Intermediate SAS® ODS Graphics

Chuck Kincaid, Experis Business Analytics, Kalamazoo, MI

ABSTRACT

This paper will build on the knowledge gained in the Intro to SAS® ODS Graphics. The capabilities in ODS Graphics grow with every release as both new paradigms and smaller tweaks are introduced. After talking with the ODS developers, a selection of the many wonderful capabilities was selected. This paper will look at that selection of both types of capabilities and provide the reader with more tools for their belt.

Visualization of data is an important part of telling the story seen in the data. And while the standards and defaults in ODS Graphics are very well done, sometimes the user has specific nuances for characters in the story or additional plot lines they want to incorporate. Almost any possibility, from drama to comedy to mystery, is available in ODS Graphics if you know how. We will explore tables, annotation and changing attributes, as well as the BLOCK plot.

Any user of Base SAS on any platform will find great value from the SAS ODS Graphics procedures. Some experience with these procedures is assumed, but not required.

INTRODUCTION

In the previous paper, the basics of SAS ODS Graphics were presented and with these the user can accomplish most of their tasks. However, we often need to do more than the basic graphs. The questions asked by the business and the data we work with to answer them are complex. It takes a complex system to tell this story in ways that are easily and accurately understood. Not coincidently, SAS ODS Graphics are such a system. The good thing is that it is also an immensely simple system to use, as shown in the previous paper. That simplicity allows us to quickly get into the more advanced capabilities so that we can tell the story even better.

This paper will introduce the reader to those more advanced capabilities. Users often want to show the data alongside of the graphs or provide additional information with relevant data not currently visualized, so we will explore how to add tables of information to the graph. Annotation, familiar to those who used it in SAS/Graph, allows the user to add additional, often less structured, information to the graph. The user has the amazing, almost dumbfounding, power to customize the attributes of every graph component if they so choose. This allows them endless possibilities of comparing and contrast, highlighting and subtlety, asceticism and style. We will introduce how to control these style attributes inline and with attribute maps. Finally, a new types of plot, BLOCK, will be introduced that can be paired with the standard plots or even used on its own.

TABLES

Graphics are a brilliant way to help someone easily comprehend complex information. Comparing and contrasting, for example, is done better with bar charts than with tables of information. However, often both the big picture and the details are needed and it is better to present tables with the graph, such as the one in the graphic below. Creating these used to be possible only with the SCATTER statement. Let's use some data about villains on the British TV series Doctor Who to see how. The show has been running since 1963 except for a short break, so there are a lot of villains. We've subset the data to include only the top 14 villains according to number of episodes. The code below creates a scatter plot of the number of episodes for each of the villains.

```
proc sgplot data=who14;
    scatter x=villain y=episodes;
run;
```
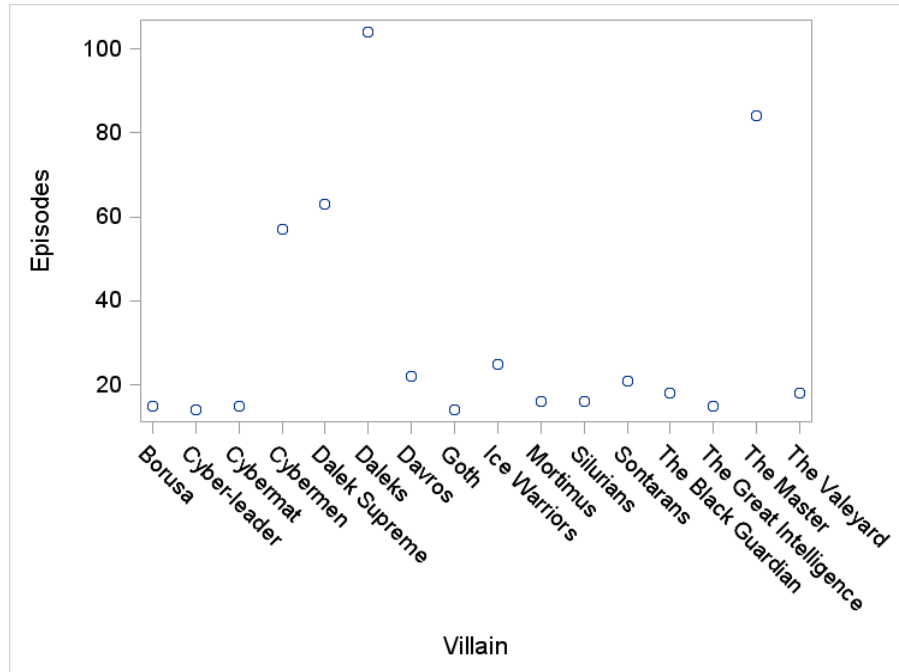
Figure 1

Suppose we would also like to show the year that the villain first appeared on the show. A table of the year will be created within the wall area (i.e. within the axes) of the graph.

OLD SCHOOL

First, we'll do this the "old" way, which can still be useful. We start by adding a new variable, X1, to the data set whose value is "First Aired."

```
data who14;
    set who14;
    x1 = "First Aired";
run;
```

This variable will serve as both the label and the location on the Y axis for the table. SGPLOT has no problem combining a quantitative variable, EPISODES, with a qualitative variable, X1, on the same plot as long as we put them on different axes. We'll do this by using the Y2AXIS for our table.

Now we add a SCATTER plot with VILLAIN on the x-axis just like the other plot and X1 on the y-axis. We get the information we want on there by telling SAS that the MARKERCHAR should be YEAR_FIRST. Notice that we specify the Y2AXIS.

```
proc sgplot data=who14 noautolegend;
    scatter x=villain y=episodes;
    scatter x=villain y=x1 / markerchar=year_first y2axis;
    yaxis offsetmin=0.1;
    y2axis display=(nolabel) offsetmax=0.9;
run;
```

2

We also add an offset to both the YAXIS and the Y2AXIS so that the numbers don't overlap with the plotted points. OFFSETMIN=0.1 moves the bottom edge of the y-axis up by ten percent of the range. OFFSETMAX moves the top edge of the y2-axis down by 90%. That way they do not overlap. We also tell the Y2AXIS to not display the label. If it did, then we'd see a weird "x1" on the right edge of the graph.
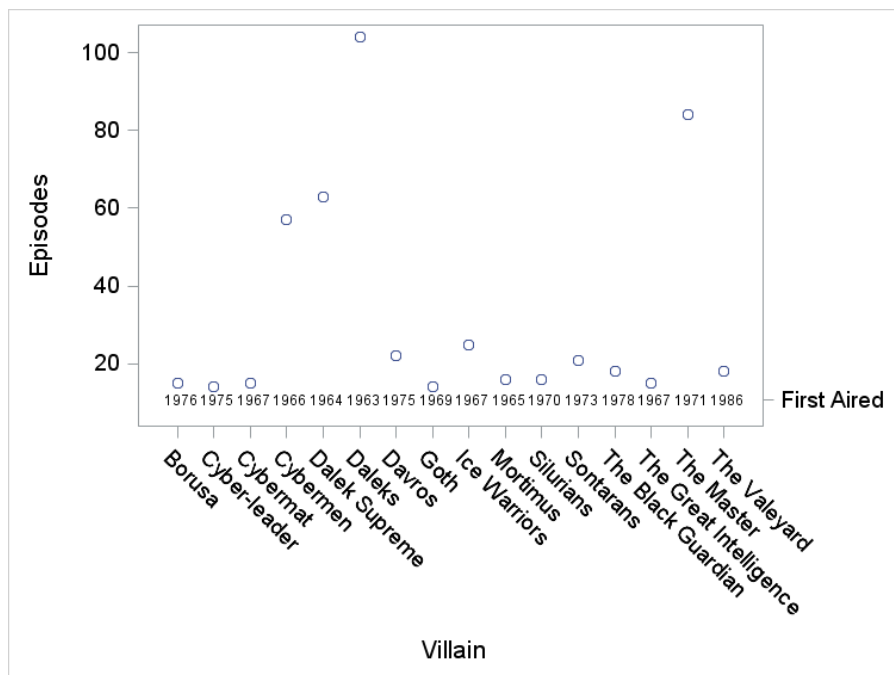

Figure 2

We can have as many rows as we want in this table by adding more variables and moving the offsets. For example, if we wanted to show the year the villains first aired, the year they last aired and the total number of story lines they showed up in, then our code could look like this.

```
data who14;
    set who14;
    x1 = "First Aired";
    x2 = "Total Storylines";
    x3 = "Last Aired";
run;

proc sgplot data=who14 noautolegend;
    scatter x=villain y=episodes;
    scatter x=villain y=x1 / markerchar=year_first y2axis;
    scatter x=villain y=x2 / markerchar=total_stories y2axis;
    scatter x=villain y=x3 / markerchar=year_last y2axis;
    yaxis offsetmin=0.3;
    y2axis display=(nolabel) offsetmin=0.1 offsetmax=0.8;
run;
```
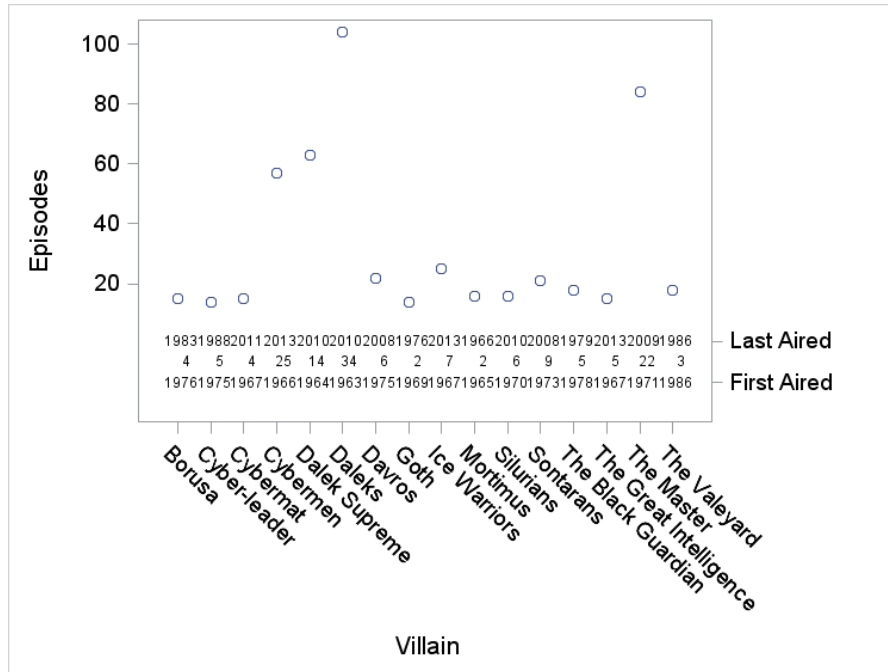
Figure 3

Here we've added X2 and X3 with appropriate values and appropriate MARKERCHARs. We also added another offset control so that the rows would be closer to each other and the "table" would look better. Try it without the new OFFSETMIN=0.1 to see what we mean.

