Chapter 4

Graph Template Modification

Contents

This chapter explains how to modify the templates that SAS provides.

4.1 Dynamic Variables and Graph Template Modification

This section, along with many other sections throughout the rest of this book, examines some of the graph templates that SAS provides, shows what they have in common and what is different, and shows ways to modify them, thus allowing you to customize the graphs that SAS produces. Understanding how to customize the graphs that SAS produces also helps you learn aspects of the GTL that you can use when writing your own templates. The first few templates that are discussed are chosen because they are small, and they provide complementary insights into template modification. Most templates that SAS provides are more complicated than these.

The default templates supplied by SAS for statistical procedures are often lengthy and complex, because they provide ODS Graphics with comprehensive and detailed information about graph construction. They contain many of the same statements that you saw in the previous examples. However, they often contain statements that do not appear in the previous examples. Some statements are typically not needed when you write your own templates, yet they are important for the templates that SAS provides. This section concentrates on the DYNAMIC statement and also introduces the NOTES and LINEPARM statements.

The rest of this book assumes that the following statements are in effect:

```
ods path (prepend) work.templat(update);
ods graphics on;
ods trace on;
```

The ODS PATH statement stores the modified templates in the SAS WORK library so that they are deleted at the end of your SAS session. The ODS GRAPHICS statement enables ODS Graphics so that graphs are automatically produced by the analytical procedures. The ODS TRACE statement enables ODS trace output so that information about each graph and table (including the graph or table name and the template name) is displayed in the SAS log.

The following template is one of the simplest graph templates that SAS provides for a statistical procedure:

```
define statgraph Stat.MDS.Graphics.Fit;
   notes "MDS Fit Plot";
   dynamic head;
   begingraph / designwidth=defaultdesignheight;
      entrytitle HEAD;
      layout overlayequated / equatetype=square;
         scatterplot y=FITDATA x=FITDIST / markerattrs=(size=5px);
         lineparm slope=1 x=0 y=0 / extend=true lineattrs=GRAPHREFERENCE;
      endlayout;
   endgraph;
end;
```

This template, supplied by SAS 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 NOTES statement explains the purpose of the template and does not affect any aspect of the graph. The DYNAMIC statement provides a dynamic variable, Head, that is set by the procedure and used to customize the template at procedure run time. This is analogous to the MVAR and NMVAR statements that are used in previous examples. In previous examples, you specify an MVAR or an NMVAR statement, and then set macro variables for the template to use at procedure run time. In the templates that SAS provides, and with a DYNAMIC statement, the SAS procedure sets dynamic variables for the template to use at procedure run time. If the entry title had been constant, it would have been specified directly in the template. It is not set directly since it can vary depending on the nature of the analysis.

The box that contains the plot is square since the design width is set to the default design height by the option: DESIGNWIDTH=DEFAULTDESIGNHEIGHT on the BEGINGRAPH statement. The LAYOUT OVERLAYEQUATED statement along with the option EQUATETYPE=SQUARE produces a square plot with the axes 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 run-time evaluation of the dynamic variable Head. 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 used by default in most styles.

The LINEPARM statement draws the diagonal reference line, which shows the line of perfect fit. The options X=0 and Y=0 specify coordinates of a point on the line, and the option SLOPE= specifies the slope. The slope of 1 provides a diagonal line. The values of these parameters can be constant, dynamic variables, or macro variables. The option EXTEND=TRUE draws the line to the axes, ignoring any offset area. The LINEATTRS= option specifies the `GraphReference` style element so that the line has the same style as the default reference line style drawn by the REFERENCELINE statement. Notice that the line is drawn after the scatter plot, since its statement comes last. If the statements had been in the reverse order, most or all of the line would have been obscured with large analyses.

The following statements display another of the simpler template definitions—the definition of the scatter plot available in PROC KDE:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      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;
      EndGraph;
   end;
run;
```

The PROC TEMPLATE and RUN statements have been added to show how you would compile the template if you wanted to modify it.

The DYNAMIC statement specifies four dynamic variables. The dynamic variables _Var1Name and _Var2Name contain the names of the X and Y variables, respectively. The dynamic variables _Var1Label and _Var2Label contain the labels. The names are used in the title. The labels are not used anywhere, but you could modify the template to use them.

There are two ways that you might modify a template. Both are illustrated next. First, you might want to modify the template to use with the procedure for which it is written, PROC KDE. You do so by modifying and compiling the template, and then running the procedure to use the modified template. Alternatively, you might want to use the template outside of the procedure with PROC SGRENDER.

The following steps use PROC KDE and a modified template that uses variable labels instead of the variable names:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryTitle "Distribution of " _VAR1label " by " _VAR2label;
         layout Overlay / xaxisopts=(offsetmin=0.05 offsetmax=0.05)
                          yaxisopts=(offsetmin=0.05 offsetmax=0.05);
            ScatterPlot x=X y=Y / markerattrs=GRAPHDATADEFAULT;
         EndLayout;
      EndGraph;
   end;
run;
```

Figure 4.1 Plot with Variable Labels

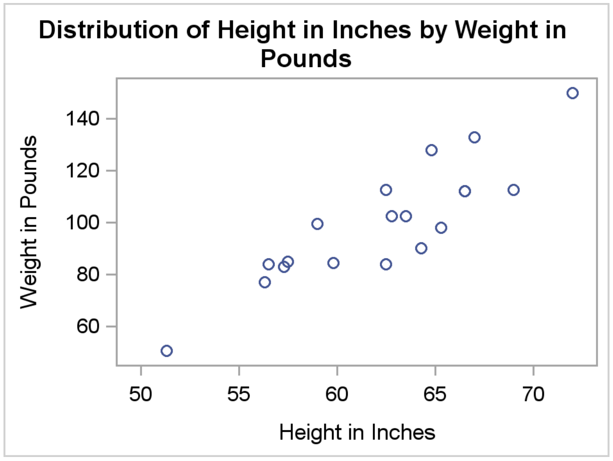
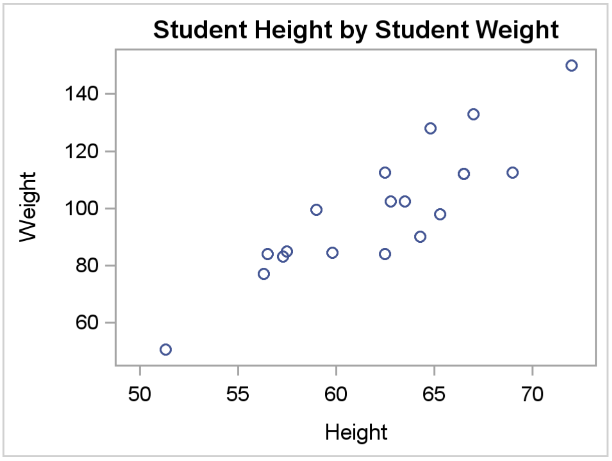


Figure 4.2 PROC SGRENDER's DYNAMIC Statement



```
proc kde data=sashelp.class;
   bivar height weight / plots=scatter;
   label height = 'Height in Inches' weight = 'Weight in Pounds';
run;
```

The results are displayed in Figure 4.1. This template modification is straightforward. The dynamic variables containing the variable labels are used in the entry title in place of the dynamic variables containing the variable names.

The following steps modify the original template for use outside of PROC KDE but with a data set made from the data object that PROC KDE uses to make the plot:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryTitle _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;
      EndGraph;
   end;
run;

ods graphics on;
proc kde data=sashelp.class;
   ods output scatterplot=sp;
   bivar height weight / plots=scatter;
run;

