SESUG Paper 77-2021

Getting Started with Attribute Maps: Methods for Creating and Storing Custom Style Definitions for Graphs, Charts, and Maps

James Blum, University of North Carolina Wilmington
Jonathan Duggins, North Carolina State University

Abstract

In this paper, methods for using attribute maps for controlling attribute map and a few simple examples to contrast setting styles via the map to setting them directly in the chosen ODS Graphics procedure. Both discrete attribute maps and range attribute maps are covered (note: range attribute maps are only available for SAS 9.4M3 and later releases). Advanced examples include defining multiple attribute maps in a single data set and using multiple attribute maps in the same graph. Strategies for making efficient, general use of attribute maps are presented. Attendees should have a decent working knowledge of the SGPLOT Procedure to get the most out of this presentation.

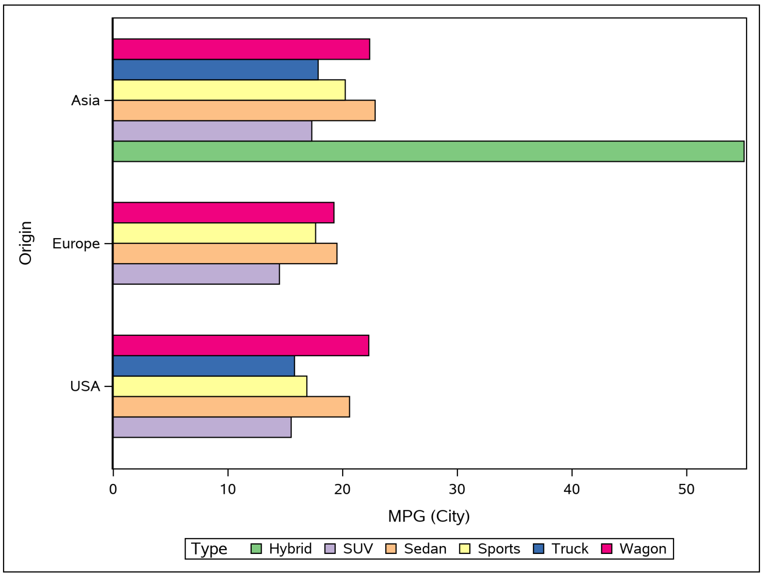
Introduction

Attribute maps are a valuable tool for creating styles for graphs that allow broad application of consistent style attributes. To give a basic illustration of what this means, consider the following examples.

Program 1: Fill Colors in a Grouped Bar Chart

```
proc sgplot data=sashelp.cars;
 styleattrs datacontrastcolors=(black)
  datacolors=(cx7fc97f cxbeaed4 cxfdc086 cxffff99 cx386cb0 cxf0027f);
 hbar origin / group=type groupdisplay=cluster
               response=mpg_city stat=mean;
run;
```

Output 1: Fill Colors in a Grouped Bar Chart

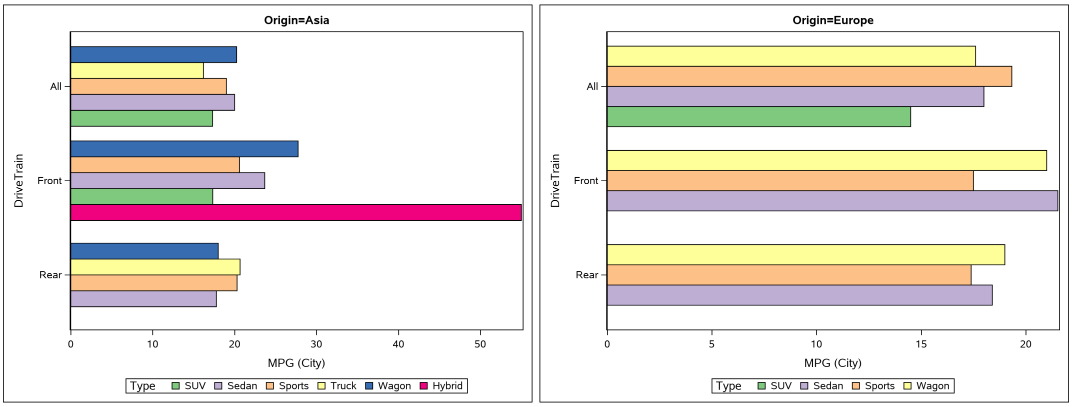


Here, we have set a fill color list (and a single color for all bar outlines), and those colors are applied to the Type variable used in the GROUP= option with the HBAR statement. Not all levels of type appear for each level of the charting variable, Origin, but the coloring of the bars for each Type is consistent across all levels of Origin. This is a characteristic of how colors are applied to different levels of a GROUP= variable in various plotting statements in PROC SGPLOT; however, across multiple plots, no such consistency should be expected, as Program 2 demonstrates.

Program 2: Fill Colors Across Multiple Grouped Bar Charts

```
proc sort data=sashelp.cars out=carSort;
 by origin;
run;

proc sgplot data=carSort;
 styleattrs datacontrastcolors=(black)
  datacolors=(cx7fc97f cxbeaed4 cxfdc086 cxffff99 cx386cb0 cxf0027f);
 by origin;
 hbar drivetrain / group=type groupdisplay=cluster
                response=mpg_city stat=mean;
run;
```

Output 2: First Two Bar Charts from Program 2



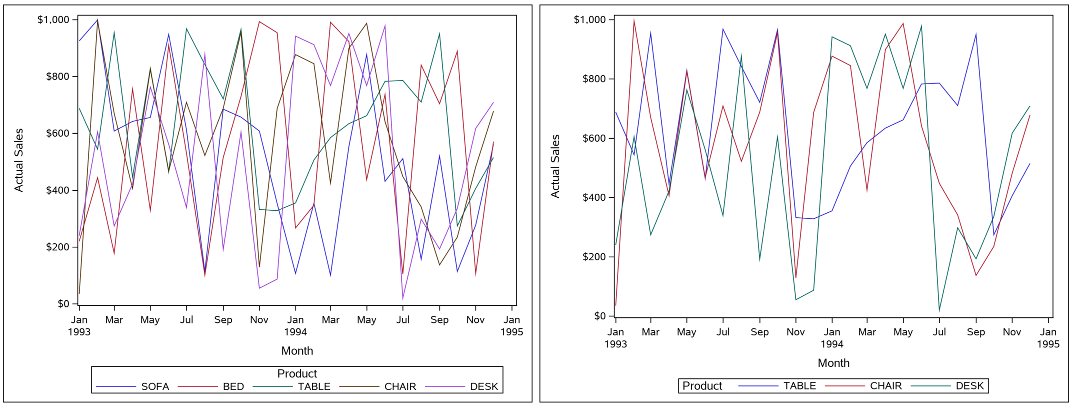
Now the coloring is quite inconsistent across levels of Type. Indeed, you can find quite a bit of chaos–*e.g.*: Where, and how often, is the last color in the DATACOLORS= list used? We could intervene to fix this by making one graph at a time using WHERE subsetting and making careful choices for our color list in each, but it becomes quite cumbersome.

