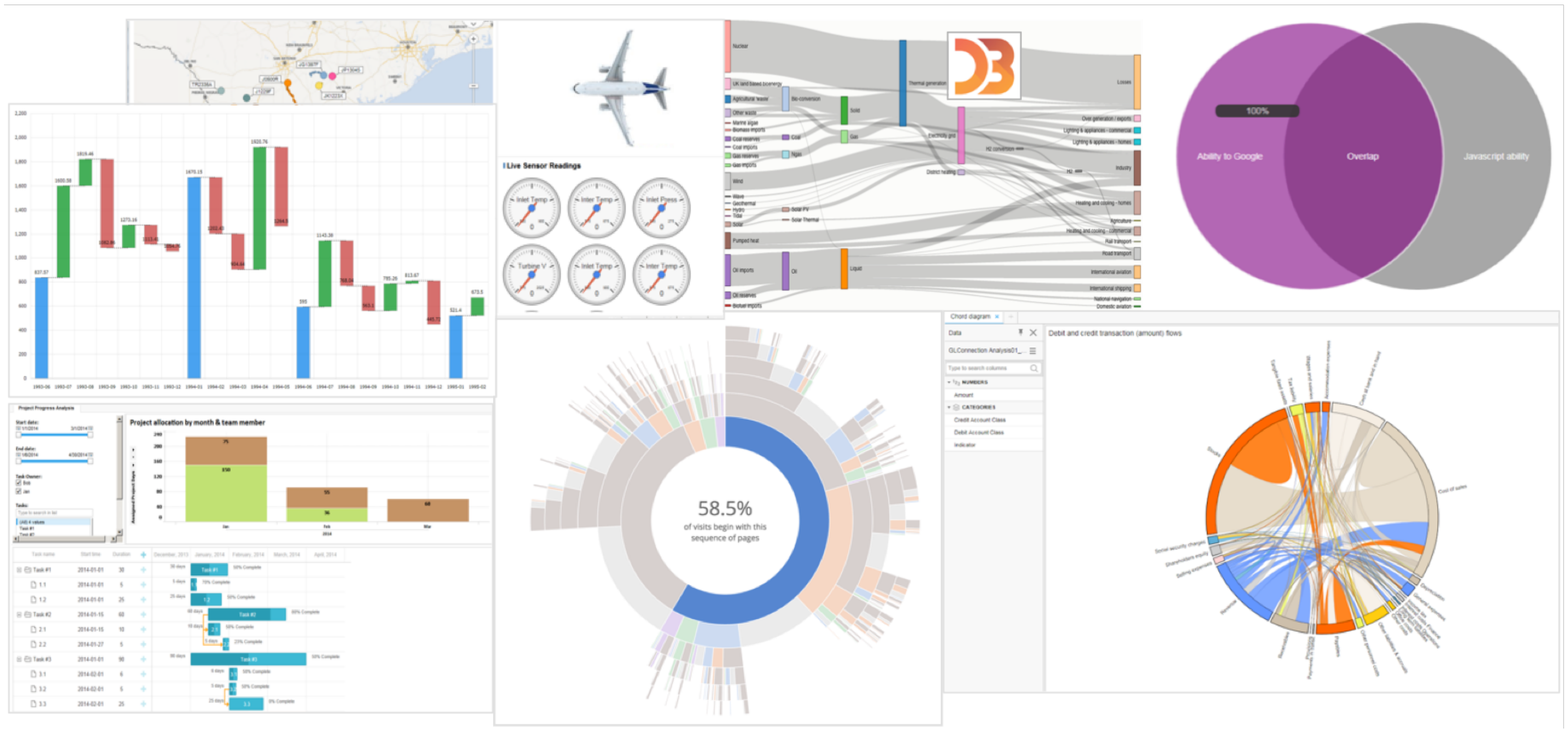
JavaScript Visualization Framework - JSViz

