http://support.sas.com/documentation/prod-p/grstat/

Advanced ODS Graphics Examples

Warren F. Kuhfeld

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *Advanced ODS Graphics Examples*. Cary, NC: SAS Institute Inc.

Advanced ODS Graphics Examples

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

November 2015

Contents

Preface

If you are an experienced SAS user, you probably already know about ODS Graphics (also called ODS Statistical Graphics) and how to do the following:

- enable ODS Graphics so that analytical procedures produce graphical output together with tabular output
- modify graph templates to modify the output that analytical procedures produce
- work with SAS styles
- use SG procedures such as PROC SGPLOT to produce custom plots

If these topics are not already familiar to you, then you should start by looking at the chapters and book listed in the section "About the Author" on page v. If you are familiar with these topics and want to learn more, then you have found the right book. This book goes beyond the basics and works through a series of advanced examples. Each example illustrates a small fraction of the incredible power of ODS Graphics. In particular, there are many examples of PROC SGPLOT and the SG annotation facility. The last chapter shows you how to customize every aspect of the graphs that analytical procedures produce.

Sample Code

The beginning of every example has a link that you can double click to see the SAS code for that example. The first example has the following link:

Double Click for Example Code

These links work in Adobe Acrobat Reader and Internet Explorer but not in Google Chrome or Mozilla Firefox.

About the Author

This book was written by Warren F. Kuhfeld, Distinguished Research Statistician Developer in SAS/STAT Research and Development. Warren received his PhD in psychometrics from the University of North Carolina at Chapel Hill in 1985 and joined SAS in 1987. He has used SAS since 1979 and has developed SAS procedures since 1984.

In addition to writing the SAS Press book *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*, Warren also wrote the following SAS/STAT documentation chapters: Chapter 20, "Using the Output Delivery System" (*SAS/STAT User's Guide*), Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*), Chapter 22, "ODS Graphics Template Modification" (*SAS/STAT User's Guide*), Chapter 23, "Customizing the Kaplan-Meier Survival Plot" (*SAS/STAT User's Guide*).

Warren maintains 11 SAS/STAT procedures. He developed 20 SAS macros for experimental designs for linear and choice models. His 1,309-page book about discrete choice and other marketing research methods, *Marketing Research Methods in SAS*, is free on the web: `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. Warren has also developed the world's largest collection of orthogonal arrays: `http://support.sas.com/techsup/technote/ts723.html`.

Acknowledgments

I would like to thank my editor, Anne Baxter, who always finds a way to make my writing clearer, and our documentation tools specialist, Tim Arnold, without whom none of the Advanced Analytics Division's documentation would be possible. I would also like to thank the members of the ODS and ODS Graphics groups who have helped me and worked with me on many things over the years: Dan Heath, Prashant Hebbar, Wayne Hester, David Kelley, Lingxiao Li, Sanjay Matange, Dan O'Connor, and Xiao Le Xu. ODS Graphics would not exist without their hard work and the hard work of the rest of the ODS and ODS Graphics teams. Finally, I would like to extend a special thanks to Bob Rodriguez. ODS Graphics would not exist without his vision and perseverance.

Chapter 1
Introduction

This book discusses a series of examples that use ODS Graphics. A few examples use the graph template language (GTL), most use the SGPLOT procedure, and many use statistical graphics (SG) annotation. The first example (see the section "Multiple Axes, Offsets, and Drop Lines" on page 3) introduces multiple axes and offsets, which are used throughout many of the other examples. This book also discusses axis tables (Chapter 3, "Axis Tables,"), scatter plot label placement (Chapter 6, "Plots of Labeled Points,"), forest plots (see the section "Creating a Forest Plot Using PROC SGPLOT" on page 42), adverse event plots (see the section "Adverse Events Plot" on page 145), dynamic variable modification (see the section "Changing Dynamic Variables by Using the ODS Document" on page 197), attribute maps (see the section "Attribute Maps" on page 148), and many other topics. The last three examples (see the sections "Changing Dynamic Variables by Using the ODS Document" on page 197, "Annotating Single-Panel Graphs That Analytical Procedures Produce" on page 217, and "Annotating Multiple-Panel Graphs That Analytical Procedures Produce" on page 228) show you how to modify every aspect of the graphs that analytical procedures produce and even annotate the graphs. For a list of all SG annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133.

Earlier examples explain the code in great detail. However, even the earlier examples assume that you are already familiar with ODS Graphics; they do not explain all the basics (such as styles, paths, templates, item stores, destinations, attribute priority, and so on). The functions of many options are obvious from their names, so they are often not explained. Later examples mostly explain newly introduced features and options.

For more information about ODS Graphics, see Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*), and Chapter 22, "ODS Graphics Template Modification" (*SAS/STAT User's Guide*). ODS Graphics documentation includes: *SAS ODS Graphics: Procedures Guide*, *SAS Graph Template Language: User's Guide*, and *SAS Graph Template Language: Reference*. SAS Press books include Kuhfeld (2010) and Matange and Heath (2011).

References

Kuhfeld, W. F. (2010). *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*. Cary, NC: SAS Institute Inc.

Matange, S., and Heath, D. (2011). *Statistical Graphics Procedures by Example: Effective Graphs Using SAS*. Cary, NC: SAS Institute Inc.

Chapter 2
Axes

Contents

A graph can have up to four axes: Y on the left, Y2 on the right, X on the bottom, and X2 on the top. Typically, most graphs have two axes (Y on the left and X on the bottom) and all points have coordinates on these two axes. Additional axes enable you to display more information in a single graph than you could reasonably display by using only X and Y axes. Multiple axes and offsets are heavily used in subsequent examples. These techniques, while simple, provide powerful tools for making advanced graphs.

Multiple Axes, Offsets, and Drop Lines

Double Click for Example Code

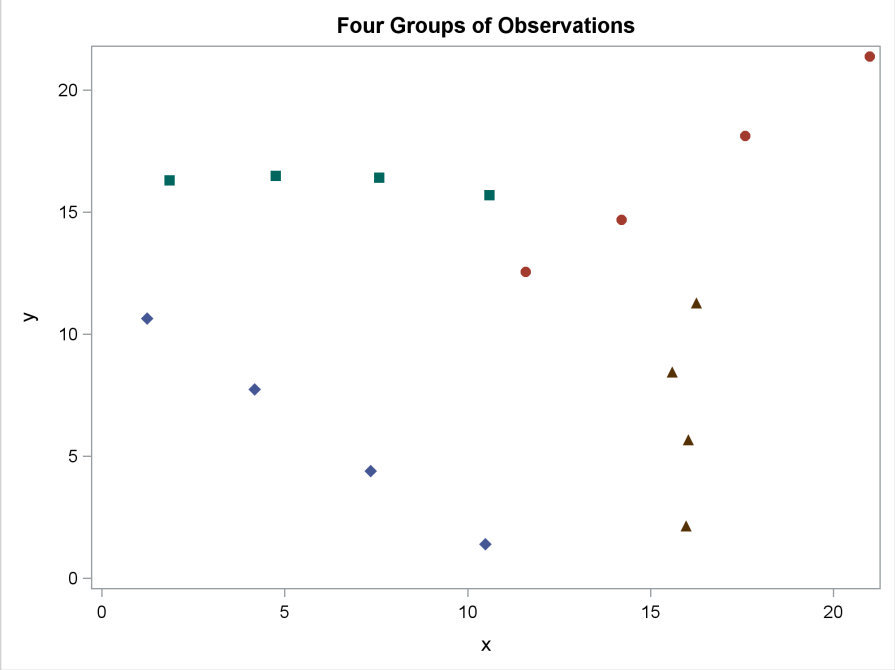
This example uses a small number of artificial data points to illustrate multiple axes and offsets. This example also illustrates drop lines, which are used to show the correspondence between points and axes.

The following steps create and display the data in Figure 2.1:

```
data x(drop=i);
   do i = 1 to 10 by 3;
      x  = i       + uniform(104);
      y  = 11 - x  + uniform(104);

      x2 =  i + 10 + uniform(104);
      y2 = x2      + uniform(104);

      x3 = i       + uniform(104);
      y3 = 15.5    + uniform(104);

      x4 = 15.5    + uniform(104);
      y4 = x3      + uniform(104);
      output;
   end;
run;
```

```
proc sgplot noautolegend;
   title 'Four Groups of Observations';
   scatter  y=y  x=x  / markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / markerattrs=(symbol=circlefilled);
   scatter  y=y3 x=x3 / markerattrs=(symbol=squarefilled);
   scatter  y=y4 x=x4 / markerattrs=(symbol=trianglefilled);
run;
```

Figure 2.1 Four Groups of Observations



The NOAUTOLEGEND option in the PROC SGPLOT statement suppresses the legend. The four SCATTER statements make four graphs, all in the same set of X and Y axes, and each statement explicitly controls the marker attributes. PROC SGPLOT automatically cycles the unspecified attributes across statements, selecting attributes from the **GraphData***n* style elements. Hence, the first points, the filled diamonds, are blue (which comes from the **GraphData1** style element). Continuing, the circles are red (**GraphData2**), the squares are green (**GraphData3**), and the triangles are brown (**GraphData4**). Almost every graph in this book is constructed by using the HTMLBlue style. Other styles use other colors. You can disable automatic attribute cycling by specifying the NOCYCLEATTRS statement in the PROC SGPLOT statement.

The axis labels correspond to the name of the first variable that is plotted on each axis. All graphs in this example are constructed so that the variable names that are displayed as axis labels also identify the axis type: X, X2, Y, and Y2. The first few parts of this example begin by using a subset of these data (the diamonds and circles). The remainder of the example uses all four groups.

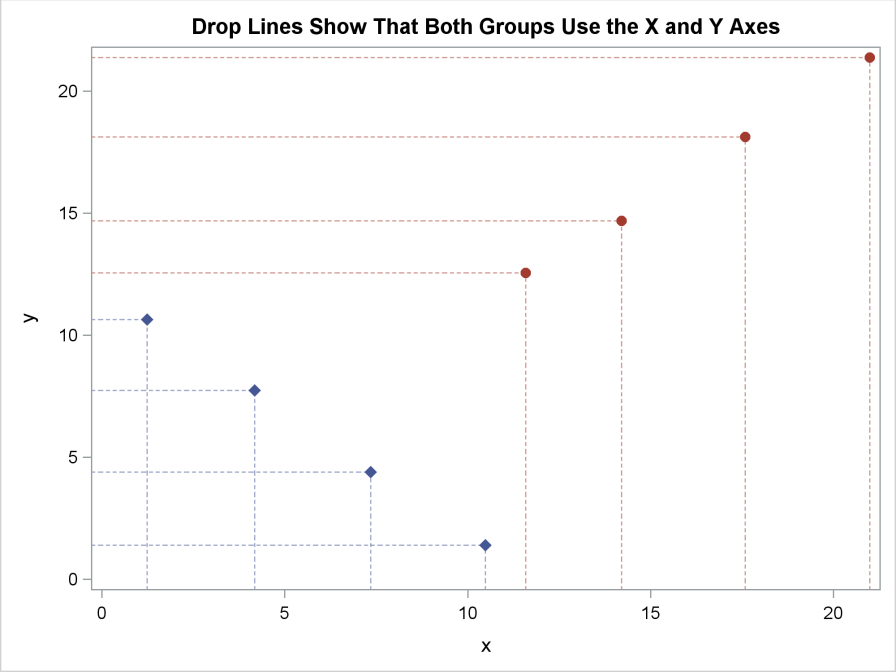
All of the following steps in this example display data and drop lines that run from each point to the axes that are used for that point. The following macro controls the appearance of the drop lines:

```
%macro drop(n);
   dropto=both lineattrs=graphdata&n(pattern=3) transparency=0.5
%mend;
```

A macro is not needed; it is used to minimize typing. The DROPTO=BOTH option extends lines to both axes. The PATTERN=3 option specifies a dashed line, and TRANSPARENCY=0.5 makes the line partially transparent. The colors of the drop lines are determined by the style elements **GraphData1** through **GraphData4**, where the index of the **GraphData***n* style element is controlled by the macro variable &n. The following step creates Figure 2.2:

```
proc sgplot noautolegend;
   title 'Drop Lines Show That Both Groups Use the X and Y Axes';
   scatter  y=y  x=x  / markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / markerattrs=(symbol=circlefilled);
   dropline y=y  x=x  / %drop(1);
   dropline y=y2 x=x2 / %drop(2);
run;
```

Figure 2.2 Simple Drop Lines

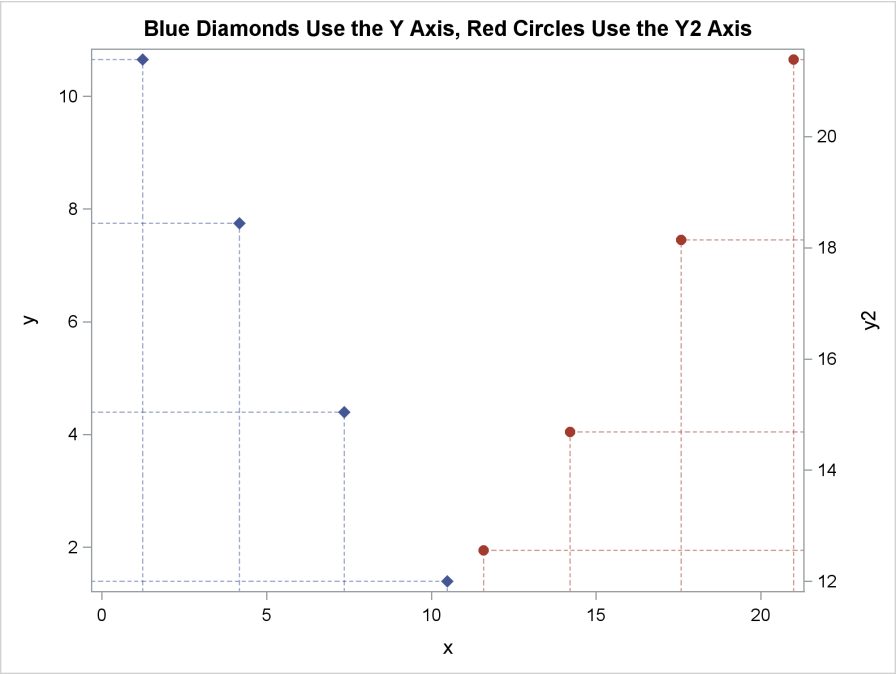


Like most scatter plots, this graph has an X axis and a Y axis. For both sets of points, the drop lines extend downward to the X axis and to the left to the Y axis.

The following step creates Figure 2.3:

```
proc sgplot noautolegend;
   title 'Blue Diamonds Use the Y Axis, Red Circles Use the Y2 Axis';
   scatter  y=y  x=x  /        markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / y2axis markerattrs=(symbol=circlefilled);
   dropline y=y  x=x  /        %drop(1);
   dropline y=y2 x=x2 / y2axis %drop(2);
run;
```

Figure 2.3 Two Y Axes

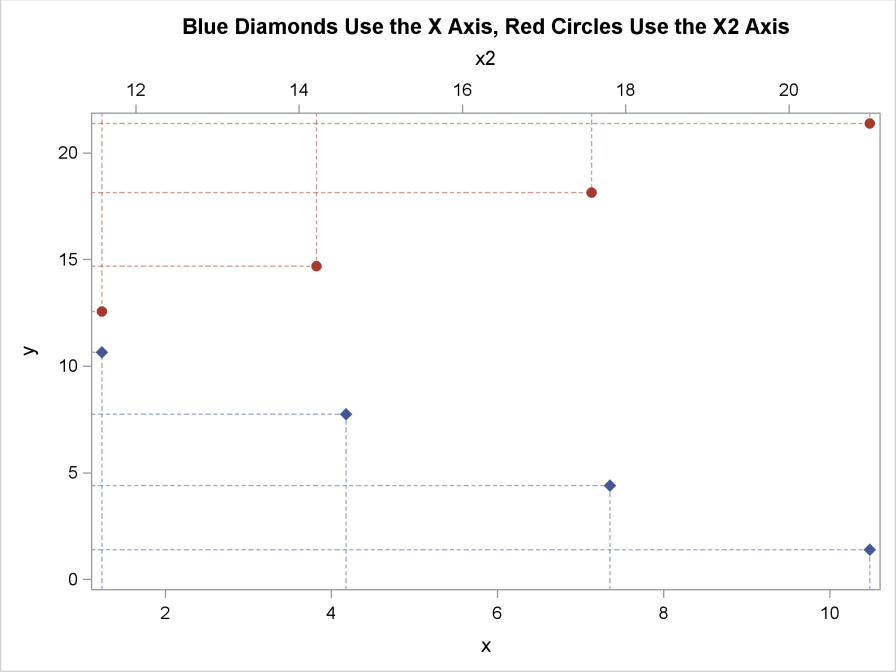


This graph has X, Y, and Y2 axes. The Y2 axis is used when the Y2AXIS option is specified. The X2 axis is used when the X2AXIS option is specified (which is first shown in the next step). X and Y axes are used when no other axis options are specified. The drop lines for the blue diamonds extend to the left to the Y axis and downward to the X axis, which is the default behavior when neither the X2AXIS nor the Y2AXIS option is specified. Because both the second SCATTER and the second DROPLINE statements contain the Y2AXIS option, the drop lines for the red circles extend downward to the X axis and to the right to the Y2 axis. The range of values for the Y axis is independent of the range of values for the Y2 axis.

The following step creates Figure 2.4:

```
proc sgplot noautolegend;
   title 'Blue Diamonds Use the X Axis, Red Circles Use the X2 Axis';
   scatter  y=y  x=x  /        markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / x2axis markerattrs=(symbol=circlefilled);
   dropline y=y  x=x  /        %drop(1);
   dropline y=y2 x=x2 / x2axis %drop(2);
run;
```

Figure 2.4 Two X Axes

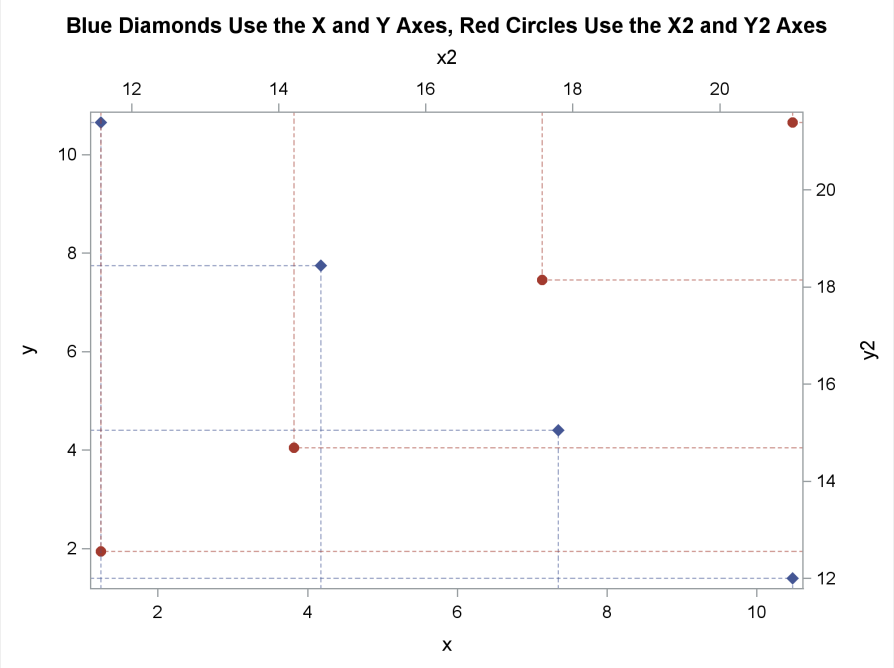


This graph has X, X2, and Y axes. The drop lines for the blue diamonds extend downward to the X axis and to the left to the Y axis. The drop lines for the red circles extend upward to the X2 axis and to the left to the Y axis. The range of values for the X axis is independent of the range of values for the X2 axis.

The following step creates Figure 2.5:

```
proc sgplot noautolegend;
   title 'Blue Diamonds Use the X and Y Axes, '
         'Red Circles Use the X2 and Y2 Axes';
   scatter  y=y  x=x  /               markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / x2axis y2axis markerattrs=(symbol=circlefilled);
   dropline y=y  x=x  /               %drop(1);
   dropline y=y2 x=x2 / x2axis y2axis %drop(2);
run;
```

Figure 2.5 Two Y Axes and Two X Axes

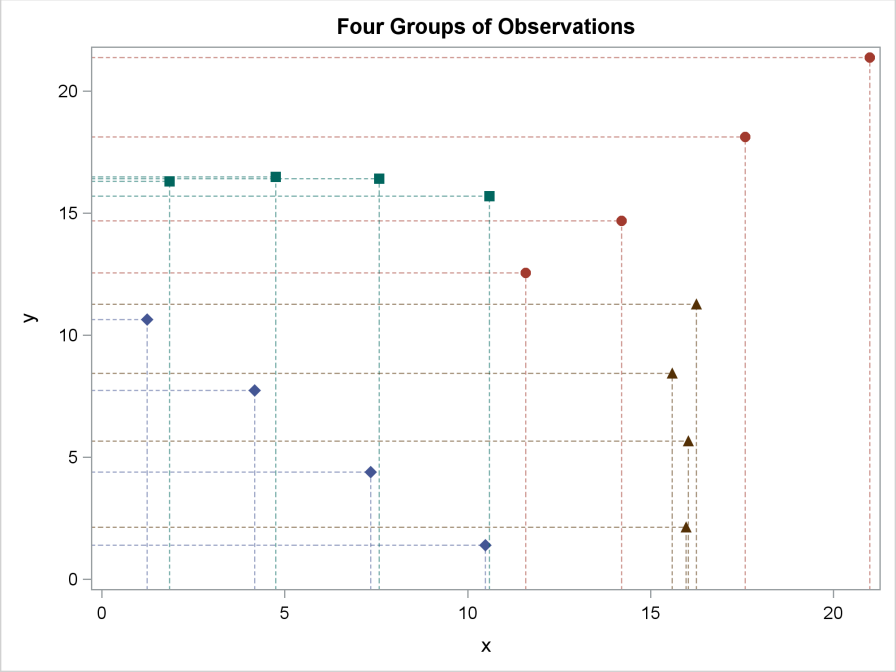


This graph has X, X2, Y, and Y2 axes. The first scatter plot uses the X and Y axes, and the second scatter plot uses the X2 and Y2 axes. The drop lines for the blue diamonds extend downward to the X axis and to the left to the Y axis. The drop lines for the red circles extend to the upward to the X2 axis and to the right to the Y2 axis. The range of values for the X axis is independent of the range of values for the X2 axis, and the range of values for the Y axis is independent of the range of values for the Y2 axis.

The following step creates Figure 2.6:

```
proc sgplot noautolegend;
   title 'Four Groups of Observations';
   scatter  y=y  x=x  / markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / markerattrs=(symbol=circlefilled);
   scatter  y=y3 x=x3 / markerattrs=(symbol=squarefilled);
   scatter  y=y4 x=x4 / markerattrs=(symbol=trianglefilled);
   dropline y=y  x=x  / %drop(1);
   dropline y=y2 x=x2 / %drop(2);
   dropline y=y3 x=x3 / %drop(3);
   dropline y=y4 x=x4 / %drop(4);
run;
```

Figure 2.6 Four Groups

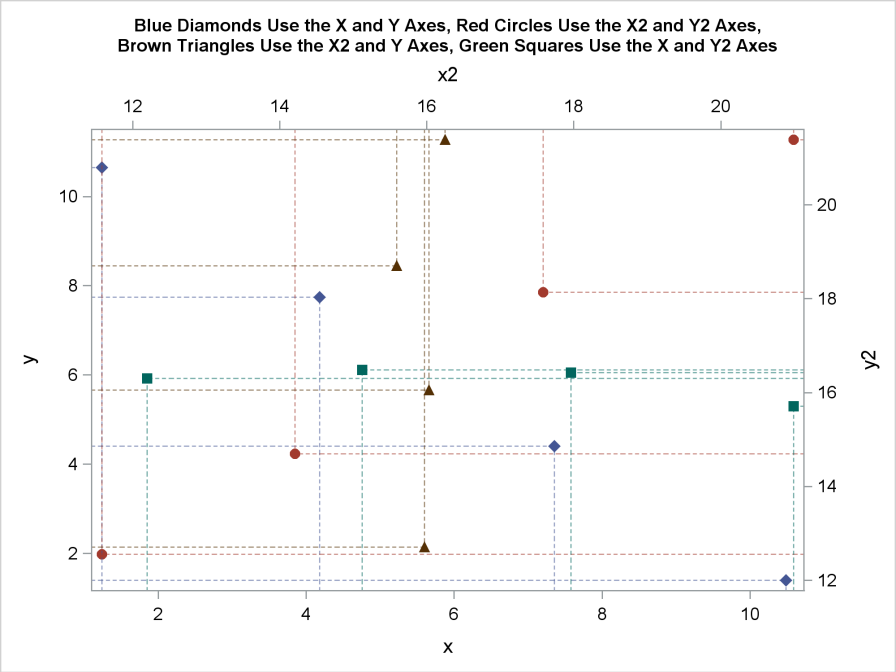


In this and subsequent steps, all four groups of data are displayed. This graph has X and Y axes. The drop lines for all points extend downward to the X axis and to the left to the Y axis.

The following step creates Figure 2.7:

```
proc sgplot noautolegend;
   title  h=1 'Blue Diamonds Use the X and Y Axes, '
              'Red Circles Use the X2 and Y2 Axes,';
   title2 h=1 'Brown Triangles Use the X2 and Y Axes, '
              'Green Squares Use the X and Y2 Axes';
   scatter  y=y  x=x  /               markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / x2axis y2axis markerattrs=(symbol=circlefilled);
   scatter  y=y3 x=x3 /        y2axis markerattrs=(symbol=squarefilled);
   scatter  y=y4 x=x4 / x2axis        markerattrs=(symbol=trianglefilled);
   dropline y=y  x=x  /               %drop(1);
   dropline y=y2 x=x2 / x2axis y2axis %drop(2);
   dropline y=y3 x=x3 /        y2axis %drop(3);
   dropline y=y4 x=x4 / x2axis        %drop(4);
run;
```

Figure 2.7 All Four Axes



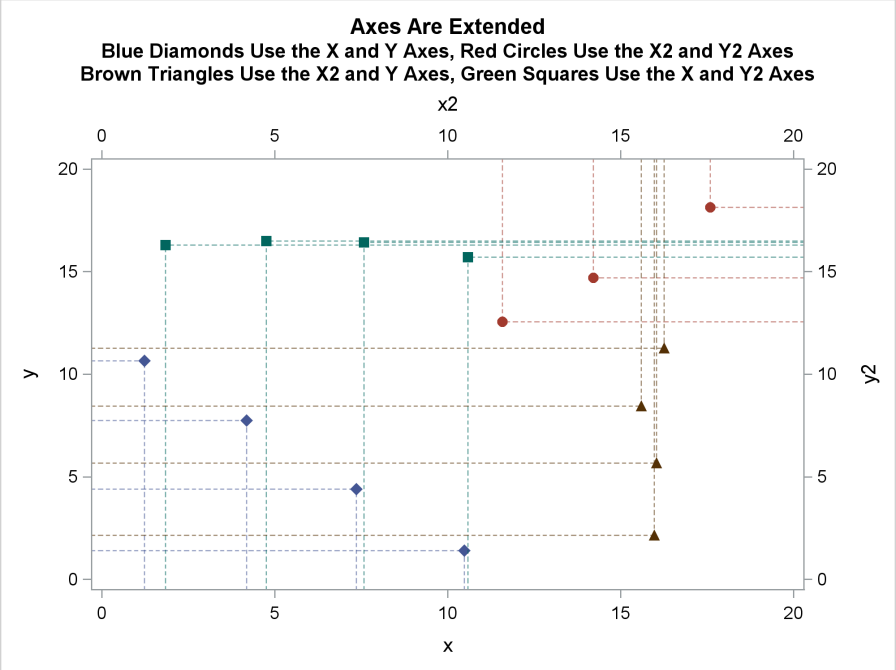
This graph has X, X2, Y, and Y2 axes. The drop lines for the blue diamonds extend downward to the X axis and to the left to the Y axis. The drop lines for the red circles extend upward to the X2 axis and to the right to the Y2 axis. The drop lines for the brown triangles extend upward to the X2 axis and to the left to the Y axis. The drop lines for the green squares extend downward to the X axis and to the right to the Y2 axis. The range of values for the X axis is independent of the range of values for the X2 axis, and the range of values for the Y axis is independent of the range of values for the Y2 axis.

The following step creates Figure 2.8:

```
proc sgplot noautolegend;
   title  'Axes Are Extended';
   title2 'Blue Diamonds Use the X and Y Axes, '
          'Red Circles Use the X2 and Y2 Axes';
   title3 'Brown Triangles Use the X2 and Y Axes, '
          'Green Squares Use the X and Y2 Axes';
   scatter  y=y  x=x  /               markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / x2axis y2axis markerattrs=(symbol=circlefilled);
   scatter  y=y3 x=x3 /        y2axis markerattrs=(symbol=squarefilled);
   scatter  y=y4 x=x4 / x2axis        markerattrs=(symbol=trianglefilled);
   dropline y=y  x=x  /               %drop(1);
   dropline y=y2 x=x2 / x2axis y2axis %drop(2);
   dropline y=y3 x=x3 /        y2axis %drop(3);
   dropline y=y4 x=x4 / x2axis        %drop(4);
   xaxis  min=0 max=20;
   yaxis  min=0 max=20;
   x2axis min=0 max=20;
   y2axis min=0 max=20;
run;
```

This graph has X, X2, Y, and Y2 axes. All four axes are extended so that the ranges for the X and X2 axes match each other and the ranges of the Y and Y2 axes also match each other. The X, X2, Y, and Y2 axes are controlled by the statements XAXIS, X2AXIS, YAXIS, and Y2AXIS, respectively; the MIN= and MAX= options control the ranges.

Figure 2.8 All Four Axes Extended



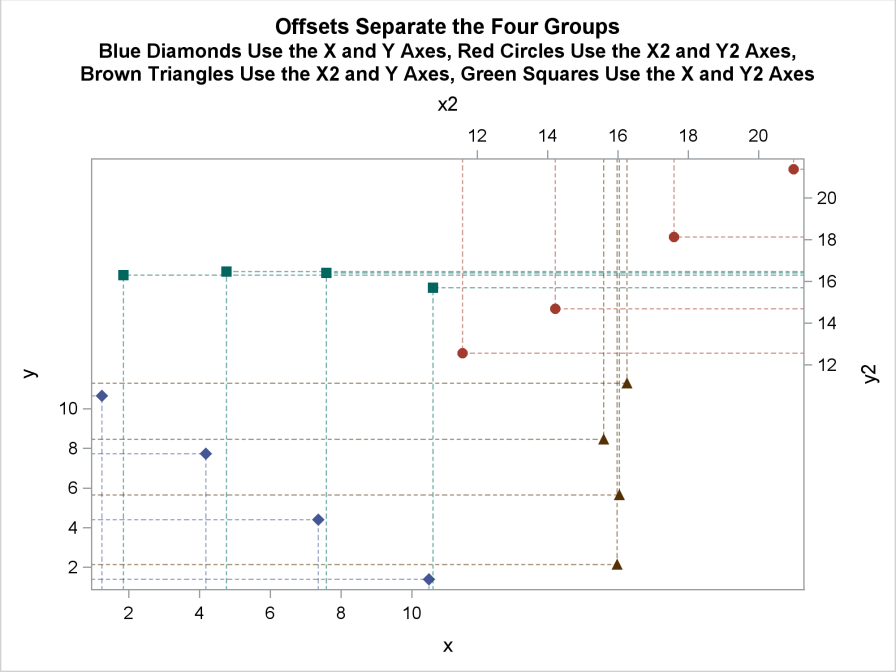
The drop lines for the blue diamonds extend downward to the X axis and to the left to the Y axis. The drop lines for the red circles extend upward to the X2 axis and to the right to the Y2 axis. The drop lines for the brown triangles extend upward to the X2 axis and to the left to the Y axis. The drop lines for the green squares extend downward to the X axis and to the right to the Y2 axis.

The following step creates Figure 2.9:

```
proc sgplot noautolegend;
   title  'Offsets Separate the Four Groups';
   title2 'Blue Diamonds Use the X and Y Axes, '
          'Red Circles Use the X2 and Y2 Axes,';
   title3 'Brown Triangles Use the X2 and Y Axes, '
          'Green Squares Use the X and Y2 Axes';
   scatter  y=y  x=x  /               markerattrs=(symbol=diamondfilled);
   scatter  y=y2 x=x2 / x2axis y2axis markerattrs=(symbol=circlefilled);
   scatter  y=y3 x=x3 /        y2axis markerattrs=(symbol=squarefilled);
   scatter  y=y4 x=x4 / x2axis        markerattrs=(symbol=trianglefilled);
   dropline y=y  x=x  /               %drop(1);
   dropline y=y2 x=x2 / x2axis y2axis %drop(2);
   dropline y=y3 x=x3 /        y2axis %drop(3);
   dropline y=y4 x=x4 / x2axis        %drop(4);
   xaxis   offsetmax=.52;
   yaxis   offsetmax=.52;
   x2axis  offsetmin=.52;
   y2axis  offsetmin=.52;
run;
```

This graph has X, X2, Y, and Y2 axes, and each axis has an offset. The OFFSETMAX= option reserves space to the right or above, and the OFFSETMIN= option reserves space to the left or below. The blue diamonds are plotted in the bottom left, and space is reserved to the right and above for other parts of the graph. The red circles are plotted in the top right, and space is reserved to the left and below for other parts of the graph. The brown triangles are plotted in the bottom right, and space is reserved to the left and above for other parts of the graph. The green squares are plotted in the top left, and space is reserved to the right and below for other parts of the graph.

Figure 2.9 Offsets Separate the Four Groups



The drop lines for the blue diamonds extend downward to the X axis and to the left to the Y axis. The drop lines for the red circles extend upward to the X2 axis and to the right to the Y2 axis. The drop lines for the brown triangles extend upward to the X2 axis and to the left to the Y axis. The drop lines for the green squares extend downward to the X axis and to the right to the Y2 axis. The range of values for the X axis is independent of the range of values for the X2 axis, and the range of values for the Y axis is independent of the range of values for the Y2 axis.

This approach (multiple axes and offsets that reserve space for other parts of the graph) is extensively used in subsequent examples. It is important to understand the techniques illustrated in this example before going on to subsequent examples.

Multiple Axes and Highlighted Points

Double Click for Example Code

This example shows how to construct a graph that has multiple axes, and it highlights selected points in the graph. The data are from the mileage records of someone training to run a marathon (26.2 miles). The following step reads the data:

```
%let start = '01jan2013'd;
%let stop  = '29apr2014'd;
data runs(keep=date distance totalmiles);
   retain year '     ';
   input d $ Distance @@;
   if nmiss(distance) then do; year = '/' || d; return; end;
   Date = input(trim(d) || year, mmddyy10.);
   TotalMiles + distance;
   if date ge &stop - 364 then PastYear + distance;
   call symputx('t', put(pastyear, 4.));
   output;
datalines;
2013 .  1/1 7.6 1/3 7.6 1/5 10.4 1/7 7.6 1/9 7.6 1/13 10.4 1/24 7.6 1/28 10.4
1/30 7.6 2/1 7.6 2/3 12.2 2/5 7.6 2/10 7.6 2/12 8.2 2/14 8.2 2/16 13.1 2/19 8.2
2/21 8.2 2/24 10.4 2/26 7.6 2/28 8.2 3/2 13.65 3/4 8.2 3/6 8.2 3/8 13.9 3/10
10.4 3/12 8.2 3/14 8.2 3/16 8.2 3/17 5.2 3/19 8.2 3/21 8.2 3/23 8.2 3/27 8.2
3/28 5.2 3/30 14 4/1 5.2 4/3 8.2 4/5 5.2 4/6 10.4 4/8 8.2 4/10 8.2 4/11 5.2 4/13
10.4 4/15 8.2 4/17 5.2 4/20 8.2 4/22 8.2 4/24 8.2 4/26 8.2 4/27 3 4/28 4 4/29
4.25 4/30 5.2 5/1 4.1 5/2 4.1 5/4 8.2 5/6 5.2 5/7 8.2 5/9 5.2 5/11 10.4 5/13
4.35 5/14 4.25 5/15 4.25 5/22 5.2 5/25 10.4 5/27 10.4 5/29 10.4 5/31 14 6/2 10.4
6/4 10.4 6/6 10.4 6/8 10.4 6/9 10.4 6/11 10.4 6/15 8.2 6/19 5.2 6/23 10.4 6/25
8.2 6/26 5.2 6/28 15 6/30 11.2 7/2 3 7/3 8.2 7/5 8.2 7/6 8.2 7/7 8.2 7/9 8.2
7/10 8.2 7/12 16 7/15 8.2 7/16 8.2 7/17 5.2 7/19 14 7/20 8.2 7/22 8.2 7/23 5.2
7/24 10.4 7/26 8.2 7/28 4.1 7/29 6.3 7/30 10.4 7/31 8.2 8/3 11.9 8/4 4.5 8/5 5
8/6 6.1 8/7 4.2 8/9 5.2 8/10 4 8/11 12 8/13 5.2 8/16 5.2 8/17 10.4 8/18 8.2 8/19
5.2 8/20 5.2 8/21 8.2 8/23 5.4 8/27 5.2 8/28 5.2 8/29 3 9/1 5.2 9/3 5.2 9/4 8.2
9/5 10.4 9/6 5.2 9/7 5.2 9/8 8.9 9/9 7 9/12 5.2 9/14 18 9/17 5.2 9/18 10.4 9/19
5.2 9/20 6.75 9/21 10.4 9/23 8.2 9/24 8.2 9/27 8.2 9/28 20 10/2 5.2 10/3 8.2
10/4 5.2 10/5 14 10/8 9.8 10/10 10 10/11 5.2 10/13 20 10/15 8.2 10/16 5.2 10/17
8.2 10/22 21.3 10/24 8 10/25 5.2 10/26 14 10/30 5.2 10/31 5.2 11/1 5.2 11/2 10.4
11/5 3.1 11/7 3.1 11/9 27.36 11/11 5.2 11/12 5.2 11/13 5.2 11/15 14 11/16 8.2
```

```
11/17 5.2 11/18 5.2 11/20 10.4 11/21 5.24 11/24 10.4 11/25 8.2 11/26 5.2 11/28
10.4 12/2 5.2 12/3 8.2 12/4 10.4 12/5 5.2 12/6 10.4 12/7 10.4 12/8 5.2 12/9 8.3
12/11 10.5 12/12 8.2 12/13 5.3 12/14 10.4 12/15 8.2 12/16 5.2 12/17 5.3 12/18
10.4 12/19 8.4 12/20 5.2 12/22 8.2 12/23 8.2 12/24 8.2 12/25 8.2 12/26 8.2 12/27
8.2 12/28 5.2 12/30 14 12/31 5.2 2014 .  1/1 5.2 1/2 5.2 1/3 5.2 1/4 5.2 1/5
10.2 1/9 5.2 1/10 5.2 1/11 6.8 1/14 5.2 1/17 5.2 1/18 6.8 1/19 6.8 1/20 5.2 1/21
5.2 1/23 5.2 1/24 5.3 1/25 5.2 1/26 10.4 1/27 7.6 1/28 8.2 1/29 3.3 2/1 10.4 2/2
10.4 2/4 5.2 2/5 8.2 2/6 5.2 2/8 14 2/10 5.2 2/11 5.2 2/12 8.3 2/15 5.2 2/16 8.2
2/18 8.2 2/20 5.2 2/23 18.2 2/25 5.2 2/27 5.2 2/28 21 3/2 8.2 3/4 5.2 3/8 10.4
3/10 3.1 3/12 3.1 3/16 26.3 3/20 3.1 3/21 5.2 3/22 5.2 3/27 5.2 3/28 5.2 3/30 14
3/31 8.2 4/2 5.2 4/4 5.2 4/5 14 4/6 10.4 4/8 5.2 4/10 8.2 4/11 3 4/12 14 4/13
9.1 4/14 5.2 4/16 5.2 4/17 5.2 4/18 14 4/20 14 4/21 5.2 4/22 8.2 4/23 5.2 4/24
5.2 4/25 14 4/27 14 4/29 5.3
;
```

The Runs data set has three variables: Distance (distance run that day), Date (date of the run), and TotalMiles (total miles recorded since the start date). The statement **`TotalMiles + distance`** is a SAS sum statement (which is equivalent to **`TotalMiles = TotalMiles + distance`** and also retains TotalMiles and initializes it to zero). The following steps prepare the data for graphing:

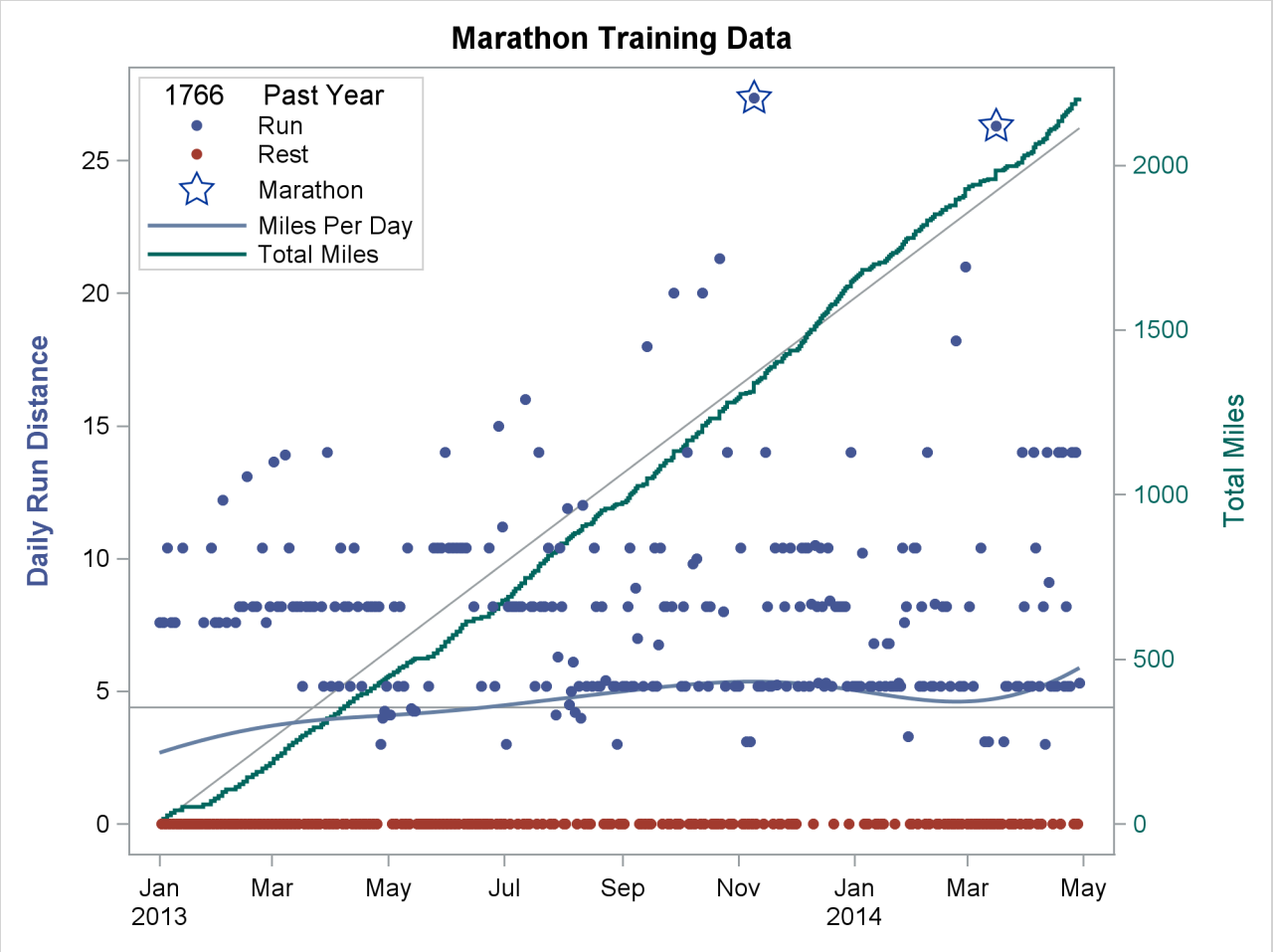
```
data all;
   Distance = 0;
   do Date = &start to &stop;
      output;
   end;
run;

data r2;
   update all runs(in=r);
   by date;
   Day = ifc(r, 'Run ', 'Rest');
   if distance ge 26.2 then Star = distance;
   x = date - &start;
   output;
run;

proc reg data=r2 noprint;
   model totalmiles = x / noint;
   output p=pred out=r;
run;
```

The first two DATA steps add an observation for each date that the runner did not run. The IFC function returns `'Run'` when r is true (the Runs data set is being processed) and otherwise returns `'Rest'`. The PROC REG step fits the model TotalMiles = (Date – '01Jan2013'd), where '01Jan2013'd is the starting date for training. The predicted values, Pred, provide a reference line for the graph of the total miles. The following step creates the graph displayed in Figure 2.10:

```
proc sgplot data=r;
   title 'Marathon Training Data';
   refline  %sysevalf(1600 / 365);
   series   y=pred x=date      / y2axis lineattrs=graphreference;
```

```
    step     y=totalmiles x=date/ y2axis lineattrs=graphdata3(thickness=2px)
                                  name='t' legendlabel='Total Miles';
    pbspline y=distance x=date / nomarkers name='p' legendlabel='Miles Per Day';
    scatter  y=distance x=date / group=Day name='r'
                                  markerattrs=(size=5px symbol=circlefilled);
    scatter  y=star     x=date / markerattrs=(size=15px symbol=star)
                                  legendlabel='Marathon' name='s';
    format date mmddyy8.;
    xaxis display=(nolabel);
    yaxis label='Daily Run Distance' labelattrs=graphdata1(weight=bold);
    y2axis labelattrs=graphdata3 valueattrs=graphdata3 label='Total Miles';
    keylegend 'r' 's' 'p' 't' / title="   &t     Past Year     "
                                location=inside position=topleft across=1;
run;
```

Figure 2.10 Graph That Has Two Y Axes



The blue dots show daily runs, and the daily run distance in miles is displayed on the Y axis. The red dots show resting days.

The horizontal reference line is at approximately 4.4 miles—the average number of miles needed per day to reach 1,600 miles for the year. The blue curve is a penalized B-spline function that is fit to the distances (all running and resting days); it shows a smooth function of distance run per day. When the curve is above the

reference line, the runner is exceeding his goal. Notice that the reference line and the penalized B-spline function would be misleading if the resting days had been excluded from the graph. The graph displays stars to highlight the marathons (runs of 26.2 miles or longer).

This graph also shows the total miles run over the course of the marathon training. The graph needs two Y axes because daily runs are on a scale of 0 to 27.36 miles whereas the total distance is on a scale of 0 to more than 2,204.5 miles. If there were one combined axis, the daily runs would be too compressed to provide information. Total distance is displayed in the green step function and the Y2 axis (the Y axis on the right). The diagonal reference line is a regression line that fits total miles as a function of time. In some periods the green step function is increasing faster than the average slope, and in other periods it is increasing at a slower rate.

The graph is constructed as follows:

- The REFLINE statement draws a horizontal reference line at 1,600 miles divided by 365 days and uses the Y axis.
- The SERIES statement displays the diagonal reference line and uses the Y2 axis.
- The STEP statement displays total miles and uses the Y2 axis.
- The PBSPLINE statement displays a smooth function of distance per day as a function of time and uses the Y axis.
- The first SCATTER statement displays distances as 5-pixel filled circles and uses the Y axis. The group variable Day differentiates resting days from running days.
- The second SCATTER statement displays a 15-pixel star for the marathon runs.
- The FORMAT statement declares the Date variable as a date variable. PROC SGPLOT does not display date variables as literally prescribed by the format. Rather, it extracts the years and labels the ticks in an elegant way.
- The XAXIS statement suppresses the X axis label. The tick labels are sufficient to show that the X axis is a date axis.
- The YAXIS statement sets the Y axis label.
- The Y2AXIS statement uses the right Y axis to display the total miles in the same style as the step plot (**`GraphData3`**).
- The KEYLEGEND statement displays a legend that identifies the points in the plot, the smooth function of average miles per day, and the total miles.
- Fittingly, a RUN statement ends the step.

The **`GraphReference`** style element is used implicitly and explicitly for the two reference lines. The PBSPLINE fit function uses the **`GraphFit`** style element. Running days (the first group) are displayed by using the **`GraphData1`** style element (blue), resting days by using **`GraphData2`** (red), and total miles by using **`GraphData3`** (green). Reference lines are drawn first so that they never obscure any other graph elements. The total miles function is drawn next so that it never obscures distances per day. The PBSPLINE function is fit to all distances (ignoring groups), whereas the scatter plot of distances displays the running and resting days as separate groups.

Multiple Axes, Axis Alignment, and Many Tick Labels

Double Click for Example Code

The previous example showed how to make a graph that has multiple axes that are independent of each other. That is the typical usage of multiple axes. This example shows you how to make two axes that have the same range and same ticks in the same positions, but use different tick labels. The X axis displays the date, and the X2 axis displays the day of the week. To illustrate, the following step reads a data set that contains datetime values and a variable y:

```
data x;
   input Time datetime16. y @@;
   time2 = time;
   datalines;
22AUG15:17:47:59 4 30AUG15:04:46:07 3 05SEP15:23:50:59 4 30AUG15:22:00:02 4
22AUG15:09:59:36 4 28AUG15:11:07:32 5 30AUG15:00:33:31 4 07SEP15:12:40:46 4
23AUG15:04:51:09 4 26AUG15:07:30:54 4 22AUG15:04:29:50 3 07SEP15:21:22:41 4
07SEP15:15:38:58 5 06SEP15:18:10:15 3 08SEP15:14:52:29 4 27AUG15:10:01:01 5
24AUG15:14:49:55 4 04SEP15:09:57:37 4 07SEP15:06:18:21 3 04SEP15:21:26:21 4
25AUG15:23:30:27 3 27AUG15:23:53:22 4 30AUG15:11:14:32 5 30AUG15:06:52:19 4
06SEP15:01:41:58 5 05SEP15:20:03:54 3 04SEP15:20:05:47 3 30AUG15:04:17:00 3
02SEP15:04:33:24 4 23AUG15:19:21:22 4 08SEP15:02:44:52 4 30AUG15:16:46:16 3
22AUG15:17:16:58 4 29AUG15:04:46:29 3 03SEP15:20:07:47 3 23AUG15:15:29:14 5
08SEP15:22:42:49 3 05SEP15:22:53:44 4 22AUG15:06:23:28 5 30AUG15:19:50:41 3
04SEP15:11:07:21 5 22AUG15:09:43:14 5 07SEP15:04:31:59 4 30AUG15:09:43:30 4
23AUG15:13:22:42 6 07SEP15:22:11:57 4 25AUG15:01:36:36 4 04SEP15:03:37:37 4
03SEP15:09:09:53 3 02SEP15:00:57:35 2 27AUG15:20:44:19 4 29AUG15:03:31:12 5
28AUG15:18:05:29 4 31AUG15:18:17:15 4 24AUG15:12:01:32 4 07SEP15:08:43:56 2
03SEP15:13:08:22 3 02SEP15:08:17:34 3 28AUG15:08:09:03 2 24AUG15:15:05:34 3
22AUG15:07:51:43 5 01SEP15:05:52:43 3 01SEP15:04:02:02 4 05SEP15:20:10:14 4
25AUG15:08:49:33 2 06SEP15:03:46:10 3 28AUG15:23:45:55 4 27AUG15:18:56:00 4
23AUG15:02:28:42 4 30AUG15:18:41:04 6 25AUG15:22:52:42 4 31AUG15:04:31:22 4
29AUG15:07:31:17 4 30AUG15:00:04:45 6 25AUG15:21:15:18 6 08SEP15:07:11:50 6
27AUG15:10:42:16 3 30AUG15:18:25:38 4 28AUG15:19:44:59 1 23AUG15:15:37:26 4
25AUG15:13:13:21 5 31AUG15:06:47:43 3 05SEP15:00:51:34 6 07SEP15:13:33:42 5
28AUG15:19:20:32 4 03SEP15:08:08:12 4 09SEP15:05:36:24 3 29AUG15:11:31:29 5
26AUG15:09:41:25 5 30AUG15:16:27:16 4 06SEP15:04:18:38 5 23AUG15:19:51:50 4
29AUG15:07:14:18 6 09SEP15:00:20:35 5 04SEP15:14:47:57 4 25AUG15:13:40:57 5
06SEP15:10:06:47 3 05SEP15:20:13:34 4 25AUG15:06:20:25 4 23AUG15:08:43:42 5
;
```

This DATA step creates the variable Time (which will be displayed on the X axis) and creates a duplicate time variable Time2 (which will be used to construct the X2 axis). Although ODS Graphics has sophisticated options for handling datetime data that work well for a single axis, you will use none of them in this example. When you assign a date format to an axis, ODS Graphics behaves differently than it behaves for other types of axes. It can extract the year and display it separately to minimize redundancy in the tick labels (as shown in Figure 2.10). Furthermore, you lose control over tick label thinning. When there is a single date axis, these behaviors are generally good. Because this example has two axes and you will force them to correspond, you will need to do much of the axis labeling work yourself instead of relying on ODS Graphics. You need to begin by finding the range of datetime values.

The following step finds the minimum and maximum datetimes and stores them in a data set:

```
proc means data=x noprint;
   var time;
   output min=min max=max out=m;
run;
```

SAS date variables contain the number of days since January 1, 1960, and SAS datetime variables contain the number of seconds since January 1, 1960. You can use SAS functions and formats to manipulate date and datetime variables. The following step manipulates the minimum and maximum datetimes, and creates four macro variables:

```
data _null_;
   set m;
   call symputx('s'   , 24 * 3600);
   call symputx('min' , dhms(datepart(min)    , 0, 0, 0));
   call symputx('max' , dhms(datepart(max) + 1, 0, 0, 0));
   call symputx('year', year(datepart(min)));
run;
```

The macro variable S contains the number of seconds in a day: $24 \times 60 \times 60$. The macro variable Min contains the SAS datetime value for the beginning of the day that the first time occurred. The macro variable Max contains the SAS datetime value for the beginning of the day after the last time occurred. The macro variable Year contains the year in which the first time occurred.

You can construct a data set and use it to make one or more formats. The following steps create two formats, WKDAY and WKDATE:

```
data c;
   length l1 l2 $ 12;
   retain f1 'wkday' f2 'wkdate' sexcl 'N' eexcl 'Y' ;
   do start = &min to &max by &s;
      end   = start + &s;
      l1    = left(put(datepart(start), downame12.));
      l2    = left(put(start, datetime12.));
      l2    = substr(l2, 1, 2) || propcase(substr(l2, 3, 3));
      output;
   end;
run;

proc format cntlin=c(rename=(l1=label f1=fmtname));
run;

proc format cntlin=c(rename=(l2=label f2=fmtname));
run;
```

This is a one-pass DATA step. There are no input data, so the DATA step ends when it hits the RUN statement. In contrast, DATA steps that have an INPUT or SET statement usually pass through the step many times (usually once for every input observation). When there are multiple passes through the DATA step, the RETAIN statement retains values across passes rather than initializing them to missing each time. The RETAIN statement provides a convenient syntax for creating and initializing constant variables even when retaining is not required.

The first format displays the values in the l1 variable (days of the week). The values come from writing a SAS date value by using the DOWNAME (day-of-the-week name) format. The second format displays the values in the l2 variable (date). The values come from writing a SAS datetime value by using the DATETIME format and extracting some of the results. The PROPCASE function capitalizes the first letter of the month and sets the remaining two letters to lowercase.

The following steps display the input to the two formats:

```
proc print noobs data=c(rename=(l1=label f1=fmtname) drop=f2 l2);
   format start end datetime16.;
run;

proc print noobs data=c(rename=(l2=label f2=fmtname) drop=f1 l1);
   format start end datetime16.;
run;
```

The FORMAT procedure uses the CNTLIN= option to make a format whose name is extracted from the FmtName variable. The other variables control what is displayed for each input value. The data in Figure 2.11 contain a Label variable that contains the values to be displayed (the day of the week) in place of each datetime range, a Start variable that specifies the start of the range, an End variable that specifies the end of the range, an sExcl variable that includes the starting value in the range, and an eExcl variable that excludes the ending value from the range. Datetime values are displayed as *DayMonthYear*:*Hours*:*Minutes*:*Seconds*. The data in Figure 2.11 are similar, but the Label variable contains the date and the FmtName variable contains a different format name. In both cases, the DO statement loops from the starting date to the ending date and creates one observation for every day in the range of days for which there are data.

Figure 2.11 Input to the WKDAY Format Definition

label	fmtname	sexcl	eexcl	start	end
Saturday	wkday	N	Y	22AUG15:00:00:00	23AUG15:00:00:00
Sunday	wkday	N	Y	23AUG15:00:00:00	24AUG15:00:00:00
Monday	wkday	N	Y	24AUG15:00:00:00	25AUG15:00:00:00
Tuesday	wkday	N	Y	25AUG15:00:00:00	26AUG15:00:00:00
Wednesday	wkday	N	Y	26AUG15:00:00:00	27AUG15:00:00:00
Thursday	wkday	N	Y	27AUG15:00:00:00	28AUG15:00:00:00
Friday	wkday	N	Y	28AUG15:00:00:00	29AUG15:00:00:00
Saturday	wkday	N	Y	29AUG15:00:00:00	30AUG15:00:00:00
Sunday	wkday	N	Y	30AUG15:00:00:00	31AUG15:00:00:00
Monday	wkday	N	Y	31AUG15:00:00:00	01SEP15:00:00:00
Tuesday	wkday	N	Y	01SEP15:00:00:00	02SEP15:00:00:00
Wednesday	wkday	N	Y	02SEP15:00:00:00	03SEP15:00:00:00
Thursday	wkday	N	Y	03SEP15:00:00:00	04SEP15:00:00:00
Friday	wkday	N	Y	04SEP15:00:00:00	05SEP15:00:00:00
Saturday	wkday	N	Y	05SEP15:00:00:00	06SEP15:00:00:00
Sunday	wkday	N	Y	06SEP15:00:00:00	07SEP15:00:00:00
Monday	wkday	N	Y	07SEP15:00:00:00	08SEP15:00:00:00
Tuesday	wkday	N	Y	08SEP15:00:00:00	09SEP15:00:00:00
Wednesday	wkday	N	Y	09SEP15:00:00:00	10SEP15:00:00:00
Thursday	wkday	N	Y	10SEP15:00:00:00	11SEP15:00:00:00

Figure 2.12 Input to the WKDATE Format Definition

label	fmtname	sexcl	eexcl	start	end
22Aug	wkdate	N	Y	22AUG15:00:00:00	23AUG15:00:00:00
23Aug	wkdate	N	Y	23AUG15:00:00:00	24AUG15:00:00:00
24Aug	wkdate	N	Y	24AUG15:00:00:00	25AUG15:00:00:00
25Aug	wkdate	N	Y	25AUG15:00:00:00	26AUG15:00:00:00
26Aug	wkdate	N	Y	26AUG15:00:00:00	27AUG15:00:00:00
27Aug	wkdate	N	Y	27AUG15:00:00:00	28AUG15:00:00:00
28Aug	wkdate	N	Y	28AUG15:00:00:00	29AUG15:00:00:00
29Aug	wkdate	N	Y	29AUG15:00:00:00	30AUG15:00:00:00
30Aug	wkdate	N	Y	30AUG15:00:00:00	31AUG15:00:00:00
31Aug	wkdate	N	Y	31AUG15:00:00:00	01SEP15:00:00:00
01Sep	wkdate	N	Y	01SEP15:00:00:00	02SEP15:00:00:00
02Sep	wkdate	N	Y	02SEP15:00:00:00	03SEP15:00:00:00
03Sep	wkdate	N	Y	03SEP15:00:00:00	04SEP15:00:00:00
04Sep	wkdate	N	Y	04SEP15:00:00:00	05SEP15:00:00:00
05Sep	wkdate	N	Y	05SEP15:00:00:00	06SEP15:00:00:00
06Sep	wkdate	N	Y	06SEP15:00:00:00	07SEP15:00:00:00
07Sep	wkdate	N	Y	07SEP15:00:00:00	08SEP15:00:00:00
08Sep	wkdate	N	Y	08SEP15:00:00:00	09SEP15:00:00:00
09Sep	wkdate	N	Y	09SEP15:00:00:00	10SEP15:00:00:00
10Sep	wkdate	N	Y	10SEP15:00:00:00	11SEP15:00:00:00

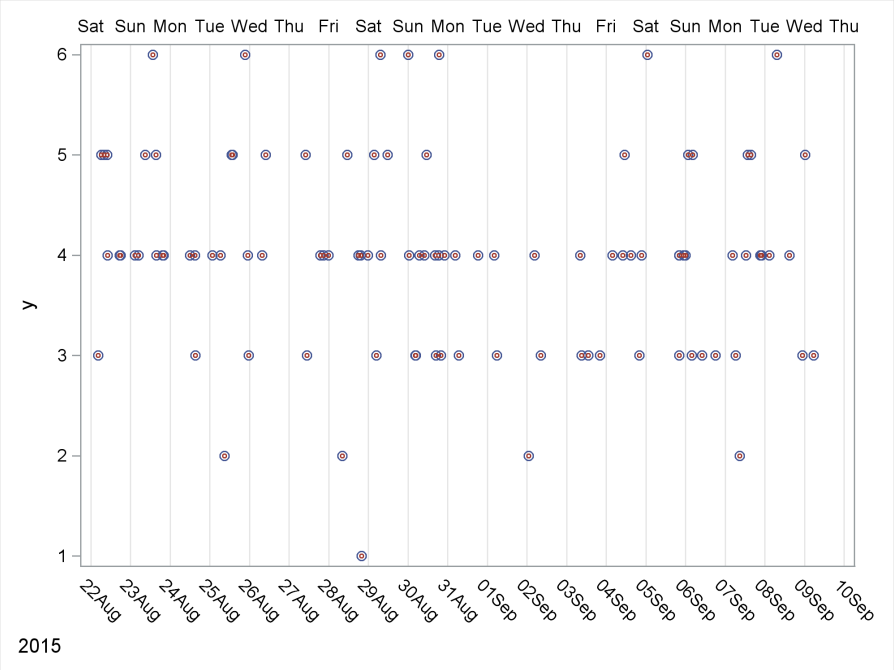
The following step checks the logic of the code so far to ensure that the X and X2 axes correspond as planned:

```
proc sgplot data=x noautolegend;
   scatter y=y x=time;
   scatter y=y x=time2 / x2axis markerattrs=(size=3px);
   format time wkdate5. time2 wkday3.;
   xaxis grid display=(nolabel noticks) values=(&min to &max by &s)
             fitpolicy=rotate;
   x2axis    display=(nolabel noticks) values=(&min to &max by &s)
             fitpolicy=none;
   footnote justify=left "&year";
run;
```

The NOAUTOLEGEND option in the PROC SGPLOT statement suppresses the legend. The first SCATTER statement creates the graph of interest, and it uses the X and Y axes. The second SCATTER statement is used to check results (for now) and create the X2 axis. For now, the MARKERATTRS=(SIZE=3PX) option in the second SCATTER statement displays small markers that should occupy the same position as the actual markers. These markers are removed in the next step. The first graph uses the Time variable, and the second graph uses the identical Time2 variable. The X and X2 axes both have the same underlying tick values because both have the same VALUES= option that specifies ticks that range from the first day to the last day by one-day increments. The X and X2 axes have different tick labels because each X= variable has a different format. For both axes, tick marks are suppressed because their locations are clear from the grid lines. The FITPOLICY= option in the X2AXIS statement displays the values on the X2 axis with no thinning, and the FITPOLICY= option in the XAXIS statement rotates the values on the X axis, again with no thinning. Thinning refers to the process of removing some ticks for axes that would otherwise become congested. The year is displayed in the bottom left of the graph.

The results are displayed in Figure 2.13.

Figure 2.13 Check the Logic (Red Markers Should Be inside Blue Markers)



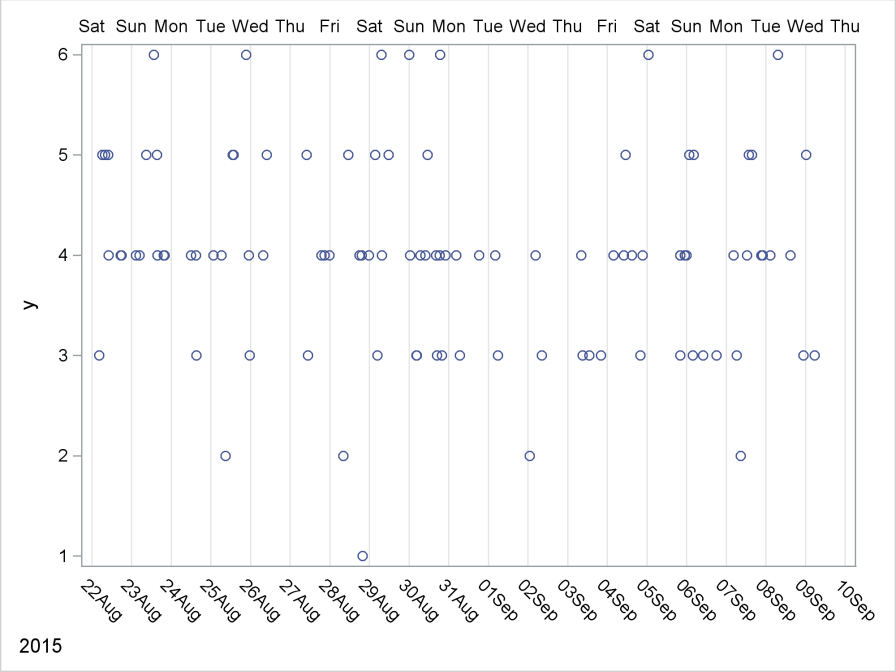
The grid lines show that the ticks align and show the correspondence between the day and date: August 22, 2015, is a Saturday; ...; August 30 is a Sunday; ...; and September 10 is a Thursday. The small red (**Graphdata2**) markers all appear inside the larger blue (**Graphdata1**) markers, showing that the X and X2 axes are set up correctly.

The following step makes the final graph:

```
proc sgplot data=x noautolegend;
   scatter y=y x=time;
   scatter y=y x=time2 / x2axis markerattrs=(size=0px);
   format time wkdate5. time2 wkday3.;
   xaxis grid display=(nolabel noticks) values=(&min to &max by &s)
             fitpolicy=rotate;
   x2axis     display=(nolabel noticks) values=(&min to &max by &s)
             fitpolicy=none;
   footnote justify=left "&year";
run;
```

The results are displayed in Figure 2.14.

Figure 2.14 Days on the X2 Axis and Dates on the X Axis



Now there is one set of markers. The second SCATTER statement makes an invisible plot (zero-size markers), but it populates the X2 axis.

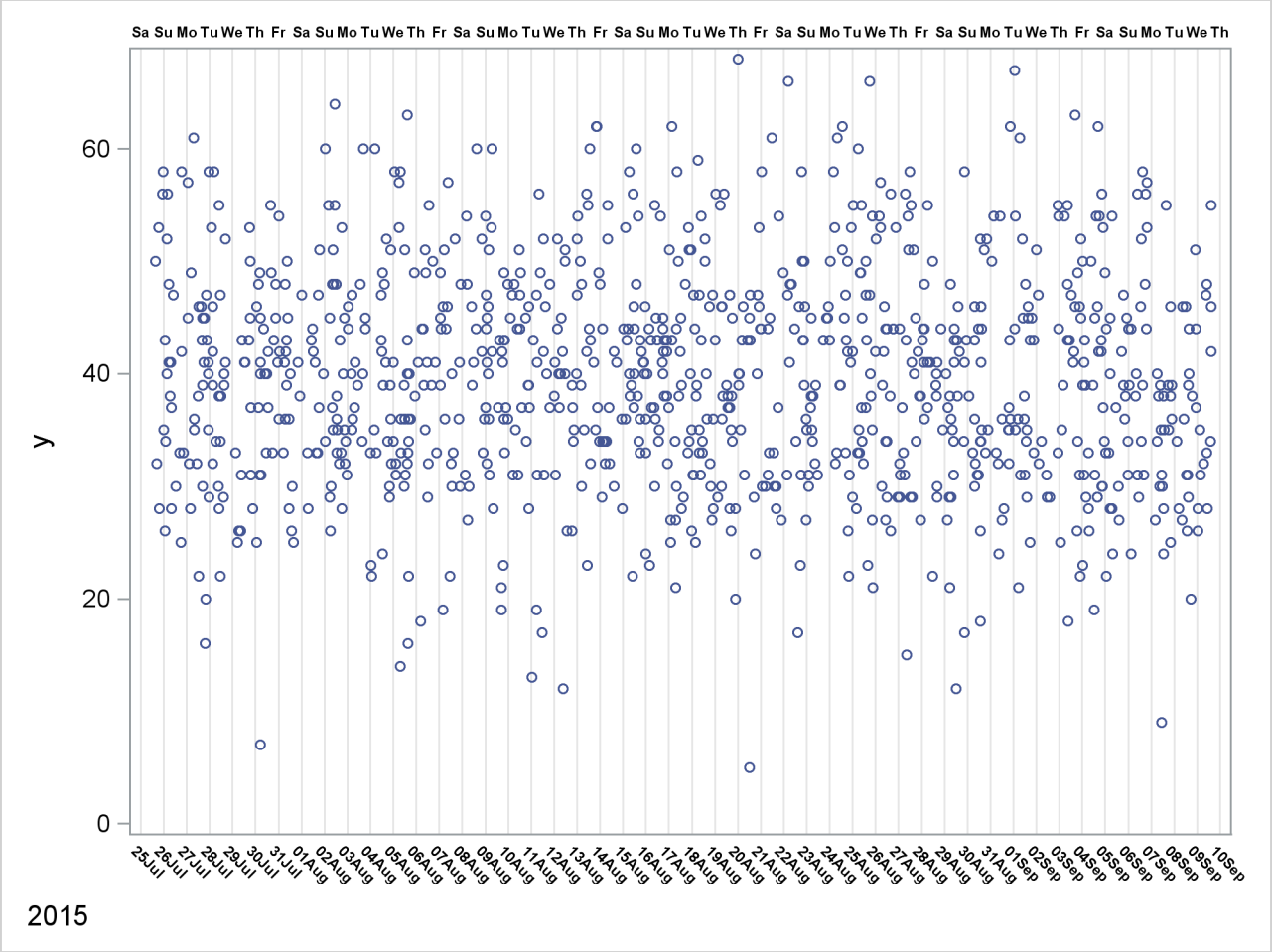
You can make the fonts smaller and display fewer characters on the X2 axis in order to display more data. The following steps generate some random data and illustrate:

```
%let s = %eval(24 * 3600);

data x(drop=i);
   do i = 1 to 1000;
      time  = '10SEP2015:15:15:34'dt - 47 * &s + floor(46 * &s * uniform(368));
      time2 = time;
      y     = round(10 * (4 + normal(368)));
      output;
   end;
run;

proc means data=x noprint;
   var time;
   output min=min max=max out=m;
run;

data _null_;
   set m;
   call symputx('min' , dhms(datepart(min)    , 0, 0, 0));
   call symputx('max' , dhms(datepart(max) + 1, 0, 0, 0));
   call symputx('year', year(datepart(min)));
run;
```

```
data c;
   length l1 l2 $ 12;
   retain f1 'wkday' f2 'wkdate' sexcl 'N' eexcl 'Y' ;
   do start = &min to &max by &s;
      end   = start + &s;
      l1    = left(put(datepart(start), downame12.));
      l2    = left(put(start, datetime12.));
      l2    = substr(l2, 1, 2) || propcase(substr(l2, 3, 3));
      output;
   end;
run;

proc format cntlin=c(rename=(l1=label f1=fmtname));
run;

proc format cntlin=c(rename=(l2=label f2=fmtname));
run;

%let o = display=(nolabel noticks) values=(&min to &max by &s)
         valueattrs=(size=7px weight=bold);
proc sgplot data=x noautolegend;
   scatter y=y x=time  /        markerattrs=(size=5px);
   scatter y=y x=time2 / x2axis markerattrs=(size=0px);
   format time wkdate5. time2 wkday2.;
   xaxis grid &o fitpolicy=rotate;
   x2axis     &o fitpolicy=none;
   footnote justify=left "&year";
run;
```

The marker size is decreased in the first SCATTER statement. Decreasing the marker size is not required, but as graphs become more congested, smaller markers often look better. Some axis options are stored in a macro variable to minimize typing. This too is not required. The VALUEATTRS=(SIZE=7PX WEIGHT=BOLD) option decreases the size of the tick labels but makes them bold so that they are easier to read. The width of the WKDAY format is decreased from 3 to 2. The results are displayed in Figure 2.15.

Figure 2.15 Two Aligned X Axes, Many Ticks, and Smaller Tick Labels



Broken Axes

Double Click for Example Code

This example illustrates broken axes, which are useful when there are extreme observations that compress relevant patterns when you use ordinary axes. To illustrate, the following step creates some artificial data that have three groups:

```
data x(drop=i);
   do g = 1 to 3;
      do i = 1 to 100;
         x = 10 * uniform(7);
         y = 10 ** g + log(x) + sin(g * (x + 1));
         output;
      end;
   end;
run;
```

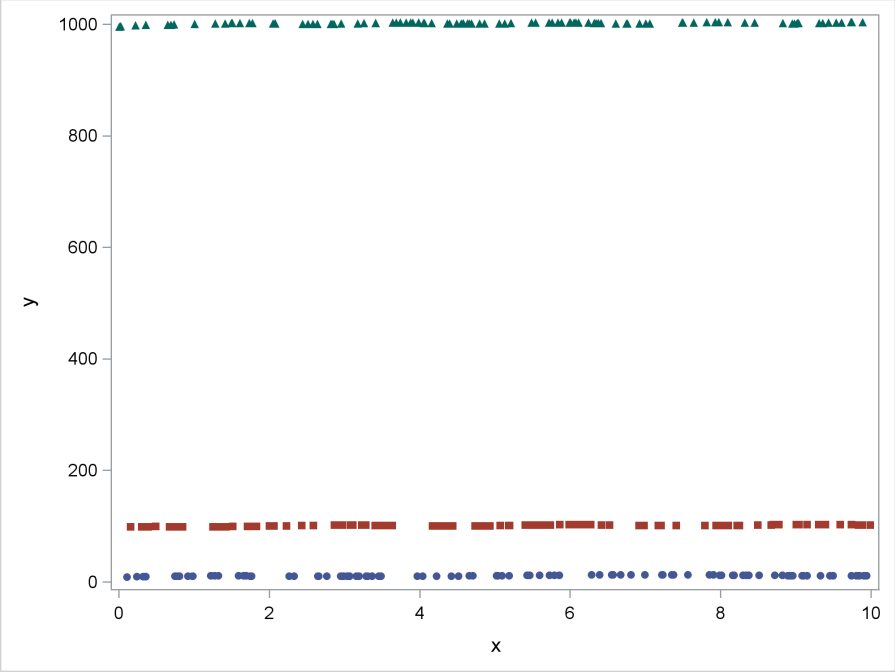
Because of the expression `10 ** g`, Y values in the first group are of the order 10, Y values in the second group are of the order 100, and Y values in the third group are of the order 1,000.

The following step creates an ordinary scatter plot:

```
ods graphics on / attrpriority=none;
proc sgplot data=x noautolegend;
   styleattrs datasymbols=(circlefilled squarefilled trianglefilled);
   scatter y=y x=x / group=g markerattrs=(size=5px);
run;
```

You can specify the STYLEATTRS statement along with the ATTRPRIORITY=NONE option to distinguish groups by colors and markers. The results are displayed in Figure 2.16. Because of the extreme range of values, the patterns in the three groups are not apparent.

Figure 2.16 Extreme Values Squash the Display of Other Values



You can use PROC MEANS to find the minimum and maximum value in each group:

```
proc means;
   var y;
   class g;
run;
```

The results of this step are displayed in Figure 2.17.

Figure 2.17 Minima and Maxima

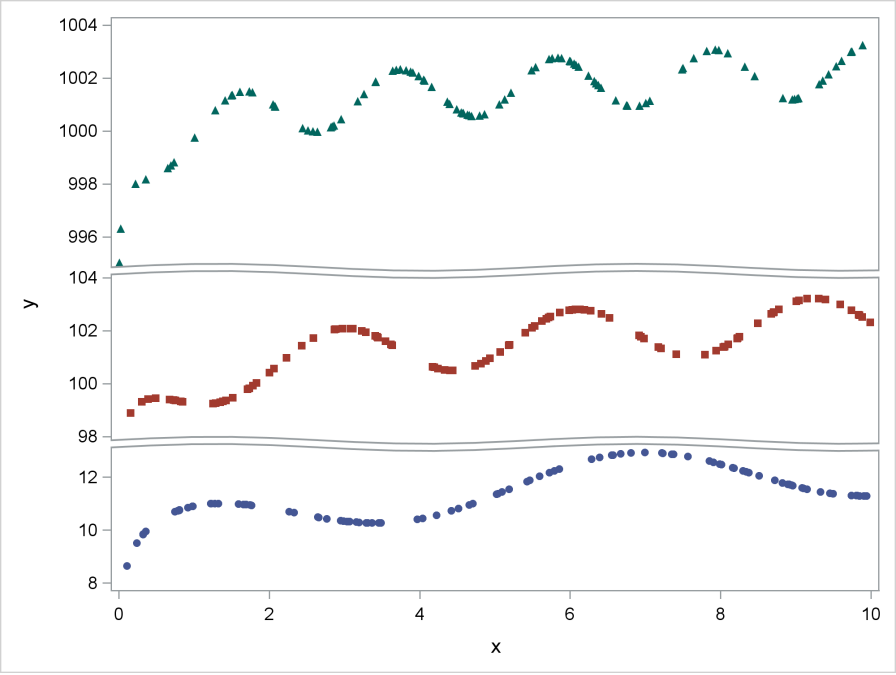
Analysis Variable : y						
g	N Obs	N	Mean	Std Dev	Minimum	Maximum
1	100	100	11.3205312	0.9202651	8.6475467	12.9352554
2	100	100	101.4222774	1.2060505	98.8909941	103.2134315
3	100	100	1001.33	1.3677978	995.0185733	1003.24

The results are used in the next step to specify three value ranges, each of which consists of values smaller than the minimum and larger than the maximum for each group:

```
proc sgplot data=x noautolegend;
   styleattrs datasymbols=(circlefilled squarefilled trianglefilled);
   scatter y=y x=x / group=g markerattrs=(size=5px);
   yaxis   ranges=(8 - 13    98 - 104   995 - 1004);
run;
```

The results are displayed in Figure 2.18.

