Paper 1119-2017

A Step by Step Solution to Create a Customized Graph for Grouped Data Using SAS® Graph Template Language

Elva Chen, Pharmacyclics, An AbbVie Company

ABSTRACT

Meaningful and interpretable visualization is critical to turn data into knowledge. Almost every software systems provide settings to create default graphs. Although creating default graphs provided by these software is straightforward, creating customized graphs for specialized analysis and specific interpretation are challenging. Furthermore, in most analysis, data are collected and summarized by groups. To create customized graph for group data for specific layouts and desired appearances are even more complicated. This paper provides a step by step, start-from-scratch approach to create a customized graph for group data using SAS Graph Template Language (GTL).

INTRODUCTION

GTL is a powerful language to visualize data. An extension to the Output Delivery System (ODS), GTL is designed to create complex analytical graphics that are not available from traditional SAS procedures. GTL is powerful and flexible, but its power and flexibility come with some complexity. Syntax and style of GTL are perceived deviant from conventional SAS syntax, which makes most SAS users feeling not knowing enough to use GTL. When there are needs to create customized graphs, most people feel not sure how to start.

In fact, any complex graph starts from simple graph designs. Understanding the basic graphics structures and GTL language elements can enable us to establish a thinking process which can help us to first build points of references to start with, then moving from simple patterns to a highly complex graph. Following the thinking process and GTL language elements, to create a highly complex, customized graph from basic patterns are approachable.

STEP 1: ANALIZE INPUT DATASET AND TARGET GRAPH

Before starting writing codes to create plot, the first step is to examine the desired graph, figuring out the basic patterns then relating them to the data. For example, the target plot is to display means with confidence intervals at different visits for four individual groups from the dataset in Table 1. The dataset contains groups, visit weeks, means, lower and upper confidence intervals. In the plot, x-axis is the visit week, y-axis illustrates the means and confidence interval bands. To make the graph interpretable, each group for each visit week needs to form a cluster.

	groupcd	visit	Mean	Lower CI	Upper CI
1	Eastern	Week 1	45	25.955413024	67.044586976
2	Eastern	Week 2	40.875	25.137855134	59.612144866
3	Eastern	Week 3	40.5	23.940012755	60.059987245
4	Eastern	Week 4	38.25	20.551998459	57.948001541
5	Eastern	Week 5	27.5	12.776709223	44.223290777
6	Eastern	Week 6	22.5	10.510694271	37.489305729
7	Eastern	Week 7	20.5	7.8210450527	35.178954947
8	Eastern	Week 8	16.25	3.8755315022	30.624468498
9	Central	Week 1	83.75	74.367915403	98.132084597
10	Central	Week 2	84.125	76.789358769	96.460641231
11	Central	Week 3	81.375	73.62604454	92.12395546
12	Central	Week 4	80.75	71.377659498	92.122340502
13	Central	Week 5	63.75	41.947484206	88.552515794
14	Central	Week 6	51.125	29.620903841	75.629096159
15	Central	Week 7	39.75	21.277848901	61.222151099
16	Central	Week 8	29.5	13.64871895	48.35128105
17	Mountain	Week 1	95.75	89.974555968	106.52544403
18	Mountain	Week 2	92.375	85.861678287	103.88832171
19	Mountain	Week 3	92.25	88.526032625	99.973967375
20	Mountain	Week 4	93.875	86.545451344	103.20454866
21	Mountain	Week 5	79.5	71.686306891	90.313693109
22	Mountain	Week 6	73.25	53.900144804	95.599855196
23	Mountain	Week 7	55.5	25.899872214	88.100127786
24	Mountain	Week 8	29.875	4.1655193511	58.584480649
25	Pacific	Week 1	85	75.80371875	97.19628125
26	Pacific	Week 2	86	78.578319856	95.421680144
27	Pacific	Week 3	81.666666667	73.056066706	93.277266628
28	Pacific	Week 4	82.666666667	76.253709135	93.079624198
29	Pacific	Week 5	51.166666667	19.533324033	85.8000093
30	Pacific	Week 6	45.333333333	16.818988195	76.847678471
31	Pacific	Week 7	17.833333333	7.4443614988	31.222305168
32	Pacific	Week 8	12.166666667	5.8231162688	20.510217065

Table 1. Sample Data

STEP 2: UNDERSTAND THE BUILDING BLOCKS OF GTL

Two procedures are essential building blocks in GTL: PROC TEMPLATE and PROC SGRENDER. PROC TEMPLATE defines basic plot types and their appearances; PROC SGRENDER associates the data to the graph template.

The structures of PROC TEMPLATE and SGRENDER are outlined as below:

```
proc template;
   Define statgraph template-name;
          Beginegraph / <options>;
                        layout overlay;
                        type-of-plot
```

```
            Endgraph;
        end ;
run;

proc sgrender data=data-set-name template=template-name;
……
run;
```

DEFINE statement of PROC TEMPLATE creates specialized SAS templates, that are used for controlling the appearance of ODS output. STATGRAPH is a reserved graph template from PROC TEMPLATE. Running PROC TEMPLATE compiles and saves the template; running PROC SGRENDER creates actual graph.

STEP 3: START FROM SCRATCH

Plot patterns available in GTL are points of references to start with. These plot patterns are the initial tools for creating complicated graphs. Following are basic types of plots GTL provides:

A. BASIC PLOTS

bandplot
blockplot
fridgeplot
needleplot
scatterplot
seriesplot
stepplot
vectorplot

B. CATEGORICAL PLOTS

barchart
linechart
piechart
waterfallchart

C. DISTRIBUTION PLOTS

boxplot
densityplot
ellipse

histogram

D. FIT PLOTS

loessplot

pbsplineplot

regressionplot

modelband

Analyzing the plot types, data we have and desired graph, we need a scatter plot to display means and their confidence interval bands, a series plot superimposes on the scatter plot and connects the means for each visit, and each group in same visit week need to form a cluster.

