

## Paper 324-2009

## Secrets of the SG Procedures

Dan Heath, SAS Institute Inc., Cary, NC

### ABSTRACT

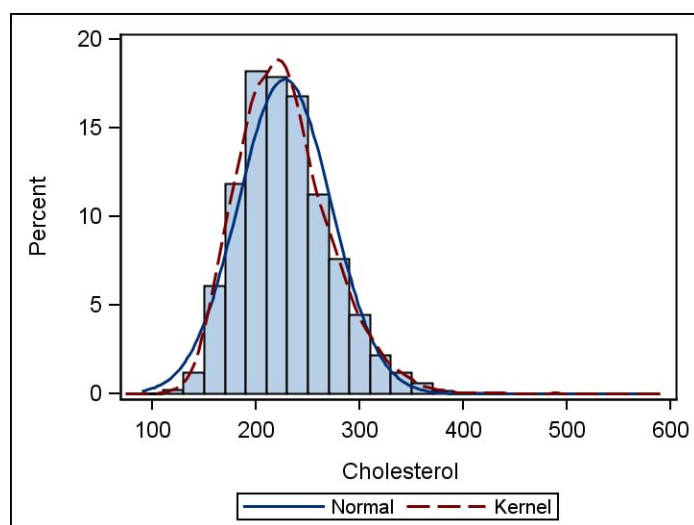
The SAS/GRAPH® SG procedures provide an extensive set of plots and supporting statements to create common graphs. However, there are creative ways to combine the statements and options in these procedures that might not be obvious. This presentation will reveal some of these techniques, enabling you to create many of the specialized graphs you need in your industry.

These are some of the techniques that will be discussed:

- creating an area plot from a band plot
- displaying your custom limit calculations
- using a vector plot for event and high-low types of plots
- using paneling options for group axis effects

### INTRODUCTION

The SAS/GRAPH SG procedures provide an extensive set of plot statements and supporting statements to create common graphs. This is particularly true of the SGPLOT and SGPANEL procedures, where plot statements can be combined to create more complex plots and charts. For example, the distribution plot in Figure 1 was created by combining a histogram with two density plots. Plot statements in these procedures are drawn in the order they are specified, which is why the histogram is specified before the density plot statements.



```
proc sgplot data=sashelp.heart;  
  histogram cholesterol;  
  density cholesterol;  
  density cholesterol / type=kernel;  
run;
```

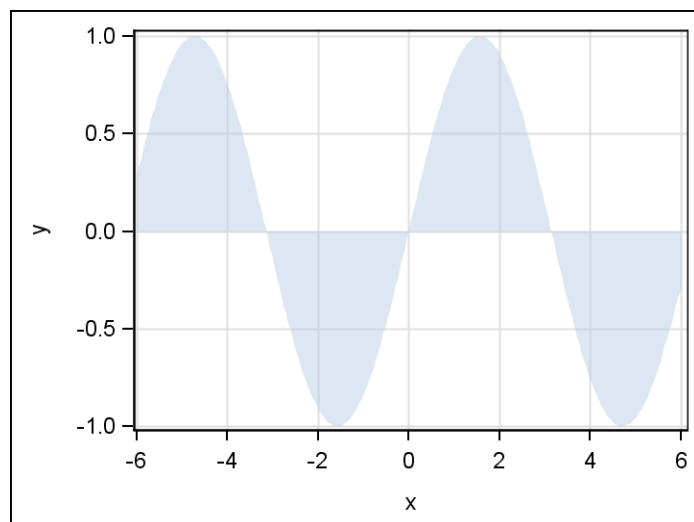
**Figure 1. Histogram with Density Plot Overlays**

One of the things you will discover in this presentation is that you can use different plot overlays in non-standard ways to enhance the primary plots, such as for annotation or axis reference. In addition, some of these plot statements can be used in ways that seem different from their standard usage, such as using a band plot to create an area plot. Most of the options and techniques discussed in this presentation apply to both the SAS 9.2 Phase 1 and SAS 9.2 Phase 2 releases of SAS/GRAPH; however, SAS 9.2 Phase 2-specific options will be noted as they are used.

### THE FLEXIBLE BAND PLOT

The band plot is an interesting construct in the SG procedures. Typically, users would display the upper and lower confidence limits of a series by using two additional series overlays. Because these additional series were independent plots, they had no knowledge of one another. The band plot construct combines both limits into the same statement, making them part of the same plot. Because the limits are contained in the same plot, they can be represented as a filled area or as two series lines. Attributes such as line attributes, fill attributes, and transparency are applied equally to both limits. The limits in the band plot can be specified as either constant values or as variables. This flexibility enables you to use bands in many creative ways.

An example of a creative use is to create an area plot. In an area plot, the threshold of the area is set to a constant Y axis value (typically, zero), while the Y axis response value varies along the X axis. Figure 2 shows a simple example of how you can use a band plot to draw an area plot.



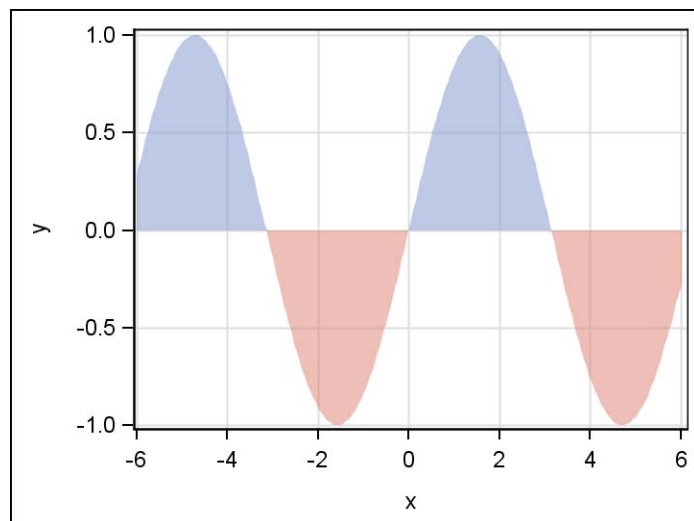
```
data wave;
  do x=-6 to 6 by .05;
    y=sin(x);
    output;
  end;
run;

proc sgplot data=wave;
  yaxis grid;
  xaxis grid;
  band x=x lower=0 upper=y /
    transparency=0.5;
run;
```

**Figure 2. Sine Wave Using a BAND Plot**

Notice that the lower limit is set to a constant threshold value. Even though the upper limit goes above and below the lower limit, the band is still filled appropriately.

It is also possible to color the positive and negative regions differently. You do this by overlaying two band plots—one for the positive data and one for the negative data (Figure 3). Because there are two band statements, you can control the visual attributes of your positive and negative data independently.



```
%let THRESHOLD = 0;
data wave;
  do x=-6 to 6 by .05;
    y=sin(x);
    if (y >= &THRESHOLD) then do;
      positive = y;
      negative = &THRESHOLD;
    end;
    else do;
      positive = &THRESHOLD;
      negative = y;
    end;
    output;
  end;
run;

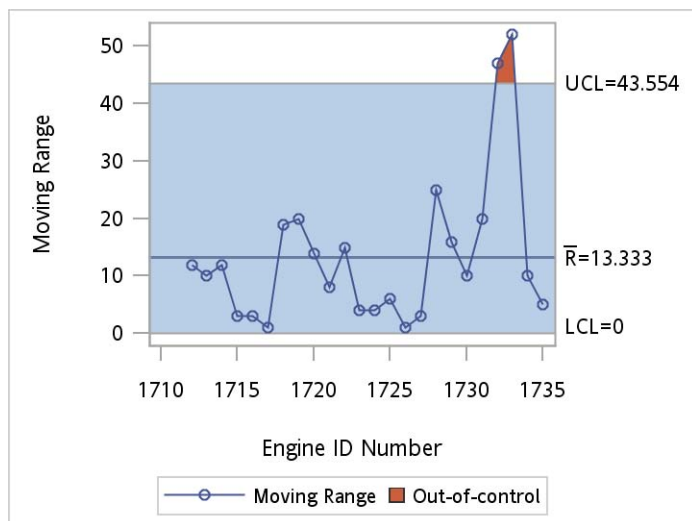
proc sgplot data=wave noautolegend;
  yaxis grid label="y";
  xaxis grid;
  band x=x lower=&THRESHOLD
    upper=positive / transparency=0.5;
  band x=x lower=negative
    upper=&THRESHOLD /
    transparency=0.5;
run;
```

**Figure 3. Different Attributes for Positive and Negative Areas**

The key to Figure 3 is in the data construction. The Y value is added to the appropriate positive or negative column, while the threshold value is added to the opposite column. The threshold value is needed because missing values are ignored by the band plot, and the band plot will try to connect all of the nonmissing values. Therefore, not having the threshold values in the data could result in the area being drawn incorrectly.

Constant values can be specified for both the upper and lower limits to create a reference region instead of using two reference lines. Creating a reference region can be a very effective way to highlight good and bad ranges. These regions can also be used creatively to obscure good points and emphasize points outside of the limits. In Figure 4,

the area concept in Figure 2 is combined with a reference region to create a simple range chart that highlights out-of-control points.