NEW SCHOOL

While this works, and is the only option prior to release 9.4, it takes more work and can become cumbersome. Because the developers are all about flexibility and ease of use, and because quite a few users asked for the capability, they added XAXISTABLE and YAXISTABLE. These options not only do what we just did a bit easier, but they give us greater control and flexibility over how it's done. To reproduce the second example above, we would just submit this code.

```
proc sgplot data=who14;
    scatter x=villain y=episodes;
    xaxistable year_last total_stories year_first / location=inside;
run;
```

Notice that we were able to put the labels for the table on the YAXIS. We can actually move it back to the Y2AXIS by adding that option to the statement.
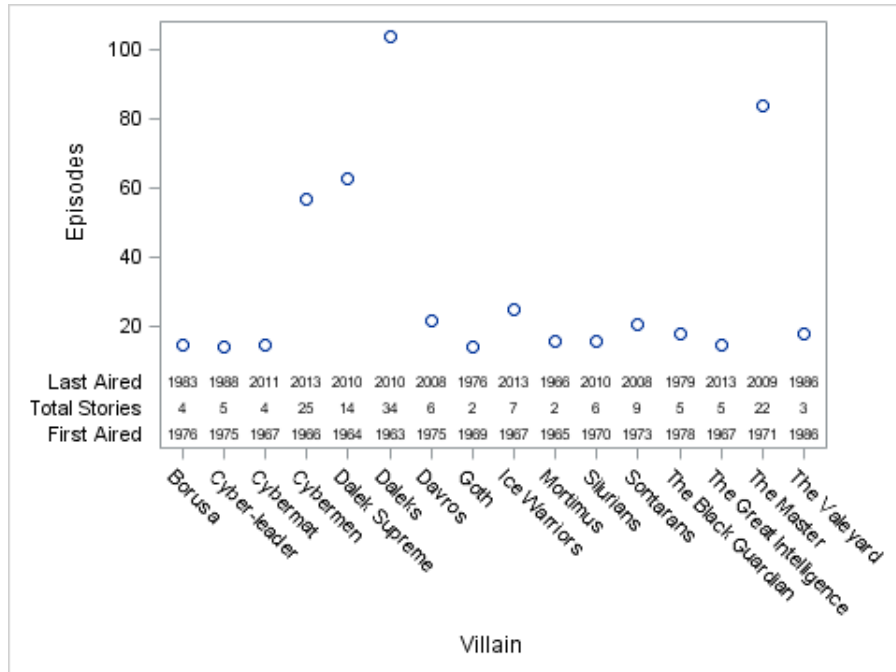
Figure 4

	Borusa	Cyber-leader	Cybermat	Cybermen	Dalek Supreme	Daleks	Davros	Goth	Ice Warriors	Mortimus	Silurians	Sontarans	The Black Guardian	The Great Intelligence	The Master	The Valeyard
Last Aired	1983	1988	2011	2013	2010	2010	2008	1976	2013	1986	2010	2008	1979	2013	2009	1986
Total Stories	4	5	4	25	14	34	6	2	7	2	6	9	5	5	22	3
First Aired	1976	1975	1967	1966	1964	1963	1975	1969	1967	1965	1970	1973	1978	1967	1971	1986

With these statements, we can also control the LOCATION of the table as INSIDE (like we did above) or OUTSIDE of the axis area, as well as POSITION it either at the TOP or BOTTOM of graph. You can also control how the labels and the data values are displayed. Here's the code for a full graph with most of the options specified.

```
proc sgplot data=who14;
    scatter x=villain y=episodes;
    xaxistable year_last year_first / location=inside position=bottom
            droponmissing separator title="Other Info" titleattrs=(weight=bold)
            valueattrs=(color=red) x2axis labelpos=right
            labelattrs=(style=italic) name="other";
run;
```
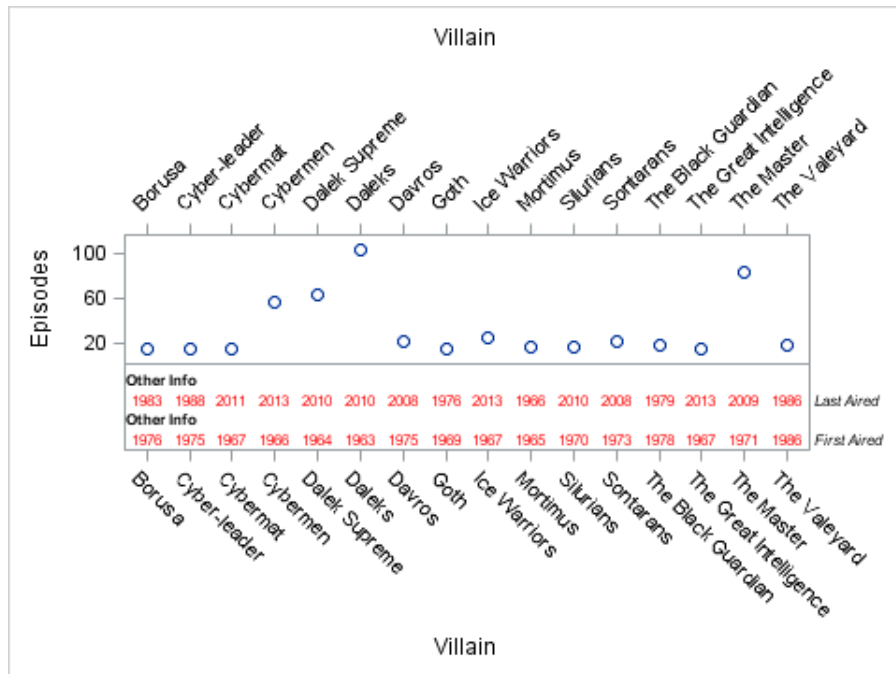
and the graph.

Figure 5

As silly as this graph is, it shows the effects of most of the available options. Notice that the TITLE applies to every row in the table. If this is useful for you, then good. The labels on the x-axis are on the top and the bottom because of the X2AXIS keyword. This actually allows us to have two sets of horizontal axes. An X= option would allow the values in the table to be placed according to another scale than the one in the primary graph.

The XAXISTABLE statement can be used multiple times. This allows you to place tables on the top and bottom of the graph (or left and right with YAXISTABLE) at the same time. It also allows you to do things like have a title for only one row of the table or customize the text or label attributes per row. You can also use XAXISTABLE and YAXISTABLE in the same plot.

STAT=

The axistables can also be used to show summary statistics related to the plot, particularly when the plot is categorical, such as the bar charts, line charts and dot plots. These plots are often summarizing a variable for particular levels of a categorical variable. This situation is well suited to display a statistic on the plotted variable or a related variable. In our current data set everything is summarized, so this type of table isn't necessary. One can see that by comparing the graphs from these two pieces of code.

```
proc sgplot data=who14;
    scatter x=villain y=villain_episodes;
    xaxistable total_stories;
run;
```
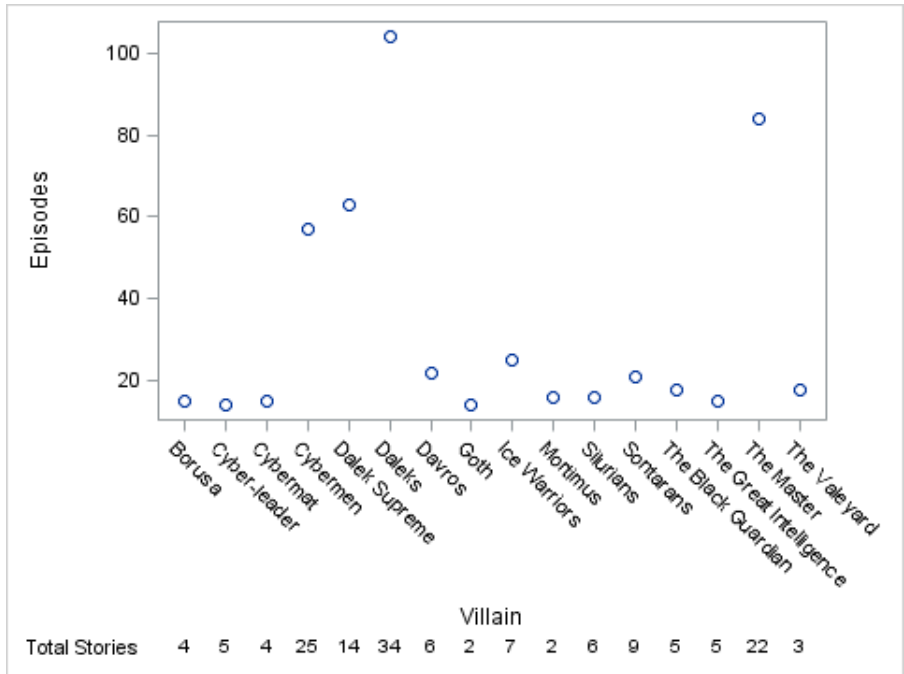
Figure 6

```
proc sgplot data=who14;
    scatter x=villain y=villain_episodes;
    xaxistable total_stories / stat=mean;
run;
```
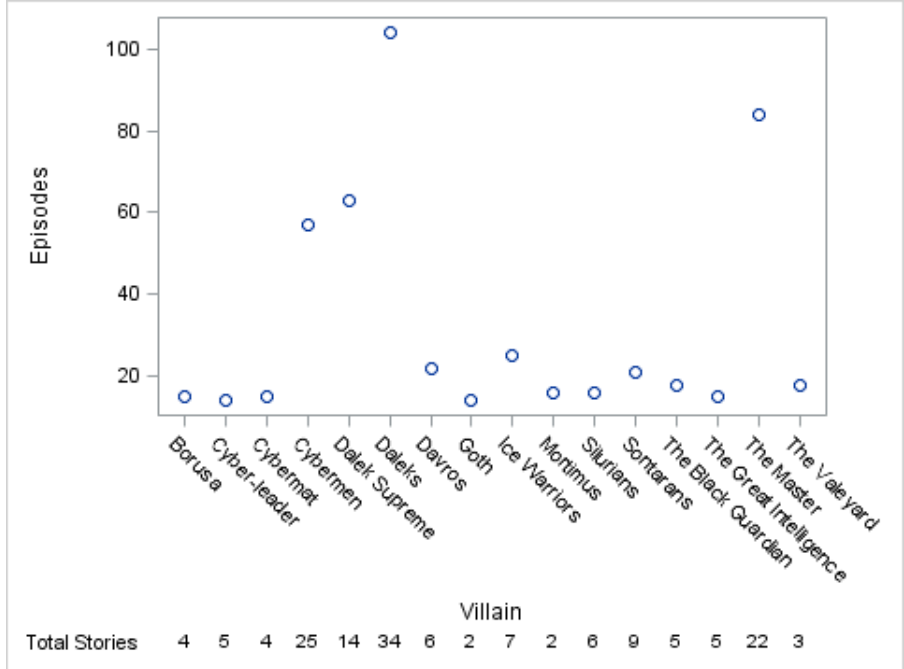

Figure 7

Let's use another dataset, then, called WHO_EP with the following variables, among others.

