NBDESCRIB: A Dataset for Text Description Generation from Tables and Code in Jupyter Notebooks with Guidelines

Anonymous ACL submission

Abstract

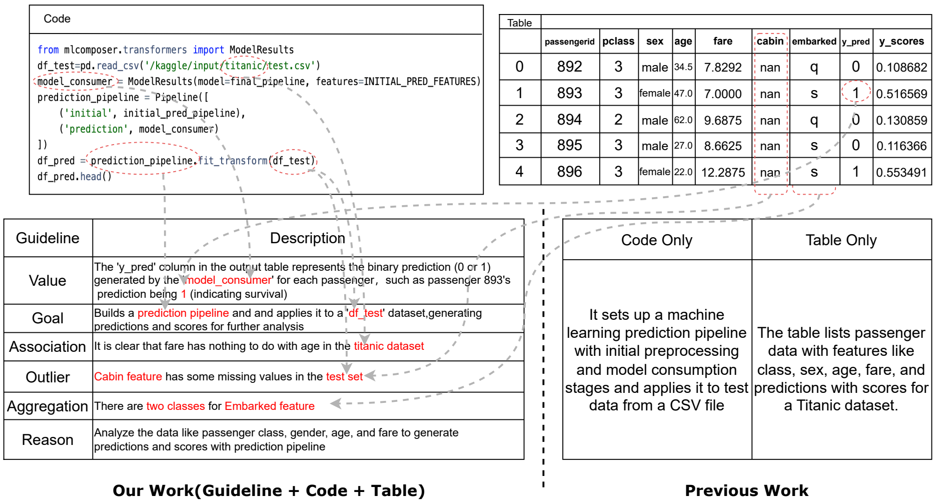
Generating cell-level descriptions for Jupyter Notebooks, which is a major resource consisting of codes, tables, and descriptions, has been attracting increasing research attention. However, existing methods for Jupyter Notebooks mostly focus on generating descriptions from code snippets or table outputs independently. On the other side, descriptions should be personalized as users have different purposes in different scenarios while previous work ignored this situation during description generation. In this work, we formulate a new task, personalized description generation with code, tables, and user-written guidelines in Jupyter Notebooks. To evaluate this new task, we collect and propose a benchmark, namely NBDESCRIB, containing code, tables, and user-written guidelines as inputs and personalized descriptions as targets. Extensive experiments show that while existing models of text generation are able to generate fluent and readable descriptions, they still struggle to produce factually correct descriptions without user-written guidelines. CodeT5 achieved the highest scores in Orientation (1.27) and Correctness (-0.43) among foundation models in human evaluation, while the ground truth scored higher in Orientation (1.45) and Correctness (1.19). Common error patterns involve misalignment with guidelines, incorrect variable values, omission of important code information, and reasoning errors. Moreover, ablation studies show that adding guidelines significantly enhances performance. both qualitatively and quantitatively.

1 Introduction

In recent years, computational notebooks like Jupyter have become popular among data scientists and machine learning researchers for documenting ideas, writing code, and visualizing results in a single document (Wang et al., 2021a). Description in a notebook provides a rich medium for users to record what the code does and the reasoning behind it. Description is found essential for data scientists to share or reuse code (Zhang et al., 2020; Chattopadhyay et al., 2020). However, research has shown that many data scientists still neglect to write appropriate descriptions for their code in notebooks, especially for code output (*i.e.* table), as they feel writing description will slow down their coding process (Ramasamy et al., 2023). Rule et al. (2018a) reported that among one million computational notebooks on Github, 25% of them have no comment. Besides, unlike other integrated development environments (IDEs) such as Visual Studio, data scientists working in computational notebooks often write concise descriptions, typically less than 100 words(Wang et al., 2022). This brevity poses challenges for large language models (LLMs), such as GPT and LLaMA, limiting their ability to generate accurate markdown descriptions(Park and Choi, 2024).

Existing literature has explored techniques to generate descriptions for code snippets or table outputs independently (Richardson et al., 2017; Li et al., 2021; Liu et al., 2018; Parikh et al., 2020a; An et al., 2022; Zhao et al., 2023b,a; Ding and Xu, 2023; Cm et al., 2023), achieving satisfying performance. However, in real-world applications, documentations are always related to the corresponding code and table output, at the same time. For example, ground truth descriptions require summarization from the code and table output simultaneously, as shown in Section 1.

Moreover, while the same code-table output pair could serve different purposes under different user requirements, existing work (Zhao et al., 2023b; Chen et al., 2020; Zhao et al., 2023a; Ding and Xu, 2023; Cm et al., 2023; Richardson et al., 2017; An et al., 2022; Liu et al., 2021; Khan and Uddin, 2022; Koehn and Knowles, 2017; Pasupat and Liang, 2015; Ye et al., 2023) only focuses on description generation for either code or table without considering various user requirements. As shown in Figure



Code

```
from mlcomposer.transformers import ModelResults
df_test=pd.read_csv('/kaggle/input/titanic/test.csv')
model_consumer = ModelResults(model=final_pipeline, features=INITIAL_PRED_FEATURES)
prediction_pipeline = Pipeline([
    ('initial', initial_pred_pipeline),
    ('prediction', model_consumer)
])
df_pred = prediction_pipeline.fit_transform(df_test)
df_pred.head()
```

Table

	passengerid	pclass	sex	age	fare	cabin	embarked	y_pred	y_scores
0	892	3	male	34.5	7.8292	nan	q	0	0.108682
1	893	3	female	47.0	7.0000	nan	s	1	0.516569
2	894	2	male	62.0	9.6875	nan	q	0	0.130859
3	895	3	male	27.0	8.6625	nan	s	0	0.116366
4	896	3	female	22.0	12.2875	nan	s	1	0.553491

Guideline	Description
Value	The 'y_pred' column in the output table represents the binary prediction (0 or 1) generated by the 'model_consumer' for each passenger, such as passenger 893's prediction being 1 (indicating survival)
Goal	Builds a prediction pipeline and and applies it to a 'df_test' dataset,generating predictions and scores for further analysis
Association	It is clear that fare has nothing to do with age in the titanic dataset
Outlier	Cabin feature has some missing values in the test set
Aggregation	There are two classes for Embarked feature
Reason	Analyze the data like passenger class, gender, age, and fare to generate predictions and scores with prediction pipeline

Our Work(Guideline + Code + Table)

Code Only	Table Only
It sets up a machine learning prediction pipeline with initial preprocessing and model consumption stages and applies it to test data from a CSV file	The table lists passenger data with features like class, sex, age, fare, and predictions with scores for a Titanic dataset.

Previous Work

Figure 1: An example in our proposed NBDESCRIB dataset, which targets generating **high-fidelity and personalized descriptions** based on the input of codes, tables, and user-written guidelines. Previous methods and benchmarks focus on understanding the codes or tables only, which makes the generated description unfaithful.

1, under different user-written guidelines, the focus of different ground-truth descriptions could differ from each other.

To address the aforementioned issues and step forward in faithful and personalized description generation, in this paper, we propose a challenging user-written guideline-based text generation task focusing on the table and code description generation (**TCDG**) for Jupyter Notebooks. Given codes and output tables, the goal of **TCDG** is to produce a concise description under different user-written guidelines. The guidelines will be of a given category corresponding to the type of target text as shown in Table 2.

Moreover, we construct the first benchmark (**NBDESCRIB**) for TCDG that contains 3,924 processed code-table-description pairs extracted from highly-ranked notebooks from Kaggle competitions and identifies 15 guideline categories of the texts (details in Section3). Specifically, the raw Jupyter Notebook data with tables, code, and associated text from popular Kaggle competitions is crawled. However, the raw data cannot be directly used due to the large amount of noise (Mondal et al., 2023; Lin et al., 2022). For example, *"I plan to refine the models by using more sophisticated machine learning techniques."* is about personal experiences and future plans, which is not useful for text generation tasks. On the other side, the user-written guidelines and ground-truth descriptions in the markdown cell generally contain multiple different purposes or facts on the tables. To reduce the noise in the raw data, we recruit annotators to first break down the markdown cells and make each piece of text only contain one purpose or fact. For each guideline category, we create the label as well as descriptions, and then curate the tables and filter out the noise text. Finally, the text descriptions in our data are natural, faithful, and specifically targeted under different guidelines (Figure 1).

Next, we evaluate the performance of existing pretrained models. The ablation study shows that guidelines significantly affect the final performance at different levels, which demonstrates the validity of our task. Human evaluation further shows that these advanced models (*i.e.*, CodeT5, Llama, and GPT-4o) still struggle to produce faithful enough results, regardless of high-quality training data. Moreover, common error patterns involve misalignment with guidelines, incorrect variable values, omission of important code information, and reasoning errors.

Based on these results, we integrated our approach into a user-facing application to explore Human-AI collaboration in code documentation. In a follow-up study, users found that the generated documentation reminded them of documenting codes they might have overlooked and increased their satisfaction with their notebooks.

The main contributions of our work are:

- We formulate a novel task, **TCDG**, and collect a high-quality benchmark, namely NBDESCRIB;
- Experiments show that fine-tuned LMs (CodeT5) outperform powerful pretrianed models, *i.e.*, GPT-4o and Llama, highlighting the vulnerability of LLMs on TCDG. Ablation studies and downstream application demonstrate that guidelines significantly enhance model performance, helping users create accurately oriented and reasonable descriptions.
- Error analysis shows that LLMs fail to align with guidelines, understand variable values, and rea-

son.

