Text2Analysis: A Benchmark of Table Question Answering with Advanced Data Analysis and Unclear Queries

Xinyi He[1]*, Mengyu Zhou[2]†, Xinrun Xu[3]*, Xiaojun Ma[2], Rui Ding[2], Lun Du[2], Yan Gao[2], Ran Jia[2], Xu Chen[2], Shi Han[2], Zejian Yuan[1], Dongmei Zhang[2]

[1] Xi'an Jiaotong University
[2] Microsoft
[3] Institute of Software Chinese Academy of Science
hxyhxy@stu.xjtu.edu.cn, xuxinrun20@mails.ucas.ac.cn, yuan.ze.jian@xjtu.edu.cn,
{mezho, xiaojunma, juding, lun.du, gaoya, raji, xu.chen, shihan,dongmeiz}@microsoft.com

Abstract

Tabular data analysis is crucial in various fields, and large language models show promise in this area. However, current research mostly focuses on rudimentary tasks like Text2SQL and TableQA, neglecting advanced analysis like forecasting and chart generation. To address this gap, we developed the Text2Analysis benchmark, incorporating advanced analysis tasks that go beyond the SQL-compatible operations and require more in-depth analysis. We also develop five innovative and effective annotation methods, harnessing the capabilities of large language models to enhance data quality and quantity. Additionally, we include unclear queries that resemble real-world user questions to test how well models can understand and tackle such challenges. Finally, we collect 2249 query-result pairs with 347 tables. We evaluate five state-of-the-art models using three different metrics and the results show that our benchmark presents introduces considerable challenge in the field of tabular data analysis, paving the way for more advanced research opportunities.

1 Introduction

Tabular data analysis plays a crucial role in various fields, and automated data analysis has the potential to enhance people's work efficiency significantly (Delen and Ram 2018a). The emergence of large language models has shown promising capabilities to accelerate tabular data analysis (Chen 2023; Ye et al. 2023; Ma et al. 2023; Jiang et al. 2023). Understanding the analytical abilities of these models, identifying the analysis processes they can replace, and determining the analysis steps they can assist with have become pressing questions in the field.

Existing research on tabular data analysis has limited coverage of data analysis. As shown in Figure 2, data analysis can be divided into descriptive, diagnostic, predictive, and prescriptive analytics (Delen and Ram 2018b). The existing Text2SQL and TableQA datasets (Dong and Lapata 2016; Katsogiannis-Meimarakis and Koutrika 2021) focus primarily on rudimentary operations that are part of descriptive analytics and can be mostly solved by SQL and OLAP operations. They pay limited attention to **advanced analysis** (see §2.1) that require advanced operations and visualizations beyond rudimentary operations, such as calculating insights, forecasting, and chart generation (see examples in Figure 1).

In the real world, many user queries are often described in unclear ways (Wang et al. 2023). When solving advanced or

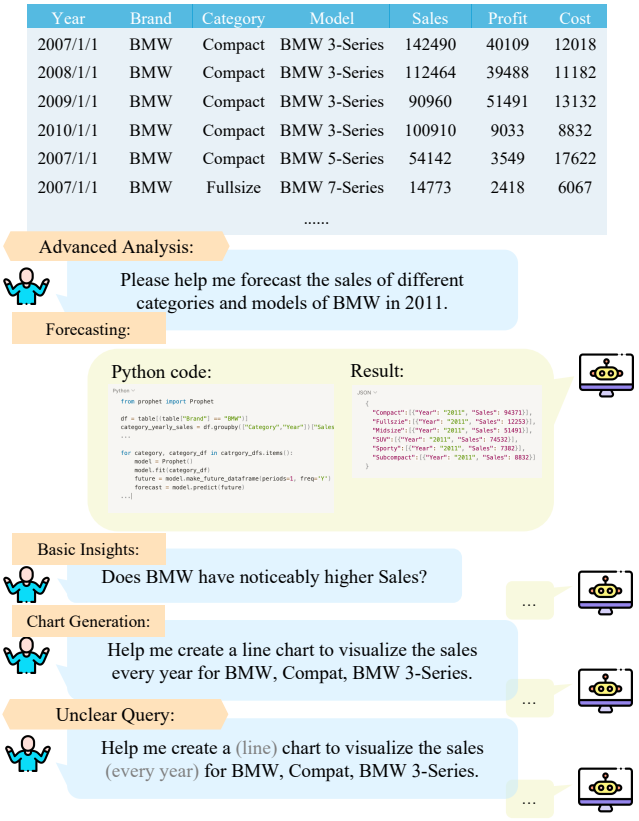

Figure 1: Examples of Text2Analysis Benchmark.

* The contributions by Xinyi He and Xinrun Xu have been conducted and completed during their internships at Microsoft.
† Corresponding author.

complex data analysis tasks with a large set of available tools and APIs, it is hardly the case that a user could write clear instructions with complete intent and parameters. As we will discuss in §2.2, the most common **"unclear query"** type is missing parameters for analysis tasks. *E.g.*, the query "Help me create a chart to visualize the sales for BMW, Compat, BMW 3-Serie" does not explicitly specify the chart type to be drawn or the field to be mapped to the x axis. Responding accurately to these queries not only demands the semantic parsing abilities of large language models but also requires them to possess strong data analysis capabilities to recommend intent beyond the query.

In this paper, we propose the **Text2Analysis** benchmark which expands beyond rudimentary operations and clear instructions. The benchmark incorporates unclear queries that involve advanced data analysis. Similar to Text2SQL datasets, in Text2Analysis the input is the (table, query) pair, and the output is the (code, result) pair. The ground-truth code only leverages a set of chosen data analysis APIs / operations from public and customized Python libraries such as Pandas, Prophet and Matplotlib.

