CHAPTER 2

Line Plots

Purpose: This chapter demonstrates how to create line plots, overlaid lines, shaded area under the line, forecast lines, step lines, and logarithmic line plots.

Basic Line Plot

A line plot is basically a scatter plot where the markers are connected. A line plot should only be used when the order of the markers is important – for example, when the data is plotted over time, or when the values of a function are being plotted.

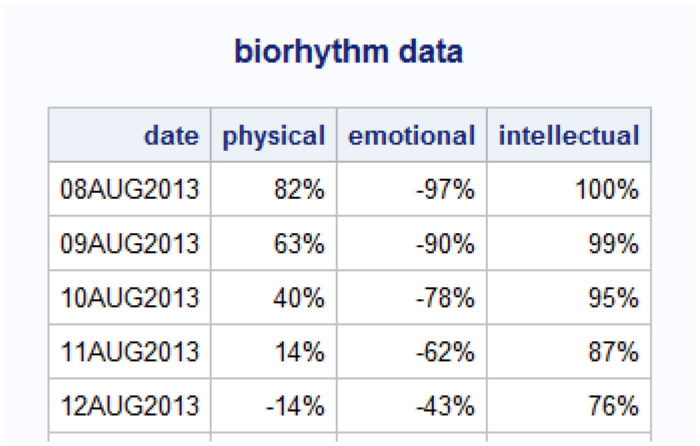
In this first example, we will be plotting the values of a function. You've probably heard of the biorhythm cycles (intellectual, physical, and emotional), where each cycle starts when you are born, and continues throughout your life. Each of the three cycles has a different number of days (23, 28, and 33), and therefore at any given point in your life you might be high or low (good or bad) in any of the three cycles. Many people like to know where they are in each cycle on a given day, and this example shows how to calculate that, and display the results with a line graph.

We'll use a data step with a loop to calculate the data values, and plot 4 weeks of data (28 days), starting on the current date (which was August 8, when I ran this code). I store the birthdate in a macro variable (BDATE), so I can easily change it in a single location, and use it throughout the SAS job. Note that the SAS sin() function expects the values to be in radians, therefore I create a D2R (degrees-to-radians) variable to help easily convert degrees to radians. This is all done in the small block of code below.

```
%let bdate=27jun1984;

data my_data;
format date date9.;
format physical emotional intellectual percentn7.0;
d2r=(atan(1)/45);
do date = today() to today()+27 by 1;
  t=date-"&bdate"d; /* t = days since birth */
  physical     = sin((t/23)*360*d2r);
  emotional    = sin((t/28)*360*d2r);
  intellectual = sin((t/33)*360*d2r);
  output;
  end;
run;
```

Here are the first few lines of data:



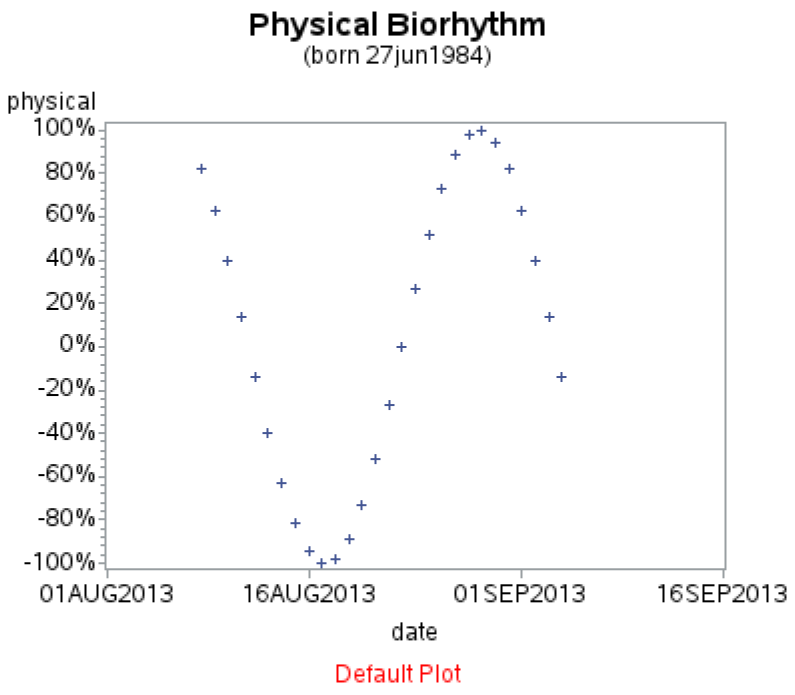
biorhythm data

date	physical	emotional	intellectual
08AUG2013	82%	-97%	100%
09AUG2013	63%	-90%	99%
10AUG2013	40%	-78%	95%
11AUG2013	14%	-62%	87%
12AUG2013	-14%	-43%	76%

Let's first work towards a basic line plot of just one of the three variables (physical). We'll start by creating just a simple scatter plot, taking all the defaults. This will allow us to get a feel for the data, and see what customizations we might want to add.

```
title1 ls=1.5 "Physical Biorhythm";
title2 "(born &bdate)";
proc gplot data=my_data;
plot physical*date;
run;
```

Even though we do not have a line yet, the scatter plot markers are lining up in the familiar shape of a sine wave, as expected, which indicates we are on the right track.



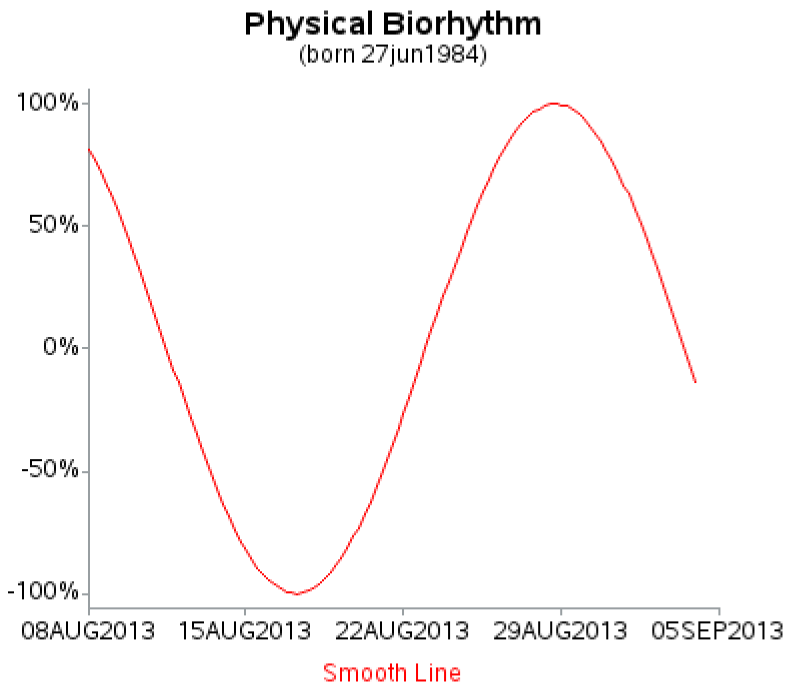
Let's add a SYMBOL statement to use a line rather than scatter markers. We could connect the points with straight line segments using INTERPOL=JOIN, but in this case we want a smooth curve therefore let's use INTERPOL=SM so a smooth line is fit to the data using a spline routine.

Also, the default auto-scaled date axis doesn't allow the data to utilize all the available space. It is difficult for software to always pick good date axis ranges and tick marks, because months have different number of days. In this case, Gplot starts the axis on August 1st and places a tick mark every two weeks for six weeks total. To utilize all the available space, we will start the axis on August 8, and extend through four weeks (28 days). We'll calculate the desired date values in a data step, save them into macro variables (DATE1 and DATE2), and then use them in an AXIS statement.

