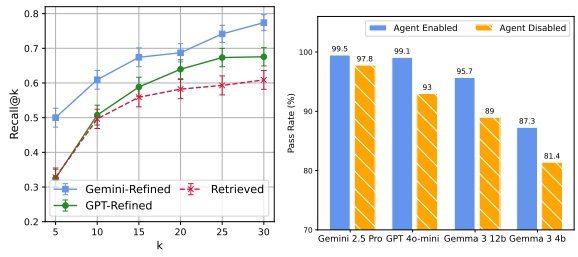


Figure 5: Ablations of the impact of LLM refinement on Recall@k (left) and EDA coding agent on PassRate.

### 4.9 Ablation Studies

We analyze the impact of key design choices. Additional ablations are included in the Appendix.

#### 4.9.1 Impact of LLM Refinement

We first quantify the LLM refinement impact on RAGvis's recall. Figure 5 (left) shows three settings: full RAGvis pipeline with refinement of Gemini 2.5 Pro (Gemini-Refined), GPT 4o-mini (GPT-Refined), and the retrieved and aligned EDA operations without applying any LLM refinement (Retrieved). Crucially, the figure reveals the competitive recall achieved with retrieval only, validating the effectiveness of our retrieval and alignment processes. Further, RAGvis takes advantage of advanced reasoning models, showing higher recalls with Gemini 2.5 Pro compared to GPT 4o-mini.

#### 4.9.2 Impact of EDA Coding Agent

To assess the contribution of the EDA coding agent, we conducted an ablation study where this component was replaced with a single LLM prompt without any execution feedback. The results, depicted in Figure 5 (right), demonstrate the agent's significant role in enhancing the pass rate of generated code. Notably, the EDA coding agent provides a considerable boost in pass rates for open-sourced models reaching approximately 6%. While strong coding models such as Gemini 2.5 Pro also benefit from the agent, the magnitude of this improvement is comparatively smaller (1.7%) than that observed for other models.

## 5 Related Work

**Automated Data Visualization (AutoVis)** focuses on automatically generating appropriate data visualizations (charts) directly from raw datasets. Traditional AutoVis systems use techniques like reinforcement learning (Bar El et al., 2020), sequence-to-sequence models (Dibia and Demiralp, 2019), or heuristic-based recommendations (Wongsuphasawat et al., 2017; Ma et al., 2021), but often rely on brittle heuristics, support a limited set of EDA operations, and produce declarative specifications (e.g., Vega-Lite (Satyanarayan et al., 2017)) instead of executable code. More recent LLM-only methods like LIDA (Dibia, 2023) generate code directly but lack explainability and suffer from hallucinations. RAGvis employs a hybrid RAG approach to mitigate these limitations: grounding recommendations in a knowledge graph enhances accuracy and provides implicit explainability, while LLM capabilities enable flexible EDA refinement and code generation.

**Natural Language to Visualization (NL2Vis)** systems generate visualizations from a dataset guided by a natural language query specifying the desired analysis or target columns (Hu et al., 2024; Yang et al., 2024; Maddigan and Susnjak, 2023; Zhang et al., 2024), often using LLMs for interpretation and generation. This task is distinct from and largely orthogonal to ours. While NL2Vis receives the analytical specification via the query, a core challenge for RAGvis is the automatic selection and prioritization of relevant EDA operations directly from the raw dataset itself.

## 6 Conclusion

We introduced RAGvis, a novel RAG approach designed to automatically generate comprehensive and executable EDA notebooks directly from raw datasets. RAGvis utilizes an EDA-augmented knowledge graph built offline from real-world EDA notebooks. Given a new dataset, RAGvis performs novel EDA retrieval and alignment, LLM-based refinement, and generates the corresponding Python code via a self-correcting agent. Our comprehensive evaluation shows significant improvements over an LLM-only baseline in recall, pass rate, visual quality, and cost. Notably, RAGvis allows smaller, open-source models to achieve highly competitive results, highlighting the practical benefits of combining structured knowledge with LLM generation for automating complex data science tasks.