Collecting the dataset is a difficult task because each sample in the Text2Aanlysis dataset simultaneously contains a table, query, Python code, and result. It requires annotators with related expert backgrounds and would consume a lot of time. To accelerate the annotation process and increase the volume of annotated data, we have developed five innovative and reliable annotation methods. Those methods make full use of large language models to perform forward annotations, expansion with new tables, and expansion unclear queries. Meanwhile, some methods also collect data from the output, such as reverse generation from codes or results. We collect 2249 (query, code, result) pairs with 347 tables. To ensure annotation quality, iterative annotation and human evaluation are employed. Their results and dataset distribution indicate that Text2Analysis has a diverse, high-quality data analysis dataset.

Due to the numerous tasks involved in the problem and the outputs consisting of both code and results, evaluating the generated solutions with appropriate metrics poses a challenge. We have selected three metrics to evaluate from different perspectives: executable code ratio, pass rate, and regression metrics. The executable code ratio evaluates the model's ability to generate executable code. Pass rate evaluates the correctness of the generated code. Regression scores measure the predicting capability of the chosen model within the generated code.

Furthermore, we provide an evaluation of five current state-of-the-art models, including GPT family models, code generation models, and tabular models. We evaluate their performance in handling advanced analysis and unclear queries. Our experiment indicates that large language models exhibit robust parsing and code generation aptitudes for data analysis in the context of clear queries. However, they grapple with complex libraries and unclear queries. To augment their efficacy, future research can concentrate on bolstering the capacity to recommend fields for sophisticated analyses and tackling complex operations such as operations with complex parameter input and model training.

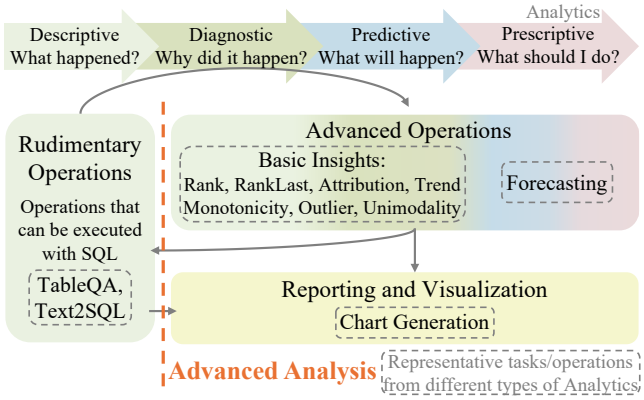

Figure 2: Advanced Analysis consists of Advanced Operations and Visualizations that are not covered by Rudimentary Operations across descriptive, diagnostic, predictive, and prescriptive analytics.

In summary, our main contributions are:

- We create the Text2Analysis benchmark which includes advanced analysis tasks and unclear queries that were rarely addressed in previous research work. The dataset and code will be open-sourced on https://github.com/microsoft/Text2Analysis.
- We propose five innovative and reliable annotation methods for the construction of NL2Code datasets. They utilize large language models to accelerate the annotation process and increase the volume of annotation.
- The performance of the baseline models is systematically evaluated against our Text2Analysis benchmark. Our experiments show the challenges to be solved in the future to satisfy real-world table analysis needs.

2 Problem Definition

We introduce the Text2Analysis problem as follows: $(table,\ query) \rightarrow (code,\ result)$. The input consists of a $table$ and a user $query$. The output consists of Python $code$ snippet(s) and the corresponding $result$(s). A $table$ has n fields $T = (f_1, ..., f_n)$, and each field consists of a field header and field values. A $query$ is related to data analysis, particularly focusing on advanced analysis (§2.1) that addresses the shortcomings of existing work and presents a greater and more difficult challenge for models. Additionally, it includes unclear queries (§2.2), which are often found in real-world user scenarios and can more effectively evaluate the model's analytical capabilities.

2.1 Analysis Operations and Tasks

Text2Analysis expands the data analysis dataset to advanced analysis tasks. As shown in Figure 2, Data analysis can be divided into descriptive (what happened?), diagnostic (why did it happen?), predictive(what will happen), and prescriptive analytics (what should I do?) (Delen and Ram 2018b). And reporting and visualization may follow each type of analytics. Existing research on table-based data analysis tasks,

such as TableQA and Text2SQL (Dong and Lapata 2016; Katsogiannis-Meimarakis and Koutrika 2021), has focused mainly on part of descriptive analytics that can be solved by SQL. They pay insufficient attention to advanced analysis that are beyond the rudimentary operations and require more in-depth analysis.

The advanced analysis portion of Text2Analysis selects representative tasks from each type of analytics to form the dataset. From descriptive and diagnostic analytics, basic insights are chosen. From predictive analytics, forecasting is selected. And from reporting and visualization, chart generation is chosen. A more detailed introduction to each task will be provided after the following paragraph.

Advanced analysis, along with rudimentary operations, form the Text2Analysis dataset. They can be combined to form a complex analysis. rudimentary operations and advanced operations (tasks in advanced analysis that output data such as tables and values, that is, tasks excluding reporting and visualization) can be interconnected, and reporting and visualization can be performed subsequently for display.

We introduce the involved tasks one by one as follows:

1. **Rudimentary Operations**: These operations encompass a set of functions and procedures that can be executed using the Structured Query Language (SQL) (Date 1989). Their primary purpose is to enable users to perform data management, manipulation, and transformation on multi-dimensional structured data. The main operations include group by, aggregation, filter, sort, and so on.

