How to Create New Content

Intro

In this guide we will take a look at how I take an interesting looking visualization and convert it so that it can be used inside of a SAS Visual Analytics Data Driven Content object (I will shorten this to *DDC* from now on).

If you haven't worked with a DDC please read this great intro blog post.

Also check out the SAS Documentation on working with DDC objects and Programming Considerations if you have feel like I skipped something in this guide.

With these things out of the way here is what we are going to do: We have found a d3.js chart that we would like to use inside of SAS Visual Analytics.

Adapt, Integrate, Distribute

The chart example we are looking for can be found in this Stack Overflow question.

Ok so lets start by looking at the accept answer. We see three code snippets here:

1. Some JavaScript also containing some example data
2. Some CSS to make things look nice
3. A bit of HTML to display our graph

What we will know do is what I call the 3 Step DDC Conversion process:

1. Make it run in a DDC without any input from SAS Visual Analytics
2. Make it run with the sample data handed over by SAS Visual Analytics
3. Make it run for any data source (within the bounds of the graphics of course)

I'm providing the step by step material for this example in the folder d3-js-bar-chart-with-information - the main file in the Steps will always be called *index.html* but each steps result will be included in the folder as *index-step-**X**.html*.

In this example we will keep everything, except for external scripts, just within one file (*index.html*). Of course as you progress to bigger and bigger applications it is recommended that you separate concerns or start looking into using a framework.

Step 1: Making it Run

Create an *index.html* file. Now lets first create an HTML baseline:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

The title won't really matter but it is always good to give things proper names, e.g. *Bar Chart*. Next after the title tag add an opening and closing *style* tag and add the CSS from the Answer to it:

```
    <style>
      #graph {
        width: 700px;
        height: 500px;
      }

      text {
        font-size: 12px;
        font-family: 'Ubuntu';
      }

      .toolTip {
        position: absolute;
        display: none;
        min-width: 80px;
        height: auto;
        background: none repeat scroll 0 0 #ffffff;
        border: 1px solid #6f257f;
        padding: 5px;
        text-align: left;
      }
    </style>
```

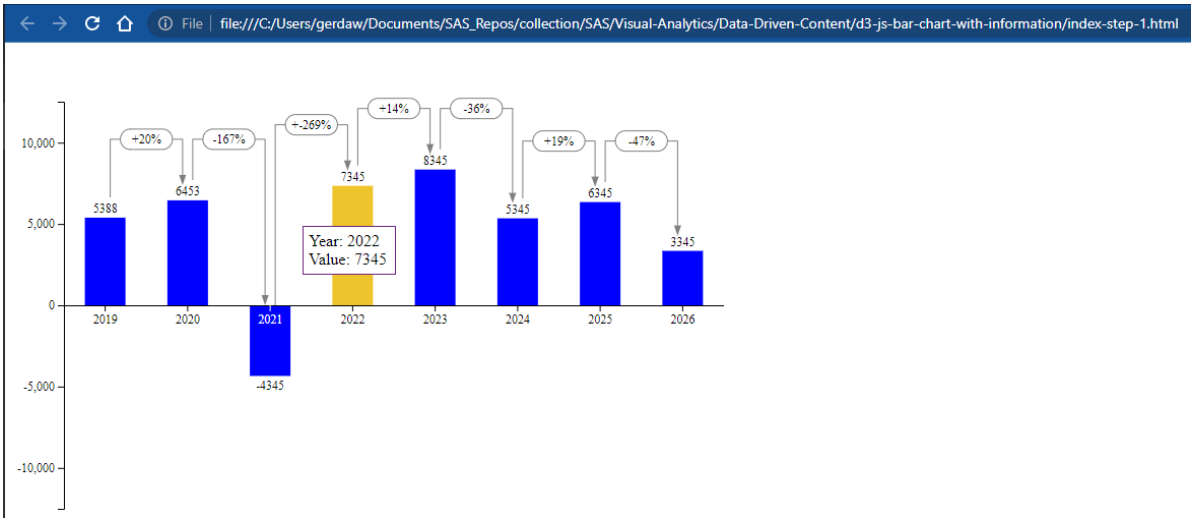
Now in the *body* tag first add the HTML from the answer - please note that this pulls in a script form the public internet, if this is not allowed at your organization download the file from the URL in the *script* tag, save it along side the *index.html*, adjust the src attribute to be *src='./d3.min.js'* and upload the file to your webserver:

```
    <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/5.7.0/d3.min.js">
</script>
    <div id="graph"></div>
```

And finally we are adding a opening and closing *script* tag after the graph div and paste the JavaScript from the answer into it:

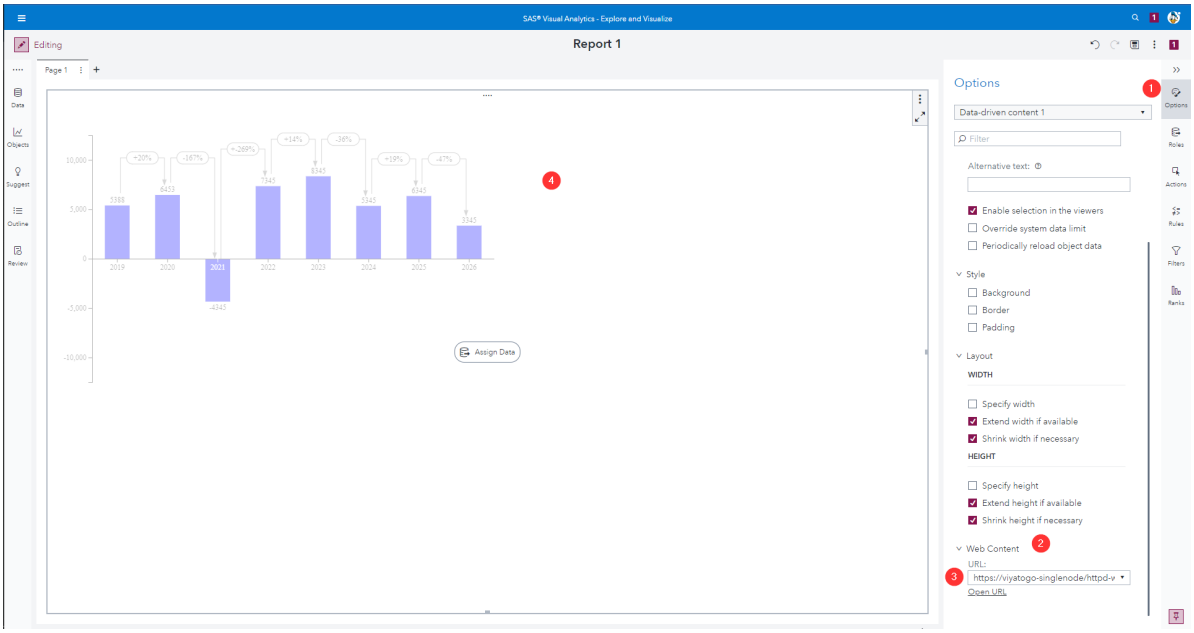
```
    <script>
      // JavaScript from Answer
    </script>
```