## Limitations

While RAGvis demonstrates promising results, we identify several avenues for future expansion. Firstly, our current implementation primarily addresses EDA for single-table tabular datasets. Extending RAGvis to effectively handle multi-table relational datasets or non-tabular data modalities (e.g., images, time series) would require significant adaptations to our EDA taxonomy, semantic extraction, and the EDA retrieval and alignment processes. Secondly, our EDA taxonomy was developed through an empirical survey of Kaggle notebooks. While it was designed to be broad, it might not cover the entire spectrum of all possible EDA operations, particularly those that are highly specialized or unique to domains not extensively represented on Kaggle. Therefore, it could be expanded further. Exploring EDA patterns from other domains, such as private industry settings, could enrich the concrete chart types in our taxonomy. Finally, evaluating EDA quality is inherently challenging due to its subjectivity. We utilized metrics for semantic accuracy (recall) and visual quality (VLM score). However, dimensions like the insightfulness an EDA operation or the cognitive load required to understand it might not be fully covered. Developing more comprehensive and nuanced evaluation metrics remains an important direction for future research in automated EDA.

## References


Anaconda. 2022. Anaconda 2022 state of data science.

Ori Bar El, Tova Milo, and Amit Somech. 2020. Automatically generating data exploration sessions using deep reinforcement learning. SIGMOD '20, page 1527–1537. Association for Computing Machinery.

Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2025. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*, 31(1):1301–1311.

Victor Dibia. 2023. LIDA: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 113–126.

Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications*, 39(5):33–46.

Gemini-Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, and 1331 others. 2024. Gemini: A family of highly capable multimodal models.

Mossad Helali, Niki Monjazeb, Shubham Vashisth, Philippe Carrier, Ahmed Helal, Antonio Cavalcante, Khaled Ammar, Katja Hose, and Essam Mansour. 2024. Kglids: A platform for semantic abstraction, linking, and automation of data science. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 179–192.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8).

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. InfiAgent-DABench: Evaluating agents on data analysis tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 19544–19572. PMLR.

Jun-Peng Jiang, Si-Yang Liu, Hao-Run Cai, Qile Zhou, and Han-Jia Ye. 2025. Representation learning for tabular data: A comprehensive survey.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Kaggle. 2022. Kaggle ml survey 2022. accessed: 2025-05-01.

Moe Kayali, Fabian Wenz, Nesime Tatbul, and Çağatay Demiralp. 2025. Mind the data gap: Bridging llms to enterprise data integration. In *Proceedings of the 2025 Conference on Innovative Data Systems Research*.

Pingchuan Ma, Rui Ding, Shi Han, and Dongmei Zhang. 2021. Metainsight: Automatic discovery of structured knowledge for exploratory data analysis. In *Proceedings of the 2021 International Conference on Management of Data*, page 1262–1274. Association for Computing Machinery.

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.

Paula Maddigan and Teo Susnjak. 2023. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *IEEE Access*, 11:45181–45193.

meta-kaggle. 2022. https://www.kaggle.com/kaggle/meta-kaggle.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report.

Jim Plotts and Megan Risdal. 2023. Meta kaggle code.

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350.

Revathy Venkataramanan, Aalap Tripathy, Tarun Kumar, Sergey Serebryakov, Annmary Justine, Arpit Shah, Suparna Bhattacharya, Martin Foltin, Paolo Faraboschi, Kaushik Roy, and Amit Sheth. 2025. Constructing a metadata knowledge graph as an atlas for demystifying ai pipeline optimization. *Frontiers in Big Data*, Volume 7 - 2024.

Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, page 2648–2659. Association for Computing Machinery.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, Zhiyuan Liu, Xiaodong Shi, and Maosong Sun. 2024. MatPlotAgent: Method and evaluation for LLM-based agentic scientific data visualization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11789–11804, Bangkok, Thailand. Association for Computational Linguistics.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7443–7464.

Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. 2024. MLCopilot: Unleashing the power of large language models in solving machine learning tasks. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2931–2959. Association for Computational Linguistics.

