TabaQA at SemEval-2025 Task 8: Column Augmented Generation for Question Answering over Tabular Data

Ekaterina Antropova[1] **Egor Kratkov**[1] **Roman Derunets**[4,5]
Margarita Trofimova[1] **Ivan Bondarenko**[4] **Alexander Panchenko**[3,2]
Vasily Konovalov[2,1] **Maksim Savkin**[1]
[1]Moscow Institute of Physics and Technology
[2]AIRI [3]Skoltech [4]Novosibirsk State University
[5]Siberian Neuronets LLC
{antropova.eg, savkin.mk, vasily.konovalov}@phystech.edu

Abstract

The DataBench shared task in the SemEval-2025 competition aims to tackle the problem of QA from data in tables. Given the diversity of the structure of tables, there are different approaches to retrieving the answer. Although Retrieval-Augmented Generation (RAG) is a viable solution, extracting relevant information from tables remains challenging. In addition, the table can be prohibitively large for direct integration into the LLM context. In this paper, we address QA over tabular data first by identifying relevant columns that might contain the answers, then the LLM generates answers by providing the context of the relevant columns, and finally, the LLM refines its answers. This approach secured us 7th place in the DataBench lite category.

1 Introduction

Question Answering (QA) is a long-standing challenge in artificial intelligence, with numerous variations and applications. QA systems are extensively used in a variety of real-world scenarios, such as virtual assistants and chatbots for customer support. They serve as powerful tools for extracting relevant information from large and diverse datasets.

A crucial challenge arises when the required information is not just embedded in natural language but also stored in structured formats, such as tabular data. The primary challenge in tabular QA stems from the need to bridge the gap between structured data representation and natural language understanding. While substantial progress has been made in developing models that handle either natural language or structured data independently, effectively integrating both remains an open research problem.

Traditional retrieval methods struggle to select relevant table segments, and large language models (LLMs) can be inefficient when tasked with processing extensive tabular data directly. To advance research in this area, the DataBench shared task (Os'es Grijalba et al., 2025) was introduced, focusing on answering questions from tabular datasets. In particular, the shared task proposes to answer the question on the DataBench (Grijalba et al., 2024) datasets and a smaller DataBench lite version (a reduced version of each dataset from the original DataBench). In this paper, we focus primarily on a lite subtask. Our approach incorporates several key strategies inspired by existing methodologies. First, we observe that the number of relevant columns required to answer each question is typically limited. We leverage column augmented generation (CAG), a technique that dynamically selects relevant columns based on LLM-generated hints and helps retain only the most relevant columns. CAG addresses multiple challenges simultaneously by filtering out irrelevant columns, reducing noise, and significantly shrinking the table size, which enhances model efficiency. We then explore the effect of iterative self-refinement by employing a feedback loop where the model iteratively refines its own responses. Finally, we employ a self-consistency prompting technique to improve the reliability of stochastic LLM generations by aggregating multiple generated answers.

By systematically analyzing the impact of each individual modification, we investigate how these techniques can be effectively combined to make our best performing solution. Our submission is built entirely on open-source models and achieves competitive results: we rank 7th out of 35 teams on the DataBench lite leaderboard and secure third place in the small models category ($\leq$8B) in the post-evaluation phase (see Table 2). Our findings highlight the potential of combining structured data processing techniques with robust QA methodologies to improve tabular QA capabilities.

Our main contributions are threefold:

- **Column Augmented Generation** We enhance context retrieval by dynamically selecting relevant table segments using LLM-generated column hints, improving the quality of retrieved evidence.

- **Reasoning Prompts** We systematically experiment with different reasoning strategies to enhance LLM reasoning capabilities and identify the most effective prompting techniques.

- **Reasoning Scaling** We analyze how different levels of reasoning complexity influence performance, demonstrating that increased reasoning depth can significantly improve tabular QA accuracy.