Now you can just double click the index.html in your file explorer, it should open up in a Web Browser and look like this:



You can now take this *index.html* file and upload it to a web server that your SAS Viya server can reach. I am using a simple Apache Webserver that is deployed in its own namespace in the same cluster as my SAS Viya environment, to which I can upload files. Upload the *index.html* to the webserver and note down its URL, for me it is *https://viyatogo-singlenode/httpd-webserver/d3-js-bar-chart-with-information/index-step-1.html*. You can also take a look at the SAS provided DDC-Server deployment if you want to.

Now lets head on over to SAS Visual Analytics, create a new report (you can start without data), search in the Objects pane for Data Driven Content, drag & drop it into the center of the report. Now on the right hand side open the Options pane **(1)**, scroll down to the Web Content section **(2)**, replace the URL with the one from your webserver **(3)** and click out of the input field so that the object can refresh in the center **(4)**:



Now the application is running with static data and can be looked at inside of SAS Visual Analytics.

Step 2: Making it Interactive

Now we want to run this application but send the data from SAS Visual Analytics to our application through the DDC. This chapter is divided into two steps the first is about uploading the example data to CAS and the second on modifying the *index.html* so that it can receive data from SAS Visual Analytics.

Getting data into SAS

Now let us take the data from the example JavaScript and upload it to CAS - you may note that we are saving the data as just one row, we will change that in the next step - you can run this inside of SAS Studio:

```
cas mysess;
libname casuser cas caslib='casuser';

proc casutil;
    droptable incaslib='casuser' casdata='sample_bar_chart_data' quiet;
run;

* Read the array of JSON objects as a string into SAS;
data casuser.sample_bar_chart_data(promote=yes);
    barData = '[{
"Time": "2019",
"Value": 5388
  },
  {
"Time": "2020",
"Value": 6453
  },
  {
"Time": "2021",
"Value": -4345
  },
  {
"Time": "2022",
```

```
"Value": 7345
  },
  {
"Time": "2023",
"Value": 8345
  },
  {
"Time": "2024",
"Value": 5345
  },
  {
"Time": "2025",
"Value": 6345
  },
  {
"Time": "2026",
"Value": 3345
  }]';
run;

cas mysess terminate;
```

Receiving the data

SAS provides some amazing helper functions to make use of inside of a DDC embedded application you can find them util folder and even one specfic for d3.js.

For this guide we will not make use of them, but rather work from first principles. The baseline code for this is provided in the SAS documentation:

```
      if (window.addEventListener) {
        // For standards-compliant web browsers
        window.addEventListener('message', onMessage, false);
      } else {
        window.attachEvent('onmessage', onMessage);
      }

      // Retrieve data and begin processing
      function onMessage(event) {
        if (event && event.data) {
          console.log(event)
          console.log('*************************')
          console.log(event.data)
        }
      }
```

If we would just add this to our script tag, upload it the webserver, add the *casuser.sample_bar_chart_data* data set to our report, assign barData variable as a role to our DDC, open up the developer console of our browser (keyboard short cut is *F12*, or use the three vertical dots > More tools > Developer tools) and the we would see the below output. The first section is the complete event that is passed to our application, the two most notable attributes are the data **(1)**, more about this in the next section, and the origin **(2)**, where the data was sent from. The second section concerns just the data that was passed into our application. The availableRowCount **(3)** indicates how many rows we have received, the columns **(4)** contains an

array of JavaScript objects describing each column and finally our array of data **(5)**, which contains what we are looking for:

Ok now that we know how to pass data into our object let us refine the JavaScript to actually take advantage of it. In the first step we will parse our returned String:

```
// Retrieve data and begin processing
function onMessage(event) {
  if (event && event.data) {
    let barData1 = JSON.parse(event.data.data);
    console.log(barData1);
  }
}
```

This will produce an array of JavaScript objects exactly like the sample we add in the step before. Now we need to do one very important thing, we need to create the chart once we have received data. We will create a function called *createBarChart* and pass the parsed data into it. The function will contain everything from the original answer, except the creation of the barData variable. Because of how the d3.js code is implemented it actually creates a new *svg* every time it is called so we need to clean up before we create a new one. Then we have to make a call to that function in our *onMessage* received function. Our total JavaScript will look like this:

```
if (window.addEventListener) {
  // For standards-compliant web browsers
  window.addEventListener('message', onMessage, false);
} else {
  window.attachEvent('onmessage', onMessage);
}

// Retrieve data and begin processing
function onMessage(event) {
  if (event && event.data) {
    // Clean up the previous graph
    document.getElementById('graph').innerHTML = '';
    // Parse the incoming data from SAS Visual Analytics
    let barData = JSON.parse(event.data.data);
    // Create a new bar chart
    createBarChart(barData);
  }
}

/**
```

```
       * createBarChart, function to create a d3.js bar chart with negative
values and tooltips
       * @param {Array} barData - an Array of JavaScript objects, with two
elements, the first for the x-axis, second for the y-axis
       **/
      function createBarChart(barData) {
        // JavaScript from answer, except const barData = [...]
      }
```

Now upload this new and improved *index.html* to our webserver, add the data source to our VA report, add the barData variable as a role to the DDC and enjoy the view.

