Introduction to ggplot2

Background

While R provides ready-to-go tools for basic data visualization, it can quickly become cumbersome trying to customize these to more specific needs. The R package *ggplot2* provides an alternative "grammar of graphics" that makes constructing high-quality, nuanced graphics a lot easier. At the same time, *ggplot2* accomplishes this by adopting its own syntax and structure, so it takes a little getting used to. It's well worth it though, as I hope this module will show.

Objectives

We really have one objective today: make a pretty plot! But we can break that down a bit:

1. Install and load ggplot2
2. Discuss basics of ggplot syntax
3. Build a plot
4. Save the graphic to a file

Install and Load ggplot2

To install an R package, use the `install.packages()` function.

```
install.packages("ggplot2")
```

Installing the package gets it onto our computer, but not loaded into our current R session. To do that, we need `library()`.

```
library(ggplot2)
```

Basics of ggplot Syntax

What’s in a ggplot graphic?

The flexibility and elegance of ggplot comes in part from its “layered grammar of graphics” (http://vita.had.co.nz/papers/layered-grammar.pdf). In essence, ggplot allows us to build complex graphics from multiple “layers” of information. The resulting graphics objects have the following components:

1. data: the data you want to visualize
2. mapping: how data are assigned to different visual “aesthetics” – x-axis, y-axis, shape of points, etc…
3. geom: a geometric object used to represent the data – points, bars, lines, etc…
4. scaling: how values of variables are translated to distances/colors/shapes/sizes
5. faceting: representing subsets of data using subplots

How do I build one?

The structure of a ggplot graphic also gives us a nice workflow for building one of our own:

1. specify the data using the `ggplot()` function.
2. add aesthetic mapping using the `aes()` function. - you can set aesthetics within the `ggplot()` function to have them apply to all subsequent layers, or you can set them within each individual `geom_` function you use to create the objects representing the data.
3. Define the geometric objects you’d like to represent your data using the various `geom_` functions. These are added *after* we define the data and any global aesthetic mapping in the `gplot()` function. We simply add them to the end of our ggplot workflow with a `+`. Some example geoms are:

- `geom_point()` : represents the data with points (requires at least x, y coordinates)
- `geom_boxplot()` : represents the data with boxplots (requires at least one variable x)
- `geom_violin()` : represents the data with violin plots (requires at least one variable x)
- `geom_histogram()` : represents the frequency of different values of variables (requires at least one variable x)
- `geom_line()` : represents the data with a line – helpful for time series. Careful– not the same as fitting a line to the data.
- `geom_smooth()` : represents the data with a smoothed curve and optional confidence intervals. There are options to change what statistical method is used to generate the curve, including linear models, loess regressions, and generalized additive models.

Each type of geom can represent different aesthetics. Points, for example, have a position, color, shape, and size that we can use to represent data. Boxes in a boxplot, on the other hand, have heights, widths, and colors. It’s often helpful to check the help file for whatever geom you’d like to use to see what aesthetics are available to map variables to.

4. Add any additional plot elements (e.g., other data subsets, other response variables, trend lines) by appending more `geom_` or `stat_` functions to the end with `+`. There are a lot of different options available, and you can even write your own functions to use.
5. Define scaling of variables if needed (e.g., adjust color range used to represent a variable)
6. Make adjustments to “fine tune” your graphic – axes, legends, etc.

A Step-by-Step Example with Scatterplots

We'll demonstrate this workflow by building a scatterplot, step-by-step. Scatterplots show the relationship between two continuous variables.

Load some data

First, we'll need to load some data. As someone who really enjoys playing around with real data, I thought we could work with some published data on Antarctic penguins. These data were collected by Kristen Gorman with the Palmer Station Long Term Ecological Research Program and later developed into an R package for educational uses by Allison Horst, and Alison Hill, and Kristen Gorman. The published manuscript focused on differences in sexual size dimorphism among 3 species of penguin and their relation to sex differences in foraging ecology. The full data set has really neat info on reproductive success and blood measurements of stable isotopes, but for the sake of brevity we will focus on the morphological measures used to quantify sexual size dimorphism.

Let's install the *palmerpenguins* package.

```
install.packages("palmerpenguins") # install the package
```

And now let's load it.

```
library(palmerpenguins) # load it in current R session
```

The data we'll be using are now available in an object called `penguins`. The full data are in `penguins_raw`, which I encourage you to explore on your own later.

Let's take a look at our data set using the `print()` function.

```
print(penguins)
```

```
## # A tibble: 344 × 8
##    species island    bill_length_mm bill_depth_mm flipper_…¹ body_…² sex    year
##    <fct>   <fct>              <dbl>         <dbl>      <int>   <int> <fct> <int>
##  1 Adelie  Torgersen           39.1          18.7        181    3750 male   2007
##  2 Adelie  Torgersen           39.5          17.4        186    3800 fema…  2007
##  3 Adelie  Torgersen           40.3          18          195    3250 fema…  2007
##  4 Adelie  Torgersen          NA            NA            NA      NA <NA>   2007
##  5 Adelie  Torgersen           36.7          19.3        193    3450 fema…  2007
##  6 Adelie  Torgersen           39.3          20.6        190    3650 male   2007
##  7 Adelie  Torgersen           38.9          17.8        181    3625 fema…  2007
##  8 Adelie  Torgersen           39.2          19.6        195    4675 male   2007
##  9 Adelie  Torgersen           34.1          18.1        193    3475 <NA>   2007
## 10 Adelie  Torgersen           42            20.2        190    4250 <NA>   2007
## # … with 334 more rows, and abbreviated variable names ¹flipper_length_mm,
## #   ²body_mass_g
```

We see that we have variables (columns) for species, island, 4 different morphological variables, sex, and the year of observation. R also gives us some information about this object – it says that it is a `tibble`, tells us the size of the tibble (344 rows by 8 columns), and under each variable we get a nice summary of the type of data stored there (factors ("fctr"), machine precision numbers ("dbl"), integers ("int), etc.). The `tibble` is a particular type of `data.frame` that is used as part of the tidyverse approach to data analysis and visualization. *ggplot2* is part of the tidyverse, but we can also use it as a standalone package without getting into all the other features of the tidyverse. This is what we'll do here. If you're interested in learning more about tidyverse, check out this module (TidyverseTest.md).

Let's start by turning the `tibble` into a regular `data.frame`. This isn't strictly necessary because *ggplot2* would largely treat the data the same way, but we'll do this anyway to formally sever our ties to the rest of the tidyverse.

```
penguins_df = as.data.frame(penguins) # Create a new object called penguins_df. Thi
s object is the penguins tibble, but as a data frame.
```

Let's check that this did what we intended before we move on. One of the nastier elements of base R's `data.frame` is that if we use our `print()` function from above to look at it, it will flood the console with data. To get a nice little preview like the tibble gave us, use `head()`.

```
head(penguins_df) # use the head() function to look at the first several rows of th
e penguins_df data frame. Careful that you are now using penguins_df!
```

```
##   species    island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
## 1  Adelie Torgersen           39.1          18.7               181        3750
## 2  Adelie Torgersen           39.5          17.4               186        3800
## 3  Adelie Torgersen           40.3          18.0               195        3250
## 4  Adelie Torgersen             NA            NA                NA          NA
## 5  Adelie Torgersen           36.7          19.3               193        3450
## 6  Adelie Torgersen           39.3          20.6               190        3650
##      sex year
## 1   male 2007
## 2 female 2007
## 3 female 2007
## 4   <NA> 2007
## 5 female 2007
## 6   male 2007
```

Great, that looks like the data made it through unscathed.

Deciding what to plot

Now we need to decide what we want our plot to look like. Based on an exploration of these data I did in the module on tidyverse (TidyverseTest.md), I would be interested to know how a penguin's mass scales with its body size, and whether this differs between males and females. I could represent this with a scatterplot, where body size (e.g., flipper length) is on the x-axis and body mass is on the y-axis, and where the points are colored or shaped differently depending on whether the penguin is male or female. It would also be great to add a trend line to plots for males and females to see if the relationship seems different for the sexes, but we'll get to that later.

Try making a sketch of this plot on paper or a whiteboard *before* you start trying to build it in R. I never cease to surprise myself with how often I *think* I know what I want to plot, only to find out later that I cannot, in fact, convey my message with that type of plot. While exploring is a necessary part of the process, having a clear mental image of your goal at the outset is invaluable for guiding that journey.

Defining the data and aesthetics

Now that we know what data we want to plot, and which variables we'd like where, we can tell R about this using the `ggplot()` function.

But, if you're like me, you've already forgotten what those variables were named in the data set. Use `names()` to refresh your memory first.

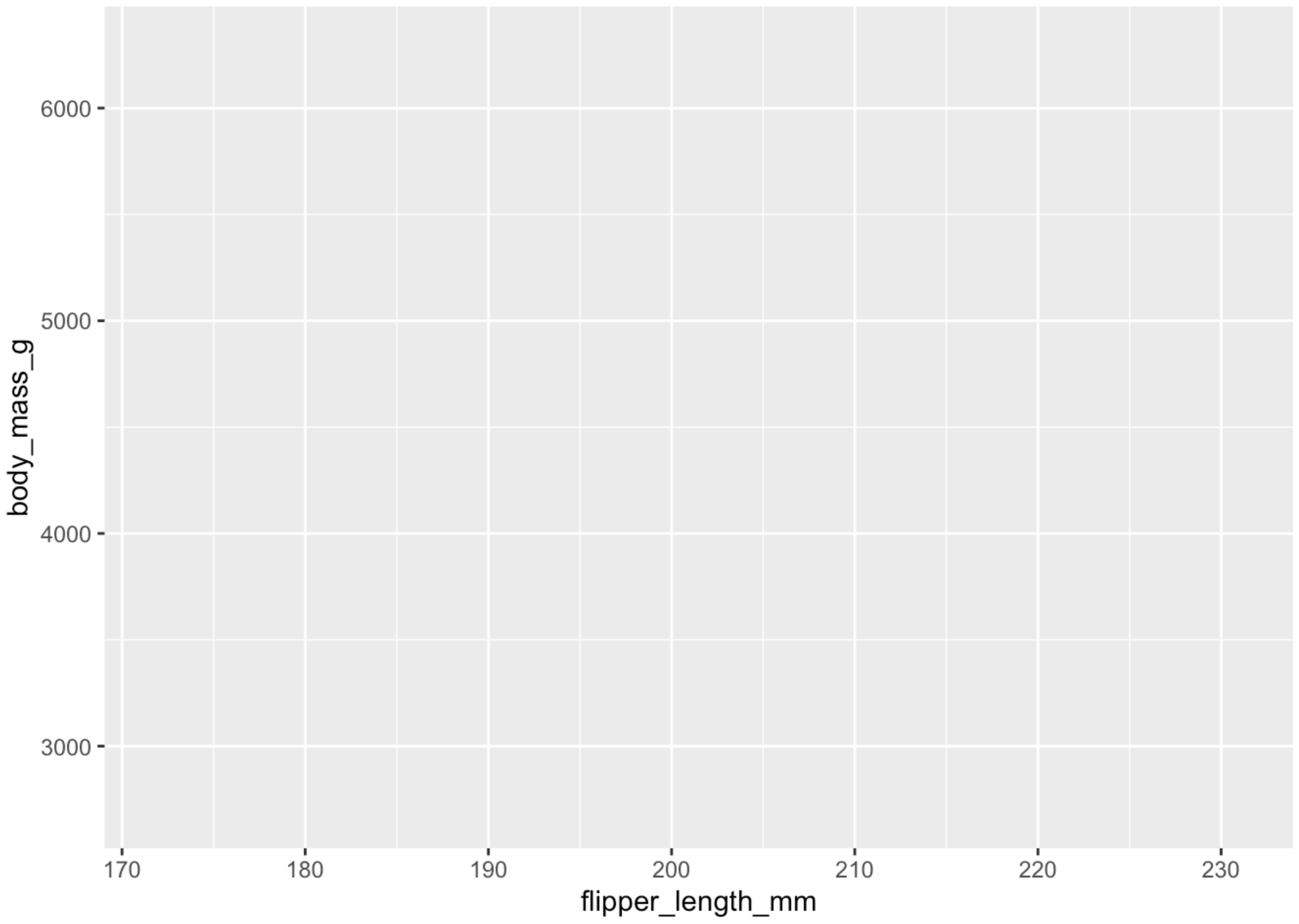
```
names(penguins_df)
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

Okay, now I know I want `flipper_length_mm` on the x-axis and `body_mass_g` on the y-axis. Eventually, we'll want to color/shape those points by `"sex"`, but we'll come back to that in a moment.

Let's inform `ggplot()` of our intention:

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g)) # data = penguins_
df, aes(x = ..., y = ...) defines which variables are mapped to which aesthetics
```



