Paper 2342-2014

Using SAS® GTL with 9.3 Updates to Visualize Data When there is Too Much of It to Visualize

Perry Watts, Stakana Analytics; Nate Derby, Stakana Analytics

ABSTRACT

Developing a good graph with ODS statistical graphics becomes a challenge when input data map to crowded displays with overlapping points or lines. Such is the case with the Framingham Heart Study of 5209 subjects captured in the **`sashelp.heart`** data set, a series of 100 booking curves for the airline industry, and interleaving series plots that capture closing stock values over a twenty year period for three giants in the computer industry. In the paper, transparency, layering, data point rounding, and color coding are evaluated for their effectiveness to add visual clarity to graphics output. Version 9.2 compatible Graph Template Language (GTL) plotting statements referenced in the paper include **`HISTOGRAM`**, **`SCATTERPLOT`**, **`BOXPLOT`**, **`SERIESPLOT`** and **`BANDPLOT`** plus layouts **`OVERLAY`**, **`GRIDDED,`** **`DATAPANEL`** and **`LATTICE`** that produce single or multiple-panel graphs.

GTL 9.3 updates have also been added to the paper. They include **`HEATMAPPARM`** for heat maps that add a third dimension to a graph via color, the **`DISCRETEOFFSET`** option that makes it possible to insert output from additional plotting statements into the area between ticks on a discrete axis, the **`RANGEATTRMAP`** statement for grouping continuous data in a legend, and **`DISCRETEATTRMAP`** for assigning colors to 100 series plot lines. If you have 9.3 SAS, you automatically have access to GTL. Since this is not a tutorial, you will benefit from reading introductory SAS papers and Warren Kuhfeld's book *Statistical Graphics in SAS®: An Introduction to the Graph Template Language and the Statistical Graphics Procedures.* Source code for this paper is available upon request.

KEY WORDS: Graph Template Language, SG Procedures, ODS Statistical Graphics, Versions 9.2 and 9.3 SAS®.

INTRODUCTION AND DESCRIPTION OF THE DATA THAT SUPPORT THE GRAPHS

An incremental approach is taken in this paper. We go from preliminary graphs that are less than optimal to output that conveys its message more effectively. Problems are defined along the way, and suggestions for solving them take advantage of the new features available in ODS statistical graphics. Source code is integrated into the discussion about the graphs; however the intention here is not to present a tutorial on the Graph Template Language. Instead, the paper focuses on how to graph dense and challenging data sets. For a discussion about an effective programming strategy to adopt when coding in GTL see Yang **(2011).**

Three data sets motivate the graphs presented in this paper. The first is the **`sashelp.heart`** data set that comes from the Framingham Heart Study of 5,209 men and women that have been followed biennially for the development of cardiovascular disease since 1948 **(Hubert et al.1983, p.968)**. The second data set from the airlines industry contains coordinates for an ordered progression of 100 time series plots; about 97 more than are typically graphed. The progression is ordered by date of departure which spans 100 consecutive Wednesdays between 12/10/08 and 11/03/10 **(Derby and Vo, 2010)**. Each plot in the progression is a cumulative distribution of booking lead times measured in days prior to the scheduled date for departure. What the graph is attempting to do is to trace the relationship between booking lead times and flight departure dates.

With the **`sashelp.stocks`** data only three not 100 series plots have to be accommodated. However, the visual display is still chaotic, because plot lines overlap. Plotting them separately in a **`DATAPANEL`** plot with three panels is not satisfactory, because the separation of plot lines makes data comparison difficult. To overcome the difficulty, we develop a new interleaving band plot and make pair-wise comparisons sending the output to another 3-paneled display.

HEART DATA: DEALING WITH DATA POINT OVERLAY

INITIAL VERSION:

The original graph of the heart data appears in a publication that introduced Graph Template Language to attendees at the 2009 SAS Global Forum **(Matange et al., 2009)**. The graph displays heights and weights for participants in the Framingham Heart Study as a scatter plot with marginal histograms. The graph highlights the need for dealing with symbol overlap that hides the full data display from the viewer. Cleveland recommends combining unfilled circles as plotting symbols with jittering to create partial overlap in a graphics display that has total overlap **(Cleveland, 1994, p.158).** Unfortunately this method won't work in Figure 1, because overlap in the raw data is *both* partial *and* complete. In this situation, jittering would only move the problem slightly, elsewhere. Therefore, what the SAS developers did was to use filled circles as plotting symbols while setting transparency, a new capability in ODS graphics, to 95%. This method would work well with sparser data. Symbols would simply become progressively darker as overlay increases. However, it doesn't solve the problem here. Outliers are barely visible, and symbol overlay results in a scatter plot that contains a large, undifferentiated dark “blob” in the middle. While the marginal histograms offset the overlay in the scatter plot it is difficult to map histogram frequencies to indefinable grids in the scatter plot. There are also additional problems with this graph that are addressed in the first revision below.

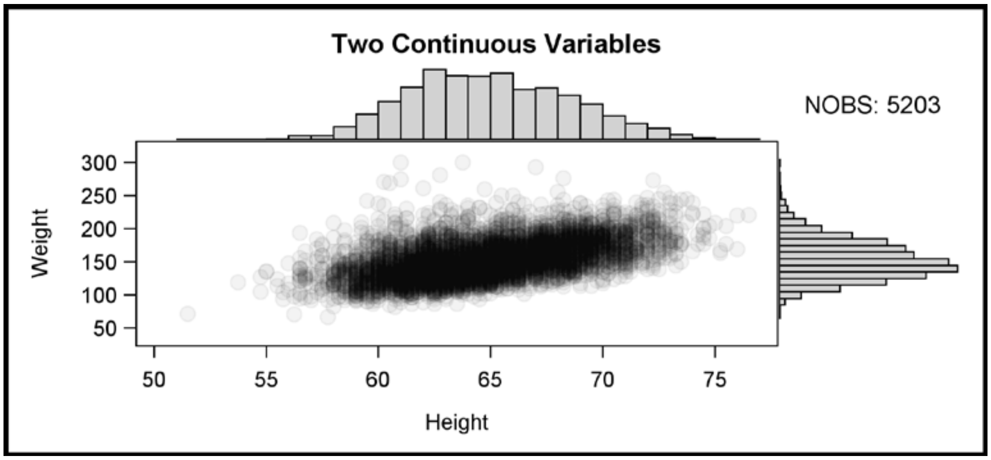


Figure 1. The graph is stretched out horizontally here and in the original publication to increase visibility. By examining the source code below, you will see that LAYOUT LATTICE is used to create a four-paneled graph. One panel is for the scatter plot, two separate panels are for the marginal histograms, and the fourth panel displays the number of observations for HEIGHT.

While source code is fragmentary in much of the paper, **`PROC TEMPLATE`** and **`PROC SGRENDER`** from SAS Sample #35172 **(SAS Institute, 2009a)** used by Matange et al. **(2009)** are reproduced almost in full below. Graphs produced in GTL are defined in a **`STATGRAPH`** template and generated in **`PROC SGRENDER`** where **`STATGRAPH`** template meets input data set. GTL works hierarchically and top-down in code blocks to produce a graph. In the source code that follows, the hierarchy is outlined by rectangle overlays:

```
PROC TEMPLATE;
  DEFINE STATGRAPH scatterhist;
    DYNAMIC xvar yvar title;
    BEGINGRAPH / DESIGNWIDTH=600px DESIGNHEIGHT=400px BORDERATTRS = (thickness=3px);
      ENTRYTITLE title;
      LAYOUT LATTICE / ROWS=2 COLUMNS=2 ROWWEIGHTS = (.2 .8) COLUMNWEIGHTS = (.8 .2)
                       ROWDATARANGE=union COLUMNDATARANGE=union
                       ROWGUTTER=0 COLUMNGUTTER=0;
        /*Row#1, Column#1) HISTOGRAM at X2 axis position */
          LAYOUT OVERLAY / WALLDISPLAY = (fill) XAXISOPTS = (DISPLAY=none)
                           YAXISOPTS = (DISPLAY=none OFFSETMIN=0);
            HISTOGRAM xvar / binaxis=false;
          ENDLAYOUT; /*OVERLAY*/
        /* Row#1, Column#2) TEXT ENTRY*/
          LAYOUT OVERLAY;
            ENTRY 'NOBS = ' EVAL( N( xvar ) );
          ENDLAYOUT; /*OVERLAY*/
        /* Row#2, Column#1) SCATTER PLOT */
          LAYOUT OVERLAY;
            SCATTERPLOT Y=yvar X=xvar / DATATRANSPARENCY=.95
                                        MARKERATTRS = (SYMBOL=circlefilled SIZE=11px);
          ENDLAYOUT; /*OVERLAY*/
        /* Row#2, Column#2) HISTOGRAM at Y2 axis position */
          LAYOUT OVERLAY / WALLDISPLAY = (fill)
                           XAXISOPTS = (display=none offsetmin=0)
                           YAXISOPTS = (display=none);
            HISTOGRAM yvar / ORIENT=horizontal BINAXIS=false;
          ENDLAYOUT; /*OVERLAY*/
      ENDLAYOUT; /*LATTICE*/
    ENDGRAPH; /*END GRAPH BLOCK*/
  END; /*END DEFINE BLOCK*/
RUN;
PROC SGRENDER DATA=sashelp.heart TEMPLATE=scatterhist;
  DYNAMIC yvar="Weight" xvar="Height" title="Two Continuous Variables";
RUN;
```

Figure 2. The SAS code for Figure 1 comes from SAS Institute **(2009a)**. Code highlighted in **blue** is reviewed below.

Examining SAS Code at the Block Level: Introducing LATTICE and OVERLAY Layouts

The outermost **DEFINE** block names the **STATGRAPH** template **scatterhist**. **DYNAMIC** variables that make the template re-usable are declared here and assigned values in **PROC SGRENDER**. Next up is the **GRAPH** block where the graph is sized and title is inserted.

The **LAYOUT** block in **scatterhist** supports both **LATTICE** and **OVERLAY** statements. **LAYOUT LATTICE** defines

> a multi-cell grid of graphs that can automatically align plot areas and tick display areas across grid cells to facilitate data comparisons among plots.... The number of cells must be predefined and *you define the content of each cell separately* **(SAS Institute, 2009c, p.155).** (Emphasis is added for later contrast to the DATAPANEL statement).

In **scatterhist**, a 2X2 matrix of cells holds four graphs in top down, left to right order. The cells are not equal in size. Instead, row #1 with the top-most histogram and text entry is ¼ the height of the plotting region and row #2 with the scatter plot takes up the remaining ¾ in height. Column widths also display the same ¼:¾ ratio. In this example, axes dimensions are equalized with **ROWDATARANGE** and **COLUMNDATARANGE** options, although the equality is difficult to see since the corresponding histogram X and Y axes are hidden.

LAYOUT OVERLAY is used to fill all four cells in the graph. **LAYOUT OVERLAY** "builds a 2D, single-cell graph by overlaying the results of statements that are contained in the layout block" **(SAS Institute, 2009c, p.39)**. If multiple plotting statements are issued in a single **OVERLAY** statement, the second plot is placed over the first, the third goes over the second and so on. While axes ranges are defined in Layout **LATTICE**, additional options are specified for the histogram axes in **LAYOUT OVERLAY**. **WALLDISPLAY**= is included as an axis option in **OVERLAY**, because the wall or plotting region is delimited by the rectangle that results when the four axes scale lines X1, X2, Y1 and Y2 are joined **(Cleveland, 1994, p. 33).** Wall outlines are removed from both marginal histograms in first graph from Figure 1 when **WALLDISPLAY** is set to **FILL**. By default, the plotting region is outlined in the scatter plot.

The **HISTOGRAM, ENTRY** and **SCATTERPLOT** statements are embedded in **LAYOUT OVERLAY** in this graph. The **HISTOGRAM** statement is used to create "a univariate histogram from input data" **(SAS Institute, 2009b, p.309)**. As explained in many basic statistics textbooks (e.g., **Siegel and Morgan (1996, p. 29)**, **McClave and Sincich (2003, pp. 31-33)**), a *histogram* displays frequencies (or percentages) of measurements falling into specified intervals as proportional rectangles.

The **ENTRY** Statement "displays a line of text in the plot area" **(SAS Institute, 2009b, p.641)** whereas the **SCATTERPLOT** statement is used to create "a scatter plot of input data" **(SAS Institute, 2009b, p.401)**. As explained by statistical textbooks (e.g., **Siegel and Morgan (1996, p. 517)**, **McClave and Sincich (2003, p. 86)**), a scatter plot is a two-dimensional plot of the data points, representing two numeric variables.

