PharmaSUG 2015 - Paper HT01

Picture this: Hands-on SAS Graphics Session

Kriss Harris, SAS Specialists Ltd, Hertfordshire, United Kingdom

ABSTRACT

Would you like to be more confident in producing graphs and figures? Do you understand the differences between SGPLOT, SGSCATTER and SGPANEL? Would you like to know the different layout options in Graph Template Language (GTL)? Would you like to know how to easily create industry standard graphs such as Adverse Event timelines, Kaplan-Meier plots and Waterfall plots? Finally, would you like to learn all of these methods in a relaxing open environment that fosters questions? Great, then this topic is for you. In this Hands-on SAS Graphics Session you will be skillfully guided through SGPLOT, SGSCATTER, SGPANEL and GTL. You will also complete fun and challenging SAS Graphics exercises to enable you to retain what you have learned more easily. This Hands-on SAS Graphics session is structured so that you will learn how to create the standard plots that your manager requests, how to easily create simple ad-hoc plots for your customers and also how to create complex graphics. You will be shown how to annotate your plots and this includes how to add Unicode characters on to your plots. You will find out how to create reusable templates, which can be used by your team. Years of information have been carefully condensed into this 90 minutes Hands-on SAS Graphics session. This session will give you insight and will be very interactive! Feel free to bring some of your challenging graphical questions along!

INTRODUCTION

Before giving detailed solutions on frequently asked graphic questions, an overview of the different types of SG Procedures and when to use each one is presented.

SGPLOT

Use SGPLOT when you want to produce single-cell graphs. Figure 1 below, shows an example of a shell of a single-cell graph. Figure 1 is displayed to aid in remembering the use of SGPLOT. Figure 2 is an example of a single-cell graph produced with SGPLOT and shows a box plot of the intensity by visit number for a particularly Laboratory Test.

Figure 1: SGPLOT Blank Canvas

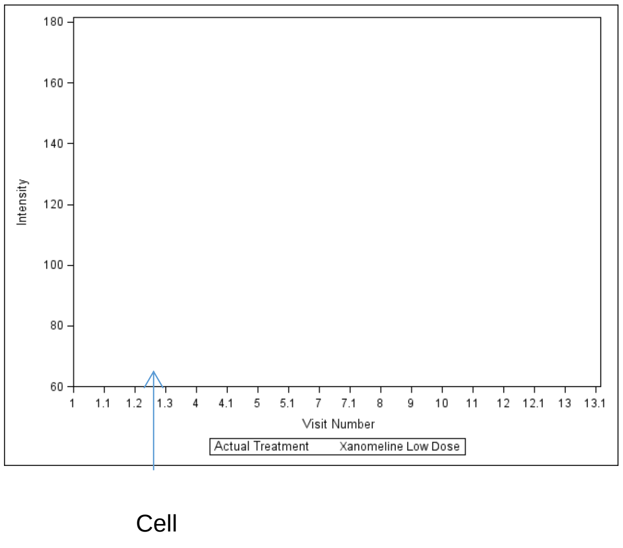
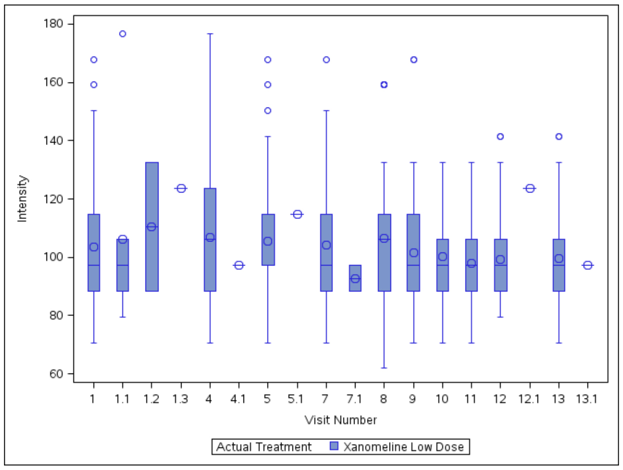


Figure 2: SGPLOT



SGPANEL

Use SGPANEL when you want to produce multi-cell graphs. Think of SGPANEL as being similar to SGPLOT; however instead of producing single-cell graphs, you are producing multi-cell graphs.

In SGPANEL, either each row or each column must have identical values, and this can be very useful when comparing results between rows and/or columns. In this context, values are also known as tickvalues. Because each row or each column must have identical values in SGPANEL, if you want each cell to have values that constitute its own axis range, you will need to use SGSCATTER or Graph Template Language (GTL) to achieve this. Figure 9 shows an example of how you can use GTL to enable each cell to have its own independent axes.

Figure 3 shows an example of a shell of a multi-cell graph. Typically when using SGPANEL on Study Data Tabulation Model (SDTM) data you will panel by the --TEST variables, such as LBTEST, EGTEST or VSTEST, or you will panel by the VISIT variable. In addition, when using SGPANEL on Analysis Data Model (ADaM) data, you will usually panel by the PARAM variable as in Figure 3 and Figure 4 or the AVISIT variable.

