SAS5300-2016

Annotating the SAS® ODS Graphics Way!

Dan Heath, SAS Institute Inc., Cary, NC

ABSTRACT

For some users, having an annotation facility is an integral part of creating polished graphics for their work. To meet that need, we created a new annotation facility for the SAS® ODS Graphics (SG) procedures in SAS® 9.3. Now, with SAS® 9.4, the Graph Template Language (GTL) supports annotation as well! In fact, the GTL annotation facility has some unique features not available in the SG procedures, such as using multiple sets of annotation in the same graph and the ability to bind annotation to a particular cell in the graph. This presentation will cover some basic concepts of annotating that are common to both GTL and the SG procedures. Then, I will apply those concepts to demonstrate some unique abilities of GTL annotation.

INTRODUCTION

The abilities of the ODS graphics system have grown significantly since the system was first released in SAS® 9.2. SAS development needs and customer feedback have helped ODS graphics become a feature-rich system for creating a variety of graphics used in many industries.

One important feature we added was annotation support. This support was initially added to the SG procedures in SAS 9.3, and included a new keyword-based data set definition and new capabilities. I wrote a paper called, "Now You Can Annotate Your Statistical Graphics Procedure Graphs" (Heath, 2011), which presents the details of this functionality, along with many examples. It is interesting to note that, since that paper was written, several of the examples no longer require annotation to create them, including the following:

- Adding Unicode characters in tick values – You can now add Unicode characters to format definitions.
- Splitting of tick values – You can specify this behavior as an axis fit policy.
- Creating axis-aligned tables – You can now use the AXISTABLE statement to help create these displays.
- Using images as scatter points – You can now use the SYMBOLIMAGE statement to define an image as a marker type that can be used anywhere that a scatter marker can be used.

Despite the advances we continue to make in functionality, there is always a need for annotation. This is why, in SAS 9.4, we took the next step and added annotation support to the SAS® Graph Template Language (GTL). While adding this functionality, we extended the capabilities of the annotation facility to give users the ability to draw annotations in multiple regions and data spaces within the same graph.

In this paper, I want to begin by establishing a foundation of how the annotation facility works, both from an SG procedure and a GTL context. Then, I will show several examples featuring the various annotation types, focusing primarily on GTL examples.

DRAWING SPACES

Before creating annotations, it is important to understand the concept of drawing spaces. A graph contains multiple drawing areas that can be used as a basis for drawing annotations. This is particularly true for GTL, where layouts can be nested and combined to create more complex graphs. In Figure 1, this diagram shows the four types of drawing spaces inside of a typical single-celled graph. These types of graphs can be generated from PROC SGPLOT, or from GTL using a LAYOUT OVERLAY as your root layout.

The DRAWSPACE keywords used in the data set are a combination of the drawing area and the drawing unit. All drawing areas support pixel and percentage units. The data drawing area also supports data

value units for positioning annotations using axis values. For example, the keyword for drawing an annotation in the wall area using percentage units is WALLPERCENT. These keywords can be specified in the DRAWSPACE column to apply to all coordinates of the drawing function, or they can be specified on specific drawing space columns for each coordinate of the function.

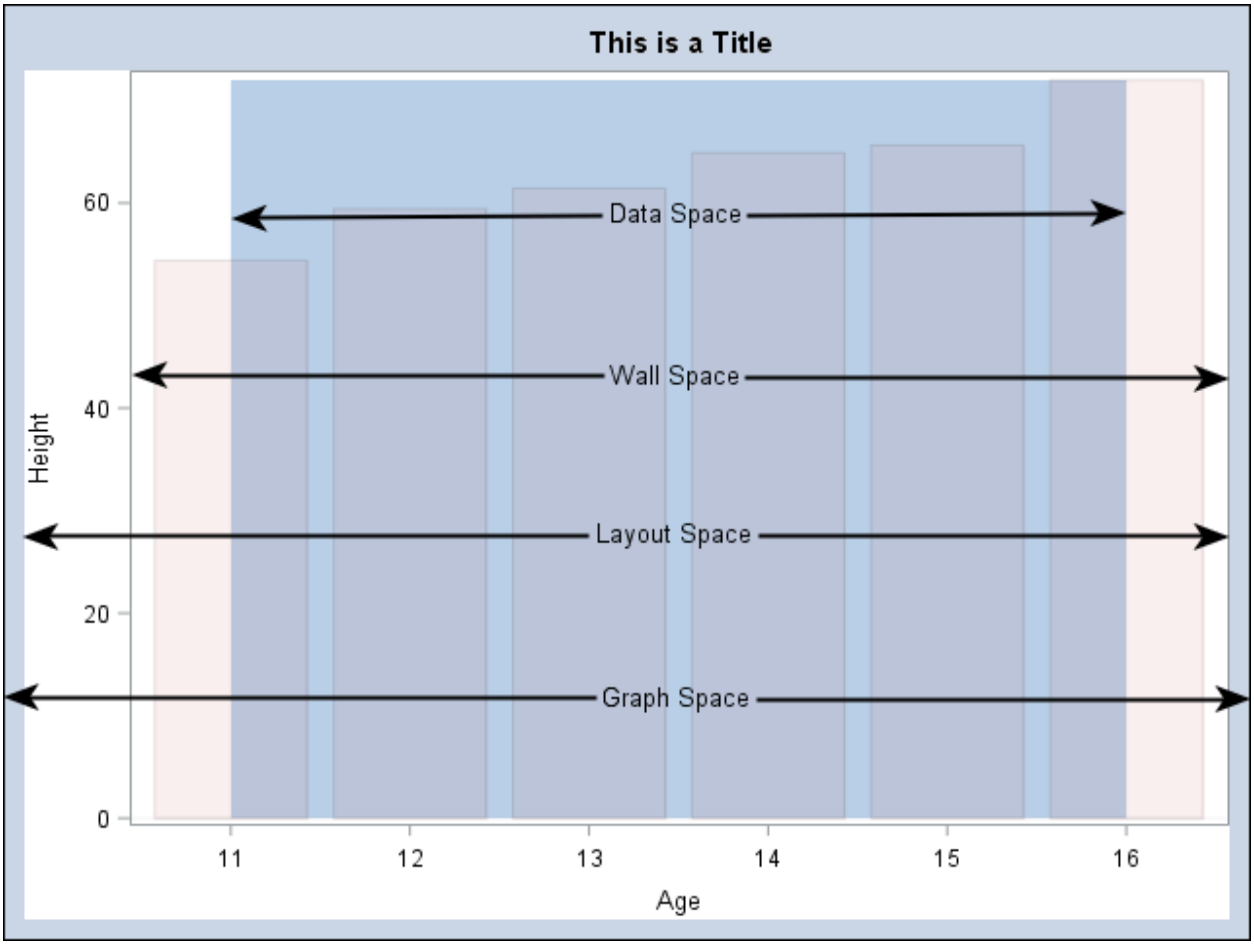

Figure 1. Drawing Spaces for a Single-Celled Graph

When using percent or pixel units, the origin for all spaces is the bottom left corner. The origin for the DATAVALUE space depends on the direction of the axes. The annotation facility allows you to specify values outside of the normal range to place annotations. For example, specifying WALLPERCENT and an X coordinate of 105% will put the annotation 5% outside of the right edge of the wall. This technique can be useful when you want to bind the placement of an exterior annotation to the position of the wall. There are some interesting examples in my previous SG procedure annotation paper involving axis-aligned tables that demonstrate this technique.