Program 3 gives a pair of SERIES plots with a similar result.

Program 3: Grouped Series Plots

```
proc sort data=sashelp.prdsale out=salesort;
 where country eq 'CANADA' and region eq 'EAST'
       and division eq 'EDUCATION';
 by region month;
run;

proc sgplot data=salesort;
 series x=month y=actual / group=product;
 format month monyy.;
run;

proc sgplot data=salesort;
 where prodtype eq 'OFFICE';
 series x=month y=actual / group=product;
 format month monyy.;
run;
```

Output 3: Grouped Series Plots



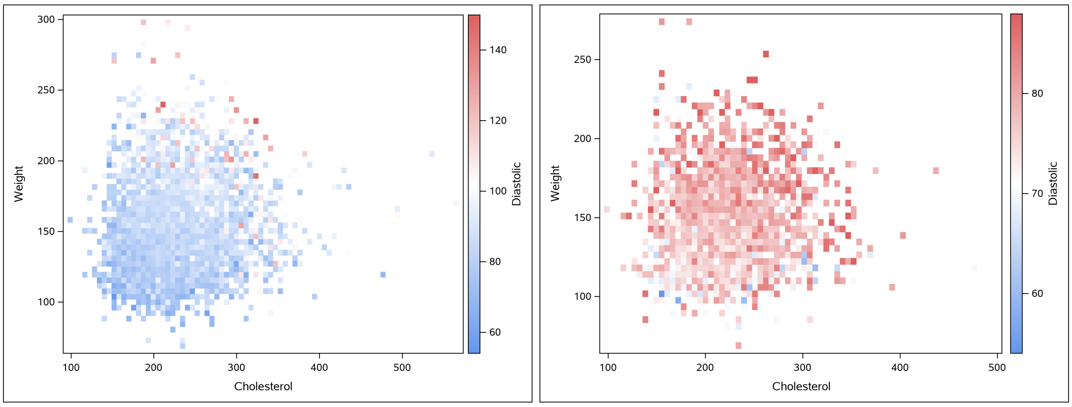
Fixing the coloring consistency for this one would be a little bit easier, but we would have to set correct lists for DATACONTRASTCOLORS= in both calls to PROC SGPLOT.

Program 4 shows one more example, this time where subsetting the data for a heatmap alters the color scale.

Program 4: Scales Are Dynamic for Heatmaps

```
proc sgplot data=sashelp.heart;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean;
run;

proc sgplot data=sashelp.heart;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean;
 where bp_status ne 'High';
run;
```

Output 4: Scales Are Dynamic for Heatmaps



So the color ramps are obviously going to be different in the two graphs and, due to the way PROC SGPLOT constructs the color ramp, it can be quite difficult to align them.

For any of these situations, the attribute map lets you define, as data, rules for how to set these attributes under various conditions. ***Discrete attribute maps*** are designed for setting attributes for specific data values, corresponding to levels of GROUP= variables. ***Range attribute maps*** let you assign attributes to ranges of quantitative (or numeric) values, like those built in the color ramp for a heatmap, or polygon colors in a choropleth map. In the following sections, we will see how to use attribute maps to build consistency in graph style schemes, and create strategies for effective use of attribute maps in other scenarios.

Creating and Using a Discrete Attribute Map

Program 5 builds a discrete attribute map and uses it to get consistent colorings across the graphs originally produced in Output 2.

Program 5: Using a Discrete Attribute Map to Get Consistent Fill Colors

```
data attrmap1;❶
 input ID$ value$ fillcolor$;❷
 datalines;❸
map1 Hybrid cx7fc97f
map1 SUV cxbeaed4
map1 Sedan cxfdc086
map1 Sports cxffff99
map1 Truck cx386cb0
map1 Wagon cxf0027f
 ;
run;

proc sgplot data=carSort dattrmap=attrmap1❹;
 styleattrs datacontrastcolors=(black);
 by origin;
 hbar drivetrain / group=type groupdisplay=cluster
             response=mpg_city stat=mean
             attrid=map1❺;
run;
```

❶ The attribute map is stored in a data set that is referenced in the PROC SGPLOT statement.

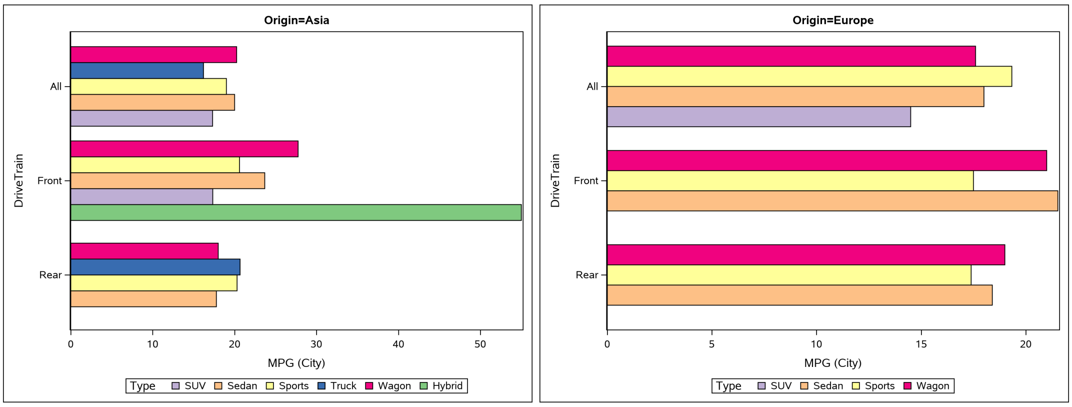
❷ The variable names for attribute maps are prescribed (much like annotation data sets, if you have done those). A variable named ID is required, and it names the attribute map. For discrete attribute maps, a variable named Value is also required, and it corresponds to the values of a GROUP= variable used in a graph or chart. Other variables are for specific attributes, see the list in Properties 2 in the Appendix.