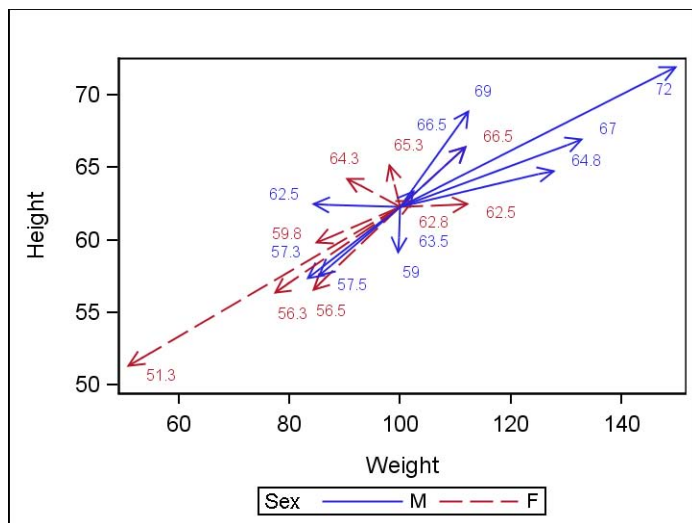
```
ods escapechar='~';
proc sgplot data=control;
  band upper=_subr_ lower=13.333
    x=id / fillattrs=GraphError
    legendlabel="Out-of-control"
    name="error";
  band x=id upper=43.554 lower=0 /
    fillattrs=GraphConfidence;
  refline 13.333 /
    label="R~{unicode bar}=13.333"
    lineattrs=GraphFit;
  refline 43.554 /
    label="UCL=43.554";
  refline 0 / label="LCL=0";
  series x=id y=_subr_ / markers
    name="moving";
  keylegend "moving" "error";
run;
```

**Figure 4. A Simple Range Chart**

The key technique for creating a plot such as Figure 4 is layering the plot correctly. Notice that the first band is drawn using the same data as the series plot. The first band is rendering the series data as an area plot with a threshold at 13.333. Next, a second band is specified for the confidence limits that are drawn on top of the first band, obscuring any data that is inside the confidence limits. Finally, the series plot is drawn on top of the confidence band so that all of the moving range data can be seen.

## INTRODUCING THE VECTOR PLOT

The vector plot is another plot that has many uses beyond its original intent. Basically, the vector plot draws line segments, with or without arrowheads, either from a constant origin (such as (0, 0)) or from a variable-based origin. In Figure 5, the origin is set to be the mean of the height and weight values, and the length of the vectors shows the distance from the mean.



```
proc sgplot data=sashelp.class;
  vector x=weight y=height /
    xorigin=100 yorigin=62.3
    group=sex datalabel;
run;
```

**Figure 5. A Simple Vector Plot**

As with the limits in a band plot, the X and Y origin specification can be a combination of constant and variable values. This functionality enables you to draw disconnected line segments anywhere you need them within the axis area. For example, you can use the vectors to represent high and low ranges for a large volume of data, such as in a high-low-close plot (Figure 6). Notice that the arrowheads are turned off so that only the line segment is used to represent the intra-day high and low values for the stock. A scatter plot is overlaid on the vector plot to represent the close value, and the band and series plots display additional moving average information.

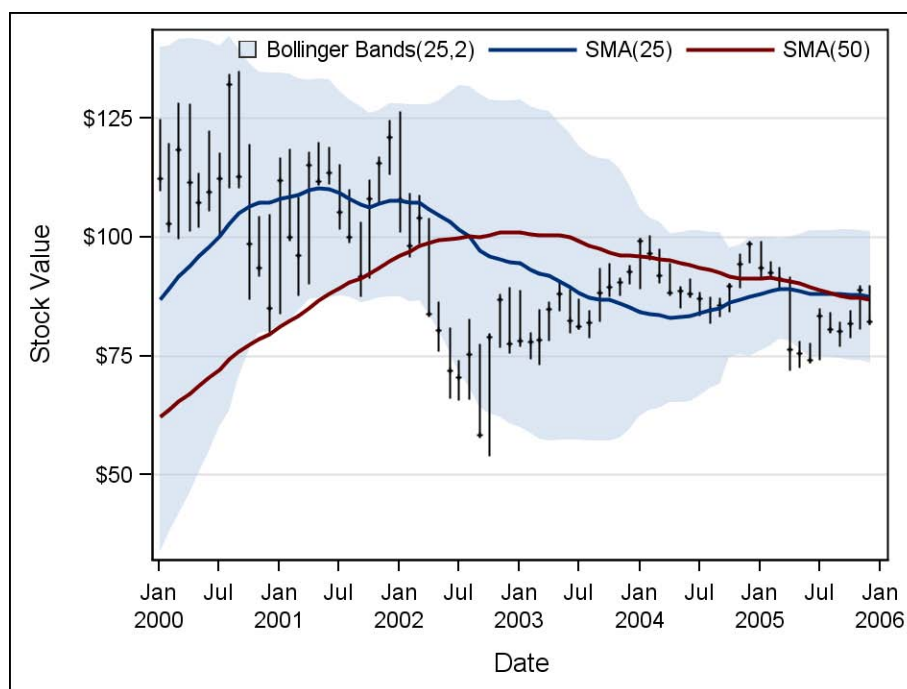


Figure 6. High-Low-Close Plot

```
proc sgplot data=moveavg;
  yaxis grid label="Stock Value";
  band x=date upper=bolupper lower=bollower /
    transparency=0.5 legendlabel="Bollinger Bands(25,2)" name="boll";
  vector x=date y=high / yorigin=low xorigin=date noarrowheads;
  scatter x=date y=close / markerattrs=(symbol=plus size=3);
  series x=date y=avg25 / lineattrs=GraphFit legendlabel="SMA(25)" name="d25";
  series x=date y=avg50 / lineattrs=GraphFit2(pattern=solid)
    legendlabel="SMA(50)" name="d50";
  keylegend "boll" "d25" "d50" / across=4 noborder position=TopRight
    location=inside;
run;
```

The key to using vector plots in this way is to think about your data as a starting point drawing to an ending point. The XORIGIN/YORIGIN options represent your starting point, while the X/Y options are your ending point. In Figure 6, all of the lines are drawn perpendicular to the X axis, which is why the XORIGIN and X options are using the same variable (date). Because the YORIGIN option is using the low variable and Y is using the high variable, the plot is drawing from the low value up to the high value for each date value.

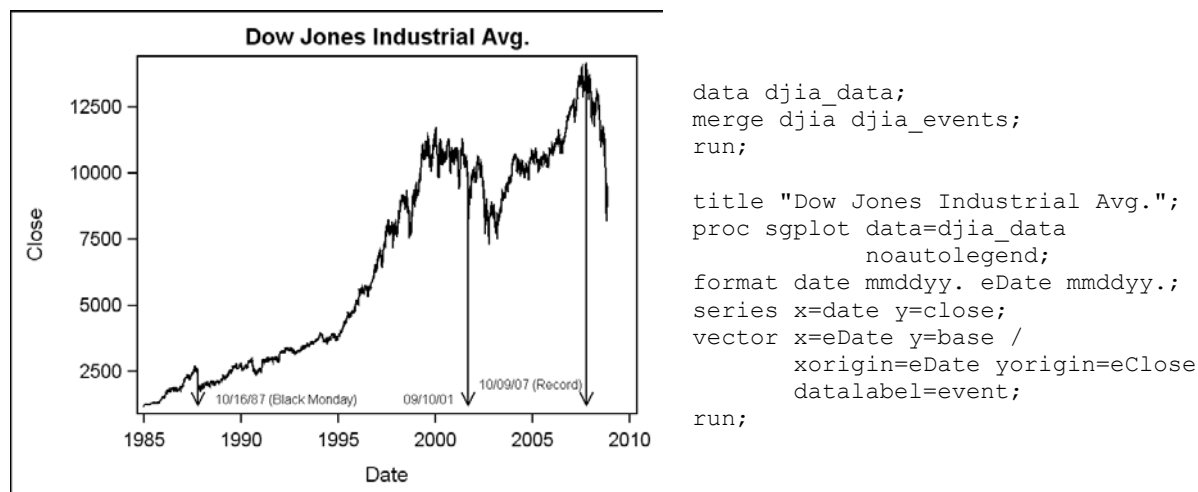


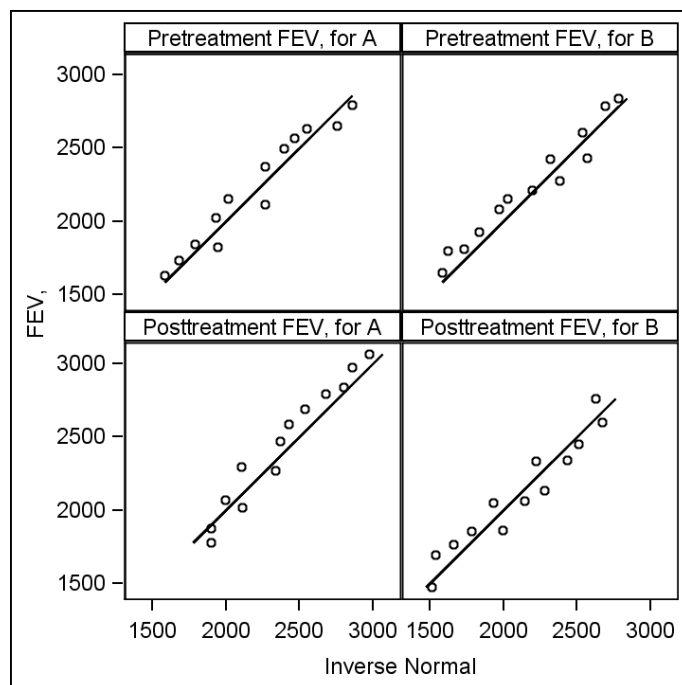
Figure 7. Event Plot of Stock Data

Vector plots can also be used to mark events along a series plot. In Figure 7, the Dow Jones Industrial Average is plotted from January 1, 1985 to November 07, 2008. A vector plot is overlaid on the series plot to mark key events that affected the average. The key to creating a plot of this type is the data organization. The original stock data is contained in the DJIA data set, while a second data set is created to hold the data for the events (DJIA\_Events). The two data sets are then merged, and the resulting data set is used in the procedure. The event data set would look something like the following:

```
data djia_events;
informat eDate mmddyy.;
input eDate eClose event $ 18-41 base;
cards;
10/09/07 14164.53 10/09/07 (Record)          1184.96
09/10/01  9605.51 09/10/01                  1184.96
10/16/87  2246.73 10/16/87 (Black Monday) 1184.96
;
```

The Base column contains the minimum close value from the DJIA data set, which is the end point for the vectors. It is important to note for this example that data labels are drawn on the ends of the vectors instead of on the origins. This is why the origins of the vectors were set to be along the series line instead of the base line.

