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 documentation, has been attracting increasing research attention. However, existing methods for Jupyter Notebooks mostly focused on generating descriptions from code snippets or table outputs solely. On the other side, the descriptions for Jupyter cell should be personalized as users have their own preferences or user-written guidelines while previous work ignores these informative guidelines during description generation. In this work, we formulate a new task, personalized description generation with code, tables, and user-written guidelines in Jupyter Notebooks along with a novel collected new dataset, nbDescrib. Specifically, the proposed benchmark, namely nbDescrib, contains code, tables, and user-written guidelines paired with target personalized descriptions. Extensive experiments show that existing models on text generation, *e.g.*, can generate fluent and readable text as well as different types of text for the same input according to different user-written guidelines. However, they still struggle to produce faithful descriptions that are factually correct. To understand how each component contributes to the generated descriptions, we conduct extensive experiments and show that guidelines significantly enhance model performance, helping users create accurately oriented and reasonable descriptions. Moreover, by analyzing the error patterns of the model-generated text, we found that the most frequent errors involve generating incorrectly oriented text based on the guidelines, with additional common errors related to incorrect value generation and reasoning mistakes. The dataset and processing code will be released until the paper is published.

1 Introduction

Data-to-text generation (Kukich, 1983) is the task of generating a textual description from structured data such as tables and codes (Richardson et al., 2017; Li et al., 2021; Liu et al., 2018a; Parikh et al., 2020a; An et al., 2022). It has been applied to various scenarios, for example, generating sentences based on biographical data (Lebret et al., 2016), basketball game reports based on boxscore statistics (Wiseman et al., 2017), and fact descriptions from Wikipedia's superlative tables (Korn et al., 2019). Moreover, it served as an important testbed for large language models (LLMs) and neural generation models (Bahdanau et al., 2014) for faithful text generation (Koehn and Knowles, 2017; Lee et al., 2018) and models ability of reasoning and numerical inference (Wiseman et al., 2017; Liu et al., 2018b; Pasupat and Liang, 2015).

Literature on data-to-text generation has been focusing on generating text for software code snippets or tables separately (Puduppully and Lapata, 2021; Richardson et al., 2017). While they achieve remarkable performance on benchmarks, they suffer from several major issues, making them suboptimal.

First, in Jupyter Notebooks, one cell (in table code associated Markdown cell) may contain both code and its corresponding table outputs which are useful for generating descriptions. Using only one type of information will make the description generation unfaithful sometimes. In this scenario, code and table are complementary to each other for generating descriptions. The code provides essential context and explanation to enhance the comprehension of the text, while the table provides a concise and visual representation of the analysis output, supporting the text by presenting the key data points. For instance, the ground truth descriptions in Figure 1 are six different categories of description covering both code and its table output.

Second, in the Jupyter Notebooks, the cells are always with human-written guidelines supporting the corresponding descriptions. This kind of human-written guidelines usually describes a sub-