2 Related Work

In this work, we focus on table and code description generation (TCDG) tasks. Our work is closely related to table-to-text generation and code documentation generation (CDG). Most existing datasets for table-to-text generation (Li et al., 2021; Liu et al., 2018; Parikh et al., 2020a; Dhingra et al., 2019; Zhao et al., 2023b; Chen et al., 2020; Zhao et al., 2023a; Ding and Xu, 2023; Cm et al., 2023; Zhang et al., 2024; Guo et al., 2024; Min et al., 2024) or code documentation (Richardson et al., 2017; An et al., 2022; Liu et al., 2021; Khan and Uddin, 2022; Dvivedi et al., 2024; Luo et al., 2024) generation contain one text per table or code on a specific topic and schema. For instance, (Suadaa et al., 2021) contains 1.3K table-documentation pairs with richer inference from scientific papers and CodeSearchNet (Husain et al., 2019) contains 2M function-documentation pairs across six programming languages (e.g., java, php, python). Differing from previous CDG and table-to-text datasets, a documentation text can correspond to both code and its table output in ours.

Previous work on table-to-text focuses on text generation for standalone table data. Parikh et al. (2020b) proposed an open domain table-to-text dataset. They collected tables from Wikidepia and paired them with single-sentence documentation. They then requested annotators to revise the Wikipedia candidate sentences into target sentences, rather than writing new ones. Several studies focused on a specific topic and schema such as WEATHERGOV (Liang et al., 2009) and ROBOCUP (Chen and Mooney, 2008), Rotowire (Wiseman et al., 2017), Wikibio (Lebret et al., 2016, Biographies), E2E (Novikova et al., 2016, Restaurants). However, they cannot provide different target texts for various data facts in tables, resulting in too singular results during model training.

Another task similar to table-to-text is table question answering (Pasupat and Liang, 2015; Wang et al., 2018; Nan et al., 2022; Cheng et al., 2021; Ye et al., 2023). While they can locate relevant tables and provide answers by tagging relevant cells, they do not provide a meaningful explanation of different kinds of data facts. There are also other sources of information that may be used in data science projects. Without following appropriate guidelines and integrating these sources, we would not be able to produce satisfactory results. Chen et al. (2019); Gupta et al. (2020) attempted to verify whether a provided textual statement is entailed or refuted by the given table. But they only address verification issues and cannot generate descriptive statements for different data fact types.

Since our work focuses on both code and table, it is essential to discuss related work on CDG, which aims to understand the code and generate the code descriptions. Typical datasets include CodeSearchNet (Husain et al., 2019) and some datasets collected from GitHub (Kanade et al., 2019) or BigQuery (Yue Wang, 2021). Recently, LLMs have been applied to the CDG task such as CodeLlama (Roziere et al., 2023), CuBERT (Kanade et al., 2019), CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2020)) or GPT (Svyatkovskiy et al., 2020; Lu et al., 2021). Some recent works explore encoder-decoder models such as PLBART (Ahmad et al., 2021), CodeT5 (Yue Wang, 2021), and TreeBERT (Jiang et al., 2021). The documentation in this task often does not cover the facts of the output from the code, only focusing on the description of the code.

Different from the aforementioned works that only focus on one text generation for a single standalone code or table, in our new TCDG task for computational notebooks, code and its table output can correspond to one description, and these descriptions may have many categories depending on the needs of the user. We thus propose to construct a notebookTCDG dataset to handle text generation of multiple guideline categories for code and its table output.

3 TCDG - Task Description

In our task, the model is provided with a long text including a code cell and its table output, as well as the corresponding guideline category description. The guideline indicates the direction of the target description generation, and the specific category is described in Section 4.3. The model is asked to read the input and generate reasonable descriptions based on the given guideline, code, and table.

3.1 Input

The input to a text generation model consists of an input text and a target document:

(1) Codes and tables from the input texts are extracted from notebooks crawled from the Kaggle website. The code provides the necessary context to understand how the table output was generated.

	Overall	Train	Dev	Test
Code-Table-Description pairs	3,924	2,747	393	785
Code vocabulary size	3,497			
Table vocabulary size	16,424			
Description vocabulary size	4,481			
Avg. # token in Description	12.41	12.37	12.51	12.52
Max. # token in Description	66	57	46	66
Std. # token in Description	7.45	7.43	7.26	7.66
Avg. # token in code cell(s)	10.68	11.17	10.46	10.22
Max. # token in code cell(s)	310	310	131	131
Std. # token in code cell(s)	19.54	21.40	16.42	15.64
Avg. # token in table	13.85	13.92	12.84	14.11
Max. # token in table	272	272	97	261
Std. # token in table	17.27	16.47	11.97	21.68

Table 1: **NBDESCRIB** dataset statistics.

By analyzing the code, one can infer the logic and algorithms applied to the input data, which facilitates accurate interpretation of the table's contents.

(2) A **guideline category description** serves as a guiding principle for generating the target description. Some descriptions focus on interpreting table content while code snippets also provide essential contextual information. At the same time, while describing the purpose of the code, we also need tables to provide essential data-driven explanations. This setup mimics the real scenario.

3.2 Output

A text generation model is employed to predict the specific guideline category of descriptions. Table code associated Markdown cells are the target documents that we collect since these cells are typically used to provide descriptive text for code and tables. Also, some Markdown cells can be used only for headings in the notebook. To exclude such Markdown cells, search for key characters like #, which generally refers to the titles.

4 NBDESCRIB

4.1 Data Collection

As we are focusing on the description generation in Jupyter Notebooks, we need to crawl a sufficient number of code-table-description pairs first. Publicly shared notebooks on GitHub are often ill-documented (Rule et al., 2018b) and do not have many tables, thus are not suitable for this task.

On the other side, Kaggle allows community members to vote up and down on uploaded notebooks, and findings show that the highly-voted notebooks are of good quality and quantity for code documentation (Wang et al., 2021b; Liu et al., 2021). Thus, we decided to utilize the top-voted and well-documented Kaggle notebooks. We crawled notebooks from seven popular competitions, *i.e.*, seven top popular Kaggle competitions - House Price Prediction, Titanic Survival Prediction, Predict Future Sales, Spaceship Titanic, U.S. Patent Phrase to Phrase Matching, JPX Tokyo Stock Exchange Prediction, and Ubiquant Market Prediction, and built around 4,000 pairs of code-table-description pairs. Links for these competitions can be found in Appendix C. To build this dataset, we also filtered out the description which is not in English. We checked the data policy of each of the competitions, and none of them have copyright issues. We also contacted Kaggle to make sure our data collection complies with the platform's policy.

4.2 Data Preprocessing

We employed the following heuristics to collect codes, tables, and Markdown:

Cell Matching: We search for codes that produce tables in the notebooks and check if there is a corresponding Markdown cell describing the code and table above. The sentences are also split if there are multiple sentences in the corresponding Markdown cell. However, there are instances where the text content is inaccurate. For example, in texts labelled as value guidelines, the extracted value might be incorrect, or in texts describing Feature Engineer guidelines, the variable name extracted from the code might be misspelled. Thus, we label each sentence and let annotators rewrite it accordingly, since each sentence may have a different description angle. Details about how annotators code the sentence and reach an agreement are shown in Section 4.3. The specific guideline details will be described in the following.

Table Processing: Since the table in Jupyter Notebook is in HTML code, to transfer it into a table format, we use HTMLParser[1] to get the data value for each row, column, and their relationship based on the tags, such as <th>, <td>. We first drop their parent tags <table> to simplify the document format. Next, we remove the tags <td> and <th> from cells to extract variables and corresponding values from the HTML code. Then we concatenate variables and values with pipe("|") to generate table description. An example of table processing is shown in Appendix I.

Table Curation: If the description contains variable names in a table, the corresponding rows and columns containing those variables are extracted to create a new table. If no key variables are included, we keep the original tables. This process

---

[1] https://docs.python.org/3/library/html.parser.html

aims to minimize the inclusion of irrelevant information. Appendix I.3 provides an example of how to implement table curation.

4.3 Guideline Category Description

Three members of the research team conducted an iterative open-coding process to analyze the collected notebooks. Differing from Wang et al. (2020), where their qualitative coding stopped at the tabular data level, and our analysis goes deep to the granularity of the cell, the cell be used to explain beyond the adjacent code cell whose output is the table: we annotate these cells' purposes and types of content. Each annotator independently analyzed the same five notebooks to develop a codebook. After discussing and refining the codebook, they again went back to recode those five notebooks and achieved pairwise inter-rater reliability ranged 0.81–0.93 (Cohen's K). To further determine the correctness of inter-annotator agreement, we let these three annotators analyze another undiscussed five notebooks and get pairwise inter-rater reliability ranging from 0.78 to 0.89 (Cohen's K) which is convincing to demonstrate the reliability of our codebooks. After getting a reliable agreement, the three coders divided and coded the remaining notebooks. In total, we identified fifteen guideline categories for the content of the markdown cells (Table 2, Appendix E provides examples).

