A (very) brief introduction to wfaplot

Cian Siôn

January 2023

Contents

Installing wfaplot

To get started, let's install wfaplot from the github repository using `install_github()`. This function is part of the remotes package which can be installed from CRAN using the more familiar `install_packages()` function.

During the installation process, we may be prompted to update the package dependencies. If so, choose which packages to update or hit Enter to skip this step.

```r
install.packages("remotes", repos = "http://cran.us.r-project.org")

remotes::install_github("ciantudur/wfaplot", dependencies = TRUE)
```

Once wfaplot is installed, don't forget to load the package using `library(wfaplot)`. You will need to do this every time you use R, but you will only be required to install the package once.

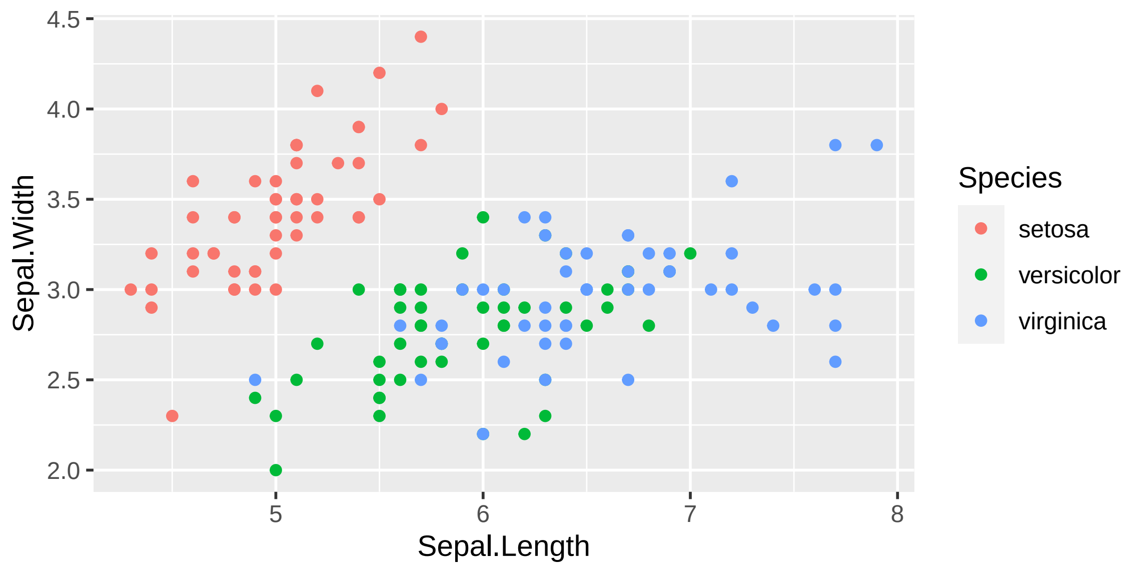
```r
library(wfaplot)
```

Using colour with wfaplot

Now that wfaplot is installed and loaded into the environment, let's begin by creating a basic scatter plot using the Iris dataset which is bundled with base R.

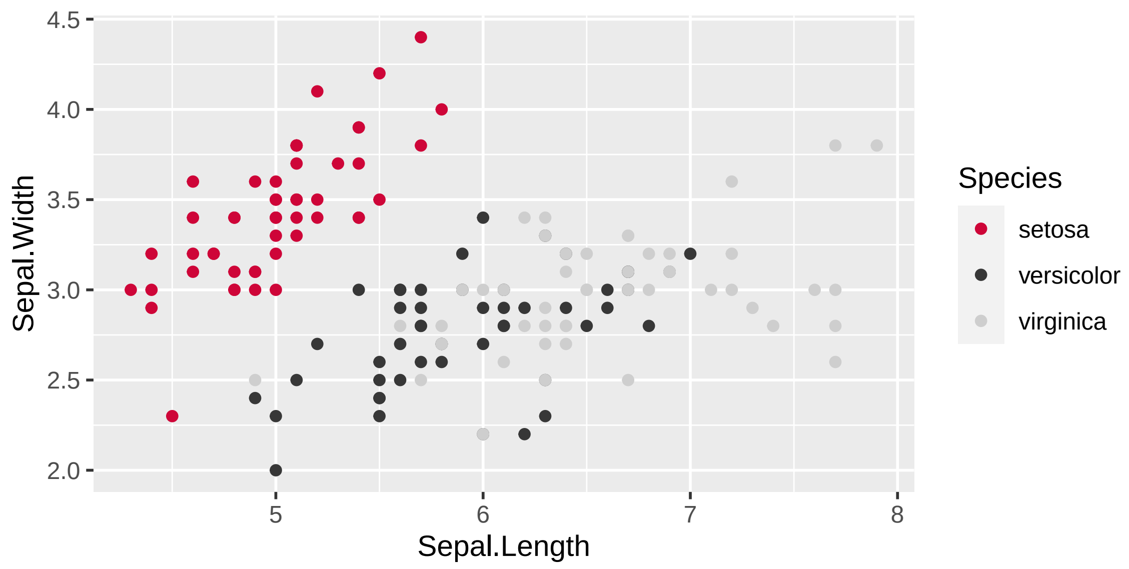
If you have never used ggplot2 before, you will need to install the package using `install_packages("ggplot2")` before running the next chunk of script.

```r
library(ggplot2)

iris_plot <- ggplot(data = iris,
               aes(x = Sepal.Length,
                   y = Sepal.Width,
                   colour = Species))+
  geom_point()

plot(iris_plot)
```



