

Design better graphs: Tips and Tricks for efficient charts and plots

Aaron Winkel, Physician Resource Northwest, Portland, OR

ABSTRACT

A good graph is easy on the eyes and attracts the viewer to the content and information contained in the graph. There are default features in SAS/GRAPH that are not always necessary – borders, frames, gridlines, legends, etc. With a little TLC, graphs can be scaled down to wash away the “busyness”. This paper will demonstrate techniques for eliminating chartjunk to enhance the graphical presentation of information. At the same time, it will show ways to augment the relevant portions of the graph.

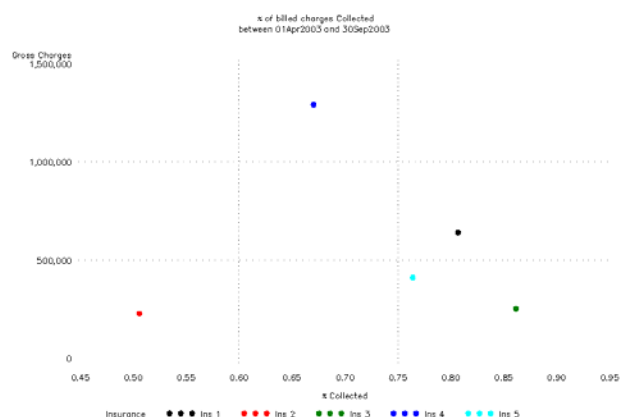
INTRODUCTION

A few years ago, I attended an Edward Tufte seminar about presenting data and information. I was very impressed with his presentation and came to realize that the graphs I was producing left much to be desired. As such, I began including code on a regular basis that would eliminate chartjunk from my graphs. Many of the suggestions below are things I incorporated in my graphs after the Tufte seminar, and some are things I've learned over the years. These tips are not absolutes, but are helpful in many cases to visually display data in a clear and concise manner.

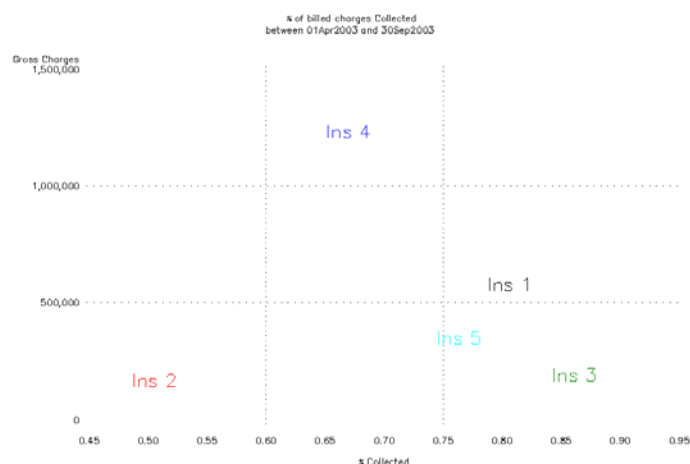
#1. Eliminate the legend, if possible

Legends in and of themselves are not bad and can contain relevant information. The basic problem with the legend is that it is often not part of the graph and the viewer's eyes must yo-yo back and forth between the graph and the legend. A better approach is to put the information from the legend into the graph itself.

In this example, one must use the legend to determine what each dot represents:



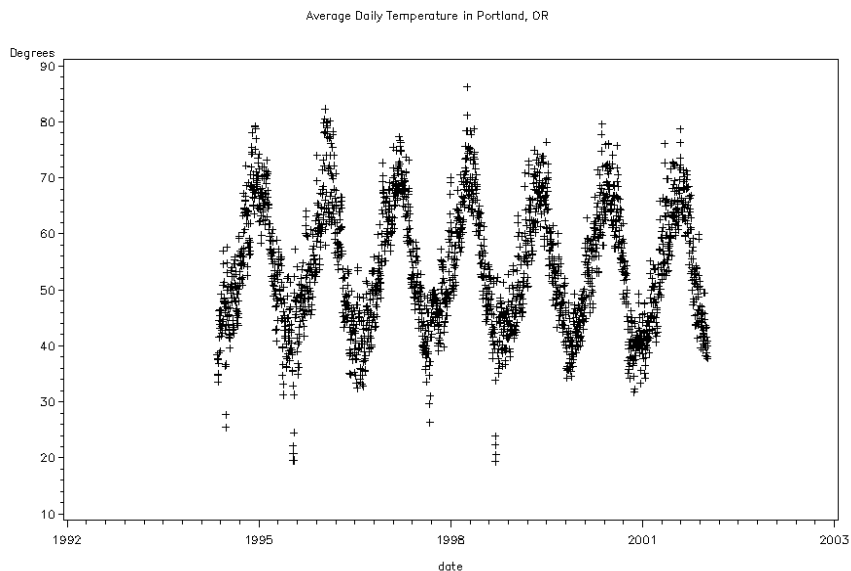
With the use of the annotate facility, the text is now available on the graph itself and the comparisons can occur much faster. The legend is no longer necessary.



#2. Eliminate Chartjunk

In *The Visual Display of Quantitative Information*, Tufte states, “The interior decoration of graphics generates a lot of ink that does not tell the viewer anything new. The purpose of decoration varies – to make the graphic appear more scientific and precise, to enliven the display, to give the designer an opportunity to exercise artistic skills. Regardless of its cause, it is all non-data-ink or redundant data-ink, and it is often chartjunk”. Chartjunk refers to the elements in a graph that are not necessary – tick marks, borders, legends, frames, etc.

Using default proc gplot code, the following graph of Portland’s average daily temperature is laden with chartjunk.



The code below helps to eliminate much of the chartjunk and makes for a better graph.

```
goptions reset=all ftext=simplex htext=1;
axis1 order=(0 to 100 by 10);
axis2 order=("01jan1995"d to "31dec2002"d by year);
symbol1 value=circle interpol=none width=1;
proc gplot data=temp;
    title h=1 "Average Daily Temperature in Portland, OR";
    label temp="Degrees";
    plot temp*date / vaxis=axis1 haxis=axis2 vref=30,80 lvref=33 caxis=white cvref=black;
run;
quit;
```

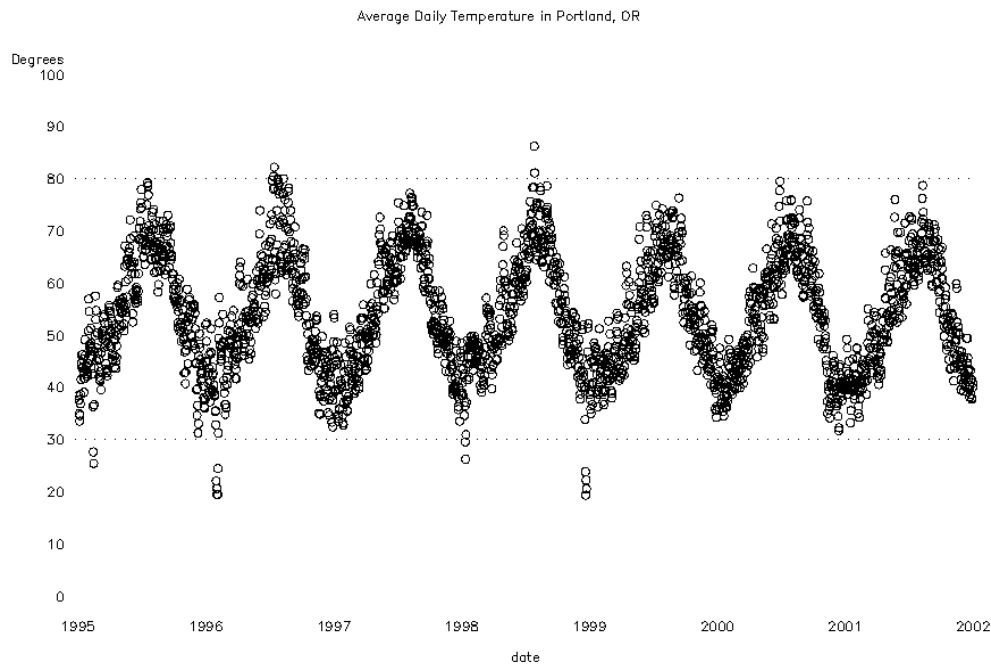
Vaxis and **Haxis** are used to bring in the axis properties of axis1 and axis2, respectively.

