Paper 1419-2017

Stored Processes or How to Make You Use SAS® Without Even Knowing It!

Mahmoud Mamlouk, BMO Harris Bank; Edmund Lee, Bank of Montreal

ABSTRACT

Dealing with analysts and managers who do not know how to or want to use SAS® can be quite tricky if everything you are doing uses SAS. This is where stored processes using SAS® Enterprise Guide® comes in handy. Once you know what they want to get out of the code, prompts can be defined in a smart and flexible way to give all users (whether they are SAS or not) full control over the output of the code. The key is having code that requires minimal maintenance and for you to be very flexible so that you can accommodate anything that the user comes up with. This session provides examples of credit risk stress testing where loss forecasting results were presented using different levels. Results were driven by a stored process prompt using a simple DATA step, PROC SQL, and PROC REPORT. This functionality can be used in other industries where data is shown using different levels of granularity.

INTRODUCTION

If you are reading this paper, you are probably looking for ways to allow report users to run code by themselves even when they are not SAS programmers. These reports users could be either data analysts that are not familiar with SAS or high level managers who are not interested in programming. Whatever the reason may be, you can either keep running these reports for them, or build a tool that can allow them to produce these reports with a great deal of flexibility and even empowering them to explore the data.

This is where SAS Stored Procedures (STP) are helpful. An STP is a packaged piece of code that can be accessed from numerous applications, that can be controlled with a carefully designed prompt window and that can output the data in most of the SAS output formats.

There are few simple tricks to configure the SAS code in order to make it controllable via the prompt window, keeping the user in full control of the output.

SAS STORED PROCEDURE IN A NUTSHELL

WHAT IS A SAS STORED PROCEDURE?

A SAS Stored Process is a SAS program that is stored on the server and that can be accessed by multiple applications such as SAS Enterprise Guide, Web applications, and most importantly, Microsoft Excel. The ability for SAS code to be run in Excel, and have the results reported back to Microsoft Excel is a functionality that is very appealing to data users that are not fan of programming and that are not SAS savvy.

One of the main advantages of stored procedures is that the code is made available to every user that has access to the server. This is a very useful feature for codes that generate standard reports.

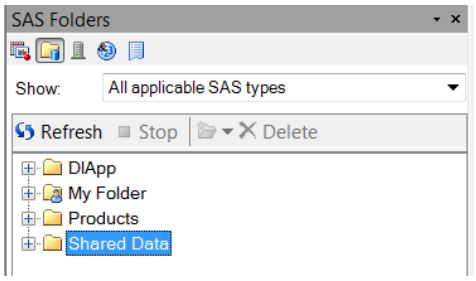
Another advantage of using a stored procedure is that a prompt window can be defined for the users to input all the required parameters to control the underlying code. The user will have no visibility to the code hence a carefully designed stored procedure can give the user full control over the process without actually coding.

HOW CAN YOU ACCESS A STORED PROCEDURE?

A stored procedure can be accessed from multiple applications. The two common vehicles are SAS Enterprise Guide and Microsoft Excel.

From SAS Enterprise Guide (7.1):

Once the stored procedure is registered in the metadata, it can be accessed through the metadata folder explorer:



From Microsoft Excel:

Users who have access to the SAS add-in for Microsoft Office will find that interacting with SAS is very intuitive through the SAS stored procedures.

Similarly to the SAS Enterprise Guide access, the users can navigate to the metadata folder where the stored procedure is registered and run it from there.

GUIDE ON HOW TO MAKE USE OF PROMPTS FOR NON-SAS USERS

One of the advantages of using a SAS Stored Process is to give the user some or full control over the program flow through an interactive prompt window, without needing to write or understand any of the codes used. There are several prompts type allowed for Stored Processes, with the most commonly used ones being Text, Numeric, and Date. Each of these can also be populated with users entering the values manually, or selecting from a set of predefined values.

