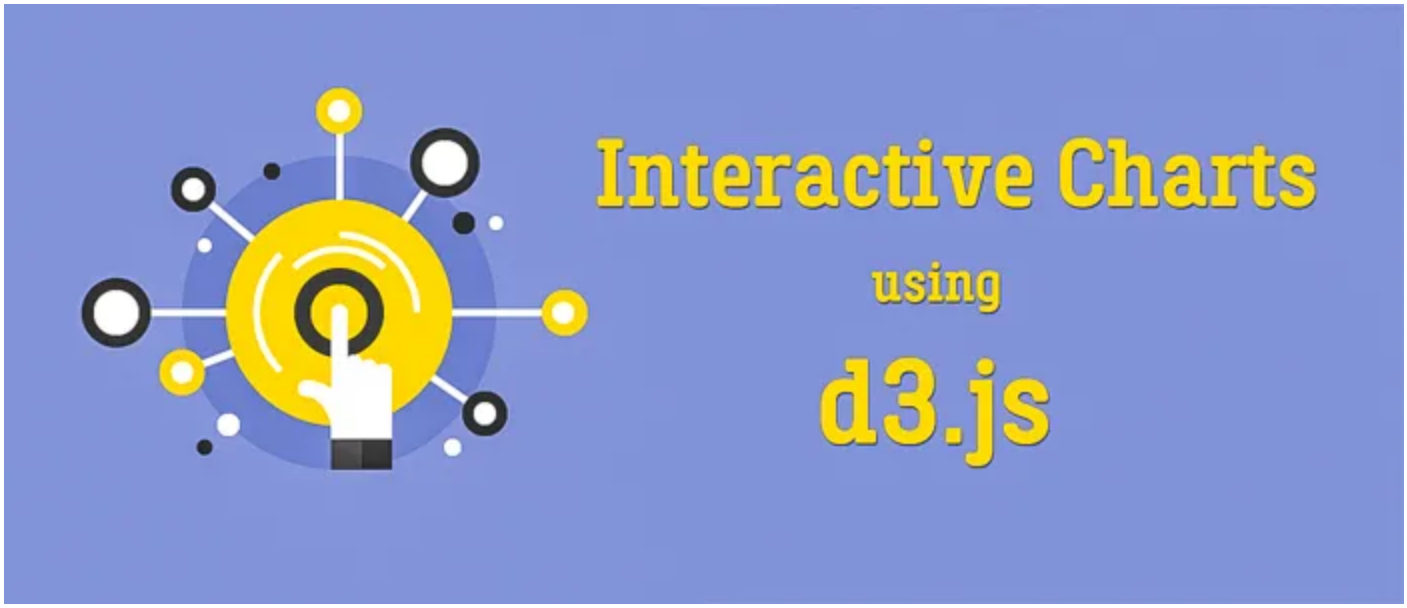
October 10, 2016 By Abhisek Jana — Leave a Comment (Edit)

How to Create Interactive Charts using D3.js



Today we will learn How to Create Interactive Charts using D3.js. In order to make our learning close to real life problem we will use the data provided by Nasa and use that to create our Interactive Charts. We will be using GISS Surface Temperature Analysis data provided by National Aeronautics and Space Administration. You can find more details at http://data.giss.nasa.gov/gistemp.

We will use GISS Surface Temperature Analysis data to display the temperature rise over time in three different region – Northern Hemisphere, Southern Hemisphere and Global Average. The data provided by NASA is not really formatted to be used in JavaScript right away, so it has been formatted as a JSON object. The `data/data.js` has the raw data we will be using (Find the full codebase in GitHub, link provided at the end of the article).