Vref puts a vertical reference line at points 30 and 80.

Caxis changes the axis color to white so it will blend in with the background.

Cvref instructs SAS to make the vertical reference lines black.

The result is below:



#3. Use Reference Lines as necessary

There are 46 different line styles that can be added to graphs. They should be used in cases where they make a graph easier to read. A table of line styles is below:

Line Types for Vertical and Horizontal Reference Lines

Number		Number
1	————	1
2	————	2
3	————	3
4	————	4
5	————	5
6	————	6
7	————	7
8	————	8
9	————	9
10	————	10
11	————	11
12	————	12
13	————	13
14	————	14
15	————	15
16	————	16
17	————	17
18	————	18
19	————	19
20	————	20
21	————	21
22	————	22
23	————	23
24	————	24
25	————	25
26	————	26
27	————	27
28	————	28
29	————	29
30	————	30
31	————	31
32	————	32
33	————	33
34	————	34
35	————	35
36	————	36
37	————	37
38	————	38
39	————	39
40	————	40
41	————	41
42	————	42
43	————	43
44	————	44
45	————	45
46	————	46

This table can be designed using SAS/GRAPH with the following code:

```
* Reference Lines graph;
```

```
data l;
  do line=1 to 46 by 1;
    line2=line;
    output;
  end;
run;
```

```
data _null_;
```

```

set l end=last;
length linetype $200;
retain linetype;
linetype=compbl(trim(linetype)||' '||put(line,2.));
if last then call symput('linetypes',linetype);
run;

goptions reset=all;
axis1 order=(46 to 1 by -1);
axis2 c=white;
symbol1 v=none;
proc gplot data=l;
title "Line Types for Vertical and Horizontal Reference Lines";
label line='Number' line2='00'x;
plot line*line2 / vref=(&linetypes) lvref=(&linetypes) cvref=black vaxis=axis1
caxis=white haxis=axis2;
plot2 line*line2 / vaxis=axis1 caxis=white;
run;
quit;

```

Each reference line can be tailor made in terms of color and line style. The following code will place vertical reference lines at 0, 30, and 80. The one at 0 will have a line style of 1 (solid) and the color blue. The other 2 will have a style of 33 (dotted line) and will be black. This is done using vref, lvref, and cvref.

```

proc gplot data=temp;
plot temp*date / vaxis=axis1 haxis=axis2 vref=0,30,80 lvref=(1 33 33) cvref=(blue black
black) chref=black caxis=white lhref=33 href=("01Jan1995"d);
run;
quit;