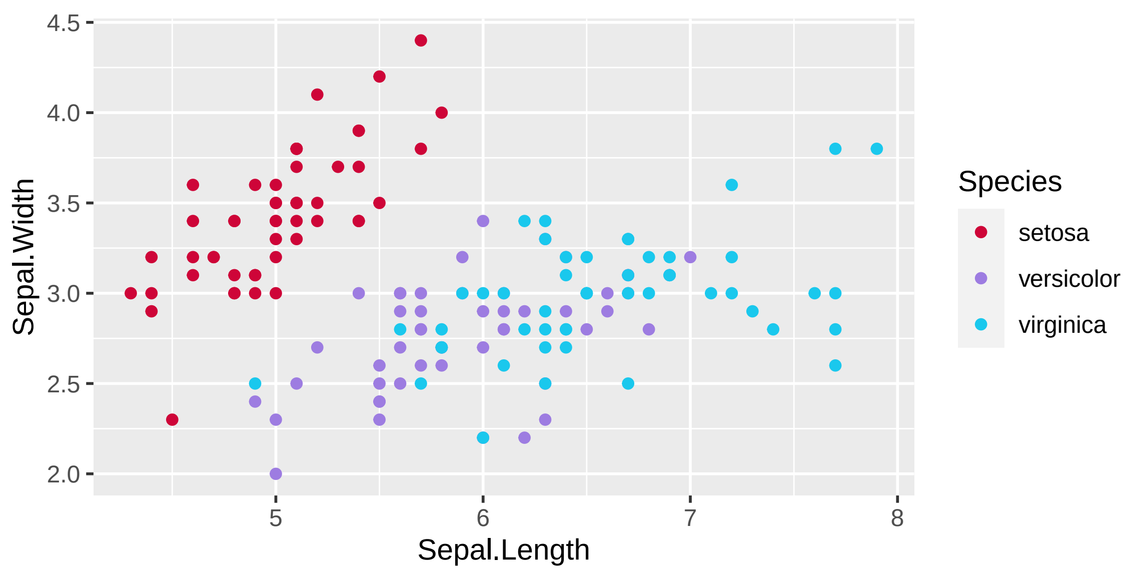
We can apply an WFA colour palette to the chart with just one additional line of code.

```r
iris_plot_wfa <- iris_plot +
  scale_colour_wfa()

plot(iris_plot_wfa)
```