proc sgrender data=sp template=Stat.KDE.Graphics.ScatterPlot;
   dynamic  _VAR1NAME='Student Height' _var2name='Student Weight';
run;
```

Figure 4.3 Modifying Column Names

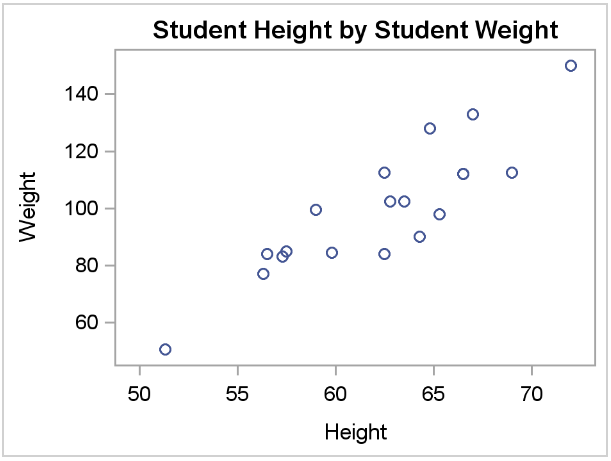
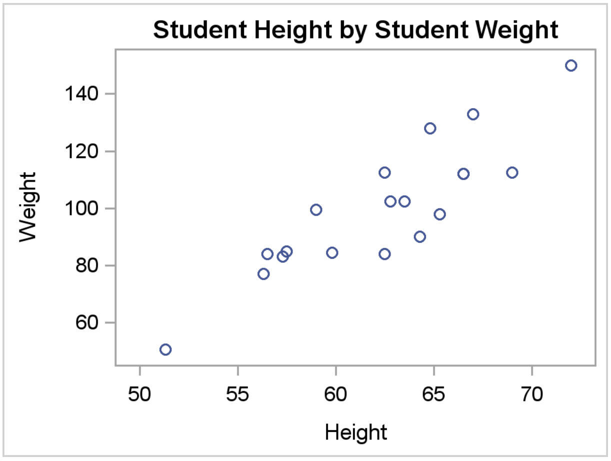


Figure 4.4 Modifying Dynamics



The results are displayed in Figure 4.2. The ODS OUTPUT statement in PROC KDE outputs to a SAS data set the data object that PROC KDE creates and uses to make the plot. Then that data set is input to PROC SGRENDER along with the modified template. The original template could have just as easily been used. This usage of PROC SGRENDER differs from the uses seen previously since this template has dynamic variables. A DYNAMIC statement must be provided with PROC SGRENDER to set the values of the dynamic variables. Note that all of the dynamic variables that are mentioned in the template do not need to be set. Just set the ones that you need. In this case, it is the two variable names. The values of these two dynamic variables appear in the title. The values that are specified on the DYNAMIC statement are not valid SAS variable names, but that does not matter since the dynamic variables are only used to provide title text. You could modify the template for use outside of PROC KDE but with the original Class data set as follows:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryTitle _VAR1NAME " by " _VAR2NAME;
         layout Overlay / xaxisopts=(offsetmin=0.05 offsetmax=0.05)
                          yaxisopts=(offsetmin=0.05 offsetmax=0.05);
            ScatterPlot x=height y=weight / markerattrs=GRAPHDATADEFAULT;
         EndLayout;
      EndGraph;
   end;
run;

proc sgrender data=sashelp.class template=Stat.KDE.Graphics.ScatterPlot;
   dynamic  _VAR1NAME='Student Height' _var2name='Student Weight';
run;
```

The results are displayed in Figure 4.3. In this step, the SCATTERPLOT statement is modified to specify X=HEIGHT and Y=WEIGHT instead of the original X=X and Y=Y.

The following steps modify the template for use outside of PROC KDE, with the original Sashelp.Class data set, but in a very different way:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL x y;
      BeginGraph;
         EntryTitle _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;
      EndGraph;
   end;
run;

proc sgrender data=sashelp.class template=Stat.KDE.Graphics.ScatterPlot;
   dynamic  _VAR1NAME='Student Height' _var2name='Student Weight'
            x='height' y='weight';
run;
```

The results are displayed in Figure 4.4. The original template was written for two data object columns named X and Y. The data set contains two variables named Height and Weight. You can make this template work with the original data set by adding X and Y as dynamic variables in the template definition and then adding `x='height' y='weight'` to the DYNAMIC statement in the PROC SGRENDER step. This is more complicated than directly changing the X= and Y= specifications in this template. However, if this were a complicated template with the same columns being used multiple places, this would be the easier approach.

Figure 4.5 Default Scree Plot

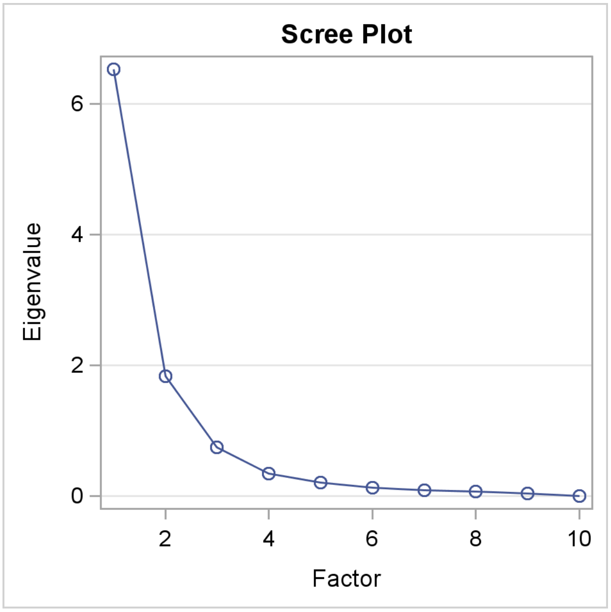
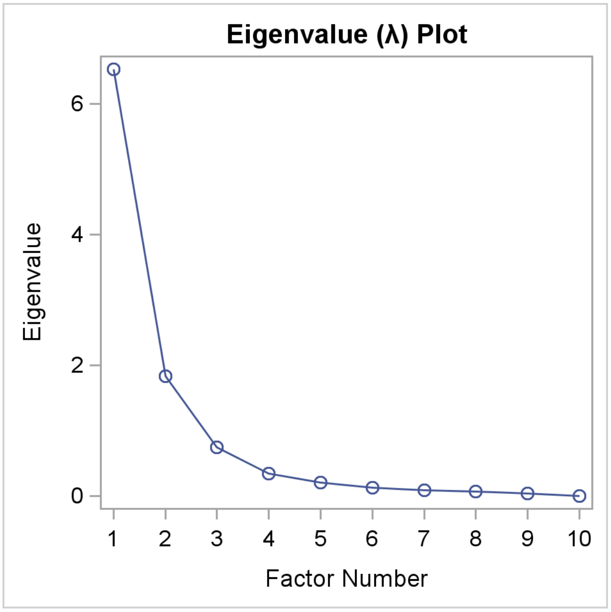


Figure 4.6 Modified Scree Plot



4.2 Changing Titles and Axis Labels

This example shows how to change titles and axis labels in the context of a very simple template. The following step runs PROC FACTOR and produces the eigenvalue (or scree) plot displayed in Figure 4.5:

```
proc factor data=sashelp.cars plots(unpack)=scree;
run;
```

The **`plots(unpack)=scree`** option produces the scree plot by itself—“unpacked” from its usual location as part of a two-graph panel with the variance-explained plot.

The ODS trace output for the scree plot is as follows:

```
Name:        ScreePlot
Label:       Scree Plot
Template:    Stat.Factor.Graphics.ScreePlot1
Path:        Factor.InitialSolution.ScreeAndVarExp.ScreePlot
```

The following statements display the graph template for the scree plot:

```
proc template;
   source Stat.Factor.Graphics.ScreePlot1;
run;
```

The template source statements are as follows:

```
define statgraph Stat.Factor.Graphics.ScreePlot1;
   notes "Scree Plot for Extracted Eigenvalues";
   BeginGraph / designwidth=DefaultDesignHeight;
      Entrytitle "Scree Plot" / border=false;
      layout overlay / yaxisopts=(label="Eigenvalue" gridDisplay=auto_on)
         xaxisopts=(label="Factor" linearopts=(integer=true));
         seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
      endlayout;
   EndGraph;
end;
```

The DESIGNWIDTH=DEFAULTDESIGNHEIGHT option on the BEGINGRAPH statement specifies that the outer box which contains the graph should be a square whose width is equal to the default graph height. This creates a graph that is designed for a size 480 pixels wide by 480 pixels high. The default design size is 640 pixels wide by 480 pixels high. The ENTRYTITLE statement provides the graph title, in this case “Scree Plot”. The BORDER=FALSE option specifies that the title is displayed without a border. In fact, this is the default behavior, so the option is unnecessary. However, it is not unusual to see default specifications in the templates that SAS provides.

The LAYOUT OVERLAY statement provides the label “Eigenvalue” for the vertical or Y axis, provides the label “Factor” for the horizontal or X axis, specifies that grid lines should be produced for the Y axis when the output style favors grids, and specifies that the X axis ticks must be integers. The LINEAROPTS= option is used for options specific to standard axes that depict a linear scaling (as opposed to LOGOPTS=, which is used for log-scale axes).

The graph is a series plot. The Y axis column in the ODS data object is named EigenValue, and the X axis column in the ODS data object is named Number (the factor number). The standard series plot display is a series of lines, but the DISPLAY=ALL option additionally displays the markers (in this case, circles) for the data values.

Notice that the title and the axis labels are all specified directly as literal character strings in this template. You can change any of them and submit the results to SAS. From then on, until you change or delete your custom template in Work.Templat or until you end your SAS session, you will see your customization whenever you run PROC FACTOR.

The following example adds a PROC TEMPLATE statement and a RUN statement, changes the title and an axis label, specifies explicit tick values, and removes the grid and the unnecessary BORDER= option:

```
proc template;
   define statgraph Stat.Factor.Graphics.ScreePlot1;
      notes "Scree Plot for Extracted Eigenvalues";
      BeginGraph / designwidth=DefaultDesignHeight;
         Entrytitle "Eigenvalue ((*ESC*){Unicode Lambda}) Plot";
         layout overlay / yaxisopts=(label="Eigenvalue")
            xaxisopts=(label="Factor Number"
                       linearopts=(tickvaluelist=(1 2 3 4 5 6 7 8 9 10)));
            seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
         endlayout;
      EndGraph;
   end;
run;
```

The title now contains the Greek letter λ, which is specified as an escape sequence followed by a Unicode specification. The tick value list is specified in full because the GTL does not accept standard SAS shorthand lists. The only output from this step is the following log note:

```
NOTE: STATGRAPH 'Stat.Factor.Graphics.ScreePlot1' has been saved to: WORK.TEMPLAT
```

The following step uses the new template to create a scree plot and produces Figure 4.6:

```
proc factor data=sashelp.cars plots(unpack)=scree;
run;
```

The following step restores the default template:

