

Bird Cave Workshop 2021

Creating Figures as an Intro to R

Using the **ggplot2** package

Steffi LaZerte



Artwork by [@allison_horst](#)

Introductions

Instructor

Dr. Steffi LaZerte

- Background in Biology (Animal Behaviour)
- Working with R since 2007
- Professional R programmer/consultant since 2017



Introductions

Assistant

Dr. Alex Koiter

- Physical Geographer
- Working with R since 2010
- Assistant Professor in Geography and Environment, Brandon University



What about you?

- Name
- Background (Area of study, etc.)
- Familiarity with R (or other programming languages)
 - I've heard of R
 - I've used R
 - I use R all the time

Outline

1. A little about R

2. Creating figures with `ggplot2`

3. Combining figures with `patchwork`

4. Saving figures

Outline

1. A little about R

2. Creating figures with `ggplot2`

3. Combining figures with `patchwork`

4. Saving figures

Taken this or a similar workshop before?

During activities consider...

- Extra activities labeled "Too Easy?"
- Using your own data
- Exploring other aspects of `ggplot2` that interest you

Feel free to ask questions even if it's not the "official" activity!

What is R?

R is Programming language

A programming **language** is a way to give instructions in order to get a computer to do something

- You need to know the language (i.e., the code)
- Computers don't know what you mean, only what you type (unfortunately)
- Spelling, punctuation, and capitalization all matter!

For example

R, what is 56 times 5.8?

```
56 * 5.8
```

```
## [1] 324.8
```

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

R, multiply this value by 6

```
steffis_mean * 6
```

```
## [1] 15
```

Why R?

R is hard

```
# Get in circle around city
circle <- data.frame()
cutoff <- 10
for(i in unique(gps$region)) {
  n <- nrow(gps[gps$region == i,]) ##number of IDs
  if(i == "wil") tmp <- geocode("Williams Lake, Canada")
  if(i == "kam") tmp <- geocode("Kamloops, Canada")
  if(i == "kel") tmp <- geocode("Kelowna, Canada")
  temp <- data.frame()
  for(a in 1:n){
    if(a <= cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                       lat = tmp$lat,
                                                       bearing = (a*(360/(cutoff))-360/(cutoff)),
                                                       dist = 20,
                                                       dist.units = "km",
                                                       model = "WGS84"))
    if(a > cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                       lat = tmp$lat,
                                                       bearing = ((a-cutoff)*(360/(max(table(gps$region
)))-10))-360/(max(table(gps$region))-cutoff)),
                                                       dist = 35,
                                                       dist.units = "km",
                                                       model = "WGS84"))
  }
  circle <- rbind(circle, cbind(temp,
                                 region = i,
                                 hab = gps$hab[gps$region == i],
                                 spl = gps$spl.orig[gps$region == i],
                                 lon = tmp$lon,
                                 lat = tmp$lat)))
}
circle
```

Why R?

But R is powerful (and reproducible)!

The screenshot shows the 'Global Environment' pane in RStudio. It is organized into three sections: 'Data', 'Values', and 'Functions'. The 'Data' section contains two entries: 'fish' and 'telem_total'. The 'fish' entry is described as '172 obs. of 13 variables'. The 'telem_total' entry is described as '12950046 obs. of 10 variables'. A large green oval highlights the 'telem_total' entry. The 'Values' section contains one entry: 'tz' with the value '"Etc/GMT+8"'. The 'Functions' section contains one entry: 'load_data' with the definition 'function (x)'. Each entry has a small icon to its right.

Category	Object	Description	Icon
Data	fish	172 obs. of 13 variables	
	telem_total	12950046 obs. of 10 variables	
Values	tz	"Etc/GMT+8"	
Functions	load_data	function (x)	

Why R?

But R is powerful (and reproducible)!

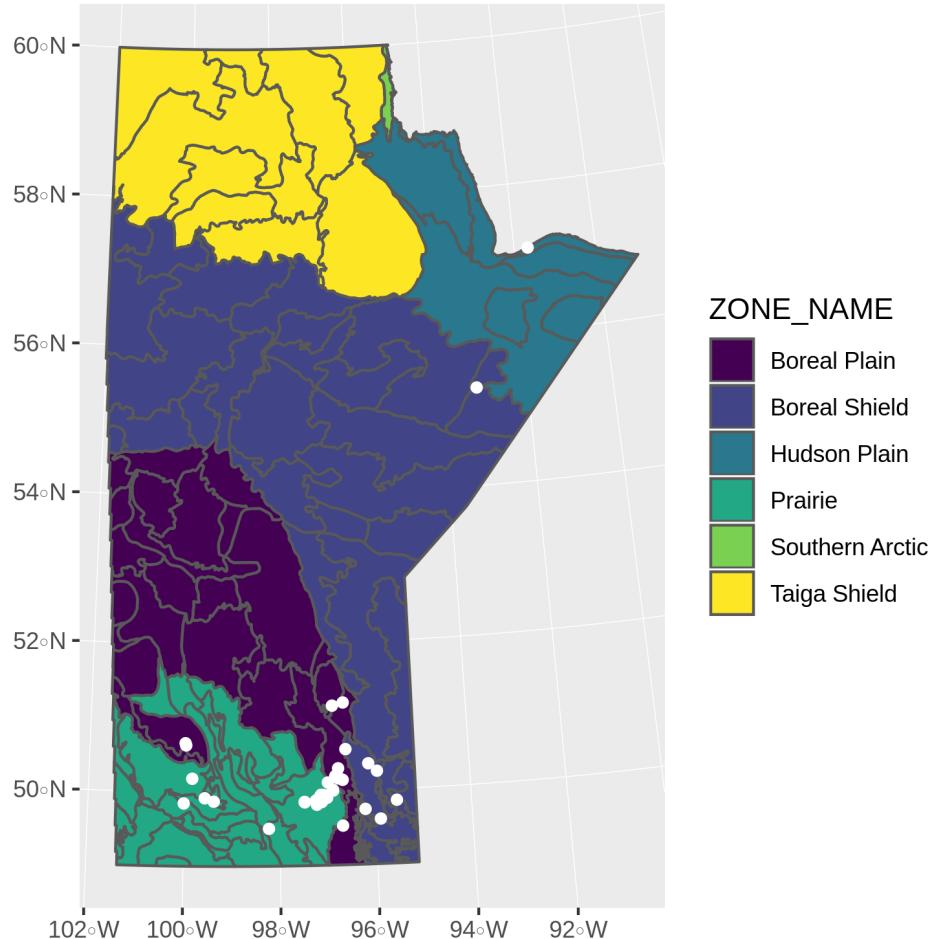
The screenshot shows the 'Global Environment' pane in RStudio. The 'Data' section is expanded, displaying the following entries:

Object	Description
<code>fish</code>	172 obs. of 13 variables
<code>telem_total</code>	12950046 obs. of 10 variables

A green oval highlights the 'telem_total' entry, specifically the value '12950046 obs. of 10 variables'. The 'Values' and 'Functions' sections below are also visible.

Why R?

R is also beautiful



Why R?

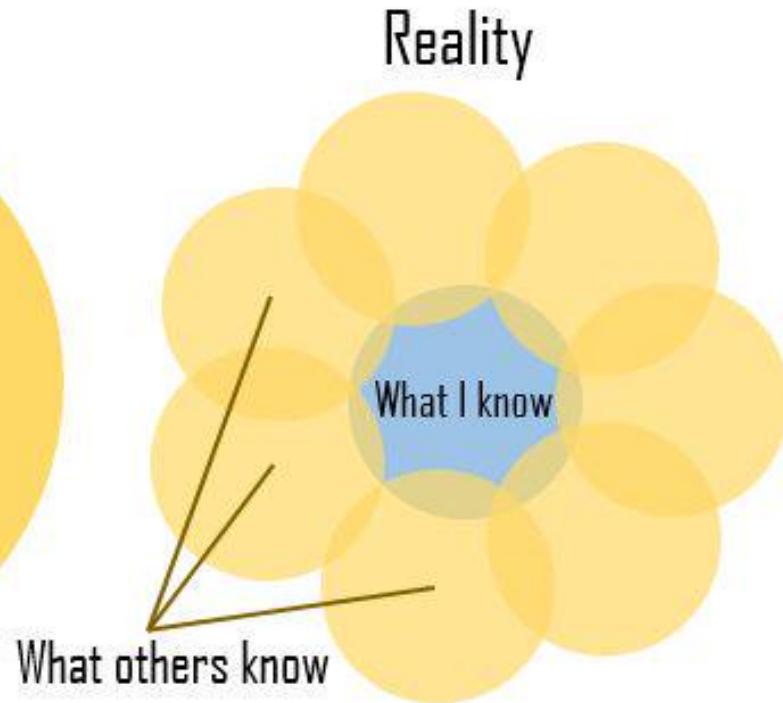
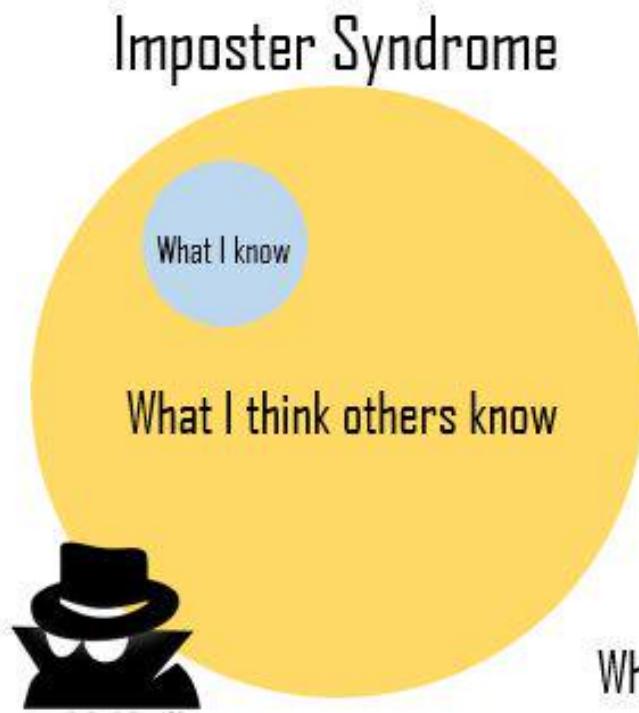
R is affordable (i.e., free!)

R is available as Free Software under the terms of the [Free Software Foundation's GNU General Public License](#) in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

ImpostR Syndrome

ImpostR
Syndrome

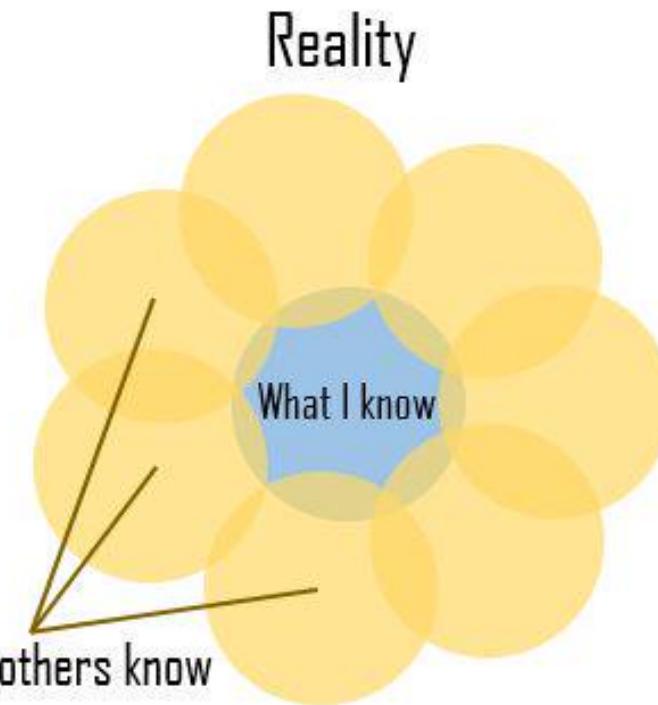
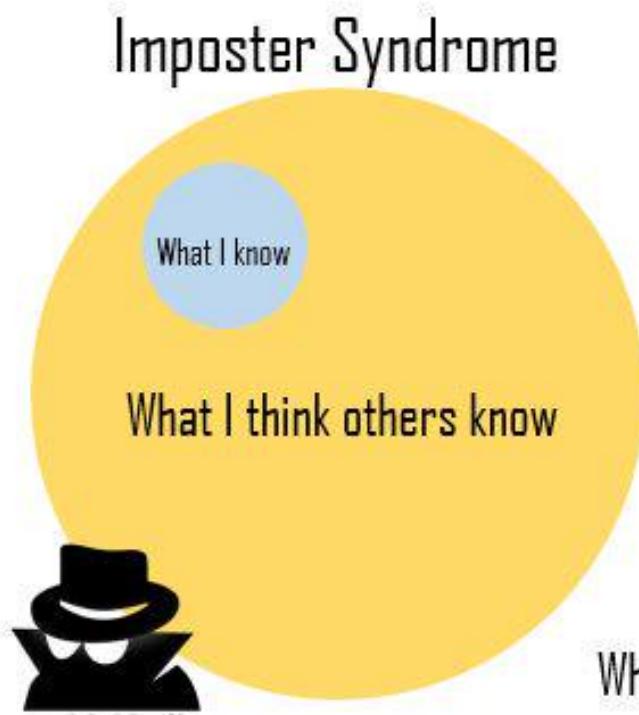
ImpostR Syndrome



David Whittaker

ImpostR
Syndrome

ImpostR Syndrome

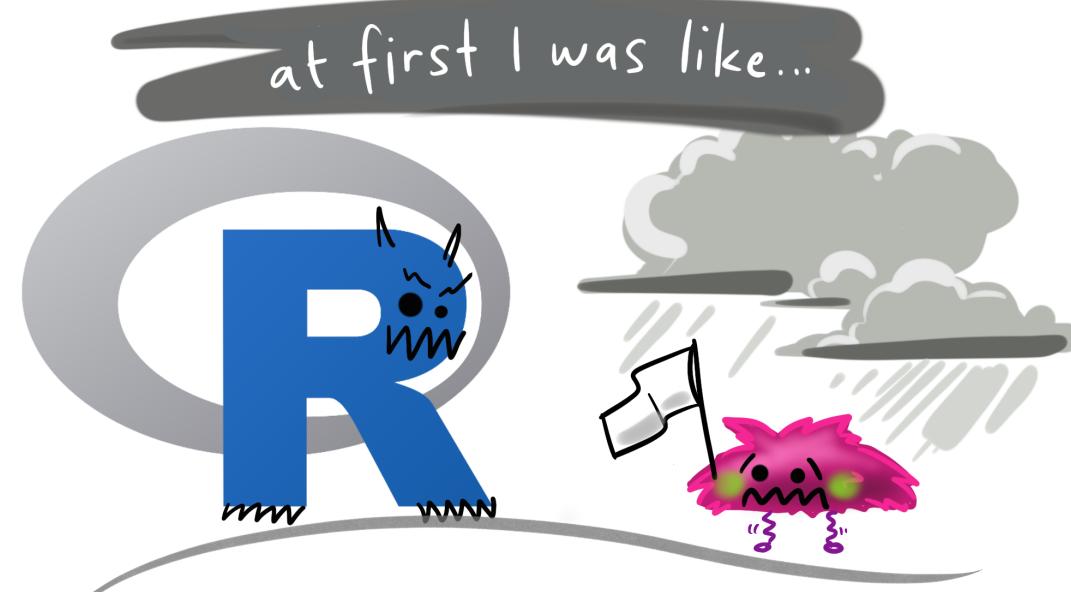


David Whittaker

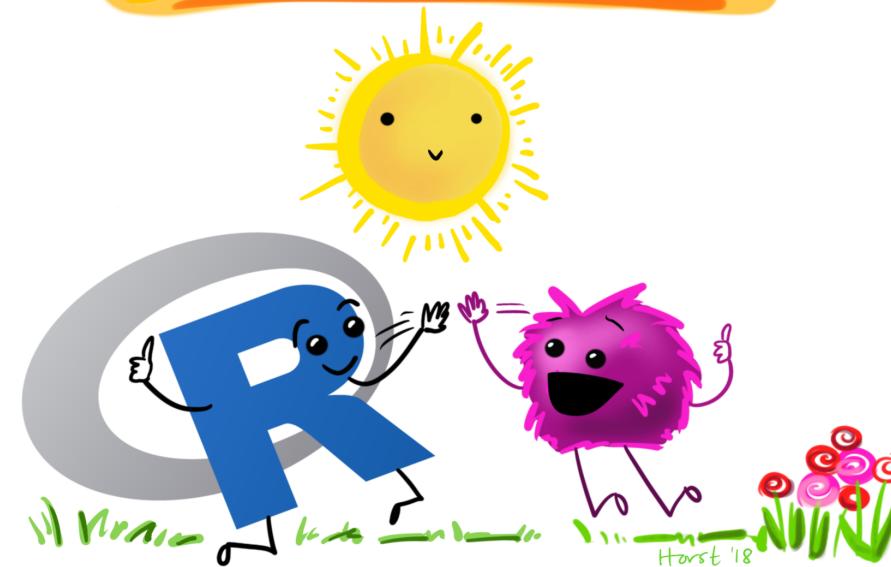
ImpostR
Syndrome

Moral of the story?

