Basic ODS Graphics Examples

Warren F. Kuhfeld

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *Basic ODS Graphics Examples*. Cary, NC: SAS Institute Inc.

Basic ODS Graphics Examples

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

March 2016

Contents

Preface

Purpose

This book introduces the Graph Template Language (GTL) and the statistical graphics (SG) procedures SGPLOT, SGSCATTER, and SGPANEL. The GTL enables you to make a wide variety of modern statistical graphs by using a powerful language. The SG procedures enable you to make a more limited set of graphs by using a simpler but less powerful syntax.

Whenever possible, this book shows you how to make a graph two ways: by using an SG procedure and by using the GTL. These examples are designed to help you get started using both the SG procedures and the GTL. Understanding the simple GTL examples, even when they are harder to program than their SG counterpart, enables you to understand how to use the GTL for more complicated graphs and for modifying the templates used by the analytical procedures (such as those in the SAS/STAT, SAS/QC, and SAS/ETS products).

Is This Book for You?

If you are a SAS user and you need to create graphs, then this book is for you.

Scope of This Book

This book illustrates the major statements in the GTL and the SG procedures. It shows how to make a wide variety of statistical graphs. However, it does not discuss the underlying statistical methods, data analysis, or interpretation of results.

Software in This Book

- The GTL and SG procedures are part of Base SAS software.
- Some of the examples use SAS/STAT software to prepare data sets for display.

Sample Code

The beginning of every example (or every chapter for the short chapters) has a link that you can double-click to see the SAS code for that example. The examples in this preface are available from here:

Double-Click for Example Code

Code links work in Adobe Reader and Microsoft Internet Explorer but not in most other browsers. The sample code is embedded in this PDF file, so you can download the PDF file and then use it without Internet access. Some browsers cannot handle that. Code links might not be displayed if you print this book. You can enable code link printing in Adobe Reader by selecting **Edit ► Preferences ► Commenting** and checking **Print notes and pop-ups** before printing.

ODS Graphics

Effective graphics are indispensable for modern statistical analysis. They reveal patterns, differences, and uncertainty that are not readily apparent in tabular output. Graphics provoke questions that stimulate deeper investigation, and they add visual clarity and rich content to reports and presentations.

There are three major ways in which you can use statistical graphics:

- **Exploring your data.** Many statistical analyses begin by producing simple graphs of data, such as scatter plots, histograms, and box plots.
- **Analyzing your data.** Graphical results are an integral part of the results of modern statistical analyses.
- **Presenting your data.** The final results of statistical analyses are often presented by using graphs. Presentation graphics might be the same as the graphical results of analyses, or they might be customized in various ways.

The SAS System provides many tools for creating statistical graphics. In many cases, multiple tools can be used to make a graph.

- **Automatically created graphs.** Statistical graphs are created by using an extension of the Output Delivery System (ODS). ODS manages procedure output and lets you display it in a variety of destinations, such as HTML and RTF. When you use ODS Statistical Graphics (ODS Graphics for short), statistical procedures produce graphs as automatically as they produce tables, and graphs and tables appear together in the ODS output. You only need to enable ODS Graphics by using the `ods graphics on` statement to get default graphical output. Some graphs also require you to specify one or more simple options.
- **SG procedures.** The SG (statistical graphics) procedures SGPLOT, SGSCATTER, and SGPANEL provide a simple and convenient syntax for producing many types of statistical graphs. They are particularly convenient for exploring and presenting data.
- **GTL.** The Graph Template Language and PROC SGRENDER provide a powerful syntax for creating custom graphs. You can also modify the templates that SAS provides for use with SAS/STAT procedures to create customized results.

Table 1 provides a summary of ODS Graphics functionality.

Table 1 Summary of ODS Graphics Functionality

	Graphical task	Audience	What do you use?	What should you learn?	
✔	Create graphs in the context of statistical analyses	Statistical users	Statistical procedures in SAS/STAT, SAS/ETS, SAS/QC, and Base SAS that support ODS Graphics	Specify ODS GRAPHICS ON; graphs are then created by default or by procedure options, which are documented in the procedure chapters.	**Minimal graph syntax**
	Enhance specific graphs for a paper or presentation	Statistical and general SAS users	ODS Graphics Editor	How to request editable graphs, invoke the editor, use point-and-click features; see *SAS 9.4 ODS Graphics Editor: User's Guide*.	
✔	Create stand-alone graphs for data exploration or for customized displays	Statistical and general SAS users	SGPLOT, SGPANEL, SGSCATTER procedures in Base SAS	SG procedure syntax; see *SAS ODS Graphics: Procedures Guide*.	**Some syntax**
✔	Change the overall appearance of graphs and tables	Statistical and general SAS users	ODS styles	STYLE= option in ODS destination statement	
✔	Save and manage graphs for papers and presentations	Statistical and general SAS users	ODS GRAPHICS options, ODS destination options	How to specify size and resolution; how to name and access image files	
✔	Make persistent changes in graphs produced by statistical procedures (apply whenever you run your program)	Advanced SAS programmers	User modifications of graph templates that SAS provides	Basic features of the Graph Template Language and PROC TEMPLATE; see *SAS Graph Template Language: User's Guide*.	
	Create a highly customized stand-alone graph	Advanced SAS programmers	ODS Graphics Designer	GUI for creating graph templates	**Graphics programming**
✔	Create a highly customized stand-alone graph	Advanced SAS programmers	User-written graph templates	Graph Template Language, PROC TEMPLATE, and PROC SGRENDER; see *SAS Graph Template Language: User's Guide* and *SAS Graph Template Language: Reference*.	

✔ Discussed in this book.

The graphs that SAS/STAT procedures automatically produce are suitable for use in reports. However, you might want to customize them first (for example, by using the point-and-click ODS Graphics Editor, as described in *SAS 9.4 ODS Graphics Editor: User's Guide*). This book focuses primarily on the GTL, its use in creating customized graphs, and the SG procedures. Both simple and complex examples are provided to help you get started creating modern statistical graphics by using the GTL and SG procedures.[1]

ODS Graphics is available in procedures in SAS/STAT, Base SAS, SAS/ETS, SAS/QC, SAS Enterprise Miner, and other products. The following step creates the regression fit plot displayed in Figure 1 along with the ANOVA table and fit statistics in Figure 2:

```
ods graphics on;

proc glm data=sashelp.class;
   model weight = height;
quit;
```

Figure 1 Fit Plot Created by PROC GLM

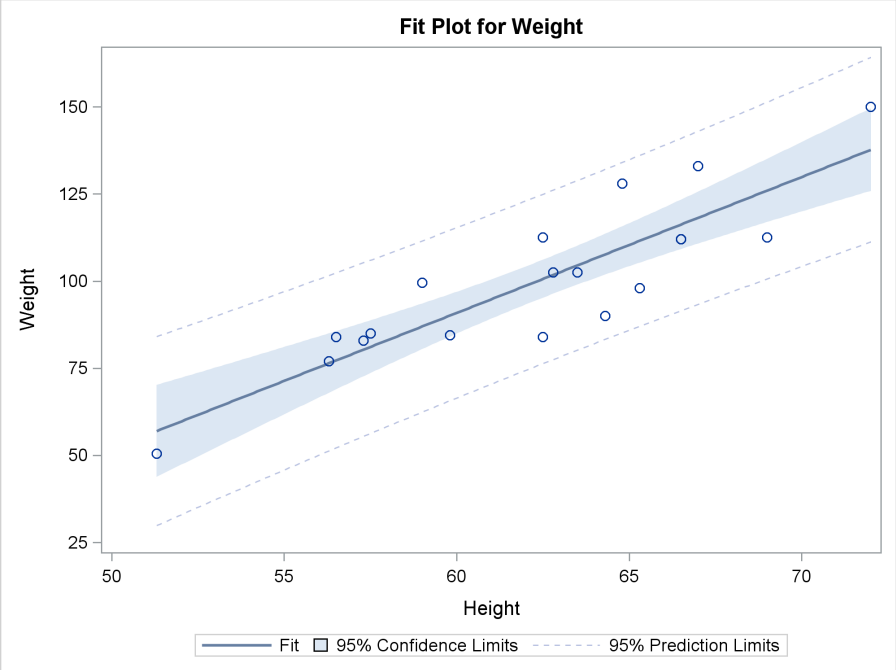


Figure 2 ANOVA and Fit Statistics

Dependent Variable: Weight

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	7193.249119	7193.249119	57.08	<.0001
Error	17	2142.487723	126.028690		
Corrected Total	18	9335.736842			

[1] Emphasis is placed on the code and the graphs, not on the data or the results.

Figure 2 *continued*

R-Square	Coeff Var	Root MSE	Weight Mean
0.770507	11.22330	11.22625	100.0263

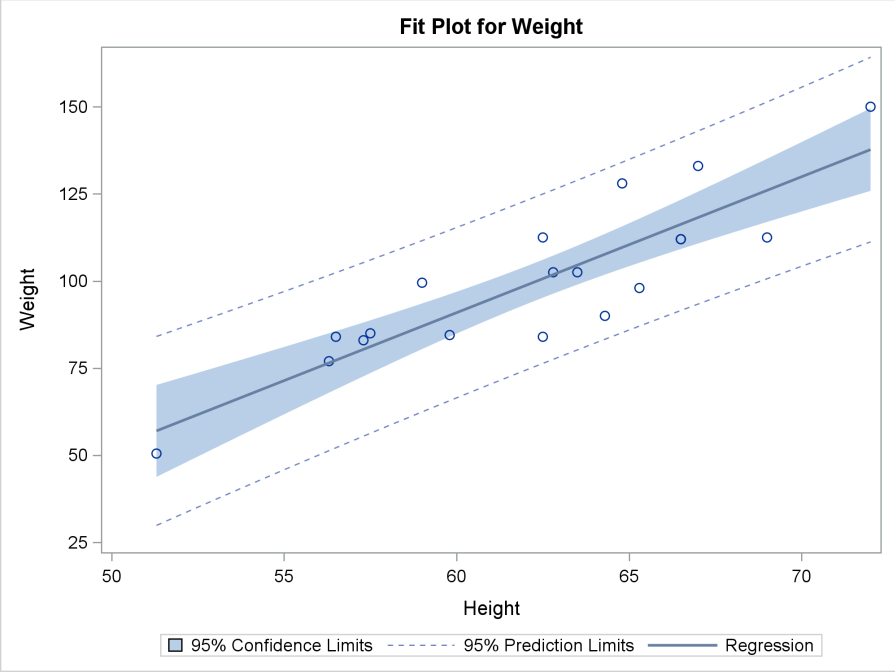
Basic ODS Graphics functionality is discussed in many places, including in "A Primer on ODS Statistical Graphics" (Chapter 21, *SAS/STAT User's Guide*). The documentation chapter for each SAS/STAT, SAS/QC, Base SAS, and SAS/ETS procedure that uses ODS Graphics provides examples, information about which graphs are produced by each procedure, and information about what syntax (if any) is required to make each plot.

ODS Graphics provides more than just a way to automatically produce graphs from analytical procedures. It also provides two powerful ways to produce custom graphs. First, there are the SG procedures: PROC SGPLOT, PROC SGSCATTER, and PROC SGPANEL.[2] They provide a high-level syntax for producing scatter plots, histograms, bar charts, box plots, scatter plot matrices, and many other types of statistical graphs. You can use these procedures instead of PROC GPLOT, PROC GCHART, and other legacy SAS/GRAPH procedures. The SG procedures are designed specifically for statistical work, and they have statistical computation facilities built directly into them. For example, the following step produces a scatter plot that has a linear regression line, confidence limits, and prediction limits and is very similar to the plot that PROC GLM produces:

```
proc sgplot data=sashelp.class;
   title 'Fit Plot for Weight';
   reg y=weight x=height / cli clm;
run;
```

The results are displayed in Figure 3.

Figure 3 Fit Plot Created by PROC SGPLOT



[2]ODS Graphics does not need to be enabled by specifying the ODS GRAPHICS ON statement in order for you to use the SG procedures or PROC SGRENDER.

The plots in Figure 1 and Figure 3 differ in the degree of transparency of the band that displays the confidence limits and in the legend entries.

In addition to the SG procedures, Base SAS provides you with the Graph Template Language, PROC TEMPLATE, and PROC SGRENDER.[3]

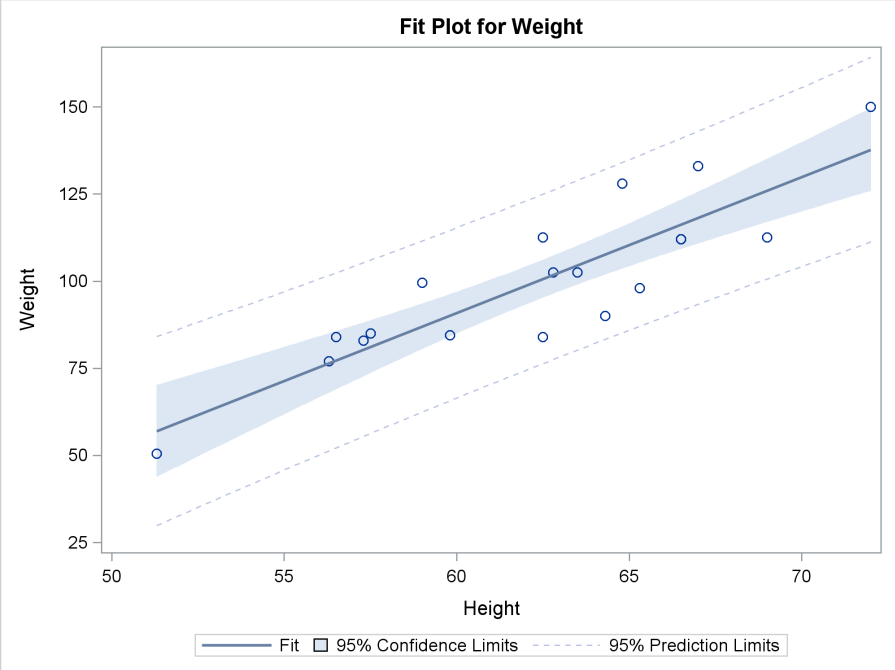
The GTL is a powerful language for defining the layout and composition of a graph or a panel of graphs. You can use it to make very simple and very complex graphs. It is the same language that SAS procedures use to define their ODS Graphics. SAS procedures that provide ODS Graphics use a template, written in the GTL, for each graph. You too can use the GTL and PROC TEMPLATE to create graph templates. Then you can use PROC SGRENDER to produce graphs from a SAS data set and the instructions provided in the graph template. For example, the following step produces a scatter plot that has a linear regression line, confidence limits, and prediction limits and is identical to the plot that PROC GLM produces:

```
proc template;
   define statgraph FitPlot;
      begingraph;
         entrytitle 'Fit Plot for Weight';
         layout overlay;
            modelband "cliband" / display=(outline)
                      outlineattrs=GraphPredictionLimits
                      name='cli' legendlabel='95% Prediction Limits'
                      datatransparency=0.5;
            modelband "clmband" /
                      fillattrs=GraphConfidence
                      name='clm' legendlabel='95% Confidence Limits'
                      datatransparency=0.5;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / name='reg'
                           clm="clmband" cli="cliband" legendlabel='Fit';
            discretelegend 'reg' 'clm' 'cli';
         endlayout;
      endgraph;
   end;
run;

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

The results are displayed in Figure 4. The graph template statements and options that are used in this example are explained in detail throughout the book.

[3]The procedure name "SGRENDER" begins with "SG"; however, the phrase "SG procedures" usually refers only to the SGPLOT, SGSCATTER, and SGPANEL procedures.

Figure 4 Fit Plot Created with the GTL



These three examples illustrate an important point. In most cases, you can simply enable ODS Graphics and use an analytical procedure to produce your graphs. When you need additional graphs or customized graphs, you can often use an SG procedure to get what you need. However, you can always use the GTL to specify precisely what you need. Using the GTL will almost always require more work, but it gives you the most power. The SG procedures and ODS Graphics are documented in *SAS ODS Graphics: Procedures Guide*, *SAS Graph Template Language: User's Guide*, and *SAS Graph Template Language: Reference*.

This book is organized by type of graph. Alternative ways of creating the same graph are presented together and in parallel. The book provides a gentle, parallel, and example-driven introduction to the SG procedures and to the GTL. Most graphs are produced in at least two ways. One graph is created by using the GTL. Whenever possible, the other graph is created more directly by using one of the SG procedures. Some graphs are also produced by SAS/STAT procedures or by using additional options. Each example provides prototype code for getting started using the SG procedures and the GTL. Examples of all the basic graphs that can be produced by the SGPLOT, SGSCATTER, and SGPANEL procedures are provided. Although you do not need to write a template to make many useful graphs, understanding the GTL enables you to create custom graphs that the SG procedures cannot produce. It also helps you modify the sometimes complex templates that SAS provides.

Examples of all the major statements in the GTL (excluding annotation statements) are provided. These statements can be classified as follows:

- Plot statements specify a number of commonly used displays, including scatter plots, histograms, contour plots, surface plots, and box plots.
- Layout statements specify the arrangement of the components of the graph.
- Text statements specify the descriptions that accompany graph elements.
- Control statements specify the conditional or iterative flow of control.

This book is about graphics, not statistics. Statistical graphics are created and displayed, but the book does not talk about interpretation or the complexities of data analysis. It uses simple data sets, often the Sashelp.Class data set that is familiar to most SAS users. Other data sets are used as needed. The goal is to show you how to make graphs, not explain why you would make them or what you would do with them.

Role of the GTL in SAS

It is important to understand that the GTL is a language developed for internal use at SAS. SAS procedure writers use it to make the graphs that are automatically produced by analytical procedures. You can use the same GTL to make your graphs. The GTL was developed to be a comprehensive language for capturing the definition of potentially very complex graphs. Although SAS makes this language available to you, it is not designed to be an obvious step away from syntax that is familiar to longtime SAS users. It is a new and different language. However, as you become experienced with the GTL, you will find that it is not difficult to use. You will simply find that it differs from what you might expect if you are an experienced SAS user. This is discussed in more detail in Appendix B, "Some Tips and Techniques for Understanding the Graph Template Language." If you find that your templates are not working the way you think they should, you might find the answer in this appendix.

Navigating This Book

Much of this book assumes that you are already familiar with using ODS Graphics together with SAS analytical procedures but are not familiar with the SG procedures or the GTL. If you are not familiar with the ODS Graphics statements, styles, destinations, and so on, see "Introduction to ODS Graphics" on page 181 for an introduction. You can find an even more detailed introduction in Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*).

Appendix B, "Some Tips and Techniques for Understanding the Graph Template Language," provides you with some background information about the GTL. It is important that you understand these concepts when you start writing your own templates. Sections in Chapter 1, "The Graph Template Language and the SG Procedures," illustrate ways to make different types of graphs and illustrate one or more statements in the GTL and in the SG procedures. The common statements (DEFINE, BEGINGRAPH, LAYOUT OVERLAY, and so on) are explained in detail only in the section "Scatter Plot" on page 3. Subsequent examples assume that you have read this first example. You should also read the section "Regression Fit Plot" on page 6 before proceeding to other examples. Other than

those sections, you do not need to read the sections in order. If you come across statements that are unfamiliar, you can use the index to find earlier examples of them.

The following list can help you find the section that illustrates each type of plot:

Resources

SAS offers you a rich variety of resources to help build your SAS skills and to explore and apply the full power of SAS software. Whether you are in a professional or academic setting, SAS provides learning products that can help you maximize your investment in SAS.

Bookstore	http://support.sas.com/publishing/
Training	http://support.sas.com/training/
Certification	http://support.sas.com/certify/
Knowledge Base	http://support.sas.com/resources/
Support	http://support.sas.com/techsup/
Learning Center	http://support.sas.com/learn/
Community	http://support.sas.com/community/
Documentation	http://support.sas.com/documentation/
Base SAS	http://support.sas.com/documentation/onlinedoc/base/
ODS	http://support.sas.com/documentation/onlinedoc/ods/
ODS Graphics	http://support.sas.com/documentation/prod-p/grstat/
SAS/STAT	http://support.sas.com/documentation/onlinedoc/stat/
Sanjay Matange	
Author Page	http://support.sas.com/publishing/authors/matange.html
Blog	http://blogs.sas.com/content/graphicallyspeaking

SAS Press books about ODS Graphics include the following:

- *Statistical Graphics Procedures by Example: Effective Graphs Using SAS* by Sanjay Matange and Dan Heath
- *Getting Started with the Graph Template Language in SAS: Examples, Tips, and Techniques for Creating Custom Graphs* by Sanjay Matange
- *Clinical Graphs Using SAS* by Sanjay Matange

Typographical Conventions Used in This Book

This book uses several type styles for presenting information. The following list explains the meaning of the typographical conventions used in this book:

roman	is the standard type style used for most text.
UPPERCASE ROMAN	is used for SAS options, and other SAS language elements when they appear in the text. However, you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
MonoSpace	is used for longer SAS options, statements, and statement fragments when they appear in the text. Lowercase or mixed case is usually used,

	but you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
oblique	is used for user-supplied values for options in the syntax definitions.
helvetica	is used for the names of variables and data sets when they appear in the text.
`monospace`	is used for example code. This book usually uses lowercase or mixed case for SAS code.

About This Edition

This book is a substantial revision of *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures* (last published in 2010) and is a companion to *Advanced ODS Graphics Examples* (2015). Both were written by the current author. This revision contains many new statements that have been added to PROC SGPLOT. They are all covered in Chapter 1, "The Graph Template Language and the SG Procedures." Chapter 3, "Style Templates," and Chapter 4, "Graph Template Modification," are now shorter because these topics are well covered in Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*), and Chapter 22, "ODS Graphics Template Modification" (*SAS/STAT User's Guide*). For more information about ODS and ODS Graphics, see Chapter 20, "Using the Output Delivery System" (*SAS/STAT User's Guide*), and Chapter 23, "Customizing the Kaplan-Meier Survival Plot" (*SAS/STAT User's Guide*).

This book was written for the third maintenance release of SAS 9.4.

About the Author

Warren F. Kuhfeld is a Distinguished Research Statistician Developer in SAS/STAT Research and Development at SAS. 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 software since 1979 and has developed SAS procedures since 1984.

In addition to writing the companion book, *Advanced ODS Graphics Examples*, Warren has written 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*), and Chapter 23, "Customizing the Kaplan-Meier Survival Plot" (*SAS/STAT User's Guide*).

Warren maintains 11 SAS/STAT procedures. He has developed 20 SAS macros for experimental designs for linear and choice models. His 1,300-page book about discrete choice and other marketing research methods, *Marketing Research Methods in SAS*, is available 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`.

First Edition Acknowledgments

I would like to thank Bob Rodriguez and Bob Derr for their helpful comments on an earlier draft of this book. I would also like to thank my editors, George McDaniel and Caroline Brickley. Finally, I would like to acknowledge my colleague Jeff Cartier, who sadly passed away shortly after the first draft of this book was written. Jeff wrote much of the documentation for ODS Graphics that SAS/STAT developers such as me used to learn the GTL.

Second Edition Acknowledgments

I would like to thank my editors, Ed Huddleston and Anne Baxter, who always find ways 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

The Graph Template Language and the SG Procedures

Contents

This book provides an introduction to the statistical graphics (SG) procedure SGPLOT and the Graph Template Language (GTL). Creating a graph by using PROC SGPLOT is a one-step process. You call PROC SGPLOT and provide one or more additional statements that describe the type of plot, the variables, and so on. Then ODS Graphics creates the plot, and you look at it in one or more SAS destinations as you would any other SAS output. Creating a graph by using the GTL is a two-step process. You use PROC TEMPLATE to define a graph template that includes multiple statements that describe the type of plot, the variables, and so on. You call PROC SGRENDER, specifying the template and the data. Then ODS Graphics creates the plot, and you look at the results. Most examples show you how to make a graph with PROC SGPLOT and then show you how to make the same graph by using PROC TEMPLATE, the GTL, and PROC SGRENDER. Some examples also show statistical procedures that make the same type of graph. If you can make the graph you need by using PROC SGPLOT or a statistical procedure, those methods will usually be easier than using the GTL. Throughout this book, both methods are shown together to help you understand simple GTL examples, even when the GTL is not needed, so that you can use it to easily construct more complicated graphs that are not available with PROC SGPLOT.

Sections are devoted to different types of plots, beginning with sections on the scatter plot and fit plots. The first few examples provide much greater detail than the later examples, so you should read the first few examples before moving on to subsequent examples. Specifically, the common GTL statements (DEFINE, BEGINGRAPH, LAYOUT OVERLAY, and so on) are explained in detail only in the section "Scatter Plot" on page 3. Subsequent examples assume that you have read this first example. You should also read the section "Regression Fit Plot" on page 6 before proceeding to other examples. Other than those sections, you do not need to read the sections in order. If you come across statements that are unfamiliar, you can use the index to find the first example of them. The earlier examples go into more detail about the basics, and the later examples reveal more of the nuances and power of the GTL and the SG procedures.

You can click on each of the following PROC SGPLOT statements or go to the page listed to see the example that introduces that statement:

1.1 Scatter Plot

Double-Click for Example Code

(Code links work in Adobe Reader and Internet Explorer.)

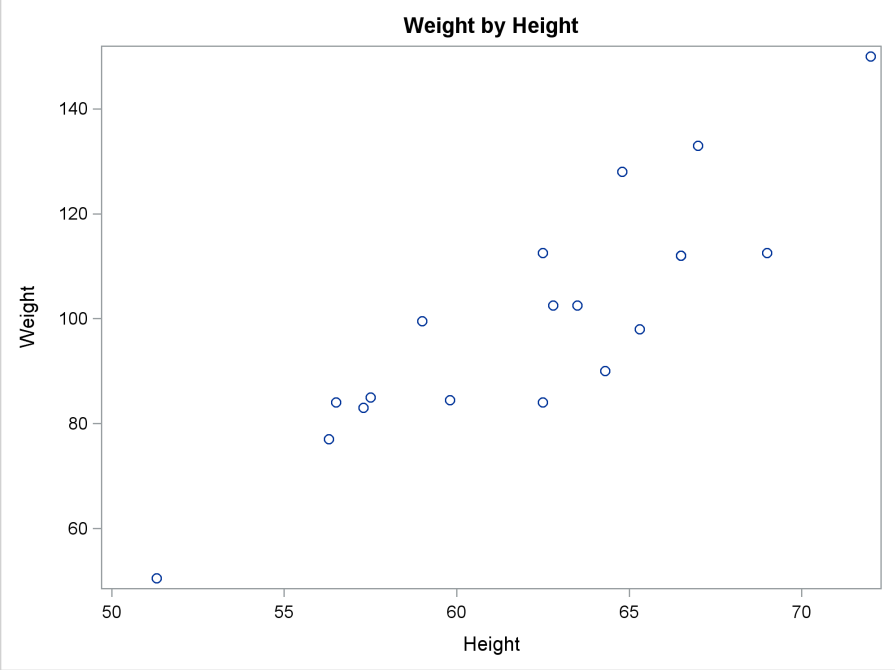
A scatter plot is a graphical display of two quantitative variables that uses Cartesian coordinates. The data are displayed as points, each having the value of one variable on the horizontal axis and the value of the other variable on the vertical axis.

You can use PROC SGPLOT to make a scatter plot as follows:

```
proc sgplot data=sashelp.class;
   title 'Weight by Height';
   scatter y=weight x=height;
run;
```

The input SAS data set is specified in the DATA= option, and the title is specified in the TITLE statement. The SCATTER statement constructs a plot that has the variable Weight on the Y axis and the variable Height on the X axis. The results are displayed in Figure 1.1.

Figure 1.1 Simple Scatter Plot



Alternatively, you can use PROC TEMPLATE and the GTL. The following step creates a graph template that can be used to create a scatter plot of the Sashelp.Class data set:

```
proc template;
   define statgraph classscatter;
      begingraph;
         entrytitle 'Weight by Height';
         layout overlay;
            scatterplot y=weight x=height;
         endlayout;
      endgraph;
   end;
run;
```

The step consists of a PROC TEMPLATE statement followed by three or more nested blocks of statements: DEFINE/END, BEGINGRAPH/ENDGRAPH, and LAYOUT/ENDLAYOUT. The outermost layer of a graph template begins with a DEFINE STATGRAPH statement and ends with an END statement. The final RUN statement is not required. Each template is compiled when SAS encounters the END statement that matches the DEFINE statement. This template, like all graph templates, begins with a DEFINE STATGRAPH statement followed by a template name. This step creates a graph template called **ClassScatter**. The BEGINGRAPH statement is next. It provides a place for you to specify options that affect the graph size, the graph border, and the graph background color. Most statements that construct the graph are specified inside the BEGINGRAPH/ENDGRAPH block. Exceptions include the DYNAMIC, MVAR, and NMVAR statements, which provide names for dynamic and macro variables that control details of how the graph is created. The graph title is specified in the ENTRYTITLE statement that follows the BEGINGRAPH statement. System titles and footnotes (specified in TITLE*n* and FOOTNOTE*n* statements) do not appear in the graph.

For a single graph such as this one (in contrast to a panel that contains two or more graphs), the next statement is usually a LAYOUT OVERLAY statement. It is accompanied by an ENDLAYOUT statement, and the statements that create the graph are in between. The LAYOUT OVERLAY statement provides a place for you to specify options that control the ticks, tick labels, axes, axis type, axis labels, grids, and so on. In this case, no options are specified.

The only graph statement is a SCATTERPLOT statement that plots the variable Weight on the Y axis and the variable Height on the X axis. You can use this template to create the scatter plot by using the following step:

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

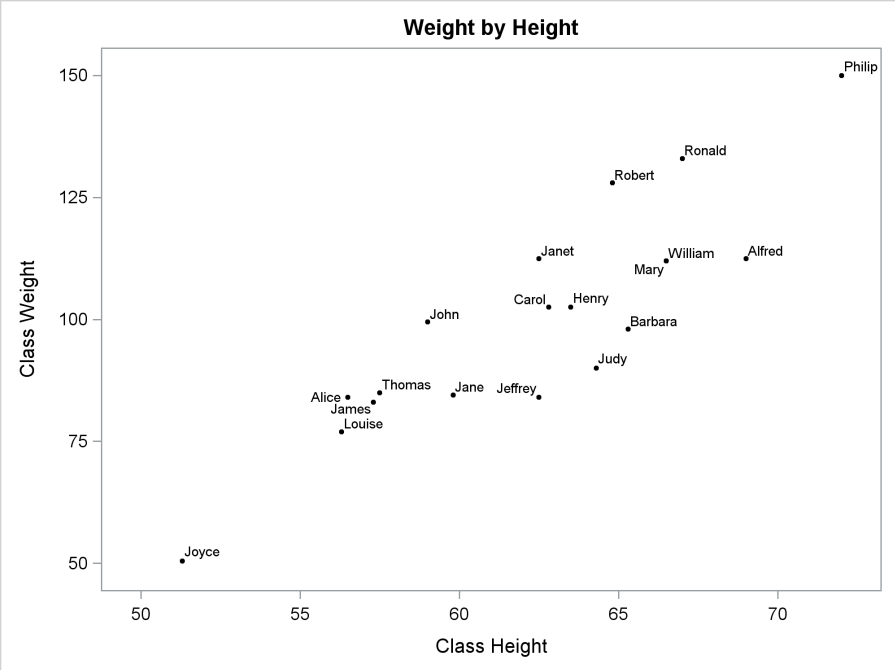
This PROC SGRENDER step consists of a DATA= option and a TEMPLATE= option. Some PROC SGRENDER steps are more involved, but most are this simple. All the other instructions are in the template. The results match those displayed in Figure 1.1.

The following step changes the axes and specifies other options:

```
proc sgplot data=sashelp.class;
   title 'Weight by Height';
   scatter y=weight x=height / datalabel=name
                               markerattrs=(symbol=circlefilled
                                            color=black size=3px);
   xaxis offsetmin=0.05 offsetmax=0.05 label='Class Height';
   yaxis offsetmin=0.05 offsetmax=0.05 label='Class Weight'
         values=(50 to 150 by 25);
run;
```

In PROC SGPLOT, the X- and Y-axis options are specified in the XAXIS and YAXIS statements, respectively. Offsets are specified so that the first 5% (OFFSETMIN=0.05) and the last 5% (OFFSETMAX=0.05) of each axis are left blank. Aesthetically, it is often good to have a small amount of separation between the axes and the extreme points in the plot. The axis label is specified by using the LABEL= option. The VALUES= option specifies the Y-axis ticks. The DATALABEL=Name option in the SCATTER statement uses the Name variable to label the points. The **`markerattrs=(symbol=circlefilled color=black size=3px)`** option specifies black filled circle markers, three pixels in size. The results are displayed in Figure 1.2.

Figure 1.2 Scatter Plot of Labeled Points



The following steps make a graph that is identical to the graph in Figure 1.2:

```
proc template;
   define statgraph classscatter;
      begingraph;
         entrytitle 'Weight by Height';
         layout overlay /
            xaxisopts=(offsetmin=0.05 offsetmax=0.05 label='Class Height')
            yaxisopts=(offsetmin=0.05 offsetmax=0.05 label='Class Weight'
                       linearopts=(tickvaluesequence=(start=50
                       end=150 increment=25) viewmin=50));
            scatterplot y=weight x=height / datalabel=name
                        markerattrs=(symbol=circlefilled
                                     color=black size=3px);
         endlayout;
      endgraph;
   end;
run;

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

In the GTL, X- and Y-axis options are specified in the LAYOUT OVERLAY statement in the XAXISOPTS= and YAXISOPTS= options, respectively. The tick values are specified using these options: `linearopts=(tickvaluesequence=(start=50 end=150 increment=25) viewmin=50))`. The LINEAROPTS= option is used for linear-axis options. Other axis types (log, discrete, and time) have different options. This set of options produces ticks from 50 to 150 by an increment of 25. However, ODS Graphics does not automatically use all the ticks if they are too far outside the range of the data. The VIEWMIN=50 option ensures that the smallest tick is displayed. Without this option, the first tick is not displayed for these data. There is also a VIEWMAX= option that is not used in this example.

The next section starts with a scatter plot and adds regression fit functions to it.

1.2 Regression Fit Plot

Double-Click for Example Code

A regression fit plot consists of a scatter plot of two quantitative variables along with an overlaid linear or nonlinear fit function. The Y coordinates of the fit function are often computed by using the method of least squares, although other methods are illustrated in later examples. You can use PROC SGPLOT to make a linear fit plot as follows:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Linear Regression';
   reg y=weight x=height;
run;
```

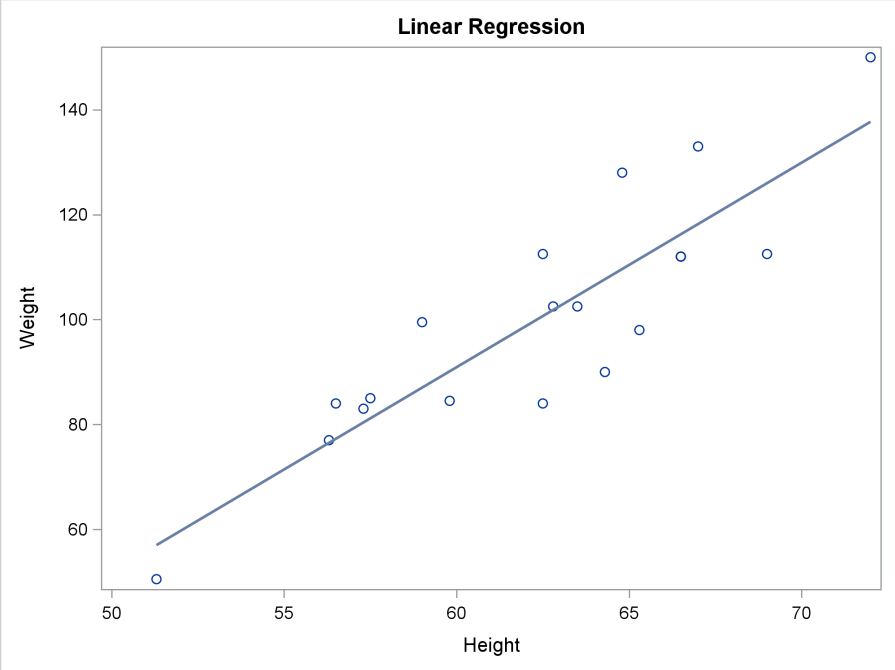
The results are displayed in Figure 1.3. The REG statement plots the variable Weight on the Y axis and the variable Height on the X axis. A single statement produces both the scatter plot and the fit line. The NOAUTOLEGEND statement in PROC SGPLOT suppresses the display of the

automatically generated legend. In a simple fit plot such as this, you do not need a legend to explain that the graph contains a scatter plot and a fit function.

The REG statement in PROC SGPLOT automatically specifies subpixel rendering (SUBPIXEL=ON). This option produces a smoother and less jagged fit function, at a cost of slower run times. For an example such as this, the extra run time is negligible. The following statement enables subpixel rendering for all subsequent examples (so that you do not need to specify it in any of the templates):

```
ods graphics on / subpixel=on;
```

Figure 1.3 Linear Fit Function



The following step creates a template that can be used to create a scatter plot of the Sashelp.Class data set along with a linear fit:

```
proc template;
   define statgraph classreg;
      begingraph;
         entrytitle 'Linear Regression';
         layout overlay;
            scatterplot    y=weight x=height;
            regressionplot y=weight x=height;
         endlayout;
      endgraph;
   end;
run;
```

The SCATTERPLOT statement plots the variable Weight on the Y axis and the variable Height on the X axis. The REGRESSIONPLOT statement fits a line through the scatter plot that is defined by the variable Weight on the Y axis and the variable Height on the X axis. The fit function is linear, because the default degree is 1. You can use this template and create the plot by using PROC SGRENDER:

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

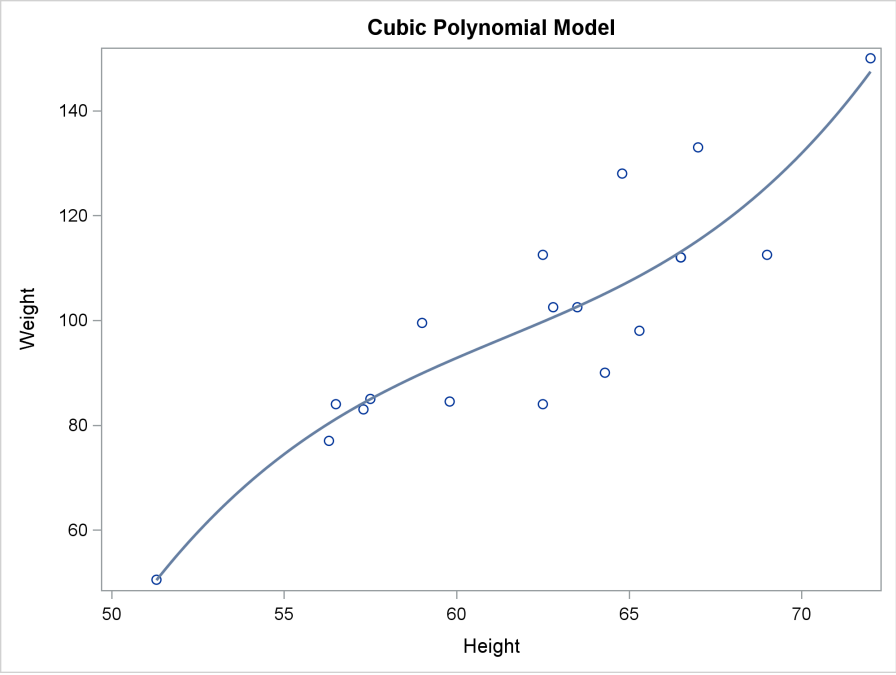
The results match those displayed in Figure 1.3.

You can specify the DEGREE=3 option for a cubic polynomial fit function through the points that are defined by the variables. You can specify DEGREE=2 for a quadratic fit function. The following steps use PROC SGPLOT and the GTL to create a cubic fit function:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Cubic Polynomial Model';
   reg y=weight x=height / degree=3;
run;

proc template;
   define statgraph classreg;
      begingraph;
         entrytitle 'Cubic Polynomial Model';
         layout overlay;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3;
         endlayout;
      endgraph;
   end;
run;

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

The graphs are identical; one is displayed in Figure 1.4.

Figure 1.4 Cubic Fit Function



The SG procedures work by writing a template in the GTL and using it to produce a graph. PROC SGPLOT has an option in the PROC statement, the TMPLOUT= option, that writes the generated template to a file. You can look at that template, use it with PROC SGRENDER, or modify it first and then use it with PROC SGRENDER.

The following step illustrates this option:

```
proc sgplot data=sashelp.class noautolegend tmplout='fittmplt.sas';
   title 'Cubic Fit Function';
   reg y=weight x=height / degree=3;
run;
```

The generated template is displayed in Figure 1.5, and it closely matches the template that is used in this example. The generated template usually specifies some options that you do not need in order to re-create the graph.

Figure 1.5 Fit Plot Generated Template

```
proc template;
define statgraph sgplot;
begingraph / collation=binary subpixel=on;
EntryTitle "Cubic Fit Function" /;
layout overlay / yaxisopts=(labelFitPolicy=Split) y2axisopts=(labelFitPolicy=Split);
   ScatterPlot X=Height Y=Weight / primary=true;
   RegressionPlot X=Height Y=Weight / NAME="REG" LegendLabel="Regression" Degree=3;
endlayout;
endgraph;
end;
run;
```

You can use SAS/STAT procedures to create fit plots when you want more explicit control over the model or more detailed output. Fit plots are created directly by several procedures, including the REG, GLM, and TRANSREG procedures. The following steps use PROC REG and PROC GLM to create fit plots. The results are displayed in Figure 1.6 and Figure 1.7.

```
proc reg data=sashelp.class;
   model weight = height;
quit;

proc glm data=sashelp.class;
   model weight = height;
quit;
```

Figure 1.6 PROC REG Fit Plot

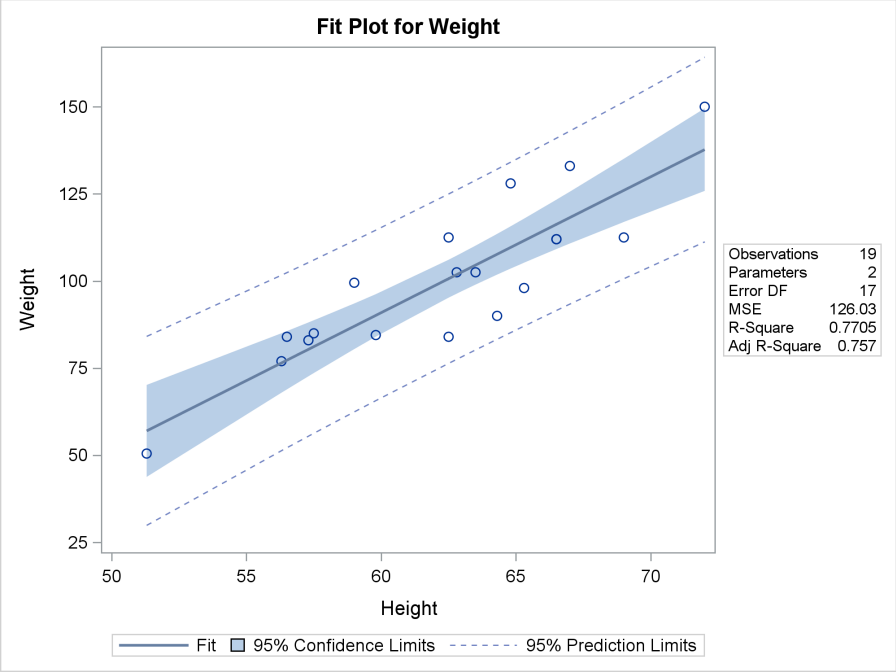
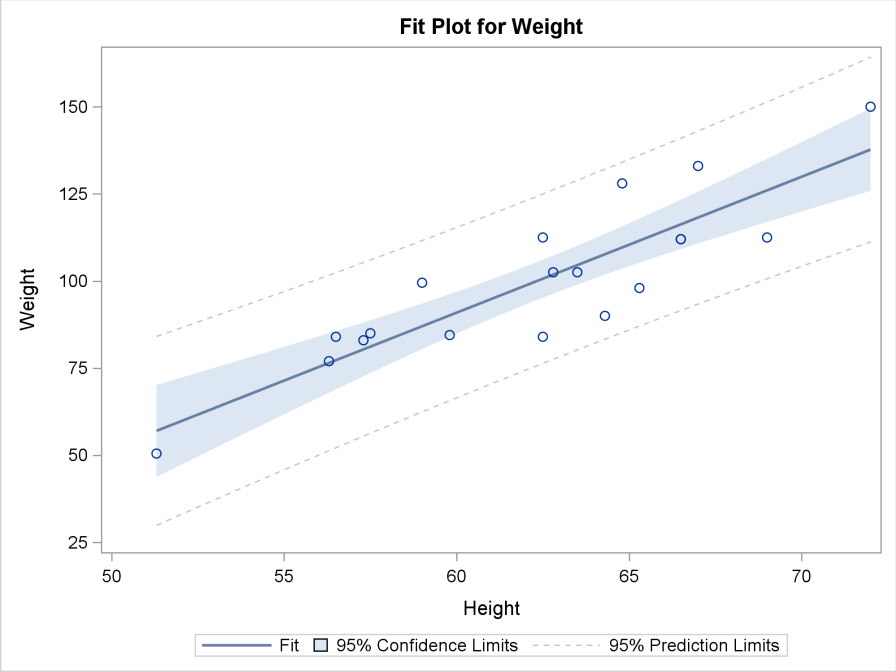


Figure 1.7 PROC GLM Fit Plot



1.2.1 Loess Fit Plot

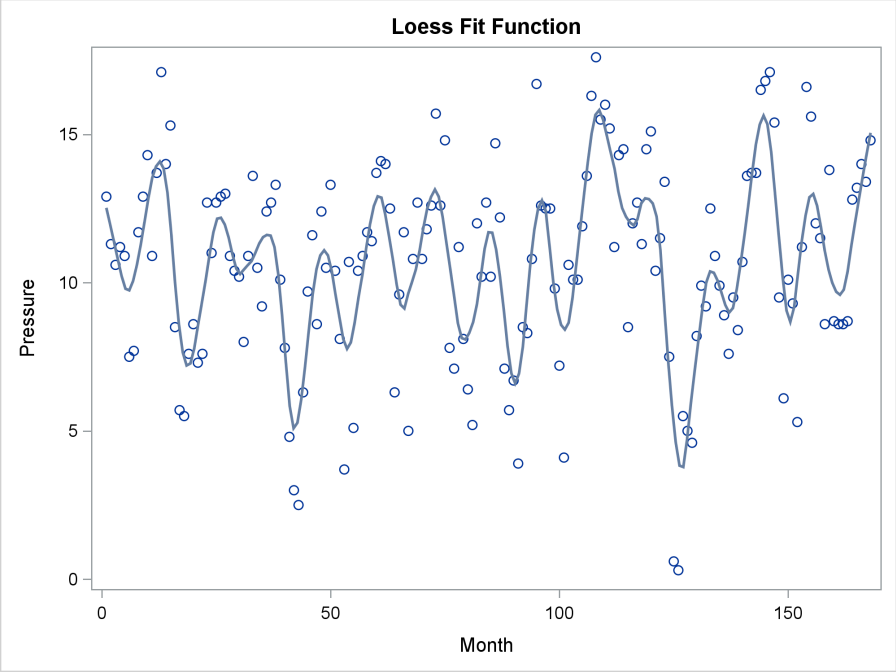
A loess fit plot consists of a scatter plot of two quantitative variables along with an overlaid nonlinear fit function. The loess fit function is found by using a nonparametric locally weighted scatter plot smoothing technique (Cleveland, Devlin, and Grosse 1988). The following steps show two ways to display a loess fit in a scatter plot:

```
proc sgplot data=sashelp.enso noautolegend;
   title 'Loess Fit Function';
   loess y=Pressure x=Month;
run;

proc template;
   define statgraph loess;
      begingraph;
         entrytitle 'Loess Fit Function';
         layout overlay;
            scatterplot y=Pressure x=Month;
            loessplot   y=Pressure x=Month;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.enso template=loess;
run;
```

The El Niño Southern Oscillation (ENSO) data set contains measurements of monthly averaged atmospheric pressure differences between Easter Island and Darwin, Australia, for a period of 168 months (National Institute of Standards and Technology 1998). The ENSO data set is particularly well suited for illustrating nonlinear fit functions. The data show both seasonal variations and variations caused by El Niño. The ENSO data set is available in the Sashelp library.

The LOESS statement in PROC SGPLOT and the LOESSPLOT statement in the GTL fit a nonparametric regression function through the points that are defined by the variable Pressure on the Y axis and the variable Month on the X axis. The NOAUTOLEGEND statement is used in PROC SGPLOT to suppress the display of the automatically generated legend. The graphs are identical; one is displayed in Figure 1.8.

Figure 1.8 Loess Fit Plot



You can use PROC LOESS to create a loess fit plot when you want more explicit control of the results and when you are interested in a detailed statistical analysis of your data rather than simply a graph with a fit function. You can compute a loess fit for these data in two different ways by using the following steps:

```
proc loess data=sashelp.enso;
   model Pressure=Month;
run;

proc loess data=sashelp.enso;
   model Pressure=Month / select=AICC(global);
run;
```

The results of these steps are displayed in Figure 1.9 and Figure 1.10. The plot in Figure 1.9 displays a local optimum (El Niño cycle), and the plot in Figure 1.10 displays the global optimum (seasonal variation). PROC LOESS finds the local optimum by default and needs some nudging to find the global optimum. For more information about loess and this example, see Chapter 71, "The LOESS Procedure" (*SAS/STAT User's Guide*).

Figure 1.9 Locally Optimal Loess Fit

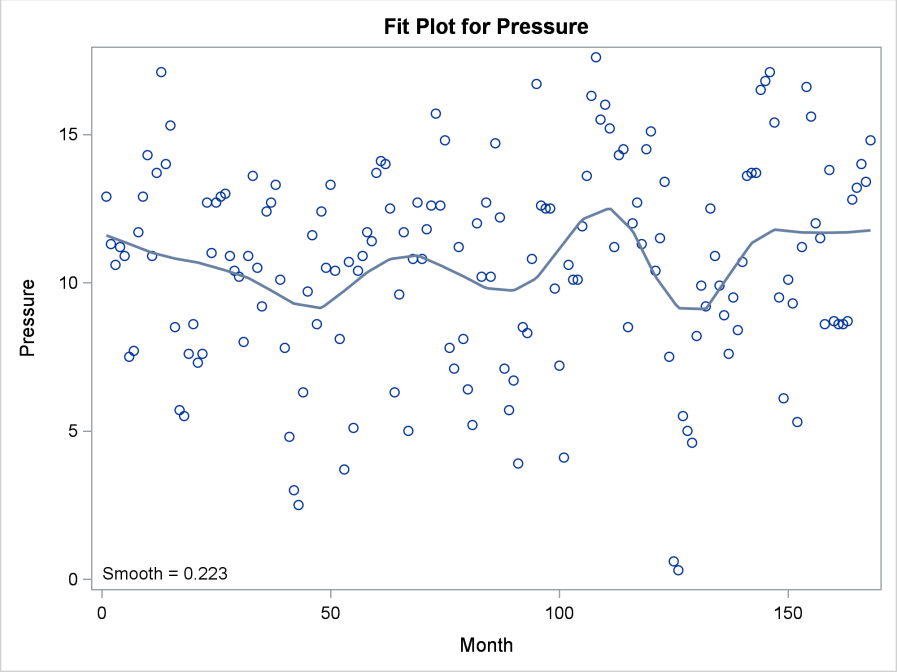
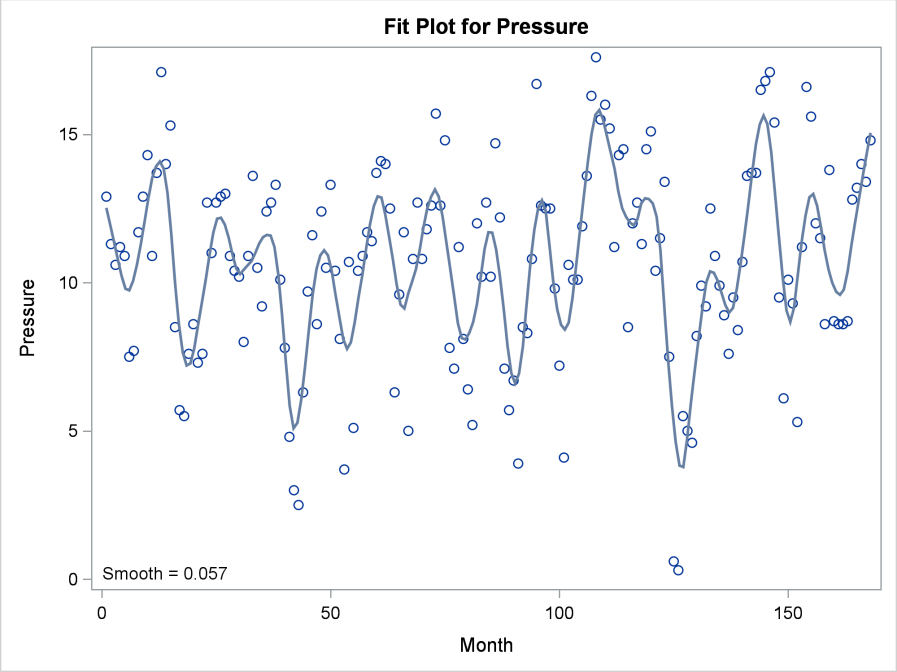


Figure 1.10 Globally Optimal Loess Fit



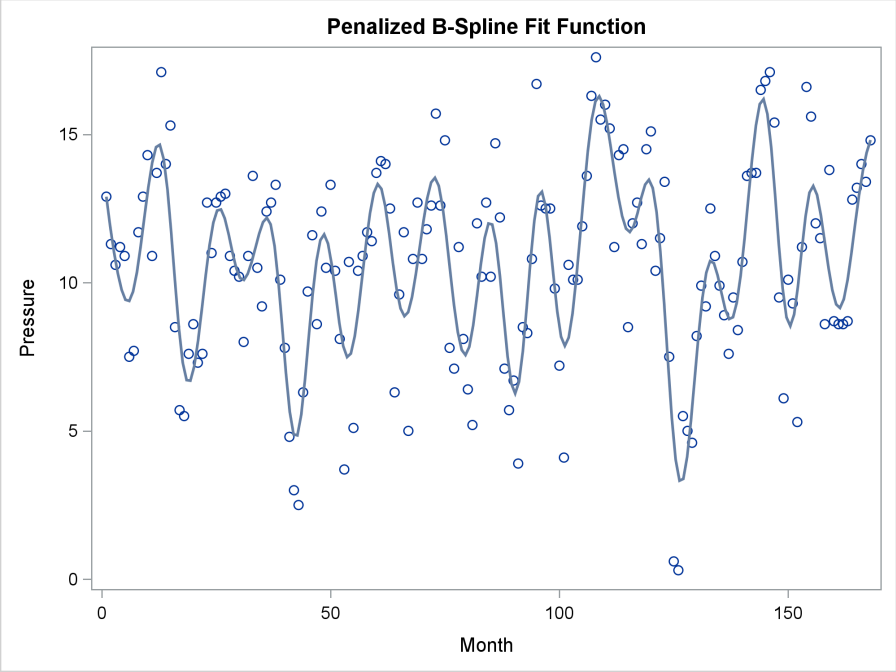
1.2.2 Penalized B-Spline Fit Plot

A penalized B-spline fit plot consists of a scatter plot of two quantitative variables along with an overlaid nonlinear fit function. The penalized B-spline fit function is found by using a flexible method (Eilers and Marx 1996) that automatically chooses the smoothing parameter that minimizes the corrected AIC criterion (AICC). The following steps show two ways to make a penalized B-spline fit plot:

```
proc sgplot data=sashelp.enso noautolegend;
   title 'Penalized B-Spline Fit Function';
   pbspline y=Pressure x=Month;
run;

proc template;
   define statgraph pbs;
      begingraph;
         entrytitle 'Penalized B-Spline Fit Function';
         layout overlay;
            scatterplot  y=Pressure x=Month;
            pbsplineplot y=Pressure x=Month;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.enso template=pbs;
run;
```

The PBSPLINE statement in PROC SGPLOT and the PBSPLINEPLOT statement in the GTL fit a smooth function with an automatically chosen smoothing parameter through the points that are defined by the variable Pressure on the Y axis and the variable Month on the X axis. The graphs are identical; one is displayed in Figure 1.11.

Figure 1.11 Penalized B-Spline Fit Plot



You can use PROC TRANSREG to create a penalized B-spline fit plot when you want more explicit control over the results and when you are interested in a detailed statistical analysis of your data rather than simply a graph with a fit function. You can compute a penalized B-spline fit for these data in two different ways by using the following steps:

```
proc transreg data=sashelp.enso;
   model identity(pressure) = pbspline(month);
run;

proc transreg data=sashelp.enso;
   model identity(pressure) = pbspline(month / sbc lambda=2 10000 range);
run;
```

The results of these steps are displayed in Figure 1.12 and Figure 1.13. The plot in Figure 1.12 displays the global optimum (seasonal variation), and the plot in Figure 1.13 displays a local optimum (El Niño cycle). PROC TRANSREG finds the global optimum by default and needs some nudging away from small smoothing parameters to find the local optimum. For more information about penalized B-splines and this example, see Chapter 117, "The TRANSREG Procedure" (*SAS/STAT User's Guide*).

Figure 1.12 Globally Optimal Penalized B-Spline Fit

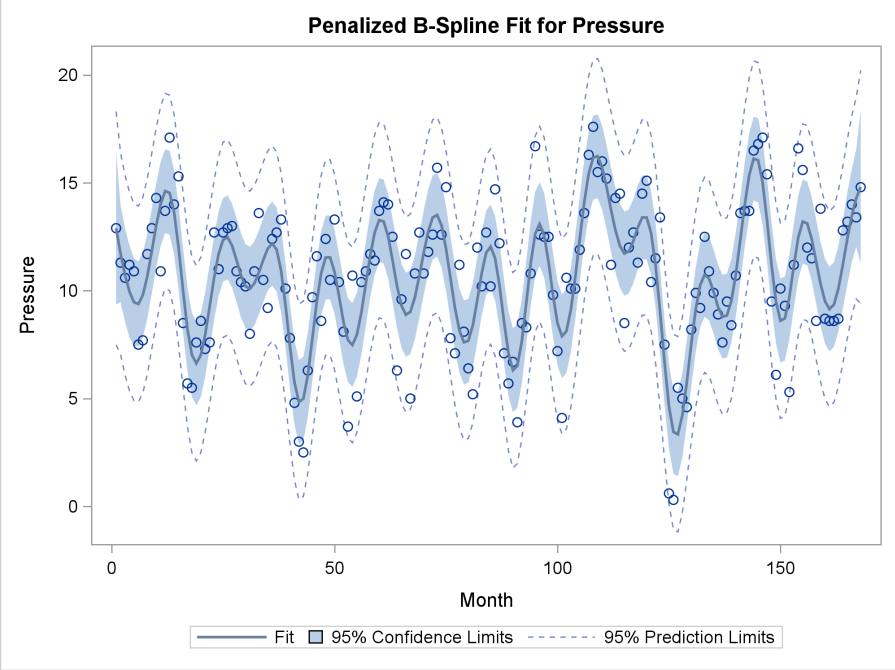
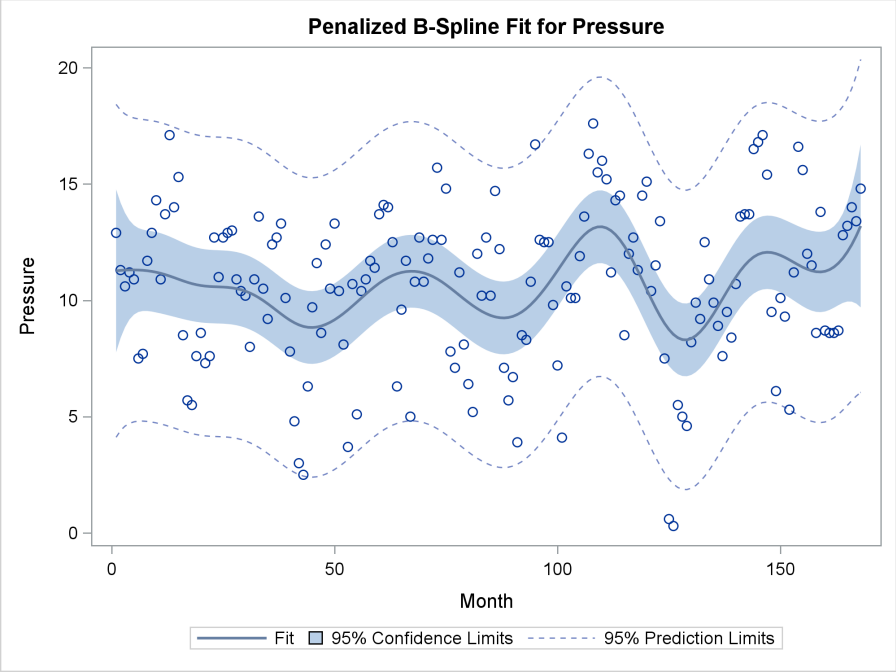


Figure 1.13 Locally Optimal Penalized B-Spline Fit



1.2.3 Grouped Regression Fit Plot

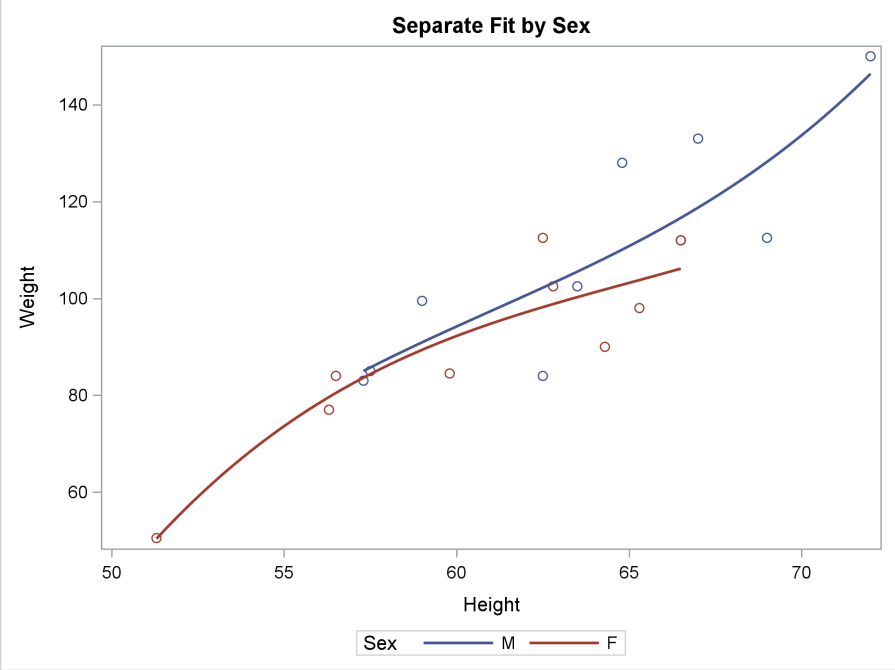
Previous examples showed several types of fit plots that have a single quantitative dependent variable and a single quantitative independent variable. This example uses a model that also has a classification or group variable, and fits a model that has separate intercepts and functions for each group. The following steps show two ways to produce a graph, each displaying a separate cubic fit function for males and for females:

```
proc sort data=sashelp.class out=class;
   by descending sex;
run;

proc sgplot data=class;
   title 'Separate Fit by Sex';
   reg y=weight x=height / group=sex degree=3;
run;

proc template;
   define statgraph classgroup;
      begingraph;
         entrytitle 'Separate Fit by Sex';
         layout overlay;
            scatterplot    y=weight x=height / group=sex;
            regressionplot y=weight x=height / group=sex degree=3
                                               name='reg';
            discretelegend 'reg' / title='Sex';
         endlayout;
      endgraph;
   end;
run;
```

