Reasoning through Code: Question Answering on Spanish Tabular Data using LLMs

Adarsh Prakash Vemali[1,*,†], R Raghav[2,*,†]

[1]University of California, San Diego
[2]Independent Researcher

Abstract

Tabular Question Answering (QA) plays a key role in enabling structured data understanding. This paper presents our system for the IberLEF 2025 PRESTA Task on Question Answering over Spanish Tabular Data. We target the DataBenchSPA QA task using a code-generation based strategy with Large Language Models (LLMs) in a zero-shot setting. Our approach formulates QA as a Python code generation task, leveraging LLMs to produce executable Pandas code that queries the relevant tabular data. We adopt a unified LLM model that jointly performs reasoning and code generation. We optimize prompt design by introducing schema compression techniques such as column aliasing and sampling representative rows to reduce context size. Our execution-aware retry mechanism improves output correctness by iteratively refining erroneous code based on runtime feedback. Our system achieved an accuracy of 73.0% on the blind test set, ranking 6th overall. These results highlight the feasibility of efficient and accurate tabular QA.

Keywords

Tabular Question Answering, Large Language Models, Code Generation, Prompt Engineering, Zero-Shot Prompting

1. Introduction

Tabular Question Answering (QA) is a pivotal area in Natural Language Processing (NLP), enabling automated information extraction and enhancing accessibility to structured datasets. The IberLEF 2025 [1] task, PRESTA: Question Answering over Tabular Data in Spanish / Preguntas y Respuestas sobre Tablas en Español [2], advances this field through the introduction of DataBenchSPA [3] - the first large-scale, Spanish-language benchmark comprising diverse tabular datasets across multiple domains. This task requires systems to accurately answer natural language questions over tables, with expected answer types including boolean values, categories, numbers, or lists of these.

Our system adopts a code-generation based methodology, employing Python and Pandas in combination with open-source Large Language Models (LLMs) to produce executable code for answering questions. Given a question and its associated table, the system generates code that loads the data, performs computations, and outputs the answer. This strategy enhances interpretability by encoding the reasoning process directly into code and remains agnostic to specific table schemas. Importantly, we optimize for a zero-shot setting to minimize the amount of table data passed to the LLM, ensuring low resource usage and scalability.

Our previous work [4] evaluated both agentic and unified LLM approaches, with the unified pipeline proving more effective in maintaining consistency and reducing errors. Accordingly, we adopt the unified approach for this Spanish-language task, leveraging prompt engineering and input minimization - via row sampling and column aliasing - to enhance performance and efficiency.

Since our method relies on code execution, one core challenge was ensuring the syntactic and semantic correctness of generated code. We addressed this by implementing iterative retry mechanisms that feed back error messages to the LLM for refinement. These retries proved effective in producing robust, executable code.

IberLEF 2025, September 2025, Zaragoza, Spain
*Corresponding authors.
†These authors contributed equally.
vemali.adarsh@gmail.com (A. P. Vemali); rraghav5600@gmail.com (R. Raghav)

Dataset	Accuracy	General
DatabenchSPA (Test)	73%	6^{th}
DatabenchSPA (Validation)	75%	–

Table 1
Performance on the DataBench QA and DataBench Lite QA subtasks, showing accuracy and rank among all systems (General), open-sourced models, and small open-sourced models ($\leq 8B$ parameters).

Our system achieved an accuracy of 73% on the test set and ranked 6^{th} overall, across both proprietary and open-source models, as shown in Table 1. Phi-4, a Small Language Model (under $8B$ parameters) proved to be our most effective system for the task. Key takeaways from our participation include the significant impact of prompt engineering in guiding LLM reasoning and the feasibility of high performance even with limited table input. Our approach and code is publicly available[1]. This is the same codebase used in our earlier English-language SemEval task [4], and it works out of the box for this Spanish variant with minor modifications.