2. **Basic Insights**: In the context of a multi-dimensional dataset, an insight represents an interesting observation about a particular subject from a specific perspective (Ding et al. 2019; Ma et al. 2021; Chen, Yang, and Ribarsky 2009). Text2Analysis incorporates seven commonly insights:

- Rank: Within a group comprising multiple values, the highest value significantly exceeds all other values.
- RankLast: Within a group comprising multiple values, the lowest value is notably smaller than all other values.
- Attribution: In a group of multiple non-negative values, the highest value is equal to or larger than the sum of all other values.
- Trend: A time series (segment) exhibits an increasing or decreasing trend.
- Monotonicity: A time series (segment of) exhibits a consistent and unidirectional increasing or decreasing trend.
- Outlier: A time series contains outliers, which deviate significantly from the trend compared to the majority of points and their neighbors.
- Unimodality: A (segment of) time series exhibits an unimodal distribution, characterized by a single peak or turning point, and may display U-shaped patterns.

3. **Forecasting**: Forecasting involves predicting future trends and outcomes by analyzing historical data using statistical methods, machine learning algorithms, and time series models (Taylor and Letham 2018; Hosseini et al. 2021). This process identifies patterns and relationships within the data, enabling informed predictions about future events.

4. **Chart Generation**: Chart generation refers to the recommendation and construction of prevalent charts derived from a given table (Moritz et al. 2019; Luo et al. 2018; Zhou et al. 2021).

We choose commonly used Python libraries for each task as follows, to address the corresponding analysis query:

- Rudimentary Operations: Pandas[1] (excluding *plotting*[2]).
- Each task of Basic Insights: Custom functions are implemented to perform the mentioned tasks, and provide results for evaluation.
- Forecasting: Greykite[3] (*Forecaster*), Prophet[4] (*Prophet*).
- Chart Generation: Matplotlib[5] (*pyplot*).

2.2 Unclear Queries

In many real situations, users do not directly provide complete queries, but rather give queries with some unclear intents. There are various ways to address them, such as recommending completions for the missing intents or guiding users to complete the query. This paper focuses on proposing a benchmark and does not explore the solution methods in depth. We only use the model for recommendations, which can also satisfy the exploration of the next purpose.

Secondly, the analysis and recommendation capabilities of large language models can be explored through unclear queries. When recommending for unclear queries, the model not only needs to possess semantic parsing capabilities but also requires analytical recommendation capabilities. Exploring these capabilities of large language models is crucial for better utilizing them in the analytical domain.

An unclear query lacks the essential information required to perform tasks. In other words, in the query, there are missing parameters for generating the analysis code which consists of operations from the chosen libraries. Since the same task may require different parameters in different libraries, we have combined the representatively used libraries for each task in §2.1 and selected the essential parameters as shown in Table 1. Some parameters are not provided for missing parameters as follows:

- When a parameter is absent, the associated operator will be excluded from use. The parameters for this scenarios are *dimension field* for rudimentary operations, *filter condition* for rudimentary operations, *insight type* for basic insights.
- Parameters are typically not mentioned in the query or possess standard default values. The parameters for similar scenarios are *confidence* for forecasting, *p-value* for basic insights, *measure aggregation* for basic insights.

In addition to missing parameters, there are other types of unclear queries, such as, ambiguous parameters, unclear tasks. For ambiguous parameters, a query may have all parameters provided, but they are ambiguous or vague. *E.g.*, a table has two fields, UnitPrice and TotalPrice, but the query

[1] https://pandas.pydata.org/

[2] https://pandas.pydata.org/docs/reference/plotting.html

[3] https://github.com/linkedin/greykite

[4] https://github.com/facebook/prophet

[5] https://matplotlib.org/

Tasks	Parameters	Meanings of Parameters and Missing Parameters Query
Rudimentary Operations	clear	E.g., Which brand has the highest total sales in 2023?
	field (msr_field)	Measure field for sort or aggregation. E.g., Which brand had the best overall in 2023?
	agg (agg_func)	Aggregation function, such as sum, average... E.g., Which brand has the highest sales in 2023?
Basic Insights	clear	E.g., Does total increase over time?
	field	Field for the insight. E.g., Is there an increase over time?
Forecasting	clear	E.g., Forecast the cost data of different brands, categories and models of cars in 2012.
	forecast field	Measure field used for forecasting. E.g., What will be for different categories and models of cars in 2012?
	steps / freq	Forecasting steps and/or frequency. E.g., What will be the sales and cost data of different brands, categories and models of cars?
Visualization	clear	E.g., Help me create a bar chart to visualize the Frequency field for the HH field.
	chart type	Char type, including lineChart, barChart, scatterChart, pieChart. E.g., Help me create a chart to visualize the Frequency field for the HH field.
	x fields	Fields for x-axis. E.g., Help me create a bar chart to visualize the Frequency field.
	y fields	Fields for y-axis. E.g., Help me create a bar chart to visualize for the HH field.

Table 1: Taxonomy and Examples of Unclear Queries

only mentions "price", resulting in ambiguity. Another example is when a query mentions filtering "young people", but there is no universally accepted definition of "young", leading to varied age filters. There are more details in (Wang et al. 2023), and we will not discuss this further in this paper. For unclear tasks, a query does not explicitly specify what task to use for analysis, *e.g.*, "What should I do if I want to get more profits". This query only proposes a goal without specifying any tasks, and solving such problems requires stronger problem-solving abilities. In this work, we will not discuss this further and will consider it as future work.

3 Text2Analysis Dataset

In this section, we introduce how to collect the Text2Analysis dataset and ensure its quality. To better accomplish the following annotation, we established a tabular data annotation website based on Label Studio[6].

