Adapt LLM for Multi-turn Reasoning QA using Tidy Data

Jan Strich
Language Technology Group
Universität Hamburg
Germany

Abstract

This paper presents our submission to the FinDBQA shared task at the 9th FinNLP workshop. The task involves answering finance-focused questions in a multi-turn environment, requiring step-by-step reasoning and Python code generation. We propose a novel approach to tackling this multidimensional problem by preprocessing the data into tidy format so that each column represents a variable and each row an observation. Our experiments demonstrate that using the tidy data format allows all models to surpass SOTA, with GPT-4o achieving a 50.62% accuracy on the DBQR-QA benchmark, placing it second in the Shared Task Leaderboard. These findings suggest that transforming data into the tidy data format enhances reasoning capabilities, reduces syntax errors, and improves performance on table-reasoning QA tasks. The code is available online[1].

1 Introduction

When analyzing data in finance, the ability to write code that can answer complex, multi-step questions is crucial. These questions often require reasoning across multiple data sources and steps. As financial data becomes increasingly intricate, this ability to reason and generate code is vital for deriving insights and making informed decisions.

Recent advancements in NLP, particularly with the rise of large language models (LLMs), are crucial for tackling the multi-step reasoning tasks commonly encountered in finance. Early research in question answering (QA) focused on zero-shot tasks using general benchmarks (Rajpurkar et al., 2016; Chen et al., 2021a). However, more recent work has shifted towards more complex challenges, such as multi-hop reasoning (Chen et al., 2021b) and reasoning over tabular data (Zhang et al., 2020; Pal et al., 2023), including hybrid approaches that combine it with text (Chen et al., 2022).

[1] https://github.com/pesc101/dbqr

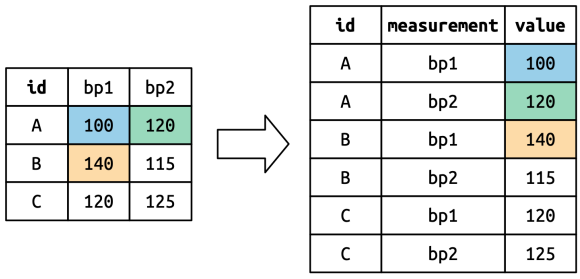

Figure 1: Transform table into the tidy data format (Wickham et al., 2023).

This paper explores approaches for the FinDBQA shared task (Nararatwong et al., 2024), which involves answering finance-focused questions in a multi-turn conversation using a data chunk of a graph database. We approach the task using three approaches testing them on four models from two model families (Llama 3.1 & GPT-4o). We generate Python code using the pandas package. In addition to the question and the variables, custom Python functions are integrated, and the table information from the data chunks is passed to the model. Our main assumption is that models will generate code more accurately and reason better by converting tables into tidy data format (Wickham, 2014), as shown in Figure 1. Therefore, we compare three approaches: one using the raw data, another using a tidy-data format, and a third that extends the tidy-data approach by adding few-shot examples. The contributions of the paper are:

- **Testing Prompt Templates:** Robust prompt templates are essential for generating Python code with reasoning steps, clarity, error handling, and the correct output format.
- **Testing Tidy Data approach:** Tidy data provides a clear, repeatable, and reliable format, making it easier for models to reason through steps and generate accurate code.

Proceedings of the Joint Workshop of the 9th FinNLP, the 6th FNP, and the 1st LLMFinLegal, pages 392–400
January 19–20, 2025.

2 Related Work

Recent advancements in reasoning-based QA have been fueled by key datasets like MMLU (Hendrycks et al., 2021) and GSM-8K (Cobbe et al., 2021). In tabular QA, which requires complex reasoning, notable contributions include TAT-QA (Zhu et al., 2021) and FinQA (Chen et al., 2021b), which focus on hybrid financial tabular and textual data, and numerical reasoning in finance. ConvFinQA (Chen et al., 2022) introduces multi-hop conversational QA using a single table, while FeTaQA (Nan et al., 2021) expands to free-form table QA.

Tidy data (Wickham, 2014), provides a structured framework for organizing datasets to facilitate manipulation, modeling, and visualization that is used widely in data science and statistical computing. In tidy datasets, each variable is a column, each observation is a row, and each type of observational unit forms a table. This structure simplifies analysis, reduces errors, and ensures consistency across workflows.

