Publication-Ready Figures Using R ggplot2

Laura Logozzo and Sam Woodman

2023-10-13

Contents

Required Packages for this Tutorial

Download the following packages if you do not yet have them:

- ggplot2
- ggpubr
- ggpmisc
- dplyr (optional)
- stringr (optional)

There are some custom color palette functions for download on my (Laura's) website that are useful for plotting:

- Custom color palettes: https://lauralogozzo.github.io/Color-Palette/

Introduction

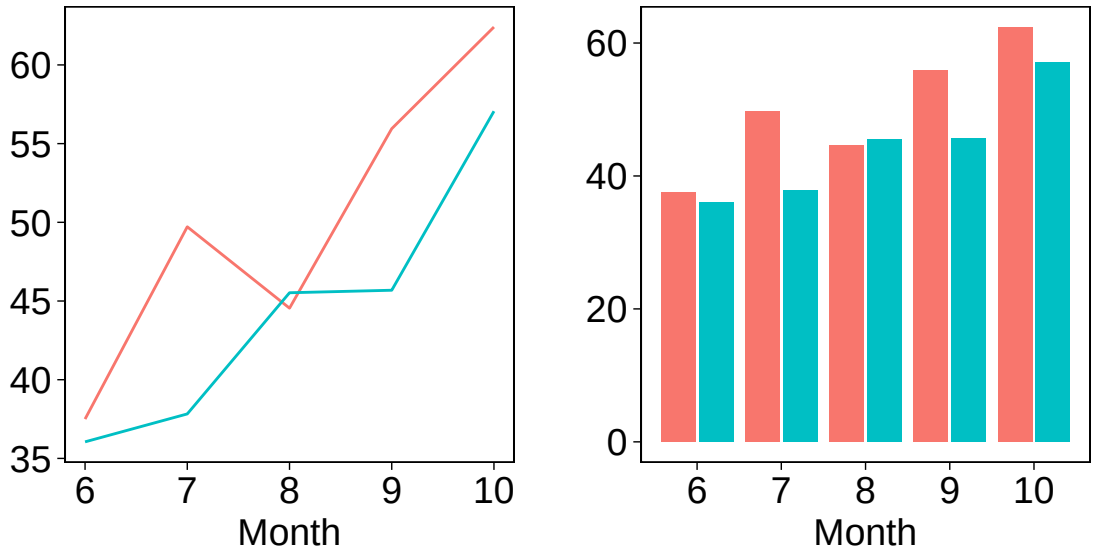
1. What are figures for

Figures exist to tell a story about your data and are often the first part of a paper that the editors and readers will examine. Therefore, figures should be **designed with intention** to concisely convey your data. Designing effective figures relies on (Rolandi et al., 2011):

- **Knowing your audience**
  - Is your audience the general public or your peers?
  - What background knowledge do they have?
- **Focus on most important information**

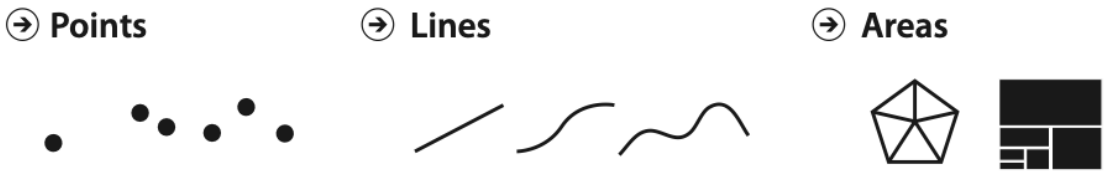
  - Identify storyline
  - Establish key message
- **Clear visual structure**
  - Easily enter the image (top left to lower right)
  - Consistent visual elements

Based on these design principles, the same datasets can be displayed several ways.



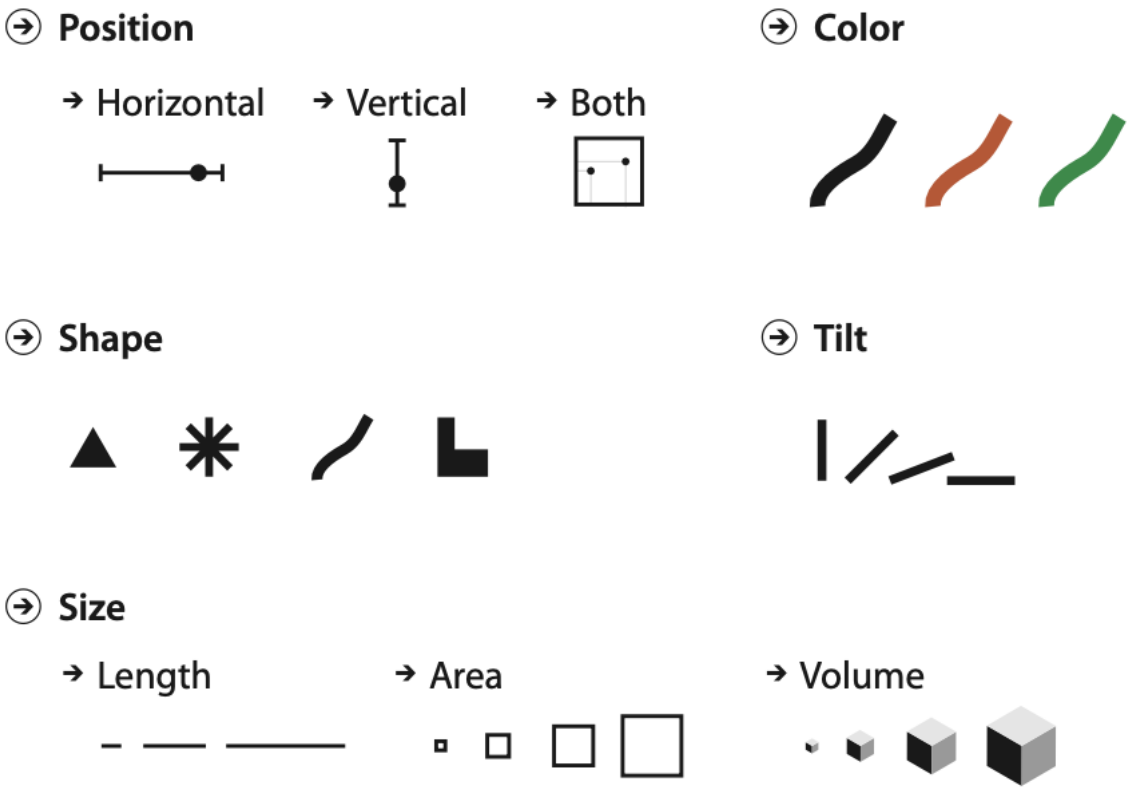
2. What makes a figure

Marks are the basic graphical element of an image. These are primitive geometric objects ranging from 0 to 3 dimensions.



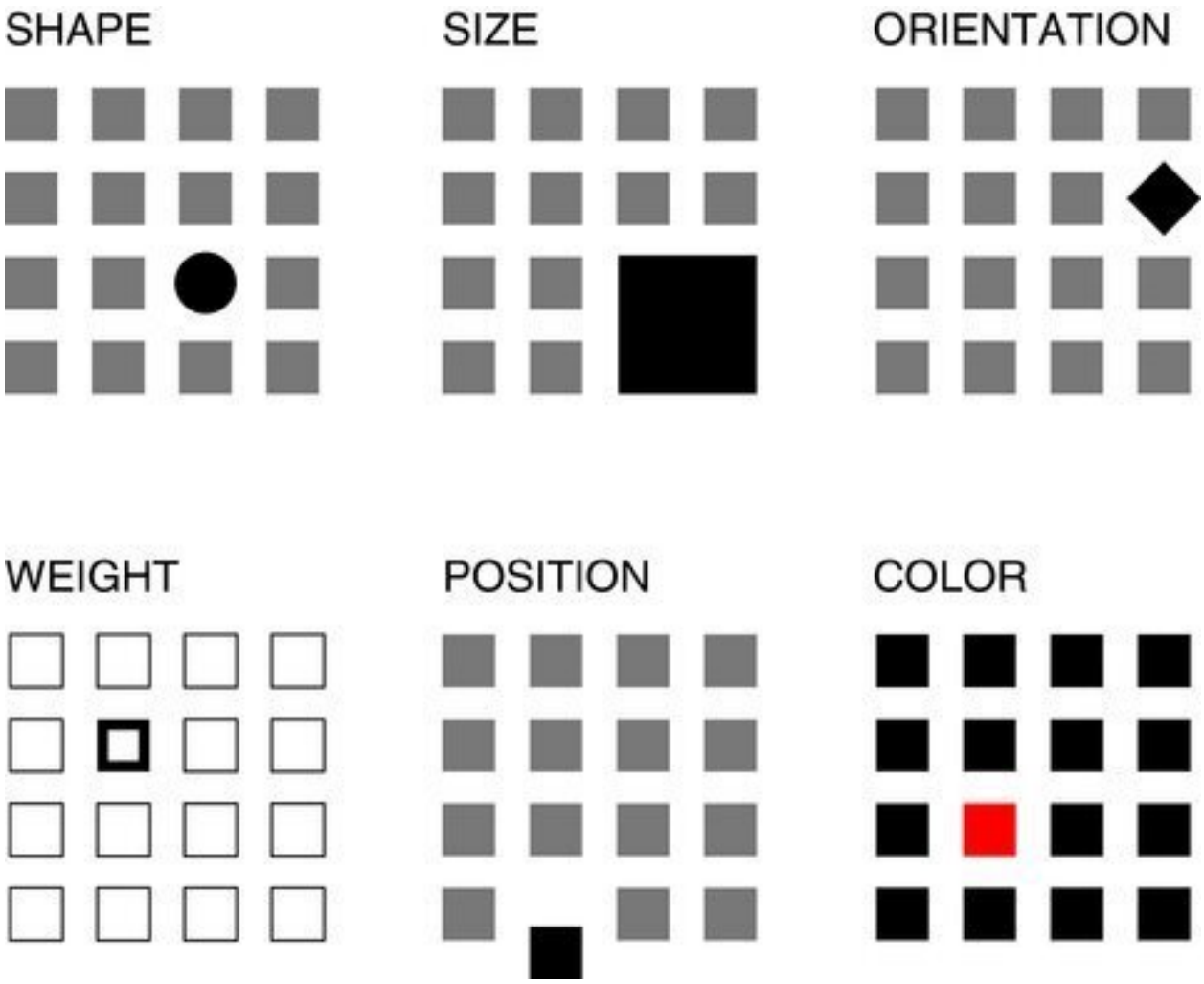
from 'Visualization Analysis and Design' by Tamara Munzner

Visual **channels** are ways to control the appearance of marks. They are independent from the dimensionality and include position, colour, shape, and size.



