Paper SAS1800-2017

Heat Maps: Graphically Displaying Big Data and Small Tables

Warren F. Kuhfeld, SAS Institute Inc.

ABSTRACT

Heat maps use colors to communicate numeric data by varying the underlying values that represent red, green, and blue (RGB) as a linear function of the data. You can use heat maps to display spatial data, plot big data sets, and enhance tables. You can use colors on the spectrum from blue to red to show population density in a US map. In fields such as epidemiology and sociology, colors and maps are used to show spatial data, such as how rates of disease or crime vary with location. With big data sets, patterns that you would hope to see in scatter plots are hidden in dense clouds of points. In contrast, patterns in heat maps are clear, because colors are used to display the frequency of observations in each cell of the graph. Heat maps also make tables easier to interpret. For example, when displaying a correlation matrix, you can vary the background color from white to red to correspond to the absolute correlation range from 0 to 1. You can shade the cell behind a value, or you can replace the table with a shaded grid. This paper shows you how to make a variety of heat maps by using PROC SGPLOT, the Graph Template Language, and SG annotation.

MAPS

You can use PROC SGPLOT and the POLYGON statement to create maps. For a map of the United States, each state is a polygon, and each state boundary is constructed from a data set of X and Y coordinates of points along the borders. The borders of relatively square states, such as Colorado and Wyoming, are defined by a small number of points. Borders of states that have more complex shapes, such as Alaska and Maryland, are defined by many more points. The coordinates for each state's borders are in the **MapsGfk.US** data set, which is available in SAS/GRAPH® software. The population density of each state in the year 2010 is in the **Sashelp.US_data** data set. The next steps sort both data sets and then merge them:

```
proc sort data=mapsgfk.us out=USBorders equals;
   by statecode;
run;

proc sort data=sashelp.us_data(keep=density_2010 statecode) out=USDensity;
   by statecode;
run;

data USDensity;
   merge USBorders USDensity;
   by statecode;
   if statecode ne 'DC';
run;
```

The District of Columbia is not displayed along with the states, because its population density is too great. If the District of Columbia were included, the population densities of the states would not be apparent in the map.

The following steps create the final data set for plotting:

```
%annomac /* Make the SAS/GRAPH annotate macros available */

%centroid(mapsgfk.us,centers,id) /* Polygon center is used to place each label */

data map(drop=statecode segment);
   set USDensity centers(rename=(x=xCen y=yCen) in=a);
   if a then Label=fipstate(input(substr(id,4,2), 2.)); /* State postal code */
   id = catx('-', id, segment);  /* Combine ID and Segment to make unique ID */
   if label = 'ID' then ycen + -.025;      /* Adjust a few label coordinates */
   if label = 'MI' then ycen + -.025;
```

```
   if label = 'HI' then ycen + -.01;
   if label = 'NH' then ycen + -.012;
   if label = 'VT' then ycen +  .01;
   if label = 'MD' then ycen +  .007;
   if label = 'AK' then ycen +  .01;
   if label = 'DE' then do; ycen + -.005; xcen + .005; end;
   if label = 'DC' then delete;
run;
```

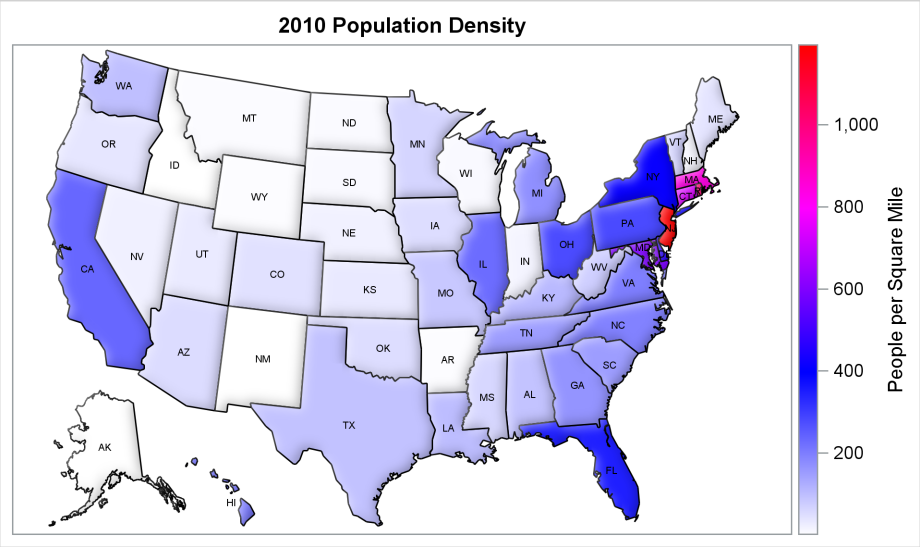
The %AnnoMac autocall macro provides handy macros for graphics and mapping, including the %Centroid macro, which finds the centroid (geographic center) of each state. The centroids provide the coordinates for labeling each state. For some states, the centroid is a poor choice, so the DATA step performs ad hoc adjustments of those coordinates. For example, Michigan has three distinct sections (two peninsulas and an island), so its label is moved down into the body of the lower peninsula. The label for Vermont, which is wider in the north, is moved up. The label for New Hampshire, which is wider in the south, is moved down.

The data set **Map** contains 3,858 observations that contain the coordinates for the map. These observations contain nonmissing values for the variables named in the POLYGON statement (**Y**, **X**, **ID**, and **Density_2010**) and missing values for the variables named in the SCATTER statement (**xCen**, **yCen**, and **Label**). The remaining 50 observations contain nonmissing values for the variables named in the SCATTER statement and missing values for the variables (except **ID**) named in the POLYGON statement. Hence, the POLYGON statement uses only the observations that have borders, and the SCATTER statement uses only the observations that have labels. The following step uses a POLYGON statement to create the map and a SCATTER statement to label each state:

```
ods graphics on / height=3.8in width=6.4in;
proc sgplot data=map noautolegend;
   title '2010 Population Density';
   polygon x=x y=y id=id / outline lineattrs=(color=black)
           colorresponse=density_2010 dataSkin=matte
           fill colormodel=(white blue magenta red) name='map';
   scatter x=xcen y=ycen / markerchar=label markercharattrs=(size=5);
   gradlegend 'map';
   label density_2010 = 'People per Square Mile';
   xaxis offsetmin=0.01 offsetmax=0.01 display=none;
   yaxis offsetmin=0.01 offsetmax=0.01 display=none;
run;
ods graphics on / reset=all;
```

The graph is displayed in Figure 1.

Figure 1 US Population Density in 2010



The COLORRESPONSE=**Density_2010** and COLORMODEL=(WHITE BLUE MAGENTA RED) options use shades of white to color the states that have the smallest population densities and shades of red to color the states that have the largest population densities. States whose densities are in between are displayed in shades of blue and magenta. Red, green, and blue (RGB) values vary as a linear function of density. The RGB values vary from white (255, 255, 255) to blue (0, 0, 255) to magenta (255, 0, 255) to red (255, 0, 0) as density ranges from its minimum (1.2) to its maximum (1195.5).

HEAT MAPS FOR BIG DATA SETS

Scatter plots display a marker at the intersection of the values of an X variable and a Y variable. In contrast, heat maps divide the graph into rectangular (or hexagonal) bins and use colors to show how many observations fall in each bin. If you have a large number of data points, then ordinary scatter plots, fit plots, residual plots, and so on become hard to interpret. If you have enough data, then points merge into large blobs that do not always reveal the underlying structure of the data. Heat maps differentiate more clearly between the denser and less dense portions of the data. The following steps create a scatter plot of 25,000 artificial data points:

```
title;
data x(drop=i);
   do i = 1 to 25000;
      x = 2 * normal(104);
      y = x + sin(x * 2) + 3 * normal(104);
      output;
   end;
run;

proc sgplot data=x;
   scatter y=y x=x;
run;
```

The results are displayed in Figure 2.