PROBLEMS WITH THE INITIAL VERSION ARE ADDRESSED

Unfortunately the count assigned to **NOBS** is wrong. By definition, points graphed in a scatter plot must have values for both X and Y coordinates that are within axes bounds. Missing values simply cannot be accommodated. When **NOBS** is set to **EVAL(N(height))** for the graph in Figure 1, three individuals with missing weight values are erroneously added to the count. A simple solution to this problem involves setting **NOBS** equal to **EVAL(N(xvar + yvar))**. Using the '**+**' operator returns a missing value when at least **xvar** or **yvar** are missing; just what we want! This is one instance where the **SUM** function would not give us the results we are looking for.

GTL summary statistic functions are used with **EVAL** such that **NUMBER = EVAL(function-name(numeric-column)).** There are 32 summary statistic functions available in GTL: pretty much a subset of the statistics that are provided by **PROC UNIVARIATE**. A full listing can be found in **SAS Institute (2009c, pp. 262-263)**. In addition to being used for labeling graphs, summary statistic functions can help compensate for the single data set restriction imposed upon GTL from **PROC SGRENDER. Kuhfeld (2010, pp. 30-31)**, for example, uses the **MEAN** and **STDERR** functions to define coordinates on the fly for the **ERRORLOWER** and **ERRORUPPER** options in the **BARCHARTPARM** statement. Nevertheless, while **EVAL** can often serve as a workaround, restricting GTL to a single input data set increases the complexity of the data processing task defined as programming strategy step #2 in **Yang (2011)**. We address this issue several more times in the paper.

A second problem with the graph in Figure 1 involves relative bin heights in the two marginal histograms. Since all points in a scatter plot must have values for both X and Y coordinates it follows that total counts for both histograms should be the same. That means bin dimensions for the two histograms should be comparable, but they are not.

There are two reasons why you can't visually compare the marginal histograms in Figure 1. First of all, the plotting region for the scatter plot is not squared off, and secondly, the hidden axes maximum values are different for the two histograms. For **weight**, the maximum is a little less than15% whereas it is roughly 11% for **height**. In Figure 3, a corrected version of the Figure 1 graph is presented along with a side-by-side version of the marginal histograms.

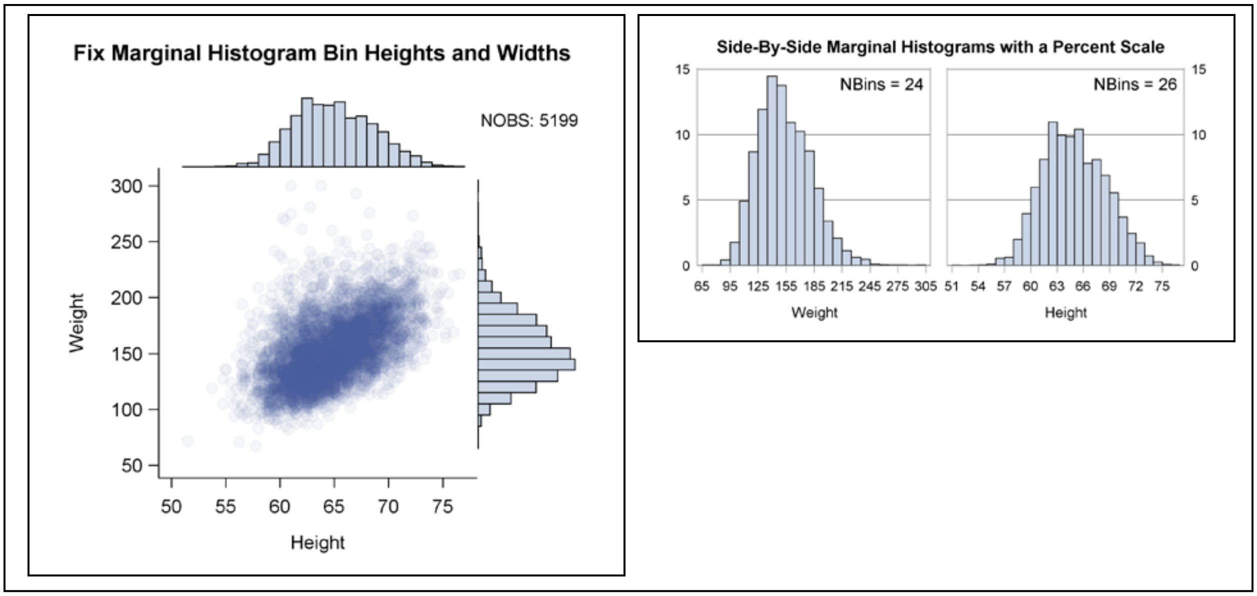


Figure 3. **`NOBS`** is now correct with 5,199 individuals having both heights and weights. Bin heights and widths in the marginal histograms are now comparable, since a check on the side-by-side graph confirms that **`weight`** is taller and slightly wider than **`height`**.

Relevant Code Fragments for Figure 3

Changes are made to **`STYLE MYSTYLE`** in **`PROC TEMPLATE`** to remove all wall borders with **`FRAMEBORDER`**. With their removal, bin ranges in both histograms become more visible.

```
PROC TEMPLATE;
  DEFINE STYLE MyStyle;
  PARENT = styles.statistical;
  CLASS graphWalls /
    FRAMEBORDER=off;
RUN;
```

To find out more about integrating style templates into GTL, see programming strategy step #4 in Yang **(2011)**.

An **`AXIS LENGTH`** option does not exist in GTL that would guarantee a square plotting region. Instead the whole graph is squared off with **`DESIGNWIDTH=DEFAULTDESIGNHEIGHT`**. The vertical axis has to accommodate titles, and the more titles, the greater the disparity between axes lengths. Nevertheless this is the best that can be done.

```
PROC TEMPLATE;
  DEFINE STATGRAPH scatterhist;
    DYNAMIC xvar yvar title;
    BEGINGRAPH / DESIGNWIDTH=DEFAULTDESIGNHEIGHT;
    ...
    ENDGRAPH;
  END;
RUN;
```

A brief code addition to **`YAXISOPTS`** and **`XAXISOPTS`** for **`xvar`** and **`yvar`** extends the maximum viewable percent for the response axis in both histograms to 15%.

```
/* For the histogram at scatter plot's X2 axis */
  LAYOUT OVERLAY / XAXISOPTS=( DISPLAY=none )
                   YAXISOPTS=( DISPLAY=none OFFSETMIN=0 LINEAROPTS=( VIEWMAX=15 ) );
    HISTOGRAM xvar / BINAXIS=false;
  ENDLAYOUT;
/* Histogram at scatter plot's Y2 axis */
  LAYOUT OVERLAY / YAXISOPTS=( OFFSETMIN=0 DISPLAY=none )
                   XAXISOPTS=( OFFSETMIN=0 DISPLAY=none LINEAROPTS=( VIEWMAX=15 ) );
    HISTOGRAM yvar / ORIENT=horizontal BINAXIS=false;
  ENDLAYOUT;
```

APPLY ROUNDING TO THE SCATTER PLOT SQUARED-OFF VERSION OF THE GRAPH

Now with a squared-off graph that has a smaller plotting region, solving the overlay problem with the scatter plot becomes even more imperative. Instead of jittering, let's do the opposite and round the data so that there is only full symbol overlapping in the output. Unfortunately, the results displayed in Figure 3 are far from satisfactory.

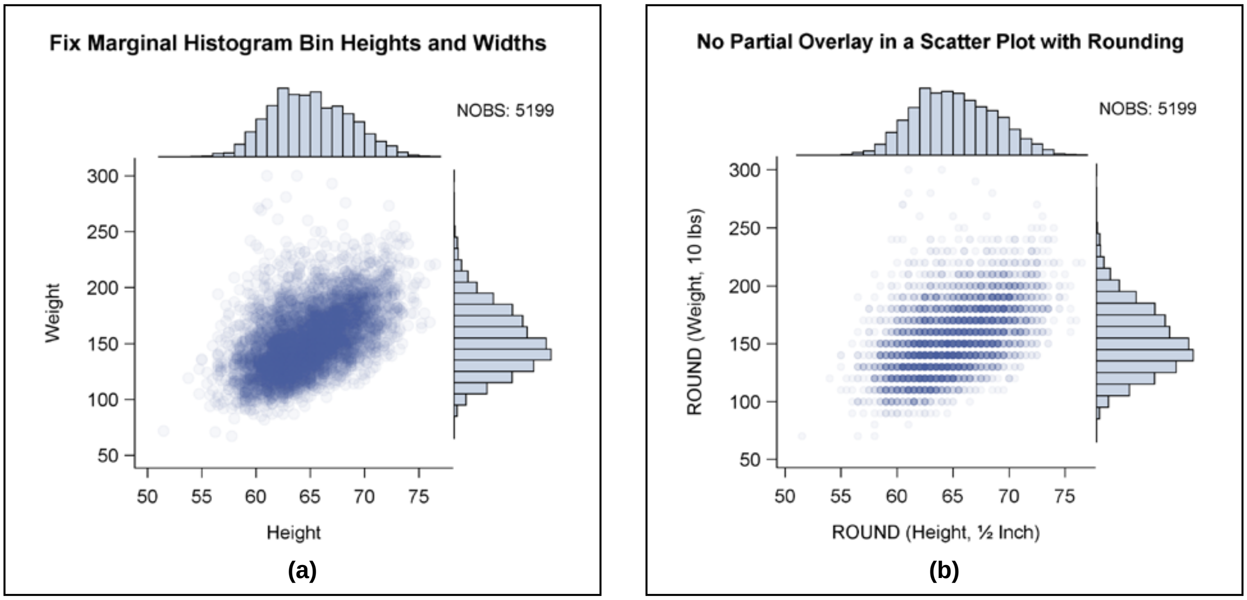


Figure 4. The graph on the left **(a)** is copied from Figure 3. In addition to rounding, the circle symbol size is reduced in the graph on the right **(b)**. With such small symbols, rounding does not convey additional information about the frequency distribution in the scatter plot. To put it bluntly, we have just created a digitized blob. With rounding, the marginal histogram for **`height`** in the digitized plot is also shifted slightly to the right.

Despite the disappointing results in Figure 4, let's continue to explore rounding as a new technique for managing densely packed data displays. As hinted above, jittering and rounding are inverses of each other. The inverse relationship between these functions is depicted in Figure 5:

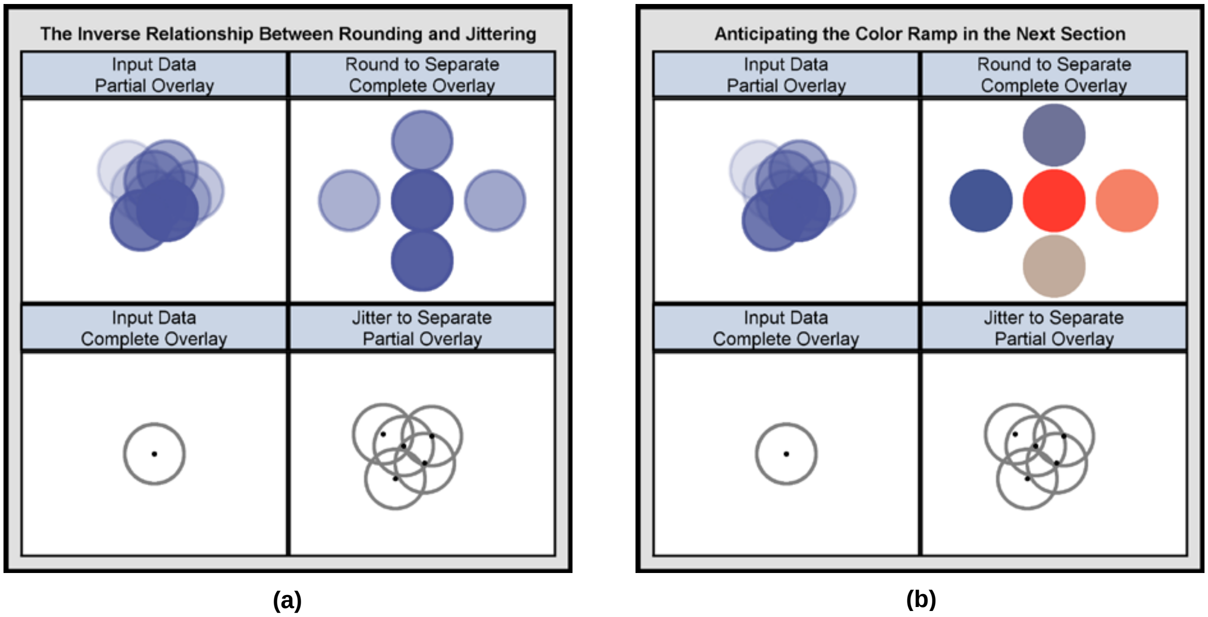


Figure 5. Both **(a)** and **(b)** demonstrate the inverse relationship between rounding and jittering where ROUND(partial overlay) Ù complete overlay and JITTER(complete overlay) Ù partial overlay. In the rounded panel for **(a)**, it is hard to rank frequencies when the blues are almost identical. In **(b)**, frequencies go from cold blue for low values to hot red for high ones.

