Visualization Linter

Wei Zheng, Youssef Sherif

Abstract—In our project, we developped two packages for linting data visualization practices. One package is for rules that are feasible to be dealt with static analysis and another package for rules that depends on data runtime. In our process we found Eslint to provide a sturdy framework to build static analysis plugins on. We found linting D3.js is challenging since the library is low level. Other higher level packages such as Chart.js were easier to lint. Besides the packages, we developped two documentation websites that contain further readings for data visualization rules. Users can use these readings to educate themselves about data visualization. Finally, we have also set up an online demo that displays the usage of the chart.js rumtime checker.

Index Terms—data visualization, D3, Chart.js, linter

Introduction

A wide range of users uses visualization libraries. Some of these users are experts in data visualization, and others are not. They would benefit immensely by checking whether they are adhering to visualization best practices or not, because scientists may not understand the meaning of some designing rules in data visualization. Even experts in data visualization might sometimes miss minor details while they are prototyping in a hurry. Displaying warning and error messages for visualization practices can be useful for those users and would help improve the overall quality visualizations produced by the community.

Linter applies rules to source code and provides information to users when they violate a set of predetermined rules. Meeks borrows the term linter and introduces the idea of "vis linting."[13]

One of the goals of designing a visualization linter is to help users understand whether the visualization they have built can have higher effectiveness. We are proposing a static analysis tool and runtime library that allows users to adhere to data visualization best practices. Our primary research goal is to target the most commonly used web-based data visualization library, so the library we chose at first was D3.js. However, during the research, we found that D3 is difficult to work with; therefore, we turned to chart.js.

But why two tools? Why not just a linter? Linters, which are static analysis tools, are not meant to check data-related issues. They are only able to check for logical problems in the code. The reason linters cannot check data-related matters is that data are dynamic. Data might not even exist in code and can be retrieved by an HTTP request from a backend server. We can use static analysis tools to check data-independent code, such as the usage of a specific kind of chart, for example. Here comes the run time library to the rescue. Rules that check issues with data, such as the color mismatch, should be implemented in a runtime library, because a runtime library has access to the data, even if it is coming from a different server.

1 Related Work

Researchers have sorted out and summarized data visualization rules or guidelines in the Visualization Guidelines Repository (VGR) [4]. It involves guidelines curated from books, papers, and blogs on the topic of visualization and is maintained by the DBVIS group at the University of Konstanz.

Data-to-viz [6] provides the caveats for data visualization, which is a similar but more detailed rules repository. The library of Duke University summarizes the Dos and Don'ts for visualization, offering a guide to users.

- *Wei Zheng. E-mail: williamubc19@gmail.com.*
- *Youssef Sherif. E-mail: sharief@aucegypt.edu.*

Although rules or guidelines can help the data visualization process, those visualizing data face the challenge of internalizing, remembering, and consistently applying the guidelines throughout their work. Alternatively, systems embedded with rules can provide automated recommendations, which is a more practical way to implement data visualization while following the rules. Some commercial tools have embodied this recommendation function, such as Tableau [11] and Spotfire [12]. For professional users such as data scientists, they usually use a language visualization library, instead, to display their research results.

McNutt and Kindlmann [1] have developed vislint_mpl, which is a tool for matplotlib, the widely used visualization library for Python. Mcnutt proposed a long list of data visualization rules, but he implemented a few of them in his prototype. Although vislint_mpl only implemented a small number of rules, it demonstrates the practical feasibility of vis linting. While Mcnutt has named his solution Vislint and is referring to it as a linter, he is using a runtime library to check for errors and warnings. The term "linter" is usually associated with static analysis tools, and this is the first time we find that "linter" is called for a runtime checker. We have found JavaScript runtime checker libraries that are not involved with data visualization.

2 Task abstraction

Our work library consists of two parts; one is the static part which takes in JavaScript source code and evaluates it statically, to check if the code makes logical mistakes. The other part is the run-time part, which can access to the visualized data processed by visualization libraries like D3 or Chart.js. In this project, for the run-time part, we developed rules that interface with Chart.js. After being imported into their JavaScript-based project, our visualization linter will provide unambiguous feedback with the automation of visualization guidelines for the users.