The following step creates a heat map of the same data:

```
proc sgplot data=x;
   heatmap y=y x=x;
run;
```

The results are displayed in Figure 3.

Figure 2 Ordinary Scatter Plot

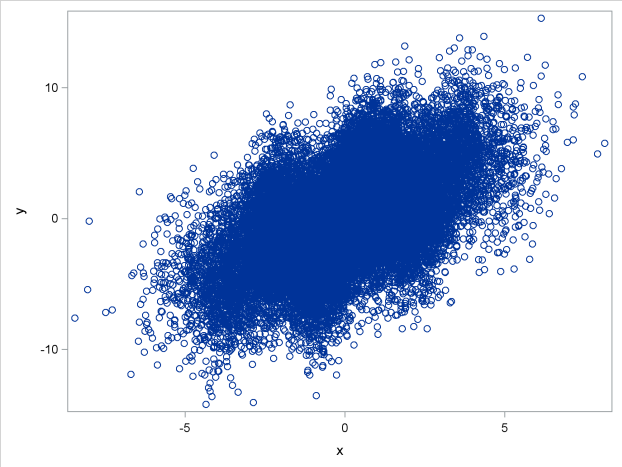
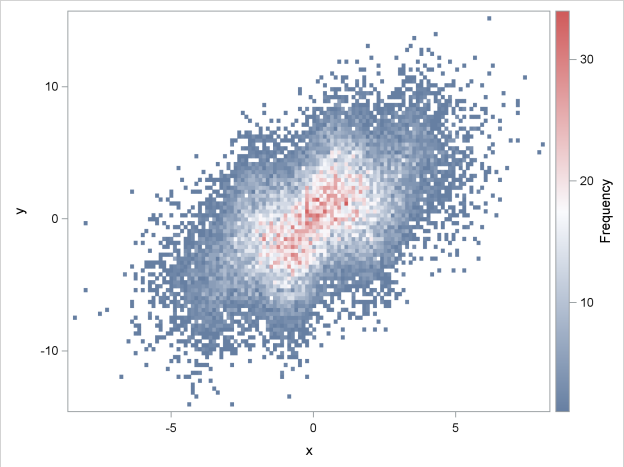


Figure 3 Default Heat Map



The underlying function, which has a linear component and a sine wave component, is apparent in Figure 3 but not in Figure 2. The colors in the heat map progress from blue to light gray to red. These colors are called a *color ramp*. You can reorder the colors to progress from light gray (similar to the white background) to blue to red for an even clearer effect. You are not required to specify the same colors that are used in the HTMLBlue style; however, you can find them by listing the style source code:

```
proc template;
   source styles.htmlblue / expand;
quit;
```

When the EXPAND option is specified, style listings are large, so the results of this step are not displayed. Instead, the following steps write the same information to a file and display just the components of the three-color ramp for the HTMLBlue style and its parents:

```
proc template;
   source styles.htmlblue / file='junk.tpl' expand;
quit;

data junk;
   retain Style '           ';
   infile 'junk.tpl' lrecl=80 pad;
   input Color $ 1-80;
   if index(Color, 'define style') then style = scan(Color, 4, ' .;');
   if index(Color, 'gramp3') and not index(Color, 'GraphColors');
run;

proc print noobs;
run;
```

The results are displayed in Figure 4. The listing displays the style names and the color names for the HTMLBlue style, its parent Statistical, and its parent Default.

Figure 4 Three-Color Ramp Components

Style	Color
Htmlblue	'gramp3cend' = cxD05B5B
Htmlblue	'gramp3cneutral' = cxFAFBFE
Htmlblue	'gramp3cstart' = cx667FA2
statistical	'gramp3cend' = cx667FA2
statistical	'gramp3cneutral' = cxFFFFFF
statistical	'gramp3cstart' = cxAFB5A6
default	'gramp3cend' = cxDD6060
default	'gramp3cneutral' = cxFFFFFF
default	'gramp3cstart' = cx6497EB

You can use the COLORMODEL= option as follows to specify the three colors—the neutral color (gray), the starting color (blue), and the ending color (red):

```
proc sgplot data=x;
   heatmap y=y x=x / colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
run;
```

All colors are specified in values of the form CX*rrggbb*, where the last six characters specify RGB (red, green, blue) values on the hexadecimal scale of 00 to FF (or 0 to 255 base 10). The results are displayed in Figure 5. The data pattern is even clearer when you use this color scheme.

You can create a heat map and control the binning yourself by using the HEATMAPPARM statement. The following steps illustrate:

```
data grid;
   set x;
   x = round(x, 0.1);
   y = round(y, 0.2);
run;

proc freq noprint;
   tables x * y / out=grid2;
run;
```

```
proc sgplot data=grid2;
   heatmapparm y=y x=x colorresponse=count /
                       colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
run;
```

The DATA step rounds similar values to a common value, creating a grid. The PROC FREQ step counts the number of values in each cell of the grid. The resulting variable, **Count**, is specified in the COLORRESPONSE= option in the HEATMAPPARM statement. The results are displayed in Figure 6.

Figure 5 Alternative Ordering of the Color Ramp

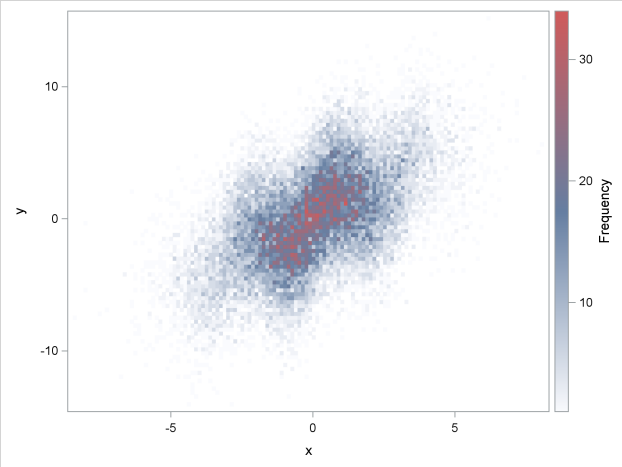
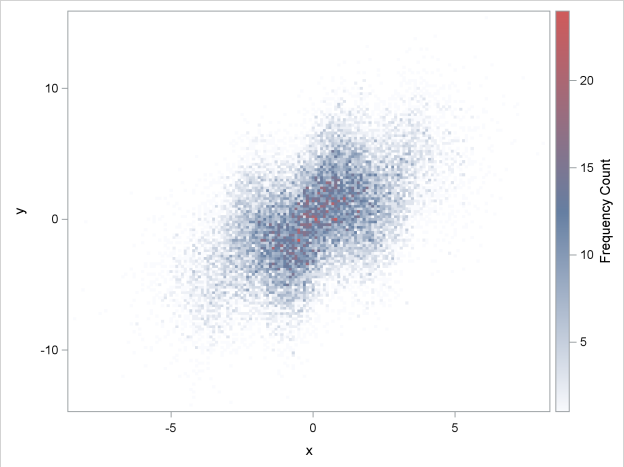


Figure 6 Controlling the Heat Map Grid



You can use the GRADLEGEND statement to control the color gradient legend:

```
data grid3;
   set grid2;
   count = count * constant('pi') * 1e5;
run;

proc sgplot data=grid3;
   heatmapparm y=y x=x colorresponse=count / name='a'
                       colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
   gradlegend 'a' / integer title = 'Frequency' extractscale;
run;
```

The DATA step artificially creates larger frequencies. The GRADLEGEND statement controls the legend for the statement named `'a'`. Values in the legend are required to be integers, the label is set to “Frequency”, and a common scale (millions) is extracted and added to the label. The results are displayed in Figure 7.