Make friends, code in groups, learn together and don't beat yourself up



...but now it's like...



About R

Code, Output, Scripts

Code

- The actual commands

Output

- The result of running code or a script

Script

- A text file full of code that you want to run
- You should always keep your code in a script

Code, Output, Scripts

Code

- The actual commands

Output

- The result of running code or a script

Script

- A text file full of code that you want to run
- You should always keep your code in a script

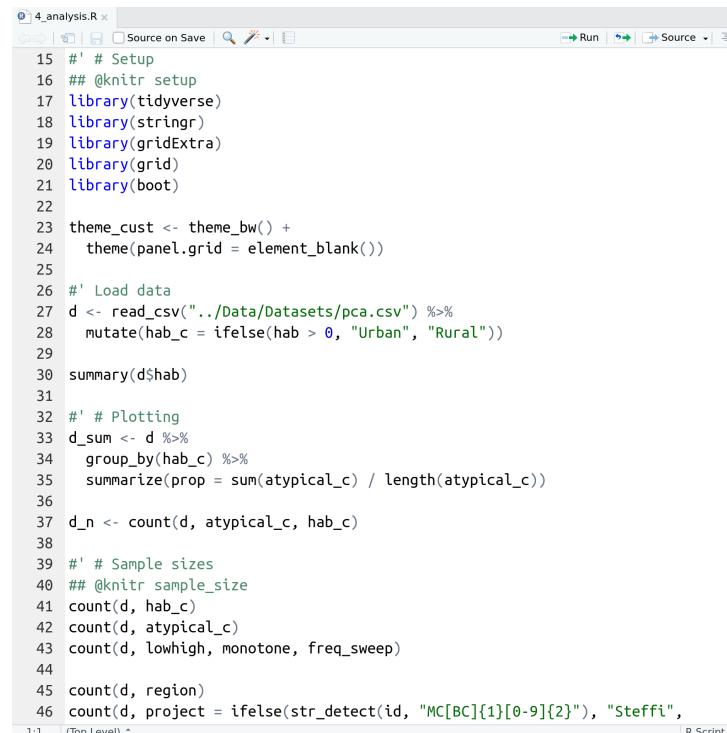
For example:

```
mean(c(1, 2, 3, 4))
```

Code

```
## [1] 2.5
```

Output



```
15 #! # Setup
16 ## @knitr setup
17 library(tidyverse)
18 library(stringr)
19 library(gridExtra)
20 library(grid)
21 library(boot)
22
23 theme_cust <- theme_bw() +
24   theme(panel.grid = element_blank())
25
26 #' Load data
27 d <- read_csv("../Data/Datasets/pca.csv") %>%
28   mutate(hab_c = ifelse(hab > 0, "Urban", "Rural"))
29
30 summary(d$hab)
31
32 #' # Plotting
33 d_sum <- d %>%
34   group_by(hab_c) %>%
35   summarize(prop = sum(atypical_c) / length(atypical_c))
36
37 d_n <- count(d, atypical_c, hab_c)
38
39 #' # Sample sizes
40 ## @knitr sample_size
41 count(d, hab_c)
42 count(d, atypical_c)
43 count(d, lowhigh, monotone, freq_sweep)
44
45 count(d, region)
46 count(d, project = ifelse(str_detect(id, "MC[BC]{1}[0-9]{2}"), "Steffi",
```

Script

RStudio vs. R



RStudio



R

- **RStudio** is not **R**
- RStudio is a User Interface or IDE (integrated development environment)
 - (i.e., Makes coding simpler)

functions() - Do things, Return things

mean(), read_csv(), ggplot(), c(), etc.

functions() - Do things, Return things

`mean()`, `read_csv()`, `ggplot()`, `c()`, etc.

- Always have ()
- Can take **arguments** (think 'options')
 - `mean(x = c(2, 10, 45))`,
 - `mean(x = c(NA, 10, 2, 65), na.rm = TRUE)`

functions() - Do things, Return things

`mean()`, `read_csv()`, `ggplot()`, `c()`, etc.

- Always have ()
- Can take **arguments** (think 'options')
 - `mean(x = c(2, 10, 45))`,
 - `mean(x = c(NA, 10, 2, 65), na.rm = TRUE)`
- Arguments defined by **name** or by **position**
- With correct position, do not need to specify by name

By name:

```
mean(x = c(1, 5, 10))
```

```
## [1] 5.333333
```

By position:

```
mean(c(1, 5, 10))
```

```
## [1] 5.333333
```

R documentation

?mean

R documentation

?mean

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

```
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Data

Generally kept in **vectors** or **data.frames**

- These are objects with names (like functions)
- We can use `<-` to assign values to objects (assignment)

Vector (1 dimension)

```
my_data <- c("a", 100, "c")
my_data
## [1] "a"   "100" "c"
```

Data frame (2 dimensions)

```
my_data <- data.frame(site = c("s1", "s2", "s3"),
                      count = c(101, 102, 103),
                      treatment = c("a", "b", "c"))
my_data
```

```
##   site count treatment
## 1   s1    101         a
## 2   s2    102         b
## 3   s3    103         c
```

rows X
columns

Your first *real* code!

First Code

```
# First load the packages  
library(palmerpenguins)  
library(ggplot2)  
  
# Now create the figure  
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```

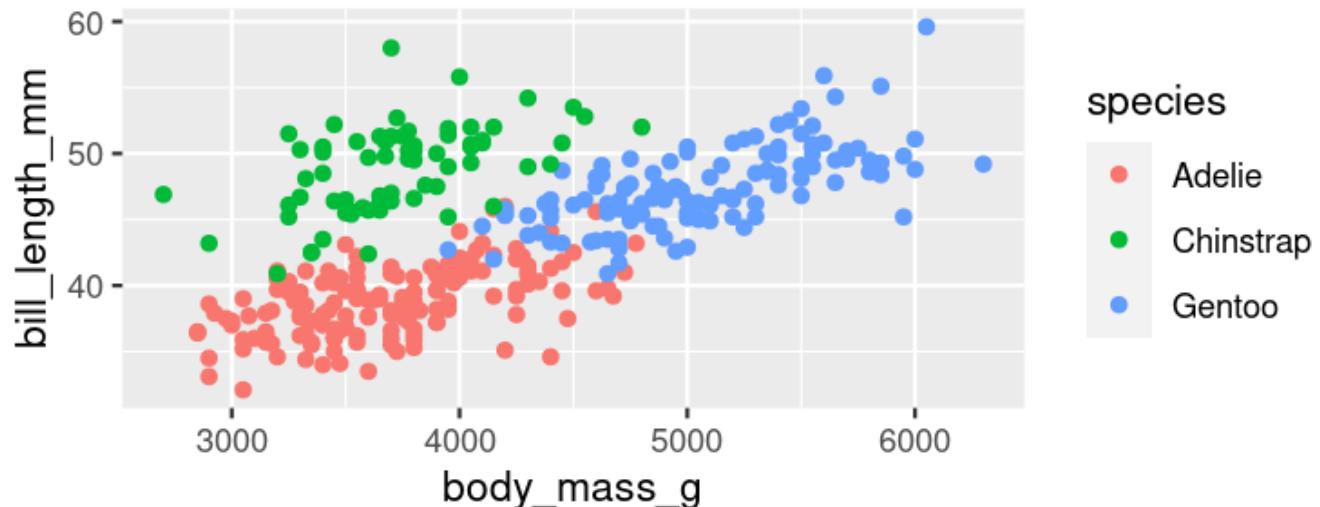
- Copy/paste or type this into the script window in RStudio
 - You may have to go to File > New File > R Script
- Click anywhere on the first line of code
- Use the 'Run' button to run this code, **or** use the short-cut **Ctrl-Enter**
 - Repeat until all the code has run

First Code

```
# First load the packages
library(palmerpenguins)
library(ggplot2)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
  geom_point()
```

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



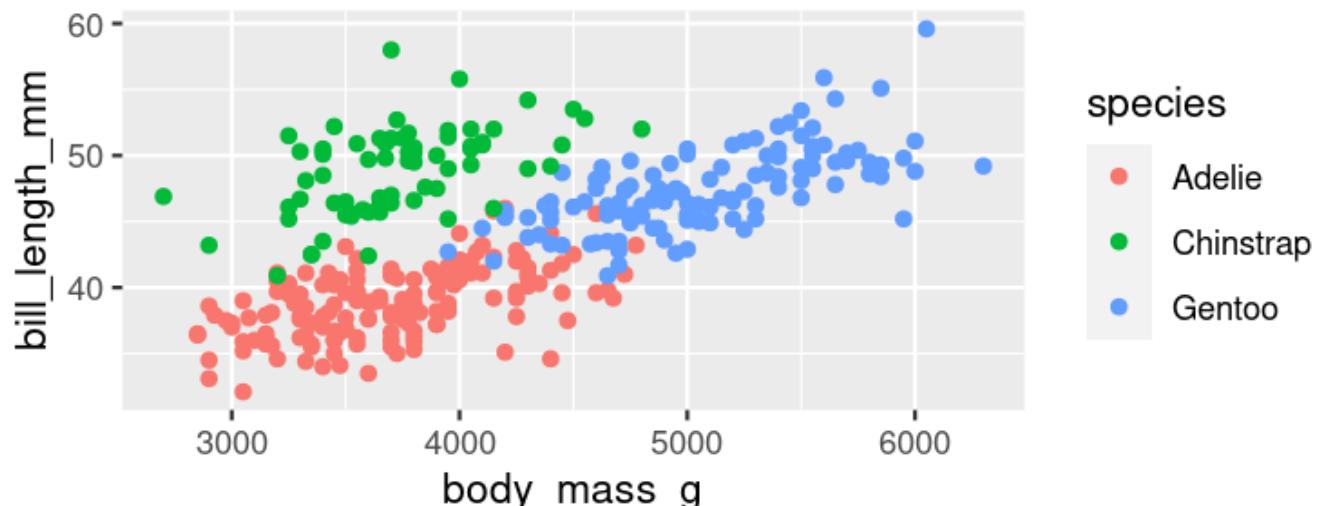
First Code

```
# First load the packages  
library(palmerpenguins)  
library(ggplot2)
```

Packages
ggplot2 and **palmerpenguins**

```
# Now create the figure  
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```

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



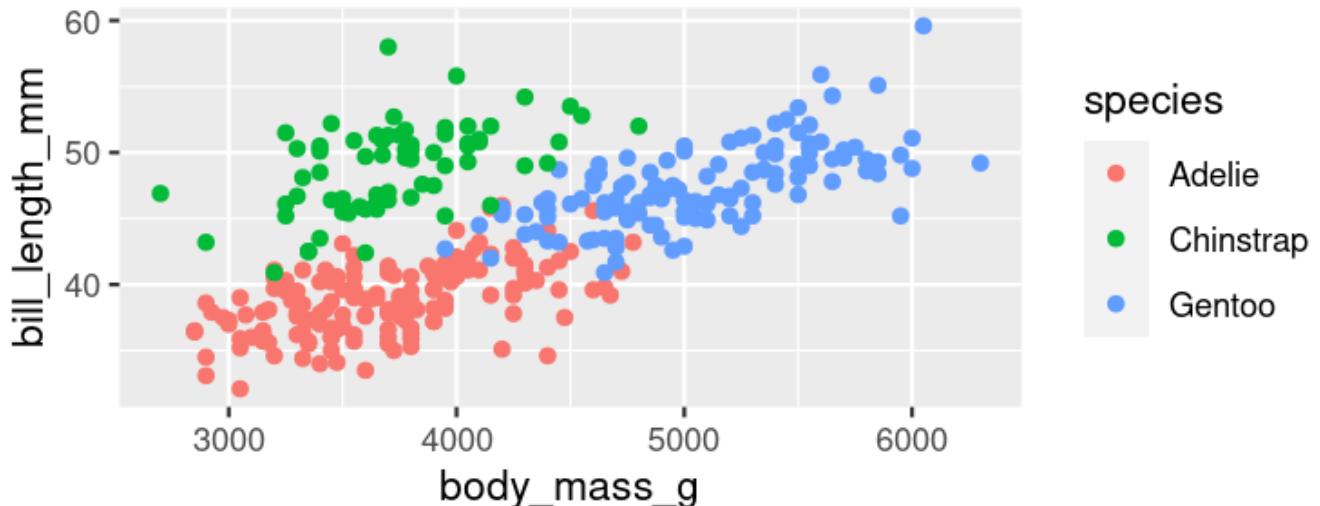
First Code

```
# First load the packages  
library(palmerpenguins)  
library(ggplot2)
```

```
# Now create the figure  
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```

