SESUG Paper 227-2022

Custom Panel Graphs Using PROC TEMPLATE

James Blum, University of North Carolina Wilmington

Abstract

PROC SGPANEL gives a variety of styles of panel graphs, but they have significant limitations in the scope of layouts available. Layouts have equal cell sizes, structured in some form of a grid. They also use the same plotting statement(s) for each of the cells, distinguished only by the variables used in the PANELBY statement. PROC TEMPLATE can construct a wide variety of panel graphs, including those with graphs of different sizes, types, and variable sets, This paper gives an overview of the PROC TEMPLATE tools that mimic common PROC SGPLOT figures and the PROC SGPANEL structures, and then focuses on structures that cannot be built in PROC SGPANEL. Example output and code for graphs built from common SASHELP data sets are provided.

Introduction

PROC SGPLOT and SGPANEL (and others) are designed to make creation of basic charts and graphs fairly straightforward to implement. Plus, they allow some fairly complex graphics to be readily attainable as you learn more about plotting statements and options. However, unlocking the full potential of ODS Graphics necessitates learning about the Graph Template Language (GTL) available through PROC TEMPLATE, and implemented with PROC SGRENDER. The first section opens by creating graphs in SGPLOT and matches them via TEMPLATE/SGRENDER. Later, similar examples are constructed with SGPANEL. In each of these sections, similarities and differences in the syntax for common requests are highlighted. Finally, examples expand into graphics that cannot be constructed directly using SGPLOT or SGPANEL.

Single Cell Graphs

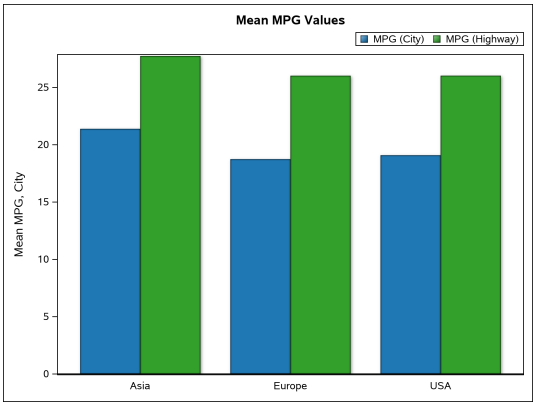
Bar Charts

As a first example of the use of PROC TEMPLATE and PROC SGRENDER, the basic structure of the bar chart created in Program 1 via PROC SGPLOT is rebuilt in Program 2 with PROC TEMPLATE and PROC SGRENDER. Hopefully, all of the statements and options used in the SGPLOT call are familiar to you.

Program 1: A Vertical Bar Chart Overlay with Many Options Applied

```
title 'Mean MPG Values';
proc sgplot data=sashelp.cars;
 styleattrs datacolors=(cx1f78b4 cx33a02c);
 vbar origin / response=mpg_city stat=mean discreteoffset=-0.2
               barwidth=0.4 dataskin=matte;
 vbar origin / response=mpg_highway stat=mean discreteoffset=0.2
               barwidth=0.4 dataskin=matte;
 where type ne 'Hybrid';
 xaxis display=(nolabel);
 yaxis label='Mean MPG, City';
 keylegend / position=topright;
run;
```

Output 1: A Vertical Bar Chart Overlay with Many Options Applied



The most obvious difference between the use of PROC SGPLOT and PROC TEMPLATE and SGRENDER in Program 2 is the need for two procedure calls, but several other components of the syntax can look quite different even if they are used to perform the same action.

Program 2: A First Look at TEMPLATE and SGRENDER

```
proc template;
  define statgraph MultBar1;❶
    begingraph / datacolors=(cx1f78b4 cx33a02c);❷
      entrytitle 'Mean MPG Values';❸
      layout overlay ❹ / cycleattrs=true❷
            xaxisopts=(display=(tickvalues))
            yaxisopts=(label='MPG');❺
        barchart x=origin y=mpg_city / stat=mean dataskin=matte
                  discreteoffset=-0.2 barwidth=0.4 name='City';❻
        barchart x=origin y=mpg_highway / stat=mean dataskin=matte
                  discreteoffset=0.2  barwidth=0.4 name='Highway';
        discretelegend 'City' 'Highway' / halign=right valign=top;❼
      endlayout;❹
    endgraph;❷
  end;❶
run;

proc sgrender data=sashelp.cars template=MultBar1;
  where type ne 'Hybrid';
run;❽
```

❶ The DEFINE STATGRAPH statement names the template, which is stored in the Work library by default. The STORE= option is available to choose a specified template store and library combination. This statement defines a block of code, and is closed with an END statement.

❷ The BEGINGRAPH statement, which is closed by the ENDGRAPH statement, is a block of code that contains the graph definitions. A variety of options are permitted, including those given in the STYLEATTRS statement in SGPLOT or SGPANEL, like DATACOLORS=. However, for this liste to be applied, the CYCLEATTRS=TRUE specification must be included as an option in whichever LAYOUT statement is in use. The ATTRPRIORITY option, usually set in the ODS GRAPHICS statement, can be set here also. If you have ever wondered why marker attributes can cycle differently in the HTML destination versus the listing destination, this is why–those options can be set differently in different templates.

❸ Before the graph layout is chosen, the ENTRYTITLE statement is used to put a title in the image (ENTRYFOOTNOTE is also available). Attempting to use a TITLE statement prior to invoking PROC SGRENDER does not have the same effect.

❹ Graphs are built in layouts, and OVERLAY is the most common single-cell layout (you can see

the rest in the SAS Documentation). More than one layout can be used in a graph definition, and each is a block that closes with the ENDLAYOUT statement.

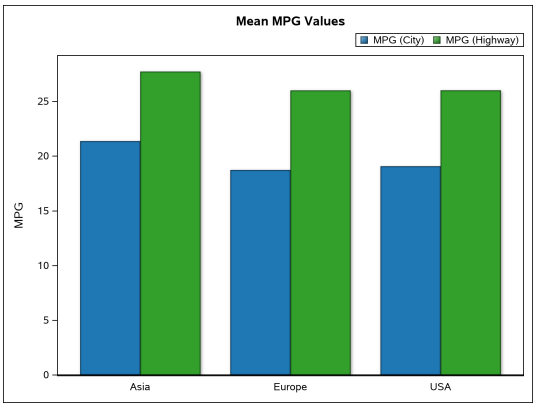
❺ There are no axis statements available here, instead axis options are set as options in the LAYOUT statement. Some of these behave quite differently; for example, DISPLAY= lists what to display, rather than what not to display. So to suppress the tick marks for the category (x) axis, the instruction is to display the tick values, implying other elements are not displayed.

❻ The BARCHART plotting statements used here use options identical to their use in HBAR or VBAR statements previously (default orientation in vertical). However, RESPONSE= is not an option here, instead the Y= variable is used, and the X= variable is the charting variable. It is also possible to give CATEGORY= and RESPONSE= as the charting and response variable set (try it).

❼ A legend is not automatic for an overlay here, and must be requested in the DISCRETELEGEND statement, naming all plotting statements to be included. The positioning of the legend is done with the HALIGN= and VALIGN= options, rather than the single POSITION= option.

❽ PROC SGRENDER then works with the data set and the template to render the graph.

Output 2: A First Look at TEMPLATE and SGRENDER



To make a horizontal bar chart, the plotting statements in Program 1 can be changed to HBAR, and the roles of the XAXIS and YAXIS statements reversed. Program 3 reverses the role of the X= and Y= variables from Program 2 in an attempt to make the bars horizontal.

Program 3: Creating a Horizontal Bar Chart, First Attempt

```
proc template;
  define statgraph MultBar2;
    begingraph / datacolors=(cx1f78b4 cx33a02c);
      entrytitle 'Mean MPG Values';
      layout overlay / cycleattrs=true
            xaxisopts=(display=(tickvalues))
            yaxisopts=(label='MPG' offsetmax=0.05);
        barchart y=origin x=mpg_city / stat=mean dataskin=matte
                         discreteoffset=-0.2 barwidth=0.4 name='City';
        barchart y=origin x=mpg_highway / stat=mean dataskin=matte
                         discreteoffset=0.2 barwidth=0.4 name='Highway';
        discretelegend 'City' 'Highway' / halign=right valign=top;
      endlayout;
    endgraph;
  end;
run;
proc sgrender data=sashelp.cars template=MultBar2;
  where type ne 'Hybrid';
run;
```

No graph is rendered in this case, and warnings are produced in the log (WARNING: Y=ORIGIN is invalid. The column has the wrong data type.). The role of the variable X is always taken as the charting variable, and Y is the response, with the orientation set via a separate option. Because of this, it is a good programming practice (GPP) to use CATEGORY= and RESPONSE= instead of X= and Y= to declare variables in the BARCHART statement. Program 4 uses this GPP and the proper option to change the orientation.