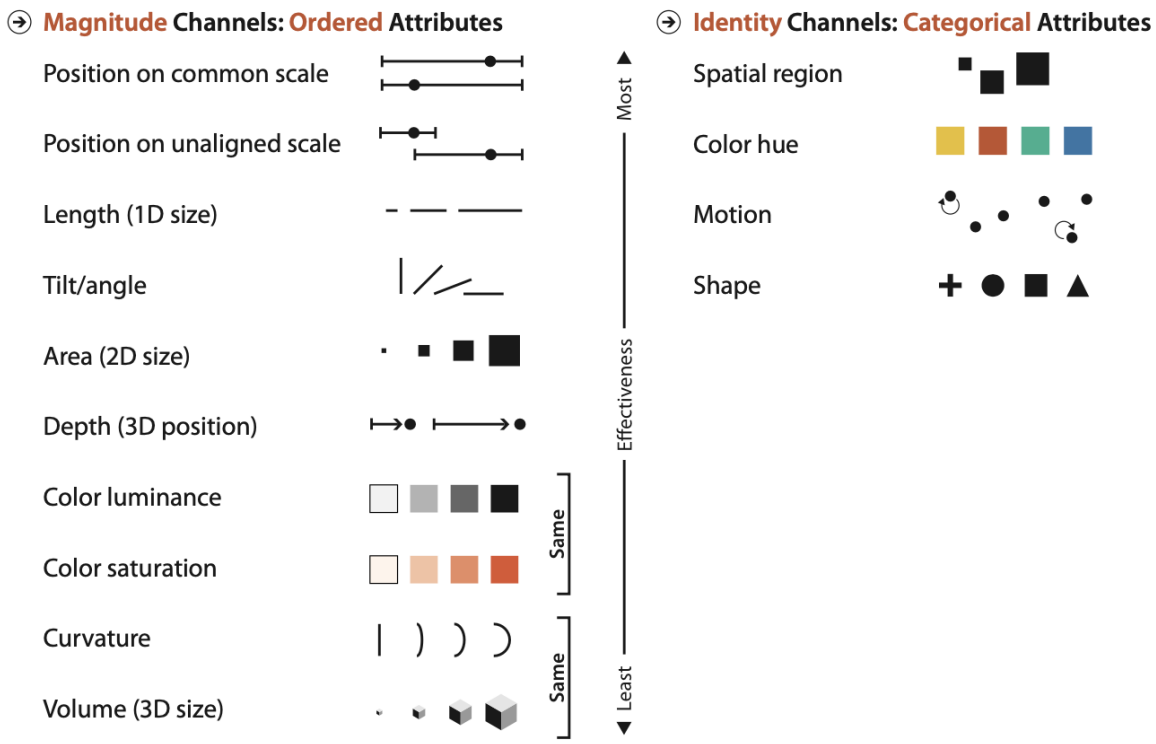
from 'Visualization Analysis and Design' by Tamara Munzner

Marks typically represent items (such individual samples) while channels show how the identity and magnitude. Channels therefore allow the data to stand out relative to other information.



from Rolandi et al. (2011)

However, not all channels are equally effective. Based on the type of data the human brain will be better at interpreting certain channels.

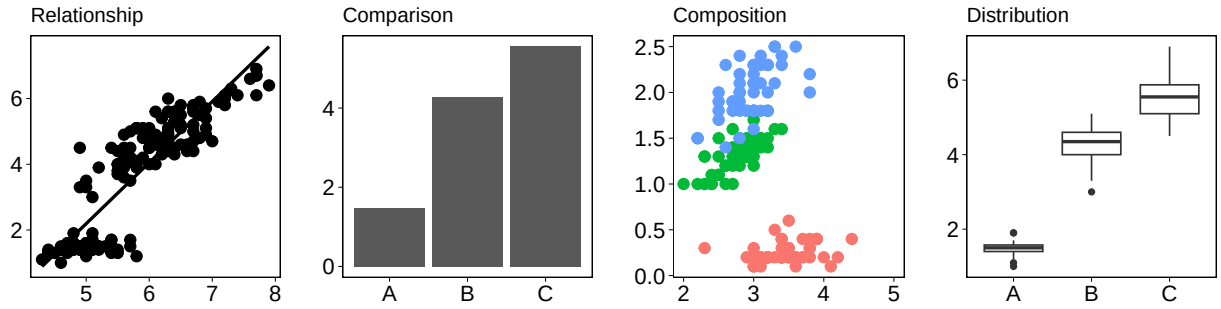


from 'Visualization Analysis and Design' by Tamara Munzner

Layout also determines how a figure is interpreted. This includes alignment, balance, and hierarchy.

3. What can figures illustrate

Combining different **marks**, **channels**, and **layout** produce different methods of conveying information about your data. Deciding what to illustrate relies on knowing the story you want your audience to understand.



Tutorial

1. Load required packages

```
library(ggplot2)
library(ggpubr)
library(ggpmisc)
library(dplyr)
library(stringr)
# You will need internet for these functions unless you download them:
source("https://lauralogozzo.github.io/assets/myCols.R.txt") # Custom color palette functions
```

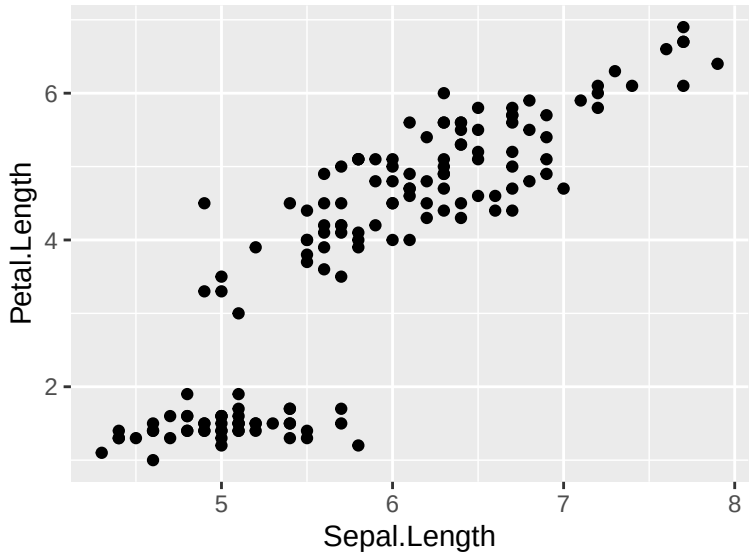
2. Load sample dataset

This loads the dataset into your environment as "iris":

```
data(iris)
```

3. Simple scatterplot in ggplot

```
# What data and what parameters to apply:
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  # Adding points to the plot:
  geom_point()
```



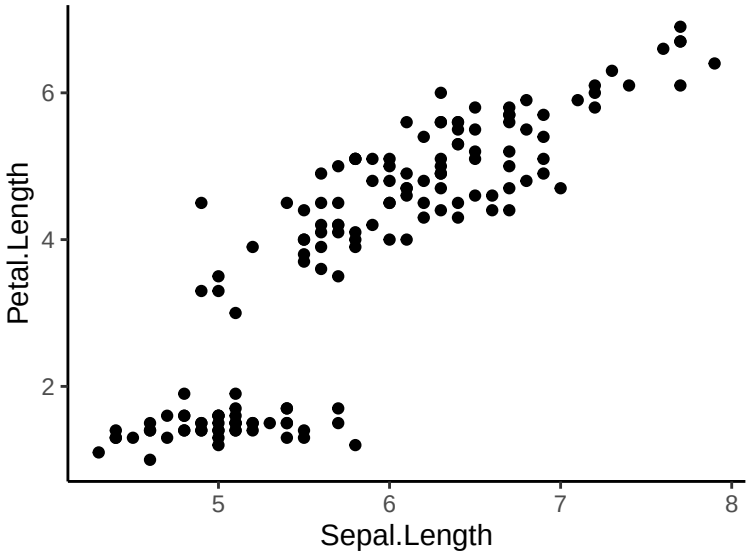
4. ggplot themes

Ggplot has some standard themes that you can use to change the background color, fonts, etc. of your plots. Try them out!

Here are some of my favorite themes and what they look like.

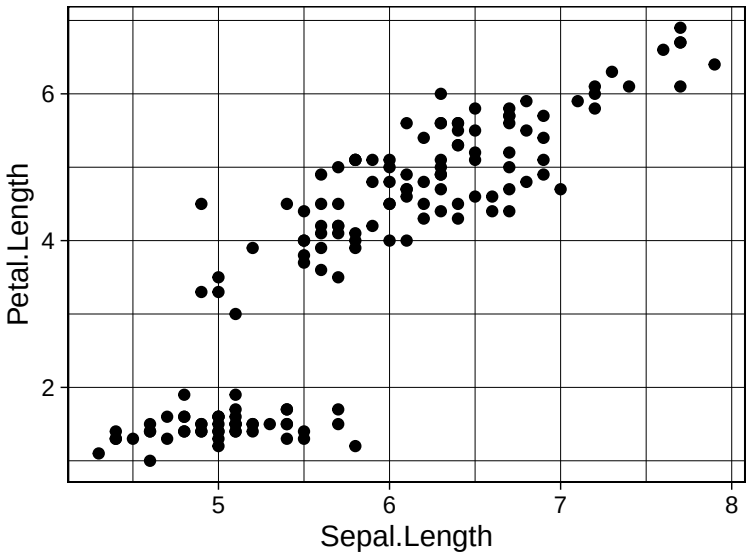
"Classic"

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  theme_classic()
```



"Linedraw"

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  theme_linedraw()
```

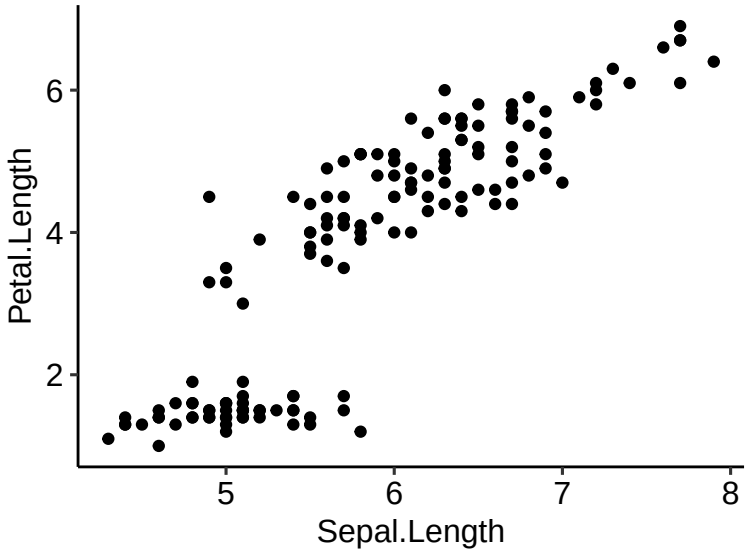


"Pubr"

Similar to classic, but consistent text sizes between axis labels and ticks

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
```

```
  theme_pubr()
```



I prefer a custom theme I created that I load in at the start of every code I use. It's pretty similar to "pubr" and sets most of the fonts to size 14 and includes a box around the plot.

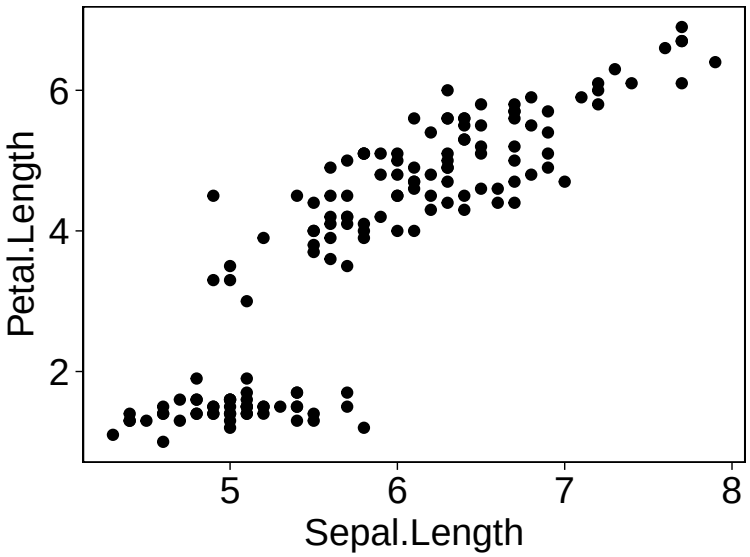
I load this in at the start of my code. If you load this in, *ALL* of your graphs will now be in that theme unless you specify a different theme. It basically overwrites ggplot's default theme *for this R session* (not permanently).

To customize a theme, we will modify the theme components. All of these components are listed here: https://ggplot2.tidyverse.org/reference/theme.html

Create a custom theme, and set that as the default

```
theme_set(theme_light() +
            theme(
              panel.grid = element_blank(),
              panel.border = element_rect(color = "black"),
              axis.title = element_text(size = 14),
              axis.text = element_text(color = "black", size = 14),
              axis.ticks = element_line(color = "black"),
              legend.text = element_text(size = 12),
              legend.title = element_text(size = 12)
            )
  )
```

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point()
```



Now that we are happy with our theme, we will add onto our plot.