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

Functions:
library(), **ggplot()**
aes(), and **geom_point()**



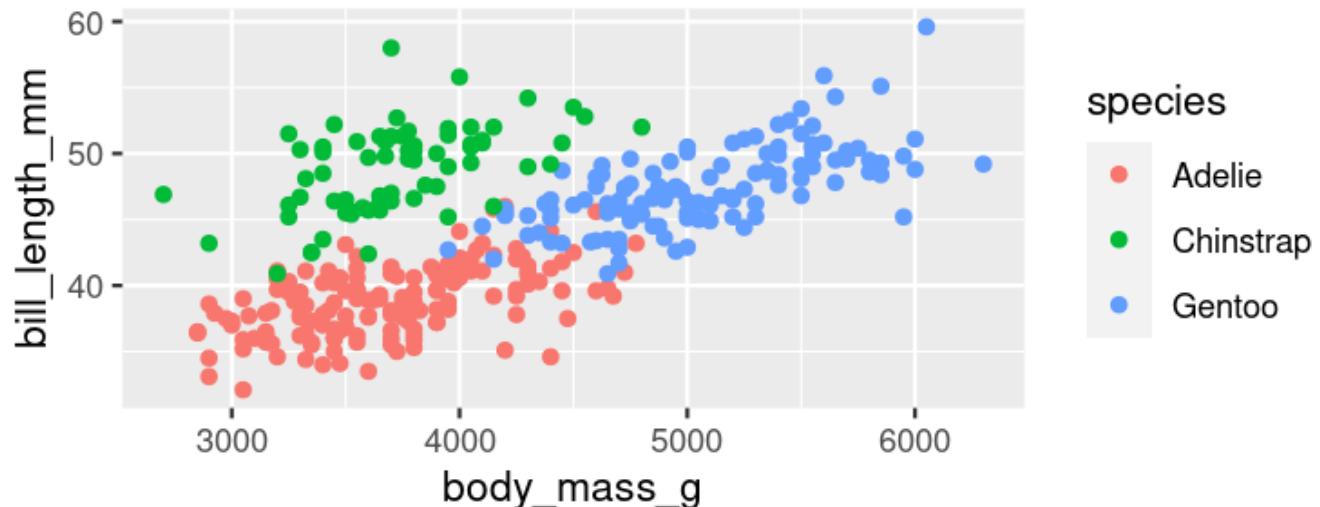
First Code

```
# First load the packages  
library(palmerpenguins)  
library(ggplot2)
```

```
# Now create the figure  
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```

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

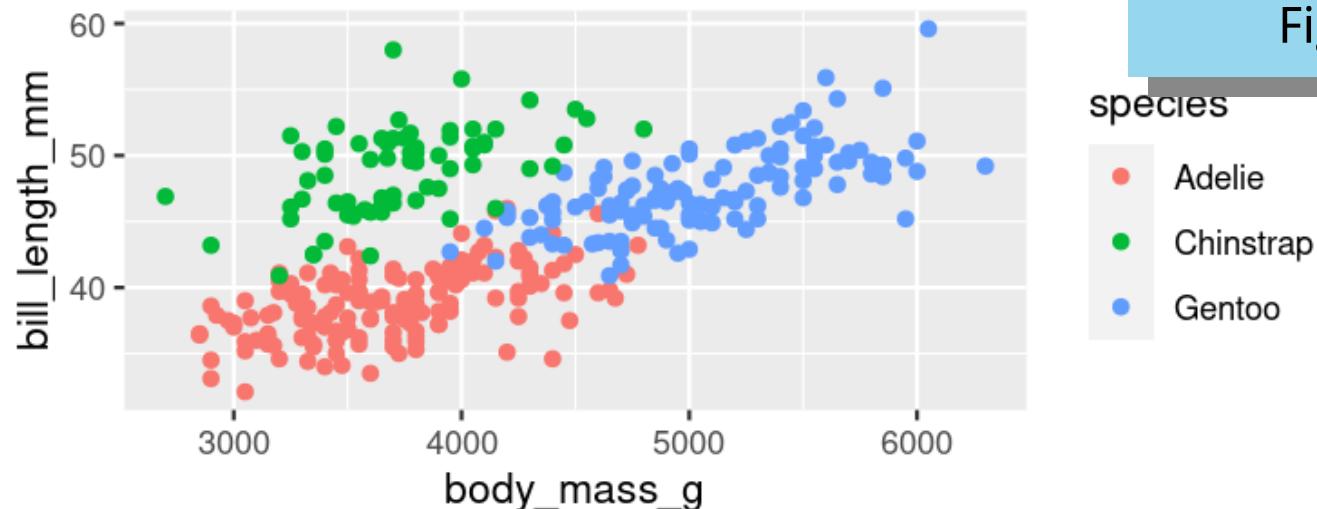
+
(Specific to **ggplot2**)



First Code

```
# First load the packages  
library(palmerpenguins)  
library(ggplot2)  
  
# Now create the figure  
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```

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



Figure!

species

- Adelie
- Chinstrap
- Gentoo

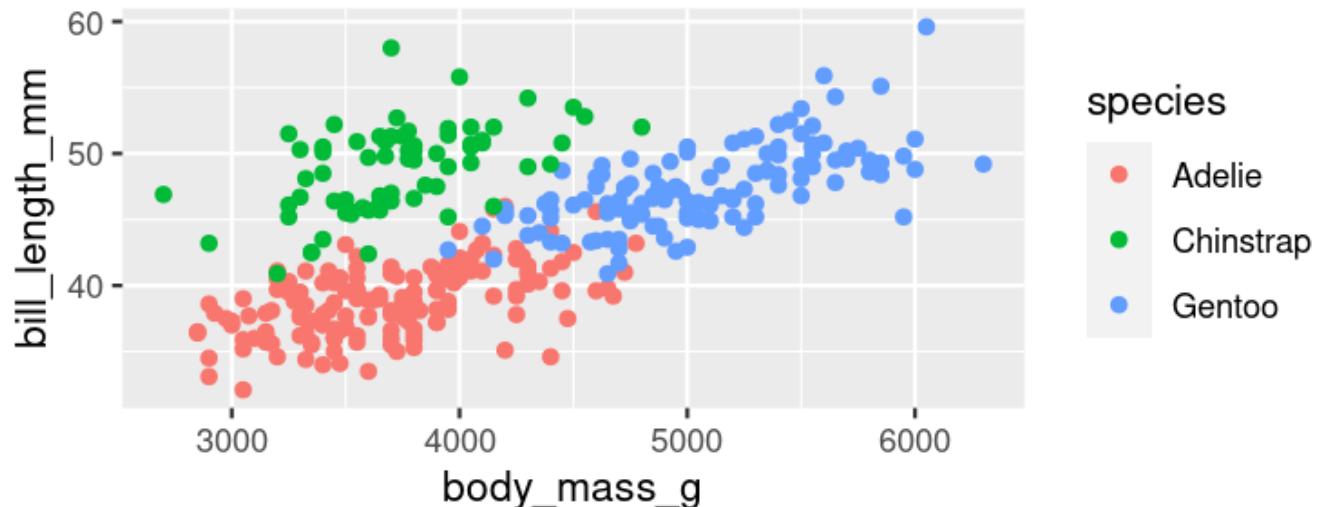
First Code

```
# First load the packages
library(palmerpenguins)
library(ggplot2)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
  geom_point()

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

Warning



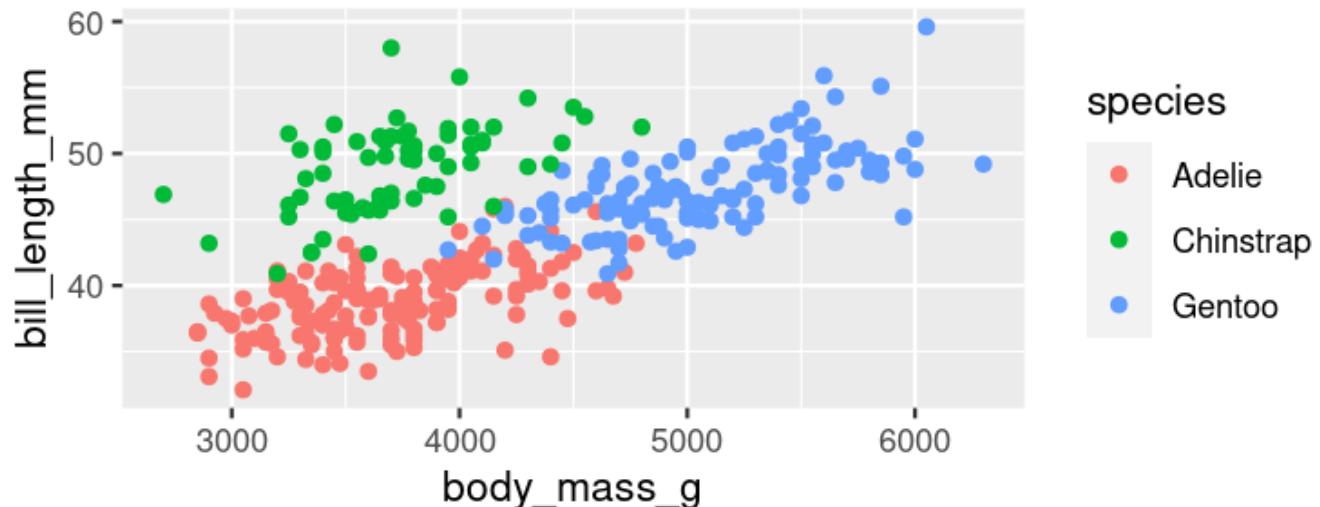
First Code

```
# First load the packages  
library(palmerpenguins)  
library(ggplot2)
```

Comments

```
# Now create the figure  
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```

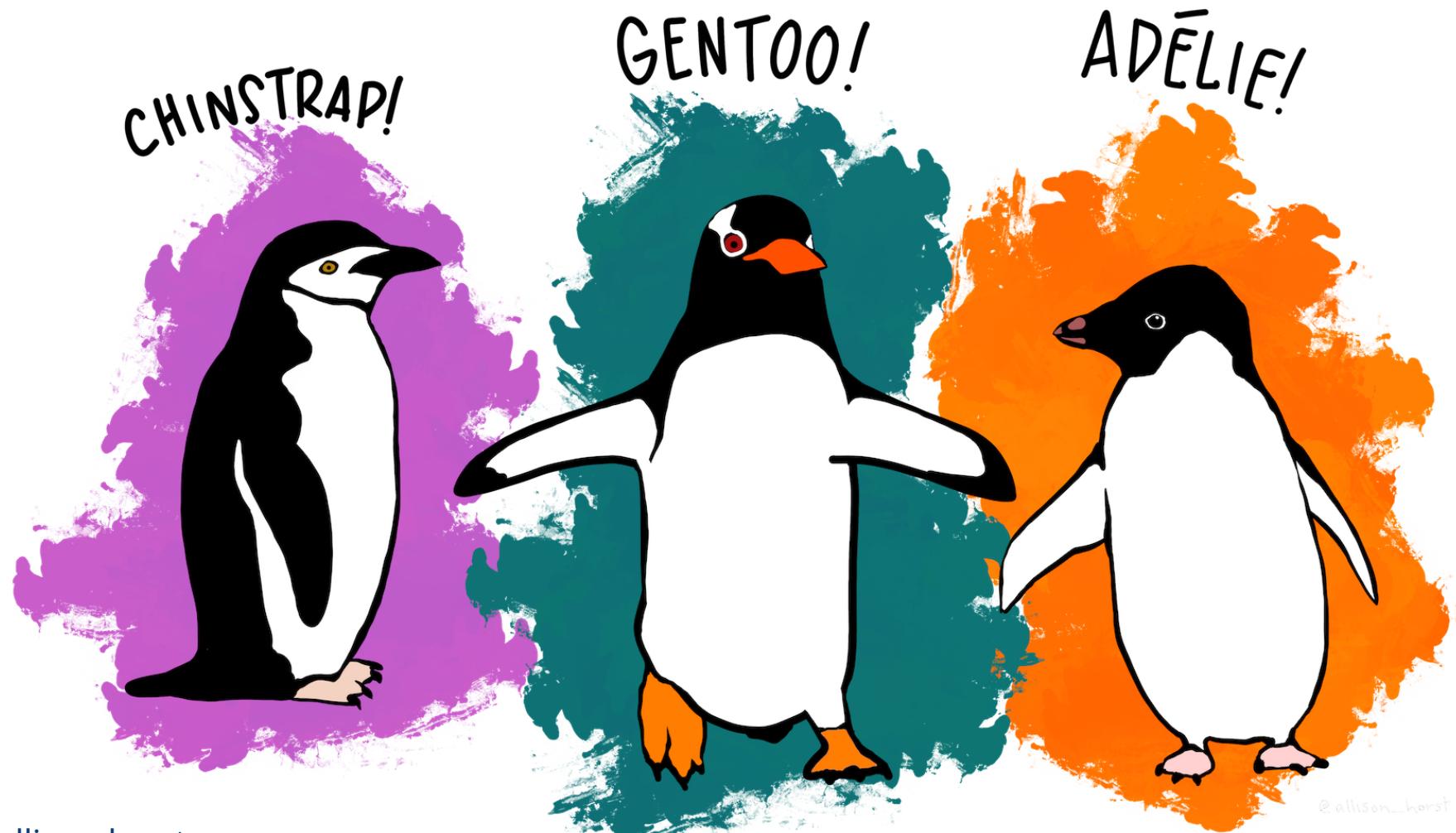
```
## Warning: Removed 2 rows containing missing values (geom_point).
```



Now you know R!

Let's get started

Our data set: Palmer Penguins!



Artwork by [@allison_horst](#)

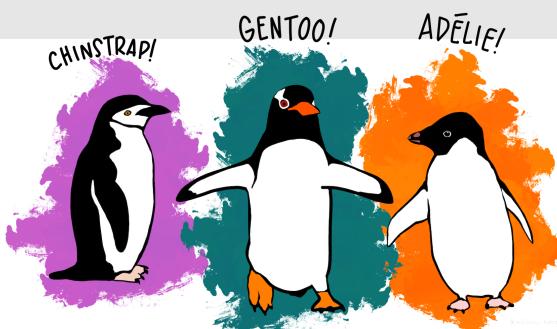
@allison_horst



Our data set: Palmer Penguins!

```
library(palmerpenguins)  
penguins
```

```
## # A tibble: 344 x 8  
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g 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 female 2007  
## 3 Adelie  Torgersen     40.3           18              195       3250 female 2007  
## 4 Adelie  Torgersen     NA             NA              NA        NA <NA>  2007  
## 5 Adelie  Torgersen     36.7           19.3            193       3450 female 2007  
## 6 Adelie  Torgersen     39.3           20.6            190       3650 male  2007  
## 7 Adelie  Torgersen     38.9           17.8            181       3625 female 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
```



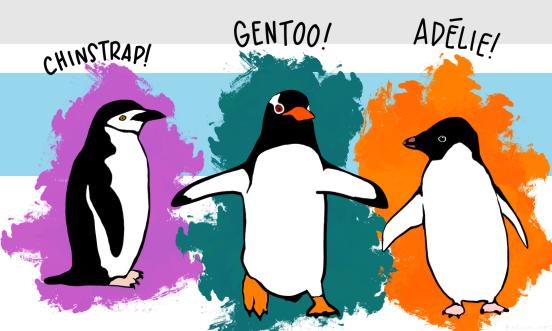


Our data set: Palmer Penguins!

```
library(palmerpenguins)  
penguins
```

```
## # A tibble: 344 x 8  
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g 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 female 2007  
## 3 Adelie  Torgersen     40.3           18              195       3250 female 2007  
## 4 Adelie  Torgersen     NA             NA              NA         NA <NA>  2007  
## 5 Adelie  Torgersen     36.7           19.3            193       3450 female 2007  
## 6 Adelie  Torgersen     39.3           20.6            190       3650 male  2007  
## 7 Adelie  Torgersen     38.9           17.8            181       3625 female 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
```