Figure 2.18 Broken Axes

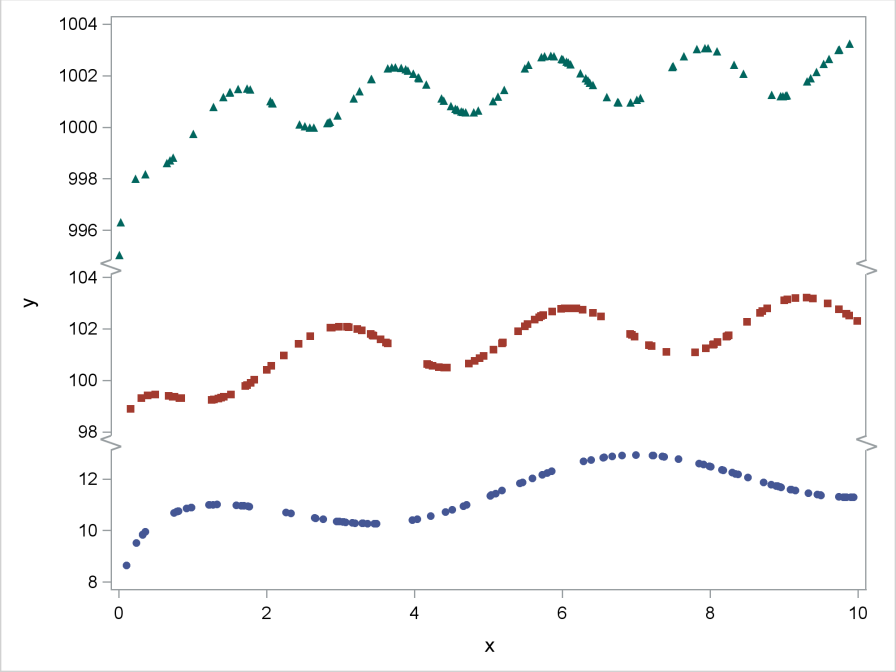


Now the three patterns are clear. Also, the broken axis makes it clear that parts of the graph are omitted. The breaks are slightly curved rather than linear to show that there is one broken axis and not three separate graphs. Other options for how to break the axes are as follows: AXISBREAK=BRACKET | NOTCH | SLANTEDLEFT | SLANTEDRIGHT | SQUIGGLE | SPARK | Z. The next step illustrates specifying AXISBREAK=SPARK in the STYLEATTRS statement:

```
proc sgplot data=x noautolegend;
   styleattrs datasymbols=(circlefilled squarefilled trianglefilled)
              axisbreak=spark;
   scatter y=y x=x / group=g markerattrs=(size=5px);
   yaxis   ranges=(8 - 13    98 - 104   995 - 1004);
run;
```

The results are displayed in Figure 2.19.

Figure 2.19 Broken Axes



Multiple Plots with Equated Axes

Double Click for Example Code

This example shows you how to construct a panel that contains multiple plots, each having equated axes (a centimeter on the Y axis represents the same data range as a centimeter on the X axis). Furthermore, in the final graph, a centimeter on every X axis and Y axis represents the same data range as a centimeter on every other X axis and Y axis.

This example begins by using PROC CANCORR to output two sets of canonical variables (v1–v6 and w1–w6):

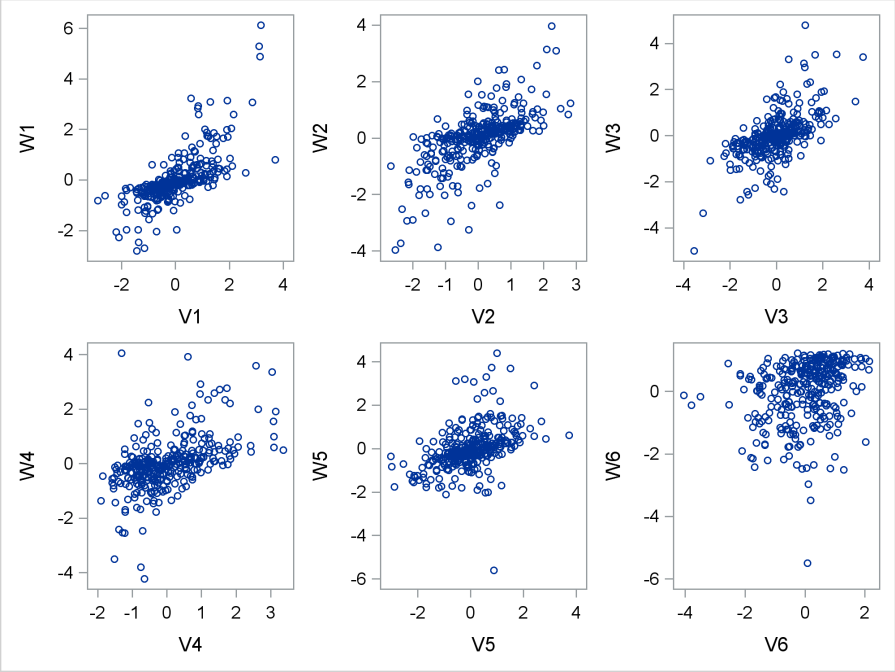
```
proc cancorr data=sashelp.baseball(drop=name) out=c(keep=v: w:);
   var n:;
   with cr:;
run;
```

You can plot each pair of canonical variables by using PROC SGSCATTER:

```
proc sgscatter;
   plot w1*v1 w2*v2 w3*v3 w4*v4 w5*v5 w6*v6;
run;
```

The results are displayed in Figure 2.20

Figure 2.20 Pairwise Plots of Canonical Variables



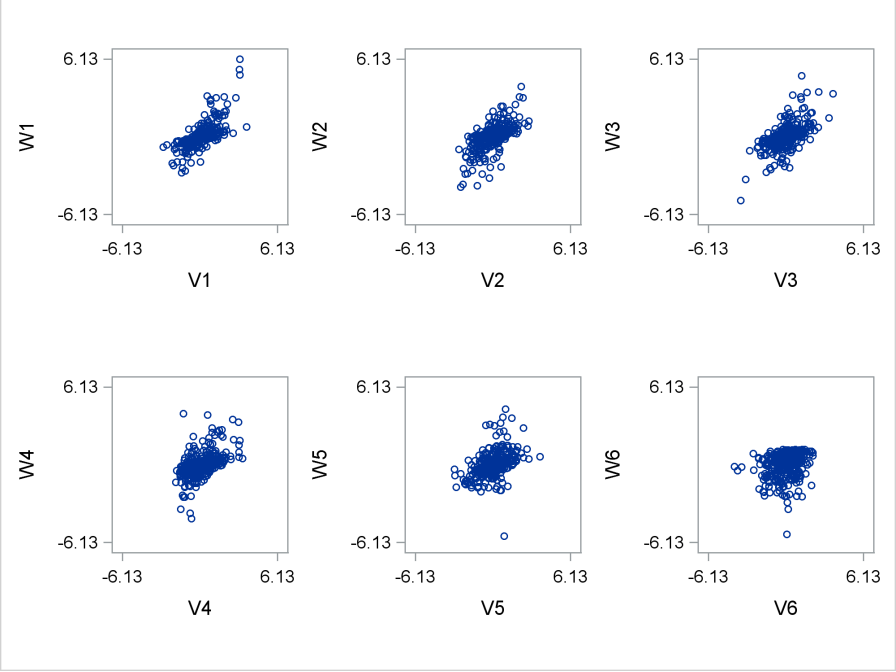
For this particular application, this graph could be improved. Canonical variables are standardized to mean 0 and variance 1. Because all the variables are on the same scale, it would be nice if all the axes were equated within and across the plots. It is easy to use the SG procedures to make the overall size of the graph space (which includes the data area, ticks, axis labels, titles, and footnotes) square and make the ticks the same. However, for truly equated plots, you must use the GTL. These steps extract the maximum absolute value, *t*, from both sets of canonical variables and use it to create a common set of equated axes that extend from –*t* to *t*. The following steps create an intermediate graph that shows the axis ranges:

```
data _null_;
   retain m 0;
   set c end=eof;
   m = max(m, max(of v: w:), abs(min(of v: w:)));
   if eof then call symputx('t', m);
run;

%let nplots = 6;
%let nrows  = 2;
%let ncols  = 3;
%let height = 480px;
```

```
proc template;
   define statgraph canvars;
      begingraph / designheight=&height;
         layout lattice / columns=&ncols rows=&nrows
                          rowgutter=10 columngutter=10;
            %macro plot;
               %do i = 1 %to &nplots;
                  layout overlayequated / equatetype=square
                     commonaxisopts=(viewmin=-&t viewmax=&t
                                     tickvaluelist=(-&t &t));
                     scatterplot y = w&i x = v&i;
                  endlayout;
               %end;
            %mend; %plot
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=c template=canvars;
run;
```

The DATA _NULL_ step reads the data set of canonical variables, finds the maximum absolute value across both sets of variables, and outputs that value to a macro variable. The template is explained in detail when the final version is presented. The results are displayed in Figure 2.21.

Figure 2.21 Pairwise Equated Plots of Canonical Variables



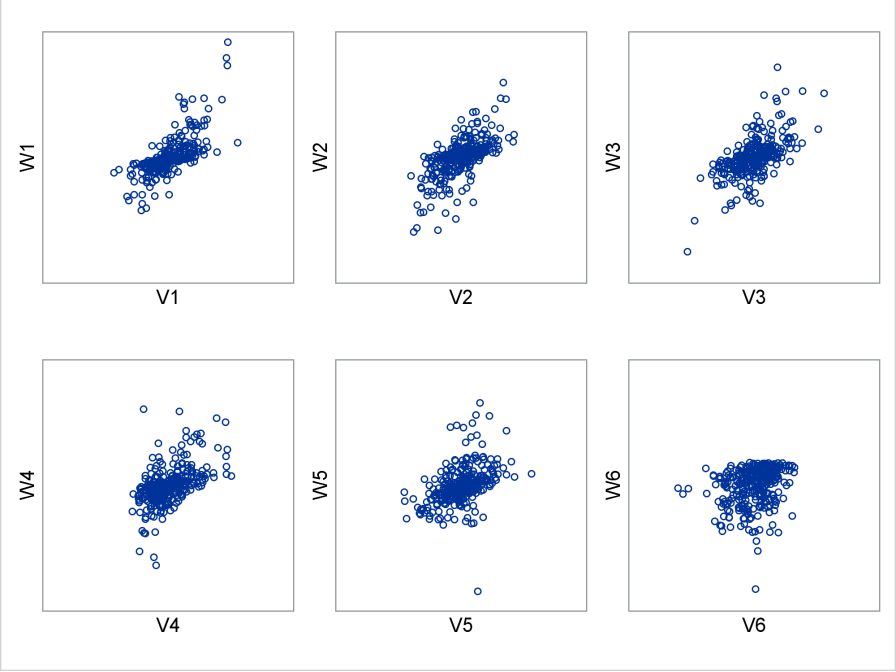
You can see that all graphs are square and all axes range from –6.13 to 6.13. For graphs like these, you do not need to see the tick or tick labels, but they are shown here to ensure that the results are correct.

The following steps make the final graph:

```
proc template;
   define statgraph canvars;
      begingraph / designheight=&height;
         layout lattice / columns=&ncols rows=&nrows
                          rowgutter=10 columngutter=10;
            %macro plot;
               %do i = 1 %to &nplots;
                  layout overlayequated / equatetype=square
                     yaxisopts=(display=(label)) xaxisopts=(display=(label))
                     commonaxisopts=(viewmin=-&t viewmax=&t
                                     tickvaluelist=(-&t &t));
                     scatterplot y = w&i x = v&i;
                  endlayout;
               %end;
            %mend; %plot
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=c template=canvars;
run;
```

The results are displayed in Figure 2.22.

Figure 2.22 Final Pairwise Equated Plots of Canonical Variables



The template relies on the macro variables and DATA _NULL_ step shown previously. The template creates a lattice that has three columns and two rows. Row and column gutter space provide separation between the graphs. The template has six equated overlays (one for each graph), which are constructed by using a macro to minimize typing. Each graph is specified as EQUATETYPE=SQUARE. The DISPLAY=(LABEL) option suppresses the ticks and tick labels for each graph, displaying only the axis label and providing more space for the graphs. The common axis options ensure that all axes extend from $-t$ to t. Specifying the desired tick value list without specifying the VIEWMIN= and VIEWMAX= options is not sufficient. The SCATTERPLOT statement creates the scatter plot.

You could make the code more flexible by finding the minimum and maximum and using those to scale the axes. However, for standardized variables such as these, the simpler approach is reasonable. You could increase the design height in the BEGINGRAPH statement and make 3×3 or 4×4 displays. You could decrease the design width (not currently specified) in the BEGINGRAPH statement and make 2×2 displays.

Chapter 3
Axis Tables

Contents

An axis table displays a table of values together with a graph. Examples of the axis table include the inertia table displayed by PROC CORRESP (see the first graph in Figure 3.1), the Kaplan-Meier plot with at-risk information displayed by PROC LIFETEST (see the second graph in Figure 3.1), and the studentized residuals and Cook's *D* charts displayed by PROC REG (see Figure 3.3 and Figure 3.4). Either the columns of the table (as in PROC LIFETEST) or the rows (as in PROC CORRESP and PROC REG) align with elements of the graph.

Figure 3.1 Axis Table Examples

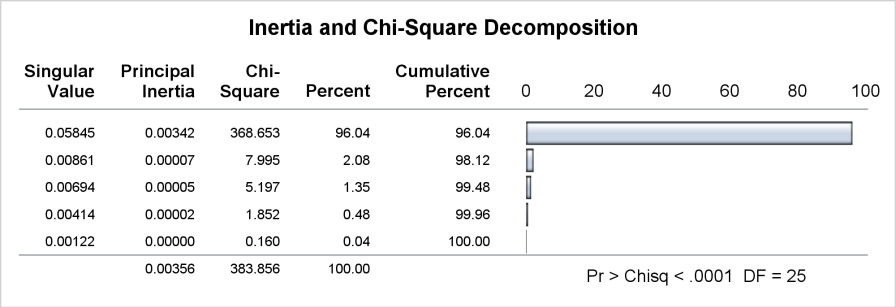
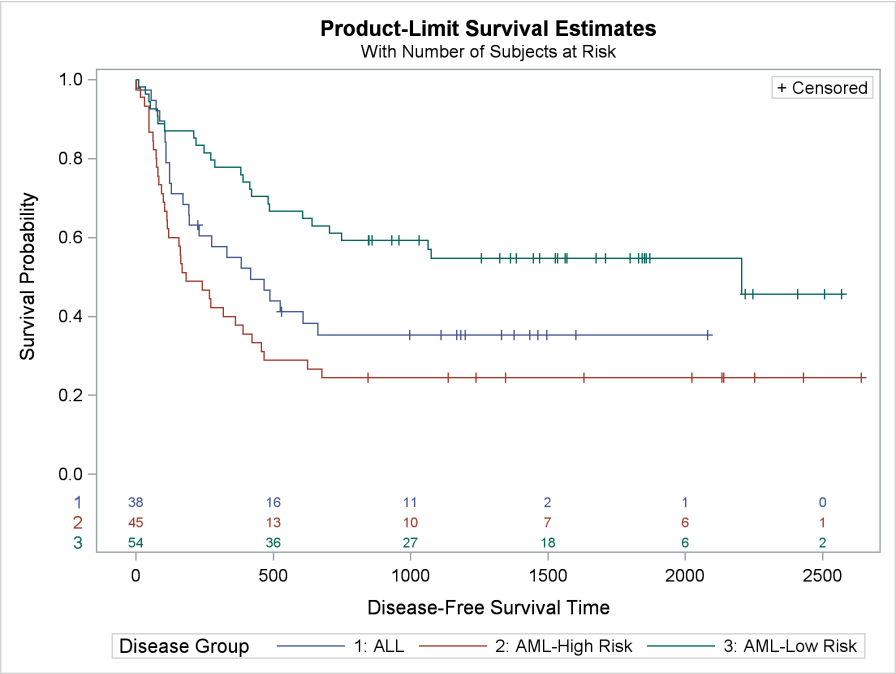


Figure 3.1 *continued*



Axis Table Example Using PROC REG

Double Click for Example Code

This example uses PROC REG to introduce the axis table. When ODS Graphics is enabled and a table of residuals and other observationwise statistics is requested, PROC REG produces the residual table (shown in Figure 3.2) and the graphical display of studentized residuals and Cook's *D* (shown in Figure 3.3). The following step illustrates:

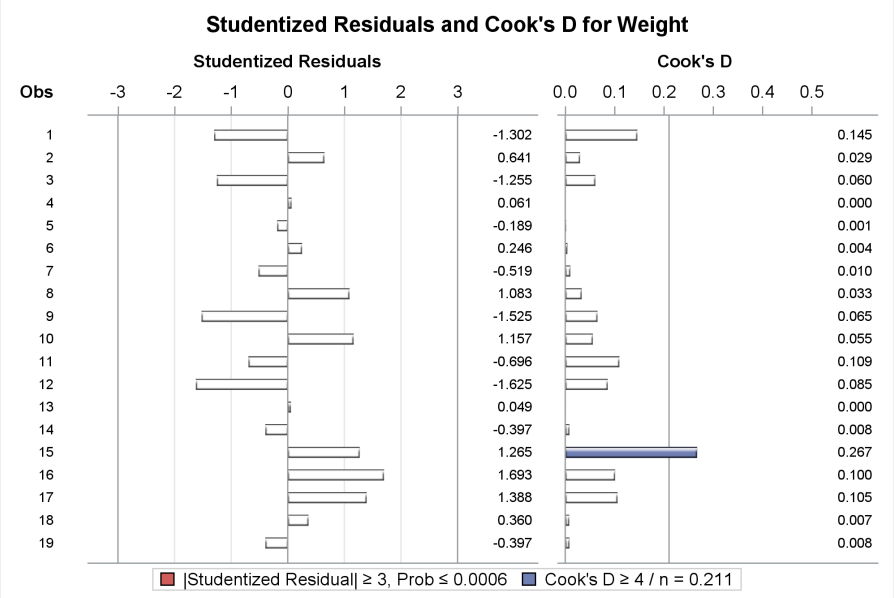
```
ods graphics on;
proc reg data=sashelp.class;
   model weight = height / r;
quit;
```

Figure 3.2 Residuals Table

Model: MODEL1
Dependent Variable: Weight

Output Statistics								
Obs	Dependent Variable	Predicted Value	Std Error Mean Predict	Residual	Std Error Residual	Student Residual	Cook's D
1	112.5	126.0062	4.2963	-13.5062	10.372	-1.302	0.145
2	84.0	77.2683	3.9633	6.7317	10.503	0.641	0.029
3	98.0	111.5798	2.9953	-13.5798	10.819	-1.255	0.060
4	102.5	101.8322	2.5865	0.6678	10.924	0.061	0.000
5	102.5	104.5615	2.6445	-2.0615	10.910	-0.189	0.001
6	83.0	80.3875	3.6593	2.6125	10.613	0.246	0.004
7	84.5	90.1351	2.8892	-5.6351	10.848	-0.519	0.010
8	112.5	100.6625	2.5769	11.8375	10.927	1.083	0.033
9	84.0	100.6625	2.5769	-16.6625	10.927	-1.525	0.065
10	99.5	87.0159	3.0982	12.4841	10.790	1.157	0.055
11	50.5	56.9933	6.2512	-6.4933	9.325	-0.696	0.109
12	90.0	107.6807	2.7676	-17.6807	10.880	-1.625	0.085
13	77.0	76.4885	4.0423	0.5115	10.473	0.049	0.000
14	112.0	116.2586	3.3540	-4.2586	10.714	-0.397	0.008
15	150.0	137.7033	5.6129	12.2967	9.722	1.265	0.267
16	128.0	109.6302	2.8721	18.3698	10.853	1.693	0.100
17	133.0	118.2081	3.5249	14.7919	10.659	1.388	0.105
18	85.0	81.1673	3.5867	3.8327	10.638	0.360	0.007
19	112.0	116.2586	3.3540	-4.2586	10.714	-0.397	0.008

Figure 3.3 Studentized Residuals and Cook's D

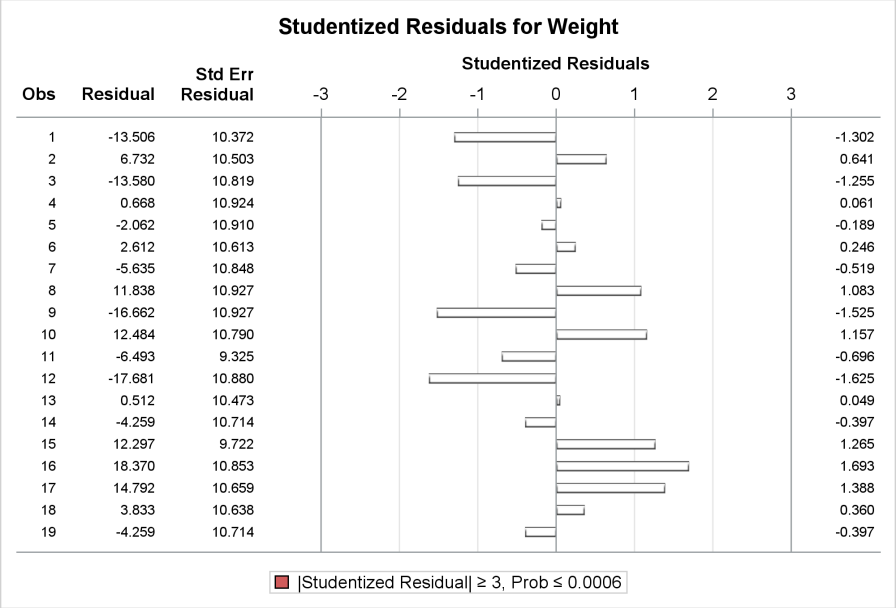


You can use the UNPACK option to display the studentized residuals and Cook's *D* in separate charts as follows:

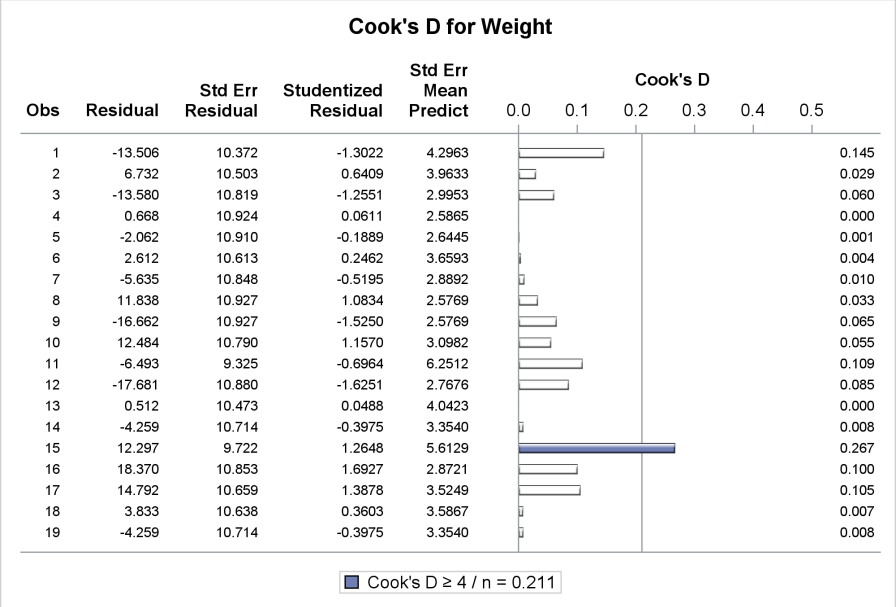
```
proc reg data=sashelp.class plots=residualchart(unpack);
   model weight = height / r;
quit;
```

The results are displayed in Figure 3.4.

Figure 3.4 Separate Charts for Studentized Residuals and Cook's *D*



Obs	Residual	Std Err Residual	
1	-13.506	10.372	-1.302
2	6.732	10.503	0.641
3	-13.580	10.819	-1.255
4	0.668	10.924	0.061
5	-2.062	10.910	-0.189
6	2.612	10.613	0.246
7	-5.635	10.848	-0.519
8	11.838	10.927	1.083
9	-16.662	10.927	-1.525
10	12.484	10.790	1.157
11	-6.493	9.325	-0.696
12	-17.681	10.880	-1.625
13	0.512	10.473	0.049
14	-4.259	10.714	-0.397
15	12.297	9.722	1.265
16	18.370	10.853	1.693
17	14.792	10.659	1.388
18	3.833	10.638	0.360
19	-4.259	10.714	-0.397



Obs	Residual	Std Err Residual	Studentized Residual	Std Err Mean Predict	
1	-13.506	10.372	-1.3022	4.2963	0.145
2	6.732	10.503	0.6409	3.9633	0.029
3	-13.580	10.819	-1.2551	2.9953	0.060
4	0.668	10.924	0.0611	2.5865	0.000
5	-2.062	10.910	-0.1889	2.6445	0.001
6	2.612	10.613	0.2462	3.6593	0.004
7	-5.635	10.848	-0.5195	2.8892	0.010
8	11.838	10.927	1.0834	2.5769	0.033
9	-16.662	10.927	-1.5250	2.5769	0.065
10	12.484	10.790	1.1570	3.0982	0.055
11	-6.493	9.325	-0.6964	6.2512	0.109
12	-17.681	10.880	-1.6251	2.7676	0.085
13	0.512	10.473	0.0488	4.0423	0.000
14	-4.259	10.714	-0.3975	3.3540	0.008
15	12.297	9.722	1.2648	5.6129	0.267
16	18.370	10.853	1.6927	2.8721	0.100
17	14.792	10.659	1.3878	3.5249	0.105
18	3.833	10.638	0.3603	3.5867	0.007
19	-4.259	10.714	-0.3975	3.3540	0.008

The graphs of studentized residuals and Cook's *D* that are displayed in Figure 3.3 and Figure 3.4 are not intended to duplicate the table, although the Cook's *D* chart in Figure 3.4 comes close. This example shows

how you can add a graph to the table in Figure 3.2. The following step creates an ODS output data set r, which contains the information in the table:

```
proc reg data=sashelp.class;
   ods output OutputStatistics=r;
   model weight = height / r;
quit;
```

PROC CONTENTS displays the variable names and labels:

```
proc contents varnum;
   ods select position;
run;
```

The results are displayed in Figure 3.5.

Figure 3.5 Variable Names in the Residuals Table

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	Model	Char	32		
2	Dependent	Char	32		
3	Observation	Num	8	3.	Observation Number
4	DepVar	Num	8	6.1	Dependent Variable
5	PredictedValue	Num	8	D9.3	Predicted Value
6	StdErrMeanPredict	Num	8	D9.3	Std Err Mean Predict
7	Residual	Num	8	D9.3	
8	StdErrResidual	Num	8	D7.3	Std Err Residual
9	StudentResidual	Num	8	7.3	Student Residual
10	Picture	Char	15		-2-1 0 1 2
11	CooksD	Num	8	8.3	Cook's D

The following steps create a template and the table and graph in Figure 3.6:

```
proc template;
   define statgraph StudentizedResidualChart;
      begingraph; * add: / designheight=height (example: / designheight=2000px);
         entrytitle 'Residuals';
         layout overlay / walldisplay=none
            yaxisopts=(display=none type=discrete reverse=true)
            x2axisopts=(label='Studentized Residuals' displaysecondary=(line)
                        labelattrs=graphvaluetext(weight=bold)
                        griddisplay=auto_on linearopts=(integer=true));
            innermargin / align=left gutter=5px;
               %let o = y = observation / xaxis=x2
                        labeljustify=right labelattrs=(weight=bold);
               axistable value=Observation       &o;
               axistable value=DepVar            &o;
               axistable value=PredictedValue    &o;
               axistable value=Residual          &o;
               axistable value=StdErrResidual    &o;
               axistable value=StudentResidual   &o;
               axistable value=StdErrMeanPredict &o;
```

```
            axistable value=CooksD            &o;
         endinnermargin;
         highlowplot y=observation
            low =eval(min(StudentResidual, 0))
            high=eval(max(StudentResidual, 0)) /
            xaxis=x2 dataskin=gloss type=bar barwidth=0.5;
         referenceline x=0 / xaxis=x2;
      endlayout;
   endgraph;
end;
quit;

proc sgrender data=r template=StudentizedResidualChart;
   label Observation = 'Obs' DepVar = 'Weight';
   format PredictedValue Residual StdErrResidual StudentResidual
          StdErrMeanPredict CooksD 6.2;
run;
```

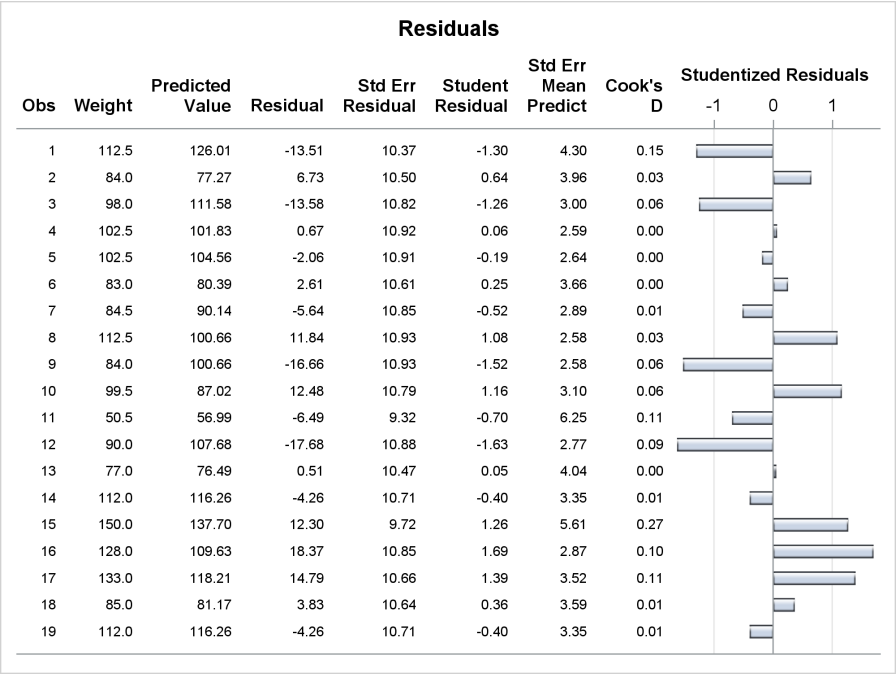
The PROC TEMPLATE step creates the graph template, and the PROC SGRENDER step creates the graph. The HIGHLOWPLOT statement creates the high-low plot. Bars extend from the studentized residual to 0 for negative studentized residuals. Bars extend from 0 to the studentized residual for positive studentized residuals. A reference line is drawn at X=0, which provides a Y axis for the high-low plot. Most graphs have an X axis on the bottom of the graph. In this case, the X2 axis is used at the top, so the tick labels and axis label appear at the top. Additional options control the width and appearance of the bar.

Most of the rest of the template controls the title, axes, and the table. The INNERMARGIN block defines the table. It is aligned in the left of the graphical display, and 5 pixels are inserted after the graph and before the table (GUTTER=5PX). The inner margin block consists of a series of AXISTABLE statements, one for each column in the table. The %LET statement after the INNERMARGIN statement creates a macro variable that contains the options that are common to all of the AXISTABLE statements in order to minimize typing. As in the HIGHLOWPLOT statement, the X2 axis is used, so the headers for the table appear above the columns of numbers. Headers are right-justified and appear in a bold font.

The LAYOUT OVERLAY statement controls the axes. The WALLDISPLAY=NONE option removes the box that surrounds most graphs. Specifically, the goal is to suppress vertical lines on the right and left. The YAXISOPTS= option DISPLAY=NONE suppresses the Y-axis ticks and labels. TYPE=DISCRETE specifies the axis type as discrete (the observation number is the Y-axis variable in all the statements), and REVERSE=TRUE reverses the axis (values decrease as you move up the Y axis). The X2AXISOPTS= options specify the axis label, a secondary line (the line at the bottom of the graph), a bold font for the axis label, grid lines, and integer ticks. No X-axis options are specified.

For most of the columns, the variable label (which comes from the header in the PROC REG table) is displayed above each column. The LABEL statement in the PROC SGRENDER step provides shorter headers for some of the columns, and the FORMAT statement controls the formats of the columns.

Figure 3.6 Axis Table by Using GTL



Obs	Weight	Predicted Value	Residual	Std Err Residual	Student Residual	Std Err Mean Predict	Cook's D
1	112.5	126.01	-13.51	10.37	-1.30	4.30	0.15
2	84.0	77.27	6.73	10.50	0.64	3.96	0.03
3	98.0	111.58	-13.58	10.82	-1.26	3.00	0.06
4	102.5	101.83	0.67	10.92	0.06	2.59	0.00
5	102.5	104.56	-2.06	10.91	-0.19	2.64	0.00
6	83.0	80.39	2.61	10.61	0.25	3.66	0.00
7	84.5	90.14	-5.64	10.85	-0.52	2.89	0.01
8	112.5	100.66	11.84	10.93	1.08	2.58	0.03
9	84.0	100.66	-16.66	10.93	-1.52	2.58	0.06
10	99.5	87.02	12.48	10.79	1.16	3.10	0.06
11	50.5	56.99	-6.49	9.32	-0.70	6.25	0.11
12	90.0	107.68	-17.68	10.88	-1.63	2.77	0.09
13	77.0	76.49	0.51	10.47	0.05	4.04	0.00
14	112.0	116.26	-4.26	10.71	-0.40	3.35	0.01
15	150.0	137.70	12.30	9.72	1.26	5.61	0.27
16	128.0	109.63	18.37	10.85	1.69	2.87	0.10
17	133.0	118.21	14.79	10.66	1.39	3.52	0.11
18	85.0	81.17	3.83	10.64	0.36	3.59	0.01
19	112.0	116.26	-4.26	10.71	-0.40	3.35	0.01

You can use the following statements to display the templates that PROC REG uses to make the three different versions of the residual charts:

```
proc template;
   source Stat.REG.Graphics.StudResCooksDChart;
   source Stat.REG.Graphics.StudentResChart;
   source Stat.REG.Graphics.CooksDChart;
quit;
```

The results of this step are not shown. However, the template used in this example contains many of the elements found in the PROC REG templates. Procedure templates are often complicated because they need to handle a wide variety of results. In addition, although the procedure developer might have used the macro language to simplify template development, no macros or macro variables are ever displayed in the template source. The PROC REG template contains the DESIGNHEIGHT=HEIGHT option in the BEGINGRAPH statement, where Height is a dynamic variable that is set by the procedure. For most graphs, the size of the graph does not need to vary as a function of the data set size, but it does need to vary for this graph, which displays one row for each input data set observation. For larger data sets, you could directly specify a height or set it by using a macro variable. This graph is useful only for small data sets.

You can make a similar plot by using a DATA step and PROC SGPLOT as follows:

```
data r2;
   set r;
   low  = min(StudentResidual, 0);
   high = max(StudentResidual, 0);
   format PredictedValue Residual StdErrResidual StudentResidual
          StdErrMeanPredict CooksD 6.2;
   label  Observation = 'Obs' DepVar = 'Weight' high = 'Studentized Residuals';
run;
```

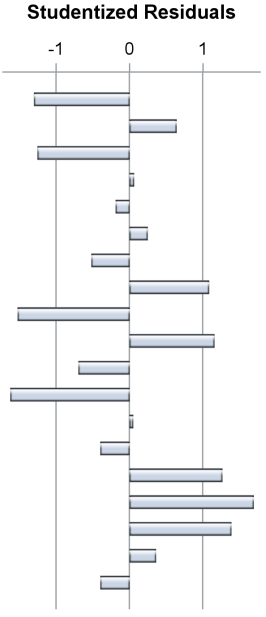
```
proc sgplot data=r2 noborder;
   title 'Residuals';
   yaxistable Observation DepVar PredictedValue Residual StdErrResidual
              StudentResidual StdErrMeanPredict CooksD /
              y=observation position=left labeljustify=right
              labelattrs=(weight=bold);
   refline    -1 0 1 / axis=x2;
   highlow    y=observation low=low high=high /
              dataskin=gloss type=bar barwidth=0.5 x2axis;
   x2axis     labelattrs=(weight=bold);
   yaxis      display=none reverse offsetmin=0 offsetmax=0;
run;
```

Because PROC SGPLOT does not support expressions, you must use a DATA step to compute the end points for each bar. You can also set the formats and provide labels in the DATA step and use PROC SGPLOT to make the graph. The NOBORDER option suppresses the border around the high-low plot. The YAXISTABLE statement creates the axis table for the named variables. The Y variable is observation, the axis table is positioned to the left of the graph, and right-justified and bold headers appear over each column. The HIGHLOW statement creates the graph; its syntax is similar to the HIGHLOWPLOT statement in the GTL. The REFLINE statement creates the vertical reference lines. The X2AXIS statement displays a bold header (axis label) over the graph. The Y axis is reversed, and the DISPLAY=NONE option suppresses the tick marks and labels (the observation numbers). The zero offsets on the Y axis minimize the space between the column headers and the body of the graph and table. The results are displayed in Figure 3.7.

Figure 3.7 Axis Table with X Axis Produced by PROC SGPLOT

Residuals

Obs	Weight	Predicted Value	Residual	Std Err Residual	Student Residual	Std Err Mean Predict	Cook's D
1	112.5	126.01	-13.51	10.37	-1.30	4.30	0.15
2	84.0	77.27	6.73	10.50	0.64	3.96	0.03
3	98.0	111.58	-13.58	10.82	-1.26	3.00	0.06
4	102.5	101.83	0.67	10.92	0.06	2.59	0.00
5	102.5	104.56	-2.06	10.91	-0.19	2.64	0.00
6	83.0	80.39	2.61	10.61	0.25	3.66	0.00
7	84.5	90.14	-5.64	10.85	-0.52	2.89	0.01
8	112.5	100.66	11.84	10.93	1.08	2.58	0.03
9	84.0	100.66	-16.66	10.93	-1.52	2.58	0.06
10	99.5	87.02	12.48	10.79	1.16	3.10	0.06
11	50.5	56.99	-6.49	9.32	-0.70	6.25	0.11
12	90.0	107.68	-17.68	10.88	-1.63	2.77	0.09
13	77.0	76.49	0.51	10.47	0.05	4.04	0.00
14	112.0	116.26	-4.26	10.71	-0.40	3.35	0.01
15	150.0	137.70	12.30	9.72	1.26	5.61	0.27
16	128.0	109.63	18.37	10.85	1.69	2.87	0.10
17	133.0	118.21	14.79	10.66	1.39	3.52	0.11
18	85.0	81.17	3.83	10.64	0.36	3.59	0.01
19	112.0	116.26	-4.26	10.71	-0.40	3.35	0.01

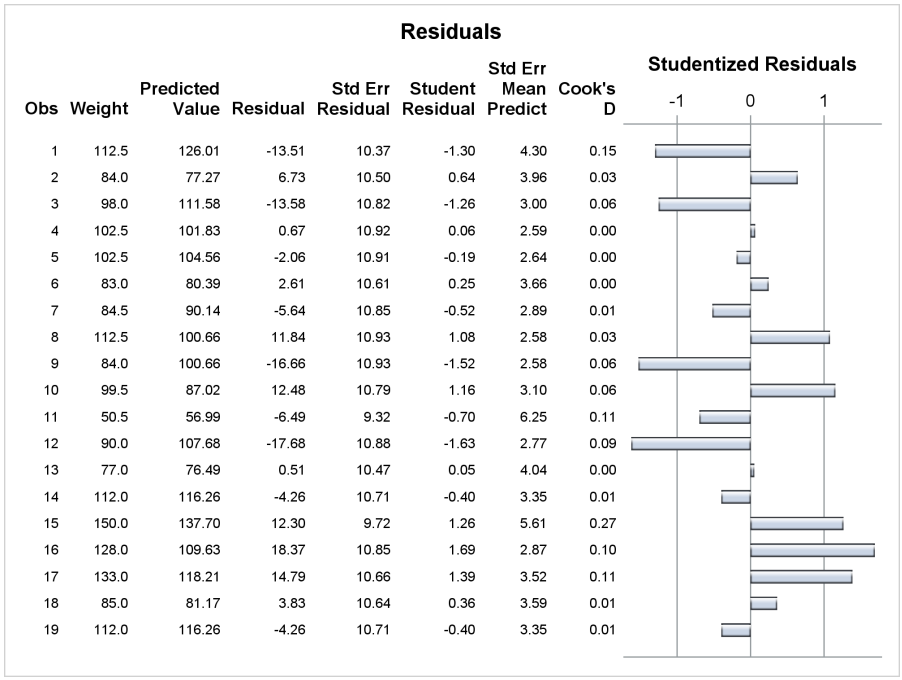


You can add an X-axis line below the graph as follows:

```
proc sgplot data=r2 noborder noautolegend;
   title 'Residuals';
   yaxistable Observation DepVar PredictedValue Residual StdErrResidual
              StudentResidual StdErrMeanPredict CooksD /
              y=observation position=left labeljustify=right
              labelattrs=(weight=bold);
   refline    -1 0 1 / axis=x2;
   scatter    y=observation x=low / markerattrs=(size=0);
   highlow    y=observation low=low high=high /
              dataskin=gloss type=bar barwidth=0.5 x2axis;
   xaxis      display=(nolabel noticks novalues);
   x2axis     labelattrs=(weight=bold);
   yaxis      display=none reverse offsetmin=0 offsetmax=0;
run;
```

The XAXIS statement suppresses labels, ticks, and values so that only a line is displayed. You can specify a SCATTER statement to add an invisible plot that uses the X axis so that the X-axis line is displayed. The NOAUTOLEGEND option in the PROC SGPLOT statement suppresses the legend. The results are displayed in Figure 3.8.

Figure 3.8 Axis Table Produced by PROC SGPLOT



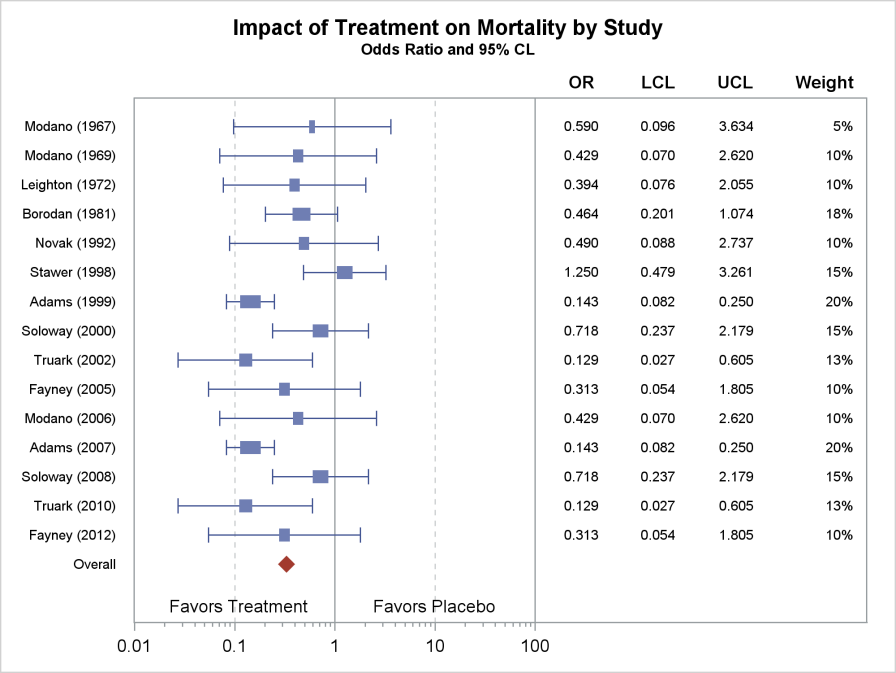
Obs	Weight	Predicted Value	Residual	Std Err Residual	Student Residual	Std Err Mean Predict	Cook's D
1	112.5	126.01	-13.51	10.37	-1.30	4.30	0.15
2	84.0	77.27	6.73	10.50	0.64	3.96	0.03
3	98.0	111.58	-13.58	10.82	-1.26	3.00	0.06
4	102.5	101.83	0.67	10.92	0.06	2.59	0.00
5	102.5	104.56	-2.06	10.91	-0.19	2.64	0.00
6	83.0	80.39	2.61	10.61	0.25	3.66	0.00
7	84.5	90.14	-5.64	10.85	-0.52	2.89	0.01
8	112.5	100.66	11.84	10.93	1.08	2.58	0.03
9	84.0	100.66	-16.66	10.93	-1.52	2.58	0.06
10	99.5	87.02	12.48	10.79	1.16	3.10	0.06
11	50.5	56.99	-6.49	9.32	-0.70	6.25	0.11
12	90.0	107.68	-17.68	10.88	-1.63	2.77	0.09
13	77.0	76.49	0.51	10.47	0.05	4.04	0.00
14	112.0	116.26	-4.26	10.71	-0.40	3.35	0.01
15	150.0	137.70	12.30	9.72	1.26	5.61	0.27
16	128.0	109.63	18.37	10.85	1.69	2.87	0.10
17	133.0	118.21	14.79	10.66	1.39	3.52	0.11
18	85.0	81.17	3.83	10.64	0.36	3.59	0.01
19	112.0	116.26	-4.26	10.71	-0.40	3.35	0.01

Creating a Forest Plot Using PROC SGPLOT

Double Click for Example Code

This example, which is based on an example from Matange and Heath (2011), shows you how to make a forest plot (see Figure 3.9). Forest plots are used to display the results of a meta-analysis and are typically used in medical and pharmaceutical research.

Figure 3.9 Forest Plot



This example has three parts, each of which uses PROC SGPLOT:

- "Using an Axis Table to Make a Forest Plot" on page 43. This part shows the easiest way to make the forest plot: by using the latest capabilities of PROC SGPLOT, which includes the YAXISTABLE statement.
- "Background: Graphs of Constant Variables" on page 48. This part provides background for the third part and shows how to display columns of values by plotting constant variables.
- "Using Constant Variables to Make a Forest Plot" on page 54. This part uses constant variables to make the axis table in a forest plot.

Before SAS 9.4, the only way you could use PROC SGPLOT to make graphs like the forest plot was by using the constant-variable approach. Both the axis-table method and the constant-variable method are explained in this example because both illustrate important tools in graph construction. The constant-variable example also shows other older techniques such as using a SCATTER statement to display error bars rather than using a HIGHLOW statement. Furthermore, it uses the CNTLIN= data set in PROC FORMAT to generate a format from a data set.

Using an Axis Table to Make a Forest Plot

This section shows you how to use a YAXISTABLE statement to construct a forest plot. The following step reads the data set:

```
data forest;
   input Study $1-16 OR LCL UCL Weight;
   format weight percent5. OR LCL UCL 5.3;
datalines;
Modano (1967)     0.590 0.096 3.634  1
Modano (1969)     0.429 0.070 2.620  2
Leighton (1972)   0.394 0.076 2.055  2
Borodan (1981)    0.464 0.201 1.074  3.5
Novak (1992)      0.490 0.088 2.737  2
Stawer (1998)     1.250 0.479 3.261  3
Adams (1999)      0.143 0.082 0.250  4
Soloway (2000)    0.718 0.237 2.179  3
Truark (2002)     0.129 0.027 0.605  2.5
Fayney (2005)     0.313 0.054 1.805  2
Modano (2006)     0.429 0.070 2.620  2
Adams (2007)      0.143 0.082 0.250  4
Soloway (2008)    0.718 0.237 2.179  3
Truark (2010)     0.129 0.027 0.605  2.5
Fayney (2012)     0.313 0.054 1.805  2
Overall           0.328 0.233 0.462  .
;
```

The variable OR contains the odds ratio, LCL contains the lower confidence limit, UCL contains the upper confidence limit, and Weight contains the study weight. The FORMAT statement assigns formats to the specified variables.

You will need some additional variables to display the study names on the Y axis. The following steps create and display these and other new variables (which are explained after the data set is displayed in Figure 3.10):

```
data forest;
   set forest nobs=nobs;
   if _n_ eq 1 then call symputx('nobs', nobs);
   nAll     = nobs + 1 - _n_;                      * 1-16 for everything;
   nStudy   = ifn(study eq 'Overall', .   , nall); * 2-16 for studies;
   nOverall = ifn(study eq 'Overall', nall, .   ); * 1    for overall;
   if n(weight) then do;
      weight = weight * 0.05;       /* Rescale weights      */
      lo = OR / (10 ** (weight/2)); /* Bar width            */
      hi = OR * (10 ** (weight/2)); /* shows study's weight */
   end;
run;

proc print noobs;
run;
```

The first IFN function returns a missing value when study = `'Overall'` and otherwise returns the value of the variable nAll. The results are displayed in Figure 3.10.

Figure 3.10 Forest Plot Format Data

Study	OR	LCL	UCL	Weight	nAll	nStudy	nOverall	lo	hi
Modano (1967)	0.590	0.096	3.634	5%	16	16	.	0.55700	0.62496
Modano (1969)	0.429	0.070	2.620	10%	15	15	.	0.38235	0.48135
Leighton (1972)	0.394	0.076	2.055	10%	14	14	.	0.35115	0.44208
Borodan (1981)	0.464	0.201	1.074	18%	13	13	.	0.37933	0.56757
Novak (1992)	0.490	0.088	2.737	10%	12	12	.	0.43671	0.54979
Stawer (1998)	1.250	0.479	3.261	15%	11	11	.	1.05174	1.48563
Adams (1999)	0.143	0.082	0.250	20%	10	10	.	0.11359	0.18003
Soloway (2000)	0.718	0.237	2.179	15%	9	9	.	0.60412	0.85334
Truark (2002)	0.129	0.027	0.605	13%	8	8	.	0.11171	0.14897
Fayney (2005)	0.313	0.054	1.805	10%	7	7	.	0.27896	0.35119
Modano (2006)	0.429	0.070	2.620	10%	6	6	.	0.38235	0.48135
Adams (2007)	0.143	0.082	0.250	20%	5	5	.	0.11359	0.18003
Soloway (2008)	0.718	0.237	2.179	15%	4	4	.	0.60412	0.85334
Truark (2010)	0.129	0.027	0.605	13%	3	3	.	0.11171	0.14897
Fayney (2012)	0.313	0.054	1.805	10%	2	2	.	0.27896	0.35119
Overall	0.328	0.233	0.462	.	1	.	1	.	.

The data set has three Y-axis variables (nAll, nStudy, and nOverall), each of which contains integers that indicate the number for the relevant study names. The nStudy (15 nonmissing values) and nOverall (one nonmissing value) Y-axis variables are needed because the individual study results are displayed differently from the overall results. Hence, separate plotting statements are needed, and each needs a different Y-axis variable. The nAll variable (no missing values) is used to create the other two variables. The widths of the bars show the weights of the studies. Widths are computed on an exponential scale and are later displayed on a log scale.

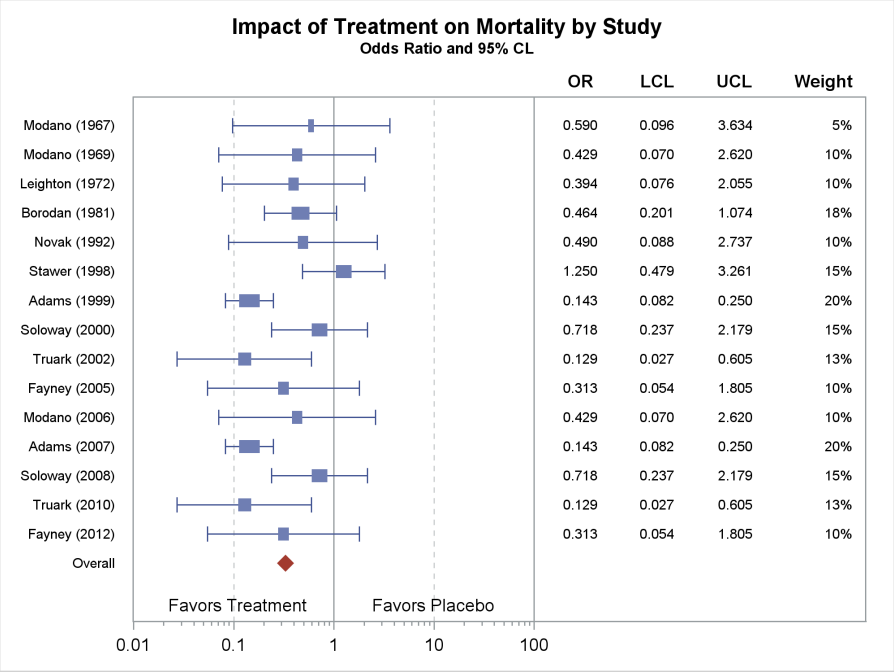
The next step creates the forest plot:

```
title 'Impact of Treatment on Mortality by Study';
title2 h=8pt 'Odds Ratio and 95% CL';

proc sgplot data=forest noautolegend nocycleattrs;
   refline 1 100  / axis=x;
   refline 0.1 10 / axis=x lineattrs=(pattern=shortdash) transparency=0.5;
   highlow y=nStudy low=LCL high=UCL / lowcap=serif highcap=serif
                                       lineattrs=graphdata1;
   scatter y=nOverall x=OR          / markerattrs=graphdata2
                                      (symbol=diamondfilled size=10);
   highlow y=nStudy low=lo high=hi / type=bar intervalbarwidth=.1in
                                      nooutline fillattrs=graphdata1;
   yaxistable study                 / y=nAll position=left location=outside
                                      nolabel valuejustify=right pad=(right=0px);
   yaxistable OR LCL UCL weight     / y=nStudy position=right location=inside
                                      labelhalign=center labelattrs=(weight=bold)
                                      pad=(left=20px right=10px);
   inset '     Favors Treatment'    / position=bottomleft;
   inset 'Favors Placebo      '     / position=bottomright;
   xaxis  type=log min=0.01 max=100 minor display=(nolabel)
          offsetmin=0 offsetmax=0;
   yaxis  display=none offsetmin=0.1 offsetmax=0.05 values=(1 to &nobs);
```

```
run;
```

The NOCYCLEATTRS option in the PROC SGPLOT statement specifies that PROC SGPLOT should not automatically create unique attributes (such as colors) for the elements in the graph. All the attributes are specified in the options. The X axis ranges from MIN=0.01 to MAX=100 and has solid references lines at 1 and 100 and dashed reference lines at 0.1 and 10. The first HIGHLOW statement displays the individual study results as error bars. The SCATTER statement displays the overall results as a filled diamond. The second HIGHLOW statement displays the boxes that show the study weights. The first YAXISTABLE statement displays the study names on the Y axis, outside the graph and to its left. The second YAXISTABLE statement displays the odds ratio, confidence limits, and weight. The inset strings `'Favors Treatment'` and `'Favors Placebo'` label the two parts of the X axis. Labels are placed on the bottom left and bottom right of the graph. (The OFFSETMIN=0.1 option in the YAXIS statement reserves space for those two strings.) In the XAXIS statement, the TYPE=LOG option specifies a log scale and the OFFSETMIN=0 and OFFSETMAX=0 options ensure that there is no extra space beyond the minimum and maximum values (0.01 and 100, respectively). The YAXIS statement specifies the values 1 to 16 (1 to the value of the macro variable &nobs) on the Y axis. However, actual study names are displayed instead of the integers. The results are displayed in Figure 3.11.

Figure 3.11 Forest Plot Produced by Two YAXISTABLE Statements



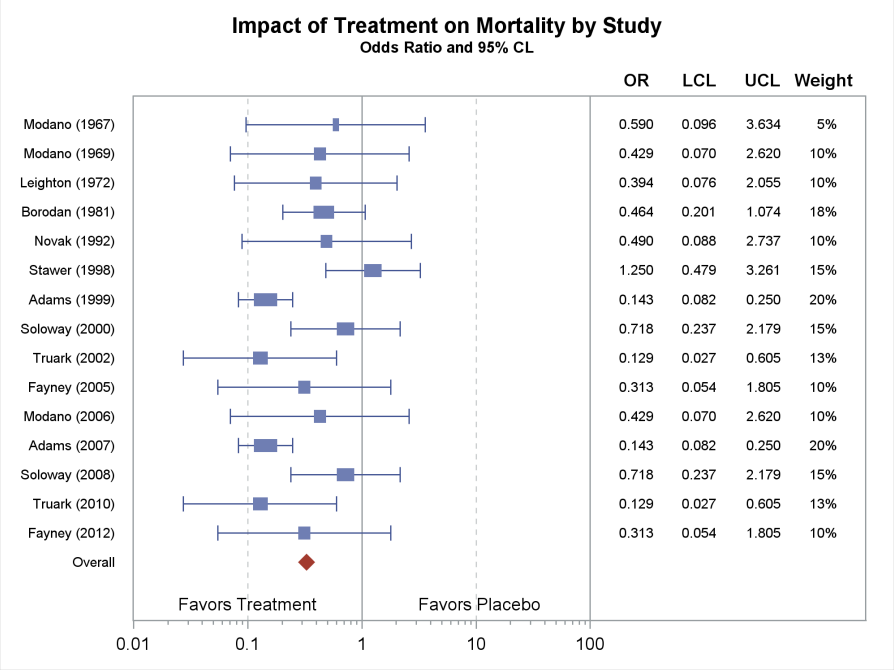
The second YAXISTABLE statement creates an axis table that has four columns and is positioned inside the graph and to the right. The Y-axis variable is nStudy, and headers are centered and bold. Padding controls the space to the left and right of the table. The axis table in the graph in Figure 3.11 could be improved. The following step adjusts the columns so that they are more evenly spaced:

```
proc sgplot data=forest noautolegend nocycleattrs;
   refine 1 100  / axis=x;
   refine 0.1 10 / axis=x lineattrs=(pattern=shortdash) transparency=0.5;
   highlow y=nStudy low=LCL high=UCL / lowcap=serif highcap=serif
                                       lineattrs=graphdata1;
   scatter y=nOverall x=OR / markerattrs=graphdata2
                                        (symbol=diamondfilled size=10);
   highlow y=nStudy low=lo high=hi / type=bar intervalbarwidth=.1in
```

```
                                    nooutline fillattrs=graphdata1;
   yaxistable study      / y=nAll position=left location=outside
                           nolabel valuejustify=right pad=(right=0px);
   yaxistable OR LCL UCL / y=nStudy position=right location=inside
                           labelhalign=center labelattrs=(weight=bold)
                           pad=(left=20px right=0px);
   yaxistable weight     / y=nStudy position=right location=inside
                           labelhalign=center labelattrs=(weight=bold)
                           pad=(left=10px right=10px) valuehalign=center;
   inset '       Favors Treatment' / position=bottomleft;
   inset 'Favors Placebo        '  / position=bottomright;
   xaxis  type=log min=0.01 max=100 minor display=(nolabel)
          offsetmin=0 offsetmax=0;
   yaxis  display=none offsetmin=0.1 offsetmax=0.05 values=(1 to &nobs);
run;
```

The results are displayed in Figure 3.12.

Figure 3.12 Forest Plot Produced by Three YAXISTABLE Statements



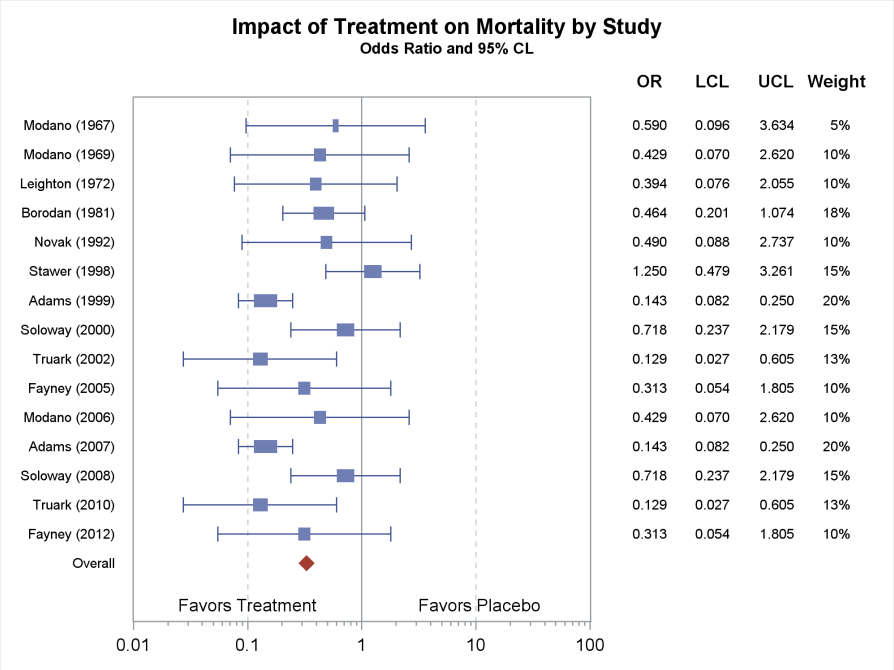
The default spacing of the columns depends on the widths of the values and column headers. The first three columns (OR, LCL, and UCL) have a uniform width and headers that are less wide than the values, but the Weight variable is different. The graph in Figure 3.12 is created by separating the Weight variable from the OR, LCL, and UCL variables, and specifying it in a separate YAXISTABLE statement in order to individually control its position. The YAXISTABLE statement for the Weight variable centers the values relative to the header (but maintains the right-justification relative to each other). All the YAXISTABLE statements control the padding to provide more uniform spacing between the columns.

You can achieve a different look by specifying LOCATION=OUTSIDE in all YAXISTABLE statements to move the axis table outside the graph as follows:

```
proc sgplot data=forest noautolegend nocycleattrs;
   refline 1 / axis=x;
   refline 0.1 10 / axis=x lineattrs=(pattern=shortdash) transparency=0.5;
   highlow y=nStudy low=LCL high=UCL / lowcap=serif highcap=serif
                                       lineattrs=graphdata1;
   scatter y=nOverall x=OR / markerattrs=graphdata2
                                         (symbol=diamondfilled size=10);
   highlow y=nStudy low=lo high=hi / type=bar intervalbarwidth=.1in
                                     nooutline fillattrs=graphdata1;
   yaxistable study      / y=nAll position=left location=outside
                           nolabel valuejustify=right pad=(right=0px);
   yaxistable OR LCL UCL / y=nStudy position=right location=outside
                           labelhalign=center labelattrs=(weight=bold)
                           pad=(left=20px right=0px);
   yaxistable weight     / y=nStudy position=right location=outside
                           labelhalign=center labelattrs=(weight=bold)
                           pad=(left=10px right=10px) valuehalign=center;
   inset '       Favors Treatment' / position=bottomleft;
   inset 'Favors Placebo        '  / position=bottomright;
   xaxis  type=log min=0.01 max=100 minor display=(nolabel)
          offsetmin=0 offsetmax=0;
   yaxis  display=none offsetmin=0.1 offsetmax=0.05 values=(1 to &nobs);
run;
```

The results are displayed in Figure 3.13.

Figure 3.13 Forest Plot with an Outside Axis Table



Background: Graphs of Constant Variables

The rest of this example shows an alternative way to make the forest plot that involves constant variables and offsets. This section illustrates those concepts by using simple artificial data sets. Section "Using Constant Variables to Make a Forest Plot" on page 54 shows you how to construct the forest plot. Also see the section "Multiple Axes and Highlighted Points" on page 13 for an introduction to multiple axes.

The following steps create and display a SAS data set that will be helpful in understanding this method of constructing forest plots:

```
data x;
   retain x1 1   x2 2   x3 3   c1 'A'   c2 'B'   c3 'C';
   do y = 1 to 10;
      l1 = substr('ABCDEFGHIJ', y, 1.);
      l2 = put(y, words12.);
      l3 = y;
      output;
   end;
run;

proc print noobs;
run;
```

The data are displayed in Figure 3.14.

Figure 3.14 Artificial Data

x1	x2	x3	c1	c2	c3	y	l1	l2	l3
1	2	3	A	B	C	1	A	one	1
1	2	3	A	B	C	2	B	two	2
1	2	3	A	B	C	3	C	three	3
1	2	3	A	B	C	4	D	four	4
1	2	3	A	B	C	5	E	five	5
1	2	3	A	B	C	6	F	six	6
1	2	3	A	B	C	7	G	seven	7
1	2	3	A	B	C	8	H	eight	8
1	2	3	A	B	C	9	I	nine	9
1	2	3	A	B	C	10	J	ten	10

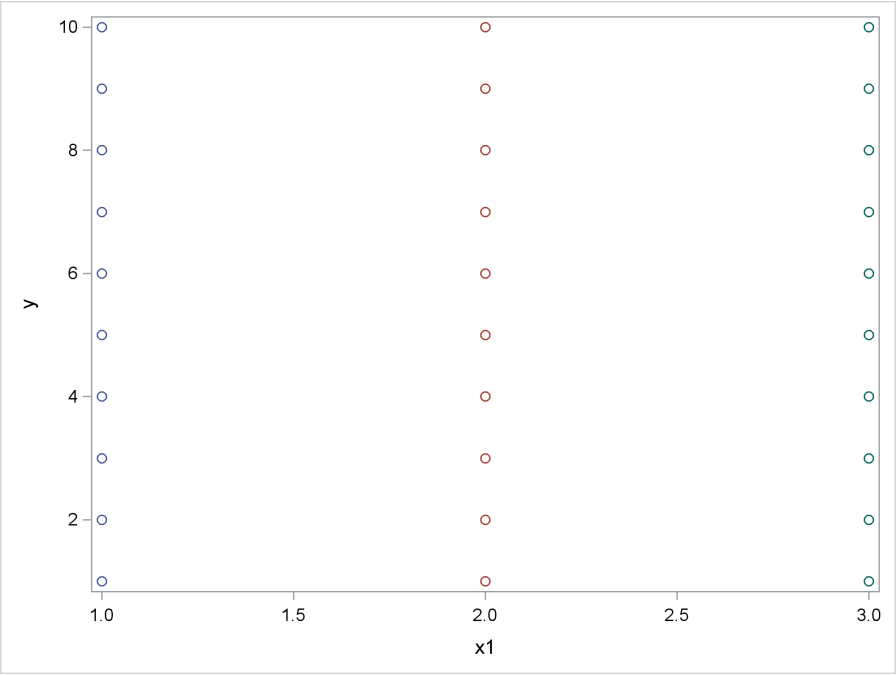
The data consist of three constant numeric variables (x1–x3), three constant character variables (c1–c3), a y variable that contains the observation number, and three label variables (l1–l3). Plotting constant variables is central to this method of constructing the forest plot. The RETAIN statement retains values across multiple passes through the DATA step rather than initializing them to missing each time. This is a one-pass DATA step (so retaining is not required), but the RETAIN statement provides a convenient syntax for creating and initializing constant variables.

The following step creates a graph that has the variable y on the Y axis and the constant numeric variables x1–x3 on the X axis:

```
proc sgplot noautolegend;
   scatter x=x1 y=y;
   scatter x=x2 y=y;
   scatter x=x3 y=y;
run;
```

The results are displayed in Figure 3.15. The X axis is linearly scaled, and all three x variables share the same X axis.

Figure 3.15 Plotting Constant Numeric Variables

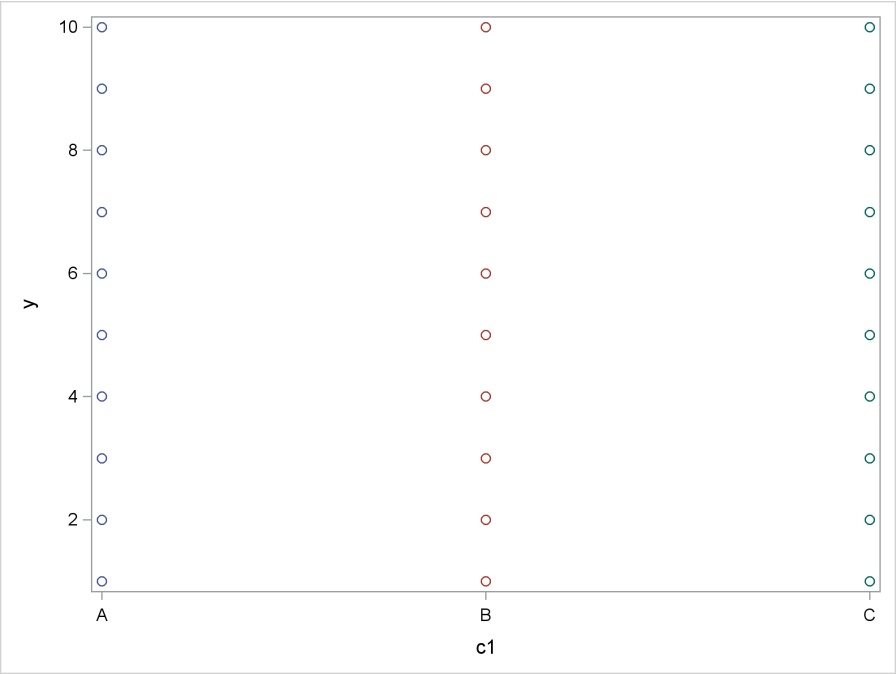


The following step creates a graph that has the variable y on the Y axis and the constant character variables c1–c3 on the X axis.

```
proc sgplot noautolegend;
   scatter x=c1 y=y;
   scatter x=c2 y=y;
   scatter x=c3 y=y;
run;
```

The results are displayed in Figure 3.16. The X axis is discrete, and values appear in the order in which they are encountered.

Figure 3.16 Plotting Constant Character Variables

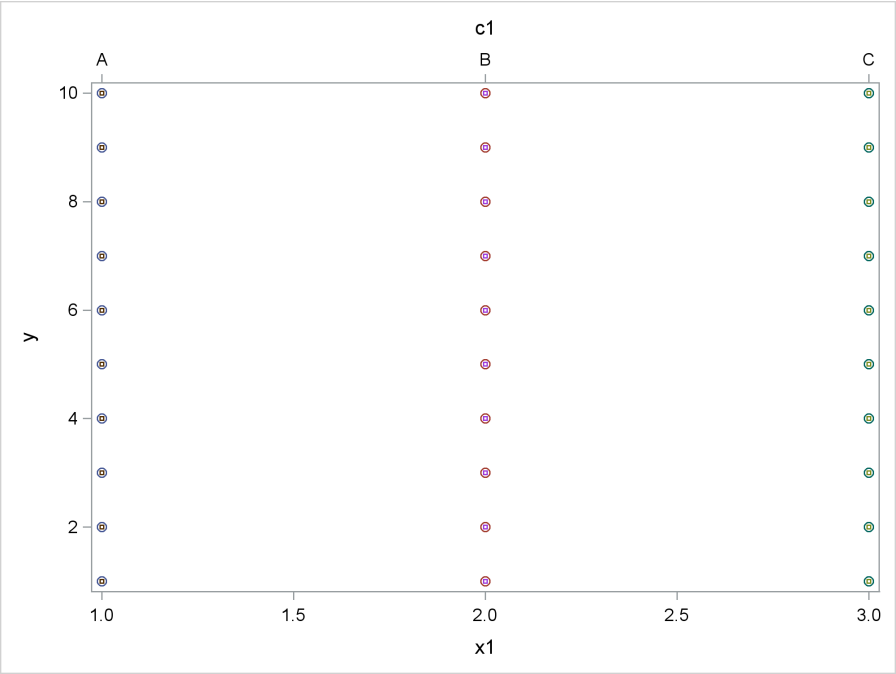


The following step overlays Figure 3.15 and Figure 3.16 and moves the character variables to the X2 axis:

```
proc sgplot noautolegend;
   scatter x=x1 y=y;
   scatter x=x2 y=y;
   scatter x=x3 y=y;
   scatter x=c1 y=y / x2axis markerattrs=(symbol=square size=3px);
   scatter x=c2 y=y / x2axis markerattrs=(symbol=square size=3px);
   scatter x=c3 y=y / x2axis markerattrs=(symbol=square size=3px);
run;
```

The results are displayed in Figure 3.17. It might not be obvious yet, but this is an important intermediate step in understanding how to construct the forest plot.

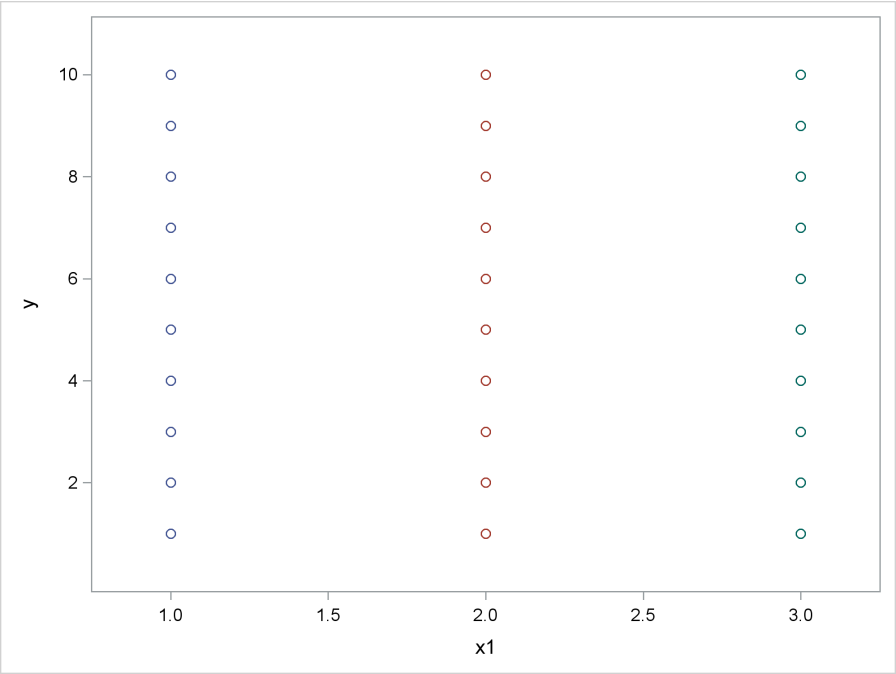
Figure 3.17 Using the X and X2 Axis



Next, you need offsets. Offsets are commonly used to move the points in from the axes. The following step creates Figure 3.18:

```
proc sgplot noautolegend;
   scatter x=x1 y=y;
   scatter x=x2 y=y;
   scatter x=x3 y=y;
   xaxis  offsetmin=0.1 offsetmax=0.1;
   yaxis  offsetmin=0.1 offsetmax=0.1;
run;
```

Figure 3.18 Standard Offsets

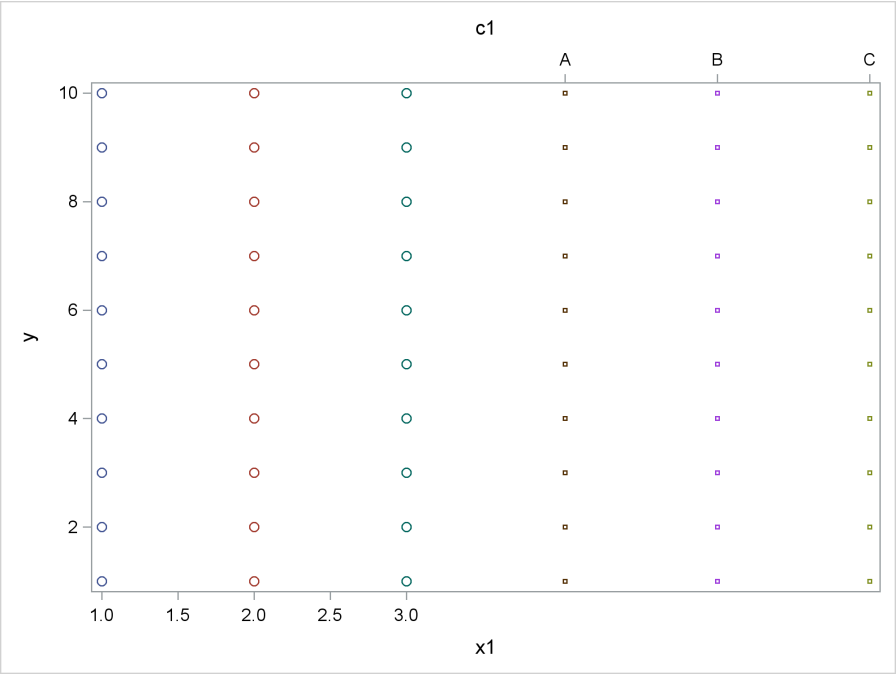


Compare Figure 3.18 and Figure 3.15 and notice how Figure 3.18 has extra space at the minimum and maximum end of both the X and Y axes. The 0.1 offsets move the points in from the axes by a distance that is 10% of the axis length.

