Paper DG11-2011

Clearing Hurdles in Report Optimization: Custom Sorting Macros, Traffic-lighting Entire Rows, and Forcing SAS Formats into Microsoft Excel®

Jacob Serafino, Mitch Weiler and Asha Jayaprakash, Analytics
Infogroup®, Papillion, NE

Abstract

Comparative reports tend to pop up in almost every analytical context: penetration reports compare client characteristics to the general population; pilot campaigns compare treatment groups to a control group; fiscal reports compare month-to-month sales. With so many reports to run, use of the SAS® macros language is an absolutely critical first step toward optimizing your reporting processes. By adding clarity and flexibility to your code, macros increase efficiency and reduce human error.

Chances are that you've run into some of the same hurdles that we encountered while optimizing Infogroup's reporting solutions. This paper will demonstrate how we overcame these obstacles, featuring macro code that custom-sorts your categorical variables in any user-defined order, highlights any given row in your report, and forces SAS formats into Excel. With the tips and tricks presented here, you'll quickly leave these hurdles far behind and sprint effortlessly into a more streamlined reporting process.

Introduction

Infogroup Analytics creates about a hundred market penetration reports every year. Our custom and standard reports summarize dozens of indicator variables in both tabular and graphical form. Recently, we optimized our reporting process, which has allowed us to dedicate more resources to the development of our increasing array of analytical product offerings.

The biggest candidate for optimization was the part of the code devoted to the generation and output of the actual summary tables. A single, protracted code segment was repeated over and over again with only slight changes to accommodate the idiosyncrasies of each variable. Notwithstanding this excess of code, a great deal of touch-up remained to be done in Excel. Clearly, this was a job for macros!

It was easy to envision several flexible macros that could adapt to the idiosyncrasies of each variable. The redundant, run-on code could be replaced by calling a few quick macros for each variable. Thus, three macros began to take shape: the first generates the summary table; the second exports the table to Excel; and the third creates a graph.

This paper examines the challenges that arose during the development of these macros and the solutions that we discovered. At the end of the paper, the macros will be applied to an example of a realistic reporting situation.

Hurdle 1: Custom-Sorting Macro Code

The first macro was the most important and the most challenging to write. It creates a SAS data set containing the report table, which will eventually resemble Table 1.

The ordering of this table intuitively makes sense, but it not been sorted neither alphabetically, nor in any other way that SAS natively knows how to sort data. The first hurdle that we faced was the need to sort the values of any character variable in a specialized order.

This dilemma typically arises when a character variable has an underlying continuous or ordinal distribution. For example, educational attainment might be encoded as a character variable, and SAS won't see it any other way. However, there is a natural, underlying order to these values that should be reflected in the report, as in Table 1.

Table 1 – Educational Attainment

Educational Attainment	Control Group	Comparison Group	Ratio
High School Diploma	26	32	1.23
Some College	28	12	0.43
Associate Degree	33	31	0.94
Bachelor's Degree	35	37	1.06
Master's Degree	32	32	1.00
Doctorate	22	32	1.45
Total:	**176**	**176**	

We have encountered numerous such character variables that each need to be sorted according to some specialized, human-defined order. The solution in the legacy code was to hard-code the sorting logic for each variable. This redundant and inflexible approach can be avoided by using a macro that generates the sorting code for each variable automatically.

In order to understand what kind of code the macro should generate, it is instructive to examine an example from the hard-coded version. Suppose, for example, that we want to report quantities of barnyard animals of various types. It happens that these animals have been swallowed by an elderly woman. In the report, they will be compared to the quantities allegedly ingurgitated by the proverbial Old Lady Who Swallowed a Fly. Clearly, sorting such a report alphabetically would lead to pure chaos. However, the traditional song provides a natural order for listing the animals in the report table, which can be implemented via the following code.

Code Example 1

```
data ReportTable;
	set ReportTable;

	select (Species);
		when ('Fly') SortAlias = 1; /* Motive for fly-swallowing behavior unknown. */
		when ('Spider') SortAlias = 2; /* Swallowed to catch fly. */
		when ('Bird') SortAlias = 3; /* Absurd, I know. */
		when ('Cat') SortAlias = 4; /* Things go downhill after the cat. */
		when ('Dog') SortAlias = 5; /* Not unheard of in parts of the world. */
		when ('Goat') SortAlias = 6; /* Realized via pharyngeal dilation. */
		when ('Cow') SortAlias = 7; /* An admittedly implausible feat. */
		when ('Horse') SortAlias = 8; /* You know how the story ends... */
		otherwise SortAlias = 9;
	end;
run;

proc sort data=ReportTable; by SortAlias; run;
```

The data step cycles through the table, assigning a new, numerical rank to each row value. Then PROC SORT sorts the rows according to that rank. The ranking variable is called an "alias" because PROC SORT only considers the values assigned to the ranking variable, not the actual row values. Once the sorting is done, the alias variable can be dropped.