Tutorial #2 – Re-Creating the DoughnutChart Visualization



February 2017

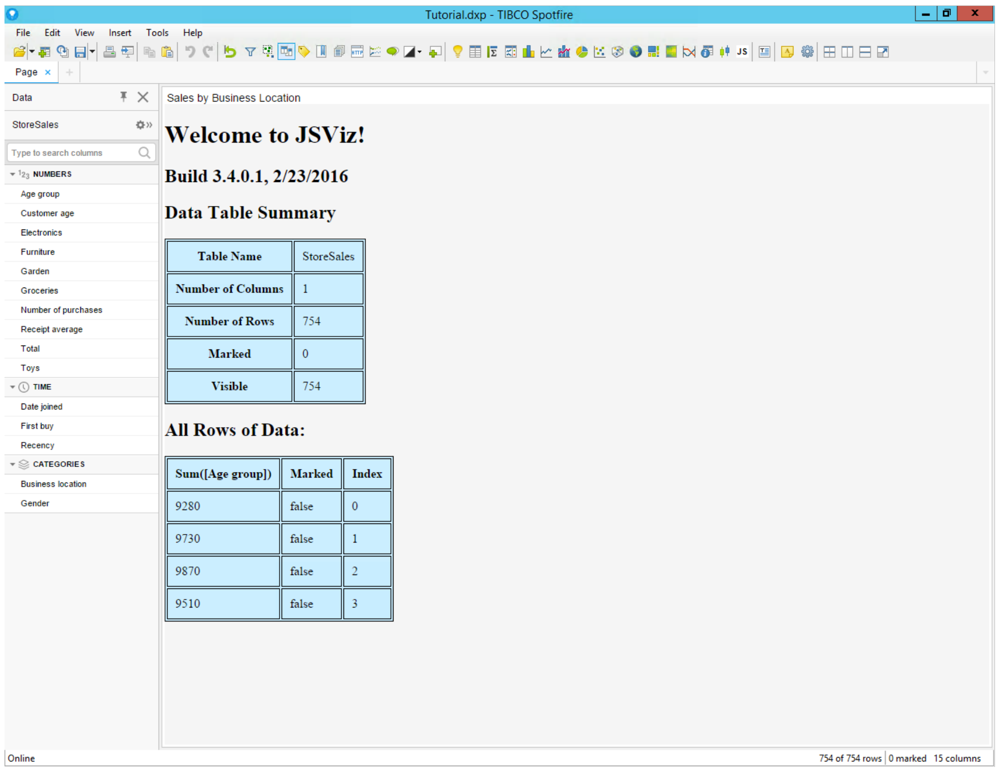
1. Introduction

This tutorial builds on the first step in which we created a default visualization using the template files supplied with JSViz.

This tutorial will walk through the following steps:

- Configuring JSViz to provide the JSON data we need for our Doughnut Chart
- Creating a Tester file to debug our chart code in a web browser
- Creating the actual JavaScript code
- Updating JSViz and seeing our chart

This is where we left off on the last tutorial:



Welcome to JSViz!

Build 3.4.0.1, 2/23/2016

Data Table Summary

Table Name	StoreSales
Number of Columns	1
Number of Rows	754
Marked	0
Visible	754

All Rows of Data:

Sum([Age group])	Marked	Index
9280	false	0
9730	false	1
9870	false	2
9510	false	3

2. Configuring the JSON Data

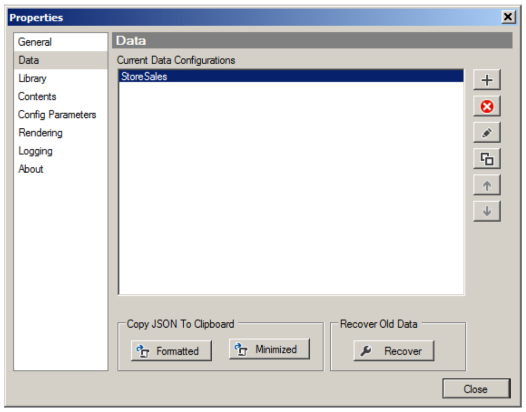
By default, JSViz picked the first numerical column “`Age Group`” and grouped it by the first categorical column “`Business Location`”. For our Doughnut Chart what we actually want to display is Total Sales grouped by Business Location.

We also want the following behavior:

- Our chart will only show marked items
- When no items are marked our chart will show all items

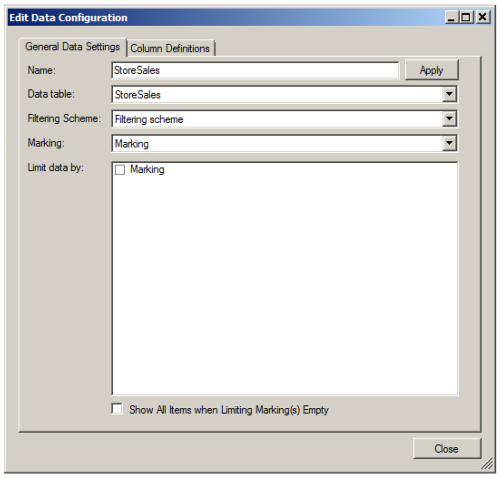
Start by opening the JSViz Configuration by clicking on the configuration cog.

Switch to the “Data” Property Page.



