Day 3: Continued Data Visualization & Data tidying

How are we feeling today?

R Terminology

- **Source pane:** Where we type code to be saved
- **Console:** Where code outputs appear directly beneath the input you type in the source pane. Also where you can get packages and functions explained by looking them up with a ? or ?? in front of the name.
- **Environment:** Where variables are stored
- **Variable:** Labeled, stored data that can be "called" later
- **"Calling" variable/function:** Inserting a name into your code that matches a variable in your environment
- **library():** Used to load packages
- **Vector:** A list of data, all the same type (for example, a list of all numbers or a list of all names)

Types of plots to use for different types of data

Categorical

Bar chart
Pie chart
Box and Whisker Plot

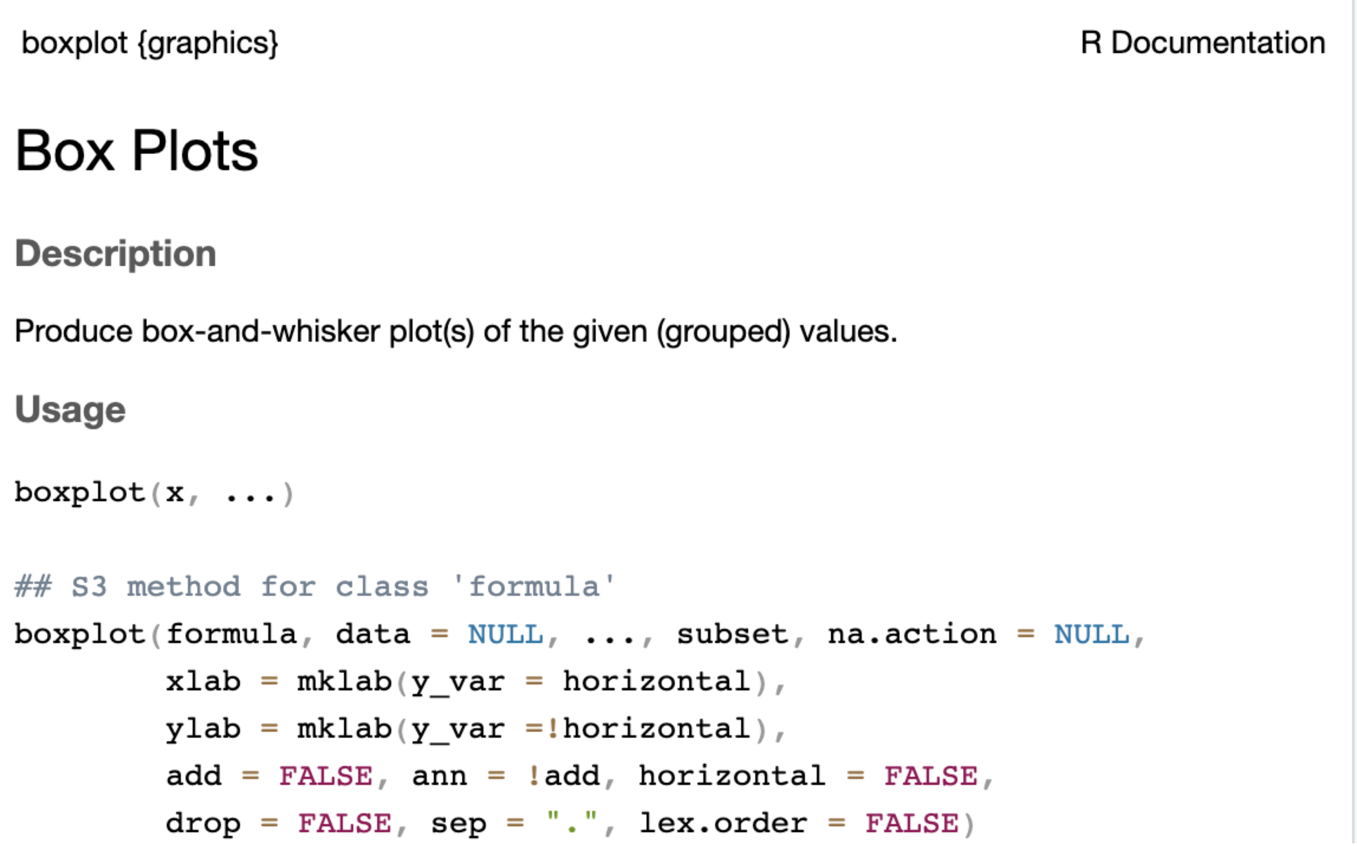
Continuous

Line graph
Scatter Plot
Histogram

How do we find out how to use an R function like boxplot()?

Nobody has responded yet.

Hang tight! Responses are coming in.



boxplot {graphics} R Documentation

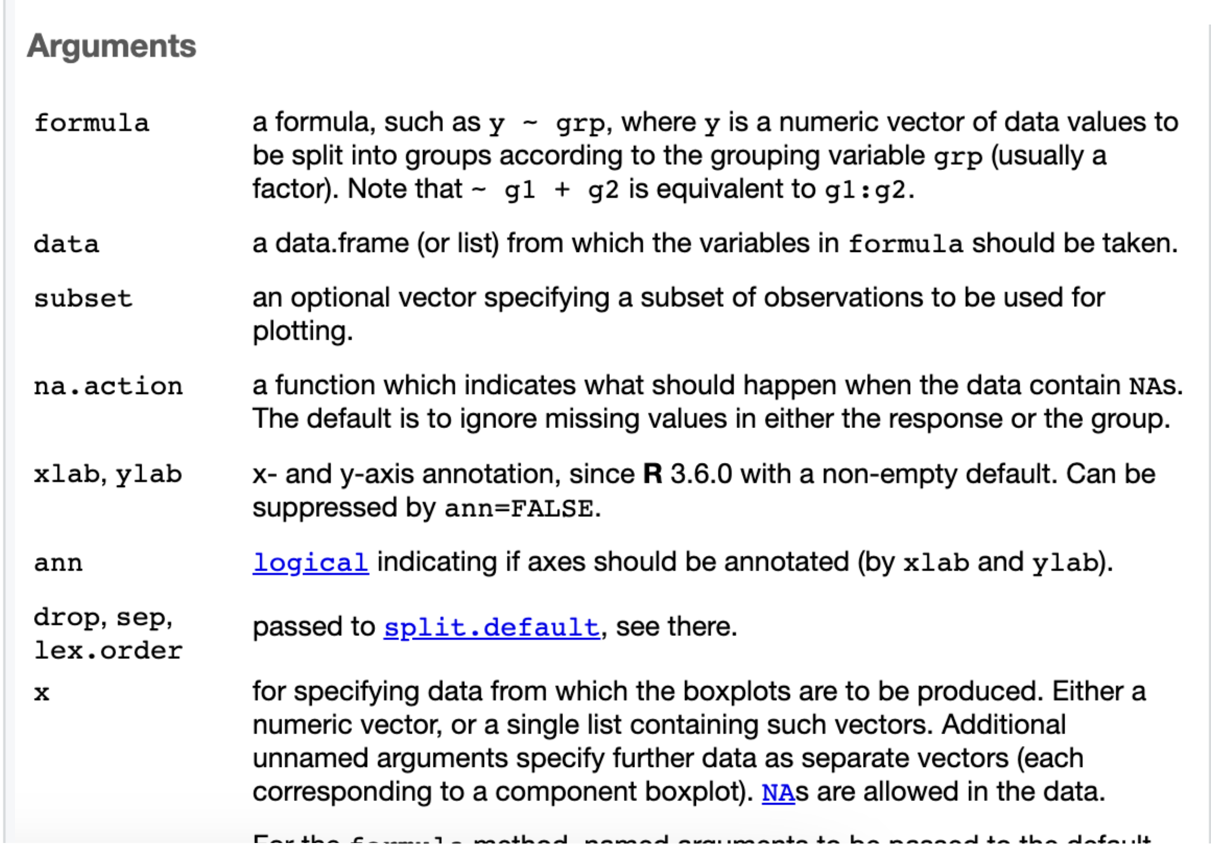
Box Plots

Description

Produce box-and-whisker plot(s) of the given (grouped) values.

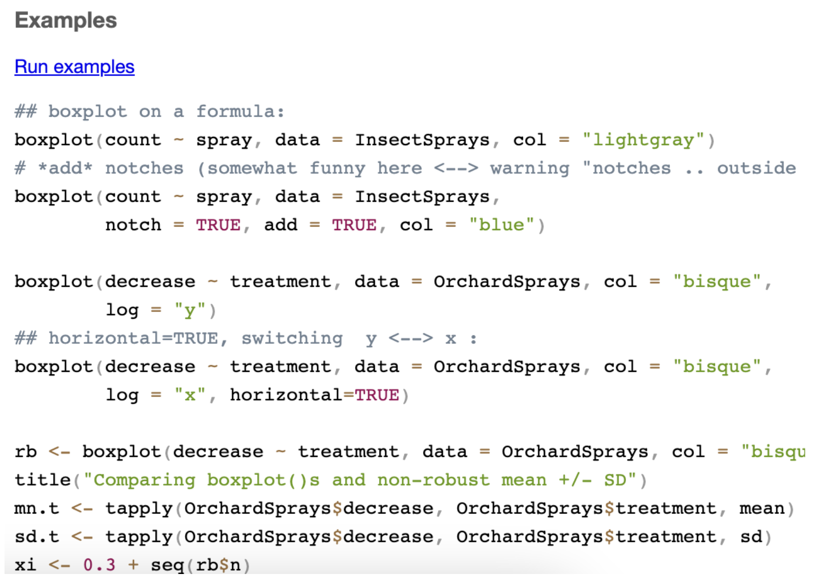
Usage

```
boxplot(x, ...)

## S3 method for class 'formula'
boxplot(formula, data = NULL, ..., subset, na.action = NULL,
        xlab = mklab(y_var = horizontal),
        ylab = mklab(y_var =!horizontal),
        add = FALSE, ann = !add, horizontal = FALSE,
        drop = FALSE, sep = ".", lex.order = FALSE)
```



Arguments

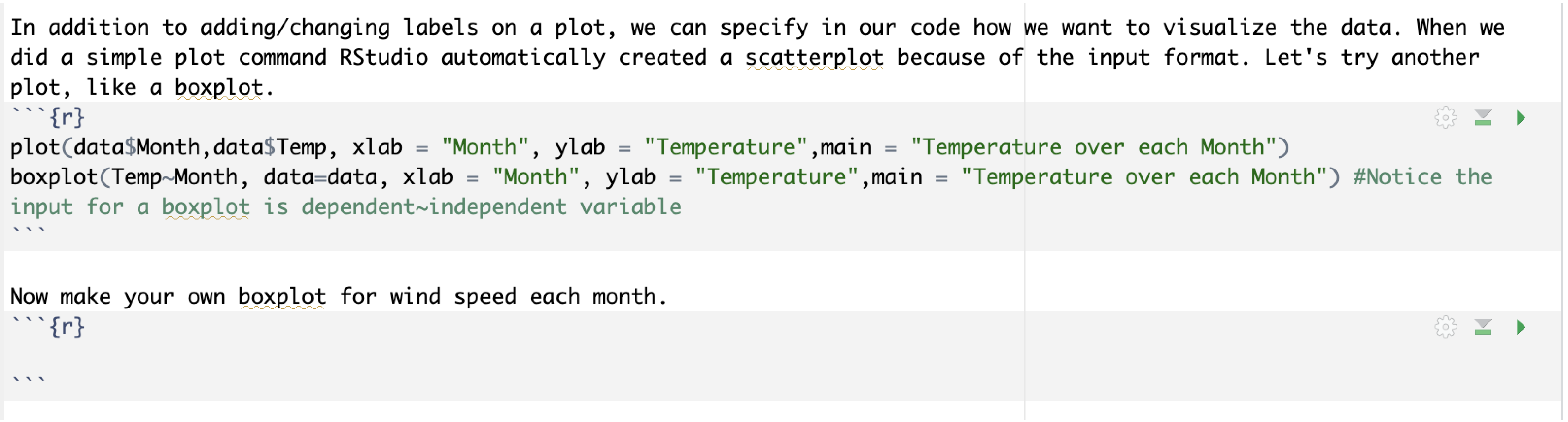
`formula`	a formula, such as `y ~ grp`, where `y` is a numeric vector of data values to be split into groups according to the grouping variable `grp` (usually a factor). Note that `~ g1 + g2` is equivalent to `g1:g2`.
`data`	a data.frame (or list) from which the variables in `formula` should be taken.
`subset`	an optional vector specifying a subset of observations to be used for plotting.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is to ignore missing values in either the response or the group.
`xlab, ylab`	x- and y-axis annotation, since **R** 3.6.0 with a non-empty default. Can be suppressed by `ann=FALSE`.
`ann`	`logical` indicating if axes should be annotated (by `xlab` and `ylab`).
`drop, sep, lex.order`	passed to `split.default`, see there.
`x`	for specifying data from which the boxplots are to be produced. Either a numeric vector, or a single list containing such vectors. Additional unnamed arguments specify further data as separate vectors (each corresponding to a component boxplot). `NA`s are allowed in the data.
	For the `formula` method, named arguments to be passed to the default



Examples

Run examples

```
## boxplot on a formula:
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")
# *add* notches (somewhat funny here <--> warning "notches .. outside
boxplot(count ~ spray, data = InsectSprays,
        notch = TRUE, add = TRUE, col = "blue")

boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisque",
        log = "y")
## horizontal=TRUE, switching  y <--> x :
boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisque",
        log = "x", horizontal=TRUE)

rb <- boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisqu
title("Comparing boxplot()s and non-robust mean +/- SD")
mn.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, mean)
sd.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, sd)
xi <- 0.3 + seq(rb$n)
```



In addition to adding/changing labels on a plot, we can specify in our code how we want to visualize the data. When we did a simple plot command RStudio automatically created a scatterplot because of the input format. Let's try another plot, like a boxplot.

