Data Visualization with SAS/Graph®

Keith Cranford
Office of the Attorney General, Child Support Division

Abstract

With the increase use of Business Intelligence, data visualization is becoming more important as well. SAS/Graph provides many tools to perform data visualization quite well. Some procedures, particularly the new Statistical Graphics procedures, can be used for this function without much data manipulation, while other procedures may require a bit more work.

This paper will present several data visualization techniques, such as cycle plots, heatmaps, dot plots and deviation graphs. The use of these techniques will be illustrated along with SAS/Graph code to generate the visuals. In some cases alternative approaches are presented.

Data Visualization

The increased use of Business Intelligence has brought data visualization a higher profile. What is data visualization, though? One source defined data visualization as the process of representing abstract business or scientific data as images that can aid in understanding the meaning of the data.[1] Stephen Few uses "data visualization as an umbrella term to cover all types of visual representations that support the exploration, examination, and communication of data." [2] Both of these definitions focus on two key aspects: (1) use of visuals, and (2) insight. Data visualization must provide a way to gain new insight into the data that it is trying to represent. It is not just a pretty picture, but must convey meaning to the consumer.

There are many examples of these visualizations, many of them developed years ago but not have heightened use in this new BI environment. This paper will present three of these – heat maps, dot plots and cycle plots. Each of these will be accompanied with SAS code to create them.

Heat maps

A heat map is a visual display that encodes quantitative values as variations in color. A common, and literal, example is the temperature map you see in USA Today. Quite often a geographical map is used, but this is not necessary for using this technique. A great advantage of heat maps is its ability to convey a large amount of information in a relatively small space in a way that patterns can be identified fairly quickly (if they exist).

Figure 1 is an example of a heat map of average monthly temperature for United States climate divisions as determined by the National Climatic Data Center for 1931-2000. The darkest blue sections denote

[1] www.bitpipe.com/tlist/Data-Visualization.html
[2] Few, Stephen. Now You See It: Simple Visualization Techniques for Quantitative Analysis. Page 12.

average temperatures below 40°, whereas the darkest red is for average temperatures above 80°. This heat map uses months (time) as the second dimension.

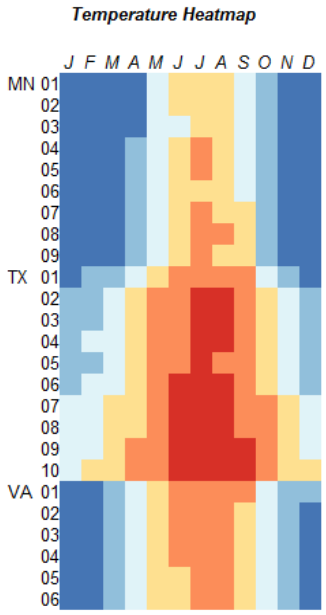

Figure 1: Heat map example

SAS/Graph could be used to create heat maps (I've actually done it), but an easier, niftier approach uses PROC REPORT. The idea is to conditionally assign background colors to the cells of the report.

First, the data needs to be coded for the ranges. An easy way to do this is using a user-defined format for the ranges and creating a coded variable with a put function. This is done in the code below. The source data includes average temperature variables (temp1, temp2, …) for each month. A series of coded variables (c1, c2, …) are created using a put function and the user-define $range format. Additionally, a label variable, c0, contains the division within a state.

```
proc format ;
   value range
      .          = '0'
      low -< 40  = '1'
      40 -< 50   = '2'
      50 -< 60   = '3'
      60 -< 70   = '4'
      70 -< 80   = '5'
      80 - high  = '6' ;
   value $st
      '21' = 'MN'
      '41' = 'TX'
      '44' = 'VA' ;
run ;

%macro heatmap ;

   data test(keep=state c0-c12) ;
```

```
    set input ;
    where state in ('41','21','44') and strip(year)='31-00' ;
    length c0 $ 4 c1-c12 $ 2 ;
    c0 = substr(site,3,2) ;
    %do i=1 %to 12 ;
      c&i=put(temp&i.,range.) ;
    %end ;
    output ;
  run ;
```