As shown in Table 2, eleven guideline categories mainly focus on the data facts of a table. It is worth noting while these guidelines focus more on the description of the table data, the code still provides contextual information to supplement their description, as shown in Figure 1. Our analysis revealed that markdown cells are mostly used to describe the specific attribute values from the table (Value, 7.29%). Second to the Value category, 6.55% markdown cells are used to specify the outliers from the table output (Outliers).

However, these guidelines do not meet the needs of Jupyter Notebook users. This kind of markdown cells can also be used to mainly explain the beyond adjacent code cells. Even though they mainly focus on the code, a clear understanding of table data is also crucial for understanding the code logic.

We found that some of these markdown cells describe the motivation from the code descriptive text(Goal, 19.64%), to explain the results or critical decisions (Reason, 7.03%), or to describe a combination of mathematical transformations from the code (Feature Engineer, 10.02%). We also found that some markdown cells aim to provide additional context to help with code and table understanding (Complementary Details, 22.17%).

Guideline	**#**	**Description**
Value	286 (7.29%)	Get the exact data attribute values for a set of criteria.
Difference	138 (4o2%)	A comparison between at least two distinct attributes within the target object, or a comparison between the target object and previously measured values.
Trend	31 (0.79%)	Indicates a general tendency over a period of time.
Proportion	120 (3.06%)	Measure the proportion of selected data attribute(s) within a specified set
Categorization	74 (1.89%)	Select the data attribute(s) that meet the condition.
Distribution	127 (3.20%)	Show the amount of shared value for the selected data attributes or present a breakdown of all data attributes.
Rank	73 (1.86%)	Sort data attributes by their values and display a breakdown of selected attributes.
Association	165 (4.21%)	Identify the useful relationship between two or more data attributes.
Extreme	227 (5.78%)	Identify the data cases that are the most extreme in relation to the data attributes or within a specific range
Outlier	257 (6.55%)	Determine whether there are unexpected data attributes or statistically significant outliers.
Aggregation	125 (3.19%)	Calculate the descriptive statistical indicators (e.g., average, sum, count, etc.) based on the data attributes.
Goal	771 (19.64%)	Express user's goal. To say what value or function they tend to use for the later research
Reason	276 (7.03%)	Express reason using the data from the table or explain the reasons why certain functions are used or why a task is performed.
Feature Engineer	393 (10.02%)	The process of selecting, transforming, extracting, combining, and manipulating raw data to generate the desired variables for analysis or predictive modeling.
Complementary Details	870 (22.17%)	Express additional contextual elements and supporting information to enhance understanding of the primary content

Table 2: We identify 15 guideline categories based on the types of descriptions in the Markdown cells which are below the code whose output is a table.

4.4 Train / Dev / Test Splits

Overall, the dataset contains 2747 Code-Table-Description pairs in the training set, 393 pairs in the development set, and 785 pairs in the test set (see Table 1 for more statistics).

5 Experiments

Baselines. We utilized three types of models: a fine-tuned encoder-decoder-based CodeT5, the popular decoder-only LLMs (an off-the-shelf GPT-4o, Llama3, CodeLlama3), and a fine-tuned GPT-3.0. Details are shown in Appendix D.

5.1 Evaluation Metrics

We use the ROUGE scores (ROUGE-1, ROUGE-2, and ROUGE-L) (Lin, 2004), BERTScore (Zhang et al., 2019), and G-EVAL (Liu et al., 2023a) to evaluate our model's performance with regard to the ground-truth description content. Details are shown in Appendix H.

5.2 Results

The numbers in Table 3 show that this guideline-based text generation task is very challenging,

Models	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore	Pyramid Evaluation	G-EVAL:Coherence	G-EVAL:Correctness	G-EVAL:Consistency	G-EVAL:Fluency
Baselines									
CodeT5	29.61	14.38	26.72	59.51	19.37	3.12	2.43	2.78	3.98
GPT-4o	26.06	3.57	26.18	53.62	18.29	3.23	2.38	2.61	4.13
GPT-3	19.25	4.42	20.72	51.26	13.37	2.64	2.19	2.16	2.89
Llama3	19.57	2.07	20.91	52.41	13.43	3.06	2.38	2.72	3.88
CodeLlama3	19.63	2.46	21.18	52.85	15.72	2.35	1.98	2.06	2.61
Ablation Study									
CodeT5 without table	22.40	9.54	20.49	55.91	18.52	2.55	2.06	2.17	3.74
CodeT5 without code	26.35	11.72	24.01	57.93	19.14	3.12	2.31	2.59	3.83
CodeT5 without guideline description	25.09	10.78	22.61	56.63	18.75	3.07	2.99	2.56	3.90
GPT-4o with chain of thoughts	24.91	3.82	25.42	51.25	16.91	3.22	2.87	2.69	3.84
GPT-4o with in context leaning	23.64	3.52	24.62	49.10	15.92	3.03	2.81	2.54	3.86

Table 3: ROUGE scores and BERTScore for the baselines, our model, and the ablation studies. Results show that this task is challenging though we use it in the state-of-art text generation models.

while the fine-tuned CodeT5 obtained the best performance. It is interesting to note that CodeLlama3 has bad performance in this task. The possible reason is that CodeLlama3 focuses mainly on code generation and is not suitable for description generation. As shown in Table 3, CodeT5-Large outperforms the GPT-4o, GPT-3, Llama3, and CodeLlama3 in this task. Additionally, we notice that the ROUGE-L of CodeT5-Large is above 25, and the ROUGE-2 is around 15, indicating that our dataset can produce more accurate and fluent text in response to different guidelines in this task. However, these metrics primarily check for semantic similarity. To further evaluate generated descriptions in diverse dimensions and achieve higher human correspondence, we conduct LLM-based evaluators G-EVAL (Liu et al., 2023b) in four dimensions (Coherence, Correctness, Consistency, and Fluency) on a 5-point Likert Scale. Details of G-EVAL are shown in Appendix H. we can find that CodeT5-Large outperforms the GPT-4o, GPT-3, Llama3, and CodeLlama3 in this task. It is interesting to note that existing LLM can generate fluent descriptions but struggle to produce coherent and consistent descriptions.

Ablation Study: To better understand the impact of each component on this new task, we perform ablation studies(Table 3) to evaluate how table, code, and guideline description contribute to the model performance separately. More concretely, we generate ablation models with the following settings: (1) without table, (2) without code, (3) without guideline description, (4) chain of thought prompting on GPT-4o, (5) in-context learning on GPT-4o.

Since CodeT5 performs best in the task, we use it as a backbone to test its performance without code, table, and guideline description. In general, all the elements contribute to the performance, and removing one element will lead to a significant performance drop.

Note that table content has a bigger effect on model performance compared to code. Code also influences performance by providing the necessary context to infer the logic, which aids in interpreting the table's content accurately. Guideline description can be seen as a synergy of tables that guide the generation system to generate desirable topics, and without it, the performance is slightly higher than one without any table content.

One intuitive method to enhance the reasoning ability of LLMs is Chain-of-Thoughts (CoT). Here we want to further answer this question: using CoT, can a large language model automatically find an optimum guideline and generate summaries better aligned with human interests? CoT is well known to work well for GPT models, so we experimented on GPT-4o with a CoT prompt containing both an example and middle steps of guessing a guideline (prompts shown in Appendix F). For a fair comparison, we also added the performance of in-context-learning for GPT-4o, by removing the provided guideline and directly providing the example (prompts shown in Appendix G). The result of CoT improved over in-context learning but is still inferior to the performance of the original GPT-4o with ground-truth guidelines (except ROUGE-2).

Then we analyzed the match rate between guidelines generated through the CoT process and the ground truth. Results show that 72% of the guidelines did not match. Thus, although LLM can often generate readable and decent descriptions for code and table(see results in Table 5 and Table 3), most of the generated descriptions are not as the users expected (see result in Orientation dimension in Table 5). This demonstrates the necessity of guidelines. To fairly compare the generation models and standardize evaluation, we need to specify our guidelines for generating descriptions.

Pyramid Evaluation: We employ an automatic evaluation method based on pyramid evaluation (PyrEval) (Gao et al., 2019) to assess the faithfulness of generation. This metric, which correlates well with human evaluation, extracts and filters key phrases to preserve factual information, offering a more reliable metric to ROUGE. The results in Table 3 show consistent trends between PyrEval and ROUGE, validating the impact of table, code, and guideline descriptions in our ablation studies.

5.3 Human Evaluation

We conducted a human evaluation with 10 participants to assess the performance of various LLM models in generating code documentation. Participants rated 50 pairs of code and documentation across three dimensions: Correctness, Orientation, and Readability. The results showed that the Ground Truth outperformed other baselines. Results and analysis are deferred to Appendix A.

6 Error Analysis

In this section, we analyze some common error cases in this guideline-based text generation task. Some examples can be found in the Appendix B.

(1) Variable values were generated and matched incorrectly. As shown in the example in Table 4, even though CodeT5-Large, GPT-4o, GPT-3, Llama3, and CodeLlama3 are capable of generating keywords such as ”highest” based on the ”Extreme” guideline, it remains difficult to produce accurate text content based on the variables in the table. For example, CodeT5-Large incorrectly predicted the oldest passenger as 80 years old. Table 4 also has this kind of error.

(2) The generated text focuses solely on the table and ignores important information in the code. In the example from Table 10, ground truth is in the “Extreme” guideline and tends to convey that the first red wine has the highest pH value. However, the table does not have a related keyword “red wine.” And CodeT5 failed to extract this information and also extracted the wrong value. GPT-4o cannot generate correct description either. Example from Table 8 also has this kind of error.