To begin, we will create an OrionOrders dataset by combining the ORDERS, CUSTOMER_DIM, and PRODUCT_DIM tables. The detail of how this data is created is shown in APPENDIX A

. We also created two new variables PROFIT and PROCESSDAYS defined as follows:

```
Profit = RetailPrice - (Quantity * Cost);

ProcessDays = Delivery_Date - Order_Date;
```

With this data, we can plot the customer demographic by age, customer country, and the average quantity ordered, as shown in **Figure 1**. Each of the colored dots represents one country, and since there are 47 distinct countries, there are too many to be listed in a legend.

```
proc sgplot data=OrionOrders;
    dot Customer Age /
        response=Quantity
        group=Customer_Country
        stat=mean
        markerattrs=(symbol='CircleFilled');
    xaxis label='Average Quantity Per Order';
run;
```

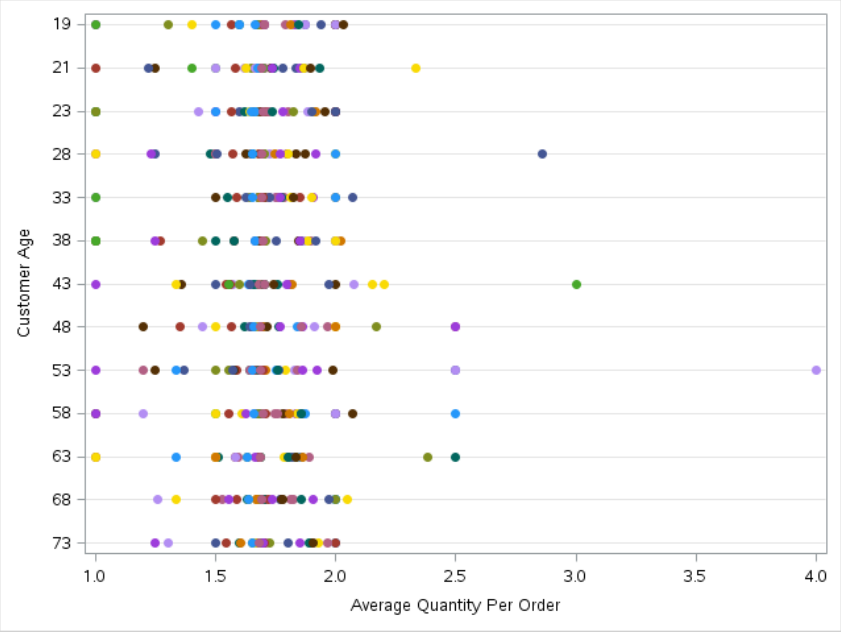


Figure 1: Simple plot of average quantity ordered by customer age and country

In the following paragraphs, we will explore this data in a few different ways, all manipulated by the stored process prompt.

SUBSETTING THE DATA VIA SELECTION IN PROMPT

One simple way to make the plot more information is to subset the country/continent of interest, which would reduce the number of dots. This is an ideal use of prompt since it may not be clear what region is of interest to the user. In the following example, a text prompt that allows users to select multiple predefined continents is used to subset the data, and this creates a macro call `continent_list` (see **Figure 2**).

