Paper DV09-2014

Interactive HTML Reporting Using D3

Naushad Pasha Puliyambalath Ph.D., Nationwide Insurance, Columbus, OH

ABSTRACT

Data visualization tools using JavaScript have been flourishing recently. Of notable importance is D3 (Data-Driven Documents http://d3js.org/). In this paper we will see how to create dynamic HTML reports from SAS by making use of D3. More precisely, we will create a SAS program which upon execution will produce an HTML file with an interactive dashboard containing a bar chart, a pie chart, and a table. The paper assumes a basic knowledge of HTML and JavaScript.

INTRODUCTION

The needs of each report are unique and the use of any generic template highly limits the report's ability to convey information effectively. Thus there is a need for a technique where the programmer has complete freedom in designing the report down to the last pixel. This paper demonstrates such a method.

A good report has to be easy to navigate, aesthetically pleasing, and cater to the data summarization needs of all its users. Features in HTML5, CSS3, and JavaScript make all of these possible. Having the reports in HTML format also makes it easier to integrate them to the internet or intranet. The use of JavaScript allows the programmer to let the user decide how they want to see the report by having the raw data in the report and modifying it based on user interaction. There are countless JavaScript libraries that one can use to avoid writing everything from scratch. We will be using the D3 (http://d3js.org/) JavaScript library due to its generic nature and animation capabilities.

Developing a SAS program which produces a customized HTML report upon execution involves first designing the HTML/JavaScript template and then integrating it into a SAS program. The goal of the SAS program is to read the HTML template and output it to an HTML file along with summarized data.

It is not the intention of this paper to educate the reader on HTML and JavaScript. The goal of this paper is to demonstrate a technique where the programmer has the ability to design highly interactive and visually stunning reports from SAS. I will use a simple and completely working example to show the main elements involved in such a design.

SAMPLE DATA AND END REPORT

We will use the following fictitious table called Frequencies for our report. The table has a column called State and three segment columns called Low, Mid, and High. The values of Low, Mid and High can be thought of as frequencies for the respective segment and State.

```
data Frequencies;
    input State $1-2 Low Mid High;
datalines;
AL 4786 1319 249
AZ 1101 412 674
CT 932 2149 418
DE 832 1152 1862
FL 4481 3304 948
GA 1619 167 1063
IA 1819 247 1203
IL 4498 3852 942
IN 797 1849 1534
KS 162 379 471
;run;
```

The HTML report should look as in Figure 1 and consist of a bar chart, a pie chart and a table. By default the bar chart shows the sum of segments by state as the height of each bar, the pie chart shows the percentage of total in each segment, and the table shows the segment total and countrywide percentage along with color and name.

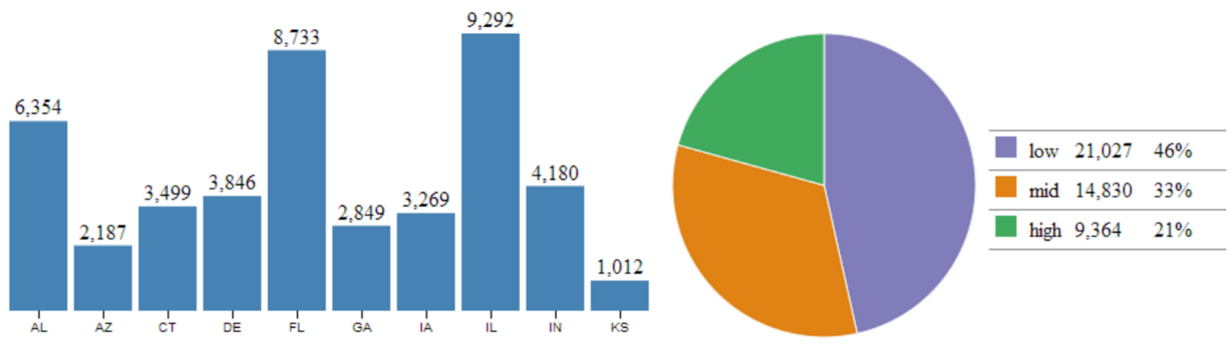

Figure 1. End Report

We also want to implement two interactions. The first interaction is that when the user hovers mouse over any of the state bars, the pie and table should change to show the frequency only for that state (Figure 2).