The code below plots one of the three cycles (physical). You could similarly plot the other two cycles, but we are going to take a slightly different approach … in the next example.

```
data _null_;
call symp用('date1',today());
call symput('date2',today()+28);
run;

title1 ls=1.5 "Physical Biorhythm";
title2 "(born &bdate)";
symbol1 value=none interpol=sm color=red;
axis1 label=none order=(-1 to 1 by .50) minor=none
 offset=(2,2);
axis2 label=none order=(&date1 to &date2 by 7) minor=none
 offset=(0,0);
proc gplot data=my_data;
plot physical*date=1 /
 vaxis=axis1 haxis=axis2 noframe;
run;
```



✓

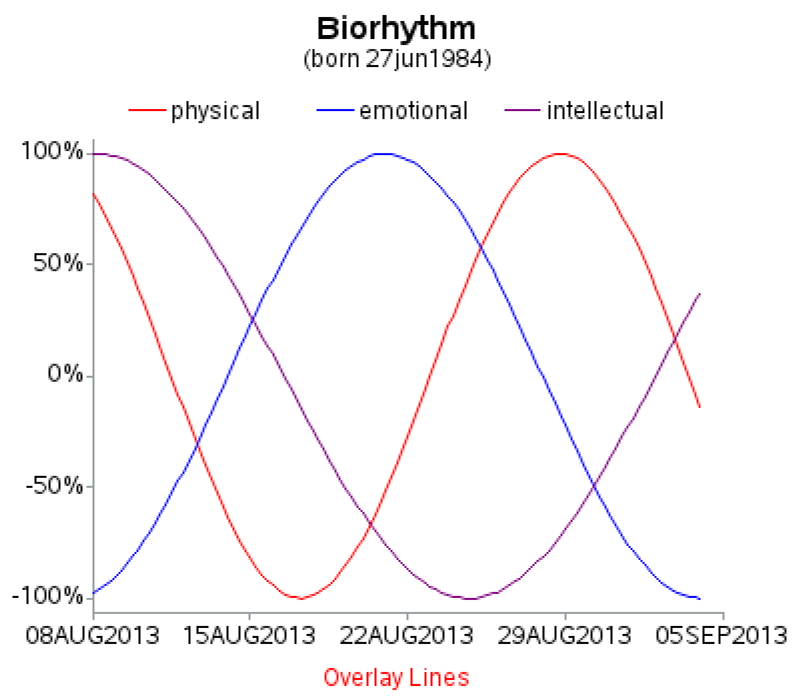
Let's Talk: Writing code to create graphs might seem like a lot of work, when there are so many tools that allow you to create graphs at the click of a button. But writing code allows you to easily create hundreds (or thousands) of graphs using a BY statement, or looping through the data. Also, writing code gives you the ability to customize your graphs in ways not provided by the point-and-click software.

Overlay Line Plot

We’ve got the plot for the physical biorhythm cycle, but what about the other lines for the emotional and intellectual cycles? Let’s add them to the same plot.

First, we’ll define SYMBOL statements for all three lines, specifying a different color for each of them (note that I RESET the symbol statements, in case you are running this code directly after running the previous example). Then plot the three Y*X pairs, and use the OVERLAY option to have them all appear on the same graph (otherwise they would appear on three separate graphs).

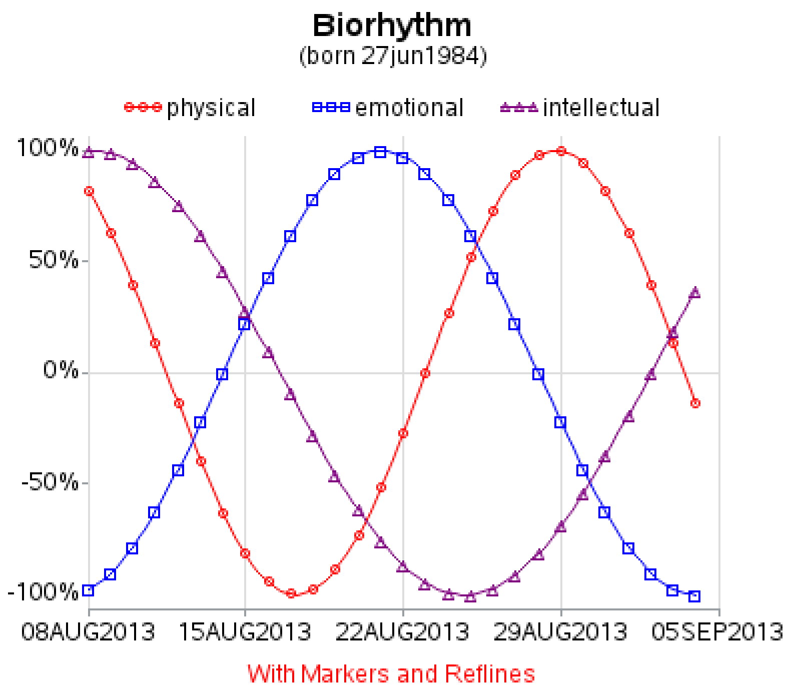
```
title1 ls=1.5 "Biorhythm";
title2 "(born &bdate)";
goptions reset=symbol;
symbol1 value=none interpol=sm color=red;
symbol2 value=none interpol=sm color=blue;
symbol3 value=none interpol=sm color=purple;
axis1 label=none order=(-1 to 1 by .50) minor=none
 offset=(2,2);
axis2 label=none order=(&date1 to &date2 by 7) minor=none
 offset=(0,0);
legend1 label=none position=(top center outside) across=3;
proc gplot data=my_data;
plot physical*date=1 emotional*date=2 intellectual*date=3 /
 overlay legend=legend1
 vaxis=axis1 haxis=axis2 noframe;
run;
```



The previous graph looks good in general, but it is a bit difficult to determine exactly what your biorhythm values are for a specific day. To help in that regard, let's add some reference lines and plot markers. The reference lines will help you easily see whether the biorhythm values are above or below zero, and where the weekly endpoints fall. The plot markers help you visually count the number of days before or after the weekly reference lines, as well as reinforce the association between the lines and the legend (you can identify the lines by both color and marker shape now).

Let's Talk: Opinions vary when it comes to using a plain line plot, or including markers on the line. I typically prefer having markers on the line. The markers let you easily tell what is the data, versus what is the interpolated (ie, "made up") part of the graph. The distance between the markers also lets you see how fast the slope of the line is changing, which is something that is difficult to do with just a line (especially if the line is at a steep angle). Also, if you are producing output for the web, the plot markers give you a place to put HTML hover-text. And, last but not least, if someone reading your plot is colorblind, the marker shapes allow them to distinguish the lines, and match them to the legend.

```
title1 ls=1.5 "Biorhythm";
title2 "(born &bdate)";
symbol1 value=circle height=2.5 interpol=sm color=red;
symbol2 value=square height=2.5 interpol=sm color=blue;
symbol3 value=triangle height=2.5 interpol=sm color=purple;
axis1 label=none order=(-1 to 1 by .50) minor=none
 offset=(2,2);
axis2 label=none order=(&date1 to &date2 by 7) minor=none
 offset=(0,0);
legend1 label=none position=(top center outside) across=3;
proc gplot data=my_data;
plot physical*date=1 emotional*date=2 intellectual*date=3 /
 overlay legend=legend1
 vaxis=axis1 haxis=axis2 noframe
 vref=0 cvref=graydd
 autohref chref=graydd;
run;
```



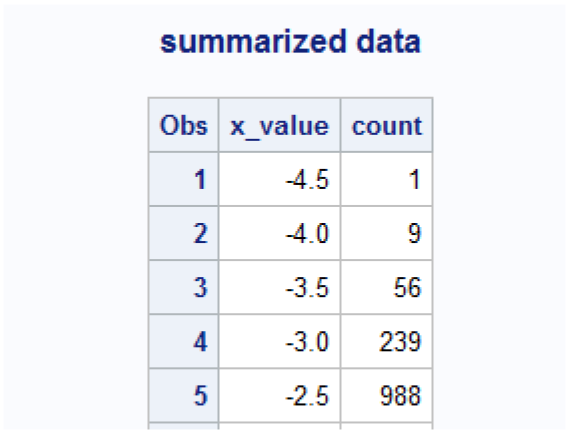
Area Under Line

One variation of a line plot is to shade the area under the line. This type of plot is typically used by scientists, engineers, and statisticians (and not so much by people creating business graphs).