❸ Here, instream data is used to build the attribute map, though generally other strategies would be employed (see Strategies for Building Attribute Maps). Indeed, you should be careful with the data specification–leading spaces in any data line can get translated into leading spaces for the ID variable here, and it will not be properly recognized when referenced in the plotting statement discussed in ❺.

❹ The DATTRMAP= refers to the data set that contains the attribute map you wish to use. However, since a data set can contain multiple attribute maps, the data set name is not sufficient information to apply an attribute map.

❺ The ATTRID= is specified in the plotting statement. What is placed here determines which rows of the DATTRMAP= data set are used to set attributes for that plot. See Properties 1 in the Appendix for more details. This must reference a legal name under the V7 standard for variable and data set names no matter what specification of VALIDVARNAME= or VALIDMEMNAME= is in use. For example, if you build the variable ID as Attr-Map and attempt to use ATTRID='Attr-Map'n, as is done for VALIDVARNAME=ANY, an error is produced in the log and PROC SGPLOT fails to execute.

Output 5: Using a Discrete Attribute Map to Get Consistent Fill Colors (Partial Output)



So now the fill color applied to type is consistent across all graphs generated, since they all use the same attribute map. Examples that follow illustrate some more properties of the attribute map data sets. There are many attributes than can be set, and many variations on the usage of those and other variables. Not all can be covered here, so it is important to look at the Appendix and the SAS documentation for more information.

Program 6 sets two attribute variables to make the line colors and styles consistent across the series plots in Output 3. It also intentionally puts the Value variable in different casing than the GROUP= variable, and uses the NoCase variable to modify matching rules to correct this potential mis-alignment.

Program 6: Using a Discrete Attribute Map to Get Consistent Line Colors and Patterns

```
data attrmap2;
 retain❶ ID 'PMap' linepattern 1❷ linethickness 5❸ nocase 'true'❹;
 input value$ linecolor$ @@;
 datalines;
Bed cxbeaed4 Chair cxfdc086 Desk cxffff99 Sofa cx386cb0 Table cx7fc97f
 ;
run;
proc sgplot data=salesort dattrmap=attrmap2❺;
 series x=month y=actual / group=product attrid=PMap❺;
 format month monyy.;
run;
proc sgplot data=salesort dattrmap=attrmap2❺;
 where prodtype eq 'OFFICE';
 series x=month y=actual / group=product attrid=PMap❺;
 format month monyy.;
run;
```

❶ If you are using instream data to create a single attribute map, the RETAIN statement can be quite helpful for setting values that are constant for all records, like those for ID.

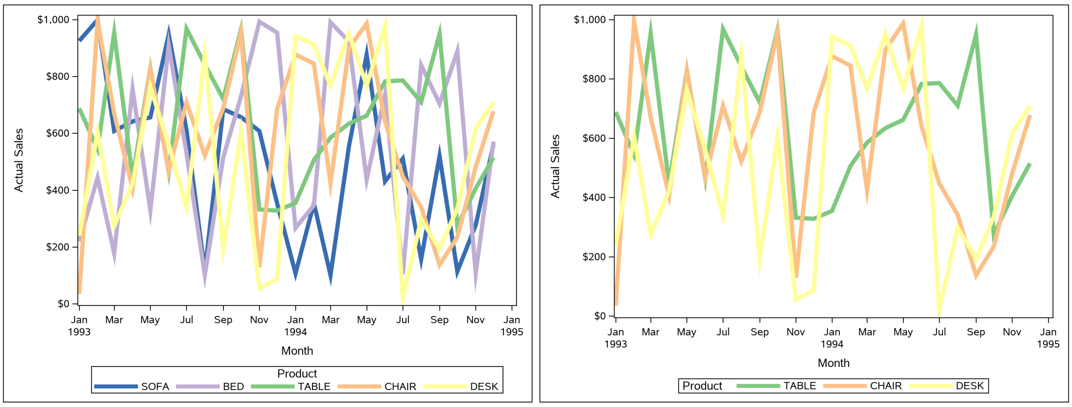
❷ LinePattern is documented as a character variable, and it must be if you are using a pattern name. However, if you use a pattern number, it can be defined as either character or numeric.

❸ LineThickness must be numeric and is automatically interpreted as having pixels as the unit. This is one limitation of attribute maps–when setting the thickness using an ATTRS option, we get a choice of units, not so for attribute maps. In general, attributes that allow units are limited to the default unit for attribute maps.

❹ The optional NoCase variable is included in this attribute map, but does not set a graph attribute. Its setting of TRUE means the matching of values in the Value variable in the attribute map and the values of the GROUP= variable in the plotting statement ***ignore casing differences***. Note that the values in the data set for the Product variable are in upper case, while the data built for the attribute map has them in proper case, but the attributes are correctly matched with casing ignored. If NoCase were FALSE for this example, no observations for Value in the attribute map are matched to the values for Product, and default colors are used for the bar fills.

❺ Of course, the key to getting the consistency in the graphs is to use the same attribute map for all of them. So each call to PROC SGPLOT uses the same attribute map data set in DATTRMAP=, and each plotting statement (SERIES) uses the same ATTRID= value.

Output 6: Using a Discrete Attribute Map to Get Consistent Line Colors and Patterns



So the five levels of the variable Product are each assigned three attributes for line styling: color, thickness, and pattern. In any plot using this attribute map that uses Product as the GROUP= variable, those line styles are applied to line elements generated in the plot. So if it is applied to a bar graph, the bar outlines are altered, but the fills are in the default color, as this attribute map makes no specification for those. In general, we want to think broadly about how the map may be applied whenever the Value variable corresponds to the GROUP= variable.