2. Background

Our work builds upon the dataset collection originally introduced in [3]. As summarized in Table 2, the training set comprises 6 distinct datasets containing a total of 150 questions. The development set includes 4 datasets with 100 questions, while the blind test set consists of 10 datasets and 100 questions specifically designed to assess LLM performance on question answering over structured, real-world Spanish tabular data.

Dataset Split	Number of Datasets	Number of Questions
Training Set	6	150
Development Set	4	100
Blind Test Set	10	100

Table 2
Summary of the dataset splits used in our work, originally presented in [3].

2.1. Dataset Details

The entire dataset collection comprises a total of 31,318 rows and 1,740 columns, spanning a wide variety of day-to-day domains. Across all datasets, 250 questions were designed to evaluate question answering performance. For both the training and development sets, each dataset contains a consistent 25 questions, whereas the test set includes 10 questions per dataset, as shown in Table 3. An overview of the datasets used is presented in Table 4.

Answer Type	Sample	Number of Questions		
		Train	Dev	Test
Boolean	True/False	30	19	20
Number	4, 10	40	22	20
Category	Automotive, United States	28	22	20
List[category]	[apple, mango]	25	18	20
List[number]	[2, 4, 6, 8, 10]	26	18	20

Table 3
Distribution of Answer Types in train, development and test data

[1] https://github.com/Adarsh-Vemali/LLM-Driven-Code-Generation-for-Zero-Shot-Question-Answering-on-Tabular-Data

#	Name	Rows	Columns	#QA	Source (Reference)
1	Encuesta de Igualidad	2000	105	25	40dB
2	Calidad del Sueño	2000	80	25	40dB
3	Fusión Barómetros	7430	161	25	CIS
4	Barómetro Andaluz	5349	85	25	CEA
5	Juventud	1510	236	25	CRS
6	Política Fiscal	3011	198	25	CIS
7	Relaciones	2491	186	25	CIS
8	Barómetro Mensual	2444	185	25	CIS
9	Percepción del Amor	2000	150	25	40dB
10	Salud Mental	3083	354	25	CIS
	Total	**31318**	**1740**	**250**	

Table 4
Summary of datasets with QA counts, sources and

2.2. Related Work

LLMs have significantly transformed code generation, especially through the incorporation of zero-shot reasoning and structured prompting strategies such as Chain-of-Thought (CoT) prompting [5]. CoT enables models to break down tasks into intermediate reasoning steps, which is critical for complex logical tasks. However, in code generation scenarios, particularly those involving syntactic correctness and structural precision, CoT alone often proves insufficient. Approaches like CodeCoT [6] mitigate these shortcomings by leveraging self-examination mechanisms that iteratively refine outputs based on execution feedback.

In contrast to agentic CoT systems, which rely on iterative, autonomous execution of subtasks [7], structured prompting techniques like SCoT [8] incorporate programmatic structures (e.g., sequences, branches, and loops) directly into the reasoning process. These structured methods provide more deterministic and syntactically aligned outputs compared to agentic retries, which often incur higher computational cost without guaranteed improvements in correctness.

Despite these advances in general-purpose code generation, applying LLMs effectively to code generation over tabular datasets - especially in languages such as Spanish - remains underexplored. Recent efforts like MarIA [9] introduce high-quality Spanish language models that support downstream tasks like Spanish QA. These models, when paired with tabular QA tasks, present a unique challenge due to the complexity of schema representations and the variability of column labels.

One promising direction in this space is the use of summarization techniques for label compression. Tools like AutoDDG [10] demonstrate that LLMs can generate semantic summaries of column metadata, enabling more human-readable, compressed, and consistent schema representations. This not only facilitates better generalization across datasets but also supports unified modeling strategies that reduce dependency on handcrafted agentic pipelines. By simplifying schema understanding through summarization, models are better positioned to align input representations with reasoning steps needed for accurate QA and code generation.

Recent studies [11, 12, 13, 14] have delved into fine-grained, task-specific adaptations of LLMs across a range of domains, underscoring the value of domain-specific prompting and structured reasoning techniques. Likewise, generative methods have been extended to structured tasks such as sentiment analysis [15], showcasing the versatility of structured generation beyond conventional language tasks.