3 DBQR-QA Benchmark

The shared task FIN-DBQA involves answering finance-focused questions in a multi-turn conversation using a data chunk of a graph database. The DBQR-QA benchmark is used for this purpose. However, in this task, only the reasoning component is tested, as the data chunks of the graph database are already given. The reasoning step involves writing a Python program (with pandas) that includes logical steps and mathematical calculations. Figure 3 in Appendix A shows an example question from the practice dataset. Each dataset contains a question, a set of variables, the queries used to fetch the data from the graph database, and a pickle file containing the data chunks.

Dataset Details The dataset is divided into five categories, each targeting specific aspects of reasoning within financial datasets: **Simple:** Basic queries, such as finding the best year to exclude to maximize metrics for a specific company. **Complex:** No explicit company names; involves thresholds and optimizing parameters across companies. **Multi-Table:** Queries span multiple tables, requiring data integration and comparative analysis. **Multi-Hop:** Multi-step reasoning, analyzing trends or comparisons across industries and periods. **Instruction:** Real-world queries, guiding models through multi-step, multi-dimensional analysis.

Figure 1 shows the distribution of the categories based on the dataset splits. For the categories Simple, Complex, and Multi-Table, the benchmark consists of ten conversations, and Multi-Hop and Instruction consists of five. Each conversation is a collection of ten questions built up on each other.

Evaluation Metric The dataset is evaluated using three metrics (Nararatwong et al., 2024). These include a custom heuristic evaluator, a custom GPT and human score. In our work, we focus solely on the heuristic evaluator, which operates as follows: The prediction must match the label in these ways: for a single numeric label (excluding years), the prediction should be a number with two decimal places. For a set of numbers, the prediction must have the same number of values, each matching the label. For years, all elements in the label must appear in the prediction.

	Practice	Train	Test
Simple	10	50	40
Complex	10	50	40
Multi-table	10	50	40
Multi-hop	10	30	10
Instruction	10	20	20
Total	50	200	150

Table 1: Distribution of samples per dataset category.

4 Methodology

Section 4.1 outlines the prompt templates, Section 4.2 covers converting data chunks into tidy data, and Section 4.3 details few-shot example creation. Figure 2 summarizes the three tested approaches.

We provided the information displayed for each approach and the models conducted each conversation within a multi-turn environment. After each conversation, the generated code was executed in a sandboxed Python environment with the Pandas package installed. The custom function was imported, and all attributes were stored in `globals()` to allow the reuse of variables during the conversation. The value stored in the RESULT var is then evaluated using the heuristic evaluator.

4.1 Prompt Templates

Figure 4 in Appendix B presents the system prompt used for each approach. The system prompt provides clear instructions on guidelines, provided information, behavior, and formatting. The models

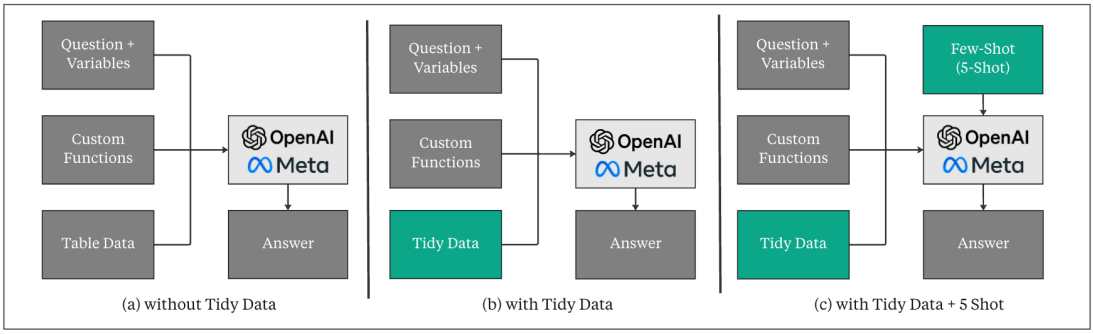

Figure 2: Overview of proposed approaches. (a): Prompt template using custom functions and data chunks as table data. (b): Transferring the table data to a tidy data format. (c): Adding additional few-shot examples showing how to reason step-by-step.

