SAS/STAT® 14.1 User's Guide
ODS Graphics Template Modification

This document is an individual chapter from *SAS/STAT® 14.1 User's Guide*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *SAS/STAT® 14.1 User's Guide*. Cary, NC: SAS Institute Inc.

SAS/STAT® 14.1 User's Guide

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

July 2015

Chapter 22
ODS Graphics Template Modification

Contents

Graph Templates

This chapter discusses the graph template language and graph template modification in ODS Graphics. Be sure that you are familiar with Chapter 21, "Statistical Graphics Using ODS," before reading this chapter.

Graph templates control the layout and details of graphs produced with ODS Graphics. The SAS System provides a template for every graph produced by statistical procedures. Graph template definitions are written in the Graph Template Language (GTL). This powerful language includes statements for specifying plot layouts (such as lattices or overlays), plot types (such as scatter plots and histograms), and text elements (such as titles, footnotes, and insets). It also provides support for built-in computations (such as histogram binning) and the evaluation of expressions. Options are available for specifying colors, marker symbols, and other attributes of plot features.

Graphs, like all SAS output, are constructed from two underlying components, a data component (or data object) and a template. Procedures supply a table of data values and statistical results to plot. Together, the data object and the template form an output object that ODS displays in one or more output destinations. You can control this display in two ways. You can use the ODS Graphics Editor (discussed in the section "ODS Graphics Editor" on page 637 in Chapter 21, "Statistical Graphics Using ODS") to modify the output object (but not the underlying data object or template), and you can use the GTL to modify the template. With just a little knowledge of the GTL, you can modify or edit templates, even when you do not understand most of the syntax used in the template definition. See examples starting with Example 22.1.

NOTE: You do not need to know anything about the GTL to create statistical graphics.

This section provides an overview of the Graph Template Language. It also describes how to locate, display, edit, and save templates. A *template definition* is a set of SAS statements that is used together with PROC TEMPLATE to create a compiled template. In addition to graph templates, two other common types of templates are table templates and style templates. A table template describes how to display the output for an output object that is rendered as a table. A style template provides formatting information for visual aspects of your SAS output, including both tables and graphs. In most applications, you do not have to modify the templates that SAS supplies. However, when customization is necessary, you can modify the default template with the template language and PROC TEMPLATE.

Compiled templates are stored in a template store, which is a type of item store. (An item store is a special type of SAS file.) The default templates that SAS supplies are stored in template store in the Sashelp library. If you are using the SAS windowing environment, an easy way to display, edit, and save your templates is by using the Templates window. For an introduction to the graph template language, see Kuhfeld (2010).

For detailed information about managing templates, see the *SAS Output Delivery System: User's Guide* and the *SAS Graph Template Language: User's Guide*. For more information about the syntax of the graph template language, see the *SAS Graph Template Language: Reference*.

The Graph Template Language

Graph template definitions begin with a DEFINE STATGRAPH statement in PROC TEMPLATE, and they 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. You can specify the DYNAMIC statement to define dynamic variables (which the procedure uses to pass values to the template definition), the MVAR and

NMVAR statements to define macro variables (which you can use to pass values to the template definition), and the NOTES statement to provide descriptive information about the graph. The default templates that SAS supplies for statistical procedures are often lengthy and complex, because they provide ODS Graphics with comprehensive and detailed information about graph construction. Here is one of the simpler graph templates for a statistical procedure:

```
define statgraph Stat.MDS.Graphics.Fit;
   notes "MDS Fit Plot";
   dynamic head _byline_ _bytitle_ _byfootnote_;
   begingraph / designwidth=defaultdesignheight;
      entrytitle HEAD;
      layout overlayequated / equatetype=square;
         scatterplot y=FITDATA x=FITDIST / markerattrs=(size=5px);
         lineparm slope=1 x=0 y=0 / clip=true extend=true
            lineattrs=GRAPHREFERENCE;
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   endgraph;
end;
```

This template, which is supplied for the MDS procedure, creates a scatter plot of two variables, FitData and FitDist, along with a diagonal reference line that passes through the origin. The plot is square and the axes are equated so that a centimeter on one axis represents the same data range as a centimeter on the other axis. The plot title is provided by the evaluation of the dynamic variable Head, which is set by PROC MDS. It is not unusual for this plot to contain hundreds or even thousands of points, so a five-pixel marker is specified, which is smaller than the seven-pixel marker that most styles use by default.

The IF/ELSE construction is common to most templates and enables ODS Graphics to optionally display a BY line with BY-group information as either a footnote or a second title line.

The statements available in the graph template language can be classified as follows:

- Control statements specify the conditional or iterative flow of control. By default, flow of control is sequential. In other words, each statement is used in the order in which it appears.
- Layout statements specify the arrangement of the components of the graph. Layout statements are arranged in blocks that begin with a LAYOUT statement and end with an ENDLAYOUT statement. The blocks can be nested. Within a layout block, there can be plot, text, and other statements that define one or more graph components. Options provide control for attributes of layouts and components.
- Plot statements specify a number of commonly used displays, including scatter plots, histograms, contour plots, surface plots, and box plots. Plot statements are always provided within a layout block. The plot statements include options to specify the data columns from the data object that is used in the graph. For example, in the SCATTERPLOT statement, there are mandatory X= and Y= arguments that specify which data columns are used for the X (horizontal) and Y (vertical) axes in the plot. (In the preceding example, FitData and FitDist are the names of columns in the data object that PROC MDS

creates for this graph.) There is also a GROUP= option that specifies a data column as an optional classification variable.

- Text statements specify the descriptions that accompany graphs. An entry is any textual description, including titles, footnotes, and legends; it can include symbols to identify graph elements.

The following statements display another of the simpler template definitions—the definition of the scatter plot available in PROC KDE (see Figure 66.6.1 in Chapter 66, "The KDE Procedure"):

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL
              _byline_ _bytitle_ _byfootnote_;
      BeginGraph;
         EntryTitle "Distribution of " _VAR1NAME " by " _VAR2NAME;
         Layout Overlay / xaxisopts=(offsetmin=0.05 offsetmax=0.05)
                          yaxisopts=(offsetmin=0.05 offsetmax=0.05);
            ScatterPlot x=X y=Y / markerattrs=GRAPHDATADEFAULT;
         EndLayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
run;
```

Here, the PROC TEMPLATE and RUN statements have been added to show how to compile the template if you want to modify it. The DEFINE STATGRAPH statement in PROC TEMPLATE begins the graph template definition, and the END statement ends the definition. The DYNAMIC statement defines four dynamic variables that PROC KDE sets at run time. Three additional DYNAMIC variables control the BY line. The dynamic variables _Var1Name and _Var2Name provide the variable names for the title of the graph. The dynamic variables _Var1Label and _Var2Label contain the labels of the X and Y variables, respectively. The title of the graph is specified by the ENTRYTITLE statement. You can change the ENTRYTITLE statement to display variable labels instead of variable names in the title. You can also use any of the dynamic variables elsewhere in the graph template.

The axes are controlled by the LAYOUT OVERLAY statement inside the BEGINGRAPH/ENDGRAPH statement block. The offset options add a small amount of padding between the axes and the most extreme points. The SCATTERPLOT statement creates the scatter plot. The options in the SCATTERPLOT statement are specified after the slash and control the marker attributes (symbol, color, and size). You can specify these attributes directly, as in the PROC MDS template, or more typically through style elements, as in the PROC KDE template. Style elements are defined in ODS styles, and elements can vary across different styles. For more information about ODS styles, see the section "ODS Styles" on page 643 in Chapter 21, "Statistical Graphics Using ODS." The ENDLAYOUT statement ends the main layout block. For information about the syntax of the graph template language, see the *SAS Graph Template Language: Reference*.

You can write your own templates and use them to display raw data or output from procedures. For example, consider the iris data from Example 35.1 of Chapter 35, "The DISCRIM Procedure." The iris data set is available from the Sashelp library.

The following statements create a template for a scatter plot of the variables PetalLength and PetalWidth with a legend:

```
proc template;
   define statgraph scatter;
      begingraph;
         entrytitle 'Fisher (1936) Iris Data';
         layout overlayequated / equatetype=fit;
            scatterplot x=petallength y=petalwidth /
                        group=species name='iris';
            layout gridded / autoalign=(topleft);
               discretelegend 'iris' / border=false opaque=false;
            endlayout;
         endlayout;
      endgraph;
   end;
run;
```

The layout is OVERLAYEQUATED, which equates the axes in the plot. However, unlike the PROC MDS template, which used EQUATETYPE=SQUARE to make a square plot, the EQUATETYPE=FIT option specifies that the lengths of the axes in this plot should fill the entire plotting area. A legend is placed internally in the top-left portion of the plot. There are three groups of observations, indicated by the three species, and each group is plotted with a separate color and symbol that depends on the ODS style. The legend identifies each group. The NAME= option provides the link between the SCATTERPLOT statement and the DISCRETELEGEND statement. An explicit link is needed since some graphical displays are based on multiple plotting statements.

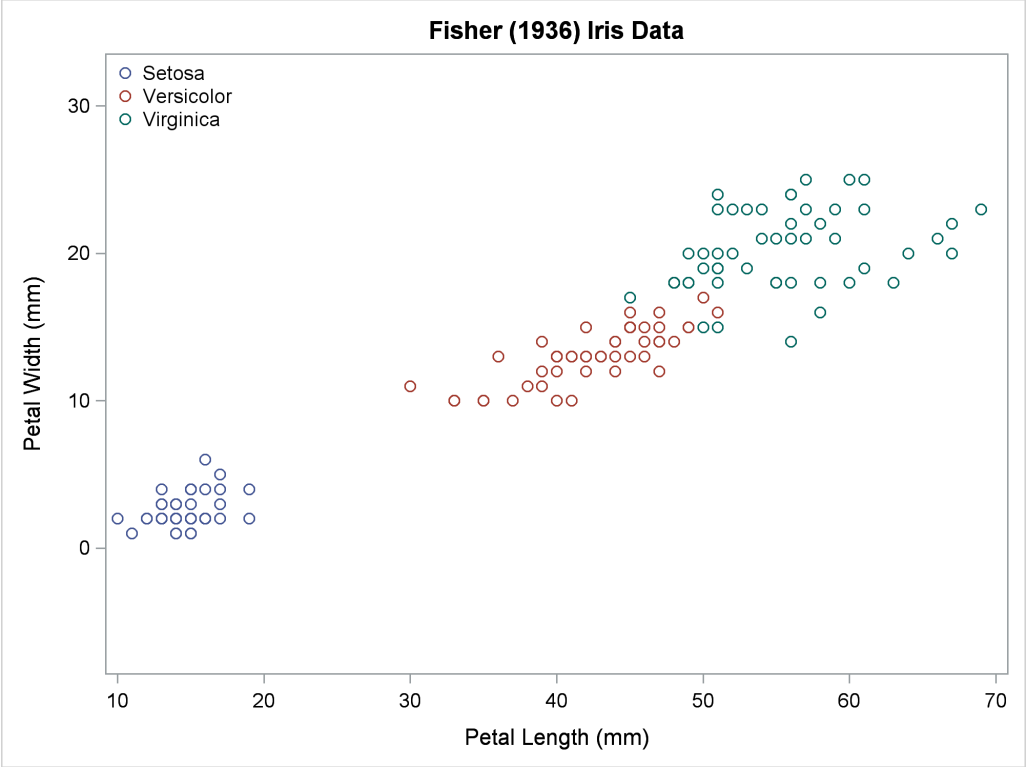
The following step creates the plot by using the SGRENDER procedure, the Sashelp.Iris data set, and the custom template **scatter**:

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

The syntax of PROC SGRENDER is very simple, because all of the graphical options appear in the template. The scatter plot in Figure 22.1 shows the results.

The intent of this example is to illustrate how you can write a template to create a scatterplot. PROC TEMPLATE and PROC SGRENDER provide you with the power to create highly customized displays. However, usually you can use the SGPLOT, SGSCATTER or SGPANEL procedures instead, which are much simpler to use. These procedures are discussed in the section "Statistical Graphics Procedures" on page 699 in Chapter 21, "Statistical Graphics Using ODS." See the section "Grouped Scatter Plot with PROC SGPLOT" on page 606 and Figure 21.12 in Chapter 21, "Statistical Graphics Using ODS," for an example that plots these data with PROC SGPLOT.

Figure 22.1 Petal Width and Petal Length in Three Iris Species



Locating Templates

Before you can customize a graph, you must determine which template is used to create the original graph. You can do this by submitting the ODS TRACE ON statement before the procedure statements that create the graph. The template name is displayed in the SAS log. Here is an example:

```
ods trace on;
ods graphics on;

proc reg data=sashelp.class;
   model Weight = Height;
run; quit;
```

The preceding statements create the following trace output, which provides information about both the graphs and tables produced by PROC REG:

```
Output Added:
-------------
Name:        NObs
Label:       Number of Observations
Template:    Stat.Reg.NObs
Path:        Reg.MODEL1.Fit.Weight.NObs
-------------
```

```
Output Added:
-------------
Name:        ANOVA
Label:       Analysis of Variance
Template:    Stat.REG.ANOVA
Path:        Reg.MODEL1.Fit.Weight.ANOVA
-------------

Output Added:
-------------
Name:        FitStatistics
Label:       Fit Statistics
Template:    Stat.REG.FitStatistics
Path:        Reg.MODEL1.Fit.Weight.FitStatistics
-------------

Output Added:
-------------
Name:        ParameterEstimates
Label:       Parameter Estimates
Template:    Stat.REG.ParameterEstimates
Path:        Reg.MODEL1.Fit.Weight.ParameterEstimates
-------------

Output Added:
-------------
Name:        DiagnosticsPanel
Label:       Fit Diagnostics
Template:    Stat.REG.Graphics.DiagnosticsPanel
Path:        Reg.MODEL1.ObswiseStats.Weight.DiagnosticPlots.DiagnosticsPanel
-------------

Output Added:
-------------
Name:        ResidualPlot
Label:       Height
Template:    Stat.REG.Graphics.ResidualPlot
Path:        Reg.MODEL1.ObswiseStats.Weight.ResidualPlots.ResidualPlot
-------------

Output Added:
-------------
Name:        FitPlot
Label:       Fit Plot
Template:    Stat.REG.Graphics.Fit
Path:        Reg.MODEL1.ObswiseStats.Weight.FitPlot
-------------
```

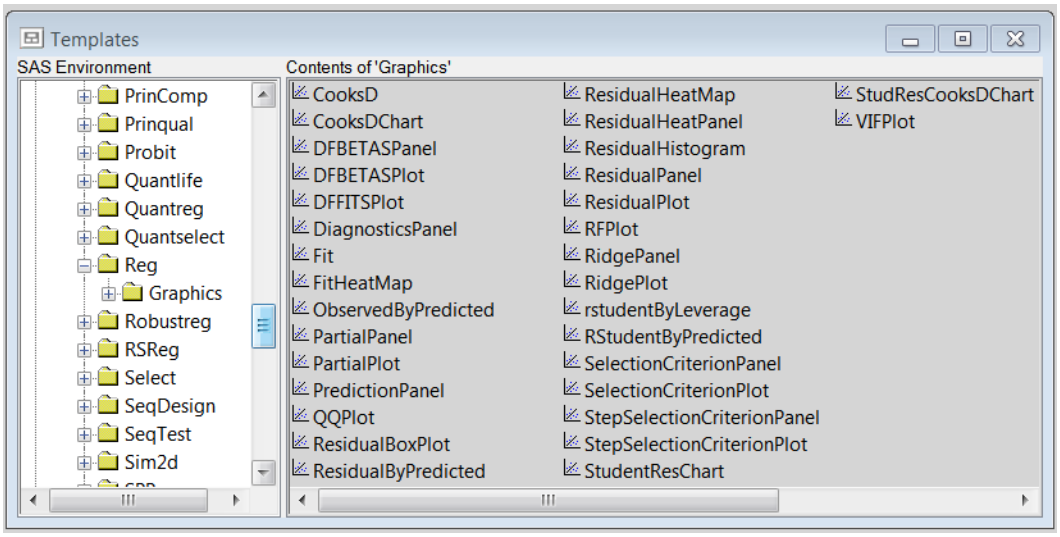
This is also illustrated in Example 22.1 and the section "The ODS Statement" on page 529 in Chapter 20, "Using the Output Delivery System."

Displaying Templates

Once you have found the name of a template, you can display its definition (source program) by using one of these methods:

- Open the Templates window by issuing the command **odstemplates** (**odst** for short) in the command line of the SAS windowing environment. The template window is shown in Figure 22.2. If you expand the Sashelp.Tmplstat node, you can view all the available templates and double-click any template icon to display its definition. This is illustrated in Example 22.1.
- Use the SOURCE statement in PROC TEMPLATE to display a template definition in the SAS log or write the definition to a file.

Figure 22.2 Requesting the Templates Window in the Command Line



For example, the following statements display the template for the PROC REG residual plot:

```
proc template;
   source Stat.REG.Graphics.ResidualPlot;
run;
```

The template is displayed as follows:

```
define statgraph Stat.Reg.Graphics.ResidualPlot;
   notes "Residual Plot";
   dynamic _XVAR _SHORTXLABEL _TITLE _LOESSLABEL _DEPNAME _MODELLABEL _SMOOTH
      _byline_ _bytitle_ _byfootnote_;
   BeginGraph;
      entrytitle halign=left textattrs=GRAPHVALUETEXT _MODELLABEL halign=
         center textattrs=GRAPHTITLETEXT _TITLE " for " _DEPNAME;
      entrytitle textattrs=GRAPHVALUETEXT _LOESSLABEL;
      layout overlay / xaxisopts=(shortlabel=_SHORTXLABEL);
         referenceline y=0;
```

```
            scatterplot y=RESIDUAL x=_XVAR / primary=true rolename=(_tip1=
               OBSERVATION _id1=ID1 _id2=ID2 _id3=ID3 _id4=ID4 _id5=ID5) tip=(y
               x _tip1 _id1 _id2 _id3 _id4 _id5);
            if (EXISTS(_SMOOTH))
               loessplot y=_SMOOTH x=_XVAR / tiplabel=(y="Smoothed Residual");
            endif;
         endlayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
```

PROC TEMPLATE also tells you where the template is located. In this case, it prints the following note:

```
NOTE: Path 'Stat.Reg.Graphics.ResidualPlot' is in: SASHELP.TMPLSTAT
      (via SASHELP.TMPLMST).
```

The word “Path” in ODS has several meanings. In this context, it refers to any name or label hierarchy. In the note, the levels of the template name form a path. In the trace output, the levels of the plot name form a different path.

Editing Templates

You can modify the format and appearance of a particular graph by doing the following:

- Modify its template definition (source program).
- Submit the revised template to create a new compiled template, which is stored in a template store.
- Ensure that the ODS template search path includes the template store that contains your new template.

Template stores are designated read-only (such as Sashelp.Tmplmst) or updatable (such as Sasuser.Templat).

If you view the templates in an updatable template store from the Templates window, you can select **Open** or **Edit** from the pop-up menu. Either the Template Browser or Template Editor window opens. In the Template Editor window, you can make changes and submit the code directly. For read-only templates or when you select **Open**, the Template Browser window opens and you must copy the definition to an editor window to make changes. Since templates that SAS supplies are in the read-only Sashelp library, an easy way to obtain an editable program file is to use the SOURCE statement with the FILE= option in PROC TEMPLATE to write the template definition to a file as follows:

```
proc template;
   source Stat.REG.Graphics.ResidualPlot / file="residtpl.sas";
run;
```

By default, the file is saved in the SAS current folder. Alternatively, you can omit the slash and the FILE= option and copy and paste the source from the SAS log into an editor. Either way, you must add a PROC TEMPLATE statement before the generated source statements and optionally a RUN statement after the END statement before you submit your modified definition.

Graph templates are self-contained and do not support inheritance (via the PARENT= option) as do table templates. Consequently, the EDIT statement in PROC TEMPLATE is not supported for graph templates.

Here are some important points about what you can and cannot change in a template that SAS supplies while preserving its overall functionality:

- Do not change the template name. A statistical procedure can access only a predefined list of templates. If you change the name, the procedure cannot find your template. You must keep the original name and make sure that it is in a template store that is searched before Sashelp.Tmplmst. You control this with the ODS PATH statement. For more information about the template search path and the ODS PATH statement, see the section "The Default Template Stores and the Template Search Path" on page 641 in Chapter 21, "Statistical Graphics Using ODS," the section "Saving Customized Templates" on page 731, and subsequent sections.
- Do not change the names of columns. The underlying data object contains predefined column names that you must use. Be very careful if you change how a column is used in a template. Usually, columns are not interchangeable.
- Do not change the names of DYNAMIC variables. Procedures set values only for a predefined list of dynamic variables. Changing dynamic variable names can lead to runtime errors. Do not add dynamic variables, because the procedure cannot set their values. A few procedures document additional dynamic variables that can be defined in the template if you want to add more information to the output, such additional statistics in an inset table.
- Do not change the names of statements (for example, from a SCATTERPLOT to a NEEDLEPLOT or other type of plot).

You can change any of the following:

- You can add macro variables that behave like dynamic variables. They are resolved at the time that the statistical procedure is run, and not at the time that the template is compiled. They are defined with an MVAR or NMVAR statement at the beginning the template. You can set the value of each macro variable with a %LET statement before the statistical procedure is run. See Example 22.5. You can also move a variable from a DYNAMIC statement to an MVAR or NMVAR statement if you want to set it yourself rather than letting the procedure set it.
- You can change the graph size.
- You can change graph titles, footnotes, axis labels, and any other text that appears in the graph.
- You can change which plot features are displayed.
- You can change axis features, such as grid lines, offsets, view ports, tick value formatting, and so on.
- You can change the content and arrangement of insets (small tables of statistics embedded in some graphs).
- You can change the legend location, contents, border, background, title, and so on.

See the *SAS Graph Template Language: Reference* for information about the syntax of the statements in the Graph Template Language.

Saving Customized Templates

After you edit the template definition, you can submit your PROC TEMPLATE statements as you would any other SAS program. If you are using the Template Editor window, select **Submit** from the **Run** menu. See Example 22.1. Alternatively, submit your PROC TEMPLATE statements from the Program Editor. ODS automatically saves the compiled template in the first template store that it can update, according to the currently defined template search path. If you have not changed the template search path, then the modified template is saved in the Sasuser.Templat template store. You can use the following statement to display the current template search path:

```
ods path show;
```

The log messages for the default template search path are as follows:

```
Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

The ODS PATH statement template search pathname Sashelp.Tmplmst refers to the template stores that are shipped with the SAS System. More precisely, the ODS PATH name Sashelp.Tmplmst refers to multiple template store files, including Sashelp.Tmplmst, Sashelp.Tmplstat, Sashelp.Tmplets, Sashelp.Tmplqc, Sashelp.Tmpliml, and others. The name Sashelp.Tmplmst refers to both the entire template store that is shipped with the SAS System and a particular file in that template store. Earlier releases of the SAS System had just one template store file, Sashelp.Tmplmst, and there was a one-to-one correspondence between the ODS PATH statement name and the template store filename. Now there are multiple files (because certain products have their own template store files), but the ODS PATH statement syntax for selecting all of them is unchanged. You do not need to be concerned about this, and you should not specify any of these template stores individually. You simply specify Sashelp.Tmplmst to get the item stores that are shipped with the SAS System. However, you will sometimes see these other names in the SAS log and output when you are working with templates, so you need to know what that means.

If you want to store modified templates in another template store, you can use the ODS PATH statement to add that template store to the front of the list. To use these templates, you must make sure the template search path is set correctly before you attempt to access them in the other SAS sessions. See the section "Using Customized Templates" on page 731.

Using Customized Templates

When you create ODS output (either graphs or tables), ODS searches sequentially through each template store in the template search path for a template that matches the one requested. If you have not changed the default template search path, then ODS searches the Sasuser.Templat store first, then Sashelp.Tmplmst. ODS uses the first template that it finds with the requested name. **NOTE:** Templates with the same name can exist in more than one template store.

The ODS PATH statement specifies the template stores to search, as well as the order in which to search them. You can change the default template search path by using the ODS PATH statement. For example, the

following statement sets the template search path so that the template store Work.Mystore is searched first, followed by Sashelp.Tmplmst:

```
ods path work.mystore(update) sashelp.tmplmst(read);
```

The UPDATE option provides both update access and read access to Work.Mystore. The READ option provides read-only access to Sashelp.Tmplmst. With this path, the template store Sasuser.Templat is no longer searched. You can verify this by using the following statement:

```
ods path show;
```

The log messages that the preceding statement generates are as follows:

```
Current ODS PATH list is:

1. WORK.MYSTORE(UPDATE)
2. SASHELP.TMPLMST(READ)
```

For more information, see the *SAS Output Delivery System: User's Guide* and the *SAS Graph Template Language: User's Guide*. Example 22.1 illustrates all the steps of displaying, editing, saving, and using customized templates.

Reverting to the Default Templates

Customized templates are stored in Sasuser.Templat or in some other template store that you create. The templates that SAS supplies are in the read-only template store Sashelp.Tmplmst. If you modify any of the supplied templates and you want to use the original default templates, you can change your template search path as follows:

```
ods path sashelp.tmplmst(read) sasuser.templat(update);
```

This way the default templates are found first. Alternatively, you can save all your customized templates in a user-defined template store (for example Mylib.Mystore). To access these templates, you submit the following statement before running your analysis:

```
libname mylib '.';
ods path mylib.mystore(update) sashelp.tmplmst(read);
```

When you are done, you can reset the default template search path as follows:

```
ods path reset;
```

This restores the template search path to its original state (**sasuser.templat(update) sashelp.tmplmst(read)**). You can also save your customized template as part of your SAS program. When you are done, you can delete your customized template from the Sasuser.Templat template store, as in the following statements:

```
proc template;
   delete Stat.REG.Graphics.ResidualPlot / store=sasuser.templat;
run;
```

The option STORE=SASUSER.TEMPLAT is not required. However, if you have administrator privileges on your computer, this option helps you ensure that you do not accidentally delete templates from Sashelp.Tmplmst.

The following note is printed in the SAS log:

```
NOTE: 'Stat.REG.Graphics.ResidualPlot' has been deleted from: SASUSER.TEMPLAT
```

You can run the following step to delete the entire Sasuser.Templat store of customized templates:

```
ods path sashelp.tmplmst(read);
proc datasets library=sasuser nolist;
   delete templat(memtype=itemstor);
run;
ods path sasuser.templat(update) sashelp.tmplmst(read);
```

Graph Template Modification Macro

You can use the **%ModTmplt** autocall macro to insert BY line information, titles, and footnotes in ODS Graphics. You can also use it to remove titles and perform other template modifications. For more information about this macro, see Kuhfeld (2009).