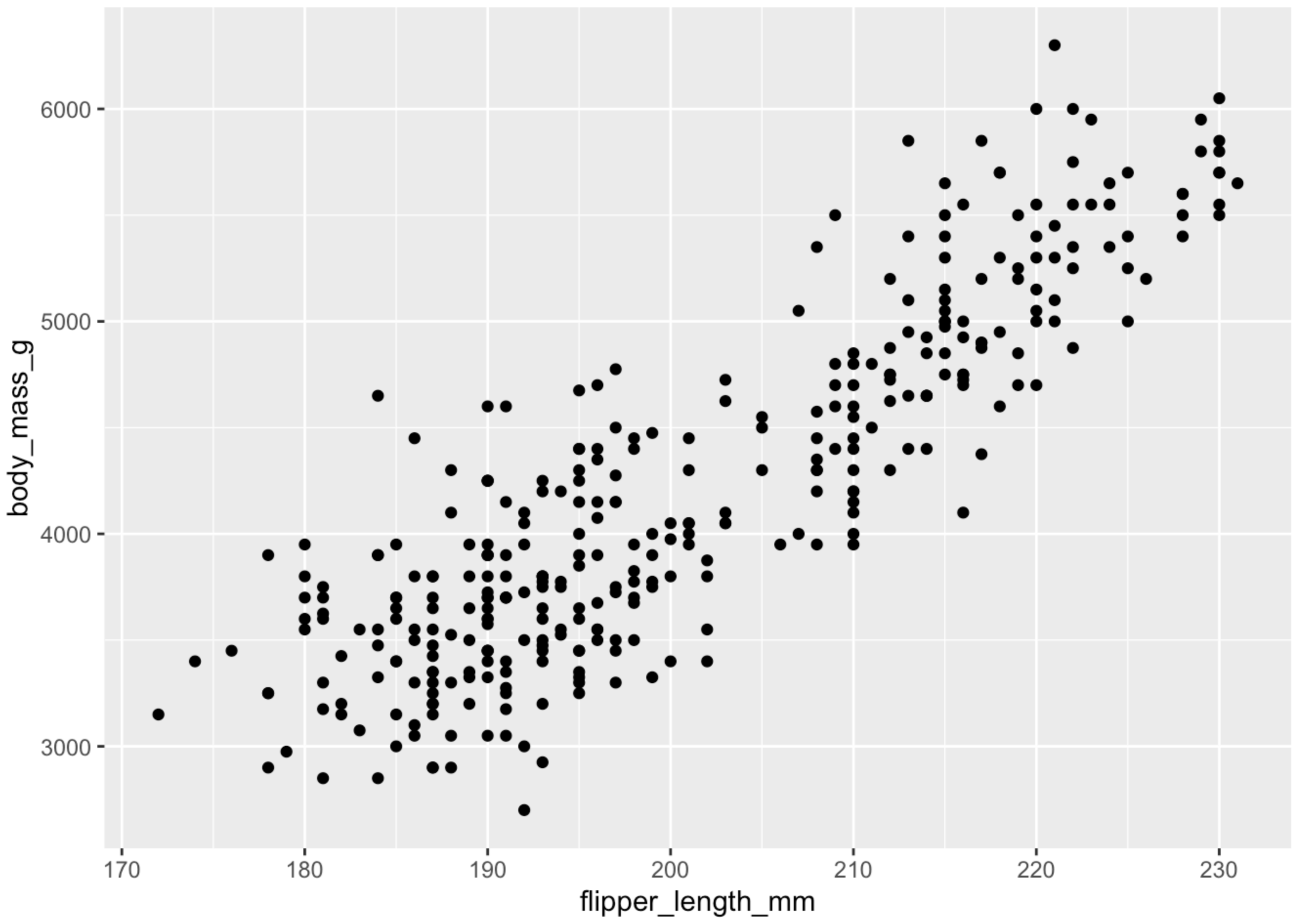
Okay, so we got our x and y axes, and they're appropriately named, but there are no data represented. What gives? Well, we haven't yet added a `geom_` function to tell ggplot how to represent those data with a geometric object.

Defining the geometric object(s)

We'd like these geometrics objects to be points for our scatterplot, so let's use `geom_point()`. Remember that we add this *after* the `ggplot()` function, using a `+`. *Make sure that the `+` comes *before* you start a new line with the return key. This tell R that you're not done with ggplot yet, so it will expect more ggplot functions on the next line.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g)) + # notice the plu
s sign here *before* the return to start a new line
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

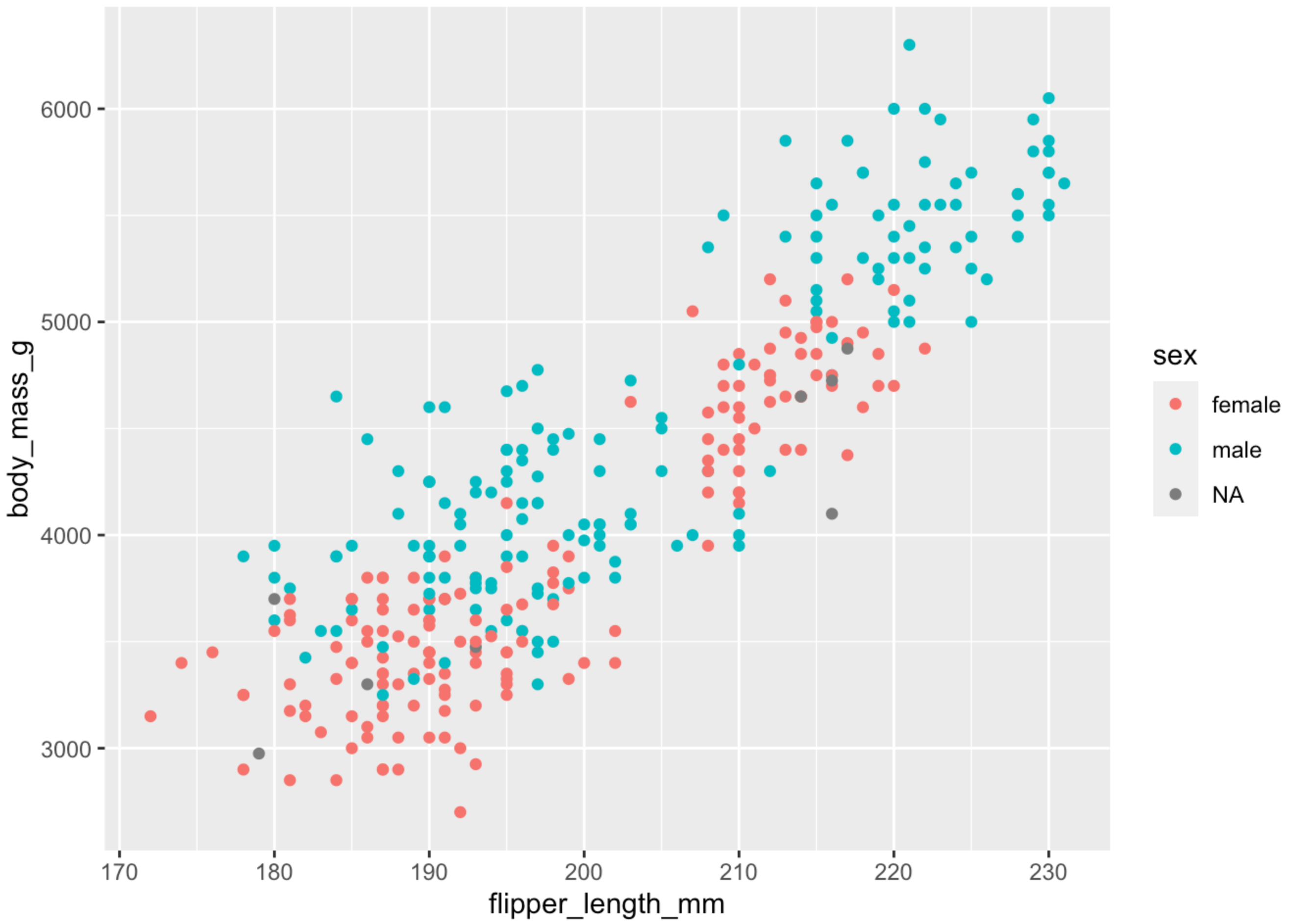


Hey, now those are some data! Note that we didn't have to specify the aesthetics again inside the `geom_point()` function because we set them up for all the layers in the plot using the `ggplot()` function. We could have also specified the aesthetics inside `geom_point()`, for example if we wanted to plot different variables on the same plot.

Now that we have some points, it's clear that we could show the data by sex if we could just map the points' shape or color to the "sex" variable in our data set. Let's return to our aesthetic mapping to take advantage of this opportunity by using the "color =" argument to the aesthetics.

```
 # add ", color = sex" to the aesthetic mapping in the ggplot() function
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, color = sex)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



Well isn't that nice. Okay, so we have specified our data, the mapping of variables onto aesthetic elements of our geometric objects (now with sex as a color!), and the types of geometric objects (points) representing the data. I don't think I want to add any other geometric objects for now (lines, bars, more points, etc.), so we're ready to make adjustments to the scaling if we'd like.

Define scaling of variables

The colors above are okay, but you want some control over this. The way variables are mapped to aesthetics like size, shape, and color can be controlled through additional `scale_` functions like `scale_color_manual()` and `scale_shape_manual()`. These get pretty deep pretty quickly – there are, for example, several different `scale_color_` functions that have different behaviors, and it takes a while to get a good grasp on them.

Let’s try using a different color scale. First we will build our own scale using colors we choose based on some identifier (e.g., a name, hex code, rgb value, or number). Then, we’ll briefly check out how to use existing color palettes to pick a set of colors at once.

We can change the color of the points in our scatterplot above using `scale_color_manual()` , provided we tell the function what values of color we’d like to have. We can use (and mix!) syntax for the colors themselves. See the available colors using `colors()` . Pick a color by name (in quotes), number (e.g., colors()[1]), RGB value, or hex code (also in quotes).

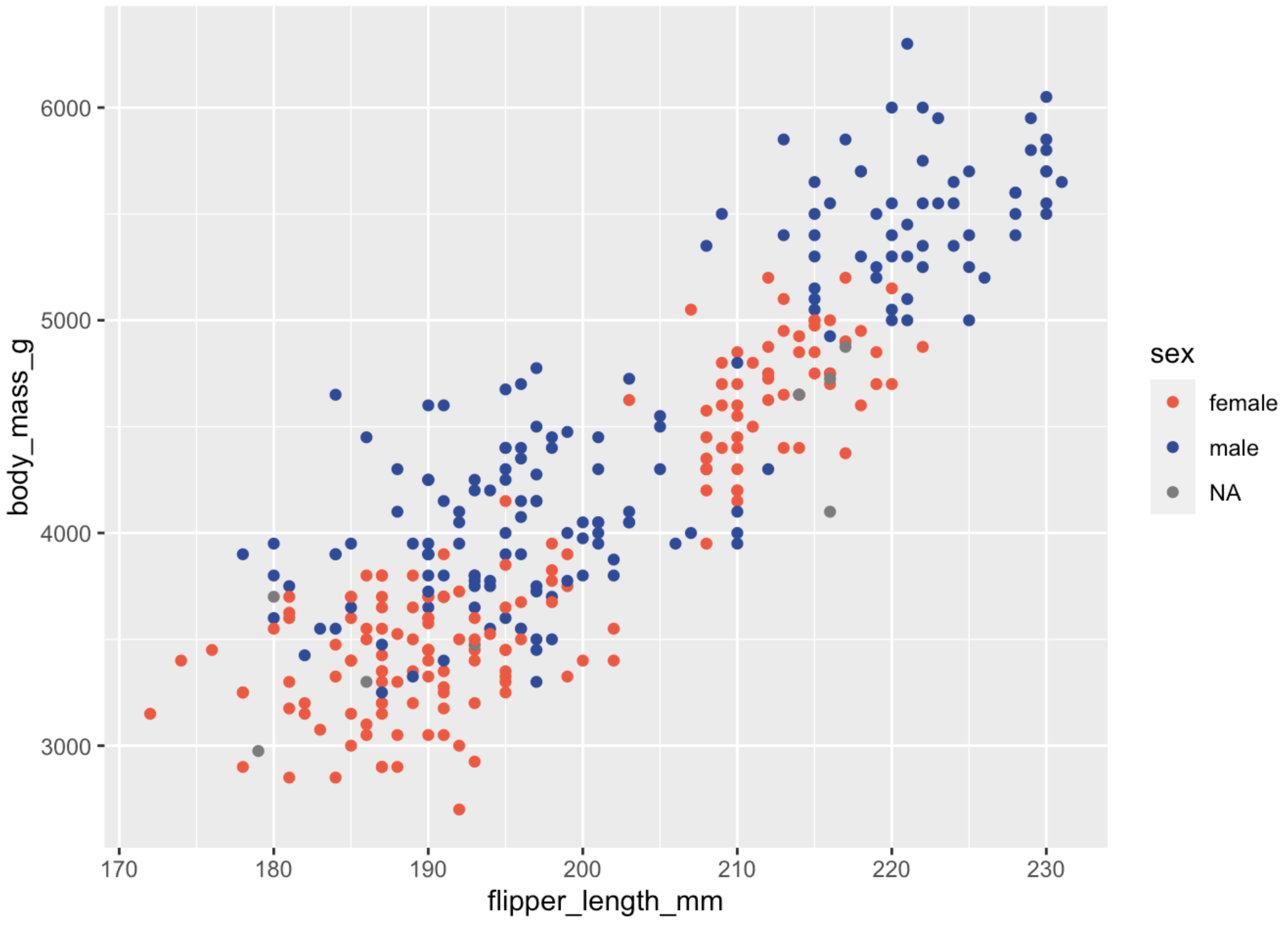
```
colors()
```