Vector plots can also be used for special reference lines. In Figure 8, a vector plot is used to create Y=X reference lines for an inverse normal plot. For this example, a reference data set is created that contains the minimum and maximum values for each FEV test. This data set is then match-merged with the original data set to create the data set used by PROC SG PANEL. The vector plot uses the Min and Max columns to draw the reference lines.



```
data breath;
merge breath_data reference;
by test;
run;

proc sgpanel data=breath noautolegend;
format test testtype.;
panelby test / novarname;
vector y=max x=max / xorigin=min
      yorigin=min noarrowheads;
scatter x=inverse y=fev;
run;
```

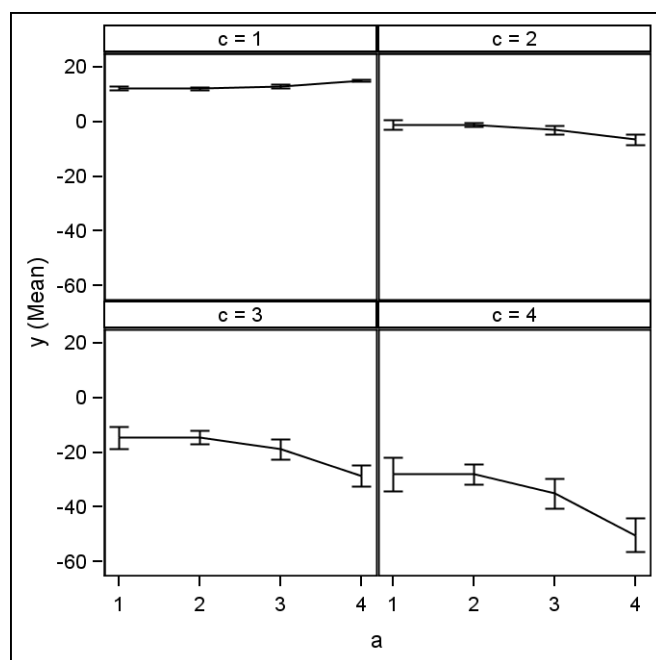
Figure 8. Inverse Normal Plot

As a final note, the vector plot is part of the Graph Template Language (GTL) in all versions of SAS 9.2; however, it was not surfaced in the SG procedures until the SAS 9.2 Phase 2 release.

## DISPLAYING CUSTOM LIMITS

The SGPLOT and SG PANEL procedures have built-in support to compute limits for bar charts, line charts, and dot plots. Users can set the statistic for the limit to be the confidence limit of the mean (CLM), standard error, or standard deviation. Users can also set the alpha value for the CLM, as well as a multiplier for the standard error or standard deviation (for example, two standard errors). Figure 9 shows an example of a paneled mean plot using a standard VLINE statement.

There might be situations, however, when a user needs to compute and display custom limits. The bar chart, line chart, and dot plot only display limits computed within the procedure. Fortunately, you can construct these types of displays using other plot types within the procedures.

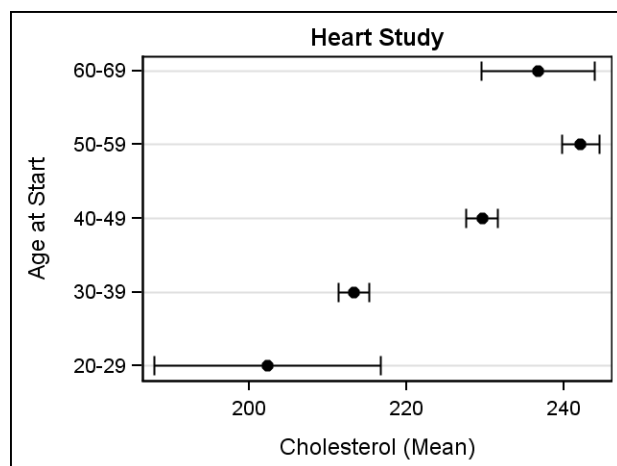


```
proc sgpanel data=sim noautolegend;
  panelby c;
  vline a / response=y stat=mean
           limitstat=clm;
run;
```

**Figure 9. Mean Plot Using a VLINE Statement**

The following examples demonstrate how you can create equivalent dot plots, line charts, and bar charts using alternate plot types and custom limits. The data should be pre-summarized with the custom limits contained in the correct observations.

Figure 10 demonstrates how you can use a standard SCATTER statement to create a dot plot. The SCATTER statement supports both X and Y error limits that are assigned to columns. These options can be used to display any pre-computed limit information.

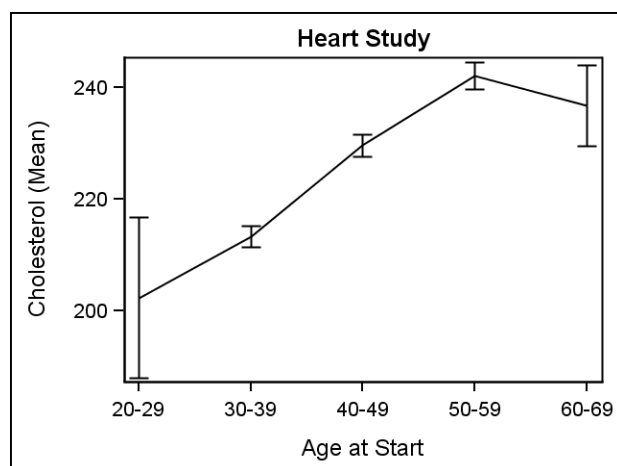


```
Title "Heart Study";
proc sgplot data=precomputed;
  yaxis grid type=discrete;
  scatter y=AgeAtStart x=mean_cholesterol /
    markerattrs=(symbol=circlefilled)
    xerrorlower=lower_limit
    xerrorupper=upper_limit;
run;
```

**Figure 10. Dot Plot with Custom Limits Using a SCATTER Statement**

In Figure 10, the display is horizontally oriented, just like the display you would get with the DOT statement. Therefore, the limits were specified on the X error options instead of the Y error options for proper rendering. The DISCRETE setting on the axis is important for numerical data because the default behavior for the axis is to treat numeric columns as continuous data. That behavior could cause the categorical values to be placed at non-regular intervals. The GRID option is added to guide the eye when associating data points with tick values. This plot can also be created with a vertical orientation by swapping the scatter plot variables, using the Y error options, and configuring the X axis instead of the Y axis.

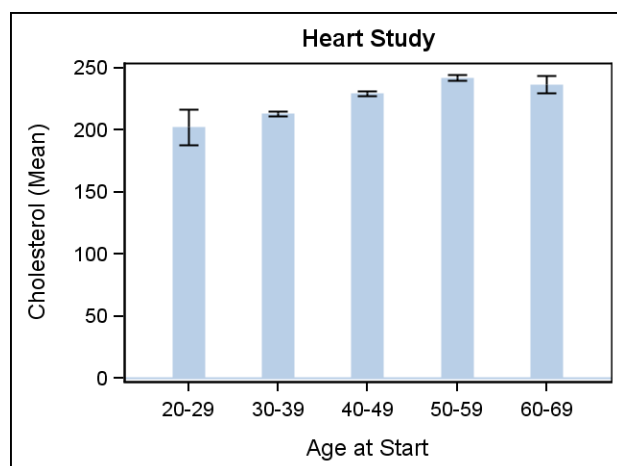
To create a mean plot similar to the one in Figure 9, you need to use two plots to generate all of the features. First, you use a SERIES statement to create a line plot connecting all of the mean values. Then, you overlay a SCATTER statement using the same mean values and use the Y error options to display your limits. This SCATTER statement is needed because the SERIES statement does not support error limits. You can also set the marker size in the SCATTER statement to be zero so that all you see are the limit bars along the series line (Figure 11).



```
proc sgplot data=precomputed
    noautolegend;
axis type=discrete;
series x=AgeAtStart y=mean_cholesterol;
scatter x=AgeAtStart y=mean_cholesterol /
    markerattrs=(size=0)
    yerrorlower=lower_limit
    yerrorupper=upper_limit;
run;
```

**Figure 11. Mean Plot Using SERIES and SCATTER Statements**

Displaying custom limits using a bar chart might require a little more manual adjustment. For this example, you need to use a needle plot instead of a bar chart, because bar charts do not allow custom limits. The thickness of the needles might need to be adjusted based on the number of categories in your data. For the graph in Figure 12, a line thickness of 20 is specified to give the needles the appearance of bars. Because the line colors in an ODS style tend to be more bold (for contrast), the default fill color from the style was used to give the needles a more subtle appearance. The scatter plot was then overlaid on the needle plot in the same manner as Figure 11.



```
proc sgplot data=precomputed
    noautolegend;
axis type=discrete;
needle x=AgeAtStart y=mean_cholesterol /
    lineattrs=(color=cxB9CFE7
                thickness=20);
scatter x=AgeAtStart y=mean_cholesterol /
    markerattrs=(size=0)
    yerrorlower=_errorlower1_
    yerrorupper=_errorupper1_;
run;
```

**Figure 12. Bar Chart with Custom Limits Using NEEDLE and SCATTER Statements**

## BAR CHART OVERLAYS AND GROUPS

There are other interesting techniques available to render bar charts with multiple measures or groups variables. In Figure 13, the actual versus predicted values for furniture sales are compared. To overlay the two measures, two bar chart statements are specified. Notice, however, that the second bar chart has a bar width of 0.5 specified. This is important because overlaid bar charts do not offset—they occupy the exact same data locations along the axis. Bar widths are specified as a percentage value between 0.0 and 1.0, regardless of the number of bars in the chart. A value of 1.0 will make the bars in the chart touch, but not overlap. The default bar width is 0.85; therefore, a value of 0.5 for the second bar chart guarantees that the bars from the second bar chart will fit inside the first bar chart.