You do not have to include autocall macros (for example, with a **%include** statement). You can call them directly once they are properly installed. If your site has installed the autocall libraries that SAS supplies and uses the standard configuration of SAS supplied software, you need to ensure that the SAS system option MAUTOSOURCE is in effect to begin using the autocall macros. For more information about autocall libraries, see the *SAS Macro Language: Reference*. For more information about installing autocall macros, consult your host documentation.

The **%ModTmplt** macro has the following options:

BY=*by-variable-list*
: specifies the list of BY variables. Also see BYLIST=. When graphs are produced (by default or when the STEPS= value contains 'G'), you must specify the BY= option. Otherwise, when you are only modifying the template, you do not need to specify the BY= option. **NOTE:** This option has been rendered obsolete by the BYLINE= option in the ODS GRAPHICS statement.

BYLIST=*by-statement-list*
: specifies the full syntax of the BY statement. You can specify a full BY statement syntax including the DESCENDING or NOTSORTED options. If only BY variables are needed, specify only BY=. If you also need options, then specify the BY variables in the BY= option and the full syntax in the BYLIST= option (for example, specify BY=A B and BYLIST=A DESCENDING B). **NOTE:** This option has been rendered obsolete by the BYLINE= option in the ODS GRAPHICS statement.

DATA=*SAS-data-set*
: specifies the input SAS data set. If you do not specify the DATA= option, the macro uses the most recently created SAS data set.

FILE=*filename*
specifies the file in which to store the original templates. This is a temporary file. You can specify either a quoted filename or the name from a FILENAME statement that you provide before you call the macro. The default is *"template.txt"*.

OPTIONS=*options*
specifies one or more of the following options (case is ignored):

LOG
displays a note in the SAS log when each BY group has finished.

FIRST
adds the ENTRYTITLE or ENTRYFOOTNOTE statements as the first titles or footnotes. By default, the statements are added after the last titles or footnotes. Most graph templates provided by SAS do not use footnotes; so this option usually affects only entry titles.

NOQUOTES
specifies that the values of the system titles and footnotes are to be moved to the ENTRYTITLE or ENTRYFOOTNOTE statements without the outer quotation marks. With OPTIONS=NOQUOTES, you can specify options in the titles or footnotes in addition to the text. However, you must ensure that you quote the text that provides the actual title or footnote.

The following is an example of an ordinary footnote:

```
footnote "My Footer";
```

With this FOOTNOTE statement and without OPTIONS=NOQUOTES, the macro creates the following ENTRYFOOTNOTE statement:

```
entryfootnote "My Footer";
```

The following footnotes are used with OPTIONS=NOQUOTES:

```
footnote 'halign=left "My Footer"';
footnote2 '"My Second Footer"';
```

With these FOOTNOTE statements and OPTIONS=NOQUOTES, the macro creates the following ENTRYFOOTNOTE statements:

```
entryfootnote halign=left "My Footer";
entryfootnote "My Second Footer";
```

REPLACE
replaces the unconditionally added entry titles and entry footnotes in the templates (those that are not part of IF or ELSE statements) with the system titles and footnotes. The system titles and footnotes are those that are specified in the TITLE or FOOTNOTE statements. You can instead use the TITLES=*SAS-data-set* option to specify titles and footnotes with a data set. If OPTIONS=REPLACE is specified, then OPTIONS=TITLES is ignored.

SOURCE
displays the generated source code. By default, the template source code is not displayed.

TITLES
displays the system titles and footnotes with the graphs. The system titles and footnotes are those that are specified in the TITLE or FOOTNOTE statements. You can instead use the TITLES=*SAS-data-set* option to specify titles and footnotes with a data set. If you also specify OPTIONS=FIRST, the system titles and footnotes are inserted before the previously existing entry titles and entry footnotes in the templates. Otherwise, they are inserted at the end.

You can specify OPTIONS=TITLES or OPTIONS=REPLACE, or insert BY lines, or do both. If you do both, and you do not like where the BY line is inserted relative to your titles and footnotes, just specify OPTIONS=NOQUOTES and _ByLine0 to place the BY line wherever you choose. The following TITLE statements illustrate:

```
title1 '"My First Title"';
title2 '_byline0';
title3 '"My Last Title"';
```

Also, you can embed BY information in a title or a footnote, again with OPTIONS=NOQUOTES. For example:

```
title '"Spline Fit By Sex, " _byline0';
```

When _ByLine0 is specified in any of the titles or footnotes, then the usual BY line is not added.

The following example removes all titles and footnotes:

```
footnote;
title;
%modtmplt(options=replace, template=Stat.Transreg.Graphics, steps=t)
```

STATEMENT=*entry-statement-fragment*
specifies the statement that contains the BY line that gets added to the template along with any statement options. The default is **Statement=EntryFootNote halign=left TextAttrs=GraphValueText**. Other examples include:

```
Statement=EntryTitle
Statement=EntryFootNote halign=left TextAttrs=GraphLabelText
```

STEPS=*steps*
specifies the macro steps to run. Case and white space are ignored. the macro modifies the templates (when 'T' is specified), produces the graphs for each BY group (when 'G' is specified), and deletes the modified templates (when 'D' is specified). The default is STEPS=TGD. You can instead have it perform a subset of these three tasks by specifying a subset of terms in the STEPS= option.

When you use the **%ModTmplt** macro to add BY lines, you usually do not need to delete the templates before you run your procedure again in the normal way. The template modification inserts the BY line through a macro variable and an MVAR statement. When the macro variable _ByLine0 is undefined, the ENTRYTITLE or ENTRYFOOTNOTE statement drops out as if it were not there at all.

STMTOPTS1= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS2= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS3= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS4= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS5= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS6= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS7= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS8= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS9= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS10= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >

These ten options add or replace options in up to 10 selected statements. The following example illustrates:

```
%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=t,
          stmtopts1=.  add  discretelegend  autoalign=(topleft),
          stmtopts2=1  add  densityplot     legendlabel='Normal Density',
          stmtopts3=2  add  densityplot     legendlabel='Kernel Density',
          stmtopts4=1  add  overlay         yaxisopts=(griddisplay=on)
                            yaxisopts=(label='Normal and Kernel Density'))

proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
run;

%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=d)
```

These options require you to specify a series of values. The first value is the statement number (or missing to modify options on all statements that match the statement name). The second value is: ADD, REPLACE, DELETE, BEFORE, or AFTER. When the second value is ADD or REPLACE, it controls whether you add new options or replace existing options. Alternatively, the second value can be BEFORE or AFTER to add a new statement before or after the named statement. When the value is DELETE, the corresponding statement is deleted. The third value is a statement name. All remaining options are options for the statement named by the third value (with ADD and REPLACE) or for a new statement (with BEFORE and AFTER). In the STMTOPTS1= example, an option is added to all DISCRETELEGEND statements. In the STMTOPTS2= example, an option is added to the first DensityPlot statement. In the STMTOPTS4= example, an option is added to the LAYOUT OVERLAY statement. In most cases, the statement name is the first name that begins the statement. The LAYOUT statement is an exception. In the case of layouts, specify the second name (OVERLAY, GRIDDED, LATTICE, and so on) for the third value. Note that a statement such as **`if (expression) EntryTitle...;`** is an IF statement not an ENTRYTITLE statement.

If an option is specified multiple times on a GTL statement, the last specification overrides previous specifications. Hence, you do not need to know and respecify all of the options. You can just add an option to the end, and it overrides the previous value. You can use these options only to modify

statements that contain a slash, and only to modify the options that come after the slash. Note that in STMTOPTS4=, the YAXISOPTS= option is specified twice. It could have been equivalently specified once as follows:

```
yaxisopts=(griddisplay=on label='Normal and Kernel Density'))
```

The actual specification adds the GRIDDISPLAY=ON to the Y axis options (which by default has only a label specification). The old label is unchanged until the LABEL= option in the second YAXISOPTS= specification overrides it. In other words, YAXISOPTS=(GRIDDISPLAY=ON) augments the old YAXISOPTS= option; it does not replace it.

The following steps delete the legend and instead provide a footnote:

```
%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=t,
          stmtopts1=. delete discretelegend,
          stmtopts2=1 after  begingraph entryfootnote
                      textattrs=GraphLabelText(color=cx445694) 'Normal  '
                      textattrs=GraphLabelText(color=cxA23A2E) 'Kernel')

proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
run;

%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=d)
```

TEMPLATE=*SAS-template*

specifies the name of the template to modify. You can specify just the first few levels to modify a series of templates. For example, to modify all of PROC REG's graph templates, specify TEMPLATE=**Stat.Reg.Graphics**. This option is required.

TITLES=*SAS-data-set*

specifies a data set that contains titles or footnotes or both. By default, when the system titles or footnotes are used (when OPTIONS=TITLES or OPTIONS=REPLACE is specified), PROC SQL is used to determine the titles and footnotes. You can instead create this data set yourself so that you can set the graph titles independently from the system titles and footnotes. The data set must contain two variables: Type (Type='T' for titles and Type='F' for footnotes), and Text, which contains the titles and footnotes. Other variables are ignored. Specify the titles and footnotes in the order in which you want them to appear.

TITLEOPTS=*entry-statement-options*

specifies the options for system titles and footnotes. For example, you can specify the HALIGN= and TEXTATTRS= options as in the STATEMENT= option. By default, no title options are used. With OPTIONS=NOQUOTES, you can specify options individually.

Examples of ODS Graphics Template Modification

Example 22.1: Customizing Graphs Through Template Changes

This example shows how to use PROC TEMPLATE to customize the appearance and content of an ODS graph. It is divided into several parts; each part illustrates a different aspect of the template that you can easily change. You are never required to change a template, but you can if you want to change aspects of the plot.

Modifying Graph Titles and Axis Labels

This section illustrates the discussion in the section "Graph Templates" on page 722 in the context of changing the default title and Y-axis label for a Q-Q plot created with PROC ROBUSTREG. The data set Stack is created by the following statements:

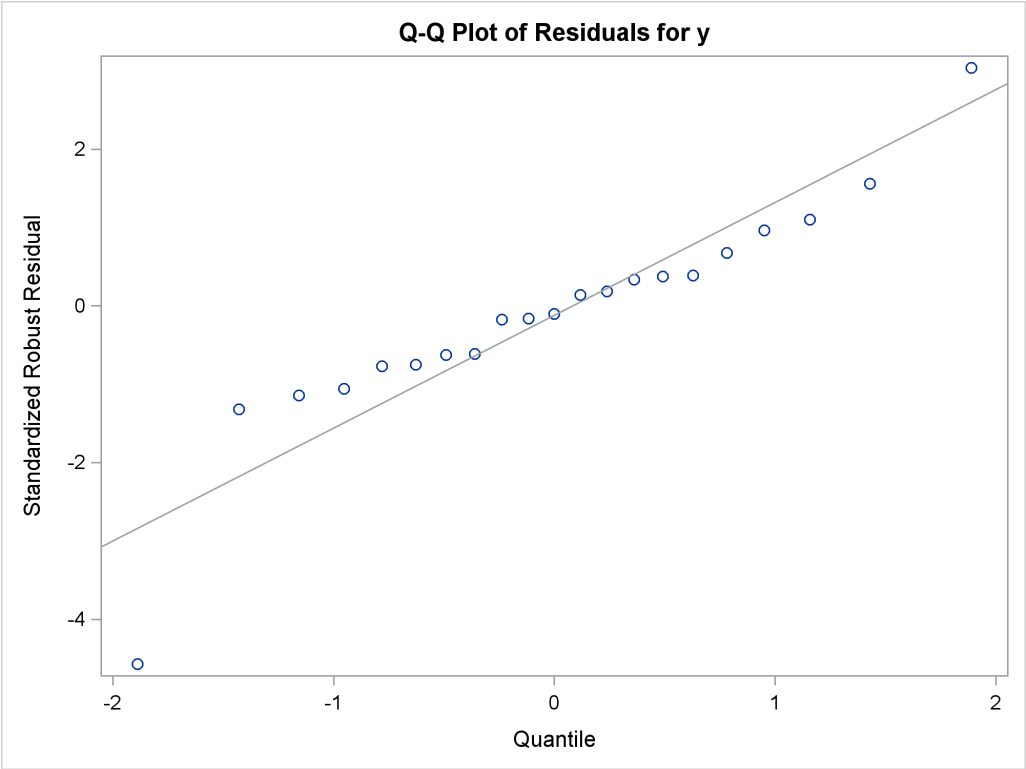
```
data stack;
   input  x1 x2 x3 y @@;
   datalines;
80  27  89  42    80  27  88  37    75  25  90  37    62  24  87  28
62  22  87  18    62  23  87  18    62  24  93  19    62  24  93  20
58  23  87  15    58  18  80  14    58  18  89  14    58  17  88  13
58  18  82  11    58  19  93  12    50  18  89   8    50  18  86   7
50  19  72   8    50  19  79   8    50  20  80   9    56  20  82  15
70  20  91  15
;
```

The following statements request a Q-Q plot for robust residuals created by PROC ROBUSTREG:

```
ods trace on;
ods graphics on;

proc robustreg data=stack plots=qqplot;
   ods select QQPlot;
   model y = x1 x2 x3;
run;

ods trace off;
```

The Q-Q plot is shown in Output 22.1.1.

Output 22.1.1 Default Q-Q Plot from PROC ROBUSTREG



The ODS TRACE ON statement requests a record of all the ODS output objects created by PROC ROBUSTREG. The trace output is as follows:

```
Output Added:
-------------
Name:        QQPlot
Label:       Residual Q-Q Plot
Template:    Stat.Robustreg.Graphics.QQPlot
Path:        Robustreg.DiagnosticPlots.QQPlot
-------------
```

ODS Graphics creates the Q-Q plot from an ODS data object named QQPlot and a graph template named **`Stat.Robustreg.Graphics.QQPlot`**, which is the default template provided by SAS. Default templates that SAS supplies are saved in the Sashelp.Tmplmst template store (see the section "Graph Templates" on page 722).

To display the default template definition, open the Templates window by typing **odstemplates** (or **odst** for short) in the command line. Expand Sashelp.Tmplstat and click the **Stat** folder. Next, open the **Robustreg** folder and then open the **Graphics** folder. Then right-click the **`QQPlot`** template icon and select **Open**. This opens the Template Browser. You can copy this template to an editor to edit it.

Alternatively, you can submit the following statements to display the **`QQPlot`** template definition in the SAS log:

```
proc template;
   source Stat.Robustreg.Graphics.QQPlot;
run;
```

The SOURCE statement specifies the template name. You can copy and paste the template source into the Program Editor and modify it. The template, with a PROC TEMPLATE and RUN statement added, is shown next:

```
proc template;
   define statgraph Stat.Robustreg.Graphics.QQPlot;
      notes "Q-Q Plot for Standardized Robust Residuals";
      dynamic _DEPLABEL Residual _byline_ _bytitle_ _byfootnote_;
      BeginGraph;
         ENTRYTITLE "Q-Q Plot of Residuals for " _DEPLABEL;
         Layout Overlay / yaxisopts=(label="Standardized Robust Residual")
            xaxisopts=(label="Quantile");
            SCATTERPLOT y=eval (SORT(DROPMISSING(RESIDUAL))) x=eval (
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) / primary=true
               markerattrs=GRAPHDATADEFAULT
               rolename=(q=eval(
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) s=eval (SORT(DROPMISSING(RESIDUAL))))
               tip=(q s) tiplabel=(q= "Quantile" s="Residual");
            lineparm slope=eval (STDDEV(RESIDUAL)) Y=eval (MEAN(RESIDUAL)) X=0 /
               clip=false lineattrs=GRAPHREFERENCE extend=true;
         EndLayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
run;
```

In the template, the default title of the Q-Q plot is specified by the ENTRYTITLE statement. The variable _DepLabel is a dynamic variable that provides the name of the dependent variable in the regression analysis. In this case, the name is y. In this template, the label for the axes are specified by the LABEL= suboption of the YAXISOPTS= option for the LAYOUT OVERLAY statement. In other templates, the axis labels come from the column labels of the X-axis and Y-axis columns of the data object. You can see these labels by specifying ODS OUTPUT with the plot data object and running PROC CONTENTS with the resulting SAS data set.

Suppose you want to change the title to “Analysis of Residuals”, and you want the Y-axis label to display the name of the dependent variable. First, replace the ENTRYTITLE statement with the following statement:

```
entrytitle "Analysis of Residuals";
```

Next, replace the LABEL= suboption with the following:

```
label=("Standardized Robust Residual for " _DEPLABEL)
```

You can use dynamic text variables such as _DepLabel in any text element.

You can then submit the modified template definition as you would any SAS program, for example, by selecting **Submit** from the **Run** menu. After submitting the PROC TEMPLATE statements, you should see the following message in the SAS log:

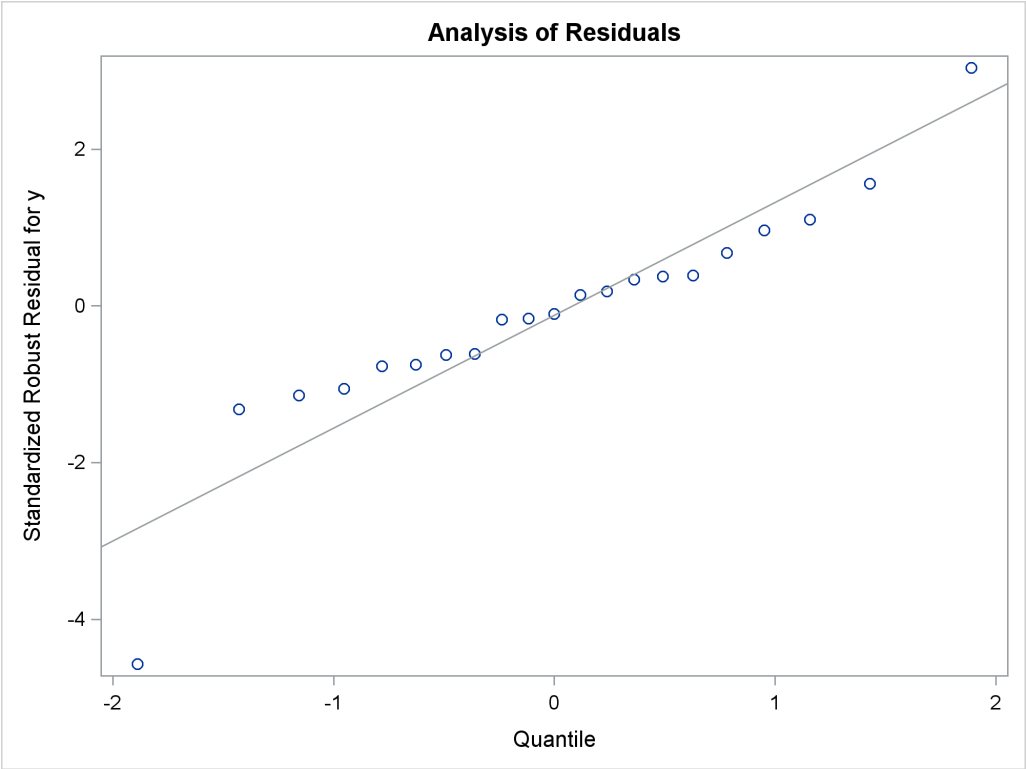
```
NOTE: STATGRAPH 'Stat.Robustreg.Graphics.QQPlot' has been
  saved to: SASUSER.TEMPLAT
```

For more information about graph templates and the graph template language, see the section "Graph Templates" on page 722.

Finally, resubmit the PROC ROBUSTREG statements to display the Q-Q plot created with your modified template. The following statements create Output 22.1.2:

```
proc template;
   define statgraph Stat.Robustreg.Graphics.QQPlot;
      notes "Q-Q Plot for Standardized Robust Residuals";
      dynamic _DEPLABEL Residual _byline_ _bytitle_ _byfootnote_;
      BeginGraph;
         entrytitle "Analysis of Residuals";
         Layout Overlay /
            yaxisopts=(label=("Standardized Robust Residual for " _DEPLABEL))
            xaxisopts=(label="Quantile");
            SCATTERPLOT y=eval (SORT(DROPMISSING(RESIDUAL))) x=eval (
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) / primary=true
               markerattrs=GRAPHDATADEFAULT
               rolename=(q=eval(
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) s=eval (SORT(DROPMISSING(RESIDUAL))))
               tip=(q s) tiplabel=(q= "Quantile" s="Residual");
            lineparm slope=eval (STDDEV(RESIDUAL)) Y=eval (MEAN(RESIDUAL)) X=0 /
               clip=false lineattrs=GRAPHREFERENCE extend=true;
         EndLayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
run;

proc robustreg data=stack plots=qqplot;
   ods select QQPlot;
   model y = x1 x2 x3;
run;
```

Output 22.1.2 Q-Q Plot with Modified Title and Y-Axis Label



If you have not changed the default template search path, the modified template **QQPlot** is used automatically because Sasuser.Templat occurs before Sashelp.Tmplmst in the ODS search path. For more information about the template search path and the ODS PATH statement, see the sections "Saving Customized Templates" on page 731, "Using Customized Templates" on page 731, and "Reverting to the Default Templates" on page 732.

You do not need to rerun the PROC ROBUSTREG analysis after you modify a graph template if you have stored the plot in an ODS document. After you modify your template, you can submit the PROC DOCUMENT statements in Example 21.4 in Chapter 21, "Statistical Graphics Using ODS," to replay the Q-Q plot with the modified template. You can run the following statements to revert to the default template:

```
proc template;
   delete Stat.Robustreg.Graphics.QQPlot / store=sasuser.templat;
run;
```

Modifying Colors, Line Styles, and Markers

This section shows you how to customize colors, line attributes, and marker symbol attributes by modifying a graph template. In the **QQPlot** template definition shown previously, the SCATTERPLOT statement specifies a scatter plot of normal quantiles versus ordered standardized residuals. The attributes of the marker symbol in the scatter plot are specified by: **MarkerAttrs=GraphDataDefault**. This is a reference to the style element **GraphDataDefault**. For more information, see the section "ODS Style Elements and Attributes" on page 649 in Chapter 21, "Statistical Graphics Using ODS."

The actual value of the marker symbol depends on the style that you are using. In this case, since the HTMLBLUE style is used, the marker symbol is a circle. You can specify a filled cir-

cle as the marker symbol by overriding the symbol portion of the style specification as follows: **MarkerAttrs=GraphDataDefault(symbol=CircleFilled)**.

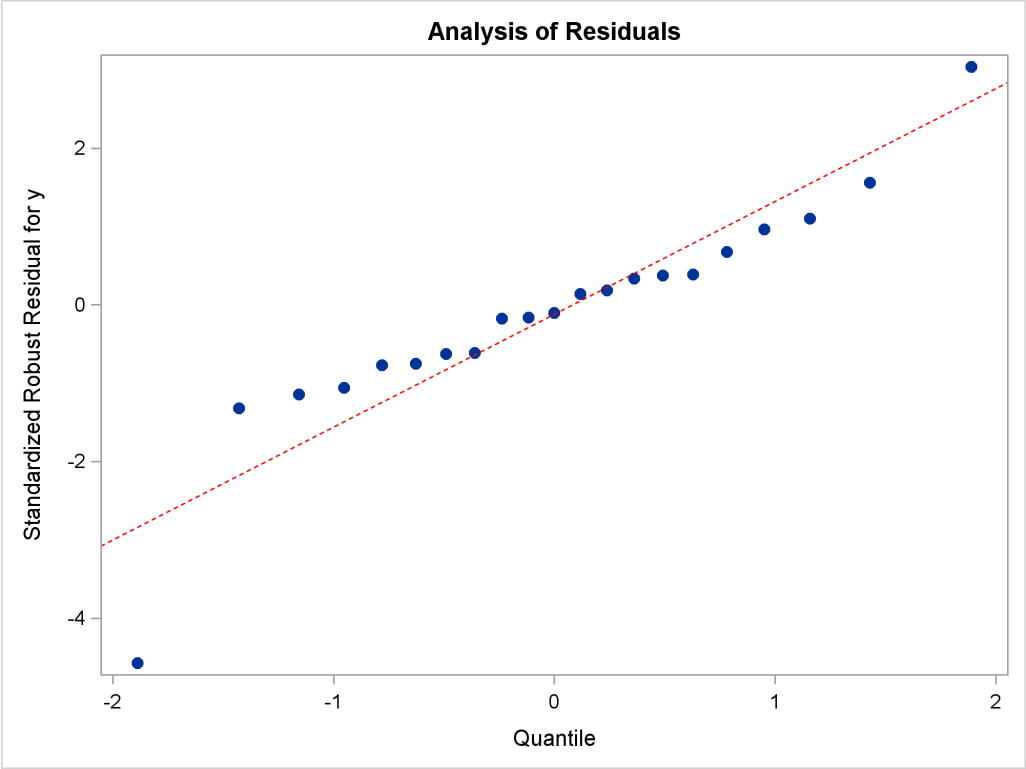
The value of the SYMBOL= option can be any valid marker symbol or a reference to a style attribute of the form **style-element:attribute**. It is recommended that you use style attributes because they are chosen to provide consistency and appropriate emphasis based on display principles for statistical graphics. If you specify values directly in a template, you are overriding the style and you run the risk of creating a graph that is inconsistent with the style template. For more information about the syntax of the Graph Template Language and style elements for graphics, see the *SAS Graph Template Language: Reference* and the *SAS Output Delivery System: User's Guide*.

Similarly, you can change the line color and pattern with the LINEATTRS= option in the LINEPARM statement. The LINEPARM statement displays a straight line specified by slope and intercept parameters. The following option changes the color of the line to red and the line pattern to dashed, by overriding those aspects of the style specification: **LineAttrs=GraphReference(color=red pattern=dash)**. To see the results, submit the modified template definition and the PROC ROBUSTREG statements as follows to create Output 22.1.3:

```
proc template;
   define statgraph Stat.Robustreg.Graphics.QQPlot;
      notes "Q-Q Plot for Standardized Robust Residuals";
      dynamic _DEPLABEL Residual _byline_ _bytitle_ _byfootnote_;
      BeginGraph;
         entrytitle "Analysis of Residuals";
         Layout Overlay /
            yaxisopts=(label=("Standardized Robust Residual for " _DEPLABEL))
            xaxisopts=(label="Quantile");
            SCATTERPLOT y=eval (SORT(DROPMISSING(RESIDUAL))) x=eval (
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) / primary=true
               markerattrs=GraphDataDefault(symbol=CircleFilled)
               rolename=(q=eval(
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) s=eval (SORT(DROPMISSING(RESIDUAL))))
               tip=(q s) tiplabel=(q= "Quantile" s="Residual");
            lineparm slope=eval (STDDEV(RESIDUAL)) Y=eval (MEAN(RESIDUAL)) X=0 /
               clip=false lineattrs=GraphReference(color=red pattern=dash)
               extend=true;
         EndLayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
run;
```

```
ods graphics on;

proc robustreg data=stack plots=qqplot;
   ods select QQPlot;
   model y = x1 x2 x3;
run;
```

Output 22.1.3 Q-Q Plot with Modified Marker Symbols and Line



Alternatively, you can replay the plot with PROC DOCUMENT, as in Example 21.4 in Chapter 21, "Statistical Graphics Using ODS."