```
proc template;
   delete Stat.Factor.Graphics.ScreePlot1;
run;
```

The only output from this step is the following log note:

```
NOTE: 'Stat.Factor.Graphics.ScreePlot1' has been deleted from: WORK.TEMPLAT
```

Figure 4.7 Default Box Plots

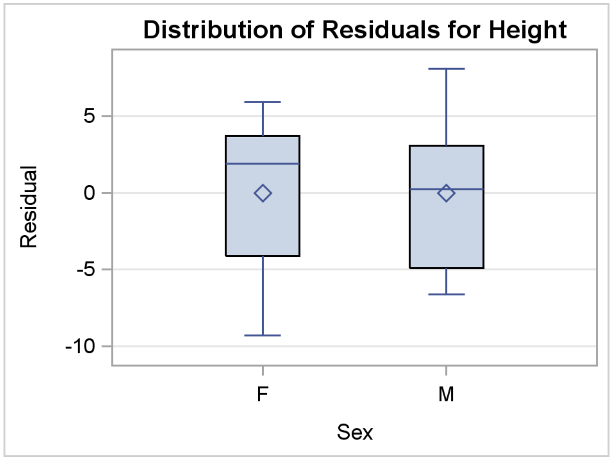
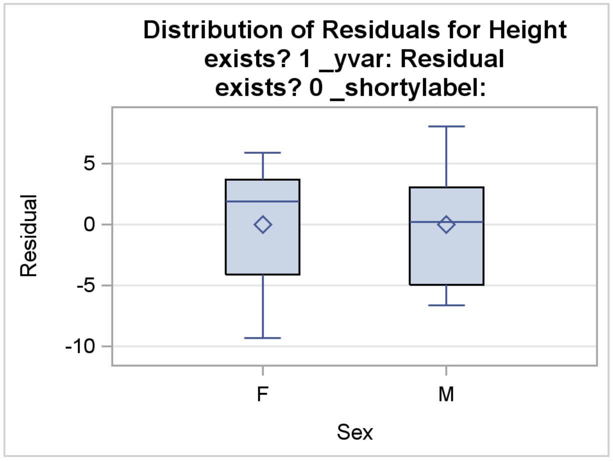


Figure 4.8 Examining Dynamic Variables



4.3 Changing Titles and Axis Labels Set by Dynamic Variables

In the previous example, the title and both axis labels are specified directly as literal strings in the template. In this example, the procedure provides the title and axis labels at run time. The following step uses the GLIMMIX procedure to create the box plot in Figure 4.7:

```
proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    model height = sex;
run;
```

The trace output for the box plot is as follows:

```
Name:        BoxPlot
Label:       Residuals by Sex
Template:    Stat.Glimmix.Graphics.BoxPlot
Path:        Glimmix.Boxplots.BoxPlot
```

The following statements display the graph template for the box plot:

```
proc template;
   source Stat.Glimmix.Graphics.BoxPlot;
run;
```

The template source statements are as follows:

```
define statgraph Stat.Glimmix.Graphics.BoxPlot;
   dynamic _TITLE _YVAR _SHORTYLABEL;
   BeginGraph;
      entrytitle _TITLE;
      layout overlay / yaxisopts=(gridDisplay=auto_on shortlabel=_SHORTYLABEL)
         xaxisopts=(discreteopts=(tickvaluefitpolicy=rotatethin));
         boxplot y=_YVAR x=LEVEL / labelfar=on datalabel=OUTLABEL
            primary=true freq=FREQ;
      endlayout;
   EndGraph;
end;
```

The procedure uses dynamic variables to provide text strings and option values to the template. In this case, the dynamic variables are a title, a variable name for the Y axis, and a short variable label for the Y axis. Note that dynamic variables cannot provide any arbitrary syntax. For example, they can provide title text and values of options, but not option names, statement names, layout names, and so on.

Axes can have labels and optionally short labels. The label is displayed if there is sufficient space. Otherwise, the short label is used instead. Axis labels (and short labels) can be specified in the template with a literal string, in the template through a dynamic variable, or implicitly. The axis label comes from the first source that provides a value: the LABEL= option in the template (or the SHORTLABEL= option), the data object column label, or the data object column name.

As a SAS user, you cannot peek into the SAS procedure code to see how the dynamic variables, column names, and column labels are set. However, you can do a bit of detective work to learn about these things. The following steps illustrate one approach:

```
proc template;
   define statgraph Stat.Glimmix.Graphics.BoxPlot;
      dynamic _TITLE _YVAR _SHORTYLABEL;
      BeginGraph;
         entrytitle _TITLE;
         entrytitle "exists? " eval(exists(_yvar)) " _yvar: " _yvar;
         entrytitle "exists? " eval(exists(_shortylabel))
                    " _shortylabel: " _shortylabel;
         layout overlay / yaxisopts=(gridDisplay=auto_on shortlabel=_SHORTYLABEL)
            xaxisopts=(discreteopts=(tickvaluefitpolicy=rotatethin));
            boxplot y=_YVAR x=LEVEL / labelfar=on datalabel=OUTLABEL
               primary=true freq=FREQ;
         endlayout;
      EndGraph;
   end;

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    ods output boxplot=bp;
    model height = sex;
run;

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

The graph is displayed in Figure 4.8, and the data object contents are displayed in Figure 4.9. The first title is unmodified and simply displays the value of the _Title dynamic variable. Following that, this template is temporarily modified by adding two new ENTRYTITLE statements to report on both the existence and the value of two of the dynamic variables. The expression `eval(exists(`*dynamic-variable*`))` resolves to 1 (for true) when the dynamic variable is set by the procedure and 0 (for false) when it is not set. It is not unusual for a procedure to conditionally set dynamic variables. A specification of *option=dynamic* is ignored when the dynamic variable does not exist. After the existence information is displayed, the value (if any) is displayed.

Figure 4.9 Contents of a Data Object

```
                              The CONTENTS Procedure

                            Variables in Creation Order

  #    Variable                             Type    Len    Format    Label

  1    BOX__YVAR_X_LEVEL_DATALABEL_O__Y     Num       8              Residual
  2    BOX__YVAR_X_LEVEL_DATALABEL_O_ST     Char     10
  3    BOX__YVAR_X_LEVEL_DATALABEL_O__X     Char      1              Sex
  4    BOX__YVAR_X_LEVEL_DATALABEL_O_DL     Num       8    BEST8.    Index
  5    Residual                             Num       8
  6    Level                                Char      1              Sex
  7    OutLabel                             Num       8    BEST8.    Index
```

Figure 4.8 shows that the Y axis column is Residual and the short Y axis label is undefined. The PROC CONTENTS information confirms that the data object has a column called Residual for the Y axis and a column called Level with a label of "Sex" for the X axis. The Y axis column name and the X axis column label become the axis labels. The contents information also displays other columns in the data object.[1]

There are a number of ways that you can modify templates beyond simply adding or replacing text. For example, you can use the dynamic variables that are provided in creative ways, such as using the title as a label for the Y axis: **`yaxisopts=(label=_title . . .)`**.

[1]Data objects come in many varied forms. You should not expect them to be pretty or well organized for display or subsequent processing. Although you can process them in any way you choose, they are designed for input to one or more templates and very little else. On some occasions, extra columns or extra dynamic variables might be created but not used. These represent cases where the procedure writer recognized possibilities for future processing and tried to facilitate them. They might be helpful when they occur, but most data objects or templates do not have such information. This data object has a number of manufactured and verbose names. You often see names like these when the values that are plotted are statistics of some sort or are computed by ODS Graphics.

Figure 4.10 Modified Box Plots

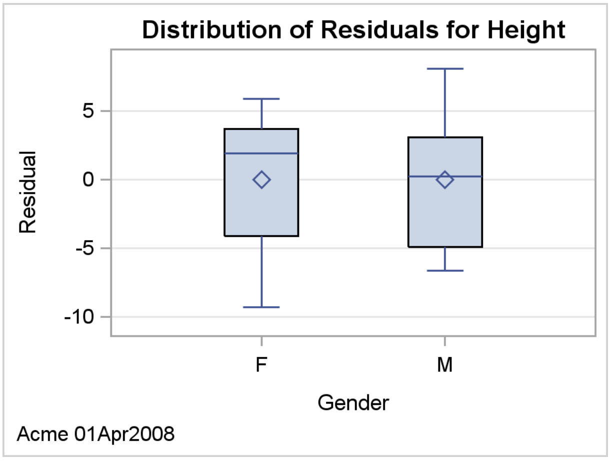
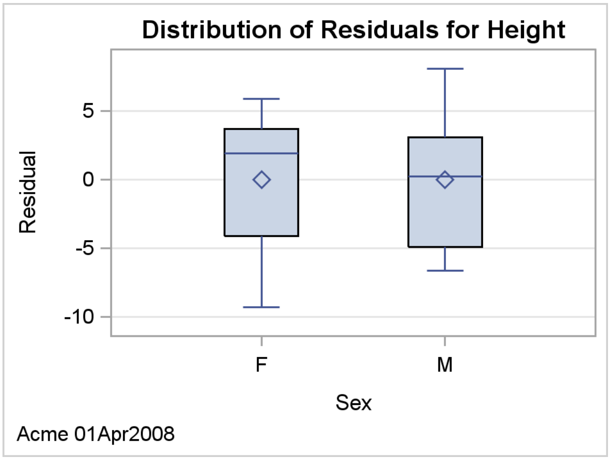


Figure 4.11 Footnote Added with a Macro



This next example will instead replace the X axis label and add a footnote (horizontally aligned on the left using a font that is appropriate for footnotes or second title lines), both through macro variables, as follows:

```
proc template;
   define statgraph Stat.Glimmix.Graphics.BoxPlot;
      dynamic _TITLE _YVAR _SHORTYLABEL;
      mvar datetag xlabel;
      BeginGraph;
         entrytitle _TITLE;
         entryfootnote halign=left textattrs=graphvaluetext datetag;
         layout overlay / yaxisopts=(gridDisplay=auto_on shortlabel=_SHORTYLABEL)
            xaxisopts=(label=xlabel
                       discreteopts=(tickvaluefitpolicy=rotatethin));
            boxplot y=_YVAR x=LEVEL / labelfar=on datalabel=OUTLABEL
               primary=true freq=FREQ;
         endlayout;
      EndGraph;
   end;
run;

%let DateTag = Acme 01Apr2008;
%let xlabel  = Gender;

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    ods output boxplot=bp;
    model height = sex;
run;
```

The new graph is displayed in Figure 4.10.