Following is the example to build basic plot types in GTL:

```
proc template;
   define statgraph meanplot;

   begingraph;
      layout overlay / xaxisopts=(griddisplay=on)
                                 yaxisopts=(griddisplay=on);

                    scatterPlot X=visit y=mean
                    /YErrorUpper=eval(uppercl)
                    YErrorLower=eval(lowercl);
       endlayout;
   endgraph;
   end;
run;

proc sgrender data=final template=meanplot;
run;
```

The STATGRAPH template is named 'meanplot'. Statements within "*begingraph*" and "*endgraph*" is the key block: '*layout overlay'* defines xaxis and yaxis; *griddisply 'on*' creates grid in the background; "*YErrorUpper*" and "*ErrorLow*er" draws lower and upper confidence intervals. PROC SGRENDER identifies the dataset that contains the plot variables.

Run above codes produces Figure 1:

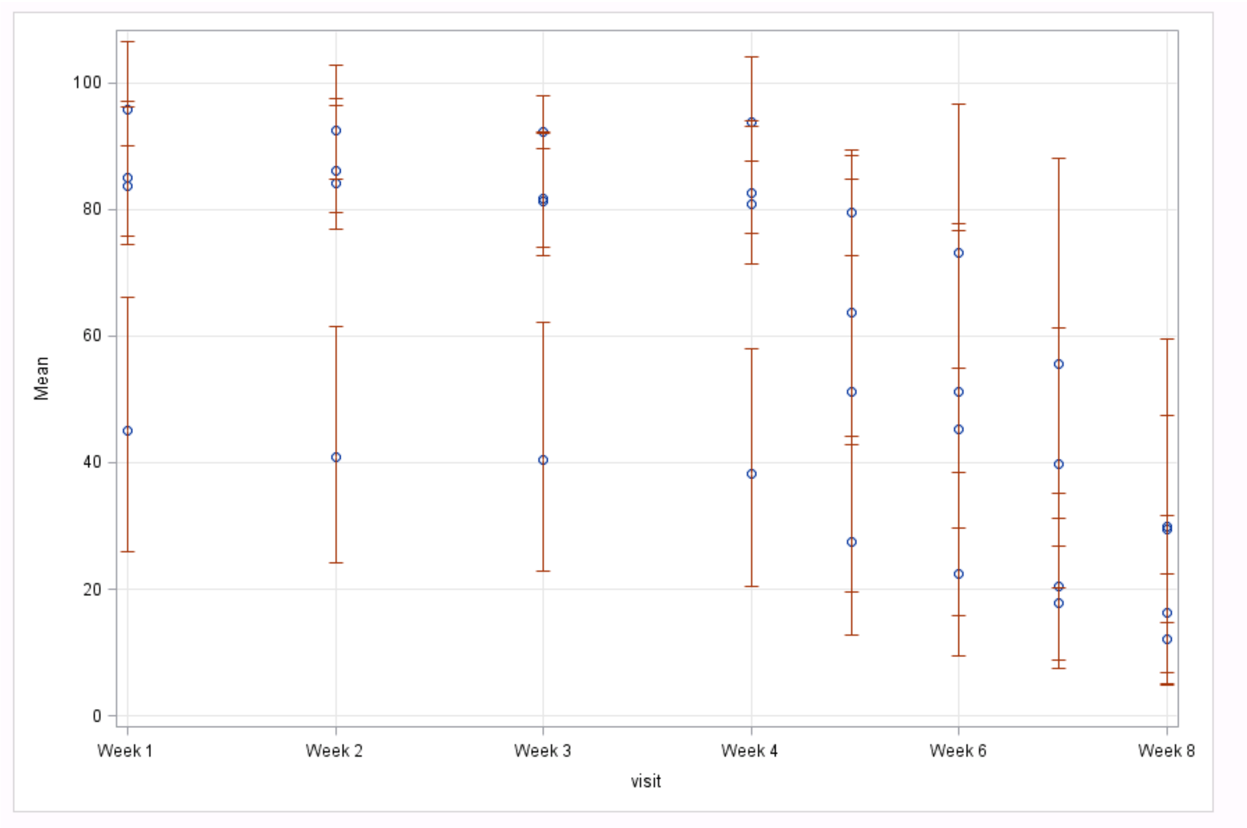


Figure 1.

STEP 4: ADD GROUP DATA

In Figure 1, the values of means and their upper and lower confidence intervals are aggregated on the same vertical lines. The group information is indiscernible. To separate group information, option *group=GROUPCD* is added.

Following is the syntax showing how to add option '*group*' after scatterplot statement:

```
   proc template;
      define statgraph meanplot;

      begingraph;
         layout overlay / xaxisopts=(griddisplay=on)
                          yaxisopts=(griddisplay=on);

                          scatterPlot X=visit y=mean
                                     /group=groupcd
                                      YErrorUpper=eval(uppercl)
                                      YErrorLower=eval(lowercl);
        endlayout;
      endgraph;
end;
run;
```

```
proc sgrender data=final template=meanplot;
run;
```

Figure 2 displays the results created by the above codes. Each color represents a group:

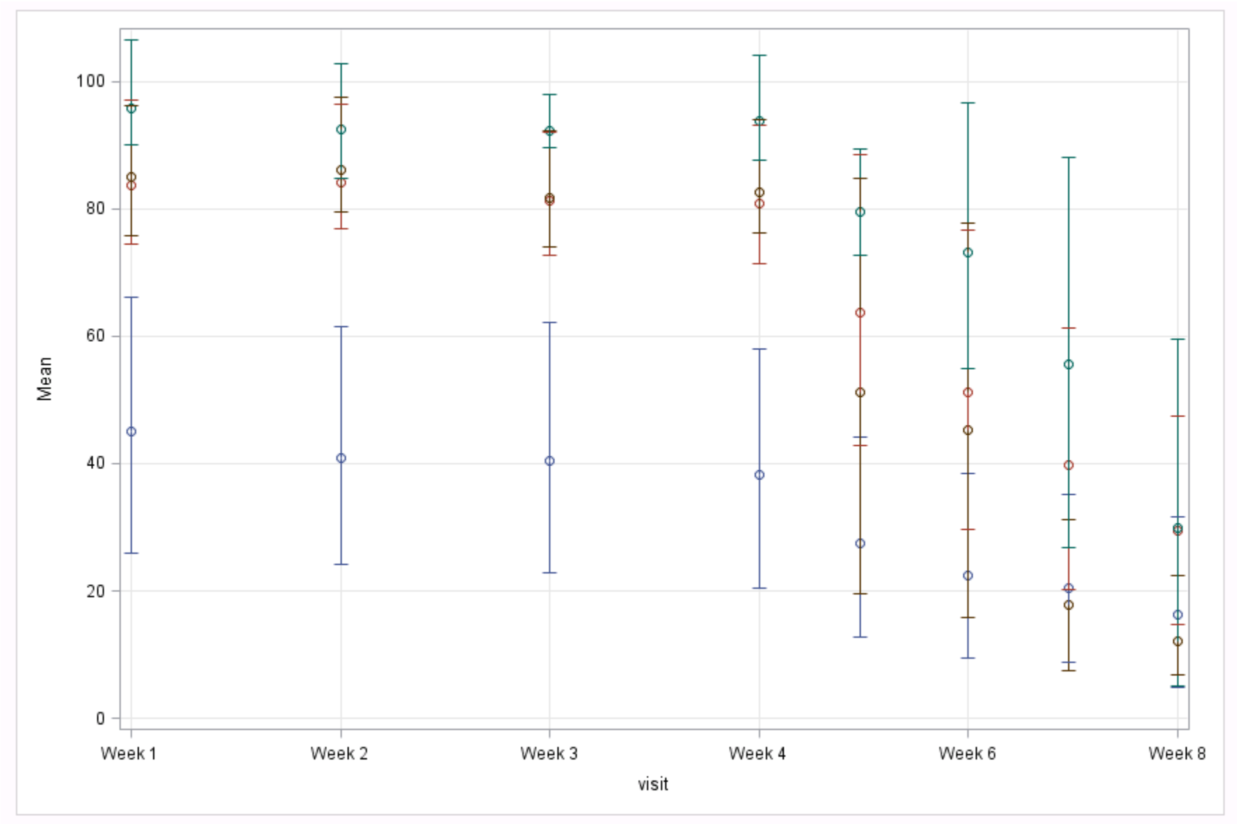


Figure 2.

STEP 5: PLACE DIFFERENT GROUPS IN ONE CLUSTER

Although the above graph has individual groups with different colors, all groups are aligned on the same lines. We need to displays means and confidence intervals for individual group with different colors to make the plot interpretable.

In the following example, *type=discrete* and *groupdisplay=cluster* options are to create four separate lines in one cluster:

```
proc template;
    define statgraph meanplot;

    begingraph;
      layout overlay / xaxisopts=(griddisplay=on type=discrete)
```

```
                                          yaxisopts=(griddisplay=on);

                    ScatterPlot X=visit y=mean
                                /group=groupcd groupdisplay=cluster
                                YErrorUpper=eval(uppercl)
                                YErrorLower=eval(lowercl);
                endlayout;
            endgraph;
        end;
        run;

proc sgrender data=final template=meanplot;
run;
```

Figure 3 displays plot with clusters containing four groups. Each visit is one cluster:

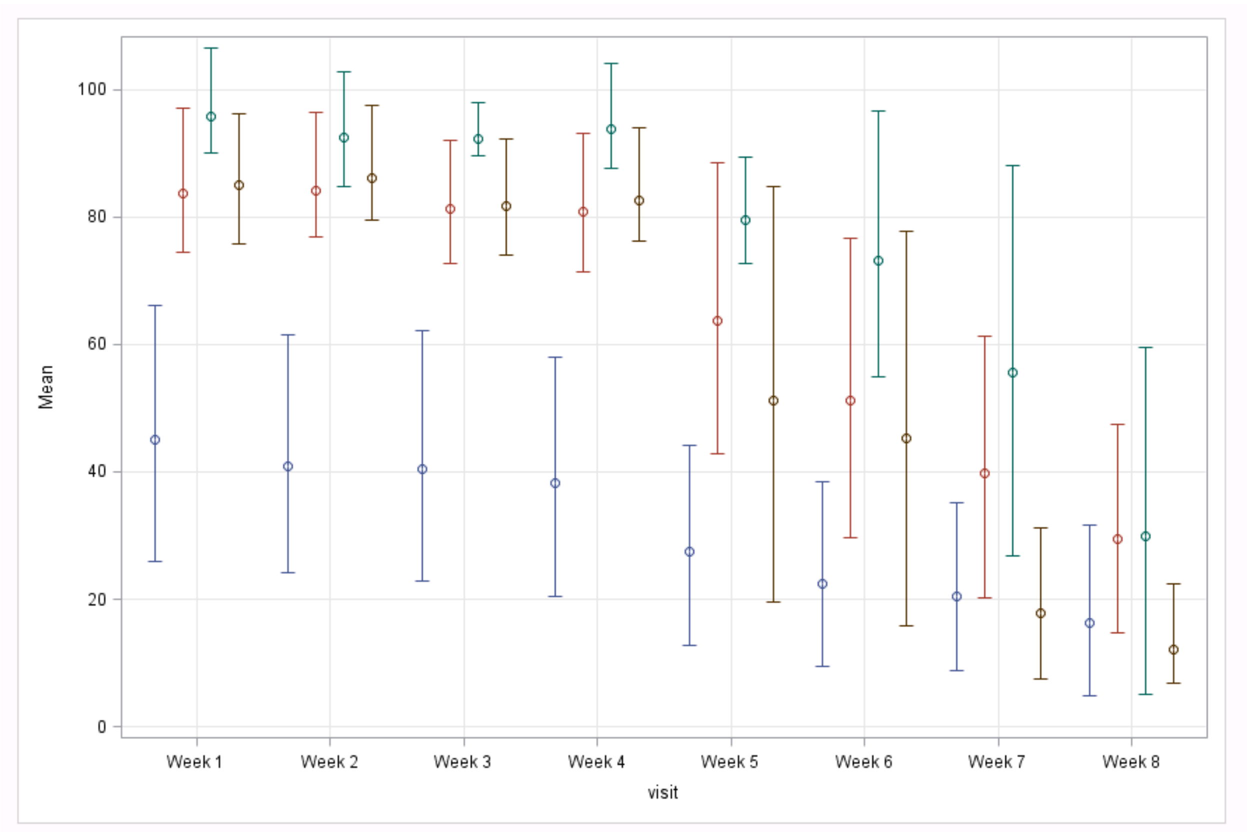


Figure 3.