Data Structure:

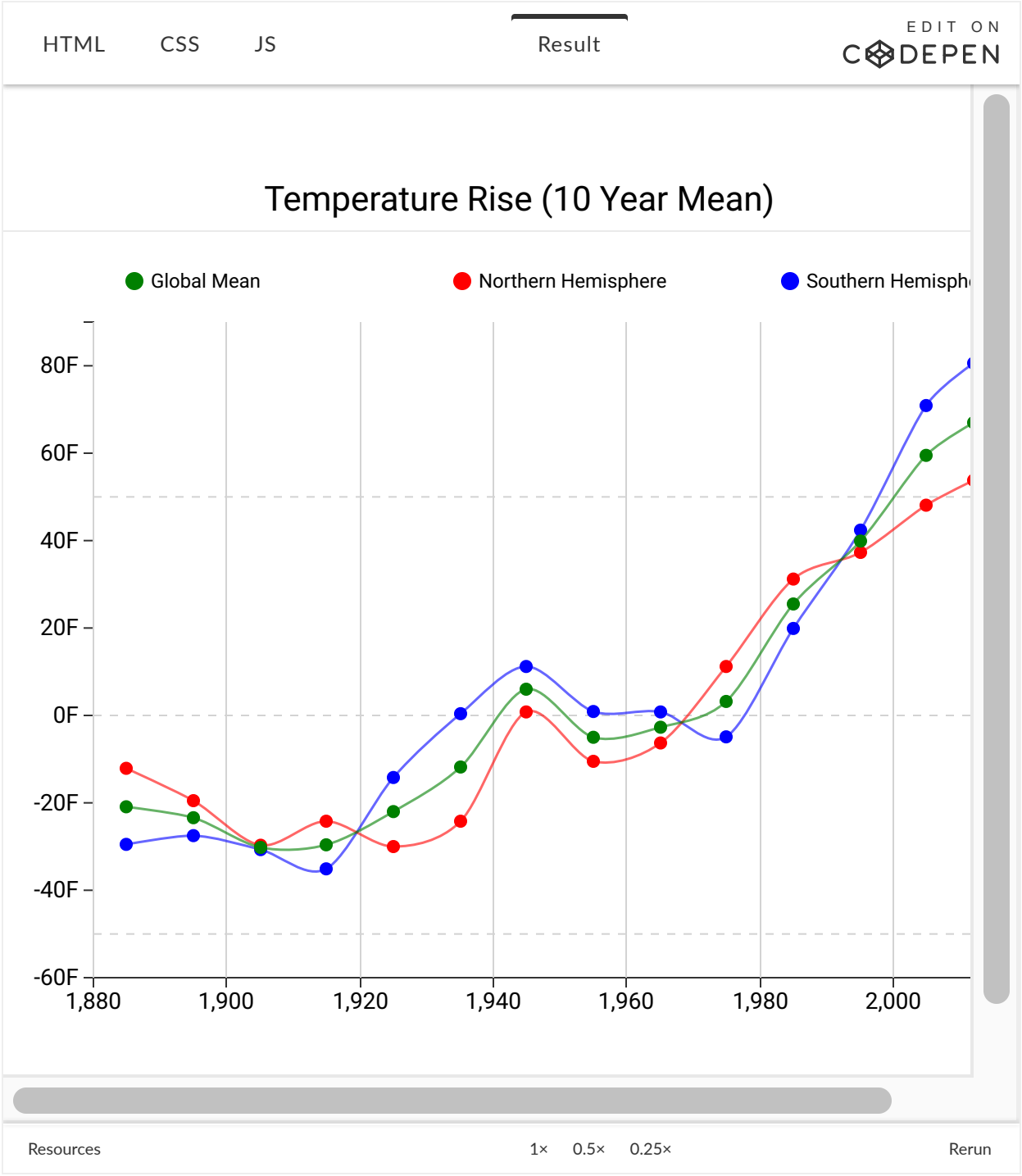
Lets take a look at the data first. We have the Year, Global (`Glob`) , `NHem` and `SHem` which we will be using. You can ignore rest of the elements. We have total 135 elements in the data array. Displaying all of them at once would make the visualization very complicated, so will average 10 years of data can calculate the mean of the temperatures.

```
var data = [{
        "Year": 1880,
        "Glob": -19,
        "NHem": -33,
        "SHem": -5,
        "24N-90N": -38,
        "24S-24N": -16,
        "90S-24S": -5,
        "64N-90N": -89,
        "44N-64N": -54,
        "24N-44N": -22,
        "EQU-24N": -26,
        "24S-EQU": -5,
        "44S-24S": -2,
        "64S-44S": -8,
        "90S-64S": 39
    },
    {
        "Year": 1881,
        "Glob": -10,
        "NHem": -18,
        "SHem": -2,
        "24N-90N": -27,
        "24S-24N": -2,
        "90S-24S": -5,
        "64N-90N": -54,
        "44N-64N": -40,
        "24N-44N": -14,
        "EQU-24N": -5,
        "24S-EQU": 2,
        "44S-24S": -6,
        "64S-44S": -3,
        "90S-64S": 37
```

```
},
.....
.....
```