Notice that a transparency value of 0.3 is specified for both charts. Transparency is specified as a percentage value between 0.0 and 1.0. A value of 1.0 means that the chart is completely transparent, while a value of 0.0 (the default) means that the chart is completely opaque. For the first bar chart, the transparency was specified for aesthetic reasons. The colors in the ODS style used by this chart are designed to be of approximately equal saturation. Because the second bar chart needed the transparency, the same value is put on the first bar chart to keep the color saturation consistent. The transparency on the second bar chart was added so that if any of its bar heights exceeds the first bar chart, you will still be able to see the top of the first bar chart through the second bar, making it easier to determine the amount of difference between the two values.





```

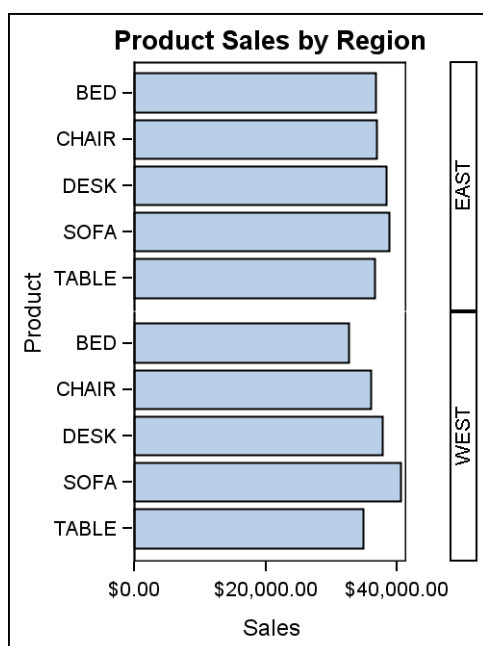
title1 "Product Sales";
proc sgpanel data=sashelp.prdsale;
  by year;
  panelby quarter;
  rowaxis label="Sales";
  vbar product / response=predict
               transparency=0.3;
  vbar product / response=actual
               barwidth=0.5
               transparency=0.3;
run;

```

**Figure 13. Overlaid Bar Charts**

One of the nice things about the SGPanel procedure is that it is able to automatically generate aesthetically pleasing graphs, regardless of the number of classification values in the PANELBY statement variables. Layout and pagination algorithms in the procedure attempt to maximize the space for each cell in the panel, while minimizing the number of empty cells in the last graph. Figure 13 is a simple example of this principle. In the Sashelp.Prdsale data set, there are two years of data with four quarters per year. The procedure determined that the best layout was a 2x2 layout for each year (the 1994 graph is not shown). There are times, however, when you might want to create a classic grouped bar chart, where all of the bars are displayed along a group axis. You'll find that the SGPlot procedure does not support this type of bar chart, but you can easily create it by using the SGPanel procedure.

In Figure 14, there is a bar chart displaying the actual sales for 1993 by region. The region variable is used as the group variable for this chart.



```

title1 "Product Sales by Region";
proc sgpanel data=sashelp.prdsale;
  where year=1993;
  panelby region / novarname onepanel noborder
               layout=rowlattice;
  colaxis label="Sales";
  hbar product / response=actual;
run;

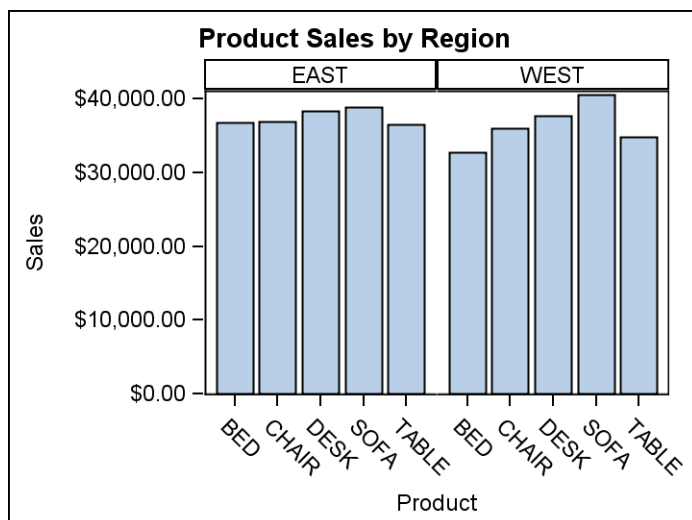
```

**Figure 14. Grouped Horizontal Bar Chart**

There are three key features for creating this display that are new in SAS 9.2 Phase 2. The first feature is the ONEPANEL option. This option tells the procedure to lay out all cells in one graph, regardless of the number of



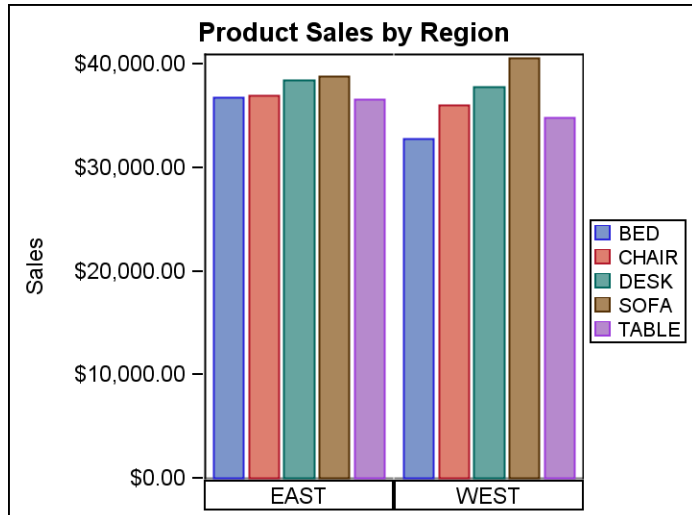
classifier values. The second feature is the NOBORDER option, which turns off the borders between the cells. The last feature is the two new layouts added to the LAYOUT option: ROWLATTICE and COLUMNLATTICE. The ROWLATTICE layout puts each cell in its own row, creating a one-column panel (Figure 14). The COLUMNLATTICE layout puts each cell in its own column, creating a one-row panel. This layout is the one you should use when creating a vertically oriented grouped bar chart (Figure 15). Users who have the SAS 9.2 Phase 1 release can simulate the results of the ONEPANEL option and the new layouts by explicitly setting the number of rows or columns to the number of classifier values in the data.



```
title1 "Product Sales by Region";
proc sgpanel data=sashelp.prdsale;
  where year=1993;
  panelby region / novarname
    onepanel noborder
    layout=columnlattice;
  rowaxis label="Sales";
  vbar product / response=actual;
run;
```

**Figure 15. Grouped Vertical Bar Chart**

Notice that to go from Figure 14 to Figure 15, all that is required is to change the layout to COLUMNLATTICE and change the orientation of the bar chart. The tick values along the column axis can become crowded in vertical bar charts. One way to relieve this crowding is to turn off the column axis, color the bars by category value, and use a legend to identify the category values (Figure 16).



```
proc sgpanel data=sashelp.prdsale;
  where year=1993;
  panelby region / onepanel
    novarname noborder
    layout=columnlattice
    colheaderpos=bottom;
  rowaxis label="Sales";
  colaxis display=none;
  vbar product / response=actual
    group=product;
  keylegend / position=right;
run;
```

