Paper CT12

FUN WITH FIGURES

Miriam Amor, Veramed, Madrid, Spain

ABSTRACT

Emojis, memes, pictures, diagrams, etc: images are increasingly used in the way we communicate, and clinical trials are not the exception.

With this new trend, other programming languages like R are being used to create figures. However, when it comes to SAS, the community is facing a lack of resources with deep knowledge in graphics, specifically in the Graphic Template Language (GTL). It is not uncommon that project leads are on the hunt for skilled programmer in those SAS procedures across their teams.

Graphics and not only a powerful tool for data-driven decision making, but also an artistic way of displaying data. Though GTL can be scary at the beginning, like any art, one can easily learn and develop new technical skills with a few tips and a bit of practice.

This presentation aims to share some tips and tricks to start having fun with figures in SAS.

INTRODUCTION

From emojis and memes to infographics and dashboards, visuals are at the heart of how information is shared today, and clinical trials are no exception. Figures have evolved from occasional add-ons to essential elements in clinical outputs, now regularly featured in regulatory submissions, publications, and exploratory analyses.

In SAS, figures are typically created using two main approaches: the **SAS Graphics**, which includes procedures such as PROC SGPLOT, PROC GCHART, SGPANNEL, etc. and the more advanced and flexible **Graph Template Language (GTL)**. GTL

As figure programming becomes more frequent, the need for scalable, customizable, and expressive solutions has grown. Yet, many programmers may still be finding their way through the visual side of SAS, especially when it comes to unlocking the full potential of GTL. GTL operates in two steps: first, a template is defined using PROC TEMPLATE, and then the figure is created using PROC SGRENDER based on the created template. This structure enables the creation of highly customized, complex, and reusable graphics.

However, GTL is not without its challenges. The syntax is extensive, and remembering all the available statements and options can be overwhelming, even for experienced SAS programmers. Some have even gone so far as to call it “evil” [1], a reflection of its steep learning curve. Many programmers with years of SAS experience may still lack familiarity with GTL and PROC TEMPLATE, making figure programming feel more intimidating than it needs to be.

As the saying goes, *“Creativity is intelligence having fun.”* This paper embraces that spirit, offering six practical tips for figure creation in SAS, showing how figure programming can be not just effective, but a creative and genuinely enjoyable part of the clinical programming toolkit.

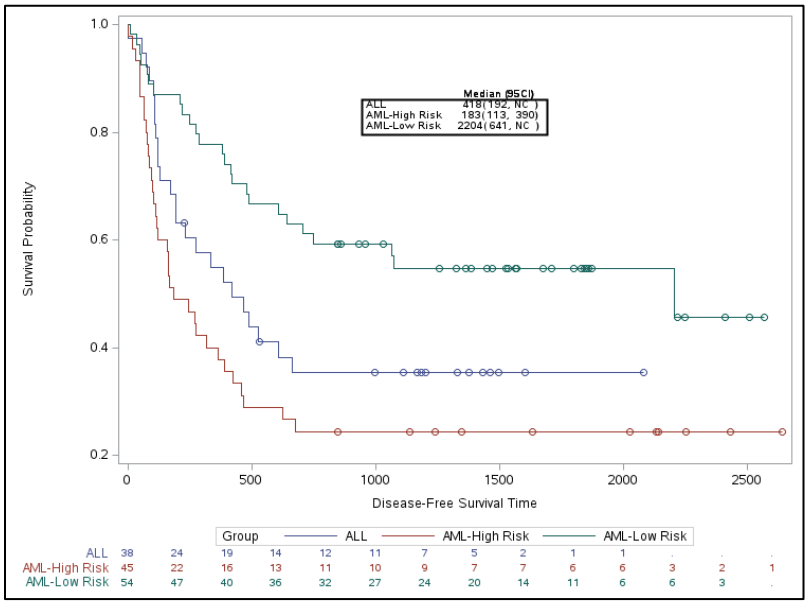
TIP 1: PLOT TWIST: FROM SAS GRAPHS TO GRAPH TEMPLATE LANGUAGE

Getting started with SAS graphics is often easiest through SG Procedures (such as PROC SGPLOT), thanks to its intuitive syntax and the abundance of examples available online. However, when more advanced customization is needed, it is time to step into the world of GLT, and that’s where things can start to feel a bit more complex.