2 Related Work

Prior work in tabular QA falls into two main paradigms. **Text-to-SQL (or Text-to-Code)** methods transform natural language queries into executable code. We further distinguish several subtypes. *Schema-Aware Parsing* models such as TAPAS (Herzig et al., 2020), T5-SQL (Arcadinho et al., 2022), TAPEX (Liu et al., 2021) and OmniTab (Jiang et al., 2022) learn representations that encode table schemas (column names, data types) to generate SQL queries. *Intermediate Logical Forms* such as DIN-SQL (Pourreza and Rafiei, 2023) first parse questions into intermediate logical forms (e.g., step-by-step operations) before converting to SQL, improving robustness. *Text-to-Code Extensions* like text2pandas (Venturi, 2023) that leverage libraries such as `pandas` for more flexible data manipulation. This strategy is particularly effective when dealing with irregular or heterogeneous table structures.

Retrieval-Augmented Generation (RAG) techniques condition the LLM's output on retrieved evidence from the table, thereby reducing hallucination and grounding the generation in the provided data. Within this category, we identify several subtypes. *Table-Segment Retrieval* methods like those described in TableRAG (Chen et al., 2024) dynamically retrieve relevant segments of a table based on the query, ensuring that the model's focus is limited to the most informative sections. *Row-Column Retrieval* approaches such as ITR (Lin et al., 2023) and TAP4LLM (Sui et al., 2023) address scalability by selectively encoding and retrieving essential rows and columns. ReasTAP (Zhao et al., 2022) demonstrates that high-level table reasoning sckills can be injected into models during pre-training without comples table-specific architecture design. While this strategy reduces input lengths for downstream reasoning, it often incurs substantial computational overhead and may suffer from degraded embedding quality when processing long sequences. *Hybrid Retrieval and Synthesis* methods such as ReAcTable (Zhang et al., 2023) combine retrieval of relevant context with program synthesis and iterative refinement. Although ReAcTable leverages retrieval cues similar to RAG methods, its integration of an execution-based refinement process places it at the intersection of text-to-SQL and RAG paradigms. H-STAR (Abhyankar et al., 2024) advances the field by introducing a hybrid prompting strategy that dynamically adapts between structured and unstructured approaches based on the complexity of the query, allowing for more efficient processing while maintaining high accuracy across query types. Hybrid systems demonstrate the potential of combining multiple paradigms and leveraging the strengths of each methodology.

Inspired by TableRAG's ability to handle extensive table data, recent RAG methods have demonstrated that dynamically retrieving and conditioning on relevant table segments can significantly improve performance, especially in the presence of complex or unstructured tables. By focusing on the most informative parts of a table, these approaches better ground the LLM's output, thereby reducing errors and hallucinations. However, they depend critically on the quality of the retrieval process — noisy or imprecise retrieval can compromise performance.

3 Dataset

The DataBench dataset consists of 65 publicly available datasets, with 3 269 975 and 1615 columns in total, and 1300 questions in 5 domains: Health public data, Business data, Social Networks and Surveys including X (formerly Twitter), Sports and Entertainment data and Travel and Locations data. The dataset is split into three sections: training (comprising 988 questions), testing (featuring 522 questions and 517 questions after changes), and development (consisting of 320 questions) (Grijalba et al., 2024).

Column types presented in DataBench:

- Boolean: Valid answers include True/False, Y/N, Yes/No (all case insensitive).

- Category: A value from a cell (or a substring of a cell) in the dataset.
- Number: A numerical value from a cell in the dataset, which may represent a computed statistic (e.g., average, maximum, minimum).
- List[category]: A list containing a fixed number of categories. The expected format is: "['cat', 'dog']". Pay attention to the wording of the question to determine if uniqueness is required or if repeated values are allowed.
- List[number]: Similar to List[category], but with numbers as its elements.

The DataBench lite represents a smaller sample with the first 20 rows.

4 Evaluation

Participants must answer questions using only the provided dataset, including development and training sets, without external data. Accuracy scores for DataBench and DataBench lite are ranked separately. An automated evaluation framework[1] streamlines the process, enabling customization of prompt building, model calling, and result evaluation, while supporting batch processing and standardized response handling.

