Paper PP24

Figures are Basic(ally Overlooked)

Kaleb Mead, MAC Clinical Research, Swansea, Wales
Craig Parry, MAC Clinical Research, Swansea, Wales

ABSTRACT

Many people seem to think that Figures (or Graphs) are more complicated than they actually are, resulting in programmers developing a lack of confidence or never producing any in their career. But why? SAS® makes the production of Figures rather straight forward and hugely customizable. Identifying and exploring the common reasons as to why programmers may avoid Figures, will encourage more programmers to seek out Figure development and in turn see the industry adopt the forgotten child i.e., the Figures in "tables, Figures and listings". With Figures being this secondary thought, it's no surprise we see very little thought put towards the colour palettes used. The colours always seem quite arbitrary, when colour should be seen as a key element in a Figure. Colour can denote many things (red=bad, green=good, etc.), but poor colour choice can introduce unconscious bias and or accessibility issues for people with colour-blindness. This poster illustrates the simple building blocks to create any Figure in SAS® while emphasising a more conscious approach to Figure development.

INTRODUCTION

Figures (also known as graphs or visualizations), play an important role in communicating clinical trial data. They offer a means to convey what can be quite complex information in a visually intuitive manner, enabling anyone to grasp critical insights effectively. Despite their potential for better insight and communication, figures are often underutilised and undervalued, leading to a hesitancy among SAS® programmers to fully incorporate them into their skillset.

This paper aims to shed light on the simple building blocks required to create figures using SAS®. By demystifying the process and offering practical guidance, we strive to empower SAS® programmers to adopt a more conscious and inclusive approach to figure development. Furthermore, we will highlight the significance of thoughtful colour palette selection, which through the use of some comparisons we will illustrate the effect colour vision deficiency can have on a figure. This will hopefully greatly enhance figure clarity and ensure accessibility for all users.
Through this exploration, we hope to bring a renewed appreciation for the impact figures can have in clinical trial data presentations and encourage a shift towards their more frequent and effective usage. By embracing figures as fundamental tools in data communication, we can elevate their role and facilitate better-informed decision-making in clinical trials.

REASONS FOR AVOIDING FIGURES

Despite the benefits figures have in communicating clinical trial data, SAS® programmers and other professionals in the industry often prefer the use of tables to illustrate their findings, as the complexity that figures have associated with their production and understanding is perceived to be daunting. This section looks at the common reasons behind these views and how we can move past these barriers.

REASON 1: PERCEIVED COMPLEXITY

One of the primary reasons we may find SAS® programmers avoiding figures is their perception of the complexity of figures. The assumption is made that figure development requires some advanced technical skills and this leads to a feeling of intimidation when having to deal with the visualisation of the data. This lack of confidence can lead to the belief that figures are best left to “the experts”, causing some SAS® programmers to shy away from figure work.

REASON 2: UNCERTAINTY IN DATA INTERPRETATION

It isn’t just programmatic skills that make a competent figure SAS® programmer, it is also an understanding of how this information is best communicated. The fear of misinterpreting or misunderstanding the data can lead to the same issues as with the previous point, a reluctance to contribute to figure work. For example, a long table (with many variables) may be a better way to display data than an overly complicated figure. Inversely, the use of a pie chart as a figure could display the data in a way where you have oversimplified what you’re trying to display. [1]

REASON 3: LACK OF TRAINING AND KNOWLEDGE

A SAS® programmer may be confident and skilled in other areas of their work, but may not have received proficient formal training or guidance on figure development. The lack of adequate resources or knowledge in this domain could deter them from delving into figure creation and cause uncertainty about where to begin.

REASON 4: MISCONCEPTIONS ABOUT AUDIENCE PREFERENCE

These reservations have fed their way down the line, causing a preference to be made before data has even reached the SAS® programmers, that tabular data (in the form of tables and listings) should be presented over figures. Clients

may believe that tables are better at conveying critical information than figures. This misconception can influence the decision to forgo figure development in favour of tables.

BASIC FIGURE CREATION

With the reasons covered in the previous section in mind, it now becomes clear that covering the basics of figure creation (with examples) would be rather beneficial to many SAS® programmers. A more in-depth perspective can be found on SAS®s documentation website [2].

Creating these figures in SAS® can be achieved by using multiple methods, such as the older lesser used PROC GPLOT which relies on annotations, or with the more modern Graphic Template Language (GTL), which we will be looking at.

GRAPHIC LANGUAGE TEMPLATE

GTL is a collection of procedures i.e., PROCs, we will use a combination of PROC TEMPLATE and PROC SGRENDER however there are other procedures available under GTL (PROC SGPLOT, PROC SGBAR, etc.). We will illustrate the use of the GTL procedures by creating some of the more commonly seen figure types.

PROC TEMPLATE

Within a PROC TEMPLATE procedure, there will be 4 sections to the figure structure.

```
proc template;
   define statgraph template_name;                                    1
      begingraph;                                                     2
         layout overlay;                                              3
            plot_type x = xvariable y = yvariable;                    4
         endlayout;
      endgraph;
   end;
run;
```

Figure 1: Example code of a PROC TEMPLATE procedure, labelled numerically.

1. **DEFINE STATGRAPH** – This is required to open a definition block for defining and naming a graphics template. The name should be a clear reflection of the figure (shown in Figure 1 as *template_name*).
2. **BEGINGRAPH** – This defines the outermost container for a graph template that is defined with GTL-statements. All template definitions in the Graphics Template Language must start with a BEGINGRAPH statement and end with an ENDGRAPH statement.
3. **LAYOUT** – Layout blocks always begin with the LAYOUT keyword followed by a keyword indicating the purpose of the layout. All layout blocks end with an ENDLAYOUT statement. These statements function like do/end blocks in SAS®.
4. **PLOT** – The Plot statement is where the desired plot type will be defined, this could be a SERIESPLOT, SCATTERPLOT, BOXPLOT, etc. (these would replace the *plot_type* in Figure 1).

Each of the sections in the PROC TEMPLATE have options or statements that can be used to further customise the figure to create the desired output. A basic output can be generated with no to minimal use of these options or statements, so apart from ones that may be essential to producing an acceptable output, these will be covered in more depth in the next section.

PROC SGRENDER

To use the newly made template to output a figure, a PROC SGRENDER procedure is used. This takes both the template and the data we wish to use to create the figure. An example can be seen in Figure 2.

```
proc sgrender

data = figure_data

template = template_name;

run;
```

Figure 2: Example code of a PROC SGRENDER procedure.

First, the data that is to be used for the figure is defined (shown as *figure_data*). Then the template used for the figure is defined (shown as *template_name*). These together will generate our figure in the output window.

PLOT TYPES

Now we have the basic layout of the code covered we will explore its implementation in the creation of certain types of figures. Commonly used and fairly simple figures have been selected, such as series plots, scatter plots and bar charts.