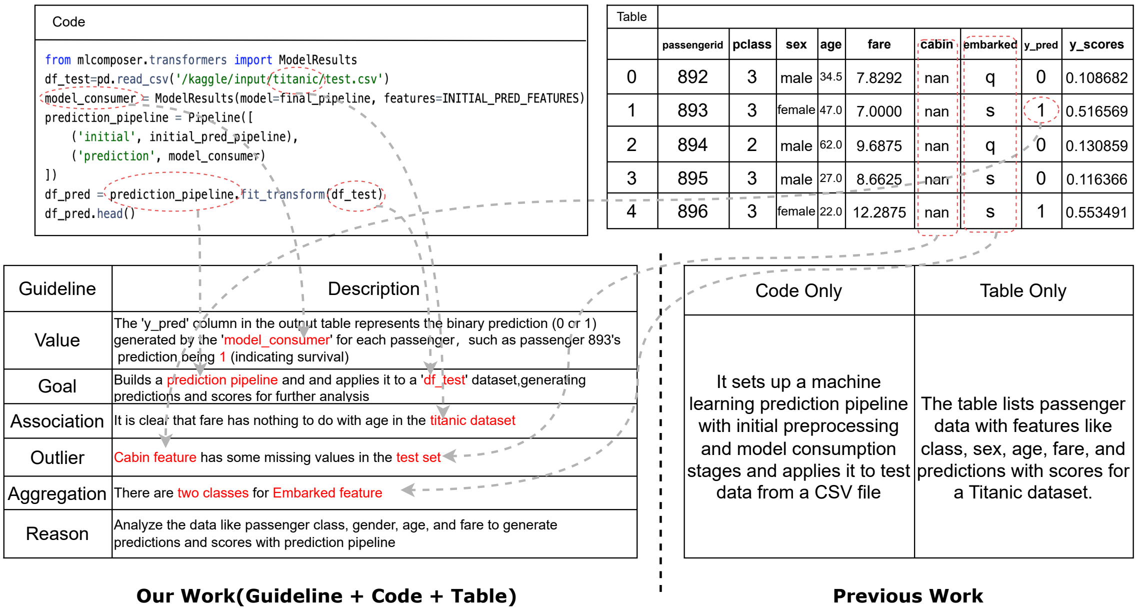

Figure 1: An example in our proposed **notebookTCDG** dataset, which targets generating **high-fidelity and personalized descriptions** based on the input of codes, tables, and user-written guidelines. Previous methods and benchmarks focus on understanding the codes or tables only, which makes the generated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

set of data facts of the table or the purpose of the code. Moreover, as the cell with the same code could serve as different purposes under different scenarios, user-written guidelines become important in the pipeline of generating descriptions.

To step forward in faithful and personalized description generation for tables and codes, in this paper, we introduce a challenging user-written guideline-based text generation task, while focusing on the table and code description generation (**TCDG**) for Jupyter Notebooks. Given a table, the relevant code, and user-written guidelines, the goal of **TCDG** is to produce a concise description under user-written guidelines. The guidelines will be of a given category corresponding to the type of target text as shown in table 11.

As previous benchmarks focus on generating descriptions for codes or tables only without the user-written guidelines, for advancing the research on the TCDG task, we construct a new dataset (**nbDescrib**) that contains around 3,924 processed code-table-description pairs extracted from 4,863 highly-ranked notebooks from Kaggle competitions and identifies 15 guideline categories of the texts (details in Section 3). 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 automate the evaluation of this dataset and investigate the performance of different models, especially the ones based on LLMs. The ablation study shows that guidelines significantly affect to the final performance at different levels, which demonstrates the validity of our task. We further conduct human evaluation and find that these advanced models still struggle to produce faithful enough results, regardless of high-quality training data. We analyze the error patterns of the models to inspire future work. In short, our data can be used to develop a unique model of the selected table and code within Jupyter Notebook, facilitating real-world applications such as the automatic creation of slides and reports. Moreover, the dataset itself is also beneficial for the NLP community as well in evaluating the capabilities of advanced NLP models.

In summary, the main contributions of our work are: (1) We formulate a novel task namely, **TCDG**, and collect a high high-quality benchmark with 2747 Code-Table-Description pairs training and 785 pairs testing data for automatic evaluation; (2) Experiments using LMs, *i.e.*, CodeT5, GPT-3, and GPT3.5 show that fine-tuned LMs (CodeT5) outperform powerful GPT-3.5, highlighting the vulnerability of LLMs on TCDG. (3) Extensive ablation studies demonstrate that guidelines significantly enhance model performance, helping users create accurately oriented and reasonable descriptions. (4) Error analysis conducted in this guideline-based text generation task to gain insights into the limitations and shortcomings of the implemented approach, paving the way for future improvements in the development of models and techniques for tackling similar challenges.

2 Related Work

To automate the machine learning and AI workflow, researchers have used automation techniques for a variety of table-related and code-related text generation tasks, including table-to-text generation, table question answering, and table-based fact verification, code documentation generation.

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., 2018a; Parikh et al., 2020a; Dhingra et al., 2019) or code documentation (Richardson et al., 2017; An et al., 2022; Liu et al., 2021; Khan and Uddin, 2022) 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 these Wikipedia candidate sentences into target sentences, instead of asking them to write new target sentences. 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 are not able to provide different target texts for different data facts in tables, resulting in too singular results when the model is trained.

Another task similar to table-to-text is table question answering (Pasupat and Liang, 2015; Wang et al., 2018). 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 focus on verification issues and cannot generate descriptive statements of the different types of 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 and most advanced models are based on BERT (e.g. 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 documentation and these documentations 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 category text generation 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. 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 prioritize interpreting table content and code snippets provide contextual information. Some other descriptions emphasize explaining the purpose of the code, requiring data-driven explanations from tables. Moreover, target description is not always relevant to the code and table themselves in the notebook, such as "From the table above, it is obvious a few things." For these kinds of documents, we label them "Other." This setup mimics the real-life 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.

