PhUSE 2018

Paper DV01

Turning Fantasy into Reality - Creating Custom Data Visualisations for Future Reporting

James Diserens, Veramed, Twickenham, UK

ABSTRACT

Recent experience indicates that there is a gap between exploratory analysis using products such as R-shiny and Spotfire with html outputs, and SAS rtf outputs for reporting packages.

Are the plots that we are producing restricted by standards and expectations, or by an unwillingness to change from what has been done before? Could we be producing outputs that are more useful and appropriate for analysing the data? In addition, with the improvements in SAS ODS graphics from SAS 9.4, is there scope for future reporting packages to use further data visualisations for study overview?

This paper looks at examples of figures suggested by experienced statisticians that could provide useful data visualisation in a reporting package.

INTRODUCTION

Data visualisations or graphical displays are a highly effective way of displaying large amounts of data and effectively communicating results and therefore should be ideal for presenting results of clinical trials. Their purpose is to represent numerical results in a visual manner that is more intuitively understood by the human brain. Increasingly, figures are being included to communicate key information in clinical study reports (CSR) and for submission to regulatory authorities, but these remain a supplement to the required tables and listings. With little guidance on what is required from a figure to be used to present the results of a clinical trial, figures are under used for reporting clinical trial data.

Typically, and understandably, Statisticians are conservative when preparing Statistical Analysis Plans (SAP) and shells for TFLs used in the CSR. As a result, the planned outputs often include only Tables and Figures that have been produced before for previous studies where it is understood how they can be produced, rather than producing the figures best suited to representing the data. In contrast, exploratory analysis and post-hoc research are becoming increasingly innovative in the way that data visualizations are used. In my opinion there are two apparent reasons behind this contrast. Firstly, exploratory analysis can be performed with a far greater range of software packages. Products such as R-shiny and Spotfire are designed with the aim of producing interactive graphics that allow users to drill down, giving greater freedom to investigate the data. Secondly, having planned the outputs in advance of programming, when writing the SAP and shells, the figures are restricted and may not be able to be adapted to best represent the data.

The use of data visualisations has been improving in recent times but there are still opportunities to better utilize the functionality of SAS graphics. For example, in recent years, Forest Plots have become a “gold standard” of how figures are able to present data spatially. The advantage of displaying data in this way is that a large cross-section of data can be reported together on a single page, which enables a reviewer to compare the results of analyses across various endpoints or studies. The improvement in SAS graphics for producing these plots has enabled greater flexibility and a wider range of possibilities but these are not being utilized for study reporting. This paper will now look at some opportunities for improved reporting visualisations.

SAS GRAPHICS

“R is better than SAS for graphs” is a not an unusual statement. Despite pharmaceutical programming being heavily dominated by SAS, R is widely regarded as superior when it comes to producing graphics. But how true is it? Historically before version 9 and ODS, SAS graphics were restrictive and hard to use, whilst R has a reputation for quality graphics that are easy to produce. R also has the advantage of being free and open source software meaning that users could readily have access to new software advances and packages, written by other users. SAS users in

contrast must wait for the next release of software which they then must also purchase. As a result, many programmers shifted to using R for producing graphics. It might therefore be more correct to say that “R is simpler to use for producing graphs than SAS”. However, if we can understand the possibilities of SAS with the latest version, we can prepare interesting and useful data visualisations for study reporting to match the graphics that are becoming common elsewhere.

In the latest version of SAS, the standard graphical procedures such as SGPLOT and SGPANEL are used for the majority of outputs but they do have limitations. Graph Template Language (GTL) (introduced in SAS 9.2) underlies these procedures, specifying the layout and details of each graph produced by ODS. The ability to create custom templates allows the user more flexibility in creating customized graphics and overcomes many of the issues or shortcomings that occur when producing graphs in SAS.

While in an ideal world everyone would have a good understanding of both languages, this is not necessarily realistic, and many programmers do not have the best understanding of the latest advances in SAS graphics. If programmers have become reliant on R for producing graphics, they may not be best placed to advise statisticians on what is possible in SAS. We therefore suggest that statisticians and programmers work together outside of studies to develop graphics that can then be used in future reporting. Simple prepared macro code can then be made available to produce these kinds of plots. To best use the advances of SAS graphics with the introduction of SGPLOT, we should keep aiming to produce graphics in SAS as well as R.