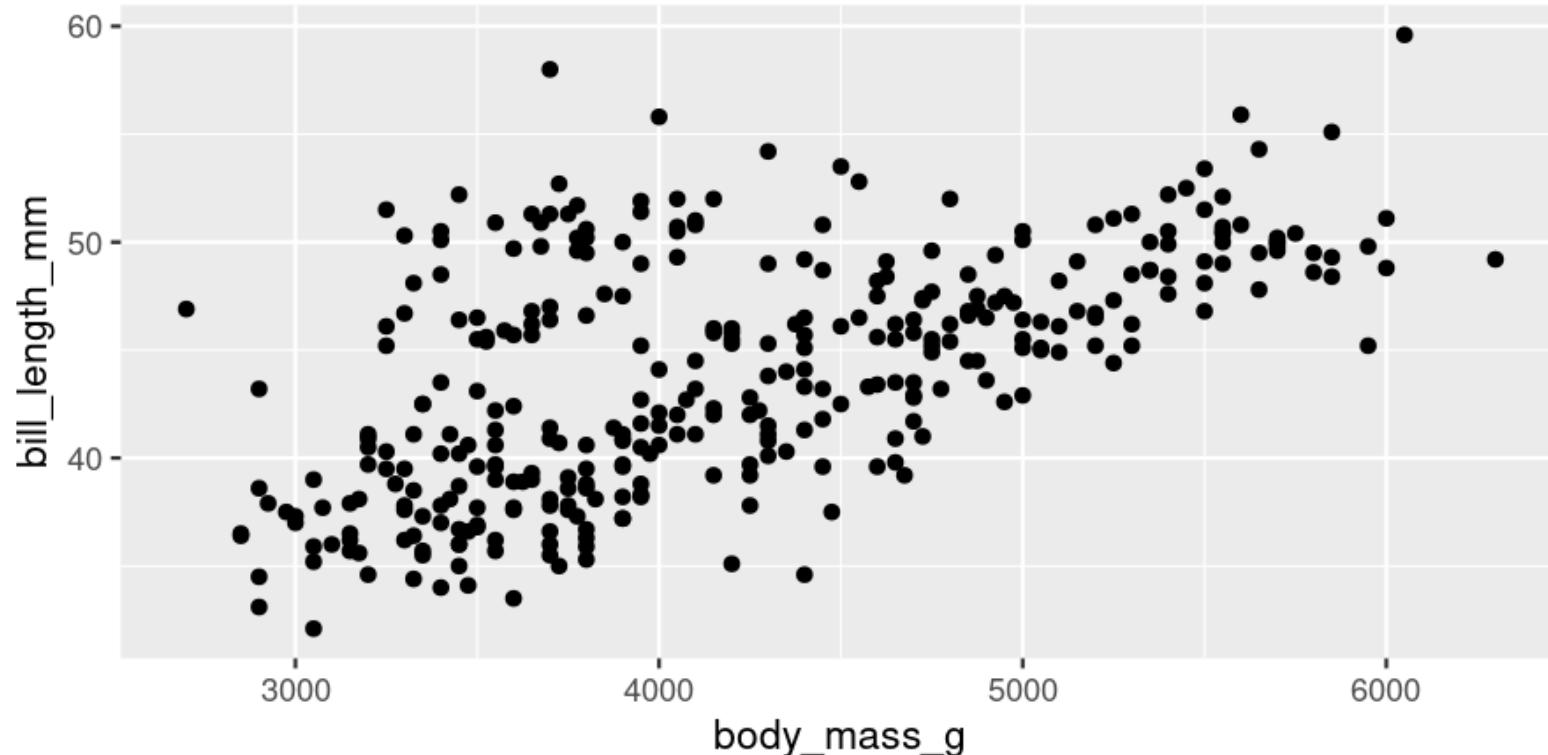
Your turn! Run this code and look at the output in the console



A basic plot

```
library(palmerpenguins)
library(ggplot2)

ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +
  geom_point()
```



Break it down

```
library(palmerpenguins)
library(ggplot2)

ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +
  geom_point()
```

library(palmerpenguins)

- Load the **palmerpenguins** package so we have access to **penguins** data

Break it down

```
library(palmerpenguins)
library(ggplot2)

ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +
  geom_point()
```

library(ggplot2)

- Load the **ggplot2** package (which gives us access to the **ggplot()** function among others)

Break it down

```
library(palmerpenguins)
library(ggplot2)

ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +
  geom_point()
```

ggplot()

- Set the attributes of your plot
- **data** = Dataset
- **aes** = Aesthetics (how the data are used)
- Think of this as your plot defaults

Break it down

```
library(palmerpenguins)
library(ggplot2)

ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +
    geom_point()
```

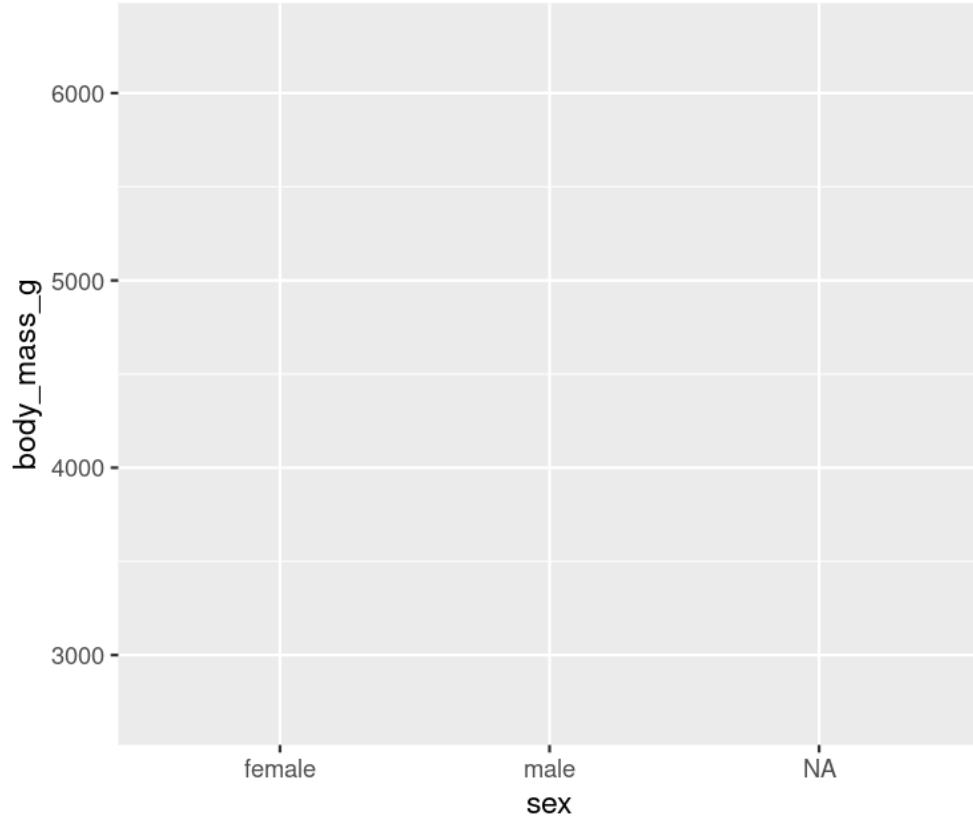
geom_point()

- Choose a **geom** function to display the data
- Always *added* to a **ggplot()** call with **+**

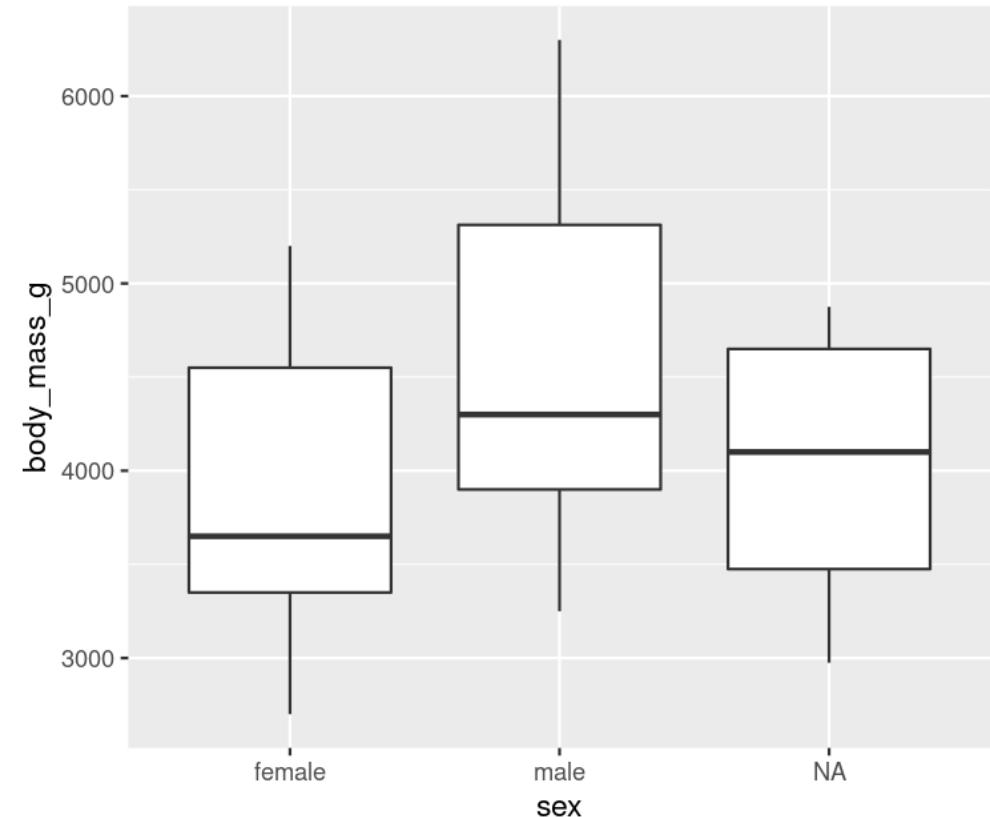
ggplots are essentially layered objects, starting with a call to **ggplot()**

Plots are layered

```
ggplot(data = penguins, aes(x = sex, y = body_mass_g))
```

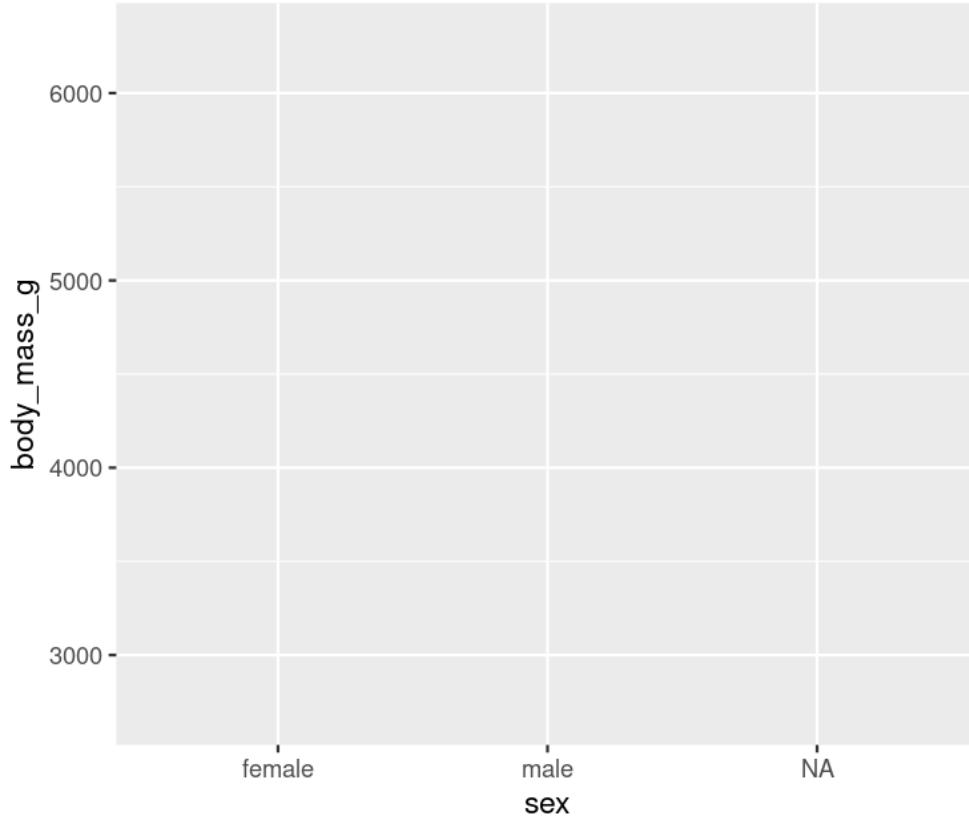


```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_boxplot()
```

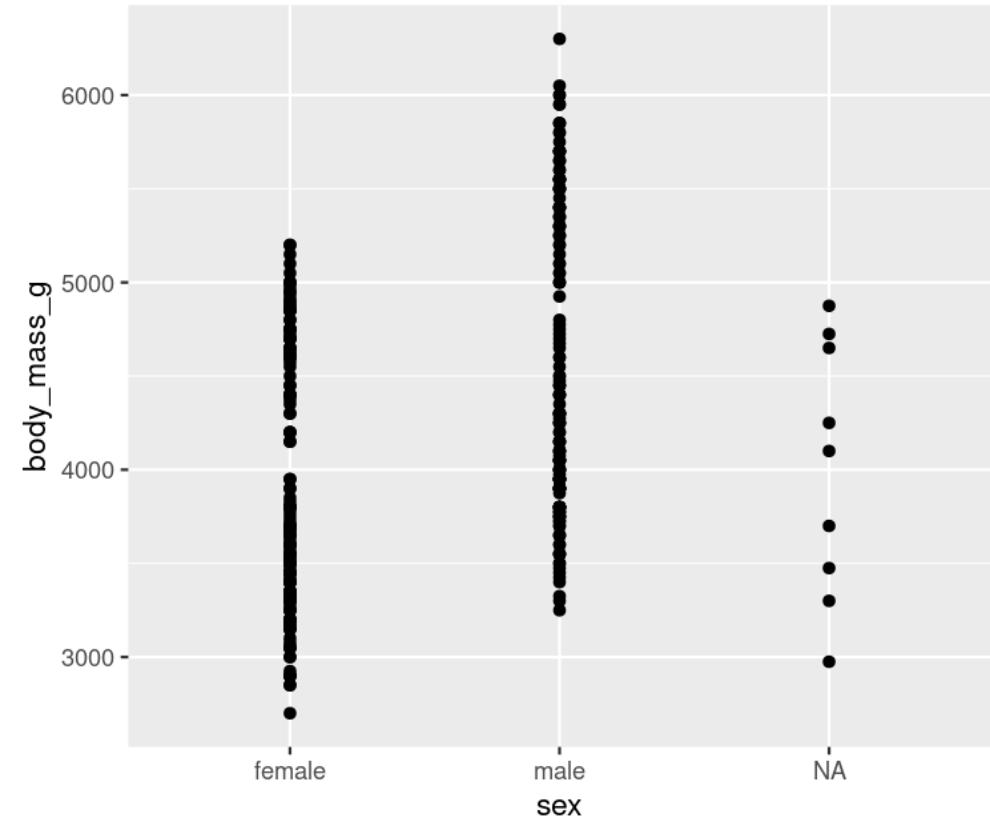


Plots are layered

```
ggplot(data = penguins, aes(x = sex, y = body_mass_g))
```