In order to create a one-size-fits-all macro, this sorting logic must be generalized to any specialized sorting order. Our macro accomplishes this by taking in a list of row values and reading through it, creating a "when" statement for each value in that list.

A %do %while loop provides the perfect framework for the macro to loop through the list until there are no more values to be read. The loop iterates once for each value, creating the corresponding "when" statement. Then, once it gets to the end of the list, it quits. Here is the loop that we used:

Code Example 2

```
select (&ReportVariable);
	%let SortIndex=1;
	%do %while(%scan(&SortList,&SortIndex,' ',q) NE %str());
		when ( %scan(&SortList,&SortIndex,' ',q) ) SortAlias=&SortIndex;
		%let SortIndex=%eval(&SortIndex+1);
	%end;
	otherwise SortAlias=&SortIndex;
end;
```

The %do %while continues to loop as long as the condition inside the parentheses is true. Inside of the parentheses, we have a %scan function, which returns the value at the current position of the list. (The SortIndex variable serves as a place-holder for the current position in the list.) If the value is null, then we must be at the end of the list; In this case, the condition in the parentheses becomes false, and the loop quits. Otherwise, the code inside of the loop is executed.

The second %scan function, in the body of the loop, returns the value at the current position in the list and feeds it out into the "when" statement. After the loop terminates, the "select" is closed with an "otherwise" statement, which assigns all values not in the list to the lowest position in the table.

For a complete view of the macro that generates our report table, have a look at the appendix. To generate the sorting code in Example 1, the report macro can be called using the following parameters.

```
%CreateReportTable( ProverbialOldLady, ExperimentalOldLady, AnimalType,
       SortList=%str('Fly' 'Spider' 'Bird' 'Cat' 'Dog' 'Goat' 'Cow' 'Horse'));
```

The macro invoked above incorporates the sorting code from Example 2, resulting in the . For a view of the see in our macro (see Appendix), we can generate reports on *any* variable that needs custom-sorted, and all that is required is an ordered sorting list. This allows us to leave our first hurdle behind, and move ahead toward the next two hurdles.

Hurdle 2: Traffic-Lighting any Entire Row

The next two hurdles involve the macro that outputs the report table into an Excel file. We accomplish this by setting the ODS output destination to HTML with a filename extension of XLS, and then running a PROC REPORT.

The second hurdle was the need to highlight rows whose key statistic is within a certain range. This is a common practice called traffic lighting. For example, we could "traffic-light" every row where the value in the "ratio" column is greater than or equal to 1.10, as in Table 2.

Traffic-lighting is well-documented. (See Recommended Reading.) However, the literature focuses on traffic-lighting based on *computed* values, i.e. values that are calculated on the fly, within the PROC REPORT itself. When the key statistic has already been calculated (as it was, by our first macro), these papers leave unanswered questions.

Table 2

Location Sales	Control Group	Comparison Group	Ratio
$0 - $15	12	6	0.50
$16 - $50	24	21	0.88
$51 - $100	**29**	**32**	**1.10**
$101 - $250	107	115	1.07
$251 - $475	159	174	1.09
$476 - $750	199	194	0.97
$751 - $875	98	85	0.87
$876 - $950	**53**	**60**	**1.13**
$951+	39	33	0.85
Total:	**720**	**720**	

For purposes of our macros, this was a serious limitation, because our macros need to be modular. In other words, each needs to perform a few tasks, independently of the others. For example, we might sometimes want to run only the macro that creates the report graph, without first running the macro that generates Excel output. For this reason, we want to calculate the key statistic, along with the rest of the report table, in our first macro. That way, if we want the graph without the Excel report, all of the statistics will already be there, and only the graph macro will need to be run.