Let's demonstrate with a statistical example – showing a normal distribution curve. SAS has special functions for generating random numbers with various distributions. The RANNOR() function generates random numbers with a normal distribution. We'll use a data step to go through a loop and generate 100,000 random numbers, and then round the numbers to the nearest .5 and do a frequency count. Your actual data could be from a function, experimental results, and so on.

```
data raw_data;
do n = 1 to 100000;
 x_value=round(rannor(1),.5);
 output;
 end;
run;
proc sql;
 create table my_data as
 select unique x_value, count(*) as count
 from raw_data group by x_value;
quit; run;
```

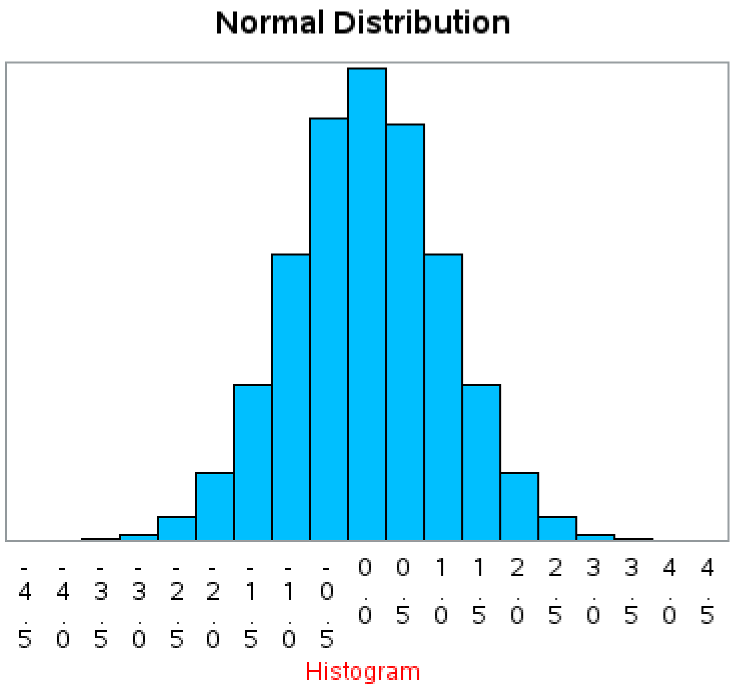
Here are the first few observations of the summarized data.



summarized data

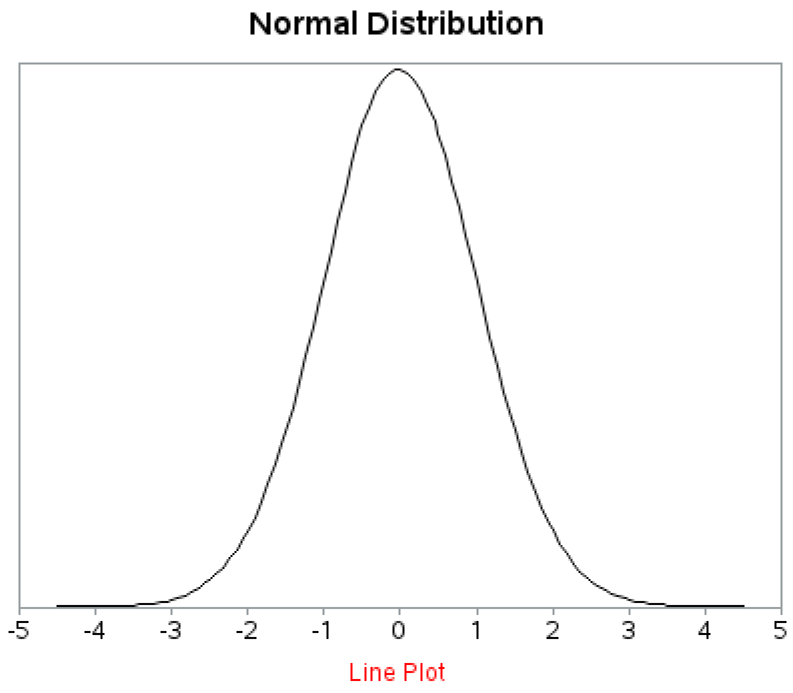
Obs	x_value	count
1	-4.5	1
2	-4.0	9
3	-3.5	56
4	-3.0	239
5	-2.5	988

One easy way to display such data is to use a bar chart and create a histogram. But PROC GCHART labels each bar, and at some point you will have too many bars to fit all of the bar labels in a graceful manner. Here is a basic histogram of this data, for example.



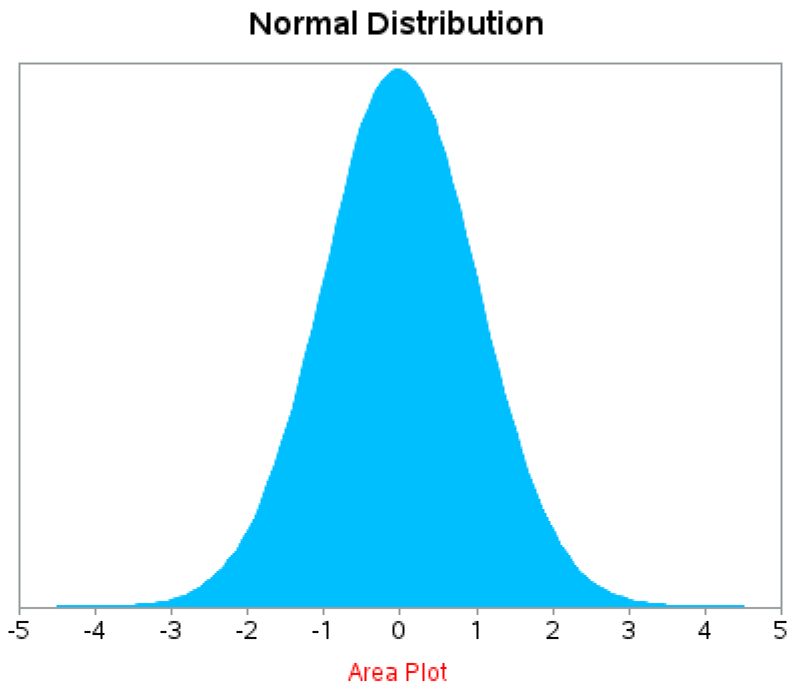
By comparison, if you use a line plot, you can have a continuous axis scale (rather than discrete), and therefore you do not need a label for each data point. Here is an example of a simple line plot of the data. Note that I add a slight OFFSET so that the line does not blend in with the horizontal axis.

```
title1 ls=1.5 "Normal Distribution";
symbol1 interpol=sm color=black value=none width=1;
axis1 minor=none label=none order=(-5 to 5 by 1)
 offset=(0,0);
axis2 minor=none label=none major=none value=none
 offset=(.1,0);
proc gplot data=my_data;
plot count*x_value /
 haxis=axis1 vaxis=axis2;
run;
```



But the simple line does not quite have the same impact as the colored bars. Therefore let's color the area under the line using the AREAS=1 option. And rather than taking the default color, we'll control the color of the area under the line using a PATTERN statement.