```
##   [1] "white"                "aliceblue"            "antiquewhite"        
##   [4] "antiquewhite1"        "antiquewhite2"        "antiquewhite3"       
##   [7] "antiquewhite4"        "aquamarine"           "aquamarine1"         
##  [10] "aquamarine2"          "aquamarine3"          "aquamarine4"         
##  [13] "azure"                "azure1"               "azure2"              
##  [16] "azure3"               "azure4"               "beige"               
##  [19] "bisque"               "bisque1"              "bisque2"             
##  [22] "bisque3"              "bisque4"              "black"               
##  [25] "blanchedalmond"       "blue"                 "blue1"               
##  [28] "blue2"                "blue3"                "blue4"               
##  [31] "blueviolet"           "brown"                "brown1"              
##  [34] "brown2"               "brown3"               "brown4"              
##  [37] "burlywood"            "burlywood1"           "burlywood2"          
##  [40] "burlywood3"           "burlywood4"           "cadetblue"           
##  [43] "cadetblue1"           "cadetblue2"           "cadetblue3"          
##  [46] "cadetblue4"           "chartreuse"           "chartreuse1"         
##  [49] "chartreuse2"          "chartreuse3"          "chartreuse4"         
##  [52] "chocolate"            "chocolate1"           "chocolate2"          
##  [55] "chocolate3"           "chocolate4"           "coral"               
##  [58] "coral1"               "coral2"               "coral3"              
##  [61] "coral4"               "cornflowerblue"       "cornsilk"            
##  [64] "cornsilk1"            "cornsilk2"            "cornsilk3"           
##  [67] "cornsilk4"            "cyan"                 "cyan1"               
##  [70] "cyan2"                "cyan3"                "cyan4"               
##  [73] "darkblue"             "darkcyan"             "darkgoldenrod"       
##  [76] "darkgoldenrod1"       "darkgoldenrod2"       "darkgoldenrod3"      
##  [79] "darkgoldenrod4"       "darkgray"             "darkgreen"           
##  [82] "darkgrey"             "darkkhaki"            "darkmagenta"         
##  [85] "darkolivegreen"       "darkolivegreen1"      "darkolivegreen2"     
##  [88] "darkolivegreen3"      "darkolivegreen4"      "darkorange"          
##  [91] "darkorange1"          "darkorange2"          "darkorange3"         
##  [94] "darkorange4"          "darkorchid"           "darkorchid1"         
##  [97] "darkorchid2"          "darkorchid3"          "darkorchid4"         
## [100] "darkred"              "darksalmon"           "darkseagreen"        
## [103] "darkseagreen1"        "darkseagreen2"        "darkseagreen3"       
## [106] "darkseagreen4"        "darkslateblue"        "darkslategray"       
## [109] "darkslategray1"       "darkslategray2"       "darkslategray3"      
## [112] "darkslategray4"       "darkslategrey"        "darkturquoise"       
## [115] "darkviolet"           "deeppink"             "deeppink1"           
## [118] "deeppink2"            "deeppink3"            "deeppink4"           
## [121] "deepskyblue"          "deepskyblue1"         "deepskyblue2"        
## [124] "deepskyblue3"         "deepskyblue4"         "dimgray"             
## [127] "dimgrey"              "dodgerblue"           "dodgerblue1"         
## [130] "dodgerblue2"          "dodgerblue3"          "dodgerblue4"         
## [133] "firebrick"            "firebrick1"           "firebrick2"          
## [136] "firebrick3"           "firebrick4"           "floralwhite"         
## [139] "forestgreen"          "gainsboro"            "ghostwhite"          
## [142] "gold"                 "gold1"                "gold2"               
## [145] "gold3"                "gold4"                "goldenrod"           
## [148] "goldenrod1"           "goldenrod2"           "goldenrod3"          
## [151] "goldenrod4"           "gray"                 "gray0"               
## [154] "gray1"                "gray2"                "gray3"               
## [157] "gray4"                "gray5"                "gray6"
## [160] "gray7"                "gray8"                "gray9"
## [163] "gray10"               "gray11"               "gray12"
## [166] "gray13"               "gray14"               "gray15"
## [169] "gray16"               "gray17"               "gray18"
## [172] "gray19"               "gray20"               "gray21"
## [175] "gray22"               "gray23"               "gray24"
## [178] "gray25"               "gray26"               "gray27"
## [181] "gray28"               "gray29"               "gray30"
## [184] "gray31"               "gray32"               "gray33"
## [187] "gray34"               "gray35"               "gray36"
## [190] "gray37"               "gray38"               "gray39"
## [193] "gray40"               "gray41"               "gray42"
## [196] "gray43"               "gray44"               "gray45"
## [199] "gray46"               "gray47"               "gray48"
## [202] "gray49"               "gray50"               "gray51"
## [205] "gray52"               "gray53"               "gray54"
## [208] "gray55"               "gray56"               "gray57"
## [211] "gray58"               "gray59"               "gray60"
## [214] "gray61"               "gray62"               "gray63"
## [217] "gray64"               "gray65"               "gray66"
## [220] "gray67"               "gray68"               "gray69"
## [223] "gray70"               "gray71"               "gray72"
## [226] "gray73"               "gray74"               "gray75"
## [229] "gray76"               "gray77"               "gray78"
## [232] "gray79"               "gray80"               "gray81"
## [235] "gray82"               "gray83"               "gray84"
## [238] "gray85"               "gray86"               "gray87"
## [241] "gray88"               "gray89"               "gray90"
## [244] "gray91"               "gray92"               "gray93"
## [247] "gray94"               "gray95"               "gray96"
## [250] "gray97"               "gray98"               "gray99"
## [253] "gray100"              "green"                "green1"
## [256] "green2"               "green3"               "green4"
## [259] "greenyellow"          "grey"                 "grey0"
## [262] "grey1"                "grey2"                "grey3"
## [265] "grey4"                "grey5"                "grey6"
## [268] "grey7"                "grey8"                "grey9"
## [271] "grey10"               "grey11"               "grey12"
## [274] "grey13"               "grey14"               "grey15"
## [277] "grey16"               "grey17"               "grey18"
## [280] "grey19"               "grey20"               "grey21"
## [283] "grey22"               "grey23"               "grey24"
## [286] "grey25"               "grey26"               "grey27"
## [289] "grey28"               "grey29"               "grey30"
## [292] "grey31"               "grey32"               "grey33"
## [295] "grey34"               "grey35"               "grey36"
## [298] "grey37"               "grey38"               "grey39"
## [301] "grey40"               "grey41"               "grey42"
## [304] "grey43"               "grey44"               "grey45"
## [307] "grey46"               "grey47"               "grey48"
## [310] "grey49"               "grey50"               "grey51"
## [313] "grey52"               "grey53"               "grey54"
## [316] "grey55"               "grey56"               "grey57"
## [319] "grey58"               "grey59"               "grey60"
## [322] "grey61"               "grey62"               "grey63"
## [325] "grey64"               "grey65"               "grey66"
## [328] "grey67"               "grey68"               "grey69"
## [331] "grey70"               "grey71"               "grey72"
## [334] "grey73"               "grey74"               "grey75"
## [337] "grey76"               "grey77"               "grey78"
## [340] "grey79"               "grey80"               "grey81"
## [343] "grey82"               "grey83"               "grey84"
## [346] "grey85"               "grey86"               "grey87"
## [349] "grey88"               "grey89"               "grey90"
## [352] "grey91"               "grey92"               "grey93"
## [355] "grey94"               "grey95"               "grey96"
## [358] "grey97"               "grey98"               "grey99"
## [361] "grey100"              "honeydew"             "honeydew1"
## [364] "honeydew2"            "honeydew3"            "honeydew4"
## [367] "hotpink"              "hotpink1"             "hotpink2"
## [370] "hotpink3"             "hotpink4"             "indianred"
## [373] "indianred1"           "indianred2"           "indianred3"
## [376] "indianred4"           "ivory"                "ivory1"
## [379] "ivory2"               "ivory3"               "ivory4"
## [382] "khaki"                "khaki1"               "khaki2"
## [385] "khaki3"               "khaki4"               "lavender"
## [388] "lavenderblush"        "lavenderblush1"       "lavenderblush2"
## [391] "lavenderblush3"       "lavenderblush4"       "lawngreen"
## [394] "lemonchiffon"         "lemonchiffon1"        "lemonchiffon2"
## [397] "lemonchiffon3"        "lemonchiffon4"        "lightblue"
## [400] "lightblue1"           "lightblue2"           "lightblue3"
## [403] "lightblue4"           "lightcoral"           "lightcyan"
## [406] "lightcyan1"           "lightcyan2"           "lightcyan3"
## [409] "lightcyan4"           "lightgoldenrod"       "lightgoldenrod1"
## [412] "lightgoldenrod2"      "lightgoldenrod3"      "lightgoldenrod4"
## [415] "lightgoldenrodyellow" "lightgray"            "lightgreen"
## [418] "lightgrey"            "lightpink"            "lightpink1"
## [421] "lightpink2"           "lightpink3"           "lightpink4"
## [424] "lightsalmon"          "lightsalmon1"         "lightsalmon2"
## [427] "lightsalmon3"         "lightsalmon4"         "lightseagreen"
## [430] "lightskyblue"         "lightskyblue1"        "lightskyblue2"
## [433] "lightskyblue3"        "lightskyblue4"        "lightslateblue"
## [436] "lightslategray"       "lightslategrey"       "lightsteelblue"
## [439] "lightsteelblue1"      "lightsteelblue2"      "lightsteelblue3"
## [442] "lightsteelblue4"      "lightyellow"          "lightyellow1"
## [445] "lightyellow2"         "lightyellow3"         "lightyellow4"
## [448] "limegreen"            "linen"                "magenta"
## [451] "magenta1"             "magenta2"             "magenta3"
## [454] "magenta4"             "maroon"               "maroon1"
## [457] "maroon2"              "maroon3"              "maroon4"
## [460] "mediumaquamarine"     "mediumblue"           "mediumorchid"
## [463] "mediumorchid1"        "mediumorchid2"        "mediumorchid3"
## [466] "mediumorchid4"        "mediumpurple"         "mediumpurple1"
## [469] "mediumpurple2"        "mediumpurple3"        "mediumpurple4"
## [472] "mediumseagreen"       "mediumslateblue"      "mediumspringgreen"
## [475] "mediumturquoise"      "mediumvioletred"      "midnightblue"
## [478] "mintcream"            "mistyrose"            "mistyrose1"
## [481] "mistyrose2"           "mistyrose3"           "mistyrose4"
## [484] "moccasin"             "navajowhite"          "navajowhite1"
## [487] "navajowhite2"         "navajowhite3"         "navajowhite4"
## [490] "navy"                 "navyblue"             "oldlace"
## [493] "olivedrab"            "olivedrab1"           "olivedrab2"
## [496] "olivedrab3"           "olivedrab4"           "orange"
## [499] "orange1"              "orange2"              "orange3"
## [502] "orange4"              "orangered"            "orangered1"
## [505] "orangered2"           "orangered3"           "orangered4"
## [508] "orchid"               "orchid1"              "orchid2"
## [511] "orchid3"              "orchid4"              "palegoldenrod"
## [514] "palegreen"            "palegreen1"           "palegreen2"
## [517] "palegreen3"           "palegreen4"           "paleturquoise"
## [520] "paleturquoise1"       "paleturquoise2"       "paleturquoise3"
## [523] "paleturquoise4"       "palevioletred"        "palevioletred1"
## [526] "palevioletred2"       "palevioletred3"       "palevioletred4"
## [529] "papayawhip"           "peachpuff"            "peachpuff1"
## [532] "peachpuff2"           "peachpuff3"           "peachpuff4"
## [535] "peru"                 "pink"                 "pink1"
## [538] "pink2"                "pink3"                "pink4"
## [541] "plum"                 "plum1"                "plum2"
## [544] "plum3"                "plum4"                "powderblue"
## [547] "purple"               "purple1"              "purple2"
## [550] "purple3"              "purple4"              "red"
## [553] "red1"                 "red2"                 "red3"
## [556] "red4"                 "rosybrown"            "rosybrown1"
## [559] "rosybrown2"           "rosybrown3"           "rosybrown4"
## [562] "royalblue"            "royalblue1"           "royalblue2"
## [565] "royalblue3"           "royalblue4"           "saddlebrown"
## [568] "salmon"               "salmon1"              "salmon2"
## [571] "salmon3"              "salmon4"              "sandybrown"
## [574] "seagreen"             "seagreen1"            "seagreen2"
## [577] "seagreen3"            "seagreen4"            "seashell"
## [580] "seashell1"            "seashell2"            "seashell3"
## [583] "seashell4"            "sienna"               "sienna1"
## [586] "sienna2"              "sienna3"              "sienna4"
## [589] "skyblue"              "skyblue1"             "skyblue2"
## [592] "skyblue3"             "skyblue4"             "slateblue"
## [595] "slateblue1"           "slateblue2"           "slateblue3"
## [598] "slateblue4"           "slategray"            "slategray1"
## [601] "slategray2"           "slategray3"           "slategray4"
## [604] "slategrey"            "snow"                 "snow1"
## [607] "snow2"                "snow3"                "snow4"
## [610] "springgreen"          "springgreen1"         "springgreen2"
## [613] "springgreen3"         "springgreen4"         "steelblue"
## [616] "steelblue1"           "steelblue2"           "steelblue3"
## [619] "steelblue4"           "tan"                  "tan1"
## [622] "tan2"                 "tan3"                 "tan4"
## [625] "thistle"              "thistle1"              "thistle2"
## [628] "thistle3"             "thistle4"              "tomato"
## [631] "tomato1"              "tomato2"               "tomato3"
## [634] "tomato4"              "turquoise"             "turquoise1"
## [637] "turquoise2"           "turquoise3"            "turquoise4"
## [640] "violet"               "violetred"             "violetred1"
## [643] "violetred2"           "violetred3"            "violetred4"
## [646] "wheat"                "wheat1"                "wheat2"
## [649] "wheat3"               "wheat4"                "whitesmoke"
## [652] "yellow"               "yellow1"               "yellow2"
## [655] "yellow3"              "yellow4"               "yellowgreen"
```

Let’s pick one of those and a random set of rgb() values and use them to set the colors manually.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, color = sex)) +
  geom_point() +
  # add scale_ function -- I used a name and a set of rgb() values to specify 2 col
ors
  scale_color_manual(values = c("tomato2", rgb(0.2, 0.3, 0.6)))
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



That’s handy, but you can spend a long time sampling colors and trying to assemble a color palette. Luckily R has lots of built in color palettes that we can call on to generate a set of colors to use. These can be divided into palettes roughly analogous to RGB color space (called hue-saturation-value, or HSV colors) and those based on human color perception (hue-chroma-luminance, or HCL colors). I don't have a strong opinion on these at present, but I will focus on HCL here.

What color palettes are available in R if we want to use the HCL color space? We can check using the function `hcl.pals()`.

