Do Text-to-Vis Benchmarks Test Real Use of Visualisations?

Hy Nguyen[1*], **Xuefei He**[1], **Andrew Reeson**[2], **Cécile Paris**[2], **Josiah Poon**[1], and **Jonathan K. Kummerfeld**[1]
The University of Sydney[1] CSIRO's Data61[2]
`nngu0448@uni.sydney.edu.au`[*]

Abstract

Large language models are able to generate code for visualisations in response to simple user requests. This is a useful application and an appealing one for NLP research because plots of data provide grounding for language. However, there are relatively few benchmarks, and those that exist may not be representative of what users do in practice. This paper investigates whether benchmarks reflect real-world use through an empirical study comparing benchmark datasets with code from public repositories. Our findings reveal a substantial gap, with evaluations not testing the same distribution of chart types, attributes, and actions as real-world examples. One dataset is representative, but requires extensive modification to become a practical end-to-end benchmark. This shows that new benchmarks are needed to support the development of systems that truly address users' visualisation needs. These observations will guide future data creation, highlighting which features hold genuine significance for users.

1 Introduction

Text-to-Vis is the task of receiving data and a request for a visualisation expressed in human language and generating code that will produce the visualisation. A system with this ability would enable faster and more complex data analysis, but there are relatively few benchmark datasets for the task. Those that do exist either focus on generating a single response (Luo et al., 2021; Srinivasan et al., 2021; Chen et al., 2021), or consider dialogue, but with limited flexibility in code (Shao and Nakashole, 2020; Song et al., 2024). Most of these datasets use generated data. The space of code variation was defined by researchers. This raises the question of whether these datasets are representative of real-world use of data visualisations.

In this study, we gathered publicly available code from the Stack[1] to analyse human preferences when making visualisations using libraries across four programming languages: Python, R, Javascript, and Vega. Since each library has different names for the same visualisation types and properties, we extracted key visualisation code and developed a cross-language mapping for several hundred functions and arguments.[2]

Using this aligned data, we analysed user behaviours when making visualisations and identified similarities and differences between real-world and benchmark datasets. Our analysis considered the chart types, functions called to define properties, and the arguments that modify how those functions behave. We observed that (1) existing benchmarks tend to focus on only one aspect of the Text-to-Vis challenge, either code synthesis, data presentation, or aesthetic attribute adjustment, and (2) only one of the datasets is consistent with real-world use and that dataset is limited by its lack of executability of code outputs.

Success on existing benchmarks does not mean systems are useful for real-world use. We need new benchmarks that cover all aspects of the problem and are consistent with patterns of use. Only then will we be able to measure progress on this valuable and challenging task.

2 Related Work

A common approach to creating Text-to-Vis datasets involves automatic synthesis of visualisations followed by human intervention for annotation (Luo et al., 2021; Shao and Nakashole, 2020; Srinivasan et al., 2021; Song et al., 2024; Chen et al., 2021). Although this approach is straightfor-

[1]A 6TB collection of open source code from GitHub (Kocetkov et al., 2022)

[2]For example, a bar plot is produced with `bar()` and `barh()` in Python, but `barplot()` in R. Our data and code are at https://github.com/giahy2507/text-to-vis-bench-assessment.

ward, datasets produced using this method often contain inherent problems. For instance, nvBench, the largest benchmark dataset for this task, was synthesised from Spider (Yu et al., 2018), a text-to-SQL dataset containing several limitations (Suhr et al., 2020), and was only partially reviewed by novices and experts for quality assurance, resulting in numerous issues (Li et al., 2024). Similarly, ChartDialogs contains limitations as the data for visualisation were automatically generated. These shortcomings underscore the need for improved methodologies in dataset creation to ensure validity and usability.

Recent research in AI has prioritised ecological validity to enhance benchmark dataset quality across various domains, aiming to align them with real-world applications (De Vries et al., 2020; Qi et al., 2023; Lu et al., 2023). Ensuring that the data used to train and test models accurately reflects users' objectives in practical scenarios is crucial. However, prior research on Text-to-Vis has not considered this aspect of dataset creation.

3 Data Collection

Instead of examining user preferences through chart images or relying on experts to comprehend how visualisations are made, our approach involves the analysis of publicly available programs specifically written for creating visualisations, e.g., line, bar, and scatter charts. This means we consider a wide range of samples from different programmers and their preferences when making visualisations. As a result, our analysis can provide a broad understanding of the essential components that are widely used.

We used code from The Stack [3] to conduct our investigation. We consider four diverse and widely used visualisation libraries: **Matplotlib**[4], **Graphics**[5], **ChartJS**[6], and **Vega-Lite**[7], which are for Python, R, Javascript, and JSON, respectively. After downloading, we selected files containing code indicative of visualisation library usage (e.g., `import matplotlib.pyplot as plt`). Finally, we used abstract syntax tree (AST) parsers and heuristics to accurately extract library-related variables, function names, arguments, and explicit values. The details are described in Appendix A, while Table 1 (upper) presents their statistics.