(3) Generating incorrectly oriented text based on guidelines. For example, GPT-3, Llama3, and CodeLlama3 produce text related to “Difference”, “Extreme”, or “Distribution”, but not “Trend” in the example from Table 7. Another example in Table 8, requires models generating text related to “Goal”, but GPT-3 generates text related to “Association”, describing the relationship between SibSP and Parch. Llama3 also generates text in “Feature Engineer” but not “Aggregation” in the example from Table 6.

Guideline category description

Extreme: Identify the data cases that are the most extreme in relation to the data attributes or within a specific range

Code Cells

```
cols2 = X_test.columns.tolist() # List of column names
X_test = X_test[cols2] # Applying the new order
X_test
```

Table

Passenger	Sex	Age	Embarked	FamilySize
0	0	34.5	1	1
1	1	47.0	2	2
2	0	62.0	1	1
3	0	27.0	2	1
4	1	22.0	2	3

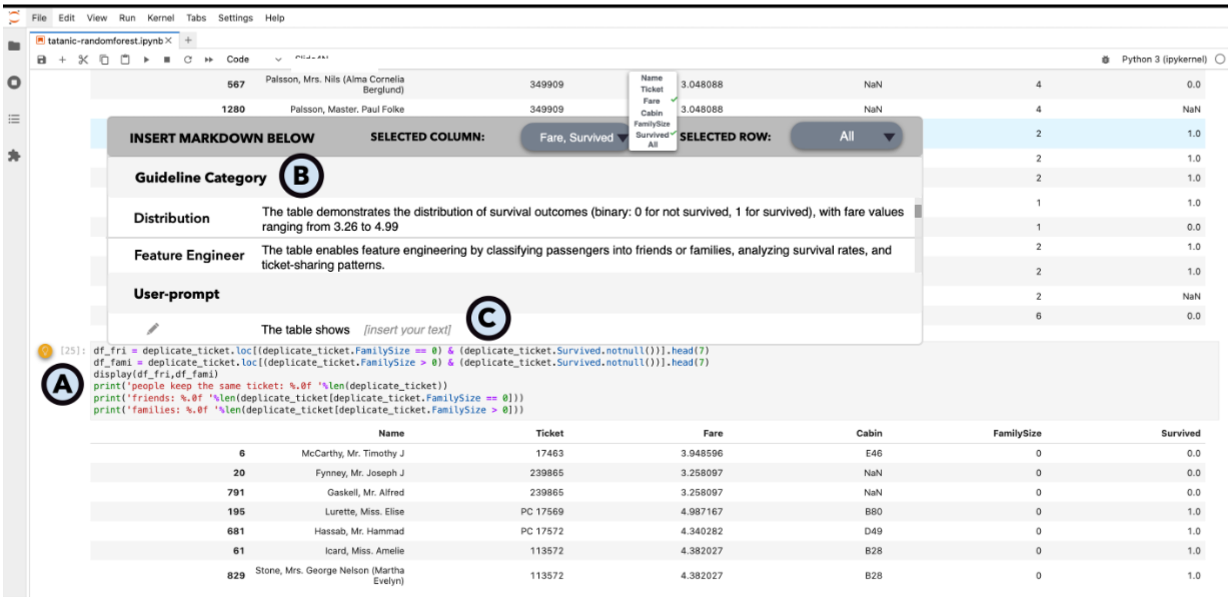
Description

ground truth	The oldest passenger in X_test dataset is 62 years old
CodeT5-Large	Oldest person in the titanic was 80 years old and youngest person was less than one year
GPT-4o	It shows the initial rows of the table providing insights into the age distribution of passengers, with the label drawing attention to the oldest passenger
GPT-3	The oldest passenger is a man in his fifties
Llama3	The ages of passengers ranging from 27 to 62 years old with the youngest passenger being 27 years old
CodeLlama	The average age of the passengers is 34.5 years old.

Table 4: An example of code-table cells and different models. Note that the table above is the original table, which will be processed by the table curation in Section 4.2 when generating the description. In this case, the curated table only keeps the passenger and age columns.

(4) Reasoning error. If operating under Aggregation guidelines, CodeT5, GPT-4o, and GPT-3 may generate incorrect Aggregation data (count, mean, sum). In this example (Table 6), GPT-3 can generate text such as this feature has many null values, but cannot obtain the count of null values.

We manually check 50 examples of CodeT5, GPT-4o, GPT-3, Llama3, and CodeLlama3 models used in our user study and label the type of errors made. The most errors are made when they generate incorrectly oriented text (3rd type) **(54.1%)**.



Guideline Category

Distribution	The table demonstrates the distribution of survival outcomes (binary: 0 for not survived, 1 for survived), with fare values ranging from 3.26 to 4.99
Feature Engineer	The table enables feature engineering by classifying passengers into friends or families, analyzing survival rates, and ticket-sharing patterns.

User-prompt

The table shows *[insert your text]*

```
[25]: df_fri = deplicate_ticket.loc[(deplicate_ticket.FamilySize == 0) & (deplicate_ticket.Survived.notnull())].head(7)
df_fami = deplicate_ticket.loc[(deplicate_ticket.FamilySize > 0) & (deplicate_ticket.Survived.notnull())].head(7)
display(df_fri,df_fami)
print('people keep the same ticket: %.0f '%len(deplicate_ticket))
print('friends: %.0f '%len(deplicate_ticket[deplicate_ticket.FamilySize == 0]))
print('families: %.0f '%len(deplicate_ticket[deplicate_ticket.FamilySize > 0]))
```

	Name	Ticket	Fare	Cabin	FamilySize	Survived
6	McCarthy, Mr. Timothy J	17463	3.948596	E46	0	0.0
20	Fynney, Mr. Joseph J	239865	3.258097	NaN	0	0.0
791	Gaskell, Mr. Alfred	239865	3.258097	NaN	0	0.0
195	Lurette, Miss. Elise	PC 17569	4.987167	B80	0	1.0
681	Hassab, Mr. Hammad	PC 17572	4.340282	D49	0	1.0
61	Icard, Miss. Amelie	113572	4.382027	B28	0	1.0
829	Stone, Mrs. George Nelson (Martha Evelyn)	113572	4.382027	B28	0	1.0

Figure 2: We implement a downstream application as a Jupyter Notebook plugin (A) to assist users in documentation writing for selected rows and columns from different guideline categories (B) and a user-prompt approach (C).

This is because the model tends to generate documents related to the best-trained guideline type in the dataset, such as "Association" or "Value". There are also two common errors made by generating documents with wrong values (1st type) and wrong reason (4th type) (**27%** respectively). Such errors are commonly made by generating "Value" or "Aggregation" type documents. There are also **13.5%** errors made by generating documents without considering the code. The dataset has numerous examples with insufficient code, leading the model to overlook some code instances.

These errors show that our task and dataset provide some challenges for existing foundation models. We firmly believe that researchers can enhance the existing foundation models in the future when they address the challenges. By building on our work and leveraging the valuable insights gained from it, they can push the boundaries even further, contributing to the continuous evolution of foundation models.

7 Downstream User Application

To demonstrate the application of our benchmark NBDESCRIB, we designed a Jupyter Notebook plugin to assist document writing in data science programming (as shown in Figure 2).

The plugin is triggered when detecting users focusing on a code cell (Figure 2.A). The plugin then reads the contents from the focused cell and its curated table output based on the selected columns and rows and sends the content to the backend. The backend server first generates documentation under different guideline categories using fine-tuned CodeT5-Large(Figure 2.B). In addition, we used prompts to nudge users to explain an output (Figure 2.C). If the user selects one of his preferred candidates, the chosen documentation will be inserted below the code cell.

Our plugin went through several rounds of pilot testing and iterative design. Participants found that it reminded them to document code they would have ignored, reducing the time spent developing documentation while actively exploring the data science task. Full implementation details and a formal evaluation of the human-AI collaboration for automatic documentation will be reported separately after the paper is accepted.

8 Conclusion

In this paper, we formulated a new task, TCDG, that aimed to automatically generate descriptive text for code and table based on the given guidelines for a computational notebook. We collected a large amount of well-documented Jupyter Notebooks from Kaggle, resulting in a new benchmark dataset **NBDESCRIB**. From our analysis, our task imposed unique challenges to the current generation methods, including CodeT5 and LLMs. This dataset facilitated the creation of practical slides for Jupyter notebooks and enabled evaluations on faithful, high-fidelity, and factual generation.

Limitations and Potential Risk

As annotations are often performed by multiple individuals, there may be a degree of subjectivity and bias in guidelines and datasets used for text generation. As a result, text can be generated that does not reflect a diverse range of perspectives. Furthermore, although we have automatic evaluation metrics such as ROUGE and BERTScore, the correctness of the generated texts is primarily evaluated through human evaluation, which is accurate but not efficient. Future research should focus on developing methods for automatically evaluating the factual correctness of the generated texts, in order to ensure that the generated text is accurate, unbiased, and representative of a diverse range of perspectives.

One potential risk involves the substantial computational resources needed to run state-of-the-art language models. These resources consume significant amounts of energy, which not only raises the carbon footprint of such research but also leads to environmental degradation.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668, Online. Association for Computational Linguistics.