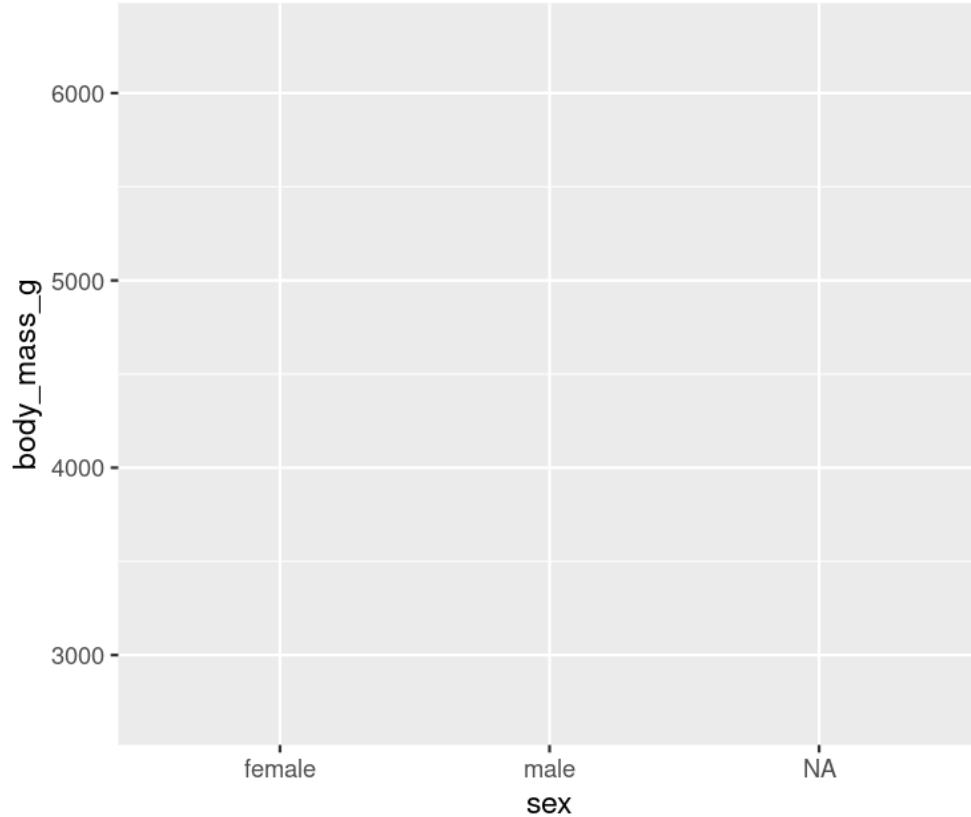


```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_point()
```

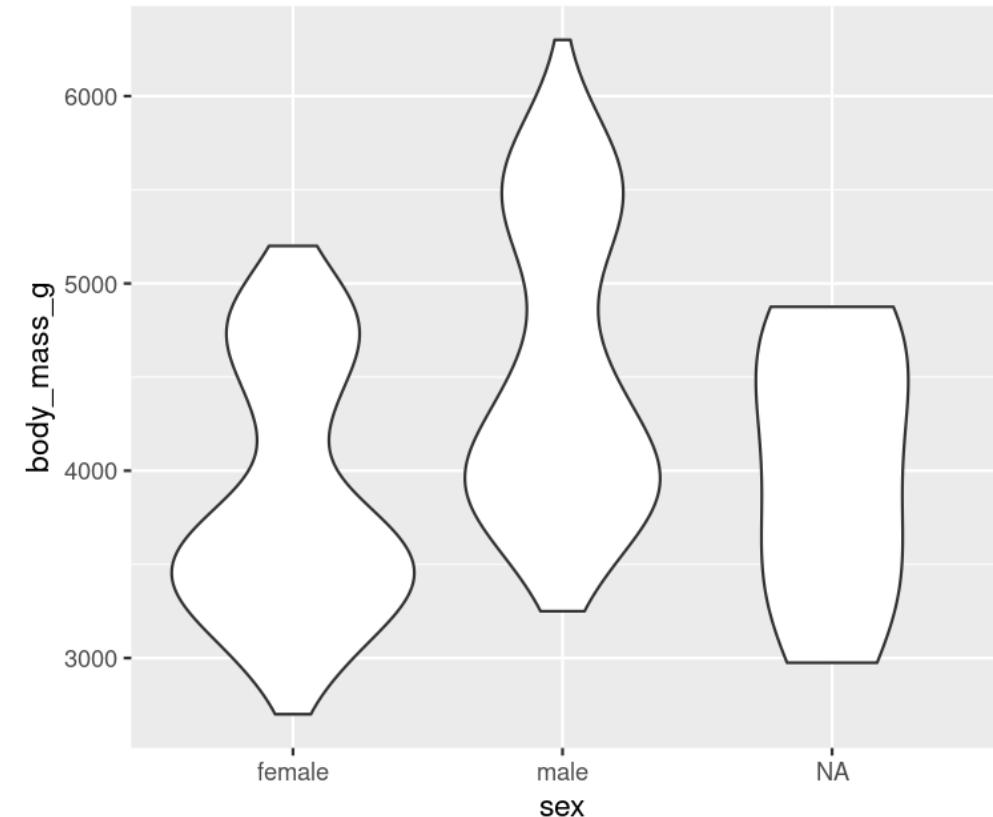


Plots are layered

```
ggplot(data = penguins, aes(x = sex, y = body_mass_g))
```



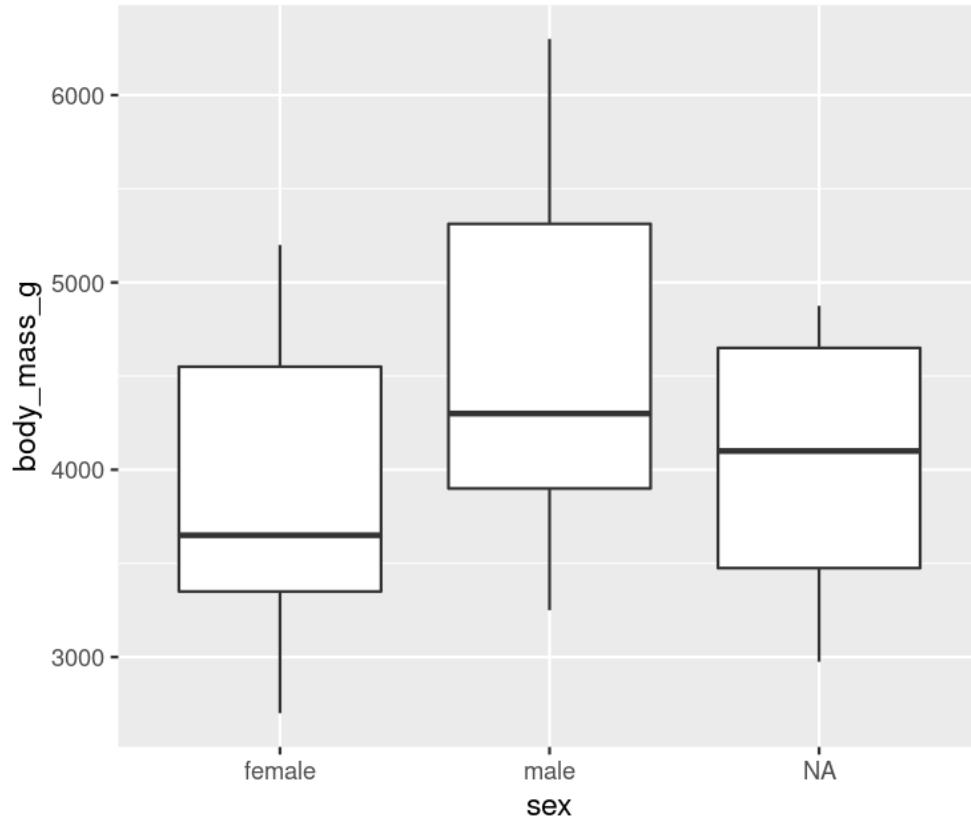
```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_violin()
```



Plots are layered

You can add multiple layers

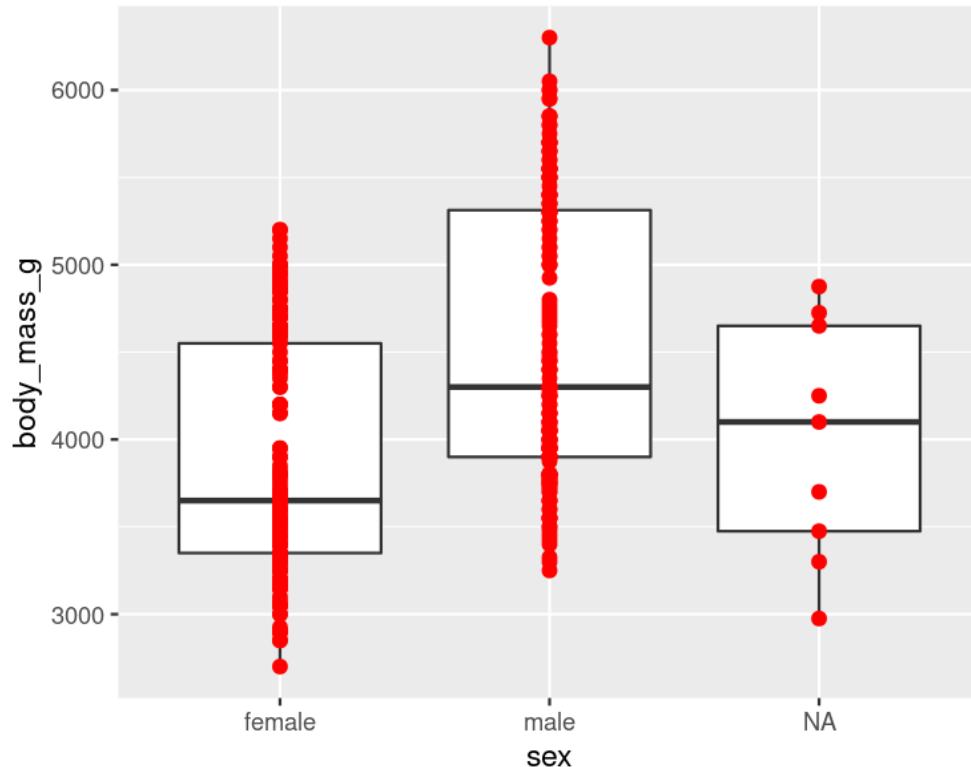
```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_boxplot()
```



Plots are layered

You can add multiple layers

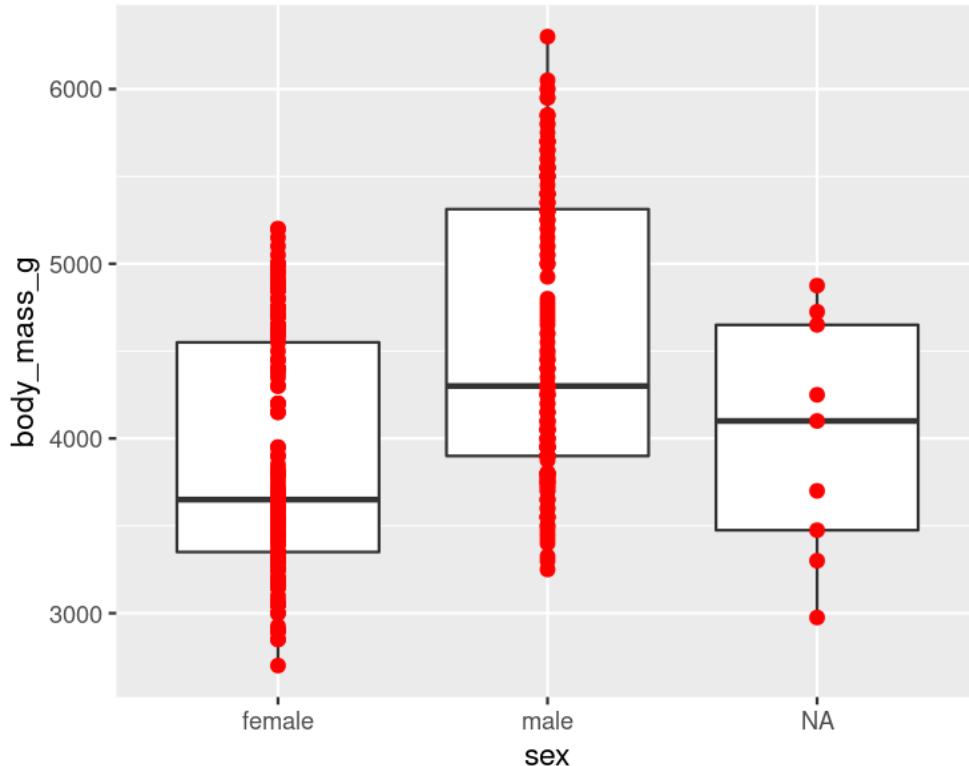
```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_boxplot() +  
  geom_point(size = 2, colour = "red")
```



Plots are layered

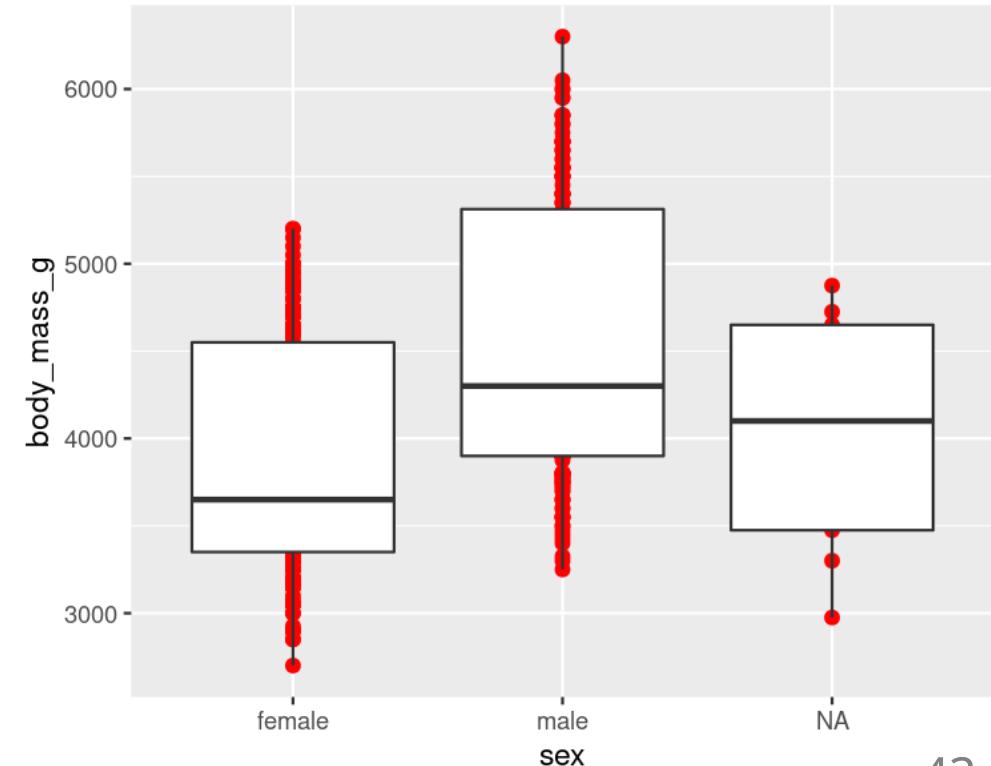
You can add multiple layers

```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_boxplot() +  
  geom_point(size = 2, colour = "red")
```



Order matters

```
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +  
  geom_point(size = 2, colour = "red") +  
  geom_boxplot()
```



