line-chart-comp-d3

line-chart-comp-d3 is a Vue.js (>= 2.5) web component that draws svg scatter/line charts. **line-chart-comp-d3** depends on the vue.js, various modules from d3.js, along with button-comp, input-comp, and select-comp from the deandevl repositories. The dependencies can be installed via npm install with the included `package.json` file. **line-chart-comp-d3** offers several features including:

- clickable x or y axis for redefining the axis scaling
- legends and axis titles are draggable to new locations
- clickable lines where the selected line is highlighted and the others dimmed
- can plot numeric, time or categorical based data
- tooltip showing a points x/y values on hover
- control over decimal places for both axis and tooltip values
- points can be line connected/unconnected
- CSS variables are provided for easily controlling chart colors, backgrounds and font sizes
- data can be sub-divided into groups with specific colors and icons
- curve fit functions can defined and plotted along with the points

A demo folder is provided that used Parcel together with its associated `package.json` file to bundle together **line-chart-comp-d3** along with its dependencies for a simple application. Further details are provided below for running the demo.

Props

A prop in Vue.js is a custom attribute for passing information from a parent component hosting **line-chart-comp-d3** instance(s) to an **line-chart-comp-d3** as a child component. **line-chart-comp-d3** has the following props for a parent to bind and send information to:

- chart_data -- an array of javascript objects with pairs of variable names for keys and a string/numeric value as the keys ' values
- axis -- an object for defining properties for both the x and y axis including title, data_type, key, tic_format, and value_format. Below is an example from demo_1 for Fisher's Iris data:

```
axis: {
        x: {
          title: 'Petal Length',
          data_type: 'linear',
          key: 'petal_length',
          tic_format: '.1f',
          value_format: '.1f'
        },
        y: {
          title: 'Sepal Length',
          data_type: 'linear',
          key: 'sepal_length',
          tic_format: '.1f',
```

```
          value_format: '.1f'
        }
```

- points -- an object for defining properties for points or groups of points including `text`, `select`(function), `icon`, `color`, `connect`(boolean). Below is an example from demo_1 for Fisher's Iris data:

```
points: [
        {
          text: 'Iris-setosa',
          select:
            function(d) {
              if(d['class'] === 'Iris-setosa') {
                return true;
              }
            },
          icon: '\u2605',
          color: 'red',
          connect: false
        },
        {
          text: 'Iris-versicolor',
          select:
            function(d){
              if(d['class'] === 'Iris-versicolor') {
                return true;
              }
            },
          icon: '\u26AB',
          color: 'white',
          connect: false
        },
        {
          text: 'Iris-virginica',
          select:
            function(d){
              if(d['class'] === 'Iris-virginica') {
                return true;
              }
            },
          icon: '\u2206',
          color: 'yellow',
          connect: false
        }
      ],
```

- fits -- an array of objects that defines curve fit functions for plotting with properties for the equation function, equation text, number of points to plot, and color. See the 'Demonstration' section below for an example defining this property.
- title_1 -- a string for the chart's main title
- title_2 -- a string for the chart's sub title
- margin_left -- defines the number of pixels for the chart's left margin (default is 80)
- margin_bottom -- defines the number of pixels for the chart's bottom margin (default is 50)
- css_variables -- a javascript object that defines the css variables (see below)

Note that for the `icon` and `connect` properties for objects under the `points` array, either one is optional but as least one or both of them must be defined or there is no graph of those points.

Styling

The **css_variables** prop is a javascript object that contains any combination of css variable names as keys and associated values. The following list are the css variable names along with their default values for a quick styling of **line-chart-comp-d3**:

```
{
    line_chart_compD3_font_family: Verdana,serif,
    line_chart_compD3_color: black,
    line_chart_compD3_background_color: white,

    line_chart_compD3_axis_font_size: .6rem,
    line_chart_compD3_axis_color: black,

    line_chart_compD3_icon_opacity: .6,

    line_chart_compD3_tooltip_fill: black,

    line_chart_compD3_line_width: 1.0
}
```

Interactivity

Clicking either the x or y axis brings up a set of inputs appearing at the top left of the chart. Clicking the axis a second time hides the inputs. For the y axis the inputs are `y minimum`, `y maximum`, and `y step` from which you can control the range and step size (in units of the y variable) of the axis. The minimum and maximum input boxes are always filled in with the current values. You can enter a new minimum/maximum pair and click the `Update` button to update the axis. Clicking the `Reset` button returns the axis back to its auto formatted scaling. If the x axis is `linear` then these same input boxes will appear with their current values.