You can create side-by-side plots by using multiple axes and much larger offsets. The following step creates Figure 3.19:

```
proc sgplot noautolegend;
   scatter x=x1 y=y;
   scatter x=x2 y=y;
   scatter x=x3 y=y;
   scatter x=c1 y=y / x2axis markerattrs=(symbol=square size=3px);
   scatter x=c2 y=y / x2axis markerattrs=(symbol=square size=3px);
   scatter x=c3 y=y / x2axis markerattrs=(symbol=square size=3px);
   xaxis   offsetmax=0.6;
   x2axis  offsetmin=0.6;
run;
```

Figure 3.19 Two Sets of Plots with Each Set Offset

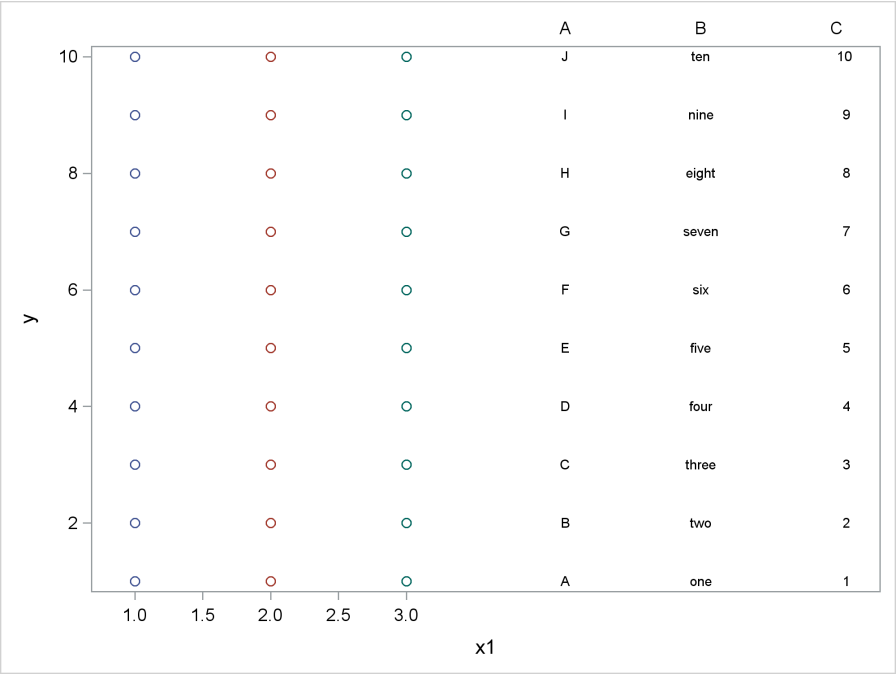


This graph is starting to resemble a forest plot. There is a graph on the left, a numeric axis on the bottom, tick labels on the top that are starting to resemble column headers, and a graph on the right that will become the table. The OFFSETMAX=0.6 option in the XAXIS statement reserves 60% of the graph space for the graph on the right. The OFFSETMIN=0.6 option in the X2AXIS statement reserves 60% of the graph space for the graph on the left.

The next step, which creates Figure 3.20, is the final step before you learn how to construct a forest plot by using the constant-variable approach:

```
proc sgplot noautolegend;
   scatter x=x1 y=y;
   scatter x=x2 y=y;
   scatter x=x3 y=y;
   scatter x=c1 y=y / x2axis markerchar=l1;
   scatter x=c2 y=y / x2axis markerchar=l2;
   scatter x=c3 y=y / x2axis markerchar=l3;
   xaxis   offsetmax=0.6;
   x2axis  offsetmin=0.6 display=(noticks nolabel);
run;
```

Figure 3.20 A Graph and a Table Displayed Together



In this step, the MARKERCHAR= option is used to display the label variables in the right part of the graph. The DISPLAY=(NOTICKS NOLABEL) option displays only the constant character variable values on the X2 axis, which now look like column headers, completing the transformation from a graph to a table. The constant variables are required because all the values in each table column have a constant X2 coordinate.

Using Constant Variables to Make a Forest Plot

The next step creates the data set that is used to make the forest plot:

```
data forest;
   input Study $1-16 OddsRatio LowerCL UpperCL Weight;
   format weight percent5. oddsratio lowercl uppercl 5.3;
   retain OR 'OR' LCL 'LCL' UCL 'UCL' WT 'Weight' FmtName 'Study';
   if n(weight) then weight = weight * 0.05;
datalines;
Modano (1967)      0.590 0.096 3.634  1
Modano (1969)      0.429 0.070 2.620  2
Leighton (1972)    0.394 0.076 2.055  2
Borodan (1981)     0.464 0.201 1.074  3.5
Novak (1992)       0.490 0.088 2.737  2
Stawer (1998)      1.250 0.479 3.261  3
Adams (1999)       0.143 0.082 0.250  4
Soloway (2000)     0.718 0.237 2.179  3
Truark (2002)      0.129 0.027 0.605  2.5
Fayney (2005)      0.313 0.054 1.805  2
Modano (2006)      0.429 0.070 2.620  2
Adams (2007)       0.143 0.082 0.250  4
Soloway (2008)     0.718 0.237 2.179  3
Truark (2010)      0.129 0.027 0.605  2.5
Fayney (2012)      0.313 0.054 1.805  2
Overall            0.328 0.233 0.462  .
;
```

The RETAIN statement creates constant character variables that provide the positions and column headers for the table. The variable OR (odds ratio) has the constant value `'OR'`, LCL (lower confidence limit) has the constant value `'LCL'`, UCL (upper confidence limit) has the constant value `'UCL'`, and Wt has the constant value `'Weight'`.

The constant variable FmtName is used in a subsequent step to create a SAS format. You will use the format to display the study names on the Y axis while specifying two integer variables in the plotting statements. The following steps create and display these variables, which are shown in Figure 3.21:

```
data forest2; /* Make format input data and Y-axis variables */
   set forest nobs=nobs;
   Start    = nobs + 1 - _n_;                          * Need 1-16 for format;
   nStudy   = ifn(study eq 'Overall', .    , start); * Need 2-16 for studies;
   nOverall = ifn(study eq 'Overall', start, .    ); * Need 1    for overall;
run;

proc print noobs;
run;
```

Figure 3.21 Forest Plot Data

Study	OddsRatio	LowerCL	UpperCL	Weight	OR	LCL	UCL	WT	FmtName	Start	nStudy	nOverall
Modano (1967)	0.590	0.096	3.634	5%	OR	LCL	UCL	Weight	Study	16	16	.
Modano (1969)	0.429	0.070	2.620	10%	OR	LCL	UCL	Weight	Study	15	15	.
Leighton (1972)	0.394	0.076	2.055	10%	OR	LCL	UCL	Weight	Study	14	14	.
Borodan (1981)	0.464	0.201	1.074	18%	OR	LCL	UCL	Weight	Study	13	13	.
Novak (1992)	0.490	0.088	2.737	10%	OR	LCL	UCL	Weight	Study	12	12	.
Stawer (1998)	1.250	0.479	3.261	15%	OR	LCL	UCL	Weight	Study	11	11	.
Adams (1999)	0.143	0.082	0.250	20%	OR	LCL	UCL	Weight	Study	10	10	.
Soloway (2000)	0.718	0.237	2.179	15%	OR	LCL	UCL	Weight	Study	9	9	.
Truark (2002)	0.129	0.027	0.605	13%	OR	LCL	UCL	Weight	Study	8	8	.
Fayney (2005)	0.313	0.054	1.805	10%	OR	LCL	UCL	Weight	Study	7	7	.
Modano (2006)	0.429	0.070	2.620	10%	OR	LCL	UCL	Weight	Study	6	6	.
Adams (2007)	0.143	0.082	0.250	20%	OR	LCL	UCL	Weight	Study	5	5	.
Soloway (2008)	0.718	0.237	2.179	15%	OR	LCL	UCL	Weight	Study	4	4	.
Truark (2010)	0.129	0.027	0.605	13%	OR	LCL	UCL	Weight	Study	3	3	.
Fayney (2012)	0.313	0.054	1.805	10%	OR	LCL	UCL	Weight	Study	2	2	.
Overall	0.328	0.233	0.462	.	OR	LCL	UCL	Weight	Study	1	.	1

In the next step, PROC FORMAT creates the format from an input SAS data set. It needs a variable called Label, which contains the values that are displayed. The variable can be created by renaming the variable Study. PROC FORMAT also needs a variable called FmtName, which contains the name of the format that you want to create (Study). Finally, PROC FORMAT needs a variable called Start, which contains the input values (1–16) that are replaced by the values of the Label variable. The new format is applied to the variables nStudy and nOverall, which will both be specified on the Y axis. Two study name variables are needed because the overall results are displayed differently from the individual study results. Hence, separate plotting statements are needed for the studies and for the overall results, each of which uses separate Y-axis variables. The following step runs PROC FORMAT and creates the format STUDY, which maps the integers 1–16 to the study names:

```
proc format library=work cntlin=forest2(keep=fmtname study start
                                        rename=(study=label));
run;
```

The next step applies the format and creates a new data set that has two more variables that are needed (lo and hi):

```
data forest3;
   set forest2(drop=fmtname start) nobs=nobs;

   /* Compute the width of the box proportional to weight on log axis.     */
   if n(weight) then lo = oddsratio / (10 ** (weight/2)); /* Bar width     */
   if n(weight) then hi = oddsratio * (10 ** (weight/2)); /* shows study's */
                                                          /* weight.       */
   if _n_ eq 1 then call symputx('nobs', nobs);
   format nStudy nOverall study.;
run;
```

The widths of the bars show the weights of the studies. Widths are computed on an exponential scale and are later displayed on a log scale.

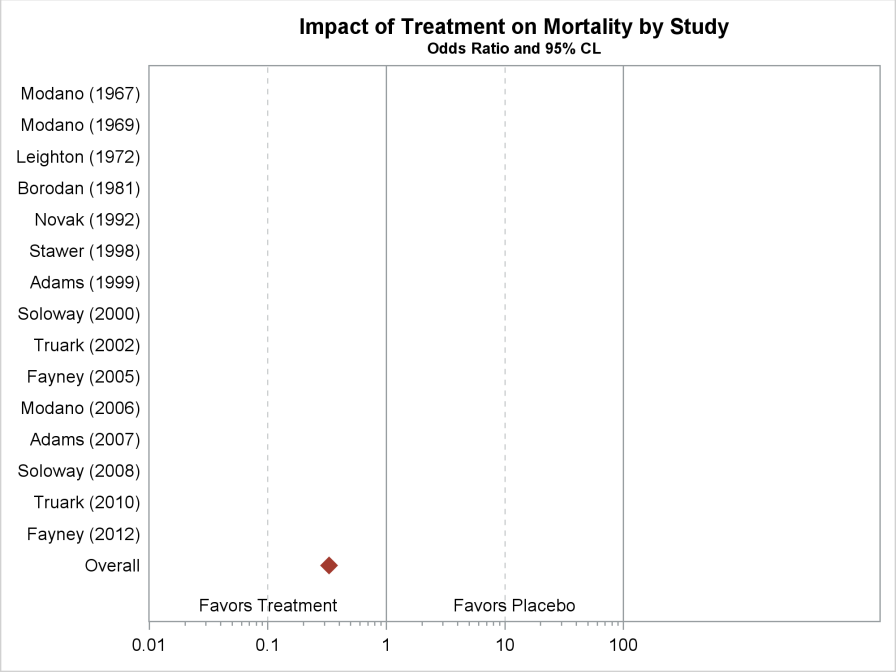
The next series of steps builds up to the forest plot by first constructing the shell of the plot and then adding to it. This step creates Figure 3.22:

```
title 'Impact of Treatment on Mortality by Study';
title2 h=8pt 'Odds Ratio and 95% CL';

proc sgplot data=forest3 noautolegend nocycleattrs;
   refline 1 100  / axis=x;
   refline 0.1 10 / axis=x lineattrs=(pattern=shortdash) transparency=0.5;
   scatter y=nOverall x=oddsratio    / markerattrs=graphdata2
                                       (symbol=diamondfilled size=10);
   inset '        Favors Treatment'  / position=bottomleft;
   inset 'Favors Placebo'            / position=bottom;
   xaxis  type=log offsetmin=0 offsetmax=0.35 min=0.01 max=100
          minor display=(nolabel);
   x2axis offsetmin=0.7 display=(noticks nolabel);
   yaxis  display=(noticks nolabel) offsetmin=0.1 offsetmax=0.05
          values=(1 to &nobs);
run;
```

The SCATTER statement produces the display of the overall results. All the other statements control titles, reference lines, inset text, and the axes. The NOCYCLEATTRS option in the PROC SGPLOT statement specifies that PROC SGPLOT should not automatically create unique attributes for the elements in the graph. All the attributes are specified in the options. The OFFSETMAX=0.35 option in the XAXIS statement reserves 35% of the space to the right of the graph for the table. The OFFSETMIN=0.7 option in the X2AXIS statement reserves 70% of the space to the left of the table for the graph. The YAXIS statement specifies the values 1 to 16 (1 to the value of the macro variable &nobs) on the Y axis. Because the Y-axis variables have formats associated with them, the study names are displayed instead of integers.

The X axis has a log scale, and the X2 axis is discrete (because it displays character variables). The X axis ranges from 0.01 to 100 and has a solid reference line at 1 and dashed reference lines at 0.1 and 10. The inset strings `'Favors Treatment'` and `'Favors Placebo'` label the two parts of the X axis. The OFFSETMIN=0.1 option in the YAXIS statement reserves space for those two strings.

Figure 3.22 Step 1: The Shell of the Forest Plot

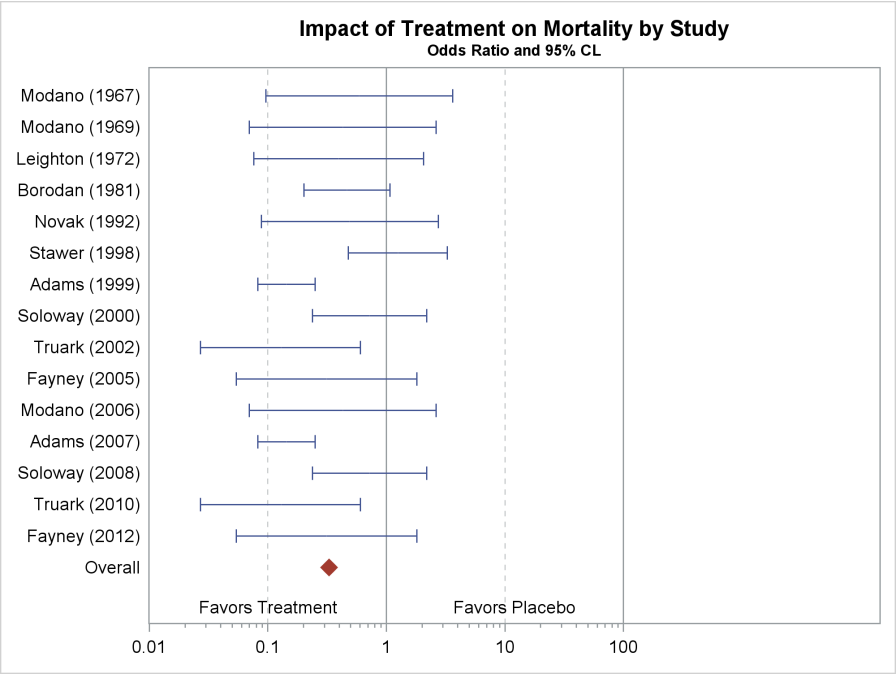


Adding the following statement after the REFLINE statements produces the results displayed in Figure 3.23.

```
scatter y=nStudy   x=oddsratio    / xerrorupper=uppercl xerrorlower=lowercl
                                     markerattrs=(size=0)
                                     errorbarattrs=graphdata1;
```

Error bars are displayed after the reference lines so that they are at the back of the graph and do not obscure any other graphical elements.

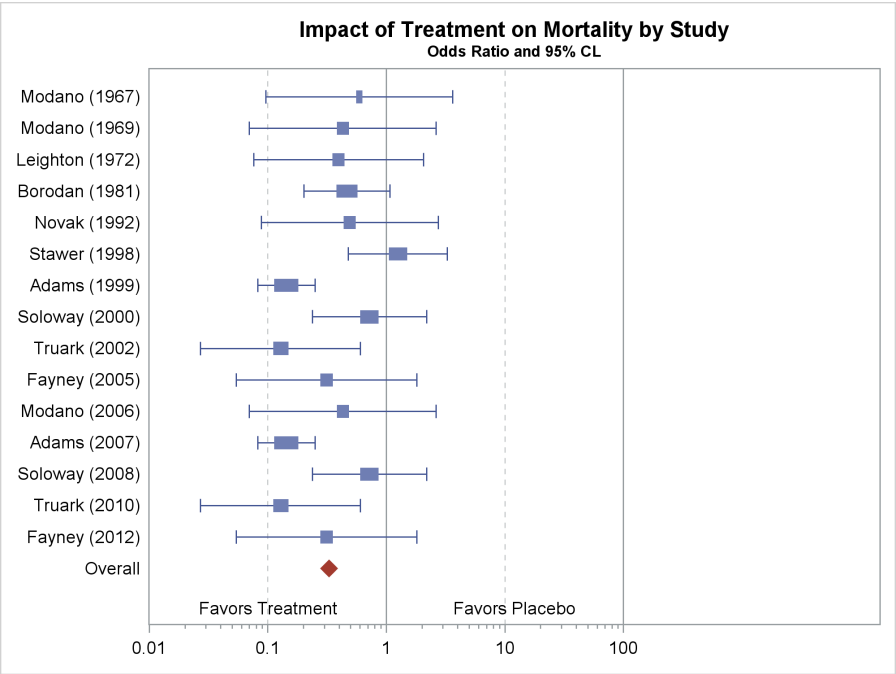
Figure 3.23 Step 2: Adding Error Bars



Adding the following statement after the SCATTER statement produces the results displayed in Figure 3.24.

```
highlow y=nStudy low=lo high=hi  / type=bar intervalbarwidth=.1in
                                   nooutline fillattrs=graphdata1;
```

Figure 3.24 Step 3: Adding Study Weights

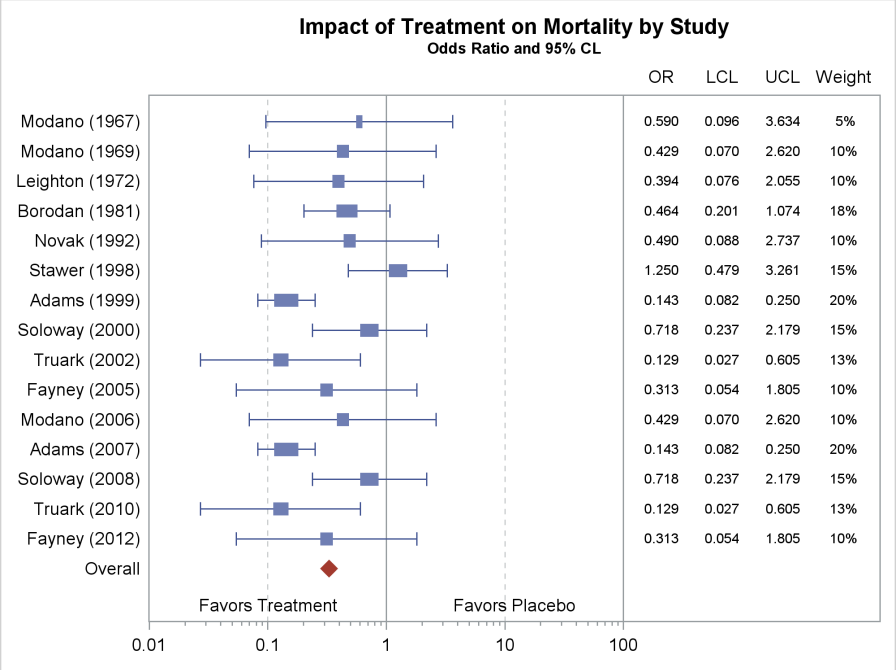


The following statements add the table:

```
scatter y=nStudy x=or  / markerchar=oddsratio x2axis;
scatter y=nStudy x=lcl / markerchar=lowercl   x2axis;
scatter y=nStudy x=ucl / markerchar=uppercl   x2axis;
scatter y=nStudy x=wt  / markerchar=weight    x2axis;
```

The results are displayed in Figure 3.25. This is the full forest plot.

Figure 3.25 Forest Plot



The complete PROC SGPLOT step that uses the constant-variable approach is as follows:

```
proc sgplot data=forest3 noautolegend nocycleattrs;
   refine 1 100  / axis=x;
   refine 0.1 10 / axis=x lineattrs=(pattern=shortdash) transparency=0.5;
   scatter y=nStudy   x=oddsratio  / xerrorupper=uppercl xerrorlower=lowercl
                                     markerattrs=(size=0)
                                     errorbarattrs=graphdata1;
   scatter y=nOverall x=oddsratio  / markerattrs=graphdata2
                                     (symbol=diamondfilled size=10);
   highlow y=nStudy low=lo high=hi / type=bar intervalbarwidth=.1in
                                     nooutline fillattrs=graphdata1;
   scatter y=nStudy x=or  / markerchar=oddsratio x2axis;
   scatter y=nStudy x=lcl / markerchar=lowercl   x2axis;
   scatter y=nStudy x=ucl / markerchar=uppercl   x2axis;
   scatter y=nStudy x=wt  / markerchar=weight    x2axis;
   inset '        Favors Treatment'  / position=bottomleft;
   inset 'Favors Placebo'            / position=bottom;
   xaxis  type=log offsetmin=0 offsetmax=0.35 min=0.01 max=100
          minor display=(nolabel);
   x2axis offsetmin=0.7 display=(noticks nolabel);
   yaxis  display=(noticks nolabel) offsetmin=0.1 offsetmax=0.05
          values=(1 to &nobs);
run;
```

Stem-and-Leaf Plot with a Box Plot

Double Click for Example Code

The UNIVARIATE procedure produces a stem-and-leaf plot. In its simplest implementation, a stem-and-leaf plot displays a column of the first digits of a list of two digit numbers, and each digit in the column is followed by the second digit of each number that shares the same first digit. Of all graphical displays of data, the stem-and-leaf plot is probably the one that benefits least from high-resolution graphics. However, stem-and-leaf plots are often displayed along with a box plot, and that type of joint display does benefit from graphics.

The following steps read some data and use PROC UNIVARIATE to create a stem-and-leaf plot:

```
data x;
   input x @@;
   datalines;
21 82 79 55 5 81 51 69 59 89 83 64 24 90 48 78 27 63 72 85 25 97 70 72 70 56 61
54 99 76 74 53 17 57 7 78 69 83 15 50 99 72 58 60 80 91 59 81 13 59 11 99 85 74
44 69 89 93 27 79 26 56 27 77 24 67 77 23 55 22 68 4 93 76 27 88 74 62 90 91 51
63 0 65 78 93 91 62 97 69 45 58 24 91 5 72 24 56 84 56 52 60 67 88 18 59 52 67
63 91 46 87 68 75 85 54 78 65 80 94 62 76 12 77 50 87 23 81 98 54 55 86 74 55
44 97 27 60 44 66 54 78 61 61 90 97 53 72 94 87 69 58 42 94 57 93 64 55 28 65
55 50 94 90 2 60 99 73 13 77 68 88 8 77 2 53 76 98 22 52 7 98 26 60 14 57 52
60 26 71 2 78 83 74
;

ods graphics off;
proc univariate plot;
   ods select plots;
run;
```

Because ODS Graphics is not enabled, PROC UNIVARIATE creates a stem-and-leaf printer plot. If ODS Graphics had been enabled, PROC UNIVARIATE would create a bar chart. The aesthetic quality of the printer plot is not sufficient to merit an appearance in a book on advanced graphics, but you can look at Chapter 4, "The UNIVARIATE Procedure" (*Base SAS Procedures Guide: Statistical Procedures*), if you want to see the printer plot.

The next steps use many of the same techniques that were introduced in the section "Creating a Forest Plot Using PROC SGPLOT" on page 42. PROC SGPLOT is used to create a table (composed in part from a constant variable) and the stem-and-leaf plot and the box plot share a common Y axis.

The following steps sort the data and create several new variables, which are described after the step:

```
proc sort data=x out=stem; by x; run;

data stem(drop=l s x);
   length stem l s $ 1 leaf $ 50;
   retain leaf ' '  n 5 t 1;
   set stem end=eof;
   s    = put(floor(x / 10), 1.);
```

```
   l    = put(mod  (x , 10), 1.);
   stem = lag(s);
   if stem ne s then do;
      if _n_ gt 1 then link maken;
      leaf = ' ';
   end;
   leaf = cats(leaf, l);
   if eof then link maken;
   return;
maken: n    = input(stem || '5', 2.);
       leaf = ' ' || leaf;
       output;
   return;
run;

data stem; merge stem x; run;
```

The CATS function converts numeric values to character, left-justifies and trims the values, and then concatenates the values.

The new variables are as follows:

Stem	is a character variable of length one that contains the stem.
Leaf	is a character variable that contains all of the leaves in order.
n	is a numeric variable that contains the midpoints of each interval (5, 15, 25, ..., 95). It is used as the Y-axis variable for the stem-and-leaf plot. The code works only for data in the range 0–99.
t	is a constant variable that provides the X coordinate for the stem-and-leaf plot.

The END= option in the SET statement creates a new variable, eof (end of file), which is 1 (true) when the last observation in the stem data set is processed and 0 otherwise. This construction is used extensively throughout the examples.

The stem-and-leaf data are merged with the original data, which contain the variable x. The numbers of nonmissing observations in the two partitions of the data set are different. This is expected and fine. The variables Stem, Leaf, n, and t contain 9 nonmissing values (not 10, because there are no observations in the 30–39 range), whereas the x variable contains 194 values.

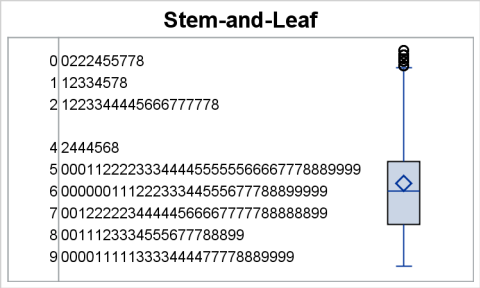
The following step creates the stem-and-leaf and box plot:

```
ods graphics on / width=250px height=150px;
proc sgplot data=stem noautolegend;
   title 'Stem-and-Leaf';
   refline 1 / axis=x2 discreteoffset=0.05;
   scatter y=n x=t / x2axis markerchar=stem;
   scatter y=n x=t / x2axis markerattrs=(size=0px) datalabel=leaf
                             datalabelpos=right;
   vbox    x / boxwidth=0.7;
   x2axis  display=none offsetmin=0   offsetmax=0.8 type=discrete;
   xaxis   display=none offsetmin=0.8 offsetmax=0.1;
   yaxis   display=none reverse;
run;
```

The first SCATTER statement displays the stem as a marker character. The second SCATTER statement displays the leaf as a data label that appears to the right of the marker. These two statements cannot be

combined. The second SCATTER statement suppresses markers by specifying their size as 0 pixels. Both SCATTER statements use the X2 and Y axes. The VBOX statement draws the vertical box plot and uses the X and Y axes. The REFLINE statement provides a line between the stem and the leaf. The stem-and-leaf plot uses the X2 axis. It is a discrete axis, and the DISCRETEOFFSET= option in the REFLINE statement moves the reference line to the right of the stem. Notice that there is only one X2 axis coordinate. The X2 axis is discrete so that there is a mechanism to position the reference line. The discrete offset of 0.05 was empirically derived for these data and this graph size. The X2 axis reserves 80% of its space to the right for the box plot. The stem-and-leaf plot extends into this area, but the 80% specification pushes the stem-and-leaf plot to the left and the box plot to the right. The X axis reserves 80% of its space to the left for the stem-and-leaf plot. Nothing is displayed on any of the axes. The Y axis is reversed so that smaller numbers are displayed at the top and values increase as you move down the axis. The two Y-axis variables, n and x, are on the same scale. This aligns the stem-and-leaf and box plots and enables the graph to skip a line for the range 30–39, which does not appear in the data. The results are displayed in Figure 3.26.

Figure 3.26 Stem-and-Leaf and Box Plot



Axis Table Example Using PROC AUTOREG

Double Click for Example Code

This example continues illustrating uses of the axis table and introduces annotation at the end, which is discussed in detail in the next chapter. You can use annotation along with PROC SGPLOT to position text in various places in the plot, draw lines and shapes, insert images, and do many other things. This example illustrates one of the most basic uses of annotation (positioning text in a graph).

The AUTOREG procedure produces a table that displays lags, autocovariances, autocorrelations, and a bar chart composed of asterisks that graphically shows the autocorrelations. You can use the output data set from the autocorrelation table, along with PROC SGPLOT to create a side-by-side display of the table and autocorrelation graph. Annotation is used to provide a header over the graph.

The following steps create and analyze an artificial data set that has second-order autocorrelation:

```
data a;
   ul = 0; ull = 0;
   do Time = -10 to 36;
      u = + 1.3 * ul - .5 * ull + 2 * normal(104);
      y = 10 + .5 * time + u;
      if time > 0 then output;
      ull = ul; ul = u;
   end;
run;
```

```
proc autoreg data=a;
   ods output corrgraph=cg;
   model y = time / method=ml nlag=5 backstep;
run;
```

The aesthetic quality of the resulting "graph" is not sufficient to merit an appearance in a book on advanced graphics, but you can look at Chapter 9, "The AUTOREG Procedure" (*SAS/ETS User's Guide*), if you want to see the bar chart composed of asterisks.

The next steps use many of the same techniques that were introduced in the section "Creating a Forest Plot Using PROC SGPLOT" on page 42. PROC SGPLOT is used to create a table (composed in part from constant variables) and each row corresponds to a bar in a high-low plot (see Figure 3.29). The following step creates the input data set:

```
data autocorr;
   retain x1 'Lag' x2 'Covariance' x3 'Autocorrelation';
   set cg;
   low  = ifn(autocorr ge 0, 0, autocorr);
   high = ifn(autocorr lt 0, 0, autocorr);
   cov  = put(autocov,  7.4);
   cor  = put(autocorr, 7.4);
run;
```

The constant variables x1–x3 provide the three headers for the tables of the column: `'Lag'`, `'Covariance'` and `'Autocorrelation'`. For positive autocorrelations, the high-low plot bar extends from 0 to the autocorrelation. For negative autocorrelations, the high-low plot bar extends from the autocorrelation to 0. PUT functions are used to format the values for the table. The following step creates the graph:

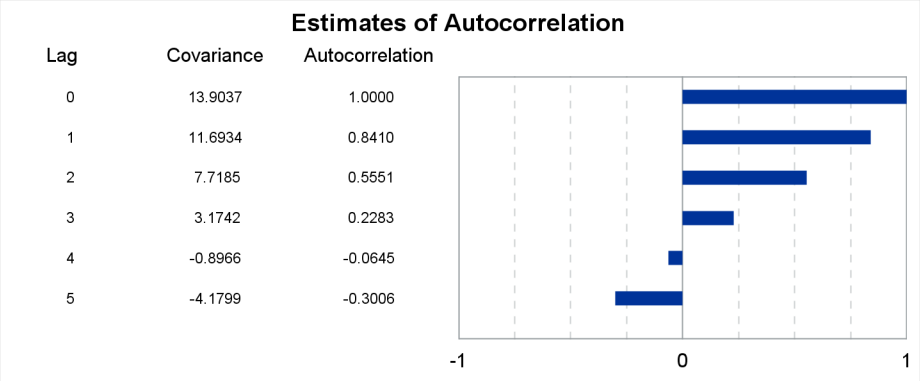
```
ods graphics on / width=4.8in height=2in;
proc sgplot data=autocorr noautolegend noborder nocycleattrs;
   title 'Estimates of Autocorrelation';
   %let o = x2axis markerattrs=(size=0) datalabelpos=center;
   scatter  x=x1 y=lag / datalabel=lag &o;
   scatter  x=x2 y=lag / datalabel=cov &o;
   scatter  x=x3 y=lag / datalabel=cor &o;
   refline  -1 0 1 / axis=x;
   refline  -0.75 -0.5 -0.25 0.25 0.5 0.75 / axis=x transparency=0.5
            lineattrs=(pattern=shortdash);
   lineparm x=-1 y=6    slope=0 / noextend lineattrs=graphreference;
   lineparm x=-1 y=-0.5 slope=0 / noextend lineattrs=graphreference;
   highlow  y=lag low=low high=high / lineattrs=(thickness=0.1in);
   x2axis   offsetmax=0.6 display=(noticks nolabel noline);
   xaxis    offsetmin=0.5  offsetmax=0 display=(nolabel noticks noline)
            values=(-1 to 1);
   yaxis    offsetmax=0 offsetmin=0.0 display=none reverse;
run;
```

The size of the graph, 2 inches by 4.8 inches, is ad hoc and makes a display that is smaller than default graphs. The NOBORDER option suppresses the border around the graph.

The three SCATTER statements create the table. Each uses lag, which contains consecutive integers, as the Y-axis variable; each uses a constant variable as the X-axis variable; and each specifies the values to be displayed in the DATALABEL= option. The data label is centered, and markers are suppressed. The SCATTER statement options that are common to all three statements are entered in the macro variable O to

minimize typing. Each SCATTER statement uses the X2 axis so that the values of the constant variables appear as headers above the columns of numbers. The REFLINE and LINEPARM statements create a box around the high-low plot. The vertical lines are at X = –1 and 1, and the horizontal lines are at Y = –0.5 and 6. The Y values are ad hoc and are slightly smaller than the smallest lag and slightly larger than the largest lag. The two vertical lines and the two horizontal lines, along with the reference line at X = 0 are displayed by using the **GraphReference** style element. Additional reference lines, which are transparent and dashed, are drawn at increments of 0.25. Offsets on the axis statements provide room to the right of the table for the graph and room to the left of the graph for the table. The Y axis is reversed so that lags become larger as you move down the table. (By default, Y axis values become smaller as you move down the Y axis.) The results are displayed in Figure 3.27.

Figure 3.27 Autocorrelations



The graph in Figure 3.27 needs one more refinement: a header that labels the bars as autocorrelations. You can provide this header by using SG annotation as follows:

```
data anno;
   retain Function 'Text' Label 'Autocorrelation' DrawSpace 'DataValue'
          x1 0 y1 -1 Width 100 TextSize 9 TextWeight 'Bold';
run;

proc print noobs;
   title;
run;

ods graphics on / width=4.8in height=2in;
proc sgplot data=autocorr noautolegend noborder nocycleattrs sganno=anno;
   title 'Estimates of Autocorrelation';
   %let o = x2axis markerattrs=(size=0) datalabelpos=center;
   scatter  x=x1 y=lag / datalabel=lag &o;
   scatter  x=x2 y=lag / datalabel=cov &o;
   scatter  x=x3 y=lag / datalabel=cor &o;
   refline  -1 0 1 / axis=x;
   refline  -0.75 -0.5 -0.25 0.25 0.5 0.75 / axis=x transparency=0.5
            lineattrs=(pattern=shortdash);
   lineparm x=-1 y=6    slope=0 / noextend lineattrs=graphreference;
   lineparm x=-1 y=-0.5 slope=0 / noextend lineattrs=graphreference;
   highlow  y=lag low=low high=high / lineattrs=(thickness=0.1in);
   x2axis   offsetmax=0.6 display=(noticks nolabel noline);
```

```
    xaxis    offsetmin=0.5  offsetmax=0 display=(nolabel noticks noline)
             values=(-1 to 1);
    yaxis    offsetmax=0 offsetmin=0.0 display=none reverse;
run;
```

The annotation data set is displayed in Figure 3.28.

Figure 3.28 Annotation Data Set

Function	Label	DrawSpace	x1	y1	Width	TextSize	TextWeight
Text	Autocorrelation	DataValue	0	-1	100	9	Bold

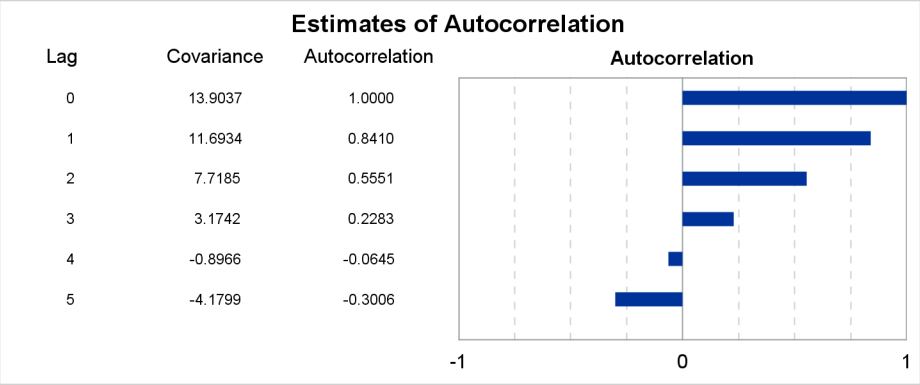
This SG annotation data set has the following variables:

Function	specifies the annotation function. In this example, the function is `'Text'`.
Label	specifies the text to insert in the graph.
DrawSpace	specifies the X- and Y-coordinate space. In this example, all coordinates are based on the data values.
x1	specifies the X coordinate (x1 = 0 positions the text above the 0 autocorrelation reference line).
y1	specifies the Y coordinate (y1 = –1 positions the text above lag=0).
Width	specifies the text width as a percentage of the X-axis space. In this example, text strings larger than 100% of the available width will wrap.
TextSize	specifies the text size (font height).
TextWeight	specifies a bold font. In this example, TextWeight=`'Bold'` sets to bold the font for the label (the "Autocorrelation" graph header).

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

The annotation data set is specified in the SGANNO= option in the PROC SGPLOT statement. The results are displayed in Figure 3.29. Now, the high-low plot is clearly labeled.

Figure 3.29 Autocorrelations (with a Header)



References

Matange, S., and Heath, D. (2011). *Statistical Graphics Procedures by Example: Effective Graphs Using SAS*. Cary, NC: SAS Institute Inc.

Chapter 4
Annotation

Contents

Replacing Tick Labels

Double Click for Example Code

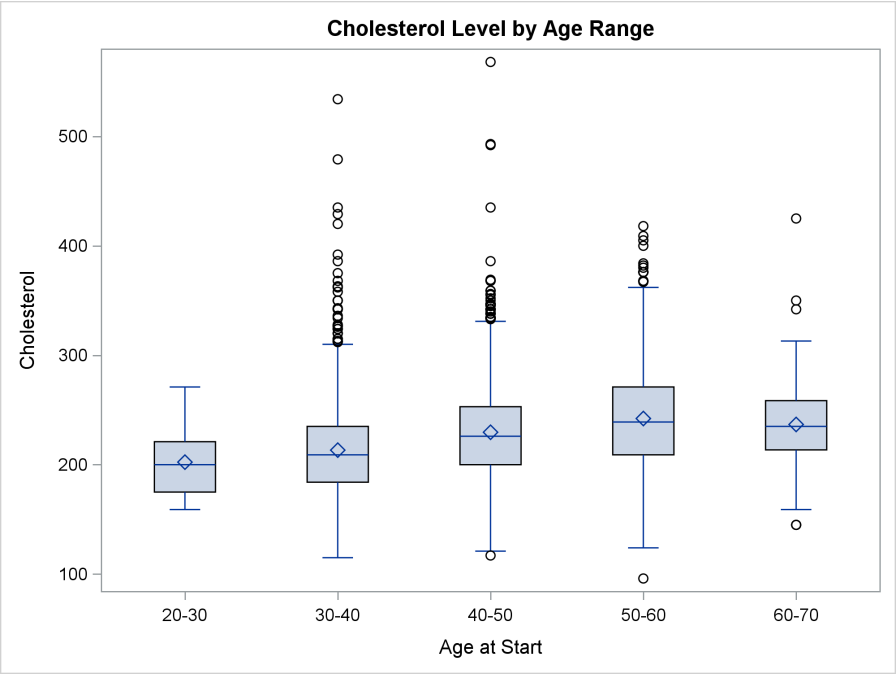
This example, which is based on an example from Matange and Heath (2011), introduces annotation. It illustrates one of the most basic uses of annotation (positioning text in a graph) and compares SG annotation to other ways of positioning text in a graph.

The following step creates an ordinary set of box plots, one for each of several age groups:

```
title 'Cholesterol Level by Age Range';

proc format;
   value agefmt 20-29.5 = '20-30' 30-39.5 = '30-40' 40-49.5 = '40-50'
                50-59.5 = '50-60' 60-69.5 = '60-70';
run;

proc sgplot data=sashelp.heart;
   vbox cholesterol / category=AgeAtStart;
   format AgeAtStart agefmt.;
run;
```

The results are displayed in Figure 4.1.

Figure 4.1 Ordinary Grouped Box Plots



This example shows how to replace X-axis tick labels by tick labels that contain special Unicode characters. For more information about Unicode, see Example 22.2 (*SAS/STAT User's Guide*) in Chapter 22, "ODS Graphics Template Modification" (*SAS/STAT User's Guide*). In the next steps, the tick labels "20–30", "30–40", and so on are replaced by "20 ≤ Age < 30", "30 ≤ Age < 40", and so on. You can use a DATA step to create the new tick labels along with instructions on where to place them as follows:

```
data anno;
   retain Function 'Text' x1 . y1 7 x1Space 'DataValue'
          y1Space 'GraphPercent' Width 21;
   do x1 = 20 to 60 by 10;
      Label = catx(' ', x1,"(*ESC*){Unicode '2264'x}", 'Age', '<', x1 + 10);
      output;
   end;
run;

proc print noobs;
run;
```

The CATX function converts numeric values to character values, left-justifies and trims the second and subsequent values, and then concatenates the values by using the first value as a separator. The annotation data set is displayed in Figure 4.2.

Figure 4.2 Annotation Data Set

Cholesterol Level by Age Range

Function	x1	y1	x1Space	y1Space	Width	Label
Text	20	7	DataValue	GraphPercent	21	20 (*ESC*){Unicode '2264'x} Age < 30
Text	30	7	DataValue	GraphPercent	21	30 (*ESC*){Unicode '2264'x} Age < 40
Text	40	7	DataValue	GraphPercent	21	40 (*ESC*){Unicode '2264'x} Age < 50
Text	50	7	DataValue	GraphPercent	21	50 (*ESC*){Unicode '2264'x} Age < 60
Text	60	7	DataValue	GraphPercent	21	60 (*ESC*){Unicode '2264'x} Age < 70

This annotation data set has the following variables:

Function	specifies the annotation function. In this example, the function is always `'Text'`.
x1	specifies the X coordinate. In this example, the X coordinates correspond to the data values at the beginning of each age range.
y1	specifies the Y coordinate. In this example, the Y coordinates are all a constant 7, which displays each string 7% of the way up from the bottom of the graph window.
x1Space	specifies the X-coordinate space. In this example, the X coordinates all correspond to X-axis data values.
y1Space	specifies the Y-coordinate space. In this example, the Y coordinates all correspond to a percentage of the entire vertical space.
Width	specifies the text width as a percentage of the X-axis space. In this example, text strings longer than 21% of the available width will wrap.
Label	specifies the text to insert in the graph.

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

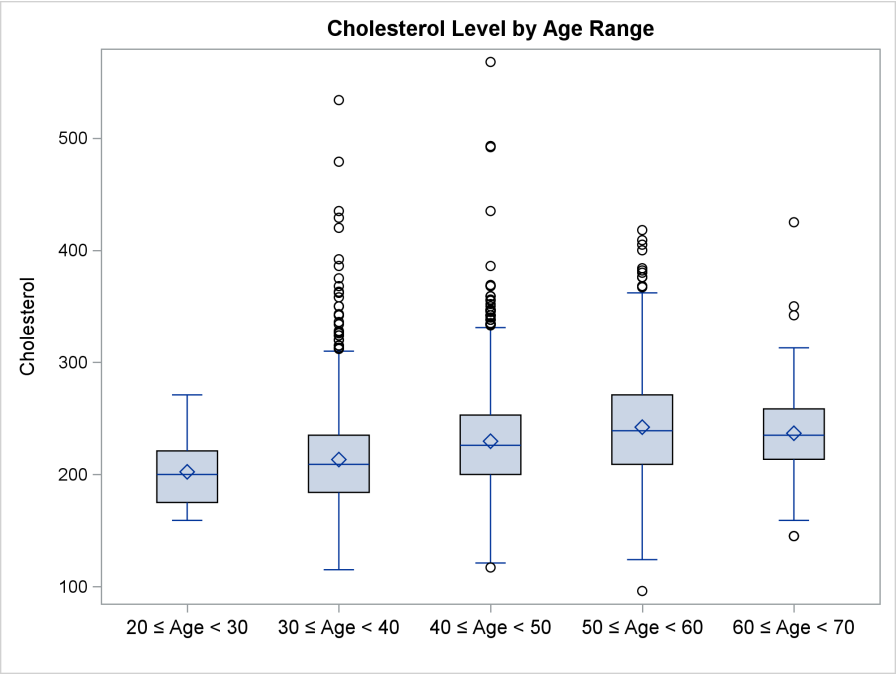
In this data set, Unicode is used to add the '≤' sign. The text **`(*ESC*)`** alerts ODS that a Unicode specification is coming and it needs to be resolved. The specification **`{Unicode '2264'x}`** creates the '≤' sign.

The following step makes the graph:

```
proc sgplot data=sashelp.heart sganno=anno pad=(bottom=8%);
   vbox cholesterol / category=AgeAtStart;
   xaxis display=(nolabel novalues);
   format AgeAtStart agefmt.;
run;
```

The DISPLAY=(NOLABEL NOVALUES) option suppresses the original axis and tick labels. By default, the space for the tick labels is lost, so the option PAD=(BOTTOM=8%) reserves 8% of the vertical space on the bottom for annotation. The results are displayed in Figure 4.3.

Figure 4.3 Grouped Box Plots That Have Customized Labels

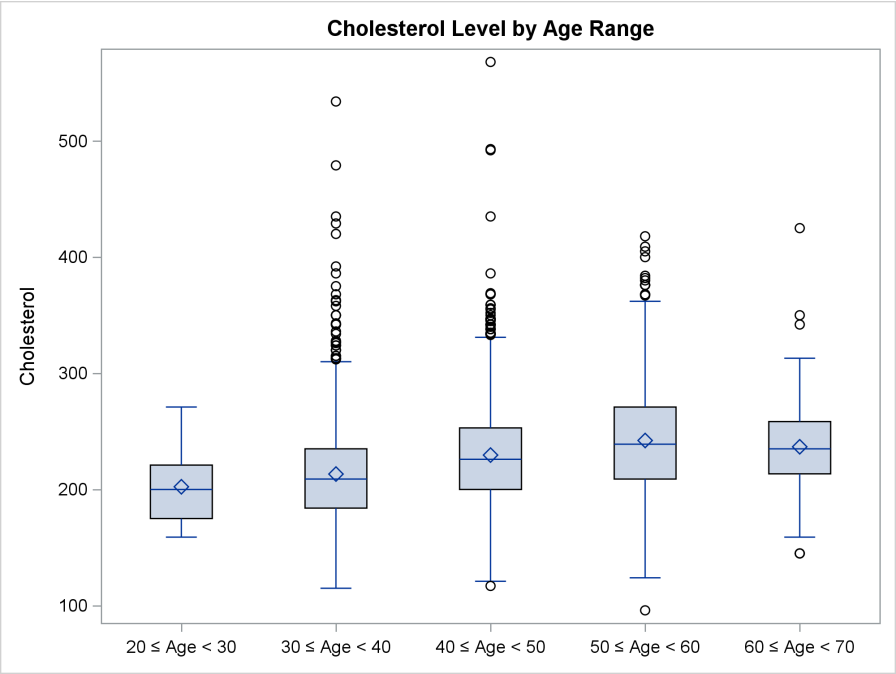


As is often the case in SAS, there is more than one way to accomplish the same goal. The following step shows how to use formats to directly add special characters to the tick labels:

```
proc format;
   value agefmt 20-29.5 = "20 (*ESC*){Unicode '2264'x} Age < 30"
                30-39.5 = "30 (*ESC*){Unicode '2264'x} Age < 40"
                40-49.5 = "40 (*ESC*){Unicode '2264'x} Age < 50"
                50-59.5 = "50 (*ESC*){Unicode '2264'x} Age < 60"
                60-69.5 = "60 (*ESC*){Unicode '2264'x} Age < 70";
run;

proc sgplot data=sashelp.heart;
   vbox cholesterol / category=AgeAtStart;
   format AgeAtStart agefmt.;
   xaxis display=(nolabel);
run;
```

The results are displayed in Figure 4.4.

Figure 4.4 Unicode and Formats



You can also manufacture a format from a SAS data set as follows:

```
data fmt;
   retain FmtName 'AgeFmt' eExcl 'Y' sExcl 'N';
   do Start = 20 to 60 by 10;
      End = Start + 10;
      Label = catx(' ', Start, "(*ESC*){Unicode '2264'x}", 'Age', "<", End);
      output;
   end;
run;

proc format cntlin=fmt;
run;

proc print noobs;
   title;
run;

proc sgplot data=sashelp.heart;
   title 'Cholesterol Level by Age Range';
   vbox cholesterol / category=AgeAtStart;
   format AgeAtStart agefmt.;
   xaxis display=(nolabel);
run;
```

The CNTLIN= data set is displayed in Figure 4.5. The graph from the PROC SGPLOT step is not displayed, but it matches the graph produced by SG annotation and displayed in Figure 4.4.

Figure 4.5 CNTLIN= Data Set

FmtName	eExcl	sExcl	Start	End	Label
AgeFmt	Y	N	20	30	20 (*ESC*){Unicode '2264'x} Age < 30
AgeFmt	Y	N	30	40	30 (*ESC*){Unicode '2264'x} Age < 40
AgeFmt	Y	N	40	50	40 (*ESC*){Unicode '2264'x} Age < 50
AgeFmt	Y	N	50	60	50 (*ESC*){Unicode '2264'x} Age < 60
AgeFmt	Y	N	60	70	60 (*ESC*){Unicode '2264'x} Age < 70

The variables in this data set are as follows:

FmtName	contains the format name.
eExcl	is set to `'Y'`, which means the End value is excluded.
sExcl	is set to `'N'`, which means the Start value is not excluded.
Start	contains the start of the range.
End	contains the end of the range.
Label	specifies the value to display in place of the values in each range.

Understanding the Drawing Spaces

Double Click for Example Code

This example explains the four drawing spaces. In the preceding example, each Y coordinate is a percentage of the entire vertical graph space and X coordinates are data values. This example more fully explains these and the other possibilities. The following step creates and displays an annotation data set that illustrates drawing spaces:

```
data anno;
   retain Function 'Arrow' x1 x2 y1 y2 .
          x1Space x2Space y1Space y2Space 'GraphPercent  ' Direction 'Both'
          Width 40 FillColor 'White';
   length Label $ 40;
   input y1 x1Space label;
   x2Space = x1Space;
   label = catt(label, ' Space');
   x1 =  0; x2 = 100; y2 = y1; function = 'Arrow'; output;
   x1 = 50; x2 =   .; y2 = .;  function = 'Text '; output;
   datalines;
25 DataPercent   Data
40 WallPercent   Wall
70 LayoutPercent Layout
85 GraphPercent  Graph
;

proc print noobs;
   title;
   footnote;
run;
```

```
data x;
   input x y @@;
   datalines;
1 1 -1 -1
;

proc sgplot data=x sganno=anno noautolegend
   pad=(top=5% bottom=5% left=5% right=5%);
   title 'Title Area';
   scatter y=y x=x / markerattrs=(size=0);
   xaxis offsetmin=0.1 offsetmax=0.1 label='X-Axis Label';
   yaxis offsetmin=0.1 offsetmax=0.1 label='Y-Axis Label';
   footnote 'Footnote Area';
run;
```

When you write a statement that initializes a character variable, such as a RETAIN or assignment statement, you need to ensure that the length is long enough to hold subsequent values. The initial value of the space variables is padded by blanks to accommodate the longer value `'LayoutPercent'`, which is used in later observations. The annotation data set is displayed in Figure 4.6, and the results are displayed in Figure 4.7.

Figure 4.6 Annotation Data Set

Function	x1	x2	y1	y2	x1Space	x2Space	y1Space	y2Space	Direction	Width	FillColor	Label
Arrow	0	100	25	25	DataPercent	DataPercent	GraphPercent	GraphPercent	Both	40	White	Data Space
Text	50	.	25	.	DataPercent	DataPercent	GraphPercent	GraphPercent	Both	40	White	Data Space
Arrow	0	100	40	40	WallPercent	WallPercent	GraphPercent	GraphPercent	Both	40	White	Wall Space
Text	50	.	40	.	WallPercent	WallPercent	GraphPercent	GraphPercent	Both	40	White	Wall Space
Arrow	0	100	70	70	LayoutPercent	LayoutPercent	GraphPercent	GraphPercent	Both	40	White	Layout Space
Text	50	.	70	.	LayoutPercent	LayoutPercent	GraphPercent	GraphPercent	Both	40	White	Layout Space
Arrow	0	100	85	85	GraphPercent	GraphPercent	GraphPercent	GraphPercent	Both	40	White	Graph Space
Text	50	.	85	.	GraphPercent	GraphPercent	GraphPercent	GraphPercent	Both	40	White	Graph Space

This annotation data set has the following variables:

Function — specifies the annotation function. In this example, the function is always `'Arrow'` or `'Text'`.

x1 — specifies the first X coordinate for arrows or the X coordinate for text.

x2 — specifies the second X coordinate for arrows. This variable is ignored when Function=`'Text'`.

y1 — specifies the first Y coordinate for arrows or the Y coordinate for text.

y2 — specifies the second Y coordinate for arrows. This variable is ignored when Function=`'Text'`.

x1Space — specifies the coordinate space for variable x1.

x2Space — specifies the coordinate space for variable x2. This variable is ignored when Function=`'Text'`.

y1Space — specifies the coordinate space for variable y1.

y2Space — specifies the coordinate space for variable y2. This variable is ignored when Function=`'Text'`.

Direction — draws arrowheads on both ends of each line. This variable is ignored when Function=`'Text'`.

Width — specifies the text width as a percentage of the X-axis space. In this example, text strings longer than 40% of the available width will wrap. This variable is ignored when Function=`'Arrow'`.

FillColor — specifies the fill color for the background behind the text. In this example, the fill color is white, which hides the line in the area behind the text. This variable is ignored when

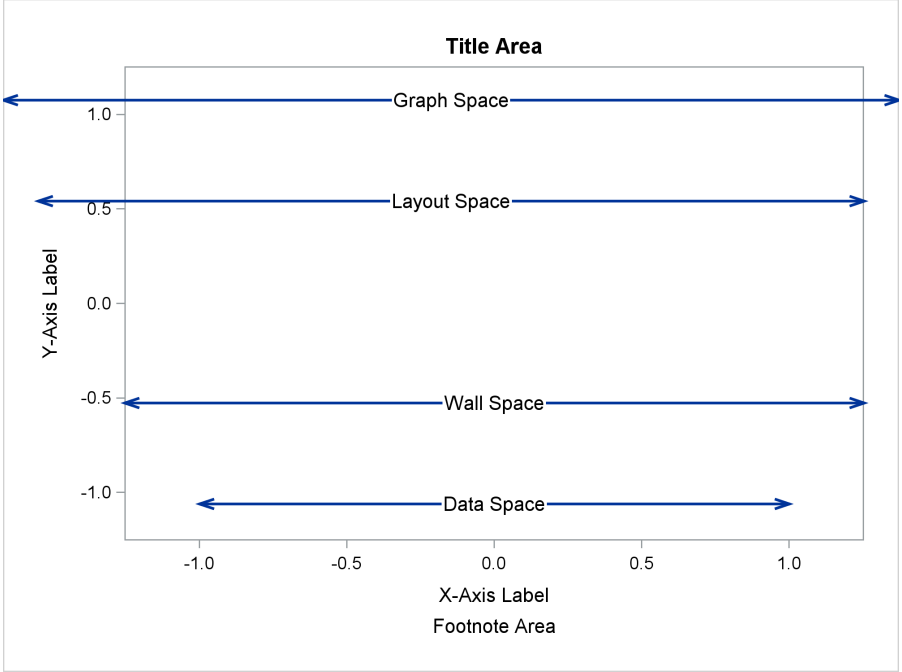
Function='Arrow'.

Label — specifies the text to insert in the graph. This variable is ignored when Function='Arrow'.

You could set the ignored values to missing (as was done for x2 and y2), but that requires more programming and is not necessary for any of the ignored variables.

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

Figure 4.7 Horizontal Drawing Spaces



The graph shows four drawing spaces: graph, layout, wall, and data. Padding and offsets are specified in the PROC SGPLOT step to show the differences more clearly. The graph space extends the full width of the graphical display. The layout space does not include the 5% padding specified in the PAD= option. The wall space extends from axis to axis. The data space extends from the smallest X coordinate to the largest. If less padding and smaller offsets were used, the data and layout spaces would extend further than they do here.

The annotation data set contains eight observations: four that draw an arrow for each of the four spaces and four that label each arrow. Arrows are drawn from (x1, y1) to (x2, y2). All coordinates are specified in percentages. The X coordinates for each arrow range from 0 to 100 in order to have the arrow point to the beginning and end of each space. The Y coordinates are arbitrary and distribute the arrows vertically.

The following steps are similar and show the vertical drawing spaces:

```
data anno;
   retain Function 'Arrow' x1 x2 y1 y2 .
          x1Space x2Space y1Space y2Space 'DataValue    ' Direction 'Both'
          Width 12 FillColor 'White';
   length Label $ 40;
   input x1 y1Space label;
   y2Space = y1Space;
```

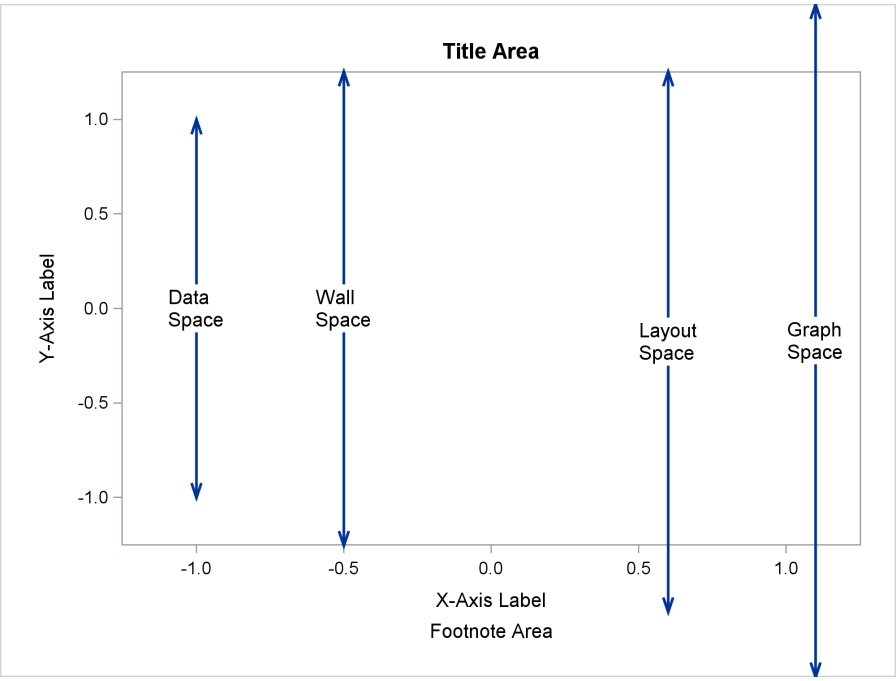
```
   label = catt(label, ' Space');
   y1 =  0; y2 = 100; x2 = x1; function = 'Arrow'; output;
   y1 = 50; y2 = .;   x2 = .;  function = 'Text '; output;
   datalines;
 -1 DataPercent   Data
-.5 WallPercent   Wall
 .6 LayoutPercent Layout
1.1 GraphPercent  Graph
;

proc print noobs;
   title;
   footnote;
run;

proc sgplot data=x sganno=anno noautolegend
            pad=(top=5% bottom=5% left=5% right=5%);
   title 'Title Area';
   scatter y=y x=x / markerattrs=(size=0);
   xaxis offsetmin=0.1 offsetmax=0.1 label='X-Axis Label';
   yaxis offsetmin=0.1 offsetmax=0.1 label='Y-Axis Label';
   footnote 'Footnote Area';
run;
```

The annotation data set is displayed in Figure 4.8, and the results are displayed in Figure 4.9.

Figure 4.8 Annotation Data Set

Function	x1	x2	y1	y2	x1Space	x2Space	y1Space	y2Space	Direction	Width	FillColor	Label
Arrow	-1.0	-1.0	0	100	DataValue	DataValue	DataPercent	DataPercent	Both	12	White	Data Space
Text	-1.0	.	50	.	DataValue	DataValue	DataPercent	DataPercent	Both	12	White	Data Space
Arrow	-0.5	-0.5	0	100	DataValue	DataValue	WallPercent	WallPercent	Both	12	White	Wall Space
Text	-0.5	.	50	.	DataValue	DataValue	WallPercent	WallPercent	Both	12	White	Wall Space
Arrow	0.6	0.6	0	100	DataValue	DataValue	LayoutPercent	LayoutPercent	Both	12	White	Layout Space
Text	0.6	.	50	.	DataValue	DataValue	LayoutPercent	LayoutPercent	Both	12	White	Layout Space
Arrow	1.1	1.1	0	100	DataValue	DataValue	GraphPercent	GraphPercent	Both	12	White	Graph Space
Text	1.1	.	50	.	DataValue	DataValue	GraphPercent	GraphPercent	Both	12	White	Graph Space

Figure 4.9 Vertical Drawing Spaces



A smaller width is used in this example to force the annotation text to wrap. Also, to illustrate other options, the X drawing spaces are set to `'DataValue'`. Notice that the `'Graph Space'` X coordinate is 1.1, which extends beyond the range of the data. SG annotation interpolates linearly, so this does not pose a problem.

When all coordinates use the same drawing space, you can specify the variable DrawSpace. Otherwise, you can control the coordinates individually by specifying the variables x1Space, x2Space, y1Space, and y2Space. You do not need to specify x2Space and y2Space when there are no X2 and Y2 coordinates.

You can specify the following values to control the drawing space:

```
'DataPercent'   'GraphPercent'   'LayoutPercent'   'WallPercent'
'DataPixel'     'GraphPixel'     'LayoutPixel'     'WallPixel'
'DataValue'
```

`'GraphPercent'` is the default.

The following steps show how to use each of the nine data spaces to draw a series of lines, each having approximately the same x1 and x2 coordinates on the X axis:

```
data anno(drop=h1 h2 hs);
   retain Function 'Line' x1 x2 y1 y2 0 Width 50 Anchor 'Left '
          hs x1Space x2Space y1Space y2Space 'GraphPercent  ';
   length Label $ 40;
   input x1Space x1 x2;
   function = 'Line';
   y1Space  = 'GraphPercent';
   y2Space  = y1Space;
   x2Space  = x1Space;
   y1       = 25 + 6 * (10 - _n_); y2 = y1;                output;
   h1       = x1;
```

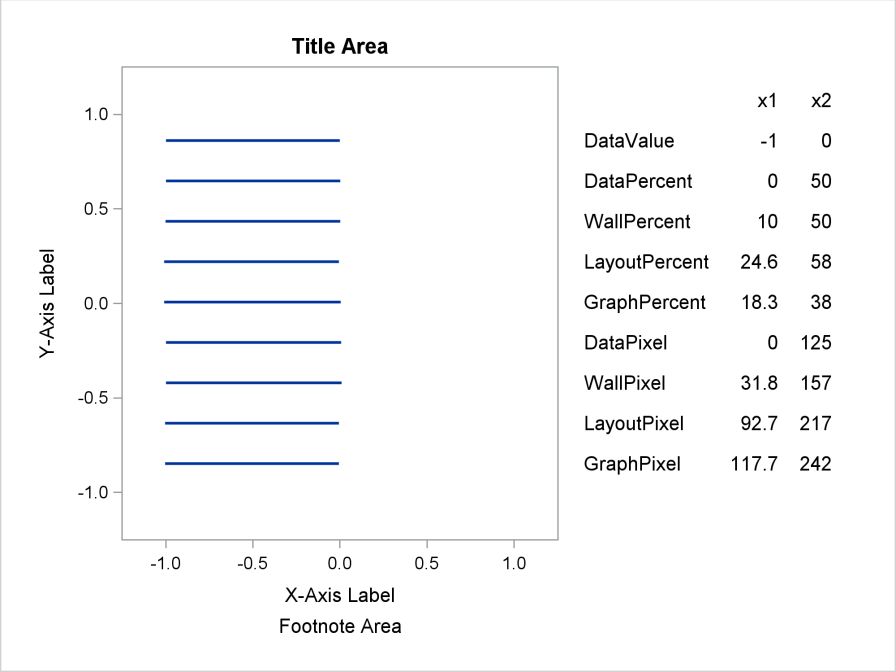
```
   h2       = x2;
   hs       = x1space;
   function = 'Text';
   x1Space  = 'GraphPercent';
   x2Space  = x1Space;
   label    = hs;    ;         anchor = 'Left '; x1 = 65; output;
   label    = put(h1, best6.); anchor = 'Right'; x1 = 87; output;
   label    = put(h2, best6.);                   ; x1 = 93; output;
   if _n_ = 1 then do;
      label = 'x2'; y1 = 85;                              output;
      label = 'x1'; x1 = 87;                              output;
   end;
   datalines;
DataValue       -1     0
DataPercent      0    50
WallPercent     10    50
LayoutPercent   24.6  58
GraphPercent    18.3  38
DataPixel        0   125
WallPixel       31.8 157
LayoutPixel     92.7 217
GraphPixel     117.7 242
;

proc print noobs;
   title;
   footnote;
run;

proc sgplot data=x sganno=anno noautolegend
            pad=(top=5% bottom=5% left=5% right=50%);
   title 'Title Area';
   scatter y=y x=x / markerattrs=(size=0);
   xaxis offsetmin=0.1 offsetmax=0.1 label='X-Axis Label' ;
   yaxis offsetmin=0.1 offsetmax=0.1 label='Y-Axis Label';
   footnote 'Footnote Area';
run;
```

The annotation data set is displayed in Figure 4.10, and the results are displayed in Figure 4.11.

Figure 4.10 Annotation Data Set

Function	x1	x2	y1	y2	Width	Anchor	x1Space	x2Space	y1Space	y2Space	Label
Line	-1.0	0	79	79	50	Left	DataValue	DataValue	GraphPercent	GraphPercent	
Text	65.0	0	79	79	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	DataValue
Text	87.0	0	79	79	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	-1
Text	93.0	0	79	79	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	0
Text	93.0	0	85	79	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	x2
Text	87.0	0	85	79	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	x1
Line	0.0	50	73	73	50	Right	DataPercent	DataPercent	GraphPercent	GraphPercent	
Text	65.0	50	73	73	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	DataPercent
Text	87.0	50	73	73	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	0
Text	93.0	50	73	73	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	50
Line	10.0	50	67	67	50	Right	WallPercent	WallPercent	GraphPercent	GraphPercent	
Text	65.0	50	67	67	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	WallPercent
Text	87.0	50	67	67	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	10
Text	93.0	50	67	67	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	50
Line	24.6	58	61	61	50	Right	LayoutPercent	LayoutPercent	GraphPercent	GraphPercent	
Text	65.0	58	61	61	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	LayoutPercent
Text	87.0	58	61	61	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	24.6
Text	93.0	58	61	61	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	58
Line	18.3	38	55	55	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	
Text	65.0	38	55	55	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	GraphPercent
Text	87.0	38	55	55	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	18.3
Text	93.0	38	55	55	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	38
Line	0.0	125	49	49	50	Right	DataPixel	DataPixel	GraphPercent	GraphPercent	
Text	65.0	125	49	49	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	DataPixel
Text	87.0	125	49	49	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	0
Text	93.0	125	49	49	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	125
Line	31.8	157	43	43	50	Right	WallPixel	WallPixel	GraphPercent	GraphPercent	
Text	65.0	157	43	43	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	WallPixel
Text	87.0	157	43	43	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	31.8
Text	93.0	157	43	43	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	157
Line	92.7	217	37	37	50	Right	LayoutPixel	LayoutPixel	GraphPercent	GraphPercent	
Text	65.0	217	37	37	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	LayoutPixel
Text	87.0	217	37	37	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	92.7
Text	93.0	217	37	37	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	217
Line	117.7	242	31	31	50	Right	GraphPixel	GraphPixel	GraphPercent	GraphPercent	
Text	65.0	242	31	31	50	Left	GraphPercent	GraphPercent	GraphPercent	GraphPercent	GraphPixel
Text	87.0	242	31	31	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	117.7
Text	93.0	242	31	31	50	Right	GraphPercent	GraphPercent	GraphPercent	GraphPercent	242

Figure 4.11 Drawing Spaces



The right portion of Figure 4.11 contains a table, constructed by annotation, that shows the X coordinates for each draw space needed to draw a line from X = –1 to X = 0. Obviously, some drawing spaces are much more convenient than others, depending on what you are doing. The annotation data set contains one variable that was not used in other parts of this example: the variable Anchor, which anchors the text strings. The value of `'Left'` left-justifies character strings such as `'GraphPercent'`. The value of `'Right'` right-justifies numbers such as 50. The variable Anchor is ignored for lines. This example shows you how to combine a table and a graph. The sections "Axis Table Example Using PROC REG" on page 34, "Creating a Forest Plot Using PROC SGPLOT" on page 42, "Stem-and-Leaf Plot with a Box Plot" on page 60, and "Axis Table Example Using PROC AUTOREG" on page 62 show other ways to combine them.

Displaying Text in a Graph

Double Click for Example Code

This example shows how to draw a normal density function, draw drop lines from the X axis to the density function, and use SG annotation to display the area under different parts of the curve. The following step creates a data set that contains some coordinates of the normal density function. The Y-axis variable is y, the X-axis variable is z, and the variable dl (for drop line) is nonmissing for integer values of z.

```
data x(drop=c);
   c = sqrt(2 * constant('pi'));
   do z = -5 to 5 by 0.05;
      y  = exp(-0.5 * z ** 2) / c;
      dl = ifn(abs(z - round(z)) < 1e-8, z, .);
      output;
   end;
run;
```

The following steps create and display the annotation data set in Figure 4.12:

```
data anno;
   retain Function 'Text' DrawSpace 'DataValue' Label '          '
          Width 20 Anchor 'Center' TextSize . TextWeight '    ';
   do x1 = -4.5 to 4.5;
      y1 = 0.05; Label = put(probnorm(x1 + 0.5), 6.5);  output;
      y1 = 0.07; Label = put(probnorm(x1 + 0.5) -
                             probnorm(x1 - 0.5), 6.5);  output;
   end;
   Anchor = 'Right'; x1 = -5.02; TextSize = 7; TextWeight = 'Bold';
   y1 = 0.05; Label = 'Cumulative';  output;
   y1 = 0.07; Label = 'Area';        output;
run;

proc print noobs;
run;
```

The PROBNORM function computes the area from minus infinity to the specified X value. The PUT function formats this area into a label. The areas are computed in a DO loop. The labels for the two types of areas are created outside the DO loop.

Figure 4.12 Annotation Data Set

Function	DrawSpace	Label	Width	Anchor	TextSize	TextWeight	x1	y1
Text	DataValue	.00003	20	Center	.		-4.50	0.05
Text	DataValue	.00003	20	Center	.		-4.50	0.07
Text	DataValue	.00135	20	Center	.		-3.50	0.05
Text	DataValue	.00132	20	Center	.		-3.50	0.07
Text	DataValue	.02275	20	Center	.		-2.50	0.05
Text	DataValue	.02140	20	Center	.		-2.50	0.07
Text	DataValue	.15866	20	Center	.		-1.50	0.05
Text	DataValue	.13591	20	Center	.		-1.50	0.07
Text	DataValue	.50000	20	Center	.		-0.50	0.05
Text	DataValue	.34134	20	Center	.		-0.50	0.07
Text	DataValue	.84134	20	Center	.		0.50	0.05
Text	DataValue	.34134	20	Center	.		0.50	0.07
Text	DataValue	.97725	20	Center	.		1.50	0.05
Text	DataValue	.13591	20	Center	.		1.50	0.07
Text	DataValue	.99865	20	Center	.		2.50	0.05
Text	DataValue	.02140	20	Center	.		2.50	0.07
Text	DataValue	.99997	20	Center	.		3.50	0.05
Text	DataValue	.00132	20	Center	.		3.50	0.07
Text	DataValue	1.0000	20	Center	.		4.50	0.05
Text	DataValue	.00003	20	Center	.		4.50	0.07
Text	DataValue	Cumulative	20	Right	7	Bold	-5.02	0.05
Text	DataValue	Area	20	Right	7	Bold	-5.02	0.07

This annotation data set has the following variables:

Function	specifies the annotation function. In this example, the function is always `'Text'`.
DrawSpace	specifies the X- and Y-axis coordinate spaces. In this example, all coordinates correspond to data values.
Label	specifies the text to insert in the graph.
Width	specifies the text width as a percentage of the X-axis space. In this example, text strings longer than 20% of the available width will wrap.
Anchor	specifies the anchor point for text. In this example, labels that contain the areas under the curve are centered and labels for the two rows (area and cumulative area) are right-justified.
TextSize	specifies the text sizes. In this example, the areas use the default size, and the labels are set to 7 pixels.
TextWeight	specifies the text weights. In this example, the areas use the default weight (normal fonts), and the labels are set to a bold font.
x1	specifies the X coordinates. In this example, the X coordinates for the labels are outside the range of the data, which places the labels outside the graph.
y1	specifies the Y coordinates.

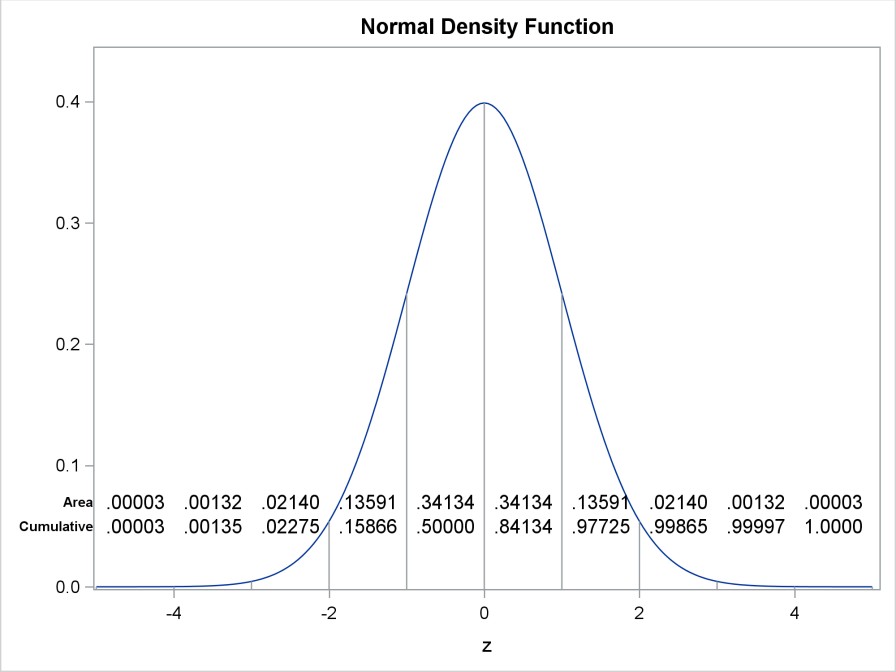
For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

The following step creates the graph that is displayed in Figure 4.13:

```
proc sgplot data=x sganno=anno pad=(left=8%);
   title 'Normal Density Function';
   series   y=y x=z;
   dropline y=y x=dl;
   yaxis offsetmax=0.1 display=(nolabel);
run;
```

The density function is drawn by a SERIES statement. The drop lines are specified by using a DROPLINE statement. Most values of the variable dl are missing, so drop lines are not drawn for most of the nonmissing values of y. Drop lines are drawn only when both y and dl are nonmissing. This is the way missing values are usually treated in ODS Graphics, and you can use this to your advantage when you are drawing graphs. The PAD= option is used to provide sufficient room for the `'Cumulative'` label.

Figure 4.13 Drop Lines and Annotation



Drawing Lines

Double Click for Example Code

This example, which is based on an example from Matange and Heath (2011), uses PROC SGPLOT to produce a bar chart and the SG annotation facility to add lines and text. The results are displayed in Figure 4.15. The following step reads the data:

```
data autos;
   input AutoMaker $ 1-16 Units;
   label units='Units (in Millions)';
   ColorVar = (index(Automaker,'Chrysler') or
               index(Automaker,'Fiat'));
   datalines;
Mitsubishi        1.1
Mazda             1.4
BMW               1.4
Daimler           1.9
Chrysler          2.0
Suzuki            2.4
Fiat              2.5
PSA               3.2
Honda             3.8
Hyundai           4.2
Fiat + Chrysler   4.5
Ford              5.4
Renault-Nissan    5.8
```

```
Volkswagen          6.0
GM                  7.7
Toyota              8.7
;
```

The binary variable ColorVar is 1 when the automaker is Chrysler or Fiat and 0 otherwise. This variable is specified as a group variable in PROC SGPLOT to produce two different bar colors, one for Chrysler and Fiat (set by the **GraphData2** style element) and another for the other automakers (set by the **GraphData1** style element).

The following step creates the annotation data set that is displayed in Figure 4.14:

```
data anno;
   length yC1 $ 15;
   retain DrawSpace 'DataValue';
   Function='PolyLine'; yc1='Chrysler';          x1=2.5;     output;
   function='PolyCont';                          x1=3.5;     output;
                        yc1='Fiat';                          output;
                                                 x1=3.0;     output;

   function='PolyLine'; yc1='Suzuki';            x1=3.5;     output;
   function='PolyCont';                          x1=6.0;     output;
                        yc1='Fiat + Chrysler';               output;
   function='Arrow';    yC2=yc1;                 x2=5.0;     output;

   function='Text';     yc1='Honda';             x1=6.1;
   Anchor='Left';       Width=30;
   Label='Alliance creates the #6 global automaker by volume.';
                        x2 = .;                  yc2 = ' ';  output;
run;

proc print noobs;
run;
```

This DATA step is a one-pass DATA step. Multiple observations are created because there are multiple OUTPUT statements, but there is only one pass through the code. Because variables are never reset to missing, the DATA step specifies a value only when it changes and not each time it is output. Also, not all variables are relevant to all annotation functions, so many values are missing.

Figure 4.14 Annotation Data Set

yC1	DrawSpace	Function	x1	yC2	x2	Anchor	Width	Label
Chrysler	DataValue	PolyLine	2.5		.		.	
Chrysler	DataValue	PolyCont	3.5		.		.	
Fiat	DataValue	PolyCont	3.5		.		.	
Fiat	DataValue	PolyCont	3.0		.		.	
Suzuki	DataValue	PolyLine	3.5		.		.	
Suzuki	DataValue	PolyCont	6.0		.		.	
Fiat + Chrysler	DataValue	PolyCont	6.0		.		.	
Fiat + Chrysler	DataValue	Arrow	6.0	Fiat + Chrysler	5		.	
Honda	DataValue	Text	6.1		.	Left	30	Alliance creates the #6 global automaker by volume.