Step 3: Making it Run Properly

We are ready to take the final step on our journey and really customize the chart so that it really feels SAS Visual Analytics native. We want to do three things here:

1. Actually send row and column based data into our DDC, instead of one string containing all data
2. Make the tooltips contain the variable name defined in SAS Visual Analytics
3. Add message that guides the user on the usage of this application

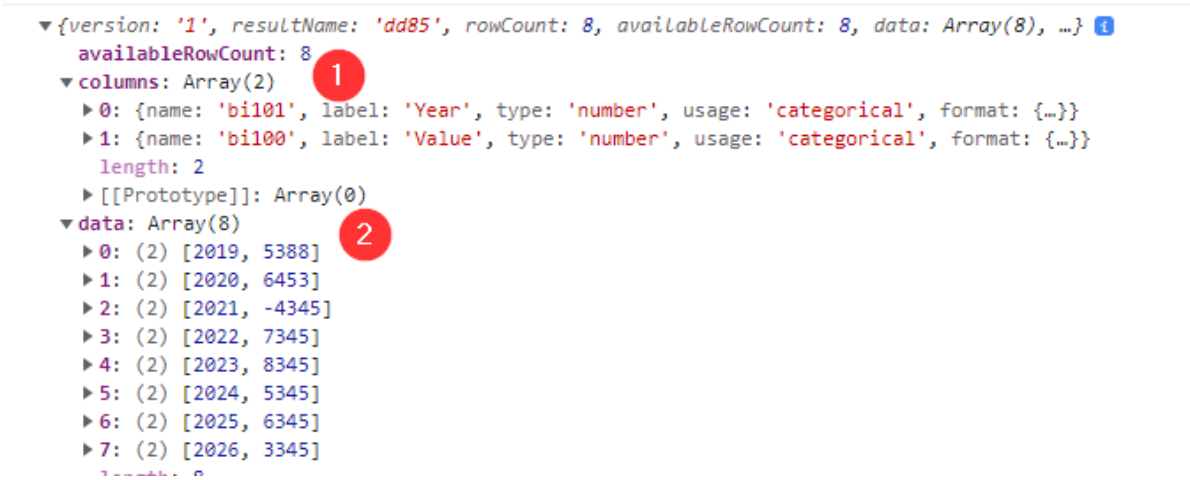
Getting better data

This time around we will read the example data as a real table, please run the following code inside of SAS Studio and add it to your SAS Visual Analytics report (change their type from Measure to Category while you are at it (I also capitalized the variable names in the report directly)):

```
cas mysess;
libname casuser cas caslib='casuser';

proc casutil;
    droptable incaslib='casuser' casdata='sample_bar_chart_data_2' quiet;
run;

data casuser.sample_bar_chart_data_2(promote=yes);
    input year value;
datalines;
2019 5388
2020 6453
2021 -4345
2022 7345
2023 8345
2024 5345
2025 6345
2026 3345
;
run;

cas mysess terminate;
```

Receiving Real Data

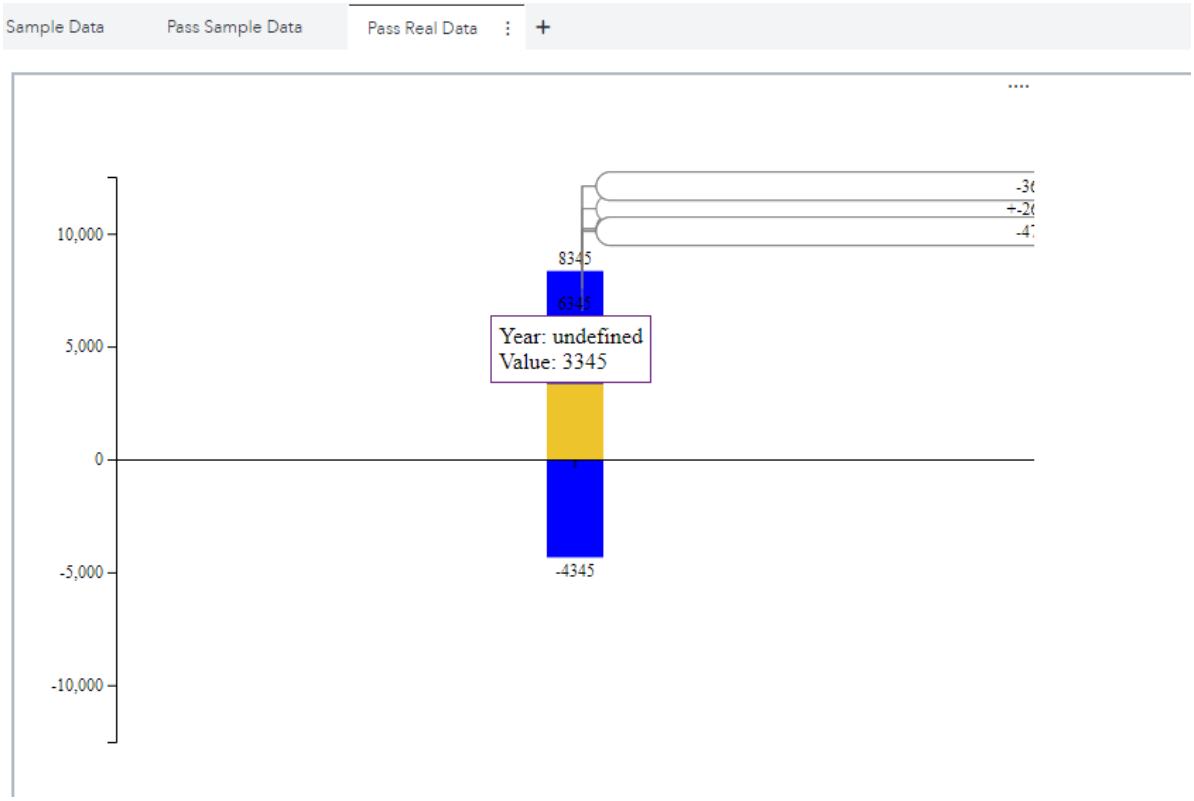
Let us pass our two variables into the Roles pane (I put the Year first and the Value second). Now if we just take a look at the data in the console, there is two interesting things to note. First is the the column attribute **(1)** now contains two elements and the order of the elements corresponds to the order in the Roles pane. Second the data attribute **(2)** now contains multiple rows each being an array of the column values (note again that the order corresponds to the order we specified in the Roles pane):