To ease the transition, SAS offers a helpful bridge: the `TMPLOUT=` option. When included in an SG procedure, this option generates the underlying PROC TEMPLATE code that SAS uses behind the scenes. This can be a valuable learning tool, helping programmers understand how their SG procedure translates into GTL syntax.

The example below shows a Kaplan–Meier plot (**Figure 1**), based on the sashelp.BMT dataset, using `TMPLOUT=` to reveal the GTL code behind the figure.

Figure 1. Kaplan – Meier plot



This figure is initially created with PROC SGPLOT, along with an annotate dataset to display additional information (in this case the median and its Confidence Interval (95%CI). When working with annotations, a dataset stores both the content and the instructions for drawing elements on a figure. The annotate dataset contains key variables as `function`, which defines what is being drawn (e.g., text or shapes) and `label`, which contains the text to be displayed (in this example, it is the Median and its 95% CI).

This annotate dataset is integrated into the figure using the `SGANNO=<dataset name>` option. **Tables 1 and 2** display the annotate dataset and code used to create Figure 1.

Table 1. Annotate dataset used for Kaplan-Meier figure.

function	label	anchor	width	textsize	x1space	y1space	x1	y1	textweight	height	display	linecolor
text	Median (95CI)		20	7	datavalue	datavalue	1500	0.87				
text	418(92, NC)		20	7	datavalue	datavalue	1500	0.85				
text	ALL	left	20	7	datavalue	datavalue	950	0.85	bold			
text	Median (95CI)		20	7	datavalue	datavalue	1500	0.87				
text	183(113, 390)		20	7	datavalue	datavalue	1500	0.83				
text	AML-High Risk	left	20	7	datavalue	datavalue	950	0.83	bold			
text	Median (95CI)		20	7	datavalue	datavalue	1500	0.87				
text	2204(641, NC)		20	7	datavalue	datavalue	1500	0.81				
text	AML-Low Risk	left	20	7	datavalue	datavalue	950	0.81	bold			
rectangle		topleft	28		datavalue	datavalue	948	0.86		8	outline	black

Table 2. Code for Kaplan-Meier figure using PROC SGPLOT and TMPLOUT.

```
/* DATA PREPARATION */
/* Get KM results using proc lifetest */
proc lifetest data=sashelp.BMT atrisk timelist=(0 to 2600 by 200);
 time T * Status(0);
 strata Group;
 ods output  ProductLimitEstimates=ple survivalplot=survival  Quartiles=quart;

run;
/* Get time, censorship and n at risk in same dataset*/
data mydata;
 length section $200;
 set survival2(keep=stratum time survival censored in=A)
       ple( keep=group  timelist failed left numberatrisk rename=(group=stratum)
in=B)
run;

/* Run figure with PROC SGPLOT */
proc sgplot data=mydata sganno=myannot tmplout="C:\Documents\mypgm.sas";
 step    x=time y=survival/group=stratum;
 scatter x=time y=censored/group=stratum;
 xaxistable numberatrisk / x=timelist class=stratum colorgroup=stratum
run;
```

By adding the `TMPLOUT=` option to an SG procedure, SAS generates a file at the specified location containing the equivalent PROC TEMPLATE code.

While `TMPLOUT=` is a valuable tool for learning and transitioning to GTL, it should be used with caution. In the Kaplan–Meier example, as shown in **Table 3**, the generated PROC TEMPLATE code hardcodes the median values using `drawtext` functions, rather than referencing the annotation dataset shown in **Table 1**. This violates good programming practices: if the underlying data changes, the figure will not automatically update, potentially leading to inconsistencies in reporting.