[3]https://huggingface.co/datasets/bigcode/the-stack-dedup
[4]https://matplotlib.org/
[5]https://www.rdocumentation.org/packages/graphics
[6]https://www.chartjs.org/docs/latest/
[7]https://vega.github.io/vega-lite/

	# samples	Proportion	# functions
nb-Matplotlib	385,338	35.89%	6,443,220
py-Matplotlib	464,463	3.5%	4,484,368
Graphics	6,721	17.15%	53,325
ChartJS	2,714	0.0128%	8,847
Vega-Lite	1,093	0.0013%	15,664
nvBench	7,241		40,478
ChartDialogs	3,284		14,690
PlotCoder	97,706		254,251

Table 1: Statistics of real-world and benchmark data. "Proportion" indicates the proportion of the library's code in the investigated programming languages. "nb-Matplotlib" indicates code from Jupyter notebooks, while "py-Matplotlib" indicates code from Python files.

Benchmark	Input	Output	Annotation note
nvBench	prompt, data	code, vis	Auto-generated based on a text-to-SQL benchmark
ChartDialogs	prompt, data, vis	code, vis	Manually annotated
PlotCoder	prompt, code	code	Auto-extracted

Table 2: Description of benchmark datasets.

We examined three publicly available benchmark datasets: nvBench (Luo et al., 2021), ChartDialogs (Shao and Nakashole, 2020), and PlotCoder (Chen et al., 2021). They vary in settings and scales, as described in Table 1 and 2. While nvBench and ChartDialogs are end-to-end Text-to-Vis benchmarks, with queries & data as input and code & visualisations as output, PlotCoder is purely a code synthesis dataset, with no data or visualisations. Appendix B shows examples from each dataset.

4 Cross-language Mapping Table

To compare the data described in the previous section, we constructed a cross-language mapping table based on frequently used parameters. This involved selecting the top 500 frequently used parameters, identifying categories and attributes, and checking correctness based on the libraries' documentation and code execution. Ultimately, the mapping table comprises 8 categories, 62 attributes, and around 850 parameters across the 4 visualisation languages. Figure 1 shows an example for the attribute "x-axis title". Details can be found in Appendix C.

Category	axes
Attribute	x-axis title
Matplotlib	xlabel ǀ xlabel set_xlabel ǀ xlabel
Graphics	plot ǀ xlab hist ǀ xlab
ChartJS	options.scales.x.title ǀ text
Vega-lite	encoding.x ǀ title encoding.x.axis ǀ title
ChartDialogs	Not found

For a given parameter "A ǀ B", A indicates the function name while B indicates the keyword argument.

A called function plot(xlab="X") containing a parameter "plot|xlab"

Figure 1: Example of the cross-language mapping

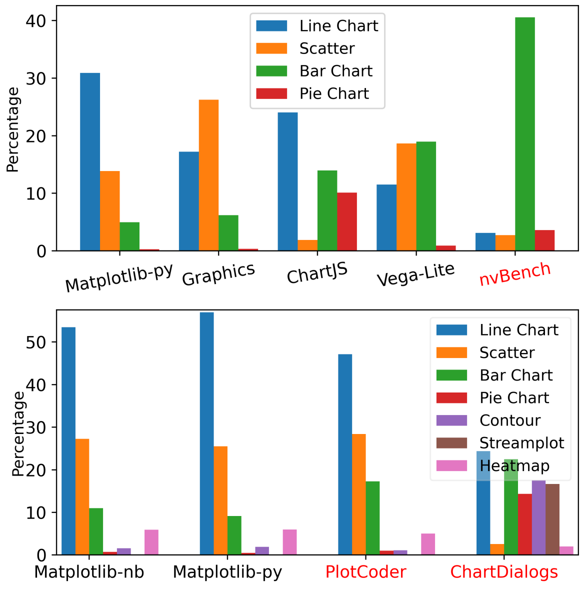


Figure 2: Plot type distribution over eight datasets.

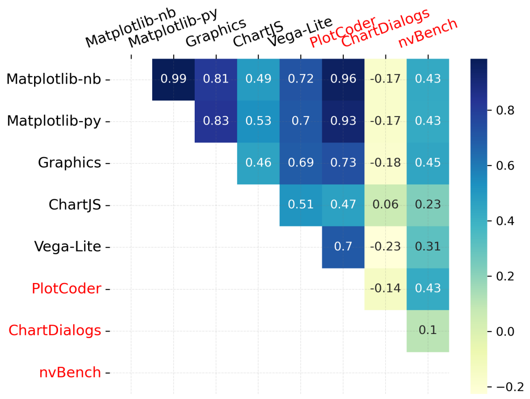


Figure 3: Spearman's rank correlation coefficient in terms of frequent attributes

5 Analysis & Discussion

5.1 Comparison of chart types

Figure 2 (upper) depicts the distribution of four common plot types across real-world datasets and nvBench [8]. Each dataset shows distinct preferences for specific plot types. The distribution of nvBench, a benchmark based on the Vega-Lite grammar, is significantly misaligned with that of Vega-Lite, where the bar chart dominates other types, accounting for over 80%, while the remaining are around 7%.

Figure 2 (lower) depicts the distribution of seven plot types across four Python-based datasets. Generally, the distribution between Matplotlib and PlotCoder shows notable similarity. This trend is because both are derived from GitHub. In contrast, ChartDialogs contains a more uniform distribution of plot types. This is the result of its design, and differs from what we observe in the wild. Specifically, ChartDialogs has fewer scatter plots and an overabundance of pie charts, contours, and stream plots.