```
proc sgrender data=class template=classgroup;
run;
```

The graphs are identical; one is displayed in Figure 1.14.

Figure 1.14 Regression with a Group Variable



Because of the GROUP= variable in the REG and REGRESSIONPLOT statements, separate computations are performed for each group. Because of the GROUP= variable in the REG, SCATTERPLOT, and REGRESSIONPLOT statements, the two groups of observations are displayed differently according to the rules specified in the ODS style. With the HTMLBlue style, which is the default style in this book, members of the first group (males) are displayed as blue circles with a solid blue fit function, and members of the second group (females) are displayed as red circles with a red solid fit function. Males are sorted ahead of females because of the DESCENDING option in the BY statement in PROC SORT.

The GTL and PROC SGRENDER require the input data set to be sorted by the group variable. In PROC SGPLOT, the GROUP= option in the REG statement creates the proper subsets of data without requiring you to sort first. Both methods use the sorted data in this example to ensure the same grouping for both methods.

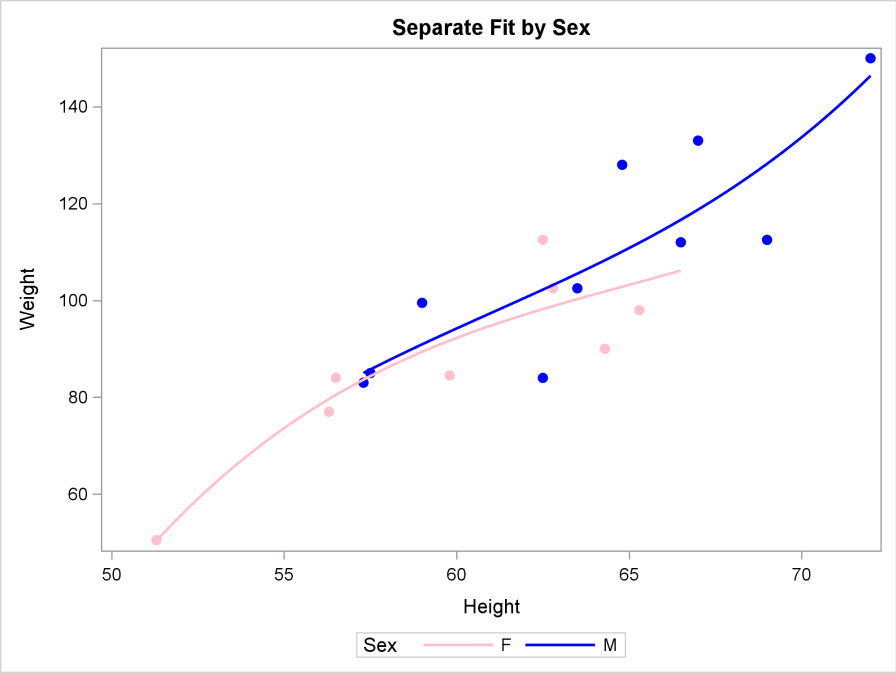
The REGRESSIONPLOT statement is given a name with the NAME='reg' option. That name is specified in the DISCRETELEGEND statement to produce the legend. A different name could also have been provided in the SCATTERPLOT statement, and that name could also have been specified in the DISCRETELEGEND statement. However, with a fit plot, the fit functions are the same colors as their corresponding points, so the legend for the REGRESSIONPLOT statement is sufficient. The TITLE= option in the DISCRETELEGEND statement provides a title for the legend.

This example shows how you can change the group colors:

```
proc sort data=sashelp.class out=class;
   by sex;
run;

proc sgplot data=class;
   title 'Separate Fit by Sex';
   styleattrs datacontrastcolors=(pink blue) datasymbols=(circlefilled);
   reg y=weight x=height / group=sex degree=3;
run;

proc template;
   define statgraph classgroup;
      begingraph / datacontrastcolors=(pink blue) datasymbols=(circlefilled);
         entrytitle 'Separate Fit by Sex';
         layout overlay;
            scatterplot    y=weight x=height / group=sex;
            regressionplot y=weight x=height / group=sex degree=3
                                               name='reg';
            discretelegend 'reg' / title='Sex';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=class template=classgroup;
run;
```

The graphs are identical; one is displayed in Figure 1.15. The data are sorted by ascending values of the variable Sex. Colors are specified in the STYLEATTRS statement in PROC SGPLOT and in the BEGINGRAPH statement in the GTL. The color pink is used for the first group (females), and the color blue is used for the second group (males). The markers are changed to filled circles because pink (which is a light color) tends to fade into the background when it is used for open circles.

Figure 1.15 Controlling the Appearance with a Group Variable

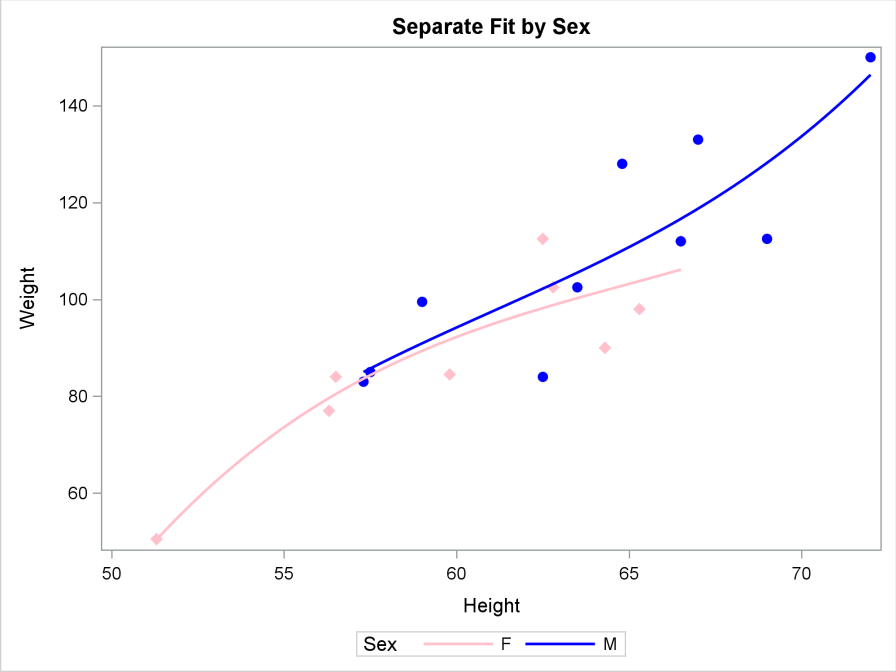


In the previous graphs, the same marker was used for males and females. This example shows how you can vary the markers:

```
ods graphics on / attrpriority=none subpixel=on;

proc sort data=sashelp.class out=class;
   by sex;
run;
proc sgplot data=class;
   title 'Separate Fit by Sex';
   styleattrs datacontrastcolors=(pink blue)
              datalinepatterns=(solid)
              datasymbols=(diamondfilled circlefilled);
   reg y=weight x=height / group=sex degree=3;
run;

proc template;
   define statgraph classgroup;
      begingraph / datacontrastcolors=(pink blue)
                   datalinepatterns=(solid)
                   datasymbols=(diamondfilled circlefilled);
         entrytitle 'Separate Fit by Sex';
         layout overlay;
            scatterplot    y=weight x=height / group=sex;
            regressionplot y=weight x=height / group=sex degree=3
                                               name='reg';
            discretelegend 'reg' / title='Sex';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=class template=classgroup;
run;

ods graphics on / reset=all subpixel=on;
```

The graphs are identical; one is displayed in Figure 1.16.

Figure 1.16 Varying Markers by Using a Group Variable



Styles have varying attribute priorities. An ATTRPRIORITY=NONE style uses a different combination of colors, markers, and line patterns to distinguish every group. An ATTRPRIORITY=COLOR style uses only colors to distinguish the first 12 groups. Most styles (such as Default, Statistical, and Analysis) are ATTRPRIORITY=NONE styles. A few styles (such as HTMLBlue, Pearl, and Sapphire) are ATTRPRIORITY=COLOR styles. You can specify the ATTRPRIORITY= option in the ODS GRAPHICS statement to change the attribute priority of any style. If the ATTRPRIORITY=NONE option had not been specified in the ODS GRAPHICS statement for this style (HTMLBlue), the first marker (the filled diamond) would have been used for both groups.

When you use the ATTRPRIORITY=NONE option, the first line is solid and the second line is dashed by default. To vary markers and colors but not line patterns, specify a single line pattern. To vary line patterns and colors but not markers (not shown), specify a single marker.

When you have grouped data, the combination of ATTRPRIORITY=NONE and DATALINEPATTERNS=(SOLID) usually produces nice graphs:

```
ods graphics on / attrpriority=none subpixel=on;

proc sgplot data=class;
   styleattrs datalinepatterns=(solid);
   . . .
run;
```

Points on the graph are more easily distinguished when markers vary, but varying line patterns are often not necessary in order to distinguish the different functions.

If you want your regression line to extend the length of the plot, you can use the LINEPARM statement. The following steps create the graphs in Figure 1.17 and Figure 1.18:

```
ods graphics on / attrpriority=none;
title;

proc reg data=sashelp.iris noprint
         outest=s(rename=(petallength=Slope species=s)
                  keep=petallength int: species);
   model sepallength = petallength;
   by species;
quit;

data iris;
   merge sashelp.iris s;
run;

proc sgplot;
   title 'Separate Fit by Sex';
   styleattrs datasymbols=(circlefilled squarefilled trianglefilled)
              datalinepatterns=(solid);
   lineparm x=0 y=intercept slope=slope / group=s lineattrs=(thickness=2)
                                          nomissinggroup;
   scatter  y=sepallength x=petallength / group=species;
   xaxis min=10;
run;

proc sgplot;
   title 'Separate Fit by Sex';
   styleattrs datasymbols=(circlefilled squarefilled trianglefilled)
              datalinepatterns=(solid);
   reg y=sepallength x=petallength / group=species;
run;

ods graphics on / reset=attrpriority;
```

The results in Figure 1.17 are created by PROC REG and are displayed by the LINEPARM statement. The results in Figure 1.18 are directly computed and displayed by the REG statement. The MERGE statement merges a data set that has 150 observations (Sashelp.Iris) and a data set that has 3 observations (S). The Intercept, Slope, and S variables have 147 missing values in the merged data set. The LINEPARM statement uses the 3 observations that have a nonmissing slope and intercept to make the three regression lines. The NOMISSINGGROUP option ensures that a fourth group is not created for missing values of S.

Figure 1.17 Regression Lines Produced by the LINEPARM Statement

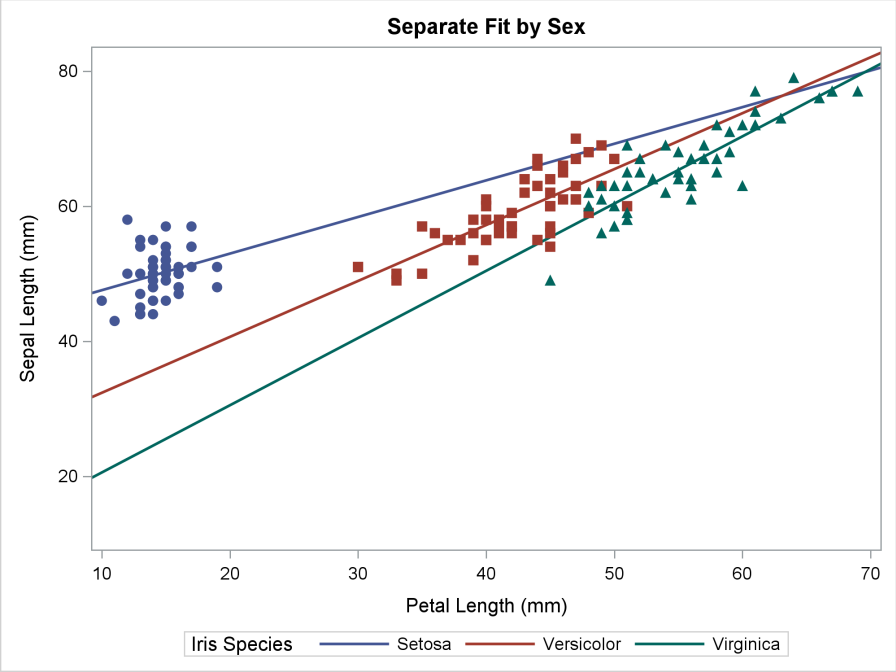
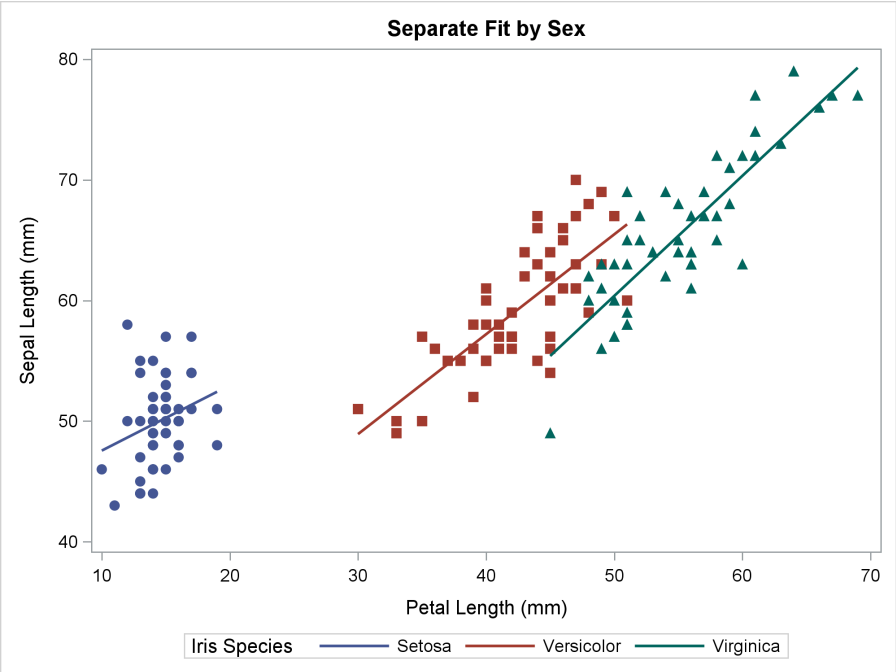


Figure 1.18 Regression Lines Produced by the REG Statement



1.2.4 Regression Fit Plot with Confidence and Prediction Limits

This example introduces band plots and series plots. Band plots display confidence limits as a filled band. A series plot is a series of line segments that connect consecutive points. You can use series plots to display predicted values, confidence limits, prediction limits, and so on. The following steps illustrate how much SAS/STAT procedures that create ODS Graphics automatically do for you, how much PROC SGPLOT automatically does for you, and how much you have to do for yourself if you want to create the same or similar plots by explicitly specifying each of the graph components in either PROC TEMPLATE or PROC SGPLOT. These explicit-specification examples provide an opportunity for you to understand statements and compare different ways of doing things; they do not provide models for actual graph construction.

The following steps show two ways to create a cubic fit plot with confidence and prediction limits:

```
proc transreg data=sashelp.class;
   model identity(weight) = spline(height);
   output out=class clm cli p;
run;

proc sgplot data=sashelp.class;
   title 'Cubic Fit';
   reg y=weight x=height / degree=3 clm cli;
run;
```

The graph in Figure 1.19 is created automatically by PROC TRANSREG when ODS Graphics is enabled. Only the PROC and MODEL statements are required. The OUTPUT statement and its options create an output data set that is used in subsequent steps, but they are not required in order to make the fit plot. The graph in Figure 1.20 is created by PROC SGPLOT. The CLM option (confidence limits for mean predicted values) produces the inner band, which shows the confidence limits. The CLI option (confidence limits for the individual predicted values) in the REG statement produces the outer series plots (or equivalently, the outer unfilled band plot), which show the prediction limits.

Figure 1.19 Cubic Fit Plot from PROC TRANSREG

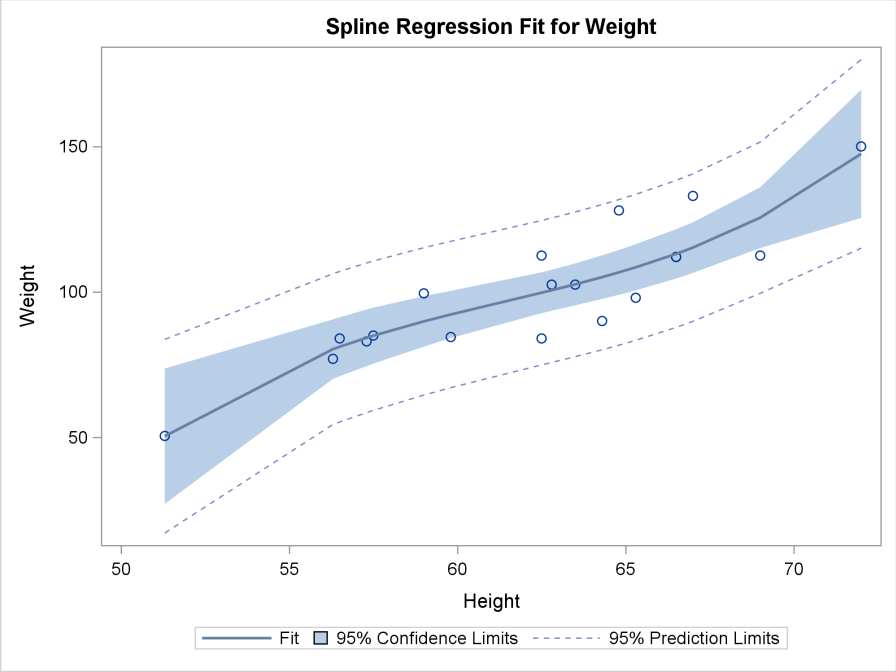
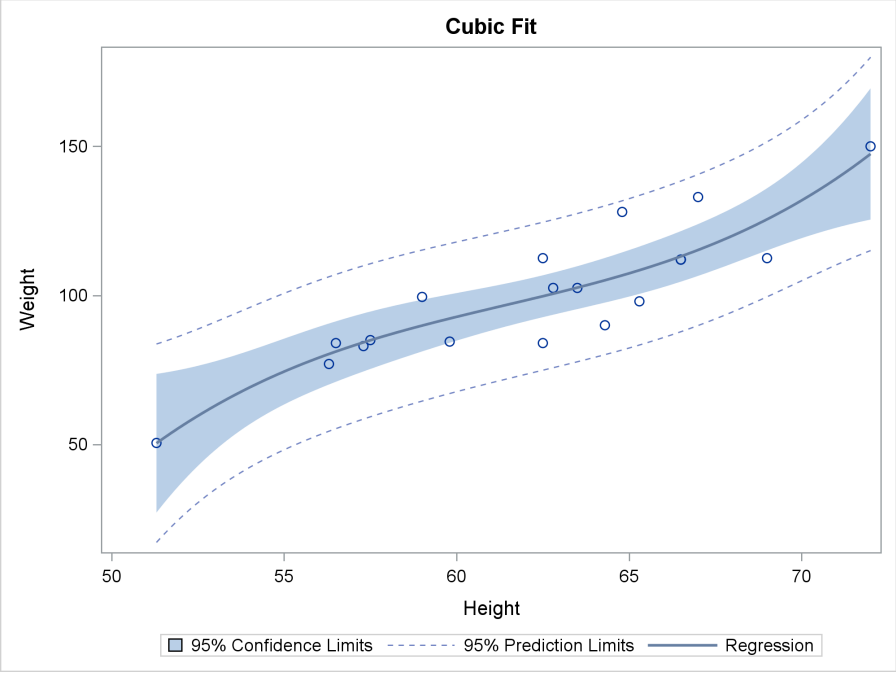


Figure 1.20 Cubic Fit Plot from PROC SGPLOT



The two plots are similar but not identical. The titles are different, the legends are ordered differently, and the fit function legend entries are labeled differently. In addition, PROC SGPLOT automatically creates smoother functions by computing predicted values, confidence limits, and prediction limits for intermediate interpolated values. PROC TRANSREG, by default, computes these values for each

observed X value and does not add intermediate points. This difference is usually noticeable only for small data sets such as the one used in this example.

The following steps use the output data set from PROC TRANSREG and produce the same plots by manually specifying the scatter plot, fit function, confidence limits, and prediction limits, each with a separate statement:

```
proc template;
   define statgraph classfit;
      begingraph;
         entrytitle 'Cubic Fit';
         layout overlay;
            bandplot limitupper=cmuweight limitlower=cmlweight x=height /
                  connectorder=axis outlineattrs=GraphConfidence
                  name='95% Confidence Limits';
            bandplot limitupper=ciuweight limitlower=cilweight x=height /
                  connectorder=axis display=(outline)
                  outlineattrs=GraphPredictionLimits
                  name='95% Prediction Limits';
            scatterplot y=weight  x=height;
            seriesplot  y=pweight x=height / name='Fit' legendlabel='Fit'
                        connectorder=xaxis lineattrs=GraphFit;
            discretelegend 'Fit' '95% Confidence Limits'
                           '95% Prediction Limits';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=class template=classfit;
run;

proc sort data=class;
   by height;
run;

proc sgplot data=class;
   title 'Cubic Fit';
   band upper=cmuweight lower=cmlweight x=height /
        fillattrs=GraphConfidence
        name='b1' legendlabel='95% Confidence Limits';
   band upper=ciuweight lower=cilweight x=height /
        nofill lineattrs=GraphPredictionLimits
        name='b2' legendlabel='95% Prediction Limits';
   scatter y=weight  x=height;
   series  y=pweight x=height / legendlabel='Fit' lineattrs=GraphFit name='f';
   keylegend 'f' 'b1' 'b2';
run;
```

The following GTL statement produces the confidence limits:

```
bandplot limitupper=cmuweight limitlower=cmlweight x=height /
      connectorder=axis outlineattrs=GraphConfidence
      name='95% Confidence Limits';
```

A filled band is displayed across the range of the X-axis variable Height between the lower-limit variable cmlweight and the upper-limit variable cmuweight. The CONNECTORDER=AXIS option connects the data points from left to right along the X axis. The OUTLINEATTRS= option specifies the **GraphConfidence** style element for the outline. The NAME= option provides a statement name so that the confidence limit information can be added to the legend.

The following GTL statement produces the prediction limits:

```
bandplot limitupper=ciuweight limitlower=cilweight x=height /
      connectorder=axis display=(outline)
      outlineattrs=GraphPredictionLimits
      name='95% Prediction Limits';
```

The limits are displayed across the range of the X-axis variable Height between the lower-limit variable cilweight and the upper-limit variable ciuweight. The CONNECTORDER=AXIS option connects the data points from left to right along the X axis. The DISPLAY= option displays only the outline of the band rather than the entire band. The OUTLINEATTRS= option specifies the **GraphPredictionLimits** style element for the outline. The NAME= option provides a statement name so that the prediction limit information can be added to the legend.

The following GTL statement produces the scatter plot:

```
scatterplot y=weight x=height;
```

It plots the variable Weight on the Y axis and the variable Height on the X axis.

The following GTL statement produces the series plot that displays the fit function:

```
seriesplot y=pweight x=height / name='Fit' legendlabel='Fit'
           connectorder=xaxis lineattrs=GraphFit;
```

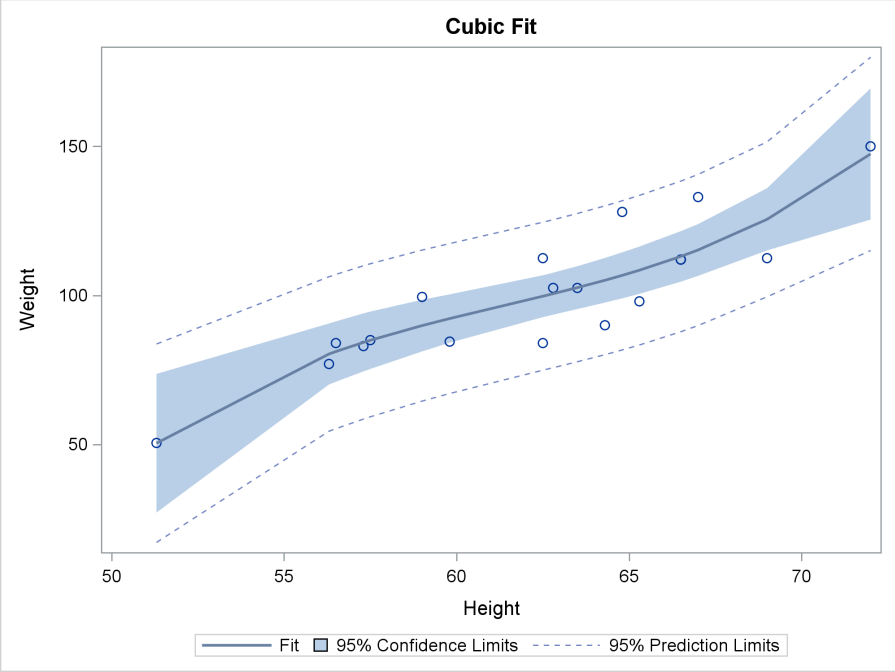
The predicted values were previously computed by PROC TRANSREG, and the SERIESPLOT statement connects them to provide the fit function. The label for the legend is `'Fit'`. The LINEATTRS= specifies the **GraphFit** style for the fit line. As in the BANDPLOT statements, the CONNECTORDER=AXIS and NAME= options are specified.

The following GTL statement produces the legend from the three named statements:

```
discretelegend 'Fit' '95% Confidence Limits' '95% Prediction Limits';
```

The results are displayed in Figure 1.21.

Figure 1.21 Cubic Fit Plot Constructed by Using the GTL



PROC SGPLOT statements do not have a CONNECTORDER=AXIS option, so the data are sorted by the X=Height variable before the graph is made. The first plotting statement in PROC SGPLOT is as follows:

```
band upper=cmuweight lower=cmlweight x=height / fillattrs=GraphConfidence
     name='b1' legendlabel='95% Confidence Limits';
```

A filled band is displayed across the range of the X-axis variable Height between the lower-limit variable cmlweight and the upper-limit variable cmuweight. The FILLATTRS= option specifies the **GraphConfidence** style element for the fill (the band). The NAME= option provides a statement name so that the confidence limit information can be added to the legend. The LEGENDLABEL= option provides the text that is displayed in the legend along with the confidence limits information.

The following PROC SGPLOT statement produces the prediction limits:

```
band upper=ciuweight lower=cilweight x=height /
     nofill lineattrs=GraphPredictionLimits
     name='b2' legendlabel='95% Prediction Limits';
```

The limits are displayed across the range of the X-axis variable Height between the lower-limit variable cilweight and the upper-limit variable ciuweight. The NOFILL option produces an outline instead of a band. The LINEATTRS= option specifies the **GraphPredictionLimits** style element for the outline. The NAME= option provides a statement name so that the prediction limit information can be added to the legend. The LEGENDLABEL= option provides the text that is displayed in the legend along with the prediction limits information.

The following PROC SGPLOT statement produces the scatter plot:

```
scatter y=weight x=height;
```

It plots the variable Weight on the Y axis and the variable Height on the X axis.

The following PROC SGPLOT statement produces the series plot that displays the fit function:

```
series y=pweight x=height / legendlabel='Fit' lineattrs=GraphFit name='f';
```

The predicted values were previously computed by PROC TRANSREG, and the SERIES statement connects them to provide the fit function. The label for the legend is `'Fit'`. The LINEATTRS= specifies the **`GraphFit`** style for the fit line. As in the BAND statements, the NAME= option is specified.

The following PROC SGPLOT statement produces the legend from the three named statements:

```
keylegend 'f' 'b1' 'b2';
```

The results are identical to the graph displayed in Figure 1.21.

Both the REG statement in PROC SGPLOT and the REGPLOT statement in the GTL accept the following options and many others:

- ALPHA=0.05, the default, produces 95% confidence and prediction limits. Specify ALPHA=0.01 for 99% confidence and prediction limits.
- CURVELABEL=*'string'* specifies a label for the regression function.

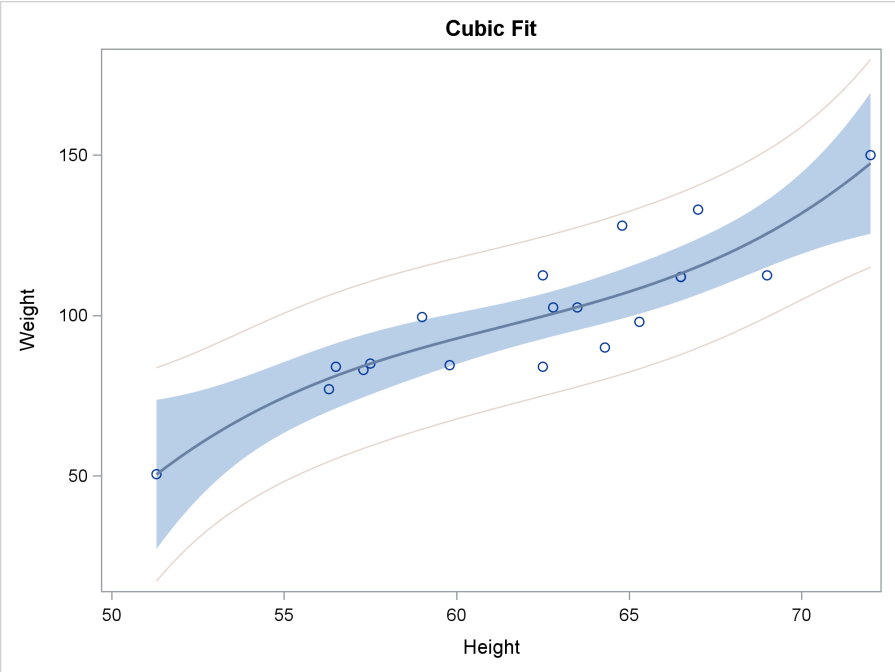
The previous steps used the GTL, PROC TEMPLATE, and PROC SGRENDER to display regression results that were computed by another procedure (PROC TRANSREG). The next series of steps does all the computations within the GTL. They illustrate the MODELBAND statement and also the technique of using and overriding style elements to control the appearance of the confidence and prediction limits.

The following steps create the graph in Figure 1.22:

```
proc template;
   define statgraph classfit1;
      begingraph;
         entrytitle 'Cubic Fit';
         layout overlay;
            modelband "cliband" / display=(outline);
            modelband "clmband";
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3
                                               clm='clmband' cli='cliband';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classfit1;
run;
```

The CLM='clmband' option in the REGRESSIONPLOT statement displays the confidence limits according to the options specified in the MODELBAND statement named 'clmband'. This MODELBAND statement has no options, so a default band plot, which consists of a filled band with no outline, is produced. Similarly, the CLI='cliband' option displays the prediction limits according to the options specified in the MODELBAND statement named 'cliband'. This MODELBAND statement specifies that an outline of the band is to be displayed, but the band is not filled. Statements are executed in the order in which they are specified in the template. The prediction limits are drawn first, then the confidence limits, then the scatter plot, then the fit function. Plot elements that are placed later can obscure previously placed plot elements. Hence, you do not want to draw the band plots last, because you will not be able to see the fit function and many of the points.

Figure 1.22 Prediction Limits Outlined



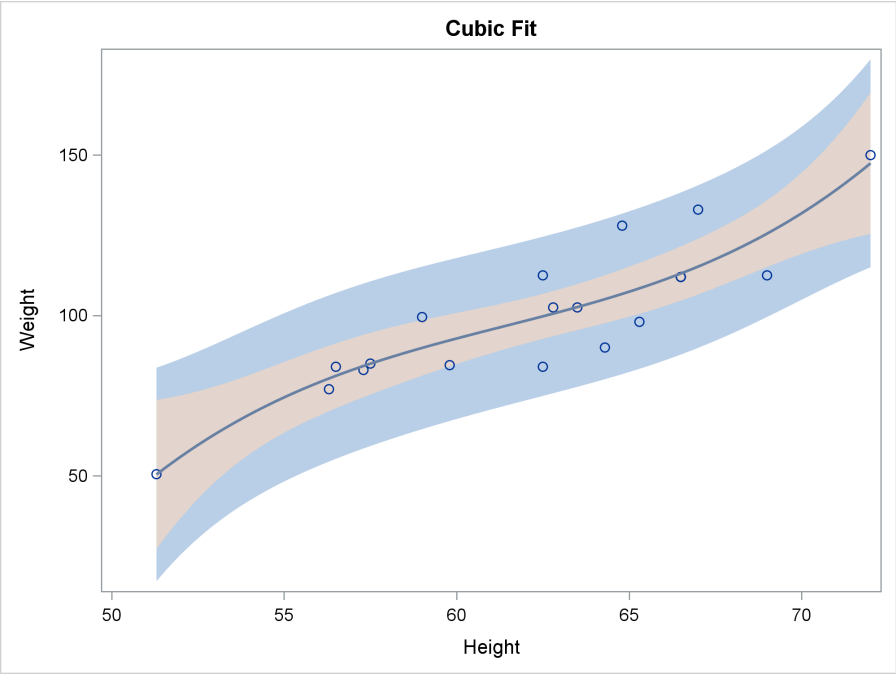
The following steps create the graph in Figure 1.23:

```
proc template;
   define statgraph classfit2;
      begingraph;
         entrytitle 'Cubic Fit';
         layout overlay;
            modelband "cliband" / fillattrs=GraphConfidence;
            modelband "clmband" / fillattrs=GraphConfidence2;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3
                           clm="clmband" cli="cliband";
         endlayout;
      endgraph;
   end;
run;
```

```
proc sgrender data=sashelp.class template=classfit2;
run;
```

In this example, the **FillAttrs=GraphConfidence** option uses the **GraphConfidence** style element to control the appearance of the prediction limits, and the **FillAttrs=GraphConfidence2** option uses the **GraphConfidence2** style element to control the appearance of the confidence limits.

Figure 1.23 Confidence Style Elements

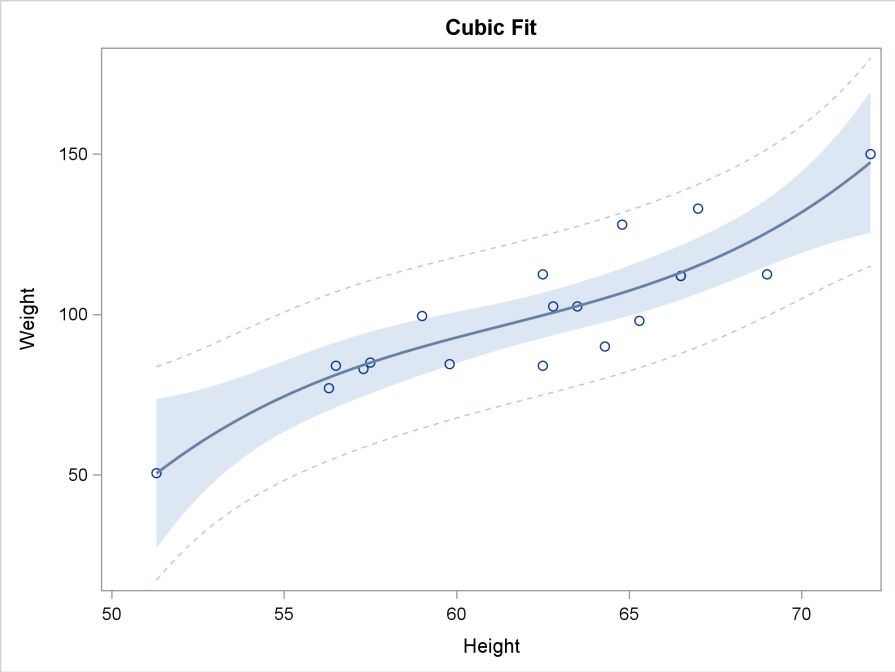


The following steps create the graph in Figure 1.24:

```
proc template;
   define statgraph classfit3;
      begingraph;
         entrytitle 'Cubic Fit';
         layout overlay;
            modelband "cliband" / outlineattrs=GraphPredictionLimits
                                  display=(outline)
                                  datatransparency=0.5;
            modelband "clmband" / fillattrs=GraphConfidence
                                  datatransparency=0.5;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3
                           clm="clmband" cli="cliband";
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classfit3;
run;
```

The graph in Figure 1.24 has an appearance similar to that of the graph produced directly by PROC TRANSREG (shown in Figure 1.19) and to the graph produced directly by PROC SGPLOT (shown in Figure 1.20). The prediction limits are displayed as an outline without fill because of the DISPLAY=(OUTLINE) option. The style of the line is controlled by the **`OutLineAttrs=GraphPredictionLimits`** option, which uses the **`GraphPredictionLimits`** style element, which specifies a dashed line in this style. The DATATRANSPARENCY=0.5 option is specified so that the bands and outlines are 50% as transparent as the default. The confidence limits are displayed as a band without an outline because of the default setting of the DISPLAY= option, DISPLAY=(FILL). The style of the fill is controlled by the **`FillAttrs=GraphConfidence`** option, which uses the **`GraphConfidence`** style element.

Figure 1.24 Outline, Fill, and Transparency

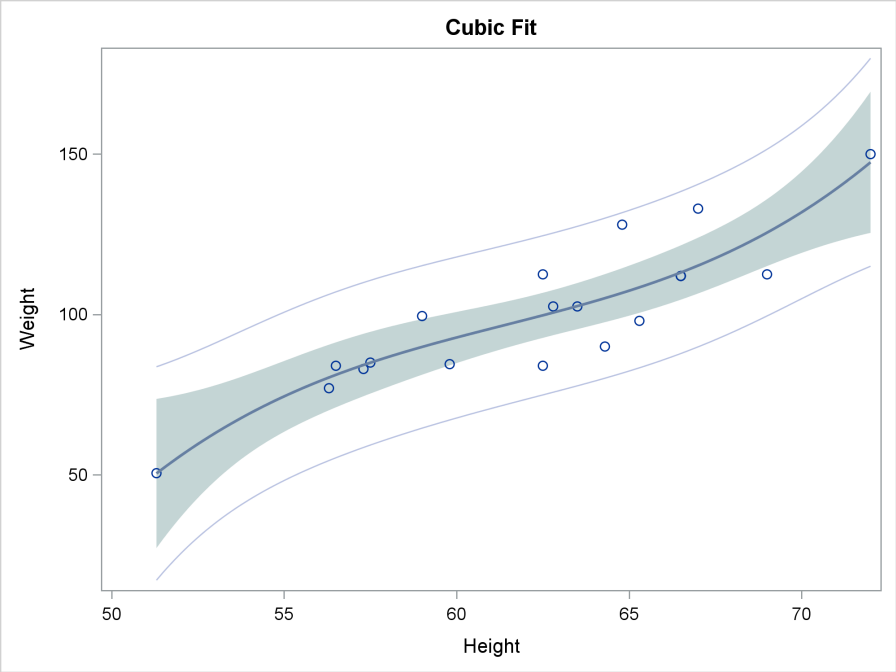


The following steps create the graph in Figure 1.25:

```
proc template;
   define statgraph classfit4;
      begingraph;
         entrytitle 'Cubic Fit';
         layout overlay;
            modelband "cliband" /
                      outlineattrs=GraphPredictionLimits(pattern=solid)
                      display=(outline)
                      datatransparency=0.5;
            modelband "clmband" /
                      fillattrs=GraphConfidence(color=cx88AAAA)
                      datatransparency=0.5;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3
                           clm="clmband" cli="cliband";
```

```
        endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classfit4;
run;
```

Figure 1.25 Style Element Overrides



These steps are similar to the preceding steps. However, one part of the **GraphPredictionLimits** style element is overridden—namely, the line pattern. The option **OutLineAttrs=GraphPredictionLimits(Pattern=Solid)** creates prediction limits by using all style elements from the **GraphPredictionLimits** element except for the default line style. Similarly, the **FillAttrs=GraphConfidence(Color=cx88AAAA)** option overrides the color of the **GraphConfidence** style element. All colors can be specified in values of the form CX*rrggbb*, where the last six characters specify RGB (red, green, blue) values on the hexadecimal scale of 00 to FF (0 to 255 base 10). HLS (hue/light/saturation) color specifications and all other color specifications available in SAS/GRAPH are also available in ODS Graphics.

1.2.5 Text Insets and Special Characters

This section shows how to insert text into a graph, including text with superscripts (such as R^2) and Greek letters with special characters (such as $\hat{\mu}$). The graph is the cubic fit function from the section "Regression Fit Plot" on page 6. The following step runs PROC TRANSREG with a cubic polynomial model and creates an ODS output data set with regression fit statistics, including R-square and the mean of the dependent variable:

```
proc transreg data=sashelp.class ss2;
   ods output fitstatistics=fs;
   model identity(weight) = spline(height);
run;
```

The following step stores the R-square in a macro variable, R2, and the mean in a macro variable, Mean:

```
data _null_;
   set fs;
   if _n_ = 1 then call symputx('R2',   put(value2, 4.2   ), 'G');
   if _n_ = 2 then call symputx('mean', put(value1, best6.), 'G');
run;
```

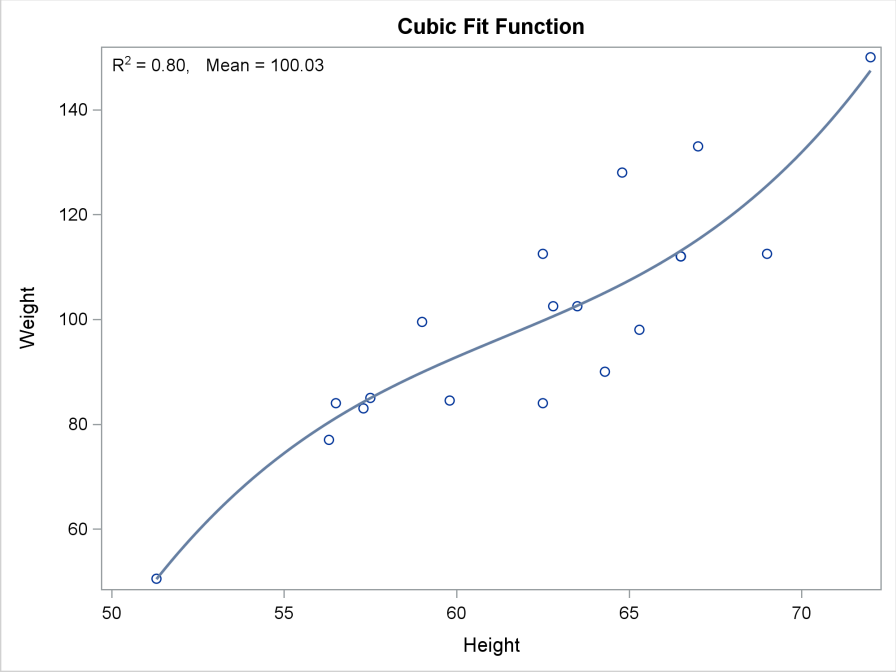
The R-square value is stored with a format of 4.2, and the mean is stored with the BEST6. format. Both are stored in the global macro symbol table. These values are inserted into plots as inset text entries in the next steps.

The following steps create a plot with the R-square and the mean in the top left corner:

```
proc template;
   define statgraph classreg1;
      mvar r2 mean;
      begingraph;
         entrytitle 'Cubic Fit Function';
         layout overlay;
            entry halign=left 'R' {sup '2'} ' = ' r2
                  ",   Mean = " mean / valign=top;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classreg1;
run;
```

The MVAR statement specifies the two macro variables, R2 and Mean, so that they can be used in other parts of the template. The ENTRY statement in this example adds to the top left of the plot a single line of text that consists of an "R^2", an equal sign, the formatted R-square value, a comma and spaces, "Mean =", and the mean value. The inset text entry is created from a series of values in the ENTRY statement, including constant text, special characters, and values that are retrieved from macro variables. The syntax `'R' {sup '2'}` is the syntax for superscripts. (Similarly, the syntax for subscripts is `'x' {sub 'i'}`.) Notice that the macro variables are specified without ampersands. Therefore, their values are not retrieved before PROC TEMPLATE compiles the template; rather, the values are retrieved when PROC SGRENDER is run. This approach enables you to create the template once and use it repeatedly for different analyses that have different values for the two statistics. Also notice that the specification `{sup '2'}` does not appear in quotation marks. If it had appeared in quotes, then "{sup '2'}" would appear in the graph and not "^{2}". The results are displayed in Figure 1.26.

Figure 1.26 Inset Text with the GTL



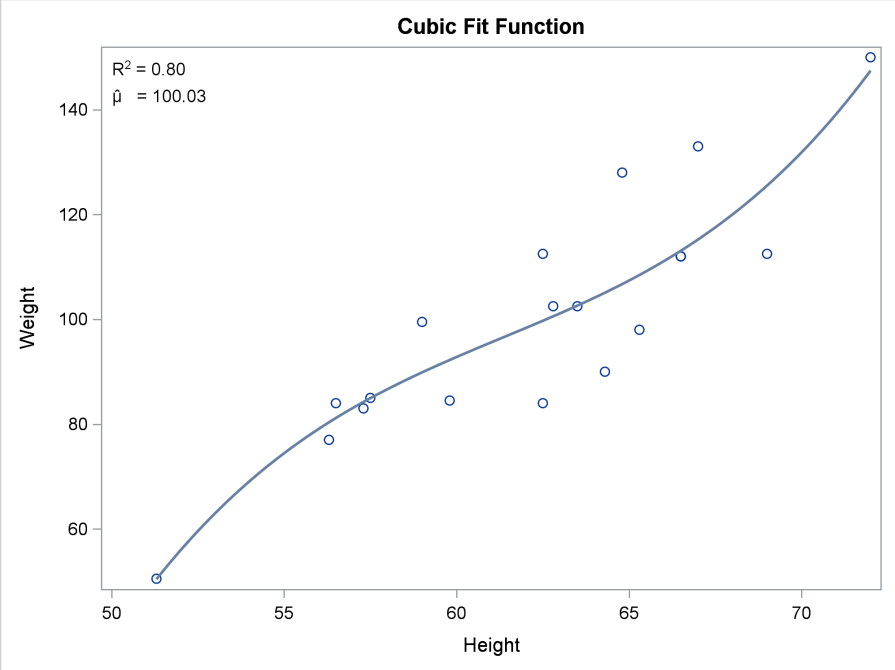
The following steps insert the same information into the plot, but with two differences:

```
proc template;
   define statgraph classreg2;
      mvar r2 mean;
      begingraph;
         entrytitle 'Cubic Fit Function';
         layout overlay;
            layout gridded / autoalign=(topright topleft
                                        bottomright bottomleft);
               entry halign=left 'R' {sup '2'} ' = ' r2;
               entry halign=left "(*ESC*){unicode mu}(*ESC*){unicode hat}    = "
                     mean / textattrs=GraphValueText(
                                    family=GraphUnicodeText:FontFamily);
               endlayout;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classreg2;
run;
```

The text is split onto two lines, and the mean is labeled by "$\hat{\mu}$" instead of "Mean." The results of this step are displayed in Figure 1.27. The LAYOUT GRIDDED statement groups the two ENTRY statements, enabling them to be moved and positioned as a group with one immediately following the other. You can place any number of ENTRY statements between the LAYOUT and ENDLAYOUT statements. The LAYOUT GRIDDED statement has several options. For example, you can specify

the options OPAQUE=TRUE, BORDER=TRUE, and BACKGROUNDCOLOR=*style-reference* | *color* to create an opaque box with a background and color. In this example, the AUTOALIGN= option specifies a list of preferred positions. ODS Graphics places the entries in the first available position. In this example, there are data in the top right, so the entries are placed in the top left.

Figure 1.27 Special Characters with the GTL



Both ENTRY statements specify the HALIGN=LEFT option to left-align both entries. The second ENTRY statement specifies "$\hat{\mu}$" with the specification **`(*ESC*){unicode mu}(*ESC*){unicode hat}`**, which appears in quotation marks. Special characters are specified as Unicode characters with an escape sequence. The **`(*ESC*)`** specification is a flag that indicates that a special character follows. Guidelines for when you use the **`(*ESC*)`** specification are discussed later in this section. For now, simply note that the **`{sup '2'}`** specification is not in quotes and is not escaped, whereas the Unicode specifications are in quotes and are escaped. The TEXTATTRS= option is specified so that a font that has the Unicode characters is used. Notice that extra white space was added to better align the parts of each line.

Greek letters are specified by **`{unicode`** *letter***`}`** or **`{unicode`** *letter*`_U}`**. Lowercase Greek letters are specified by name (for example, **`Alpha`**, **`Beta`**, **`Gamma`**). Uppercase Greek letters are specified by appending an **`_U`** to the name (for example, **`Alpha_U`**, **`Beta_U`**, **`Gamma_U`**). Additional keywords include **`Bar`**, **`Bar2`**, **`Hat`**, **`Tilde`**, and **`Prime`**. These keywords are part of the Unicode range that is called combining (nonspacing) diacritical marks, which means that they are drawn superimposed on the previous character rather than in the next position. These marks should always immediately follow the character they are part of. Hence, **`{Unicode Mu}{Unicode Hat}`** (which is correct) is different from **`{Unicode Hat}{Unicode Mu}`** (which is backward and incorrect) and also different from **`{Unicode Mu} {Unicode Hat}`** (which includes a space and so is also incorrect). The Unicode Consortium (`http://unicode.org/`) provides a page of character codes at `http://www.unicode.org/charts/charindex.html`.

The following step uses PROC SGPLOT and produces a plot that is almost identical to the plot displayed in Figure 1.27 (which was created in the previous steps by PROC TEMPLATE and PROC SGRENDER):[1]

```
proc sgplot data=sashelp.class noautolegend;
   title 'Cubic Fit Function';
   inset "R(*ESC*){sup '2'} =  &r2"
         "(*ESC*){unicode mu}(*ESC*){unicode hat}   = &mean" /
         position=topleft;
   reg y=weight x=height / degree=3;
run;
```

The INSET statement in this example produces two entries because two quoted strings are specified. In the INSET statement (unlike the ENTRY statement in the GTL), each string forms a new line, and each string must be fully quoted, even when special characters are provided. Each Unicode, superscript, and subscript specification must be escaped because each must appear inside a quoted string. The escape sequence tells ODS Graphics that a special character is coming. Otherwise, each of these specifications appears "as is" in the graph. Because PROC SGPLOT does not have an MVAR statement as the GTL does, the macro variables appear inside the double quotes and appear with an ampersand. Their values are resolved when PROC SGPLOT is run. Note that the specification `"R(*ESC*){sup '2'} = &r2"` contains double outer quotes and single inner quotes. Double quotes are required for the outer quotes so that the macro variables will resolve. Double quotes could also be used for the inner quotes, but they have to be doubled so that they do not end the string (for example, `"R(*ESC*){sup ""2""} = &r2"`). No TEXTATTRS= option is needed in PROC SGPLOT, which automatically switches to the appropriate font when it encounters the Unicode characters.

The final steps in this example use PROC TEMPLATE and PROC SGRENDER and produce the plot without any escape characters. The following steps make a graph that is identical to the one in Figure 1.27:

```
proc template;
   define statgraph classreg3;
      mvar r2 mean;
      begingraph;
         entrytitle 'Cubic Fit Function';
         layout overlay;
            layout gridded / autoalign=(topright topleft
                                        bottomright bottomleft);
               entry halign=left 'R' {sup '2'} ' = ' r2;
               entry halign=left {unicode mu}{unicode hat} '   = ' mean /
                     textattrs=GraphValueText(family=GraphUnicodeText:FontFamily);
               endlayout;
            scatterplot y=weight x=height;
            regressionplot y=weight x=height / degree=3;
         endlayout;
      endgraph;
   end;
run;
```

[1] Many graphs in this book are said to be "almost identical" to some other graph. In most cases, a small amount of white space shifts parts of one of the graphs. Some differences are as small as a pixel.

```
proc sgrender data=sashelp.class template=classreg3;
run;
```

1.3 Heat Maps

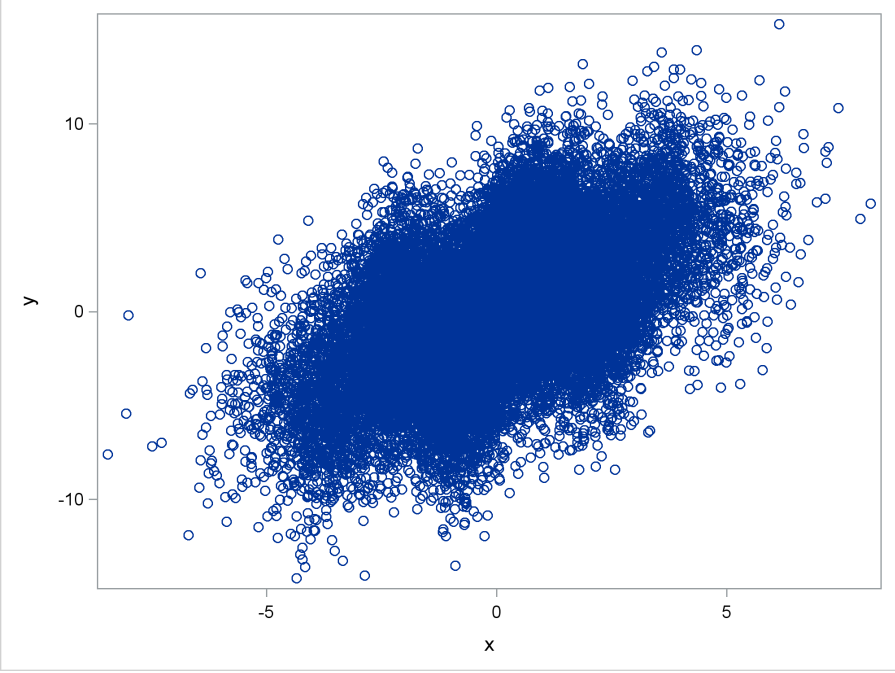
Double-Click for Example Code

Scatter plots display a marker at the intersection of the values of an X and a Y variable. In contrast, heat maps divide the graph into rectangular (or hexagonal) bins and use colors to show how many observations fall in each bin. When there are a large number of data points, ordinary scatter plots, fit plots, residual plots, and so on become hard to interpret. With enough data, points merge into large blobs that do not always reveal the underlying structure of the data. Heat maps better differentiate between the denser and less dense portions of the scatter plot. The following steps create a scatter plot of 25,000 artificial data points:

```
title;
data x(drop=i);
   do i = 1 to 25000;
      x = 2 * normal(104);
      y = x + sin(x * 2) + 3 * normal(104);
      output;
   end;
run;

proc sgplot data=x;
   scatter y=y x=x;
run;
```

The results are displayed in Figure 1.28.

Figure 1.28 Ordinary Scatter Plot

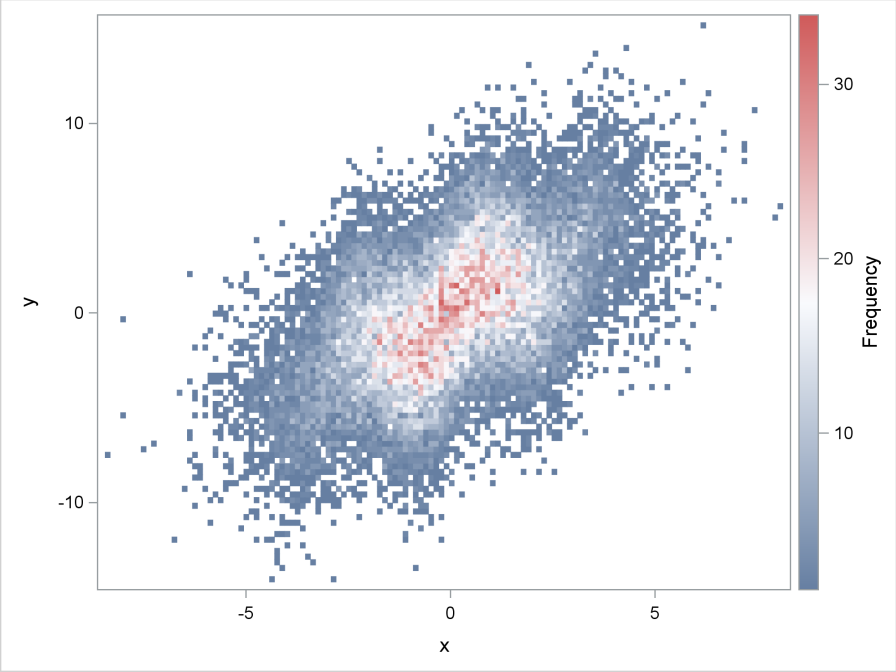


The following step creates a heat map of the same data:

```
proc sgplot data=x;
   heatmap y=y x=x;
run;
```

The results are displayed in Figure 1.29.

Figure 1.29 Default Heat Map



The underlying function, which has a linear component and a sine wave component, is apparent in Figure 1.29 but not in Figure 1.28. The colors in the heat map progress from blue to light gray to red. You can reorder the colors to progress from light gray (similar to the white background) to blue to red for an even clearer effect. You are not required to specify the same colors that are used in the HTMLBlue style; however, you can find them by listing the style source code:

```
proc template;
   source styles.htmlblue / expand;
quit;
```

Style listings tend to be large, so the results of this step are not displayed. Instead, the following steps write the same information to a file and display just the three-color ramp colors for the HTMLBlue style and its parents:

```
proc template;
   source styles.htmlblue / file='junk.tpl' expand;
quit;

data junk;
   retain Style '           ';
   infile 'junk.tpl' lrecl=80 pad;
   input Color $ 1-80;
   if index(Color, 'define style') then style = scan(Color, 4, ' .;');
   if index(Color, 'gramp3') and not index(Color, 'GraphColors');
run;

proc print noobs;
run;
```

The results are displayed in Figure 1.30. The listing displays the style names and the color names for the HTMLBLue style, its parent Statistical, and its parent Default.

Figure 1.30 Three-Color Ramp Components

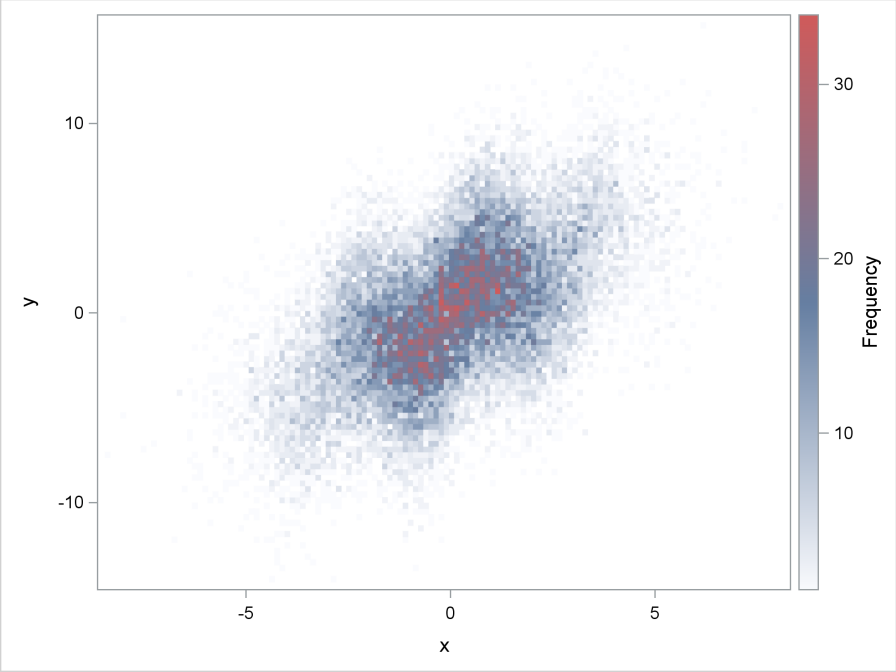
Style	Color
Htmlblue	'gramp3cend' = cxD05B5B
Htmlblue	'gramp3cneutral' = cxFAFBFE
Htmlblue	'gramp3cstart' = cx667FA2
statistical	'gramp3cend' = cx667FA2
statistical	'gramp3cneutral' = cxFFFFFF
statistical	'gramp3cstart' = cxAFB5A6
default	'gramp3cend' = cxDD6060
default	'gramp3cneutral' = cxFFFFFF
default	'gramp3cstart' = cx6497EB

You can use the COLORMODEL= option to specify the three colors, starting with the neutral color (gray), continuing with the starting color (blue), and followed by the ending color (red):

```
proc sgplot data=x;
   heatmap y=y x=x / colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
run;
```

The results are displayed in Figure 1.31. The pattern is even clearer with this color scheme.

Figure 1.31 Alternative Ordering of the Color Ramp

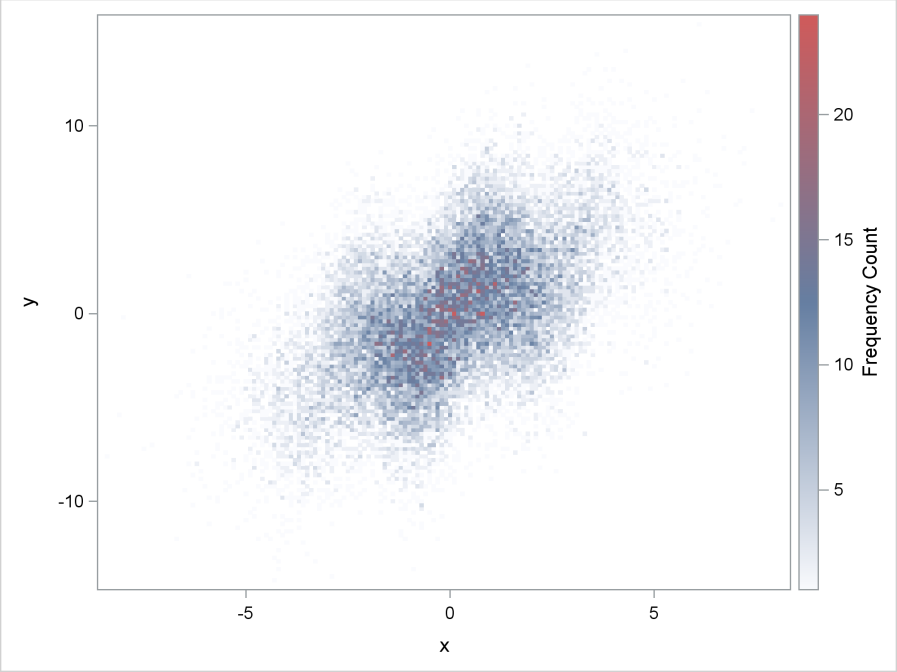


You can create a heat map and control the binning yourself by using the HEATMAPPARM statement. The following steps illustrate:

```
data grid;
   set x;
   x = round(x, 0.1);
   y = round(y, 0.2);
run;

proc freq noprint;
   tables x * y / out=grid2;
run;

proc sgplot data=grid2;
   heatmapparm y=y x=x colorresponse=count /
                       colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
run;
```

The DATA step rounds similar values to a common value, creating a grid. The PROC FREQ step counts the number of values in each cell of the grid. The resulting variable Count is specified in the COLORRESPONSE= option in the HEATMAPPARM statement. The results are displayed in Figure 1.32.

Figure 1.32 Controlling the Heat Map Grid

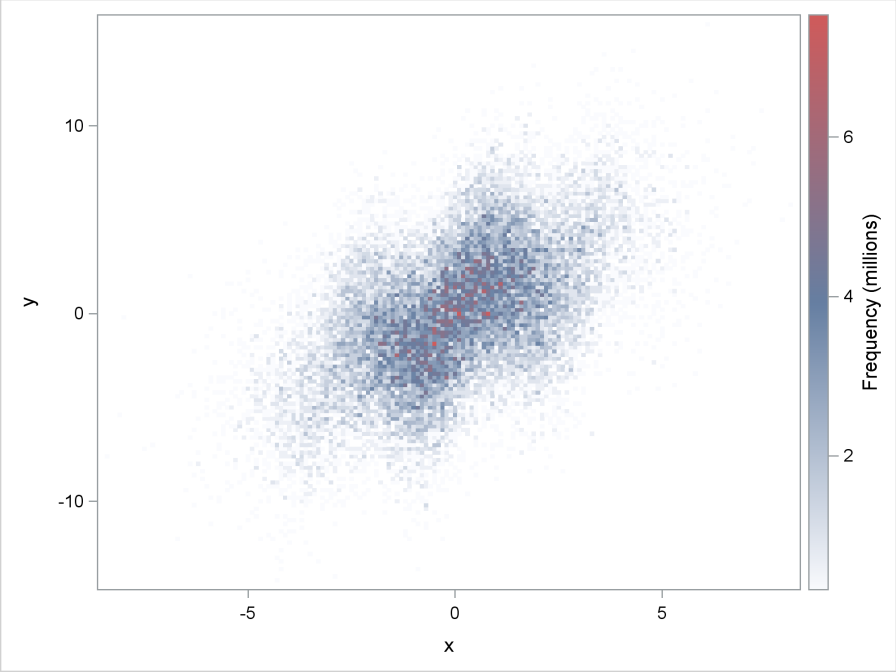


You can use the GRADLEGEND statement to control the color gradient legend:

```
data grid3;
   set grid2;
   count = count * constant('pi') * 1e5;
run;

proc sgplot data=grid3;
   heatmapparm y=y x=x colorresponse=count / name='a'
                       colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
   gradlegend 'a' / integer title = 'Frequency' extractscale;
run;
```

The DATA step artificially creates larger frequencies. The GRADLEGEND statement controls the legend for the statement named `'a'`. Values in the legend are required to be integers, the label is set to “Frequency”, and a common scale is extracted (millions) and added to the label. The results are displayed in Figure 1.33.

Figure 1.33 Controlling the Heat Map Legend



You can use the GTL to create a heat map as follows:

```
proc template;
   define statgraph heatmap;
      begingraph;
         layout overlay;
            heatmap x=x y=y / xbinaxis=false ybinaxis=false name="heatmap"
                              colormodel=(cxfafbfe cx667fa2 cxd05b5b);
            continuouslegend "heatmap" / title="Frequency" integer=true;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=x template=heatmap;
run;
```

The XBINAXIS=FALSE and YBINAXIS=FALSE options specify that bins not provide the bases for the X-axis and Y-axis ticks. The results are identical to the graph displayed in Figure 1.31.

You can use the GTL to create a parameterized heat map as follows:

```
proc template;
   define statgraph heatmap;
      begingraph;
         layout overlay;
            heatmapparm x=x y=y colorresponse=count /
                               xbinaxis=false ybinaxis=false name="heatmap"
                               colormodel=(cxfafbfe cx667fa2 cxd05b5b);
            continuouslegend "heatmap" / integer=true title="Frequency Count";
```

```
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=grid2 template=heatmap;
run;
```

The results are identical to the graph displayed in Figure 1.32.

The following steps show SAS/STAT procedures that create heat maps:

```
ods graphics on;

proc reg data=x;
   model y = x;
quit;

proc surveyreg data=x;
   model y = x;
run;

proc surveyreg data=x plots=fit(shape=hexagonal);
   model y = x;
run;
```

The results are displayed in Figure 1.34 through Figure 1.37. PROC SURVEYREG has an option for hexagonal binning. Three bin shapes are possible: triangle, rectangle, and hexagon. Rectangles are commonly used. Hexagons have the advantage of having more sides than squares (hexagons are closer geometrically to circles than squares are), which means that hexagons have smaller distances to the bin centers than squares do. Other procedures do not have an option for hexagonal binning.

Figure 1.34 PROC REG Fit Plot

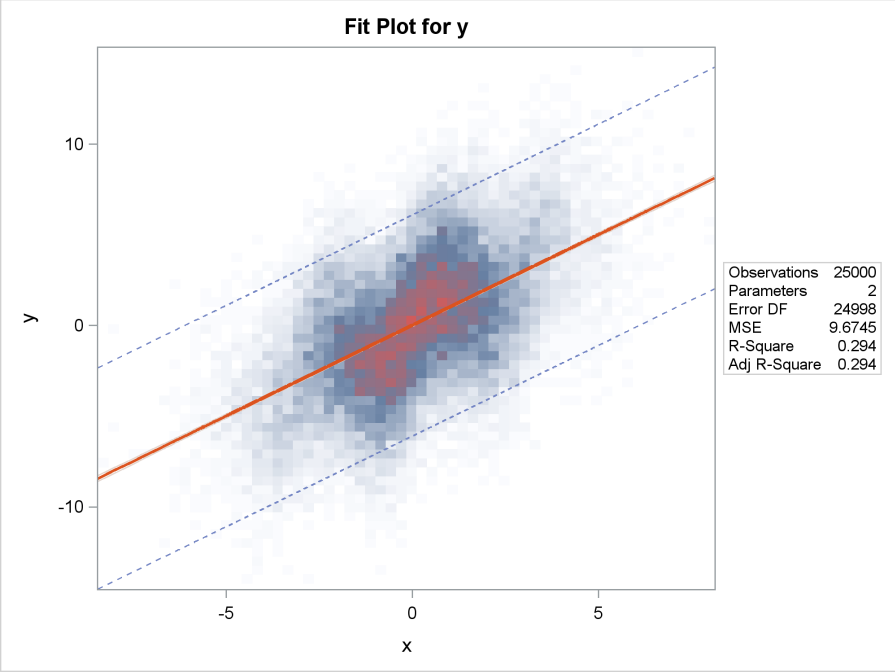


Figure 1.35 PROC REG Residual Plot

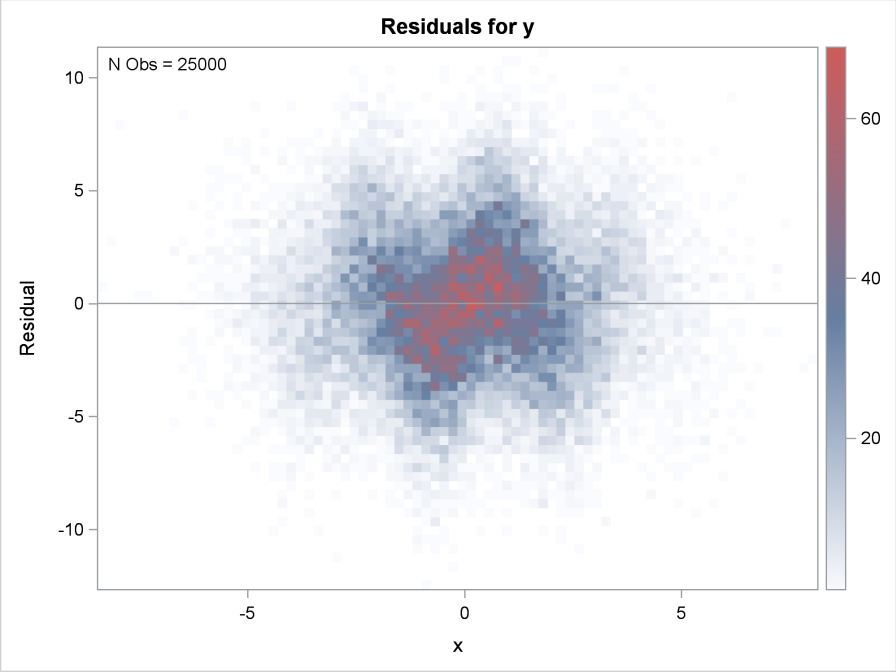


Figure 1.36 PROC SURVEYREG Fit Plot

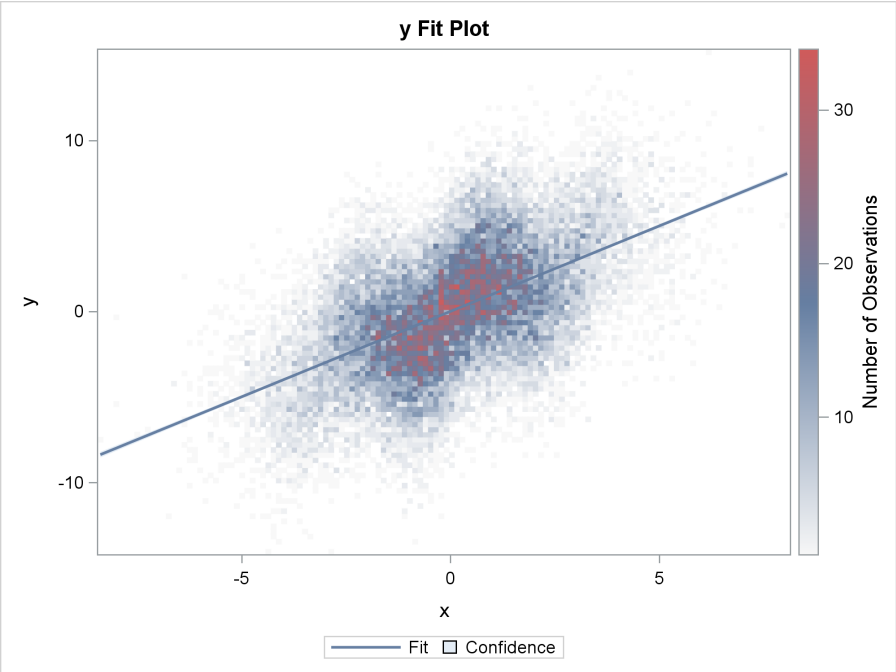
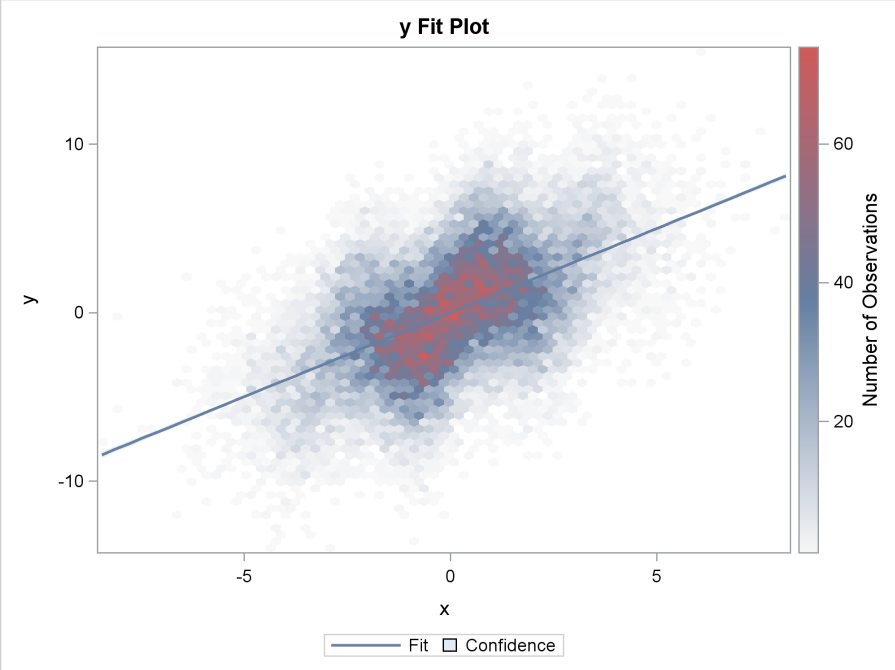


Figure 1.37 PROC SURVEYREG Fit Plot with Hexagonal Binning

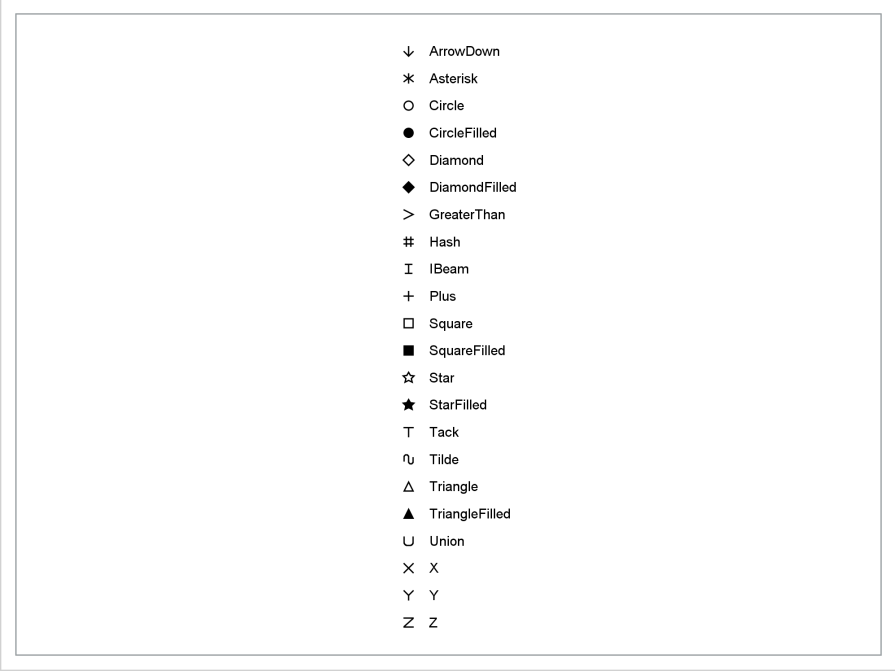


1.4 Markers

Double-Click for Example Code

ODS Graphics provides the markers displayed in Figure 1.38.

Figure 1.38 Markers



You can use the SYMBOLCHAR statement to add thousands of characters to this list. The first part of this example shows how to use multiple SYMBOLCHAR statements when you have multiple plotting statements. Using multiple SYMBOLCHAR statements requires some data manipulations that you usually would not need to do. A subsequent step shows how to use SYMBOLCHAR statements and an attribute map along with the more typical data set shape.

The first step processes the Sashelp.Class data set so that there are separate height and weight variable for males and females rather than the customary single pair of variables:

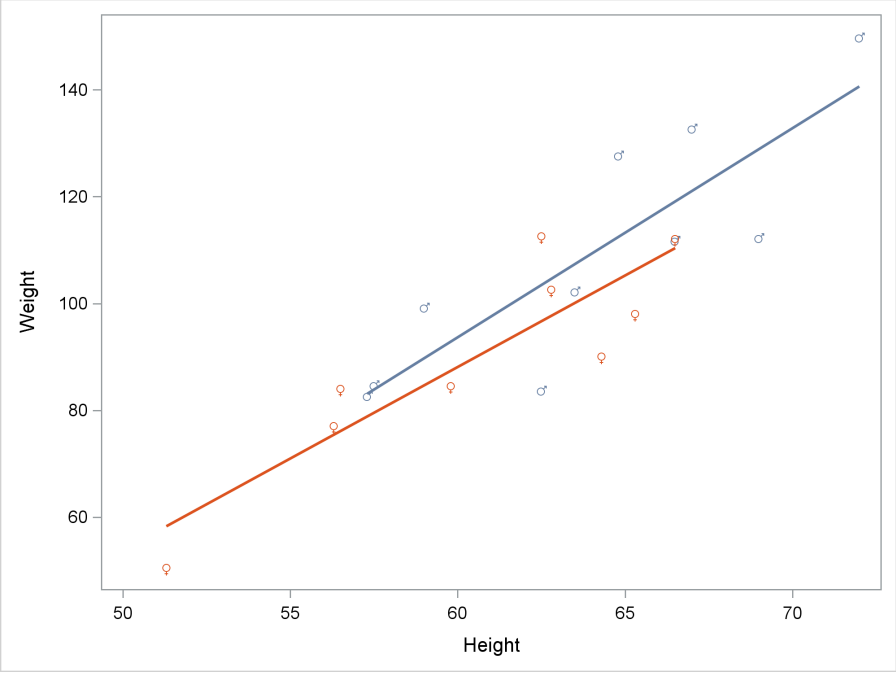
```
data class(drop=name sex age);
   merge sashelp.class(where=(sex='F') rename=(weight=fweight height=fheight))
         sashelp.class(where=(sex='M') rename=(weight=mweight height=mheight));
   label fheight = 'Height' fweight = 'Weight';
run;
```

The next step uses the SYMBOLCHAR statement to name two new markers: `'m'` (which displays the Unicode symbol for male) and `'f'` (which displays the Unicode symbol for female).

```
ods graphics on / attrpriority=none subpixel=on;
title;

proc sgplot noautolegend;
   symbolchar name=m char='2642'x;
   symbolchar name=f char='2640'x;
   reg y=mweight x=mheight / markerattrs=(symbol=m size=15);
   reg y=fweight x=fheight / markerattrs=(symbol=f size=15);
run;
```

The results are displayed in Figure 1.39.

Figure 1.39 Controlling Markers



The data are set up so that the results could be displayed using two REG statements. The first uses the option SYMBOL=M, which uses the character specified in the SYMBOLCHAR NAME=M statement. The second uses the option SYMBOL=F, which uses the character specified in the SYMBOLCHAR NAME=F statement. This example uses the ATTRPRIORITY=NONE option, which is explained in detail in the section "Grouped Regression Fit Plot" on page 16. Without this option, groups would be distinguished by color changes but not marker changes.

When the data are grouped, as the Sashelp.Class data set is, you can use SYMBOLCHAR statements along with an attribute map to display the results:

```
data attrmap;
   retain ID 'Stmt1' LinePattern 1;
   input Value $ MarkerSymbol $ LineColor $;
   MarkerColor = linecolor;
   datalines;
M m cx445694
F f cxA23A2E
;

proc print noobs;
run;

proc sgplot noautolegend data=sashelp.class dattrmap=attrmap;
   symbolchar name=m char='2642'x;
   symbolchar name=f char='2640'x;
   reg y=weight x=height / group=sex attrid=Stmt1 markerattrs=(size=15);
run;
```

The results are almost identical to the graph displayed in Figure 1.39. Attribute maps are an advanced topic and are covered in *SAS ODS Graphics: Procedures Guide* and *Advanced ODS Graphics Examples*. The attribute map is displayed in Figure 1.40. Attribute maps have special variables. The ID variable has the value `'Stmt1'`. Observations with this value are used in the REG statement, because it has the ATTRID=**Stmt1** option. There can be multiple ID values, and each ID group can be used in a different statement. The LinePattern variable has a value of 1 for a solid line. The Value variable contains the values that will be found in the DATA= data set (`'M'` and `'F'`). The MarkerSymbol variable contains the names of the corresponding SYMBOLCHAR statements (`'m'` and `'f'`). The LineColor variable contains the two line colors (a shade of blue for males and a shade of pink for females). The MarkerColor variable contains the same two marker colors.

Figure 1.40 Attribute Map

ID	LinePattern	Value	MarkerSymbol	LineColor	MarkerColor
Stmt1	1	M	m	cx445694	cx445694
Stmt1	1	F	f	cxA23A2E	cxA23A2E

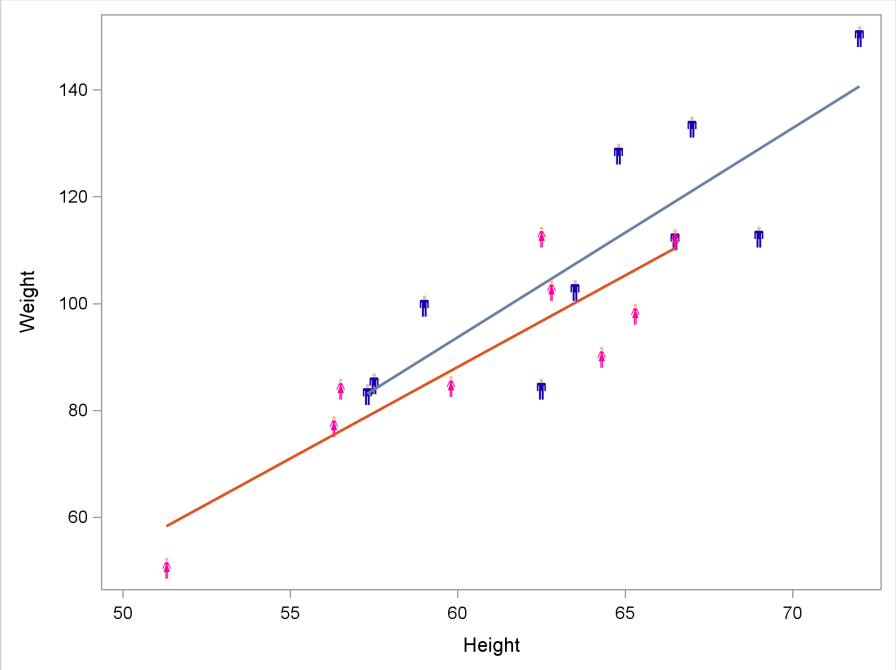
You can specify images as markers:

```
proc sgplot noautolegend data=class;
   symbolimage name=m image='images/male.png';
   symbolimage name=f image='images/female.png';
   reg y=mweight x=mheight / markerattrs=(symbol=m size=15);
   reg y=fweight x=fheight / markerattrs=(symbol=f size=15);
run;
```

```
proc sgplot noautolegend data=sashelp.class dattrmap=attrmap;
   symbolimage name=m image='images/male.png';
   symbolimage name=f image='images/female.png';
   reg y=weight x=height / group=sex attrid=Stmt1 markerattrs=(size=15);
run;

ods graphics on / reset=attrpriority;
```

The graphs are almost identical; one is displayed in Figure 1.41.

Figure 1.41 Controlling Markers



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 in 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.

1.5 Bar Chart

Double-Click for Example Code

A bar chart is a graph that has rectangular bars whose lengths are proportional to the values they represent (means, percentages, sums, and so on). Bar charts are used to compare the values of a quantitative variable at two or more levels of a categorical variable. The following steps show two ways to produce a horizontal bar chart of mileage by vehicle type:

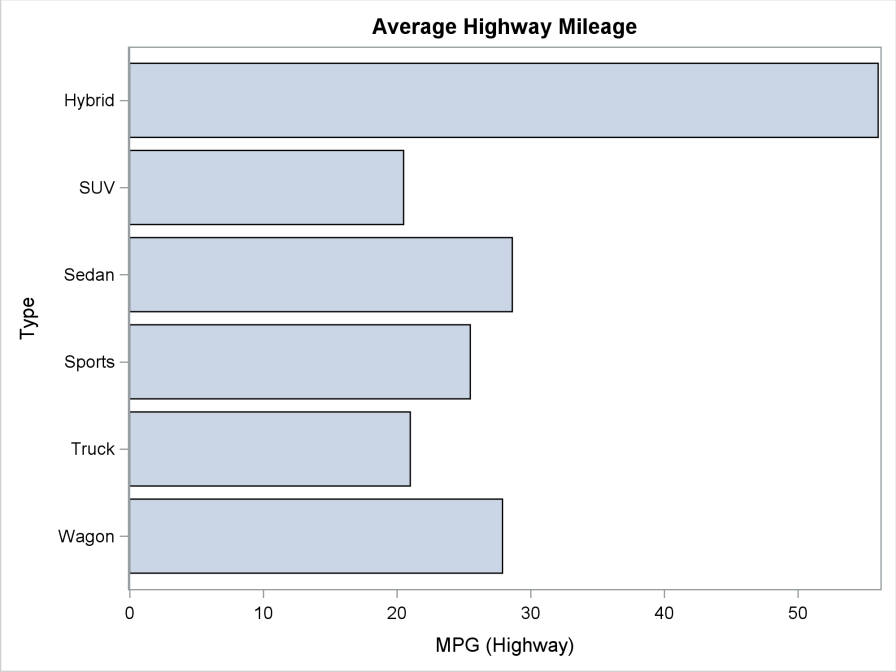
```
proc sgplot data=sashelp.cars;
   title 'Average Highway Mileage';
   hbar type / response=mpg_highway stat=mean;
run;

proc template;
   define statgraph barchart;
      begingraph;
         entrytitle 'Average Highway Mileage';
         layout overlay;
            barchart x=type y=mpg_highway / stat=mean orient=horizontal;
         endlayout;
      endgraph;
   end;
run;

proc sort data=sashelp.cars out=cars;
   by descending type;
run;

proc sgrender data=cars template=barchart;
run;
```

PROC SGPLOT uses an HBAR statement, which specifies the discrete variable first before the slash and specifies the response variable in the RESPONSE= option. Possible values for the STAT= option are FREQ, MEAN, MEDIAN, PERCENT, and SUM.

In the GTL, the ORIENT=HORIZONTAL option in the BARCHART statement creates a horizontal bar chart. With the STAT=MEAN option, the mean of the Y= variable is displayed for each category of the X= discrete variable. Possible values of the STAT= option are FREQ, PCT (percentages), PROPORTION, SUM, and MEAN. The data are sorted before PROC SGRENDER is run so that the vehicle types appear in the same sorted order that PROC SGPLOT uses. Without this step, the vehicle types appear in the order in which they appear in the original data set. You sort the data in descending order to display the bars in alphabetical order from top to bottom.

The graphs are identical; one is displayed in Figure 1.42.

Figure 1.42 Horizontal Bar Chart

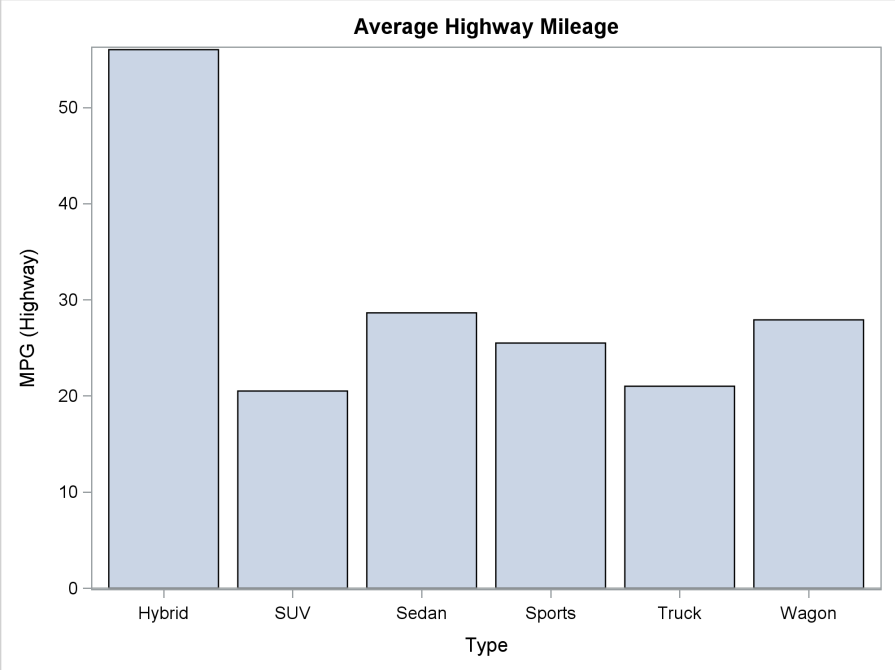


The following steps show two ways to produce a vertical bar chart of mileage by vehicle type:

```
proc sgplot data=sashelp.cars;
   title 'Average Highway Mileage';
   vbar type / response=mpg_highway stat=mean;
run;

proc template;
   define statgraph barchart;
      begingraph;
         entrytitle 'Average Highway Mileage';
         layout overlay;
            barchart x=type y=mpg_highway / stat=mean;
         endlayout;
      endgraph;
   end;
run;

proc sort data=sashelp.cars out=cars;
   by type;
run;

proc sgrender data=cars template=barchart;
run;
```

You sort the data in ascending order to display the bars in alphabetical order from left to right. The other options here are an obvious variation on the options in the previous steps. In the GTL, the default orientation is ORIENT=VERTICAL, and in PROC SGPLOT the VBAR statement produces a vertical bar chart. The graphs are identical; one is displayed in Figure 1.43.

Figure 1.43 Vertical Bar Chart

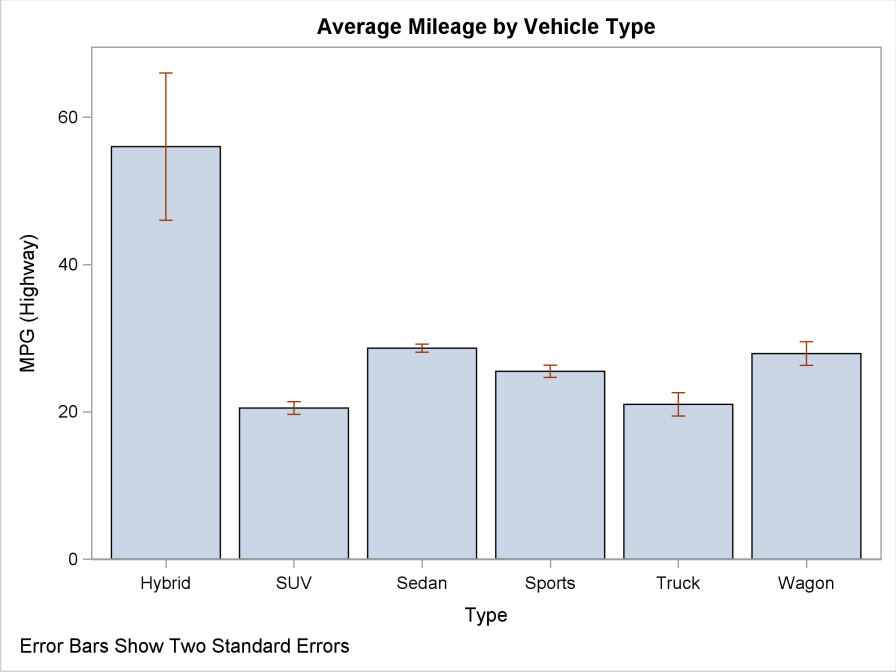


The following step adds to the vertical bar chart error bars, showing two standard errors by using the NUMSTD=2 and LIMITSTAT=STDERR options:

```
proc sgplot data=sashelp.cars noautolegend;
   title 'Average Mileage by Vehicle Type';
   vbar type / response=mpg_highway stat=mean numstd=2 limitstat=stderr;
   footnote justify=left 'Error Bars Show Two Standard Errors';
   yaxis    offsetmin=0 offsetmax=0.05;
run;

footnote;
```

The results of this step are displayed in Figure 1.44.

Figure 1.44 Vertical Bar Chart with Error Bars



The preceding bar chart examples provide ODS Graphics with raw data and multiple observations in each group. Through either the BARCHART statement in PROC SGRENDER or the VBAR or HBAR statement in PROC SGPLOT, ODS Graphics does the computations to get the means, percentages, or other statistics and produce the results. The remainder of this section uses "PARM" statements (including HBARPARM and VBARPARM in PROC SGLOT, BARCHARTPARM in the GTL, and many others that produce other types of graphs). Statements whose name contains "PARM" do not do computations to summarize the data. You provide a data set that has the summarized information (in this case, one observation per group) that needs to be displayed. The advantage of using a PARM statement is that it gives you precise control over how the results are created. You can create this data set from the raw data by using a procedure such as PROC SUMMARY or PROC MEANS. The following steps use the VBARPARM statement to produce a bar chart that shows group means with error bars:

```
proc summary data=sashelp.cars nway;
   class type;
   var mpg_highway;
   output out=mileage mean=mean stderr=stderr;
run;

data m2;
   set mileage;
   low = mean - 2 * stderr;
   hi  = mean + 2 * stderr;
run;
```

```
proc sgplot data=m2 noautolegend;
   title 'Average Mileage by Vehicle Type';
   vbarparm category=type response=mean;
   highlow  high=hi low=low x=type / highcap=serif lowcap=serif
            lineattrs=GraphError;
   yaxis    offsetmin=0 offsetmax=0.05;
   footnote justify=left 'Error Bars Show Two Standard Errors';
run;

footnote;
```

The results are almost identical to the graph displayed in Figure 1.44.

The following steps use the BARCHARTPARM statement to produce a vertical bar chart that shows group means and error bars:

```
proc summary data=sashelp.cars nway;
   class type;
   var mpg_highway;
   output out=mileage mean=mean stderr=stderr;
run;

proc template;
   define statgraph barchartparm;
      begingraph;
         entrytitle    "Average Mileage by Vehicle Type";
         entryfootnote halign=left textattrs=GraphValueText
                       "Error Bars Show Two Standard Errors";
         layout overlay / yaxisopts=(offsetmin=0 offsetmax=0.05);
            barchartparm x=type y=mean /
                      errorlower=eval(mean - 2 * stderr)
                      errorupper=eval(mean + 2 * stderr);
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=mileage template=barchartparm;
run;
```

The results are almost identical to the graph displayed in Figure 1.44.

PROC SUMMARY creates an output data set that contains descriptive statistics. The NWAY option outputs only *n*-way (in this case one-way) frequencies. This suppresses the creation of an observation with the overall mean. One observation is created for each level of the CLASS statement variable Type with the group means computed from the variable MPG_Highway. The means are stored in the variable Mean, and the standard errors are stored in the variable StdErr.

In the GTL, the ENTRYTITLE statement provides the title and the ENTRYFOOTNOTE statement provides the footnote. Options in the ENTRYFOOTNOTE statement control the position and the appearance of the footnote. The HALIGN=LEFT option aligns the footnote on the left; the default is centered. The TEXTATTRS= option uses the **`GraphValueText`** style element to display the footnote. The **`GraphFootnoteText`** style element is the default, which in the HTMLBlue style uses an italic font. The X-axis options put the origin at 0 and add a small amount (5%) of padding at the right side of the X axis.

The BARCHARTPARM statement specifies the variable Type in the X= option and the variable Mean in the Y= option. These variables are computed in the PROC SUMMARY step that precedes the PROC TEMPLATE step. The GTL can perform computations and use variables that are functions of the input data. These computations are always performed in the context of the EVAL function. This BARCHARTPARM statement creates lower and upper error bars by using the syntax **ErrorLower=eval(Mean - 2 * StdErr) ErrorUpper=eval(Mean + 2 * StdErr)**. The variables Mean and StdErr are computed by PROC SUMMARY. The expressions in the EVAL function add and subtract two standard errors from the mean to make the error bars.

The following steps use the HBARPARM statement in PROC SGPLOT and the ORIENT=HORIZONTAL option in the GTL to create a horizontal bar chart:

```
proc sgplot data=m2 noautolegend;
   title 'Average Mileage by Vehicle Type';
   hbarparm category=type response=mean;
   highlow  high=hi low=low y=type / highcap=serif lowcap=serif
            lineattrs=GraphError;
   xaxis offsetmin=0 offsetmax=0.05;
   footnote justify=left 'Error Bars Show Two Standard Errors';
run;

footnote;

proc template;
   define statgraph barchartparm;
      begingraph;
         entrytitle    "Average Mileage by Vehicle Type";
         entryfootnote halign=left textattrs=GraphValueText
                       "Error Bars Show Two Standard Errors";
         layout overlay;
            barchartparm x=type y=mean / orient=horizontal
                     datatransparency=0.75
                     errorlower=eval(mean - 2 * stderr)
                     errorupper=eval(mean + 2 * stderr);
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=mileage template=barchartparm;
run;
```

By default, PROC SGPLOT creates bars that have the default transparency (0), as shown in Figure 1.45. In Figure 1.46, the GTL option DATATRANSPARENCY=0.75 option makes the bars 75% as transparent as the default. Values range from 0 to 1; values closer to 1 create bars that are nearly invisible. The default is DATATRANSPARENCY=0. By default, the two axes are reversed.

Figure 1.45 Parameterized Horizontal Bar Chart (SGPLOT)

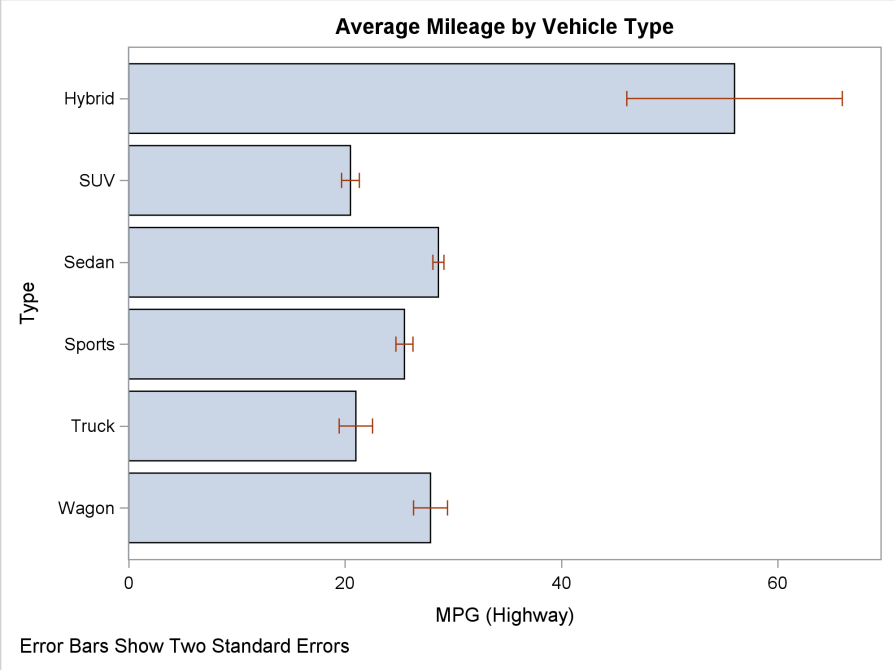
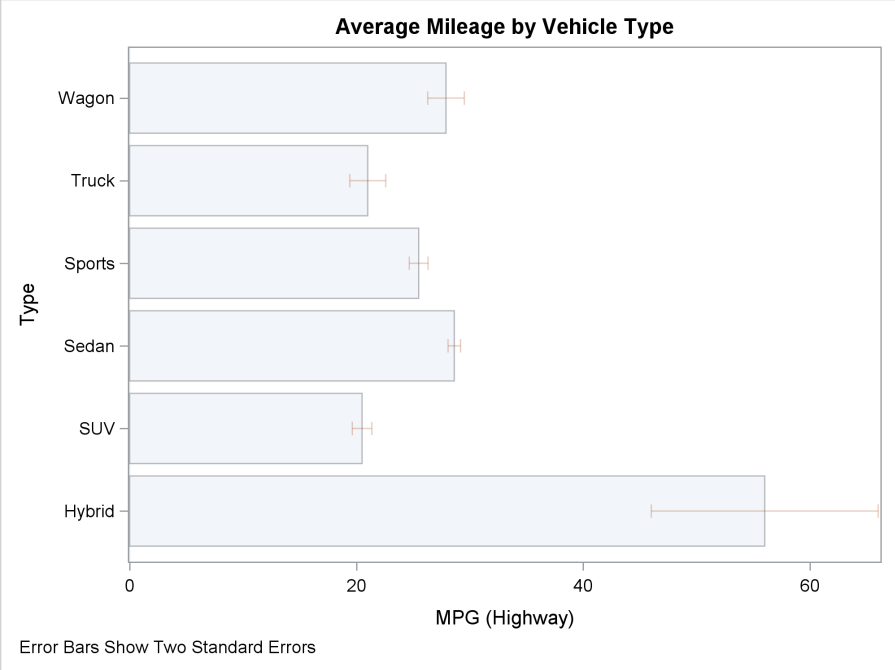


Figure 1.46 Parameterized Horizontal Bar Chart (GTL)

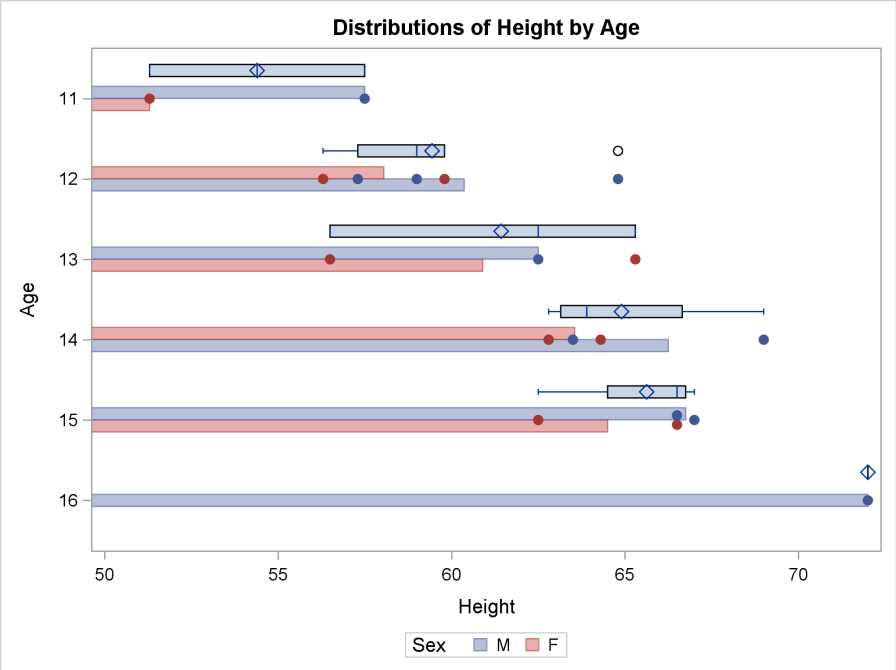


There is another class of bar chart statements in PROC SGPLOT: the basic bar chart statements HBARBASIC and VBARBASIC. In PROC SGPLOT, you cannot overlay ordinary bar charts on basic plots such as scatter plots. You can overlay the basic bar charts on other basic plots. The following step uses the HBARBASIC statement to illustrate this and creates the graph in Figure 1.77:

```
proc sgplot data=sashelp.class;
  title 'Distributions of Height by Age';
  hbox height   / category=age discreteoffset=-0.35 boxwidth=.15;
  hbarbasic age / response=height stat=mean
                  group=sex groupdisplay=cluster
                  clusterwidth=0.3 transparency=0.5;
  scatter x=height y=age / group=sex markerattrs=(symbol=circlefilled) jitter;
  xaxis min=50;
run;
```

The graph contains a clustered bar chart. There is one pair of bars for each age. The upper blue bars show the average height for males, and the lower red bars show the average height for females. The cluster width makes the bars narrower than they would be by default. This reserves space for the box plots, which are offset by –0.35, thus moving them up so that they do not overlap the bars. The scatter plot is overlaid on top of the bars. The JITTER option moves coincident points. Without this option, you would not see that there are three 15-year-olds whose height is above 65 inches. The bars are 50% transparent, which makes it easier to see the overlaid scatter plot. The XAXIS statement starts the graph at Height=50 instead of the default Height=0 to avoid squashing the box plots.

Figure 1.47 Horizontal Bar Chart, Box Plot, and Scatter Plot Overlaid

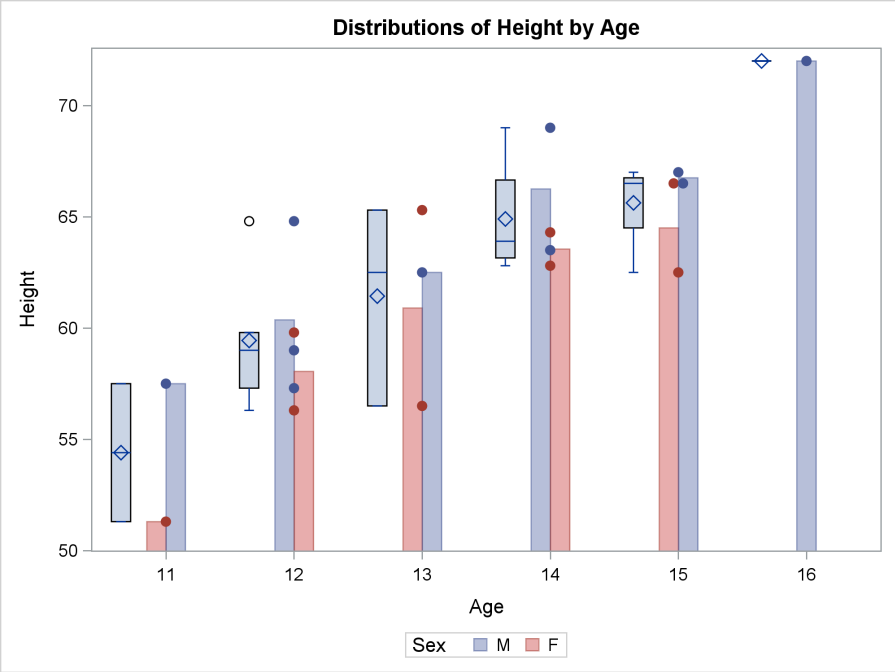


The following step is similar to the preceding step and illustrates the VBARBASIC statement:

```
proc sgplot data=sashelp.class;
  title 'Distributions of Height by Age';
  vbox height   / category=age discreteoffset=-0.35 boxwidth=.15;
  vbarbasic age / response=height stat=mean
                  group=sex groupdisplay=cluster
                  clusterwidth=0.3 transparency=0.5;
  scatter y=height x=age / group=sex markerattrs=(symbol=circlefilled) jitter;
  yaxis offsetmin=0 min=50;
run;
```

The results are displayed in Figure 1.48.

Figure 1.48 Vertical Bar Chart, Box Plot, and Scatter Plot Overlaid



1.6 Multiple Axes

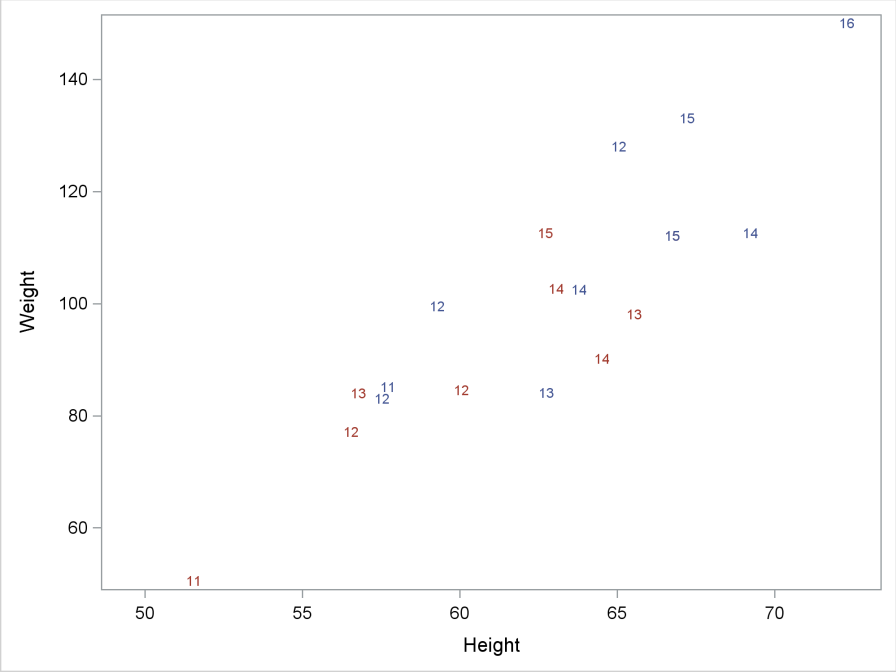
Double-Click for Example Code

A graph can have up to four independent axes: X, Y, X2, and Y2. All graphs up to this point have an X axis (on the bottom) and a Y axis (on the left). In addition or instead, graphs can have an X2 axis (on top) or a Y2 axis (on the right). The following step creates an ordinary scatter plot:

```
title;
proc sgplot data=sashelp.class noautolegend;
   scatter y=weight x=height / markerchar=age group=sex jitter;
run;
```

The JITTER option slightly offsets coincident values. The results are displayed in Figure 1.49.

Figure 1.49 X and Y Axes

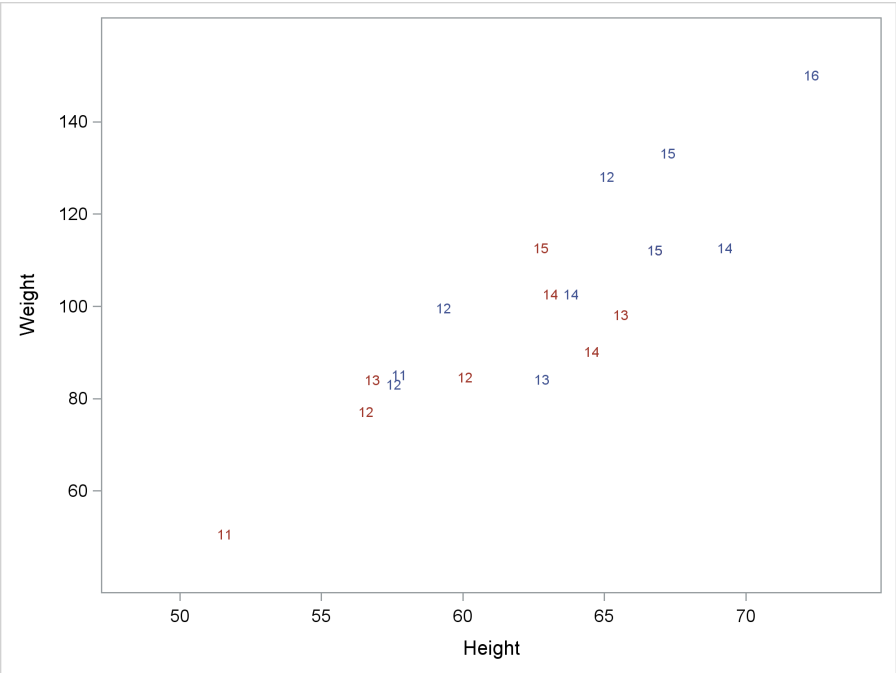


The following step uses the XAXIS and YAXIS statements to add offsets to each axis:

```
proc sgplot data=sashelp.class noautolegend;
   scatter y=weight x=height / markerchar=age group=sex jitter;
   xaxis offsetmin=0.1 offsetmax=0.1;
   yaxis offsetmin=0.1 offsetmax=0.1;
run;
```

The results are displayed in Figure 1.50.

Figure 1.50 X and Y Axes with Offsets

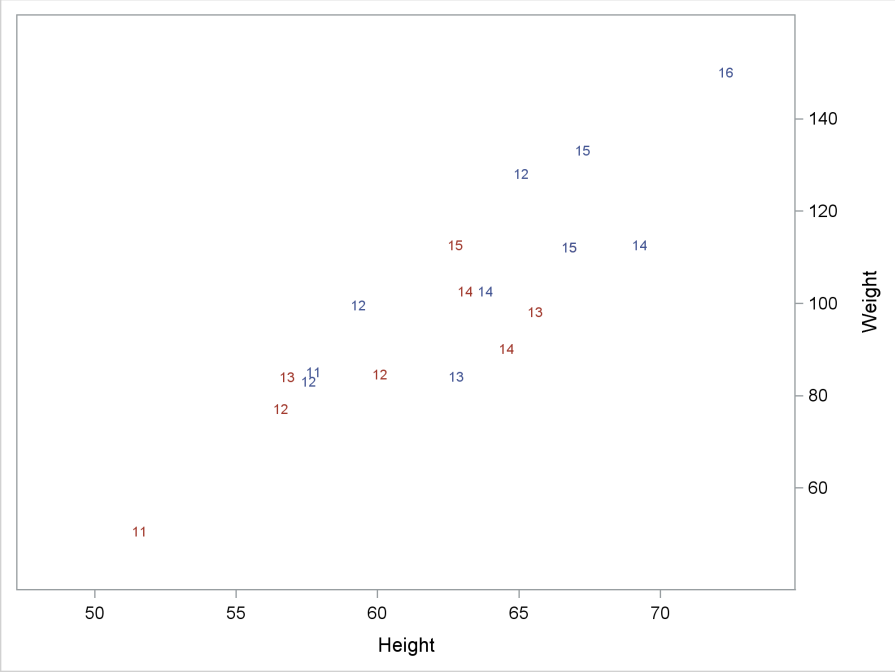


The following step creates a graph that has an X axis and a Y2 axis:

```
proc sgplot data=sashelp.class noautolegend;
   scatter y=weight x=height / markerchar=age group=sex jitter y2axis;
   xaxis offsetmin=0.1 offsetmax=0.1;
   y2axis offsetmin=0.1 offsetmax=0.1;
run;
```

The Y2AXIS statement controls the appearance of the Y2 axis. The results are displayed in Figure 1.51.

Figure 1.51 X and Y2 Axes

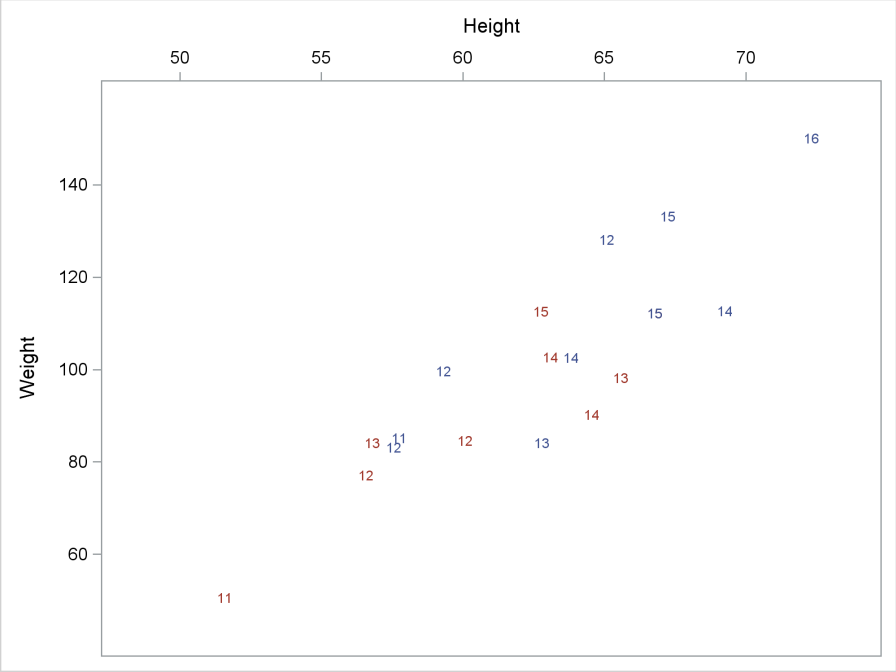


The following step creates a graph that has an X2 axis and a Y axis:

```
proc sgplot data=sashelp.class noautolegend;
   scatter y=weight x=height / markerchar=age group=sex jitter x2axis;
   x2axis offsetmin=0.1 offsetmax=0.1;
   yaxis offsetmin=0.1 offsetmax=0.1;
run;
```

The X2AXIS statement controls the appearance of the X2 axis. The results are displayed in Figure 1.52.

Figure 1.52 X2 and Y Axes

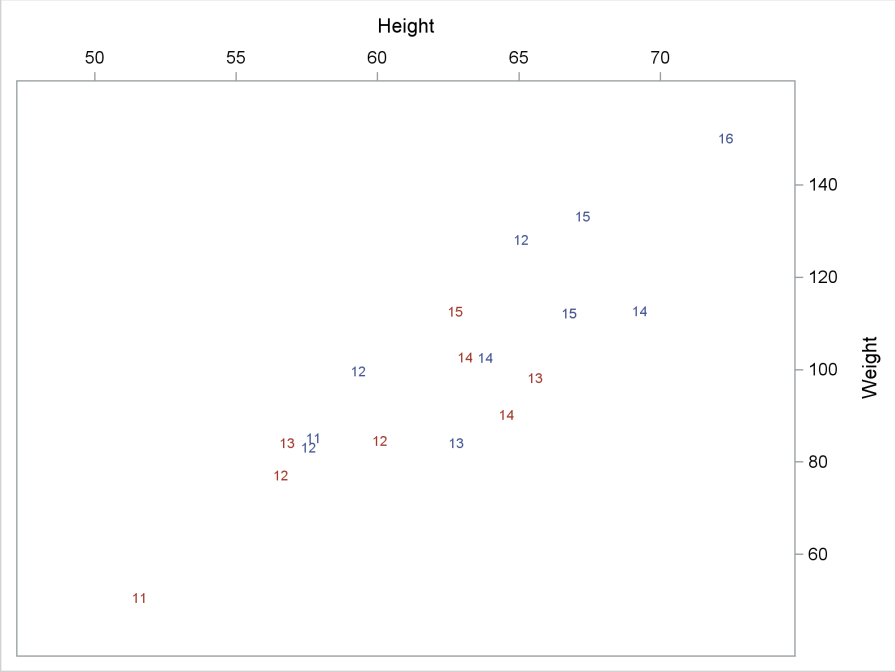


The following step creates a graph that has an X2 axis and a Y2 axis:

```
proc sgplot data=sashelp.class noautolegend;
   scatter y=weight x=height / markerchar=age group=sex jitter x2axis y2axis;
   x2axis offsetmin=0.1 offsetmax=0.1;
   y2axis offsetmin=0.1 offsetmax=0.1;
run;
```

The results are displayed in Figure 1.53.

Figure 1.53 X2 and Y2 Axes

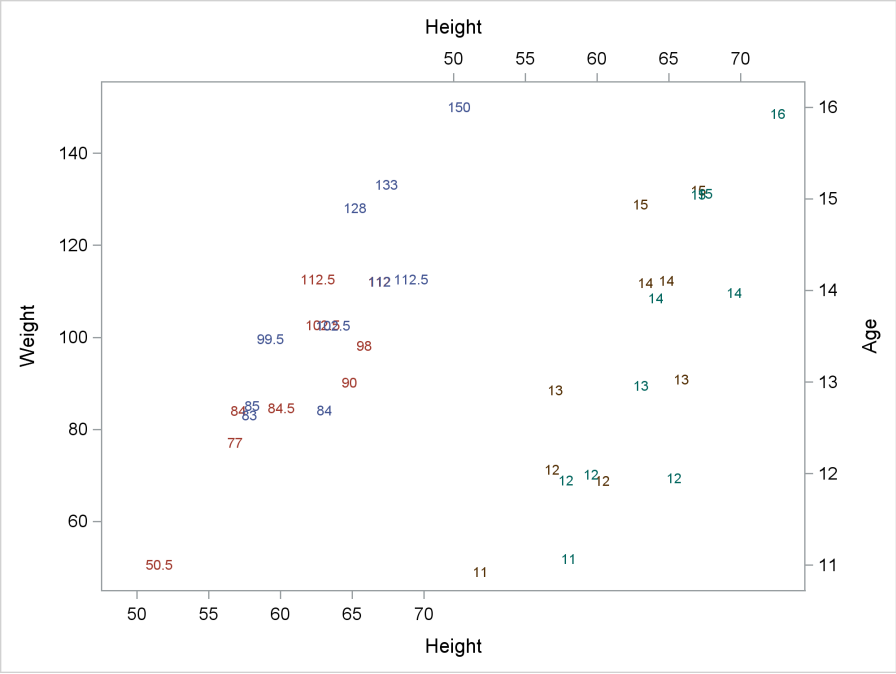


The following step creates nonoverlapping scatter plots by using offsets along with X, X2, Y, and Y2 axes:

```
proc sgplot data=sashelp.class noautolegend;
   scatter y=weight x=height / markerchar=weight group=sex jitter;
   scatter y=age    x=height / markerchar=age    group=sex jitter x2axis y2axis;
   xaxis  offsetmin=0.05 offsetmax=0.5;
   x2axis offsetmin=0.5  offsetmax=0.05;
   yaxis  offsetmin=0.05 offsetmax=0.05;
   y2axis offsetmin=0.05 offsetmax=0.05;
run;
```

Two scatter plots are overlaid on each other, but the options XAXIS OFFSETMAX=0.5 and X2AXIS OFFSETMIN=0.5 prevent the two graphs from overlapping. The results are displayed in Figure 1.54.

Figure 1.54 X, X2, Y, and Y2 Axes

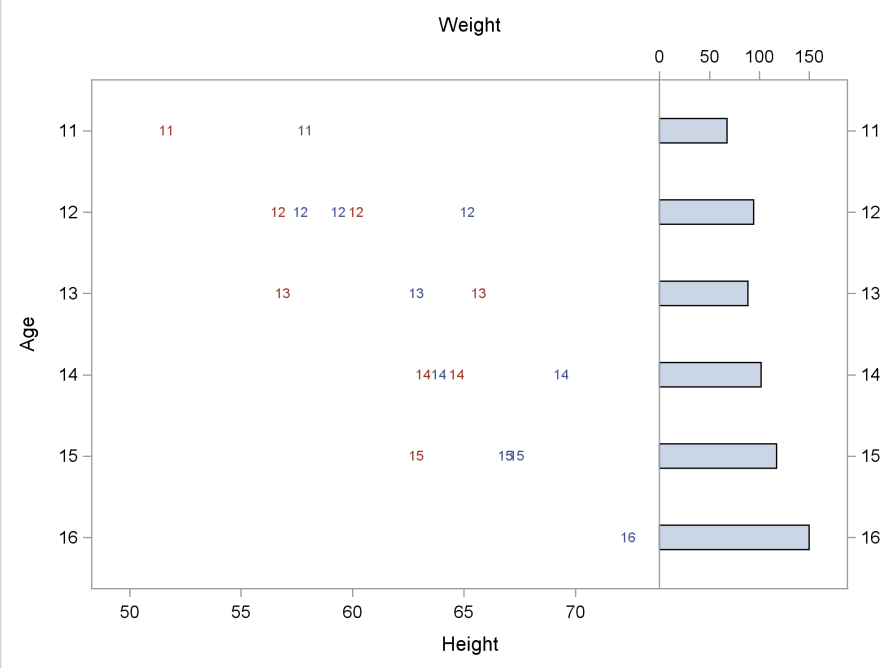


The following step uses X, X2, Y, and Y2 axes along with offsets to create a scatter plot and a bar chart:

```
proc sgplot data=sashelp.class noautolegend;
   scatter y=age x=height / markerchar=age group=sex;
   hbarbasic age / response=weight stat=mean x2axis y2axis barwidth=0.3;
   xaxis  offsetmin=0.05 offsetmax=0.30;
   x2axis offsetmin=0.75 offsetmax=0.05;
   yaxis  offsetmin=0.10 offsetmax=0.10;
   y2axis offsetmin=0.10 offsetmax=0.10 display=(nolabel);
run;
```

The left part of the graph displays a scatter plot, and the right part displays a bar chart of average weight for each of the age groups. The results are displayed in Figure 1.55.

Figure 1.55 Using Multiple Axes, Offsets, and Different Graph Types



There are many other ways to use multiple axes.

In the GTL, you specify axis option in the XAXISOPTS=, YAXISOPTS=, X2AXISOPTS=, and Y2AXISOPTS= options in the LAYOUT OVERLAY statement. You use the XAXIS= and YAXIS= options to specify which axis to use for each statement. The following steps illustrate:

```
proc template;
   define statgraph overlaid;
      begingraph;
         layout overlay / xaxisopts=(offsetmin=0.05 offsetmax=0.3)
                          yaxisopts=(reverse=true offsetmin=0.1 offsetmax=0.1)
                         x2axisopts=(offsetmin=0.75 offsetmax=0.05)
                         y2axisopts=(reverse=true display=(ticks tickvalues line)
                                     offsetmin=0.1 offsetmax=0.1 type=discrete);
            scatterplot x=height y=age / group=sex markercharacter=age;
```

```
            barchart x=age y=weight / xaxis=x2 yaxis=y2 orient=horizontal
                                      barwidth=0.3 stat=mean;
         endlayout;
      endgraph;
   end;
run;

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

The results match those displayed in Figure 1.55.

1.7 Axis Tables

Double-Click for Example Code

You can use an axis table to display a graph and a table simultaneously. The following steps create and display a data set that has the average manufacturer's suggested retail price, miles per gallon in city driving, miles per gallon in highway driving, and horsepower, along with the frequency of occurrence of the three origins (Asia, Europe, and United States):

```
proc summary data=sashelp.cars;
   class origin;
   var mpg: horsepower msrp;
   ways 1;
   output out=means(where=(_stat_='MEAN') drop=_type_);
   format msrp comma6. hor: 6.2;
run;

proc print;
run;
```

The statistics that the graph will show are displayed in Figure 1.56.