Moreover, retrieval-augmented approaches [16, 17] and extensive instructive fine-tuning [18] present viable avenues for strengthening reasoning reliability and generalization in tabular data contexts.

In summary, while agentic CoT pipelines provide flexibility, our work argues for a shift towards unified, summarization-augmented modeling frameworks that leverage structured CoT and semantic compression of tabular schemas. This direction is particularly beneficial for under-resourced languages like Spanish, where task adaptation, schema abstraction, and robust reasoning are essential for scalable and accurate code generation in QA systems.

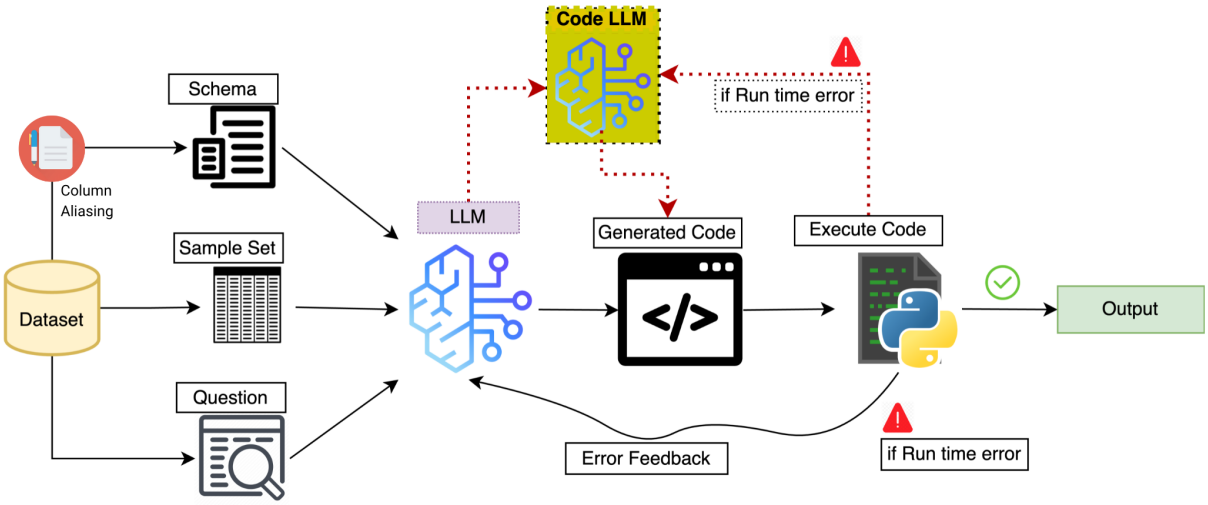

Figure 1: Flowchart illustrating the data preprocessing and model workflow using a unified and an agentic CoT approach. In the CoT setup, a reasoner LLM plans the solution steps, and a code LLM converts those steps into executable code. If the execution fails, the resulting errors are routed back to both models, enabling a feedback loop for correction. This flow is visually represented in the figure with red dashed arrows. For this task, we only experiment with the unified approach, where we have a single LLM doing both reasoning and code generation. Additionally, due to long column names in some datasets, we use the same LLM to summarize the column names to help fit our prompts within the LLM's context window.

3. System Overview

As illustrated in Figure 1, our system reframes the tabular QA task as a code generation problem in a zero-shot setting. Given a tabular dataset D and a natural language question Q, the system operates through a structured pipeline designed to maximize the effectiveness of LLMs. Our approach closely follows the methodology outlined in [4], with minor adaptations. First, it extracts the dataset schema (which includes the column names and data types), and a sample of the first η rows (with $\eta = 3$ yielding the best results; see Table 5). This is then used to construct a structured prompt in the system-user-assistant format, providing the model with essential context while minimizing unnecessary verbosity.