The default evaluation function compares submissions against the truth set, featuring:

- Null: Treats null-like values as equivalent.
- Boolean: Normalizes inputs and checks against true/false lists.
- Category/String: Strips and compares strings, with date parsing for format variations.
- Number: Rounds values to two decimal places for tolerance.
- List[Category]: Normalizes and checks set equivalence (order-agnostic).
- List[Number]: Normalizes, rounds, and ensures order-agnostic equivalence.

In addition, organizers decided to manually review the results of the top scores to ensure fairness and accuracy in determining the winner.

[1] https://github.com/jorses/databench_eval

Model	Precision	Recall	F1
Llama-3.3-70B-Instruct	84.60	98.33	90.95
Llama-3.1-8B-Instruct	61.80	96.90	75.46

Table 1: The micro-averaged results for detecting the required columns (on dev)

Model	EM
Python Baselines	
CodeLlama-7B (Grijalba et al., 2024)	30.3
CodeLlama-13B (Grijalba et al., 2024)	33.1
Llama-3.1-8B-Instruct	47.51
Llama-3.1-70B-Instruct	66.28
Zero-Shot In-Context Learning Baselines	
Llama-2-7B (Grijalba et al., 2024)	14.8
Llama-2-13B (Grijalba et al., 2024)	20.7
Llama-3.1-8B-Instruct	27.78
+CAG	37.93
+CAG+Refine	47.13
+CAG+CoT	54.59
+CAG+CoT+Refine	52.30
+CAG+CoT+Consistency(10)	66.86
Llama-3.3-70B-Instruct	49.81
+CAG	64.37
+CAG+Refine	69.36
+CAG+CoT	81.03
+CAG+CoT+Refine	77.59
+CAG+CoT+Consistency(10)	81.99
DeepSeek-R1-32B	77.78
+CAG	**83.52**

Table 2: The results for DataBench lite test set. EM denotes the exact match metric with custom comparison function presented by competitions authors, see Section 4. The upper sections contains the baselines approaches. Our approaches (denoted as CAG) consistently outperforms all the baselines and can be further improved by applying advanced prompt engineering techniques.

5 System Overview

5.1 Column Augmented Generation

The Column Augmented Generation (CAG) approach is a two-step method inspired by Retrieval-Augmented Generation (RAG) techniques. Since large tables cannot be easily incorporated into an LLM's context due to size constraints, the first step in our pipeline involves identifying the most relevant columns needed to answer a given question

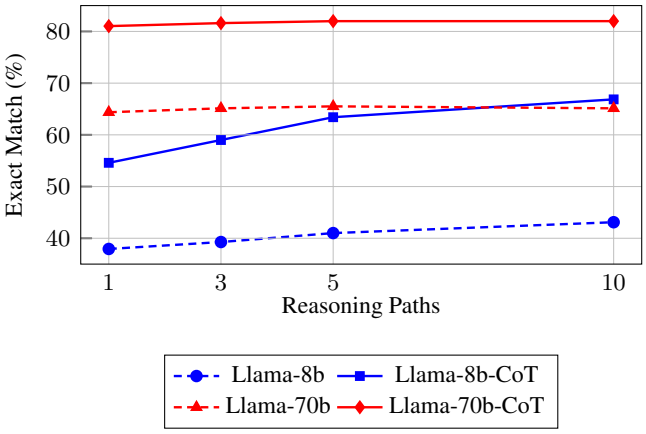

Figure 1: Self-Consistency results for DataBench lite test set. Exact Match is a metric with custom comparison function presented by competitions authors, see Section 4. Reasoning Paths means the number of times the model was called to generate an answer, see Section 5.2. CoT denotes Chain-of-Thought

(see Appendix A for the relevant prompt). This column selection process is treated as a multi-label classification task, evaluated using micro-averaged precision, recall, and F1 score (see Table 1). The selected column names are then validated for format compliance, and if necessary, the generated response is refined or corrected.