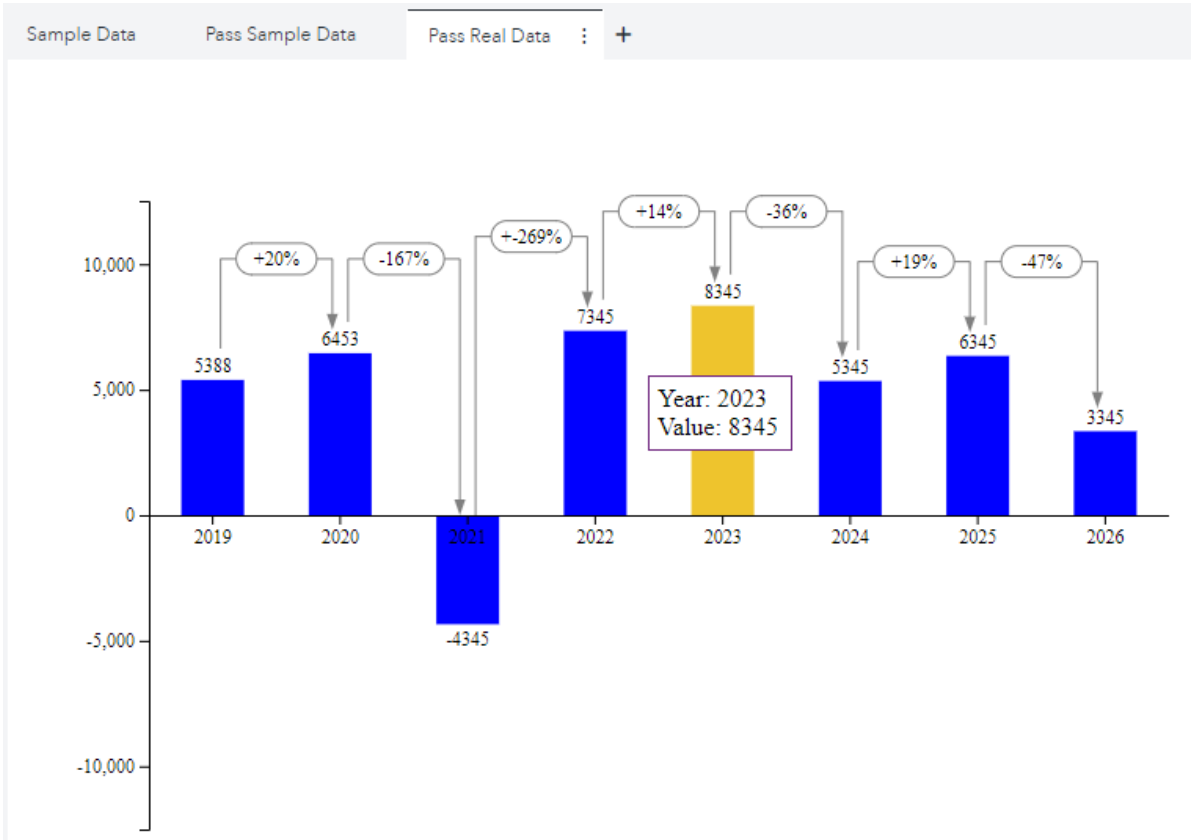
Now what we need is to transform this into an array of JavaScript objects:

```
function onMessage(event) {
  if (event && event.data) {
    // Bring the data into the correct format
    let xLabel = event.data.columns[0].label;
    let yLabel = event.data.columns[1].label;

    let barData = event.data.data.map((row) => {
      return { [xLabel]: row[0], [yLabel]: row[1] };
    });

    // Clean up the previous graph
    document.getElementById('graph').innerHTML = '';
    // Create a new bar chart
    createBarChart(barData);
  }
}
```

If we uploaded this two our webserver we would get the following output:



This indicates while our data was passed correctly, the original code must still have some hard references to variable names in there. Taking a look back that could be the case because the original example named it Time and not Year - so lets correct that inside of Data pane and voila:



Making it variable

Ok hardcoded values are nice and all but we want this to be far more flexible. So we will pass the *xLabel* and *yLabel* variables into to our graph function and make us of them there:

```
          // Create a new bar chart
          createBarChart(barData, xLabel, yLabel);
```

In the function we have to make quite extensive changes. We have to replace references to Time (I found 4) and Value (I found 25). As these are used as a JavaScript object key reference via dot notation we have to replace the *.Time* and *.Value* with *[xLabel]* and *[yLabel]* respectively. There is two special cases that we have to adjust that relate to the tooltip. First we need to soften the comparison for tooltip creation as the original assumed the year to be a string and we do not make this assumption:

```
// Original
const data = barData.find((d) => d.Time === text);
// Adjusted
const data = barData.find((d) => d[xLabel] == text);
```

And the actual text of the tooltip, as it has the values Time and Value embedded in the text and we want to use the our variables again:

```
// Original
.html('Year: ' + d.Time + '<br>' + 'Value: ' + d.Value);
// Adjusted
.html(`${xLabel}: ` + d[xLabel] + '<br>' + `${yLabel}: ` + d[yLabel]);
```