This limitation can be addressed by replacing the `drawtext` statements in the generated `PROC TEMPLATE` code with the annotate option and passing the annotated dataset directly into `PROC SGRENDER` using the `SGANNO` as shown in **Table 4.**

Table 3. Code for Kaplan-Meier figure created with TMPLOUT option

```
proc template;
define statgraph sgplot;
begingraph / collation=binary;
layout lattice / columnweights=preferred rowweights=preferred
                 columndatarange=union rowdatarange=union columns=1;
   layout overlay;
      StepPlot X=TIME Y=SURVIVAL / primary=true Group=STRATUM
      LegendLabel="Survival Probability" NAME="STEP";
      ScatterPlot X=TIME Y=CENSORED / subpixel=off Group=STRATUM
      LegendLabel="CENSORED" NAME="SCATTER";
     DiscreteLegend "STEP"/ title="GROUP";
     DrawText  TEXTATTRS=( SIZE=7)  "Median (95CI%)" /
                          X=1500 Y=0.9 XSPACE=datavalue YSPACE=datavalue
                   WIDTH=20;
     DrawText  TEXTATTRS=( SIZE=7)  " 418 ( 192, NC  )" /
                           X=1500 Y=0.85 XSPACE=datavalue YSPACE=datavalue
                   WIDTH=20;
     DrawText  TEXTATTRS=( SIZE=7 WEIGHT=bold)  "ALL" /

                          X=950 Y=0.85 XSPACE=datavalue YSPACE=datavalue
                   ANCHOR=left WIDTH=20;
…
      …..
   endlayout;
Layout Overlay /  walldisplay=none xaxisopts=(display=none griddisplay=off
                    displaySecondary=none)  x2axisopts=(display=none
griddisplay=off displaySecondary=none);
                     AxisTable Value=NUMBERATRISK X=TIMELIST / labelPosition=min
Class=STRATUM ValueAttrs=( Weight=bold )
       colorGroup=STRATUM Display=(Label);
```

Hard-coding

Table 4. Code for Kaplan-Meier figure using GTL with annotate dataset.

```
proc template;
define statgraph sgplot;
begingraph / collation=binary;
layout lattice / columnweights=preferred rowweights=preferred columndatarange=union
rowdatarange=union columns=1;
layout overlay;
  StepPlot X='Time'n Y='Survival'n / primary=true Group='Stratum'n
 LegendLabel="Survival Probability" NAME="STEP";
  ScatterPlot X='Time'n Y='Censored'n / subpixel=off Group='Stratum
 LegendLabel="Censored" NAME="SCATTER";
  annotate;
  DiscreteLegend "STEP"/ title="Group";
endlayout;
Layout Overlay /  walldisplay=none xaxisopts=(display=none
   griddisplay=off displaySecondary=none) x2axisopts=(display=none griddisplay=off
   displaySecondary=none);
    AxisTable Value=NumberAtRisk X=Timelist / labelPosition=min Class=Stratum
    ValueAttrs=( Weight=bold ) colorGroup=Stratum Display=(Label);
 endlayout;
endgraph;
end;
run;

proc sgrender data=mydata template=sgplot sganno=myannot;
run;
```

Creates a grid with 1 column and as many rows as cells defined

First row: displays the plot itself with the annotations

Second row: displays the axis table with no. of subjects at risk

There are other restrictions for `TMPLOUT` option, as the `GTL` output created in `PROC SGPANEL` is not always correct, and SAS 9.3 has discontinued supporting this option for SGPANEL onwards for just this reason [2]. Additionally, the indexing of the generated code is not great, making it harder to navigate and maintain.

Therefore, while `TMPLOUT=` can be a helpful shortcut (whether as a starting point for writing `PROC TEMPLATE` code or for identifying the GTL equivalent of an SG option), it should never be used blindly. A thorough review and, when necessary, manual updates to the generated code are strongly recommended before using it in production environments.

TIP 2: FIGURES IN STYLE: THE ATTRIBUTES MAP

Consistency in styling is key when creating multiple figures, where visual clarity and reproducibility matter. Whether it is ensuring that treatment groups are always represented with the same colors or that markers and line styles remain consistent across outputs, attribute maps are the go-to solution. There are two types of attributes map:

- Discrete: maps discrete values
- Range: maps numeric values or ranges of values.

The example below, extracted from Blum and Duggins [3], demonstrates how attributes is used to keep consistency across different categories of a discrete variable.