```
hcl.pals()
```

```
##   [1] "Pastel 1"      "Dark 2"        "Dark 3"        "Set 2"        
##   [5] "Set 3"         "Warm"          "Cold"          "Harmonic"     
##   [9] "Dynamic"       "Grays"         "Light Grays"   "Blues 2"      
##  [13] "Blues 3"       "Purples 2"     "Purples 3"     "Reds 2"       
##  [17] "Reds 3"        "Greens 2"      "Greens 3"      "Oslo"         
##  [21] "Purple-Blue"   "Red-Purple"    "Red-Blue"      "Purple-Orange"
##  [25] "Purple-Yellow" "Blue-Yellow"   "Green-Yellow"  "Red-Yellow"   
##  [29] "Heat"          "Heat 2"        "Terrain"       "Terrain 2"    
##  [33] "Viridis"       "Plasma"        "Inferno"       "Rocket"       
##  [37] "Mako"          "Dark Mint"     "Mint"          "BluGrn"       
##  [41] "Teal"          "TealGrn"       "Emrld"         "BluYl"        
##  [45] "ag_GrnYl"      "Peach"         "PinkYl"        "Burg"         
##  [49] "BurgYl"        "RedOr"         "OrYel"         "Purp"         
##  [53] "PurpOr"        "Sunset"        "Magenta"       "SunsetDark"   
##  [57] "ag_Sunset"     "BrwnYl"        "YlOrRd"        "YlOrBr"       
##  [61] "OrRd"          "Oranges"       "YlGn"          "YlGnBu"       
##  [65] "Reds"          "RdPu"          "PuRd"          "Purples"      
##  [69] "PuBuGn"        "PuBu"          "Greens"        "BuGn"         
##  [73] "GnBu"          "BuPu"          "Blues"         "Lajolla"      
##  [77] "Turku"         "Hawaii"        "Batlow"        "Blue-Red"     
##  [81] "Blue-Red 2"    "Blue-Red 3"    "Red-Green"     "Purple-Green" 
##  [85] "Purple-Brown"  "Green-Brown"   "Blue-Yellow 2" "Blue-Yellow 3"
##  [89] "Green-Orange"  "Cyan-Magenta"  "Tropic"        "Broc"         
##  [93] "Cork"          "Vik"           "Berlin"        "Lisbon"       
##  [97] "Tofino"        "ArmyRose"      "Earth"         "Fall"         
## [101] "Geyser"        "TealRose"      "Temps"         "PuOr"         
## [105] "RdBu"          "RdGy"          "PiYG"          "PRGn"         
## [109] "BrBG"          "RdYlBu"        "RdYlGn"        "Spectral"     
## [113] "Zissou 1"      "Cividis"       "Roma"
```

Ooooh, lots of options! Let's pick one and try to see if we can get a set of colors from it. I'm going to try "Harmonic", because that sounds like a cool name. Use the `hcl.colors()` function to specify the number of colors you need (n = …) and the palette you'd like them from (palette = …). Try one (or a few) of your own!

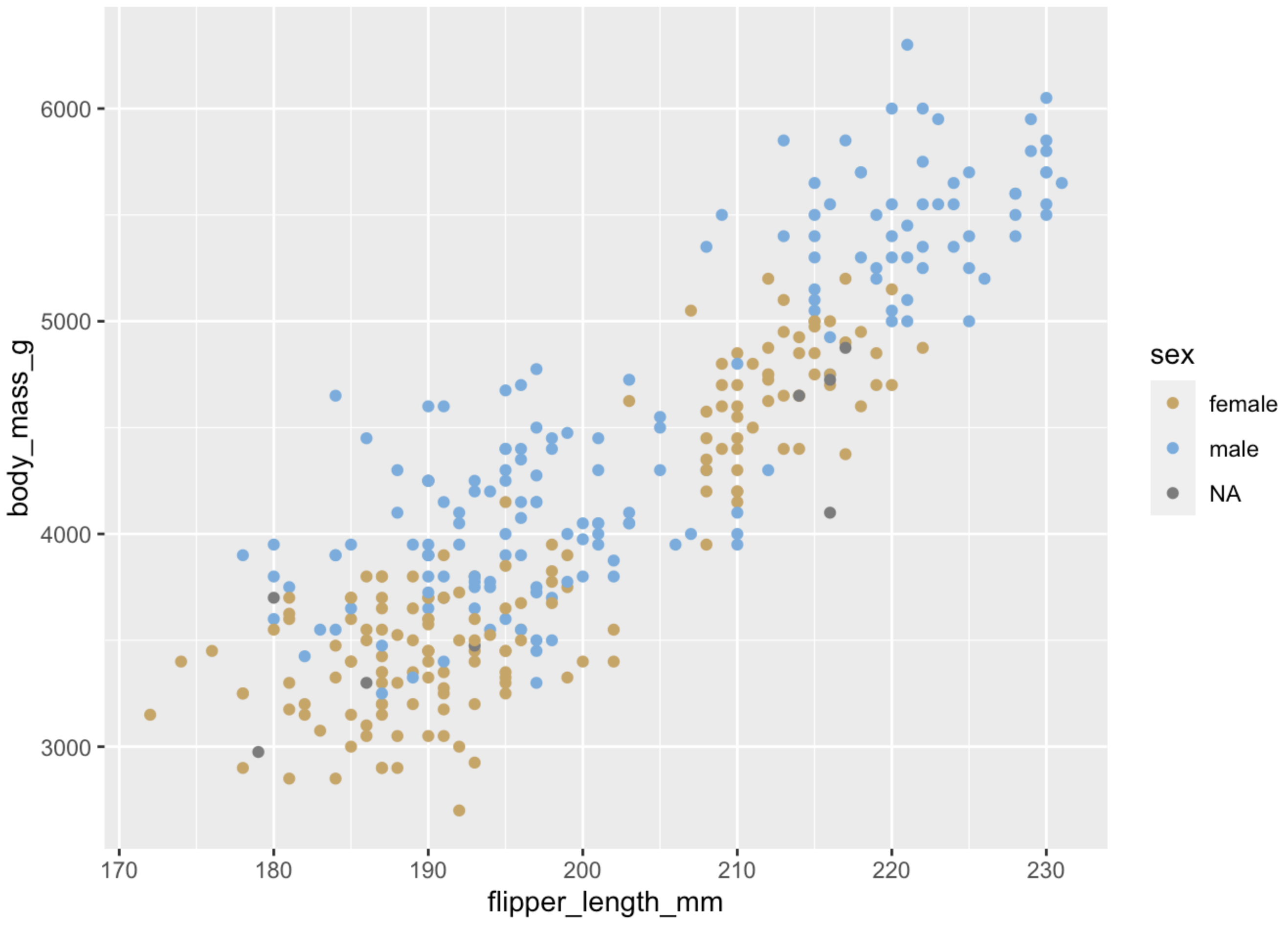
```
hcl.colors(n = 2, palette = "Harmonic")
```