The rules or guidelines for visualization we consider to implement are in Table 1. Due to space limitations, we do not show all the 41 guidelines; instead, provide the categories of the guidelines and an example for each category. We implemented only a portion of these rules.

Table 1. Visualization guidelines

Category	Count	Example
Bar chart	6	Use horizontal labels. Avoid steep diagonal or vertical type, as it can be difficult to read.
Chart junk	1	Chart junks are bad, do not use it.
Colors in vis	7	Don't use more than 6 colors together. Too many colors mean similar hues will appear, like BLUE and GREEN which can be difficult to tell apart.
General	6	Visual displays of information should present both cause and effect.

Graphs and network visualization	3	Change the layout for the graphical design to improve the readability of the design.
Interaction	4	Do not use blow-apart effects.
Pie chart	5	Visualize no more than 5 categories per pie chart.
Process	2	Do ask for the opinion of others after designing your visualizations.
Text and document vis	1	All nouns should be encoded in black, verbs in red, adjectives in green, determiners in grey, particles in brown, conjunctions in blue, and Interjection in yellow.
Time-series plots	6	Set the height of a line chart such that the data in the line chart takes up approximately two-thirds of the y-axis' maximum scale.

3 SOLUTION

Figure 1 shows that D3 is the most widely used data visualization JavaScript library. Therefore, we first try to apply rules for D3. During the process of implementing, we analyzed these rules to distinguish which rules are suitable for static analysis, which are suitable for run-time analysis, and which cannot be even implemented for some reason. Afterward, we implement the rules for static and run-time parts separately as the two functional modules of our library.

Fig. 1. Download numbers of four popular JS visualization libraries.

3.1 Rules analysis

3.1.1 Run-time analysis

Figure 5 illustrates the DOM tree of the bar chart, and it indicates that D3 charts are rendered in SVG (Scalable Vector Graphics), which enables us to interact with each element of the chart.

```
▼<body>
  ▼<svg width="600" height="500">
    ▼<g transform="translate(100,100)">
      ▼<g transform="translate(0,300)" fill="none" font-size="10" font-
      family="sans-serif" text-anchor="middle">
          <path class="domain" stroke="#000" d="M0.5,6V0.5H400.5V6">
          </path>
        ▼<g class="tick" opacity="1" transform="translate(75,0)">
            <line stroke="#000" y2="6"></line>
            <text fill="#000" y="9" dy="-.55em" dx="-.8em" transform=
            "rotate(-90)" style="text-anchor: end;">2013</text>
          </g>
        ▶<g class="tick" opacity="1" transform="translate(200,0)">…</g>
        ▶<g class="tick" opacity="1" transform="translate(325,0)">…</g>
        </g>
      ▶<g fill="none" font-size="10" font-family="sans-serif" text-
      anchor="end">…</g>
        <rect class="bar" x="25" y="196.078431372549" width=
        "100.00000000000001" height="103.921568627451"></rect>
        <rect class="bar" x="150" y="0" width="100.00000000000001" height=
        "300"></rect>
        <rect class="bar" x="275" y="58.823529411764696" width=
        "100.00000000000001" height="241.1764705882353"></rect>
      </g>
    </svg>
  ▶<script src="barChart.js">…</script>
  </body>
```

Fig. 5. SVG structure of the bar chart in Figure 2.

By traversing the DOM tree, we can get the x label degree of the bar chart, and in this way, we can implement the rule for checking if the x label is in sharp degree. The code is shown in Figure 6. When a bar chart violates this rule, the linter will generate an error in the console of the browser, as in Figure 7.

```
1  function checkTick(){
2      var svg = document.getElementsByTagName("svg")[0];
3      var label = svg.querySelector(".tick > text");
4      var valueStr = label.getAttribute("transform");
5      var num= valueStr.replace(/[^0-9]/ig,"");
6      if(num>60){
7          console.error("Use horizontal labels. Avoid steep diagonal or vertical \
8          type, as it can be difficult to read.");
9      }
10 }
```

Fig. 6. Run-time implementation of the bar chart rule.

```
▶Use horizontal labels. Avoid steep diagonal or vertical type, as it can be difficult to read.   barChart.js:66
```

Fig. 7. Visualization Linter information.

