ANSWERS - Colorblind-friendly & Publication-Quality Data Visualization in R

The goal of this tutorial is to introduce you to the basics of using ggplot2 by exploring various dimensions of a data frame. All commands are in *italics and a distinct font.* All prompts that require a response are in blue.

NOTE: Copying and pasting commands into R will not work due to subtle differences in default characters used in word and R. Please type out all commands. This will also help in getting familiar and learning to code.

If you do not have ggplot2, ggfortify, reshape2, ggExtra, and RColorBrewer installed, please execute the following line of code:

install.packages(c("ggplot2", "ggfortify", "reshape2", "ggExtra","RColorBrewer", "cowplot", "devtools"),dep=T)

devtools::install_github("JLSteenwyk/ggpubfigs")

1) Examine the toy data
Objective: Familiarize yourself with the data

i) We will use 'iris' data that is automatically loaded into the R environment. Take a look at the data structure by typing the name of the data structure and hitting 'enter.'

```
iris
```

This is probably more data to look at then we need at the moment. Take a look at the first few lines of the data frame using the 'head' command.

```
head(iris)
```

How many columns of data are there?
There are 5 columns

ii) Sometimes when working with data frames, we are only interested in working with one column. To examine only one column, type in the name of the data frame, a '$' sign, and then the column you wish to examine.

```
iris$Petal.Width
```

This is useful when we want to know statistics about a given column of data. For example, we can calculate the mean petal width using the following command.

```
mean(iris$Petal.Width)
```

What is the mean petal width?
1.199333

Using the same function, what is the mean petal length?
3.758

Another useful detail to know about the data frame is how many species of flowers are represented. To examine all unique values in the species column, use the levels function.

```
levels(iris$Species)
```

How many species of flowers are there in the dataset?
There are three species of flowers

2) Load the ggplot2 package and the foundation of making a figure
Objective: Learn to load and interact with a package

i) Before being able to use a package in R the package must be loaded in your session. To load a package, execute the following command

library(ggplot2)

Examine that ggplot2 has been properly loaded using the following command

sessionInfo()

You can see that under 'other attached packages,' ggplot2 is present.

If you come across a command you are not familiar with, you can obtain more information about it by putting a '?' before the command

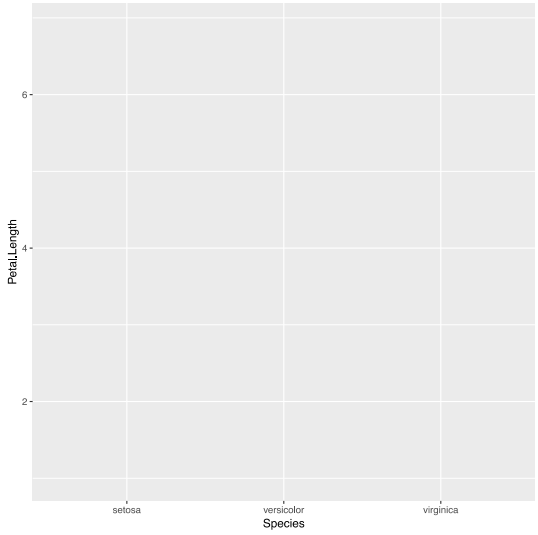
?sessionInfo()

Alternatively, google will be your best friend in accomplishing tasks in R.

ii) Now, let's get started on building the foundation of a figure by executing the following command

ggplot(iris, aes(x=Species, y=Petal.Length))

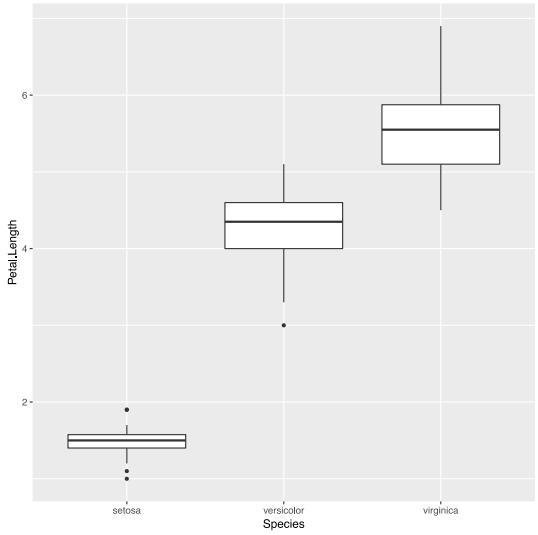
Information about species is depicted on the x-axis and petal length is depicted on the y-axis; however, no data is shown. Let's learn how to display the data as a box plot.



3) Qualitatively comparing distributions between species
Objective: display data as a boxplot

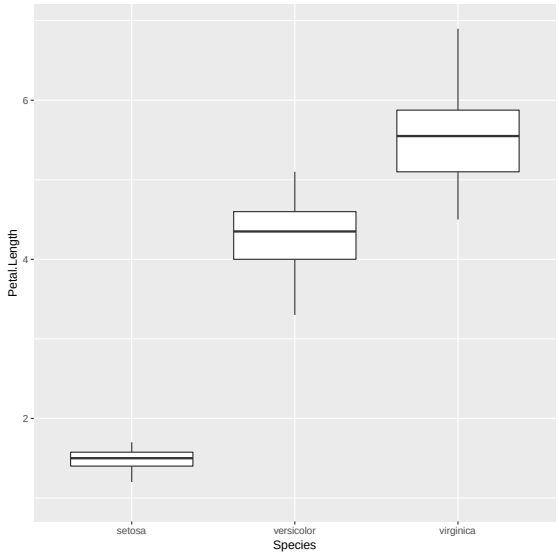
i) To display the data as a boxplot, add geom_boxplot() to the previous command.

ggplot(iris, aes(x=Species, y=Petal.Length)) + geom_boxplot()



During the presentation, we learned that boxplots can (at times) be deceiving about the underlying data. Therefore, it may be of interest to include the raw data as well as a scatterplot. To do so, we will use the geom_jitter() argument in our ggplot2 command.

ggplot(iris, aes(x=Species, y=Petal.Length)) + geom_boxplot(outlier.shape=NA) + geom_jitter(width=.25, alpha=0.5)



The addition of ‘outlier.shape=NA’ removes outliers so that outlier are not represented twice.

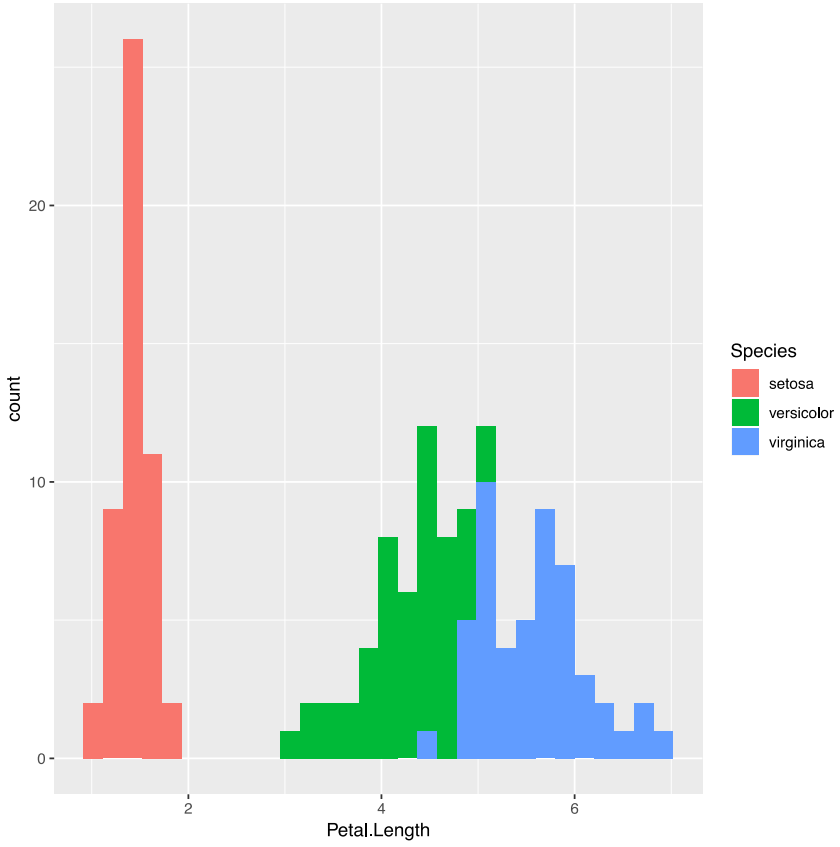
Determine what ‘width=.25’ and ‘alpha=0.5’ do by removing the commands or changing their parameters.