When creating multi-cell displays using LAYOUT LATTICE or LAYOUT GRIDDED (see Figure 2), the graph will contain multiple layout, wall, and data spaces. In order to address the space in each of these areas, we created a new GTL statement called ANNOTATE that binds the annotation to the layout in which it is defined. Using the code from Figure 2 as an example, the "anno1" ANNOTATE statement can address the spaces in the first cell and the "anno2" ANNOTATE statement can address the spaces in the second cell. The "anno3" ANNOTATE is defined outside of the LAYOUT OVERLAYs, but inside of the LAYOUT LATTICE. This means that the layout, wall, and data spaces of the cells are not addressable, but you can address the layout space of the LAYOUT LATTICE (The long line under "Layout Space" in Figure 2). The "Graph Space" is addressable from any location in the template.

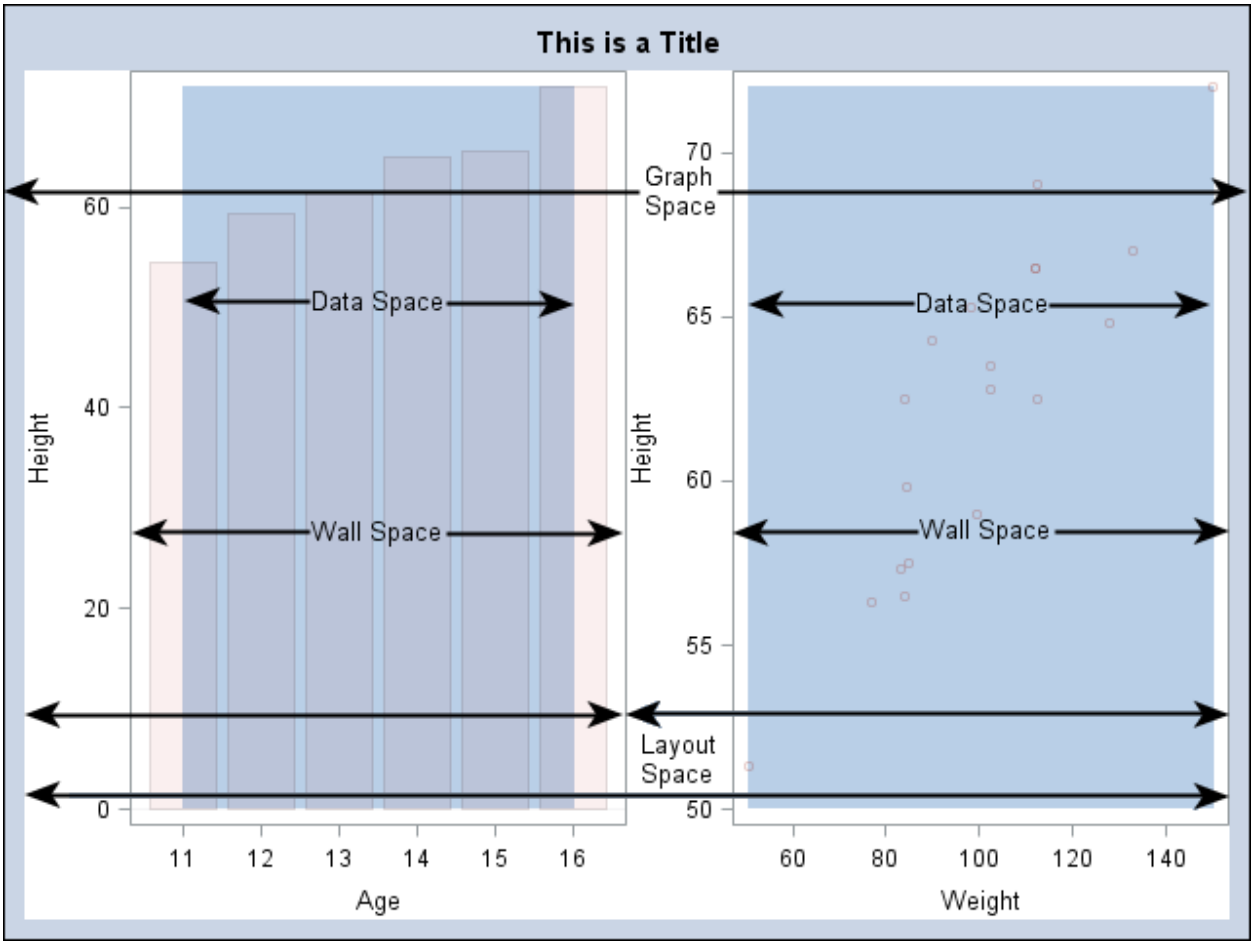

Figure 2. Drawing Spaces in a Multi-Cell Layout

```
define statgraph two_cell;
begingraph / backgroundcolor=GraphDataDefault:color;
  entrytitle "This is a Title";
  layout lattice / columns=2;
    layout overlay / opaque=true;
      bandplot x=age limitupper=72 limitlower=0;
      barchart category=age response=height / stat=mean
               datatransparency=0.9 fillattrs=graphdata2;
      annotate / id="anno1"; /* can address first cell spaces */
    endlayout;
    layout overlay / opaque=true;
      bandplot x=age limitupper=72 limitlower=50;
      scatterplot x=weight y=height / datatransparency=0.75
                  markerattrs=graphdata2;
      annotate / id="anno2"; /* can address second cell spaces */
    endlayout;
    annotate / id="anno3"; /* can address layout space of LAYOUT LATTICE */
  endlayout;
endgraph;
end;
```

The ID option in the ANNOTATION statement gives you the ability to reference specific pieces of annotation from the annotation data set. I will use this functionality in the "embedded charts"embedded chartsembedded charts example.