These findings imply that nvBench and ChartDialogs are not testing the same distribution of plot types as real-world data. As suggestions for future dataset makers, it is crucial to tailor the distribution of chart types according to the specific needs and domains of the intended users. At the same time, given the imbalanced distributions in real-world data, it is also valuable to conduct separate evaluations focusing specifically on rarer plot types, acknowledging their distinct value.

5.2 Comparison of attributes

Using the cross-language mapping table and parsed data (function names and arguments), we computed the normalised frequency for 62 attributes within each dataset, as shown in Figure 8 located in Appendix D. We used these frequencies to determine the Spearman's rank correlation coefficient across eight datasets, as illustrated in Figure 3.

The real-world datasets have a significant correlation, with Spearman's values surpassing 0.7, except for ChartJS, which displays a moderate correlation with coefficients around 0.5. As for the benchmarks, ChartDialogs and nvBench show a weak correlation with their direct counterparts, Matplotlib and Vega-Lite, respectively. This means many attributes that were frequently used by end users have not been tested in these benchmarks. These include titles, axes-scale limits, tick labels, opacity, histogram bins, legend visibility, and multiple plots handling, as visualised in Figure 8 in

[8]We categorise histograms as bar charts, while polar pie and doughnut charts are grouped as pie charts

Appendix D.

Conversely, PlotCoder demonstrates a strong alignment with real-world data, with Spearman's values ranging from 0.7 to 0.9. The correlation highlights PlotCoder's potential as a resource for crafting end-to-end Text-to-Vis benchmarks. However, it is an automatically extracted dataset. It lacks data to plot in the input, visualisations in the output, and information on which versions of libraries it has as dependencies. This means the code cannot be executed as is. Without executing it, there is not way to confirm whether the visual output aligns with user goals.

5.3 Comparison of attributes when permitted

Some attributes can only be activated (or permitted) if specific preconditions are met. For instance, the "bar thickness" attribute can only be set if the user plots data on a bar chart and adjusts the width parameter. Consequently, these attributes may appear infrequently in the dataset, but users often specify their preferred values. Therefore, analysing these attributes is crucial for a deeper understanding of end users' preferences.

In this analysis, we computed the frequency of attributes for a given visualisation type or action (e.g. `plt.bar()`). This calculation is applied to each attribute in the mapping table and visualised as a heat map in Figure 9 located in Appendix D. We focus solely on examining this behaviour in Python-based datasets, including Matplotlib-nb, Matplotlib-py, PlotCoder, and ChartDialogs. This is because nvBench does not prioritise user intention for modifying aesthetic attributes while others have different characteristics.

The Spearman's coefficient calculation among these datasets reinforces our findings in the previous section. Matplotlib-nb, Matplotlib-py, and PlotCoder show significant correlations, with Spearman's scores above 0.8, whereas ChartDialogs scores below 0.1. While attributes such as axes' scales, edge colour, marker size, pie chart characteristics, legend labels, and grid line attributes receive considerable attention in ChartDialogs, end users less frequently specify them and often rely on the library's defaults.

Dataset creators should consider attributes such as histogram bins, pie precision digits, error-bar visibility, and annotation attributes, which are frequently customised by end users.

	No. Funcs	No. Params
py-Matplotlib	6.40	10.61
nb-Matplotlib	6.19	10.54
PlotCoder	4.05	6.03
Graphics	3.20	13.82
ChartJS	6.51	12.08
Vega-Lite	10.73	19.27
nvBench	4.59	10.02

Table 3: Average number of functions and parameters

5.4 Comparison of program complexity

To compare complexity, we calculate the average count of distinct visualisation functions and parameters within each code file and present the findings in Table 3. In this section, we omit ChartDialogs because it is a slot-filling dataset, with a fixed number of functions and parameters.

Benchmarks differ significantly from their direct counterparts. They use far fewer functions and parameters. In most real-world data, users employ 3 to 7 functions and 10 to 14 parameters. The top 7 functions used in Matplotlib-py include plotting the data, saving figures, assigning titles, and adjusting legends. The higher figures in the Vega-Lite dataset can be explained by its nature as a visualisation language (not a library built on top of a programming language).

6 Conclusion

In this paper, we analysed whether Text-to-Vis benchmarks accurately reflect real-world usage by presenting analyses of chart types, frequent attributes, and program complexity. Our results show that only one of the standard three benchmarks is aligned with real-world use. That dataset has its own critical limitation: it cannot be used as an end-to-end benchmark, going from a request and data as input to a visualisation as output. As well as critiquing current benchmarks, we provide guidance for future benchmark development, suggesting the evaluation of relevant attributes and challenging charts that better reflect end users' preferences. Such a benchmark would guide the development of useful and impactful systems.

Limitations

This study offers analyses of datasets and acknowledges several limitations. Firstly, our examination was restricted to only four visualisation libraries, each corresponding to a different programming language. This narrow scope may not adequately capture the diversity of applications and use cases within the field. Although we attempted to analyse MatLab code files in The Stack dataset, they are miscategorised in The Stack, processed with the wrong extension.[9] Despite our efforts to clarify this issue by reaching out to the project authors, we have yet to receive a response. Secondly, our investigation is based on public code, mainly representing programmers with different visualisation levels, including novices, practitioners, and experts. If the target users are in a visualisation application like Tableau [10], our results may not be representative. Lastly, this study concludes with an analysis and assessment of existing benchmark datasets without proposing solutions. Nevertheless, we believe that the insights and recommendations provided in this work are valuable for any dataset maker and future studies.

