My Top 10 ways to use SAS® Stored Processes

Author: Philip Mason

## Abstract

SAS Stored Processes are a powerful facility within SAS. Having recently written a book about SAS Stored Processes I have picked up the 10 best ways to use them so that I can illustrate the different abilities and ways to use them. I will explain how to run almost any code from your web browser, how to use them to export data from SAS to systems like Tableau, how to optimise their use for thousands of users, how to build mobile applications, build visualisations like seen in SAS Visual Analytics, how to make web services to integrate with other clients, and much more. All of these techniques are not generally well known, although they are not complex. If you don’t already understand these techniques, then adding these to your skill set will enable to you to achieve much more.

Table of Contents

[**Abstract** 1](#_Toc503534289)

[1. SAS stored process reports 2](#_Toc503534290)

[2. Run code from web browser 3](#_Toc503534291)

[3. Writing output directly to browser 6](#_Toc503534292)

[4. Returning output to the same page 8](#_Toc503534293)

[5. Export SAS data to other clients 10](#_Toc503534294)

[6. Feeding SAS data to JavaScript objects 11](#_Toc503534295)

[7. Make tables in HTML work properly 13](#_Toc503534296)

[8. Putting together a dashboard 16](#_Toc503534297)

[9. Making web services 21](#_Toc503534298)

[10. Using input streams, output streams and output parameters 24](#_Toc503534299)

[Appendix 28](#_Toc503534300)

[Source code for stored process CROSSFILTER 28](#_Toc503534301)

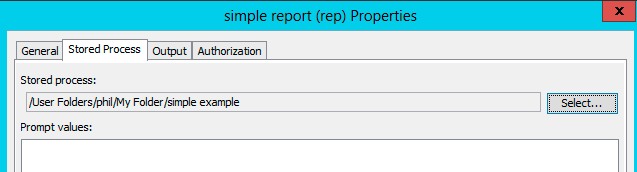
[Source code for stored process EXPORT 29](#_Toc503534302)

[Source code for stored process DataTables 31](#_Toc503534303)

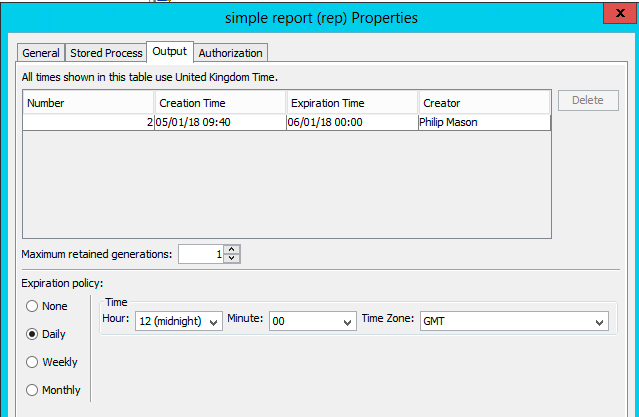
[Source code for stored process DataTables\_DATA 32](#_Toc503534304)

# SAS stored process reports

A SAS stored process report is a cached version of a stored process. You define a stored process report by pointing it at a stored process. When you run the stored process report it will return the results for this stored process. If the stored process has been run within a certain timeframe then those results will come back. If the stored process was run too long ago then it will be run again to update the results, and then those results will come back.



You also define an expiration policy for the stored process which defines how long the stored process results are kept. If the results have expired, then the next time the report is accessed it will be run again to freshen up the results.



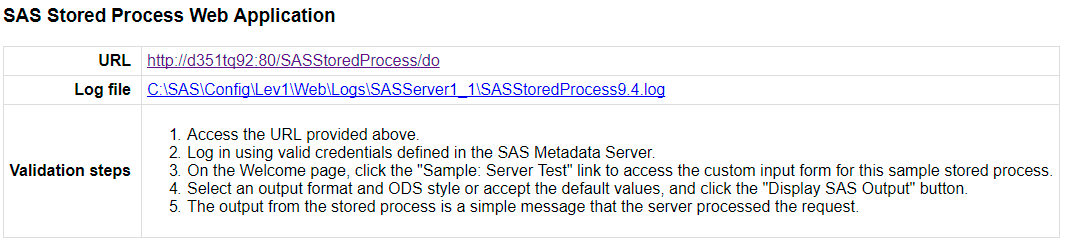
You can also define how many generations of results to keep, so that you can go back and look at previous results.

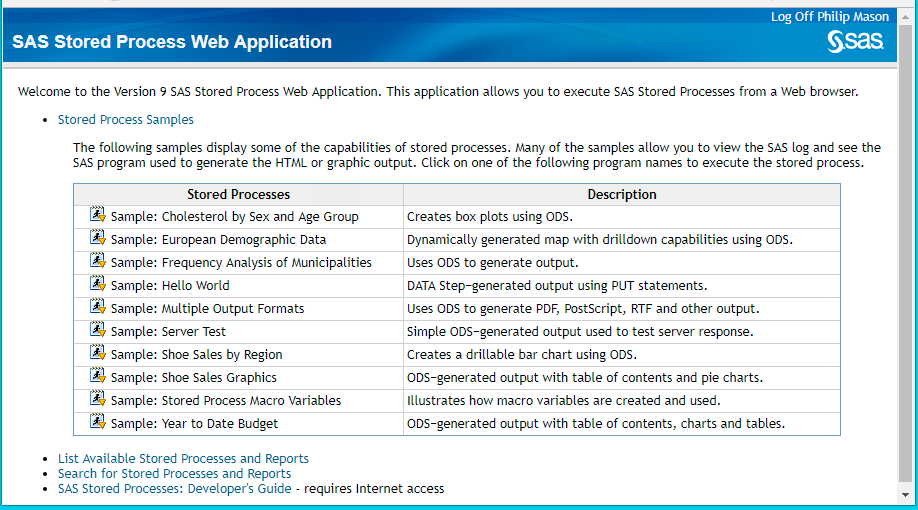
# Run code from web browser

When installing SAS an instructions.html file is created which has all the steps for installing SAS. If you have installed the SAS Stored Process Web Application, then you will find out about it in the list of steps under the SAS Web Applications step. For example, here is what is displayed in my file.