STUDY REPORTING

Most graphs that appear in CSRs consist of the traditional line graphs, histograms, and bar charts for single endpoints. Graphing techniques have evolved to make data and results easier to read and interpret. In recent times, this has seen a greater range of graphics become used in study reporting: it is now not unusual to see modelled data presented in graphs, such as Kaplan-Meier plots for time to event data or line graphs for pharmacokinetic data.

A prime example of the evolution in the way that we use graphs to represent data in study reporting is the increased use of Forest Plots. As stated above, this style of representing data allows the reviewer to compare the results of multiple analyses on one page. This is a fantastic way to provide an overview of the analyses performed and the spatial nature of the figure makes any differences clear.

Understanding why Forest plots are a useful tool for study reports gives us a good base for developing future graphics that we want to include in a CSR and why they could be useful. It is very easy to just stick to producing a CSR, based on outputs that we have seen before, or to a standard set of shells. Instead we should constantly be thinking about how we can use graphs to more usefully or more appropriately present the data that is being reported. It is much harder to design new output styles without being able to visualize the data, so review of previous studies is very important in this process. To ensure the quality of future reporting we need to be aware of how we could have presented data better previously.

I spoke to several senior statisticians, that I have worked with at Veramed, to understand the kind of graphs that they felt could be used to better display the data, and to allow them to be included in analysis plans and shells for CSRs. With these ideas I developed SAS code into macros that can be included in a graphics catalogue so that anyone can use them for future studies. In this paper I will present the reasoning for including a graph, the macro code developed and example outputs for 3 cases using simplified dummy data:

1) Embedded plots for PK line graphs.
2) Lattice scatter plots for multiple tests.
3) Swimmer plots for a timeline of subject events.

EMBEDDED LINE GRAPHS

Line Graphs are the most common of data visualisations. However, that doesn't mean that they are being used as effectively in reporting as they can be. It is very easy to think that because something has always been done a certain way that there is not a way to improve it. Looking at the example of the forest plot, a display could be improved by simultaneously showing a large cross section of data in one place while also reducing the clutter of a table to the important elements. If required, the reviewer can then look at the tables for further detail.

Concentration v Time line graphs are a standard for reporting PK data. The main issue with reporting PK data in this way, is that there is often a large variation in the concentration levels over the reporting time. As a result, a large percentage of the data is displayed in a very condensed area of the figure, which makes it very difficult to distinguish potential differences across treatment groups. This effect is worsened if individual PK profiles (requiring a greater number of plots overlaid) are produced, or if error bars are included. The baseline can be obscured, with different points overlaying each other even if jittering is applied.