In this example, there is a new footnote, which comes from the value of the macro variable DateTag. DateTag is specified in the MVAR (macro variable) statement. When the MVAR statement is used, the template is compiled and the value of the macro variable is substituted when the template is used by the procedure. This approach lets you modify and compile the template once and then use it repeatedly with different values of the macro variable without ever having to recompile the template.

You usually use this approach when you make persistent changes in Sasuser.Templat or some other permanent item store. An alternate approach is to use the following statement without using an MVAR statement:

```
entryfootnote "&datetag";
```

In this approach, the template is compiled and the value of the macro variable is substituted at compile time. The value cannot change in this approach unless you recompile the template. In this case, the approach does not matter because the template is compiled and immediately used.

The X axis label is set to "Gender" using the macro variable xlabel. The X axis change is ad hoc, so changes such as this are usually made temporarily.

The following step restores the default template:

```
proc template;
   delete Stat.Glimmix.Graphics.BoxPlot;
run;
```

If your only goal is to add or change a footnote or title, there is an easier, alternative mechanism. SAS provides a new autocall macro, **ModTmplt**, that you can use for this purpose (Kuhfeld 2009). This macro is used in the following example:

```
title;
footnote 'halign=left textattrs=graphvaluetext "Acme 01Apr2008"';
%modtmplt(template=Stat.Glimmix.Graphics.BoxPlot, steps=t,
          options=titles noquotes)
footnote;

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    ods output boxplot=bp;
    model height = sex;
run;

%modtmplt(template=Stat.Glimmix.Graphics.BoxPlot, steps=d)
```

The TITLE statement clears the default title of "The SAS System". The FOOTNOTE statement provides the footnote along with options to place the footnote on the left using the font that is used for values. The **ModTmplt** macro modifies the box plot template. Only one macro step is run: the template modification step (STEPS=T). OPTIONS=TITLES adds SAS system titles and footnotes (those specified in TITLE and FOOTNOTE statements) to the existing graph titles and footnotes. OPTIONS=NOQUOTES moves the footnote from the FOOTNOTE statement to the ENTRYFOOTNOTE statement but without the outer quotes. You must specify this option if you want to specify ENTRYTITLE or ENTRYFOOTNOTE options in your TITLE or FOOTNOTE statement. The next FOOTNOTE statement clears the footnote so that it affects only the box plot template and does not otherwise affect the analysis. PROC GLIMMIX makes the plot. The final call to the macro deletes the modified template (STEPS=D). The results are displayed in Figure 4.11.

Figure 4.12 Transposed Scatter Plot with Labels

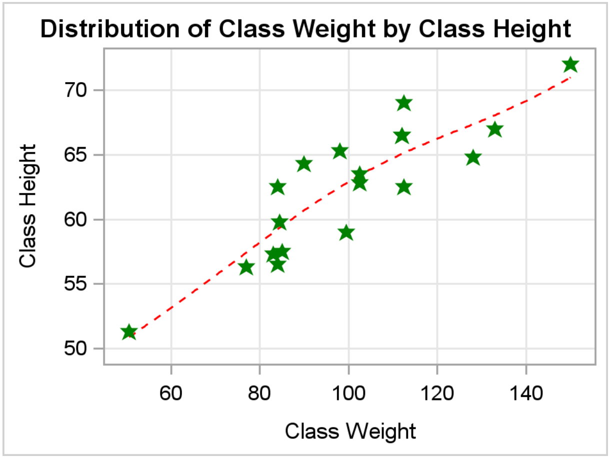
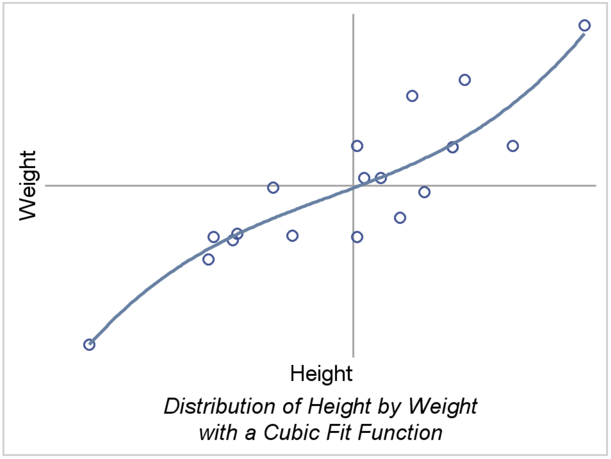


Figure 4.13 Scatterplot with Numerous Modifications



4.4 Modifying Colors, Lines, Markers, Axes, and Reference Lines

This example shows how to change the axes and the inside of the plot including the lines and markers. The following template source appeared in the section “Dynamic Variables and Graph Template Modification” on page 153:

```
define statgraph Stat.KDE.Graphics.ScatterPlot;
   dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
   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;
   EndGraph;
end;
```

The entry title is a mix of constant text and dynamic variables that provide variable names. The procedure writer has provided you with additional dynamic variables that provide the variable labels. This template also has offset options specified. These options are frequently used in scatter plots and other graphs. They add a small amount of white space to the left or bottom (OFFSETMIN=) and to the right or top (OFFSETMAX=) of the specified axis. The SCATTERPLOT statement has a MARKERATTRS= option, which references the `GraphDataDefault` style element. This style element also controls line color and thickness.

The following statements switch the Y and X axis variables, use variable labels instead of variable names in the title, change the marker characteristics, add a nonlinear penalized B-spline fit function, and add grids:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryTitle "Distribution of " _VAR2LABEL " by " _VAR1LABEL;
         layout Overlay /
            xaxisopts=(offsetmin=0.05 offsetmax=0.05 griddisplay=on)
            yaxisopts=(offsetmin=0.05 offsetmax=0.05 griddisplay=on);
            pbsplineplot x=y y=x / lineattrs=(color=red pattern=2 thickness=1);
            ScatterPlot  x=y y=x / markerattrs=(color=green size=5px
                                   symbol=starfilled weight=bold);
         EndLayout;
      EndGraph;
   end;
run;

proc kde data=sashelp.class;
   label height = 'Class Height' weight = 'Class Weight';
   bivar height weight / plots=scatter;
run;
```

The results are displayed in Figure 4.12.

Note that the addition of the variable labels in the PROC KDE step also changes the axis labels because no axis labels are explicitly specified in the template. The PBSPLINEPLOT statement includes the LINEATTRS= option which specifies the color (red), pattern (2, dashed), and thickness (1 pixel) of the fit function. The SCATTERPLOT statement includes the MARKERATTRS= option which specifies the color (green), size (5 pixels), symbol (a filled star), and weight (bold) of the marker or symbol.

The starting point for the next step is the original PROC KDE scatter plot template, rather than the modified template. The goal in this example is to produce the scatter plot displayed in Figure 4.13 with axes passing through the center of the data. The following steps make a highly modified scatter plot:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryFootNote "Distribution of " _VAR1NAME " by " _VAR2NAME;
         EntryFootNote "with a Cubic Fit Function";
         layout Overlay / walldisplay=none
            xaxisopts=(display=(label))
            yaxisopts=(display=(label));
            referenceline y=eval(mean(y));
            referenceline x=eval(mean(x));
            ScatterPlot x=X y=Y / markerattrs=GRAPHDATADEFAULT;
            regressionplot x=x y=y / degree=3;
         EndLayout;
      EndGraph;
   end;
run;
```

```
proc kde data=sashelp.class;
   bivar height weight / plots=scatter;
run;
```

The template is modified by adding an MVAR statement to use the mean height and mean weight. Specifically, a reference line is displayed at the mean for each axis. The title is changed to a footnote, and a second footnote is added. The LAYOUT OVERLAY statement now has a WALLDISPLAY=NONE option to suppress the axes, and only the labels are displayed on each axis. A cubic-polynomial fit function is also added. The results are displayed in Figure 4.13.