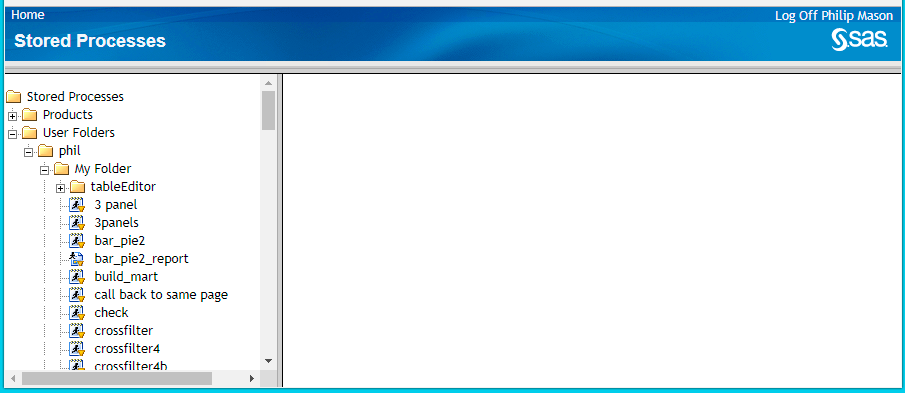


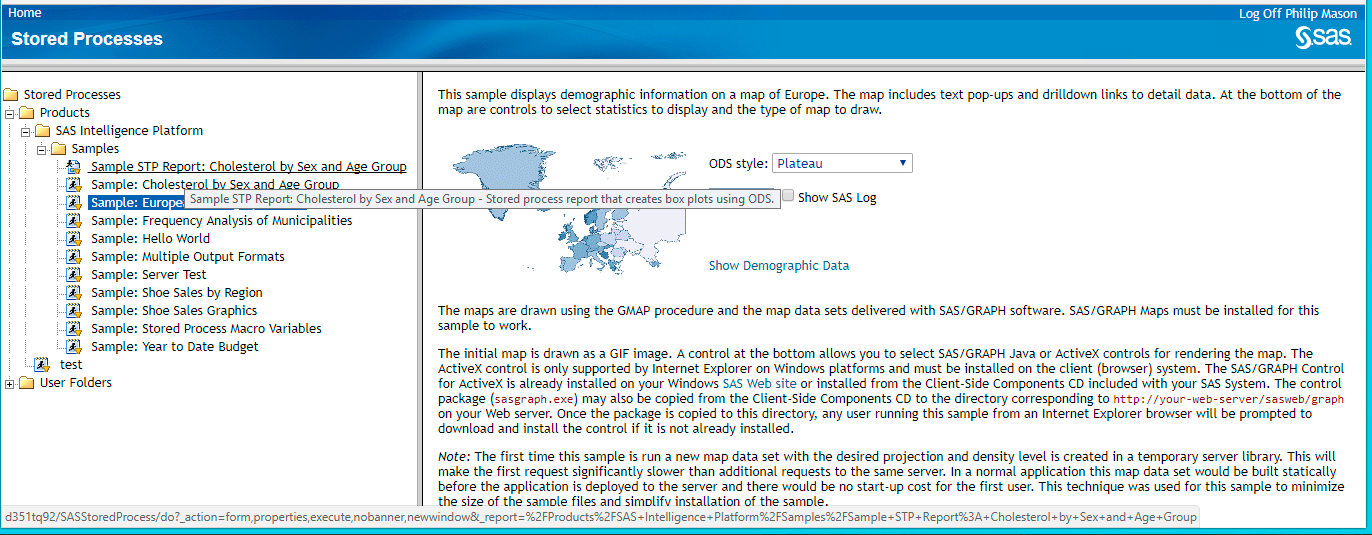
I can then go and find the link for the SAS Stored Process Web Application.



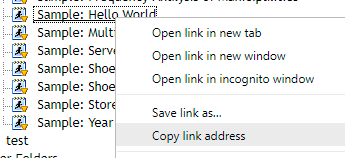
Going to that link I will see the Welcome page for the SAS Stored Process Web Application.  


Clicking on “List Available Stored Processes and Reports” displays the following screen in which you can navigate through metadata folders to find any Stored Processes and Reports you have access to.



Clicking on a stored process in the left panel will run it. If there are parameters to prompt for, then they will be shown in the right panel.  


The results of the stored process might be shown in the right panel or a new tab, depending on the Stored Process.  
You can get the URL to run a Stored Process using the web application by right clicking on one of the stored processes in the left panel and copying the link.

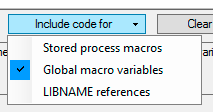


Pasting this in to a web browser will let you run the stored process without going through the general menu.

# Writing output directly to browser

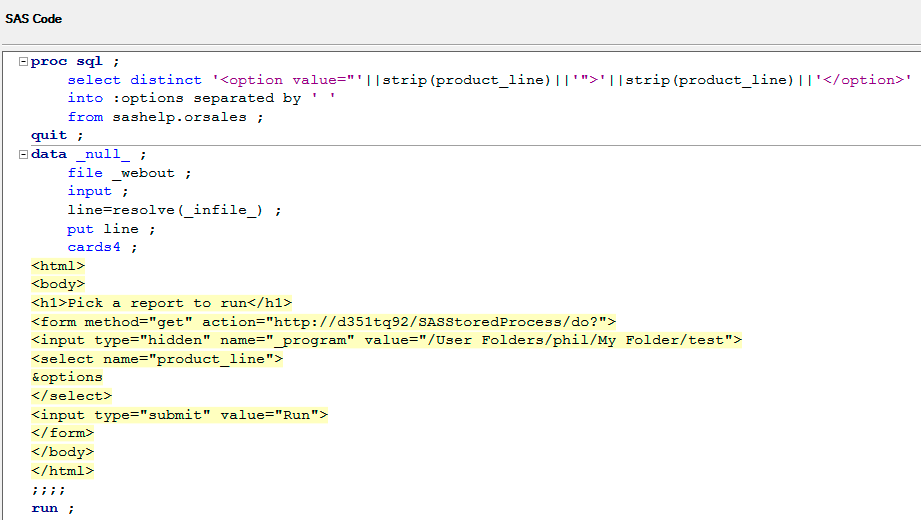
If you create a stored process using the defaults in the wizard from Enterprise Guide, which is the most common way to do so, then it will add in the stpbegin and stpend macro calls. These setup ODS for whatever context you are using the stored process in. However, you can turn these off and take full control of what is returned when the stored process is called.

You turn them off by unchecking the “Stored process macros” item in the SAS Code part of the wizard.



You can return content to the caller of the stored process by writing to the fileref **\_webout**. When calling the stored process using the stored process web application, writing to **\_webout** will write content directly into the web browser. This is how we can write custom HTML from a stored process.

The following code in a stored process will write some HTML to the web browser which creates a menu, allowing the user to make a selection and then call another stored process.



The proc sql creates some HTML tags that will have the values for product\_line so that they can be selected from a list. That data step writes to \_webout, which will write the lines to the browser. The HTML is a simple FORM that lets the user select a value for product\_line and then runs a stored process passing that value to it. Here is the menu displayed.



Here are the results after it has run.



# Returning output to the same page

Usually when using stored processes in the web browser you will run them and be prompted for choices (if there are any) and then the output will be displayed in a new page. To run the stored process again you can go back to the prompts and make another selection, then producing new output on a new page.

It is useful to combine the prompts and the output onto a single page. This is sometimes called a single page application, although these can be far more complex.

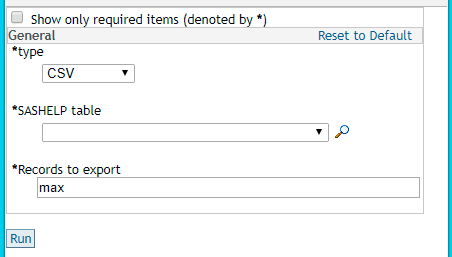
There is a simple way to do this with a stored process. Picking up an example from earlier, we can modify this to prompt us for parameters and then display the output on the same page. We simply add a parameter **target=’content’** to the FORM tag. This means that when the form is submitted the output will be sent to the target which is named content. Then we define an iFrame tag which can accept the content of an HTML page and show it on another page. We specify **name=’content’** in the iFrame and also give it a height and width.



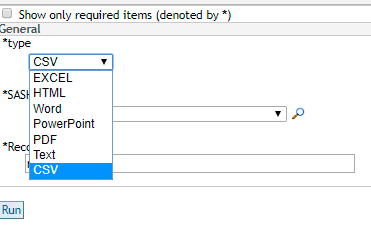
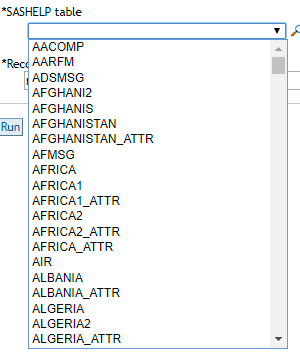
When a selection is made, and this is run, we see the following.  


# Export SAS data to other clients

A great use of stored processes is to provide a smooth way to get data from SAS into some other application or format such as Microsoft EXCEL, Tableau, PDF, etc. It is easy for anyone to run and requires no SAS knowledge or anything to be setup on their machine.

To show how to make a flexible stored process to do this I have made one whose code is included in the appendix. This is the screen that prompts you first.  


You can select the type of file you want to export and the table to be exported. You can also enter the number of records to export or max if you want them all.

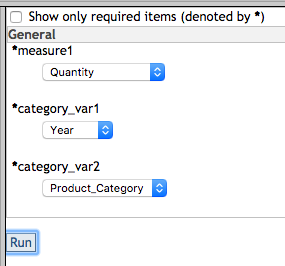
The stored process makes use of several techniques for producing the data in the required format. Some types use ODS and others use PROC EXPORT. It is also very important to set the HTTP headers using the http\_header function. This ensures that the application receiving the data from the stored process understands what type of data it is. For example if the header is set to PowerPoint then the data will be sent to the Microsoft PowerPoint application for opening.