3.3 Evaluation Metrics

We use the ROUGE scores (Lin, 2004) and BERTScore (Zhang et al., 2019) to evaluate our model's performance with regard to the ground-truth documentation content. We report ROUGE-1, ROUGE-2, and ROUGE-L.

4 nbDescrib

4.1 Data Collection

As we are focusing on the text 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., 2018) and do not have many tables, thus are not suitable for constructing the training dataset for this text generation 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., 2021; 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 B. To build this dataset, we also filter 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 the Kaggle administrators 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 determine if the code and table are described with a Markdown cell below. We collect these eligible code-table pairs as input. The sentences are also split if there is more than one sentence in the corresponding Markdown cell. 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

	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.

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 documentation.

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 aims to minimize the inclusion of irrelevant information.

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 11, Appendix D provides examples).

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

As shown in Table 11, 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 are more general and not highly related to the data variables or functions from the code/table (Other, 22.17%).

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

5.1 Baselines

Evaluating existing models on nbDescrib is challenging. Unlike code documentation generation, table question answering, and table-to-text, our task requires both the code and table to help generate target text documents in different guidelines. In general, we utilized three representative types of models: a fine-tuned encoder-decoder-based CodeT5, the popular decoder-only LLMs (an off-the-shelf GPT-3.5 and a fine-tuned GPT-3.0). Details are shown in Appendix C.

5.2 Results

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

Guideline	#	Description
Value	286 (7.29%)	Get the exact data attribute values for a set of criteria.
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.
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.
Other	870 (22.17%)	Other description providing supplementary details

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.

Models	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore	Pyramid Evaluation
Baselines					
CodeT5	29.61	14.38	26.72	59.51	19.37
GPT-3.5	25.19	3.32	26.31	53.00	18.43
GPT-3	19.25	4.42	20.72	51.26	13.37
Ablation Study					
CodeT5 without table	22.40	9.54	20.49	55.91	18.52
CodeT5 without code	26.35	11.72	24.01	57.93	19.14
CodeT5 without guideline description	25.09	10.78	22.61	56.63	18.75
GPT-3.5 with chain of thoughts	24.80	3.77	25.56	50.4	16.89
GPT-3.5 with in context leaning	23.09	3.50	24.62	49.6	15.88

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. As shown in Table 3, CodeT5-Large outperforms the GPT-3.5 and GPT-3 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.

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-3.5, (5) in-context learning on GPT-3.5.

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-3.5 with a CoT prompt containing both an example and middle steps of guessing a guideline (prompts shown in Appendix E). For a fair comparison, we also added the performance of in-context-learning for GPT-3.5, by removing the provided guideline and directly providing the example (prompts shown in Appendix F). The result of CoT improved over in-context learning but is still inferior to the performance of the original GPT-3.5 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, even though LLM can often generate readable and decent descriptions for code and table(see the results from Table 5 and Table 3), most of the generated descriptions are not as the users expected (see the result in Orientation dimension in Table 5). This demonstrates the necessity of guidelines. In order to fairly compare the generation models and standardize the evaluation, we need to specify what we want to generate guideline-based descriptions.

Pyramid Evaluation: To further evaluate the faithfulness of generation, we design an automatic evaluation method based on the idea of pyramid evaluation (PyrEval) (Gao et al., 2019), which is com-

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

Documentation

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-3.5	It displays the first few rows of the table which includes the data attribute age and the label indicates that oldest passenger
GPT-3	The oldest passenger is a man in his fifties

Table 4: An example of code-table cell and different model outputs.

monly used in document summarization and correlates well with human evaluation. In detail, we first extract key phrases from all generations and manually filter them. It is a more reliable metric than ROUGE since key phrases can preserve important factual information while removing unimportant tokens. In Table 3, we find that this approach and the ROUGE evaluation come up with the same trend, indicating that irrelevant tokens may not interfere much during our evaluation step. It also further validates the effectiveness of table, code, and especially guideline descriptions in ablation studies.

5.3 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 our dataset. 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 6 candidate documentation for the same code snippet-table-guideline: one by GPT-3.5 with chain-of-thought, one by GPT-3.5 with in-context-learning, one as the ground truth, and another three by a three different models. The order of these three is also randomized, so participants do not know which descriptive text is from which model. The participant is asked to rate the 4 documentation texts along three dimensions using a five-point Likert scale from -2 to 2.