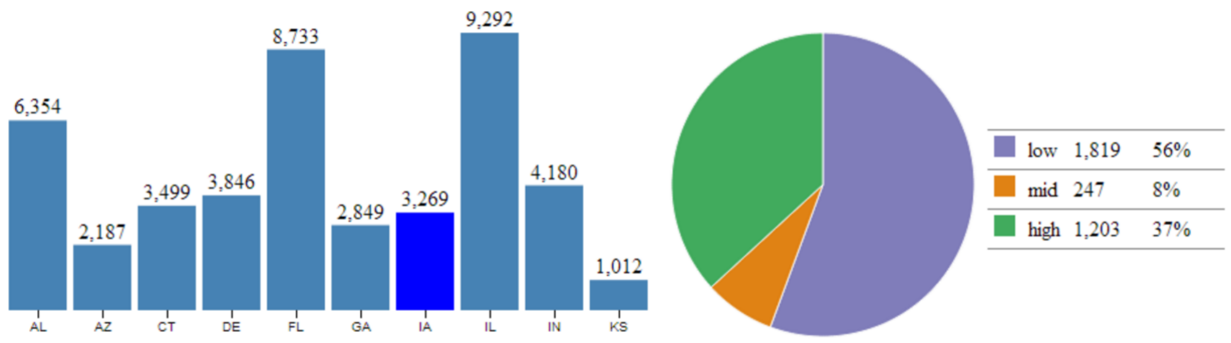

Figure 2. Hovering mouse over a bar modifies the pie chart and table.

The second interaction is that when the user hover mouse over a pie slice, the histogram changes to show the frequencies only for the selected segment by state and also the color of the bars change to the color of the selected segment (Figure 3).

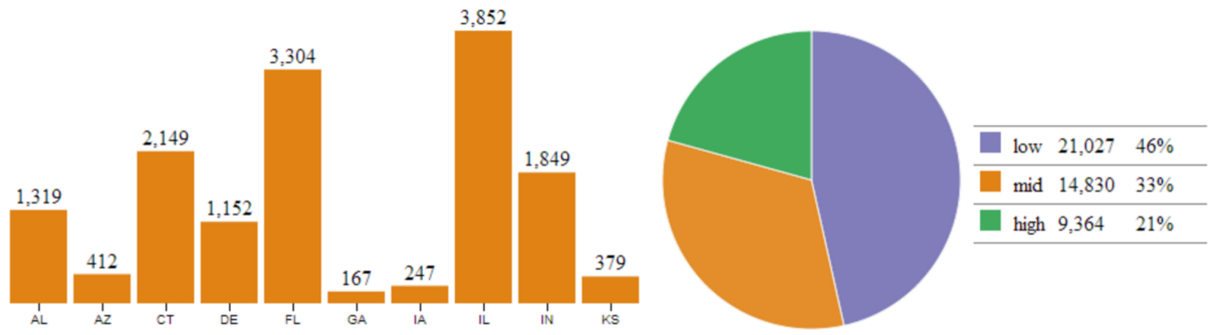

Figure 3. Hovering mouse over a pie slice modifies the histogram.

DESIGN OF HTML REPORT

The solution will consist of three programs. First program is a CSS (Cascading Style Sheet) we will save it as dashboard.css. The second program is a JavaScript, we will save this as dashboard.js. The third program is a SAS program which will read the dashboard.css and dashboard.js and combine them with data to create the final HTML output file.