3.1 Data Collection

According to the definition in §2, our benchmark requires a quadruple $(table, query, code, result)$ for each example. However, it is extremely challenging to collect such quadruples as the annotation cost is quite high. It requires annotators with related expert backgrounds. Moreover, due to the involvement of code, the quality and time requirements for the annotators are even more stringent, resulting in an increased annotation cost. To address this issue, we propose the following ingenious and reliable annotation methods utilizing large language models as illustrated in Figure 3.

Forward Annotation: $(table, query) \rightarrow (code, result)$. 9 experts with extensive experience in the field of data analysis are invited to participate in the annotation process. First, we collected real tables and queries from the annotation. Subsequently, on the basis of them, the annotators labeled the code. To accelerate the annotation process, we designed an initial code generation tool and an automatic debugging tool using large language models. Annotators generated the initial version of the code, modified it, and iterated with the automatic debugging tool to ensure the accuracy and executability of the code.

In the initial code generation tool, we used the selected libraries and encapsulated some functions for different tasks, allowing the large language model to generate code using these functions. For the automatic debugging tool, we input the original query, original code, and error messages into the large language model and carried out multiple iterations to obtain executable code.

Reverse Generation from Codes Snippets: $(table, code, result) \rightarrow (query)$. We have crawled and collected executable code and tables from major data analysis libraries, as detailed in Section §2.1. By employing a few-shot setting of GPT-4 and human verification, we reverse-engineer the data operations performed by the code to generate corresponding user-friendly natural language queries. To enhance the dataset's diversity, we adjusted parameters within the code

[6]https://labelstud.io/

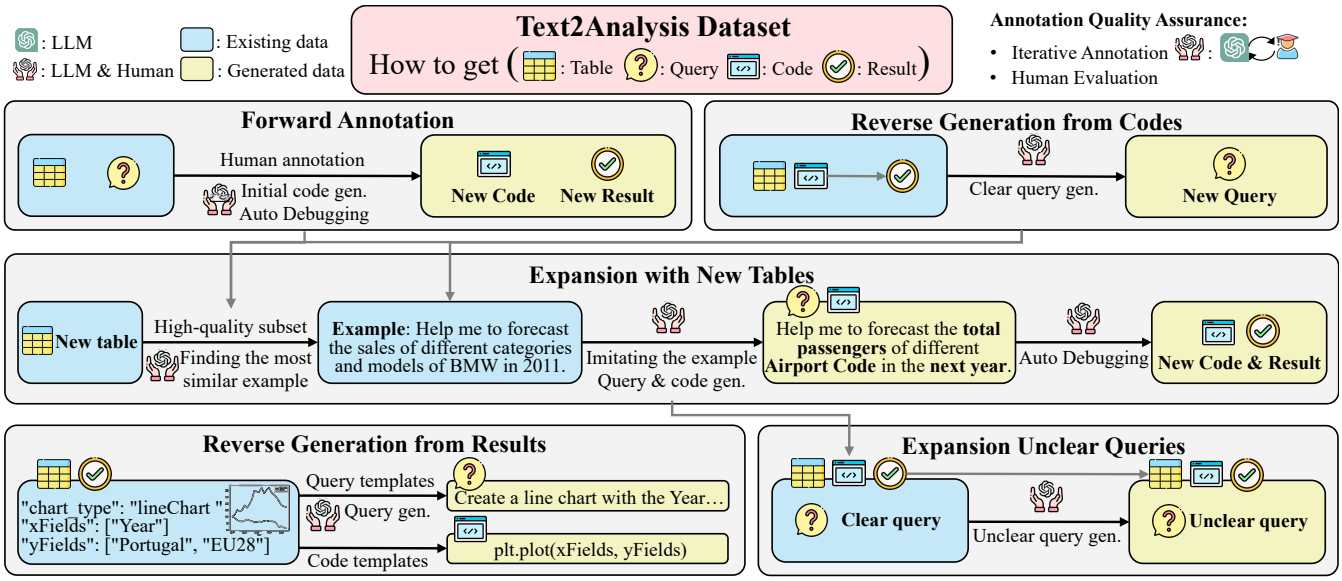

Figure 3: Collection and Generation of (table, query, code, result) Tuples in Text2Analysis.

snippets, resulting in a broader range of queries.

Reverse Generation from Results: $(table,\ result) \rightarrow (code,\ query)$. Some existing datasets provide table and analysis result but lack queries and code. We utilized these results and designed code templates, which were filled in to generate the corresponding code. Then, using result, code, tables, and large language models, we generated queries. We perform this method for chart generation on a high-quality subset of chart corpus in AnaMeta (Zhou et al. 2021).

Expansion with New Tables: $(table) \rightarrow (query,\ code,\ result)$. In order to enrich the diversity of tables and expand our dataset, we expand with new high-quality tables. After the previous processes, we collect a sub-dataset involving various tasks. We identify new tables and matched them with similar example tables in the sub-dataset. We then use large language models to mimic and generate corresponding queries, code, and results for new tables from examples.

Expansion Unclear Queries: $(clear\ \ query) \rightarrow (unclear\ query)$. In order to obtain more unclear queries, we remove the corresponding parameters from the existing queries and rewrite them. In the collected sub-dataset, we have already annotated the task taxonomy. For the corresponding task, we explicitly specify the corresponding parameter that needs to be removed for a query ($query_{old}$) and utilizes large language models to assist in generating new unclear query ($query_{new}$) in a user-oriented tone. The unclear query also requires recommended lists of corresponding codes ($code_{new}$) and results ($result_{new}$). We use the table's ($table_{old}$) existing codes ($code_{old}$) and results ($result_{old}$) that can serve as outputs for the unclear query, forming the recommended list. Because the existing data collected for the table is filled in by real users or comes from real data, it represents the most common analysis queries and results for the table, making them suitable for forming the recommended list of unclear query outputs.