Figure 1.56 X-Axis Table Input Data

Obs	Origin	_FREQ_	_STAT_	MPG_City	MPG_Highway	Horsepower	MSRP
1	Asia	158	MEAN	22.0127	28.2658	190.70	24,741
2	Europe	123	MEAN	18.7317	26.0081	251.89	48,350
3	USA	147	MEAN	19.0748	26.0136	212.82	28,377

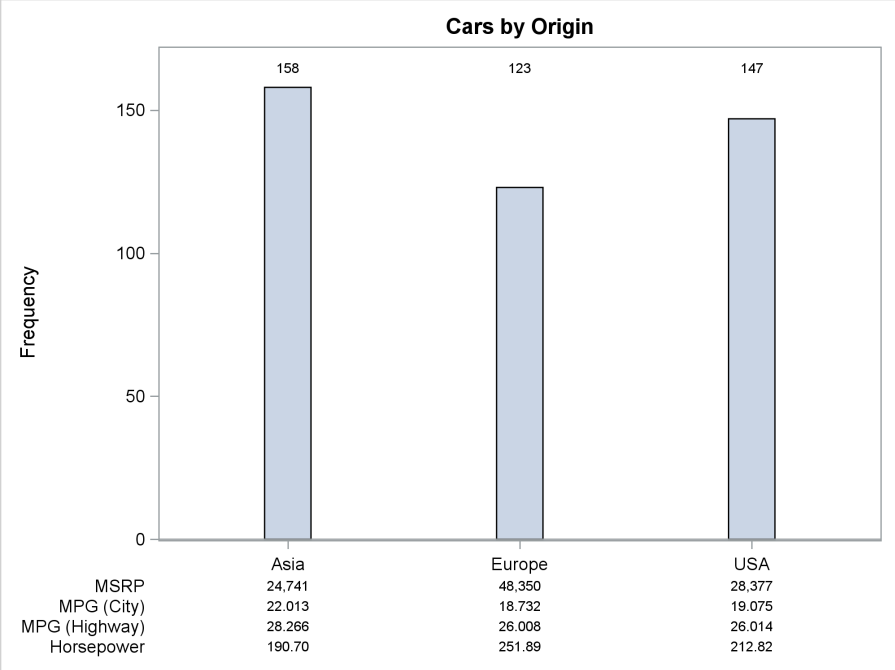
The following step creates a vertical bar chart that shows the frequency of occurrence of the three origins along with averages of the other variables:

```
proc sgplot data=means;
   title 'Cars by Origin';
   vbarparm category=origin response=_freq_ / barwidth=0.2;
   xaxistable msrp mpg: horse:;
   xaxistable _freq_ / location=inside position=top nolabel pad=(top=10px);
   xaxis display=(nolabel);
   yaxis label='Frequency';
run;
```

The VBARPARM statement creates the vertical bar chart. The categorical variable is Origin, and the response variable is the frequency count _Freq_. The bar width is set to 20% of the default.

The results are displayed in Figure 1.57. The first XAXISTABLE statement creates the table below the graph. By default, X-axis tables are displayed outside and below the graph. The second XAXISTABLE statement creates the table above the bars, which displays the frequencies. The results are displayed in Figure 1.57.

Figure 1.57 X-Axis Table



The following steps make the same image by using the GTL:

```
proc template;
   define statgraph axistab;
      begingraph;
      entrytitle "Cars by Origin";
         layout lattice / rowweights=preferred;
            layout overlay / xaxisopts=(display=(ticks tickvalues line))
                             yaxisopts=(label="Frequency");
               barchartparm x=origin y=_freq_ / barwidth=0.2;
               innermargin / align=top;
                  axistable value=_freq_ x=origin / labelposition=min
                                                    pad=(top=10px) display=(values);
               endinnermargin;
            endlayout;
            layout overlay / xaxisopts=(display=none) walldisplay=none;
               axistable value=msrp x=origin;
            endlayout;
            layout overlay / xaxisopts=(display=none) walldisplay=none;
               axistable value=mpg_city x=origin;
            endlayout;
            layout overlay / xaxisopts=(display=none) walldisplay=none;
               axistable value=mpg_highway x=origin;
            endlayout;
```

```
            layout overlay / xaxisopts=(display=none) walldisplay=none;
               axistable value=horsepower x=origin;
            endlayout;
         endlayout;
      endgraph;
   end;
run;

proc sgrender template=axistab data=means;
run;
```

The LAYOUT LATTICE block provides an outer container for the five LAYOUT OVERLAY blocks. The option ROWWEIGHTS=PREFERRED is used with lattice layouts that mix one-dimensional and two-dimensional plots in the grid. It enables the layout to compute the weights automatically for rows that contain one-dimensional plots. If you do not specify this option, then each overlay is allocated the same amount of vertical space, resulting in a squashed graph and extra white space between the axis table rows. The axis table that displays the frequencies is included in an INNERMARGIN block. The AXISTABLE statement option DISPLAY=(VALUES) suppresses the label (which would have been "_Freq_"). Each of the exterior axis tables appears in a separate overlay layout. No axis information or wall (box around each row) is displayed.

Because many options are duplicated, you can make the PROC TEMPLATE step simpler by using the macro language as follows:

```
proc template;
   define statgraph axistab;
      begingraph;
      entrytitle "Cars by Origin";
         layout lattice / rowweights=preferred;
            layout overlay / xaxisopts=(display=(ticks tickvalues line))
                             yaxisopts=(label="Frequency");
               barchartparm x=origin y=_freq_ / barwidth=0.2;
               innermargin / align=top;
                  axistable value=_freq_ x=origin / labelposition=min
                                                    pad=(top=10px) display=(values);
               endinnermargin;
            endlayout;
            %macro axist(name);
               layout overlay / xaxisopts=(display=none) walldisplay=none;
                  axistable value=&name x=origin;
               endlayout;
            %mend;
            %axist(msrp)
            %axist(mpg_city)
            %axist(mpg_highway)
            %axist(horsepower)
         endlayout;
      endgraph;
   end;
run;
```

The following example constructs a Y-axis table that displays autocorrelations:

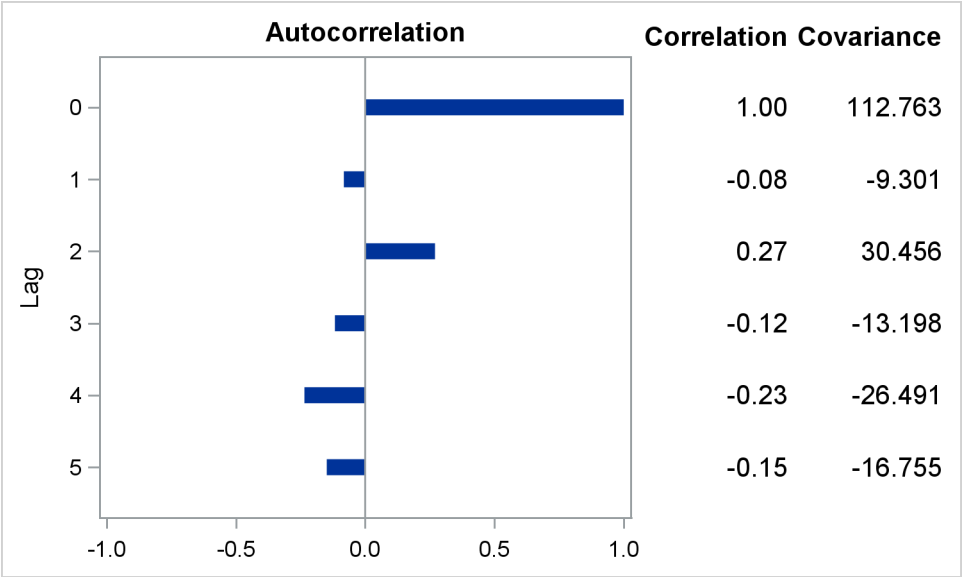
```
proc autoreg data=sashelp.class;
   ods output corrgraph=cg;
   model weight = height / nlag=5;
run;

data cg2;
   set cg;
   lo = ifn(autocorr ge 0, 0, autocorr);
   hi = ifn(autocorr ge 0, autocorr, 0);
run;

ods graphics on / height=3in width=5in subpixel=on;
title;

proc sgplot noautolegend;
   scatter    x=lo y=lag        / x2axis markerattrs=(size=0);
   yaxistable autocorr autocov  / valueattrs=(size=15px)
                                  labelattrs=(size=15px weight=bold);
   highlow low=lo high=hi y=lag / lineattrs=(thickness=0.1in);
   yaxis   reverse;
   xaxis   values=(-1 to 1 by 0.5) display=(nolabel);
   x2axis  display=(noticks novalues) label='Autocorrelation'
           labelattrs=(size=15px weight=bold);
   refline 0 / axis=x;
   format autocov 7.3 autocorr 5.2;
run;
```

The graph is displayed in Figure 1.58. It consists of a high-low plot that displays the autocorrelations and a table that displays the autocorrelations and autocovariances. 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.

The YAXISTABLE statement creates the Y-axis table for the variables AutoCorr and AutoCov. Additional options produce bold headers and increase the font sizes. The FORMAT statement sets the formats for the columns of the axis table.

The HIGHLOW statement creates the high-low plot. The REFLINE statement draws a vertical reference line at 0. The remaining statements control the axes and formats. The SCATTER statement produces an invisible plot (which has markers of size 0) and uses the X2 axis. Its only purpose is to provide a graph that specifies the X2AXIS option so that the X2AXIS statement can put the label "Autocorrelation" above the plot. Other ways to do this are available, but invisible plots are often handy when you want to do something slightly nonstandard. The YAXIS statement reverses the Y axis so that lags increase as you move down the axis. The XAXIS statement controls the ticks and suppresses the label.

Figure 1.58 Y-Axis Table



You can make an identical graph by using the GTL as follows:

```
proc template;
   define statgraph yaxistab;
      begingraph;
         layout lattice / columnweights=preferred columns=3;
            layout overlay / xaxisopts=(display=(ticks tickvalues line)
                                        linearopts=(tickvaluelist=
                                                    (-1 -0.5 0 0.5 1)
                                        viewmin=-1 viewmax=1))
                             yaxisopts=(reverse=true)
                             x2axisopts=(display=(line label)
                                         label="Autocorrelation"
                                         labelattrs=(size=15px weight=bold));
               scatterplot x=lo y=lag / xaxis=x2 markerattrs=(size=0);
               highlowplot y=lag high=hi low=lo / lineattrs=(thickness=0.1in);
               referenceline x=0;
            endlayout;
            layout overlay / yaxisopts=(reverse=true display=none)
                             walldisplay=none;
               axistable value=autocorr y=lag / labelposition=min
                         labelattrs=(size=15px weight=bold)
                         valueattrs=(size=15px) display=(label);
            endlayout;
            layout overlay / yaxisopts=(reverse=true display=none)
                             walldisplay=none;
               axistable value=autocov y=lag / labelposition=min
                         labelattrs=(size=15px weight=bold)
                         valueattrs=(size=15px) display=(label);
            endlayout;
         endlayout;
      endgraph;
   end;
run;
```

```
proc sgrender template=yaxistab data=cg2;
   format autocov 7.3 autocorr 5.2;
run;

ods graphics on / reset=all subpixel=on;
```

The following steps show other examples of creating axis tables and produce the results in Figure 1.59 through Figure 1.64:

```
data PhD;
   input Science $ 1-19 y1973-y1978;
   label y1973 = '1973' y1974 = '1974' y1975 = '1975'
         y1976 = '1976' y1977 = '1977' y1978 = '1978';
   datalines;
Life Sciences       4489 4303 4402 4350 4266 4361
Physical Sciences   4101 3800 3749 3572 3410 3234
Social Sciences     3354 3286 3344 3278 3137 3008
Behavioral Sciences 2444 2587 2749 2878 2960 3049
Engineering         3338 3144 2959 2791 2641 2432
Mathematics         1222 1196 1149 1003  959  959
;

proc corresp data=PhD out=Results short chi2p;
   var y1973-y1978;
   id Science;
run;

proc lifetest data=sashelp.BMT
              plots=survival(maxlen=13 atrisk=0 to 2500 by 500);
   time T * Status(0);
   strata Group / test=logrank adjust=sidak;
run;

proc lifetest data=sashelp.BMT
              plots=survival(maxlen=13 atrisk=0 to 2500 by 500 outside);
   time T * Status(0);
   strata Group / test=logrank adjust=sidak;
run;

proc reg data=sashelp.class;
   model weight = height / r;
   id name;
quit;

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

Figure 1.59 Correspondence Analysis

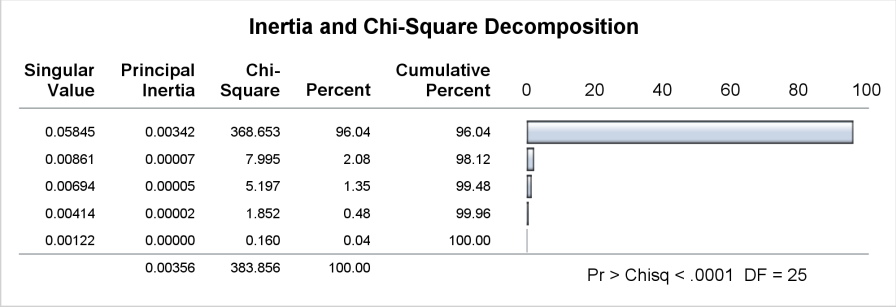


Figure 1.60 Survival Plot, Table Inside

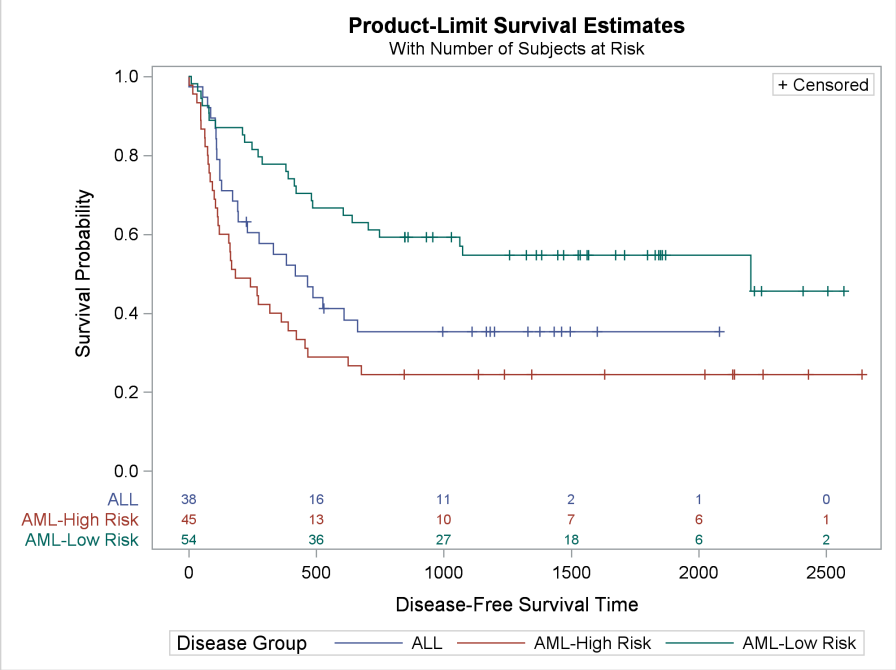


Figure 1.61 Survival Plot, Table Outside

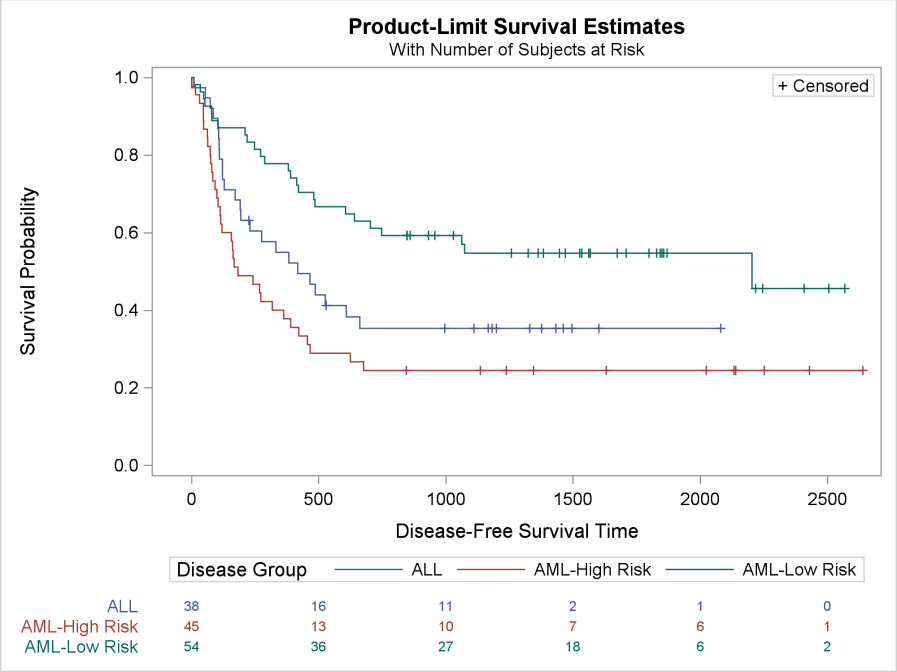


Figure 1.62 Studentized Residuals and Cook's D

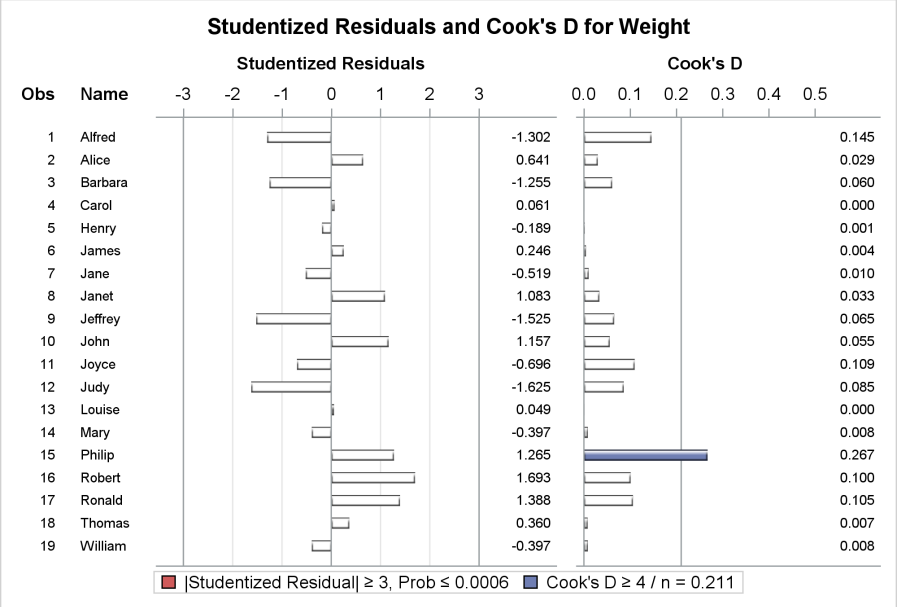
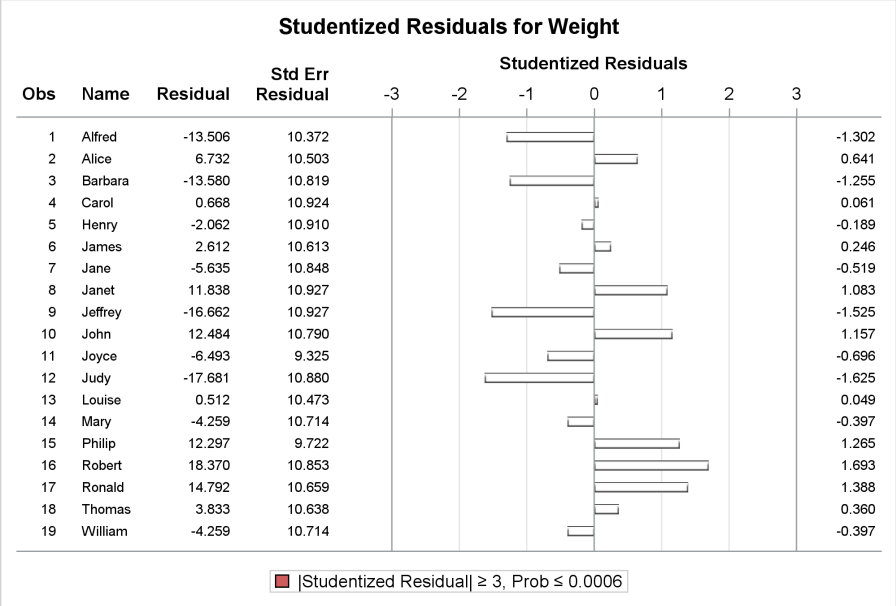
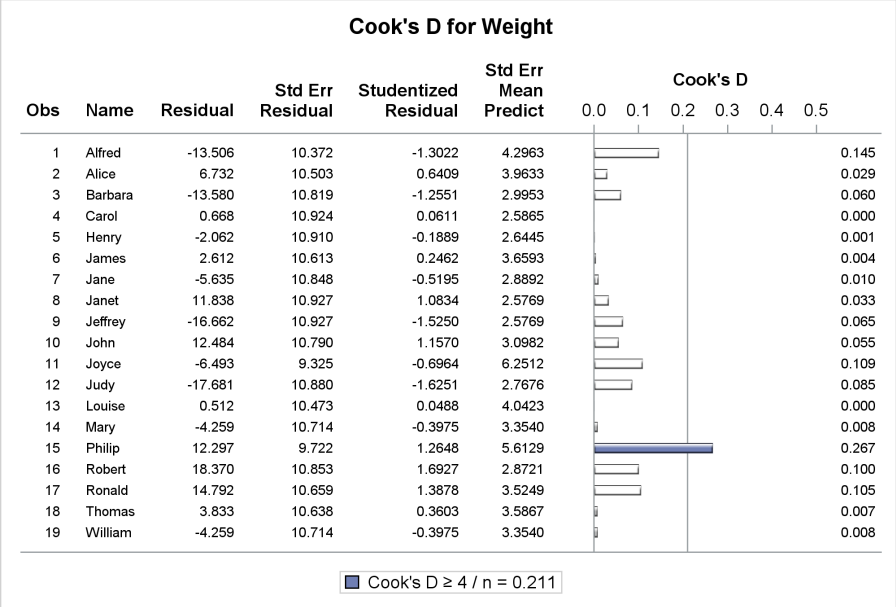


Figure 1.63 Studentized Residuals



Obs	Name	Residual	Std Err Residual	
1	Alfred	-13.506	10.372	-1.302
2	Alice	6.732	10.503	0.641
3	Barbara	-13.580	10.819	-1.255
4	Carol	0.668	10.924	0.061
5	Henry	-2.062	10.910	-0.189
6	James	2.612	10.613	0.246
7	Jane	-5.635	10.848	-0.519
8	Janet	11.838	10.927	1.083
9	Jeffrey	-16.662	10.927	-1.525
10	John	12.484	10.790	1.157
11	Joyce	-6.493	9.325	-0.696
12	Judy	-17.681	10.880	-1.625
13	Louise	0.512	10.473	0.049
14	Mary	-4.259	10.714	-0.397
15	Philip	12.297	9.722	1.265
16	Robert	18.370	10.853	1.693
17	Ronald	14.792	10.659	1.388
18	Thomas	3.833	10.638	0.360
19	William	-4.259	10.714	-0.397

Figure 1.64 Cook's D



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

Figure 1.59 through Figure 1.64 show various types of axis tables. Figure 1.62 shows that you can have multiple tables and multiple graphs within a single panel.

1.8 Histogram

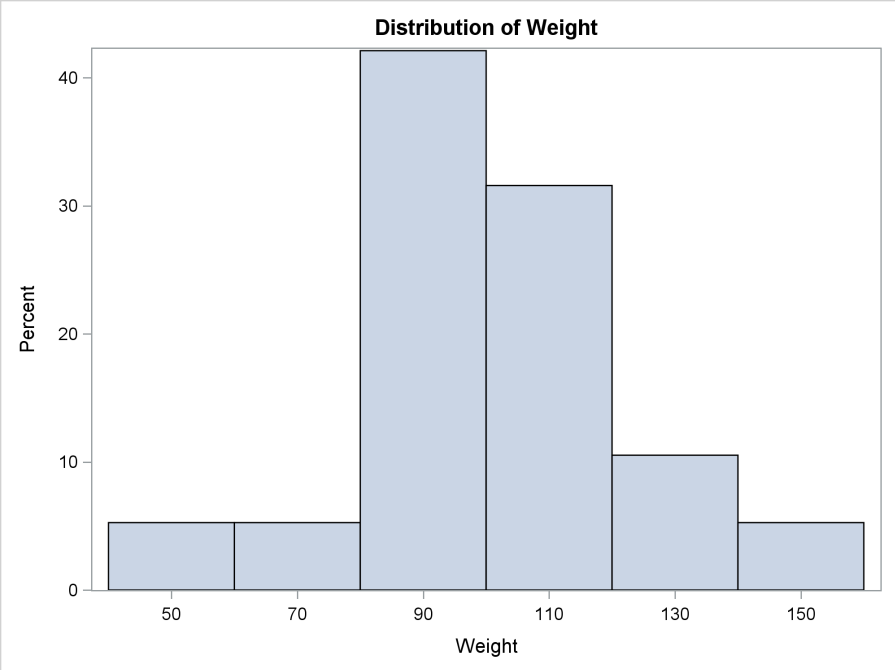
Double-Click for Example Code

A histogram displays tabulated frequencies, percentages, or proportions as bars. Each bar represents a nonoverlapping interval of a quantitative variable. The following steps produce a histogram and show the distribution of weight in the Sashelp.Class data set:

```
proc sgplot data=sashelp.class;
   title 'Distribution of Weight';
   histogram weight / showbins;
run;

proc template;
   define statgraph classhist;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogram weight;
         endlayout;
      endgraph;
   end;
run;

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

The HISTOGRAM statements in both PROC SGPLOT and the GTL name the variable to display. In PROC SGPLOT, the SHOWBINS option specifies that the midpoints of the value bins be used to create the tick marks for the horizontal axis. By default, the tick marks are created at regular intervals based on the minimum and maximum values. The graphs are identical; one is displayed in Figure 1.65.

Figure 1.65 Histogram with Midpoint Ticks

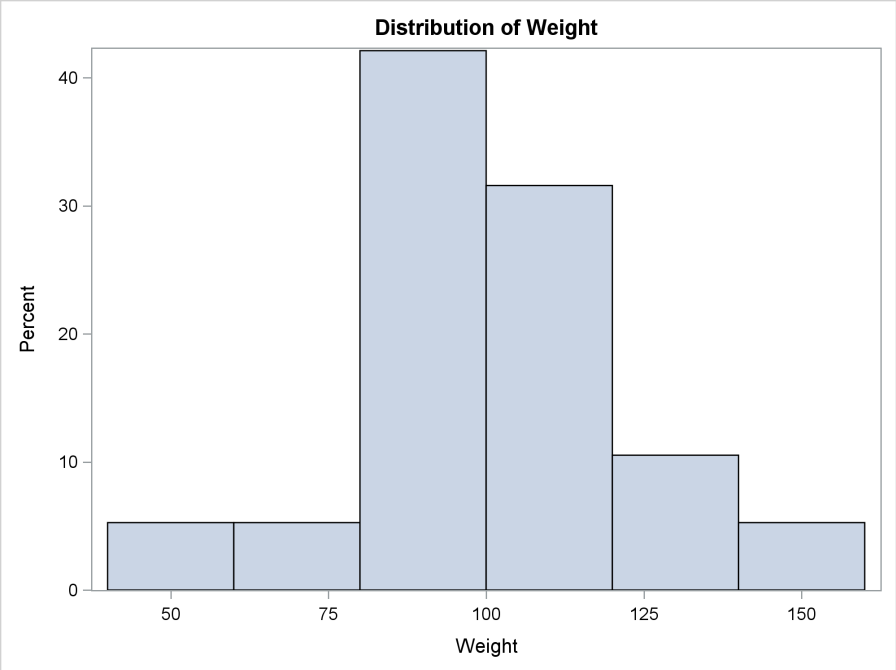


The following steps create a histogram with axes that are not binned:

```
proc template;
   define statgraph classhist;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogram weight / binaxis=false;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classhist;
run;

proc sgplot data=sashelp.class;
   title 'Distribution of Weight';
   histogram weight;
run;
```

The graphs are identical; one is displayed in Figure 1.66. Ticks range from a minimum to a maximum based on an increment; all of them are independent of the bin width.

Figure 1.66 Histogram with Standard Axis Ticks

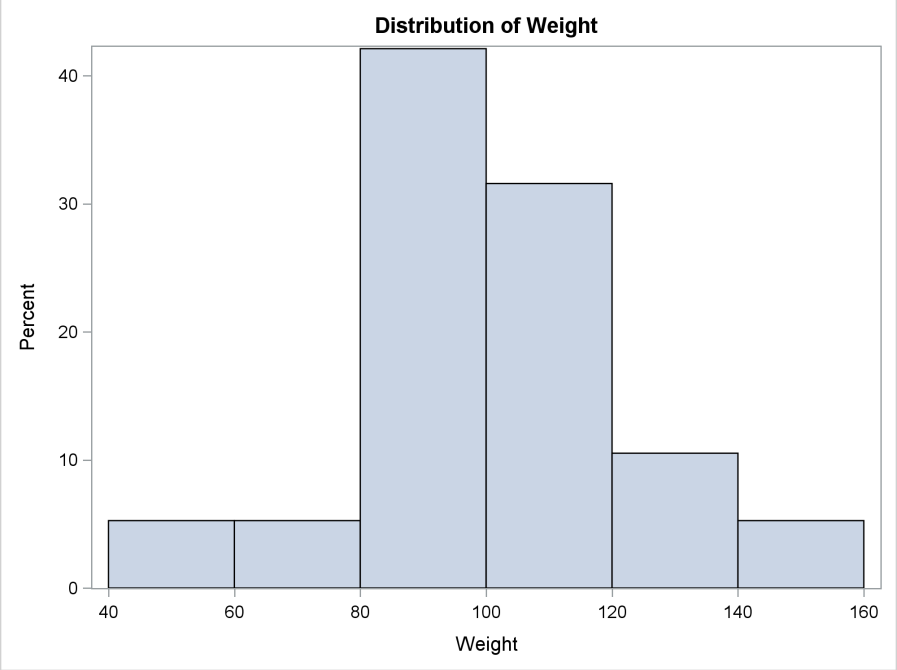


The following steps produce a histogram with bin labels at the end of each bin:

```
proc template;
   define statgraph classhist;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogram weight / endlabels=true;
         endlayout;
      endgraph;
   end;
run;

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

The results are displayed in Figure 1.67.

Figure 1.67 Histogram with Ticks at Bin End



The preceding histogram examples provide ODS Graphics with raw data and with multiple observations in each data range. ODS Graphics does the computations and produces the results. The remainder of this section uses only PROC TEMPLATE and PROC SGRENDER with the HISTOGRAMPARM statement. This statement requires the data to be summarized. You provide a data set with one observation per bar and the information that needs to be displayed. The advantage of using the HISTOGRAMPARM statement is that it gives you precise control over how the data are grouped. You can create this data set from the raw data by using a procedure such as PROC FREQ.

The following steps use the HISTOGRAMPARM statement to produce a histogram:

```
data class;
   set sashelp.class;
   w1 = round(weight - 10, 20) + 10;
   w2 = round(weight, 25);
   label w1 = 'Weight' w2 = 'Weight';
run;

proc freq data=class;
   tables w1 / out=freqs1 noprint;
   tables w2 / out=freqs2 noprint;
run;

proc template;
   define statgraph classhist;
      mvar x;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogramparm x=x y=count;
         endlayout;
      endgraph;
   end;
run;

%let x = w1;
proc sgrender data=freqs1 template=classhist;
run;

%let x = w2;
proc sgrender data=freqs2 template=classhist;
run;
```

The first DATA step creates two new variables, W1 and W2. The variable W1 contains each individual's weight, rounded, with increments between the rounded values of 20. Each rounded value is a multiple of 10 (50, 70, 90, . . .) but not a multiple of 20 (40, 60, 80, . . .). This is accomplished by subtracting 10, rounding to the nearest multiple of 20, and then adding 10 to the rounded values. The variable W1 contains each individual's weight, rounded to the nearest multiple of 25. PROC FREQ tabulates the number of values in each weight group for both variables. The results are stored in SAS data sets in the variable Count. The results are displayed in Figure 1.68 and Figure 1.69.

Figure 1.68 Histogram with Bin Width of 20

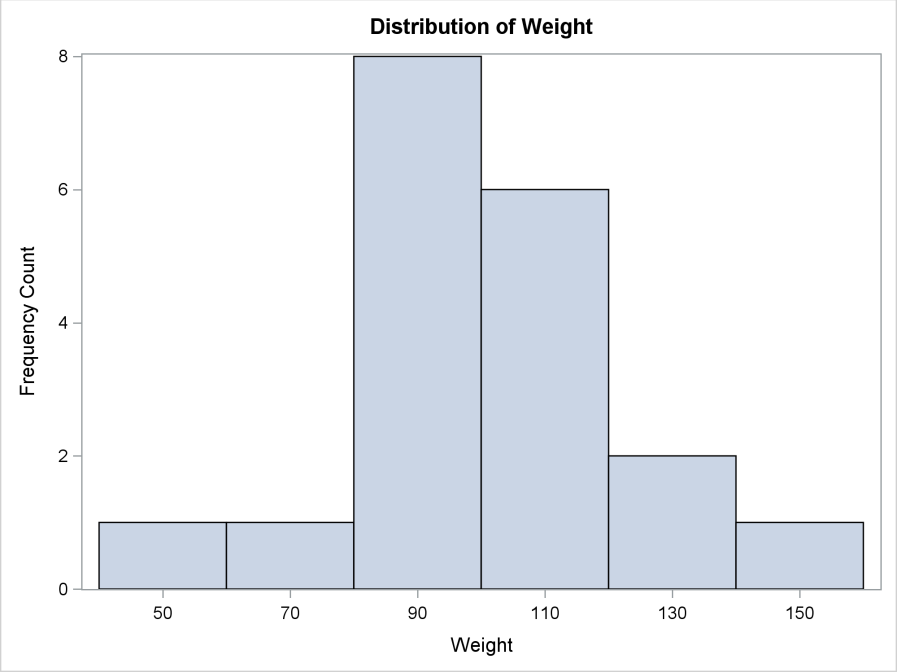
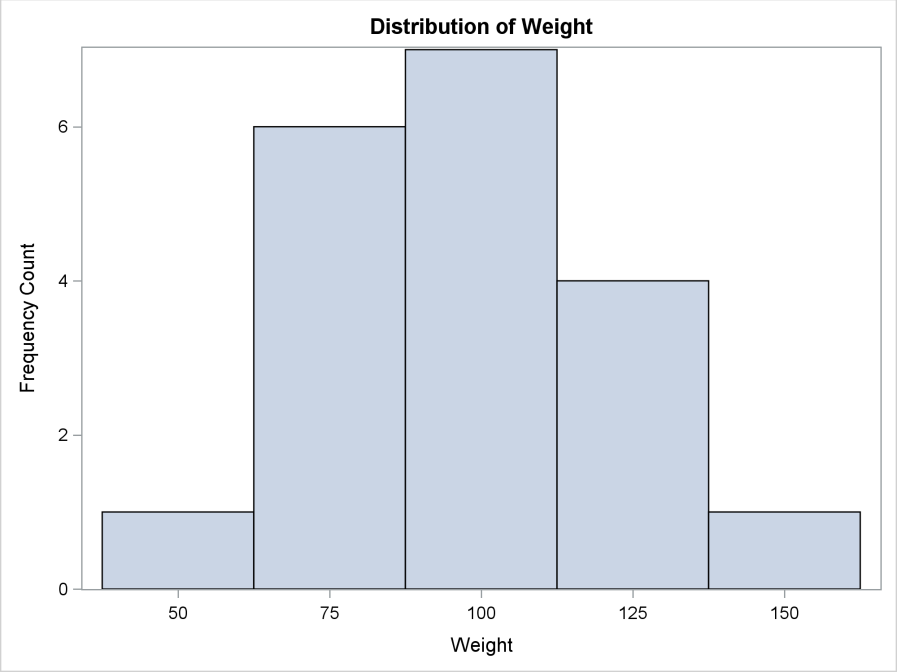


Figure 1.69 Histogram with Bin Width of 25



The HISTOGRAMPARM statement specifies the frequency variable for each interval in the Y= option. The X= option specifies the variable that contains the midpoints of the intervals. In this case, rather than naming a variable in the data set, the specification X=X specifies the macro variable X that appears in the MVAR statement. The MVAR statement specifies that the value of the macro

variable X be retrieved when PROC SGRENDER is run (or more generally when the procedure that uses the template is run, not when PROC TEMPLATE is run). Notice that the macro variables are specified without ampersands. The value of the macro variable X is W1 when the first PROC SGRENDER step is run, so the X= specification becomes in effect X=W1. For the second PROC SGRENDER step, the X= specification becomes in effect X=W2. The MVAR statement is used instead of the direct specification so that the same template can be used to make both plots, one for W1 and the other for W2.

1.9 Density Plot

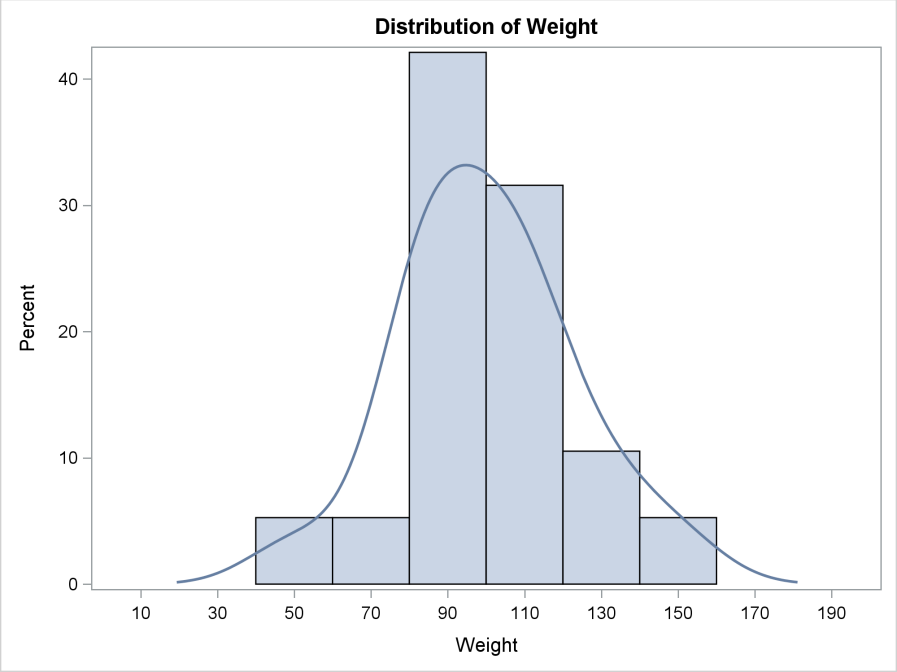
Double-Click for Example Code

A probability density function describes the density of probability at each point in the sample space. The probability that a random variable will fall within a given set is given by the integral of the density over the set. Common probability density functions include normal, chi-square, *F*, and Student's *t*. Kernel density estimation is a nonparametric way to estimate the probability density function of a random variable. The following steps produce a histogram and a kernel density function and show the distribution of weight in the Sashelp.Class data set:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Distribution of Weight';
   histogram weight / showbins;
   density weight / type=kernel;
run;

proc template;
   define statgraph classhistden;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogram weight;
            densityplot weight / kernel();
         endlayout;
      endgraph;
   end;
run;

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

Both procedures have a HISTOGRAM statement. In PROC SGPLOT, the DENSITY statement uses the specification TYPE=KERNEL to create the kernel density. In the GTL, the DENSITYPLOT statement uses the specification KERNEL() to create the kernel density. The graphs are identical; one is displayed in Figure 1.70.

Figure 1.70 Density Plot



The following steps produce a histogram and a normal density function and show the distribution of weight in the Sashelp.Class data set:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Distribution of Weight';
   histogram weight / showbins;
   density weight / type=normal;
run;

proc template;
   define statgraph classhistden2;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogram weight;
            densityplot weight / normal();
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classhistden2;
run;
```

The results of these steps are not shown. They differ from the results of previous steps only in the shape of the density function.

You can use PROC UNIVARIATE or PROC KDE to directly create histograms and perform kernel density estimation when you want more explicit control of the results and when you want a detailed statistical analysis of your data rather than simply a graph.

The following steps perform univariate and bivariate kernel density estimation by using PROC KDE and PROC UNIVARIATE:

```
ods graphics on;

proc kde data=sashelp.class;
   univar weight;
run;

proc kde data=sashelp.class;
   bivar height weight;
run;

proc univariate data=sashelp.class;
   var weight;
   histogram weight / kernel normal;
run;

proc univariate data=sashelp.class;
   class sex;
   var weight;
   histogram weight / kernel normal;
run;
```

The results of these steps are displayed in Figure 1.71, Figure 1.72, Figure 1.73, and Figure 1.74. For more information about kernel density estimation and PROC KDE, see Chapter 66, "The KDE Procedure" (*SAS/STAT User's Guide*). For more information about PROC UNIVARIATE, see Chapter 4, "The UNIVARIATE Procedure" (*Base SAS Procedures Guide: Statistical Procedures*).

Figure 1.71 PROC KDE Univariate Density

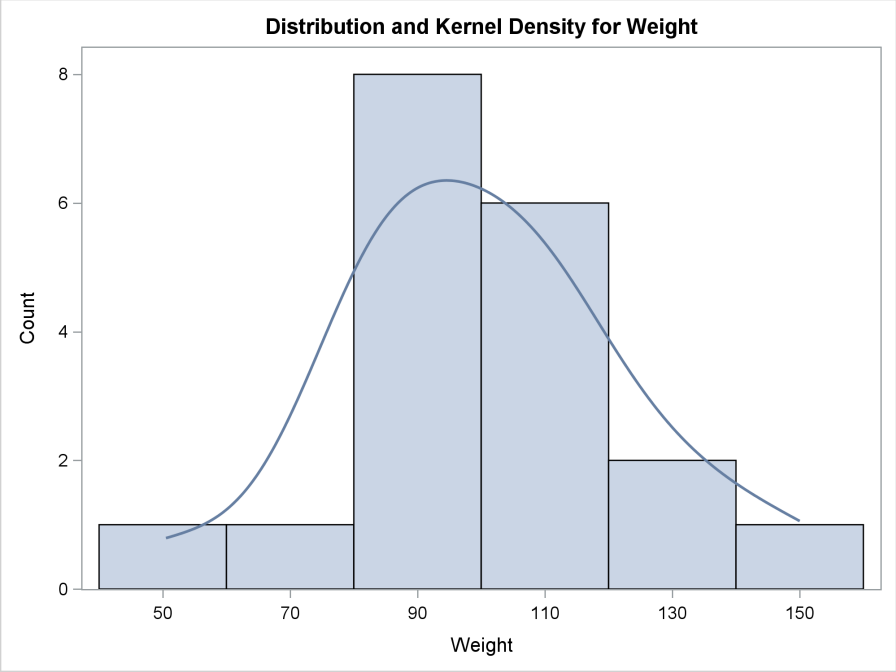


Figure 1.72 PROC KDE Bivariate Density

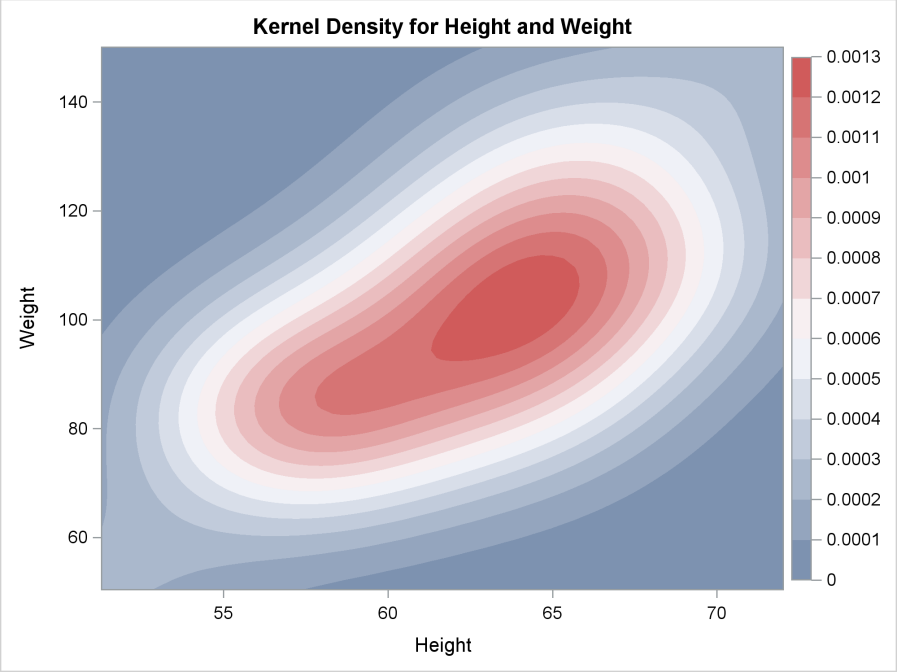


Figure 1.73 PROC UNIVARIATE Histogram

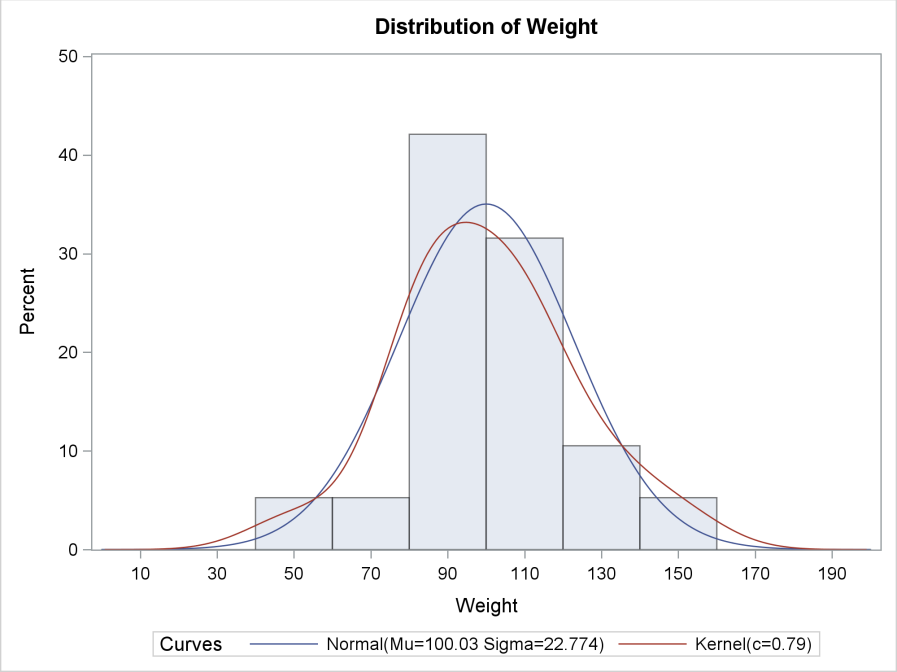
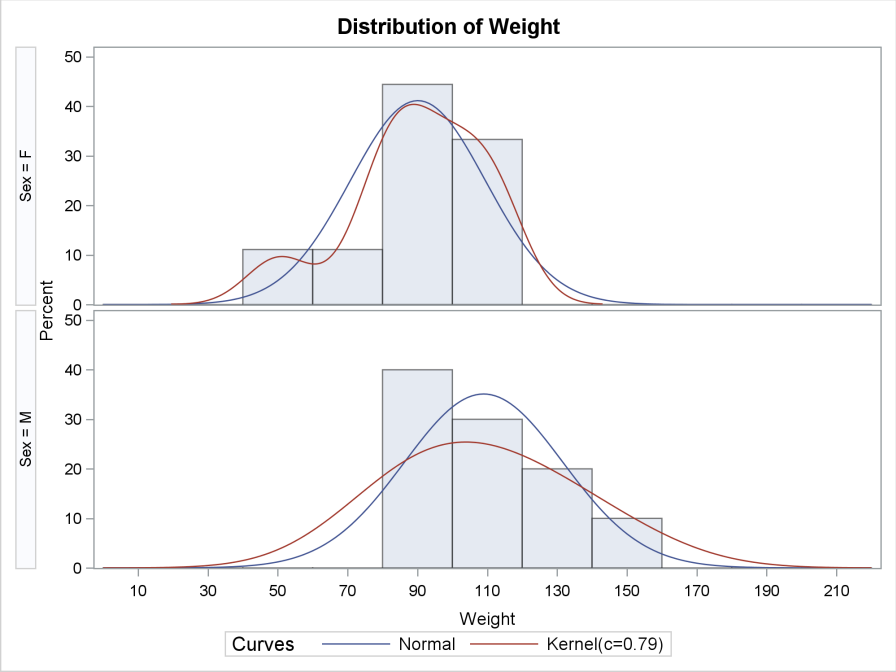


Figure 1.74 PROC UNIVARIATE Grouped Histogram



1.9.1 Fringe Plot

A fringe plot consists of small vertical lines that show the location of each individual data value in the bottom portion of the graph. Fringe plots are most frequently overlaid on other graphs, such as histograms. The following steps create a histogram, a kernel density function, and a fringe plot:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Distribution of Weight';
   histogram weight;
   density weight / type=normal;
   fringe weight;
run;

proc template;
   define statgraph classhistden;
      begingraph;
         entrytitle 'Distribution of Weight';
         layout overlay;
            histogram weight;
            densityplot weight / kernel();
            fringeplot weight;
         endlayout;
      endgraph;
   end;
run;

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

Fringe plots are displayed in Figure 1.75 and Figure 1.76. The PROC SGPLOT results, displayed in Figure 1.75, have an axis that does not correspond to the bars. The PROC SGRENDER results, displayed in Figure 1.76, have ticks at the midpoint of each bar.

Figure 1.75 Histogram, Density, and Fringe Plot, Ordinary Axis

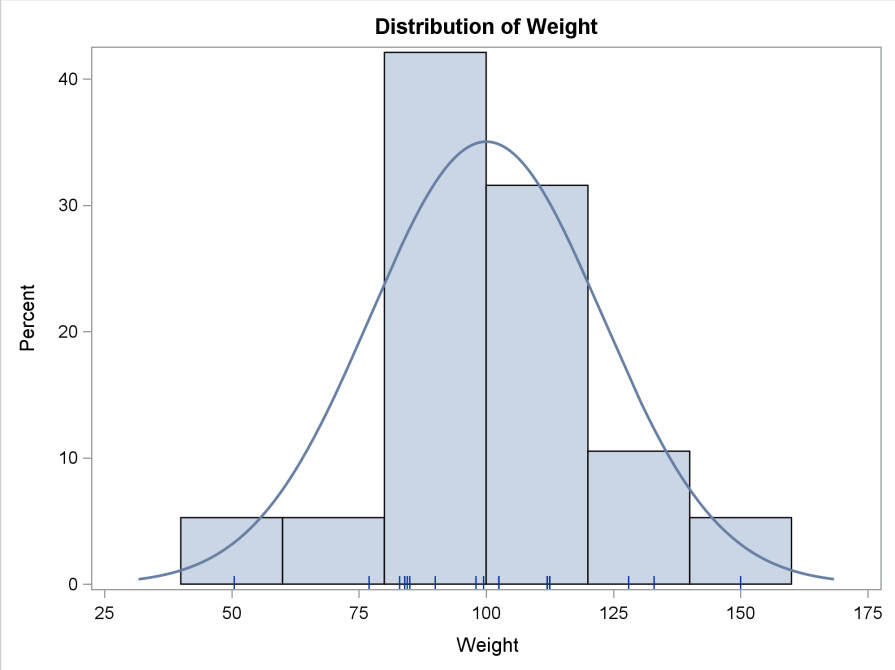
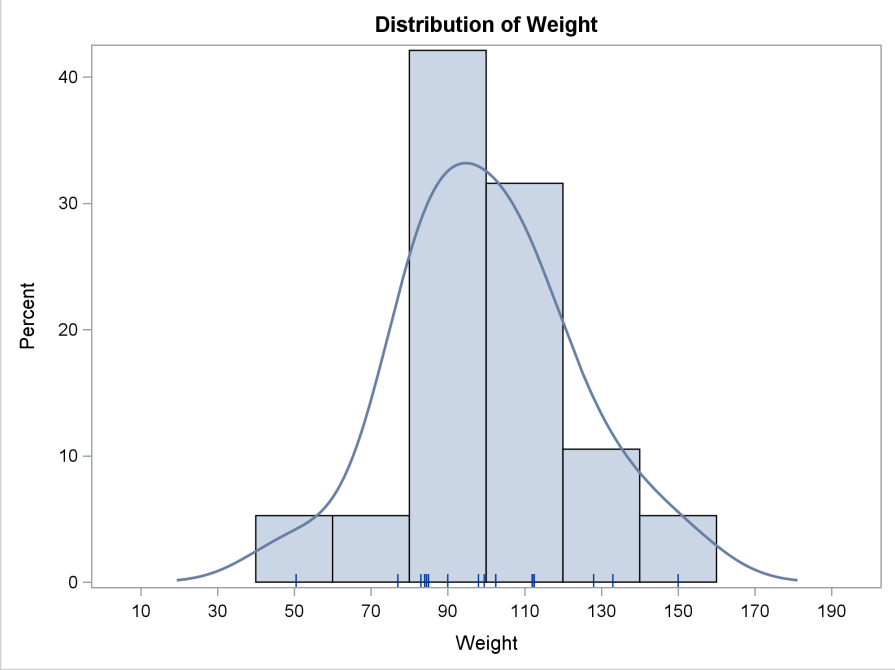


Figure 1.76 Histogram, Density, and Fringe Plot, Binned Axis



1.9.2 Drop Lines

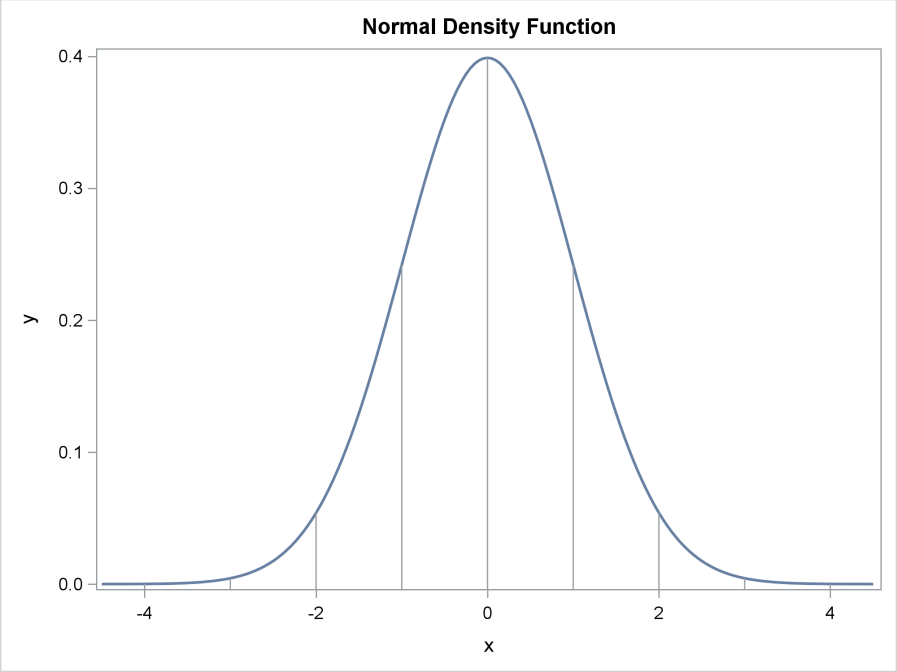
A drop line is a line from a point in the graph to one of the axes. The first part of this section shows how to add a drop line to a plot. The second part of this section shows some creative uses of expressions. The following steps create a normal density function along with vertical drop lines at –4, –3, –2, –1, 0, 1, 2, 3, and 4:

```
data x(drop=c);
   c = sqrt(2 * constant('pi'));
   do x = -4.5 to 4.5 by 0.05;
      y = exp(-0.5 * x ** 2) / c;
      z = ifn(abs(x - round(x)) < 1e-8, y, .);
      output;
   end;
run;

proc template;
   define statgraph normal;
      begingraph;
         entrytitle 'Normal Density Function';
         layout overlay;
            seriesplot x=x y=y / lineattrs=graphfit;
            dropline x=x y=z / dropto=x;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=x template=normal;
run;
```

The DATA step creates the X and Y coordinates of the normal density function, $y = e^{\frac{-1}{2}x^2}/\sqrt{2\pi}$. It also creates a variable z that equals y when x is an integer. The variable z contains the Y-axis coordinates of the drop lines and missing values for all other values of the variable x.

The template contains a SERIESPLOT statement that draws the normal density function by using the **GraphFit** style element. In the HTMLBlue style, this makes a function that is thicker than the default function. The DROPLINE statement draws lines from the X and Y coordinates to the X axis because of the DROPTO=X option. The plot is produced by PROC SGRENDER, and the results are displayed in Figure 1.77.

Figure 1.77 Normal Density with Drop Lines

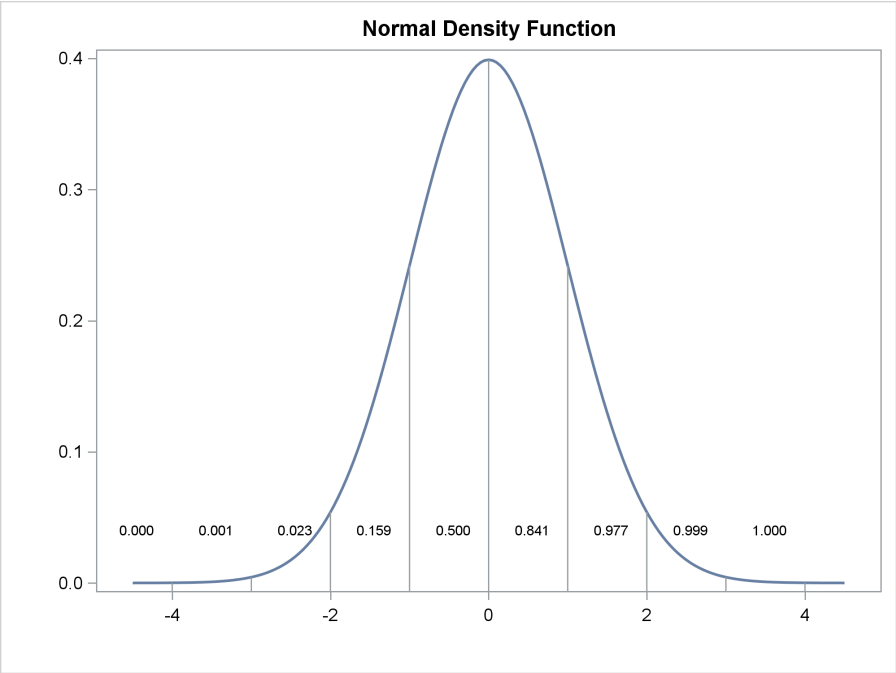


The next example uses a data set with only the X and Y coordinates of the normal density function and uses expressions to manufacture all the other information that it needs. It might be easier to make all this information in a DATA step. However, it is instructive to see that you can do it without a DATA step. When you modify a template that SAS provides to use with a SAS analytic procedure, you cannot add new columns to the data object. You can work only with the information that the procedure provides. This example creates a normal density function with drop lines as in the preceding example. However, it also provides a label for each line that displays the proportion of the area under the curve that falls below the drop line, and it suppresses the axis labels. The following steps create the plot that is displayed in Figure 1.78:

```
data x(drop=c);
   c = sqrt(2 * constant('pi'));
   do x = -4.5 to 4.5 by 0.05;
      y = exp(-0.5 * x ** 2) / c;
      output;
   end;
run;

proc template;
   define statgraph normal;
      begingraph;
         entrytitle 'Normal Density Function';
         layout overlay;
            seriesplot x=x y=y / lineattrs=graphfit;
            scatterplot x=eval(ifn(abs(x - round(x)) < 1e-8, x - 0.45, .))
                        y=eval(0 * x + 0.04) /
                        markercharacter=eval(put(probnorm(x), 5.3));
            dropline x=x y=eval(ifn(abs(x - round(x)) < 1e-8, y, .)) / dropto=x;
         endlayout;
```

```
      endgraph;
   end;
run;

proc sgrender data=x template=normal;
   label x = '00'x y = '00'x;
run;
```

Figure 1.78 Normal Density with Area



The DATA step creates the variables x and y with the normal density function. The SERIESPLOT statement displays this function and matches the SERIESPLOT statement in the first part of this section. The DROPLINE statement now specifies an expression of the form Y=EVAL(. . .) rather than a column such as the Y=Z specification shown previously. This expression is as follows:

```
eval(ifn(abs(x - round(x)) < 1e-8, x - 0.45, .))
```

When the X value is within rounding error of an integer, drop lines are drawn. Otherwise, no lines are drawn.

The template contains a new statement, a SCATTERPLOT statement, that displays the area under the curve preceding each drop line. The SCATTERPLOT statement is as follows:

```
scatterplot x=eval(ifn(abs(x - round(x)) < 1e-8, x - 0.45, .))
            y=eval(0 * x + 0.04) /
            markercharacter=eval(put(probnorm(x), 5.3));
```

The goal is to place areas under the curve to the left of each line and positioned slightly above the X axis, where they will not conflict with the density function. The expression results in x – 0.45 when the X value is within rounding error of an integer and missing otherwise. This expression provides the X coordinates for the scatter plot that displays the area under the curve. The Y-axis coordinates are **y=eval(0 * x + 0.04)**, which produce constant Y coordinates of 0.04. Simpler expressions such as Y=0.04 and Y=EVAL(0.04) are not accepted, so you can specify a constant by adding a constant to 0 times one of the variables. The MARKERCHARACTER= option specifies a label that is displayed in place of a symbol. This variable or expression can be character or numeric, so you could specify **MarkerCharacter=Eval(ProbNorm(x))**. However, that would produce more decimal places than you need. The PUT function specifies a 5.3 format.

The graph in Figure 1.78 has no labels on its axes. You accomplish this by specifying the statement **label x = '00'x y = '00'x** in the PROC SGRENDER step. This sets the label for the two variables to null, specified as a hexadecimal constant. Alternatively, you could specify the options **xAxisOpts=(Display=(Line Ticks TickValues)) yAxisOpts=(Display=(Line Ticks TickValues))** in the LAYOUT OVERLAY statement.

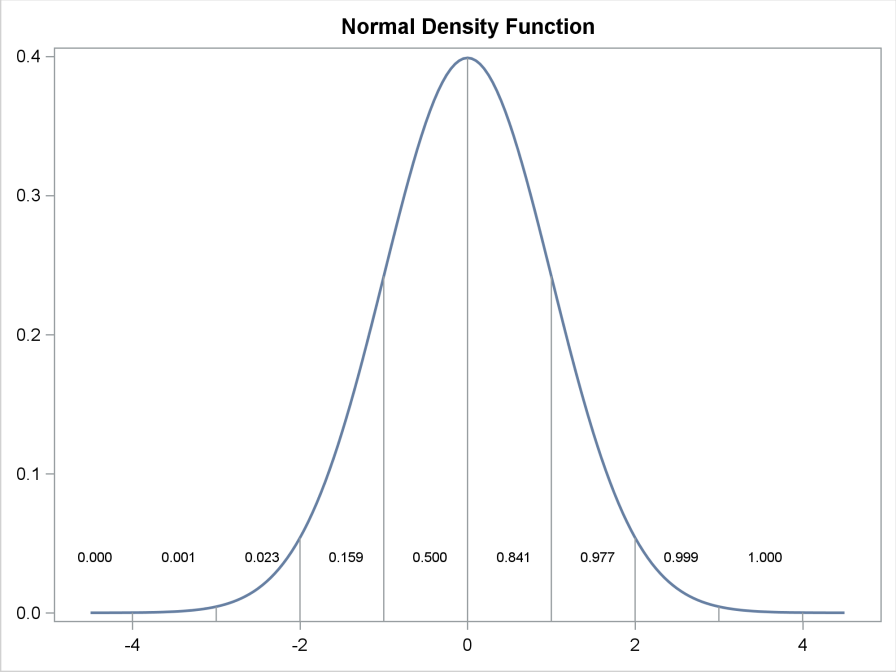
There are other things that you can do to this graph. Notice that neither Figure 1.77 nor Figure 1.78 displays precisely the correct geometry. That is, both have a Y axis in which $y = 0$ does not correspond to the X axis. To correct this, you could specify **yAxisOpts=(OffsetMin=0)** in the LAYOUT OVERLAY statement. Although this fixes the geometry problem, it makes the drop lines at –4, –3 , 3, and 4 difficult to see.