One challenge we encountered was that some datasets contained very long column names. Including these in the prompt often pushed the input size beyond the LLM's token limit of 16,384 tokens. To address this, we used the same LLM to first automatically generate concise aliases for the column names, effectively summarizing them while preserving their semantics. These shorter names were then substituted throughout the schema and dataset before constructing the final prompt. Once the prompt is prepared, the LLM then generates Python code intended to load the dataset and compute the answer to the question. The generated code is then parsed, checked for errors, and executed to produce the final output.

We introduce a robust retry strategy that takes execution outcomes into account. When the generated code fails, the resulting error message is returned to the language model, enabling it to revise and improve the code. This process can repeat up to three times, leading to more reliable outputs and a notable decrease in execution errors.

By minimizing the input size, we significantly improve computational efficiency and scalability, while also enabling the language model to better contextualize relevant information. Additionally, the introduction of column name aliasing represents a targeted adaptation of our earlier methodology, further enhancing the system's flexibility and robustness.

3.1. Agentic CoT Approach

In our previous work [4], we implemented an Agentic Chain-of-Thought (CoT) approach that separated reasoning and code generation across two specialized LLMs. While this method offered explicit reasoning traces and modularity, it suffered from frequent mismatches between the reasoning output and the code execution logic. For instance, LLaMA often produced incomplete or imprecise reasoning steps, which CodeLLaMA or Phi-4 then misinterpreted, leading to cascading execution failures. These issues, compounded by the limitations of smaller model sizes due to resource constraints, ultimately hindered performance. Motivated by these limitations, we adopt a unified LLM architecture for this task, enabling seamless integration of reasoning and code synthesis within a single model. This approach reduces misalignment and enhances overall robustness.

3.2. Unified LLM Approach

Building on [4], which showed that a unified LLM pipeline outperforms multi-step CoT approaches by enhancing consistency and reducing error propagation, we adopt a similar strategy in this study. To accommodate lengthy column names - we introduce an intermediate column aliasing step, replacing them with concise, semantically meaningful alternatives. Given the effectiveness of integrating reasoning and code generation into a single inference pass, we hypothesized that this unified approach would again yield strong performance. In this paper, we experimented with a range of LLMs, including LLaMA, CodeLLaMA, and Phi-4, to evaluate their ability to handle diverse question types and table structures within this framework.

3.3. Challenges and Solutions

We encountered several challenges in this task:

- **Long Column Names:** Some datasets included excessively long column names, which exceeded the LLM's token limit of 16,384 tokens when including the dataset schema in the LLM prompt. For column names that were more than 50 characters long, we used the LLM itself to generate concise aliases that preserved the semantics of the original names. Refer to Appendix subsection A.2 for examples.
- **Table Reasoning:** Following prior results, we pass only the schema and first three rows of the dataframe to help the model reason over tabular inputs without exceeding token limits.
- **Output Formatting:** We enforce strict formatting constraints in the prompt to ensure that generated outputs match expected types and structures.
- **Code Reliability:** We reuse the error feedback loop from our previous work, limiting to three retries. As before, this balances correction and efficiency, with negligible gains beyond three attempts.
- **Prompt Strategy:** We retain the System-User-Assistant template introduced earlier, which continues to perform well with the Spanish variant of the task. A detailed description of our prompt template can be found in the Appendix subsection A.1

4. Experimental Setup

Following our previous work [4], we adopt a zero-shot prompting strategy using quantized large language models (LLMs), with a strong emphasis on prompt engineering to enhance performance. All interactions are framed using a structured System–User–Assistant format, which has consistently yielded reliable results. For efficient inference with minimal performance trade-offs, we leverage dynamically quantized 4-bit models from Unsloth [19]. Specifically, we experiment with Meta LLaMA

3.1 (8B Instruct)[2], CodeLLaMA (7B)[3], and Microsoft Phi-4 (8.48B)[4]. As a pre-processing step, we use the respective LLM to alias columns exceeding 50 characters, replacing them with shorter, semantically meaningful names before generating predictions on the transformed dataset. All models were run using a single NVIDIA T4 GPU on Google Colab, demonstrating that strong performance can be achieved even under constrained computational budgets.