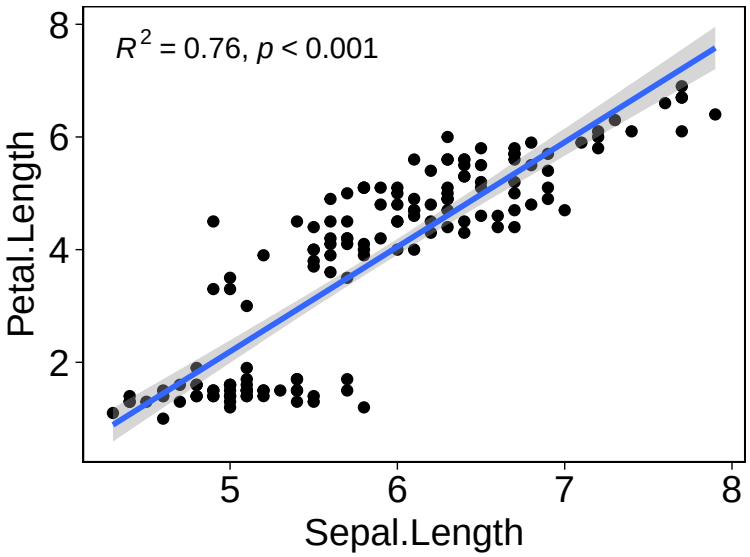
5. Adding a linear model trendline and R2

Geom_smooth adds a loess trendline by default (for n < 1000), so you need to specify that you want a linear model (method = "lm"). The default formula for method = "lm" is y~x, but this can be changed (e.g., y~log(x)). The default is to also add the standard error to the model (se = T).

We will also add linear model R2 and p values using the function stat_poly_eq() from *ggpmisc*.

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  # Adding linear model (method = "lm") with standard error (default is se = T):
  geom_smooth(method = "lm") +
  # Adding R2 and p value (the default is rr.digits = 2 and p.figits = 3. Play around with this!)
  stat_poly_eq(use_label(labels = c("R2","p")), small.p = T, rr.digits = 2)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

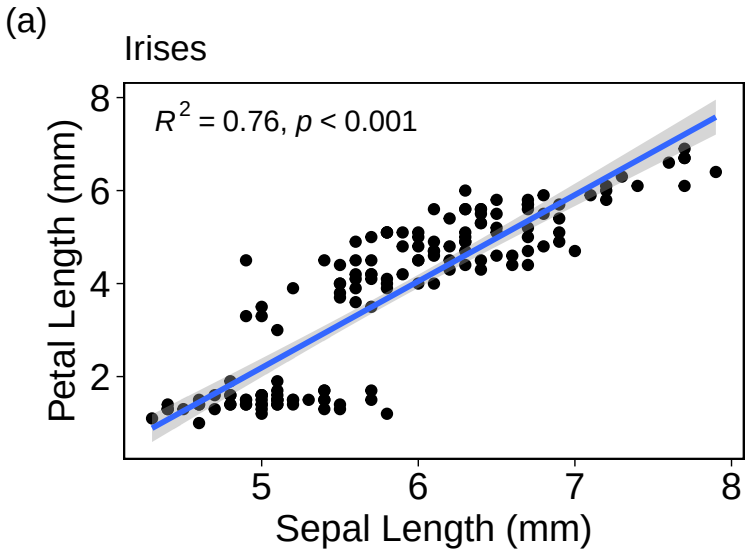


6. Adding axis labels, a title, and a "tag" for multi-panel plots

We will change the axes labels and add a title and tag (a), as we are going to add a second plot to the right of this one soon.

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  geom_smooth(method = "lm") +
  stat_poly_eq(use_label(labels = c("R2","p")), small.p = T) +
  # Changing axis labels, title, and adding a tag
  labs(x = "Sepal Length (mm)",
       y = "Petal Length (mm)",
       title = "Irises",
       tag = "(a)")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



7. Coloring plot elements by group

Coloring points and linear regressions by group. How about by species?

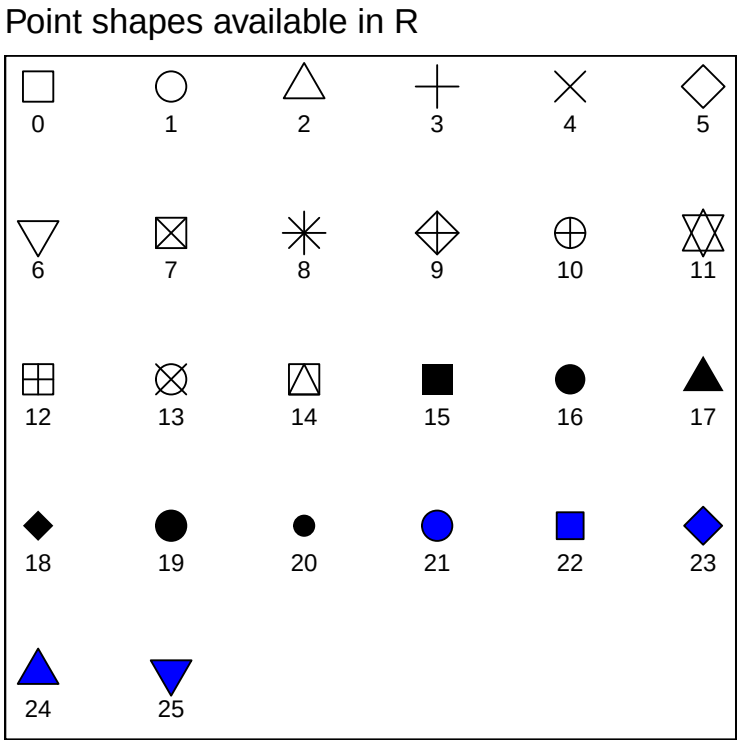
Let's use a custom color palette function to choose a color palette. Alternative color palette packages and functions are at the end of this tutorial, but I use this one because: 1- it is simple to use, 2- I like to create my own palettes and, 3- it works for both base plot and *ggplot2*.

```
plotCols()
```

Let's grab a color palette we like, and name it "cols" for later:

```
cols <- getCols("summer")[c(1,3,6)]
# In this case, we are grabbing colors 1, 3, and 6 from the palette only
```

Let's also change the shape of the points for each group. Find the available shapes using this function:

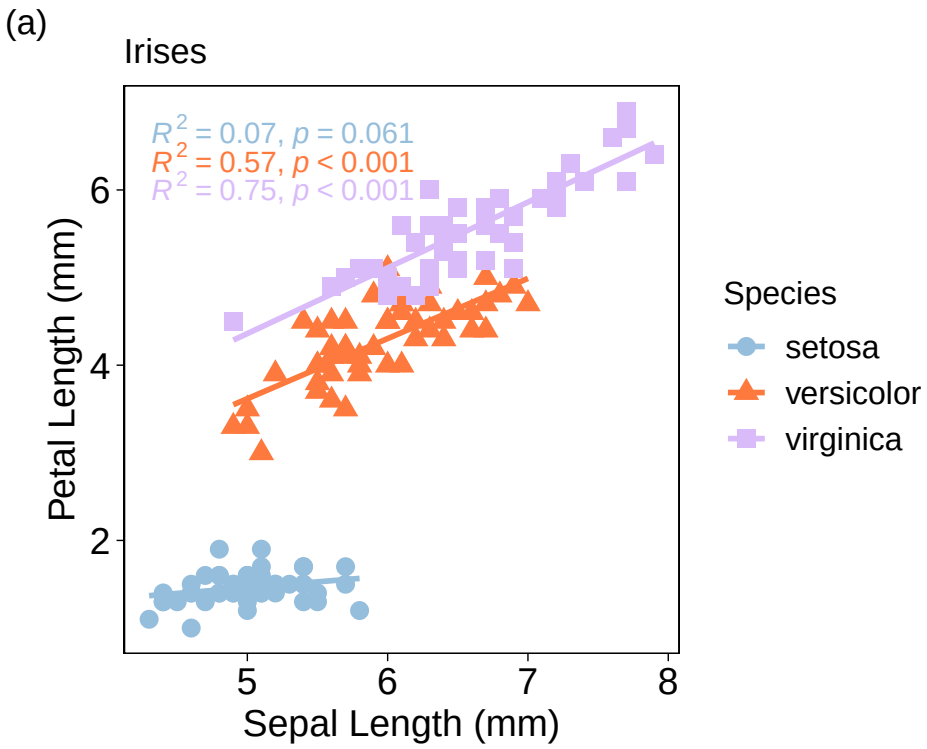


You can find the names of each shape here:
https://www.datanovia.com/en/blog/ggplot-point-shapes-best-tips/