The full function code is below:

```
      /**
       * createBarChart, function to create a d3.js bar chart with negative
values and tooltips
       * @param {Array} barData - an Array of JavaScript objects, with two
elements, the first for the x-axis, second for the y-axis
       * @param {String} xLabel - Name of the X Axis value in the Object
       * @param {String} yLabel - Name of the Y Axis value in the Object
       **/
      function createBarChart(barData, xLabel, yLabel) {
        const container = d3.select('#graph');
        const divWidth = parseInt(container.style('width'));
        const divHeight = parseInt(container.style('height'));

        // Consider this width and Height are dynamic for div "graphID" because
I am trying to responsive design
        const margin = {
          top: 50,
          right: 50,
          bottom: 50,
          left: 50,
        };
        const width = divWidth - margin.left - margin.right;
        const height = divHeight - margin.top - margin.bottom;

        //To add svg in the visualization node i.e Dome node
        const svg = container
          .append('svg')
          .attr('width', divWidth)
```

```
    .attr('height', divHeight)
    .append('g')
    .attr('transform', `translate(${margin.left},${margin.top})`);

  //To add tooltip for bar
  const tooltip = d3
    .select('body')
    .append('div')
    .attr('class', 'toolTip');

  const defs = svg.append('defs');

  const marker = defs
    .append('marker')
    .attr('id', 'arrowhead')
    .attr('markerWidth', '10')
    .attr('markerHeight', '7')
    .attr('refX', '0')
    .attr('refY', '3.5')
    .attr('orient', 'auto');

  const polygon = marker
    .append('polygon')
    .attr('fill', 'gray')
    .attr('points', '0 0, 10 3.5, 0 7');

  const xScale = d3
    .scaleBand()
    .domain(barData.map((d) => d[xLabel]))
    .range([0, width + margin.right]);

  const xAxis = d3.axisBottom(xScale);

  //Adding g attribute to svg for x axis
  const yAxisMax =
    barData.reduce((max, item) => Math.max(max, item[yLabel]), 0) * 1.5;

  const yAxisMin =
    barData.reduce((min, item) => Math.min(min, item[yLabel]), 0) * 1.5;

  const yAxisRange = Math.max(yAxisMax, Math.abs(yAxisMin));

  const yScale = d3
    .scaleLinear()
    .domain([-yAxisRange, yAxisRange])
    .range([height, 0]);

  const yAxis = d3.axisLeft(yScale).ticks(4);

  svg.append('g').call(yAxis);

  const bars = svg
    .selectAll('g.bar')
    .data(barData)
    .enter()
```

```
    .append('g')
    .classed('bar', true)
    .attr(
      'transform',
      (d) => `translate(${xScale(d[xLabel]) + xScale.bandwidth() / 2}, 0)`
    );

  bars
    .append('rect')
    .attr('x', -20)
    .attr('width', 40)
    .attr('y', (d) => Math.min(yScale(d[yLabel]), height / 2))
    .attr('height', (d) =>
      d[yLabel] > 0
        ? height / 2 - yScale(d[yLabel])
        : yScale(d[yLabel]) - height / 2
    )
    .attr('fill', 'blue')
    .on('mousemove', onMouseOver)
    .on('mouseout', onMouseOut);

  function onMouseOver(d, i) {
    tooltip
      .style('left', d3.event.pageX - 50 + 'px')
      .style('top', d3.event.pageY - 70 + 'px')
      .style('display', 'inline-block')
      .html(
        `${xLabel}: ` + d[xLabel] + '<br>' + `${yLabel}: ` + d[yLabel]
      );
    d3.select(this).attr('fill', '#eec42d');
  }

  function onMouseOut(d, i) {
    tooltip.style('display', 'none');
    d3.select(this).attr('fill', 'blue');
  }

  bars
    .append('text')
    .text((d) => d[yLabel])
    .attr('text-anchor', 'middle')
    .attr('alignment-baseline', (d) =>
      d[yLabel] > 0 ? 'baseline' : 'hanging'
    )
    .attr('y', (d) => yScale(d[yLabel]))
    .attr('dy', (d) => (d[yLabel] > 0 ? -5 : 5));

  bars
    .filter((d, i) => i < barData.length - 1)
    .append('path')
    .attr(
      'd',
      (d, i) =>
        `M 5,${Math.min(yScale(d[yLabel]) - 20, height / 2)} V ${
          Math.min(yScale(d[yLabel]), yScale(barData[i + 1][yLabel])) - 60
```

```
        } H ${xScale.bandwidth() - 5} V ${Math.min(
          yScale(barData[i + 1][yLabel]) - 25,
          height / 2 - 10
        )}`
    )
    .style('stroke', 'gray')
    .style('fill', 'none')
    .attr('marker-end', 'url(#arrowhead)');

  bars
    .filter((d, i) => i < barData.length - 1)
    .append('rect')
    .attr('x', 15)
    .attr(
      'y',
      (d, i) =>
        Math.min(yScale(d[yLabel]), yScale(barData[i + 1][yLabel])) - 70
    )
    .attr('width', xScale.bandwidth() - 30)
    .attr('height', 20)
    .attr('rx', 10)
    .style('fill', 'white')
    .style('stroke', 'gray');

  bars
    .filter((d, i) => i < barData.length - 1)
    .append('text')
    .text(
      (d, i) =>
        `${barData[i + 1][yLabel] > d[yLabel] ? '+' : ''}${Math.round(
          (barData[i + 1][yLabel] / d[yLabel]) * 100 - 100
        )}%`
    )
    .attr('x', xScale.bandwidth() / 2)
    .attr(
      'y',
      (d, i) =>
        Math.min(yScale(d[yLabel]), yScale(barData[i + 1][yLabel])) - 56
    )
    .attr('text-anchor', 'middle')
    .style('fill', 'black');

  const xAxisG = svg
    .append('g')
    .attr('transform', `translate(0,${height / 2})`)
    .call(xAxis);

  xAxisG.selectAll('.tick').each(function () {
    const tick = d3.select(this);
    const text = tick.select('text').text();
    const data = barData.find((d) => d[xLabel] == text);
    if (data[yLabel] < 0) {
      tick.select('text').style('fill', 'white');
      tick.select('line').style('stroke', 'white');
    }
```

```
      });
    }
```

What a journey we have completely converted this string d3.js graph into a reusable component.