Once the data is encoded, PROC REPORT can display the heat map. This is accomplished through conditionally assigning the background color in the style with a CALL DEFINE. In addition to the color, the width of the cell is set as well as using a non-proportional font family. This makes the cells in the map a uniform size. Finally, the actual cell value is set to blank (v1, v2, …), so that no text is printed, only the cell background color.

```
  title  h=10pt "Temperature Heatmap" ;
  proc report data = test nowd
     style(report) = {frame=void rules=none cellspacing=0 cellpadding=0}
     style(header) = {fontsize=10pt fontweight=medium color=cx000000}
     style(column) = {fontsize=10pt verticalalign=center};
     column state c0-c12 v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 ;
     define c0 / center ' ' style={color=cx000000} ;
     define state / order ' ' format=$st. style={width=18pt} ;

  %do a=1 %to 12 ;
    define c&a. / noprint ;
    define v&a. / computed center
        "%sysfunc(putn(%sysfunc(mdy(&a.,1,2000)),monname1.))";
    compute v&a. / char length=1 ;
      v&a. = ' ' ;
      if c&a. = '1' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cx4575B4]");
      else if c&a. = '2' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cx91BFDB]");
      else if c&a. = '3' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cxE0F3F8]");
      else if c&a. = '4' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cxFEE090]");
      else if c&a. = '5' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cxFC8D59]");
      else if c&a. = '6' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cxD73027]");
      else if c&a. = '0' then call define(_col_, "style",
        "style=[fontfamily='Courier' width=12pt
         backgroundcolor = cxE0E0E0]");
    endcomp ;
  %end ;
```

```
   run ;

%MEND heatmap ;

%heatmap
```

Colors should be chosen that work well together. There are many resources for this, but one of the best is Colorbrewer (http://colorbrewer2.org/). This site allows you to set the number of categories and the type of scale that is used. In the example above there were six categories and a divergent scale was used. This allowed the heat map to differentiate easily the two ends of the scale – hot or cold. This could also be used when there is a comparison to a goal.

Dot Plot

Dot plots are essentially bar charts with the bars replaced by dots. This allows comparisons of a greater number of groups and easier pair-wise comparisons. These plots are less cluttered than a similar bar chart and follows Edward Tufte's principle of minimizing data ink very well.

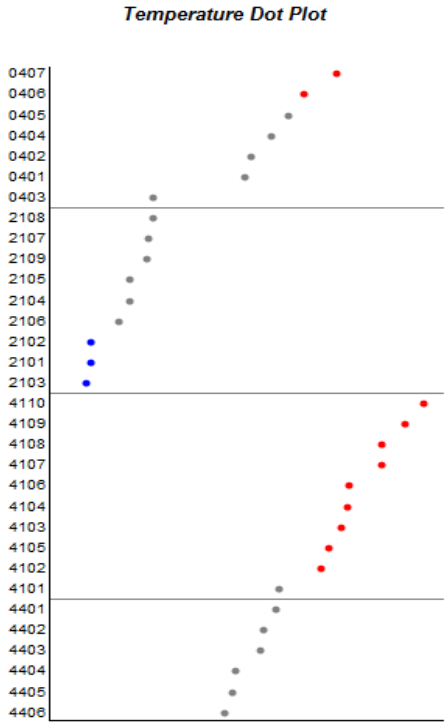

Figure 2: Dot plot example

Figure 2 is an example of a dot plot comparing average annual temperature at various climate divisions within four states (CA, MN, TX, VA). The divisions are ordered by the average temperature within each state. Break points in the temperatures have also been color-coded to better distinguish between low, medium and high temperatures. Some interesting aspects of this plot are the low value in one division in California (0403), which is in the northwestern part of the state bordering Nevada, the lower value in

the panhandle of Texas (4101) and the very low values in the extreme north section of Minnesota (2101 – 2103).