Let's also remove the standard error from the linear models since we now have a few linear regressions on the plot and it can get busy.

```
# Adding into aes: color = Species, shape = Species
ggplot(data = iris, aes(x= Sepal.Length, y = Petal.Length, color = Species, shape = Species)) +
  # Increase point size to 3
  geom_point(size = 3) +
  # Adding linear model withOUT standard error (se = F):
  geom_smooth(method = "lm", se = F) +
  stat_poly_eq(use_label(labels = c("R2","p")), small.p = T) +
  # Changing axis labels, title, and tag
  labs(x = "Sepal Length (mm)",
       y = "Petal Length (mm)",
       title = "Irises",
       tag = "(a)") +
  # Setting the color scale to our chosen color palette
   scale_color_manual(values = cols)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



An important thing to note here is that we only include aesthetic changes inside the aes() function if we are referencing our data in some way (i.e., if we want that aesthetic to change by a variable). If you want to specify aesthetic elements that are not dependent on your data (like in this case, all point sizes = 3), then we would include it when we add that ggplot element to the plot - do not include it inside the aes() function. Try adding "size" inside aes() to see what happens.

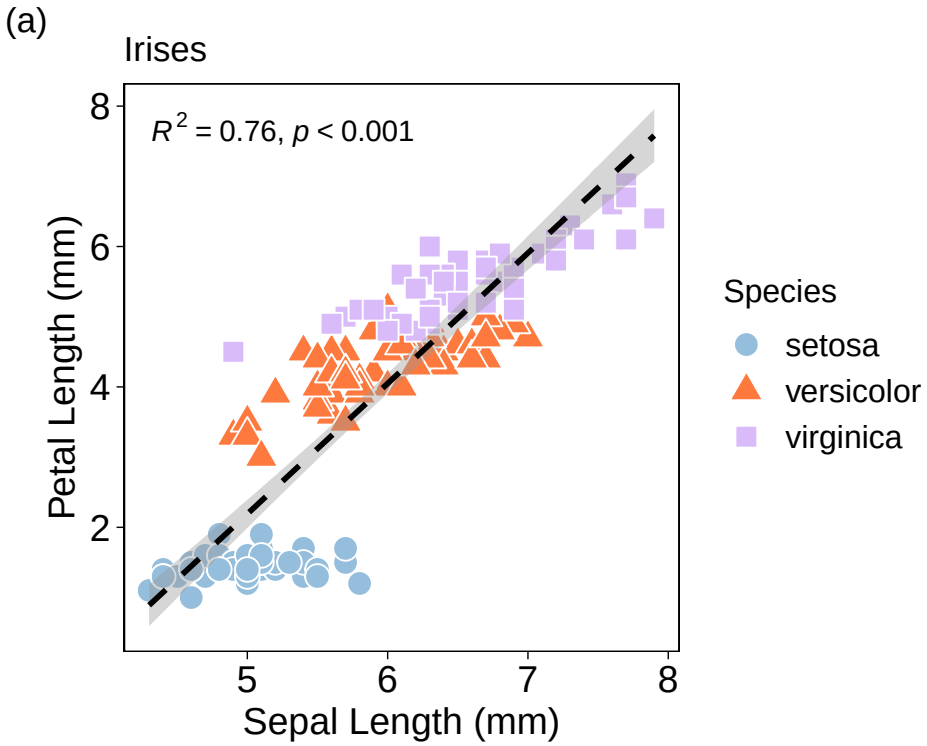
What if we don't want an individual linear model for each group, but just the points to be colored/shaped by group?

We can move the color and shape to the geom_point() function, rather than in the ggplot() function (which applies those aesthetics to every single ggplot element from then on).

This is our final graph, so we will label it p1 so that we can add an additional graph to its right later. The parentheses around the entire code chunk is so that the plot is still displayed while we name it something.

```
(p1 <- ggplot(data = iris, aes(x= Sepal.Length, y = Petal.Length)) +
  # Specifying point fill color and shape by species, and make the outline of all the points white:
  geom_point(aes(fill = Species, shape = Species), size = 4, color = "white") +
  # Changing the line color to black and linetype to "dashed". We can also include the se again.
  geom_smooth(method = "lm", color = "black", linetype = "dashed") +
  stat_poly_eq(use_label(labels = c("R2","p")), small.p = T) +
  labs(x = "Sepal Length (mm)",
       y = "Petal Length (mm)",
       title = "Irises",
       tag = "(a)") +
   scale_fill_manual(values = cols) +
   # Specify shapes
   scale_shape_manual(values = c("circle filled","triangle filled", "square filled"))
)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

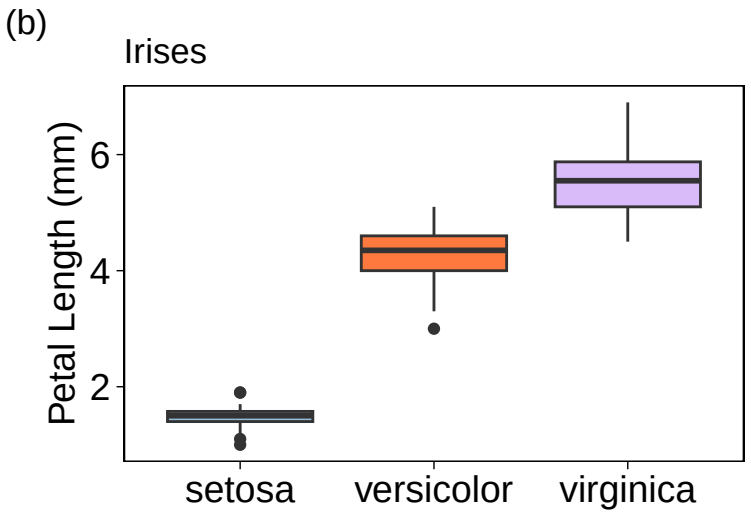


Note: if you want to color or shape something by a data element (like Species), you need to include this within an aes() function. If you want to just color an element by a color that you dictate (like how we made the linear regression line black), do NOT include the color within aes(), and make the color or shape equal to the color or shape you want, in quotes.

8. Boxplot

Create a simple boxplot

```
# We are telling it to fill the boxplot colors by Species:
ggplot(data = iris, aes(x = Species, y = Petal.Length, fill = Species)) +
  # Add boxplots (no need to show a legend for the boxplot colors):
  geom_boxplot(show.legend = F) +
  # Change labels
  labs(y = "Petal Length (mm)",
       x = "",
       title = "Irises",
       tag = "(b)") +
  # Adding fill color:
  scale_fill_manual(values = cols)
```

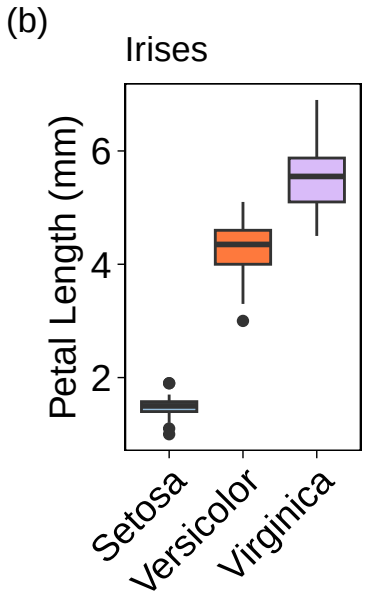


Some aesthetic changes to the boxplot

First, in order for the x-axis to be displayed, the plot width needs to be much bigger making the plot look stretched out. So let's rotate the x-axis labels by 45° so that we can make our plot width smaller.

Next, the x-axis labels are all lowercase, but let's pretend that we want them to be sentence case. The easiest way to change this is using the str_to_sentence function on our x axis values when we first call this into ggplot().

```
(p2 <- ggplot(data = iris, aes(x = str_to_sentence(Species), y = Petal.Length, fill = Species)) +
    geom_boxplot(show.legend = F) +
    labs(y = "Petal Length (mm)",
         x = "",
         title = "Irises",
         tag = "(b)") +
   # Adding fill colors:
   scale_fill_manual(values = cols) +
   # Rotate the x-axis labels
   theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))
)
```



9. Multi-panel figures

Let's combine the two plots into one figure using ggarrange() from *ggpubr*

In ggarrange(), you need to specify the number of columns (ncol) and the number of rows (nrow) of graphs that you want. For example, ncol = 3, and nrow = 2 makes 6 graphs in this orientation:

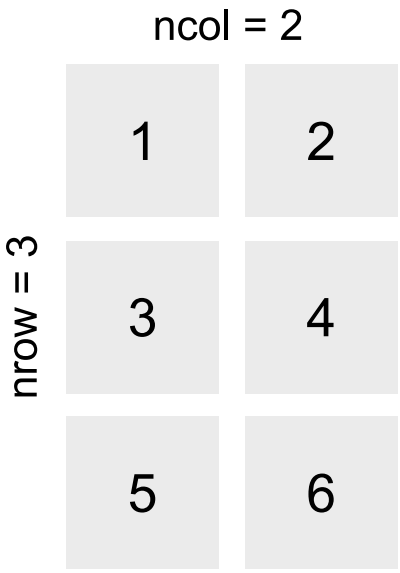


Figure 1: ncol = 2, nrow = 3

We want two graphs horizontally next to each other. So we would specify ncol = 2, nrow = 1:

```
ggarrange(p1, p2, ncol = 2, nrow = 1)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

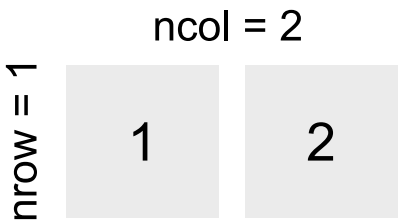
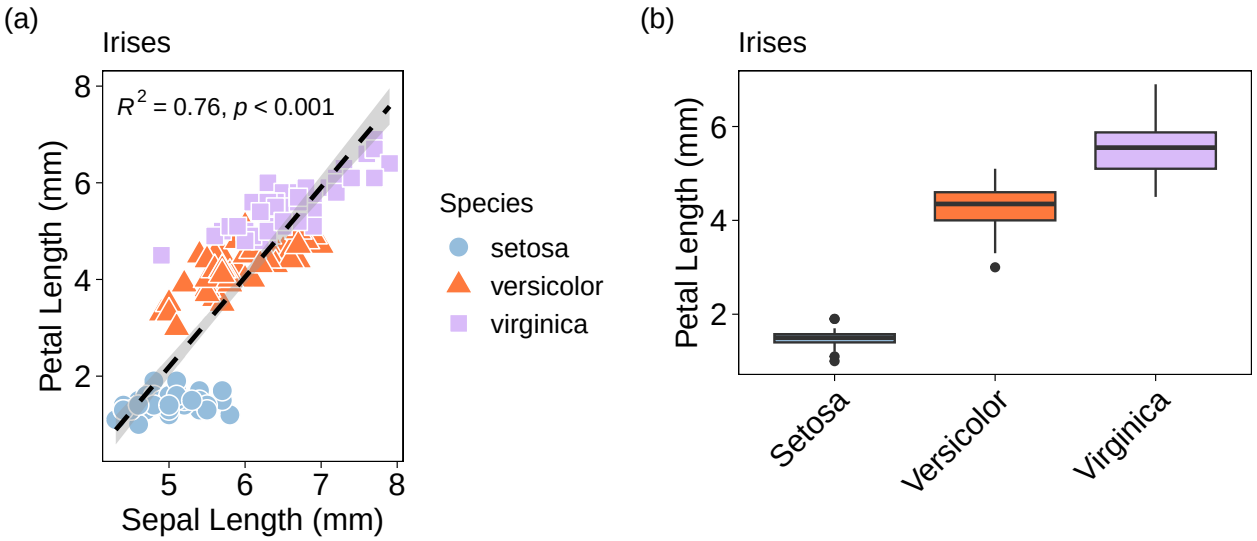


Figure 2: ncol = 2, nrow = 1



This looks really messy. The height of the plots don't match, the boxplot is a bit wide. We can change up some of the parameters in ggarrange() to fix these.

- Making the width of the first plot greater than the second plot with widths = c(2,1) (first graph = 2x the width of the second graph):

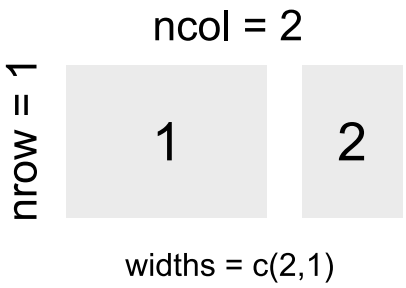


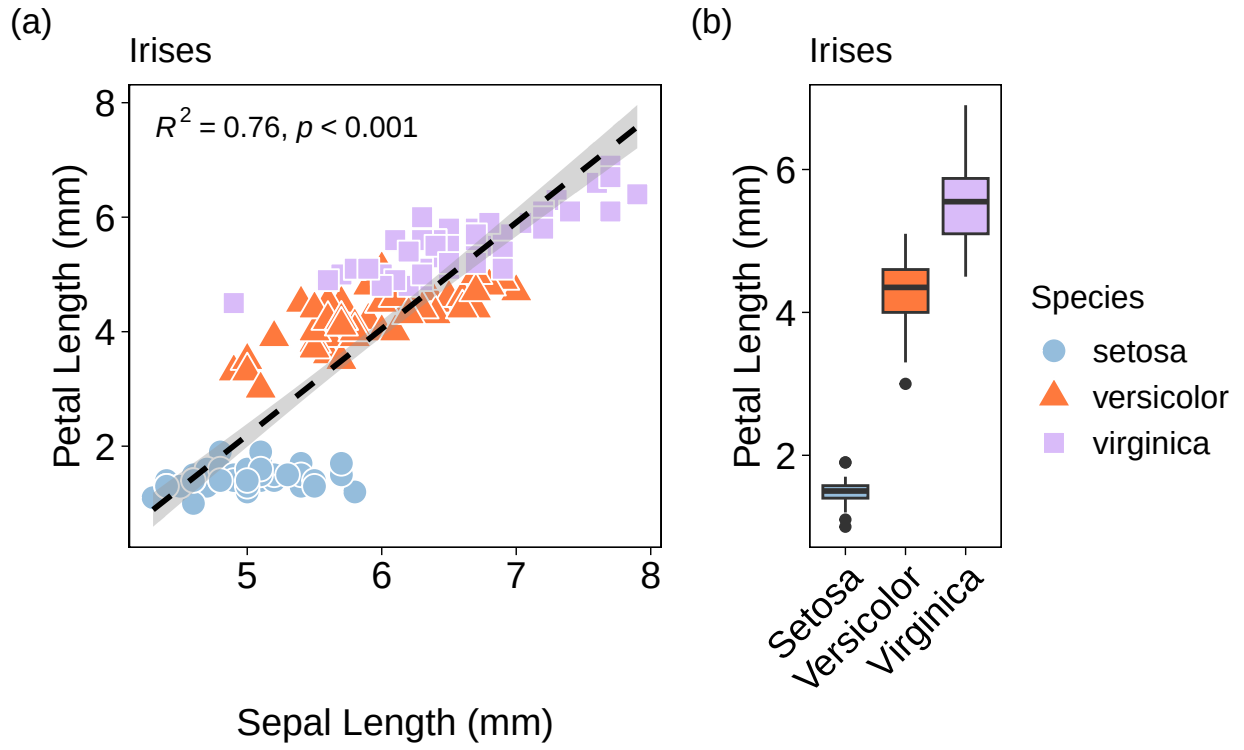
Figure 3: ncol = 2, nrow = 1, widths = c(2,1)

- Aligning the graph heights with align = "h" (h stands for align horizontally)
- Making the legend common to all graphs, which moves it to the outside (**Note**: this won't work right if you have different legends for different plots!). Alternatively, when you create p1 earlier, you can instead move the legend to inside the plot instead of outside.

- Aligning the legend right