3.2 Annotation Quality Assurance

To ensure the quality of Text2Analysis, we perform iterative annotation and human evaluation.

Iterative Annotation and Refinement To enhance the credibility of the large language model's generation, each time we use them for generation, we annotate and verify at least 100 samples afterward. We then modify the instructions and examples, iterating through this process for at least two rounds. This iterative approach ensure the accuracy and quality of the generated examples, resulting in a more reliable and representative dataset for evaluation purposes.

Human Evaluation To evaluate the quality of Text2Analysis annotations, we also perform a human evaluation after the data collection is completed. We sample 100 $(table,\ query,\ code,\ result)$ pairs and invited 7 experts with data analysis experience to perform the annotation. The evaluation is conducted from five perspectives: query, task taxonomy, unclear query taxonomy, code, and result correctness using scores 0 or 1 (incorrect or correct). Each sample is annotated twice. The average score of each perspective is 0.96, 0.92, 0.93, 0.89, and 0.88, respectively. The overall average score is 0.91 and Cohen's Kappa (Cohen 1960) is 0.79 (value range:$[-1,\ 1]$). This indicates that Text2Analysis has a high annotation quality and inter-annotator agreement.

3.3 Data Statistics and Distribution

Text2Analysis encompasses a total of 2249 $(table,\ query,\ code,\ result)$ pairs, sourced from 347 distinct tables. Queries of Text2Analysis encompass a variety of tasks, as demonstrated in Figure 4. And they encompass a diversity of unclear queries, as demonstrated in

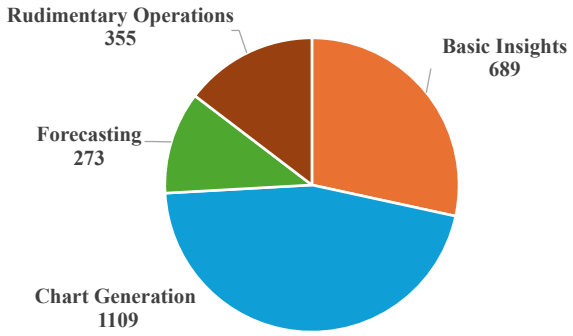

Figure 4: Analysis Task Distribution of All Queries.

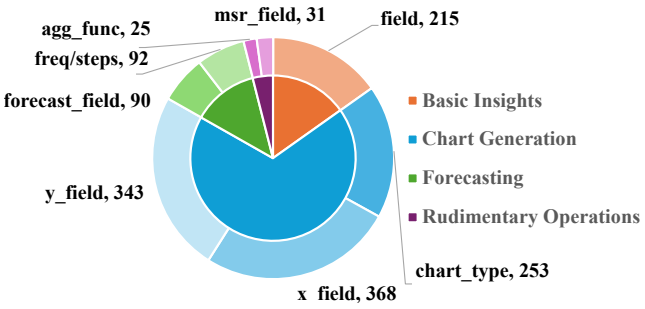

Figure 5: Task & Parameter Distribution of Unclear Queries.

Figure 5. Those figures highlight the distribution of queries and code and further showcase the diversity of the dataset and the difficulty of the problem.

4 Evaluation Methodology

4.1 Baselines

We evaluate the performance of three types models namely GPT family models, code Generation models and tabular models on Text2Analysis:

GPT family models: GPT-4 models (OpenAI 2023) are potent large-scale language models with the ability to generate human-like text and high-quality code. They perform a wide range NLP tasks well with zero or few shots.

Code Generation models: StarChat-α/β (Tunstall et al. 2023) and CodeGen2.5 (Nijkamp et al. 2023) are language models specifically designed to serve as effective coding assistants, providing valuable support to programmers.

StarChat-α/StarChat-β, derived from the StarCoder(Li et al. 2023) family, are fine-tuned language models with 15.5 billion parameters, adept at aiding programmers across 80+ programming languages. Unlike original StarCoder models that focused on code completion, StarChat versions are better suited for Text2Analysis tasks that require query instructions and task explanations.

CodeGen2.5, an autoregressive language model built upon CodeGen2. The model is trained on StarCoderData for 1.4T tokens, achieving competitive results compared to StarCoderBase-15.5B with less than half the size.

Tabular models: TAPEX(Liu et al. 2022) (Table Pre-training via Execution) is a straightforward yet highly effective pre-training method designed to enhance existing models with table reasoning capabilities. It achieves state-of-the-art performance on TableQA, Text2SQL and TabFact datasets such as WikiSQL(Zhong, Xiong, and Socher 2017).

4.2 Evaluation Metrics

Due to the numerous tasks involved in the problem and the outputs consisting of both code and results, evaluating the generated solutions with appropriate metrics poses a challenge. We have selected three metrics to evaluate from different perspectives. Executable code ratio evaluates the model's ability to generate executable code. Pass rate evaluates the correctness of the generated code. Regression scores measure the predicting capability of the chosen model within the generated code.

Executable code ratio (ECR): It refers to the proportion of generated code that is executable. It evaluates the model's ability to generate executable code.

$$ECR = \frac{\#(samples\_with\_executable\_code)}{\#(total\_samples)} \quad (1)$$

Pass Rate (pass@1): It is a metric for code generation testing that indicates the proportion of code generated accurately. In this work, there exists only one singular test case for each sample, as queries are aimed at a specific table (test case). So pass rate is equivalent to the accuracy in this work.

$$pass@1 = \frac{\#(samples\_with\_correct\_code)}{\#(samples\_with\_executable\_code)} \quad (2)$$

When determining whether a code snippet passes, it should exactly match the results of executing the generated code with the results of executing the annotated code. To ensure a fair comparison, we standardize the output format and establish a checker to avoid misidentifying results with different formats but identical content as true negatives. The standardized output format is as follows:

• The result must be one variable, one of pd.DataFrame, List[int], List[float], List[str], int, float, str, dict.