The following steps use PROC SGPLOT to show the normal density function:

```
data x;
   c = sqrt(2 * constant('pi'));
   do x  = -4.5 to 4.5 by 0.05;
      y  = exp(-0.5 * x ** 2) / c;
      z  = ifn(abs(x - round(x)) < 1e-8, x, .);
      l  = ifc(abs(x - round(x)) < 1e-8, put(probnorm(x), 5.3), ' ');
      x2 = z - 0.45;
      y2 = 0.04;
      output;
   end;
run;

proc sgplot data=x noautolegend;
   title 'Normal Density Function';
   series   y=y  x=x  / lineattrs=graphfit;
   dropline y=y  x=z  / dropto=x;
   scatter  y=y2 x=x2 / markerchar=l;
   xaxis display=(nolabel);
   yaxis display=(nolabel);
run;
```

Because PROC SGPLOT does not support function evaluation, all the variables are created in a DATA step. The results are displayed in Figure 1.78.

Figure 1.79 Normal Density with Area



1.10 Box Plot

Double-Click for Example Code

A box plot displays the distribution of groups of data through their five-number summaries (the minimum, 25th percentile, median, 75th percentile, and maximum). Optionally, means and outliers can also be displayed. The following steps produce vertical box plots of weight by sex:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Distribution of Weight by Sex';
   vbox weight / category=sex;
run;

proc template;
   define statgraph classbox;
      begingraph;
         entrytitle 'Distribution of Weight by Sex';
         layout overlay;
            boxplot y=weight x=sex;
         endlayout;
      endgraph;
   end;
run;

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

In PROC SGPLOT, you request a vertical box plot in the VBOX statement by specifying a quantitative variable before the slash and specifying the categorical variable in the CATEGORY= option. In

the GTL, you request a vertical box plot in the BOXPLOT statement by naming the categorical variable in the X= option and naming the quantitative variable in the Y= option. The results, which are displayed in Figure 1.80 and Figure 1.81, are identical except for the order of the categories. In the GTL, the categories appear in the same order as they do in the input data set. In PROC SGPLOT, the categories appear in sorted order.

Figure 1.80 Box Plots with PROC SGPLOT

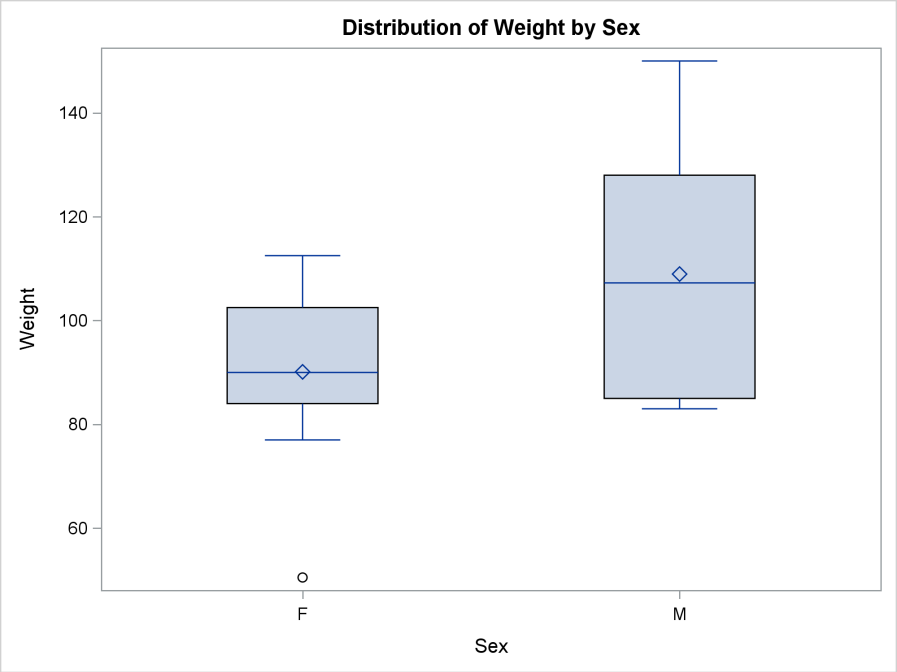
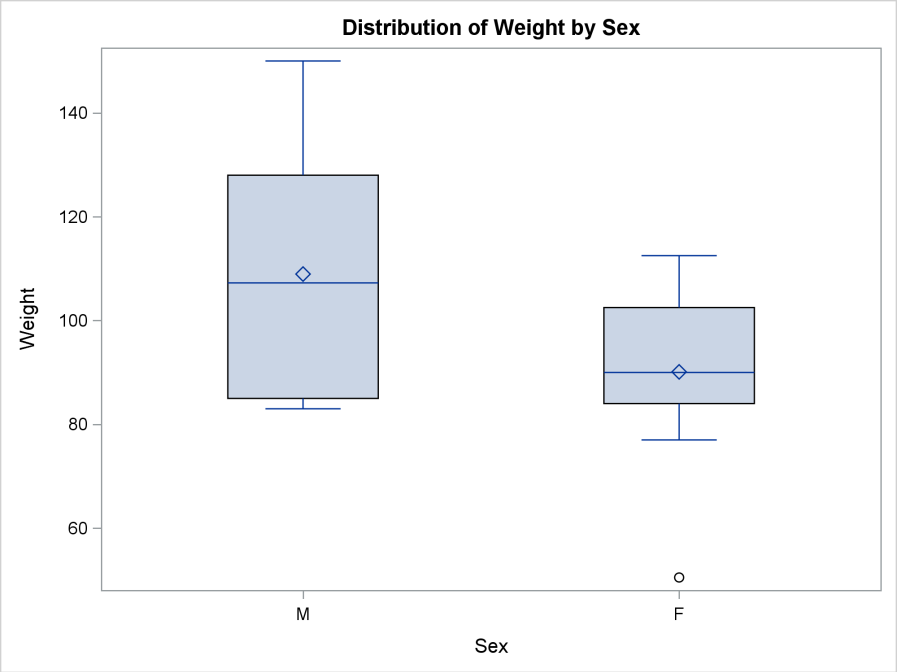


Figure 1.81 Box Plots with Custom Template

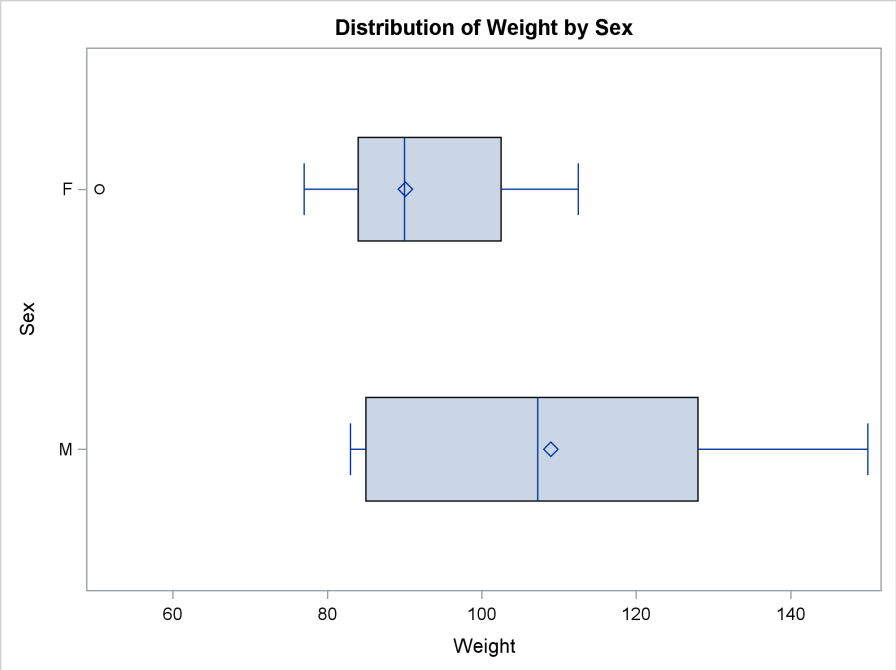


The following steps produce horizontal box plots of weight by sex:

```
proc sgplot data=sashelp.class noautolegend;
   title 'Distribution of Weight by Sex';
   hbox weight / category=sex;
run;

proc template;
   define statgraph classbox;
      begingraph;
         entrytitle 'Distribution of Weight by Sex';
         layout overlay;
            boxplot y=weight x=sex / orient=horizontal;
         endlayout;
      endgraph;
   end;
run;

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

The options here are an obvious variation on the options in the previous steps. The graphs are identical; one is displayed in Figure 1.82.

Figure 1.82 Horizontal Box Plots



You can also use PROC BOXPLOT to produce box plots. The following steps produce vertical and horizontal box plots:

```
proc sort data=sashelp.class out=class;
   by sex;
run;

ods graphics on;

proc boxplot data=class;
   plot weight*sex;
run;

proc boxplot data=class;
   plot weight*sex / horizontal;
run;
```

The results are displayed in Figure 1.83 and Figure 1.84. For more information about PROC BOXPLOT, see Chapter 28, "The BOXPLOT Procedure" (*SAS/STAT User's Guide*).

Figure 1.83 Vertical Box Plot with PROC BOXPLOT

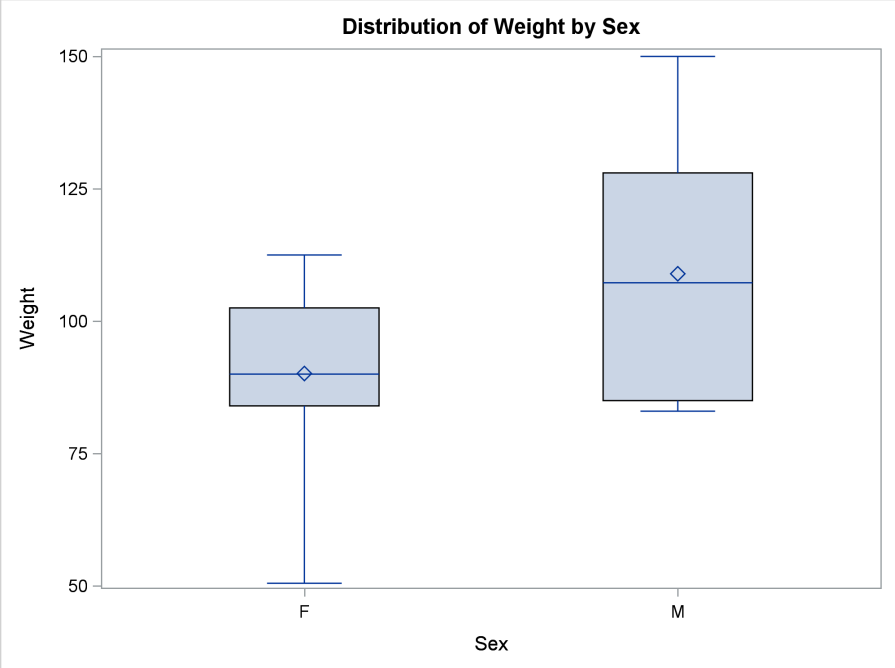
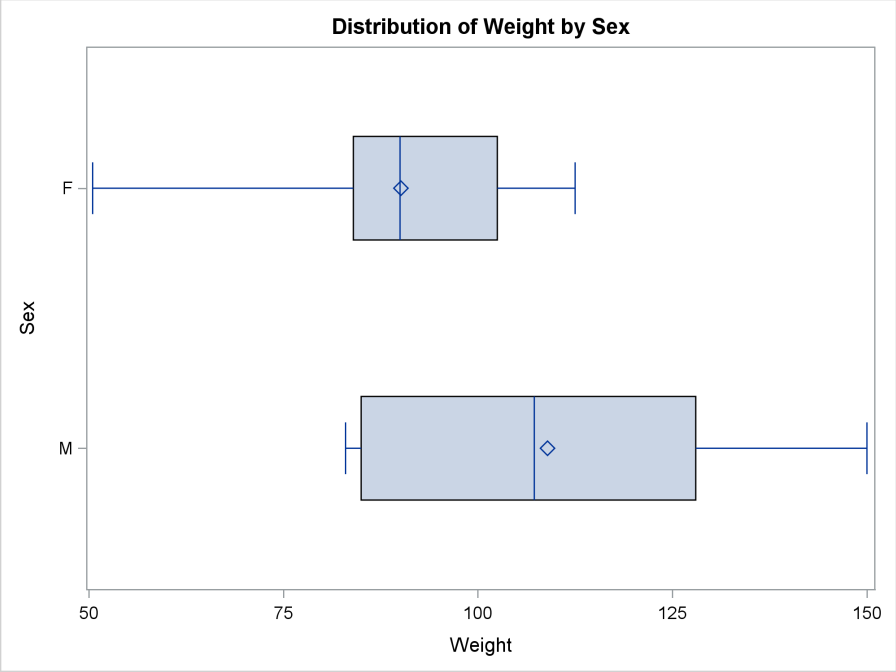


Figure 1.84 Horizontal Box Plot with PROC BOXPLOT



The preceding box plot examples provide ODS Graphics or PROC BOXPLOT with raw data and with multiple observations in each group. ODS Graphics or PROC BOXPLOT does the computations to get the statistics and produce the results. The remainder of this section uses only PROC TEMPLATE and PROC SGRENDER with the BOXPLOTPARM statement. This statement requires summarized data, not raw data. You provide a data set that has one observation per statistic per group and the information that needs to be displayed. The advantage of using the BOXPLOTPARM statement is that it gives you precise control over the calculations. You can create this data set from the raw data by using a procedure such as PROC UNIVARIATE.

The following steps create the input data set that the BOXPLOTPARM statement requires:

```
proc univariate data=sashelp.class;
   var weight;
   class sex;
   ods output quantiles=q;
run;

data q2;
   set q;
   quantile = scan(quantile, 2, ' ');
   if quantile ne ' ';
run;

proc print;
run;
```

PROC UNIVARIATE is run to get the quantiles (minimum, 25th percentile, median, 75th percentile, and maximum) for the variable Weight for each level of the classification variable Sex. The ODS OUTPUT statement creates an output data set to contain the quantile information. This data set is not in the form that the BOXPLOTPARM statement expects, so a DATA step puts the data into the proper form. In the PROC UNIVARIATE table and ODS output data set, there is a variable Estimate with the quantiles and a variable Quantile with the quantile labels, which are as follows: 100% Max, 99%, 95%, 90%, 75% Q3, 50% Median, 25% Q1, 10%, 5%, 1%, 0% Min. Only the Max, Q3, Median, Q1, and Min statistics are needed for a basic box plot, and only those labels are expected (that is, without the "*n*%" part). The DATA step statement `quantile = scan(quantile, 2, ' ')` extracts only the second value, where values are delimited by blanks. This creates the labels that the BOXPLOTPARM statement expects. Uppercase, lowercase, and mixed-case values work equally well. The statement `if quantile ne ' '` discards observations with blank labels, leaving the values labeled by Max, Q3, Median, Q1, and Min. These values are displayed in Figure 1.85. Additional statistics could include MEAN (data mean), OUTLIER (observation outside the lower and upper fences), FAROUTLIER (observation outside the lower and upper far fences), and STD (standard deviation).

Figure 1.85 Input for a Parameterized Box Plot

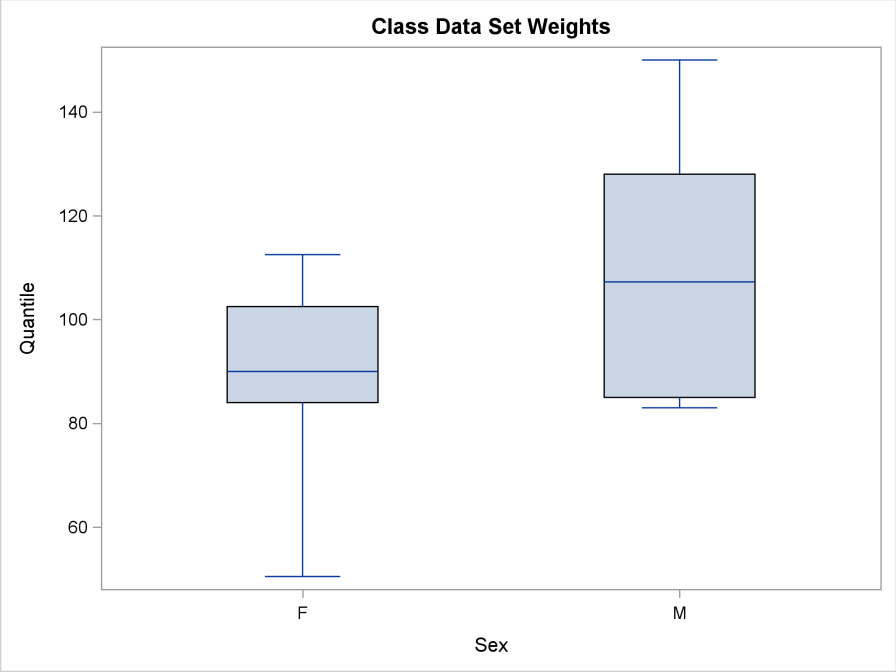
Distribution of Weight by Sex

Obs	VarName	Sex	Quantile	Estimate
1	Weight	F	Max	112.50
2	Weight	F	Q3	102.50
3	Weight	F	Median	90.00
4	Weight	F	Q1	84.00
5	Weight	F	Min	50.50
6	Weight	M	Max	150.00
7	Weight	M	Q3	128.00
8	Weight	M	Median	107.25
9	Weight	M	Q1	85.00
10	Weight	M	Min	83.00

The following steps produce the box plot of weight by sex that is displayed in Figure 1.86:

```
proc template;
   define statgraph boxplotparm1;
      begingraph;
         entrytitle "Class Data Set Weights";
         layout overlay;
            boxplotparm y=estimate x=sex stat=quantile;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=q2 template=boxplotparm1;
run;
```

Figure 1.86 Parameterized Vertical Box Plot with Error Bars

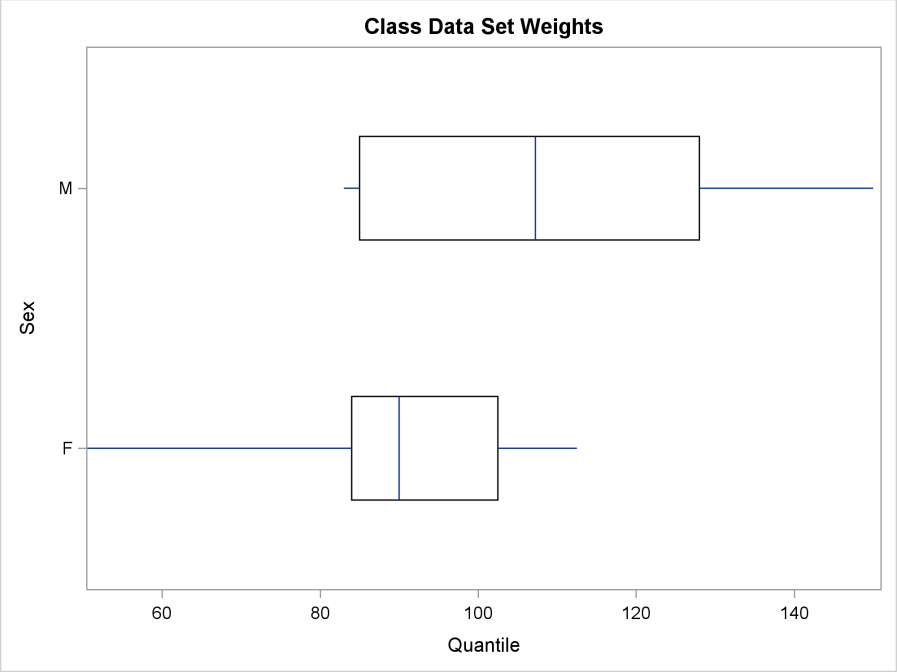


The BOXPLOTPARM statement specifies the quantile and other statistics in the Y= option, the groups in the X= option, and the labels for the statistics in the STAT= option. The following steps create a horizontal box plot without fill and produce the display in Figure 1.87:

```
proc template;
   define statgraph boxplotparm2;
      begingraph;
         entrytitle "Class Data Set Weights";
         layout overlay;
            boxplotparm y=estimate x=sex stat=quantile /
                        orient=horizontal display=(median);
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=q2 template=boxplotparm2;
run;
```

The option ORIENT=HORIZONTAL creates the horizontal box plot. The default value for the DISPLAY= option is DISPLAY=STANDARD, which is the same as DISPLAY=(CAPS FILL MEAN MEDIAN OUTLIERS). The option DISPLAY=(MEDIAN) displays the median in the box without filling the boxes, without the caps on the lines, and without the means or outliers (if provided).

Figure 1.87 Parameterized Horizontal Box Plot with Error Bars



1.11 Series Plot

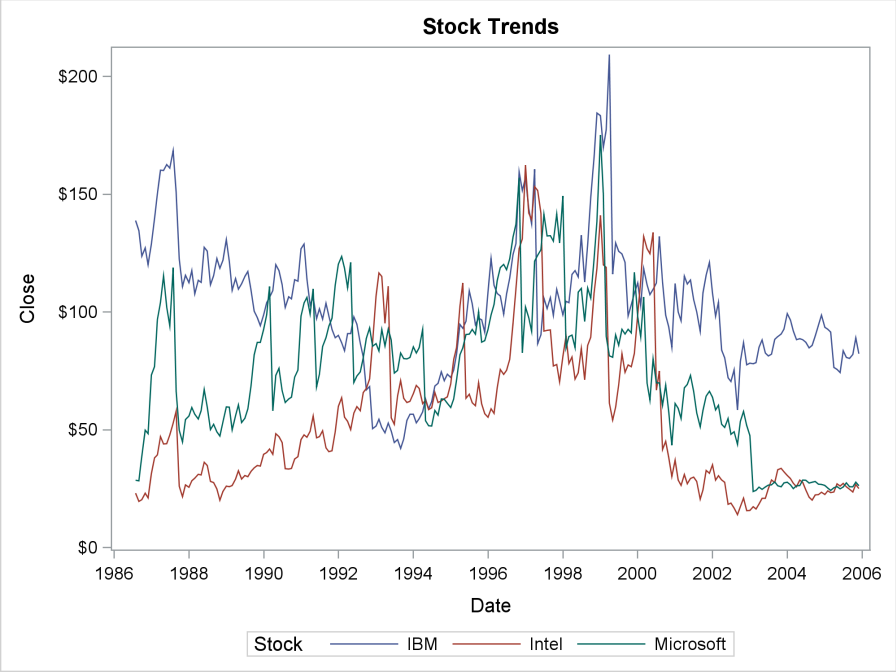
Double-Click for Example Code

A series plot is a graphical display of two quantitative variables in which consecutive points are connected by straight line segments. The points are usually not displayed. When the points are close together, the results can appear to be a smooth curve. Series plots show regression fit functions and trends over time. The following steps produce a series plot for each of three stocks:

```
proc sgplot data=sashelp.stocks;
   title 'Stock Trends';
   series x=date y=close / group=stock;
run;

proc template;
   define statgraph seriesplot;
      begingraph;
         entrytitle 'Stock Trends';
         layout overlay;
           seriesplot x=date y=close / group=stock name='stocks';
           discretelegend 'stocks' / title='Stock';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.stocks template=seriesplot;
run;
```

The SERIESPLOT statement in the GTL and the SERIES statement in PROC SGPLOT connect the data points for the X=Date variable on the X axis and the Y=Close variable on the Y axis for each value in the categorical GROUP=Stock variable. The SERIESPLOT statement is named so that a legend, titled `'Stock'`, is produced. In PROC SGPLOT, the legend is automatically produced. The graphs are identical; one is displayed in Figure 1.88.

Figure 1.88 Series Plots



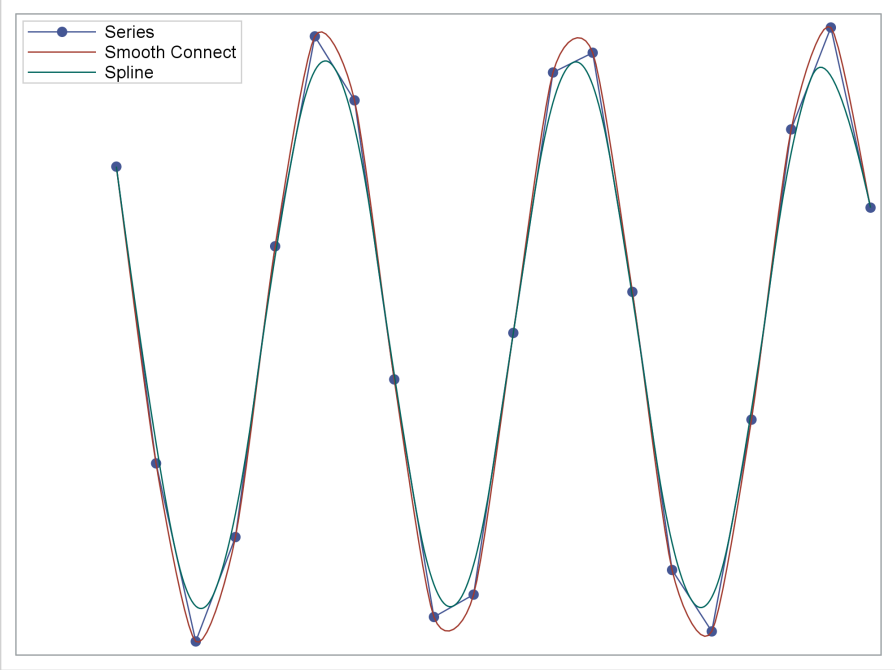
You can connect the points with a smooth curve. Or you can request a curve that is not required to touch every point. The following steps illustrate by using some artificial data:

```
data x;
   do x = 1 to 20;
      y = cos(x);
      output;
   end;
run;

title;
proc sgplot data=x;
   series y=y x=x / legendlabel='Series' markers
                    markerattrs=(symbol=circlefilled);
   series y=y x=x / legendlabel='Smooth Connect' smoothconnect;
   spline y=y x=x / legendlabel='Spline';
   keylegend / location=inside across=1 position=topleft;
   xaxis display=none offsetmin=0.07;
   yaxis display=none;
run;
```

```
proc template;
   define statgraph connect;
      begingraph;
         layout overlay / xaxisopts=(display=none offsetmin=0.07)
                          yaxisopts=(display=none) cycleattrs=true;
            seriesplot x=x y=y / display=(markers) legendlabel="Series"
                                 name="se" markerattrs=(symbol=CircleFilled);
            seriesplot x=x y=y / smoothconnect=true legendlabel="Smooth Connect"
                                 name="sm";
            seriesplot x=x y=y / splinetype=QuadraticBezier legendlabel="Spline"
                                 name="sp";
            discretelegend "se" "sm" "sp" / location=inside across=1
                                            halign=left valign=top;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=x template=connect;
run;
```

The graphs are identical; one is displayed in Figure 1.89.

Figure 1.89 Series Plots



The SERIES and SERIESPLOT statements with no options connect the dots. The SERIES and SERIESPLOT statements with the SMOOTHCONNECT option connect the dots with a smooth curve. The SPLINE statement and SERIESPLOT statement with the SPLINETYPE=QUADRATICBEZIER option supply a smoother curve that does not connect all dots.

1.12 Dot Plot

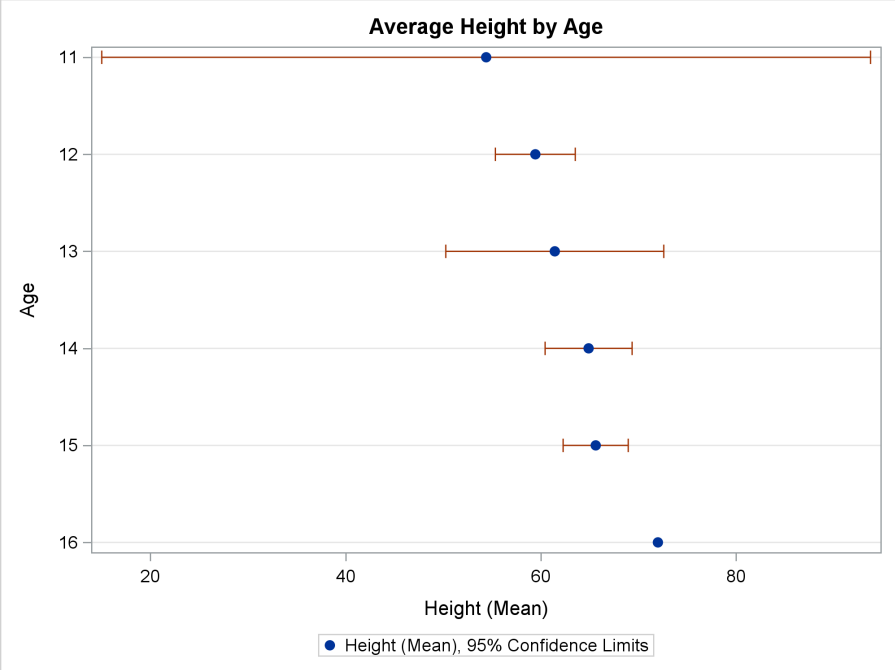
Double-Click for Example Code

A dot plot displays summary statistics computed from a quantitative variable for each level of a classification or group variable. The following steps show two ways to produce dot plots:

```
proc means data=sashelp.class noprint nway;
   var height;
   class age;
   output out=class mean=height lclm=lower uclm=upper;
   label height = 'Height(Mean)';
run;

proc sgplot data=sashelp.class;
   title 'Average Height by Age';
   dot age / response=height stat=mean limits=both;
run;

proc template;
   define statgraph dotplot;
      begingraph;
         entrytitle 'Average Height by Age';
         layout overlay / yaxisopts=(type=discrete griddisplay=on
                                     reverse=true);
            scatterplot y=age x=height / xerrorlower=lower xerrorupper=upper
                        markerattrs=(symbol=circlefilled) name='dot'
                        legendlabel='Height(Mean), 95% Confidence Limits';
            discretelegend 'dot';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=class template=dotplot;
run;
```

In PROC SGPLOT, the DOT statement makes a dot plot for the RESPONSE=Height variable for the levels of the variable Age, which is specified before the slash. The mean height is shown for each age along with confidence limits in both directions. The graphs are almost identical; one is displayed in Figure 1.90.

Figure 1.90 Dot Plot



The GTL does not have a dot plot statement; however, the SCATTERPLOT statement has all the needed options after the data have been processed with PROC MEANS. PROC MEANS creates an output data set that contains for each age group the mean height and two other variables: Lower, which contains the lower end of the confidence interval, and Upper, which contains the upper end of the confidence interval. Additionally, a label is provided for the mean height; this is the same label that PROC SGPLOT automatically generates. The YAXISOPTS=(TYPE=DISCRETE) option in the LAYOUT OVERLAY statement creates a discrete Y axis; the default axis is linear. The YAXISOPTS=(REVERSE=TRUE) option reverses the Y axis so that the values decrease rather than increase as you move vertically up the Y axis.

The SCATTERPLOT statement displays the X=Height response variable for the levels of the Y=Age categorical variable. Error bars are drawn in the range from the values in the XERRORLOWER=Lower variable to the values in the XERRORUPPER=Upper variable. The means are displayed as solid filled circles because of the option MARKERATTRS=(SYMBOL=CIRCLEFILLED). The statement is named so that a legend can be produced. A legend label is provided that matches the legend label that PROC SGPLOT automatically displays. The graphs are almost identical; one is displayed in Figure 1.90.

1.13 Polygon

Double-Click for Example Code

A polygon is a shape with three or more sides. You can use the POLYGON statement in PROC SGPLOT to draw polygons. The following steps draw a square, pentagon, hexagon, and octagon:

```
title;
data x(drop=pi tau);
   pi = constant('pi');
   id = 1;
   type = 'Square  ';
   do tau = 0 to 2 * pi by 2 * pi / 4;
      x = -1 + cos(tau - pi / 4);
      y =  1 + sin(tau - pi / 4);
      output;
   end;
   id = 2;
   type = 'Pentagon';
   do tau = 0 to 2 * pi by 2 * pi / 5;
      x =  1 + cos(tau + pi / 10);
      y =  1 + sin(tau + pi / 10);
      output;
   end;
   id = 3;
   type = 'Hexagon';
   do tau = 0 to 2 * pi by 2 * pi / 6;
      x = -1 + cos(tau);
      y = -1 + sin(tau);
      output;
   end;
   id = 4;
   type = 'Octagon';
   do tau = 0 to 2 * pi by 2 * pi / 8;
      x =  1 + cos(tau - pi / 8);
      y = -1 + sin(tau - pi / 8);
      output;
   end;
run;

proc print noobs label;
   id type id;
   by notsorted type id;
   label type = '00'x;
run;

ods graphics on / width=4.8in height=4.8in subpixel=on;
proc sgplot;
   polygon x=x y=y id=id;
   xaxis display=none;
   yaxis display=none;
run;
```

The vertices of the polygons are equally spaced points on a circle. The data set contains an X variable, a Y variable, and two ID variables (ID and Type). You need only one ID variable, and either can be used. For each polygon with n sides, there are $n + 1$ points. The first and final points are identical for each polygon. The POLYGON statement starts by drawing from the first point to the second point, then to the third, and so on until the final point is connected. The input data set is displayed in Figure 1.91. The four shapes are displayed in Figure 1.92.

Figure 1.91 Polygon Coordinates

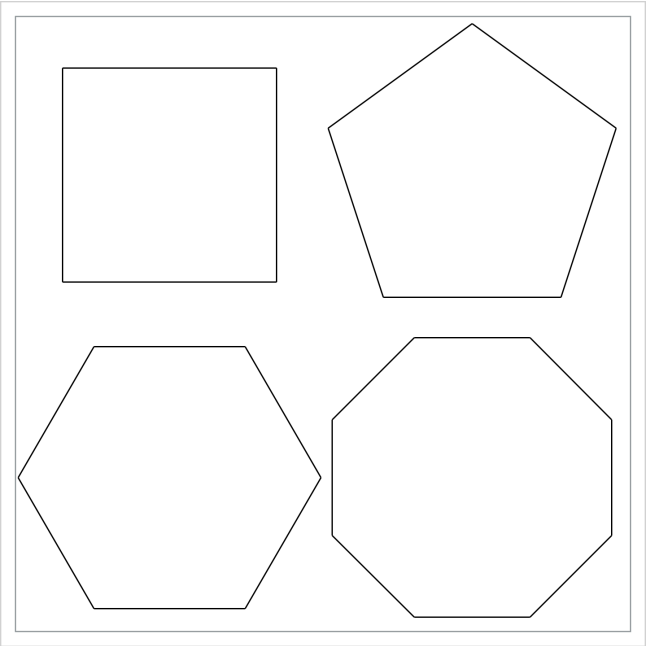
	id	x	y
Square	**1**	-0.29289	0.29289
		-0.29289	1.70711
		-1.70711	1.70711
		-1.70711	0.29289
		-0.29289	0.29289

	id	x	y
Pentagon	**2**	1.95106	1.30902
		1.00000	2.00000
		0.04894	1.30902
		0.41221	0.19098
		1.58779	0.19098
		1.95106	1.30902

	id	x	y
Hexagon	**3**	0.00000	-1.00000
		-0.50000	-0.13397
		-1.50000	-0.13397
		-2.00000	-1.00000
		-1.50000	-1.86603
		-0.50000	-1.86603
		0.00000	-1.00000

	id	x	y
Octagon	**4**	1.92388	-1.38268
		1.92388	-0.61732
		1.38268	-0.07612
		0.61732	-0.07612
		0.07612	-0.61732
		0.07612	-1.38268
		0.61732	-1.92388
		1.38268	-1.92388
		1.92388	-1.38268

Figure 1.92 Polygons

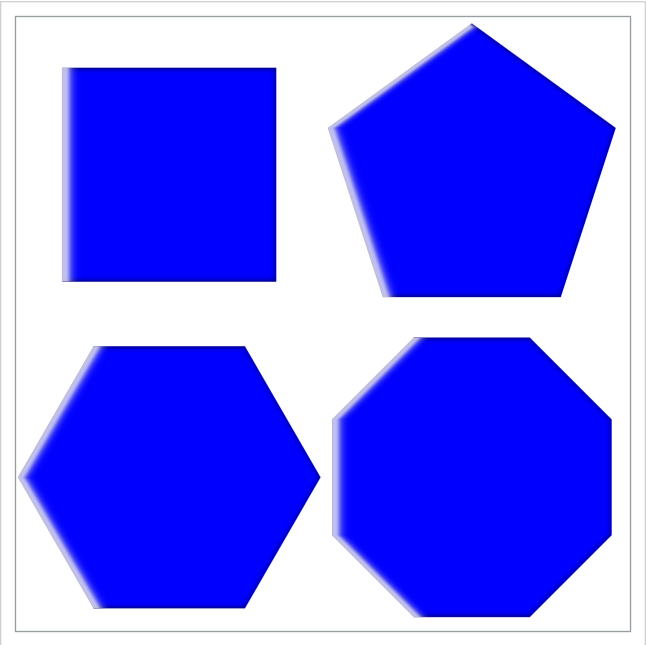


There are options for creating filled shapes and controlling the data skin (which provides three-dimensional effects):

```
proc sgplot;
   polygon x=x y=y id=id / fill fillattrs=(color=blue) dataskin=gloss;
   xaxis display=none;
   yaxis display=none;
run;
```

The results are displayed in Figure 1.93.

Figure 1.93 Filled Polygons



You can use the GTL to draw polygons as follows:

```
proc template;
   define statgraph polygon;
      begingraph;
         layout overlay / xaxisopts=(display=none) yaxisopts=(display=none);
            polygonplot X=x Y=y ID=id;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=x template=polygon;
run;
```

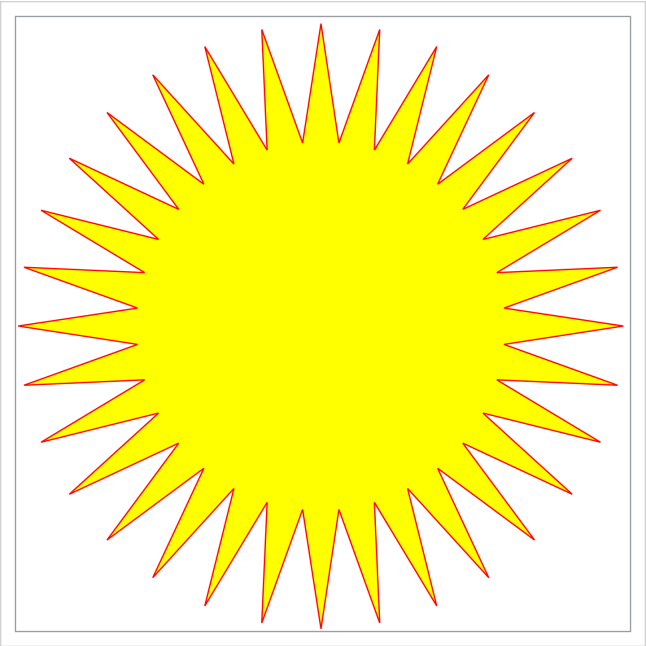
The results are identical to the graph displayed in Figure 1.92.

You are not required to specify a final point that matches the first point. The last segment is provided automatically. Furthermore, polygons are not required to be regular (equal angles and side lengths). The angles in the 64-sided polygon that is generated by the following steps are both acute and reflex:

```
data x(keep=id x y);
   pi  = constant('pi');
   id  = 1;
   inc = 2 * pi / 32;
   do tau = 0 to 2 * pi - 0.5 * inc by inc;
      x  = cos(tau);
      y  = sin(tau);
      output;
      x2 = cos(tau + inc);
      y2 = sin(tau + inc);
      m  = -(x2 - x) / (y2 - y);
      t1 = mean(x, x2);
      t2 = mean(y, y2);
      x  = t1 + ifn(t1 gt 0, -1, 1) * sqrt(0.15 / (1 + m * m));
      y  = m * (x - t1) + t2;
      output;
   end;
output;
run;

proc sgplot;
   polygon x=x y=y id=id / fill    fillattrs=(color=yellow)
                           outline lineattrs=(color=red);
   xaxis display=none;
   yaxis display=none;
run;

ods graphics on / reset=all subpixel=on;
```

The results are displayed in Figure 1.94.

Figure 1.94 Fun in the Sun



1.14 Ellipses

Double-Click for Example Code

When the data have a bivariate normal distribution, a confidence ellipse for the population mean or a prediction ellipse for a new observation can be computed and displayed. A prediction ellipse is a region for predicting a new observation in the population. It also approximates a region that contains a specified percentage of the population. The ellipse indicates the correlation between the two standardized variables. The following steps produce a scatter plot with a 95% prediction ellipse:

```
proc sgplot data=sashelp.class;
   title 'Prediction Ellipse';
   scatter x=height y=weight;
   ellipse x=height y=weight;
   keylegend / location=inside position=bottomright;
run;

proc template;
   define statgraph ellipse;
      begingraph;
         entrytitle 'Prediction Ellipse';
         layout overlay;
            scatterplot x=height y=weight;
            ellipse x=height y=weight / type=predicted
                    legendlabel='95% Prediction Ellipse' name='ellipse';
            discretelegend 'ellipse' / location=inside
                    halign=right valign=bottom;
```

```
        endlayout;
      endgraph;
   end;
run;

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

In PROC SGPLOT, the SCATTER statement displays the X=Height response variable for the levels of the Y=Weight categorical variable. The ELLIPSE statement displays a 95% prediction ellipse. The KEYLEGEND statement places the legend inside the plot at the bottom right. The results are displayed in Figure 1.95 and Figure 1.96.

In the GTL, the SCATTERPLOT statement displays the X=Height response variable for the levels of the Y=Weight categorical variable. The ELLIPSE statement displays a 95% prediction ellipse. The ELLIPSE statement is named so that it can appear in the legend, and a legend label is provided. The DISCRETELEGEND statement places the legend inside the plot, horizontally positioned on the right and vertically positioned at the bottom of the plot.

Figure 1.95 Prediction Ellipse with PROC SGPLOT

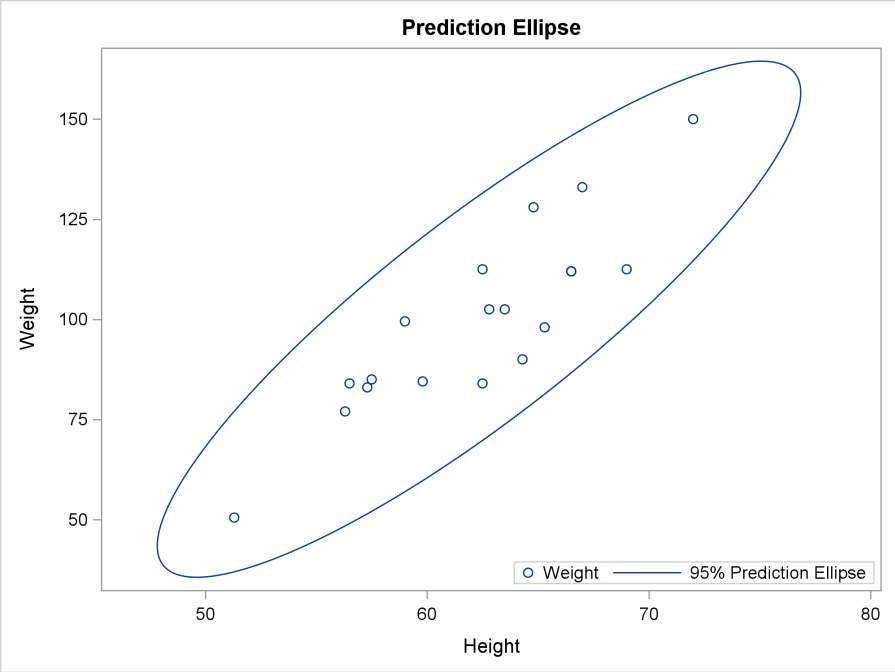
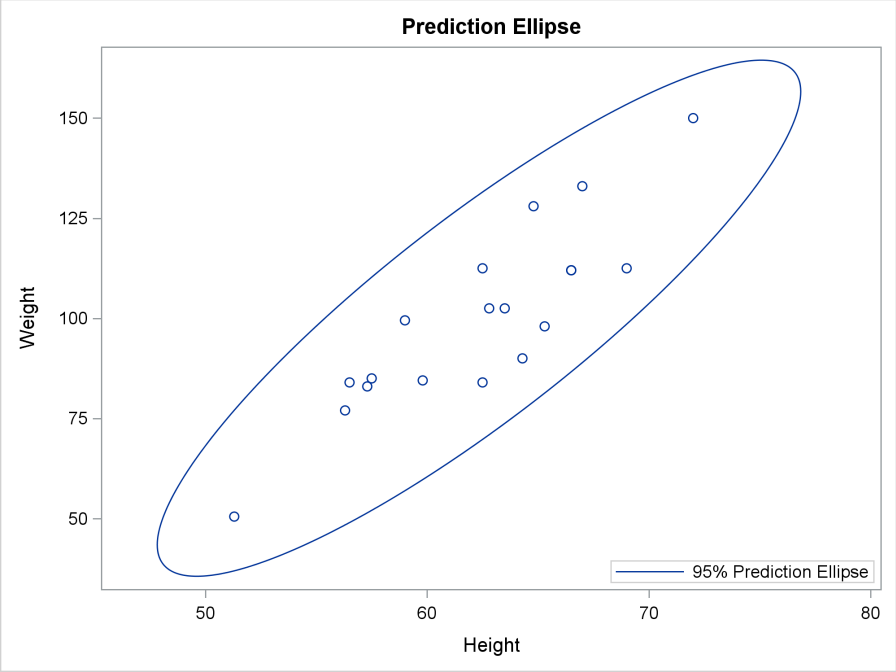


Figure 1.96 Prediction Ellipse with Custom Template



The two plots differ in that PROC SGPLOT by default places information about both the scatter plot and the ellipse in the legend. However, the graph template places only the ellipse information in the legend. The following step produces exactly the same plot as is produced using the GTL.

```
proc sgplot data=sashelp.class;
   title 'Prediction Ellipse';
   scatter x=height y=weight;
   ellipse x=height y=weight / name='e';
   keylegend 'e' / location=inside position=bottomright;
run;
```

Because only one statement name is specified in the KEYLEGEND statement, only the legend for the ellipse appears in the plot. The results of this step are not shown.

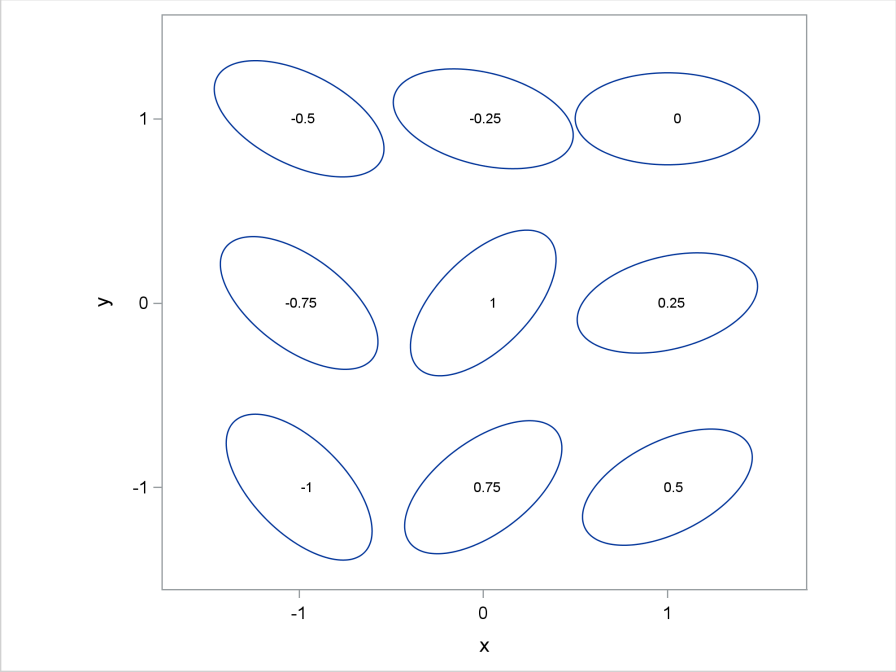
The preceding ellipse examples provide ODS Graphics with raw data, and ODS Graphics does the computations and produces the results. The remainder of this section uses only PROC TEMPLATE and PROC SGRENDER with the ELLIPSEPARM statement. The ELLIPSEPARM statement requires you to provide all the parameters for drawing each ellipse.

The following steps use the ELLIPSEPARM statement to produce some ellipses to illustrate how ellipses are defined and how this statement works:

```
data x;
   major = 0.5;
   minor = 0.25;
   input slope x y;
   l = slope;
   datalines;
-1.00  -1 -1
-0.75  -1  0
-0.50  -1  1
-0.25   0  1
 0.00   1  1
 0.25   1  0
 0.50   1 -1
 0.75   0 -1
 1.00   0  0
;

proc template;
   define statgraph ellipse;
      begingraph;
         layout overlayequated / equatetype=square;
            scatterplot x=x y=y / markercharacter=l;
            ellipseparm semimajor=major semiminor=minor slope=slope
                        xorigin=x yorigin=y;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=x template=ellipse;
run;
```

The results are displayed in Figure 1.97. Ellipses are defined by five parameters, and all must be specified in the ELLIPSEPARM statement. The values can be constant, or they can be variables in a data set. Consider the ellipse in Figure 1.97 that is displayed in the bottom left corner of the plot. It is centered at X=–1 and Y=–1. These X and Y coordinates are the first two parameters. If you were to draw a line through this (X, Y) point to the two points on the ellipse that are farthest from X and Y, the slope of this line would be –1. Half the length of this line is the semimajor parameter. Half the length of the line that connects two points on the ellipse that is perpendicular to the semimajor line is the semiminor parameter. When the semimajor parameter equals the semiminor parameter, the ellipse is a circle and the slope is irrelevant. The ellipse that is centered on the (–1,–1) coordinates has a slope of –1, because that is the slope of the major axis. The length of the major axis is 2×0.5, and the length of the minor axis is 2×0.25. The shapes of the ellipses in Figure 1.97 differ only in their slopes. Starting in the bottom left corner, continuing clockwise, and ending in the middle, the slopes range from –1 to 1.

Figure 1.97 Understanding Ellipses' Slopes



Here is the LAYOUT statement that you use to make this graph:

```
layout overlayequated / equatetype=square;
```

This layout is designed to produce a square plot with equated axes. However, plot elements such as ellipses can cause the axes to be padded and thus rectangular, not square. Nevertheless, a centimeter on one axis represents the same data range as a centimeter on the other axis, and both axes have the same ticks and data range. If these axes had not been equated, the geometry of the ellipses would not be properly displayed in the plot. This would be most noticeable in Figure 1.98 (which is created by the next steps), where the circles would be displayed as ellipses. The SCATTERPLOT statement displays the label variable L, which contains the slope instead of a marker, by using the MARKERCHARACTER= option. The ELLIPSEPARM statement specifies the input data set variables for the five parameters.

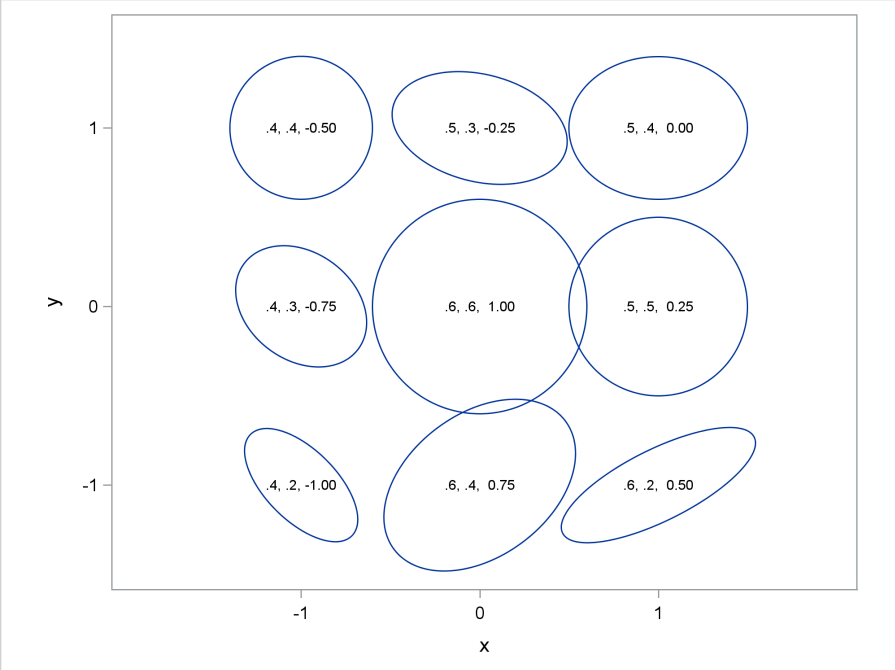
The following steps create a series of ellipses with different parameters:

```
data y;
   input slope major minor x y;
   l = put(major, 2.1) || ', ' || put(minor, 2.1) || ', ' || put(slope, 5.2);
   datalines;
-1.00  .4  .2  -1  -1
-0.75  .4  .3  -1   0
-0.50  .4  .4  -1   1
-0.25  .5  .3   0   1
 0.00  .5  .4   1   1
 0.25  .5  .5   1   0
 0.50  .6  .2   1  -1
 0.75  .6  .4   0  -1
 1.00  .6  .6   0   0
;
```

```
proc sgrender data=y template=ellipse;
run;
```

The label inside each ellipse consists of the semimajor parameter, the semiminor parameter, and the slope. The results are displayed in Figure 1.98.

Figure 1.98 Understanding Ellipses



1.15 Bubble Plot

Double-Click for Example Code

A bubble plot is a scatter plot of two quantitative variables in which the symbols are circles and the area of each circle conveys the value of a third quantitative variable.

The following steps create a bubble plot that shows the height (X axis), weight (Y axis), and age (bubble area) of each member of the Sashelp.Class data set:

```
proc sgplot data=sashelp.class noautolegend;
   format age 2.;
   title 'Bubble Plot';
   bubble  y=weight x=height size=age / group=sex transparency=0.5;
   text    y=weight x=height text=age / group=sex textattrs=(size=5)
                                        position=center;
run;
```

The results are displayed in Figure 1.99. You can remove the fill by specifying the NOFILL option. For this example, the fill helps make it clear that two 15-year-olds share the same height and

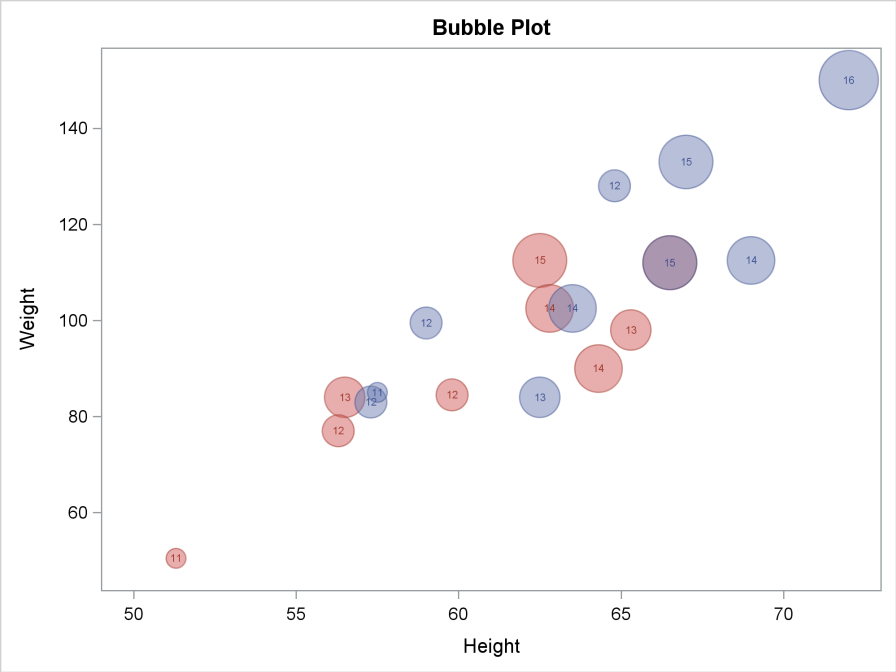
weight. You can use the options BRADIUSMIN= and BRADIUSMAX= to control the minimum and maximum radius, respectively.

This example uses a TEXT statement to label each bubble. You can use either the TEXT statement or the following SCATTER statement:

```
scatter y=weight x=height / group=sex markerchar=agelabel
                            markercharattrs=(size=5);
```

The TEXT statement provides more options for displaying text than the SCATTER statement.

Figure 1.99 Bubble Plot



The following step uses the GTL to make the identical graph:

```
proc template;
   define statgraph classsbubble;
      begingraph;
         entrytitle 'Bubble Plot';
         layout overlay;
            bubbleplot  y=weight x=height size=age / group=sex
                                                    datatransparency=0.5;
            textplot    y=weight x=height text=age / group=sex position=center
                                                    textattrs=(size=5);
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=classsbubble;
   format age 2.;
run;
```

1.16 Line Plot

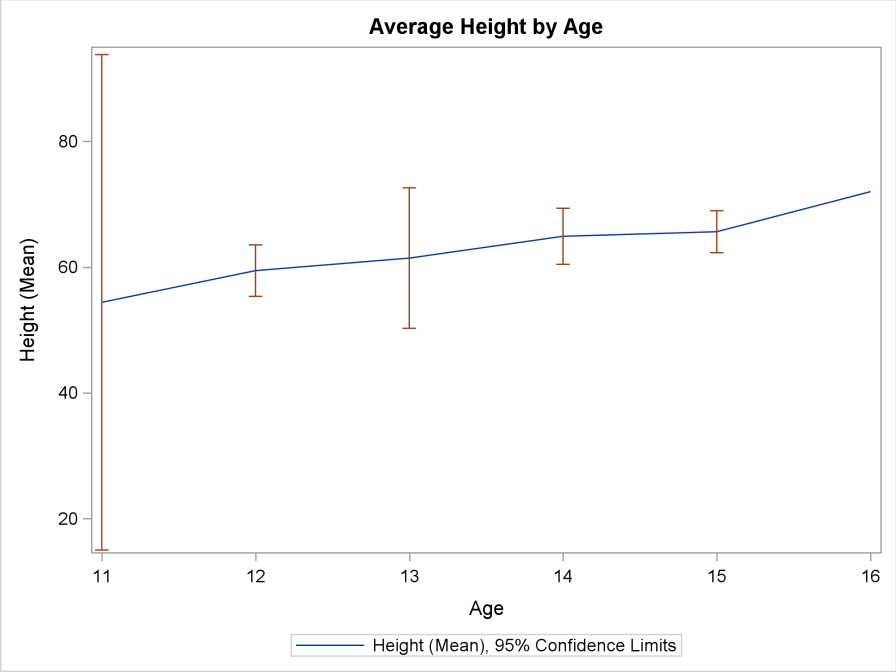
Double-Click for Example Code

A line plot displays summary statistics computed from a quantitative variable for each level of an ordered categorical variable. The points in the plot are connected by line segments. The following steps produce a vertical line plot with 95% confidence limits:

```
proc means data=sashelp.class noprint nway;
   var height;
   class age;
   output out=class mean=height lclm=lower uclm=upper;
   label height = 'Height(Mean)';
run;

proc sgplot data=sashelp.class;
   title 'Average Height by Age';
   vline age / response=height stat=mean limits=both;
run;

proc template;
   define statgraph vline;
      begingraph;
         entrytitle 'Average Height by Age';
         layout overlay / xaxisopts=(type=discrete);
            seriesplot x=age y=height /
                legendlabel='Height (Mean), 95% Confidence Limits' name='vline';
             scatterplot x=age y=height / markerattrs=(size=0)
                yerrorupper=upper yerrorlower=lower;
            discretelegend 'vline';
         endlayout;
       endgraph;
   end;
run;

proc sgrender data=class template=vline;
run;
```

In PROC SGPLOT, the VLINE statement makes a vertical line plot for the RESPONSE=Height variable for the levels of the variable Age, which is specified before the slash. The mean height is shown for each age along with confidence limits in both directions. The graphs are almost identical; one is displayed in Figure 1.100.

Figure 1.100 Vertical Line Plot



The GTL does not have a vertical line plot statement. However, the SCATTERPLOT and SERIESPLOT statements have all the needed options after the data have been processed with PROC MEANS. PROC MEANS creates an output data set that contains for each age group the mean height and two other variables: Lower, which contains the lower end of the confidence interval, and Upper, which contains the upper end of the confidence interval. Additionally, a label is provided for the mean height; this is the same label that PROC SGPLOT automatically generates. The XAXISOPTS=(TYPE=DISCRETE) option in the LAYOUT OVERLAY statement creates a discrete X axis; the default axis is linear. The SCATTERPLOT statement displays the X=Height response variable for the levels of the Y=Age categorical variable. However, the points themselves are not displayed because of the specification MARKERATTRS=(SIZE=0). Error bars are drawn in the range from the values in the YERRORLOWER=Lower variable to the values in the YERRORUPPER=Upper variable. The means are displayed as a series plot. No bar is displayed for the mean age of 16, because there is only one observation for that age group. The SERIESPLOT statement is named so that a legend can be produced. A legend label is provided that matches the legend label that PROC SGPLOT automatically displays. The results are almost identical to the graph displayed in Figure 1.100.

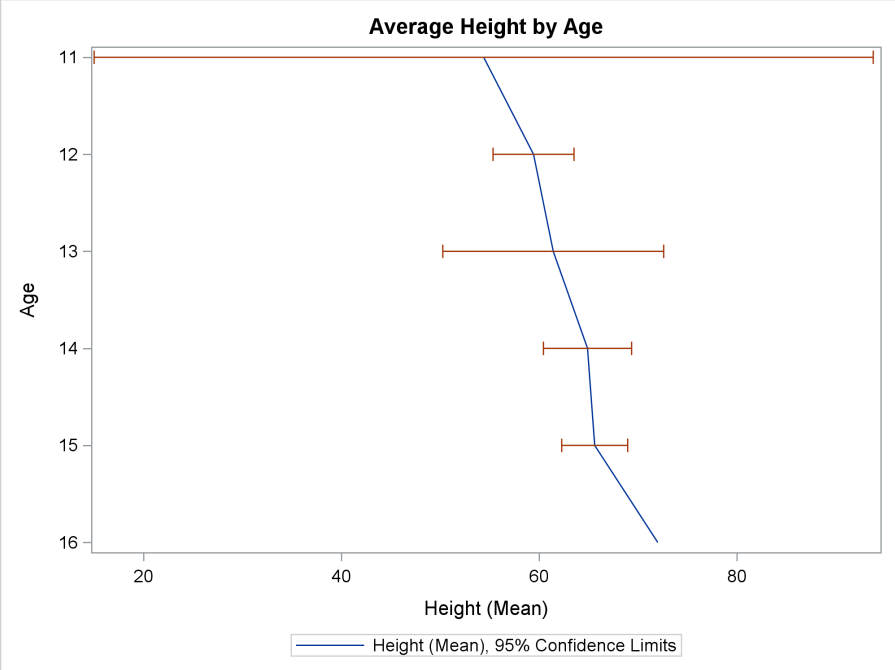
The following steps produce a horizontal line plot with 95% confidence limits:

```
proc means data=sashelp.class noprint nway;
   var height;
   class age;
   output out=class mean=height lclm=lower uclm=upper;
   label height = 'Height(Mean)';
run;
```

```
proc sgplot data=sashelp.class;
   title 'Average Height by Age';
   hline age / response=height stat=mean limits=both;
run;

proc template;
   define statgraph hline;
      begingraph;
         entrytitle 'Average Height by Age';
         layout overlay / yaxisopts=(type=discrete reverse=true);
            seriesplot x=height y=age /
                legendlabel='Height (Mean), 95% Confidence Limits'
                            name='hline';
             scatterplot x=height y=age / markerattrs=(size=0)
                xerrorupper=upper xerrorlower=lower;
            discretelegend 'hline';
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=class template=hline;
run;
```

The steps that produce the horizontal line plots are obvious variations on the steps that produce the vertical line plots. The graphs are almost identical; one is displayed in Figure 1.101.

