PharmaSUG China 2017 - Paper 007

Hands-on Graph Template Language (GTL)

Kriss Harris, SAS Specialists Limited, Hertfordshire, United Kingdom

ABSTRACT

Do you need to add annotations to your graphs? Do you need to specify your own colors on the graph? Would you like to add Unicode characters to your graph, or would you like to create templates that can also be used by non-programmers to produce the required figures? Great, then this topic is for you! In this hands-on workshop, you are guided through the more advanced features of the GTL procedure. There are also fun and challenging SAS® graphics exercises to enable you to more easily retain what you have learned.

INTRODUCTION

Graph Template Language (GTL) can be used to create sophisticated graphs, and there are some graphs which only GTL can create. Examples of these are trellised graphs with independent axes, and multi-cell graphs with different cell proportions, and multi-cell graphs with different plot types, i.e. one cell is a bar chart, and another is a box plot. With GTL you can also produce reproducible templates.

This paper intends to motivate you to use GTL, to show you how you can create plots in GTL which cannot be produced with the SG Procedures, and to show you how to create reusable templates. SAS® 9.4 was used to run the code in this paper.

If you're reading this paper it is assumed that you already know how to create graphs in GTL, and that you would like to know how to use some of the more advanced features of GTL. If you're unfamiliar with GTL then please see this paper first (Harris, Hands-on Graph Template Language: Part A, 2017).

The typical dataset used in this paper is from the CDISC SDTM / ADaM Pilot Project and this was obtained from the CDISC website (CDISC, 2013). The analysis dataset ADLBC was used in the below examples, and this dataset contained the laboratory results of 254 subjects. 86 of whom were treated with Placebo, 84 were treated with Xanomeline Low Dose and 84 subjects were treated with Xanomeline High Dose. In the examples below, the Creatinine laboratory parameter was typically used. Other times the laboratory parameters Alanine Aminotransferase, Aspartate Aminotransferase and Creatine Kinase were also used.

VARIABLES USED IN ADLBC

The examples in this paper use the variables *trtan* and *aval*. *Trtan* contains the numeric value of each treatment, and is used to display the treatments in the desired order. There are 3 numeric values for *trtan* and the treatments that these values correspond to are: 0 -> Placebo; 54 -> Xanomeline Low Dose; 81-> Xanomeline High Dose.

A format was used to apply the treatment labels to the numeric values. *Aval* contains the laboratory intensity measurements and depending on the filtering used, the intensities can be either the post-dose values or the baseline and post-dose values.

ADDING ANNOTATION TO FIGURES

The desire to annotate your figures is very common and there are at least four ways to add annotation. These are using the:

- AXISTABLE statement.
- DRAWTEXT statement.
- SCATTERPLOT statement with the MARKERCHARACTER = option.
- ENTRY statement

This paper shows examples of the first two options: AXISTABLE and DRAWTEXT, because they are believed to be slightly more useful. If you want to add summary statistics and align the summary statistics with the values on the x-axis, you can use the AXISTABLE statement. This can be seen in Figure 1. Firstly, you would need to calculate the summary statistics, using PROC MEANS, PROC UNIVARIATE or PROC SQL, and then add (merge) these summary statistics to the dataset that you want to plot. In the example below, the columns denoting the summary statistics are *bigN, mean* and *stddev*, and more details about these variables can be seen below:

- ***bigN*** contains the number of subjects in each treatment group.
- ***mean*** contains the mean of the post dose Creatinine values for each treatment group.
- ***stddev*** contains the standard deviation of the post dose Creatinine values for each treatment group.

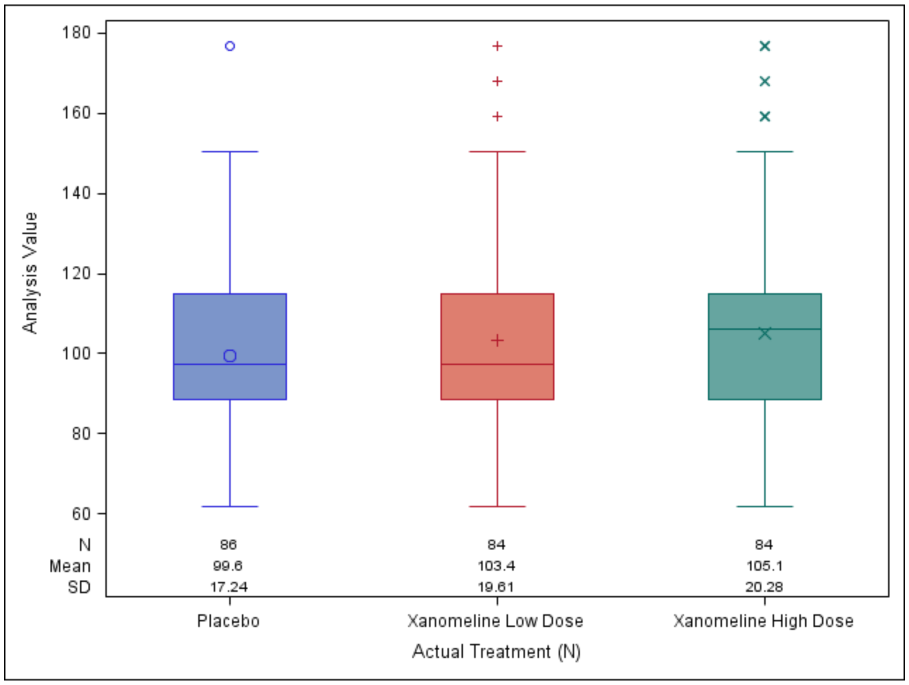
AXISTABLE

The code below shows how to use the AXISTABLE statement. The *x=trtan* option ensures that the values specified in the *value* option, align with the treatment values. This is the benefit of the AXISTABLE statement because it allows you to easily align the x-axis on the plot with the x-axis on the table. In the code below you will notice that the variable used for the x-axis in the BOXPLOT statement is the same as the variable used for the x-axis in the AXISTABLE statement: *trtan*. This allows for the table and the plot to be easily aligned.