- *Correctness*: The generated documentation matches the code and table content.
- *Orientation*: The generated documentation is written in the correct guideline category.
- *Readability*: The generated documentation 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-3.5 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), and GPT-3.5 (V = 5639, p = 2.84e-30). It is also worth noting that CodeT5 performs slightly better than GPT-3.5 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), and GPT-3.5 (V = 3675, p = 1.64e-20).

For the Readability dimension which considers whether the generated documentation is a valid English sentence, Ground Truth outperforms all models once again: CodeT5-Large (V = 4363, p =

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-3.5	$\overline{x}$ = -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
GPT-3.5 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-3.5 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.

1.40e-7), GPT-3 (V = 4030, p = 3.81e-14), and GPT-3.5 (V = 4135, p = 2.81e-10). It is also worth noting that GPT-3.5 with chain-of-thought and in-context-learning have worse performance than GPT-3.5 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.

5.4 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.

(1) Variable values were generated and matched incorrectly. As shown in the example in Table 4, even though CodeT5-Large, GPT-3.5 and GPT-3 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. Example from Table 8 also has this kind of error.

(3) Generating incorrectly oriented text based on guidelines. For example, GPT-3 produces text related to “Difference” but not “Trend” in the example from Table 7. Another example in Table 8, requiresmodels generating text related to “Goal”, but GPT-3 generates text related to “Association”, describing the relationship between SibSP and Parch.

(4) Reasoning error. CodeT5, GPT-3.5, and GPT-3 may generate incorrect Aggregation data (count, mean, sum) if they are operating under Aggregation guidelines. 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-3.5, and GPT-3 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%)**. This is due to the fact that the model has a tendency 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. There are many examples with insufficient code in the dataset, which causes the model to ignore the code instance in some cases.

From these errors, we can clearly see 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.

6 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 guideline 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 currentgeneration 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.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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, 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*.

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*.

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*.

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.

Flip Korn, Xuezhi Wang, You Wu, and Cong Yu. 2019. Automatically generating interesting facts from wikipedia tables. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 349–361, New York, NY, USA. Association for Computing Machinery.

Karen Kukich. 1983. Design of a knowledge-based report generator. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 145–150, Cambridge, Massachusetts, USA. 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.

Katherine Lee, Orhan Firat, Ashish Agarwal, Clara Fannjiang, and David Sussillo. 2018. Hallucinations in neural machine translation.

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. 2018a. Table-to-text generation by structure-aware seq2seq learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2018b. Table-to-text generation by structure-aware seq2seq learning. AAAI'18/IAAI'18/EAAI'18. AAAI Press.

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.

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.

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.

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*.

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.

Ratish Puduppully and Mirella Lapata. 2021. Data-to-text generation with macro planning. *Transactions of the Association for Computational Linguistics*, 9:510–527.

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

Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. 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.

April Yi Wang, Dakuo Wang, Jaimie Drozdal, Xuye Liu, Soya Park, Steve Oney, and Christopher A. Brooks. 2021. 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*.

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.

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*.

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

A 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-3.5	It includes information on the cabin attribute for the first three rows
GPT-3	The cabin column has many null values

Table 6: An example of code and its table output pair(cabin is the Cabin number)

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-3.5	it shows the survival rate for each class with higher survival rates for lower class numbers
GPT-3	the survived rate for the pclass 1 is higher than the pclass 2

Table 7: An example of code and its table output pair (Pclass is the ticket level. The smaller the value, the higher the class)

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-3.5	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

Table 8: An example of code and its table output pair

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-3.5	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

Table 9: An example of code and its table output pair

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	3.51	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	3.51	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-3.5	It aims to find extreme data cases and their corresponding attributes within a certain range in a dataset
GPT-3	data frame sort ph values

Table 10: An example of code and its table output pair

B 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]

C 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 220 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. GPT-3 achieves strong performance on many NLP tasks such as text completion, translation, and text summarization. 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-3.5 is an advanced iteration of the GPT-3 model with around 200 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.

D Appendix: Guideline Categories

E Appendix: Prompt for doing chain of thought on GPT-3.5

```
Given the 15 guidelines describing the
code cell and its table output in the
Jupiter 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)
```

[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

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 survived`
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`
Other	870 (22.17%)	Other description providing supplementary details	`It is quite handy when you can see all at once column names counts unique counts and data types`

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.

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

```
<Code>
```

F Appendix: Prompt for doing in-context learning on GPT-3.5

```
Q: When using Jupiter 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
survived rate is 38.3 denoting most of
the passengers have not survived

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