```
## [1] "#C7A76C" "#7DB0DD"
```

So this gives us hex codes for our colors. Nice! We can enter these manually into our `scale_` function like we did above, or we can nest the `hcl.colors()` function inside the `scale_` function to skip a step.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, color = sex)) +
  geom_point() +
  scale_color_manual(values = hcl.colors(n = 2, palette = "Harmonic"))
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



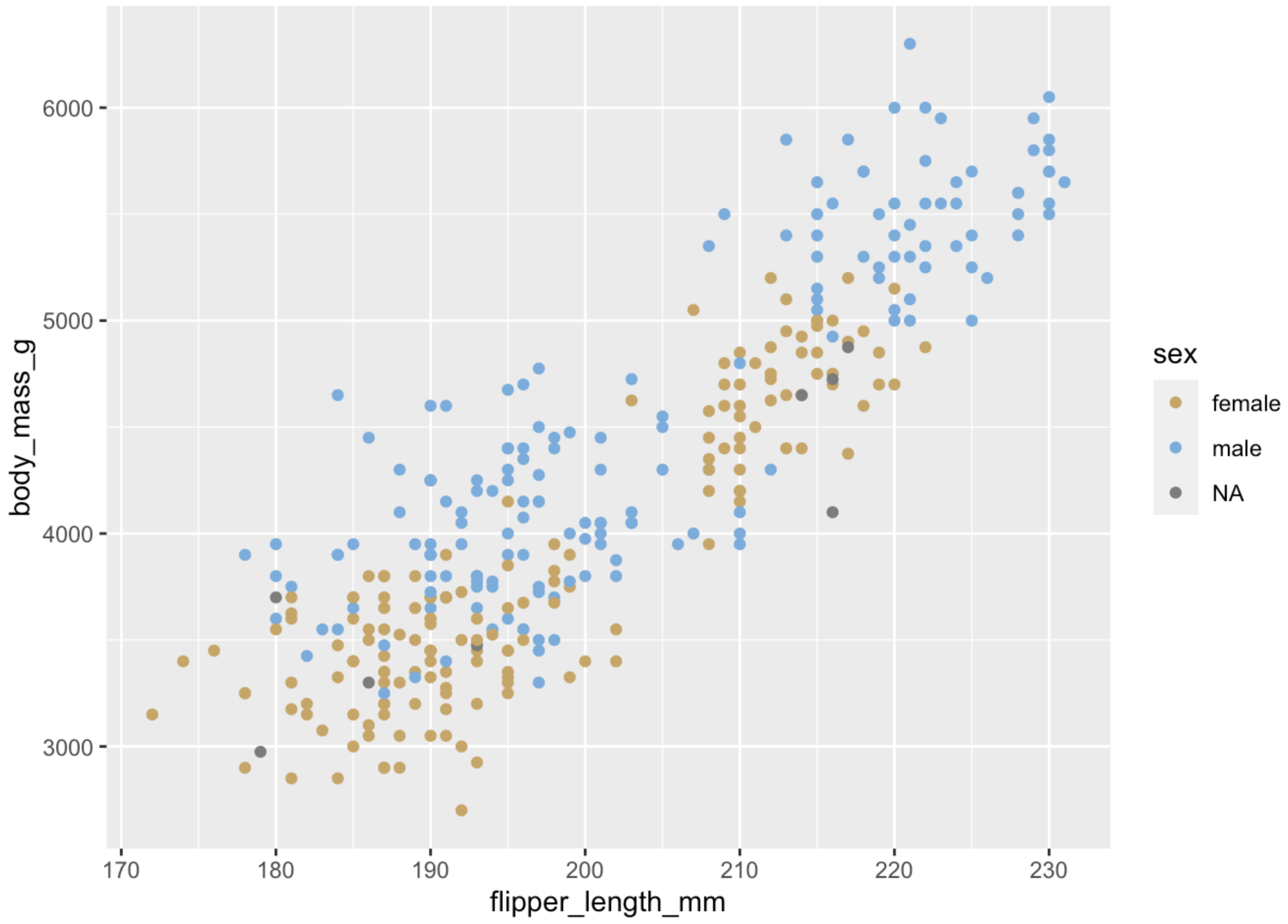
```
# the code above is equivalent to running the code below. The below uses the hex co
des.
# ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, color = sex)) + #
add ", color = sex" to the aesthetic mapping in the ggplot() function
#   geom_point() + scale_color_manual(values = c("#C7A76C", "#7DB0DD"))
```

So this might not be my favorite color palette, but now we know how to create our own or use one of the many existing ones.

Notice that ggplot won't apply a color to instances where the variable mapped to the color is NA (see the greyed out points?).

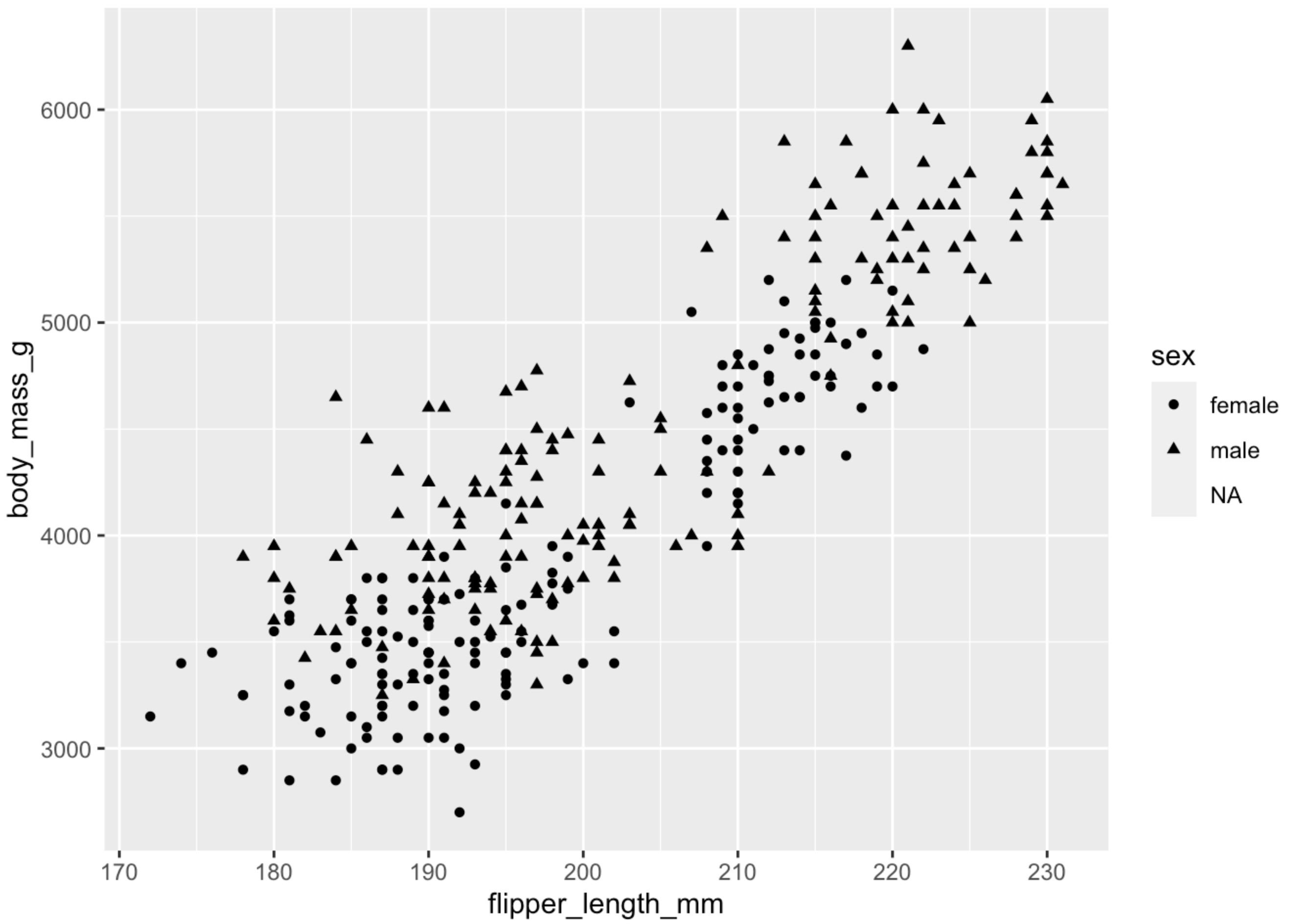
Notice also that ggplot is warning you that there are missing values that are dropped during plotting (i.e., their x or y coordinates are unknown). If we want to formally address this, we can add the `na.rm = TRUE` option to our `ggplot()` function (affects all layers) or to the corresponding `geom_`.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, color = sex)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 2, palette = "Harmonic"))
```



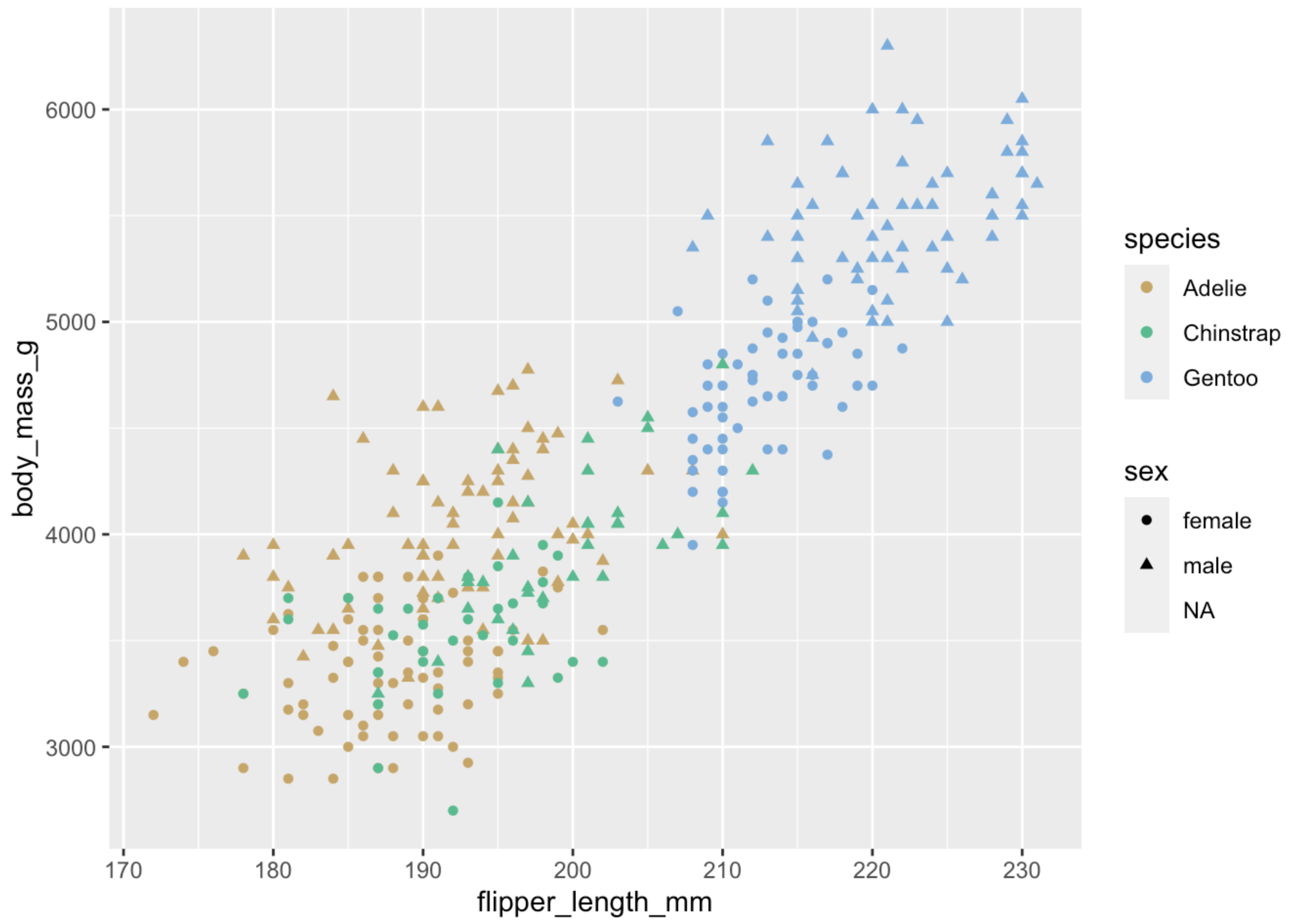
Let’s try mapping sex to a point shape, too, since you will sometimes need to do this (e.g., for printing in greyscale)

```
# put ", shape = sex" in the aesthetic mapping in the ggplot() function instead of
color.
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex)) +
  geom_point(na.rm = TRUE)
```



Finally, we can combine shapes and colors to show even more information (though there may be clearer ways to do this– see faceting later!). Let’s keep shapes as representative of sex, but let colors vary according to species. Note that we’ll now need 3 colors, one for each of the 3 species.

```
# now with shape = sex and color = species
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic"))
```



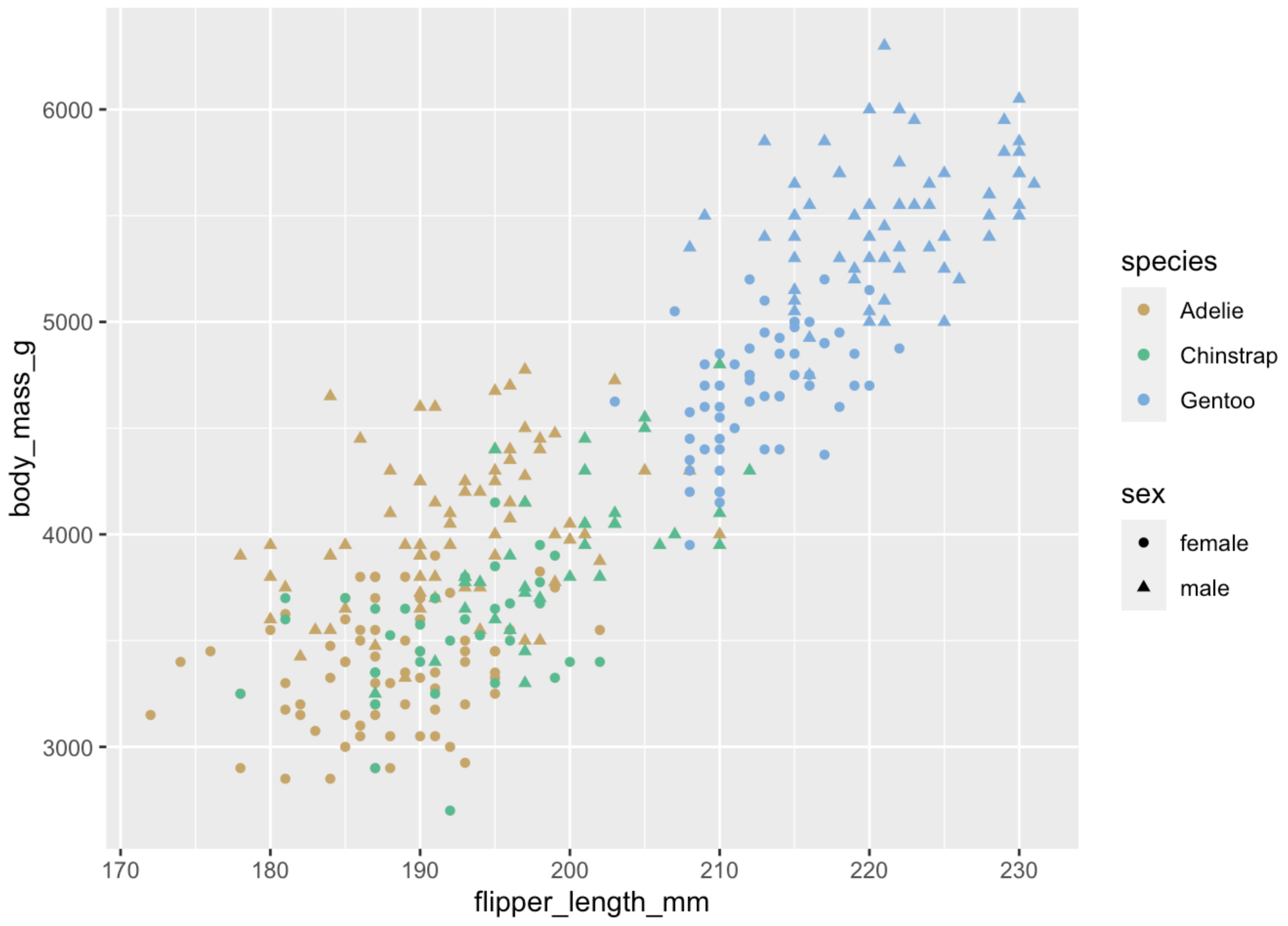
You'll notice that when we mapped sex to the shape aesthetic for our points, the NAs for sex don't get assigned a symbol type. This means they don't show up in the plot, but NA still shows up in the legend for some reason. That's pretty annoying. In regular practice, this wouldn't really be an issue because we typically would have already filtered our data to remove observations where sex was unknown. However, our present issue gives us an opportunity to explore how to adjust other aspects of our plots like the axes and legends.