**Figure 16. Grouped Bar Chart with a Category Legend**

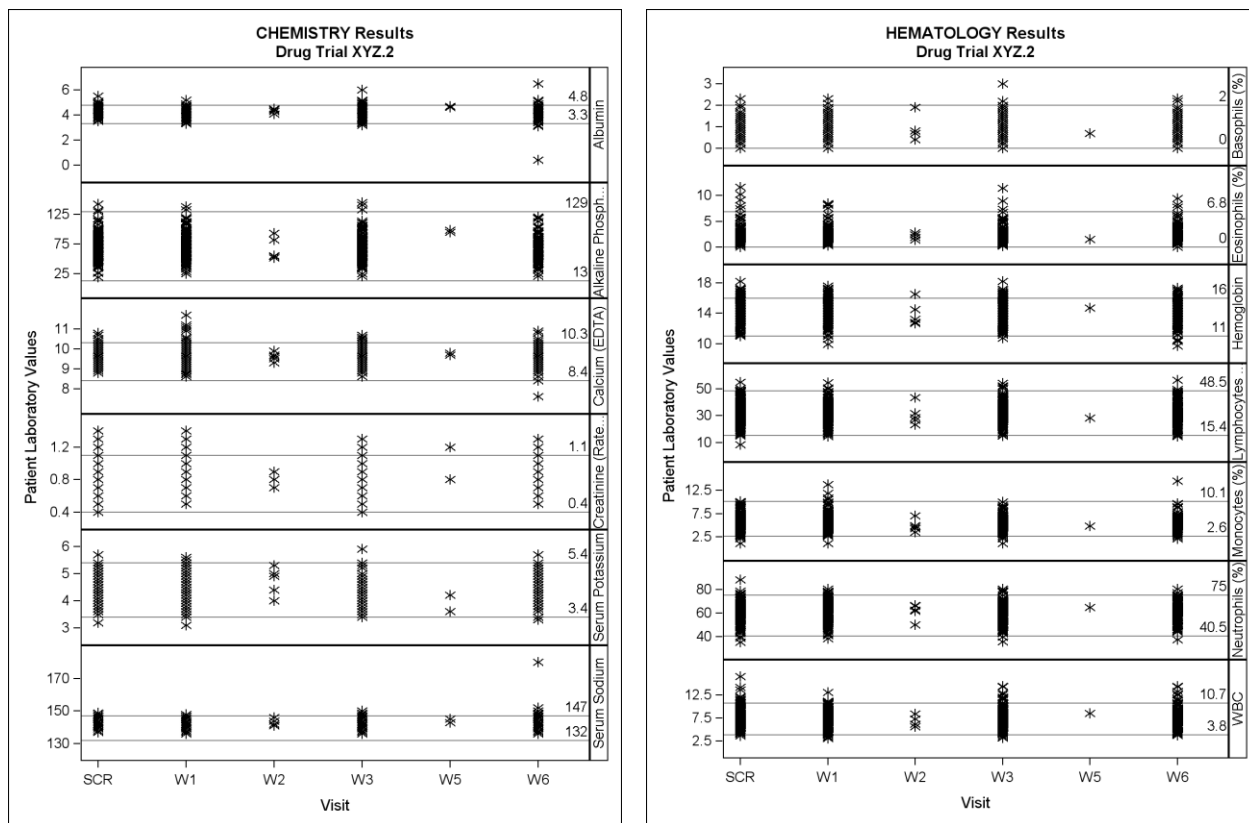
This example uses a new option in SAS 9.2 Phase 2 called COLHEADERPOS to move the cell headers below the column axis. The column axis is turned off, leaving only the group values along the bottom. The category variable is also assigned to the GROUP option to generate the category coloration. The legend is automatically generated because of the GROUP option, but the default position is along the bottom. Therefore, a KEYLEGEND statement is specified to move the legend to the right for aesthetics.

## PANELED PLOTS WITH DIFFERENT DATA RANGES

There might be times when you need to create a panel of plots to convey a complete message about your data, but the plots in the panel have different data ranges. Both the SGPanel and SGScatter procedures have facilities to help you create this type of display.

By default, the SGPPANEL procedure creates uniform row axes and uniform column axes across all graphs generated in a run. However, the PANELBY statement contains an option called UNISCALE that enables you turn off the uniformity of either the row or column axes. By setting the uniform scale for one axis, the range for the other axis is set to be independent. When used in conjunction with the ROWLATTICE or COLUMNLATTICE layout options, each plot can have one axis with an independent range.

An example of this technique can be seen in Figure 17. Each test in the blood chemistry and hematology panels has a different result range. Because the axis is shared within a single row or column (even with the UNISCALE option), this example uses the ROWLATTICE layout so that there is only one plot per row. For the SAS 9.2 Phase 1 release, you will need to set the number of rows equal to the number of tests to achieve this result.



### Figure 17. Blood Panels Displaying Multiple Tests

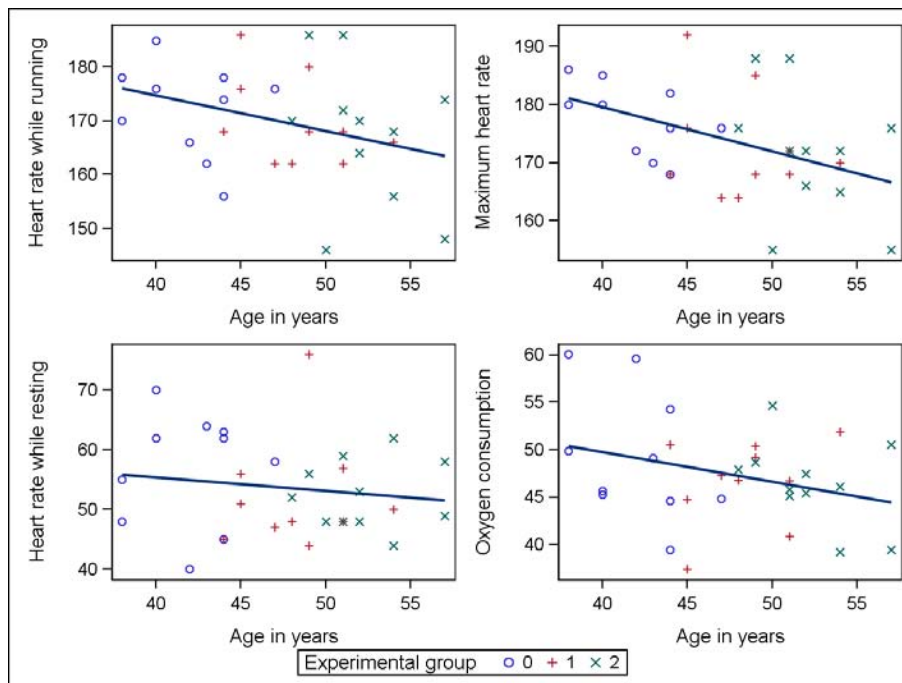
```
option nobylines;
title1 '#BYVAL1 Results';
title2 'Drug Trial XYZ.2';

proc sgpanel data=blood_tests;
  by ordname;
  panelby testname / onepanel uniscale=column
    layout=rowlattice novarname;
  refline numlow / label noclip;
  refline numhi / label noclip;
  scatter x=visitnum y=result /
    markerattrs=(symbol=asterisk size=15);
  rowaxis label='Patient Laboratory Values';
  colaxis offsetmax=0.1;
run;
```

Another interesting feature in Figure 17 is the data-driven reference lines. In the SAS 9.2 Phase 2 release, you can now use a column variable instead of constant values to create your reference lines. In this example, a DATA step was created that iterates by TestName to set the NumHi and NumLow values at the first.testname observation. These two variables are then referenced by the REFLINE statement. Another new reference line option called NOCLIP tells the plot to expand the axis range to include the reference lines instead of clipping them from the display. That way, you can always tell what the high and low normal range values are for each test, even if there is not any out-of-range data.

In PROC SGSCATTER, you can also create paneled plots with independent ranges by using either the PLOT statement or the COMPARE statement. The determination of which statement to use depends on your application and the layout you desire.

The PLOT statement enables you to create a panel of scatter plots or series plots with completely independent axes. In Figure 18, fitness data is displayed for three experimental groups. In addition to the scatter plot, a regression fit is requested to be overlaid on the scatter plot to gauge the trend of the results. Notice the use of the NOGROUP option with the REG option. The NOGROUP option tells the procedure to fit the data across all groups, rather than to generate a fit for each group.

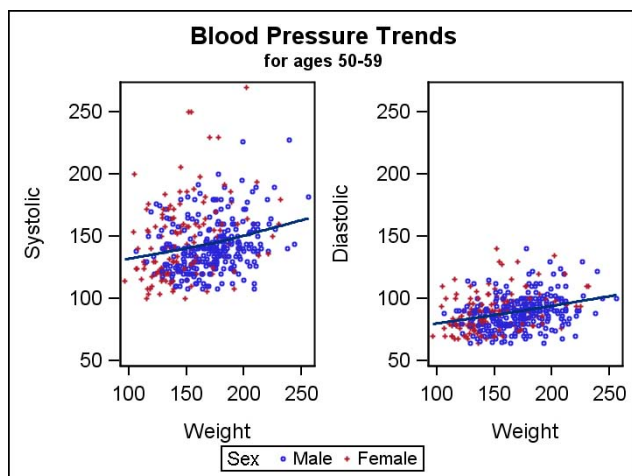


**Figure 18. Scatter Plot Panel of Fitness Data**

```
proc sgscatter data=fitness;
  plot (runpulse maxpulse rstpulse oxygen) * age / group=group reg=(nogroup);
run;
```

There might be situations, however, where you want a unified axis range for comparison purposes. This action can be specified in the PLOT statement by using the UNISCALE option. The UNISCALE option enables you to unify all Y ranges, all X ranges, or all ranges for each axis direction together across all plots in the panel. In Figure 19, the systolic and diastolic blood pressures are being compared against weight for the 50- to 59-year-olds in a study group. Because there is a relationship between the systolic range and the diastolic range, the Y axis ranges are unified using the UNISCALE option so that the diastolic data is in the correct location against the systolic data along the Y axis.

The COMPARE statement is designed to take a list of X and Y variables and create an MxN matrix with shared axes based on the crossing of those variables. If you have lists of both X and Y variables, there will be no range-independent plots in the panel because all axes are shared. However, if you specify only one X or Y variable, the axis of that variable is the only axis shared, while the other axis ranges are independent. In Figure 20, the X option is set to the Age variable, while the Y option is set to a list of variables. Therefore, the Y axis ranges vary independently, while the X axis range is uniform. This example could be created using a shared Y axis by simply swapping the X and Y variables. Notice that the LOESS option is used without the NOGROUP option. Because the NOGROUP option is not set, the loess fit is computed and drawn for each group in the plot.