The annotation data set contains several variables:

yC1	is a character variable that provides Y-axis coordinates.
DrawSpace	has the value `'DataValue'` for every observation, so all coordinates are data values.
Function	contains instructions for the type of annotation.
x1	is a numeric variable that provides X-axis coordinates.
yC2 and x2	provide additional X- and Y-axis coordinates for the `'Fiat + Chrysler'` annotation, which draws an arrow from the point (yC1, x1) to the point (yC2, x2).
Anchor	specifies the anchor point for text.
Width	wraps the label if it would otherwise occupy more than the specified percentage of the horizontal width.
Label	provides the annotation text.

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

The first four lines of the annotation data set draw the line segments that connect Fiat and Chrysler:

- The first line of the annotation data set (Function=`'PolyLine'`) starts a line at the coordinates yC1=`'Chrysler'` and x1=2.5. This starts the annotation line at the bottom of Figure 4.15.
- The second line of the annotation data set (Function=`'PolyCont'`) continues a line to the coordinates yC1=`'Chrysler'` and x1=3.5. This completes the annotation line at the bottom of Figure 4.15.
- The third line (Function=`'PolyCont'`) continues drawing a line from the previous coordinates to the coordinates yC1=`'Fiat'` and x1=3.5. This completes the left vertical annotation line in Figure 4.15.
- The fourth line (Function=`'PolyCont'`) continues drawing a line from the previous coordinates to the coordinates yC1=`'Fiat'` and x1=3.0.

This completes the open polygon linking Chrysler and Fiat.

The next four lines draw the segments of the arrow that connects the Fiat and Chrysler open polygon to the value `'Fiat + Chrysler'`:

- The fifth line (Function=`'PolyLine'`) starts a line at the coordinates yC1=`'Suzuki'` (between Fiat and Chrysler) and x1=3.5. This starts the first segment of the arrow.
- The sixth line (Function=`'PolyCont'`) continues a line to the coordinates yC1=`'Suzuki'` and x1=6.0. This completes the first line segment of the arrow.
- The seventh line (Function=`'PolyCont'`) continues drawing a line from the previous coordinates to the coordinates yC1=`'Fiat + Chrysler'` and x1=6.0. This completes the right vertical segment.
- The eighth line (Function=`'Arrow'`) draws the arrow segment from the coordinates yC1=`'Fiat + Chrysler'` and x1=6.0 to yC2=`'Fiat + Chrysler'` and x1=5. It ends with an arrowhead.

This completes the arrow segment.

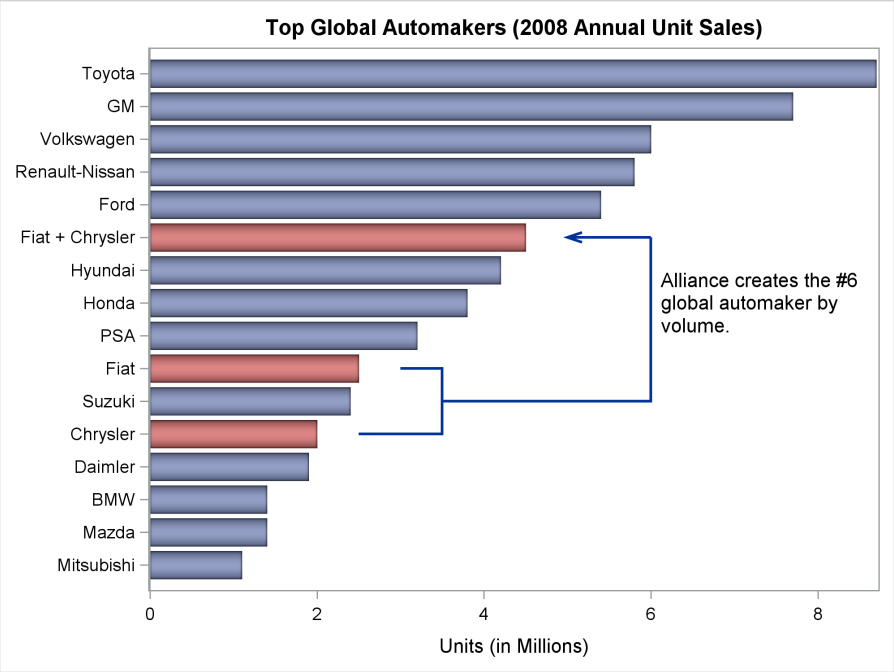
The last line positions the text at the coordinates yC1=`'Honda'` (which is in the range of automakers defined by the vertical arrow segment) and x1=6.1 (to the right of the vertical arrow segment). The variable Anchor=`'Left'` specifies that the text annotation position start at the left part of the label at that point and continue to the right.

The following step creates the graph in Figure 4.15:

```
title 'Top Global Automakers (2008 Annual Unit Sales)';
proc sgplot data=autos noautolegend sganno=anno;
   hbarparm category=Automaker response=Units / datalabel
            group=colorvar dataskin=pressed;
   yaxis display=(nolabel) reverse;
run;
```

The DATASKIN=PRESSED option specifies the appearance of the bars. Data skins add three-dimensional effects to two-dimensional bars.

Figure 4.15 Adding Lines and Text to a Graph



Custom Markers, No Markers, and the Data Region

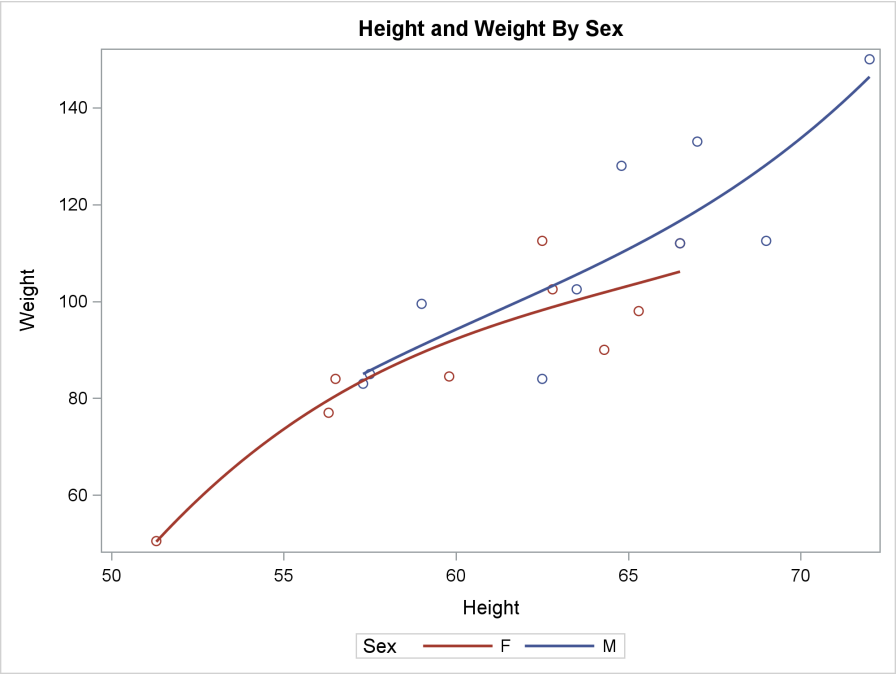
Double Click for Example Code

This example illustrates how to display Unicode characters as markers in a graph. It also illustrates how the data region is determined and explains the effect of excluding markers versus making them invisible.

The following step creates a fit plot that has two groups, and it creates Figure 4.16:

```
proc sgplot data=sashelp.class;
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3;
run;
```

Figure 4.16 Fit Plot with Two Groups

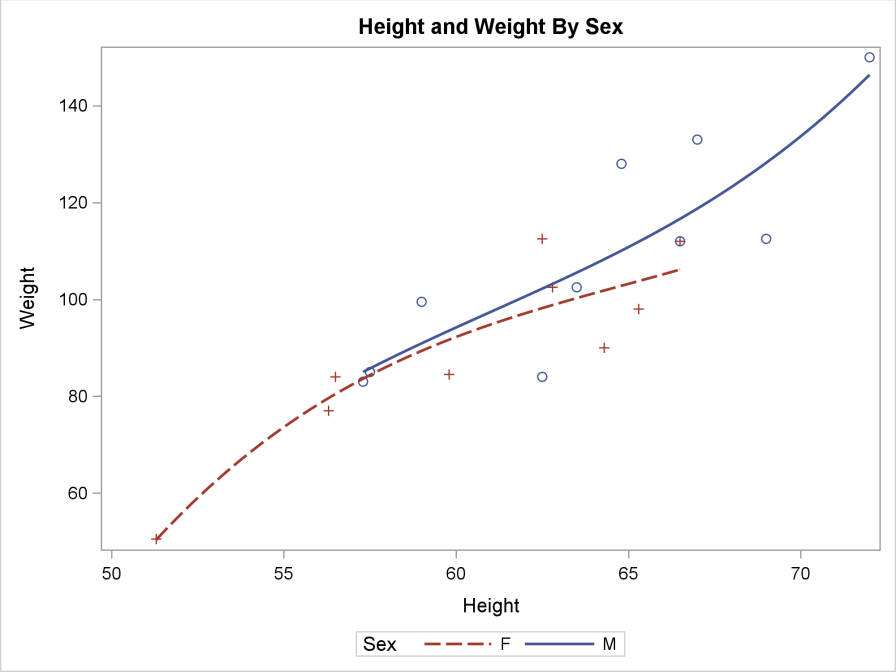


The graph is created by using the HMTLBlue style, which is an ATTRPRIORITY=COLOR style, so groups are distinguished by color. You can specify the ATTRPRIORITY=NONE option to distinguish groups by colors, markers, and lines:

```
ods graphics on / attrpriority=none;
proc sgplot data=sashelp.class;
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3;
run;
```

The results are displayed in Figure 4.17.

Figure 4.17 Groups Distinguished by Colors, Markers, and Lines

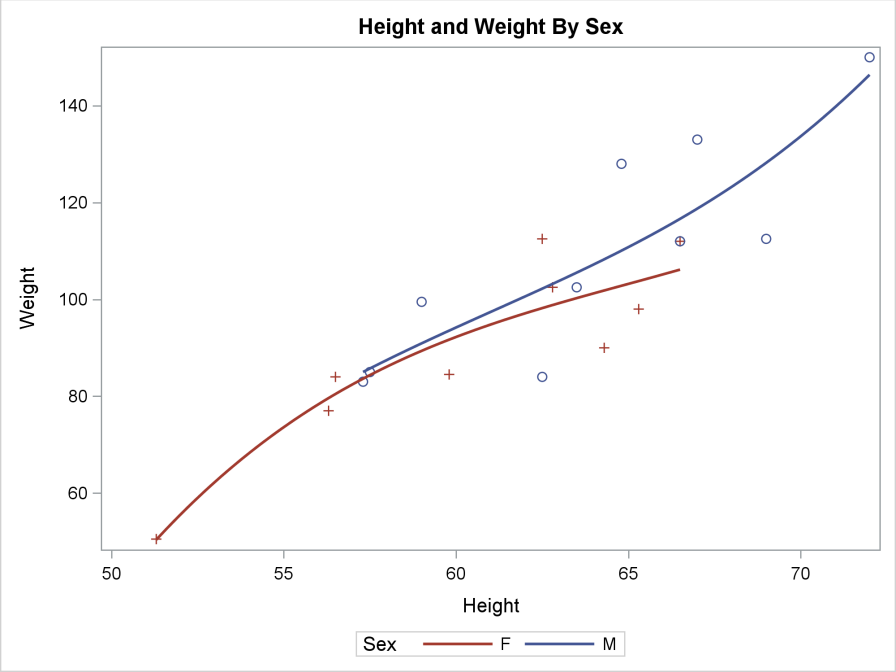


You can specify the DATALINEPATTERNS=(SOLID) option in the STYLEATTRS statement along with the ATTRPRIORITY=NONE option to distinguish groups by colors and markers:

```
ods graphics on / attrpriority=none;
proc sgplot data=sashelp.class;
   styleattrs datalinepatterns=(solid);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3;
run;

ods graphics on / reset=attrpriority;
```

The results are displayed in Figure 4.18.

Figure 4.18 Groups Distinguished by Colors and Lines



The ATTRPRIORITY= option and STYLEATTRS statement are discussed in more detail in Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*), and throughout this book.

You can specify markers that are not in the standard marker list. The following steps create an annotation data set that contains a Label variable that contains the Unicode characters for female and male:

```
data anno(drop=name sex age rename=(weight=y1 height=x1));
   set sashelp.class;
   retain Function 'Text' DrawSpace 'DataValue';
   Label = '(*ESC*){Unicode "' || ifc(sex eq 'F', '2640', '2642') || '"x}';
   TextColor = ifc(sex eq 'F', 'red', 'blue');
run;

proc print noobs;
run;

proc sgplot data=sashelp.class sganno=anno noautolegend;
   styleattrs datacontrastcolors=(blue pink);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3 markerattrs=(size=0);
run;
```

The annotation data set is displayed in Figure 4.19.

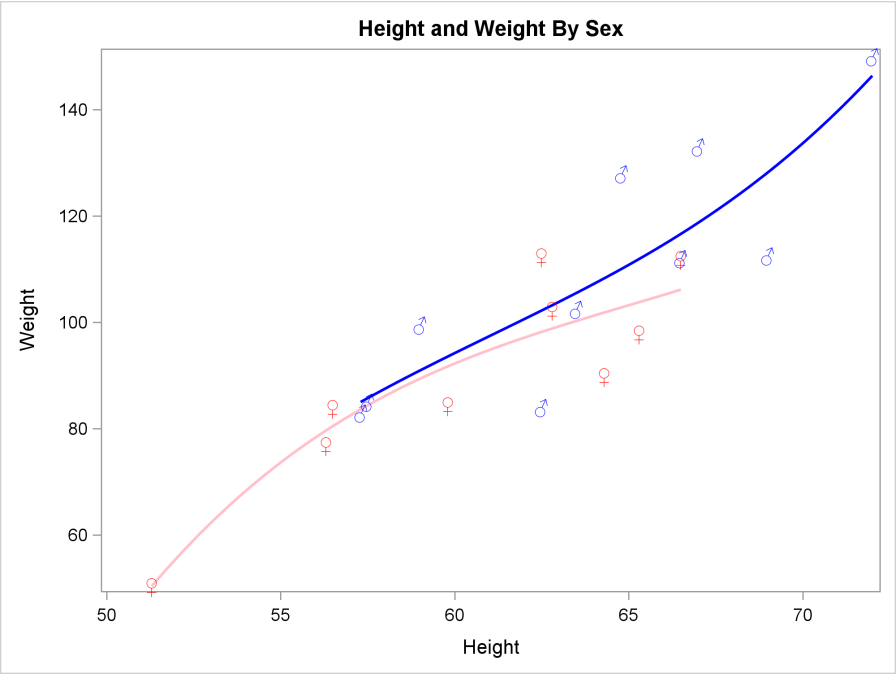
Figure 4.19 Annotation Data Set

x1	y1	Function	DrawSpace	Label	TextColor
69.0	112.5	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
56.5	84.0	Text	DataValue	(*ESC*){Unicode "2640"x}	red
65.3	98.0	Text	DataValue	(*ESC*){Unicode "2640"x}	red
62.8	102.5	Text	DataValue	(*ESC*){Unicode "2640"x}	red
63.5	102.5	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
57.3	83.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
59.8	84.5	Text	DataValue	(*ESC*){Unicode "2640"x}	red
62.5	112.5	Text	DataValue	(*ESC*){Unicode "2640"x}	red
62.5	84.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
59.0	99.5	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
51.3	50.5	Text	DataValue	(*ESC*){Unicode "2640"x}	red
64.3	90.0	Text	DataValue	(*ESC*){Unicode "2640"x}	red
56.3	77.0	Text	DataValue	(*ESC*){Unicode "2640"x}	red
66.5	112.0	Text	DataValue	(*ESC*){Unicode "2640"x}	red
72.0	150.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
64.8	128.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
67.0	133.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
57.5	85.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue
66.5	112.0	Text	DataValue	(*ESC*){Unicode "2642"x}	blue

This annotation data set is built from the same data set that provides the data to PROC SGPLOT. To provide the expected annotation variable names, the Weight variable is renamed to y1 and the Height variable is renamed to x1. All observations in the annotation data set provide text, and all coordinates are data values. The STYLEATTRS statement in PROC SGPLOT sets the line colors to blue and pink. The TextColor variable in the annotation data set controls the colors of the markers: red and blue. The markers are red and not pink because red markers as faint as these look pink, and pink markers are hard to see. It will become important to notice that the colors of the markers are explicitly controlled in the annotation data set, whereas the colors of the fit functions are controlled by PROC SGPLOT. The REG statement sets the size of the markers to 0 so that the Label variable from the annotation data set is displayed instead of the customary markers from the REG statement. The results are displayed in Figure 4.20.

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

Figure 4.20 Custom Markers

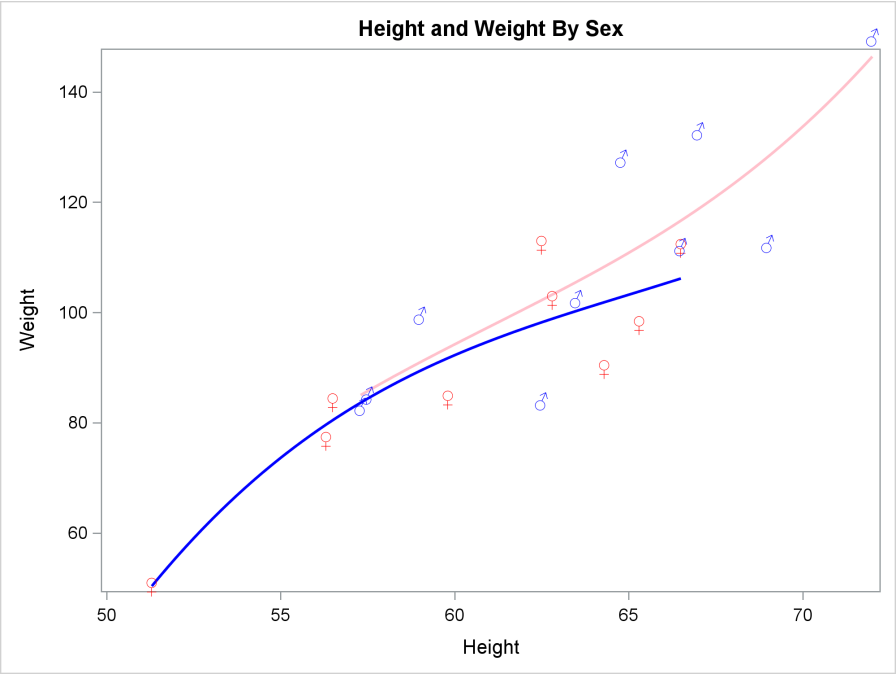


The following step shows what happens if you remove the markers by using the NOMARKERS option instead of setting the marker size to 0 (as specified by SIZE=0 in the previous step):

```
proc sgplot data=sashelp.class sganno=anno noautolegend;
   styleattrs datacontrastcolors=(blue pink);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3 nomarkers;
run;
```

The results are displayed in Figure 4.21.

Figure 4.21 Incorrect Use of the NOMARKERS Option



Compare the top right corners of Figure 4.20 and Figure 4.21. In Figure 4.20, space was reserved for the markers (by the size specification of 0) even though they were not displayed. In Figure 4.21, no space was reserved for the markers because the NOMARKERS option is specified. In Figure 4.21, the Y coordinate for the largest member of the class extends beyond the top of the Y axis, so one point appears outside the graph. This is the correct result for the specification, but the specification is wrong. It is important to understand that annotation is added after the graph is formed and that the graph size is based on only the values that are displayed by the plotting statements.

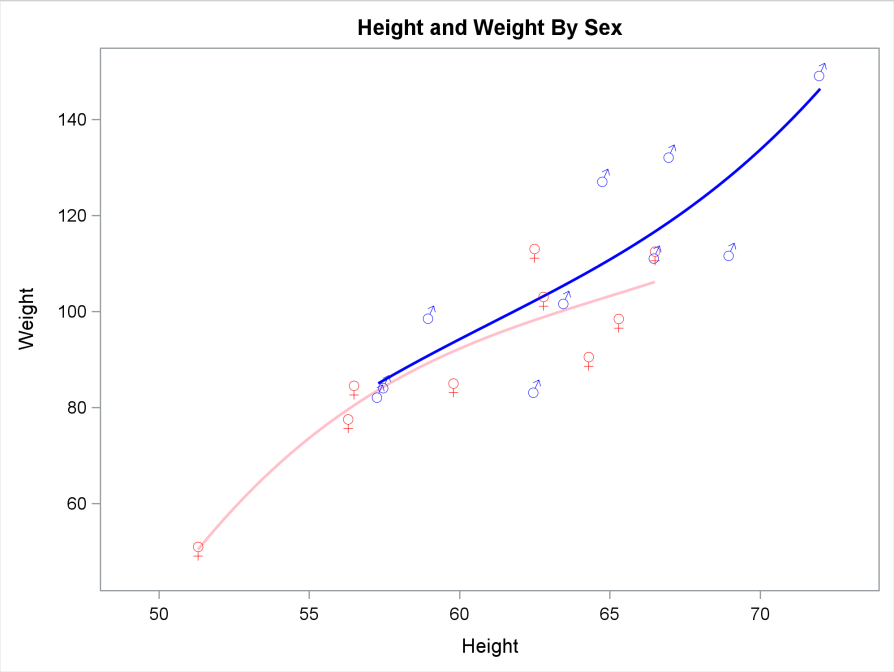
Also compare the fit functions in Figure 4.20 and Figure 4.21. In Figure 4.20, the points for males and the fit function are all blue. In Figure 4.21, the points for males are blue, but the fit function is pink. The annotation data set explicitly controls the colors for the markers. Fit function color is controlled in the usual way by PROC SGPLOT. Because the data set has two groups, the color for first group comes from the **`GraphData1`** style element and the color for second group comes from the **`GraphData2`** style element. When markers are present (even when their size is 0 and they are not visible), the first group corresponds to males, because the first student in the class data set is Alfred, a male. When markers are not present, the first group corresponds to first regression function, which is the female function.

You can also get the correct results by using offsets to move the points in and away from the axes and by modifying the order of the colors in the STYLEATTRS statement:

```
proc sgplot data=sashelp.class sganno=anno noautolegend;
   styleattrs datacontrastcolors=(pink blue);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3 nomarkers;
   xaxis offsetmin=0.075 offsetmax=0.075;
   yaxis offsetmin=0.075 offsetmax=0.075;
run;
```

The results are displayed in Figure 4.22.

Figure 4.22 Use of the NOMARKERS Option and Offsets

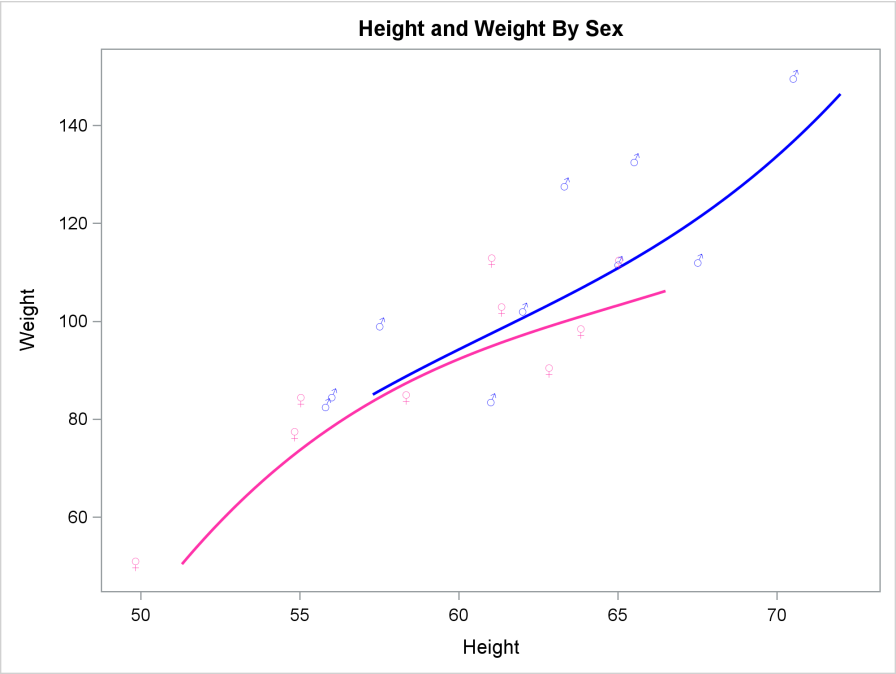


You can use a format to specify Unicode markers:

```
proc format;
   value $sexfmt 'F' = "(*ESC*){Unicode '2640'x}"
                 'M' = "(*ESC*){Unicode '2642'x}";
run;

proc sgplot data=sashelp.class noautolegend;
   styleattrs datacontrastcolors=(cxFF33AA blue);
   title 'Height and Weight By Sex';
   reg     y=weight x=height / group=sex degree=3 nomarkers;
   scatter y=weight x=height / group=sex markerchar=sex;
   xaxis offsetmin=0.05 offsetmax=0.05;
   yaxis offsetmin=0.05 offsetmax=0.05;
   format sex $sexfmt24.;
run;
```

The results are displayed in Figure 4.23.

Figure 4.23 Unicode Markers Controlled by a Format



Displaying Images in a Graph

Double Click for Example Code

This example shows how to display images in a graph by using the annotation facility. It builds on techniques discussed in previous examples. The following steps create and display the annotation data set and then create the graph:

```
data anno(drop=name sex age rename=(weight=y1 height=x1));
   set sashelp.class;
   retain Function 'Image' DrawSpace 'DataValue' Width 1.5;
   Image = 'images\' || ifc(sex eq 'F', 'female', 'male') || '.png';
run;

proc print noobs;
run;

proc sgplot data=sashelp.class sganno=anno noautolegend;
   styleattrs datacontrastcolors=(blue pink);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3 markerattrs=(size=0);
   xaxis offsetmin=0.05 offsetmax=0.05;
   yaxis offsetmin=0.05 offsetmax=0.05;
run;
```

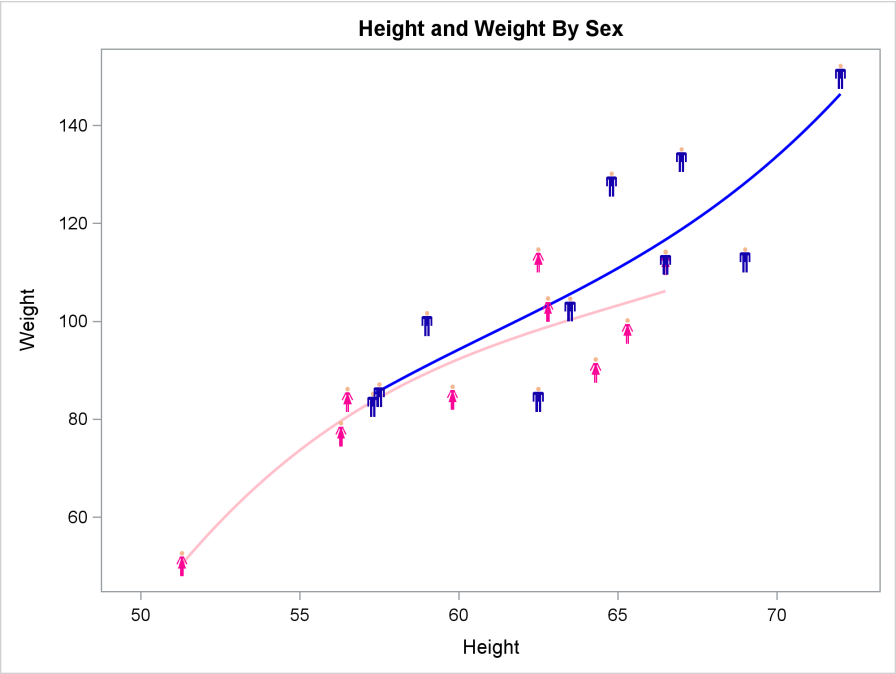
This annotation data set is built from the same data set that provides the data to PROC SGPLOT. To provide the expected annotation variable names, the Weight variable is renamed to y1 and the Height variable is renamed to x1. All observations in the annotation data set provide images, and all coordinates are data values. The images are stored in two PNG files, *female.png* (Double Click for Female Icon) and *male.png* (Double Click for Male Icon), which are stored in the *images* directory under the current working directory. In the SAS windowing environment, you can view and modify the current working directory by selecting **Tools ► Options ► Change Current Folder** from the menu at the top of the main SAS window. The Width variable provides the width of the image in percentages. The STYLEATTRS statement in PROC SGPLOT sets the line colors to blue and pink. The REG statement sets the size of the markers to 0 so that the images are displayed instead of markers. The annotation data set is displayed in Figure 4.24, and the results are displayed in Figure 4.25.

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

Figure 4.24 Annotation Data Set

x1	y1	Function	DrawSpace	Width	Image
69.0	112.5	Image	DataValue	1.5	images\male.png
56.5	84.0	Image	DataValue	1.5	images\female.png
65.3	98.0	Image	DataValue	1.5	images\female.png
62.8	102.5	Image	DataValue	1.5	images\female.png
63.5	102.5	Image	DataValue	1.5	images\male.png
57.3	83.0	Image	DataValue	1.5	images\male.png
59.8	84.5	Image	DataValue	1.5	images\female.png
62.5	112.5	Image	DataValue	1.5	images\female.png
62.5	84.0	Image	DataValue	1.5	images\male.png
59.0	99.5	Image	DataValue	1.5	images\male.png
51.3	50.5	Image	DataValue	1.5	images\female.png
64.3	90.0	Image	DataValue	1.5	images\female.png
56.3	77.0	Image	DataValue	1.5	images\female.png
66.5	112.0	Image	DataValue	1.5	images\female.png
72.0	150.0	Image	DataValue	1.5	images\male.png
64.8	128.0	Image	DataValue	1.5	images\male.png
67.0	133.0	Image	DataValue	1.5	images\male.png
57.5	85.0	Image	DataValue	1.5	images\male.png
66.5	112.0	Image	DataValue	1.5	images\male.png

Figure 4.25 Images in the Graph



The following steps create Figure 4.27 and use annotation to label each curve:

```
proc transreg data=sashelp.class;
   model identity(weight) = class(sex) | spline(height / degree=3) / solve;
   output out=fit p;
run;

proc summary data=fit;
   class sex;
   var height pweight;
   output out=max(where=(_type_ ne 0) drop=_freq_) max=x1 y1;
run;

data anno;
   retain Function 'Image' DrawSpace 'DataValue' Width 2.5;
   set max(drop=_type_ rename=(sex=Label));
   Image = 'images\' || ifc(label eq 'F', 'female', 'male') || '.png';
   x1 + 1;
run;

proc print noobs;
   title;
run;

ods graphics on / attrpriority=none;
proc sgplot data=sashelp.class sganno=anno noautolegend;
   styleattrs datacontrastcolors=(blue pink) datalinepatterns=(solid)
              datasymbols=(squarefilled circlefilled);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3;
   xaxis offsetmin=0.05 offsetmax=0.1;
   yaxis offsetmin=0.05 offsetmax=0.1;
run;
```

The first step uses PROC TRANSREG to fit the same model that the REG statement in PROC SGPLOT fits: a cubic polynomial function for each sex. PROC SUMMARY identifies the maximum X and Y coordinate for the predicted values for each sex. A DATA step creates the annotation data set, which is displayed in Figure 4.26. The statement `x1 + 1` is a sum statement, which is equivalent to `x1 = x1 + 1` and also retains x1 and initializes it to zero.

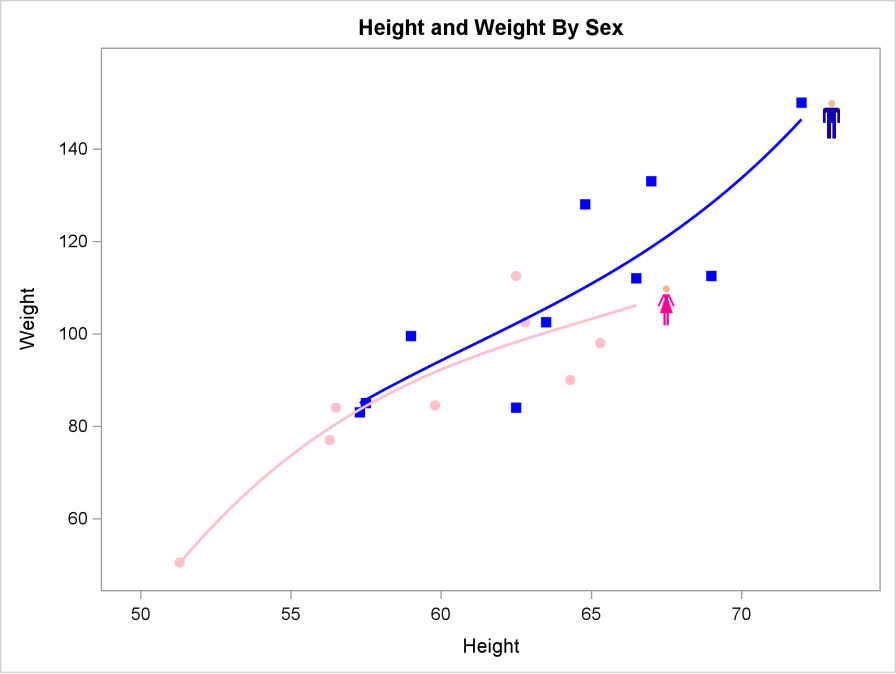
Figure 4.26 Annotation Data Set

Function	DrawSpace	Width	Label	x1	y1	Image
Image	DataValue	2.5	F	67.5	106.158	images\female.png
Image	DataValue	2.5	M	73.0	146.394	images\male.png

All observations in the annotation data set provide images, and all coordinates are data values. The image width is 2.5%. The X coordinate for each image is 1 plus the maximum to locate each image to the right of each curve.

The ODS GRAPHICS statement specifies ATTRPRIORITY=NONE, which enables markers to vary in each group. The STYLEATTRS statement specifies the colors, line pattern, and markers. The maximum offset is greater than the minimum offset in order to provide additional space for the images.

Figure 4.27 Images as Curve Labels



You can put curve labels outside the graph as follows:

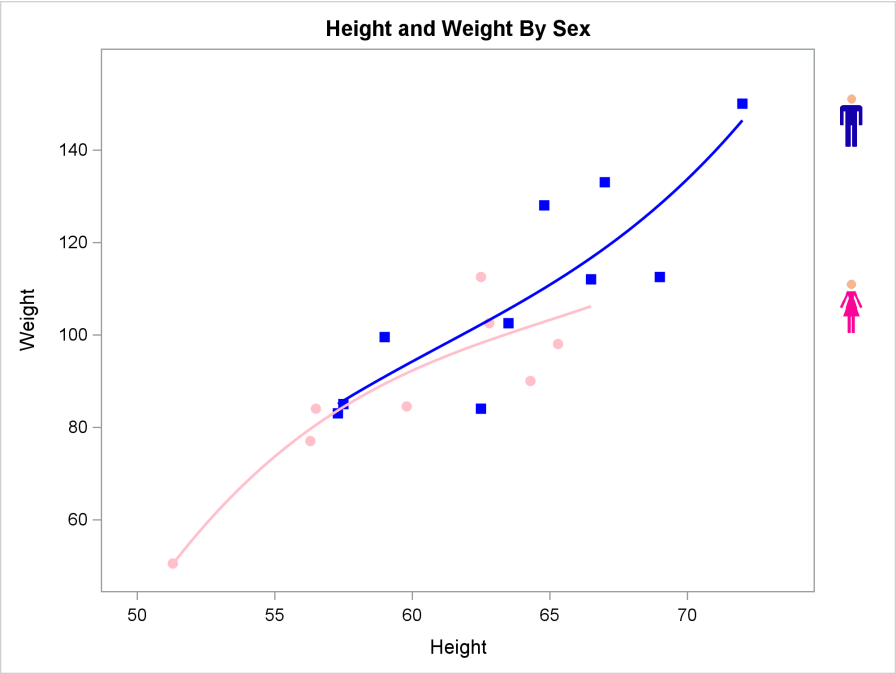
```
data anno;
   retain Function 'Image' y1Space 'DataValue' x1Space 'GraphPercent'
          Width 2.5 x1 95;
   set max(drop=_type_ rename=(sex=Label) drop=x1);
   Image = 'images\' || ifc(label eq 'F', 'female', 'male') || '.png';
run;

proc print noobs;
   title;
run;

ods graphics on / attrpriority=none;
proc sgplot data=sashelp.class sganno=anno noautolegend pad=(right=12%);
   styleattrs datacontrastcolors=(blue pink) datalinepatterns=(solid)
              datasymbols=(squarefilled circlefilled);
   title 'Height and Weight By Sex';
   reg y=weight x=height / group=sex degree=3;
   xaxis offsetmin=0.05 offsetmax=0.1;
   yaxis offsetmin=0.05 offsetmax=0.1;
run;
ods graphics on / reset=attrpriority;
```

This time, the annotation data set (shown in Figure 4.28) sets the X-axis drawing space to `'GraphPercent'` and the X-axis coordinate to a constant 95. In the PROC SGPLOT step, the PAD= option reserves the rightmost 12% of the axis for the curve labels. The results are displayed in Figure 4.29.

Figure 4.28 Annotation Data Set

Function	y1Space	x1Space	Width	x1	Label	y1	Image
Image	DataValue	GraphPercent	2.5	95	F	106.158	images\female.png
Image	DataValue	GraphPercent	2.5	95	M	146.394	images\male.png

Figure 4.29 Curve Labels outside the Graph



Lines, Circles, Ovals, Rectangles, and Other Shapes

Double Click for Example Code

This example shows how to draw some basic shapes. The first part of this example uses annotation to draw a circle, oval, square, rectangle, and triangle. Subsequent parts illustrate options for controlling fill and line styles. The following steps create and display the annotation data set and then create the graph:

```
data x;
   input x y @@;
   datalines;
1 1 -1 -1
;

data anno;
   retain DrawSpace 'DataValue' Function 'Oval     '
          HeightUnit WidthUnit 'Data' x1 y1 0;;
                           Height = 0.5; Width = 0.5; output;
                                         Width = 1;   output;
   function = 'Rectangle'; Height = 1.4; Width = 1.9; output;
   function = 'Rectangle'; Height = 1;   Width = 1;   output;
   heightunit = ' '; widthunit = ' '; height = .; width=.;
   function = 'Polygon';   x1 = -1; y1 = -1;          output;
   function = 'PolyCont';  x1 =  1;                   output;
   function = 'PolyCont';  x1 =  0; y1 =  1;          output;
run;
```

```
proc print noobs;
run;

ods graphics on / width=4.8in height=4.8in;
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
```

The annotation data set is displayed in Figure 4.30, and the results are displayed in Figure 4.31. The graph is square because of the square graph area of 4.8 × 4.8 inches. The annotation code draws the following:

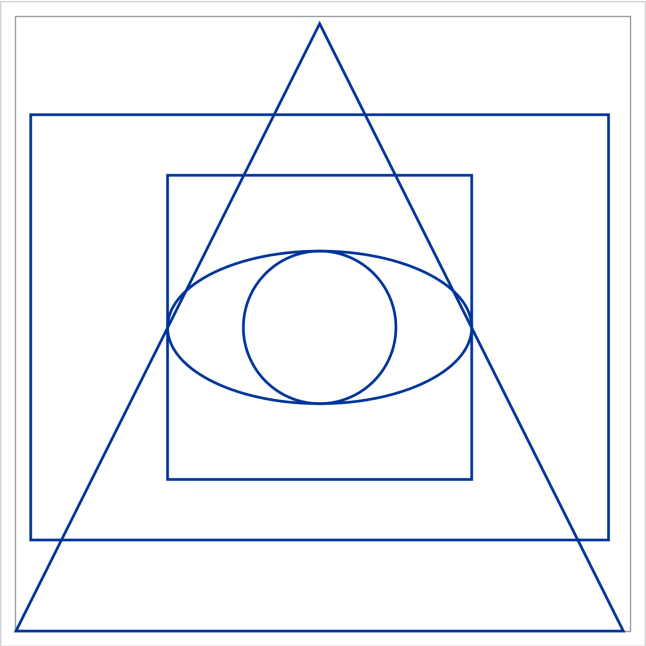
- a circle by setting the function to `'Oval'` and the height equal to the width
- an oval
- a square by setting the function to `'Rectangle'` and the height equal to the width
- a rectangle
- a triangle by setting the function to `'Polygon'` to start drawing at one vertex and then setting the function to `'PolyCont'` to draw the first two sides. The final side is automatically drawn by Function=`'Polygon'`.

The DATA step sets the height and width variables to missing before drawing the triangle. This is unnecessary because these variables are ignored in polygons, but it makes a cleaner-looking annotation data set.

Figure 4.30 Annotation Data Set

DrawSpace	Function	HeightUnit	WidthUnit	x1	y1	Height	Width
DataValue	Oval	Data	Data	0	0	0.5	0.5
DataValue	Oval	Data	Data	0	0	0.5	1.0
DataValue	Rectangle	Data	Data	0	0	1.4	1.9
DataValue	Rectangle	Data	Data	0	0	1.0	1.0
DataValue	Polygon			-1	-1	.	.
DataValue	PolyCont			1	-1	.	.
DataValue	PolyCont			0	1	.	.

Figure 4.31 Basic Shapes



The next part of this example shows the various options for filling or not filling polygons:

```
data anno;
   retain DrawSpace 'DataValue' Function 'Oval     ' Display '       '
          HeightUnit WidthUnit 'Data' x1 y1 0 Height 0.3 Width 0.5;
   function = 'Oval';      width = 0.3; x1 = -0.7; y1 = -0.8;  output;
   function = 'Rectangle'; width = 0.5; x1 =  0.7; y1 = -0.8;  output;

   function = 'Polygon';                x1 = -0.2; y1 = -0.9;  output;
   function = 'PolyCont';               x1 =  0.2;             output;
                                        x1 =  0;   y1 = -0.7;  output;
run;

data anno;
   length Label $ 20;
   i = 1; set anno point=i;
   Function = 'Text'; x1 = 0; y1 = 0.9; Label   = 'Default';       output;
   do i = 1 to 5; set anno point=i;     display = ' ';   y1 + 1.5; output; end;
   Function = 'Text'; x1 = 0; y1 = 0.4; Label   = 'Outline';       output;
   do i = 1 to 5; set anno point=i;     display = label; y1 + 1.0; output; end;
   Function = 'Text'; x1 = 0; y1 = -.1; Label   = 'Fill';          output;
   do i = 1 to 5; set anno point=i;     display = label; y1 + 0.5; output; end;
   Function = 'Text'; x1 = 0; y1 = -.6; Label   = 'All';           output;
   do i = 1 to 5; set anno point=i;     display = label;           output; end;
   stop;
run;

proc print noobs;
run;
```

```
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
```

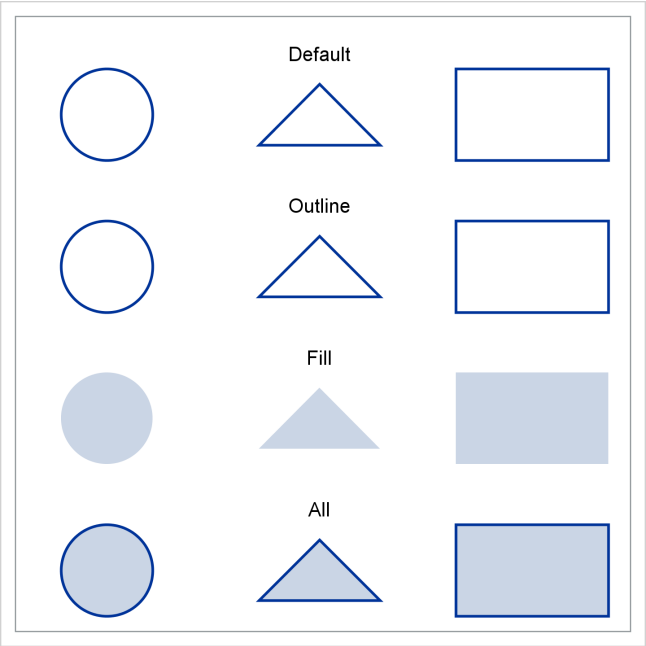
The annotation data set is displayed in Figure 4.32, and the graph is displayed in Figure 4.33.

Figure 4.32 Annotation Data Set

Label	DrawSpace	Function	Display	HeightUnit	WidthUnit	x1	y1	Height	Width
Default	DataValue	Text		Data	Data	0.0	0.9	0.3	0.3
Default	DataValue	Oval		Data	Data	-0.7	0.7	0.3	0.3
Default	DataValue	Rectangle		Data	Data	0.7	0.7	0.3	0.5
Default	DataValue	Polygon		Data	Data	-0.2	0.6	0.3	0.5
Default	DataValue	PolyCont		Data	Data	0.2	0.6	0.3	0.5
Default	DataValue	PolyCont		Data	Data	0.0	0.8	0.3	0.5
Outline	DataValue	Text		Data	Data	0.0	0.4	0.3	0.5
Outline	DataValue	Oval	Outline	Data	Data	-0.7	0.2	0.3	0.3
Outline	DataValue	Rectangle	Outline	Data	Data	0.7	0.2	0.3	0.5
Outline	DataValue	Polygon	Outline	Data	Data	-0.2	0.1	0.3	0.5
Outline	DataValue	PolyCont	Outline	Data	Data	0.2	0.1	0.3	0.5
Outline	DataValue	PolyCont	Outline	Data	Data	0.0	0.3	0.3	0.5
Fill	DataValue	Text	Outline	Data	Data	0.0	-0.1	0.3	0.5
Fill	DataValue	Oval	Fill	Data	Data	-0.7	-0.3	0.3	0.3
Fill	DataValue	Rectangle	Fill	Data	Data	0.7	-0.3	0.3	0.5
Fill	DataValue	Polygon	Fill	Data	Data	-0.2	-0.4	0.3	0.5
Fill	DataValue	PolyCont	Fill	Data	Data	0.2	-0.4	0.3	0.5
Fill	DataValue	PolyCont	Fill	Data	Data	0.0	-0.2	0.3	0.5
All	DataValue	Text	Fill	Data	Data	0.0	-0.6	0.3	0.5
All	DataValue	Oval	All	Data	Data	-0.7	-0.8	0.3	0.3
All	DataValue	Rectangle	All	Data	Data	0.7	-0.8	0.3	0.5
All	DataValue	Polygon	All	Data	Data	-0.2	-0.9	0.3	0.5
All	DataValue	PolyCont	All	Data	Data	0.2	-0.9	0.3	0.5
All	DataValue	PolyCont	All	Data	Data	0.0	-0.7	0.3	0.5

The first row corresponds to the default, Display=`' '` (or no Display variable). The first row is the same as the second row, which has the default Display=`'Outline'`. The third row displays Display=`'Fill'`. The fourth row displays Display=`'All'`

Figure 4.33 Fill Options



The next steps set Display='All' for all observations, control the fill colors by setting the variable FillStyleElement to **GraphData1** through **GraphData4** for rows 1 to 4, and set the FillTransparency for the circles to 0 (no transparency), the triangles to 0.4 (some transparency), and the rectangles to 0.9 (mostly transparent):

```
data anno(drop=i);
   retain FillStyleElement '           ' FillTransparency 0;
   set anno end=eof;
   Display = 'All';
   if function = 'Text' then do;
      i + 1;
      FillStyleElement = cats('GraphData', i);
      label = FillStyleElement;
      Width = 50;
   end;
   else label = ' ';
   FillTransparency = ifn(function = 'Oval', 0,
                      ifn(function = 'Polygon', 0.4, 0.9));
   output;
   if eof then do;
      Function = 'Text'; label='Less Transparent'; x1 = -.7; y1 = 0.95; output;
                         label='More Transparent'; x1 = 0.7;            output;
   end;
run;

proc print noobs;
run;
```

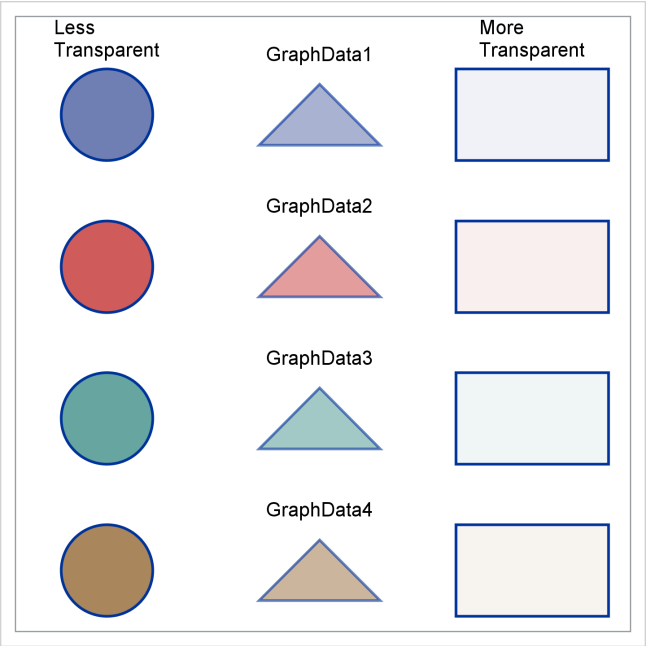
```
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
```

The annotation data set is displayed in Figure 4.34, and the graph is displayed in Figure 4.35.

Figure 4.34 Annotation Data Set

FillStyleElement	FillTransparency	Label	DrawSpace	Function	Display	HeightUnit	WidthUnit	x1	y1	Height	Width
GraphData1	0.9	GraphData1	DataValue	Text	All	Data	Data	0.0	0.90	0.3	50.0
GraphData1	0.0		DataValue	Oval	All	Data	Data	-0.7	0.70	0.3	0.3
GraphData1	0.9		DataValue	Rectangle	All	Data	Data	0.7	0.70	0.3	0.5
GraphData1	0.4		DataValue	Polygon	All	Data	Data	-0.2	0.60	0.3	0.5
GraphData1	0.9		DataValue	PolyCont	All	Data	Data	0.2	0.60	0.3	0.5
GraphData1	0.9		DataValue	PolyCont	All	Data	Data	0.0	0.80	0.3	0.5
GraphData2	0.9	GraphData2	DataValue	Text	All	Data	Data	0.0	0.40	0.3	50.0
GraphData2	0.0		DataValue	Oval	All	Data	Data	-0.7	0.20	0.3	0.3
GraphData2	0.9		DataValue	Rectangle	All	Data	Data	0.7	0.20	0.3	0.5
GraphData2	0.4		DataValue	Polygon	All	Data	Data	-0.2	0.10	0.3	0.5
GraphData2	0.9		DataValue	PolyCont	All	Data	Data	0.2	0.10	0.3	0.5
GraphData2	0.9		DataValue	PolyCont	All	Data	Data	0.0	0.30	0.3	0.5
GraphData3	0.9	GraphData3	DataValue	Text	All	Data	Data	0.0	-0.10	0.3	50.0
GraphData3	0.0		DataValue	Oval	All	Data	Data	-0.7	-0.30	0.3	0.3
GraphData3	0.9		DataValue	Rectangle	All	Data	Data	0.7	-0.30	0.3	0.5
GraphData3	0.4		DataValue	Polygon	All	Data	Data	-0.2	-0.40	0.3	0.5
GraphData3	0.9		DataValue	PolyCont	All	Data	Data	0.2	-0.40	0.3	0.5
GraphData3	0.9		DataValue	PolyCont	All	Data	Data	0.0	-0.20	0.3	0.5
GraphData4	0.9	GraphData4	DataValue	Text	All	Data	Data	0.0	-0.60	0.3	50.0
GraphData4	0.0		DataValue	Oval	All	Data	Data	-0.7	-0.80	0.3	0.3
GraphData4	0.9		DataValue	Rectangle	All	Data	Data	0.7	-0.80	0.3	0.5
GraphData4	0.4		DataValue	Polygon	All	Data	Data	-0.2	-0.90	0.3	0.5
GraphData4	0.9		DataValue	PolyCont	All	Data	Data	0.2	-0.90	0.3	0.5
GraphData4	0.9		DataValue	PolyCont	All	Data	Data	0.0	-0.70	0.3	0.5
GraphData4	0.9	Less Transparent	DataValue	Text	All	Data	Data	-0.7	0.95	0.3	0.5
GraphData4	0.9	More Transparent	DataValue	Text	All	Data	Data	0.7	0.95	0.3	0.5

Figure 4.35 FillStyleElement and FillTransparency



These next steps create four sets of rotated shapes:

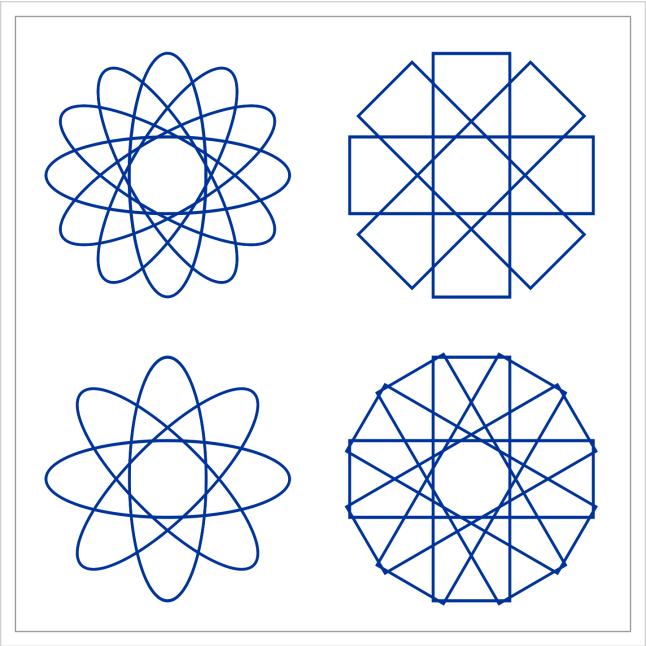
```
data anno;
   retain DrawSpace 'DataValue' Function 'Oval     '
          HeightUnit WidthUnit 'Data' x1 y1 -0.5 Height 0.25 Width .8;
                                      do Rotate = 0 to 315 by 45; output; end;
                           y1 =  0.5; do Rotate = 0 to 330 by 30; output; end;
   function = 'Rectangle'; x1 =  0.5; do Rotate = 0 to 315 by 45; output; end;
                           y1 = -0.5; do Rotate = 0 to 330 by 30; output; end;
run;

proc print noobs;
run;

proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
```

The annotation data set is displayed in Figure 4.36, and the results are displayed in Figure 4.37. The bottom left of the graph has a few rotated ovals, the top left has a few more, the top right has a few rotated rectangles, and the bottom right has a few more. The annotation variable Rotate contains rotations in degrees.

Figure 4.36 Annotation Data Set

DrawSpace	Function	HeightUnit	WidthUnit	x1	y1	Height	Width	Rotate
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	0
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	45
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	90
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	135
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	180
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	225
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	270
DataValue	Oval	Data	Data	-0.5	-0.5	0.25	0.8	315
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	0
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	30
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	60
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	90
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	120
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	150
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	180
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	210
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	240
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	270
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	300
DataValue	Oval	Data	Data	-0.5	0.5	0.25	0.8	330
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	0
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	45
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	90
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	135
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	180
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	225
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	270
DataValue	Rectangle	Data	Data	0.5	0.5	0.25	0.8	315
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	0
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	30
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	60
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	90
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	120
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	150
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	180
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	210
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	240
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	270
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	300
DataValue	Rectangle	Data	Data	0.5	-0.5	0.25	0.8	330

Figure 4.37 Rotated Shapes



These next steps create four shapes (two octagons, a hexagon, and a pentagon):

```
data x;
   input x y @@;
   datalines;
2 2 -2 -2
;

data anno(drop=pi radians sides c1 c2 shift scale);
   pi = constant('pi');
   retain Function '        ' DrawSpace 'DataValue' LineColor 'Magenta';
   c1 = -1; c2 =  1;
   do radians = 0 to 7 * pi / 4 by pi / 4;
      function = ifc(radians = 0, 'Polygon', 'PolyCont');
      x1 = c1 + cos(radians);
      y1 = c2 + sin(radians);
      output;
   end;

   LineColor = 'Blue';   c2 = -1;
   do radians = 0 to 5 * pi / 3 by pi / 3;
      function = ifc(radians = 0, 'Polygon', 'PolyCont');
      x1 = c1 + cos(radians);
      y1 = c2 + sin(radians);
      output;
   end;

   LineColor = 'Red';    c1 =  1; c2 =  1;
   sides = 8; shift = -pi / sides;
   do radians = 0 to 2 * pi * (sides - 1) / sides by 2 * pi / sides;
```

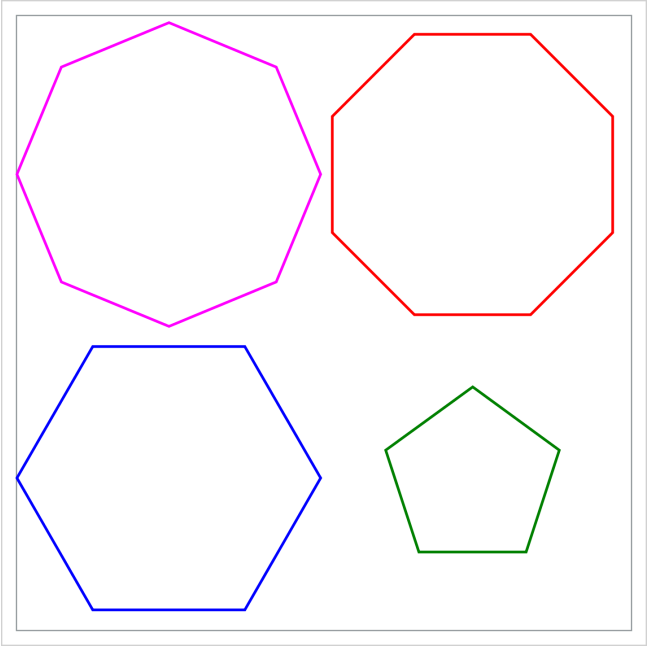
```
      function = ifc(radians = 0, 'Polygon', 'PolyCont');
      x1 = c1 + cos(radians + shift);
      y1 = c2 + sin(radians + shift);
      output;
   end;

   LineColor = 'Green';  c2 = -1;
   sides = 5; scale = 0.6; shift = 0.5 * pi / sides;
   do radians = 0 to 2 * pi * (sides - 1) / sides by 2 * pi / sides;
      function = ifc(radians = 0, 'Polygon', 'PolyCont');
      x1 = c1 + scale * cos(radians + shift);
      y1 = c2 + scale * sin(radians + shift);
      output;
   end;
run;

proc print noobs;
run;

proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
ods graphics on / reset=all;
```

The annotation data set is displayed in Figure 4.38, and the results are displayed in Figure 4.39. The vertices of regular polygons can be constructed by selecting evenly spaced points along a circle. The magenta octagon on the top left is constructed using ad hoc code. Cosines and sines provide the X and Y coordinates for an octagon centered at (–1, 1). The blue hexagon on the bottom left is constructed by using similar ad hoc code. The red octagon on the top right is constructed from a different set of points from those used to construct the magenta octagon. The red octagon starts at $(\cos(-\pi/8), \sin(-\pi/8))$ instead of $(\cos(0), \sin(0))$, which rotates the shape to the customary stop-sign orientation. The green pentagon on the bottom right further generalizes the method by adding a scaling factor (in this case 0.8) that changes the size of the polygon.

Figure 4.38 Annotation Data Set

Function	DrawSpace	LineColor	x1	y1
Polygon	DataValue	Magenta	0.00000	1.00000
PolyCont	DataValue	Magenta	-0.29289	1.70711
PolyCont	DataValue	Magenta	-1.00000	2.00000
PolyCont	DataValue	Magenta	-1.70711	1.70711
PolyCont	DataValue	Magenta	-2.00000	1.00000
PolyCont	DataValue	Magenta	-1.70711	0.29289
PolyCont	DataValue	Magenta	-1.00000	0.00000
PolyCont	DataValue	Magenta	-0.29289	0.29289
Polygon	DataValue	Blue	0.00000	-1.00000
PolyCont	DataValue	Blue	-0.50000	-0.13397
PolyCont	DataValue	Blue	-1.50000	-0.13397
PolyCont	DataValue	Blue	-2.00000	-1.00000
PolyCont	DataValue	Blue	-1.50000	-1.86603
PolyCont	DataValue	Blue	-0.50000	-1.86603
Polygon	DataValue	Red	1.92388	0.61732
PolyCont	DataValue	Red	1.92388	1.38268
PolyCont	DataValue	Red	1.38268	1.92388
PolyCont	DataValue	Red	0.61732	1.92388
PolyCont	DataValue	Red	0.07612	1.38268
PolyCont	DataValue	Red	0.07612	0.61732
PolyCont	DataValue	Red	0.61732	0.07612
PolyCont	DataValue	Red	1.38268	0.07612
Polygon	DataValue	Green	1.57063	-0.81459
PolyCont	DataValue	Green	1.00000	-0.40000
PolyCont	DataValue	Green	0.42937	-0.81459
PolyCont	DataValue	Green	0.64733	-1.48541
PolyCont	DataValue	Green	1.35267	-1.48541

Figure 4.39 Other Shapes



The next steps control the line colors by setting the variable LineStyleElement to **`GraphData1`** through **`GraphData4`**, and control the line pattern and line thickness:

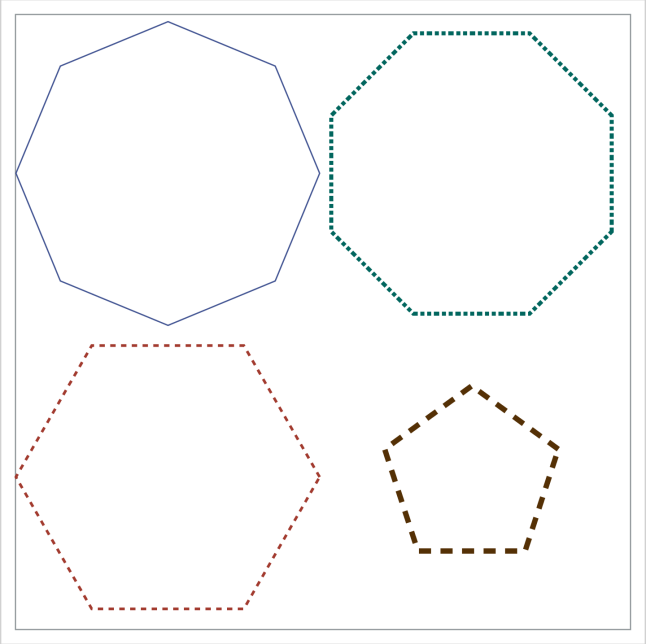
```
data anno(drop=linecolor);
   set anno;
   select (LineColor);
      when ('Magenta') do;
         LineStyleElement = 'GraphData1';
         LinePattern      = 1; /* Solid */
         LineThickness    = 1;
      end;
      when ('Blue') do;
         LineStyleElement = 'GraphData2';
         LinePattern      = 2; /* ShortDash */
         LineThickness    = 2;
      end;
      when ('Red') do;
         LineStyleElement = 'GraphData3';
         LinePattern      = 3; /* (shorter dash that has no named alias) */
         LineThickness    = 3;
      end;
      when ('Green') do;
         LineStyleElement = 'GraphData4';
         LinePattern      = 20; /* Dash */
         LineThickness    = 4;
      end;
      otherwise;
   end;
   output;
run;

proc print noobs;
run;

proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
```

The annotation data set is displayed in Figure 4.40, and the graph is displayed in Figure 4.41. The LinePattern variable can be character or numeric. For more a complete list of numeric and character values, see Figure 4.74.

Figure 4.40 Annotation Data Set

Function	DrawSpace	x1	y1	LineStyleElement	LinePattern	LineThickness
Polygon	DataValue	0.00000	1.00000	GraphData1	1	1
PolyCont	DataValue	-0.29289	1.70711	GraphData1	1	1
PolyCont	DataValue	-1.00000	2.00000	GraphData1	1	1
PolyCont	DataValue	-1.70711	1.70711	GraphData1	1	1
PolyCont	DataValue	-2.00000	1.00000	GraphData1	1	1
PolyCont	DataValue	-1.70711	0.29289	GraphData1	1	1
PolyCont	DataValue	-1.00000	0.00000	GraphData1	1	1
PolyCont	DataValue	-0.29289	0.29289	GraphData1	1	1
Polygon	DataValue	0.00000	-1.00000	GraphData2	2	2
PolyCont	DataValue	-0.50000	-0.13397	GraphData2	2	2
PolyCont	DataValue	-1.50000	-0.13397	GraphData2	2	2
PolyCont	DataValue	-2.00000	-1.00000	GraphData2	2	2
PolyCont	DataValue	-1.50000	-1.86603	GraphData2	2	2
PolyCont	DataValue	-0.50000	-1.86603	GraphData2	2	2
Polygon	DataValue	1.92388	0.61732	GraphData3	3	3
PolyCont	DataValue	1.92388	1.38268	GraphData3	3	3
PolyCont	DataValue	1.38268	1.92388	GraphData3	3	3
PolyCont	DataValue	0.61732	1.92388	GraphData3	3	3
PolyCont	DataValue	0.07612	1.38268	GraphData3	3	3
PolyCont	DataValue	0.07612	0.61732	GraphData3	3	3
PolyCont	DataValue	0.61732	0.07612	GraphData3	3	3
PolyCont	DataValue	1.38268	0.07612	GraphData3	3	3
Polygon	DataValue	1.57063	-0.81459	GraphData4	20	4
PolyCont	DataValue	1.00000	-0.40000	GraphData4	20	4
PolyCont	DataValue	0.42937	-0.81459	GraphData4	20	4
PolyCont	DataValue	0.64733	-1.48541	GraphData4	20	4
PolyCont	DataValue	1.35267	-1.48541	GraphData4	20	4

Figure 4.41 Line Options



For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

Watermarks

Double Click for Example Code

You can use annotation to add watermarks to graphs. The watermarks can contain words such as `'Draft'` or `'Do Not Circulate'`, or they can contain the date the graph was created. The following steps illustrate:

```
data anno;
   retain Function 'Text' Width 200 Rotate -38 Transparency 0.8
          TextSize 25 Layer 'Back';
   Label = left(put(date(), weekdate32.));
run;

proc print noobs;
run;

ods graphics on / width=4.8in height=4.8in;

proc sgplot data=sashelp.iris sganno=anno nowall;
   pbspline y=petalwidth x=petallength / group=species;
run;
```

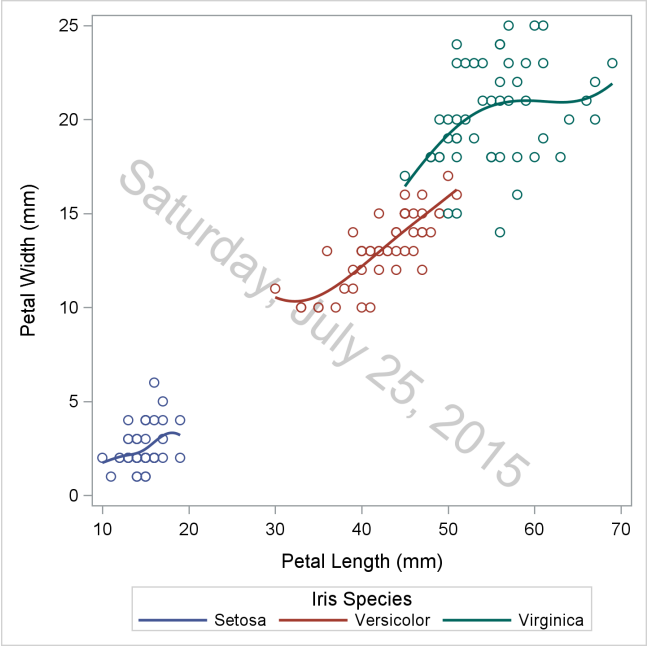
The annotation data set is displayed in Figure 4.42, and the results are displayed in Figure 4.43. This is one of the few examples in which you do not need to specify coordinates. The Layer variable places the watermark in the background of the graph behind the points. You must disable the graph wall by specifying

the NOWALL option in PROC SGPLOT in order for Layer='Back' to have a visible effect. Without the NOWALL option, the watermark is hidden by the graph wall. The graph wall is a nontransparent box (white in this case) that fills the axes. The width is greater than 100 because a rotated label that is more than 141% of the width of a square graph can fit on the diagonal. The annotation variable Transparency sets the transparency of the watermark on a scale from 0 (not at all transparent) to 1 (completely transparent).

Figure 4.42 Annotation Data Set That Creates a Date Watermark

Function	Width	Rotate	Transparency	TextSize	Layer	Label
Text	200	-38	0.8	25	Back	Saturday, July 25, 2015

Figure 4.43 Graph with a Data Watermark



You can position the watermark more precisely on the diagonal of the graph by specifying X1 and Y1 coordinates:

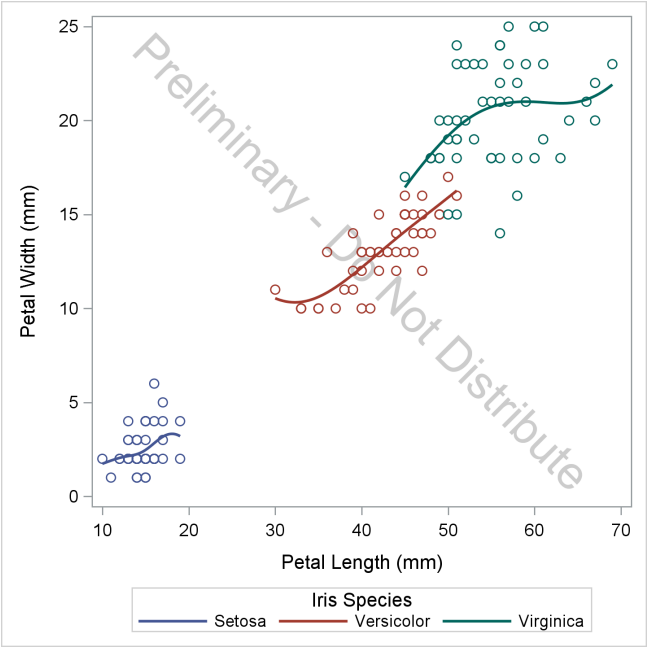
```
data anno;
   retain Function 'Text' Width 200 Rotate -45 Transparency 0.8
          TextSize 24 Layer 'Back' DrawSpace 'DataPercent' x1 50 y1 50;
   Label = 'Preliminary - Do Not Distribute';
run;

proc print noobs;
run;

proc sgplot data=sashelp.iris sganno=anno nowall;
   pbspline y=petalwidth x=petallength / group=species;
run;
ods graphics on / reset=all;
```

The annotation data set is displayed in Figure 4.44, and the results are displayed in Figure 4.45.

Figure 4.44 Annotation Data Set Provides a Watermark on the Diagonal

Function	Width	Rotate	Transparency	TextSize	Layer	DrawSpace	x1	y1	Label
Text	200	-45	0.8	24	Back	DataPercent	50	50	Preliminary - Do Not Distribute

Figure 4.45 Graph with a Data Watermark on the Diagonal



Rotating Text

Double Click for Example Code

These next steps illustrate the angle of rotation for a text string that has a left anchor:

```
data x;
   input x y @@;
   datalines;
2 2 -2 -2
;

data anno;
   length Label $ 20;
   retain DrawSpace 'DataValue' Function 'Text'
          Width 100 TextSize 15 y1 0 Anchor 'Left';
   x1 = -1.2;
   do Rotate = 0 to 315 by 45;
      Label = '    ' || catx(' ', 'Rotation', rotate);
      output;
   end;
```

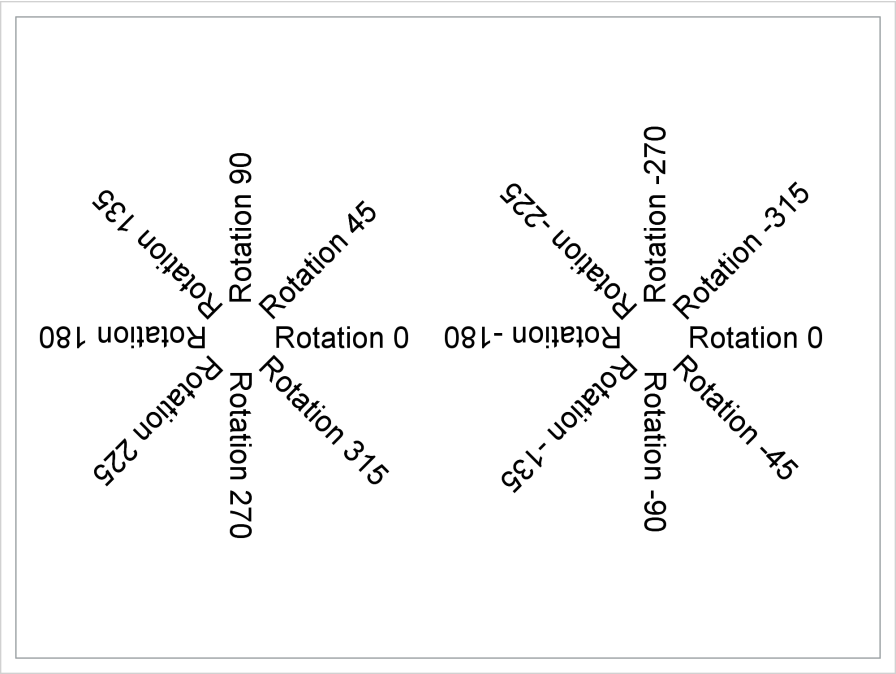
```
   x1 = 1.2;
   do Rotate = 0 to -315 by -45;
      Label = '    ' ||  catx(' ', 'Rotation', rotate);
      output;
   end;
run;

proc print noobs;
run;

proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none offsetmin=0.1 offsetmax=0.1;
   yaxis display=none offsetmin=0.1 offsetmax=0.1;
run;
```

The annotation data set is displayed in Figure 4.46, and the results are displayed in Figure 4.47.

Figure 4.46 Annotation Data Set

Label	DrawSpace	Function	Width	TextSize	y1	Anchor	x1	Rotate
Rotation 0	DataValue	Text	100	15	0	Left	-1.2	0
Rotation 45	DataValue	Text	100	15	0	Left	-1.2	45
Rotation 90	DataValue	Text	100	15	0	Left	-1.2	90
Rotation 135	DataValue	Text	100	15	0	Left	-1.2	135
Rotation 180	DataValue	Text	100	15	0	Left	-1.2	180
Rotation 225	DataValue	Text	100	15	0	Left	-1.2	225
Rotation 270	DataValue	Text	100	15	0	Left	-1.2	270
Rotation 315	DataValue	Text	100	15	0	Left	-1.2	315
Rotation 0	DataValue	Text	100	15	0	Left	1.2	0
Rotation -45	DataValue	Text	100	15	0	Left	1.2	-45
Rotation -90	DataValue	Text	100	15	0	Left	1.2	-90
Rotation -135	DataValue	Text	100	15	0	Left	1.2	-135
Rotation -180	DataValue	Text	100	15	0	Left	1.2	-180
Rotation -225	DataValue	Text	100	15	0	Left	1.2	-225
Rotation -270	DataValue	Text	100	15	0	Left	1.2	-270
Rotation -315	DataValue	Text	100	15	0	Left	1.2	-315

Figure 4.47 Rotation from a Left Anchor



These next steps illustrate the angle of rotation for a text string that has a right anchor:

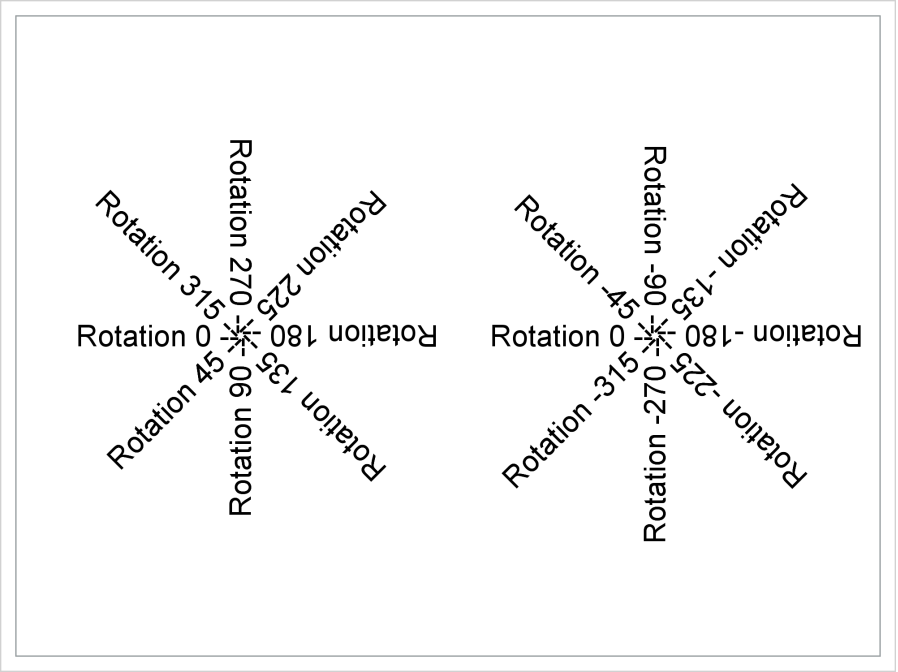
```
data anno;
   length Label $ 30;
   retain DrawSpace 'DataValue' Function 'Text'
          Width 100 TextSize 15 y1 0 Anchor 'Right';
   x1 = -1.2;
   do Rotate = 0 to 315 by 45;
      Label = catx(' ', 'Rotation', rotate, '--');
      output;
   end;
   x1 = 1.2;
   do Rotate = 0 to -315 by -45;
      Label = catx(' ', 'Rotation', rotate, '--');
      output;
   end;
run;

proc print noobs;
run;

proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none offsetmin=0.1 offsetmax=0.1;
   yaxis display=none offsetmin=0.1 offsetmax=0.1;
run;
```

The annotation data set is displayed in Figure 4.46, and the results are displayed in Figure 4.47.