Fine Tuning

Legends

Let's take care of that legend first– it's driving me crazy! We can adjust the legend using the `breaks =` argument inside the `scale_` function for our corresponding aesthetic. For example, here we were using sex as a shape. We are getting three discrete "breaks" in sex – female, male, and NA, but we only want female and male. So we need to use something like `scale_shape_discrete(breaks = c("female", "male"))` to specify our own breaks.

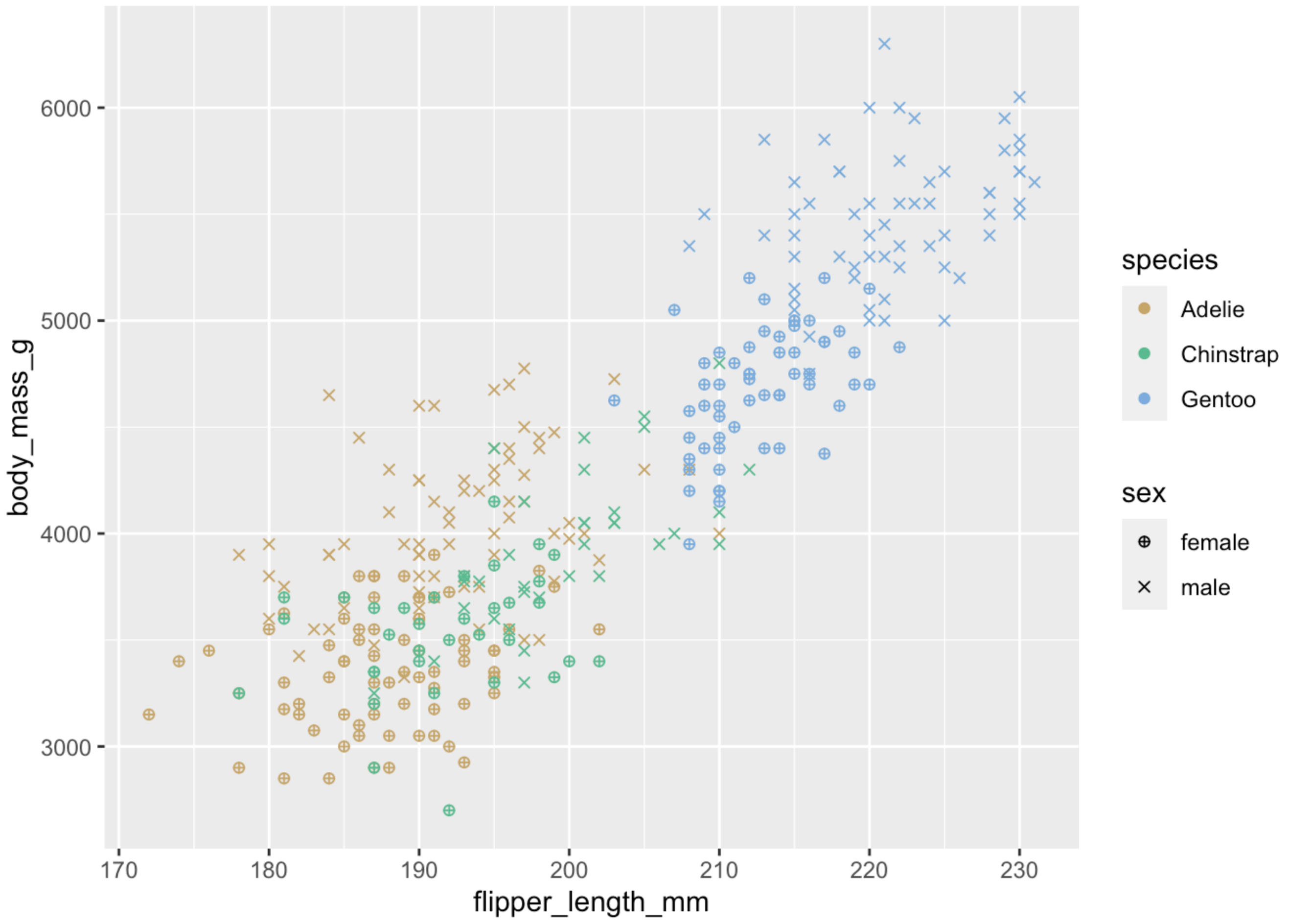
```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_discrete(breaks = c("female", "male"), )
```



Much better. Here, we changed the legend as a workaround to a problem you shouldn't encounter too often, but you'll often need to modify them as part of your plotting anyway.

Don't like those symbol types? Use `scale_shape_manual()` to set them using the `values =` argument like we did with `scale_color_manual()` above. This is often necessary when you have a lot of groups, because `scale_shape_discrete()` only generates up to 6 shapes and then starts recycling them! Try typing ?pch to see some of the shape types available in R.

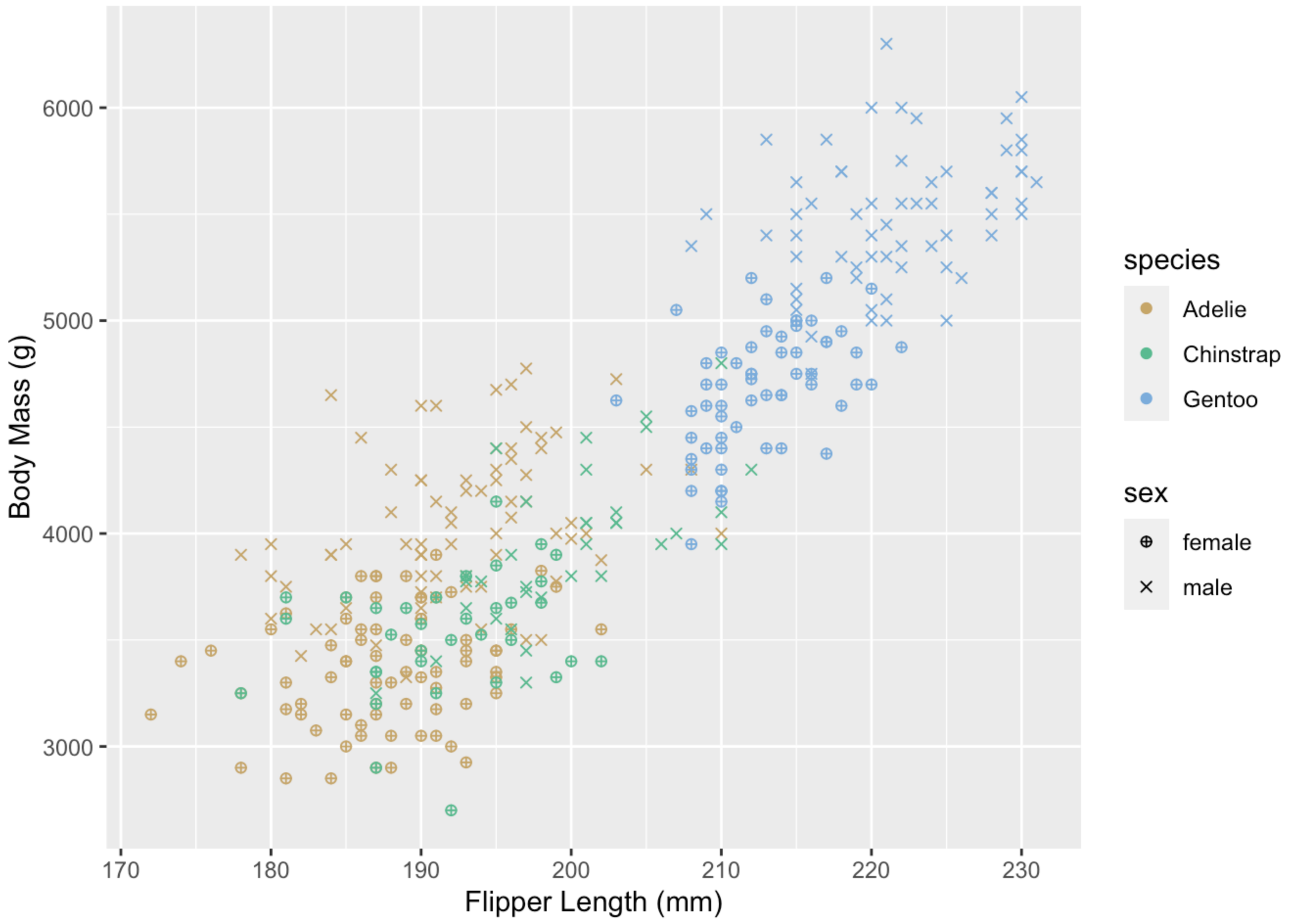
```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4))
```



Axis Labels

This one is easy. To specify the x-axis label, just add `+ xlab("YOUR LABEL HERE")` to your plot code. I bet you can guess how to change the y-axis label now, too!

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)")
```



```
## The below is equivalent, but specifies the x and y labels separately.
# ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, colo
r = species)) +
#   geom_point(na.rm = TRUE) +
#   scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
#   scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
#   xlab("Flipper Length (mm)") +
#   ylab("Body Mass (g)")
```

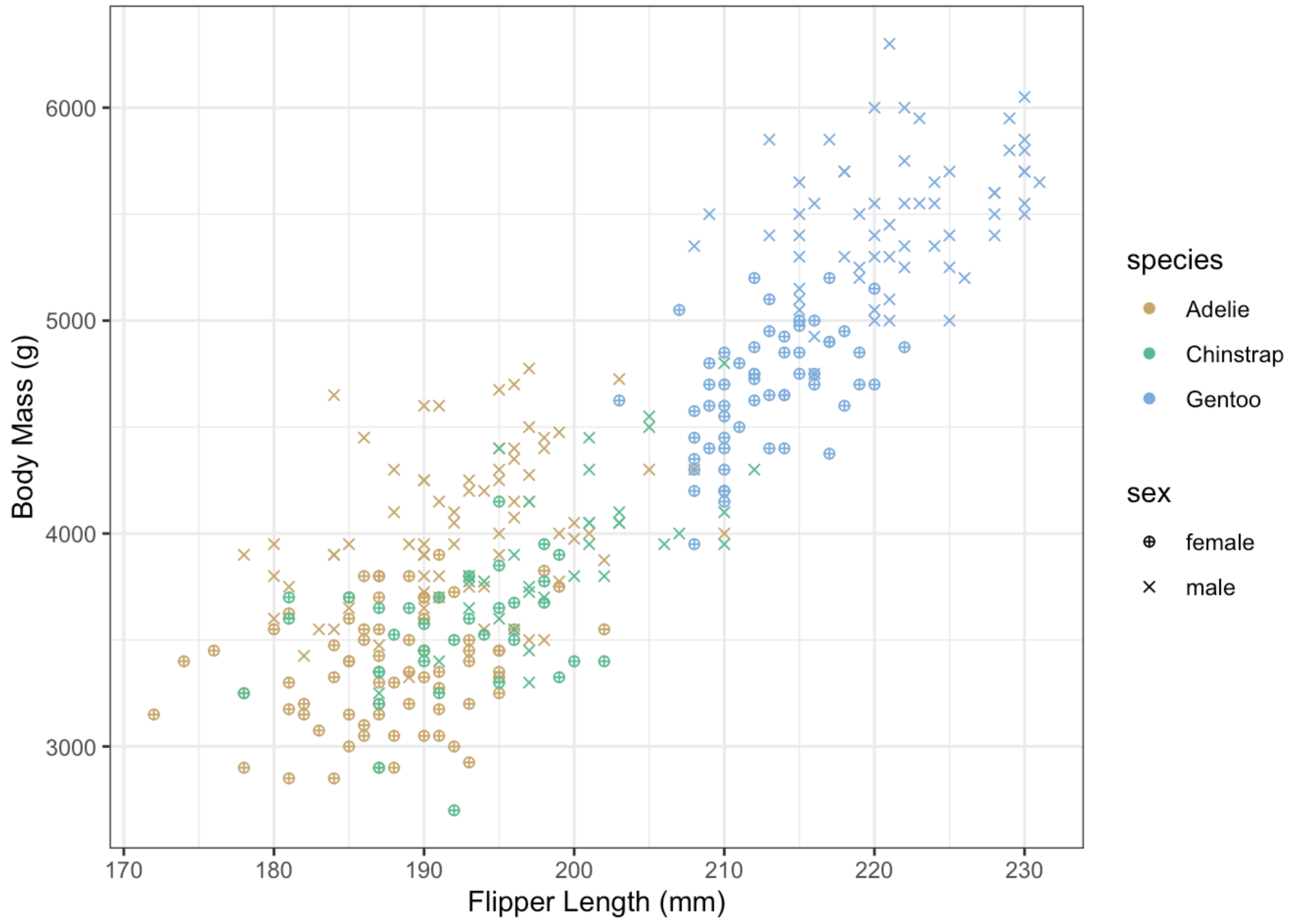
Now this is starting to look like a publication-quality graphic! How exciting.

Themes

Unfortunately the default ggplot “theme” (the collection of non-data options like background colors, grid lines, font sizes, etc.) isn’t all that wonderful. We can change these using default themes, specifying our own, or a combination of the two.

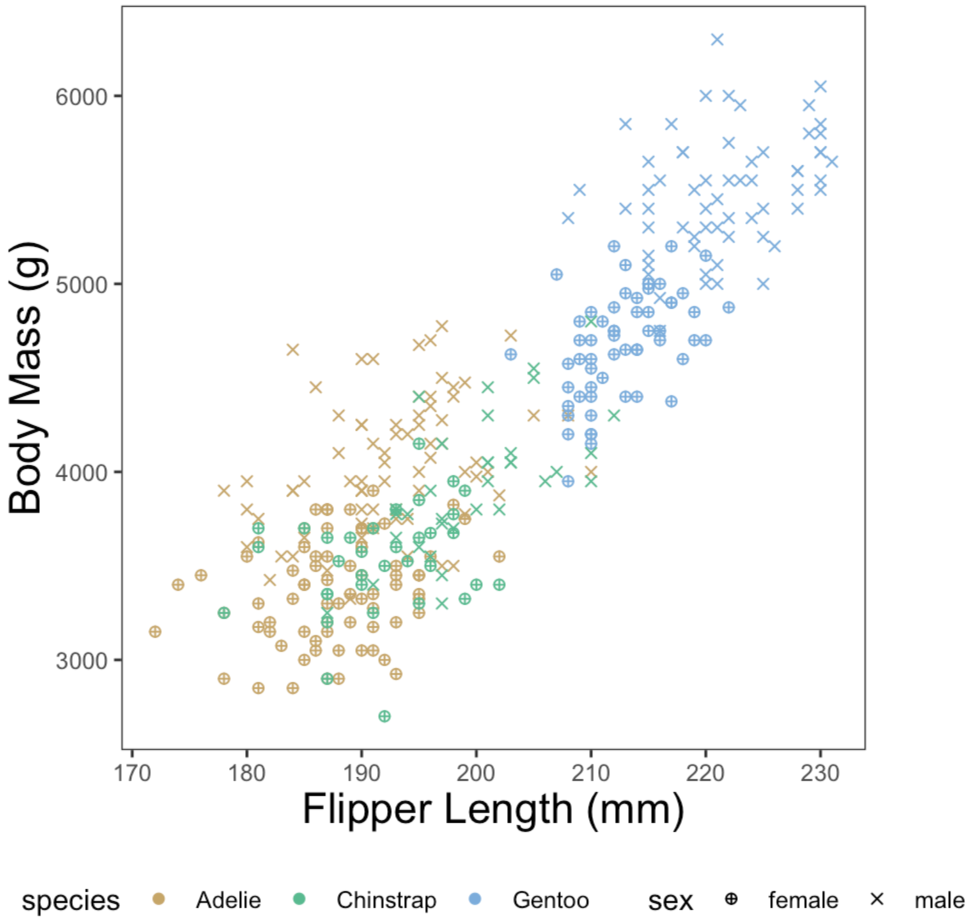
A useful starting place is `theme_bw()`. This black and white theme gets rid of the grey background.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw()
```