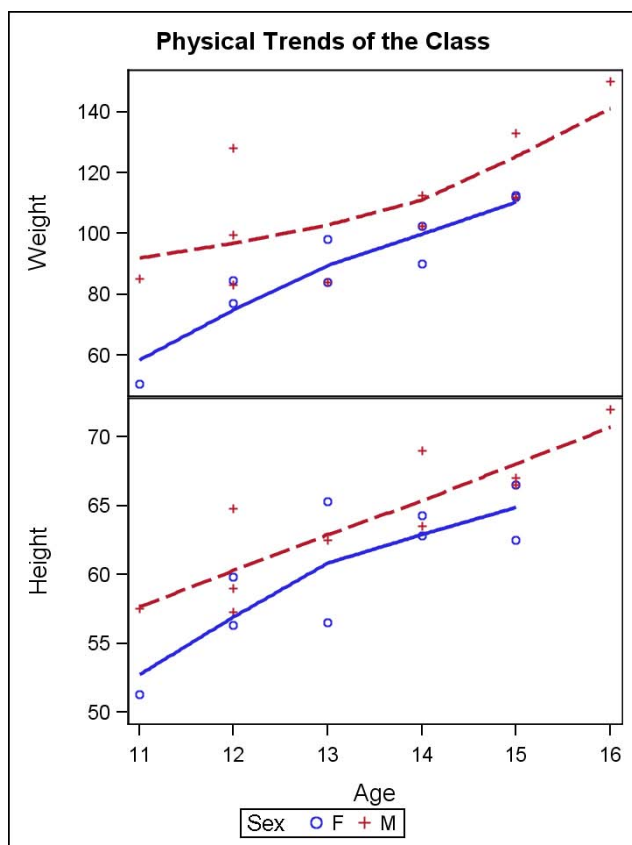


```

title1 "Blood Pressure Trends";
title2 h=8pt "for ages 50-59";
proc sgscatter data=sashelp.heart;
  where AgeCHDdiag>=50 and
        AgeCHDdiag<60;
  plot (systolic diastolic)*weight /
    group=sex loess=(nogroup)
    markerattrs=(size=3)
    uniscale=Y;
run;

```

Figure 19. The PLOT Statement Using the UNISCALE Option



```

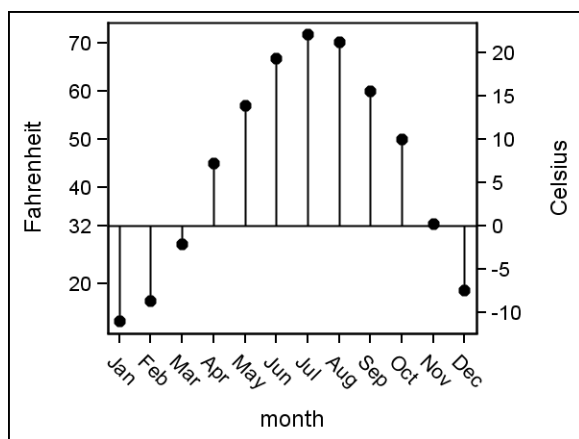
title1 "Physical Trends of the Class";
proc sgscatter data=sashelp.class;
  compare x=age y=(weight height) /
    loess group=sex;
run;

```

Figure 20. The COMPARE Statement with One Shared Axis

## AXIS TECHNIQUES

The linear axes in GTL and the SG procedures have some very interesting and useful properties. Two of those properties involve the layout of tick marks along the axis. Unlike many axis schemes, where the first and last tick marks always bound the data area, these axes only include the bounding tick mark value if the tick value minus the min/max data value is 30% or less of the inter-tick value range (this 30% threshold is configurable in GTL). The primary benefit of this behavior is that it prevents the data area from getting compressed just to include nice tick mark values at the end of the axes. Another important property is that tick marks are placed in their correct locations along the number line, enabling you to create irregular intervals along the linear axis. Figure 21 demonstrates the effect of both of those properties.



```
proc sgplot data=minntemp noautolegend;
  format month mymonnm.;
  xaxis type=discrete;
  yaxis values=(10 20 32 40 to 80 by 10)
        valueshint label="Fahrenheit";
  y2axis values=(-20 to 30 by 5)
        valueshint label="Celsius";
  scatter x=month y=f2 /
        markerattrs=(symbol=circlefilled);
  needle x=month y=c2 / y2axis baseline=0;
run;
```

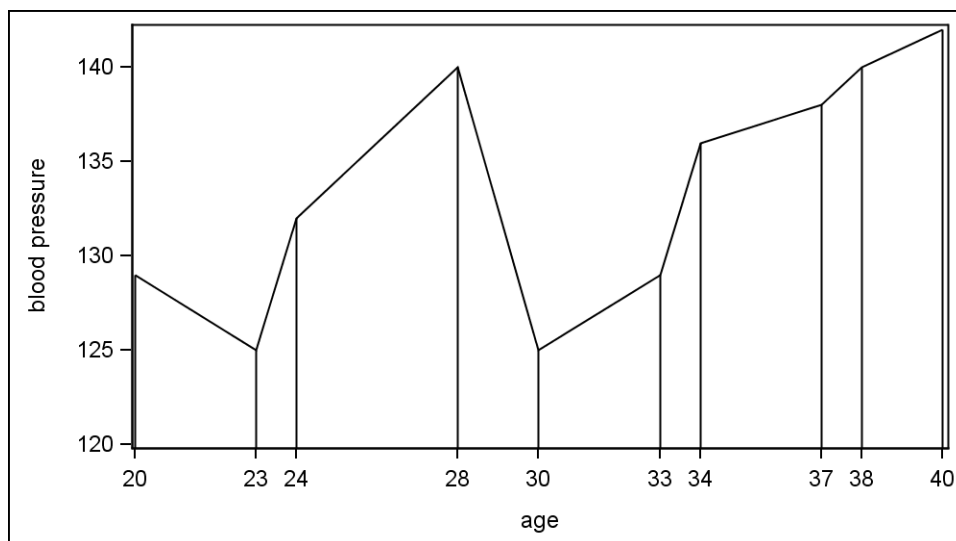
**Figure 21. Needle Plot with Secondary Axis Scale**

In Figure 21, the year's Fahrenheit data was plotted along the Y axis, while the corresponding Celsius data was plotted along the Y2 axis. To create the secondary axis scale, the correct data must be assigned to the Y2 axis. However, if you want to assign data to an axis without actually displaying the plot, you can set the plot's TRANSPARENCY option to 1.0 to make the plot completely transparent.

Notice that the data extends from one edge of the frame to the other, but the axis tick marks do not bound the data—even though the tick values are specified on the VALUES option. This behavior happens because of the VALUESHINT option. This option tells the procedure to not use the min/max tick values from the VALUES option to constrain the axis, but to display those tick values only if they fall within the full data range. In the case of the Fahrenheit axis, there were no data values of 10 degrees or less, so the tick value was not drawn. The VALUES and VALUESHINT combination enables you to define a set of standard tick mark values for your data one time, but vary which of those ticks are displayed based on the input data range.

Figure 21 also demonstrates using tick marks with irregular intervals. The tick interval for the Fahrenheit axis is set to be 10 degrees, except for the 32-degree tick mark. The 32-degree value was added to the VALUES option to create a tick mark at the freezing level that lines up with 0 degrees Celsius. Notice that the distance between 32 and 40 is less than the distance between 20 and 32, showing that the 32-degree tick mark is located correctly along the linear axis. This behavior enables you to define standard tick intervals and add special ticks as needed, rather than define an odd interval just so you can include the special tick value.

The irregular interval capability can be used to create a data-driven axis. In Figure 22, the blood pressure of an experimental group is plotted against the age of the group members. The ages are placed into macro variables in a DATA \_NULL\_ step that are later replayed by a %value() macro into the VALUES option of the X axis. The result is that the ages in the data are the only values on the axis.



**Figure 22. Data-Driven Tick Values**

```
data _null_;
```

```

set physical end=eof;
where group="exper";
call symput('val' || left(trim(_n_)), trim(left(age)));
if eof then call symput('tot',trim(left(_n_)));
run;

%macro value;
  %do i=1 %to &tot;
    &&val&i
  %end;
%mend value;

proc sort data=physical; by age; run;
proc sgplot data=physical noautolegend;
  where group="exper";
  xaxis values=(%value);
  yaxis min=120;
  needle y=bp x=age;
  series y=bp x=age;
run;

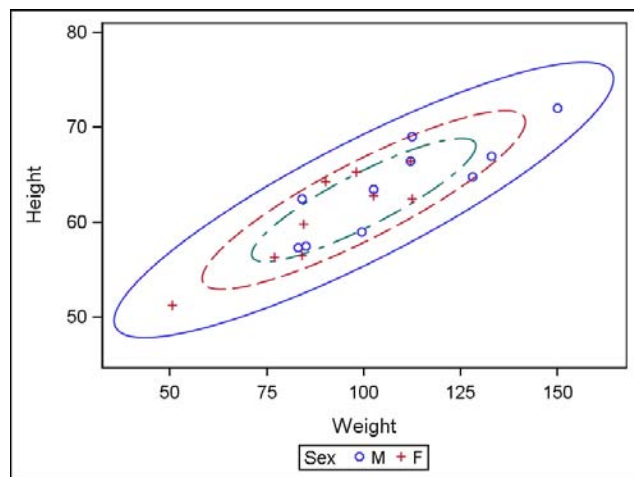
```

## LEGEND TECHNIQUES

Construction and positioning of legends can be an important consideration when creating a graph. You want to have legends that convey the necessary information for decoding the graph, while using a minimal amount of space. The next few examples will demonstrate some techniques you can use to accomplish this.