Plots are objects

Any ggplot can be saved as an object

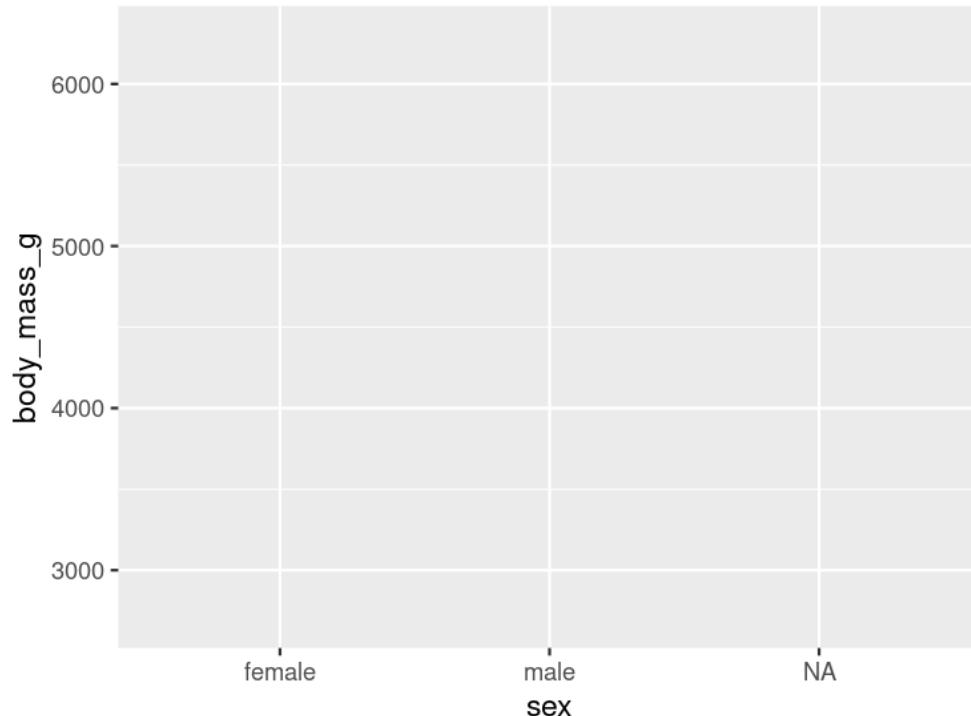
```
g <- ggplot(data = penguins, aes(x = sex, y = body_mass_g))
```

Plots are objects

Any ggplot can be saved as an object

```
g <- ggplot(data = penguins, aes(x = sex, y = body_mass_g))
```

```
g
```

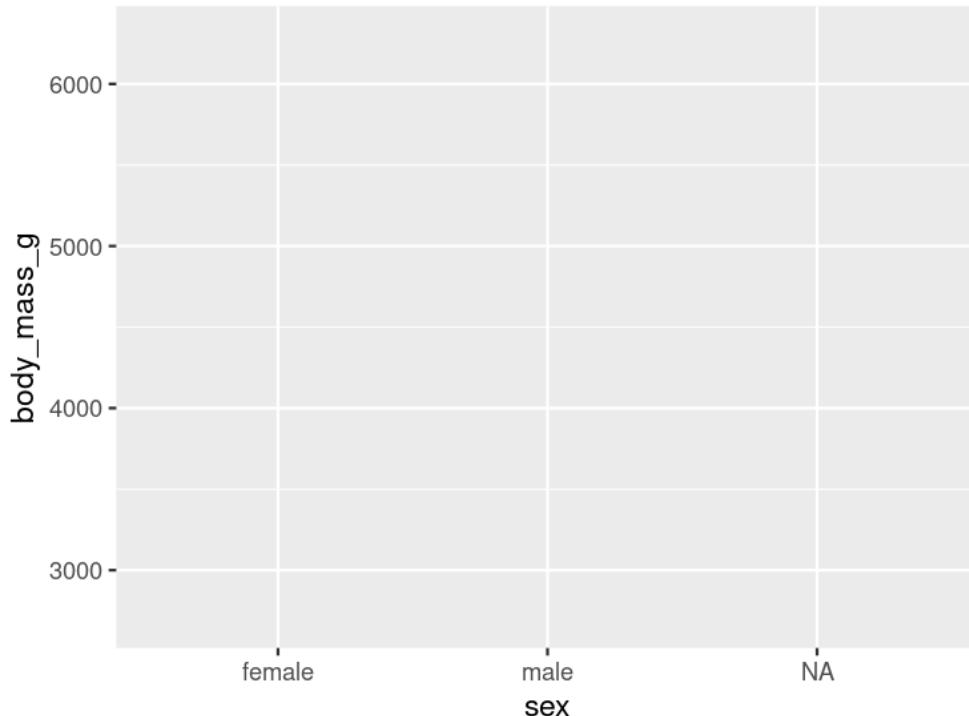


Plots are objects

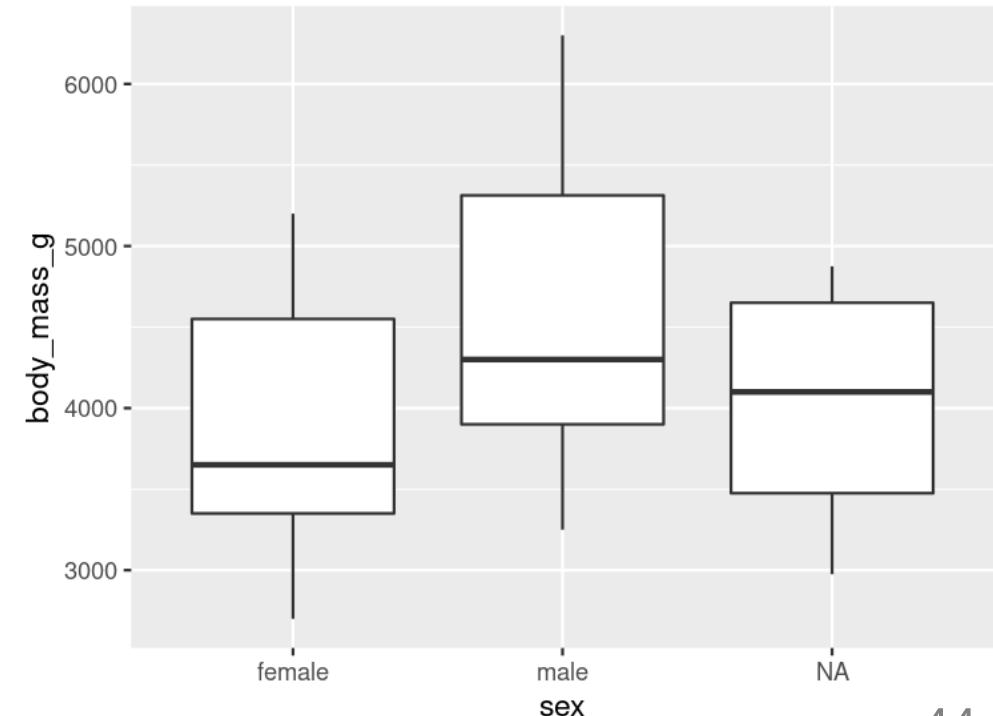
Any ggplot can be saved as an object

```
g <- ggplot(data = penguins, aes(x = sex, y = body_mass_g))
```

```
g
```



```
g + geom_boxplot()
```

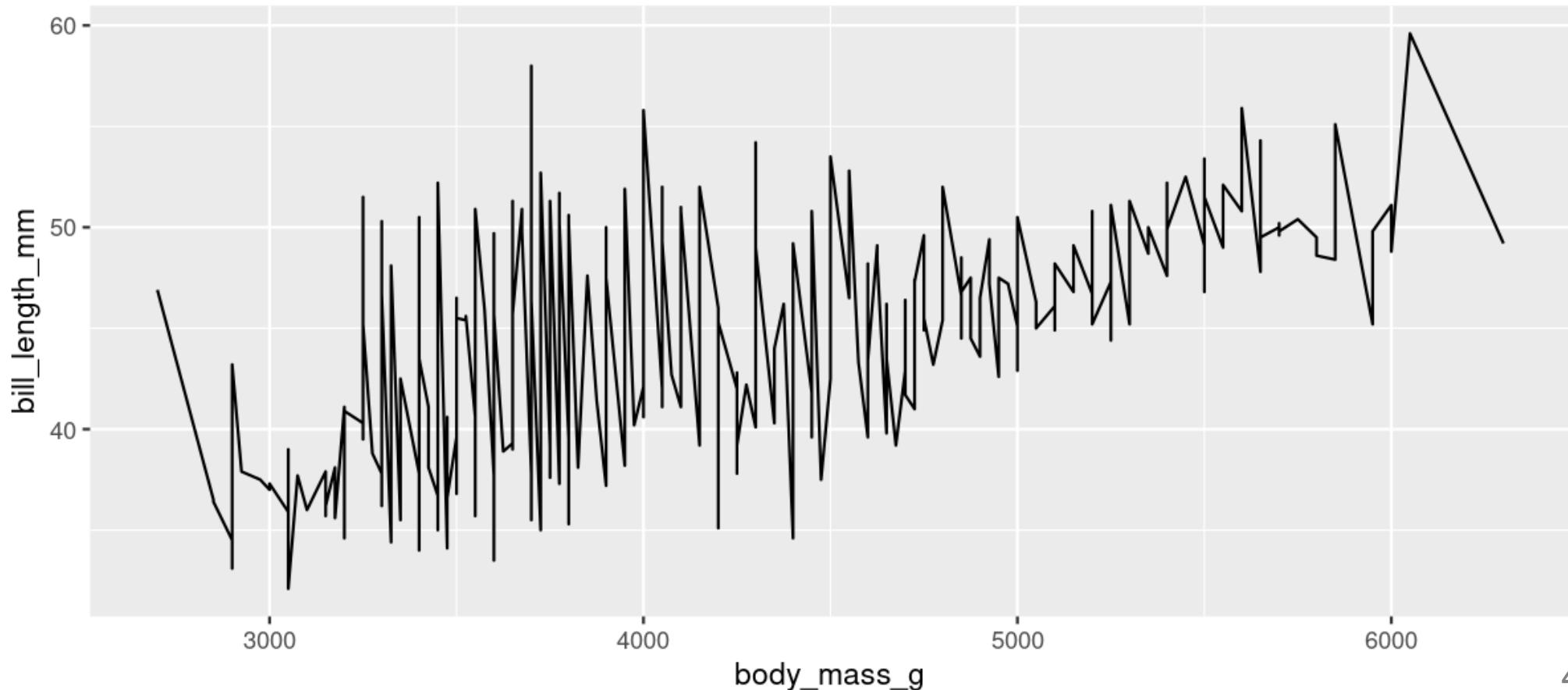


More Geoms

(Plot types)

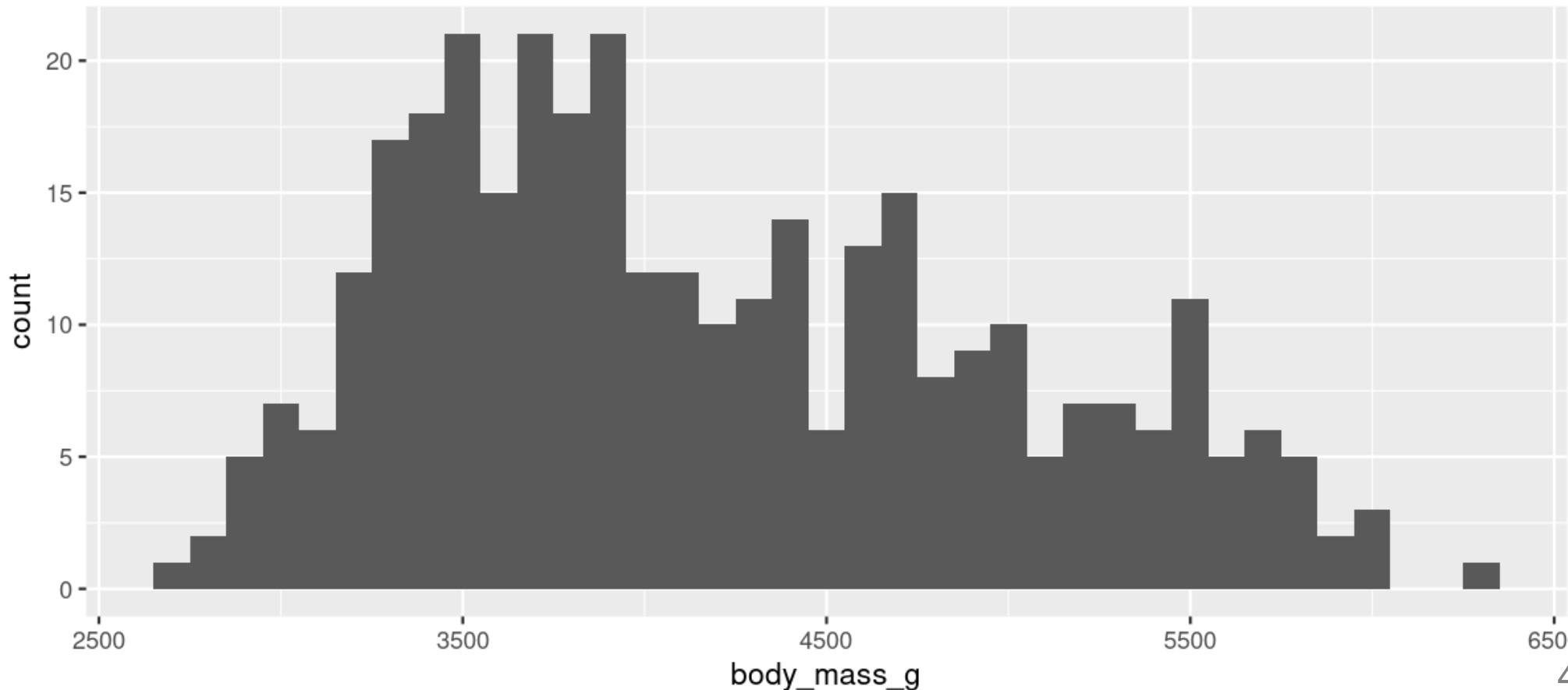
Geoms: Lines

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +  
  geom_line()
```



Geoms: Histogram

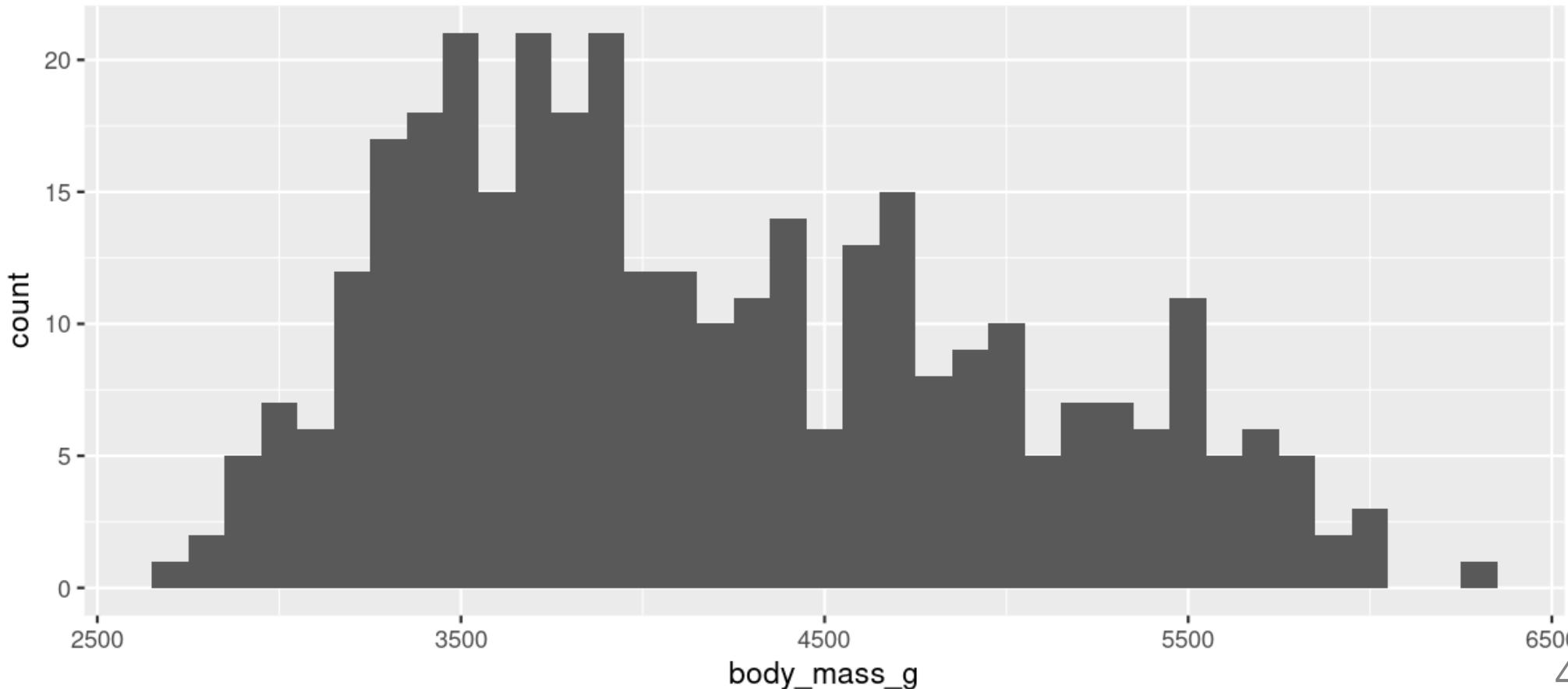
```
ggplot(data = penguins, aes(x = body_mass_g)) +  
  geom_histogram(binwidth = 100)
```



Geoms: Histogram