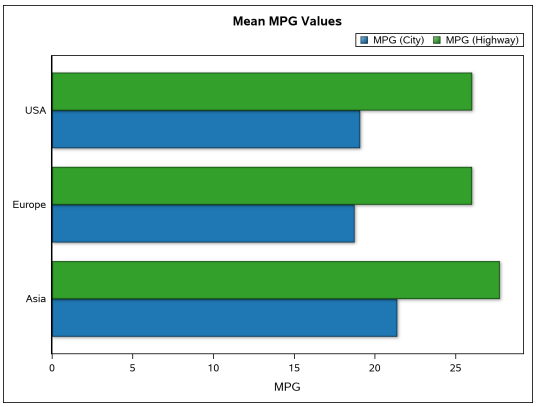
Program 4: Correctly Creating a Horizontal Bar Chart

```
proc template;
  define statgraph MultBar3;
    begingraph / datacolors=(cx1f78b4 cx33a02c);
      entrytitle 'Mean MPG Values';
      layout overlay / cycleattrs=true
            xaxisopts=(label='MPG' offsetmax=0.05)
            yaxisopts=(display=(tickvalues));❶
        barchart category=origin response=mpg_city / stat=mean
                        dataskin=matte discreteoffset=-0.2 barwidth=0.4
                        name='City' orient=horizontal❷;
        barchart category=origin response=mpg_highway / stat=mean
                        dataskin=matte discreteoffset=0.2 barwidth=0.4
                        name='Highway' orient=horizontal❷;
        discretelegend 'City' 'Highway' / halign=right valign=top;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=MultBar3;
  where type ne 'Hybrid';
run;
```

❶ Since the orientation horizontal, the roles of the X and Y axis are changed. This can be exceptionally confusing if X= and Y= are used–The YAXISOPTS would actually apply to the X= variable, and XAXISOPTS to Y=, for a horizontal chart. Yet another reason why CATEGORY= and RESPONSE= are preferred.

❷ The ORIENT=HORIZONTAL option changes the orientation, and must be in both charting statements when these are overlayed.

Output 4: Correctly Creating a Horizontal Bar Chart



Making Templates Dynamic

Templates have difficult syntax, and thus become far less useful if variable names are hard-coded into them. There are two ways to make these choices dynamic (changeable in various calls to PROC SGRENDER), using macro variables or dynamic variables. Program 5 defines some parameters as macro variables, and then calls PROC SGRENDER in a macro definition.

Program 5: Using Macro Variables in TEMPLATE

```
proc template;
  define statgraph MacroBar;
    mvar _TitleText_ _ResponseLabel_ _category_ _response1_ _response2_
         _stat_;❶
    nmvar _OffsetMax_;❷
    begingraph / datacolors=(cx1f78b4 cx33a02c);
      entrytitle _TitleText_;
      layout overlay / cycleattrs=true
            yaxisopts=(display=(tickvalues))
            xaxisopts=(label=_ResponseLabel_ offsetmax=_OffsetMax_);
        barchart category=_category_ response=_response1_ ❸/
                 stat=_stat_ dataskin=matte discreteoffset=-0.2
                 barwidth=0.4 name='A'❹ orient=horizontal;
        barchart category=_category_ response=_response2_ ❸/
                 stat=_stat_ dataskin=matte discreteoffset=0.2
                 barwidth=0.4 name='B'❹ orient=horizontal;
        discretelegend 'A' 'B' ❹/ halign=right valign=top;
      endlayout;
    endgraph;
  end;
run;

%macro SideBySide(_TitleText_=,_ResponseLabel_=,_category_=,
                  _response1_=,_response2_=,_stat_=mean,
                  _OffsetMax_=0.05,lib=sashelp,dset=cars);❺
proc sgrender data=&lib..&dset template=MacroBar;
run;
%mend;
%SideBySide(_category_=origin,_response1_=mpg_city,
            _response2_=mpg_highway);
%SideBySide(_category_=weight_status,_response1_=systolic,
        _response2_=diastolic,dset=heart,
        _ResponseLabel_=Mean Blood Pressure);❻
```

❶ The MVAR statements lists macro variables that are used in the template and take on character values. The role is different in TEMPLATE than you may expect–no & is used when they are referenced, and character and numeric-valued variables are separately defined. Note, once PROC TEMPLATE executes, the template is created, subsequent uses of that template do not recompile/re-execute the TEMPLATE code, so the usual scanning rules for & and % are not going to occur.

❷ One macro variable is set up to contain numeric values; hence, it must be placed in an NMVAR statement.

❸ The two BARCHART calls use the same macro variable for the CATEGORY= value, but different ones for the REPSONSE=value, as expected in this type of overlay.

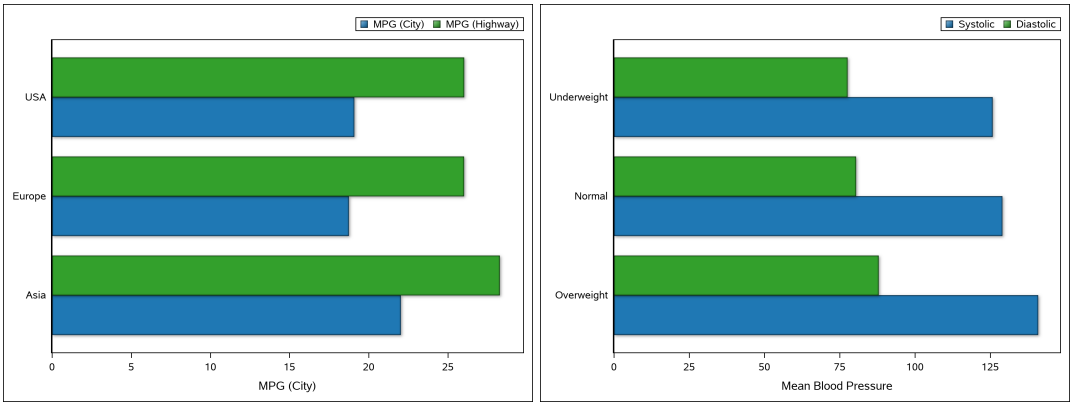
❹ Generic names are used for the legend definition–they do not appear in the output, so they suit the more generic template well.

❺ A call to PROC SGRENDER is defined in a macro that uses a keyword list to set parameters. Any form of macro definition can be used; however, the keyword list does have the advantage of

permitting default values.

❻ Here, two calls to the macro are made under different conditions.

Output 5: Using Macro Variables in TEMPLATE



The use of an underscore prefix and suffix for the macro variable names is a stylistic choice to make it easier to distinguish between those names and other names or keywords. This technique is also used in Program 6. This program alters the approach of Program 5, defining the parameters as dynamic variables, using the DYNAMIC statement in PROC SGRENDER.

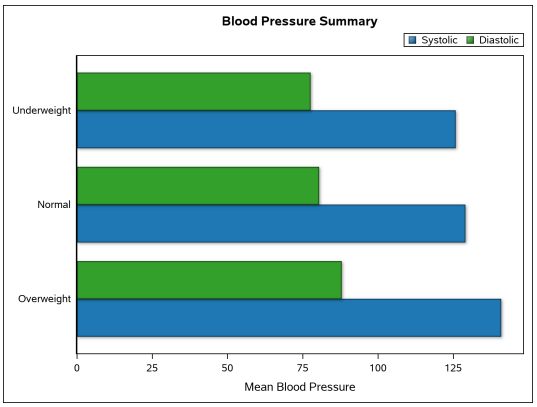
Program 6: Using Dynamic Variables in TEMPLATE and SGRENDER

```
proc template;
  define statgraph DynamicBar;
    dynamic _TitleText_ _ResponseLabel_ _category_
            _response1_ _response2_ _stat_ _OffsetMax_;❶
    begingraph / datacolors=(cx1f78b4 cx33a02c);
      entrytitle _TitleText_;
      layout overlay / cycleattrs=true
            yaxisopts=(display=(tickvalues))
            xaxisopts=(label=_ResponseLabel_ offsetmax=_OffsetMax_);
        barchart category=_category_ response=_response1_ /
                 stat=_stat_ dataskin=matte discreteoffset=-0.2
                 barwidth=0.4 name='A' orient=horizontal;
        barchart category=_category_ response=_response2_ /
                 stat=_stat_ dataskin=matte discreteoffset=0.2
                 barwidth=0.4 name='B' orient=horizontal;
        discretelegend 'A' 'B' / halign=right valign=top;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=DynamicBar;
  dynamic❷ _category_='weight_status' _response1_='systolic'
           _response2_='diastolic' _ResponseLabel_='Mean Blood Pressure'
           _stat_='mean' _OffsetMax_=0.05
           _TitleText_='Blood Pressure Summary';
run;
```

❶ The DYNAMIC statement in PROC TEMPLATE lists the dynamic variable references that occur in the template. No separation between numeric and character variables is made.