This method also has some drawbacks. Firstly, it is difficult to find the related nodes and properties from D3 DOM tree. Secondly, this method turns out to be fragile when users intentionally change the class of DOM nodes generated by D3.

Since dealing with SVG is hard, we thought about asking the user to manually provide in code features of the graph like the type of graph, the x-axis of the graph if applicable, and the y-axis of the graph if applicable. Unlike SVG's data structure, data structures of other graph's elements such as the axes are easier to parse and comprehend.

However, there are also some problems with this method. A D3 user might want to have a customized graph that does not fit in one of the types we support, like a "bar chart" or "pie chart." It is an annoyance for the user to provide all of these data. The method is fragile since the user via D3 can transform the SVG itself later, and these data structures might not be representatives of the real graph.

3.1.2 Unimplemented rules

During the research process, we find that the implementation complexity for some rules is dramatically high because those rules have no relation to the object, such as a chart in Chart.js and SVG node in D3, which is the basic visual unit of a visualization library.

For example, the rule "Visual displays of information should present both cause and effect" is just a high-level idea, and have no direct connection with the visualization process. Therefore, it has a low probability of being implemented in either library.

3.2 Implement rules for Chart.js

From the analysis above, we believe that developing a linter for D3 might be fragile because it is hard to know the intent of what the user wants to visualize from the code. Besides, the variety and possibilities of visualizations that might be drawn via D3 are almost infinite. Therefore, we decide to change our target from D3 to another visualization Javascript library---Chart.js. Chart.js supports six basic chart types, and it is best used for standard and simple graphs.

There are two main reasons why we choose Chart.js. The first one is that its data structure is more straightforward and easy to understand, shown in Figure 8. The other reason is that it is the second most adopted Javascript visualization library, as shown in Figure 1. However, there are still some rules that cannot be implemented by Chart.js, like "Space bars appropriately. Space between bars should be ½ bar width." The reason is Chart.js internally add spaces between bars, and the space value is not reflected in the chart object, so it is not possible to check if space is of the half of the bar width.

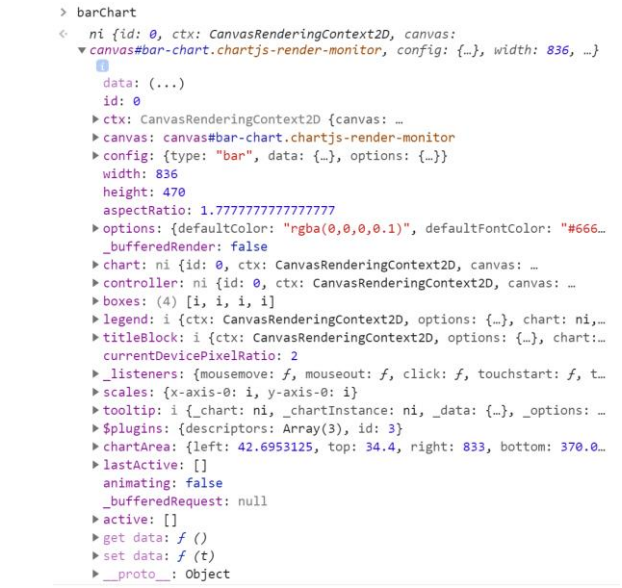

Fig. 8. Chart.js data structure.

4 Implementation

4.1 Static analysis tool

There was more than one approach for implementing the static analysis part. The first approach is to use a JavaScript parser like Esprima [14] to perform tokenization and parsing of the JavaScript program. This is the direct approach for building a static analysis tool for JavaScript. However, a better approach is to build a plugin for a pluggable static analysis tool like Eslint, which uses a JavaScript parser in the background. Eslint is helpful in more than one area. First, numerous Eslint plugins are published on the web like "eslint-plugin-jquery"[16]. We can look at these projects and follow best practices regarding project structure and architecture. Second, numerous IDEs have extensions for Eslint so that warnings and errors are visibly identified to the user. This means that users can access the linter both via the command line and via the IDE's visual alert. Third, Eslint has a testing module that helps developers develop readable unit tests. Fourth, Eslint has a clear developer's guide[15] to help build plugins on top of Eslint. Fifth, Eslint enables the user to turn on or off custom rules from plugins "out-of-the-box". Finally, Eslint supports rule categories "out-of-the-box". For example, we can have a "strict category" where all rules are turned on, and a "flexible" category where only important rules are turned on.