See the source code in the appendix.

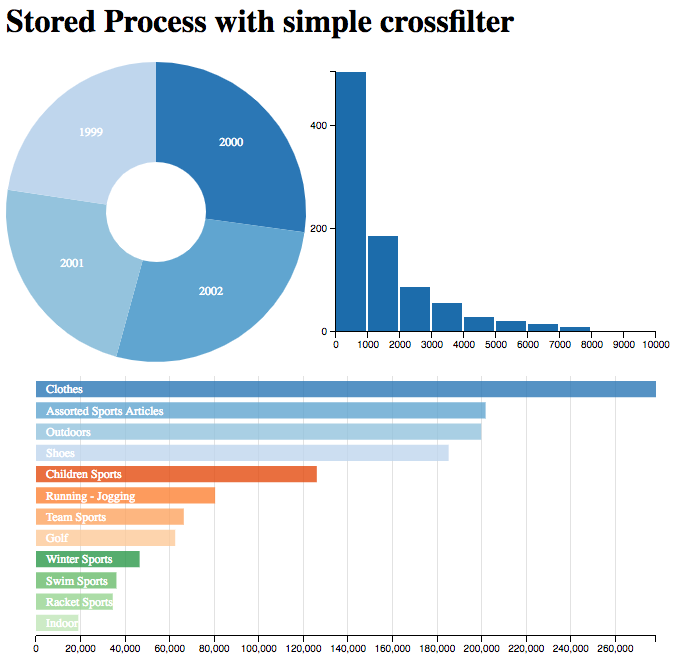
# Feeding SAS data to JavaScript objects

You can make use of many JavaScript objects to make visualizations, such as Crossfilter and D3.

For example, I have made a stored process called crossfilter, which displays the following prompts when run.



This lets you choose some variables from sashelp.orsales to be used to create a visualisation. Selecting these variables makes the following interactive visualisation.







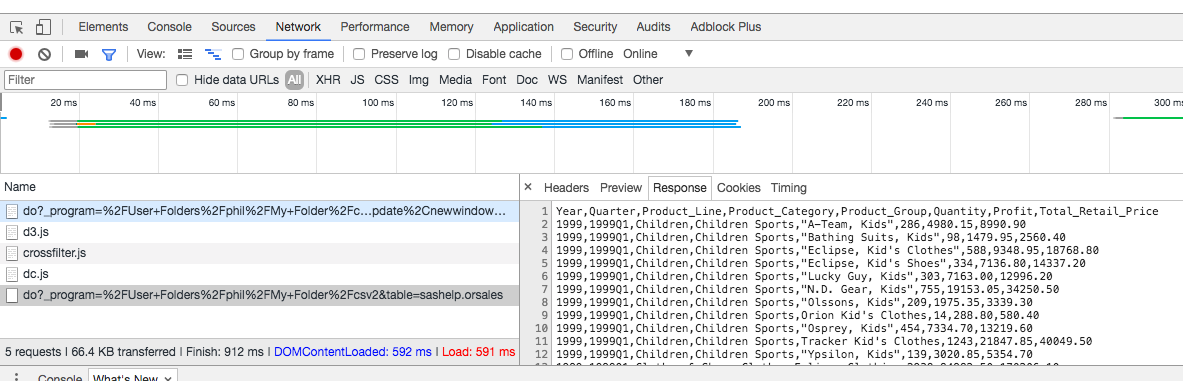
The code used in the stored process csv2, which feeds data to the D3 JavaScript objects has very simple code as follows:

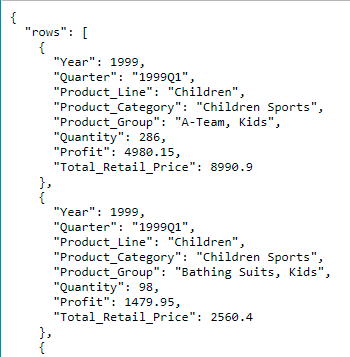
proc export data=&table outfile=\_webout dbms=csv replace ;

run ;

This code just uses PROC EXPORT to send the table passed in as a parameter to the file \_webout in CSV format. \_webout is used to send output to the browser when used with the SAS Stored Process web application, so using it in this way just returns the CSV data to the JavaScript function that calls it. The replace parameter is needed since \_webout already exists and otherwise it won’t work with proc export.

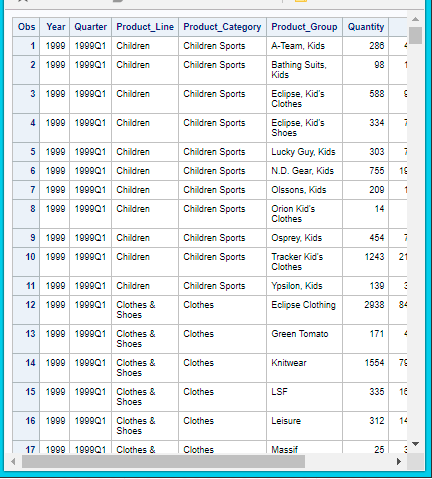
You can use the Developer Tools in your web browser to look at what is going on behind the scenes. Here I am using Chrome, and I can use the Network tab to see what calls are made when running the stored process. I have chosen the stored process that provides the data to the JavaScript function, and I can look at the response to see the CSV data produced by PROC EXPORT.



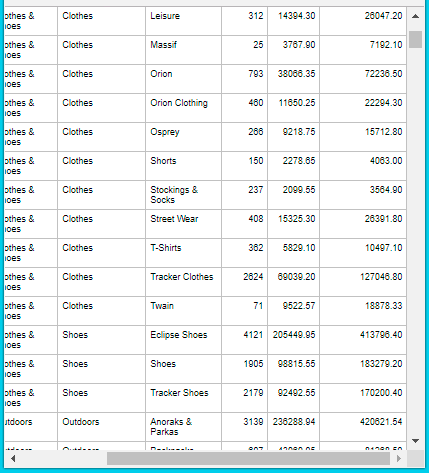
Some other JavaScript objects require different data, such as the grids in jqgrid which need JSON data. Here is some JSON data I use in one of the grids produced by PROC JSON.  


# Make tables in HTML work properly

When viewing a table in HTML produced by a stored process you will typically run into some problems when the table is large. For example just doing a proc print of sashelp.orsales from a stored process would produce this.



If I scroll down and to the right I am left with something that looks like this. However as you can see we no longer have the row numbers displayed as they scrolled off to the left, and the column titles have scrolled off at the top.

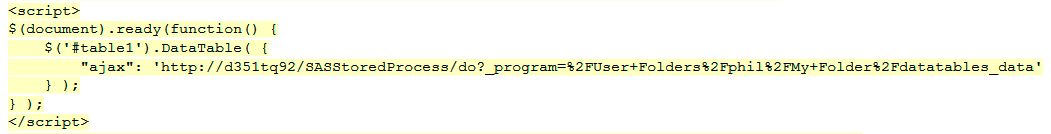


I have heard many a manager over my years complain about SAS tables displayed like this, and insist on them being in EXCEL so that the titles can always be seen, data can be sorted, and so on.

By using a JavaScript object such as DataTables (from <https://datatables.net/examples/index>) we can display a table in a web page and then feed that object with data from another stored process. That enables us to produce something like this.



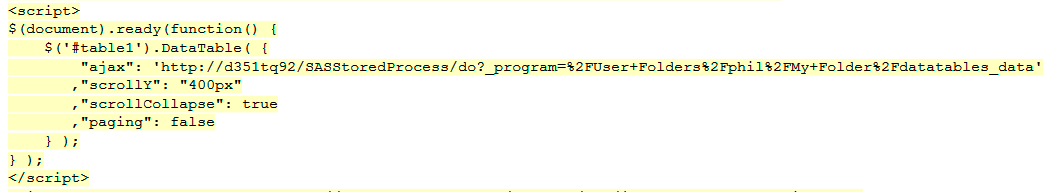
The key JavaScript part that defines the table and where its data comes from is like this.