SERIES PLOT

A series plot is a type of graph that displays data points connected by lines, typically used to visualise trends or patterns in sequential data. The data used for this example is the pre-packaged SAS® dataset detailing failures from the SASHELP library within SAS®. It is a dataset with the count of causes of failure over five days across three failure causes.

```
proc template;
  define statgraph series_template;

    begingraph;

      layout overlay;

1     seriesplot x = day y = count / group = cause
                                     name = "series1";

2     discretelegend "series1" / title = "Cause of Failure:";

      endlayout;

    endgraph;

  end;
run;

proc sgrender
  data = series_plot_data
  template = series_template;
run;
```

Figure 3: Example code for a Series Plot, with notable sections highlighted.

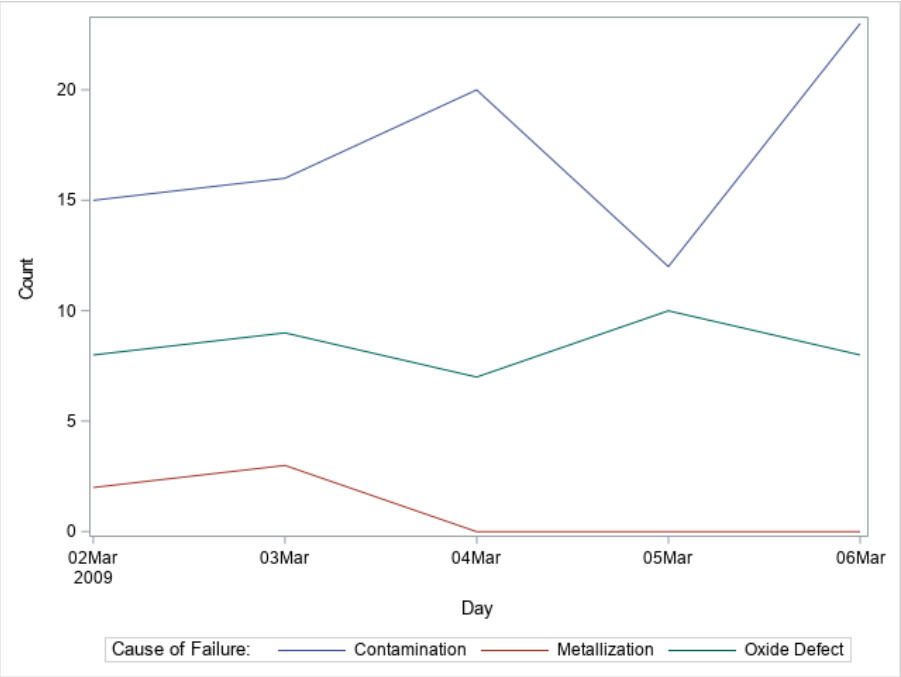

Figure 4: The Series Plot generated from the provided code and data.

The code in Figure 3 is very similar to the example code we initially saw in Figure 1 and Figure 2, with minor modifications which are highlighted. The highlighted section marked as “1” is as follows:

- (*seriesplot*) - specified that we will be using a series plot.
- (*x = visit y = val*) - specified the x and y axes.
- (/) - then it’s opened the options for that portion.
- (*group = trt*) - grouped the plot by the treatment variable, *trt*.
- (*name = “series1”*) - tagged the plot by the name, “series1”.

The highlighted section marked as “2” has modifications:

- (*discretelegend “series1”*) - produces a legend for the plot tagged “series1”.
- (/) - then it’s opened the options for that portion.
- (*title = “Treatment:”*) – gives a title to the legend.

SCATTER PLOT

A scatter plot is a type of graph that displays individual data points on a two-dimensional plane, often used to show the relationship between two continuous variables. The data used for this example is the pre-packaged SAS® dataset detailing car information from the SASHELP library within SAS®. It is a dataset listing models of car, and their observed miles per gallon in a city (MPG_CITY), as well as their horsepower (HORSEPOWER), and body type (TYPE).

```
proc template;
  define statgraph scatter_template;

    begingraph;

      layout overlay;

        scatterplot x = horsepower y = mpg_city /
                                   group = type
                                   name = "scatter1"
                                   markerattrs = (symbol = circlefilled);

        discretelegend "scatter1" / title = "Car Body Type:";

      endlayout;

    endgraph;

  end;
run;

proc sgrender
  data = scatter_plot_data
  template = scatter_template;
run;
```

Figure 5: Example code for a Scatter Plot, with notable sections highlighted.

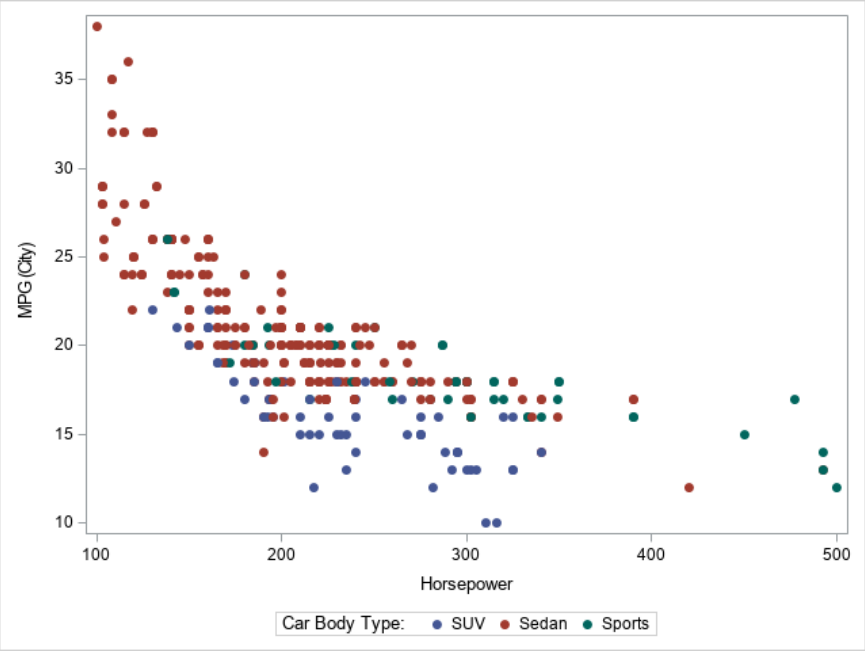

Figure 6: The Scatter Plot generated from the provided code and data.

The code in Figure 5 is again, very similar to the example code we initially saw in Figure 1 and Figure 2, the additions/modifications are highlighted. We will look at the modifications made that weren't previously covered.

- (*scatterplot*) - specified that we will be using a scatter plot.
- (*markerattrs*) - opens the attributes for the markers used on the figure.
- *(symbol = circlefilled*) - swaps the symbol used for the markers out for a filled circle.