The SG procedures automatically analyze your plot request and try to determine the best legend to display. There are times, however, when you might want more than one legend or you want to change the content of the legend. The KEYLEGEND statement in the SGPLOT and SG PANEL procedures is used to create custom legends.

In Figure 23, the SG PLOT procedure determines that the best legend for the plot is a legend displaying the group values for the scatter plot. However, there are three prediction ellipses in the plot for which there is no information. Therefore, you might want to create an additional legend for the ellipses.



```

proc sgplot data=sashelp.class;
  ellipse x=weight y=height / alpha=0.05;
  ellipse x=weight y=height / alpha=0.25;
  ellipse x=weight y=height / alpha=0.5;
  scatter x=weight y=height / group=sex;
run;

```

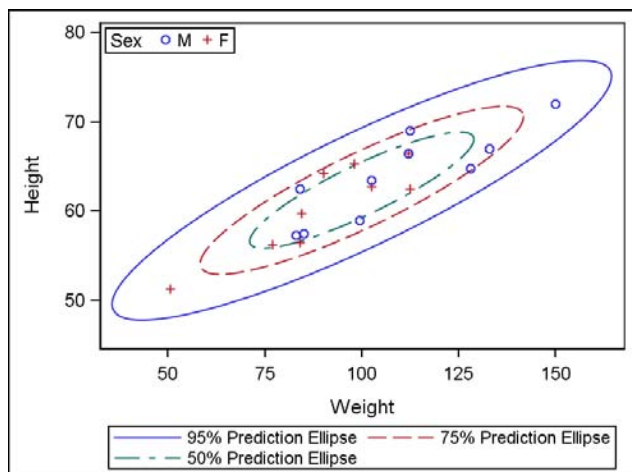
**Figure 23. Scatter Plot with Automatic Legend**

In Figure 24, two KEYLEGEND statements are specified to create a legend for both the grouped scatter plot and the prediction ellipses. The content of the legend is controlled by referring to the plots by name from the KEYLEGEND statement. The legend referring to the scatter plot retrieves the group information for display in the legend. If the GROUP option had not been specified, then the legend would contain only a single scatter marker labeled with the Y variable label. The legend referring to the ellipses creates one entry per ellipse. The descriptive information about each ellipse was automatically constructed in the procedure. These labels can be overridden by using the LEGENDLABEL option on each ELLIPSE statement. Also note that when a custom legend is created, the automatic legend is disabled, which is why the scatter plot legend had to be defined along with the ellipse legend.

Position of these legends is important. The more legends you have outside of your plot area, the more space is taken away for that area. Therefore, you might want to consider moving smaller legends inside of the plot area to reclaim the space, as was done in Figure 24. In this example, an explicit position of TOPLEFT was specified to force the legend into that position; however, if the position had not been specified, the legend would have moved into a position



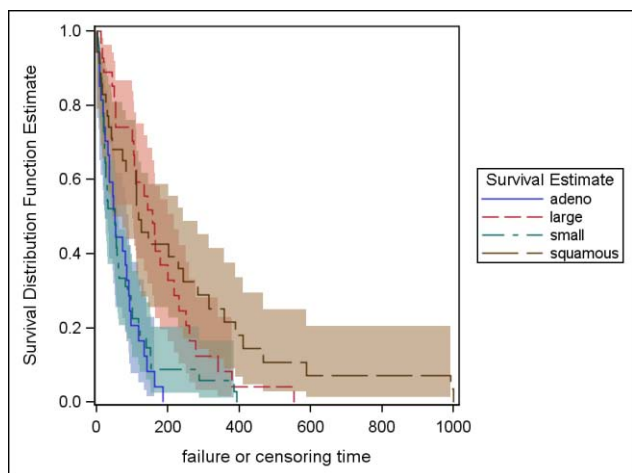
with the least amount of collision with the plot. This is a nice feature to use if your data is changing between program runs.



```
proc sgplot data=sashelp.class;
  ellipse x=weight y=height /
    alpha=0.05 name="e1";
  ellipse x=weight y=height /
    alpha=0.25 name="e2";
  ellipse x=weight y=height /
    alpha=0.5 name="e3";
  scatter x=weight y=height /
    group=sex name="s1";
  keylegend "e1" "e2" "e3";
  keylegend "s1" / location=inside
    position=topleft title="Sex";
run;
```

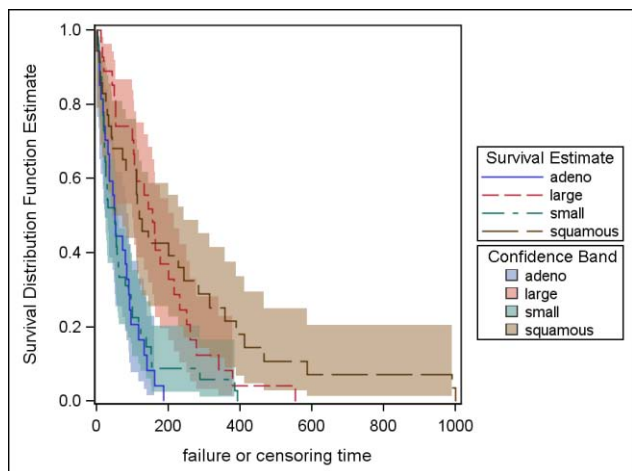
**Figure 23. Scatter Plot with Custom Legends**

Another technique for legend positioning is to reuse space created by the first legend to position the second legend. This situation occurs when legends are positioned to the left or right of the plot area. In Figure 24, the survival plot has a single legend to the right of the plot to identify each curve.



```
proc sgplot data=lungsurv2;
  band x=survtime lower=sdf_lcl
    upper=sdf_ucl / modelname="step"
    group=cell transparency=0.4
    name="band";
  step x=survtime y=survival /
    group=cell name="step";
  Keylegend "step" /
    title="Survival Estimate"
    position=right;
run;
```

**Figure 24. Survival Plot with a Single Legend**



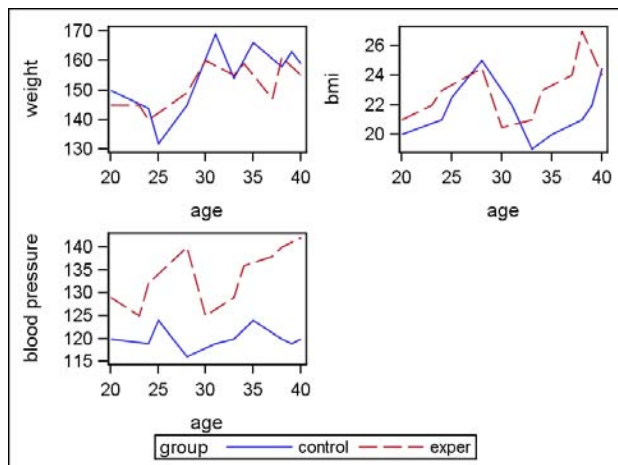
```
proc sgplot data=lungsurv2;
  band x=survtime lower=sdf_lcl
    upper=sdf_ucl / modelname="step"
    group=cell transparency=0.4
    name="band";
  step x=survtime y=survival /
    group=cell name="step";
  Keylegend "step" /
    title="Survival Estimate"
    position=right;
  Keylegend "band" /
    title="Confidence Band"
    position=right
  ;
run;
```

**Figure 25. Survival Plot with Two Legends**



If you want to add an additional legend to identify the confidence bands better, you can add that legend to the right of the plot without making the plot area any smaller. The procedure manages the legends in a way that allows the legends to be offset in the same location (Figure 25). If the second legend was added to the bottom of the graph, the plot area would have lost space from both the right and the bottom of the graph.

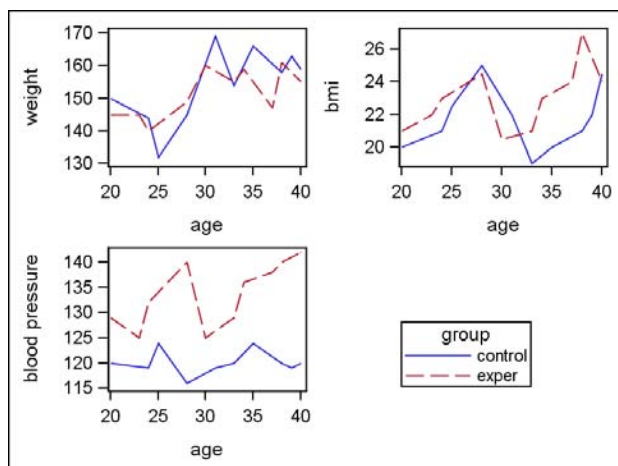
The SGSCATTER procedure also has an option to improve legend positioning for the PLOT statement in certain situations. For example, you can create a plot panel that does not fill every cell with a plot. If a group variable is used, the legend is automatically positioned as if the panel were fully populated (Figure 26).



```
proc sgscatter data=physical;
  plot (weight bmi bp) * age /
    group=group join
    markerattrs=(size=0);
run;
```

**Figure 26. Plot Panel with a Group Variable**

For situations like this, where you can use the empty space for the legend, you can specify the LOCATION=CELL option on the LEGEND option to make the procedure move the legend into the empty cell (Figure 27).



```
proc sgscatter data=physical;
  plot (weight bmi bp) * age /
    group=group join
    markerattrs=(size=0)
    legend=(location=cell);
run;
```

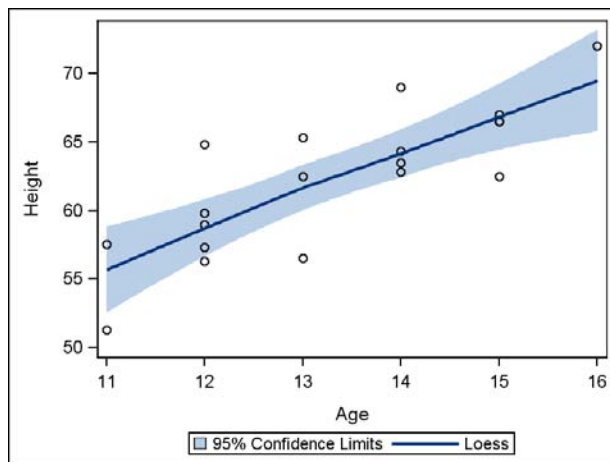
**Figure 27. Plot Panel Using the LOCATION=CELL Option**

One other item to note about Figures 26 and 27 is the JOIN option. This is a new option introduced in SAS 9.2 Phase 2 that connects the scatter points in each plot, much like the SERIES statement in PROC SGPLOT and SGPPANEL. The scatter markers are removed from the display by setting their size to zero.