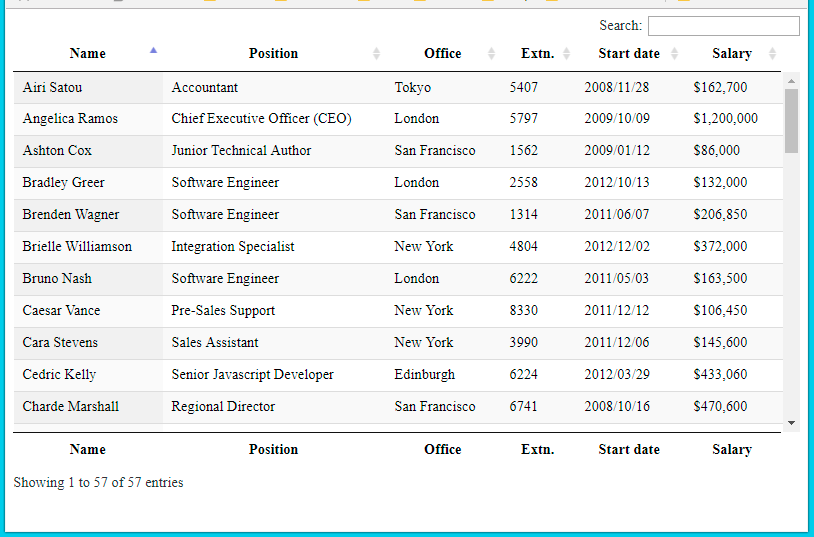


This object just shows what fits on the screen and lets us move through other pages of data. It is produced by a stored process called DataTables (code in appendix). We can also sort by the values in columns. There are many options available with this object so we can actually achieve much of what can be done in EXCEL as far as viewing tables.

The data is provided by a second stored process called DataTables\_data (code in appendix). This data comes from the stored process in JSON format, and is consumed by the JavaScript table. Not only does it provide a better looking table, but it gets past any objections people have about functionality.

By adding a few statements to the JavaScript we can get a scroll bar, rather than using the default paging.



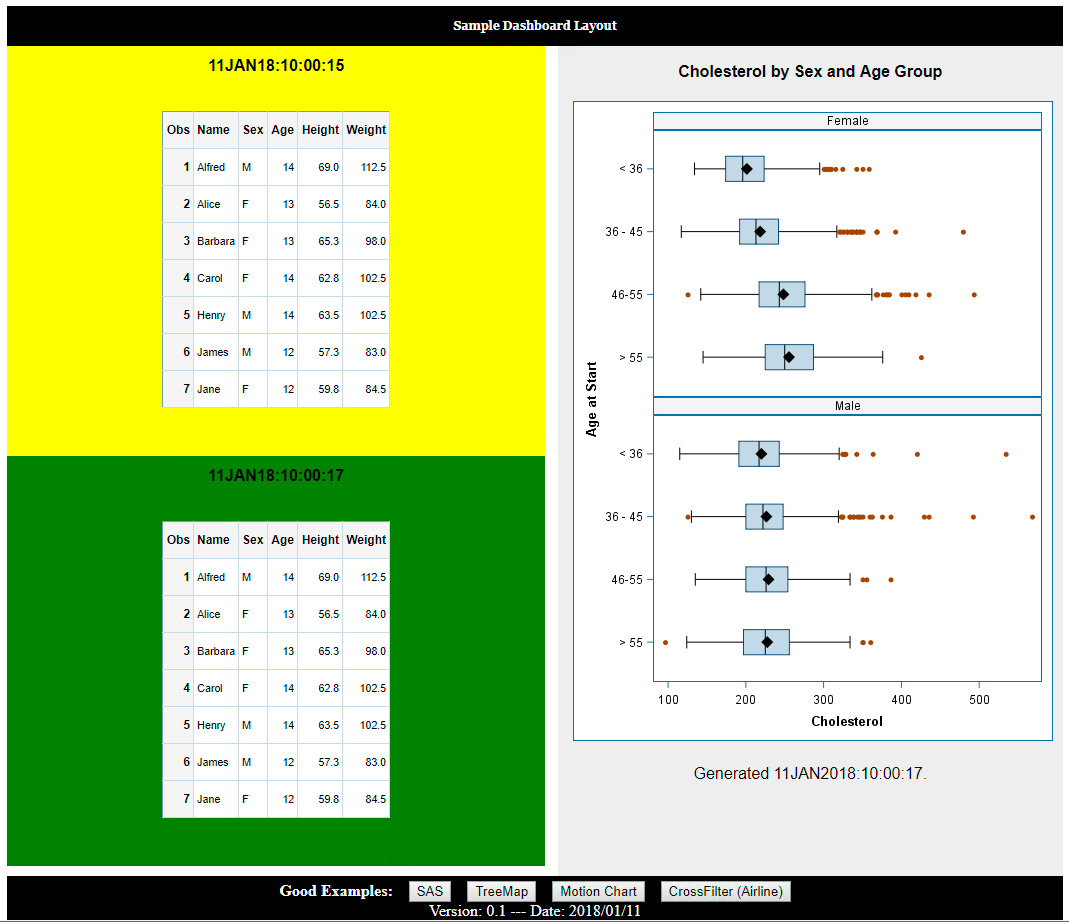
This produces this table, which many people prefer.  


# Putting together a dashboard

There are a number of ways to make a simple dashboard. The key thing about dashboards is that the screen is divided into parts and each part is filled with some information. That information could be a graph, table, indicator, symbol, number, color, etc. You can use your imagination really. To divide the screen up like this you could create a single image using SAS/Graph annotate facility which has all the information in the image. But a better way is to use a web page and populate sections of the page with different content.

We can create parts of a page with stored processes. Each of these parts can be placed on a page by creating a layout with CSS. Each part can be loaded using AJAX, which will load them as required. This gives us lots of flexibility in that each part could contain almost anything, and include interactive functionality or not as required.

The following screen shot shows a simple dashboard layout which is populated by running 3 stored processes.



Here is the code used to create it. The key things to note in the code are highlighted in **bold**.

* We point to the jQuery library, which allows us to use the .load functions to easily load sections of the HTML with the output of our stored processes.
* In the CSS code by using height, float and width we are able to define where our HTML sections are laid out on the screen.
* By using the jQuery .click functions we can define what happens when the pushbuttons are pressed. The .open functions will open a new window with the link.

<!DOCTYPE html>

<meta charset="utf-8">

<head>

**<script type="text/ecmascript" src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.1.min.js"></script>**

<style>

#header {

background-color:black;

font: bold 12px/30px Georgia, serif;

color:white;

text-align:center;

padding:5px;

}

#aside {

**width: 45%;**

**height: 600px;**

background-color:#eee;

**float:right;**

padding:15px;

}

#section1 {

line-height:30px;

background-color:#fff;

**height:400px;**

**width:50%;**

**float:left;**

padding:5px;

}

#section2 {

font: bold 12px/30px Georgia, serif;

line-height:30px;

background-color:#fff;

**height:200px;**

**width:50%;**

**float:left;**

padding:5px;

}

#footer {

background-color:black;

color:white;

clear:both;

text-align:center;

padding:5px;

}

img {

float: left ;

height: 40px;

}

</style>

<script>

$(document).ready(function(){

$("#section1").**load**("http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2FTest+1") ;

$("#section2").**load**("http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2FTest+2") ;

$("#aside").**load**("http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2FTest+3") ;

$('#link1').**click**(function(){

window.**open**('http://support.sas.com/','\_blank') ;

})

$('#link2').**click**(function(){

window.**open**('http://mbostock.github.io/d3/talk/20111018/treemap.html','\_blank') ;

})

$('#link3').**click**(function(){

window.**open**('http://bost.ocks.org/mike/nations/','\_blank');

})

$('#link4').**click**(function(){

window.**open**('http://square.github.io/crossfilter/','\_blank');

})

var d = new Date();

var month = d.getMonth()+1;

var day = d.getDate();

var output = d.getFullYear() + '/' +

((''+month).length<2 ? '0' : '') + month + '/' +

((''+day).length<2 ? '0' : '') + day;