Ethics Statement

The data used in this research can be found publicly in the repositories of the cited papers, GitHub, or HuggingFace. Those who want to use the processed data in our repository will need to follow the terms and conditions of The Stack dataset[11].

Acknowledgments

This material is partially supported by the Australian Research Council through a Discovery Early Career Researcher Award and the Commonwealth Scientific and Industrial Research Organisation (CSIRO). We extend our gratitude to the anonymous reviewers for their constructive feedback and valuable advice on our submissions.

References

Xinyun Chen, Linyuan Gong, Alvin Cheung, and Dawn Song. 2021. Plotcoder: Hierarchical decoding for synthesizing visualization code in programmatic context. 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 2169–2181.

Harm De Vries, Dzmitry Bahdanau, and Christopher Manning. 2020. Towards ecologically valid research on language user interfaces. *arXiv preprint arXiv:2007.14435*.

Denis Kocetkov, Raymond Li, LI Jia, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, et al. 2022. The stack: 3 tb of permissively licensed source code. *Transactions on Machine Learning Research*.

Guozheng Li, Xinyu Wang, Gerile Aodeng, Shunyuan Zheng, Yu Zhang, Chuangxin Ou, Song Wang, and Chi Harold Liu. 2024. Visualization generation with large language models: An evaluation. *arXiv preprint arXiv:2401.11255*.

Xing Han Lu, Siva Reddy, and Harm De Vries. 2023. The statcan dialogue dataset: Retrieving data tables through conversations with genuine intents. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2791–2821.

Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing natural language to visualization (nl2vis) benchmarks from nl2sql benchmarks. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1235–1247.

Peng Qi, Nina Du, Christopher D Manning, and Jing Huang. 2023. Pragmaticqa: A dataset for pragmatic question answering in conversations. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6175–6191.

Yutong Shao and Ndapa Nakashole. 2020. ChartDialogs: Plotting from Natural Language Instructions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3559–3574, Online.

Yuanfeng Song, Xuefang Zhao, and Raymond Chi-Wing Wong. 2024. Marrying dialogue systems with data visualization: Interactive data visualization generation from natural language conversations. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2733–2744.

Arjun Srinivasan, Nikhila Nyapathy, Bongshin Lee, Steven M Drucker, and John Stasko. 2021. Collecting and characterizing natural language utterances for specifying data visualizations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–10.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online.

[9]https://huggingface.co/datasets/bigcode/the-stack-dedup/blob/main/programming-languages.json
[10]https://www.tableau.com/
[11]https://huggingface.co/datasets/bigcode/the-stack-dedup

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium.

A Code Parsing

After obtaining code files for Python and R, we used abstract syntax tree (AST) parsers and heuristics to accurately extract variables, function names, arguments, and explicit values. Subsequently, we tracked the assigned variables to correctly select the functions used in Matplotlib while a list of Graphics' functions was used to filter for this library.

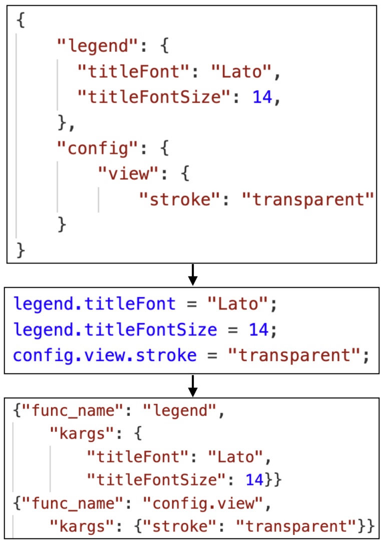
To extract ChartJS specifications, we initially used an AST parser to extract all JSON data from the Javascript code files. Subsequently, a heuristic selection method was applied to filter JSON containing the three essential components of this library, namely "type," "data," and "options." This is because ChartJS relies on the JSON format as its foundation, serving as the input for executing functions in Javascript.

Vega-Lite can appear in both JSON and Javascript files, as it is a JSON schema visualisation language. Therefore, we used the above methods for extraction. In detail, after extracting JSON data from code files, we exclusively extracted snippets containing Vega-Lite schema [12], which is a mandatory field of Vega-Lite specification.

After extracting functions, arguments, assigned values, and JSON specifications, targeting the visualisation libraries, we transformed them into a universal format to facilitate more accessible analysis and further processing. For instance, a command in Python `ax.plot(x, color='green', marker='o')`, which plots a line graph of 'x', with marker 'o' and colour 'green', can be parsed into a JSON as `{"func_name": "plot", args: ["x"], kargs: {"color": "green", "marker": "o"}}`. An example of translating JSON to universal format can be seen in Figure 4.