The following steps show SAS/STAT® procedures that create heat maps:

```
proc reg data=x;
   model y = x;
quit;

proc surveyreg data=x;
   model y = x;
run;

proc surveyreg data=x plots=fit(shape=hexagonal);
   model y = x;
run;
```

The results are displayed in Figure 8 through Figure 11. PROC SURVEYREG has an option for hexagonal binning. Three bin shapes are possible: triangle, rectangle, and hexagon. Rectangles are commonly used. Hexagons have the advantage of having more sides than squares (hexagons are closer geometrically to circles than squares are), which means that hexagons have shorter distances to the bin centers than squares do. No other procedure has a hexagonal binning option.

Figure 7 Controlling the Heat Map Legend

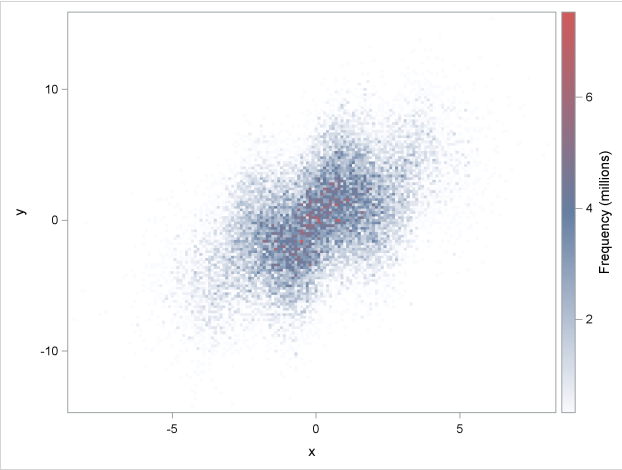


Figure 8 PROC REG Fit Plot

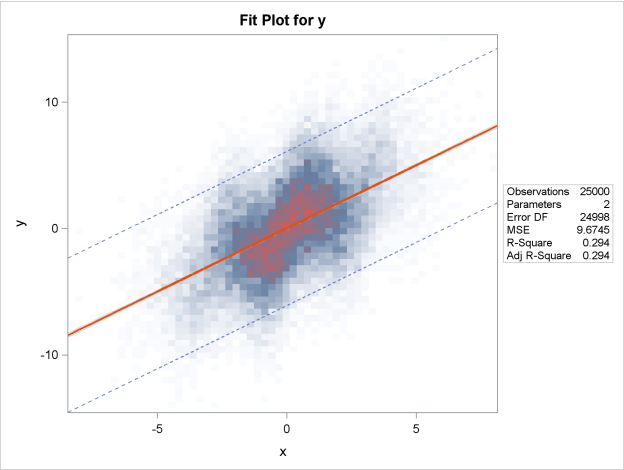


Figure 9 PROC REG Residual Plot

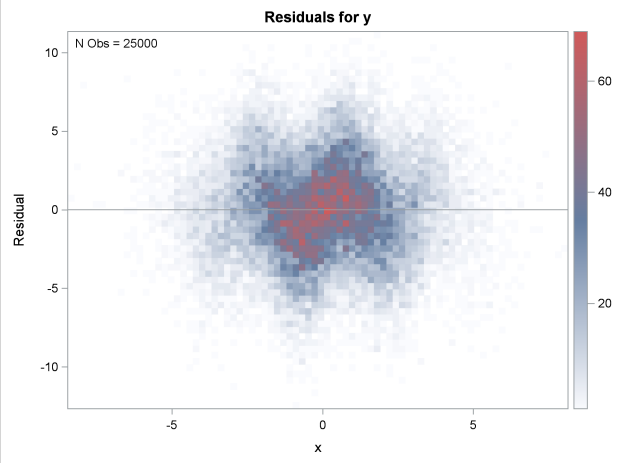


Figure 10 PROC SURVEYREG Fit Plot

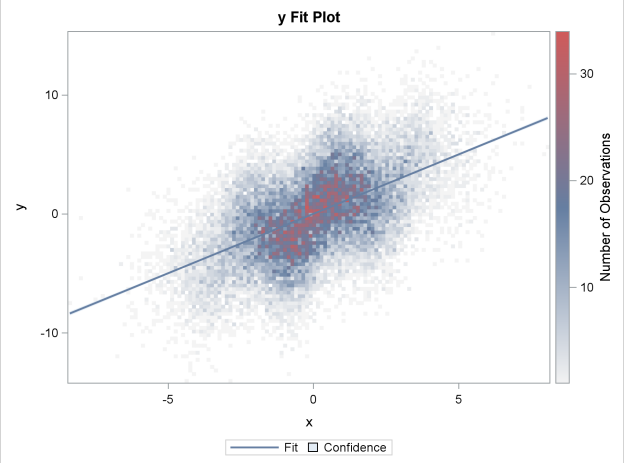
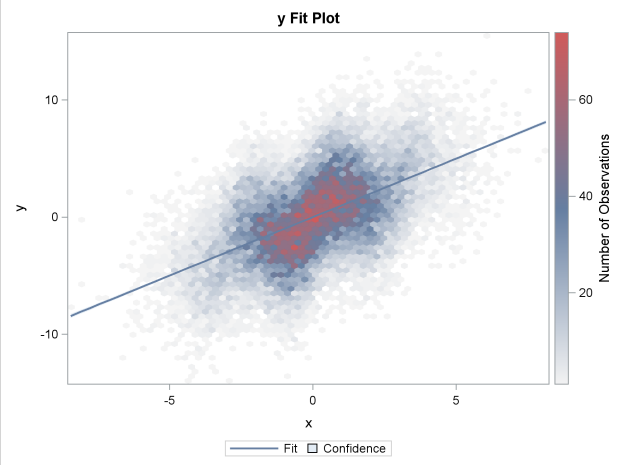


Figure 11 PROC SURVEYREG with Hexagonal Binning



HEAT MAPS FOR ENHANCING TABLES

Heat maps can also be used to enhance small tables. In this example, you learn how to put an outline around some of the cells (those with values greater than 3). A Toeplitz matrix is used to illustrate how to display a heat map together with a table, because the values have a simple pattern. The first step creates an 11 × 11 Toeplitz matrix that contains entries of 0 to 10. PROC IML creates a matrix that is arrayed in a single column for plotting, and it creates a SAS data set of row labels, column labels, and values.

```
proc iml;       /* Create a Toeplitz matrix  */
   t = 0:10;    /* Values range from 0 to 10 */
   x = expandgrid(t, t) || shape(toeplitz(t), 1)`;
   create tmat from x[colname={"RowLab" "ColLab" "Value"}];
   append from x;
quit;
```

The function **toeplitz(t)** creates an 11 × 11 matrix, **shape(toeplitz(t), 1)** converts the matrix to a 1 × 121 row vector, and **shape(toeplitz(t), 1)`** transposes the row vector to make a 121 × 1 column vector to plot.

The next step uses PROC SGPLOT to create a table in which each cell is shaded to show the pattern of values:

```
ods graphics on / height=2.5in width=2.25in;
proc sgplot noautolegend;
   title h=7pt 'Toeplitz Matrix';
   heatmapparm y=rowlab x=collab colorresponse=value /
               colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
   text        y=rowlab x=collab text=value;
   %let opts = values=(0 to 10) display=(nolabel noticks) valueattrs=(size=7)
               offsetmin=0.05 offsetmax=0.05;
   xaxis &opts;
   yaxis &opts reverse;
   format value 2.;
run;
```

The entries 0 to 10 are displayed as a color ramp that ranges from off-white to shades of blue and finally to shades of red. The results are displayed in Figure 12.

The next example creates the results shown in Figure 13. It uses a two-color ramp (white to blue) and outlines values that exceed a certain threshold (in this case, 3).