❷ The DYNAMIC statement in PROC SGRENDER sets the values for dynamic variables for the template in use. Character values are sent as quoted literals.

Output 6: Using Dynamic Variables in TEMPLATE and SGRENDER



The choice between using macro variables or dynamic variables probably comes down to certain personal or organizational preferences. If graphing tools are being built on these principles, it helps to take certain precautions with parameters, and provide information when parameters are not specified correctly. Consider the following modification of the macro given in Program 5, which puts information into the log any of the required variables are not present.

Program 7: Checking Macro Parameters

```
%macro SideBySide(_TitleText_=,_ResponseLabel_=,_category_=,_response1_=,
                  _response2_=,_stat_=mean,_OffsetMax_=0.05,lib=sashelp,
                  dset=cars);
 %if(&_category_ eq  or &_response1_ eq  or &_response2_ eq ) %then %do;
  %put WARNING: Some charting variables are not specified.;
  %put A value must be specified for _category_,
        which is the charting variable.;
  %put Values for _response1_ and _response2_ must be specified.;
  %put The default statistic is the mean, and can be changed via _stat_.;
  %put Optionally, _TitleText_ sets the title,;
  %put .   _ResponseLabel_ sets the label for the response axis,;
  %put .   and _OffsetMax_ sets the offset from the end of the axis
  (5% is the default).;
  %end;
  %else %do;
   proc sgrender data=&lib..&dset template=MacroBar;
   run;
  %end;
%mend;
```

With this modification, a failure to specify all required parameters (or an intentional call without them) sends information to the log about the requirements and options. Unfortunately, this information is only indirectly tied to the template through the macro, rather that being stored in and generated by the template. It is possible to use conditional logic on dynamic variables in the template; unfortunately, there is no provision to writing information to the log from the template. If you have done other work in PROC TEMPLATE, you may have seen that the PUTLOG statement is available in some contexts, but it is not available in GTL. It is possible to store information like this as part of the template using the NOTES statement. Those statements become part of the compiled template and are viewable using SOURCE–comments in PROC TEMPLATE code do not become part of the source. Documentation and robustness are important when developing these tools, but some compromises may be necessary.

When conditioning on parameter values is needed, special conditional logic structures are available in the template definition. Program 8 draws the previous bar graphs differently depending on whether or not a second response variable is chosen.

Program 8: Conditioning on Dynamic Variables in TEMPLATE

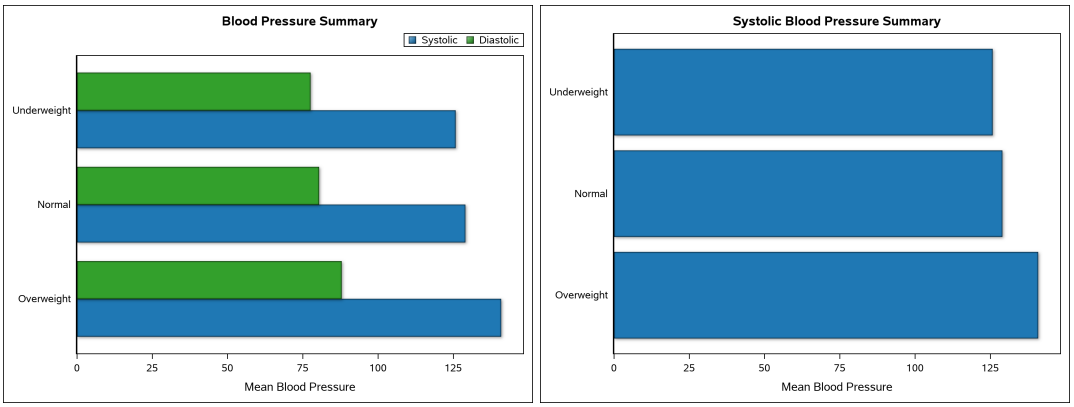
```
proc template;
  define statgraph DynamicBar2;
   dynamic _TitleText_ _ResponseLabel_ _category_ _response1_ _response2_
           _stat_ _OffsetMax_;
    begingraph / datacolors=(cx1f78b4 cx33a02c);
      entrytitle _TitleText_;
      layout overlay / cycleattrs=true
           yaxisopts=(display=(tickvalues))
           xaxisopts=(label=_ResponseLabel_ offsetmax=_OffsetMax_);
        if (exists(_response2_))❶
         barchart category=_category_ response=_response1_ /
                  stat=_stat_ dataskin=matte discreteoffset=-0.2
                  barwidth=0.4 name='A' orient=horizontal;
         barchart category=_category_ response=_response2_ /
                  stat=_stat_ dataskin=matte discreteoffset=0.2
                  barwidth=0.4 name='B' orient=horizontal;
         discretelegend 'A' 'B' / halign=right valign=top;
        else❷
         barchart category=_category_ response=_response1_ /
                  stat=_stat_ dataskin=matte orient=horizontal;
        endif;❸
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=DynamicBar2;
  dynamic _category_='weight_status' _response1_='systolic'
   _response2_='diastolic' _ResponseLabel_='Mean Blood Pressure'
   _stat_='mean' _OffsetMax_=0.05 _TitleText_='Blood Pressure Summary';
run;

proc sgrender data=sashelp.heart template=DynamicBar2;
  dynamic _category_='weight_status' _response1_='systolic'
   _ResponseLabel_='Mean Blood Pressure' _stat_='mean'
   _OffsetMax_=0.05
   _TitleText_='Systolic Blood Pressure Summary';
run;
```

❶ Unlike other SAS syntax, IF is not a statement on its own. The syntax includes IF with a condition, which must be given in parentheses. Here, the condition is on whether or not the dynamic variable _Response2_ has been defined, using the EXISTS function (do not try to condition on a null or missing value, it does not work). When the condition is true, the statements between it and the ELSE are used.

❷ ELSE precedes the set of statements that are invoked if the condition is false, which here amounts to making a single bar chart when only a single response is present.

❸ This chain concludes with the required ENDIF statement.

Output 8: Conditioning on Dynamic Variables in TEMPLATE



Now the template is a bit more dynamic than before, but it does require understanding some unusual syntax for conditioning. Also, if the same template is built with macro variables, the condition is different–the keyword passing ensures that the macro variable always exists, so this time the condition must be on whether or not the value is missing. Of course, a macro could be constructed around SGPLOT code to achieve the same types of graphs, so TEMPLATE/SGRENDER is not required to create this effect.

Program 9 provides the option to orient the graph as vertical or horizontal in the dynamic variable set. The conditioning required here emphasizes some of the peculiarities encountered with the IF-ELSE-ENDIF structures.