# A Appendix

This section contains additional details of experiments and method design.

## A.1 KG-Assisted EDA Abstraction Algorithm

Algorithm 2 illustrates the approach described in Section 2.3.

**Algorithm 2:** KG-Guided EDA Semantic Extraction

**Input:** EDA Notebook $\mathcal{N}$ consisting of $n$ EDA blocks $\{b_1, ..., b_n\}$,
Data Science Knowledge Graph $\mathcal{G}$, Large Language Model $LLM$

**for** $b \in \mathcal{N}$:
  block_context = { }
  **for** *Statement:* $s \in b$ :
    parents = {SELECT p FROM $\mathcal{G}$ WHERE p $\xrightarrow{\text{DataFlow}\{1...*\}} s$}
    block_context.add(parents)
  block_context.sort() ▷ Sort by statement order
  backtracked_block = block_context + $b$
  $EDA\_OP_b$ = $LLM$("Abstract EDA Code", backtracked_block)
  $\mathcal{G} \leftarrow \mathcal{G} \bigcup EDA\_OP_b$
return $\mathcal{G}$

## A.2 Benchmark Comparison

We utilized two benchmarks in our evaluation: VisEval (Chen et al., 2025) and our own KaggleVisBench. Table 4 illustrates the differences between the two benchmarks.

Table 4: A comparison between the utilized benchmarks. While VisEval has more tables, KaggleVisBench has higher variety and number of EDA operations. Median values shown between brackets.

| | VisEval | KaggleVisBench |
|---|---|---|
| # Tables | 239 | 50 |
| Avg. # EDA Ops. | 2.9 (2) | 20.0 (16) |
| Supported Charts | Bar, Pie, Scatter, Line | Bar, Pie, Scatter, Line, Box, ColumnCorr., Histogram, HeatMap, PairwisePlot |
| Annotation | Manually-labelled | Automatically-labelled |
| Total Size (MB) | 4.2 | 1,547.3 |
| Avg. # Rows | 291.6 (12) | 16,514.9 (4,500) |
| Avg. # Columns | 6.2 (6) | 13.2 (12) |

## A.3 Supplementary Experiments

### A.3.1 Impact of EDA Retrieval Methods

This study investigates how different strategies for retrieving similar columns—which subsequently determines the retrieved EDA operations—affect performance. We compare four settings: i) **LLM-Refined:** The full RAGvis pipeline output, using Gemini 2.5 Pro for the refinement step, ii) **Content-Label Retrieval:** Uses the direct output after retrieving and aligning EDA operations based on column similarity calculated using both column name and raw values embeddings. This is the default retrieval strategy in RAGvis, presented here without LLM refinement, iii) **Content Retrieval:** Uses only raw value embeddings for column similarity (no LLM refinement), and iv) **Label Retrieval:** Uses only column name embeddings for column similarity (no LLM refinement).

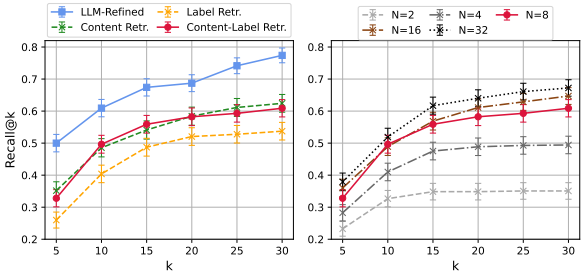


Figure 6: Ablation studies on VisEval (Chen et al., 2025) with different embedding retrieval methods (left) and different numbers of similar columns to retrieve, $N$ (right).

Observing the results for the retrieval-only methods in Figure 6 (left). Comparing the three retrieval strategies on VisEval, using only Label embeddings performs slightly worse than using Content or Content-Label embeddings, indicating that cell values are better predictors of column similarity in the benchmark. Notably, our default Content-Label approach performs consistently well across both datasets among the retrieval-only methods.