Figure 12 Heat Map and Table

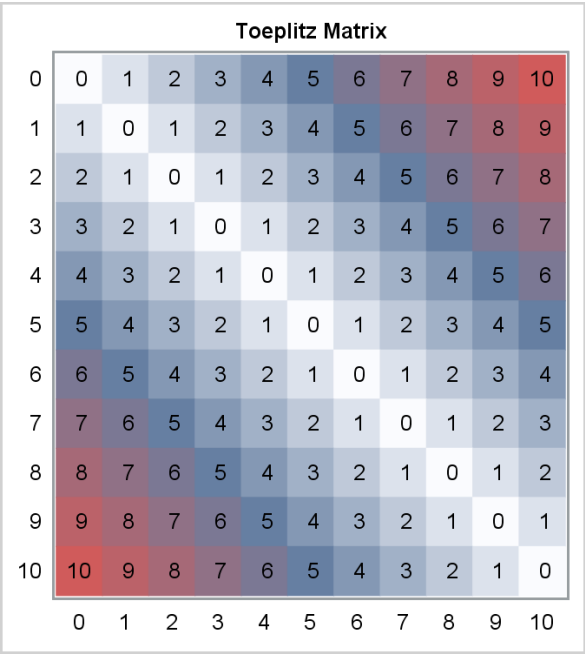
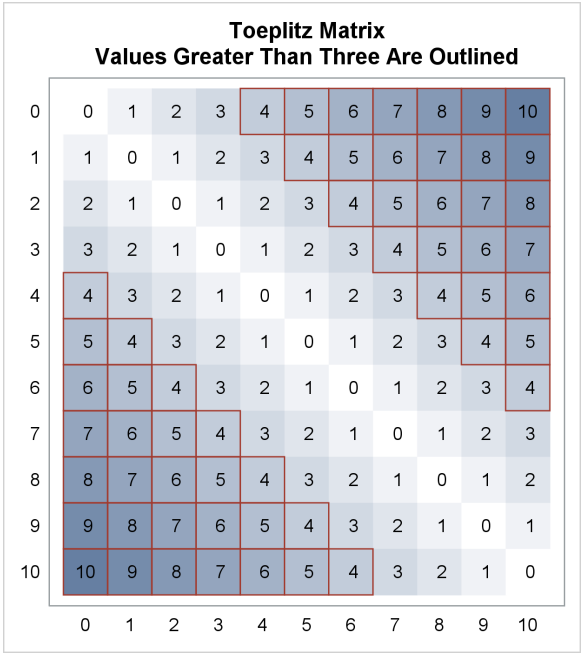


Figure 13 Heat Map, Table, and Outlines



The next step creates a macro variable whose value controls the size of the cells for plotting. An approximately 18 × 18 pixel square is reserved for each cell. The **Outline** variable has nonmissing values when the value of the **Value** variable is greater than 3. These are the cells that are outlined.

```
data mat2;
   if _n_ = 1 then /* Scale matrix size by the number or rows/columns */
      call symputx('size', ceil(18 * sqrt(n)));
   set tmat nobs=n;
   Outline = ifn(Value > 3, Value, .); /* Outline the nonmissing values (Value > 3) */
   output;
run;

proc print data=mat2(obs=11);
   id RowLab ColLab;
run;

%put &size;
```

Figure 14 displays 11 of the 121 observations.

Figure 14 The First 11 Observations

Toeplitz Matrix

RowLab	ColLab	Value	Outline
0	**0**	0	.
0	**1**	1	.
0	**2**	2	.
0	**3**	3	.
0	**4**	4	4
0	**5**	5	5
0	**6**	6	6
0	**7**	7	7
0	**8**	8	8
0	**9**	9	9
0	**10**	10	10

The template creates a graph that is 250+198=448 pixels high and 200+198=398 pixels wide. The height is greater than the width to provide space for titles. Unlike PROC SGPLOT, the GTL enables you to specify **style-element:attribute** rather than looking up colors in the style. The first HEATMAPPARM statement creates a heat map that uses colors from white (**GraphWalls:Color**) to blue (**ThreeColorRamp:StartColor**) to display the values of the **Value** variable from 0 to 10. The second HEATMAPPARM statement outlines the nonmissing values of the **Outline** variable. It uses the option FILLATTRS=(TRANSPARENCY=1) to suppress the heat map and to display only outlines. The TEXTPLOT statement displays the values overlaid on the shaded cells.

The following steps create the matrix shown in Figure 13:

```
proc template;
   define statgraph matrix;
      begingraph / designheight=%eval(250+&size) /* Size: a bit higher than wide */
                   designwidth =%eval(200+&size);
         entrytitle "Toeplitz Matrix";
         entrytitle "Values Greater Than Three Are Outlined";
         layout overlay / yaxisopts=(discreteopts=(tickvaluefitpolicy=none)
                                    display=(tickvalues) reverse=true)
                          xaxisopts=(discreteopts=(tickvaluefitpolicy=rotate)
                                    display=(tickvalues));
            * Heat map provides the background color for each cell;
            heatmapparm y=RowLab x=ColLab colorresponse=Value /
               ColorModel=(GraphWalls:Color ThreeColorRamp:StartColor);
            * Heat map provides the outlines;
```

```
            heatmapparm y=RowLab x=ColLab colorresponse=Outline /
               ColorModel=(GraphWalls:Color ThreeColorRamp:StartColor)
               display=all includemissingcolor=false fillattrs=(transparency=1)
               outlineattrs=graphdata2(thickness=1);
            * Textplot provides the values;
            textplot y=RowLab x=ColLab text=eval(put(Value, 2.)) /
                     textattrs=(size=12px) position=center;
         endlayout;
      endgraph;
   end;
quit;

proc sgrender data=mat2 template=matrix;
run;
```

The next example creates the same Toeplitz matrix as before. However, this time the DATA step creates character row and column labels. The DATA step also creates a second variable that contains the cell indices. Larger cells are created (approximately 32×32 pixels), which are large enough to display the additional values. A second TEXTPLOT statement displays the indices. Each TEXTPLOT statement has a POSITION= option that positions text at either the top or bottom of each cell. The table is displayed in Figure 15.

```
data mat3;
   if _n_ = 1 then /* Scale matrix size by the number or rows/columns */
      call symputx('size', ceil(25 * sqrt(n)));
   set tmat(rename=(rowlab=r collab=c)) nobs=n;
   Outline = ifn(Value > 3, Value, .); /* Outline the nonmissing values (Value > 3) */
   RowLab = put(r, words5.);
   ColLab = put(c, words5.);
   Cell   = cats('(', r, ',', c, ')');
   output;
run;

proc template;
   define statgraph matrix;
      begingraph / designheight=%eval(250+&size) /* Size: a bit higher than wide */
                   designwidth =%eval(200+&size);
         entrytitle "Toeplitz Matrix";
         entrytitle "Values Greater Than Three Are Outlined";
         layout overlay / yaxisopts=(discreteopts=(tickvaluefitpolicy=none)
                                    display=(tickvalues) reverse=true)
                          xaxisopts=(discreteopts=(tickvaluefitpolicy=rotate)
                                    display=(tickvalues));
            * Heat map provides the background color for each cell;
            heatmapparm y=RowLab x=ColLab colorresponse=Value /
               ColorModel=(GraphWalls:Color ThreeColorRamp:StartColor);
            * Heat map provides the outlines;
            heatmapparm y=RowLab x=ColLab colorresponse=Outline /
               ColorModel=(GraphWalls:Color ThreeColorRamp:StartColor)
               display=all includemissingcolor=false fillattrs=(transparency=1)
               outlineattrs=graphdata2(thickness=1);
            * Textplot provides the values;
            textplot y=RowLab x=ColLab text=eval(put(Value, 6.)) /
                     position=top textattrs=(size=12px);
            textplot y=RowLab x=ColLab text=cell / position=bottom
                     textattrs=(size=12px);
         endlayout;
      endgraph;
   end;
quit;

proc sgrender data=mat3 template=matrix;
run;
```