```
ggplot(data = penguins, aes(x = body_mass_g)) +  
  geom_histogram(binwidth = 100)
```

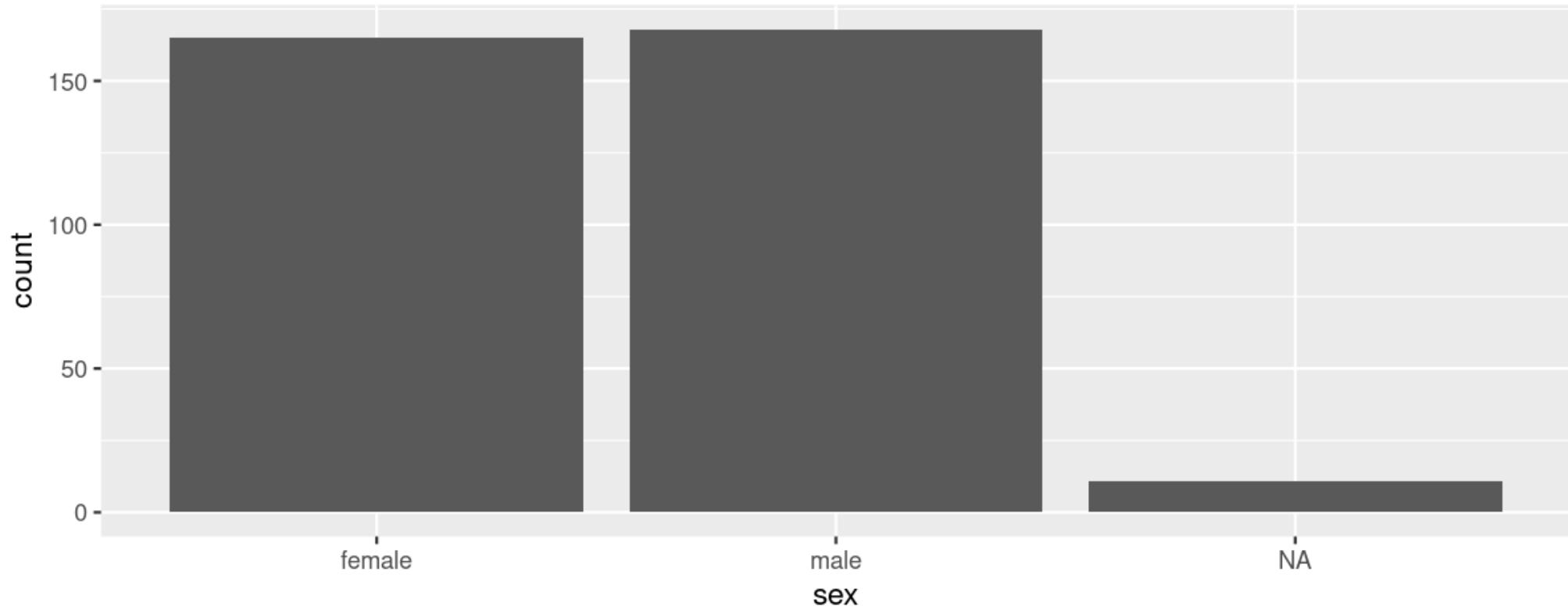
Note: We only need 1 aesthetic here (x)



Geoms: Barplots

Let **ggplot** count your data

```
ggplot(data = penguins, aes(x = sex)) +  
  geom_bar()
```

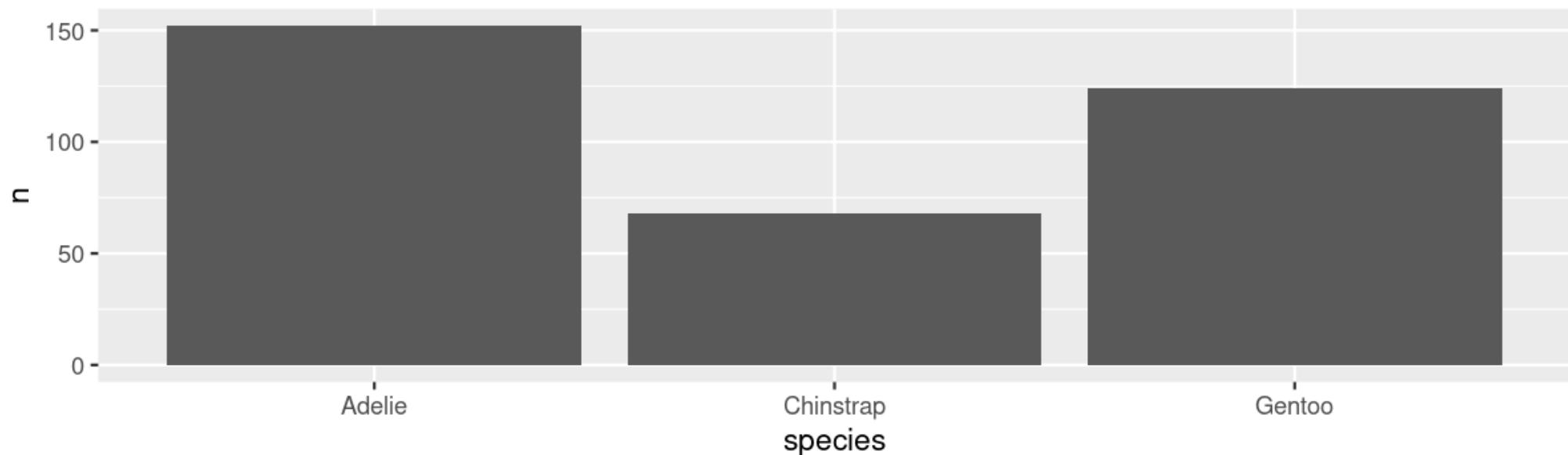


Geoms: Barplots

You can also provide the counts

```
# Create our own data frame
species <- data.frame(species = c("Adelie", "Chinstrap", "Gentoo"),
                      n = c(152, 68, 124))

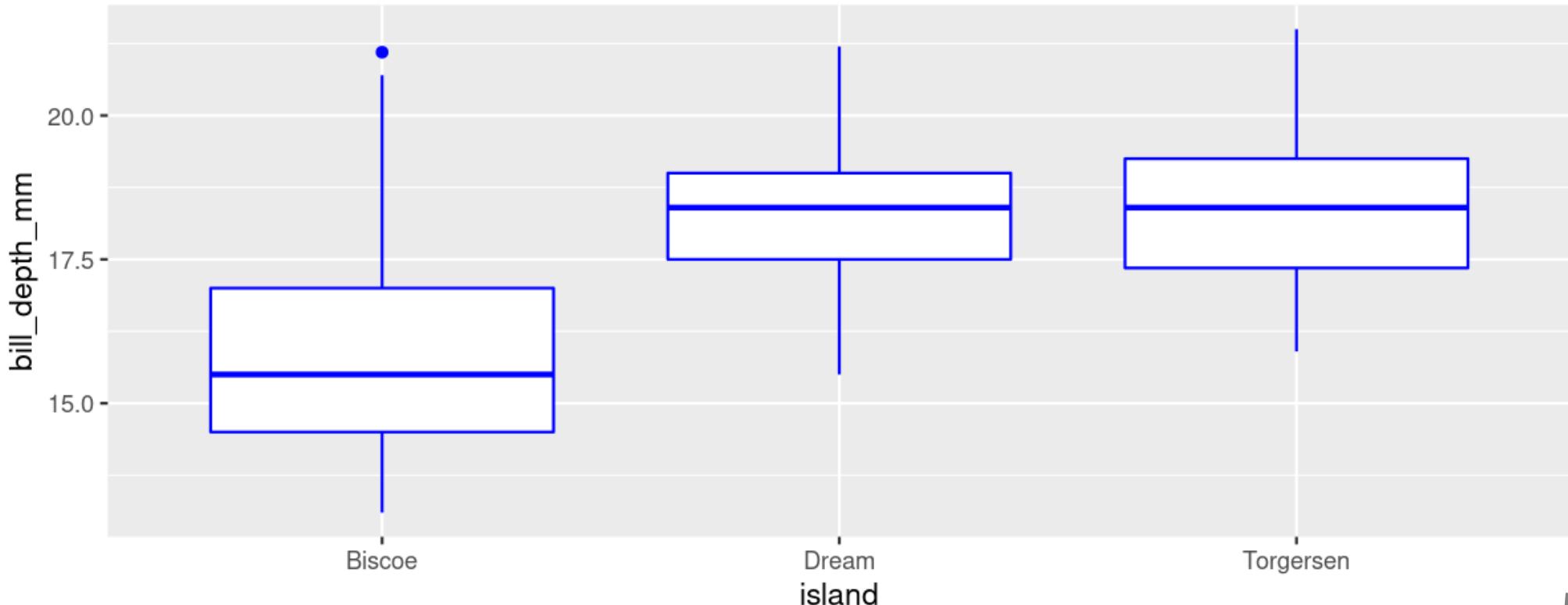
ggplot(data = species, aes(x = species, y = n)) +
  geom_bar(stat = "identity")
```



Your Turn: Create this plot

```
library(ggplot2)

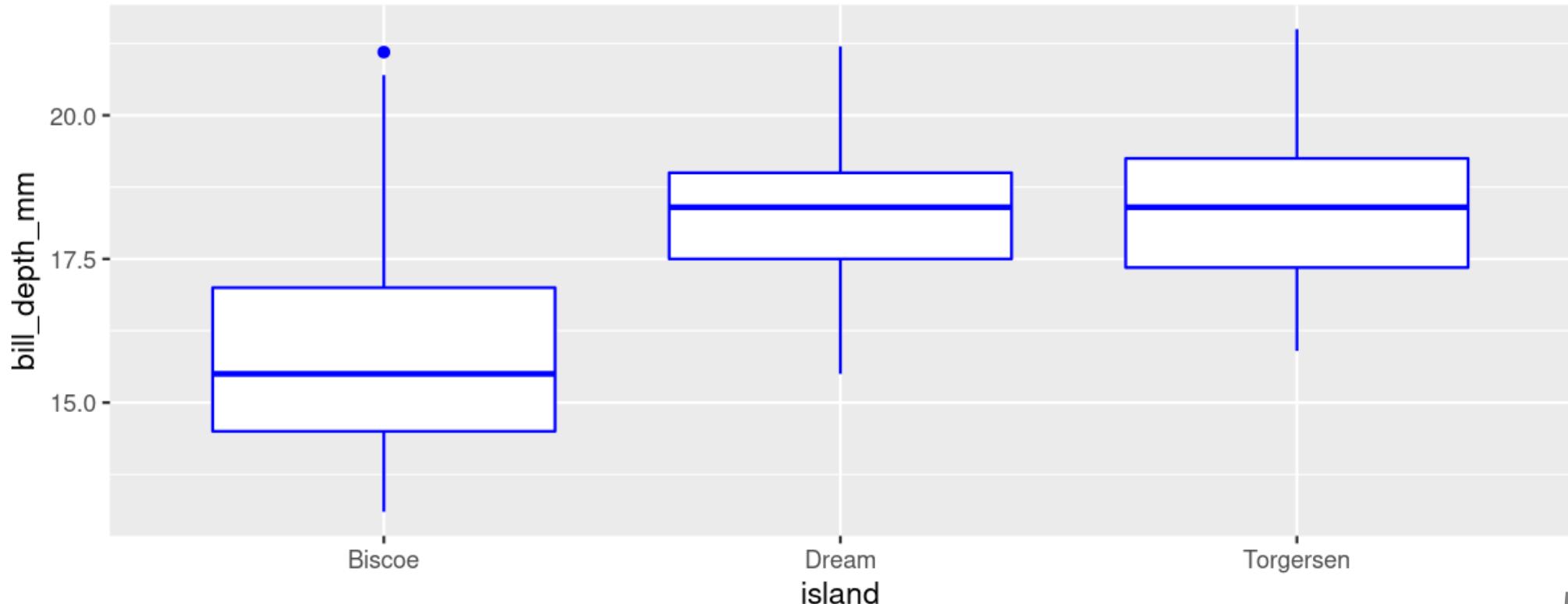
ggplot(data = [REDACTED], aes(x = [REDACTED], y = [REDACTED])) +
  geom_[REDACTED]([REDACTED])
```



Your Turn: Create this plot

```
library(ggplot2)