$('#now').text('Version: 0.1 --- Date: ' + output) ;

});

</script>

</head>

<body>

<header>

<div id="header">

<h2>Loading Header</h2>

</div>

</header>

<aside id="aside">

<h2>Loading Aside</h2>

</aside>

<section id="section1">

<h2>Loading Section1</h2>

</section>

<section id="section2">

<h2>Loading Section2</h2>

</section>

<footer id="footer">

<b>Good Examples:&nbsp;&nbsp;&nbsp;</b>

<button id="link1">CrossFilter (Court)</button>&nbsp;&nbsp;&nbsp;

<button id="link2">TreeMap</button>&nbsp;&nbsp;&nbsp;

<button id="link3">Motion Chart</button>&nbsp;&nbsp;&nbsp;

<button id="link4">CrossFilter (Airline)</button>

<div id="now"></div>

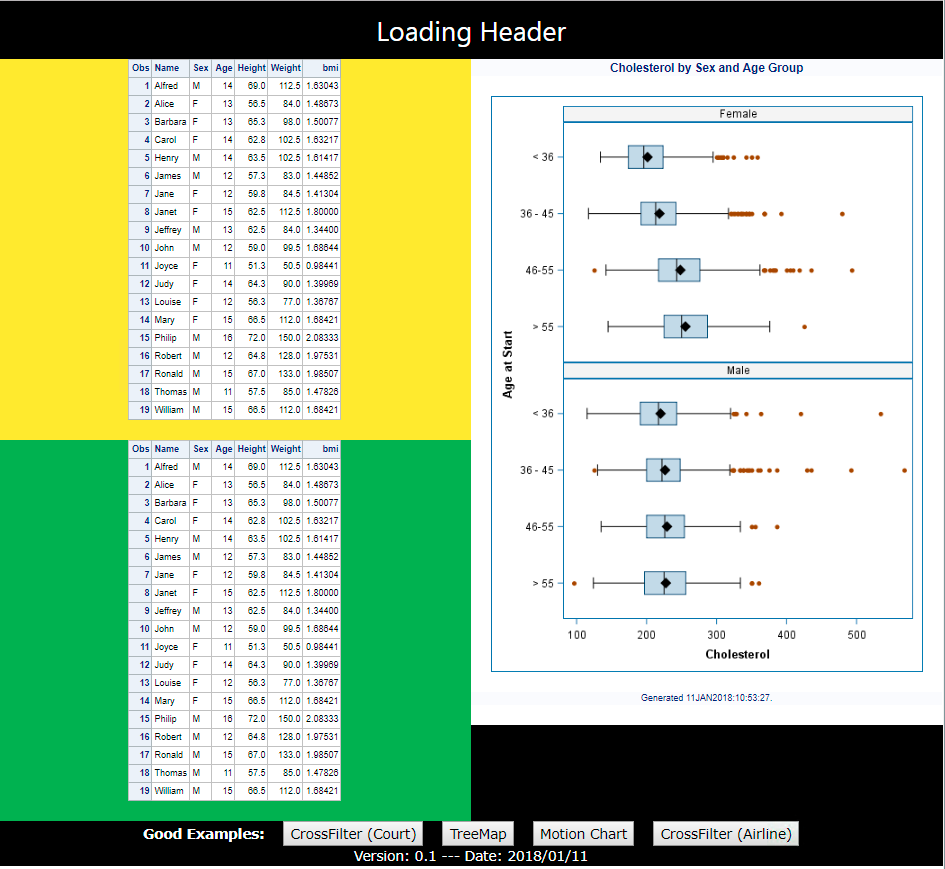
</footer>

</body>

</html>

We could improve this code a bit by using a CSS framework to lay it out. This does a lot of the hard work for you, in the same way that jQuery does the hard JavaScript work for you.

So we can achieve the following output using a CSS framework.



Here is the code, with some important parts bolded. We have been able to simplify the HTML somewhat and replace the CSS directives with some W3.CSS directives contained in class= parameters on HTML tags. We can use the W3.CSS directives to choose background color, alignment of content, how much space sections take up across the page, and much more.

Check <https://www.w3schools.com/w3css/default.asp> for info.

<!DOCTYPE html>

<meta charset="utf-8">

<head>

**<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">**

<script type="text/ecmascript" src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.1.min.js"></script>

<script>

$(document).ready(function(){

$("aside").load("http://my\_server/SASStoredProcess/do?\_program=%2FProducts%2FSAS+Intelligence+Platform%2FSamples%2FSample%3A+Cholesterol+by+Sex+and+Age+Group") ;

$("#section1").load("http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Ftest9") ;

$("#section2").load("http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Ftest9") ;

$('#link1').click(function(){

window.open('http://support.sas.com/','\_blank') ;

})

$('#link2').click(function(){

window.open('http://mbostock.github.io/d3/talk/20111018/treemap.html','\_blank') ;

})

$('#link3').click(function(){

window.open('http://bost.ocks.org/mike/nations/','\_blank');

})

$('#link4').click(function(){

window.open('http://square.github.io/crossfilter/','\_blank');

})

var d = new Date();

var month = d.getMonth()+1;

var day = d.getDate();

var output = d.getFullYear() + '/' +

((''+month).length<2 ? '0' : '') + month + '/' +

((''+day).length<2 ? '0' : '') + day;

$('#now').text('Version: 0.1 --- Date: ' + output) ;

});

</script>

</head>

<body>

<header **class="w3-row-padding w3-black w3-center"**>

<div id="header">

<h2>Loading Header</h2>

</div>

</header>

<aside **class='w3-half w3-white w3-right'**>

<h2>Loading Aside</h2>

</aside>

<section id='section1' **class='w3-half w3-yellow'**>

<h2>Loading Section1</h2>

</section>

<section id='section2' **class='w3-half w3-green'**>

<h2>Loading Section2</h2>

</section>

<footer **class='w3-row-padding w3-black w3-center'**>

<b>Good Examples:&nbsp;&nbsp;&nbsp;</b>

<button id="link1">CrossFilter (Court)</button>&nbsp;&nbsp;&nbsp;

<button id="link2">TreeMap</button>&nbsp;&nbsp;&nbsp;

<button id="link3">Motion Chart</button>&nbsp;&nbsp;&nbsp;

<button id="link4">CrossFilter (Airline)</button>

<div id="now"></div>

</footer>

</body>

</html>

# Making web services

From SAS 9.3 onwards whenever you create a stored process it is also available as a web service. There is a great paper that describes this which can be found here <http://www2.sas.com/proceedings/forum2008/010-2008.pdf>

If I have a URL that I use to call a stored process with the stored process web application then it might look like this. Note: the bold text in the URLs highlights what is different between them.

http://my\_server/**SASStoredProcess/do?\_program=**/User+Folders/phil/My+Folder/webservice1

If I want to use that stored process as a web service then I would use the following URL.

http://my\_server/**SASBIWS/services**/User+Folders/phil/My+Folder/webservice1

If you put **?wsdl** on the end of the URL for the web service then it will return the Web Service Description Language for that web service (stored process) so that it can be used by a client. The URL would look like this.

http://my\_server/**SASBIWS/services**/User+Folders/phil/My+Folder/webservice1**?wsdl**

To use a web service, you can use some code like this, demonstrated on the “Who’s in space, news feed” web service.

filename resp temp;

/\* Neat service from Open Notify project \*/

proc http

url="http://api.open-notify.org/astros.json"

method= "GET"

out=resp;

run;

/\* Assign a JSON library to the HTTP response \*/

libname space JSON fileref=resp;

data \_null\_ ;

infile resp ;

input ;

put \_infile\_ ;

run ;

/\* Print result, dropping automatic ordinal metadata \*/

title "Who is in space right now? (as of &sysdate)";

proc print data=space.people (drop=ordinal:);

run;

To read in JSON data from a web service that you might make can be done easily with the JSON libname engine. Here is some sample JSON data.