Figure 15 Heat Map and Table: Two Values in a Cell

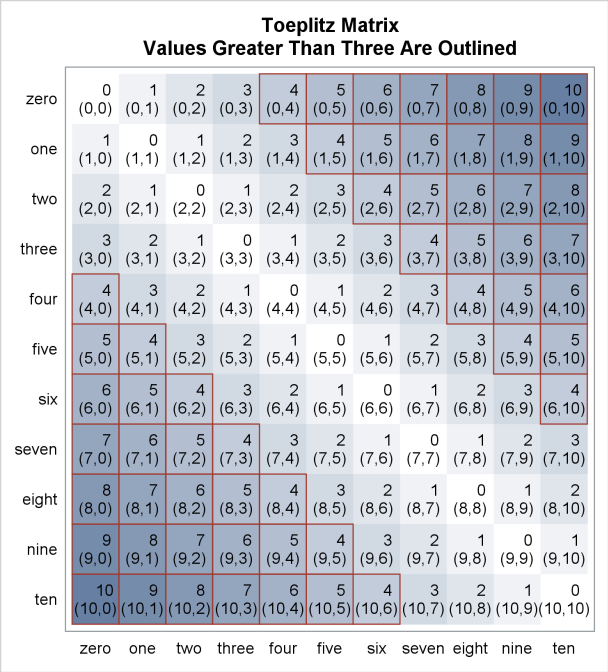
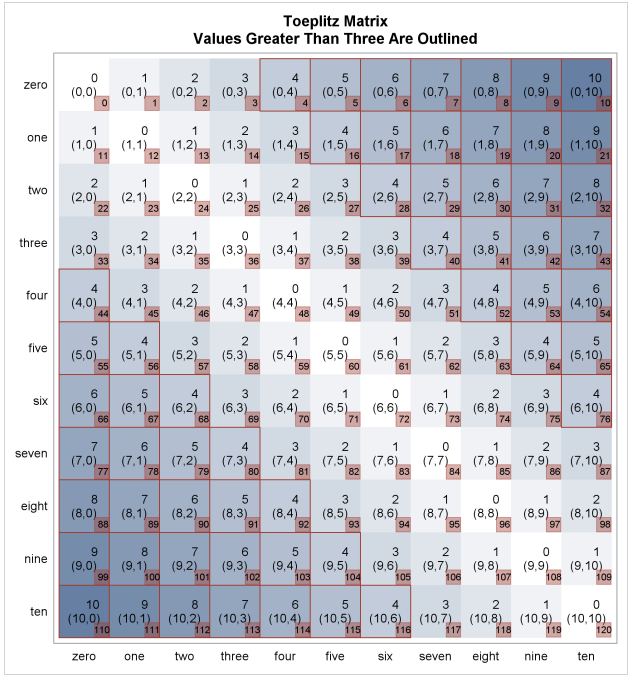


Figure 16 Heat Map and Table: Three Values in a Cell



The next example adds a box within each cell and adds values to each box. The first SCATTERPLOT statement adds the box to each cell. The second SCATTERPLOT statement adds text to the box. In both cases, DISCRETEOFFSET=0.35 shifts the center of the box and text 35% to the right, and both axes are discrete. The table is displayed in Figure 16.

```
data mat4;
   if _n_ = 1 then /* Scale matrix size by the number or rows/columns */
      call symputx('size', ceil(40 * sqrt(n)));
   set tmat(rename=(rowlab=r collab=c)) nobs=n;
   Outline = ifn(Value > 3, Value, .); /* Outline the nonmissing values (Value > 3) */
   RowLab = put(r, words5.);
   ColLab = put(c, words5.);
   Cell   = cats('(', r, ',', c, ')');
   output;
   Entry  + 1;
run;

proc template;
   define statgraph matrix;
      begingraph / designheight=%eval(250+&size) /* Size: a bit higher than wide */
                   designwidth =%eval(200+&size);
         entrytitle "Toeplitz Matrix";
         entrytitle "Values Greater Than Three Are Outlined";
         layout overlay / yaxisopts=(discreteopts=(tickvaluefitpolicy=none)
                                    display=(tickvalues) reverse=true)
                          xaxisopts=(discreteopts=(tickvaluefitpolicy=rotate)
                                    display=(tickvalues));
            * Heat map provides the background color for each cell;
            heatmapparm y=RowLab x=ColLab colorresponse=Value /
               ColorModel=(GraphWalls:Color ThreeColorRamp:StartColor);
            * Heat map provides the outlines;
            heatmapparm y=RowLab x=ColLab colorresponse=Outline /
               ColorModel=(GraphWalls:Color ThreeColorRamp:StartColor)
               display=all includemissingcolor=false fillattrs=(transparency=1)
               outlineattrs=graphdata2(thickness=1);
            * Textplot provides the values;
```

```
            textplot y=RowLab x=ColLab text=eval(put(Value, 7.)) /
                     position=top textattrs=(size=12px);
            textplot y=RowLab x=ColLab text=cell / position=bottom
                     textattrs=(size=12px);
            * Add a small square;
            scatterplot y=RowLab x=ColLab /
               markerattrs=graphdata2(symbol=squarefilled size=15 transparency=0.6)
               discreteoffset=0.35;
            * Add text to the square;
            scatterplot y=RowLab x=ColLab / markercharacter=eval(put(entry, 3.))
               discreteoffset=0.35;
         endlayout;
      endgraph;
   end;
quit;

proc sgrender data=mat4 template=matrix;
run;
```

HEAT MAPS FOR ENHANCING CORRELATION MATRICES

The %MktEx autocall macro creates orthogonal and nonorthogonal experimental designs. The %MktEval macro displays canonical correlations between coded factors and other information, including a heat map of the canonical correlations. The following steps create the table in Figure 17:

```
%mktex(2 ** 9 3 4 4, n=18, seed=205)

%mkteval(print=graph)
```

Figure 17 displays colors but no values. This works well when there are many variables. When there are not many variables, as in this example, you can take explicit control of the heat map by using PROC SGPLOT. The following step transposes the OUTCORR=**Corr** data set of canonical correlations, which the %MktEval macro creates by default:

```
proc transpose data=corr(rename=(_name_=RowLab))
               out=t(rename=(col1=Value _name_=ColLab));
   by notsorted rowlab;
run;
```

This creates a SAS data set that is arrayed for plotting that contains row labels, column labels, and values. The following steps create the final data set for plotting and create the graph displayed in Figure 18:

```
data t2;
   length v $ 5;
   set t;
   v = ifc(-1e-8 le value le 1e-8, '  0',
       ifc(rowlab eq collab,       ' ---', put(value, 5.2)));
   AbsValue = ifn(rowlab eq collab, 0, abs(value));
run;

ods graphics on / height=3.25in width=3in;
proc sgplot noautolegend;
   title h=7pt 'Factor Correlations in a Nonorthogonal Design';
   heatmapparm y=rowlab x=collab colorresponse=absvalue /
               colormodel=(cxFAFBFE cx667FA2 cxD05B5B);
   text        y=rowlab x=collab text=v;
   %let opts = display=(nolabel noticks) valueattrs=(size=7)
               offsetmin=0.05 offsetmax=0.05;
   xaxis &opts;
   yaxis &opts reverse;
run;
title;
```