### A.3.2 Number of Retrieved Similar Columns

We also analyzed the sensitivity of RAGvis's Recall to N, the number of similar columns retrieved per target column, testing values from 2 to 32 while fixing the retrieval method to Content-Label embeddings. Although larger N retrieves more initial EDA operation candidates, we select the final top-k based on frequency to have a fair comparison between the different settings, meaning N primarily impacts the confidence and ranking of the suggested EDA operations. Figure 6 (right) illustrates the results. On VisEval, Recall@k clearly benefits from larger N, plateauing around N=8-32. These results suggest N=8, used in other experiments, is a suitable default.

### A.3.3 Evaluation of Different LLMs

The following table shows a comparison between the Recall (R@k) and Pass Rate (PR@k) for RAGvis and LIDA with more LLMs on the two benchmarks used.

| Method | VisEval | | | | | | KaggleVisBench | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R@5 | R@15 | R@30 | PR@5 | PR@15 | PR@30 | R@5 | R@15 | R@30 | PR@5 | PR@15 | PR@30 |
| **Gemini 2.5 Pro (25/03)** | | | | | | | | | | | | |
| **RAGvis** | **0.5** | **0.67** | **0.77** | **99.9%** | **100%** | **100%** | **0.12** | **0.29** | **0.43** | **99.2%** | **99.9%** | **99.5%** |
| **LIDA** | 0.15 | 0.36 | 0.49 | 95.6% | 90.3% | 96.3% | 0.01 | 0.06 | 0.14 | 89.2% | 92.4% | 93.5% |
| **Gemini 2.0 Flash** | | | | | | | | | | | | |
| **RAGvis** | **0.46** | **0.66** | **0.72** | **100%** | **99.9%** | **100%** | **0.12** | **0.23** | **0.34** | **100%** | **100%** | **99.8%** |
| **LIDA** | 0.16 | 0.44 | 0.48 | 92.5% | 90.7% | 91.0% | 0.03 | 0.11 | 0.17 | 77.6% | 81.6% | 84.2% |
| **Claude Sonnet 3.7** | | | | | | | | | | | | |
| **RAGvis** | **0.43** | **0.66** | **0.74** | **100%** | **100%** | **100%** | **0.1** | **0.27** | **0.43** | **100%** | **100%** | **100%** |
| **LIDA** | 0.12 | 0.28 | 0.38 | 90.3% | 91.6% | 91.8% | 0.02 | 0.06 | 0.16 | 78.0% | 82.5% | 88.0% |
| **GPT-4o mini** | | | | | | | | | | | | |
| **RAGvis** | **0.33** | **0.59** | **0.68** | **100%** | **99.8%** | **99.8%** | **0.07** | **0.18** | **0.3** | **99.6%** | **99.2%** | **99.1%** |
| **LIDA** | 0.24 | 0.40 | 0.52 | 93.2% | 88.4% | 89.3% | 0.06 | 0.12 | 0.18 | 84.4% | 81.6% | 72.3% |
| **Gemma 3 12b** | | | | | | | | | | | | |
| **RAGvis** | **0.34** | **0.63** | **0.7** | **98.5%** | **97.7%** | **95.0%** | **0.07** | **0.18** | **0.22** | **96.8%** | **96.3%** | **95.7%** |
| **LIDA** | 0.14 | 0.28 | 0.43 | 88.9% | 88.7% | 91.1% | 0.02 | 0.09 | 0.13 | 78.4% | 82.9% | 66.5% |
| **Gemma 3 4b** | | | | | | | | | | | | |
| **RAGvis** | **0.22** | **0.46** | **0.61** | **98.3%** | **98.6%** | **98.2%** | **0.05** | **0.1** | **0.15** | **92.0%** | **89.2%** | **87.3%** |
| **LIDA** | 0.03 | 0.15 | 0.28 | 86.6% | 93.3% | 90.3% | 0.01 | 0.06 | 0.11 | 79.2% | 77.9% | 68.3% |

## A.4 RAGvis Prompt Templates

### EDA Semantic Extraction Prompt

Your task is to analyze an EDA code snippet and extract answers for the following questions:

1. Is the analysis in the code univariate, bivariate, or multivariate?
2. What is the high level chart category? Possible chart category types: Bar, Pie, Distribution (e.g. Histogram, KDEPlot), Box (e.g. BoxPlot, ViolinPlot), Scatter (e.g. ScatterPlot, RegressionPlot), Line, ColumnCorrelation, HeatMap, Pairwise.
3. Which columns in the original table are used in the chart axes?
4. Is the data grouped by a certain column before plotting (different from chart axes)?