The *stat=mean* option in the AXISTABLE statement specifies that the mean of the value you indicated should be calculated and displayed. For example, in the first AXISTABLE statement, this indicates that the mean of the *bigN* value should be displayed for each treatment group. For each treatment the *bigN* value only contains 1 distinct record, and so the same (correct) value of *bigN* for each treatment is displayed. The same thing happens for *mean*, and *stddev*. Other *stat* options in the AXISTABLE statement are *auto* and *sum*.

```
proc template;
  define statgraph boxplot_template;
    begingraph;
      layout overlay;
        boxplot x = trtan y = aval / group = trtan groupdisplay = cluster;
          innermargin / align=bottom opaque=true;
            axistable x = trtan value = bigN / stat=mean label = "N";
            axistable x = trtan value = mean / stat=mean label = "Mean";
            axistable x = trtan value = stddev / stat=mean label = "SD";
          endinnermargin;
      endlayout;
    endgraph;
  end;
run;
proc sgrender data = adlbc_all template = boxplot_template;
  by param;
  format trtan trtfmt.;
run;
```

Figure 1: Using AXISTABLE to show Summary Statistics

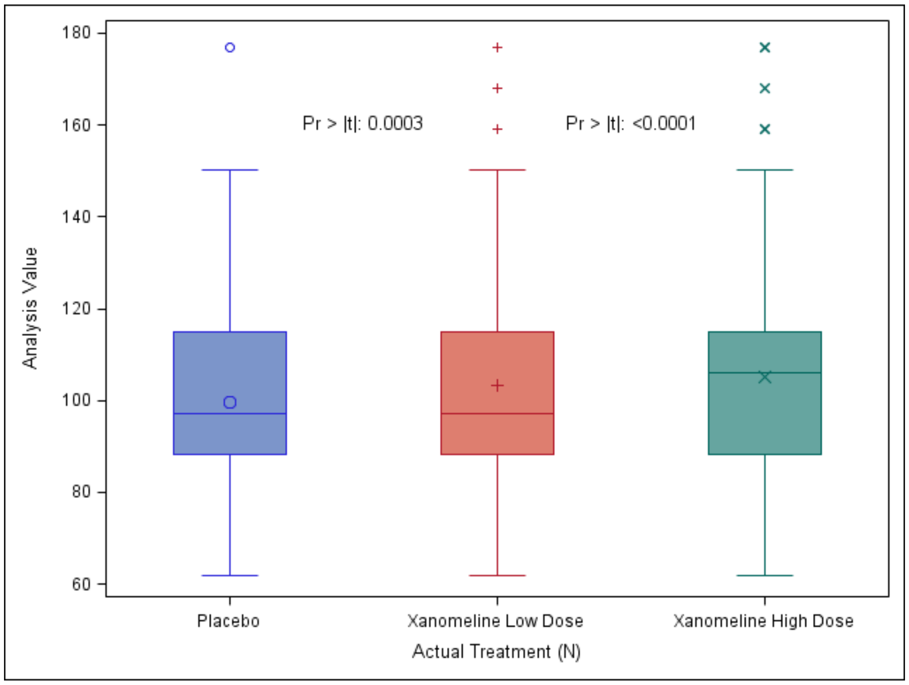


DRAWTEXT

If you want to add text anywhere on your graph you can use the DRAWTEXT statement. This is useful because it is extremely flexible in terms of placing text anywhere on your graph, e.g. placing text beside the Boxplots and/or beside the x-axis label. Placement of the text is controlled by the **x** and **y** values and also the options that are selected for the **xspace** and **yspace**. Figure 2 shows a Boxplot of Intensity by Treatment with p-values of the treatment different that were added using the DRAWTEXT statement. For more information on this type of annotation please see (Matange, Annotate your SGPLOT Graphs, 2014).

```
proc template;
  define statgraph boxplot_template;
    begingraph;
      layout overlay;
        boxplot x = trtan y = aval / group = trtan groupdisplay = cluster;
         drawtext "Pr > |t|: 0.0003" / x = 25 y = 160 xspace = datapercent
           yspace = datavalue width = 30;
        drawtext "Pr > |t|: <0.0001" / x = 75 y = 160 xspace = datapercent
           yspace = datavalue width = 30;
      endlayout;
    endgraph;
  end;
run;
```

Figure 2: Using DRAWTEXT to Annotate



CHANGING COLOR ORDER

There are at least two ways to change the color order in GTL. One way is to use the options in the begingraph statement with the options ***datacolors*** and ***datacontrastcolors***. Another way is to use **Attribute Maps**.

DATACOLORS AND DATACONTRASTCOLORS

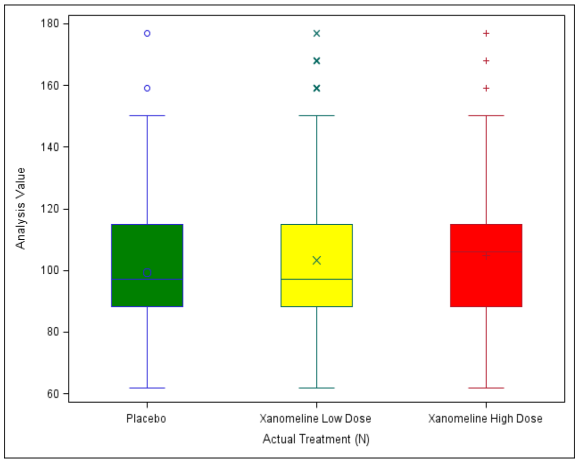
Using the datacolors option to assign colors to a group variable such as treatment group is fairly simple, all you do is put a list of the colors in the order to display, and then the color and order are associated with the **order of the data in the grouping variable**. The output in Figure 3 shows that the color of the treatment groups within the interquartile box has now changed. Green denotes Placebo, yellow denotes Xanomeline Low Dose and red denotes Xanomeline High Dose. You may be wondering why the treatment colors in Figure 3 are green, yellow and red, whilst within the *datacolors* option the colors are ordered (green red yellow). That is, it appears that the last two treatment groups have swapped colors. This is because by default the grouped values are mapped in the order of the data, and in this example the Placebo group was first in the data, followed by Xanomeline High Dose and then Xanomeline Low Dose. Therefore, the order of the groups within the data is very important if you decide to use *datacolors* to assign colors to your group variable. If you use *datacolors* then it is a good idea to also use *datacontrastcolors*. This is because in Figure 3 you will notice that the color of the markers and the whiskers do not always match the color within the interquartile range, whereas the template that produces Figure 4 uses *datacontrastcolors* and you will notice that the colors match up in Figure 4.