Program 9: More Conditioning on Dynamic Variables in TEMPLATE

```
proc template;
  define statgraph DynamicBar3;
   dynamic _TitleText_ _ResponseLabel_ _category_ _response1_ _response2_
           _orient_ _stat_ _OffsetMax_;
    begingraph / datacolors=(cx1f78b4 cx33a02c);
      entrytitle _TitleText_;
      if(upcase(substr(_orient_,1,1)) eq 'H')❶
        layout overlay / cycleattrs=true
             yaxisopts=(display=(tickvalues))
             xaxisopts=(label=_ResponseLabel_ offsetmax=_OffsetMax_);
          if (exists(_response2_))
           barchart category=_category_ response=_response1_ /
                    stat=_stat_ dataskin=matte discreteoffset=-0.2
                    barwidth=0.4 name='A' orient=horizontal;
           barchart category=_category_ response=_response2_ /
                    stat=_stat_ dataskin=matte discreteoffset=0.2
                    barwidth=0.4 name='B' orient=horizontal;
           discretelegend 'A' 'B' / halign=right valign=top;
          else
           barchart category=_category_ response=_response1_ /
                      stat=_stat_ dataskin=matte orient=horizontal;
          endif;
         endlayout;❷
       else❶
        layout overlay / cycleattrs=true
             yaxisopts=(label=_ResponseLabel_ offsetmax=_OffsetMax_)
             xaxisopts=(display=(tickvalues));
          if (exists(_response2_))
           barchart category=_category_ response=_response1_ /
                    stat=_stat_ dataskin=matte discreteoffset=-0.2
                    barwidth=0.4 name='A';
           barchart category=_category_ response=_response2_ /
                    stat=_stat_ dataskin=matte discreteoffset=0.2
                    barwidth=0.4 name='B';
           discretelegend 'A' 'B' / halign=right valign=top;
          else
           barchart category=_category_ response=_response1_ /
                      stat=_stat_ dataskin=matte;
          endif;
         endlayout;❷
       endif;
    endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=DynamicBar3;
  dynamic _category_='weight_status' _response1_='systolic'
   _response2_='diastolic' _ResponseLabel_='Mean Blood Pressure'
   _stat_='mean' _OffsetMax_=0.05 _TitleText_='Blood Pressure Summary';❸
run;

proc sgrender data=sashelp.heart template=DynamicBar3;
  dynamic _category_='weight_status' _response1_='systolic'
   _orient_='horizontal' _ResponseLabel_='Mean Blood Pressure'
   _stat_='mean' _OffsetMax_=0.05 _TitleText_='Blood Pressure Summary';❹
run;
```

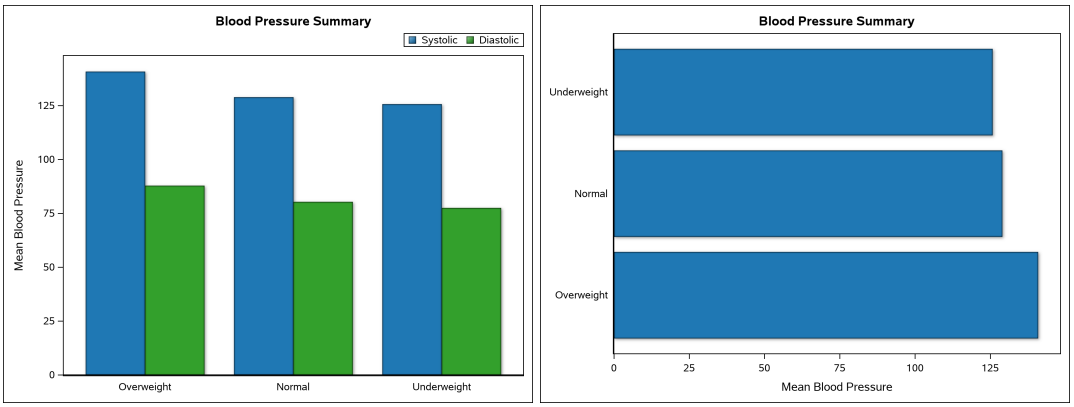
❶ The _Orient_ variable is used to set the orientation via this conditioning. Resist the temptation to think this is like conditioning in the macro language, where conditions can substitute various amounts of text, including portions of statements. Here, the conditional branches must refer to full statements at a minimum. Since the choice of axis options depends on the choice of orientation, the first statement for this conditioning must be the LAYOUT statement.

❷ Each conditional branch includes a full set of statements through the ENDLAYOUT statement. This may be seen as a matter of convenience, as opposed to trying to then also condition on _Orient_ in each BARCHART statement, but it is actually a requirement. In fact, attempting to simplfy the code by removing these two ENDLAYOUT statements and placing a single ENDLAYOUT between ENDIF and ENGRAPH also fails–failing to work at compilation because the compiler sees two LAYOUT statements and only one ENDLAYOUT statement. Essentially, the syntax check does not check the branching in the way you might want/expect, so the scope of the initial statement in the branching determines what set of statements must be included in the branches.

❸ This call to SGRENDER includes two response variables and no reference to the _Orient_ variable. So, since vertical is the default orientation, a side-by-side vertical bar chart is produced.

❹ This does set a value of horizontal for the _Orient_ variable, and only includes one response variable. So, the result is a horizontal bar chart on a single variable.

Output 9: More Conditioning on Dynamic Variables in TEMPLATE



Other Plot Types

Next up, in Program 10, horizontal and vertical boxplots are constructed in PROC SGPLOT with a couple of tricks to show some contrasts with TEMPLATE.

Program 10: Boxplots in SGPLOT

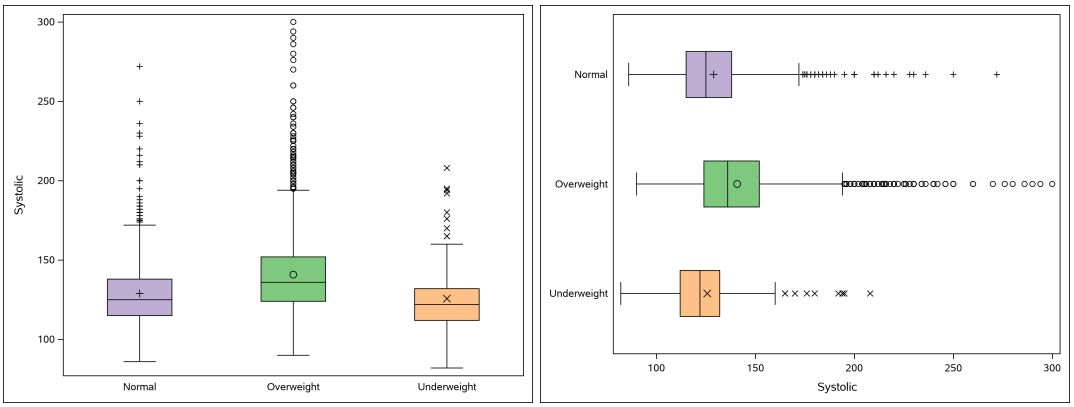
```
proc sgplot data=sashelp.heart noautolegend;
  styleattrs datacontrastcolors=(black)
             datacolors=(cx7fc97f cxbeaed4 cxfdc086);❶
  vbox systolic / group=weight_status category=weight_status;❷
  xaxis display=(noticks nolabel);❸
run;

proc sgplot data=sashelp.heart noautolegend;
  styleattrs datacontrastcolors=(black)
             datacolors=(cx7fc97f cxbeaed4 cxfdc086);
  hbox systolic / group=weight_status category=weight_status;❷
  yaxis display=(noticks nolabel);❸
run;
```

❶ Since this graph will cycle through colors due to the GROUP= option, the fill color set is controlled here. The line and marker colors are fixed to black for each box.

❷ Here, a little trick is employed to get a category axis and get different fill colors for the boxes–the same variable is used for the CATEGORY= and GROUP= option.

❸ The legend is suppressed via the NOAUTOLEGEND option in the PROC SGPLOT statement, and the category axis is styled as desired. For the VBOX, it is the x-axis, for HBOX it is the y-axis.

Output 10: Boxplots in SGPLOT



Program 11 does a similar, but not exactly equivalent set of boxplots via TEMPLATE.

Program 11: Boxplots with TEMPLATE

```
proc template;
  define statgraph VBox;
    dynamic _TitleText_ _ResponseLabel1_ _x_ _y1_;
    begingraph / datacontrastcolors=(black)
                 datacolors=(cx7fc97f cxbeaed4 cxfdc086);❶
      entrytitle _TitleText_;
        layout overlay / yaxisopts=(label=_ResponseLabel1_)
                         xaxisopts=(display=(tickvalues));❷
          boxplot x=_x_ y=_y1_ / group=_x_;❸
        endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=VBox;
  dynamic _x_='weight_status' _y1_='systolic'
  _ResponseLabel1_='Systolic'
  _TitleText_='Systolic Blood Pressure Summary';
run;
```

Program 11 (cont.): Boxplots with TEMPLATE

```
proc template;
  define statgraph HBox;
    dynamic _TitleText_ _ResponseLabel1_ _x_ _y1_;
    begingraph / datacontrastcolors=(black)
                 datacolors=(cx7fc97f cxbeaed4 cxfdc086);❶
      entrytitle _TitleText_;
        layout overlay / xaxisopts=(label=_ResponseLabel1_)
                         yaxisopts=(display=(tickvalues));❹
          boxplot x=_x_ y=_y1_ / group=_x_ orient=horizontal;❹
        endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=HBox;
  dynamic _x_='weight_status' _y1_='systolic'
  _ResponseLabel1_='Systolic'
  _TitleText_='Systolic Blood Pressure Summary';
run;
```