Building Multiple Discrete Attribute Maps in a Single Data Source

One of the major advantages to using discrete attribute maps is that a large collection of them can be stored and maintained in a single location. The presence of the ID variable permits storage of multiple attribute maps in a single data set. Program 7 creates one attribute map data set, containing two attribute maps, and uses one of each in separate calls to PROC SGPLOT.

Program 7: Multiple Discrete Attribute Maps in a Single Data Set

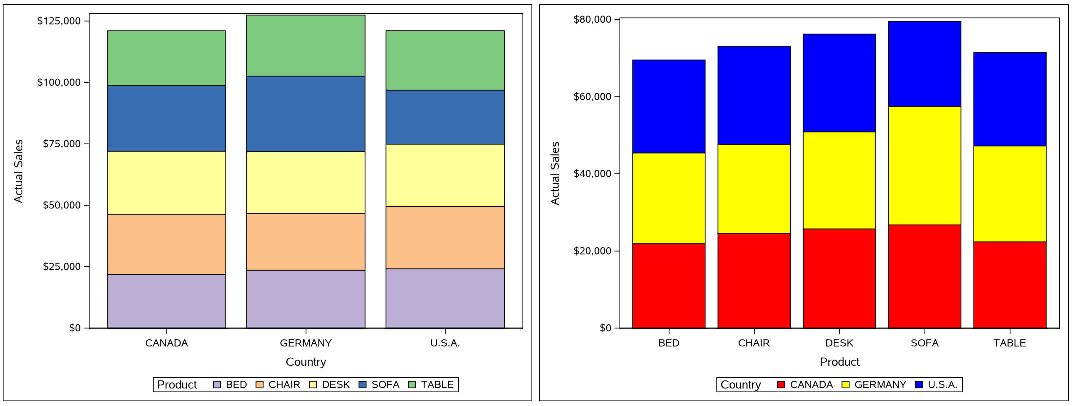
```
data attrmap3;
 retain nocase 'True' linecolor 'black';
 input ID$ value$ fillcolor$;
 datalines;❶
map1 Bed cxbeaed4
map1 Chair cxfdc086
map1 Desk cxffff99
map1 Sofa cx386cb0
map1 Table cx7fc97f
map2 Canada red
map2 Germany yellow
map2 _other_ blue
 ;
run;

proc sgplot data=sashelp.prdsale dattrmap=attrmap3❷;
 vbar country  / group=product response=actual stat=sum attrid=map1❸;
 where year eq 1993;
run;

proc sgplot data=sashelp.prdsale dattrmap=attrmap3❷;
 vbar  product / group=country response=actual stat=sum attrid=map2❸;
 where year eq 1993;
run;
```

❶ This collection of data now uses two different values for the ID variable (Map1 and Map2). The values for the first correspond to those for the Product variable (and it is the same map used in Program 5), while the second correspond to the Country variable. Since it is intended to use the same outline color for bars in either map, the LineColor attribute has a common setting for both maps.

❷ Both calls to PROC SGPLOT can use the same attribute map data set since it contains an attribute map that is appropriate for each.

❸ Each plot has its ATTRID= value set to the map that corresponds to the values present for the GROUP= variable. Accidentally selecting Map1 in the second SGPLOT has no serious consequences, it just uses default colors since none of the values of Country are in that map. Accidentally selecting Map2 for the first SGPLOT produces an extremely poor result–try it, and see if you can determine why.

Output 7: Multiple Discrete Attribute Maps in a Single Data Set



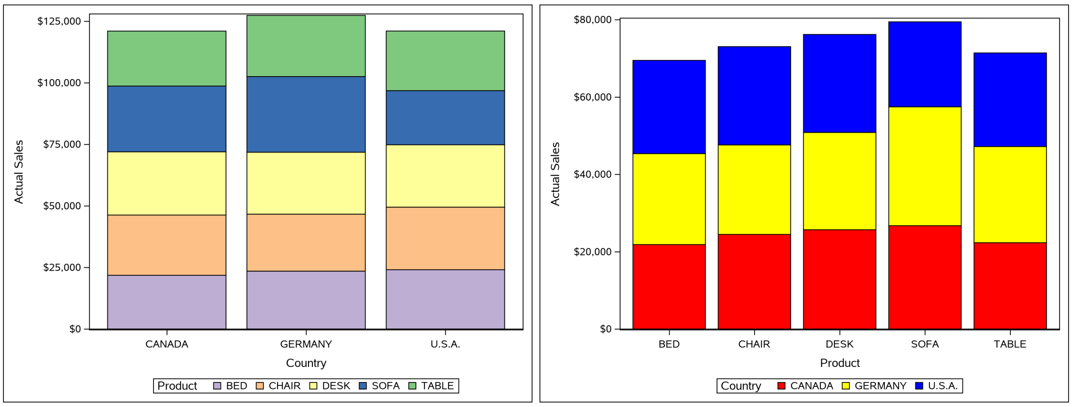
When defining multiple attribute maps in a data set (discrete or range), the documentation states: *The ID values in the attribute map data set must be continuous (in a sorted order). If they are not, use the SORT procedure to sort the data set by ID, in ascending or descending order.* Values of the ID variable indeed must form contiguous blocks, but the blocks need not be in any particular sort order (not unlike use of the NOTSORTED option). Program 8 illustrates this.

Program 8: Grouping Multiple Discrete Attribute Maps in a Single Data Set

```
data attrmap3B;
 retain nocase 'True' linecolor 'black';
 input ID$ value$ fillcolor$;
 datalines;
map1 Bed cxDDDD33
map1 Chair cyan
map1 Desk cx33FF33
map1 Sofa cx3333FF
map1 Table magenta
map3 Canada red
map3 Germany yellow
map3 _other_ blue
map2 Bed cxbeaed4
map2 Chair cxfdc086
map2 Desk cxffff99
map2 Sofa cx386cb0
map2 Table cx7fc97f
 ;
run;

proc sgplot data=sashelp.prdsale dattrmap=attrmap3B;
 vbar country  / group=product response=actual stat=sum attrid=map2;
 where year eq 1993;
run;

proc sgplot data=sashelp.prdsale dattrmap=attrmap3B;
 vbar  product / group=country response=actual stat=sum attrid=map3;
 where year eq 1993;
run;
```

The values of the ID variable are not in a sorted order, but they are in groups of records in the data set, so they work as expected when applied in the same manner as Program 7. Try sorting Attrmap3B by the Value variable and then resubmit the calls to PROC SGPLOT. Only one color from the attribute map is applied, and it is the last one–the final contiguous block of the ID variable determines the attribute map, with particularly poor results for Map3. If you do something like that, there are also no notes, errors, or warnings in the log indicating that the expected structure of the attribute map data set is violated, it just uses the last contiguous block.