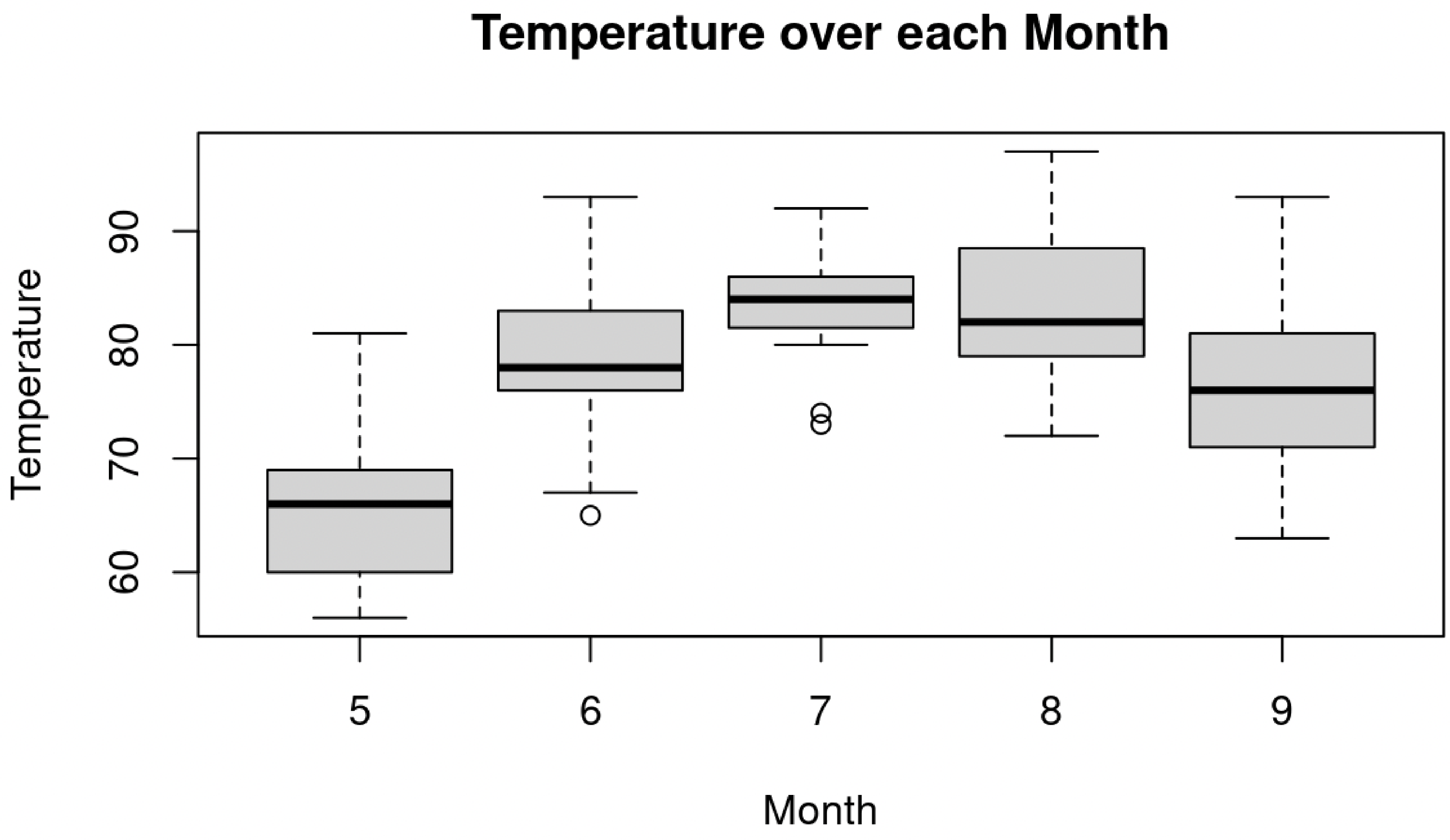
~~~
```{r}
plot(data$Month,data$Temp, xlab = "Month", ylab = "Temperature",main = "Temperature over each Month")
boxplot(Temp~Month, data=data, xlab = "Month", ylab = "Temperature",main = "Temperature over each Month") #Notice the input for a boxplot is dependent~independent variable
```
~~~

Now make your own boxplot for wind speed each month.

~~~
```{r}

```
~~~

Steps for making a plot:

1. What are your independent and dependent variables? Remember, the independent variable will go on your x-axis, dependent variables go on your y-axis. Dependent variables are the response you are measuring.
2. What kind of variables are you working with, qualitative or quantitative? Are your variables categorical/discrete or are they continuous?
3. Decide what you want to communicate with your figure, and use that to choose a plot type. A bar chart can be used to quickly and easily compare total measurements between variables and are especially good for count data but may mask the range of the distribution. Box plots, on the other hand, show the distribution.


Temperature over each Month
Temperature
Month
60
70
80
90
5
6
7
8
9

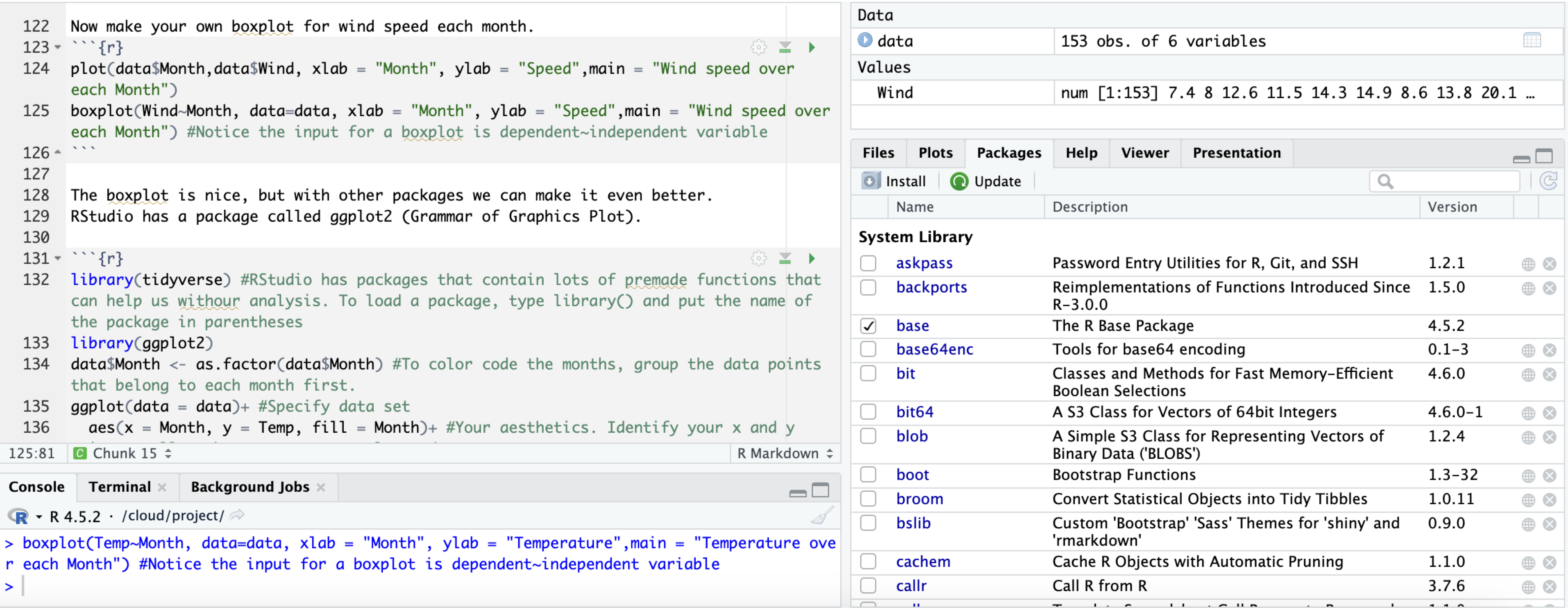
Have you made a box plot?

Yes

0%

No

0%



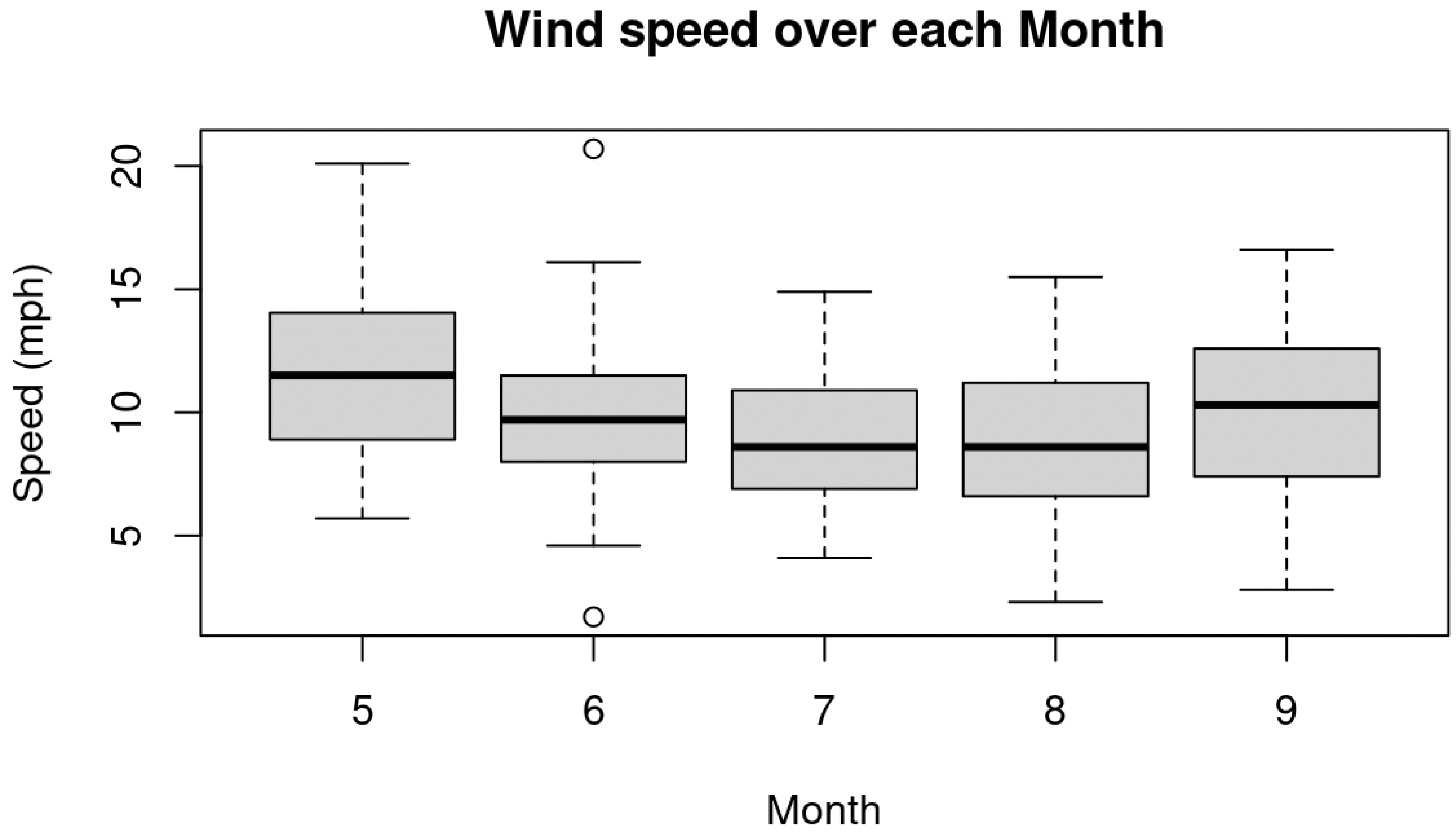
~~~
Now make your own boxplot for wind speed each month.
```{r}
plot(data$Month,data$Wind, xlab = "Month", ylab = "Speed",main = "Wind speed over
each Month")
boxplot(Wind~Month, data=data, xlab = "Month", ylab = "Speed",main = "Wind speed over
each Month") #Notice the input for a boxplot is dependent~independent variable
```

The boxplot is nice, but with other packages we can make it even better.
RStudio has a package called ggplot2 (Grammar of Graphics Plot).