What does ‘width=.25’ do?
Changes the width of the jitterplot

What does ‘alpha=0.5’ do?
Changes the transparency of jitterplot data points

ii) An alternative to the previous commands would be a histogram. In the same way geom_boxplot() intuitively creates a boxplot, geom_histogram() will create a histogram. Plot a histogram of petal length using the following command:

ggplot(iris, aes(Petal.Length, fill=Species)) + geom_histogram()

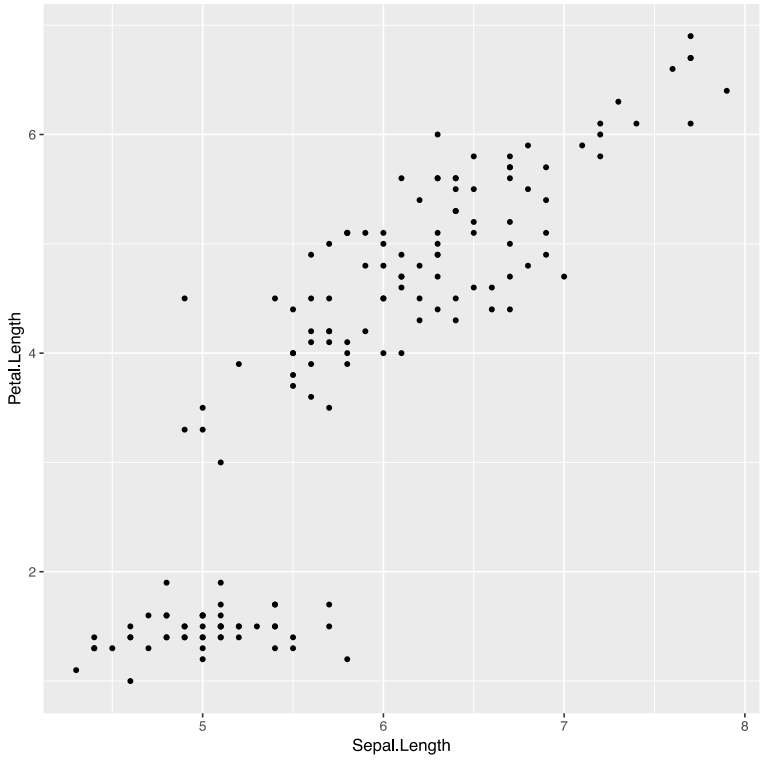


4) Examine the relationship between two variables

Objective: Learn to make a scatterplot and evaluate the direction of the relationship between two variables.

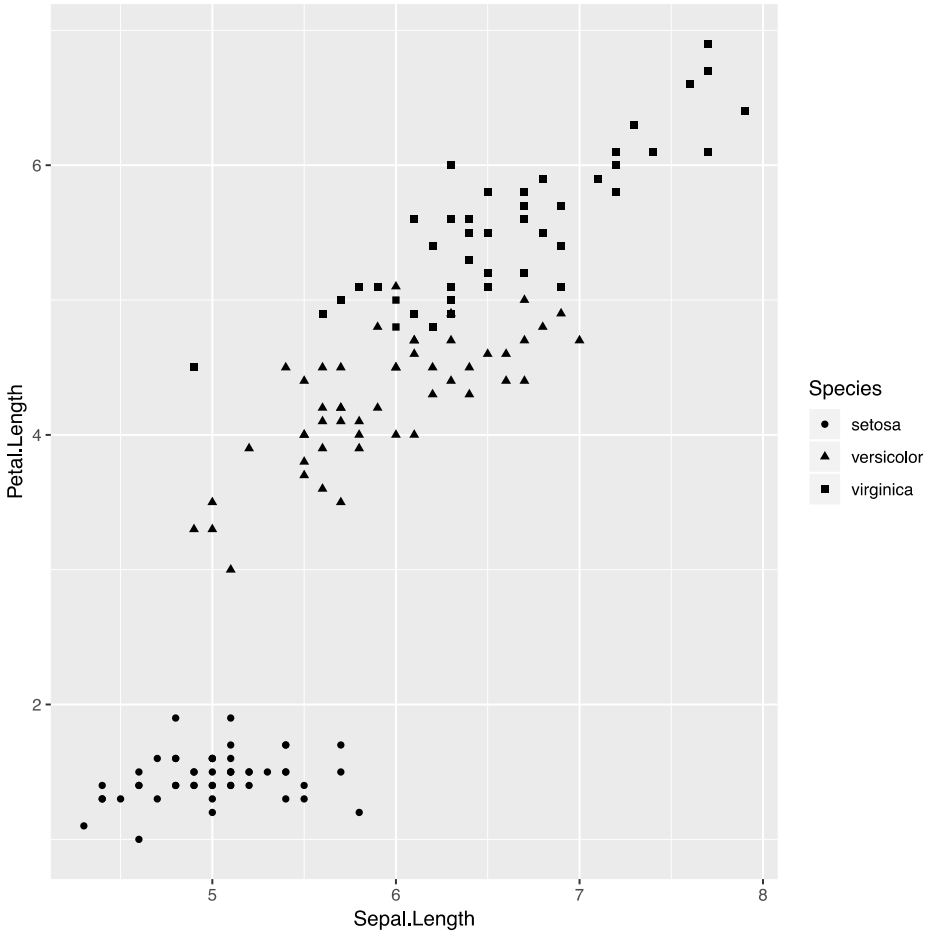
i) To add data points to the figure, execute the following command

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



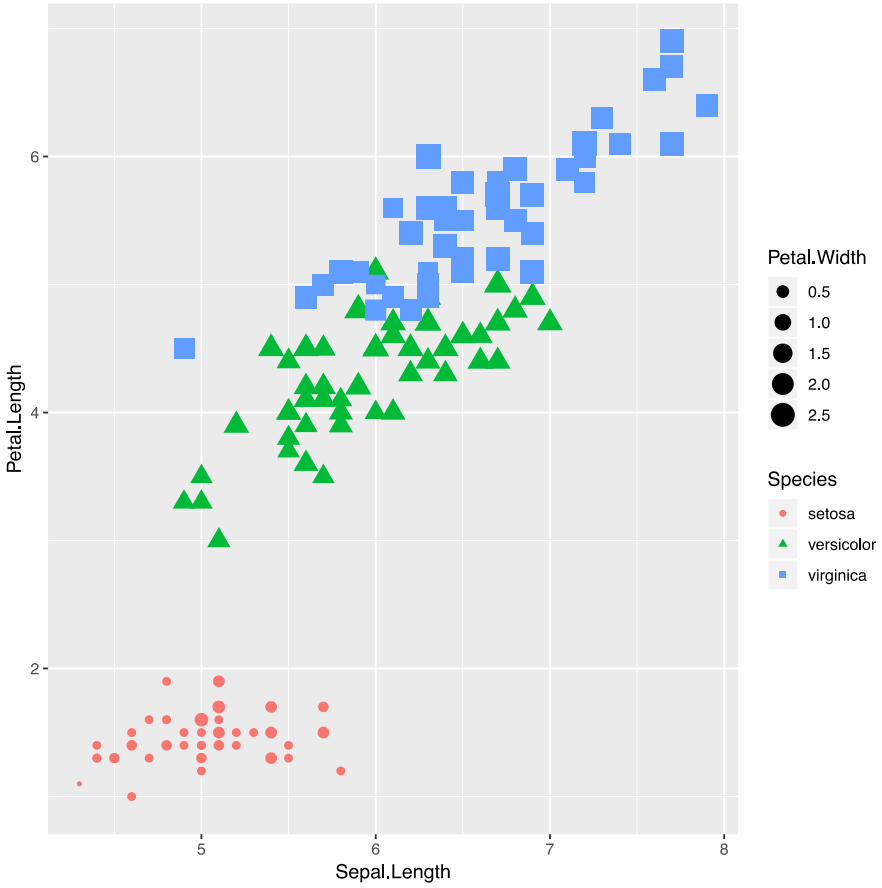
Previously, we noted that there are multiple species of flowers represented in the data set. To differentiate the flowers by giving each species their own shape, execute the following command.

ggplot(iris, aes(x=Sepal.Length, y=Petal.Length, shape=Species)) + geom_point()