Final Demo:

Before we move further, lets see the chart we will be building. Here we will display 10Years of average data, then on `mouseover` to any of the line chart we will highlight the specific line chart and then display the data for each year (total 135 years). Again on mouse out we will reset the details.



Calculate the Mean/Average:

We have to calculate the Mean/Average of 10 years of temperature from the dataset we have. We will create a function named `getMean()` and pass the JSON Element name and the number of years (in this case 10) as the argument to the method. `data` is available globally. I am also looking for `"NA"` just to make sure the we dont have any missing data point.

```
function getMean(dataPoint,count) {

    var mean=[];

    for (var i = 0, k = 0; i < data.length; i = i +
count, k++) {
        var total = 0;
        for (var j = 0; j < count; ++j) {

            if (data[i + j] != undefined && data[i + j]
['dataPoint'] != "NA") {
                total += data[i + j][dataPoint];
            } else {

            }
        }
        if (data[i] != undefined && data[i][dataPoint] !=
"NA") {
            if(i<130)
                mean[k] = {val: total / count, "Year":
data[i+count/2].Year};
            else {
                // This can be generalized even further
                mean[k] = {val: total / (count/2),
"Year": data[i+2].Year};
            }
        }
    }
    return mean;
}
```

Note :

Now since we have 135 years of data, and we are calculating mean for every 10 years we can only procceed till 130 years, thats from 1980 to 2009. Since the last set has only 5 years, we need to take that as an exception and calculate the mean for 5 years only.

The `getMean()` function would return an array object with the mean temparature and the year.

Setup Chart Elements:

Next we will setup the default chart elements, such as margin, x, y, xAxis, yAxis, xGrid, yGrid, svg etc. If you need to understand more about them please refer my earlier tutorials where I have explained in detail.

```
        var margin = {top: 50, right: 35, bottom: 50,
left: 50},
            w = 630 - (margin.left + margin.right),
            h = 500 - (margin.top + margin.bottom);

    var x = d3.scale.linear()
            .domain(d3.extent(data, function (d) {
                return d.Year;
            }))
            .rangeRound([0, w]);

    var y = d3.scale.linear()
            .domain([-60,90])
            .range([h, 0]);

    var xAxis = d3.svg.axis()
            .scale(x)
```

```
            .orient('bottom')
            .ticks(10);

        var yAxis = d3.svg.axis()
            .scale(y)
            .orient('left')
            .ticks(10)
            .tickFormat(function(d){
                return d+"F";
            });

        var xGrid = d3.svg.axis()
            .scale(x)
            .orient('bottom')
            .ticks(5)
            .tickSize(-h, 0, 0)
            .tickFormat("");

        var yGrid = d3.svg.axis()
            .scale(y)
            .orient('left')
            .ticks(5)
            .tickSize(-w, 0, 0)
            .tickFormat("");

        var svg = d3.select("#chart").append('svg')
            .attr({
                width: w + margin.left + margin.right,
                height: h + margin.top + margin.bottom
            })
            .append('g')
            .attr('transform', 'translate(' + margin.left
    + ',' + margin.top + ')');
```

```
svg.append('g')
        .attr({
            class: "x axis",
            transform: 'translate(0,' + h + ')'
        })
        .call(xAxis);

svg.append('g')
        .attr({
            class: "y axis"
        }).call(yAxis);

svg.append('g')
        .attr({
            class: "grid",
            transform: 'translate(0,' + h + ')'
        })
        .call(xGrid);

svg.append('g')
        .attr({
            class: "y-grid"
        }).call(yGrid);
```

Draw the Lines

Now lets draw 3 lines, one for each mean array. We will create a function named `plotLine()` so that we can avoid duplicate code needed for all three line charts.