Output 8: Grouping Multiple Discrete Attribute Maps in a Single Data Set



Next, we explore using different attribute maps in the same plot call–it can be done, but it will likely confound your expectations. Program 9 makes two grouped bar charts in the same graph, but uses two different grouping variables and, therefore, a different attribute map for each.

Program 9: Using Multiple Discrete Attribute Maps in the Same Plot

```
data attrmap4;
 retain nocase 'True' linecolor 'Black' show 'ATTRMAP';
 input ID$ value$ fillcolor$;
 datalines;❶
product Bed cxEEEE33
product Chair cxEEBB33
product Desk cx33FF33
product Sofa cx3333FF
product Table cxDD33DD
region East cx6666FF
region West cxFF6666
;
run;

proc sgplot data=sashelp.prdsale dattrmap=attrmap4;
 vbarbasic❷ country  / group=product response=actual stat=sum
   attrid=product❸ discreteoffset=-0.21 barwidth=0.4 ❹ name='Product'❺;
 vbarbasic❷ country  / group=region response=actual stat=sum
   attrid=region❸ discreteoffset=0.21 barwidth=0.4 ❹ name='Region'❺;
 where year eq 1993;
 xaxis display=(nolabel);
 keylegend 'Product' / position=topleft title='Product';
 keylegend 'Region' / position=bottomright title='Region';
run;
```

❶ Maps are built with values that correspond to the Product and Region variables.

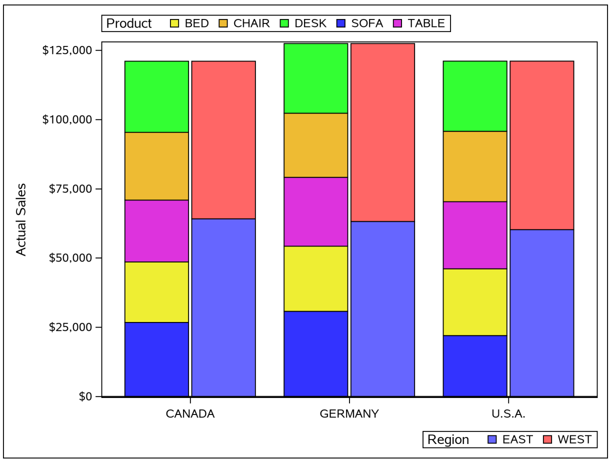
❷ VBAR will not overlay bar charts with different GROUP= variables, they are considered incompatible. VBARBASIC allows for more freedom in overlaying "different" charts and must be used to make this overlay work.

❸ ATTRID= can now refer to different ID variables in the attribute map data set.

❹ The overlaying of the plots is not actually going to look good if the bars intersect, so some intervention is done to place them side-by-side (the grouping is done as a stack).

❺ Each plot produces a legend, so each is named and given its own attributes, particularly the position.

Output 9: Using Multiple Discrete Attribute Maps in the Same Plot



It would seem then, that an easier case might arise when using two different attribute maps when overlaying plots on the same GROUP= variable. However, Program 10 shows that this is not the case.

Program 10: Using Multiple Discrete Attribute Maps on the Same GROUP= Variable

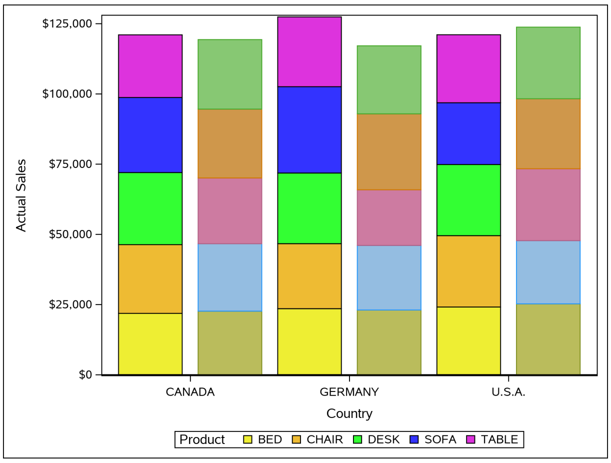
```
data attrmap5;
 retain nocase 'True' linecolor 'Black';
 input ID:$9. value$ fillcolor$;
 datalines;❶
actual Bed cxEEEE33
actual Chair cxEEBB33
actual Desk cx33FF33
actual Sofa cx3333FF
actual Table cxDD33DD
predicted Bed cxEEEE66
predicted Chair cxEEBB66
predicted Desk cx66FF66
predicted Sofa cx6666FF
predicted Table cxDD66DD
;
run;

proc sgplot data=sashelp.prdsale dattrmap=attrmap5;
 vbar country / group=product response=actual❷ stat=sum
   attrid=actual❸ discreteoffset=-0.25 barwidth=0.4;
 vbar country / group=product response=predict❷ stat=sum
   attrid=predicted❸ discreteoffset=0.25 barwidth=0.4;
 where year eq 1993;
run;
```

❶ Here two attribute maps are made, each for use with Product as the GROUP= variable, but potentially for different RESPONSE= variables of Actual and Predicted.

❷ The GROUP= variable is the same in both plots, but the response variables are different.

❸ The attribute map is chosen differently for different responses.

Output 10: Using Multiple Discrete Attribute Maps on the Same GROUP= Variable



The first set of stacked bars get the colors mapped correctly, but the second set (corresponding to Product as the response) do not. A warning in the log indicates the problem: *WARNING: Only one attrid can be assigned to the "PRODUCT" variable. The additional attrid will be ignored.* So even though both attribute maps align with the values of the GROUP= variable, and they are used in separate plotting statements, this request is still not permitted. And, no, it does not work with VBARBASIC either. In order to get this done, a new grouping variable needs to be constructed that includes information on the grouping and response variables simultaneously, effectively making one map with 10 levels for this case, instead of two with 5 levels each–the details are not provided here.