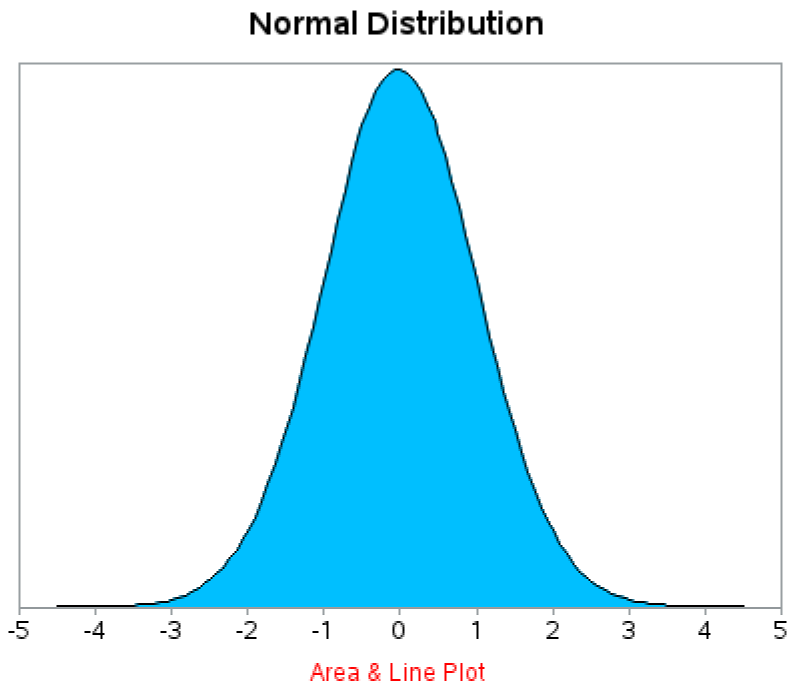
```
title1 ls=1.5 "Normal Distribution";
pattern1 v=s c=cx00BFFF;
symbol1 interpol=sm color=black value=none width=1;
axis1 minor=none label=none order=(-5 to 5 by 1)
 offset=(0,0);
axis2 minor=none label=none major=none value=none
 offset=(.1,0);
proc gplot data=my_data;
plot count*x_value / areas=1
 haxis=axis1 vaxis=axis2;
run;
```



✓

That was a fine area plot, but there is one more finishing-touch I like to add. Similar to the outlines around the bars in a bar chart, I think it helps visually define the area to draw a line along the edge. There's no built-in feature, but we can overlay a line plot on top of the area plot to produce this effect.

```
title1 ls=1.5 "Normal Distribution";
pattern1 v=s c=cx00BFFF;
symbol1 interpol=sm color=black value=none width=1;
axis1 minor=none label=none order=(-5 to 5 by 1)
 offset=(0,0);
axis2 minor=none label=none major=none value=none
 offset=(.1,0);
proc gplot data=my_data;
plot count*x_value count*x_value=1 / overlay areas=1
 haxis=axis1 vaxis=axis2;
run;
```


Normal Distribution
-5
-4
-3
-2
-1
0
1
2
3
4
5
Area & Line Plot

✓+

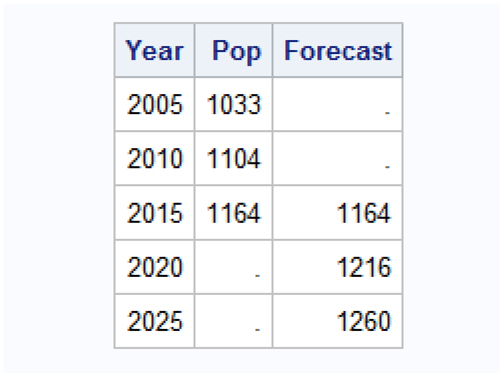
Forecast Line

Many people use SAS for analytics, and especially forecasting. And what better way to *see* a forecast than to plot it?

The following data will be used for the next few examples. It contains several variables for a forecast, and the upper & lower confidence interval – each example shows how to incorporate a few more of the variables.

```
data my_data;
input Year Pop Forecast Upper95 Lower95 Upper80 Lower80;
datalines;
1960 573  . . . . .
1965 581  . . . . .
1970 614  . . . . .
1975 650  . . . . .
1980 686  . . . . .
1985 704  . . . . .
1990 767  . . . . .
1995 855  . . . . .
2000 943  . . . . .
2005 1033 . . . . .
2010 1104 . . . . .
2015 1164 1164 1178 1151 1174 1155
2020  .   1216 1249 1185 1239 1195
2025  .   1260 1312 1203 1295 1222
2030  .   1298 1370 1213 1348 1240
2035  .   1325 1417 1217 1390 1247
2040  .   1340 1455 1202 1420 1243
2045  .   1347 1489 1182 1443 1230
2050  .   1345 1517 1147 1464 1204
2055  .   1339 1543 1105 1481 1173
2060  .   1330 1566 1059 1491 1134
2070  .   1296 1607 953 1499 1047
2075  .   1275 1625 896 1505 1005
2080  .   1256 1646 841 1517 959
;
run;
```

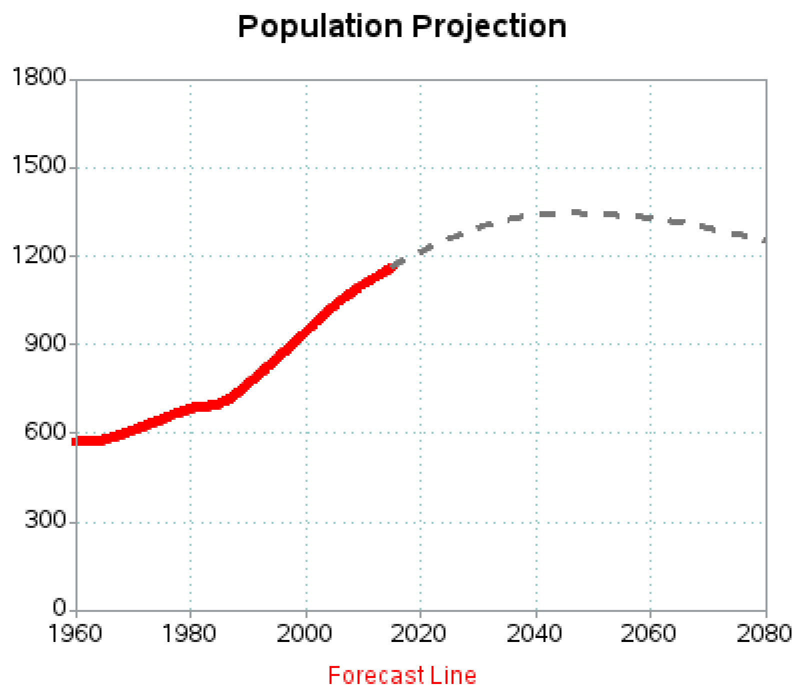
For the forecast line plot, we'll only be using the first three variables: Year, Pop, and Forecast. Here are a few rows of the data we'll be plotting.



Year	Pop	Forecast
2005	1033	.
2010	1104	.
2015	1164	1164
2020	.	1216
2025	.	1260

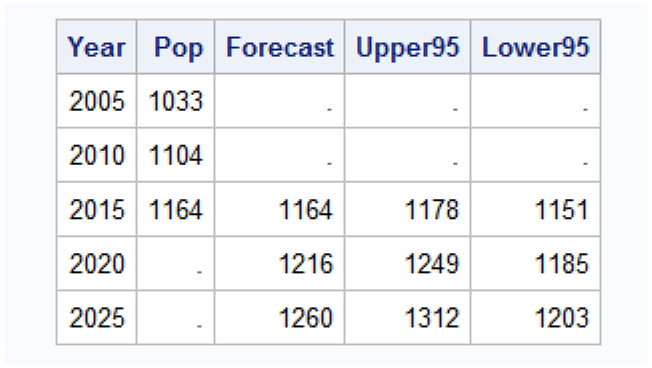
Similar to the previous overlay plot examples, we'll plot both the Pop and Forecast against year, and then use the OVERLAY option to have them both appear on the same plot. Note we have population values up until year 2015, and then we have Forecast values after that. When creating this type of plot, I like to have one year of overlap (with both Pop and Forecast) so there is no gap in the line. I use a red line for the past population, and a dashed gray line for the forecast (the LINE=33 controls the way the dashes look).

```
title1 ls=1.5 "Population Projection";
symbol1 value=none interpol=sm width=4 color=red;
symbol2 value=none interpol=sm width=2 line=33 color=gray77;
axis1 label=none order=(0 to 1800 by 300) minor=none
 offset=(0,0);
axis2 label=none order=(1960 to 2080 by 20) minor=none
 offset=(0,0);
proc gplot data=my_data;
plot pop*year=1 forecast*year=2 / overlay
 vaxis=axis1 haxis=axis2
 autohref lhref=33 chref=cx99cccc
 autovref lvref=33 cvref=cx99cccc;
run;
```