Modifying Tick Marks and Grid Lines

This section illustrates how to modify axis tick marks and control grid lines. For example, you can specify the following statement to request tick marks ranging from –4 to 2 in the Y-axis:

```
layout Overlay / yaxisopts=(linearopts=(tickvaluelist=(-4 -3 -2 -1 0 1 2)));
```

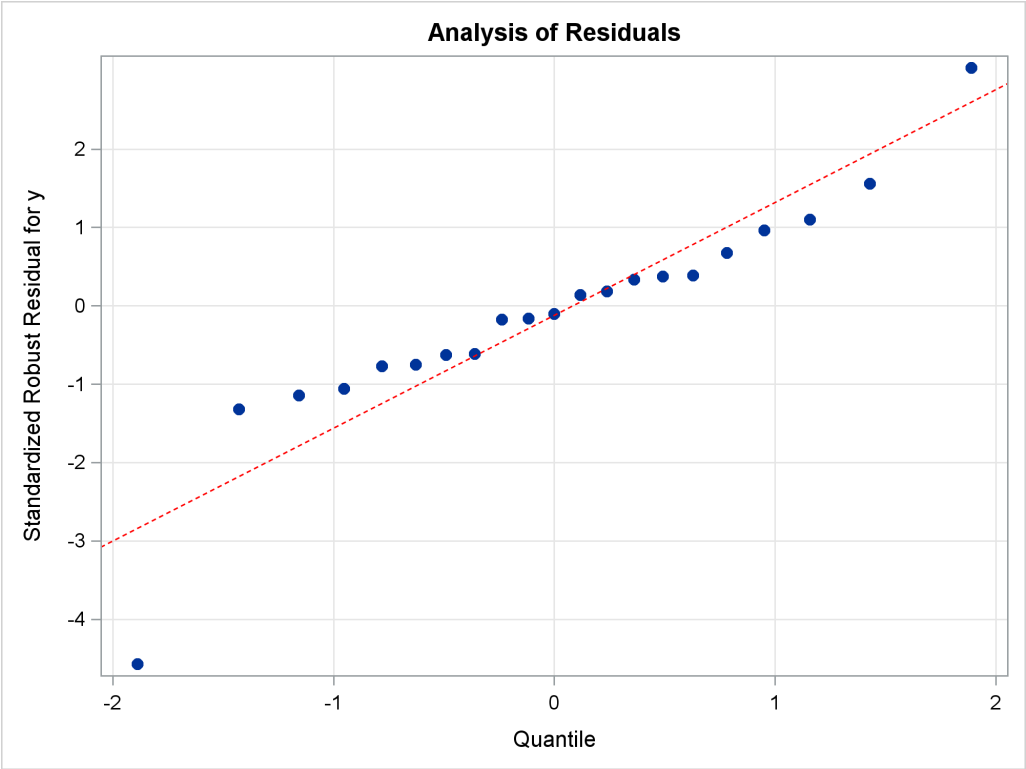
The LINEAROPTS= option is used for standard linearly scaled axes (as opposed to log-scaled axes). You use the TICKVALUELIST= to specify the tick marks.

You can control the grid lines by using the GRIDDISPLAY= suboption in the YAXISOPTS= option. Typically, you specify either GRIDDISPLAY=AUTO_OFF (grid lines are not displayed unless the **GraphGridLines**

element in the current style contains **DisplayOpts="ON"**) or GRIDDISPLAY=AUTO_ON (grid lines are displayed unless the **GraphGridLines** element in the current style contains **DisplayOpts="OFF"**). Here, the template is modified by specifying GRIDDISPLAY=AUTO_ON for both axes. The following statements produce Output 22.1.4:

```
proc template;
   define statgraph Stat.Robustreg.Graphics.QQPlot;
      notes "Q-Q Plot for Standardized Robust Residuals";
      dynamic _DEPLABEL Residual _byline_ _bytitle_ _byfootnote_;
      BeginGraph;
         entrytitle "Analysis of Residuals";
         Layout Overlay /
            yaxisopts=(gridDisplay=Auto_On
                       linearopts=(tickvaluelist=(-4 -3 -2 -1 0 1 2))
                       label=("Standardized Robust Residual for " _DEPLABEL))
            xaxisopts=(gridDisplay=Auto_On label="Quantile");
            SCATTERPLOT y=eval (SORT(DROPMISSING(RESIDUAL))) x=eval (
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) / primary=true
               markerattrs=GraphDataDefault(symbol=CircleFilled)
               rolename=(q=eval(
               PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
               N(RESIDUAL)))) s=eval (SORT(DROPMISSING(RESIDUAL))))
               tip=(q s) tiplabel=(q= "Quantile" s="Residual");
            lineparm slope=eval (STDDEV(RESIDUAL)) Y=eval (MEAN(RESIDUAL)) X=0 /
               clip=false lineattrs=GraphReference(color=red pattern=dash)
               extend=true;
         EndLayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      EndGraph;
   end;
run;

ods graphics on;

proc robustreg data=stack plots=qqplot;
   ods select QQPlot;
   model y = x1 x2 x3;
run;
```

Output 22.1.4 Q-Q Plot with Modified Y-Axis Tick Marks and Grids



You can restore the default template by running the following step:

```
proc template;
   delete Stat.Robustreg.Graphics.QQPlot / store=sasuser.templat;
run;
```

For more information about grid lines, see the section "Modifying the Style to Show Grid Lines" on page 746.

Modifying the Style to Show Grid Lines

The section "Modifying Tick Marks and Grid Lines" on page 744 explains that grid lines in graphs are controlled both by template options and by the style. Some graphs never display grid lines because they would interfere with the display. Some graphs always display grid lines because they are a critical part of the display. In both cases, grid control is so important that the template writer is not willing to give control to the style. If you want to change the grid display setting for these graphs, you must edit their templates. However, most templates let the style control the grid lines. Either they do not display grid lines unless the style forces them on, or they display grid lines unless the style forces them off. The HTMLBLUE, STATISTICAL, DEFAULT, and most other styles use the setting **`DisplayOpts = "Auto"`**. Then templates that specify GRIDDISPLAY=AUTO_OFF (the default) do not display grid lines, and templates that specify GRIDDISPLAY=AUTO_ON do display grid lines. You can easily make a new style with **`DisplayOpts = "On"`** or **`DisplayOpts = "Off"`** if you would prefer to see grid lines more or less often. This example shows how to set **`DisplayOpts = "On"`**.

First, you need to find the part of the style that enables grid lines. The following step displays the HTMLBLUE style and uses the EXPAND option to display its parent styles, STATISTICAL and DEFAULT:

```
proc template;
   source Styles.HTMLBlue / expand;
run;
```

The advantage of displaying all three styles together is that you can do one search of the results. If grids are defined in the HTMLBLUE style, you will find that first. Otherwise, you will first find the definition in one of the parent styles. An abridged version of the results follows:

```
. . .
class GraphGridLines /
   displayopts = "auto"
   linethickness = 1px
   linestyle = 1
   contrastcolor = GraphColors('ggrid')
   color = GraphColors('ggrid');
. . .
```

You can use this to create a new style that inherits from the HTMLBLUE style, but sets the display options for grids to ON, as in the following example:

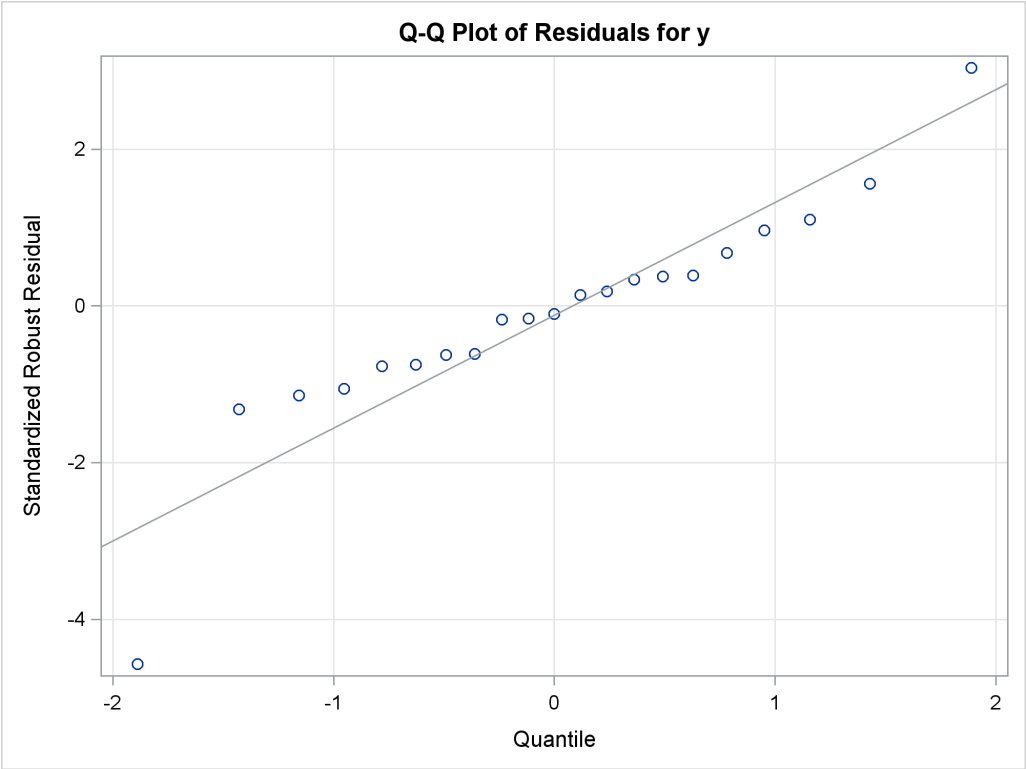
```
proc template;
   define style Styles.MyGrids;
      parent=styles.HTMLBlue;
      class GraphGridLines /
         displayopts = "on"
         linethickness = 1px
         linestyle = 1
         contrastcolor = GraphColors('ggrid')
         color = GraphColors('ggrid');
   end;
run;
```

You can use this new style as in the following example:

```
ods graphics on;
ods listing style=mygrids;

proc robustreg data=stack plots=qqplot;
   ods select QQPlot;
   model y = x1 x2 x3;
run;
```

The preceding statements produce Output 22.1.5, which shows the Q-Q plot with grid lines displayed. The default graph template that SAS supplies is used because the custom template created in the section "Modifying Tick Marks and Grid Lines" on page 744 is deleted at the end of that section.

Output 22.1.5 A Style that Makes Grid Lines the Typical Default



Example 22.2: Adding Equations and Special Characters to Fit Plots

This example shows how to run the REG and TRANSREG procedures to get fit plots. The R square, mean, and equation for the regression model are output to data sets, and the results are processed and then displayed in subsequent fit plots. This example also illustrates Unicode and how to add special characters to graphs (for example, $\hat{\mu}$ and R^2). The Unicode Consortium `http://unicode.org/` provides a list of character codes at `http://www.unicode.org/charts/charindex.html`.

Simple Linear Regression

The following step runs PROC REG to fit a simple regression model and creates Output 22.2.2 and Output 22.2.1:

```
ods graphics on;
ods trace on;

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

Output 22.2.1 PROC REG Output

The REG Procedure
Model: MODEL1
Dependent Variable: Weight

Number of Observations Read	19
Number of Observations Used	19

Analysis of Variance

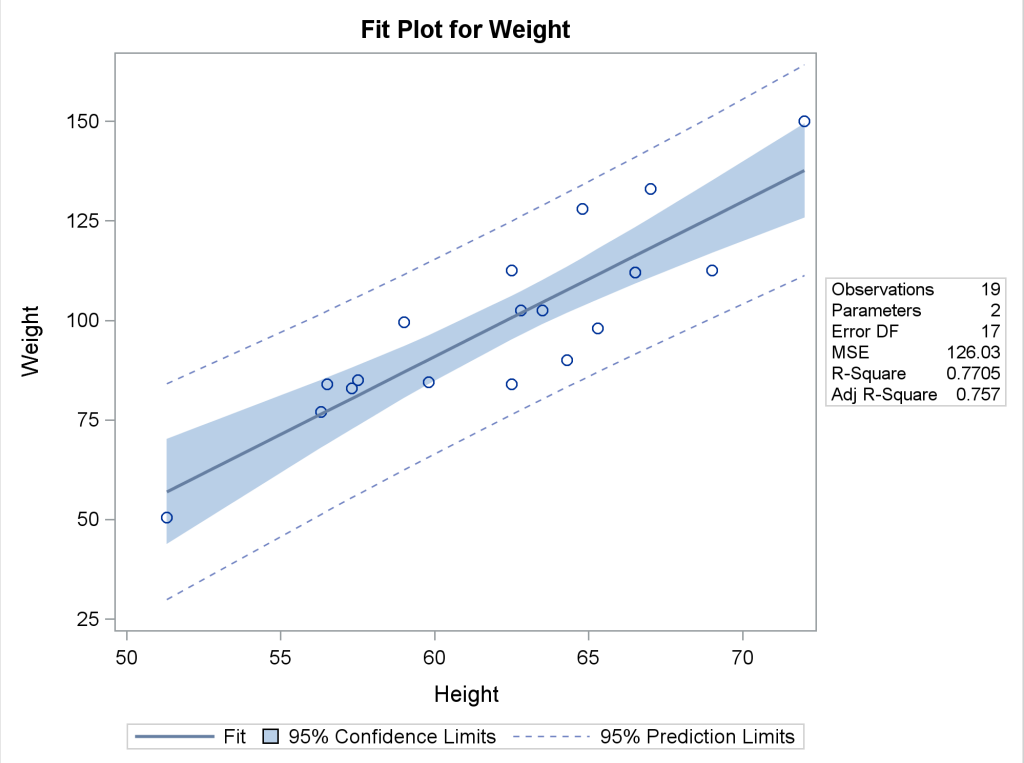
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	7193.24912	7193.24912	57.08	<.0001
Error	17	2142.48772	126.02869		
Corrected Total	18	9335.73684			

Root MSE	11.22625	**R-Square**	0.7705
Dependent Mean	100.02632	**Adj R-Sq**	0.7570
Coeff Var	11.22330		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-143.02692	32.27459	-4.43	0.0004
Height	1	3.89903	0.51609	7.55	<.0001

Output 22.2.2 PROC REG Fit Plot



The fit statistics table following the "Analysis of Variance" table displays the R square and the mean. The last table displays the parameter estimates. This information is produced and processed for inclusion in the fit plot as follows:

```
proc reg data=sashelp.class;
   ods output fitstatistics=fs ParameterEstimates=c;
   model weight = height;
run;

data _null_;
   set fs;
   if _n_ = 1 then call symputx('R2'  , put(nvalue2, 4.2)   , 'G');
   if _n_ = 2 then call symputx('mean', put(nvalue1, best6.), 'G');
run;

data _null_;
   set c;
   length s $ 200;
   retain s ' ';
   if _n_ = 1 then
      s = trim(dependent) || ' = ' ||       /* dependent =             */
          put(estimate, best5. -L);         /* intercept               */
   else if abs(estimate) > 1e-8 then        /* skip zero coefficients  */
      s = catx(' ', s,                      /* string so far           */
               ifc(estimate < 0, '-', '+'), /* subtract (-) or add (+) */
               put(abs(estimate), best5.),  /* abs(coefficient)        */
               variable);                   /* variable name           */
   call symputx('formula', s, 'G');
run;
```

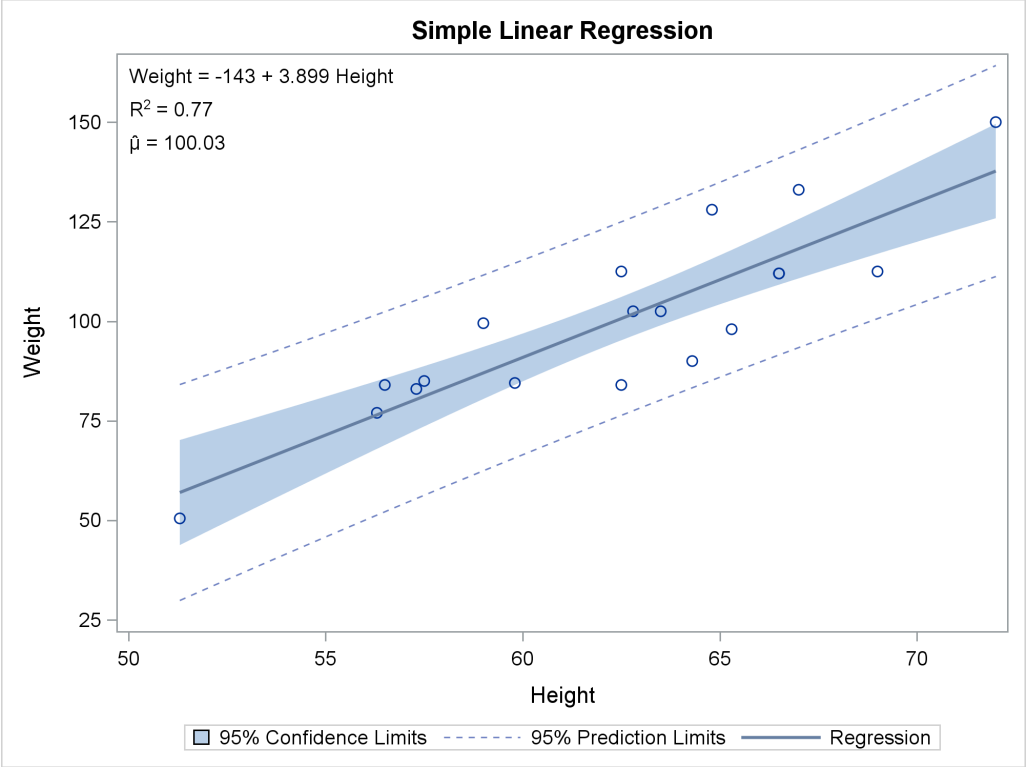
Two SAS data sets are made from the tabular output, and the R square, mean, and equation for the regression model are stored in macro variables. The following step uses PROC SGPLOT with an INSET statement to display the linear fit plot along with the R square, mean, and equation for the regression model:

```
proc sgplot data=sashelp.class;
   title 'Simple Linear Regression';
   inset "&formula"
         "R(*ESC*){sup '2'} = &r2"
         "(*ESC*){unicode mu}(*ESC*){unicode hat} = &mean" / position=topleft;
    reg y=weight x=height / clm cli;
run;
```

The results are displayed in Output 22.2.3.

Each separate string in the INSET statement is displayed in a separate line. The first string is the formula, which is generated in the second DATA step. The next string is the R square, and it consists of an 'R', an escaped superscript 2, and the value of R square (which is stored in a macro variable). The string for the mean consists of two Unicode specifications, one for the Greek letter μ, and one to put a hat over it. These special character specifications appear in quotes and are escaped with **(*ESC*)** so that they are processed as special characters rather than as literal text. Typically, you must escape special characters in quotes, and not escape them when they are not in quotes. See the section "Unicode and Special Characters" on page 756 for a list of a few of the more commonly used Unicode characters.

Output 22.2.3 Fit Plot from PROC SGPLOT with Equation



The same information can be added to the graph that PROC REG produces by adding the following statements to the PROC REG template for a fit plot:

```
mvar formula;

layout gridded / autoalign=(topleft topright bottomleft
                            bottomright);
   entry halign=left formula;
   entry halign=left "R"{sup '2'} " = " eval(put(_rsquare, 4.2));
   entry halign=left "(*ESC*){unicode mu}(*ESC*){unicode hat} = "
         eval(put(_depmean, best6.))
         / textattrs=GraphValueText
                     (family=GraphUnicodeText:FontFamily);
endlayout;
```

The MVAR statement names macro variables whose values are added to the graph. The MVAR statement is added to the PROC REG fit plot template near the top. The LAYOUT GRIDDED block creates a table that consists of the equation, R square, and mean. The LAYOUT GRIDDED block is added to the PROC REG fit plot template inside the LAYOUT OVERLAY block. The option **`autoalign=(topleft topright bottomleft bottomright)`** is used to position the table in a part of the graph that is open, first trying the top left corner.

In this example, in the LAYOUT GRIDDED block, two dynamic variables for R square, and the mean are used instead of the macro variables that were made in previous steps. The origin of the names of the two dynamic variables that are used in this example are revealed in the next step when the source code for the PROC REG fit plot is displayed. The first ENTRY statement creates a text line for the formula and left-justifies it. The

second ENTRY statement creates the R square line. It consists of a literal 'R', a specification for a superscript of 2 ({**sup 2**}), an equal sign surrounded by spaces, and the formatted value of the dynamic variable with the R square. The third ENTRY statement creates the mean line. It consists of two Unicode specifications, one for the Greek letter μ, and one to put a hat over it. These special character specifications appear in quotes (unlike the {**sup 2**}) and are escaped with **(*ESC*)** so that they are processed as special characters rather than as literal text. Typically, you must escape special characters in quotes, and not escape them when they are not in quotes. Note that **sup** along with **sub** (subscript) must not appear in quotes in the GTL, but they can appear in quotes in PROC SGPLOT (as was previously shown). The option **textattrs=GraphValueText (family=GraphUnicodeText:FontFamily)** is specified to ensure that a font that recognizes Unicode characters is used. See the section "Unicode and Special Characters" on page 756 for a list of a few of the more commonly used Unicode characters.

You can use the trace information from the PROC REG step (not shown) and the following step to display the template for the fit plot:

```
proc template;
   source Stat.Reg.Graphics.Fit;
run;
```

Some of the results are as follows:

```
if (_SHOWRSQUARE^=0)
   entry halign=left "R-Square" / valign=top;
   entry halign=right eval (PUT(_RSQUARE,BEST6.)) / valign=top;
endif;

if (_SHOWDEPMEAN^=0)
   entry halign=left "Dependent Mean" / valign=top;
   entry halign=right eval (PUT(_DEPMEAN,BEST6.)) / valign=top;
endif;
```

The preceding results show that the dynamic variables _RSquare and _DepMean contain the R square and the mean of the dependent variable. The MVAR statement and the LAYOUT GRIDDED block can be added to the template, and in the interest of maximizing graph size, the table of statistics can be removed, creating the following template:

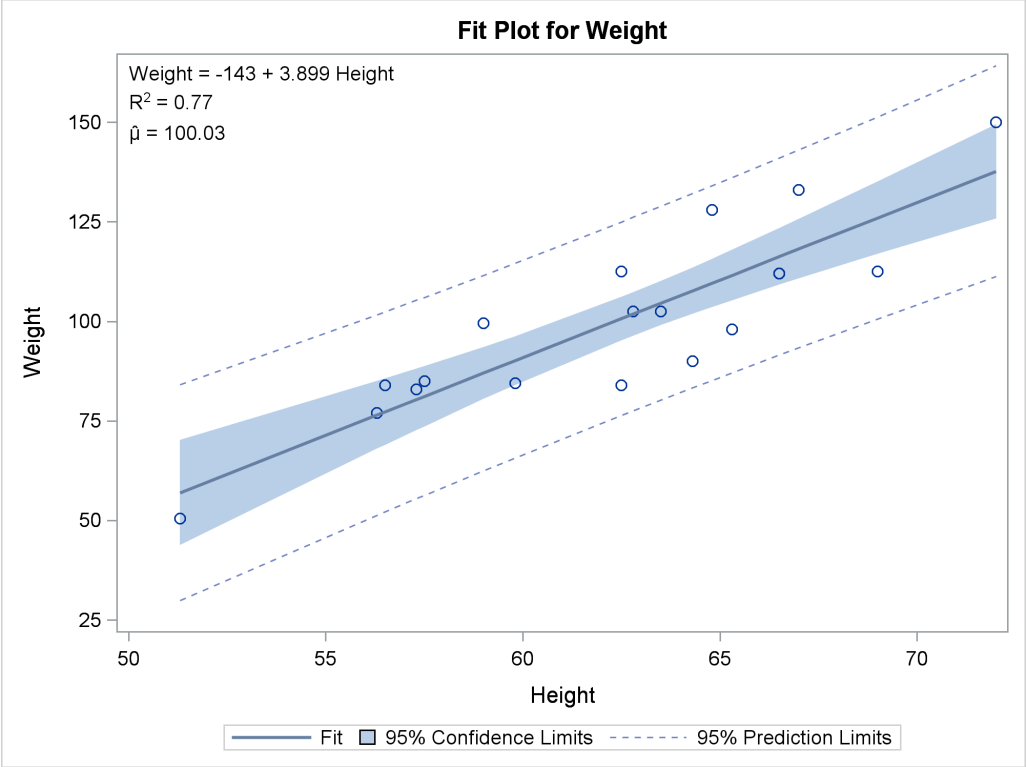
```
proc template;
   define statgraph Stat.Reg.Graphics.Fit;
      notes "Fit Plot";
      mvar formula;
      dynamic _DEPLABEL _DEPNAME _MODELLABEL _SHOWSTATS _NSTATSCOLS _SHOWNObs
         _SHOWTOTFREQ _SHOWNParm _SHOWEDF _SHOWMSE _SHOWRSquare _SHOWAdjRSq
         _SHOWSSE _SHOWDepMean _SHOWCV _SHOWAIC _SHOWBIC _SHOWCP _SHOWGMSEP
         _SHOWJP _SHOWPC _SHOWSBC _SHOWSP _NObs _NParm _EDF _MSE _RSquare
         _AdjRSq _SSE _DepMean _CV _AIC _BIC _CP _GMSEP _JP _PC _SBC _SP
         _PREDLIMITS _CONFLIMITS _XVAR _SHOWCLM _SHOWCLI _WEIGHT _SHORTXLABEL
         _SHORTYLABEL _TITLE _TOTFreq _byline_ _bytitle_ _byfootnote_;
      BeginGraph;
         entrytitle halign=left textattrs=GRAPHVALUETEXT _MODELLABEL
            halign=center textattrs=GRAPHTITLETEXT _TITLE " for " _DEPNAME;
         layout Overlay / yaxisopts=(label=_DEPLABEL shortlabel=_SHORTYLABEL)
            xaxisopts=(shortlabel=_SHORTXLABEL);
            if (_SHOWCLM=1)
```