```
(p <- ggarrange(p1, p2, ncol = 2, nrow = 1, widths = c(2,1), align = "h",
                common.legend = T, legend = "right"))
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```



Much better!

10. File formats and saving figures

We can save ggplot figures to our computer using the ggsave() function from *ggplot2*.

Before we do that, let's learn about file types.

File types

There are two types of file formats for images: rasters and vectors.

1. RASTERS: your image is made up of tiny pixels and so the resolution of your image (number of pixels per inch) will determine how clear the image is (and how LARGE your file is).

- file formats: .jpg, .png, .tif

2. VECTORS: your image is made up of points on a grid and can infinitely zoom in/out, change size, without losing resolution. **You almost always want to save your images as a vector file for this reason, and because they are MUCH smaller than raster files.**

- file formats: .pdf, .eps, .svg

You will have to check journal guidelines to see what image formats they allow. They should allow at least one vector format.

Zoom in on each of these plots saved as a raster (top) and vector (bottom) to see the difference, and try saving your plots as vectors vs rasters (in different resolutions) to see what each file format looks like.

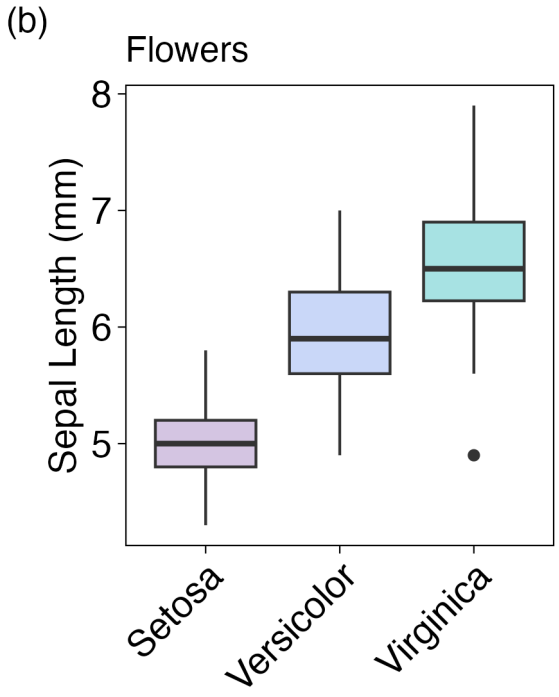


Figure 4: Raster

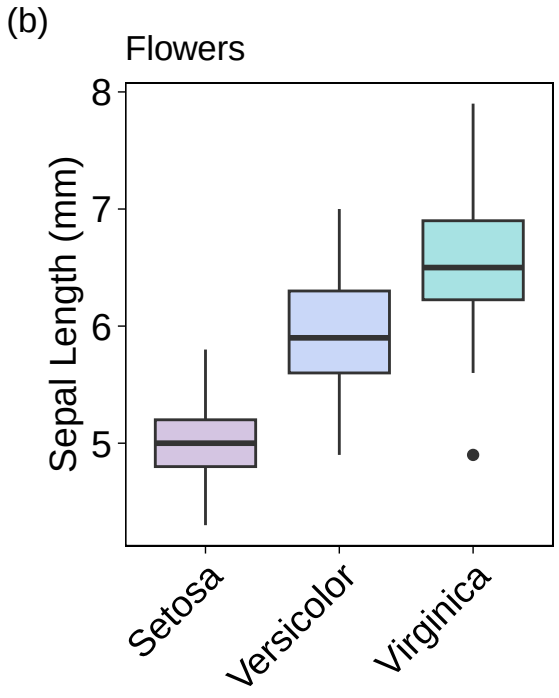


Figure 5: Vector

Save plot as .pdf vector using ggsave

```
ggsave(plot = p, filename = "Figures/Fig1.pdf",
       device = "pdf", width = 8, height = 4)
```

Save plot as .png raster using ggsave (300 dpi resolution)

```
ggsave(plot = p, filename = "Figures/Fig1.png",
       device = "png", dpi = 300, width = 8, height = 4)
```

11. Including labels with subscripts, superscripts, italics, etc.

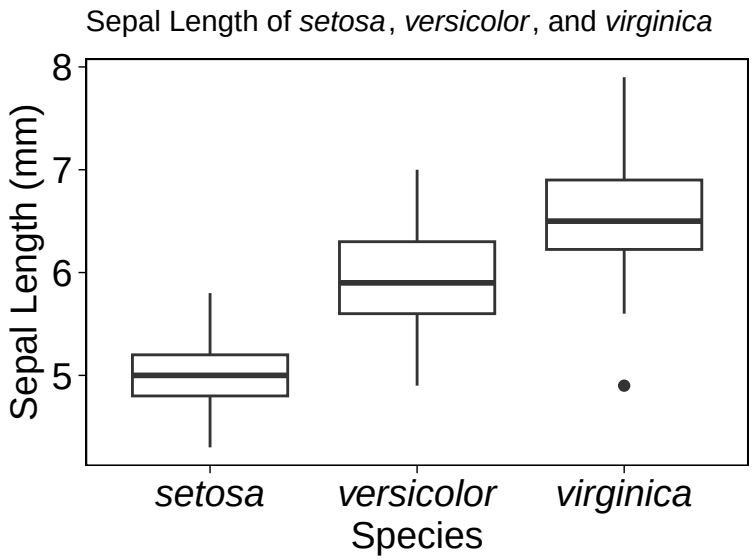
Use expression() and paste(), nested as expression(paste()).

Some syntax rules for this:

- Put all character strings (text) you want printed in the graph in quotes " "
- Separate all strings of characters that are not in the same quotes by commas, unless you are applying things like superscript or subscripts. Example: expression(paste("this",italic("is"),," an example"))
- Superscript: "word"^"superscript"
- Subscript: "word"["subscript"]
- Greek letters should NOT be written in quotes (e.g., if you wanted to write µM): expression(paste(mu,"M"))

Let's do a simple example:

```
ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  labs(subtitle = expression(paste("Sepal Length of ", italic("setosa"),", ", italic("versicolor"),", a
       y = "Sepal Length (mm)") +
  theme(axis.text.x = element_text(face = "italic"))
```



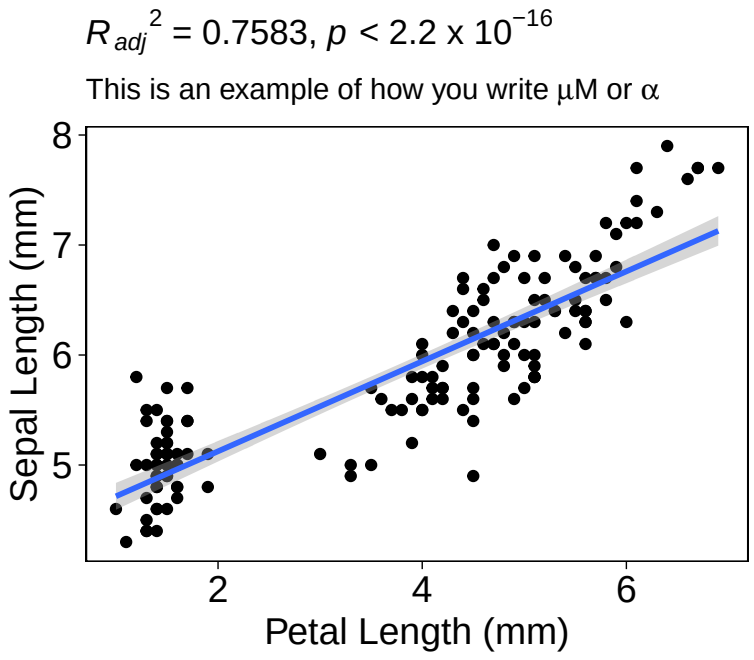
```
summary(lm(Sepal.Length~Petal.Length, data = iris))
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.24675 -0.29657 -0.01515  0.27676  1.00269
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.30660    0.07839   54.94   <2e-16 ***
## Petal.Length  0.40892    0.01889   21.65   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4071 on 148 degrees of freedom
## Multiple R-squared:   0.76,  Adjusted R-squared:  0.7583
## F-statistic: 468.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

A more complex example:

```
ggplot(data = iris, aes(x = Petal.Length, y = Sepal.Length)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = expression(paste(italic("R"["adj"])^"2", " = ", "0.7583, ",
                                italic("p"), " < 2.2 x 10"^"-16")),
       subtitle = expression(paste("This is an example of how you write ",mu ,"M or ", alpha)),
       y = "Sepal Length (mm)",
       x = "Petal Length (mm)")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



12. Plotting resources

- R graph gallery: https://r-graph-gallery.com/
- Color palette generator: https://coolors.co/
- Checking the readability of your graphs for colorblindness: https://www.color-blindness.com/coblis-color-blindness-simulator/
- Readings:
  - 'Fundamentals of Data Visualization'
  - 'Graphs, Tables, and Figures in Scientific Publications: The Good, the Bad, and How Not to Be the Latter'
  - '10 Good and Bad Examples of Data Visualization (2023)'
- Comprehensive ist of *ggplot2* extension packages: https://exts.ggplot2.tidyverse.org/gallery/
- Some good packages:
  - Color palettes: *RColorBrewer*
  - Multi-panel figures: *patchwork*

Thank concludes our tutorial! Check out some additional useful tricks below.