Once the relevant columns are identified, the second step involves prompting the LLM again, this time with the extracted column data included as context. This ensures that the model focuses only on the most relevant information when generating an answer. The CAG approach aligns closely with techniques used in TableRAG (Chen et al., 2024), improving both passage quality and model efficiency by dynamically narrowing the input scope.

5.2 Reasoning Prompting Techniques

In addition to the CAG technique, we incorporated advanced prompt engineering methods to enhance reasoning abilities, specifically Chain-of-Thought (CoT) (Wei et al., 2022), Self-Consistency (Wang et al., 2023), and Self-Refine (Madaan et al., 2023). These techniques can be applied independently or in combination with CAG to improve its effectiveness.

The Self-Consistency method involves calling the model multiple times and aggregating the results using a majority vote to determine the most frequent answer. We experimented with different numbers of model calls, such as 3, 5, and 10, to assess its impact.

The Chain-of-Thought technique encourages step-by-step reasoning by instructing the model to break down its thought process. This approach improves the model's focus on intermediate reasoning steps while also increasing token generation. To implement this, we used few-shot prompting with explicit CoT examples. Additionally, we explored the combination of CoT and Self-Consistency to further refine the reasoning process.

Finally, we tried applying Self-Refine as a post-generation technique. The model was instructed to evaluate its own responses, providing feedback on correctness and answer format. This feedback was then used to guide subsequent answers, leading to iterative improvements.

6 Experimental Setup

For our experimental setup we used two LMs: (1) `Llama-3.1-8B-Instruct`[2] is a language model with 8 billion parameters and is optimized for conversational applications, supporting a contextual length of up to 128 thousand tokens; (2) `Llama-3.3-70B-Instruct`[3] is a multitasking language model with 70 billion parameters and is optimized for conversational applications, supporting a contextual length of up to 128 thousand tokens (Touvron et al., 2023). In addition, we evaluate our CAG approach on DeepSeek-R1-32B[4] that has shown advanced reasoning abilities over the LMs of a comparable number of parameters (DeepSeek-AI, 2025). All LMs were evaluated for `temperature = 0.7`

7 Results and Discussion

Our main results on the DataBench lite test split are summarized in Table 2.

The findings confirm that the 70B model consistently outperforms the smaller 8B model across all settings. This is evident in the first step of the CAG pipeline, where LLMs identify relevant columns, as shown in Table 1. The 70B Llama achieves a micro-average F1 score of 90.95%, outperforming the 8B models.

Table 2 further demonstrates that the CAG technique significantly enhances performance across all prompting strategies that do not require code execution, yielding an absolute percentage increase of

[2]Llama-3.1-8B-Instruct
[3]Llama-3.3-70B-Instruct
[4]DeepSeek-R1-Distill-Qwen-32B

10.1 and 12.6 for the 8B and 70B models, respectively. When combined with Chain-of-Thought, performance improves even further, with percentage gains of 26.8 and 19.5 over the baseline prompt.

Reasoning-based prompting techniques, including CoT, self-refinement, and self-consistency, contribute to additional accuracy gains when combined with CAG. By aggregating multiple responses through self-consistency, we aimed to mitigate the randomization effect of suboptimal prompts, thereby enhancing model reliability. This approach proved effective, boosting EM scores by 12.3 and 1 absolute percent when compared to the CAG+CoT approach. Experiments with different numbers of reasoning paths, as illustrated in Figure 1, indicate a stable increase in EM scores up to 10 paths. Beyond this, setting up to 40 paths for the 8B model resulted in only a marginal 0.8 EM gain. Due to the high computational cost, we opted to defer further experiments.

Unlike standard CoT, which generates a reasoning chain before arriving at an answer, self-refinement enables the model to verify and correct its responses to ensure accuracy and proper format adherence. This process improves reliability, particularly in tasks requiring structured outputs such as JSON format. However, while refinement techniques improved accuracy through self-correction, pure CoT performed better. One possible explanation is that CoT inherently incorporates iterative refinements, which may have confused the model when combined with additional refinement steps.