Regarding nvBench and PlotCoder, they contain visualisation code in Vega-Lite and Python, so the process was the same as described above. When it comes to ChartDialogs, a slot-filling dataset, we converted each user's intent to a function with changed slots as keyword parameters. For example, a user's intent "smaller radius, increase text size" modifying a pie chart is transformed as universal JSON format `{'func_name':'pie', kargs:{'radius: 'small', 'font_size':'large'}}`.

[12]Vega-Lite schema: v1, v2, v3, v4, v5,



```
{
    "legend": {
      "titleFont": "Lato",
      "titleFontSize": 14,
    },
    "config": {
        "view": {
            "stroke": "transparent"
    }
}
```

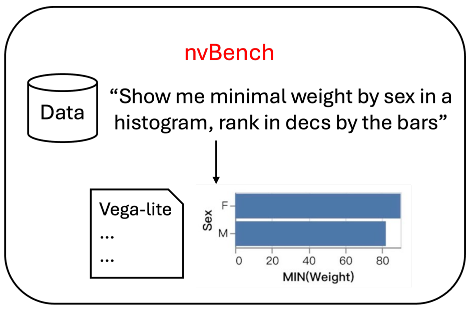
```
legend.titleFont = "Lato";
legend.titleFontSize = 14;
config.view.stroke = "transparent";
```

```
{"func_name": "legend",
   "kargs": {
       "titleFont": "Lato",
       "titleFontSize": 14}}
{"func_name": "config.view",
   "kargs": {"stroke": "transparent"}}
```

Figure 4: The process of converting JSON to universal format

B Benchmarks' Examples

Figures 5, 6, and 7 illustrate examples for nvBench, ChartDialogs, and PlotCoder, respectively.



"Show me minimal weight by sex in a histogram, rank in decs by the bars"

Figure 5: A sample from the nvBench dataset.

C Cross-language Mapping Table

The procedure for making the cross-language table is as follows. Initially, we compiled the top 100 frequent parameters from the real-world dataset in 4 languages: Matplotlib, Graphics, ChartJS, and Vega-Lite. Subsequently, the parameters were grouped into different categories and attributes. The mapping table was further expanded by investigating relevant parameters within the top 500.

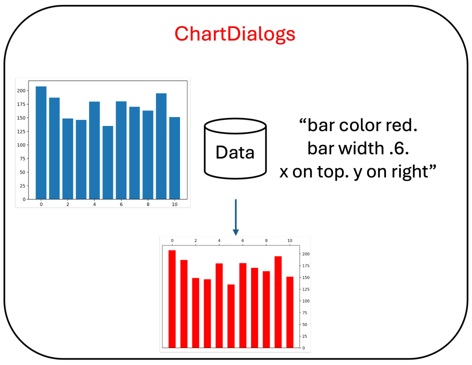
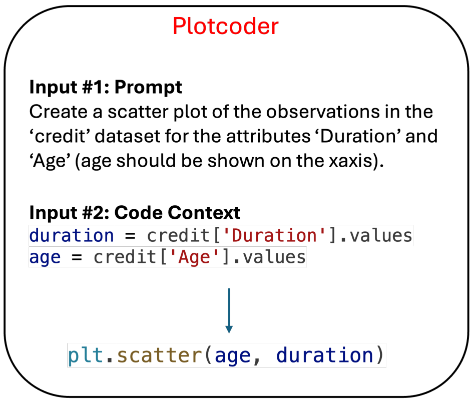


Figure 6: A sample in ChartDialogs dataset. This dataset was built in a slot-filling manner. The visualisation is generated by a hard-coded program.



Input #1: Prompt
Create a scatter plot of the observations in the 'credit' dataset for the attributes 'Duration' and 'Age' (age should be shown on the xaxis).

Input #2: Code Context
```
duration = credit['Duration'].values
age = credit['Age'].values
```

```
plt.scatter(age, duration)
```

Figure 7: A sample from the PlotCoder dataset.

If a specific language lacked relevant parameters for a given attribute in the top 500 (resulting in a blank cell), we persistently searched through the remaining list until a match was found. Cells where no relevant parameter was identified led to the annotation of "not found." This identification and verification process includes understanding plotting parameters, identifying them in API documents, asking ChatGPT [13] for explanations and relevant parameters and executing example codes.

Table 4 shows a small part of the table for context. The whole table can be found in our repository at https://github.com/giahy2507/text-to-vis-bench-assessment.

D Calculation for heat map figures

Figure 8 shows heat maps of the most common visualisation attributes over 7 datasets, where the more intense green colour indicates a higher percentage of usage within the dataset. The calculation for each attribute is k/n, where:

[13] https://chat.openai.com https://bard.google.com/chat

Category	Attribute
Axes	x-title, y-title, x-y-title-fontsize, x-y-title-color, x-y-lim, x-y-ticks-labels, x-y-ticks-labels-color, x-y-ticks-labels-rotation, x-y-scale, x-y-ticks-fontsize, x-axis-ticks-visible, y-axis-ticks-visible, x-y-scale-position, invert-x-y-axis
Data Appearance	filled-color, edge-color, opacity, linewidth, markersize, linestyle, line-capstyle, markerstyle, bar-thickness, bar-data-stacking, hist-bins, pie-explode, pie-label-distance, pie-percentage-distance, pie-precision-digits, pie-radius, errbar-cap-size, errbar-cap-thick, errbar-color, errbar-visible
Annotation	ann-text/label, ann-fontsize, ann-possition, ann-font
Main title	title, title-fontsize, title-color, title-position, subtitle, subtitle-fontsize
Legend	legend-title, legend-fontsize, legend-position, legend-labels, legend-labels-color, legend-is-display
Grid	grid-visible, grid-color, grid-linestyle, grid-linewidth
Format	size, dpi, saving-format
Other	bounding-box/border, background, margin/padding, multiple-plots

Table 4: Categories and Attributes in the cross-language mapping table.