However, other parameters can further differentiate the species and add in additional data. The next command will change the size of the data points according to the width of the flower petals as well as differentiate flower species by color

ggplot(iris, aes(Sepal.Length, Petal.Length, color=Species)) + geom_point(aes(size=Petal.Width, shape=Species))



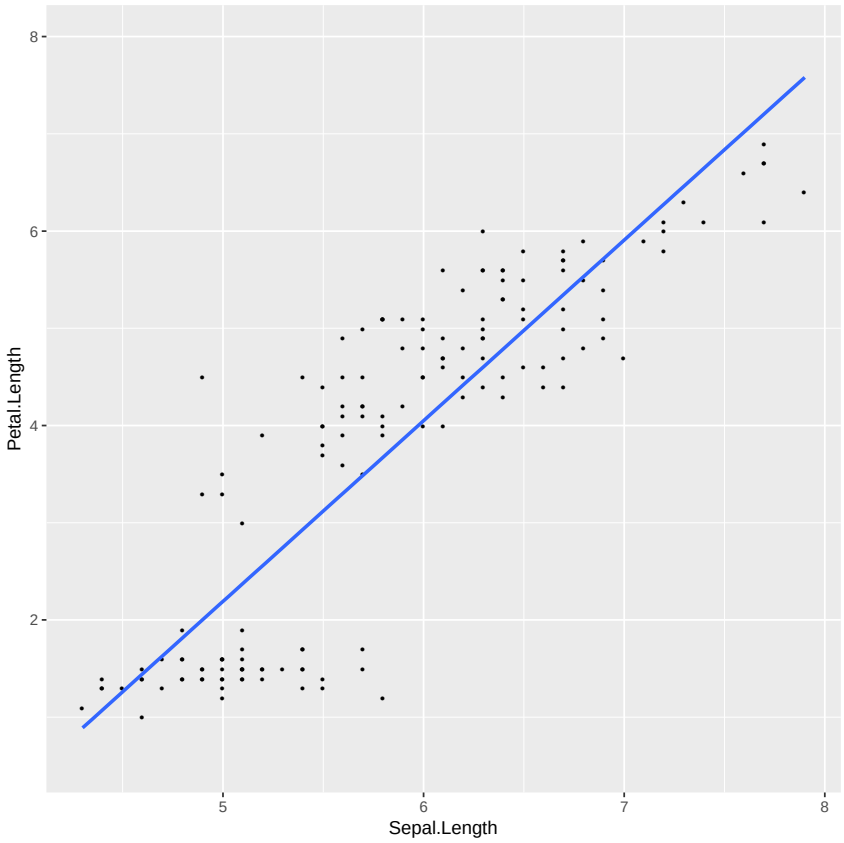
What part of the command specifies that each species will be colored differently?
color=Species

What part of the command specifies that size will reflect petal width?
size=Petal.Width

What part of the command species that each flower species will have a distinct shape?
shape=Species

ii) Our data suggests that there is a linear relationship between sepal and petal length. Let's examine this relationship closer by executing the following command.

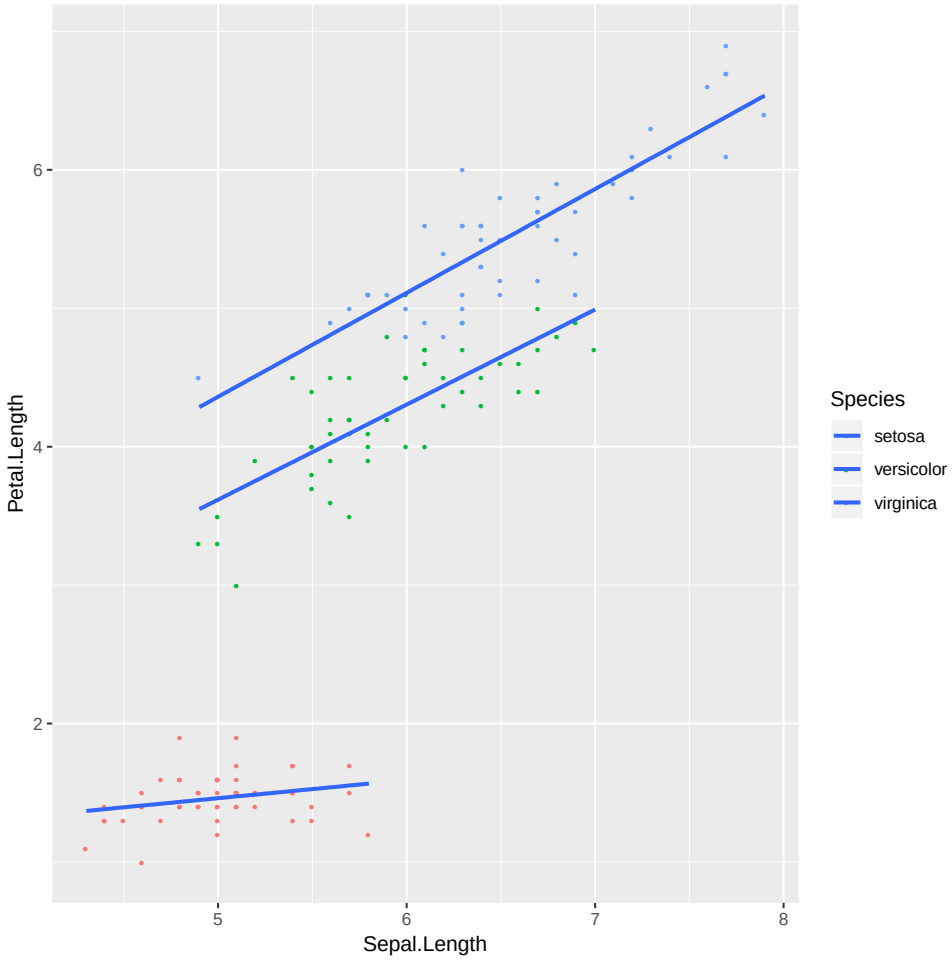
ggplot(iris, aes(Sepal.Length, Petal.Length)) + geom_point() + geom_smooth(method='lm')



You may have noticed that this command no longer differentiates the different species of flowers. Build upon the previous command to differentiate between species using the following command.

ggplot(iris, aes(Sepal.Length, Petal.Length, fill=Species)) + geom_point(aes(color=Species)) +

geom_smooth(method='lm')



iii) When examining the relationship between two variables, it may also be useful to provide information about the distribution of variables along the x- and y-axes. To plot distributions along the x- and y-axes, we will use the R package, ggExtra.
Please load the ggExtra package before moving on using the library() function.

To do so, first save the previous command to a variable called ‘p’

p<-ggplot(iris, aes(Sepal.Length, Petal.Length, fill=Species)) + geom_point(aes(color=Species)) + geom_smooth(method='lm')

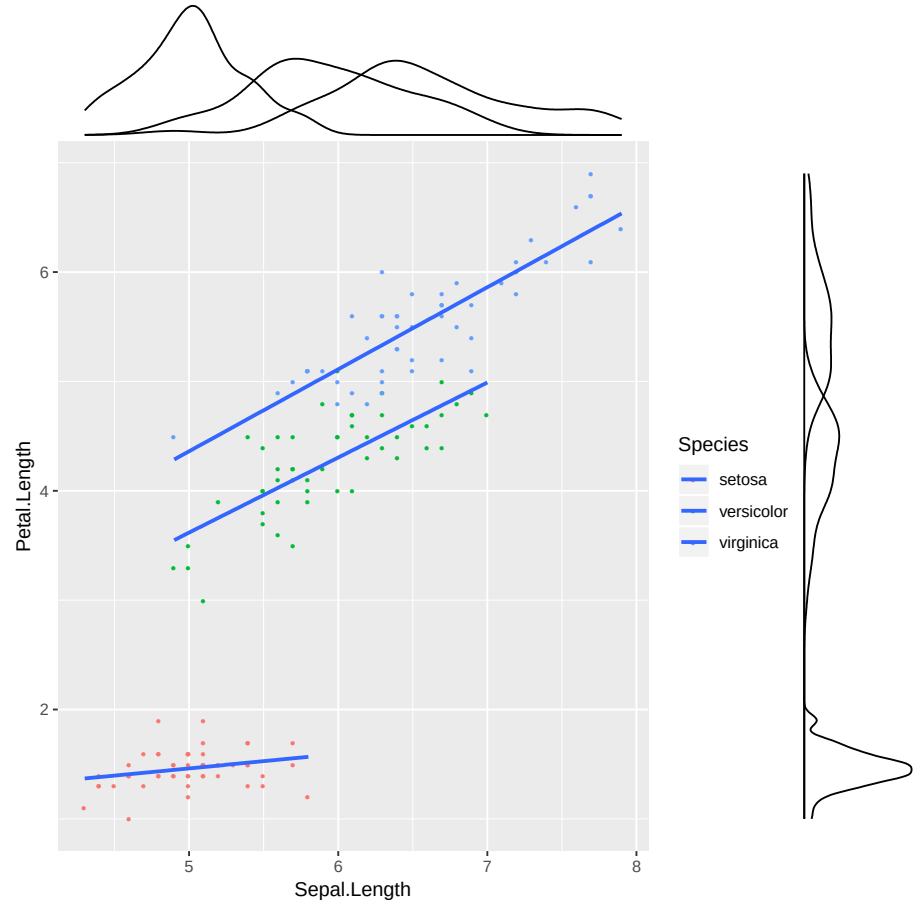
Type in ‘p’ and then hit enter to show that you have saved the figure to the variable ‘p’