Admittedly, traffic-lighting should be a simple enough task, whether the statistics are calculated beforehand or ad-hoc in the PROC REPORT. Because the challenge involved in this task is largely due to the complexity of PROC REPORT, it will be instructive to examine the structure of PROC REPORT in a bit more detail.

After invoking PROC REPORT, a COLUMN statement lists the columns in the order in which they will appear. Then a DEFINE statement for each column describes the features of how the column will be displayed. The two other components of PROC REPORT that we incorporated into our macro are the COMPUTE block and the BREAK block. The BREAK block is beyond the scope of this paper, but all of the traffic-lighting magic happens inside of the DEFINE statement and COMPUTE block.

As mentioned previously, traffic-lighting is typically done on values calculated on the fly within the PROC REPORT. When this is done, one of the columns must be defined as "computed" in the DEFINE statement, and then the calculations must be done within a COMPUTE block. It is there, within the COMPUTE block, that highlighting is done, based on an "if" statement that checks to see whether the value is within the critical range. If it is, then a CALL DEFINE function changes the row's background color and makes the text bold.

At this point, it should be sufficiently apparent that PROC REPORT can be complicated. Luckily, it only requires a few simple steps to traffic-light rows based on pre-computed values. First, in the define statement, the column containing the statistic of interest must be defined as "display" instead of "computed." In Code Example 3, below, the statistic of interest is "ratio."

Code Example 3

```
proc report data=ReportTable nofs headline headskip split='*';
      column &ReportVariable CountControl count ratio;
      define &ReportVariable /display "&VariableLabel";
      define CountControl /sum format=comma12. 'Control*Group';
      define count /sum format=comma10. 'Comparison*Group';
      define ratio /display format=6.2 'Ratio';
```

Next, although the column merely displays residual values and doesn't require any computation, we run the COMPUTE block shown in Code Example 4. Rather than actually computing the column's values, this COMPUTE block simply checks whether the ratio statistic is in the critical range, and then highlights the row if it is.

Code Example 4

```
compute ratio;
      if ratio >= &ThreshHold then do;
      call define(_row_, "style", style=[background=CX93CDDD font_Weight=bold]);
      end;
endcomp;
```

That's all there is to the simple, but counter-intuitive trick of traffic-lighting rows based on already-computed values. Simply define the row as display-type, but run a compute block anyway, including a call define function that follows the syntax shown above.

To reproduce Excel colors, find the RGB (red-green-blue) values in Excel and put them into an online RGB to hex calculator. There is a plethora of such calculators, which can be easily found by searching for "RGB to hex converter." The calculator will give you a six-digit hex code. Simply add "CX" added to the beginning of the hex code, and then it can be used in SAS to reproduce exactly the color selected in Excel.

Hurdle 3: Forcing SAS formats into Excel

The final hurdle results from Excel's quirky formatting habits. For example, Excel insists on dropping trailing zeros from decimal numbers and on dropping leading zeros from *all* numbers. Moreover, it seems to have an insatiable proclivity for scientific notation, triggered by exposure to "long" numbers of more than 11 digits or so.

Excel's most egregiously bad manners surfaced in our reporting process when we were dealing with binned numeric variables. It turns out that Excel consistently misinterprets dash-separated numbers as dates. Consequently, *1 - 5* becomes *5 - Jan*. Excel must think itself very clever to be able to spot these dates in the wild. But if we don't want to hand-correct every Excel report we run, we'll have to be even cleverer than Excel, and somehow force it to accept the formats that SAS outputs the data in.

Several papers have been written on workarounds for this problem. For the most part, they involve learning the Excel tag sets and then inserting the appropriate tags into parts of the report. However, figuring out which are the "appropriate tags" can be unduly complicated. In some cases, it may even require you to "reverse-engineer" Excel files. Then it still remains to determine how to insert those tags into the right part of your PROC REPORT.

The solution that we arrived at may seem anticlimactic, but it can save a lot of headache. It turns out that one short line of code can solve the whole problem. Specifically, the following option must be added to the ODS statement:

Code Example 5

```
headtext='<Style>td {mso-number-format:\@}</Style>'
```

This cryptic little piece of code puts a style tag at the beginning of the Excel file, which tells Excel to interpret the entire report as text. The "mso-number-format" portion of the tag tells Excel that the style that needs applied is a Microsoft Office format, and the "\@" bit is the code for the text format. "TD" stands for *Table Data*, and tells Excel exactly which part of the report to apply the style to. Of course every cell in the table contains "table data," so the style will be applied to every cell.