Figure 4.48 Annotation Data Set

Label	DrawSpace	Function	Width	TextSize	y1	Anchor	x1	Rotate
Rotation 0 --	DataValue	Text	100	15	0	Right	-1.2	0
Rotation 45 --	DataValue	Text	100	15	0	Right	-1.2	45
Rotation 90 --	DataValue	Text	100	15	0	Right	-1.2	90
Rotation 135 --	DataValue	Text	100	15	0	Right	-1.2	135
Rotation 180 --	DataValue	Text	100	15	0	Right	-1.2	180
Rotation 225 --	DataValue	Text	100	15	0	Right	-1.2	225
Rotation 270 --	DataValue	Text	100	15	0	Right	-1.2	270
Rotation 315 --	DataValue	Text	100	15	0	Right	-1.2	315
Rotation 0 --	DataValue	Text	100	15	0	Right	1.2	0
Rotation -45 --	DataValue	Text	100	15	0	Right	1.2	-45
Rotation -90 --	DataValue	Text	100	15	0	Right	1.2	-90
Rotation -135 --	DataValue	Text	100	15	0	Right	1.2	-135
Rotation -180 --	DataValue	Text	100	15	0	Right	1.2	-180
Rotation -225 --	DataValue	Text	100	15	0	Right	1.2	-225
Rotation -270 --	DataValue	Text	100	15	0	Right	1.2	-270
Rotation -315 --	DataValue	Text	100	15	0	Right	1.2	-315

Figure 4.49 Rotation from a Right Anchor



These next steps illustrate the angle of rotation for a text string that has a center anchor:

```
data anno;
   length Label $ 30;
   retain DrawSpace 'DataValue' Function 'Text'
          Width 100 TextSize 12 Anchor 'Center';
   y1 = 1; x1 = -1;
```

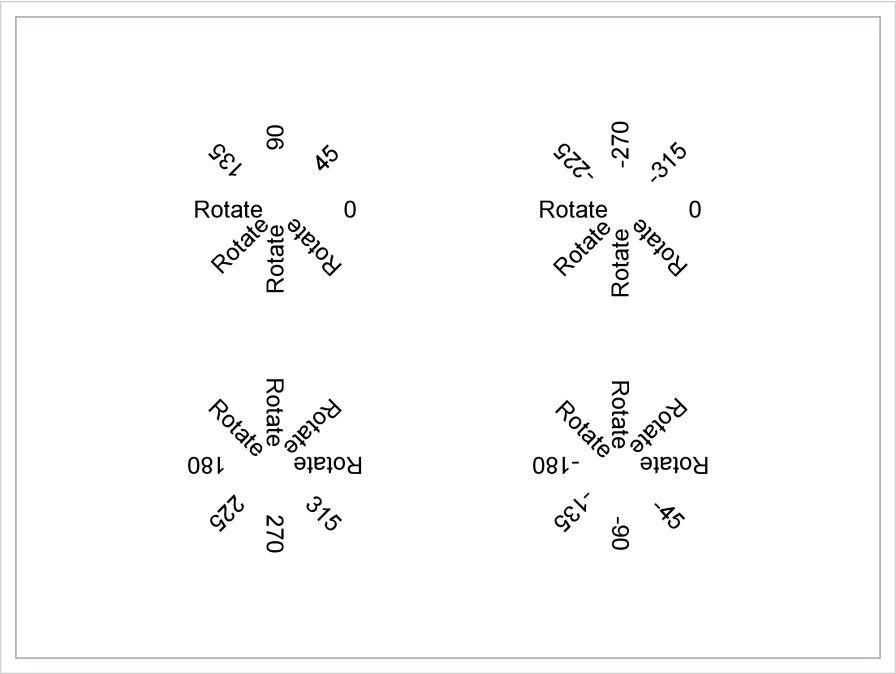
```
   do Rotate = 0 to 135 by 45;
      Label = cat('Rotate          ', put(rotate, 4.0));
      output;
   end;
   x1 = 1;
   do Rotate = 0, -225 to -315 by -45;
      Label = cat('Rotate          ', put(rotate, 4.0));
      output;
   end;
   y1 = -1; x1 = -1;
   do Rotate = 180 to 315 by 45;
      Label = cat('Rotate          ', put(rotate, 4.0));
      output;
   end;
   x1 = 1;
   do Rotate = -45 to -180 by -45;
      Label = cat('Rotate          ', put(rotate, 4.0));
      output;
   end;
run;

proc print noobs;
run;

proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none offsetmin=0.1 offsetmax=0.1;
   yaxis display=none offsetmin=0.1 offsetmax=0.1;
run;
```

The annotation data set is displayed in Figure 4.50, and the results are displayed in Figure 4.51.

Figure 4.50 Annotation Data Set

Label		DrawSpace	Function	Width	TextSize	Anchor	y1	x1	Rotate
Rotate	0	DataValue	Text	100	12	Center	1	-1	0
Rotate	45	DataValue	Text	100	12	Center	1	-1	45
Rotate	90	DataValue	Text	100	12	Center	1	-1	90
Rotate	135	DataValue	Text	100	12	Center	1	-1	135
Rotate	0	DataValue	Text	100	12	Center	1	1	0
Rotate	-225	DataValue	Text	100	12	Center	1	1	-225
Rotate	-270	DataValue	Text	100	12	Center	1	1	-270
Rotate	-315	DataValue	Text	100	12	Center	1	1	-315
Rotate	180	DataValue	Text	100	12	Center	-1	-1	180
Rotate	225	DataValue	Text	100	12	Center	-1	-1	225
Rotate	270	DataValue	Text	100	12	Center	-1	-1	270
Rotate	315	DataValue	Text	100	12	Center	-1	-1	315
Rotate	-45	DataValue	Text	100	12	Center	-1	1	-45
Rotate	-90	DataValue	Text	100	12	Center	-1	1	-90
Rotate	-135	DataValue	Text	100	12	Center	-1	1	-135
Rotate	-180	DataValue	Text	100	12	Center	-1	1	-180

Figure 4.51 Rotation from a Center Anchor



Continuing Text

Double Click for Example Code

You can start a text string by using one set of text annotation variables and continue by using a different set of variables. This example illustrates:

```
data x;
   input x y @@;
   datalines;
2 2 -2 -2
;

data anno;
   length Label $ 20 TextColor $ 8;
   retain DrawSpace 'DataValue' Function 'Text    '
          TextSize 25 Width 80 x1 -1.5 y1 0 Anchor 'Left';
   input label $ @@;
   if _n_ gt 1 then label = ' ' || label;
   textcolor = scan('red orange green blue magenta', mod(_n_, 5) + 1);
   output;
   function = 'TextCont';
   textsize + -1;
   x1 = .;
   y1 = .;
   datalines;
Each word gets smaller and changes colors, and sooner or later the text
wraps onto multiple lines.
;
```

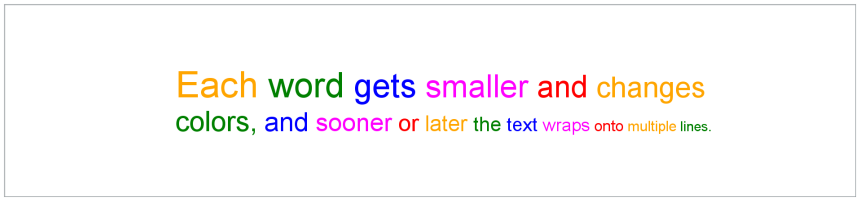
```
proc print noobs;
run;

ods graphics on / width=640px height=150px;
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none offsetmin=0.1 offsetmax=0.1;
   yaxis display=none offsetmin=0.1 offsetmax=0.1;
run;
```

The annotation data set is displayed in Figure 4.52, and the graph is displayed in Figure 4.53.

Figure 4.52 Annotation Data Set

Label	TextColor	DrawSpace	Function	TextSize	Width	x1	y1	Anchor
Each	orange	DataValue	Text	25	80	-1.5	0	Left
word	green	DataValue	TextCont	24	80	.	.	Left
gets	blue	DataValue	TextCont	23	80	.	.	Left
smaller	magenta	DataValue	TextCont	22	80	.	.	Left
and	red	DataValue	TextCont	21	80	.	.	Left
changes	orange	DataValue	TextCont	20	80	.	.	Left
colors,	green	DataValue	TextCont	19	80	.	.	Left
and	blue	DataValue	TextCont	18	80	.	.	Left
sooner	magenta	DataValue	TextCont	17	80	.	.	Left
or	red	DataValue	TextCont	16	80	.	.	Left
later	orange	DataValue	TextCont	15	80	.	.	Left
the	green	DataValue	TextCont	14	80	.	.	Left
text	blue	DataValue	TextCont	13	80	.	.	Left
wraps	magenta	DataValue	TextCont	12	80	.	.	Left
onto	red	DataValue	TextCont	11	80	.	.	Left
multiple	orange	DataValue	TextCont	10	80	.	.	Left
lines.	green	DataValue	TextCont	9	80	.	.	Left

Figure 4.53 Continuing Text



As the graph states, each word gets smaller and changes colors, and eventually the text wraps onto multiple lines. All values of the Label variable except the first begin with a blank. Otherwise, there would be no separation between the words, because trailing blanks are ignored. The value of the Function variable is `'Text'` for the first observation and `'TextCont'` for all subsequent observations. The DATA step sets the variables x1 and y1 to missing after the first observation is written. This is not necessary, but it emphasizes that these variables are ignored when Function = `'TextCont'`. Wrapping occurs when the width of the text exceeds the specified 80% of the width.

This next example changes the TextStyleElement, TextStyle, TextWeight, and TextFont variables:

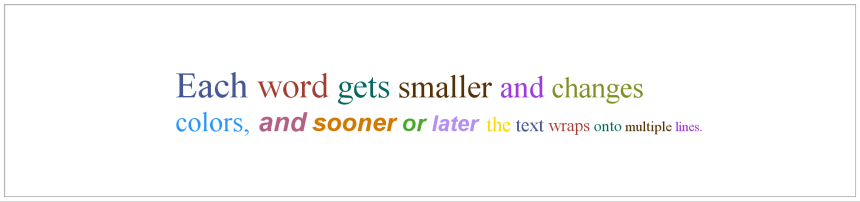
```
data anno;
   length Label $ 20 TextStyleElement TextStyle TextWeight $ 12
          TextFont $ 24;
   retain DrawSpace 'DataValue' Function 'Text    '
          TextSize 25 Width 80 x1 -1.5 y1 0 Anchor 'Left';
   input label $ @@;
   if _n_ gt 1 then label = ' ' || label;
   textstyleelement = cats('GraphData', mod(_n_ - 1, 12) + 1);
   textstyle  = ifc(8 le _n_ le 11, 'Italic', 'Normal');
   textweight = ifc(8 le _n_ le 11, 'Bold'  , 'Normal');
   textfont   = ifc(8 le _n_ le 11, 'Arial' , 'Times New Roman');
   output;
   function = 'TextCont';
   textsize + -1;
   x1 = .;
   y1 = .;
   datalines;
Each word gets smaller and changes colors, and sooner or later the text
wraps onto multiple lines.
;

proc print noobs;
run;

ods graphics on / width=640px height=150px;
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none offsetmin=0.1 offsetmax=0.1;
   yaxis display=none offsetmin=0.1 offsetmax=0.1;
run;
```

The annotation data set is displayed in Figure 4.54, and the graph is displayed in Figure 4.55.

Figure 4.54 Annotation Data Set

Label	TextStyleElement	TextStyle	TextWeight	TextFont	DrawSpace	Function	TextSize	Width	x1	y1	Anchor
Each	GraphData1	Normal	Normal	Times New Roman	DataValue	Text	25	80	-1.5	0	Left
word	GraphData2	Normal	Normal	Times New Roman	DataValue	TextCont	24	80	.	.	Left
gets	GraphData3	Normal	Normal	Times New Roman	DataValue	TextCont	23	80	.	.	Left
smaller	GraphData4	Normal	Normal	Times New Roman	DataValue	TextCont	22	80	.	.	Left
and	GraphData5	Normal	Normal	Times New Roman	DataValue	TextCont	21	80	.	.	Left
changes	GraphData6	Normal	Normal	Times New Roman	DataValue	TextCont	20	80	.	.	Left
colors,	GraphData7	Normal	Normal	Times New Roman	DataValue	TextCont	19	80	.	.	Left
and	GraphData8	Italic	Bold	Arial	DataValue	TextCont	18	80	.	.	Left
sooner	GraphData9	Italic	Bold	Arial	DataValue	TextCont	17	80	.	.	Left
or	GraphData10	Italic	Bold	Arial	DataValue	TextCont	16	80	.	.	Left
later	GraphData11	Italic	Bold	Arial	DataValue	TextCont	15	80	.	.	Left
the	GraphData12	Normal	Normal	Times New Roman	DataValue	TextCont	14	80	.	.	Left
text	GraphData1	Normal	Normal	Times New Roman	DataValue	TextCont	13	80	.	.	Left
wraps	GraphData2	Normal	Normal	Times New Roman	DataValue	TextCont	12	80	.	.	Left
onto	GraphData3	Normal	Normal	Times New Roman	DataValue	TextCont	11	80	.	.	Left
multiple	GraphData4	Normal	Normal	Times New Roman	DataValue	TextCont	10	80	.	.	Left
lines.	GraphData5	Normal	Normal	Times New Roman	DataValue	TextCont	9	80	.	.	Left

Figure 4.55 Text Style Element, Style, Weight, and Font



For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

Shape and Scale of Arrowheads

Double Click for Example Code

The following steps illustrate how to use the Shape and Scale annotation variables, which control the shape and size of the arrowheads:

```
data x;
   input x y @@;
   datalines;
-1 1 1 8
;
```

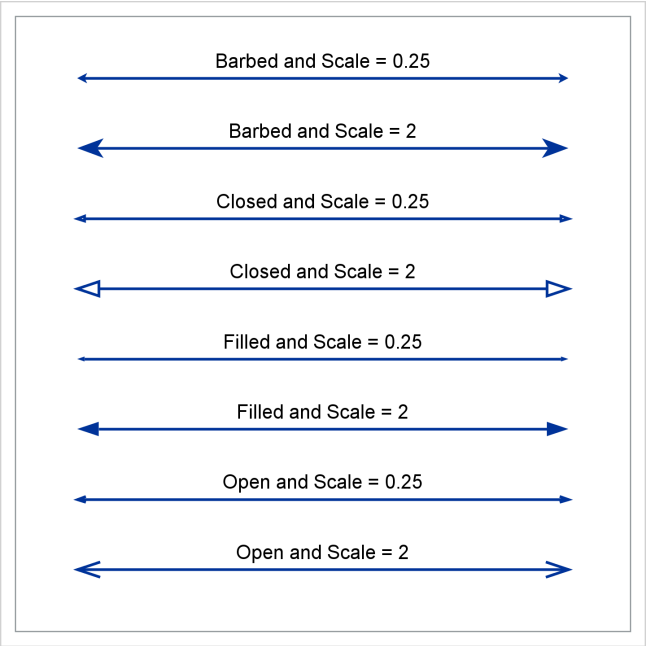
```
data anno;
   retain Function 'Arrow' DrawSpace 'DataValue' Direction 'Both' x1 -1 x2 1;
   do y1 = 1 to 8;
      y2    = y1;
      Shape = scan('Barbed Closed Filled Open', floor((y1 - 1) / 2) + 1);
      Scale = 0.25 * (mod((y1 - 1), 2) + 1) ** 3;
      output;
   end;
run;

data anno;
   set anno;
   output;
   function = 'Text'; x1 = 0; y1 + -0.25; Width = 50;
   Label = catx(' ', shape, 'and Scale =', scale);
   output;
run;

proc print noobs;
run;

ods graphics on / width=4.8in height=4.8in;
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis  offsetmin=0.1 offsetmax=0.1 display=none;
   yaxis  offsetmin=0.1 offsetmax=0.1 display=none reverse;
run;
```

The annotation data set is displayed in Figure 4.56, and the results are displayed in Figure 4.57.

Figure 4.56 Annotation Data Set

Function	DrawSpace	Direction	x1	x2	y1	y2	Shape	Scale	Width	Label
Arrow	DataValue	Both	-1	1	1.00	1	Barbed	0.25	.	
Text	DataValue	Both	0	1	0.75	1	Barbed	0.25	50	Barbed and Scale = 0.25
Arrow	DataValue	Both	-1	1	2.00	2	Barbed	2.00	.	
Text	DataValue	Both	0	1	1.75	2	Barbed	2.00	50	Barbed and Scale = 2
Arrow	DataValue	Both	-1	1	3.00	3	Closed	0.25	.	
Text	DataValue	Both	0	1	2.75	3	Closed	0.25	50	Closed and Scale = 0.25
Arrow	DataValue	Both	-1	1	4.00	4	Closed	2.00	.	
Text	DataValue	Both	0	1	3.75	4	Closed	2.00	50	Closed and Scale = 2
Arrow	DataValue	Both	-1	1	5.00	5	Filled	0.25	.	
Text	DataValue	Both	0	1	4.75	5	Filled	0.25	50	Filled and Scale = 0.25
Arrow	DataValue	Both	-1	1	6.00	6	Filled	2.00	.	
Text	DataValue	Both	0	1	5.75	6	Filled	2.00	50	Filled and Scale = 2
Arrow	DataValue	Both	-1	1	7.00	7	Open	0.25	.	
Text	DataValue	Both	0	1	6.75	7	Open	0.25	50	Open and Scale = 0.25
Arrow	DataValue	Both	-1	1	8.00	8	Open	2.00	.	
Text	DataValue	Both	0	1	7.75	8	Open	2.00	50	Open and Scale = 2

Figure 4.57 Arrowhead Shapes and Sizes



Text Justification and Anchoring

Double Click for Example Code

The following steps illustrate how to use the Justify and Anchor annotation variables, which control the position of text:

```
data x;
   input x y @@;
   datalines;
0 0 2 9
;

data anno(drop=a);
retain Function 'Text' DrawSpace 'DataValue' Width 15
       Justify 'Left   ' Border 'True';
   do a = 1 to 9;
      Anchor = scan('Top-Left Top Top-Right Right Bottom-Right Bottom
                     Bottom-Left Left Center', a, ' ');
      Label  = tranwrd(anchor, '-', ' ');
      anchor = compress(anchor, '-');
      if a gt 3 then width = 20;
      y1 + 1; x1 = 0; justify = 'Left  '; output;
              x1 = 1; justify = 'Center'; output;
              x1 = 2; justify = 'Right '; output;
   end;
run;
```

```
proc print noobs;
run;

ods graphics on / width=4.8in height=6in;
proc sgplot data=anno sganno=anno;
   title justify=left  'Justify=Left' justify=center 'Justify=Center'
         justify=right 'Justify=Right';
   scatter y=y1 x=x1 / markerattrs=(symbol=circlefilled);
   xaxis display=none offsetmin=0.2  offsetmax=0.2;
   yaxis display=none offsetmin=0.03 offsetmax=0.03 reverse;
run;

title;
```

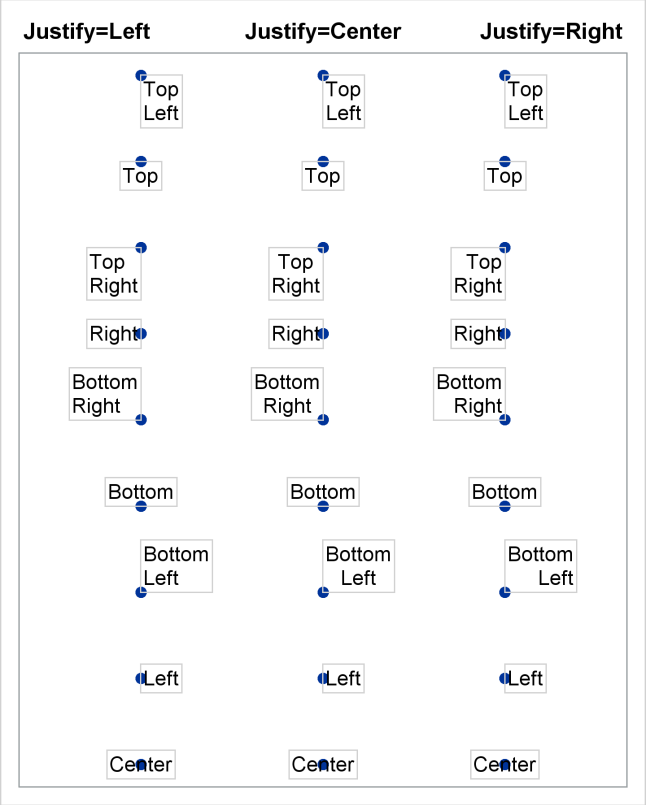
The annotation data set is displayed in Figure 4.58, and the results are displayed in Figure 4.59.

Figure 4.58 Annotation Data Set

Function	DrawSpace	Width	Justify	Border	Anchor	Label	y1	x1
Text	DataValue	15	Left	True	TopLeft	Top Left	1	0
Text	DataValue	15	Center	True	TopLeft	Top Left	1	1
Text	DataValue	15	Right	True	TopLeft	Top Left	1	2
Text	DataValue	15	Left	True	Top	Top	2	0
Text	DataValue	15	Center	True	Top	Top	2	1
Text	DataValue	15	Right	True	Top	Top	2	2
Text	DataValue	15	Left	True	TopRight	Top Right	3	0
Text	DataValue	15	Center	True	TopRight	Top Right	3	1
Text	DataValue	15	Right	True	TopRight	Top Right	3	2
Text	DataValue	20	Left	True	Right	Right	4	0
Text	DataValue	20	Center	True	Right	Right	4	1
Text	DataValue	20	Right	True	Right	Right	4	2
Text	DataValue	20	Left	True	BottomRight	Bottom Right	5	0
Text	DataValue	20	Center	True	BottomRight	Bottom Right	5	1
Text	DataValue	20	Right	True	BottomRight	Bottom Right	5	2
Text	DataValue	20	Left	True	Bottom	Bottom	6	0
Text	DataValue	20	Center	True	Bottom	Bottom	6	1
Text	DataValue	20	Right	True	Bottom	Bottom	6	2
Text	DataValue	20	Left	True	BottomLeft	Bottom Left	7	0
Text	DataValue	20	Center	True	BottomLeft	Bottom Left	7	1
Text	DataValue	20	Right	True	BottomLeft	Bottom Left	7	2
Text	DataValue	20	Left	True	Left	Left	8	0
Text	DataValue	20	Center	True	Left	Left	8	1
Text	DataValue	20	Right	True	Left	Left	8	2
Text	DataValue	20	Left	True	Center	Center	9	0
Text	DataValue	20	Center	True	Center	Center	9	1
Text	DataValue	20	Right	True	Center	Center	9	2

The Border='True' variable displays a frame around each label. This frame is helpful for understanding how anchors and justification work. The values of the Label variable, which are displayed in the body of the graph, describe the anchors. The anchor is the place where the frame intersects the x1 and y1 coordinates. The Justify variable shows how the text is justified within each frame.

Figure 4.59 Combinations of Anchors and Justifications



In the third row from the bottom, the anchor is `'BottomLeft'` and the label is `'Bottom Left'`. Notice that the values of the Anchor variable contain no embedded blanks, whereas the labels have embedded blanks, which enables them to split onto two lines and illustrate justification. Justify=`'Left'` aligns the `'B'` and the `'L'`, Justify=`'Center'` aligns the centers of `'Bottom'` and the `'Left'`, and Justify=`'Right'` aligns the `'m'` and the `'t'`.

Selecting the X, X2, Y, and Y2 Axes

Double Click for Example Code

The following steps illustrate how to use the xAxis and yAxis annotation variables, which control which axes are used:

```
data x;
   input x1 y1 x2 y2 @@;
   datalines;
0 0 0 0 10 10 100 100
;

data anno;
   retain Function 'Arrow' DrawSpace 'DataValue' Width 100
          Scale 1e-6 Transparency 0.6 x1 y1 10 FillColor 'White';
```

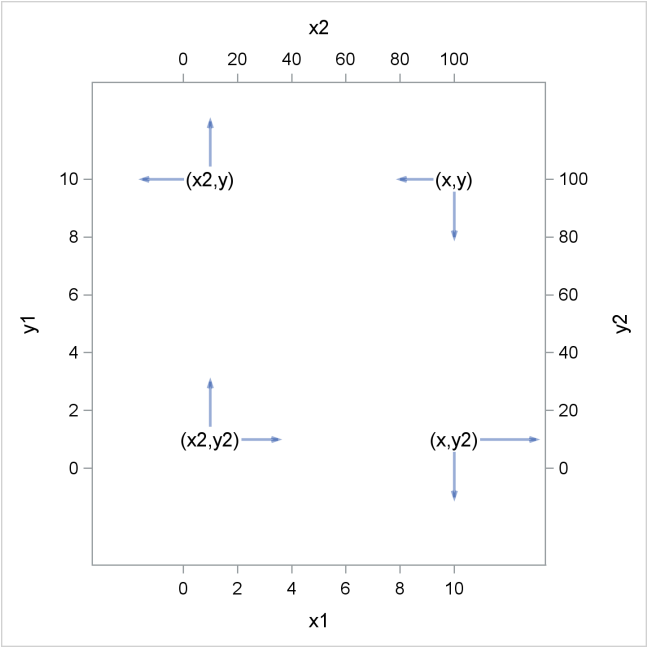
```
   xAxis = 'x '; yAxis = 'y '; x2 =   8; y2 =  10; output;
                               x2 =  10; y2 =   8; output;

   xAxis = 'x '; yAxis = 'y2'; x2 =  13; y2 =  10; output;
                               x2 =  10; y2 = -10; output;

   xAxis = 'x2'; yAxis = 'y '; x2 = -15; y2 =  10; output;
                               x2 =  10; y2 =  12; output;

   xAxis = 'x2'; yAxis = 'y2'; x2 =  35; y2 =  10; output;
                               x2 =  10; y2 =  30; output;

   function = 'Text'; Transparency = 0;
   xAxis = 'x '; yAxis = 'y '; Label = cats('(',xaxis,',',yaxis,')'); output;
   xAxis = 'x '; yAxis = 'y2'; label = cats('(',xaxis,',',yaxis,')'); output;
   xAxis = 'x2'; yAxis = 'y '; label = cats('(',xaxis,',',yaxis,')'); output;
   xAxis = 'x2'; yAxis = 'y2'; label = cats('(',xaxis,',',yaxis,')'); output;
run;

proc print noobs;
run;

ods graphics on / width=4.8in height=4.8in;
proc sgplot data=x sganno=anno noautolegend;
   scatter  y=y1 x=x1 / markerattrs=(size=0);
   scatter  y=y2 x=x2 / markerattrs=(size=0) y2axis x2axis;
   xaxis  min=0 max=10  offsetmin=0.2 offsetmax=0.2;
   yaxis  min=0 max=10  offsetmin=0.2 offsetmax=0.2;
   x2axis min=0 max=100 offsetmin=0.2 offsetmax=0.2;
   y2axis min=0 max=100 offsetmin=0.2 offsetmax=0.2;
run;
```

The annotation data set is displayed in Figure 4.60, and the results are displayed in Figure 4.61. The FillColor='White' variable eliminates collisions between the arrows and the labels. The data set has one set of coordinates: x1 = 10 and y1 = 10. There are four points on the graph because the four axis combinations (X and Y, X and Y2, X2 and Y, and X2 and Y2) are specified. The two annotation variables, xAxis and yAxis, specify which axes are used. The labels and arrows show which axis combination is used for each point.

Figure 4.60 Annotation Data Set

Function	DrawSpace	Width	Scale	Transparency	x1	y1	FillColor	xAxis	yAxis	x2	y2	Label
Arrow	DataValue	100	.000001	0.6	10	10	White	x	y	8	10	
Arrow	DataValue	100	.000001	0.6	10	10	White	x	y	10	8	
Arrow	DataValue	100	.000001	0.6	10	10	White	x	y2	13	10	
Arrow	DataValue	100	.000001	0.6	10	10	White	x	y2	10	-10	
Arrow	DataValue	100	.000001	0.6	10	10	White	x2	y	-15	10	
Arrow	DataValue	100	.000001	0.6	10	10	White	x2	y	10	12	
Arrow	DataValue	100	.000001	0.6	10	10	White	x2	y2	35	10	
Arrow	DataValue	100	.000001	0.6	10	10	White	x2	y2	10	30	
Text	DataValue	100	.000001	0.0	10	10	White	x	y	10	30	(x,y)
Text	DataValue	100	.000001	0.0	10	10	White	x	y2	10	30	(x,y2)
Text	DataValue	100	.000001	0.0	10	10	White	x2	y	10	30	(x2,y)
Text	DataValue	100	.000001	0.0	10	10	White	x2	y2	10	30	(x2,y2)

Figure 4.61 The xAxis and yAxis Annotation Variables



Scaling Images

Double Click for Example Code

This example places images in the back of a graph to provide a graph legend. The following steps illustrate how to use the ImageScale annotation variable to control the size of images:

```
data anno;
   retain Function 'Image' DrawSpace 'WallPercent' Width 50 Height 100
          HeightUnit WidthUnit 'Percent' Layer 'Back' Transparency 0.8
         ImageScale 'Fit       ' y1 50;
   x1 = 25; Image = 'images\male.png  '; output;
   x1 = 75; Image = 'images\female.png'; output;
run;

proc print noobs;
   title;
run;

proc sgplot data=sashelp.class sganno=anno nowall noautolegend;
   title 'Height by Sex';
   vbox height / group=sex lineattrs=(thickness=0.05in) boxwidth=0.2
                 whiskerattrs=(thickness=0.05in) dataskin=matte;
run;

data anno;
   set anno;
   ImageScale = 'Tile';
run;
```

```
proc print noobs;
   title;
run;

proc sgplot data=sashelp.class sganno=anno nowall noautolegend;
   title 'Height by Sex';
   vbox height / group=sex lineattrs=(thickness=0.05in) boxwidth=0.2
                 whiskerattrs=(thickness=0.05in) dataskin=gloss;
run;

data anno;
   set anno;
   ImageScale = 'FitHeight'; Height = 80; Width = .;
   x1 = ifc(_n_ = 1, 15, 85);
run;

proc print noobs;
   title;
run;

proc sgplot data=sashelp.class sganno=anno nowall noautolegend;
   title 'Height by Sex';
   vbox height / group=sex lineattrs=(thickness=0.05in) boxwidth=0.2
                 whiskerattrs=(thickness=0.05in) dataskin=crisp;
run;

data anno;
   set anno;
   ImageScale = 'FitWidth'; Height = .; Width = 20;
run;

proc print noobs;
   title;
run;

proc sgplot data=sashelp.class sganno=anno nowall noautolegend;
   title 'Height by Sex';
   vbox height / group=sex lineattrs=(thickness=0.05in) boxwidth=0.2
                 whiskerattrs=(thickness=0.05in) dataskin=sheen;
run;

title;
```

The results are displayed in Figure 4.62 through Figure 4.69.

Figure 4.62 ImageScale='Fit'Annotation Data Set

Function	DrawSpace	Width	Height	HeightUnit	WidthUnit	Layer	Transparency	ImageScale	y1	x1	Image
Image	WallPercent	50	100	Percent	Percent	Back	0.8	Fit	50	25	images\male.png
Image	WallPercent	50	100	Percent	Percent	Back	0.8	Fit	50	75	images\female.png

ImageScale='Fit', which is displayed in Figure 4.63, shows that the images are expanded horizontally and vertically to fill the available space.

Figure 4.63 ImageScale='Fit'

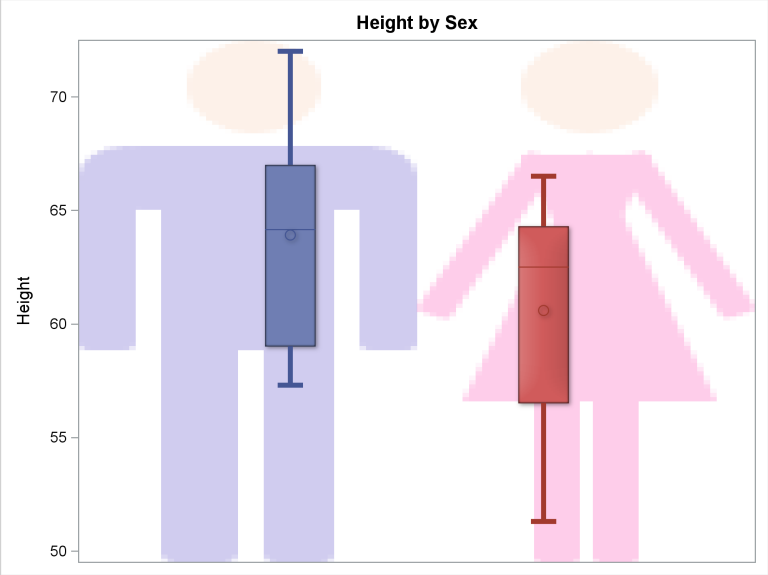


Figure 4.64 ImageScale='Tile' Annotation Data Set

Function	DrawSpace	Width	Height	HeightUnit	WidthUnit	Layer	Transparency	ImageScale	y1	x1	Image
Image	WallPercent	50	100	Percent	Percent	Back	0.8	Tile	50	25	images\male.png
Image	WallPercent	50	100	Percent	Percent	Back	0.8	Tile	50	75	images\female.png

ImageScale='Tile', which is displayed in Figure 4.65, shows images that are repeated or "tiled."

Figure 4.65 ImageScale='Tile'

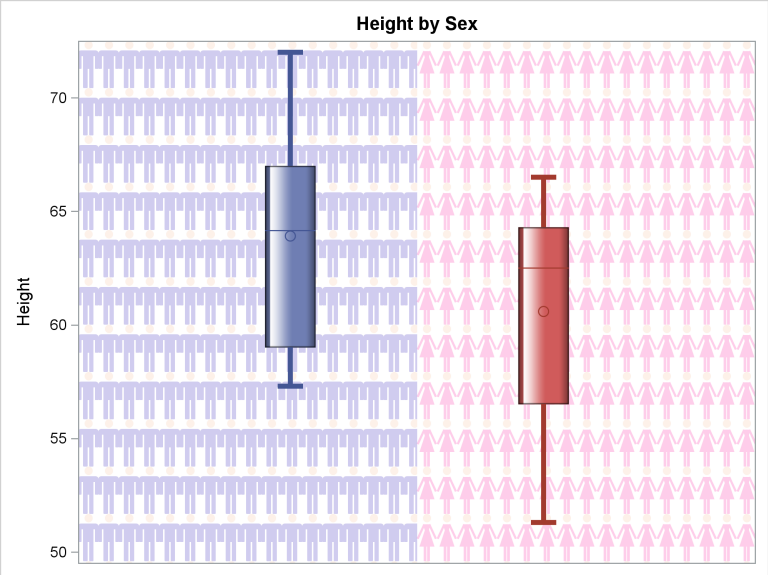


Figure 4.66 ImageScale='FitHeight'Annotation Data Set

Function	DrawSpace	Width	Height	HeightUnit	WidthUnit	Layer	Transparency	ImageScale	y1	x1	Image
Image	WallPercent	.	80	Percent	Percent	Back	0.8	FitHeight	50	15	images\male.png
Image	WallPercent	.	80	Percent	Percent	Back	0.8	FitHeight	50	85	images\female.png

ImageScale='FitHeight', which is displayed in Figure 4.67, shows images that are scaled vertically.

Figure 4.67 ImageScale='FitHeight'

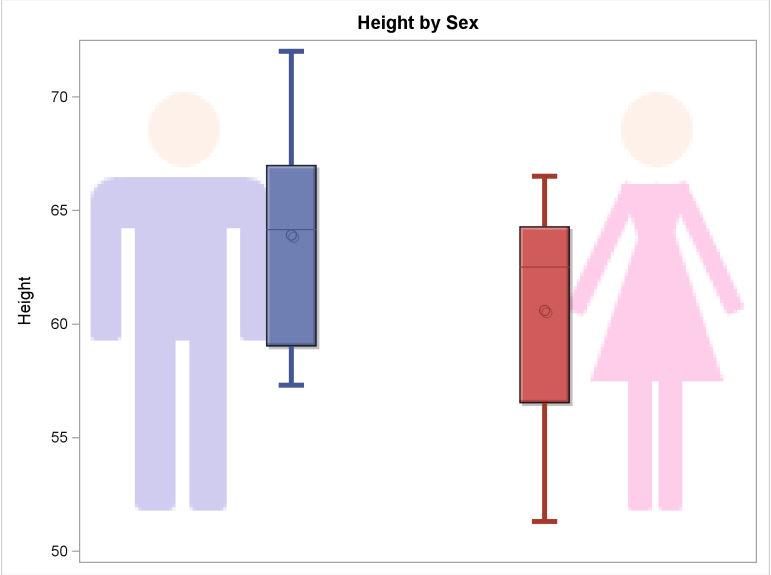
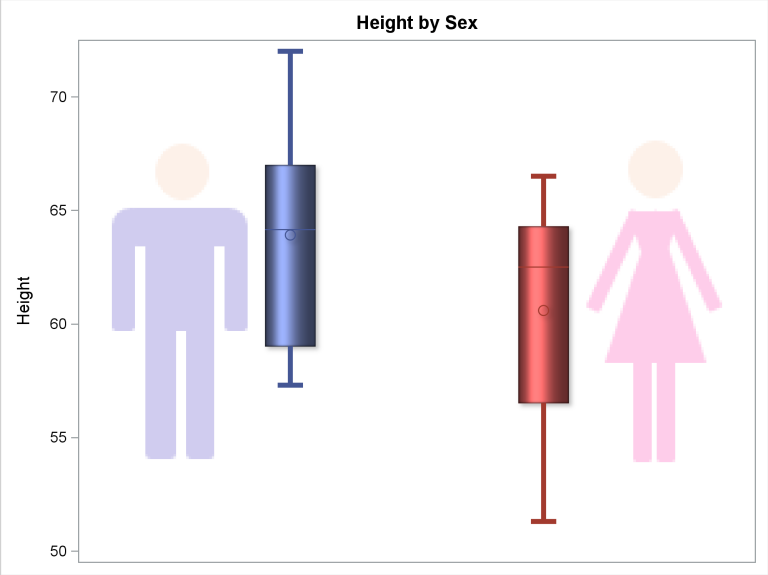


Figure 4.68 ImageScale='FitWidth'Annotation Data Set

Function	DrawSpace	Width	Height	HeightUnit	WidthUnit	Layer	Transparency	ImageScale	y1	x1	Image
Image	WallPercent	20	.	Percent	Percent	Back	0.8	FitWidth	50	15	images\male.png
Image	WallPercent	20	.	Percent	Percent	Back	0.8	FitWidth	50	85	images\female.png

ImageScale='FitWidth', which is displayed in Figure 4.68, shows images that are scaled horizontally. This example also show several skins, which provide a three-dimensional appearance to the bars.

Figure 4.69 ImageScale='FitWidth'



Adding Links to Graphs

Double Click for Example Code

This example creates a graph that has clickable links to web pages. The following steps illustrate how to use the URL annotation variable to add a link to a graph:

```
data x;
   input x y @@;
   datalines;
0 0 1 5
;

%let u = http://support.sas.com/documentation/cdl/en/statug;
%let u = &u/67523/HTML/default/viewer.htm#statug_;
data anno;
   retain DrawSpace 'DataValue' Function 'Text' Width 100 x1 0.5;
   y1 + 1; Label = 'Using the Output Delivery System          ';
           URL   = "&u.ods_toc.htm     ";                          output;
   y1 + 1; Label = 'Statistical Graphics Using ODS            ';
           URL   = "&u.odsgraph_toc.htm";                          output;
   y1 + 1; Label = 'ODS Graphics Template Modification        ';
           URL   = "&u.templt_toc.htm  ";                          output;
   y1 + 1; Label = 'Customizing the Kaplan-Meier Survival Plot';
           URL   = "&u.kaplan_toc.htm  ";                          output;
run;

proc print noobs;
run;
```

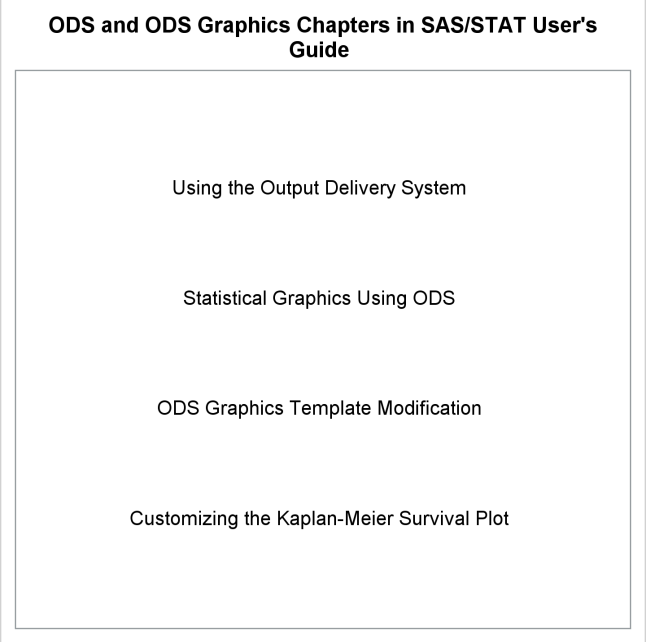
```
ods graphics on / width=4.8in height=4.8in imagemap=on;
proc sgplot data=x sganno=anno;
   title 'ODS and ODS Graphics Chapters in SAS/STAT User''s Guide';
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none reverse;
run;
```

The results are displayed in Figure 4.70 and Figure 4.71.

Figure 4.70 The Annotation Data Set

DrawSpace	Function	Width	x1	y1	Label	URL
DataValue	Text	100	0.5	1	Using the Output Delivery System	http://support.sas.com/documentation/cdl/en/statug/67523/HTML/default/viewer.htm#statug_ods_toc.htm
DataValue	Text	100	0.5	2	Statistical Graphics Using ODS	http://support.sas.com/documentation/cdl/en/statug/67523/HTML/default/viewer.htm#statug_odsgraph_toc.htm
DataValue	Text	100	0.5	3	ODS Graphics Template Modification	http://support.sas.com/documentation/cdl/en/statug/67523/HTML/default/viewer.htm#statug_templt_toc.htm
DataValue	Text	100	0.5	4	Customizing the Kaplan-Meier Survival Plot	http://support.sas.com/documentation/cdl/en/statug/67523/HTML/default/viewer.htm#statug_kaplan_toc.htm

Figure 4.71 Clickable Links in a Graph



The URL variable provides a link that you can click.[1] You must specify the IMAGEMAP=ON option in the ODS GRAPHICS statement for this to work. You do not have to use macro variables; they are used here to minimize typing and to ensure that all lines fit in a page.

[1] Although you can click on the links when you generate this graph, the linking ability is lost in the process that is used to make this documentation, so links are not enabled in this version of the graph.

SG Annotation Functions, Variables, and Their Values

Double Click for Example Code

Table 4.1 displays all the annotation functions as columns; it shows information about which annotation variables are required, which are optional, and which are ignored. When a limited number of fixed values are available, they are listed too. Coordinate variables can be numeric or character. For most functions, either x1 or xC1 and either y1 or yC1 are required. A few functions require two points. In that case, either x2 or xC2 and either y2 or yC2 also are required. Two functions, `'PolyCont'` and `'TextCont'`, provide continuation information following other function specifications (`'Polygon'` or `'PolyLine'` for `'PolyCont'` and `'Text'` for `'TextCont'`). These continuation functions use fewer annotation variables because many attributes are set by the original function. Function values are grouped so that similar functions are together. The annotation variable names are listed in alphabetical order.

The following steps create and display in Figure 4.72 a small annotation data set:

```
data anno;
   retain Function 'Text      ' DrawSpace 'GraphPercent';
   Label = 'Some Text';    x1 = 50; y1 = 50; Width = 100; output;
   label = ' ';
   function = 'Rectangle'; Height = 30;      width =  30; output;
   function = 'Oval';      height = 50;      width =  50; output;
                           height = .;       width =  .;
   function = 'PolyLine';  x1     = 20;      y1    =  80; output;
   function = 'PolyCont';  x1     = 80;                   output;
run;

proc print noobs;
run;
```

Figure 4.72 The Annotation Data Set

Function	DrawSpace	Label	x1	y1	Width	Height
Text	GraphPercent	Some Text	50	50	100	.
Rectangle	GraphPercent		50	50	30	30
Oval	GraphPercent		50	50	50	50
PolyLine	GraphPercent		20	80	.	.
PolyCont	GraphPercent		80	80	.	.

This annotation data set provides an example of five of the 10 functions, and has seven variables.

Table 4.1 SG Annotation Functions, Variables, and Their Values

	Text	**TextCont**	**Arrow**	**Line**	**Polygon**	**PolyLine**	**PolyCont**	**Oval**	**Rectangle**	**Image**
Anchor=*anchor*	O	.	.	.	.	.	.	.	O	O
Border=True | **False**	O	.	.	.	.	.	.	.	.	O
CornerRadius=*numeric*	.	.	.	.	.	.	.	.	O	.
Direction=Both | In | **Out**	.	.	O	.	.	.	.	.	.	.
DiscreteOffset=*numeric*	O	.	O	O	O	O	.	O	O	O
Display=All | Fill | **Outline**	.	.	.	.	O	.	.	O	O	.
DrawSpace=*space*	O	.	O	O	O	O	.	O	O	O
FillColor=*character*	O	.	.	.	O	.	.	O	O	.
FillStyleElement=*character*	.	.	.	.	O	.	.	O	O	.
FillTransparency=*numeric*	O	.	.	.	O	.	.	O	O	.
Height=*numeric*	.	.	.	.	.	.	.	R	R	O
HeightUnit=*unit*	.	.	.	.	.	.	.	O	O	O
Image=*character*	.	.	.	.	.	.	.	.	.	R
ImageScale=*fit*	.	.	.	.	.	.	.	.	.	O
Justify=Center | **Left** | Right	O	.	.	.	.	.	.	.	.	.
Label=*character*	R	R	.	.	.	.	.	.	.	.
Layer=Back | **Front**	O	.	O	O	O	O	.	O	O	O
LineColor=*character*	O	.	O	O	O	O	.	O	O	O
LinePattern=*pattern*	O	.	O	O	O	O	.	O	O	O
LineStyleElement=*character*	O	.	O	O	O	O	.	O	O	O
LineThickness=*numeric*	O	.	O	O	O	O	.	O	O	O
Rotate=*degrees*	O	.	.	.	.	.	.	O	O	O
Scale=*numeric*	.	.	O	.	.	.	.	.	.	.
Shape=*shape*	.	.	O	.	.	.	.	.	.	.
TextSize=*numeric*	O	O	.	.	.	.	.	.	.	.
TextColor=*character*	O	O	.	.	.	.	.	.	.	.
TextFont=*character*	O	O	.	.	.	.	.	.	.	.
TextStyle=Italic | **Normal**	O	O	.	.	.	.	.	.	.	.
TextStyleElement=*character*	O	O	.	.	.	.	.	.	.	.
TextWeight=Bold | **Normal**	O	O	.	.	.	.	.	.	.	.
Transparency=*numeric*	O	.	O	O	O	O	.	O	O	O
URL=*character*	O	.	O	O	O	O	.	O	O	O
Width=*numeric*	O	.	.	.	.	.	.	R	R	O
WidthUnit=*unit*	O	.	.	.	.	.	.	O	O	O
x1=*numeric*	O	.	1	1	1	1	1	1	1	O
x1Space=*space*	O	.	O	O	O	O	.	O	O	O
x2=*numeric*	.	.	2	2	.	.	.	.	.	.
x2Space=*space*	.	.	O	O	.	.	.	.	.	.
xAxis=**X** | X2	O	.	O	O	O	O	.	O	O	O
xC1=*character*	O	.	1	1	1	1	1	1	1	O
xC2=*character*	.	.	2	2	.	.	.	.	.	.
y1=*numeric*	O	.	1	1	1	1	1	1	1	O
y1Space=*space*	O	.	O	O	O	O	.	O	O	O
y2=*numeric*	.	.	2	2	.	.	.	.	.	.
y2Space=*space*	.	.	O	O	.	.	.	.	.	.
yAxis=**Y** | Y2	O	.	O	O	O	O	.	O	O	O
yC1=*character*	O	.	1	1	1	1	1	1	1	O
yC2=*character*	.	.	2		.	.	.	.	.	.

The column headers are functions, and the row labels are variables and values. All names and values are case-insensitive. Default values are displayed in bold. Many of the values in the table provide links to examples. Examples exist for every function and variable, but not for every combination. Most variables behave in obvious ways for other functions.

R	Required.
1	Either x1 or xC1 is required, and either y1 or yC1 is required.
2	Either x2 or xC2 is required, and either y2 or yC2 is required.
O	Optional.
anchor	TopLeft | Top | TopRight | Right | BottomRight | Bottom | BottomLeft | Left | **Center**
fit	Fit | FitHeight | FitWidth | Tile
shape	Barbed | Closed | Filled | **Open**
space	DataPercent | DataPixel | DataValue | **GraphPercent** | GraphPixel | LayoutPercent | LayoutPixel | WallPercent | WallPixel The drawing space cannot be changed by PolyCont and TextCont.
unit	Data | **Percent** | Pixel

You can use annotation to construct a table of line patterns as follows:

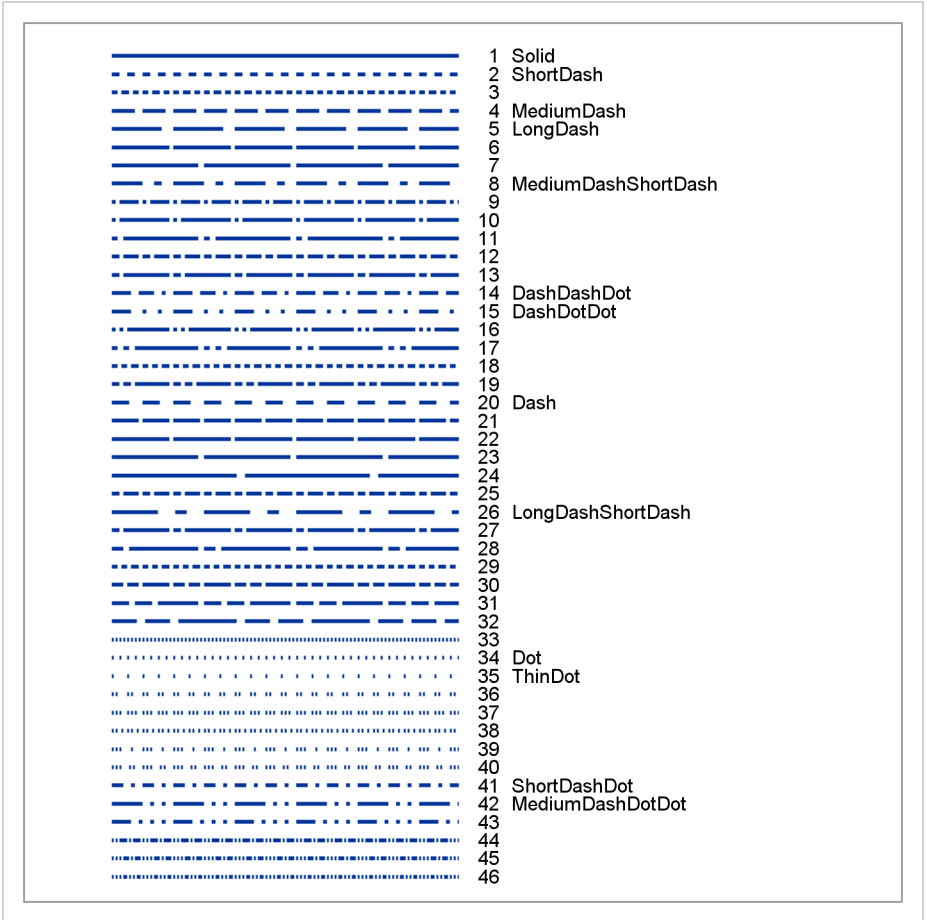
```
data x;
   input x y @@;
   datalines;
2 2 -2 -2
;

data anno;
   length Function $ 12 Label $ 20;
   retain DrawSpace 'DataPercent' TextSize 7 Width 50;
   do LinePattern = 1 to 46;
      Function = 'Line';
      y1 = 99.5 - 2.1 * linepattern;
      y2 = y1; x1 = 10; x2 = 50;
      output;
      Function = 'Text';
      Label = put(linepattern, 2.);
      Anchor = 'Right';
      x1 = 55;
      output;
      Label = ' ';
      x1 = 56;
      Anchor = 'Left';
      select (LinePattern);
         when ( 1) Label = 'Solid';
         when ( 2) Label = 'ShortDash';
         when ( 4) Label = 'MediumDash';
         when ( 5) Label = 'LongDash';
         when ( 8) Label = 'MediumDashShortDash';
         when (14) Label = 'DashDashDot';
         when (15) Label = 'DashDotDot';
         when (20) Label = 'Dash';
         when (26) Label = 'LongDashShortDash';
         when (34) Label = 'Dot';
         when (35) Label = 'ThinDot';
         when (41) Label = 'ShortDashDot';
         when (42) Label = 'MediumDashDotDot';
         otherwise;
      end;
      if label ne ' ' then output;
   end;
run;

proc print noobs data=anno(obs=15);
run;

ods graphics on / width=4.8in height=4.8in;
proc sgplot data=x sganno=anno;
   scatter y=y x=x / markerattrs=(size=0);
   xaxis display=none;
   yaxis display=none;
run;
```

Part of the annotation data set is displayed in Figure 4.73. The line patterns are displayed in Figure 4.74.

Figure 4.73 First 15 Observations of the Annotation Data Set

Function	Label	DrawSpace	TextSize	Width	LinePattern	y1	y2	x1	x2	Anchor
Line		DataPercent	7	50	1	97.4	97.4	10	50	
Text	1	DataPercent	7	50	1	97.4	97.4	55	50	Right
Text	Solid	DataPercent	7	50	1	97.4	97.4	56	50	Left
Line	Solid	DataPercent	7	50	2	95.3	95.3	10	50	Left
Text	2	DataPercent	7	50	2	95.3	95.3	55	50	Right
Text	ShortDash	DataPercent	7	50	2	95.3	95.3	56	50	Left
Line	ShortDash	DataPercent	7	50	3	93.2	93.2	10	50	Left
Text	3	DataPercent	7	50	3	93.2	93.2	55	50	Right
Line		DataPercent	7	50	4	91.1	91.1	10	50	Left
Text	4	DataPercent	7	50	4	91.1	91.1	55	50	Right
Text	MediumDash	DataPercent	7	50	4	91.1	91.1	56	50	Left
Line	MediumDash	DataPercent	7	50	5	89.0	89.0	10	50	Left
Text	5	DataPercent	7	50	5	89.0	89.0	55	50	Right
Text	LongDash	DataPercent	7	50	5	89.0	89.0	56	50	Left
Line	LongDash	DataPercent	7	50	6	86.9	86.9	10	50	Left

Figure 4.74 Line Patterns

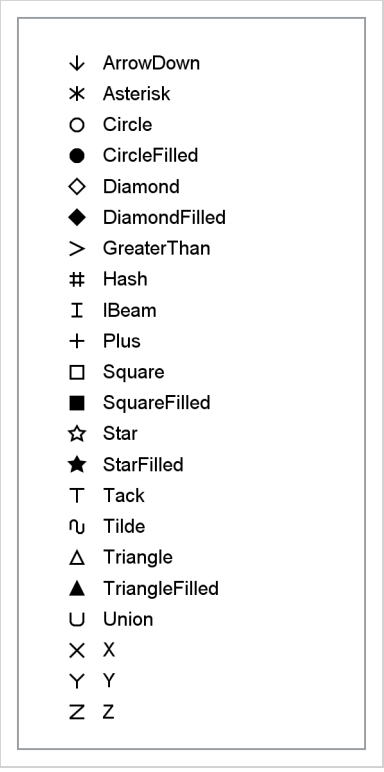


You can use PROC SGPLOT to construct a table of markers as follows:

```
data x;
   x = 5;
   length marker $ 20;
   input marker @@;
   marker = '    ' || marker;
   y = -_n_;
   datalines;
ArrowDown Asterisk Circle CircleFilled Diamond DiamondFilled
GreaterThan Hash IBeam Plus Square SquareFilled Star StarFilled
Tack Tilde Triangle TriangleFilled Union X Y Z
;

ods graphics on / width=2in height=4in attrpriority=none;
proc sgplot data=x noautolegend nocycleattrs;
   styleattrs datacontrastcolors=(black)
      datasymbols=(ArrowDown Asterisk Circle CircleFilled Diamond
                   DiamondFilled GreaterThan Hash IBeam Plus
                   Square SquareFilled Star StarFilled Tack Tilde
                   Triangle TriangleFilled Union X Y Z);
   scatter y=y x=x / group=y datalabel=marker datalabelpos=right;
   xaxis display=none offsetmin=0.2;
   yaxis display=none offsetmin=0.05;
run;
```

The markers are displayed in Figure 4.75.

Figure 4.75 Markers



It is easy to make mistakes when you create SG annotation data sets. Common mistakes include truncating character variables because of an insufficient variable length, not specifying all required variables, and specifying invalid values. When you create a SAS data set, the first instance of a variable defines the attributes of that variable. In particular, the length of a character variable is set based on its first usage. If you find that an annotation data instruction is not working, print the annotation data set and ensure that no values are truncated. You can display each unique value of each character variable as follows:

```
proc freq;
   tables _character_;
run;
```

The examples in this book use LENGTH statements or pad values with blanks to ensure that every variable is long enough.

You can run the following %AnnoCheck macro to perform more detailed checks on your SG annotation data set:

```
%macro annocheck(anno=anno);
options nonotes;

data _null_; /* get variable lengths and construct initialization */
   dsid = open("&anno");
   length s a $ 1000 name $ 16;
   name = 'Anchor';           link getlen;
   name = 'Border';           link getlen;
   name = 'Direction';        link getlen;
   name = 'Display';          link getlen;
   name = 'DrawSpace';        link getlen;
   name = 'FillColor';        link getlen;
   name = 'FillStyleElement'; link getlen;
   name = 'Image';            link getlen;
   name = 'ImageScale';       link getlen;
   name = 'Justify';          link getlen;
   name = 'Label';            link getlen;
   name = 'Layer';            link getlen;
   name = 'LineColor';        link getlen;
   name = 'LinePattern';      /* no checks, can be char or num */
   name = 'LineStyleElement'; link getlen;
   name = 'Rotate';           link getlen;
   name = 'Shape';            link getlen;
   name = 'TextColor';        link getlen;
   name = 'TextFont';         link getlen;
   name = 'TextStyle';        link getlen;
   name = 'TextStyleElement'; link getlen;
   name = 'TextWeight';       link getlen;
   name = 'URL';              link getlen;
   name = 'x1Space';          link getlen;
   name = 'x2Space';          link getlen;
   name = 'xAxis';            link getlen;
   name = 'xC1';              link getlen;
   name = 'xC2';              link getlen;
   name = 'y1Space';          link getlen;
   name = 'y2Space';          link getlen;
   name = 'yAxis';            link getlen;
```

```
   name = 'yC1';              link getlen;
   name = 'yC2';              link getlen;
   call symputx('len', s);    /* used to set lengths          */
   call symputx('assign', a); /* used to initialize variables */
   rc = close(dsid);
   return;
getlen:
   num = varnum(dsid, name);
   len = 1;
   if num then len = varlen(dsid, num);
   s = catx(' ', s, name, '$', len);
   a = catx(' ', a, name, '= " ";');
   return;
run;

data _null_;
   retain dsid;
   length &len;
   &assign
   x1 = .; x2 = .; y1 = .; y2 = .;
   set &anno end=eof;
   if _n_ = 1 then dsid = open("&anno");
   position = varnum(dsid, "Function");
   if not position then do;
      put 'Function variable not present.';
      stop;
      end;
   select (lowcase(function));
      when ('image') do;
         position = varnum(dsid, "Image");
         if not position then
            put "Image variable not present with Function='Image'.";
         end;
      when ('text') do;
         position = varnum(dsid, "Label");
         if not position then
            put "Label variable not present with Function='Text'.";
         end;
      when ('textcont') do;
         position = varnum(dsid, "Label");
         if not position then
            put "Label variable not present with Function='TextCont'.";
         end;
      when ('arrow', 'line') link bothpairs;
      when ('polygon', 'polyline', 'polycont', 'oval', 'rectangle')
         link onepair;
      otherwise;
   end;

   if not (lowcase(function) in (' ' 'text' 'textcont' 'arrow' 'line'
         'polygon' 'polyline' 'polycont' 'oval' 'rectangle' 'image'))
      then put 'Invalid: ' function=;
   if not (lowcase(anchor) in (' ' 'topleft' 'top' 'topright' 'right'
        'bottomright' 'bottom' 'bottomleft' 'left' 'center' ))
```

```
      then put 'Invalid: ' anchor=;
   if not (lowcase(border) in (' ' 'true' 'false'))
      then put 'Invalid: ' border=;
   if not (lowcase(direction) in (' ' 'both' 'in' 'out'))
      then put 'Invalid: ' direction=;
   if not (lowcase(display) in (' ' 'all' 'fill' 'outline'))
      then put 'Invalid: ' display=;
   if not (lowcase(justify) in (' ' 'center' 'left' 'right'))
      then put 'Invalid: ' justify=;
   if not (lowcase(layer) in (' ' 'back' 'front'))
      then put 'Invalid: ' layer=;
   if not (lowcase(textstyle) in (' ' 'normal' 'italic'))
      then put 'Invalid: ' textstyle=;
   if not (lowcase(textweight) in (' ' 'normal' 'bold'))
      then put 'Invalid: ' textweight=;
   if not (lowcase(xaxis) in (' ' 'x' 'x2')) then put 'Invalid: ' xaxis=;
   if not (lowcase(yaxis) in (' ' 'y' 'y2')) then put 'Invalid: ' yaxis=;

   %let spaces = ' ' 'datapercent' 'datapixel' 'datavalue' 'graphpercent'
        'graphpixel' 'layoutpercent' 'layoutpixel' 'wallpercent' 'wallpixel';
   if not (lowcase(drawspace) in (&spaces)) then put 'Invalid: ' drawspace=;
   if not (lowcase(x1space) in (&spaces)) then put 'Invalid: ' x1space=;
   if not (lowcase(x2space) in (&spaces)) then put 'Invalid: ' x2space=;
   if not (lowcase(y1space) in (&spaces)) then put 'Invalid: ' y1space=;
   if not (lowcase(y2space) in (&spaces)) then put 'Invalid: ' y2space=;

   if eof then rc = close(dsid);
   return;
bothpairs:
      n1 = varnum(dsid, "x1" ) ne 0;    n2 = varnum(dsid, "x2" ) ne 0;
      n3 = varnum(dsid, "y1" ) ne 0;    n4 = varnum(dsid, "y2" ) ne 0;
      c1 = varnum(dsid, "xc1") ne 0;    c2 = varnum(dsid, "xc2") ne 0;
      c3 = varnum(dsid, "yc1") ne 0;    c4 = varnum(dsid, "yc2") ne 0;
      if ((n1 + c1) ge 1) + ((n2 + c2) ge 1) +
         ((n3 + c3) ge 1) + ((n4 + c4) ge 1) ne 4 then link rpt2;
      n1 = n(x1);         n2 = n(x2);
      n3 = n(y1);         n4 = n(y2);
      c1 = xc1 ne ' ';    c2 = xc2 ne ' ';
      c3 = yc1 ne ' ';    c4 = yc2 ne ' ';
      if ((n1 + c1) ge 1) + ((n2 + c2) ge 1) +
         ((n3 + c3) ge 1) + ((n4 + c4) ge 1) ne 4 then do;
         link rpt2;
         put x1= x2= y1= y2= xc1= xc2= yc1= yc2=;
      end;
      if n1 + c1 = 0 then put 'Error: x1 and xC1 are both missing.';
      if n2 + c2 = 0 then put 'Error: x2 and xC1 are both missing.';
      if n3 + c3 = 0 then put 'Error: y1 and yC1 are both missing.';
      if n4 + c4 = 0 then put 'Error: y2 and yC1 are both missing.';
      if n1 & c1 then put 'Warning: x1 and xC1 are both nonmissing. ' x1= xc1=;
      if n2 & c2 then put 'Warning: x2 and xC1 are both nonmissing. ' x2= xc2=;
      if n3 & c3 then put 'Warning: y1 and yC1 are both nonmissing. ' y1= yc1=;
      if n4 & c4 then put 'Warning: y2 and yC1 are both nonmissing. ' y2= yc2=;
   return;
onepair:
```

```
      n1 = varnum(dsid, "x1")  ne 0;    n2 = varnum(dsid, "y1")  ne 0;
      c1 = varnum(dsid, "xc1") ne 0;    c2 = varnum(dsid, "yc1") ne 0;
      if ((n1 + c1) ge 1) + ((n2 + c2) ge 1) ne 2 then link rpt1;
      n1 = n(x1);         n2 = n(y1);
      c1 = xc1 ne ' ';    c2 = yc1 ne ' ';
      if ((n1 + c1) ge 1) + ((n2 + c2) ge 1) ne 2 then do;
         link rpt1;
         put x1= y1= xc1= yc1=;
      end;
      if n1 + c1 = 0 then put 'Error: x1 and xC1 are both missing.';
      if n2 + c2 = 0 then put 'Error: y1 and yC1 are both missing.';
      if n1 & c1 then put 'Warning: x1 and xC1 are both nonmissing. ' x1= xc1=;
      if n2 & c2 then put 'Warning: y1 and yC1 are both nonmissing. ' y1= yc1=;
   return;
rpt2:
   put / 'At least one of each pair must be present:' /
        '(x1,xc1), (x2,xc2), (y1,yc1), (y2,yc2)';
   put 'Present variables are: ' @;
   if n1 then put 'x1 '  @;    if n2 then put 'x2 '  @;
   if n3 then put 'y1 '  @;    if n4 then put 'y2 '  @;
   if c1 then put 'xC1 ' @;    if c2 then put 'xC2 ' @;
   if c3 then put 'yC1 ' @;    if c4 then put 'yC2 ' @;
   put;
   return;
rpt1:
   put / 'At least one of each pair must be present:' /
        '(x1,xc1), (x2,xc2)';
   put 'Present variables are: ' @;
   if n1 then put 'x1 '  @;    if n2 then put 'y1 '  @;
   if c1 then put 'xC1 ' @;    if c2 then put 'yC1 ' @;
   put;
   return;
run;

options notes;
%mend;
```

To illustrate how to use this macro, consider the annotation data set from the section "Annotating Multiple-Panel Graphs That Analytical Procedures Produce" on page 228:

```
data anno;
   length ID $ 3 Function $ 9 Label $ 40;
   retain x1Space y1Space x2Space y2Space 'DataPercent' Direction 'In';
   length Anchor $ 10 xC1 xC2 $ 20;
   retain Scale 1e-12 Width 100 WidthUnit 'Data' CornerRadius 0.8
          TextSize 7 TextWeight 'Bold'
          LineThickness 0.7 DiscreteOffset -0.3 LineColor 'Green';

   ID       = 'LO1';            Function  = 'Text';
   Anchor   = 'Right';          TextColor = 'Green';
   x1       = 55;               y1        = 94;
   Label    = 'Coefficients for the Selected Model';
   output;
```

```
   Function = 'Line';           x1        = .;
   x1Space  = 'DataValue';      x2Space   = x1Space;
   xc1      = '9+CrBB';         xc2       = '8+CrRuns*CrRuns';
   y1       = 94;               y2        = 94;
   output;

   Function = 'Rectangle';      y1Space   = 'WallPercent';
   Anchor   = 'BottomLeft';     y1        = 10;
   Height   = 80;               Width     = 0.6;
   output;

   ID       = 'LO3';            Width     = 100;
   Function = 'Text ';          Label     = 'Selected Value';
   x1Space  = 'DataPercent';    y1Space   = x1Space;
   Anchor   = 'Left';           TextColor = 'Blue';
   x1       = 86;               y1        = 84;
   output;

   Function = 'Arrow';          LineColor = 'Blue';
   x1Space  = 'DataValue';      x2Space   = x1Space;
   xc1      = '9+CrBB';         xc2       = '12+CrHits*CrHits';
   y1       = 4;                y2        = 83;
   DiscreteOffset = .1;         x1        = .;
   output;
run;
```

The following step checks the annotation data set:

```
%annocheck;
```

This step prints no messages because the data set is fine.

The following steps introduce an error and check the data set:

```
data anno2(drop=xc2);
   set anno;
run;

%annocheck(anno=anno2)
```

The results are displayed in Figure 4.76.

Figure 4.76 Missing Variable Annotation Errors

```
At least one of each pair must be present:
(x1,xc1), (x2,xc2), (y1,yc1), (y2,yc2)
Present variables are: x1 y1 y2 xC1

At least one of each pair must be present:
(x1,xc1), (x2,xc2), (y1,yc1), (y2,yc2)
Present variables are: y1 y2 xC1
x1=. x2=. y1=94 y2=94 xC1=9+CrBB xC2=  yC1=  yC2=
Error: x2 and xC1 are both missing.

At least one of each pair must be present:
(x1,xc1), (x2,xc2), (y1,yc1), (y2,yc2)
Present variables are: x1 y1 y2 xC1

At least one of each pair must be present:
(x1,xc1), (x2,xc2), (y1,yc1), (y2,yc2)
Present variables are: y1 y2 xC1
x1=. x2=. y1=4 y2=83 xC1=9+CrBB xC2=  yC1=  yC2=
Error: x2 and xC1 are both missing.
```

The following steps introduce two errors and check the data set:

```
data anno2;
   length y1space function $ 6;
   set anno;
run;

%annocheck(anno=anno2)
```

The results are displayed in Figure 4.77.

Figure 4.77 Short Lengths Annotation Errors

```
Invalid: y1Space=DataPe
Invalid: y1Space=DataPe
Invalid: function=Rectan
Invalid: y1Space=WallPe
Invalid: y1Space=DataPe
Invalid: y1Space=DataPe
```

References

Matange, S., and Heath, D. (2011). *Statistical Graphics Procedures by Example: Effective Graphs Using SAS*. Cary, NC: SAS Institute Inc.

Chapter 5
Bars, Lines, Curves, and Arrows

Contents

Adverse Events Plot

Double Click for Example Code

This example describes how to make adverse events plots (see Figure 5.1). Patient IDs are displayed on the Y axis, time is displayed on the X axis, and the graph shows the onset and duration of adverse events that can happen during an illness.

The data set consists of adverse events (complaints), starting date, ending date (if available), and patient ID. There can be multiple complaints per patient, and different complaints can occur simultaneously. When the end date is missing, the complaint is ongoing until the end of the study. This example builds on concepts (multiple axes, offsets, and constant variables) that are introduced in preceding examples. The following step reads the data:

```
data AdverseEvents;
   input Complaint $ 1-10 @11 StartDate Date9. @21
         EndDate Date9. Patient $ 31-40;
   datalines;
Nausea    23feb2011 23feb2011 1020120010
Vomiting  23feb2011 23feb2011 1020120010
Vomiting  19apr2011 19apr2011 1020120010
Diarrhea  20nov2010 25nov2010 1012120020
Nausea    21nov2010 25nov2010 1012120020
Nausea    21jul2011           1063120030
Diarrhea  04may2011 02jun2011 1070120060
Nausea    01may2011 01may2011 1070120060
Nausea    05may2011 05may2011 1070120060
Nausea    30may2011 31may2011 1070120060
Nausea    13jun2011 13jun2011 1070120060
Nausea    25jul2011 25jul2011 1070120060
Nausea    23aug2011 23aug2011 1070120060
Vomiting  10jun2011           1070120070
Nausea    06feb2011           1070120070
Nausea    04may2011 16jul2011 1030120022
Diarrhea  07may2011 10jul2011 1030120022
```

```
Vomiting  06may2011 12jul2011 1030120022
Diarrhea  25mar2011 26mar2011 1002120003
Nausea    25mar2011 04apr2011 1002120003
Diarrhea  11jan2011 31jan2011 1060120013
Diarrhea  04mar2011 05mar2011 1060120013
Diarrhea  25feb2011 26feb2011 1011120063
Diarrhea  18feb2011 16may2011 1011120024
Diarrhea  08may2011 10may2011 1080120054
Vomiting  08may2011 10may2011 1180120054
;
```

The analysis begins by sorting the data by patient ID and by complaint. PROC MEANS is run to obtain the last ending date. A DATA step substitutes the ending date for missing dates. The following steps process the input data:

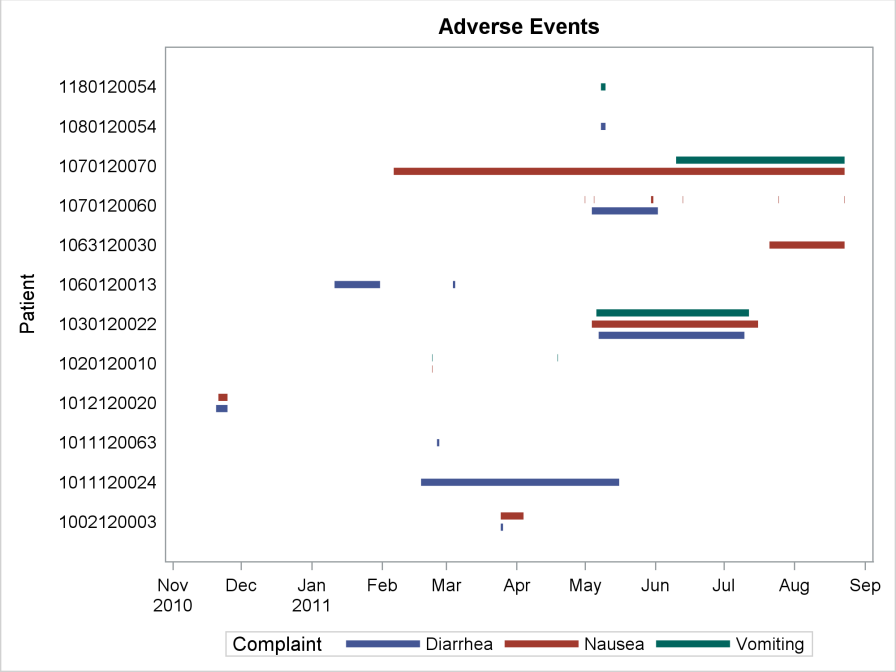
```
proc sort data=AdverseEvents;
   by Patient Complaint;
run;

proc means noprint data=AdverseEvents;
   output out=m(drop=_:) max(EndDate)=ma;
run;

data ae;
   set AdverseEvents;
   if _n_ = 1 then set m;
   if nmiss(enddate) then enddate = ma;
run;
```

PROC SGPLOT is used to make the plot:

```
title 'Adverse Events';
proc sgplot data=ae nocycleattrs;
   highlow y=patient low=startdate high=enddate /
           groupdisplay=cluster group=complaint lineattrs=(thickness=5px);
   format enddate mmddyy8.;
   xaxis display=(nolabel);
   yaxis display=(noticks);
run;
```

The HIGHLOW statement draws the bars from the start date to the end date for each adverse event for each patient. The GROUPDISPLAY=CLUSTER option displays different complaints in slightly offset lines. (See the section “Attribute Maps” on page 148 for another example of the GROUPDISPLAY=CLUSTER option.) The GROUP=COMPLAINT option displays each complaint in a different color. Because it is obvious from the tick labels that the X axis is a date axis, the axis label is suppressed. Tick marks are suppressed on the Y axis. The results are displayed in Figure 5.1.

Figure 5.1 Adverse Events Plot



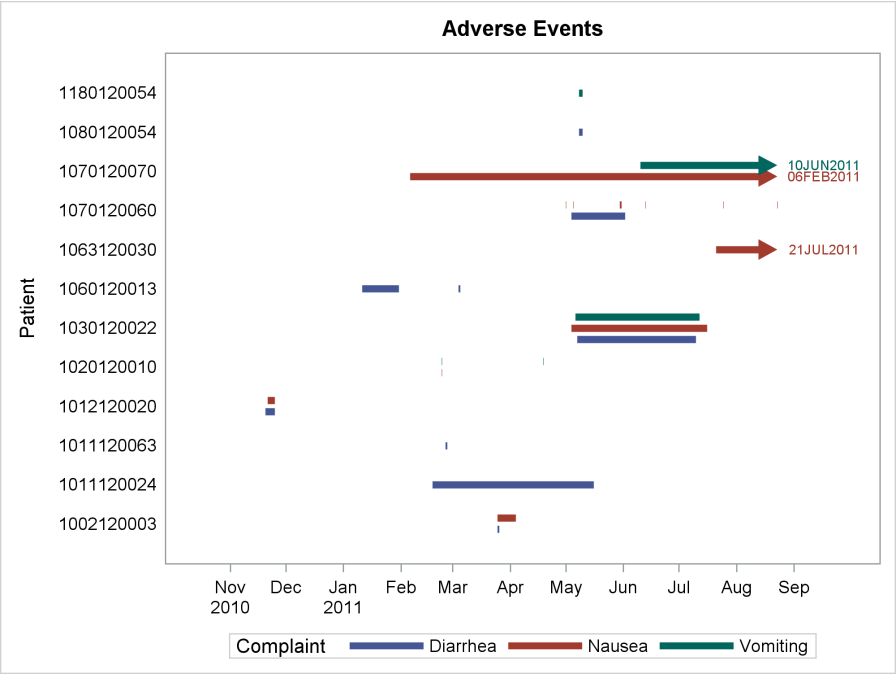
The next steps display an arrowhead on the right of each line for adverse events that are continuing. The onset date is displayed to the right of the arrowhead, and the dates are displayed using the same colors that are used for the bars in the graph. The following steps process the data and create the graph:

```
data ae;
   set AdverseEvents;
   if _n_ = 1 then set m;
   if nmiss(enddate) then do;
      hc = 'FILLEDARROW';
      enddate = ma;
      l = put(startdate, date9.);
   end;
   retain t 1;
run;

proc sgplot data=ae nocycleattrs;
   highlow y=patient low=startdate high=enddate /
           groupdisplay=cluster highcap=hc
           group=complaint lineattrs=(thickness=5px);
   scatter y=patient x=t / markerchar=l
           group=complaint groupdisplay=cluster x2axis;
   format enddate mmddyy8.;
   xaxis  display=(nolabel) offsetmax=0.12;
   yaxis  display=(noticks);
   x2axis display=(noticks nolabel novalues) offsetmin=0.92;
run;
```

This example uses a new variable, hc, which has the value `'FILLEDARROW'` when the ending date is missing and is blank otherwise. The HIGHCAP=HC option in the HIGHLOW statement selectively displays the arrowheads, and the SCATTER statement displays the onset dates. The constant variable t in the SCATTER statement maps to the X2 axis. Its value (1) is arbitrary because only one value maps to the X2 axis. The OFFSETMIN=0.92 option in the X2AXIS statement reserves 92% of the space to the right of the graph for the dates. Similarly, the OFFSETMAX=0.12 option in the XAXIS statement reserves 12% of the space to

the right of the graph for the dates. Nothing is displayed on the X2 axis; it serves only to reserve space for the display of the onset dates. The results are displayed in Figure 5.2.