```

#4. Use a vertical axis on both sides of a plot

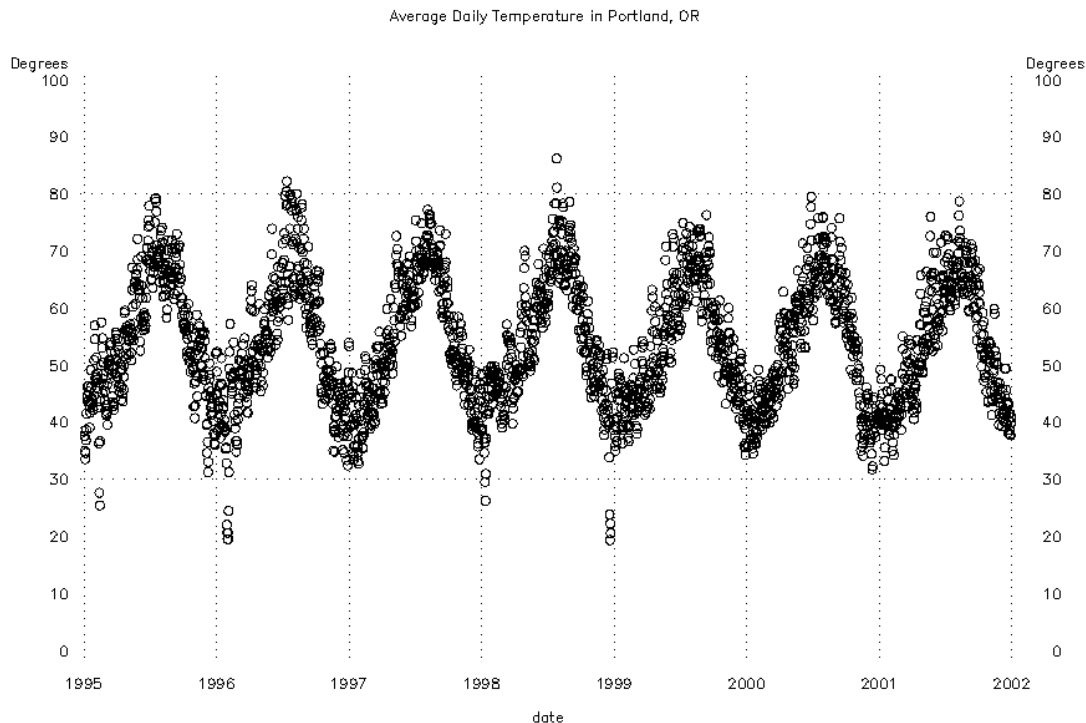
Since we read graphs from left to right in many cases, it is often helpful to add a vertical axis to the right side. This second vertical axis will assist the eyes and prevent them yo-yoing back to the left side to get a sense of scale. This is particularly helpful when the data values are not included in the graph. Be sure to use the appropriate scale on both axes since differing scales can lead to misinterpretations. It is also important to ensure that the symbol statements mirror each other. The key is to use a plot2 statement.

```

goptions reset=all ftext=simplex htext=1;
axis1 order=(0 to 100 by 10);
axis2 order=("01jan1995"d to "31dec2002"d by year);
symbol1 value=circle interpol=none c=black;
symbol2 value=circle interpol=none c=black;
proc gplot data=temp;
title h=1 "Average Daily Temperature in Portland, OR";
format date year4.;
label temp="Degrees";
plot temp*date / vaxis=axis1 haxis=axis2 vref=30,80 lvref=34 caxis=white cvref=black
chref=black lhref=34
href=("01Jan1995"d "01Jan1996"d "01Jan1997"d "01Jan1998"d "01Jan1999"d "01Jan2000"d
"01Jan2001"d "01Jan2002"d);
plot2 temp*date / vaxis=axis1 caxis=white;
run;
quit;

```

The result is below:



#5. Place the “start” values in the lower left corner when designing plots

The scale used when displaying plots depends on the data, but as a general rule, it is wise to begin the data values in the lower left corner. Whenever possible, use 0 as the start value of the vertical and/or horizontal axis.