Figure 3: SGPANEL Blank Canvas

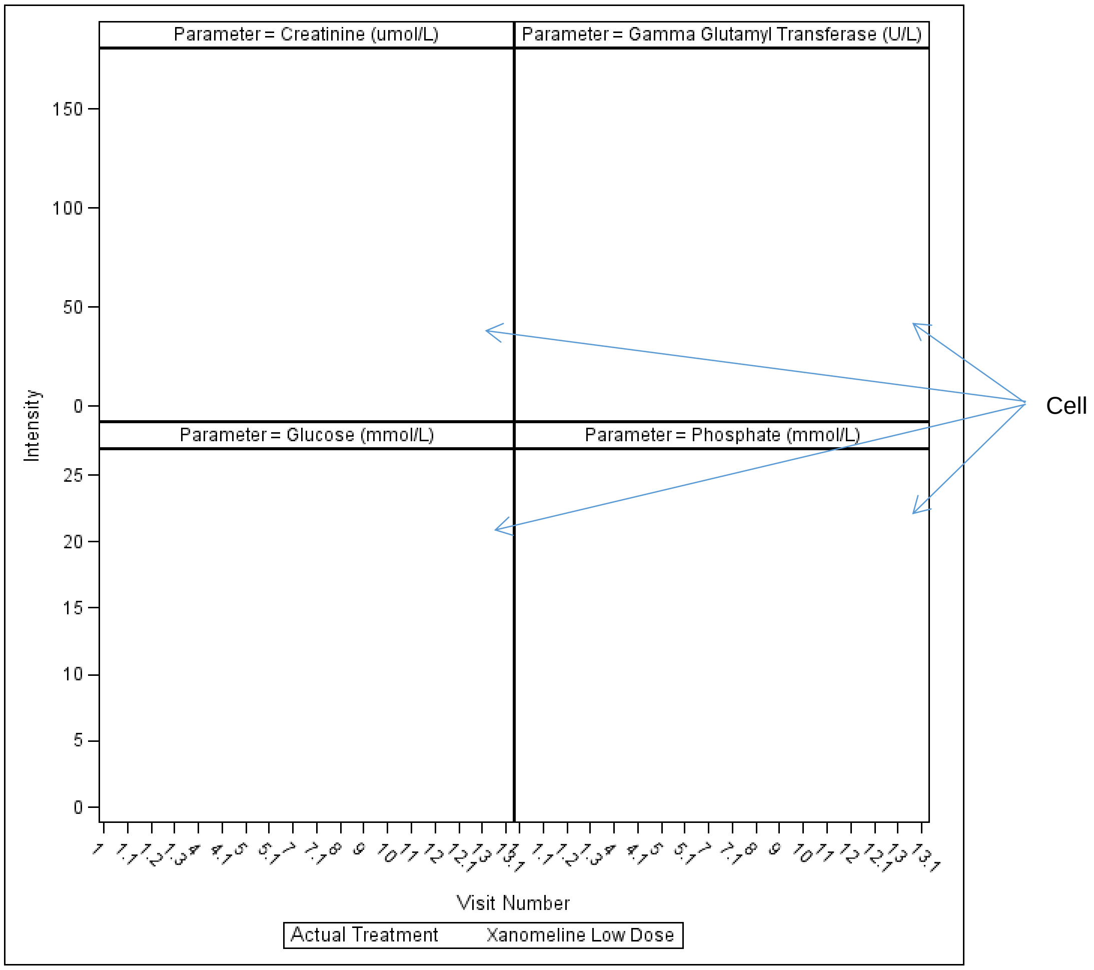
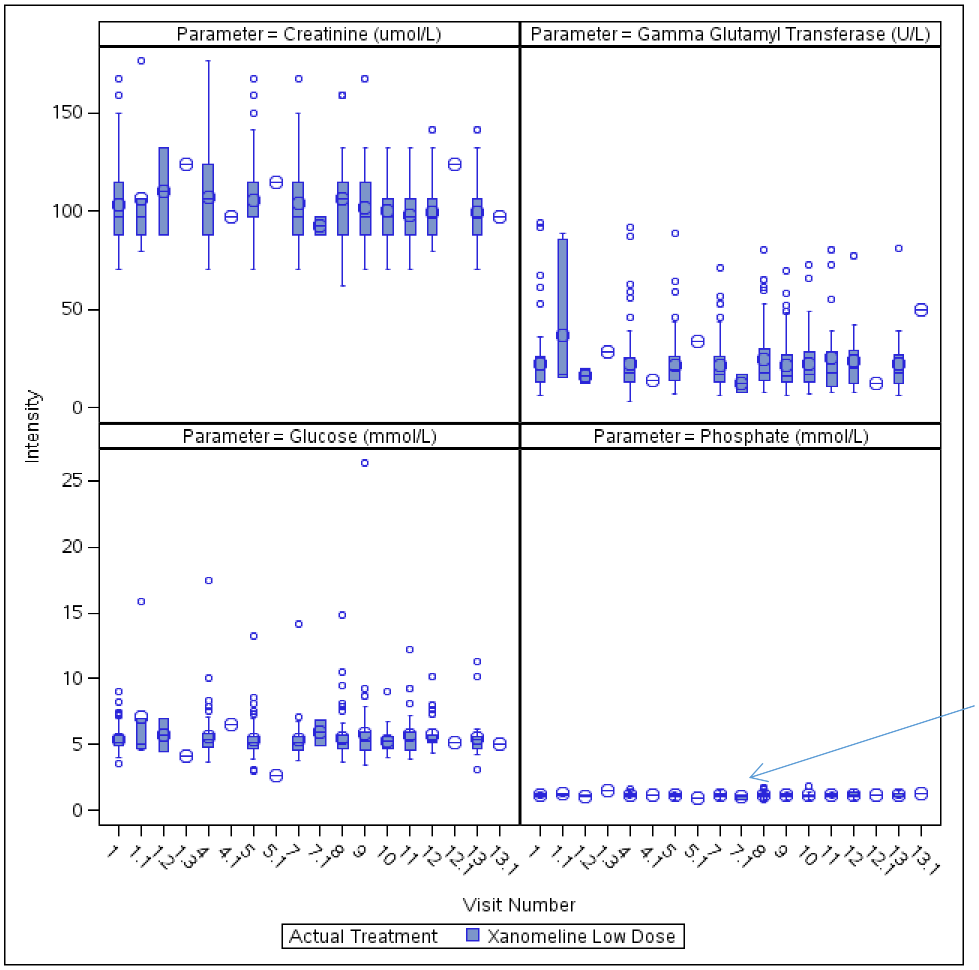


Figure 4 is an example of a multi-cell plot produced with SGPANEL and shows a box plot of the intensity by visit number paneled by the PARAM variable, which in this case are the Laboratory Tests. In Figure 4, the Phosphate intensity values are much lower than the Glucose intensity values, and because the Phosphate and Glucose cells share a common intensity axis scale, this makes the variation of Phosphate intensity values at each visit difficult to see. Figure 9 shows an example of how you can use GTL to achieve independent axes in each cell, and therefore see the variation of the Phosphate intensity values.

Figure 4: SGPANEL



Difficult to see Phosphate intensity values on this scale.

SGCATTER

Use SGSCATTER when you want to produce multi-cell scatter plots that have independent axes or matrix plots. SGSCATTER has three statements that you can use to create a paneled graph of scatter plots. They are:

- PLOT
- COMPARE
- MATRIX

This paper only shows graphs from two of the statements in SCATTER: PLOT and MATRIX. Figure 5 shows an example of a shell of a multi-cell graph which has independent axes.