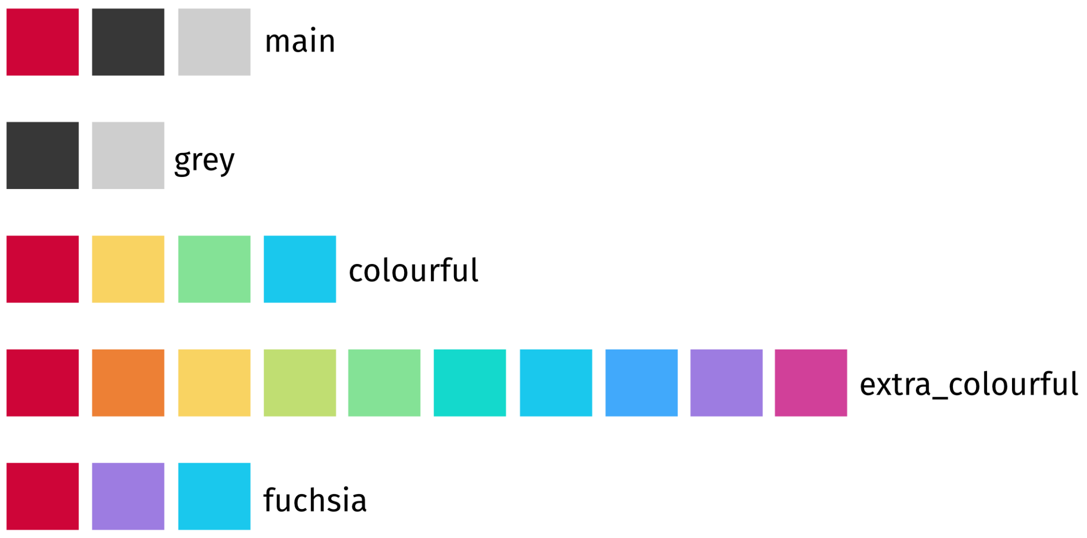


By default, `scale_color_wfa()` uses the main WFA colour palette (red, dark grey, light grey), but we can also specify different palettes.

```r
iris_plot_wfa <- iris_plot +
  scale_colour_wfa(palette = "fuchsia")

plot(iris_plot_wfa)
```



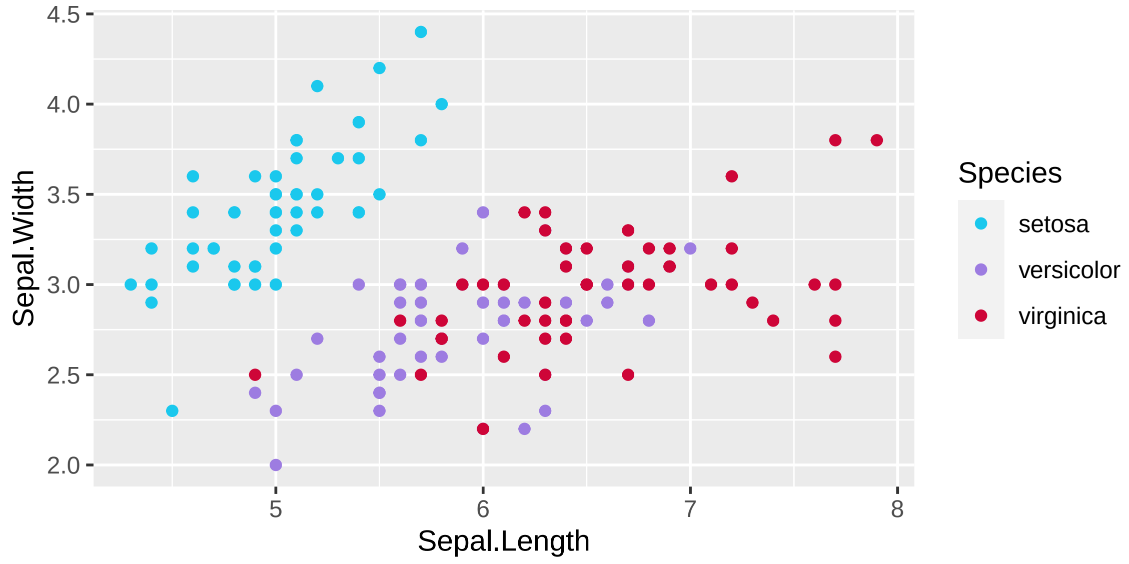
There are several colour palettes to choose from. Don't worry if there aren't enough unique colours to assign to each legend item, just choose any palette and wfaplot will automatically interpolate additional colours if required.



We can easily reverse the order of the colour palette by setting the `reverse` argument to `TRUE`.

```r
iris_plot_wfa <- iris_plot +
  scale_colour_wfa(palette = "fuchsia",
                   reverse = TRUE)
```

```r
plot(iris_plot_wfa)
```