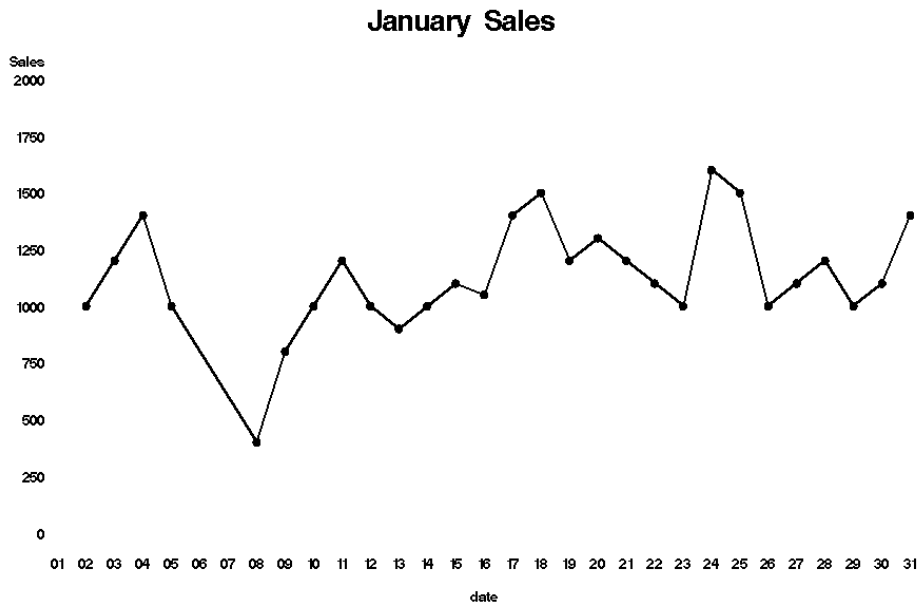
#6. Create dummy data as needed

There are situations where one of the variables being plotted is not continuous. For example, a business that is closed for a holiday might not have any sales on that day, though they would for all other days that month. When using `interpol=join` in the symbol statement, the line connects each of the plotted points and does not take “missing” values (a date in this case) into account. Using dummy data is one way to work around this.

Suppose we had the following dataset:

Date	Sales
02Jan2004	1000
3Jan2004	1200
04Jan2004	1400
05Jan2004	1000
08Jan2004	400
09Jan2004	800
10Jan2004	1000
11Jan2004	1200
.....	
31Jan2004	1400

Notice that January 1, 6, and 7 are missing from the data. If we graph the plot as is, we get the following:



Since this graph is using `interpol=join` on the `symbol` statement, the value from the 5th is connected with the value from the 8th as the 6th and 7th don't exist. To correct this, we can create some dummy data with all of the days in January and merge with our sales data.

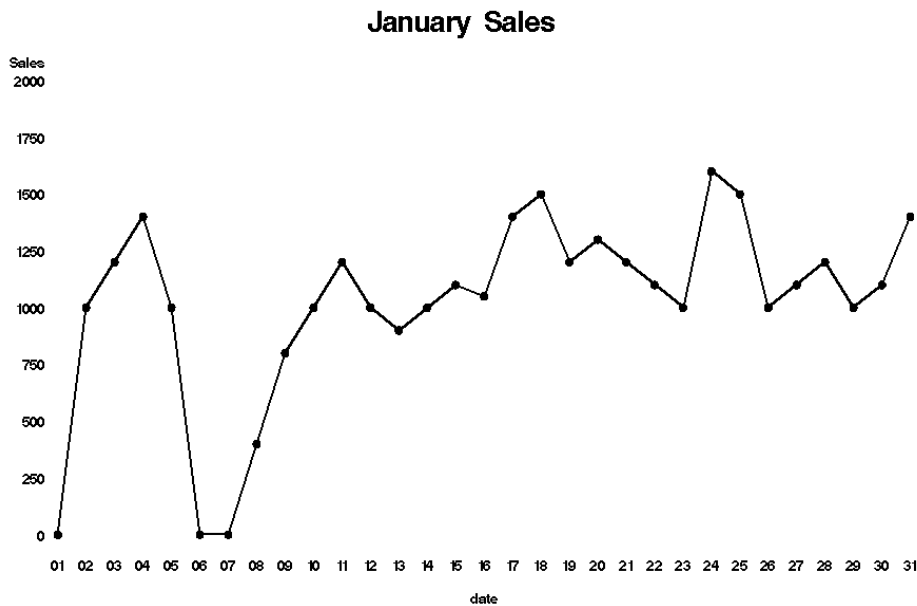
```
data dummy;
  do date="01Jan2004"d to "31Jan2004"d by 1;
    output;
  end;
run;

data sales2;
  merge dummy sales;
  by date;
  if sales=. then sales=0;
run;
```

Part of the resulting dataset is as follows:

01Jan2004	0
02Jan2004	1000
03Jan2004	1200
04Jan2004	1400
05Jan2004	1000
06Jan2004	0
07Jan2004	0

The plot of `sales2` now reflects the previously missing data:



#7. Submit a phantom word after the "/" in a plot or chart statement to see what's available