BOX PLOT

A box plot, also known as a box-and-whisker plot, is a type of graph showing the median, quartiles, and potential outliers in a dataset. They are commonly used for comparing the spread and central tendency of data across different categories or groups. The data used for this example is the pre-packaged SAS® dataset detailing car information from the SASHELP library within SAS®. It is a dataset listing the make of car, and their observed miles per gallon in a city (MPG_CITY), as well as their body type (TYPE).

```
proc template;
  define statgraph box_template;

    begingraph;

      layout overlay /

    1  xaxisopts = (offsetmin = 0.1 offsetmax = 0.1);

    2  boxplot y = mpg_city x = type / datalabel = make
                                       spread = true;

      endlayout;

    endgraph;

  end;
run;

proc sgrender
  data = sashelp.cars
  template = box_template;
  label type = "Vehicle Type";   3
run;
```

Figure 7: Example code for the Box Plot, with notable sections highlighted.

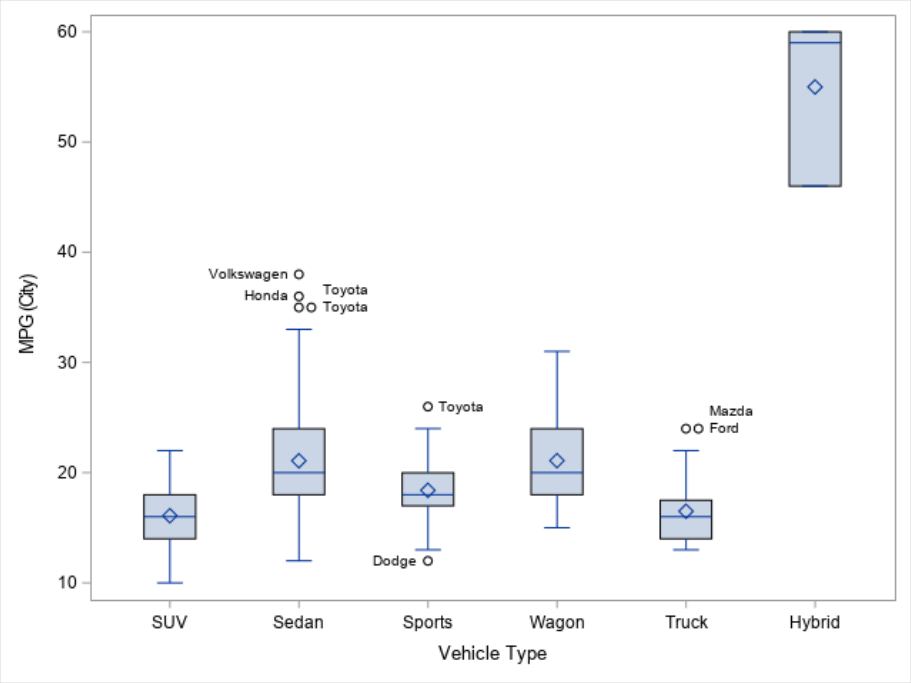

Figure 8: The Box Plot generated from the provided code and data.

The code in Figure 7 is again, very similar to the example code we initially saw in Figure 1 and Figure 2, with any modifications highlighted. The highlighted section marked as “1” is as follows:

- (*xaxisopts*) – opens options for the x-axis.
- (*offsetmin = 0.1*) – this creates some space at lower end of the axis selected, in this case it reserves 10% of the space on the left of the graph.
- (*offsetmax = 0.1*) – this creates some space at upper end of the axis selected, in this case it reserves 10% of the space on the right of the graph.

The highlighted section marked as “2” has modifications:

- (*boxplot*) - specified that we will be using a box plot.
- (*datalabel = make*) - the outliers in the output will be denoted by their make.
- (*spread = true*) - specifies that outliers with the same value are spread out to avoid overlap.

The highlighted section marked as “3” has modifications:

- (*label type = “Vehicle Type”*) – Assigns the label “Vehicle Type” to the variable TYPE, so the output shows this in place of “type”.

ADVANCED TECHNIQUES

While covering basics of GTL programming we utilised some of the modifications we can implement to enhance the output we want to produce. We will now build on these to further customise our figures but note that we will not cover every eventuality as these options are quite extensive.

BEGINGRAPH OPTIONS

The format for adding these options is shown in Figure 9.

```
begingraph / OPTION1
             OPTION2 ;
```

Figure 9: Example code for BEGINGRAPH options.

As with each section, there are a range of options you could use, but there are few for this section that you will use regularly aside from defining the output border or selecting the dimensions for the figure.
To define a border, you use *border = true* or if a border is not wanted *border = false*
To select the dimensions of the figure you use *designheight = XXXpx*
designwidth = ZZZpx
where *XXX* is the desired pixel height and *ZZZ* is the desired pixel width.

LAYOUT OPTIONS

The format for adding options here is the same, however there is the addition of determining how you want your output displayed, and if there is more than one option, how they interact together. This is shown in Figure 10.

```
layout LAYOUT TYPE / OPTION1
                     OPTION2 ;
```

Figure 10: Example code for LAYOUT options.

First, we will look at the layout type. These will decide how the plots you create will be displayed. Although there are differences between the options LATTICE and GRIDDED, as well as DATAPANEL and DATALATTICE, these present similarly so will be shown as the same in Figure 12.

OVERLAY	One plot per page
LATTICE GRIDDED	Multiple Grouped Plots per page independent of data values
DATAPANEL DATALATTICE	Multiple Grouped Plots per page with shared axes

Figure 11: On left is the LAYOUT OPTION code and on the right the description of each option.

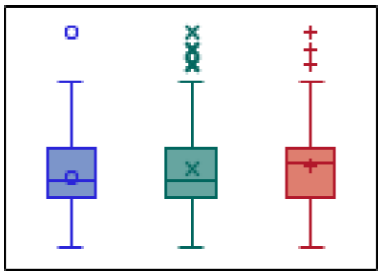

Figure 12a: Example of an OVERLAY layout option. [3]

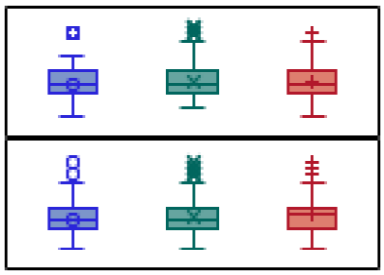

Figure 12b: Example of a GRIDDED layout option. [3]

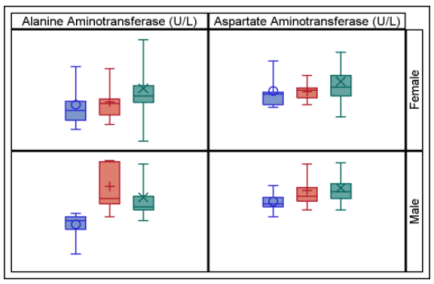

Figure 12c: Example of a DATALATTICE layout option. [3]

AXIS OPTIONS

Next, we will look at some of the options available within the layout section. While looking at the boxplots earlier we covered the LAYOUT option *xaxisopts = (OPTION)*, there is also the y-axis version *yaxisopts = (OPTION)*. These axis options will introduce a lot more flexibility into our figures.