```
            BANDPLOT limitupper=UPPERCLMEAN limitlower=LOWERCLMEAN x=_XVAR
               / fillattrs=GRAPHCONFIDENCE connectorder=axis
               name="Confidence" LegendLabel=_CONFLIMITS;
         endif;
         if (_SHOWCLI=1)
            if (_WEIGHT=1)
               SCATTERPLOT y=PREDICTEDVALUE x=_XVAR / markerattrs=(size=0)
                  datatransparency=.6 yerrorupper=UPPERCL
                  yerrorlower=LOWERCL name="Prediction"
                  LegendLabel=_PREDLIMITS;
            else
               BANDPLOT limitupper=UPPERCL limitlower=LOWERCL x=_XVAR /
                  display=(outline) outlineattrs=GRAPHPREDICTIONLIMITS
                  connectorder=axis name="Prediction"
                  LegendLabel=_PREDLIMITS;
            endif;
         endif;
         SCATTERPLOT y=DEPVAR x=_XVAR / markerattrs=GRAPHDATADEFAULT
            primary=true rolename=(_tip1=OBSERVATION _id1=ID1 _id2=ID2
            _id3=ID3 _id4=ID4 _id5=ID5)
            tip=(y x _tip1 _id1 _id2 _id3 _id4 _id5);
         SERIESPLOT y=PREDICTEDVALUE x=_XVAR / lineattrs=GRAPHFIT
            connectorder=xaxis name="Fit" LegendLabel="Fit";
         if (_SHOWCLI=1 OR _SHOWCLM=1)
            DISCRETELEGEND "Fit" "Confidence" "Prediction" / across=3
               HALIGN=CENTER VALIGN=BOTTOM;
         endif;
         layout gridded / autoalign=(topleft topright bottomleft
                                     bottomright);
            entry halign=left formula;
            entry halign=left "R"{sup '2'} " = " eval(put(_rsquare, 4.2));
            entry halign=left "(*ESC*){unicode mu}(*ESC*){unicode hat} = "
                  eval(put(_depmean, best6.))
                  / textattrs=GraphValueText
                              (family=GraphUnicodeText:FontFamily);
         endlayout;
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   EndGraph;
end;
run;
```

The following step uses the modified template to create Output 22.2.4:

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

Output 22.2.4 PROC REG Fit Plot with the Equation



You can restore the default template by running the following step:

```
proc template;
   delete Stat.Reg.Graphics.Fit / store=sasuser.templat;
run;
```

Cubic Fit Function

The following steps run PROC TRANSREG to find a cubic fit function and display the equation in a plot generated by PROC SGPLOT:

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

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;

data _null_;
   set c end=eof;
   length s $ 200 c $ 1;
   retain s ' ';
```

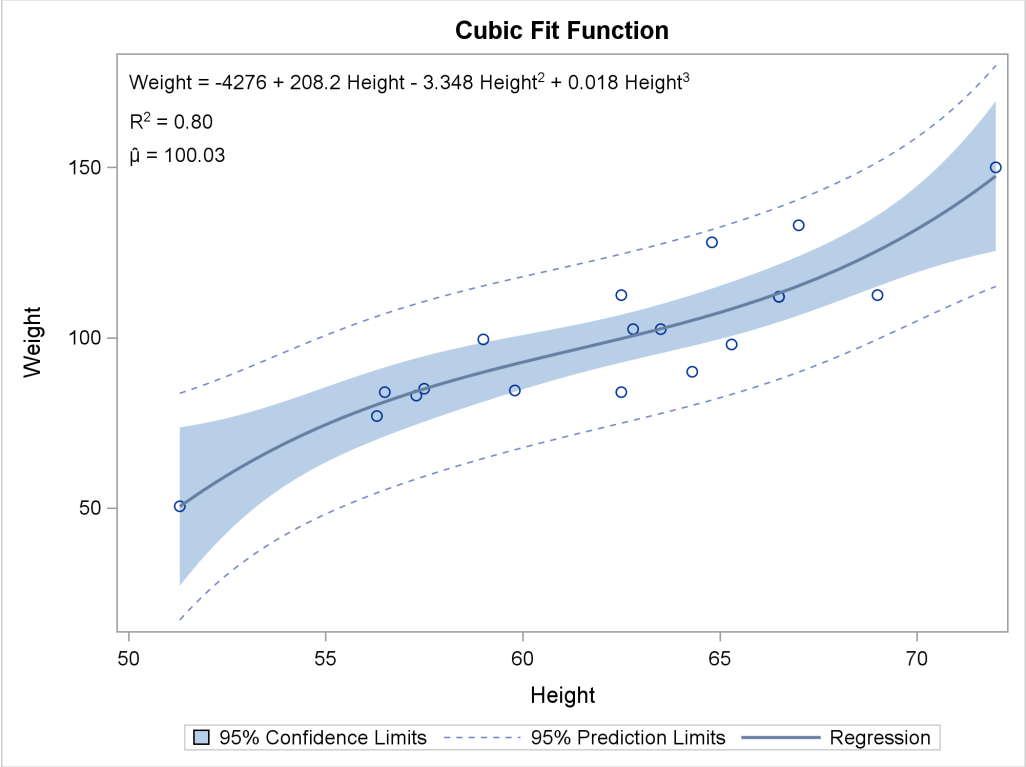
```
   if _n_ = 1 then
      s = scan(dependent, 2, '()') || ' = ' ||    /* dependent =             */
          put(coefficient, best5. -L);            /* intercept               */
   else if abs(coefficient) > 1e-8 then do;       /* skip zero coefficients  */
      s = catx(' ', s,                            /* string so far           */
               ifc(coefficient < 0, '-', '+'),    /* subtract (-) or add (+) */
               put(abs(coefficient), best5. -L ), /* abs(coefficient)        */
               scan(variable, 2, '._'));          /* variable name           */
      c = scan(variable, 2, '_');                 /* grab power              */
      if c ne '1' then                            /* skip power for linear   */
         s = cats(s, "(*ESC*){sup '", c, "'}");   /* add superscript         */
   end;
   if eof then call symputx('formula', trim(s), 'G');
run;

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

These steps create Output 22.2.5.

The PROC TRANSREG MODEL statement fits a model with an untransformed dependent variable and a cubic polynomial function of the independent variable. By default, PSPLINE fits a cubic polynomial spline with no knots, which is simply a cubic polynomial. The fit statistics and parameter estimates are output to data sets, and their values are stored in macro variables. There are three independent variables plus the intercept. Variable names and exponents are extracted from the TRANSREG parameter names of Pspline.Height_1, Pspline.Height_2, and Pspline.Height_3 by using the SCAN function. Exponents are added by using the specifications `"(*ESC*){sup '2'}"` and `"(*ESC*){sup '3'}"`, which are explained in more detail with the INSET statement.

PROC SGPLOT with an INSET statement makes the plot. Each separate string is displayed in a separate line. The first string is the formula, which is generated in the second DATA step. The next string is the R square: it consists of an 'R', an escaped superscript 2, and the value of R square (which is stored in a macro variable). The string for the mean consists of two Unicode specifications, one for the Greek letter μ, and one to put a hat over it. These special character specifications appear in quotes and are escaped with `(*ESC*)` so that they are processed as special characters rather than as literal text. Typically, you must escape special characters in quotes, and not escape them when they are not in quotes. For more information about Unicode characters, see section "Simple Linear Regression" on page 748. For a list of a few of the more commonly used Unicode characters, see the section "Unicode and Special Characters" on page 756.

Output 22.2.5 Cubic Fit Function with the Equation



Unicode and Special Characters

The following steps illustrate Unicode specifications for a number of commonly used characters and create Output 22.2.6 and Output 22.2.7, which are charts of Unicode characters:

```
%let l = halign=left;
proc template;
   define statgraph class;
      begingraph / designheight=550px designwidth=520px;
         layout overlay / xaxisopts=(display=none) yaxisopts=(display=none);
            layout gridded / columns=3 autoalign=(topleft);
               entry &l textattrs=(weight=bold) 'Description';
               entry &l textattrs=(weight=bold) 'Displayed';
               entry &l textattrs=(weight=bold) "Unicode";
               entry &l 'R Square';
               entry &l 'R' {sup '2'};
               entry &l "'R' {sup '2'}";
               entry &l 'y hat sub i';
               entry &l 'y' {unicode hat}{sub 'i'};
               entry &l "'y' {unicode hat}{sub 'i'}";
               entry &l 'less than or equal       ';
               entry &l 'a ' {unicode '2264'x} ' b';
               entry &l "'a ' {unicode '2264'x} ' b'";
               entry &l 'greater than or equal    ';
               entry &l 'b ' {unicode '2265'x} ' a';
               entry &l "'b ' {unicode '2265'x} ' a'";
```

```
entry &l 'infinity';
entry &l {unicode '221e'x};
entry &l "{unicode '221e'x}";
entry &l 'almost equal';
entry &l 'a ' {unicode '2248'x} ' b';
entry &l "'a ' {unicode '2248'x} ' b'";
entry &l 'combining tilde';
entry &l 'El nin' {unicode tilde} 'o';
entry &l "'El nin' {unicode tilde} 'o'";
entry &l 'grave accent';
entry &l 'cre' {unicode '0300'x} 'me';
entry &l "'cre' {unicode '0300'x} 'me'";
entry &l 'circumflex, acute accent   ';
entry &l 'bru' {unicode '0302'x} 'le' {unicode '0301'x} 'e';
entry &l "'bru' {unicode '0302'x} 'le' {unicode '0301'x} 'e'";
entry &l 'alpha';
entry &l {unicode alpha} '   ' {unicode alpha_u};
entry &l "{unicode alpha} '   ' {unicode alpha_u}";
entry &l 'beta';
entry &l {unicode beta} '   ' {unicode beta_u};
entry &l "{unicode beta} '   ' {unicode beta_u}";
entry &l 'gamma';
entry &l {unicode gamma} '   ' {unicode gamma_u};
entry &l "{unicode gamma} '   ' {unicode gamma_u}";
entry &l 'delta';
entry &l {unicode delta} '   ' {unicode delta_u};
entry &l "{unicode delta} '   ' {unicode delta_u}";
entry &l 'epsilon';
entry &l {unicode epsilon} '   ' {unicode epsilon_u};
entry &l "{unicode epsilon} '   ' {unicode epsilon_u}";
entry &l 'zeta';
entry &l {unicode zeta} '   ' {unicode zeta_u};
entry &l "{unicode zeta} '   ' {unicode zeta_u}";
entry &l 'eta';
entry &l {unicode eta} '   ' {unicode eta_u};
entry &l "{unicode eta} '   ' {unicode eta_u}";
entry &l 'theta';
entry &l {unicode theta} '   ' {unicode theta_u};
entry &l "{unicode theta} '   ' {unicode theta_u}";
entry &l 'iota';
entry &l {unicode iota} '   ' {unicode iota_u};
entry &l "{unicode iota} '   ' {unicode iota_u}";
entry &l 'kappa';
entry &l {unicode kappa} '   ' {unicode kappa_u};
entry &l "{unicode kappa} '   ' {unicode kappa_u}";
entry &l 'lambda';
entry &l {unicode lambda} '   ' {unicode lambda_u};
entry &l "{unicode lambda} '   ' {unicode lambda_u}";
entry &l 'mu';
entry &l {unicode mu} '   ' {unicode mu_u};
entry &l "{unicode mu} '   ' {unicode mu_u}";
entry &l 'nu';
entry &l {unicode nu} '   ' {unicode nu_u};
entry &l "{unicode nu} '   ' {unicode nu_u}";
```

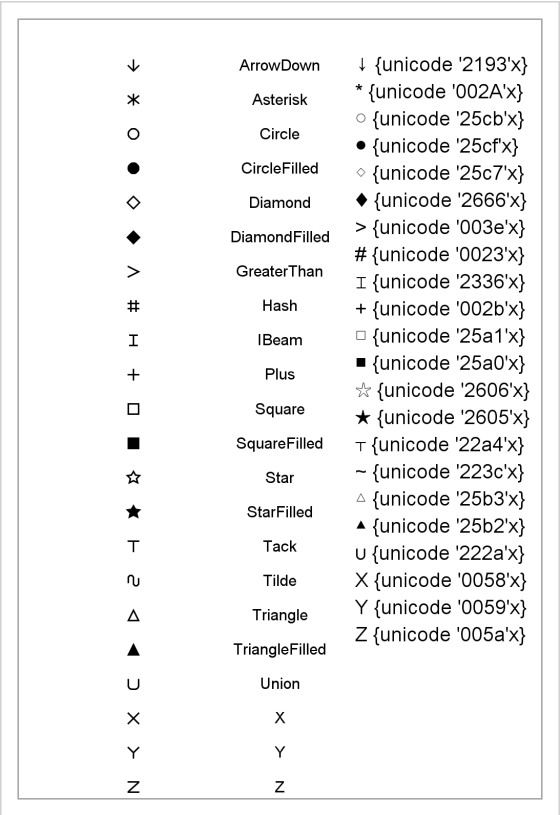
```
               entry &l 'xi';
               entry &l {unicode xi} '   ' {unicode xi_u};
               entry &l "{unicode xi} '   ' {unicode xi_u}";
               entry &l 'omicron';
               entry &l {unicode omicron} '   ' {unicode omicron_u};
               entry &l "{unicode omicron} '   ' {unicode omicron_u}";
               entry &l 'pi';
               entry &l {unicode pi} '   ' {unicode pi_u};
               entry &l "{unicode pi} '   ' {unicode pi_u}";
               entry &l 'rho';
               entry &l {unicode rho} '   ' {unicode rho_u};
               entry &l "{unicode rho} '   ' {unicode rho_u}";
               entry &l 'sigma';
               entry &l {unicode sigma} '   ' {unicode sigma_u};
               entry &l "{unicode sigma} '   ' {unicode sigma_u}";
               entry &l 'tau';
               entry &l {unicode tau} '   ' {unicode tau_u};
               entry &l "{unicode tau} '   ' {unicode tau_u}";
               entry &l 'upsilon';
               entry &l {unicode upsilon} '   ' {unicode upsilon_u};
               entry &l "{unicode upsilon} '   ' {unicode upsilon_u}";
               entry &l 'phi';
               entry &l {unicode phi} '   ' {unicode phi_u};
               entry &l "{unicode phi} '   ' {unicode phi_u}";
               entry &l 'chi';
               entry &l {unicode chi} '   ' {unicode chi_u};
               entry &l "{unicode chi} '   ' {unicode chi_u}";
               entry &l 'psi';
               entry &l {unicode psi} '   ' {unicode psi_u};
               entry &l "{unicode psi} '   ' {unicode psi_u}";
               entry &l 'omega';
               entry &l {unicode omega} '   ' {unicode omega_u};
               entry &l "{unicode omega} '   ' {unicode omega_u}";
            endlayout;
            scatterplot y=weight x=height / markerattrs=(size=0);
         endlayout;
      endgraph;
   end;
run;

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

%macro m(u);
   entry halign=left "(*ESC*){unicode &u.x} {unicode &u.x}" /
         textattrs=GraphValueText (family=GraphUnicodeText:FontFamily);
   %mend;

proc template;
   define statgraph markers;
      begingraph / designheight=510px designwidth=350px;
         layout overlay / xaxisopts=(display=none) yaxisopts=(display=none);
            layout gridded / columns=1 autoalign=(topright);
               entry " ";
```

```
               %m('2193')   %m('002A')   %m('25cb')   %m('25cf')
               %m('25c7')   %m('2666')   %m('003e')   %m('0023')
               %m('2336')   %m('002b')   %m('25a1')   %m('25a0')
               %m('2606')   %m('2605')   %m('22a4')   %m('223c')
               %m('25b3')   %m('25b2')   %m('222a')   %m('0058')
               %m('0059')   %m('005a')
            endlayout;
         scatterplot x=x1 y=y / group=m;
         scatterplot x=x2 y=y / markercharacter=m;
         scatterplot x=x3 y=y / markerattrs=(size=0);
         endlayout;
      endgraph;
   end;
run;

%modstyle(name=mark, parent=statistical, markers=
   ArrowDown Asterisk Circle CircleFilled Diamond DiamondFilled GreaterThan
   Hash IBeam Plus Square SquareFilled Star StarFilled Tack Tilde Triangle
   TriangleFilled Union X Y Z, linestyles=1, colors=black)

data x;
   retain x1 1 x2 2 x3 3;
   length m $ 20;
   input m @@;
   y = -_n_;
   datalines;
ArrowDown Asterisk Circle CircleFilled Diamond DiamondFilled GreaterThan
Hash IBeam Plus Square SquareFilled Star StarFilled Tack Tilde Triangle
TriangleFilled Union X Y Z
;

ods listing style=mark;
proc sgrender data=x template=markers;
run;
ods listing;
```

Output 22.2.6 Commonly Used Unicode and Special Characters

Description	Displayed	Unicode
R Square	R^2	'R' {sup '2'}
y hat sub i	$\hat{y}_i$	'y' {unicode hat}{sub 'i'}
less than or equal	a ≤ b	'a ' {unicode '2264'x} ' b'
greater than or equal	b ≥ a	'b ' {unicode '2265'x} ' a'
infinity	∞	{unicode '221e'x}
almost equal	a ≈ b	'a ' {unicode '2248'x} ' b'
combining tilde	El niño	'El nin' {unicode tilde} 'o'
grave accent	crème	'cre' {unicode '0300'x} 'me'
circumflex, acute accent	brûlée	'bru' {unicode '0302'x} 'le' {unicode '0301'x} 'e'
alpha	α A	{unicode alpha} ' ' {unicode alpha_u}
beta	β B	{unicode beta} ' ' {unicode beta_u}
gamma	γ Γ	{unicode gamma} ' ' {unicode gamma_u}
delta	δ Δ	{unicode delta} ' ' {unicode delta_u}
epsilon	ε E	{unicode epsilon} ' ' {unicode epsilon_u}
zeta	ζ Z	{unicode zeta} ' ' {unicode zeta_u}
eta	η H	{unicode eta} ' ' {unicode eta_u}
theta	θ Θ	{unicode theta} ' ' {unicode theta_u}
iota	ι I	{unicode iota} ' ' {unicode iota_u}
kappa	κ K	{unicode kappa} ' ' {unicode kappa_u}
lambda	λ Λ	{unicode lambda} ' ' {unicode lambda_u}
mu	μ M	{unicode mu} ' ' {unicode mu_u}
nu	ν N	{unicode nu} ' ' {unicode nu_u}
xi	ξ Ξ	{unicode xi} ' ' {unicode xi_u}
omicron	ο O	{unicode omicron} ' ' {unicode omicron_u}
pi	π Π	{unicode pi} ' ' {unicode pi_u}
rho	ρ P	{unicode rho} ' ' {unicode rho_u}
sigma	σ Σ	{unicode sigma} ' ' {unicode sigma_u}
tau	τ T	{unicode tau} ' ' {unicode tau_u}
upsilon	υ Y	{unicode upsilon} ' ' {unicode upsilon_u}
phi	φ Φ	{unicode phi} ' ' {unicode phi_u}
chi	χ X	{unicode chi} ' ' {unicode chi_u}
psi	ψ Ψ	{unicode psi} ' ' {unicode psi_u}
omega	ω Ω	{unicode omega} ' ' {unicode omega_u}

Output 22.2.7 Markers, Marker Names, Unicode Characters, Unicode Specifications



The Unicode Consortium `http://unicode.org/` provides a list of character codes at `http://www.unicode.org/charts/charindex.html`.

The following rules apply to Unicode and special character specifications in ODS graphics:

- Each character can be specified by looking up its code and specifying it as a hexadecimal constant. Example: {**unicode '221e'x**}.
- Lower case Greek letters can be specified by using names instead of hexadecimal constants. Example: {**unicode alpha**}.
- Upper case Greek letters can be specified by using names followed by **_u** instead of a hexadecimal constants. Example: {**unicode alpha_u**}.
- Superscript and subscript have special abbreviations. Examples: {**sup 2**} and {**sub 2**}.
- The **sup** and **sub** specifications must not appear escaped and in quotes in the GTL. They must appear outside of quotes.
- Some characters overprint the character that comes before. Example: **'El nin' {tilde} 'o'**, which is equivalent to **'El nin' {unicode '0303'x} 'o'** creates 'El niño'.
- Specifications inside quotes are escaped. Example: **"(*ESC*){unicode beta}"**.
- Specifications outside quotes are not escaped. Example: {**unicode beta**}.

Example 22.3: Customizing Panels

This example illustrates how to modify the regression fit diagnostics panel shown in Figure 21.1 in Chapter 21, "Statistical Graphics Using ODS," so that it displays a subset of the component plots. The original panel consists of eight plots and a summary statistics box. The ODS trace output from PROC REG shown previously shows that the template for the diagnostics panel is **Stat.REG.Graphics.DiagnosticsPanel**. The following statements display the template:

```
proc template;
   source Stat.REG.Graphics.DiagnosticsPanel;
run;
```

An abridged version of the results is shown next:

```
define statgraph Stat.Reg.Graphics.DiagnosticsPanel;
   notes "Diagnostics Panel";
   dynamic . . .;
   BeginGraph / designheight=defaultDesignWidth;
      entrytitle halign=left textattrs=GRAPHVALUETEXT _MODELLABEL halign=
         center textattrs=GRAPHTITLETEXT "Fit Diagnostics" " for " _DEPNAME;
      layout lattice / columns=3 rowgutter=10 columngutter=10
                       shrinkfonts=true rows=3;
         layout overlay / xaxisopts=(shortlabel='Predicted');
            . . .
         endlayout;
         layout overlay / xaxisopts=(shortlabel='Predicted');
            . . .
         endlayout;
         layout overlay / xaxisopts=(label='Leverage' offsetmax=0.05)
            yaxisopts=(offsetmin=0.05 offsetmax=0.05);
            . . .
         endlayout;
         layout overlay / yaxisopts=(label="Residual" shortlabel="Resid")
            xaxisopts=(label="Quantile");
            . . .
         endlayout;
         layout overlayequated / xaxisopts=(shortlabel='Predicted')
            yaxisopts=(label=_DEPLABEL shortlabel="Observed") equatetype=square;
            . . .
         endlayout;
         layout overlay / xaxisopts=(linearopts=(integer=true) label=
            "Observation" shortlabel="Obs" offsetmax=0.05)
            yaxisopts=(offsetmin=0.05 offsetmax=0.05);
            . . .
         endlayout;
         layout overlay / xaxisopts=(label="Residual") yaxisopts=(label="Percent");
            . . .
         endlayout;
         layout lattice / columns=2 rows=1 rowdatarange=unionall columngutter=0;
            . . .
         endlayout;
         if (_SHOWSTATS =1)
            layout overlay;
```

```
            . . .
            endLayout;
         endif;
         if (_SHOWSTATS = 2)
            layout overlay / yaxisopts=(gridDisplay=auto_off label="Residual");
               . . .
            endlayout;
         endif;
      endlayout;
      . . .
   EndGraph;
end;
```

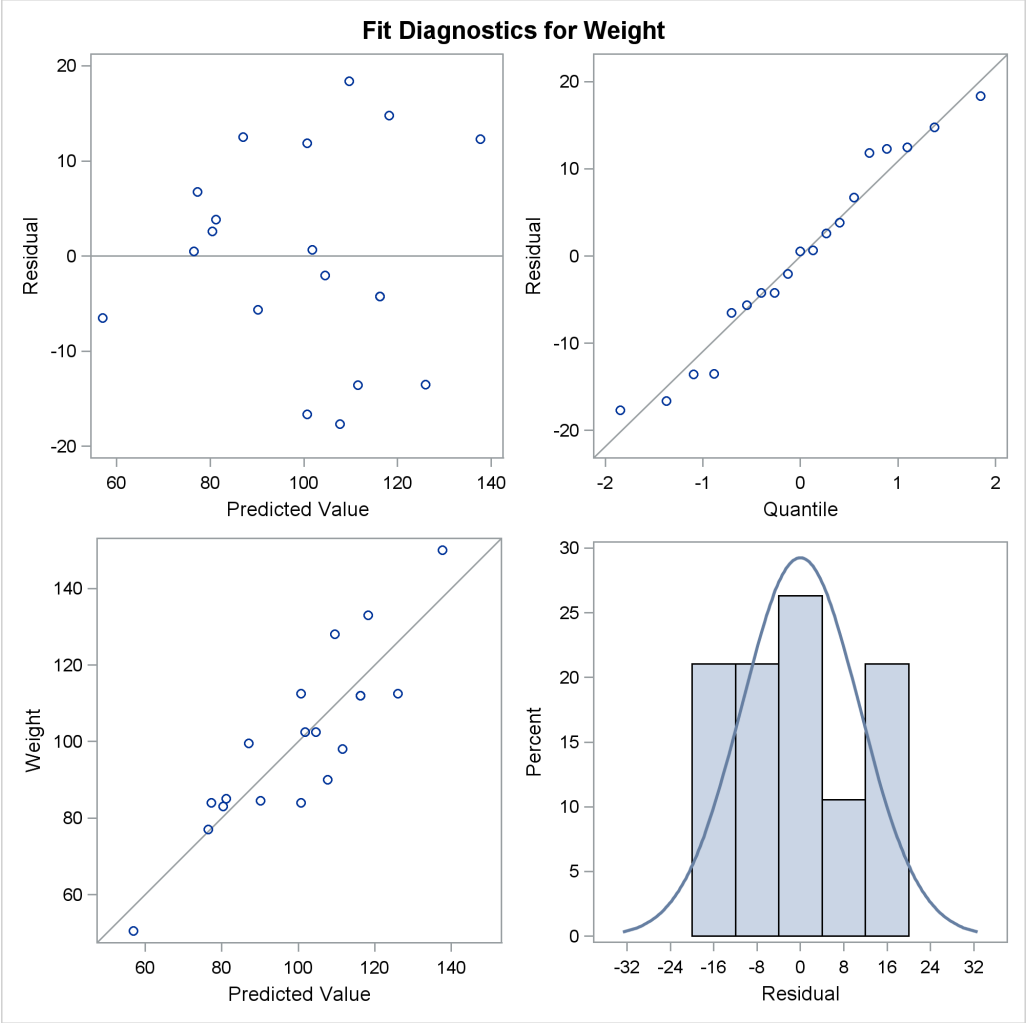
The outermost components of the template are a BEGINGRAPH/ENDGRAPH block with a lattice layout with ROWS=3 and COLUMNS=3 that defines the 3×3 panel of plots. Inside that are nine layouts, one for each cell, the last of which is conditionally defined. The LAYOUT statements define the components of the panel from left to right and top to bottom. You can eliminate some of the panels and produce a 2×2 panel as follows:

```
proc template;
   define statgraph Stat.Reg.Graphics.DiagnosticsPanel;
      notes "Diagnostics Panel";
      dynamic _DEPLABEL _DEPNAME _MODELLABEL _OUTLEVLABEL _TOTFREQ _NPARM _NOBS
         _OUTCOOKSDLABEL _SHOWSTATS _NSTATSCOLS _DATALABEL _SHOWNObs
         _SHOWTOTFREQ _SHOWNParm _SHOWEDF _SHOWMSE _SHOWRSquare _SHOWAdjRSq
         _SHOWSSE _SHOWDepMean _SHOWCV _SHOWAIC _SHOWBIC _SHOWCP _SHOWGMSEP
         _SHOWJP _SHOWPC _SHOWSBC _SHOWSP _EDF _MSE _RSquare _AdjRSq _SSE
         _DepMean _CV _AIC _BIC _CP _GMSEP _JP _PC _SBC _SP _byline_ _bytitle_
         _byfootnote_;
      BeginGraph / designheight=defaultDesignWidth;
         entrytitle halign=left textattrs=GRAPHVALUETEXT _MODELLABEL halign=
            center textattrs=GRAPHTITLETEXT "Fit Diagnostics" " for " _DEPNAME;
         layout lattice / columns=2 rowgutter=10 columngutter=10
            shrinkfonts=true rows=2;
            layout overlay / xaxisopts=(shortlabel='Predicted');
               referenceline y=0;
               scatterplot y=RESIDUAL x=PREDICTEDVALUE / primary=true datalabel=
                  _OUTLEVLABEL rolename=(_tip1=OBSERVATION _id1=ID1 _id2=ID2
                  _id3=ID3 _id4=ID4 _id5=ID5) tip=(y x _tip1 _id1 _id2 _id3 _id4
                  _id5);
            endlayout;
            layout overlay / yaxisopts=(label="Residual" shortlabel="Resid")
               xaxisopts=(label="Quantile");
               lineparm slope=eval (STDDEV(RESIDUAL)) y=eval (MEAN(RESIDUAL))
                  x=0 / clip=false extend=true lineattrs=GRAPHREFERENCE;
               scatterplot y=eval (SORT(DROPMISSING(RESIDUAL))) x=eval(
                  PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
                  N(RESIDUAL)))) / markerattrs=GRAPHDATADEFAULT primary=true
                  rolename=(s=eval (SORT(DROPMISSING(RESIDUAL))) nq=eval (
                  PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/(0.25 +
                  N(RESIDUAL))))) tiplabel=(nq="Quantile" s="Residual")
                  tip=(nq s);
            endlayout;
            layout overlayequated / xaxisopts=(shortlabel='Predicted')
```

```
                yaxisopts=(label=_DEPLABEL shortlabel="Observed")
                equatetype=square;
                lineparm slope=1 x=0 y=0 / clip=true extend=true lineattrs=
                   GRAPHREFERENCE;
                scatterplot y=DEPVAR x=PREDICTEDVALUE / primary=true datalabel=
                   _OUTLEVLABEL rolename=(_tip1=OBSERVATION _id1=ID1 _id2=ID2
                   _id3=ID3 _id4=ID4 _id5=ID5) tip=(y x _tip1 _id1 _id2 _id3 _id4
                   _id5);
             endlayout;
             layout overlay / xaxisopts=(label="Residual") yaxisopts=(label=
                "Percent");
                histogram RESIDUAL / primary=true;
                densityplot RESIDUAL / name="Normal" legendlabel="Normal"
                   lineattrs=GRAPHFIT;
             endlayout;
          endlayout;
          if (_BYTITLE_)
             entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
          else
             if (_BYFOOTNOTE_)
                entryfootnote halign=left _BYLINE_;
             endif;
          endif;
       EndGraph;
    end;
run;

ods graphics on;

proc reg data=sashelp.class;
   model Weight = Height;
run; quit;
```

This template plots the residuals by predicted values, the Q-Q plot, the actual by predicted plot, and the residual histogram. The results are shown in Output 22.3.1.

Output 22.3.1 Diagnostics Panel with Four Plots



This new template is a straightforward modification of the original template. The COLUMNS=2 and ROWS=2 options in the LAYOUT LATTICE statement request a 2×2 lattice. The LAYOUT statement blocks for components 1, 3, 6, 8, and 9 are deleted. **NOTE:** You do not need to understand every aspect of a template to modify it if you can recognize the overall structure and a few key options.

You can restore the original template as follows:

```
proc template;
   delete Stat.REG.Graphics.DiagnosticsPanel / store=sasuser.templat;
run;
```

Example 22.4: Customizing Axes and Reference Lines

This example illustrates several ways that you can change the plot axes in a scatter plot. The example uses PROC CORRESP to perform a correspondence analysis. It is taken from the section "Getting Started: CORRESP Procedure" on page 2130 in Chapter 34, "The CORRESP Procedure." It uses the following data:

```
title "Number of Ph.D.'s Awarded from 1973 to 1978";

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
;
```

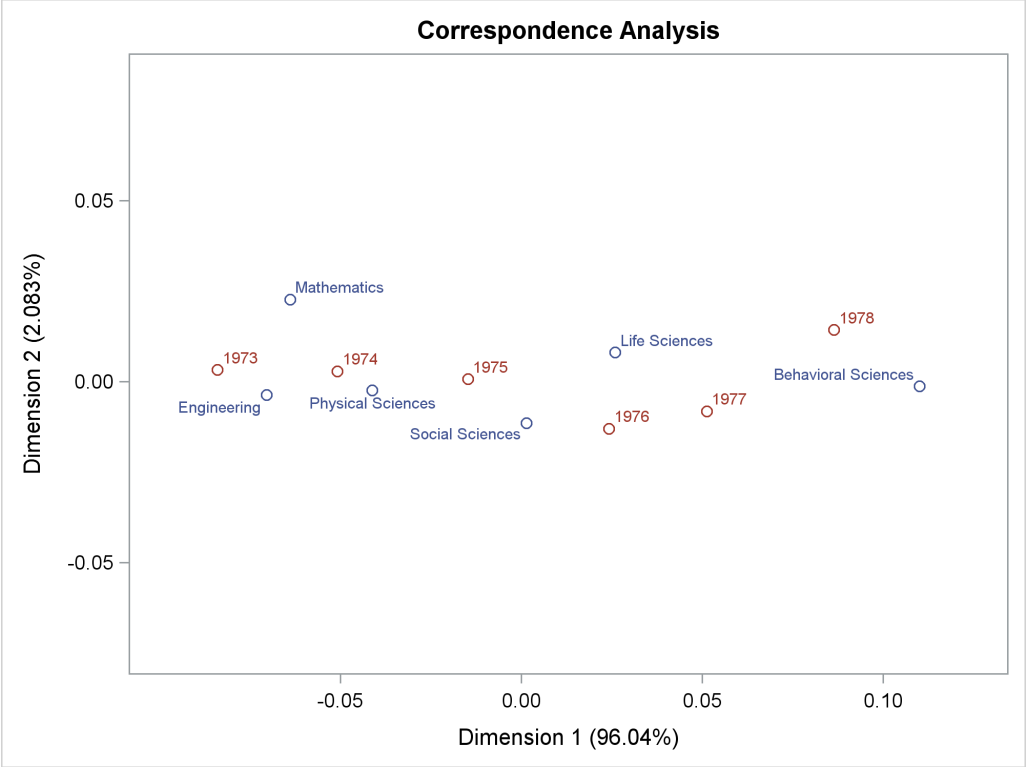
The following steps perform the correspondence analysis and create Output 22.4.1:

```
ods graphics on;
ods trace on;

proc corresp data=PhD short;
   ods select configplot;
   var y1973-y1978;
   id Science;
run;
```

The trace output for this step (not shown) shows that the template for this plot is **Stat.Corresp.Graphics.Configuration**. The following step displays this template:

```
proc template;
   source Stat.Corresp.Graphics.Configuration;
run;
```

Output 22.4.1 Default Scatter Plot



The results are as follows:

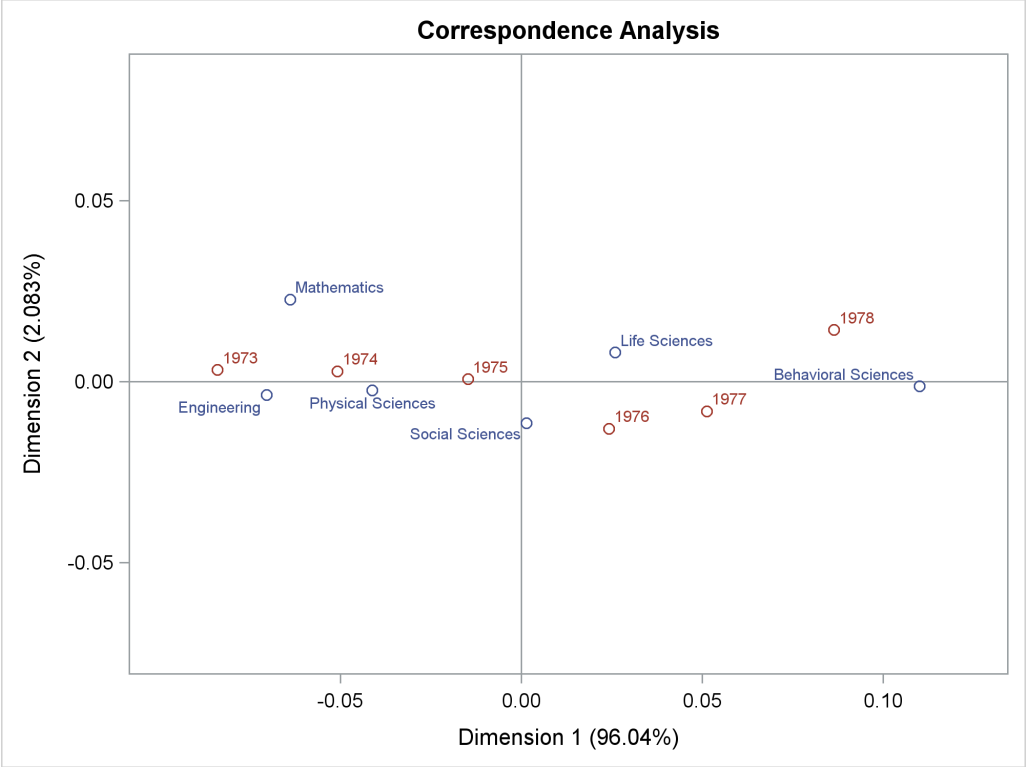
```
define statgraph Stat.Corresp.Graphics.Configuration;
   dynamic xVar yVar head legend _byline_ _bytitle_ _byfootnote_;
   begingraph;
      entrytitle HEAD;
      layout overlayequated / equatetype=fit xaxisopts=(offsetmin=0.1
         offsetmax=0.1) yaxisopts=(offsetmin=0.1 offsetmax=0.1);
         scatterplot y=YVAR x=XVAR / group=GROUP index=INDEX datalabel=LABEL
            name="Type" tip=(y x datalabel group) tiplabel=(group="Point");
         if (LEGEND)
            discretelegend "Type";
         endif;
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   endgraph;
end;
```

You can add reference lines to the scatter plot at specified X and Y values by using the REFERENCELINE statement, as in the following example:

```
proc template;
   define statgraph Stat.Corresp.Graphics.Configuration;
      dynamic xVar yVar head legend _byline_ _bytitle_ _byfootnote_;
      begingraph;
         entrytitle HEAD;
         layout overlayequated / equatetype=fit xaxisopts=(offsetmin=0.1
            offsetmax=0.1) yaxisopts=(offsetmin=0.1 offsetmax=0.1);

            referenceline x=0;
            referenceline y=0;

            scatterplot y=YVAR x=XVAR / group=GROUP index=INDEX datalabel=LABEL
               name="Type" tip=(y x datalabel group) tiplabel=(group="Point");
            if (LEGEND)
               discretelegend "Type";
            endif;
         endlayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      endgraph;
   end;
run;

proc corresp data=PhD short;
   ods select configplot;
   var y1973-y1978;
   id Science;
run;
```

When you modify templates, it is important to note that the order of the statements within the LAYOUT OVERLAYEQUATED block (or more typically, the LAYOUT OVERLAY block) is significant. Here, the reference lines are added before the scatter plot so that the reference lines are drawn before the scatter plot. Consequently, labels and markers that coincide with the reference lines are drawn over the reference lines. The results, with reference lines, are displayed in Output 22.4.2.

Output 22.4.2 Scatter Plot with Reference Lines Added



You can restore the default graph template as follows:

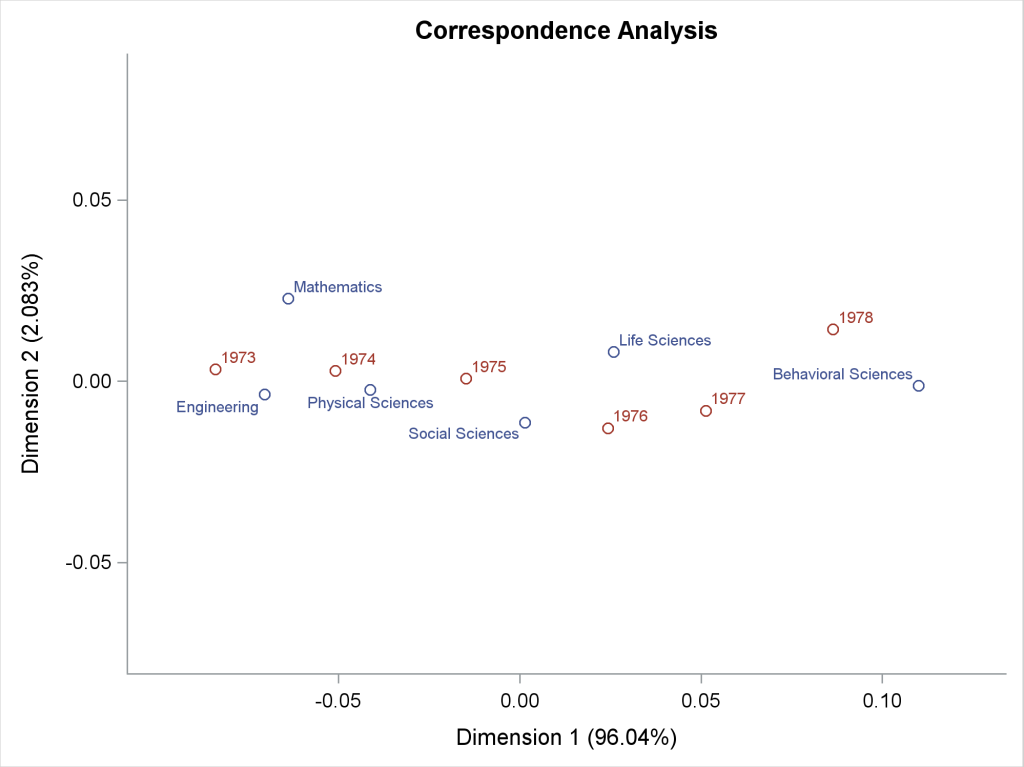
```
proc template;
   delete Stat.Corresp.Graphics.Configuration / store=sasuser.templat;
run;
```

The next steps show how you can change the style so that a frame is not shown:

```
proc template;
   define style noframe;
      parent=styles.htmlblue;
      style graphwalls from graphwalls / frameborder=off;
   end;
run;

ods listing style=noframe;

proc corresp data=PhD short;
   ods select configplot;
   var y1973-y1978;
   id Science;
run;
```

The results, shown in Output 22.4.3, display an X-axis and a Y-axis without a frame. Unlike the previous change, which affects only the ConfigPlot display, this change affects all plots created with the NOFRAME style.

Output 22.4.3 Scatter Plot with No Axis Frame



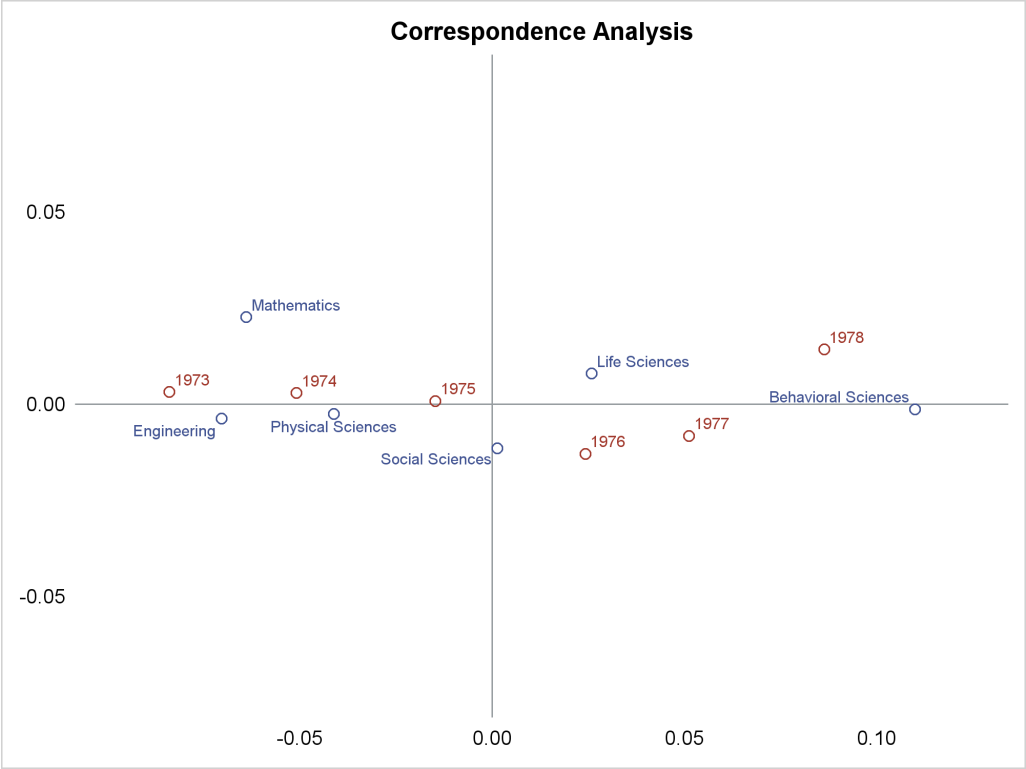
Alternatively, you can also add reference lines and delete the entire axis frame using the WALLDISPLAY=NONE and the DISPLAY= option in the graph template, as in the following example:

```
proc template;
   define statgraph Stat.Corresp.Graphics.Configuration;
      dynamic xVar yVar head legend _byline_ _bytitle_ _byfootnote_;
      begingraph;
         entrytitle HEAD;

         layout overlayequated / equatetype=fit walldisplay=none
            xaxisopts=(display=(tickvalues) offsetmin=0.1 offsetmax=0.1)
            yaxisopts=(display=(tickvalues) offsetmin=0.1 offsetmax=0.1);

            referenceline x=0;
            referenceline y=0;

            scatterplot y=YVAR x=XVAR / group=GROUP index=INDEX datalabel=LABEL
               name="Type" tip=(y x datalabel group) tiplabel=(group="Point");
            if (LEGEND)
               discretelegend "Type";
            endif;
         endlayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
```

```
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   endgraph;
end;
run;

ods listing style=htmlblue;

proc corresp data=PhD short;
   ods select configplot;
   var y1973-y1978;
   id Science;
run;
```

The results are shown in Output 22.4.4.

Output 22.4.4 Scatter Plot with Internal Axes



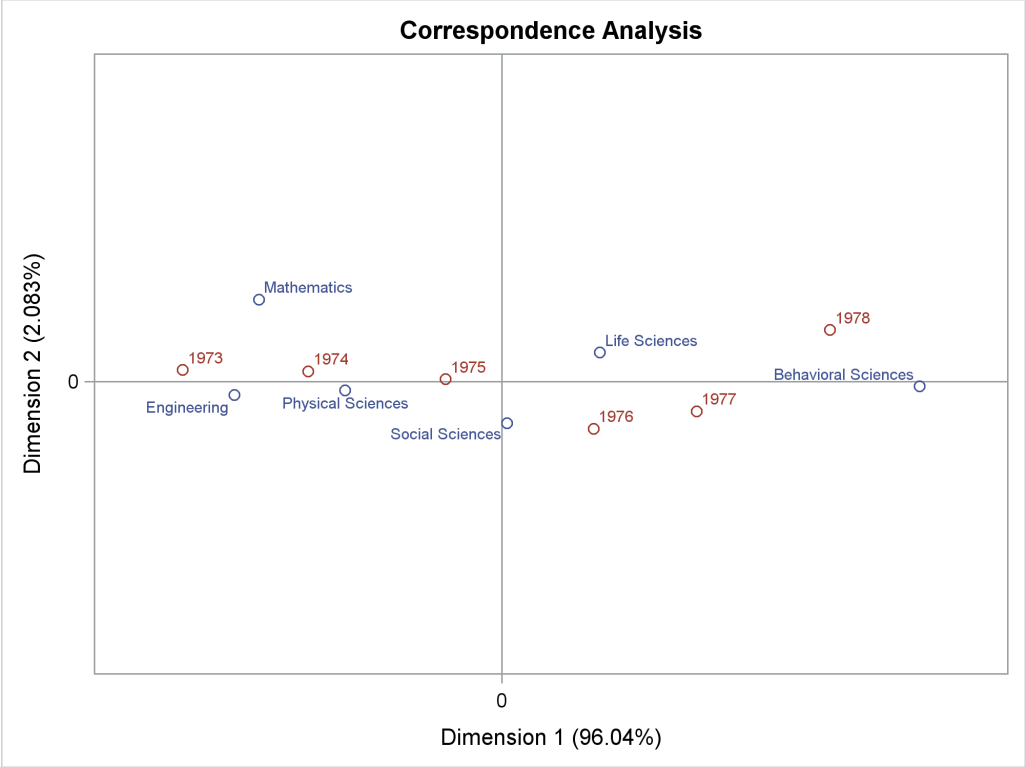
Instead of DISPLAY=(TICKVALUES), you can use DISPLAY=NONE (not shown) to remove the tick values from the display as well. You can change the tick values, as in the following example:

```
proc template;
   define statgraph Stat.Corresp.Graphics.Configuration;
      dynamic xVar yVar head legend _byline_ _bytitle_ _byfootnote_;
      begingraph;
         entrytitle HEAD;

         layout overlayequated / equatetype=fit
```

```
            commonaxisopts=(tickvaluelist=(0))
            xaxisopts=(offsetmin=0.1 offsetmax=0.1)
            yaxisopts=(offsetmin=0.1 offsetmax=0.1);


            referenceline x=0;
            referenceline y=0;

            scatterplot y=YVAR x=XVAR / group=GROUP index=INDEX datalabel=LABEL
               name="Type" tip=(y x datalabel group) tiplabel=(group="Point");
            if (LEGEND)
               discretelegend "Type";
            endif;
         endlayout;
         if (_BYTITLE_)
            entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
         else
            if (_BYFOOTNOTE_)
               entryfootnote halign=left _BYLINE_;
            endif;
         endif;
      endgraph;
   end;
run;

proc corresp data=PhD short;
   ods select configplot;
   var y1973-y1978;
   id Science;
run;
```

Since the axes in this plot are equated, the ticks are specified using the option **commonaxisopts = (tickvaluelist = (tick-value-list))**. This example only shows ticks at zero, but you can specify lists of values instead. The results are shown in Output 22.4.5.

Output 22.4.5 Scatter Plot with Tick Marks Specified



If the axes are not equated, then the tick value list is specified with the LINEAROPTS= option, as in the following statement:

```
layout overlay / xaxisopts=(linearopts=(viewmin=-0.1 viewmax=0.1
                                        tickvaluelist=(-0.1 0 0.1))
                            offsetmin=0.1 offsetmax=0.1)
                 yaxisopts=(linearopts=(viewmin=-0.1 viewmax=0.1
                                        tickvaluelist=(-0.1 0 0.1))
                            offsetmin=0.1 offsetmax=0.1);
```

The preceding statement uses the VIEWMIN= and VIEWMAX= options to specify the beginning and end of the data range that is shown. Specifying a tick value list does not extend or restrict the range of data shown in the plot. When axes share common options, it might be more convenient to use a macro to specify the options. The following two statements are equivalent to the preceding statement:

```
%let opts = linearopts=(viewmin=-0.1 viewmax=0.1
            tickvaluelist=(-0.1 0 0.1)) offsetmin=0.1 offsetmax=0.1;
layout overlay / xaxisopts=(&opts) yaxisopts=(&opts);
```

You can restore the default graph template as follows:

```
proc template;
   delete Stat.Corresp.Graphics.Configuration / store=sasuser.templat;
run;
```

Example 22.5: Adding Text to Every Graph

This example shows how to add text to one or more graphs. For example, you can create a macro variable, with project and date information, as follows:

```
%let date = Project 17.104, &sysdate;
```

In order to add this information to a set of graphs, you need to first know the names of their templates. You can list the names of every graph template for SAS/STAT procedures or for a particular procedure as follows:

```
proc template;
   list stat     / where=(type='Statgraph');
   list stat.reg / where=(type='Statgraph');
run;
```

The results for PROC REG are shown in Output 22.5.1.

Output 22.5.1 PROC REG Templates

Listing of: SASHELP.TMPLSTAT
Path Filter is: Stat.Reg
Sort by: PATH/ASCENDING

Obs	Path	Type
1	Stat.Reg.Graphics.CooksD	Statgraph
2	Stat.Reg.Graphics.CooksDChart	Statgraph
3	Stat.Reg.Graphics.DFBETASPanel	Statgraph
4	Stat.Reg.Graphics.DFBETASPlot	Statgraph
5	Stat.Reg.Graphics.DFFITSPlot	Statgraph
6	Stat.Reg.Graphics.DiagnosticsPanel	Statgraph
7	Stat.Reg.Graphics.Fit	Statgraph
8	Stat.Reg.Graphics.FitHeatMap	Statgraph
9	Stat.Reg.Graphics.ObservedByPredicted	Statgraph
10	Stat.Reg.Graphics.PartialPanel	Statgraph
11	Stat.Reg.Graphics.PartialPlot	Statgraph
12	Stat.Reg.Graphics.PredictionPanel	Statgraph
13	Stat.Reg.Graphics.QQPlot	Statgraph
14	Stat.Reg.Graphics.RFPlot	Statgraph
15	Stat.Reg.Graphics.RStudentByPredicted	Statgraph
16	Stat.Reg.Graphics.ResidualBoxPlot	Statgraph
17	Stat.Reg.Graphics.ResidualByPredicted	Statgraph
18	Stat.Reg.Graphics.ResidualHeatMap	Statgraph
19	Stat.Reg.Graphics.ResidualHeatPanel	Statgraph
20	Stat.Reg.Graphics.ResidualHistogram	Statgraph
21	Stat.Reg.Graphics.ResidualPanel	Statgraph
22	Stat.Reg.Graphics.ResidualPlot	Statgraph
23	Stat.Reg.Graphics.RidgePanel	Statgraph
24	Stat.Reg.Graphics.RidgePlot	Statgraph
25	Stat.Reg.Graphics.SelectionCriterionPanel	Statgraph
26	Stat.Reg.Graphics.SelectionCriterionPlot	Statgraph
27	Stat.Reg.Graphics.StepSelectionCriterionPanel	Statgraph
28	Stat.Reg.Graphics.StepSelectionCriterionPlot	Statgraph
29	Stat.Reg.Graphics.StudResCooksDChart	Statgraph
30	Stat.Reg.Graphics.StudentResChart	Statgraph
31	Stat.Reg.Graphics.VIFPlot	Statgraph
32	Stat.Reg.Graphics.rstudentByLeverage	Statgraph

You can show the source for the graph templates for SAS/STAT procedures or for a particular procedure as follows:

```
options ls=96;
proc template;
   source stat     / where=(type='Statgraph');
   source stat.reg / where=(type='Statgraph');
options ls=80;
```

The results of this step are not shown. However, Example 22.3 shows a portion of the template for the PROC REG diagnostics panel. Here, the OPTIONS statement is used to set a line size of 96, which sometimes works better than the smaller default line size when showing the source for large and complicated templates.

An abridged version of the first few lines of the diagnostics panel template is displayed next:

```
define statgraph Stat.Reg.Graphics.DiagnosticsPanel;
   notes "Diagnostics Panel";
   dynamic . . .;
   BeginGraph / designheight=defaultDesignWidth;
      entrytitle halign=left textattrs=GRAPHVALUETEXT _MODELLABEL
         halign=center textattrs=GRAPHTITLETEXT "Fit Diagnostics"
         " for " _DEPNAME;
      . . .
```

Adding a Date and Project Stamp to a Few Graphs

You can add the project and date to the bottom of all graphs produced with PROC REG by putting a PROC TEMPLATE statement in front of the template source code, and adding an MVAR and ENTRYFOOTNOTE statement after every BEGINGRAPH statement, as in the following example:

```
proc template;
   define statgraph Stat.Reg.Graphics.DiagnosticsPanel;
      notes "Diagnostics Panel";
      dynamic . . .;
      BeginGraph / designheight=defaultDesignWidth;

         mvar date;
         entryfootnote halign=left textattrs=GraphValueText date;

         entrytitle halign=left textattrs=GRAPHVALUETEXT _MODELLABEL
            halign=center textattrs=GRAPHTITLETEXT "Fit Diagnostics"
            " for " _DEPNAME;
         . . .
```

The MVAR statement enables you to dynamically customize the template and graph at procedure run time, just as the DYNAMIC statement enables the procedure to dynamically customize the template and graph. With the MVAR statement, you can modify the template once and reuse that modification as the macro changes over time: Alternatively, you can modify the templates as follows:

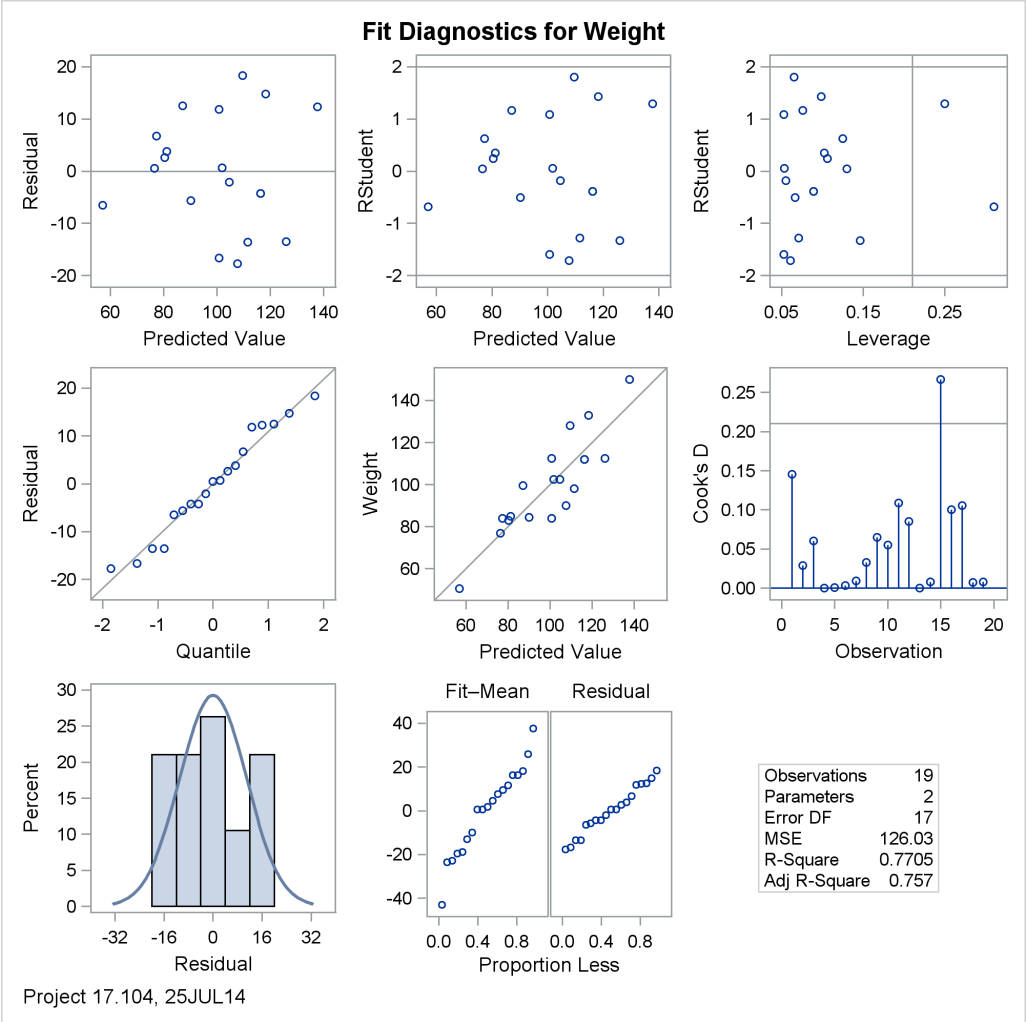
```
entryfootnote halign=left textattrs=GRAPHVALUETEXT "&date";
```

However, you would then have to resubmit your templates every time the macro variable changed. The substitution for the macro variable date occurs at different times in the two preceding cases. In the former case, ODS looks for the value of the macro variable date at the time the template is used, and then the current date variable is used to set the text in the ENTRYFOOTNOTE statement, every time the template is used. In the latter case, SAS substitutes the value of the macro variable once, at the time that the PROC TEMPLATE step is executed.

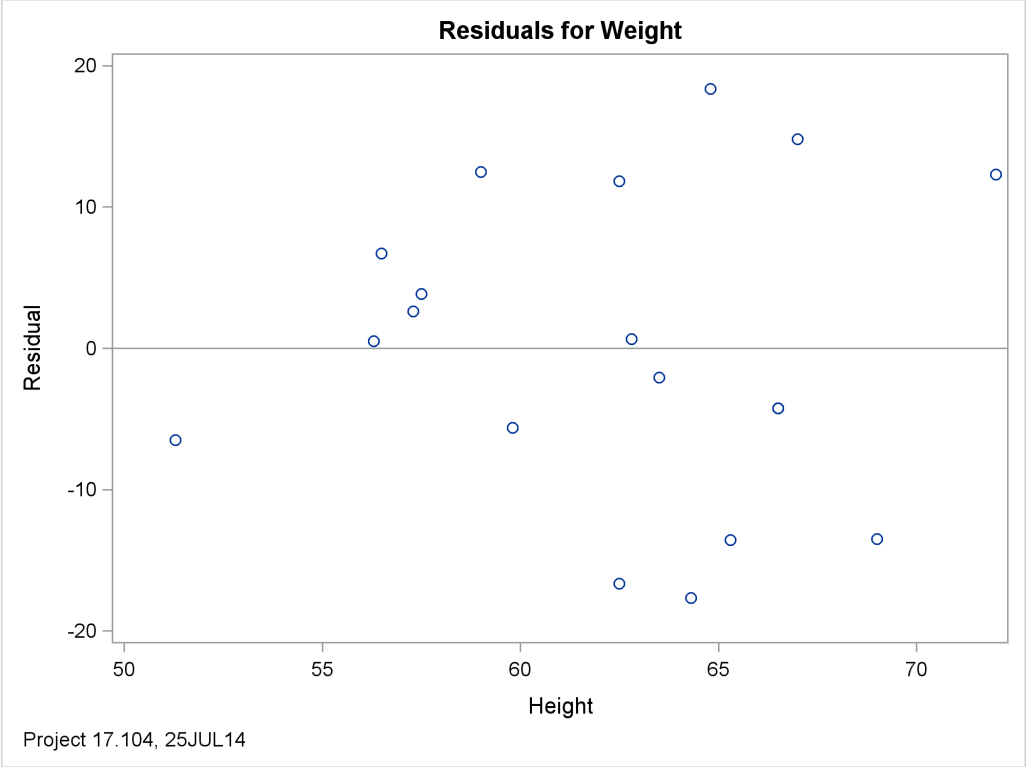
The following steps use the Class data set and produce Output 22.5.2:

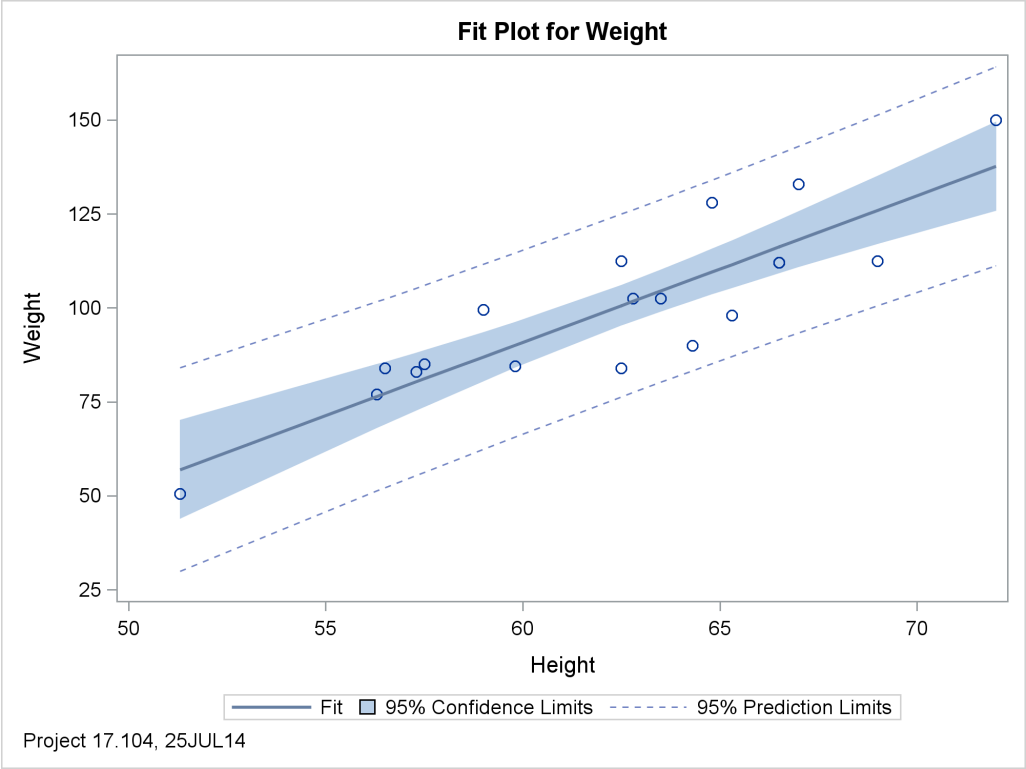
```
ods graphics on;

proc reg data=sashelp.class plots=fit(stats=none);
   model weight = height;
run; quit;
```

Output 22.5.2 PROC REG Plots with Project and Date Stamp



Output 22.5.2 *continued*





You can restore all of the default templates for PROC REG by running the following step:

```
proc template;
   delete stat.reg / store=sasuser.templat;
run;
```

Alternatively, you can specify **delete stat** to restore all SAS/STAT templates to their default definitions.

You can add text to the top or the bottom of a graph by using the ENTRYTITLE or the ENTRYFOOTNOTE statement, respectively. With both statements, you can put the text in the HALIGN=RIGHT, HALIGN=LEFT, or HALIGN=CENTER positions. You can add text to titles even if they already have a centered title. For example, the ENTRYTITLE statement in the diagnostic panel has text on the left (which is conditionally displayed) and a centered title:

```
entrytitle halign=left textattrs=GraphValueText _MODELLABEL
   halign=center textattrs=GraphTitleText "Fit Diagnostics"
   " for " _DEPNAME;
```

The current title can be followed by HALIGN=RIGHT and more text.

Adding Data Set Information to a Graph

You might, for example, want to add text to a set of graphs that indicates the most recently created data set. The following example shows you how you can do this with the syslast macro variable:

```
%let data = &syslast;

. . .

mvar data;
entrytitle halign=left textattrs=GraphValueText "Data: " data
   halign=center textattrs=GraphTitleText "Fit Diagnostics"
   " for " _DEPNAME;

. . .
```

Of course, this only makes sense when you are analyzing the last data set created. Alternatively, you can incorporate the name of the data set in the title, as in the following example:

```
%let data = &syslast;

. . .

mvar data;
entrytitle halign=center textattrs=GraphTitleText
           "Fit Diagnostics for Data Set " data;

. . .
```

Adding a Date and Project Stamp to All Graphs

Sometimes, you can automate the process of template modification. For example, you can automatically add an MVAR and ENTRYFOOTNOTE statement to every graph template, as in the following example:

```
ods path sashelp.tmplmst(read);
proc datasets library=sasuser nolist;
   delete templat(memtype=itemstor);
run;
ods path sasuser.templat(update) sashelp.tmplmst(read);

options ls=256;

proc template;
   source / where=(type='Statgraph') file="tpls.sas";
run;

options ls=80;

data _null_;
   infile 'tpls.sas' lrecl=256 pad;
   input line $ 1-256;
   file 'newtpls.sas';
   put line;
   line = left(lowcase(line));
   if line =: 'begingraph' then
      put 'mvar __date;' /
          'entryfootnote halign=left textattrs=GraphValueText __date;';

   file log;
   if index(line, '__date') then
      put 'ERROR: Name __date already used.' / line;
   if index(line, 'entryfootnote') then put line;
run;

proc template;
   %include 'newtpls.sas' / nosource;
run;
```

These statements write all ODS graph templates to a file, read that file, and write out a new file with an MVAR and ENTRYFOOTNOTE statement added after every BEGINGRAPH statement. Then these new templates are compiled with PROC TEMPLATE. These steps assume that no BEGINGRAPH statement is longer than 256 characters. Most graphs do not have footnotes. Those that do will now have multiple footnotes. You might want to manually combine them or write a more complicated program to handle them. These steps also assume that the name __date is not used anywhere. However, the program does check this and also lists all ENTRYFOOTNOTE statements. Be careful to check the SAS log to ensure that all templates compile without error. Also, before using templates that are automatically modified, make sure your modifications are reasonable.

You can delete Sasuser.Templat and hence all modified templates (assuming the default template search path) as follows:

```
ods path sashelp.tmplmst(read);
proc datasets library=sasuser nolist;
   delete templat(memtype=itemstor);
run;
ods path sasuser.templat(update) sashelp.tmplmst(read);
```

Example 22.6: PROC TEMPLATE Statement Order and Primary Plots

This example uses artificial data to illustrate two basic principles of template writing: that statement order matters and that one of the plotting statements is the primary statement. The data are a sample from a bivariate normal distribution. A custom graph template and PROC SGRENDER are used to plot the data along with vectors and ellipses. The plot consists of four components: a scatterplot of the data; vectors whose end points come from other variables in the data set; ellipses whose parameters are specified in the template; and reference lines whose locations are specified in the template. Initially, thick lines are used to show what happens at the places where the lines and points intersect.

The following steps create the input SAS data set:

```
data x;
   input x y;
   label x = 'Normal(0, 4)' y = 'Normal(0, 1)';
   datalines;
-4  0
 4  0
 0 -2
 0  2
;

data y(drop=i);
   do i = 1 to 2500;
      r1 = normal( 104 );
      r2 = normal( 104 ) * 2;
      output;
   end;
run;

data all;
   merge x y;
run;
```

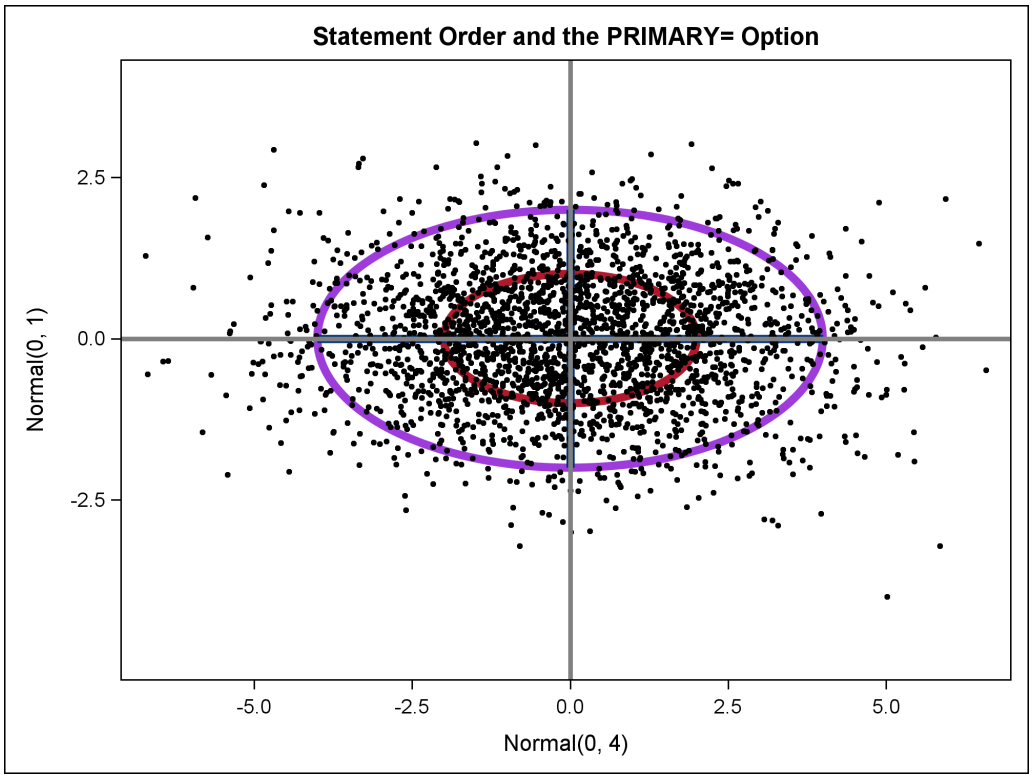
The data set All contains four variables. The variables r1 and r2 contain the random data. These variables contain 2500 nonmissing observations. The data set also contains the variables x and y, which contain the end points for the vectors. These variables contain four nonmissing observations and 2496 observations that are all missing. A data set like this is not unusual when creating overlaid plots. Different overlays often require input data with very different sizes. First, the data are plotted by using a template that is deliberately constructed to demonstrate a number of problems that can occur with statement order.

The following steps create Output 22.6.1:

```
proc template;
   define statgraph Plot;
      begingraph;
         entrytitle 'Statement Order and the PRIMARY= Option';
         layout overlayequated / equatetype=fit;
            ellipseparm semimajor=eval(sqrt(4)) semiminor=1
                        slope=0 xorigin=0 yorigin=0 /
                        outlineattrs=GraphData2(pattern=solid thickness=5);
            ellipseparm semimajor=eval(2 * sqrt(4)) semiminor=2
```

```
                        slope=0 xorigin=0 yorigin=0 /
                        outlineattrs=GraphData5(pattern=solid thickness=5);
            vectorplot  y=y x=x xorigin=0 yorigin=0 /
                        arrowheads=false lineattrs=GraphFit(thickness=5);
            scatterplot y=r1 x=r2 /
                        markerattrs=(symbol=circlefilled size=3);
            referenceline x=0 / lineattrs=(thickness=3);
            referenceline y=0 / lineattrs=(thickness=3);
         endlayout;
      endgraph;
   end;
run;

ods listing style=listing;

proc sgrender data=all template=plot;
run;
```

Output 22.6.1 Statements Specified in a Nonoptimal Order

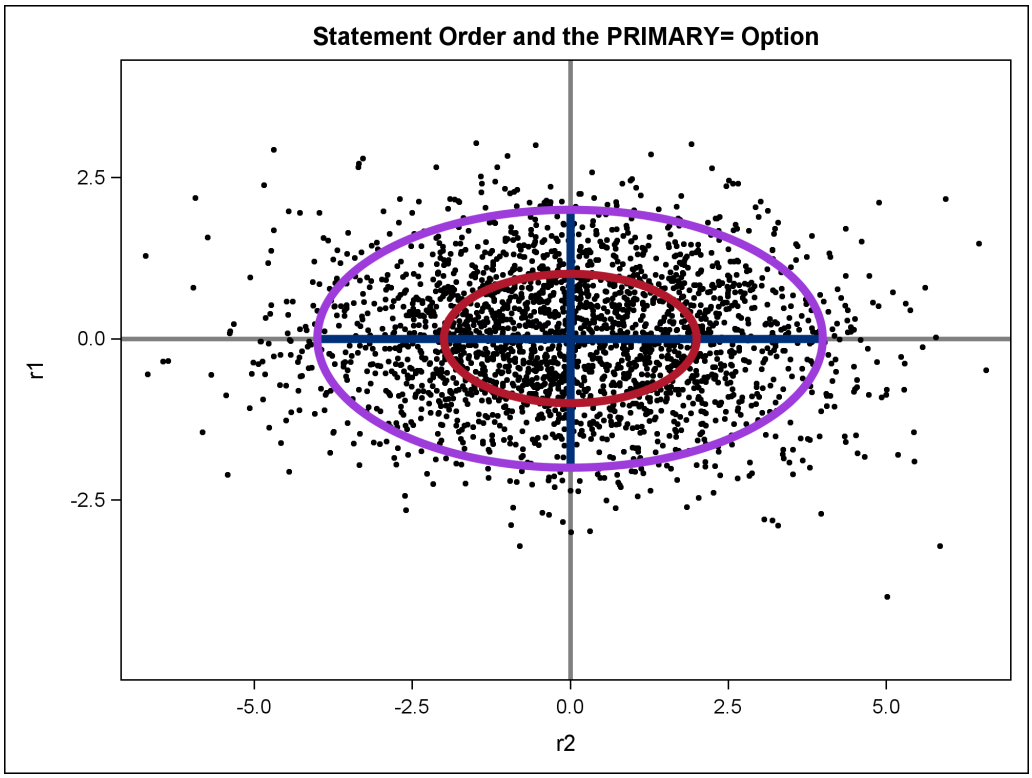


There are a number of problems with the plot in Output 22.6.1. The reference lines obliterate the vectors, and the data are on top of everything but the reference lines. It might be more reasonable to plot the reference lines first, the data next, the vectors next, and the ellipses last. The following steps do this and produce Output 22.6.2:

```
proc template;
   define statgraph Plot;
      begingraph;
         entrytitle 'Statement Order and the PRIMARY= Option';
```

```
         layout overlayequated / equatetype=fit;
            referenceline x=0 / lineattrs=(thickness=3);
            referenceline y=0 / lineattrs=(thickness=3);
            scatterplot y=r1 x=r2 /
                        markerattrs=(symbol=circlefilled size=3);
            vectorplot  y=y x=x xorigin=0 yorigin=0 /
                        arrowheads=false lineattrs=GraphFit(thickness=5);
            ellipseparm semimajor=eval(sqrt(4)) semiminor=1
                        slope=0 xorigin=0 yorigin=0 /
                        outlineattrs=GraphData2(pattern=solid thickness=5);
            ellipseparm semimajor=eval(2 * sqrt(4)) semiminor=2
                        slope=0 xorigin=0 yorigin=0 /
                        outlineattrs=GraphData5(pattern=solid thickness=5);
         endlayout;
      endgraph;
   end;
run;

ods listing style=listing;

proc sgrender data=all template=plot;
run;
```

Output 22.6.2 Statement Order Fixed



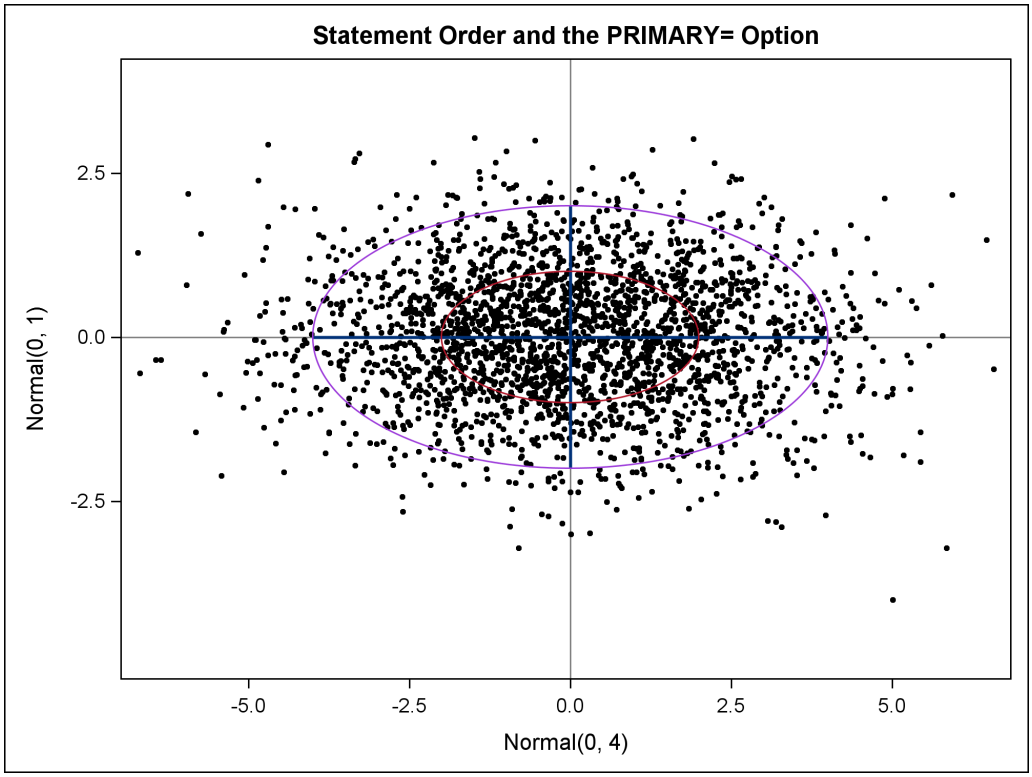
Output 22.6.2 looks better than Output 22.6.1, but the labels for the axes have changed. Output 22.6.1 has the labels of the variables x and y as axis labels, whereas Output 22.6.2 uses the names of the variables r1 and r2. This is because in the Output 22.6.1, the first plot is the vector plot of x and y (which have labels), and in Output 22.6.2, the first plot is the scatter plot of r1 and r2 (which do not have labels). By default, the first plot

is the *primary plot*, and the primary plot is used to determine the axis type and labels. You can designate the vector plot as the primary plot with the PRIMARY=TRUE option.

The following statements make the final plot, this time with default line thicknesses, and produce Output 22.6.3:

```
proc template;
   define statgraph Plot;
      begingraph;
         entrytitle 'Statement Order and the PRIMARY= Option';
         layout overlayequated / equatetype=fit;
            referenceline x=0;
            referenceline y=0;
            scatterplot y=r1 x=r2 / markerattrs=(symbol=circlefilled size=3);

            vectorplot  y=y x=x xorigin=0 yorigin=0 / primary=true
                        arrowheads=false lineattrs=GraphFit;

            ellipseparm semimajor=eval(sqrt(4)) semiminor=1
                        slope=0 xorigin=0 yorigin=0 /
                        outlineattrs=GraphData2(pattern=solid);
            ellipseparm semimajor=eval(2 * sqrt(4)) semiminor=2
                        slope=0 xorigin=0 yorigin=0 /
                        outlineattrs=GraphData5(pattern=solid);
         endlayout;
      endgraph;
   end;
run;

ods listing style=listing;

proc sgrender data=all template=plot;
run;
```

Output 22.6.3 Statement Order Fixed and Primary Plot Specified



The axis labels in Output 22.6.3 and the overprinting of plot elements look better than in the previous plots. You can further adjust the line thicknesses if you want to emphasize or deemphasize components of this plot. The following list discusses the syntax of the GTL statements used in this example.

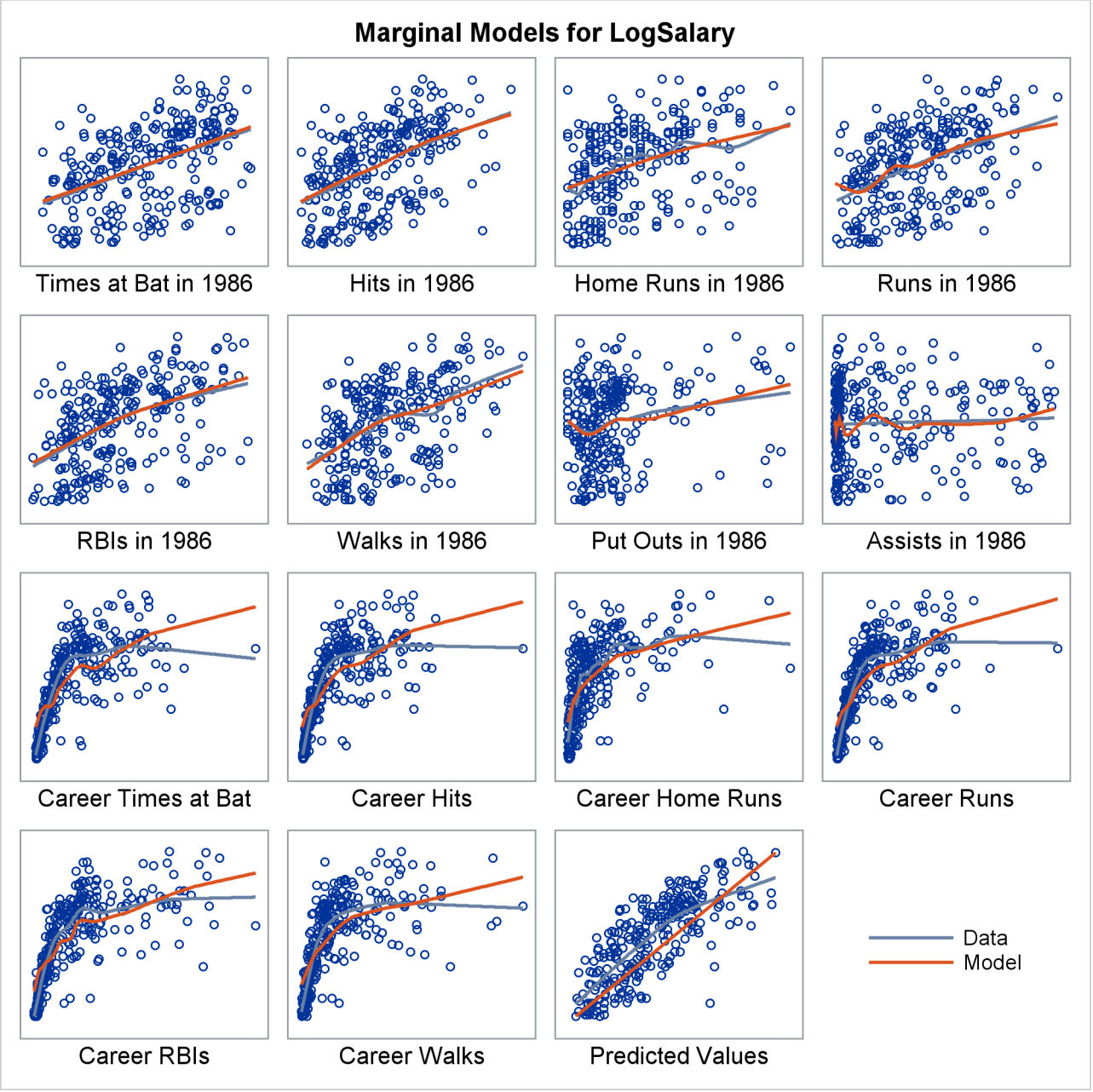
- The template has an ENTRYTITLE statement that specifies the title.
- The template has an equated overlay. This means that a centimeter on one axis represents the same data range as a centimeter on the other axis. This is done instead of the more common LAYOUT OVERLAY statement since with these data, the shape and geometry of the data have meaning even though the ranges of the two axis variables are different. The option EQUATETYPE=SQUARE is used to make a square plot, but since the X-axis variable has a larger range than the Y-axis variable, and since the default plot size is wider than high, EQUATETYPE=FIT is specified. The axes are equated but use the available space.
- A vertical reference line is drawn at X=0, and a horizontal reference line is drawn at Y=0.
- The scatter plot is based on the Y-axis variable r2 and the X-axis variable r1. The markers are filled circles with a size of three pixels. This is smaller than the default size and works well with a plot that displays many points.

- The vector plot is based on the Y-axis variable `y` and the X-axis variable `x`. The vectors are solid lines with no heads emanating from the origin (X=0 and Y=0). The color and other line attributes such as thickness come from the attributes of the **`GraphFit`** style element. This is the primary plot, so the default axis labels are the variable labels for the X= and Y= variables if they exist or the variable names if the variables do not have labels.
- The plot also displays two ellipses with X=0 and Y=0 at their center. Their widths are expressions, and their heights are constant. The expressions are not needed in this example; they are used to illustrate the syntax. The SEMIMAJOR= option specifies half the length of the major axis for the ellipse, and the SEMIMINOR= option specifies half the length of the minor axis for the ellipse. The SLOPE= option specifies the slope of the major axis for the ellipse. The colors of the ellipses and other line properties are based on the **`GraphData2`** and **`GraphData5`** style elements, but the line pattern attribute from the style is overridden.

Example 22.7: Marginal Model Plots

Marginal model plots (proposed by Cook and Weisberg 1997 and discussed by Fox and Weisberg 2011) display the marginal relationship between the response and each predictor. Marginal model plots display the dependent variable on each vertical axis and each independent variable on a horizontal axis. There is one marginal model plot for each independent variable and one additional plot that displays the predicted values on the horizontal axis. Each plot contains a scatter plot of the two variables, a smooth fit function for the variables in the plot (labeled "Data"), and a function that displays the predicted values as a function of the horizontal axis variable (labeled "Model"). (See Figure 22.7.1.) When the two functions are similar in each of the graphs, there is evidence that the model fits well. When the two functions differ in at least one of the graphs, there is evidence that the model does not fit well. The two functions correspond well for some independent variables and deviate for others largely because of the outlier, Pete Rose, the career hits leader.

Output 22.7.1 Marginal Model Plot for the 1986 Baseball Data



Marginal model plots provide useful diagnostic information for both linear and generalized linear models.You can use the %Marginal macro to produce marginal model plots after fitting a model. You must specify the dependent variable, the independent variables, and the variable that contains the predicted values. By default, the last SAS data set that is created is displayed, a loess fit function is used, and the macro chooses the number of graphs (from 2 to 15) to display in each panel and the size of the panel. Multiple panels are displayed when there are too many graphs to fit in a single panel. You can specify the number and size of the graphs in each panel rather than relying on the defaults.

The following steps illustrate the default settings:

```
%let vars =  nAtBat  nHits  nHome  nRuns  nRBI  nBB  nOuts nAssts
           CrAtBat CrHits CrHome CrRuns CrRbi CrBB;

proc glm noprint data=sashelp.baseball;
   class div;
   model logsalary = div &vars;
   output out=pvals p=p;
run; quit;

%marginal(independents=&vars, dependent=LogSalary, predicted=p)
```

Figure 22.7.1 displays the results.

%Marginal Macro Options

You must specify the DEPENDENT=, INDEPENDENTS=, and PREDICTED= options. The remaining options have defaults and are not required. The macro provides the following options:

DATA=*SAS-data-set*
specifies the SAS data set to be displayed. If you do not specify this option, the %Marginal macro uses the most recently created SAS data set.

DEPENDENT=*variable*
specifies the dependent variable. You must specify this option.

GOPTS=*begingraph-statement-options*
specifies GTL BEGINGRAPH statement graph size options. The default size depends on the number of rows and columns in the panel.

Rows and Columns	Default GOPTS=
1×1, 2×2	DESIGNWIDTH=DEFAULTDESIGNHEIGHT
1×2	DESIGNHEIGHT=360PX
2×3	
3×4	DESIGNHEIGHT=520PX
3×3, 4×4, others	DESIGNHEIGHT=DEFAULTDESIGNWIDTH

INDEPENDENTS=*variable-list*
specifies the independent variables. You must specify this option.

PANEL=< *number-of-rows* **> <** *number-of-columns* **>**
specifies the number of rows and columns in the panel. Specify two values (rows then columns) or one value (when the number of rows equals the number of columns). For example, PANEL=2 3. Additional panels are automatically created when you have more graphs than available cells in the current panel. The default depends on both the number of independent variables and the number of graphs that can fit in a panel. For example, the default for 16 cells (14 independent variables, one predicted values plot, and a legend) is a 4×4 panel; the default for 17 cells (15 independent variables, one predicted values plot, and a legend) is two 3×3 panels (each panel has eight graphs and a legend that fills 18 cells). By default, the macro chooses to fill two panels that have fewer cells rather than to display more graphs in a first panel that has more cells and the last few graphs in the second panel. Multiple panels always have the same number of cells, although some cells might not be filled in the last panel. Defaults include 2×2, 2×3, 3×3, 3×4, and 4×4.

PREDICTED=*variable*
specifies the variable that contains the predicted values. You must specify this option.

SMOOTH=LOESS | PBSPLINE | REGRESSION
specifies the smoothing method.

- LOESS specifies a loess fit
- PBSPLINE specifies a penalized B-spline
- REGRESSION specifies a linear regression
- REGRESSION along with SMOOTHOPTS=DEGREE=3 specifies a cubic-polynomial regression.

By default, SMOOTH=LOESS.

SMOOTHOPTS=*smooth-options*
specifies GTL options for smoothing. These options are added to the GTL LOESSPLOT, PBSPLINEPLOT or REGRESSIONPLOT statement. By default, no options are specified.

%Marginal Macro Examples

The following steps show how to create displays that have varying numbers of graphs:

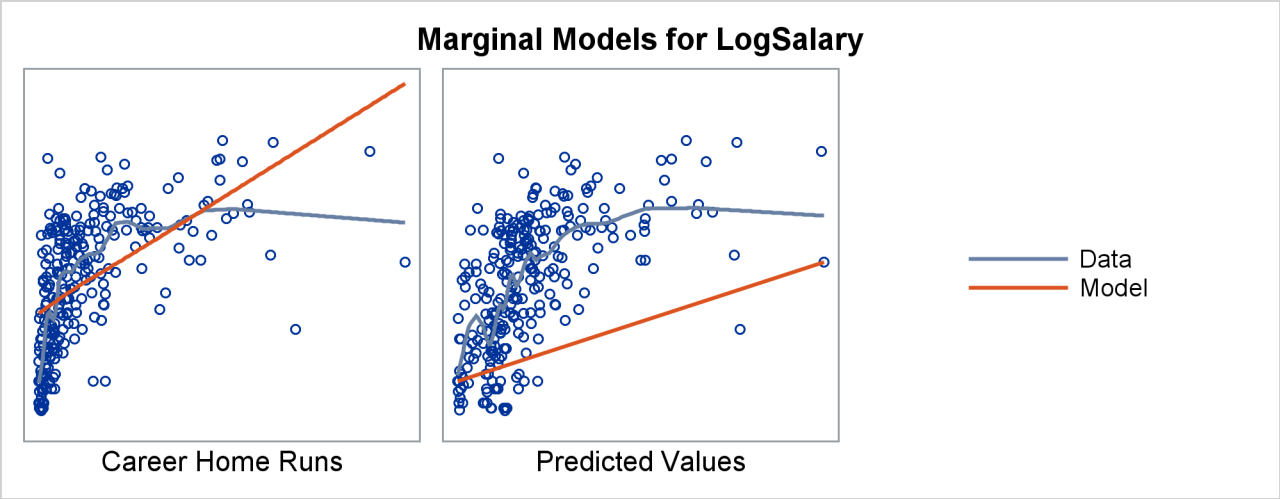
```
proc glm noprint data=sashelp.baseball;
   class div;
   model logsalary = div CrHome;
   output out=pvals p=p;
run; quit;

%marginal(independents=CrHome, dependent=LogSalary, predicted=p,
          panel=1 3, gopts=designheight=250px)

proc glm noprint data=sashelp.baseball;
   class div;
   model logsalary = div CrHome CrRuns;
   output out=pvals p=p;
run; quit;

%marginal(independents=CrHome CrRuns, dependent=LogSalary, predicted=p)

%let vars = nHits nHome nRuns CrHits CrHome CrRuns CrRbi;

proc glm noprint data=sashelp.baseball;
   class div;
   model logsalary = div &vars;
   output out=pvals p=p;
run; quit;

%marginal(independents=&vars, dependent=LogSalary, predicted=p)

proc glm noprint data=sashelp.baseball;
   class div;
   model logsalary = div CrAtBat CrHits CrHome;
   output out=pvals p=p;
```

```
run; quit;

%marginal(independents=CrAtBat CrHits CrHome,
          dependent=LogSalary, predicted=p, panel=1)
```

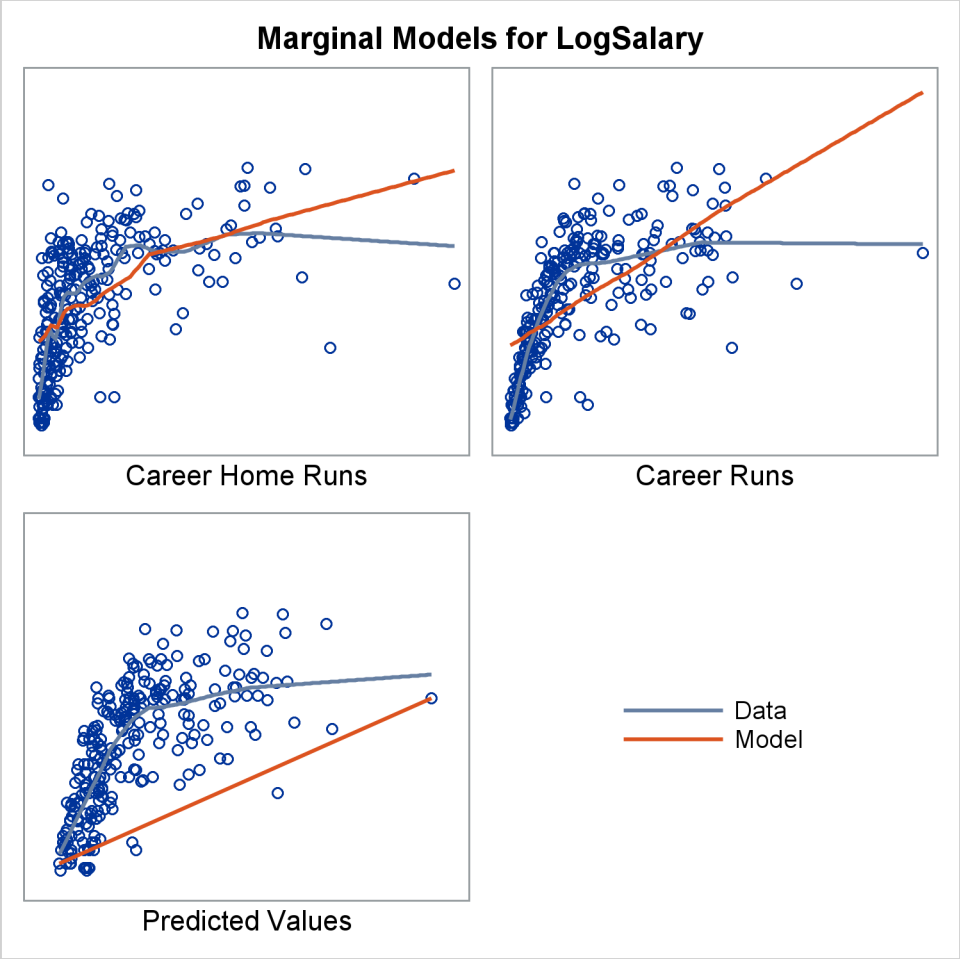
In the first model, the PANEL=1 3 and GOPTS=DESIGNHEIGHT=250PX options create a 1 × 3 display. The results are displayed in Figure 22.7.2. In the second model, the default display for three graphs is 2 × 2. The results are displayed in Figure 22.7.3. In the third model, the default display for eight graphs is 3 × 3. The results are displayed in Figure 22.7.4. In the fourth model, the PANEL=1 option creates single graphs. The results are displayed in Figure 22.7.5.

When you display graphs in panels, the legend occupies one of the cells. When you produce single graphs, the legend appears inside each graph.

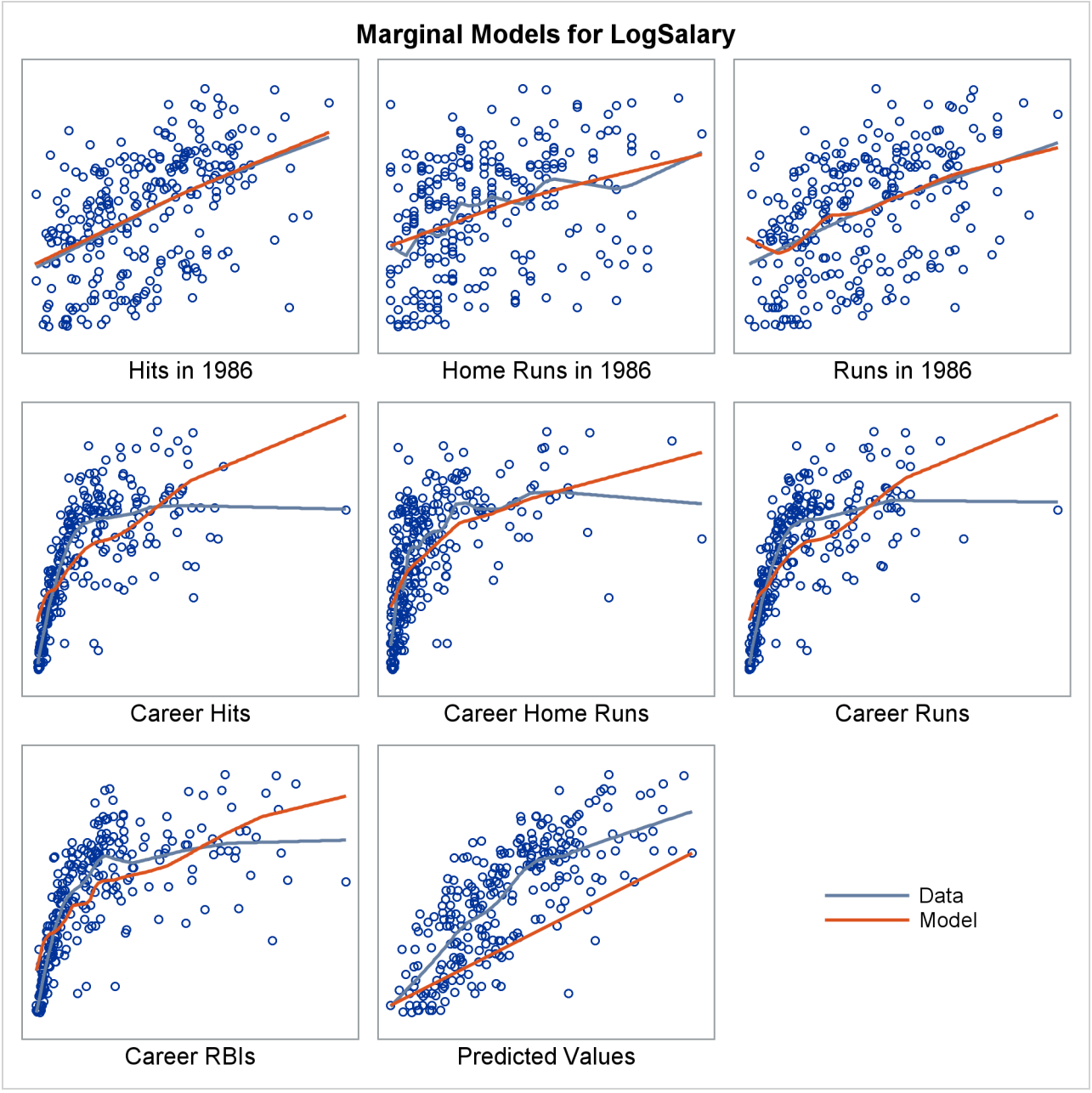
Output 22.7.2 1 by 3 Display



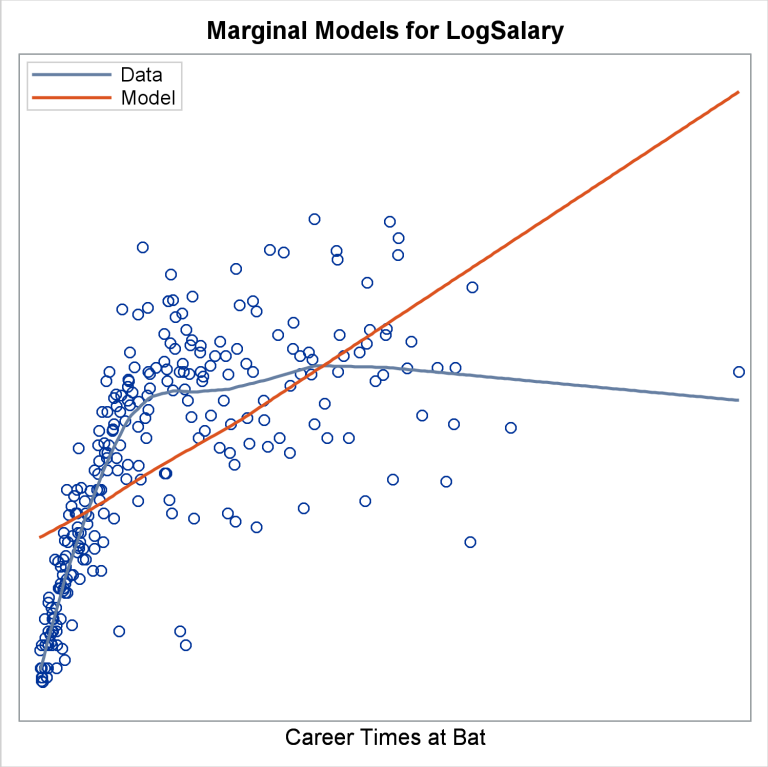
Output 22.7.3 2 by 2 Display

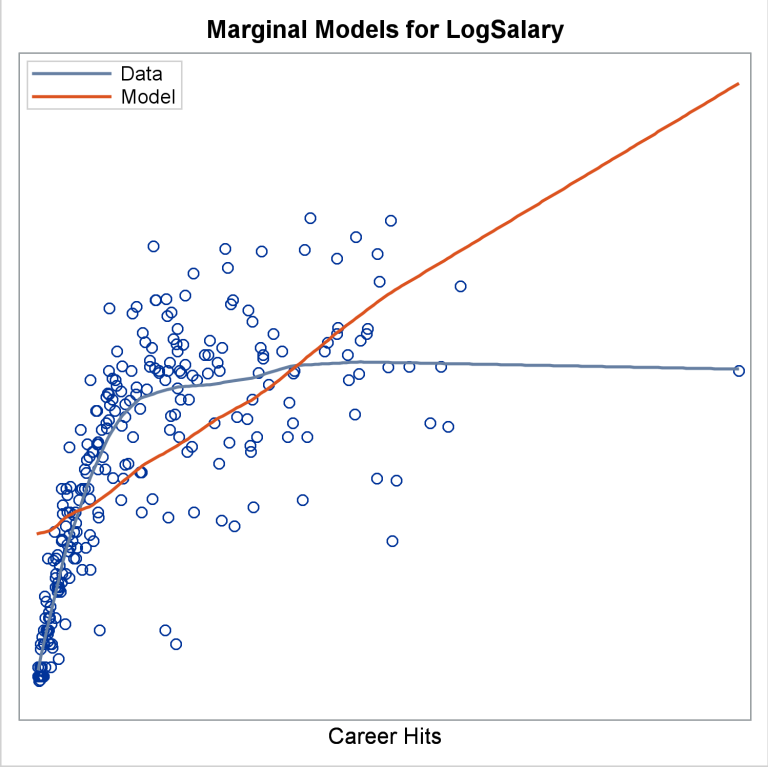


Output 22.7.4 3 by 3 Display

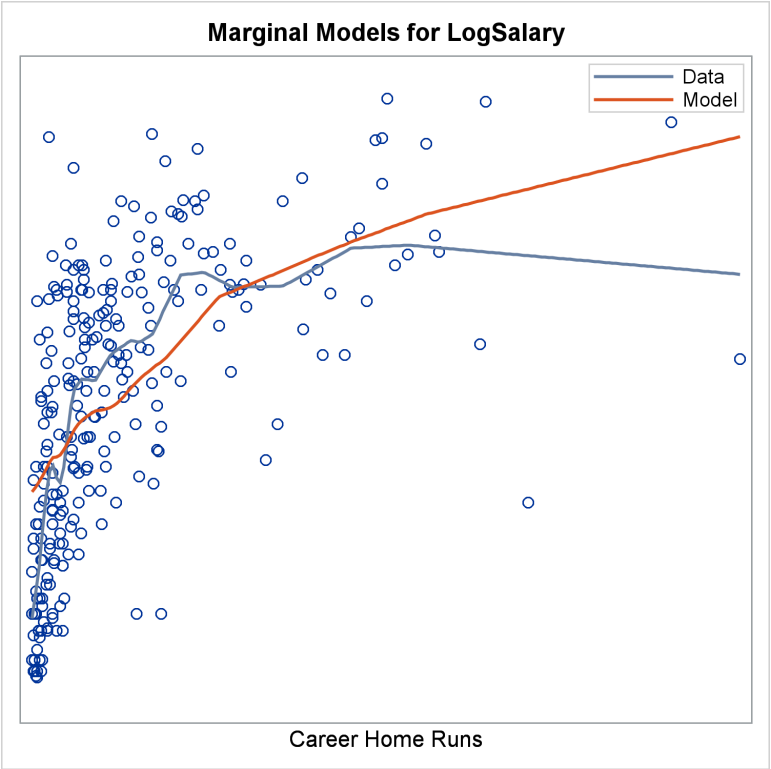


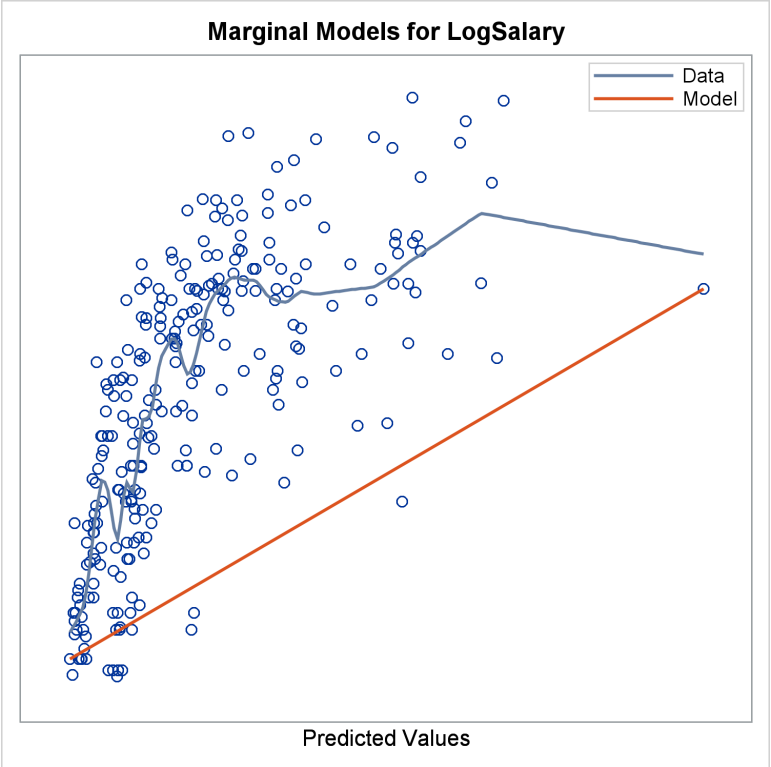
Output 22.7.5 Single-Graph Display





Output 22.7.5 *continued*





%Marginal Macro

You can use the %Marginal macro to create marginal model plots. The macro begins by initializing some macro variables and by checking the options. The macro gathers variable names, checks them, and stores them in the order that you specify. The macro determines the number of rows and columns in the panel along with the size. Next, it determines the endpoints for the predicted values plot. The macro constructs a custom graph template, and then one or more graphs are made using the template. The macro code follows.