4.1. Evaluation Function

We follow the official evaluation setup provided by the organizers, based on the databench_eval[5] package. The evaluation function has been adapted to allow for minor formatting differences, making it more forgiving of small variations in output. It uses flexible matching strategies depending on the data type, including tolerant comparisons for booleans, numbers, and categorical outputs.

The evaluation function applies type-specific heuristics to allow for flexible matching. Boolean values are accepted in multiple valid forms (e.g., "true/false," "yes/no"), categorical outputs use string comparison, and dates are parsed for equivalence. Numerical answers are rounded to two decimal places, and lists are compared as sets to tolerate reordering.

5. Results

Model	η	DataBenchSPA (Validation)	DataBenchSPA (Test)
LLaMA	**1**	**51%**	**51%**
	3	46%	47%
CodeLLaMA	1	62%	60%
	3	**65%**	**62%**
Phi-4	1	71%	69%
	3	**75%**	**73%**

Table 5
Ablation study on the validation dataset to decide on the ideal number of rows to be provided to the model. η is the number of rows chosen from the dataset which is sampled and provided to the model

Our system achieved strong results, ranking 6^{th} in the General category (see Table 1). Improvements in prompt design and the integration of execution-aware retry strategies further enhanced both accuracy and efficiency. These findings reinforce the suitability of our approach for real-world tabular QA scenarios. Comprehensive results and ablation analyses are provided in Table 5.

We found that Phi-4 achieved the best performance when provided with three example rows ($\eta = 3$). Consequently, we used Phi-4 for our final test predictions, making our system a lightweight solution based on a Small Language Model (under $8B$ parameters).

5.1. Key Findings

Our key findings through our ablations (Table 5) include:

- As our system achieved similar results similar to [4], incorporating column aliasing did not degrade the system's performance.
- Our system is highly adaptable to diverse datasets across different languages, implying robust performance across a wide range of tabular QA tasks.
- Execution-aware retry mechanisms improved answer correctness with minimal additional cost, effectively resolving common syntactic and formatting errors.

[2] https://huggingface.co/unsloth/Meta-Llama-3.1-8B-Instruct-unsloth-bnb-4bit
[3] https://huggingface.co/unsloth/codellama-7b-bnb-4bit
[4] https://huggingface.co/unsloth/phi-4-unsloth-bnb-4bit
[5] https://github.com/jorses/databench_eval

6. Conclusion

Our system demonstrates the viability of LLM-driven code generation for zero-shot question answering over tabular data in Spanish. Building on our prior work [4], we reaffirm the effectiveness of a unified LLM pipeline in maintaining logical coherence and minimizing error propagation. Prompt engineering remains central to guiding model behavior, while input minimization strategies - such as sampling representative rows and aliasing long column names proved effective for improving contextual relevance and computational efficiency.

While our current setup showcases the strength of LLMs in multilingual tabular QA, there remains significant untapped potential in agentic approaches. Future work will focus on refining the agentic framework to better accommodate multilingual reasoning, complex transformations, and more sophisticated interactions with structured data.

Declaration on Generative AI

During the preparation of this work, ChatGPT was used to provide suggestions for academic writing. The authors subsequently reviewed and edited the content as necessary and take full responsibility for the final version of this publication.

References

[1] J. Á. González-Barba, L. Chiruzzo, S. M. Jiménez-Zafra, Overview of IberLEF 2025: Natural Language Processing Challenges for Spanish and other Iberian Languages, in: Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2025), co-located with the 41st Conference of the Spanish Society for Natural Language Processing (SEPLN 2025), CEUR-WS. org, 2025.

[2] J. Osés-Grijalba, L. A. Ureña-López, E. M. Cámara, J. Camacho-Collados, Overview of PRESTA at IberLEF 2025: Question Answering Over Tabular Data In Spanish, in: Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2025), co-located with the 41st Conference of the Spanish Society for Natural Language Processing (SEPLN 2025), CEUR-WS. org, 2025.