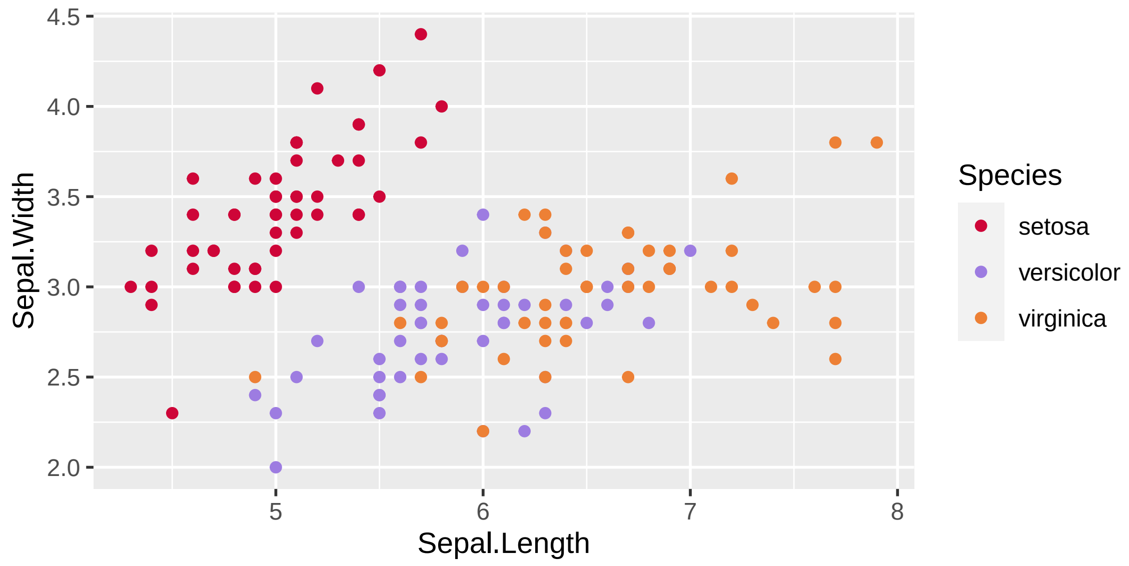
There might be times when we need more granular control over the colours used. Let's bring up a list of all WFA-defined colours using `wfa_cols()`.

```r
wfa_cols()
```

```
##         red      orange      yellow        lime       green        teal        blue 
##   "#CE0538"   "#ED8035"   "#F9D362"   "#C0DE72"   "#84E296"   "#14D9CC"   "#1AC8ED" 
##        pink      purple      indigo   dark grey  light grey 
##   "#D14099"   "#9D7CE1"   "#41A9FB"   "#373737"   "#CECECE"
```

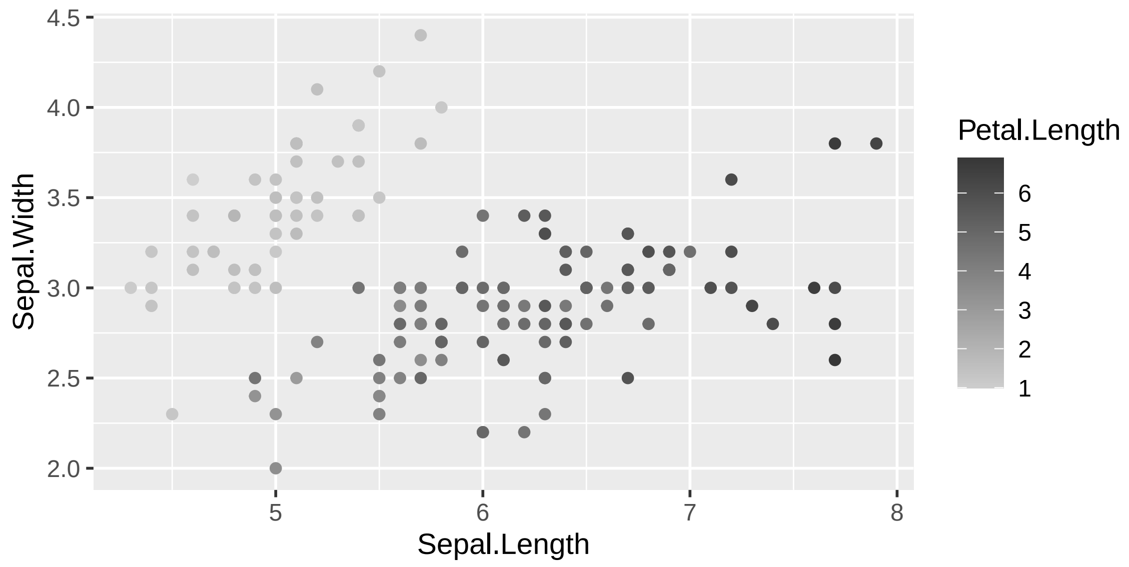
Now, let's create a manual colour scale with Red, Purple, and Orange.

```r
iris_plot_wfa <- iris_plot +
  scale_colour_manual(values = c(
    wfa_cols("red"),
    wfa_cols("purple"),
    wfa_cols("orange")))

plot(iris_plot_wfa)
```