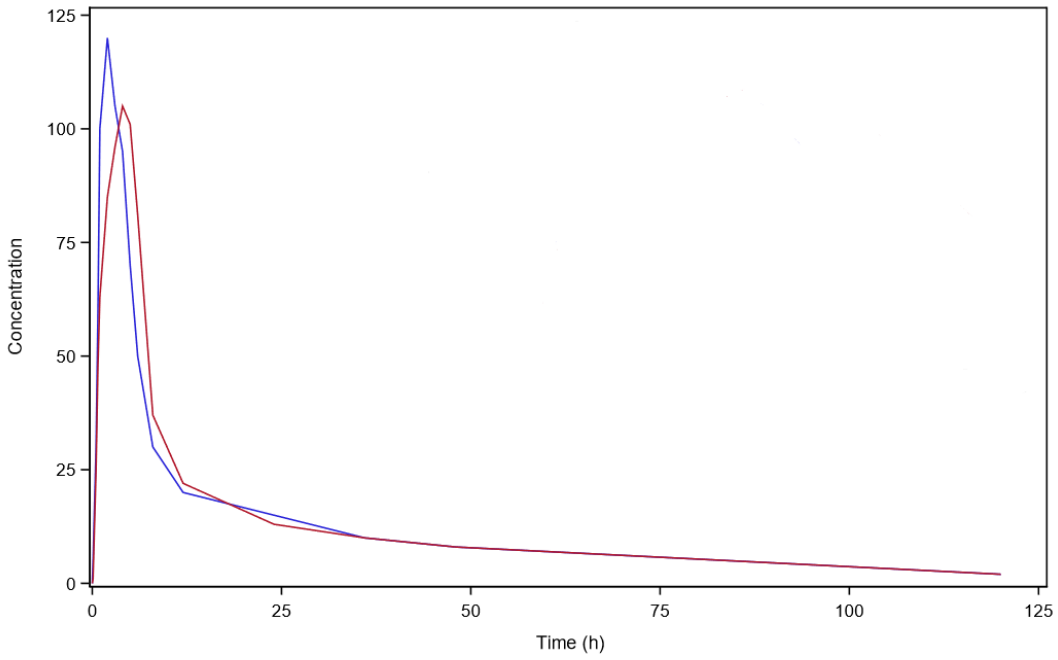


Figure 1 – Standard PK Plot

One solution to this problem is to focus on the area where there is a higher concentration of points, giving a much clearer comparison between the different treatments. However, we still want to maintain the overall picture, for the full duration of PK collection. On a graph like the example above, there is a large amount of white space that could be used. It therefore makes sense to embed a section of the graph within this white space, which focuses on the desired area.

This is done programmatically by producing two separate plots and inserting the smaller area into the area of the main plot using the drawimage option within GTL. The key elements in the code shown below are the ODS graphics ability to save a graphical output as a png file that can then be recalled. In this same way any graph could be embedded within another. Below is the macro code that would enable a close up of a specific section of the graph to be inserted into the top right corner and output in an rtf file.

```
%macro insert(dsin=,       /* Input dataset */
              xvar=,       /* x-axis variable */
              yvar=,       /* y-axis variable */
              group=       /* Grouping variable (plot a line for each value) */
              zxmax=,      /* Maximum x value for zoom area */
              xlabel=,     /* x-axis label */
              ylabel=,     /* y-axis label */
              ods=,        /* ods output type (rtf/ods) */
              lib=         /* Libname output */);
```

```
** Set up libnames and outputs;
filename fout "&lib";
libname odsutil "&lib";

** Create PNG image for insert plot;
ods graphics on / reset=index imagename="insert" width=20cm height=20cm imagefmt=png;
ods listing image_dpi=300 gpath=fout;

** Use sgplot to plot a series for the insert;
proc sgplot data=&dsin.(where=(&xvar. < &zxmax.)) noautolegend;
       series x=&xvar. y=&yvar. / group=&group. lineattrs=(pattern=1);
       xaxis label="&xlabel" labelattrs=(size=16) valueattrs=(size=16);
       yaxis label="&ylabel" labelattrs=(size=14) valueattrs=(size=14);
run;

** Close output for png;
ods graphics off;
ods listing;
filename fout clear;
goptions reset=all;

** Proc template for the main plot area;
proc template;
   define style MyDefault;
      parent = Styles.Listing;
      class GraphData1 /linestyle = 1 contrastcolor = GraphColors('gcdata1');
      class GraphData2 / linestyle = 1 contrastcolor = GraphColors('gcdata2');
   end;
   define statgraph insert;
   dynamic grp1;
      begingraph;
      layout overlay /xaxisopts = (label="&xlabel")
                      yaxisopts = (label="&ylabel");
                     seriesplot x=&xvar. y=&yvar. /group=&group. name="main";
       discretelegend "main" / title='Treatment' across=2 border=true;
       endlayout;

       /* Re-draw png image from series plot */
       drawimage "&lib\insert.png" / border=false layer=front anchor=topright
       x=99 y=99 height=60 width=60;
   endgraph;
   end;
run;

** Create ods ouput;
goptions reset=goption;
ods graphics / noborder height=6in width=9in;

options papersize=A4 orientation=landscape topmargin='1in' leftmargin='1in'
bottommargin='1in' rightmargin='1in';

ods _all_ close;
ods &ods. file  = "&lib\inset.&ods." style =  MyDefault;

proc sgrender data=&dsin. template=insert;
run;

ods &ods. close;
ods listing;
ods graphics off;

%mend;
```

If each graph were being created individually, SGPLOT would be used for both, but in order to use the drawimage option the overall output is produced by creating a custom ODS Graphics template, with proc template and proc sgrender. The syntax required for the GTL code is discussed elsewhere but can be generated from the standard behind SGPLOT. The only non-standard section of this macro is the of drawimage to write in the first graph created. In this case the file "insert.png" is written back into the template, with no border, layered on top of the second graph and anchored 1% down and left of the top corner:

```
/* Re-draw png image from series plot */
drawimage "&lib\insert.png" / border=false layer=front anchor=topright
x=99 y=99 height=60 width=60;
```

The final output here is a PK line graph that enables a reviewer to see the complete picture of the PK data as well as comparing any differences at early times. In this case the focus is on the first 8 hours and it can be seen how the two treatments cause differing profiles.