DeepSeek-R1-32B, which performs extended reasoning before generating an answer, emerged as a strong alternative to traditional prompting techniques. It achieved an EM score of 77.78, which increased to 83.52 with CAG—the highest result obtained. Increasing test-time compute, whether through the progressive combination of prompting techniques (CAG $\rightarrow$ CAG+CoT $\rightarrow$ CAG+CoT+self-consistency $\rightarrow$ CAG+reasoning model) or through the use of reasoning-specialized models, consistently led to improved results. These findings highlight the benefits of test-time compute scaling for tabular QA.

Conclusion

In this work we applied the CAG approach for SemEval-2025 Task 8: TabularQA competition. The CAG is a two-step approach in which we first use an LLM to identify columns relevant to the question. The LLM then generates an answer based on the content of these columns. Using the CAG approach, we outperformed all of the baselines mentioned. In addition, we applied reasoning prompt engineering techniques to further improve the generated answers. Moreover, we revealed that CAG based on `DeepSeek-R1-32B` outperformed all sizes of Llamas, which confirms the superiority of the reasoning language models for TabularQA.

Limitations

Our experiments were limited by time and computational resources, which prevented us from fully optimizing the suggested prompting techniques. Wang et al. (2023) noted that increasing the number of reasoning paths could further improve performance. Additionally, we did not compute scores for the combination of self-refine and self-consistency prompting, as the computational cost of such an approach would have been prohibitively high. The `DeepSeek-R1-32B` model also presented challenges, as it generated extensive reasoning traces, often exceeding 16 000 tokens for even simple prompts, making it impractical to test additional prompting techniques within our resource constraints. Furthermore, we were unable to evaluate all potentially valuable open-source LLMs and did not test any closed-source models. Since we primarily focused on no-code solutions, we did not explore code-generation-specific optimizations in detail, though all the described approaches can be easily transferred to code generation.

References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K Reddy. 2024. H-star: Llm-driven hybrid sql-text adaptive reasoning on tables. *arXiv preprint arXiv:2407.05952*.

Samuel Arcadinho, David Aparício, Hugo Veiga, and António Alegria. 2022. T5ql: Taming language models for sql generation. *Preprint*, arXiv:2209.10254.

Si-An Chen, Lesly Miculicich, Julian Martin Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. Tablerag: Million-token table understanding with language models. *arXiv preprint arXiv:2410.04739*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Jorge Osés Grijalba, Luis Alfonso Ureña López, Eugenio Martínez Cámara, and José Camacho-Collados. 2024. Question answering over tabular data with databench: A large-scale empirical evaluation of llms. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 13471–13488. ELRA and ICCL.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pretraining with natural and synthetic data for few-shot table-based question answering. *arXiv preprint arXiv:2207.03637*.

Weizhe Lin, Rexhina Blloshmi, Bill Byrne, Adrià de Gispert, and Gonzalo Iglesias. 2023. An inner table retriever for robust table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9909–9926.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jorge Os'es Grijalba, Luis Alfonso Ure na-L'opez, Eugenio Mart'inez C'amara, and Jose Camacho-Collados. 2025. SemEval-2025 task 8: Question answering over tabular data. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, Vienna, Austria. Association for Computational Linguistics.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Preprint*, arXiv:2304.11015.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2023. Tap4llm: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. *arXiv preprint arXiv:2312.09039*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Gabriele Venturi. 2023. PandaAI: the conversational data analysis framework.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. Reactable: Enhancing react for table question answering. *Preprint*, arXiv:2310.00815.

Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir Radev. 2022. Reastap: Injecting table reasoning skills during pre-training via synthetic reasoning examples. *arXiv preprint arXiv:2210.12374*.

A CAG Prompts

CAG step 1: Prompt for extracting relevant columns

Given a table contains columns with names <*list of column names*>, I want to answer a question: <*question*>.

Please select a few column names from the list <*list of column names*>, the values of which will help answer the question.