The LAYOUT OVERLAY block has four statements in it. The statements are executed in the order in which they are specified in the LAYOUT OVERLAY block. Reference lines are displayed first. Therefore any point or function that coincides with the reference line is displayed on top of the reference line. Similarly, the fit function is displayed rather than the points in the places where they coincide. You can vary the order of the statements if you prefer some other effect. The plot has no axes, no ticks, no tick labels, and no wall. (The wall is the area inside the plot axes, which can be a different color from the background color outside of the axes.) Instead, the plot simply has reference lines at the means and axis labels. Many variations can be tried. In the interest of space, several variations are discussed but their results are not shown.

The following statement displays a left axis and a bottom axis (but no top axis or right axis), and the color outside the axes matches the color inside:

```
 layout Overlay / walldisplay=none;
```

The following statement displays all four axes, and the color outside the axes matches the color inside:

```
 layout Overlay / walldisplay=(outline);
```

The following statement suppresses all axis information (the axes, the ticks, the tick labels, the axis labels, and the wall):

```
layout Overlay / walldisplay=none
   xaxisopts=(display=none) yaxisopts=(display=none);
```

Figure 4.14 Default Residual Histogram

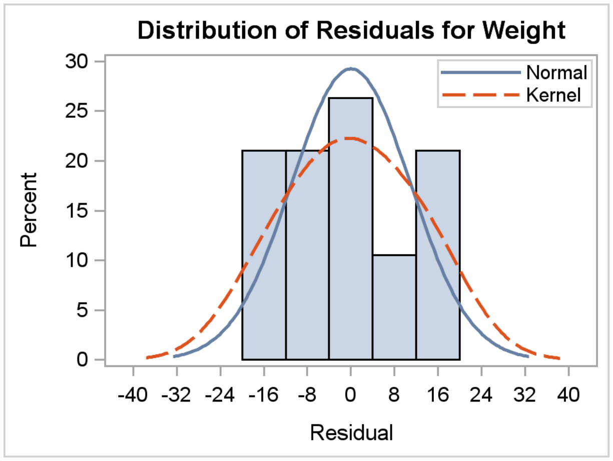
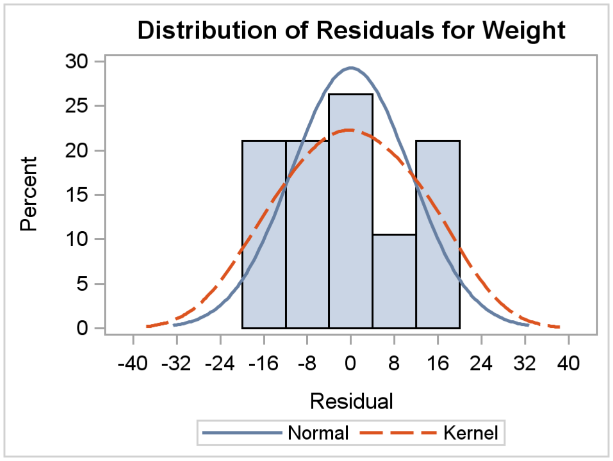


Figure 4.15 Residual Histogram with Repositioned Legend



4.5 Legends

This section creates a plot with a legend. The following step runs the GLM procedure and produces a residual histogram:

```
proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
   ods output residualhistogram=hr;
run;

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

This type of graph (shown in Figure 4.14) is used in many procedures.

The trace output for the residual histogram is as follows:

```
Name:        ResidualHistogram
Label:       Residual Histogram
Template:    Stat.GLM.Graphics.ResidualHistogram
Path:        GLM.ANOVA.Weight.DiagnosticPlots.ResidualHistogram
```

The following statements display the graph template for the residual histogram:

```
proc template;
   source Stat.GLM.Graphics.ResidualHistogram;
run;
```

The template source statements are as follows:

```
define statgraph Stat.GLM.Graphics.ResidualHistogram;
   notes "Residual Histogram with Overlayed Normal and Kernel";
   dynamic Residual _DEPNAME;
   BeginGraph;
      entrytitle "Distribution of Residuals" " for " _DEPNAME;
      layout overlay / xaxisopts=(label="Residual")
         yaxisopts=(label="Percent");
         histogram RESIDUAL / primary=true;
         densityplot RESIDUAL / name="Normal" legendlabel="Normal"
            lineattrs=GRAPHFIT;
         densityplot RESIDUAL / kernel () name="Kernel" legendlabel="Kernel"
            lineattrs=GRAPHFIT2;
         discretelegend "Normal" "Kernel" / across=1 location=inside
            autoalign=(topright topleft top);
      endlayout;
   EndGraph;
end;
```

This graph template creates a histogram of residuals. On top of the histogram is a normal density plot, and on top of both is a kernel density plot. Additionally, a legend is positioned inside the graph. The preferred position is in the top right, but ODS Graphics automatically repositions the legend in the top left or top center if there are conflicts between the legend and the histogram or functions in the top right.

The ENTRYTITLE statement specifies the title, which consists of literal text and a dynamic variable that contains the dependent variable name. The LAYOUT OVERLAY statement specifies the labels for both axes. Since the labels never vary in this template, they are specified directly in the template. The HISTOGRAM statement creates a histogram from the data object column named Residual. It is the primary statement in the overlay. The data columns from the primary statement determine the default axis types and default axis labels. By default, the first graph statement is the primary statement. Hence, in this case the PRIMARY= option is not needed. You must specify PRIMARY=TRUE when you want a statement other than the first to control the axes. The color, width, and line style for the normal density plot comes from the **`GraphFit`** style element (blue and solid in this style), and for the kernel density plot comes from the **`GraphFit2`** style element (red and dashed in this style). All graphs are based on the same data object column, Residual, and the kernel density plot uses default options for finding the kernel density.

The contents of the data object are displayed in Figure 4.16. From the original input variable Residual, six other variables are created by the HISTOGRAM and the two DENSITYPLOT statements. The X and Y axis variables for the density plot are BIN_RESIDUAL___X and BIN_RESIDUAL___Y; for the normal density plot they are NORMAL_RESIDUAL___X and NORMAL_RESIDUAL___Y; and for the kernel density plot they are KERNEL_RESIDUAL___X and KERNEL_RESIDUAL___Y. If you display the output data set created from this data object, you will see that the variables do not have the same number of nonmissing values. Some, such as the histogram values, have many fewer than the others. In this case, the computed density values have many more values than the raw residuals. Data objects are often constructed from pieces of very different sizes.

Figure 4.16 Contents of the Residual Histogram Data Object

```
                       The CONTENTS Procedure

                    Variables in Creation Order

          #    Variable               Type    Len    Label

          1    Dependent              Char      8
          2    BIN_RESIDUAL___X       Num       8    Residual
          3    BIN_RESIDUAL___Y       Num       8    Percent
          4    NORMAL_RESIDUAL___X    Num       8    Residual
          5    NORMAL_RESIDUAL___Y    Num       8    Percent
          6    KERNEL_RESIDUAL___X    Num       8    Residual
          7    KERNEL_RESIDUAL___Y    Num       8    Percent
          8    Residual               Num       8
```

All of the remaining options concern the legend. The discrete legend is produced by the DISCRETELEGEND statement. In contrast, a continuous legend is used to produce a color “thermometer” legend when point or surface colors vary continuously as a function of a third variable (see the section “Continuous Legend” on page 72). The legend is constructed from the statements named “Normal” and “Kernel” by the NAME= option in each of the two DENSITYPLOT statements. The labels for these two legend components come from the LEGENDLABEL= options. The legend has only one component in each row due to the ACROSS=1 option.

The following steps modify the graph by moving the legend outside the graph and by removing the ACROSS= option, which for this graph produces a legend with one row and two entries:

```
proc template;
   define statgraph Stat.GLM.Graphics.ResidualHistogram;
      notes "Residual Histogram with Overlayed Normal and Kernel";
      dynamic Residual _DEPNAME;
      BeginGraph;
         entrytitle "Distribution of Residuals" " for " _DEPNAME;
         layout overlay / xaxisopts=(label="Residual")
            yaxisopts=(label="Percent");
            histogram RESIDUAL / primary=true;
            densityplot RESIDUAL / name="Normal"
               legendlabel="Normal" lineattrs=GRAPHFIT;
            densityplot RESIDUAL / kernel ()
               name="Kernel" legendlabel="Kernel" lineattrs=GRAPHFIT2;
            discretelegend "Normal" "Kernel";
         endlayout;
      EndGraph;
   end;
run;

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

The results are displayed in Figure 4.15.

4.6 Lattice Layout and Panels

The templates that SAS provides have so far been simple in that they produce a graph with one panel. Those templates consist of a single LAYOUT OVERLAY block with one or more plotting statements inside. However, many graphs consist of two or more panels within a single display. For example, the scree plot displayed in Figure 4.5 is, by default, part of a two-panel display. It is produced when you run PROC FACTOR without the UNPACK option as follows:

```
proc factor data=sashelp.cars plots=scree;
run;
```

The graph is displayed in Figure 4.17.