At first call our `getMean()` function and create 3 array objects for the 3 lines.

```
var count=10;

var Glob=getMean('Glob',count);
var NHem=getMean('NHem',count);
var SHem=getMean('SHem',count);
```

The `plotLine()` would take the `mean_data` and class names as arguments. We need to pass different class name to have different color for each line chart. We will create the `line` object and draw the path using d3's built-in functions.

We will also highlight each year by drawing a scatter plot so that we can interract with the chart using it later.

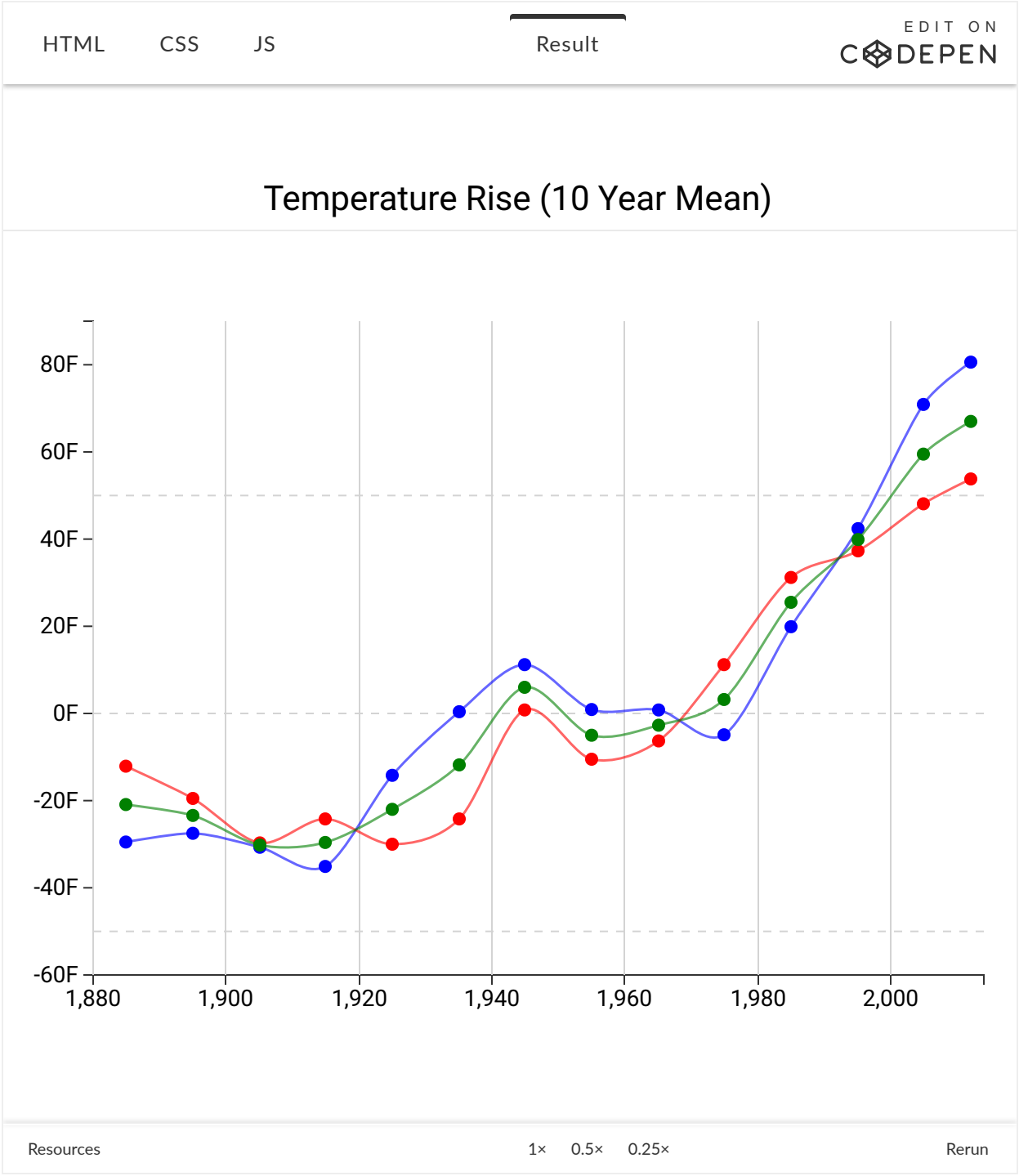
```
function plotLine(mean_data,className,cirClass,attr){

  var line = d3.svg.line()
  .interpolate("cardinal")
  .x(function (d) {
    return x(d.Year);
  })
  .y(function (d) {
    return y(d.val);
  });

  svg.append('path')
    .datum(mean_data)
    .attr({
    class: className,
    d: line
  });
```

```
  svg.selectAll(".dot")
    .data(mean_data)
    .enter().append("circle")
    .attr("class", cirClass)
    .attr("cy", function (d) { return y(d.val); } ) //set
y
    .attr("cx", function (d,i) { return x(d.Year); } )
//set x
    .attr("r", 4);
 }
```