Supplement

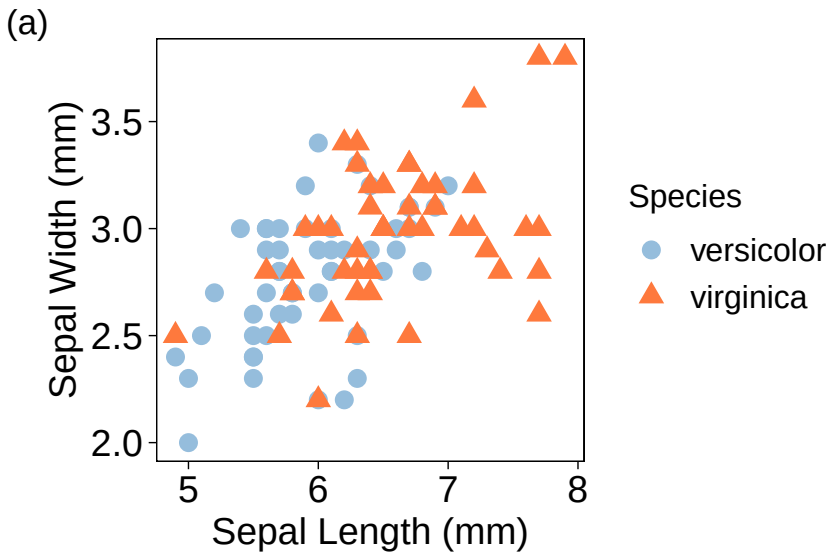
Here are some additional things that you may need to do when plotting using *ggplot2*.

S1. Filtering data before plotting (i.e., only certain sites, species, etc.)

I find that using *dplyr* is easiest. With *dplyr*, you can modify dataframe elements before plotting them and then "pipe" them into ggplot. The benefit of this is that it will not change your saved dataset.

- A pipe is: %>% and it basically means "then do this."
- With following code, we are saying, take the dataset "iris" then (%>%) filter out "setosa" from the Species column then (%>%) pipe that into ggplot.
- **Notice how we do not include the data command in the *ggplot* code. This is because we are piping the modified data straight into ggplot in the previous step!**

```
iris %>%
  filter(Species != "setosa") %>%
  ggplot(aes(x=Sepal.Length, y = Sepal.Width)) +
  geom_point(size = 3, aes(color = Species, shape = Species)) +
  scale_color_manual(values = cols) +
  labs(x = "Sepal Length (mm)", y = "Sepal Width (mm)",
       tag = "(a)")
```



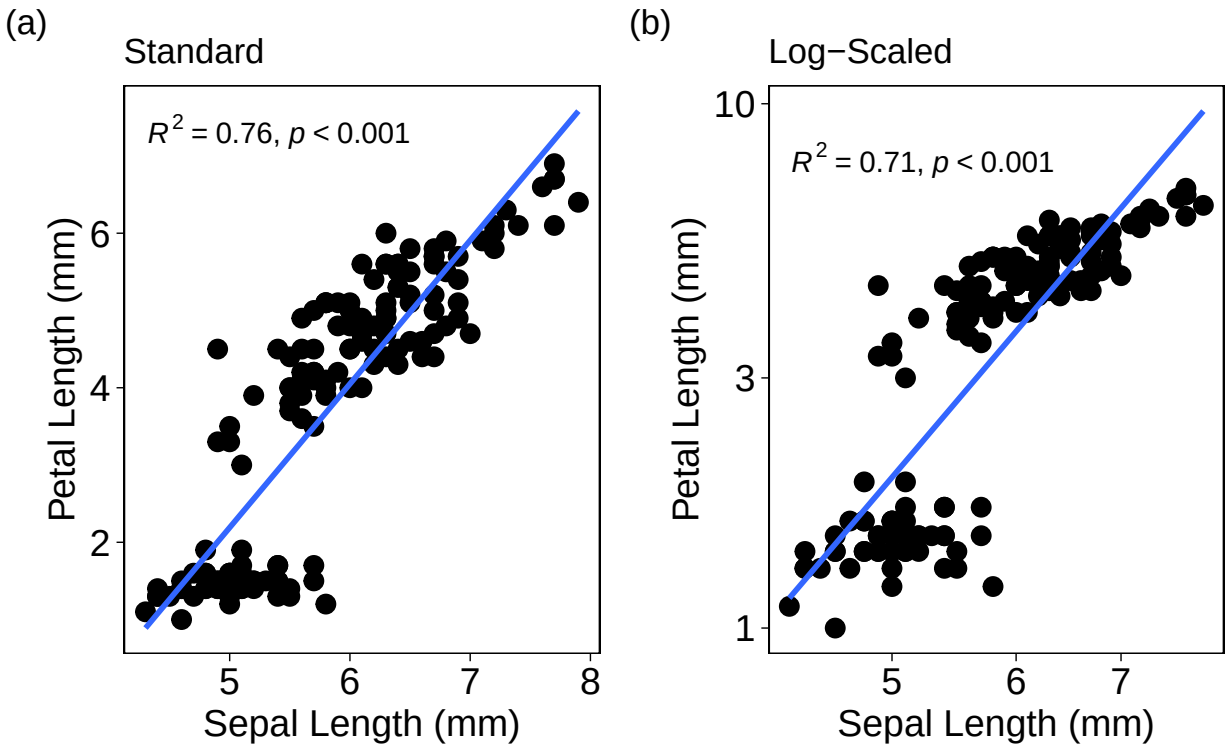
S2. Changing axes (e.g., to log-scaled or changing axis limits)

Log-scale

Log-scaling the axis automatically also applies to the linear regression

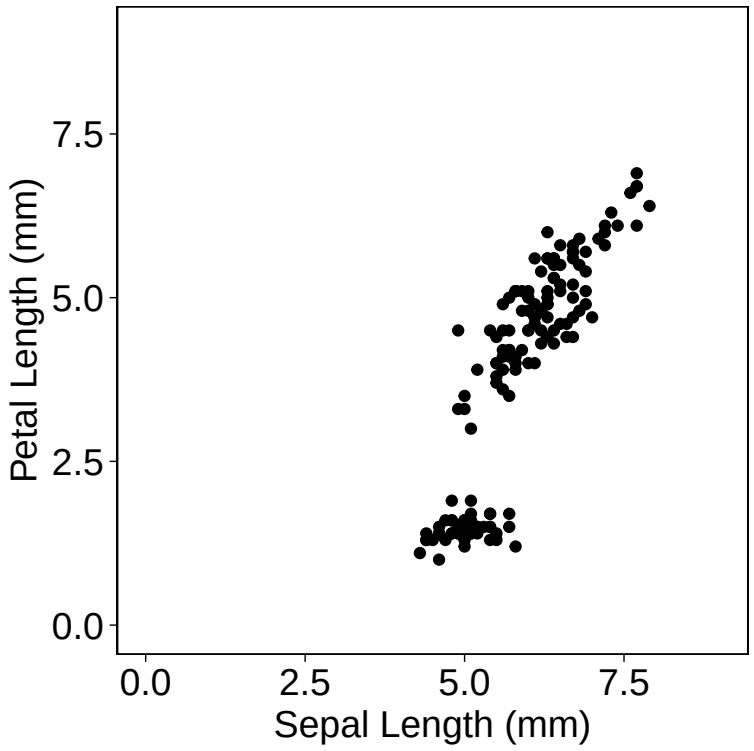
```
p1 <- iris %>%
  ggplot(aes(x=Sepal.Length, y = Petal.Length)) +
  geom_point(size = 3) +
  scale_color_manual(values = cols) +
  labs(x = "Sepal Length (mm)", y = "Petal Length (mm)", title = "Standard", tag = "(a)") +
  geom_smooth(method = "lm", se = F) +
  stat_poly_eq(use_label(labels = c("R2","p")), small.p = T)

p2 <- iris %>%
  ggplot(aes(x=Sepal.Length, y = Petal.Length)) +
  geom_point(size = 3) +
  scale_color_manual(values = cols) +
  labs(x = "Sepal Length (mm)", y = "Petal Length (mm)", title = "Log-Scaled", tag = "(b)") +
  geom_smooth(method = "lm", se = F) +
  stat_poly_eq(use_label(labels = c("R2","p")), small.p = T, label.y = 0.9) +
  scale_y_log10() +
  scale_x_log10()

ggarrange(p1,p2)
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```



Changing axis limits

```
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point() +
  # One method of setting axis limits:
  xlim(0,9) +
  # Another method:
  scale_y_continuous(limits = c(0, 9)) +
  labs(x = "Sepal Length (mm)", y = "Petal Length (mm)")
```

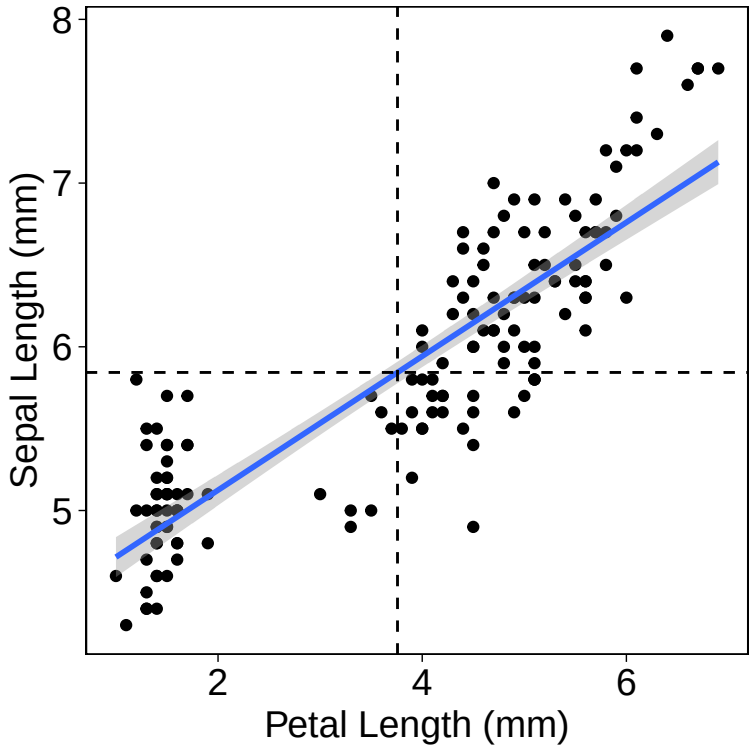


S3. Adding lines to scatterplot

Adding a line for the mean of each variable