Please provide a list of column names in the list format without any additional explanations.

Example of output: ["column name1", "column name2", "column name3"]

CAG step 2: Prompt for generating an answer

You are a data analysis assistant. Given a table with the following **columns**: <*list of retrieved column names*>, and the **table data**: <*table as string*>, answer the following **question**: <*question*>. Your task is to analyze the table and provide a concise answer to the question. The answer must strictly adhere to one of the following formats, depending on the question:

1. **Boolean**: True or False
2. **Category**: A single category as a Python string (e.g., "category_name")
3. **Number**: A single number (e.g., 42)
4. **List[Category]**: A list of strings (e.g., ['category_1', 'category_2'])
5. **List[Number]**: A list of numbers (e.g., [1, 2, 3])

Rules:
- Do not include any additional text, comments, or explanations.

Examples:
Question 1: <*question*>
Answer 1: <*answer*>
...
Now, provide the answer to the following question: <*question*>

B Reasoning Prompting Techniques

Chain-of-Thought Prompt

Example 1:
Table columns: <*list of column names*>
Question: <*question*>
Answer: <*answer*>

Example 2: ...
—

You are a data analysis assistant. You are given a table and a question. Answer the question based only on the information in the table. The answer must be in one of the 5 following formats:
- **Boolean** (True or False) if the question requires a yes/no answer.
Example: [ANSWER] True
- **Number** if the question requires a numerical response.

Example: [ANSWER] 15
- **Category** (a string) if the question requires a categorical response.
Example: [ANSWER] London
- **List[Category]**: A list of strings.
Example: [ANSWER] ['San Francisco', 'New York', 'Wuhan', 'Bangalore']
- **List[Number]**: A list of numbers.
Example: [ANSWER] [1, 2, 3, 4, 5]

Rules:
- Start the answer with: "Let's think step by step". Then give your reasoning to your answer.
- Finish your answer with [ANSWER] and give the final answer.
- [ANSWER] part should contain only one of the following types: boolean, number, category, list[category], list[number]
- Don't give any additional text, comments or explanation in the [ANSWER] part.

Table: <*table as string*>
Table columns: <*list of column names*>
Question: <*question*>
Answer:
Let's think step by step.

Self-Refine Step 1: Prompt for extracting relevant columns

You are an AI assistant specialized in data analysis.
Given a table with the following columns: <*list of column names*>
Task: Select the most relevant columns that are necessary to answer the following question:**"question"**

Step-by-step reasoning:

1. Identify the key concepts in the question.
2. Match these concepts to the relevant columns in the table.
3. If multiple columns could provide similar information, select the **most informative** ones.
4. Ensure that the chosen columns are **minimal yet sufficient**.

Example:
Question: "What is the average salary of employees in the IT department?"

- **Step 1:** Key concepts → "average salary", "IT department"
- **Step 2:** Relevant columns → ["Salary", "Department"]
- **Step 3:** "Salary" contains numerical salary data; "Department" helps filter IT employees
- **Step 4:** Selected columns → ["Salary", "Department"]

IMPORTANT:

- Your step-by-step reasoning **MUST NOT** exceed 5 sentences.
- Your final JSON **MUST** be included within the response.
- If you fail to comply, your response will be discarded as invalid.

Your Task:

Perform the same step-by-step reasoning for the given question and return the final selected columns in **strict JSON format**.

Example Outputs:

Example 1 (if the question is about employee age):
json
["Age"]

Example 2 (if the question is about employee salary growth):
json
["Salary", "YearsAtCompany", "PercentSalaryHike"]

Example 3 (if the question is about employee job satisfaction and department):
json
["Department", "JobSatisfaction"]

CRITICAL REQUIREMENTS:

- You **MUST** return your response in two parts:
  1. **Your step-by-step reasoning.**
  2. **The final validated response in STRICT JSON FORMAT.**
- The JSON response **must be the last thing in your answer** and formatted correctly.

Now, provide your response in the exact same JSON format.