Chenxin An, Jiangtao Feng, Kai Lv, Lingpeng Kong, Xipeng Qiu, and Xuanjing Huang. 2022. Cont: Contrastive neural text generation. *Advances in Neural Information Processing Systems*, 35:2197–2210.

Souti Chattopadhyay, Ishita Prasad, Austin Z Henley, Anita Sarma, and Titus Barik. 2020. What's wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12.

David L Chen and Raymond J Mooney. 2008. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, pages 128–135.

Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. 2020. Logical natural language generation from open-domain tables. *arXiv preprint arXiv:2004.10404*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164*.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2021. Hitab: A hierarchical table dataset for question answering and natural language generation. *arXiv preprint arXiv:2108.06712*.

Harika Cm, Debasmita Das, Rajesh Kumar Ranjan, Siddhartha Asthana, et al. 2023. Loki: Money laundering report generation via logical table-to-text using meta learning. In *Proceedings of the Fifth Workshop on Financial Technology and Natural Language Processing and the Second Multimodal AI For Financial Forecasting*, pages 104–110.

Hoa Trang Dang. 2005. Overview of duc 2005. In *Proceedings of the document understanding conference*, volume 2005, pages 1–12. Citeseer.

Bhuwan Dhingra, Manaal Faruqui, Ankur Parikh, Ming-Wei Chang, Dipanjan Das, and William W Cohen. 2019. Handling divergent reference texts when evaluating table-to-text generation. *arXiv preprint arXiv:1906.01081*.

Haijie Ding and Xiaolong Xu. 2023. San-t2t: An automated table-to-text generator based on selective attention network. *Natural Language Engineering*, pages 1–25.

Shubhang Shekhar Dvivedi, Vyshnav Vijay, Sai Leela Rahul Pujari, Shoumik Lodh, and Dhruv Kumar. 2024. A comparative analysis of large language models for code documentation generation. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, pages 65–73.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Yanjun Gao, Chen Sun, and Rebecca J. Passonneau. 2019. Automated pyramid summarization evaluation. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 404–418, Hong Kong, China. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Zhixin Guo, Jianping Zhou, Jiexing Qi, Mingxuan Yan, Ziwei He, Guanjie Zheng, Zhouhan Lin, Xinbing

Wang, and Chenghu Zhou. 2024. Towards controlled table-to-text generation with scientific reasoning. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9951–9955. IEEE.

Vivek Gupta, Maitrey Mehta, Pegah Nokhiz, and Vivek Srikumar. 2020. INFOTABS: Inference on tables as semi-structured data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2309–2324, Online. Association for Computational Linguistics.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Xue Jiang, Zhuoran Zheng, Chen Lyu, Liang Li, and Lei Lyu. 2021. Treebert: A tree-based pre-trained model for programming language. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 54–63. PMLR.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2019. Pre-trained contextual embedding of source code. *ArXiv*, abs/2001.00059.

Junaed Younus Khan and Gias Uddin. 2022. Automatic code documentation generation using gpt-3. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–6.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213, Austin, Texas. Association for Computational Linguistics.

Tongliang Li, Lei Fang, Jian-Guang Lou, and Zhoujun Li. 2021. Twt: Table with written text for controlled data-to-text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1244–1254.

Percy Liang, Michael Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 91–99, Suntec, Singapore. Association for Computational Linguistics.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Sherry Lin, Winthrop F Gillis, Caleb Weinreb, Ayman Zeine, Samuel C Jones, Emma M Robinson, Jeffrey Markowitz, and Sandeep Robert Datta. 2022. Characterizing the structure of mouse behavior using motion sequencing. *arXiv preprint arXiv:2211.08497*.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2018. Table-to-text generation by structure-aware seq2seq learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Xuye Liu, Dakuo Wang, April Wang, Yufang Hou, and Lingfei Wu. 2021. HAConvGNN: Hierarchical attention based convolutional graph neural network for code documentation generation in Jupyter notebooks. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4473–4485, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023a. G-eval: Nlg evaluation using gpt-4 with better human alignment. *Preprint*, arXiv:2303.16634.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, et al. 2024. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667*.

Dehai Min, Nan Hu, Rihui Jin, Nuo Lin, Jiaoyan Chen, Yongrui Chen, Yu Li, Guilin Qi, Yun Li, Nijun Li, et al. 2024. Exploring the impact of table-to-text methods on augmenting llm-based question answering with domain hybrid data. *arXiv preprint arXiv:2402.12869*.

Tamal Mondal, Scott Barnett, Akash Lal, and Jyothi Vedurada. 2023. Cell2doc: Ml pipeline for generating documentation in computational notebooks. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 384–396. IEEE.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. 2022. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Jekaterina Novikova, Oliver Lemon, and Verena Rieser. 2016. Crowd-sourcing NLG data: Pictures elicit better data. In *Proceedings of the 9th International Natural Language Generation conference*, pages 265–273, Edinburgh, UK. Association for Computational Linguistics.

Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020a. Totto: A controlled table-to-text generation dataset. *arXiv preprint arXiv:2004.14373*.

Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020b. Totto: A controlled table-to-text generation dataset. *arXiv preprint arXiv:2004.14373*.

Bumjin Park and Jaesik Choi. 2024. Identifying the source of generation for large language models. *arXiv preprint arXiv:2407.12846*.

Panupong Pasupat and Percy Liang. 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)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Dhivyabharathi Ramasamy, Cristina Sarasua, Alberto Bacchelli, and Abraham Bernstein. 2023. Visualising data science workflows to support third-party notebook comprehension: an empirical study. *Empirical Software Engineering*, 28(3):58.

Kyle Richardson, Sina Zarrieß, and Jonas Kuhn. 2017. The code2text challenge: Text generation in source code libraries. *arXiv preprint arXiv:1708.00098*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Adam Rule, Aurélien Tabard, and James D Hollan. 2018a. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12.

Adam Rule, Aurélien Tabard, and James D. Hollan. 2018b. Exploration and explanation in computational notebooks. CHI '18, page 1–12, New York, NY, USA. Association for Computing Machinery.

Lya Hulliyyatus Suadaa, Hidetaka Kamigaito, Kotaro Funakoshi, Manabu Okumura, and Hiroya Takamura. 2021. Towards table-to-text generation with numerical reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1451–1465, Online. Association for Computational Linguistics.

Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. Intellicode compose: Code generation using transformer. ESEC/FSE 2020, page 1433–1443, New York, NY, USA. Association for Computing Machinery.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

April Yi Wang, Dakuo Wang, Jaimie Drozdal, Xuye Liu, Soya Park, Steve Oney, and Christopher Brooks. 2021a. What makes a well-documented notebook? a case study of data scientists' documentation practices in kaggle. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7.

April Yi Wang, Dakuo Wang, Jaimie Drozdal, Xuye Liu, Soya Park, Steve Oney, and Christopher A. Brooks. 2021b. What makes a well-documented notebook? a case study of data scientists' documentation practices in kaggle. *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*.

April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. 2022. Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks. *ACM Trans. Comput.-Hum. Interact.*, 29(2).

Hao Wang, Xiaodong Zhang, Shuming Ma, Xu Sun, Houfeng Wang, and Mengxiang Wang. 2018. A neural question answering model based on semi-structured tables. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1941–1951, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Yun Wang, Zhida Sun, Haidong Zhang, Weiwei Cui, Ke Xu, Xiaojuan Ma, and Dongmei Zhang. 2020. Datashot: Automatic generation of fact sheets from tabular data. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):895–905.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Robert F Woolson. 2007. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. *arXiv preprint arXiv:2301.13808*.

Shafiq Joty Steven C.H. Hoi Yue Wang, Weishi Wang. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*.

Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *arXiv preprint arXiv:2001.06684*.

Haowei Zhang, Shengyun Si, Yilun Zhao, Lujing Xie, Zhijian Xu, Lyuhao Chen, Linyong Nan, Pengcheng Wang, Xiangru Tang, and Arman Cohan. 2024. Opent2t: An open-source toolkit for table-to-text generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 259–269.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Yilun Zhao, Zhenting Qi, Linyong Nan, Lorenzo Jaime Yu Flores, and Dragomir Radev. 2023a. Loft: enhancing faithfulness and diversity for table-to-text generation via logic form control. *arXiv preprint arXiv:2302.02962*.

Yilun Zhao, Zhenting Qi, Linyong Nan, Boyu Mi, Yixin Liu, Weijin Zou, Simeng Han, Ruizhe Chen, Xiangru Tang, Yumo Xu, et al. 2023b. Qtsumm: Query-focused summarization over tabular data. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

A Appendix: Human Evaluation

We also conduct a human evaluation to further evaluate whether those models can generate reasonable and oriented text with our dataset.

Participants: Our human evaluation task involves reading the code snippet, its output table, and a guideline description and rating the generated documentation from them. We recruited 10 participants (6 male, 4 female) who are fluent English speakers with around six years of experience in the data science and machine learning field. We conducted a rigorous qualification process, evaluating their knowledge of coding practices and data analysis, to ensure high-quality annotations. We hired them by sending invited emails to graduate students who have experience in data science work. We allocated up to 90 minutes for each participant to complete the study, and for their valuable time and input, each participant received a compensation of $20.