Confidence Bands

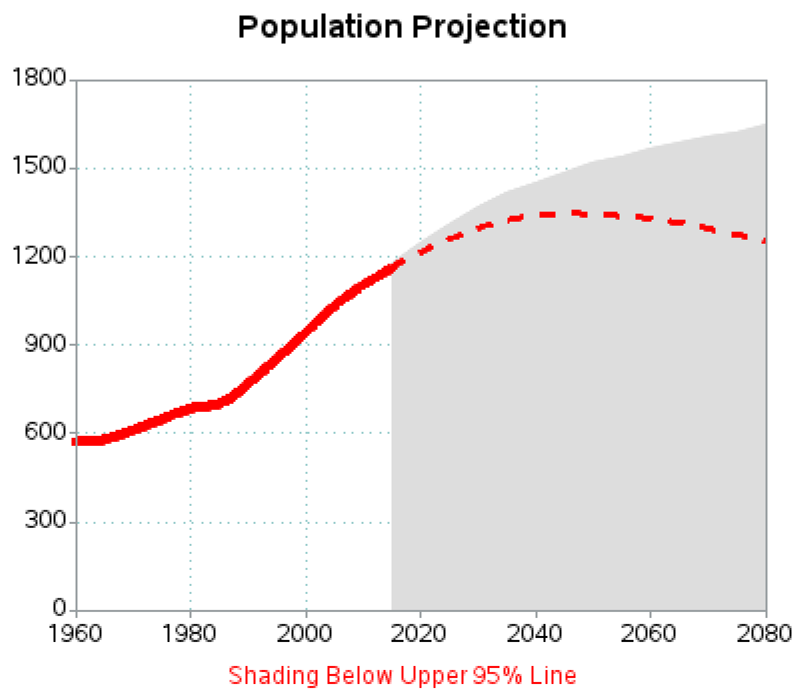
Usually with a forecast, you also have confidence band data. As the forecast gets farther into the future, the confidence band gets wider (in other words, there is more uncertainty). It is often useful to show the edges of the confidence interval with lines, or to color the area (or band) between the lines. Our data has UPPER95 and LOWER95 variables that we can use for this purpose.



Year	Pop	Forecast	Upper95	Lower95
2005	1033	.	.	.
2010	1104	.	.	.
2015	1164	1164	1178	1151
2020	.	1216	1249	1185
2025	.	1260	1312	1203

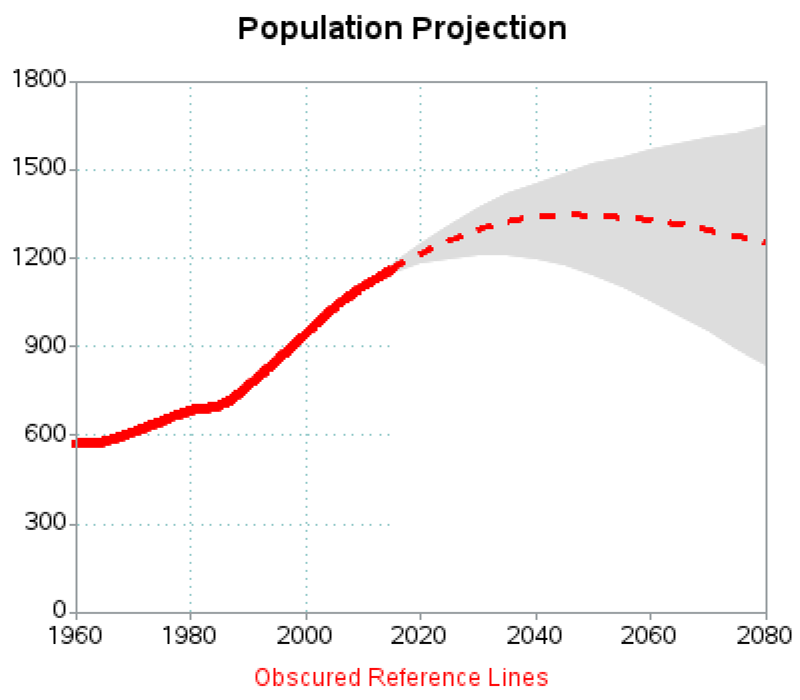
First, using the techniques you previously learned, let’s shade the area below the UPPER95 line.

```
title1 ls=1.5 "Population Projection";
pattern1 v=s c=graydd;
symbol1 value=none interpol=sm width=4 color=red;
symbol2 value=none interpol=sm width=2 line=33 color=red;
symbol3 value=none interpol=join color=pink;
axis1 label=none order=(0 to 1800 by 300) minor=none
 offset=(0,0);
axis2 label=none order=(1960 to 2080 by 20) minor=none
 offset=(0,0);
proc gplot data=my_data;
plot Upper95*year=3
 Forecast*year=2 Pop*year=1 / overlay areas=1
 vaxis=axis1 haxis=axis2
 autohref lhref=33 chref=cx99cccc
 autovref lvref=33 cvref=cx99cccc;
run;
```



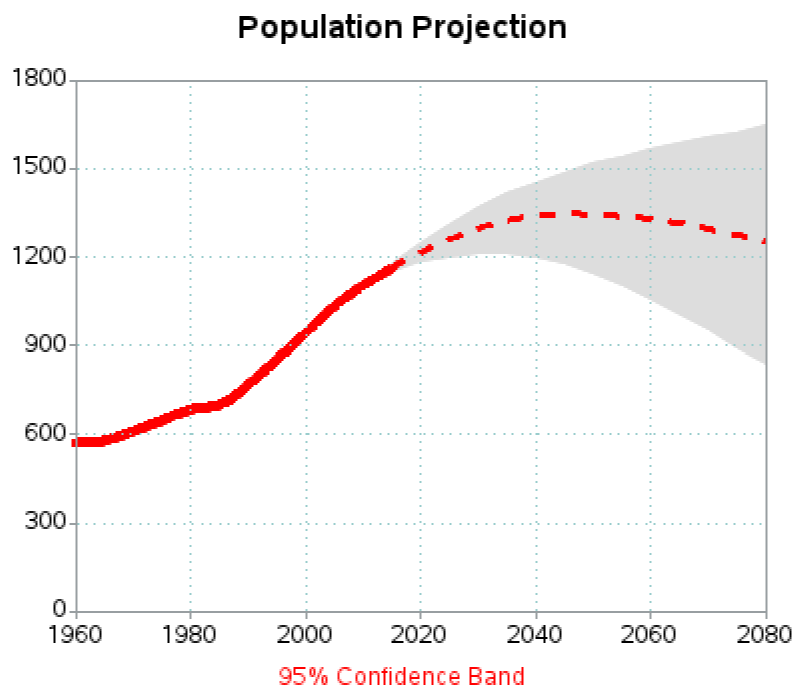
But this isn't quite what we want. We only want the confidence band shaded, not all the way to the bottom of the graph. The trick you can use to accomplish this is to add another shaded area, to cover the area from the lower edge of the confidence band to the bottom of the graph with white color (making it look blank).

```
title1 ls=1.5 "Population Projection";
pattern1 v=s c=white;
pattern2 v=s c=graydd;
symbol1 value=none interpol=sm width=4 color=red;
symbol2 value=none interpol=sm width=2 line=33 color=red;
symbol3 value=none interpol=join color=pink;
axis1 label=none order=(0 to 1800 by 300) minor=none
 offset=(0,0);
axis2 label=none order=(1960 to 2080 by 20) minor=none
 offset=(0,0);
proc gplot data=my_data;
plot Lower95*year=3 Upper95*year=3
 Forecast*year=2 Pop*year=1 / overlay areas=2
 vaxis=axis1 haxis=axis2
 autohref lhref=33 chref=cx99cccc
 autovref lvref=33 cvref=cx99cccc;
run;
```