STEP 6: CREATE A GAP BETWEEN EACH CLUSTER

Now each line represents one group's data, but all lines are evenly spaced. In step 4, although *type=discreate* delineates separate line for individual group, *type=discrete* also creates lines that are evenly spaced. To create gaps between groups, we use dummy datasets. Dummy datasets and variables in the dummy datasets are like virtual variables that create space in between lines thus create gaps but will not display images and values.

Figure 4 displays the plot after dummy datasets are set with the original dataset. Gaps between clusters for each visit are created:

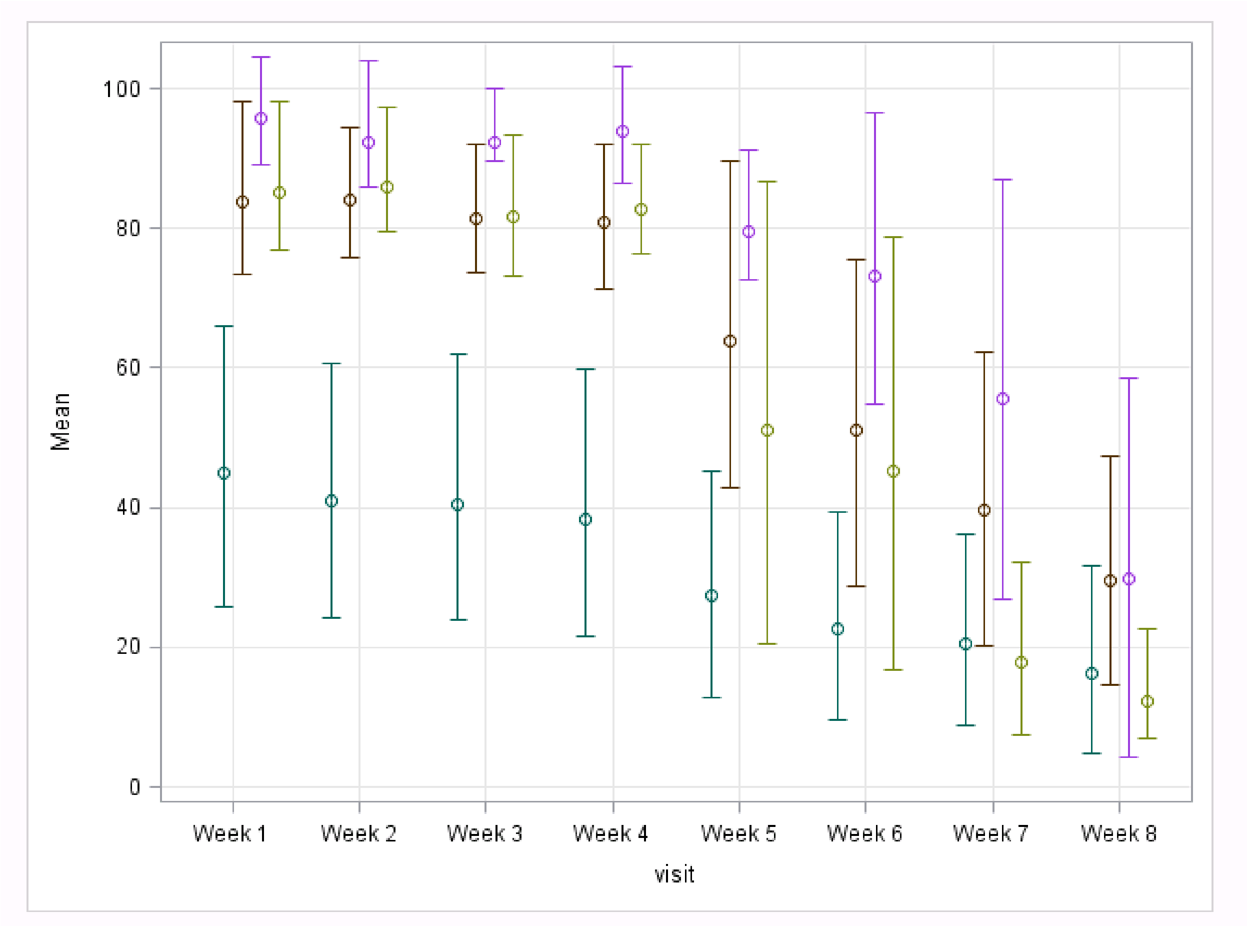


Figure 4.

Codes below created two dummy datasets:

```
data new1;
   input visit : mean : groupcd : $;
   retain lowercl . uppercl .;
   datalines;
   0 0 . Group;
run;

data new2;
   input visit : mean : groupcd : $;
   retain lowercl . uppercl .;
   datalines;
   0 0 . Group;
run;
```

STEP 7: SUPERIMPOSE A SERIES PLOT TO CONNECT DOTS FOR EACH GROUP

We want to have lines to connect the means of individual group for each week. One of GTL's most powerful feature is the flexibility to combine different types of plots together. Figure 5 presents a series plot superimposing on a scatter plots:

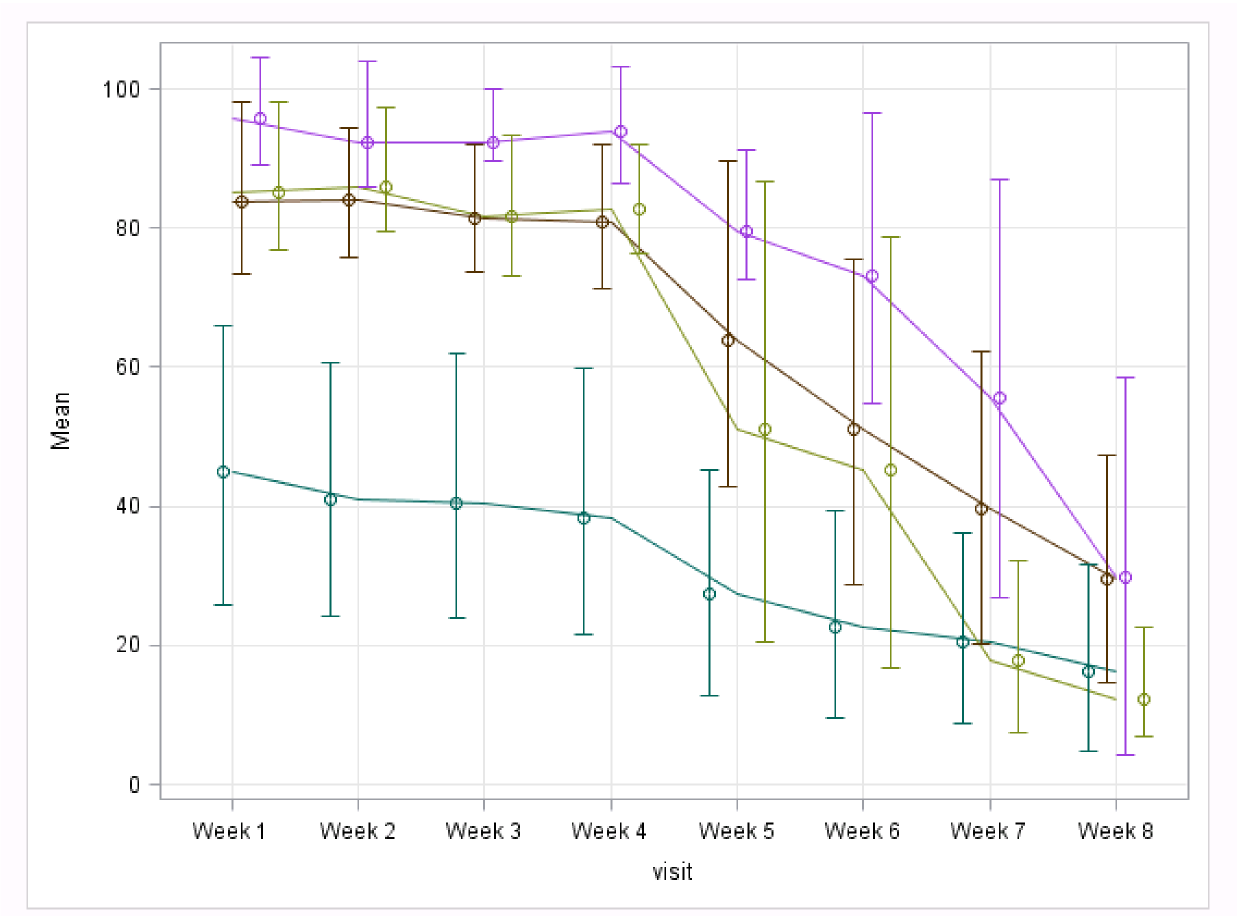


Figure 5.

To superimpose a series plot to the scatter plot, we just need to add a series plot statement:

```
proc template;
   define statgraph meanplot;
   begingraph;
      entrytitle '';
      layout overlay / xaxisopts=(griddisplay=on type=discrete)
                            yaxisopts=(griddisplay=on);
            seriesplot x=visit  y=mean/group=groupcd;
            scatterPlot X=visit y=mean/group=groupcd
                                       groupdisplay=cluster
                                       YErrorUpper=eval(uppercl)
```

```
                                                    YErrorLower=eval(lowercl);
      endlayout;
     endgraph;
   end;
run;
```

STEP 8: DISPLAY ATTRIBUTES

We have successfully built essential elements into the graph, next step is to add attributes. Attributes are important for interpretation. In ODS, attributes are intuitively defined as using options after '*xaxisopts*' and '*yaxisopts*' statements.

Followings are examples for adding attributes such as '*label*', '*labelattrs*', '*tickvalues*', and '*tickdisplaylist*':

```
   proc template;
      define statgraph meanplot;
      begingraph;
         layout overlay /xaxisopts=(griddisplay=on type=discrete
                                  label="Visit"
                                  labelattrs = (size=9pt))
                        yaxisopts=(griddisplay=on
                                  label="% Maximum"
                                  labelattrs      = (size=9pt)
                                  tickvalueattrs  = (size=9pt)
                                  linearopts=(viewmin=0 viewmax=120
                                  tickvaluelist=(0 10 20 30 40 50 60 70 80
                                                 90 100 110 120)
                                  tickdisplaylist=('0' '10' '20' '30' '40'
                                                   '50' '60'70' '80' '90'
                                                   '100' '110' '120'))
                      seriesplot  x=visit y=mean/group=groupcd
                                     lineattrs=(thickness=2) name="SP";
           ScatterPlot X=visit y=mean/group=groupcd
                                     markerattrs=(size=8)
                                     YErrorUpper=eval(uppercl)
                                     YErrorLower=eval(lowercl)
                                     groupdisplay=cluster;
           discretelegend "SP" /title="Area " exclude=('Group5' 'Group6');

      endlayout;
   endgraph;
end;
run;
```

In x-axis, the tickmark of x-axis is not specified so the tickmarks and their values are displayed by default. In y-axis, options tickvaluelist and tickdisplaylist are added to customize for individual plot. Serialplot is named "SP" which is called to draw a legend box in *'discretelegend'* statement. Dummy datasets Group 5 and Group 6 are excluded from the legend, so legend will not show Group 5 and 6.