p

Build upon ‘p’ using the following command

library(ggExtra)

ggMarginal(p, type = "density", groupFill=TRUE)



5) Stylize your figure
Objective: learn various ways to stylize your figure.

i) Although we have built a beautiful figure, there is still a lot that could be done to improve the visualization. For example, you may have noticed that many figures in 'big' journals typically have a white background. To use a white background and showcase other pre-made themes, we will use various ggplot2 functions that will do a lot of the work for us.

Start with the following command:

p<-ggplot(iris, aes(Sepal.Length, Petal.Length, fill=Species)) + geom_point(aes(color=Species)) + geom_smooth(method='lm')

test out different themes by executing the following commands one-by-one.

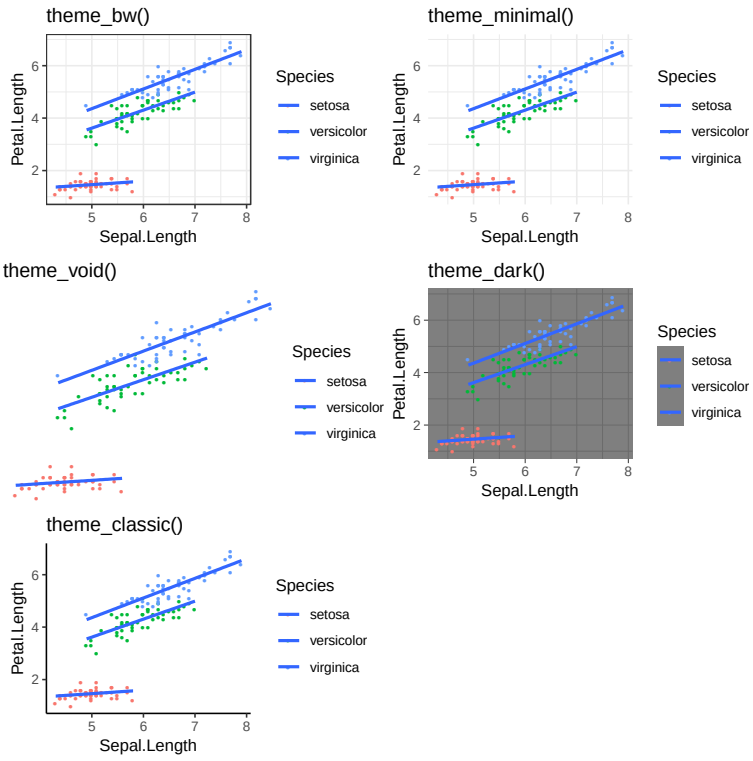
p + theme_bw()

p + theme_minimal()

p + theme_void()

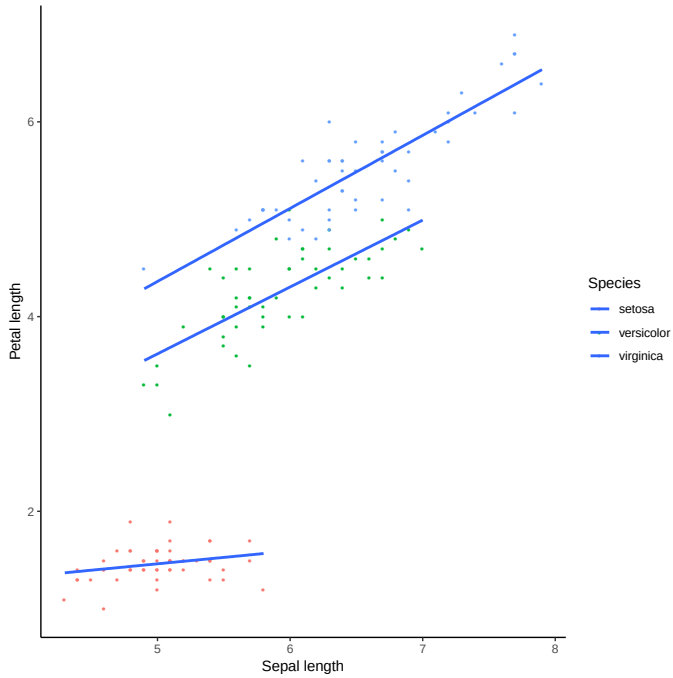
p + theme_dark()

p + theme_classic()



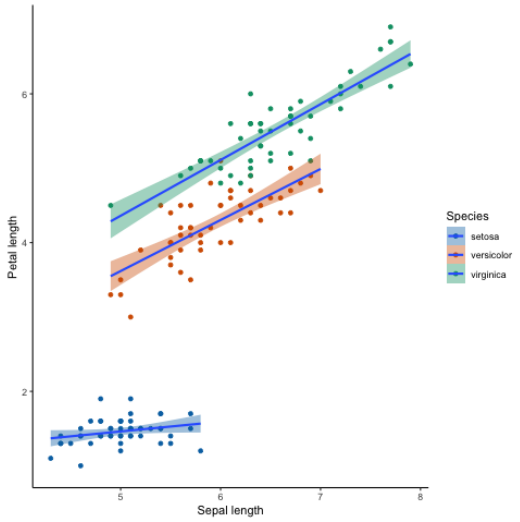
ii) Let's build upon the last command but update our axis labeling.

p + xlab(“Sepal length”) + ylab(“Petal length”) + theme_classic()



iii) Next, we may want to change the color scheme especially because green and red are not color-blind accessible. To do so, we will use the package RColorBrewer. Load RColorBrewer using the library() function. After that, we will use the *ito_seven* color palette from ggpubfigs to automatically color our plot.

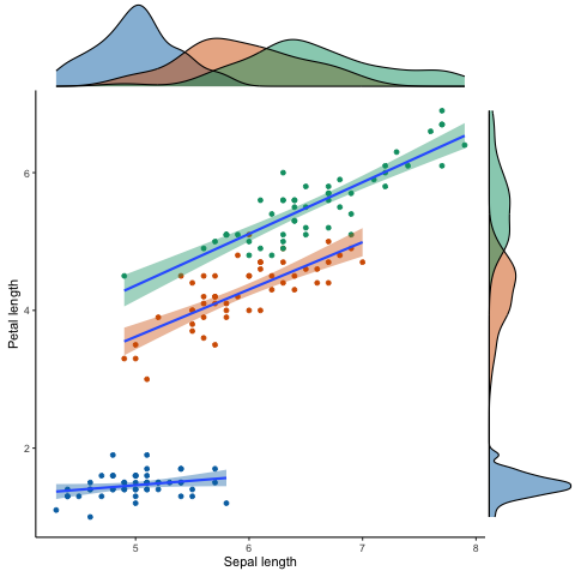
p + xlab("Sepal length") + ylab("Petal length") + theme_classic() + scale_fill_manual(values = friendly_pal("ito_seven")) + scale_color_manual(values = friendly_pal("ito_seven"))



iv) Lastly, let's add in the marginal plot but remove the legend before doing so to create a nice, clean, and clear figure. To do so, we will have to execute two commands. Note, the meaning of colors should be specified elsewhere which, can be done post-production.

p<-p + xlab("Sepal length") + ylab("Petal length") + theme_classic() + scale_color_manual(values = friendly_pal("ito_seven")) + scale_fill_manual(values = friendly_pal("ito_seven")) + guides(fill=FALSE, color=FALSE)