were prompted to first create a step-by-step plan (Wei et al., 2022) and then generate the Python program to solve the question. The model processes three components required to answer the questions: the question, the variables, and the tables, which are passed via the user prompt (Figure 5). Additionally, the system prompt mentions using two custom functions for formatting: `word_to_int(str)` for transferring numbers from variables in string format to integers and `reformat_result(Any)`. We also experimented with passing a list of custom functions to the model, but often the functions were not generic enough or the models had problems using them consistently over all data samples. This did not lead to any improvements.

4.2 Tidy Data

For approaches (b) and (c) in Figure 2, we implemented an algorithm to convert table data into tidy format. This format is achieved when each variable is represented as a column and each observation is a row. Therefore, the year and value columns were melted. Then, an algorithm identified column names for companies, concepts, persons, and industries. This involves matching column names using information from the Cypher query and general pattern matching, such as the regex 'usgaap:' to find the concept column.

4.3 Creation of Few-Shot Examples

As the final approach, the model was fed five static few-shot examples. These examples should guide the models on how reasoning steps should appear. For each of the five categories, a manually constructed example from a validated correct data sample from the practice dataset was added to the prompt. Figure 6 in Appendix B illustrates one of the five few-shot examples.

5 Evaluation

This section presents the results of the three proposed approaches, summarized in Table 2. Each approach was evaluated across dataset splits using four models: Llama 3.1 8B Instruct, Llama 3.1 70B Instruct with FB8, GPT-4o-mini, and GPT-4o (Details in Appendix C). These models were selected to represent a mix of widely used lightweight and large open and closed models. For each model, we used the same model parameter for reproducible results: `temperature: 0, max_tokens: 2000, top_p: 0.95`. Each run was evaluated using the heuristic evaluator as explained in Section 3.

5.1 Main Results

Table 2 shows the evaluation results of each approach. The tidy data approach with five few-shot examples outperforms the original paper, achieving SOTA test pass rates across all models. For LLama 3.1 8B and Lllama 3.1 70B, the pass percentage increases when using tidy data and is even better when five few-shot examples are used. Although the overall pass rate of 19% and 22% is still low, the crash rate has been significantly reduced. This pattern is particularly evident in the train and test splits. Interestingly, the practice split shows a higher pass rate without using tidy data, but the error rate is notably lower with tidy data.

For GPT-4o-mini, performance increased by using tidy data and adding few-shot examples. Interestingly, the crash rate was lower for both models when no few-shot examples were used. This was particularly noticeable in the practice split, where the error rate dropped to 0. For GPT-4o, the performance remains consistent across all three approaches. On average, the performance with tidy data is better than without and with few-shot exam-

Model	Overall			Practice (N=50)			Train (N=200)			Test (N=150)		
	Pass	Fail	Crash	Pass	Fail	Crash	Pass	Fail	Crash	Pass	Fail	Crash
GPT-4 (Baseline)	18.2	52.4	26.8	-	-	-	-	-	-	-	-	-
LLama 3.1 8B												
+ *w/o Tidy Data*	7.6	39.6	52.7	16.0	42.0	42.0	5.5	44.5	50.0	7.0	33.0	60.0
+ *Tidy Data*	12.1	55.2	32.6	4.0	74.0	22.0	14.5	62.0	23.5	11.0	40.7	49.3
+ *Tidy Data + FW*	**19.6**	60.7	**20.0**	4.0	74.0	22.0	18.0	61.0	21.0	27.0	56.0	18.0
LLama 3.1 70B												
+ *w/o Tidy Data*	12.3	35.8	52.0	34.0	28.0	38.0	7.0	33.5	59.5	12.0	41.3	46.7
+ *Tidy Data*	16.0	50.5	33.5	12.0	52.0	36.0	17.0	53.0	30.0	16.0	46.7	37.3
+ *Tidy Data + FW*	**22.1**	61.4	**16.5**	30.0	56.0	14.0	16.5	65.0	18.5	27.0	58.3	14.7
GPT-4o-mini												
+ *w/o Tidy Data*	31.6	47.8	20.6	34.0	46.0	20.0	26.5	52.0	21.5	37.0	44.7	19.3
+ *Tidy Data*	34.1	61.0	**4.9**	32.0	68.0	0.0	31.0	66.5	2.5	39.0	52.3	8.7
+ *Tidy Data + FW*	**39.4**	53.0	7.6	62.0	38.0	0.0	33.5	57.0	9.5	39.0	54.3	6.7
GPT-4o												
+ *w/o Tidy Data*	50.5	41.8	7.7	<u>66.0</u>	34.0	0.0	45.5	42.5	12.0	<u>52.0</u>	43.3	4.7
+ *Tidy Data*	**50.6**	45.9	**3.5**	<u>66.0</u>	34.0	0.0	<u>49.5</u>	46.0	4.5	47.0	49.7	3.3
+ *Tidy Data + FW*	48.9	45.4	5.7	58.0	42.0	0.0	<u>49.5</u>	44.0	6.5	45.0	48.3	6.7