If the x axis is `time` based (as in `demo_2`, `demo_3`) then the minimum/maximum input boxes are NOT filled in. The reason for this is that their inputs must agree with the % time directive you enter for the `x tick format` input box. The `step` is a whole number and refers to the number of step time units (defined in the next input box `step time units`) between tics. The choices for step units are `auto`, `year`, `month`, `week`, `day`, `hour`, `minute`, `second`, and `millisecond`. The final time related input box is `x tick format` containing a % time directive formed from those listed at d3.time. Again, the inputs to minimum/maximum must agree with the `x tick format`.

Demonstration

Three demonstrations of **line-chart-comp-d3** are provided in the folders named `demo_1`, `demo_2`, and `demo_3`. They can be viewed by hosting their respective `index.html` files. demo_1 is a chart of the Fisher Iris data grouped into the three major classes. A pair of fitted regression lines are also plotted.

As a suggestion, install http-server locally/globally via npm then enter the command `http-server` in a **line-chart-comp-d3** `dist` directory. From a browser enter the url: `localhost:8080/` to view the demo.

The demo folder contains a `package.json` file that can be used to setup dependencies for this demo and as a template for other applications using **line-chart-comp-d3**.

The following is the setup for this chart contained in the entry.js file where the `axis` and `points` properties are as we defined in the above examples:

```
    <line-chart-comp-d3>
        title_1="Fisher's Iris Flower Data"
        title_2="(petal_length vs sepal_length)
        :chart_data="chart_data"
        :axis="axis"
        :points="points"
        :fits="fits"
        :css_variables="css_variables">
        <svg class="svg_chart_1" width="1200" height="800"></svg>
      </line-chart-comp-d3>
```

Note how the `line-chart-comp-d3` tag wraps around the `<svg>` element. It is important that a class be assigned to the `svg` for **line-chart-comp-d3** to locate the element.

Among line-chart-comp-d3's attributes, it makes a reference to `chart_data` for the chart_data attribute. `chart_data` is defined using a function called `read_csv` (from the d3fetchmodule module). Below is some of the code for reading the iris.csv file in the demo:

```
const get_data = async () => {
        const convert = [
          {field: 'petal_length', type: 'linear'},
          {field: 'sepal_length', type: 'linear'},
          {field: 'class', type: 'head'}
        ];
        try {
          this.chart_data = await read_csv('iris.csv',convert);
          //debug
          //console.log(JSON.stringify(this.chart_data));
          //define 2 fits
          const regress_data = [];
          this.fits = [];
          for(let row of this.chart_data){
            regress_data.push([row['petal_length'],row['sepal_length']]);
          }
          const regress_1 = regression.linear(regress_data);
          const fit_1 = {};
          fit_1.text = regress_1.string + `  (R2: ${regress_1.r2})`;
          fit_1.equation = regress_1.predict;
          fit_1.number_of_points = 50;
          fit_1.color = 'violet';
          this.fits.push(fit_1);

          const regress_2 = regression.polynomial(regress_data);
          const fit_2 = {};
          fit_2.text = regress_2.string + `  (R2: ${regress_2.r2})`;
          fit_2.equation = regress_2.predict;
```

```
            fit_2.number_of_points = 100;
            fit_2.color = 'lime';
            this.fits.push(fit_2);
          }catch(e){
            console.log(e);
          }
        };
        get_data();
```

Note that we are defining and referencing the `fits` array property with two objects that define a linear and polynomial regression. The `regression.linear` and `regression.polynomial` functions come from Tom Alexander .