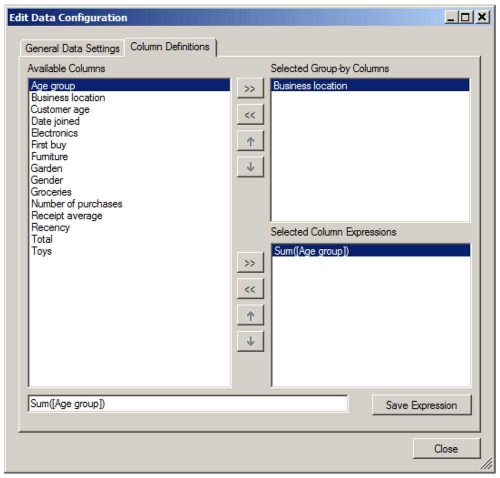
Current Data Configurations: StoreSales

Edit the “`StoreSales`” data configuration by either selecting it and clicking the edit icon or by double-clicking it.



Name: StoreSales
Data table: StoreSales
Filtering Scheme: Filtering scheme
Marking: Marking
Limit data by: ☐ Marking
☐ Show All Items when Limiting Marking(s) Empty

Switch to the "Column Definitions" tab.



Available Columns: Age group, Business location, Customer age, Date joined, Electronics, First buy, Furniture, Garden, Gender, Groceries, Number of purchases, Receipt average, Recency, Total, Toys

Selected Group-by Columns: Business location

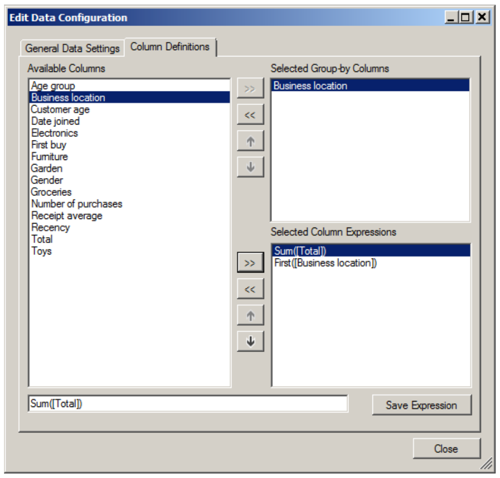
Selected Column Expressions: Sum([Age group])

Sum([Age group])

Add the following columns into the Column Expressions list:

- `Total`
- `Business Location`

Remove the existing "`Sum([Age Group])`" Column Expression.



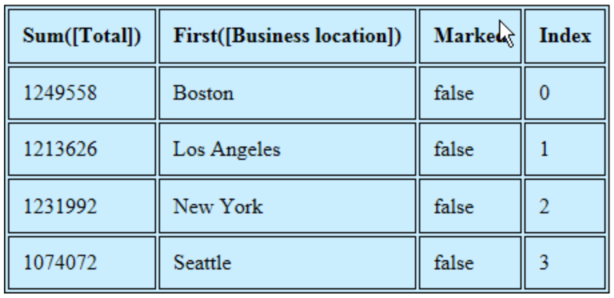
Available Columns: Age group, Business location, Customer age, Date joined, Electronics, First buy, Furniture, Garden, Gender, Groceries, Number of purchases, Receipt average, Recency, Total, Toys

Selected Group-by Columns: Business location

Selected Column Expressions: Sum([Total]), First([Business location])

Sum([Total])

If we leave things as they are, the column names passed in the JSON will literally be "`Sum([Total])`" and "`First([Business location])`".



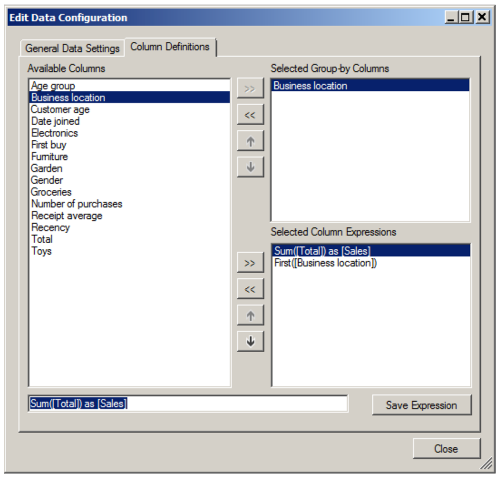
Sum([Total])	First([Business location])	Marked	Index
1249558	Boston	false	0
1213626	Los Angeles	false	1
1231992	New York	false	2
1074072	Seattle	false	3

This is not very user friendly so we would prefer the columns to be simply "`Sales`" and "`Location`". To do this we need to rename the columns using the built-in Spotfire expression "`as`".

Select the “`Sum([Total])`” Column Expression and edit the Expression to read:

```
Sum([Total]) as [Sales]
```

Click on “Save Expression”.