We cloned "eslint-plugin-jquery" library and used its structure, dependencies, and scripts. We removed the original library's rules and added our own.

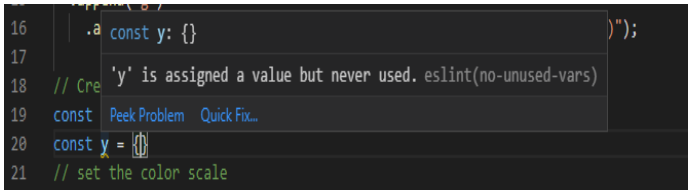

Fig. 9. An example that shows Eslint's IDE integration via an extension

```
ruleTester.run('no-ajax', rule, {
  valid: [`d3.bar()`, 'd3.line()'],
  invalid: [
    {
      code: 'd3.pie()',
      errors: [{message: pieChartError, type: 'CallExpression'
    }
  ]
```

Fig. 10. Eslint's intuitive testing module

4.2 Run-time analysis tool

Our prototype, Javascript-based visualization linter, evaluates visualizations, created in Chart.js of a set of rules. After creating a Chart.js chart, the run-time analysis tool checks the chart object with embedded rules, and the user then receives a list of rules that failed, along with a link where they can find detail information and explanation of the failed rules. An example of the type of guidance can be seen in Figure 11. After clicking the link in the messages, instructions will show in a new tab of the browser, as shown in Figure 12.

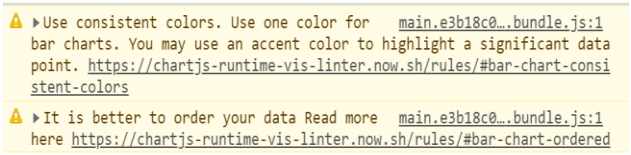

Fig. 11. Interactive demo of some run-time rules.

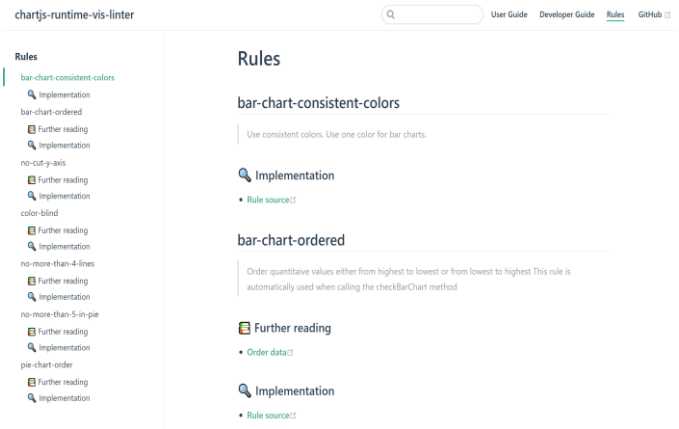

Fig. 12. Visualization Linter instruction.

The implementation of the lint rules in Visualization Linter typically involves the inspection of the chart object on Chart.js. The object of the chart is an encapsulated visualization node in the HTML DOM tree. Encapsulation means that we cannot inspect the structure of the chart by examining its DOM tree. For example, after creating a line chart with Chart.js, the user can only see a canvas tag in the HTML file body part, not presenting each part of the chart as a single DOM node. Figure 13 illustrates this idea clearly. Therefore, the user cannot explicitly select a line from the chart by selecting its node.