Self-Refine Step2: Prompt for validating the retrieved columns

You have selected the following columns to answer the question: <*question*>

Selected columns: <*list of retrieved column names*>
Your previous reasoning was: <*list of column names*>
Full list of available columns in the dataset: <*list of column names*>

Validation Task

- If the selection is **correct and minimal**, return **"VALID"**.
- If the selection is **incorrect, incomplete, or redundant**, return the corrected column list in **strict JSON format**.
- If additional columns are **necessary**, add them.
- If some columns are **not needed**, remove them.

New Validation Rules

- You **must** justify any change in column selection.
- If the selection is **"VALID"**, explain **why**.
- If changes are needed, return **only** the corrected column list in JSON format.

IMPORTANT:

- Your step-by-step reasoning **MUST NOT** exceed 5 sentences.
- Your final JSON **MUST** be included within the response.
- If you fail to comply, your response will be discarded as invalid.

Example Outputs (JSON format)

If the selected columns are correct and minimal:
json
"VALID"

If some columns are incorrect or missing:
json
["Department", "EmployeeCount"]

If unnecessary columns are included:
json
["Age"]

CRITICAL REQUIREMENTS

- You **MUST** return your response in two parts:
  1. **Your step-by-step reasoning.**
  2. **The final validated response in STRICT JSON FORMAT.**
- The JSON response **must be the last thing in your answer** and formatted correctly.

Now, validate the selected columns and return your response in the correct JSON format.

Self-Refine Step 3: Prompt for generating an answer

You are a data analysis assistant. Given the following table: <*retrieved columns*>

Your previous reasoning was: <*response*>

Step-by-step reasoning before answering:

1. Identify the key data points in the table that are needed to answer the question.
2. Analyze how these data points interact with each other.
3. Perform any necessary calculations or logical deductions.
4. Formulate the final answer based on these steps.

IMPORTANT:

- Your step-by-step reasoning **MUST NOT** exceed 5 sentences.
- Your final JSON **MUST** be included within the response.
- If you fail to comply, your response will be discarded as invalid.

<*question*>

Answer format (strict JSON):

- Boolean: {"answer": true}
- Category: {"answer": "category_name"}
- Number: {"answer": 42}
- List of Categories: {"answer": ["category_1", "category_2"]}
- List of Numbers: {"answer": [1, 2, 3]}

CRITICAL REQUIREMENTS:

- You **MUST** return your response in two parts:
  1. **Your step-by-step reasoning.**
  2. **The final validated response in STRICT JSON FORMAT.**
- The JSON response **must be the last thing in your answer** and formatted correctly.

Now, provide the answer.

Self-Refine Step 4: Prompt for validating a generated answer

You have the following **table**: <*table as string*>
Given question: <*question*>
Proposed answer: <*answer*>
Your previous reasoning was: <*response*>

Validation Task

1. Check whether the provided answer is logically correct based on the table data.
2. If it is incorrect or incomplete, identify the error.
3. Provide the corrected answer if needed, using **strict JSON format**.
4. If the original answer is correct, return **"VALID"** (as a JSON string).

CRITICAL REQUIREMENTS

- The response **MUST BE STRICTLY IN JSON FORMAT** with **NO explanations, no additional text, and no reasoning**.
- **DO NOT** include **"Reasoning"** or **"Validation"** steps in the output.
- **ONLY return one of the following:**
  - **"VALID"** (as a JSON string)
  - A corrected JSON answer in the exact same format as the proposed answer.

IMPORTANT:

- Your step-by-step reasoning **MUST NOT** exceed 5 sentences.
- Your final JSON **MUST** be included within the response.
- If you fail to comply, your response will be discarded as invalid.

Example Outputs

If the answer is correct: "VALID"
If the answer needs correction: {"answer": 42}

DO NOT RETURN outputs like:

- "The answer is correct. Here is my reasoning..."
- "After analysis, I found a mistake. The correct answer is: ..."

Now, validate the answer and provide the response in the correct format.