Figure 5: SGSCATTER Blank Canvas

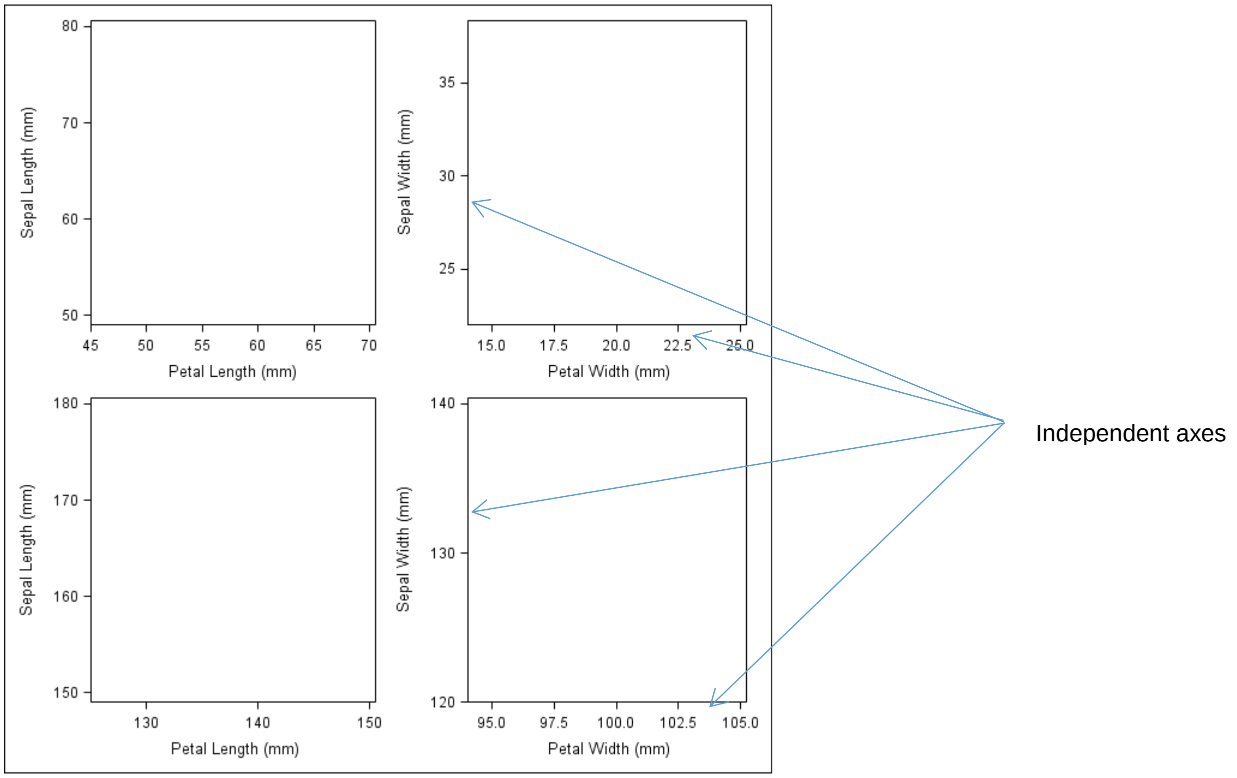


Figure 6 shows an example of a multi-cell scatter plot with independent x-axis that was produced using SGSCATTER in conjunction with the PLOT statement where one of the x-axis denotes the Petal Length and the other x-axis denotes the Petal Width. With SGSCATTER it is possible to also produce a scatter plot that have both the x-axis and the y-axis independent of each other; although the caveat with this method is that the plot type needs to be a scatter plot.

Figure 7 shows an example of a multi-cell matrix plot that was produced using SGSCATTER in conjunction with the MATRIX statement. A matrix plots each variable that you specify against the other.

Figure 6: SGSCATTER – Plot

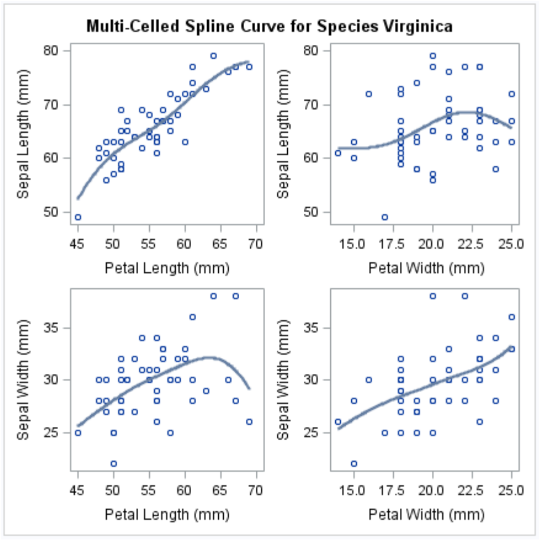
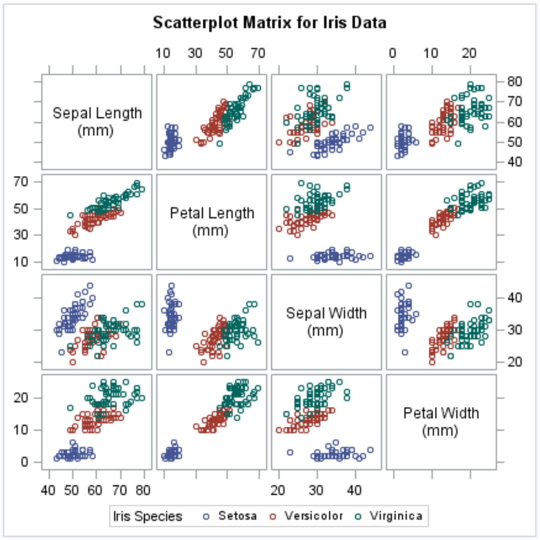


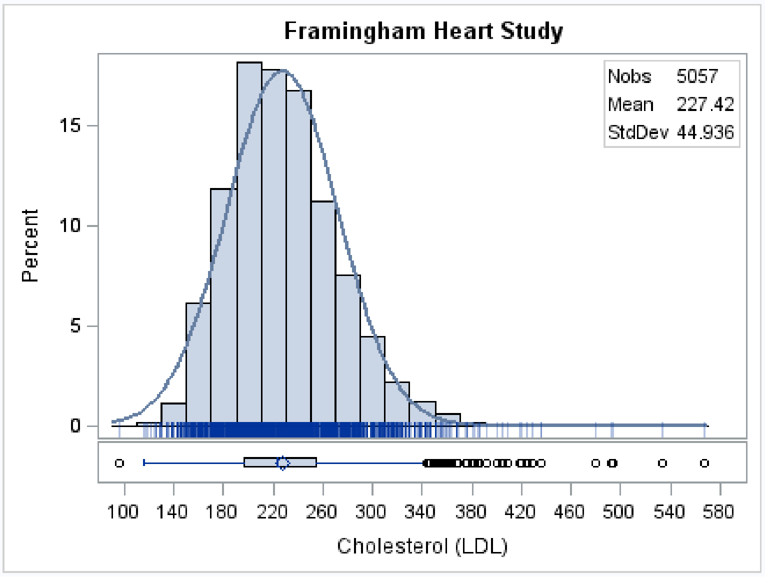
Figure 7: SGSCATTER – Matrix



GRAPHICS TEMPLATE LANGUAGE - GTL

GTL is used to produce complex graphs. An example is where you want to produce a histogram and a box plot on the same graph, such as in Figure 8. Another example of a complex graph is a graph where you want each cell to have its own independent axes and the graph is not a scatter plot, such as in Figure 9.