Figure 6 is the presentation after using the attribute options:

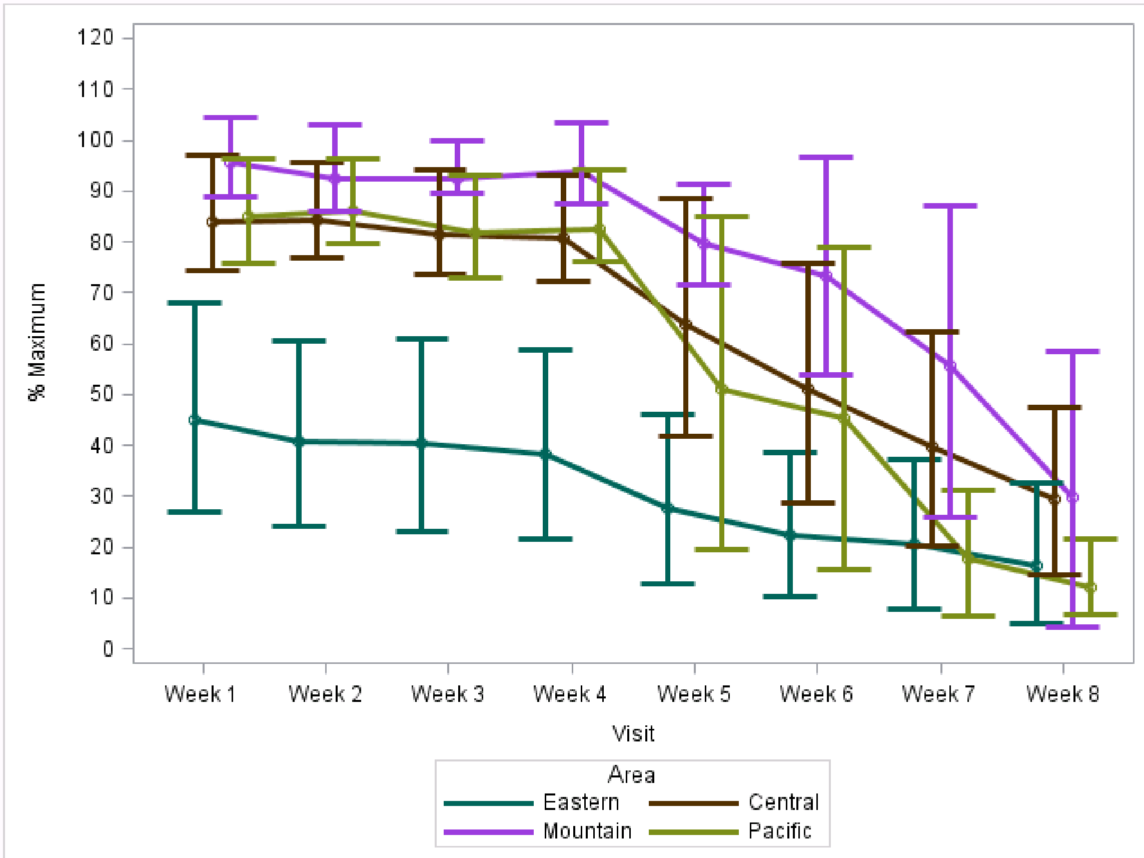


Figure 6.

STEP 9: BRING IN PROFESSIONAL QUALITY WITH STYLE TEMPLATE

Default attributes are handy and easy to use, but when it comes to customize graph, default attributes usually fall short. GTL further provides '*style template*' for the need of customizing appearance of graphs, especially for the case of group data in which further differentiation among groups are often requested. Generally, default attributes are not enough for individual group to display its own character. GTL '*style template*' is a great tool to let each individual group having its own color and style.

Following is the structure of PROC TEMPLATE. Using GraphData1 – GraphDataN. Attributes such as colors, line patterns, and marker symbols for each group can customize and differentiate nicely:

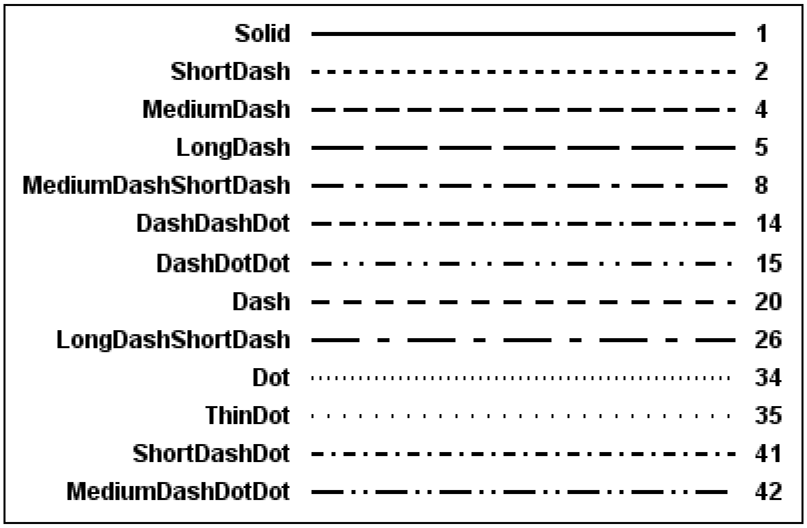
```
proc template;
   define style styles.name;
      parent = styles.default;

      style GraphData1 from GraphData1 /markersymbol =
                                         linestyle =
```

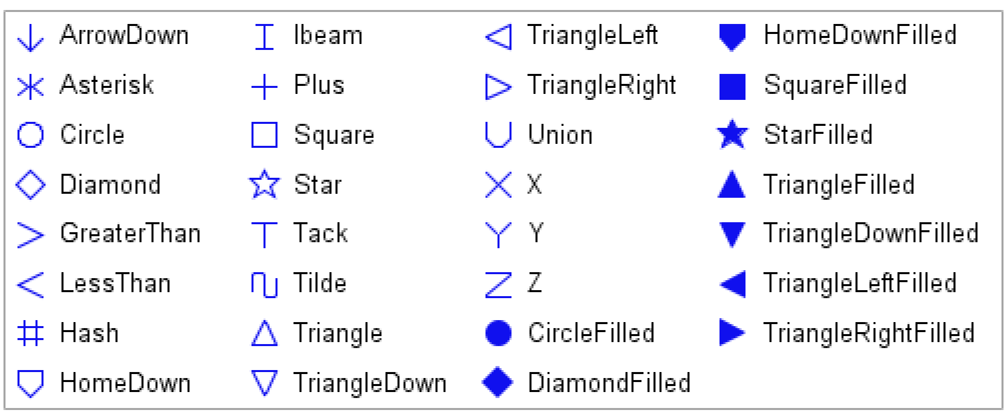
```
contrastcolor = ;
```

Followings are common line patterns and marker symbol:

Common line patterns:



Marker symbols:



Although only four groups in the graph, two additional styles are needed for the dummy groups. To apply the style to a graph, STYLE= option is used in the ODS HTML statement to specify the style name.