```
iris %>%
  ggplot(aes(x = Petal.Length, y = Sepal.Length)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(y = "Sepal Length (mm)",
       x = "Petal Length (mm)") +
  # y mean line:
  geom_hline(yintercept = mean(iris$Sepal.Length, na.rm = T), linetype = "dashed") +
  # x mean line:
  geom_vline(xintercept = mean(iris$Petal.Length, na.rm = T), linetype = "dashed")
```

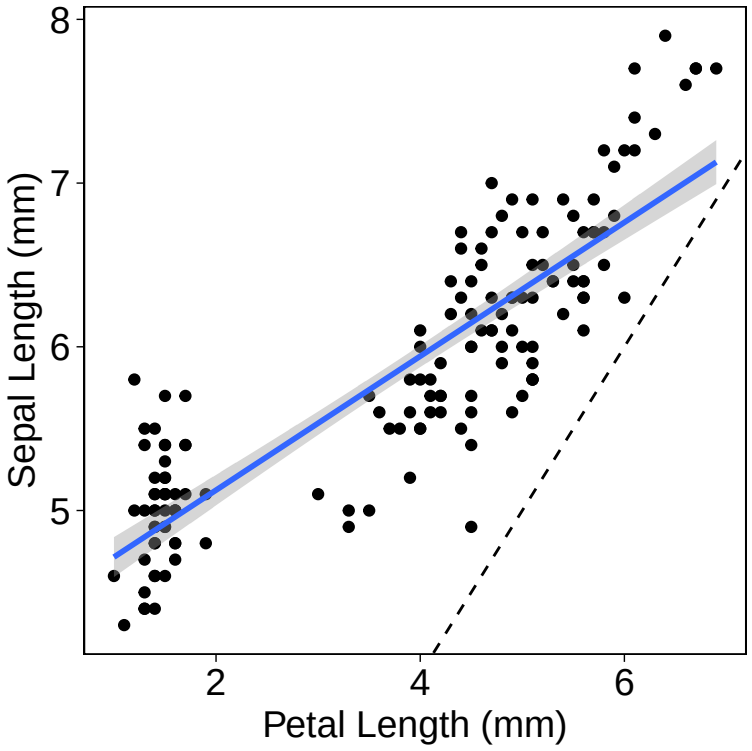
```
## `geom_smooth()` using formula = 'y ~ x'
```



Adding a 1:1 line

```
iris %>%
  ggplot(aes(x = Petal.Length, y = Sepal.Length)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(y = "Sepal Length (mm)",
       x = "Petal Length (mm)") +
  # 1:1 line:
  geom_abline(intercept = 0, slope = 1, linetype = 2)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

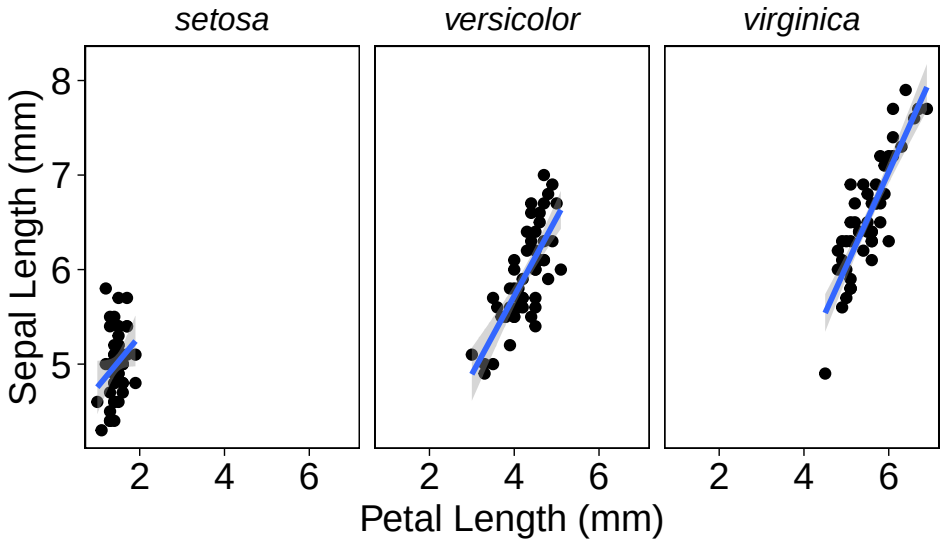


S4. Facet wrap: plotting a different plot for each factor level (i.e., Species).

Let's also make the titles of each facet italic since they are species names.

```
iris %>%
  ggplot(aes(x = Petal.Length, y = Sepal.Length)) +
  facet_wrap(~Species) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(y = "Sepal Length (mm)",
       x = "Petal Length (mm)") +
  theme(strip.background = element_blank(),
        strip.text = element_text(color = "black", size = 12, face = "italic"))
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

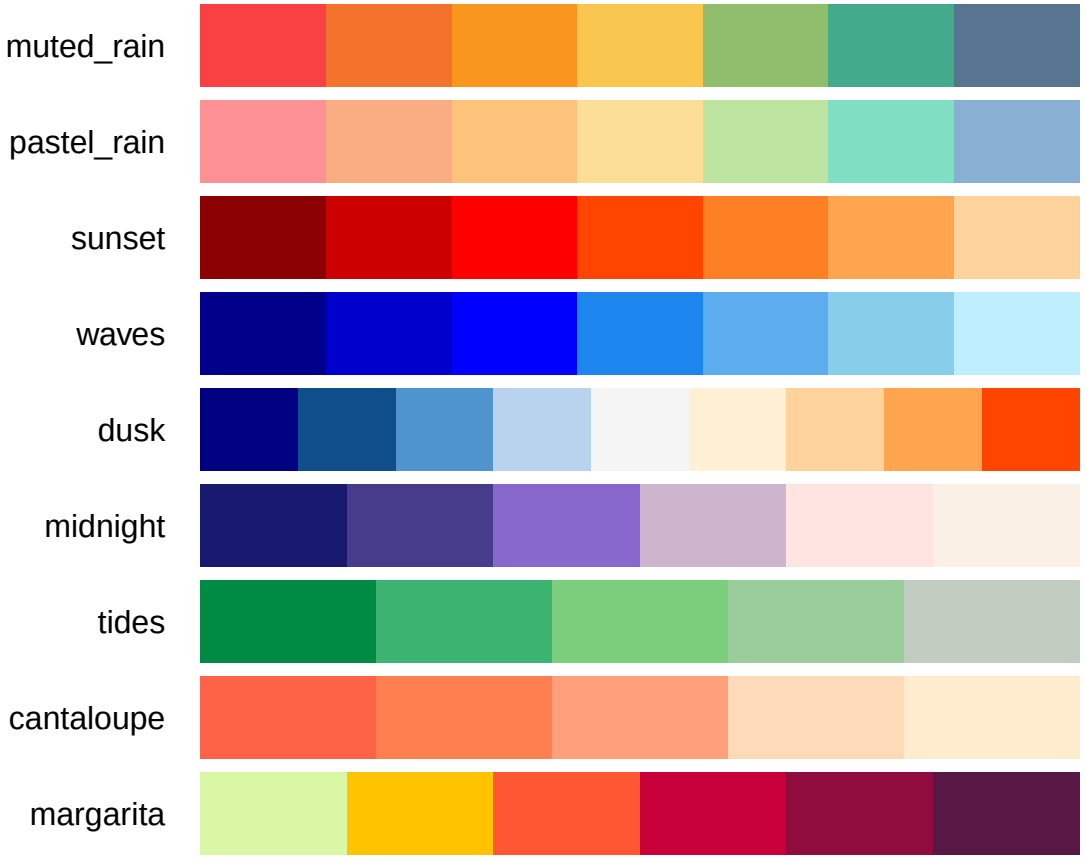


S5. Continuous color scales and changing the legend label

We can also have the point color vary by a continuous data variable, such as the Petal Length, instead of groups. We will need to pick a color palette that works well continuously if the colors are blended together

Try seeing the "ramped" color palettes available, which are great for plotting continuous data!

```
plotCols("ramp")
```

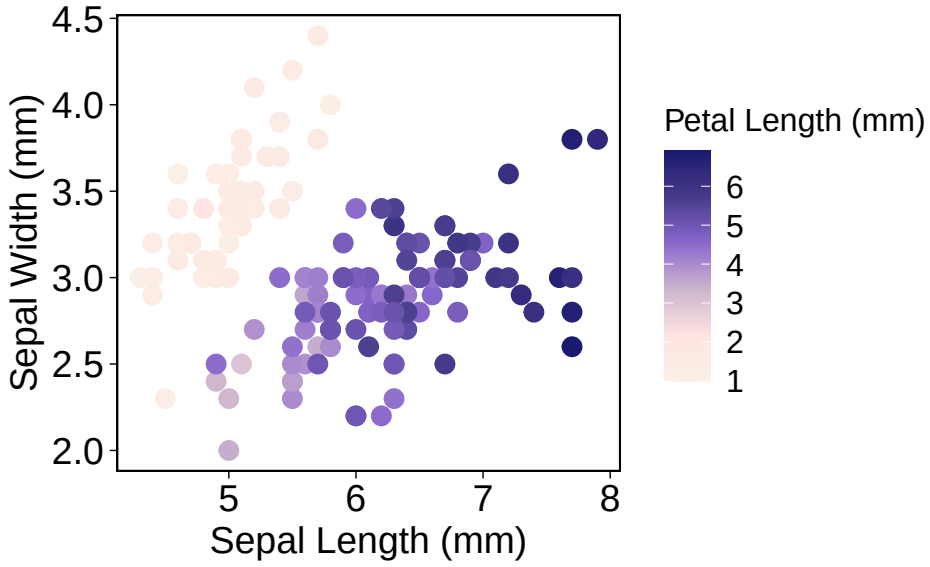


Let's choose the midnight palette (but put it in reverse, so that low value are lighter colors).

We can add this to the plot by using scale_color_gradientn() and making the colours argument equal to our new palette.

We can also label the legend using the name argument in scale_color_gradientn(), OR the colors argument in labs()

```
cols.cont <- rev(getCols("midnight"))