Figure 1.101 Horizontal Line Plot



1.17 Needle Plot

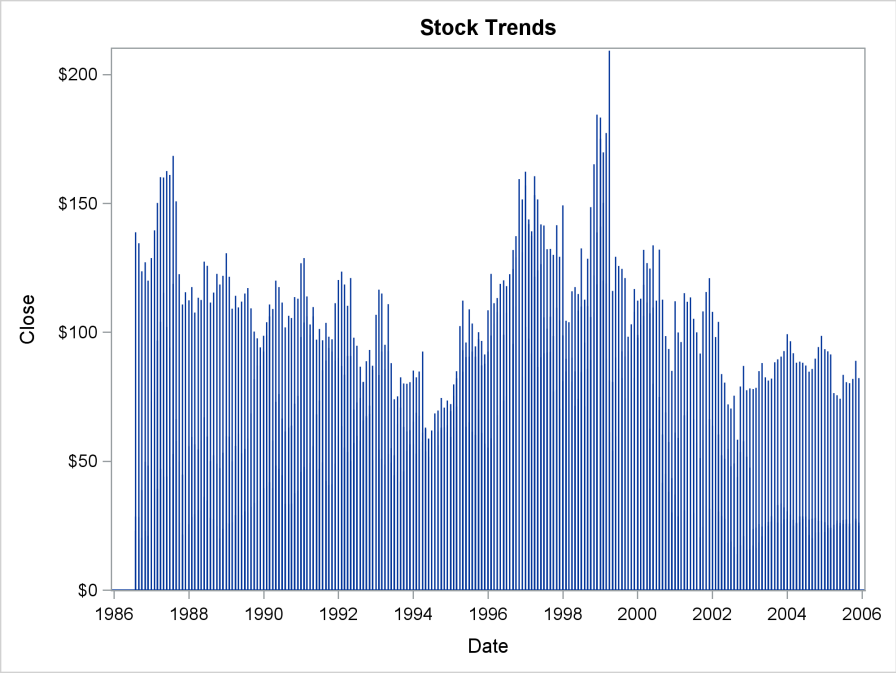
Double-Click for Example Code

A needle plot is a scatter plot in which each point is displayed as a line that connects the horizontal axis and the coordinates of the point. The following steps produce a needle plot:

```
proc sgplot data=sashelp.stocks;
   title 'Stock Trends';
   needle x=date y=close;
run;

proc template;
   define statgraph needle;
      begingraph;
         entrytitle 'Stock Trends';
         layout overlay;
            needleplot x=date y=close;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.stocks template=needle;
run;
```

In PROC SGPLOT, the NEEDLE statement produces a needle plot by using the same options. In the GTL, the NEEDLEPLOT statement produces a needle plot for the X=Date variable on the X axis and for the Y=Close variable on the Y axis. The graphs are almost identical; one is displayed in Figure 1.102. With sparser data sets, the distinct lines are visible.

Figure 1.102 Needle Plot



1.18 Step Plot

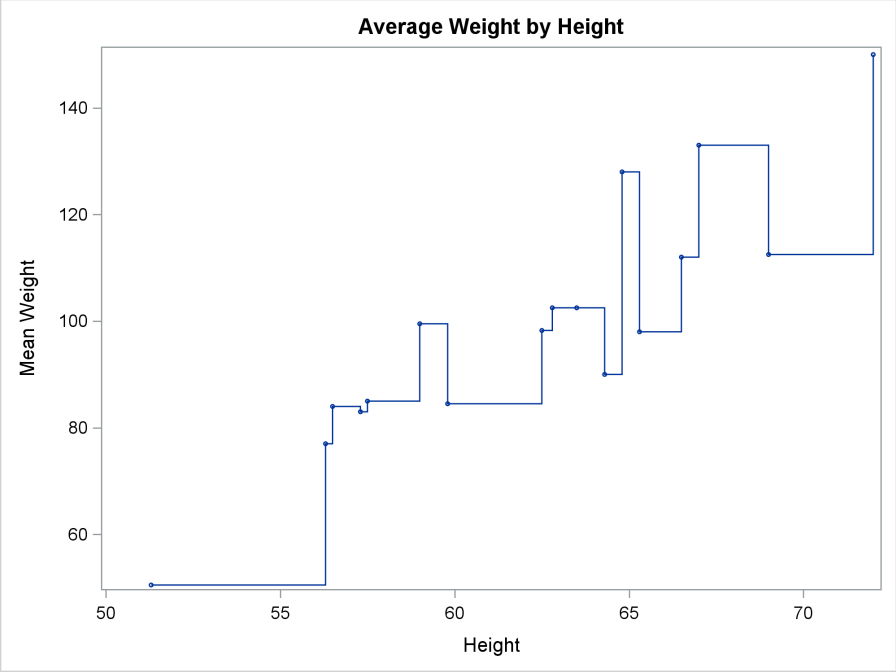
Double-Click for Example Code

A step plot is a graphical display of two quantitative variables in which the points are connected by a combination of horizontal and vertical lines. The points are optionally displayed on the lines. The function resembles a set of stairs that has horizontal landings and vertical risers but no diagonal lines. You can choose to have the vertical segment rise from a point, to a point, or in the middle between points. The following steps produce a step plot:

```
proc means data=sashelp.class noprint nway;
   var weight;
   class height;
   output out=class mean=weight lclm=lower uclm=upper;
   label weight = 'Mean Weight';
run;

proc sgplot data=class;
   title 'Average Weight by Height';
   step x=height y=weight;
run;

proc template;
   define statgraph step;
      begingraph;
         entrytitle 'Average Weight by Height';
         layout overlay;
            stepplot x=height y=weight / display=(markers)
                                         markerattrs=(size=3px);
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=class template=step;
run;
```

In the GTL, the STEPPLOT statement connects the data points for the X=Height variable on the X axis and for the Y=Weight variable on the Y axis. The results are displayed in Figure 1.103. In PROC SGPLOT, the STEP statement connects the data points by using the same options. The graphs are almost identical; one is displayed in Figure 1.103.

Figure 1.103 Step Plot



The plots differ because the options DISPLAY=(MARKERS) MARKERATTRS=(SIZE=3PX) are specified in the GTL. These options display the markers at the means by using a three-pixel symbol. The default marker in the HTMLBlue style is a five-pixel circle. The following step uses PROC SGPLOT to create the same plot that was created by PROC SGRENDER:

```
proc sgplot data=class;
   title 'Average Weight by Height';
   step x=height y=weight / markers markerattrs=(size=3px);
run;
```

The results are identical to the graph displayed in Figure 1.103.

1.19 Vector Plot

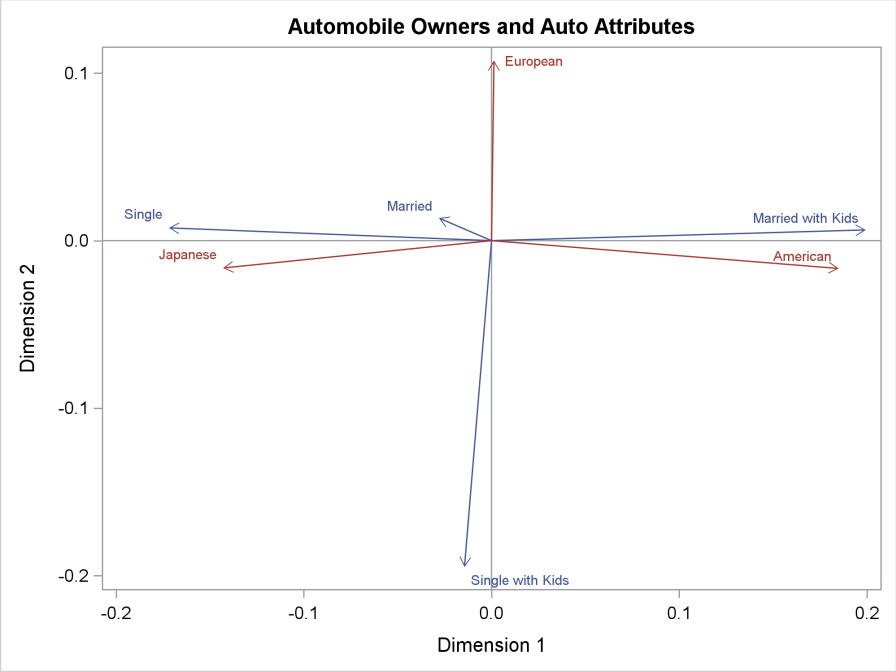
Double-Click for Example Code

A vector plot is a graphical display of two quantitative variables in which each point is represented by a vector or line that emanates from the origin and ends at the point. A reference line is a horizontal or vertical line that runs the width or length of the graph. The following steps produce a vector plot with reference lines from some correspondence analysis results:

```
data corresp;
   input Type $ Name $ 5-25 Dim1 Dim2;
   label dim1 = 'Dimension 1' dim2 = 'Dimension 2';
   datalines;
OBS  Married              -0.02783     0.01339
OBS  Married with Kids     0.19912     0.00639
OBS  Single               -0.17160     0.00762
OBS  Single with Kids     -0.01440    -0.19470
VAR  American              0.18472    -0.01660
VAR  European              0.00129     0.10734
VAR  Japanese             -0.14278    -0.01630
;

proc sgplot data=corresp noautolegend;
   title 'Automobile Owners and Auto Attributes';
   refline 0 / axis=x;
   refline 0 / axis=y;
   vector x=dim1 y=dim2 / datalabel=name group=type;
run;

proc template;
   define statgraph vector;
      begingraph;
         entrytitle 'Automobile Owners and Auto Attributes';
         layout overlayequated / equatetype=fit;
            referenceline x=0;
            referenceline y=0;
            vectorplot y=dim2 x=dim1 xorigin=0 yorigin=0 /
                       datalabel=name group=type;
          endlayout;
      endgraph;
   end;
run;

proc sgrender data=corresp template=vector;
run;
```

In PROC SGPLOT, the VECTOR statement produces vectors that emanate from the origin and end at the coordinates specified by the X=Dim1 variable on the X axis and the Y=Dim2 variable on the Y axis. The vectors are labeled by the values of the DATALABEL=Name variable. There are two groups of observations that are distinguished by the GROUP=Type variable. The results are displayed in Figure 1.104.

Figure 1.104 Vector Plot with PROC SGPLOT



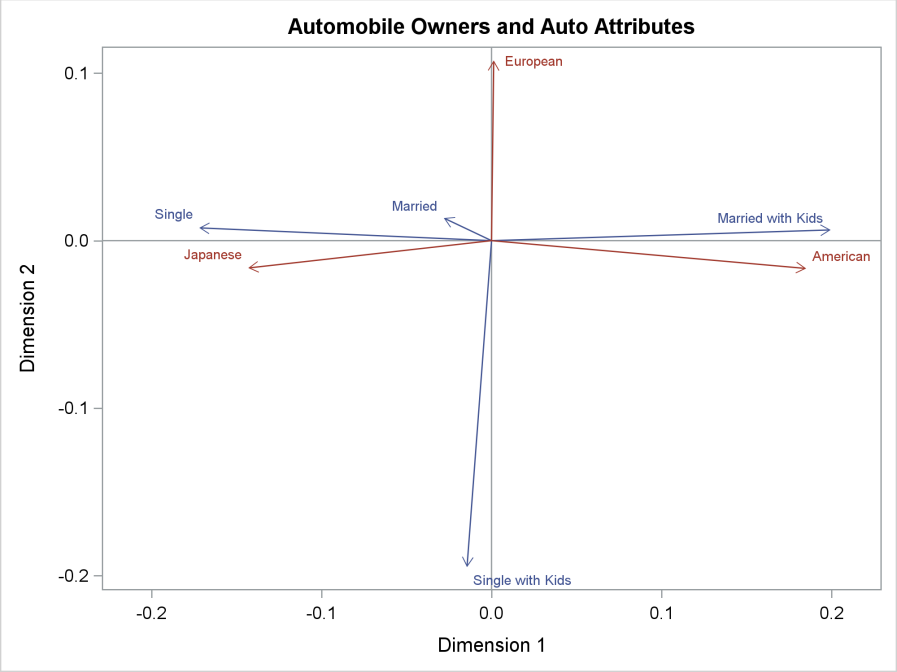
The GTL REFERENCELINE statement options and the PROC SGPLOT REFLINE statements produce reference lines along the X and Y axes at 0. The reference line statements are specified ahead of the vector plot statements so that the reference lines are drawn first and the vectors next. This way, if there are any collisions, the reference lines are in the background and are covered by the information in the vector plot.

In the GTL, the VECTORPLOT statement produces vectors that emanate from the origin (XORIGIN=0 YORIGIN=0) and end at the coordinates specified by the X=Dim1 variable on the X axis and the Y=Dim2 variable on the Y axis. The vectors are labeled by the values of the DATALABEL=Name variable. There are two groups of observations that are distinguished by the GROUP=Type variable. The LAYOUT statement is as follows:

```
layout overlayequated / equatetype=fit;
```

This layout produces a plot with equated axes, where a centimeter on one axis represents the same data range as a centimeter on the other axis. The lengths of the axes are stretched to fit in the outer box. The results are displayed in Figure 1.105.

Figure 1.105 Vector Plot with Equated Axes



The two plots are similar but slightly different because the axes in Figure 1.105 are equated, whereas the axes in Figure 1.104 are not. PROC SGPLOT does not support equated axes.

1.20 Contour Plot

Double-Click for Example Code

A contour plot displays three quantitative variables. The X= and Y= variables form a regular and rectangular grid, and the values of the Z= variable are displayed with shades of colors that vary over the range of the Z= variable. A contour plot can be used to show probability density functions, surfaces, and terrain. The following steps create contour plots of two surfaces: $z = xy(x^2 - y^2)/(x^2 + y^2)$ and a bivariate normal density function.

```
proc template;
   define statgraph contour;
      mvar title;
      begingraph;
         entrytitle title;
         layout overlayequated / equatetype=square
                xaxisopts=(offsetmin=0 offsetmax=0)
                yaxisopts=(offsetmin=0 offsetmax=0);
            contourplotparm x=x y=y z=z;
         endlayout;
      endgraph;
   end;
run;
```

```
data swirl;
   do x = -5 to 5 by 0.1;
      do y = -5 to 5 by 0.1;
         z = x * x + y * y;
         if z > 1e-16 then z = x * y * (x * x - y * y) / z;
         output;
      end;
   end;
run;

data normal;
   do x = -2 to 2 by 0.1;
      do y = -2 to 2 by 0.1;
         z = 0.164 * exp(-0.5 * ((x + y * 0.25) * x + (x * 0.25 + y) * y));
         output;
      end;
   end;
run;

%let title = Contour Plot of the Swirl Data;
proc sgrender data=swirl template=contour;
run;

%let title = Contour Plot of a Bivariate Normal Density;
proc sgrender data=normal template=contour;
run;
```

The results are displayed in Figure 1.106 and Figure 1.107. Both plots are created by the GTL and PROC SGRENDER. Although PROC SGPLOT and the other SG procedures create a rich variety of plots, they cannot be used to create every type of plot that you can create with the GTL. In particular, the SG procedures cannot make contour plots. Also, PROC SGPLOT cannot be used to make equated axes, where a centimeter on one axis represents the same data range as a centimeter on the other axis.

Figure 1.106 Contour of Swirl Data

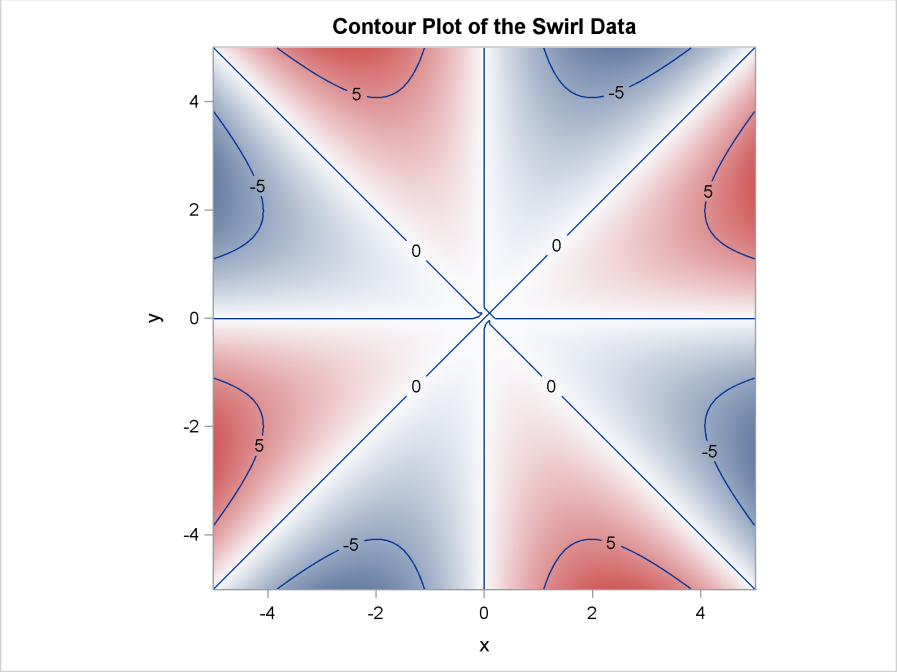
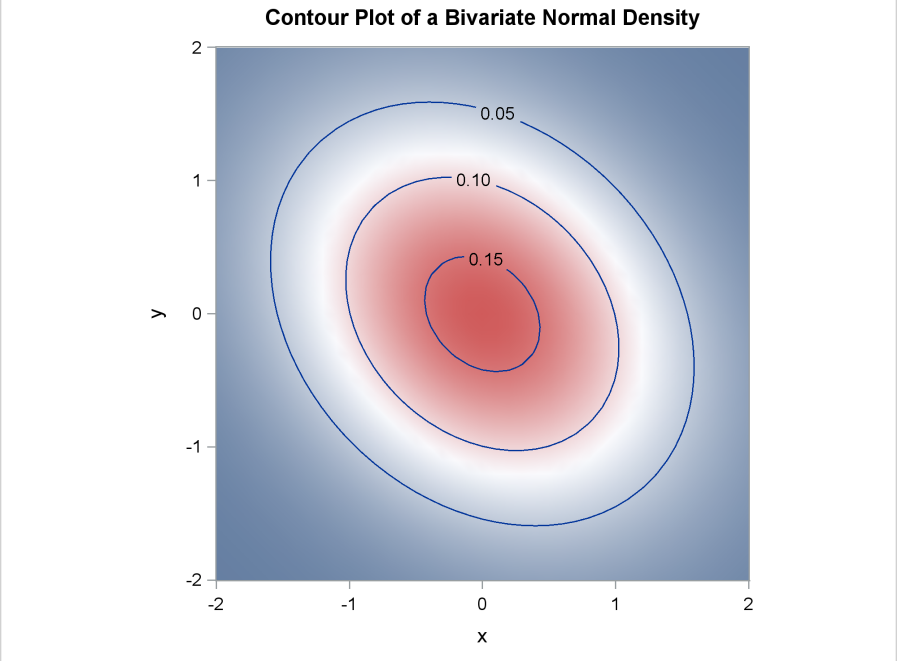


Figure 1.107 Contour of Bivariate Normal Density



This template has a LAYOUT OVERLAYEQUATED statement with the option EQUATETYPE=SQUARE. This creates an equated plot that is square. By default with the LAYOUT OVERLAYEQUATED statement, the axes are equated but the plot is rectangular. The options **xAxisOpts=(OffsetMin=0 OffsetMax=0)** and **yAxisOpts=(OffsetMin=0 OffsetMax=0)** ensure that no extra white space appears between the axes and the edges of the contours. By default, a small offset or white space might appear on the axes. The CONTOURPLOTPARM statement specifies the X-axis variable X and the Y-axis variable Y along with the density variable Z. The macro variable that is named in the MVAR statement sets the title when PROC SGRENDER is run instead of when PROC TEMPLATE is run.

1.20.1 Continuous Legend

The following steps create contour plots with a continuous legend and override the colors in the style:

```
proc template;
   define statgraph contour;
      mvar title;
      begingraph;
         entrytitle title;
         layout overlayequated / equatetype=square
                xaxisopts=(offsetmin=0 offsetmax=0)
                yaxisopts=(offsetmin=0 offsetmax=0);
            contourplotparm x=x y=y z=z / name='cont'
                                         colormodel=(CXFF6666 CXFFBBFF CX6666FF);
            continuouslegend 'cont';
         endlayout;
      endgraph;
   end;
run;

%let title = Contour Plot of the Swirl Data;
proc sgrender data=swirl template=contour;
run;

%let title = Contour Plot of a Bivariate Normal Density;
proc sgrender data=normal template=contour;
run;
```

The results are displayed in Figure 1.108 and Figure 1.109. The continuous legend is created by the CONTINUOUSLEGEND statement with the name `'cont'`, which is also specified in the CONTOURPLOTPARM statement in the NAME= option. By default, the continuous legend is displayed vertically on the right side of the plot. It shows the mapping between the values of a variable and a range of colors, in this case from red to magenta to blue. You can also use continuous legends with scatter plots and other types of graphs. For example, you can create a scatter plot of two variables, y and x, on the Y and X axes and use color to show a third variable, z.

Figure 1.108 Contour of Swirl Data

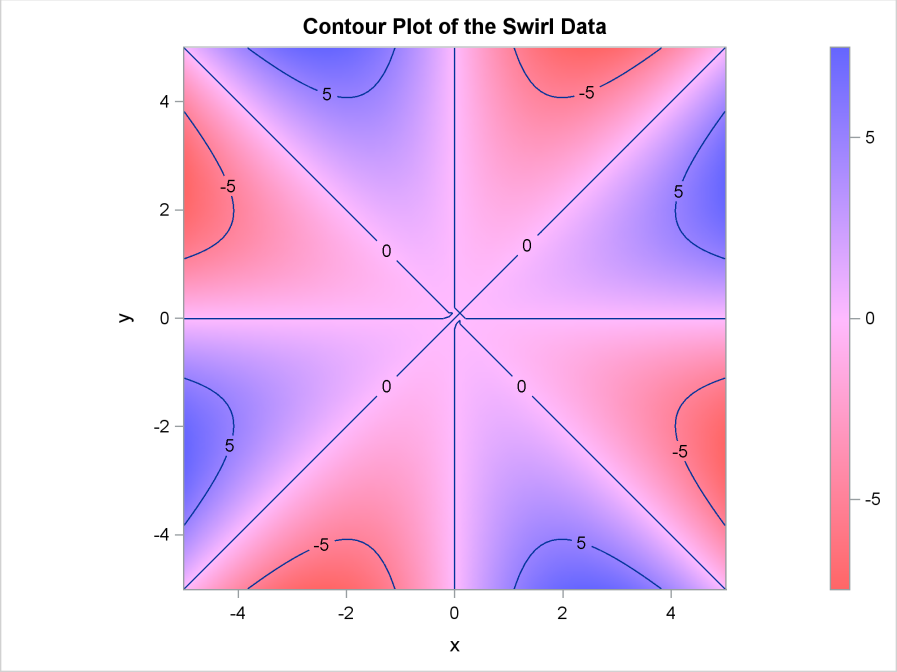
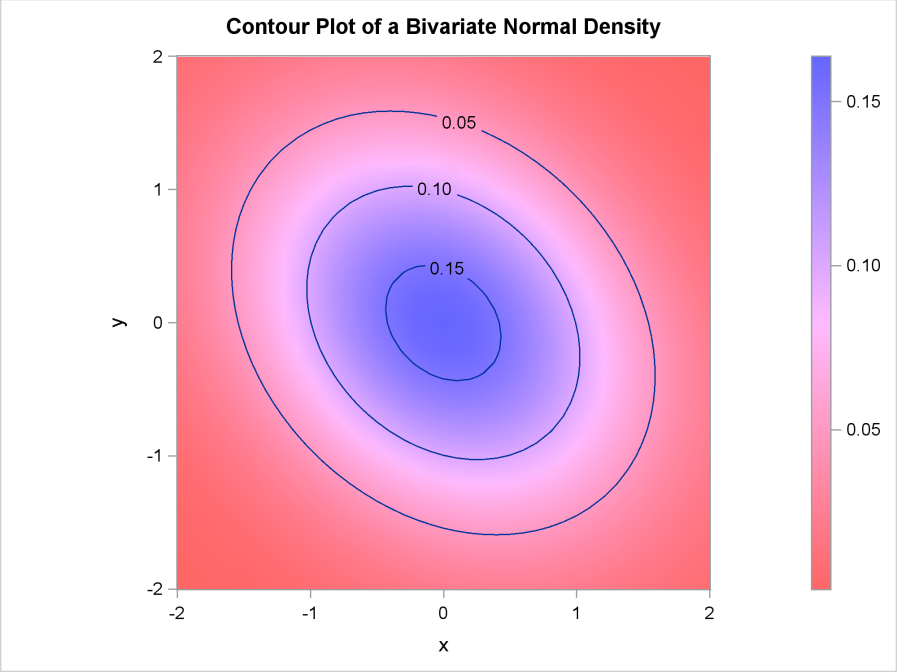


Figure 1.109 Contour of Bivariate Normal Density



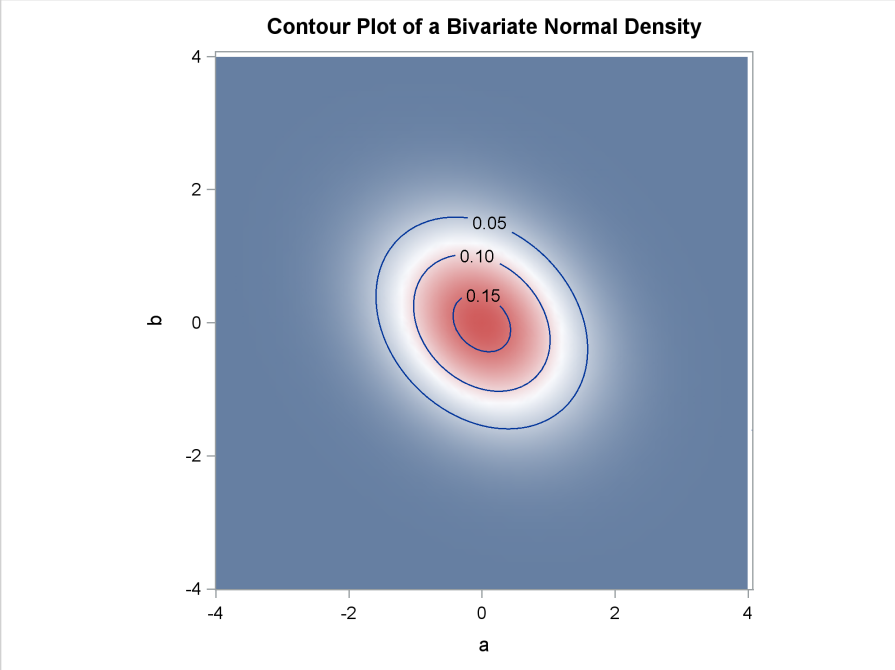
The colors in Figure 1.108 and Figure 1.109 differ from the colors in Figure 1.110 and Figure 1.111 because the **`ThreeColorRamp`** style element was overridden by the COLORMODEL= option.

1.20.2 Contour Plot and Scatter Plot Overlaid

This example makes several important points about template writing. In this first part, a SCATTERPLOT statement is added before the CONTOURPLOTPARM statement. You will see that this particular specification overlays the scatter plot and the contour plot but not in the most desirable way. The following steps create the plot in Figure 1.110:

```
proc template;
   define statgraph contourplotparm;
      mvar title;
      begingraph;
         entrytitle title;
         layout overlayequated / equatetype=square
            xaxisopts=(offsetmin=0 offsetmax=0)
            yaxisopts=(offsetmin=0 offsetmax=0);
            scatterplot x=a y=b / markerattrs=(size=1px);
            contourplotparm x=x y=y z=z;
         endlayout;
      endgraph;
   end;
run;

data normal;
   do x = -4 to 4 by 0.1;
      do y = -4 to 4 by 0.1;
         z = 0.164 * exp(-0.5 * ((x + y * 0.25) * x + (x * 0.25 + y) * y));
         output;
      end;
   end;
run;

data sample(drop=i);
   do i = 1 to 10000;
      a = normal(104);
      b = -0.25 * a + sqrt(1 - 0.25 ** 2) * normal(104);
      output;
   end;
run;

data normal;
   merge normal sample;
run;

proc sgrender data=normal template=contourplotparm;
run;
```

Figure 1.110 Contour of Bivariate Normal Density

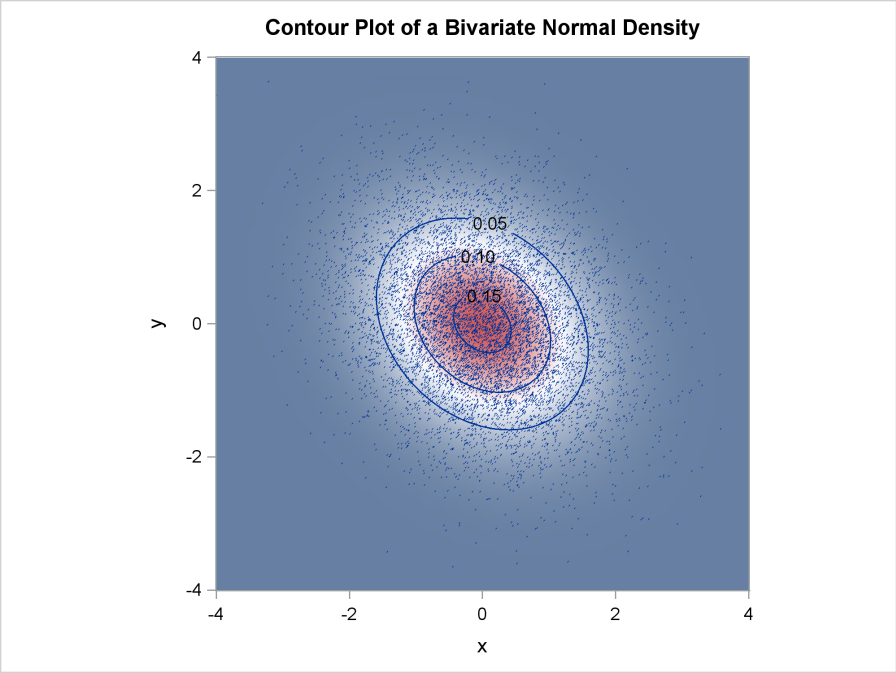


In this example, four variables are plotted: X and Y (as in the previous example) and also A and B, which contain a random sample from the bivariate normal distribution. The plot displayed in Figure 1.110 has 'a' and 'b' (the scatter plot variable names) on the axes and has white space between the contours and the axes. However, the scatter plot itself is not displayed. Although the GTL syntax in this example is perfectly valid, it does not make the most desirable graph for several reasons that are discussed in the rest of this section. The graph that displays both the scatter plot and the contour plot is displayed in Figure 1.111 and is produced by the following steps:

```
proc template;
   define statgraph contourplotparm;
      mvar title;
      begingraph;
         entrytitle title;
         layout overlayequated / equatetype=square
            commonaxisopts=(viewmin=-4 viewmax=4)
            xaxisopts=(offsetmin=0 offsetmax=0)
            yaxisopts=(offsetmin=0 offsetmax=0);
            contourplotparm x=x y=y z=z / contourtype=gradient;
            scatterplot x=a y=b / markerattrs=(size=1px);
            contourplotparm x=x y=y z=z / contourtype=labeledline;
         endlayout;
      endgraph;
   end;
run;
```

```
data normal;
   do x = -4 to 4 by 0.1;
      do y = -4 to 4 by 0.1;
         z = 0.164 * exp(-0.5 * ((x + y * 0.25) * x + (x * 0.25 + y) * y));
         output;
      end;
   end;
run;

data sample(drop=i);
   do i = 1 to 10000;
      a = normal(104);
      b = -0.25 * a + sqrt(1 - 0.25 ** 2) * normal(104);
      output;
   end;
run;

data normal;
   merge normal sample;
run;

proc sgrender data=normal template=contourplotparm;
run;
```

Figure 1.111 Bivariate Normal Density and Sample



The graph in Figure 1.110 is produced by first creating a scatter plot (because the SCATTERPLOT statement appears first) and then creating a contour plot on top of the scatter plot. Statements are executed in the order in which they are specified in the template. The scatter plot is created in Figure 1.110, but it is completely overwritten by the contour plot. In Figure 1.111, the contour plot is written first, so the scatter plot is visible, and then the contour lines are written on top, over both the contour plot and the scatter plot. In Figure 1.110, the SCATTERPLOT statement is the primary plot statement because it is specified first in the layout. The default axis features (axis type, axis labels, and so on) are set by the primary plot. When the statement order is reversed to create the plot in Figure 1.111, the CONTOURPLOTPARM statement becomes primary, and the axis labels are 'x' and 'y' instead of 'a' and 'b'. Alternatively, you can designate any plot statement as primary by specifying the option PRIMARY=TRUE after a slash.

The plot in Figure 1.110 is produced using the options **`xAxisOpts=(OffsetMin=0 OffsetMax=0) yAxisOpts=(OffsetMin=0 OffsetMax=0)`** to eliminate white space between the data values and the axes. In Figure 1.111, the options **`CommonAxisOpts=(ViewMin=-4 ViewMax=4)`** are additionally specified. In Figure 1.110, there is some additional white space between the axes and the contour plot near the maximum value of 4 on both axes. This does not happen in Figure 1.111. The difference occurs because the random sample contains values slightly outside the range of –4 to 4, which is used for the population distribution. The options **`CommonAxisOpts=(ViewMin=-4 ViewMax=4)`** that are specified for Figure 1.111 ensure that these values are not displayed and do not add extra white space. These options are also useful when there are outliers that compress the display of other points.

The option **`MarkerAttrs=(Size=1Px)`** creates one-pixel markers. The scatter plot consists of 10,000 points, and larger markers (such as the default seven-pixel markers) are too large to appropriately display both the sample and the population.

The data set Normal has five variables: X, Y, and Z contain 6,561 nonmissing values, and A, and B contain 10,000 nonmissing values. Hence the variables X, Y, and Z contain a block of 3,439 missing values. This is not a problem, and it is not unusual for some variables to have a large block of missing values when different plots are overlaid.

There are two basic structures for the input data sets that are used when the graphical display consists of multiple components. Graphs such as those shown in Figure 1.14, Figure 1.88, and elsewhere use a GROUP= variable to display multiple groups of observations in the same plot. Data from different groups are stacked in the same variables, and a single plotting statement is used in the template. In contrast, and in this example, different graph types are produced, multiple plotting statements are used, and separate variables in the input data set are used to make each plot.

1.21 Block Plot

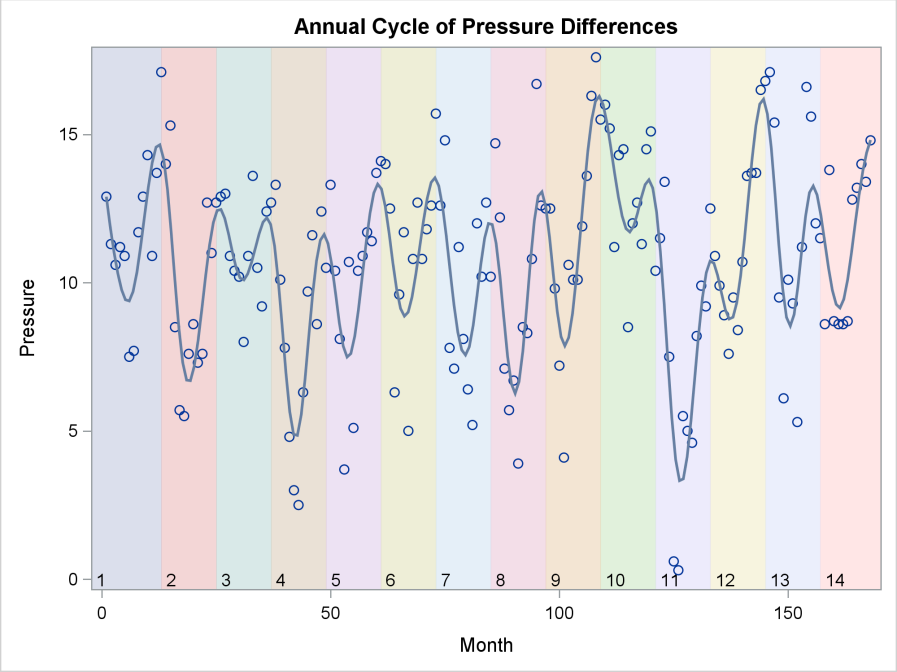
Double-Click for Example Code

A block plot visually enhances an overlaid graph by providing a set of colors that highlight and distinguish intervals of one of the plotted variables. The following steps create a fit plot overlaid on a block plot:

```
proc template;
   define statgraph block;
      begingraph;
         entrytitle 'Annual Cycle of Pressure Differences';
         layout overlay;
            blockplot x=month  block=year / valuevalign=bottom
               datatransparency=0.75 display=(fill values);
            scatterplot  y=pressure x=month;
            pbsplineplot y=pressure x=month;
         endlayout;
      endgraph;
   end;
run;

data Enso;
   set sashelp.enso;
   Year   = ceil(month / 12);
   ElNino = ceil(month / 42);
run;

proc sgrender data=enso template=block;
run;
```

The results are displayed in Figure 1.112. The Enso data set contains a variable, Year, that contains integer values in the range 1 to 14. The block plot displays each year by using a color that is distinguishable from the other colors in the vicinity (although colors can eventually repeat). The X-axis variable is Month, which is the same as in the scatter plot and penalized B-spline plot. The values of the Year variable are displayed because the DISPLAY= option specifies VALUES. They are displayed at the bottom of the graph because the option VALUEVALIGN=BOTTOM is specified. By default, the standard display is unoutlined, filled bands without labels. The locations where the values are displayed are controlled by the options VALUEHALIGN=LEFT | CENTER | RIGHT | START and VALUEVALIGN=TOP | CENTER | BOTTOM.

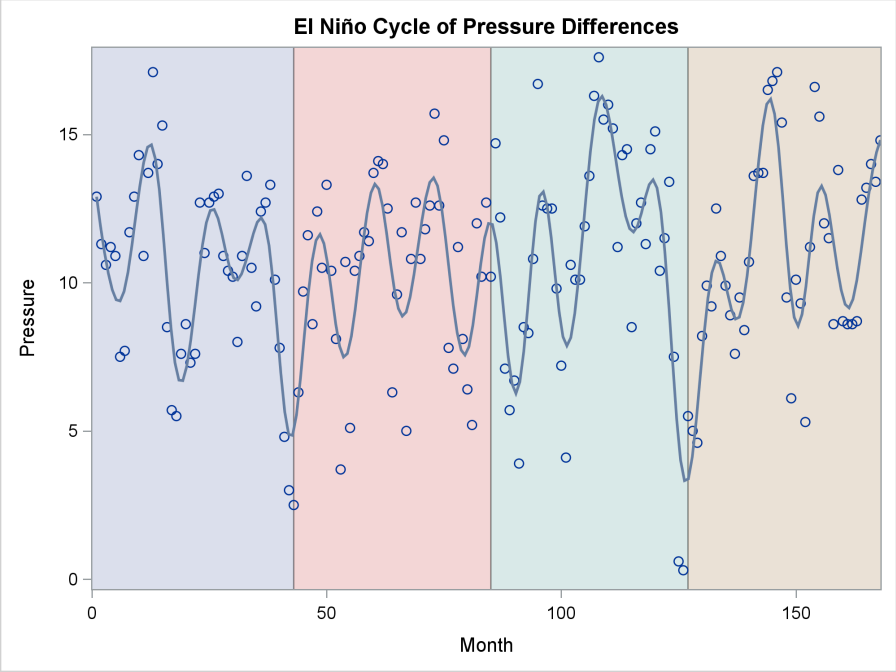
Figure 1.112 Block Plot with Annual Cycle



The following steps create a fit plot overlaid on a block plot of the 42-month El Niño weather cycle, which is displayed in Figure 1.113:

```
proc template;
   define statgraph block2;
      begingraph;
         entrytitle 'El Ni(*ESC*){Unicode "00F1"x}o '
                    'Cycle of Pressure Differences';
         layout overlay / xaxisopts=(offsetmin=0 offsetmax=0);
            blockplot x=month  block=elnino /
               datatransparency=0.75 display=(fill outline);
            scatterplot  y=pressure x=month;
            pbsplineplot y=pressure x=month;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=enso template=block2;
run;
```

Figure 1.113 Block Plot with El Niño Cycle



In Figure 1.112, the values are displayed along with a filled area for each year because of the option DISPLAY=(FILL VALUES). The boxes at both ends of the X axis are a little larger than the other boxes. In Figure 1.113, the values are not displayed, and an outlined filled area for each cycle is displayed because of the option DISPLAY=(FILL OUTLINE). All boxes are the same width because of the specification **`xAxisOpts=(OffsetMin=0 OffsetMax=0)`**. There is a trade-off here. In Figure 1.112, the boxes are not all uniform, but there is room on both edges for the markers, which are centered at the beginning and end of each year and hence extend beyond the year. In Figure 1.113, the boxes are uniform, but the extreme markers are clipped. In both graphs, the option DATATRANSPARENCY=0.75 is specified so that the boxes are 75% as transparent as the default. This creates lighter boxes and a more subtle effect. In Figure 1.113, the unicode specification **`(*ESC*){Unicode "00F1"x}`** creates the letter "ñ" in "El Niño" in the title. A quoted string followed by an "x" means that the value is interpreted as a hexadecimal constant. The Unicode Consortium (`http://unicode.org/`) provides a page of character codes at `http://www.unicode.org/charts/charindex.html`.

1.22 Waterfall Chart

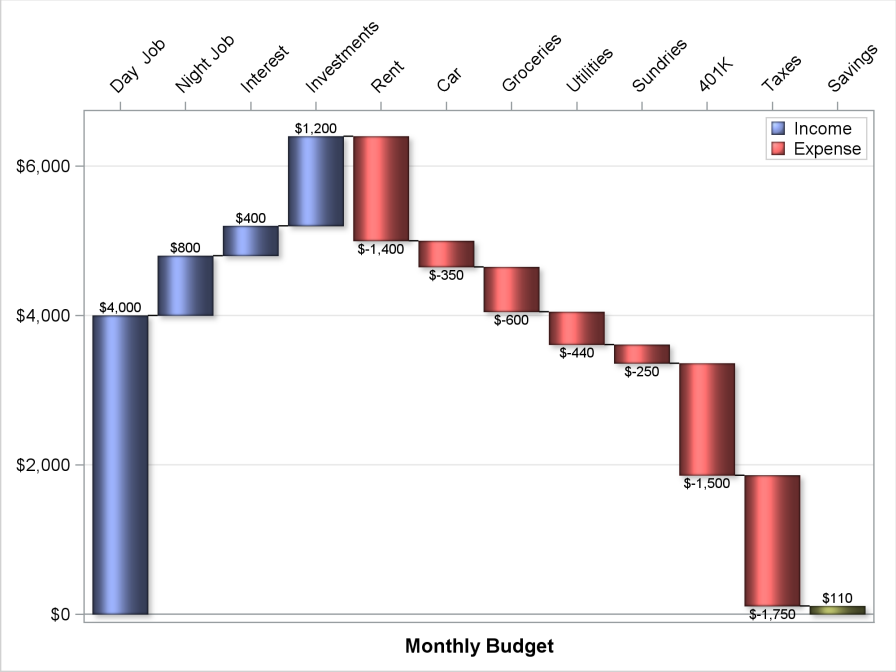
Double-Click for Example Code

A waterfall chart shows an initial value, changes in that value, and its final value. The following steps illustrate by showing monthly income and expenses:

```
data budget;
   input ID $ 1-11 Amount type $;
   datalines;
Day  Job      4000 Income
Night Job      800 Income
Interest       400 Income
Investments   1200 Income
Rent         -1400 Expense
Car           -350 Expense
Groceries     -600 Expense
Utilities     -440 Expense
Sundries      -250 Expense
401K         -1500 Expense
Taxes        -1750 Expense
;

proc sgplot data=budget;
   footnote bold 'Monthly Budget';
   waterfall category=id response=amount / colorgroup=type dataskin=sheen
             datalabel name='a' x2axis finalbartickvalue='Savings';
   keylegend 'a' / location=inside position=topright across=1;
   x2axis      display=(nolabel);
   yaxis  grid display=(nolabel) offsetmax=0.05;
   format Amount dollar.;
run;

footnote;
```

The results are displayed in Figure 1.114. The waterfall chart enables you to compare all sources of income and expenses in a graph. The values of the CATEGORY= variable appear on the horizontal axis, which because of the X2AXIS option is the X2 axis. The RESPONSE= variable appears on the Y axis. The DATASKIN=SHEEN option gives the bars a three-dimensional effect.

Figure 1.114 Waterfall Chart



The following steps use the GTL to create the identical chart:

```
proc template;
   define statgraph water;
      begingraph;
         layout overlay / yaxisopts=(display=(ticks tickvalues line)
                                     offsetmax=0.05 griddisplay=on)
                          x2axisopts=(display=(ticks tickvalues line));
            waterfallchart category=id response=amount / xaxis=x2
                           colorgroup=type legendlabel="Amount"
                           barlabel=true name="a" dataskin=sheen
                           finalbartickvalue="Savings";
            discretelegend "a" / location=inside across=1
                                 halign=right valign=top;
         endlayout;
      entryfootnote textattrs=(weight=bold) "Monthly Budget";
      endgraph;
   end;
run;

proc sgrender data=budget template=water;
   format Amount dollar.;
run;
```

1.23 Three-Dimensional Surface Plot

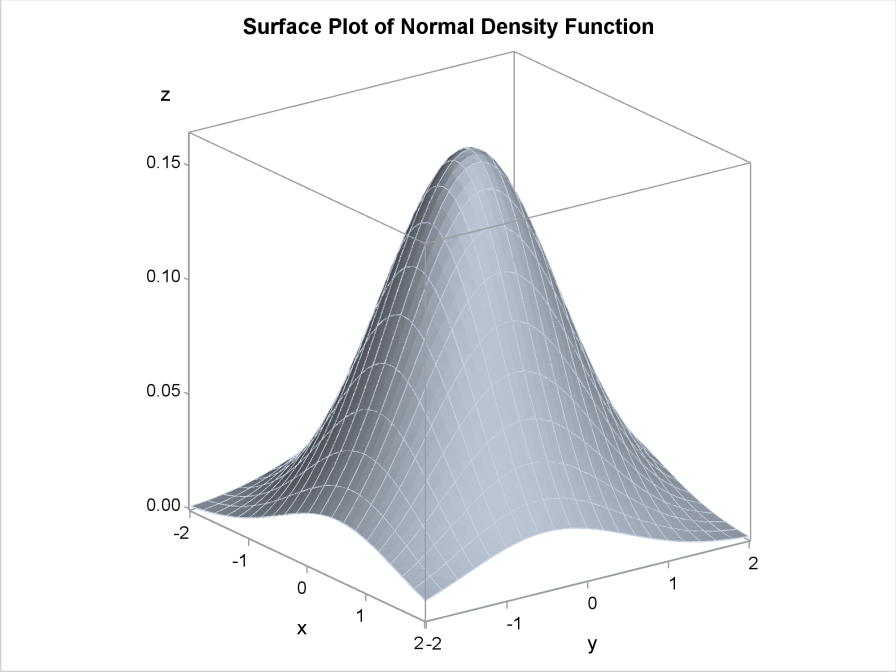
Double-Click for Example Code

A three-dimensional surface plot is a projection of a three-dimensional surface such as a bivariate probability density function or other mathematical surface onto a plane. The shape of the surface can be displayed by using lines, or color and shading, or both. The following steps create a surface plot of a bivariate normal density function and display the results in Figure 1.115:

```
data normal;
   do x = -2 to 2 by 0.1;
      do y = -2 to 2 by 0.1;
         z = 0.164 * exp(-0.5 * ((x + y * 0.25) * x + (x * 0.25 + y) * y));
         output;
      end;
   end;
run;

proc template;
   define statgraph surfaceplotparm1;
      begingraph;
         entrytitle "Surface Plot of Normal Density Function";
         layout overlay3d;
            surfaceplotparm x=x y=y z=z;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=normal template=surfaceplotparm1;
run;
```

The LAYOUT OVERLAY3D statement makes three-dimensional plots such as a three-dimensional surface plot. The SURFACEPLOTPARM statement creates the surface from the X-axis, Y-axis, and Z-axis variables specified in the X=, Y=, and Z= options. By default, the full cubic frame is drawn, and the surface is displayed by a fill color and grid lines. This corresponds to the default option SURFACETYPE=FILLGRID. Other options include SURFACETYPE=FILL for fill only and SURFACETYPE=WIREFRAME for a wire-frame grid only.

Figure 1.115 Surface Plot

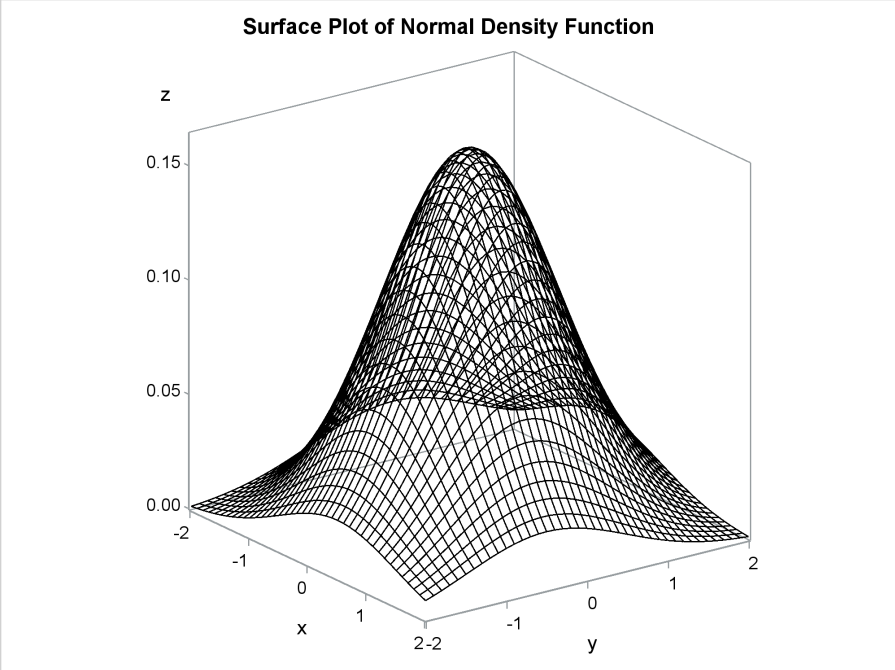


The following steps create a surface plot and display the results in Figure 1.116:

```
proc template;
   define statgraph surfaceplotparm2;
      begingraph;
         entrytitle "Surface Plot of Normal Density Function";
         layout overlay3d / cube=false;
            surfaceplotparm x=x y=y z=z / surfacetype=wireframe
                                          fillattrs=(color=black);
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=normal template=surfaceplotparm2;
run;
```

The CUBE=FALSE option in the LAYOUT OVERLAY3D statement suppresses the display of the full cubic frame. The SURFACETYPE=WIREFRAME option suppresses the fill, and the FILLATTRS=(COLOR=BLACK) option sets the color of the wire-frame grid to black.

Figure 1.116 Wire-Frame Surface Plot



1.24 Three-Dimensional Histogram

Double-Click for Example Code

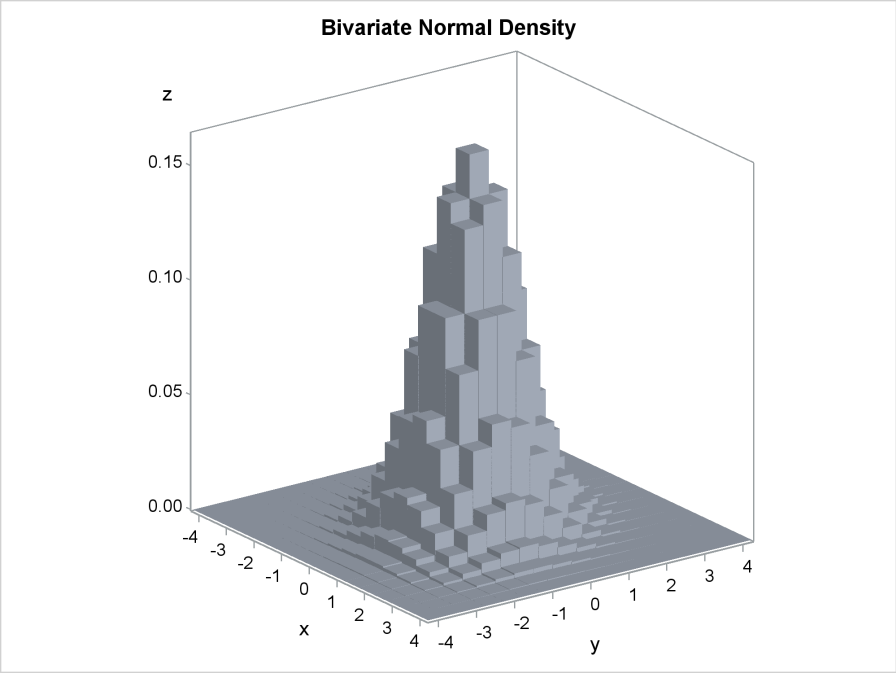
A three-dimensional histogram displays crosstabulations, percentages, density, and so on as solid bars. Each bar represents a nonoverlapping interval of two quantitative variables. The resulting three-dimensional representation is projected onto a plane for display. The following steps create a three-dimensional histogram of a bivariate normal density function and display the results in Figure 1.117:

```
data normal;
   do x = -4 to 4 by 0.5;
      do y = -4 to 4 by 0.5;
         z = 0.164 * exp(-0.5 * ((x + y * 0.25) * x + (x * 0.25 + y) * y));
         output;
      end;
   end;
run;

proc template;
   define statgraph bihistogram1;
      begingraph;
         entrytitle "Bivariate Normal Density";
         layout overlay3d / cube=false;
            bihistogram3dparm x=x y=y z=z;
         endlayout;
      endgraph;
   end;
run;
```

```
proc sgrender data=normal template=bihistogram1;
run;
```

The LAYOUT OVERLAY3D statement makes three-dimensional plots. The CUBE=FALSE option in the LAYOUT OVERLAY3D statement suppresses the display of the full cubic frame. The BIHISTOGRAM3DPARM statement creates the three-dimensional histogram from the X-axis, Y-axis, and Z-axis variables specified in the X=, Y=, and Z= options. Statements whose name contains "PARM" do not do computations to summarize the data. They take results that are already summarized and are suitable for display. In this case, the data set provides a complete grid of X and Y values with a grid increment of 0.5 in both directions. The next example shows how to handle data that are not already in this form.

Figure 1.117 Three-Dimensional Histogram



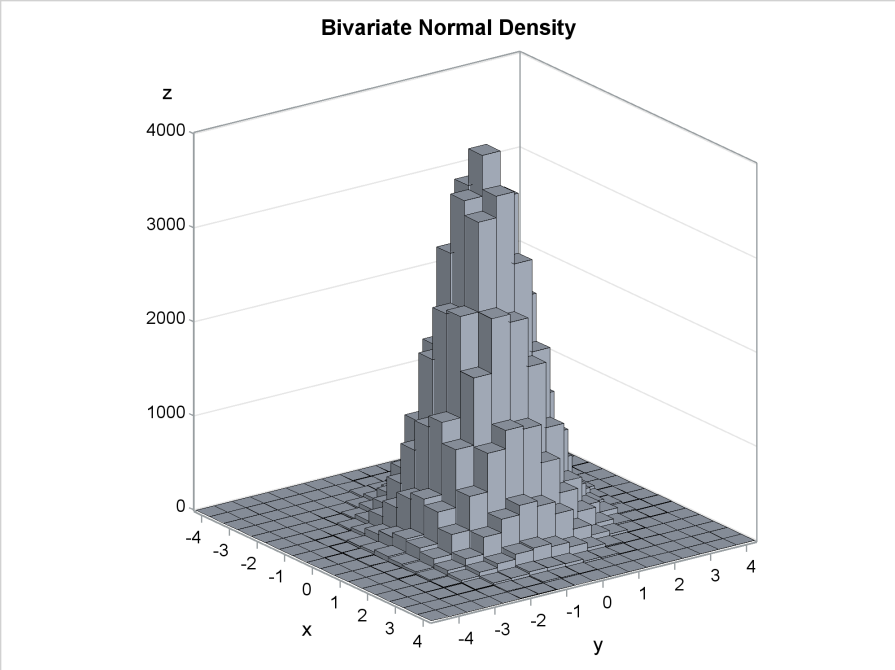
The following steps create a three-dimensional histogram of a random sample of observations from a bivariate normal distribution and display the results in Figure 1.118:

```
data sample(drop=i);
   do i = 1 to 100000;
      x = normal(104);
      y = -0.25 * x + sqrt(1 - 0.25 ** 2) * normal(104);
      output;
   end;
run;

data sample2;
   set sample;
   x = round(x, 0.5);
   y = round(y, 0.5);
run;
```

```
proc summary data=sample2 nway completetypes;
   class x y;
   var y;
   output out=sample3(keep=x y z) n=z;
run;

proc template;
   define statgraph bihistogram2;
      begingraph;
         entrytitle "Bivariate Normal Density";
         layout overlay3d / cube=false zaxisopts=(griddisplay=on);
            bihistogram3dparm x=x y=y z=z / display=all;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sample3 template=bihistogram2;
run;
```

The first DATA step creates the random sample of observations. The second DATA step rounds them to the nearest multiple of 0.5. The PROC SUMMARY step counts how many values occur for every pair of rounded X and Y values. The NWAY option outputs only *n*-way (in this case two-way) frequencies. The COMPLETETYPES option creates all possible combinations of the classification variables even if the combination does not occur in the input data set. This option is required, because the BIHISTOGRAM3DPARM option requires a complete grid. The number of observations in each X and Y pair is stored in the variable Z, which is specified in the N= option.

Figure 1.118 Three-Dimensional Histogram



In the GTL, the **zAxisOpts=(GridDisplay=On)** option in the LAYOUT OVERLAY3D statement displays grid lines along the vertical or Z axis. The DISPLAY=ALL option in the BIHISTOGRAM3DPARM statement displays outlined and filled bins. By default, filled bins with no outlines are produced (as in Figure 1.117).

1.25 References

Cleveland, W. S., Devlin, S. J., and Grosse, E. (1988). "Regression by Local Fitting." *Journal of Econometrics* 37:87–114.

Eilers, P. H. C., and Marx, B. D. (1996). "Flexible Smoothing with *B*-Splines and Penalties." *Statistical Science* 11:89–121. With discussion.

National Institute of Standards and Technology (1998). "Statistical Reference Data Sets." Accessed June 6, 2011. http://www.itl.nist.gov/div898/strd/general/dataarchive.html.

Chapter 2
Panels

Contents

2.1 Scatter Plot Matrix

Double-Click for Example Code

(Code links work in Adobe Reader and Internet Explorer.)

A scatter plot matrix is a rectangular display of scatter plots. It is common for a scatter plot matrix to display scatter plots (for each pair of the variables in the list) in the off-diagonal cells along with univariate histograms and densities (for each variable in the list) in the diagonal cells.

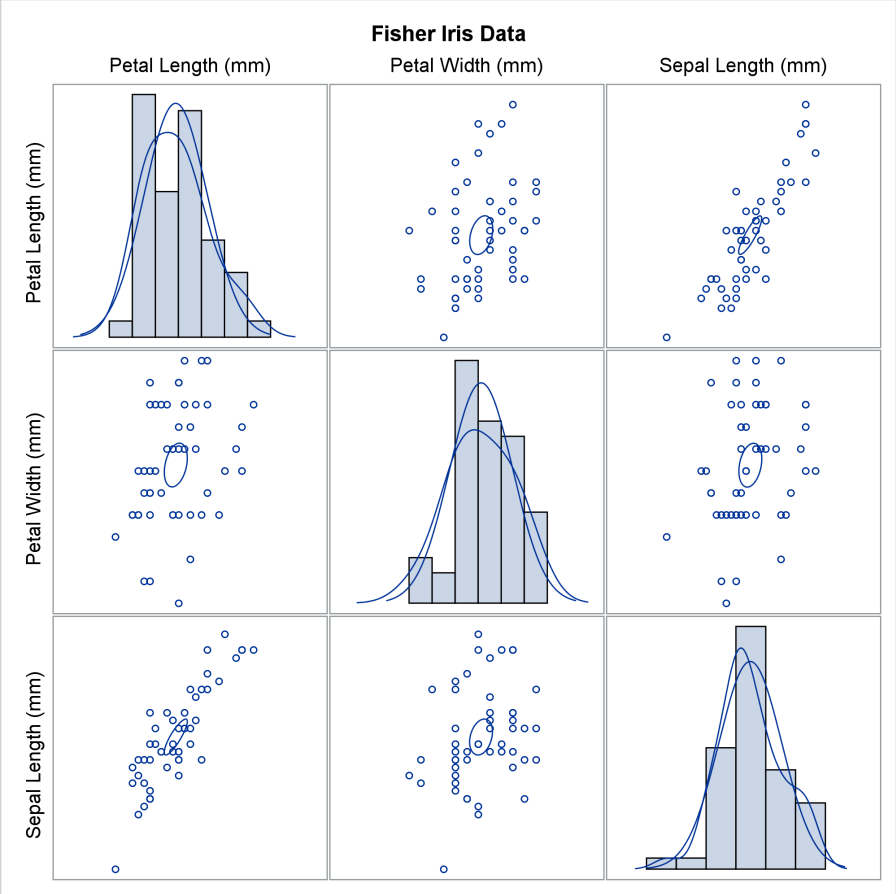
Examples in Chapter 1, "The Graph Template Language and the SG Procedures," rely heavily on PROC SGPLOT. This is because those examples produce single plots. The following example uses PROC SGSCATTER and the GTL to make a scatter plot matrix:

```
proc sgscatter data=sashelp.iris(where=(species eq 'Virginica'));
   title 'Fisher Iris Data';
   matrix petallength petalwidth sepallength /
          ellipse=(type=mean) diagonal=(histogram normal kernel);
run;

proc template;
   define statgraph matrix;
      begingraph / designheight=defaultdesignwidth;
         entrytitle 'Fisher Iris Data';
         layout gridded;
            scatterplotmatrix petallength petalwidth sepallength /
                        ellipse=(type=mean) diagonal=(histogram normal kernel);
         endlayout;
      EndGraph;
   end;
run;

proc sgrender data=sashelp.iris(where=(species eq 'Virginica'))
              template=matrix;
run;
```

Both the MATRIX statement and the SCATTERPLOTMATRIX statement contain a variable list, an ELLIPSE= option to produce a confidence ellipse for the mean in the off-diagonal plots, and

a DIAGONAL= option that controls the displays of density in the diagonal cells. The graphs are almost identical; one is displayed in Figure 2.1.[1]

Figure 2.1 Scatter Plot Matrix



The BEGINGRAPH statement has the option DESIGNHEIGHT=DEFAULTDESIGNWIDTH, which sets the height of the outer box that contains the graph to the default box width. The default design width is 640 pixels, and the default design height is 480 pixels. Hence, DESIGNHEIGHT=DEFAULTDESIGNWIDTH and DESIGNHEIGHT=640px are equivalent. The plot can actually be produced at any size, but by default it is designed at a size that is 480/640 or 3/4 as high as it is wide. The option DESIGNHEIGHT=DEFAULTDESIGNWIDTH is used to make a large square graph, designed for a size of 640 × 640 pixels. Other common sizes in SAS/STAT templates include DESIGNWIDTH=DEFAULTDESIGNHEIGHT (a 480 × 480-pixel square design) and DESIGNHEIGHT=360px (which works well for producing two side-by-side square plots).