Table 2: Performance comparison of models across Practice, Train, and Test splits. The evaluation is done with the heuristic evaluator. Values are all calculated in percentages. **Pass**: Result is equal to gold label. **Fail**: Result is unequal to gold label. **Crash**: Executed code crashed in execution. Baseline taken from Nararatwong et al. (2024). **Bold**: Best performance per model. <u>Underline</u>: Best overall performance per split. **FW**: Add 5 few-shot examples.

ples, but concerning the dataset splits, the results vary in an insignificant way. The best result on the test set was achieved without tidy data, but the lowest crash rate was observed with tidy data and without few-shot examples. Important to note here is, that results from OpenAI models can vary even with `temperature=0` caused by the closed API.

In addition the best runs were evaluated with GPT and human score (Appendix D), which is consistent with the results presented.

5.2 Takeaways

These results suggest that tidy data improves performance on the DBQR-QA, particularly with small or quantized models. Llama models make fewer syntax errors, generate more error-handling code, and answer a greater number of questions correctly. This improvement is primarily because the fundamental design of Pandas and most of its functions align with the tidy data format. Transforming data into this format reduces the complexity of the code the model needs to generate and enables more straightforward function calls to perform reasoning steps. However, despite these improvements, models still struggle with correctly identifying the necessary reasoning steps to solve problems.

The performance of the GPT family highlights the ability of LLMs to construct stable reasoning processes. However, the smaller improvement observed with GPT-4o suggests that larger models benefit less from tidy data compared to smaller ones. Additionally, while adding few-shot examples significantly benefits GPT-4o-mini, this approach shows diminishing returns with GPT-4o.

6 Conclusion

In this paper, we present a novel approach using tidy data to improve performance on the DBQR-QA benchmark. We were able to show that small and quantized models perform better on the benchmark and produce fewer syntax errors using tidy data. The best performance was achieved using GPT-4o with tidy data. The results demonstrate that tidy data has a notable impact, particularly for smaller models, by simplifying data input and enhancing the model's ability to perform reasoning. For larger models, it remains highly effective, significantly reducing the error rate.

References

Chung-Chi Chen, Hen-Hsen Huang, and Hsin-Hsi Chen. 2021a. NQuAD: 70,000+ Questions for Machine Comprehension of the Numerals in Text. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management*, pages 2925–2929, Virtual Event, Queensland, Australia. ACM.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021b. FinQA: A Dataset of Numerical Reasoning over Financial Data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance Question Answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022*, pages 6279–6292, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *CoRR*, abs/2110.14168. ArXiv: 2110.14168.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *9th International Conference on Learning Representations, ICLR 2021*, Virtual Event, Austria. OpenReview.net.

Linyong Nan, Chia-Hsuan Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryscinski, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Benjamin Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir R. Radev. 2021. FeTaQA: Free-form Table Question Answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Rungsiman Nararatwong, Chung-Chi Chen, Natthawut Kertkeidkachorn, Hiroya Takamura, and Ryutaro Ichise. 2024. DBQR-QA: A Question Answering Dataset on a Hybrid of Database Querying and Reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15169–15182, Bangkok, Thailand. Association for Computational Linguistics.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. MultiTabQA: Generating Tabular Answers for Multi-Table Question Answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334, Toronto, Canada. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, USA. The Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed 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*, volume 35, pages 24824–24837, New Orleans, Louisiana. Curran Associates, Inc.