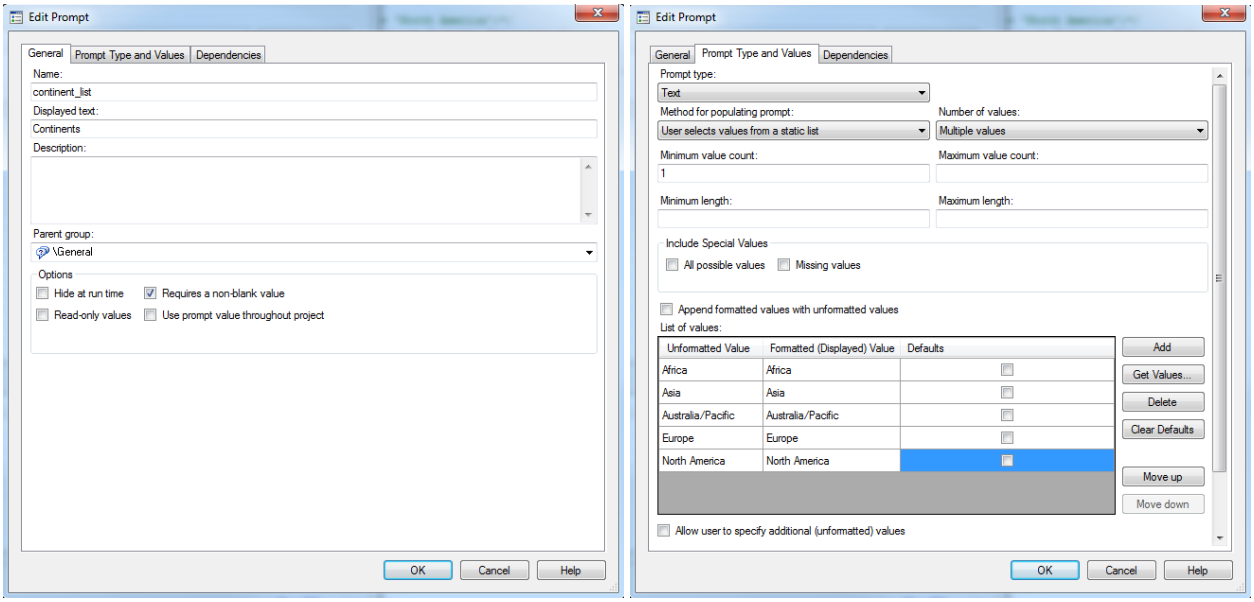


Figure 2: Prompt menu for creating Continent prompt with text selection

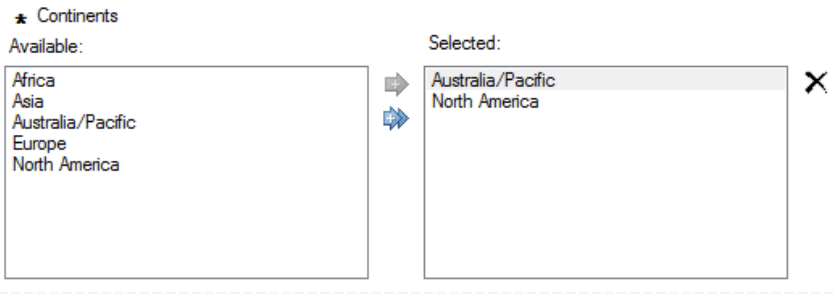


Figure 3: Prompt selection menu as appear to users for selecting Continents

However, the difficulty in allowing the user to select multiple values in the prompt is that different macros are created depending on whether one value is selected, or multiple. In the case that one continent is selected (eg. 'North America'), two macros are created by the prompt, `CONTINENT_LIST` containing the continent selected, and `CONTINENT_LIST_COUNT` which equals 1:

scope	name	offset	value
GLOBAL	CONTINENT_LIST	0	North America
GLOBAL	CONTINENT_LIST_COUNT	0	1

On the other hand, if multiple values are selected (eg. 'North America' and 'Europe'), the following macros are created by the prompt, where `CONTINENT_LIST` now only contains one of the continents selected:

scope	name	offset	value
GLOBAL	CONTINENT_LIST	0	North America
GLOBAL	CONTINENT_LIST0	0	2
GLOBAL	CONTINENT_LIST1	0	North America
GLOBAL	CONTINENT_LIST2	0	Europe
GLOBAL	CONTINENT_LIST_COUNT	0	2

The programmer can create a macro like %ConcatPromptList (shown in Appendix B, based on suggestions from the SAS communities) that returns all the selected values in one macro, separated by an optional delimiter, regardless of the number of values selected. For example, for the prompt above that generates a set of `continent_list` macros, we can create a single `prompt_continents` macro, with a comma as a delimiter between the values, by invoking

```
%let prompt_continents = %ConcatPromptList(prompt=continent_list, dlm=%str(,));
```

which creates `PROMPT_CONTINENTS = North America,Europe.`

With this ability to subset the data, the plot shown in **Figure 1** can be reduced to a plot shown in **Figure 4** when only selecting 'North America' and 'Australia/Pacific'.

```
%let prompt_continents = %ConcatPromptList(prompt=continent_list, dlm=%str(","));

proc sgplot data=OrionOrders;
    dot Customer_Age /
        response=Quantity
        group=Customer_Country
        stat=mean
        markerattrs=(symbol='CircleFilled');
    where Continent in ("&prompt continents");
    xaxis label='Average Quantity Per Order';
run;
```