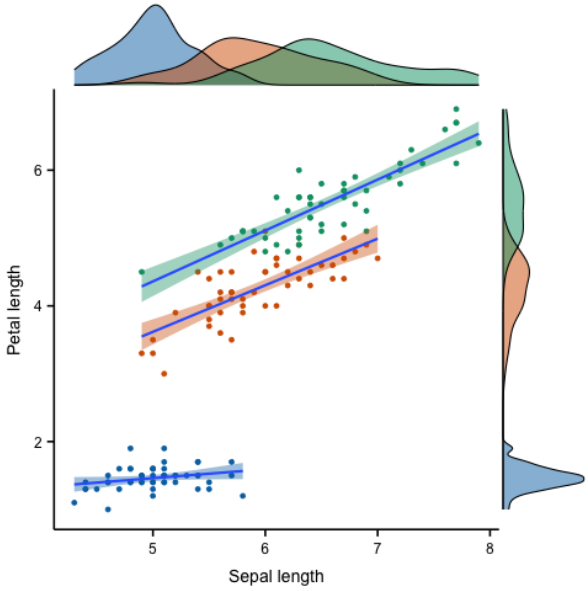
ggMarginal(p, type = "density", groupFill=TRUE)



v) I find it challenging to see the font along the axis. Let's use the ggpubfigs theme theme_simple() to make it clearer.

p<-p + xlab("Sepal length") + ylab("Petal length") + theme_simple() + scale_color_manual(values = friendly_pal("ito_seven")) + scale_fill_manual(values = friendly_pal("ito_seven")) + guides(fill=FALSE, color=FALSE)

ggMarginal(p, type = "density", groupFill=TRUE)



vi) Now that we have spent time making this figure, let's learn how to export the figure onto our desktop. To do so, execute the following commands which,
first, specify the destination of the pdf file you will make (change this for your computer)
second, generate the graph (note, no figure will appear on your computer screen)
and third, tell R that you are done developing your figure.

pdf("path_to_desired_location_to_make_the_file/name_of_file.pdf")

ggMarginal(p, type = "density", groupFill=TRUE)

dev.off()

A figure in the location you specified in command one should now appear on your computer – check it out!

6) Display multiple dimensions of data
Objective: Modify a data frame for displaying multiple dimensions of a data set.

i) As previously established, this data has a total of 5 dimensions each represented by a column in the data frame. To display data about all five dimensions at once, we will combine previous commands with new ones.

First, we need to reshape our data set to make it amendable to this sections goals. To do so, execute the following command which, uses the melt function from reshape2. If you get an error, did you load reshape2? See '2) Load the ggplot2 package and the foundation of making a figure' to remind yourself how to load a package.

```
library(reshape2)
iris.df<-melt(iris, id.vars="Species")
```

Take a look at the new data frame using the head() function.

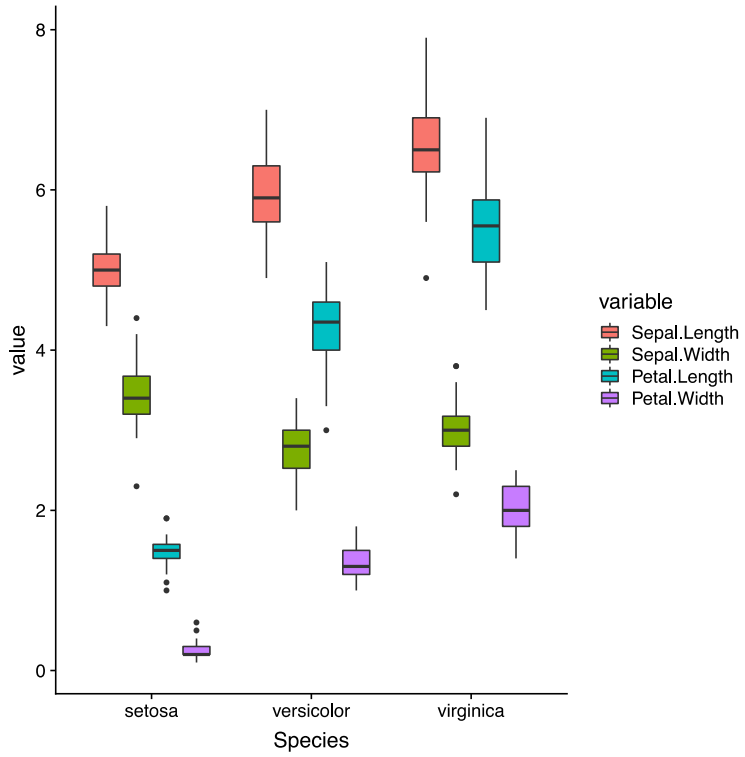
```
head(iris.df)
```

```
> head(iris.df)
  Species     variable value
1  setosa Sepal.Length   5.1
2  setosa Sepal.Length   4.9
3  setosa Sepal.Length   4.7
4  setosa Sepal.Length   4.6
5  setosa Sepal.Length   5.0
6  setosa Sepal.Length   5.4
```

How has the data frame changed? Understanding how data frames are organized is key to learning the basics of computer programming. I encourage you to explore the data frame further.
Similar to the previous data frame, all species are in one column. In contrast to the previous data frame, all variables and their values are in two columns rather than each variable having their own column.

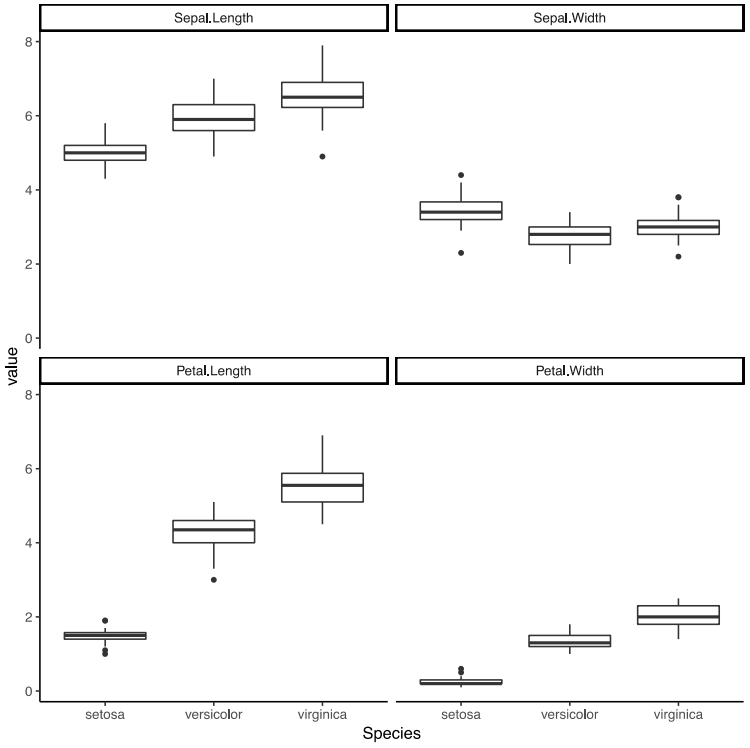
ii) To plot data for all variables, execute the following command

```
ggplot(iris.df, aes(Species, value, fill=variable)) + geom_boxplot() + theme_classic()
```