Figure 8: GTL – LAYOUT LATTICE, Histogram and Horizontal Box Plot



INDEPENDENT AXES

Hebbar and Matange (2013) show how you can simulate the subsetting of observation in GTL. You can use these ideas as illustrated in Program 1 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 another method to create cells that have independent axes 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, you will end up with a dataset with a large number of columns.

Program 1 shows a snippet of the full GTL code which produces the template for Figure 9. Figure 9 shows boxplots of intensity by visit number for four laboratory tests, and each laboratory test has its own independent axes. Figure 9 was created using the layouts LAYOUT LATTICE and LAYOUT OVERLAY in GTL.

In Program 1 the EVAL and IFN functions are used to only show the Creatinine 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.

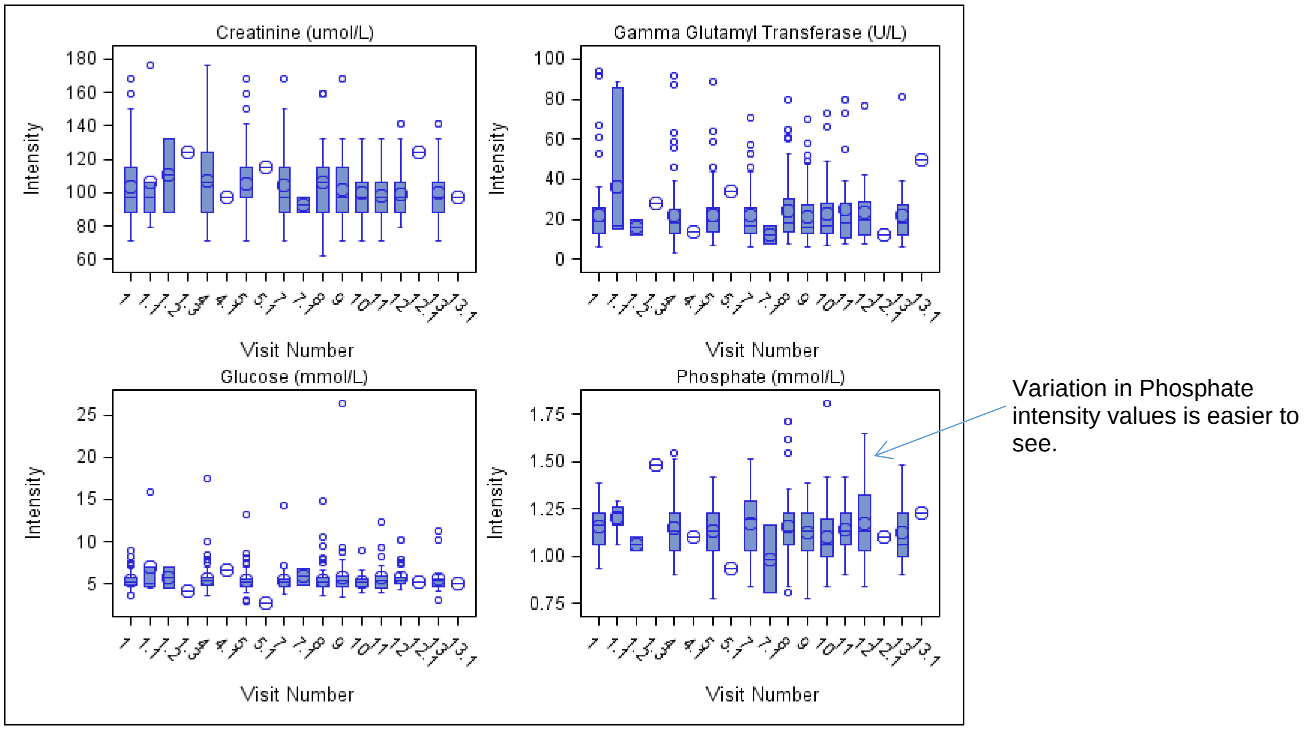
Regarding the IFN function, there is another function called the IFC function, and these functions often get mixed up. A way to remember the differences between the IFN and IFC functions is to remember that the “N” or “C” part refers to the type of value that the function is returning, and does not take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

Program 1

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

  cell;
    cellheader;
      entry "Creatinine (umol/L)";
    endcellheader;
    layout overlay / yaxisopts = (label = "Intensity")
                     xaxisopts = (label = "Visit Number");
    boxplot x = eval(ifn(param = "Creatinine (umol/L)", visitnum, .))
            y = eval(ifn(param = "Creatinine (umol/L)", aval, .)) / group = trta;
    endlayout;
  endcell;
```

Figure 9: GTL – Boxplot of Intensity by Visit Number with Independent Axes



A few drawbacks of using the method in Program 1 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.

To address a few of the drawbacks in Program 1, Program 2 was created and shows how Program 1 can be made more generalized with the aid of a Macro. In Program 2, the %DO loop is used to obtain and subsequently input the value of each parameter to subset on. The following code, which is extracted from Program 2, uses %QSCAN to obtain the value of the parameter based on the value of the macro variable **i**.