If you execute whatever we have written so far you can see all three line charts are getting drawn. Here is the demo.



Adding Interactivity:

So we want to change the charts on `mouseover` and reset them on `mouseout`. However before that we shall draw the line with all the 135 data points and hide the chart.

Like the mean line chart, we will add another line chart with the global `data` object itself but would hide it using css class. Here is the code.

```
var lineAllData = d3.svg.line()
        .interpolate("cardinal")
        .x(function (d) {
            return x(d.Year);
        })
        .y(function (d) {
            return y(d[attr]);
        });

svg.append('path')
        .datum(data)
        .attr({
            class: className+"0",
            d: lineAllData
        });
```

Next on mouseover we will first reduce the opacity of all the 3 charts. Then increase the radius of the selected circle (We will do this by `event.target`). Then we will change the opacity & thickness of the selected line chart by the classname passed.
We do the same for the scatter plot (dots) also. At the end we will set the `opacity` of the 2nd line chart to `1.0`. This will display the chart with actual data set with 135 data elements.

```
svg.selectAll(".dot")
    .data(mean_data)
    .enter().append("circle")
    .attr("class", cirClass)
    .attr("cy", function (d) { return y(d.val); } ) //set
y
    .attr("cx", function (d,i) { return x(d.Year); } )
//set x
```

```
    .attr("r", 4)
    .on("mouseover", function(d) {
                d3.select(event.target).attr("r", 7);

                d3.select('.line1').style("opacity",
".1");
                d3.select('.line2').style("opacity",
".1");
                d3.select('.line3').style("opacity",
".1");
                d3.select('.'+className).style("opacity",
"1");
                d3.select('.'+className).style("stroke-
width", "2px");

                d3.selectAll('.dot1').style("opacity",
0.1);
                d3.selectAll('.dot2').style("opacity",
0.1);
                d3.selectAll('.dot3').style("opacity",
0.1);

d3.selectAll('.'+cirClass).style("opacity", 1);


d3.select('.'+className+"0").style("opacity", 1);
        })
```

Now on mouseout we need to reset everything. Here is the code.