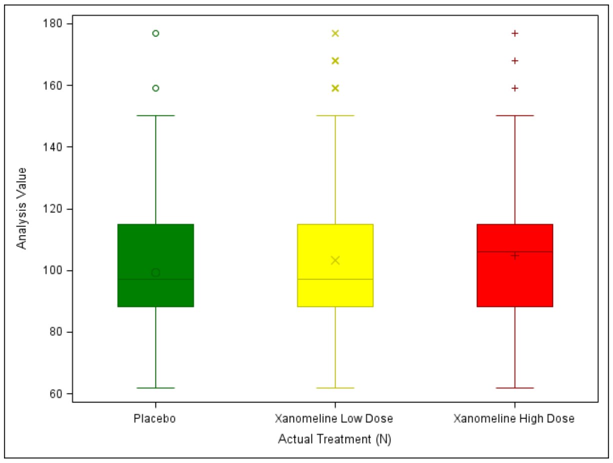
```
proc template;
  define statgraph boxplot_template2;
    begingraph / datacolors = (green red yellow);
      layout overlay;
        boxplot x = trtan y = aval / group = trtan groupdisplay = cluster;
      endlayout;
    endgraph;
  end;
run;
```

```
proc template;
  define statgraph boxplot_template2;
    begingraph / datacolors = (green red yellow) datacontrastcolors =
        (darkgreen darkred darkyellow);
      layout overlay;
        boxplot x = trtan y = aval / group = trtan groupdisplay = cluster;
      endlayout;
    endgraph;
  end;
run;
```

Figure 3: DATACOLORS

Figure 4: DATACOLORS and DATACONTRASTCOLORS

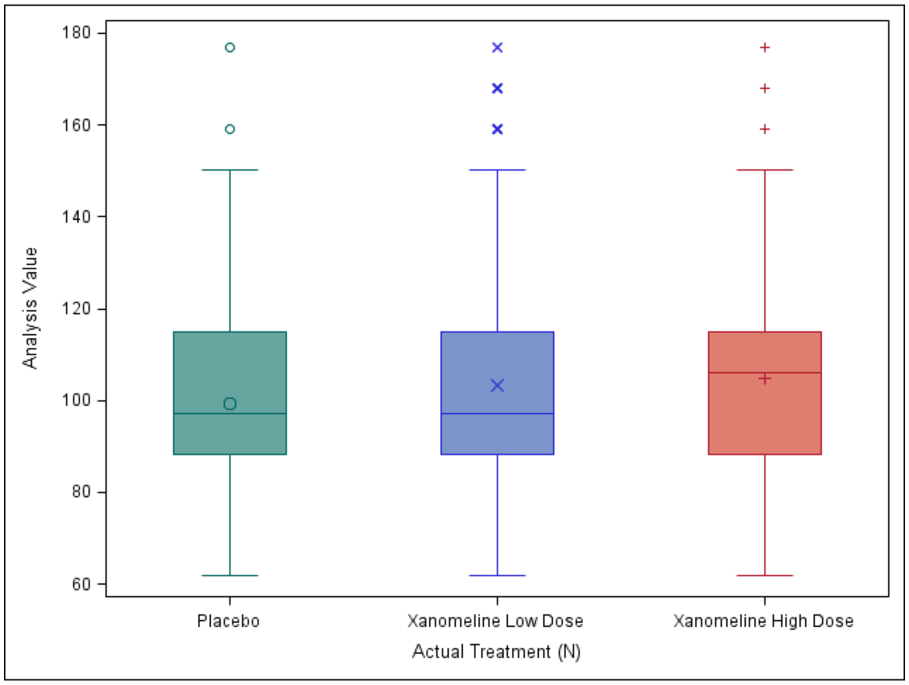




If you like the default SAS colors but just want to rearrange the order than you can use the style attributes such as GraphData3:color and GraphData2:color to choose your colors. The code below shows an example of the options that can be used within the begingraph statement and Figure 5 shows the result of the style attributes using a different order.

```
begingraph / datacolors = (graphdata3:color graphdata2:color
                             graphdata1:color)
             datacontrastcolors = (graphdata3:contrastcolor
               graphdata2:contrastcolor graphdata1:contrastcolor);
```

Figure 5: Using Style Attribute to change the color order



Attribute Maps

If you want to clearly specify the colors of the levels within your group variable, such as the colors of different treatments, then you should use Attribute Maps. The drawback of using the *datacolors* and *datacontrastcolors* options to set your colors is that you have no guarantee on the colors that are going to be assigned to the group levels. By default the colors are assigned by the order which the grouping variable appears in your data, and therefore if one of the grouping variables are missing or the grouping variable is not sorted as expected, then the colors can be assigned wrongly or inconsistently. Attribute Maps provides a way to specify the colors, as well as other options such as line patterns, symbols and text size for each of the levels in your group.

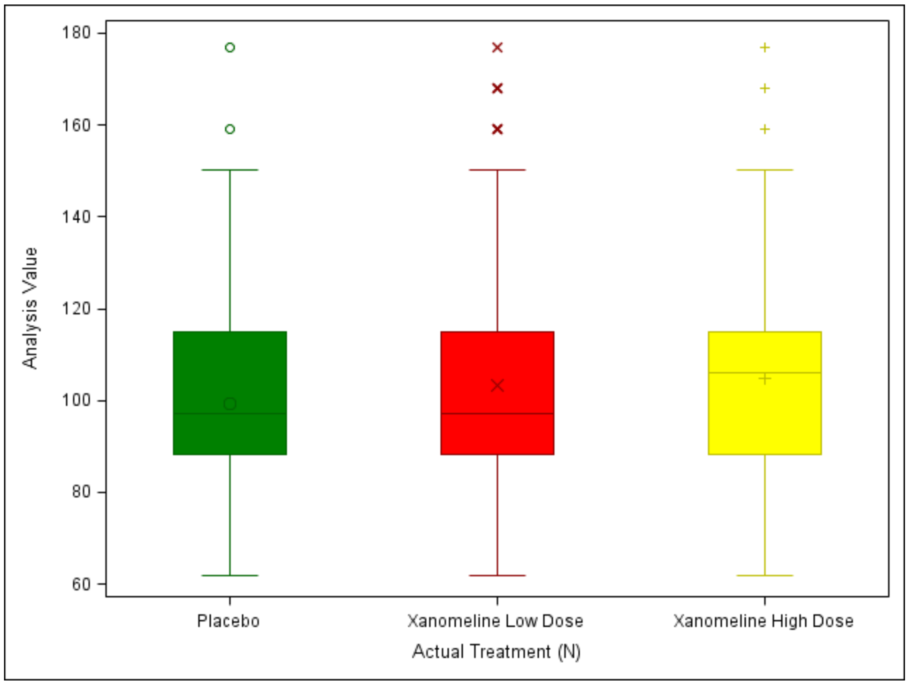
In the example below the DISCRETEATTRMAP statement was used to assign the attributes for the different treatments. To assign the treatment colors in the boxplots the ***fillattrs*** and ***markerattrs*** was used. *Fillattrs* controls the fill color of the interquartile range within the boxplot and *markerattrs* controls the color of the mean and the whiskers. In the code below, Placebo was given the color green for the *fillattrs* and dark green for the *markerattrs*. This is interesting, and this shows that in general the *markerattrs* list of colors use colors that are in the *contrastcolors* list. The colors in *contrastcolors* are slightly darker.