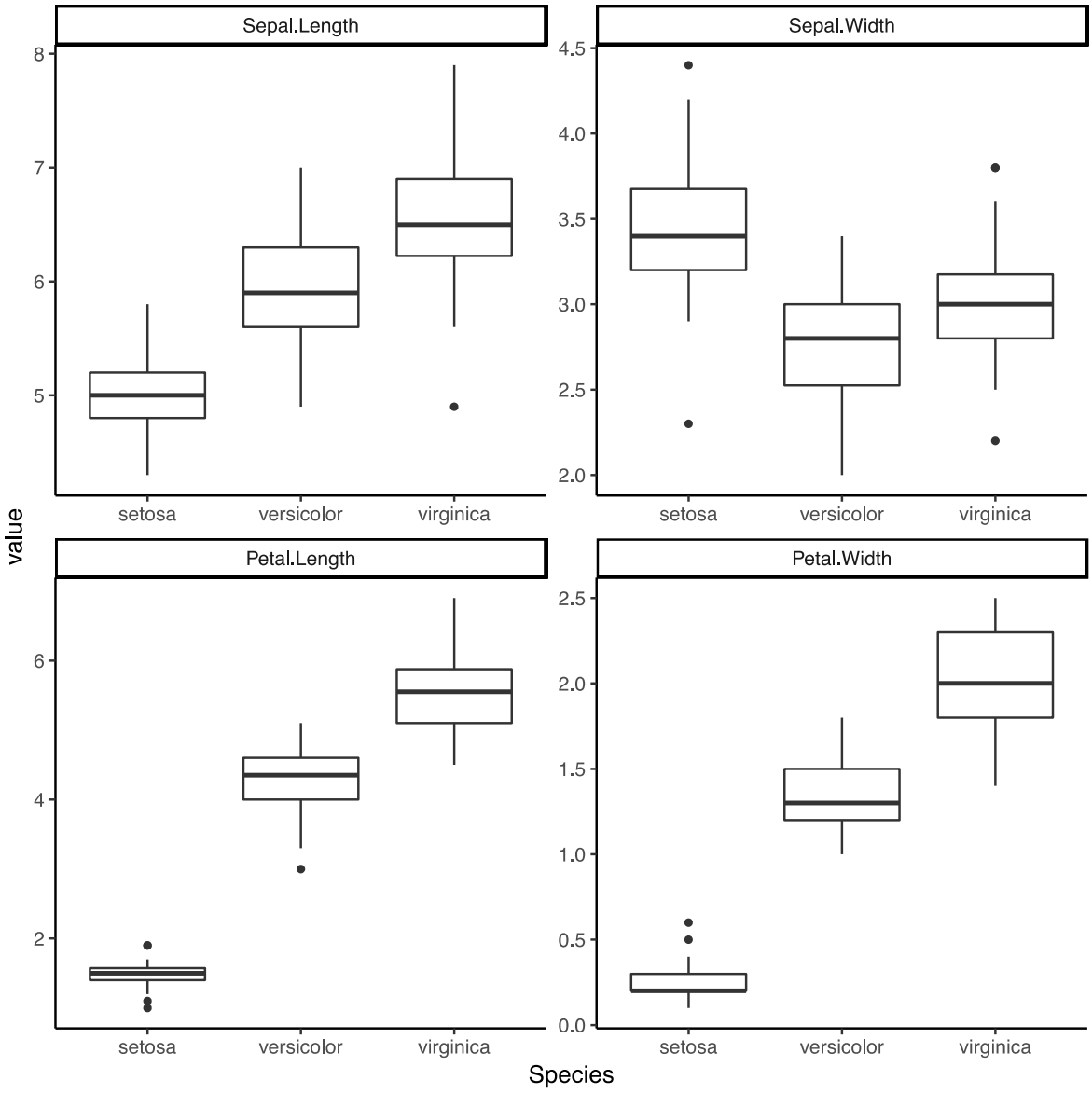


An alternative approach to display all the variables at once is the following

ggplot(iris.df, aes(Species, value)) + geom_boxplot() + theme_classic() + facet_wrap(~variable)



Modify the last part of the ggplot2 argument to be *facet_wrap(~variable, scales="free")* to allow each variable to have their own y-axis ranges.



Between the grouped box plots and the faceted box plots, which did you like more and why?
Opinion

7) Summarizing multiple dimensions of data
Objective: Summarize all four dimensions of data across the three species of flowers.

i) As previously established, this data has a total of 5 dimensions each represented by a column in the data frame. We will use principal component analysis (PCA) to summarize all five dimensions in one plot.

Again, most data analyses require different data structures to be inputted. For PCA, we do not want to consider 'Species' as another dimension so we will need to subset the data frame to exclude 'Species' with the following command:

iris.df <- iris[c(1, 2, 3, 4)]

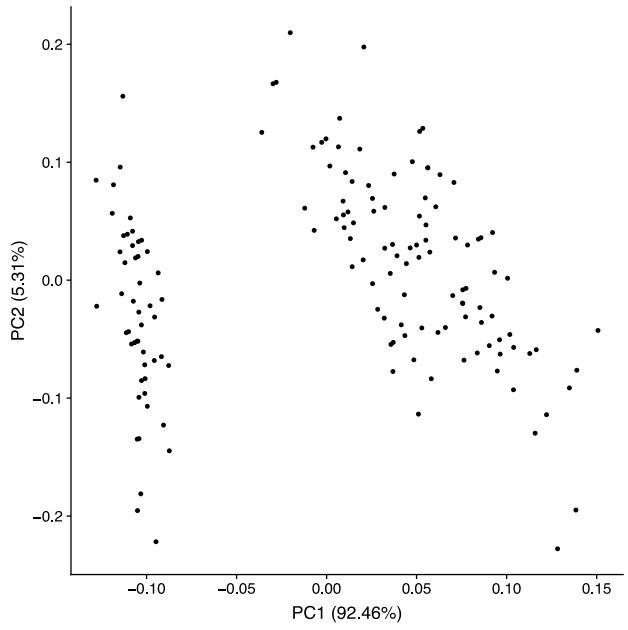
Take a look at the new data frame using the head() function.

head(iris.df)

```
> head(iris.df)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
```

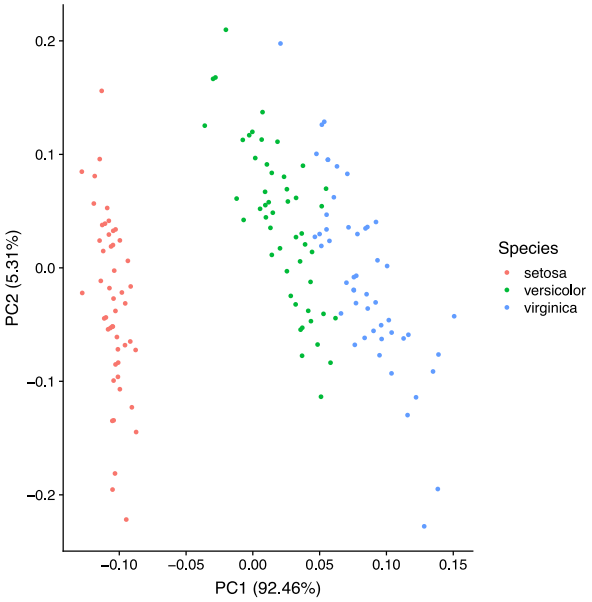
ii) Conduct and plot PCA on the iris.df data frame using the prcomp() and autoplot() function from ggfortify. Note, if you get an error message, it is probably because ggfortify is not loaded.

autoplot(prcomp(iris.df)) + theme_classic()



Differentiate the species from one another by building upon the previous command and executing the following command:

autoplot(prcomp(iris.df), data=iris, colour="Species") + theme_classic()



How much variance between the individuals is explained along principal component 1?
92.46%

How much variance between the individuals is explained along principal component 2?
5.31%

Which two species are more similar to one another?
Virginica and versicolor

8) Creating multi-panel figures
Objective: Create multi-panel figures

i) Load the cowplot package

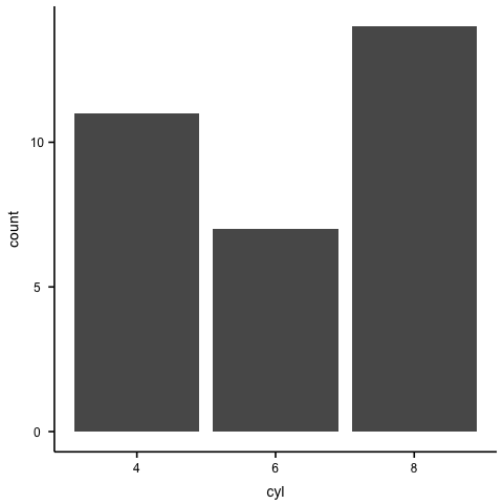
library(cowplot)

ii) Take a look at the mtcars dataset and make a bar plot of counts per cyl

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

ggplot(mtcars, aes(cyl)) + geom_bar() + theme_simple() +

scale_x_continuous(breaks=c(unique(mtcars$cyl)))



What does scale_x_ scale_x_continuous(breaks=c(unique(mtcars$cyl))) do?
scale_x_continuous enables users to manually define what values to show on the x-axis. Here, we specify to only plot unique values that are present in the mtcars dataset cyl column.

iii) Save the resulting plot to a variable named “p_cyl”, which stands for “plot_cyl”.