[3] J. O. Grijalba, L. A. U. López, J. Camacho-Collados, E. M. Cámara, Towards quality benchmarking in question answering over tabular data in spanish, Proces. del Leng. Natural 73 (2024) 283–296. URL: http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6617.

[4] R. Raghav, A. P. Vemali, D. Aswal, R. Ramesh, A. Bhupal, Scottyposeidon at semeval-2025 task 8: Llm-driven code generation for zero-shot question answering on tabular data, in: Proceedings of the 19th International Workshop on Semantic Evaluation, SemEval 2025, Vienna, Austria, 2025.

[5] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, Advances in neural information processing systems 35 (2022) 22199–22213.

[6] D. Huang, Q. Bu, Y. Qing, H. Cui, Codecot: Tackling code syntax errors in cot reasoning for code generation, arXiv preprint arXiv:2308.08784 (2023).

[7] J. Tang, T. Fan, C. Huang, Autoagent: A fully-automated and zero-code framework for llm agents, arXiv e-prints (2025) arXiv–2502.

[8] J. Li, G. Li, Y. Li, Z. Jin, Structured chain-of-thought prompting for code generation, ACM Transactions on Software Engineering and Methodology 34 (2025) 1–23.

[9] A. Gutiérrez-Fandiño, J. Armengol-Estapé, M. Pàmies, J. Llop-Palao, J. Silveira-Ocampo, C. P. Carrino, A. Gonzalez-Agirre, C. Armentano-Oller, C. Rodriguez-Penagos, M. Villegas, Maria: Spanish language models, arXiv preprint arXiv:2107.07253 (2021).

[10] H. Zhang, Y. Liu, A. Santos, J. Freire, et al., Autoddg: Automated dataset description generation using large language models, arXiv preprint arXiv:2502.01050 (2025).

[11] A. Mullick, A. Nandy, M. N. Kapadnis, S. Patnaik, R. Raghav, Fine-grained intent classification in the legal domain, arXiv preprint arXiv:2205.03509 (2022).

[12] A. Mullick, A. Nandy, M. Kapadnis, S. Patnaik, R. Raghav, R. Kar, An evaluation framework for legal document summarization, in: N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, J. Odijk, S. Piperidis (Eds.), Proceedings of the Thirteenth Language Resources and Evaluation Conference, European Language Resources Association, Marseille, France, 2022, pp. 4747–4753. URL: https://aclanthology.org/2022.lrec-1.508/.

[13] A. Mullick, I. Mondal, S. Ray, R. Raghav, G. Chaitanya, P. Goyal, Intent identification and entity extraction for healthcare queries in Indic languages, in: A. Vlachos, I. Augenstein (Eds.), Findings of the Association for Computational Linguistics: EACL 2023, Association for Computational Linguistics, Dubrovnik, Croatia, 2023, pp. 1870–1881. URL: https://aclanthology.org/2023.findings-eacl.140/. doi:10.18653/v1/2023.findings-eacl.140.

[14] R. Raghav, J. Rauchwerk, P. Rajwade, T. Gummadi, E. Nyberg, T. Mitamura, Biomedical question answering with transformer ensembles, in: CLEF (Working Notes), 2023.

[15] R. Raghav, A. Vemali, R. Mukherjee, Etms@ iitkgp at semeval-2022 task 10: Structured sentiment analysis using a generative approach, in: Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022), 2022, pp. 1373–1381.

[16] P. Carragher, A. Jha, R. Raghav, K. M. Carley, Quantifying memorization and retriever performance in retrieval-augmented vision-language models, arXiv preprint arXiv:2502.13836 (2025).

[17] P. Carragher, N. Rao, A. Jha, R. Raghav, K. M. Carley, Segsub: Evaluating robustness to knowledge conflicts and hallucinations in vision-language models, arXiv preprint arXiv:2502.14908 (2025).

[18] R. Raghav, A. P. Vemali, D. Aswal, R. Ramesh, P. Tusham, P. Rishi, Tartantritons at semeval-2025 task 10: Multilingual hierarchical entity classification and narrative reasoning using instruct-tuned llms, in: Proceedings of the 19th International Workshop on Semantic Evaluation, SemEval 2025, Vienna, Austria, 2025.