The DISCRETEATTRVAR statement was used to link the graphical properties of the DISCRETEATTRMAP namely “Attributes” with the variable in the dataset named *trtan*. Therefore when the FORMATTED value in *trtan* matches with a value in DISCRETEATTRMAP, then those specified attributes for the treatment will appear on the graph. The formatted value is important here because, the variable *trtan* is a numeric variable which has been formatted to show Placebo, Xanomeline Low Dose, and Xanomeline High Dose, instead of the numbers 0, 54, and 81 respectively. Using the original values of 0, 54 and 81 instead of the formatted values will not give the desired attributes in this instance.

Figure 6, shows the results of using the attribute maps to assign the colors.

```
/* Define the attribute map and assign the name "attributes" */
discreteattrmap name="attributes" /discretelegendentrypolicy=attrmap;

   value "Placebo" / fillattrs=(color=green)  markerattrs=(color=darkgreen)
       lineattrs=(color=darkgreen); *lineattrs is needed for the whiskers;

   value "Xanomeline Low Dose" / fillattrs=(color=red) markerattrs=(color=darkred)
       lineattrs=(color=darkred);

   value "Xanomeline High Dose" /fillattrs=(color=yellow)
       markerattrs=(color=darkyellow) lineattrs=(color=darkyellow);

enddiscreteattrmap;

/* Create attribute map variable GROUPATTRIBUTES to associate attribute
 map ATTRIBUTES with Treatment Group */
discreteattrvar attrvar=groupattributes var=trtan attrmap="attributes";

layout overlay;
    boxplot x = trtan y = aval / group = groupattributes groupdisplay = cluster;
endlayout;
```

Figure 6: Attribute Maps



So far all the graphs in this paper have only one layout type: LAYOUT OVERLAY. All options that you have seen so far will also work for multi-cell graphs

MULTI-CELL GRAPHS

Independent Axes – LATTICE layout

Plots with independent axes can be created using layouts that you are already familiar with: LATTICE and OVERLAY. (Hebber & Matange, 2013) shows how you can simulate the subsetting of observations in GTL. You can use these ideas as illustrated in the program below to create cells that have independent axes. Evidently, it is possible for you to create cells using GTL that have independent axes without simulating the subsetting of observations. However, using other methods to create cells that have independent axes involves either using the ODS LAYOUT statement, which will be discussed later, or involves reformatting your dataset, so that there is an x-variable and a y-variable for every parameter that you want to plot. Therefore if you have lots of parameters, and you need to reformat your dataset you will end up with a dataset with a large number of columns.

The program below shows a snippet of the full GTL code which produces the template for Figure 7. The full code is provided in the Appendix. The snippet shows the code used to produce the independent axes for the laboratory parameters Alanine Aminotransferase and Aspartate Aminotransferase. Figure 7 shows boxplots of intensity by visit number for four laboratory parameters, and each laboratory parameter has its own independent axes. Figure 7 was created using the layouts LAYOUT LATTICE and LAYOUT OVERLAY.

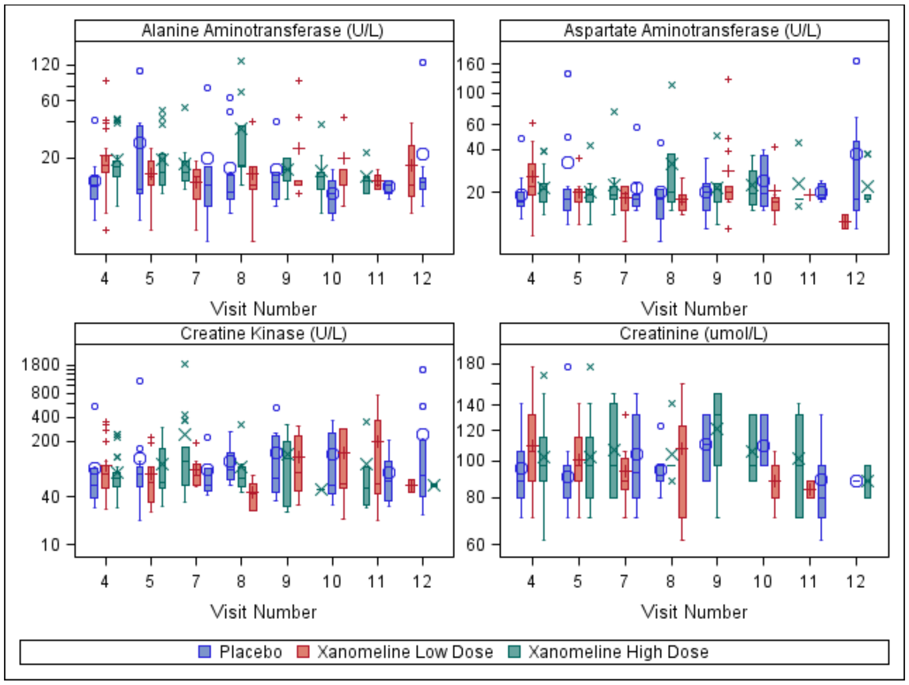
In the program below the EVAL and IFN functions are used to only show the Alanine Aminotransferase intensity results in the first cell. This was then repeated for the second, third and fourth cell so that each cell only showed the result of one parameter and each cell had its own independent axes.

Regarding the IFN function, there is another matching function called the IFC function, and these functions often get mixed up. A way to remember the differences between the IFN and IFC functions is to remember that the “N” (numeric) or “C” (character) part refers to the type of value that the function is returning, and does not take into consideration the value type of the logical expression, i.e. the original value being tested. For example, for the y-axis, IFN was used to return the numeric AVAL values when the PARAM was equal to “Alanine Aminotransferase (U/L)”, otherwise (numeric) missing values were returned.