DASHBOARD.CSS

```
body{ width:1060px;  margin:200px auto;  }
path {  stroke: #fff; }
```

```
path:hover {  opacity:0.9; }
rect:hover {  fill:blue; }
.axis {  font: 10px sans-serif; }
.legend tr{    border-bottom:1px solid grey; }
.legend tr:first-child{    border-top:1px solid grey; }
.axis path, .axis line { fill:none;  stroke:#000;  shape-rendering:crispEdges; }

.x.axis path {  display: none; }
.legend{
    margin-bottom:76px;
    display:inline-block;
    border-collapse: collapse;
    border-spacing: 0px;
}
.legend td{padding:4px 5px; vertical-align:bottom;}
.legendFreq, .legendPerc{  align:right;  width:50px; }
```

DASHBOARD.JS

```
function dashboard(id, fData){
    var hG, pC, leg, barColor = 'steelblue';
    function segColor(c){
           return {low:"#807dba", mid:"#e08214",high:"#41ab5d"}[c];
    }

    // compute total for each state.
    fData.forEach(function(d){d.total=d.freq.low+d.freq.mid+d.freq.high;});

    // function to handle histogram.
    function histoGram(fD){
        var hG={},    hGDim = {t: 60, r: 0, b: 30, l: 0};
        hGDim.w = 500 - hGDim.l - hGDim.r,
        hGDim.h = 300 - hGDim.t - hGDim.b;

        //create svg for histogram.
        var hGsvg = d3.select(id).append("svg")
            .attr("width", hGDim.w + hGDim.l + hGDim.r)
            .attr("height", hGDim.h + hGDim.t + hGDim.b).append("g")
            .attr("transform", "translate(" + hGDim.l + "," + hGDim.t + ")");

        // create function for x-axis mapping.
        var x = d3.scale.ordinal().rangeRoundBands([0, hGDim.w], 0.1)
                .domain(fD.map(function(d) { return d[0]; }));

        // Add x-axis to the histogram svg.
        hGsvg.append("g").attr("class", "x axis")
            .attr("transform", "translate(0," + hGDim.h + ")")
            .call(d3.svg.axis().scale(x).orient("bottom"));

        // Create function for y-axis map.
        var y = d3.scale.linear().range([hGDim.h, 0])
                .domain([0, d3.max(fD, function(d) { return d[1]; })]);

        // Create bars for histogram to contain rectangles and freq labels.
        var bars = hGsvg.selectAll(".bar").data(fD).enter()
                .append("g").attr("class", "bar");

        //create the rectangles.
        bars.append("rect")
            .attr("x", function(d) { return x(d[0]); })
            .attr("y", function(d) { return y(d[1]); })
            .attr("width", x.rangeBand())
            .attr("height", function(d) { return hGDim.h - y(d[1]); })
            .attr('fill',barColor)
            .on("mouseover",mouseover)// mouseover is defined below.
            .on("mouseout",mouseout);// mouseout is defined below.
        //Create the frequency labels above the rectangles.
        bars.append("text").text(function(d){ return d3.format(",")(d[1])})
            .attr("x", function(d) { return x(d[0])+x.rangeBand()/2; })
            .attr("y", function(d) { return y(d[1])-5; })
            .attr("text-anchor", "middle");

        function mouseover(d){  // utility function to be called on mouseover.
            // filter for selected state.
            var st = fData.filter(function(s){ return s.State == d[0];})[0],
                nD = d3.keys(st.freq).map(function(s){
                        return {type:s, freq:st.freq[s]};});

            // call update functions of pie-chart and legend.
            pC.update(nD);
            leg.update(nD);
        }

        function mouseout(d){    // utility function to be called on mouseout.
            // reset the pie-chart and legend.
            pC.update(tF);
            leg.update(tF);
        }

        // create function to update the bars. This will be used by pie-chart.
        hG.update = function(nD, color){
            // update the domain of the y-axis map to reflect change.
            y.domain([0, d3.max(nD, function(d) { return d[1]; })]);

            // Attach the new data to the bars.
            var bars = hGsvg.selectAll(".bar").data(nD);

            // transition the height and color of rectangles.
            bars.select("rect").transition().duration(500)
                .attr("y", function(d) {return y(d[1]); })
                .attr("height", function(d) { return hGDim.h - y(d[1]); })
                .attr("fill", color);

            // transition the frequency labels location and change value.
            bars.select("text").transition().duration(500)
                .text(function(d){ return d3.format(",")(d[1])})
                .attr("y", function(d) {return y(d[1])-5; });
        }
        return hG;
    }

    // function to handle pieChart.
    function pieChart(pD){
        var pC ={},    pieDim ={w:250, h: 250};
        pieDim.r = Math.min(pieDim.w, pieDim.h) / 2;