```{r}
library(tidyverse) #RStudio has packages that contain lots of premade functions that
can help us withour analysis. To load a package, type library() and put the name of
the package in parentheses
library(ggplot2)
data$Month <- as.factor(data$Month) #To color code the months, group the data points
that belong to each month first.
ggplot(data = data)+ #Specify data set
  aes(x = Month, y = Temp, fill = Month)+ #Your aesthetics. Identify your x and y
~~~

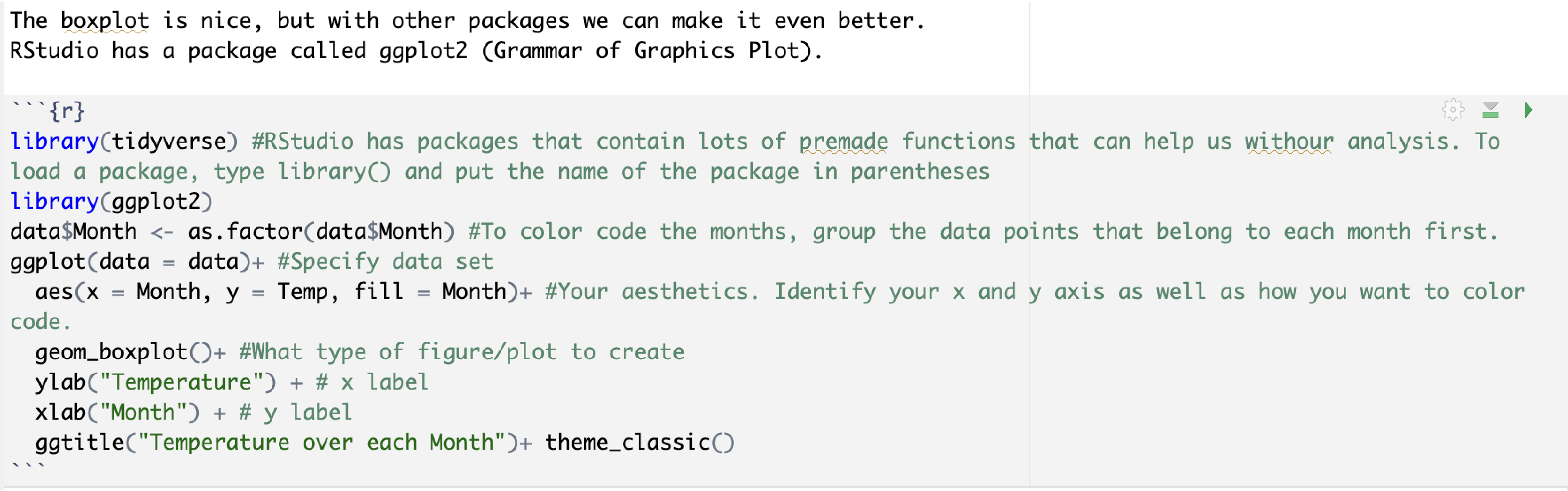
```
125:81   Chunk 15   R Markdown
```

```
R 4.5.2 · /cloud/project/
> boxplot(Temp~Month, data=data, xlab = "Month", ylab = "Temperature",main = "Temperature ove
r each Month") #Notice the input for a boxplot is dependent~independent variable
>
```

```
Data
data     153 obs. of 6 variables
Values
Wind     num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 …
```

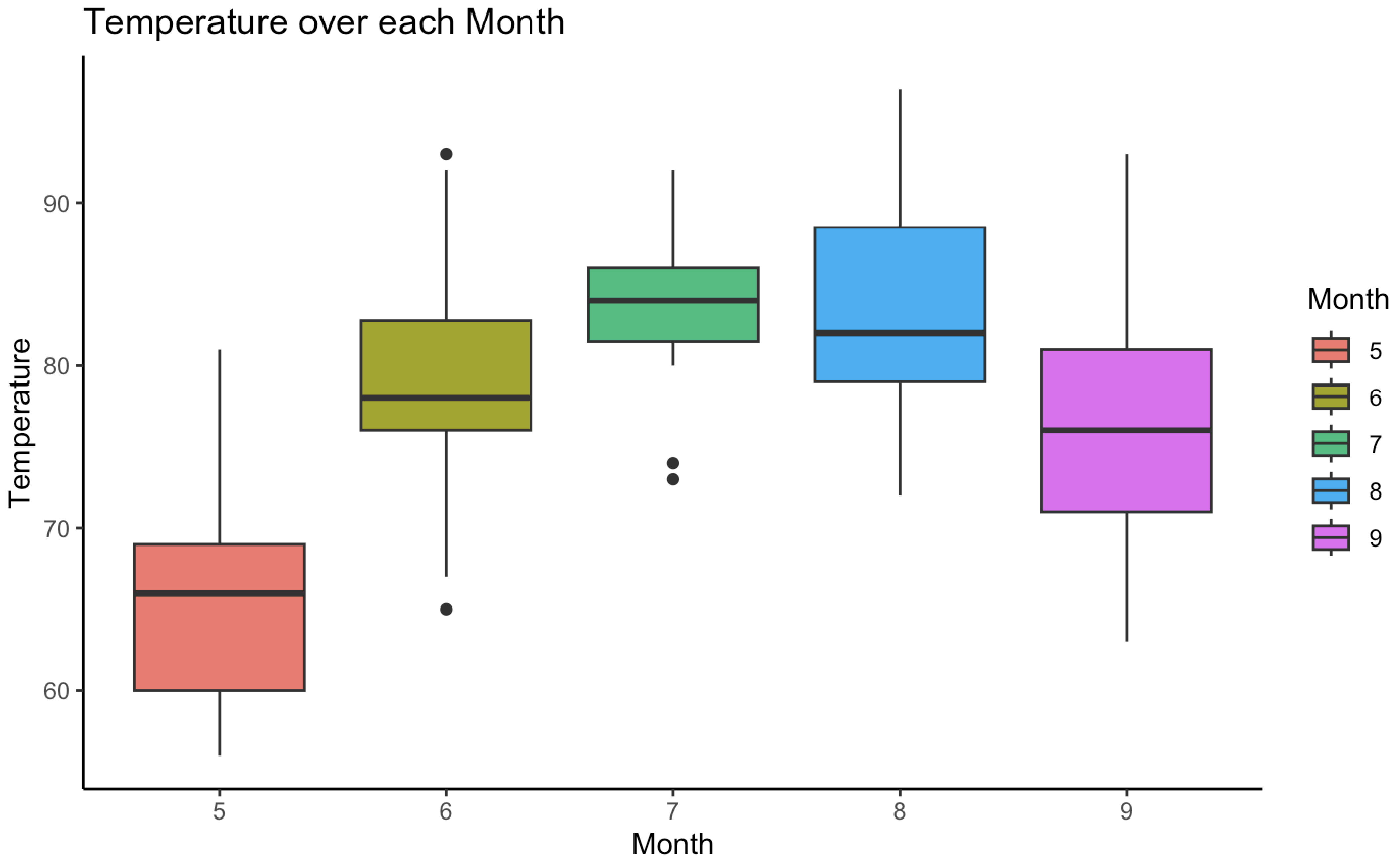
	Name	Description	Version
	System Library		
☐	askpass	Password Entry Utilities for R, Git, and SSH	1.2.1
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.5.0
✓	base	The R Base Package	4.5.2
☐	base64enc	Tools for base64 encoding	0.1-3
☐	bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.6.0
☐	bit64	A S3 Class for Vectors of 64bit Integers	4.6.0-1
☐	blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.4
☐	boot	Bootstrap Functions	1.3-32
☐	broom	Convert Statistical Objects into Tidy Tibbles	1.0.11
☐	bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.9.0
☐	cachem	Cache R Objects with Automatic Pruning	1.1.0
☐	callr	Call R from R	3.7.6

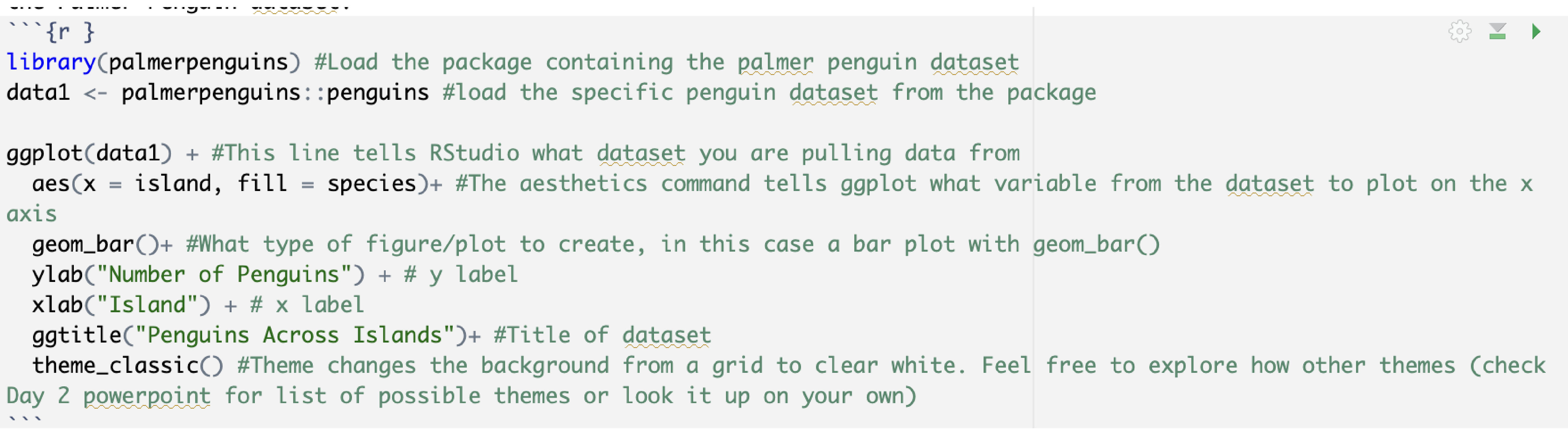

Wind speed over each Month
Speed (mph)
5
10
15
20
5
6
7
8
9
Month



The boxplot is nice, but with other packages we can make it even better.
RStudio has a package called ggplot2 (Grammar of Graphics Plot).

~~~
```{r}
library(tidyverse) #RStudio has packages that contain lots of premade functions that can help us withour analysis. To load a package, type library() and put the name of the package in parentheses
library(ggplot2)
data$Month <- as.factor(data$Month) #To color code the months, group the data points that belong to each month first.
ggplot(data = data)+ #Specify data set
  aes(x = Month, y = Temp, fill = Month)+ #Your aesthetics. Identify your x and y axis as well as how you want to color code.
  geom_boxplot()+ #What type of figure/plot to create
  ylab("Temperature") + # x label
  xlab("Month") + # y label
  ggtitle("Temperature over each Month")+ theme_classic()
```
~~~


Temperature over each Month
Temperature
90
80
70
60
5
6
7
8
9
Month
Month
5
6
7
8
9



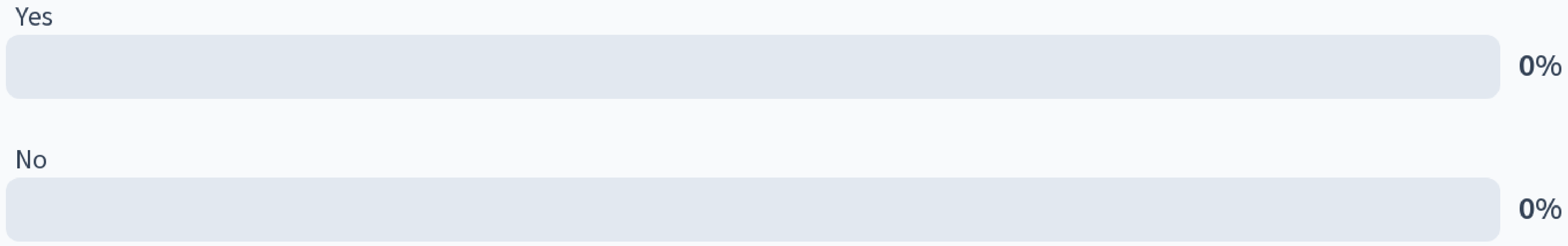
the Palmer Penguin dataset.

~~~
```{r }
library(palmerpenguins) #Load the package containing the palmer penguin dataset
data1 <- palmerpenguins::penguins #load the specific penguin dataset from the package

ggplot(data1) + #This line tells RStudio what dataset you are pulling data from
  aes(x = island, fill = species)+ #The aesthetics command tells ggplot what variable from the dataset to plot on the x
axis
  geom_bar()+ #What type of figure/plot to create, in this case a bar plot with geom_bar()
  ylab("Number of Penguins") + # y label
  xlab("Island") + # x label
  ggtitle("Penguins Across Islands")+ #Title of dataset
  theme_classic() #Theme changes the background from a grid to clear white. Feel free to explore how other themes (check
Day 2 powerpoint for list of possible themes or look it up on your own)
```
~~~

Imitating the code above, run your own code creating a bar graph for the number of penguins in each species sampled and how many of each sex there were.

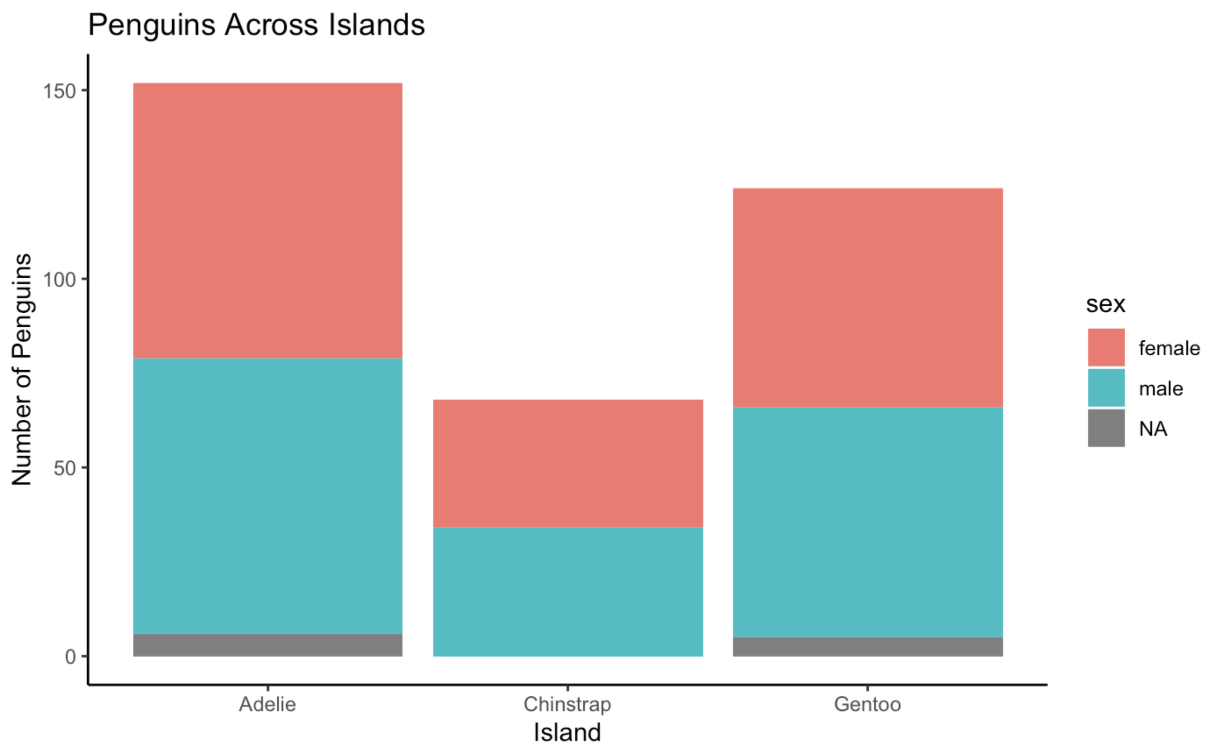
~~~
```{r}