`GRIDDISPLAY = XX`	Main inputs for `XX` here is `ON` or `OFF`. This will specify whether the axis lines are displayed.
`GRIDATTRS = (XX = YY)`	Controls the style of the axis lines, if displayed. This follows the general rule for attributes, which will be looked at momentarily.
`LABEL = "XX"`	Specifies the axis label. Can either a string or work dynamically.
`LABELATTRS = (XX = YY)`	Specifies the colour and font attributes of the axis label. Follows the general rule for attributes, which will be looked at after.
`TYPE = XX`	Specifies the type of axis wanted. Default is `AUTO`, which automatically determines the axis type (best practice is to select the axis type manually). Most of the time options `LINEAR` which uses the linear axis, and `LOG` which uses the log axis are used. More are available.

Figure 13: On left is the AXIS OPTION code and on the right the description of each option.

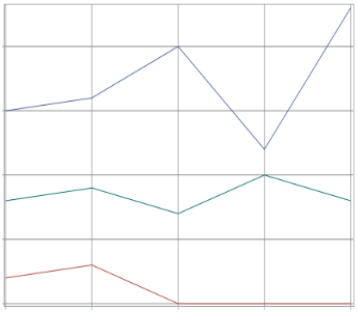

Figure 14a: Example output with `GRIDDISPLAY = ON`.

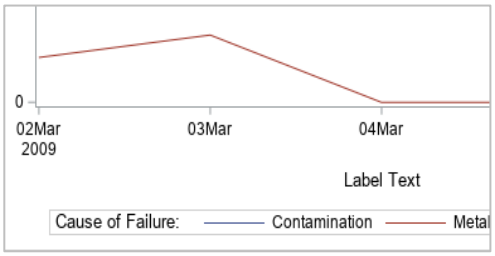

Figure 14b: Example output with `LABEL = "Label Text"`.

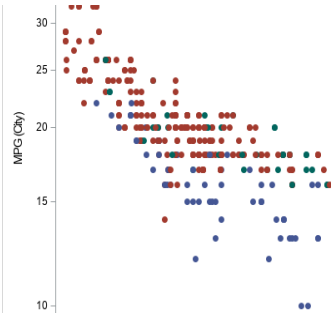

Figure 14c: Example output with `TYPE = LOG`.

STYLE OPTIONS

For attribute options, it will depend on the category it would fall into. There are several but we will look at the ones covered. GRIDATTRS falls into Line Options, we can specify the lines colour, pattern, and thickness.

`LINEATTRS = (COLOR = XX)`	Specifies colour. `XX` can be given as a valid colour name, such as RED, or a colour code, such as CXFF0000 or #FF0000.
`LINEATTRS = (PATTERN = XX)`	Specifies the line pattern. `XX` can be given as the pattern number or as the pattern name (`Solid`, `ShortDash`, `LongDash`, etc.)
`LINEATTRS = (THICKNESS = XX)`	Specifies the thickness of the line. `XX` must be given as the desired thickness and the associated dimension (`0.2in`, `3mm`, `10pct`, etc.)

Figure 15: On left is the STYLE OPTION code and on the right the description of each option.

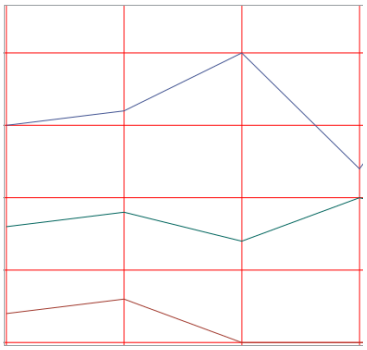

Figure 16a: Example of output with `GRIDATTRS = (COLOR = RED).`

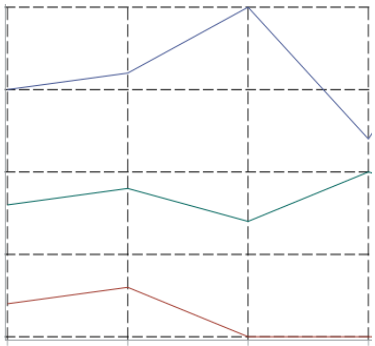

Figure 16b: Example of output with `GRIDATTRS = (PATTERN = 4).`

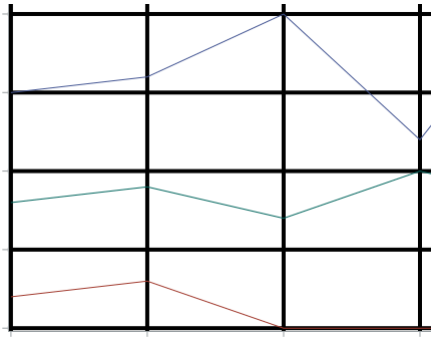

Figure 16c: Example of output with `GRIDATTRS = (THICKNESS = 1pct).`

Whereas LABELATTRS fall into the category of Text Options, where we can specify the colour, font family, size, style, and weight of the text.

`TEXTATTRS = (COLOR = XX)`	Specifies the colour. `XX` can be given as a valid colour name, such as RED, or a colour code, such as CXFF0000 or #FF0000.
`TEXTATTRS = (FAMILY = "XX")`	Specifies the font used. `XX` can be any of the fonts available in SAS® (`Arial, Helvetica,` etc.).
`TEXTATTRS = (SIZE = XX)`	Specifies the font size of the text (`14`).
`TEXTATTRS = (STYLE = XX)`	Specifies the style of the text (`NORMAL` or `ITALIC`).
`TEXTATTRS = (WEIGHT = XX)`	Specifies the style of the text (`NORMAL` or `BOLD`).

Figure 17: On left is the STYLE OPTION code and on the right the description of each option.

Figure 18a: Example of output with `LABELATTRS = (COLOR = RED FAMILY = "HELVETICA").`

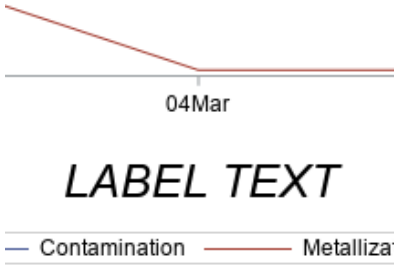

Figure 18b: Example of output with `LABELATTRS = (SIZE = 18 STYLE = ITALIC).`

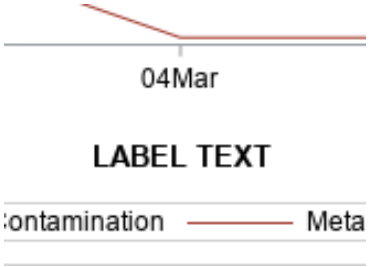

Figure 18c: Example of output with `LABELATTRS = (WEIGHT = BOLD).`

PLOT OPTIONS

As previously covered, here we can specify the plot we wish to create. As we have already looked at a few types I won't cover anymore, however there are many more types of figures you can create, depending on the data you have and what you wish to display.