Range Attribute Maps

Range attribute maps are used to assign colors, or color ramps, to value ranges, as used in heatmaps or choropleth maps. Range attribute maps differ significantly from discrete attribute maps beyond merely having a different name. As you might expect, the attribute variables are different; however, you might not expect that the options to request them in SGPLOT are different as well. Program 11 modifies Program 4 in an attempt to get the same mapping of colors to values in both heatmaps.

Program 11: A Range Attribute Map

```
data rngAttMap;
 retain ID 'Range1';❶
 input min$ max$ color$; ❷
 datalines;❸
_min_ 70 blue
70 80 yellow
80 _max_ red
;
run;

proc sgplot data=sashelp.heart rattrmap=rngAttMap❹;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range1❺;
run;

proc sgplot data=sashelp.heart  rattrmap=rngAttMap❹;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range1❺;
 where bp_status ne 'High';
run;
```

❶ One item that is absolutely consistent across discrete and range attribute maps is the requirement of an ID variable. The same rules apply, including those for cases where multiple attribute maps are stored in the same data set.

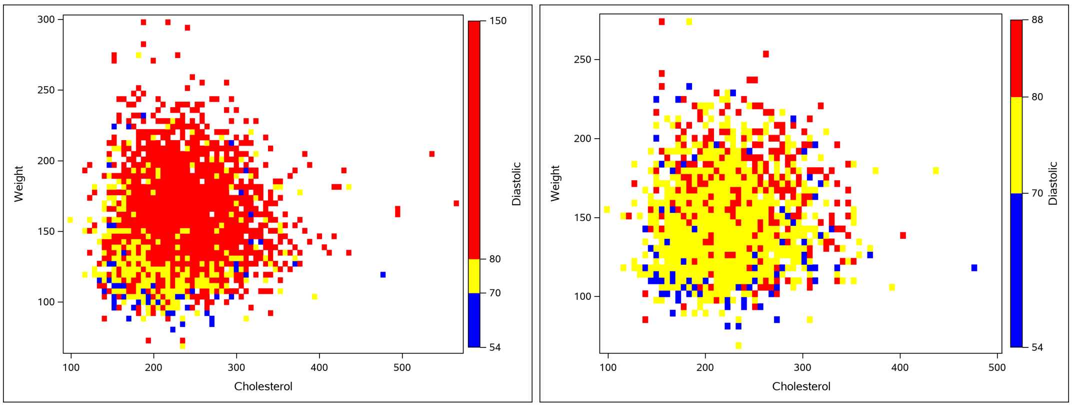
❷ There is no Value variable in use for range attribute maps; instead, the Min and Max variables are expected to define the groups. (Max may not be required when certain special keywords are used for Min, see Properties 3.) The Color variable specifies a single fill color for the range.

❸ _MIN_ and _MAX_ are keywords that can be used for the Min and Max variables, respectively. They represent the lowest and highest values in the data selected for the plot.

❹ A range attribute map data set is selected with RATTRMAP= (so the D in DATTRMAP= stands for discrete, not data). It is possible that some overlayed plots can use a range attribute map for one plot, and a discrete attribute map for another, so the distinction in name makes it possible to use both in one call to PROC SGPLOT.

❺ The option that identifies the ID variable for the map also has a different name for range attribute maps, RATTRID=. (No, I do not know why it is not DATTRID= for discrete attribute maps.) In this case, since HEATMAP does not permit groups, it does not permit the ATTRID= option at all.

Output 11: A Range Attribute Map



To say that this output is less than desirable would be quite an understatement. The single color variable on each range effectively reduces it to something akin to a discrete attribute map. Program 12 makes use of color attributes in a different way to produce a better result.

Program 12: An Improved Range Attribute Map

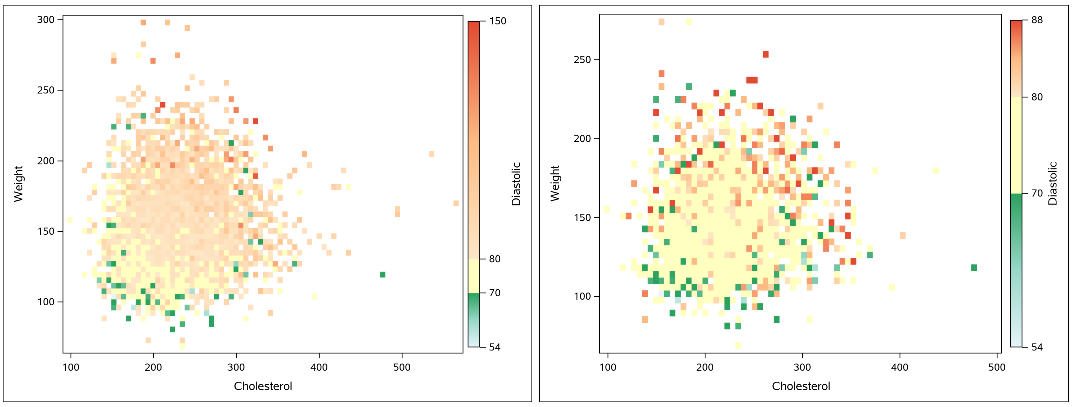
```
data rngAttMap2;
retain ID 'Range2';
length colormodel1-colormodel3 $8;①
infile datalines missover;②
input min$ max$ colormodel1-colormodel3;
datalines;③
_min_ 70 cxe5f5f9 cx99d8c9 cx2ca25f
70 80 cxffffbf
80 _max_ cxfee8c8 cxfdbb84 cxe34a33
;
run;

proc sgplot data=sashelp.heart rattrmap=rngAttMap2;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range2;
run;

proc sgplot data=sashelp.heart  rattrmap=rngAttMap2;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range2;
 where bp_status ne 'High';
run;
```

① The ColorModel variables (with a positive integer suffix) allow a range to be associated with a set of colors, which are used to form a color ramp.

② The INFILE statement with the MISSOVER option is used here because not all ranges are associated with the same number of colors to create the ramp.

③ The top and bottom ranges use a three color ramp, but the middle range only references a single color.