- k is the number of times that attribute's arguments are specified
- n is the number of times that all arguments are specified

As for the heat map in Figure 9, there are two cases influencing different levels. For attributes impacting the program level, such as title, x-axis title, and x-y tick labels, the percentage is derived from how frequently a program includes arguments for a specific attribute. Conversely, for local attributes affecting the function level, like filled colour, opacity, and bar thickness, the percentage is calculated based on the frequency of functions containing arguments for the given attribute. The calculation is as follows:

- k is the number of times that attribute's arguments are specified. p is the number of times that attribute's functions are used
- z represents the total number of programs in the dataset, while g denotes the number of programs in which the attribute is used (any of the attribute arguments is used).

While a figure for a given program-level attribute is g/z, that for function-level one is k/p.

Attribute	Matplotlib-nb	Matplotlib-py	PlotCoder	ChDialogs	Graphics	ChartJS	Vega-Lite	nvBench
x-title	8.2	6.9	12.8	0.0	5.9	1.5	6.3	45.8
y-title	8.3	7.2	12.7	0.0	5.6	1.8	6.4	45.8
x-y-title-fontsize	2.6	2.2	2.4	0.0	0.7	0.3	0.5	0.0
x-y-title-color	0.1	0.2	0.1	0.0	0.0	0.5	0.2	0.0
x-y-lim	2.3	3.5	1.5	0.0	5.6	6.4	6.0	0.0
x-y-ticks-labels	3.0	3.4	2.4	0.0	3.3	1.3	2.5	0.0
x-y-ticks-labels-color	0.1	0.1	0.0	0.0	0.1	0.6	0.3	0.0
x-y-ticks-labels-rotation	0.7	0.5	0.9	0.0	1.3	0.4	1.3	0.0
x-y-scale	0.3	0.4	0.3	11.4	0.3	1.9	0.8	0.0
x-y-ticks-fontsize	0.8	1.0	0.4	0.0	1.1	0.8	0.6	0.0
x-axis-ticks-visible	0.9	1.3	0.7	0.0	0.9	1.4	0.5	0.0
y-axis-ticks-visible	0.6	1.0	0.4	0.0	0.6	1.6	0.5	0.0
x-y-scale-position	0.2	0.3	0.1	13.4	1.8	0.3	0.5	0.0
invert-x-y-axis	0.1	0.1	0.0	7.9	0.0	0.1	0.2	0.0
filled-color	11.2	11.7	10.2	6.1	15.3	21.5	22.8	4.4
edge-color	0.6	0.7	0.6	3.5	1.2	13.9	1.5	0.0
opacity	2.4	2.4	2.3	0.0	0.0	0.0	4.6	0.0
linewidth	2.5	3.2	2.3	6.7	5.9	5.5	3.3	0.0
markersize	1.7	1.9	2.0	3.0	2.0	2.3	1.6	0.0
linestyle	2.9	3.3	2.2	2.9	4.9	0.8	0.7	3.1
line-capstyle	0.0	0.0	0.0	0.0	0.0	0.4	0.0	0.0
markerstyle	3.5	3.6	3.0	1.8	5.3	0.2	0.4	0.9
bar-thickness	0.4	0.5	0.5	3.1	0.1	0.5	0.1	0.0
bar-data-stacking	0.0	0.1	0.0	0.0	0.2	1.2	0.4	0.0
hist-bins	1.0	0.6	2.7	0.4	0.8	0.0	1.5	0.0
pie-explode	0.1	0.0	0.1	1.4	0.0	0.0	0.0	0.0
pie-label-distance	0.0	0.0	0.0	2.5	0.0	0.0	0.0	0.0
pie-percentage-distance	0.0	0.0	0.0	2.6	0.0	0.0	0.0	0.0
pie-precision-digits	0.1	0.1	0.3	1.4	0.0	0.0	0.0	0.0
pie-radius	0.0	0.0	0.0	2.9	0.0	0.0	0.1	0.0
errbar-cap-size	0.1	0.1	0.0	3.0	0.3	0.0	0.0	0.0
errbar-cap-thick	0.0	0.0	0.0	2.4	0.0	0.0	0.0	0.0
errbar-color	0.0	0.1	0.0	1.8	0.2	0.0	0.0	0.0
errbar-visible	0.4	0.5	0.1	1.6	0.3	0.0	0.0	0.0
ann-text-label	0.5	0.5	0.2	0.0	3.9	0.2	2.0	0.0
ann-fontsize	0.7	1.0	0.2	0.0	2.8	0.1	0.8	0.0
ann-possition	0.9	1.6	0.3	0.0	2.9	0.2	3.2	0.0
ann-font	0.1	0.2	0.0	0.0	0.7	0.1	0.7	0.0
title	8.5	6.7	11.5	0.0	5.6	5.7	3.4	0.0
title-fontsize	1.2	0.8	1.1	0.0	0.5	1.0	0.6	0.0
title-color	0.0	0.0	0.0	0.0	0.0	0.5	0.4	0.0
title-position	0.1	0.1	0.0	0.0	0.0	0.1	0.4	0.0
subtitle	0.4	0.7	0.5	0.0	0.3	0.0	0.4	0.0
subtitle-fontsize	0.2	0.2	0.2	0.0	0.1	0.0	0.1	0.0
legend-title	0.1	0.1	0.0	0.0	0.2	0.0	1.1	0.0
legend-fontsize	0.5	0.5	0.2	0.0	0.8	0.8	0.5	0.0
legend-position	2.3	2.4	1.9	0.0	3.0	1.6	1.5	0.0
legend-labels	7.9	7.3	5.8	3.9	1.5	9.5	1.0	0.0
legend-labels-color	0.0	0.0	0.0	0.0	0.1	0.6	0.2	0.0
legend-is-display	5.0	4.5	4.4	0.0	2.0	4.3	0.0	0.0
grid-visible	0.7	0.8	0.7	0.0	0.2	3.5	1.8	0.0
grid-color	0.1	0.2	0.1	5.0	0.0	1.8	0.0	0.0
grid-linestyle	0.2	0.2	0.1	4.9	0.0	1.7	0.1	0.0
grid-linewidth	0.1	0.1	0.0	6.4	0.0	0.2	0.0	0.0
size	8.3	5.1	7.2	0.0	2.6	0.0	14.7	0.0
dpi	0.9	1.6	0.3	0.0	0.5	0.0	0.0	0.0
saving-format	0.2	0.5	0.1	0.0	1.8	0.0	0.0	0.0
bounding-box-border	0.1	0.3	0.2	0.0	1.2	0.0	0.0	0.0
background	0.1	0.1	0.0	0.0	0.8	0.0	0.6	0.0
margin-padding	3.0	5.1	1.7	0.0	3.0	2.9	0.9	0.0
multiple-plots	2.8	2.6	2.4	0.0	1.8	0.0	2.0	0.0