The response must be in JSON in the following format:

```
{
"analysis_type": "univariate | bivariate | multivariate",
"chart_type": "Bar | Pie | Distribution | Box | Scatter | Line | ColumnCorrelation | HeatMap | Pairwise",
"chart_axes_columns": ["<COLUMN_NAME>", ...],
"grouping_column": "<COLUMN_NAME>", // null if no grouping column
}
```

If the code snippet contains multiple EDA operations, return a list of json objects, one for each operation. If the previous information is not available or the code snippet is not an EDA code, return an empty json object. return only the raw json object without any code.

Here is some example inputs and outputs:

EXAMPLE 1:
EXAMPLE 1 Input:

Table:
Country name,Regional indicator,Ladder score,upperwhisker,lowerwhisker,Log GDP per capita,Social support,Healthy life expectancy,Freedom to make life choices,Generosity,Perceptions of corruption,Dystopia + residual
Finland,Western Europe,7.741,7.815,7.667,1.844,1.572,0.695,0.859,0.142,0.546,2.082
Denmark,Western Europe,7.583,7.665,7.500,1.908,1.520,0.699,0.823,0.204,0.548,1.881
Iceland,Western Europe,7.525,7.618,7.433,1.881,1.617,0.718,0.819,0.258,0.182,2.050
Sweden,Western Europe,7.344,7.422,7.267,1.878,1.501,0.724,0.838,0.221,0.524,1.658

Code:
```
df2024 = pd.read_csv("/kaggle/input/world-happiness-report-2024-yearly-updated/World-happiness-report-2024.csv", encoding='latin-1')

plt.figure(figsize = (15,8))
sns.kdeplot(x=df2024['Ladder score'], hue = df2024['Regional indicator'], fill = True, linewidth = 2)
plt.axvline(df2024['Ladder score'].mean(),c= 'black')
plt.title('Ladder Score Distribution by Regional Indicator')
plt.show()
```

EXAMPLE 1 Output:
```
[{
"analysis_type": "bivariate",
"chart_type": "Distribution",
"chart_axes_columns": ["Ladder score"],
"grouping_column": "Regional indicator",
}]
```

EXAMPLE 2:
EXAMPLE 2 Input:

Table:
year_film,year_award,ceremony,category,nominee,film,win
1943,1944,1,Best Performance by an Actress in a Supporting Role in any Motion Picture,Katina Paxinou,For Whom The Bell Tolls,True
1943,1944,1,Best Performance by an Actor in a Supporting Role in any Motion Picture,Akim Tamiroff,For Whom The Bell Tolls,True
1943,1944,1,Best Director - Motion Picture,Henry King,The Song Of Bernadette,True
1943,1944,1,Picture,The Song Of Bernadette,,True

Code:
```
df = pd.read_csv('/kaggle/input/golden-globe-awards/golden_globe_awards.csv')

film_awards_year = df.groupby(['film', 'year_award'])['win'].sum()

film_award_df = pd.DataFrame(film_awards_year).reset_index()
film_four_awards = film_award_df[film_award_df['win'] >= 4].sort_values(ascending = False, by='win')

film_four_awards = film_four_awards.rename({'film': 'movie', 'win': 'awards'})

plt.figure(figsize=(20,8))
sns.set_style('whitegrid')
sns.barplot(x='movie', y='awards', data=film_four_awards, palette='hls')
plt.title('Movies with atleast 4 awards in a single year', fontsize=12)
plt.xlabel('Movie', fontsize=12)
plt.ylabel('Award Count', fontsize=12)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(rotation=90, fontsize=12)
plt.show()
```