Output 12: An Improved Range Attribute Map

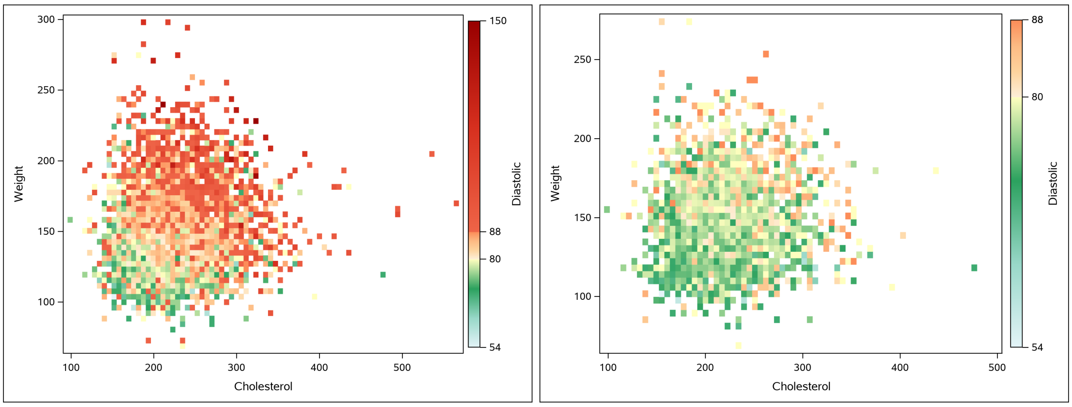


This is better, but the less-than-careful use of the _Max_ keyword means that the lowest two color ranges are consistent across the two graphs, but the upper range spans different values in the two plots. Program 13 makes a further improvement.

Program 13: More Improvement to the Range Attribute Map

```
data rngAttMap3;
retain ID 'Range3';
length colormodel1-colormodel4 $8;
infile datalines missover;
input min$ max$ colormodel1-colormodel4;
datalines;
_min_ 80 cxe5f5f9 cx99d8c9 cx2ca25f cxffffbf
80 88 cxfef0d9 cxfdd49e cxfdbb84 cxfc8d59
88 _max_ cxef6548 cxd7301f cx990000
;
run;

proc sgplot data=sashelp.heart rattrmap=rngAttMap3;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range3;
run;

proc sgplot data=sashelp.heart  rattrmap=rngAttMap3;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range3;
 where bp_status ne 'High';
run;
```

Output 13: More Improvement to the Range Attribute Map



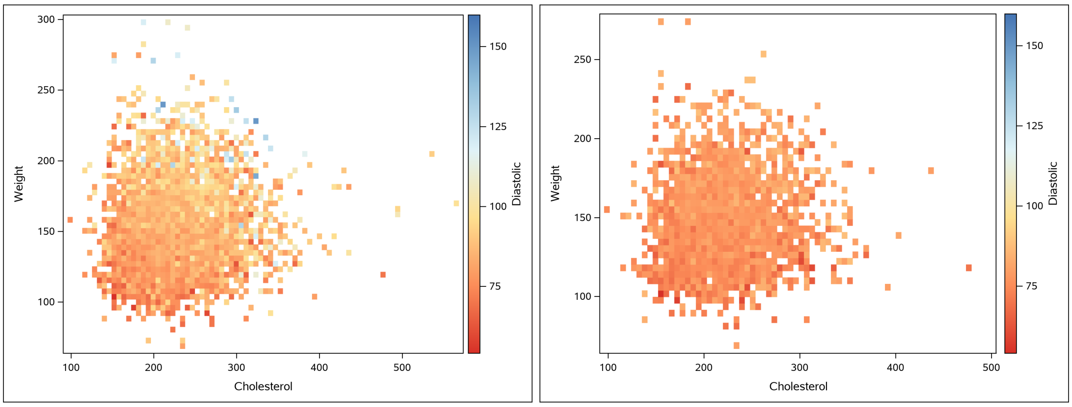
So the color ramping is consistent, but it was required to know that 88 was the highest value in the data selected for the second plot. If the 88 is replaced with any larger value, say 89, the value range on the last element of the attribute map is invalid for the second plot–the value corresponding to _MAX_ (now 88) is less than 89. Such a violation prevents the range attribute map from being used.

Program 14 avoids using the keyword _MAX_ to make the color ramping consistent across both graphs. Instead it just uses a color ramp from the minimum to a specific value just above the maximum for all cases, but this introduces another characteristic that may be undesirable.

Program 14: A Range Attribute Map with a Fixed Upper Limit

```
data rngAttMap4;
retain ID 'Range4';
length colormodel1-colormodel6 $8;
input min$ max$ colormodel1-colormodel6;
datalines;
_min_ 160 cxd73027 cxfc8d59 cxfee090 cxe0f3f8 cx91bfdb cx4575b4
;
run;

proc sgplot data=sashelp.heart rattrmap=rngAttMap4;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range4;
run;

proc sgplot data=sashelp.heart  rattrmap=rngAttMap4;
 heatmap x=cholesterol y=weight /
   colorresponse=diastolic colorstat=mean rattrid=Range4;
 where bp_status ne 'High';
run;
```

Output 14: A Range Attribute Map with a Fixed Upper Limit



So the color ramp is quite consistent for the two plots, and perhaps even too consistent. Not using the _MAX_ keyword removes some of the dynamics from the gradient legend construction, and the full legend is constructed for each plot. This may be desirable, but if it is not, it cannot be altered using this range attribute map–even subsetting on the response variable in the second plot will not change this behavior.

It should be noted that the examples given in this section are some of the more problematic applications for range attribute maps. If your plots have roughly the same range of values for the responses in question, application of the range attribute map will give you consistent coloring quite easily. However, if you want consistent color ramping across data values, but the ranges within each plot differ substantially, some compromises will have to be made, and/or extra work done.

Strategies for Building Attribute Maps

The examples in this paper use instream data to create the attribute maps, but for large-scale use, this is inefficient. Though a full discussion of methods for building attribute maps is beyond the scope of this paper, a few general guidelines are itemized below:

- Maintain data sets that provide color and other style lists. Several resources are available for choosing colors, and for both discrete and continuous groups, a favorite is ColorBrewer.