```
~~~

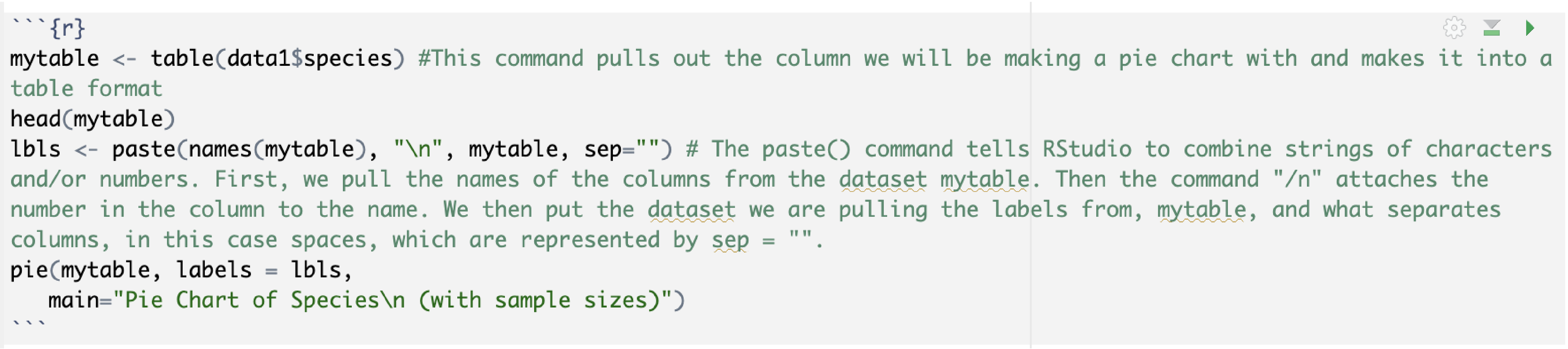
Have you made a bar graph?



Imitating the code above, run your own code creating a bar graph for the number of penguins in each species sampled and how many of each sex there were.

~~~
```{r}
ggplot(data1) + #This line tells RStudio what dataset you are pulling data from
  aes(x = species, fill = sex)+ #The aesthetics command tells ggplot what variable from the dataset to plot on the x axis
  geom_bar()+ #What type of figure/plot to create, in this case a bar plot with geom_bar()
  ylab("Number of Penguins") + # y label
  xlab("Island") + # x label
  ggtitle("Penguins Across Islands")+ #Title of dataset
  theme_classic() #Theme changes the background from a grid to clear white. Feel free to explore how other themes (check Day 2 powerpoint for list of possible themes or look it up on your own)
```
~~~



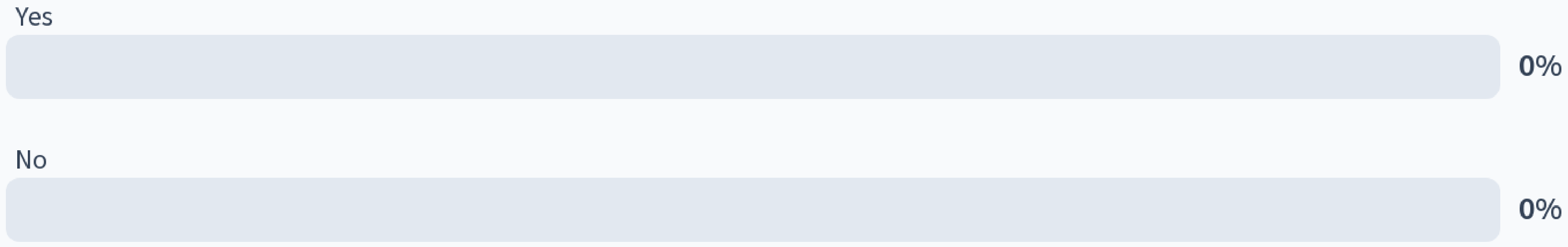


~~~
```{r}
mytable <- table(data1$species) #This command pulls out the column we will be making a pie chart with and makes it into a table format
head(mytable)
lbls <- paste(names(mytable), "\n", mytable, sep="") # The paste() command tells RStudio to combine strings of characters and/or numbers. First, we pull the names of the columns from the dataset mytable. Then the command "/n" attaches the number in the column to the name. We then put the dataset we are pulling the labels from, mytable, and what separates columns, in this case spaces, which are represented by sep = "".
pie(mytable, labels = lbls,
   main="Pie Chart of Species\n (with sample sizes)")
```
~~~

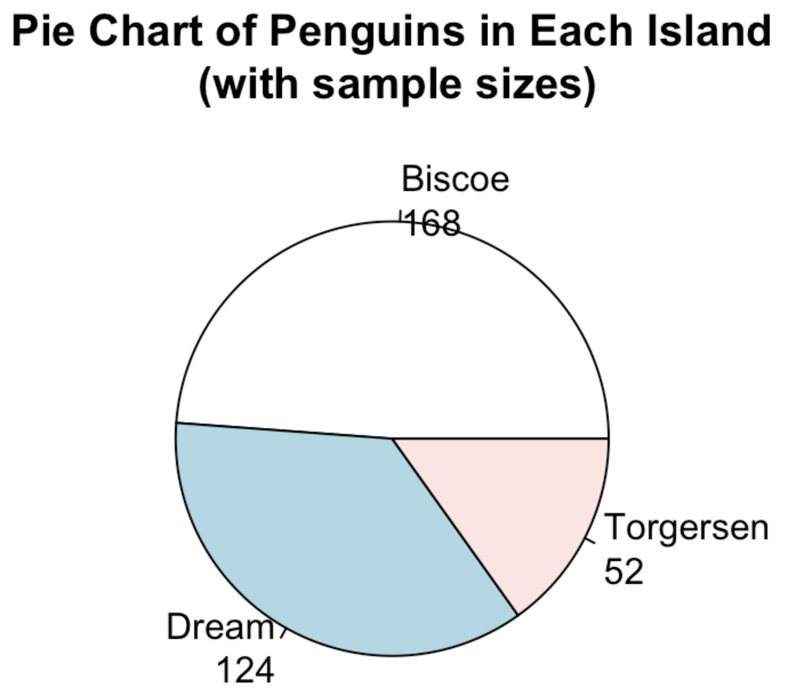
Create a pie chart showing how many penguins were sampled on each island.

~~~
```{r}

```
~~~

Have you made a pie chart?



```{r}
mytable <- table(data1$island) #This command pulls out the column we will be making a pie chart with and makes it into a table format
head(mytable)
lbls <- paste(names(mytable), "\n", mytable, sep="") # The paste() command tells RStudio to combine strings of characters and/or numbers. First, we pull the names of the columns from the dataset mytable. Then the command "/n" attaches the number in the column to the name. We then put the dataset we are pulling the labels from, mytable, and what separates columns, in this case spaces, which are represented by sep = "".
pie(mytable, labels = lbls,
   main="Pie Chart of Penguins in Each Island\n (with sample sizes)")
```

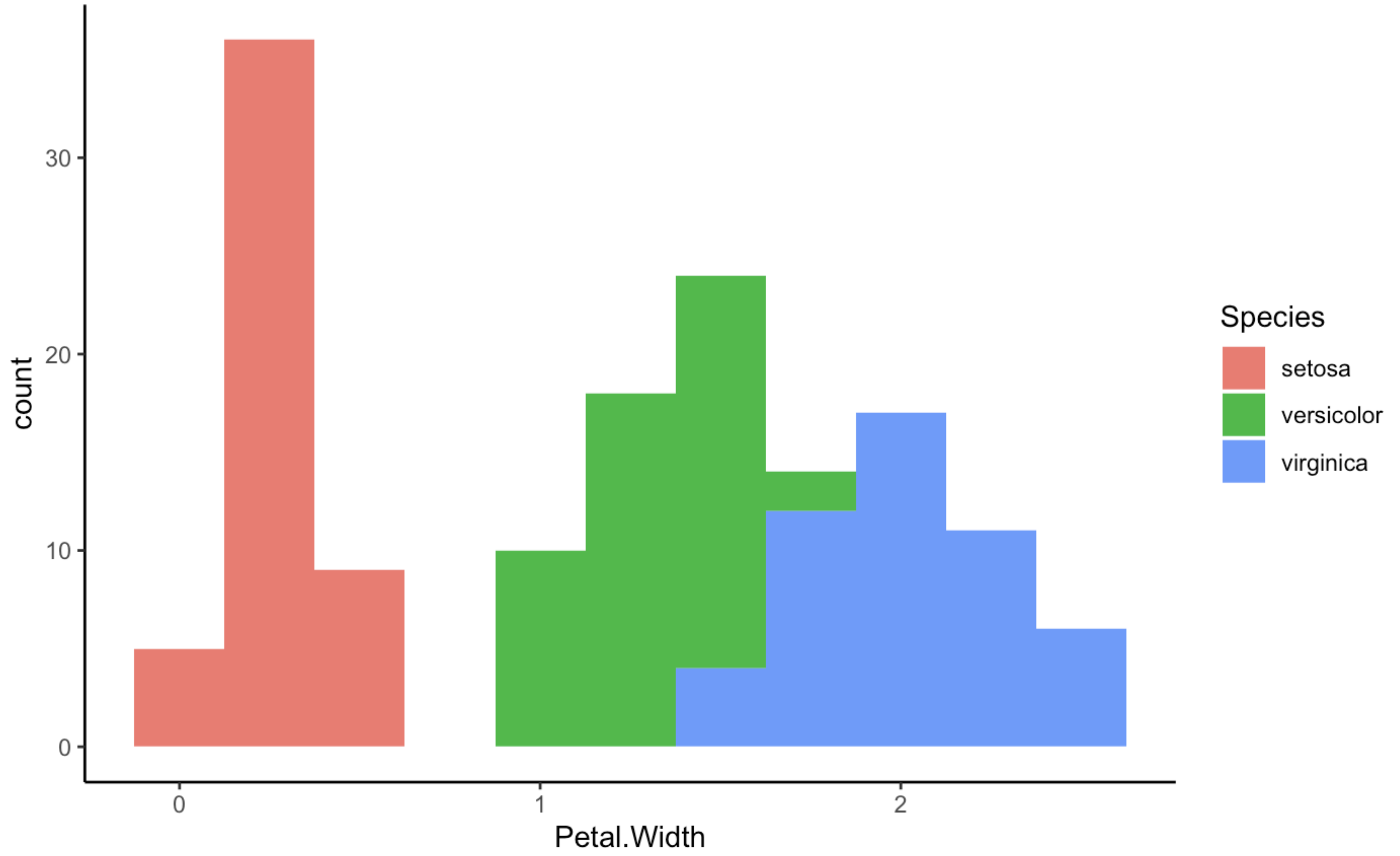


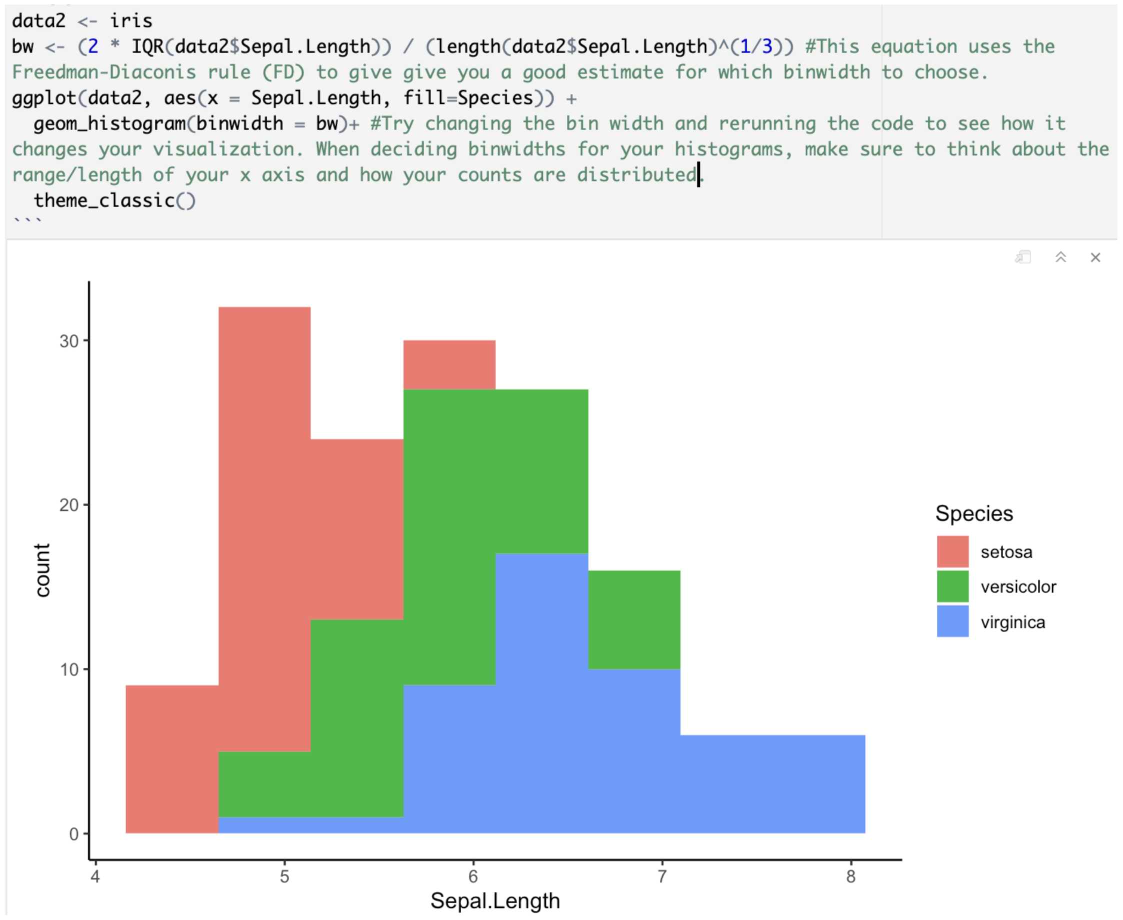
Create your own histogram looking at petal width across species.

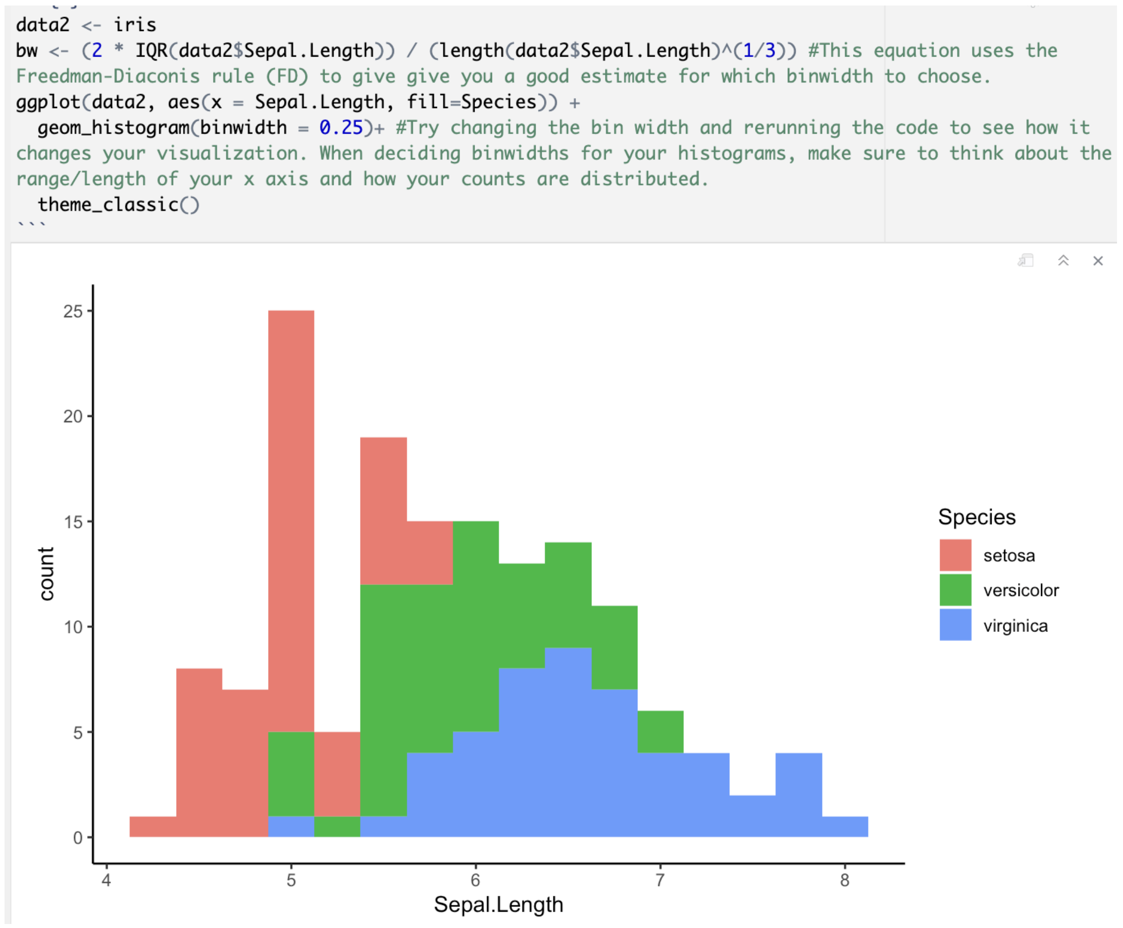
~~~
```{r}