{

"name": "Phil Mason",

"home": {

"town": "Wallingford",

"phone": 1491824891,

"current home": true,

"ages": [

51,

46,

18,

16,

13

]

}

}

Here is some code to read it in. Note that a number of tables are mapped to display the JSON data.

\* generate a map to see what is in the JSON, and what datasets are automatically made ;

filename ex 'C:\Users\phil\Documents\My SAS Files\9.4\exsmple JSON.sas' ;

filename jmap temp ;

libname in JSON fileref=ex map=jmap automap=create ;

\* look at the map that was generated ;

data \_null\_ ;

infile jmap ;

input ;

put \_infile\_ ;

run ;

\* look at each of the tables that were mapped ;

title 'in.root' ;

proc print data=in.root ;

run ;

title 'in.home' ;

proc print data=in.home ;

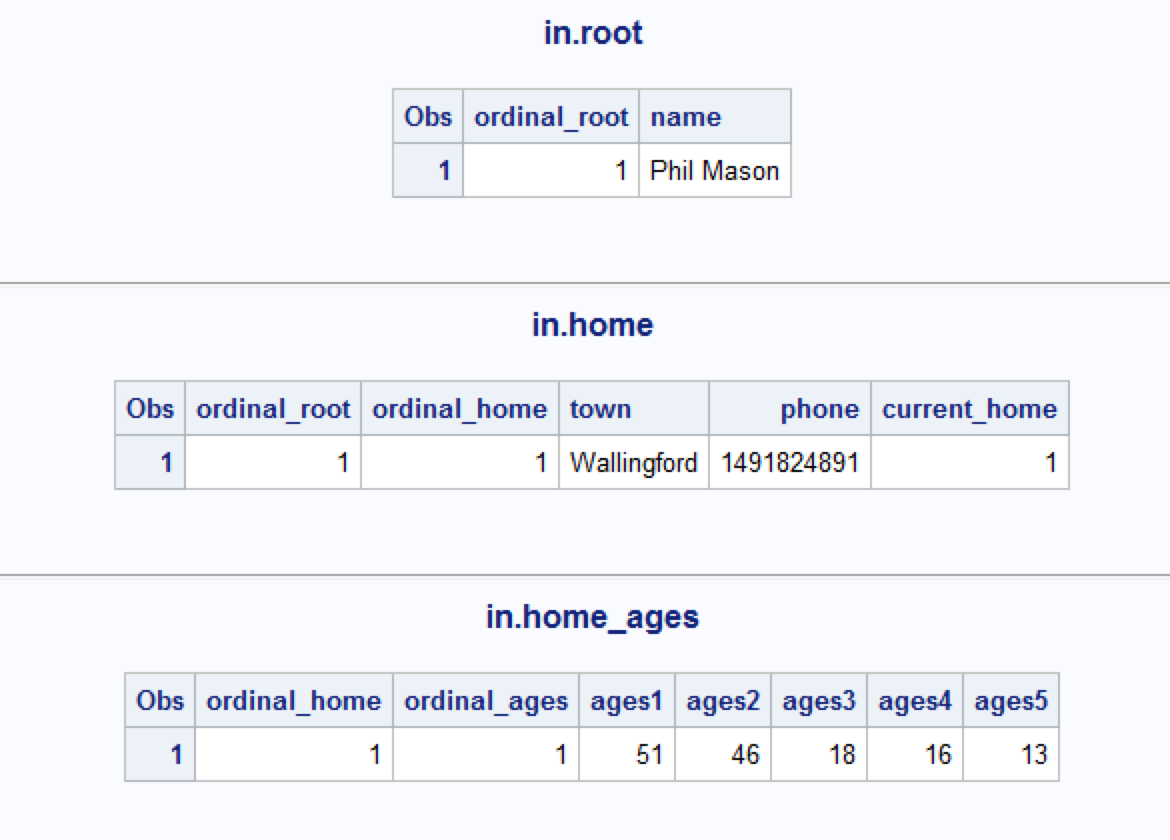
run ;

title 'in.home\_ages' ;

proc print data=in.home\_ages ;

run ;

Produces this output



# Using input streams, output streams and output parameters

If you have the Microsoft Office SAS add-in then you can use stored processes to read data in from EXCEL, process it and write data back out.

Here is the SAS source code I have used to demonstrate this technique.

* My libname statement points to the fileref instr using the XML engine, and it will read the specified data in from EXCEL.
* &\_webin\_sasname is an automatically populated macro variable holding the table name for the EXCEL data I am reading in.
* I set a macro variable call out\_value which I will be able to write into an EXCEL cell, along with 2 other automatic macro variables.
* My Proc Print will produce some ODS output to be written back to EXCEL too.

libname **instr** **xml** ;

**data** input\_from\_excel(drop=i) ;

     set instr.**&\_webin\_sasname** ;

     array numbers(\*) \_numeric\_ ;

     do i=**1** to dim(numbers) ;

           calculated\_total+numbers(i) ;

     end ;

**run**;

\* ODS output ;

title "Data read in from EXCEL, numbers totalled and written back to EXCEL" ;

title2 "Ran at %sysfunc(datetime(),datetime.)" ;

**proc** **print** ;

**run**;

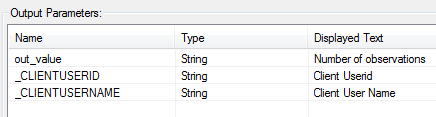
%\* Get the number of observations in the table, and assign to output parameter ;

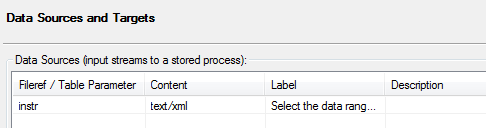
%let dsid=%sysfunc(open(input\_from\_excel)) ;

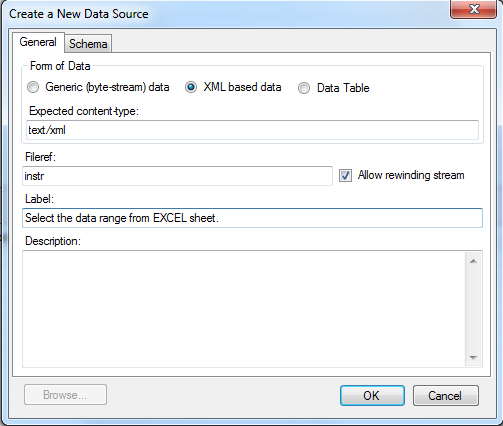
%let **out\_value**=%sysfunc(attrn(&dsid,nobs)) ;

%let dsid=%sysfunc(close(&dsid)) ;

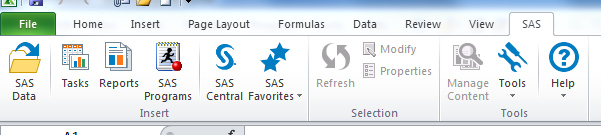
 When defining your stored process you can define some output parameters, which will be able to be written to EXCEL in specific cells.

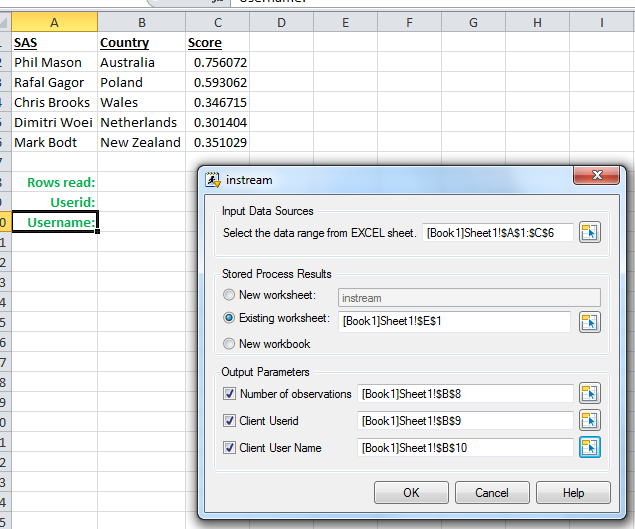


You can define a data source, which you will be able to select a range from an EXCEL sheet for this and then the data in that range will be read in to SAS for use there.  