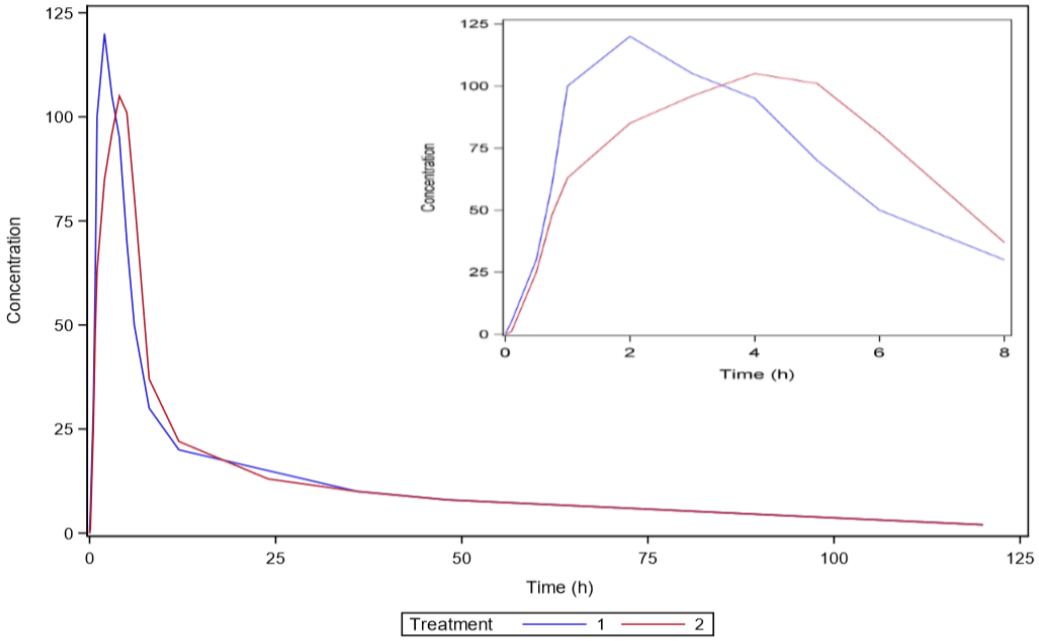
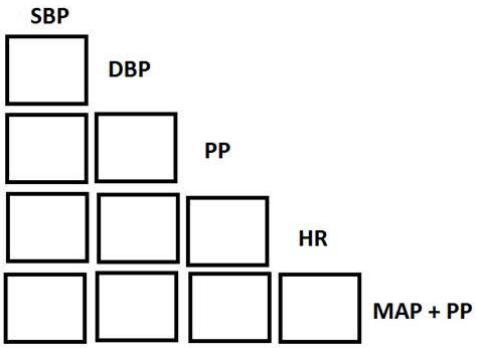


Figure 2 – PK Plot with embedded plot of smaller region

LATTICE SCATTERS

Similarly, a statistician requested a data visualisation that could display an overview of multiple scatter plots in a single place. This will show the relationship between two parameters or results at visits, for continuous variables, in a way that is easy to compare. As with the previous examples this gives an overview of several analyses side by side which can be compared. The detail of each should then still be reported elsewhere in tables and figures. The below is a layout requested for reporting by the statistician. It shows a structure where four recorded parameters and one derived parameter, for vital signs at baseline are plotted against each other in a lattice. In this way multiple graphs that may be plotted individually are produced on one output for comparison.



In SAS there is a standard way for doing something very similar to this, using the SGSCATTER procedure, a variation on SGPLOT, which creates a paneled graph of scatter plots for multiple combinations of variables and the MATRIX statement. As with many SAS graph procedures, this is restrictive and is far from ideal. Using the SGSCATTER procedure creates an output (shown below) with each combination of variables repeated either side of the diagonal but with the axes flipped. This creates noise and confusion on the output, especially as in the case below where the relationships between parameters are similar and could easily be confused.