But I'd like to get rid of those gridlines too, and maybe change the axis label font size… and what if I want to move the legend? Let's dive into `theme()` to modify `theme_bw()`!

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(), # use panel.grid.major argument set to
element_blank() to get rid of major grid
        panel.grid.minor = element_blank(), # use panel.grid.minor argument set to
element_blank() to get rid of minor grid
        axis.title = element_text(size = 16), # use axis.title argument set to elem
ent_text() to specify options like size, color, etc.
        legend.position = "bottom", # use legend.position set to bottom, top, left,
right, or x,y coords (scaled 0 to 1). Use "none" to omit.
        aspect.ratio = 1) # use aspect.ratio to set the height:width ratio of the p
lot.
```



There are *a lot* of theme options available. This is where you can really stretch your creative muscles and have fun! Check out the ?theme() page for a staggering list of tweakable plotting elements. Don't worry–unless you're creating some really complex graphics (or just having fun with it), you can usually make attractive and informative plots with code that is not much more complicated than the code above.

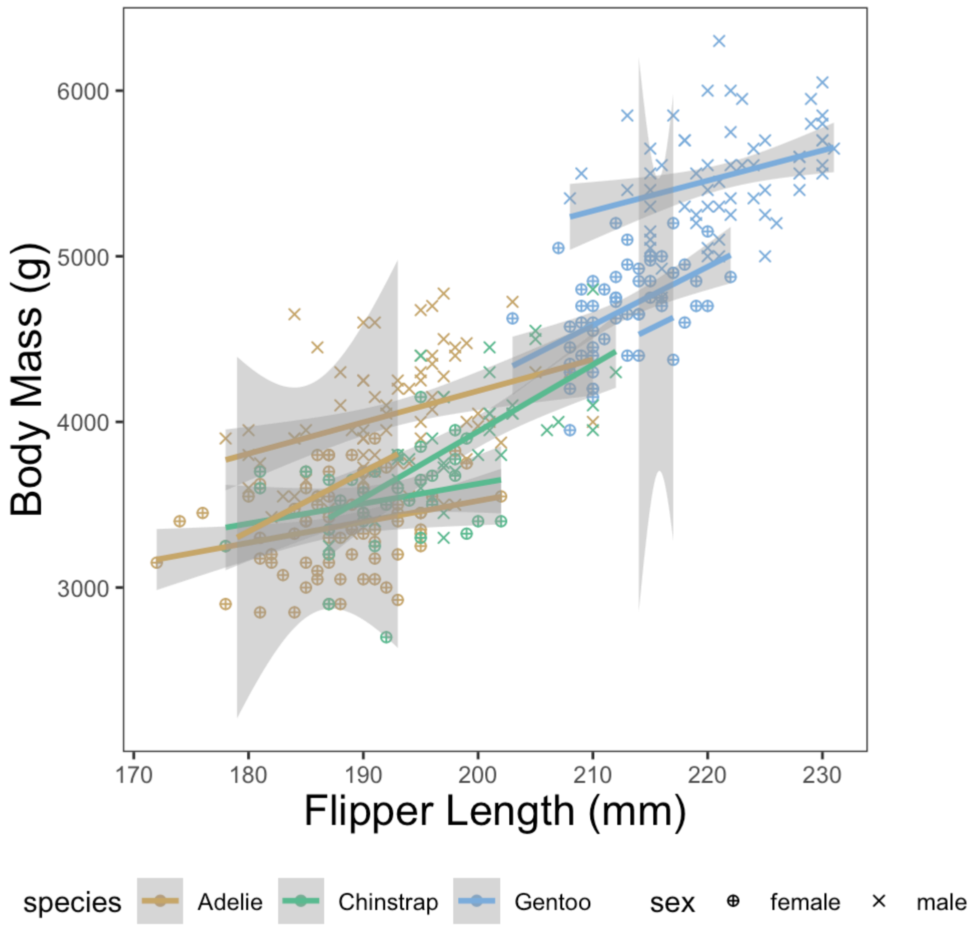
Trendlines

It's worth a brief aside to note that ggplot also has built-in tools for visualizing trends in your plotted data. One helfpul function for this is `geom_smooth()` . This function takes a `method =` argument that allows you to specify how you want R to calculate the trendline. Popular options include a linear model (lm), loess regression, general additive model (gam), and generalized linear models (glm). Let's try adding a linear model fit to our plot.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_text(size = 16),
        legend.position = "bottom",
        aspect.ratio = 1) +
  geom_smooth(method = "lm", se = TRUE) # the additional argument se = TRUE tells R
to show the confidence intervals too.
```

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

```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```



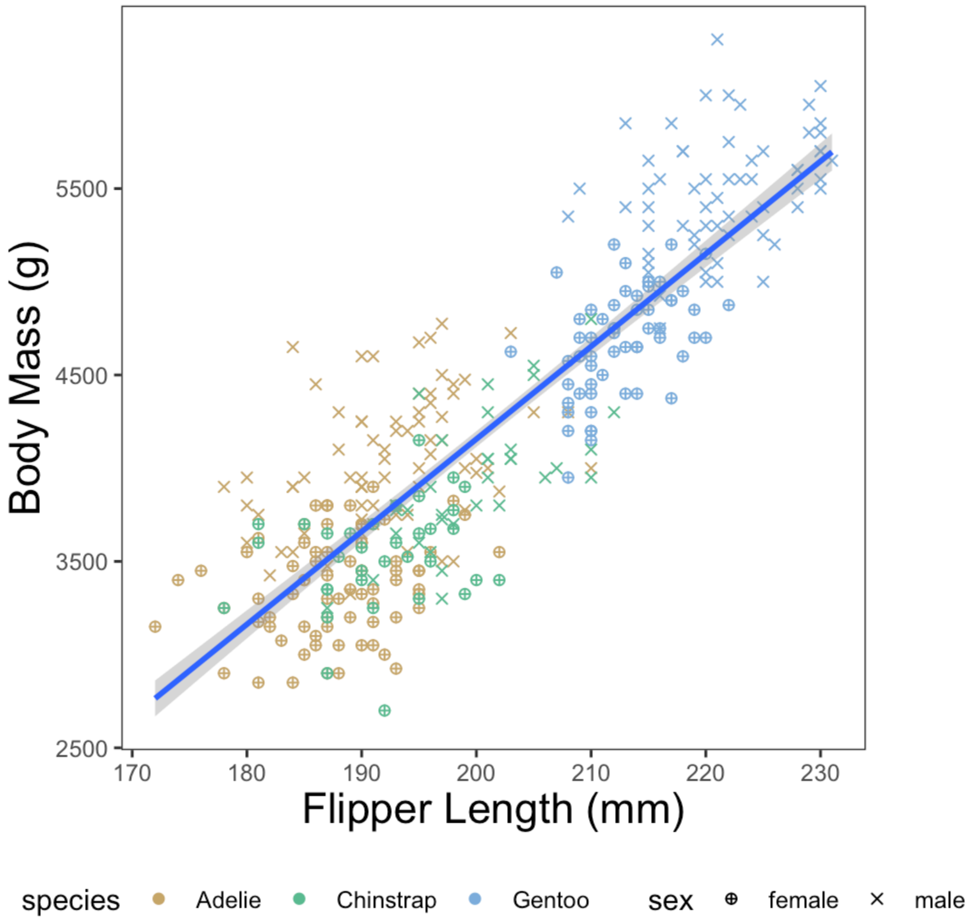
Whoa, what happened here? Well, because we map sex and species onto shape and color, when we call geom_smooth() it assumes that we would like to do this for each group separately. You'll also notice there are more than 2 groups for Adelies and Gentoos – those NAs are getting treated as separate groups! Did I mention that filtering data is important?

We can override the aesthetics we previously specified by defining new aesthetics in geom_smooth() and asking it to bypass the old aesthetics by setting `inherit.aes = FALSE`. This will leave all preceding lines of code unaffected, just our geom_smooth() changes. In this way you can build up layers based on different variables, subsets of data, etc.

```
ggplot(penguins_df, aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color
= species)) +
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_text(size = 16),
        legend.position = "bottom",
        aspect.ratio = 1) +
  # add inherit.aes = FALSE to prevent geom_smooth() from inheriting the previous a
esthetics
  # define new aesthetics specific to geom_smooth() using aes() inside geom_smooth
()
  geom_smooth(inherit.aes = FALSE, aes(x = flipper_length_mm, y = body_mass_g), met
hod = "lm", se = TRUE)
```

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

```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```



Great, now we have points grouped by sex and species, but one trend line to rule them all.

This is probably a bad idea though. Can we split the trendlines up by sex and species again, but present them neater? We need `facet_wrap()` !