- Derive your attribute maps from data (or samples or mock-ups of data) that they will be used with. Using PROC SORT with a NODUPKEY option, PROC FREQ, or PROC SQL with DISTINCT are great ways to build data sets that contain appropriate information for the Value variable. Various percentiles generated with PROC MEANS or PROC UNIVARIATE can be helpful in building ranges in range attribute maps.
- Use tools like merging in the DATA step or joins in PROC SQL to put together information generated by the previous two strategies.
- If you are a user of CNTLIN= to use data to create formats, the data sets you are using to build formats can make excellent starting points for attribute maps as well.

Conclusion

Attribute maps are a powerful tool for setting consistent styling specifications across large sets of graphs. They are not necessarily trivial, and may require some planning and preparation to get the most out of them. However, if you do graphics on a large-scale, the time invested can save significant effort when it comes time to code up a set of graphs, charts, or maps.

Contact Information

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

James Blum

Univeristy of North Carolina Wilmington

blumj@uncw.edu

http://people.uncw.edu/blumj

Appendix

Properties 1: Non-Attribute Variable Names and Roles for Discrete Attribute Maps

- *Required Variables*

  ID: A character variable, the value of which is referenced in the ATTRID= option. Not only is this variable required, but every row must have a value. Though it is not explicitly stated in the documentation, the value of this variable must follow the V7 naming conventions for variables/data sets. It is possible to have multiple values if multiple attribute maps are defined in a single data set–see Building Multiple Discrete Attribute Maps in a Single Data Source for more requirements under these conditions.

  Value: A variable with values that correspond the values of a variable used in a GROUP= option in a plotting statement in PROC SGPLOT (or SGPANEL or SGSCATTER). It is documented as having the character type, but it can be numeric or character if the group variable has numeric type. It also accepts keyword values of _OTHER_ (which operates like the Other keyword in a format definition) and _MISSING_ (to set attributes for missing values if they are in use in the plot)–these are never case sensitive.

  If either the ID or Value variable is missing from a chosen discrete attribute map, a Warning message is printed in the log and the attribute map request is ignored.

- *Other Variables*

  NoCase: This is a character variable with valid values of True or False (but not case-sensitive). If it is not present, the default value is False; which, in a somewhat circular way, says that not matching casing in comparisons is false. Or, matching of character values is case-sensitive, as we generally expect. Setting this to True turns off case-sensitivity in matching values between the Value variable in the attribute map and the GROUP= variable in the plotting statement.

  Show: This is a character variable with valid values of ATTRMAP or DATA (also not case-sensitive). If it is not present, the default value is DATA, which means that the legend for the GROUP= variable only contains entries in the data set. When set to ATTRMAP, all values present (but not keywords) for the Value variable are populated into the legend.

- ***Variable names in attribute map data sets are not case-sensitive.***

Properties 2: Discrete Attribute Variable Names and Roles

FillColor: A character variable that sets the color for fill elements. Values can use any of the color-naming schemes available in SAS.

FillTransparency: A numeric variable that sets the transparency level for fills. The range is 0 to 1, 0 is no transparency/fully opaque, 1 is completely transparent (thus, no fill).

LineColor: A character variable that sets the color for lines or curves. Values can use any of the color-naming schemes available in SAS. If line labels are present, this color is also applied to them.

LinePattern: A variable that sets the pattern, solid or various forms of dotting/dashing, for lines or curves. Typically, this is a character variable, but if the numbered references to line patterns are in use, it can be either numeric or character. See the documentation for line style names and numbers.

LineThickness: A numeric variable that sets the line thickness in pixels.

MarkerColor: A character variable that sets the color for plot markers. Values can use any of the color-naming schemes available in SAS. If marker labels are present, this color is also applied to them.

MarkerSize: A numeric variable that sets the marker size in pixels.

MarkerSymbol: A character variable that sets the symbol for the markers–see the documentation for symbol names.

MarkerTransparency: A numeric variable that sets the transparency level for markers, with the same property as that for fills.

TextColor: A character variable that sets the color for text in text plots or axis tables. Values can use any of the color-naming schemes available in SAS.

TextFamily: A character variable that sets the font for text in axis tables. See the documentation for font specifications.

TextStyle: A character variable that can be set to Normal or Italic (values are not case-sensitive) to alter the style of text in axis tables.

TextWeight: A character variable that can be set to Normal or Bold (values are not case-sensitive) to alter the weight of text in axis tables.

- *Each attribute group also has a variable that allows for selection of existing style elements to set attributes. See the SAS documentation for further details.*

Properties 3: Variable Names and Roles for Range Attribute Maps

- *Required Variables*
  - **ID:** A character variable, the value of which is referenced in the ATTRID= option. Not only is this variable required, but every row must have a value. Though it is not explicitly stated in the documentation, the value of this variable must follow the V7 naming conventions for variables/data sets. It is possible to have multiple values if multiple attribute maps are defined in a single data set–see Building Multiple Discrete Attribute Maps in a Single Data Source for more requirements under these conditions.
  - **Min and Max:** The Min and Max variables define the value ranges for the attribute map. The Min variable is required, but the Max variable is sometimes not necessary when certain keywords are used for the MIN variable. See the SAS documentation for further details.
- *Attribute Variables*
  - **AltColor:** Assigns a single color to lines, markers, or both for the range given.
  - **AltColor1-AltColorN:** Specifies several consecutive line/marker colors for the range, creating a gradient across the range. This must be a contiguous set, any gaps in numbering prevent colors after the gap from being used.
  - **Color:** Assigns a single color to fills for the range given.
  - **ColorModel1-ColorModelN:** Specifies several consecutive fill colors for the range, creating a gradient across the range. This must be a contiguous set, any gaps in numbering prevent colors after the gap from being used.
  - Each attribute group also has a variable that allows selection of existing style elements to set attributes. See the SAS documentation for further details.
  - Values for any color attribute variable can use any of the color-naming schemes available in SAS.
- *Other Variables*
  - **ExcludeMin/ExcludeMax:** Specify whether to include the minimum or maximum value in the range specification. Default is True or 1, can be set to False or 0 (using an appropriate type for the variable). Value ranges can only overlap on a single value, and the common value belongs to the lower range if ExcludeMin or ExcludeMax are not in use.
- ***Variable names in attribute map data sets are not case-sensitive.***