Before looking at the options available, we will look at what else can be done here. First note how in Figure 19, multiple plots can be defined in the same PROC TEMPLATE. Depending on how the PROC TEMPLATE is set up, these can be overlayed on the same axes or can be put on separate axes next to each other, as seen in Figure 12.

```
PLOT_TYPE1 x = X_VARIABLE y = Y_VARIABLE / OPTION1
                                           OPTION2;

PLOT_TYPE2 x = X_VARIABLE y = Y_VARIABLE / OPTION3
                                           OPTION4;
```

Figure 19: Example code showing how multiple plots can be in same PROC TEMPLATE.

Next, we will look at the options available within the plot section. While looking at each of the earlier plots we covered some PLOT options.

```
PLOT_TYPE x = X_VARIABLE y = Y_VARIABLE / OPTION1
                                          OPTION2;
```

Figure 20: Example code for plot options.

In the Series Plot we saw the use of *GROUP = VARIABLE* which groups the output by each of the unique instances in *VARIABLE*, and *NAME = "PLOT_TAG"* which tags the plot by the specific name, so it can be called within the PROC TEMPLATE. Another option that would be useful for a Series Plot would be *DISPLAY = (MARKERS)*, which would also display the datapoints used and not just the line itself.

In the Scatter Plot we saw the use of *markerattrs = (symbol = circlefilled)*, this changed the preset symbol for the datapoints to the selected filled circles.
In the Box Plot we saw the use of *datalabel = make* which gives the outliers shown in the output a label, that in this case will be denoted by their make, and *spread = true* which specifies that outliers with the same value are spread out to avoid overlap.

These are great examples of the plot options available for each type of plot. Although the amount of customisation can be daunting, this is a good starting point and can be built on.

TITLE/FOOTNOTE OPTIONS
A title and footnote can also easily be added to the output figure. These statements must be placed in the BEGINGRAPH block.
To showcase these, we will modify the series plot code we used in Figure 3.

```
proc template;
  define statgraph series_template;

    begingraph;

  1 entrytitle "Figure Title";

  2 entryfootnote "First Footnote";
    entryfootnote "Second Footnote";

      layout overlay;

        seriesplot x = day y = count / group = cause
                                       name = "seriesplot";

        discretelegend "seriesplot" / title = "Cause of Failure:";

      endlayout;

    endgraph;

  end;
run;
```

Figure 21: Example code for title and footnote options, with notable sections highlighted.

In Figure 21 the highlighted section marked as "1" the modifications are as follows:

- (*entrytitle "Figure Title"*) – *entrytitle* gives us the ability to select a title for the figure, *"Figure Title"* is what would become our new title.

In the highlighted section marked as "2" the modifications are as follows:

- (*entryfootnote "First Footnote"*) – *entryfootnote* gives us the ability to select a footnote for the figure, *"First Footnote"* is what would become our first footnote.
- (*entryfootnote "Second Footnote"*) – *entryfootnote* gives us the ability to select a footnote for the figure, *"Second Footnote"* is what would become our second footnote.

A figure created with these additions is shown in Figure 22.

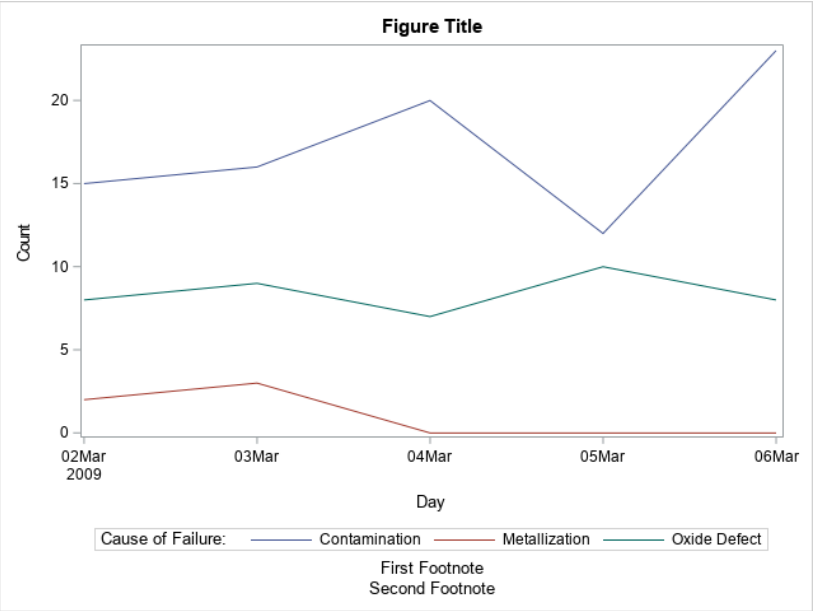

Figure 22: The series plot with the title and footnotes added to the updated code.

LEGEND OPTIONS

If desired the legend can be customised to further enhance the output figure, this is usually done to place the legend in a specific location.

To showcase this, we will modify the series plot code we used in Figure 3.

```
proc template;
  define statgraph series_template;

    begingraph;

      layout overlay;

        seriesplot x = day y = count / group = cause
                                       name = "seriesplot";

        discretelegend "seriesplot" / title = "Cause of Failure:"
                                      location = outside
                                      halign = left
                                      valign = top;

      endlayout;

    endgraph;

  end;
run;
```

Figure 23: Example code for the legend options, with notable sections highlighted.

In the highlighted section in Figure 23 the modifications are as follows:

- (*location* = *outside*) - *location* gives us the ability to select if the legend is located inside or outside of the plot itself, the second part is the option of choice (*inside* or *outside*).
- (*halign* = *left*) – *halign* gives us the ability to select where the legend will lie on the horizontal axis, the second part is the option selected (*left*, *center* or *right*).
- (*valign* = *top*) - *valign* gives us the ability to select where the legend will lie on the vertical axis, the second part is the option selected (*top*, *center* or *bottom*).

A figure created with these additions is shown in Figure 24.

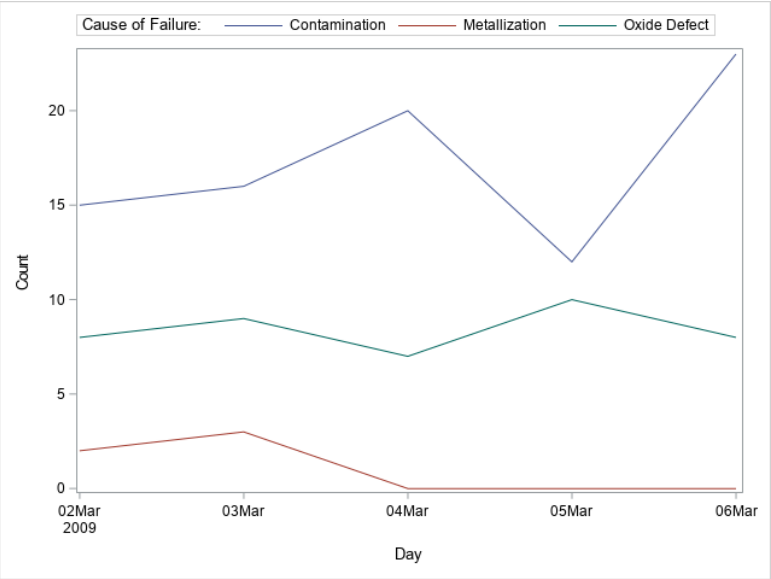

Figure 24: The series plot with the legend options added to the updated code.