## THE GTL OPTION

There might be times that you want to create a custom graph that you cannot completely make using the SG procedures. In those cases, using GTL might be your best option. However, there might be some users who are not sure how to create a GTL definition or would like to have a good starting point. Fortunately, the SGPLOT and SGSCATTER procedures have an option on the procedure statement called TMPLOUT that writes out a SAS file of the GTL definition for the graph you have defined in the procedure. You can then work with this definition to add options not supported in the SG procedures, or use this definition as a building block for a larger paneled graph.

For example, suppose you wanted to create a paneled graph containing a scatter plot and a box plot. You can create each of these plots individually using PROC SGPLOT and bind them together using GTL. The next three figures show the steps in this process.

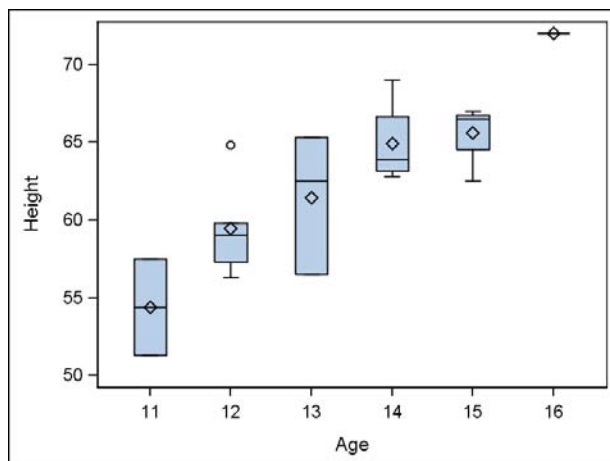


```
proc sgplot data=sashelp.class
            tplout="templ.sas";
    loess x=age y=height / clm;
run;
```

**Figure 28. First Plot in the Panel**

Figure 28 shows the first plot for panel. This plot is a scatter plot overlaid with a loess fit and confidence limits. The temp1.sas file contains the GTL definition for this plot, which looks like the following:

```
proc template;
define statgraph sgplot;
begingraph;
layout overlay;
    ModelBand "G2DJAN9I" / Name="MODEL BAND" LegendLabel="95% Confidence Limits";
    ScatterPlot X=Age Y=Height / primary=true;
    LoessPlot X=Age Y=Height / NAME="LOESS" LegendLabel="Loess" clm="G2DJAN9I";
    DiscreteLegend "MODEL BAND" "LOESS";
endlayout;
endgraph;
end;
run;
```



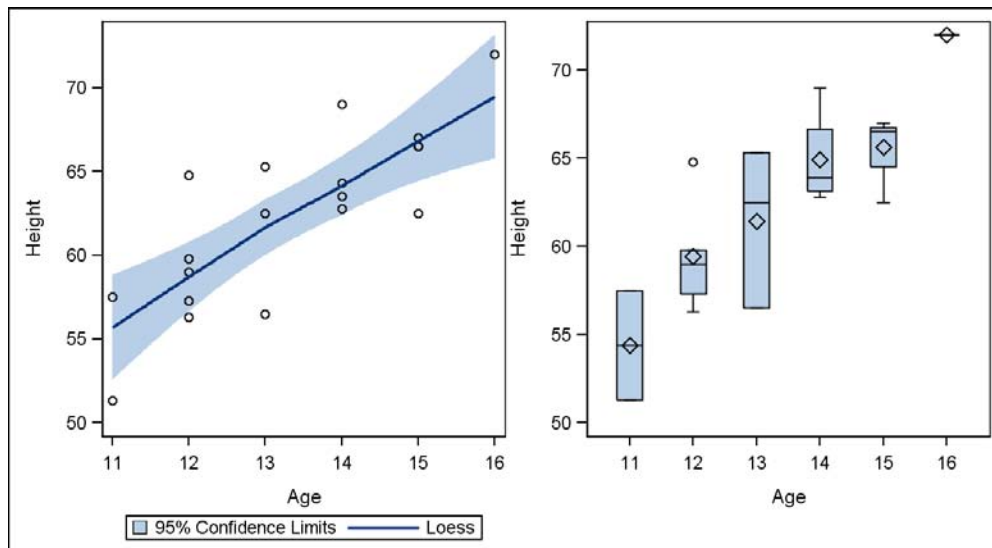
```
proc sgplot data=sashelp.class
            tplout="temp2.sas";
    vbox height / category=age;
run;
```

**Figure 29. Second Plot in the Panel**

Using the same approach as Figure 28, the GTL definition for the box plot in Figure 29 is written to temp2.sas using the following definition:

```
proc template;
define statgraph sgplot;
dynamic _ticklist_;
begingraph;
layout overlay / xaxisopts=(type=Discrete discreteOpts=(tickValueList=_ticklist_));
    BoxPlot X=Age Y=Height / SortOrder=Internal primary=true LegendLabel="Height"
            NAME="VBOX";
endlayout;
endgraph;
end;
run;
```

Now that you have your two plot definitions, you can extract the necessary pieces to construct the paneled graph. The key statement for creating the panel is either the LAYOUT GRIDDED or the LAYOUT LATTICE statements. In Figure 30, the LAYOUT LATTICE statement is used with two columns to put the plots in the same row. Notice that only the LAYOUT OVERLAY portions of the single-plot definitions are used (the highlighted areas). Also notice the dynamic variable `_TickList_` on the TICKVALUELIST option. Because a value for that variable was not specified in the SGRENDER procedure, the TICKVALUELIST option is ignored. There might be situations where variables like this might need their values specified, or the data might need to be pre-computed to achieve the same output appearance as from the procedure.



**Figure 30. Paneled Graph Using GTL**

```
proc template;
define statgraph panel;
dynamic _ticklist_;
begingraph;
layout lattice / columns=2;
  layout overlay;
    ModelBand "G2DJAN9I" / Name="MODEL BAND" LegendLabel="95% Confidence Limits";
    ScatterPlot X=Age Y=Height / primary=true;
    LoessPlot X=Age Y=Height / NAME="LOESS" LegendLabel="Loess" clm="G2DJAN9I";
    DiscreteLegend "MODEL BAND" "LOESS";
  endlayout;
  layout overlay / xaxisopts=(type=Discrete
                             discreteOpts=(tickValueList=_ticklist_));
    BoxPlot X=Age Y=Height / SortOrder=Internal primary=true
      LegendLabel="Height" NAME="VBOX";
  endlayout;
endgraph;
end;
run;

proc sgrender data=sashelp.class template=panel;
run;
```

There are other things that can be done to improve the appearance of this graph, such as sharing the Height axis or moving the legend to the center of the graph. All of these things can be accomplished in GTL, but they are beyond the scope of this paper. For further information on creating GTL definitions, read Sanjay Matange's paper, [Introduction to the Graph Template Language](#).

## CONCLUSION

There are many options available in the SG procedures to help you create the custom graphics you need. Hopefully, the techniques discussed here will help you in that goal. Remember that plot types and options can be used in creative ways to achieve results beyond their standard usage. Just use your imagination.

## ACKNOWLEDGMENTS

I want to thank Susan Schwartz, Melisa Turner, and Sanjay Matange for their input and review of this paper.

## RECOMMENDED READING

- Heath, Dan. 2007. "New SAS/GRAPH Procedures for Creating Statistical Graphics in Data Analysis." *Proceedings of the 2007 SAS Global Forum*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/rnd/datavisualization/papers/sgf2007/SGF2007-Proc.pdf>.
- Heath, Dan. 2008. "Effective Graphics Made Simple Using SAS/GRAPH SG Procedures." *Proceedings of the 2008 SAS Global Forum*. Cary, NC: SAS Institute Inc. Available at <http://www2.sas.com/proceedings/forum2008/255-2008.pdf>.
- Matange, Sanjay. 2008. "Introduction to the Graph Template Language." *Proceedings of the 2008 SAS Global Forum*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/resources/papers/sgf2008/gtl.pdf>.

## CONTACT INFORMATION

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

Dan Heath  
SAS Institute Inc.  
500 SAS Campus Dr.  
Cary, NC 27513  
(919) 531-6657  
Dan.Heath@sas.com

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

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