Available Columns: Age group, Business location, Customer age, Date joined, Electronics, First buy, Furniture, Garden, Gender, Groceries, Number of purchases, Receipt average, Recency, Total, Toys

Selected Group-by Columns: Business location

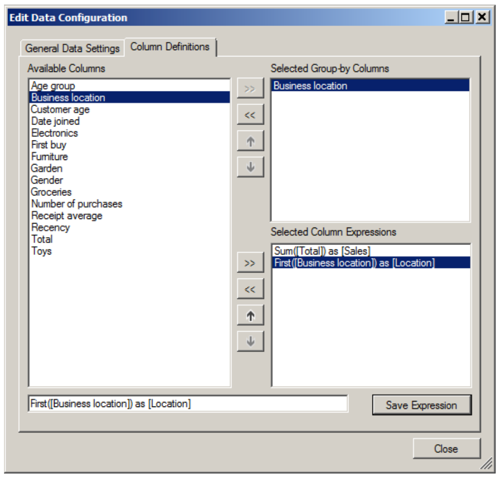
Selected Column Expressions: Sum([Total]) as [Sales], First([Business location])

Sum([Total]) as [Sales]

Select the “`First([Business Location])`” Column Expression and edit the Expression to read:

```
First([Business Location]) as [Location]
```

Click on “Save Expression”.



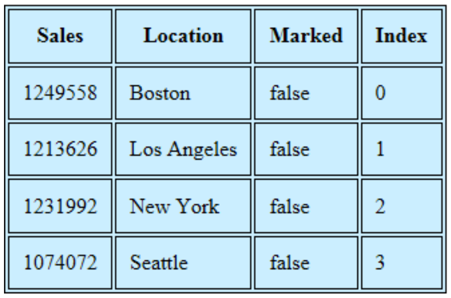
Available Columns: Age group, Business location, Customer age, Date joined, Electronics, First buy, Furniture, Garden, Gender, Groceries, Number of purchases, Receipt average, Recency, Total, Toys

Selected Group-by Columns: Business location

Selected Column Expressions: Sum([Total]) as [Sales], First([Business location]) as [Location]

First([Business location]) as [Location]

Now if we look at our JSViz visualization we can see that the column names have been updated.

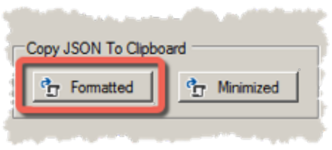


Sales	Location	Marked	Index
1249558	Boston	false	0
1213626	Los Angeles	false	1
1231992	New York	false	2
1074072	Seattle	false	3

We can also take a look at the JSON that JSViz will pass to our visualization.

Close the “Edit Data Configuration” dialog.

Click on the “Formatted” button to copy formatted JSON data to the Clipboard.



If you view the clipboard contents in your favorite text editor, it should look like this:

```
{
  "columns": [
    "Sales",
    "Location"
  ],
  "baseTableHints": {
    "rows": 754,
    "visible": 754,
    "marked": 0,
    "tableName": "StoreSales",
    "settingName": "StoreSales"
  },
  "data": [
    {
      "items": [
        1249558,
        "Boston"
      ],
      "hints": {
        "index": 0
      }
    },
    {
      "items": [
        1213626,
        "Los Angeles"
      ],
      "hints": {
        "index": 1
      }
    },
    {
      "items": [
        1231992,
        "New York"
      ],
      "hints": {
        "index": 2
      }
    },
    {
      "items": [
        1074072,
        "Seattle"
      ],
      "hints": {
        "index": 3
      }
    }
  ],
  "additionalTables": [],
  "user": "spotfireadmin",
  "static": false,
  "wait": 0,
  "runtime": {},
  "legend": true,
  "config": {},
  "style": "font-family:\"Arial\",sans-serif;font-size:11px;font-style:Normal;font-
weight:Normal;color:#000000;background-color:transparent;border-style:None;border-top-color:#FFFFFF;border-
right-color:#FFFFFF;border-bottom-color:#FFFFFF;border-left-color:#FFFFFF;border-top-width:0px;border-right-
width:0px;border-bottom-width:0px;border-left-width:0px;border-top-left-radius:0px;border-top-right-
radius:0px;border-bottom-right-radius:0px;border-bottom-left-radius:0px;padding-top:0px;padding-
bottom:0px;padding-left:0px;padding-right:0px;margin-top:0px;margin-bottom:0px;margin-left:0px;margin-
right:0px;"
}
```

3. Setting up a Tester File

In this tutorial we are going to provide the final JavaScript code for the Doughnut Chart so it might not make sense to walk through the process of setting up a testing environment. However with your own chart code, you will no doubt need to set breakpoints, step through your code and see the values of variables as your development progresses.

Navigate to the folder on your Web Server where you installed the JSViz sample files.

Within the "`testing`" folder, make a copy of the "`Tester.html`" file and call it "`Tester-MyDoughnutChart.html`". Open the file using your favorite text editor.