Task: We randomly selected 50 pairs of documentation and code from NBDESCRIB. Note that each pair has only one code, one table, and one guideline description, but may have one descriptive text. Each participant is assigned 50 pairs. Each pair is evaluated by 10 individuals. In each trial, a participant reads 8 candidate descriptions for the same code snippet-table-guideline: one by GPT-4o with chain-of-thought, one by GPT-4o with in-context-learning, one as the ground truth, and another five by five different models. The order of these eight is also randomized, so participants do not know which descriptive text is from which model. The participant is asked to rate the 8 documentation texts along three dimensions using a five-point Likert scale from -2 to 2.

- *Correctness*: The generated description matches the code and table content.
- *Orientation*: The generated description is written in the correct guideline category.
- *Readability*: The generated description is in readable English grammar and words.

Evaluation Results: We conducted *Wilcoxon tests* (Woolson, 2007) with a significance level of 0.05 to compare the performance of Ground Truth against CodeT5-Large, GPT-3, and GPT-4o in the Correctness, Orientation, and Readability dimensions. The Wilcoxon test is a non-parametric statistical test used to compare two paired groups of data. The obtained p-values indicate the probability of observing the reported differences if there were no true differences between the models. The results indicate significant differences in the Correctness dimension, where Ground Truth outperforms CodeT5-Large (V = 5628, p = 1.74e-30), GPT-3 (V = 5635, p = 5.46e-31), GPT-4o (V = 5639, p = 2.84e-30), Llama3 (V = 5732, p = 3.32e-30), and CodeLlama (V = 5948, p = 1.27e-30). It is also worth noting that CodeT5 performs slightly better than GPT-4o in terms of correctness from Table 5, possibly because it handles code-containing data sets better.

Similarly, in the Orientation dimension, Ground Truth surpasses CodeT5-Large (V = 3567, p = 1.59e-20), GPT-3 (V = 3731, p = 1.77e-20), GPT-4o (V = 3675, p = 1.64e-20), Llama3 (V = 3722, p = 1.89e-20), and CodeLlama (V = 3883, p = 1.33e-20).

For the Readability dimension which considers whether the generated description is a valid English sentence, Ground Truth outperforms all models once again: CodeT5-Large (V = 4363, p = 1.40e-7), GPT-3 (V = 4030, p = 3.81e-14), GPT-4o (V = 4135, p = 2.81e-10), Llama3 (V = 4556, p = 2.38e-13), and CodeLlama (V = 4573, p = 1.62-15). It is also worth noting that GPT-4o with chain-of-thought and in-context-learning have worse performance than GPT-4o which demonstrates that guidelines can better assist the description generation for code and table.

The statistically significant p-values (all below 0.05) in each dimension demonstrate it is difficult to meet the correctness, orientation, and readability requirements of the user due to the difficulty of the task. Future work can be accomplished by designing an innovative model to address this challenge.

Model	**Correctness**	**Orientation**	**Readability**
Groundtruth	$\overline{x}$ = 1.19, σ=1.32	$\overline{x}$ = 1.45, σ=1.02	$\overline{x}$ = 1.61, σ=0.78
CodeT5-Large	$\overline{x}$ = -0.43, σ=1.55	$\overline{x}$ = 1.27, σ=1.11	$\overline{x}$ = 0.55, σ=1.60
GPT-4o	$\overline{x}$ = -0.42, σ=1.49	$\overline{x}$ = 1.15, σ=1.24	$\overline{x}$ = 0.53, σ=1.72
GPT-3	$\overline{x}$ = -0.41, σ=1.58	$\overline{x}$ = 0.98, σ=1.39	$\overline{x}$ = 0.51, σ=1.61
Llama3	$\overline{x}$ = -0.44, σ=1.41	$\overline{x}$ = 1.03, σ=1.42	$\overline{x}$ = 0.51, σ=1.82
CodeLlama	$\overline{x}$ = -0.61, σ=1.39	$\overline{x}$ = 0.88, σ=1.36	$\overline{x}$ = 0.52, σ=1.69
GPT-4o with chain-of-thought	$\overline{x}$ = -0.39, σ=1.54	$\overline{x}$ = 0.94, σ=1.35	$\overline{x}$ = 0.48, σ=1.66
GPT-4o with in-context-learning	$\overline{x}$ = -0.35, σ=1.60	$\overline{x}$ = 0.91, σ=1.28	$\overline{x}$ = 0.46, σ=1.83

Table 5: Human Evaluation Result.

B Appendix: Guideline-Code snippets-Table-Documentation Pair Examples

Guideline description

Aggregation: Calculate the descriptive statistical indicators (e.g., average, sum, count, etc.) based on the data attributes

Code Cells

```
for dataset in [titanic_train,titanic_test]:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
titanic_train.head(3)
```

Table

	PassengerId	Survived	Sex	Ticket	Cabin
0	1	0	male	A/5 21171	NaN
1	2	1	female	PC 17599	C85
2	3	1	female	STON/O2.3101282	NaN

Documentation

ground truth	Cabin feature has 2 missing values
CodeT5-Large	These five passengers are in the same cabin
GPT-4o	It adds an IsAlone column to the Titanic datasets and displays the first three rows data
GPT-3	The cabin column has many null values
Llama3	Creates a new column to indicate if passengers are traveling alone or with family.
CodeLlama	Average fare paid by the passengers

Table 6: An example of code and its table output pair(cabin is the Cabin number). In this case, the curated table only keeps the cabin columns.

Guideline description

Trend: Indicates a general tendency over a period of time

Code Cells

```
train[['Pclass', 'Survived']].groupby(['Pclass'],
↪   as_index=False).mean().sort_values(by='Survived',
↪   ascending=False)
```

Table

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

Documentation

ground truth	Higher class survived more
CodeT5-Large	The survived rate for this new feature varies whether it is a tend to be a higher class passengers or a very high class passengers
GPT-4o	it presents the survival rates for each passenger class, demonstrating a clear pattern where lower class numbers correspond to higher survival rates
GPT-3	the survived rate for the pclass 1 is higher than the pclass 2
Llama3	First-class passengers having the highest survival rate and third-class passengers having the lowest.
CodeLlama	The most common class is 3rd class with 24% of the survived passengers

Table 7: An example of code and its table output pair (Pclass is the ticket level. The smaller the value, the higher the class). In this case, the table is not curated because ground truth have class and survived variables.

Guideline description

Goal: Express user's goal. To say what value or function they tend to use for the later research

Code Cells

```
for dataset in [titanic_train,titanic_test]:
    dataset['FamilySize'] = dataset['SibSp'] +
    ↪  dataset['Parch'] + 1
titanic_train.head(3)
```

Table

	PassengerId	Survived	SibSp	Parch	IsAlone	FamilySize
0	1	0	1	0	0	2
1	2	1	1	0	0	2
2	3	1	0	0	1	2

Documentation

ground truth	Checking if the person is alone or with a family by checking the SibSp and Parch column in Titanic passenger data and add a FamilySize column in titanic_train and titanic_test datasets
CodeT5-Large	We can create another feature called IsAlone
GPT-4o	The goal is to create a new attribute called family size in both the titanic train and titanic test datasets
GPT-3	we can see that sib sp and parch are highly correlated
Llama3	Understand the relationships between different passenger characteristics in the Titanic dataset
CodeLlama	To predict the family size of a passenger in the Titanic disaster

Table 8: An example of code and its table output pair. In this case, the curated table only keeps the passenger, SibSp, Parch, and FamilySize column.

Guideline description

Extreme: Identify the data cases that are the most extreme in relation to the data attributes or within a specific range

Code Cells

```
Tuned_rf = tune_model(rf)
```

Table

Model	Accuracy	AUC	Recall	...
0	0.7895	0.8864	0.6250	...
1	0.9474	1.000	0.8750	...
2	0.8947	0.9318	0.8750	...
3	0.7368	0.8523	1.0000	...
4	0.8947	0.8667	0.8889	...
5	0.9473	0.9444	0.8889	...
6	0.8947	0.9111	0.7778	...
7	0.7895	0.8333	0.6667	...
Mean	0.8617	0.9189	0.8222	...
SD	0.0675	0.0556	0.1348	...

Documentation

ground truth	Model 1 has the highest accuracy while the code tune with random forest
CodeT5-Large	The highest accuracy is 0.8442
GPT-4o	It shows the accuracy values for different sequence numbers where the mean accuracy is 0 8617 and the standard deviation is 0 0675
GPT-3	the highest accuracy is 0.7895
Llama3	Identify the most extreme data points in relation to their corresponding data attributes
CodeLlama	The Random Forest model is the best model for this dataset.

Table 9: An example of code and its table output pair. In this case, the curated table only keeps the Accuracy and Model columns.

Guideline description

Extreme: Identify the data cases that are the most extreme in relation to the data attributes or within a specific range

Code Cells

```
df =
↪  pd.read_csv("../input/red-wine-quality-cortez-et-al-2009/
winequality-red.csv")
df.head()
```

Table

Wine	fixed acidity	volatile acidity	pH	sulphates	alcohol	quality
0	7.4	0.70	4o1	0.56	9.4	5
1	7.8	0.88	3.20	0.68	9.8	5
2	7.8	0.76	3.26	0.65	9.8	5
3	11.2	0.28	3.16	0.58	9.8	5
4	7.4	0.70	4o1	0.56	9.4	5

Documentation

ground truth	The first red wine has the highest pH value
CodeT5-Large	the biggest ph is 3.20
GPT-4o	It determine how these values deviate significantly from the normal range, providing insights into outliers.
GPT-3	data frame sort ph values
Llama3	Identifies the top 4 most extreme data points for each attribute.
CodeLlama	The wine quality is a continuous variable with a range of 3 to 9

Table 10: An example of code and its table output pair. In this case, the curated table only keeps the Wine and pH column.