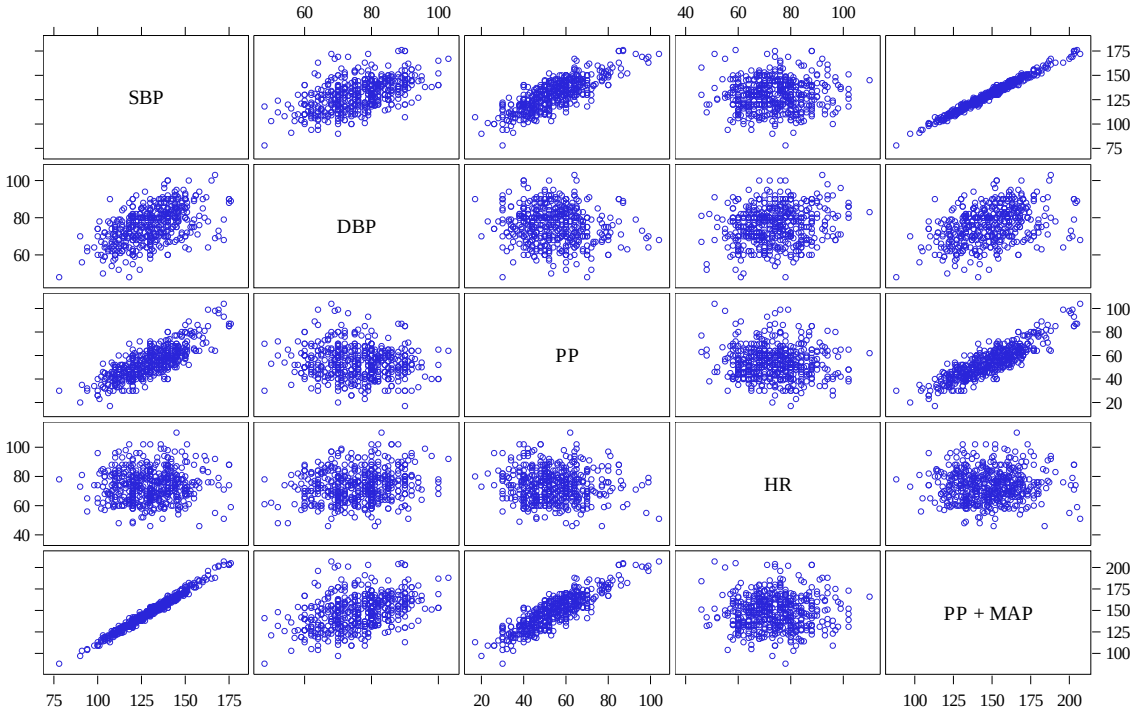


Figure 3 – Vital signs parameter plotted with SGSCATTER

To be able to produce the desired output, again GTL is required. The flexibility of GTL allows the creation of these figures in a customized format. The macro below can be used to create a lattice without repeats, for up to 10 named variables. This lattice will be created from the top left as in the requested example, leaving blank space in the top right of the page. Unlike PROC SGSCATTER the lattice macro will use a consistent axis scale for all the plots. The advantage of this is that it makes it easier to quickly compare the relationship between variables and the differences between them. However, this may cause some issues if working with variables that have a large variation as the trend may be difficult to see. As such it is important to edit the macro to fit to exactly what is required.

```
%macro lattice (dsin=,           /* data set name */
                numvar=,         /* number of variables to be plotted */
                var1=,           /* variable #1 for matrix */
                var2=,           /* variable #2 for matrix */
                var3=,           /* variable #3 for matrix */
                var4=,           /* variable #4 for matrix */
                var5=,           /* variable #5 for matrix */
                var6=,           /* variable #6 for matrix */
                var7=,           /* variable #7 for matrix */
                var8=,           /* variable #8 for matrix */
                var9=,           /* variable #9 for matrix */
                var10=,          /* variable #10 for matrix */
                minval=,         /* Axis start tick value*/
                maxval=,         /* Axis end tick value */
                valby=1,         /* axis tick value increment--*/
                fontsize=8,      /* plot font size */
                titlefontsize=12, /* title font size */
                footnote=,       /* footnote 1*/
                title=           /* title */);

** Will need one less row/ column than number of variables;
%let numvar = %eval(&numvar. - 1);

/* Create Graph Template */
proc template;
   define statgraph lattice_out;
   begingraph;
      entrytitle "&title"                    / textattrs=(size=&titlefontsize.);
      entryfootnote halign=left "&footnote"  / textattrs=(size=10);

/* Create n * n dimension matrix */
      layout lattice / columns=&numvar. rows=&numvar. rowgutter=5 columngutter=5
                       rowdatarange=union columndatarange=union;

/* Set up n rows for i variables */
       rowaxes;
       %do i=1 %to &numvar.;
       rowaxis /    tickvalueattrs=(size=&fontsize.) labelattrs=(size=&fontsize.)
                    linearopts=(tickvaluesequence=(start=&minval. end=&maxval.
                    increment=&valby.) tickvaluepriority=true);
       %end;
       endrowaxes;

/* Set up n columns for j variables */
       columnaxes;
       %do j=1 %to &numvar.;
       columnaxis / tickvalueattrs=(size=&fontsize.) labelattrs=(size=&fontsize.)
                    linearopts=(tickvaluesequence=(start=&minval. end=&maxval.
                    increment=&valby.)tickvaluepriority=true);
       %end;
       endcolumnaxes;

/* Create individual scatter plots */
/* Loop through to complete each row at a time where k =row, l = column */
/* Plot var(k) on x axis and var(l) on y axis */
       %do k = 1 %to &numvar.;
          %let y = %eval(&k. + 1);
          %do l = 1 %to &k.;
             layout overlay;
                    scatterplot y=&&&var&y. x=&&&var&l. / group=trt;
              endlayout;
          %end;
```

```
/* Create blank space at any point where l >= k */
          %do z=&l. %to &numvar.;
              layout overlay;
                    entry '';
              endlayout;
          %end;
       %end;
       endlayout;
       endgraph;
       end;
run;

proc sgrender data=&dsin. template=lattice_out;
run;
%mend;
```