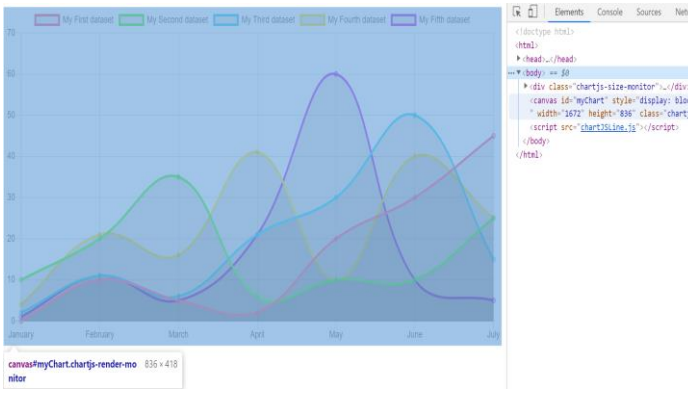

Fig. 13. Chart.js canvas node.

The way we implement the rules for Chart.js is by analyzing the properties of the chart object. Figure 14 indicates how to implement the rule "Set the height of a line chart such that the data in the line chart takes up approximately two-thirds of the y-axis' maximum scale." We first obtain the y-axis range by checking the "end" property of the "y-axis-0" of the scales property of the chart object. Then get the lines from checking the dataset property. After retrieving the maximum value of all the lines, we compare this value to the range of the y-axis. In this way, we define the function to check if the user violates the rule.

```
function checLineRange(lineChart){
    var yRange = lineChart.scales["y-axis-0"].end;
    var lines = lineChart.chart.config.data.datasets;
    var yMax;
    for (let index = 0; index < lines.length; index++) {
        const data = lines[index].data;
        yMax = data[0];
        for (let j = 0; j < data.length; j++) {
            const element = data[j];
            if(yMax<element){
                yMax = element;
            }
        }
    }
    if(yMax>(2/3*yRange)){
        console.error("Set the height of a line chart such that the data in \
        the line chart takes up approximately two-thirds of the y-axis' maximum scal
    }
}
```
Fig. 14. The implementation of one rule in Chart.js.

Table 2 contains all the implemented rules of the run-time analysis tool.

Table 2. Implemented rules

No.	Category	Example
1	Line chart	Don't plot more than 4 lines in one chart. If you need to display more, break them out into separate charts for better comparison.
2	Line chart	Set the height of a line chart such that the data in the line chart takes up approximately two-thirds of the y-axis' maximum scale.
3	Line chart	Use solid lines only because dashed and dotted lines can be misleading.
4	Line chart	Always include a Zero Baseline if possible.
5	Bar chart	Use horizontal labels. Avoid steep diagonal or vertical type, as it can be difficult to read.
6	Bar chart	Use consistent colors. Use one color for bar charts. You may use an accent color to highlight a significant data point.
7	Bar chart	Order data appropriately. Order categories alphabetically, sequentially, or by value.
8	Colors in vis	Don't use more than 6 colors together.
9	Colors in vis	Double-check your colors for the color blind.
10	Pie chart	Visualize no more than 5 categories per pie chart.
11	Pie chart	Order slices correctly by first placing the largest section at 12 o'clock stretching clockwise, and then placing the remaining sections consistently either clockwise or counter-clockwise in descending order.
12	Bar chart	Do not cut y-axis in bar chart.
13	Line chart	Do not cut y-axis in line chart.

4.3 Educating users

Besides linting, we were interested in educating users of our library and plugin on data visualization best practices. We have added a link for each warning message which links to our documentation. In our documentation, we provide a description of the rule, the code source of the rule, and a test source of the rule. More importantly, we provide further readings on the reasoning behind using a rule. We used vuepress which is a static site generator to build the documentation rapidly. We cloned "eslint-plugin-vue" project, which uses vuepress, and dissected the vuepress related code and scripts from it. We have built a documentation for both the eslint plugin[4] and the runtime library[18].

4.4 Building demo and packages

We used the package "storybook/html" library to develop the online demo[20]. Storybook[19] helps users develop isolated UI components. We have built an npm package for the eslint-plugin and for the chartjs-runtime-library. This helps other users to use our code easily. There is still an issue with the runtime library package which we believe needs about 4 hours of work to be resolved.

5 Discussion and Future Work

We believe that our visualization linter provides a good starting point for providing chart creators with direct feedback during their visualization implementations. The future improvements for this vis linting tool are as follows.

