Visualization Linter

Static and runtime check tool for D3.js

Youssef Sherif
Wei Zheng
Background

- Why do we need a visualization linter?
  - To help the “average Joe” data visualization user adhere to visualization best practices.
- Why two tools?
  - Each of the static analysis tool and run-time library have its own uses, pros, and cons.
Static Analysis Tools

• Not meant to check data-related issues
  ○ Why? Because data is dynamic
    ■ Example: Http request from a backend server

• Meant to check for logical problems
Runtime Tools

- Has access to data during runtime
- Cons
  - Warnings and Errors are displayed on runtime (not immediately)
Programming Language and Framework:

- **JavaScript**
  - Web applications are on the rise
    - JavaScript is the default web language
- **D3.js**
  - Most popular
  - Open source
Previous Works

Andrew Mcnutt’s Vislinter

- Run-time library checker for Matplotlib
- He proposed a long list of data visualization rules
- Implemented few of them
- Wrote “Linting for Visualization: Towards a Practical Automated Visualization Guidance System” paper
Existing JavaScript Runtime libraries checkers

- Check most popular run time checker libraries on npm
- We are planning to check the public API for at least one of these libraries to conform to best practices for runtime library checkers
## Matplotlib vs D3.js

<table>
<thead>
<tr>
<th>Matplotlib</th>
<th>D3.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>High level library</td>
<td>Low-level library</td>
</tr>
<tr>
<td>Less control</td>
<td>More control</td>
</tr>
<tr>
<td>Easier to build simple visualizations</td>
<td>Harder to build simple visualizations</td>
</tr>
<tr>
<td>Can easily infer statically the visualization the user wants to build</td>
<td>Hard to infer statically the visualization the user wants to build</td>
</tr>
</tbody>
</table>
Implementation

- Static Analysis Tool
  - ESLint plugin
    - ESLint is the most widely used JavaScript pluggable static linter

- Run-time library checker
  - A regular npm package
What are our personal expertise?

Youssef

- Worked as a full-stack web developer
  - Used JavaScript and other JavaScript libraries
- Partially built static analysis tools

William

- Experienced in Data Analysis with Python, and visualization tools, including Matplotlib and Seaborn
- Built a web app using vanilla JavaScript
- Applied machine learning algorithms with Java
What we are supposed to do?

Youssef

- Build the static analysis tool
- Structure the runtime checker library and set the public API
- Set webpack and npm scripts to be used for the library

William

- Select the rules for runtime checking
- Implement the runtime check part
Resources for Rules for Best Practices

- Tamara’s book “Visualization Analysis and Design”
  - Example: order of effectiveness
- The Visualization Guidelines Repository
- Yan Holtz’s online guideline
Scenarios of Use

Static analysis tool

● Run a command line prompt

Run-time library checker

● Check console warnings
Future Advancements

Static analysis tool

- IDE extension

Run-time library checker

- Unobtrusive toasts
Attempt to implement a rule

“No horizontal labels”
Tried to implement the rule statically

**Current Solution**: use node.js to parse the entire js file written by users as string and detect key words such as `.selectAll("text")`, `.attr("transform", "rotate(-90)")` to detect the part which users try to deal with the labels of x-axis.

**Problem**: This would not work if the text is the same as it is. For example using a variable and the string ‘text’ would make our tool fail. This is where runtime checks shine.