PADDING

It is important to note that annotations do not reserve any space on the graph – they simply draw on top of (or behind) what exists in the graph. There might be times where you want to reserve space in the graph for your annotations, such as a small table outside of the data area. Both GTL and the SG procedures have an option called PAD that can be used to create this extra space. For the SG procedures, the PAD option can control only the padding around the outside of the graph (see Figure 3), whereas GTL supports the PAD (and OUTERPAD) options on a number of graphics features. The values can be specified with dimensions such as inches, percent, or pixels.

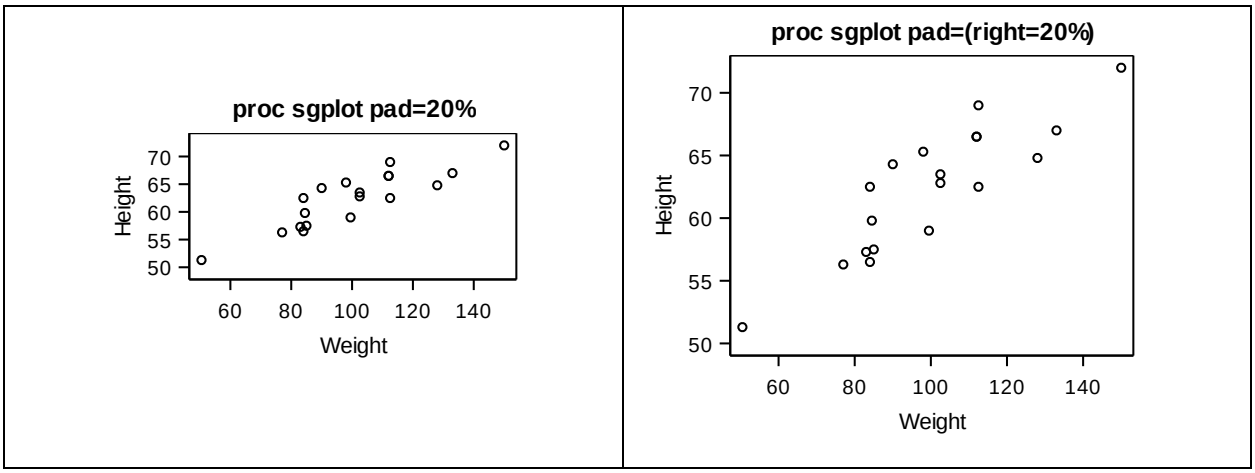

Figure 3. PROC SGPLOT with the PAD Option

The PAD option is not your only option for creating space for annotations. If you need space within the wall area, you can consider using the OFFSETMIN and OFFSETMAX options to move the endpoints of the data space away from the wall edge to create some extra space that does not collide with your plot. Also, having smaller graphical components, such as legends and entry text, inside of their own layout cells will inherently create some extra space that can be used for annotations.

FUNCTIONS

The following list (Table 1) contains the supported annotation functions as of SAS 9.4:

Function	Description
Text	Draw a text string on the graph
Textcont	Continue a text string from the Text function. This function is typically used for rich text situations where you want to change the attributes of the text somewhere in the string.
Image	Draw an image on the graph.
Line	Draw a line on the graph.
Arrow	Draw a line with an arrowhead on the graph.
Rectangle	Draw a square or rectangle on the graph.
Oval	Draw a circle or oval on the graph.
Polygon	Draw a closed polygon on the graph. The last point is automatically connected to the first point.
Polyline	Draw a multi-line figure on the graph.
Polycont	Continue a polygon or polyline. The Polygon and Polyline functions specify the starting point for the figure. The Polycont function adds an additional point per function call.

Table 1. Annotation Functions

The annotation data set definition currently has 50 reserved column names, but you will never use them all at once. Each function uses a subset of those columns for its required and optional arguments to define the annotation. Most functions are self-contained, meaning that they can be drawn using the information in one observation without any dependence on another function call. The two main exceptions to this rule are POLYGON and POLYLINE. These functions must be followed by one or more POLYCONT functions to draw anything. The TEXTCONT function can be used to extend a string from the TEXT function, but it is not required to draw a text string.

Starting at SAS 9.4, maintenance 1, we created a set annotation macros to help in the data set creation process. There is macro for each function, as well as initialization and help macros (see Table 2). I will show an example using these macros in the "custom legends" and "Annotated drilldown" examples.

Macro	Description
%SGANNO	Initializes the annotation macros. This macro must be called before you run any of the other annotation macros. This initialization is good for the lifetime of the session.
%SGANNO_HELP	Give help information for macros. Pass it a particular macro name or the keyword ALL for information about all macros.
%SGTEXT	Draw a text string on the graph
%SGTEXTCONT	Continue a text string from the Text function. This function is typically used for rich text situations where you want to change the attributes of the text somewhere in the string.
%SGIMAGE	Draw an image on the graph.
%SGLINE	Draw a line on the graph.
%SGARROW	Draw a line with an arrowhead on the graph.
%SGRECTANGLE	Draw a square or rectangle on the graph.
%SGOVAL	Draw a circle or oval on the graph.
%SGPOLYGON	Draw a closed polygon on the graph. The last point is automatically connected to the first point.
%SGPOLYLINE	Draw a multi-line figure on the graph.
%SGPOLYCONT	Continue a polygon or polyline. The Polygon and Polyline functions specify the starting point for the figure. The Polycont function adds an additional point per function call.

Table 2. Annotation Macros

When using these macros to create annotation, be careful if you intermix the macros with normal DATA step processing, as the macros can clear out retained annotation variables and create unexpected results.

EMPHASIS ANNOTATION

In this first example (Figure 5), I am adding a very common type of annotation that is used to notate and emphasize part of a plot. This notation demonstrates the use of the TEXT, TEXTCONT, and ARROW functions.

The TEXT function is probably one of the most used annotation functions. The key features of this function include the following:

- text wrapping
- text rotation
- Unicode, superscript, and subscript support
- transparency

There are several examples using these features in my previously mentioned annotation paper. In this example, I want to focus on creating rich text by combining the TEXT and TEXTCONT functions

The TEXTCONT function is used primarily to create rich text strings by changing the text attributes of a string that is already started by a TEXT function. Because of this, The TEXTCONT function takes no positional arguments. Once you start using the TEXTCONT function, you must continue to use it until you complete the string. If any other function type occurs in the data set, the string is closed and drawn.

The ARROW and LINE functions are very similar. (You will see the LINE function used later.) The main difference between them is that the ARROW function supports an additional set of columns that let you modify the arrowhead behavior. The definition for arrows and lines are self-contained, meaning that the positional information and the visual attributes are all defined in one observation. The X1, Y1, X2, and Y2 columns (or their character counterparts) must be specified for the annotation to be drawn.