C Appendix: Kaggle competition link

We crawled highly voted notebooks from seven top popular Kaggle competitions - House Price Prediction[2], Titanic Survival Prediction[3], Predict Future Sales[4], Spaceship Titanic[5], U.S. Patent Phrase to Phrase Matching[6], JPX Tokyo Stock Exchange Prediction[7], Ubiquant Market Prediction[8]

D Appendix: Detail of Baseline Models

CodeT5 is a large pre-trained encoder-decoder Transformer model that better leverages the code semantics conveyed from the developer-assigned identifiers. Since CodeT5 is a competitive code-related text generation model, when using this model in our task, we converted the relevant table and guideline category description into an inline comment in code and then fine-tuned the model. It has 770 million parameters and the computational budget is around 3 hours.

GPT-3 (Generative Pre-training Transformer 3) is an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model. To use the GPT3 model for our task, we combine guideline description, code, and table as input text. It has 175 billion parameters. The computational budget is around 1 hour. To use the GPT-3 model, we register an account on OpenAI and use the related API (openai api fine_tunes.create[9]) to fine-tune the GPT-3 model. Also, we built a dataset suitable for GPT-3 training, which can shared with the public.

GPT-4o is an advanced iteration of the GPT-3 model with around 12 billion parameters and a default backend of free ChatGPT. The computational budget is around 1 hour and 15 minutes. Its ability to comprehend context, generate coherent and contextually relevant responses, and perform a wide array of language-related tasks is further refined. It is an easily accessible tool and has been widely used in real life. So we add it as an advanced baseline.

Llama3 (Touvron et al., 2023). We use Llama3 3.1-70B in this task. It is an advanced language model with approximately 70 billion parameters. Its default backend is designed for efficiency and scalability. The computational budget for Llama3 is approximately 1 hour and 30 minutes. Its ability to understand context, generate coherent and contextually relevant responses, and perform a wide range of language-related tasks is significantly enhanced. Llama3 is a powerful and accessible tool, widely used in various applications. Therefore, it is included as an advanced baseline.

CodeLlama3 (Roziere et al., 2023) is an advanced version of the Llama3 model. We use the 13B version in this task. The model aims to handle complex code generation and comprehension tasks efficiently. The computational budget of CodeLlama3 is about 2 hours. CodeLlama3 performs well in understanding intricate programming contexts, generating accurate and contextually appropriate code, and performing various code-related tasks. Its accessibility and versatility have made it a valuable tool for developers and researchers, serving as an advanced baseline for code-centric applications.

E Appendix: Guideline Categories

---

[2] https://www.kaggle.com/c/house-prices-advanced-regression-techniques

[3] https://www.kaggle.com/c/titanic/

[4] https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales

[5] https://www.kaggle.com/competitions/spaceship-titanic

[6] https://www.kaggle.com/competitions/us-patent-phrase-to-phrase-matching

[7] https://www.kaggle.com/competitions/jpx-tokyo-stock-exchange-prediction

[8] https://www.kaggle.com/competitions/ubiquant-market-prediction

[9] https://beta.openai.com/docs/guides/fine-tuning

F Appendix: Prompt for doing chain of thought on GPT-4o

```
Given the 15 guidelines describing the code cell and its table output in the Jupyter Notebook:
1. Value(Get the exact data attribute values for a set of criteria)
2. Difference(A comparison between at least two distinct attributes within the target object, or a comparison between the target object and previously measured values)
3. Trend(Indicates a general tendency over a period of time)
4. Proportion(Measure the proportion of selected data attribute(s) within a specified set )
5. Categorization(Select the data attribute(s) that meet the condition)
6. Distribution(Show the amount of shared value for the selected data attributes or present a breakdown of all data attributes)
7. Rank(Sort data attributes by their values and display a breakdown of selected attributes)
8. Association (Identify the useful relationship between two or more data attributes)
9. Extreme(Identify the data cases that are the most extreme in relation to the data attributes or within a specific range )
10. Outlier(Determine whether there are unexpected data attributes or statistically significant outliers)
11. Aggregation(Calculate the descriptive statistical indicators (e.g., average, sum, count, etc. ) based on the data attributes.)
12. Goal(Express user's goal. To say what value or function they tend to use for the later research)
13. Reason(Express reason using the data from the table or explain the reasons why certain functions are used or why a task is performed.)
14. Feature Engineer(The process of selecting, transforming, extracting, combining, and manipulating raw data to generate the desired variables for analysis or predictive modeling)
15. Other(Other description providing supplementary details)

Q: When using Jupyter Notebook, the data scientist wants to write a description in the Markdown cell covering the code cell and its table output. The description should be less than 50 tokens.
Table Sequence:
    | passengerid| survived
mean| 446.000000 | 0.383838
Code: train = pd.read_csv(”../input/titanic/train.csv”)
    # take a quick look at the training data
    train.describe(include=”all”)”

A: The data scientist wants to write a description in Extreme guideline, the description he writes is: the mean survived rate is 38.3 denoting most of the passengers have not survived

Q: When using Jupyter Notebook, the data scientist wants to write a description in the Markdown cell covering the code cell and its table output:
<Table>
<Code>
```

G Appendix: Prompt for doing in-context learning on GPT-4o

```
Q: When using Jupyter Notebook, the data scientist wants to write a description in the Markdown cell covering the code cell and its table output.
Table Sequence:
          | passengerid| survived
      mean| 446.000000| 0.383838
train = pd.read_csv(”../input/titanic/train.csv”)
# take a quick look at the training data
train.describe(include=”all”)”

A: The data scientist wants to write a description in Extreme guideline, the description he writes is: the mean
```

Guideline	N	Description	Example
Value	286 (7.29%)	Get the exact data attribute values for a set of criteria	The mean survived rate is 38.3 denoting most of the passengers did not survive
Difference	138 (3.52%)	A comparison between at least two distinct attributes within the target object, or a comparison between the target object and previously measured values.	The difference though narrows down considerably if we were to consider groups of 2 woman travelers
Trend	31 (0.79%)	Indicates a general tendency over a period of time.	table is displayed in a descending trend in accuracy
Proportion	120 (3.06%)	Measure the proportion of selected data attribute(s) within a specified set	8 of 10 passengers have parents
Categorization	74 (1.89%)	Select the data attribute(s) that meet the condition.	1 denotes survived while 0 denote not survived
Distribution	127 (3.20%)	Show the amount of shared value for the selected data attributes or present a breakdown of all data attributes.	Fare value range from 7 to 13
Rank	73 (1.86%)	Sort data attributes by their values and display a breakdown of selected attributes.	Selecting the top 3 classifiers for model prediction
Association	165 (4.21%)	Identify the useful relationship between two or more data attributes.	These two passengers are in the same PClass
Extreme	227 (5.78%)	Identify the data cases that are the most extreme in relation to the data attributes or within a specific range	Model 1 has the highest accuracy
Outlier	257 (6.55%)	Determine whether there are unexpected data attributes or statistically significant outliers.	Age column has some missing values
Aggregation	125 (3.19%)	Calculate the descriptive statistical indicators (e.g., average, sum, count, etc.) based on the data attributes.	There are 2 classes in the Deck
Goal	771 (19.64%)	Express user's goal. To say what value or function they tend to use for the later research	We use the Gaussian Process Classifier to plot the confusion matrix
Reason	276 (7.03%)	Express reason using the data from the table or explains the reasons why certain functions are used or why a task is performed.	We go through deleting the column for Cabin deleting 2 rows for Emabarked and since Age plays some role we can ...
Feature Engineer	393 (10.02%)	The process of selecting, transforming, extracting, combining, and manipulating raw data to generate the desired variables for analysis or predictive modeling.	Delete Name and Ticket due to it s high cardinality
Complementary Details	870 (22.17%)	Express additional contextual elements and supporting informa- tion to enhance understanding of the primary content	Column details, counts, and data types provide supplementary technical information about the Titanic passenger dataset structure.

Table 11: We identify 15 guideline categories based on the types of descriptions in the Markdown cells which are below the code whose output is a table.

```
survived rate is 38.3 denoting most of
the passengers have not survived

Q: When using Jupyter Notebook, the data
 scientist wants to write a description
in the Markdown cell covering the code
cell and its table output
<Table>
<Code>
```

H Appendix: G-Eval in Coherence, Consistency, Correctness, and Fluency

H.1 Definition

G-EVAL is a prompt-based evaluation system with three main components: 1) a prompt defining the evaluation task and criteria, 2) a chain-of-thoughts (CoT) that includes intermediate instructions generated by the LLM to outline the detailed evaluation steps, and 3) a scoring function that uses the LLM to calculate scores based on the probabilities of the returned tokens. The prompt should also contain customized evaluation criteria for different NLG tasks.