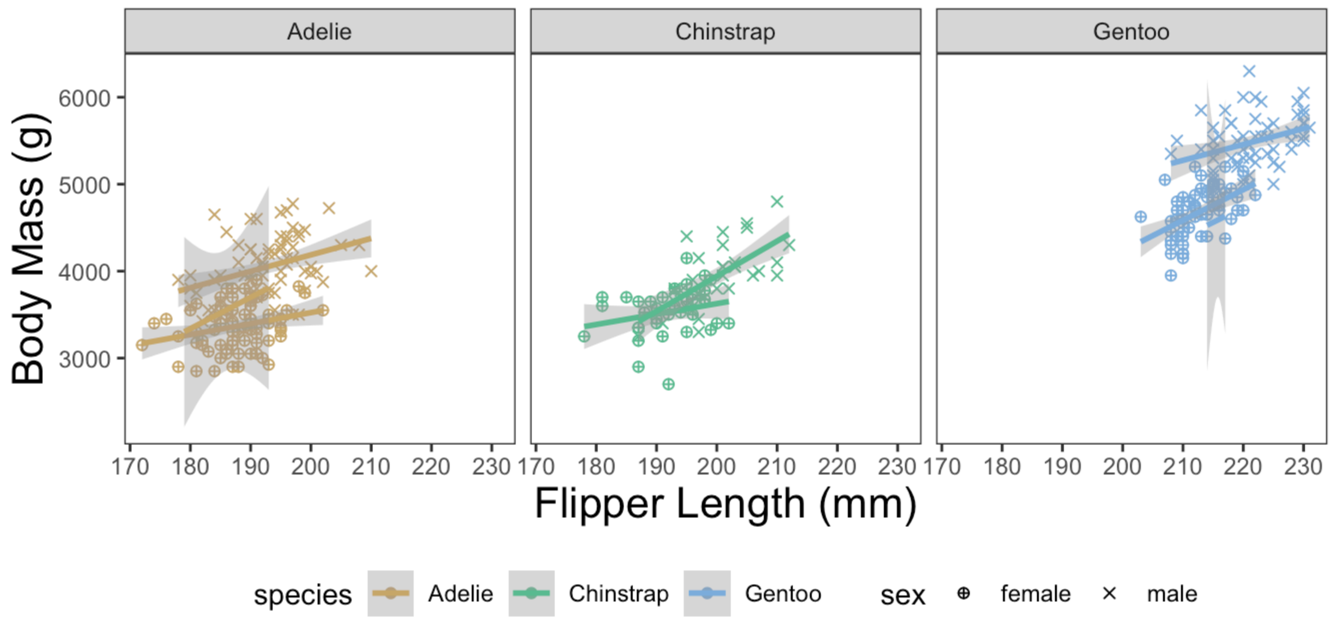
Facetting

“Facetting” is a tool in ggplot to let you create multiple panels of a plot with ease. The syntax is to add `facet_wrap(~ YOUR FACETTING VARIABLE HERE)` to your ggplot code. Let’s leave the two sexes in the same plot, but facet by species so each species has a separate panel.

```
ggplot(penguins_df,
       aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color = species)) +
# can still color by species for emphasis
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_text(size = 16),
        legend.position = "bottom",
        aspect.ratio = 1) +
  # change geom_smooth back so that it uses the "global" aesthetics
  geom_smooth(method = "lm", se = TRUE) +
  facet_wrap(~ species) # make subplots for each species using facet_wrap().
```

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

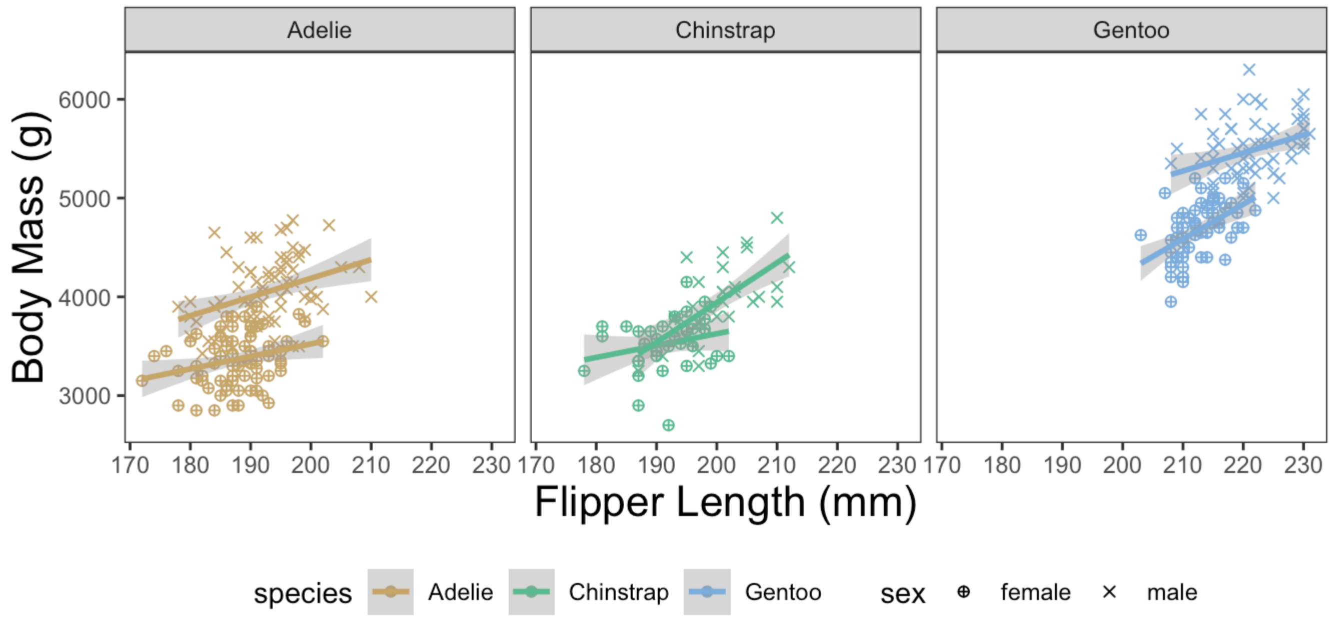
```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```



Finally, let's get rid of those cases where the sex variable is NA. This isn't really a ggplot step, but it's helpful to see how you could give ggplot a subset of your data to start with. We'll use `penguins_df[!is.na(penguins_df$sex),]` as a way to select only the rows in our data set for which the sex variable is *not* (notice the logical negation symbol `!`) NA.

```
ggplot(penguins_df[!is.na(penguins_df$sex),],
       aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color = species)) +
# can still color by species for emphasis
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_text(size = 16),
        legend.position = "bottom",
        aspect.ratio = 1) +
  # change geom_smooth back so that it uses the "global" aesthetics
  geom_smooth(method = "lm", se = TRUE) +
  facet_wrap(~ species) # make subplots for each species using facet_wrap().
```

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



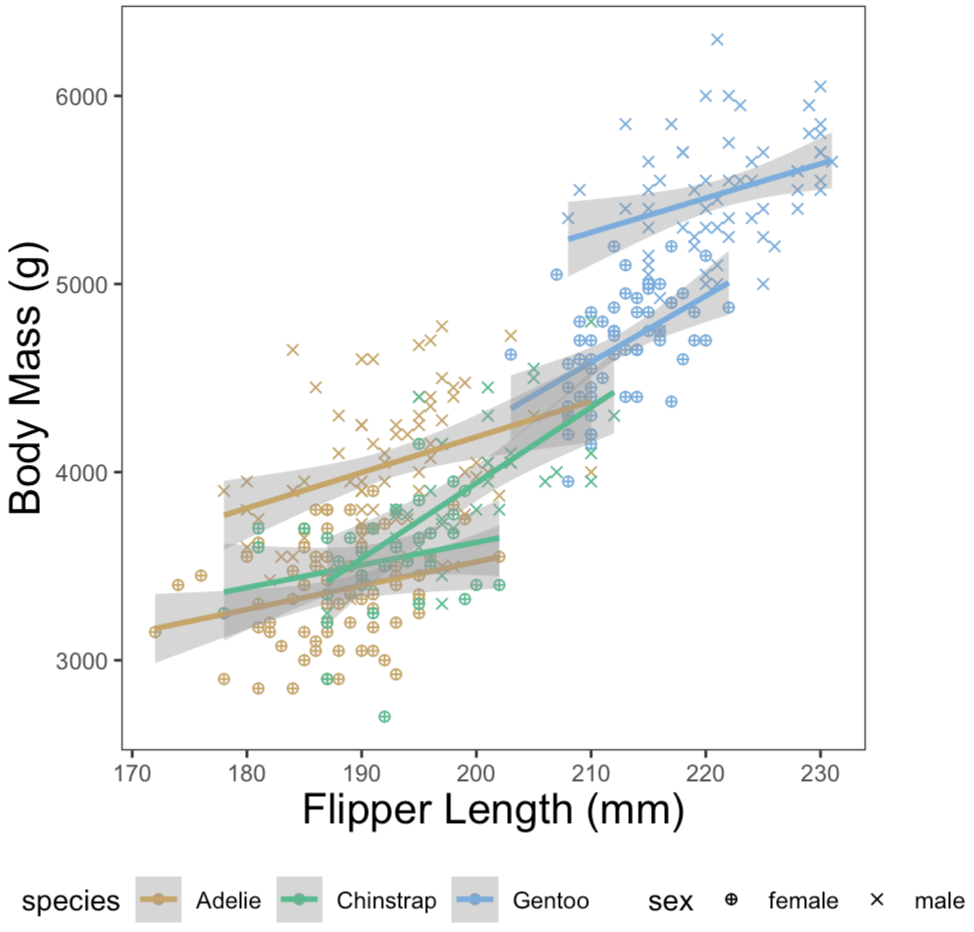
There we have it! It looks like body mass increases more with flipper length in male Chinstraps compared to female Chinstraps (notice the steeper slope of the line running through the x's compared to the crossed circles), but this is weaker or maybe even opposite in the others. Weird!

Saving ggplots

We can save a ggplot to our local environment at any time by saving it as an object. One really handy feature of this is that we can then continue to add layers to the plot by just adding them to that object.

```
# create an object, called pbase here, to store a graphic. Let's store the graphic
with sex mapped to shape and color to species, with a trendline, but no facetting.
pbase = ggplot(penguins_df[!is.na(penguins_df$sex),],
       aes(x = flipper_length_mm, y = body_mass_g, shape = sex, color = species)) +
# can still color by species for emphasis
  geom_point(na.rm = TRUE) +
  scale_color_manual(values = hcl.colors(n = 3, palette = "Harmonic")) +
  scale_shape_manual(breaks = c("female", "male"), values = c(10, 4)) +
  labs(x = "Flipper Length (mm)", y = "Body Mass (g)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_text(size = 16),
        legend.position = "bottom",
        aspect.ratio = 1) +
  # change geom_smooth back so that it uses the "global" aesthetics
  geom_smooth(method = "lm", se = TRUE)
# look at the object
pbase
```

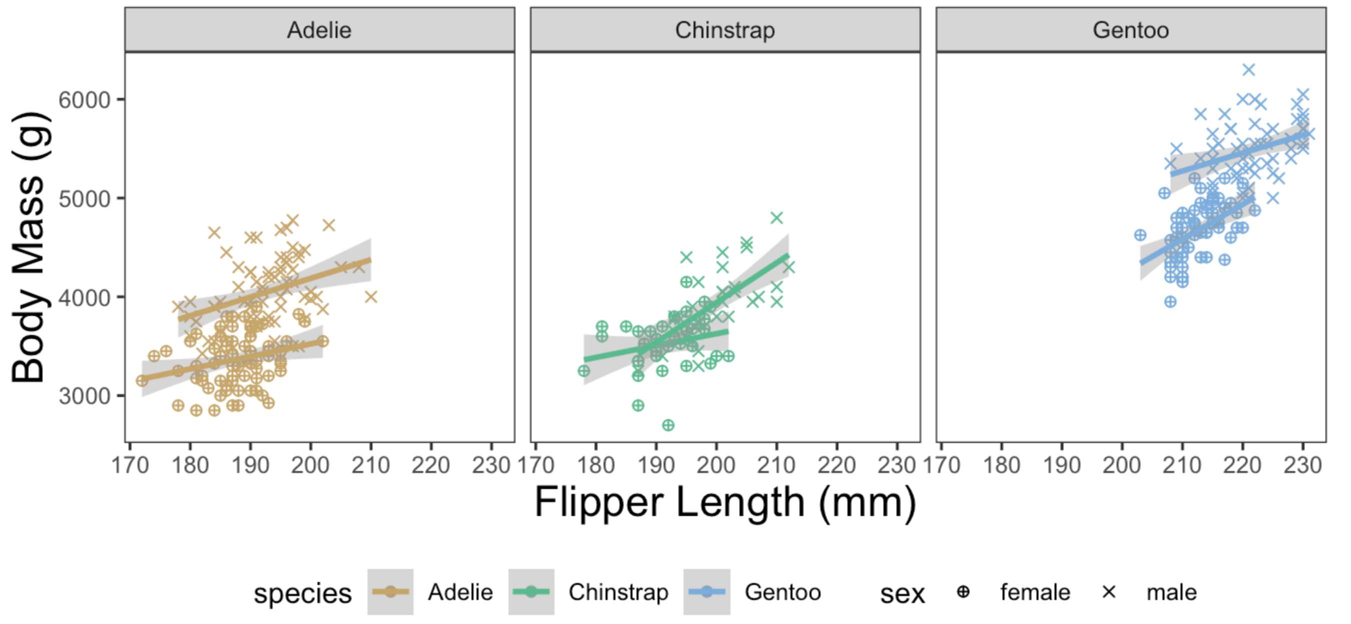
```
## `geom_smooth()` using formula 'y ~ x'
```



Now we can call on our object `pbase` to add more plot elements. Let’s add facetting by species.

```
pbase + facet_wrap(~ species)
```

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



So when we get to a good "stopping point" with our graphic building, we can save a ggplot graphic as an object to shorten the code in further tweaking. Handy!

How about saving it for use outside of R? We can use the function `ggsave()`.

NOTE: `ggsave()` always writes the last ggplot you *displayed* to the file name and format you specify. It's good practice to save your ggplot to a local object, view the object, and then run `ggsave()`.

```
ptop = pbase + facet_wrap(~ species) # save our final plot as an object by adding f
acet_wrap() to the base plot (pbase)
ptop # view ptop
ggsave("bodyMassbyFlipperLength.png") # pick a file name, including the extension t
o choose the file format. I used .png here.
```

You can also edit some of the saved file's attributes. For example, we can specify the width and height of the saved plot and the units in which these dimensions are measured.

```
ggsave("bodyMassbyFlipperLength.pdf", height = 4, width = 6, units = "in") # specif
y a 4x6" PDF of the plot, in this case.
```