Additional Comments about the Graph in Figure 5(a)

While jittering depends on hollow circles to promote visibility, partial overlay with transparency breaks down when dense data sets are plotted. It doesn't take too many partial overlays to generate dark blue regions in a graph. Nevertheless the problem is still not solved when data points are separated by rounding. Transparency looses its effect so that all you see are separate circles with slightly different shades of blue. In Figure 5(a), for example, it is not possible to rank the five points by their associated frequencies.

Long before the advent of desktop computers, Josef Albers described a study in *Interaction of Color* where 60% of the advanced painting students in a basic color class were unable to distinguish lighter colors from darker ones **(Albers1975, p. 12)**. As Albers points out, humans are much more adept at distinguishing between higher and lower musical tones. As a result, he concludes that we are all pretty much "tone deaf" when it comes to working with lightness scales in color. As early as 1975 he insists that we can develop more discriminating sensitivity by looking at color scales **(Albers 1975, p.16)**. However, illustrated color scales really only became available in 1990 with Tufte's publication of *Envisioning Information* **(Tufte 1990, pp. 81-96).** The print quality and color resolution in *Envisioning Information* is far superior to anything that has come since. To see displays of lesser quality color scales and spaces in SAS, we had to wait another 12 years for **Watts (2002, 2003, 2004).** Even the HLS (hue-lightness-saturation) color standard displayed in version 9.1.3 of the SAS/GRAPH manual is printed in black and white **(SAS Institute 2004, p.98)**!

APPLY A SAS COLOR "RAMP" TO THE ROUNDED SCATTER PLOT

Since we are almost "tone deaf" when it comes to distinguishing different shades of the same color, we need an alternative color scale that will in Cleveland's view provide us with an:

> effortless perception of the *order* of the encoded quantities, and clearly perceived *boundaries between adjacent levels* (**Cleveland 1994, forward)**. (Our emphasis added).

Brewer (1994) achieves these goals when she uses diverging sequences of lightness steps from two hues to compare employment rates among labor forces in European countries. SAS translates Cleveland's and Brewer's insights into a three-color RAMP that is applied to the scatter plot in Figure 6.

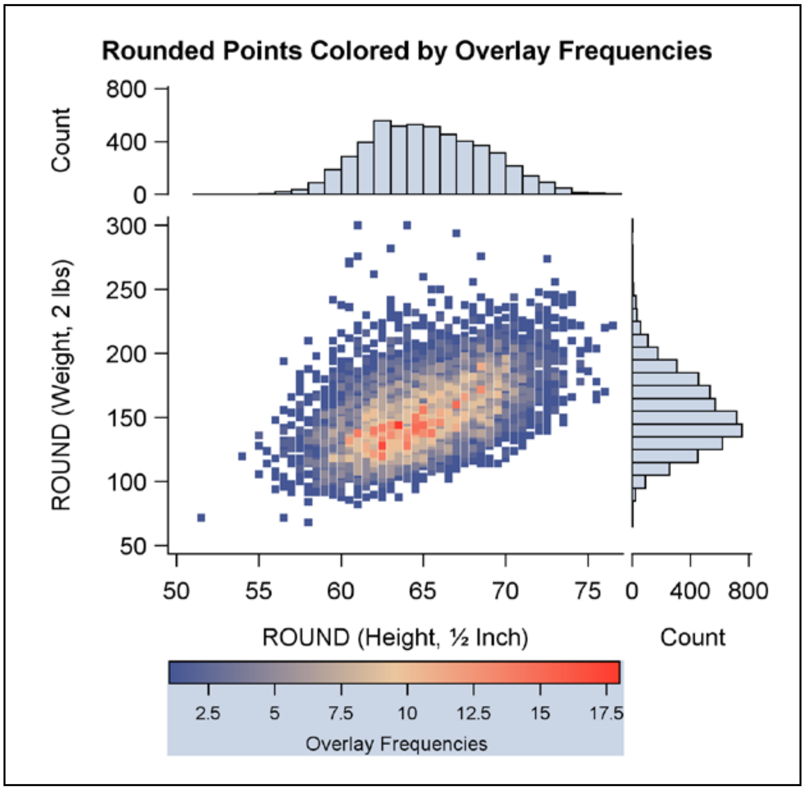


Figure 6. The three color ramp reflected in the legend is defined in the **`COLORMODEL`** option of the **`SCATTERPLOT`** statement.

Updates Made to the Graph in Figure 6

First noted is that the marker symbol in the graph is changed from **`CIRCLEFILLED`** to **`SQUAREFILLED`** so that histogram bin heights can be easily aligned with scatter plot frequencies. Now it is possible to see that the tallest bins line up with the brightest points in the data. Next, with the removal of transparency from the graph, all data points become fully visible. Full visibility makes it easy to line up histogram end bins with data outliers. Finally, the legend text is set off with a light blue background to distinguish it from the X-axis label and values. Adding a horizontal legend results in a graph that is stretched out in the horizontal direction similar to the one displayed in Figure 1. Histogram response axes are fully displayed to partially compensate for the distortion caused by the stretching. The more descriptive **`COUNT`** is selected as the axis scale in the display.

Relevant Code Fragments for Figure 6

Below, the derived data set **sTiedObs** (sorted-tied-observations) is created then used to generate the output displayed in Figure 6. The new variable, **count** is discussed when the code for **PROC TEMPLATE** is reviewed. Remember, more rounding leads to a greater incidence of tied observations. Less rounding means less distortion of the original data. Tradeoffs have to be made. In this instance **roundWeight**, originally an integer, can be off by 0 or 1 pounds; **roundHeight** recorded to the nearest ½ inch in the raw data can be off by at most ¼ inch.

```
DATA roundHeart;
  SET sashelp.heart;
  roundWeight=ROUND(weight,2);
  roundHeight=ROUND(height,.5);
RUN;
PROC SUMMARY DATA=roundHeart NWAY;
  CLASS roundWeight roundHeight;
  OUTPUT OUT=TiedObs (KEEP=roundWeight roundHeight _FREQ_ RENAME = (_FREQ_ = count));
RUN;
PROC SORT DATA=tiedObs OUT=sTiedObs;
  BY count;
RUN;
```

An ascending sort is applied to the data so that coordinates with the highest frequencies will be fully visible, since they are plotted last. The code fragment for **PROC TEMPLATE** below shows how the newly created variable **count** in data set **sTiedObs** is used to create the desired graphics output.

```
PROC TEMPLATE;
  DEFINE STATGRAPH scatterhist;
  ...
  /* histogram at X2 axis position */
      LAYOUT OVERLAY / ...;
        HISTOGRAM roundHeight / FREQ=count BINAXIS=false;
      ENDLAYOUT;
  /* scatter plot: */
      LAYOUT OVERLAY / ...;
        SCATTERPLOT
          Y=roundWeight X=roundHeight / name="colorByfreq" MARKERCOLORGRADIENT = freq;
        CONTINUOUSLEGEND "colorByfreq" / ...;
      ENDLAYOUT;
  /* histogram at Y2 axis position */
      LAYOUT OVERLAY / ...;
        HISTOGRAM RoundWeight / FREQ=count ORIENT=horizontal BINAXIS=false;
      ENDLAYOUT;
      ...
  END;
RUN;
```

MARKERCOLORGRADIENT is the option in the **SCATTERPLOT** statement that specifies the column (variable) used for mapping marker colors to a continuous gradient. Even though the **HISTOGRAM** statement works on raw data, it still supports a **FREQ** option with **FREQ=count**.

ADD THE BODY MASS INDEX TO THE ROUNDED SCATTER PLOT AND GROUP BY GENDER

An article published by the American Heart Association describes what happens over a 26 year period to subjects who participated in the Framingham Heart Study. Those studied had to be free of cardiovascular disease at their first medical examination. The paper concludes that "obesity, measured by **M**etropolitan **R**elative **W**eight, was a significant independent predictor of CVD, particularly among women" **(Hubert et al., 1983, p.968)**.

Unfortunately we are unable to underscore the study's findings graphically, because we lack complete information, and some of what we have is inconsistent. First off, there is no way to identify the 5,070 out of 5,209 men and women who were disease free at the beginning of the study. Also we cannot duplicate the SAS column MRW calculated as (*Actual Weight* /*Desirable Weight*) X100 by using an *adjusted* value for *Desirable Weight* supplied from Table 1 in the publication. We assume a different table with an *unadjusted* desirable weight was used to calculate MRW in the SASHELP.HEART data set.

What we can do, however, is to categorize study subjects by their BMI scores to confirm that as a group they were considerably overweight especially by standards for the 1950's. Also the axes are being rotated since the Body Mass Index is a measure of obesity (i.e. weight). The categorization is done graphically by plotting BMI curves over a heat map that replaces the rounded scatter plot in Figure 6. The output is displayed in Figure 7.

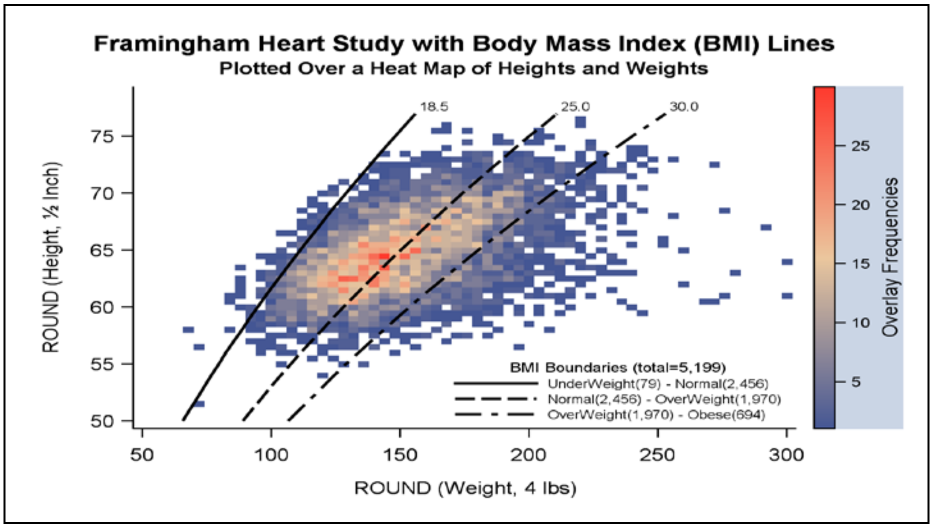


Figure 7. All the components of the graph are nested within a single **LAYOUT OVERLAY** statement. A single **SERIESPLOT** statement with a GROUP option is used to produce the three BMI boundary lines, and the heat map that replaces the scatter plot is generated with the **HEATMAPPARM** statement, new in version 9.3 SAS. Stretching the graph along the horizontal axis increases readability and does not compromise the results, since there are no marginal histograms in the display.

Due to paper-length restrictions, source code for the **SERIESPLOT** statement is not being reviewed in this section. The program listing is available upon request, however. If you are able to look at the code, keep in mind that only a single data set can be processed in **PROC TEMPLATE**. Two would be preferable: one for the heat map and a second for the BMI boundary lines.

The most convenient way to cope with the single data set restriction is to concatenate multiple data sets with a **SET** statement. With concatenation, different variables can be constructed independently for exclusive assignment to the various plotting statements. For example, **height_BMI** and **weight_BMI** are assigned to **X** and **Y** in the **SERIESPLOT** statement whereas **X** and **Y** point to **roundHeight** and **roundWeight** in the **SCATTERPLOT** statement. When SAS is plotting **height_BMI**, the value for **roundHeight** in the same observation is missing, since it is not defined prior to concatenation. In this instance, the missing value is our friend; nothing is plotted by mistake.