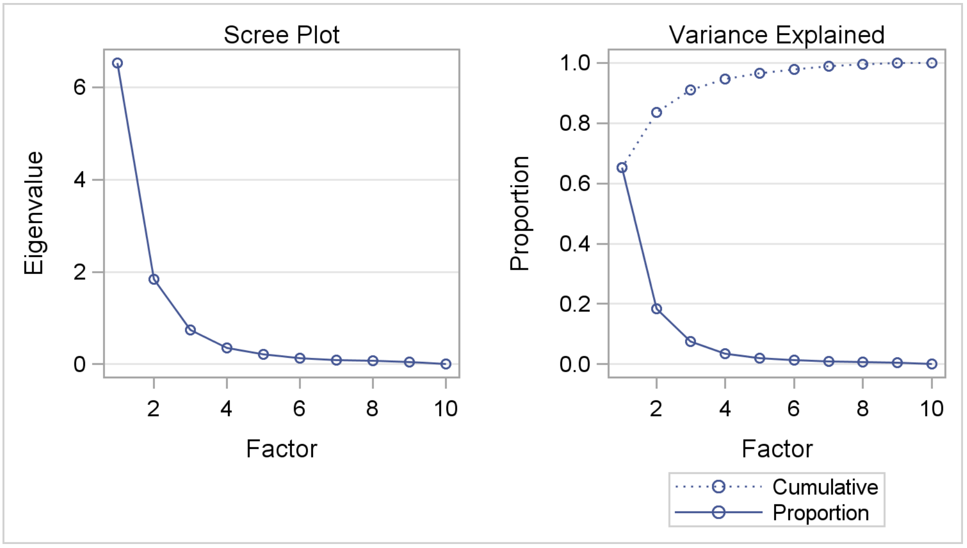
The trace output (not shown) shows that the template is called `Stat.Factor.Graphics.ScreePlot2`. A slight simplification of the template source statements is as follows:

```
define statgraph Stat.Factor.Graphics.ScreePlot2;
   notes "Scree and Proportion Variance Explained Plots";
   BeginGraph / DesignHeight=360px;
      layout lattice / rows=1 columns=2 columngutter=30;
         layout overlay / yaxisopts=(label="Eigenvalue" gridDisplay=auto_on)
            xaxisopts=(label="Factor" linearopts=(integer=true));
            entry "Scree Plot" / textattrs=GRAPHLABELTEXT location=outside;
            seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
         endlayout;
         layout overlay / yaxisopts=(label="Proportion" gridDisplay=auto_on)
            xaxisopts=(label="Factor" linearopts=(integer=true));
            entry "Variance Explained" / textattrs=GRAPHLABELTEXT
                                         location=outside;
            seriesplot y=PROPORTION x=NUMBER / display=ALL legendlabel=
               "Proportion" name="Proportion";
            seriesplot y=CUMULATIVE x=NUMBER / lineattrs=GRAPHDATADEFAULT (
               pattern=dot) display=ALL LegendLabel="Cumulative" name=
               "Cumulative" primary=true;
            DiscreteLegend "Cumulative" "Proportion" / across=1 border=1;
         endlayout;
      endlayout;
   EndGraph;
end;
```

The template begins with a BEGINGRAPH statement. Most templates do not contain a specific numerical size for the overall graph area. This template does, so that the two resulting plots are approximately square. At the default size, the plots are tall and thin. Note that size is a “design height” rather than a hardcoded size. The graph is designed at a height of 360 pixels, but it can be stretched or shrunk to other sizes while preserving the aspect ratio.

The next layer is a LAYOUT LATTICE block that creates a display with one row and two columns. Row and column gutters are frequently specified in lattice layouts. The COLUMNGUTTER=30 option ensures that there are 30 pixels between the two columns of graphs. (By default, the plots are closer than that.) Inside of the lattice layout are two LAYOUT OVERLAY blocks, one for each graph. Each individual LAYOUT OVERLAY block is designed in much the same way it would be designed if it were in a one-panel display. However, in practice it is not unusual for an "unpacked graph" (a graph produced in a single panel) to be different from the same graph packed into a display with other graphs.

Figure 4.17 Panel with Two Graphs



In some cases, the unpacked plot might have additional graph elements due to the increased room in the unpacked plot. One difference in the paneled plot is the title. In this case, the goal is to have two titles, one for each plot with no overall title. Hence, there is no ENTRYTITLE statement. Instead, there are two ENTRY statements, which place the title outside (and above) each plot by using the `GraphLabelText` style element. By default, without this style specification, the text would be smaller and would not look like other titles. The LOCATION=OUTSIDE option is an example of one of the undocumented options that are specified in some templates.

The last SERIESPLOT statement specifies the `lineattrs=GraphDataDefault(pattern=dot)` option. The properties of the line produced by this statement are controlled by the `GraphDataDefault` style element. However, one aspect of the style (namely the line pattern) is overridden and a dotted line is used instead. PATTERN= is one of the options in the LINEATTRS= option, rather than the name of a style element (which is `MarkerSymbol`). Since `GraphDataDefault` is the default style for the first series plot in the second overlay layout, the specification in the second series plot ensures that the two series have identical styles (except for one aspect) so that they can be distinguished in the legend. Most SAS/STAT templates do not hardcode graph elements such as this (the dotted line); usually they strictly use style elements. However, on occasion you will see explicit specifications. For example, the LOESS and TRANSREG procedures use the specification `MarkerAttrs=GraphData1(symbol=star size=15)` to mark the minimum of a function that is being optimized. This specification produces a large star. Numerical sizes like SIZE=15 are design sizes; they can be stretched or shrunk to other sizes while preserving the aspect ratio.

You can do many things to modify this template. You can change the titles, axis labels, colors, markers, and so on. All of these are illustrated in other parts of this book. You can switch the order of the layouts and put the variance-explained plot first; you can provide an overall title; and so on. However, rather than perform familiar or obvious changes, the rest of the book concentrates on understanding other aspects of the GTL and the complex templates that you might encounter.

4.7 Conditional Template Logic

This example explains the layout of a template with conditional logic and nested IF statements. You might need to understand conditional template logic when you determine which part of a template to modify. The survival estimate plot from the LIFETEST procedure has a long and complicated template, which includes nested IF statements. We assume at this point that you know how to run the procedure and find the name of the template. A very small part of the template is as follows:

```
define statgraph Stat.Lifetest.Graphics.ProductLimitSurvival;
   dynamic . . .;
   BeginGraph;
      if (NSTRATA=1)
         if (EXISTS(STRATUMID))
         entrytitle "Product-Limit Survival Estimate" " for " STRATUMID;
      else
         entrytitle "Product-Limit Survival Estimate";
      endif;
      if (PLOTATRISK)
         entrytitle "with Number of Subjects at Risk" / textattrs=
         GRAPHVALUETEXT;
      endif;
      layout overlay . . .;
          . . .
      endlayout;
      else
         entrytitle "Product-Limit Survival Estimates";
      if (EXISTS(SECONDTITLE))
         entrytitle SECONDTITLE / textattrs=GRAPHVALUETEXT;
      endif;
      layout overlay . . .;
         . . .
      endlayout;
      endif;
   EndGraph;
end;
```

This layout is confusing even when displayed like this with most details removed. In its original form at 145 lines, it is even more confusing. To understand the layout of this template, you must carefully evaluate the IF, ELSE, ENDIF, structure. The following step shows the structure manually re-indented, with additional details removed and additional white space added:

```
define statgraph Stat.Lifetest.Graphics.ProductLimitSurvival;
   dynamic . . .;
   BeginGraph;
      if (NSTRATA=1)

         if (EXISTS(STRATUMID)) entrytitle . . .;
         else entrytitle . . .;
         endif;

         if (PLOTATRISK) entrytitle . . .;
         endif;

         layout overlay ...;
             . . .
         endlayout;

      else

         entrytitle . . .;

         if (EXISTS(SECONDTITLE)) entrytitle . . .;
         endif;

         layout overlay . . .;
            . . .
         endlayout;

      endif;
   EndGraph;
end;
```

The IF and ELSE statements do not perform as do the similarly named statements in the DATA step. There are no DO and END statements. When the first IF condition is true, the statements under the first IF statement are executed until control reaches the ELSE statement at the same indentation level. The statements in the ELSE block include everything below the ELSE and through the ENDIF statement at the same level. Inside the first IF block, a title is provided if a condition is true. Otherwise a different title is provided, and that block ends with the first ENDIF statement.

If there is one stratum (NSTRATA=1), then the graph consists of one of two conditional titles followed by a conditional second title followed by a graph defined in a layout. With more than one stratum, the graph consists of an unconditional title, a conditional second title, and a graph defined in a different layout. Sometimes the easiest way to understand a template structure is to do precisely what is done here: copy the template and remove details until you are left with an outline of the overall structure. Then use that knowledge to go back and evaluate and modify the full template.