EXAMPLE 2 Output:
[{
"analysis_type": "bivariate",
"chart_type": "Bar",
"chart_axes_columns": ["film", "win"],
"grouping_column": null,
}]

EXAMPLE 3:
EXAMPLE 3 Input:

Table:
#,Name,Type 1,Type 2,Total,HP,Attack,Defense,Sp. Atk,Sp. Def,Speed,Generation,Legendary
1,Bulbasaur,Grass,Poison,318,45,49,49,65,65,45,1,False
2,Ivysaur,Grass,Poison,405,60,62,63,80,80,60,1,False
3,Venusaur,Grass,Poison,525,80,82,83,100,100,80,1,False
3,VenusaurMega Venusaur,Grass,Poison,625,80,100,123,122,120,80,1,False

Code:
```
measurements = pd.read_csv('/kaggle/input/air-pollution-in-seoul/AirPollutionSeoul/Original Data/Measurement_info.csv')
measures = measurements.pivot_table(index=['Measurement date', 'Station code', 'Instrument status'], columns='Item code',
values='Average value').reset_index()

measures = measures.loc[measures['Status'] == 'Normal', :]
overview = measures.groupby('Date').mean().loc[:, 'SO2 (ppm)':'PM2.5 (mircrogram/m3)']
for pol, func in evaluators.items():
    overview[pol.split()[0] + ' Level'] = overview[pol].map(func)

level_counts = pd.concat([overview[col].value_counts() for col in overview.loc[:, 'SO2 Level':]], axis=1, join='outer', sort=True).fillna(0.0)
level_counts = level_counts.loc[['Very bad', 'Bad', 'Normal', 'Good'], :]

level_counts.T.plot(kind='bar', stacked=True, figsize=(8,6), rot=0,
                    colormap='coolwarm_r', legend='reverse')
plt.title('Levels of pollution in Seoul from 2017 to 2019', fontsize=16, fontweight='bold')
plt.show()
```

EXAMPLE 3 Output:
[]

Keeping in mind the previous examples, what is the corresponding output for the following table sample and code snippet?

Table:
**{{TABLE_SAMPLE}}**

Code:
```
**{{SOURCE_CODE}}**
```

## EDA Refinement Prompt

I have a dataset in CSV format with the following information about its columns:

**{{COLUMN_INFO}}**

Your task is to predict a list of EDA operations that comprehensively analyze the dataset.
Each EDA operation consists of the chart type and the columns to analyze.

The following are suggestions based on historical EDA operations for similar datasets, sorted from the least to the most frequent:

**{{EDA_OPERATIONS}}**

The previous are suggestions for EDA operations. They might have incorrect columns or chart types. You may remove EDA operations that are not suitable for this dataset. You may also add EDA operations that are are not in the previous list.
Before predicting the EDA operations, think about the following:
1. Which are the most important columns in the dataset to analyze?
2. What univariate, bivariate and multivariate analyses are suitable for this dataset?
3. Which EDA operations in the previous list are applicable to this dataset?
4. Are there any relevant or infrormative EDA operations not in the previous list?
5. The EDA operations in the previous list are sorted from the least to the most frequent.
6. You must generate EXACTLY **{{NUM_EDA_OPERATIONS}}** DISTINCT EDA operations.

The generated EDA operations must be in the following JSON format:
[{"chart_type": "<CHART_TYPE>",
"chart_columns": ["<COLUMN_NAME>", "<COLUMN_NAME>", ...]}]

Possible values for <CHART_TYPE>: Bar | Pie | Distribution | Box | Scatter | Line | ColumnCorrelation | HeatMap | Pairwise

For exapmle, a bar chart with columns `name` and `age` would have the following corresponding JSON:
[{"chart_type": "Bar", "chart_columns": ["name", "age"]}]

You must generate exactly **{{NUM_EDA_OPERATIONS}}** EDA operations. Return only the JSON list.