Hadley Wickham. 2014. Tidy Data. *Journal of Statistical Software*, 59(10):1 – 23.

Hadley Wickham, Mine Çetinkaya Rundel, and Garrett Grolemund. 2023. *R for data science*. O'Reilly Media, Inc.

Shuo Zhang, Zhuyun Dai, Krisztian Balog, and Jamie Callan. 2020. Summarizing and Exploring Tabular Data in Conversational Search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, pages 1537–1540, New York, NY, USA. Association for Computing Machinery.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers)*, pages 3277–3287, Virtual Event. Association for Computational Linguistics.

A Dataset Example

```
## Parsed JSON Data
{
  "sectionID": 1,
  "sectionTitle": "simple", % Category
  "question":
  "Was Caterpillar's average total revenue
  higher or lower than Realogy's lowest net
  income from 2019 to 2021?",
  % Variables
  "vars": {
    "year_1": 2019,   % Start Year
    "year_2": 2021    % End Year
    "company_1": {
      "mention": "Caterpillar",
      "name": "CATERPILLAR INC"
    "concept_1": {
      "mention": "total revenue",
    },
  ...
  },
  "queries": {
   "var_q1_p1": "WITH [\"CATERPILLAR INC\"] AS companies, [\"us-gaap:Revenues\"] ...",
   "var_q1_p2": "WITH [\"Realogy Holdings Corp.\"] AS companies, [\"us-gaap:ProfitLoss\"] ..."
  }
  "answer": "higher"
}

## Parsed Pickle Data
{
'var_q1_p1': 2019 2020 2021
CATERPILLAR INC us-gaap:Revenues  5.380000e+10 4.174800e+10 5.097100e+10,
'var_q1_p2': 2019 2020 2021
Realogy Holdings Corp. us-gaap:ProfitLoss  185000000.0  356000000.0 350000000.0
}
```

Figure 3: Dataset sample from Practice split.

B Prompt Templates

Instructions:
You are a coding assistant tasked with providing a part of a Python script to solve the given question.

- **Guidelines**:
1. **Think step by step**: Break down the problem into logical steps before writing the Python code.
2. **Clarity**: Write clean, modular, and reusable code wherever possible, adhering to Python best practices.
3. **Edge Case Handling**: Handle potential edge cases, such as empty DataFrames, missing columns, NaN values, or division by zero.
4. **Commenting**: Include meaningful high-level comments for each step of the solution, summarizing the logic where needed.
5. **Error-Checking**: Ensure error-free code by validating inputs and providing meaningful fallbacks (e.g., handle missing rows gracefully).
6. **Answer Format**: Stick to the desired format and answer with one number or word (e.g. higher/lower, yes/no, ...) without repeat the question.
7. **Real Scenario**: Do not create mock data or create new functions, only use the provided df and vars.

- **Provided Information**:
- Variables: A pre-initialized dictionary "vars" contains all necessary static variables for solving the question. Do not reinitialize it.
- If any value in "vars" represents a number as a string (e.g., "two", "three"), the "words_to_int" function must be called to convert it into an integer before further processing.
- Always check for and handle such cases before using the variable in numerical operations.
- Table(s): Data is provided in one or more preloaded Pandas DataFrames ("df_0", "df_1", ..., "df_x"). Do not initialize or modify their structure directly.
- The column "'value'" always contains numerical values as floats unless otherwise specified.
- Always verify column names in DataFrames before using them to avoid KeyErrors.
- Be always aware of type of the DataFrames column when using them.

- **Formatting**:
- Use "format_result" as the final step to format the solution output. Assign it to "RESULT".
- Ensure all outputs conform to the required format and handle multiple result types (e.g., scalars, lists, or dicts).
- If you would return a DataFrame convert it to a dict or list with the most relevant information.
- If you calculate a differences always use the absolute value.

- **Behavior**:
- If data or inputs are invalid or missing, handle the situation gracefully, providing a fallback or skipping steps as appropriate.