To create, this structure, the basic proc template code that underlies PROC SGSCATTER was used as a starting point. A lattice for n variables, needs to be (n-1) * (n-1) structure, so the %EVAL macro function is used to create a new macro variable that holds the value of n -1. This can then be called back to set up a (n-1) * (n-1) lattice, using LAYOUT LATTICE.

Having created a blank lattice, the next stage is to create the individual plots and insert them in the right area of the graph. The first stage is to set up the axes in each section of the lattice. This is done by looping over a single axis for each row and column to (N – 1), thus creating only axis for the far left and far bottom, of the graph which will apply to all plots.

The individual plots are then created using the standard LAYOUT OVERLAY and SCATTERPLOT. To make sure that all variable permutations are combined and placed in the correct place in the lattice a combination of macro loops is used. On the x-axis we use a column for each variable from var1 to varN, where N is the total number of variables to be plotted (&numvar), using '`%do k = 1 %to &numvar.;`'. This means that the first row should use var2 on the y-axes. The loop '`%do l = 1 %to &k.;`' plots variable (k+1), where k is the column number, against variables 1 to k in a row. Thethird loop '`%do z=&l. %to &numvar.;`' then creates a blank space in the remaining columns with LAYOUT OVERLAY and a blank entry (where row number is greater or equal to column number). The process the repeats over each row. The template procedure is labelled as '`lattice_out`' and the final graph is then produced by recalling the template.