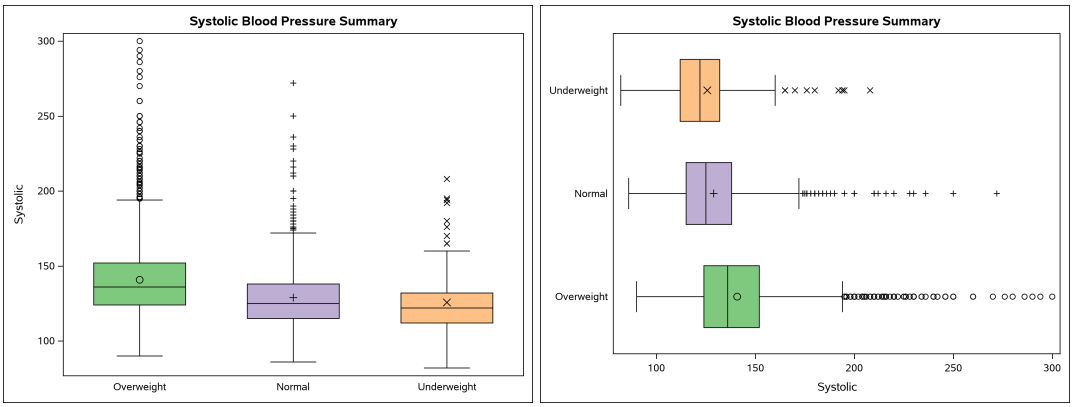
❶ As before, the options typically used in a STYLEATTRS statement are supplied as options in the BEGINGRAPH statement.

❷ For the vertical boxplot, the x-axis is the category axis and the y-axis is the response axis, so options corresponding to those of the VBOX plot in Program 10 are provided here.

❸ The BOXPLOT call looks similar to our first call to BARCHART in Program 2. However, there are no CATEGORY= and RESPONSE= aliases for this call, the category variable is given by X=, and the response as Y=.

❹ Transitioning to a horizontal boxplot then follows the bar chart logic–the ORIENT= option makes the change. Oddly, the y-axis options are now associated with the X= variable, and the x-axis options associated with Y=.

Output 11: Boxplots with TEMPLATE



Note the difference in Output 10 and Output 11. The first color in the DATACOLORS= list is a green hue, and it is applied to the first group in data order–which happens to be Overweight. For the vertical box plots generated by SGPLOT, this is different from the category order, which is the alphabetical order (internal) for those values. In the TEMPLATE results, the category order is also data order. Note further that these differences are also true for HBOX, but there is another difference. The category axis for a horizontal bar chart in SGPLOT starts at the top, in TEMPLATE it starts at the bottom. The upshot of all of this is that not only should you expect syntax differences when using TEMPLATE instead of SGPLOT, but default behaviors can be quite different as well. Also, you may not have the same settings for all options across both platforms.

The next plots are grouped spline plots with markers, again taken up with SGPLOT first, followed by TEMPLATE.

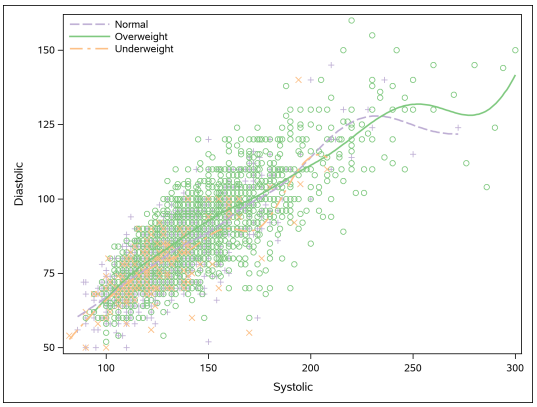
Program 12: Spline Plot with Data Markers in SGPLOT

```
proc sgplot data=sashelp.heart;
  styleattrs datacontrastcolors=(cx7fc97f cxbeaed4 cxfdc086);
  pbspline x=systolic y=diastolic / group=weight_status smooth=15000;❶
  where weight_status ne '';❷
  keylegend  / location=inside position=topleft title=''
               across=1 noborder;
run;
```

❶ By default, a PBSPLINE statement gives both the spline fits and the data markers (NOMARKERS is available to suppress those). Grouping is available, but missing values are taken as a group level, when present.

❷ Of course, the values with a missing group level can always be removed via where subsetting.

Output 12: Spline Plot with Data Markers in SGPLOT



Program 13: Spline Plot with Data Markers in TEMPLATE

```
proc template;
  define statgraph SplinePlot;
    dynamic _TitleText_ _xLabel_ _yLabel_ _g_ _x_ _y_;
    begingraph / datacontrastcolors=(cx7fc97f cxbeaed4 cxfdc086);
      entrytitle _TitleText_;
       layout overlay/ xaxisopts=(label=_xLabel_)
                       y2axisopts=(label=_yLabel_);❶
            scatterplot x=_x_ y=_y_ / group=_g_ name='Scatter'
                                      includemissinggroup=false;❷
            pbsplineplot x=_x_ y=_y_ / group=_g_ smooth=15000
                                      includemissinggroup=false;❷
            discretelegend 'Scatter' ❸ / location=inside halign=left
                                 valign=top title='' across=1 border=false;
      endlayout;
    endgraph;
  end;
run;
```

Program 13 (cont.): Spline Plot with Data Markers in TEMPLATE

```
proc sort data=sashelp.heart out=HeartSort;
  by descending weight_status systolic;❹
run;

proc sgrender data=HeartSort template=SplinePlot;
  dynamic _g_='weight_status' _x_='systolic' _y_='diastolic'
  _xLabel_='Systolic' _yLabel_='Diastolic';
run;
```

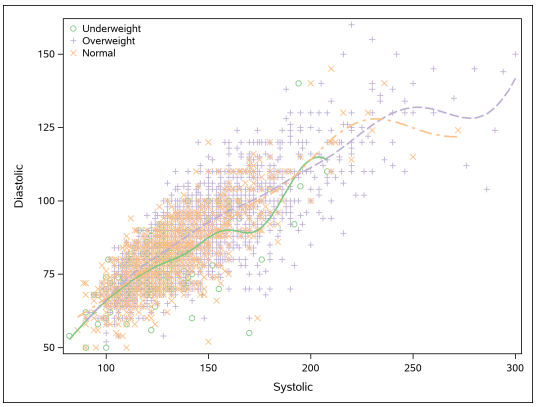
❶ Unlike the SGPLOT version, which only needs a single plotting call, here we require an overlay of a scatter plot and a spline plot to get both the curves and the data markers–PBSPLINEPLOT in template does not permit an option to include the markers.

❷ Both SCATTERPLOT and PBSPLINEPLOT include a means for removing missing values for the group variable. For SCATTERPLOT, it works as you would likely suspect, for PBSPLINEPLOT, it probably does not–more on that in ❹.

❸ Here, only one of the plots is used in creating the legend and, just to illustrate the difference, the legend is made to reference the markers by pointing to the SCATTERPLOT statement.

❹ In order for the grouping to work for PBSPLINEPLOT in TEMPLATE, the data must be sorted on the group variable (and then on the x variable). This would be a good time subset out the missing values and not have to worry about the missing group setting in the template. However, if we are relying on that, having the default group of missing come first can cause problems. A descending sort avoids this issue, but does permute the colors as they are applied in data order. In general, for either the SGPLOT or TEMPLATE approach, it is wisest to subset out missing values on a group variable.

Output 13: Spline Plot with Data Markers in TEMPLATE



Multi-Panel Layouts with Common Panel Types

PROC SGPANEL provides (of course) the ability to create panel graphs. PROC TEMPLATE has several layouts that create panel graphs. This section shows how to mimic the some standard SGPANEL graphs with TEMPLATE, while Section vers the greater flexibility available in PROC TEMPLATE. The SGPANEL examples are offered without specific commentary, if you are unfamiliar with SGPANEL, have a look at some of the examples in the SAS Documentation.