        // create svg for pie chart.
        var piesvg = d3.select(id).append("svg")
            .attr("width", pieDim.w).attr("height", pieDim.h).append("g")
            .attr("transform", "translate("+pieDim.w/2+","+pieDim.h/2+")");

        // create function to draw the arcs of the pie slices.
        var arc = d3.svg.arc().outerRadius(pieDim.r - 10).innerRadius(0);

        // create a function to compute the pie slice angles.
        var pie = d3.layout.pie().sort(null).value(function(d) { return d.freq; });

        // Draw the pie slices.
        piesvg.selectAll("path").data(pie(pD)).enter()
            .append("path").attr("d", arc)
            .each(function(d) { this._current = d; })
            .style("fill", function(d) { return segColor(d.data.type); })
```

```
            .on("mouseover",mouseover).on("mouseout",mouseout);

        // create function to update pie-chart. This will be used by histogram.
        pC.update = function(nD){
            piesvg.selectAll("path").data(pie(nD)).transition().duration(500)
                .attrTween("d", arcTween);
        }
        // Utility function to be called on mouseover a pie slice.
        function mouseover(d){
            // call the update function of histogram with new data.
            hG.update(fData.map(function(v){
                return [v.State,v.freq[d.data.type]];}),segColor(d.data.type));
        }
        //Utility function to be called on mouseout a pie slice.
        function mouseout(d){
            // call the update function of histogram with all data.
            hG.update(fData.map(function(v){
                return [v.State,v.total];}), barColor);
        }
        // Animating the pie-slice requires a custom function which specifies
        // how the intermediate paths should be drawn.
        function arcTween(a) {
            var i = d3.interpolate(this._current, a);
            this._current = i(0);
            return function(t) { return arc(i(t));    };
        }
        return pC;
    }

    // function to handle legend.
    function legend(lD){
        var leg = {};

        // create table for legend.
        var legend = d3.select(id).append("table").attr('class','legend');

        // create one row per segment.
        var tr = legend.append("tbody").selectAll("tr").data(lD)
            .enter().append("tr");

        // create the first column for each segment.
        tr.append("td").append("svg").attr("width", '16')
            .attr("height", '16').append("rect")
            .attr("width", '16').attr("height", '16')
            .attr("fill",function(d){ return segColor(d.type); });

        // create the second column for each segment.
        tr.append("td").text(function(d){ return d.type;});

        // create the third column for each segment.
        tr.append("td").attr("class",'legendFreq')
            .text(function(d){ return d3.format(",")(d.freq);});

        // create the fourth column for each segment.
        tr.append("td").attr("class",'legendPerc')
            .text(function(d){ return getLegend(d,lD);});

        // Utility function to be used to update the legend.
        leg.update = function(nD){
            // update the data attached to the row elements.
            var l = legend.select("tbody").selectAll("tr").data(nD);

            // update the frequencies.
            l.select(".legendFreq")
                .text(function(d){ return d3.format(",")(d.freq);});

            // update the percentage column.
            l.select(".legendPerc").text(function(d){ return getLegend(d,nD);});
        }

        function getLegend(d,aD){ // Utility function to compute percentage.
            return d3.format("%")(d.freq/d3.sum(aD.map(function(v){
                        return v.freq; })));}

        return leg;
    }

    // calculate total frequency by segment for all state.
    var tF = ['low','mid','high'].map(function(d){
        return {type:d, freq:d3.sum(fData.map(function(t){ return t.freq[d];}))};
    });

    // calculate total frequency by state for all segment.
    var sF = fData.map(function(d){return [d.State,d.total];});

    var hG = histoGram(sF), // create the histogram.
        pC = pieChart(tF),  // create the pie-chart.
        leg= legend(tF);    // create the legend.
}
```

The dashboard.js program has a single function called `dashboard`. This function has three sub functions called `histoGram`, `pieChart`, and `legend`. These sub functions handle all the initiation and interaction needs of the respective component of the report. To create the report we will call the function `dashboard` by passing an id of a div element, and data to it as parameters. This design makes the dashboard.js a reusable program.