Figure 5.2 Adverse Events Plot



Attribute Maps

Double Click for Example Code

When you want groups of observations to be distinguished from other groups, you can specify a GROUP= variable. For example, in the baseball data set that is used in this example, the player's position is a logical group variable. Other common group variables are sex, age group, and so on. By default, groups are distinguished based on the colors, markers, and line styles that are specified in the style elements **GraphData1** for the first group, **GraphData2** for the second group, and so on. The correspondence between groups and style elements is implicit. You can modify styles to change how groups are differentiated, or you can use attribute maps instead. Attribute maps enable you to specify an explicit mapping between groups and attributes.

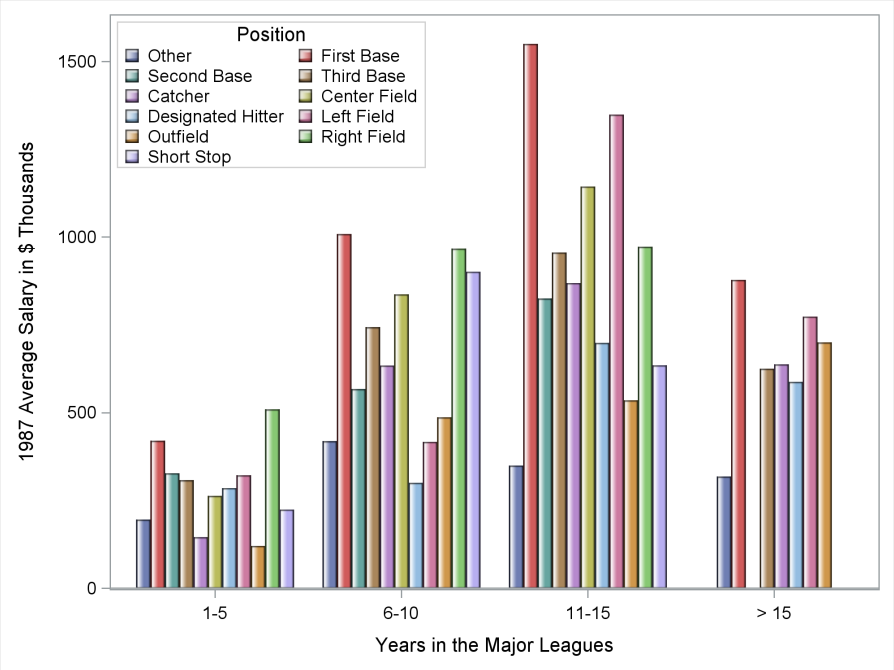
Before illustrating attribute maps, the following step uses the default group colors to display the average salary for various baseball positions by years in the major leagues:

```
proc format;
   value $pos  '1B' = 'First Base'    '2B' = 'Second Base'
               '3B' = 'Third Base'    'C'  = 'Catcher'
               'CF' = 'Center Field'  'DH' = 'Designated Hitter'
               'LF' = 'Left Field'    'OF' = 'Outfield'
               'RF' = 'Right Field'   'SS' = 'Short Stop'
               '  ' = '  '           Other = 'Other';
   value yearf 1-5 = '1-5' 6-10 = '6-10' 11-15 = '11-15' other = '> 15' . = ' ';
run;
```

```
proc sgplot data=sashelp.baseball;
   vbar yrmajor / response=salary stat=mean groupdisplay=cluster
        group=position dataskin=gloss name='pos';
   keylegend 'pos' / location=inside position=topleft title='Position' across=2;
   yaxis offsetmax=0.05 label='1987 Average Salary in $ Thousands';
   format position $pos. yrmajor yearf.;
run;
```

The results are displayed in Figure 5.3. This example illustrates how to use a group variable along with the GROUPDISPLAY=CLUSTER option. (See the section "Adverse Events Plot" on page 145 for another example of GROUPDISPLAY=CLUSTER option.) You can see the effect of the GROUPDISPLAY=CLUSTER option in Figure 5.3. For each year group on the X axis, there is a cluster of bar charts that has one bar for each level of the GROUP= variable Position. The color for each position comes from the **GraphData1**–**GraphData11** style elements.

Figure 5.3 Default Bar Chart of the Baseball Data



The following step creates and displays an attribute map:

```
data attrmap;
   retain ID 'map' LineColor 'Black'
          Value '                  ' FillColor '         ';
   input Value $ 1-18 FillColor $;
   datalines;
First Base            Green
Third Base            Red
Center Field          Blue
Left Field            Cyan
Right Field           Magenta
Second Base           Orange
```

```
Catcher               Yellow
Designated Hitter     Gray
Outfield              Brown
Short Stop            Black
Other                 White
;

proc print noobs;
run;
```

The attribute map is displayed in Figure 5.4. The ID variable contains the name of the attribute map. There can be more than one attribute map name, but for now there is only one, and it is simply called `'map'`. The LineColor variable contains the color of the lines that surround each bar, the Value variable contains the group value, and the FillColor variable contains the color for each group.

Specifying an attribute map is easier than making the corresponding style change. Furthermore, attribute maps enable you to target specific parts of the graph. Attribute maps resemble SG annotation data sets in that both require special variable names. However, annotation data sets contain instructions for adding elements to a graph, whereas attribute maps change how specific statements distinguish between groups of observations.

Figure 5.4 Attribute Map

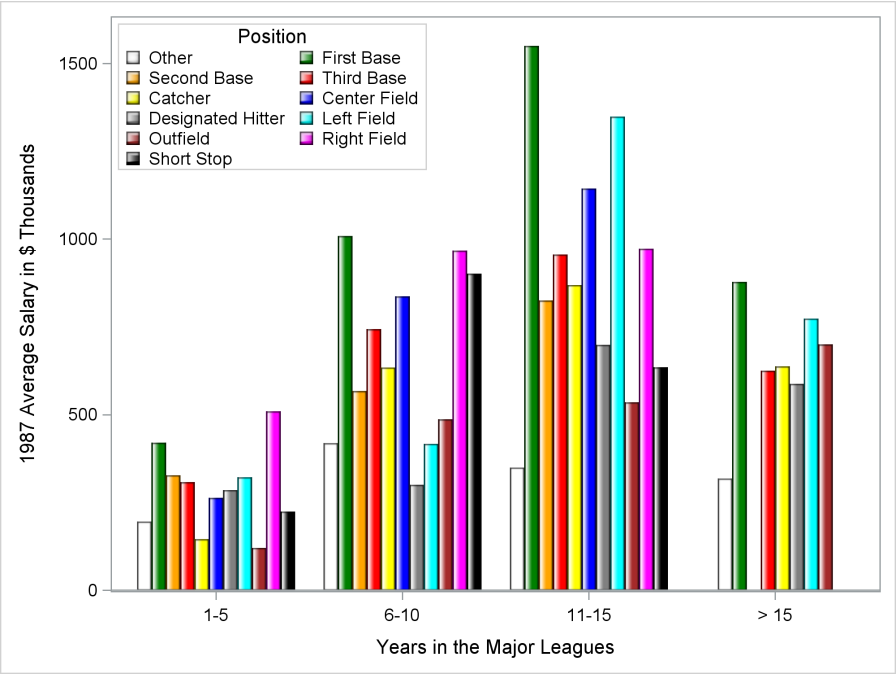
ID	LineColor	Value	FillColor
map	Black	First Base	Green
map	Black	Third Base	Red
map	Black	Center Field	Blue
map	Black	Left Field	Cyan
map	Black	Right Field	Magenta
map	Black	Second Base	Orange
map	Black	Catcher	Yellow
map	Black	Designated Hitter	Gray
map	Black	Outfield	Brown
map	Black	Short Stop	Black
map	Black	Other	White

The following step uses the attribute map to create the graph:

```
proc sgplot data=sashelp.baseball dattrmap=attrmap;
   vbar yrmajor / response=salary stat=mean groupdisplay=cluster
        group=position dataskin=gloss name='pos' attrid=map;
   keylegend 'pos' / location=inside position=topleft title='Position' across=2;
   yaxis offsetmax=0.05 label='1987 Average Salary in $ Thousands';
   format position $pos. yrmajor yearf.;
run;
```

The DATTRMAP= option in the PROC SGPLOT statement names the attribute map data set, and the ATTRID= option in the VBAR statement names the attribute map ID that is used for that statement. The results are displayed in Figure 5.5.

Figure 5.5 Bar Chart of the Baseball Data



The next steps process the data to display two bar charts for each year group: position (as before) and division (American League East, American League West, National League East, and National League West):

```
proc summary data=sashelp.baseball;
   class  yrmajor div position;
   format position $pos. yrmajor yearf.;
   output out=means(where=(_way_ = 2)) mean(salary)=ms / ways;
run;

data all(drop=_:);
   divmean = .;
   merge means(where=(_type_=5) rename=(ms=posmean) drop=div)
         means(where=(_type_=6) rename=(ms=divmean) drop=position);
   by yrMajor;
run;
```

You cannot use two VBAR statements in this context. You must summarize the data yourself and then use the VBARPARM statement. When you have done the data summarization yourself, you use statements that contain PARM in their names. The preceding steps create the mean salary for each YrMajor × Position combination and YrMajor × Div combination (in addition to other combinations), discard the unnecessary combinations, and create a data set that contains two mean salary variables, one for YrMajor × Position and one for YrMajor × Div.

The following steps create a data set that contains two attribute maps, one for the position and one for division:

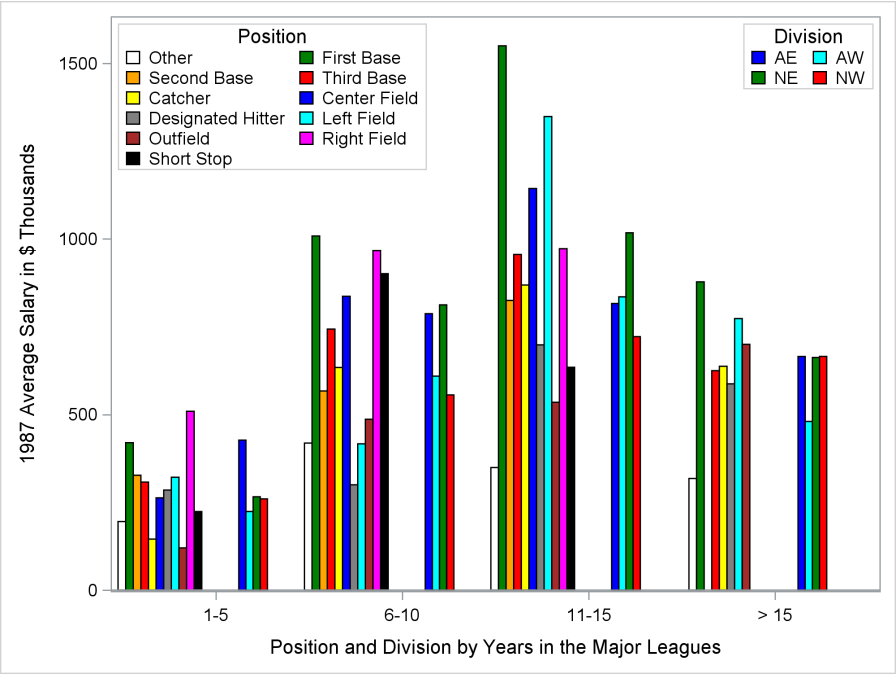
```
data attrmap;
   retain ID '    ' LineColor 'Black' FillColor '         ';
   input ID $ Value $ 6-22 FillColor $;
   datalines;
map1 First Base            Green
map1 Third Base            Red
map1 Center Field          Blue
map1 Left Field            Cyan
map1 Right Field           Magenta
map1 Second Base           Orange
map1 Catcher               Yellow
map1 Designated Hitter     Gray
map1 Outfield              Brown
map1 Short Stop            Black
map1 Other                 White

map2 NE                    Green
map2 NW                    Red
map2 AE                    Blue
map2 AW                    Cyan
;
```

The ID variable contains the names of the two attribute maps. The colors in the two maps overlap, but the meaning is obvious from the context of the graph. Green in the right cluster corresponds to green in the right legend (`'NE'`), and green in the left cluster corresponds to green in the left legend (`'First Base'`). Also, the two sets of bars are distinguished by the outline color. The following step creates the graph:

```
proc sgplot data=all dattrmap=attrmap;
   vbarparm category=yrmajor  response=posmean /  group=position
            clusterwidth=0.45 discreteoffset=-.3  name='pos' attrid=map1
            groupdisplay=cluster;
   vbarparm category=yrmajor  response=divmean /  group=div
            clusterwidth=0.15 discreteoffset= .2  name='div' attrid=map2
            groupdisplay=cluster;
   keylegend 'pos' / across=2 location=inside position=topleft
                     title='Position';
   keylegend 'div' / across=2 location=inside position=topright
                     title='Division';
   yaxis label='1987 Average Salary in $ Thousands' offsetmax=0.05;
   xaxis label='Position and Division by Years in the Major Leagues';
run;
```

The ATTRID= option in each VBARPARM statement specifies which attribute map applies to which statement. The results are displayed in Figure 5.6.

Figure 5.6 Bar Chart of the Baseball Data



The graph is created by two VBARPARM statements, each of which has a CATEGORY=YrMajor option, so each statement creates one bar chart for each year group. The first VBARPARM statement has a cluster width of 0.45, so it uses 45% of the available area for each group. The second chart uses 15%. The first set of bars in each group is offset to the left –0.3, and the second set of bars is offset to the right 0.2. You can specify values in the range –0.5 to 0.5. Jointly specifying CLUSTERWIDTH= and DISCRETEOFFSET= enables you to display multiple graphs, side-by-side.

Connecting Points with Lines, Arrows, and Curves

Double Click for Example Code

This example creates artificial data and uses them to show how you can connect pairs of points by using lines, arrows, and curves. It begins by creating a data set that contains the coordinates of random points along a circle:

```
data x(drop=t);
   do id = 1 to 100;
      t = uniform(17) * 2 * constant('pi');
      x = cos(t);
      y = sin(t);
      output;
   end;
run;
```

The next step selects 20 random pairs of points from data set x:

```
data lines(drop=i id);
   do i = 1 to 20;
      g   + 1;
      id1 = ceil(uniform(17) * 100);
      id2 = ceil(uniform(17) * 100);
      set x(rename=(x=x1 y=y1)) point=id1; output;
      set x(rename=(x=x1 y=y1)) point=id2; output;
   end;
   stop;
run;
```

This step creates two random ID variables and then creates two coordinate variables, x1 and y1, from the original random data set. A group variable g identifies each pair of points (each group of two observations) that is to be connected.

The following step merges the two data sets:

```
data both;
   merge x lines;
run;
```

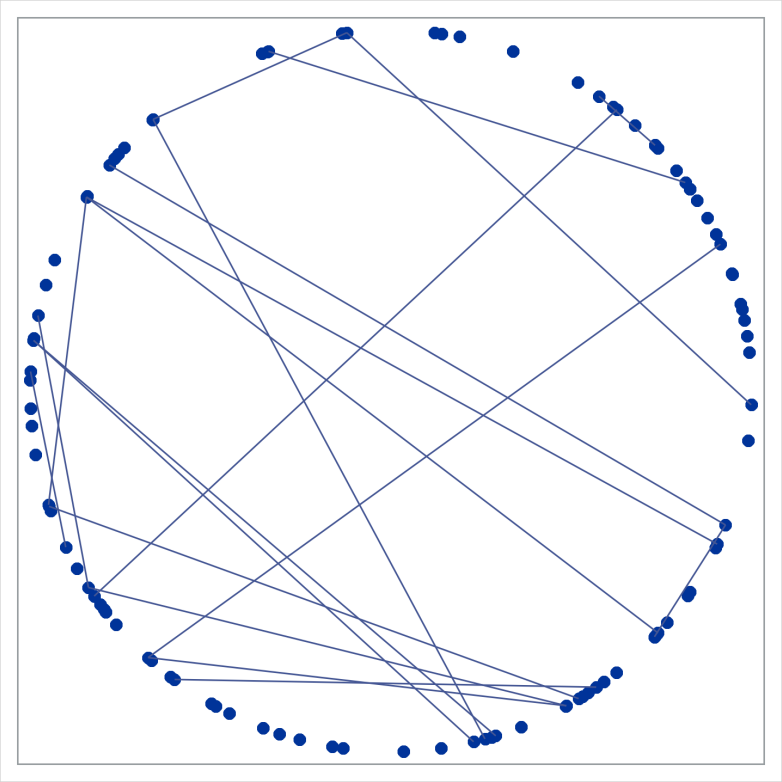
The variables x and y contain the coordinates of the points on the circle. The variables x1 and y1 contain the coordinates of the points that are connected. The numbers of nonmissing observations in the two partitions of the data set are different. This is expected and fine.

The following step displays the points and the connections:

```
ods graphics on / width=4.8in height=4.8in;
proc sgplot data=both noautolegend;
   scatter y=y  x=x  / markerattrs=(symbol=circlefilled);
   series  y=y1 x=x1 / group=g lineattrs=graphdata1(pattern=solid);
   xaxis display=none;
   yaxis display=none;
run;
```

The SERIES statement along with a group variable displays the connections as straight lines. The results are displayed in Figure 5.7.

Figure 5.7 Points Connected by Lines

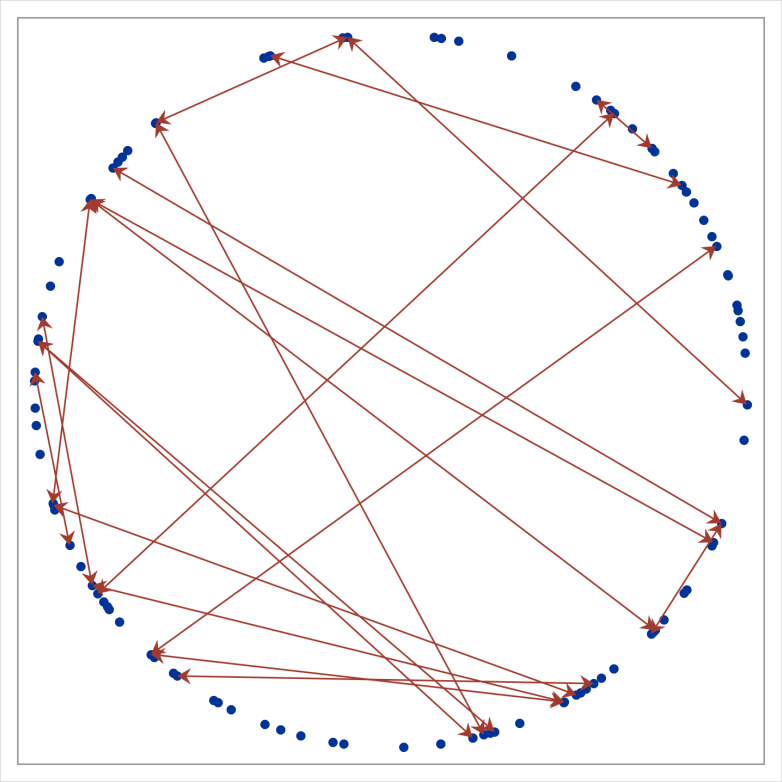


You can add the ARROWHEADPOS=BOTH option to the SERIES statement to add arrowheads to both ends of each line:

```
proc sgplot data=both noautolegend;
   scatter y=y  x=x  / markerattrs=(symbol=circlefilled size=5px);
   series  y=y1 x=x1 / group=g lineattrs=graphdata2(pattern=solid)
                       arrowheadpos=both arrowheadshape=barbed;
   xaxis display=none;
   yaxis display=none;
run;
```

The style element was changed along with the size of the markers to more clearly display the parts of the arrowheads that coincide with the markers. The results are displayed in Figure 5.8.

Figure 5.8 Points Connected by Arrows



This graph and other graphs in this example illustrate several options for arrowhead construction. The shape option is ARROWHEADSHAPE=OPEN | FILLED | BARBED, and the size option is ARROWHEADSCALE=*positive-number*. By default, ARROWHEADSHAPE=OPEN and ARROWHEADSCALE=1.

You might prefer to connect the points by using something less rectilinear. The next steps show you how to use curves to connect the points. The curves are drawn by the SPLINE statement. The following step creates the data set Curves:

```
data Curves(drop=i id t: m d);
   do i = 1 to 20;
      g   + 1;
      id1 = ceil(uniform(17) * 100);
      id2 = ceil(uniform(17) * 100);
      set x(rename=(x=t1 y=t2)) point=id1;
      set x(rename=(x=t3 y=t4)) point=id2;
      d  = (t4 - t2) ** 2 + (t3 - t1) ** 2;
      x1 = t1;
      y1 = t2;
      output;                              /* output the starting point */
      td = ifn(abs(t4 - t2) lt 1e-12, 1e-12, t4 - t2);
      m  = -(t3 - t1) / td;
      t1 = mean(t1, t3);
      t2 = mean(t2, t4);
      x1 = t1 + ifn(t1 gt 0 or td eq 1e-12, -1, 1) *
               sqrt(0.1 * d / (1 + m * m));
      y1 = m * (x1 - t1) + t2;
```

```
      output;                              /* output the midpoint       */
      x1 = t3;
      y1 = t4;
      output;                              /* output the ending point   */
      end;
   stop;
run;
```

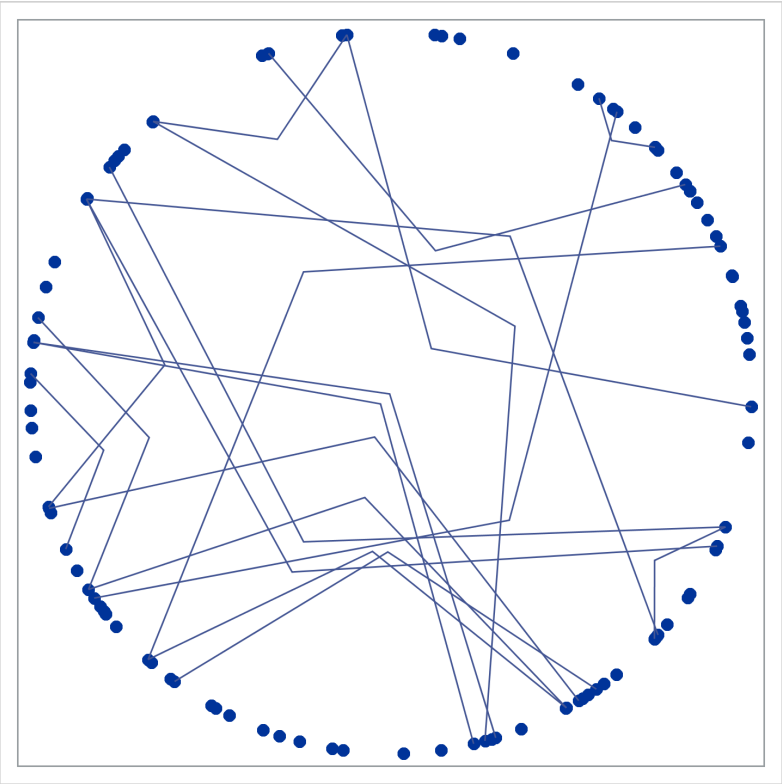
The data set Curves has three observations for each curve: the starting point, a middle point, and the end point. In contrast, the data set Lines has two observations for each line: the starting point and the end point. The DATA step code for the Curves data set is more complicated than the DATA step code for the Lines data set because of the computation of this middle point. First, both end points are read and saved so that they can be used in middle-point computations and be output in order. Then the midpoint of the straight connector line is found. Next, a point on the line perpendicular to the connector line and passing through the midpoint is found. (The variable m is the slope of a line that is perpendicular to the straight connector.) The function **`ifn(t1 gt 0 or td eq 1e-12, -1, 1)`** ensures that the middle point is in the direction of the center of the circle. It provides the sign that is used to move the point away from the connector line. A value of –1 moves the point to the left when the X coordinate for the midpoint is greater than 0 and to the right otherwise. It also moves the point down when the denominator of the slope is 0. The distance between the middle point of the curve and the midpoint of the connector line is scaled by the squared length of the connector line (**`0.1 * d`**). Hence, the longer curves pass through points farther from the straight connector line than do the shorter curves.

The following steps merge the two data sets and create a graph, which has piecewise linear connections between the points (using the starting, middle, and ending points):

```
data both;
   merge x curves;
run;

proc sgplot data=both noautolegend;
   scatter y=y  x=x  / markerattrs=(symbol=circlefilled);
   series  y=y1 x=x1 / group=g lineattrs=graphdata1(pattern=solid);
   xaxis display=none;
   yaxis display=none;
run;
```

The results are displayed in Figure 5.9.

Figure 5.9 Piecewise Linear Connections

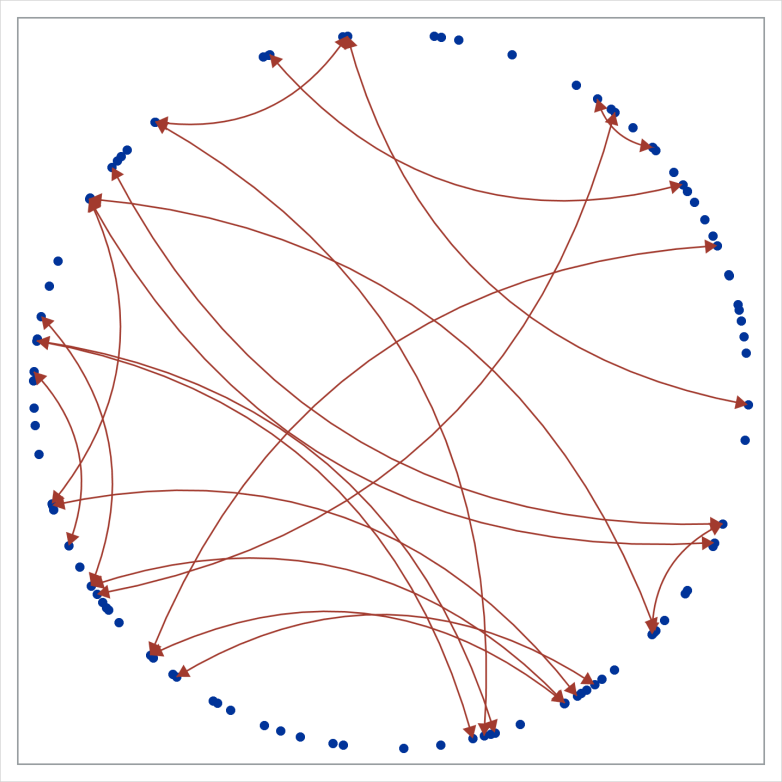


The following step creates the desired graph, in which curved arrows connect the points:

```
proc sgplot data=both noautolegend;
   scatter y=y  x=x  / markerattrs=(symbol=circlefilled size=5px);
   spline  y=y1 x=x1 / group=g lineattrs=graphdata2(pattern=solid)
                       arrowheadpos=both arrowheadshape=filled;
   xaxis display=none;
   yaxis display=none;
run;
```

The results are displayed in Figure 5.10.

Figure 5.10 Points Connected by Curved Arrows



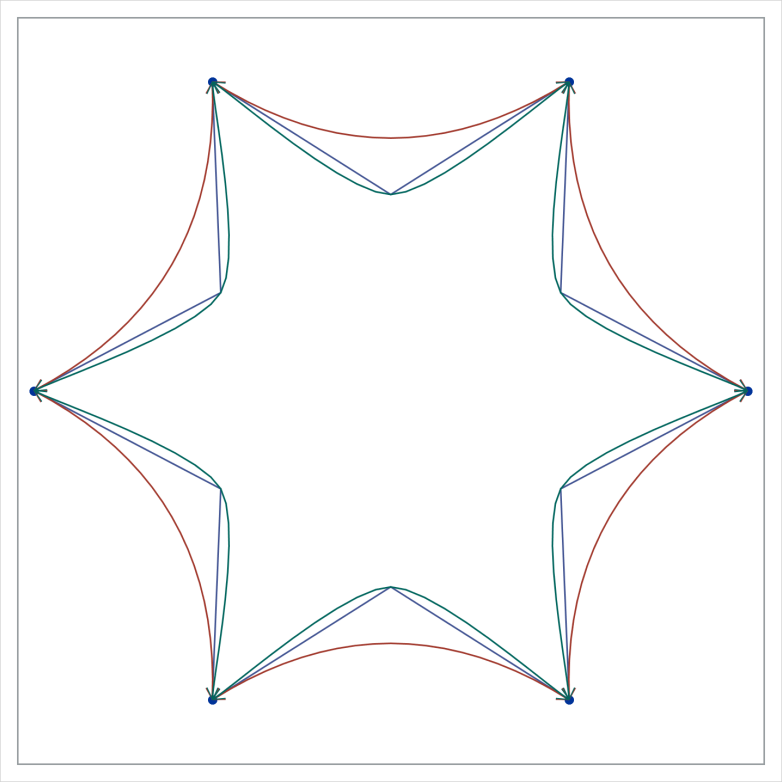
By comparing Figure 5.9 and Figure 5.10, you can see that the spline does not precisely hit the middle point. The SPLINE statement smooths the points to create the curve, but it does not connect all the points. Both the SERIES statement and the SPLINE statement enable you to draw lines or curves with arrowheads. You can smoothly connect the points by using the SMOOTHCONNECT option in the SERIES statement, for example as follows:

```
data x(drop=t);
   do id = 1 to 6;
      t = (id - 1) * 2 * constant('pi') / 6;
      x = cos(t);
      y = sin(t);
      output;
   end;
run;

data curves(drop=id t: m d);
   do id1 = 1 to 6;
      id2 = mod(id1, 6) + 1;
      g   + 1;
      set x(rename=(x=t1 y=t2)) point=id1;
      set x(rename=(x=t3 y=t4)) point=id2;
      d  = (t4 - t2) ** 2 + (t3 - t1) ** 2;
      x1 = t1;
      y1 = t2;
      output;                              /* output the starting point */
      td = ifn(abs(t4 - t2) lt 1e-12, 1e-12, t4 - t2);
```

```
      m  = -(t3 - t1) / td;
      t1 = mean(t1, t3);
      t2 = mean(t2, t4);
      x1 = t1 + ifn(t1 gt 0 or td eq 1e-12, -1, 1) *
               sqrt(0.1 * d / (1 + m * m));
      y1 = m * (x1 - t1) + t2;
      output;                          /* output the midpoint       */
      x1 = t3;
      y1 = t4;
      output;                          /* output the ending point   */
      end;
   stop;
   run;

data both;
   merge x curves;
run;

proc sgplot data=both noautolegend;
   scatter y=y  x=x  / markerattrs=(symbol=circlefilled size=5px);
   series  y=y1 x=x1 / group=g lineattrs=graphdata1(pattern=solid)
                       arrowheadpos=both arrowheadscale=2;
   spline  y=y1 x=x1 / group=g lineattrs=graphdata2(pattern=solid)
                       arrowheadpos=both arrowheadscale=2;
   series  y=y1 x=x1 / group=g lineattrs=graphdata3(pattern=solid)
                       arrowheadpos=both arrowheadscale=2 smoothconnect;
   xaxis display=none;
   yaxis display=none;
run;
```

These steps create equally spaced points and connect the adjacent points as shown in Figure 5.11. The blue piecewise linear connectors and the green smooth connectors connect the same three points. The red splines do not reach the middle point.

Figure 5.11 Piecewise Linear, Smooth, and Spline Connectors



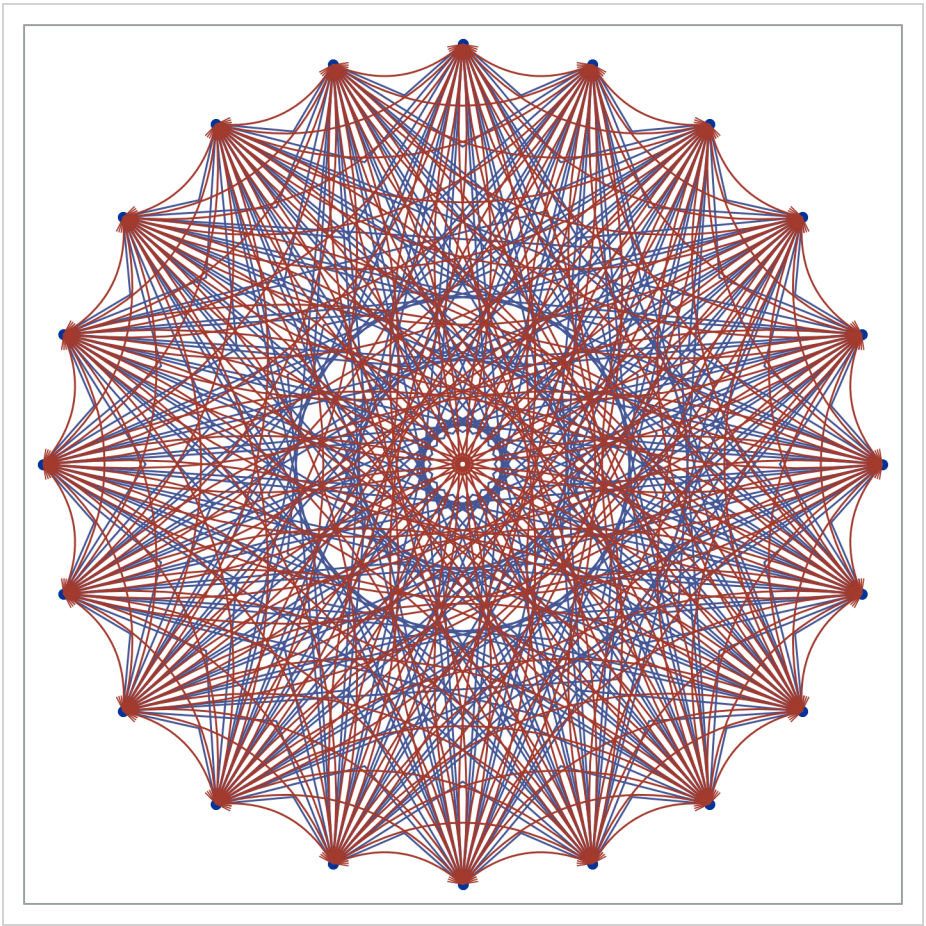
You can display all pairs of piecewise linear and spline connectors between 20 evenly spaced points on the circle as follows:

```
data x(drop=t);
   do id = 1 to 20;
      t = (id - 1) * 2 * constant('pi') / 20;
      x = cos(t);
      y = sin(t);
      output;
   end;
run;

data curves(drop=id t: m d);
   do id1 = 1 to 20;
      do id2 = id1 + 1 to 20;
         g  + 1;
         set x(rename=(x=t1 y=t2)) point=id1;
         set x(rename=(x=t3 y=t4)) point=id2;
         d  = (t4 - t2) ** 2 + (t3 - t1) ** 2;
         x1 = t1;
         y1 = t2;
         output;                              /* output the starting point */
         td = ifn(abs(t4 - t2) lt 1e-12, 1e-12, t4 - t2);
         m  = -(t3 - t1) / td;
         t1 = mean(t1, t3);
         t2 = mean(t2, t4);
```

```
            x1 = t1 + ifn(t1 gt 0 or td eq 1e-12, -1, 1) *
                      sqrt(0.1 * d / (1 + m * m));
            y1 = m * (x1 - t1) + t2;
            output;                                  /* output the midpoint       */
            x1 = t3;
            y1 = t4;
            output;                                  /* output the ending point   */
            end;
         end;
      stop;
      run;

   data both;
      merge x curves;
   run;

   proc sgplot data=both noautolegend;
      scatter y=y  x=x  / markerattrs=(symbol=circlefilled size=5px);
      series  y=y1 x=x1 / group=g lineattrs=graphdata1(pattern=solid);
      spline  y=y1 x=x1 / group=g lineattrs=graphdata2(pattern=solid)
                          arrowheadpos=both;
      xaxis display=none;
      yaxis display=none;
   run;
```

The results are displayed in Figure 5.12.

Figure 5.12 Fun with Splines



Chapter 6
Plots of Labeled Points

Contents

Placing Labels in Scatter Plots

Double Click for Example Code

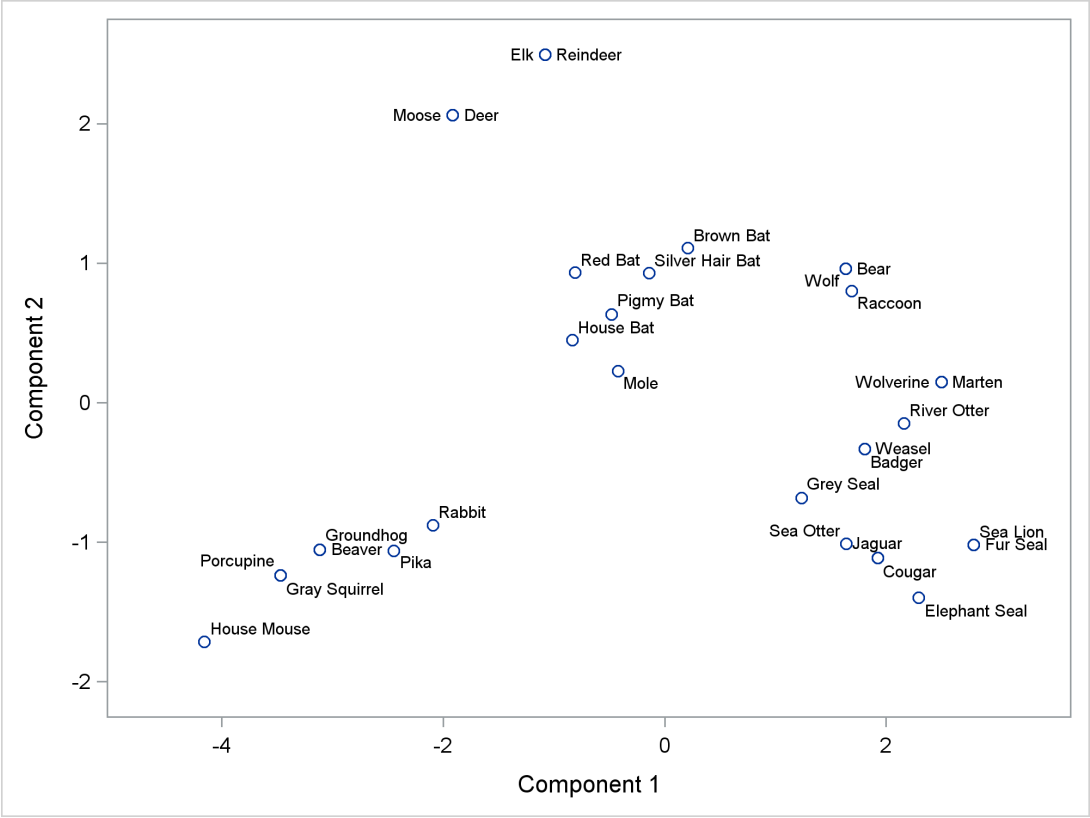
ODS Graphics can display labels in scatter plots. The following steps illustrate:

```
title 'Mammals'' Teeth';

data teeth;
   input Mammal $ 1-16 @21 (v1-v8) (1.);
   label v1='Top Incisors'    v2='Bottom Incisors'
         v3='Top Canines'     v4='Bottom Canines'
         v5='Top Premolars'   v6='Bottom Premolars'
         v7='Top Molars'      v8='Bottom Molars';
   datalines;
Brown Bat           23113333
Mole                32103333
Silver Hair Bat     23112333
Pigmy Bat           23112233
House Bat           23111233

   ... more lines ...

Reindeer            04103333
Elk                 04103333
Deer                04003333
Moose               04003333
;

proc princomp data=teeth out=teeth(drop=v:) n=2;
run;

data teeth;
   set teeth;
   label prin1 = 'Component 1' prin2 = 'Component 2';
run;
```

```
proc sgplot data=teeth;
   scatter y=prin2 x=prin1 / datalabel=mammal;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

The first two principal components of the mammals' teeth data set are particularly well suited for illustrating label placement. There are several pairs of coincident points and several clusters of points that all need to be reasonably labeled. The label placement produced by the DATALABEL= option in the SCATTER statement and displayed in Figure 6.1 is easy and automatic, and it is reasonable for most purposes. However, if you are making the final graph for inclusion in a paper or presentation, you might want to have more explicit control over the label placements. This example shows you ways in which you can do that.

Figure 6.1 Default Label Placement (Greedy Algorithm)

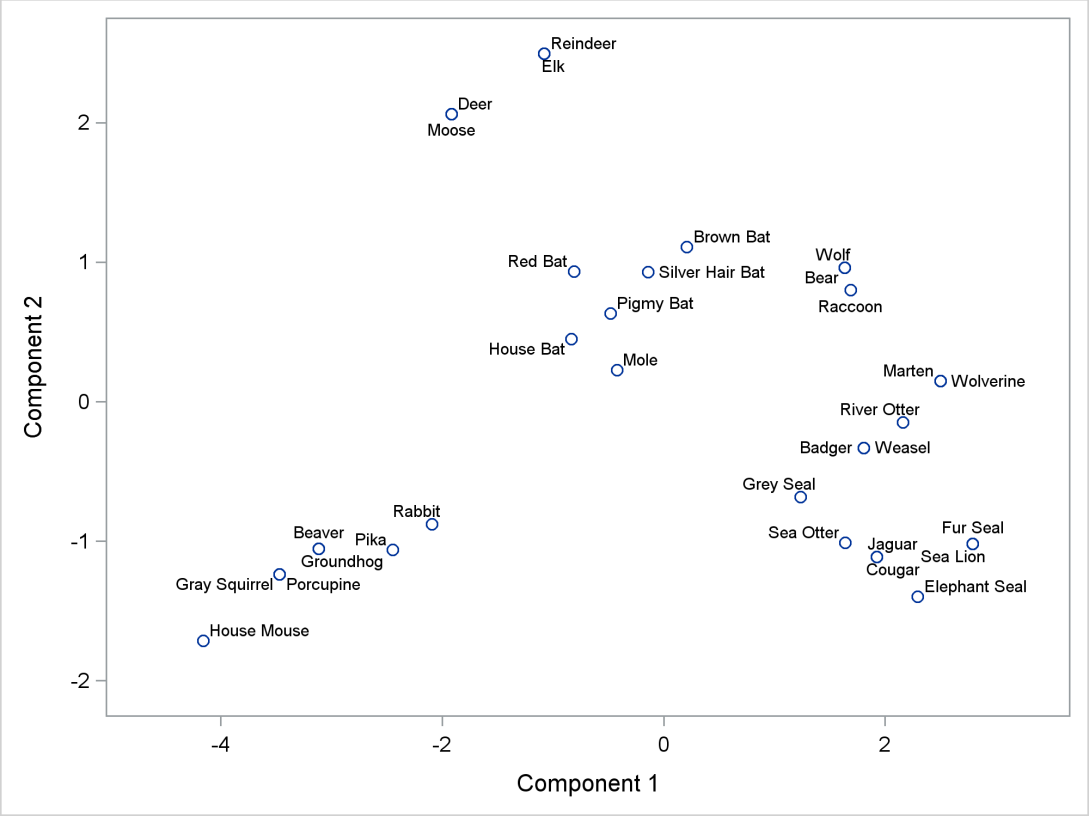


The default label placement algorithm that is used to make Figure 6.1 is a greedy algorithm. It seeks to improve the placements at every step of the iterations. You can instead request a simulated annealing approach that permits placements to get worse at times in the hopes of improving the final label positions. The following step illustrates:

```
ods graphics on / labelplacement=sa;
proc sgplot data=teeth;
   scatter y=prin2 x=prin1 / datalabel=mammal;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;

ods graphics on / reset=labelplacement;
```

The results are displayed in Figure 6.2. The results in Figure 6.1 and Figure 6.2 are quite different. Each approach has advantages and disadvantages.

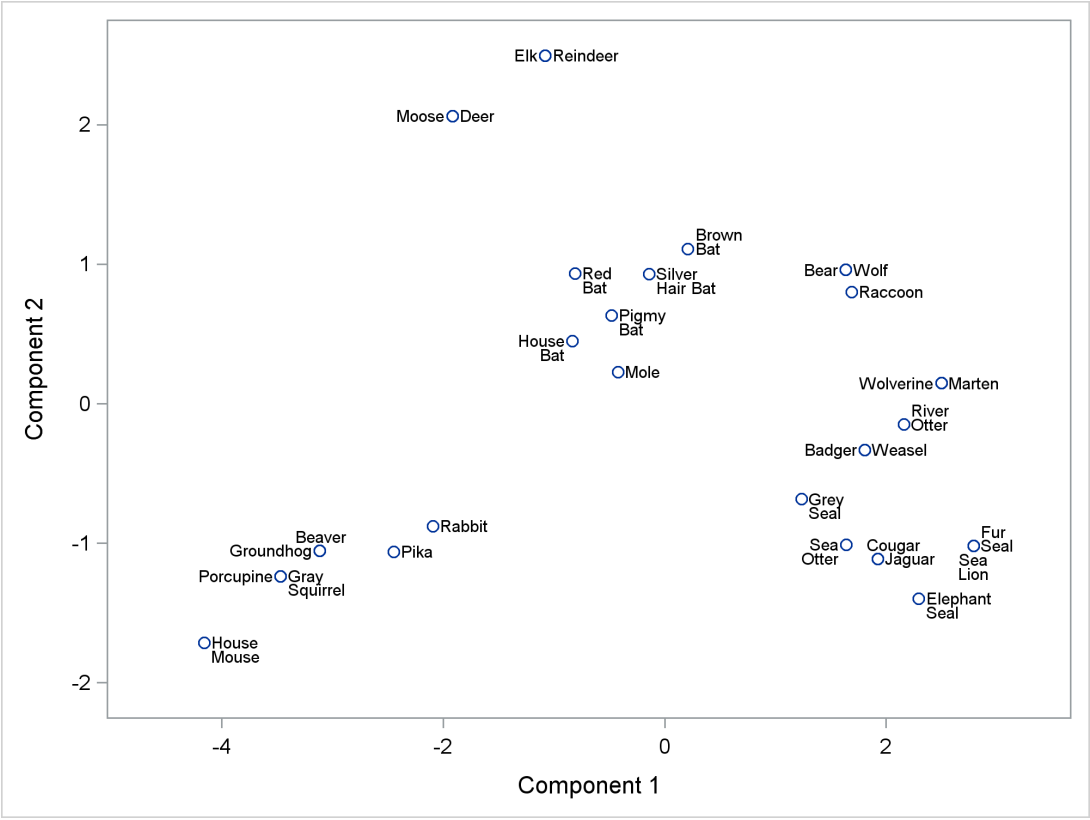
Figure 6.2 Label Placement via Simulated Annealing



Neither of these approaches enables you to control label placements or even specify your preferences. For example, you might prefer labels to the right and left of the point and you might prefer the first character of the label not to be immediately below the point (like many of the labels in Figure 6.2).

The rest of this example discusses an approach you can take to control label placement. This approach requires more work and is more fragile than the two approaches previously shown, but it enables you to make very nice graphs. Figure 6.3 illustrates. The code that is used to make this graph is displayed later in this example.

Figure 6.3 More Flexible Label Placement



In Figure 6.3, labels that contain blanks are split onto two lines. All coincident points are nicely labeled without ambiguity. Furthermore, if you prefer any other changes, you can easily make them on a point-by-point basis. For example, the label for Red Bat is unambiguous, but you might prefer it on the left and away from the other bats. You can easily change only that one label.

It is instructive at this point to review the history of label placement in SAS. The IDPLOT procedure in SAS Version 5 positioned labels in scatter plots and tried to avoid collisions. In SAS Version 6, the PLOT procedure was modified to incorporate some of these capabilities. Both procedures produced printer plots. Later the %PlotIt macro was developed to use PROC PLOT to position labels, but it uses the SAS/GRAPH annotate facility to display the results. In SAS 9, ODS Graphics provided the first truly graphical approach to label placement along with collision avoidance.

The rest of this example combines PROC PLOT, PROC SGPLOT, and SG annotation to position labels in scatter plots. PROC PLOT options control the range of label movement and specify preferences for label placement. However, the graph is drawn by PROC SGPLOT, and the labels are placed by using SG annotation. The macro %LabelIt bundles these steps.

Using the %LabelIt Macro

PROC PLOT knows nothing about PROC SGPLOT, and PROC SGPLOT knows nothing about PROC PLOT. PROC PLOT creates a grid of characters and maps each symbol and each label character to one cell of the grid.[1] It optionally produces a table that contains each point, the cell location for the symbol, and the starting

[1] The size of the grid is determined by the line size (LINESIZE= or LS=) and page size (PAGESIZE= or PS=) system options.

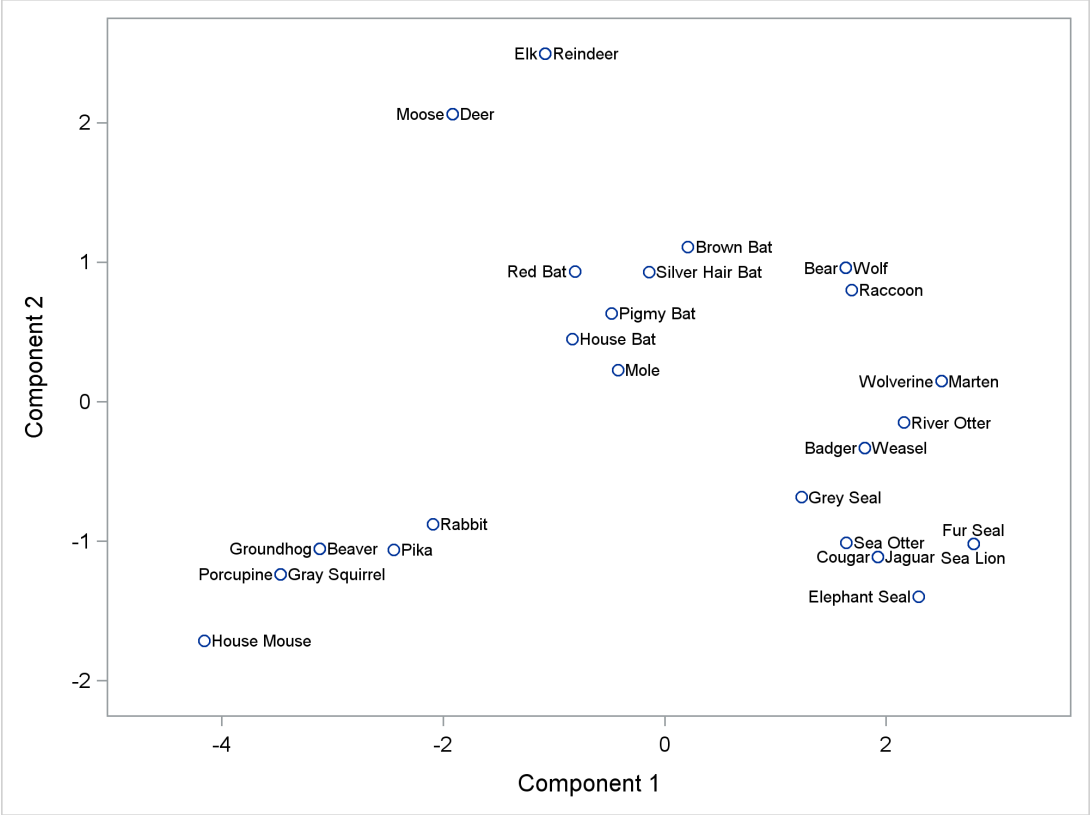
position and offsets that are used to position the label. PROC SGPLOT is graphical. The %LabelIt macro reads the table produced by PROC PLOT, and it uses simple regression to determine the mapping from cells to X- and Y-axis values. Then it constructs an annotation data set, which contains the labels and the data values needed to provide the coordinates for the labels. %LabelIt is not a full-featured macro. If you want to use it, you need to modify it in various ways to make it better suited to your problem. The rest of this example illustrates.

The following step, which creates Figure 6.4, is most basic example of using this macro:

```
%labelit(prin2, prin1, mammal, data=teeth)
```

The arguments are the Y-axis variable Prin2, the X-axis variable Prin1, the label variable Mammal, and the input data set. The %LabelIt macro uses default label placement.

Figure 6.4 Basic Label Placement



Notice that Fur Seal and Sea Lion are not positioned to the right of the point even though there appears to be room. Recall that PROC PLOT knows nothing about PROC SGPLOT and PROC SGPLOT knows nothing about PROC PLOT. There might be room for the label in PROC SGPLOT, but there might not be room when PROC PLOT positions the labels. Particularly when you are creating a graph of labeled points, you should ensure that you add extra space around the margins to accommodate point labels. The %LabelIt macro runs PROC SGPLOT as follows:

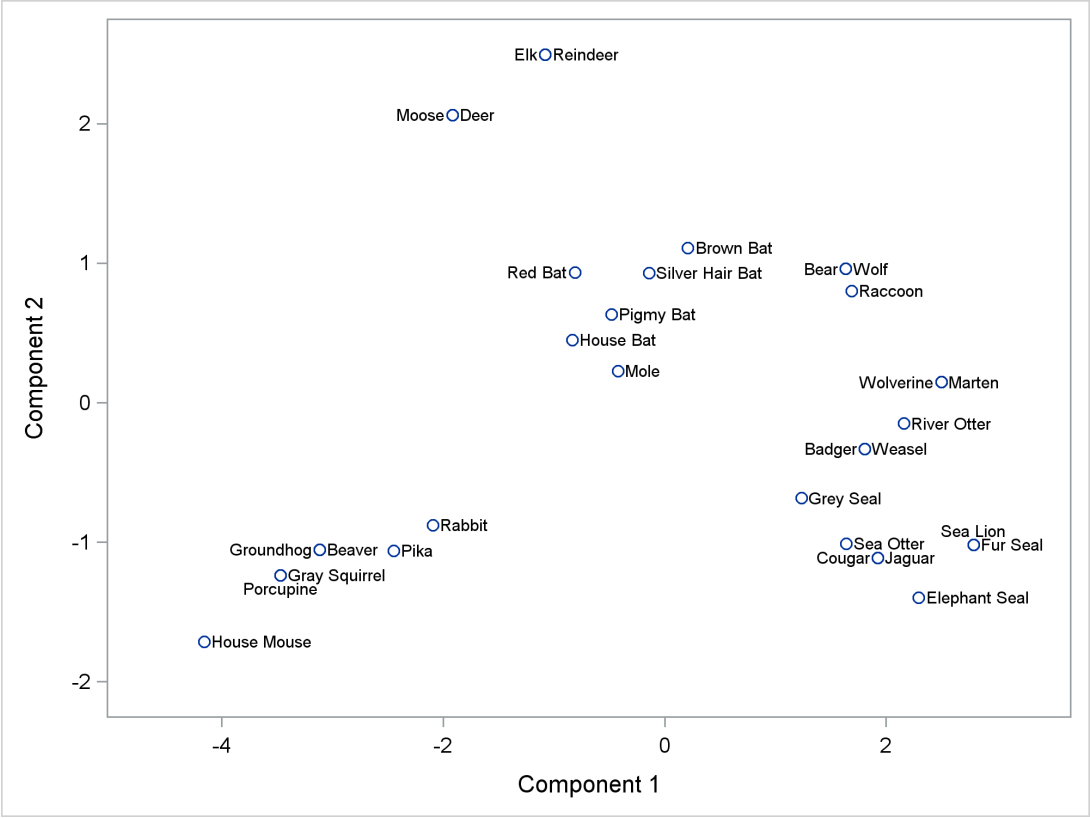
```
proc sgplot data=&data sganno=anno;
   scatter y=&y x=&x;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

The X- and Y-axis offsets add extra space for labels. One way of providing extra space on the right in PROC PLOT is by increasing the X-axis tick range. The following step creates Figure 6.5:

```
%labelit(prin2, prin1, mammal, data=teeth, opts=haxis=-4 to 4)
```

The OPTS= option adds options to the PLOT statement in PROC PLOT. The HAXIS=–4 TO 4 option adds additional space to the right of the graph to accommodate the labels. Be aware that when you specify the HAXIS= option (or the VAXIS= option for the vertical axis), PROC PLOT discards all points that have a coordinate that extends beyond the minimum or maximum axis values.

Figure 6.5 Adding More Room for the Labels



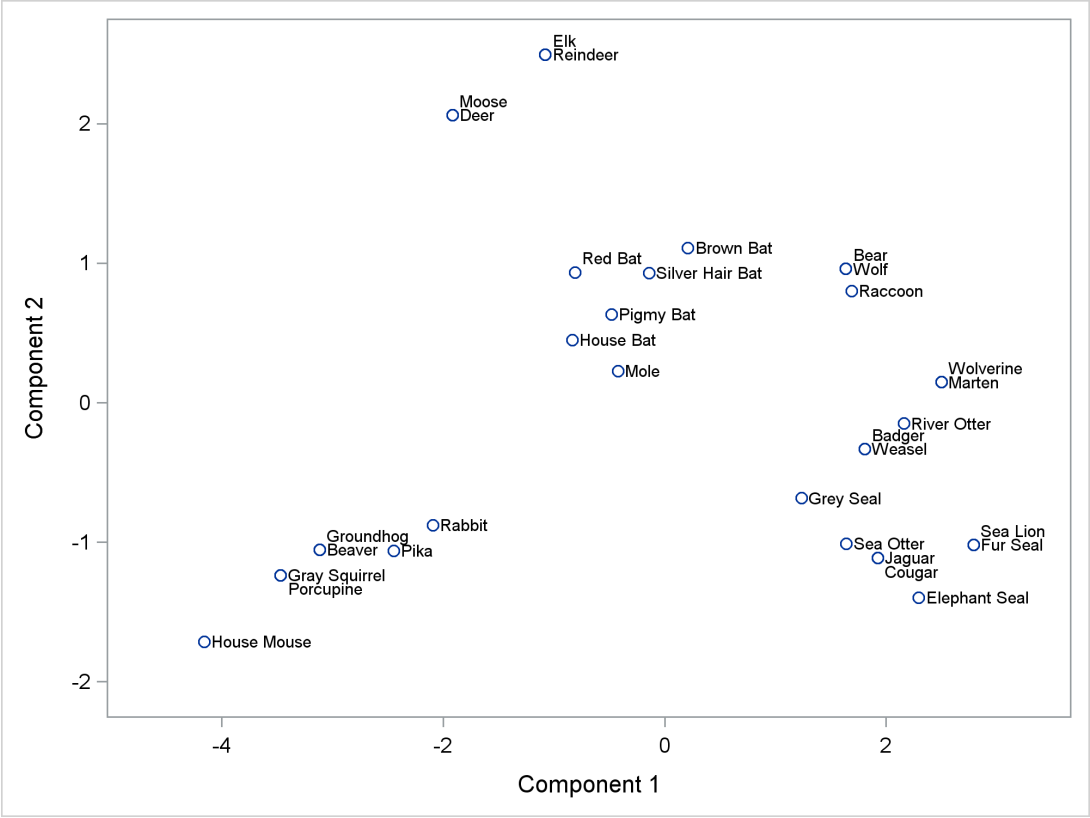
Now the Fur Seal label is to the right of the point.

The PLACE= option enables you to have complete control over the positions that the macro tries. The following step, which creates Figure 6.6, does not produce elegant results for these data, but it illustrates a simple example of the label placement control syntax:

```
%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=(h=2 * s=right * v=0 to 2 by alt))
```

The starting points for label placement are S=RIGHT (the label starts on the symbol and continues to the right), S=LEFT (the label starts to the left of the symbol and ends on the symbol), S=CENTER (the label is centered around the symbol). Labels can be shifted horizontally (by the H= option), vertically (by the V= option), and they can be split onto multiple lines (by the L= option). Combinations of the S=, H=, V=, and L= options are called *placement states*. For example, the placement state S=RIGHT * H=2 starts the label two cells to the right of the symbol, which positions the label after the symbol and a blank space. The placement state S=LEFT * H=–2 ends the label two cells to the left of the symbol, which positions the label before the symbol and a blank space. The syntax **`h=2 * s=right * v=0 to 2 by alt`** creates five placement states that contain five different vertical shifts: H=2 * S=RIGHT * V=0, H=2 * S=RIGHT * V=1, H=2 * S=RIGHT * V=–1, H=2 * S=RIGHT * V=2, H=2 * S=RIGHT * V=–2. The syntax V=0 to 2 BY ALT creates an alternating-sign list of vertical shifts: 0, 1, –1, 2, –2. The asterisk creates every possible combination of values in the expression list. This specification places all the labels to the right of the symbol and enables them to shift up or down as needed to avoid collisions.

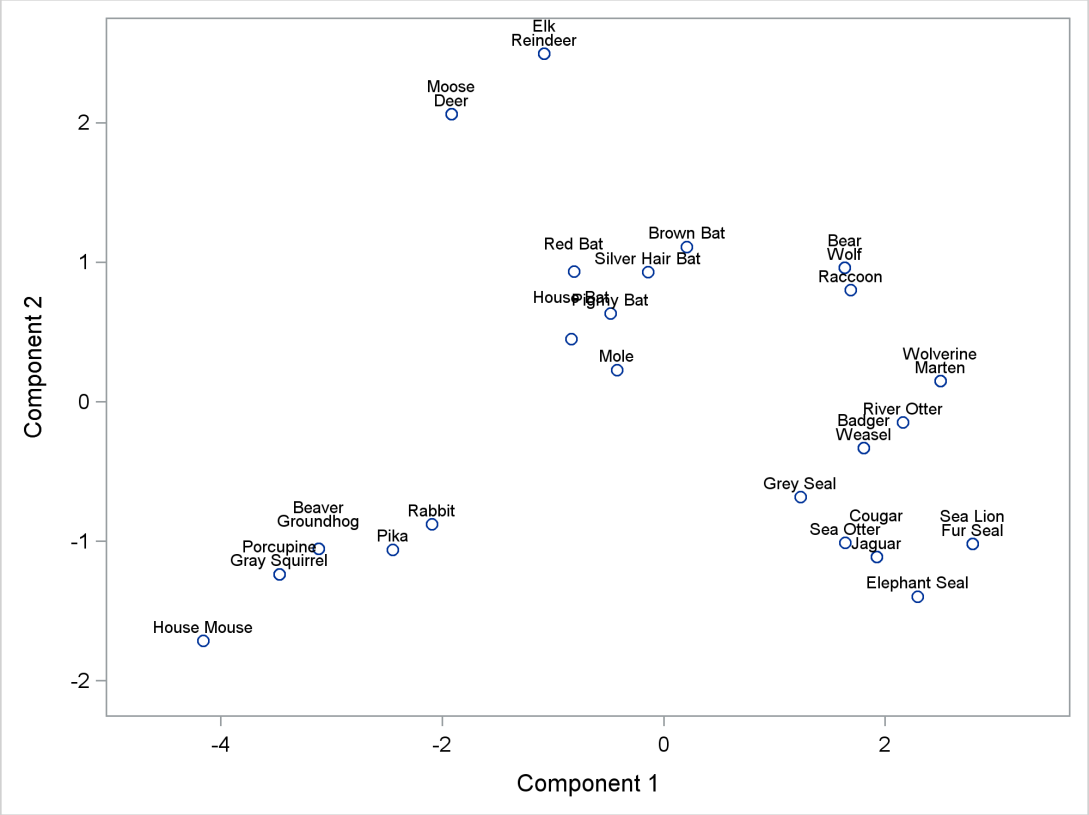
Figure 6.6 All Labels on the Right



The following step creates Figure 6.7, in which all labels are centered above the symbols and shifted up one to three lines:

```
%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=(s=center * v=1 2 3))
```

Figure 6.7 All Labels Centered Above



Again, this specification does not produce elegant results for these data. However, specifications like these can be helpful. When you limit the label placements, it is easier to identify which label corresponds to which point in the more congested parts of the plot. You can use graphs with limited label placements as aids in later steps when you fine-tune graphs with more label placements.

The following step, which creates Figure 6.8, illustrates additional capabilities in specifying placement states:

```
%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))
```

This syntax requests the following:

- labels on the right and left, split onto two lines or not split
- labels centered above or below the symbol, possibly shifted to the right or left, and split onto two lines or not split
- labels on the right or left and vertically shifted, and split onto two lines or not split

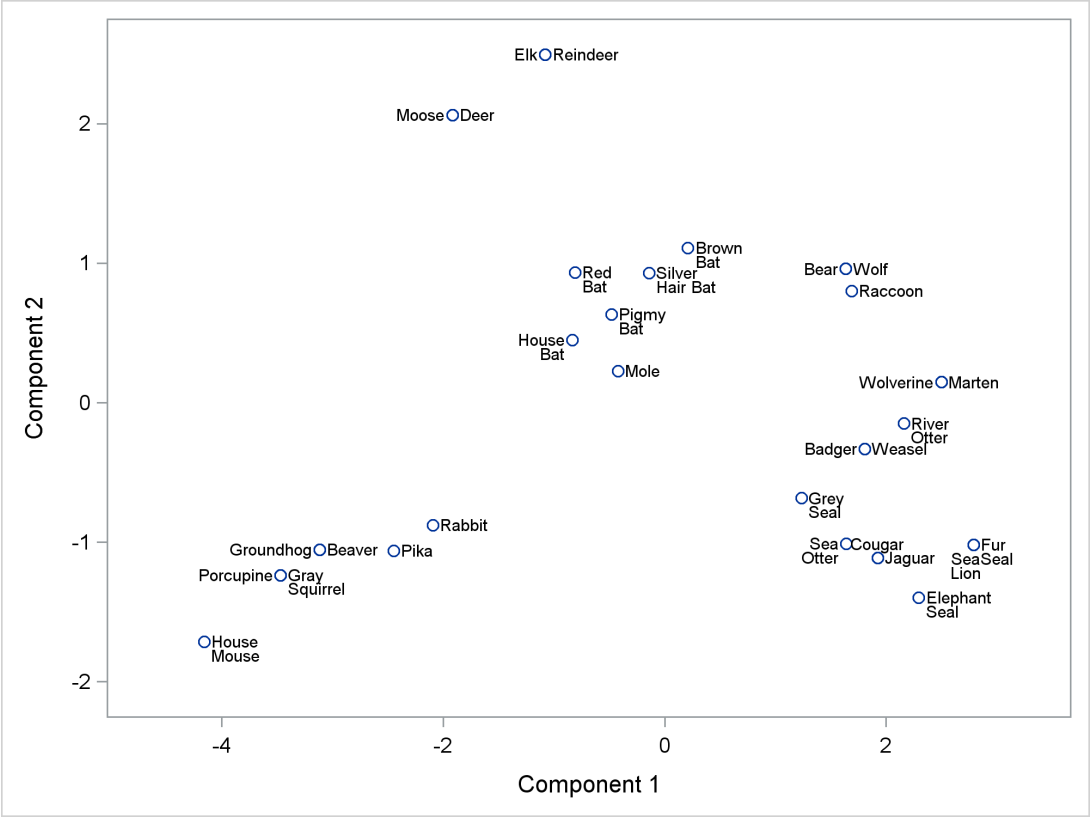
The algorithm tries to use placement states near the beginning of the list if possible.

The asterisk creates every possible combination of values in the expression list; a colon creates only pairwise combinations. The first expression, **`(l=2 1 * (s=right left : h=2 -2))`**, first creates S=RIGHT with H=2 and S=LEFT with H=–2. Then each of these two is combined with L=2 and L=1 to create four placement states. L=2 splits the label onto two lines, and L=1 does not split the label. L=2 was specified first, so two-line placements are preferred over one-line placements. However, splitting a word anywhere except on a blank or punctuation character is considered bad, and the algorithm tries to minimize badness. Hence, the two-line placement states are preferred for the other labels.

The second expression, **`(l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)`**, combines two lines or one line, a centered start that has a vertical shift of 1 to 3 lines (up or down because of the option BY ALT), and horizontal shifts that range from 0 to 5 in both directions for a total of $2 \times 1 \times 6 \times 11 = 132$ placement states.

The third expression, **`(l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)`**, combines two lines or one line, a label to the right or left, and a vertical shift of 1 to 2 lines in both directions, for a total of $2 \times 2 \times 4 = 16$ placement states.

Figure 6.8 Labels That Contain Blanks Are Split



The label placement in Figure 6.8 looks reasonably good. However, some label placements can still be improved. Brown Bat, River Otter, and Fur Seal might look better if they were moved up slightly. If Beaver were moved up and to the left, it would be more obvious that it shares a symbol with Groundhog. It would be better if the Cougar label were shifted to the right, away from the Sea Otter point. Finally, it would be better if the Sea Lion label were shifted to the right and the Fur Seal moved up.

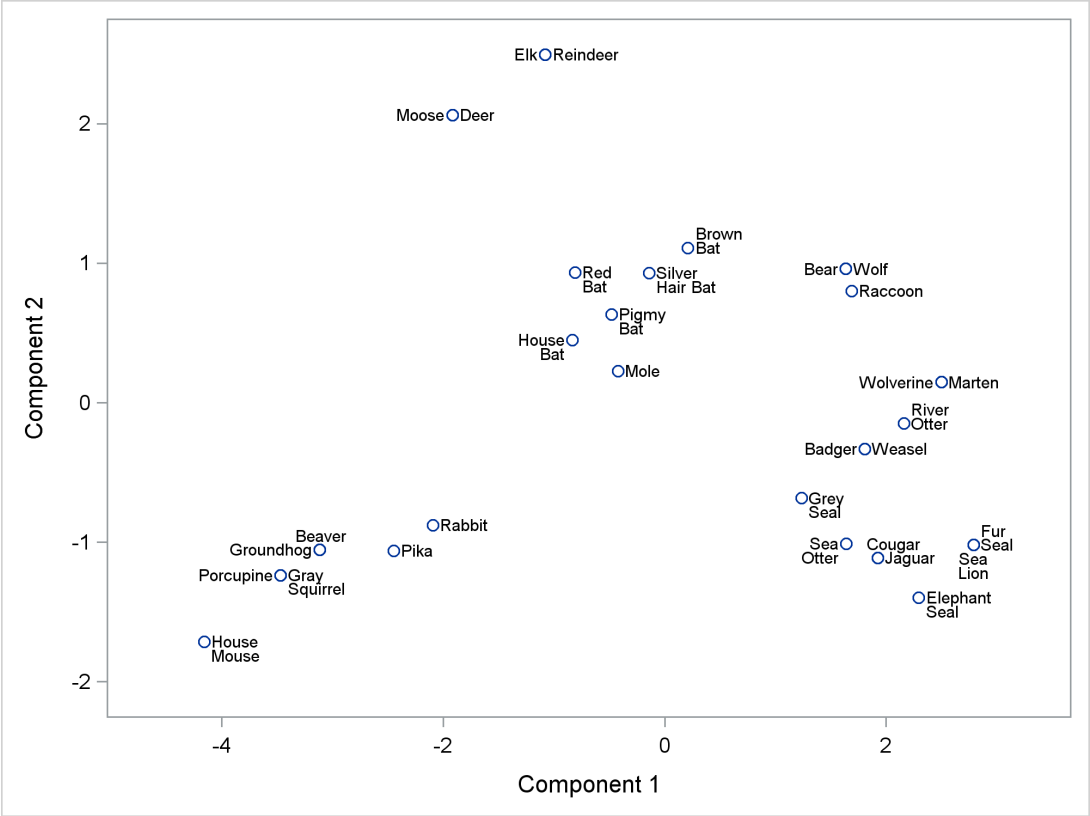
The following step creates Figure 6.9, whose label placements are suitable for use in a paper or presentation (or they can be further refined):

```
%macro tweak;
   if trim(id) in ('Brown Bat', 'River Otter', 'Fur Seal') then vshift + 1;
   if ID = 'Beaver'   then do; vshift + 1;    hshift + -4; end;
   if ID = 'Cougar'   then hshift + 2;
   if ID = 'Sea Lion' then hshift + 1;
%mend;

%labelit(prin2, prin1, mammal, data=teeth, adjust=tweak,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))
```

The ADJUST= argument in the %LabelIt macro enables you to specify the name of a macro (%Tweak in this example) that inserts label adjustment code into the part of the %LabelIt macro that creates the annotation data set.

In the %Tweak macro, horizontal shifts are modified by changing the value of the hShift variable, and vertical shifts are modified by changing the value of the vShift variable. These variables are then used to create the final coordinates for the label segments. The SG annotation data set will not contain either of the shift variables. The unit for horizontal shifts is the approximate width of a character, and the unit for vertical shifts is the approximate height of a character. The statements **`vshift + 1`** and **`hshift + 1`** are sum statements. The statement **`vshift + 1`** is equivalent to **`vshift = vshift + 1`**; it also retains vshift and initializes it to zero. Sum statements are used here to minimize typing; retaining is not required.

Figure 6.9 Tweak Cougar, River Otter, Beaver, Fur Seal, Sea Lion, and Brown Bat



The labels that are adjusted are identified by the ID variable. This variable appears in the annotation data set, but it is not used for annotation. Instead, the annotation data set uses the Label variable. When a label is split onto multiple lines, then the annotation data set contains multiple lines for it, and each line represents a label segment. The Label variable contains each label segment, whereas the ID variable contains the entire label.

The following step displays the final annotation data set as shown in Figure 6.10:

```
proc print noobs data=anno;
run;
```

To summarize and expand this part of the example, recall that you specify a list of placement states. Specify your most-preferred states first and less-preferred states next. The algorithm tries to use your most-preferred states as it tries to resolve collisions. When labels are placed nonoptimally, some labels can be obscured. The algorithm assigns penalties to nonoptimal placement and strives to minimize those penalties. Penalties are assigned when label characters are obscured. The penalties for not displaying the first few characters are higher than the penalties for not displaying the last few characters. Penalties are also assigned for splitting a label on a nonblank or nonpunctuation character and when a label collides with a point. Movements of the label near the point are free; movements farther away are penalized. You can modify any of the penalties by specifying the PENALTIES= option that is documented in PROC PLOT, although that is rarely needed. In contrast, ODS Graphics label-placement algorithms give you no control over preferred placements, nor do they provide differential weighting of the bad things that occur when labels collide.

Figure 6.10 Annotation Data Set

n	Function	DrawSpace	Width	TextSize	ID	Label	Anchor	x1	y1
29	Text	DataValue	100	7	Reindeer	Reindeer	Left	-1.02506	2.49571
30	Text	DataValue	100	7	Elk	Elk	Right	-1.13216	2.49571
31	Text	DataValue	100	7	Deer	Deer	Left	-1.86291	2.06224
32	Text	DataValue	100	7	Moose	Moose	Right	-1.97000	2.06224
1	Text	DataValue	100	7	Brown Bat	Brown	Left	0.26317	1.21031
1	Text	DataValue	100	7	Brown Bat	Bat	Left	0.26317	1.11008
14	Text	DataValue	100	7	Wolf	Wolf	Left	1.68937	0.95908
15	Text	DataValue	100	7	Bear	Bear	Right	1.58227	0.95908
6	Text	DataValue	100	7	Red Bat	Red	Left	-0.75773	0.93458
6	Text	DataValue	100	7	Red Bat	Bat	Left	-0.75773	0.83435
3	Text	DataValue	100	7	Silver Hair Bat	Silver	Left	-0.08829	0.92850
3	Text	DataValue	100	7	Silver Hair Bat	Hair Bat	Left	-0.08829	0.82827
16	Text	DataValue	100	7	Raccoon	Raccoon	Left	1.74176	0.79958
4	Text	DataValue	100	7	Pigmy Bat	Pigmy	Left	-0.42836	0.63206
4	Text	DataValue	100	7	Pigmy Bat	Bat	Left	-0.42836	0.53183
5	Text	DataValue	100	7	House Bat	House	Right	-0.88691	0.45049
5	Text	DataValue	100	7	House Bat	Bat	Right	-0.88691	0.35026
2	Text	DataValue	100	7	Mole	Mole	Left	-0.36829	0.22803
17	Text	DataValue	100	7	Marten	Marten	Left	2.55236	0.14660
19	Text	DataValue	100	7	Wolverine	Wolverine	Right	2.44526	0.14660
21	Text	DataValue	100	7	River Otter	River	Left	2.21228	-0.04960
21	Text	DataValue	100	7	River Otter	Otter	Left	2.21228	-0.14983
18	Text	DataValue	100	7	Weasel	Weasel	Left	1.86083	-0.33141
20	Text	DataValue	100	7	Badger	Badger	Right	1.75373	-0.33141
27	Text	DataValue	100	7	Grey Seal	Grey	Left	1.28987	-0.68499
27	Text	DataValue	100	7	Grey Seal	Seal	Left	1.28987	-0.78522
8	Text	DataValue	100	7	Rabbit	Rabbit	Left	-2.04123	-0.87943
22	Text	DataValue	100	7	Sea Otter	Sea	Right	1.58807	-1.01148
22	Text	DataValue	100	7	Sea Otter	Otter	Right	1.58807	-1.11171
25	Text	DataValue	100	7	Fur Seal	Fur	Left	2.84439	-0.91923
25	Text	DataValue	100	7	Fur Seal	Seal	Left	2.84439	-1.01947
26	Text	DataValue	100	7	Sea Lion	Sea	Center	2.79084	-1.11970
26	Text	DataValue	100	7	Sea Lion	Lion	Center	2.79084	-1.21993
9	Text	DataValue	100	7	Beaver	Beaver	Left	-3.34771	-0.95470
10	Text	DataValue	100	7	Groundhog	Groundhog	Right	-3.16922	-1.05493
7	Text	DataValue	100	7	Pika	Pika	Left	-2.39269	-1.06101
23	Text	DataValue	100	7	Jaguar	Jaguar	Left	1.97844	-1.11384
24	Text	DataValue	100	7	Cougar	Cougar	Center	2.06769	-1.01361
11	Text	DataValue	100	7	Gray Squirrel	Gray	Left	-3.41359	-1.23651
11	Text	DataValue	100	7	Gray Squirrel	Squirrel	Left	-3.41359	-1.33674
13	Text	DataValue	100	7	Porcupine	Porcupine	Right	-3.52068	-1.23651
28	Text	DataValue	100	7	Elephant Seal	Elephant	Left	2.34936	-1.39702
28	Text	DataValue	100	7	Elephant Seal	Seal	Left	2.34936	-1.49725
12	Text	DataValue	100	7	House Mouse	House	Left	-4.10512	-1.71452
12	Text	DataValue	100	7	House Mouse	Mouse	Left	-4.10512	-1.81475

Understanding the %LabelIt Macro

The %Labelit macro is defined as follows:

```
%macro labelit(y, x, d, data=, opts=, adjust=);

ods select none;
options ls=128 ps=60 nonotes;
proc plot data=&data;
   ods output locate=loc;
   plot &y * &x $ &d / list=-1 &opts;
run;

proc reg data=loc;
   ods output parameterestimates=p(keep=estimate);
   model vaxis = vposition;
   model haxis = hposition;
quit;

data anno(drop=vs hs vaxis haxis vshift hshift vposition hposition
          splitadj lines estimate);
   retain n vs hs . Function 'Text' DrawSpace 'DataValue'
          Width 100 TextSize 7 ID;
   if _n_ = 1 then do;
      i = 2; set p point=i; vs = estimate;
      i = 4; set p point=i; hs = estimate;
   end;
   set loc(drop=symbol length penalty rename=(startposition=Anchor));
   by notsorted n;
   if first.n then do;
      splitadj = floor((lines - 1) / 2);
      id = label;
      end;
   if last.n or not first.n; /* one line or when split, skip the first */
   vshift + splitadj;
   splitadj + -1;
   if anchor eq 'Right' and hshift ge  2 then hshift + -1.25;
   if anchor eq 'Left'  and hshift le -2 then hshift +  1.25;
   if anchor ne 'Center' then Anchor = ifc(anchor = 'Left', 'Right', 'Left');
   %if &adjust ne %then %do; %&adjust %end;
   x1 = haxis + hs * hshift;
   y1 = vaxis + vs * vshift;
run;

options notes;
ods select all;
proc sgplot data=&data sganno=anno;
   scatter y=&y x=&x;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
%mend;
```

The macro begins by specifying a PROC PLOT step:

```
ods select none;
options ls=128 ps=60 nonotes;
proc plot data=&data;
   ods output locate=loc;
   plot &y * &x $ &d / list=-1 &opts;
run;
```

All output from this step is suppressed by the ODS SELECT NONE statement, but an output data set that contains the label placements is created. Because PROC PLOT makes printer plots, the page size and line size system options affect the graph. The graph is constructed by using a line size of 128 and a page size of 60. You can modify these numbers. For example, if you want to use larger fonts, you might need to make those numbers smaller. You can use the OPTS= macro option to provide the &Opts macro variable and add more options to the PLOT statement.