ACCESSIBILITY

THE BASICS OF CVD

Colour vision deficiency (CVD), commonly known as colour-blindness, affects about 8% of men and 0.5% of women. [4] There are many types of CVD with varying degrees of effect, from issues distinguishing shades of a colour, to seeing no colour at all (although the latter is incredibly rare). 98% of these people dealing with CVD have “red-green colour-blindness”, which is actually two different types that present similarly (Deutan and Protan). [5] These individuals have trouble telling the difference between red and green. Although there are other types, this is by far the largest group we can easily improve accessibility for.

In Figure 25, on the left a colour palette is shown containing greens, reds and oranges. On the right is a simulation of what that may look to someone with protanope CVD.

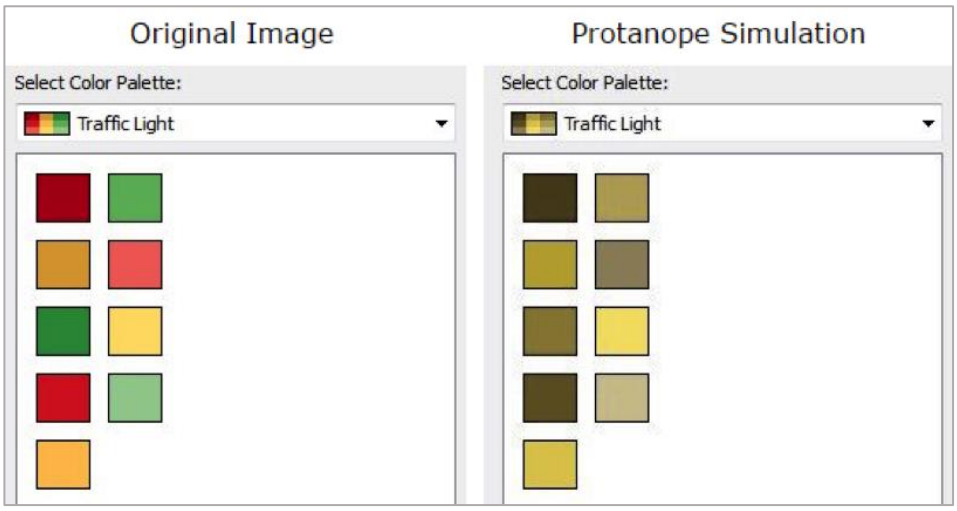

Figure 25: Comparison of colours for normal vision vs a protanope simulation. [6]

Here we see that for someone with strong protanope CVD their reds, greens, and oranges seem to blend together into shades of brown. This is a good illustration of how even the commonly used red vs green, as a depiction of bad vs good, isn’t a viable option for people with CVD.

However, when looking further at how the colours used in outputs may differ from someone without CVD we see the issue is more complex than just not mixing reds and greens.

Figure 26: Comparison of colours for normal vision vs a deuteranope simulation. [6]

In Figure 26, on the left a colour palette is showing a variety of colours, many being “mixtures”. On the right is a simulation of what that may look to someone with deuteranope CVD.
Now we see that when colours are mixed, it further complicates things. Purple (a mixture of blue and red) can look like blue to someone who has issue with red, which can cause confusion when using blue and purple together. Other problematic combinations are pink and grey or grey and brown.

INDUSTRY GUIDANCE

Considering how CVD affects a notable percentage of the population, it would be easy to assume that significant resources, guidelines, and standards would be available from industry regulators and/or leaders to ensure these people are properly accounted for, this is not however the case.

CDISC, being the main standards organization for data in the clinical trials industry has only one notable mention of its accommodations for people with CVD. Version 2 of their Study Data Tabulation Model Metadata Submission Guidelines: Human Clinical Trials [7] mentions four colours that have been selected for use when annotating CRFs. These have been tested using a CVD simulator [8] and the RGB colour codes have also been provided (RGB, an acronym of red, green, blue, refers to a system that combines these colours in various proportions to obtain any colour [9]). It then mentions *“use consistent colors and take color blindness into consideration*”, when more than four colours are required, which is a little vague. This is the extent to which CDISC accommodates CVD.

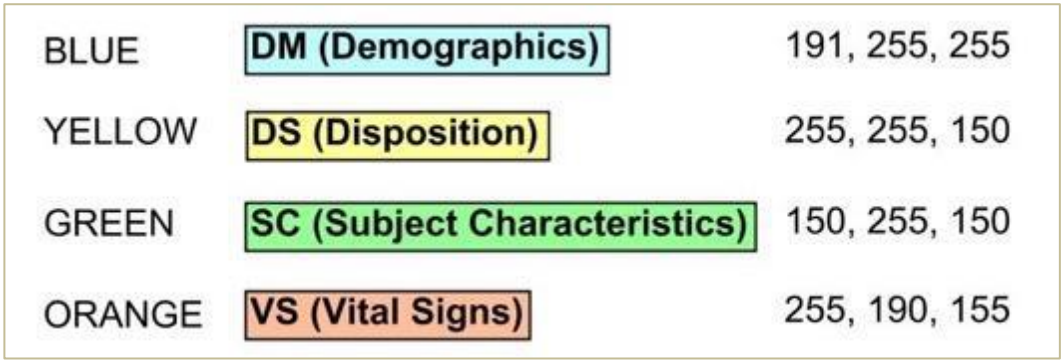

Figure 27: CDSIC CRF colour recommendations. [7]

The FDA has some general guidance on submissions in electronic format. This mostly covers the expected formats of documents to be submitted to the FDA. Colour is mentioned only to altogether discourage its use, as it can be lost when printing or photocopying. [10] [11]Thankfully there are other organisations out there that have clearer information on what can be done.

Tableau offers five tips to help design CVD friendly visualisations. Notably, colour palettes should be chosen that are CVD friendly, and other methods should be given to distinguish between data classes (such as icons or patterns as well as the colour). [6]Towards Data Science [12] offer two tips, with both being the same that Tableau had raised.

A commonly referenced resource for picking out suitable colour palettes was ColorBrewer2.0. [13] Here you have the ability to pick out different characteristics of the data you intend to display, select to only show “colorblind safe” palettes, and you are given a colour palette that is deemed suitable for your use. ColorBrewer2.0 has its limitations but is a useful resource, nonetheless.