```
~~~

Have you made a histogram?

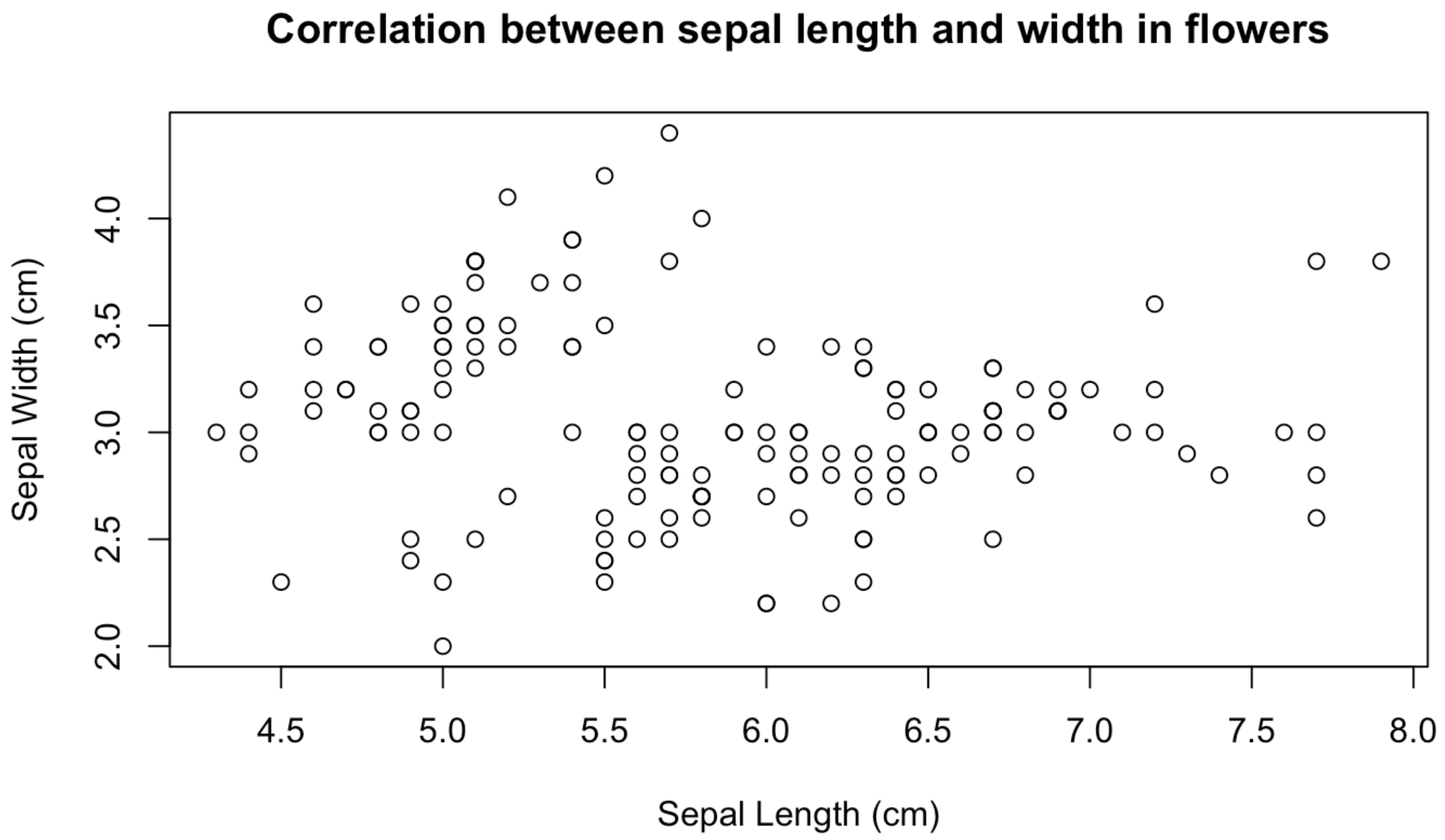
~~~
```{r}
ggplot(data2, aes(x = Petal.Width, fill=Species)) +
  geom_histogram(binwidth = 0.25)+ #Try changing the bin width and rerunning the code to see how it
changes your visualization. When deciding binwidths for your histograms, make sure to think about the
range/length of your x axis.
  theme_classic()
```
~~~



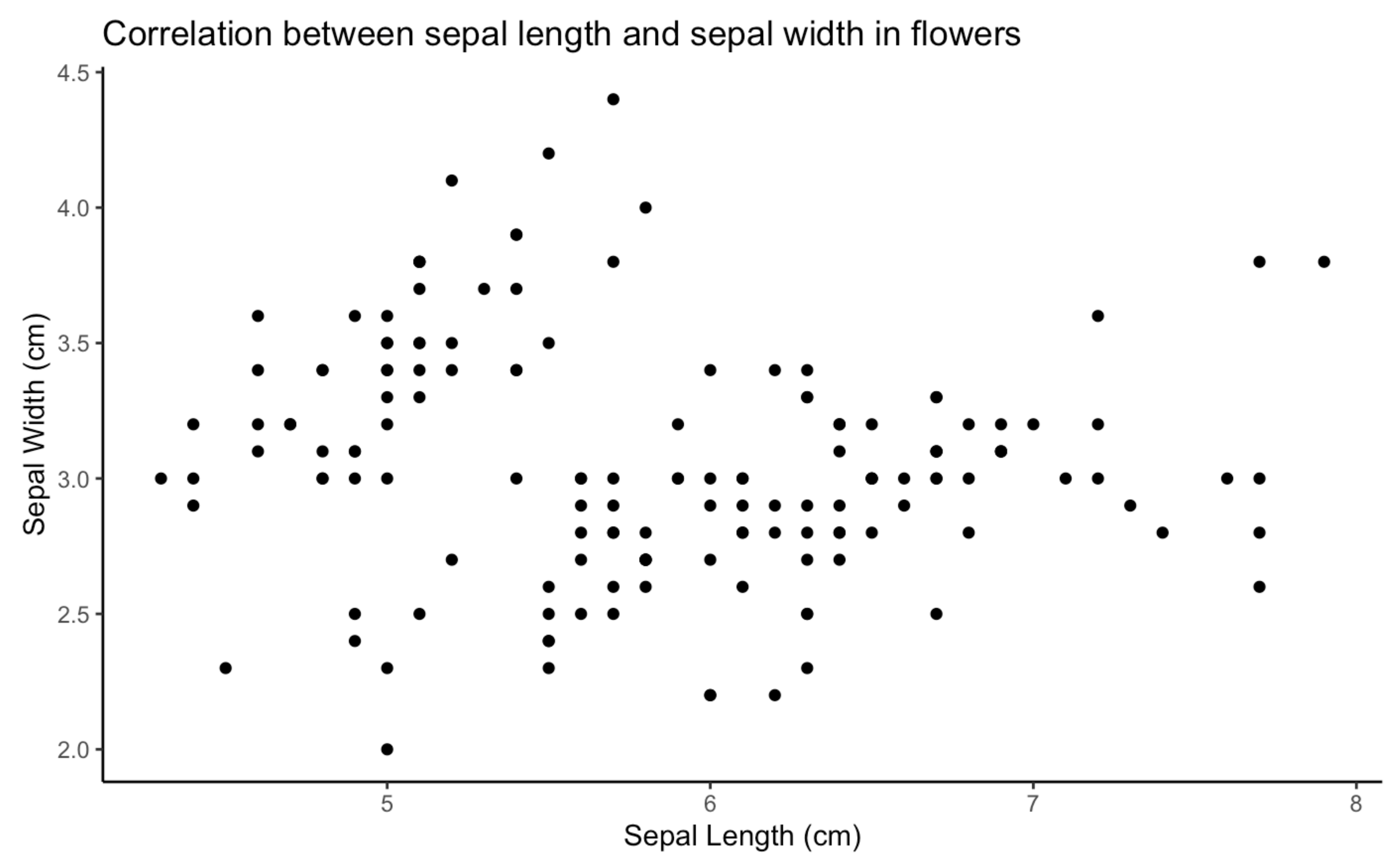


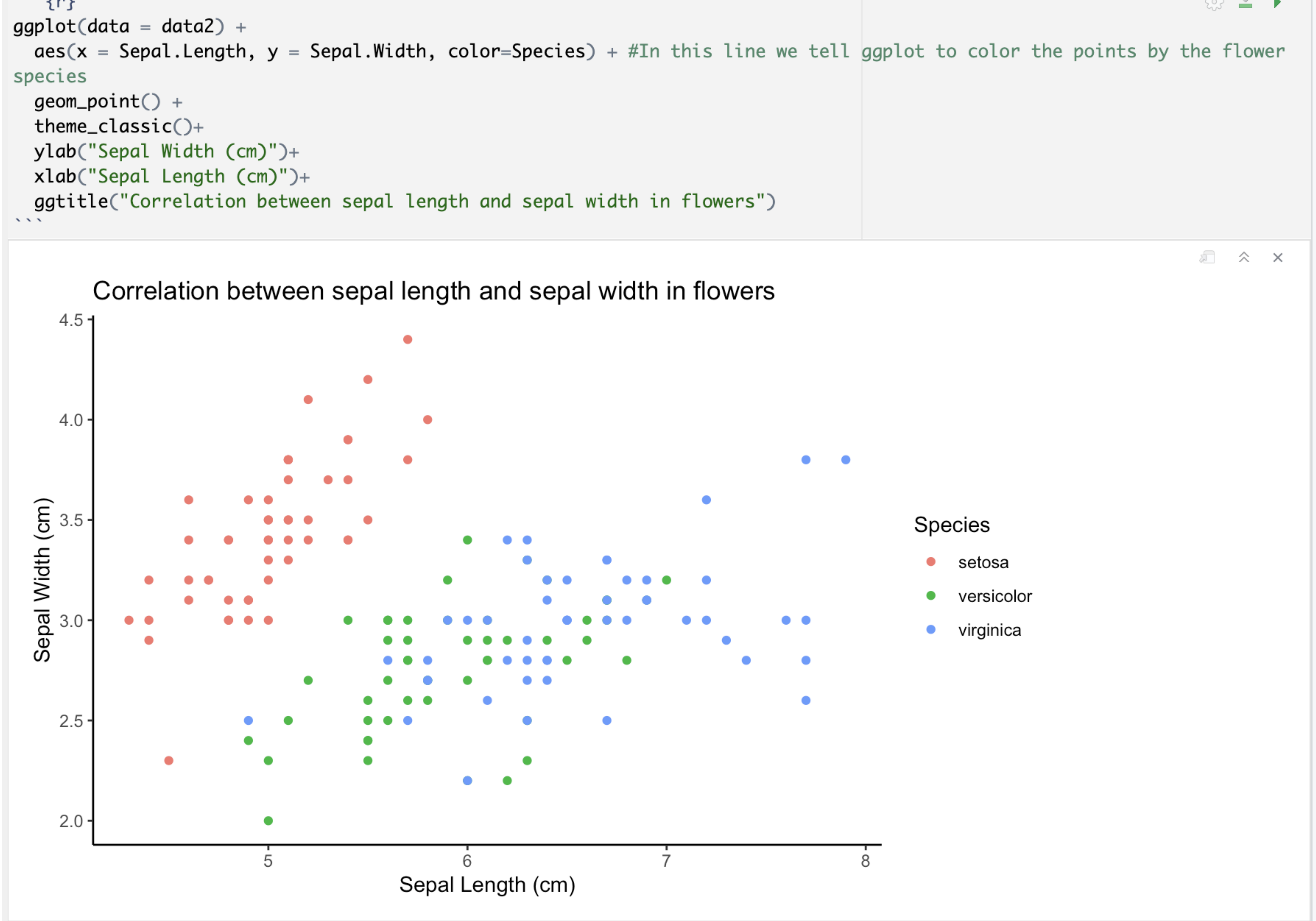
```r
data2 <- iris
bw <- (2 * IQR(data2$Sepal.Length)) / (length(data2$Sepal.Length)^(1/3)) #This equation uses the
Freedman-Diaconis rule (FD) to give give you a good estimate for which binwidth to choose.
ggplot(data2, aes(x = Sepal.Length, fill=Species)) +
  geom_histogram(binwidth = bw)+ #Try changing the bin width and rerunning the code to see how it
changes your visualization. When deciding binwidths for your histograms, make sure to think about the
range/length of your x axis and how your counts are distributed.
  theme_classic()
```
~~~