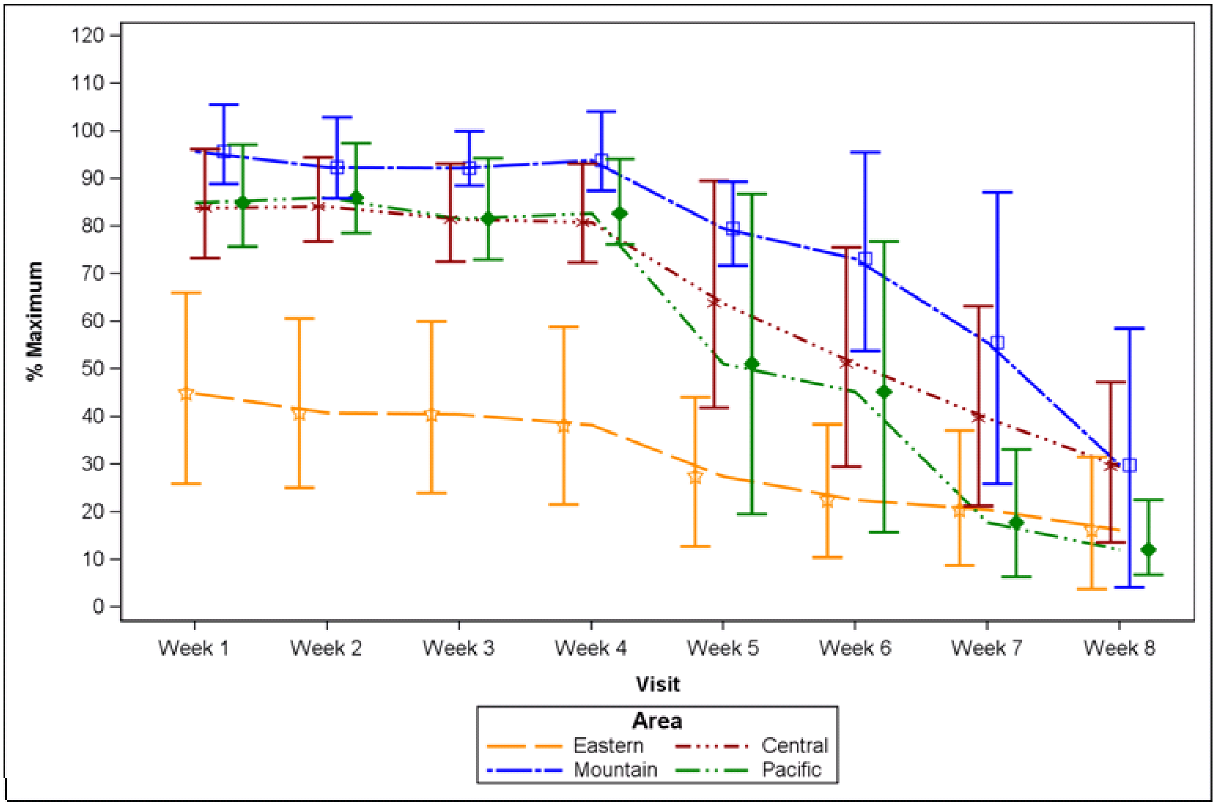


Figure 7.

Following are style template used to create Figure. 7:

```
proc template;
   define style styles.blue;
       parent = styles.default;
       style color_list from color_list
       "Abstract colors used in graph styles" / 'bgA'   = cxfcffff;

       style GraphData1 from GraphData6 /
             markersymbol = "circle"
             linestyle = 2
             contrastcolor = darkyellow;
       style GraphData2 from GraphData5 /
             markersymbol = "Circle"
             linestyle = 3
             contrastcolor = black;
       style GraphData3 from GraphData3 /
             markersymbol = "star"
             linestyle = 5
             contrastcolor = darkorange;
       style GraphData4 from GraphData4 /
             markersymbol = "asterisk"
             linestyle = 43
             contrastcolor = darkred;
       style GraphData5 from GraphData2 /
```

```
                markersymbol = "square"
                linestyle = 27
                contrastcolor = blue;
          style GraphData6 from GraphData1 /
                markersymbol = "DiamondFilled"
                linestyle = 42
                contrastcolor = green;
      end;
   run;
```

STEP 10: FINAL ADJUSTMENT, ADD TITLES AND FOOTNOTES

Final size of the output graph can also customize when bringing ODS. Titles, footnotes, size of font, type of fonts can always defined easily. In *discretelegend* statement, the appearance of legend can customize; *order=rowmajor* and *across=2* are to display legends in two rows.