Figure 17 Canonical Correlations from %MktEval

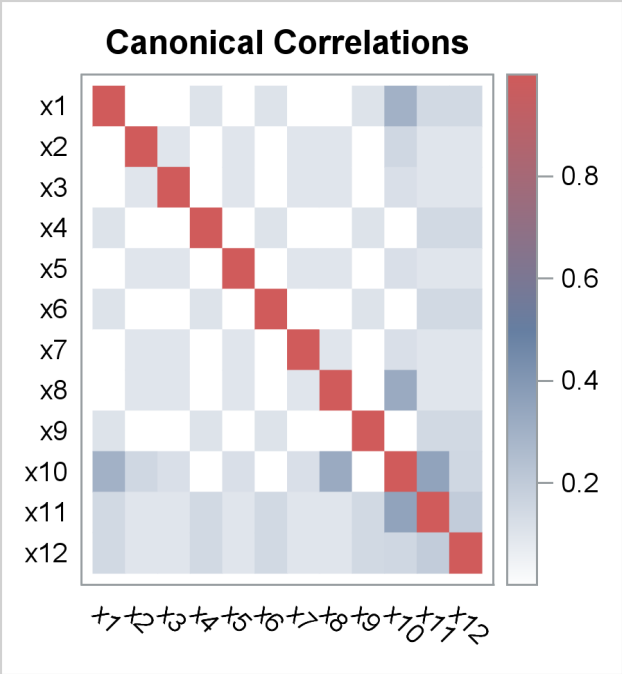
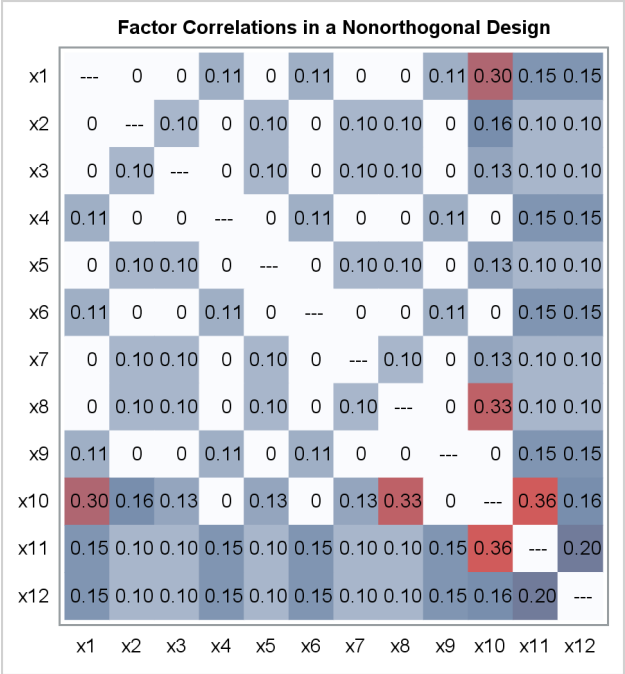


Figure 18 Canonical Correlations from PROC SGPLOT



The DATA step creates two variables: **v** contains the character string to display, and **AbsValue** contains the color response variable for the heat map. Dashes are displayed on the diagonal. When the off-diagonal correlation is essentially zero, the numeral 0 is displayed; otherwise, the formatted correlation is displayed. The heat map is constructed from the absolute value of the correlation (ignoring the ones on the diagonal). PROC SGPLOT creates the table along with the heat map and the customized formatting of the canonical correlations.

Canonical correlations are nonnegative. However, the preceding steps use a 5.2 format, which reserves room for a minus sign. The preceding steps were designed to work equally well using as input a matrix of Pearson correlations that is created as follows:

```
proc corr data=design outp=corr(where=(_type_='CORR')) noprint;
run;
```

TABLE TEMPLATES

You can also display colors in tables. The following step modifies the table template so that the cell background is displayed in shades of cyan, magenta, and red as the absolute value of each correlation coefficient ranges from 0 to 1:

```
proc template;
   delete Base.Corr.StackedMatrix / store=sasuser.templat;
   edit Base.Corr.StackedMatrix;
      column (RowName RowLabel) (Matrix);
      header 'Pearson Correlation Coefficients';
      edit matrix;
         format=5.2;
         cellstyle
            _val_ <= -1    as {backgroundcolor=CXFF0000},
            _val_ <= -0.75 as {backgroundcolor=CXFF007D},
            _val_ <= -0.5  as {backgroundcolor=CXFF00FA},
            _val_ <= -0.25 as {backgroundcolor=CX7D82FF},
            _val_ <=  0    as {backgroundcolor=CX00FFFF},
            _val_ <=  0.25 as {backgroundcolor=CX7D82FF},
            _val_ <=  0.5  as {backgroundcolor=CXFF00FA},
            _val_ <=  0.75 as {backgroundcolor=CXFF007D},
            _val_ <=  1    as {backgroundcolor=CXFF0000};
```

```
      end;
   end;
quit;

ods pdf style=sapphire;
options nolabel;
proc corr data=sashelp.cars(drop=mpg:) noprob;
   ods select PearsonCorr;
run;
```

The CELLSTYLE statement enables you to specify colors for ranges of values. The results are displayed in Figure 19. The table uses the Sapphire style, which is similar to the HTMLBlue style but is more appropriate for the PDF and RTF destinations.

Figure 19 Changing Background Colors in a Table

Pearson Correlation Coefficients

	MSRP	Invoice	EngineSize	Cylinders	Horsepower	Weight	Wheelbase	Length
MSRP	1.00	1.00	0.57	0.65	0.83	0.45	0.15	0.17
Invoice	1.00	1.00	0.56	0.65	0.82	0.44	0.15	0.17
EngineSize	0.57	0.56	1.00	0.91	0.79	0.81	0.64	0.64
Cylinders	0.65	0.65	0.91	1.00	0.81	0.74	0.55	0.55
Horsepower	0.83	0.82	0.79	0.81	1.00	0.63	0.39	0.38
Weight	0.45	0.44	0.81	0.74	0.63	1.00	0.76	0.69
Wheelbase	0.15	0.15	0.64	0.55	0.39	0.76	1.00	0.89
Length	0.17	0.17	0.64	0.55	0.38	0.69	0.89	1.00

The syntax of the CELLSTYLE statement is convenient when you want to specify a few arbitrary colors—for example, to control the background for significant p-values. It is not convenient when you want more colors, so the rest of this section shows alternative methods.

The SAS autocall macro %Paint performs color interpolation. The following step produces a table that matches the one displayed in Figure 19:

```
%paint(values=-1 to 1 by 0.25, macro=setstyle, colors=red magenta cyan magenta red)

proc template;
   delete Base.Corr.StackedMatrix / store=sasuser.templat;
   edit Base.Corr.StackedMatrix;
      column (RowName RowLabel) (Matrix);
      header 'Pearson Correlation Coefficients';
      edit matrix;
         format=5.2;
         %setstyle(backgroundcolor)
      end;
   end;
quit;

proc corr data=sashelp.cars(drop=mpg:) noprob;
   ods select PearsonCorr;
run;
```

The %Paint macro accepts a list of values (in this case, –1 to 1 by 0.25) and a list of colors (red, magenta, cyan, magenta, and red). You can optionally specify a number list after the color list. The number list specifies the precise numeric value that is mapped to each color. If you omit the number list (as in this example), the macro creates a default list. The %Paint autocall macro creates a macro called %SetStyle that provides the CELLSTYLE statement.

The following steps use an alternative color list and create the table in Figure 20:

```
%paint(values=-1 to 1 by 0.01, macro=setstyle,
       colors=CXFF6767 magenta CX6767FF cyan white white white
              white white white cyan CX6767FF magenta CXFF6767)

proc template;
   delete Base.Corr.StackedMatrix / store=sasuser.templat;
   edit Base.Corr.StackedMatrix;
      column (RowName RowLabel) (Matrix);
      header 'Pearson Correlation Coefficients';
      edit matrix;
         format=5.2;
         %setstyle(backgroundcolor)
      end;
   end;
quit;

proc corr data=sashelp.cars(drop=mpg:) noprob;
   ods select PearsonCorr;
run;
```

Colors are repeated so that more of the smaller correlations have white backgrounds.