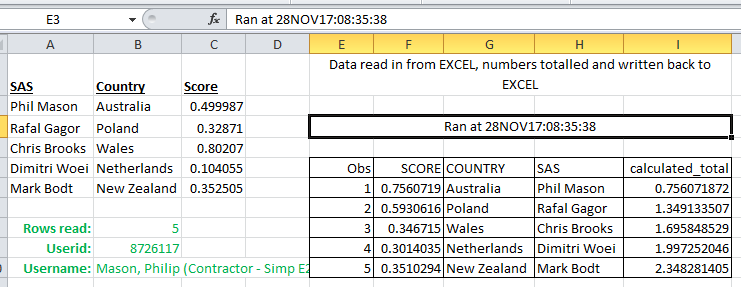
When defining a data source to use from EXCEL you should choose XML based data, give it a fileref (which you will then refer to from the SAS code in your stored process) and tick the box for allow rewinding stream.  


From the SAS add-in toolbar you can click on reports and choose the stored process to run.



The stored process will prompt you to select the input data source. In this example I have made up a table of random values. You also select the destination for the ODS results that the stored process will produce. Finally you can select cells for any output parameters you might like to use.  


 After running the stored process you can see the results produced here.



# Appendix

## Source code for stored process CROSSFILTER

\* Begin EG generated code (do not edit this line);

\*

\* Stored process registered by

\* Enterprise Guide Stored Process Manager V7.1

\*

\* ====================================================================

\* Stored process name: crossfilter

\* ====================================================================

\*

\* Stored process prompt dictionary:

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* CATEGORY\_VAR1

\* Type: Text

\* Label: category\_var1

\* Attr: Visible, Required

\* Default: Year

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* CATEGORY\_VAR2

\* Type: Text

\* Label: category\_var2

\* Attr: Visible, Required

\* Default: Product\_Category

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* MEASURE1

\* Type: Text

\* Label: measure1

\* Attr: Visible, Required

\* Default: Quantity

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*;

\*ProcessBody;

%global CATEGORY\_VAR1

CATEGORY\_VAR2

MEASURE1;

\* End EG generated code (do not edit this line);

/\*%let category\_var1=Year ;\*/

/\*%let category\_var2=Product\_Category ;\*/

/\*%let measure1=Quantity ;\*/

/\*proc stream outfile=\_webout prescol ;\*/

/\*BEGIN\*/

data \_null\_ ;

file \_webout ;

input ;

line=resolve(\_infile\_) ;

put line ;

cards4 ;

<!DOCTYPE html>

<html lang="en">

<head>

<title>dc.js & crossfilter example</title>

<meta charset="UTF-8">

<link rel="stylesheet" type="text/css" href="http://dc-js.github.io/dc.js/css/dc.css"/>

</head>

<body>

<h1>Stored Process with simple crossfilter</h1>

<div id="chart-1"></div>

<div id="chart-2"></div>

<div id="chart-3"></div>

<script type="text/JavaScript" src="http://dc-js.github.io/dc.js/js/d3.js"></script>

<script type="text/JavaScript" src="http://dc-js.github.io/dc.js/js/crossfilter.js"></script>

<script type="text/JavaScript" src="http://dc-js.github.io/dc.js/js/dc.js"></script>

<script type="text/JavaScript">

var Chart1 = dc.pieChart("#chart-1"),

Chart2 = dc.barChart("#chart-2"),

Chart3 = dc.rowChart("#chart-3");

// d3.csv("http://khv-sas-iis2.cfsi.local/SASStoredProcess/do?\_program=/Shared+Data/SASTesting/csv", function(error, spendData) {/\* do stuff \*/});

d3.csv("http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Fcsv2&table=sashelp.orsales", function(spendData) {

// normalize/parse data

spendData.forEach(function(d) {

d.&measure1. = d.&measure1..match(/\d+/);

});

// set crossfilter

var ndx = crossfilter(spendData),

var1Dim = ndx.dimension(function(d) {return +d.&category\_var1;}),

var2Dim = ndx.dimension(function(d) {return Math.floor(d.&measure1./1000);}),

var3Dim = ndx.dimension(function(d) {return d.&category\_var2.;}),

groupVar1 = var1Dim.group().reduceSum(function(d) {return +d.&measure1.;}),

groupVar3 = var3Dim.group().reduceSum(function(d) {return +d.&measure1.;}),

groupVar2 = var2Dim.group().reduceCount() ;

Chart1

.width(300).height(300)

.dimension(var1Dim)

.group(groupVar1)

.innerRadius(50);

Chart2

.width(400).height(300)

.dimension(var2Dim)

.group(groupVar2)

.x(d3.scale.linear().domain([0,10]))

.elasticY(true);

Chart2.xAxis().tickFormat(function(d) {return d\*1000}); // convert back to base unit

Chart2.yAxis().ticks(2);

Chart3

.width(700).height(300)

.dimension(var3Dim)

.group(groupVar3)

.elasticX(true);

dc.renderAll();

});

</script>

</body>

</html>

;;;;

run ;

\* Begin EG generated code (do not edit this line);

;\*';\*";\*/;quit;

\* End EG generated code (do not edit this line);

## Source code for stored process EXPORT

%macro set\_content\_type(type,table) ;

\* type: H = HTML

E = EXCEL

C = CSV

P = PowerPoint

T = Text

W = Word ;

\* tell browser we are sending a file back for a specific application ;

data \_null\_ ;

select(upcase("&type")) ;

when("E") do ;

rc = stpsrv\_header('Content-Type', 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet') ;

suffix = "xlsx" ;

end ;

when("C") do ;

rc = stpsrv\_header('Content-Type', 'application/vnd.ms-excel') ;

suffix = "csv" ;

end ;

when("H") do ;

rc = stpsrv\_header('Content-type', 'text/html%str(;) encoding=utf-8') ;

suffix = "html" ;

end ;

when("W") do ;

rc = stpsrv\_header('Content-Type', 'application/vnd.openxmlformats-officedocument.wordprocessingml.document') ;

/\*application/msword\*/

suffix = "docx" ;

end ;

when("PPT") do ;

rc = stpsrv\_header('Content-Type', 'application/vnd.ms-powerpoint') ;

/\*application/vnd.ms-powerpoint\*/

suffix = "pptx" ;

end ;

when("PDF") do ;

rc = stpsrv\_header('Content-Type', 'application/pdf') ;

suffix = "pdf" ;

end ;

otherwise do ;

rc = stpsrv\_header('Content-type', 'text/plain%str(;) encoding=iso-8859-1') ;

suffix = "txt" ;

end ;

end ;

\* set the filename ;

rc = stpsrv\_header('Content-Disposition', "attachment%str(;)filename=""&table.."||strip(suffix)||'"') ;

run ;

\* output the file ;

%if %upcase(&type)=C %then %do ;

proc export data=sashelp.&table(obs=&obs) dbms=csv outfile=\_webout replace ;

run ;

%end ;

%else %if %upcase(&type)=E %then %do ;

ods excel file=\_webout ;

proc print data=sashelp.&table(obs=&obs) ;

run ;

ods excel close ;

%end ;

%else %if %upcase(&type)=H %then %do ;

ods html5 file=\_webout ;

proc print data=sashelp.&table(obs=&obs) ;

run ;

ods html5 close ;

%end ;

%else %if %upcase(&type)=W %then %do ;

ods rtf file=\_webout ;

proc print data=sashelp.&table(obs=&obs) ;

run ;

ods rtf close ;

%end ;

%else %if %upcase(&type)=PPT %then %do ;

ods powerpoint file=\_webout layout=TitleandContent nogtitle nogfootnote;

title "&table" ;

proc print data=sashelp.&table(obs=&obs) ;

run ;

ods powerpoint close ;

%end ;

%else %if %upcase(&type)=PDF %then %do ;

ods pdf file=\_webout ;

title "&table" ;

proc print data=sashelp.&table(obs=&obs) ;

run ;

ods pdf close ;

%end ;