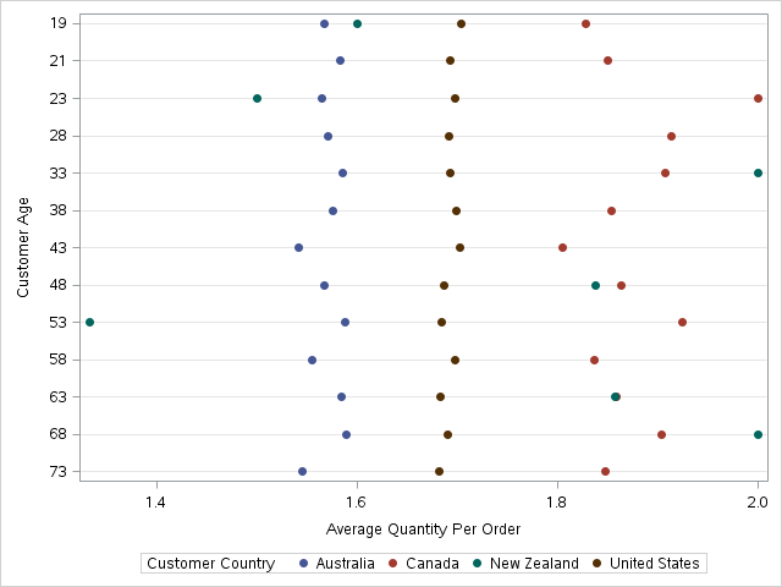


Figure 4: Using prompt selection to subset the continent of interest, reducing the amount of data plotted

Now the user can start to see interesting characteristics of the data. For example, the average quantity per order from the United States is around 1.7 regardless of the customer age. Similarly, the average quantity per order in Australia is around 1.6 with very few fluctuations given the customer age. On the other hand, there is no clear pattern for the Canadian orders, where the average quantity varies given the customer age. Further exploration is necessary to understand the data.

DEFINING DATA BINNING VIA TEXT PROMPT

Although SAS Stored Processes allow users to enter value ranges with a start and end value, for example January 2017 – December 2017, it may not be obvious as to how to enter multiple ranges to be used as bins for grouping continuous data. This may be useful in profiling customer demographic, where instead of looking at the sales for discrete age numbers, the user may want to group the customers into groups (eg. <20, 20-30, 30-45, 45-60, >60).

One way this can be accomplished is by entering the groupings as a text prompt, with a predefined delimiter separating the groups. For example, the four groups above can be entered as "`20:30:45:60`", creating a macro `ageGroup` with the same text.

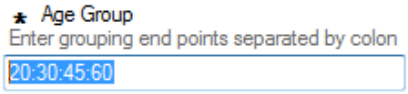


Figure 5: Prompt input menu as appear to users for selecting age bins

This has the added advantage that the number of bins is not fixed, and the user is welcome to create as many or as little bins as needed. To group the data using the macro generated, we can create a separate macro for each group using the `%scan` macro with ":" as a delimiter, and create a new variable in the data that contains the bin name as values through if/else statements or formats. In the example below, a macro `nGroup` is created that stores the number of bins specified, and `ageGroup#` is created for each bin. A new variable `Input_Age_Group` is created based on how the customer's age falls within each bin.

```
%macro CustomGrouping;
%let nGroup = %eval(%sysfunc(countw("&ageGroup",":")));
%do i = 1 %to &nGroup;
    %let ageGroup&i = %scan("&ageGroup", &i, ":");
```

```
%end;

data OrionOrders AgeGroup;
    set OrionOrders;
    format Input Age Group $10.;
    label Input_Age_Group='Age Group';

    if Customer Age <= &ageGroup1 then
        Input Age Group = "<= &ageGroup1";
    else if Customer Age > &&ageGroup&nGroup then
        Input Age Group = "> &&ageGroup&nGroup";
    %do j = 1 %to &nGroup-1;
    else if &&ageGroup&j < Customer Age <= %superq(ageGroup%eval(&j+1)) then
        Input_Age_Group = "&&ageGroup&j-%superq(ageGroup%eval(&j+1))";
    %end;
run;
%mend;
%CustomGrouping
```