[19] M. H. Daniel Han, U. team, Unsloth, 2023. URL: http://github.com/unslothai/unsloth.

A. Appendix

A.1. Prompt Template

We use a system–user–assistant chat-based prompting framework. For this task, we design two separate prompt templates: one for column aliasing and another for code generation to answer questions over tabular data. Both templates are provided below.

A.1.1. Column Aliasing Prompt Template

System

```
You are a helpful assistant who understands Spanish and can shorten long column
names in tabular datasets. Your goal is to generate concise aliases (in the same
language as the original column) for column names that are longer than 50
characters. The alias should preserve the original meaning while being as short
and clear as possible
```

User

```
Generate the alias for the following column name -
Column name: <original column name from the dataset>

Output Format:
Return a JSON array, where each output is structured as follows:
{
    "original_name": <original column name from the dataset>,
    "aliased_name": "your_answer_here",
}
The value to "aliased_name" key in the JSON should be the aliased column name for
the original column
```

A.1.2. Question Answering Prompt Template

System

```
You are an expert Python data engineer.
Your task is to generate pandas code based on a structured reasoning process
You only generate code, no references or explanation - just code
You generate only 20 lines of code at max
```

User

```
Dataframe Schema:
<The schema of the dataset which would contain the column name and its data type>


Sample Rows:
<The first 3 rows of the dataset serialized into a list of dictionaries, where
each dictionary represents a row with column names as keys>

User Question:
<The question that is asked about the dataset>

Expected Output Format:
Generate runnable Python code that follows the given reasoning using pandas.
The code should assume that the dataframe is already loaded as `df`.
The final output should be stored in a variable named `result`.
```

```
The expected answer type is unknown, but it will always be one of the following:
    * Boolean: True/False, "Y"/"N", "Yes"/"No" (case insensitive).
    * Category: A value from a cell (or substring of a cell) in the dataset.
    * Number: A numerical value from a cell or a computed statistic.
    * List[category]: A list of categories (unique or repeated based on context).
    Format: ['cat', 'dog'].
    * List[number]: A list of numbers.

Given the user question, you need to write code in pandas, assume that you already
have df.
Generate only the code.
```

The assistant prompt was left blank during inference. The code obtained from the model would then be run on the dataset to evaluate the answer to the question.

A.2. Column Aliasing

Across the 10 test datasets provided for the task, 765 out of 1,740 total columns had names longer than 50 characters. All of these lengthy column names were aliased using the LLM. The table below presents examples of column names that were aliased in the test set

Dataset	Original Column Name	Aliased Column Name
ES_01_40db_Igualdad	Imagina una pareja formada por un hombre y una mujer que tuvieran exactamente las mismas condiciones laborales. En caso de que fuera necesario que una de las dos personas dejase de trabajar para cuidar de familiares dependientes (hijos/as, padres, madre)	Rol_cuidados_pareja_igual_laboral
ES_02_40dB_Dormir	Si mañana se celebraran unas nuevas elecciones generales, ¿cuál sería la probabilidad de que acudieras a votar? Utiliza una escala de 0 a 10, en la que '0' representa 'con toda seguridad, no iría a votar' y '10' 'con toda seguridad, sí iría a votar'	Prob_Votar_0a10
ES_03_CIS_Enero_Marzo_2023	Motivos entrevista incorrecta. Negativa que ofrece dudas, no coincide la dirección ni datos personales. Posibilidad de error en el teléfono	Motivo_duda_datos_contacto
ES_04_CEA_Barometro_Andaluz_Septiembre_2023	Del siguiente conjunto de medidas relacionadas con la gestión del agua, ¿cuál considera usted que sería la más adecuada para Andalucía? Dos opciones de respuesta_-Construir más desaladoras para potabilizar el agua	Medida_gestion_agua_deseada_-desaladoras

Table 6
Examples of column aliasing across datasets using LLMs