Variable Name	Type	Meaning
Doctor_number	Numeric	The number of the regenerations of the Doctor. Also the number of the actor playing the Doctor.
Doctor	Character	The actor playing the Doctor.
Season	Numeric	The season number that this actor part of.
Title	Character	The title of the stories this actor played in.
Story_episodes	Numeric	The number of episodes made to tell the story.

We'll create a dotplot of the total number of episodes in all the stories for each doctor and add a YAXISTABLE to show the median number of episodes per story for each doctor. Note that we specify STATLABEL to show, in this case, "(Median)" on the table. The default is to show the statistic calculated, but for this example, PROC IMPORT was used to import the dataset from a spreadsheet and PROC IMPORT adds labels to the variables. In this situation, i.e. when a custom label is assigned to the variable, the statistic is not displayed. We can easily force it to be displayed, though, as below.

```
proc sgplot data=who_ep;
    dot doctor / response=story_episodes;
    yaxistable story_episodes / stat=median statlabel;
run;
```
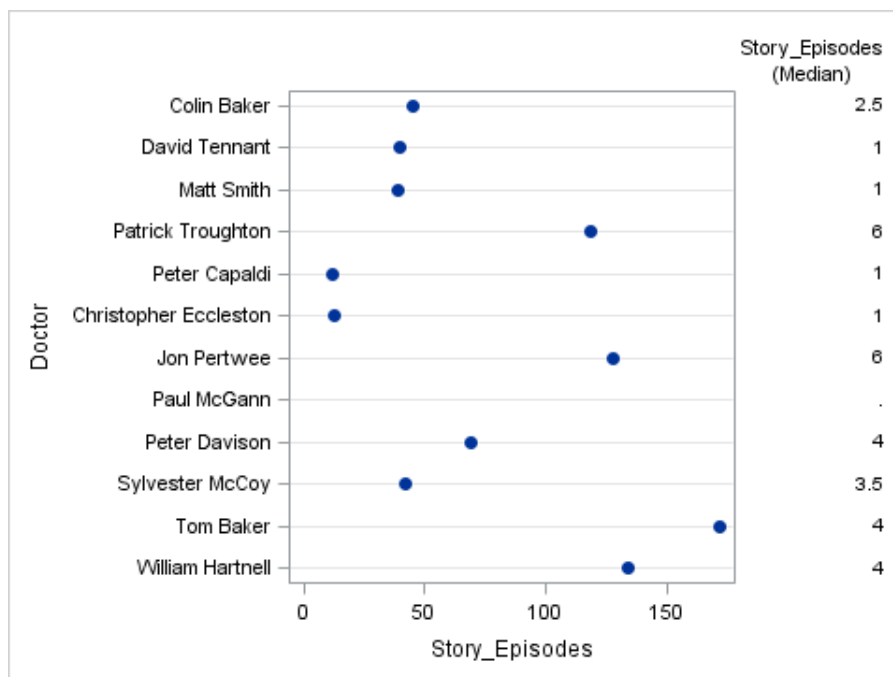

Figure 8

The X|YAXISTABLE statements are much easier to use for adding tabular information to the plot. As with the rest of ODS, they give the user almost complete control over the layout and look of the tables. What if we want to add information that is not so structured? With SAS 9.x we now have the option of adding annotations to our plots.

ANNOTATION

For those who are familiar with SAS/Graph, annotating a graph is likely very familiar. Some people have even skipped the built in graphs and created their entire visualization with annotations. These are the true artists. Since SAS 9.x annotation has been available to users of ODS Graphics with much of the functionality as before.

Many of the things you might want to do with SG Annotation can be done with other ODS statements, such as putting text on the plot or drawing lines. Let's see some examples of both.

If we want a horizontal line on my plot such as a line to indicate that the villains who appeared in more than 40 episodes are very scary, then we can use the REFLINE statement.

```
proc sgplot data=who14 noautolegend;
    scatter x=villain y=episodes;
    refline 40 / label="Very scary!";
run;
```
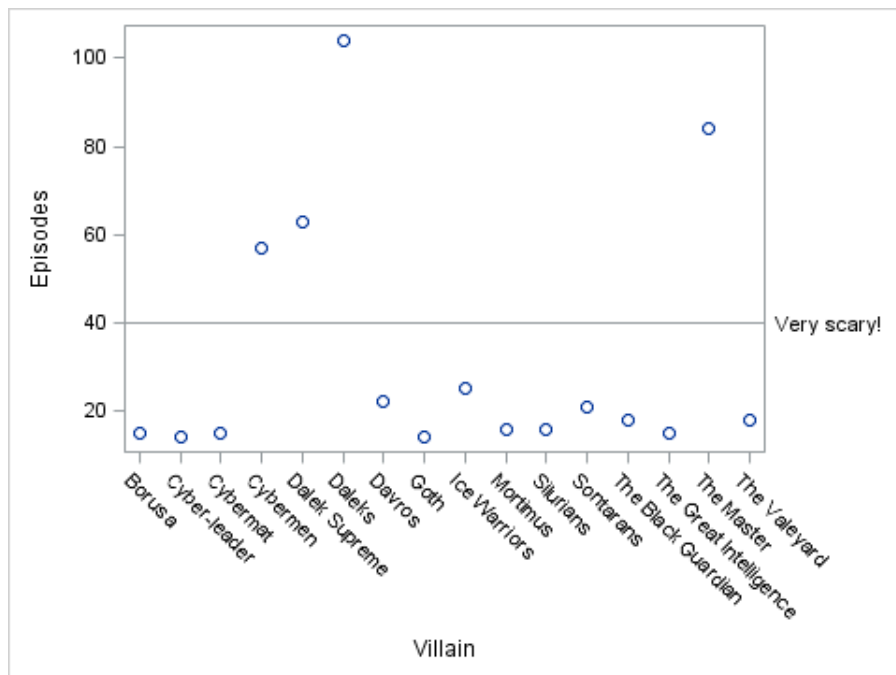

Figure 9

We can also create an annotate data set and draw it "manually." Just like in SAS Graphics, there are particular variable names and keywords to use. We specify

- the FUNCTION that we want the graphics engine to perform

- where we want it to perform that function

- the coordinate system it uses to do the positioning

- how we want the output to look

For our reference line we might have

```
data annoline;
    infile datalines dlm="#";
    retain x1space "DATAPERCENT" y1space "DATAVALUE" x2space "DATAPERCENT" y2space
"DATAVALUE";
    length label $ 15;
    input function $ x1 y1 label x2 y2;
datalines;
TEXT # 90 # 45 # Very scary! # . # .
LINE # 1 # 40 # # 99 # 40
;;
run;
```

to create the following data set.

OBS	x1space	y1space	x2space	y2space	label	function	x1	y1	x2	y2
1	DATAPERCENT	DATAVALUE	DATAPERCENT	DATAVALUE	Very scary!	TEXT	90	45	.	.
2	DATAPERCENT	DATAVALUE	DATAPERCENT	DATAVALUE		LINE	1	40	99	40

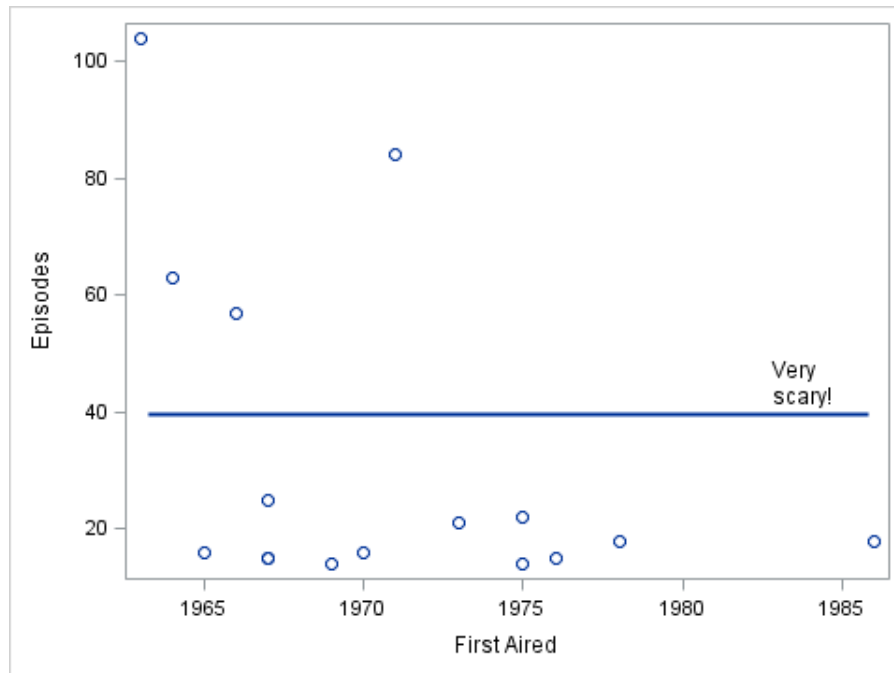With this data set we will be telling the SGPLOT procedure to draw TEXT at coordinates (90, 45) and a LINE from (1,40) to (99, 40).
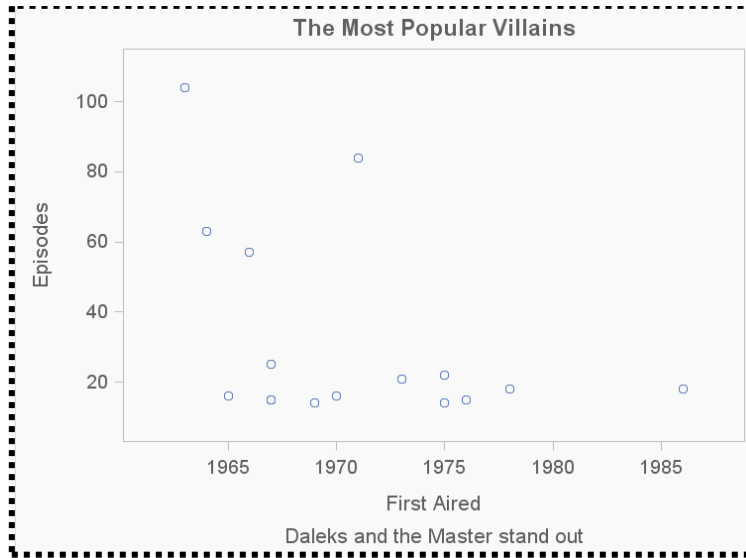

Figure 10

Be aware, though, that those instructions are not as simple as they may seem. Each of those coordinates is relative to some "drawing space." There are four drawing areas that we can use: graph, layout, wall and data. Suppose we run this code.

```
proc sgplot data=who14;
    title "The Most Popular Villains";
    scatter x=year_first y=episodes;
    footnote "Daleks and the Master stand out";
    xaxis offsetmin=0.1 offsetmax=0.1;
    yaxis offsetmin=0.1 offsetmax=0.1;
run;
```

Notice that we have a title and a footnote, as well as offsets for the X and Y axes.

Figure 11a-d

The Most Popular Villains

Episodes

1965   1970   1975   1980   1985

First Aired

Daleks and the Master stand out

The GRAPH drawing area encompasses the entire graph: plot, axes, titles, footnotes, etc.