The results are shown in the following table:

Customer_Name	Customer_Age	Input_Age_Group
Indu Mele	19	<= 20
Wendy Diab	38	30-45
Ande Mika	38	30-45
Ande Mika	38	30-45
Suzanne Creasy	38	30-45
Gabriele Cola	53	45-60
Beate Schnettler	58	45-60
Beate Schnettler	58	45-60

Using this grouping, we can produce a plot that summarizes the average purchase order quantity for each age group rather than each distinct age. This plot can be seen in **Figure 6**.

```
%let prompt continents = %ConcatPromptList(prompt=continent_list, dlm=%str(","));
proc sgplot data=OrionOrders_AgeGroup;
    dot Input Age Group /
        response=Quantity
        group=Customer_Country
        stat=mean
        markerattrs=(symbol='CircleFilled');
    where Continent in ("&prompt continents");
    xaxis label='Average Quantity Per Order';
run;
```

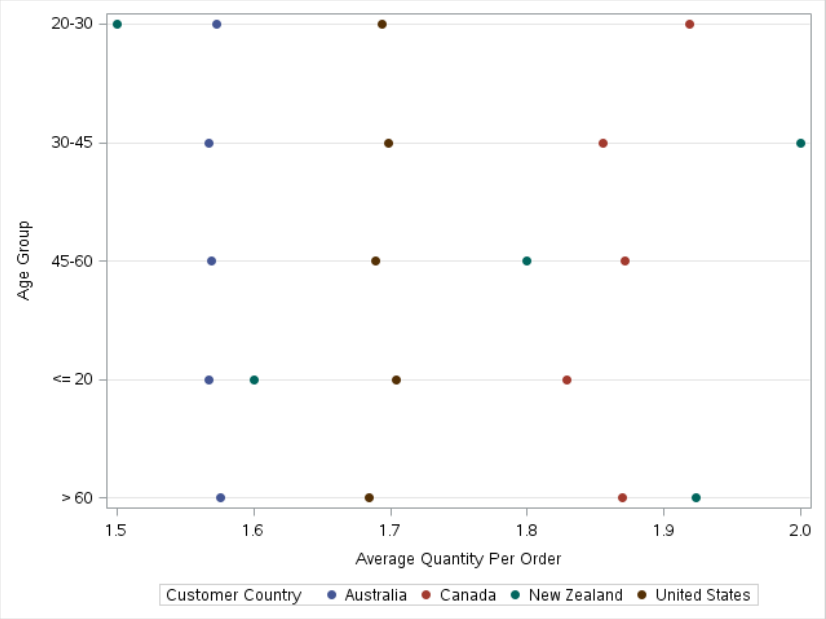


Figure 6: Specifying age bins in prompt to group data for plotting

The user can now start to see cleaner data, grouped by age bins. This helps in clarifying what age group Orion needs to target if it wants to increase the average quantity per order.

PREPOPULATING THE INPUT FIELDS IN THE PROMPT WINDOW

By prepopulating the prompt input field with default values that is most often used, it saves a lot of effort by the users to fill out fields that are often the same between runs. Moreover, a prepopulated field can serves as a guide to users as to how properly fill the input field.

Setting default values is easily done while creating the prompt. Highlighted below in **Figure 7** are two examples of where default values can be set for prompts that require users to manually enter values and for prompts with users selecting values from a predefined list.

Assigning default values are especially important when there are a long list of available prompts for the users to fill. An example of a prompt menu fully populated with default values can be seen in **Figure 8**, allowing users to run without needing to modify any of the inputs.