The BLOCKPLOT statement was used to mimic the appearance of the ***classvars*** in the DATAPANEL layout. The BLOCKPLOT creates one or more strips of rectangular blocks containing text values. In Figure 7 the BLOCKPLOT has been used in conjunction with the INNERMARGIN block to plot the laboratory parameter names at the top of each cell. The variable ***ALT*** contains the value “Alanine Aminotransferase (U/L)”, and the option *block=alt* is what allows the text to be displayed within the cell header. The *visitnum* variable used in the BLOCKPLOT statement is the same as the variable in the BOXPLOT statement. Although the reason why there is only one value of “Alanine Aminotransferase (U/L)” in the cell header is because by default there is a *repeatedvalues=false* option within the BLOCKPLOT statement. If the *repeatedvalues=true* option was used then “Alanine Aminotransferase (U/L)” would have been repeated for each visit number. In the BLOCKPLOT statement the options *display = (outline values)* were also used. The *outline* option displays the outline around the block to give the rectangle around the laboratory parameter values, and the *values* option displays the actual value which is in the variable *alt*, in this case the laboratory value “Alanine Aminotransferase (U/L)”.

```
layout lattice / columns = 2 rows = 2;

  * First Column, First Row;
  layout overlay / yaxisopts = (type = log display = (line ticks tickvalues));

     innermargin / align = top;
        blockplot x = visitnum block = alt / valuehalign = center display = (outline values);
     endinnermargin;

     boxplot x = visitnum y = eval(ifn(param = "Alanine Aminotransferase (U/L)", aval, .)) /
        group = trtan groupdisplay = cluster;

  endlayout;

  * Second Columnm, First Row;
  layout overlay / yaxisopts = (type = log display = (line ticks tickvalues));

     innermargin / align = top;
        blockplot x = visitnum block = ast / valuehalign = center display = (outline values);
     endinnermargin;

     boxplot x = visitnum y = eval(ifn(param = "Aspartate Aminotransferase (U/L)", aval, .)) /
        group = trtan groupdisplay = cluster;

  endlayout;
```

Figure 7: Boxplot with Independent Axes – LATTICE and OVERLAY layout



A few drawbacks of using the method in the above program to obtain independent axes in each cell are:

- This is a manual method, and therefore if you have 9 cells, you will need to create 9 different cell groupings of cell header, entry, layout overlay and boxplot statements, and also carefully add each logical expression.
- The default label of the x-axis and y-axis will need to be changed, otherwise a string that looks very similar to the expression will be shown as the x-axis and y-axis label on each cell.

(Harris, Picture this: Hands-on SAS Graphics Session, 2015, pp. 6-7) shows how you can address some of the drawbacks in the above code to make it more generalized with the aid of a Macro.

INDEPENDENT AXES – ODS LAYOUT

The ODS LAYOUT statement can be used to produce independent axes in a simpler way than using the LATTICE layout. This is because there is no need to use the EVAL, IFC, and IFN functions and also there's no need for all your data to be in one dataset. Although the drawbacks of using the ODS LAYOUT statement for independent axes are that it only works well when you produce a .pdf file, and also the .pdf typically has less resolution than the .png files or other image files that you can create when using the LATTICE layout. It is better to view the plot when it is in the .pdf file than to copy and paste the image from the .pdf file into a word document.

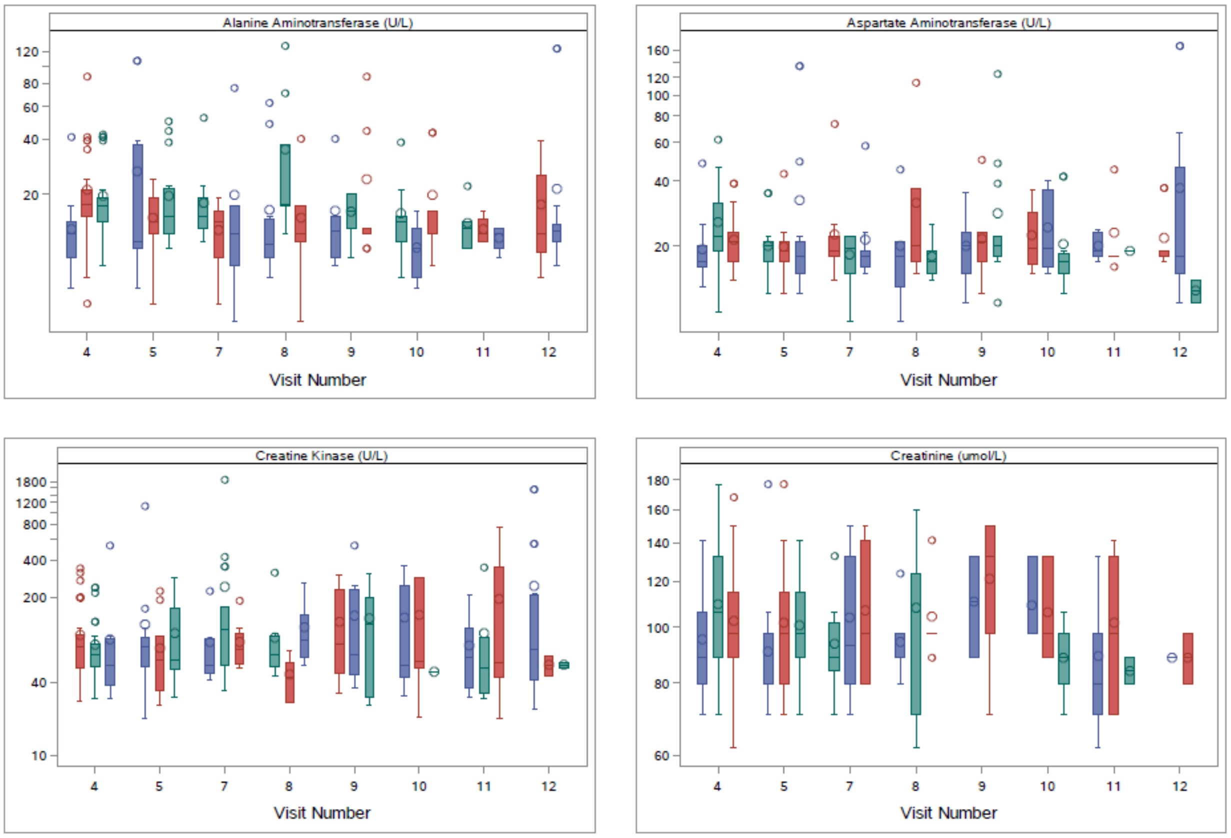
You can use a simple reusable template to produce a plot with independent axes, as shown below. The GTL code below will produce a boxplot of the intensity by the visit number for a given dataset. The DYNAMIC statement is used to display the name of the laboratory parameter by substituting *cellhead* for another variable in the dataset that will be rendered.

```
proc template;
   define statgraph boxplot_template;
      begingraph;
        dynamic cellhead;
        layout overlay / yaxisopts=(type=log display = (line ticks tickvalues));
            innermargin / align = top;
              blockplot x = visitnum block = cellhead / valuehalign = center
                  display = (outline values);
            endinnermargin;
            boxplot x = visitnum y = aval / group = trtan groupdisplay = cluster;
         endlayout;
      endgraph;
   end;
run;
```