Figure 20 Color Ramp from White to Blue to Red

Pearson Correlation Coefficients								
	MSRP	Invoice	EngineSize	Cylinders	Horsepower	Weight	Wheelbase	Length
MSRP	1.00	1.00	0.57	0.65	0.83	0.45	0.15	0.17
Invoice	1.00	1.00	0.56	0.65	0.82	0.44	0.15	0.17
EngineSize	0.57	0.56	1.00	0.91	0.79	0.81	0.64	0.64
Cylinders	0.65	0.65	0.91	1.00	0.81	0.74	0.55	0.55
Horsepower	0.83	0.82	0.79	0.81	1.00	0.63	0.39	0.38
Weight	0.45	0.44	0.81	0.74	0.63	1.00	0.76	0.69
Wheelbase	0.15	0.15	0.64	0.55	0.39	0.76	1.00	0.89
Length	0.17	0.17	0.64	0.55	0.38	0.69	0.89	1.00

You can use the %Paint macro to create an output data set, which by default is named **Colors**. The following steps create a color interpolation format and use it to control the text colors:

```
%let inc = 0.01;
%paint(values=-1 to 1 by &inc, colors=red magenta cyan magenta red)

data cntlin;
   set colors;
   FmtName = 'paintfmt';
   Label = _rgb_;
   End = round(start + &inc, &inc);
   keep start end label fmtname;
run;
```

```
proc format cntlin=cntlin;
run;

proc template;
   delete Base.Corr.StackedMatrix / store=sasuser.templat;
   edit Base.Corr.StackedMatrix;
      column (RowName RowLabel) (Matrix);
      header 'Pearson Correlation Coefficients';
      edit matrix;
         format=5.2 style = {foreground=paintfmt8. font_weight=bold};
      end;
   end;
quit;

proc corr data=sashelp.cars(drop=mpg:) noprob;
   ods select PearsonCorr;
run;
ods pdf close;
```

Each observation in the **Colors** data set contains a value in the specified VALUES= option range and a color. Because both a small increment and the default value for the RGBROUND= option are used, the number of different colors (50) is smaller than the number of observations (1 + (*max* – *min*) / *increment*) = 201). The **Colors** data set is not in the correct form to make a format, so the DATA step creates a CNTLIN= data set, which PROC FORMAT uses to make the format. Then the format is specified in the STYLE= option to control the colors without your having to specify a long CELLSTYLE statement. The results are displayed in Figure 21.

Figure 21 Changing Foreground Colors in a Table

Pearson Correlation Coefficients								
	MSRP	Invoice	EngineSize	Cylinders	Horsepower	Weight	Wheelbase	Length
MSRP	1.00	1.00	0.57	0.65	0.83	0.45	0.15	0.17
Invoice	1.00	1.00	0.56	0.65	0.82	0.44	0.15	0.17
EngineSize	0.57	0.56	1.00	0.91	0.79	0.81	0.64	0.64
Cylinders	0.65	0.65	0.91	1.00	0.81	0.74	0.55	0.55
Horsepower	0.83	0.82	0.79	0.81	1.00	0.63	0.39	0.38
Weight	0.45	0.44	0.81	0.74	0.63	1.00	0.76	0.69
Wheelbase	0.15	0.15	0.64	0.55	0.39	0.76	1.00	0.89
Length	0.17	0.17	0.64	0.55	0.38	0.69	0.89	1.00

A few of the observations in the **Cntlin** data set are displayed in Figure 22.

Figure 22 Creating a Format from a Data Set

Start	FmtName	Label	End
-1.00	paintfmt	CXFF0000	-0.99
-0.99	paintfmt	CXFF0000	-0.98
-0.98	paintfmt	CXFF000A	-0.97
.			.
.			.
.			.
-0.02	paintfmt	CX0AF5FF	-0.01
-0.01	paintfmt	CX00FFFF	0.00
0.00	paintfmt	CX00FFFF	0.01
.			.
.			.
.			.
0.98	paintfmt	CXFF000A	0.99
0.99	paintfmt	CXFF0000	1.00
1.00	paintfmt	CXFF0000	1.01

You can simultaneously control the colors of values and perform any other customization. This next example displays only the lower triangle of the correlation matrix:

```
proc template;
   delete Base.Corr.StackedMatrix / store=sasuser.templat;
   edit Base.Corr.StackedMatrix;
      column (RowName) (Matrix);
      header 'Pearson Correlation Coefficients';
      edit matrix;
         translate _val_ = ._ into ' ';
         format=5.2 style = {foreground=paintfmt8. font_weight=bold};
      end;
      edit rowname;
         header= ' ';
      end;
   end;
   source Base.Corr.StackedMatrix;
quit;

ods select none;
proc corr data=sashelp.cars(drop=mpg:) noprob;
   ods output PearsonCorr=p;
run;
ods select all;

ods pdf style=sapphire;
data p2;
   set p end=eof;
   array __n[*] _numeric_;
   do __i = _n_ to dim(__n); __n[__i] = ._; end;
   if _n_ = 1 then do;
      call execute('data _null_; set p2;');
      call execute('file print ods=(template="Base.Corr.StackedMatrix"');
      call execute('columns=(rowname=variable');
   end;
   call execute(cats('matrix=',vname(__n[_n_]),'(generic)'));
   if eof then call execute('); put _ods_; run;');
run;
ods pdf close;
```

The TRANSLATE statement in the template displays underscore missing values (designated as '._') as blanks. PROC CORR creates a SAS data set that contains the correlation matrix. The DATA step simultaneously sets the upper triangle to underscore missing and generates DATA step code to display the correlation matrix. Columns in a correlation matrix are generic. That is, a single generic column definition in the template is used for every variable in the data set. The columns are not predictable in a correlation matrix, so appropriate code is needed to use the correlation matrix template outside PROC CORR.

The CALL EXECUTE statement writes code to a buffer. That code is executed when the DATA step completes. On the first pass, the DATA step generates the DATA and SET statements and starts writing the FILE statement. Then for every observation, the CALL EXECUTE statement generates more of the FILE statement and maps each variable to the generic column **Matrix**. On the last pass, it completes the FILE statement and adds the PUT and RUN statements. The DATA step and CALL EXECUTE statements generate the following code to display the correlation matrix:

```
data _null_;
   set p2;
   file print ods=(template="Base.Corr.StackedMatrix"
        columns=(rowname=variable
                 matrix=MSRP(generic)
                 matrix=Invoice(generic)
                 matrix=EngineSize(generic)
                 matrix=Cylinders(generic)
                 matrix=Horsepower(generic)
                 matrix=Weight(generic)
                 matrix=Wheelbase(generic)
                 matrix=Length(generic)));
   put _ods_;
run;
```

The COLUMNS= option assigns each of the variable names in the data set (**Variable** and **MSRP** through **Length**) to the appropriate ODS table column (**RowName** and **Matrix**). You can see the generated code by submitting the following statement before you submit a DATA step that contains CALL EXECUTE statements:

```
options source;
```

The correlation matrix is displayed in Figure 23.