Since there are so many options in SAS/GRAPH, it is difficult to remember them all. In those instances where I can't remember a particular option, I use a phantom word so that SAS will generate an error. The text in the error message is kind enough to list what SAS is expecting. From that list, I can usually identify the option I need.

```
24 proc gplot data=temp;
25 ...
29     plot temp*julian=year / Phantom_Word_Here vaxis=axis1 haxis=axis2 vref=0,40,60,80
caxis=white cvref=black chref=black
```

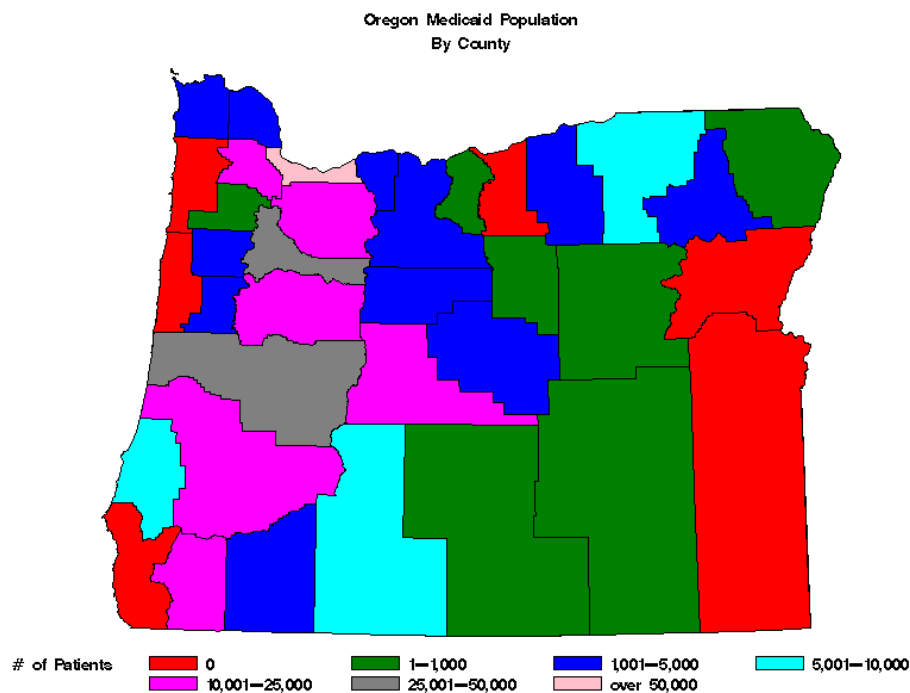
```
-----
22          202
```

ERROR 22-322: Syntax error, expecting one of the following: ;, ANNOTATE, AREAS, AUTOHREF, AUTOVREF, C, CAUTOHREF,CAUTOVREF, CAXIS, CFRAME, CHREF, COUTLINE, CTEXT, CVREF, DATAORDER, DESCRIPTION, FRAME, GRID, HAXIS, HMINOR, HREF, HREVERSE, HTML, HTML_LEGEND, HZERO, IFRAME, IMAGESTYLE, LAUTOHREF, LAUTOVREF, LEGEND, LHREF, LVREF, NAME, NOAXIS, NOFRAME, NOLEGEND, OVERLAY, REGEQN, SKIPMISS, VAXIS, VMINOR, VREF, VREVERSE, VZERO.
ERROR 202-322: The option or parameter is not recognized and will be ignored.

```
30     href=("01Jan1995"d);
NOTE: The previous statement has been deleted.
31     run;
```