Desired Format:
To solve the question we will follow these steps:
1.: First thinking step
2.: ...
Now, let's write the Python code to implement these steps.
[Python Code START]
Write the Python code to solve the problem here
Step 1: <Brief description of the first step>
<code>
Step 2: <Brief description of the second step>
<code>
...
# Final Step: Format the result
RESULT = format_result(<final_output>)
[Python Code END]

Figure 4: System prompt to answer the questions.

```
Question:
{{ user_prompt }}
Variables:
vars = {{ vars }}

{% for i in range(tablesllength) %}
Table (df_{{ i }}):
Table Head (df_{{ i }}):
{{ tables[i] }}
Table Information (df_{{ i }}):
{{ table_infos[i] }}
{% endfor %}
Answer:
```

Figure 5: User prompt template to dynamically add the context.

```
Example 1: Extracting the right year
To solve the question we will follow these steps:
1. Filter the DataFrame for rows where the company is vars['company_1']['name'] and the concept is vars['concept_1']['name'].
2. Check if the filtered DataFrame is empty. If it is empty, return a formatted result with None.
3. Calculate the total deferred revenues for all years by summing up the value column in the filtered DataFrame.
4. Initialize variables highest_average to 0 and year_to_exclude to None to keep track of the highest average and the year excluded during calculations.
5. Iterate through each row in the filtered DataFrame and calculate.
6. Final Step: Format the result with the year_to_exclude that maximized the average and return it.
Now, let's write the Python code to implement these steps.

[Python Code START]
```python
# Step 1: Filter the DataFrame for the specific company and concept
filtered_df = df_0[
(df_0['company'] == vars['company_1']['name']) &
(df_0['concept'] == vars['concept_1']['name'])
]

# Step 2: Check if the filtered DataFrame is not empty
if filtered_df.empty:
        RESULT = format_result(None)
else:
# Step 3: Calculate the total deferred revenues for all years
        total_revenue = filtered_df['value'].sum()

# Step 4: Initialize variables to track the highest average and the year to exclude
highest_average = 0
year_to_exclude = None

# Step 5: Iterate over each year, calculate the average excluding that year, and find the maximum
for index, row in filtered_df.iterrows():
# Calculate the sum excluding the current year
sum_excluding_year = total_revenue - row['value']

# Calculate the average excluding the current year
average_excluding_year = sum_excluding_year / (vars['num_1'] - 1)

# Check if this is the highest average found so far
if average_excluding_year > highest_average:
        highest_average = average_excluding_year
        year_to_exclude = row['year']

# Final Step: Format the result
RESULT = format_result(year_to_exclude)
```
[Python Code END]
```

Figure 6: One few-shot example added to the system prompt.

C Model Details

Table 3 presents the details of the models used for evaluation. All models are openly accessible, ensuring reproducibility of the results.

Category	Model Name	Model Weights
Hugging Face	meta-llama/Llama-3.1-8B-Instruct	8B
Hugging Face	neuralmagic/Meta-Llama-3.1-70B-Instruct-FP8	70B
Snapshot	gpt-4o-mini-2024-07-18	-
Snapshot	gpt-4o-2024-08-06	-

Table 3: Presentation of models used for the evaluation.

D Evaluation Results - GPT/ Human Score

In addition to the evaluation using the heuristic evaluator presented in the main results, two additional metrics were employed in the shared task to assess the outcomes. Specifically, GPT-4 served as an evaluator with a tailored prompt, while the best run from each dataset was also manually reviewed by human evaluators. Table D presents the results for the best run across each metric and dataset split.

The findings reveal a consistent alignment between the GPT and human evaluation scores with those of the heuristic evaluator. Notably, the GPT scores tend to be slightly lower, whereas human scores are slightly higher than the grader scores. This consistency highlights the robustness of the results, irrespective of the metric applied.

	Practice (N=50)	Train (N=200)	Test (N=150)
Grader Score	54.0	33.0	52.0
GPT Score	54.0	31.0	51.0
Human Score	**56.0**	**37.0**	**55.0**
Average	54.67	33.67	52.67

Table 4: Comparison of grader, GPT, and human scores across practice, train, and test datasets. **Bold**: Best performance per dataset split.)