The macro continues by specifying a PROC REG step:

```
proc reg data=loc;
   ods output parameterestimates=p(keep=estimate);
   model vaxis = vposition;
   model haxis = hposition;
quit;
```

This step creates a data set p, which contains the regression coefficients necessary to obtain the X and Y coordinates from the discrete label positions that PROC PLOT produces. You usually do not need to modify this step.

The macro continues by specifying a DATA step:

```
data anno(drop=vs hs vaxis haxis vshift hshift vposition hposition
          splitadj lines estimate);
   retain n vs hs . Function 'Text' DrawSpace 'DataValue'
          Width 100 TextSize 7 ID;
   if _n_ = 1 then do;
      i = 2; set p point=i; vs = estimate;
      i = 4; set p point=i; hs = estimate;
   end;
   set loc(drop=symbol length penalty rename=(startposition=Anchor));
   by notsorted n;
   if first.n then do;
      splitadj = floor((lines - 1) / 2);
      id = label;
      end;
   if last.n or not first.n; /* one line or when split, skip the first */
   vshift   + splitadj;
   splitadj + -1;
   if anchor eq 'Right' and hshift ge  2 then hshift + -1.25;
   if anchor eq 'Left'  and hshift le -2 then hshift +  1.25;
   if anchor ne 'Center' then Anchor = ifc(anchor = 'Left', 'Right', 'Left');
   %if &adjust ne %then %do; %&adjust %end;
   x1 = haxis + hs * hshift;
   y1 = vaxis + vs * vshift;
run;
```

Here, you can add new annotation variables to set the text color, control fonts, and so on. This step initializes the annotation variables, reads in the regression coefficients, reads the label locations, adjusts the locations when labels are split, creates the ID variable that contains the full label, discards the line that contains the full label when labels are split, decreases the space between a symbol and its label when the label is to the right or left, and recodes the PROC PLOT starting position to match the definition of an annotation anchor. Next, if you specified an adjustment macro, that code is inserted by the %IF statement. Finally, the annotation variables x1 and y1, which contain the coordinates, are created. Each coordinate is the sum of the coordinate for the symbol and the product of the shift and the regression coefficient.

The last step creates the graph:

```
options notes;
ods select all;
proc sgplot data=&data sganno=anno;
   scatter y=&y x=&x;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

You will want to modify this step in ways appropriate for your analysis. In particular, you might want to modify the offsets or add additional graphing statements. You might want to move the PROC SGPLOT outside the macro and use the macro only to create the annotation data set. If you do that, you will need to specify the X- and Y-axis variables directly in the SCATTER statement. Similarly, you could move the LS= and PS= options outside. You could then adjust the annotation data set outside the macro. You could instead add macro options to control the line size and provide more flexibility. This version was deliberately made short so that it would be easier to explain and easier for you to modify.

There is no guarantee that combinations of line size, page size, axis options, ODS graph sizes, font sizes, and so on will all work well together. It might take a few tries to get reasonable results. The line size of 128 is 1/5 of the ODS Graphics default width of 640 pixels. The page size of 60 is 1/8 of the ODS Graphics default height of 480 pixels. (A different ratio is used, because label characters are taller than they are wide.) Also, because ODS Graphics uses proportional fonts and PROC PLOT does not, labels usually take up less space in the graph than was allotted for them.

Advanced Usage of the %LabelIt Macro

If you want to take more control over label placement, you might prefer to define the macro like this:

```
%macro labelit(y, x, d, data=, opts=, adjust=);

ods select none;
options nonotes;
proc plot data=&data;
   ods output locate=loc;
   plot &y * &x $ &d / list=-1 &opts;
run;

proc reg data=loc;
   ods output parameterestimates=p(keep=estimate);
   model vaxis = vposition;
   model haxis = hposition;
quit;

data anno(drop=vs hs vaxis haxis vshift hshift vposition hposition
          splitadj lines estimate);
   retain n vs hs . Function 'Text' DrawSpace 'DataValue'
          Width 100 TextSize 7 ID;
   if _n_ = 1 then do;
      i = 2; set p point=i; vs = estimate;
      i = 4; set p point=i; hs = estimate;
   end;
   set loc(drop=symbol length penalty rename=(startposition=Anchor));
   by notsorted n;
   if first.n then do;
      splitadj = floor((lines - 1) / 2);
      id = label;
      end;
   if last.n or not first.n; /* one line or when split, skip the first */
   vshift + splitadj;
   splitadj + -1;
   if anchor eq 'Right' and hshift ge  2 then hshift + -1.25;
   if anchor eq 'Left'  and hshift le -2 then hshift +  1.25;
   if anchor ne 'Center' then Anchor = ifc(anchor = 'Left', 'Right', 'Left');
   %if &adjust ne %then %do; %&adjust %end;
   x1 = haxis + hs * hshift;
   y1 = vaxis + vs * vshift;
run;

options notes;
ods select all;
%mend;
```

Then you would use it like this:

```
options ls=128 ps=60;

%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))

proc sgplot data=teeth sganno=anno;
   scatter y=prin2 x=prin1;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

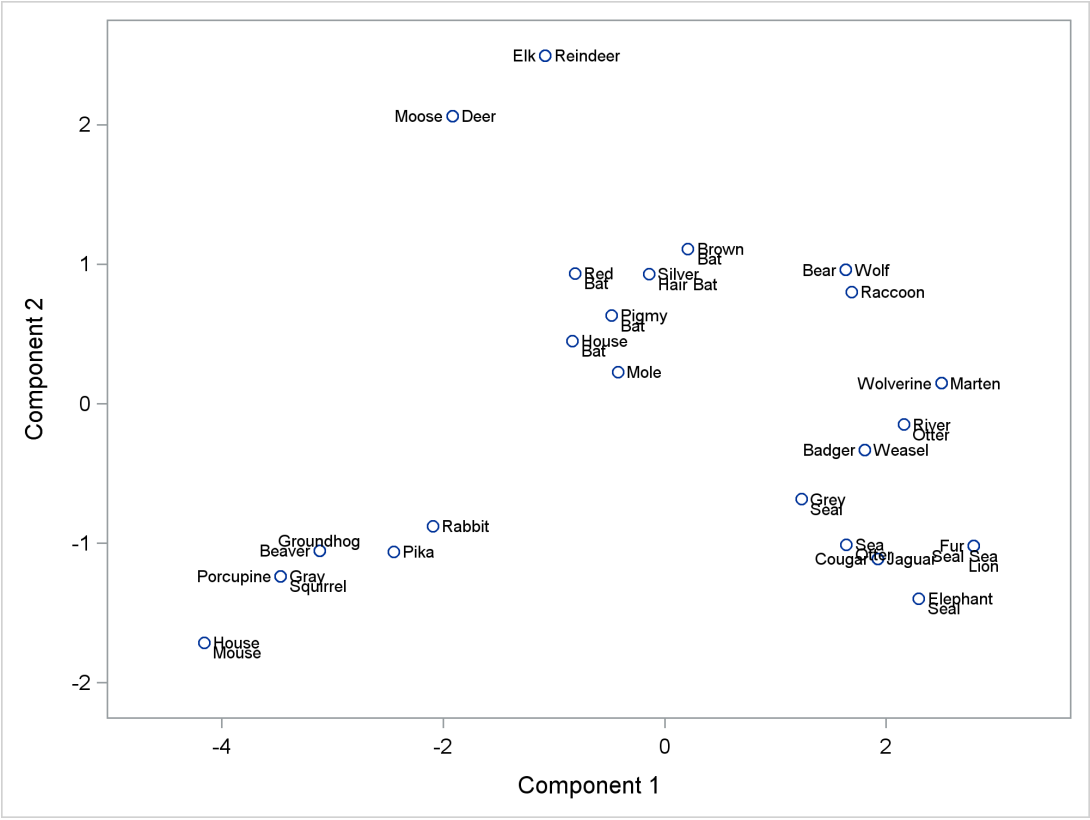
In this case, the line size and page size are set and PROC SGPLOT is run outside the macro.

The following step runs the macro with altered line and page sizes and creates Figure 6.11:

```
options ls=100 ps=80;

%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))

proc sgplot data=teeth sganno=anno;
   scatter y=prin2 x=prin1;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

Here, the ratio of page size to line size is 80 / 100 = 0.8. Previously, the ratio was 60 / 128 = 0.46875. Because of this larger ratio, the lines in Figure 6.11 are too close together. However, this might be desirable when there are a lot of points and parts of the graph are dense.

Figure 6.11 Page Size Too Large

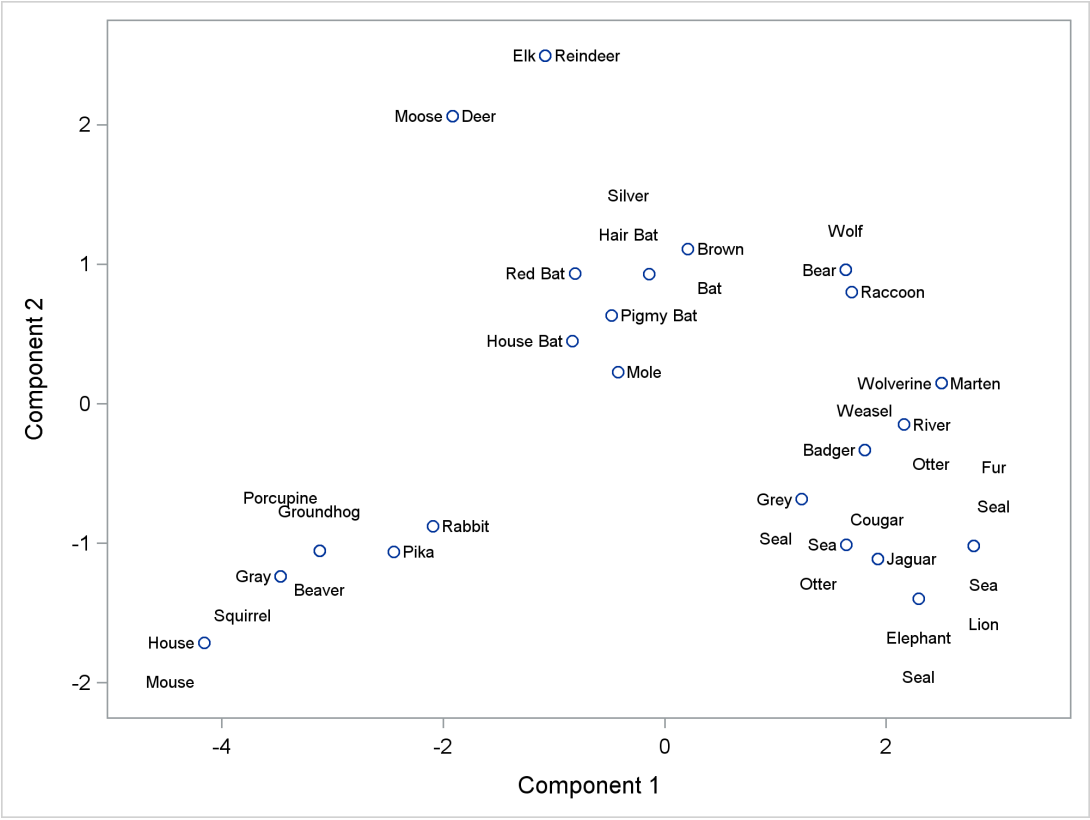


The following step runs the macro with a smaller page size and creates Figure 6.12:

```
options ls=100 ps=32;

%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))

proc sgplot data=teeth sganno=anno;
   scatter y=prin2 x=prin1;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

Here, the ratio of page size to line size is 32 / 100 = 0.32. Because of this smaller ratio, the lines in Figure 6.12 are too far apart.

Figure 6.12 Page Size Too Small

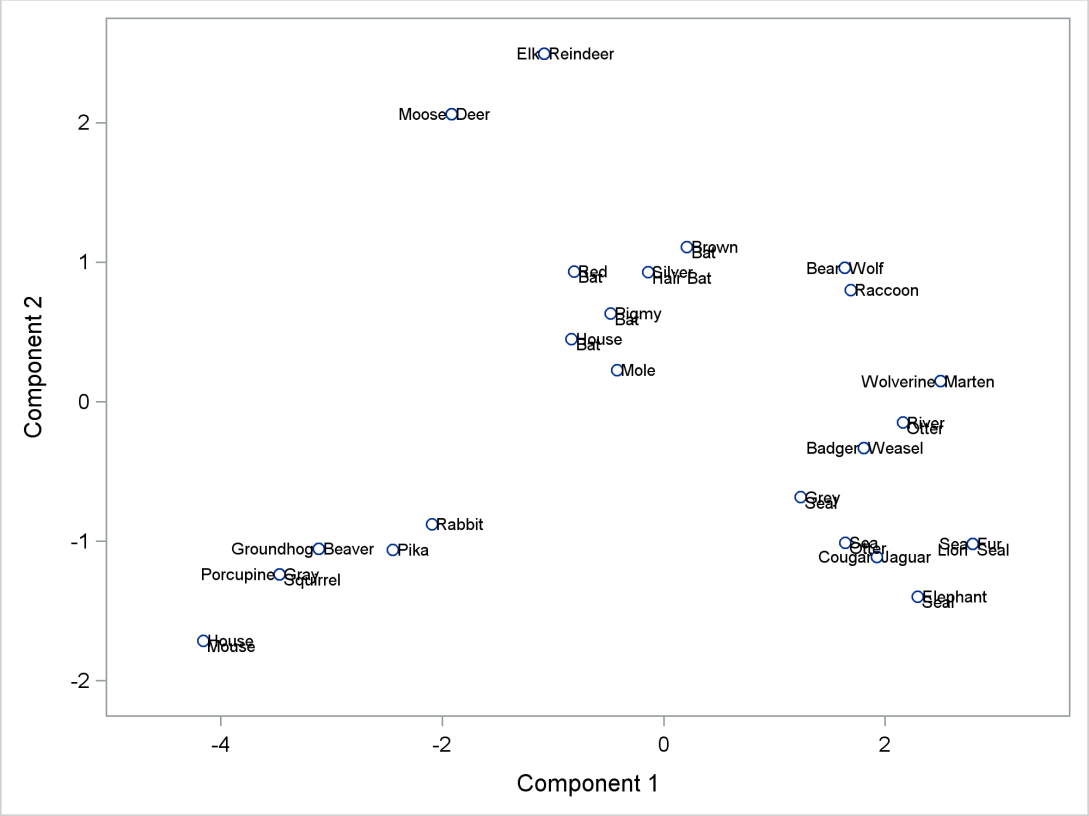


The following step runs the macro with larger line and page sizes and creates Figure 6.13:

```
options ls=256 ps=120;

%labelit(prin2, prin1, mammal, data=teeth,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))

proc sgplot data=teeth sganno=anno;
   scatter y=prin2 x=prin1;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

Here, the ratio of page size to line size is 120 / 256 = 0.46875. Although the ratio matches the original ratio, there are many more cells. With this many cells, the results in Figure 6.13 are not very pleasing.

Figure 6.13 Too Many Cells



In order for this approach to work, you need a closer correspondence between the cells that are defined by line size and page size and what you would see if you filled the graph with characters. Values close to LS=128 and PS=60 should work well when you use 7-pixel fonts. Slightly increasing the numbers can sometimes help reduce collisions. Choose larger values for smaller fonts.

The following step creates a new data set:

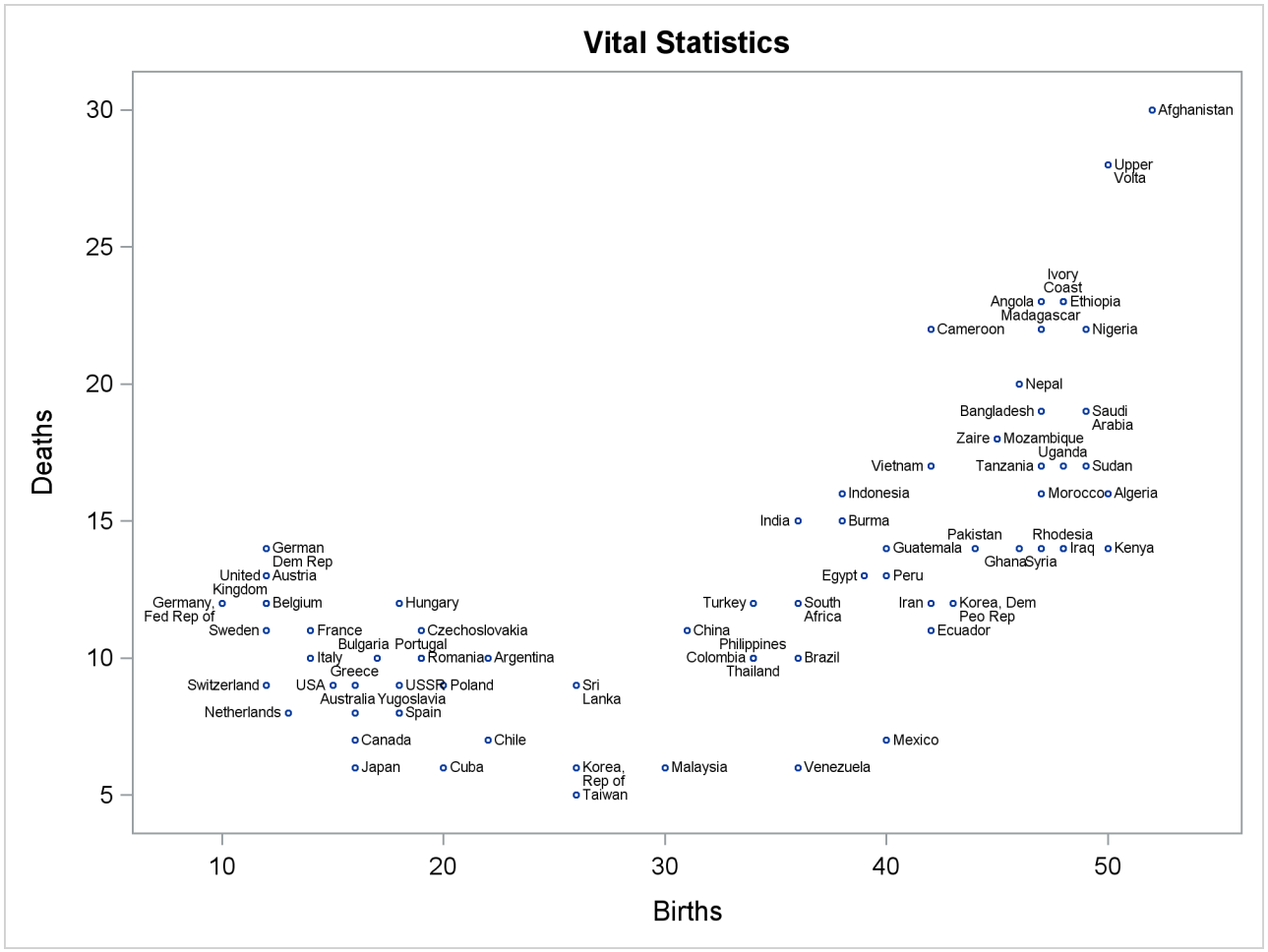
```
title 'Vital Statistics';
data vital;
   input country $ 1-20 Births Deaths;
   datalines;
USA                  15  9
Afghanistan          52 30
Algeria              50 16
Angola               47 23
Argentina            22 10
Australia            16  8

   ... more lines ...

Yugoslavia           18  8
Zaire                45 18
;
```

This data set has more values, and the graph has more congestion. The following steps change the text size from 7 to 5 pixels and add more cells to minimize collisions:

```
options ls=200 ps=80;

%macro tweak;
   TextSize = 5;
%mend;

%labelit(deaths, births, country, data=vital, adjust = tweak,
     opts=place=((l=2 1 * (s=right left : h=2 -2))
                 (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                 (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt))
          haxis=5 to 55 by 5)

proc sgplot data=vital sganno=anno;
   scatter y=deaths x=births / markerattrs=(size=3px);
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.08 offsetmax=0.08;
run;
```

The results are displayed in Figure 6.14.

Figure 6.14 Smaller Fonts

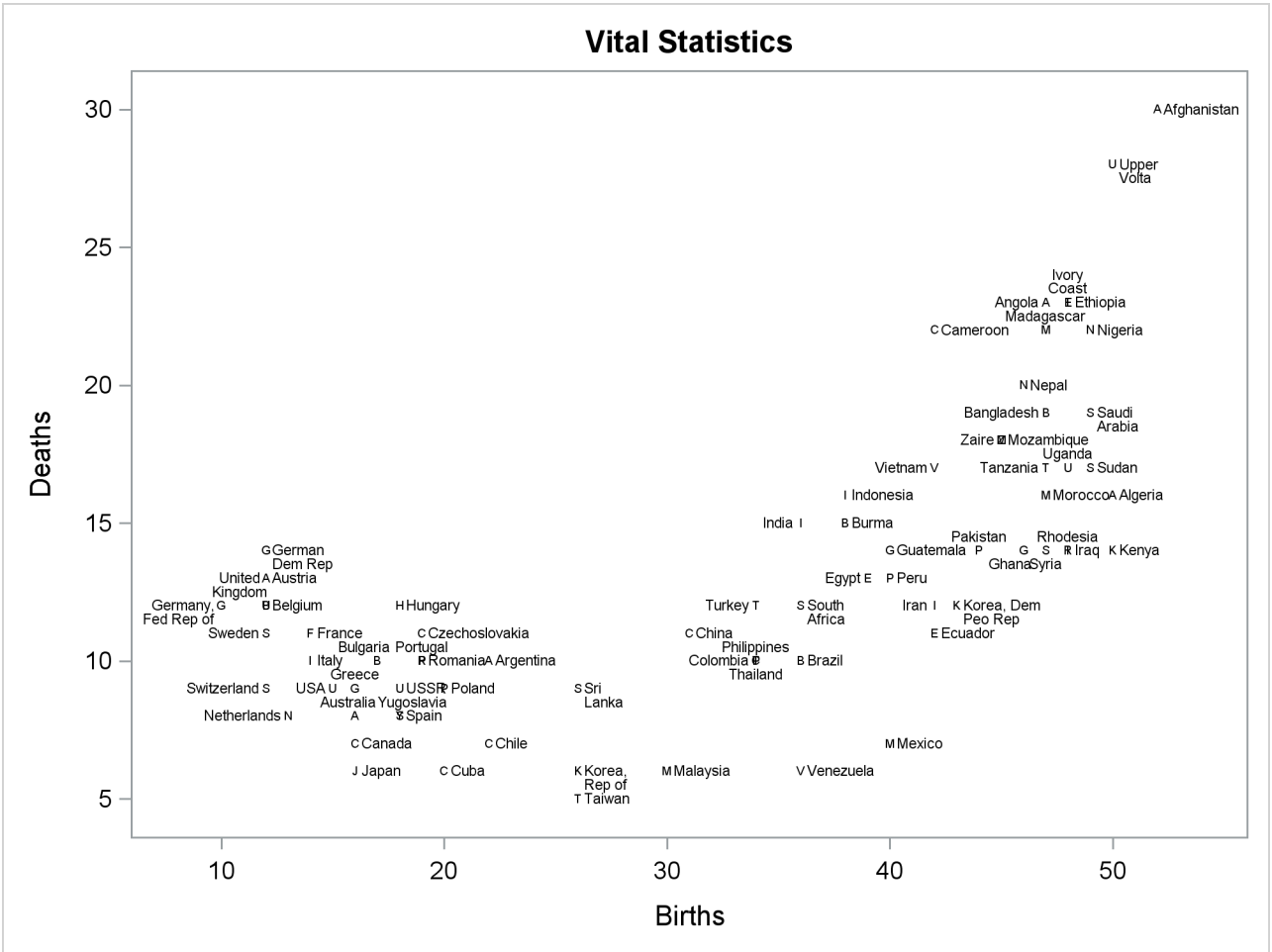


You can also set the marker to the first character of the label as follows:

```
proc sgplot data=vital sganno=anno;
   scatter y=deaths x=births / markercharattrs=(size=3px) markerchar=country;
   format country $1.;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.08 offsetmax=0.08;
run;
```

The MARKERCHAR= option sets the marker variable. Markers can contain more than one character, so a format is specified to use only the first character. The MARKERCHARATTRS= option sets the marker size to 3 pixels. The results are displayed in Figure 6.15. These results are not maximally elegant, but a graph like this can help you understand marker and label associations as you fine-tune a graph. Even in the parts of the graph that contain label congestion and coincident points, you can easily map every label to its symbol.

Figure 6.15 Marker Matches First Character of Label

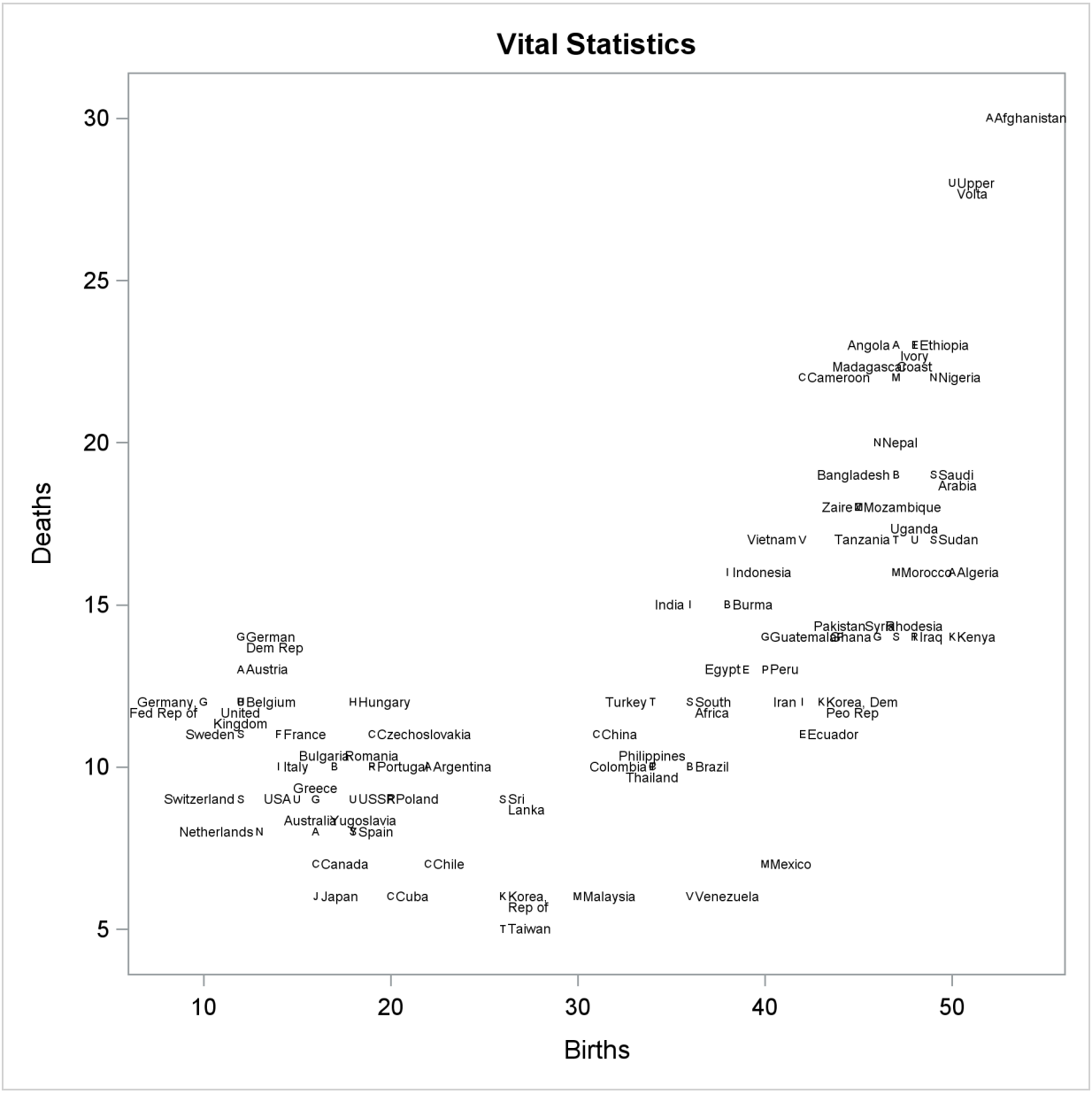


You can make the graph larger as follows:

```
options ls=210 ps=110;

%macro tweak;
   TextSize = 5;
%mend;

%labelit(deaths, births, country, data=vital, adjust = tweak,
     opts=place=((l=2 1 * (s=right left : h=2 -2))
                 (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                 (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt))
          haxis=5 to 55 by 5)

ods graphics on / height=640px width=640px;
proc sgplot data=vital sganno=anno;
   scatter y=deaths x=births / markercharattrs=(size=3px) markerchar=country;
   format country $1.;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.08 offsetmax=0.08;
run;
```

The results are displayed in Figure 6.16.

Figure 6.16 Larger Graph

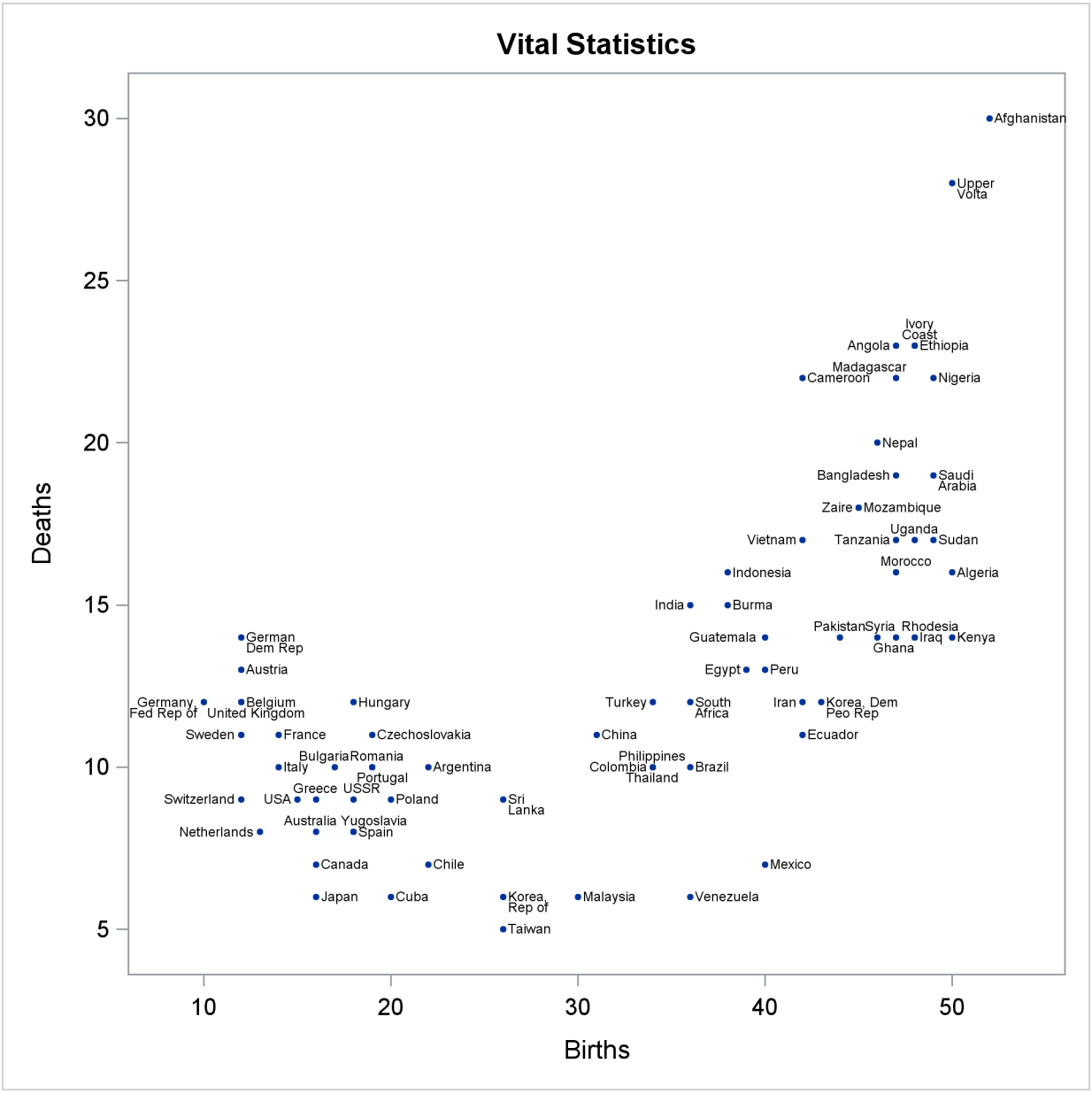


You can create a final graph by adjusting a few label placements as follows:

```
%macro tweak;
   TextSize = 5;
   if ID =: 'Ivory' then do; hshift +  1; vshift +  3;                    end;
   if ID =: 'Moroc' then do; hshift + -4; vshift +  1;                    end;
   if ID =: 'Portu' then do; hshift + -4; vshift + -1;                    end;
   if ID =: 'USSR ' then do; hshift + -3; vshift +  1;                    end;
   if ID =: 'Ghana' then do; hshift +  4; vshift + -1; anchor = 'Center'; end;
   if ID =: 'Guate' then do; hshift + -2;              anchor = 'Right';  end;
   if ID =: 'Rhode' then     hshift +  3;
   if ID =: 'Yugos' then     hshift +  2;
   if ID =: 'Roman' then     hshift +  1;
   if ID =: 'United K' and label =: 'U' then do; label = id; hshift + 3;  end;
   if ID =: 'United K' and label =: 'K' then return;
%mend;

%labelit(deaths, births, country, data=vital, adjust = tweak,
     opts=place=((l=2 1 * (s=right left : h=2 -2))
                 (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                 (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt))
          haxis=5 to 55 by 5)

proc sgplot data=vital sganno=anno;
   scatter y=deaths x=births / markerattrs=(size=3px symbol=circlefilled);
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.08 offsetmax=0.08;
run;

ods graphics on / reset=all;
```

The graph in Figure 6.16, which uses the first country letter as the marker, is handy in developing this code, because it makes it clear which label goes with which marker. The results are displayed in Figure 6.17.

Figure 6.17 Final Graph

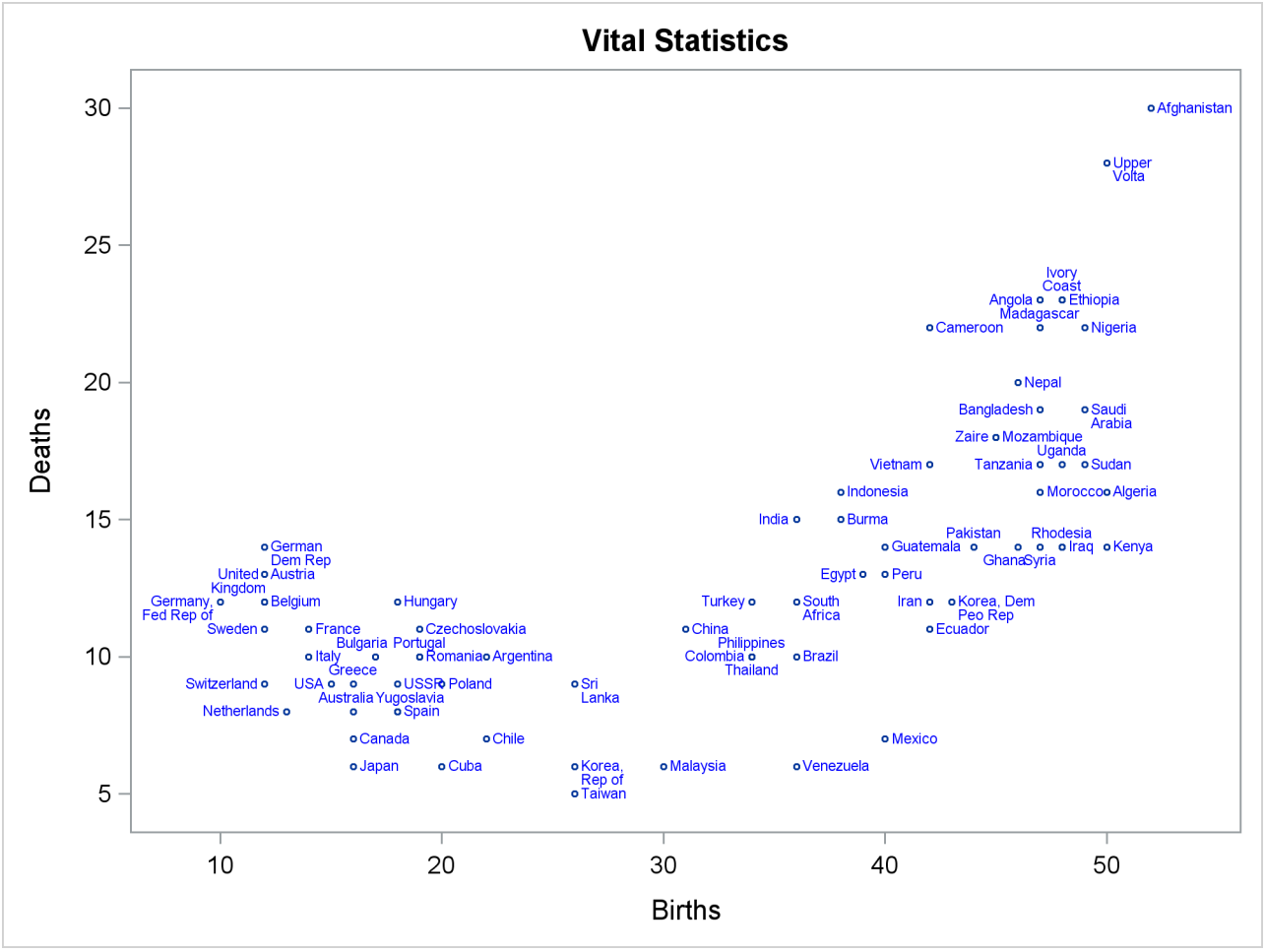


You can add annotation variables that were not defined in the macro. The following steps change the label color:

```
options ls=200 ps=80;

%macro tweak;
   TextSize  = 5;
   TextColor = 'Blue';
%mend;

%labelit(deaths, births, country, data=vital, adjust=tweak,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt))
              haxis=5 to 55 by 5)

proc sgplot data=vital sganno=anno;
   scatter y=deaths x=births / markerattrs=(size=3px);
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.08 offsetmax=0.08;
run;
```

The results are displayed in Figure 6.18.

Figure 6.18 Text Color Changes

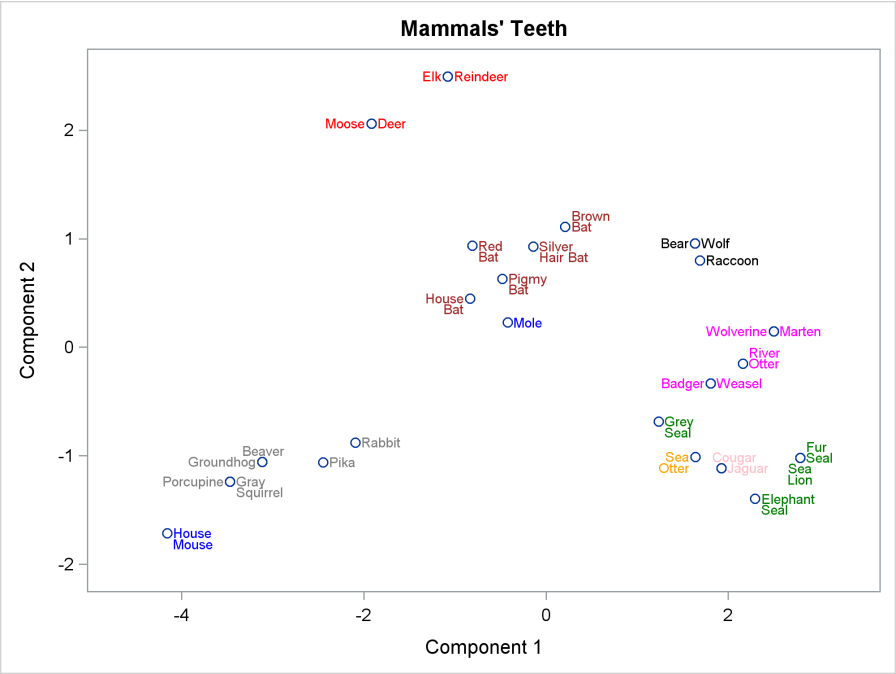


You can conditionally change the text colors too:

```
title 'Mammals'' Teeth';
options ls=128 ps=60;

%macro tweak;
   if trim(id) in ('Reindeer', 'Elk', 'Deer', 'Moose')
      then TextColor = 'Red     ';
   if index(id, 'Bat')
      then TextColor = 'Brown';
   if trim(id) in ('Wolf', 'Bear', 'Raccoon')
      then TextColor = 'Black';
   if trim(id) in ('Mole' 'House Mouse')
      then TextColor = 'Blue';
   if trim(id) in ('Marten', 'Wolverine', 'River Otter', 'Weasel', 'Badger')
      then TextColor = 'Magenta';
   if trim(id) in ('Grey Seal', 'Elephant Seal', 'Fur Seal', 'Sea Lion')
      then TextColor = 'Green';
   if trim(id) in ('Sea Otter')
      then TextColor = 'Orange';
   if trim(id) in ('Rabbit', 'Beaver', 'Groundhog', 'Pika',
                   'Gray Squirrel', 'Porcupine')
      then TextColor = 'Grey';
   if trim(id) in ('Jaguar', 'Cougar')
      then TextColor = 'Pink';

   if trim(id) in ('Brown Bat', 'River Otter', 'Fur Seal') then vshift + 1;
   if ID = 'Beaver'   then do; vshift + 1;    hshift + -4; end;
   if ID = 'Cougar'   then hshift + 2;
   if ID = 'Sea Lion' then hshift + 1;
%mend;

%labelit(prin2, prin1, mammal, data=teeth, adjust=tweak,
         opts=place=((l=2 1 * (s=right left : h=2 -2))
                     (l=2 1 * s=center * v=1 to 3 by alt * h=0 to 5 by alt)
                     (l=2 1 * (s=right left : h=2 -2) * v=1 to 2 by alt)))

proc sgplot data=teeth sganno=anno;
   scatter y=prin2 x=prin1;
   yaxis offsetmin=0.05 offsetmax=0.05;
   xaxis offsetmin=0.1  offsetmax=0.1;
run;
```

The results are displayed in Figure 6.19.

Figure 6.19 Conditional Color Changes



Changing How Vectors Are Displayed

Double Click for Example Code

This example uses the PRINQUAL procedure to perform a multidimensional preference (MDPREF) analysis. The results are displayed in a graph that has vectors emanating from the origin. Subsequent steps show how to display shorter vectors and change the positions of the vector labels. The following step reads the data:

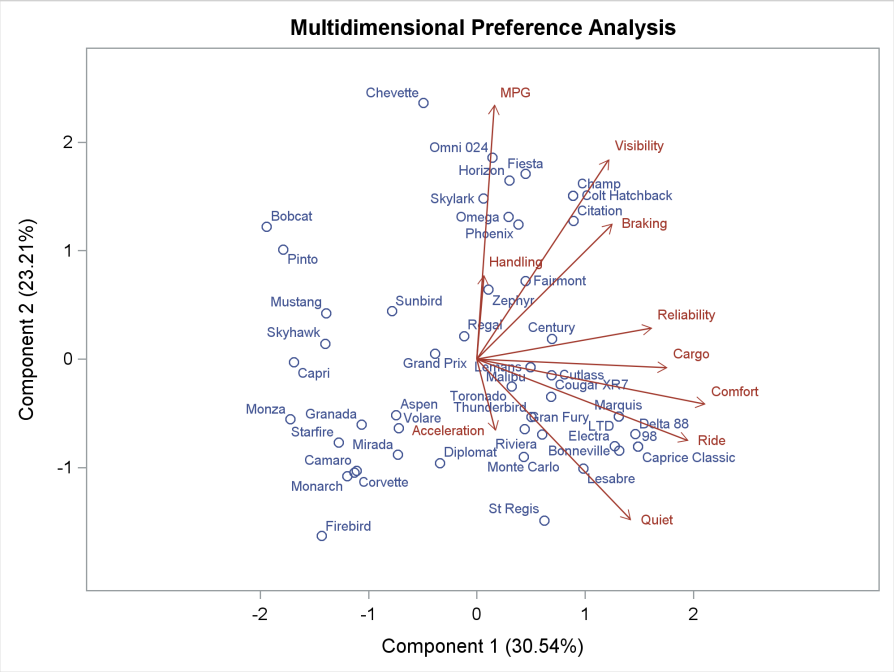
```
title 'Ratings for Automobiles Manufactured in 1980';

data cars;
   input Origin $ 1-8 Make $ 10-19 Model $ 21-36
         (MPG Reliability Acceleration Braking Handling Ride
          Visibility Comfort Quiet Cargo) (1.);
   datalines;
GMC       Buick      Century          3334444544
GMC       Buick      Electra          2434453555
GMC       Buick      Lesabre          2354353545

   ... more lines ...

GMC       Pontiac    Phoenix          4155554415
GMC       Pontiac    Sunbird          3134533234
;
```

The following step performs an MDPREF analysis and creates Figure 6.20:

```
ods graphics on;

proc prinqual data=cars mdpref replace out=b(keep=_: prin:);
   ods output mdprefplot=m;
   transform ide(mpg -- cargo);
   id model;
run;
```

The ODS OUTPUT statement creates a data set from the data object that is used to create the MDPREF plot. (The data object contains the coordinates of the points and vectors in the graph.)

Figure 6.20 Multidimensional Preference Analysis

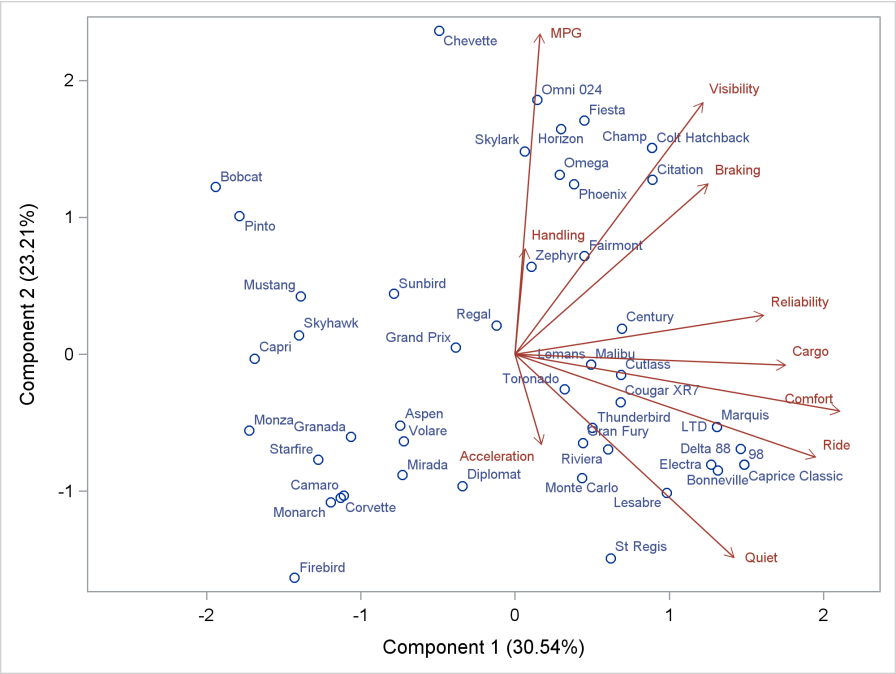


The following step uses PROC SGPLOT and the ODS output data set from PROC PRINQUAL to create the MDPREF plot:

```
proc sgplot data=m noautolegend;
   scatter y=prin2 x=prin1 / datalabel=idlab1 datalabelattrs=graphdata1;
   vector y=vec2 x=vec1 / datalabel=label2var lineattrs=graphdata2
                          datalabelattrs=graphdata2;
   xaxis offsetmax=0.05 offsetmin=0.15;
run;
```

The results are displayed in Figure 6.21 and are similar to Figure 6.20.

Figure 6.21 MDPREF Plot Made by PROC SGPLOT

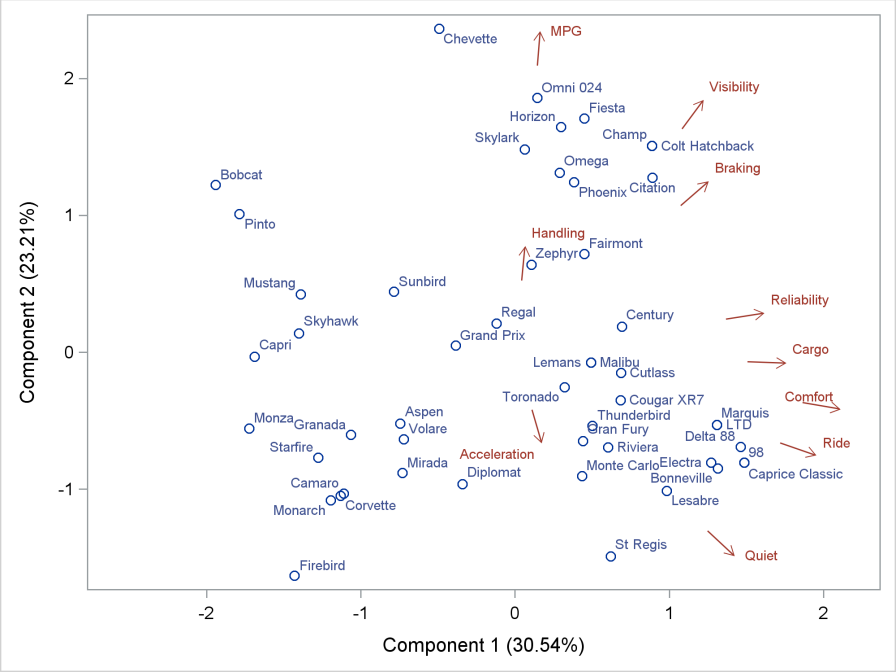


The following step shortens the vectors and displays them:

```
data m2;
   set m;
   if n(vec1, vec2) = 2 then do;
      xo = vec1 - 0.25 / sqrt(1 + (vec2 / vec1) ** 2);
      yo = (vec2 / vec1) * xo;
   end;
run;

proc sgplot data=m2 noautolegend;
   scatter y=prin2 x=prin1 / datalabel=idlab1 datalabelattrs=graphdata1;
   vector y=vec2 x=vec1 / yorigin=yo xorigin=xo
                          datalabel=label2var lineattrs=graphdata2
                          datalabelattrs=graphdata2;
   xaxis offsetmax=0.05 offsetmin=0.15;
run;
```

This step finds a point on the line defined by each vector that is 0.25 data units from the end of the vector (in the direction of the origin). That point is used as the vector origin for each vector. The new origin is based on a slope of $(y_1 - 0)/(x_1 - 0) = y_1/x_1$ and a new X coordinate for the origin of $x_0 = x_1 - 0.25\sqrt{1 + (y_1/x_1)^2}$. Given an intercept of 0, the new Y coordinate for the origin equals the new X coordinate for the origin times the slope. The results are displayed in Figure 6.22.

Figure 6.22 MDPREF Plot with Short Vectors

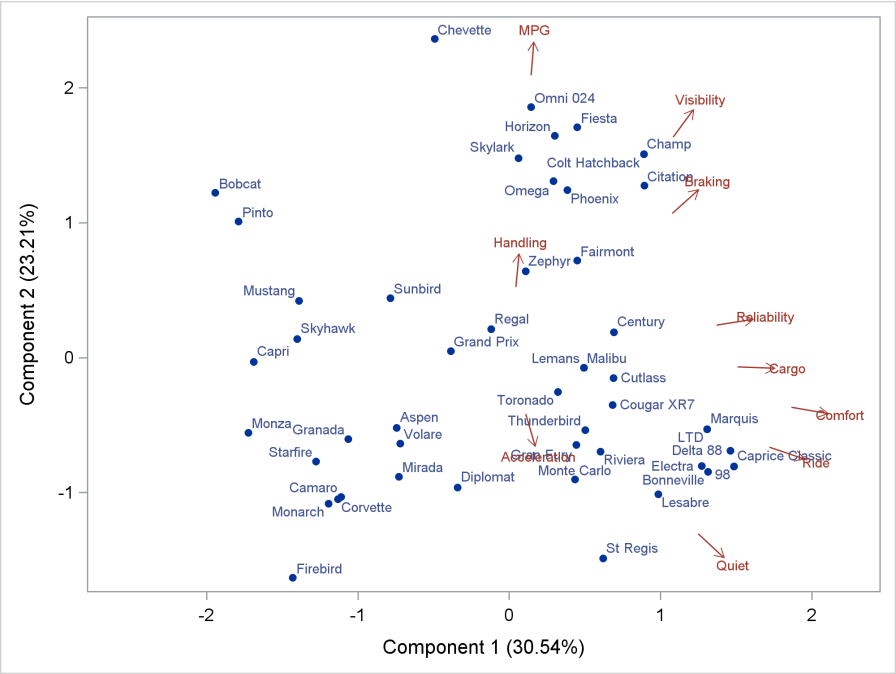


The following step creates the same vectors as the previous step and explicitly positions the center of each vector label 0.08 data units beyond the end of each vector:

```
data m2;
   set m;
   if n(vec1, vec2) = 2 then do;
      xo = vec1 - 0.25 / sqrt(1 + (vec2 / vec1) ** 2);
      yo = (vec2 / vec1) * xo;
      xc = vec1 + 0.08 / sqrt(1 + (vec2 / vec1) ** 2);
      yc = (vec2 / vec1) * xc;
   end;
run;

proc sgplot data=m2 noautolegend;
   scatter y=prin2 x=prin1 / datalabel=idlab1 datalabelattrs=graphdata1
                             markerattrs=(symbol=circlefilled size=5px);
   vector  y=vec2 x=vec1   / yorigin=yo xorigin=xo lineattrs=graphdata2;
   scatter y=yc   x=xc     / markerchar=label2var
                             markercharattrs=graphdata2;
   xaxis offsetmax=0.05 offsetmin=0.15;
run;
```

In this PROC SGPLOT step, the VECTOR statement does not produce any labels. Instead a second SCATTER statement centers labels at the specified coordinates. The results are displayed in Figure 6.23.

Figure 6.23 MDPREF Plot with Labels beyond the Vectors

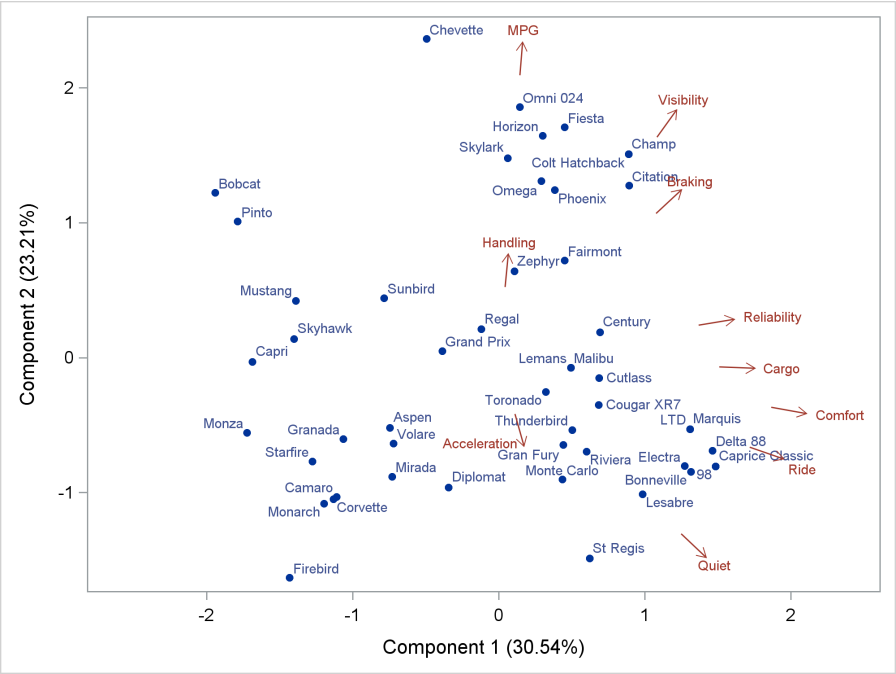


The following step explicitly adjusts the positions of the vector labels in the vicinity of the bottom-right quadrant:

```
data m2;
   set m;
   if n(vec1, vec2) = 2 then do;
      xo = vec1 - 0.25 / sqrt(1 + (vec2 / vec1) ** 2);
      yo = (vec2 / vec1) * xo;
      xc = vec1 + 0.08 / sqrt(1 + (vec2 / vec1) ** 2);
      yc = (vec2 / vec1) * xc;
      if label2var =: 'Rel' then     xc +  .18;
      if label2var =: 'Car' then     xc +  .1;
      if label2var =: 'Com' then     xc +  .15;
      if label2var =: 'Acc' then do; xc + -.32; yc +  0.10; end;
      if label2var =: 'Rid' then do; xc +  .05; yc + -0.05; end;
   end;
run;

proc sgplot data=m2 noautolegend;
   scatter y=prin2 x=prin1 / datalabel=idlab1 datalabelattrs=graphdata1
                             markerattrs=(symbol=circlefilled size=5px);
   vector  y=vec2 x=vec1   / yorigin=yo xorigin=xo lineattrs=graphdata2;
   scatter y=yc   x=xc     / markerchar=label2var
                             markercharattrs=graphdata2;
   xaxis offsetmax=0.05 offsetmin=0.15;
run;
```

The results are displayed in Figure 6.24.

Figure 6.24 MDPREF Plot with Label Placement Control



You could use annotation to place the vectors and labels, but as seen here, you can instead use PROC SGPLOT and data manipulations.

Chapter 7
Advanced Customization of Graphs That Analytical Procedures Produce

Contents

Experienced ODS users know that you can modify table, graph, and style templates and use other styles to modify the output that analytical procedures create. Most ODS users do not know that you can capture, use, and modify the dynamic variables that provide critical pieces of graphs and tables. Because you can capture and use dynamic variables, you can also add SG annotation. The examples in this chapter are long and technical, but once you master them, you can customize every aspect of the graphs that analytical procedures produce.

Changing Dynamic Variables by Using the ODS Document

Double Click for Example Code

This example has several sections that show you how to capture output in an ODS document, replay that output, and modify the output by modifying dynamic variables and templates. This example, along with the sections "Annotating Single-Panel Graphs That Analytical Procedures Produce" on page 217 and "Annotating Multiple-Panel Graphs That Analytical Procedures Produce" on page 228, show you how to modify every aspect of the graphs that are produced by analytical procedures.

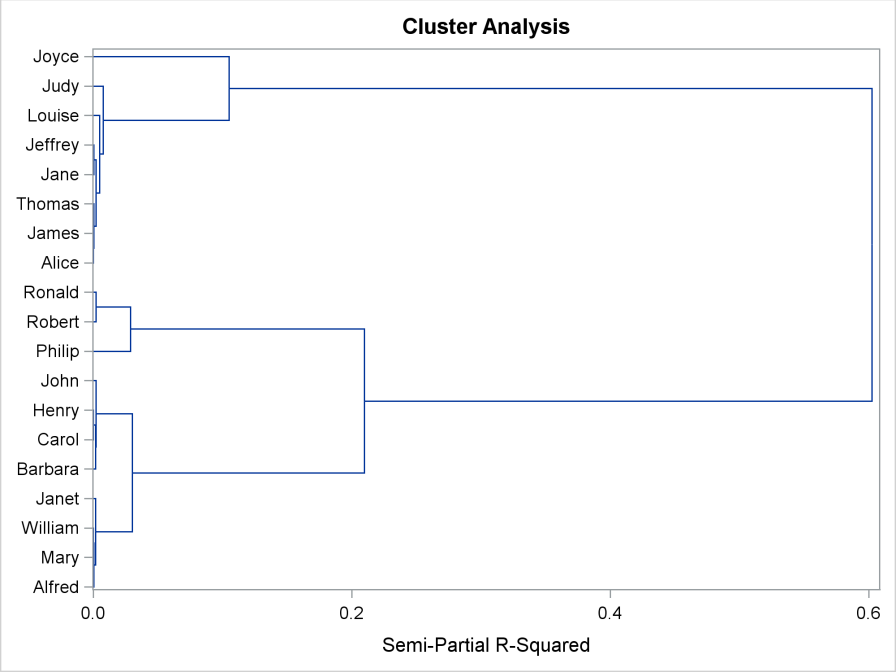
Modifying the Dendrogram in PROC CLUSTER

You can open an ODS document, run one or more procedures, store all the output (tables, graphs, notes, titles, footnotes, and so on) in the document, and then replay some or all of the output in any order that you choose. For example, SAS/STAT documentation uses the ODS document to capture output from the code that is displayed in the documentation and then replay subsets of the output. This process enables SAS documentation to display output, then add explanatory text, then display more output and more text, and so on.

The following step uses the CLUSTER procedure to create a dendrogram and display it in Figure 7.1:

```
ods graphics on;

proc cluster data=sashelp.class method=ward pseudo;
   ods select dendrogram;
   id name;
run;
```

Figure 7.1 Dendrogram



The following steps capture a dendrogram in an ODS document and then replay it:

```
ods document name=MyDoc (write);
proc cluster data=sashelp.class method=ward pseudo;
   ods select dendrogram;
   id name;
run;
ods document close;

proc document name=MyDoc;
   replay cluster\dendrogram;
quit;
```

Both steps produce a dendrogram, and each matches Figure 7.1. The ODS DOCUMENT statement opens an ODS document named MyDoc. Because the WRITE option is specified, a new document is created each time this statement is executed, and any old content is discarded.

The following step lists the contents of the ODS document:

```
proc document name=MyDoc;
   list / levels=all;
quit;
```

The results are displayed in Figure 7.2. This document contains one directory and a graph. There would be more entries in the ODS document if the ODS SELECT statement had not been specified in the PROC CLUSTER step.

Figure 7.2 Dynamic Variables

Listing of: \Work.Mydoc
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Cluster#1	Dir
2	\Cluster#1\Dendrogram#1	Graph

ODS needs the following information to make or replay a graph:

data object	the values that are displayed in the graph
dynamic variables	values (but not data) that are set by the procedure and control aspects of the graph
graph template	the program that describes how the graph is produced
style template	the program that provides colors, fonts, and the general appearance

Both the data object and the dynamic variables are stored in the ODS document. Templates are stored in special files called item stores that SAS provides. You can display the dynamic variables for the dendrogram and store them in a SAS data set as follows:

```
proc document name=MyDoc;
   ods output dynamics=dynamics;
   obdynam \Cluster#1\Dendrogram#1;
quit;

proc print noobs data=dynamics;
run;
```

The path specified in the OBDYNAM statement was copied from the results produced by the LIST statement. The DOCUMENT procedure displays a table of dynamic variables, which is shown in Figure 7.3.

Figure 7.3 Dynamic Variables

Dynamics for: \Work.Mydoc\Cluster#1\Dendrogram#1

Name	Value	Type	Namespace
BYTITLE		Data	
BYLINE		Data	
BYFOOTNOTE		Data	
DH	480PX	Data	
DW	640PX	Data	
XLABEL	Semi-Partial R-Squared	Data	
XR	FALSE	Data	
YR	FALSE	Data	
ORIENT	HORIZONTAL	Data	
___NOBS___	37	Data	
___NOBS___	37	Column	Name
___NOBS___	37	Column	Parent
___NOBS___	37	Column	Height
HVAR	Height	Column	Height

The table in Figure 7.3 is a factoid. Factoids often contain a mix of numeric and character output within a single column. This factoid contains two pairs of columns, where each pair consists of a description (or label) followed by a value. The first pair of columns contains the name of each dynamic variable and its value, and the second pair contains the type of the dynamic variable (specified in the data object or in the column) followed (when relevant) by the data object column name on which it was specified. Some dynamic variables exist in multiple parts of the data object. The dynamic variable __NOBS__ (which is automatically created by ODS) exists in the overall data object and in multiple columns. Figure 7.3 shows that not all dynamic variables have been set to values. This is both common and reasonable.

The ODS output data set made from that table is displayed in Figure 7.4.

Figure 7.4 Output Data Set of Dynamic Variables

Label1	cValue1	nValue1	Label2	cValue2	nValue2
BYTITLE		.	Data		.
BYLINE		.	Data		.
BYFOOTNOTE		.	Data		.
DH	480PX	.	Data		.
DW	640PX	.	Data		.
XLABEL	Semi-Partial R-Squared	.	Data		.
XR	FALSE	.	Data		.
YR	FALSE	.	Data		.
ORIENT	HORIZONTAL	.	Data		.
___NOBS___	37	37.000000	Data		.
___NOBS___	37	37.000000	Column	Name	.
___NOBS___	37	37.000000	Column	Parent	.
___NOBS___	37	37.000000	Column	Height	.
HVAR	Height	.	Column	Height	.

The “Name” column in the table in Figure 7.3 becomes the Label1 variable in the output data set displayed in Figure 7.4. The “Value” column in Figure 7.3 becomes two output data set variables, cValue1 and nValue1. Similarly, “Type” becomes Label2 and “Value” becomes cValue2 and nValue2. The variables nValue1 and nValue2 are numeric, and the variables cValue1 and cValue2 are character. Numeric values are captured in two forms. The actual numeric values are captured in the nValue*n* numeric variables, and formatted numeric values are captured in the cValue*n* character variables. Character values are captured in the cValue*n* variables; the nValue*n* variables have missing values for character variables.

You can replay the graph and explicitly specify that the values of the dynamic variables come from the ODS output data set (rather than from the dynamic variables that are stored in the ODS document):

```
proc document name=MyDoc;
   replay \Cluster#1\Dendrogram#1 / dynamdata=dynamics;
quit;
```

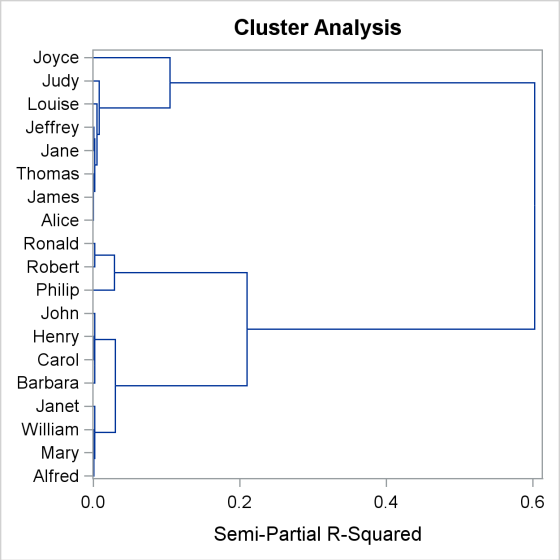
The REPLAY statement replays the graph. The DYNAMDATA= option names the data set that contains the dynamic variables. The results again match Figure 7.1.

PROC CLUSTER determines the height and width of the dendrogram at run time after evaluating the number of rows in the graph. These sizes are stored as dynamic variables. The dynamic variable DH sets the design height and DW sets the design width. You can modify the values of the dynamic variables before you use them to replay the graph. The following steps recreate the dendrogram but by specifying smaller sizes:

```
data dynamics2;
   set dynamics;
   if label1 = 'DH' then cvalue1 = '400PX';
   if label1 = 'DW' then cvalue1 = '400PX';
run;

proc document name=MyDoc;
   replay \Cluster#1\Dendrogram#1 / dynamdata=dynamics2;
quit;
```

The results are displayed in Figure 7.5.

Figure 7.5 Smaller Dendrogram



A few dynamic variable names are constant across templates and procedures. For example, the dynamic variables _BYTITLE_, _BYLINE_, and _BYFOOTNOTE_ are used to display BY lines in graphs when there are BY variables. ODS automatically creates the dynamic variable ___NOBS___ for each column. Most other dynamic variables are ad hoc, although you might see patterns within procedures or graph types. You might need to look at the graph template and GTL documentation to better understand the purpose of the dynamic variable. You can find template names by specifying the following statement before running an analysis procedure:

```
ods trace on;
```

You can display the dendrogram template by copying its name from the SAS log and specifying it in a SOURCE statement as follows:

```
proc template;
   source Stat.Cluster.Graphics.Dendrogram;
quit;
```

The template is displayed in Figure 7.6.

Figure 7.6 Dendrogram Template

```
define statgraph Stat.Cluster.Graphics.Dendrogram;
   notes "Dendrogram";
   dynamic dh dw orient xlabel ylabel hvar xr yr _byline_ _bytitle_
      _byfootnote_;
   begingraph / designheight=DH designwidth=DW;
      entrytitle "Cluster Analysis";
      layout overlay / xaxisopts=(label=XLABEL reverse=XR) yaxisopts=(label=
         YLABEL reverse=YR discreteopts=(tickvaluefitpolicy=none));
         dendrogram nodeid=NAME parentid=PARENT clusterheight=HVAR / orient=
            ORIENT;
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   endgraph;
end;
```

You can see that the dynamic variables xr and yr are used to reverse axes, the variable orient controls vertical and horizontal orientation, variables xlabel and xlabel provide axis labels, and the variable hvar provides the cluster height.

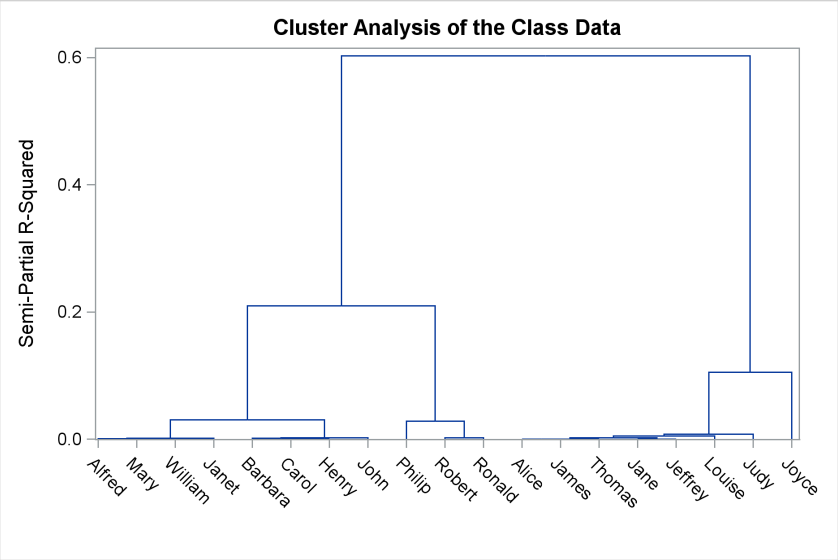
You can modify both the template and the dynamic variables:

```
proc template;
   define statgraph Stat.Cluster.Graphics.Dendrogram;
      notes "Dendrogram";
      dynamic dh dw orient xlabel ylabel hvar xr yr _byline_ _bytitle_
         _byfootnote_;
      begingraph / designheight=DH designwidth=DW;
         entrytitle "Cluster Analysis of the Class Data";
         layout overlay / xaxisopts=(label=' ' reverse=XR) yaxisopts=(label=
            YLABEL reverse=YR discreteopts=(tickvaluefitpolicy=none));
            dendrogram nodeid=NAME parentid=PARENT clusterheight=HVAR /
               orient=vertical;
         endlayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      endgraph;
   end;
quit;
```

```
data dynamics2;
   set dynamics;
   if label1 = 'DH' then cvalue1 = '400PX';
   if label1 = 'DW' then cvalue1 = '600PX';
run;

proc document name=MyDoc;
   replay \Cluster#1\Dendrogram#1 / dynamdata=dynamics2;
quit;
```

The PROC TEMPLATE step modifies the graph title, X-axis label, and the orientation of the graph. The results are displayed in Figure 7.7.

Figure 7.7 Template and Dynamic Variable Modification

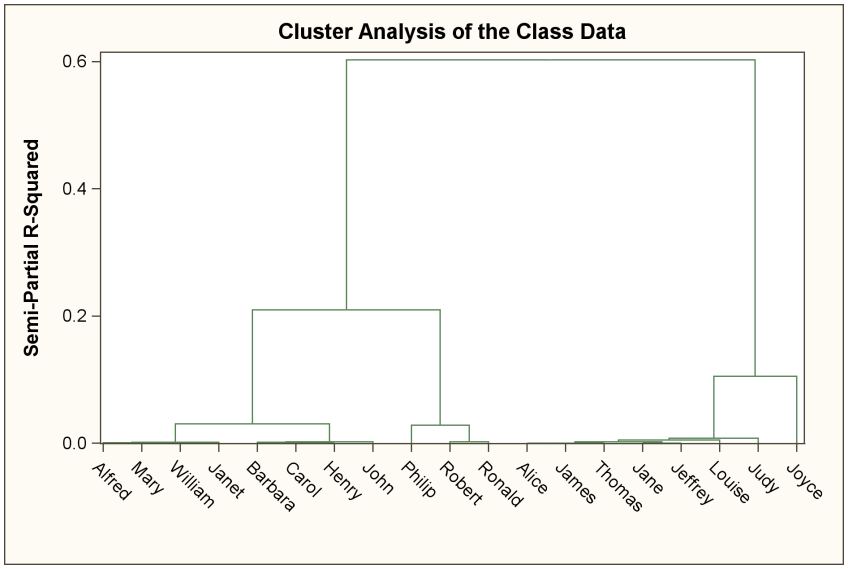


You can also change the ODS style:

```
ods listing style=analysis;
proc document name=MyDoc;
   replay \Cluster#1\Dendrogram#1 / dynamdata=dynamics2;
quit;
```

This step changes the style for the LISTING destination to Analysis. For other destinations, you would substitute for LISTING. The results are displayed in Figure 7.8.

Figure 7.8 Template, Style, and Dynamic Variable Modification



The preceding steps illustrate dynamic variable modification techniques that you can apply to other situations. However, you are not required to use them in this situation. The following step performs almost the same modification of the dendrogram:

```
proc cluster data=sashelp.class method=ward pseudo
             plots=dendrogram(vertical setheight=400 setwidth=600);
   ods select dendrogram;
   id name;
run;
```

The results (except for the title) match Figure 7.8.

The following step restores the default style and deletes the modified graph template:

```
ods listing close;
ods listing;
proc template;
   delete Stat.Cluster.Graphics.Dendrogram / store=sasuser.templat;
run;
```

Modifying the Parameter Estimates Table in PROC REG

The following step captures the output from one PROC REG step and two PROC GLM steps in an ODS document:

```
ods graphics on;
ods document name=MyDoc(write);
proc reg data=sashelp.class;
   model weight = height age / clb;
quit;

proc glm data=sashelp.class;
   model weight = height age / solution;
quit;

proc glm data=sashelp.class;
   class sex;
   model weight = height | sex / solution;
quit;
ods document close;
```

The following step lists the contents of the ODS document:

```
proc document name=MyDoc;
   list / levels=all;
quit;
```

The results are displayed in Figure 7.9.

Figure 7.9 Mydoc Contents

Listing of: \Work.Mydoc
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Reg#1	Dir
2	\Reg#1\MODEL1#1	Dir
3	\Reg#1\MODEL1#1\Fit#1	Dir
4	\Reg#1\MODEL1#1\Fit#1\Weight#1	Dir
5	\Reg#1\MODEL1#1\Fit#1\Weight#1\NObs#1	Table
6	\Reg#1\MODEL1#1\Fit#1\Weight#1\ANOVA#1	Table
7	\Reg#1\MODEL1#1\Fit#1\Weight#1\FitStatistics#1	Table
8	\Reg#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1	Table
9	\Reg#1\MODEL1#1\ObswiseStats#1	Dir
10	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1	Dir
11	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\DiagnosticPlots#1	Dir
12	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\DiagnosticPlots#1\DiagnosticsPanel#1	Graph
13	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\ResidualPlots#1	Dir
14	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\ResidualPlots#1\ResidualPlot#1	Graph
15	\GLM#1	Dir
16	\GLM#1\Data#1	Dir
17	\GLM#1\Data#1\NObs#1	Table
18	\GLM#1\ANOVA#1	Dir
19	\GLM#1\ANOVA#1\Weight#1	Dir
20	\GLM#1\ANOVA#1\Weight#1\OverallANOVA#1	Table
21	\GLM#1\ANOVA#1\Weight#1\FitStatistics#1	Table
22	\GLM#1\ANOVA#1\Weight#1\ModelANOVA#1	Table
23	\GLM#1\ANOVA#1\Weight#1\ModelANOVA#2	Table
24	\GLM#1\ANOVA#1\Weight#1\ParameterEstimates#1	Table
25	\GLM#1\ANOVA#1\Weight#1\ContourFit#1	Graph
26	\GLM#2	Dir
27	\GLM#2\Data#1	Dir
28	\GLM#2\Data#1\ClassLevels#1	Table
29	\GLM#2\Data#1\NObs#1	Table
30	\GLM#2\ANOVA#1	Dir
31	\GLM#2\ANOVA#1\Weight#1	Dir
32	\GLM#2\ANOVA#1\Weight#1\OverallANOVA#1	Table
33	\GLM#2\ANOVA#1\Weight#1\FitStatistics#1	Table
34	\GLM#2\ANOVA#1\Weight#1\ModelANOVA#1	Table
35	\GLM#2\ANOVA#1\Weight#1\ModelANOVA#2	Table
36	\GLM#2\ANOVA#1\Weight#1\ParameterEstimates#1	Table
37	\GLM#2\ANOVA#1\Weight#1\Note#1	Note
38	\GLM#2\ANOVA#1\Weight#1\ANCOVAPlot#1	Graph

This document has multiple procedures, tables, and graphs. The name that you specify in the REPLAY statement needs to match a name that PROC DOCUMENT recognizes in the ODS document. Both of the following steps replay the parameter estimates table from PROC REG:

```
proc document name=MyDoc;
  replay \Reg#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1;
quit;

proc document name=MyDoc;
  replay \Reg\MODEL1\Fit\Weight\ParameterEstimates;
quit;
```

You can often omit the `'#1'`. However, the following steps replay different tables:

```
proc document name=MyDoc;
   replay \GLM#1\ANOVA#1\Weight#1\ModelANOVA#1;
quit;

proc document name=MyDoc;
   replay \GLM\ANOVA\Weight\ModelANOVA;
quit;
```

Each of the two PROC GLM steps produces two ModelANOVA tables, so the numbers are required for reliable differentiation. It is always safest to copy the path from the PROC DOCUMENT listing. In particular, you should not rely on the ODS trace output when you determine the paths to specify.

The following steps output and display the dynamic variables for the parameter estimates table:

```
proc document name=MyDoc;
   ods output dynamics=dynamics;
   obdynam \Reg#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1;
quit;

proc print data=dynamics;
run;
```

The results are displayed in Figure 7.10.

Figure 7.10 Dynamic Variables

Dynamics for:
\Work.Mydoc\Reg#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1

Name	Value	Type
dynglue	0	Data
confidence	95	Data

Obs	Label1	cValue1	nValue1	Label2	nValue2
1	dynglue	0	0	Data	.
2	confidence	95	95.000000	Data	.

Tables usually have fewer dynamic variables than graphs have. This table has two dynamic variables. To better understand the dynamic variables, you can use the following step to display the template:

```
proc template;
   source Stat.REG.ParameterEstimates;
quit;
```

The results of this step are not displayed because the parameter estimates template is large and handles many variations on the parameter estimates table. The dynglue dynamic variable specifies the "glue" that is used to minimize the chances that the *t* statistic and its probability will be separated in a table that splits into panels. The confidence dynamic variable specifies the percentage confidence limit for the confidence limits that span the header. The confidence dynamic variable is used in the template as follows:

```
define clhead;
   text confidence BEST8. %nrstr("%% Confidence Limits");
   end = UpperCL;
   start = LowerCL;
end;
```

Because the confidence dynamic variable is followed by a numeric format, the numeric nValue1 variable (rather than the character variable cValue1) provides the value. The following step shows how to modify the template and the dynamic variable to print words instead of a number:

```
proc template;
   edit Stat.Reg.ParameterEstimates;
      define header clhead;
         text confidence " Percent Confidence Limits";
         end = UpperCL;
         start = LowerCL;
      end;
   end;
quit;

data dynamics2;
   length cvalue1 $ 20;
   set dynamics;
   if  label1 = 'confidence' then do;
      cvalue1 = 'Ninety-Five';
      nvalue1 = .;
      end;
run;

proc document name=MyDoc;
   replay \Reg#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1 /
          dynamdata=dynamics2;
quit;

proc template;
   delete Stat.Reg.ParameterEstimates;
quit;
```

The results are displayed in Figure 7.11.

Figure 7.11 Spanning Header Modification

The REG Procedure
Model: MODEL1
Dependent Variable: Weight

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Ninety-Five Percent Confidence Limits	
Intercept	1	-141.22376	33.38309	-4.23	0.0006	-211.99276	-70.45477
Height	1	3.59703	0.90546	3.97	0.0011	1.67754	5.51652
Age	1	1.27839	3.11010	0.41	0.6865	-5.31473	7.87152

The following steps show how to modify the nValue1 variable along with a corresponding change to the rest of the header:

```
proc template;
   edit Stat.Reg.ParameterEstimates;
      define header clhead;
         text ";Confidence Limits ;(Alpha=" confidence ')';
         end = UpperCL;
         start = LowerCL;
      end;
   end;
quit;

data dynamics2;
   set dynamics;
   if label1 = 'confidence' then nvalue1 = (100 - nvalue1) / 100;
run;

proc document name=MyDoc;
   replay \Reg#1\MODEL1#1\Fit#1\Weight#1\ParameterEstimates#1 /
          dynamdata=dynamics2;
quit;

proc template;
   delete Stat.Reg.ParameterEstimates;
quit;
```

Figure 7.12 Spanning Header Modification

The REG Procedure
Model: MODEL1
Dependent Variable: Weight

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Confidence Limits (Alpha=0.05)	
Intercept	1	-141.22376	33.38309	-4.23	0.0006	-211.99276	-70.45477
Height	1	3.59703	0.90546	3.97	0.0011	1.67754	5.51652
Age	1	1.27839	3.11010	0.41	0.6865	-5.31473	7.87152

You can use the following steps to list all the dynamic variables for all the tables and graphs in the document:

```
proc document name=MyDoc;
   ods output properties=p;
   list / levels=all;
quit;

data _null_;
   set p end=eof;
   if _n_ = 1 then call execute('proc document name=MyDoc;');
   if trim(type) in ('Table', 'Graph') then
      call execute(catx(' ', 'obdynam', path, ';'));
   if eof then call execute('quit;');
run;
```

The results of these steps are not shown. CALL EXECUTE is similar in many ways to the macro language. You can use it to generate the step that is run after the DATA step finishes. You usually use CALL EXECUTE when the statements in the next step depend on the contents of a data set. This example creates a PROC DOCUMENT step along with an OBDYNAM statement for each appropriate entry in the ODS document.

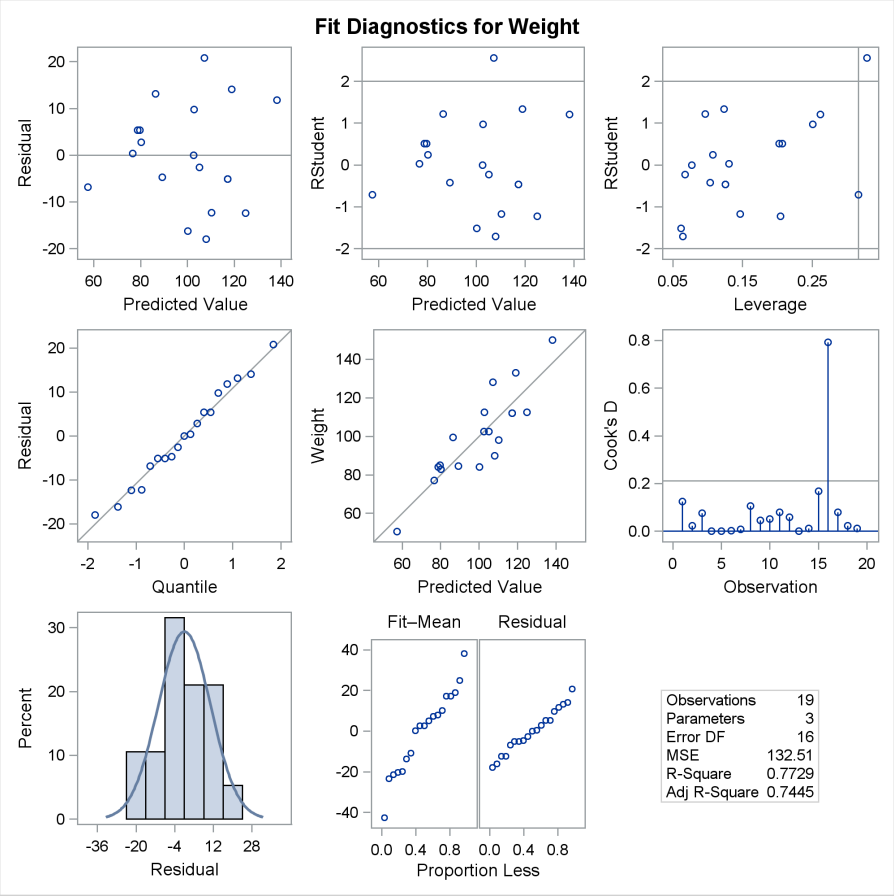
Modifying the Diagnostics Panel in PROC REG

This step runs PROC REG, as before, but this time the goal is to modify the diagnostics panel:

```
ods graphics on;
ods document name=MyDoc (write);
proc reg data=sashelp.class;
   model weight = height age / clb;
quit;
ods document close;
```

The diagnostics panel is displayed in Figure 7.13. The rest of this section modifies the table of statistics in the bottom right of the graph.

Figure 7.13 Diagnostics Panel



The following steps list the contents of the ODS document, output the dynamic variables for the diagnostics panel, suppress the adjusted R-square, and display SBC in its place:

```
proc document name=MyDoc;
   list / levels=all;
quit;

proc document name=MyDoc;
   ods output dynamics=dynamics;
   %let p = \Work.Mydoc\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1;
   obdynam &p\DiagnosticPlots#1\DiagnosticsPanel#1;
quit;

proc print noobs data=dynamics;
run;

data dynamics2;
   set dynamics;
   if label1 = '_SHOWADJRSQ' then nvalue1 = 0;
   if label1 = '_SHOWSBC'    then nvalue1 = 1;
run;
```

```
proc document name=MyDoc;
   replay &p\DiagnosticPlots#1\DiagnosticsPanel#1 / dynamdata=dynamics2;
quit;
```

The %LET statement is used so that the long path can fully be displayed in the width of a page. The output data set that contains the dynamic variables is displayed in Figure 7.14. Among other things, it contains a series of pairs of variables:

- a numeric variable that contains a statistic
- a binary variable that begins with _SHOW and specifies whether the corresponding statistic is displayed

The DATA step modifies two of the binary dynamic variables to suppress one statistic and display another. There is no need to change the variable cValue1.

Figure 7.14 Dynamic Variables

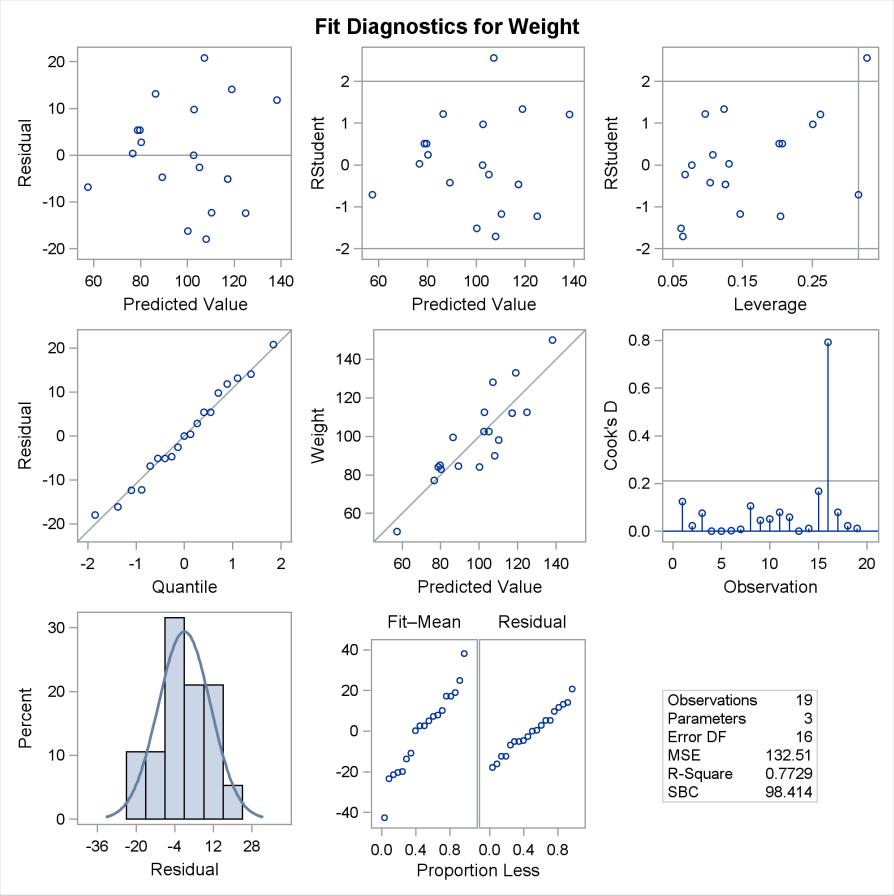
Label1	cValue1	nValue1	Label2	cValue2	nValue2
_NPARM	3	3.000000	Data		.
_TOTFREQ	19	19.000000	Data		.
_NOBS	19	19.000000	Data		.
_SHOWSTATS	1	1.000000	Data		.
_NSTATSCOLS	2	2.000000	Data		.
_SHOWNOBS	1	1.000000	Data		.
_SHOWTOTFREQ	0	0	Data		.
_SHOWNPARM	1	1.000000	Data		.
_SHOWEDF	1	1.000000	Data		.
_EDF	16	16.000000	Data		.
_SHOWMSE	1	1.000000	Data		.
_MSE	132.50623369	132.506234	Data		.
_SHOWRSQUARE	1	1.000000	Data		.
_RSQUARE	0.7729049378	0.772905	Data		.
_SHOWADJRSQ	1	1.000000	Data		.
_ADJRSQ	0.744518055	0.744518	Data		.
_SHOWSSE	0	0	Data		.
_SSE	2120.099739	2120.099739	Data		.
_SHOWDEPMEAN	0	0	Data		.
_DEPMEAN	100.02631579	100.026316	Data		.
_SHOWCV	0	0	Data		.
_CV	11.508106755	11.508107	Data		.
_SHOWAIC	0	0	Data		.
_AIC	95.580809248	95.580809	Data		.
_SHOWBIC	0	0	Data		.
_BIC	98.635496748	98.635497	Data		.
_SHOWCP	0	0	Data		.
_CP	3	3.000000	Data		.
_SHOWGMSEP	0	0	Data		.

Figure 7.14 *continued*

Label1	cValue1	nValue1	Label2	cValue2	nValue2
_GMSEP	158.07761212	158.077612	Data		.
_SHOWJP	0	0	Data		.
_JP	153.42827058	153.428271	Data		.
_SHOWPC	0	0	Data		.
_PC	0.3122557105	0.312256	Data		.
_SHOWSBC	0	0	Data		.
_SBC	98.414126185	98.414126	Data		.
_SHOWSP	0	0	Data		.
_SP	8.8337489124	8.833749	Data		.
BYTITLE		.	Data		.
BYLINE		.	Data		.
BYFOOTNOTE		.	Data		.
_DEPNAME	Weight	.	Data		.
_DEPLABEL	Weight	.	Data		.
___NOBS___	19	19.000000	Data		.
___NOBS___	19	19.000000	Column	Residual	.
___NOBS___	19	19.000000	Column	PredictedValue	.
___NOBS___	19	19.000000	Column	Observation	.
___NOBS___	19	19.000000	Column	RStudent	.
___NOBS___	19	19.000000	Column	HatDiagonal	.
___NOBS___	19	19.000000	Column	DepVar	.
___NOBS___	19	19.000000	Column	CooksD	.

The modified diagnostics panel is displayed in Figure 7.15. You will see SBC instead of adjusted R-Square in the bottom right.

Figure 7.15 Diagnostics Panel



This example illustrates dynamic-variable modification techniques that you can apply to other situations. However, you are not required to use them in this situation. The following step uses the STATS= option to perform the same modification of the diagnostics panel:

```
proc reg data=sashelp.class
         plots=diagnostics(stats=(nobs nparm edf mse rsquare sbc));
   model weight = height age / clb;
quit;
```

The results match Figure 7.15.

Modifying Contour Plots

The following steps create some random normal data, display them in a contour plot by using the KDE procedure, save the PROC KDE results in an ODS document, and replay the graph after modifying the values of the dynamic variables that contain the variable names for the title:

```
data bivnormal;
   do i = 1 to 1000;
      z1 =      rannor(17151377);
      x  = 3*z1+rannor(17151377);
      y  = 3*z1+rannor(17151377);
   output;
   end;
run;

ods graphics on;
ods document name=MyDoc(write);
proc kde data=bivnormal;
   bivar x y / plots=(contour surface);
run;
ods document close;

proc document name=MyDoc;
   list / levels=all;
quit;

proc document name=MyDoc;
   ods output dynamics=dynamics;
   obdynam \KDE#1\Bivar1#1\x_y#1\ContourPlot#1;
quit;

data dynamics2;
   length cvalue1 $ 13;
   set dynamics;
   if label1 = '_VAR1NAME' then cvalue1 = 'an X Variable';
   if label1 = '_VAR2NAME' then cvalue1 = 'a Y Variable';
run;

proc document name=MyDoc;
   replay \KDE#1\Bivar1#1\x_y#1\ContourPlot#1 / dynamdata=dynamics2;
quit;
```

The results are displayed in Figure 7.16. The original graph contains the variable names x and y in the graph title, and the new graph contains the strings `'an X Variable'` and `'a Y Variable'`.

Figure 7.16 Contour Plot Dynamic Variable Modification

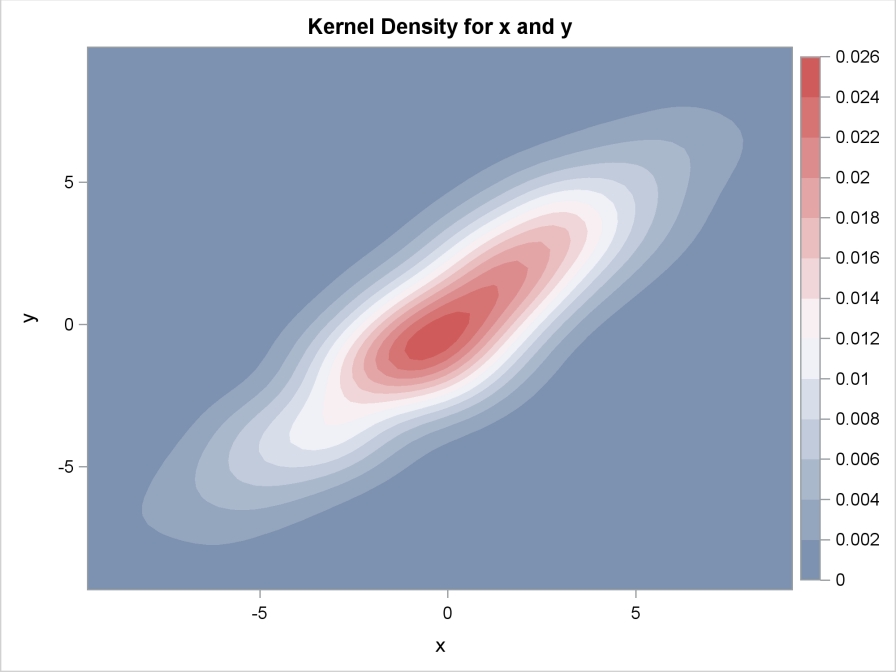


Figure 7.16 *continued*

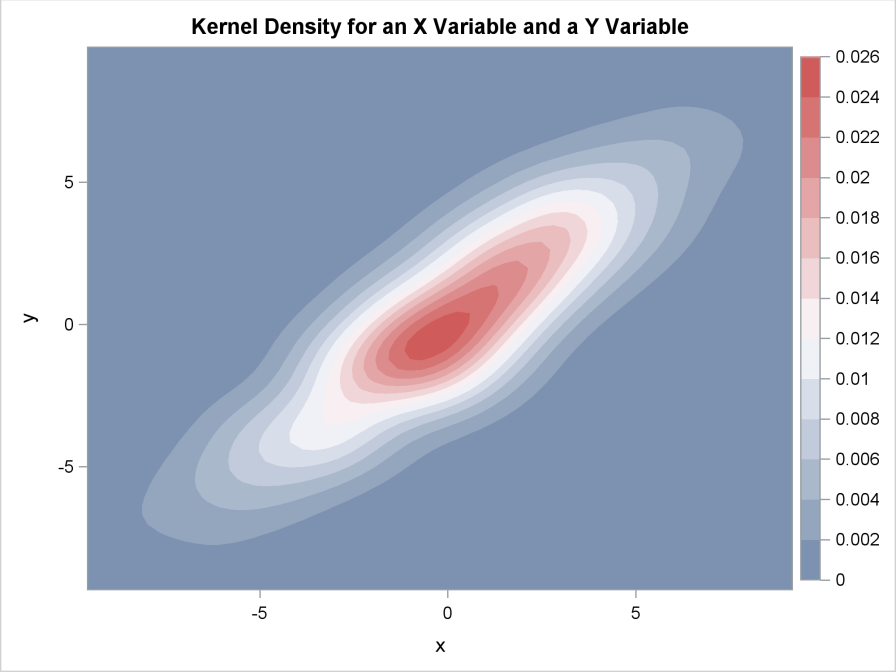
Listing of: \Work.Mydoc
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\KDE#1	Dir
2	\KDE#1\Bivar1#1	Dir
3	\KDE#1\Bivar1#1\x_y#1	Dir
4	\KDE#1\Bivar1#1\x_y#1\Inputs#1	Table
5	\KDE#1\Bivar1#1\x_y#1\Controls#1	Table
6	\KDE#1\Bivar1#1\x_y#1\ContourPlot#1	Graph
7	\KDE#1\Bivar1#1\x_y#1\SurfacePlot#1	Graph

Dynamics for: \Work.Mydoc\KDE#1\Bivar1#1\x_y#1\ContourPlot#1

Name	Value	Type	Namespace
BYTITLE		Data	
BYLINE		Data	
BYFOOTNOTE		Data	
_VAR1NAME	"x"	Data	
_VAR2NAME	"y"	Data	
___NOBS___	3600	Data	
___NOBS___	3600	Column	DensityX
___NOBS___	3600	Column	DensityY
___NOBS___	3600	Column	Density

Figure 7.16 *continued*



Annotating Single-Panel Graphs That Analytical Procedures Produce

Double Click for Example Code

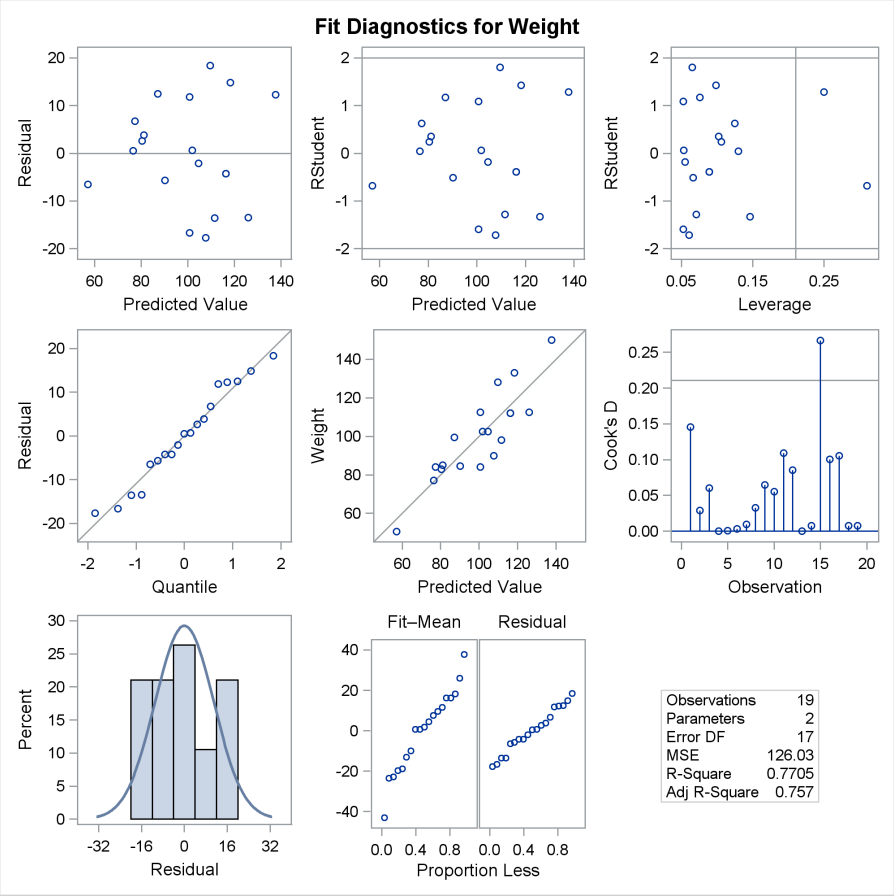
Previous examples showed many ways to modify the graphs that SAS creates. Standard graph customization methods include template modification (which most people use to modify graphs that SAS analytical procedures produce) and SG annotation (which most people use to modify graphs that procedures such as PROC SGPLOT produce). However, you can also use SG annotation to modify graphs that analytical procedures produce. This example builds on section "Changing Dynamic Variables by Using the ODS Document" on page 197, which shows you how to modify the dynamic variables and display the results by using PROC DOCUMENT. This example shows you how to capture dynamic variables, modify them, and create a modified graph by using PROC SGRENDER instead of PROC DOCUMENT. This approach enables you to use SG annotation to modify graphs that SAS analytical procedures create.

This step runs PROC REG, displays the diagnostics panel, and outputs the data object to a SAS data set:

```
ods graphics on;
proc reg data=sashelp.class;
   ods select diagnosticspanel;
   ods output diagnosticspanel=dp;
   model weight = height;
quit;
```

The results are displayed in Figure 7.17.

Figure 7.17 Diagnostics Panel

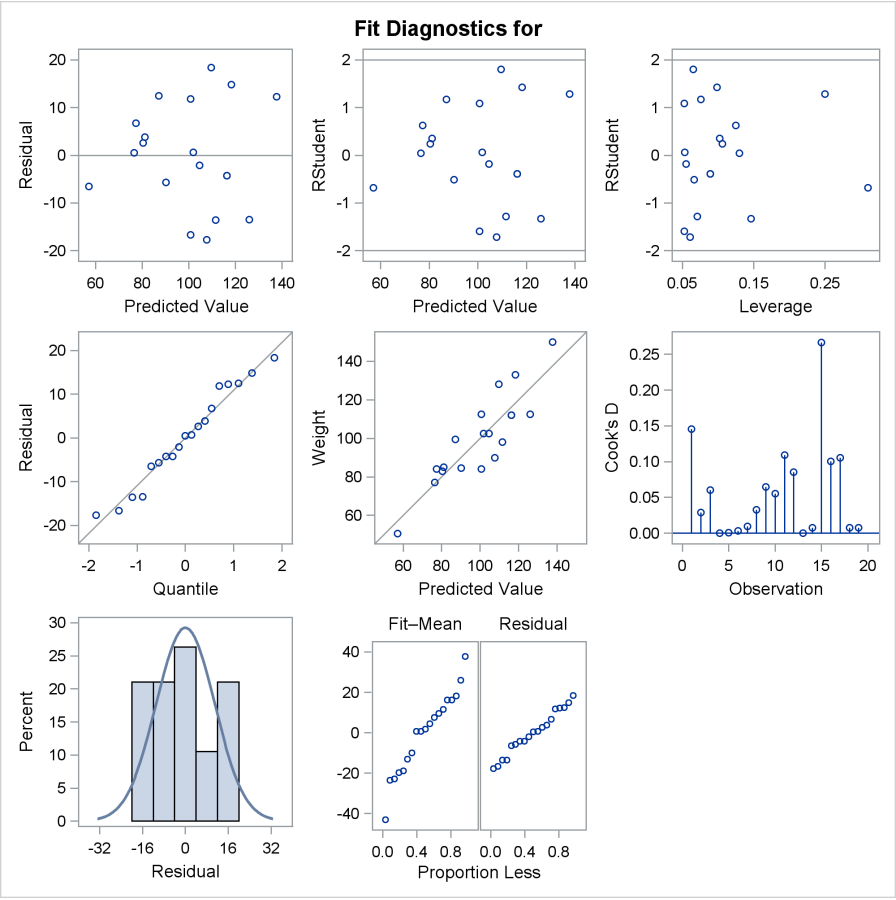


You might consider a naive approach to recreating the diagnostics panel from the data object and the graph template by using PROC SGRENDER as follows:

```
proc sgrender data=dp template=Stat.REG.Graphics.DiagnosticsPanel;
run;
```

For some graphs, this might completely work (if there are no dynamic variables) or it might completely fail (for example, if there is one graph statement and a critical part depends on dynamic variables). The preceding step partially works. In this example, the statistics table is completely missing, part of the title is missing, and some reference lines are missing. The results are displayed in Figure 7.18.

Figure 7.18 Dynamic Information Missing



You can run the following step to create the graph, output the data object to a SAS data set, and capture the dynamic variables in an ODS document:

```
ods document name=MyDoc (write);
proc reg data=sashelp.class;
   ods select diagnosticspanel;
   ods output diagnosticspanel=dp;
   model weight = height;
quit;
ods document close;
```

You can list the contents of the ODS document as follows:

```
proc document name=MyDoc;
   list / levels=all;
quit;
```

You can store the names of the dynamic variables and their values in a SAS data set as follows:

```
proc document name=MyDoc;
   ods output dynamics=outdynam;
   obdynam
   \Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\DiagnosticPlots#1\DiagnosticsPanel#1;
quit;
```

The path in the OBDYNAM statement is copied from the listing of the contents of the ODS document.

The next several steps process both the data set of dynamic variables and the graph template so that a subsequent PROC SGRENDER step can recreate the graph. Graph templates that procedures use often include a DYNAMIC statement that lists dynamic variables. Graph templates that you write can use dynamic variables, but they can also get dynamic information through macro variables. You can use an MVAR statement to provide character macro variables, and you can use an NMVAR statement to provide macro variables whose values are processed as numbers. The next steps process the dynamic variables and their values, output them to macro variables, and modify the graph template to use MVAR and NMVAR statements instead of a DYNAMIC statement.

The following step preprocesses the data set of dynamic variables:

```
data dynamics;
   length label1 $ 32;
   set outdynam;
   label1 = upcase(label1);
   if label1 ne '___NOBS___';
run;
```

Variable names are uppercased, and the automatic dynamic variables that contain the number of observations in the data object columns are discarded.

The following step writes the graph template to a file:

```
proc template;
   source Stat.REG.Graphics.DiagnosticsPanel / file='temp.tmp';
quit;
```

If you need to do ad hoc template modifications, you can do them before you perform the preceding step or build them into the subsequent DATA step that processes the template.

The following step reads the file that contains the graph template, identifies the beginning of the DYNAMIC statement (`_infile_ =: ' dynamic '`), and extracts the names of all of the dynamic variables:

```
data d(keep=label1);
   infile 'temp.tmp';
   input;
   length label1 $ 32;
   if _infile_ =: '   dynamic ' then do;
      d + 1;
      substr(_infile_, 1, 10) = ' ';
      end;
   if d then do;
      do i = 1 to 128 until(label1 eq ' ');
         label1 = upcase(scan(_infile_, i, ' ;'));
         if label1 ne ' ' then output;
         end;
      end;
   if d and index(_infile_, ';') then stop;
run;
```

The automatic variable _infile_ contains the contents of the input buffer. When the variable D is 1, the step is processing dynamic variables. This step stops when it reaches the semicolon at the end of the DYNAMIC statement (`d and index(_infile_, ';')`).

The following steps sort the two lists of dynamic variables so that they can be merged:

```
proc sort data=dynamics; by label1; run;
proc sort data=d;        by label1; run;
```

The following step merges the two dynamic variable lists and sets missing character values to ordinary blank missing:

```
data dynamics(drop=label2 cvalue2 nvalue2);
   merge d dynamics;
   by label1;
   if nmiss(nvalue1) and cvalue1 = '.' then cvalue1 = ' ';
run;
```

The following step reads the template file again and modifies it:

```
data _null_;
   infile 'temp.tmp';
   input;
   if _n_ = 1 then call execute('proc template;');
   if _infile_ =: '   dynamic ' then do;
      substr(_infile_, 1, 10) = '*';
      do i = 1 to ndynam;
         set dynamics point=i nobs=ndynam;
         call execute(catx(' ', ifc(n(nvalue1), 'nmvar', 'mvar'), label1, ';'));
         end;
      end;
   call execute(_infile_);
   if _infile_ =: '   BeginGraph' then bg + 1;
   if bg and index(_infile_, ';') then do;
      bg = 0;
      call execute('annotate;');
   end;
run;
```

This step uses CALL EXECUTE to submit a PROC TEMPLATE statement, convert the DYNAMIC statement to a comment, submit an unmodified version of every other template statement, add an ANNOTATE statement after the BEGINGRAPH statement (to enable subsequent SG annotation), and submit a series of NMVAR and MVAR statements. There are various ways to annotate by using GTL. This is the simplest, and it enables you to use annotation coordinates in graph percentage units. For other options, see other examples in this book. The following step is not necessary, but it shows the modified template:

```
proc template;
   source Stat.REG.Graphics.DiagnosticsPanel;
quit;
```

The following step creates all the macro variables that the NMVAR and MVAR statements need:

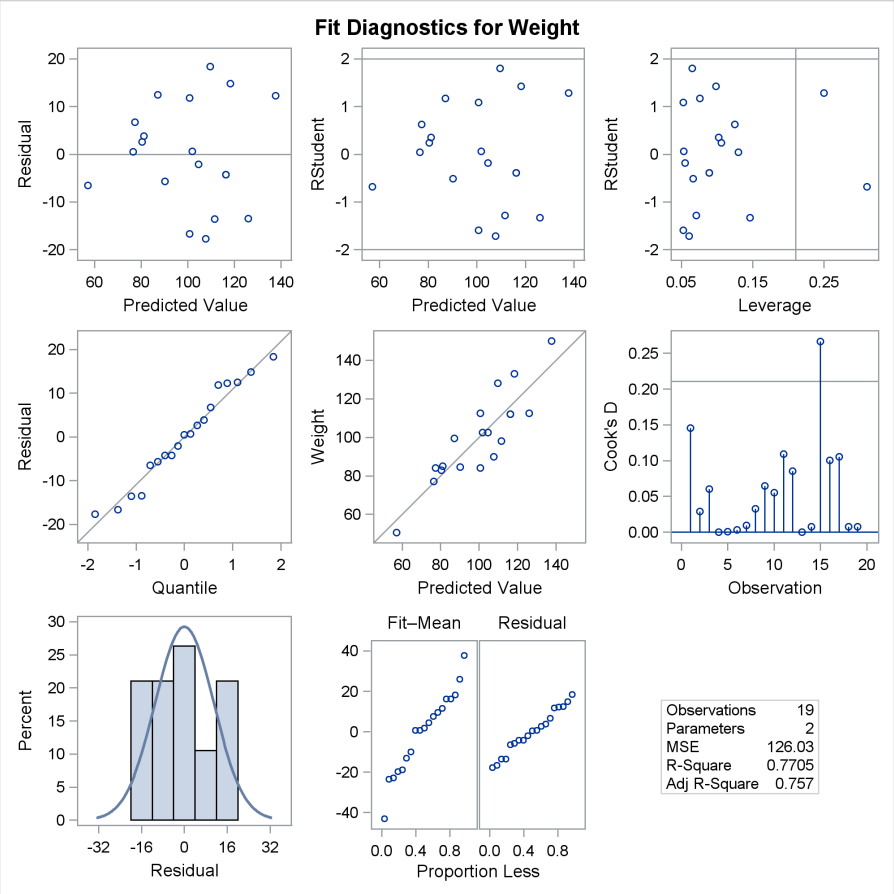
```
data _null_;
   set dynamics;
   if label1 = '_SHOWEDF' then cvalue1 = '0';
   call symputx(label1, cvalue1);
run;
```

This step also modifies one of the dynamic variables. It sets _SHOWEDF to 0 to suppress the display of the error degrees of freedom in the statistics table. (You can instead do this directly in PROC REG.) The following steps create the diagnostics panel from the data set that is made from the data object, the modified graph template, and all the dynamic variables (now stored in macro variables):

```
proc sgrender data=dp template=Stat.REG.Graphics.DiagnosticsPanel;
run;
```

The results are displayed in Figure 7.19.

Figure 7.19 Error Degrees of Freedom Suppressed

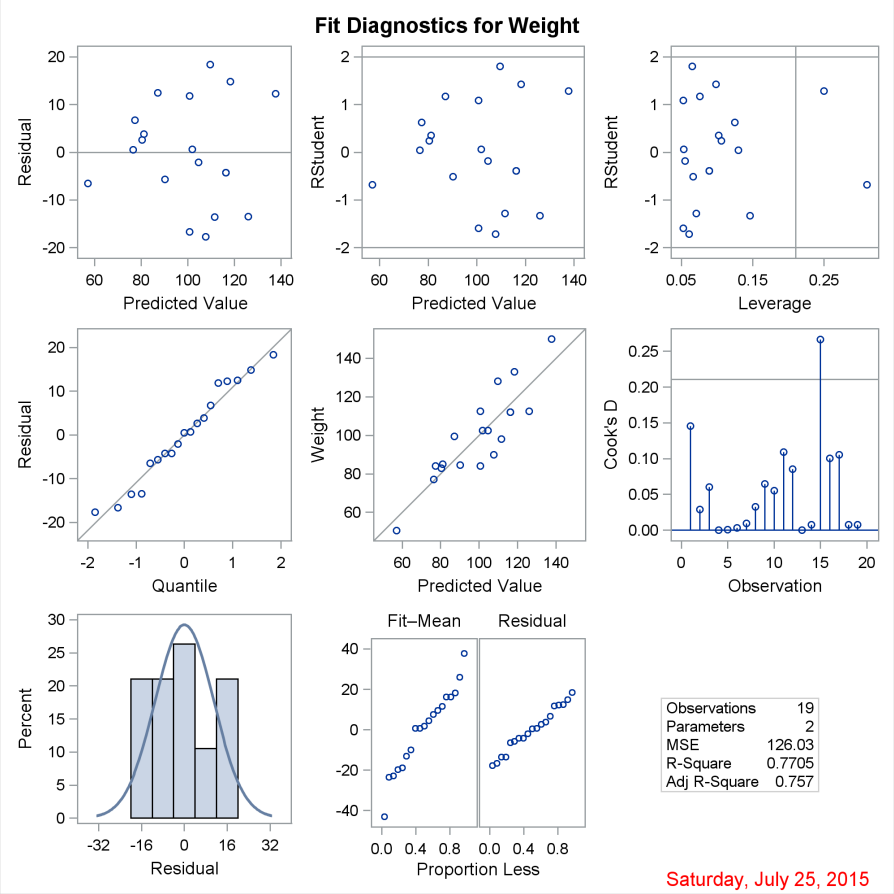


Now you can use SG annotation to modify the graph. This is illustrated in two simple examples. The first example adds a date to the bottom right corner of the graph:

```
data anno;
   Function = 'Text'; Label = 'Saturday, July 25, 2015';
   Width = 100;    x1 = 99;    y1 = .1;    Anchor = 'Right';  TextColor = 'Red';
run;

proc sgrender data=dp sganno=anno
              template=Stat.REG.Graphics.DiagnosticsPanel;
run;
```

The results are displayed in Figure 7.20.

Figure 7.20 Simple Annotation



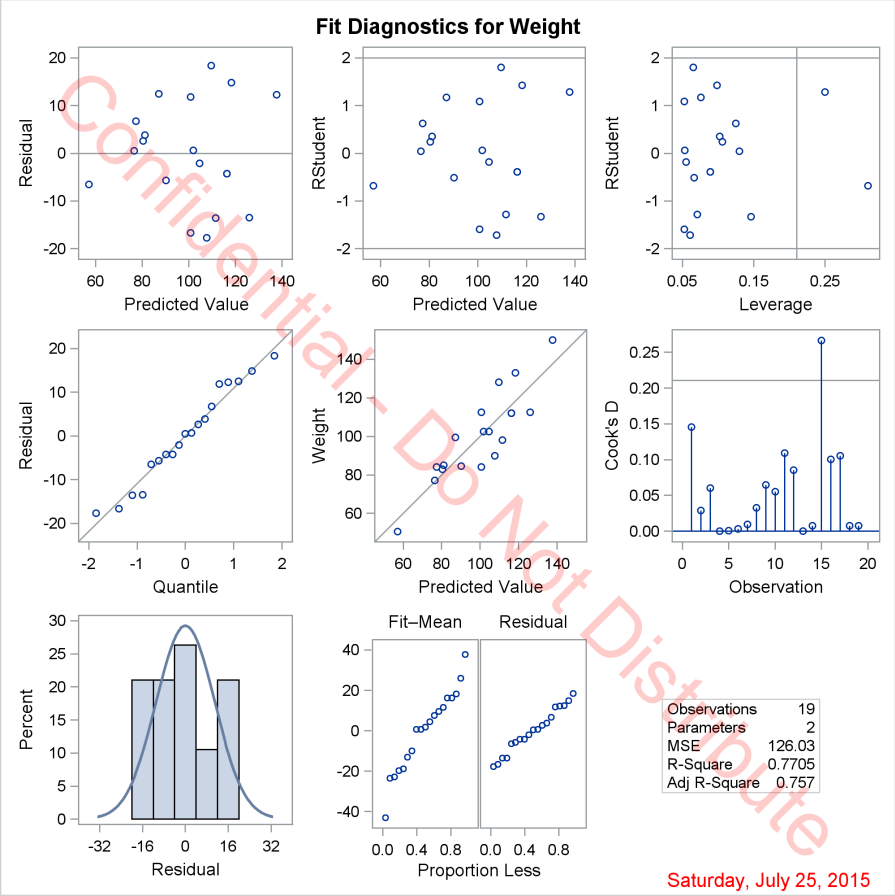
The second example also adds a watermark across the graph:

```
data anno;
   length Label $ 40;
   Function = 'Text';      Label     = 'Saturday, July 25, 2015';
   Width    = 100;         x1        = 99;   y1 = .1;
   Anchor   = 'Right';     TextColor = 'Red';
   output;

   Label = 'Confidential - Do Not Distribute';
   Width = 150;            x1        = 50;   y1     =  50;   Anchor = 'Center';
```

```
   Transparency = 0.8;    TextSize  = 40;    Rotate = -45;
   output;
run;

proc sgrender data=dp sganno=anno
              template=Stat.REG.Graphics.DiagnosticsPanel;
run;
```

The results are displayed in Figure 7.21.

Figure 7.21 Two Annotations



Like most things in SAS, there is more than one way to approach this problem. The following step combines all preceding steps that follow the creation of the OUTDYNAM data set (except the annotation data set creation step). The first step adds the ANNOTATE statement to the template:

```
data _null_;
   infile 'temp.tmp';
   input;
   if _n_ = 1 then call execute('proc template;');
   call execute(_infile_);
   if _infile_ =: '   BeginGraph' then bg + 1;
   if bg and index(_infile_, ';') then do;
      bg = 0;
      call execute('annotate;');
   end;
run;
```

Other than that, the template is not modified. The following step generates and runs the PROC SGRENDER step:

```
data _null_;
   set outdynam(where=(label1 ne '___NOBS___')) end=eof;
   if nmiss(nvalue1) and cvalue1 = '.' then cvalue1 = ' ';
   if _n_ = 1 then do;
      call execute('proc sgrender data=dp sganno=anno');
      call execute('template=Stat.REG.Graphics.DiagnosticsPanel;');
      call execute('dynamic');
   end;
   if label1 = '_SHOWEDF' then cvalue1 = '0';
   if cvalue1 ne ' ' then
      call execute(catx(' ', label1, '=',
                  ifc(n(nvalue1), cvalue1, quote(trim(cvalue1)))));
   if eof then call execute('; run;');
run;
```

The results match the previous graph. Instead of processing two lists of dynamic variables, this step runs PROC SGRENDER along with a customized DYNAMIC statement that populates the dynamic variables with values. This approach has the advantage of requiring less code. However, the final PROC SGRENDER step is entangled with the processing of dynamic variables. You might prefer to process the dynamic variables and then have a simple PROC SGRENDER step you can run each time that you want to try a new modification of the graph. Either way, SAS provides you the flexibility that you need to modify a graph.

The next example also modifies the graph template to provide the same formatting for the R-square and the adjusted R-square:

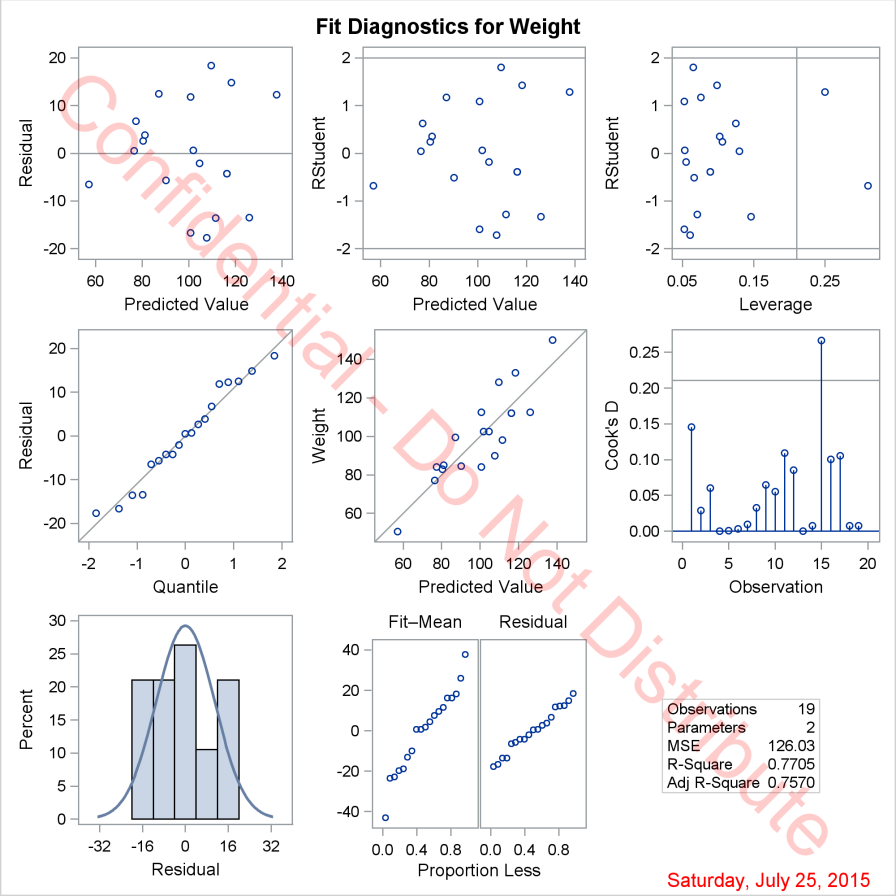
```
data _null_;
   infile 'temp.tmp';
   input;
   if _n_ = 1 then call execute('proc template;');

   i = index(_infile_, 'BEST6.');
   if i and (index(_infile_, '_ADJRSQ') or index(_infile_, '_RSQUARE'))
      then substr(_infile_, i, 6) = '6.4';

   call execute(_infile_);
   if _infile_ =: '   BeginGraph' then bg + 1;
   if bg and index(_infile_, ';') then do;
      bg = 0;
      call execute('annotate;');
   end;
run;

data _null_;
   set outdynam(where=(label1 ne '___NOBS___')) end=eof;
   if nmiss(nvalue1) and cvalue1 = '.' then cvalue1 = ' ';
   if _n_ = 1 then do;
      call execute('proc sgrender data=dp sganno=anno');
      call execute('template=Stat.REG.Graphics.DiagnosticsPanel;');
      call execute('dynamic');
   end;
   if label1 = '_SHOWEDF' then cvalue1 = '0';
```

```
   if cvalue1 ne ' ' then
      call execute(catx(' ', label1, '=',
                   ifc(n(nvalue1), cvalue1, quote(trim(cvalue1)))));
   if eof then call execute('; run;');
run;
```

The first step uses an IF statement to change the BEST6. format to a 6.4 format for the R-square and the adjusted R-square. Of course, you do not need to modify templates in a DATA step, but the DATA step provides a convenient and parsimonious way to show the change.

The results are displayed in Figure 7.22.

Figure 7.22 Formats Aligned



The following step deletes the modified template:

```
proc template;
   delete Stat.REG.Graphics.DiagnosticsPanel;
quit;
```

Assuming that you are creating exactly one graph and then annotating it, you can use the macro in the following steps to process the template and dynamic variables:

```
ods graphics on;
ods document name=MyDoc (write);
proc reg data=sashelp.class;
   ods select diagnosticspanel;
   ods output diagnosticspanel=dp;
   model weight = height;
quit;
ods document close;

data anno;
   length Label $ 40;
   Function = 'Text';     Label     = 'Saturday, July 25, 2015';
   Width    = 100;        x1        = 99;   y1 = .1;
   Anchor   = 'Right';    TextColor = 'Red';
   output;

   Label = 'Confidential - Do Not Distribute';
   Width = 150;           x1        = 50;   y1     =  50;   Anchor = 'Center';
   Transparency = 0.8;    TextSize  = 40;   Rotate = -45;
   output;
run;

%macro procanno(data=, template=, anno=anno, document=mydoc);
   proc document name=&document;
      ods exclude properties;
      ods output properties=__p(where=(type='Graph'));
      list / levels=all;
   quit;

   data _null_;
      set __p;
      call execute("proc document name=&document;");
      call execute("ods exclude dynamics;");
      call execute("ods output dynamics=__outdynam;");
      call execute(catx(' ', "obdynam", path, ';'));
   run;

   proc template;
      source &template / file='temp.tmp';
   quit;

   data _null_;
      infile 'temp.tmp';
      input;
      if _n_ = 1 then call execute('proc template;');
      call execute(_infile_);
      if _infile_ =: '   BeginGraph' then bg + 1;
      if bg and index(_infile_, ';') then do;
         bg = 0;
         call execute('annotate;');
      end;
   run;
```

```
   data _null_;
      set __outdynam(where=(label1 ne '___NOBS___')) end=eof;
      if nmiss(nvalue1) and cvalue1 = '.' then cvalue1 = ' ';
      if _n_ = 1 then do;
         call execute("proc sgrender data=&data sganno=&anno");
         call execute("template=&template;");
         call execute('dynamic');
      end;
      if cvalue1 ne ' ' then
         call execute(catx(' ', label1, '=',
                      ifc(n(nvalue1), cvalue1, quote(trim(cvalue1)))));
      if eof then call execute('; run;');
   run;

   proc template;
      delete &template;
   quit;
%mend;

%procanno(data=dp, template=Stat.REG.Graphics.DiagnosticsPanel)
```

You create the graph, capture the dynamic variables in an ODS document, and create the annotation data set; the macro does the rest. The results match Figure 7.21. If you want to modify the graph template, you could do that before you call the macro. You could instead enhance the %ProcAnno macro to accept template modification statements just as the %LabelIt macro in the section "Placing Labels in Scatter Plots" on page 163 accepts label coordinate modifications. The modifications would be inserted into the DATA step that processes the file *temp.tmp*. The next example illustrates.

Annotating Multiple-Panel Graphs That Analytical Procedures Produce

Double Click for Example Code

This example requires a thorough understanding of the previous two examples. In this example, the %ProcAnno macro from the previous example is modified to add an ANNOTATE statement to each LAYOUT OVERLAY code block rather than adding one ANNOTATE statement for the entire template. This enables you to send annotations to each graph within a panel, rather than basing the annotation coordinates on the full panel. Furthermore, the macro that processes the template now supports a macro that provides other template changes. This example also modifies the data object[1] and the graph template.

[1] **CAUTION:** Do not change the data that underlie a graph. This example changes only how parts of the graph are labeled.

This example processes the standardized coefficients progression plot in the GLMSELECT procedure. The following step creates the plot:

```
ods graphics on;
proc glmselect data=sashelp.baseball plots=coefficients;
   class league division;
   model logSalary = nAtBat nHits nHome nRuns nRBI nBB
                     yrMajor|yrMajor crAtBat|crAtBat crHits|crHits
                     crHome|crHome crRuns|crRuns crRbi|crRbi
                     crBB|crBB league division nOuts nAssts nError /
                     selection=forward(stop=AICC CHOOSE=SBC);
run;
```

The results are displayed in Figure 7.23.

The graph shows how the coefficients change as new terms enter the model. PROC GLMSELECT labels some of the series plots. It is common in this graph for several coefficients to end up having similar values. PROC GLMSELECT tries to thin labels to avoid conflicts. For example, the first term that enters the model after the intercept is CrRuns. Its label is not displayed because it would conflict with the label for CrHits. In this example, you will learn how to select a different set of labels to display. In particular, you will select the labels for the first three terms that enter the model. Doing so requires you to change the data object. Then you can add annotation to highlight the selected model. In PROC GLMSELECT, the final model does not usually correspond to the end of the progression of the coefficients. In this case, it corresponds to the vertical reference line at step 9 in the graph, which is labeled as "9+CrBB" on the X axis, and indicates that the variable CrBB entered the model at step 9. You can preview the results in Figure 7.24.

Figure 7.23 Coefficient Progression from PROC GLMSELECT

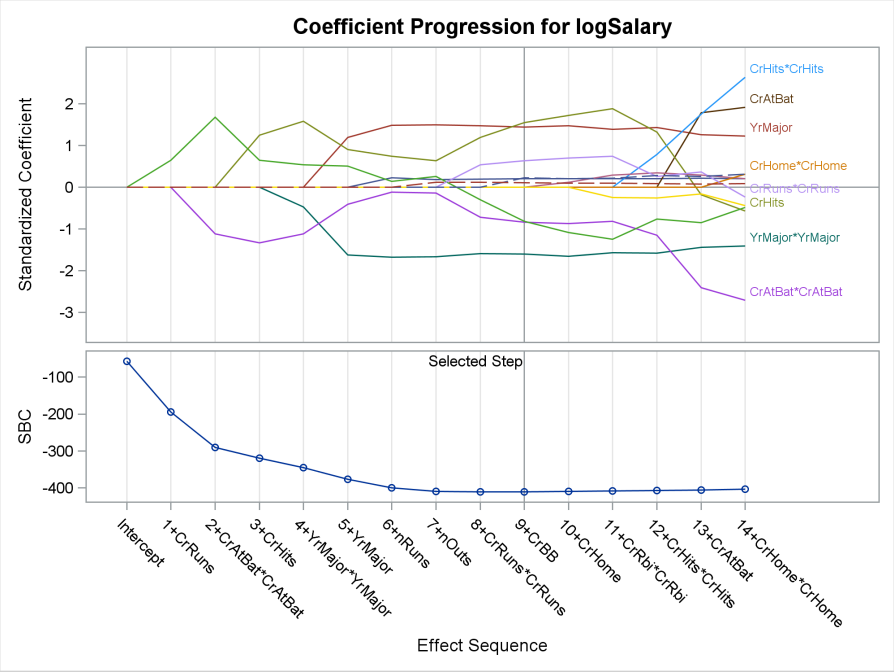
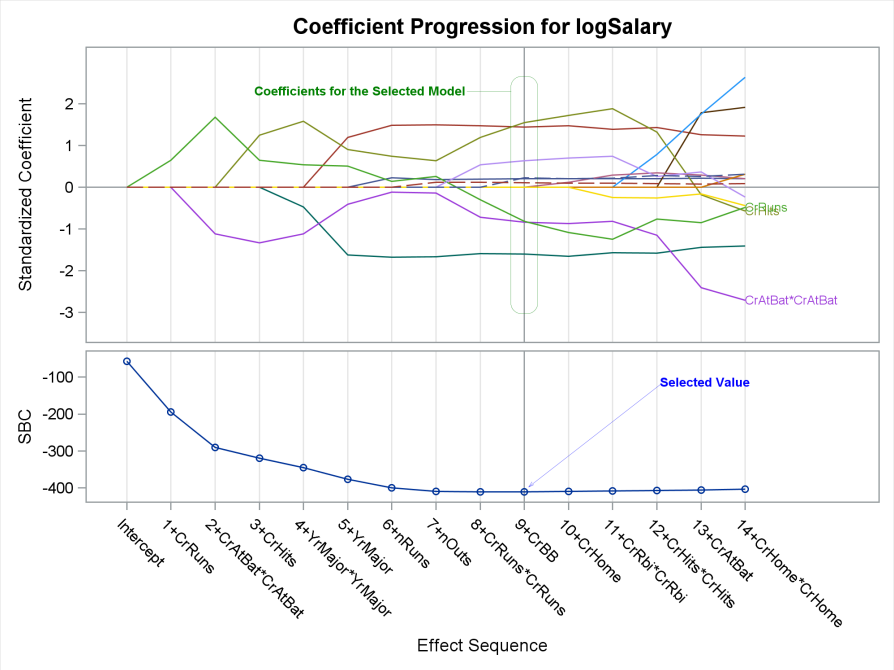


Figure 7.24 Modified Coefficient Progression



As in the previous example, you begin by creating a data object and storing the graph along with the dynamic variables in an ODS document:

```
ods document name=MyDoc (write);
proc glmselect data=sashelp.baseball plots=coefficients;
   ods select CoefficientPanel;
   ods output CoefficientPanel=cp;
   class league division;
   model logSalary = nAtBat nHits nHome nRuns nRBI nBB
                     yrMajor|yrMajor crAtBat|crAtBat crHits|crHits
                     crHome|crHome crRuns|crRuns crRbi|crRbi
                     crBB|crBB league division nOuts nAssts nError /
                     selection=forward(stop=AICC CHOOSE=SBC);
run;
ods document close;
```

The next step reads the data object, extracts the parameter labels from steps 1 through 3, and outputs the number of the last step to a macro variable:

```
data labelthese(keep=par);
   set cp end=eof;
   retain f1-f3 1;
   if f1 and steplabel =: '1+' then do; f1 = 0; link s; end;
   if f2 and steplabel =: '2+' then do; f2 = 0; link s; end;
   if f3 and steplabel =: '3+' then do; f3 = 0; link s; end;
   if eof then call symputx('_step', step);
   return;
s: par = substr(steplabel, 3);
   output;
   return;
run;
```

```
proc print noobs;
run;
```

The results are displayed in Figure 7.25.

Figure 7.25 First Three Terms

par
CrRuns
CrAtBat*CrAtBat
CrHits

The next step processes the data set that was created from the data object:

```
data cp2;
   set cp;
   match = 0;
   if step ne &_step then return;
   do i = 1 to ntolabel;
      set labelthese point=i nobs=ntolabel;
      match + (par = parameter);
      end;
   if not match then parameter = ' ';
   if nmiss(rhslabelYvalue) then rhslabelYvalue = StandardizedEst;
run;
```

This data object is typical of the data objects that are used to make graphs. It has several components of different sizes and missing values elsewhere. The last part of the data object contains the coordinates and strings that are needed to label each profile. The preceding DATA step sets the parameter value to blank in the last step that PROC GLMSELECT considers (which corresponds to the end of the profiles in the graph) for all but the first three terms. When the Y coordinate for a label is missing (because PROC GLMSELECT suppressed it because of collisions), the Y coordinate value is restored.

The next step creates the %Tweak macro, which contains the code that modifies the graph template:

```
%macro tweak;
   if index(_infile_, 'datalabel=PARAMETER') then
      _infile_ = tranwrd(_infile_, 'datalabel',
                         'markercharacterposition=right markercharacter');
   if index(_infile_, 'curvelabel="Selected Step"') then
      _infile_ = tranwrd(_infile_, 'curvelabel="Selected Step"', ' ');
%mend;
```

The macro uses two IF statements that each perform a change:

- The first IF statement removes the DATALABEL= option in a SCATTERPLOT statement and instead specifies the MARKERCHARACTER= option. You can use the MARKERCHARACTER= option to position labels precisely at a point. In contrast, the DATALABEL= option moves labels that conflict. The first IF statement also adds the MARKERCHARACTERPOSITION=RIGHT option so that labels are positioned to the right of the coordinates. The TRANWRD (translate word) function performs the change, substituting a longer string from a shorter string.
- The second IF statement removes the curve label. You will later add it back in through SG annotation.

The next step creates the annotation data set:

```
data anno;
   length ID $ 3 Function $ 9 Label $ 40;
   retain x1Space y1Space x2Space y2Space 'DataPercent' Direction 'In';
   length Anchor $ 10 xC1 xC2 $ 20;
   retain Scale 1e-12 Width 100 WidthUnit 'Data' CornerRadius 0.8
          TextSize 7 TextWeight 'Bold'
          LineThickness 0.7 DiscreteOffset -0.3 LineColor 'Green';

   ID       = 'LO1';            Function  = 'Text';
   Anchor   = 'Right';          TextColor = 'Green';
   x1       = 55;               y1        = 94;
   Label    = 'Coefficients for the Selected Model';
   output;

   Function = 'Line';           x1        = .;
   x1Space  = 'DataValue';      x2Space   = x1Space;
   xc1      = '9+CrBB';         xc2       = '8+CrRuns*CrRuns';
   y1       = 94;               y2        = 94;
   output;

   Function = 'Rectangle';      y1Space   = 'WallPercent';
   Anchor   = 'BottomLeft';     y1        = 10;
   Height   = 80;               Width     = 0.6;
   output;

   ID       = 'LO3';            Width     = 100;
   Function = 'Text ';          Label     = 'Selected Value';
   x1Space  = 'DataPercent';    y1Space   = x1Space;
   Anchor   = 'Left';           TextColor = 'Blue';
   x1       = 86;               y1        = 84;
   output;

   Function = 'Arrow';          LineColor = 'Blue';
   x1Space  = 'DataValue';      x2Space   = x1Space;
   xc1      = '9+CrBB';         xc2       = '12+CrHits*CrHits';
   y1       = 4;                y2        = 83;
   DiscreteOffset = .1;         x1        = .;
   output;
run;
```

The annotation data set has five observations:

1. the text string ‘Coefficients for the Selected Model’
2. a line from the text string to the rectangle
3. a rectangle that has rounded corners and surrounds the coefficients for the selected model
4. the text string ‘Selected Value’
5. an arrow that points from the text string to the selected value

This annotation data set has many variables and includes options that have not been shown in previous examples. The annotation data set is described further after the graph is displayed.

The new and advanced template processing macro, %ProcAnnoAdv, is next:

```
%macro procannoadv(data=, template=, anno=anno, document=mydoc, adjust=,
                   overallanno=1);

   proc document name=&document;
      ods exclude properties;
      ods output properties=__p(where=(type='Graph'));
      list / levels=all;
   quit;

   data _null_;
      set __p;
      call execute("proc document name=&document;");
      call execute("ods exclude dynamics;");
      call execute("ods output dynamics=__outdynam;");
      call execute(catx(' ', "obdynam", path, ';'));
   run;

   proc template;
      source &template / file='temp.tmp';
   quit;

   data _null_;
      infile 'temp.tmp';
      input;
      if _n_ = 1 then call execute('proc template;');
      %if &adjust ne %then %do; %&adjust %end;
      call execute(_infile_);
      if &overallanno and _infile_ =:     '   BeginGraph' then bg + 1;
      else if not &overallanno and index(_infile_, '   layout overlay')
         then lo + 1;
      if bg and index(_infile_, ';') then do;
         bg = 0;
         call execute('annotate;');
      end;
      if lo and index(_infile_, ';') then do;
         lo = 0;
         lonum + 1;
         call execute(catt('annotate / id="LO', lonum, '";'));
      end;
   run;

   data _null_;
      set __outdynam(where=(label1 ne '___NOBS___')) end=eof;
      if nmiss(nvalue1) and cvalue1 = '.' then cvalue1 = ' ';
      if _n_ = 1 then do;
         call execute("proc sgrender data=&data");
         if symget('anno') ne ' ' then call execute("sganno=&anno");
         call execute("template=&template;");
         call execute('dynamic');
      end;
      if cvalue1 ne ' ' then
         call execute(catx(' ', label1, '=',
                     ifc(n(nvalue1), cvalue1, quote(trim(cvalue1)))));
```

```
      if eof then call execute('; run;');
   run;

   proc template;
      delete &template;
   quit;
%mend;
```

As in the %LabelIt macro, you can specify a macro name in the ADJUST= argument so that you can insert code into the macro in order to edit the graph template. In this case, you add the %Tweak macro. You can set the ANNO= option to blank to prevent PROC SGRENDER from specifying the SGANNO= option. By default (or when OVERALLANNO=1) a single ANNOTATE statement is added to the template (as in the section "Annotating Single-Panel Graphs That Analytical Procedures Produce" on page 217). In this example, OVERALLANNO=0 and an ANNOTATE statement is added to each layout overlay. The following statements are added to the template:

```
annotate / id="LO1";
annotate / id="LO2";
annotate / id="LO3";
```

The ID names are arbitrary as long as the ANNOTATE statement ID name matches the name in the annotation data set. These names stand for "Layout Overlay 1", "Layout Overlay 2", and "Layout Overlay 3". The structure of the modified template, with most of the statements and options deleted, is as follows:

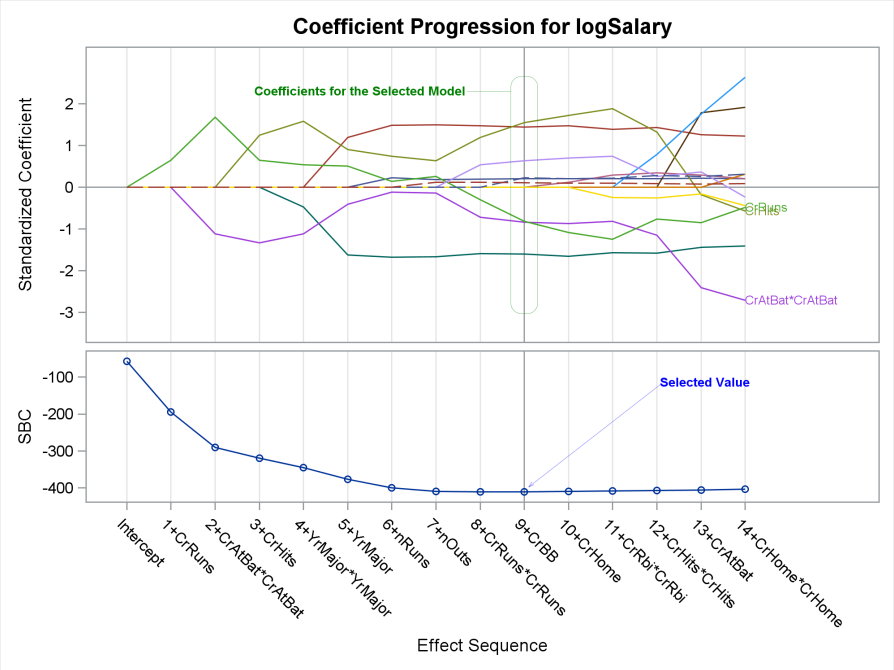
```
define statgraph Stat.GLMSelect.Graphics.CoefficientPanel;
   . . .
   BeginGraph;
      layout lattice . . .;
         layout overlay . . .;
            annotate / id="LO1";
            . . .
         endlayout;
         if (_SHOWPVAL = 1)
            layout overlay . . .;
               annotate / id="LO2";
               . . .
            endlayout;
         else
            layout overlay . . .;
               annotate / id="LO3";
               . . .
            endlayout;
         endif;
      endlayout;
      . . .
   EndGraph;
end;
```

You can use the three IDs in order to modify each of the three overlays. In this template, the first LAYOUT OVERLAY is unconditionally used and either the second and or third LAYOUT OVERLAY is conditionally used (because of the IF and ELSE statements). In this example, the first and third LAYOUT OVERLAYs are used.

The following step runs the macro and creates the graph displayed in Figure 7.26:

```
%procannoadv(data=cp2, template=Stat.GLMSELECT.Graphics.CoefficientPanel,
             adjust=tweak, overallanno=0)
```

Figure 7.26 Modified Coefficient Progression



This annotation data set is large. There are many variables, and varying subsets are used for each annotation. The following steps show the parts of the annotation data set that are used.

```
data anno2;
   set anno(keep=ID Function Label x1Space y1Space Anchor
                 Width TextSize TextWeight TextColor x1 y1);
   if _n_ = 1;
run;

proc print noobs;
   title 'Observation 1 - Text';
run;

data anno2;
   set anno(keep=ID Function x1Space y1Space x2Space y2Space
            xC1 xC2 LineThickness DiscreteOffset LineColor y1 y2);
   if _n_ = 2;
run;

proc print noobs;
   title 'Observation 2 - Line';
run;

data anno2;
   set anno(keep=ID Function x1Space y1Space Anchor xc1 Width Height WidthUnit
            CornerRadius LineThickness DiscreteOffset LineColor y1);
```

```
   if _n_ = 3;
run;

proc print noobs;
   title 'Observation 3 - Rectangle';
run;

data anno2;
   set anno(keep=ID Function Label x1Space y1Space Anchor
                 xc1 Width TextSize TextWeight TextColor y1);
   if _n_ = 4;
run;

proc print noobs;
   title 'Observation 4 - Text';
run;

data anno2;
   set anno(keep=ID Function x1Space y1Space x2Space y2Space Direction
            xc1 xc2 Scale LineThickness DiscreteOffset LineColor y1 y2);
   if _n_ = 5;
run;

proc print noobs;
   title 'Observation 5 - Arrow';
run;
```

The results are displayed in Figure 7.27.

Figure 7.27 Key Parts of the Annotation Data Set

Observation 1 - Text

ID	Function	Label	x1Space	y1Space	Anchor	Width	TextSize	TextWeight	TextColor	x1	y1
LO1	Text	Coefficients for the Selected Model	DataPercent	DataPercent	Right	100	7	Bold	Green	55	94

Observation 2 - Line

ID	Function	x1Space	y1Space	x2Space	y2Space	xC1	xC2	Line Thickness	Discrete Offset	Line Color	y1	y2
LO1	Line	DataValue	DataPercent	DataValue	DataPercent	9+CrBB	8+CrRuns*CrRuns	0.7	-0.3	Green	94	94

Observation 3 - Rectangle

ID	Function	x1Space	y1Space	Anchor	xC1	Width	WidthUnit	Corner Radius	Line Thickness	Discrete Offset	Line Color	y1	Height
LO1	Rectangle	DataValue	WallPercent	BottomLeft	9+CrBB	0.6	Data	0.8	0.7	-0.3	Green	10	80

Observation 4 - Text

ID	Function	Label	x1Space	y1Space	Anchor	xC1	Width	TextSize	TextWeight	TextColor	y1
LO3	Text	Selected Value	DataPercent	DataPercent	Left	9+CrBB	100	7	Bold	Blue	84

Observation 5 - Arrow

ID	Function	x1Space	y1Space	x2Space	y2Space	Direction	xC1	xC2	Scale	Line Thickness	Discrete Offset	Line Color	y1	y2
LO3	Arrow	DataValue	DataPercent	DataValue	DataPercent	In	9+CrBB	12+CrHits*CrHits	1E-12	0.7	0.1	Blue	4	83

Observation 1 positions a text string in the LAYOUT OVERLAY labeled `'LO1'`, which specifies coordinates that are based on the percentage of the data area. The string is anchored on the right, next to the line.

Observation 2 draws a line in the LAYOUT OVERLAY labeled `'LO1'`. The X coordinates are in the space `'DataValue'`. Because the X-axis variable is a character variable, the variables xC1 and xC2 are used. When the variables x1 and x2 exist for other observations, they must be set to missing for this observation. The Y coordinates are in the space `'DataPercent'`, and the variables y1 and y2 provide coordinates. Each pair of X and Y coordinates specifies one end of the line. The discrete offset of –0.3 moves the line 0.3 data units to the left of the coordinates that are specified in (xC1, y1) and (xC2, y2).

Observation 3 draws a rounded rectangle in the LAYOUT OVERLAY labeled `'LO1'`. There is only one set of coordinates. The X coordinates are in the data space, and the Y coordinates are in the wall percentage space. The rectangle is anchored in the bottom left (where drawing starts), and then it is drawn with a height of 80% of the wall and a width of 0.6 times the width of a discrete cell. The discrete offset of –0.3 moves the rectangle 0.3 data units to the left of the coordinates that are specified in (xC1, y1). The CornerRadius variable controls the degree of rounding. The result is a rounded rectangle that is centered around the reference line for the selected step.

Observation 4 positions a string in the LAYOUT OVERLAY labeled `'LO3'`. Notice that the layout has changed for this observation. The string is anchored on the left, next to the arrow.

Observation 5 draws an arrow in the LAYOUT OVERLAY labeled `'LO3'`. The X coordinates are in the space `'DataValue'`. Because the X-axis variable is a character variable, the variables xC1 and xC2 are used. When the variables x1 and x2 exist for other observations, they must be set to missing for this observation. The Y coordinates are in the space `'DataPercent'`, and the variables y1 and y2 provide coordinates. Each pair of X and Y coordinates specifies one end of the arrow. The discrete offset of 0.1 moves the arrow 0.1 data units to the right of the coordinates that are specified in (xC1, y1) and (xC2, y2). The Scale variable scales the size of the arrowhead, and the Direction variable points the arrow in (toward xC1 and y1).

For a list of all annotation functions and variables, see the section "SG Annotation Functions, Variables, and Their Values" on page 133. That section also provides help in finding problems in annotation data sets.

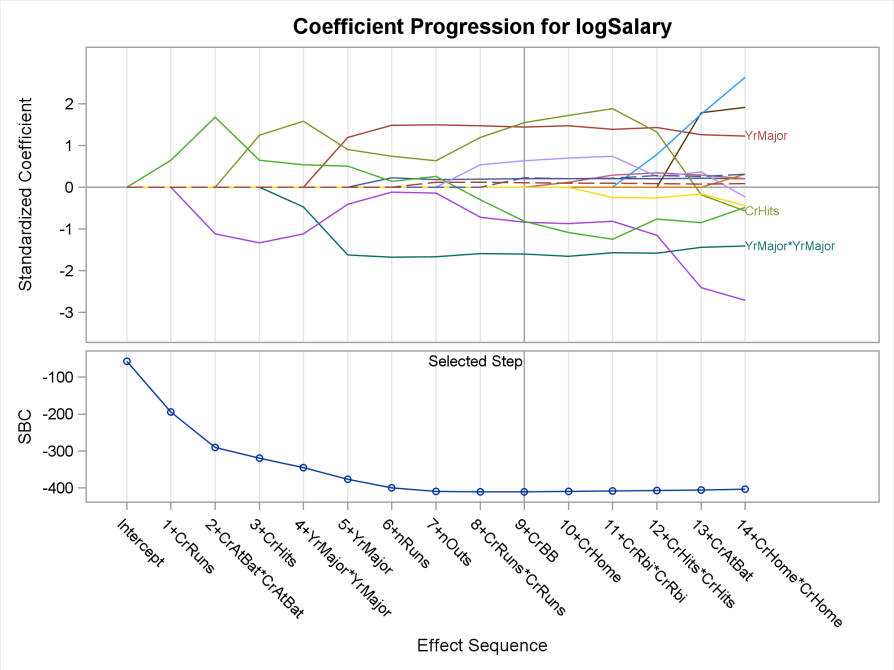
This example has one more part. This time, the annotations are disabled, and only labels that have standardized coefficients in the selected model outside the range –1 to 1 are displayed. The only template modification is the one that displays the labels as marker characters rather than as data labels. The following steps create the graph:

```
data labelthese(keep=parameter rename=(parameter=par));
   set cp end=eof;
   if eof then call symputx('_step', step);
   if step = 9 and (StandardizedEst gt 1 or StandardizedEst lt -1);
run;

data cp2;
   set cp;
   match = 0;
   if step ne &_step then return;
   do i = 1 to ntolabel;
      set labelthese point=i nobs=ntolabel;
      match + (par = parameter);
      end;
   if not match then parameter = ' ';
   if nmiss(rhslabelYvalue) then rhslabelYvalue = StandardizedEst;
run;
```

```
%macro tweak;
   if index(_infile_, 'datalabel=PARAMETER') then
      _infile_ = tranwrd(_infile_, 'datalabel',
                         'markercharacterposition=right markercharacter');
%mend;

%procannoadv(data=cp2, template=Stat.GLMSELECT.Graphics.CoefficientPanel,
             anno=, adjust=tweak)
```

The first step finds the terms that have standardized coefficients in the right ranges; it relies on knowing that PROC GLMSELECT selected the model found in step 9 as the final model. If you are writing a general purpose program to do this modification, you can process the __outdynam data set that the macro creates and output the value of the variable _ChosenValue. The second step, as before, adjusts the labels so that only the correct ones are displayed. In the macro call, the ANNO= option suppresses all annotation. The results are displayed in Figure 7.28

Figure 7.28 Larger Coefficients for the Chosen Model



Index