ODS LAYOUT in conjunction with ODS REGION can produce a plot with independent axes. Below only shows a snippet of the code which contains 2 out of the 4 regions which were used. In the first region, denoted by *ods region row=1 column=1* you can see that the SGRENDER procedure only selects the data where param = "Alanine Aminotransferase (U/L)". In the second region only the Aspartate Aminotransferase (U/L) data is used. The selection of data within each region is what makes it possible to produce a plot with independent axes. The dynamic statement is used to display the laboratory name within the cell header. In the first region, the ***ALT*** variable is specified in the *dynamic cellhead* statement and in the second region the ***AST*** variable is specified. Similarly to the data used for the LATTICE layout, the variable ***ALT*** contains the value "Alanine Aminotransferase (U/L)". Using dynamic variables makes the templates more reusable and flexible. Figure 8 shows the graph with independent axes which was produced by the ODS LAYOUT statement.

```
options orientation=landscape nonumber nodate;
ods graphics / reset = all imagename="ods_layout" height= 3in width = 4.5in;
ods pdf file = "&outpath\ods_layout.pdf";

   ods layout Start columns=2 rows=2 row_gutter=0.3in column_gutter=0.3in;
      ods region row=1 column=1;
         proc sgrender data = adlbc_all template = boxplot_template;
            dynamic cellhead = "alt";
            where param = "Alanine Aminotransferase (U/L)";
            format trtan trtfmt.;
         run;

      ods region row=1 column=2;
         proc sgrender data = adlbc_all template = boxplot_template;
            dynamic cellhead = "ast";
            where param = "Aspartate Aminotransferase (U/L)";
            format trtan trtfmt.;
         run;
   ods layout end;
ods pdf close;
```

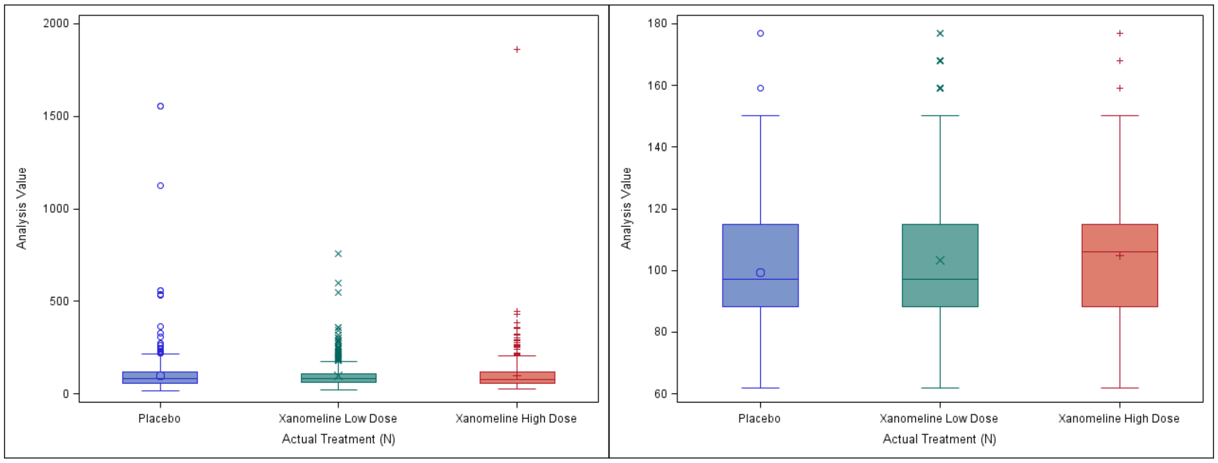
Figure 8: Independent Axes using ODS LAYOUT



CONDITIONAL BLOCK

Similar to when you use IF…THEN…ELSE in a DATASTEP, it is often necessary to add conditions. Conditional logic can be used in GTL to determine which statements to be rendered. This is useful if you want to display a variable on the log scale, but keep another variable on the original scale. The reason why you may want to do this can be seen in Figure 9. Figure 9 below show boxplots of the intensity values by treatment for two laboratory parameters. The laboratory parameter on the left is Creatine Kinase and the parameter on the right is Creatinine, and it is apparent when looking at the figures that the Creatine Kinase values deserve to be displayed on the log scale.

Figure 9: In Need of a Conditional Block

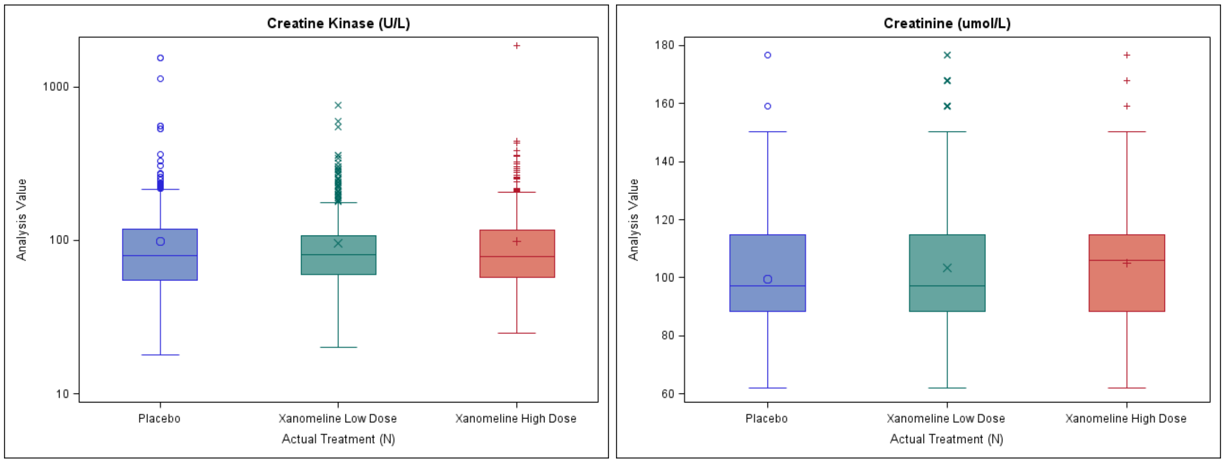


In Figure 9 you can see that there are unusually high Creatine Kinase values, and therefore using the log transformed scale may be a better approach, as seen below in Figure 10.