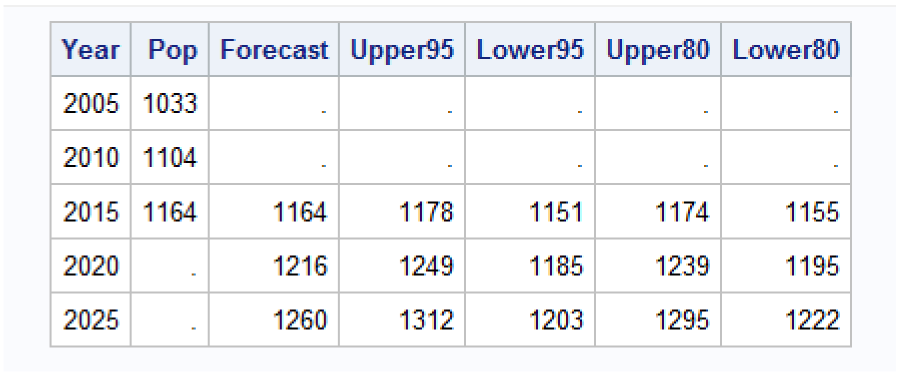


As is often the case, this is **almost** what we want … but not quite. Notice that the white area below the lower edge of the confidence band now obscures the reference lines. Let's add the FRONTREF option to have the reference lines drawn on top of the shaded areas.

```
title1 ls=1.5 "Population Projection";
pattern1 v=s c=white;
pattern2 v=s c=graydd;
symbol1 value=none interpol=sm width=4 color=red;
symbol2 value=none interpol=sm width=2 line=33 color=red;
symbol3 value=none interpol=join color=pink;
axis1 label=none order=(0 to 1800 by 300) minor=none
 offset=(0,0);
axis2 label=none order=(1960 to 2080 by 20) minor=none
 offset=(0,0);
proc gplot data=my_data;
plot Lower95*year=3 Upper95*year=3
 Forecast*year=2 Pop*year=1 / overlay areas=2
 vaxis=axis1 haxis=axis2
 autohref lhref=33 chref=cx99cccc
 autovref lvref=33 cvref=cx99cccc
 frontref;
run;
```



That's a pretty nice plot! … But what if we want to also show the 80% confidence band (between the Upper80 and Lower80 variables) on the same plot in a lighter gray?



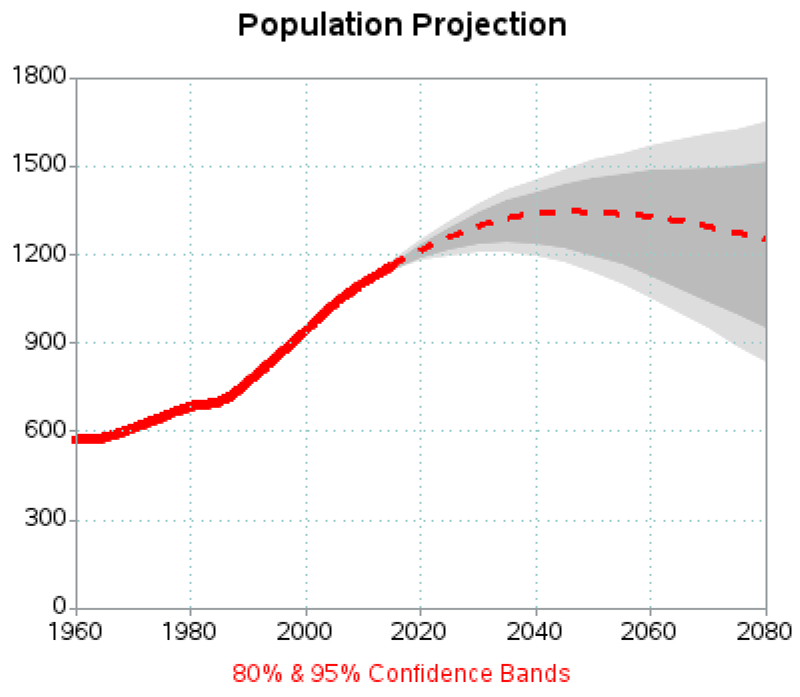
Year	Pop	Forecast	Upper95	Lower95	Upper80	Lower80
2005	1033	.	.	.	.	.
2010	1104	.	.	.	.	.
2015	1164	1164	1178	1151	1174	1155
2020	.	1216	1249	1185	1239	1195
2025	.	1260	1312	1203	1295	1222

You can add the extra band by overlaying even more areas (AREAS=4), starting at the top, and working your way down. In the PATTERN statements, the GRAYDD is the lighter gray, and the GRAYBB is the darker gray.

Let's Talk: Overlaying multiple confidence bands can be a little confusing, but just build it up slowly one area at a time. Remember that the order in which you overlay the areas and lines is important. And it is often helpful to change the color of each area while you are first creating the graph, so you know exactly where each band is showing up in your stacked areas.

```
title1 ls=1.5 "Population Projection";
pattern1 v=s c=white;
pattern2 v=s c=graydd;
pattern3 v=s c=graybb;
pattern4 v=s c=graydd;
symbol1 value=none interpol=sm width=4 color=red;
symbol2 value=none interpol=sm width=2 line=33 color=red;
symbol3 value=none interpol=join color=pink;
axis1 label=none order=(0 to 1800 by 300) minor=none
 offset=(0,0);
axis2 label=none order=(1960 to 2080 by 20) minor=none
 offset=(0,0);
proc gplot data=my_data;
plot Lower95*year=3 Lower80*year=3
 Upper80*year=3 Upper95*year=3
 Forecast*year=2 Pop*year=1 / overlay areas=4
 vaxis=axis1 haxis=axis2
 autohref lhref=33 chref=cx99cccc
 autovref lvref=33 cvref=cx99cccc frontref;
run;
```

The results are worth all the hard work – we now have a really sharp looking forecast plot, with multiple confidence bands.



Step Line

There are many ways to connect the points in a plot using a line or a curve, and it is important to know when to use (and not use) which technique. For example, if the data points are snapshots of a continuous variable (such as temperature), or represent a sum or average for a certain time period (such as monthly sales, or annual population), then the line interpolation shows an estimate of the values *between* the points being plotted.

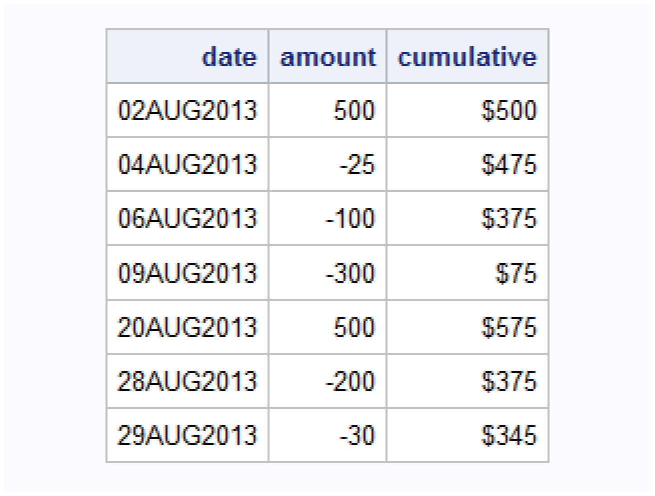
But, when you are plotting transactional data (such as individual sales, deposits, payments, and so on), then connecting the data points with a straight line or curve could be misleading. When plotting individual points of transaction data, it is usually best to use a step line.

For this example we will use a checking account. You make deposits into a checking account (positive numbers), and then you write checks (negative numbers). Here is the data we'll be plotting:

```
data my_data;
input date date9. amount;
datalines;
02aug2013 500
04aug2013 -25
06aug2013 -100
09aug2013 -300
20aug2013 500
28aug2013 -200
29aug2013 -30
;
run;
```

To prepare the data to plot, we'll use a data step to calculate a running total and store it in a variable called CUMULATIVE – take note of the syntax, as it is very useful (and something you probably would not have guessed). The data step is also a convenient place to add formats to display the values on the axes as date and the dollar amounts.