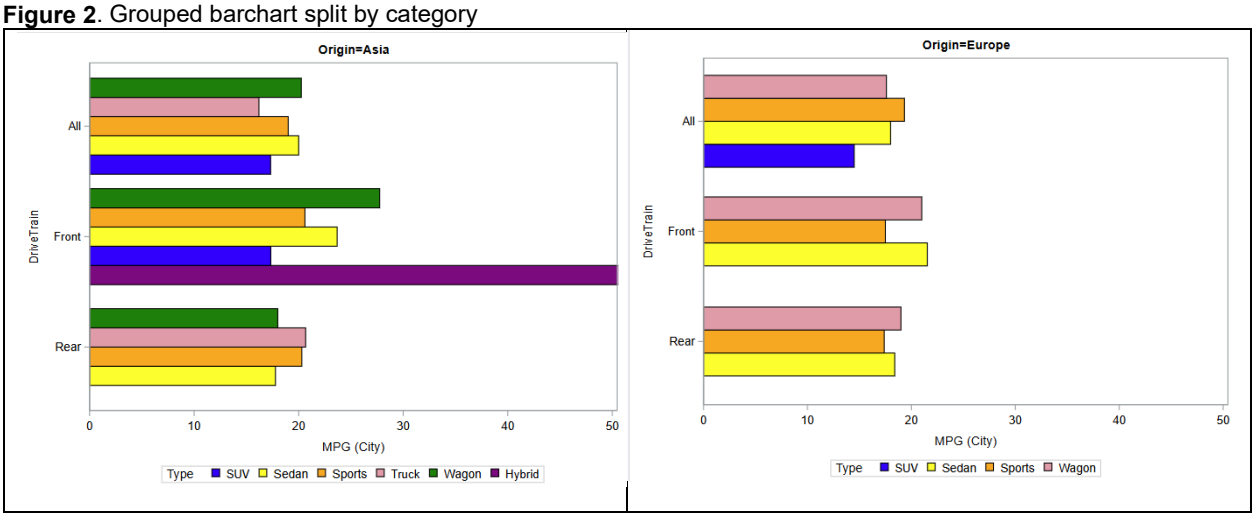
Table 5: Code to create a grouped barchart split by category

```
proc sort data=sashelp.cars out=carSort;
  by origin;
run;

proc sgplot data=carSort;
  styleattrs datacolors=(blue yellow orange lipk green purple);
  by origin;
  hbar drivetrain / group=type groupdisplay=cluster
  response=mpg_city stat=mean;
  where origin in ('Asia' 'Europe');
run;
```

In the code, colours are assigned using the `datacolors` option. However, as shown in **Figure 2**, this approach leads to inconsistencies:

- Europe lacks some vehicle categories present in Asia.
- The same category (e.g., *Wagon*) appears in different colours across panels.
- The legend order varies depending on the data, making comparisons harder.

Figure 2. Grouped barchart split by category



To use attributes map, a dataset is created with the attributes for each category and passed to the SG procedure using the dattrmap=<dataset name> option. Each row specifies styling for a category, using required variables like:

- ID: the attribute map identifier,
- VALUE: the group or category value (its content should exactly match with the value in the data)

Optional variables allow further customization:

- SHOW: Specifies whether the legend displays only the attribute map values that appear in the data (DATA) or always displays all of the values in the attribute map (ATTRMAP).
- NOCASE: Value is case-sensitive (TRUE OR FALSE)
- Styling options: *LineColor, FillColor, MarkerSymbol, TextColor* and more (full list in SAS Help Center documentation) [4].

When working with GTL, attribute maps can be applied in two ways: by referencing a dataset using the `dattrmap=` option in `PROC SGRENDER`, or by defining them directly within `PROC TEMPLATE` using the `DISCRETEATTRVAR` block.

The same example is reworked using an attribute map to ensure consistent styling. **Table 6** shows the attribute dataset and its use in PROC SGPLOT, while **Table 7**displays the corresponding GTL code.

Table 6. Code for grouped bar chart split by category using attributes map. SG procedure.