Concatenation rather than any form of merging or joining greatly simplifies the data processing often required for constructing complicated graphs. It can also be used to extend the reach of an **ANNOTATE** data set in conventional SAS/GRAPH software. To our knowledge, this technique has not been described elsewhere in the SAS literature.

Relevant Code Fragments for Figure 7 that Feature the New HEATMAPPARM Statement

Below is source code that compares the **SCATTERPLOT** statement used to create the graph in Figure 6 with the new **HEATMAPPARM** statement used in Figure 7. Note that the color variable for the **SCATTERPLOT** statement is an *option* whereas it becomes a required *parameter* in the **HEATMAPPARM** statement. Also different is the **COLORMODEL** statement for the two statements. By default **THREECOLORALTRAMP** is assigned to the **SCATTERPLOT** statement whereas it switches to **THREECOLORRAMP** in the **HEATMAPPARM** statement. The switch means that **COLORMODEL** has to be explicitly set in the **HEATMAPPARM** statement to get a comparable color scheme in the legend. Also **XBINAXIS** and **YBINAXIS** unique for **HEATMAPPARM** are set to **false** so that identical axes ranges in the Figure 6 and 7 graphs can be produced.

```
/* Scatter Plot for Figure 6: */
    LAYOUT OVERLAY / ...;
      SCATTERPLOT
        Y=roundWeight X=roundHeight / MARKERCOLORGRADIENT=freq NAME=”colorByFreq”;
    ENDLAYOUT;


/* Heat Map for Figure 7: */
    LAYOUT OVERLAY / ...;
      HEATMAPPARM
        Y=roundHeight X=roundWeight COLORRESPONSE=freq /
                      COLORMODEL=threeColorAltRamp NAME="colorByFreq"
                      XBINAXIS=false YBINAXIS=false;
    ENDLAYOUT;
```

While the code and output from the scatter plot and heat map look pretty much the same, there are significant differences in how plotting symbols are defined. With the **SCATTERPLOT** statement, a square symbol is defined in the **STYLE** template as:

```
    CLASS graphDataDefault /
      MARKERSIZE=5px
      MARKERSYMBOL="squarefilled";
```

To replace squares with circles in a scatter plot, all that has to be done is to set **MARKERSYMBOL** to **"circlefilled"**. When it comes to the **HEATMAPPARM** statement, however, the setting for **MARKERSYMBOL** in the **STYLE** template is ignored. Instead, rectangles are generated with dimensions that reflect the graph's aspect ratio and the data that are being plotted. By stretching the graph out horizontally and increasing **ROUNDWEIGHT** from two to four pounds in Figure 7, for example, the heat map produced supports short, wide rectangles.

AN ALTERNATIVE: MIXING RAW AND SUMMARY DATA WITH THE DISCRETEOFFSET OPTION

Peter Flom recommends combining bar charts with parallel box plots in a two-panel display as a way to solve the problem of point overlay that occurs when large data sets are graphed as scatter plots **(Flom, 2011)**. Flom uses 9.2 SAS to implement his solution. Starting with version 9.3 SAS, the new **DISCRETEOFFSET** option can be used to enrich Flom's basic graph. Now it is possible to insert output from additional plotting statements into the area *between* ticks on a discrete axis. In the Figure 8 graph, for example, the box plot and vertical scatter plot for female subjects in the upper panel are offset from the centered scatter plot for male subjects with an application of the **DISCRETEOFFSET** option. The vertical scatter plot for the male subjects is aligned with the X-axis ticks such that **DISCRETEOFFSET**=0, the underlying default value.

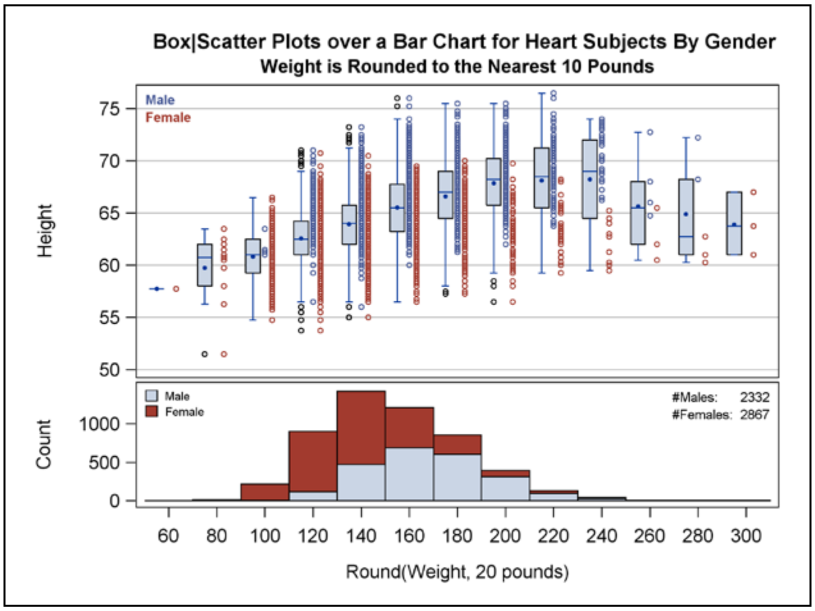


Figure 8. The common X axis in the two-panel display that uses a LATTICE layout is discrete. That means all unique values for ROUNDWEIGHT are fully enumerated along the horizontal axis. Regions between the major ticks can only be accessed by setting DISCRETEOFFSET to a non-zero number ranging from -0.5 to +0.5 where 0.5 represents half the distance between axis ticks.

Relevant Code Fragments for Figure 8 that Features the New DISCRETEOFFSET Option

Below is relevant source code for both panels in the Figure 8 graph of the Framingham Heart Study:

```
  LAYOUT LATTICE / ...
    COLUMNAXES;
      COLUMNAXIS / TYPE=discrete;
    ENDCOLUMNAXES;
/* UPPER PANEL */
    LAYOUT OVERLAY;
       BOXPLOT X=roundWeight Y=Height / DISCRETEOFFSET=-0.25;
       SCATTERPLOT X=roundWeight Y=HeightM;
       SCATTERPLOT X=roundWeight Y=HeightF / DISCRETEOFFSET=0.15
/* LOWER PANEL */
    LAYOUT OVERLAY;
       BARCHART X=roundWeight Y=count/ STAT=sum BARWIDTH=1 GROUP=sex GROUPORDER=descending;
    ENDLAYOUT;
  ENDLAYOUT;
```

All four plotting statements in the two-panel display from Figure 8 use both the **DISCRETEOFFSET** option and the variable **roundWeight** to determine the exact location of their X coordinates. When not specified, **DISCRETEOFFSET** equals zero. Therefore, to get the stacked bar chart displayed in the lower panel **DISCRETEOFFSET** is simply not set so that each bar in a **GROUP** is centered over a single axis tick. (Conventional group bar charts spread out the bars by setting **DISCRETEOFFSET** to a non-zero number).

By setting **BARWDITH** to 1, the bar chart in the lower panel of the Figure 8 graph is also made to look like a midpoint histogram. In addition, females are plotted over males when **descending** is assigned to **GROUPORDER**. By looking at the output in Figure 8 it is interesting to note that females, who are generally shorter than males, occupy both extremes of the weight scale.

AIRLINES DATA: WORKING WITH A PROGRESSION OF TIME SERIES PLOTS

In this section, we work with a time series plot that incorporates the requirement for pre-sorted data into its definition. As defined in **Brockwell and Davis (1996, p. 1)**), a *time series* is simply a set of observations indexed by time. The airlines data here are also "indexed by time, where the order of the observations matters" **(Derby and Vo, 2010).** In SAS, however, time series plots are implemented with the **SERIESPLOT** statement that accommodates both sorted and unsorted data. Since the language is so flexible, care must be taken to sort the input data by time or, if you have access to 9.3 SAS, to explicitly set **CONNECTORDER=XAXIS (SAS Institute, 2011, p.477)**. The default setting is **CONNECTORDER = XVALUES** for X values that can be in any order.

A time series *progression* orders the series plots by two time dimensions. In the case of the Airlines data, the input data are sorted first by departure date and then within each departure date by the number of days before departure that a passenger books a flight.

GRAPHING THE AIRLINES DATA IN A SINGLE CELL

In Figure 9(a), the departure date, one of the two time dimensions in the progression, is missing from the display. In Figure 9(b), color is used as an additional dimension to identify the series plots by departure date range.

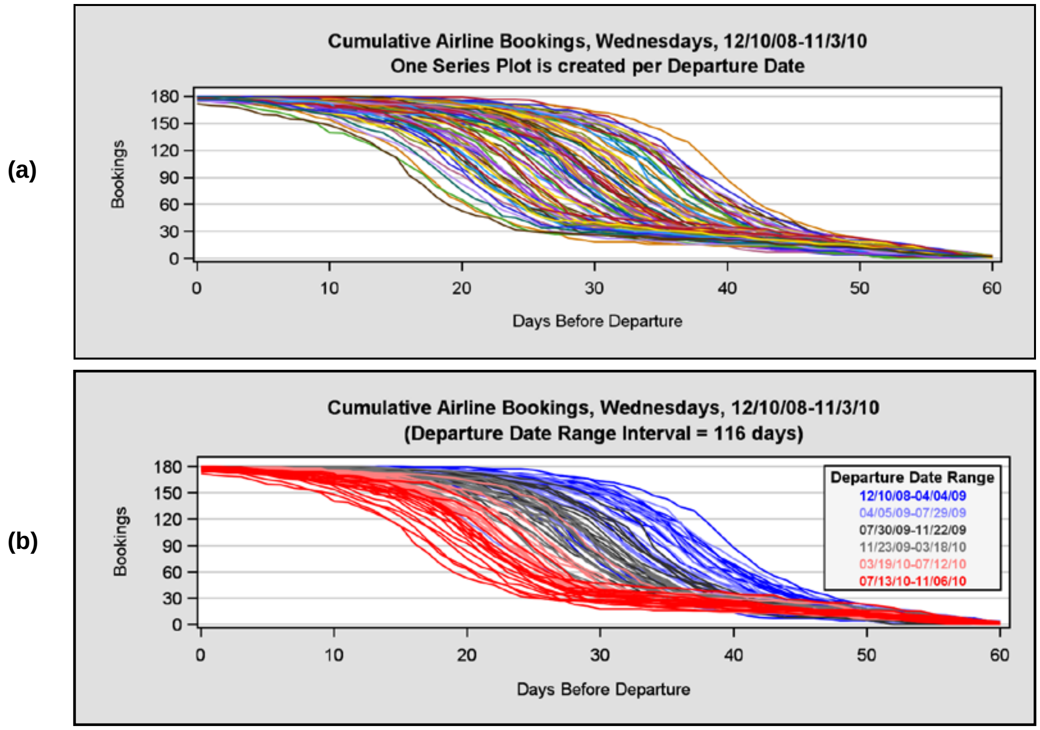


Figure 9. In **(a)**, there is no way to connect a series plot to its departure date. In **(b)**, color is based on departure date. The trend is obvious: in 2008-2009 passengers rushed out early to purchase their tickets whereas recent flights are booked at a more leisurely pace.