count
0
10
20
30
4
5
6
7
8
Sepal.Length
Species
setosa
versicolor
virginica



```r
data2 <- iris
bw <- (2 * IQR(data2$Sepal.Length)) / (length(data2$Sepal.Length)^(1/3)) #This equation uses the
Freedman-Diaconis rule (FD) to give give you a good estimate for which binwidth to choose.
ggplot(data2, aes(x = Sepal.Length, fill=Species)) +
  geom_histogram(binwidth = 0.25)+ #Try changing the bin width and rerunning the code to see how it
changes your visualization. When deciding binwidths for your histograms, make sure to think about the
range/length of your x axis and how your counts are distributed.
  theme_classic()
```
~~~

```{r}
plot(data2$Sepal.Length, data2$Sepal.Width,xlab = "Sepal Length (cm)", ylab = "Sepal Width (cm)",main = "Correlation between sepal length and width in flowers")
```



```{r}
ggplot(data = data2) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  theme_classic() +
  ylab("Sepal Width (cm)")+
  xlab("Sepal Length (cm)")+
  ggtitle("Correlation between sepal length and sepal width in flowers")

```





```
  {r}
ggplot(data = data2) +
  aes(x = Sepal.Length, y = Sepal.Width, color=Species) + #In this line we tell ggplot to color the points by the flower
species
  geom_point() +
  theme_classic()+
  ylab("Sepal Width (cm)")+
  xlab("Sepal Length (cm)")+
  ggtitle("Correlation between sepal length and sepal width in flowers")
```
```

Correlation between sepal length and sepal width in flowers
```

## Which flower species has the smallest petals?

| | |
|---|---|
| Setosa | 0 |
| Versicolor | 0 |
| Virginia | 0 |

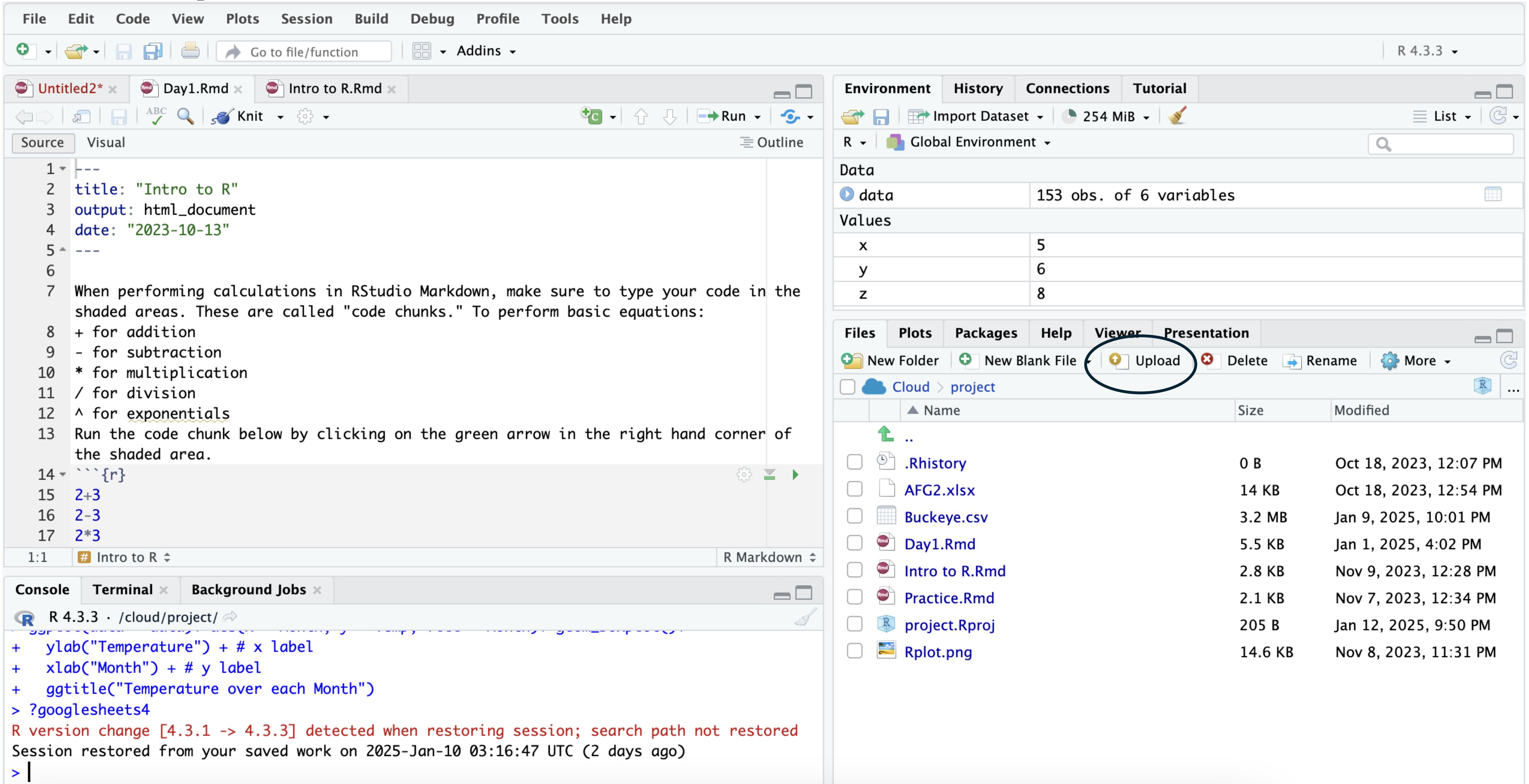
## Course Feedback After Week 1

**0 surveys completed**

0 surveys underway

## Part 2: Data Tidying

# First, download our Day3.RMD from Github and Upload to Posit Cloud



~~~
---
title: "Intro to R"
output: html_document
date: "2023-10-13"
---

When performing calculations in RStudio Markdown, make sure to type your code in the
shaded areas. These are called "code chunks." To perform basic equations:
+ for addition
- for subtraction
* for multiplication
/ for division
^ for exponentials
Run the code chunk below by clicking on the green arrow in the right hand corner of
the shaded area.
```{r}
2+3
2-3
2*3
~~~

Data

| | |
|---|---|
| data | 153 obs. of 6 variables |

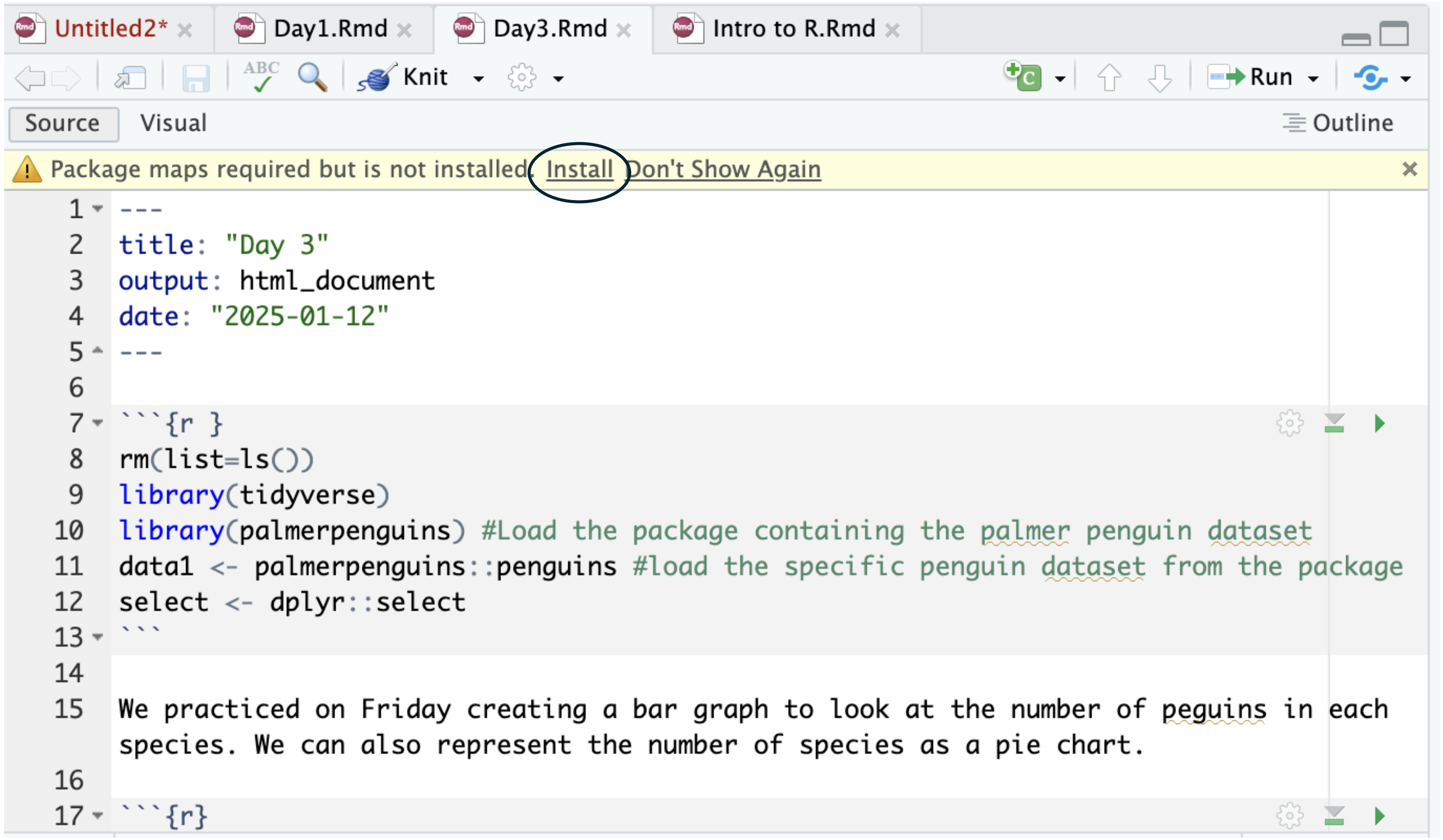
Values

| | |
|---|---|
| x | 5 |
| y | 6 |
| z | 8 |

| Name | Size | Modified |
|---|---|---|
| .. | | |
| .Rhistory | 0 B | Oct 18, 2023, 12:07 PM |
| AFG2.xlsx | 14 KB | Oct 18, 2023, 12:54 PM |
| Buckeye.csv | 3.2 MB | Jan 9, 2025, 10:01 PM |
| Day1.Rmd | 5.5 KB | Jan 1, 2025, 4:02 PM |
| Intro to R.Rmd | 2.8 KB | Nov 9, 2023, 12:28 PM |
| Practice.Rmd | 2.1 KB | Nov 7, 2023, 12:34 PM |
| project.Rproj | 205 B | Jan 12, 2025, 9:50 PM |
| Rplot.png | 14.6 KB | Nov 8, 2023, 11:31 PM |

```
R 4.3.3 · /cloud/project/
+   ylab("Temperature") + # x label
+   xlab("Month") + # y label
+   ggtitle("Temperature over each Month")
> ?googlesheets4
R version change [4.3.1 -> 4.3.3] detected when restoring session; search path not restored
Session restored from your saved work on 2025-Jan-10 03:16:47 UTC (2 days ago)
> |
```

Are all of your packages installed?