• For chart generation, assign a result dictionary: {"x_fields": field_name, "y_fields": [field_name1, field_name2...], "chart_type": chart_type}. Choose chart_type from lineChart, barChart, scatterChart, pieChart.

• For basic insights, answer the query in the same format as one or more outputs from our custom function.

The rules of the checker are as follows:

• For each value, we convert it to a string for comparison, including every value in lists, dictionaries, and DataFrames. If it is a number, convert it to a string after rounding to two decimal places. If it is a time, convert it to a string using a consistent format.

• For the comparison between DataFrames and other types: sometimes, a DataFrame is used to store a single value or a list. If the DataFrame contains only one value, that is, one row and one column, we extract it as a single value for subsequent comparisons. If there is only one column, we extract its values as a list for further comparisons.

Model	Overall		Rudimentary Operations		Basic Insights		Forecasting		Chart Generation	
	ECR	pass@1	ECR	pass@1	ECR	pass@1	ECR	pass@1	ECR	pass@1
GPT-4	**69.46%**	**41.01%**	**86.76%**	**56.82%**	**71.41%**	7.31%	5.13%	**35.71%**	79.62%	**54.36%**
StarChat-α	39.60%	32.88%	61.08%	48.34%	18.16%	**14.29%**	6.59%	0.00%	54.09%	32.84%
StarChat-β	60.06%	32.88%	46.20%	41.46%	41.22%	10.92%	**15.75%**	2.33%	**88.91%**	38.54%
CodeGen2.5	30.83%	19.39%	30.70%	36.70%	28.16%	18.56%	0.73%	0.00%	39.95%	15.58%
TAPEX	-	-	-	11.55%	-	-	-	-	-	-

Table 2: Baseline Performance on Text2Analysis. (ECR = executable code ratio, pass@1 = pass rate).

• For the comparison between lists/dictionaries and other types: if a list or dictionary contains only one element, we extract it as a single value for subsequent comparisons.

• For comparing DataFrames with DataFrames: if the operations do not include sorting, it is acceptable for the order of rows in the DataFrames to be inconsistent. The header names of newly generated columns are allowed to differ. For missing steps/frequency forecasting, it is permissible for the predicted results to be a subset of the ground truth or vice versa, i.e., one table's rows can contain another table's rows.

Regression scores: For forecasting, it measures the predicting capability of the chosen model within the generated code between the generated results and the ground truth. It includes CORR, RMSE, MAE, and MedAE.

$$regression\_score = \frac{\sum_{i=0}^{M}(regression\_score_i)}{\#(total\_samples)} \quad (3)$$

where, M is the number of $total\_samples$.

We choose to use regression scores for further evaluation because pass rate does not provide a comprehensive assessment of the quality of the generated forecasting code. For example, if the library or forecasting model is altered in the generated code, achieving an exact match with the results executed from the annotated code can be challenging. However, regression metrics can still be used to assess the quality of the predictions with ground truth.

To obtain the ground truth for forecasting, we preprocess the input table. During data collection, we truncate the original table according to the length of the time period required for the query's prediction. In other words, we extract a portion of the table corresponding to the prediction length as ground truth, while the remaining table is used as input for the Text2Analysis task. It is worth noting that when calculating the statistical values in §3.3, truncated tables from the same original table are still considered as one table.

5 Experiments

We conduct experiments on the five baselines introduced in §4.1. For the GPT family and code generation models, we design instruction prompts that include the HTML table, constraints on code generation libraries, requirements for result formatting, and so on. The parameter details for each model in the experiment are as follows:

• GPT-4: model: gpt-4-32k, temperature: 0, maximum_length: 4096.

Model	CORR ↑	RMSE ↓	MAE ↓	MedAE ↓
GPT4	0.10	0.27	0.24	0.27
StarChat-β	0.16	0.76	0.76	0.73

Table 3: Regression Scores for Forecasting. We show the models with the highest ECR and pass@1 in Table 2. ↓ means smaller is better. CORR is better when its absolute value is closer to 1. So when CORR$\in [0,\ 1]$, larger is better.

• StarChat-α: model: starchat-alpha[7], temperature: 0.2, max_new_tokens: 1024.

• StarChat-β: model: starchat-beta[8], temperature: 0.2, max_new_tokens: 1024.

• CodeGen2.5: model: codegen25-7b-instruct[9], temperature: 0.2, max_new_tokens: 1024.

• TAPEX: model: tapex-large-finetuned-wtq[10].

5.1 Main Results

As shown in Table 2, overall experimental results demonstrate that GPT-4 outperforms other models. It achieves the highest ECR on the majority of tasks and the highest pass rate across all tasks. GPT-4's relatively better performance can be attributed to its code generation capabilities and context learning abilities. The former allows it to generate more accurate and executable code. The latter enables it to better understand and integrate the given query and instructions.

Code generation models exhibit overall performance that is comparable to GPT-4 in generating executable code. However, the pass rate of the generated code is relatively low, with the overall pass rate being 24.86% lower than that of GPT-4. This can be attributed to the limited in-context learning capabilities of these models, which results in a restricted ability to capture the meaning of the given query and generate the correct code accordingly.

