Paper 3169-2019

How to Use JavaScript Libraries Easily with SAS® Stored Processes

Philip Mason, Wood Street Consultants Ltd.

ABSTRACT

Looking at websites, you can see a wide variety of ways to display data and interact with users. Much of what you see in a web browser is done by using a combination of HTML, CSS, and JavaScript. There are libraries of CSS and JavaScript code that provide a massive amount of functionality that we can use in our applications. Many of these libraries are free or very inexpensive. I describe some of the best libraries that I know of and how to use them with SAS®. SAS® Stored Processes enable SAS code to be run from a web browser using the SAS® Stored Process Web Application. They enable us to write SAS code that works with web technologies so that we can have SAS in the back end and HTML, CSS, or JavaScript in the front end. In this presentation, I show you how to get SAS to use these libraries to make impressive applications with SAS Stored Processes in the web browser.

INTRODUCTION

The World Wide Web is well known to us all. Every day as we surf the web we see a range of impressive web sites demonstrating what can be done in a web browser. I have found that my clients (especially management) get used to seeing all these impressive things on the web, but are often disappointed when shown a piece of standard SAS output. The classic complaint was how tables and graphs produced by SAS didn't look as good as EXCEL ones, or have their functionality. Of course there are things we can do in SAS to change that, and if we have access to SAS products like Visual Analytics then it isn't a problem at all. However I am going to describe an alternate approach where we can use the power of the web and its user interface combined with the power of SAS in the backend.

WHY USE A WEB BROWSER AS A USER INTERFACE

Web browsers are great pieces of software which can take a range of files and produce powerful user interfaces from them. I will make a few comments about some of the key file types but this paper won't be going into much depth on anything except the parts relating to JavaScript libraries. So you might need to go and check some other resources such as the excellent W3 Schools web site (https://www.w3schools.com), which is the best place to learn about web technologies for the beginner.

HTML

HTML is for defining the content of a web page. Generally you will start with some HTML, which can give you a basic layout and content for your page – such as a table with an image of a graph being displayed. There are different standards within HTML but the main thing is to ensure you are using HTML 5 which is widely supported and has great functionality. HTML is made up of tags which enable you to do many things. Here are some of the tags:

- <Form> - You can collect information from users with text fields, drop down menus, check boxes, buttons etc. Information collected on forms can then be sent to other HTML pages.
- <iFrame> - lets you embed another web page in your web page.

- <h1>, <h2>, etc. – different levels of heading.
- <div> - a great tag which is kind of a place holder that can be filled up with content (when we use a little JavaScript).
- <canvas> - lets you draw graphics on the web page

CSS

CSS describes the style of HTML. So you can set the color of things, size of fonts, border attributes, position on the page, animations and much more. When you declare your CSS rule you list a selector followed by one or more property/value combinations. For example the following would select the paragraph HTML tag and make the fonts light blue and 10 point in size:

p {**color**: lightblue ; **font-size**: 10px ;}

JAVASCRIPT

JavaScript is the language of the web. If you want to do some programming on a web page then JavaScript is the first choice. If HTML handles the content of a page, and CSS handles the style, then JavaScript handles the behavior of the page. JavaScript is a full programming language which lets you control the HTML and CSS dynamically, as well as interacting with the user. It means you can do almost anything that you can think of that a user interface can do.

ADVANTAGES OF LIBRARIES

There are many libraries of HTML, CSS and JavaScript code that can be downloaded from the web. The libraries provide a collection of code that has been written to make it easier to do things. Although you could avoid libraries and do everything using your own code, you can save huge amounts of time and effort using libraries, and take advantage of the expertise of others in order to accomplish far more advanced user interfaces. Some libraries are so popular that they have become defacto standards for producing web sites and carrying out certain functionality.

CSS libraries

A popular CSS framework that was created by Twitter is Bootstrap 4, and it is highly popular around the web. You can find out about it here https://www.w3schools.com/bootstrap4/default.asp. Some of the useful things that this provides is a way to layout a web page in sections, putting different content into each of the sections. It also provides lots of styling facilities such as spinners, progress bars, buttons, themes and much more.

Another great CSS library is font-awesome which gives you thousands of extra special characters/icons within a range of fonts.

JavaScript libraries

A popular JavaScript library is jQuery which aims to simplify JavaScript programming. You can read about that here https://www.w3schools.com/jquery/default.asp or https://jquery.com/.

There are a huge number of libraries that can produce graphics and tabular grids of data. Most web developers have their favorites, and my favorite ones (which are completely free for commercial use) are dygraphs (for time series graphs), dc.js (for Cross Filter graphs), muze (grids of graphs), pivotTable.js (pivot tables like EXCEL) and dataTables (EXCEL like grid in a web browser).

There are other libraries which can assist with programming such as underscore.js (for doing tricky things in JavaScript more easily) and moment.js (for working with dates & times).

I will be using the libraries mentioned above in many of the examples in this paper.

A FEW THINGS TO THINK ABOUT BEFORE YOU START PROGRAMMING

Editor - Do you have a good editor suitable for programming with HTML, CSS and JavaScript? If not then read on. On windows a good choice is NotePad++. On any platform a good choice is Microsoft VS Code. These editors will do things like FTP to a web server, context sensitive highlighting, code completion, integrate with GitHub, etc.