The three columns used to control the arrowhead behavior are DIRECTION, SCALE, and SHAPE. The DIRECTION column is used to specify which end (or both ends) of the line has or have the arrowhead. The SCALE column is used as a scale factor for the arrowhead size. The SHAPE column is used to specify the shape of the arrowhead. The valid shapes are in Figure 4.

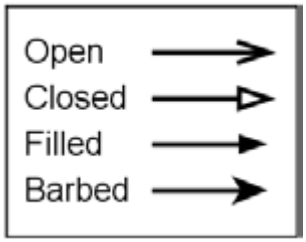

Figure 4. Arrowhead Shapes

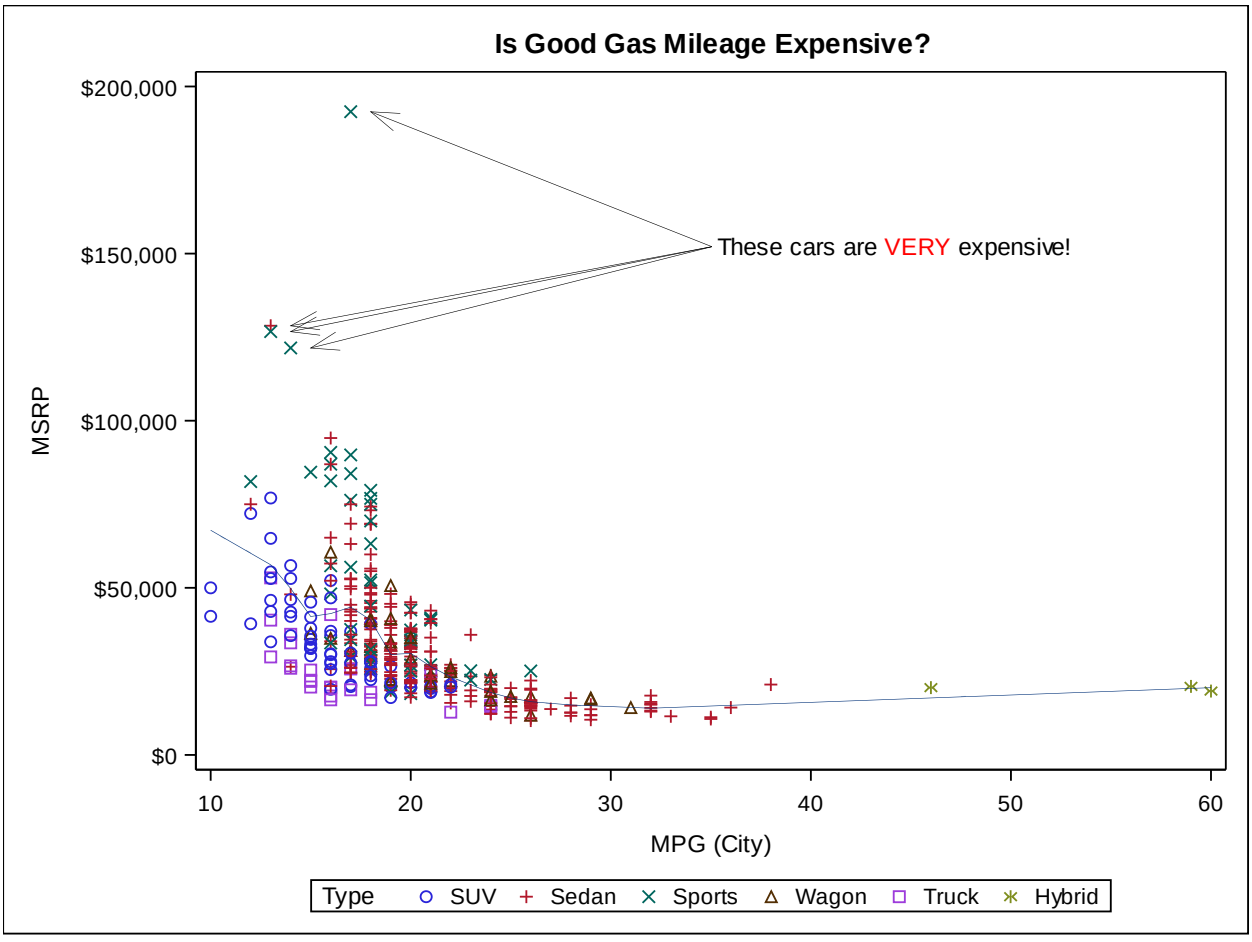

Figure 5. Emphasis Annotation

```
data expensive;
  set sashelp.cars;
  where msrp >= 100000;
run;

data anno;
  set expensive (keep=msrp mpg_city) end=_last_;
  length x1space $ 11 y1space $ 11 function $ 8 textcolor $ 5;
  retain function "Arrow" x1space "datavalue" y1space "datavalue"
         x2space "wallpercent" y2space "wallpercent" x2 50 y2 75
         direction "in" scale 0.5;
  rename msrp=y1 mpg_city=x1;
  mpg_city = mpg_city + 1; /* slightly to the right of the scatter point */
  output;
  if (_last_) then do;
    function = "text";
    x1space = "wallpercent";
    y1space = "wallpercent";
    mpg_city=50;
    msrp=75;
    anchor="left";
    width=50;
    label="These cars are";
    output;
    function = "textcont";
    label=" VERY";
    textcolor="red";
    output;
    label=" expensive!";
    textcolor="black";
    output;
  end;
run;

Title "Is Good Gas Mileage Expensive?";
proc sgplot data=sashelp.cars sganno=anno;
  scatter x=mpg_city y=msrp / group=type;
  loess x=mpg_city y=msrp / nomarkers;
run;
```

The idea here is to start from a point of origin and draw arrows out to the cars of interest. Then, label the arrows at the point of origin using rich text. The point of origin is retained in the "x2" and "y2" variable in terms of "wallpercent." (See the x2space and y2space variables.) The DATA step iterates through a data set called "expensive" that contains the cars of interest, setting the "x1" and "y1" end points of the arrow. To simplify this process, I renamed the original columns I needed to be the "x1" and "y1" needed for the annotation data set. I also moved the "x1" value slightly to prevent the arrow from sitting directly on the scatter point.

After the last observation was processed, I added the observations needed to draw the rich text. There are three observations created: one TEXT annotation and two TEXTCONT observations. The TEXT observation established four attributes about the text:

- The position of the text in terms of "wallpercent".
- The "anchor" point of the text box to attach to the position ("left", in this case). There are nine possible positions, including the center.

- The "width" of the text box. If the box is too small, the text will wrap into multiple lines.
- The initial part of the string ("These cars are")

Each of the last two TEXTCONT observations simply change the color and add another part to the string. To render the graph, I used PROC SGPLOT to create the scatterplot with loess fit, along with the annotation. This plot could have also been created using GTL, but it was very straightforward using SGPLOT syntax. The remaining examples will be created using GTL.