This works for every version of Excel since Excel 2000. By telling Excel to read everything as text, it solves all of the above-mentioned formatting issues, because Excel assumes that text doesn't need any special formats. This is a

great assumption for our purposes, because we prefer to do our formatting in SAS and keep Excel's creativity out of it.

It is worth noting that Excel gets its feelings hurt when it realizes we expect it to leave the formats alone. When you open the ODS output in Excel, Little green triangles will appear in the corners of each offending cell, indicating that an error has occurred. If annoys you, select the area of the table that contains the green arrows. A tiny, diamond-shaped caution sign will appear. Click the sign to reveal a menu, and then choose "ignore error" from the menu. Voila! Problem solved. Now you have a clean, snappy report that Excel displays properly, with all of the right formats.

Putting it all together

As an example of a realistic reporting situation, we will consider data from the SASHELP dataset SASHELP.HEART, which contains data on the participants of the Framingham Heart Study. For the sake of demonstration, we will compare the smoking habits of men and women, treating women as the reference group and men as the comparison group. First, the data needs placed into two data sets, so that our macros can compare the distribution of the smoking variable between the two data sets.

```
data Males Females;
       set sashelp.heart;

       if sex='Male' then output Males;
              else if sex='Female' then output Females;
run;
```

We will also need to set up an output path, where the macros will place the Excel report and the bar chart file.

```
%let ReportPath=C:\Jacob\Smoking Report;
```

It is a good practice to set up the sort list before invoking the macros, because the list is often too long to include in a single-line macro call. Sometimes SAS can get confused by spaces, parentheses, or other special characters that we may want in our list. So we use the %str macro function to “mask” everything in the list, so that SAS won’t even see the special characters.

```
%let SmokingSortList=%str('Non-smoker' 'Light (1-5)' 'Moderate (6-15)' 'Heavy (16-25)'
       'Very Heavy (> 25)');
```

Finally, once our macro variables have been set up, it’s time to call the macros.

```
%CreateReportTable(Females, Males, Smoking_Status, SortList=&SmokingSortList);
```

The below SAS data set (Table 3) is the result of running this macro. The “ratio” field is the ratio of number of individuals in the comparison group (Men) to the number in the reference group (Women).

Table 3

	Smoking Status	Frequency Count	Percent of Total Frequency	Frequency Count	Ratio
1	Non-smoker	1682	35.059931507	819	0.4869203329
2	Light (1-5)	422	6.720890411	157	0.3720379147
3	Moderate (6-15)	340	10.102739726	236	0.6941176471
4	Heavy (16-25)	339	30.265410959	707	2.0855457227
5	Very Heavy (> 25)	73	17.037671233	398	5.4520547945
6	Unknown	17	0.8133561644	19	1.1176470588

Table 3 contains all of the right data, sorted in the correct order, but it's far from beautiful. However, this table is not meant to be presented; it merely serves as a data source for the next two macros. Now that the report table has been created, we can run either the OutputExcelReport macro or the OutputReportGraph macro.

Of course after all of the time we've spent discussing our Excel report macro, you must be dying to see it in action. So, without further ado, here is the macro call, along with the Excel table that results:

```
%OutputExcelReport(Smoking_Status, Smoking Status, &ReportPath\Smokers.xls);
```