Web Browser – do you have to write code to work on one particular web browser? Or several? Find out how the web developer tools in that browser work. On most web pages you should be able to right click and press Inspect (or some similar text) to open up the web developer tools. In that you will see a console that will display JavaScript errors when you have them. You will also see a network tab which can show you all the various requests that run in the background of a page (like loading an image for a graph). These tools will become invaluable as you develop with javascript.

HOW DO WE GET THINGS FROM SAS TO THE WEB BROWSER?

SAS Stored Process Web Application – This makes it possible to use a URL in a web browser to run a SAS Stored Process on a SAS server. You can send data from your web browser via the web application to your stored process, run any SAS code on the server that you want to and then return data back to the web browser. So you could send some text the user has entered on the web page, some selections they have made from menus, a file they have selected to be uploaded to the server, etc. All that data is accessible to the stored process that you run, so you can access the text and selections the user made in the form of SAS macro variables and you can access the files from temporary locations recorded in SAS macro variables. You might have uploaded a CSV file and you could then use Proc Import to import it. With all that data you can run any SAS code you want to and then return data to the web page. You return data to a web page by writing to a special fileref location called **_webout**.

WHAT CAN WE SEND BACK TO THE WEB BROWSER?

Your stored process can write all kinds of things to **_webout**. You don't have to write anything to it, in which case the stored process would run and return nothing. You can use the %stpbegin and %stpend special macro programs provided by SAS which will open up an ODS destination for you and then any ODS output produced will be formatted for your web browser using HTML and CSS. This is the most easy way to get some web output back to your browser. However, a more powerful technique is to not use %stpbegin and %stpend, instead just writing content you need to **_webout**. We will see soon that we can write data out in a special format called JSON, we could write HTML & CSS code out, and an almost endless variety of other things.

HOW DO WE SEND THINGS BACK FROM A STORED PROCESS?

The simplest SAS code to send something back from a stored process would be something like this, which uses the stpbegin and stpeend macros to setup your ODS destination to write HTML code to _webout automatically when you generate ODS output with Proc Print.

```
%stpbegin ;
Proc print data=sashelp.class ;
Run ;
%stpend ;
```

Another really easy way to send something back to the browser would be just to use a data step and write something to _webout, which means that would be sent to the browser.

```
Data _null_ ;
  File _webout ;
  Put '<html><body><h1>Hello World!</h1></body></html>' ;
Run ;
```

Hopefully you get the idea that anything we write to _webout from a stored process that is being used from a web browser will be sent to the browser. This is a powerful technique because we can create all kinds of things to send back, and SAS help us to do that by providing some really useful procedures. The following procedures let us point their output to a fileref such as _webout, which means that they can send text directly to the web browser: PROC STREAM, PROC JSON and PROC EXPORT. We will cover some of their functionality as a need for it arises.

WHAT ABOUT THOSE JAVASCRIPT LIBRARIES?

Now that we have covered some of the basics we can move on to the focus for this paper. Although with ODS I can produce some nice output and I can enhance it by producing my own ODS templates or enhancing it with custom CSS in some instances, I cant get the User Experience I am looking for that I see in best of breed web applications. I could produce some complex custom HTML and CSS which could be streamed to the browser with Proc Stream and that enables me to do some impressive things, but still can fall short of the best that can be achieved. So I really want to use some JavaScript libraries which give me all kinds of functionality wrapped up and ready to be used with a little JavaScript programming. In my experience this is where most people give up as it seems quite complex and like you need to learn a new language in order to go down this route. However with a little knowledge and some simple examples I will provide I think you can get a long way towards getting value from JavaScript libraries.

JAVASCRIPT LIBRARY – DATATABLES

Let me introduce you to my favorite JavaScript library which can be found at https://datatables.net. You give it data and it makes a table (or grid) in the web browser. You can provide the data to it in many different ways. In the simplest form you could just have it defined in your JavaScript code, but you could also grab the data from an existing HTML table or call a stored process to provide the data. The table can be laid out in almost any way, columns sorted, filters applied, searched, data exported to CSV files, and many other features. It's free unless you want to do editing, and that is inexpensive. The documentation is amazing too.

The following example produces a simple table and here are some explanatory comments about the code:

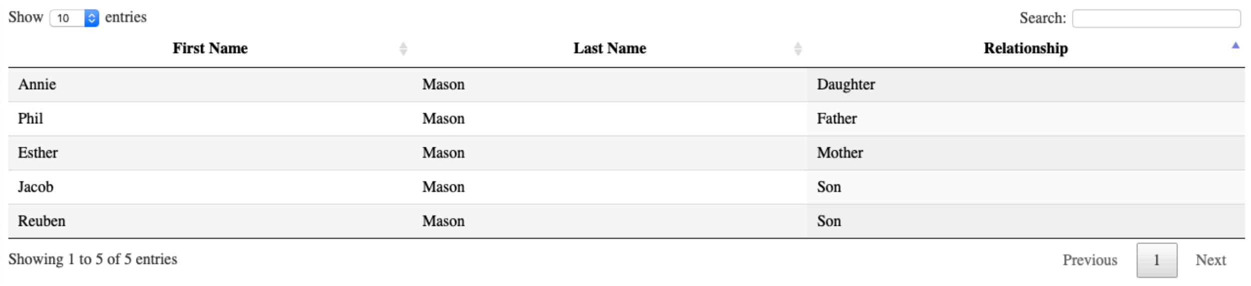
1) The link tag is used to point to a CSS library. DataTables comes with one which makes the tables look good and you can point to it on a CDN, which is a publicly available repository for libraries like this.
2) A script tag holds JavaScript, and if you specify the src attribute then you can point to a JavaScript library.
   a) We point to jQuery which lets us use some shorthand functions that start with a $ sign in the code.
   b) We also point to the datatables library which hold all the JavaScript code to create and make our table work.