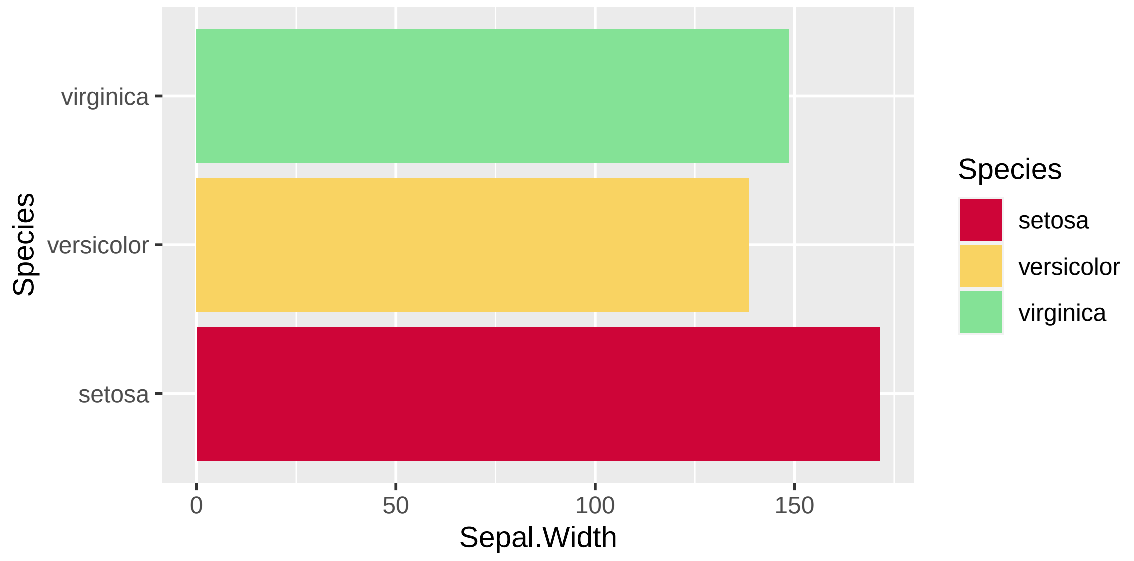
Excellent! Now suppose rather than a categorical variable, we wanted to shade the points using a continuous value. This can easily be accomplished by setting the `discrete` argument to `FALSE`.

```
iris_plot_continuous <- ggplot(data = iris,
               aes(x = Sepal.Length,
                   y = Sepal.Width,
                   colour = Petal.Length))+
  geom_point()+
  scale_colour_wfa(palette = "grey",
                   discrete = FALSE)


plot(iris_plot_continuous)
```



The same principles apply when changing the colour of fill objects (e.g. bar charts / choropleth maps). Just note that we would use `scale_fill_wfa()` rather than `scale_color_wfa()` in these instances.

```
iris_bar_wfa <- ggplot(data = iris,
               aes(x = Sepal.Width,
                   y = Species,
                   fill = Species))+
  geom_bar(stat = "identity")+
  scale_fill_wfa(palette = "colourful")


plot(iris_bar_wfa)
```

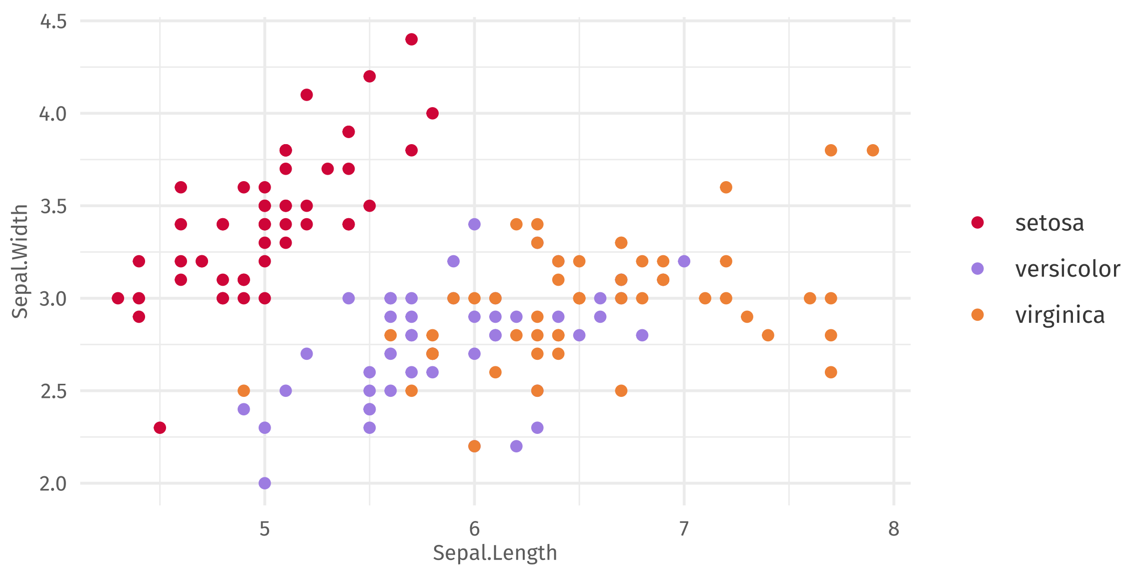


Themeing charts with wfaplot

Themeing an WFA chart couldn't be easier. Just add `theme_wfa()` to the ggplot object. This function takes no additional argument.