---

**{{COLUMN_INFO}} Example:**

Column `Age` contains values of type integer. It has 30 unique values and 2 missing values.
Column `Name` contains values of type natural language. It has 2300 unique values and 0 missing values.
...

---

**{{EDA_OPERATIONS}} Example:**

A Bar Chart using the using the column(s): {`Age`, `Gender`, `City`}.
A Scatter Plot using the column(s): {`Age`, `Salary`} grouped by the column `City`.
...

## EDA Coding Agent - Code Generation Prompt

I have a dataset in CSV format with the following information about its columns:

**{{COLUMN_INFO}}**

Your task is to write the Python code that generates the following chart:
A **{{CHART_TYPE}}** chart showing the column(s): **{{CHART_COLUMNS}}**

Instructions:
- import all necessary libraries
- read the dataset from the following file: `**{{DATASET_FILE_NAME}}**`
- perform only the requested analysis. The code should generate EXACTLY ONE chart that uses the specified columns.
- You may group by any of the given columns if suitable.
- DO NOT include plt.show() or any other blocking statement in the code.
- DO NOT save the figure to a file.
- DO NOT create user-defined functions. Have all the code in the top-level code.
- Write only the Python code without any explanation or comments.

**{{COLUMN_INFO}} Example:**

Column `Age` contains values of type integer. It has 30 unique values and 2 missing values.
Column `Name` contains values of type natural language. It has 2300 unique values and 0 missing values.
...

## EDA Coding Agent - Code Repair Prompt

The code has produced the following error:

```
**{{ERROR_STACK_TRACE}}**
```

Change your code to fix the error. DO NOT include explanation of the error or fix.
Write only the fixed code.

## EDA Notebook Assembly Prompt

I have a dataset in CSV format saved in the following file: `**{{DATASET_FILE_NAME}}**`

The following are code segments that perform EDA analysis on the dataset.
Your task to combine all the code segment into a single notebook.

Code Segments:

**{{EDA_CODE_SNIPPETS}}**

Instructions:
- Combine the previous code into a single notebook. Remove the redundant code.
- Add plt.show() after each plot.
- Include the necessary imports.
- You may add comments to explain the code.
- DO NOT change the EDA analysis code itself.

### A.5 VLM-as-Judge Prompt Template

The following is the prompt template for using a Vision Language Model (VLM) as a judge for evaluating the generated chart quality. This prompt is proposed by the authors of VisEval (Chen et al., 2025) and was shown to highly correlate with human expert ratings.

**Vision Language Model Judge Prompt Proposed By VisEval (Chen et al., 2025)**

**System Instructions**

Your task is to evaluate the readability of the visualization on a scale of 1 to 5, where 1 indicates very difficult to read and 5 indicates very easy to read. You will be given a visualization requirement and the corresponding visualization created based on that requirement. Additionally, reviews from others regarding this visualization will be provided for your reference. Please think carefully and provide your reasoning and score.

```
{
    "Rationale": "a brief reason",
    "Score": 1-5
}
```

Examples:

- If the visualization is clear and information can be easily interpreted, you might return:

```
{
    "Rationale": "The chart is well-organized, and the use of contrasting colors helps in distinguishing different data sets effectively. The labels are legible, and the key insights can be understood at a glance.",
    "Score": 5
}
```

- Conversely, if the visualization is cluttered or confusing, you might return:

```
{
    "Rationale": "While there is no overflow or overlap, the unconventional inverted y-axis and the use of decimal numbers for months on the x-axis deviate from the standard interpretation of bar charts, confusing readers and significantly affecting the chart's readability.",
    "Score": 1
}
```

**Prompt**

Visualization Requirement: **{{EDA_OPERATION}}**
Visualization image: **{{CHART_IMAGE}}**

Please assess the readability, taking into account factors such as layout, scale and ticks, title and labels, colors, and ease of extracting information. Do not consider the correctness of the data and order in the visualizations, as they have already been verified.

**{{EDA_OPERATION}} Example:**

A Scatter Plot using the column(s): {`Age`, `Salary`} grouped by the column `City`.