p_cyl<-ggplot(mtcars, aes(cyl)) + geom_bar() + theme_simple() +

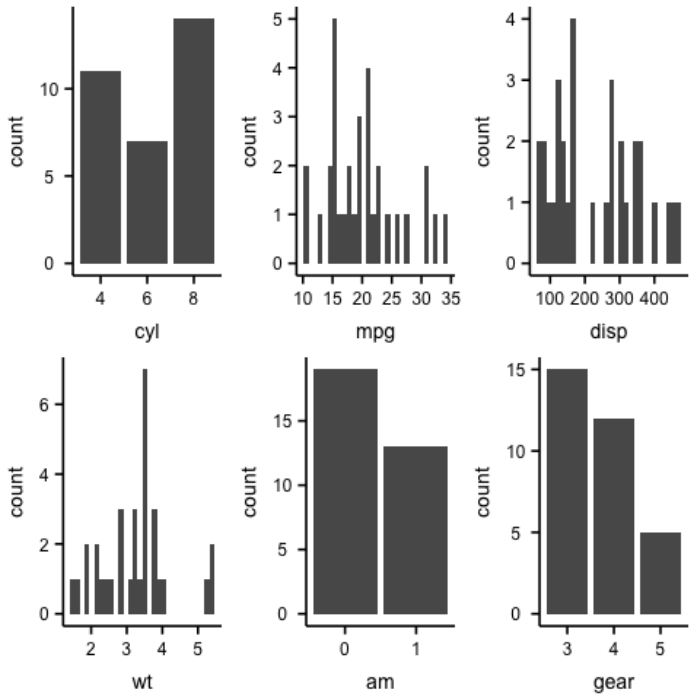
scale_x_continuous(breaks=c(unique(mtcars$cyl)))

iv) Create additional panels that depict histograms for columns mpg, disp, and wt. Also, make another bar plot for columns am and gear. Use the same naming scheme for the resulting variables as above.

```
p_mpg<-ggplot(mtcars, aes(mpg)) + geom_histogram() + theme_simple()
p_disp<-ggplot(mtcars, aes(disp)) + geom_histogram() + theme_simple()
p_wt<-ggplot(mtcars, aes(wt)) + geom_histogram() + theme_simple()
p_am<-ggplot(mtcars, aes(am)) + geom_bar() + theme_simple() +
scale_x_continuous(breaks=c(unique(mtcars$am)))
p_gear<-ggplot(mtcars, aes(gear)) + geom_bar() + theme_simple() +
scale_x_continuous(breaks=c(unique(mtcars$gear)))
```

iv) Use the plot_grid() function from cowplot to create a multi-panel figure.

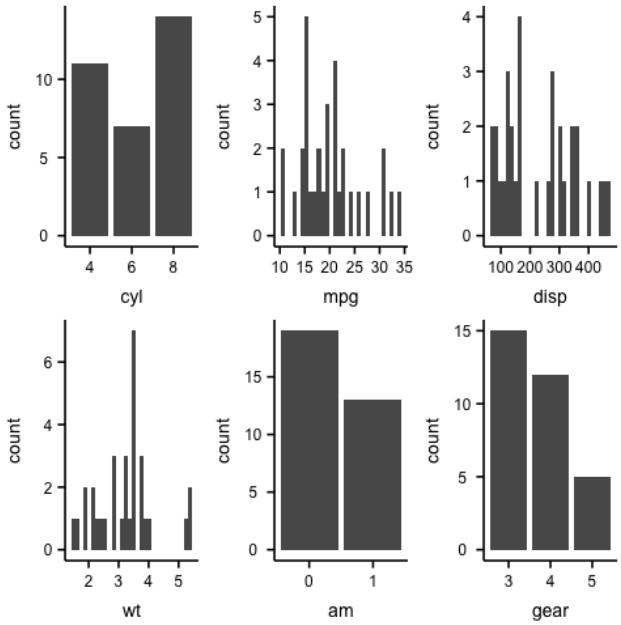
```
plot_grid(p_cyl, p_mpg, p_disp, p_wt, p_am, p_gear)
```



v) Great – we made a multi-panel figure! However, there are some small issues. For example, the axes are not aligned between each of the panels. For example, the y-axis is not perfectly aligned for the two left-most panels.

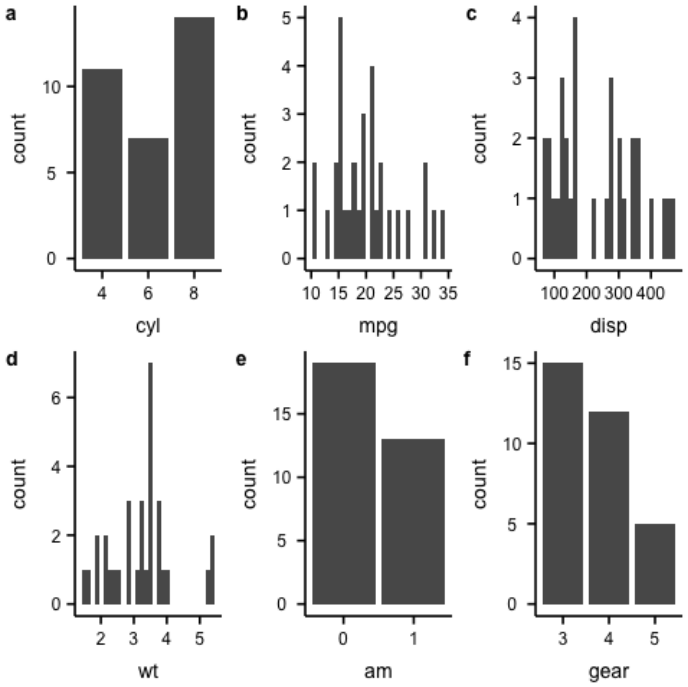
To align the various panels, set the “align” parameter to “hv”, which stands for horizontal and vertical; this way, all horizontal and vertical axes will be aligned.

plot_grid(p_cyl, p_mpg, p_disp, p_wt, p_am, p_gear, align="hv")



vi) Now, let’s add labels to each panel

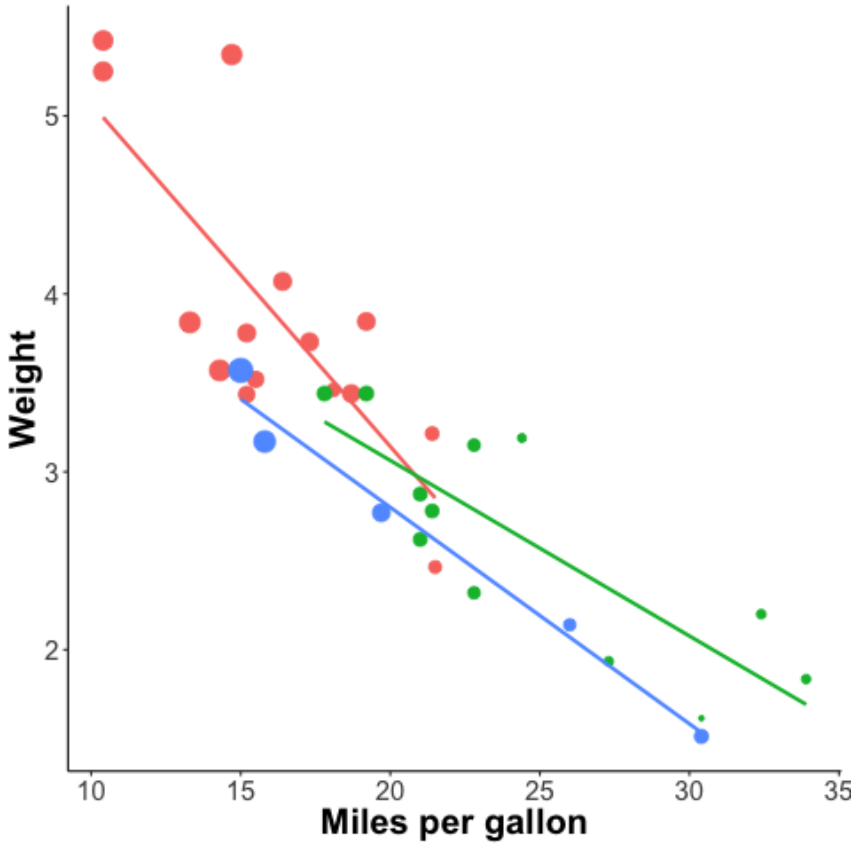
plot_grid(p_cyl, p_mpg, p_disp, p_wt, p_am, p_gear, align="hv", labels = "auto")



vii) Change labels = "auto" to be labels = "AUTO". What happened?