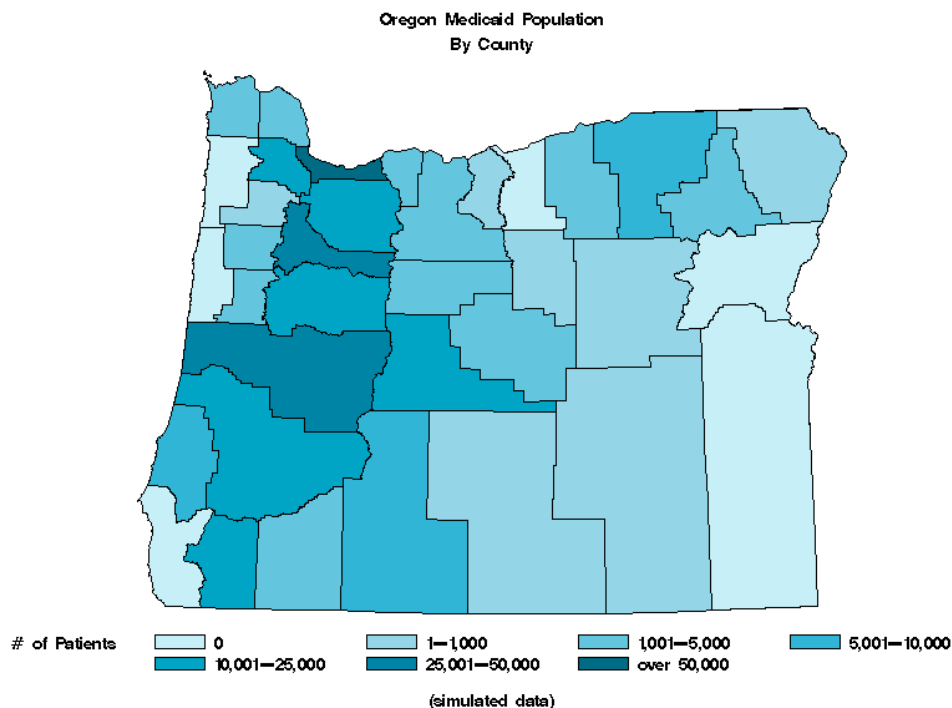
#8. Make a smooth transition by using different shades of the same color

Colors and patterns are vital elements of any graph. Use colors in such a way that they assist the viewer to decipher information quickly. The following map uses default colors and is hard to read:



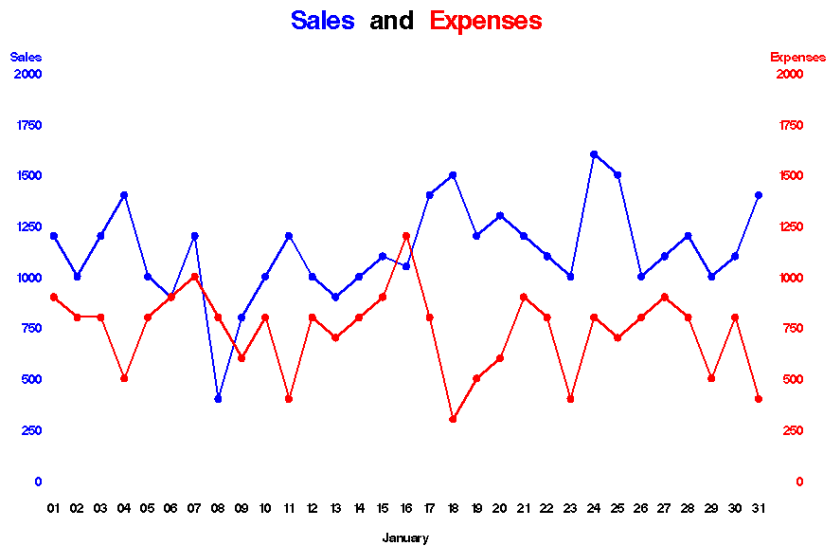
By changing the pattern, we can use different shades of the same color to make more populated areas stand out by giving them a darker shade.

```
*BLUES;
pattern1 v=s c="#C6EFF7";
pattern2 v=s c="#94D6E7";
pattern3 v=s c="#63C6DE";
pattern4 v=s c="#31B5D6";
pattern5 v=s c="#00A5C6";
pattern6 v=s c="#0084A5";
pattern7 v=s c="#006B84";
proc gmap data=products map=newmap all;
.....;
run;
quit;
```



#9. Use color to link titles, footnotes, etc. with various graphical elements

As consistent colors are used in titles, axes, plot lines, etc. they reinforce each other and demonstrate that they belong together. Consider the follow example: In the title, sales is blue, the vertical axis relating to sales is blue, and the plot line of sales is blue. Expenses are shown in like matter using red.