The Most Popular Villains

Episodes

1965   1970   1975   1980   1985

First Aired

Daleks and the Master stand out
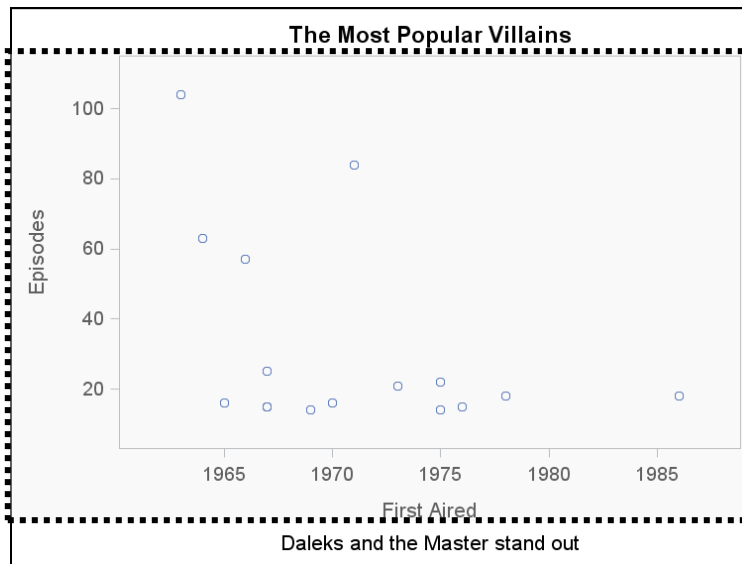
The LAYOUT drawing area does not include the titles and footnotes, but does include the rest of the graph area.

The Most Popular Villains

Daleks and the Master stand out
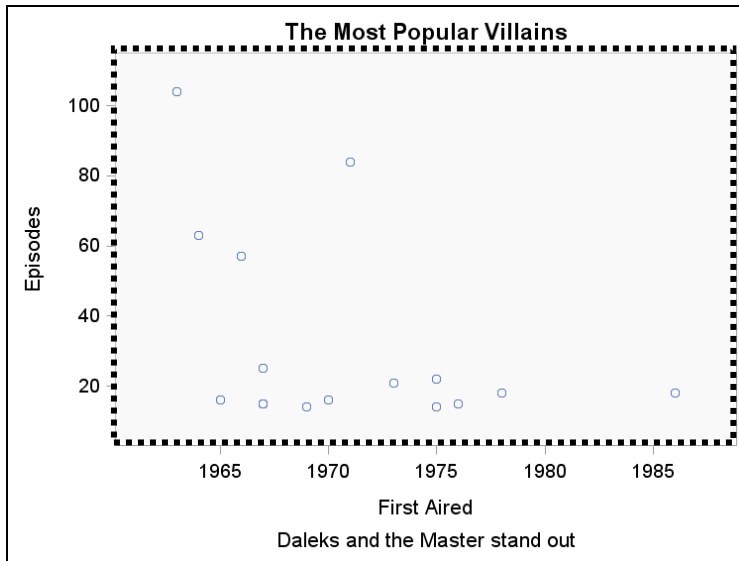
The WALL drawing area does not include titles, footnotes, and axes. It does include all the area within the axes, including any offsets.

The Most Popular Villains

Daleks and the Master stand out
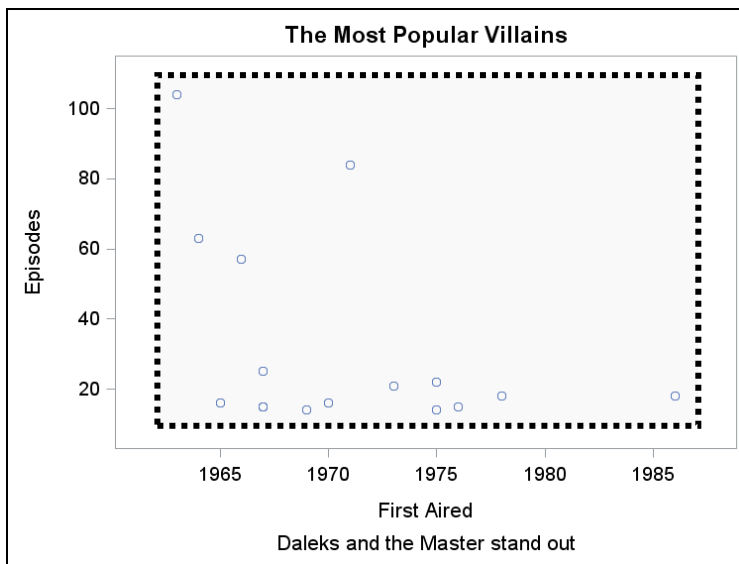
Finally, the DATA drawing area is just that. It includes only the square that the data takes up. It does not include any of the offsets.

For each of these we can specify the scaling as a percentage of the drawing space or by the number of pixels. The first is useful when you want to specify relative coordinates. For our TEXT above with the coordinates (90, 45), DATAPERCENT says to anchor the text at 90% of the horizontal space within the data area. The second scaling gives you strict control over the annotation's positioning. If we had said DATAPIXEL for X1SPACE and X2SPACE, then the line would have started at 1 pixel into the data area and ended at 99 pixels. The look of the output will depend on both the drawing space chosen and the size (width or height) of the graph. A wider graph means that there are more pixels being drawn in essentially the same drawing space, so the line will be shorter relative to the rest of the graph.

It can also be very helpful to use DATAVALUE as the drawing space. This positions and scales the annotation with respect to the values of the data, which may make it easier to draw by eye or, better yet, the coordinates could be data-driven.

Suppose we want to identify the Daleks (Daleks at (1963,104) and Dalek Supreme at (1964, 63)) on the plot below by circling them with an oval. We can use the values of the points to draw the oval. We'll center the oval at the midway point between the two data points. We'll set the WIDTH and HEIGHT to be functions of the distance between the two points and, finally, ROTATE, the oval based on the angle of the line connecting the two points. Here is the code for the calculations.

```
data oval;
    set oval;
    length function $8;
    distx = dsx - dx;
    disty = dsy - dy;
    distance = sqrt(distx**2 + disty**2);
    x1 = distx/2 + dx;
    y1 = disty/2 + dy;
    angle = atan2(disty, distx) * 180 / constant('PI');
    function = "oval";
    width = distance*0.8;
    height = distance*0.25;
    rotate = angle*0.98;
    drawspace = "datavalue";
    output;
    function = "text";
    x1 = dsx*1.001;
    y1 = dy*0.9;
    label = "Daleks are";
    rotate = 0;
    anchor = "left";
    output;
    function = "textcont";
    label = " Scary!";
    textcolor = "red";
    textsize = 20;
    textweight = "bold";
    output;
run;
```

We've also annotated the graph with an important notice about the Daleks as TEXT. For emphasis we've used the TEXTCONT function to continue the text in a different font. When TEXTCONT follows TEXT in the data set, the LABEL for TEXTCONT follows the LABEL for text so we put a space at the front to separate the two. See the code and graph below.

```
proc sgplot data=who sganno=oval;
    scatter x=year_first y=episodes;
run;
```
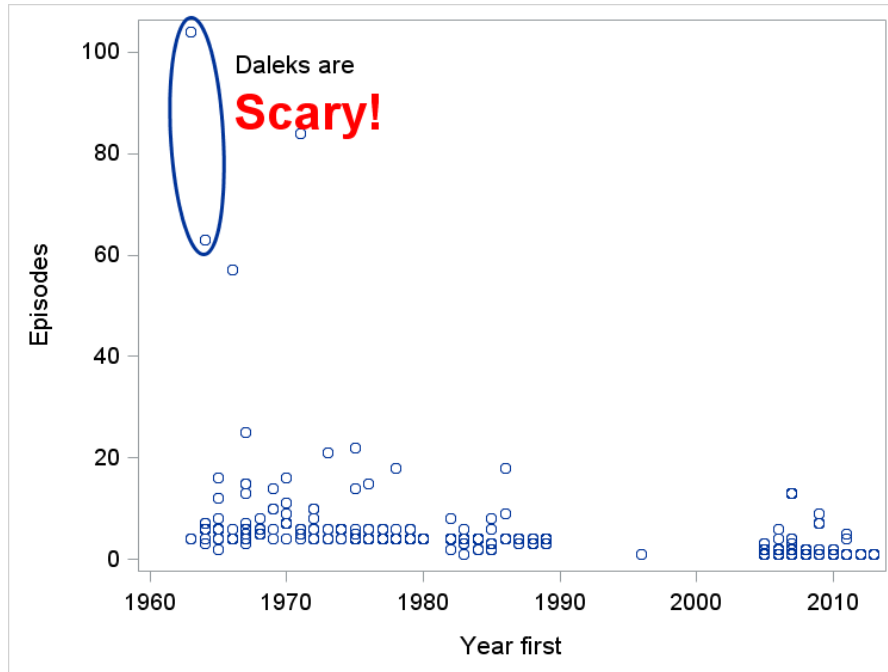
Figure 12

ATTRIBUTES

Even though the default values for ODS Graphics are well thought out and aesthetically pleasing, there are often times that we want to customize the look and feel of various parts of the graph. The good thing is that they have given us the ability to customize almost everything and a number of ways to do it. We'll look at some background first and then three ways to control the attributes.

BACKGROUND

A STYLE is what determines what ODS output is going to look like. It is a SAS template that can be associated with a particular ODS document. The default style used by the HTML destination is HTMLBlue. There is a definition for everything you can output in this template. There is a definition for how to produce the background, borders, headers, footers, table cells, web links, titles, etc. It's almost overwhelming at times how much control they give you. Each style is made up of elements that are named collections of style attributes. These style elements are the specific way the look and feel of an area of the ODS output is determined. For example, in the HTMLBlue style there is a style element for Header that specifies the bordercolor, backgroundcolor and color of the Header. For a list of the Style Elements Affecting Template-Based Graphics, refer to the SAS System Documentation at SAS Products > Base SAS > SAS Output Delivery System (ODS) > SAS 9.4 Output Delivery System: User's Guide, Third Edition > ODS Style Reference > Style Elements.