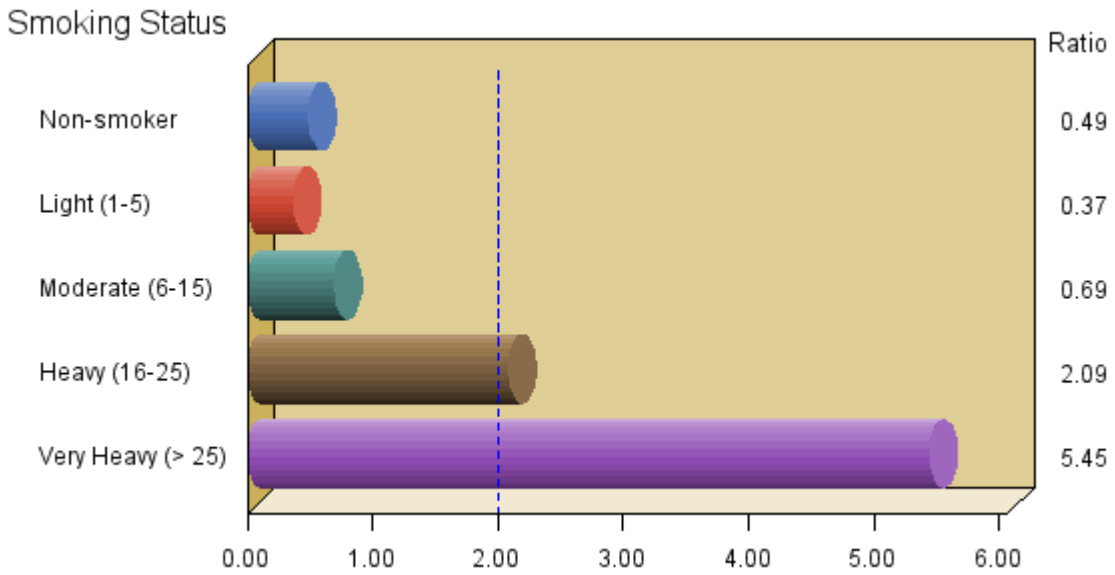
Table 4

Smoking Status	Control Group	Comparison Group	Ratio
Non-smoker	1,682	819	0.49
Light (1-5)	422	157	0.37
Moderate (6-15)	340	236	0.69
Heavy (16-25)	**339**	**707**	**2.09**
Very Heavy (> 25)	**73**	**398**	**5.45**
Unknown	17	19	1.12
Total:	**2,873**	**2,336**	

All rows with a ratio statistic greater than the default threshold of 2.0 have been traffic-lighted automatically. Consequently, our attention is immediately drawn to the "Heavy" and "Very Heavy" groups, which have a much stronger representation among men (the "comparison group") than women (the "control group"). From this, we can conclude that the men who participated in the study appear to be much heavier smokers than the women who participated in the study. This table is easy enough for statistics geeks to read, but to convince common folks, pictures might be helpful. So we will consult our graph macro.

```
%OutputReportGraph(Smoking_Status, Smoking Status, &ReportPath\Smokers.gif,
      SortList=&SmokingSortList);
```

Notice that when we invoke the graph macro, we pass in the SortList again. This is because PROC GCHART ignores the order of the data set and defaults to alphanumeric sorting. Our graph macro (see the appendix) has to tell PROC GCHART the exact order in which to place the bars. So we simply recycle the SortList used earlier, and the following, correctly-sorted graph results.



Conclusion

Your reports may look very different from the type of report discussed here, or they may appear very similar. Either way, there's a good chance that some of the above tips and tricks will help you overcome at least a few of the hurdles that you face in optimizing your own reporting code.

Incorporating these methods into our macros at Infogroup literally reduced our report scripts to a tenth their original length and saved an inestimable amount of time on touching up the Excel files by hand. With determination, ingenuity— and hopefully some of the insights provided here— you can create similar macros to make your own reporting process smarter, cleaner and quicker.

Recommended Reading

Carpenter, Arthur L. 2006. *Advanced PROC REPORT: Traffic Lighting - Controlling Cell Attributes With Your Data*. Cary, NC: SAS Institute, Inc.

Hadden, Louise. 2005. *STOP! WAIT! GO!: See What Traffic-Lighting Can Do For You!* Cary, NC: SAS Institute, Inc.

Karp, Andrew H. 2011. *Traffic-Lighting Your Reports the Easy Way with PROC REPORT and ODS*. Cary, NC: SAS Institute, Inc.

Parker, Chevell. 2003. *Generating Custom Excel Spreadsheets using ODS*. Cary, NC: SAS Institute, Inc.

Shamlin, David. 2003. *ODS for Microsoft Excel*®. Cary, NC: SAS Institute, Inc.

Zender, Cynthia L. 2011. *Don't Gamble with Your Output: How to Use Microsoft Formats with ODS*. Cary, NC: SAS Institute, Inc.