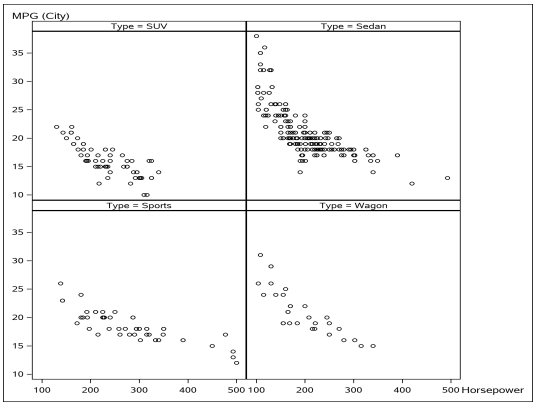
Paneling on a Single Category Variable

Program 14 creates a panel of scatter plots across a single variable, using a common set of x and y variables in each panel.

Program 14: Panel Graph on a Single Paneling Variable

```
proc sgpanel data=sashelp.cars;
  panelby type;
  where type not in ('Hybrid','Truck');
  scatter x=horsepower y=mpg_city;
  colaxis labelpos=right;
  rowaxis labelpos=top;
run;
```

Output 14: Panel Graph on a Single Paneling Variable



An equivalent layout in PROC TEMPLATE to the one generated by Program 14 is the DATAPANEL layout. Program 15 produces the same figure as Output 14 using this.

Program 15: The DATAPANEL Layout in TEMPLATE

```
proc template;
  define statgraph datapanel;
    dynamic _class_ _x_ _y_ _rows_ _cols_;
    begingraph;
      layout datapanel classvars=(_class_) ❶ /
          rows=_rows_ columns=_cols_ ❷
          columnaxisopts=(labelposition=right)
          rowaxisopts=(labelposition=top)❸;
        layout prototype;❹
          scatterplot x=_x_ y=_y_;
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=datapanel;
  dynamic _class_='Type' _x_='horsepower' _y_='mpg_city'
        _rows_=2 _cols_=2;
  where type not in ('Hybrid','Truck');
run;
```

❶ The CLASSVARS= specification is equivalent to the PANELBY statement in SGPANEL. Though this template is written to accept only a single dynamic value, a space delimited list is permissible here.

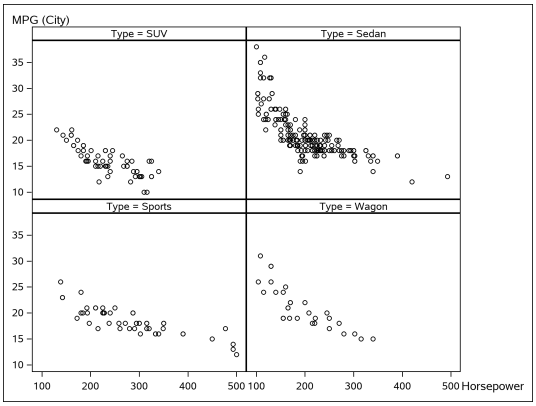
❷ Specifying the rows and columns is optional in the PANELBY statement. Technically, it is optional

for the DATAPANEL layout as well, but the automatic paneling given by DATAPANEL leaves much to be desired. It is generally better to specify ROWS= and/or COLUMNS= specifically. If both are specified, and the total panels allocated is less than the levels given by CLASSVARS=, some panels will not be drawn.

❸ Like SGPANEL, this layout refers to column and row axes in place of x and y axes. As with other layouts, the axis specifications are made as options in the LAYOUT statement.

❹ The PROTOTYPE layout is used within layouts that generate a common graph type across several panels. Only one such layout is used, so if more than one is given, the last instance determines which graph is drawn in the panels.

Output 15: The DATAPANEL Layout in TEMPLATE



If the plotting request in each panel includes grouping or overlaying, a legend is produced, which can be styled with KEYLEGEND statement. In TEMPLATE, it is a multi-layered request to get the legend, which is taken up in the following example, which also considers a two-category row/column layout.

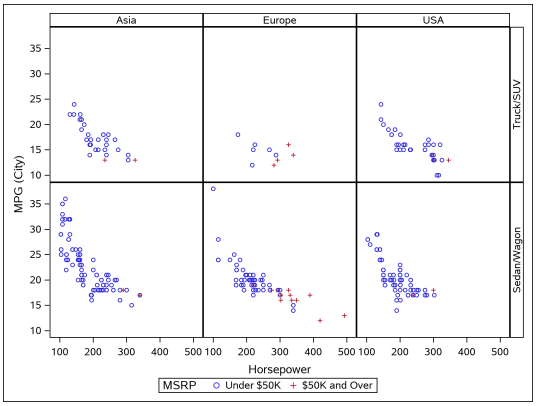
Row and Column Panels Across Two Categories

When using two, and exactly two, variables in the PANELBY statement in SGPANEL, it is possible to request a lattice layout, which puts the first variable on the columns and the second on the rows, as shown in Program 16

Program 16: Row/Column Lattice on Two Variables in SGPANEL

```
proc format;
  value $typeCompress
    'Truck','SUV'='Truck/SUV'
    'Sedan','Wagon'='Sedan/Wagon'
    ;
  value priceCat
    low-<50000='Under $50K'
    50000-high='$50K and Over'
    ;
run;

proc sgpanel data=sashelp.cars;
  panelby origin type/ layout=lattice novarname ;
  format type $typeCompress. MSRP priceCat.;
  where type not in ('Hybrid','Sports');
  scatter x=horsepower y=mpg_city / group=MSRP;
run;
```

Output 16: Row/Column Lattice on Two Variables in SGPANEL



A TEMPLATE layout that directly mimics Output 16 is the DATALATTICE layout. There is also a LATTICE layout, but it is of a more general form and is taken up in Section rogram es the DATALATTICE layout.

Program 17: Row/Column Lattice on Two Variables in TEMPLATE

```
proc template;
  define statgraph datalattice;
    dynamic _RowClass_ _ColClass_ _x_ _y_ _Group_;
    begingraph;
      layout datalattice rowvar=_RowClass_ columnvar=_ColClass_ ❶ /
          headerlabeldisplay=value ❷;
        sidebar / spacefill=false; ❸
          discretelegend 'A';❹
        endsidebar;
        layout prototype;❺
          scatterplot x=_x_ y=_y_ / group=_Group_ name='A';
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=datalattice;
  dynamic _ColClass_='origin' _RowClass_='Type'
      _x_='horsepower' _y_='mpg_city' _Group_='MSRP';
  format type $typeCompress. MSRP priceCat.;
  where type not in ('Hybrid','Sports');
run;
```

❶ The DATALATTICE layout expects one categorical variable explicitly designated for the rows (ROWVAR=) and another for the columns (COLUMNVAR=).

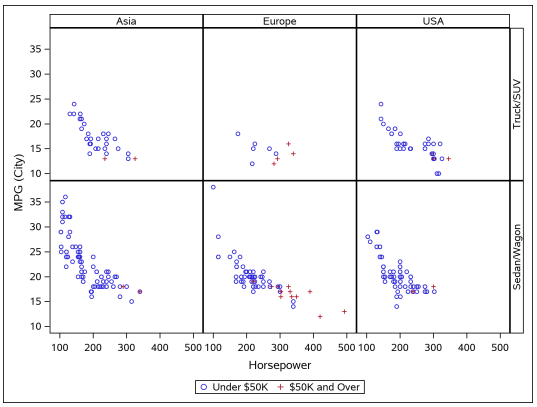
❷ Like the differences in the DISPLAY= option between TEMPLATE and SGPLOT/SGPANEL, the options for how row and column headers appear have somewhat different syntax.

❸ Legends are not drawn inside the panel set in SGPANEL, nor with the DATAPANEL or DATALATTICE layouts in TEMPLATE. In template language, the area that receives the legend is established with the SIDEBAR statement. By default, the sidebar fills the whole collection of space across the column panels or row panels (depending on its positioning), setting SPACEFILL=FALSE reduces it to only the amount of space required to fit the element placed in the sidebar.

❹ Just like other requests in TEMPLATE, plots do not automatically produce legends, so the DISCRETELEGEND statement is used in the sidebar to get the legend for the grouped plot.

❺ Since DATALATTICE makes a repeat of a plotting request across levels of the row and column variables, the required layout for the plot requests is PROTOTYPE.

Output 17: Row/Column Lattice on Two Variables in TEMPLATE



Even though Programs 14 and 17 have strong similarities, things can diverge quickly. For example, repositioning the legend to the top of the panel graph is an easy modification in Program 14, but adding ALIGN=TOP to the SIDEBAR options in Program 17 produces an unexpected result. The DATALATTICE layout relies on the column headers being positioned outside the lattice, so they can be placed on single rows and columns. However, when this is in place, they are also outside the sidebar when those occupy the same position–so the legend appears below the column headers. So, a seemingly obvious change does not produce the desired result.