As mentioned, the Style Elements are collections of attributes. According to the documentation just noted, the recognized attributes for the GRAPHDATADEFAULT style element, as an example, are Color, ContrastColor, MarkerSymbol, MarkerSize, LineStyle, LineThickness, StartColor, NeutralColor, and EndColor. These attributes determine the look and feel of non-grouped data items according to the Style set for the current destination. For example, in the graph below, the blue color for the line is defined by this style element because the style being used is HTMLBlue.
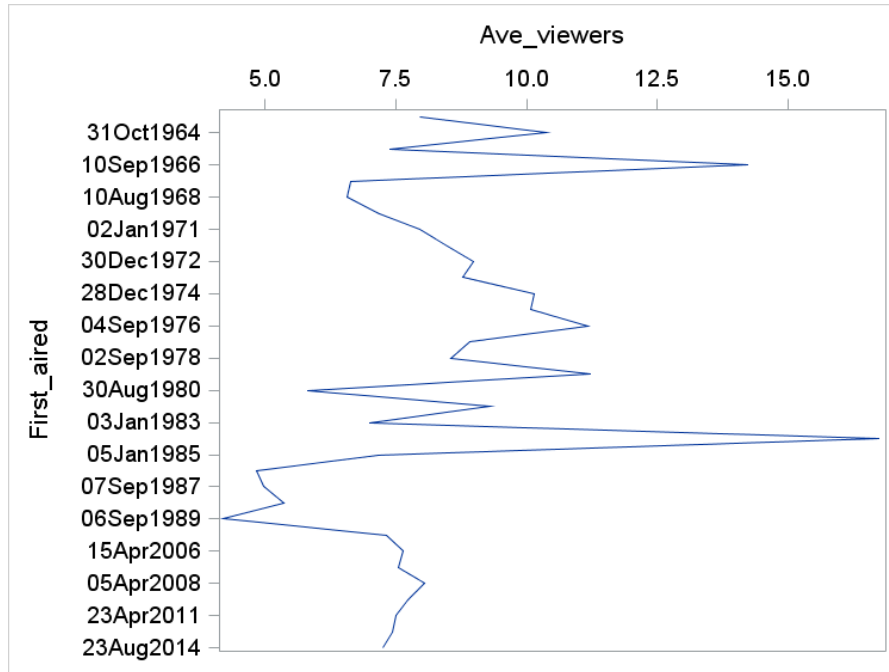
Figure 13

When we add a barchart, it keeps the same coloring so that they complement rather than contrast.
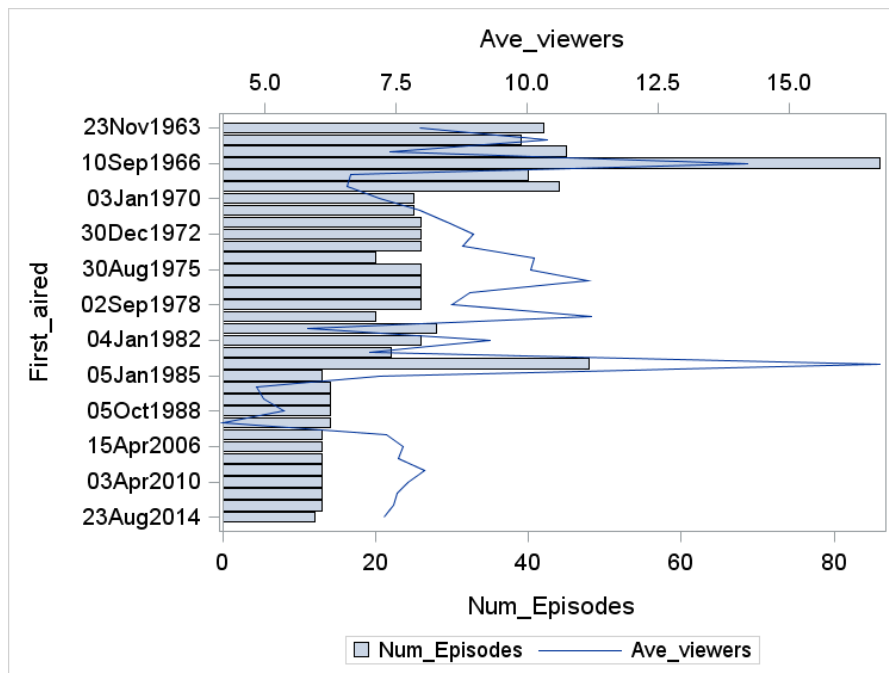

Figure 14

On the other hand, when there are grouped items or multiple items of the same type, then different style elements are used: GRAPHDATA1, GRAPHDATA2, GRAPHDATA3, etc., depending on how many data items there are. The relationship among these elements depends on the style being used. When color is the primary differentiator, then the colors for each of these are designed to enhance the differentiability of the pieces of information. For example, when we change our barchart to a line chart SAS shows the first line as blue and the second line as red. (see below)
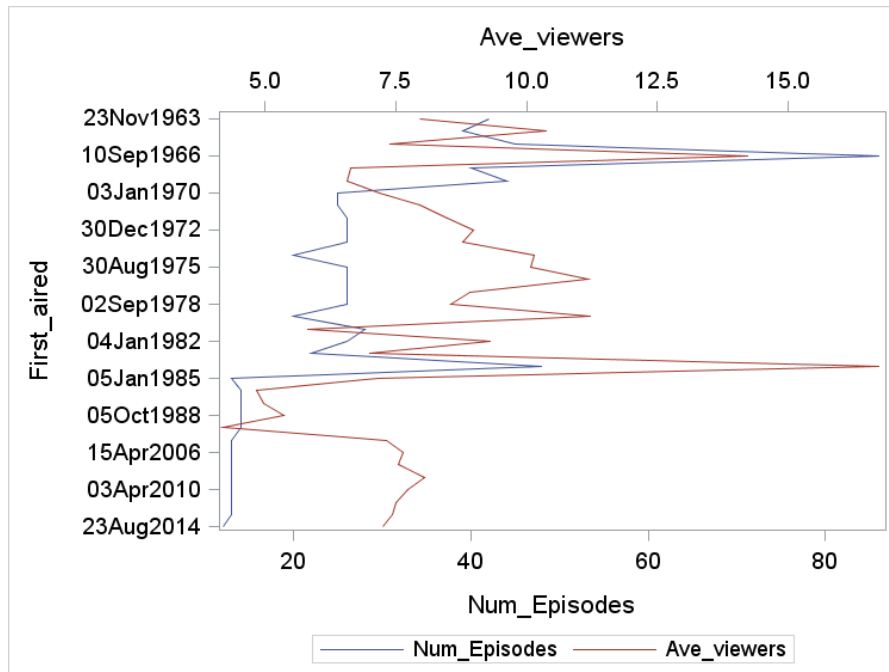
15

Figure 15

Other styles, such as those in black and white, use the pattern of the line to differentiate the grouped or multiple items.

As you can see from the example for GraphDataDefault, the style attributes are related (in a broad sense) to color, shape and size. The table below gives a variety of further examples.

"Color"	*"Shape"*	*Size*
CXFF0000	Courier New	5 CM
H07880FF	Solid	2.3 IN
Light Red	DashDashDot	20 MM
Matte	Italic	30%
Crisp	MMDDYY8.	55 PT

The color can be specified in any SAS standard format, such as a predefined SAS color ("orange" and "blue") or an RGB (red/green/blue) value. Another is HLS (hue/light/saturation) or even an English description of an HLS value. For example, you could code "very dark grayish yellow," if that's your thing. The shape, in a broad sense, can be specified by a font-definition, a SAS format, an integer list, a line pattern or marker symbol (e.g., ArrowDown, CircleFilled, or Star). The size can be specified by a number and the units of measure (CM=centimeters, IN=inches, MM=millimeters, PCT=percentage, PT=point size, PX=pixels).

With that brief background, let's look at some ways to customize the appearance of our graphs.

MARKERS

Markers are common in plots and customizing them to provide other information is very valuable. We've already seen this concept when we used SCATTER to make a data table. The "markers" were the numeric values we wanted to show. The two relevant options are

- MARKERCHAR=variable
- MARKERCHARATTRS=style-element <(options)> | (options)

The "variable" can be any quantitative or categorical variable, and if there's a format on the variable then it's applied as well.

16

The other Marker options are

- MARKERATTRS=style-element <(options)> | (options)

- MARKERFILLATTRS=style-element <(COLOR=color)> | (COLOR=color)

- MARKEROUTLINEATTRS=style-element <(options)> | (options)

- FILLEDOUTLINEDMARKERS

These are used to customize the built in marker symbols for the dot, scatter, series, step, line, needle and fit plots by specifying either a Style Element or specific Style Attributes. The first three have an interaction with FILLEDOUTLINEDMARKERS. If we don't use FILLEDOUTLINEDMARKERS, then MARKERATTRS can control the COLOR, SIZE and SYMBOL, whether we pick a filled symbol or not. The PROC also ignores MARKERFILLATTRS and MARKEROUTLINEATTRS.

If we do use FILLEDOUTLINEDMARKERS, then the COLOR option in MARKERATTRS is ignored. Instead the color is controlled by the MARKERFILLATTRS and the MARKEROUTLINEATTRS options. The MARKEROUTLINEATTRS can also control the THICKNESS of the outline on the marker. Note also, that when the FILLEDOUTLINEDMARKERS option is used, then a FILLED symbol must be used, either specified explicitly (MARKERATTRS) or defaulted from the active style.

```
proc sgplot data=whovill;
    scatter x=villain y=episodes / filledoutlinedmarkers
            markerattrs=(symbol=CircleFilled size=15);
run;
```
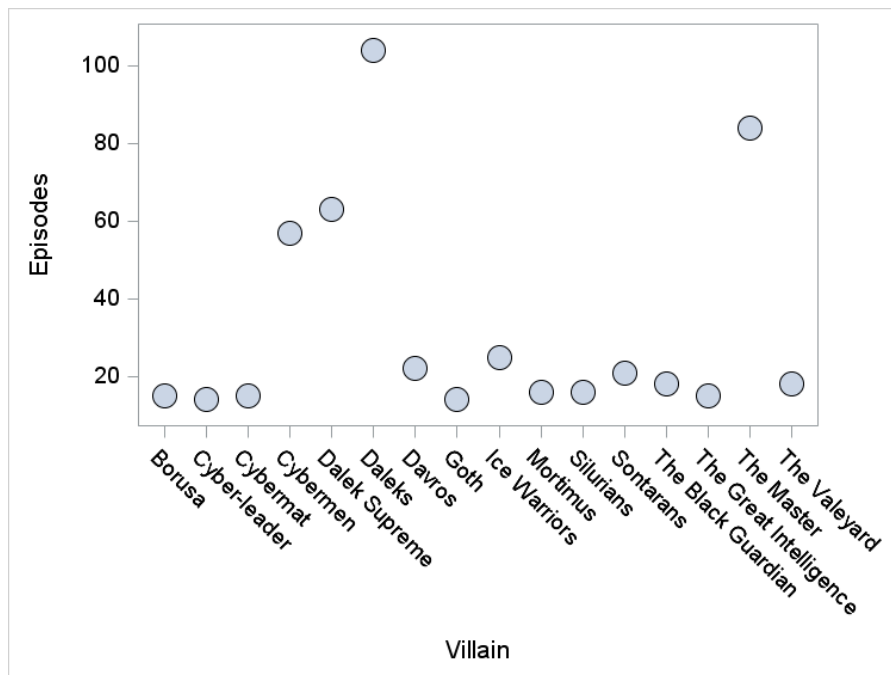