SAS/Graph can be used to produce dot plots fairly easily using PROC GPLOT. The vertical axis is used for a grouping variable. The DATA step is used to create a sequential variable (rank) that can be plotted. A user-defined format is then used to label the variable. In the example below a macro variable is used to identify points to be used as dividers in the graph, in this case for the different states.

If traffic-lighting of the graph is desired, separate response variables are needed. It is a simple matter in the DATA step of conditionally creating these variables based on the original response variable. These new variables would then be used in the plot.

The data may be sorted by the response variable or by the grouping variable, depending on how the data is being used. In the example the data is sorted by temperature within each state. This gives the patterns shown in Figure 2. However, if you were comparing several measures on different graphs, sorting by the climate divisions would allow easier comparisons by climate divisions.

```
proc sort data=input out=sort ;
     where state in ('41','21','44','04') and strip(year)='31-00' ;
   by descending state temp13 ;
run ;

data ranks ;
   set sort end=last ;
   by descending state ;
   length rlist $ 500 ;
   retain rlist ;
   rank + 1 ;
   if temp13 < 40 then low=temp13 ;
   else if temp13 < 60 then medium=temp13 ;
   else high=temp13 ;

   if last.state and not last then do ;
     rlist = strip(rlist) || put(rank+.5,6.1) ;
   end ;
   if last then call symputx('vref',rlist) ;
run ;

data fmt(keep=start fmtname type label) ;
   set ranks(keep=rank site) ;
   length label $ 10 ;
   retain fmtname 'rank' type 'N' ;
   start = rank ;
   label = site ;
run ;

proc sql noprint ;
   select max(rank) into : num
   from ranks ;
quit ;
```

```
proc format cntlin=fmt ;
run ;
```

Once the data is formed, PROC GPLOT can be used to plot the grouping variable (rank) by the response variable(s) (low, medium, high) . The OVERLAY option is used to superimpose the response variables, and the dividers are displayed through the VREF option using the list of points from the macro variable created above. The grouping variable label results from using the user-defined format .

```
ods html file='D:\Data2\SCSUG10\dotplot.htm' style=journal ;

goptions xpixels=300 ypixels=500
   htext=8pt device=activex ;
symbol1 v=dot c=blue height=5pt ;
symbol2 v=dot c=gray height=5pt ;
symbol3 v=dot c=red height=5pt ;
axis1 minor=none label=none /*major=none value=none*/ ;
axis2 minor=none label=none major=none order=(1 to &num.)
value=(height=8pt) ;
title 'Temperature Dot Plot' ;
proc gplot data=ranks ;
   plot rank  * (low medium high)
      / noframe overlay
      haxis=axis1
      vaxis=axis2
      vref=&vref.
      cvref=gray
   ;
   format rank rank. ;
   label rank='Site'
      low='Average Temp';
run ;
quit ;

ods html close ;
```

Cycle Plots

Cycle plots show overall patterns across cycles as well as the behavior within each subseries. Developed by William Cleveland and his colleagues to study seasonal patterns, these plots provide a way to see two things at once. A plot of the average cycles gives a view of the overall pattern across cycles (for example, months). Then within a cycle the subseries plots show whether the patterns within each cycle is changing over time.