Attributes Dataset

id	value	fillcolor	show
map1	Hybrid	blue	ATTRMAP
map1	SUV	yellow	ATTRMAP
map1	Sedan	orange	ATTRMAP
map1	Sports	lipk	ATTRMAP
map1	Truck	green	ATTRMAP
map1	Wagon	purple	ATTRMAP

SG code

```
proc sgplot data=carSort dattrmap=myattrib;
  by origin;
  where origin in ('Asia' 'Europe');
  hbar drivetrain / group=type
groupdisplay=cluster   response=mpg_city stat=mean
attrid=map1;
run;
```

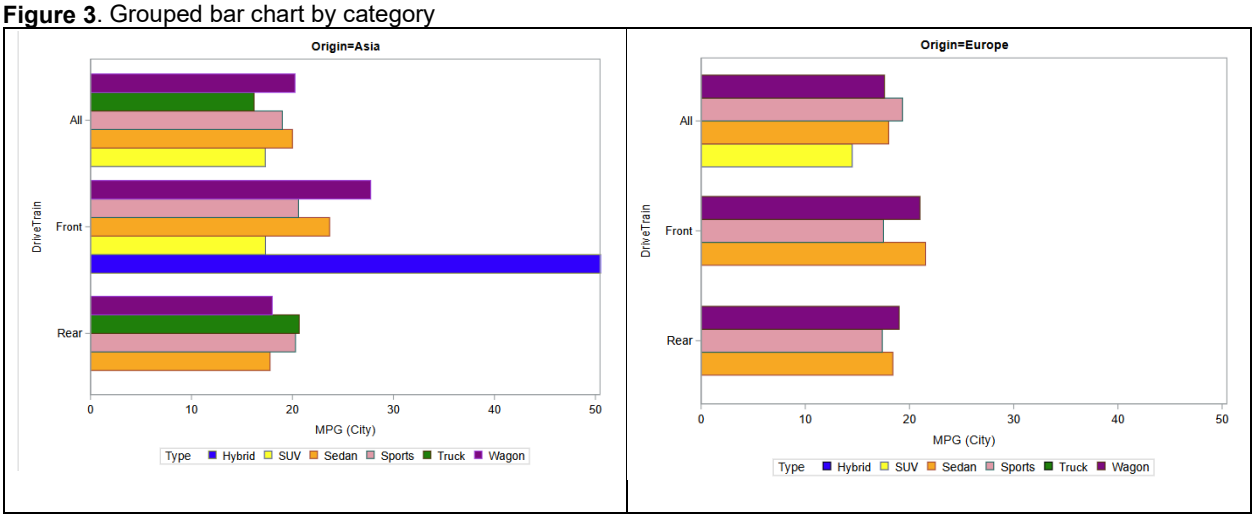
Table 7. Code for grouped bar chart split by category using attributes map. GTL.

```
proc template;
define statgraph mysgplot;

begingraph / collation=binary;
   DiscreteAttrVar attrvar=MAP1_TYPE var=TYPE attrmap="_ATTRMAP_MAP1"; /*[1]*/
    DiscreteAttrMap name="_ATTRMAP_MAP1" / discreteLegendEntryPolicy=attrmap/*[2]*/;
       Value "Hybrid" / fillattrs=(color=blue);
       Value "SUV" / fillattrs=(color=yellow);
       Value "Sedan" / fillattrs=(color=orange);
       Value "Sports" / fillattrs=( color=lipk);
       Value "Truck" / fillattrs=( color=green);
       Value "Wagon" / fillattrs=( color=purple);
 EndDiscreteAttrMap;

layout overlay / yaxisopts=(type=Discrete reverse=true);
   BarChartParm X=DriveTrain Y=MPG_City /  orient=horizontal Group=MAP1_TYPE /*[1]*/

   LegendLabel="MPG (City)" NAME="HBAR" groupdisplay=cluster;
   DiscreteLegend "HBAR"/ title="Type";
endlayout;
endgraph;
end;
run;

proc sgrender data=carSort template=mysgplot;
  by origin;
run;
```

[1]: `var=`: Specifies the variable to apply the styling. `Attrmap=`name of the attributes map. `Attrvar=` new internal variable that carries the styling, this is the variable used in the figure.
[2]: The legend displays all the values in the attributes map (similar to show='ATTRMAP').

Figure 3 displays the final plot generated using both the attribute map dataset and the GTL code shown in **Tables 7 and 8.**

Figure 3. Grouped bar chart by category



Range attribute maps work similarly to discrete ones but are designed for continuous variables. Instead of using a VALUE column, they define ranges using MIN and MAX. These maps are applied using the `RATTRMAP=` option in SG procedures, or the `RANGEATTRMAP` block within PROC TEMPLATE.