Complex Layouts with Template

As seen, any of the graphs generated in Sections d n be generated via SGPLOT or SGPANEL, TEMPLATE and SGRENDER are not required. This section considers examples that must be done with TEMPLATE–SGPLOT and SGPANEL are not capable of producing the graphs given here.

The next section looks at examples of the LATTICE layout, which provides a great deal of flexibility in multi-panel graphs and charts. Indeed, each of the panels in the lattice layout can contain a different type of plot and/or use different variables.

Fundamentals of the Lattice Layout

Program 18 gives a basic use of the LATTICE layout, using both bar charts and scatter plots among the panels, and using two different response variables for the bar charts.

Program 18: The LATTICE Layout

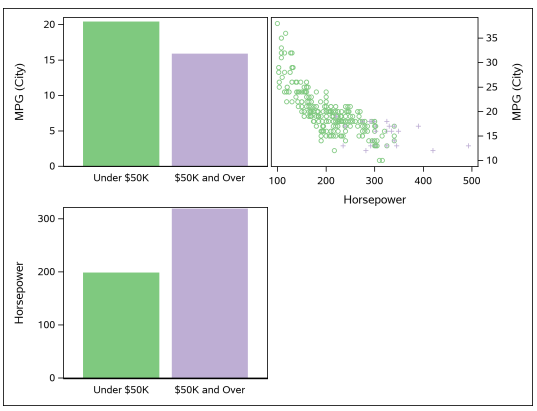
```
proc template;
  define statgraph lattice1;
    dynamic _x_ _y_ _Group_;
    begingraph / datacolors=(cx7fc97f cxbeaed4)
                 datacontrastcolors=(cx7fc97f cxbeaed4);
      layout lattice / columns=2 rows=2 order=columnmajor;❶
        layout overlay ❷ / yaxisopts=(offsetmin=0)
                                      xaxisopts=(display=(tickvalues));
          barchart category=_Group_ response=_y_ /
                                      group=_Group_ stat=mean;
        endlayout;
        layout overlay / xaxisopts=(display=(tickvalues));
          barchart category=_Group_ response=_x_ /
                                      group=_Group_ stat=mean;
        endlayout;
        layout overlay;❸
          scatterplot x=_x_ y=_y_ / group=_Group_ name='A' yaxis=y2;
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=lattice1;
  dynamic _x_='horsepower' _y_='mpg_city' _Group_='MSRP';
  format type $typeCompress. MSRP priceCat.;
  where type not in ('Hybrid','Sports');
run;
```

❶ The LATTICE layout includes COLUMNS= and ROWS= specification, and it is generally best to specify both. The ORDER=COLUMNMAJOR specification causes panels in a column fill before moving on to a new row (ROWMAJOR is also available).

❷ No repetition of plots is expected in the LATTICE layout, so the PROTOTYPE layout is not used. Each panel is seen as an independent, single-cell, so layouts appropriate for that purpose are used. Note that axis options can be set in the layout for each panel.

❸ There are three layouts nested within the LATTICE layout, so the final panel (bottom right) will be empty.

Output 18: The LATTICE Layout

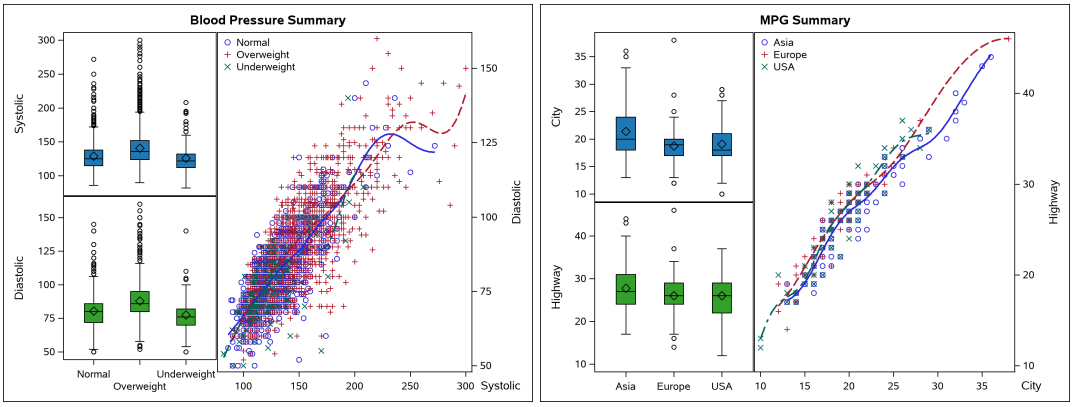


Notice in Output 18 that the x-axis for the bar charts is the same, and the y-axis is the same in the top row. However, no attempt is made to combine those axes. When it is appropriate to do so, it is possible to combine axes using an appropriate UNION option. Making unions of axes when not all align typically requires some nesting of layouts, which is taken up in the next section.

Advanced Lattice Layouts

Program 19 uses multiple layouts to create a custom panel graph, with a structure that is more general than those available in SGPANEL. This time, the target output is given first. The first instance applies to the Systolic and Diastolic variables from the Heart data set, with Weight Status as the category/grouping variable. The other uses City and Highway MPG from the Cars data, with Origin as the category. In general, this template should work with two quantitative variables and one categorical variable.

Output 19: Custom Multi-Panel Graph in TEMPLATE



So, there are two panels on the left that mimic the form of the boxplot given in Program 11, and the axis is adaptive to long category names. A larger right panel with a spline/scatter plot like Program 13 is grouped on the same variable as the category variable for the boxplots. Further, the common category axis of the boxplots is joined together into a single axis. While all panels use the same category variable, coloring is not set to be consistent across all panels–this will be taken up in the next example. Again, each panel is effectively independent to start in the LATTICE layout, so if commonality is reasonable and desired, it must be established explicitly in the template.

Program 19: Custom Multi-Panel Graph in TEMPLATE

```
proc template;
  define statgraph MultiGraph;
    dynamic _TitleText_ _ResponseLabel1_ _ResponseLabel2_
            _x_ _y1_ _y2_ _stat_;
    begingraph;
      entrytitle _TitleText_;
      layout lattice / columns=2 columnweights=(0.40 0.60);❶
        layout lattice / columndatarange=union;❷
         columnaxes;
          columnaxis / display=(tickvalues ticks) type=discrete
                       discreteopts=(tickvaluefitpolicy=stagger);
         endcolumnaxes;❸
          layout overlay / yaxisopts=(label=_ResponseLabel1_);
            boxplot x=_x_ y=_y1_ / fillattrs=(color=cx1f78b4) ;
          endlayout;❹
          layout overlay / yaxisopts=(label=_ResponseLabel2_);
            boxplot x=_x_ y=_y2_ / fillattrs=(color=cx33a02c);
          endlayout;❹
        endlayout;
        layout overlay/
                xaxisopts=(labelposition=right label=_ResponseLabel1_)
                y2axisopts=(label=_ResponseLabel2_);❺
           scatterplot x=_y1_ y=_y2_ / group=_x_ yaxis=y2 name='Scatter'
                                       includemissinggroup=false;
           pbsplineplot x=_y1_ y=_y2_ / group=_x_ yaxis=y2 smooth=15000;
           discretelegend 'Scatter' / location=inside halign=left
                          valign=top title='' across=1 border=false;❻
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sort data=sashelp.heart out=HeartSort;
  by weight_status systolic;
run;❼

proc sgrender data=HeartSort template=MultiGraph;
  dynamic _x_='weight_status' _y1_='systolic' _y2_='diastolic'
   _ResponseLabel1_='Systolic' _ResponseLabel2_='Diastolic'
   _stat_='mean' _TitleText_='Blood Pressure Summary';
run;

proc sort data=sashelp.cars out=CarSort;
  by origin mpg_city;
run;

proc sgrender data=CarSort template=MultiGraph;
  dynamic _x_='origin' _y1_='mpg_city' _y2_='mpg_highway'
   _ResponseLabel1_='City' _ResponseLabel2_='Highway'
   _stat_='mean' _TitleText_='MPG Summary';
   where type ne 'Hybrid';
run;
```

❶ Another generality, used in this first LATTICE specification, is the ability to set unequal sizes for panels. This request calls for two columns of graphics, with 40% of the available space allocated

to the first column, 60% to the second. Two columns are specified, and rows are not–this is not a problem as this lattice will be of a single row.