iris %>%
  ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Petal.Length)) +
  geom_point(size = 3) +
  labs(x = "Sepal Length (mm)",
       y = "Sepal Width (mm)") +
  scale_color_gradientn(colours = cols.cont, name = "Petal Length (mm)")
```



S6. Combining multiple graphs with different legends, and setting where the plots should go.

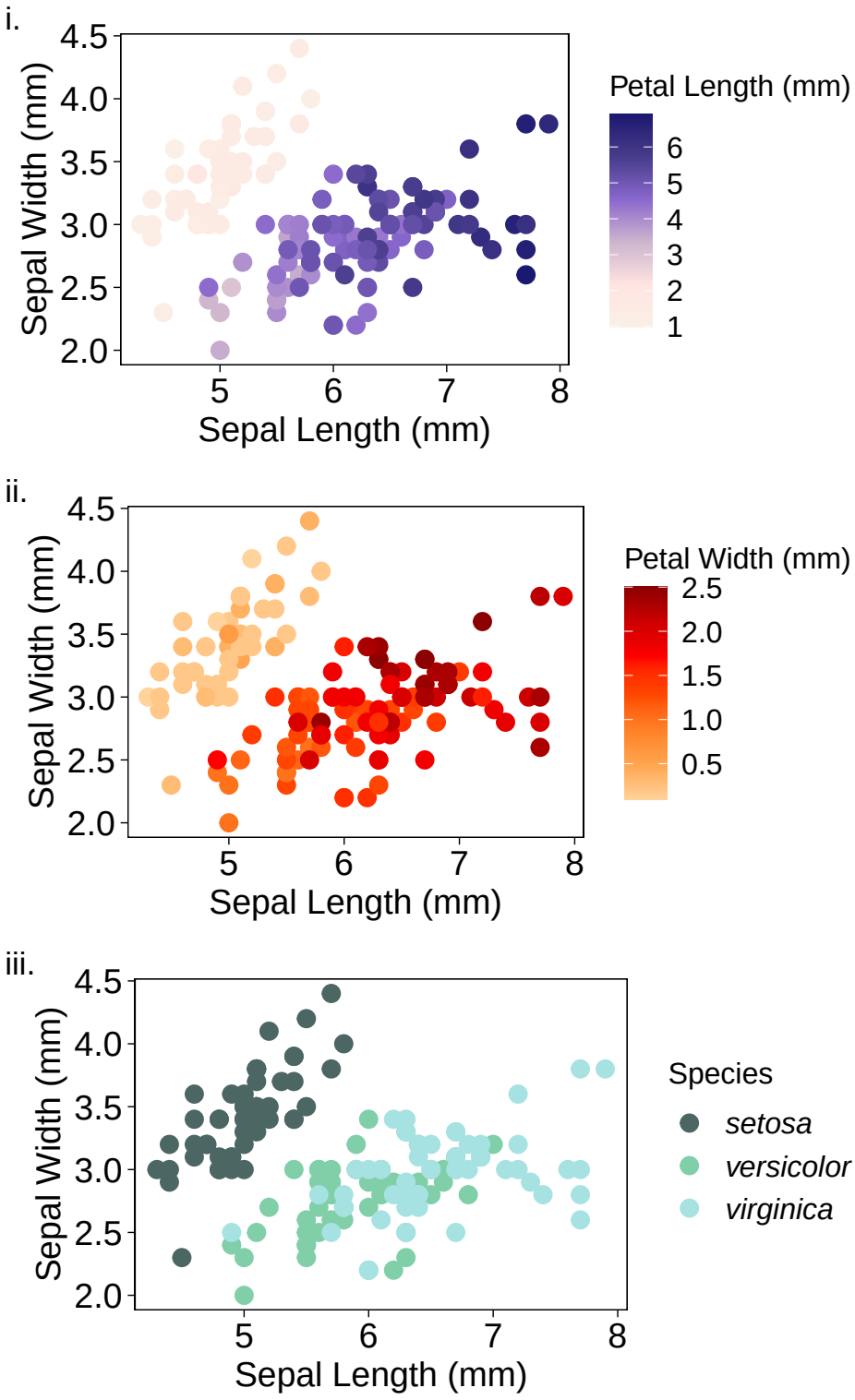
Let's create 3 plots with different legends:

```
cols.cont <- getCols("midnight")
p1 <- iris %>%
  ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Petal.Length)) +
  geom_point(size = 3) +
  labs(x = "Sepal Length (mm)", y = "Sepal Width (mm)", tag = "i.") +
  scale_color_gradientn(colours = rev(cols.cont), name = "Petal Length (mm)")

cols.cont2 <- getCols("sunset")
p2 <- iris %>%
  ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Petal.Width)) +
  geom_point(size = 3) +
  labs(x = "Sepal Length (mm)", y = "Sepal Width (mm)", tag = "ii.") +
  scale_color_gradientn(colours = rev(cols.cont2), name = "Petal Width (mm)")
```

```r
cols.sp <- getCols("seasnail")
p3 <- iris %>%
  ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point(size = 3) +
  labs(x = "Sepal Length (mm)", y = "Sepal Width (mm)", tag = "iii.") +
  scale_color_manual(values = rev(cols.sp), name = "Species") +
  theme(legend.text = element_text(face = "italic"))
```

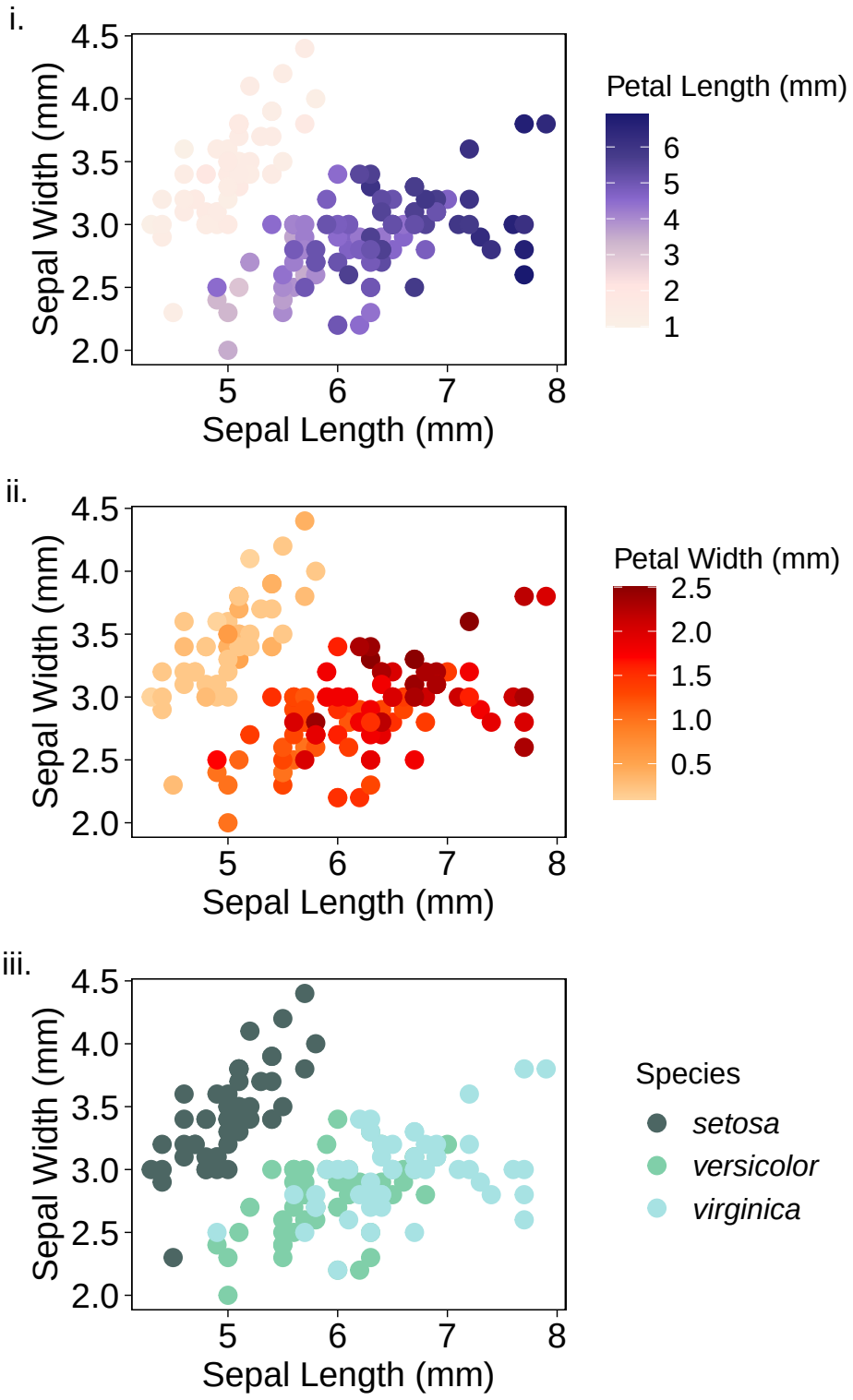
Now let's ggarrange them, and specify that we want 1 column and 3 rows of plots (i.e., 3 plots stacked on top of each other).

```r
ggarrange(p1,p2,p3, ncol = 1, nrow = 3)
```



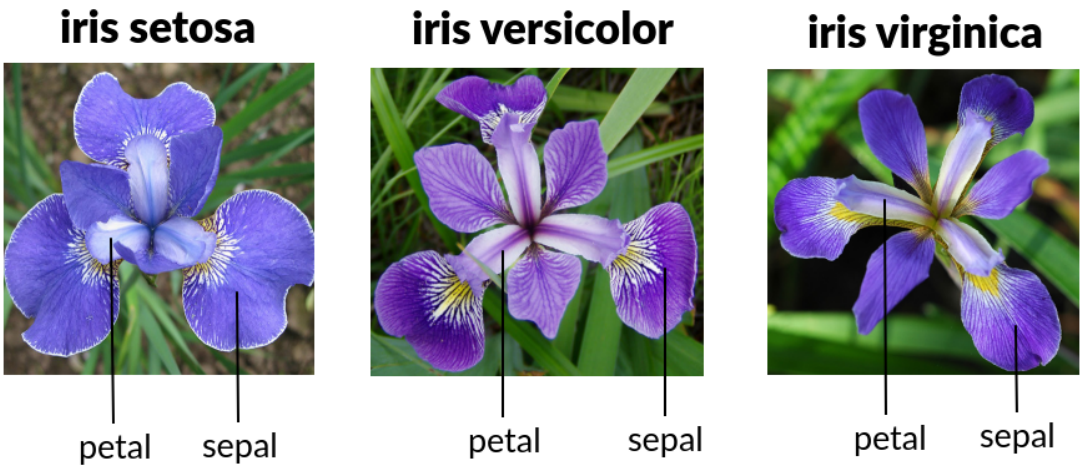
Don't forget to vertically align these plots so it's neat, given our legends are all different sizes!

```
ggarrange(p1,p2,p3, ncol = 1, nrow = 3, align = "v")
```



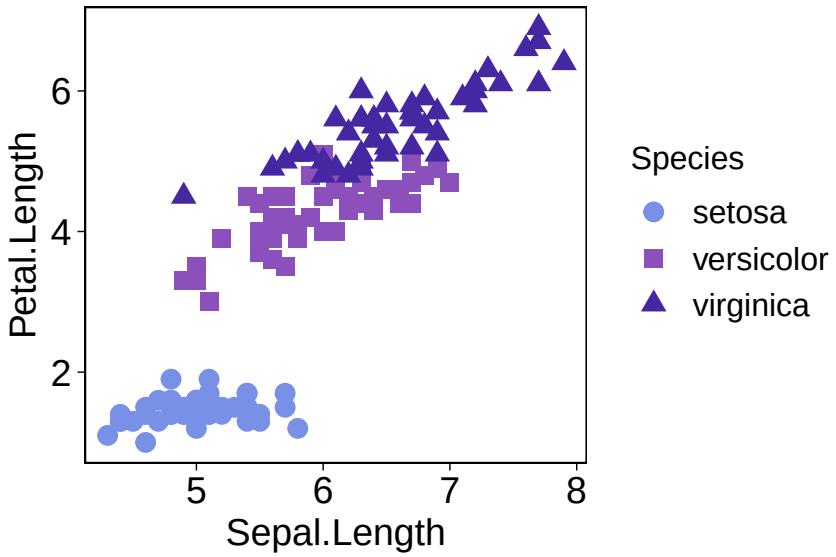
S7. Manually setting legend colors, shapes etc.

Let's say you want to set specific colors to specific factors. For example, maybe we want to make the colors of our iris species the actual color of the irises!



We can do this using coolors.co's "Create palette from photo" option.

```
iris %>%
  ggplot(aes(x = Sepal.Length, y = Petal.Length, color = Species, shape = Species)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("setosa" = "#7890E6",
                                "versicolor" = "#8C50BC",
                                "virginica" = "#4527A1")) +
  scale_shape_manual(values = c("setosa" = "circle",
                                "versicolor" = "square",
                                "virginica" = "triangle"))
```



References

Rolandi, M., Cheng, K., Pérez-Kriz, S. "A Brief Guide to Designing Effective Figures for the Scientific Paper" (2011) *Advanced Materials*