```
                        /*-----------------------------------------------*/
%macro marginal(        /*-------------Marginal Model Plots--------------*/
                        /*-----------------------------------------------*/
       data=_last_,     /* Input SAS data set.                           */
       dependent=,      /* Dependent variable.                           */
       independents=,   /* Continuous independent variables.             */
       predicted=,      /* Predicted values variable.                    */
       smooth=loess,    /* LOESS, PBSPLINE, or REGRESSION.               */
       smoothopts=,     /* GTL options for smoothing.  They are added    */
                        /* to the GTL LOESSPLOT, PBSPLINEPLOT, or        */
                        /* REGRESSIONPLOT statement.                     */
       panel=,          /* Number of rows and columns in the panel.      */
                        /* The default depends on the number of          */
                        /* independent variables.  Specify two values    */
                        /* (rows columns) or one when rows equal         */
                        /* columns.  For example: PANEL=2 3.             */
       gopts=,          /* GTL BEGINGRAPH statement graph size options.  */
                        /* The default depends on the number of          */
                        /* rows and columns in the panel.                */
                        /* Example: DESIGNHEIGHT=800px.                  */
       );;              /*-----------------------------------------------*/

%local abort holdlast nvars i ncells savenote time tmpl paneled rows cols ettl;

%let time     = %sysfunc(datetime());
%let abort    = 0;
%let tmpl     = 0;
%let holdlast = &syslast;
%let savenote = %sysfunc(getoption(notes));

options nonotes;

data _null_; /* basic option processing and checking */
   length l $ 67;
   call symputx('ettl', quote('Marginal Models for ' || symget('dependent')));
   l = symget('data');
   if l = ' ' or lowcase(l) = '_last_' then  /* default, last data set */
      call symputx('data', symget('syslast'));

   l = symget('panel');
   r = input(scan(l, 1, ' '), best12.);
   c = input(scan(l, 2, ' '), best12.);
   if nmiss(c) and n(r) then c = r;
   call symputx('rows', r);
   call symputx('cols', c);
```

```
   if n(r,c) eq 2 and
      not (r = round(r) and c = round(c) and r ge 1 and c ge 1) then do;
      put "ERROR: PANEL=&panel must specify positive integers.";
      call symputx('abort', 1);
   end;

   l = lowcase(symget('smooth'));
   if not (trim(l) in ('regression', 'loess', 'pbspline')) then do;
      put 'ERROR: Invalid SMOOTH= option.  Specify: '
          'REGRESSION, LOESS, or PBSPLINE.';
      call symputx('abort', 1);
   end;

   if symget('predicted') eq ' ' or symget('dependent') eq ' ' or
      symget('independents') eq ' ' then do;
      put 'ERROR: The PREDICTED=, DEPENDENT=, and INDEPENDENTS= '
         'options ' 'must all ' 'be specified.';
      call symputx('abort', 1);
   end;

   if _error_ then call symputx('abort', 1);
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

data _null_;  /* check the variable names,                          */
   set &data; /* some problematic specifications generate SAS errors */
   &predicted = mean(of &predicted &dependent &independents);
   &dependent = mean(of &predicted &dependent &independents);
   if _error_ then do;
      put 'ERROR: All variables must be numeric.';
      _error_ = 0;
      call symputx('abort', 1);
   end;
   stop;
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

data __tmpcon; /* reorder vars if necessary */
   length &independents 8;
   set &data(obs=1 keep=&independents);
   if _error_ then call symputx('abort', 1);
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

proc contents data=__tmpcon noprint out=__tmpcon(keep=name varnum);
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

proc sort data=__tmpcon out=__tmpcon(drop=varnum);
   by varnum;
```

```
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

data __tmpcon(keep=name); /* augment name list, determine panel size */
   length gopts $ 32;
   set __tmpcon nobs=n;

   output;

   if n = _n_ then do;
      call symputx('nvars', n + 1);

      r = input(symget('rows'), best12.);
      c = input(symget('cols'), best12.);
      if n(r,c) eq 0 then do;
         %let i = 8,9,10,16,17,18,19,20,21,30,31,32;
         select;
            when(n le 2)          do; r = 2; c = 2; end;
            when(n in (3,4))      do; r = 2; c = 3; end;
            when(n in (5,6,7,15)) do; r = 3; c = 3; end;
            when(n in (&i))       do; r = 3; c = 4; end;
            otherwise             do; r = 4; c = 4; end;
         end;
         call symputx('rows', r);
         call symputx('cols', c);
      end;
      call symputx('paneled', r * c gt 1);
      if symget('gopts') eq ' ' then do;
         select;
            when(r eq 1 and c eq 1) gopts = 'designwidth=defaultdesignheight';
            when(r eq 1 and c eq 2) gopts = 'designheight=360px';
            when(r eq 2 and c eq 2) gopts = 'designwidth=defaultdesignheight';
            when(r eq 2 and c eq 3) gopts = ' ';
            when(r eq 3 and c eq 4) gopts = 'designheight=520px';
            otherwise               gopts = 'designheight=defaultdesignwidth';
         end;
         call symputx('gopts', gopts);
      end;
      name = ' '; output;
   end;

   if _error_ then call symputx('abort', 1);
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

data __tmpdat(keep=_x _y); /* get extremes for yhat series plot */
   set &data end=eof;
   retain _minx _maxx _miny _maxy .;
   if nmiss(_minx) then _minx = &predicted;
   if nmiss(_maxx) then _maxx = &predicted;
   if n(&predicted,_minx,&dependent) = 3 and &predicted lt _minx then do;
      _minx = &predicted; _miny = &dependent;
```

```
   end;
   if n(&predicted,_maxx,&dependent) = 3 and &predicted gt _maxx then do;
      _maxx = &predicted; _maxy = &dependent;
   end;
   if eof then do;
      _x = _minx; _y = _miny; output;
      _x = _maxx; _y = _maxy; output;
   end;
   if _error_ then call symputx('abort', 1);
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

data __tmpdat;
   merge &data __tmpdat;
   if _error_ then call symputx('abort', 1);
run;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

%let tmpl = 1;
proc template;
   define statgraph __marginal;
      dynamic %do i = 1 %to &rows*&cols - &paneled; _ivar&i %end; ncells pplot;
      begingraph / &gopts;
         entrytitle &ettl;
         legenditem type=line name='a' / lineattrs=GraphFit  label='Data';
         legenditem type=line name='b' / lineattrs=GraphFit2 label='Model';
         layout lattice / columns=&cols rows=&rows
                rowdatarange=unionall rowgutter=10 columngutter=10;
            %do i = 1 %to &rows * &cols - &paneled; /* ordinary cells */
               if(&i le ncells)
                  layout overlay / yaxisopts=(display=none)
                                   xaxisopts=(display=(label));
                     scatterplot  y=&dependent  x=_ivar&i;
                     &smooth.plot y=&dependent  x=_ivar&i / &smoothopts;
                     &smooth.plot y=&predicted  x=_ivar&i / &smoothopts
                                                            lineattrs=graphfit2;
                     %if not &paneled %then %do; /* not paneled?           */
                        layout gridded /         /* then put legend inside */
                           autoalign=(topright topleft bottomright bottomleft);
                           discretelegend 'a' 'b' / location=inside across=1;
                        endlayout;
                     %end;
                  endlayout;
               endif;
            %end;
            if(pplot) /* predicted values plot is handled differently */
               layout overlay / yaxisopts=(display=none)
                      xaxisopts=(display=(label) label='Predicted Values');
                  scatterplot  y=&dependent  x=&predicted;
                  &smooth.plot y=&dependent  x=&predicted / &smoothopts;
                  seriesplot   y=_y x=_x / lineattrs=graphfit2;
                  %if not &paneled %then %do;
```

```
                    layout gridded /
                           autoalign=(topright topleft bottomright bottomleft);
                       discretelegend 'a' 'b' / location=inside across=1;
                    endlayout;
                 %end;
              endlayout;
           endif;
           %if &paneled %then %do; /* legend in a cell by itself if paneled */
              layout overlay / yaxisopts=(display=none)
                               xaxisopts=(display=none);
                 discretelegend 'a' 'b' / location=inside across=1
                                          border=false;
              endlayout;
           %end;
        endlayout;
     endgraph;
  end;
run; quit;

%if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;

%let i = 1;
%do %while(&i le &nvars); /* create as many panels as is necessary */
   data _null_;
      set __tmpcon(firstobs=&i);
      length l $ 2000;
      retain l 'dynamic';
      if name ne ' ' then /* construct independent var list */
         l = catx(' ', l, '_ivar' || put(_n_, 3. -L), '=', quote(trim(name)));
      if name eq ' ' or _n_ eq (&rows * &cols - &paneled) then do;
         call symputx('pplot', name eq ' '); /* yhat plot in this panel? */
         call symputx('dynamics', l);
         call symputx('i', &i + _n_);
         call symputx('ncells', _n_ - (name = ' '));
         if _error_ then call symputx('abort', 1);
         stop;
      end;
      if _error_ then call symputx('abort', 1);
   run;

   %if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;


   proc sgrender data=__tmpdat template=__marginal;
      &dynamics ncells=&ncells pplot=&pplot;
   run;

   %if &syserr > 4 %then %let abort = 1; %if &abort %then %goto endit;
%end;

%endit:

proc datasets nolist; delete __tmpcon __tmpdat; run; quit;
```

```
%if &syserr > 4 %then %let abort = 1;

%if &tmpl and not %symexist(tmplsave) %then %do;
   proc template; delete __marginal; run; quit;
   %if &syserr > 4 %then %let abort = 1;
%end;

%let syslast = &holdlast;

options &savenote;

%if &abort %then %do;
   %if &syserr = 1016 or &syserr = 116  %then %put ERROR: Insufficient memory.;
   %if &syserr = 2000 or &syserr = 3000 %then
      %put ERROR: Syntax error.  Check your macro arguments for validity.;
   %put ERROR: The MARGINAL macro ended abnormally.;
%end;

%let time = %sysfunc(round(%sysevalf(%sysfunc(datetime()) - &time), 0.01));
%put NOTE: The MARGINAL macro used &time seconds.;

%mend;
```

Understanding Conditional Template Logic

You do not need to understand how the %Marginal macro works to use it. However, if you are interested, this section explains how this macro creates varying graph templates that depend on the number of variables and other options. This section can help you understand how you can use both GTL and macro conditional logic to construct flexible templates.

The %Marginal macro creates a graph template and then uses it one or more times to make graphs. The macro uses one block of code to create the template for all cases: single graphs, the default sizes (2×2, 2×3, 3×3, 3×4, and 4×4), and other sizes. It displays the legend inside a cell for multicell panels and inside the graph for single graphs. It constructs the final graph of predicted values in a different way from the way that it constructs the graphs that correspond to the independent variables. It does all of this in only 49 lines by using macro %DO loops and by using conditional logic (macro %IF statements and GTL IF and ENDIF statements) to populate cells. Before going into detail about the general logic, here are some specific examples.

For a 2×2 panel, the generated template is as follows:

```
proc template;
   define statgraph __marginal;
      dynamic _ivar1 _ivar2 _ivar3 ncells pplot;
      begingraph / designwidth=defaultdesignheight;
         entrytitle "Marginal Models for LogSalary";
         legenditem type=line name='a' / lineattrs=GRAPHFIT label='Data';
         legenditem type=line name='b' / lineattrs=GRAPHFIT2 label='Model';
         layout lattice / columns=2 rows=2 rowdatarange=unionall rowgutter=10
            columngutter=10;
            if (1 LE NCELLS)
               layout overlay / yaxisopts=(display=none)
                                xaxisopts=(display=(label));
```

```
            scatterplot y=LOGSALARY x=_IVAR1;
            loessplot y=LOGSALARY x=_IVAR1 /;
            loessplot y=P x=_IVAR1 / lineattrs=GRAPHFIT2;
         endlayout;
      endif;
      if (2 LE NCELLS)
         layout overlay / yaxisopts=(display=none)
                          xaxisopts=(display=(label));
            scatterplot y=LOGSALARY x=_IVAR2;
            loessplot y=LOGSALARY x=_IVAR2 /;
            loessplot y=P x=_IVAR2 / lineattrs=GRAPHFIT2;
         endlayout;
      endif;
      if (3 LE NCELLS)
         layout overlay / yaxisopts=(display=none)
                          xaxisopts=(display=(label));
            scatterplot y=LOGSALARY x=_IVAR3;
            loessplot y=LOGSALARY x=_IVAR3 /;
            loessplot y=P x=_IVAR3 / lineattrs=GRAPHFIT2;
         endlayout;
      endif;
      if (PPLOT)
         layout overlay / yaxisopts=(display=none)
                          xaxisopts=(display=(label)
            label='Predicted Values');
            scatterplot y=LOGSALARY x=P;
            loessplot y=LOGSALARY x=P /;
            seriesplot y=_Y x=_X / lineattrs=GRAPHFIT2;
         endlayout;
      endif;
      layout overlay / yaxisopts=(display=none)
                       xaxisopts=(display=none);
         discretelegend 'a' 'b' / location=inside across=1 border=false;
      endlayout;
   endlayout;
endgraph;
end;
run; quit;
```

The template can handle up to three independent variables (and no predicted values variable) or two independent variables along with one predicted values variable. (When there are three or more independent variables, the predicted values variable appears in a subsequent panel.) The following statements control which graphs are produced: IF (1 LE NCELLS), IF (2 LE NCELLS), IF (3 LE NCELLS), and IF (PPLOT).

This template can be used with the SGRENDER procedure and two independent variables as follows:

```
proc sgrender data=__tmpdat template=__marginal;
   dynamic _ivar1 = "nAtBat" _ivar2 = "nHits" ncells=2 pplot=1;
run;
```

Dynamic variables enable templates to be more flexible. The values of the dynamic variables are substituted into the template when PROC SGRENDER runs. The dynamic variables that begin with _ivar specify the names of the independent variables. Other dynamic variables specify that there are two independent variable cells (because ncells = 2, the third LAYOUT OVERLAY block is not used) and that the predicted values plot is displayed (because pplot = 1, which is a true Boolean constant).

This template can be used with PROC SGRENDER and four independent variables as follows:

```
proc sgrender data=__tmpdat template=__marginal;
   dynamic _ivar1 = "nAtBat" _ivar2 = "nHits" _ivar3 = "nhome" ncells=3 pplot=0;
run;

proc sgrender data=__tmpdat template=__marginal;
   dynamic _ivar1 = "nrbi" ncells=1 pplot=1;
run;
```

When the dynamic variable ncells = 3, three independent variable plots are produced. When the dynamic variable pplot = 1, a predicted values plot is produced that follows the specified number of independent variable plots. The first step produces three independent variable plots, and the second step creates one independent variable plot and the predicted values plot. A legend appears in the cell after the last graph in both panels.

When you specify PANEL=1 to create single graphs, the generated template is as follows:

```
proc template;
   define statgraph __marginal;
      dynamic _ivar1 ncells pplot;
      begingraph / designwidth=defaultdesignheight;
         entrytitle "Marginal Models for LogSalary";
         legenditem type=line name='a' / lineattrs=GRAPHFIT label='Data';
         legenditem type=line name='b' / lineattrs=GRAPHFIT2 label='Model';
         layout lattice / columns=1 rows=1 rowdatarange=unionall
                          rowgutter=10 columngutter=10;
            if (1 LE NCELLS)
               layout overlay / yaxisopts=(display=none)
                                xaxisopts=(display=(label));
                  scatterplot y=LOGSALARY x=_IVAR1;
                  loessplot y=LOGSALARY x=_IVAR1 /;
                  loessplot y=P x=_IVAR1 / lineattrs=GRAPHFIT2;
                  layout gridded / autoalign=(topright topleft bottomright
                                              bottomleft);
                     discretelegend 'a' 'b' / location=inside across=1;
                  endlayout;
               endlayout;
            endif;
            if (PPLOT)
               layout overlay / yaxisopts=(display=none)
                      xaxisopts=(display=(label) label='Predicted Values');
                  scatterplot y=LOGSALARY x=P;
                  loessplot y=LOGSALARY x=P /;
                  seriesplot y=_Y x=_X / lineattrs=GRAPHFIT2;
                  layout gridded / autoalign=(topright topleft bottomright
                     bottomleft);
                     discretelegend 'a' 'b' / location=inside across=1;
                  endlayout;
               endlayout;
            endif;
         endlayout;
      endgraph;
   end;
run; quit;
```

Now, the legend appears inside each graph rather than in a separate cell. When you have two independent variables, the template is used in three PROC SGRENDER steps:

```
proc sgrender data=__tmpdat template=__marginal;
   dynamic _ivar1 = "nAtBat" ncells=1 pplot=0;
run;

proc sgrender data=__tmpdat template=__marginal;
   dynamic _ivar1 = "nHits" ncells=1 pplot=0;
run;

proc sgrender data=__tmpdat template=__marginal;
   dynamic ncells=0 pplot=1;
run;
```

When the dynamic variable ncells = 1, an independent variable plot is produced. When the dynamic variable pplot = 1, a predicted values plot is produced. The first two steps produce independent variable plots, and the third step creates a predicted values plot.

The part of the %Marginal macro that generates the template is as follows:

```
proc template;
   define statgraph __marginal;
      dynamic %do i = 1 %to &rows*&cols - &paneled; _ivar&i %end; ncells pplot;
      begingraph / &gopts;
         entrytitle &ettl;
         legenditem type=line name='a' / lineattrs=GraphFit  label='Data';
         legenditem type=line name='b' / lineattrs=GraphFit2 label='Model';
         layout lattice / columns=&cols rows=&rows
                rowdatarange=unionall rowgutter=10 columngutter=10;
            %do i = 1 %to &rows * &cols - &paneled; /* ordinary cells */
               if(&i le ncells)
                  layout overlay / yaxisopts=(display=none)
                                   xaxisopts=(display=(label));
                     scatterplot  y=&dependent  x=_ivar&i;
                     &smooth.plot y=&dependent  x=_ivar&i / &smoothopts;
                     &smooth.plot y=&predicted  x=_ivar&i / &smoothopts
                                                        lineattrs=graphfit2;
                     %if not &paneled %then %do; /* not paneled?           */
                        layout gridded /         /* then put legend inside */
                           autoalign=(topright topleft bottomright bottomleft);
                           discretelegend 'a' 'b' / location=inside across=1;
                           endlayout;
                        %end;
                     endlayout;
                  endif;
               %end;
            if(pplot) /* predicted values plot is handled differently */
               layout overlay / yaxisopts=(display=none)
                      xaxisopts=(display=(label) label='Predicted Values');
                  scatterplot  y=&dependent  x=&predicted;
                  &smooth.plot y=&dependent  x=&predicted / &smoothopts;
                  seriesplot   y=_y x=_x / lineattrs=graphfit2;
                  %if not &paneled %then %do;
```

```
                    layout gridded /
                           autoalign=(topright topleft bottomright bottomleft);
                       discretelegend 'a' 'b' / location=inside across=1;
                       endlayout;
                    %end;
                 endlayout;
              endif;
           %if &paneled %then %do; /* legend in a cell by itself if paneled */
              layout overlay / yaxisopts=(display=none)
                               xaxisopts=(display=none);
                 discretelegend 'a' 'b' / location=inside across=1
                                          border=false;
              endlayout;
           %end;
        endlayout;
     endgraph;
  end;
run; quit;
```

The LAYOUT LATTICE statement in the GTL constructs a display of &rows and &cols (macro variables that are set in a preceding step). Inside the LAYOUT LATTICE block, there are LAYOUT OVERLAY blocks for the independent variables and for the predicted values, and optionally one for the legend. When the graph is a panel that contains multiple cells, the macro %DO loop generates &rows × &cols – 1 LAYOUT OVERLAY blocks for the independent variables. Additional code produces one LAYOUT OVERLAY block for the predicted values plot and a final LAYOUT OVERLAY block for the legend. This same template is used when there are multiple panels, and the LAYOUT OVERLAY block for the predicted values plot is suppressed in all but the last panel by the IF(PPLOT) statement.

When there is one cell per panel, two GTL IF statements control which type of graph is displayed. When there is one cell per panel, a macro %IF statement adds a legend inside the body of the plot, and a different macro %IF statement excludes the creation of a separate legend cell. When there are multiple cells per panel, the legend is added to the last cell and does not appear inside the graphs.

The macro language enables you to generate a long template by using a minimum number of statements. There is one LAYOUT OVERLAY block in the macro for independent variable plots. In contrast, the generated template can be much longer and has &rows × &cols – 1 LAYOUT OVERLAY blocks because of the macro %DO loop, each conditionally executed by a GTL IF statement. The template also contains two LAYOUT OVERLAY blocks that are outside the macro %DO loop. Although you would never know it by looking at the PROC TEMPLATE SOURCE statement template display, many of the multicell templates that SAS provides for use in its analytical procedures are created in a similar way by using macros and dynamic variables.

References

Cook, R. D., and Weisberg, S. (1997). "Graphics for Assessing the Adequacy of Regression Models." *Journal of the American Statistical Association* 92:490–499.

Fox, J., and Weisberg, S. (2011). *An R Companion to Applied Regression*. Thousand Oaks, CA: Sage Publications.

Kuhfeld, W. F. (2009). "Modifying ODS Statistical Graphics Templates in SAS 9.2." http://support.sas.com/rnd/app/papers/ods/modtmplt.pdf. Revision of paper by the same title that was published in the Proceedings of the SAS Global Forum 2009 Conference.

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

Subject Index

Syntax Index