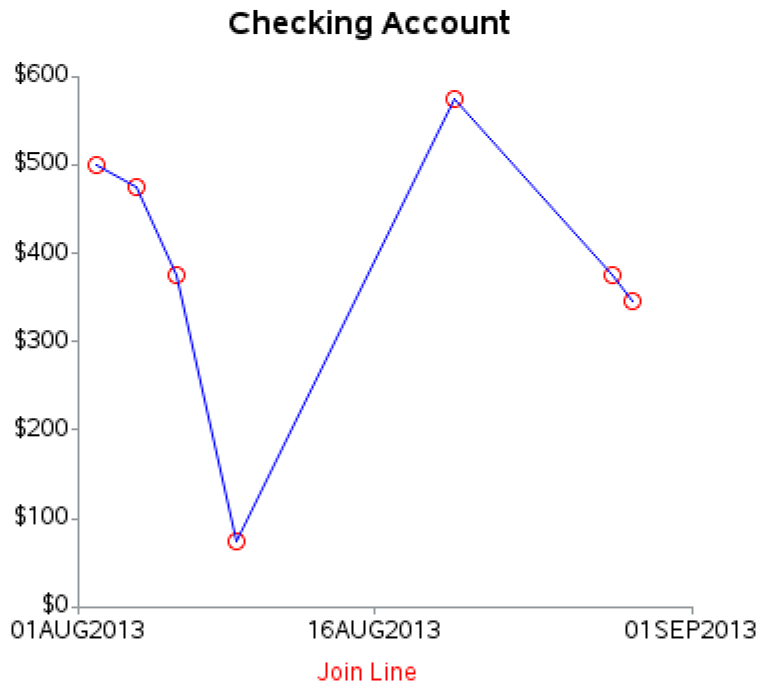
```
data my_data; set my_data;
format date date9.;
format cumulative dollar10.0;
cumulative+amount;
run;
```



date	amount	cumulative
02AUG2013	500	$500
04AUG2013	-25	$475
06AUG2013	-100	$375
09AUG2013	-300	$75
20AUG2013	500	$575
28AUG2013	-200	$375
29AUG2013	-30	$345

First, let's create a bad/misleading plot, connecting the cumulative data points with a straight line.

```
title1 ls=1.5 "Checking Account";
symbol1 value=circle height=5 cv=red interpol=join ci=blue;
axis1 label=none minor=none offset=(0,0);
axis2 label=none minor=none offset=(0,0);
proc gplot data=my_data;
plot cumulative*date /
 vaxis=axis1 haxis=axis2
 vzero noframe;
run;
```



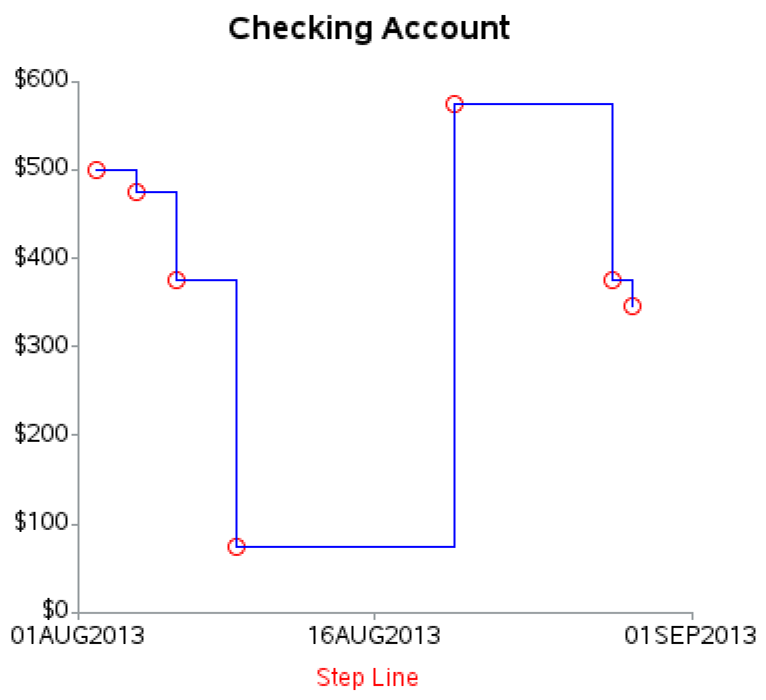
✗

How, you might ask, is this plot misleading? Looking at the diagonal line, you would assume that on August 16, this checking account had about $300 – at least, that's what the line indicates. But the actual cumulative values in the table tell a different story. The checking account had $75 on August 9, and stayed that way until a $500 deposit was made on August 20. The account had nowhere near $300 on August 16!

Let's Talk: You've probably heard the phrase "first, do no harm" in association with doctors. Well, I like to apply that phrase to graphs as well. In my opinion, it is better to have no graph at all, than to have a misleading graph.

When it comes to plotting individual transactions, the line needs to stay at the level of the previous value (ie, stay horizontal), until the next transaction is made. That's where the STEP interpolation comes is useful. INTERPOL=STEPLJ causes the line to stay at the level of the previous marker, until it reaches the next marker. Using this technique, there are no misleading diagonal lines that might be misinterpreted as approximations of the value at dates between the markers.

```
title1 ls=1.5 "Checking Account";
symbol1 value=circle height=5 cv=red interpol=steplj
 ci=blue;
axis1 label=none minor=none offset=(0,0);
axis2 label=none minor=none offset=(0,0);
proc gplot data=my_data;
plot cumulative*date /
 vaxis=axis1 haxis=axis2
 vzero noframe;
run;
```



✓

Logarithmic Axes

Sometimes when you are plotting certain types of data, it might be desirable for one (or both) of your axes to use a logarithmic scale. You can produce this type of axis using a few simple options in the AXIS statement.

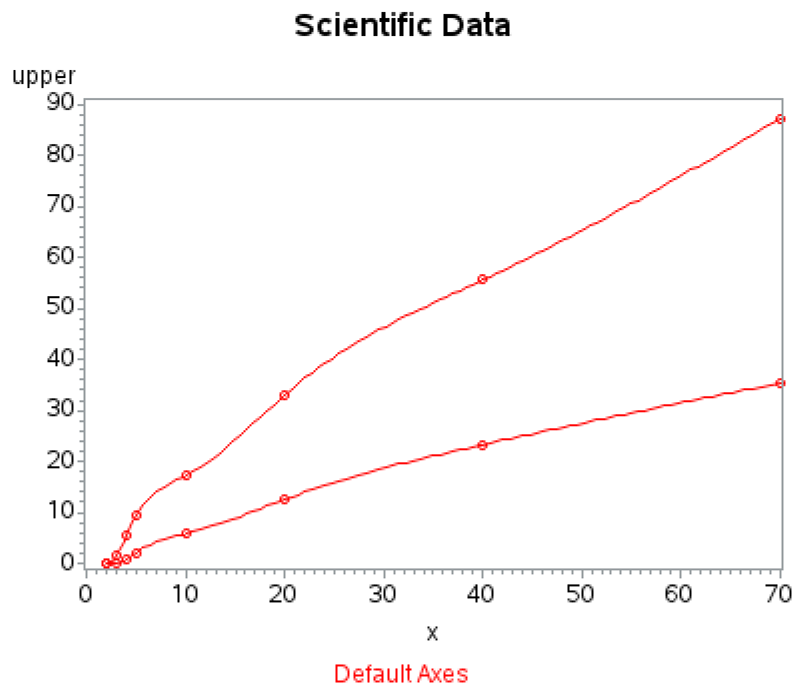
We'll use some fairly simple 'fabricated' (made up) scientific data in this example, showing both a lower and upper line. Here is the code that creates the sample data:

```
data my_data;
input x lower upper;
datalines;
2    0.01  0.01
3    0.14  1.65
4    0.70  5.70
5    2.10  9.51
10   5.80 17.21
20  12.60 33.18
40  23.30 55.60
70  35.40 87.10
;
run;
```

x	lower	upper
2	0.01	0.01
3	0.14	1.65
4	0.70	5.70
5	2.10	9.51
10	5.80	17.21
20	12.60	33.18
40	23.30	55.60
70	35.40	87.10

First, let's create a simple plot and overlay the two lines, using regular (non-logarithmic) axes. Notice that the smaller values are clustered together, and the larger values are spread out.

```
title1 ls=1.5 "Scientific Data";
symbol1 v=circle h=2 i=sm c=red;
proc gplot data=my_data;
plot upper*x=1 lower*x=1 / overlay;
run;
```