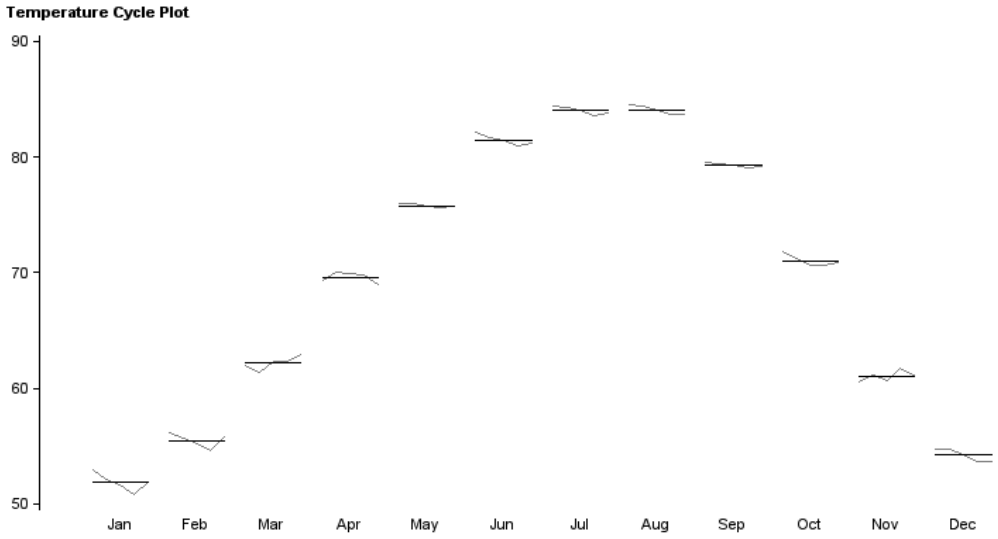

Figure 3:Cycle plot example

Figure 3 displays a cycle plot of monthly temperatures over a span of years in Central Texas. The horizontal lines indicate the average temperature of this period for each month; the line plots show the individual temperatures (actually 30-year averages) over this time span. The cycle plot shows the annual pattern of average temperatures around 85. Within each month the patterns are fairly stable (not surprising since these are 30-year averages), although the winter months show more change.

SAS/Graph can again be used to produce cycle plots, but this requires some data preparation. The general idea will be to produce a series of graphs for the monthly averages and another series for the monthly subseries. Then PROC GPLOT is used to overlay all of these graphs.

The initial DATA step transposes the data to have one observation per month. The interval variable identifies the month and is used for grouping the data in subsequent steps. PROC MEANS creates a data set SUMMARY containing the average temperature for each month over the entire time period.

```
%macro cycle_plot ;

  %let axis_val=' ' 'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun'
   'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec' ' ' ;

  data temp(keep=site interval date temp)  ;
    set input ;
    where site='4107' and strip(year) ne '31-00' ;
    yr = input(scan(year,2,'-'),2.) ;
    array temps {12} temp1-temp12 ;
    do interval=1 to 12 ;
      date = mdy(interval,1,yr) ;
      temp = temps[interval] ;
      output ;
    end ;
    format date mmyys7. ;
  run ;
```

```
proc means data=temp noprint nway ;
   class interval ;
   var temp ;
   output out=summary mean=avg_temp ;
run ;
```

PROC SQL adds the average monthly temperatures to individual monthly values.

```
proc sql ;
   create table new as
   select a.*, b.avg_temp as level
   from temp a, summary b
   where a.interval=b.interval
   order by interval, date
;
quit ;
```

The data is grouped at the interval (month) values. To produce the subseries graph the data values need to be "spread" out around the interval values. This is achieved by creating a new interval variable from adjusting the interval values by a small amount. First, PROC FREQ is used to determine the number of data points at each interval value. This is then used to create two MACRO variables: INCR holds the increment between each point and MIDDLE holds the middle value. These are computed based on the maximum number of points at each interval value, which in this example is 5. However, this code allows for an unequal number of points at each interval.

```
proc freq data=new noprint ;
   table interval /out=count(keep=interval count) ;
run ;

proc sql noprint ;
   select round(.9/max(count),.01), (max(count)+1)/2 into: incr, : middle
   from count ;
quit ;
```