3) We create a variable in JavaScript called dataset in which we define an array of arrays. Arrays use [] around them. JavaScript objects use {} around them.
    a) The outer array represents the rows in the table.
    b) The inner arrays represent the values for each column within a row.
4) The bold part below defines a function in which we put code that will run when all the HTML on the page has finished loading. This is a common JavaScript technique because we need to wait for some HTML to be created before we can carry out JavaScript functions on it.
5) The text in red is how we actually create the table.
    a) $("#example") locates the section of HTML that we set the id to example.
    b) .DataTable() calls the datatable function on that section of HTML which creates the table.
    c) Within the brackets of the function we specify an object that defines the data to use and the columns.
6) Remember that JavaScript cares about case – so dataset, Dataset and DATASET are all different variable names.
7) You can copy the code below into a text editor and save it with an html extension. Then open in in a web browser to see it work.
8) You could generate this whole program from a stored process and mix in some macros with a proc stream, which could make quite a flexible way to create a table like this with SAS data, however there are better ways I will cover.

```
<html>
<head>
<link rel="stylesheet"
href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<script type="text/javascript" src="https://code.jquery.com/jquery-
3.3.1.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></scri
pt>
<script>
var dataset=[
    ["Phil","Mason","Father"],
    ["Esther","Mason","Mother"],
    ["Jacob","Mason","Son"],
    ["Annie","Mason","Daughter"],
    ["Reuben","Mason","Son"]
] ;
$(document).ready(function(){
    $("#example").DataTable({
        data: dataset,
        columns: [
            {title: "First Name"},
            {title: "Last Name"},
            {title: "Relationship"}
        ]
    }) ;
}) ;
</script>
```

```
</head>
<body>
<table class="display" id="example">
</table>
</body>
</html>
```

The following table is displayed in the web browser.



Show 10 entries Search:

First Name	Last Name	Relationship
Annie	Mason	Daughter
Phil	Mason	Father
Esther	Mason	Mother
Jacob	Mason	Son
Reuben	Mason	Son

Showing 1 to 5 of 5 entries Previous 1 Next

HOW DO WE GET DATA FROM A URL?

In the previous example you saw how we could define some data in a JavaScript variable which would then be displayed in a datatable, but there are other ways to access data. There is a section in the datatables documentation that describes how to use AJAX sourced data. AJAX (Asynchronous Javascript And Xml) is a way for the web browser to make a request in the background for something, so that other things can be happening at the same time. We can use the AJAX parameter in datatables to go and make an AJAX request to load some data from a file. Notice that here we have not defined the columns in the datatable function, but instead have just defined them in the table itself.

```
<html>
<head>
<link rel="stylesheet"
href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<script type="text/javascript" src="https://code.jquery.com/jquery-
3.3.1.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></scri
pt>
<script>
$(document).ready(function(){
    $("#example").DataTable({
        ajax: "http://d351tq92/example2_json.txt"
    }) ;
}) ;
</script>
</head>
<body>
<table class="display" id="example">
    <thead>
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Relationship</th>
        </tr>
    </thead>
```

```
</table>
</body>
</html>
```

The data in the file looks like this.

```
{
    "data": [
        ["Phil","Mason","Father"],
        ["Esther","Mason","Mother"],
        ["Jacob","Mason","Son"],
        ["Annie","Mason","Daughter"],
        ["Reuben","Mason","Son"]
    ]
}
```

HOW DO WE GET DATA FROM SAS?

Our web page could do an AJAX request to go and get some data from SAS rather than a file as we did in the previous example. We can use the built in feature of datatables to load data via AJAX into our table. So we could simply change the URL in the previous example like this, which will run the stored process example2.

```
$(document).ready(function(){
    $("#example").DataTable({
        ajax:
"http://d351tq92/SASStoredProcess/do?_program=/User+Folders/phil/My+Folder/
example2"
    }) ;
}) ;
```

In the example2 stored process we could simply have this source code, which writes out what we had in our file in the previous example.

```
data _null_ ;
   file _webout ;
   input ;
   put _infile_ ;
cards ;
{
    "data": [
        ["Phil","Mason","Father"],
        ["Esther","Mason","Mother"],
        ["Jacob","Mason","Son"],
        ["Annie","Mason","Daughter"],
        ["Reuben","Mason","Son"]
    ]
}
;;
run ;
```

Alternatively I think a better way is that we can make an AJAX request from the JavaScript code ourselves using jQuery and then load that data into a datatable.

There are several advantages to this method:

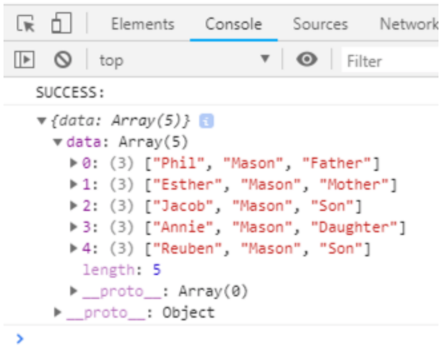
- When we make an AJAX request we can take different actions depending on whether it was successful or failed, and also actions when it completes. This allows us to do some error handling and make our web applications more robust.
- Once we have run an AJAX request and we have the data, then we are able to do multiple things with it. For instance we could load it into a table, but also create a graph and some data dependent titles.