%else %do ;

ods listing file=\_webout ;

proc print data=sashelp.&table(obs=&obs) ;

run ;

ods listing close ;

%end ;

%mend set\_content\_type ;

%set\_content\_type(&type,&table)

## Source code for stored process DataTables

Lines in **bold** are the extra lines that make the table use scrollbars instead of the default paging.

data \_null\_ ;file \_webout ;input ;put \_infile\_ ;cards4 ;<html><head><script src="http://code.jquery.com/jquery-1.12.4.js"></script><script src="https://cdn.datatables.net/1.10.15/js/jquery.dataTables.min.js"></script><script>$(document).ready(function() { $('#table1').DataTable( { "ajax": 'http://my\_server/SASStoredProcess/do?\_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Fdatatables\_data' **,”scrollY”: “400px”**

**,”scrollCollapse”: true**

**,”paging”: false**

} );} );</script><link rel="stylesheet" href="https://cdn.datatables.net/1.10.15/css/jquery.dataTables.min.css"></head><body><table id="table1" class="display" cellspacing="0" width="100%"> <thead> <tr> <th>Name</th> <th>Position</th> <th>Office</th> <th>Extn.</th> <th>Start date</th> <th>Salary</th> </tr> </thead> <tfoot> <tr> <th>Name</th> <th>Position</th> <th>Office</th> <th>Extn.</th> <th>Start date</th> <th>Salary</th> </tr> </tfoot> </table></body></html>;;;;run ;

## Source code for stored process DataTables\_DATA

data \_null\_ ;file \_webout ;input ;put \_infile\_ ;cards4 ;{ "data": [ [ "Tiger Nixon", "System Architect", "Edinburgh", "5421", "2011/04/25", "$320,800" ], [ "Garrett Winters", "Accountant", "Tokyo", "8422", "2011/07/25", "$170,750" ], [ "Ashton Cox", "Junior Technical Author", "San Francisco", "1562", "2009/01/12", "$86,000" ], [ "Cedric Kelly", "Senior JavaScript Developer", "Edinburgh", "6224", "2012/03/29", "$433,060" ], [ "Airi Satou", "Accountant", "Tokyo", "5407", "2008/11/28", "$162,700" ], [ "Brielle Williamson", "Integration Specialist", "New York", "4804", "2012/12/02", "$372,000" ], [ "Herrod Chandler", "Sales Assistant", "San Francisco", "9608", "2012/08/06", "$137,500" ], [ "Rhona Davidson", "Integration Specialist", "Tokyo", "6200", "2010/10/14", "$327,900" ], [ "Colleen Hurst", "JavaScript Developer", "San Francisco", "2360", "2009/09/15", "$205,500" ], [ "Sonya Frost", "Software Engineer", "Edinburgh", "1667", "2008/12/13", "$103,600" ], [ "Jena Gaines", "Office Manager", "London", "3814", "2008/12/19", "$90,560" ], [ "Quinn Flynn", "Support Lead", "Edinburgh", "9497", "2013/03/03", "$342,000" ], [ "Charde Marshall", "Regional Director", "San Francisco", "6741", "2008/10/16", "$470,600" ], [ "Haley Kennedy", "Senior Marketing Designer", "London", "3597", "2012/12/18", "$313,500" ], [ "Tatyana Fitzpatrick", "Regional Director", "London", "1965", "2010/03/17", "$385,750" ], [ "Michael Silva", "Marketing Designer", "London", "1581", "2012/11/27", "$198,500" ], [ "Paul Byrd", "Chief Financial Officer (CFO)", "New York", "3059", "2010/06/09", "$725,000" ], [ "Gloria Little", "Systems Administrator", "New York", "1721", "2009/04/10", "$237,500" ], [ "Bradley Greer", "Software Engineer", "London", "2558", "2012/10/13", "$132,000" ], [ "Dai Rios", "Personnel Lead", "Edinburgh", "2290", "2012/09/26", "$217,500" ], [ "Jenette Caldwell", "Development Lead", "New York", "1937", "2011/09/03", "$345,000" ], [ "Yuri Berry", "Chief Marketing Officer (CMO)", "New York", "6154", "2009/06/25", "$675,000" ], [ "Caesar Vance", "Pre-Sales Support", "New York", "8330", "2011/12/12", "$106,450" ], [ "Doris Wilder", "Sales Assistant", "Sidney", "3023", "2010/09/20", "$85,600" ], [ "Angelica Ramos", "Chief Executive Officer (CEO)", "London", "5797", "2009/10/09", "$1,200,000" ], [ "Gavin Joyce", "Developer", "Edinburgh", "8822", "2010/12/22", "$92,575" ], [ "Jennifer Chang", "Regional Director", "Singapore", "9239", "2010/11/14", "$357,650" ], [ "Brenden Wagner", "Software Engineer", "San Francisco", "1314", "2011/06/07", "$206,850" ], [ "Fiona Green", "Chief Operating Officer (COO)", "San Francisco", "2947", "2010/03/11", "$850,000" ], [ "Shou Itou", "Regional Marketing", "Tokyo", "8899", "2011/08/14", "$163,000" ], [ "Michelle House", "Integration Specialist", "Sidney", "2769", "2011/06/02", "$95,400" ], [ "Suki Burks", "Developer", "London", "6832", "2009/10/22", "$114,500" ], [ "Prescott Bartlett", "Technical Author", "London", "3606", "2011/05/07", "$145,000" ], [ "Gavin Cortez", "Team Leader", "San Francisco", "2860", "2008/10/26", "$235,500" ], [ "Martena Mccray", "Post-Sales support", "Edinburgh", "8240", "2011/03/09", "$324,050" ], [ "Unity Butler", "Marketing Designer", "San Francisco", "5384", "2009/12/09", "$85,675" ], [ "Howard Hatfield", "Office Manager", "San Francisco", "7031", "2008/12/16", "$164,500" ], [ "Hope Fuentes", "Secretary", "San Francisco", "6318", "2010/02/12", "$109,850" ], [ "Vivian Harrell", "Financial Controller", "San Francisco", "9422", "2009/02/14", "$452,500" ], [ "Timothy Mooney", "Office Manager", "London", "7580", "2008/12/11", "$136,200" ], [ "Jackson Bradshaw", "Director", "New York", "1042", "2008/09/26", "$645,750" ], [ "Olivia Liang", "Support Engineer", "Singapore", "2120", "2011/02/03", "$234,500" ], [ "Bruno Nash", "Software Engineer", "London", "6222", "2011/05/03", "$163,500" ], [ "Sakura Yamamoto", "Support Engineer", "Tokyo", "9383", "2009/08/19", "$139,575" ], [ "Thor Walton", "Developer", "New York", "8327", "2013/08/11", "$98,540" ], [ "Finn Camacho", "Support Engineer", "San Francisco", "2927", "2009/07/07", "$87,500" ], [ "Serge Baldwin", "Data Coordinator", "Singapore", "8352", "2012/04/09", "$138,575" ], [ "Zenaida Frank", "Software Engineer", "New York", "7439", "2010/01/04", "$125,250" ], [ "Zorita Serrano", "Software Engineer", "San Francisco", "4389", "2012/06/01", "$115,000" ], [ "Jennifer Acosta", "Junior JavaScript Developer", "Edinburgh", "3431", "2013/02/01", "$75,650" ], [ "Cara Stevens", "Sales Assistant", "New York", "3990", "2011/12/06", "$145,600" ], [ "Hermione Butler", "Regional Director", "London", "1016", "2011/03/21", "$356,250" ], [ "Lael Greer", "Systems Administrator", "London", "6733", "2009/02/27", "$103,500" ], [ "Jonas Alexander", "Developer", "San Francisco", "8196", "2010/07/14", "$86,500" ], [ "Shad Decker", "Regional Director", "Edinburgh", "6373", "2008/11/13", "$183,000" ], [ "Michael Bruce", "JavaScript Developer", "Singapore", "5384", "2011/06/27", "$183,000" ], [ "Donna Snider", "Customer Support", "New York", "4226", "2011/01/25", "$112,000" ] ]};;;;run ;