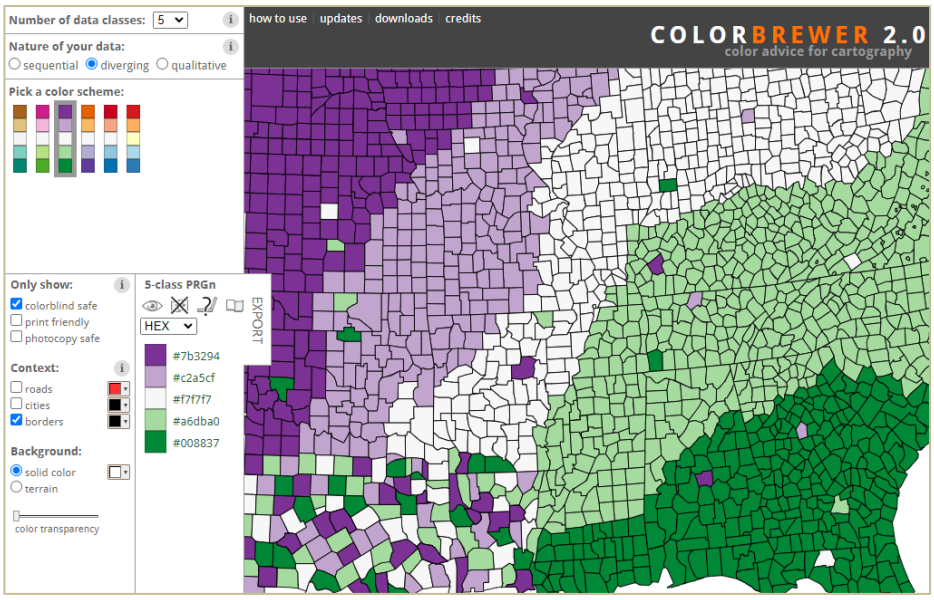

Figure 28: ColorBrewer2.0 showcase. [13]

If you already have a figure and wanted to check if it had any issues for people with CVD then resources are available for that as well, one such being color-blindness.com. Here you can use their Coblis simulator to input an image file and using different filters, decide whether it would be suitable for use or is too confusing for a person with CVD. [8]

AN EXAMPLE IN SAS®

We shall look at a scatter plot, similar to the one covered earlier. Here we will be displaying the MSRP (manufacturer’s suggested retail price) vs its MPG in the city of cars by its make.

In the data we are using we have four car makes: Audi, Jaguar, Suzuki, and Volvo. So, we shall go to ColorBrewer2.0 [13] and select our options. There are four data classes (makes), the data we are displaying is qualitative, and we need it to be usable for people with CVD (here shown as colourblind safe). With these options selected we are given the colour palette we need to move forward.

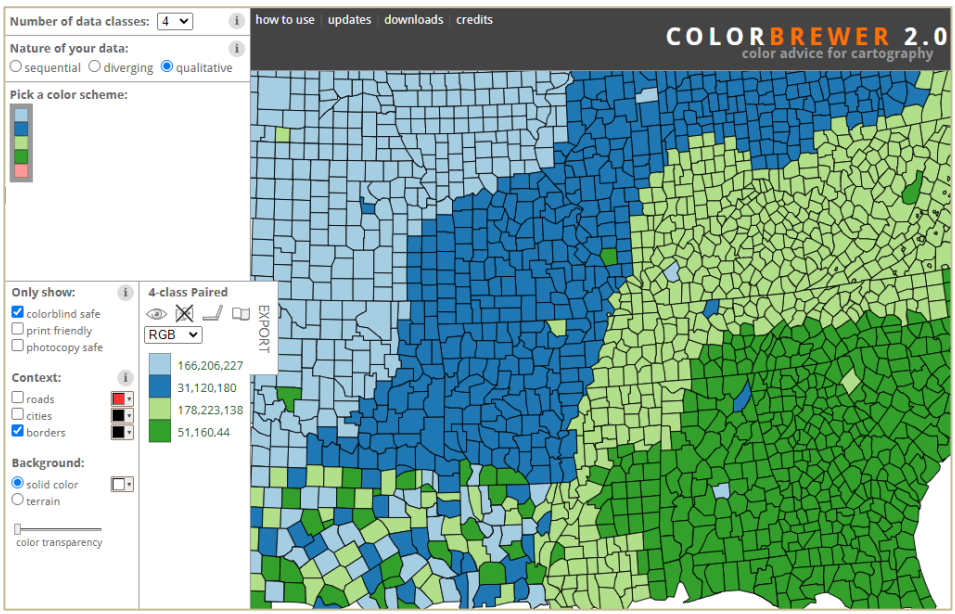

Figure 29 ColorBrewer2.0 for this specific instance. [13]

```
%colormac;                                                    1

proc template;
  define statgraph cvd_scatter_template;

    begingraph / attrpriority = none
                 datacontrastcolors=(%rgb(166,206,227)
                                     %rgb(31,120,180)          2
                                     %rgb(178,223,138)
                                     %rgb(51,160,44));

      layout overlay;

        scatterplot x = mpg_city y = msrp / group = make
                                    name = "scatter1"
                                    markerattrs = (symbol = circlefilled
                                                   size = 15px);

        discretelegend "scatter1" / title = "Make:";

      endlayout;

    endgraph;

  end;
run;

proc sgrender
  data = cvd_scatter_data
  template = cvd_scatter_template;
run;
```

Figure 30: Code used to incorporate suggested colours, with notable sections highlighted.

To use the colours we've now found, some modifications must be made to the "standard" scatter plot code. This can be seen in Figure 30.

In the highlighted section marked as "1" the modifications are as follows:

- (*%colormac*) – this macro utility was created to help SAS® users use alternatives to SAS®' inbuilt colour codes. We will be using the RGB colour codes now available.

In the highlighted section marked as "2" the modifications are as follows:

- (*attrpriority = none*) – this changes the priority list that SAS® automatically selects off and changes to NONE.
- (*datacontrastcolors*) – this allows us to change the contrast colours of the graphics element, such as lines and in this case markers.
- *(%rgb(166,206,227)*) – this macro allows us to use the RGB codes we found in ColorBrewer2.0, in SAS®.

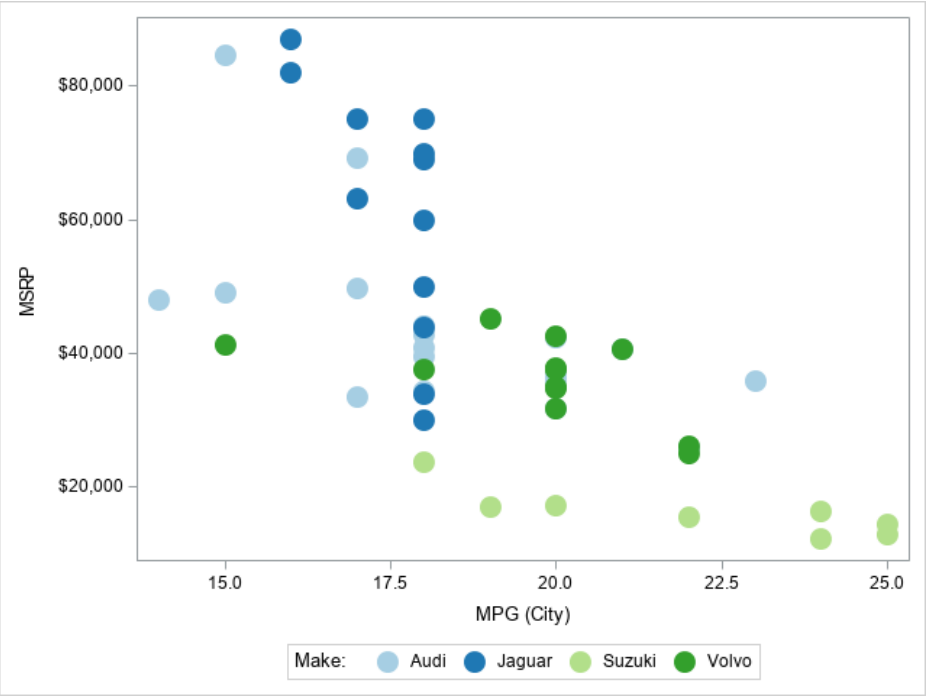

Figure 31: Figure output using the given colours and code.