The following code gets our data from the SAS Stored process using the jQuery **ajax** function which loads the JSON data into a JavaScript object. We use **datatype** to tell jQuery that we are expecting JSON data, and then it converts it automatically into a JavaScript object for us. We use **success** to define a function that will run when the AJAX request has completed successfully.

```
<html>
<head>
<link rel="stylesheet"
href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<script type="text/javascript" src="https://code.jquery.com/jquery-
3.3.1.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></scri
pt>
<script>
$(document).ready(function(){
    $.ajax({
        url:
"http://d351tq92/SASStoredProcess/do?_program=/User+Folders/phil/My+Folder/
example2",
        dataType: "json",
        success: function (data_from_stp) {
            console.log("SUCCESS:");
            console.log(data_from_stp);
            $("#example").DataTable({
                data: data_from_stp.data,
                columns: [
                    {title: "First"},
                    {title: "Last"},
                    {title: "Relationship"}
                ]
            });
        }, // success
        error: function (data) {
            console.log("ERROR:");
        } // error
    }); // ajax

}) ;
</script>
</head>
<body>
<table class="display" id="example">
</table>
</body>
</html>
```

It's really important to understand the code above and how it works. Let me make a few points and check you really have them firmly held in your head before you continue.

1) The **url** specifies the SAS Stored Process to run. The first part is the server (mine is called d351tq92), then **SASStoredProcess** which is the SAS Stored Process web application that enables stored processes to be run from a web server. After **do?** we have the parameters we are passing to the stored process. You must always specify **_program** which is the path to the stored process you want to run. You can optionally specify other parameters to pass to the stored process.
2) We are using the power of AJAX here, which is a standard feature of JavaScript. If an AJAX request is expecting some text formatted as JSON data and it receives that text, then it will read it into an object in JavaScript. It then runs the code you have in the **success** function.
3) Once you have your data in a JavaScript object you can do all kinds of things with it, including sending it to a DataTable function to populate a data grid.
4) If your JSON data has some kind of problem, then it will go into the **error** callback, which in this case just writes a message to the console. But this is useful to catch errors and you can then look at other information to work out what kind of error it was if you want to do something with that information.
5) The variable name I have called **data_from_stp** can actually be called anything. This is a variable name that will be used to hold the JavaScript object read from the JSON data coming from the URL.
6) jQuery functions are called using a **$** sign. Using **$** with something in brackets after it will select those things in HTML so that further jQuery functions just apply to those things. In our example code we have used **"#example"** which selects HTML that has an id of **"example"** (**#** means we are using an id). We could also select using the class of a tag, or name of a tag, and various other things.
7) Remember that in JavaScript we have arrays signified with **[]** and objects signified with **{}**. These can be nested so we can have an array of objects, or an object with arrays and objects in it, etc. Things in an object usually have a key and value. For example **{name:"Phil Mason"}** is an object with a key of **name** whose value is **"Phil Mason"**.
8) **console.log** is useful to write things to the JavaScript console which can be viewed by right clicking in your browser and selecting something like Inspect. In my web browser the code above shows me this (see below). Notice that objects can be expanded & collapsed for easier viewing.



```
SUCCESS:
▼{data: Array(5)}
  ▼data: Array(5)
    ▶0: (3) ["Phil", "Mason", "Father"]
    ▶1: (3) ["Esther", "Mason", "Mother"]
    ▶2: (3) ["Jacob", "Mason", "Son"]
    ▶3: (3) ["Annie", "Mason", "Daughter"]
    ▶4: (3) ["Reuben", "Mason", "Son"]
     length: 5
    ▶__proto__: Array(0)
  ▶__proto__: Object
>
```

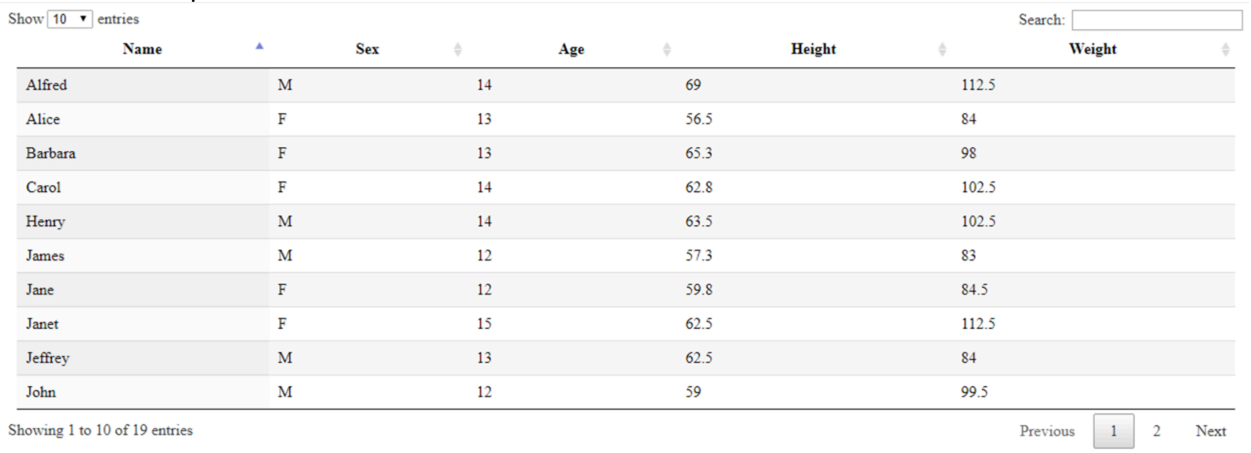
Here's a better example of a dataTable which displays sashelp.class in a datatable. The stored process code is very simple. We use **PROC JSON** to produce JSON data and send to **_webout**. Using **nosastags** cuts out some SAS info we don't need. Using **pretty** makes the JSON produced look better to humans. Then we make a JavaScript array, using **write open array** – and remembering to close the array when we are finished with it. The great thing that **export sashelp.class** does is it writes a JavaScript object for each row, with variable names as the keys and their values are the values of the variables on that row.

```
proc json out=_webout nosastags pretty ;
    write open array ;
    export sashelp.class ;
    write close ;
run ;
```