- Write friendly warning messages. Warning messages are the first interaction between the user and us. These messages should be well thought-of. A friendly and understandable warning message has a better chance to help the user continue to use our library and would make the user more enthusiastic to know more about data visualization best practices.
- If we would publish our work in a paper, we need to be more methodological on what rules to implement. We basically picked rules that are easy to implement and eye-catching(like the color-blind rule). To be more methodological, we should have had a clear strategy on what repositories to use
- Ability to turn on or turn off rules. This feature is already implemented in the Eslint plugin. We should include this feature for the runtime library too.
- Ability to have categories that contain a group of rules. As mentioned before, this is already implemented by Eslint, but we would like to have it for the runtime library too
- Ability to switch off the dynamic checks in production for better performance. Dynamic checks should only be incorporated during development to inform the first-hand user (developer). In production, the end user would probably care more about performance than from these console warnings.
- Implement more rules. Currently, our rules are far away from being conclusive. Tens or even hundreds of important rules need to be implemented.
- Higher quality resources for each rule. Currently, we provide a few links to each rule, but we think we can invest more time in these resources. Ideally, all resources should have been reviewed for accuracy.
- Add tests for edge cases for the library; We believe that unit tests are important for at least most of the rules. Each rule can have its own edge cases. Besides that, a rule can have conditions

where it does not need to be enforced. We believe more thought should be invested on those edge cases

Conclusion

We have described that visualization linting as an effective way to help scientists and others who create charts to improve their visualization's effectiveness. We have analyzed the reason why we choose to make a vis linter for Chart.js instead of D3.js. We have shown the feasibility by implementing rules to Chart.js and develop a prototype. Future developments in our visualization tool will allow users to produce better charts and help users become more educated about data visualization best practices.

References

[1] A McNutt and G Kindlmann. Linting for Visualization: Towards a Practical Automated Visualization Guidance System. 2nd IEEE VIS Workshop on Creation, Curation, Critique and Conditioning of Principles and Guidelines in Visualization ("VisGuides"), October 2018.

[2] Jonathan Harper, Maneesh Agrawala. Deconstructing and Restyling D3 Visualizations. UIST '14 Proceedings of the 27th annual ACM symposium on User interface software and technology, pages 253-262.

[3] Extracting Data and Structure from Charts and Graphs for Analysis, Reuse and Indexing. http://graphics.stanford.edu/projects/dataExtract/.

[4] Visualization Guidelines Repository. http://visguides.repo.dbvis.de/guidelines.php.

[5] Stephen Ingram, Interactive Visualization of Small World Graphs in Prefuse.https://www.cs.ubc.ca/~tmm/courses/cpsc533c-05-fall/projects.html#sfingram.

[6] A collection of dataviz caveats. https://www.data-to-viz.com/caveats.html.

[7] D3 Gallery. https://github.com/d3/d3/wiki/Gallery

[8] Tamara Munzner. "Visualization Analysis and Design". CRC Press, 2014.

[9] Data Visualization 101:How to design charts and graphs. https://cdn2.hubspot.net/hub/53/file-863940581-pdf/Data_Visualization_101_How_to_Design_Charts_and_Graphs.pdf

[10] Data Visualization: Chart Dos and Don'ts. https://guides.library.duke.edu/datavis/topten.

[11] Tableau. Tableau. https://www.tableau.com/. Accessed: 2019-12-10

[12] Spotfire. Spotfire. https://spotfire.tibco.com/. Accessed:2019-12-10.

[13] E.Meeks. Linting Rules for Complex Data Visualization. https://www.youtube.com/watch?v=_KEl-Spdaz0, 5 2017.

[14] https://esprima.org/

[15] https://eslint.org/docs/developer-guide

[16] https://github.com/dgraham/eslint-plugin-jquery#readme

[17] https://eslint-plugin-d3.now.sh/

[18] https://chartjs-runtime-vis-linter.now.sh/

[19] https://storybook.js.org/

[20] https://chartjs-runtime-vis-linter-demo.now.sh/?path=/story/demo--green-with-red-line-charts