The first step is to setup the JavaScript inclusions required for our visualization. These are:

```
http://localhost/jsviz/lib/jQuery/jQuery.js
http://localhost/jsviz/lib/d3/d3.v3.min.js
http://localhost/jsviz/lib/JSViz/Introspection.js
```

Note: `Introspection.js` is a file that is only used in the Tester to display the visualization data in a more user-friendly form. It is not required in the JSViz visualization.

The next set of includes are for our Doughnut Chart JavaScript and CSS files:

```
http://localhost/jsviz/src/css/MyDoughnutChart.css
http://localhost/jsviz/src/js/MyDoughnutChart.js
```

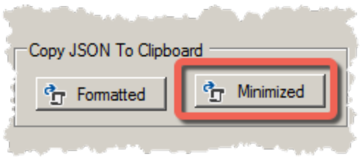
The last item to include is the JSViz JavaScript library file:

```
http://localhost/jsviz/lib/JSViz/JSViz.js
```

Next we want to copy our JSViz JSON data from our DXP file into the "`sfdata`" variable.

Within Spotfire, open the JSViz configuration and select the "data" property page.

Click on the "Minimized" button to copy unformatted JSON data to the Clipboard.



Locate the line containing the "`sfdata`" variable and paste in the clipboard text. Note that JavaScript does not require a final semicolon at the end of the line when doing this:

```
function call_renderCore()
{
    /*
    Paste in the JSON data copied to the clipboard from within your JS Visualization instance
```

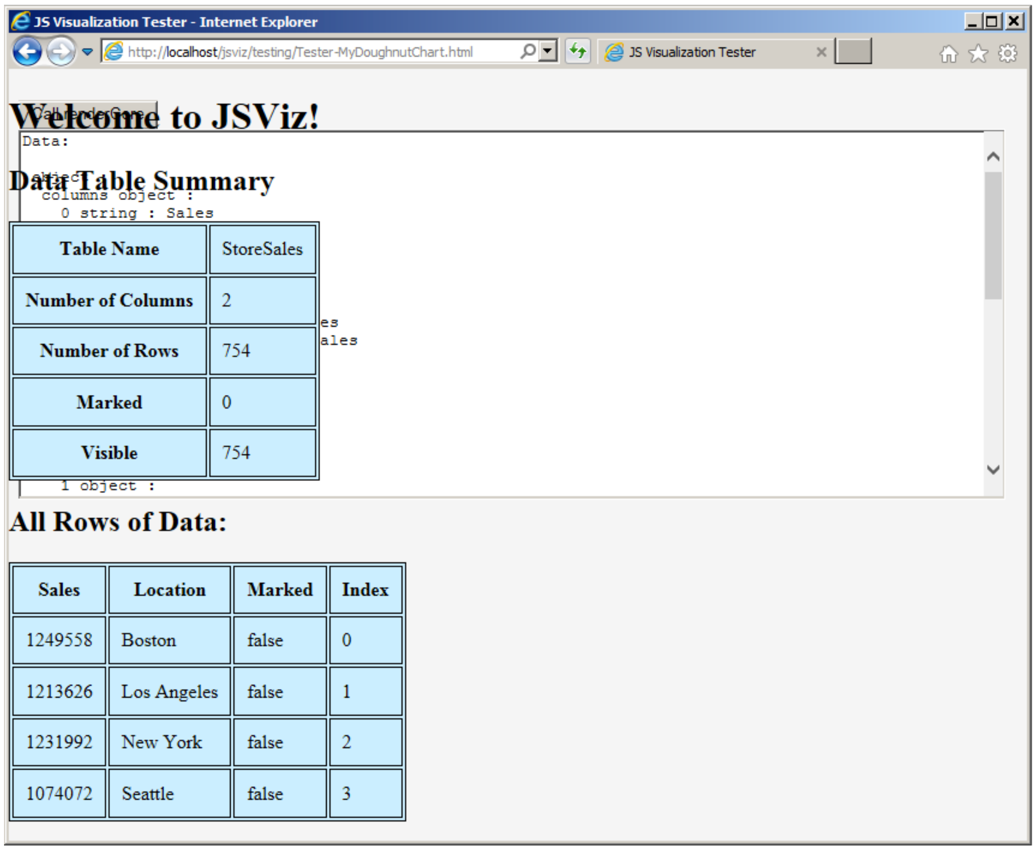
```
        */

        var sfdata = {"columns":["Sales","Location"],"baseTableHints":{ "rows":754, "visible":754 ...

        renderCore ( sfdata );
    }
```

At this point our Tester file is complete, so make sure to save your changes.

To see the Tester file in action, open your browser and navigate to the tester page. In our case it is:

```
http://localhost/jsviz/testing/Tester-MyDoughnutChart.html
```

Click on the "`Call RenderCore()`" button and you will see the same Template Visualization that we saw in Spotfire earlier.