EMBEDDED CHARTS

This example will focus on the IMAGE function of annotation. Images can be used for adding company logos, background watermarks, or even plot symbols to a graph. In this example, I will use them to embed one graph into another.

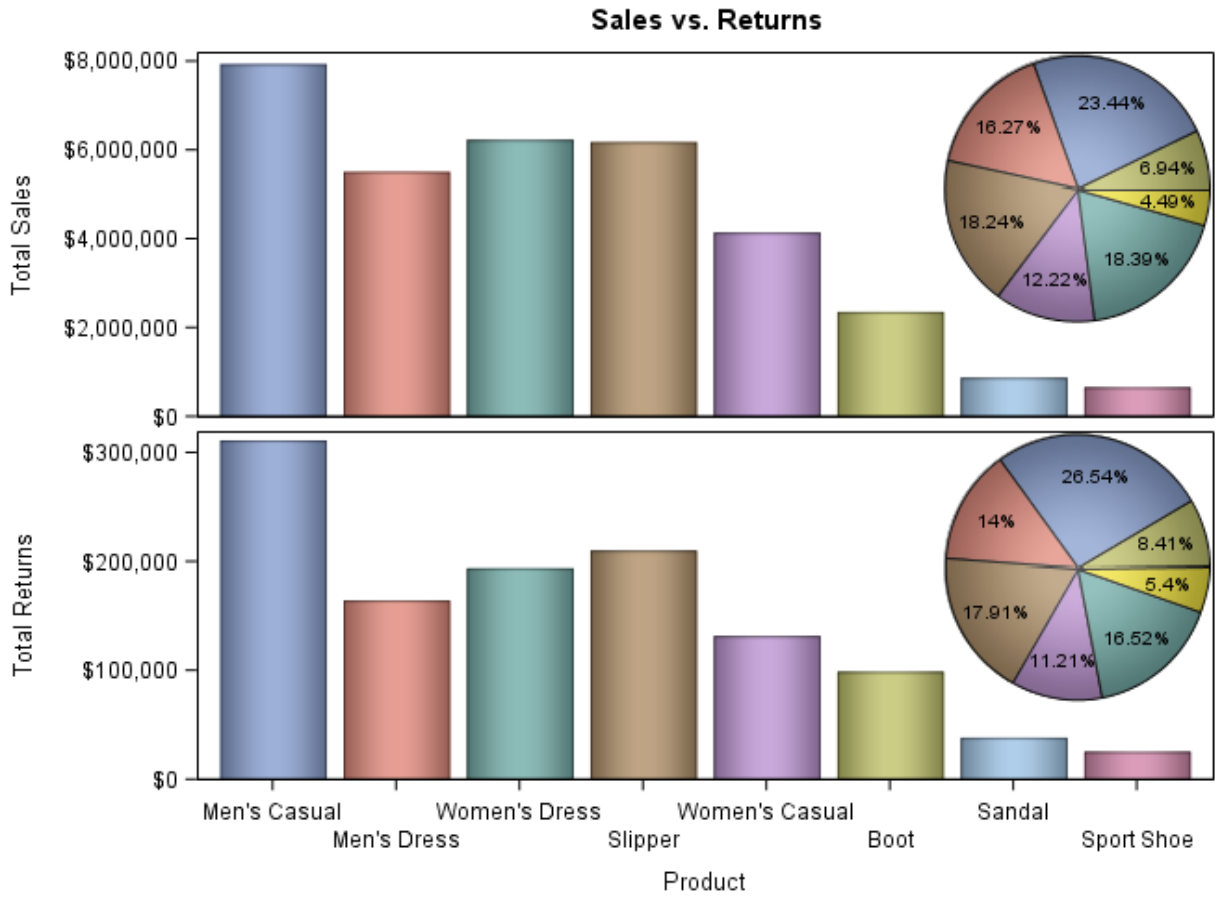

Figure 6. Embedded Charts

The first part of this process involves creating the charts to embed – in this case, a pie chart:

```
proc template;
  define statgraph pie;
    dynamic CATVAR RESPVAR;
    begingraph / opaque=false pad=0;
      layout region;
        piechart category=CATVAR response=RESPVAR / dataskin=pressed
                 datalabelcontent=(percent) datalabellocation=inside;
      endlayout;
    endgraph;
```

```
end; run;
```

One of the keys for making this graph is to make the graph background transparent. There are three options used to achieve this effect:

- OPAQUE=FALSE – By putting this option in the BEGINGRAPH statement, the graph background is made transparent.
- PAD=0 – The OPAQUE option does **not** affect the default padding around the outer edge of the graph. Therefore, you must set PAD=0 to remove the padding and make the background completely transparent.
- BORDER=off – This option in the ODS GRAPHICS statement turns off the border around the outside of the graph.

Another key for making this graph is size. You want to create the graph at the size that you want it rendered inside of the main graph. Otherwise, the text in the graph could be unreadable when you shrink it to fit. I used the ODS GRAPHICS statement to size the graphs correctly and to uniquely name the pie charts for reference from the annotation.

```
ods graphics / reset width=2in border=off imagename="pie_sales";
proc sgrender data=sashelp.shoes template=pie dattrmap=attrmap;
  dynamic catvar="product" respvar="sales";
  dattrvar product="shoes";
run;
ods graphics / imagename="pie_returns";
proc sgrender data=sashelp.shoes template=pie dattrmap=attrmap;
  dynamic catvar="product" respvar="returns";
  dattrvar product="shoes";
run;
```

Notice that the DATTRMAP option is used for both pie charts. The attributes map is needed for all graphs in this composite so that the correct color is guaranteed to be associated with the correct categories. Here is the code used to create the attributes map:

```
data attrmap;
retain id "shoes" linecolor "black";
length fillstyleelement $ 10;
input value $ 1-14 fillstyleelement $;
cards;
Men's Casual   graphdata1
Men's Dress    graphdata2
Women's Dress  graphdata3
Slipper        graphdata4
Women's Casual graphdata5
Boot           graphdata6
Sandal         graphdata7
Sport Shoe     graphdata8
;
run;
```

I used style references instead of literal fill colors because I was only interested in correct attribute assignment, not the value of the attributes. By using style references, the graph attributes can change if the ODS style is changed, but I'm still guaranteed consistent assignment.