The JavaScript part is changed to the following. The important changes are that we now point to a different stored process with the code above, and we have defined the columns we want to display in the columns array. In that array each column has an object that defines it. You need to define a value for **data** that points to the Key in the JSON data whose value will be used in that column. I also specify a value for **title** which is used as the column title in the table.

```
$(document).ready(function(){
    $.ajax({
        url:
"http://d351tq92/SASStoredProcess/do?_program=/User+Folders/phil/My+Folder/
example5",
        dataType: "json",
        success: function (data_from_stp) {
            console.log("SUCCESS:");
            console.log(data_from_stp);
            $("#example").DataTable({
                data: data_from_stp,
                columns: [
                    {data: "Name",   title: "Name"},
                    {data: "Sex",    title: "Sex"},
                    {data: "Age",    title: "Age"},
                    {data: "Height", title: "Height"},
                    {data: "Weight", title: "Weight"}
                ]
            });
        }, // success
        error: function (data) {
            console.log("ERROR:");
        } // error
    }); // ajax
}) ;
</script>
```

And the table produced looks like this.



Show 10 entries   Search:

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5

Showing 1 to 10 of 19 entries   Previous 1 2 Next

I'm hoping all that was clear. If you want some more explanation you can check my papers from Global Forum over the last 10 years as I have been writing about this topic for a long time. Also I have a book coming out from SAS Press very soon which will cover all of this in detail.

Now that we understand how to get data from SAS into JavaScript objects it opens up the world of JavaScript libraries which are out there. Many of them are quite easy to use once you have your data in JavaScript. In the remainder of the paper I will show you how to use several of these libraries in a simple way. All of these libraries have a lot more functionality available too.

JAVASCRIPT LIBRARY – PIVOT TABLES

Pivot tables are a great feature of Microsoft EXCEL where you can drag variables from a table into a pivot table to make a cross tabulation of statistics. So if you were using some data like sashelp.orsales you might like to make a table with Quarters across the top, product categories down the left and total sales in each cell. The user could then work on the table and change the dimensions in use and the statistics in each cell. Well, with the pivot table JavaScript library you can do just that.

Keeping the code as similar as possible to the other examples we can make a few changes to get a pivot table in the browser. The HTML file would look like the next piece of source code. Note that we need the pivotTable JavaScript and CSS files, as well as the jQuery-UI JavaScript file which assists in the functionality. We create 2 **div** tags that we can create pivot tables in by selecting them by their **id** values. And in the success part of the AJAX call we simply select the div we want to create a pivot table in, and then call the pivotUI function passing the data to be used.

```
<html>
<head>
    <link rel="stylesheet" 
href="https://cdnjs.cloudflare.com/ajax/libs/pivottable/2.23.0/pivot.min.cs
s" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
3.3.1.js"></script>
    <script type="text/javascript" 
src="https://cdnjs.cloudflare.com/ajax/libs/pivottable/2.23.0/pivot.min.js"
></script>
```

```
    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.11.4/jquery-
ui.min.js"></script>
    <script>
    $(document).ready(function(){
        $.ajax({
            url:
"http://d351tq92/SASStoredProcess/do?_program=/User+Folders/phil/My+Folder/
example7",
            dataType: "json",
            success: function (data_from_stp) {
                console.log("SUCCESS:");
                console.log(data_from_stp);
                $("#table1").pivotUI(data_from_stp['sashelp.orsales']) ;
                $("#table2").pivotUI(data_from_stp['sashelp.snacks']) ;
            }, // success
            error: function (data) {
                console.log("ERROR:");
            } // error
        }); // ajax
    }) ;
    </script>
</head>
<body>
    <h3>sashelp.orsales (912 rows, 200k)</h3>
    <div id="table1"></div>
    <h3>sashelp.snacks (35,770 rows, 3meg)</h3>
    <div id="table2"></div>
</body>
</html>
```