Relevant Code Fragments for Figure 9

A single **SERIESPLOT** statement is used in **PROC TEMPLATE** to create the 100 lines in Figure 9(a). In the code fragment below, the three dimensions from the input data are easily identified by variable name. They are **daysLeft**, **bookings** and **DDate** (for departure date). As a **GROUP** variable, each time the value for **DDate** changes a new plot line is generated. Line colors are also assigned on a cyclical basis from the **GRAPHDATA1 – GRAPHDATA12** style elements in the **STYLES.DEFAULT** template. All the time series plots have solid lines with **PATTERN=1**.

```
LAYOUT OVERLAY / ...;
   SERIESPLOT X=daysLeft Y=bookings / GROUP=DDate LINEATTRS = (PATTERN=1);
ENDLAYOUT;
```

Code for the graph in Figure 9(b) is more intricate with color being used as a fourth dimension to link departure date to the number of days before departure that a flight is booked. To add the fourth dimension in Version 9.2 SAS, variables **X1-X6** are created based on classification variable **byDateLbl**. For example, when **DDate** is between 12/10/08 and 04/04/09, **daysLeft** is assigned to **X1** and a point is plotted. When **DDate** is out of range, a missing value is assigned to **X1** and nothing is plotted. The same process involving **byDateLbl** determines the remaining assignments for **X2-X6.**

While **DDate** continues to define each of the 100 series plots, line color only changes when a new **SERIESPLOT** statement is issued. In addition, a legend created with **LAYOUT GRIDDED** is nested within **LAYOUT OVERLAY** along with the six **SERIESPLOT** statements. Each **ENTRY** statement in the **GRIDDED** layout becomes a row in the one-column table **(SAS Institute, 2009c, p.273)**. **LAYOUT GRIDDED** is used in preference to a **DISCRETELEGEND** statement in this graph, because text with a bold weight is thicker and more visible than discrete legend lines that in this case would only be one pixel wide. Plot lines and legend lines always have the same width. The airlines data call for narrow lines, because 100 series plots have to be crowded into a single display.

```
LAYOUT OVERLAY / ...;
  %do i= 1 %to 6;
    SERIESPLOT Y=bookings X=x&i / GROUP=DDate LINEATTRS = (COLOR=&&color&i PATTERN=1);
  %end;
  LAYOUT GRIDDED / COLUMNS=1 ...;
      ENTRY HALIGN=center TEXTATTRS = (WEIGHT=bold) "Departure Date Range";
   %do j = 1 %to 6;
      ENTRY HALIGN=center TEXTATTRS = (WEIGHT=bold COLOR=&&color&j) "&&Range&j";
   %end;
  ENDLAYOUT;
ENDLAYOUT;
```

With the advent of 9.3 SAS processing can be simplified by replacing the six SERIESPLOT statements that change plot line colors with an application of the new DISCRETEATTRMAP statement. DISCRETEATTRMAP is described in the next section.

PROBLEMS WITH DATE RANGES ARE SOLVED WITH AN APPLICATION OF RANGEATTRMAP

Although the graph in Figure 9(b) confirms the existence of a relationship between booking lead times and flight departure dates, there are serious problems with the graph. First, the only Wednesday in the legend is the first date. The remaining legend dates cannot be found in the input data set. Secondly, an arbitrary date range of 116 days making no business sense is calculated as:

$$Interval = \frac{Max\ Date - Min\ Date}{6} = 116\ days$$

What is needed here is a legend that reflects the mixed data type for the departure date variable. **DDATE** is *continuous* in that there are 100 unique values in the input data set. At the same time, **DDATE** is also *discrete* in that values are restricted to Wednesdays. Thursdays through Tuesdays simply do not exist in the data! If a continuous legend with a three-color ramp that reflects the 100 unique dates can be segmented such that only Wednesdays are labeled, we will be able to incorporate both the discrete and continuous aspects of the input data into our output.

But wait, there is a second problem! Continuous legends don't work with **SERIESPLOT** statements. They only work with **SCATTERPLOT**, **HEATMAPPARM**, and **CONTOURPLOTPARM** statements. What we do to solve this problem is to first create a scatter plot for the 100 departure dates and then set **VIEWMAX**, a continuous axis option, to a value *less than* the constant assigned to the **Y** parameter in the **SCATTERPLOT** statement. By restricting **VIEWMAX** in such a way, the scatter plot is effectively hidden from view. An X2 axis is also created for the scatter plot and hidden with **display=(LINE)**in the **X2AXISOPTS** option associated with the **LAYOUT OVERLAY** statement.

Now that we have figured out how to link a continuous legend to a hidden scatter plot, common colors have to be defined for both the legend and the series plots. Colors, in this example, are assigned by calling the **RGBcodes4ColorScale** macro twice to imitate the structure of GTL's three-color ramp. The first time the macro is called, a color scale is created that goes from the ramp's START to NEUTRAL colors, and the second call completes the a three-color ramp by going from the NEUTRAL to END colors. The two macro calls are listed below. Macro output is sent to the work data sets identified by the **DSID** parameter.

```
/* Go from START code to NEUTRAL code: blue to dark gray */
 %RGBcodes4ColorScale(sourceRGB=CX0000FF, destRGB=CX606060,
                      dsID=Start2Neutral, startobs=1, n=2)
/* Go from NEUTRAL code to END code: dark gray to red */
%RGBcodes4ColorScale(sourceRGB=CX606060, destRGB=CXFF0000,
                     dsID=Neutral2End, startobs=3, n=3)
```

Once created, **START2NEUTRAL** and **NEUTRAL2END** are concatenated to form a RAMP3 data set which is then sorted to eliminate the duplicate neutral color. RAMP3 ends up having just six records, one for each unique color. RAMP3 is used to assign colors to the 100 series plot lines *and* to define the legend via **RANGEATTRMAP**. An annotated version of **RGBcodes4ColorScale** with example calls can be found in the Appendix.

RANGEATTRMAP New in Version 9.3 SAS is Explained by Example

The **RANGEATTRMAP** statement is intricate, since its successful execution requires the completion of two additional steps. For a complete discussion with example, see the GTL Reference Manual **(SAS Institute, 2011, pp. 795-802)**. The source code below is used to generate the continuous legend displayed in Figure 10.

Step #1: Create an attribute map with **RANGEATTRMAP** that links the six RAMP colors to Departure Date Ranges:

```
RANGEATTRMAP NAME="ddateRange";
    %do i=1 %to %eval(&nGrpColors -1);
      RANGE &&sDate&i -< &&eDate&i  / RANGECOLOR=&&rColor&i;
    %end;
      RANGE &&sDate&nGrpColors - MAX  / RANGECOLOR=&&rColor&nGrpColors;
ENDRANGEATTRMAP;
```

Let's resolve the macro variables with **MPRINT** so that you can better see how the statement works:

```
RANGEATTRMAP NAME="ddateRange";
  RANGE 17876 -< 17988 / RANGECOLOR=CX0000FF
  RANGE 17988 -< 18107 / RANGECOLOR=CX3030B0
  RANGE 18107 -< 18226 / RANGECOLOR=CX606060
  RANGE 18226 -< 18345 / RANGECOLOR=CX954040
  RANGE 18345 -< 18464 / RANGECOLOR=CXCA2020
  RANGE 18464 - MAX    / RANGECOLOR=CXFF0000
ENDRANGEATTRMAP;
```

RANGEATTRMAP is always named, so that it can be identified in the second step where the **RANGEATTRVAR** statement is executed. **RANGEATTRMAP** also looks a lot like **PROC FORMAT**, but there are a few differences: **HIGH** is replaced by **MAX** in **RANGEATTRMAP**, and the "label", if you will, is always a color code. The numeric ranges and assigned colors in **RANGEATTRAMAP** translate to the Wednesday dates and color codes displayed in Figure 10.

Step #2: Create a named association between **RANGEATTRMAP** and a data set variable with **RANGEATTRVAR**:

```
RANGEATTRVAR ATTRVAR=rangeAssoc VAR=ddate ATTRMAP="ddateRange";
```

RANGEASSOC is the newly created association that is stored as a variable, **DDATE** is supplied by the input data set, and **DDATERANGE** is the name of the **RANGEATTRMAP** created in step #1. Steps #1 and #2 come just after **BEGINGRAPH** and before **LAYOUT OVERLAY** in the **STATGRAPH** template, whereas the last step, Step #3, is associated with the (hidden) **SCATTERPLOT** statement inside **OVERLAY**:

Step #3: The assignment to the **MARKERCOLORGRADIENT** option in the **SCATTERPLOT** statement is the named association created in step #2:

```
SCATTERPLOT Y=value190 X=ddateNum / name="scatter" MARKERCOLORGRADIENT=rangeAssoc ...;
CONTINUOUSLEGEND ="scatter" / ...;
```

If you look at the description for **MARKERCOLORGRADIENT** in the manual **(SAS Institute, 2011, p. 452),** you will see that the option can accommodate a third variable. In fact, such is the case when, **FREQ**, is assigned to **MARKERCOLORGRADIENT** in Figure 6. Here again is the code from Figure 6:

```
SCATTERPLOT Y=roundWeight X=roundHeight / MARKERCOLORGRADIENT=freq;
```

The variable **FREQ** comes from the input data set, and the color range is determined by the **THREECOLORALTRAMP** style element. On the other hand **RANGEASSOC** is more like a format where **DDATE** is mapped to a user defined color.

DISCRETEATTRMAP SIMPLIFIES CODING FOR THE SERIES PLOTS

In 9.2 SAS, 100 **SERIESPLOT** statements had to be issued to accommodate the six *continuous* ranges in dates for the RAMP colors. What follows is code for the **DISCRETEATTRMAP** statement that reduces the number of **SERIESPLOT** statements from 100 to just one! The code is a modification of that used for the population pyramid in *Off the Beaten Path: Create Unusual Graphs with GTL* **(Hebbar, 2012).**

```
DISCRETEATTRMAP NAME="ddateDiscrete";
    %do i=1 %to 100;
      /*DD=departure date*/
      VALUE "&&dd&i."  / LINEATTRS=(COLOR=&&color&i);
    %end;
ENDDISCRETEATTRMAP;
```

When macro variables are resolved you can see that the repeated codes for the individual plot lines conform to range definitions specified in the table insert for Figure 10. What might not be so obvious is that **DDATE**, a numeric date variable, supplies values to the macro variables **DD1-DD100**. Behind the scenes **DISCRETEATTRMAP** converts numeric variables to character strings by applying associated formats from the input data set.

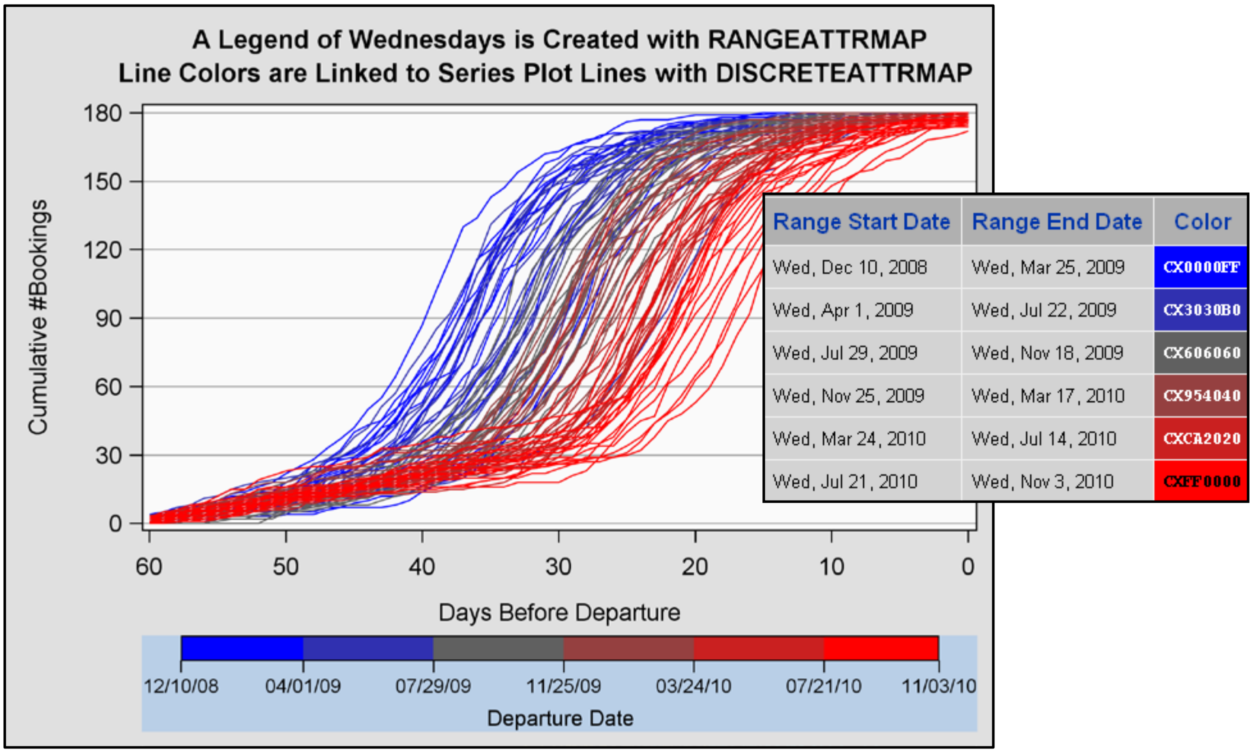
```
DISCRETEATTRMAP NAME="ddateDiscrete";
   VALUE "12/10/08" / LINEATTRS=(color=CX0000FF);
   VALUE "12/17/08" / LINEATTRS=(color=CX0000FF);
   VALUE "12/24/08" / LINEATTRS=(color=CX0000FF);
   VALUE "12/31/08" / LINEATTRS=(color=CX0000FF);
   ...
```

```
        VALUE "03/10/10" / LINEATTRS=(color=CX954040);
        VALUE "03/17/10" / LINEATTRS=(color=CX954040);
        VALUE "03/24/10" / LINEATTRS=(color=CXCA2020);
        VALUE "03/31/10" / LINEATTRS=(color=CXCA2020);
        VALUE "10/13/10" / LINEATTRS=(color=CXFF0000);
        VALUE "10/20/10" / LINEATTRS=(color=CXFF0000);
        value "10/27/10" / LINEATTRS=(color=CXFF0000);
        VALUE "11/03/10" / LINEATTRS=(color=CXFF0000);
    ENDDISCRETEATTRMAP;
```