Two new sets of variables are created in a DATA step. The ACROSS variables contain the monthly averages and the WITHIN variables contain the monthly temperatures. Additionally, a new interval variable (INTERVAL2) is computed by adding small increments around each interval value using the MIDDLE and INCR macro variables from above.

```
data new2 ;
   set new ;
   by interval ;

   array across {12} ;
   array within {12} ;
```

```
    if first.interval then t=0 ;
    t+1 ;
    across[interval] = level ;
    within[interval] = temp ;
    interval2=interval + (t-&middle.)*&incr. ;
  run ;
```

PROC GPLOT with an OVERLAY option is used to create the cycle plot, plotting all of the WITHIN and ACROSS variables against the INTERVAL2 variable. The first set of SYMBOL statements indicate that each monthly subseries are joined with gray lines, and the second set has each average monthly value displayed as a black line.

```
  goptions reset=all device=png xpixels=750 ypixels=400 htext=8pt ;
  %do i=1 %to 12 ;
    symbol&i i=join c=gray line=1 width=1.5 ;
  %end ;
  %do i=13 %to 24 ;
    symbol&i i=join c=black line=1 ;
  %end ;

  axis1 minor=none value=(' ' 'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun'
    'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec' ' ')
    major=(height=0.01 color=white) label=none style=0 ;
  axis2 minor=none major=(number=5) label=none;
  title2 h=8pt j=l " Temperature Cycle Plot" ;
  proc gplot data=new2 ;
    plot (within: across:) * interval2
      / noframe overlay
        haxis=axis1
        vaxis=axis2;
  run ;
  quit ;

%MEND cycle_plot;

%cycle_plot
```

SAS Statistical Graphics can also be used to produce cycle plots. This method is a little more straight-forward, but provides less control over the output. The previous example is shown using SAS Statistical Graphics in Figure 4.