The code below uses the conditional block to output the Creatine Kinase values on the log scale. The dynamic variable *_byval_* has been defined and works in conjunction with the variable in the by statement. The *_byval_* dynamic variable is used for two purposes in the code below. The first purpose is to help evaluate which section of the template should be run, that is if the value is equal to "Creatine Kinase (U/L)" than the first section of the template with the y-axis on the log scale will be run, otherwise the other section will be run with the y-axis on the default linear scale. The second purpose of the *_byval_* statement is to assign a title to the figures. The statement **ENTRYTITLE** *_byval_*, does this. Figure 10 shows the result of using the conditional block.

```
proc template;
  define statgraph conditional;
    begingraph;
    dynamic _byval_;

      if (_byval_ = "Creatine Kinase (U/L)")
        entrytitle _byval_;
        layout overlay / yaxisopts=(type=log);
          boxplot x = trtan y = aval / group = trtan groupdisplay = cluster;
        endlayout;

      else
        entrytitle _byval_;
        layout overlay;
          boxplot x = trtan y = aval / group = trtan groupdisplay = cluster;
        endlayout;
      endif;

    endgraph;
  end;
run;
```

Figure 10: Using the Conditional Block



ADDING UNICODE TO LEGEND

Adding Unicode symbols to your plots is a matter of using the *ODS ESCAPECHAR* statement and using the correct Unicode characters. There is a great blog on Unicode at (Heath, 2011). Table 1, below shows the Unicode characters for some of the symbols which are requested for inclusion on the plots. When using the Unicode symbols in GTL you need to make a slight change. So if you want to display the less than or equal to symbol, then instead of using **U+2264** you need to use **"2264"x** instead. This is the same pattern for all the other Unicode characters. Also in GTL, instead of using Unicode characters for alpha, you can use the actual word alpha, and instead of using Unicode characters for superscript 2, you can use the SUP function.

Table 1: Common Symbol Requests

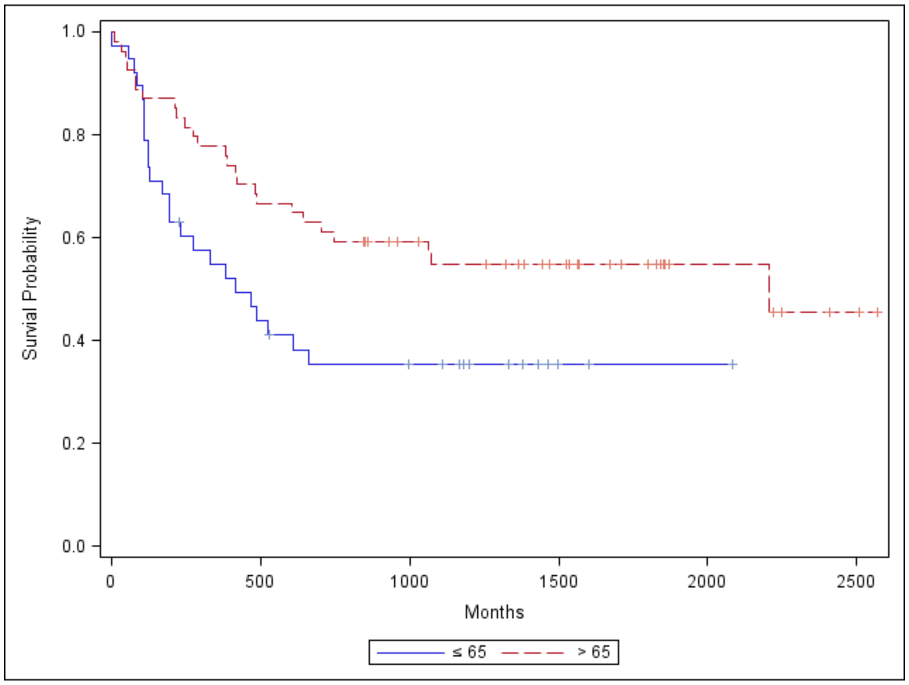
Symbol	Meaning	Unicode
≤	Less than or equal to	U+2264
≥	More than or equal to	U+2265
²	Superscript 2	U+00B2
α	Lower case Alpha	U+03B1
µ	Lower case Mu	U+03BC

Adding Unicode symbols in some places in your graph such as the legend can be fiddly when using up to and including SAS 9.4 maintenance 2. The code below shows you how you can add Unicode symbols to your legend, and the main thing that needs to be done to achieve this is to use the expression to subset the data, similar to the previous example producing the independent axes in the LATTICE layout. Using the expression to subset your data for stratum 1 and stratum 2 allows you to then explicitly specify the *legendlabel* which can contain Unicode characters. As previously mentioned, the Unicode character '2264'x denotes the less than or equal to sign, which can be seen in the legend in Figure 11. If you were to use the *group*= option to produce a separate STEPPLOT and SCATTERPLOT, then it would not be possible to add the Unicode into the legend in this version of SAS, unless you were to use the DRAWTEXT statement to create your own legend. The data used in this example is based on the Kaplan Meier results from the SASHELP.BMT dataset.