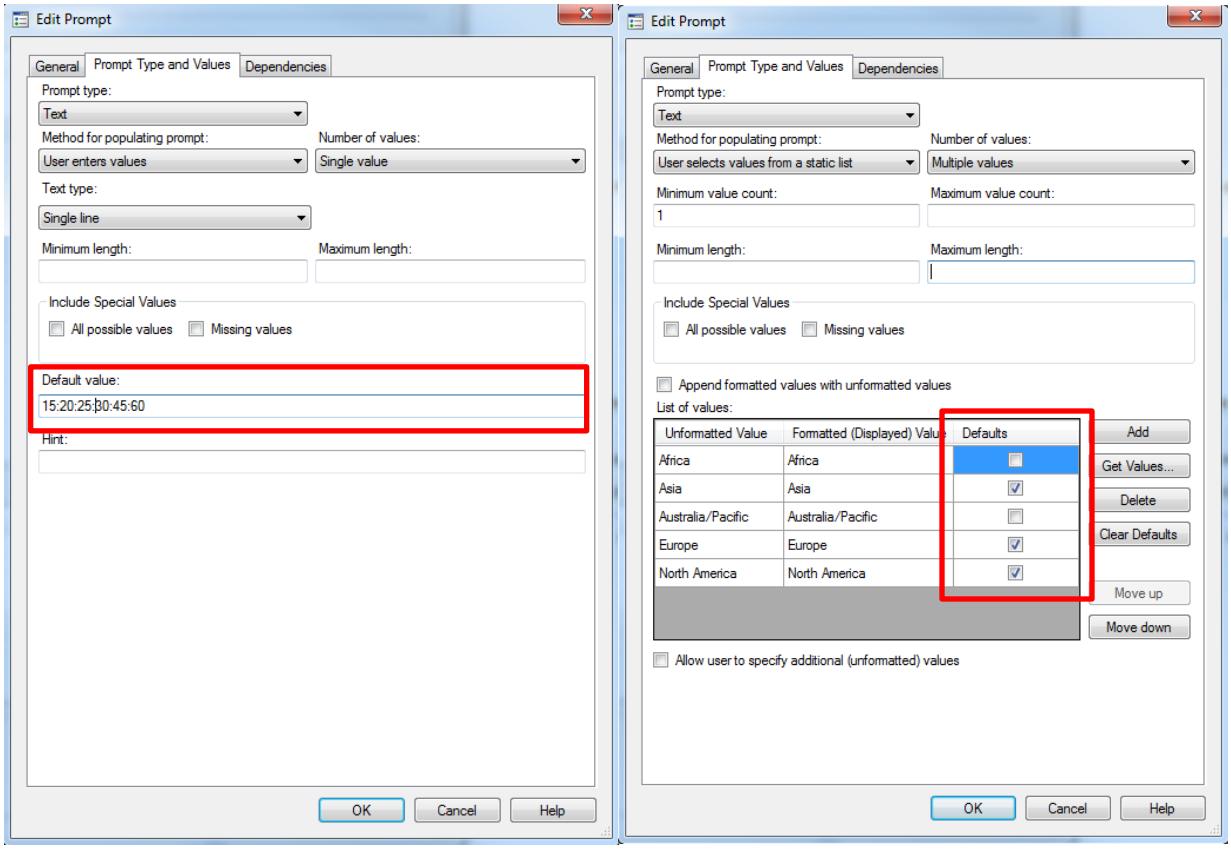


Figure 7: Default values can be set for prompts using the highlighted input areas in Edit Prompt