Contact Information

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

Jacob Serafino
Omaha, NE
JacobSerafino@gmail.com

Appendix: Here's the stuff that does it

Macro 1: Creates the report table

```
%macro CreateReportTable(ControlDataReference, ComparisonDataReference,
ReportVariable, SortList);

/* Create frequency tables for the Comparison data & the Control group data. */
PROC FREQ data=&ControlDataReference;
      tables &ReportVariable / nopercent out=ControlFreqs;
run;

PROC FREQ data=&ComparisonDataReference;
      tables &ReportVariable / nopercent out=ComparisonFreqs;
RUN;

/* Create the report table */
data ReportTable;
      /* Join the Comparison & Control group count tables into one. */
      merge ControlFreqs(rename=(Count=CountControl) in=A) ComparisonFreqs (in=B);
            by &ReportVariable;
            if A;

      /* Missing counts are assumed to be 0. */
```

```
	if count = . then count=0;
	if CountControl = . then CountControl=0;

	/* Calculate the ratios. */
	Ratio = count/CountControl;
run;

/* The individual frequency tables are no longer needed, so we delete them. */
proc datasets library=work; delete ControlFreqs ComparisonFreqs; run;

/* If &SortList is not blank, sort the report table accordingly. */
%if &SortList NE %str() %then %do;

/* Assign a rank to each row based on where its value is in the list */
	data ReportTable;
		set ReportTable;

		select (&ReportVariable);
			%let SortIndex=1;
			%do %while(%scan(&SortList,&SortIndex,' ',q) NE %str());
				when ( %scan(&SortList,&SortIndex,' ',q) )
					SortAlias=&SortIndex;
				%let SortIndex=%eval(&SortIndex+1);
			%end;
			otherwise SortAlias=&SortIndex;
			end;
	run;

	/* Sort the table by the index we created in the previous data step */

	proc sort data=ReportTable out=ReportTable(drop=SortAlias);
		by SortAlias;
	run;

%end;

%mend CreateReportTable;
```

Macro 2: Outputs the report table into an Excel file

```
%macro OutputExcelReport(ReportVariable, VariableLabel, ExcelFilename, ThreshHold=2);

ods html file ="&ExcelFilename"  style=minimal
headtext='<Style> td { mso-number-format:\@}</Style>';

proc report nowindows data=ReportTable nofs headline headskip split='*'
	/* Set the background color of the header row to light gray. */
	style(header)=[background=LIGR];
	Title1 "&VariableLabel";
	column &ReportVariable CountControl count ratio;
		define &ReportVariable /display "&VariableLabel";
		define CountControl /sum format=comma12. 'Control*Group' ;
		define count /sum format=comma10. 'Comparison*Group' ;
		define ratio /display format=6.2 'Ratio';

	/* (The ratio has actually already been computed.) */
	compute ratio;
	if ratio >= &ThreshHold then do;
		/* Change the color of the row and make the font bold */
		call define(_row_, "style",
			"style=[background=CX93CDDD font_Weight=bold]");
		/* (The color CX93CDDD was chosen in Excel, and then converted from
			RGB into hex using an online calculator.) */
```

```
        end;
        endcomp;

        /* At the end of the report, put together a summary row containing totals. */
        compute after;
                /* Write "Total:" into the report variable column. */
                &ReportVariable = 'Total: ';
                /* Make the text bold and the background gray. */
                call define(_row_, "style",
                        "style=[background=CXBFBFBF font_Weight=bold]");
        endcomp;

        rbreak after/dol skip summarize;
run;

ods html close;
Title1;

%mend OutputExcelReport;
```

Macro 3: Creates a graph based on the report table

```
%macro OutputReportGraph(ReportVariable, VariableLabel, GraphFilename, SortList,
ThreshHold=2);

goptions reset=all device=GIF gsfname=barchar gsfmode=replace xmax=6 ymax=6;

GOPTIONS ;
        Axis1 minor=none label=(justify=Center height=12pt 'Ratio');
        Axis2 label=(justify=Center height=12pt "&VariableLabel");

FILENAME barchar "&GraphFilename";

PROC GCHART DATA=ReportTable(where=(&ReportVariable NE 'Total:' and Ratio NE 0));
        Hbar3d &ReportVariable
                / description="Vertical Bar Chart for &VariableLabel"
            frame SHAPE=CYLINDER cframe=CXDECD95 RAXIS=AXIS1 MAXIS=AXIS2
            ref=&ThreshHold frontref cref=blue lref=2
               SUM cref=Blue sumvar=ratio sumlabel='Ratio'
            PATTERNID=MIDPOINT
               %if &SortList NE %str() %then %do;
                        MIDPOINTS= &SortList %end;
                        ;
               format Ratio 6.2;
               label &ReportVariable="&VariableLabel" ;
run;
quit;

%mend OutputReportGraph;
```