The tabular model is currently only capable of completing rudimentary tasks. Their performance on the Text2Analysis benchmark is subpar, with an pass rate of only 11.55%. One reason for this is that rudimentary tasks in the benchmark involve complex pivot operations and calculations, which existing tabular models struggle with. These models excel at

[7]https://huggingface.co/HuggingFaceH4/starchat-alpha

[8]https://huggingface.co/HuggingFaceH4/starchat-beta

[9]https://huggingface.co/Salesforce/codegen25-7b-instruct

[10]https://huggingface.co/microsoft/tapex-large-finetuned-wtq

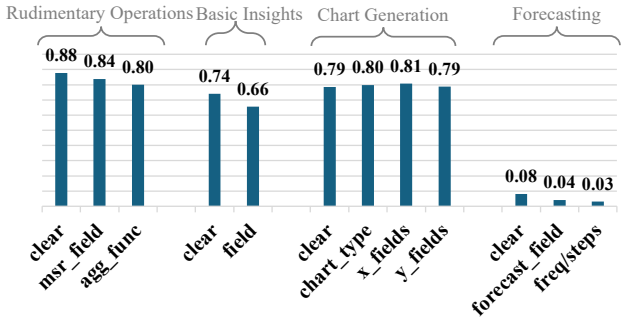

Figure 6: ECR for Unclear Queries on GPT-4.

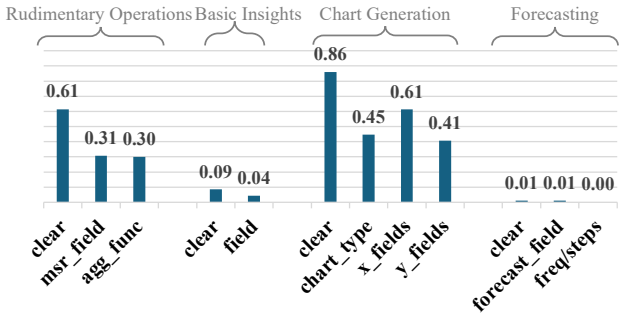

Figure 7: Pass rate for Unclear Queries on GPT-4.

querying tables to find values in the original table but falter when it comes to performing complex calculations.

Table 3 presents the regression scores for the forecasting task, demonstrating that the code generated by the baselines has limited prediction capabilities. To successfully tackle forecasting task, not only do the models need to generate correct code, but they also need to select appropriate prediction models. The current baseline models fall short in these aspects, highlighting the need for further development and research to improve their performance on forecasting tasks.

The overall performance of the baseline models leaves room for improvement. Its best performance is on rudimentary operations tasks, with an pass rate of only 57%. Moreover, on complicated forecasting, it can only reach an pass rate of 14%. This highlights the significant exploration space that still exists, presenting opportunities for further research and the development of more advanced models.

5.2 Unclear Queries Results

When facing clear queries, models have strong parsing and code generation capabilities for data analysis. As in Figure 7, when the query is clear, the pass rate of chart generation is as high as 86%. When facing a clear query, the model needs to first parse the natural language into corresponding tasks and parameters, and then generate the correct code.

The ability to recommend fields for advanced data analysis tasks, particularly measure fields (columns with numerical attributes in a table), can be enhanced in large language models. As shown in Figure 6, the ECR decreases by 8% on the basic insights task when the field is missing. As shown in Figure 7, when the measure field is missing, the pass rate has decreased for most tasks, especially the chart generation task, which has decreased by 25%. If we want to improve the recommendation analysis, future work needs to consider injecting the knowledge of recommending analytical columns into the large language models.

The code generation capability for more complex libraries needs to be enhanced. As shown in Figure 6 and Figure 7, both the ECR and pass rate for forecasting tasks are below 1%. The forecasting task library involves more complex operations such as parameter input and model training. In more than 50% of the cases, GPT-4 generates incorrect parameters or input parameters that are not included in the operations. Additionally, in some instances, the code does not select the correct model, rendering it unable to successfully fit the data.

6 Related Work

6.1 Tabular Benchmark

TableQA and Text2SQL are prevalent tasks in tabular data analysis. These tasks involve answering user queries based on a source table. Notable their datasets include WikiTableQuestions (Pasupat and Liang 2015), WikiSQL (Zhong, Xiong, and Socher 2017) and so on. Although numerous related datasets encompass a wide variety of table types, the primary focus remains on descriptive data analysis. Text2Analysis expands to more tabular analysis tasks.

To address tabular tasks, pre-trained models like TAPAS (Herzig et al. 2020), TAPEX(Liu et al. 2022), etc. have been employed. Concurrently, large language models have also been used in approaches like DATER (Ye et al. 2023), StructGPT (Jiang et al. 2023), etc. In this work, we evaluate SOTA tabular models as comparison baselines.

6.2 Large Language Models

Recent advancements in large language models (LLMs) like GPT-3.5 and GPT-4 (OpenAI 2023) have improved few-shot prediction capabilities and human instruction following. And they have shown promising capabilities to accelerate tabular data analysis (Chen 2023; Ye et al. 2023; Ma et al. 2023; Jiang et al. 2023). Text2Analysis is proposed as a new benchmark to further explore LLMs' upper limits in challenging tabular data analysis tasks.

7 Conclusion

In conclusion, we have presented the Text2Analysis dataset that addresses the research gap in advanced analysis tasks and unclear queries in the context of tabular data analysis. Our dataset provides a comprehensive taxonomy of advanced analysis and unclear queries, which enables the evaluation of the analytical abilities of large language models. We have also proposed five innovative and reliable annotation methods that leverage large language models to accelerate the annotation process and increase the volume of annotation. Our evaluation of five SOTA models on the Text2Analysis dataset reveals their strengths and weaknesses in handling advanced analysis tasks and unclear queries, providing valuable insights for future research.

References

Chen, W. 2023. Large Language Models are few(1)-shot Table Reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, 1120–1130. Dubrovnik, Croatia: Association for Computational Linguistics.