The following provides a similar manual re-edit and pruning, this time concentrating on the titles:

```
define statgraph Stat.Lifetest.Graphics.ProductLimitSurvival;
   dynamic . . .;
   BeginGraph;
      if (NSTRATA=1)
         if (EXISTS(STRATUMID))
            entrytitle "Product-Limit Survival Estimate" " for " STRATUMID;
         else
            entrytitle "Product-Limit Survival Estimate";
         endif;
         if (PLOTATRISK)
            entrytitle "with Number of Subjects at Risk" / textattrs=GRAPHVALUETEXT;
         endif;
       . . .
      else
         entrytitle "Product-Limit Survival Estimates";
         if (EXISTS(SECONDTITLE))
            entrytitle SECONDTITLE / textattrs=GRAPHVALUETEXT;
         endif;
      . . .
      endif;
   EndGraph;
end;
```

You can see that titles can come from literal strings, dynamic variables, or both. If you are unclear about which title appears in the output, you can temporarily change the titles by adding some text to clearly show which is which. The following statements show how:

```
entrytitle "(1) Product-Limit Survival Estimate" " for " STRATUMID;
entrytitle "(2) Product-Limit Survival Estimate";
entrytitle "(3) with Number of Subjects at Risk" / textattrs=GRAPHVALUETEXT;
entrytitle "(4) Product-Limit Survival Estimates";
entrytitle "(5)" SECONDTITLE / textattrs=GRAPHVALUETEXT;
```

If you submit the full template with titles like these, you can clearly see whether a title is used and where it is used. You can apply the same technique to axis labels, legend labels, and any other text in the template. Then you can remove the identification numbers, modify the text of interest, and submit the modified template. For example, you might wish to change the first, second, and fourth title to “Kaplan-Meier Plot”. Note that it is not always sufficient to find and change the first entry title in a template.

In a previous example, an ENTRY statement specified the style element `GraphLabelText` so that the entry text would look like a title. In this template, an ENTRYTITLE statement specifies the style element `GraphValueText` so that second titles are subordinate (less bold or smaller according to the style) to the first title lines.

IF, ELSE, and ENDIF statements cannot be used in arbitrary ways. The GTL code that is conditional must be complete. For example, the following statements produce an error:

```
if ( exists(SQUAREPLOT) )                       /* Wrong! */
    layout overlayequated / equatetype=square; /* Wrong! */
 else                                          /* Wrong! */
    layout overlay;                            /* Wrong! */
 endif;                                        /* Wrong! */
    scatterplot x=XVAR y=YVAR;                 /* Wrong! */
 endlayout;                                    /* Wrong! */
```

The following statements are correct:

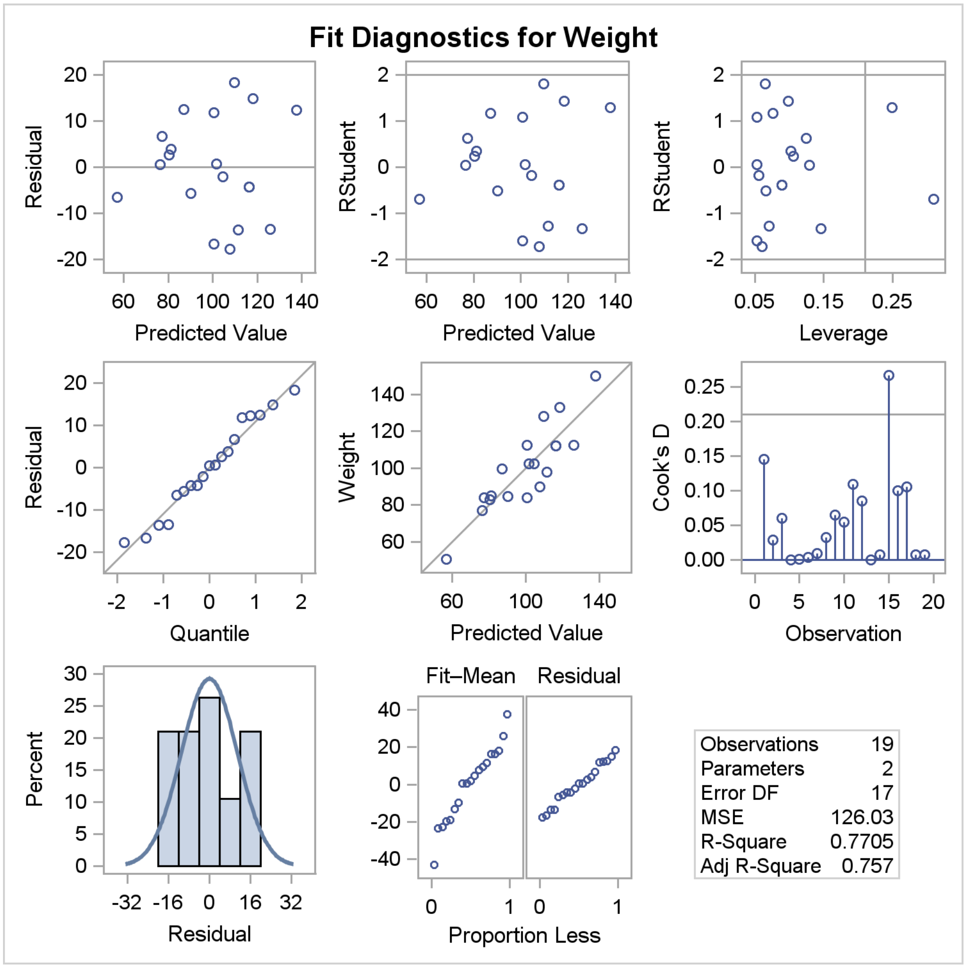
```
if (exists(SQUAREPLOT))
   layout overlayequated / equatetype=square;
      scatterplot x=XVAR y=YVAR;
   endlayout;
else
   layout overlay;
      scatterplot x=XVAR y=YVAR;
   endlayout;
endif;
```

The incorrect example attempts to conditionally execute a complete statement, but only complete layouts (not merely complete layout statements) can be conditionally executed. Also note that IF conditions determine what is rendered in the plot rather than what is computed for the data object. For example, the following step attempts to compute a LOESS fit whether or not the LOESSPLOT dynamic variable is defined:

```
if (exists(LOESSPLOT))
   loessplot y=LOESS x=X;
endif;
```

Since the LOESS fit is computationally expensive, procedure writers use a different approach to conditionally compute results only when needed. If either LOESS or X is a dynamic variable that is not defined to a data object column name, then the computation is not performed.

Figure 4.18 Diagnostics Panel for the REG Procedure



```
Observations      19
Parameters         2
Error DF          17
MSE           126.03
R-Square      0.7705
Adj R-Square   0.757
```

4.8 Paneled Displays

This example discusses how to understand the overall structure of a paneled display with many graphs and how to isolate individual graphs that you might want to modify. The following step fits a regression model and displays a set of model fit diagnostics:

```
ods graphics on;

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

The diagnostics panel is displayed in Figure 4.18.

The rendered version of the diagnostics panel template, `Stat.Reg.Graphics.DiagnosticsPanel`, has 271 lines. A very small portion of it is as follows:

```
define statgraph Stat.Reg.Graphics.DiagnosticsPanel;
   notes "Diagnostics Panel";
   dynamic . . .;
   BeginGraph / designheight=defaultDesignWidth;
      entrytitle . . .;
      layout lattice / columns=3 rowgutter=10 columngutter=10
         shrinkfonts=true rows=3;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlayequated . . . scatterplot . . . endlayout;
         layout overlay . . . needleplot . . . endlayout;
         layout overlay . . . histogram . . . densityplot . . . endlayout;
         layout lattice / columns=2 rows=1 rowdatarange=unionall columngutter=0;
            . . .
            layout overlay . . . scatterplot . . . endlayout;
            layout overlay . . . scatterplot . . . endlayout;
            . . .
         endlayout;
         . . .
         layout overlay;
            layout gridded / columns=_NSTATSCOLS valign=center border=TRUE
               BackgroundColor=GraphWalls:Color Opaque=true;
               . . . entry halign=left "Observations" / valign=top;
               . . . entry halign=right eval (PUT(_NOBS,BEST6.)) / valign=top;
               . . .
            endLayout;
         endif;
      . . .
      endlayout;
   EndGraph;
end;
```

The paneled display is large and square (although greatly reduced from the default size for this book) and is designed with a height equal to the default width. It has a single overall title for the display. It consists of a 3 by 3 lattice of nine entries. The first eight panels are graphs, and the last is a table of statistics. The graphs that are defined in the overlay layouts fill in the display in order from left to right and from top to bottom. The first four graphs are ordinary scatter plots; the fifth is an equated scatter plot where both axes are equated to represent the same data range; the sixth is a needle plot; the seventh is an overlay of a histogram and a density plot; the eighth is another lattice, this one consisting of two scatter plots; and the ninth and final panel in the outer lattice is a grid that contains statistic names and their values. The outermost lattice specifies the SHRINKFONTS=TRUE option. This option is commonly specified in outer lattices and specifies that fonts can be scaled down when the graph is reduced in size. Without this option, the text is typically too large in reduced versions of displays such as Figure 4.18.

Even when the overall template is huge, you can often find and isolate small template components that are easily understood. For example, the first plot, which displays residuals and predicted values, is created from the following layout:

```
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;
```

The DATALABEL= option provides labels for the markers when the dynamic variable _OutLevLabel exists. The ROLENAME= and TIP= options create tooltips in HTML. Tooltips are text boxes that appear in HTML output when your mouse pointer hovers over a part of the plot. Tips are produced for the Y axis column, the X axis column, and additional columns _tip1 and _id1 through _id5. The columns x and y have predefined roles as axis variables. In contrast, the other tips are provided for columns that are identified through the ROLENAME= option. You must provide role names for columns that do not have automatic roles (such as the axis columns) and use the role names rather than the column names in the TIP= option. You can modify the tooltips by adding, deleting, or changing columns specified in these lists. These options usually are specified in templates for graphs that display data or computed values with a one-to-one correspondence with the data (for example, independent variable, dependent variable, predicted values, residuals, leverage, and variables specified in the procedure's ID statement).

Part of the gridded layout that composes the ninth panel is as follows (after some manual indentation adjustments):