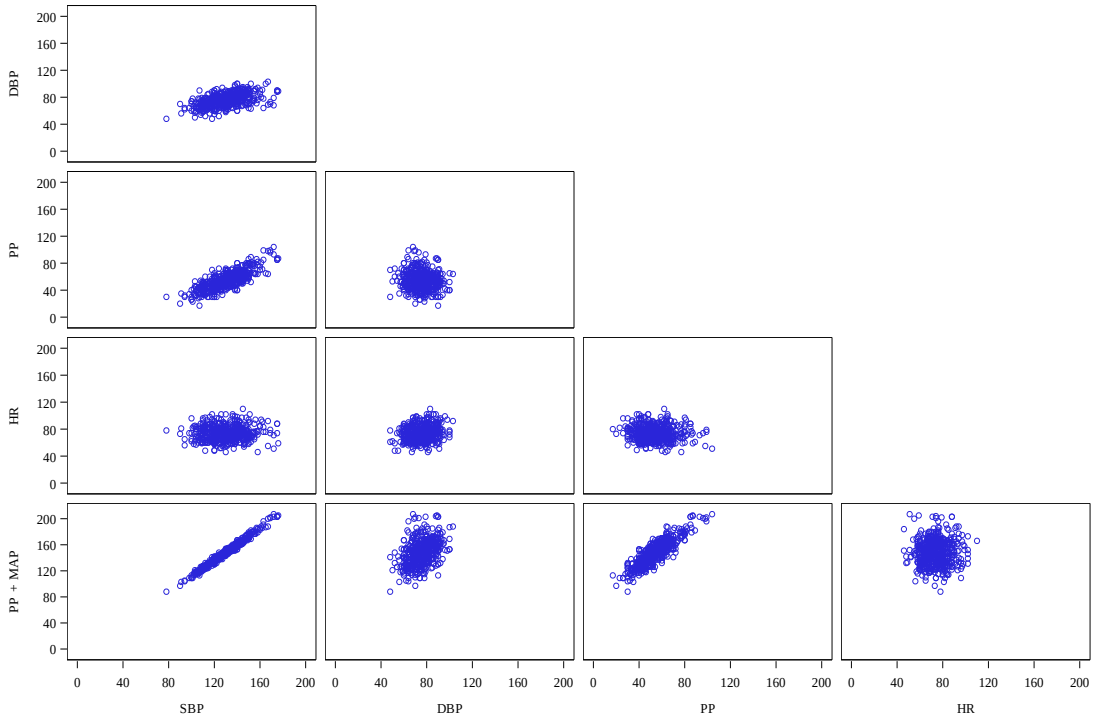


Figure 4 – Vital signs parameters plotted with as a lattice

In the figure above, you can see the resulting graph for the vital signs example. Each of the five variables is plotted against each other once. Comparing Figures 3 and 4 you can see the advantages of plotting in this way: Each combination of variables is only created once and not duplicated (or inverted), each combination is produced on the same scale which allows a much easier comparison across variables, and removing the diagonal of the graph only requires 4 rows and 4 columns instead of 5 rows and 5 columns, meaning each individual plot can be made slightly larger. All of this is useful for producing an overview of the relationship between parameters and for further detail the summary tables can be consulted.

SWIMMER PLOTS

In her paper "Swimmer Plot: Tell a Graphical Story of Your Time to Response Data Using PROC SGPLOT", Stacey Phillips details how PROC SGPLOT and ODS graphics alongside GTL allow the development of more sophisticated graphics. Stacey then demonstrates how to use SGPLOT and SAS annotations to create swimmer plots for tumor response in oncology studies. This paper will look at how Swimmer plots can be created using a combination of the SCATTER and HIGHLOW statements to compare best response to treatment in different time periods, in prior therapy and study treatment.

Plotting the time lines of prior therapies, study treatment and the best recorded response gives the reviewer a visual view of how subjects' responses compare with different therapies. If a subject has been responding poorly to previous therapies but responds well to the study treatment this might indicate a benefit, and vice-versa. This style of plot allows individual events such as treatment and surgery to be overlaid on the timeline to show how these events relate to responses. Similarly, adverse events could be plotted as swimmer plots, perhaps focusing on specific types of events or on those with a particular severity or outcome. This would enable a reviewer to see the relationship between dosing and time to event across multiple subjects in one place.
The first stage of producing these plots is to process the data so that the date of each event was expressed as number of days from a reference point – for events with start and end dates this would be a start and end day. The reference point should be picked as either dosing date, randomization date or the first date to be displayed. In the example prior therapy plot in figure 5, the first date being displayed has been used for day 0 and selected from the data using proc SQL.