Just bear in mind that you will need the Fira Sans and Source Sans Pro fonts installed on your operating system to apply the theme properly. Both fonts are available to download for free from Google Fonts – just click on the links above.

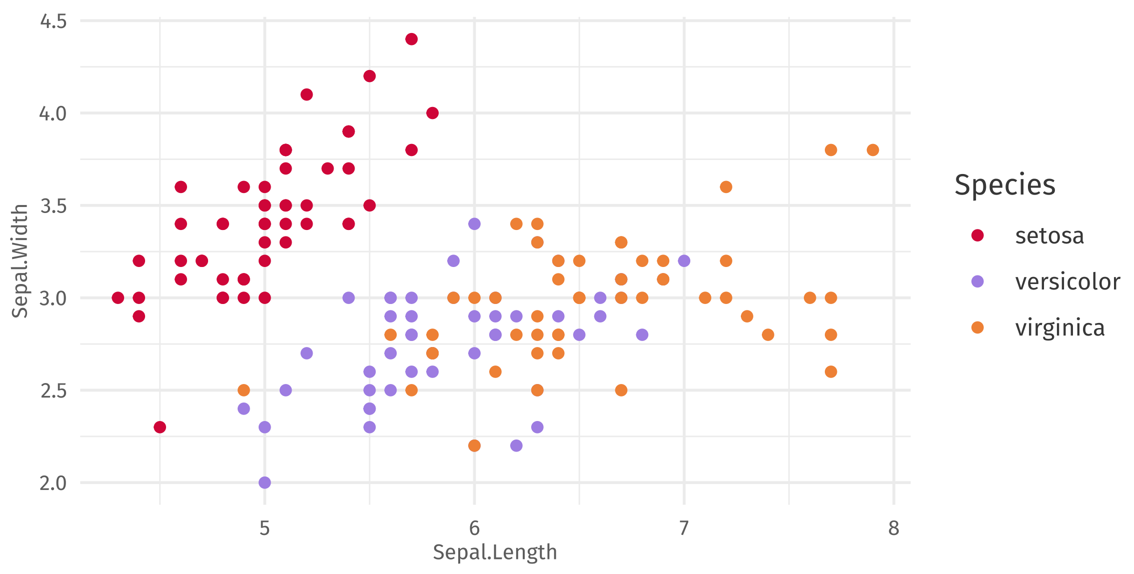
```
iris_bar_themed <- iris_plot_wfa +
  theme_wfa()

plot(iris_bar_themed)
```



The text might look a bit small right now, but don't worry, it will look fine once exported.

There might still be times when we want to tweak some of the theme elements. If so, just make sure that any additional theme customisations come **after** the `theme_wfa()` function.

```r
iris_bar_themed <- iris_bar_themed +
  theme(legend.title = element_text(family = "fira-sans",
                                    colour = wfa_cols("dark grey")))

plot(iris_bar_themed)
```



Saving charts with wfaplot

Once we're happy with the chart, we'll want to export it as an image using the `wfa_save()` function.

But before that, we must familiarise ourselves with two categories of graphic files:

- **Pixel graphics** – the most common image file extensions (png and jpeg) fall into this category. They can be opened using most operating system, the file size is relatively small, but they will start to look blurry once we zoom in enough. We can increase (or decrease) the file resolution by changing the resolution argument in `wfa_save()` e.g. `wfa_save(export_res = 600)` . The default is 400dpi (which will be more than sufficient for most scenarios).
- **Vector graphics** – these files retain their sharpness no matter how much we zoom in, but they take up much more disk space than their pixel counterparts. Common file extensions include svg and pdf. Normally, we use svg files for our reports. There is no need to specify a resolution for these type of files.