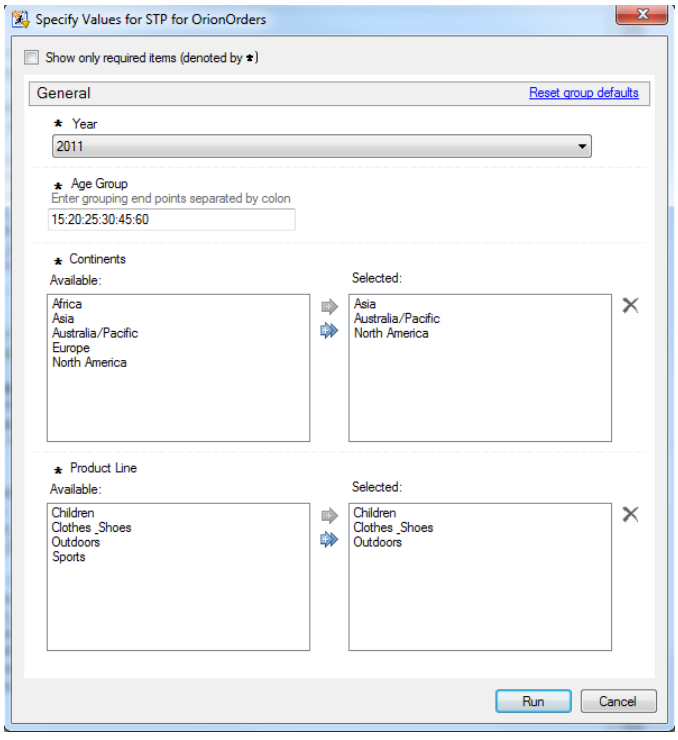


Figure 8: A prompt menu that is fully populated with default values

USER DRIVEN OUTPUT ORDER

Another very informative way to explore the data is to look at a measure on two different dimensions at the same time. The user can look at a heatmap of the average order quantity by country/continent and product line at the same time. This will help to understand the data by further targeting a specific geography and product line or putting an action plan for other combinations that have a low average quantity per order.

The prompts as defined in **Figure 8**: A prompt menu that is fully populated with default values can be a great mean of performing this. The user can select the Continents and the product lines of interest from the pre-populated lists.

There are few ways this cross view can be created, either with a heatmap type of graph or a table in a matrix format.

A heatmap is a great graph to quickly see concentrations of the measure of interest by two different dimensions at the same time. SAS can easily produce these types of graphs using PROC SGPLOT.

```
proc sgplot data=OrionOrders;
    heatmap y=Country x=Product_Category /
        colorResponse=ProcessDays colorStat=mean colorModel=(white blue);
    where year(order_date) = &year_select
        and Continent in ("&prompt_continents")
        and Product_Line in ("&prompt_product");
    title "Order Process Days in &year_select";
run;
```

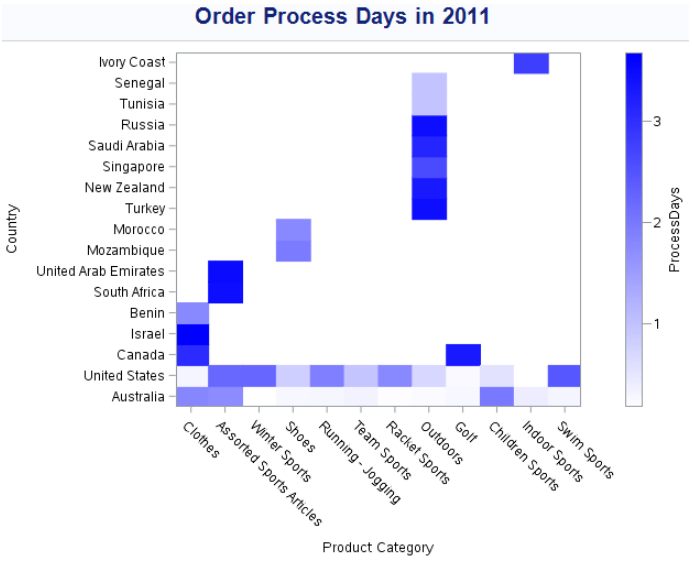


Figure 9: Two dimensional Heat Map

Figure 9: Two dimensional Heat Mapshows few interesting features of the data. It highlights that the product category “Outdoors” has a high average quantity per order in “Russia”, “Saudi Arabia” and “New Zealand”, whereas that measure is in the lower side of the spectrum in “Senegal” and “Tunisia”.

The user can also be interested in seeing the actual numbers in each cell of the heatmap. This can be done using PROC REPORT by producing a table with columns showing the product category and rows showing the countries, as in Figure 10: Two-Dimensional Matrix Report.

```
proc report data=OrionOrders
             (where=(year(order_date) = &year_select
                        and Continent in ("&prompt_continents")
                        and Product_Line in ("&prompt_product")))nofs;
```

```
column  Country (Product_Line);
define Country/ group ;
define scenario_name/ order=data across  'Country' ;
define Product_Line/ across "Product Line";
run;
```

Order Process Days in 2011

	Product Line			
Country	Children	Clothes & Shoes	Outdoors	Sports
Australia	1044	3989	2751	5722
Benin	.	5	.	.
Canada	.	80	.	607
Israel	.	25	.	.
Ivory Coast	.	.	.	6
Morocco	.	5	.	.
Mozambique	.	2	.	.
New Zealand	.	.	15	.
Russia	.	.	8	.
Saudi Arabia	.	.	6	.
Senegal	.	.	2	.
Singapore	.	.	12	.
South Africa	.	.	.	152
Tunisia	.	.	3	.
Turkey	.	.	155	.
United Arab Emirates	.	.	.	11
United States	4453	14065	8140	22424

Figure 10: Two-Dimensional Matrix Report

OUTPUT FORMATS

Throughout this paper, the data was presented either with dot plots, heatmaps or tables. These are only examples of what you can do with SAS to display the data obviously.