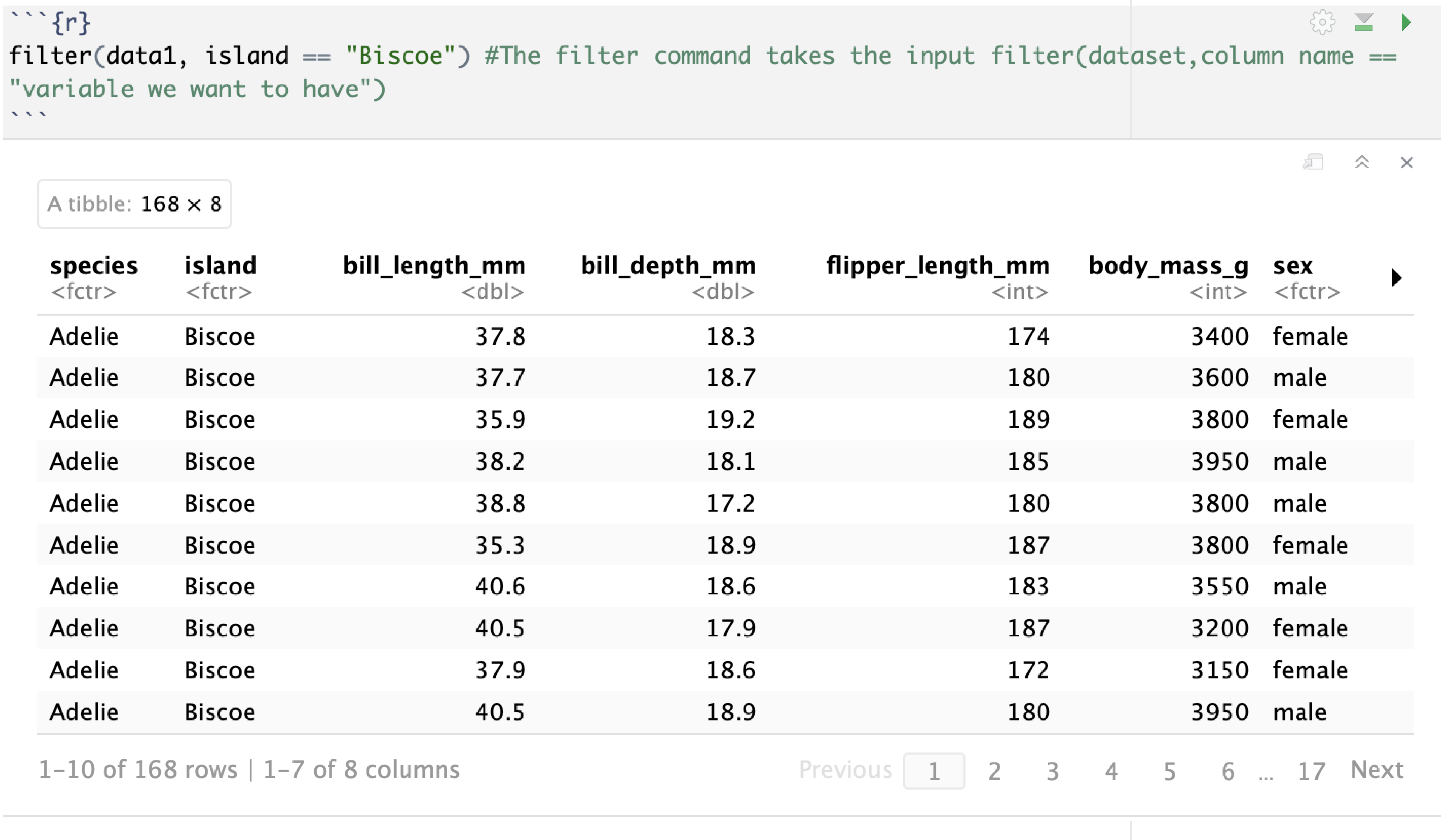


Package maps required but is not installed. Install Don't Show Again

~~~
---
title: "Day 3"
output: html_document
date: "2025-01-12"
---

```{r }
rm(list=ls())
library(tidyverse)
library(palmerpenguins) #Load the package containing the palmer penguin dataset
data1 <- palmerpenguins::penguins #load the specific penguin dataset from the package
select <- dplyr::select
~~~

We practiced on Friday creating a bar graph to look at the number of peguins in each species. We can also represent the number of species as a pie chart.

```{r}
```



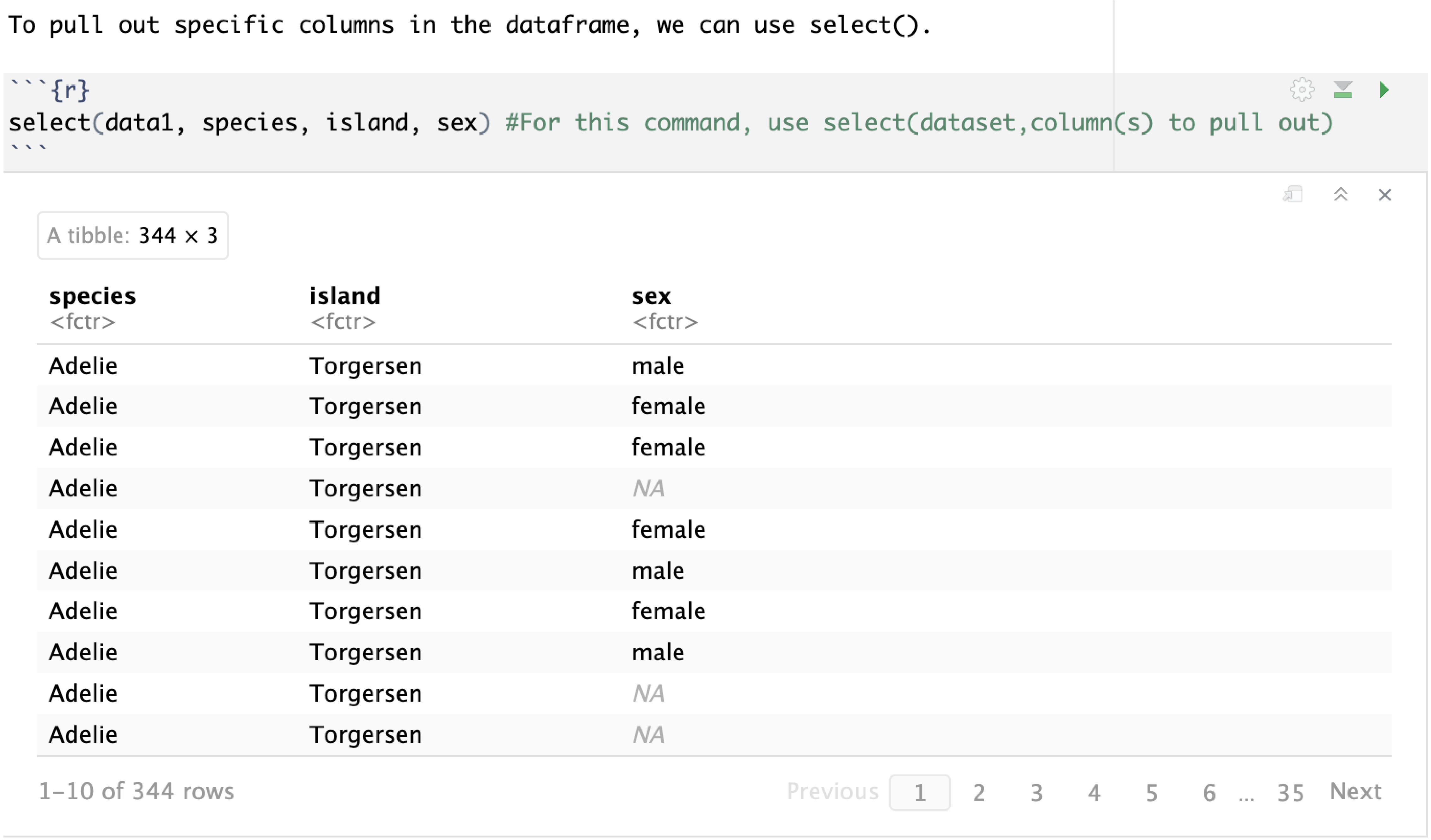
```{r}
filter(data1, island == "Biscoe") #The filter command takes the input filter(dataset,column name == "variable we want to have")
```

A tibble: 168 × 8

species <fctr>	island <fctr>	bill_length_mm <dbl>	bill_depth_mm <dbl>	flipper_length_mm <int>	body_mass_g <int>	sex <fctr>
Adelie	Biscoe	37.8	18.3	174	3400	female
Adelie	Biscoe	37.7	18.7	180	3600	male
Adelie	Biscoe	35.9	19.2	189	3800	female
Adelie	Biscoe	38.2	18.1	185	3950	male
Adelie	Biscoe	38.8	17.2	180	3800	male
Adelie	Biscoe	35.3	18.9	187	3800	female
Adelie	Biscoe	40.6	18.6	183	3550	male
Adelie	Biscoe	40.5	17.9	187	3200	female
Adelie	Biscoe	37.9	18.6	172	3150	female
Adelie	Biscoe	40.5	18.9	180	3950	male

1–10 of 168 rows | 1–7 of 8 columns

Previous 1 2 3 4 5 6 ... 17 Next



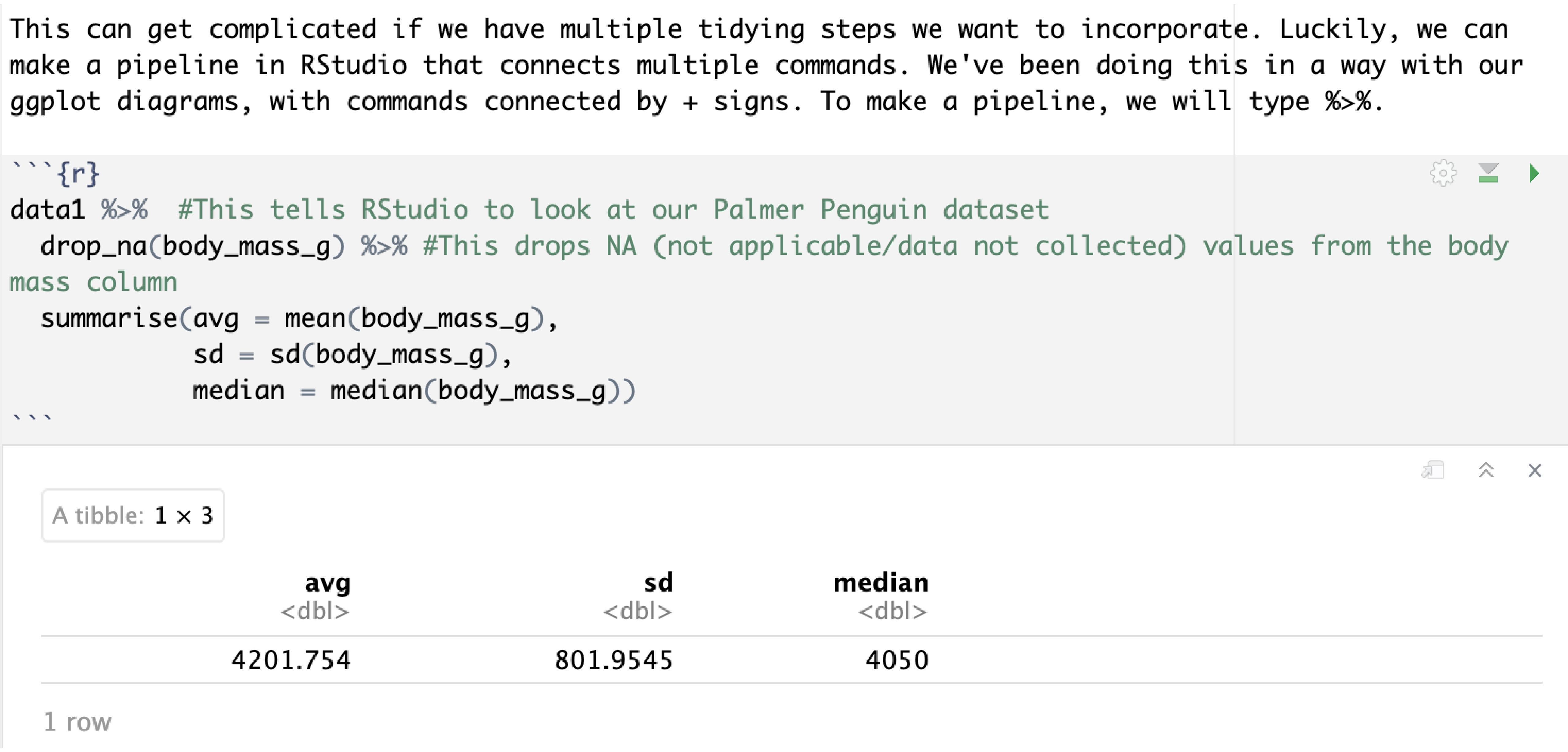
To pull out specific columns in the dataframe, we can use select().