Figure 8: Heat map of the most frequent aesthetic attributes over 7 datasets. The attributes are classified by different categories with colours, such as x and y axes , data appearance , annotation , title and subtitle , legend , grid , figure format , and others .

Attribute	Matplotlib-nb	Matplotlib-py	PlotCoder	ChDialogs
x-title	46.3	29.9	15.0	0.0
y-title	46.0	30.0	15.0	0.0
x-y-title-fontsize	7.8	4.7	1.5	0.0
x-y-title-color	0.8	0.5	0.1	0.0
x-y-lim	7.2	6.9	0.9	0.0
x-y-ticks-labels	12.4	8.7	1.9	0.0
x-y-ticks-labels-color	0.4	0.4	0.0	0.0
x-y-ticks-labels-rotation	5.6	2.5	1.1	0.0
x-y-scale	2.3	1.8	0.4	27.6
x-y-ticks-fontsize	3.3	2.7	0.3	0.0
x-axis-ticks-visible	0.0	12.0	0.0	0.0
y-axis-ticks-visible	0.0	8.6	0.0	0.0
x-y-scale-position	0.9	1.5	0.0	21.2
invert-x-y-axis	0.0	0.7	0.0	38.2
filled-color	33.6	39.3	23.5	29.5
edge-color	2.3	2.7	1.5	13.2
opacity	8.7	9.1	6.2	0.0
linewidth	10.8	14.5	6.7	20.4
markersize	9.5	11.7	7.3	26.4
linestyle	6.3	7.6	5.4	16.6
line-capstyle	0.0	100.(	0.0	0.0
markerstyle	7.6	6.6	5.3	16.1
bar-thickness	40.2	55.0	31.2	36.4
bar-data-stacking	5.1	9.5	1.9	0.0
hist-bins	72.8	77.9	62.1	44.0
pie-explode	36.3	29.7	44.6	23.2
pie-label-distance	4.0	5.1	3.1	41.1
pie-percentage-distance	6.3	7.5	1.4	43.1
pie-precision-digits	74.2	65.1	77.1	22.6
pie-radius	7.8	9.4	5.4	47.4
errbar-cap-size	24.5	23.5	6.0	15.9
errbar-cap-thick	7.4	7.2	0.9	12.8
errbar-color	11.3	15.3	6.9	9.7
errbar-visible	100.(	38.1	100.(	8.4
ann-text-label	33.6	26.1	34.9	0.0
ann-fontsize	51.1	47.4	36.4	0.0
ann-possition	63.6	76.7	43.8	0.0
ann-font	13.2	14.0	6.9	0.0
title	44.2	26.6	12.5	0.0
title-fontsize	6.7	3.3	1.2	0.0
title-color	0.3	0.1	0.0	0.0
title-position	0.3	0.2	0.0	0.0
subtitle	3.7	3.7	0.5	0.0
subtitle-fontsize	1.6	1.2	0.2	0.0
legend-title	0.7	0.5	0.1	0.0
legend-fontsize	3.4	2.2	0.2	0.0
legend-position	14.9	2.1	2.5	0.0
legend-labels	36.0	16.8	16.9	53.2
legend-labels-color	0.0	0.0	0.0	0.0
legend-is-display	79.5	40.3	8.8	0.0
grid-visible	4.5	3.1	0.9	0.0
grid-color	0.7	0.8	0.1	43.3
grid-linestyle	1.0	0.8	0.1	43.2
grid-linewidth	0.5	0.5	0.0	58.3
size	50.1	24.2	8.7	0.0
dpi	5.5	6.4	0.4	0.0
saving-format	1.1	1.2	0.1	0.0
bounding-box-border	0.1	0.1	0.0	0.0
background	0.5	0.3	0.0	0.0
margin-padding	6.7	8.6	0.6	0.0
multiple-plots	70.2	22.9	8.4	0.0