JS Visualization Tester - Internet Explorer
http://localhost/jsviz/testing/Tester-MyDoughnutChart.html
JS Visualization Tester
Call RenderCore

Welcome to JSViz!

```
Data:
  object
    columns object :
      0 string : Sales
...es
...ales
      1 object :
```

Data Table Summary

Table Name	StoreSales
Number of Columns	2
Number of Rows	754
Marked	0
Visible	754

All Rows of Data:

Sales	Location	Marked	Index
1249558	Boston	false	0
1213626	Los Angeles	false	1
1231992	New York	false	2
1074072	Seattle	false	3

Note: Some of the information is missing in the visualization such as Build Number and Build Date. This information is not passed in the JSON but is inserted as JavaScript objects by JSViz and the Tester file does not mimic this behavior.

4. Editing the JavaScript and CSS Files

Note: This tutorial is not meant to be a how-to guide for programming with d3 so we will just walk through the process of updating the code with minimal explanation.

First, open the "`MyDoughnutChart.css`" file in your favorite editor. Replace the entire contents with the contents of the "`MyDoughnutChart.css`" file from the tutorial folder.

Next, open the "`MyDoughnutChart.js`" file in your favorite editor. Locate the `renderCore()` method and insert the contents of "`MyDoughnutChart1.txt`" from the tutorial folder just before the `renderCore()` method:

```
var color = null;
var pie = null;
var svg = null;
var path = null;

function renderCore(data) {

...
```

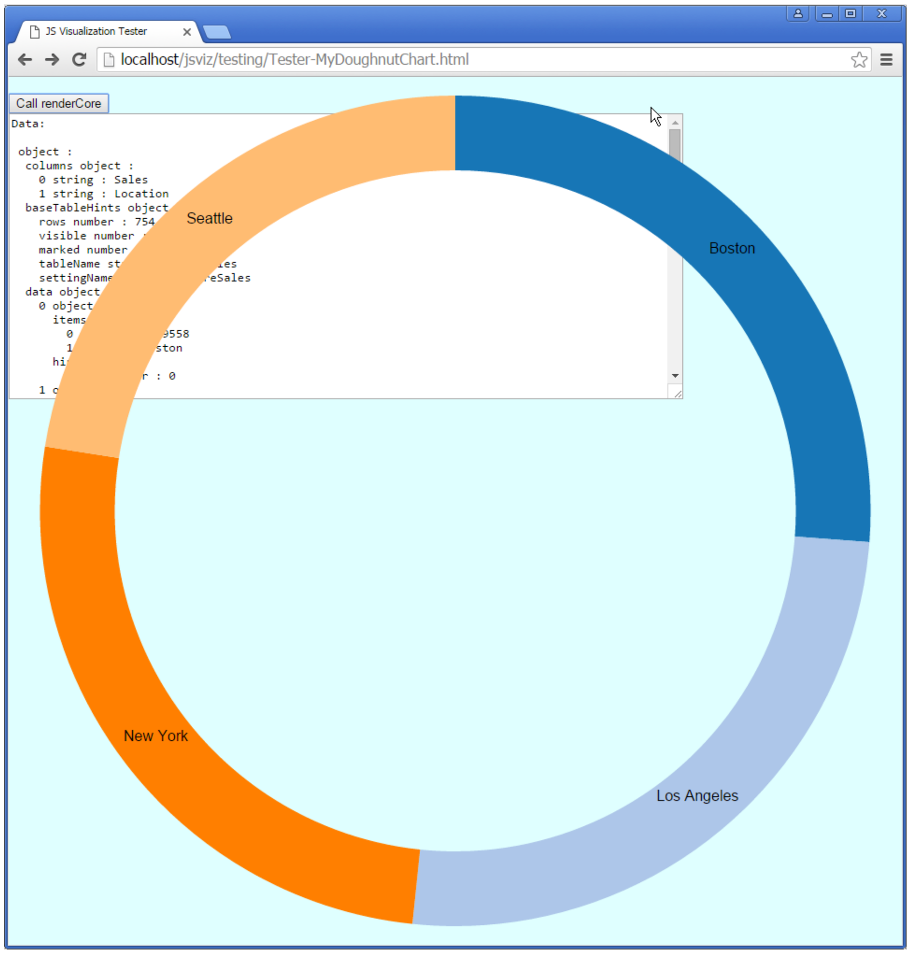
Next, insert the contents of "`MyDoughnutChart2.txt`" from the tutorial folder into the `renderCore()` method in place of the call to the "`displayWelcomeMessage`" function:

```
    function renderCore(data) {

    ...

        //
        // Replace the following code with actual Visualization code
        //

        // Check if color scheme is set up yet
        if (!color) {
            color = d3.scale.category20();
        }

    ...

        wait(sfdata.wait, sfdata.static);
    } // renderCore
```