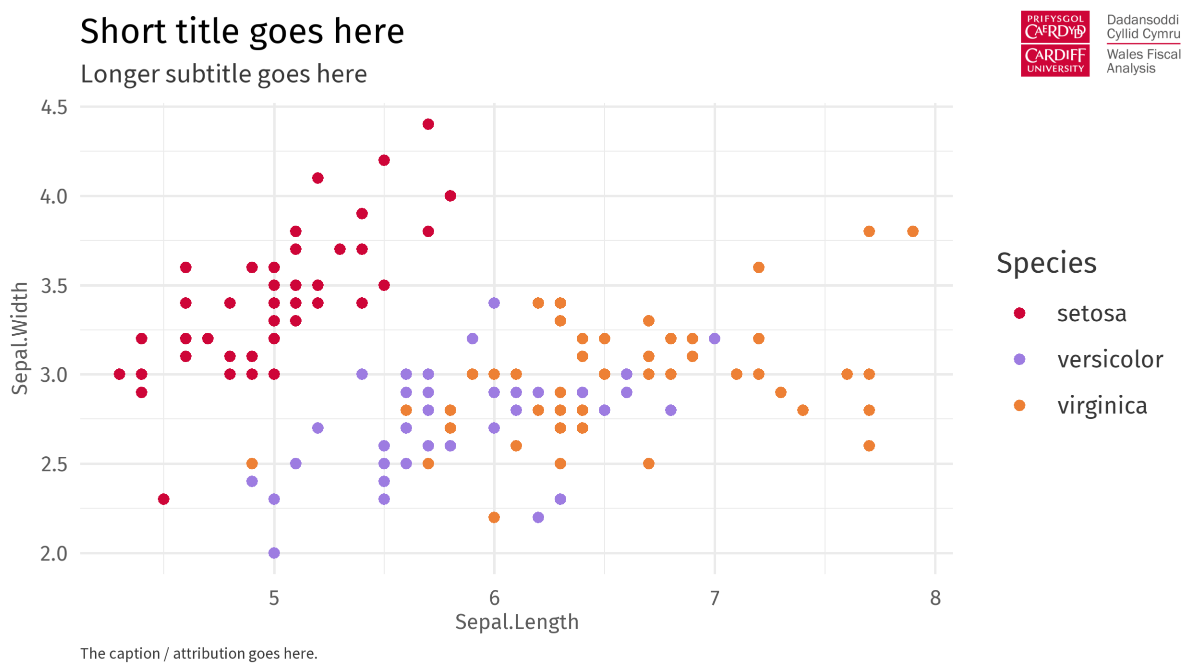
Let's assume we want to share a chart on Twitter.

First, make sure that *wfalogo.png* is placed in the root directory. You can download the logo here.

Let's add some titles thensave the chart as a png with logo. We'll use the default dimension and resolution options.

```
iris_bar_export <- iris_bar_themed +
  labs(title = "Short title goes here",
       subtitle = "Longer subtitle goes here",
       caption = "The caption / attribution goes here.")
```

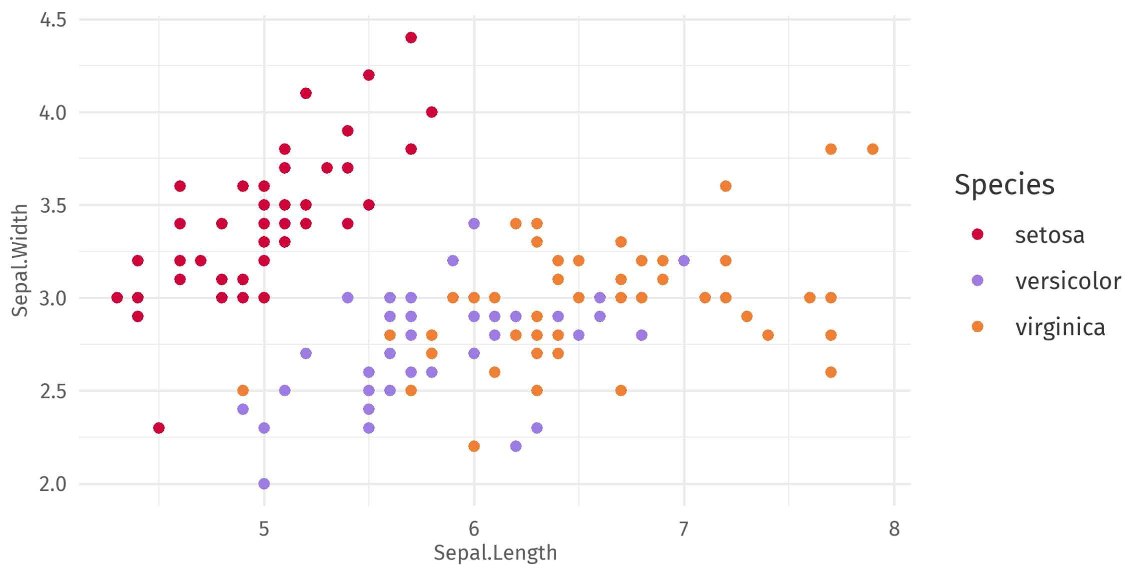
```
wfa_save(iris_bar_export, "iris.png", include_logo = TRUE)
```



And voila... the chart is ready to be shared on Twitter.