TIP 3. MIXING IT UP — CREATING TWO FIGURES IN ONE PANEL

While PROC SGPANEL is useful for displaying repeated versions of the same plot type across panels, it does not support mixing different kinds of plots within the same figure. This is achieved by defining multiple layout blocks, each corresponding to a different panel. The `layout lattice` or `layout gridded` structures serve as containers, and each panel is built using its own `layout overlay`.

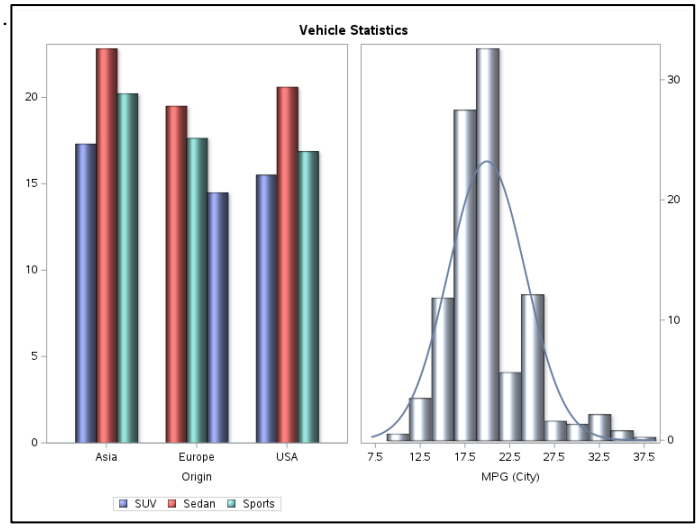
Following an example from the SAS Help Center documentation [4], this technique involves adding as many layout blocks as panels are needed.

Table 8. Code to create a figure by panels with mixed layout

```
proc template;
   define statgraph pannels;
      begingraph;
         entrytitle "Vehicle Statistics";
         layout lattice / columns=2 columngutter=10; /*[1]*/

            /*1st column, displays a barchart*/
            layout overlay /
               xaxisopts=(offsetmin=0.2 offsetmax=0.2)
               yaxisopts=(display=(ticks tickvalues));
               barchart x=origin y=mpg_city / name="bar" stat=mean
                  group=type groupdisplay=cluster clusterwidth=0.7
                  dataskin=sheen;
               discretelegend "bar";
            endlayout;

             /*2nd column, displays a histogram together with a density plot*/
            layout overlay /
               y2axisopts=(display=(ticks tickvalues));
               histogram mpg_city / dataskin=sheen yaxis=y2;
               densityplot mpg_city / yaxis=y2;
               annotate / id="HIST";
            endlayout;
         endlayout;
      endgraph;
   end;
run;
```

[1]: Defines the grid arranged in 2 columns, with the space between the columns set to 10 pixels.

Figure 4. Figure by panels with mixed plot types



TIP 4: DYNAMIC THINKING: ONE TEMPLATE, MANY FIGURES

Dynamic variables in GTL make templates adaptable and reusable. By declaring dynamic variables, conditions on the template can be included.

Two commonly used dynamic variables are BYVAR and BYVAL, which are especially useful when generating plots by groups using a BY statement.

- **BYVAR** refers to the name of the variable used in the BY statement.
- **BYVAL** refers to the current value of that variable.

In this example, dynamic variables are used to generate individual figures for each laboratory parameter. Since each parameter has its own scale, the y-axis is customized accordingly: for instance, the axis for 'Calcium' is set to a maximum of 3 mmol/L, while other parameters go up to 400.

Additionally, the x-axis is broken to display two intervals. This breaking axis is useful when there are outliers (as shown in Figure 5), different timepoints, and even for graphics by subgroup. This way of breaking axis works in PROC SGPLOT, but not for PROC SGPANNEL. That is one of the reasons why PROC TEMPLATE is often preferred for more flexible and customized graphics.