Lastly, insert the contents of "`MyDoughnutChart3.txt`" from the tutorial folder immediately after the `renderCore()` method:

```
    } // renderCore

    function archLengthAccessor(dataRow) {
        return dataRow.items[0]; // Use first cell of a row as the arch size
    }

    // Store the displayed angles in _current.
    // Then, interpolate from _current to the new angles.
    // During the transition, _current is updated in-place by d3.interpolate.
    function arcTween(a) {
        var i = d3.interpolate(this._current, a);
        this._current = i(0);
        return function (t) {
            return arc(i(t));
        };
    }
```

This completes the code changes. Now, if we go back to our Tester file, refresh the browser window and press the "`Call renderCore`" button we will see our Doughnut Chart:



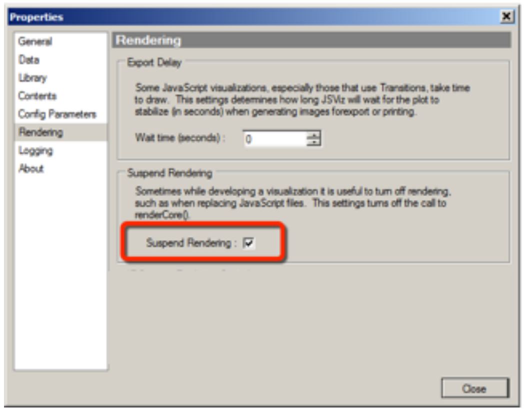
```
Data:

 object :
  columns object :
    0 string : Sales
    1 string : Location
  baseTableHints object
    rows number : 754
    visible number
    marked number
    tableName st          ies
    settingName           eSales
  data object
    0 object
      items
        0           9558
        1           ston
      hi
               r : 0
    1 o
```

5. Update JSViz

The JavaScript code for our Doughnut Chart uses the d3 library so we will need to add this to the JSViz library. Until we do that, the code will throw JavaScript errors because required objects are missing.

To prevent errors, we will disable the call to the `renderCore()` method while we add the missing library and then enable the calls once we are done.

Open the JSViz Configuration by clicking on the configuration cog.



Rendering

Export Delay

Some JavaScript visualizations, especially those that use Transitions, take time to draw. This settings determines how long JSViz will wait for the plot to stabilize (in seconds) when generating images forexport or printing.

Wait time (seconds) : 0

Suspend Rendering

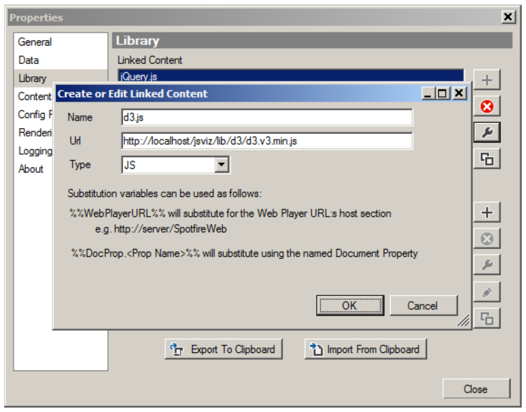
Sometimes while developing a visualization it is useful to turn off rendering, such as when replacing JavaScript files. This settings turns off the call to renderCore().

Suspend Rendering : ✓

Switch to the "Rendering" Property Page.

Check the "Suspend Rendering" option.

Answer "No" to any error messages that might appear.



Library

Linked Content

jQuery.js

Create or Edit Linked Content

Name: d3.js

Url: http://localhost/jsviz/lib/d3/d3.v3.min.js

Type: JS

Substitution variables can be used as follows:

%%WebPlayerURL%% will substitute for the Web Player URL:s host section
e.g. http://server/SpotfireWeb

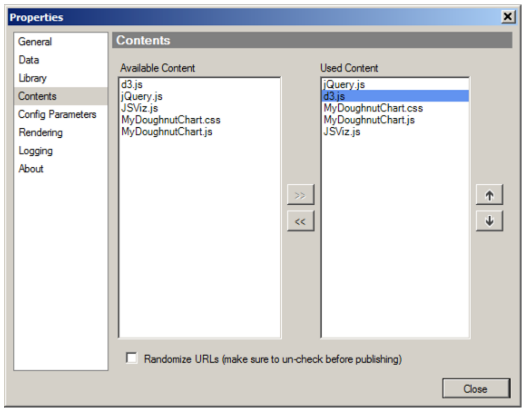
%%DocProp.<Prop Name>%% will substitute using the named Document Property

Switch to the "Library" Property Page.