```
%LET parameter = %QSCAN(&index, &i, %str(,));
```

Program 2

```
%macro plot(index);

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

         %do i = 1 %to 4;
           %let parameter = %qscan(&index, &i, %str(,));
              cell;
                cellheader;
                  entry "&parameter.";
                endcellheader;
               layout overlay / yaxisopts = (label = "Intensity")
                                xaxisopts = (label = "Visit Number");
                 boxplot x = eval(ifn(param = "&parameter", visitnum, .))
                         y = eval(ifn(param = "&parameter", aval, .)) / group =
                                                                         trta;
               endlayout;
             endcell;
         %end;

       endlayout;
     endgraph;
  end;
run;

%mend;

%plot(%str(Creatinine (umol/L),Gamma Glutamyl Transferase (U/L),Glucose
(mmol/L),Phosphate (mmol/L)));
```

Program 2 can be further generalized by using a macro variable that contains the parameters you want to plot instead of manually entering the parameters themselves.

Program 3 shows this further generalization using the INTO clause to create two macro variables. The first macro variable **count_param** contains the number of parameters, and the second macro variable **parameterlist** contains the list of parameters (separated by a comma) that we wish to plot. The CATS function is needed to ensure that the **parameterlist** is being interpreted as a string and therefore **parameterliststring** was created. Program 3 can be further generalized by substituting **count_param** for the number 4 in the %DO loop and using macro variables to automatically specify the number of rows or columns of cells that you wish to use in your output.

Program 3

```
proc sql;
  select distinct count(distinct param), param into :count_param, :parameterlist
separated by ","
  from adlbc4;
quit;

%let parameterliststring = %sysfunc(cats((&parameterlist)));

%plot(%str&parameterliststring);
```

KAPLAN-MEIER

Firstly, Kuhfeld and So (2014) show numerous examples on how to create and customize the Kaplan-Meier survival curve in PROC LIFETEST, and it is helpful to read this paper. An alternative method to create Kaplan-Meier survival curves is to output the statistics and estimates from LIFETEST into a dataset and then subsequently use GTL code to produce a survival curve. The GTL code can be constructed to output survival curves, which include a table of the median survival times, the number of events, and the hazard ratio etc.

This paper, builds upon the Harris (2013) paper. Harris (2013) shows extracts of the GTL code that can be used to produce a survival curve. Although, there are a few things you need to be aware of when using the datasets that are outputted by LIFETEST as the source data for your survival curves. One of the things you need to be aware of is that if you specify the ATRISK option in PROC LIFETEST, and you output the PRODUCTLIMITESTIMATES table such as in Program 4, then in your PRODUCTLIMITESTIMATES dataset, there will be a column named NUMBERATRISK, such as in Table 1. I know it is confusing, but you should use the column named LEFT instead of the NUMBERATRISK column when you want to display the number of subjects at risk on your survival curve.

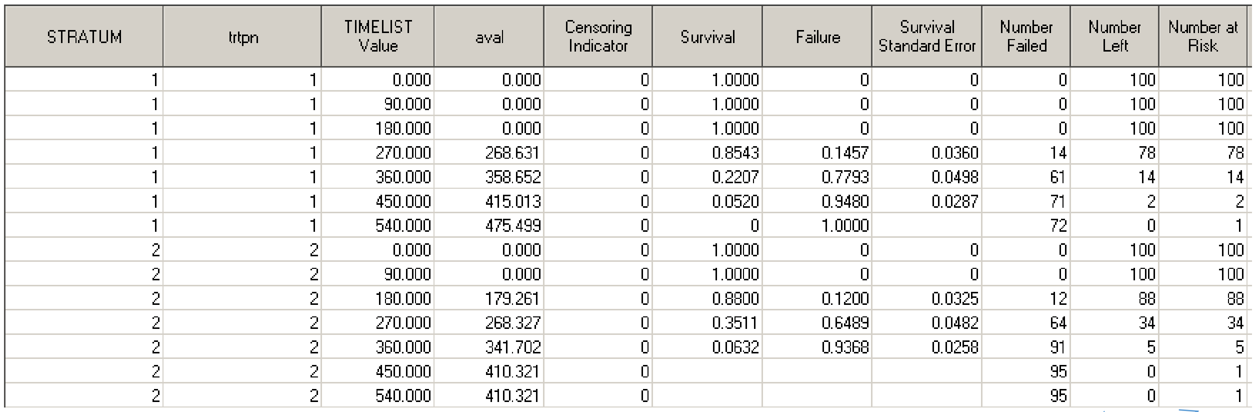
Figure 10 is an example of a survival curve the PROC LIFETEST produces. Figure 11 is an example of a survival curve produced using GTL, using the datasets outputted by PROC LIFETEST. The GTL code in Figure 11, uses the NUMBERATRISK column, instead of the LEFT column. If you compare the at risk numbers in Figure 10 with Figure 11, you will see that there are differences in the last two columns of at risk numbers. For example, at 450 days, Figure 10 shows that there are no subjects at risk in the Treatment 2 group; however Figure 11 shows that at 450 days, there is one subject at risk in the Treatment 2 group. The subjects at risk in Figure 10 have been confirmed as correct, and therefore if you use the data in the PRODUCTLIMITESTIMATES dataset to plot the subjects at risk, then you should use the LEFT column, such as in Figure 12.

Program 4: LIFETEST Code