A Message to the User

We could technically stop here and be done with it. The problem would be if somebody just wanted to use this then they would have no idea what variables to pass into this application, or how many or if the order matters - lets change that. **Note:** The complete implementation of this will be named *index.html* as I regard it as *"the solution"*.

We can make send message back to SAS Visual Analytics, the baseline code can again be found in the SAS documentation - we will also add the message as a text directly in the element just to make debugging super easy:

```
    function sendMessage(message) {
      var url =
        window.location != window.parent.location
          ? document.referrer
          : document.location.href;
      window.parent.postMessage(message, url);
    }

    // Send instructional message
    function sendNoDataMessage(resultName) {
      let textMessage =
        'This Application requires two variables as input. The first is the
value for the x-axis (any type) and second value for the y-axis (numeric).';
      document.getElementById('graph').innerText = textMessage;

      var message = {
        resultName: resultName,
        message: textMessage,
      };
      sendMessage(message);
    }

    // Retrieve data and begin processing
    function onMessage(event) {
      if (event && event.data && event.data.columns.length === 2) {
        // Bring the data into the correct format
        let xLabel = event.data.columns[0].label;
        let yLabel = event.data.columns[1].label;

        let barData = event.data.data.map((row) => {
          return { [xLabel]: row[0], [yLabel]: row[1] };
        });

        // Clean up the previous graph
        document.getElementById('graph').innerHTML = '';
        // Create a new bar chart
        createBarChart(barData, xLabel, yLabel);
      } else {
        sendNoDataMessage('Please specify input');
```

```
        }
      }
```

And that is it now others can also use our little graph if they have the URL - if you want to make it even easier for others to use this take a look at the next chapter.