Including the D3 library in the HTML creates an object called `d3` which is a library of functions. Various HTML DOM elements can be selected using the `select` and `selectAll` functions of `d3`. For example in the following line of program, we are first selecting all DOM elements with class `bar`, then attaching the elements of the array `fD` to each element in order. The enter function is used to catch any elements that are new and for these new elements we create a `g` element using the `append` function and set its class to `bar` using the `attr` function.

```
d3.selectAll(".bar").data(fD).enter().append("g").attr("class", "bar");
```

The D3 library creates HTML elements by attaching data to them. The ingenuity of this design is that when D3 need to transition the element to reflect a change in data, it will use the already attached data and the new data to create the animation and then replace the old data with the new data. For simple attributes like color or length, D3 knows how to transition, but for complex attributes like path we have to supply our own transition function. In the above program we are doing this for animating the pie slices.

```
pC.update = function(nD){
            piesvg.selectAll("path").data(pie(nD)).transition().duration(500)
                .attrTween("d", arcTween);
        }
function arcTween(a) {
            var i = d3.interpolate(this._current, a);
            this._current = i(0);
            return function(t) { return arc(i(t));    };
        }
```

DASHBOARD.SAS

```
Data _Null_;
    Length row $2000;      /* Big enough to hold every line from css and js. */
    file "DashBoard.html"; /* Output file. */

    Put "<!DOCTYPE html>";
    Put "<meta charset='utf-8'>";

    /* create css part of html. Edit infile to use the correct location. */
    Put "<style>";
    Do While(not eofCss);
        infile "dashboard.css" dsd dlm='0D'x end=eofCss;
        input row ;
```

```
            Put row;
        End;
        Put "</style>";

        Put "<body>";
        Put "<div id='dashboard'></div>";
        Put "<script src='http://d3js.org/d3.v3.min.js'></script>";

        /* create the dashboard.js part of html. */
        Put "<script>";
        Do While(not eofJS);
            infile "dashboard.js" dsd dlm='0D'x end=eofJS;
            input row ;
            Put row;
        End;
        Put "</script>";

        /* Create data and call the dashboard function. */
        Put "<script>";
        Put "var freqData=[";
        Do While(not eofData);
            Set Frequencies end=eofData;
            row = CatS("{State:'",State,
                 "',freq:{low:",low,",mid:",mid,",high:",high,"}}");
            if eofData Then row = catS(row,"];");
            else row=catS(row,",");
            put row;
        End;
        Put "dashboard('#dashboard',freqData);";
        Put "</script>";
        Put "</body>";
        Put "</html>";
        Stop;
    Run;
```

The dashboard.sas program reads in the dashboard.css and dashboard.js files and combines them with the data from the table Frequencies to create an HTML file called DashBoard.html. The length of the variable `row` is set to be high enough value to include any line of data or program. The D3 library is included from the address http://d3js.org/d3.v3.min.js. If the dashboard.html needs to be viewed without internet access, then the D3 library can be downloaded and included in the DashBoard.html itself similar to how dashboard.js was included.

CONCLUSION

The above example is meant as a demonstration of the technique and can be improved immensely to create complex reports. One limitation of this technique is the size of the data that can be included in the HTML and the internet browser being used to view the report. The size of the data that can be included in the HTML is limited by the amount of memory available on the computer. By summarizing the data in SAS before outputting to HTML, the size limitation can be handled to a certain extent. The D3 library uses features in HTML5 and hence would require modern browsers to view. This would be every browser except Internet Explorer 8 and below. More examples of graphs using D3 can be found at https://github.com/mbostock/d3/wiki/Gallery. Many of these examples are reusable and can be easily incorporated in to a SAS program.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Naushad Pasha Puliyambalath Ph. D.
Enterprise: Nationwide Insurance
Address: 3 Nationwide Plaza
City, State ZIP: Columbus, OH 43215
Work Phone: 614-677-3247
E-mail: puliyan1@nationwide.com