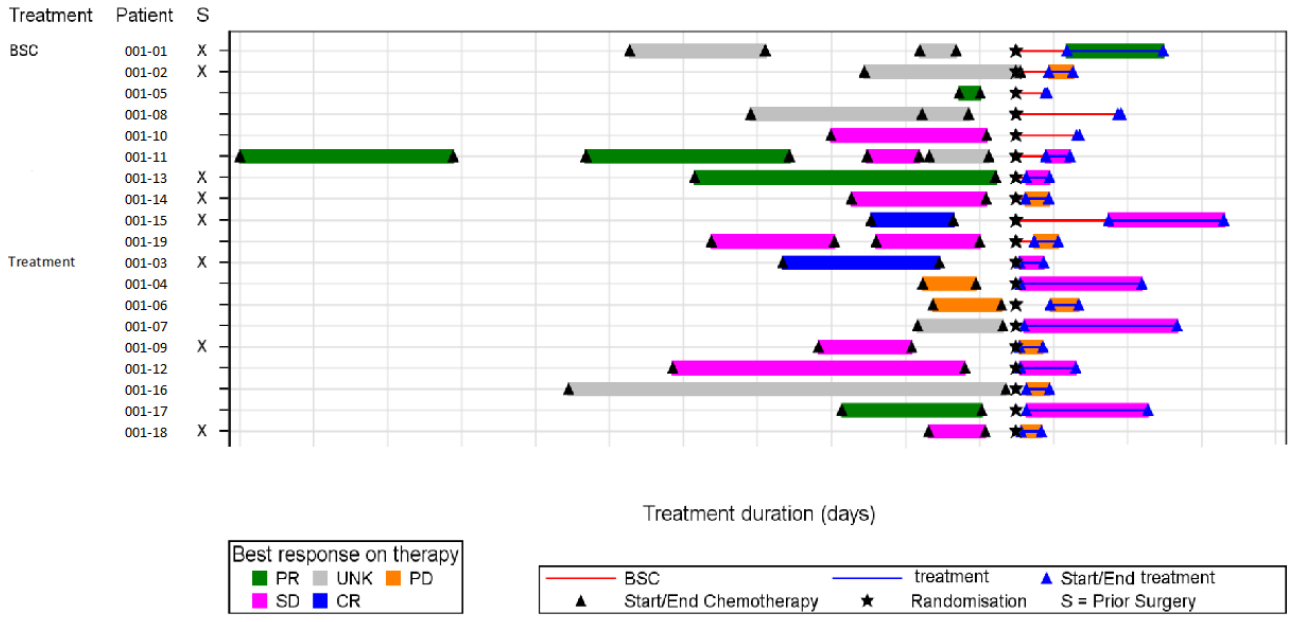


Figure 5 – Swimmer Plot comparing Prior Therapy and Study Treatment

The code below shows how to then plot various elements of a swimmer plot, displaying prior therapy, study treatment as shown in figure 5.

```
proc sgplot data=plots dattrmap=getattrs uniform=scale;
    highlow y=subjid low=ady high=aendy / type=bar group=aval  groupdisplay=overlay
                                          barwidth=0.6 name='a' attrid=Color ;
            yaxis reverse display=(nolabel novalues) grid fitpolicy=splitalways;
            xaxis label="Treatment duration (days)" grid values=(0 to 1400 by 100);

    highlow y=subjid low=adyb high=aendyb / type=line group=avalt name='b'
                                            lineattrs=(pattern=1  thickness=1pt)
                                            groupdisplay=overlay attrid=Color;

   scatter y=subjid x=adyt  / markerattrs=(symbol=TriangleFilled color=blue  size=7)
                              name='c' legendlabel="Start/End CAP7.1 treatment";
   scatter y=subjid x=aendyt/ markerattrs=(symbol=TriangleFilled color=blue  size=7);
   scatter y=subjid x=adyc  / markerattrs=(symbol=TriangleFilled color=black size=7)
                              name='e' legendlabel="Start/End Chemotherapy";
   scatter y=subjid x=adyr  / markerattrs=(symbol=StarFilled     color=white size=7)
                              name='g'  legendlabel="S = Prior Surgery";
   scatter y=subjid x=adyr  / markerattrs=(symbol=StarFilled     color=black size=7)
                              name='f'  legendlabel="Randomisation";

   yaxistable trt01p   / position=left  label="Treatment"  labelpos=bottom ;
   yaxistable subjid   / position=left  label="Patient"    labelpos=bottom ;
   yaxistable surg     / position=left  label="S"          labelpos=bottom ;

   keylegend 'a' / position=bottomleft  across = 3 title='Best response on therapy';
   keylegend 'b' 'c' 'e' 'f' 'g'/ position=bottomright across = 3;
run;
```

There are three different methods of plotting used in this procedure to combine the different event types: HIGHLOW, SCATTER and YAXISTABLE. The HIGHLOW statement is used to produce the horizontal bars representing both prior therapy and study treatment. The first use creates a bar representing response to therapy from start date (low=ady) to end date (high=aendy), coloured by result. The second statement then plots the duration of treatment, using 'type=line' to draw a line instead of a bar. The graph axis is also defined in the first statement but could be defined during any of the plots.

SCATTER statements are then used to mark the start and end of treatment, chemotherapy and date of randomization on top of the highlow bars. The YAXISTABLE in SAS version 9.4 can then be used to add one or more columns of textual data to a graph aligned with the y-axis - subject identifiers, treatment group and a flag for previous surgery is added on to the left of the plot. The final tool used in proc sgplot is the ability to combine legend statements from multiple plots into a single plot (or in this case 2). Each individual plot is labelled using 'name=' and given and defined using 'legendlabel' where no grouping is performed.

CONCLUSION

This paper gives examples of how the advances in SAS graphics and the flexibility provided by using Graphical Template Language (GTL) enables the production of more detailed and useful data visualisations and graphs. With further understanding of the improvements in SAS graphics procedures, it becomes possible to increase the use of such graphs in study reporting. Programmers and statisticians are encouraged to think about improvements in data visualisations that can be included in future study reporting.