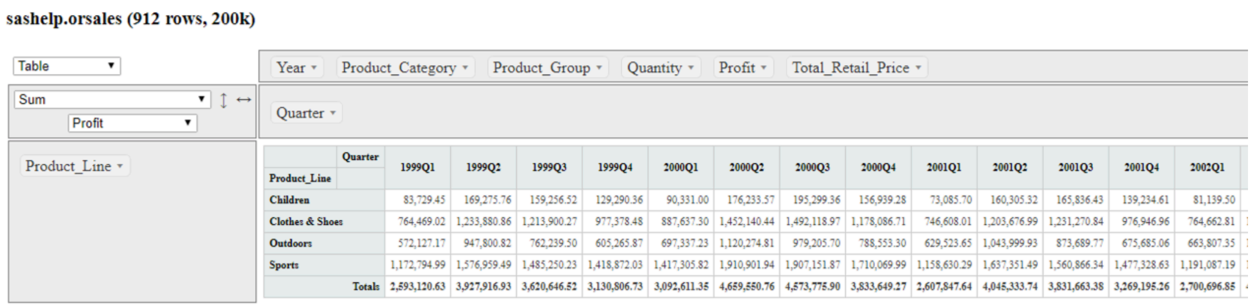
The data comes from a SAS stored process which has the code below.

```
proc json out=_webout pretty nosastags ;
   write open object ;
        write values "sashelp.orsales" ;
        write open array ;
             export sashelp.orsales ;
        write close ;

        write values "sashelp.snacks" ;
        write open array ;
             export sashelp.snacks ;
        write close ;
   write close ;
run ;
```

The start of the JSON data produced looks like this. After consulting the documentation for pivotTable I could see what was required, and I chose to put 2 arrays into an object – one for each SAS table I wanted to send. This makes it easier to send multiple tables for use in a single request. Then to use them I just have to use the correct notation to pick the appropriate array from the object.

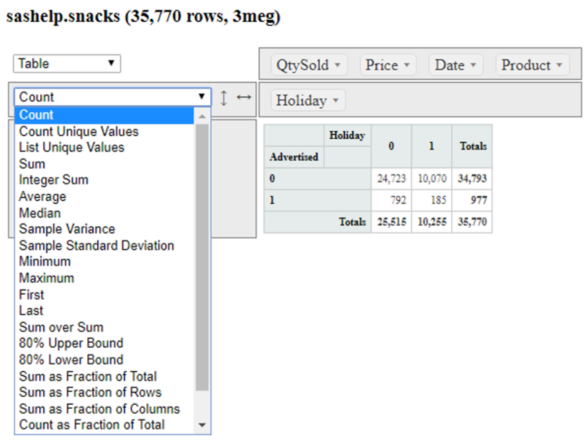
```
{
  "sashelp.orsales": [
    {
      "Year": 1999,
      "Quarter": "1999Q1",
      "Product_Line": "Children",
      "Product_Category": "Children Sports",
      "Product_Group": "A-Team, Kids",
      "Quantity": 286,
      "Profit": 4980.15,
      "Total_Retail_Price": 8990.9
    },
    {
      "Year": 1999,
      "Quarter": "1999Q1",
```

Here is what we get in the web browser.



sashelp.orsales (912 rows, 200k)

Product_Line \ Quarter	1999Q1	1999Q2	1999Q3	1999Q4	2000Q1	2000Q2	2000Q3	2000Q4	2001Q1	2001Q2	2001Q3	2001Q4	2002Q1
Children	83,729.45	169,275.76	159,256.52	129,290.36	90,331.00	176,233.57	195,299.36	156,939.28	73,085.70	160,305.32	165,836.43	139,234.61	81,139.50
Clothes & Shoes	764,469.02	1,233,880.86	1,213,900.27	977,378.48	887,637.30	1,452,140.44	1,492,118.97	1,178,086.71	746,608.01	1,203,676.99	1,231,270.84	976,946.96	764,662.81
Outdoors	572,127.17	947,800.82	762,239.50	605,265.87	697,337.23	1,120,274.81	979,205.70	788,553.30	629,523.65	1,043,999.93	873,689.77	675,685.06	663,807.35
Sports	1,172,794.99	1,576,959.49	1,485,250.23	1,418,872.03	1,417,305.82	1,910,901.94	1,907,151.87	1,710,069.99	1,158,630.29	1,637,351.49	1,560,866.34	1,477,328.63	1,191,087.19
Totals	**2,593,120.63**	**3,927,916.93**	**3,620,646.52**	**3,130,806.73**	**3,092,611.35**	**4,659,550.76**	**4,573,775.90**	**3,833,649.27**	**2,607,847.64**	**4,045,333.74**	**3,831,663.38**	**3,269,195.26**	**2,700,696.85**



sashelp.snacks (35,770 rows, 3meg)

Advertised \ Holiday	0	1	Totals
0	24,723	10,070	**34,793**
1	792	185	**977**
Totals	**25,515**	**10,255**	**35,770**

JAVASCRIPT LIBRARY – DYGRAPHS

There are many graph libraries available which have great features. I like dygraphs for time series graphs in particular as it is powerful, easy to use and free. To illustrate how easy it is to use you can see the code below. The key features to note are:

1) We need the dygraph JavaScript and CSS libraries.

2) We create a div section with an id so that we can create the graph in that element.

3) In the ready function we use the Dygraph function to make our graph. We assign it to a variable, though we don't have to do that. By assigning to a variable it means we can use that variable with the updateOptions function to change options in the graph modifying its appearance.