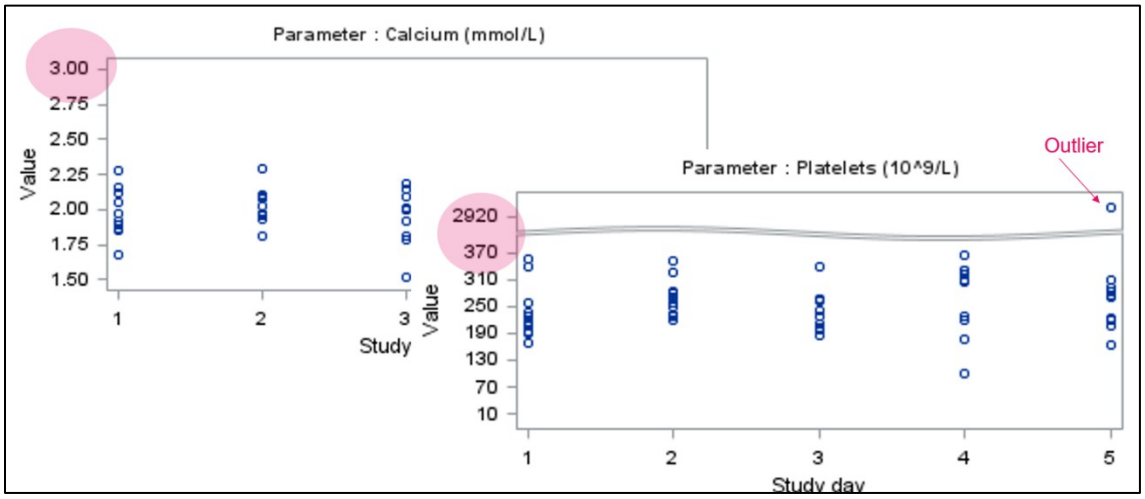
Table 9. Code with dynamic variables

```
proc template;
   define statgraph mytemplate;
   begingraph;
   dynamic _byval_ _byvar_ _byval2_ _byvar2_; /*1*/
   entrytitle "Parameter : "  _byval2_ ;

   /*Conditional template: layout for param=Calcium*/
   if ( _byval2_ = 'Calcium (mmol/L)')
      layout overlay/
          yaxisopts=(label='Value' linearopts=(viewmax=3)) ;
          scatterplot x=ady y=aval;
       endlayout;

   /*Conditional template: layout for other params*/
   else
     layout overlay/
       yaxisopts=(label='Value' linearopts=(includeranges=(1-400 2900-2950))); /*2*/
         scatterplot x=ady y=aval ;
      endlayout;
     endif;  /*End of the condition*/
endgraph;
end;
run;

options nobyline;
proc sgrender template=mytemplate data=mydata;
  by paramn param; /*1*/
run;
```

1: Dynamic variables definition: _byvar_ and _byvar2_ correspond to paramn and param, respectively and _byval_ and _byval2_ to its content.
2: Option to break the axis and specify the upper limit.

The figure obtained will display plots for both parameters, but using different ranges in the y-axis:

Figure 5. Dynamic variables



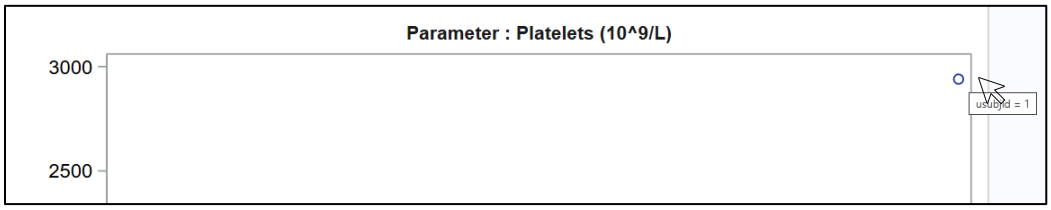
TIP 5: A TIP ON TIP OPTION: HOVER TO DISCOVER

Sometimes the best tip is indeed the `TIP= option`. This feature enhances interactive HTML plots by displaying contextual information when hovering over graphical elements, provided `ods graphics/ imagemap=on` is enabled. It's a simple yet way to surface details that are not directly shown in the figure.
For example, when investigating outliers, tooltips can reveal the subject ID associated with a data point. In the example adapted from Tip 3, the `tip=` option is added to the GTL code, specifying the subject ID to be displayed using the `rolename` option. However, this interactive feature has its limitations. When used in combination with certain axis features like `includeranges`, tooltips may not function correctly.