The output, shown in Figure 31, from our SAS® code is effectively displaying our data while also ensuring it is accessible to people with CVD.

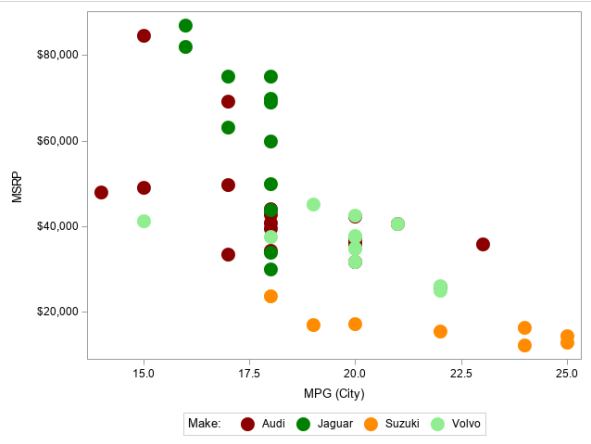

Figure 32: Figure output using colours that may confuse someone with CVD.

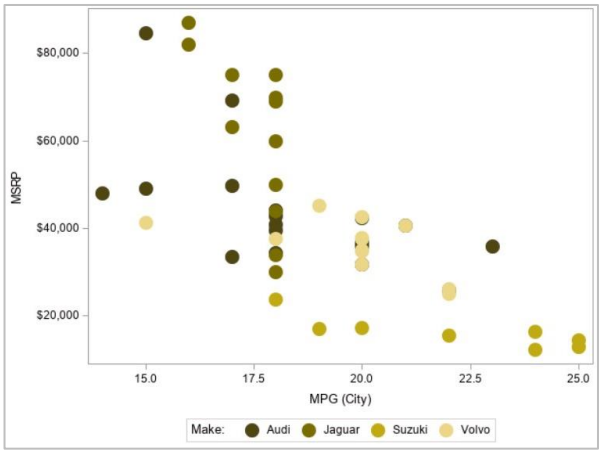

Figure 33: Simulation of how Figure 32 may look for someone with CVD. [8]

To illustrate the difference this could make, Figure 32 has the same output as Figure 31, but we've used colours that may hinder someone with CVD. To simulate its affect to wider audiences', Figure 33 shows how this may be seen with CVD.

CONCLUSION

Figures are incredible mediums for communicating complex results from data, so I hope that this paper can shed light on the ease at which these can be introduced to newcomers and their skills developed.

However, even if the use of figures becomes widespread, this will be a hollow achievement if they are inaccessible to a notable chunk of the population. I hope by covering the issues this can cause for these individuals with CVD, the ease at which this can be rectified, and the lack of current coverage by the regulatory bodies within the industry, that the industry can work together to create guidelines in the future that will make figure production and utilisation more accessible for everyone.

REFERENCES

[1] The University of Melbourne, “Why you shouldn’t use pie charts,” 30 August 2023. [Online]. Available: https://scc.ms.unimelb.edu.au/resources/data-visualisation-and-exploration/no_pie-charts.

[2] SAS®, “SAS® 9.4 Graph Template Language,” 4 August 2023. [Online]. Available: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/grstatgraph/titlepage.htm.

[3] K. Harris, “Hands-On Graph Template Language (GTL): Part A,” 28 August 2023. [Online]. Available: https://support.sas.com/resources/papers/proceedings17/0794-2017.pdf.

[4] Colour Blind Awareness, “About Colour Blindness,” 24 August 2023. [Online]. Available: https://www.colourblindawareness.org/colour-blindness/.

[5] Enchroma, “25 Facts About Color Blindness,” 22 August 2023. [Online]. Available: https://enchroma.com/blogs/beyond-color/interesting-facts-about-color-blindness.

[6] J. Shaffer, “5 tips on designing colour-blind-friendly visualizations,” 22 August 2023. [Online]. Available: https://www.tableau.com/en-gb/blog/examining-data-viz-rules-dont-use-red-green-together.

[7] CDISC, “Study Data Tabulation Model Metadata Submission Guidelines (SDTM-MSG): Human Clinical Trials,” 2021.

[8] Colblindor, “Coblis - Color Blindness Simulator,” 24 August 2023. [Online]. Available: https://www.color-blindness.com/coblis-color-blindness-simulator/.

[9] A. Zola, “TechTarget,” January 2023. [Online]. Available: https://www.techtarget.com/whatis/definition/RGB-red-green-and-blue.

[10] FDA, “Electronic Sumission File Formats and Specifications,” January 2018. [Online]. Available: https://www.fda.gov/files/medical%20devices/published/Tobacco-Electronic-Submission-File-Formats-and-Specifications.pdf.

[11] FDA, “Guidance for Industry - Providing Regulatory Submissions in Electronic Format - General Considerations,” January 1999. [Online]. Available: https://www.fda.gov/media/71200/download.

[12] C. Ferreira, “Two Simple Steps to Create Colorblind-Friendly Data Visualizations,” 2020 April 2020. [Online]. Available: https://towardsdatascience.com/two-simple-steps-to-create-colorblind-friendly-data-visualizations-2ed781a167ec.

[13] ColorBrewer2.0, “ColorBrewer2.0,” 24 August 2023. [Online]. Available: https://colorbrewer2.org/.

ACKNOWLEDGMENTS

A big thank you to my team at MAC for helping with this.

RECOMMENDED READING

https://blogs.sas.com/content/iml/2023/02/01/colorblind-safe-palettes-sas.html
https://www.lexjansen.com/nesug/nesug07/po/po11.pdf
https://www.nhs.uk/conditions/colour-vision-deficiency/
https://www.colourblindawareness.org/colour-blindness/types-of-colour-blindness/
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/grstatproc/p0edl20cvxxmm9n1i9ht3n21eict.htm
https://blogs.sas.com/content/iml/2023/01/30/tips-colorblind-safe-graphs.html

CONTACT INFORMATION

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

Kaleb Mead
MAC Clinical Research
Email: kalebmead@macplc.com
LinkedIn: https://www.linkedin.com/in/kalebmead

Brand and product names are trademarks of their respective companies.