The **DISCRETEATTRVAR** statement below is the equivalent of **RANGEATTRVAR** for **RANGEATTRMAP**. Furthermore, the newly created associative variable, **DISCRETEASSOC**, is used as an argument to the **GROUP** option in the **SERIESPLOT** statement in exactly the same way that **RANGEASSOC** is assigned to **MARKERCOLORGRADIENT** in the **SCATTERPLOT** statement.

```
    DiscreteAttrVar attrvar=DiscreteASSOC var=DDATE attrmap="ddateDiscrete";
    ...
    seriesplot y=bookings x=DaysLeft / group=DISCRETEASSOC lineattrs=(pattern=1);
```

By using **RANGEATTRMAP** and **DISCRETEATTRMAP** in the code for Figure 10 below, both the continuous and discrete natures of the departure date variable, **DDATE**, can be reflected in the graph. Ironically, **RANGEATTRMAP** highlights the discrete nature of the data in the legend whereas **DISCRETEATTRMAP** processes 100 dates that under ordinary circumstances would be regarded as continuous. Also, it should be mentioned that equal intervals in a continuous legend cannot be viewed as being “arbitrary”. Instead the legend in Figure 10 really serves as a third axis where color becomes a measure of magnitude. For additional information about discrete and continuous data, see **(Watts, 2012).**



Range Start Date	Range End Date	Color
Wed, Dec 10, 2008	Wed, Mar 25, 2009	CX0000FF
Wed, Apr 1, 2009	Wed, Jul 22, 2009	CX3030B0
Wed, Jul 29, 2009	Wed, Nov 18, 2009	CX606060
Wed, Nov 25, 2009	Wed, Mar 17, 2010	CX954040
Wed, Mar 24, 2010	Wed, Jul 14, 2010	CXCA2020
Wed, Jul 21, 2010	Wed, Nov 3, 2010	CXFF0000

Figure 10. The hybrid data type for the airlines data is better represented in a graph where a customized legend can be created with RANGEATTRMAP. Data processing is simplified with DISCRETEATTRMAP, because only one SERIESPLOT statement has to be executed. The X-axis is also reversed so that the cumulative distributions now go from left to right. This is easily done with the REVERSE option in GTL.

GRAPHING THE AIRLINES DATA WITH A MULTI-CELL LAYOUT

Although the graph in Figures 9(b) and 10 confirm the existence of a relationship between booking lead times and flight departure dates, line overlay can be significantly reduced if departure dates are grouped by calendar quarter and displayed separately in a multiple-panel display created by an application of **LAYOUT DATAPANEL**. Group comparisons can also be facilitated if the panels are numbered so that the viewer knows how to read them. The paneled graph is displayed in Figure 11.

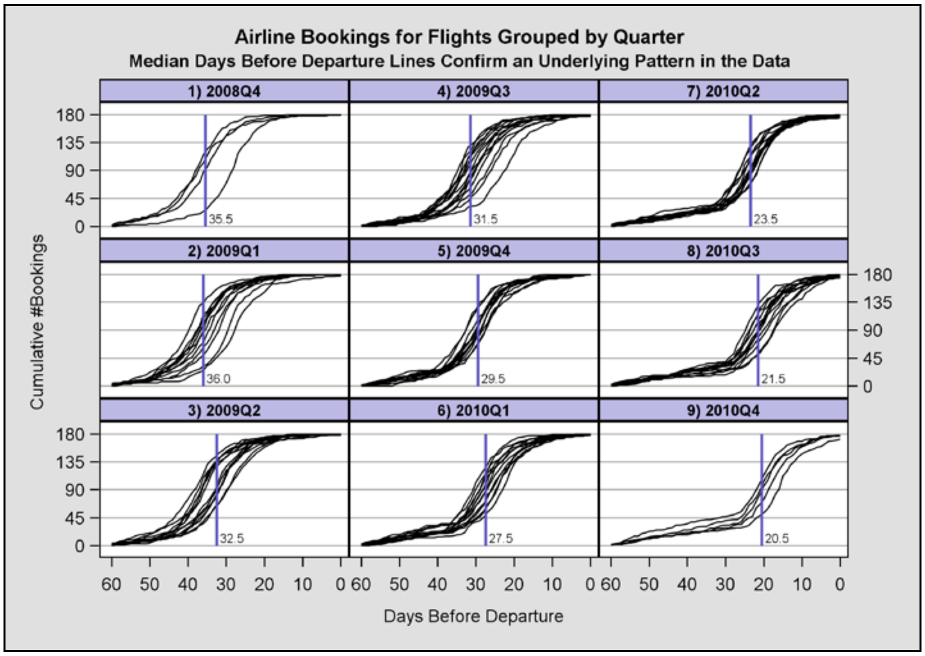


Figure 11. The series plots are more visible with a **DATAPANEL** overlay. Color is no longer necessary, since each group is displayed separately. When the graph is read in column order booking patterns are seen to shift gradually to the right demonstrating that passengers in 2010 don't book their flights as early as they once did. Median Days before departure are plotted with a DROPLINE statement in each plot.

Relevant Code Fragments for Figure 11 Introduce the DATAPANEL Layout

The **LAYOUT** block in the **SERIESPLT** template below supports a **DATAPANEL** statement that:

> creates a grid of graphs based on one or more classification variables and a graphical prototype. By default, a separate instance of the prototype (a data cell) is created for each actual combination of classification variables **(SAS Institute, 2009b, p.59).**

DATAPANEL is easier to work with than **LATTICE.** With the **LATTICE** layout illustrated in Figure 1, each cell in a grid of cells has to be defined separately whereas cell definitions are determined automatically by classification variable affiliation in **DATAPANEL**. In a **DATAPANEL** layout a single **ROWAXISOPTS** option replaces multiple **YAXISOPTS** and one **COLUMNAXISOPTS** replaces multiple **XAXISOPTS**. Likewise a single **LAYOUT PROTOTYPE** replaces multiple **LAYOUT OVERLAY** statements. With a simplified structure, all panels in a graph have an identical appearance. Below is an abbreviated code listing that shows how **DATAPANEL** produces the graph in Figure 11. Code in blue is discussed afterwards.

```
PROC TEMPLATE;
  DEFINE STATGRAPH SERIESPLT;
  BEGINGRAPH / ...
    LAYOUT DATAPANEL CLASSVARS = (QtrWprefix) /
      COLUMNS=3 ROWS=3 ROWDATARANGE=union COLUMNDATARANGE=union ORDER=columnmajor
      HEADERLABELATTRS=(WEIGHT=bold SIZE=8px) HEADERBACKGROUNDCOLOR = CXBCB9E5
      ROWAXISOPTS = (GRIDDISPLAY=on DISPLAY=all ALTDISPLAY=none
                     DISPLAYSECONDARY=none ALTDISPLAYSECONDARY = (ticks tickvalues))
      COLUMNAXISOPTS = (LABEL="Days Before Departure" REVERSE=true...);
        LAYOUT PROTOTYPE;
          SERIESPLOT Y=bookings X=DaysLeft / GROUP=DDate
                                             LINEATTRS = (PATTERN=1 COLOR=black);
          DROPLINE X=mdnDaysLeft Y=180 / DROPTO=X
                                         LINEATTRS=(PATTERN=1 THICKNESS=2px);
        ENDLAYOUT; /*prototype*/
    ENDLAYOUT; /*dataPanel */
  ENDGRAPH;
  end;
run;
```

The variable, **QtrWprefix**, contains nine unique values that correspond to the nine calendar quarters; one for each paneled display. If you look at the graph in Figure 11, you will see that the vertical axis is labeled "Cumulative #Bookings". This text string represents the label assigned to the **bookings** variable in the input data set. **DISPLAY=all** automatically picks up the variable label if there is one. **DISPLAY=all** also controls which axis features are displayed in ***odd*** row occurrences along the primary (left-side) axis **(SAS Institute, 2009b, p.584)**. To determine a row's status as odd or even, count down from the top of the graph. Correspondingly, **ALTDISPLAY= none** controls what is displayed in the ***even*** rows of the graph. The same dichotomy is copied in the secondary Y axis by the **`DISPLAYSECONDARY=`** and the **`ALTDISPLAYSECONARY=`** options. Manipulating the four display options is how to get tick marks to alternate along the primary and secondary axes in **`LAYOUT DATAPANEL`**.

STOCK DATA: WORKING WITH INTERLEAVING TIME SERIES PLOTS

Warren Kuhfeld uses the sashelp.stocks data set to illustrate how time series plots are constructed in GTL and **`PROC SGPLOT`** Kuhfeld **(2010, p.49)**. His GTL graph is reproduced in Figure 12(a) and updated in Figure 12(b). Unfortunately, the graph in Figure 12(b) doesn't represent a significant improvement over the graph in Figure 12(a). Unlike the 100 plot lines in the Airlines Data, the three lines in the Stock data are interleaved, and interleaving creates its own unique set of challenges that are addressed in this section.

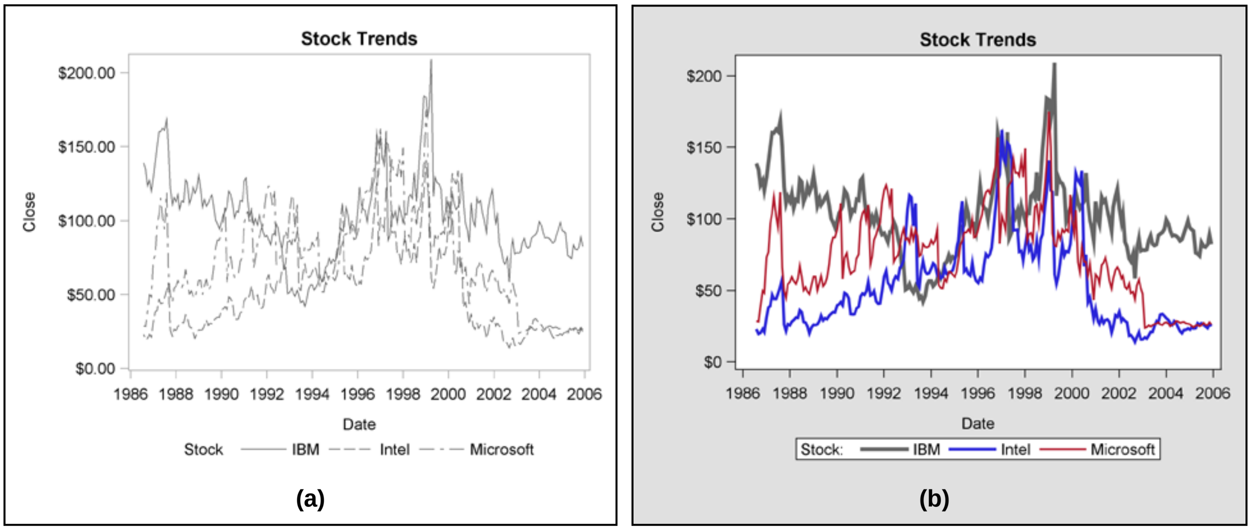


Figure 12. With interleaving lines, it is still difficult to see what happens between 1996 and 2000 in graph **(b).**

A number of changes are made to the graph in Figure 12(b). To start, the **`STATISTICAL`** style is replaced by the **`DEFAULT`** style so that the plotting region is set off from the rest of the graph. Next, the plotting region is extended slightly by reformatting the Y-axis values. Solid lines then replace dashed ones, because dashes interfere with line tracking. Finally, anti-aliasing is applied to improve resolution, and different line widths and colors make it easier but not completely possible to identify each series plot in the display.

It should be mentioned at this point that generating a single graph with multiple line widths and symbol sizes is not easy to do in GTL. These attributes are fixed in the **`GRAPHDATADEFAULT`** style element, so changes cannot be made to **`GRAPHDATA1-GRAPHDATA12`** where customization occurs. A macro that generates multiple line widths is available upon request.