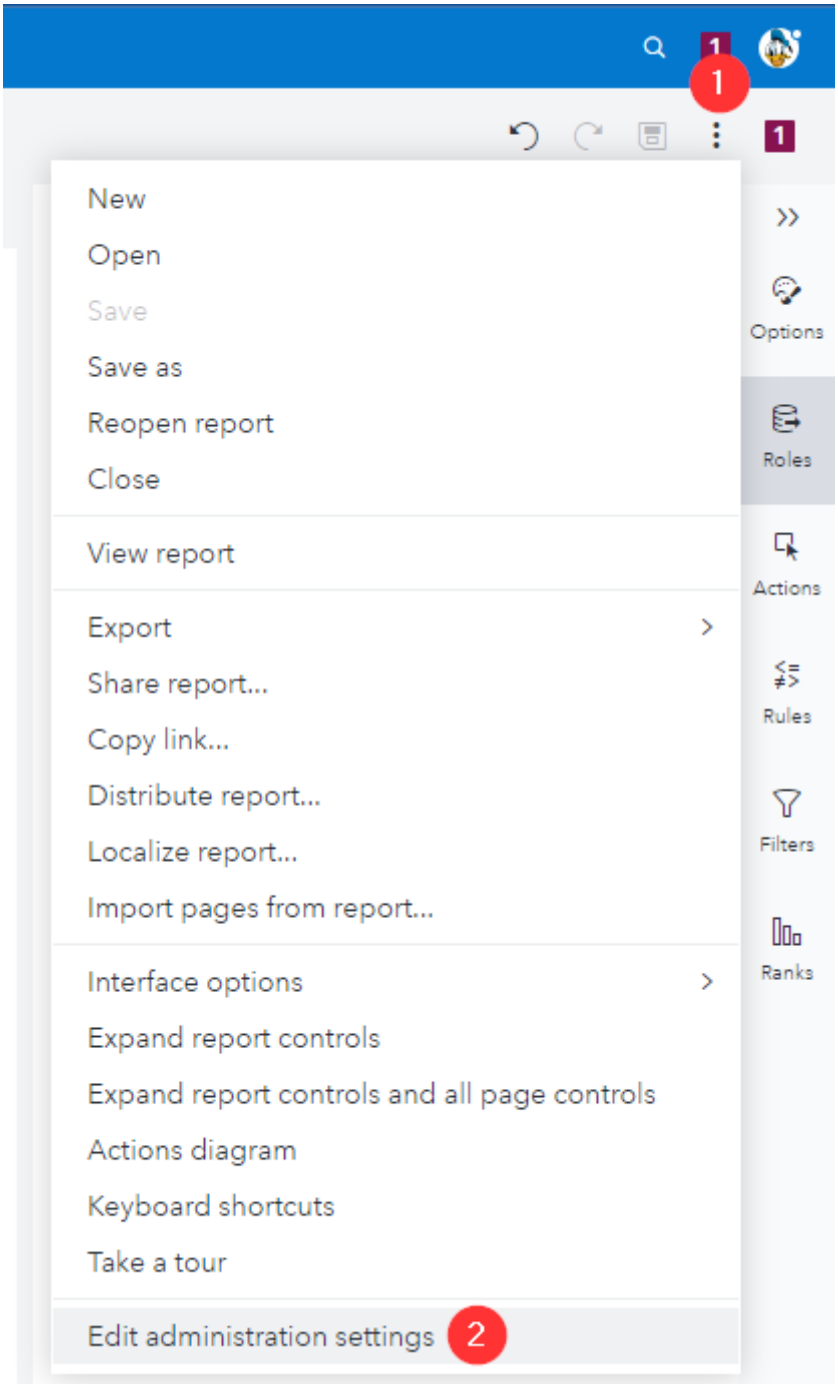
Making Applications Reusable

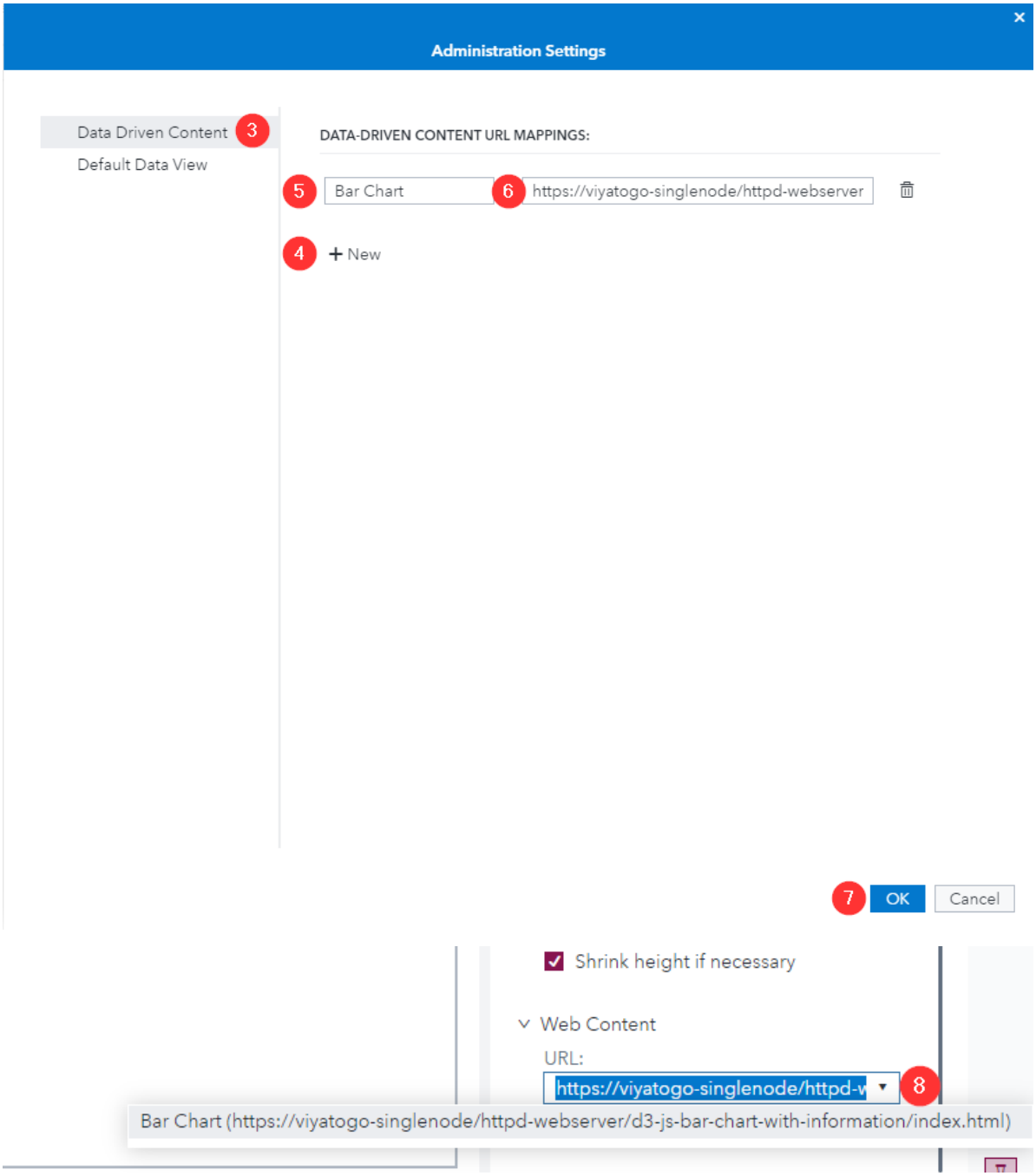
There are two major ways of making your custom DDC reusable, *URL Mapping* (intended for DDC object users) and *Object Template* (intended for all SAS Visual Analytics Report Creators). Another way to make them reusable is to store them as part of a *Page Template*, but as that is more generic in its functionality I will not cover it in details here - take a look at the SAS Documentation.

URL Mapping

This way is best if you want to make your DDC application available to other SAS Visual Analytics Report Creators that know how to use DDC objects and you just want to make it easier for them to find the URL to your application. To use this functionality you need to have permissions for */SASVisualAnalytics/rest/customGraphTypes* (by default this is enabled for all Authenticated Users).

Click on the three vertical dots on the report level **(1)**, then click on Edit administration settings **(2)**, in the pop-up ensure that you are in the Data Driven Content section **(3)**, click New **(4)**, fill in reference name **(5)** - e.g. *Bar Chart*, enter the URL to the application **(6)** - e.g. *https://viyatogo-single node/httpd-webserver/d3-js-bar-chart-with-information/index.html*, click OK **(7)** and finally if you now go to the Options pane of a DDC object and click on the URL dropdown you will see our URL Mapping **(8)** - e.g. *Bar Chart*:


1
New
Open
Save
Save as
Reopen report
Close
View report
Export
Share report...
Copy link...
Distribute report...
Localize report...
Import pages from report...
Interface options
Expand report controls
Expand report controls and all page controls
Actions diagram
Keyboard shortcuts
Take a tour
Edit administration settings
2
Options
Roles
Actions
Rules
Filters
Ranks