Chen, Y.; Yang, J.; and Ribarsky, W. 2009. Toward effective insight management in visual analytics systems. In *2009 IEEE Pacific Visualization Symposium*, 49–56. IEEE.

Cohen, J. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1): 37–46.

Date, C. J. 1989. *A Guide to the SQL Standard*. Addison-Wesley Longman Publishing Co., Inc.

Delen, D.; and Ram, S. 2018a. Research challenges and opportunities in business analytics. *Journal of Business Analytics*, 1(1): 2–12.

Delen, D.; and Ram, S. 2018b. Research challenges and opportunities in business analytics. *Journal of Business Analytics*, 1(1): 2–12.

Ding, R.; Han, S.; Xu, Y.; Zhang, H.; and Zhang, D. 2019. QuickInsights: Quick and Automatic Discovery of Insights from Multi-Dimensional Data. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, 317–332. New York, NY, USA: Association for Computing Machinery. ISBN 9781450356435.

Dong, L.; and Lapata, M. 2016. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 33–43. Association for Computational Linguistics.

Herzig, J.; Nowak, P. K.; Müller, T.; Piccinno, F.; and Eisenschlos, J. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4320–4333.

Hosseini, R.; Yang, K.; Chen, A.; and Patra, S. 2021. A flexible forecasting model for production systems. *arXiv preprint arXiv:2105.01098*.

Jiang, J.; Zhou, K.; Dong, Z.; Ye, K.; Zhao, W. X.; and Wen, J.-R. 2023. StructGPT: A general framework for Large Language Model to Reason on Structured Data.

Katsogiannis-Meimarakis, G.; and Koutrika, G. 2021. *A Deep Dive into Deep Learning Approaches for Text-to-SQL Systems*, 2846–2851. Association for Computing Machinery. ISBN 9781450383431.

Li, R.; Allal, L. B.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; Liu, Q.; Zheltonozhskii, E.; Zhuo, T. Y.; Wang, T.; Dehaene, O.; Davaadorj, M.; Lamy-Poirier, J.; Monteiro, J.; Shliazhko, O.; Gontier, N.; Meade, N.; Zebaze, A.; Yee, M.-H.; Umapathi, L. K.; Zhu, J.; Lipkin, B.; Oblokulov, M.; Wang, Z.; Murthy, R.; Stillerman, J.; Patel, S. S.; Abulkhanov, D.; Zocca, M.; Dey, M.; Zhang, Z.; Fahmy, N.; Bhattacharyya, U.; Yu, W.; Singh, S.; Luccioni, S.; Villegas, P.; Kunakov, M.; Zhdanov, F.; Romero, M.; Lee, T.; Timor, N.; Ding, J.; Schlesinger, C.; Schoelkopf, H.; Ebert, J.; Dao, T.; Mishra, M.; Gu, A.; Robinson, J.; Anderson, C. J.; Dolan-Gavitt, B.; Contractor, D.; Reddy, S.; Fried, D.; Bahdanau, D.; Jernite, Y.; Ferrandis, C. M.; Hughes, S.; Wolf, T.; Guha, A.; von Werra, L.; and de Vries, H. 2023. StarCoder: may the source be with you!

Liu, Q.; Chen, B.; Guo, J.; Ziyadi, M.; Lin, Z.; Chen, W.; and Lou, J.-G. 2022. TAPEX: Table Pre-training via Learning a Neural SQL Executor. In *International Conference on Learning Representations*.

Luo, Y.; Qin, X.; Tang, N.; and Li, G. 2018. DeepEye: Towards Automatic Data Visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 101–112. IEEE Computer Society.

Ma, P.; Ding, R.; Han, S.; and Zhang, D. 2021. Metainsight: Automatic discovery of structured knowledge for exploratory data analysis. In *Proceedings of the 2021 International Conference on Management of Data*, 1262–1274.

Ma, P.; Ding, R.; Wang, S.; Han, S.; and Zhang, D. 2023. Demonstration of InsightPilot: An LLM-Empowered Automated Data Exploration System. *arXiv preprint arXiv:2304.00477*.

Moritz, D.; Wang, C.; Nelson, G. L.; Lin, H.; Smith, A. M.; Howe, B.; and Heer, J. 2019. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 25(1): 438–448.

Nijkamp, E.; Hayashi, H.; Xiong, C.; Savarese, S.; and Zhou, Y. 2023. CodeGen2: Lessons for Training LLMs on Programming and Natural Languages. *arXiv preprint*.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Pasupat, P.; and Liang, P. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1470–1480. Beijing, China: Association for Computational Linguistics.

Taylor, S. J.; and Letham, B. 2018. Forecasting at scale. *The American Statistician*, 72(1): 37–45.

Tunstall, L.; Lambert, N.; Rajani, N.; Beeching, E.; Le Scao, T.; von Werra, L.; Han, S.; Schmid, P.; and Rush, A. 2023. Creating a Coding Assistant with StarCoder. https://huggingface.co/blog/starchat. Accessed: 2023-08-15.

Wang, B.; Gao, Y.; Li, Z.; and Lou, J.-G. 2023. Know What I don't Know: Handling Ambiguous and Unknown Questions for Text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*, 5701–5714. Toronto, Canada: Association for Computational Linguistics.

Ye, Y.; Hui, B.; Yang, M.; Li, B.; Huang, F.; and Li, Y. 2023. Large Language Models are Versatile Decomposers: Decompose Evidence and Questions for Table-based Reasoning.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103.

Zhou, M.; Li, Q.; He, X.; Li, Y.; Liu, Y.; Ji, W.; Han, S.; Chen, Y.; Jiang, D.; and Zhang, D. 2021. Table2Charts:

Recommending Charts by Learning Shared Table Representations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, 2389–2399. ISBN 9781450383325.