Figure 9: Heat map of attributes that the user often specifies values when permitted.

Category	Attribute	Matplotlib	R	ChartJS	Vega-Lite	ChartDialogs
x-y-axis	lim	add_subplot|xlim set|xlim axes|xlim set_xlim|left set_xlim|right xlim|left xlim|right xlim|xmin xlim|xmax set_xlim|xmin set_xlim|xmax add_subplot|ylim set|ylim axes|ylim set_ylim|bottom set_ylim|top ylim|bottom ylim|top ylim|ymin ylim|ymax set_ylim|ymin set_ylim|ymax	plot|xlim hist|xlim barplot|xlim plot.window|xlim plot.default|xlim matplot|xlim boxplot|xlim curve|xlim points|xlim lines|xlim plot|ylim barplot|ylim boxplot|ylim hist|ylim lines|ylim matplot|ylim points|ylim plot.window|ylim plot.default|ylim	options|scales|xAxes|ticks|beginAtZero options|scales|xAxes|ticks|suggestedMax options|scales|xAxes|ticks|min options|scales|xAxes|ticks|max options|scales|x|beginAtZero options|scales|x|min options|scales|x|max options|scales|yAxes|ticks|beginAtZero options|scales|y|beginAtZero options|scales|yAxes|ticks|suggestedMax options|scales|yAxes|ticks|min options|scales|yAxes|ticks|max	encoding|x|scale|domain encoding|x|scale|domain|selection encoding|x|scale|domain|param encoding|y|scale|domain encoding|y|scale|domain|selection encoding|y|scale|domain|param	Not found
x-y-axis	x-y-scale	xscale|value set_xscale|value yscale|value	plot|log lines|log boxplot|log	options|scales|xAxes|type options|scales|x|type options|scales|yAxes|type options|scales|y|type	encoding|x|scale|type encoding|y|scale|type	plot|x_axis_scale contour|x_axis_scale bar|x_axis_scale scatter|x_axis_scale plot|y_axis_scale contour|y_axis_scale bar|y_axis_scale scatter|y_axis_scale
data appearance	color	plot|color plot|c scatter|color scatter|c plot|fmt=color axvline|color axhline|color bar|color barh|color fill_between|color hist|color errorbar|color contour|colors set_facecolor|color vlines|color hlines|color Circle|color Line2D|color axvspan|color quiver|color pie|colors arrow|color text|color annotate|color	lines|col plot|col points|col abline|col barplot|col hist|col polygon|col barplot|col rect|col segments|col boxplot|col image|col curve|col pie|col matplot|col contour|col stripchart|col text|col mtext|col	data|datasets|backgroundColor data|datasets|pointBackgroundColor options|plugins|datalabels|color data|datasets|fillColor data|datasets|strokeColor options|elements|line|backgroundColor data|datasets|pointColor options|plugins|crosshair|line|color options|elements|point|backgroundColor data|datasets|pointStrokeColor options|plugins|datalabels|color	encoding|color|scale|range encoding|color|value encoding|color|aggregate mark|fill mark|color encoding|color|scale|scheme encoding|color|sort config|mark|color mark|color|stops|offset mark|color|stops|color encoding|color|type	bar|bar_face_color hist|bar_face_color plot|line_color plot|marker_face_color scatter|marker_face_color
data appearance	markerstyle	plot|fmt=markerstyle plot|marker scatter|marker errorbar|marker	points|pch plot|pch legend|pch lines|pch pairs|pch matplot|pch stripchart|pch par|pch	data|datasets|pointStyle	encoding|shape|type encoding|shape|value	plot|marker_type scatter|marker_type
data appearance	bar-thickness	bar|width barh|height	barplot|width	data|datasets|barPercentage data|datasets|maxBarThickness data|datasets|barThickness data|datasets|barWidth options|barThickness	mark|width mark|height	bar|bar_height bar|bar_width
data appearance	hist-bins	hist|bins	hist|breaks	Not found	encoding|x|bin encoding|x|bin|step encoding|x|bin|maxbins encoding|y|bin|maxbins encoding|x|bin|step	hist|number_of_bins
title	title	title|label set_title|label set|title set_title|title	plot|main hist|main title|main barplot|main boxplot|main pie|main image|main matplot|main pairs|main plot.default|main curve|main	options|title|display options|title|text options|plugins|title|display options|plugins|title|text options|title	title|title title|text	Not found

Table 5: Details of the cross-language mapping table for 7 attributes over 3 categories. Each parameter in attributes comprises two parts, function name and argument name, separated by “|”.