stacked-area-chart-comp-d3

stacked-area-chart-comp-d3 is a Vue.js (>= 2.5) web component that draws svg stacked area charts. **stacked-area-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. **stacked-area-chart-comp-d3** offers several features including:

- clickable x and y axis for redefining the axis scaling
- legends and axis titles are draggable to new locations
- axis tic label formatting/rotation for better readability
- chart becomes just a regular area chart when just one group is being displayed
- CSS variables are provided for easily controlling chart colors, backgrounds and font sizes

stacked-area-chart-comp-d3 can be installed via with the included `package.json` file for a local installation via the npm install command. **stacked-area-chart-comp-d3** depends on the vue.js and modules from the d3 frameworks. A demo folder is provided that used Parcel together with its associated `package.json` file to bundle together **stacked-area-chart-comp-d3** along with its vue.js and d3 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 **stacked-area-chart-comp-d3** instance(s) to an **stacked-area-chart-comp-d3** as a child component. **stacked-area-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 -- a javascript object defining the x and y axis' -- see below
- title_1 -- a string for the chart's main title
- title_2 -- a string for the chart's sub-main 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)

The `axis` property is central to the component's setup -- here is an example from the demo:

```
axis: {
        x: {
          title: 'Year',
          data_type: 'time',
          key: 'year',
          tic_format: '%Y',
          tic_rotation: 0
        },
        y: {
          title: 'Dollars',
          tic_format: '.3s',
          groups: [
            {key: 'National_Defense', format: '.1f', color: '#ff8c00'},
```

```
            {key: 'International_Affairs', format: '.1f', color: '#d0743c'},
            {key: 'Environment', format: '.1f', color: '#a05d56'},
            {key: 'Science_Space_Tech', format: '.1f', color: '#6b486b'},
            {key: 'Agriculture', format: '.1f', color: '#7b6888'}
          ]
  }
```

The data_type for the x axis can be either `linear` (i.e. numerically continuous) or `time` based. The y axis is always `linear`. The x.key/y.groups.key properties refers to the row object key in the data file. Note that for the `y` property we can have just one group or an array of groups with objects defining the row key, output format, and fill color for the area. Each entry in y.groups is a area on the chart stacked in the order in which they are listed. For the `tic_format` property values for x and y refer to d3.format for a `linear` axis and the % time directives listed at d3.time for a time based axis. Note that the x axis `data_type` in this example is `time`. Other acceptable values are `linear` for continuous data and `band` for categorical/ordinal data.

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 **stacked-area-chart-comp-d3** :

```
{
  stacked_area_chart_compD3_font_family: Verdana,serif,
  stacked_area_chart_compD3_color: black,
  stacked_area_chart_compD3_background_color: white,

  stacked_area_chart_compD3_axis_font_size: .8rem,
  stacked_area_chart_compD3_axis_color: black,

  stacked_area_chart_compD3_fill_opacity: 1.0,

  stacked_area_chart_compD3_tooltip_fill: black
}
```

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 the two demos) 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`.

As an example of updating a time based axis, in `demo_2` the raw time data is formatted with a time directive of `%Y-%m-%d` providing a resolution down to the day. Say we want to see 7 days in May of the year 1990. To do this enter `%Y-%m-%d` in the `x tick format` input box. Then following this format enter the dates `1990-05-01` and `1990-05-07` in the minimum and maximum input boxes respectively. Enter `1` for the `x step` input and select `day` for the `step time unit` input. Click the `Update` button to see a new scaling of the x axis and chart area with just the 7 days between the minimum/maximum dates. Keep in mind that the degree of resolution in your raw time data dictates how refined you can form your % time directive for the `x tick format` and specify minimum/maximum.

Demonstration

Two demonstrations of **stacked-area-chart-comp-d3** are provided in the folders named `demo_1`, `demo_2`. It can be viewed by hosting their `index.html` file. `demo_1` shows expenditures over time for several US government functions (source is Office of Management and Budget).

As a suggestion, install http-server locally/globally via npm then enter the command `http-server` in the **stacked-area-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 **stacked-area-chart-comp-d3**.

The following is the setup for this chart contained in the entry.js file where `axis` is a reference to the `axis` in our above example:

```
<stacked-area-chart-comp-d3
        title_1="Federal Expenditures by Function"
        title_2="US Office of Management and Budget Historical Tables"
        :axis="axis"
        :chart_data="chart_data"
        :css_variables="css_variables">
        <svg class="svg_chart_1" width="1200" height="700"></svg>
</stacked-area-chart-comp-d3>
```

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

Among `stacked-area-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 data.csv file:

```
read_data: function(){
      const convert = [
        {field: 'year', type: 'time', time_format: '%Y'},
        {field: 'National_Defense', type: 'linear'},
        {field: 'International_Affairs', type: 'linear'},
        {field: 'Environment', type: 'linear'},
        {field: 'Science_Space_Tech', type: 'linear'},
        {field: 'Agriculture', type: 'linear'}
      ];
```

```
    const get_data = async () => {
      try{
        this.chart_data = await read_csv('federal_spend.csv', convert);
        //debug
        console.log(JSON.stringify(this.raw_data));
      }catch(e){
        console.log(e);
      }
    };
    get_data();
  }
```