The codes below create final plot Figure 8:

```
   ods graphics on / height=6.6in width=10in outputfmt=png;


   title1 h=11pt  j=c "Results" j=r "First 8 Week" font="Courier";
   title2 h=8pt  j=c "Product A" j=r "Department" font="Courier";

   footnote h=8pt j=l "From Sept 1 to Oct 29, 2016" j=r "Page 1 of 1"
   font="Courier";;

   proc template;
      define statgraph meanplot;

      begingraph;
         layout overlay /
                        xaxisopts=(griddisplay=off  type=discrete
                                  label="Visit"
                                   labelattrs = (size=9pt)
                                   );
                         yaxisopts=(griddisplay=off
                                   label="% Maximum"
                                   labelattrs      = (size=9pt)
                                   tickvalueattrs  = (size=9pt)
                                   linearopts=(viewmin=0 viewmax=120
                                   tickvaluelist=(0 10 20 30 40 50 60 70 80
                                                  90 100 110 120)
                                   tickdisplaylist=('0' '10' '20' '30' '40'
                                                    '50' '60' '70' '80'
                                                    '90' '100' '110' '120')
                                   );
         seriesplot  x=visit y=mean/group=groupcd
                                         groupdisplay=cluster
                                         lineattrs=(thickness=2) name="SP";
         scatterPlot X=visit y=mean/group=groupcd
                                         groupdisplay=cluster
                                         markerattrs=(size=8)
                                         YErrorUpper=eval(uppercl)
                                         YErrorLower=eval(lowercl)
```

```
                                                    errorbarattrs=(thickness=2);
         discretelegend "SP" /title="Area" exclude=('Group4' 'Group5')
                                          order=rowmajor across=2;
      endlayout;
      endgraph;
   end;
 run;

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

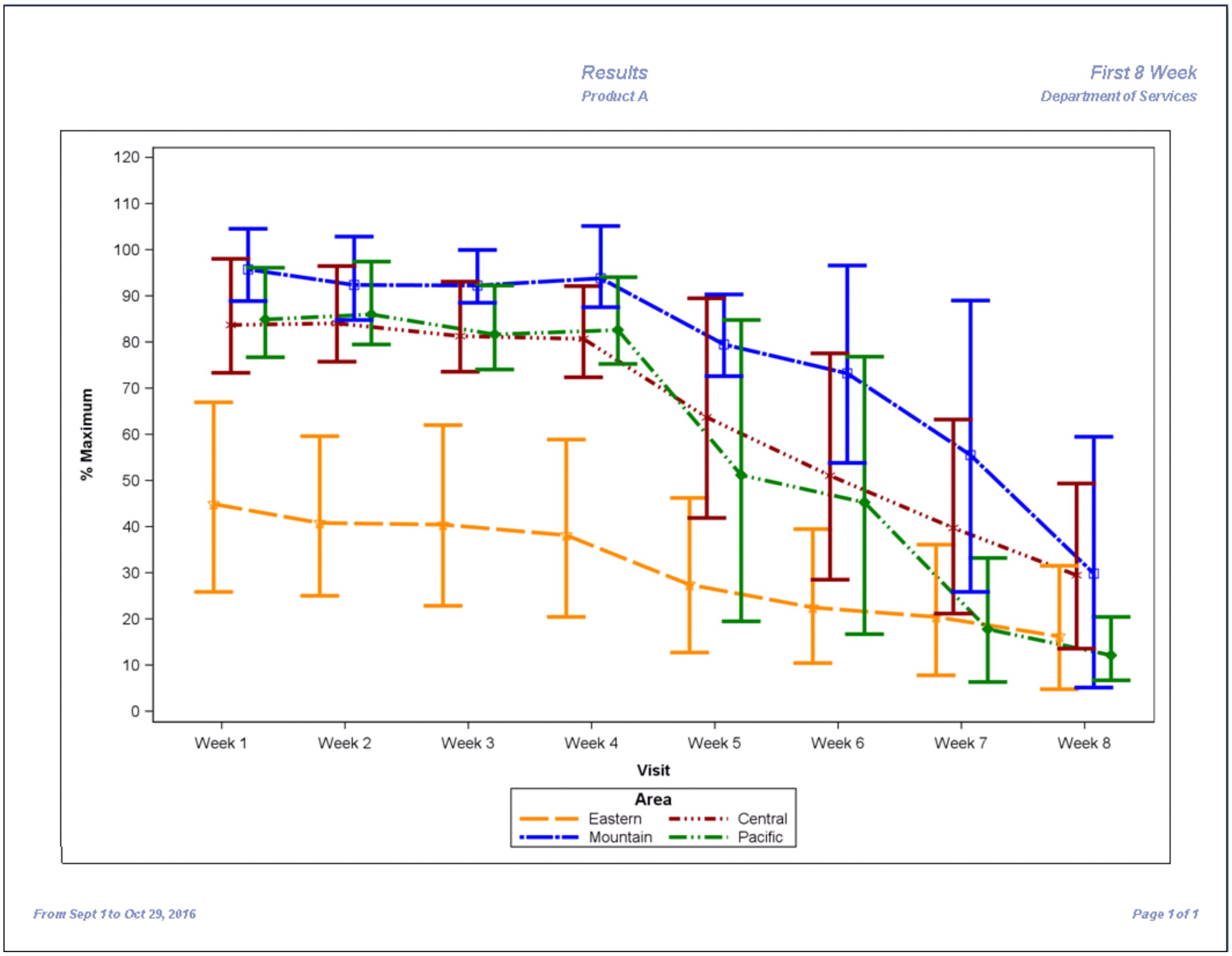


Figure 8.

CONCLUSIONS

The above steps demonstrate how simple plain patterns of plots can evolve to a highly complicated, customized graph. Understanding GTL language structures, knowing patterns of plots available in GTL are the tools to build points of references to start with; to examine data and analyze target graph are the steps to shape our thinking process. With the step by step approach, we can use GTL to create customized graph when challenge occurs.

REFERENCES

SAS Institute (2009d), SAS/GRAPH® 9.2 Statistical Graphics Procedures Guide, SAS Institute, Inc., Cary, NC.

SAS Institute (2011), SAS/GRAPH® 9.3 Graph Template Language User's Guide, SAS Institute, Inc., Matange, Sanjay. 2013. Getting Started with the Graph Template Language Graphics in SAS®: Examples, Tips and Techniques for Creating Custom Graphs. Cary, NC: SAS Institute Inc.

Cartier, Jeff, SAS Institute INC., Cary, NC. A Programmer's Introduction to the Graphics Template Language.

CONTACT INFORMATION

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

Elva Chen
Pharmacyclics, An AbbVie Company
995 E. Arques Avenue
Sunnyvale, CA 94085-4521
408-215-3100
echen@pcyc.com