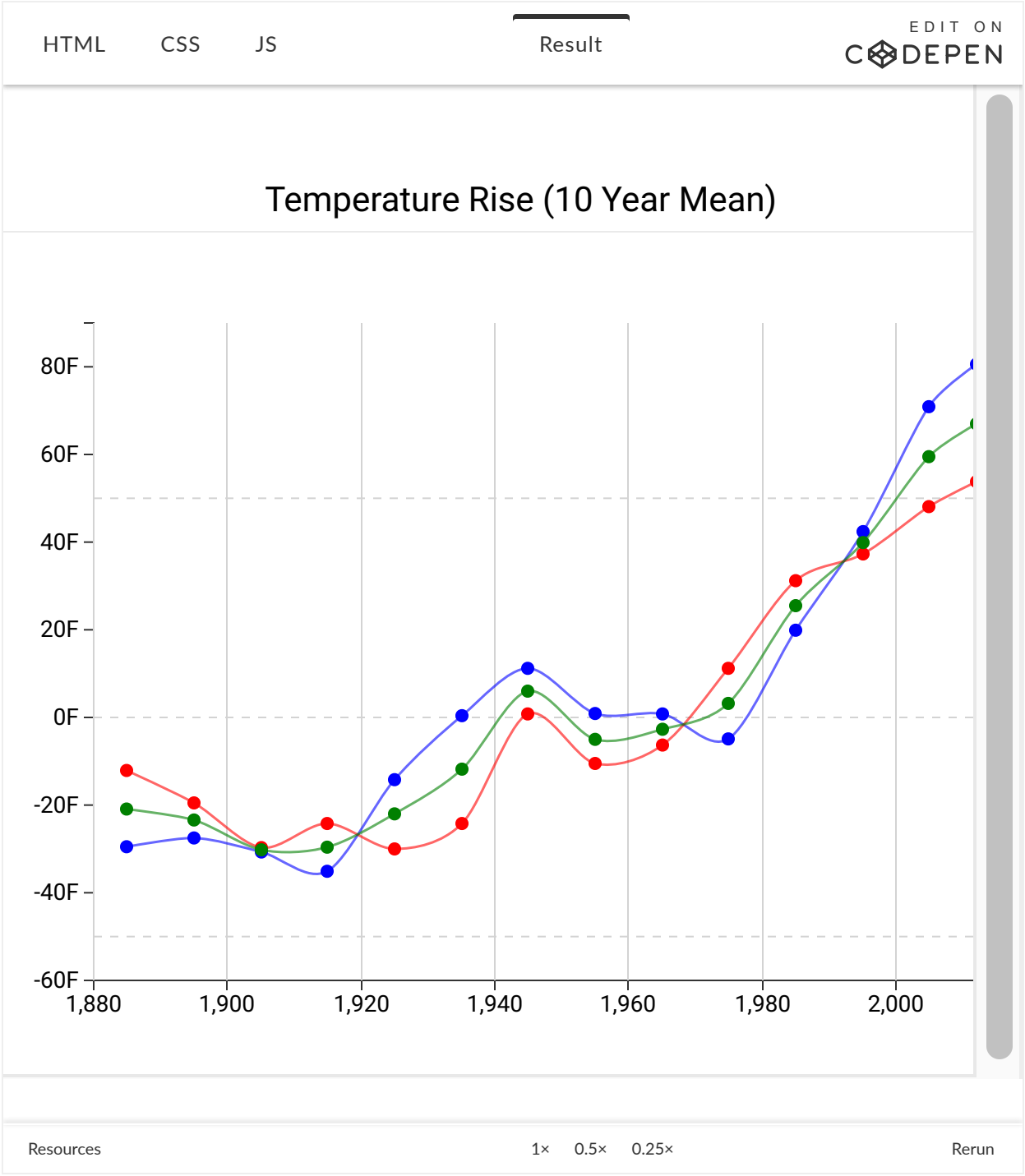
```
.....
.on("mouseout", function(d) {
        $(event.target).attr("r", 4);
```

```
        d3.select('.line1').style("opacity", ".6");
        d3.select('.line2').style("opacity", ".6");
        d3.select('.line3').style("opacity", ".6");
        d3.select('.'+className).style("stroke-width",
"1.5px");

        d3.selectAll('.dot1').style("opacity", 1);
        d3.selectAll('.dot2').style("opacity", 1);
        d3.selectAll('.dot3').style("opacity", 1);

        d3.select('.'+className+"0").style("opacity", 0);
});
```

Here is the demo of the so far completed code. Let me know if you didn't understand any of it.



Add More Visual Elements:

We will add few more visual elements to make the chart look pretty. First we will the gradient which you can see after mouseover. We need to create an area chart and add svg gradient filter to it.