For this annotation, I needed to put two images into two different cells. The easiest way to do this is to bind each image to the correct cell by using the ID option of the ANNOTATE statement. The following code is the template for the gridded bar chart. Notice how the ANNOTATE statement is specified in each cell.

```
proc template;
  define statgraph bar;
    dynamic CATVAR1 RESPVAR1 CATVAR2 RESPVAR2;
    begingraph;
      layout lattice / columns=1 rowweights=(.45 .55);
        layout overlay / xaxisopts=(display=none);
          barchart category=CATVAR1 response=RESPVAR1 / group=CATVAR1
                   dataskin=pressed;
          annotate / id="Sales";
        endlayout;
        layout overlay / xaxisopts=(display=none);
          barchart category=CATVAR2 response=RESPVAR2 / group=CATVAR2
                   dataskin=pressed;
          annotate / id="Returns";
        endlayout;
      endlayout;
    endgraph;
  end;
run;
```

The code for the annotation data set retains most of the same value for both images. The only difference is for the ID and IMAGE columns:

```
data anno;
retain function "image" width 35 x1 104 y1 99 anchor "topright"
       drawspace "wallpercent";
length id $ 7 image $ 15;
input id $ image $;
cards;
Sales   pie_sales.png
Returns pie_returns.png;
```

The ANCHOR column works the same way with images as it does with text boxes. To make the placement easier, I set the ANCHOR to be in the "topright" because I wanted the pie charts in the top right corner of each cell. I also specified only the width so that the system would render the pie chart using the aspect ratio of the image.

Finally, the composite graph is create by using SGRENDER with the bar chart template and the annotation data set. I used PROC SORT to sort the raw data in descending order to help make room for the embedded graph. Even with the sort, the style attributes are correctly assigned because of the attributes map.

```
ods graphics / width=7in imagename="bar_report";
proc sort data=sashelp.shoes out=shoes; by descending sales; run;
proc sgrender data=shoes template=bar dattrmap=attrmap sganno=anno;
  dynamic catvar1="product" respvar1="sales"
          catvar2="product" respvar2="returns";
  dattrvar product="shoes";
run;
```

CUSTOM LEGENDS

In this example, I used the RECTANGLE and OVAL functions to create a custom size legend to coexist with a standard discrete legend outside of the plot area. There are a couple of ways to reserve space for the custom legend:

- Combine the space created for the discrete legend with some extra bottom padding to create enough space for the size legend.
- Put the discrete legend and the custom legend in their own LAYOUT LATTICE cell and use ROWWEIGHTS to control the amount of space to reserve. I used this technique for this example.