4) There are 3 parts to the Dygraph call, shown in red, green and blue – all separated by commas:

   a) First we select the element (usually a div) that we want to create our graph in. Document.getElementById is used to get a reference to the div.

b) Next we pick the data source by specifying a URL which will run our stored process example8.

c) Lastly we have an object which specifies parameters we want to use with our graph. We are specifying 3 parameters here:

   i) title creates a title at the top of the graph

   ii) xlabel creates a label on the horizontal x-axis

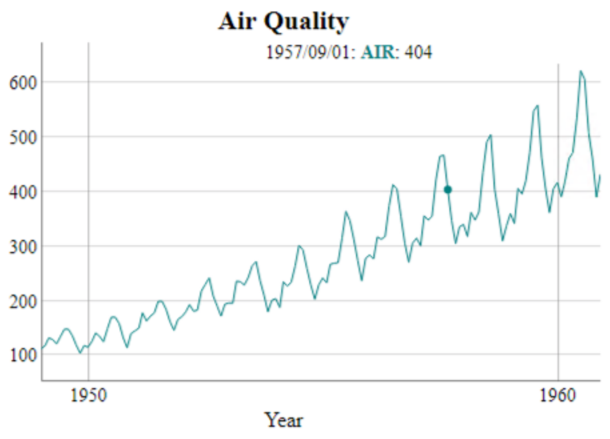
   iii) drawGrid draws a grid on the graph

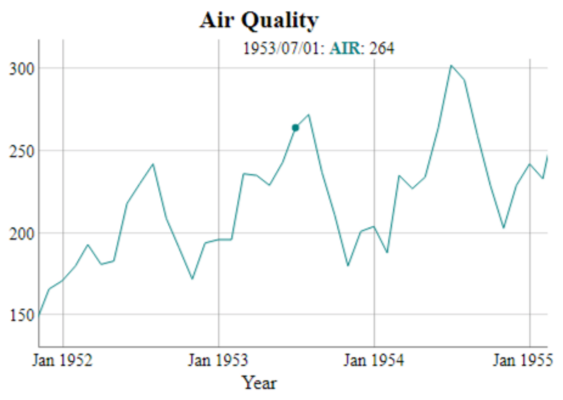
```
<html>
<head>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/dygraph/2.1.0/dygraph.css" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
3.3.1.js"></script>
    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/dygraph/2.1.0/dygraph.js"></scr
ipt>
    <script>
    $(document).ready(function(){
        var g1 = new Dygraph(
            document.getElementById("graph"),

"http://d351tq92/SASStoredProcess/do?_program=/User+Folders/phil/My+Folder/
example8",
            {title:"Air Quality"
            ,xlabel: "Year"
            ,drawGrid: true}
            );
    }) ;
    </script>
</head>
<body>
    <div id="graph"></div>
</body>
</html>
```

The stored process that is providing the data has code that looks like this. Firstly I create a dataset with the date formatted in the form that dygraph can easily interpret. The web site has information about that. Then all I need to do is to use proc export to produce a CSV file. When you stream from proc export you need the replace parameter to be set too.

```
data out;
   set sashelp.air;
   format date yymmdds10.;
run;

proc export data=out outfile=_webout dbms=csv replace ;
run;
```

The graph produced looks like this. If you hover over a point you get info about it. You can also drag the mouse to zoom in on part of the data, as shown in the 2nd image.





JAVASCRIPT LIBRARY – MUZE

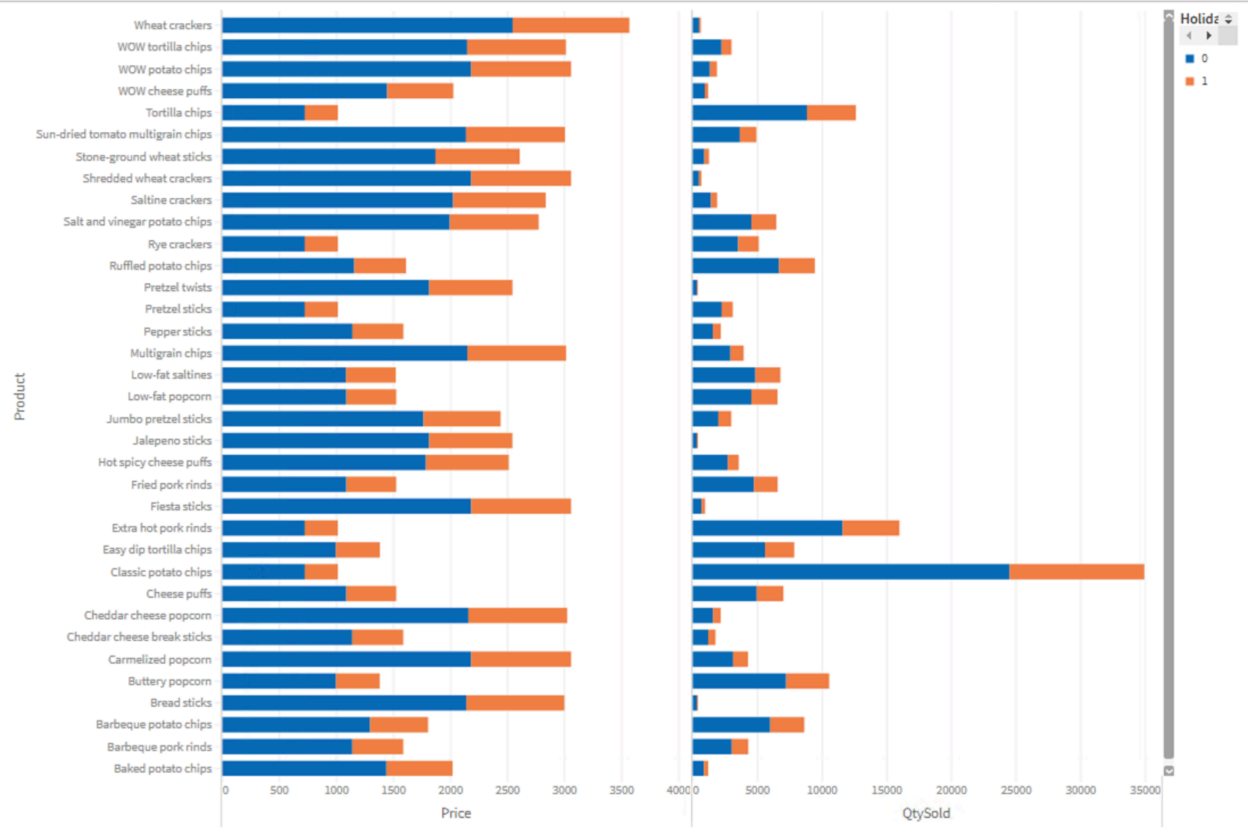
Muze is a useful graph library which produces visualizations in a slightly different way. You start by defining things about your data and then go on to define things about your graphs. Graphs react to changes in data, so if multiple graphs point to the same data, then they will change when data changes. This allows you to build cross filters which are graphs you can click on that then change what you see in other graphs. You can use multiple data formats, stack graphs on top of each other and so on.