All graph templates must contain a LAYOUT statement. Often, a LAYOUT OVERLAY statement or a LAYOUT LATTICE statement is used. However, this template uses a LAYOUT GRIDDED statement along with a SCATTERPLOTMATRIX statement. The SCATTERPLOTMATRIX statement, unlike all plotting statements discussed previously, cannot be specified in a LAYOUT OVERLAY block.

[1] Many graphs in this book are said to be "almost identical" to some other graph. In most cases, a small amount of white space shifts parts of one of the graphs.

The following steps create additional plots. However, this time a 1×3 matrix and a 2×2 matrix of plots are created.

```
proc sgscatter data=sashelp.iris;
   title 'Fisher Iris Data';
   compare y=petallength x=(petalwidth sepallength sepalwidth);
run;

proc template;
   define statgraph compare1;
      begingraph;
         entrytitle 'Fisher Iris Data';
         layout gridded;
            scatterplotmatrix petalwidth sepallength sepalwidth /
                              rowvars=(petallength);
         endlayout;
      EndGraph;
   end;
run;

proc sgrender data=sashelp.iris template=compare1;
run;

proc sgscatter data=sashelp.iris;
   title 'Fisher Iris Data';
   compare y=(petallength petalwidth) x=(sepallength sepalwidth);
run;

proc template;
   define statgraph compare2;
      begingraph / designheight=defaultdesignwidth;
         entrytitle 'Fisher Iris Data';
         layout gridded;
            scatterplotmatrix sepallength sepalwidth /
                              rowvars=(petallength petalwidth);
         endlayout;
      EndGraph;
   end;
run;

proc sgrender data=sashelp.iris template=compare2;
run;
```

In the COMPARE statement in PROC SGSCATTER, the Y= option specifies the variables that appear in the rows of the matrix, and the X= option specifies the variables that appear in the columns of the matrix. In the GTL, the ROWVARS= option specifies the variables that appear in the rows of the matrix, and the variables that appear in the columns are specified before the slash. The results are displayed in Figure 2.2, Figure 2.3, Figure 2.4, and Figure 2.5. In all four graphs, the component plots share axes for the variables that each pair of plots has in common.

The plots are not identical. The GTL uses a system in which the ticks, tick labels, and axis labels alternate from side to side. In contrast, PROC SGSCATTER labels the axes consistently on the left and on the bottom. The differences between these plots are investigated next.

Figure 2.2 1 × 3 Matrix with PROC SGSCATTER

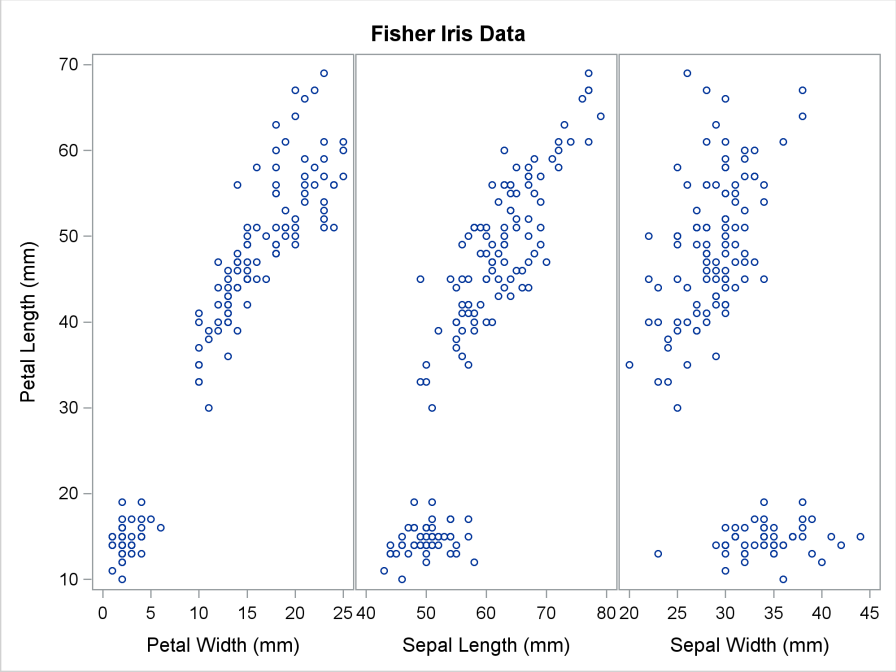


Figure 2.3 1 × 3 Matrix with the GTL

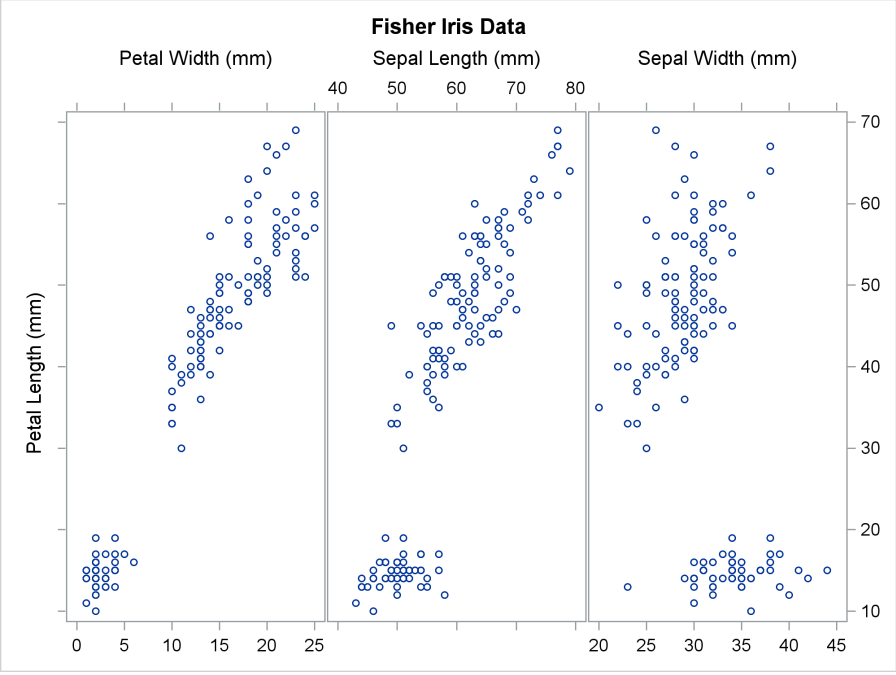


Figure 2.4 2 × 2 Matrix with PROC SGSCATTER

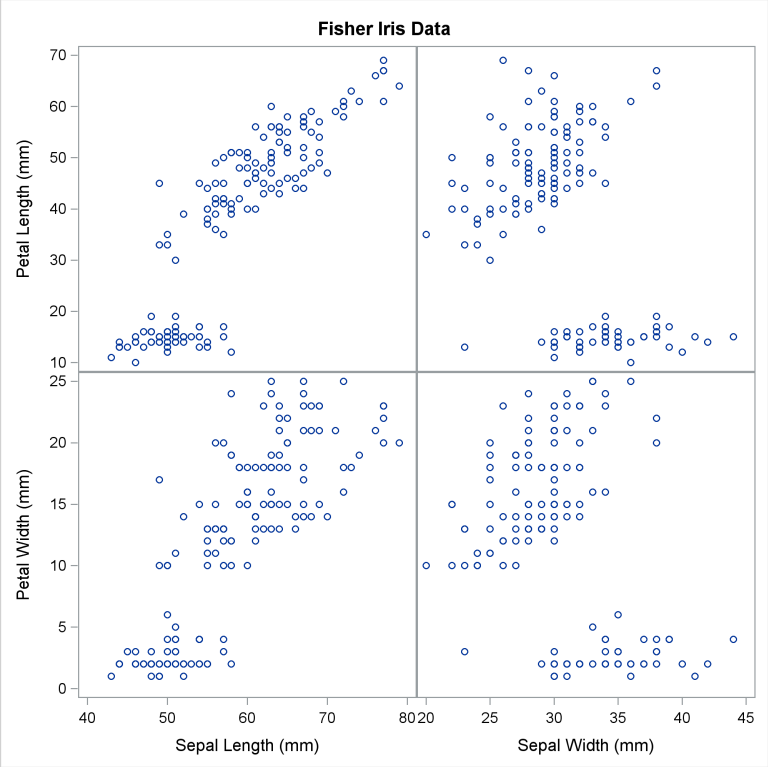
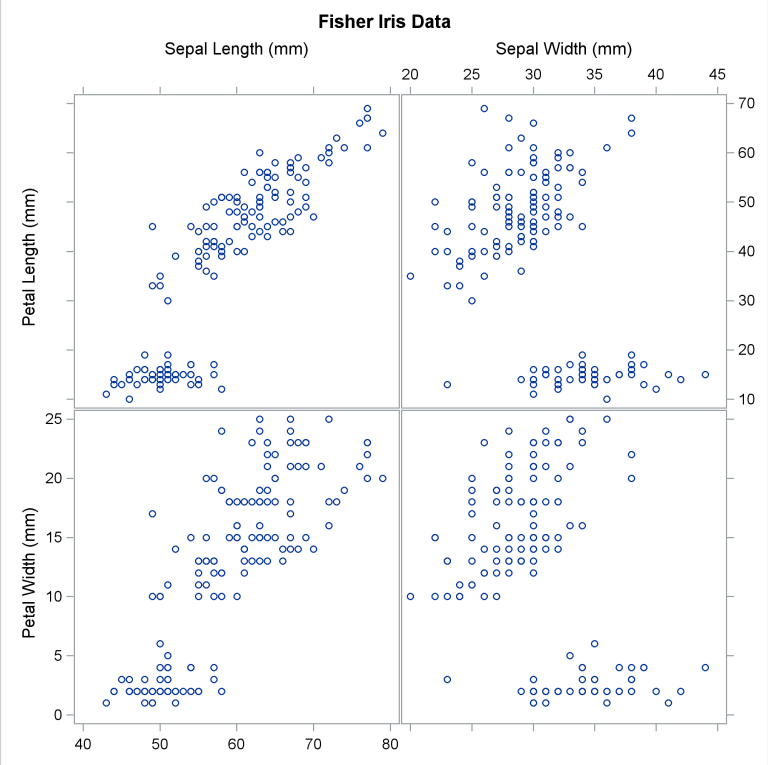


Figure 2.5 2 × 2 Matrix with the GTL



The SG procedures generate a template in the GTL and use it to produce a graph. PROC SGPLOT and PROC SGPANEL have an option in the PROC statement, the TMPLOUT= option, that writes the generated template to a file. You can look at that template and even use or modify it for use with PROC SGRENDER. The following step illustrates this option:

```
proc sgscatter data=sashelp.iris tmplout='template.sas';
   title 'Fisher Iris Data';
   compare y=(petallength petalwidth) x=(sepallength sepalwidth);
run;
```

The generated template is displayed in Figure 2.6. Clearly, the COMPARE statement in PROC SGSCATTER takes a much different approach to creating these plots than a SCATTERPLOTMATRIX statement does in the GTL.

Figure 2.6 COMPARE Statement Generated Template

```
proc template;
define statgraph sgscatter;
begingraph / designwidth=640 designheight=640;
EntryTitle "Fisher Iris Data" /;
layout lattice / pad=(top=5) columns=2 columnDataRange=union rowDataRange=union;
   ColumnAxes;
   ColumnAxis / displaysecondary=none;
   ColumnAxis / displaysecondary=none;
   EndColumnAxes;
   RowAxes;
   RowAxis / displaysecondary=none;
   RowAxis / displaysecondary=none;
   EndRowAxes;
   ScatterPlot X=SepalLength Y=PetalLength / primary=true subpixel=off;
   ScatterPlot X=SepalWidth Y=PetalLength / primary=true subpixel=off;
   ScatterPlot X=SepalLength Y=PetalWidth / primary=true subpixel=off;
   ScatterPlot X=SepalWidth Y=PetalWidth / primary=true subpixel=off;
endlayout;
endgraph;
end;
run;
```

If you want to make a graph, and if you know how to make something similar to what you want by using one of the SG procedures, then you can use the SG procedure to generate a template and use that as a starting point. You can then modify it to get precisely what you want. This might be easier than creating a template from scratch.

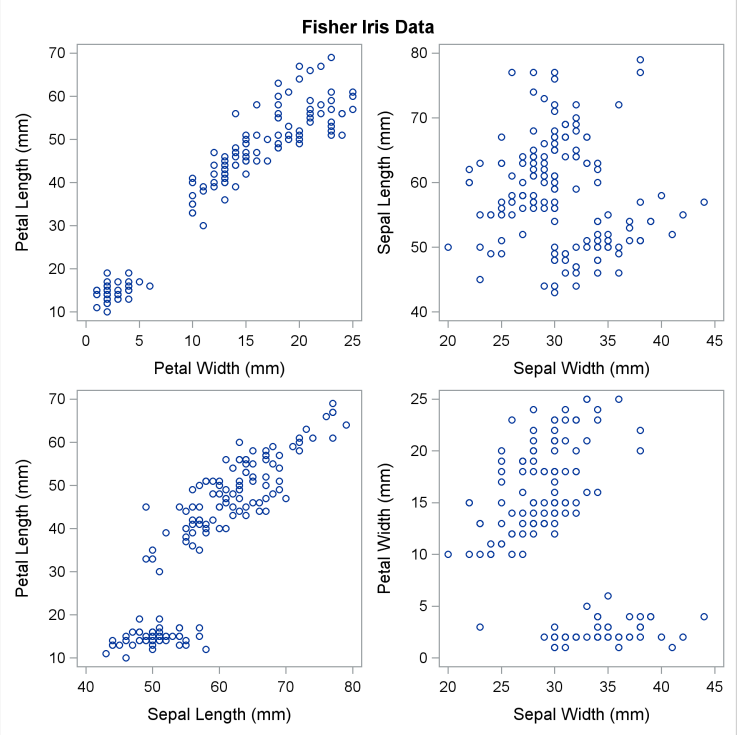
The following steps produce a panel of independent plots, each having separate and independent axes:

```
ods graphics on / height=640px width=640px;

proc sgscatter data=sashelp.iris;
   title 'Fisher Iris Data';
   plot petallength * petalwidth sepallength * sepalwidth
        petallength * sepallength petalwidth * sepalwidth;
run;
```

```
ods graphics on / reset=all;

proc template;
   define statgraph scatter;
      begingraph / designheight=defaultdesignwidth;
         entrytitle "Fisher Iris Data";
         layout lattice / rows=2 columns=2 rowgutter=10 columngutter=10;
            scatterplot x=petalwidth y=petallength;
            scatterplot x=sepalwidth y=sepallength;
            scatterplot x=sepallength y=petallength;
            scatterplot x=sepalwidth y=petalwidth;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.iris template=scatter;
run;
```

In PROC SGSCATTER, the PLOT statement contains a list of each pair of variables to plot and an asterisk between the variables in each pair. The first ODS GRAPHICS statement specifies HEIGHT=640PX so that the plot size will match the plot produced by the GTL, which uses the option DESIGNHEIGHT=DEFAULTDESIGNWIDTH. The second ODS GRAPHICS statement restores the default height. The graphs are almost identical; one is displayed in Figure 2.7.

Figure 2.7 Panel of Plots



In the GTL, the LAYOUT LATTICE statement creates a panel of multiple plots. This particular panel has two rows, two columns, and four SCATTERPLOT statements. By default, there is no space between the rows and columns. In this template, the option ROWGUTTER=10 inserts 10 pixels

between rows, and the option COLUMNGUTTER=10 inserts 10 pixels between columns. The 10 pixels are relative to the designed size of the plot (640 × 640 pixels here because the design height is set to the default design width). If the plot is shrunk, then the 10 pixels are shrunk accordingly. The four SCATTERPLOT statements create the four plots. The results are almost identical to the graph displayed in Figure 2.7.

The following template is equivalent to the template that is used to make Figure 2.7:

```
proc template;
   define statgraph scatter;
      begingraph;
         entrytitle "Fisher Iris Data";
         layout lattice / rows=2 columns=2 rowgutter=10 columngutter=10;
            layout overlay;
               scatterplot x=petalwidth y=petallength;
            endlayout;
            layout overlay;
               scatterplot x=sepalwidth y=sepallength;
            endlayout;
            layout overlay;
               scatterplot x=sepallength y=petallength;
            endlayout;
            layout overlay;
               scatterplot x=sepalwidth y=petalwidth;
            endlayout;
         endlayout;
      endgraph;
   end;
run;
```

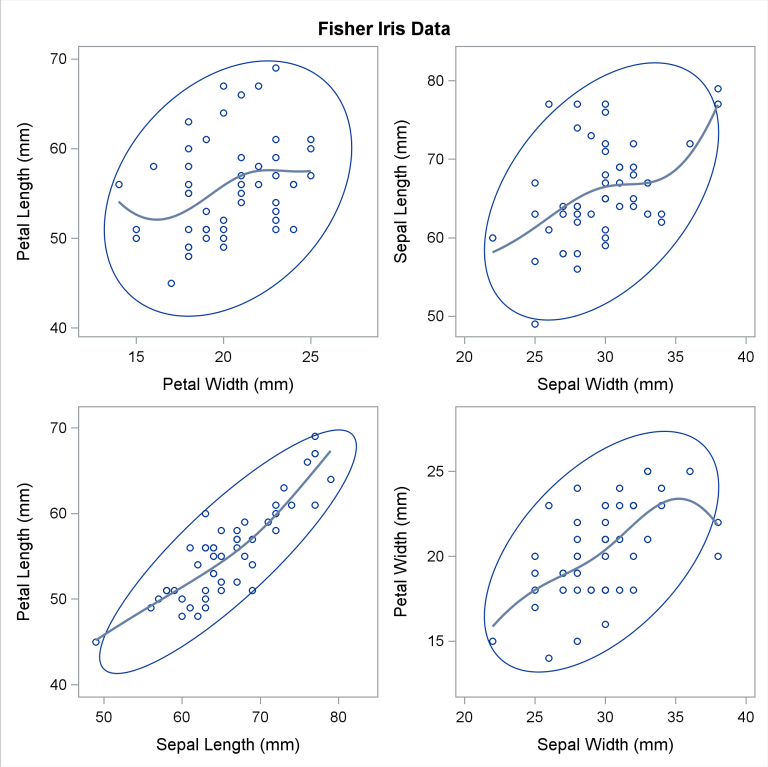
If you want to have multiple plotting statements in each graph, then you need to have a LAYOUT OVERLAY statement that groups them. The following steps illustrate:

```
ods graphics on / height=640px width=640px;

proc sgscatter data=sashelp.iris(where=(species='Virginica'));
   title 'Fisher Iris Data';
   plot petallength * petalwidth sepallength * sepalwidth
        petallength * sepallength petalwidth * sepalwidth /
        ellipse=(type=predicted) pbspline;
run;

ods graphics on / reset=all;

proc template;
   define statgraph scatter;
      begingraph / designheight=defaultdesignwidth;
         entrytitle "Fisher Iris Data";
         layout lattice / rows=2 columns=2 rowgutter=10 columngutter=10;
            layout overlay;
               scatterplot x=petalwidth y=petallength;
               ellipse x=petalwidth y=petallength / type=predicted;
               pbsplineplot x=petalwidth y=petallength;
            endlayout;
```

```
            layout overlay;
               scatterplot x=sepalwidth y=sepallength;
               ellipse x=sepalwidth y=sepallength / type=predicted;
               pbsplineplot x=sepalwidth y=sepallength;
            endlayout;
            layout overlay;
               scatterplot x=sepallength y=petallength;
               ellipse x=sepallength y=petallength / type=predicted;
               pbsplineplot x=sepallength y=petallength;
            endlayout;
            layout overlay;
               scatterplot x=sepalwidth y=petalwidth;
               ellipse x=sepalwidth y=petalwidth / type=predicted;
               pbsplineplot x=sepalwidth y=petalwidth;
            endlayout;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.iris(where=(species='Virginica')) template=scatter;
run;
```

Each LAYOUT OVERLAY block contains three plotting statements. The PROC SGSCATTER step simply adds the following options to make the same plot: **`ellipse=(type=predicted) pbspline`**. Options from many of the statements that are discussed in previous examples are available in the PLOT and COMPARE statements. The graphs are almost identical; one is displayed in Figure 2.8.

Figure 2.8 Panel of Plots



You can simplify writing templates like these by using the macro language, for example, as follows:

```
proc template;
   define statgraph scatter4;
      begingraph / designheight=defaultdesignwidth;
         entrytitle "Fisher Iris Data";
         layout lattice / rows=2 columns=2 rowgutter=10 columngutter=10;

            %macro layout(v1,v2);
               layout overlay;
                  scatterplot  x=&v1 y=&v2;
                  ellipse      x=&v1 y=&v2 / type=predicted;
                  pbsplineplot x=&v1 y=&v2;
               endlayout;
            %mend;

            %layout(petalwidth,  petallength)

            %layout(sepalwidth,  sepallength)

            %layout(sepallength, petallength)

            %layout(sepalwidth,  petalwidth)
         endlayout;
      endgraph;
   end;
run;
```

This step generates the same template that is used to make the panel of plots in Figure 2.8.

2.2 Residual Panel

Double-Click for Example Code

This example uses PROC GLM to fit a linear model and then uses PROC TEMPLATE, the GTL, and PROC SGRENDER to display the residuals as a function of each of the predictor variables. It introduces the CELL, CELLHEADER, and SIDEBAR statements and shows you how to construct common axes and axis labels. The following step fits a linear model by using PROC GLM and creates an output data set called Res that contains all the analysis variables along with a new variable, r, that contains the residuals:

```
ods graphics on;

proc glm data=sashelp.class;
   class sex;
   model weight = height age sex;
   output out=res r=r;
quit;
```

The linear model has two quantitative and continuous predictor variables, Height and Age, and one categorical predictor variable, Sex. Residuals are displayed in scatter plots for the continuous predictor variables and in box plots for the categorical predictor variable.

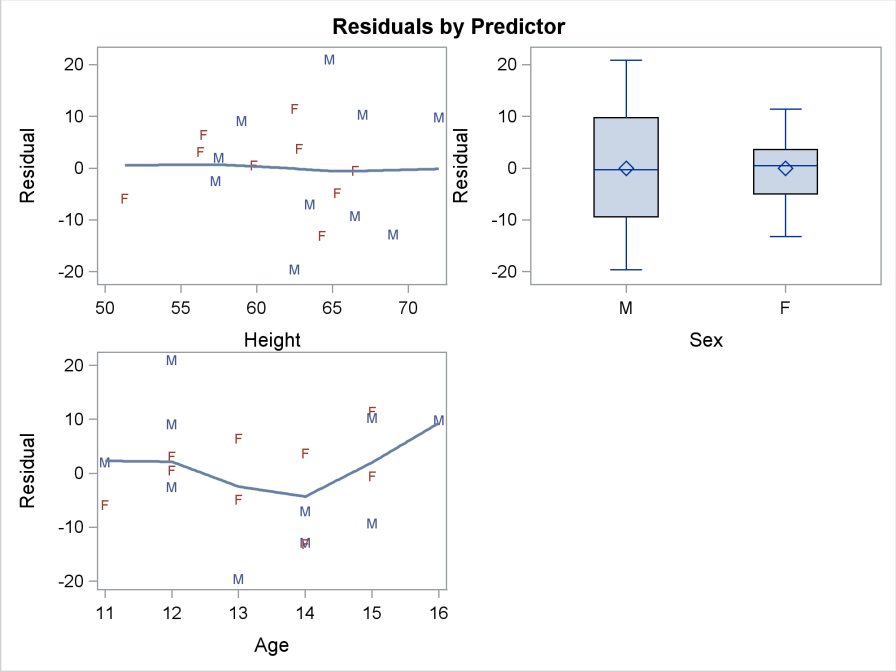
The following step creates a template in the form of a 2×2 lattice that contains three residual plots, one for each of the predictor variables:

```
proc template;
   define statgraph res;
      begingraph;
         entrytitle 'Residuals by Predictor';

         layout lattice / rows=2 columns=2;

            layout overlay;
               scatterplot y=r x=height / markercharacter=sex group=sex;
               loessplot   y=r x=height;
            endlayout;

            layout overlay;
               boxplot y=r x=sex;
            endlayout;

            layout overlay;
               scatterplot y=r x=age / markercharacter=sex group=sex;
               loessplot   y=r x=age;
            endlayout;

         endlayout;
      endgraph;
   end;
run;
```

This template is a starting point and is refined several times to illustrate various statements, options, and functionality. Inside the LAYOUT LATTICE block are three LAYOUT OVERLAY blocks. The first LAYOUT OVERLAY block creates a scatter plot of the residuals as a function of the variable Height. Each observation is displayed as a red `'F'` for females or a blue `'M'` for males. The symbol comes from the MARKERCHARACTER=SEX option, and the color comes from the GROUP=SEX option. Colors are assigned in data order, not alphabetical order (`'M'` comes before `'F'` in the data set). A loess fit plot is overlaid on the scatter plot to see if there is any systematic pattern in the residuals. The second LAYOUT OVERLAY block creates a box plot of the residuals for males and one for females. The third LAYOUT OVERLAY block is like the first, but it is for the variable Age. The following step creates the plot, which is displayed in Figure 2.9:

```
proc sgrender data=res template=res;
   label r ='Residual';
run;
```

Figure 2.9 Basic Residual Panel



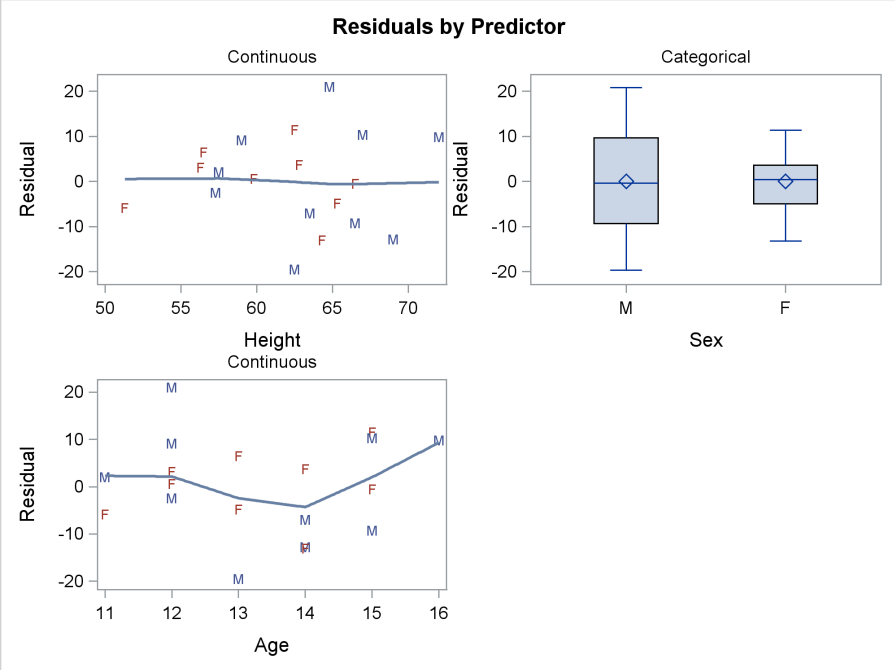
The following PROC TEMPLATE step is similar to the previous one, except that now each graph has a title and each Y axis has a label explicitly set in the template:

```
proc template;
   define statgraph res;
      begingraph;
         entrytitle 'Residuals by Predictor';
         layout lattice / rows=2 columns=2;

            layout overlay / yaxisopts=(label='Residual');
               entry "Continuous" / location=outside;
               scatterplot y=r x=height / markercharacter=sex group=sex;
               loessplot   y=r x=height;
            endlayout;

            layout overlay / yaxisopts=(label='Residual');
               entry "Categorical" / location=outside;
               boxplot y=r x=sex;
            endlayout;

            layout overlay / yaxisopts=(label='Residual');
               entry "Continuous" / location=outside;
               scatterplot y=r x=age / markercharacter=sex group=sex;
               loessplot   y=r x=age;
            endlayout;

         endlayout;
      endgraph;
   end;
run;
```

An ENTRY statement inside the LAYOUT OVERLAY block that contains the option LOCATION=OUTSIDE creates a header or title for each graph. The option `yAxisOpts=(Label='Residual')` in each LAYOUT OVERLAY statement creates the label. The following step creates the plot, which is displayed in Figure 2.10:

```
proc sgrender data=res template=res;
run;
```

Figure 2.10 Panel with Titles and Labels



The previous PROC TEMPLATE step created a title for each graph by using the ENTRY statement. If you need more than one title, the approach illustrated in the following step works better:

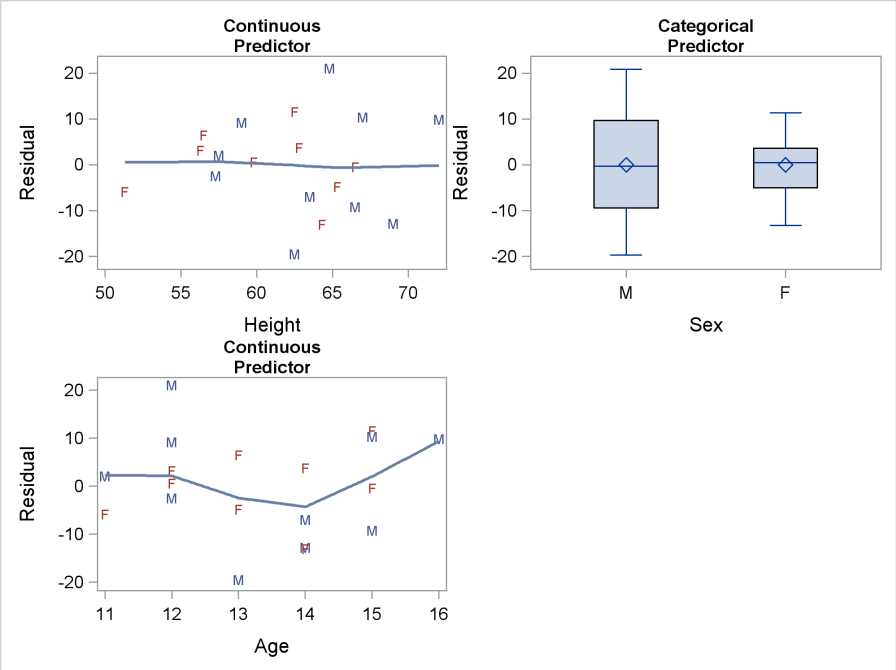
```
proc template;
   define statgraph res;
      begingraph;
         layout lattice / rows=2 columns=2;
            cell;
               cellheader;
                  entry "Continuous" / textattrs=(weight=bold);
                  entry "Predictor" / textattrs=(weight=bold);
               endcellheader;
               layout overlay / yaxisopts=(label='Residual');
                  scatterplot y=r x=height / markercharacter=sex group=sex;
                  loessplot   y=r x=height;
               endlayout;
            endcell;

            cell;
               cellheader;
                  entry "Categorical" / textattrs=(weight=bold);
                  entry "Predictor" / textattrs=(weight=bold);
               endcellheader;
```

```
            layout overlay / yaxisopts=(label='Residual');
               boxplot y=r x=sex;
            endlayout;
         endcell;

         cell;
            cellheader;
               entry "Continuous" / textattrs=(weight=bold);
               entry "Predictor" / textattrs=(weight=bold);
            endcellheader;
            layout overlay / yaxisopts=(label='Residual');
               scatterplot y=r x=age / markercharacter=sex group=sex;
               loessplot   y=r x=age;
            endlayout;
         endcell;

      endlayout;
   endgraph;
end;
run;
```

In this template, each LAYOUT OVERLAY block is wrapped in a CELL statement block that begins with a CELL statement and ends with an ENDCELL statement. Inside the CELL statement block is a CELLHEADER statement block that begins with a CELLHEADER statement and ends with an ENDCELLHEADER statement. Inside that you can provide one or more titles, each in an ENTRY statement. You could use multiple ENTRY statements with LOCATION=OUTSIDE and without the CELL blocks; however, the spacing between titles is better when you use this approach. The following step creates the plot, which is displayed in Figure 2.11:

```
proc sgrender data=res template=res;
run;
```

Figure 2.11 Panel Using Cells



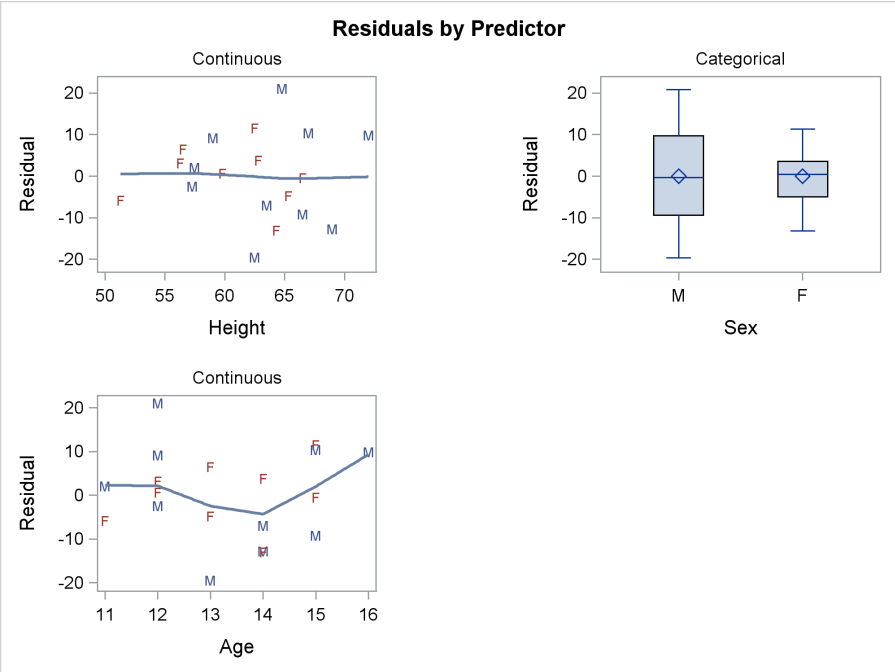
The plots in Figure 2.9 through Figure 2.11 are all close to each of the other plots in their panel. The following PROC TEMPLATE step creates a template for the residual panel, but this time inserts additional space between the plots:

```
proc template;
   define statgraph res;
      begingraph;
         entrytitle 'Residuals by Predictor';
         layout lattice / rows=2 columns=2 rowgutter=20 columngutter=100;
            layout overlay / yaxisopts=(label='Residual');
               entry "Continuous" / location=outside;
               scatterplot y=r x=height / markercharacter=sex group=sex;
               loessplot   y=r x=height;
            endlayout;
            layout overlay / yaxisopts=(label='Residual');
               entry "Categorical" / location=outside;
               boxplot y=r x=sex;
            endlayout;
            layout overlay / yaxisopts=(label='Residual');
               entry "Continuous" / location=outside;
               scatterplot y=r x=age / markercharacter=sex group=sex;
               loessplot   y=r x=age;
            endlayout;
         endlayout;
      endgraph;
   end;
run;
```

This template uses the options **RowGutter=20 ColumnGutter=100** in the LAYOUT LATTICE statement to insert 20 pixels of white space between rows of plots and 100 pixels of white space between columns. The following step creates the plot, which is displayed in Figure 2.12:

```
proc sgrender data=res template=res;
run;
```

Figure 2.12 Panel with Gutter Space



The following step adds several new features, including a common axis label:

```
proc template;
   define statgraph res;
      begingraph;
         entrytitle 'Residuals by Predictor';
         layout lattice / rows=2 columns=2 rowgutter=20 columngutter=20;

            layout overlay / xaxisopts=(offsetmin=0.1 offsetmax=0.1)
                             yaxisopts=(offsetmin=0.1 offsetmax=0.1
                             display=(line ticks tickvalues));
               scatterplot y=r x=height / markercharacter=sex group=sex;
               loessplot   y=r x=height;
            endlayout;

            layout overlay / yaxisopts=(display=(line ticks tickvalues));
               boxplot y=r x=sex;
            endlayout;

            layout overlay / xaxisopts=(offsetmin=0.1 offsetmax=0.1)
                             yaxisopts=(offsetmin=0.1 offsetmax=0.1
                             display=(line ticks tickvalues));
               scatterplot y=r x=age / markercharacter=sex group=sex;
               loessplot   y=r x=age;
            endlayout;

         sidebar / align=left;
            entry "Residual" / rotate=90;
         endsidebar;

         sidebar / align=bottom;
            entry "Analysis of the Class Data Set - July 25, 2009" /
                  textattrs=(weight=bold);
         endsidebar;

         endlayout;
      endgraph;
   end;
run;
```

Each LAYOUT OVERLAY statement has the option **yAxisOpts=(display=(line ticks TickValues))**, which differs from the default in that the axis label is omitted. Instead of individual axis labels, a single label is placed in the far left, or "sidebar," area by the following statements:

```
sidebar / align=left;
   entry "Residual" / rotate=90;
endsidebar;
```

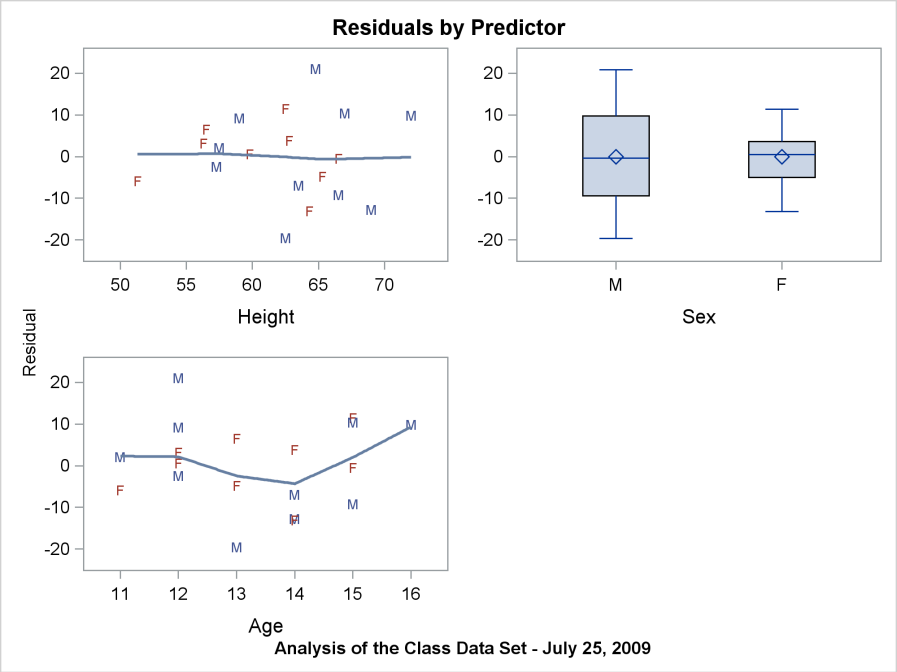
The label "Residual" is printed on the left and rotated 90 degrees to print vertically. An additional sidebar that contains ALIGN=BOTTOM is used to produce a note at the bottom of the display, although you could have done this more easily by using an ENTRYFOOTNOTE statement. Sidebar information can be placed at the top, bottom, left, and right of the graph. Additional options in this template include ROWGUTTER=20 and COLUMNGUTTER=20 in the LAYOUT LATTICE state-

ment to place 20 pixels of white space between graphs. The options **`xAxisOpts=(OffsetMin=0.1 OffsetMax=0.1) yAxisOpts=(OffsetMin=0.1 OffsetMax=0.1)`** add white space (about 10%) between the extreme data values and the axes, which for some plots improves the appearance of the display.

The following step creates the plot, which is displayed in Figure 2.13:

```
proc sgrender data=res template=res;
run;
```

Figure 2.13 Common Axis Label



The following PROC TEMPLATE step replaces the common axis label in the sidebar with a common label for each row of the plot and uses the tick marks and tick labels from the plot on the left for the plot on the right as well:

```
proc template;
   define statgraph res;
      begingraph;
         entrytitle 'Residuals by Predictor';
         layout lattice / rows=2 columns=2 rowgutter=20
                          rowdatarange=unionall;
            rowaxes;
               rowaxis / label="Residual";
               rowaxis / label="Residual";
            endrowaxes;
            layout overlay / xaxisopts=(offsetmin=0.1 offsetmax=0.1)
                             yaxisopts=(offsetmin=0.1 offsetmax=0.1);
               scatterplot y=r x=height / markercharacter=sex group=sex;
               loessplot   y=r x=height;
            endlayout;
```

```
            layout overlay;
               boxplot y=r x=sex;
            endlayout;

            layout overlay / xaxisopts=(offsetmin=0.1 offsetmax=0.1)
                             yaxisopts=(offsetmin=0.1 offsetmax=0.1);
               scatterplot y=r x=age / markercharacter=sex group=sex;
               loessplot   y=r x=age;
            endlayout;
         endlayout;
      endgraph;
   end;
run;
```

The LAYOUT LATTICE statement has the option ROWDATARANGE=UNIONALL, which creates a common Y axis. The following statements create the labels for the common axes:

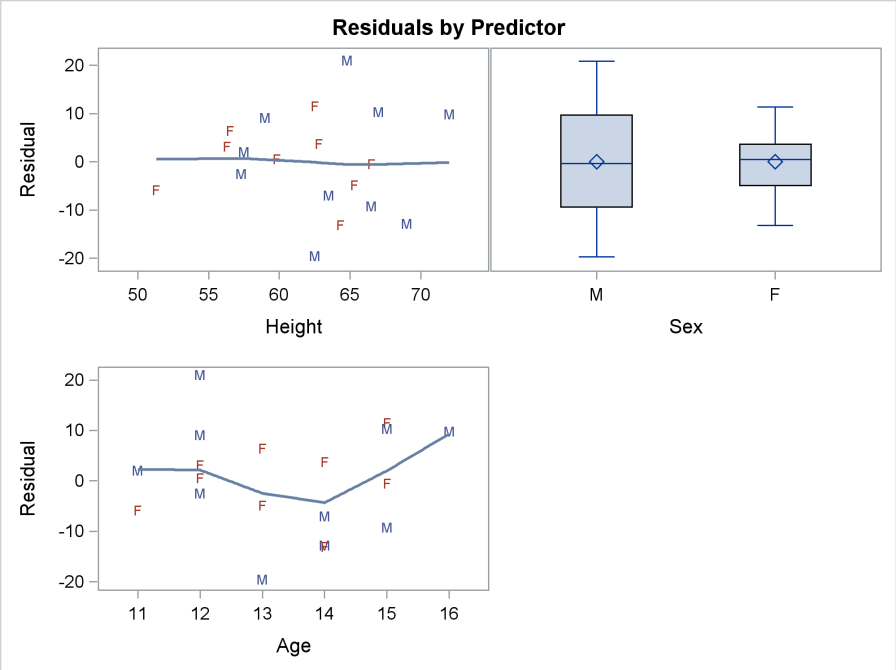
```
rowaxes;
   rowaxis / label="Residual";
   rowaxis / label="Residual";
endrowaxes;
```

There is one ROWAXIS statement for each row. Similarly, there are COLUMNAXES, COLUMNAXIS, and ENDCOLUMNAXES statements, although they are not shown here.

The following step creates the plot, which is displayed in Figure 2.14:

```
proc sgrender data=res template=res;
run;
```

Figure 2.14 Common Axes



The last part of this example illustrates a difference between the lattice layout and the gridded layout. The following two templates are similar to the template used to make the panel in Figure 2.9, but the loess fits and all extra options and statements have been removed. Also, the box plot has been moved from the first row to the second row. The following two templates differ only in that the first uses a LAYOUT LATTICE statement, as before, and the second template instead uses a LAYOUT GRIDDED statement:

```
proc template;
   define statgraph res1;
      begingraph;
         entrytitle 'Residuals by Predictor';
         layout lattice / rows=2 columns=2;
            scatterplot y=r x=height / markercharacter=sex group=sex;
            scatterplot y=r x=age    / markercharacter=sex group=sex;
            boxplot y=r x=sex;
         endlayout;
      endgraph;
   end;

   define statgraph res2;
      begingraph;
         entrytitle 'Residuals by Predictor';
         layout gridded / rows=2 columns=2;
            scatterplot y=r x=height / markercharacter=sex group=sex;
            scatterplot y=r x=age    / markercharacter=sex group=sex;
            boxplot y=r x=sex;
         endlayout;
      endgraph;
   end;
run;
```

The following steps use these templates to create the panels in Figure 2.15 and Figure 2.16:

```
proc sgrender data=res template=res1;
   label r = 'Residual';
run;

proc sgrender data=res template=res2;
   label r = 'Residual';
run;
```

Figure 2.15 Lattice Layout

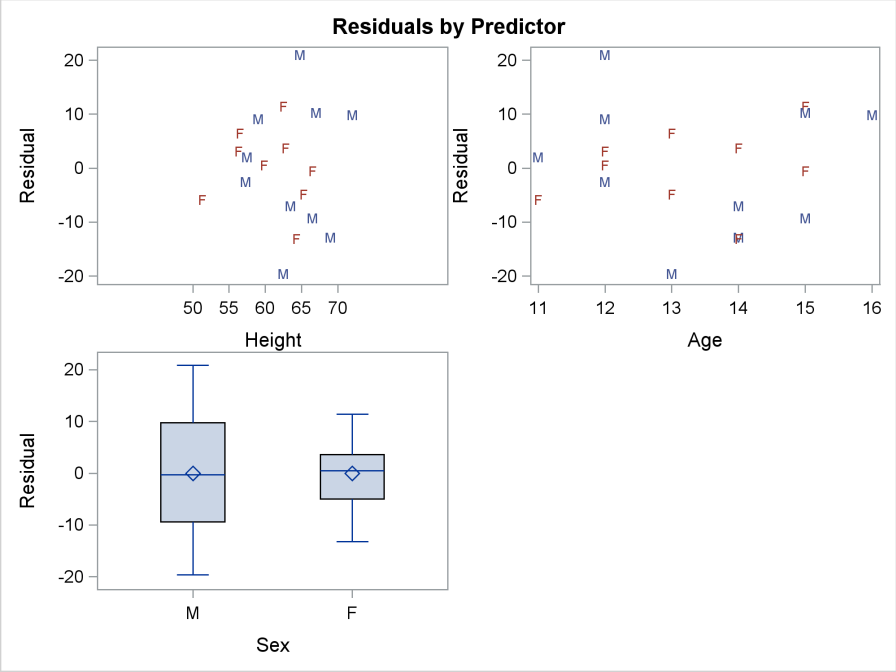
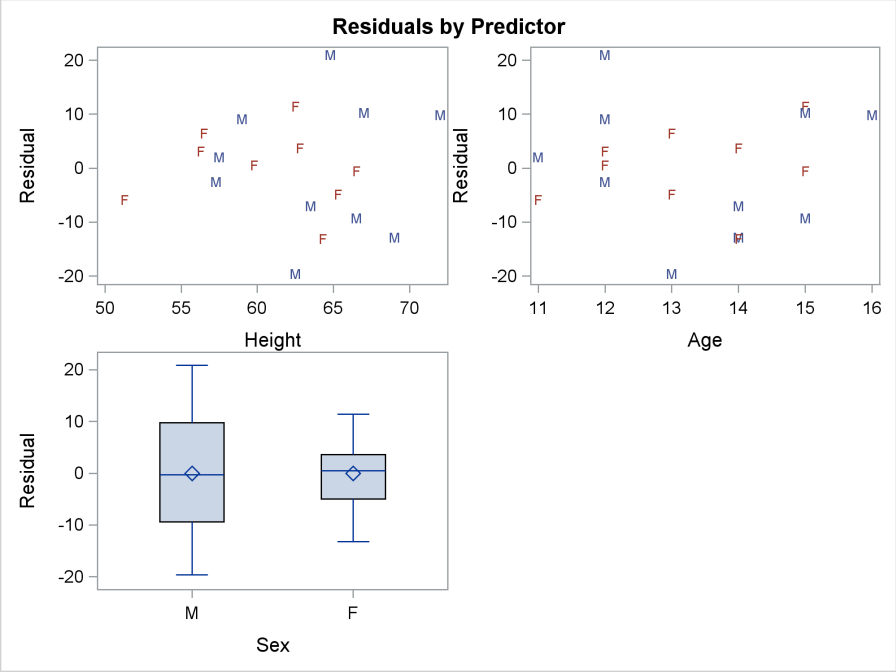


Figure 2.16 Gridded Layout



The lattice layout aligns the plot and tick display areas across graphs to facilitate comparisons. The box plot in Figure 2.15 and the scatter plot of residuals by height have a common tick display area. In contrast, when you use the gridded layout in Figure 2.16, the axes are independent. In many cases, the alignment that the lattice layout provides is preferable. In some cases it is not, and you should use the gridded layout instead.

2.3 Data Panel

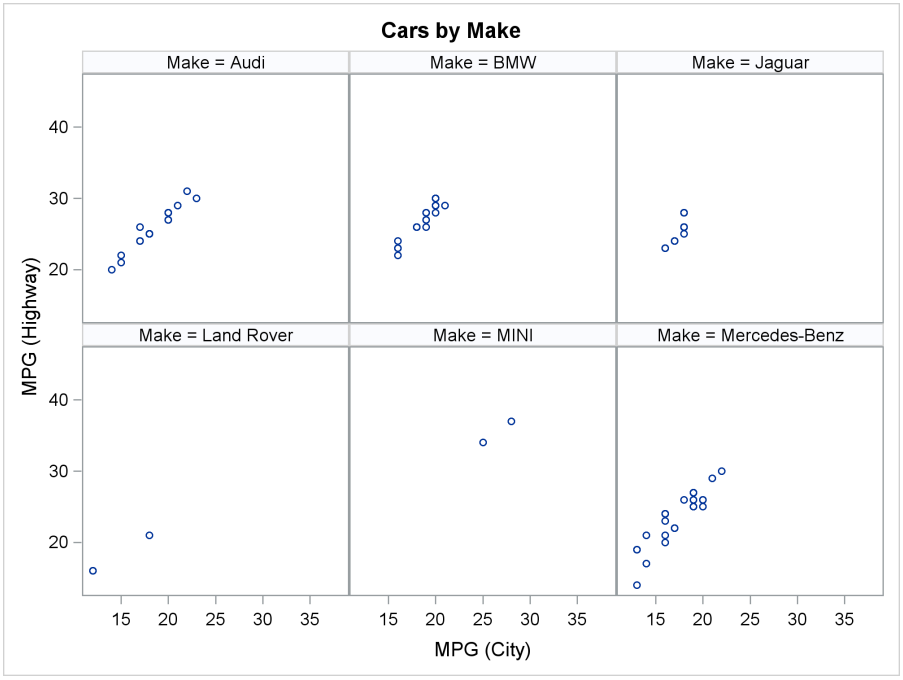
Double-Click for Example Code

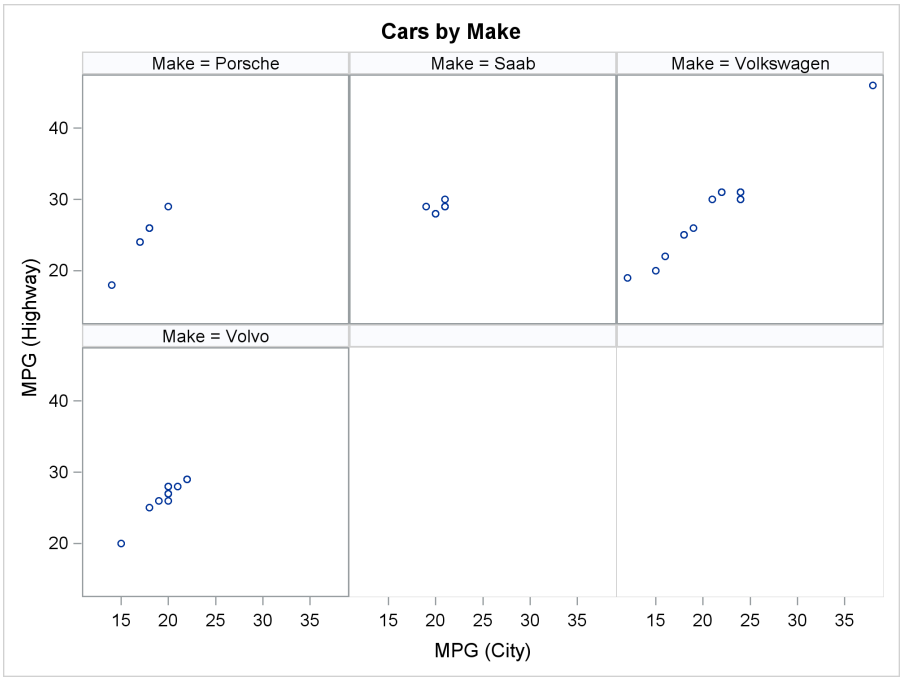
A data panel is a matrix of graphs that has one graph for each level of a classification variable (or for each combination of levels of two or more classification variables). The following steps produce a data panel of scatter plots by using PROC SGPANEL and also the GTL and PROC SGRENDER:

```
proc sgpanel data=sashelp.cars(where=(origin='Europe'));
   title 'Cars by Make';
   panelby make / rows=2 columns=3;
   scatter x=mpg_city y=mpg_highway;
run;

proc template;
   define statgraph panel;
      begingraph;
         entrytitle 'Cars by Make';
         layout datapanel classvars=(make) / rows=2 columns=3;
            layout prototype;
               scatterplot x=mpg_city y=mpg_highway;
            endlayout;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.cars(where=(origin='Europe')) template=panel;
run;
```

The PANELBY statement in PROC SGPANEL specifies the classification variable Make and the number of rows and columns. The SCATTER statement produces the scatter plot. The two graphs that PROC SGPANEL produces are identical to the two graphs produced by the GTL and PROC SGRENDER. One pair of graphs is displayed in Figure 2.17.

Figure 2.17 Data Panel





The classification variable Make has 10 levels for this subset of the data set. Data panels that contain six plots (two rows and three columns) are requested, so each step produces two panels, one that contains the first six graphs and one that contains the remaining four graphs. Each graph has the same axes and same ticks as every other graph both within and across panels. Within a panel, plots in the same row share a common Y axis and plots in the same column share a common X axis. The levels of the classification variable appear as headers above each graph.

In the GTL, the data panel is specified by a LAYOUT DATAPANEL statement, one or more classification variables are specified in the CLASSVARS= option, and the numbers of rows and columns are specified in the ROWS= and COLUMNS= options. You must specify a single LAYOUT PROTOTYPE block within the LAYOUT DATAPANEL block. One or more plotting statements (in this case a SCATTERPLOT statement) are specified inside the LAYOUT PROTOTYPE block. These statements define the common graph elements that are placed inside each cell of the panel. Each graph is constructed in the same way, but for a different level of the classification variable.

In contrast, a LAYOUT LATTICE statement that has two rows and columns is used in the section "Residual Panel" on page 148. When you use the lattice layout, four plot specifications are required because the template produces a panel that contains four graphs. Also, when you use the data panel layout, one Y-axis variable and one X-axis variable are plotted for different levels of a classification variable. In contrast, in the lattice layout, a different variable can potentially appear on each axis of each plot, there is no classification variable, and each graph in the panel can be of a different type from every other graph.

Classification variable levels in graphs that are created by the GTL and PROC SGRENDER appear in the order in which the levels of the classification variable appear in the data set. Because the data set is sorted by the variable Make, the levels appear in sorted order. Classification variable levels in graphs that are created by PROC SGPANEL appear in the sorted order of the levels of the classification variable, because that is what PROC SGPANEL does. If you want any order other than the sorted order, you must use the GTL and PROC SGRENDER and put the data in the order that you want. You do not need to sort the entire data set in that order; you need only the first few observations to be in the right order. One way to control the order is to create new observations at the beginning of the data set that contain missing values in all the variables except the classification variable. Create one observation for each level and insert them in the order in which you want the graphs to appear. For example, you could do the following to sort by descending size:

```
data cars;
   if _n_ = 1 then do;
      type = 'Truck   '; output;
      type = 'SUV     '; output;
      type = 'Wagon   '; output;
      type = 'Sedan   '; output;
      type = 'Sports  '; output;
      type = 'Hybrid  '; output;
   end;
   set sashelp.cars;
   output;
run;
```

The next three examples use the Sashelp.Cars data set and two classification variables. The first example uses PROC SGPANEL, the second uses a LAYOUT DATAPANEL statement, and the third uses a LAYOUT DATALATTICE statement. Here are examples of the LAYOUT DATAPANEL and LAYOUT DATALATTICE statements:

```
layout datapanel   classvars=(cylinders type)      / rows=2 columns=3;
layout datalattice columnvar=type rowvar=cylinders / rows=2 columns=3;
```

They differ in that the LAYOUT DATAPANEL statement does not assign specific row or column roles to the classification variables. In contrast, the LAYOUT DATALATTICE statement specifies the variable Type in the COLUMNVAR= option and the variable Cylinders in the ROWVAR= option.

The following step creates a data panel display by using PROC SGPANEL and creates the panels in Figure 2.18:

```
proc sgpanel data=sashelp.cars(where=(cylinders in (4, 6)));
   title 'Cars by Cylinders and Type';
   panelby cylinders type / rows=2 columns=3 sparse;
   scatter x=mpg_city y=mpg_highway;
run;
```

Figure 2.18 Data Panel with PROC SGPANEL

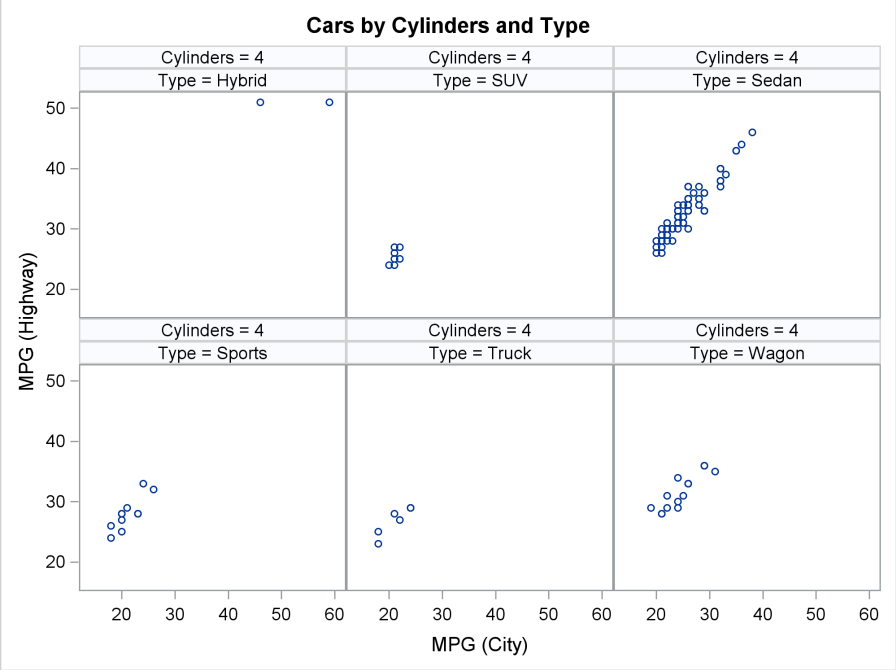
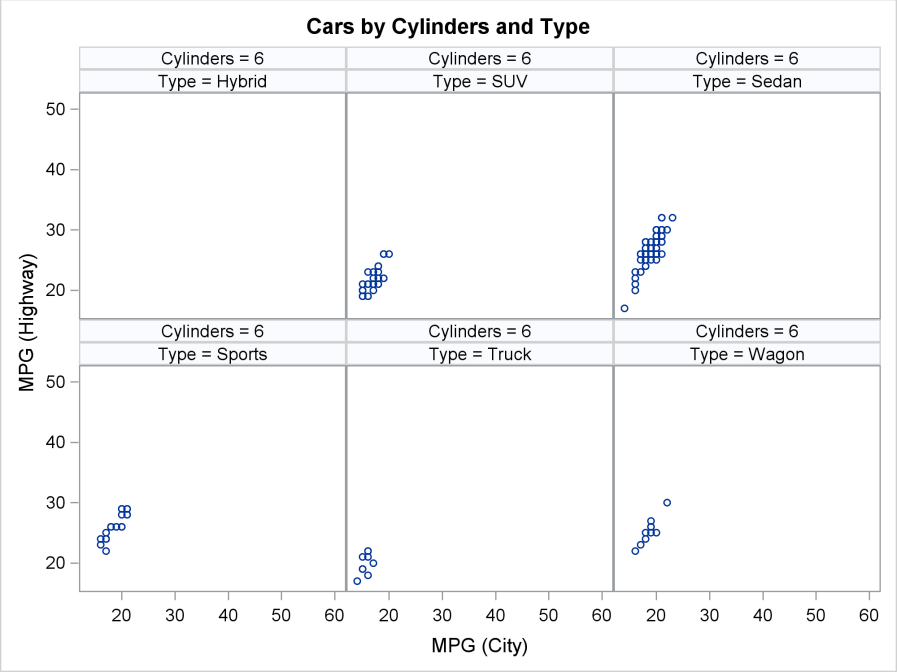


Figure 2.18 *continued*

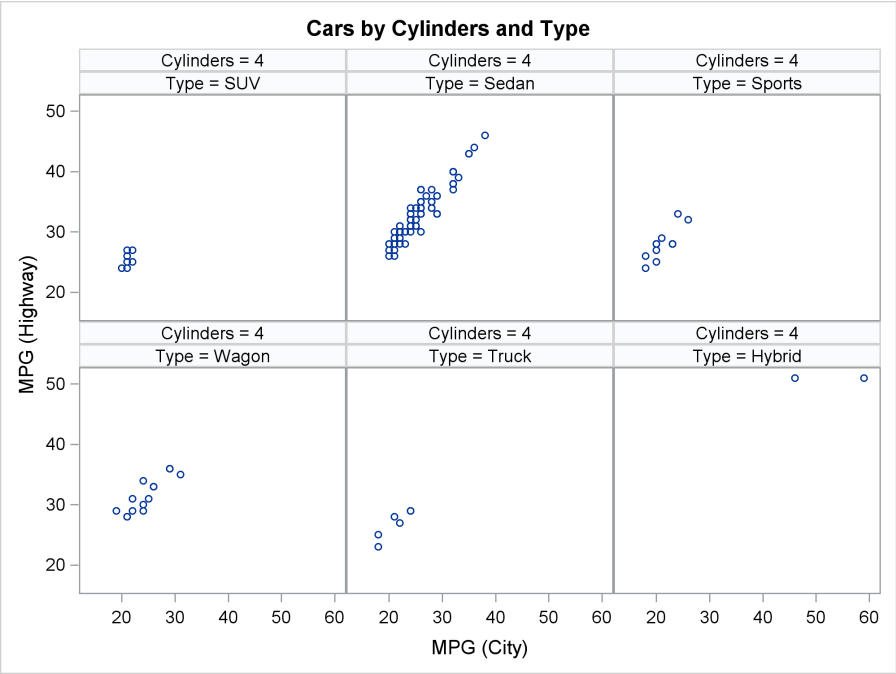


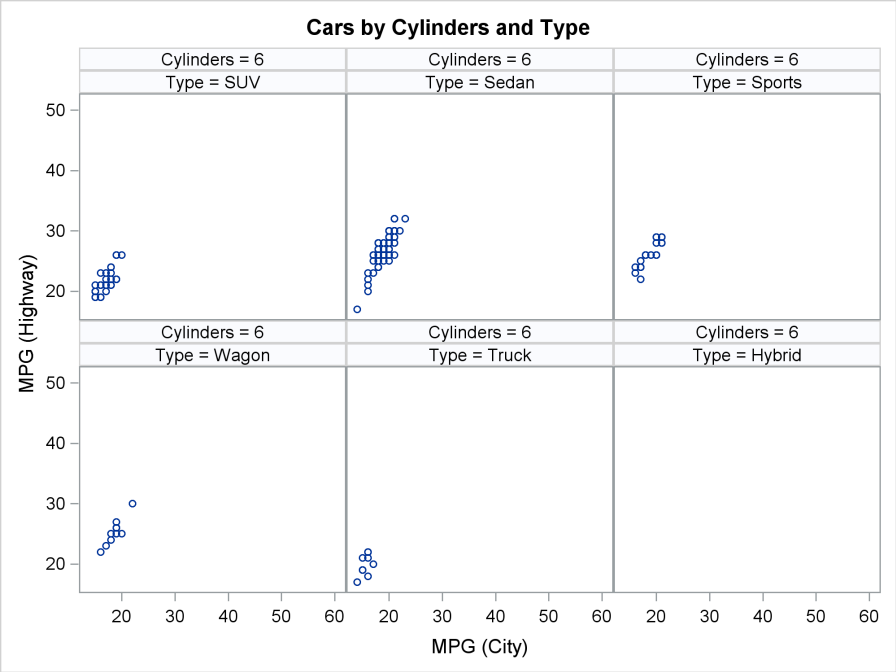
The following steps use the LAYOUT DATAPANEL statement and PROC SGRENDER and create the panels in Figure 2.19:

```
proc template;
   define statgraph panel;
      begingraph;
         entrytitle 'Cars by Cylinders and Type';
         layout datapanel classvars=(cylinders type) /
                          rows=2 columns=3 sparse=true;
            layout prototype;
               scatterplot x=mpg_city y=mpg_highway;
            endlayout;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.cars(where=(cylinders in (4, 6)))
              template=panel;
run;
```

The syntax for the preceding SGPANEL and LAYOUT DATAPANEL statement examples is very similar to the previous data panel examples. However, now there are two classification variables in each list instead of one. The only other difference is the addition of the SPARSE=TRUE option to the LAYOUT DATAPANEL statement in the template and the SPARSE option to the PANELBY statement in PROC SGPANEL. These options enable the procedures to create empty cells for combinations of the classification variables that are not present in the input data set. By default, empty cells are not created. The effect of the SPARSE options is shown by the "Cylinders=6 Type=Hybrid" empty plot.