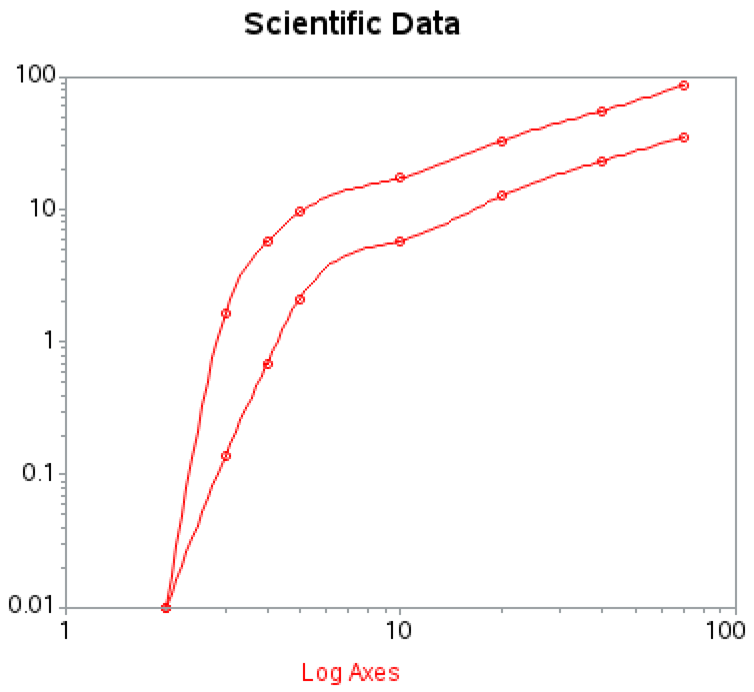


Now, let's plot the same data using logarithmic scales on the axes. The LOGBASE and LOGSTYLE options tell Gchart to use a logarithmic scale. There are several options to control the exact kind of logarithmic scale to use, and I almost always use the options shown in this example. I also typically use the ORDER option to control what range to show in the plot. Note that use the BEST12 format for the variable plotted on the vertical axis allows each tick mark to be independently formatted in the *best* way for that number – by comparison, if we had used a format such as COMMA8.2, then all the tick values would have shown two decimal places.

Let's Talk: When it comes to creating plots with logarithmic axes, I do not recommend using trial-and-error to select the LOGBASE, LOGSTYLE, and ORDER axis options. You need to know enough about your data, and enough about the chart you are trying to produce, that you know what options to use. Sit down ahead of time, and think it through.

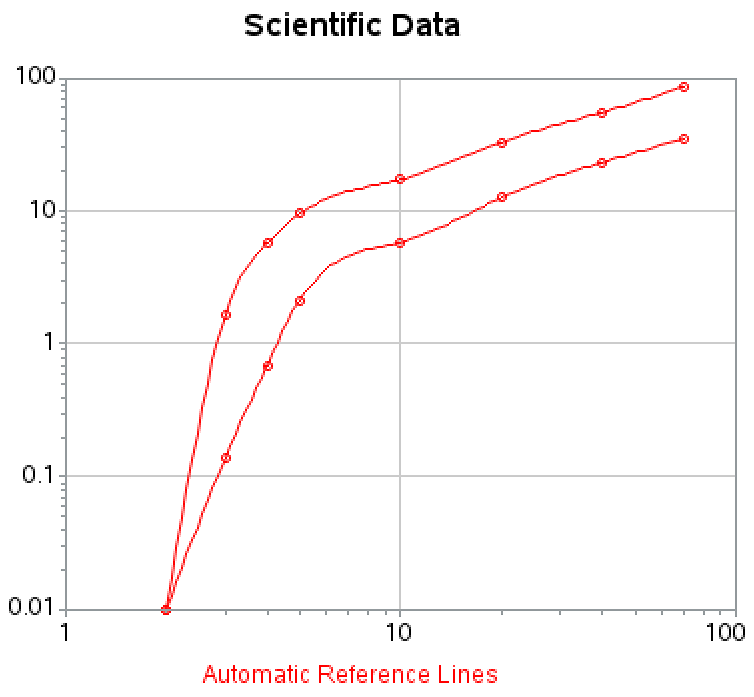
```
title1 ls=1.5 "Scientific Data";
symbol1 v=circle h=2 i=sm c=red;
axis1 logbase=10 logstyle=expand  order=(.01 .1 1 10 100)
 offset=(0,0) label=none;
axis2 logbase=10 logstyle=expand  order=(1 10 100)
 offset=(0,0) label=none;
proc gplot data=my_data;
format upper best12.;
plot upper*x=1 lower*x=1 / overlay
 vaxis=axis1 haxis=axis2;
run;
```

Notice that the markers for the small and large values are now spread across the graph, rather than the small values being crowded together in the corner. I almost always leave the minor tick marks (in other words, I do not use MINOR=NONE) in a plot with logarithmic axes. The minor tick marks help give the reader a visual indication that logarithmic axes are being used.



It is often useful to show light gray grid lines in scientific plots, and we can easily turn on grid lines at the major tick marks using the AUTOVREF and AUTOHREF options.

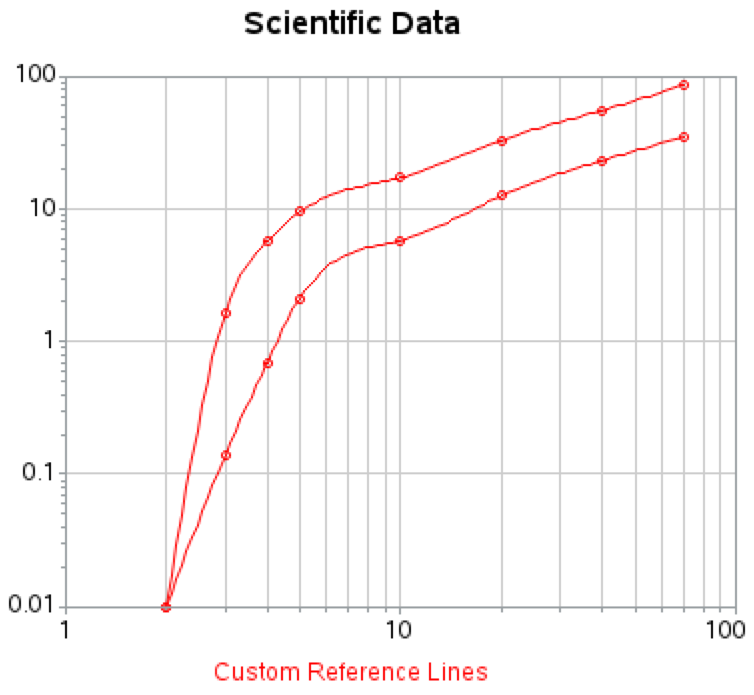
```
title1 ls=1.5 "Scientific Data";
symbol1 v=circle h=2 i=sm c=red;
axis1 logbase=10 logstyle=expand  order=(.01 .1 1 10 100)
 offset=(0,0) label=none;
axis2 logbase=10 logstyle=expand  order=(1 10 100)
 offset=(0,0) label=none;
proc gplot data=my_data;
format upper best12.;
plot upper*x=1 lower*x=1 / overlay
 vaxis=axis1 haxis=axis2
 autovref cvref=graycc
 autohref chref=graycc;
run;
```



In addition to the reference lines at the major axis tick marks, in certain situations I also add reference lines at the minor tick marks. For example, if we add them on the horizontal axis of this plot, then the person reading the graph can easily see that the red plot markers are actually at the tick marks. There is no option to automatically add these minor tick mark reference lines, therefore we have to specify each of them manually for this special graph.

```
title1 ls=1.5 "Scientific Data";
symbol1 v=circle h=2 i=sm c=red;
axis1 logbase=10 logstyle=expand  order=(.01 .1 1 10 100)
 offset=(0,0) label=none;
axis2 logbase=10 logstyle=expand  order=(1 10 100)
 offset=(0,0) label=none;
proc gplot data=my_data;
format upper best12.;
plot upper*x=1 lower*x=1 / overlay
 vaxis=axis1 haxis=axis2
 autovref cvref=graycc
 href=
  1 2 3 4 5 6 7 8 9
  10 20 30 40 50 60 70 80 90 100
  chref=graycc;
run;
```

This produces the following spiffy graph, ready to include in a scientific journal!


Scientific Data
100
10
1
0.1
0.01
1
10
100
Custom Reference Lines