Robbins **(2005, pp.172-173)** recommends placing each plot line into a separate panel of a multi-paneled display. Her trellis plot is similar in structure to the graph generated with a **`DATAPANEL`** layout in Figure 13(a). Now the time series plots are completely distinguishable, but their comparability diminishes with the increased distance between them. Robbins, in fact, hints that *visibility* and *comparability* have a conflicting relationship when she discusses the role between *distance* and *detection* in decoding a graph **(**Robbins **2005, p. 63).**

Results from an initial effort to increase comparability while maintaining visibility are displayed in Figure 13(b). Overall and individual average stock closings are added to the three plots with **`LINEPARM`** statements. Since each panel requires customization in the graph, **`LAYOUT DATAPANEL`** is replaced by **`LAYOUT LATTICE`**. Unfortunately, however, labels for the lines of averages shorten the widths of the plotting regions in each panel of Figure 13(b).

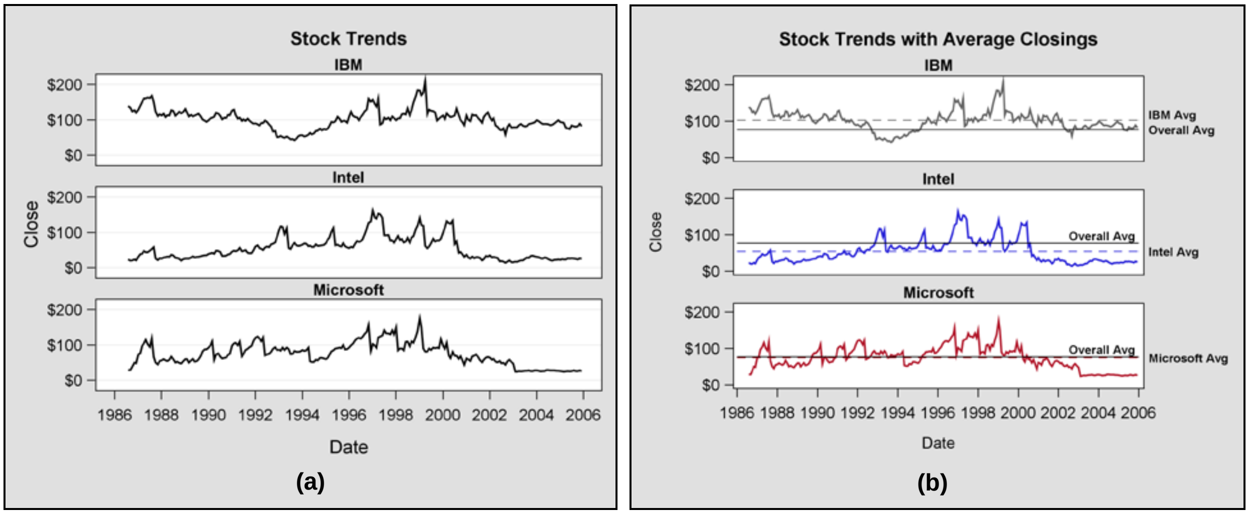


Figure 13. Comparability is compromised in **(a)**, but partially restored in **(b)** with the addition of individual and overall average plot lines. Now individual variations around the plot lines of averages can be compared.

Another way to increase comparability while maintaining visibility is to graph pair-wise comparisons. Initial versions of the comparisons are displayed in Figure 14. In graph (a) two **SERIESPLOT** statements are issued for each panel, and in graph (b), transparent bands are added to emphasize differences in stock closings between two corporations that are being compared.

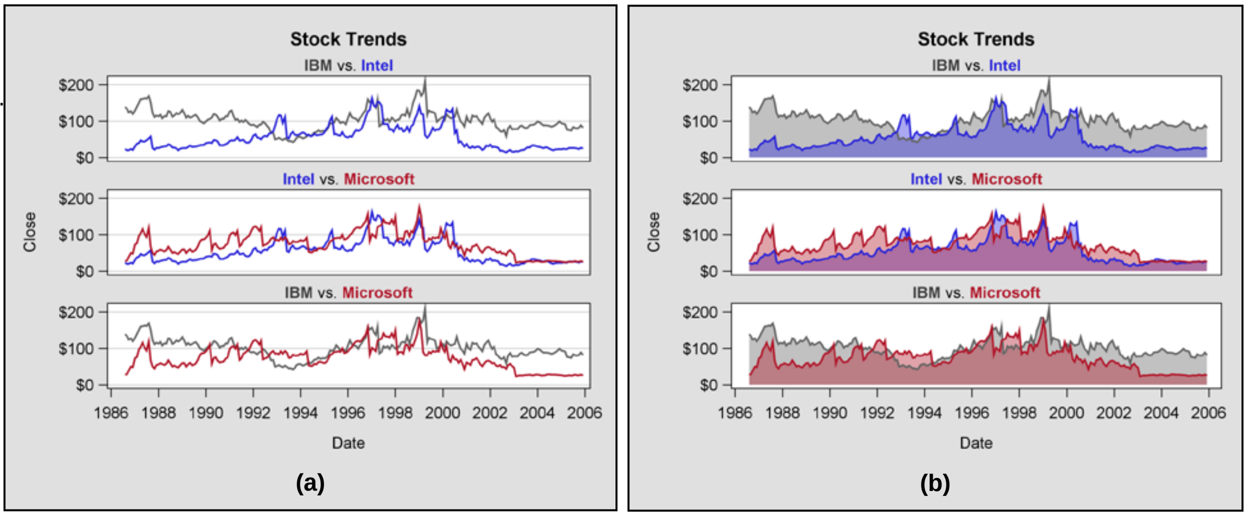


Figure 14. The area under the curve (AUC) becomes the focus of attention when transparent bands are added to the graph in **(b)**. Observe that the larger area between the minimum of the two series plots and zero dollars is also shaded. This area is of no interest. However, with the double overlay it becomes the most emphasized region in the graph.

WORKING WITH BAND PLOTS IN A CELL BLOCK OF A LATTICE LAYOUT

Before getting too involved with band plots, let's review the relatively simple code for the **latticeSeriesPlt** template that is used to generate the graph in Figure 14(b). While **LAYOUT LATTICE** has been discussed before, the subordinate **CELL** block is new. When cell contents are specified in a **CELL** block, customized headers can then become an integral part of the specification (**SAS Institute, 2009b, p.92).** Since the headers of both graphs in Figure 14 serve as "quasi-legends" with identifying colors, their customization requires a **CELL** block within **LAYOUT LATTICE**.

```
PROC TEMPLATE;
  DEFINE STATGRAPH latticeSeriesPlt;
  BEGINGRAPH;
    LAYOUT LATTICE / ...;
 /* Show code for first plot only */
     CELL;
      CELLHEADER;
        ENTRY TEXTATTRS = (COLOR=CX404040 WEIGHT=bold) "IBM"
              TEXTATTRS = (COLOR=black) " vs. "
              TEXTATTRS = (COLOR=graphdata2:CONTRASTCOLOR WEIGHT=bold) "Intel";
      ENDCELLHEADER;
      LAYOUT OVERLAY / ...;
```

```
        BANDPLOT X=x1 LIMITUPPER=y1 LIMITLOWER=0 /
          FILLATTRS = (COLOR=graphdata&i:CONTRASTCOLOR) DATATRANSPARENCY=0.8;
        BANDPLOT X=x2 LIMITUPPER=y2 lIMITLOWER=0 /
          FILLATTRS = (COLOR=graphdata&j:CONTRASTCOLOR) DATATRANSPARENCY=0.8;
        SERIESPLOT Y=y1 X=x1 / LINEATTRS=graphData1;
        SERIESPLOT Y=y2 X=x2 / LINEATTRS=graphData2;
      ENDLAYOUT; /*overlay*/
     ENDCELL;
    ENDLAYOUT; /*lattice */
  ENDGRAPH;
  END;
RUN;
```

Band plots are not explicitly defined anywhere in the SAS literature. All that is said in the Language Reference manual is that the **BANDPLOT** statement is typically used to show “confidence or prediction limits” **(SAS Institute, 2009b, p.157)**. In addition, when **X** is specified, as it is in the code fragment above, **LIMITUPPER** and **LIMITLOWER** expect references by variable or constant to a **Y** value. What is left unsaid is that **LIMITUPPER** can never go below **LIMITLOWER**. When that occurs, the “band” is completely erased from the output. What is needed in this situation is *fake interleaving* where **LIMITUPPER** looks like it can go below **LIMITLOWER** and conversely **LIMITLOWER** looks like it can go above **LIMITUPPER**. Such flexibility is reflected in the final version of the time series graph for the sashelp.stocks data set in Figure 15.

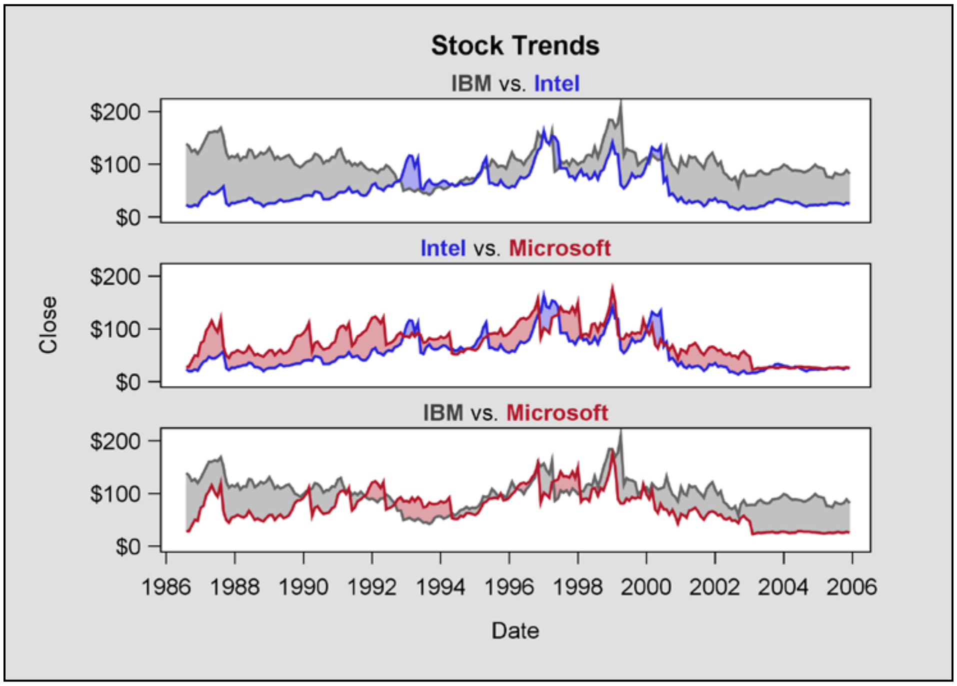


Figure 15. This time the area *between* the curves becomes the focus of attention. Now the two series plots in each panel are fully visible *and* comparable.

To see how it is possible to construct an interleaving band plot, compare Figures 15 and 16 for **Intel vs. Microsoft**. On January 4, 1988, the value for **close** was $55.75 for Microsoft and $25.50 for Intel. With these numbers, the first band plot in light red extends from $55.75 down to $25.50. A *second* virtually invisible band in Figure 15 extends from $25.50 to $25.49; a difference of one cent! One cent is still greater than zero, so **LIMITUPPER** continues to be greater than **LIMITLOWER**. How interleaving works is expressed visually in Figure 16 by increasing the lower boundary drop from one cent to $10.00.