Suppose we wanted to include the chart in an WFA report instead. Let's save it as a vectorized svg file. In this case, we won't need the logo and titles.

```
wfa_save(iris_bar_export, "iris2.svg", include_logo = FALSE)
```



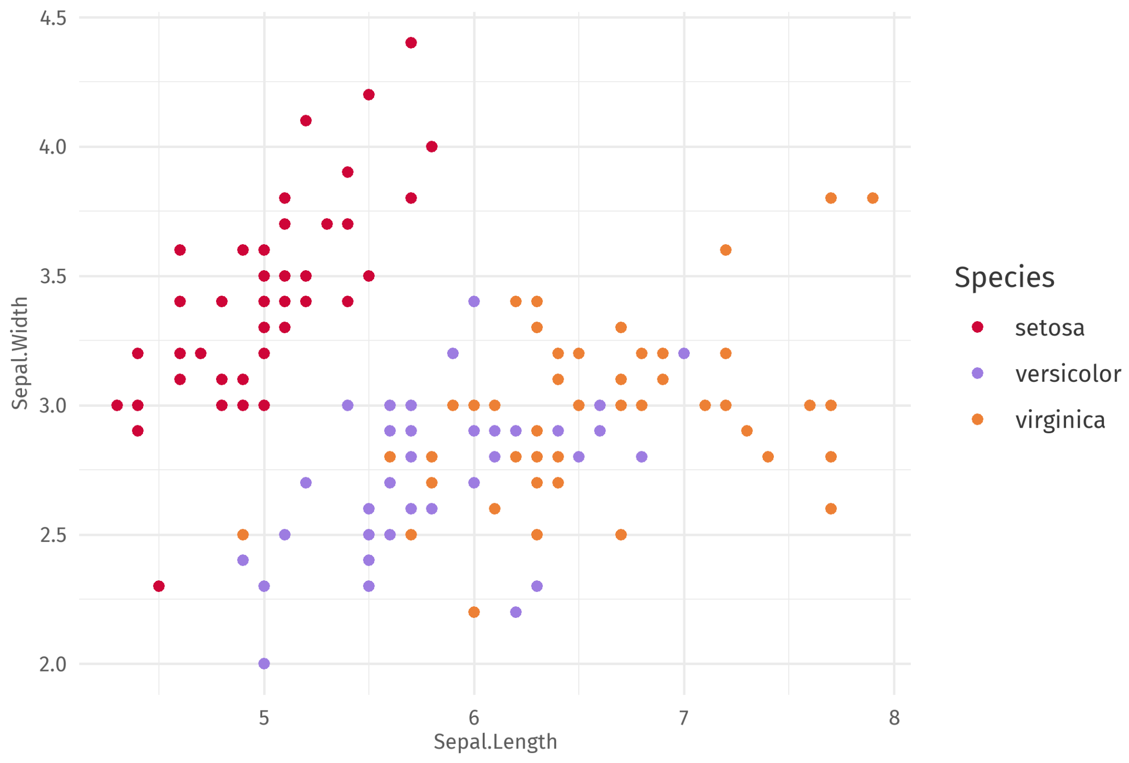
Great, although it looks a bit cramped.

The default dimensions are a good place to start, but we can always override these settings.

As a general rule, leave the width at its default setting of 6 (inches), and adjust the height from the default of 3 (e.g. if you want a taller chart, try specifying a height of 4 or 4.5).

This will take a bit of trial-and-error.

```
wfa_save(iris_bar_export, "iris_stretched.svg", include_logo = FALSE,
         export_height = 4, export_width = 6)
```



That's better!

You can always find more information about this or any of the other wfaplot functions by typing `help(wfa_save)` in the console.

Since wfaplot is built on ggplot, we can take advantage of near-infinite data visualisation options. Once you're ready to get stuck in, this is a good place to start…

PRIFYSGOL CAERDYDD CARDIFF UNIVERSITY

Dadansoddi Cyllid Cymru

Wales Fiscal Analysis