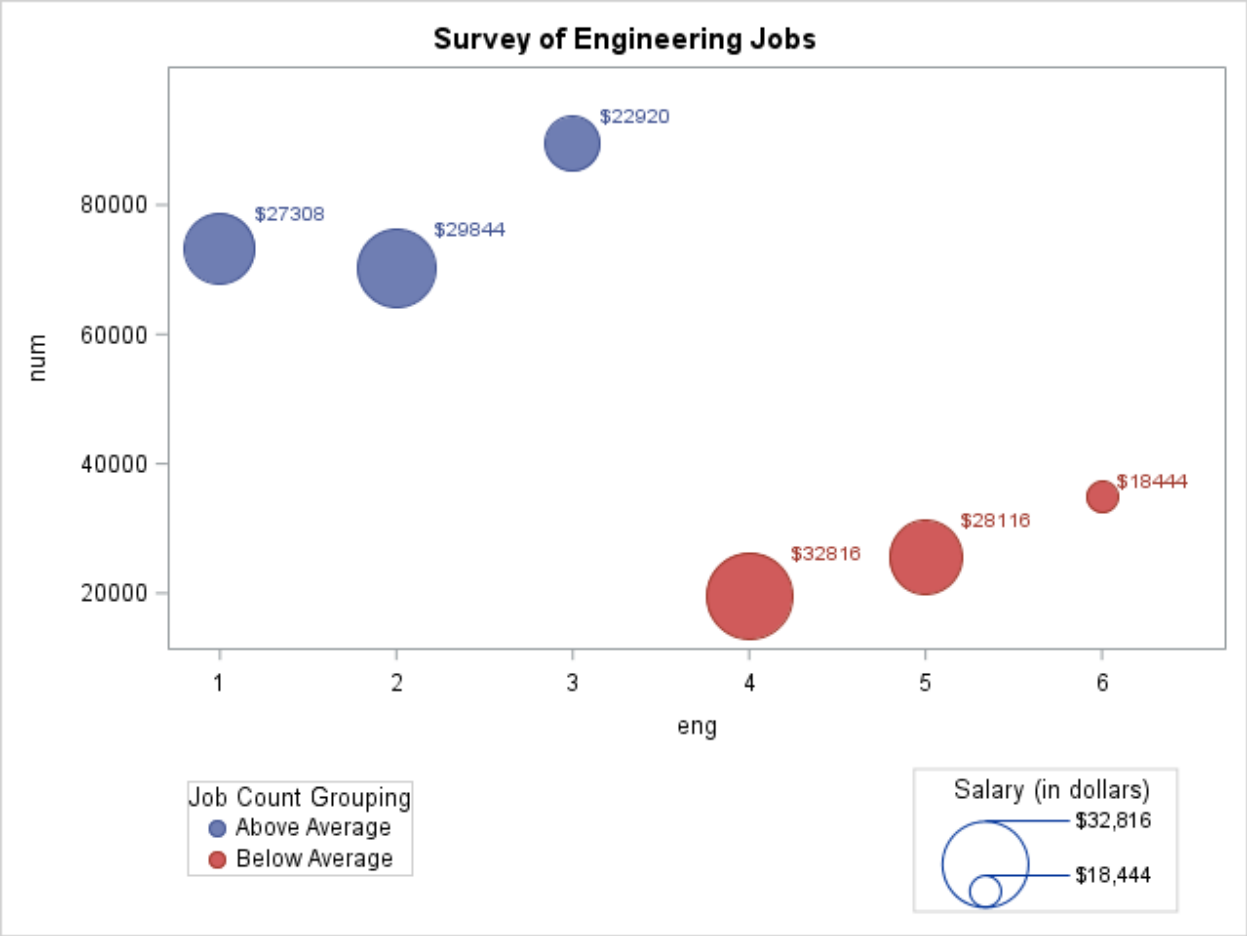

Figure 7. Bubble Plot with Custom Legend

```
%SGANNO;
data anno;
  %SGRECTANGLE(drawspace="layoutpercent",widthunit="pixel",x1=85,y1=0,
               heightunit="pixel",anchor="bottom",width=135,height=73,
               display="all",linethickness=1,fillcolor="white",
               linecolor="GraphBorderLines:contrastcolor");
  %SGOVAL(drawspace="layoutpercent",widthunit="pixel", x1=80,y1=28,
          heightunit="pixel",width=44,height=44,display="outline",
          linethickness=1,linecolor="GraphDataDefault:contrastcolor");
  %SGOVAL(drawspace="layoutpercent",widthunit="pixel", x1=80,y1=12,
          heightunit="pixel",width=16,height=16,display="outline",
          linethickness=1);
  %SGLINE(drawspace="layoutpercent",x1=80,y1=54,x2=87,y2=54,
          linethickness=1);
  %SGLINE(drawspace="layoutpercent",x1=80,y1=21.5,x2=87,y2=21.5,
          linethickness=1);
  %SGTEXT(drawspace="layoutpercent",widthunit="pixel",x1=85.5,y1=84,
          anchor="top",width=140,textsize=10,label="Salary (in dollars)");
```

```
%SGTEXT(drawspace="layoutpercent",widthunit="pixel",x1=87,y1=54,
        anchor="left",width=140,textsize=8,label="$32,816");
%SGTEXT(drawspace="layoutpercent",widthunit="pixel",x1=87,y1=21.5,
        anchor="left",width=140,textsize=8,label="$18,444");
```

Instead of using standard DATA step coding to create the annotation data set, I used the SG annotation macros instead. When you use a variety of functions and their options, it can create a lot of "holes" in your input data lines and make it difficult to line up your data correctly. These macros can make it easier to visualize the annotation functions and the arguments you are using. Be sure to specify the %SGANNO macro first to initialize the macros before you start using them.

Notice that the size specification for both the RECTANGLE and OVAL functions are in terms of width and height. For the oval shape, the width and height determine the size of the bounding box, and the oval is effectively drawn within the bounding box. Therefore, if you want a circle (as in this case), the width and height should be the same. Rectangles and ovals can be filled or unfilled, and there are various other options to control their appearance.

In the BUBBLEPLOT statement below, the BUBBLERADIUSMIN and BUBBLERADIUSMAX are set to 8 pixels and 22 pixels, respectively. To synchronize the size of the annotated bubbles with the sizes used by the bubble plot, I set the BUBBLERADIUSMIN and BUBBLERADIUSMAX (the radius) to be half of the annotation width and height (the diameter).

```
proc template;
  define statgraph pie;
    dynamic CATVAR RESPVAR;
    begingraph;
      entrytitle "Survey of Engineering Jobs";
      layout lattice / rowweights=(.8 .2);
        layout overlay;
          bubbleplot y=Num x=Eng size=Dollars / datalabel=dollars
                     bubbleradiusmax=22px bubbleradiusmin=8px
                     group=avggrp name="bubble";
        endlayout;
        layout overlay;
          discretelegend "bubble" / halign=left title="Job Count Grouping"
                         outerpad=(left=85px);
          annotate;
        endlayout;
      endlayout;
    endgraph;
  end;
run;
```

This template defines two cells – the first contains the plot and the second contains the two legends. I used the ROWWEIGHTS option to give 80% of the layout height to the bubble plot and 20% to the legends.

In the legend cell, I wanted the discrete legend on the left and the size legend on the right. I also wanted both legends to be under the plot area. To create this layout, I used HALIGN=LEFT on the discrete legend to move it to the left side of the cell. However, the cell boundaries are wider than the plot walls, so I needed to bring the legend back to the right a little to get it under the plot area. To do this, I used the OUTERPAD option to add left padding to the legend and push it back to the right. By using the OUTERPAD option instead of the PAD option, the padding was added outside of the legend border area instead of inside the legend border area.

Notice that the ANNOTATE statement has no arguments. When the ANNOTATE statement is specified this way, all annotations from the data set are rendered and are bound to the layout that contains the ANNOTATE statement. The annotations were defined to draw the size legend to the right side of the cell.

ANNOTATED DRILLDOWN

For this final example, I wanted to use polygon-based annotations to create a "drilldown" graph, with a contour plot (left) of the density values represented by the boxed observations in the heat map (right).