Figure 23 Lower Triangle of a Correlation Matrix

Pearson Correlation Coefficients

	MSRP	Invoice	EngineSize	Cylinders	Horsepower	Weight	Wheelbase	Length
MSRP								
Invoice	1.00							
EngineSize	0.57	0.56						
Cylinders	0.65	0.65	0.91					
Horsepower	0.83	0.82	0.79	0.81				
Weight	0.45	0.44	0.81	0.74	0.63			
Wheelbase	0.15	0.15	0.64	0.55	0.39	0.76		
Length	0.17	0.17	0.64	0.55	0.38	0.69	0.89	

COLOR INTERPOLATION

The %Paint macro creates colors by linearly interpolating between RGB values. Red is CXFF000 (R=255, G=0, B=0). Green is CX00FF00 (R=0, G=255, B=0). Blue is CX0000FF (R=0, G=0, B=255). Halfway between red and blue is CX800080 (R=128, G=0, B=128), which is not magenta (CXFF00FF). Halfway between blue and green is CX008080 (R=0, G=128, B=128), which is not cyan (CX00FFFF). Halfway between red and green is CX808000 (R=128, G=128, B=0), which is not yellow (CXFFFF00). In order to get more natural color interpolation, specify a list such as red, magenta, blue; red, yellow, green; or green, cyan, blue instead of red, blue; red, green; or green, blue. The %Paint macro can interpolate to create up to 256^3 intermediate colors. However, the macro limits the number of colors. Specify the RGBROUND= option to control or remove this limit.

SG ANNOTATION

This example combines techniques from two previous examples and uses statistical graphics (SG) annotation. The goal is to use the color of the state labels to represent population density in a US map. You can do this without SG annotation, but using the %Paint macro and SG annotation gives you complete control over the colors.

The following step divides the **Map** data set from the first example into a data set that contains the state boundaries and a data set that contains labels and label coordinates:

```
data map2(keep=id x y state)
     anno0(keep=xcen ycen label rename=(xcen=x1 ycen=y1));
   set map;
   if label eq ' ' then output map2;
   else                 output anno0;
run;
```

The following steps sort two data sets that are needed to create the SG annotation data set:

```
proc sort data=anno0;
   by label;
run;

proc sort data=sashelp.us_data(keep=density_2010 statecode)
          out=USDensity(where=(not (statecode in ('DC' 'PR'))));
   by statecode;
run;
```

The %Paint macro processes the **Density_2010** variable in the **USDensity** data set and creates a mapping between the population densities and a color ramp that ranges from black to blue to magenta to red:

```
%paint(var=density_2010, colors=black blue magenta red)
```

The **_RGB_** variable in the **Colors** data set contains the colors. SG annotation requires specific variable names, so the DATA step renames it **TextColor**. The following step creates the SG annotation data set:

```
data anno;
   merge anno0 colors(drop=_obstat_ den: rename=(_rgb_ = TextColor statecode=label));
   by label;
   DrawSpace  = 'DataValue';
   Function   = 'Text';
   TextSize   = 6;
   TextWeight = 'Bold';
run;
```

The first five observations of the SG annotation data set are displayed in Figure 24.

Figure 24 Five Observations from the SG Annotation Data Set

x1	y1	Label	TextColor	DrawSpace	Function	TextSize	TextWeight
-0.31184	-0.12348	AK	CX000000	DataValue	Text	6	Bold
0.13422	-0.07364	AL	CX00003D	DataValue	Text	6	Bold
0.04808	-0.04095	AR	CX000000	DataValue	Text	6	Bold
-0.22859	-0.03377	AZ	CX000026	DataValue	Text	6	Bold
-0.32985	0.04410	CA	CX00009C	DataValue	Text	6	Bold

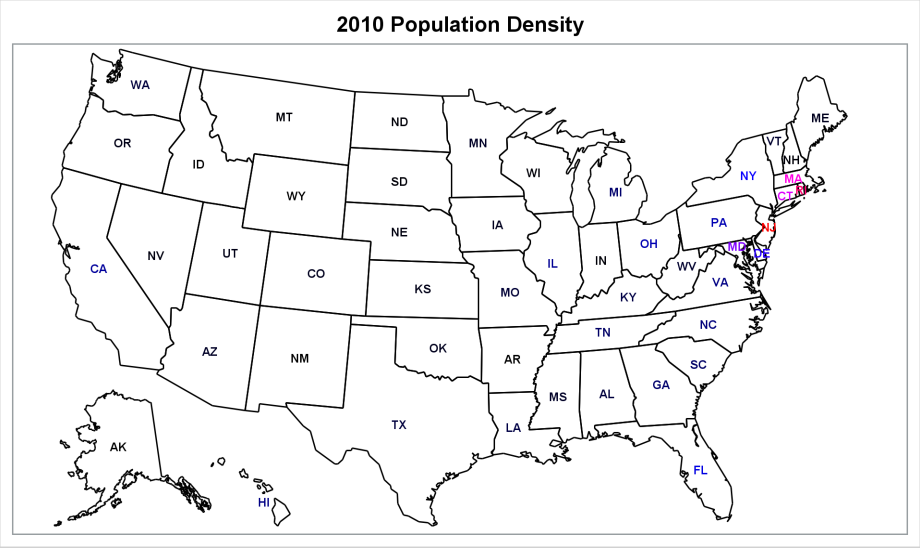
The X and Y coordinates are stored in the variables **x1** and **y1**, respectively. Because **DrawSpace** = `'DataValue'`, both coordinates are data values. That is, both coordinates are on the same scale as the X= and Y= variables that are specified in the POLYGON statement in PROC SGPLOT. The **Label** variable specifies the text, and the **TextColor** variable specifies the interpolated colors.

The following step creates the map displayed in Figure 25:

```
ods graphics on / height=3.8in width=6.4in;
proc sgplot data=map2 noautolegend sganno=anno;
   title '2010 Population Density';
   polygon x=x y=y id=id / outline lineattrs=(color=black);
   xaxis offsetmin=0.01 offsetmax=0.01 display=none;
   yaxis offsetmin=0.01 offsetmax=0.01 display=none;
run;
ods graphics on / reset=all;
```

The SGANNO= option names the SG annotation data set.

Figure 25 Using SG Annotation to Display Population Density



CONCLUSION

Heat maps enable you to display information in maps, graphs, and tables without adding extra words, numbers, or space to the display. In maps, heat maps display spatial and numeric information simultaneously. In graphs, heat maps show you patterns that are obscured in other types of graphs, such as scatter plots. In tables, heat maps call attention to key values. ODS Graphics provides options for creating heat maps, or you can explicitly control colors by using the %Paint macro.

ACKNOWLEDGMENTS

The author is grateful to his editor, Ed Huddleston, for his helpful comments.

RECOMMENDED READING

For more information about the %Paint macro, see https://support.sas.com/techsup/technote/mr2010paint.pdf. You can view the macro code, including comments that provide examples and descriptions of the options, by submitting the following statement to SAS:

```
%inc sasautos(paint.sas) / src;
```

For complete documentation about ODS and ODS Graphics, see *SAS Output Delivery System: User's Guide*, *SAS Graph Template Language: Reference*, *SAS ODS Graphics: Procedures Guide*, and *SAS Graph Template Language: User's Guide*. For more examples and documentation, see Chapter 20, "Using the Output Delivery System"; Chapter 21, "Statistical Graphics Using ODS"; Chapter 22, "ODS Graphics Template Modification"; and Chapter 23, "Customizing the Kaplan-Meier Survival Plot" in the *SAS/STAT User's Guide*.

For a gentle and parallel introduction to PROC SGPLOT and the GTL, see the free web book *Basic ODS Graphics Examples* (http://support.sas.com/documentation/prod-p/grstat/9.4/en/PDF/odsbasicg.pdf).

For more advanced topics, including an introduction to SG annotation, see the free web book *Advanced ODS Graphics Examples* (http://support.sas.com/documentation/prod-p/grstat/9.4/en/PDF/odsadvg.pdf).

For tips, tricks, and the latest developments in ODS Graphics, see Sanjay Matange's blog *Graphically Speaking* (http://blogs.sas.com/content/graphicallyspeaking/) and his books (http://support.sas.com/publishing/authors/matange.html).

CONTACT INFORMATION

Warren F. Kuhfeld
SAS Institute Inc.
Cary, NC 27513
(919) 531-7922
Warren.Kuhfeld@sas.com