```
if (_SHOWNOBS^=0)
   entry halign=left "Observations" / valign=top;
   entry halign=right eval (PUT(_NOBS,BEST6.)) / valign=top;
endif;
if (_SHOWTOTFREQ^=0)
   entry halign=left "Total Frequency" / valign=top;
   entry halign=right eval (PUT(_TOTFREQ,BEST6.)) / valign=top;
endif;
if (_SHOWNPARM^=0)
   entry halign=left "Parameters" / valign=top;
   entry halign=right eval (PUT(_NPARM,BEST6.)) / valign=top;
endif;
```

Do not rely on the indentation provided by PROC TEMPLATE and the SOURCE statement to see the structure of a template. Re-indent the template yourself to make it clearer. Each statistic is added to the display conditional on a dynamic variable. First, a label is displayed on the left followed by a value on the right. In a table such as this, you could change the labels, change the formats, remove statistics, or reorder them.

The layout for the fourth graph, the normal quantile plot of the residuals, which is displayed in the second row and first column of the panel, is as follows:

```
layout overlay / yaxisopts=(label="Residual" shortlabel="Resid")
   xaxisopts=(label="Quantile");
   lineparm slope=eval (STDDEV(RESIDUAL)) y=eval (MEAN(RESIDUAL)) x=0
      / 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;
```

Again, this code has been manually reformatted. The PROC TEMPLATE SOURCE statement struggles with complicated code like this. Besides having indentation problems, PROC TEMPLATE sometimes breaks lines in the middle of names. These problems must be fixed manually before the generated code can be compiled again by PROC TEMPLATE.

This template differs from others shown previously due to the heavy reliance on expression evaluation. The GTL provides a series of functions that can be used to make plots. The LINEPARM statement produces a diagonal reference line whose slope is the standard deviation of the residuals. A line is determined, given a slope and a point. The Y= option provides the Y coordinate of a point, which is the mean of the residuals. The X= option provides the X coordinate of that same point, which is 0. When X=0, then Y= provides the intercept. The scatter plot consists of the sorted residuals (ignoring missing values) on the Y axis and normal quantiles on the X axis. These quantities are also provided as tooltips. Functions and expressions must always be wrapped in the EVAL function.

Figure 4.19 First BY Group

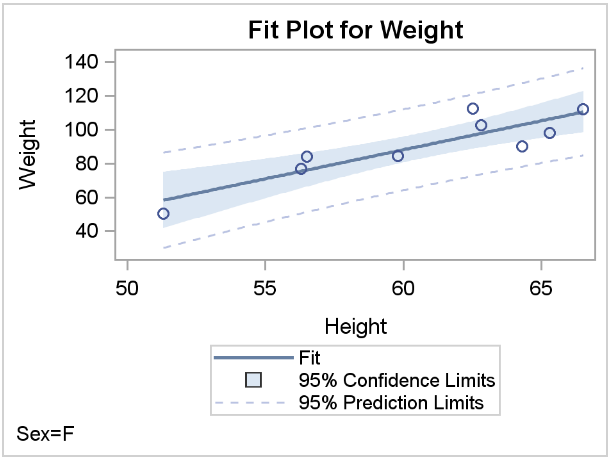
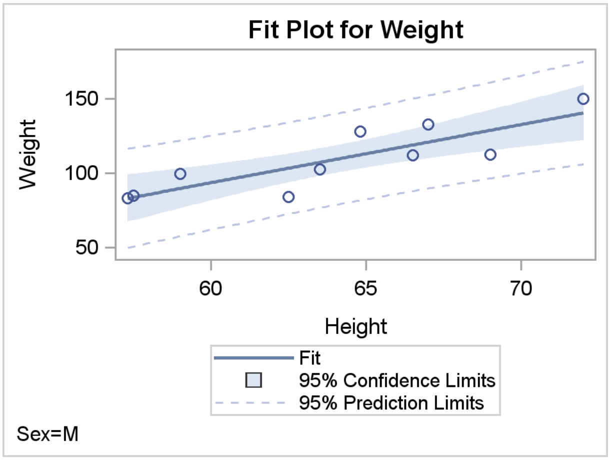


Figure 4.20 Second BY Group



4.9 BY Groups

When a SAS procedure is run with one or more BY variables, SAS provides BY group information at the top of each page of output (for example, "Sex=F" and "Sex=M"). This information does not appear in each table and each graph; it just appears on each page. If you need BY-group information on each graph, you will need to modify the template. For example, you can add the following statements:

```
mvar byline;
entryfootnote halign=left textattrs=graphvaluetext byline;
```

When you add these statements to a template, the value of the macro variable ByLine is displayed as a left aligned footnote. The size, font, and appearance of the footnote comes from the `GraphValueText` style element, which is used for tick and legend labels. Since a macro variable is used, you can modify the template once and reuse it for each BY group. To use the template, you must identify each BY group, extract the BY variable names and values, store them in the macro variable, and run the procedure with just the relevant BY group. You could automate this with a macro and a macro DO loop. The `ModTmplt` macro does all of this for you.

The following steps run PROC GLM with a BY statement and produce separate fit plots for each group:

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

proc glm data=class;
   model weight = height;
   by sex;
   ods select fitplot;
run;
```

The graphical results of this step are not shown, but the trace output for the first BY group is as follows:

```
Output Added:
-------------
Name:       FitPlot
Label:      Fit Plot
Template:   Stat.GLM.Graphics.Fit
Path:       GLM.ByGroup1.ANOVA.Weight.FitPlot
-------------
```

The following step displays the template for the fit plot:

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

The results are as follows:

```
link Stat.GLM.Graphics.Fit to Common.Zreg.Graphics.Fit;
```

Some templates are simply links to other templates. The GLM template is a link to a common template that is shared by other procedures. You have two choices. You can copy the common template, change the name to match the GLM template, and then modify it. You can instead modify the common template. This example uses the former approach. The following step displays the common template:

```
proc template;
   source Common.Zreg.Graphics.Fit;
run;
```

A portion of the results are as follows:

```
define statgraph Common.Zreg.Graphics.Fit;
   dynamic _PREDLABEL _CONFLABEL _DEPLABEL _DEPNAME _INDLABEL
      _SHORTINDLABEL _OBSNUM _Y _XVAR _UCL _LCL _UCLM _LCLM _TITLE
      _Y_OBS _XVAR_OBS _PREDICTED _UCL_OBS _LCL_OBS _UCLM_OBS
      _LCLM_OBS _FREQ _WEIGHT _ID1 _ID2 _ID3 _ID4 _ID5;
   BeginGraph;
      entrytitle _TITLE " for " _DEPNAME;
      . . .
   EndGraph;
end;
```

You can create the GLM template as follows:

```
proc template;
   define statgraph Stat.GLM.Graphics.Fit;
      dynamic _PREDLABEL _CONFLABEL _DEPLABEL _DEPNAME _INDLABEL
         _SHORTINDLABEL _OBSNUM _Y _XVAR _UCL _LCL _UCLM _LCLM _TITLE
         _Y_OBS _XVAR_OBS _PREDICTED _UCL_OBS _LCL_OBS _UCLM_OBS
         _LCLM_OBS _FREQ _WEIGHT _ID1 _ID2 _ID3 _ID4 _ID5;
      BeginGraph;
         entrytitle _TITLE " for " _DEPNAME;
         . . .
      EndGraph;
   end;
run;
```

The following steps add an MVAR and ENTRYFOOTNOTE statement to the template and run PROC GLM twice, once for each BY group:

```
%macro mygraph;
   proc glm data=__bydata;
      model weight = height;
      ods select fitplot;
%mend;

%modtmplt(by=sex, data=class, template=Stat.GLM.Graphics.Fit)
```

The results are displayed in Figure 4.19 and Figure 4.20. The **ModTmplt** macro requires you to provide a SAS macro called **MyGraph** that contains the SAS procedure that needs to be run. Notice that the BY and RUN statements are *not* specified in the **MyGraph** macro. The **ModTmplt** macro constructs and provides them. Also notice that you must use the DATA=__BYDATA option (the data set name begins with two underscores) with the procedure call in the **MyGraph** macro and specify the real input data set in the DATA= option of the **ModTmplt** macro. The **ModTmplt** macro constructs the __BYDATA data set with information from one BY group at a time. If you have more than one BY variable, specify the additional BY variables in the BY= option. You do not need to change anything if you have more BY groups.

You could display the BY group information in a title instead of a footnote as follows:

```
%modtmplt(by=sex, data=class, statement=entrytitle,
          template=Stat.GLM.Graphics.Fit)
```

The results of this step are not displayed. This step adds a second title line with the BY information. Other macro options are available. See Kuhfeld (2009) for more information and options. The macro displays the BY line in a footnote rather than in a title, because most graphs already have titles and few have footnotes. The BY line is less obtrusive in a footnote than it is in a title.

4.10 References

Kuhfeld, W. F. (2009), "Modifying ODS Statistical Graphics Templates in SAS 9.2," http://support.sas.com/rnd/app/papers/modtmplt.pdf.