#10. To eliminate a variable label, use '00'x

The null hexadecimal string , '00'x, can be used to eliminate an unwanted variable label. This is useful when the variable's values contain enough information by themselves and the label is not necessary. I refer to this as the James Bond method since the '00' part reminds me of '007'.

```
proc gplot data=temp;  
    label temp="Degrees" date='00'x;  
    plot temp*date / vaxis=axis1 .....;  
run;  
quit;
```

#11. Use macro variables to establish axis boundaries

If we use axis statements to configure our axis boundaries, we must be careful to ensure they are wide enough to display all of the data points. For example, if sales increase dramatically and we have hard coded an upper bound that is not high enough, some of the data points may not be on our graph. One way to ensure this is to use macro variables so the boundaries are data-driven. In the example below, the highest value of sales is 26,000. Suppose we have determined through experience that we will graph sales in increments of 5,000. As such, our graph needs to go to 30,000 to include all data points. We can use a macro variable to determine the upper bound. Suppose we had the following data set:

Name	Sales
Jordan	12000
Bird	15000
Johnson	26000
Thomas	14000
Regional Avg	16750
Industry Avg	14000

The following code will determine the upper bound and store it in the macro variable upper:

```
proc summary data=sales missing nway;  
    var sales;  
    output out=maxsales max=max;  
run;
```

```
data _null_;
  set maxsales;
  max=max+5000;
  if max<5000 then max=5000;
  call symput('upper', left(put(round(max, 5000), 9.)));
run;

axis1 order=(0 to &upper by 5000);
```

5000 is added to the variable max so that when it is rounded to the nearest 5000, it will be a number greater than the original value of max. Similar techniques can be used to determine lower bounds and incremental values.

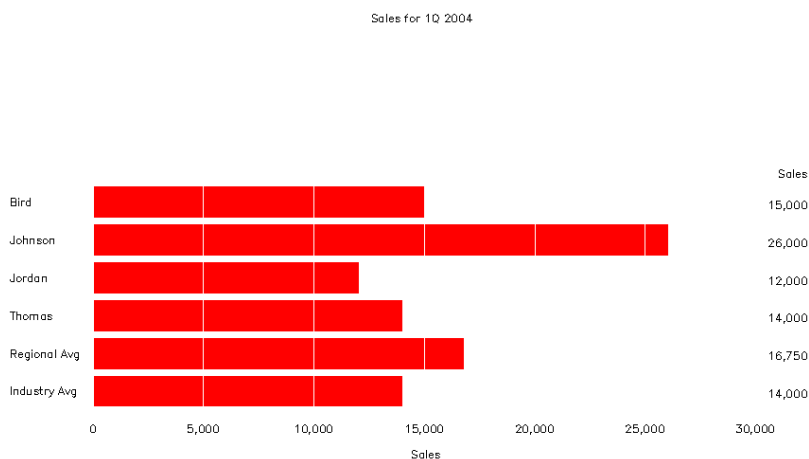
#12. Use spaces to order data alphabetically

There are times when the elements in a graph may need to appear in a certain order. Since spaces come before letters alphabetically, spaces can be used in front of the words that should appear first. Using the data in the previous example, suppose the data for each sales person needs to come first, followed by the regional average and the industry average.

The following code will add 2 spaces to the names of each of the salesmen and one space to the name when it equals "Regional Avg". In many cases, a length statement will need to be used when doing this.

```
* Use space to alphabetize;
data sales2;
  set sales;
  if index(name, 'Avg')=0 then name='  '||trim(name);
  if name='Regional Avg' then name=' '||trim(name);
run;
```

The names in the graph will now appear in the order shown below:



CONCLUSION

Graphs are an excellent way to display data. SAS programmers can use simple techniques to make the information contained in their graphs cleaner so managers can easily decipher the data.

REFERENCES

Tufte, Edward R., *The Visual Display of Quantitative Information*, Graphics Press, 1983

CONTACT INFORMATION

Aaron Winkel
 Physician Resource Northwest
 123 NE Third, #500
 Portland, OR 97232
 ph (503) 736-2303

awinkel@prnwest.com
<http://www.prnwest.com>