Table 10. Example code using TIP option with GTL.

```
proc template;
[…]
else
  layout overlay/
   yaxisopts=( display=(line ticks tickvalues label)  label='Value'
   /*linearopts=(includeranges=(1-400  2900-2950)*/);
   scatterplot x=ady y=aval /rolename=(tipvar1= usubjid) tip=(tipvar1);
  endlayout;
endif;
[…]
```

Figure 6. HTML Plot Showing Value on Hover Using TIP=



TIP 6: LINKING THE DOTS: FROM MATH TO ART

The POLYGON statement in SAS allows transforming mathematical data into visual art by drawing one or more closed shapes based on coordinates stored in a dataset. Each shape polygon is formed by a series of connected lines, and distinguished using an ID variable that groups the relevant data points.

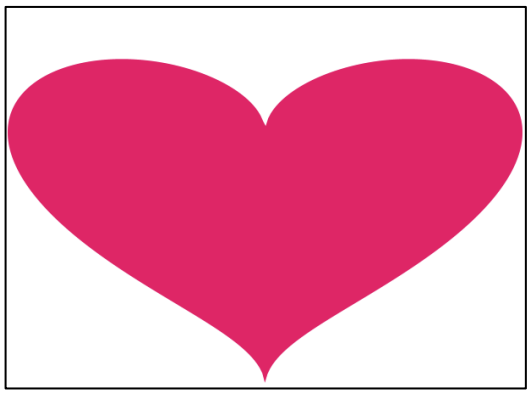
In this example, a set of datapoints are derived based on the function:

$$\begin{cases} x = \cos(t) \\ y = \sin(t) + \sqrt{|\cos(t)|} \end{cases} t \in [-\pi, \pi], x \in [-1, 1],\ y \in [-1, 2]$$

Given a function, a dataset of coordinates is created, and a lovely figure is obtained by using the `polygon` statement.

Table 11. Code to display a set of data points using polygon

```
/*Derive data points x1 and y1*/
 data mydata;
  id='pol1';
  do t=-3.1416 to 3.1416 by 0.01;
     x1=cos(t);
     y1=sin(t) + sqrt(abs(cos(t)));
     output;
  end;
  run;

/*Create the figure using polygon option */
proc sgplot data=mydata;
polygon x =x1 y=y1 id=id/ fill
fillattrs=(color=CXE70E64);
xaxis values=(-1 to 1) display=none;
yaxis values=(-1 to 2) display=none;
run;
```



CONCLUSION

Programming figures in SAS doesn't have to be intimidating, on the contrary, it can be a creative and rewarding process when one knows the right tools and techniques.

These six tips show that figure programming in SAS can be more than just functional: it can be flexible, creative, and even enjoyable. From unlocking GTL code with TMPLOUT=, styling with attribute maps, building dynamic templates, and plotting mathematical functions, to adding interactivity with TIP=, each technique adds clarity and character to your visuals.

With the right tools, turning data into decisions becomes not only effective, but fun.

REFERENCES

[1] Delaney K. Pure Evirl, or Just MDelaney K. Pure Evil, or Just Misunderstood? An Interview with PROC TEMPLATE
[2] When Do I Use SG Procedures vs. Graph Template Language? Q&A, Slides,... - SAS Support Communities
[3] Blum J, Duggins J. Using attribute maps in SAS graphs.
[4] SAS Institute Inc. Annotate Statement. SAS Help Center.
[5] SAS Institute. Inc. Specifying the Attribute Mapping Information in a SAS Data Set. SAS Help Center.
[6] D. Shen, L. Zhang, B. Adeyi, D. Zhang. SAS Can Automatically Provide GTL Templates for Graphics in Three Ways.
[7] SAS Institute Inc. SAS 9.4 Graph Template Language: User's Guide, Fifth Edition

ACKNOWLEDGMENTS

I would like to thank Tom Ratford for his careful and thorough review of this paper, and for his valuable suggestions that greatly improved its clarity and visual presentation.

CONTACT INFORMATION

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

Author Name: Miriam Amor
Company: Veramed GmBH
Address: Registered Office: 5th Floor Regal House, 70 London Road, Twickenham TW1 3QS,
Work Phone: NA
Email: Miriam.amor@veramed.co.uk
Website: https://www.veramed.co.uk