The **REFERENCELINE** statement is used for the first time in Figure 16 to mark January 4, 1988 on the graph. The code for the statement is as follows:

```
REFERENCELINE X='04Jan1988'D / LINEATTRS = (THICKNESS=4 COLOR=CX505050)
                               CURVELABEL="January 4, 1988"
                               DATATRANSPARENCY=0.5;
```

REFERENCELINE differs from **LINEPARM** in that only an **X** *or* **Y** coordinate has to be specified. Both coordinates are required for **LINEPARM** along with a value for **SLOPE**. Here is the corresponding code fragment for **LINEPARM** used in Figure 13(b) for the average lines:

```
LINEPARM X=EVAL(MIN(date)) Y=EVAL(MEAN(close)) SLOPE=0 / CURVELABEL="Overall Avg";
```

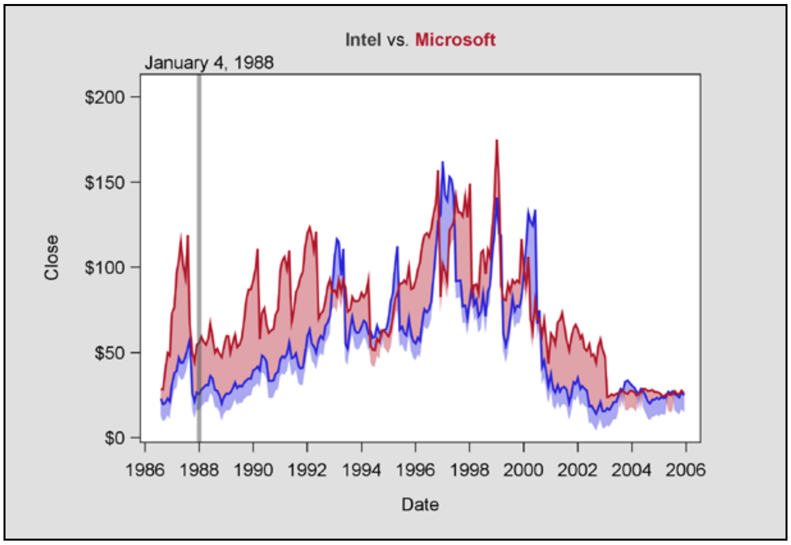


Figure 16. Two distinct continuous bands each with **LIMITUPPER** and **LIMITLOWER** values become visible when the lower boundary drop is increased from 1 cent to $10.00 in the stock data.

SUMMARY AND CONCLUSIONS

An incremental approach has been taken in this paper to address issues associated with input data that translate into graphs with overlapping points or lines. With the Framingham Heart Study we started with a scatter plot with marginal histograms that has a large undifferentiated “blob” of overlapping data points in the center. The graph was squared off so that the two histograms with uniform response axes specifications could be compared. Next *rounding*, the inverse of Cleveland’s *jittering*, was applied so that the “blob” could be color-coded by frequency. Then subjects were categorized by their BMI scores and the two axes were rotated to emphasize that the Body Mass Index is primarily a measure of weight as opposed to height. The marginal histograms were also removed from this graph and the **HEATMAPPARM** statement replaced the **SCATTERPLOT** statement. The final graph in this section mixed raw and summary data with the new **DISCRETEOFFSET** option that makes it possible to use the entire data display region to accommodate discrete data.

Color was also systematically applied to ranges of departure dates in the airlines data so that the connection between departure date and booking lead times could be made. Even though a connection was made, problems remained with the preliminary graph, because it that did not represent the departure date variable as having a hybrid data type. By using the new **RANGEATTRMAP** and **DISCRETEATTRMAP** statements in an updated graph, the problem was solved. Arbitrary ranges were also converted to more meaningful calendar quarters and a second nine-panel graph was created with the **DATAPANEL** layout. Not only did grouping by quarter fit the business paradigm, but splitting 100 series plots into nine groups increases visibility.

Lattice plots were used to promote readability with the stock data. Unlike the airlines data where the progression of time series plot lines shared the same S-curve, the stock plot lines were uncoordinated, jagged and interleaved. Several display structures were presented as improvements over the original display. We favor the new *interleaving* band plot where the area between two curves becomes the focus of attention.

A lot of ground has been covered in this paper, and if you are not familiar with GTL you may want to focus on the graphics output now and then return to the paper at a later date if you need to generate something similar in your own work. In the meantime, read introductory SAS papers and Kuhfeld’s book. The GTL manuals are also helpful, since they are motivated by example and have complete listings of available options.

REFERENCES

Albers, J. (1975), *Interaction of Color*, revised edn, Yale University Press, Westford, MA.

Brewer, C. A. (1994), Color use guidelines for mapping and visualization, in A. M. MacEachren and D. E. F. Taylor (eds),*Visualization in Modern Cartography*, Vol. 2, Elsevier Science, Inc., New York, pp. 123–147.

Brockwell, P. J. and Davis, R. A. (1996), *Introduction to Time Series and Forecasting*, Springer Verlag, New York.

Cleveland, W. S. (1994), *The Elements of Graphing Data, revised edn*, Hobart Press, Summit, NJ.

Derby, N. and Vo, L. (2010), Graphing a progression of time series plots*, Proceedings of the 2010 Western Users of SAS Software Conference.* http://www.wuss.org/proceedings10/analy/2954_6_ANL-Derby.pdf

Flom, P. (2011), Using the SG Procedures to create and enhance scatter plots*, Proceedings of the 24th* Annual Northeast *SAS Users Group Conference*, paper #PO06*.* http://www.nesug.org/proceedings/nesug11/po/po06.pdf

Hebbar, P. (2012), Off the Beaten Path: Create Unusual Graphs with GTL, *Proceedings of the SAS® Global Forum 2012 Conference*, paper 267-2012. http://support.sas.com/resources/papers/proceedings12/267-2012.pdf

Hubert, H. B., Feinleib, M., McNamara, P. M. and Castelli, W. P. (1983), Obesity as an independent risk factor for cardiovascular disease: a 26-year follow-up of participants in the Framingham Heart Study, **Circulation**, **67**, 968–977.

Kuhfeld, W. F. (2010), *Statistical Graphics in SAS®: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*, SAS Institute, Inc., Cary, NC.

Matange, S., Cartier, J. and Heath. D. (2009), *SAS/GRAPH®: Graph Template Language* distributed as a SAS Institute handout at SAS Global Forum, 2009. Code for SCATTERHIST.SAS in the handout is at **SAS Institute (2009a)**.

McClave, J. T. and Sincich, T. (2003), *Statistics, ninth edn*, Prentice-Hall, Inc., Upper Saddle River, NJ.

Robbins, N. B. (2005), *Creating More Effective Graphs*, John Wiley and Sons, Inc., Hoboken, NJ.

SAS Institute (2004), *SAS/GRAPH® 9.1 Reference, Volumes 1, 2 and 3,* SAS Institute, Inc., Cary, NC.

SAS Institute (2009a), *Sample 35172: Distribution plot*.
http://support.sas.com/kb/35/172.html

SAS Institute (2009b), *SAS/GRAPH® 9.2 Graph Template Language Reference*, SAS Institute, Inc., Cary, NC.

SAS Institute (2009c), *SAS/GRAPH® 9.2 Graph Template Language User's Guide*, SAS Institute, Inc., Cary, NC.

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

SAS Institute (2011), *SAS/GRAPH® 9.3 Graph Template Language User's Guide*, SAS Institute, Inc., Cary, NC.

Siegel, A. F. and Morgan, C. J. (1996), *Statistics and Data Analysis: An Introduction, second edition*, John Wiley and Sons, Inc. New York.

Tufte, E. R. (1990), *Envisioning Information*, Graphics Press. Cheshire, CT.

Watts, P. (2002), Using ODS and the Macro Facility to Construct Color Charts and Scales for SAS® Software Applications, *Proceedings of the Twenty-Seventh SAS Users Group International Conference*, paper 125-27.
http://www2.sas.com/proceedings/sugi27/p125-27.pdf

Watts, P. (2003), Working with RGB and HLS Color Coding Systems in SAS® Software, *Proceedings of the Twenty-Eighth SAS® Users Group International Conference*, paper 136-28.
http://www2.sas.com/proceedings/sugi28/136-28.pdf

Watts, P. (2004), Advanced Programming Techniques for Working with Color in SAS® Software, *Proceedings of the Twenty-Ninth SAS Users Group International Conference,* paper 091-29.
http://www2.sas.com/proceedings/sugi29/091-29.pdf

Watts, P. (2012), *Using Axes Options to Stretch the Limits of SAS® Graph Template Language*, Proceedings of the 25th Annual Northeast SAS Users Group Conference, paper #BB4.

Yang, D. (2011), Creating a half error bar using Graph Template Language, *Proceedings of the 2011 SAS® Global Forum*, paper 284-2011.
http://support.sas.com/resources/papers/proceedings11/284-2011.pdf

ACKNOWLEDGEMENTS

We thank Lisa Farino, technical writer at Stakana Analytics, for stressing the importance of using more meaningful units such as calendar quarters to group the airlines data. We also wish to express our gratitude to William S. Cleveland for his seminal contributions to the field of statistical graphics. His book, *The Elements of Graphing Data,* is a gem of clarity, and it inspired a lot of what appears in this paper.

TRADEMARK CITATION

CONTACT INFORMATION

Comments and questions are valued and encouraged. Contact one of the authors. Code requests should be directed to Perry Watts.

Perry Watts
Stakana Analytics
pwatts@stakana.com

Nate Derby
Stakana Analytics
nderby@stakana.com

sas Alliance Loyalty Partner

APPENDIX: USER-DEFINED MACRO *RGBCODES4COLORSCALE*

```
/*  ---------------------------------------------------------------------------
    Program  :  RGBcodes4ColorScale.sas
    Author   :  Perry Watts
    Purpose  :  Obtain the RGB codes that comprise the linear path of an
                 arbitrary color scale. Intermediate points along the scale
                 can be calculated from the single SOURCE and DESTINATION
                 codes that are input to the macro.
    ------------------------------------------------------------------------  */
%macro RGBcodes4ColorScale
  (SourceRGB=,   /* Source (begin) RGB code */
   DestRGB=,     /* Destination (end) RGB code */
   dsID=,        /* Output data set name */
   startObs=,    /* First OBSNUM to be inserted into the output data set. */
   n=,           /* Number of observations after the first one */
   by=1);        /* For evenly spacing the codes */

  %Local rr gg bb SR SG SB DR DG DB i prefix target r g b;

  %do i = 1 %to 2;
    %if &i eq 1 %then %do;
      %let prefix=S;
      %let target=SourceRGB;
     %end;
     %else %do;
       %let prefix=D;
       %let target=DestRGB;
     %end;
     %let rr=%substr(&&&target,3,2);
     %let gg=%substr(&&&target,5,2);
     %let bb=%substr(&&&target,7);
      /* SR, SG, SB, DR, DG, DB created below are decimals */
     %let &prefix.R=%sysfunc(putn(%sysfunc(inputn(&rr,hex2.)),z3.));
     %let &prefix.G=%sysfunc(putn(%sysfunc(inputn(&gg,hex2.)),z3.));
     %let &prefix.B=%sysfunc(putn(%sysfunc(inputn(&bb,hex2.)),z3.));
   %end;

 data &DSID (keep=obsNum color);
     %do i = 0 %to &n %by &by;
       %let r = %sysfunc(round(%sysevalf(&sr + (&dr - &sr.)*&i/&n)));
       %let g = %sysfunc(round(%sysevalf(&sg + (&dg - &sg.)*&i/&n)));
       %let b = %sysfunc(round(%sysevalf(&sb + (&db - &sb.)*&i/&n)));
       obsNum=%eval(&startOBS + &i);
       color="%rgbhex(&r,&g,&b)";
       output;
     %end;
 run;
%mend RGBcodes4ColorScale;

/* *****  EXAMPLE #1
   In this example from the paper, ramp colors that work with individual observations are
   discarded in favor of six ranges of departure dates that are processed by RANGEATTRMAP.
   (CX0000FF = BLUE, CX606060 = DARK GRAY, CXFF0000 = RED). The two data sets start2Neutral
   and Neutral2End are concatenated after the two macro calls are issued. The duplicate neutral
   color in the concatenated data set is removed with a NODUPS option attached to PROC SORT.
***** */
%RGBcodes4ColorScale(sourceRGB=CX0000FF, destRGB=CX606060, dsID=Start2Neutral, startobs=1, n=2)
%RGBcodes4ColorScale(sourceRGB=CX606060, destRGB=CXFF0000, dsID=Neutral2End, startobs=3, n=3)

/* *****  EXAMPLE #2
   Is almost the same as Example #1 except that each departure date is assigned a unique color
   code. This is done by increasing the values for the two ‘n’ parameters.
***** */
%RGBcodes4ColorScale(sourceRGB=CX0000FF, destRGB=CX606060, dsID=Start2Neutral, startobs=1, n=49)
%RGBcodes4ColorScale(sourceRGB=CX606060, destRGB=CXFF0000, dsID=Neutral2End, startobs=50, n=50)

/* *****  EXAMPLE #3
   A single macro call recreates the color scale used in the USA population pyramid from
   “Off the Beaten Path: Create Unusual Graphs with GTL” by Prashant Hebbar.
***** */
%RGBcodes4ColorScale(sourceRGB=CXFF0000, destRGB=CXFFFF00, dsID=Red2Yellow, startobs=1, n=70)
```