```
ods output Quartiles=Quartiles(where=(Percent=50));
ods output CensoredSummary=CensoredSummary(where=(control_var ne "-"));
ods output ProductLimitEstimates =ProductLimitEstimates;
ods output HomTests=HomTests(where=(test="Log-Rank"));
proc lifetest data = adtte outsurv = outsurv atrisk timelist = 0 to 540 by 90
plots=survival(atrisk=0 to 540 by 90);
  time AVAL * CNSR(1);
  strata TRTPN;
run;
```

Table 1 shows the output of the PRODUCTLIMITESTIMATES. As mentioned earlier, there can be differences between the results in the LEFT column (labeled Number Left) and the NUMBERATRISK column (labeled Number at Risk), and you should use the LEFT column if you want to show the number of subjects at risk.

Table 1: Product Limit Estimates



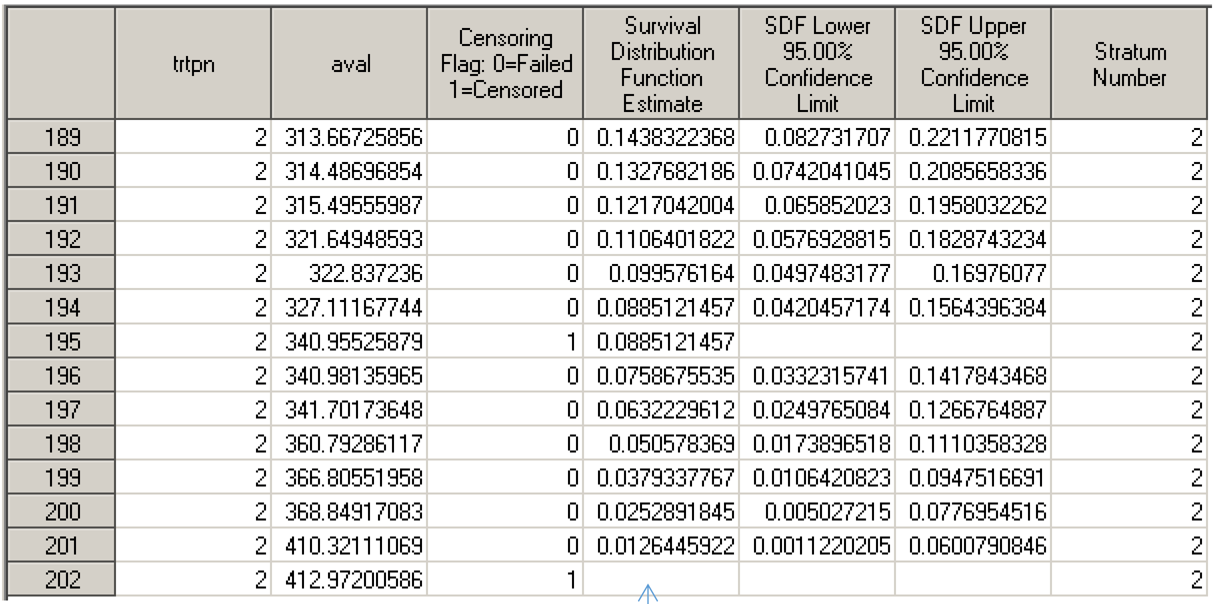
STRATUM	trtpn	TIMELIST Value	aval	Censoring Indicator	Survival	Failure	Survival Standard Error	Number Failed	Number Left	Number at Risk
1	1	0.000	0.000	0	1.0000	0	0	0	100	100
1	1	90.000	0.000	0	1.0000	0	0	0	100	100
1	1	180.000	0.000	0	1.0000	0	0	0	100	100
1	1	270.000	268.631	0	0.8543	0.1457	0.0360	14	78	78
1	1	360.000	358.652	0	0.2207	0.7793	0.0498	61	14	14
1	1	450.000	415.013	0	0.0520	0.9480	0.0287	71	2	2
1	1	540.000	475.499	0	0	1.0000		72	0	1
2	2	0.000	0.000	0	1.0000	0	0	0	100	100
2	2	90.000	0.000	0	1.0000	0	0	0	100	100
2	2	180.000	179.261	0	0.8800	0.1200	0.0325	12	88	88
2	2	270.000	268.327	0	0.3511	0.6489	0.0482	64	34	34
2	2	360.000	341.702	0	0.0632	0.9368	0.0258	91	5	5
2	2	450.000	410.321	0				95	0	1
2	2	540.000	410.321	0				95	0	1

Difference between LEFT and NUMBERATRISK column. Use LEFT column.

Table 2 is created using the OUTSURV option in PROC LIFETEST and contains the survival estimates and confidence limits. When the maximum value within a stratum is censored, then the SURVIVAL value (Survival Distribution Function Estimate) is missing for the corresponding maximum value, and occasionally is missing at previous time-points. The issue with the survival time being missing at the higher time-points is that when using the OUTSURV dataset to plot the survival curve, the curves do not look like the survival curves that are produced by the PROC LIFETEST which is incorrect as results are not present for the maximum time in the OUTSURV dataset which will lead to the curves being incorrect as at least one curve will not show the last subjects' censored result; These are displayed in Figure 11 and Figure 12. If you look at Figure 10, you will notice that there is a subject in Treatment Group 2, which was censored after day 400. However, Figure 11 and Figure 12 do not show this.

A solution for when the survival distribution function estimate is not present at the higher time-points, and should be present, is to calculate the minimum survival distribution function estimate of each stratum and then merge the calculated minimum survival distribution function estimate onto every result that has a missing survival distribution function estimate. The merge should be done at the stratum level. Figure 13 displays the corrected curve and mirrors Figure 10.

Table 2: Outsurv



	trtpn	aval	Censoring Flag: 0=Failed 1=Censored	Survival Distribution Function Estimate	SDF Lower 95.00% Confidence Limit	SDF Upper 95.00% Confidence Limit	Stratum Number
189	2	313.66725856	0	0.1438322368	0.082731707	0.2211770815	2
190	2	314.48696854	0	0.1327682186	0.0742041045	0.2085658336	2
191	2	315.49555987	0	0.1217042004	0.065852023	0.1958032262	2
192	2	321.64948593	0	0.1106401822	0.0576928815	0.1828743234	2
193	2	322.837236	0	0.099576164	0.0497483177	0.16976077	2
194	2	327.11167744	0	0.0885121457	0.0420457174	0.1564396384	2
195	2	340.95525879	1	0.0885121457			2