In the code below the STEPPLOT statement is used to display the Kaplan Meier curve, and the SCATTERPLOT statement is used to display the censored values. These are the vertical lines. The first set of STEPPLOT and SCATTERPLOT statements are to draw the blue survival curve, where the age of the subjects is ≤ 65, and the second set of statements are to draw the red survival curve for the subjects that where aged over 65 years old.

```
layout overlay /
   yaxisopts = (label = "Survial Probability" linearopts = (viewmin = 0 viewmax =1))
   xaxisopts = (label = "Months");

    stepplot y = eval(ifn(stratum = 1, survival2, .)) x=eval(ifn(stratum = 1, t, .)) /
        name = "leg" legendlabel = "^{unicode '2264'x} 65" lineattrs = GRAPHDATA1;
    scatterplot y=eval(ifn(stratum = 1 and _CENSOR_ = 1, survival2, .))
        x=eval(ifn(stratum=1 and _CENSOR_ =1, t, .)) /
        markerattrs = (color = GRAPHDATA1:color symbol = plus);

    stepplot y=eval(ifn(stratum = 2, survival2, .)) x = eval(ifn(stratum = 2, t, .)) /
        name = "leg2" legendlabel = "> 65" lineattrs = GRAPHDATA2;
   scatterplot y=eval(ifn(stratum = 2 and _CENSOR_ = 1, survival2, .))
       x= eval(ifn(stratum=2 and _CENSOR_ =1, t, .)) /
       markerattrs = (color = GRAPHDATA2:color symbol = plus);

   discretelegend "leg" "leg2";

endlayout;
```

Figure 11: Kaplan-Meier Curve with Unicode in the Legend



CONCLUSION

GTL can be used to produce very sophisticated graphs. Annotation can be added fairly easily with the AXISTABLE and DRAWTEXT statements. The colors of your categorizing variables can be changed using DATACOLORS, DATACONTRASTCOLORS and ATTRIBUTE MAPS, although ATTRIBUTE MAPS is the best one to use because you will always get the color that you expect.

The LATTICE layout and the ODS LAYOUT can be used to produce plots which have independent axes. The ODS LAYOUT is simpler to produce a plot with independent axes, but the LATTICE layout can produce plots with better resolution.

REFERENCES

CDISC. (2013). CDISC. Retrieved June 2015, from SDTM/ADaM Pilot Project: *http://www.cdisc.org/system/files/members/article/application/zip/updated_pilot_submission_package.zip*

Harris, K. (2015). Picture this: Hands-on SAS Graphics Session. *PharmaSUG 2015* (pp. 6-7). Orlando: PharmaSUG.

Harris, K. (2017). Hands-on Graph Template Language: Part A. *SAS Global Forum.* Orlando: SAS Global Forum.

Heath, D. (2011, 11 14). *The Power of Unicode*. Retrieved 03 06, 2017, from Graphically Speaking: http://blogs.sas.com/content/graphicallyspeaking/2011/11/14/the-power-of-unicode/

Hebber, P., & Matange, S. (2013). Free Expressions and Other GTL Tips. *SAS Global Forum 2013* (p. 3). San Francisco: SAS Global Forum.

ACKNOWLEDGMENTS

I would like to thank Adrienne Bonwick for reviewing this paper. I would also like to thank Sharon Carroll for the support she gave me whilst preparing the paper.

RECOMMENDED READING

- Matange, S. (2013). *Getting Started with the Graph Template Language in SAS: Examples, Tips, and Techniques for Creating Custom Graphs.* Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Kriss Harris
SAS Specialists Limited
italjet125@yahoo.com
http://www.krissharris.co.uk

APPENDIX

FULL CODE FOR INDEPENDENT AXES USING LATTICE LAYOUT

```
proc format;
  value trtfmt 0 = "Placebo"
              54 = "Xanomeline Low Dose"
              81 = "Xanomeline High Dose";
run;

*-----------------------------------------------------------------*;
*                          Datasets                                *;
*-----------------------------------------------------------------*;

* ADLBC;
data adlbc_all;
  set source.adlbc;
  where param in ("Creatine Kinase (U/L)", "Creatinine (umol/L)", "Aspartate Aminotransferase
(U/L)", "Alanine Aminotransferase (U/L)") and anl01fl = "Y";
run;
```

```
* Making labels appear similar to DataPanel layout;
data adlbc_all2;
  set adlbc_all;
  alt = "Alanine Aminotransferase (U/L)";
  ast = "Aspartate Aminotransferase (U/L)";
  ck = "Creatine Kinase (U/L)";
  creat = "Creatinine (umol/L)";
run;

proc sort data = adlbc_all2;
  by param visitnum;
run;


proc template;
  define statgraph boxplot_template;
    begingraph;
      layout lattice / columns = 2 rows = 2;

        * First Column, First Row;
        layout overlay / yaxisopts = (type = log display = (line ticks tickvalues));
             innermargin / align = top;
               blockplot x = visitnum block = alt / valuehalign = center display = (outline
values);
             endinnermargin;
          boxplot x = visitnum y = eval(ifn(param = "Alanine Aminotransferase (U/L)", aval,
.)) / group = trtan groupdisplay = cluster;
        endlayout;

        * Second Columnm, First Row;
        layout overlay / yaxisopts = (type = log display = (line ticks tickvalues));
             innermargin / align = top;
               blockplot x = visitnum block = ast / valuehalign = center display = (outline
values);
             endinnermargin;
          boxplot x = visitnum y = eval(ifn(param = "Aspartate Aminotransferase (U/L)", aval,
.)) / group = trtan groupdisplay = cluster;
        endlayout;

            * First Column, Second Row;
            layout overlay / yaxisopts = (type = log display = (line ticks tickvalues));
              innermargin / align = top;
                blockplot x = visitnum block = ck / valuehalign = center display = (outline
values);
              endinnermargin;
          boxplot x = visitnum y = eval(ifn(param = "Creatine Kinase (U/L)", aval, .)) /
group = trtan groupdisplay = cluster;
        endlayout;

        * Second Columnm, Second Row;
        layout overlay / yaxisopts = (type = log display = (line ticks tickvalues));
             innermargin / align = top;
               blockplot x = visitnum block = creat / valuehalign = center display = (outline
values);
             endinnermargin;
             boxplot x = visitnum y = eval(ifn(param = "Creatinine (umol/L)", aval, .)) /
group = trtan groupdisplay = cluster name = "leg";
        endlayout;

        * Legend;
            sidebar / align = bottom;
              discretelegend "leg";
            endsidebar;

      endlayout;
     endgraph;
  end;
run;

ods listing;
ods graphics / reset = all imagename = "boxplot_lattice_independent3";
```

```
   proc sgrender data = adlbc_all2 template = boxplot_template;
     format trtan trtfmt.;
   run;

   ods graphics / reset = all;
goptions reset = all;
```