The panel labels are now capitalized.

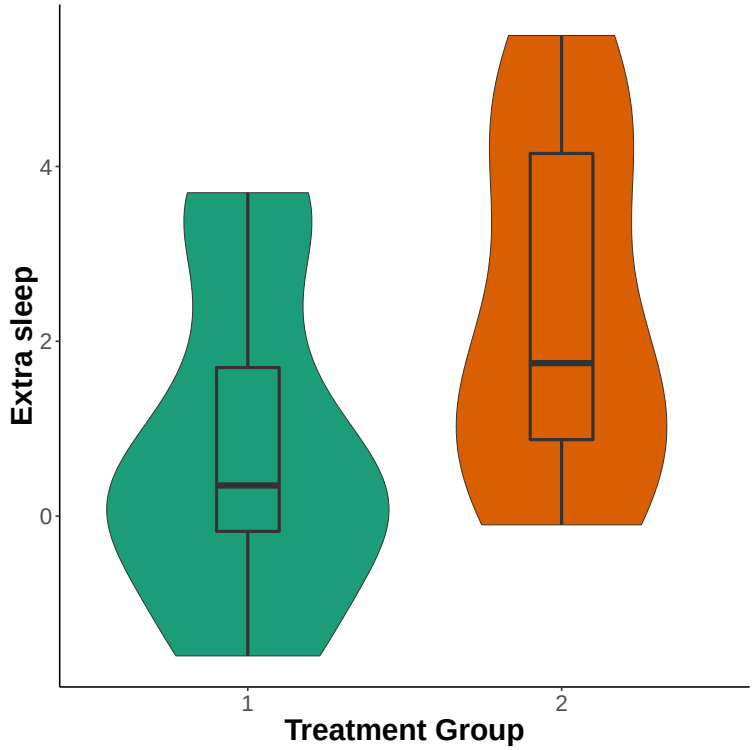
Challenge #1
Recreate this figure using the mtcars dataset, which is already provided in R. I would recommend getting familiar with the dataset first.



Note, the following factors about the figure
x-axis: mpg data (miles per gallon)
y-axis: wt data (weight)
size: hp data (horsepower)
color: gear (how many gears the car has)
The x- and y-axis labels have been changed
The font sizes of the axes have been enlarged to 15
The font sizes of the axis titles have been changed to bold and changed to size 20

```
ggplot(mtcars, aes(mpg, wt, size=hp, color=as.factor(gear))) +
geom_point(aes(size=hp,colour=as.factor(gear))) + geom_smooth(method='lm', se=FALSE) +
theme_classic() + xlab("Miles per gallon") + ylab("Weight") + theme(axis.text=element_text(size=15),
axis.title=element_text(size=20,face="bold")) + theme(legend.position = "none")
```

Challenge #2
Recreate this figure using the sleep dataset already provided in R. I would recommend getting familiar with the dataset first.

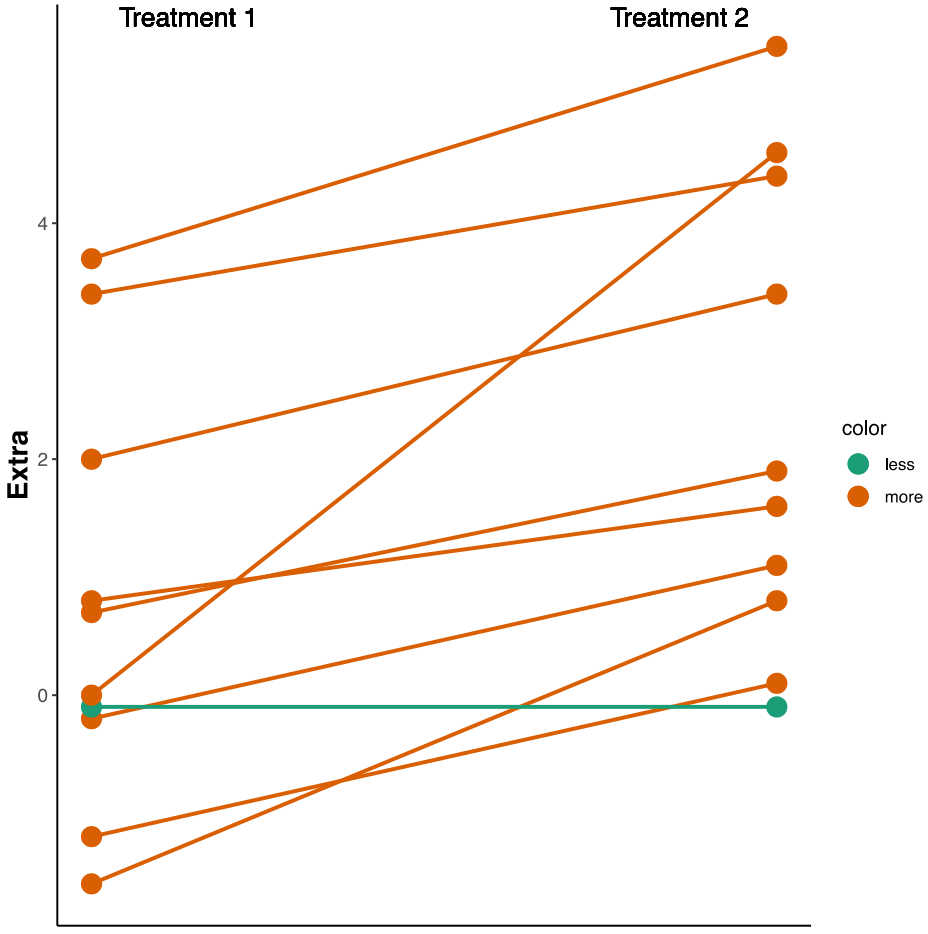


Note, the following factors about the figure
x-axis: group data
y-axis: extra data
color: group data (how many gears the car has) – Dark2 RColorBrewerPallete
Line widths of the violin plot have been set to 0
Line widths of the boxplot have been set to 1
The x- and y-axis labels have been changed
The font sizes of the axes have been enlarged to 15
The font sizes of the axis titles have been changed to bold and changed to size 20

```
ggplot(sleep, aes(group, extra, fill=group)) + geom_violin(lwd=0) + geom_boxplot(width=.2, fill="white",alpha=.8, lwd=1) + scale_fill_brewer(palette="Dark2") + theme_classic() + xlab("Treatment Group") + ylab("Extra sleep") + theme(axis.text=element_text(size=15), axis.title=element_text(size=20,face="bold")) + theme(legend.position = "none")
```

Challenge #3

Recreate this figure using the sleep dataset which, is already provided in R.
Note, this command requires reshaping the dataset.



Note, the following factors about the figure
x-axis: group data
y-axis: extra data
color: color is conditionally dependent on whether or not the treatment resulted in an increase or decrease in total sleep. Here, less sleep is green and more sleep is orange from the Dark2 RColorBrewerPallete
Line widths have been set to 1
Point size has been set to 5
Y axis label has been changed. X axis label, ticks, and text have been removed.
The font sizes of the axes have been enlarged to 10
The font sizes of the axis titles have been changed to bold and changed to size 15
Text to differentiate treatment 1 and treatment 2 have been added to the figure

```
> sleep.df<-dcast(data = sleep,formula = ID~group,fun.aggregate = sum,value.var = "extra")

> sleep.df$color<-ifelse((sleep.df$`1` - sleep.df$`2`) < 0, "more", "less")

> ggplot(sleep.df) + geom_segment(aes(x=1, xend=2, y=`1`, yend=`2`, col=color), size=1.0,
show.legend=F) + geom_point(aes(x=1, y=`1`, col=color), size=5) + geom_point(aes(x=2, y=`2`,
col=color), size=5) + scale_color_brewer(palette="Dark2") + theme_classic() + ylab("Extra") +
theme(axis.text.x = element_blank(), axis.title.x=element_blank(), axis.ticks.x=element_blank(),
axis.text=element_text(size=10), axis.title=element_text(size=15,face="bold")) +
geom_text(label="Treatment 1", x=1, y=.25+(max(sleep.df$`1`, sleep.df$`2`)), hjust=-.20, size=5) +
geom_text(label="Treatment 2", x=2.00, y=.25+(max(sleep.df$`1`, sleep.df$`2`)), hjust=1.2, size=5)
```