196	2	340.98135965	0	0.0758675535	0.0332315741	0.1417843468	2
197	2	341.70173648	0	0.0632229612	0.0249765084	0.1266764887	2
198	2	360.79286117	0	0.050578369	0.0173896518	0.1110358328	2
199	2	366.80551958	0	0.0379337767	0.0106420823	0.0947516691	2
200	2	368.84917083	0	0.0252891845	0.005027215	0.0776954516	2
201	2	410.32111069	0	0.0126445922	0.0011220205	0.0600790846	2
202	2	412.97200586	1				2

No survival value

Figure 10: Survival Curve using LIFETEST

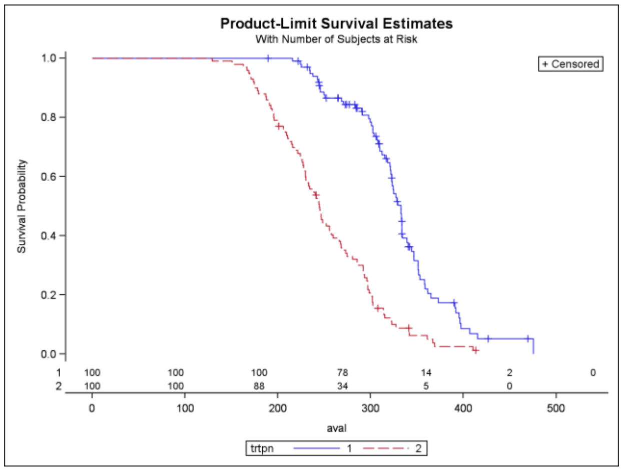
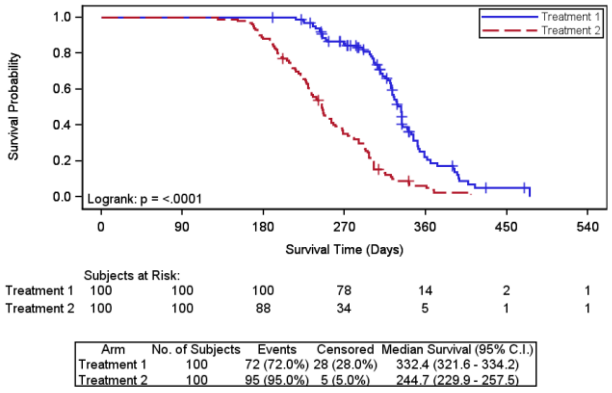


Figure 11: Survival Curve using GTL and NUMBERATRISK column

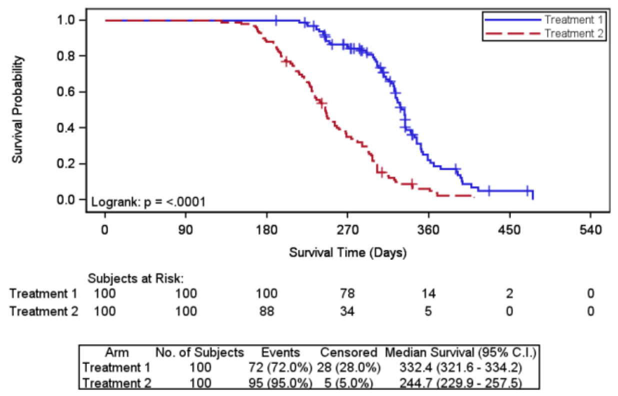


Subjects at Risk:

Treatment 1	100	100	100	78	14	2	1
Treatment 2	100	100	88	34	5	1	1

Arm	No. of Subjects	Events	Censored	Median Survival (95% C.I.)
Treatment 1	100	72 (72.0%)	28 (28.0%)	332.4 (321.6 - 334.2)
Treatment 2	100	95 (95.0%)	5 (5.0%)	244.7 (229.9 - 257.5)

Figure 12: Survival Curve using GTL and LEFT column

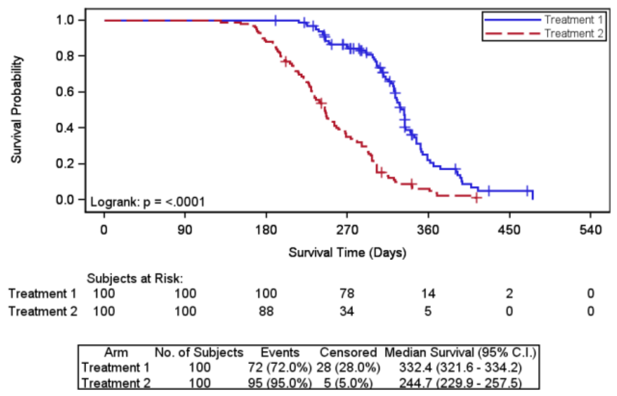


Subjects at Risk:

	0	90	180	270	360	450	540
Treatment 1	100	100	100	78	14	2	0
Treatment 2	100	100	88	34	5	0	0

Arm	No. of Subjects	Events	Censored	Median Survival (95% C.I.)
Treatment 1	100	72 (72.0%)	28 (28.0%)	332.4 (321.6 - 334.2)
Treatment 2	100	95 (95.0%)	5 (5.0%)	244.7 (229.9 - 257.5)

Figure 13: Correct Survival Curve using GTL



Subjects at Risk:

	0	90	180	270	360	450	540
Treatment 1	100	100	100	78	14	2	0
Treatment 2	100	100	88	34	5	0	0

Arm	No. of Subjects	Events	Censored	Median Survival (95% C.I.)
Treatment 1	100	72 (72.0%)	28 (28.0%)	332.4 (321.6 - 334.2)
Treatment 2	100	95 (95.0%)	5 (5.0%)	244.7 (229.9 - 257.5)

The full code for Figure 13 is in the appendix.

CONCLUSION

SGPLOT, SGPANEL, and SGSCATTER are very useful for quickly plotting basic single-cell and multi-cell figures; for more advanced figures GTL is required.

The majority of plots from the very simple to the very complex can be produced by SGPLOT, SGPANEL, LAYOUT LATTICE and LAYOUT OVERLAY.

More examples will be available during the hands-on training session, such as displaying Unicode symbols and the advantages and disadvantages of using Attribute Maps. So please attend the Picture this: Hands-on SAS Graphics Session!

REFERENCES

Knowledge Base / Samples & SAS Notes. (2011). Available at http://support.sas.com/kb/26/155.html

Harris, Kriss. 2013. “Creating High Quality Statistical Graphs for Publications” *Proceedings of the PharmaSUG 2014 Conference*. Available at http://pharmasug.org/proceedings/2013/DG/PharmaSUG-2013-DG09.pdf

Hebbar, Prashant and Matange, Sanjay. 2013. “Free Expressions and Other GTL Tips” *Proceedings of the SAS Global Forum 2013 Conference*. Available at http://support.sas.com/rnd/datavisualization/papers/sgf2013/371-2013.pdf

Kuhfeld, Warren. February 2010. *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*. Cary, North Carolina. SAS Institute.

Kuhfeld, Warren and So, Ying. 2014. “Creating and Customizing the Kaplan-Meier Survival Plot in PROC LIFETEST in the SAS/STAT® 13.1 Release” *Proceedings of the PharmaSUG 2014 Conference*. Available at http://www.pharmasug.org/proceedings/2014/SP/PharmaSUG-2014-SP14-SAS.pdf

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, Sanjay. November 2013. *Getting Started with the Graph Template Language in SAS: Examples, Tips, and Techniques for Creating Custom Graphs*. Cary, North Carolina. SAS Institute.

CONTACT INFORMATION

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

Name: Kriss Harris
Enterprise: SAS Specialists Ltd.

E-mail: italjet125@yahoo.com
Web: http://www.krissharris.co.uk
Twitter: https://twitter.com/krissharris

APPENDIX