Figure 16

```
proc sgplot data=whovill;
    scatter x=villain y=episodes / filledoutlinedmarkers
        markerattrs=(symbol=CircleFilled size=15) markerfillattrs=(color="#003865")
        markeroutlineattrs=(color=red);
run;
```
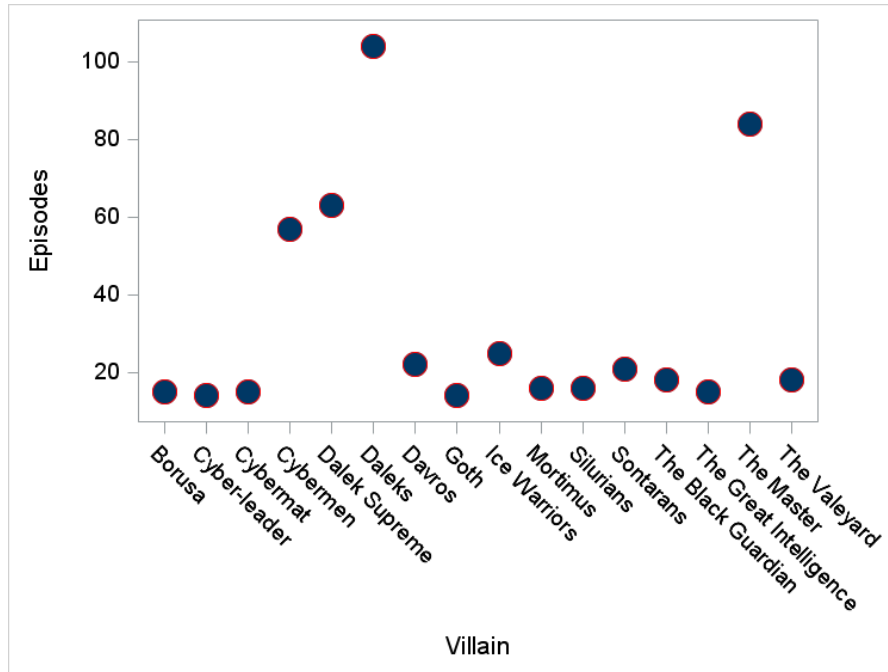
17

Figure 17

CYCLEATTRS

As we mentioned, the GraphDataX style elements are used when multiple plots of the same type are created or the data is plotted in groups so that different attributes, such as color, are used for different lines. Sometimes we want to force different colors to be used and sometimes we want to use all the same color. We have control over this through the CYCLEATTRS and NOCYCLEATTRS options. CYCLEATTRS means to cycle the attributes through the available options. In our example above, we have a VLINE and an HBAR chart that both have the same attributes. Adding the CYCLEATTRS option on the SGPLOT statement

```
proc sgplot data=who_seasons cycleattrs;
    hbar first_aired / response=num_episodes stat=sum;
    hline first_aired / response=ave_viewers x2axis;
run;
```
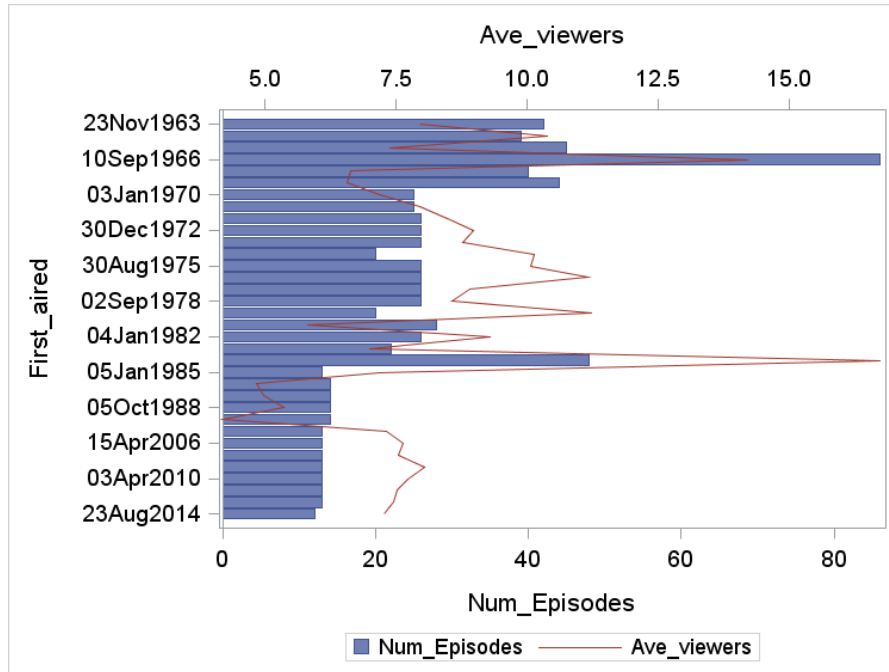
gives this graph.

Figure 18

Since the HBAR is first it is drawn in blue and the HLINE is drawn in red. The attributes that ODS cycles through depends on the style being used. The style in effect here is the default (HTMLBLUE). If we had been using the JOURNAL style, then ODS would have changed the line pattern as below, since it does not use colors.
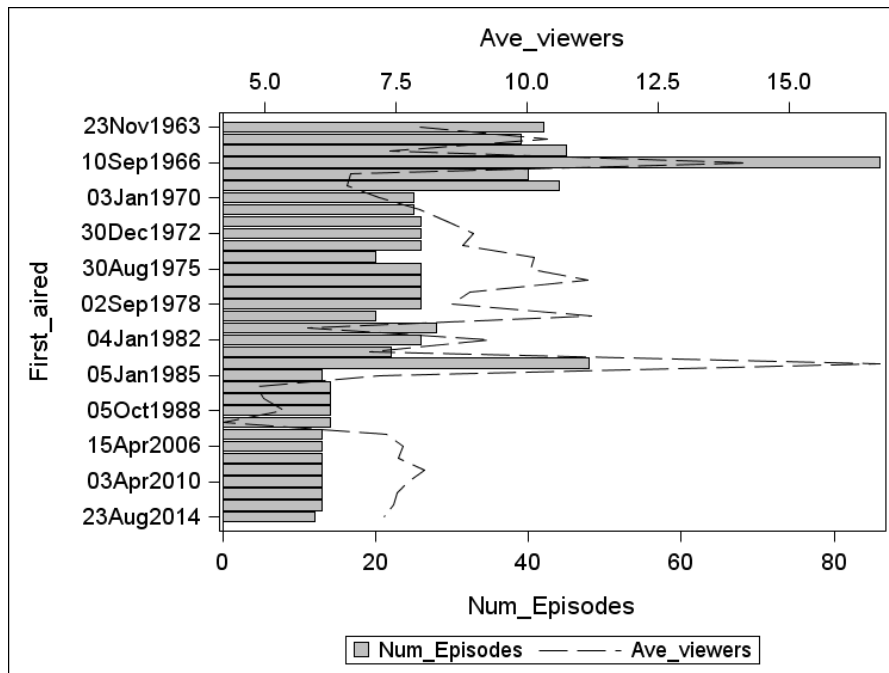

Figure 19

Going the other direction, we can force the second example above to show the same colors by adding the NOCYCLEATTRS option. We might want this if we are being very specific about some of the attributes we want displayed and we want the other attributes to stay consistent.

STYLEATTRS

The CYCLEATTRS option is nice, but it's a far cry from everything that can be done. We touched upon some of those capabilities in the previous paper. One of the newer features is the STYLEATTRS statement for SGPLOT and SGPANEL. This statement shortcuts a lot of the work of creating a new ODS style template that was previously required. Now the programmer can control the colors, contrast colors, line patters and marker symbols of graphic elements. The STYLEATTRS statement can be used in conjunction with the ATTRPRIORITY option on the ODS GRAPHICS statement, but this is still rudimentary. If the user community finds it helpful, then I expect it to be expanded in future releases.

This statement is pretty straightforward to understand and use (like most things in ODS Graphics!). The simple scatter plot below shows the average number of viewers for each season. We've grouped the display of the information by which incarnation of the Doctor was in the show.
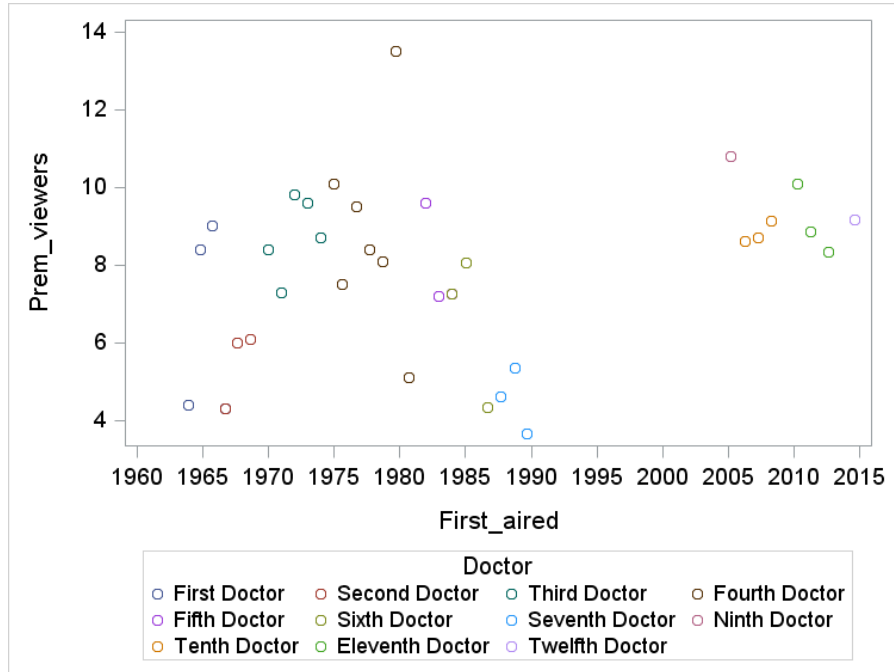

Figure 20

This graph comes because we are using the default style, HTMLBLUE, whose pattern for differentiating is based on colors. If we wanted to differentiate the markers by symbol also, then we would use the following statements.

```
ods graphics / attrpriority=none;

proc sgplot data=who_seasons nocycleattrs;
    styleattrs datasymbols=(ibeam circle circlefilled diamond diamondfilled
                            hash starfilled triangle trianglefilled square
                            squarefilled);
    scatter x=first_aired y=prem_viewers / group=doctor;
run;
```

The ATTRPRIORITY option on the ODS GRAPHICS statement is important because it tells ODS to not give color the highest priority for differentiating. If we had not used that option, then ODS would have displayed a bunch of multi-colored ibeams. Essentially, we tell it to not set a priority because we define it in the STYLEATTRS statement. Here, we have set the symbols as the first attribute to be changed and we get this plot.
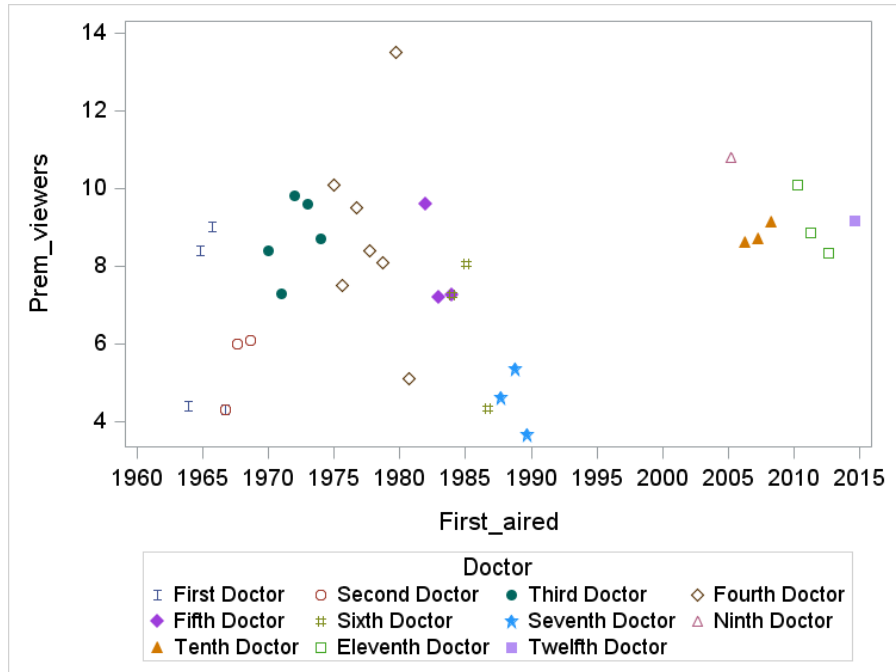
Figure 21