❷ The first column of the preceding lattice is established as another LATTICE layout. By default, this has one column (which relies on the default of the ORDER= option, shown in Program 20), and the x-axes for that column use the same variable and thus can made common with COLUMNDATARANGE=UNION.

❸ When x-axes are unionized, the COLUMNAXES block is used to set attributes (COLUMN2AXES, ROWAXES, and ROW2AXES are also available, when appropriate). COLUMNAXIS statement(s) set the axis options for each column. Since this only has a single column, only one COLUMNAXIS statement is needed. If multiple columns exist, a COLUMNAXIS statement should be set for each (if a COLUMNAXIS statement is missing, it is replaced with one that has DISPLAY=NONE, by default).

❹ This "inner" lattice contains two cells, boxplots created in separate OVERLAY layouts. Notice that each sets its y-axis attribute via the YAXISOPTS= option. XAXISOPTS= can be set in these, but is ignored in either given the union of the axes and the use of the COLUMNAXES block.

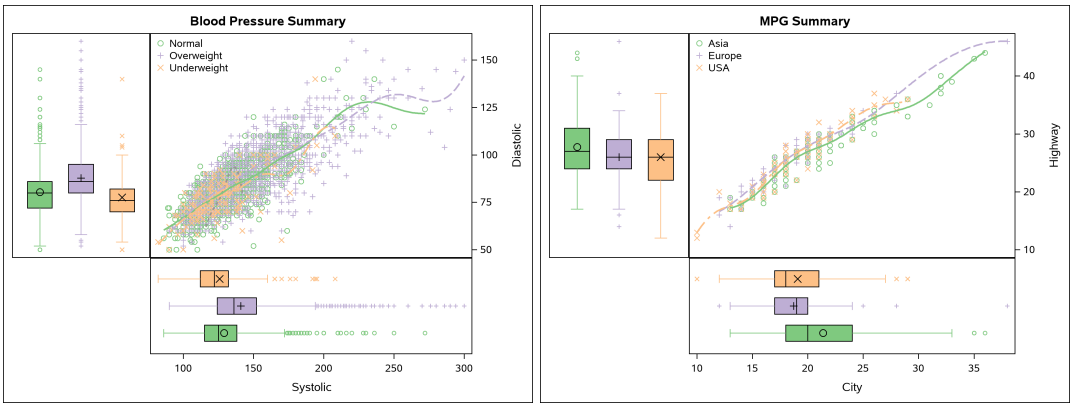
❺ This OVERLAY layout, containing the spline and scatter plot overlay, is the second column in the LATTICE layout noted in ❶. It is thus a single cell, so its x and y (y2) axis options are set here–there are no unions of axes for this plot.

❻ Since this is an overlay of two correlation plots with group variables. It only requires one legend, so only one is named and used in the DISCRETELEGEND statement. The scatterplot legend is chosen to make the legend display the marker symbols.

❼ Since one panel makes grouped spline curves, the data is sorted to this purpose. The _y1_ dynamic variable corresponds to the x variable for this plot.

Program 20 creates another multi-panel graph, illustrating a few more options and principles associated with these types of layouts. Again, the output is given first:

Output 20: Another Custom Multi-Panel Graph in TEMPLATE



Now each set of boxplots have their response axis aligned with the corresponding axis of the spline/scatter overlay. Category axes are suppressed on the boxplots, and the coloring is made to match to indicate the categories across all three plots.

Program 20: Another Custom Multi-Panel Graph in TEMPLATE

```
proc template;
  define statgraph MultiGraph2;
    dynamic _TitleText_ _ResponseLabel1_ _ResponseLabel2_
            _x_ _y1_ _y2_ _stat_;
    begingraph / datacontrastcolors=(cx7fc97f cxbeaed4 cxfdc086)
                 datacolors=(cx7fc97f cxbeaed4 cxfdc086);❶
      entrytitle _TitleText_;
      layout lattice / columns=2 rows=2
                       columnweights=(0.30 0.70) rowweights=(0.70 0.30)
                       columndatarange=union row2datarange=union
                       order=columnMajor;❷
        columnaxes;
          columnaxis / display=none;
          columnaxis / label=_ResponseLabel1_;
        endcolumnaxes;❸
        row2axes;
          rowaxis / label=_ResponseLabel2_;
          rowaxis / display=none;
        endrow2axes;❹
        layout overlay;
            boxplot x=_x_ y=_y2_ /
                    group=_x_ yaxis=y2 medianattrs=(color=black)
                    meanattrs=(color=black) outlineattrs=(color=black);
        endlayout;❺
        layout overlay /
                   yaxisopts=(display=none) xaxisopts=(display=none);
        endlayout;❻
        layout overlay;
              scatterplot x=_y1_ y=_y2_ / group=_x_ yaxis=y2
                          name='Scatter' includemissinggroup=false;
              pbsplineplot x=_y1_ y=_y2_ / group=_x_ yaxis=y2
                                           smooth=15000;
              discretelegend 'Scatter' / location=inside halign=left
                           valign=top title='' across=1 border=false;
        endlayout;❼
        layout overlay;
            boxplot x=_x_ y=_y1_ /
                    group=_x_ yaxis=y2 medianattrs=(color=black)
                    meanattrs=(color=black) outlineattrs=(color=black)
                    orient=horizontal;
        endlayout;❽
       endlayout;
    endgraph;
  end;
run;
❽
```

❶ As this plot is designed to rely on coloring and a legend to distinguish among categories only, the DATACOLORS and DATACONTRASTCOLORS lists are set to the same sequence of colors.

❷ This is a four-box lattice, with two squares–one in the upper right (large) and another in the lower left (small). The top left box is a tall, narrow rectangle, and the bottom right box is a short, wide rectangle. Column and row axes are made common, and since the plots are going to use the Y2 axis for the vertical axis, ROW2DATARANGE=UNION is required to unionize these. ORDER= determines how individual layouts go into this lattice; the default is RowMajor, meaning that layouts are placed into the lattice across a row until those cells are exhausted, then proceeding to the next row. Here, it has been set to ColumnMajor, so a column is filled before proceeding to the

next column. (There is no particular advantage to this here, it is just for demonstration purposes.)

❸ The COLUMNAXES block contains two COLUMNAXIS statements. Since there are two columns in this lattice, there are two column axes to define. The first suppresses all features, while the second sets a label and otherwise uses the default attributes.

❹ The ROW2AXES (remember, these plots use the Y2 axis) also has two ROWAXIS statements. However, the second one could be removed with no change to the result. Each axes block must have at least one axis statement, but if any other row or column axis does not have a specification, it is given the specification DISPLAY=NONE, by default. So, it is in this code for completeness and readability.

❺ The first cell, upper left, is populated with a vertical boxplot. Notice that the X= variable and the GROUP= variable are the same, this is again a method of getting the color list applied across the boxes while still preserving the axes. Since the axes are not displayed here, nor is a union applied, this is not technically required. A similar result can be achieved with a GROUP= variable and no X= variable–some option modifications are also needed in that instance. Note that some of the line and symbol colors that would otherwise be set by DATACONTRASTCOLORS= are reverted to black here.

❻ Next is the bottom left cell, and it is an empty plot. The axis options are overridden by the existence of the axes blocks, so the DISPLAY=NONE is not actually applied (and would be redundant if it was). It is here because something has to happen in the layout–it is not sufficient to say LAYOUT OVERLAY; followed by ENDLAYOUT;. The easiest way to make nothing happen is to put in some axis options that are ignored anyway (try replacing DISPLAY=NONE in either place with LABEL='This will not show up').

❼ The third cell is now in the top right, and so must be filled with the overlay of the scatter plot and spline curves.

❽ Last, is the horizontal boxplot, at the bottom right, otherwise similar to the other boxplot.

❾ For space considerations, the sorting and calls to PROC SGRENDER are the same as Program 19, except for the use of the current template in SGRENDER.

Conclusion

This is far from an exhaustive treatment of TEMPLATE–which would be exhausting. It is possible to incorporate axis tables and data labels, POLYGON plots are available, and attribute maps can be employed. Plus, there are a whole host of layouts, plotting statements, and options not yet covered. Getting good at TEMPLATE is about having a starting point and then making your way through the documentation to see what is possible and how to do it.

Contact Information

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

James Blum

Univeristy of North Carolina Wilmington

blumj@uncw.edu

http://people.uncw.edu/blumj

Appendix

References

Working in SGPLOT: Understanding the General Logic of Attributes; James Blum and Jonathan Duggins.