One very useful format is the good old fashioned flat file. Instead of pre-defining an output format, the stored procedure can dump the formatted data in Microsoft Excel for example and the user can use that data as needed by creating pivot tables, re-grouping the data by other ways that were not defined in the prompt or creating other graph types.

Another very useful output trick when using a two dimensional table, is to add more than one measure under the column dimension. PROC REPORT is a great way of doing this since the procedure will transpose and format the table without too much effort.

CONCLUSION

Throughout this paper you were presented with various tricks to help the user of your code explore the data without having to be a SAS programmer. Depending on the type reports your users are requesting, you can easily make use of stored procedures to provide them with those reports without having the run the report everytime it is requested. The burden can now be deferred to the user of the report who can generate it with just few clicks.

Data visualization is a crucial tool in times where the data is becoming very hard to manage and explore. Analysts do not necessarily know how they want to look at the data, hence a flexible reporting and graphing tool is very important to make sense of the data.

REFERENCES

SAS Communities. 2012. “STP and multiple-value prompt PROC SQL.” Accessed February 20th, 2017. https://communities.sas.com/t5/SAS-Stored-Processes/STP-and-multiple-value-prompt-PROC-SQL/td-p/97632.

CONTACT INFORMATION

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

Mahmoud Mamlouk
BMO Harris Bank
Mahmoud.Mamlouk@bmo.com

Edmund Lee
BMO Financial Group
Edmund.Lee@bmo.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APPENDIX A

CREATION OF SAMPLE DATASET FROM ORION SALES DATA

The code below creates an OrionOrders dataset from the Orders, Customer_DIM, Product_DIM data sets that SAS typically use for its courses:

```
proc sql;
    create table OrionOrders as
    select distinct
        cust.*,
        ord.*,
        prod.*
    from demo.orders as ord
    left join demo.customer dim as cust
        on ord.Customer ID = cust.Customer_ID
    left join demo.product dim as prod
        on ord.Product ID = prod.Product_ID
    order by Order_Date;
quit;

data OrionOrders(rename=(CustomerCountryLabel=Customer Country
                         SupplierCountryLabel=Supplier_Country
                         CountryLabel=Country));
    set OrionOrders;
    drop Customer ID Customer Country
        Customer FirstName Customer LastName CustomerGenderLabel
        Year Quarter Month MonthName State Code StateName Region
        Product ID Supplier ID Supplier_Country;
    format Profit dollar13.2;

    if StateName ~= '' then State_Region = StateName;
    else State Region = Region;
    Profit = RetailPrice - (Quantity * Cost);
    ProcessDays = Delivery_Date - Order_Date;
run;
```

APPENDIX B

MACRO FOR CONCATINATING MULTIPLE PROMPT SELECTIONS INTO ONE

The following macro will create one single macro that contains all the values selected with a multiple-valued prompt. For example, if the prompt generates a set of `continent_list` macros, we can create a single `prompt_continent` macro, with a comma as a delimiter between the values, by invoking the following:

```
%macro ConcatPromptList(prompt= , dlm=%str( ) ) ;
  %local i return ;
  %if &&&prompt. Count ge 2 %then %do;
    %let return=&&&prompt.1 ;
    %do i = 2 %to &&&prompt. Count ;
      %let return=&return&dlm&&&prompt&i ;
    %end ;
  %end ;
  %else %do ;
    %let return=&&&prompt ;
  %end ;

  &return;
%MEND CONCATPROMPTLIST;

%let prompt continents = %ConcatPromptList(prompt=continent_list,
dlm=%str(,));
```