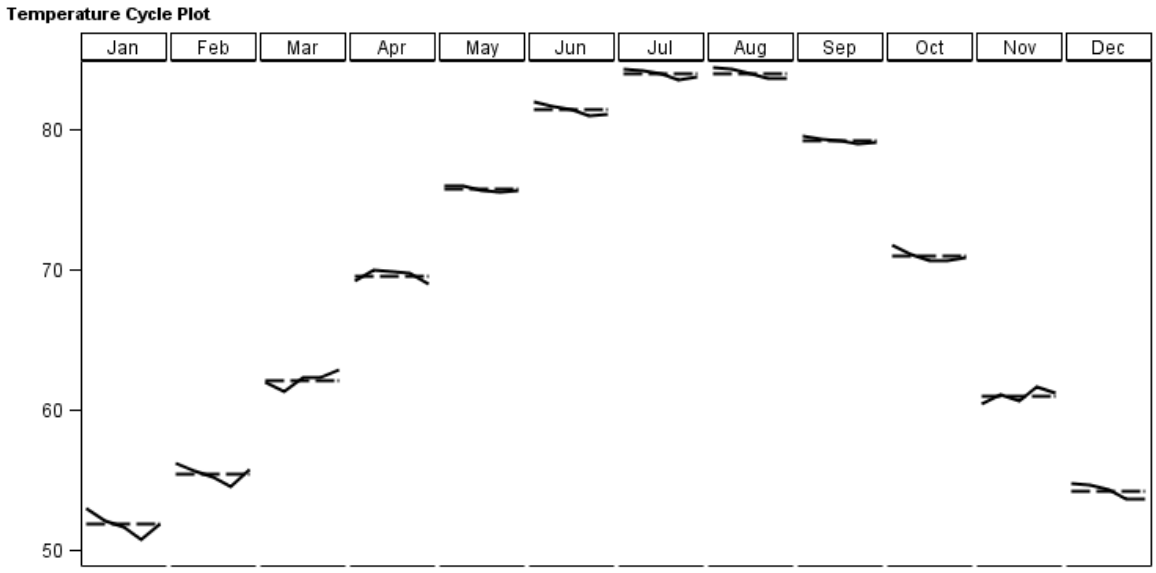

Figure 4: Cycle plot example using SAS Statistical Graphics

The initial DATA step and PROC MEANS are exactly the same as the previous example. These are used to transpose the data and compute the average monthly temperatures.

```
data temp(keep=site interval date temp)  ;
   set input ;
   where site='4107' and strip(year) ne '31-00' ;
   yr = input(scan(year,2,'-'),2.) ;
   array temps {12} temp1-temp12 ;
   do interval=1 to 12 ;
      date = mdy(interval,1,yr) ;
      temp = temps[interval] ;
      output ;
   end ;
   format date mmyys7. ;
run ;

proc means data=temp noprint nway ;
   class interval ;
   var temp ;
   output out=summary mean=avg_temp ;
run ;
```

PROC SQL is then used to expand the monthly average temperatures for each date in the original data.

```
proc sql ;
   create table summary2 as
   select a.interval, a.date, b.avg_temp as temp
   from temp a, summary b
   where a.interval=b.interval
   order by interval, date ;
quit ;
```

The transposed data and expanded summary data are then concatenated. A group variable is added to indicate whether the series in for the actual values or the monthly averages. For display purposes a user-defined format for the months is created.

```
data new ;
   set temp(in=a)
      summary2(in=s)
   ;
   if a then group='Actual';
   else if s then group='Level' ;
   year = year(date) ;
   label group='Group' ;
run ;

proc format ;
   value fmo
      1='Jan'
      2='Feb'
      3='Mar'
      4='Apr'
      5='May'
      6='Jun'
      7='Jul'
      8='Aug'
      9='Sep'
      10='Oct'
      11='Nov'
      12='Dec' ;
run ;
```

The ODS GRAPHICS statement initiates SAS Statistical Graphics. Among the available options are the size of the graphic (WIDTH, HEIGHT), not displaying a border around the entire graph (NOBORDER), and the type of image to create (IMAGEFMT).

The plot is then produced using PROC SGPANEL. The panels are identified by the PANELBY statement, where a graph is created for each month (INTERVAL). Some options include not putting a border around each panel (NOBORDER), where to place the header (COLHEADERPOS), and how much space to use between panels (SPACING). The VLINE statement indicates that a vertical line chart is to be drawn using the specified categorical variable. The REPONSE option indicates the numerical response variable to use, and the GROUP option indicates the distinct lines to draw.

```
ods graphics on /
     noborder width=750 px height=400 px reset=index
     imagename="cycle_plot"
     imagefmt=png ;

title h=8pt "Temperature Cycle Plot" ;
proc sgpanel data=new noautolegend;
    panelby interval / spacing=2 novarname onepanel
```

```
       rows=1 noborder colheaderpos=top ;
    vline year /
     response=temp group=group
     nostatlabel
     lineattrs=(color=black thickness=2);
  colaxis display=(novalues noticks) label=' ' ;
  rowaxis label=' ' ;
  format interval fmo. ;
run;
```

Notice that there are no specifications of the line types. This is done automatically with no way to control the line attributes, except for the color and thickness of all the lines. As mentioned, cycle plots can be produced using SAS Statistical Graphics fairly easily, but you have less control over some of the attributes of the graph.

Conclusion

This paper was not meant to be exhaustive on the subject of data visualization in SAS and SAS/Graph. However, by showing various techniques and how these can be implemented within SAS, it is hoped that you can explore ways to display data that enhances insight. So, go forth and explore!

Contact Information

Keith Cranford
Child Support, Office of the Attorney General of Texas
E-mail: keith.cranford@cs.oag.state.tx.us

References

Cleveland, William. The Elements of Graphing Data. Summit, NJ: Hobart Press, 1994.

Cleveland, William. Visualizing Data. Summit, NJ: Hobart Press, 1993.

Few, Stephen. Now You See It: Simple Visualization Techniques for Quantitative Analysis. Oakland, CA: Analytics Press, 2009.