Figure 2.19 Data Panel with LAYOUT DATAPANEL





The following steps create plots for the combinations of the same two classification variables, but this time by using the LAYOUT DATALATTICE statement in the GTL:

```
proc template;
   define statgraph lattice;
      begingraph;
         layout datalattice columnvar=type rowvar=cylinders /
                            rows=2 columns=3;
            layout prototype;
               scatterplot x=mpg_city y=mpg_highway;
            endlayout;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.cars(where=(cylinders in (4, 6)))
              template=lattice;
run;
```

The results are displayed in Figure 2.20. In Figure 2.19, which uses the LAYOUT DATAPANEL statement, each graph has a header that displays the names and levels of the two classification variables. In Figure 2.20, which uses the LAYOUT DATALATTICE statement, each graph has a header that displays the name of the variable Type and its level and a row header that displays the name of the variable Cylinders and its level. Recall that the LAYOUT DATAPANEL statement syntax does not assign specific row or column roles to the classification variables but the LAYOUT DATALATTICE statement does assign roles. All the graphs have the same axes.

Figure 2.20 Data Panel with LAYOUT DATALATTICE

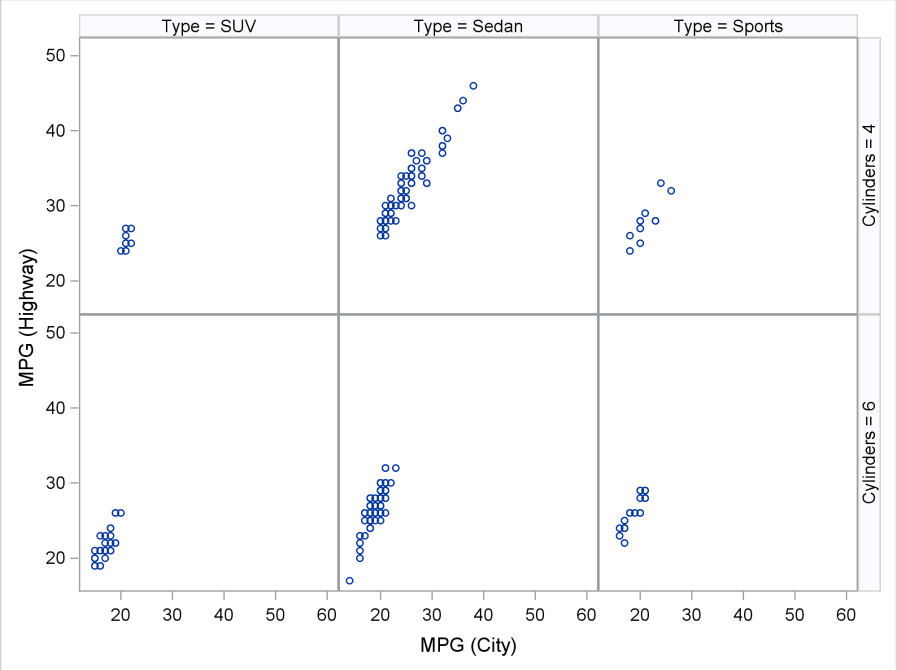
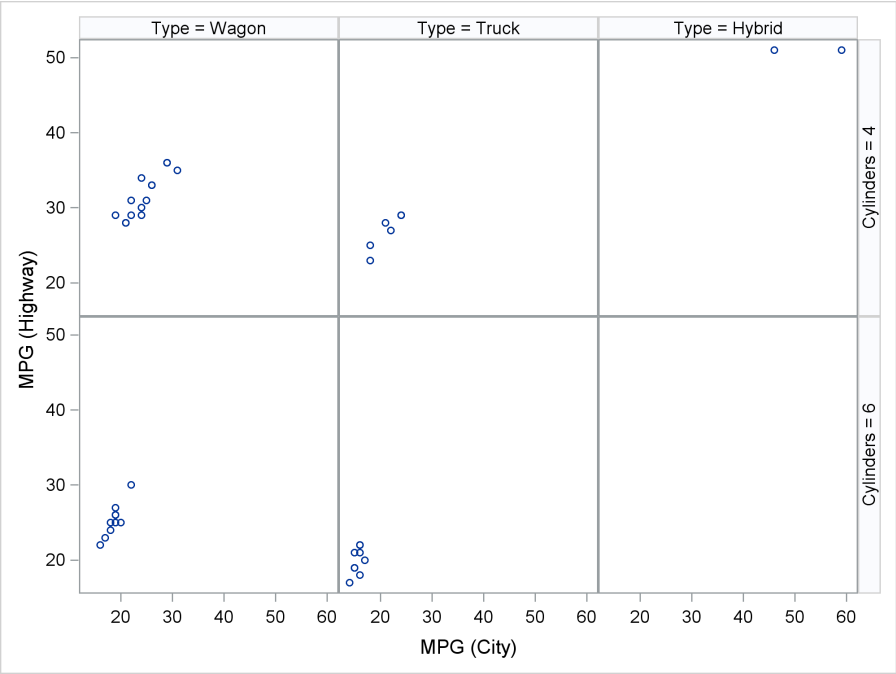


Figure 2.20 *continued*



When you have one classification variable, and you want one row of panels in your graphical display, you can use the LAYOUT DATALATTICE statement. When you have one classification variable, and you want more than one row of panels in your graphical display (as in Figure 2.17), you cannot use the LAYOUT DATALATTICE statement and must use PROC SGPANEL or the LAYOUT DATAPANEL statement instead. When you have two classification variables, and you want the row and column header effect that is displayed in Figure 2.20, use the LAYOUT DATALATTICE statement. In most other cases, you will want to use PROC SGPANEL or the LAYOUT DATAPANEL statement instead.

You can use PROC SGPANEL to get the same row and column headers as you get when you use the LAYOUT DATALATTICE statement by specifying the LAYOUT=LATTICE option in the PANELBY statement as follows:

```
proc sgpanel data=sashelp.cars(where=(cylinders in (4, 6)));
   title 'Cars by Type and Cylinders';
   panelby type cylinders / rows=2 columns=3 sparse layout=lattice;
   scatter x=mpg_city y=mpg_highway;
run;
```

The results of this step are not shown.

The following steps create a scatter plot of the iris data for each of the three species:

```
proc sgpanel data=sashelp.iris;
   title 'Fisher Iris Data';
   panelby species;
   scatter x=petallength y=petalwidth;
run;
```

```
proc template;
   define statgraph panel;
      begingraph / designheight=defaultdesignwidth;
         entrytitle 'Fisher Iris Data';
         layout datapanel classvars=(species) / rows=2 columns=2
                          rowdatarange=union columndatarange=union;
            layout prototype;
               scatterplot x=petallength y=petalwidth;
            endlayout;
         endlayout;
      endgraph;
   end;
run;

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

The results are displayed in Figure 2.21 and Figure 2.22. In the template, the options **RowDataRange=union ColumnDataRange=union** are specified to enable ODS Graphics to create separate axis scales for each row and each column. The option in the PANELBY statement that controls row and column axis scaling is the UNISCALE= option, and it is not used in this example. The differences between Figure 2.21 and Figure 2.22 occur because of the specification of the scaling options.

You should also note that computed plot statements (for example, BOXPLOT, DENSITYPLOT, ELLIPSE, HISTOGRAM, LOESSPLOT, MODELBAND, PBSPLINEPLOT, and REGRESSIONPLOT) cannot be used inside the LAYOUT PROTOTYPE block that is inside the LAYOUT DATAPANEL and LAYOUT DATALATTICE blocks. The equivalent statements are supported in PROC SGPANEL.

Figure 2.21 Data Panel with PROC SGPANEL

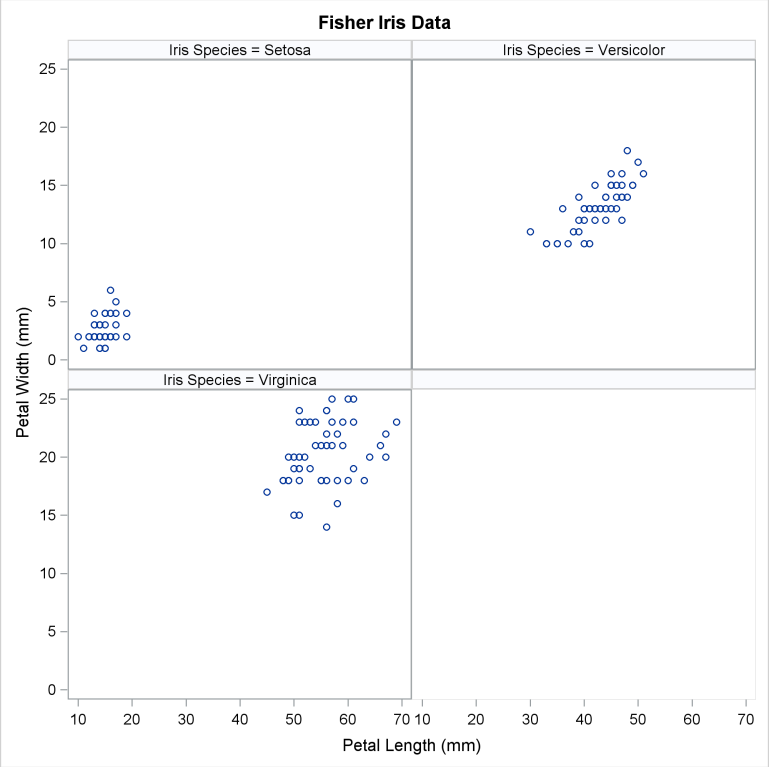
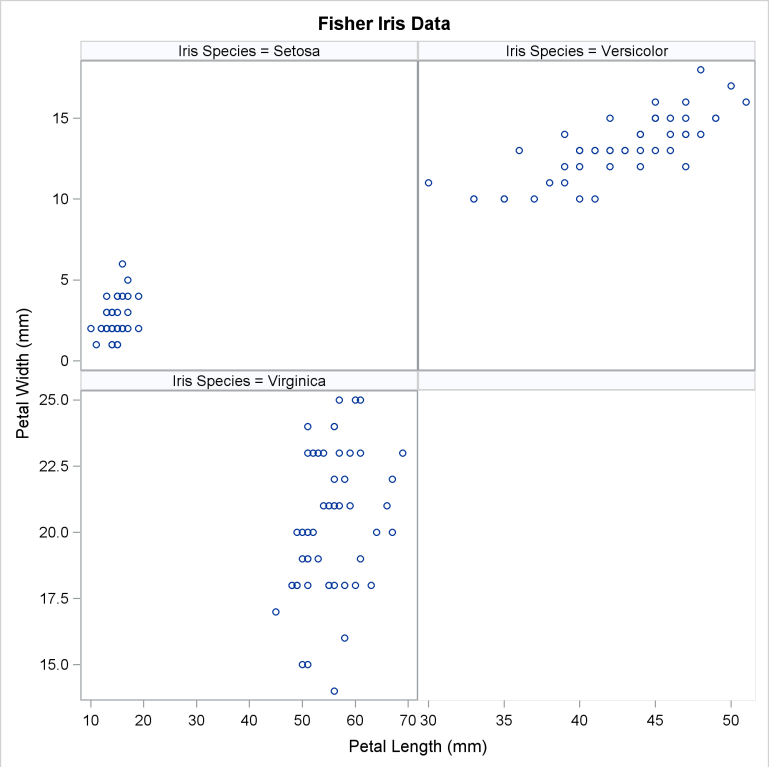


Figure 2.22 Data Panel with the GTL



Chapter 3
Style Templates

Double-Click for Example Code

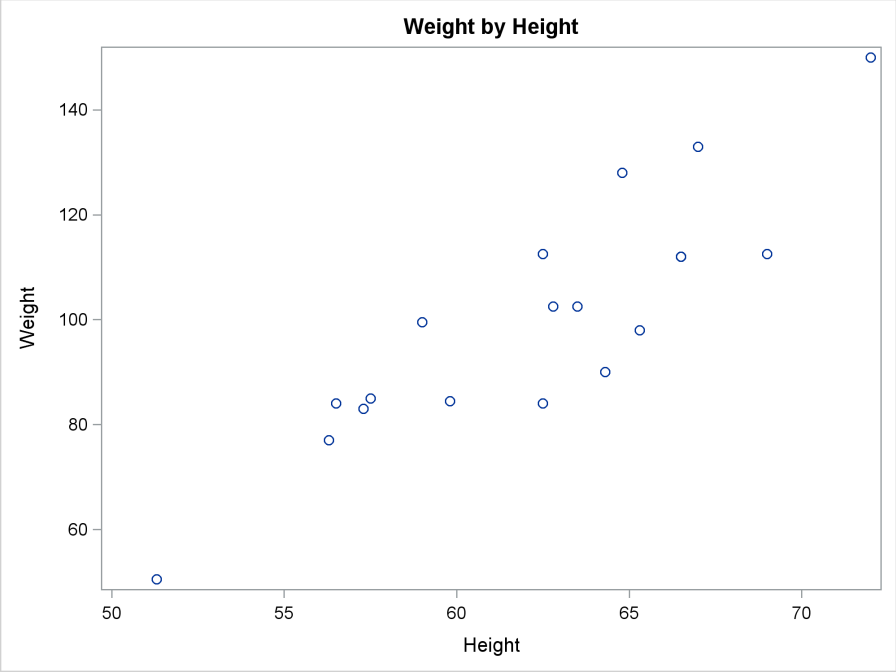
(Code links work in Adobe Reader and Internet Explorer.)

ODS styles control the colors and general appearance of all graphs and tables. Styles are composed of style elements that control specific aspects of graphs (for example, the font of a title, the color of a reference line, the style of a regression fit line, and so on). Graphs are constructed from multiple components: a data object (in the case of an analytical procedure) or a SAS data set (in the case of PROC SGRENDER or an SG procedure), dynamic and macro variables (optional), a graph template, and a style template. The following steps create a scatter plot from a graph template called **`ClassScatter`**, a style called HTMLBlue (which has a style template called **`Styles.HTMLBlue`**), and a SAS data set called Sashelp.Class:

```
proc template;
   define statgraph classscatter;
      begingraph;
         entrytitle 'Weight by Height';
         layout overlay;
            scatterplot y=weight x=height / markerattrs=GraphDataDefault;
         endlayout;
      endgraph;
   end;
run;

ods html style=htmlblue;
proc sgrender data=sashelp.class template=classscatter;
run;
```

The results are displayed in Figure 3.1. The appearance of the markers in the scatter plot is controlled by the **`GraphDataDefault`** style element in the HTMLBlue style. (This is the default style element even when the MARKERATTRS=GRAPHDATADEFAULT option is not specified.)

Figure 3.1 Original `GraphDataDefault` Style Element



The definition of a style is provided by a style template. The following step lists the names of all style templates:

```
proc template;
   list styles;
run;
```

The results of this step are not displayed, but a few of the style templates in the list include **Styles.HTMLBlue**, **Styles.Journal**, **Styles.Journal1a**, **Styles.Journal2**, **Styles.Journal2a**, **Styles.Journal3**, **Styles.Journal3a**, **Styles.Pearl**, and **Styles.PearlJ**. These templates define the styles that are most often used for statistical work. The HTMLBlue style is the default style for graphs, and the PearlJ style is the default style for tables in both SAS/STAT documentation and this book.

The following step displays the HTMLBlue style template:

```
proc template;
   source styles.htmlblue / expand;
run;
```

Here are few lines from the results:

```
define style Styles.Htmlblue;
   parent = styles.statistical;
   . . .
end;

define style styles.statistical;
   parent = styles.default;
   . . .
end;
```

```
define style styles.default;
   . . .
end;

define style Base.Template.Style;
   . . .
end;
```

The HTMLBlue style inherits many of its properties from other styles. The parent of the HTMLBlue style is the Statistical style, and the parent of the Statistical style is the Default style. **Base.Template.Style** implicitly provides the foundation for all other styles. You can search the style listing for style elements (the components that make up a style) and modify them. For example, the following steps display the definition of the **GraphDataDefault** style element:

```
proc template;
   source styles.htmlblue / expand file='temp.tmp';
run;

data _null_;
   length style $ 24;
   retain style;
   infile 'temp.tmp';
   input;
   if index(_infile_, 'define style') then style = scan(_infile_, 3, ' ;');
   if index(_infile_, 'class GraphDataDefault') then do;
      put // '**** Style = ' style /;
      gotit + 1;
      end;
   if gotit then put _infile_;
   if gotit and index(_infile_, ';') then gotit = 0;
run;
```

The results are displayed in Figure 3.2. The **GraphDataDefault** style element is defined in the Default style, and the HTMLBlue and Statistical styles inherit its definition.

Figure 3.2 **GraphDataDefault** Style Element

```
**** Style = styles.default

   class GraphDataDefault /
      endcolor = GraphColors('gramp3cend')
      neutralcolor = GraphColors('gramp3cneutral')
      startcolor = GraphColors('gramp3cstart')
      markersize = 7px
      markersymbol = "circle"
      linethickness = 1px
      linestyle = 1
      contrastcolor = GraphColors('gcdata')
      color = GraphColors('gdata');
```

The following steps display the definitions of the **RowHeader** style element:

```
data _null_;
   length style $ 24;
   retain style;
   infile 'temp.tmp';
   input;
   if index(_infile_, 'define style') then style = scan(_infile_, 3, ' ;');
   if index(lowcase(_infile_), 'class rowheader ') or
      index(lowcase(_infile_), 'style rowheader ') then do;
      put // '**** Style = ' style /;
      gotit + 1;
      end;
   if gotit then put _infile_;
   if gotit and index(_infile_, ';') then gotit = 0;
run;
```

The results are displayed in Figure 3.3. The **RowHeader** style element is defined in multiple styles. It is common for a style element to be defined in one place, and it is also common for it to be defined in more than one place.

Figure 3.3 **RowHeader** Style Element

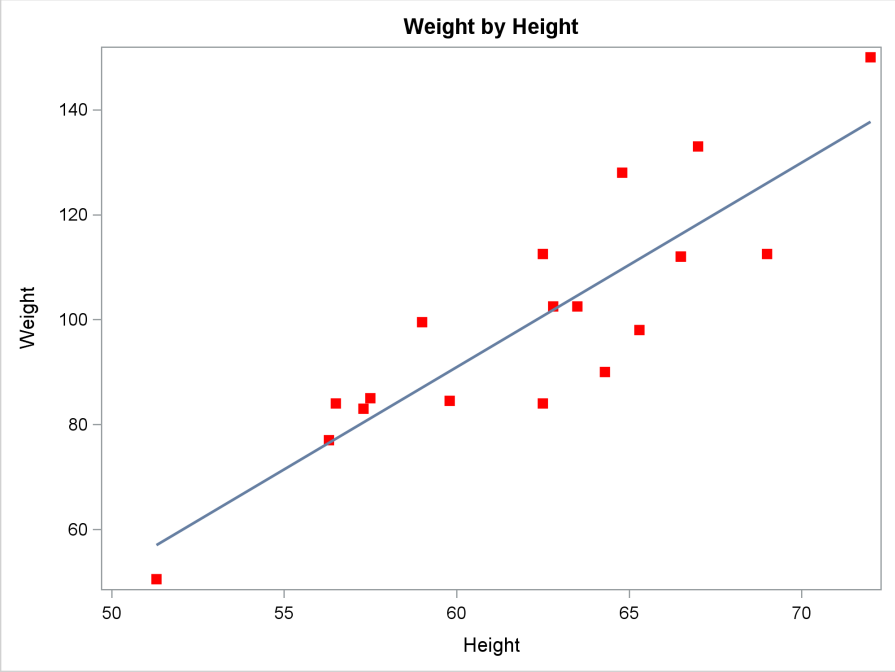
```
**** Style = Styles.Htmlblue

   class RowHeader /
      bordercolor = cxB0B7BB
      backgroundcolor = cxEDF2F9
      color = cx112277;


**** Style = styles.statistical

   style RowHeader /
      font = fonts('HeadingFont')
      backgroundcolor = cxF5F7F1
      bordercolor = cxC1C1C1
      bordertopwidth = 0px
      borderleftwidth = 0px
      borderbottomwidth = 1px
      borderrightwidth = 1px
      borderstyle = solid;


**** Style = Base.Template.Style

   style RowHeader from Header
      "Controls row headers.";
```

You can redefine and use the **GraphDataDefault** style element as follows:

```
proc template;
   define style styles.mystyle;
      parent=styles.htmlblue;
      class GraphDataDefault /
         endcolor = GraphColors('gramp3cend')
         neutralcolor = GraphColors('gramp3cneutral')
         startcolor = GraphColors('gramp3cstart')
         markersize = 7px
         markersymbol = "squarefilled"
         linethickness = 1px
         linestyle = 1
         contrastcolor = red
         color = GraphColors('gdata');
   end;
run;

ods html style=mystyle;
proc sgplot data=sashelp.class noautolegend;
   title 'Weight by Height';
   reg y=weight x=height;
run;
ods html close;
```

This style inherits from the HTMLBlue style, and differs from HTMLBlue in only that one style element definition. The results are displayed in Figure 3.4. The marker color is changed from blue to red, and the marker is changed from an open circle to a filled square.

Figure 3.4 Redefined **GraphDataDefault** Style Element



In the early days of ODS Graphics, understanding how to redefine style elements was an important basic topic. Today, the STYLEATTRS statement and new GTL options (described in Chapter 1, "The Graph Template Language and the SG Procedures") make that less necessary than it once was. Still, it is important that you understand how to choose and modify ODS styles if you want to perform advanced customization of your graphs. Basic and advanced examples of ODS styles and style modification are discussed in "ODS Styles" (Chapter 21, *SAS/STAT User's Guide*), so more information is not included here. That chapter includes the following sections:

- "An Overview of ODS Styles"
- "Attribute Priorities"
- "Overriding How Groups Are Distinguished"
- "ODS Style Elements and Attributes"
- "Style Templates and Colors"
- "Some Common ODS Style Elements"
- "ODS Style Comparisons"
- "Modifying the HTMLBLUE Style"
- "ODS Style Template Modification Macro"
- "Varying Colors and Markers but Not Lines"
- "Changing the Default Markers and Lines"
- "Modifying Graph Fonts in Styles"
- "Modifying Other Graph Elements in Styles"
- "Changing the Default Style"

Additional information about styles can be found in the section "Style Templates" (Chapter 23, *SAS/STAT User's Guide*).

Chapter 4
Graph Template Modification

Double-Click for Example Code

(Code links work in Adobe Reader and Internet Explorer.)

This chapter shows how to modify a template that SAS provides. The templates that are supplied by SAS for statistical procedures are often lengthy and complex, because they provide ODS Graphics with comprehensive and detailed information about graph construction. They contain many of the same statements that you see in the previous examples. However, they often contain statements that do not appear in the previous examples. You usually do not need certain statements when you write your own templates, but they are important for the templates that SAS provides. The template in this example was chosen because it is small and easy to modify.

You can begin by submitting the following statements:

```
ods path (prepend) work.templat(update);
ods graphics on;
ods trace on;
```

The ODS PATH statement stores the modified templates in the Work library so that they are deleted at the end of your SAS session. This is not required. By default, modified templates are stored in the Sasuser library until they are deleted. The ODS GRAPHICS statement enables ODS Graphics so that graphs are automatically produced by the analytical procedures. The ODS TRACE statement enables ODS trace output so that information about each graph and table (including the graph or table name and the template name) is displayed in the SAS log.

This example shows how to change titles and axis labels. The following step runs PROC FACTOR and produces the eigenvalue (or scree) plot displayed in Figure 4.1:

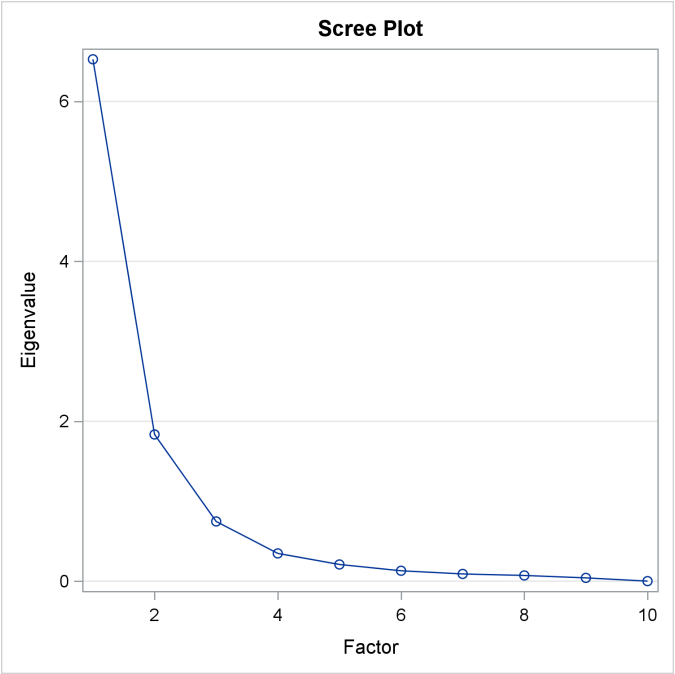
```
proc factor data=sashelp.cars plots(unpack)=scree;
run;
```

The **plots(unpack)=scree** option produces the scree plot by itself—"unpacked" from its usual location as part of a two-graph panel with the variance-explained plot.

The ODS trace output for the scree plot is as follows:

```
Name:        ScreePlot
Label:       Scree Plot
Template:    Stat.Factor.Graphics.ScreePlot1
Path:        Factor.InitialSolution.ScreeAndVarExp.ScreePlot
```

Figure 4.1 Default Scree Plot



The following statements display the graph template for the scree plot:

```
proc template;
   source Stat.Factor.Graphics.ScreePlot1;
run;
```

The template source statements are displayed in Figure 4.2.

Figure 4.2 Scree Plot Template

```
define statgraph Stat.Factor.Graphics.ScreePlot1;
   notes "Scree Plot for Extracted Eigenvalues";
   dynamic _byline_ _bytitle_ _byfootnote_;
   BeginGraph / designwidth=DefaultDesignHeight;
      Entrytitle "Scree Plot" / border=false;
      Layout overlay / yaxisopts=(label="Eigenvalue" gridDisplay=auto_on)
         xaxisopts=(label="Factor" linearopts=(integer=true));
         seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   EndGraph;
end;
```

The DESIGNWIDTH=DEFAULTDESIGNHEIGHT option in the BEGINGRAPH statement makes a square outer box that contains the graph; its width is set equal to the default height. This creates a graph that is designed to be 480 pixels wide by 480 pixels high. The default design size is 640 pixels wide by 480 pixels high. The ENTRYTITLE statement provides the graph title, in this case "Scree Plot". The BORDER=FALSE option displays the title without a border. In fact, this is the default behavior, so the option is unnecessary. However, it is not unusual to see default specifications in the templates that SAS provides.

The LAYOUT OVERLAY statement provides the label "Eigenvalue" for the Y axis, provides the label "Factor" for the X axis, produces grid lines for the Y axis when the output style favors grids, and creates integer tick marks on the X axis. The LINEAROPTS= option specifies options for standard axes that depict a linear scaling (in contrast to LOGOPTS=, which is used for log-scale axes).

The graph is a series plot. The Y-axis column in the ODS data object is named EigenValue, and the X-axis column in the ODS data object is named Number (the factor number). The standard series plot display is a series of lines, but the DISPLAY=ALL option also displays the markers (in this case, circles) for the data values.

Notice that the title and the axis labels are all specified directly as literal character strings in this template. You can change any of them and submit the results to SAS. From then on, until you change or delete your custom template in Work.Templat or until you end your SAS session, you will see your customization whenever you run PROC FACTOR.

The following example adds a PROC TEMPLATE statement and a RUN statement, changes the title and an axis label, specifies explicit tick values, and removes the grid and the unnecessary BORDER= option:

```
proc template;
   define statgraph Stat.Factor.Graphics.ScreePlot1;
      notes "Scree Plot for Extracted Eigenvalues";
      dynamic _byline_ _bytitle_ _byfootnote_;
      BeginGraph / designwidth=DefaultDesignHeight;
         Entrytitle "Eigenvalue ((*ESC*){Unicode Lambda}) Plot";
         layout overlay / yaxisopts=(label="Eigenvalue")
            xaxisopts=(label="Factor Number"
                       linearopts=(tickvaluelist=(1 2 3 4 5 6 7 8 9 10)));
            seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
         endlayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
run;
```

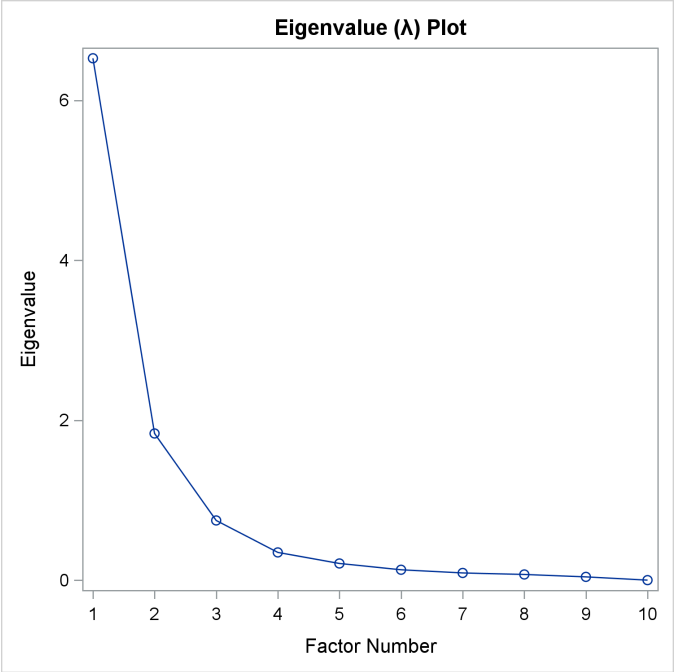
The title now contains the Greek letter λ, which is specified as an escape sequence followed by a Unicode specification. The tick value list is specified in full because the GTL does not accept standard SAS shorthand lists. The only output from this step is the following log note:

```
NOTE: STATGRAPH 'Stat.Factor.Graphics.ScreePlot1' has been saved to: WORK.TEMPLAT
```

The following step uses the new template to create the scree plot displayed in Figure 4.3:

```
proc factor data=sashelp.cars plots(unpack)=scree;
run;
```

Figure 4.3 Modified Scree Plot



The following step restores the default template:

```
proc template;
   delete Stat.Factor.Graphics.ScreePlot1;
run;
```

The only output from this step is the following log note:

```
NOTE: 'Stat.Factor.Graphics.ScreePlot1' has been deleted from: WORK.TEMPLAT
```

Chapter 22, "ODS Graphics Template Modification" (*SAS/STAT User's Guide*), provides detailed examples of graph template modification. The chapter includes the following sections:

- "The Graph Template Language"
- "Locating Templates"
- "Displaying Templates"
- "Editing Templates"
- "Saving Customized Templates"
- "Using Customized Templates"
- "Reverting to the Default Templates"
- "Graph Template Modification Macro"
- "Examples of ODS Graphics Template Modification"
- "Modifying Graph Titles and Axis Labels"
- "Modifying Colors, Line Styles, and Markers"
- "Modifying Tick Marks and Grid Lines"
- "Modifying the Style to Show Grid Lines"
- "Unicode and Special Characters"
- "Adding a Date and Project Stamp to a Few Graphs"
- "Adding Data Set Information to a Graph"
- "Adding a Date and Project Stamp to All Graphs"
- "%Marginal Macro"
- "Understanding Conditional Template Logic"

Chapter 23, "Customizing the Kaplan-Meier Survival Plot" (*SAS/STAT User's Guide*), shows numerous ways to modify the Kaplan-Meier survival plot that is produced by the LIFETEST procedure.

Appendix A
Introduction to ODS Graphics

Contents

You enable ODS Graphics by specifying the following statement:

```
ods graphics on;
```

ODS Graphics remains enabled for all procedure steps until you disable it by submitting the following statement:

```
ods graphics off;
```

After you enable ODS Graphics, creating graphical output from procedures is as simple as creating tabular output. You can control your output as follows:

- ODS destination statements (such as ODS HTML or ODS RTF) specify where you want your graphs to be displayed.
- ODS SELECT and ODS EXCLUDE statements select and exclude graphs from your output.
- ODS OUTPUT statements create SAS data sets from the data object used to make the plot.
- Procedure options specify which graphs to create. For each procedure, these options are described in the Syntax section of the procedure chapter. Usually, you use the PLOTS= option to control all graphs. The available graphs are listed in the ODS Graphics section, which is found in the Details section of each procedure chapter. Many graphs are produced by default.
- ODS styles control the general appearance and consistency of all graphs and tables.
- ODS templates control the layout and details of each graph.

A.1 ODS Destinations

ODS can send your graphs and tables to a number of different destinations, including RTF (rich text format), HTML (hypertext markup language), LISTING (the SAS LISTING destination), DOCUMENT (the ODS document), and PDF (portable document format). You use an ODS statement to open a destination, as in the following examples:

```
ods html body='b.htm';
ods rtf;
ods listing;
ods document name=MyDoc(write);
ods pdf file="contour.pdf";
```

You can close destinations individually or all at once, as in the following examples:

```
ods html close;
ods rtf close;
ods listing close;
ods document close;
ods pdf close;
ods _all_ close;
```

For most ODS destinations (for example, HTML, RTF, and PDF), graphs and tables are integrated in the output, and you view your output by using an appropriate viewer, such as a web browser for HTML. However, the LISTING destination is different. If you are using the LISTING destination in the SAS windowing environment, you view your graphs individually by clicking the graph icons in the Results window. This action invokes a host-dependent graph viewer (for example, Microsoft Photo Editor on Windows). The graphs that ODS Graphics produces are *not* displayed along with traditional graphs in the Graph window.

If you prefer to view integrated output, you should specify a destination such as HTML or RTF. At the same time, you can prevent the Output window from appearing by closing the LISTING destination, as in the following statements:

```
ods listing close;
ods html;
```

A graph is created for every open destination. When you open a new destination, you should close all destinations that you do not need. Closing destinations enables your jobs to run faster and consume fewer resources, because fewer graphs are produced.

A.2 Accessing Individual Graphs

If you are writing a paper or creating a presentation, you need to access your graphs individually. There are various ways to do this, depending on the ODS destination. Three particularly useful methods follow:

- If you are viewing RTF output, you can simply copy and paste your graphs from the viewer into a Microsoft Word document or a Microsoft PowerPoint slide.
- If you are viewing HTML output, you can copy and paste your graphs from the viewer, or you can right-click the graph and save it to a file.
- You can save your graphs in image files and then include them in a paper or presentation. For example, you can save your graphs as PNG files and include them in a paper that you are writing by using LaTeX or include them in an HTML document.

You can specify the graphic image format and the filename in the ODS GRAPHICS statement. For example, the following statements, when submitted before a procedure step that produces multiple graphs, save the graphs in PostScript files named *myname.ps*, *myname1.ps*, and so on:

```
ods listing close;
ods latex;
ods graphics on / imagefmt=ps imagename='myname';
```

If you are using the LISTING destination and the SAS windowing environment, you can also copy from the default viewer into a Microsoft Word document or a Microsoft PowerPoint slide.

A.3 Specifying the Size and Resolution of Graphs

Two factors to consider when you are creating graphs for a paper or presentation are the size of the graph and its resolution. You can specify the size of a graph in the ODS GRAPHICS statement. The following examples show typical ways to change the size of your graphs:

```
ods graphics on / width=6in;
ods graphics on / height=4in;
ods graphics on / width=4.5in height=3.5in;
```

You can change the resolution of a graph by specifying the IMAGE_DPI= option in any ODS destination statement, as in the following example:

```
ods html image_dpi=300;
```

The default resolution of graphs that are created for the HTML and LISTING destinations is 100 DPI (dots per inch), whereas the default for the RTF destination is 200 DPI. An increase in resolution often improves the quality of the graphs, but it also increases the size of the image file.

A.4 PLOTS= Option

Each statistical procedure that produces ODS Graphics has a PLOTS= option that selects graphs and specifies some options. The syntax of the PLOTS= option is as follows:

PLOTS <(*global-plot-options*)> <= *plot-request* <*(options)*>>
PLOTS <(*global-plot-options*)> <= (*plot-request* <*(options)*> <... *plot-request* <*(options)*>>)>

The PLOTS= option has a common overall syntax for all statistical procedures, but the specific global plot options, plot requests, and plot options vary across procedures. There are only a limited number of things that you can control by using the PLOTS= option. Most graphical details are controlled either by graph templates or by styles.

The PLOTS= option is usually specified in the PROC statement. However, for some procedures, certain analyses and hence certain plots can appear only if you specify an additional statement. These procedures often have a PLOTS= option in that additional statement.

The simplest PLOTS= option specifications are of the form PLOTS=*plot-request* or PLOTS=(*plot-requests*). When there is more than one plot request, the *plot-requests* must appear in parentheses. Each plot request either requests a plot (for example, RESIDUALS) or provides you with a place to specify plot-specific options (for example, DIAGNOSTICS(UNPACK)). Some simple and typical plot requests are explained next:

- PLOTS=ALL requests all plots that are relevant to the analysis.
- PLOTS=NONE disables ODS Graphics for only that step.
- PLOTS=RESIDUALS requests a plot of residuals in a modeling procedure such as PROC REG.
- PLOTS=RESIDUALS(SMOOTH) requests the residuals plot along with a smooth fit function.
- PLOTS=(TRACE AUTOCORR) requests trace and autocorrelation plots in procedures that have Bayesian analysis options.

Global plot options appear in parentheses after the option name and before the equal sign. These options affect many or all of the plots. The UNPACK option is a commonly used global plot option. It specifies that plots that are normally produced containing multiple plots per panel (or "packed") be unpacked and appear in multiple panels that contain one plot in each panel. The specification PLOTS(UNPACK)=(*plot-requests*) unpacks all paneled plots. The UNPACK option is also used as an option in a plot request when you want to unpack only certain panels. For example, the option **`plots=(diagnostics(unpack) partial predictions)`** unpacks only the diagnostics panel. In some cases, unpacked plots contain additional information that is not found in the smaller packed versions. The UNPACK option is not available for all plot requests; it is available only for plots that have multiple panels by default.

Another commonly used global plot option is the ONLY option. Many procedures produce default plots, and you can request additional plots in the PLOTS= option. Specifying PLOTS=(*plot-requests*) while omitting the default plots does not prevent the default plots from being produced. The ONLY option is used when you want to see only the plots that appear in the *plot-request*. Procedures that produce no default plots usually do not provide an ONLY option. You can use ODS SELECT and ODS EXCLUDE to select and exclude graphs, but in some situations the ONLY option is more convenient. It is typically more efficient to select plots by using the PLOTS(ONLY)= option, because the procedure does not do extra work to generate a plot that is excluded by the PLOTS(ONLY)= option. In contrast, ODS SELECT and ODS EXCLUDE have their effect after the procedure has done the work to generate the plot.

A.5 Viewing Your Graphs in the SAS Windowing Environment

The mechanism for viewing graphs can vary depending on your operating system, which viewers are installed on your computer, and which ODS destination you have selected. If you do not specify an ODS destination, then the HTML destination is usually the default and graphs are displayed automatically. If you are using the SAS windowing environment and the LISTING destination, go to the Results window and find the icon for the corresponding graph. You can double-click the graph icon to display the graph in the default viewer that is configured on your computer for the corresponding image file type.

If you are using the SAS windowing environment and you specify an HTML destination, then the results are displayed by default in the SAS Results window as they are being generated. Depending on your configuration, this statement can also apply to the PDF and RTF destinations. If you are using the LATEX or PS destination, you must use a PostScript viewer, such as GSview. For information about the windowing environment in a different operating system, see the SAS Companion for that operating system.

If you do not want to view the results as they are being generated, select **Tools ► Options ► Preferences** from the menu at the top of the main SAS window. Then, on the **Results** tab, clear the **View results as they are generated** check box.

A.6 Determining Graph Names and Labels

Procedures assign a name to each graph that ODS Graphics creates. This enables you to refer to ODS graphs in the same way that you refer to ODS tables. You can determine the names of graphs in several ways:

- Look up graph names in the ODS Graphics section of chapters for procedures that use ODS Graphics.
- Use the Results window to view the names of ODS graphs created in your SAS session.
- Use the ODS TRACE ON statement to list the names of graphs created in your SAS session. This statement adds identifying information in the SAS log (or optionally in the SAS LISTING) for each graph that is created.

The graph name is not the same as the name of the image file that contains the graph.

A.7 Default Template Stores and the ODS PATH

SAS stores certain types of information in data sets called item stores. Templates that PROC TEMPLATE has compiled are stored in item stores called template stores. The default table, graph,

and style templates that SAS provides are stored in the template store Sashelp.Tmplmst. By default, if you modify a template, it is stored in the template store Sasuser.Templat. By default, ODS searches Sasuser.Templat for templates, and if it does not find the requested template, it searches Sashelp.Tmplmst. You can see the list of template stores by submitting the following statement:

```
ods path show;
```

Here are the results:

```
Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

With this default path, SAS always finds the templates that it provides in Sashelp.Tmplmst unless you stored a modified template in Sasuser.Templat. In that case, it finds and uses your modified template in Sasuser.Templat. To see a list of all the templates that you have modified, submit the following statements:

```
proc template;
   list / store=sasuser.templat;
run;
```

You can delete any template that you modified (so that ODS finds the default template that SAS supplied) by specifying it in a DELETE statement, as in the following example:

```
proc template;
   delete Stat.REG.Graphics.ResidualPlot;
run;
```

CAUTION: Never set the access for Sashelp.Tmplmst to UPDATE. As long as the access for Sashelp.Tmplmst is READ, ODS never deletes the template in Sashelp.Tmplmst, so you can safely run the preceding step, even if the template that you specify does not exist in Sasuser.Templat. You can run the following step to delete the entire Sasuser.Templat template store of customized templates so that ODS uses only the templates supplied by SAS:

```
ods path sashelp.tmplmst(read);
proc datasets library=sasuser;
   delete templat(memtype=itemstor);
run;
ods path reset;
```

You can restore the default template path in either of the following equivalent ways:

```
ods path sasuser.templat(update) sashelp.tmplmst(read);
ods path reset;
```

It is good practice to delete templates that you have customized when you are done with them, so that they are not unexpectedly used later.

You can modify the path and insert a Work item store before the default path in either of the following equivalent ways:

```
ods path work.templat(update) sasuser.templat(update) sashelp.tmplmst(read);
ods path (prepend) work.templat(update);
```

You can see the list of template item stores by submitting the following statement:

```
ods path show;
```

The results are as follows:

```
Current ODS PATH list is:

1. WORK.TEMPLAT(UPDATE)
2. SASUSER.TEMPLAT(UPDATE)
3. SASHELP.TMPLMST(READ)
```

With this path, any template that you submit is stored in Work.Templat, which is deleted at the end of your SAS session.

The following statements illustrate how you can specify a permanent item store for your use and for the use of others:

```
libname mytpl 'C:\MyTemplateLibrary';
ods path (prepend) mytpl.templat(update);
```

Now, when you run PROC TEMPLATE, SAS creates an item store in the directory that you specified in the LIBNAME statement.

A.8 Modifying Your Graphs

Although ODS Graphics is designed to automate the creation of high-quality statistical graphics, on occasion you might need to modify your graphs. There are two ways to make modifications, depending on whether the changes you want to make are data-dependent and immediate (for a specific graph you are preparing for a paper or presentation) or persistent (applied to a graph each time you run the procedure). You can make immediate, ad hoc changes by using the ODS Graphics Editor, which provides a point-and-click interface. You can make persistent changes by modifying the ODS graph template for a particular plot. A graph template is a program, written in the Graph Template Language (GTL), that specifies the layout and details of a graph.

You can use the ODS Graphics Editor to customize titles and labels, annotate data points, add text, and change the properties of graph elements. After you modify your graph, you can save it as a PNG image file or as an SGE file, which preserves the editing context. You can open SGE files by using the ODS Graphics Editor and resume editing.

You can invoke the ODS Graphics Editor in the SAS windowing environment, provided that the LISTING destination is open and that you have enabled ODS Graphics to create editable graphs.

A.9 Data Objects

Graphs are constructed from a data component (or data object), macro and dynamic variables (optional), a graph template, and a style template. Procedures supply a table of data values and statistical results to plot. Together, the data object, the variables, and the templates form an output object that ODS displays in one or more output destinations. You can use the ODS OUTPUT statement to create a SAS data set from the underlying data object. You can use PROC PRINT and PROC CONTENTS to display the data set and its variables in order to learn about the data object and its rows and columns.

A.10 Recommended Reading

More information about ODS, ODS Graphics, the GTL, and SAS/STAT software is available in the following locations on the web:

`http://support.sas.com/documentation/`
`http://support.sas.com/documentation/onlinedoc/base/index.html`
`http://support.sas.com/documentation/onlinedoc/ods/index.html`
`http://support.sas.com/documentation/prod-p/grstat/index.html`
`http://support.sas.com/documentation/onlinedoc/graph/index.html`
`http://support.sas.com/documentation/onlinedoc/stat/index.html`

To learn more about ODS, see Chapter 20, "Using the Output Delivery System" (*SAS/STAT User's Guide*).

To learn more about ODS Graphics, see Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*). For introductory information, see the section "A Primer on ODS Statistical Graphics" (Chapter 21, *SAS/STAT User's Guide*).

To learn more about template modification, see Chapter 22, "ODS Graphics Template Modification" (*SAS/STAT User's Guide*).

To learn about modifying the Kaplan-Meier plot, see Chapter 23, "Customizing the Kaplan-Meier Survival Plot" (*SAS/STAT User's Guide*).

For complete ODS documentation, see *SAS Output Delivery System: User's Guide*.

For complete GTL documentation, see *SAS Graph Template Language: User's Guide* and *SAS Graph Template Language: Reference*.

For complete documentation about the Graphics Editor, see *SAS 9.4 ODS Graphics Editor: User's Guide*.

For information about the statistical graphics procedures, see *SAS ODS Graphics: Procedures Guide*.

Appendix B

Some Tips and Techniques for Understanding the Graph Template Language

Double-Click for Example Code

(Code links work in Adobe Reader and Internet Explorer.)

The Graph Template Language (GTL) provides a powerful and general syntax for creating graphs. However, if you are an experienced SAS programmer, you will find that some of the habits and insights that have served you well when you used other parts of SAS will not transfer to the GTL as well as you might expect. Understanding this will help you later, when you write your own templates and things do not work as you expect. The following examples illustrate.

The next steps use PROC TEMPLATE, the GTL, SG procedures, and PROC KDE. The following template, written in the GTL, illustrates the basic components of a fully functional graph template:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      BeginGraph;
         layout Overlay;
            ScatterPlot x=X y=Y;
         EndLayout;
      EndGraph;
   end;
run;
```

Graph template definitions in PROC TEMPLATE begin with a DEFINE STATGRAPH statement and end with an END statement. Embedded in every graph template is a BEGINGRAPH/ENDGRAPH block, and embedded in that block are one or more LAYOUT blocks. Each layout contains one or more statements that create a graph. The BEGINGRAPH/ENDGRAPH block can also contain entry titles and entry footnotes.

Now, to better understand the differences between the GTL and other SAS syntax, consider the following step that specifies a nonexistent variable:

```
proc print data=sashelp.class;
   var MyMadeUpVariable;
run;
```

When you run this step, SAS produces the following error message:

```
ERROR: Variable MYMADEUPVARIABLE not found.
```

Also consider the following steps, which also specify a nonexistent variable (or to be precise, a nonexistent column in the underlying table or "data object"):

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      BeginGraph;
         layout Overlay;
            ScatterPlot x=X y=Y;
            SeriesPlot  x=MyMadeUpVariable y=Y;
         EndLayout;
      EndGraph;
   end;
run;

ods graphics on;

proc kde data=sashelp.class;
   bivar height weight / plots=scatter;
run;
```

PROC KDE is a SAS/STAT procedure that performs kernel density estimation and also produces scatter plots of two data object columns, named X and Y, that it creates from the input data. It uses the template **Stat.KDE.Graphics.ScatterPlot**. The SERIESPLOT statement was added to the template. The template definition for the series plot specifies a column named MyMadeUpVariable that does not exist. No error messages are produced. First, no error can be issued when PROC TEMPLATE runs, because PROC TEMPLATE cannot see the data object to know if x, y, and MyMadeUpVariable exist. Second, no error is issued by PROC KDE or ODS Graphics. This is by design. No error is issued because there is nothing wrong with this template. It is perfectly legitimate for templates to specify a value that is nonexistent. When this happens, that part of the graph simply drops out without a note, error, or warning. The template language is designed for maximum flexibility and versatility. You can use this template to make a scatter plot (when both x and y exist), a series plot (when MyMadeUpVariable and y exist), or both (when x, MyMadeUpVariable, and y exist). No conditional logic is needed; it all happens automatically and by design. However, there is a downside. When you misspell a name, you might not get a message that helps you figure out what went wrong, because as far as SAS, ODS, and the GTL are concerned, nothing went wrong.

It is important that the GTL work this way. SAS procedures use graph templates for every data set, reasonable and unreasonable, that is input. If a template needed extra statements and logic to handle every unreasonable data set or option combination that was thrown at it, it would be much more complicated. Instead, variables that do not exist, contain all missing values, or are otherwise invalid drop out, a reasonable graph is produced, and the procedure runs quietly without displaying confusing messages about what happened. The GTL works correctly, and it works reasonably, but it might not work the way you expect when you make a mistake.

Now consider the following steps, which again specify a nonexistent data object column:

```
proc template;
   define statgraph ScatterPlot;
      BeginGraph;
         layout Overlay;
            ScatterPlot x=height y=weight;
            SeriesPlot  x=MyMadeUpVariable y=weight;
```

```
         EndLayout;
      EndGraph;
   end;
run;

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

The PROC SGRENDER step produces the following warning:

```
WARNING: The SeriesPlot statement will not be drawn because
         one or more of the required arguments were not supplied.
```

PROC SGRENDER is saying that it does not have one or more of the required arguments (the X= and Y= options), so the series plot is not drawn. Both options are specified, but one is specified with a nonexistent column, so the specification drops out as if it had not been specified at all. You are probably wondering why SAS prints nothing for the first case and a warning for this second case. The GTL is the language that SAS procedure writers use to produce graphs. When a template is used by an analytical procedure, ODS Graphics assumes that a nonexistent specification is included by design, so the procedure runs quietly, dropping the nonexistent part. However, when PROC SGRENDER is used, ODS Graphics assumes that a user-written template is being used, so it prints more detailed error messages. Understanding this distinction can be a big help when you write and modify templates. Particularly when you modify a template that SAS provides, you cannot rely solely on warnings and error messages to help you understand what went wrong when a graph does not come out right. Rather, you need a thorough understanding of the GTL and how it works. When you write your own template for use with PROC SGRENDER, you will usually get more helpful messages, but you still cannot solely rely on SAS messages.

The GTL differs from other SAS syntax in other ways as well. Most SAS procedures have two types of options. Keyword options have the form *keyword=value*. Examples include DATA=*SAS-data-set*, ORDER=DATA, and PLOTS=ALL. The other type of option consists of an option name but no equal sign or value. These options are Boolean (true or false) in nature. By default, the option is not specified; you can specify the option to negate the default behavior. For example, by default, PROC PRINT does not display variable labels, but you can specify the LABELS option to see labels. By default, PROC MEANS displays its output, but you can specify the NOPRINT option to suppress output.

In contrast, the GTL uses only keyword options: *keyword=value*. Even Boolean options are keyword options and have the form *keyword*=TRUE | FALSE. For example, the option for specifying whether or not a plot is primary is controlled by specifying PRIMARY=TRUE or PRIMARY=FALSE. Some of these options have a default of true, and some have a default of false.

The GTL does not accept lists that are accepted in other SAS syntax, and GTL syntax is sometimes verbose. For example, the legacy SAS/GRAPH statement **`axis1 order=(0 to 50 by 10)`** specifies tick values of 0, 10, 20, 30, 40, and 50. In the GTL, this option is specified as follows: **`xaxisopts=(linearopts=(tickvaluesequence=(start=0 end=50 increment=10)))`**. However, although the syntax can be verbose, ODS Graphics and the SG procedures do so much for you that ODS Graphics template and programs are often shorter and simpler than legacy SAS/GRAPH programs.

In the GTL, you cannot specify a statement that has a list like the following: **`dynamic x1-x6 y1-y6`**. You must include a full specification (**`dynamic x1 x2 x3 x4 x5 x6 y1 y2 y3 y4 y5`**

y6). Alternatively, you can use the macro language (**dynamic %macro l; %do i = 1 %to 6; x&i y&i %end; %mend; %l**).

The templates that SAS provides are sometimes written using the SAS macro language. For example, the following template is one that SAS provides, displayed in the way it was originally written by using a SAS macro:

```
define statgraph Stat.Transreg.Graphics.PBSplineCriterion;
   notes "Penalized B-spline (CV,GCV,AIC,AICC,SBC) * Lambda Plots";
   %macro plot;
   dynamic %do i = 1 %to 6; s&i l&i t&i %end; nplots nrows ncols
           criterion shrink head height;
   begingraph / designheight=height;
      entrytitle head;
      layout lattice / columns=ncols rows=nrows order=packed
                       skipemptycells=true rowgutter=10 columngutter=10
                       shrinkfonts=shrink;
         rowheaders;
            entry criterion / rotate=90 textattrs=graphlabeltext;
            entry criterion / rotate=90 textattrs=graphlabeltext;
            endrowheaders;
         %do i = 1 %to 6;
            if(&i <= nplots)
               layout overlay /
                     yaxisopts=(display=(ticks tickvalues)
                                offsetmin=0.05 offsetmax=0.05)
                     xaxisopts=(type=t&i label="Lambda"
                                offsetmin=0.05 offsetmax=0.05);
                  if (ncols > 1)
                     entry l&i / textattrs=GraphTitleText location=outside;
                     endif;
                  scatterplot y = y&i x = x&i / primary=true tip=(y x)
                        tiplabel=(x="Lambda");
                  scatterplot y=oy&i x=ox&i /
                        markerattrs=GraphData1(symbol=star size=15)
                        tip=(y x)
                        tiplabel=(x='Selected Lambda' y = criterion);
                  entry {Unicode Lambda} " = " s&i /
                        autoalign=(topright topleft bottomright bottomleft
                                   right left top bottom);
                  endlayout;
               endif;
            %end;
         endlayout;
      dynamic _byline_ _bytitle_ _byfootnote_;
      if (_bytitle_) entrytitle _byline_ / textattrs=GraphValueText;
      else if (_byfootnote_) entryfootnote halign=left _byline_;
      endif; endif;
      endgraph;
      %mend; %plot;
```

This template creates a lattice of up to six plots, arranged in a 2×3 grid. Each plot is defined by a LAYOUT OVERLAY block along with a SCATTERPLOT statement to make a plot and an ENTRY statement that displays a smoothing parameter in the plot as inset text. Notice that a macro DO loop

is used to expand the original variable list. Also, a macro DO loop is used to generate six overlay layouts, one for each of the triples x1, y1, and s1 through x6, y6, and s6.

When you display this template definition, SAS shows you the following syntax:

```
define statgraph Stat.Transreg.Graphics.PBSplineCriterion;
   notes "Penalized B-spline (CV,GCV,AIC,AICC,SBC) * Lambda Plots";
   dynamic s1 l1 t1 s2 l2 t2 s3 l3 t3 s4 l4 t4 s5 l5 t5 s6 l6 t6 nplots nrows
      ncols criterion shrink head height _byline_ _bytitle_ _byfootnote_;
   begingraph / designheight=HEIGHT;
      entrytitle HEAD;
      layout lattice / columns=NCOLS rows=NROWS order=packed skipemptycells=
         true rowgutter=10 columngutter=10 shrinkfonts=SHRINK;
         rowheaders;
            entry CRITERION / rotate=90 textattrs=GRAPHLABELTEXT;
            entry CRITERION / rotate=90 textattrs=GRAPHLABELTEXT;
         endrowheaders;
         if (1 <= NPLOTS)
            layout overlay / yaxisopts=(display=(ticks tickvalues) offsetmin=
               0.05 offsetmax=0.05) xaxisopts=(type=T1 label="Lambda"
               offsetmin=0.05 offsetmax=0.05);
               if (NCOLS > 1)
                  entry L1 / textattrs=GRAPHTITLETEXT location=outside;
               endif;
               scatterplot y=Y1 x=X1 / primary=true tip=(y x) tiplabel=(x=
                  "Lambda");
               scatterplot y=OY1 x=OX1 / markerattrs=GRAPHDATA1 (symbol=star
                  size=15) tip=(y x) tiplabel=(x='Selected Lambda' y=
                  CRITERION);
               entry { Unicode LAMBDA } " = " S1 / autoalign=(topright
                  topleft bottomright bottomleft right left top bottom);
            endlayout;
         endif;
         if (2 <= NPLOTS)
            layout overlay / yaxisopts=(display=(ticks tickvalues) offsetmin=
               0.05 offsetmax=0.05) xaxisopts=(type=T2 label="Lambda"
               offsetmin=0.05 offsetmax=0.05);
               if (NCOLS > 1)
                  entry L2 / textattrs=GRAPHTITLETEXT location=outside;
               endif;
               scatterplot y=Y2 x=X2 / primary=true tip=(y x) tiplabel=(x=
                  "Lambda");
               scatterplot y=OY2 x=OX2 / markerattrs=GRAPHDATA1 (symbol=star
                  size=15) tip=(y x) tiplabel=(x='Selected Lambda' y=
                  CRITERION);
               entry { Unicode LAMBDA } " = " S2 / autoalign=(topright
                  topleft bottomright bottomleft right left top bottom);
            endlayout;
         endif;
         if (3 <= NPLOTS)
            layout overlay / yaxisopts=(display=(ticks tickvalues) offsetmin=
               0.05 offsetmax=0.05) xaxisopts=(type=T3 label="Lambda"
               offsetmin=0.05 offsetmax=0.05);
               if (NCOLS > 1)
```

```
            entry L3 / textattrs=GRAPHTITLETEXT location=outside;
         endif;
         scatterplot y=Y3 x=X3 / primary=true tip=(y x) tiplabel=(x=
            "Lambda");
         scatterplot y=OY3 x=OX3 / markerattrs=GRAPHDATA1 (symbol=star
            size=15) tip=(y x) tiplabel=(x='Selected Lambda' y=
            CRITERION);
         entry { Unicode LAMBDA } " = " S3 / autoalign=(topright
            topleft bottomright bottomleft right left top bottom);
      endlayout;
   endif;
   if (4 <= NPLOTS)
      layout overlay / yaxisopts=(display=(ticks tickvalues) offsetmin=
         0.05 offsetmax=0.05) xaxisopts=(type=T4 label="Lambda"
         offsetmin=0.05 offsetmax=0.05);
         if (NCOLS > 1)
            entry L4 / textattrs=GRAPHTITLETEXT location=outside;
         endif;
         scatterplot y=Y4 x=X4 / primary=true tip=(y x) tiplabel=(x=
            "Lambda");
         scatterplot y=OY4 x=OX4 / markerattrs=GRAPHDATA1 (symbol=star
            size=15) tip=(y x) tiplabel=(x='Selected Lambda' y=
            CRITERION);
         entry { Unicode LAMBDA } " = " S4 / autoalign=(topright
            topleft bottomright bottomleft right left top bottom);
      endlayout;
   endif;
   if (5 <= NPLOTS)
      layout overlay / yaxisopts=(display=(ticks tickvalues) offsetmin=
         0.05 offsetmax=0.05) xaxisopts=(type=T5 label="Lambda"
         offsetmin=0.05 offsetmax=0.05);
         if (NCOLS > 1)
            entry L5 / textattrs=GRAPHTITLETEXT location=outside;
         endif;
         scatterplot y=Y5 x=X5 / primary=true tip=(y x) tiplabel=(x=
            "Lambda");
         scatterplot y=OY5 x=OX5 / markerattrs=GRAPHDATA1 (symbol=star
            size=15) tip=(y x) tiplabel=(x='Selected Lambda' y=
            CRITERION);
         entry { Unicode LAMBDA } " = " S5 / autoalign=(topright
            topleft bottomright bottomleft right left top bottom);
      endlayout;
   endif;
   if (6 <= NPLOTS)
      layout overlay / yaxisopts=(display=(ticks tickvalues) offsetmin=
         0.05 offsetmax=0.05) xaxisopts=(type=T6 label="Lambda"
         offsetmin=0.05 offsetmax=0.05);
         if (NCOLS > 1)
            entry L6 / textattrs=GRAPHTITLETEXT location=outside;
         endif;
         scatterplot y=Y6 x=X6 / primary=true tip=(y x) tiplabel=(x=
            "Lambda");
         scatterplot y=OY6 x=OX6 / markerattrs=GRAPHDATA1 (symbol=star
            size=15) tip=(y x) tiplabel=(x='Selected Lambda' y=
```

```
                  CRITERION);
               entry { Unicode LAMBDA } " = " S6 / autoalign=(topright
                  topleft bottomright bottomleft right left top bottom);
            endlayout;
         endif;
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   endgraph;
end;
```

For an example of displaying template syntax, see Chapter 4, "Graph Template Modification."

Before PROC TEMPLATE compiles the template, the macro language expands the variable lists and produces the six overlay blocks. After that, PROC TEMPLATE compiles the template and stores the results. Then you use PROC TEMPLATE to reproduce the template source from the compiled template, not from the original source. This process can change spacing, remove comments, and change capitalization. When you look at templates that SAS provides, it is important to understand that they might have looked very different when they were developed. Some templates that SAS provides are much longer than this one. However, they are usually much less complicated than they might appear at first glance. Although this version of the template is certainly longer than the original, macro version, it has the same straightforward organization of a lattice that has six layouts, each defining a scatter plot.

It is important to understand that the GTL is a language developed for internal use at SAS. It was developed to be a comprehensive language for capturing the definition of a potentially very complex graph. Although SAS makes this language available to you, it is not designed to be an obvious step away from a syntax that is familiar to longtime SAS users. It is a new and different language. However, you will find as you become experienced with the GTL that it is not difficult to use. You will also discover that it has benefits that make it well worth learning.

Index