Notice that the colors change with each of the symbols. They change because we are still in HTMLBlue with groups and it is still using GraphData1, GraphData2, …. To go all the way and have the symbol be the only differentiator, we would have to specify one color as in the following code snippet.

```
styleattrs datasymbols=(ibeam circle circlefilled diamond diamondfilled hash
                        starfilled triangle trianglefilled square squarefilled)
            datacontrastcolors=(blue);
```

We specified a different symbol for each level of the group. If we had not, then ODS would have started over with the ibeam once it got through the list. This cycling also works across attribute specs. If we had used DATACONTRASTCOLORS=(BLUE RED) instead, then the symbols would alternate between blue and red as below.
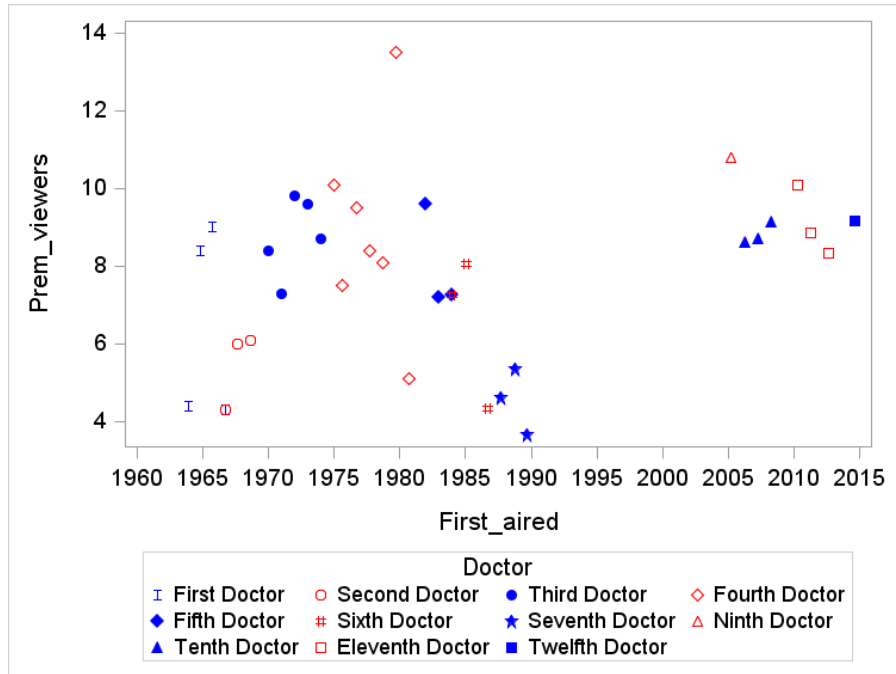
Figure 22

Stepping back now, there are 4 attributes that can be changed in the STYLEATTRS statement.

Argument	Controls
DATACOLORS	Fill colors
DATACONTRASTCOLORS	Contrast colors for lines and markers
DATALINEPATTERNS	Line patterns
DATASYMBOLS	Marker symbols

When you include a STYLEATTRS statement with one or more of these arguments, you are redefining the GraphData1, GraphData2, … GraphData*n* style elements (discussed above and shown in the table below) which are then applied to each graphic element to determine its look.

Style Element	Color	Contrast Color	Marker Symbol	Line Pattern
GraphData1	cx7C95CA	cx2A25D9	CIRCLE	1
GraphData2	cxDE7E6F	cxB2182B	PLUS	4
GraphData3	cx66A5A0	cx01665E	X	8
GraphData4	cxA9865B	cx543005	TRIANGLE	5
GraphData5	cxB689CD	cx9D3CDB	SQUARE	14
GraphData6	cxBABC5C	cx7F8E1F	ASTERISK	26
GraphData7	cx94BDE1	cx2597FA	DIAMOND	15
GraphData8	cxCD7BA1	cxB26084		20
GraphData9	cxCF974B	cxD17800		41
GraphData10	cx87C873	cx47A82A		42
GraphData11	cxB7AEF1	cxB38EF3		2

GraphData12	cxDDD17E	cxF9DA04

When we changed the DATASYMBOLS we replaced the default values of CIRCLE, PLUS, X, etc. with IBEAM, CIRCLE, CIRCLEFILLED, etc. When we included the DATACOLORS=(BLUE RED) the table becomes

Style Element	*Color*	*Contrast Color*	*Marker Symbol*	*Line Pattern*
GraphData1	blue	cx2A25D9	IBEAM	1
GraphData2	red	cxB2182B	CIRCLE	4
GraphData3	blue	cx01665E	CIRCLEFILLED	8
GraphData4	red	cx543005	DIAMOND	5
GraphData5	blue	cx9D3CDB	DIAMONDFILLED	14
GraphData6	red	cx7F8E1F	HASH	26
GraphData7	blue	cx2597FA	STARFILLED	15
GraphData8	red	cxB26084	TRIANGLE	20
GraphData9	blue	cxD17800	TRIANGLEFILLED	41
GraphData10	red	cx47A82A	SQUARE	42
GraphData11	blue	cxB38EF3	SQUAREFILLED	2
GraphData12	red	cxF9DA04		

If a 12th group had shown up, then the patterns would have been repeated to give us a red ibeam next. (NB: Even though the show is on the twelfth incarnation of the Doctor, there are only 11 groups, because the 8th Doctor is not in the dataset. He only appeared in a show twice: once in the 1996 TV movie *Doctor Who* and then in the 2013 YouTube video *The Night of the Doctor*. On the other hand, it is interesting to note that outside of the screen, he has appeared in more stories than any other Doctor. [tardis.wikia.com/wiki/Eighth_Doctor])

ATTRIBUTE MAPS

Another way to control the appearance of grouped data is with Attribute Maps. Attribute Maps are overkill for controlling attributes in single, one-time graphs. They really come into play when you want to create graphs with an attribute scheme that is independent of the data structure and/or consistent across graphs.

The map is a data set (isn't everything?) where each observation assigns attributes to one level of the group. Each element is connected to a series of attributes sort of like a SAS format. Like the Annotation data sets and the format data sets, the attribute map data set has a specific structure.

Variable	
ID (required)	The connector between the data set and the graphing element. This ID is used in the ATTRID option. It is like the FMTNAME variable in a CTRL data set for user defined formats.
VALUE (required)	The connector between the record of attributes and the level of the group. It is like the START variable in a CTRL data set. Because groups can only be categorical (and not ranges like in a format), VALUE is a case sensitive CHAR variable. _MISSING_ and _OTHER_ are also valid values to handle missing values and group values not explicitly defined, respectively.
and one or more attributes such as	FILLCOLOR, MARKERSYMBOL, LINEPATTERN, etc.

By defining these maps it doesn't matter what order the input data is in, whether any group levels are missing, or what other graph items are on the plot. The attributes will always be as you specified. Let's look at a simple example.

Suppose we wanted a bar chart that shows the average number of viewers for each doctor. We would like to group the output by the decade that the episodes were shown. Below is the first example with all the data.

```
proc sort data=who_seasons;
    by doc_no;
run;

proc sgplot data=who_seasons;
    hbar doctor / response=ave_viewers group=decade;
    yaxis discreteorder=data;
run;
```
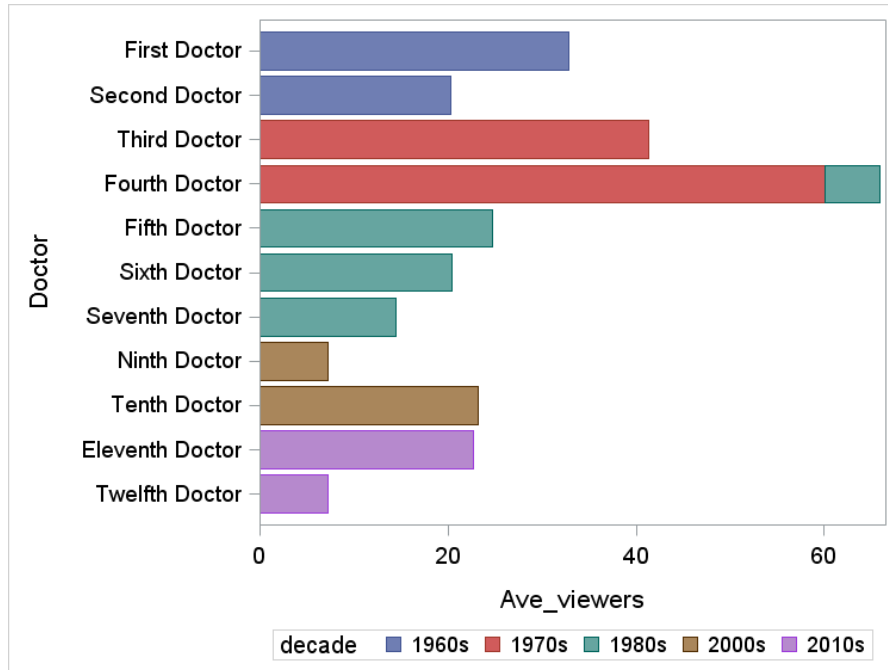

Figure 23

GraphData1 … GraphData4 use the blue, red, green, brown and violet colors to differentiate the groups (1960s, 1970s, 1980s, 2000s, and 2010s respectively). Now suppose that next month we are given different requirements for what to show. In this case, since we are deciding which shows to re-air and many of the early episodes are lost, we only care about the average number of viewers for those seasons with all episodes complete. The graph below shows this plot created with the following code.

```
proc sgplot data=who_seasons(where=(some_missing="No"));
    hbar doctor / response=ave_viewers group=decade;
    yaxis discreteorder=data;
run;
```
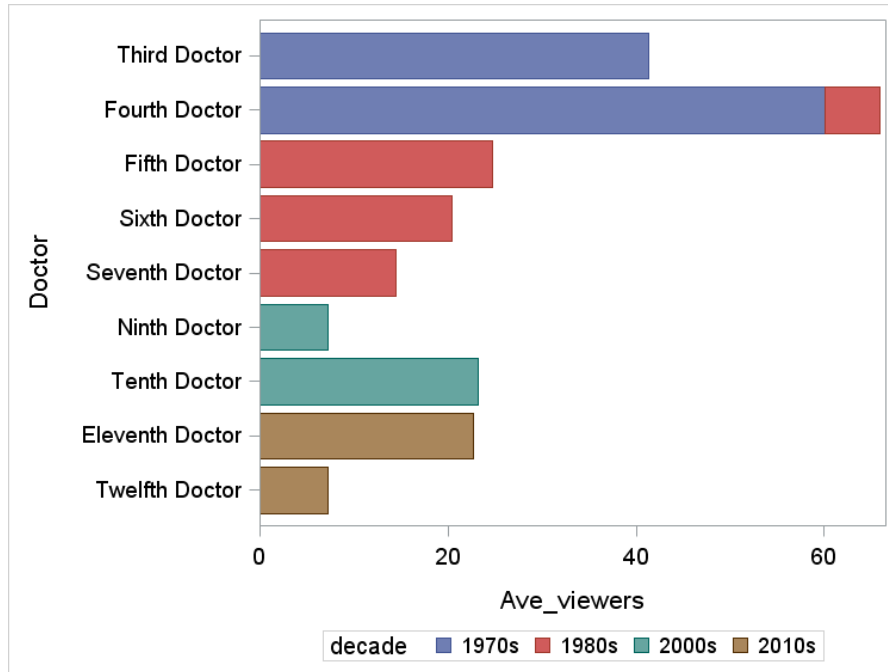
Figure 24

Note that the colors are the same – blue, red and green – but they are associated with different decades because the 1960s are missing. If we don't want to confuse our managers, then maybe we should keep the colors the same. Data attribute maps are the best way to do this.