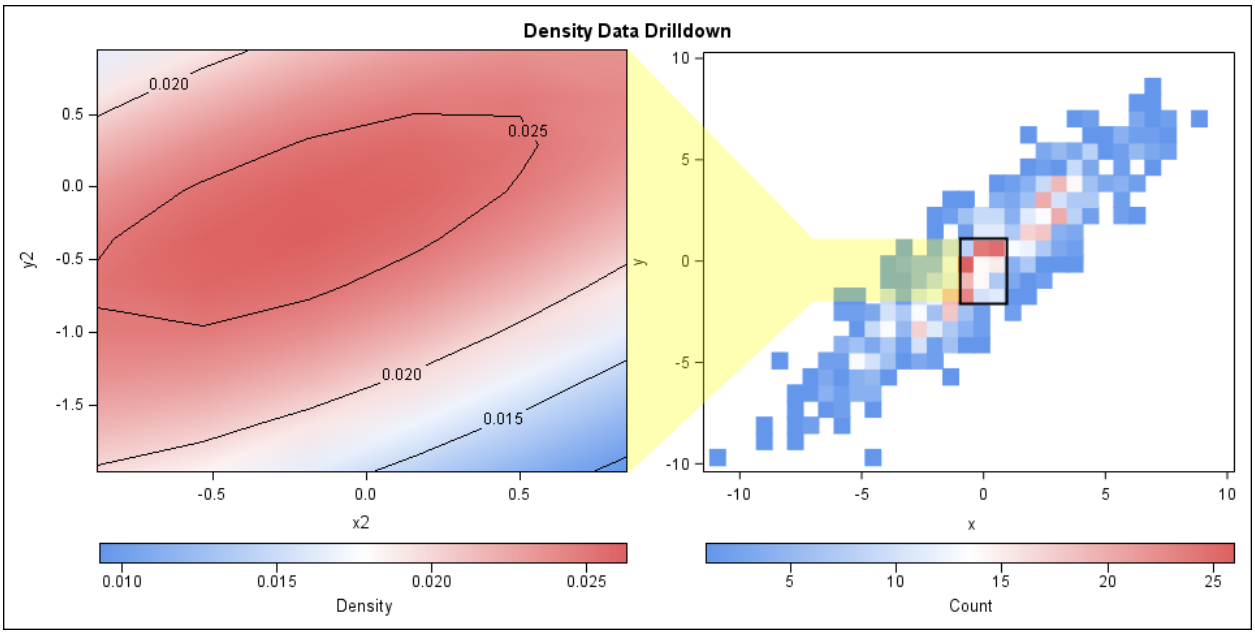

Figure 8. Drilldown Graph

One of the biggest keys for creating a graph like this is data preparation. After finding an area of interest in your plot, you need to determine the data range of that area and extract those observations into a separate data set. Then, rename the columns in the subset data set and merge it with the original data into a new data set. This merged data set is the one you will use to create the graph.

```
proc template;
  define statgraph drilldown;
    begingraph;
      entrytitle "Density Data Drilldown";
      layout lattice / columns=2 rowdatarange=data;
        layout gridded;
          layout overlay / xaxisopts=(offsetmin=0 offsetmax=0
                           linearopts=(thresholdmin=0 thresholdmax=0))
                           yaxisopts=(offsetmin=0 offsetmax=0
                           linearopts=(thresholdmin=0 thresholdmax=0));
            contourplotparm x=x2 y=y2 z=density2 / name="contour";
            continuouslegend "contour" / orient=horizontal location=outside
                             halign=center valign=bottom title="Density";
          endlayout;
        endlayout;
        layout overlay;
          heatmap x=x y=y / colorresponse=count xbinaxis=false
                  ybinaxis=false colorstat=sum name="heatmap";
          continuouslegend "heatmap" / orient=horizontal location=outside
                           halign=center valign=bottom title="Count";
          annotate;
        endlayout;
      endlayout;
    endgraph;
  end; run;
```

Without any annotation, this template produces a standard-looking gridded graph.

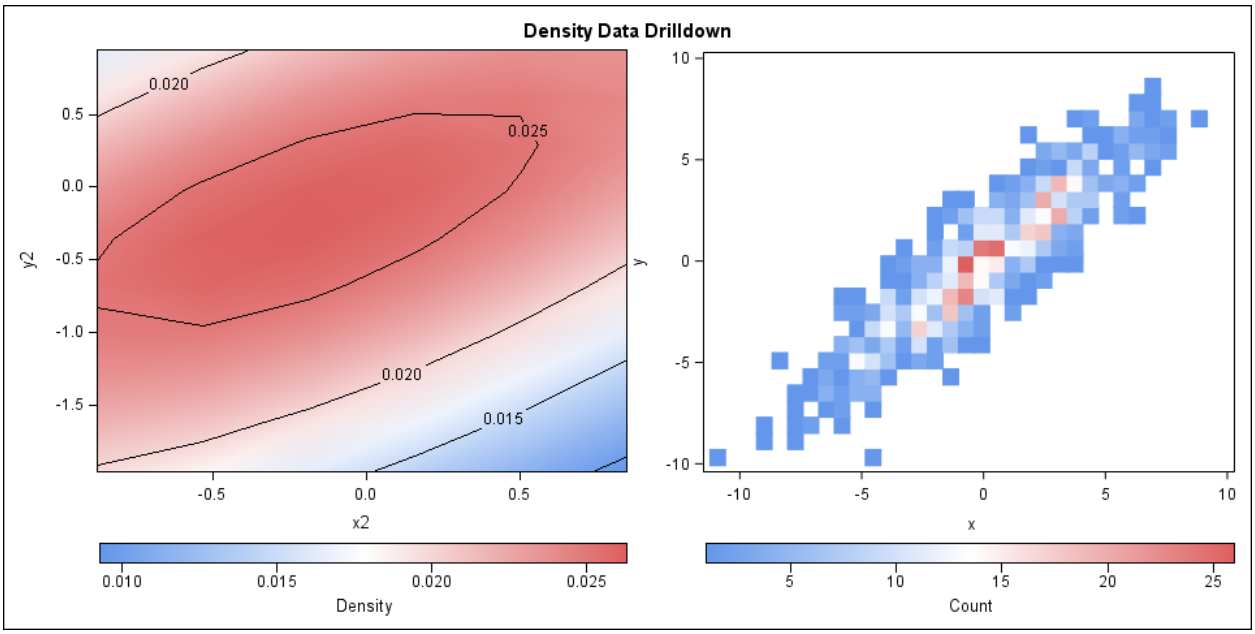

Figure 9. Drilldown Graph with No Annotation

The annotation for the drilldown effect is a combination of a rectangle and a filled polygon: The rectangle surrounds the data of interest in the heat map and is specified in data coordinates. The polygon annotation has to draw outside of the second cell area, so the drawing space changes for different segments of the polygon. Polygon annotations automatically close the last point to the first point, which is why I did not add a closing observation. Since some of these annotations use data coordinates from the second cell, the ANNOTATE statement is added to the second cell of the graph template.

```
%SGANNO;
data anno;
%SGRECTANGLE(drawspace="datavalue",x1=0,y1=-0.5,width=9,height=16);
%SGPOLYGON(x1space="graphpercent",y1space="layoutpercent",x1=50,y1=100,
           display="fill",filltransparency=0.7,fillcolor="yellow");
%SGPOLYCONT(x1space="datavalue",y1space="datavalue",x1=-7,y1=1.1);
%SGPOLYCONT(x1space="datavalue",y1space="datavalue",x1=-1,y1=1.1);
%SGPOLYCONT(x1space="datavalue",y1space="datavalue",x1=-1,y1=-2);
%SGPOLYCONT(x1space="datavalue",y1space="datavalue",x1=-7,y1=-2);
%SGPOLYCONT(x1space="graphpercent",y1space="layoutpercent",x1=50,y1=25);
```

The %SGPOLYGON macro defines the polygon as being fill-only (no outline), with a yellow color at 70% transparency. The polygon starts at the top right corner of the first cell, works over to the rectangle, runs down the rectangle, and works back over to the bottom right corner of the first cell.

Finally, to keep the plots in each cell at a good aspect, I increased the width of the graph. I also had to specify the height of the graph, else the ODS Graphics system would have also grown the height of the graph to maintain the default aspect with the width.

```
ods graphics / width=960px height=480px;
proc sgrender data=merged template=drilldown;
run;
```

CONCLUSION

The goal of this paper has been to give you a good overview of the ODS Graphics annotation facility and show you some examples of how the facility can be used. It is worth noting, however, that "annotations" can be created using standard graphics syntax. For example, if the annotation is completely in data space, consider using the POLYGON plot instead. That plot will give you the ability to draw polygons in data space while maintaining the characteristics of a regular plot. Also, creative use of ENTRY statements and LAYOUT GRIDDED statements can provide the ability to add text annotations to your graph. I encourage you to explore the entire ODS Graphics system and discover all of its capabilities.

REFERENCES

Heath, Dan. 2011. “Now You Can Annotate Your Statistical Graphics Procedure Graphs.” Proceedings of the SAS Global 2011 Conference, Las Vegas, NV: SAS Institute, Inc., Available at http://support.sas.com/resources/papers/proceedings11/TOC.html.

CONTACT INFORMATION

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

Dan Heath
SAS Institute Inc.
100 SAS Campus Drive
Cary, NC 27513
Dan.Heath@sas.com
http://www.sas.com