```{r}
select(data1, species, island, sex) #For this command, use select(dataset,column(s) to pull out)
```

A tibble: 344 × 3

species <fctr>	island <fctr>	sex <fctr>
Adelie	Torgersen	male
Adelie	Torgersen	female
Adelie	Torgersen	female
Adelie	Torgersen	NA
Adelie	Torgersen	female
Adelie	Torgersen	male
Adelie	Torgersen	female
Adelie	Torgersen	male
Adelie	Torgersen	NA
Adelie	Torgersen	NA

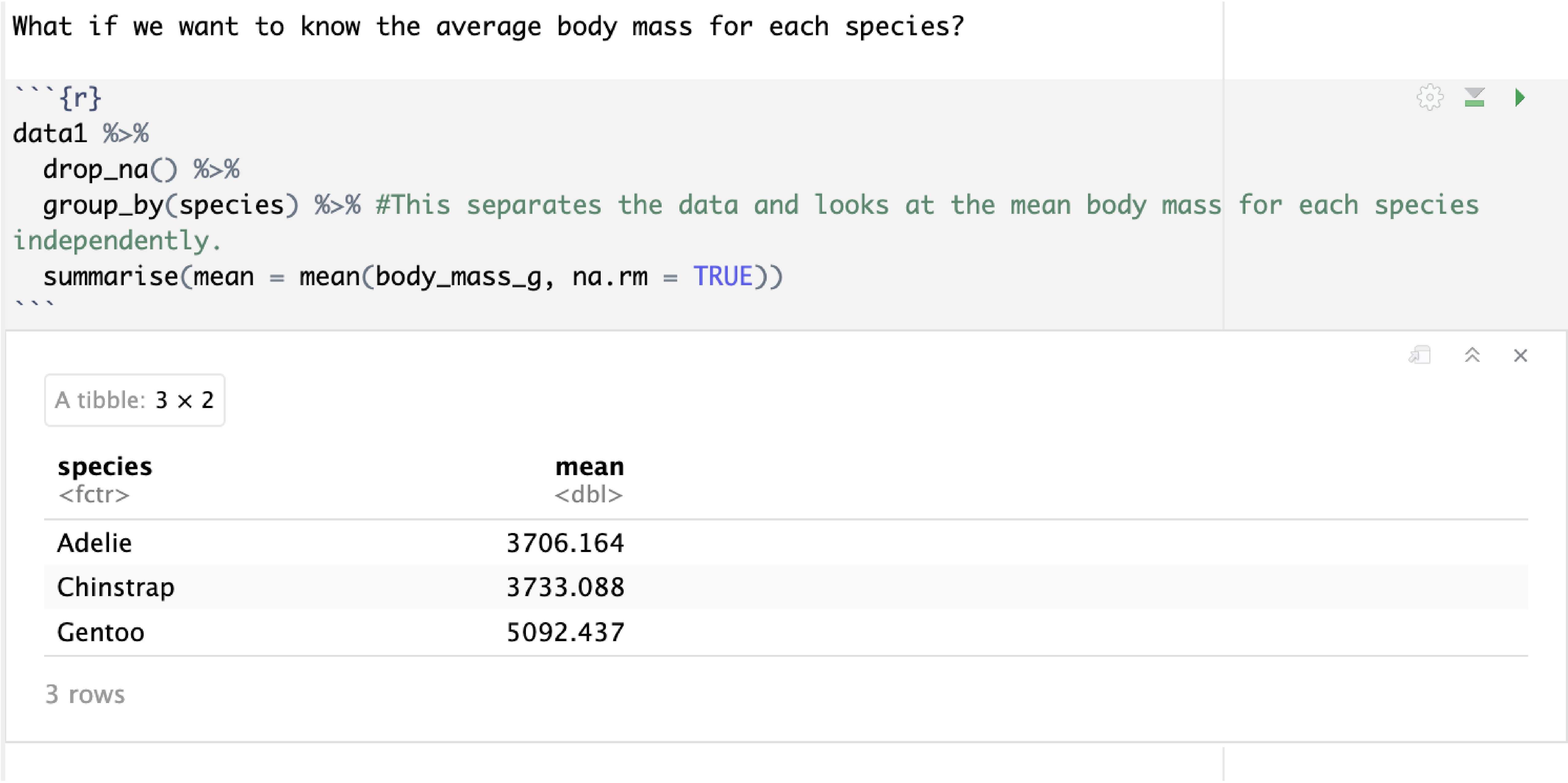
1–10 of 344 rows



This can get complicated if we have multiple tidying steps we want to incorporate. Luckily, we can make a pipeline in RStudio that connects multiple commands. We've been doing this in a way with our ggplot diagrams, with commands connected by + signs. To make a pipeline, we will type %>%.

~~~
```{r}
data1 %>%  #This tells RStudio to look at our Palmer Penguin dataset
  drop_na(body_mass_g) %>% #This drops NA (not applicable/data not collected) values from the body
mass column
  summarise(avg = mean(body_mass_g),
            sd = sd(body_mass_g),
            median = median(body_mass_g))
```
~~~

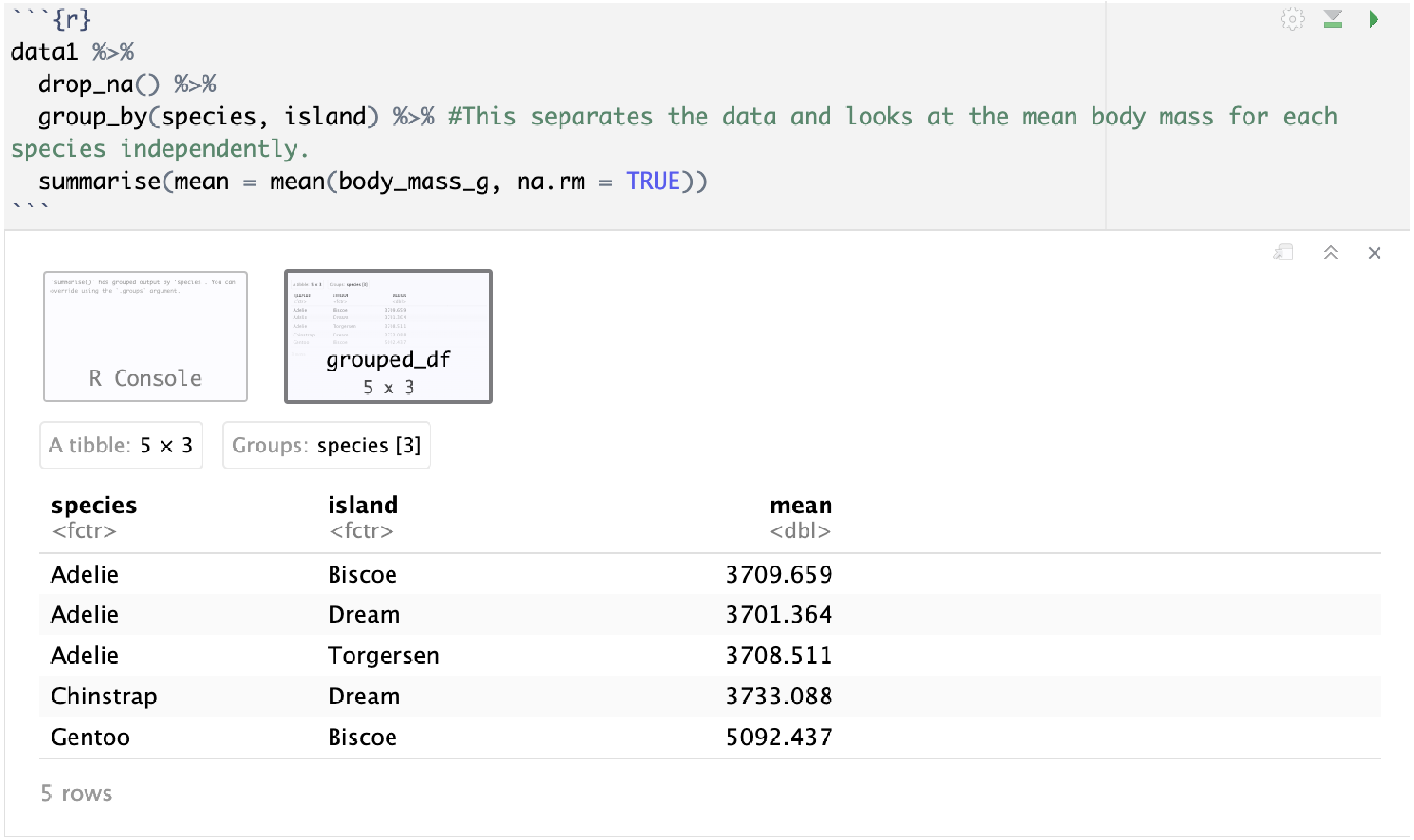
```
A tibble: 1 × 3

         avg          sd    median
       <dbl>       <dbl>     <dbl>
    4201.754    801.9545      4050

1 row
```



What if we want to know the average body mass for each species?

~~~
```{r}
data1 %>%
  drop_na() %>%
  group_by(species) %>% #This separates the data and looks at the mean body mass for each species independently.
  summarise(mean = mean(body_mass_g, na.rm = TRUE))
```
~~~

```
A tibble: 3 × 2

species        mean
<fctr>        <dbl>
Adelie     3706.164
Chinstrap  3733.088
Gentoo     5092.437

3 rows
```



~~~
```{r}
data1 %>%
  drop_na() %>%
  group_by(species, island) %>% #This separates the data and looks at the mean body mass for each
species independently.
  summarise(mean = mean(body_mass_g, na.rm = TRUE))
```
~~~

```
`summarise()` has grouped output by 'species'. You can override using the `.groups` argument.
```

A tibble: 5 × 3 Groups: species [3]

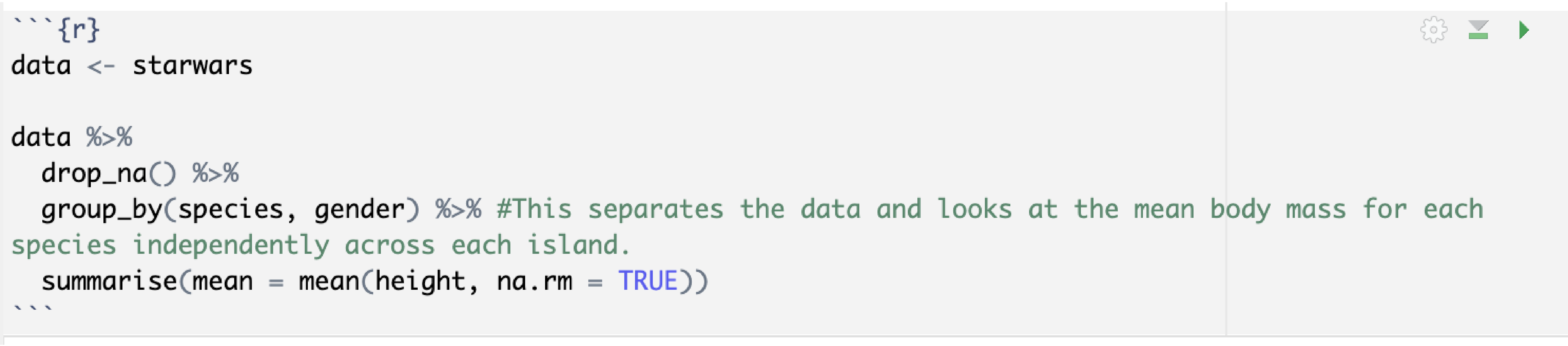
species <fctr>	island <fctr>	mean <dbl>
Adelie	Biscoe	3709.659
Adelie	Dream	3701.364
Adelie	Torgersen	3708.511
Chinstrap	Dream	3733.088
Gentoo	Biscoe	5092.437

5 rows

Share your starwars code!

Nobody has responded yet.

Hang tight! Responses are coming in.



~~~
```{r}
data <- starwars

data %>%
  drop_na() %>%
  group_by(species, gender) %>% #This separates the data and looks at the mean body mass for each
species independently across each island.
  summarise(mean = mean(height, na.rm = TRUE))
```
~~~