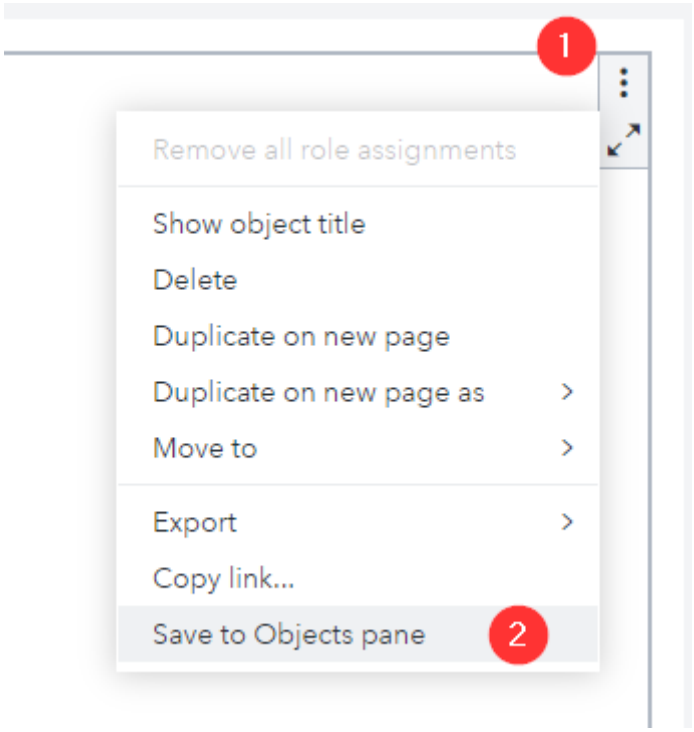


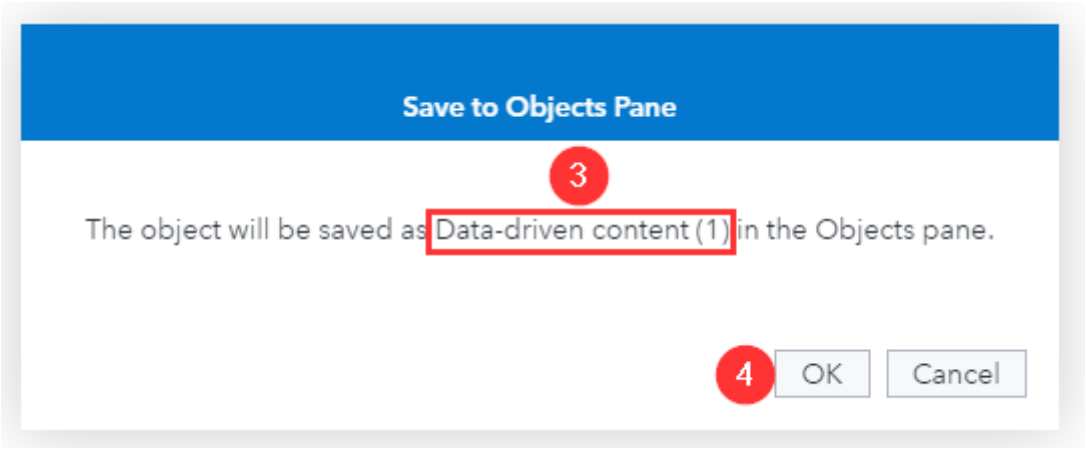
You can find out more about *URL Mappings* in the SAS Documentation.

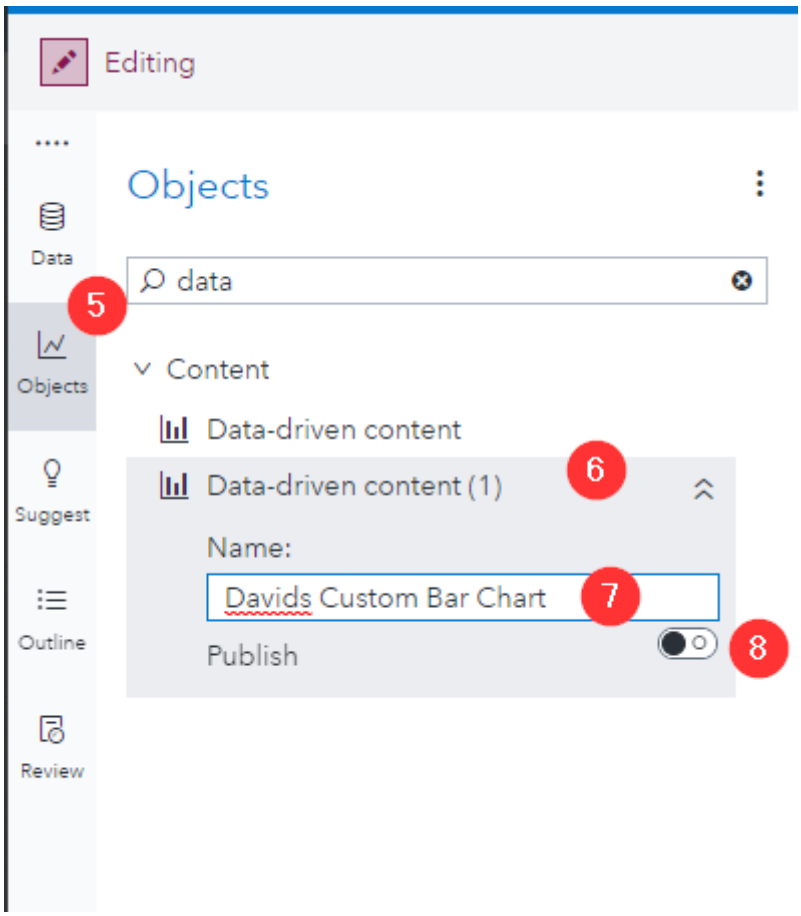
Object Template

Object Templates are perfect to use if you want to make your DDC application broadly available to everybody that is creating reports in SAS Visual Analytics. For the ability to make objects available to all you have to be a SAS Visual Analytics application administrator.

On the defined DDC object in your report go to the three vertical dots **(1)**, click Save to Objects pane **(2)**, in the pop confirm the name under which your application will be saved to the Object pane **(3)** - we will change this name later on, click OK **(4)**, now search for the object in the Object pane **(5)**, expand its properties **(6)**, change the name to something more identifiable **(7)** - i.e. *Davids Custom Bar Chart* and click the Publish button **(8)** - if you do not publish the object will only be available to you:


1
Remove all role assignments
Show object title
Delete
Duplicate on new page
Duplicate on new page as
Move to
Export
Copy link...
Save to Objects pane
2


Save to Objects Pane
3
The object will be saved as Data-driven content (1) in the Objects pane.
4
OK
Cancel



You can find out more about *Object Templates* in the SAS Documentation.

Additional Reading Material

SAS GitHub Repository for helper functions and a collection of samples: https://github.com/sassoftware/sas-visualanalytics-thirdpartyvisualizations

How to guide on deploying a custom web application for DDC usage: https://communities.sas.com/t5/SAS-Communities-Library/Deploy-a-custom-web-application-in-the-cloud-for-Data-Driven/ta-p/687141

There is also a create usage tutorial on YouTube: https://www.youtube.com/watch?v=EFQPL_FYpdk