Gradient:

We will first create a function to add svg gradient.

```
var createGradient=function(svg,color1,color2,id){

        var defs = svg.append("defs");

        var gradient=defs.append("linearGradient")
                .attr("id", id)
                .attr("x1", "0%")
                .attr("y1", "100%")
                .attr("x2", "0%")
                .attr("y2", "0%")
                .attr("spreadMethod", "pad");

        gradient.append("svg:stop")
                .attr("offset", "0%")
                .attr("stop-color", color1)
                .attr("stop-opacity", 1);

        gradient.append("svg:stop")
                .attr("offset", "100%")
                .attr("stop-color", color2)
                .attr("stop-opacity", 1);
    };
```

Then we will call it three time to create 3 gradient filter defination in svg.

```
createGradient(svg,'#fff','#CBFFDA','line111');
createGradient(svg,'#fff','#FFDCD4','line211');
createGradient(svg,'#fff','#C7D1FF','line311');
```

We will add a svg group to the parent svg element.We will add our area charts inside this group.

```
        var area_g = svg
        .append("g");
        var area = d3.svg.area()
        .interpolate("cardinal")
        .x(function (d) {
            return x(d.Year);
        })
        .y0(h)
        .y1(function (d) {
            return y(d.val);
        });
```

Next we will create the area chart inside the `mouseover` event of the `plotLine()` function. We will use the gradient id here to make sure the correct gradient is getting added here.

```
area_g.append('path')
        .datum(mean_data)
        .attr({
            class: className+'11',
            d: area,
            fill:'url(#'+className+'11)' // Gradient ID
        })
        .style({
            opacity:.5
        });
```

We shall remove the area chart all together on `mouseout`.

```
area_g.select('.'+className+'11').remove();
```

Add Tooltip :

We will add the tooltip using html & css.First define 3 divs for three types of tooltip.

```
        var div_line1 = d3.select("body").append("div")
            .attr("class", "line1_tt")
            .style("opacity", 0);

    var div_line2 = d3.select("body").append("div")
            .attr("class", "line2_tt")
            .style("opacity", 0);

    var div_line3 = d3.select("body").append("div")
            .attr("class", "line3_tt")
            .style("opacity", 0);
```

Next disply them on `mouseover` and hide them on `mouseout`. You can refer the stylesheet to find the css properties of the tooltip.

```
//on mouseover
var div=d3.select('.'+className+'_tt');

div.transition()
        .duration(200)
        .style("opacity", .9);
div.html("Year : "+d.Year + "
" +"Temp : " + d.val+"F")
        .style("left", (d3.event.pageX) + "px")
        .style("top", (d3.event.pageY - 50) + "px");


//on mouseout
var div=d3.select('.'+className+'_tt');
div.transition()
```

```
            .duration(200)
            .style("opacity", 0);
```

Add Legend

At last we will add the Legends for each line chart. It's also very straight forward to add the legend, however if needed you can refer my earlier tutorials where I have explaned how to create them...or simply ask me in the comment section.

```
            var legendRectSize=10;
        var legendSpacing=7;
        var legendHeight=legendRectSize+legendSpacing;

        var legend=svg.selectAll('.legend')
                .data(["Global Mean","Northern
Hemisphere","Southern Hemisphere"])
                .enter()
                .append('g')
                .attr({
                    class:'legend',
                    transform:function(d,i){
                        //Just a calculation for x & y
position
                        return 'translate('+((i*200)+20)+','
+ -30 + ')';
                    }
                });
        legend.append('rect')
                .attr({
                    width:legendRectSize,
                    height:legendRectSize,
                    rx:10,
```

```
            ry:10
        })
        .style({
            fill:color,
            stroke:color
        });

legend.append('text')
        .attr({
            x:15,
            y:9
        })
        .text(function(d){
            return d;
        }).style({
    fill:'#000000',
    'font-size':'10px'
```

Here is the final version of the code.

Code

Final demo.

Demo

Related

Create Pie Charts using D3.js
In "D3.js"

Create a Simple Pie Chart using D3.js
In "D3.js"

How to create reusable charts with React and D3 Part2
In "D3.js"

Filed Under: D3.js | Tagged With: area, chart, d3.js, Interactive, JavaScript, Learn, line, Programming, Visualization

Subscribe to stay in loop

* indicates required

Email Address *

Subscribe

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Copyright © 2024 A Developer Diary