Here is a simple example which uses a data source we produced for a prior example, so I wont go over the stored process code again. The HTML for the example follows and you will notice the url parameter pointing to our stored process. In the success function you will see that we do several things:

1. We create a **schema** variable which represents the structure of the data we are using.
2. We create a DataModel variable called **dm** which connects our data to our schema. The first parameter as we create it is the data we are using, and the second is the schema.
3. Next we create a variable called **env** using the muze function. Then we create a variable called canvas to let us draw on a canvas, by using the canvas function on **env**.
4. We define things about our visualization by using the dot notation with the canvas variable. We specify some things like:
    a. Data points to the dm variable, which is the data model we created
    b. Width and height define the dimensions of our graph
    c. Rows and columns defines variable that we will use for rows within our grid of graphs, and columns.
    d. Color defines a variable for which colors will indicate its variable values
    e. Legend defines some things about the legend
    f. Mount defines where the graph should be created by selecting the id tag matching the text specified.

```
<!DOCTYPE html>
<html>
```

```
<head>
    <link
href="https://cdn.charts.com/lib/muze/core/latest/themes/muze.css"
rel="stylesheet">
    <script type="text/javascript"
src="https://cdn.charts.com/lib/muze/core/latest/muze.js"></script>
    <script type="text/javascript" src="js/jquery-3.2.1.min.js"></script>
    <script>
        $(document).ready(function () {
            $.ajax({
                url:
'http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+F
older%2Fexample7',
                success: function (data) {
                    console.log("SUCCESS:");
                    console.log(data);
                    const schema = [{
                        name: "Date",
                        type: "dimension"
                    }, {
                        name: "Advertised",
                        type: "dimension"
                    }, {
                        name: "Holiday",
                        type: "dimension"
                    }, {
                        name: "Product",
                        type: "dimension"
                    }, {
                        name: "Price",
                        type: "measure"
                    }, {
                        name: "QtySold",
                        type: "measure"
                    }];
                    const DataModel = muze.DataModel;
                    const dm = new DataModel(data['sashelp.snacks'],
schema);
                    console.log(dm);

                    const env = muze();
                    const canvas = env.canvas();
                    canvas
                        .data(dm)
                        .width(1200)
                        .height(800)
                        .rows(["Product"])
                        .columns(["Price", "QtySold"], [])
                        .color("Holiday")
                        .legend({
                            align: "horizontal",
                            steps: 6
                        })
                        .mount("#chart-container");
                }, // success
                error: function (data) {
                    console.log("ERROR:");
                } // error
            }); // ajax
        }) // end ready
    </script>
</head>

<body>
    <div id="chart-container">
    </div>
</body>

</html>
```
The graph produced looks like this.



FINAL BITS AND PIECES

JavaScript Library – h54s

I you want to move data between your web browser and SAS then this is a great way to do it. I have used this in some of my work and its great especially the way it can bring down errors and logs from SAS for you to do things with.

JavaScript Library – Cross Filters

You can read all about this in my book! But they are great, and not hard to use.

Building Web Applications

This is beyond the scope of the paper, and covered in my book. However I would suggest adopting a CSS framework such as Bootstrap 4 to provide something to build your web applications with. It will let you layout things on a page say putting

CONCLUSION

JavaScript libraries provide a great deal of pre-packaged functionality which can easily be integrated into web applications you build. Whether you just want a graph or table on a page, or something far more complex and integrated, JavaScript libraries save time and effort giving you top class functionality with little effort. SAS makes it easy to get your SAS data into the form needed for the JavaScript objects, enabling you to build powerful applications with SAS providing data to them.

RECOMMENDED READING

The following web sites are great for learning about the following things:

- Pivot tables … https://pivottable.js.org/examples/
- Dygraphs … http://dygraphs.com
- dataTables … https://datatables.net
- h54s … https://boemskats.com/h54s/ & https://github.com/Boemska/h54s
- muze ... https://github.com/chartshq/muze & https://www.charts.com/muze
- jQuery … https://jquery.com/
- DC … https://dc-js.github.io/dc.js/
- Crossfilters … http://dc-js.github.io/dc.js/ or http://square.github.io/crossfilter/

CONTACT INFORMATION

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

Philip Mason
21-22 High Street, Wallingford, Oxfordshire, OX10 0BP, England
phil@woodstreet.org.uk