Coherence The overall quality of all sentences working together. This aligns with the DUC (Dang, 2005) quality question on structure and coherence, which states that "the summary should be well-structured and well-organized, building from sentence to sentence to form a coherent body of information about a topic." Followed by (Liu et al., 2023a), we conduct the G-EVAL in the four dimensions below:

Consistency The generated description is written in the correct guideline category. A factually consistent summary contains only statements that are supported by the source document. Annotators were instructed to penalize summaries containing hallucinated facts.

Fluency The quality of individual sentences. According to the DUC (Dang, 2005) quality guidelines, sentences in the summary "should have no formatting problems, capitalization errors, or obvious grammatical errors (e.g., fragments, missing components) that make the text difficult to read."

Correctness Determine whether the actual output is factually correct based on the expected output.

H.2 Prompt Details for obtaining metrics

```
You will be given one description
written for a code, its table output,
and its guideline category description.
Your task is to rate the description on
one metric. Please make sure you read
and understand these instructions
carefully. Please keep this document
open while reviewing, and refer to it as
 needed.

Evaluation Criteria:
Coherence (1-5) - the collective quality
 of all sentences. We align this
dimension with the DUC quality question
of structure and coherence whereby the
description should be well-structured
and well-organized. The descripition
should not just be a heap of related
information, but should build from
sentence to sentence to a coherent body
of information about a topic.

Evaluation Steps:
1. Read the code, its table output, and
its guideline category description
carefully and identify the main topic
and key points.
2. Read the description and compare it
to the source text including the code,
its table output, and its guideline
category description. Check if the
description covers the main topic and
key points of the source text, and if it
 presents them in a clear and logical
order.
3. Assign a score for coherence on a
scale of 1 to 5, where 1 is the lowest
and 5 is the highest
based on the Evaluation Criteria.

Example:
Source Text:
{{Guideline Category Description} {Code}
 {Table}}
Description:
{{Description}}
Evaluation Form (scores ONLY):
- Coherence:
```

```
You will be given one description
written for a code, its table output,
and its guideline category description.
Your task is to rate the description on
one metric. Please make sure you read
and understand these instructions
carefully. Please keep this document
open while reviewing, and refer to it as
 needed.

Evaluation Criteria:
Correctness (1-5) - Determine whether
the actual output is factually correct
based on the expected output.


Evaluation Steps:
1. Read the code, its table output, and
its guideline category description
carefully and identify the main topic
and key points.
2. Read the description and compare it
to the source text including the code,
its table output, and its guideline
category description. Check if the
description covers the main topic and
key points of the source text, and if it
 presents them in a clear and logical
order.
3. Assign a score for correctness on a
scale of 1 to 5, where 1 is the lowest
and 5 is the highest
based on the Evaluation Criteria.
Example:
Source Text:
{{Guideline Category Description} {Code}
 {Table}}
Description:
{{Description}}
Evaluation Form (scores ONLY):
- Correctness:
```

```
You will be given one description
written for a code, its table output,
and its guideline category description.
Your task is to rate the description on
one metric. Please make sure you read
and understand these instructions
carefully. Please keep this document
open while reviewing, and refer to it as
 needed.

Evaluation Criteria:
Consistency (1-5) - the generated
description is written in the correct
guideline category. A factually
consistent description contains only
statements that are entailed by the
source document. Annotators were also
asked to penalize descriptions that
contained hallucinated facts.

Evaluation Steps:
1. Read the code, its table output, and
its guideline category description
carefully and identify the main topic
and key points.
2. Read the description and compare it
to the source text including the code,
its table output, and its guideline
category description. Check if the
description covers the main topic and
key points of the source text, and if it
 presents them in a clear and logical
order.
3. Assign a score for consistency on a
scale of 1 to 5, where 1 is the lowest
and 5 is the highest
based on the Evaluation Criteria.
Example:
Source Text:
{{Guideline Category Description} {Code}
 {Table}}
Description:
{{Description}}
Evaluation Form (scores ONLY):
- Consistency:
```

```
You will be given one description
written for a code, its table output,
and its guideline category description.
Your task is to rate the description on
one metric. Please make sure you read
and understand these instructions
carefully. Please keep this document
open while reviewing, and refer to it as
 needed.

Evaluation Criteria:
Fluency (1-5) - the quality of
individual sentences. Drawing again from
 the DUC quality guidelines, sentences
in the summary "should have no
formatting problems, capitalization
errors or obviously ungrammatical
sentences (e.g., fragments, missing
components) that make the text difficult
 to read."

Evaluation Steps:
1. Read the code, its table output, and
its guideline category description
carefully and identify the main topic
and key points.
2. Read the description and compare it
to the source text including the code,
its table output, and its guideline
category description. Check if the
description covers the main topic and
key points of the source text, and if it
 presents them in a clear and logical
order.
3. Assign a score for fluency on a scale
 of 1 to 5, where 1 is the lowest and 5
is the highest
based on the Evaluation Criteria.
Example:
Source Text:
{{Guideline Category Description} {Code}
 {Table}}
Description:
{{Description}}
Evaluation Form (scores ONLY):tab:
example4
- Fluency:
```

I Appendix: Table preprocess example

I.1 An example of Original Table Crawled from Kaggle Notebooks

```
<div>\n
    <style scoped>
        \n .dataframe tbody tr th:only-
        of-type {\n vertical-align:
```

```
        middle;\n }\n\n .dataframe tbody
         tr th {\n vertical-align: top;\
        n }\n\n .dataframe thead th {\n
        text-align: right;\n }\n
    </style>\n
    <table border=\ "1\" class=\ "
    dataframe\">\n
        <thead>\n
            <tr style=\ "text-align:
            right;\">\n
                <th></th>\n
                <th>count</th>\n
                <th>mean</th>\n
                <th>std</th>\n
                <th>min</th>\n
                <th>25%</th>\n
                <th>50%</th>\n
                <th>75%</th>\n
                <th>max</th>\n </tr>\n </
                thead>\n
        <tbody>\n
            <tr>\n
                <th>PassengerId</th>\n
                <td>418.0</td>\n
                <td>1100.500000</td>\n
                <td>120.810458</td>\n
                <td>892.00</td>\n
                <td>996.2500</td>\n
                <td>1100.5000</td>\n
                <td>1204.75</td>\n
                <td>1309.0000</td>\n </tr
                >\n
            <tr>\n
                <th>Pclass</th>\n
                <td>418.0</td>\n
                <td>2.265550</td>\n
                <td>0.841838</td>\n
                <td>1.00</td>\n
                <td>1.0000</td>\n
                <td>3.0000</td>\n
                <td>3.00</td>\n
                <td>3.0000</td>\n </tr>\n
            <tr>\n
                <th>Age</th>\n
                <td>332.0</td>\n
                <td>30.272590</td>\n
                <td>14.181209</td>\n
                <td>0.17</td>\n
                <td>21.0000</td>\n
                <td>27.0000</td>\n
                <td>39.00</td>\n
                <td>76.0000</td>\n </tr>\
                n
            <tr>\n
                <th>SibSp</th>\n
                <td>418.0</td>\n
                <td>0.447368</td>\n
                <td>0.896760</td>\n
                <td>0.00</td>\n
                <td>0.0000</td>\n
                <td>0.0000</td>\n
                <td>1.00</td>\n
                <td>8.0000</td>\n </tr>\n
            <tr>\n
                <th>Parch</th>\n
                <td>418.0</td>\n
                <td>0.392344</td>\n
                <td>0.981429</td>\n
                <td>0.00</td>\n
                <td>0.0000</td>\n
                <td>0.0000</td>\n
                <td>0.00</td>\n
                <td>9.0000</td>\n </tr>\n
            <tr>\n
                <th>Fare</th>\n
                <td>417.0</td>\n
                <td>35.627188</td>\n
                <td>55.907576</td>\n
                <td>0.00</td>\n
                <td>7.8958</td>\n
                <td>14.4542</td>\n
                <td>31.50</td>\n
                <td>512.3292</td>\n </tr
                >\n </tbody>\n</table>\n
                </div>"
```

I.2 Table preprocessing from the the original table in H.1

After table preprocessing, table is

```
          | count | mean | std | min |
          25% | 50% | 75% | max|
PassengerId |418.0 |1100.500000
|120.810458 |892.00 |996.2500 |1100.5000
 |1204.75 | 1309.0000|
Pclass |418.0 |2.265550 | 0.841838 |1.00
 | 1.0000 |3.0000 | 3.00 3.0000|
Age |332.0 |30.272590 14.181209 | 0.17 |
 21.0000 | 27.0000 | 39.00 76.0000|
SibSp |418.0 |0.447368 | 0.896760 | 0.00
 | 0.0000 | 0.0000 | 1.00 | 8.0000|
Parch |418.0 |0.392344 | 0.981429 | 0.00
 | 0.0000 | 0.0000 | 0.00 | 9.0000|
Fare |417.0 |35.627188 | 55.907576 |
0.00 | 7.8958 | 14.4542 | 31.50 |
512.3292|
```

I.3 Table Curation from the the table in H.2

In the above example, the ground truth description is: “From the count column, we find that some variables have missing values"". Based on the Table Curation method in Section 4.2, we extract the “count"" keyword from the ground truth description and extract this related column to generate the final table.

```
           |count |
PassengerId |418.0 |
Pclass |418.0 |
Age |332.0 |
SibSp |418.0 |
Parch |418.0 |
Fare |417.0 |
```