We can define a data attribute map data set (DAMPD) to be the following.

ID	Value	Fillcolor	Linecolor
decades	1960s	brown	black
decades	1970s	green	black
decades	1980s	red	black
decades	2000s	blue	black
decades	2010s	purple	black

With this attribute map we modify the previous code in the following simple ways. First, we add the DATTRMAP option to the PROC SGPLOT statement so that we can connect the data set to the PROC (1). Then we add the ATTRID= option to the plot statement so that we can connect the appropriate ID (decades) to the values plotted (2).

```
proc sgplot data=who_seasons dattrmap=dampd;                    ①                              ②
    hbar doctor / response=ave_viewers group=decade attrid=decades;
    yaxis discreteorder=data;
run;
                                                                                                ①
proc sgplot data=who_seasons(where=(some_missing="No")) dattrmap=dampd;
    hbar doctor / response=ave_viewers group=decade attrid=decades;
    yaxis discreteorder=data;                                   ②
run;
```
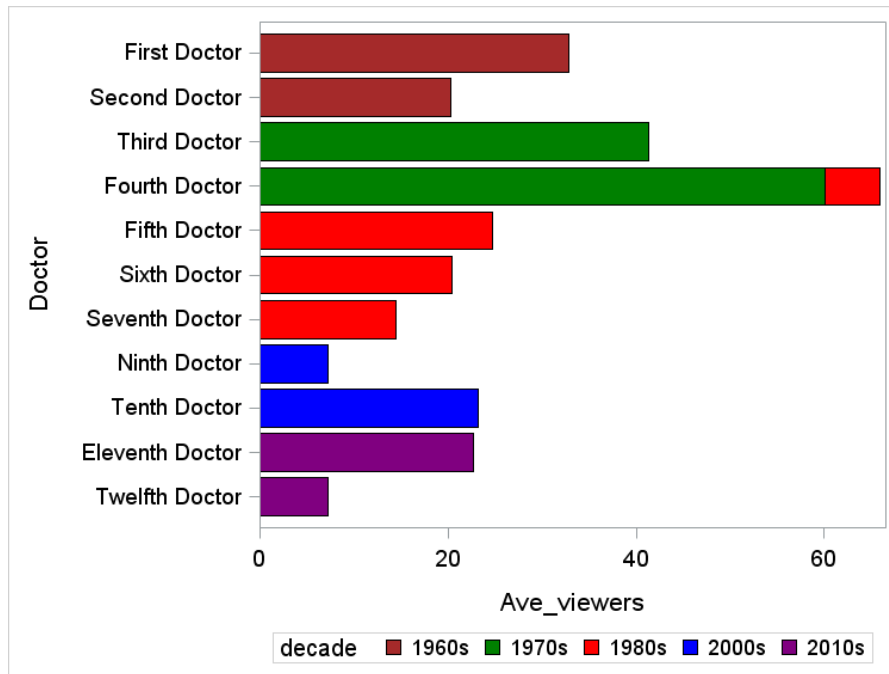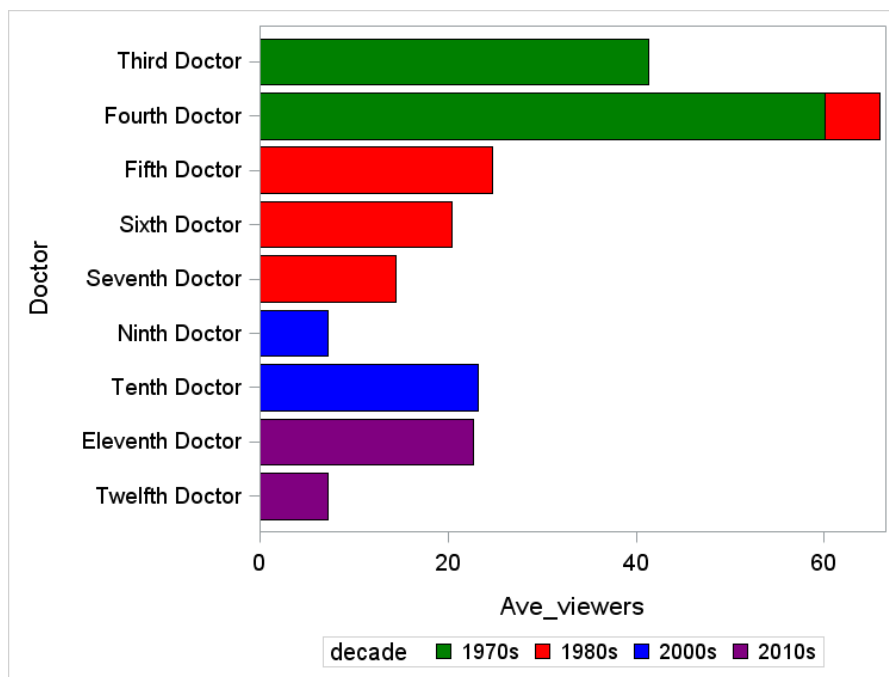
Figure 25

Figure 26

Now we see that each group level has the same color across graphs, even though some are missing. Note that we can use attribute maps for any graph with grouped data to achieve this consistency.

Just like formats, two or more maps may be combined in a single data set. The ID variable is what keeps them distinct. The other variables have values specific to the group being mapped. The two restrictions that apply are that one group cannot be associated with two maps in the same dataset, and that the data set must be sorted by ID values so that they are consecutive.

BLOCKS

The BLOCK plot is a new plot that, most of the time, is used to enhance another plot. For example, Suppose we want to show a line plot of the number of viewers who watched the premiere of each season versus the date the season first aired. We also want to identify the doctor associated with each season. We could do that by using Doctor as a group variable and we would get different colors or symbols or line patterns for each doctor. We can also use blocks as a background over which the lines would be laid. The code below does just that.

```
proc sgplot data=who_seasons;
    block x=first_aired block=doctor / transparency=0.3 valuefitpolicy=splitalways;
    series x=first_aired y= prem_viewers / markers;
    yaxis label="# of Premiere Viewers (millions)";
    xaxis label="Date First Aired";
run;
```
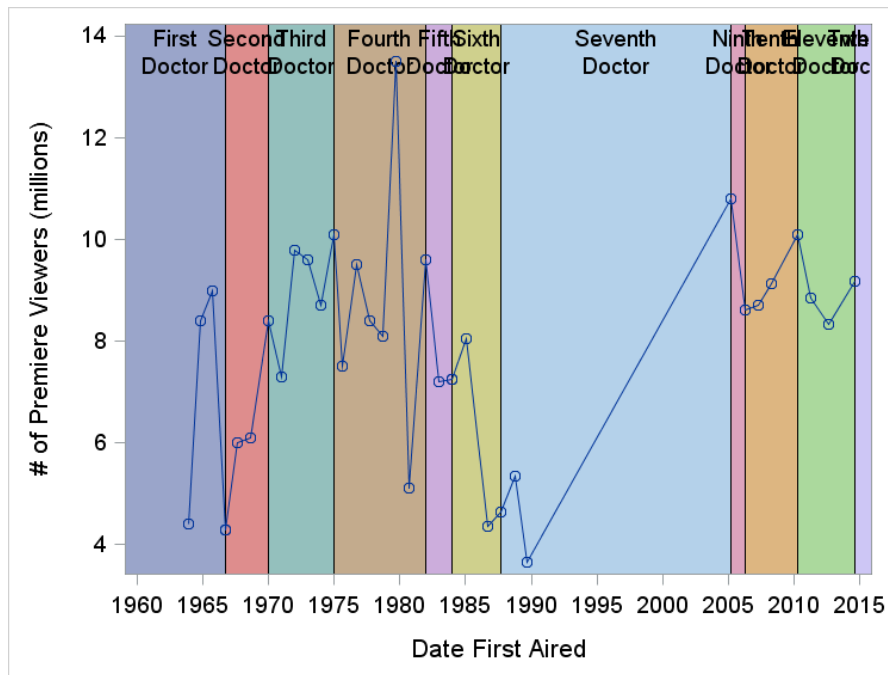

Figure 27

We create the blocks first and then plot a SERIES line with MARKERS. The blocks come out by default with a darker color than we want, since it hides the lines and the text. To reduce this we set the TRANSPARENCY of the blocks to 0.3. The block labels ("First Doctor", "Second Doctor", etc.) are also cumbersome. We can't fix that completely, but we can use one of the VALUEFITPOLICY options to make it better. Here we chose SPLITALWAYS so that it always splits the text. We use the default SPLITCHAR which is a blank space. [Note: the documentation for the BLOCK statement under PROC SGPLOT is wrong. It says that VALUEFITPOLICY=SPLITALWAYS (and SPLIT) has no effect unless the split character is defined with the SPLITCHAR= option. The BLOCKPLOT statement under the GTL section seems to be the correct description of the PROC's behavior.

As always you can control most of the appearance of the BLOCK plots: fill, outline, values, and text.

CONCLUSION

Each release the SAS ODS Graphics development team adds many new features. They listen to their customers and add minor options and major capabilities based on the requests they get. This paper is intended to highlight a few of the more advanced and some of the newer capabilities. There are so many more that they have made available to us. The documentation and the Data Visualization blog are wonderful reads. Enjoy!

REFERENCES

- Graphically Speaking, http://blogs.sas.com/content/graphicallyspeaking/

ACKNOWLEDGMENTS

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Chuck Kincaid
Experis Business Analytics
5220 Lovers Lane
Portage, MI 49002
269-553-5140
chuck.kincaid@experis.com
www.experis.us/analytics