Add a "Linked Content" item by clicking on the "+" button next to the Linked Content list. Enter the details as follows:

Name: `d3.js`
URL: `http://localhost/jsviz/lib/d3/d3.v3.min.js`
Type: `JS`

Click OK when done



Contents

Available Content:
d3.js
jQuery.js
JSViz.js
MyDoughnutChart.css
MyDoughnutChart.js

Used Content:
jQuery.js
d3.js
MyDoughnutChart.css
MyDoughnutChart.js
JSViz.js

☐ Randomize URLs (make sure to un-check before publishing)

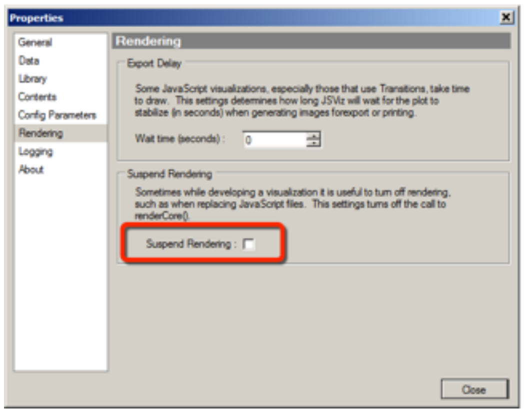
Switch to the "Contents" Property Page.

Select the `d3.js` item from the "Available Content" list and click the ">>" button to add it to the "Used Content" list.

Use the up arrow to move d3.js to just before the Doughnut Chart files.

Switch to the "Rendering" Property Page.

Uncheck the "Suspend Rendering" option.



Rendering

Export Delay

Some JavaScript visualizations, especially those that use Transitions, take time to draw. This settings determines how long JSViz will wait for the plot to stabilize (in seconds) when generating images forexport or printing.

Wait time (seconds) : 0

Suspend Rendering

Sometimes while developing a visualization it is useful to turn off rendering, such as when replacing JavaScript files. This settings turns off the call to renderCore().

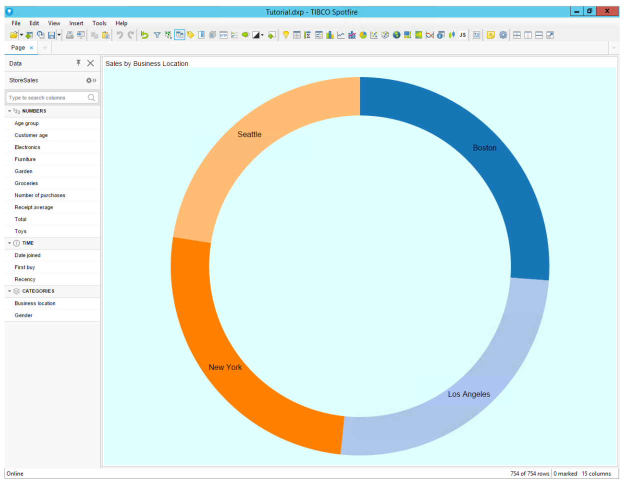
Suspend Rendering : ☐

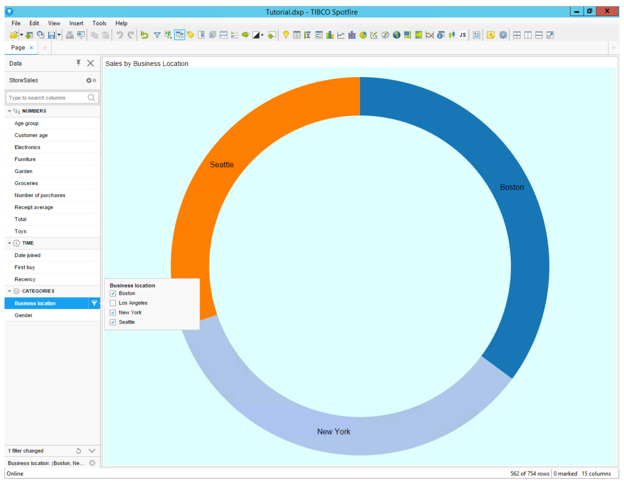
This completes our edits, so now is a good time to go ahead and save your work.

***Note**: If the Doughnut Chart does not appear right away, click somewhere within the JSViz visualization and then use the key combination Ctrl-Alt-Shift-F5 to refresh the JSViz visualization.*

6. Test Your Doughnut Chart

At this point you should see your Doughnut Chart and be able to change Filter settings and see the chart update:


Tutorial.dxp - TIBCO Spotfire
Sales by Business Location
Seattle
Boston
New York
Los Angeles
754 of 754 rows | 0 marked 15 columns


Tutorial.dxp - TIBCO Spotfire
Sales by Business Location
Business location
Boston
Los Angeles
New York
Seattle
1 filter changed
Business location: (Boston, Ne...
562 of 754 rows | 0 marked 15 columns