```
%let outpath =U:\pharmasug\2015\ZQJ_HOWHarris\HT01\output\figures\other plots;
%let progname=kaplan;

data adtte;
  call streaminit(1000888);
  do i = 1 to 100;
    aval = rand("normal",310, 50);
    trtpn = 1;
    cnsr = int(round(rand("uniform")+0.2,0.1));
    output;
  end;
  do i = 101 to 200;
    aval = rand("normal",250, 50);
    trtpn = 2;
    cnsr = int(round(rand("uniform"),0.1));
    output;
  end;
run;

proc sort data = adtte;
  by trtpn aval;
run;

data adtte;
  set adtte;
  by trtpn aval;
  if trtpn = 2 and last.trtpn then cnsr = 1;
run;

/* Finding out the maximum for calculating at risk intervals and plotting the graph */

%let interval = 90;

proc sql;
select ceil(max(aval)/&interval.)*&interval. into: max
from adtte;
quit;

ods graphics / reset = all;

ods output Quartiles=Quartiles(where=(Percent=50));
ods output CensoredSummary=CensoredSummary(where=(control_var ne "-"));
ods output ProductLimitEstimates =ProductLimitEstimates;
ods output HomTests=HomTests(where=(test="Log-Rank"));
proc lifetest data = adtte outsurv = outsurv atrisk timelist = 0 to &max. by
&interval. plots=survival(atrisk=0 to &max. by &interval.);
  time AVAL * CNSR(1);
  strata TRTPN;
run;

data quartiles2;
```

```
  set quartiles;
  LowerLimVal=strip(put(LowerLimit, 5.1));
  if LowerLimVal='' then LowerLimVal='  NE';
  UpperLimVal=strip(put(UpperLimit, 5.1));
  if UpperLimVal='' then UpperLimVal='  NE';
  mediantext = cat(strip(put(estimate, 5.1)), " (", LowerLimVal, " - ", UpperLimVal ,
")");
  keep STRATUM mediantext;
run;

data Censoredsummary2;
  set Censoredsummary;
  PctFailed = cats("(",put(((Failed/Total) * 100),8.1), "%)");
  PctCensored = cats("(",put(((Censored/Total) * 100),8.1), "%)");
  Total_text = strip(put(Total, best.));
  Event_text = catx(" ", put(Failed, best.), PctFailed);
  Censored_text = catx(" ", put(Censored, best.), PctCensored);
  keep stratum Total_text Event_text Censored_text;
run;

/* Merging censored summary with quartiles, to obtain final (bottom) table on graphic
*/

proc sql;
  create table censored_quartile as
  select a.*, b.mediantext
  from Censoredsummary2 as a inner join Quartiles2 as b
  on a.stratum = b.stratum;
quit;

data atrisk;
  set Productlimitestimates;
  Atrisk = strip(put(left, best.));
  rename STRATUM = classatrisk;
  rename Timelist = Tatrisk;
  keep STRATUM Timelist atrisk;
run;

/* Deriving minimum survival value and merging it back, so that KM plot goes
horizontally until the end.
Previously it ended quite prematurely. */
proc sql;
  create table minimum_survival_value as
  select trtpn, min(SURVIVAL) as min_survival
  from outsurv
  where survival ne .
  group by trtpn;
quit;

data outsurv2;
  merge outsurv minimum_survival_value;
  by trtpn;
  if survival = . then survival = min_survival;
run;

/* Creating final dataset */
data outsurvfinal;
  merge outsurv2 atrisk censored_quartile(rename =( Stratum = stratum_table));
  if _CENSOR_ = 1 then CENSOR_SURVIVAL = SURVIVAL;
run;

/* Creating Macro Variables to display table of events and censoring etc. */
proc sql;
```

```
  select Total_text, Event_text, Censored_text, mediantext
  into:Total_text1 - :Total_text2, :Event_text1 - :Event_text2, :Censored_text1 -
:Censored_text2, :mediantext1 - :mediantext2
  from censored_quartile;
quit;

/* Creating Macro Variables to display log rank p-value */
proc sql;
select put(ProbChiSq, PVALUE6.4) into: log_rank_pvalue
from homtests;
quit;

proc template;
define statgraph kmtemplate;
nmvar max interval;
mvar Total_text1 Event_text1 Censored_text1 Total_text2 Event_text2 Censored_text2
Mediantext1 Mediantext2 log_rank_pvalue;
  begingraph / designwidth=896 designheight=672 border = false;
    layout lattice / rows = 3 rowweights = (0.71 0.13 0.16) rowgutter = 15
columndatarange=unionall;
      layout overlay / xaxisopts = (label = "Survival Time (Days)" linearopts
=(viewmin = 0 viewmax = max tickvaluesequence = (start = 0 end = max increment =
interval)))
                                    yaxisopts = (label = "Survival Probability")
                                    walldisplay = all;

        stepplot x = aval y = survival / group = stratum  name="Survival"
legendlabel="Survival" lineattrs = (thickness = 2px);
        scatterplot x = aval y = censor_survival / group = stratum markerattrs =
(symbol = plus size = 10px);
        discretelegend "Survival" / location = inside across = 1 down = 3 autoalign =
(topright);
        entry "Logrank: p = " log_rank_pvalue / autoalign = (bottomleft);
      endlayout;

      cell;
        cellheader;
          entry halign = left "Subjects at Risk:";
        endcellheader;
        layout overlay / xaxisopts=(display = none linearopts =(viewmin = 0 viewmax =
max tickvaluesequence = (start = 0 end = max increment = interval)))
                                     yaxisopts =(display = (tickvalues) type =
discrete reverse=true) border = false;

          scatterplot x = Tatrisk y=eval(ifc(classatrisk = 1, "Treatment 1", "")) /
markercharacter = eval(ifc(classatrisk = 1, atrisk, ""));
          scatterplot x = Tatrisk y=eval(ifc(classatrisk = 2, "Treatment 2", "")) /
markercharacter = eval(ifc(classatrisk = 2, atrisk, ""));

        endlayout;
      endcell;

      layout overlay / xaxisopts=(display=none);
        layout gridded / columns=5 rows = 3 border = true;
          entry "Arm"; entry "No. of Subjects"; entry "Events"; entry "Censored";
entry "Median Survival (95% C.I.)";
                entry "Treatment 1"; entry Total_text1; entry Event_text1; entry
Censored_text1; entry Mediantext1;
                entry "Treatment 2"; entry Total_text2; entry Event_text2; entry
Censored_text2; entry Mediantext2;
              endlayout;
      endlayout;
    endlayout;
  endgraph;
end;
run;

proc format;
value trt     1 = "Treatment 1"
                      2 = "Treatment 2";
run;

proc template;
define style styles.KJHPlot;
parent = Styles.listing;
style GraphFonts from GraphFonts
         "Fonts used in graph styles" /
         'GraphDataFont' = ("<sans-serif>, <MTsans-serif>",8pt)
         'GraphValueFont' = ("<sans-serif>, <MTsans-serif>",8pt)
         'GraphLabelFont' = ("<sans-serif>, <MTsans-serif>",8pt);
CLASS graphWalls / FRAMEBORDER=off;
end;
run;

ods graphics / reset = all height = 3in width = 4.5in imagename = "&progname.";
ods listing gpath = "&outpath." image_dpi = 200 style = styles.KJHPlot;

proc sgrender data = outsurvfinal template = kmtemplate;
format CLASSATRISK stratum trt. ;
run;

ods graphics / reset = all;
```