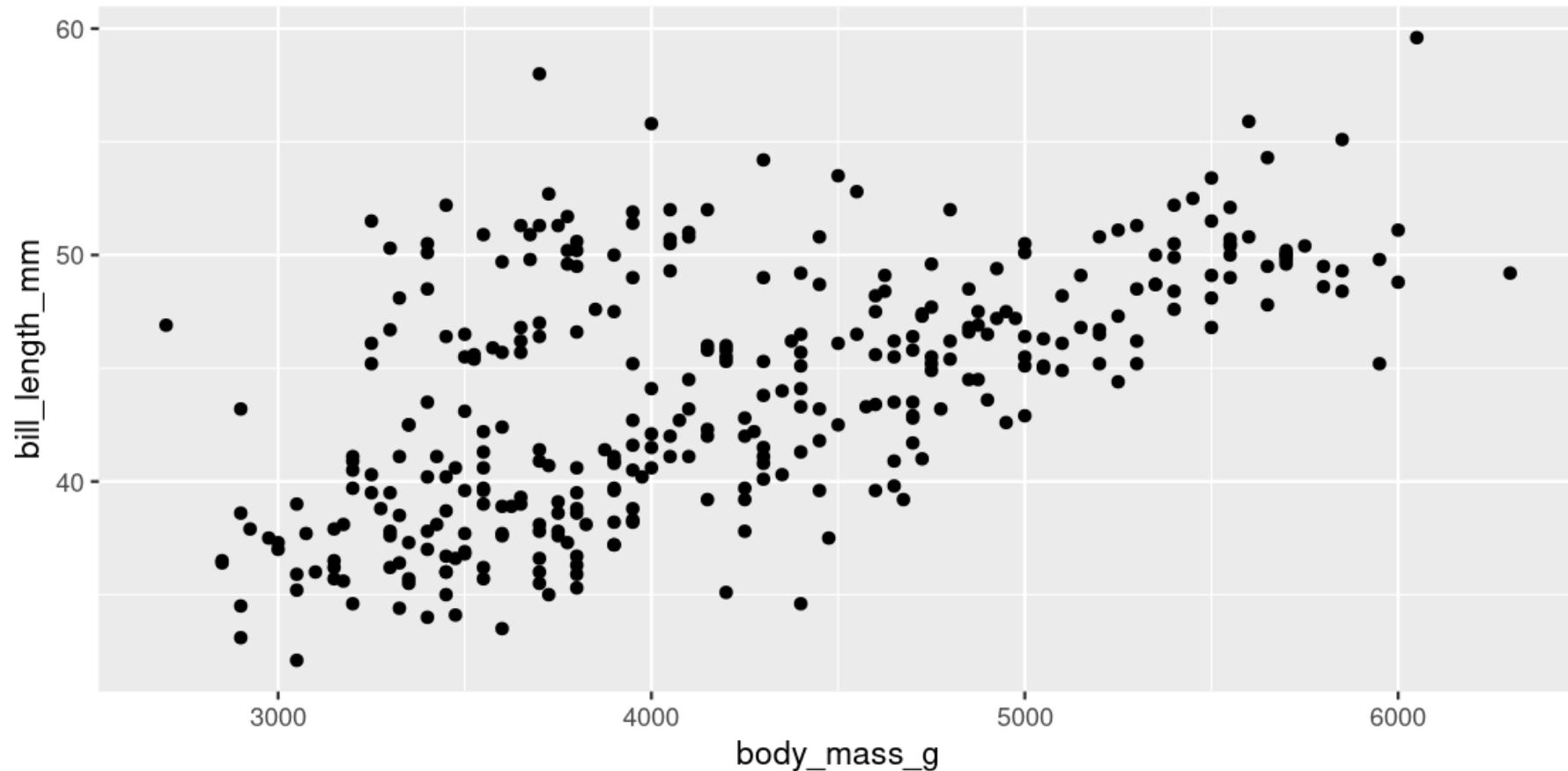
ggplot(data = penguins, aes(x = island, y = bill_depth_mm)) +
  geom_boxplot(colour = "blue")
```



Showing data by group

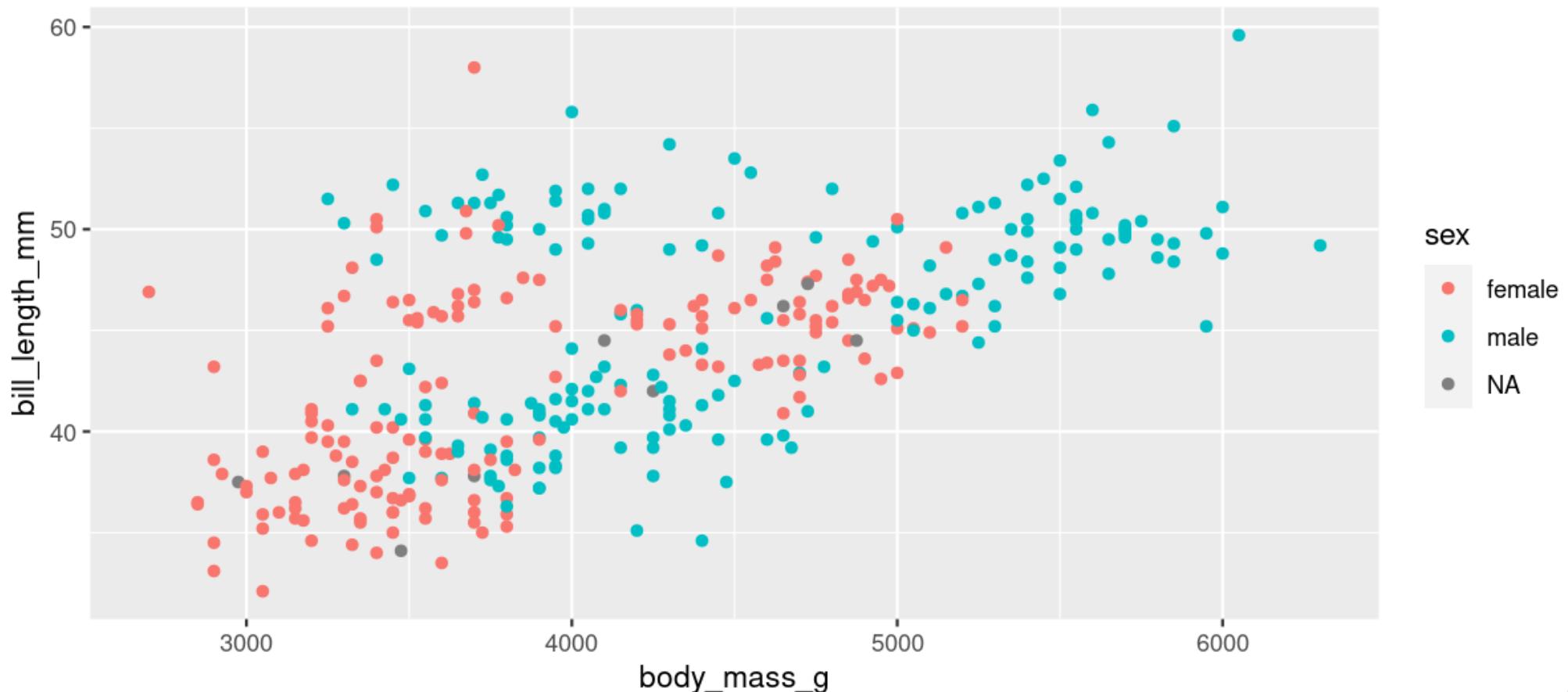
Mapping aesthetics

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +  
  geom_point()
```



Mapping aesthetics

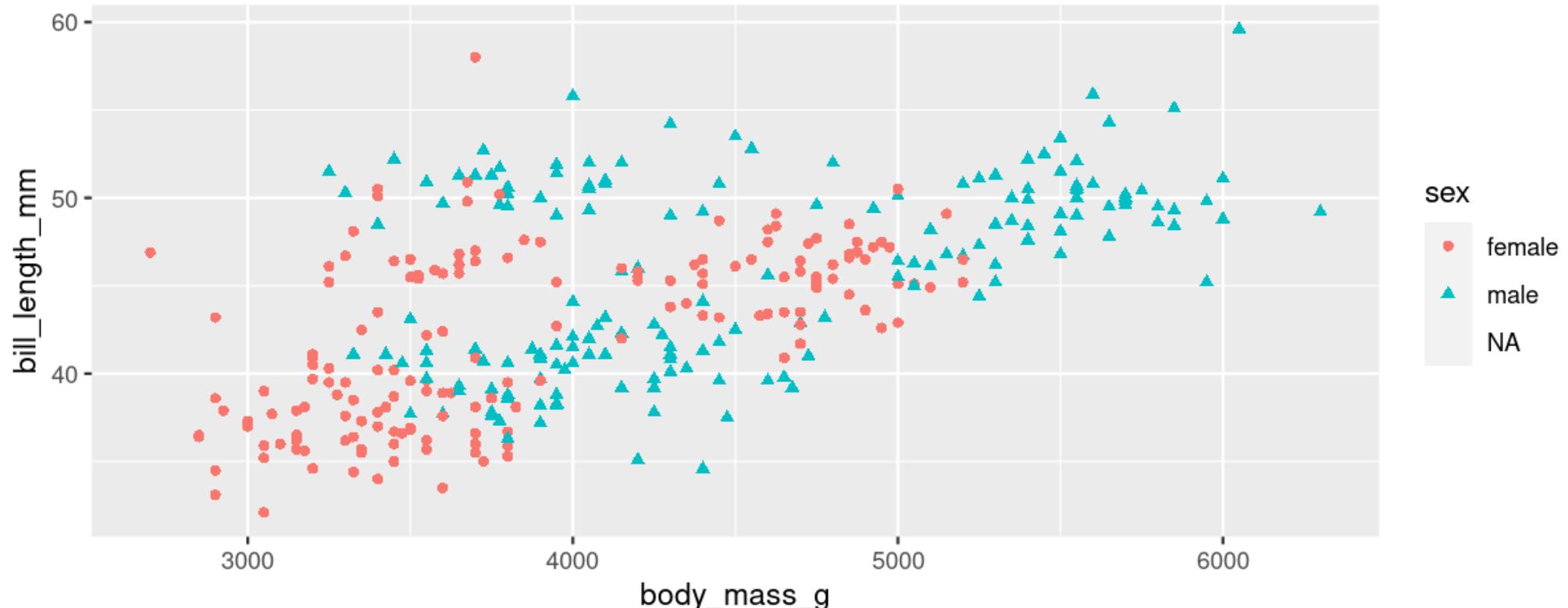
```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +  
  geom_point()
```



Mapping aesthetics

ggplot automatically populates the legends (combining where it can)

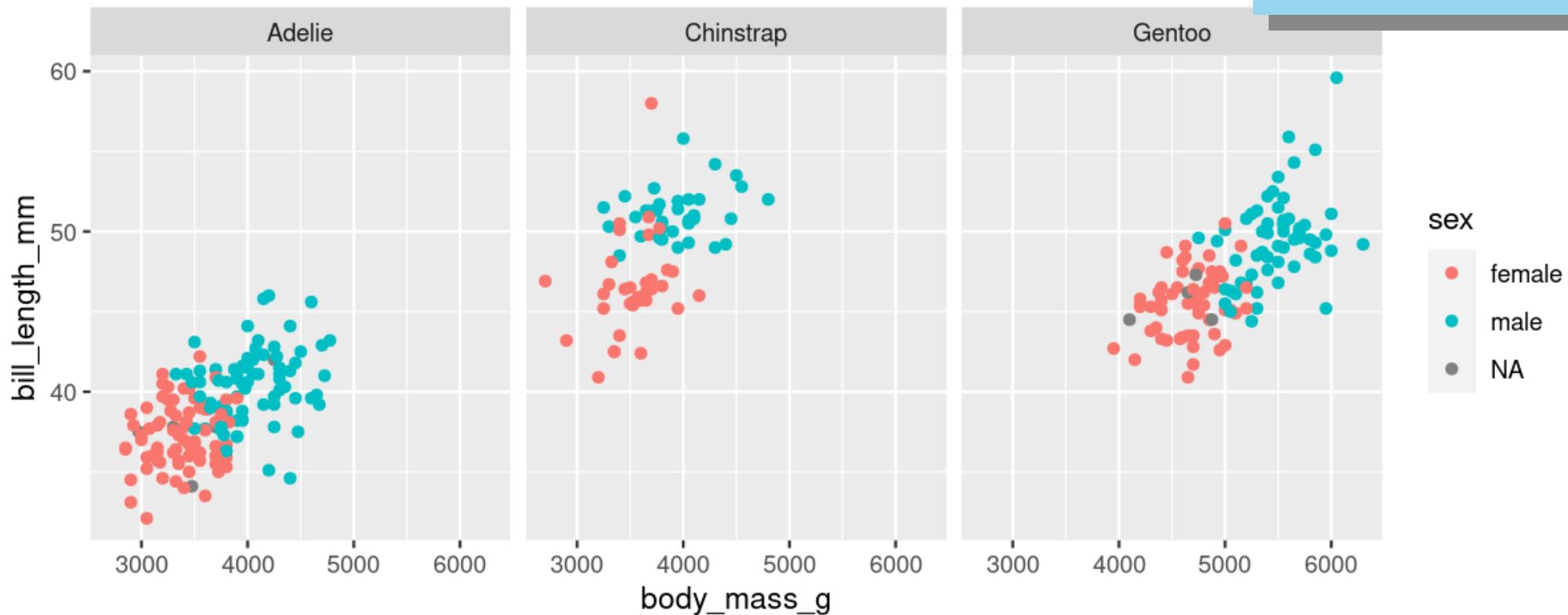
```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex, shape = sex)) +  
  geom_point()
```



Faceting: `facet_wrap()`

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +  
  geom_point() +  
  facet_wrap(~ species)
```

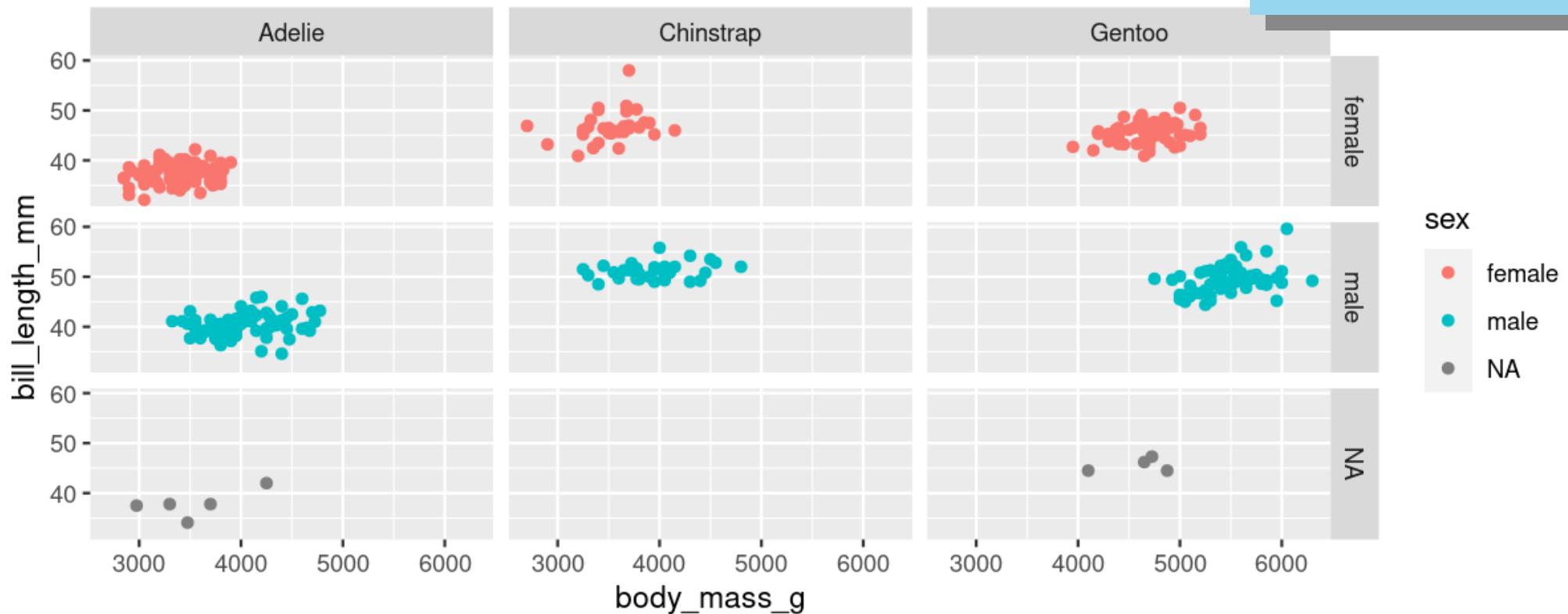
Split plots by **one** grouping variable



Faceting: `facet_grid()`

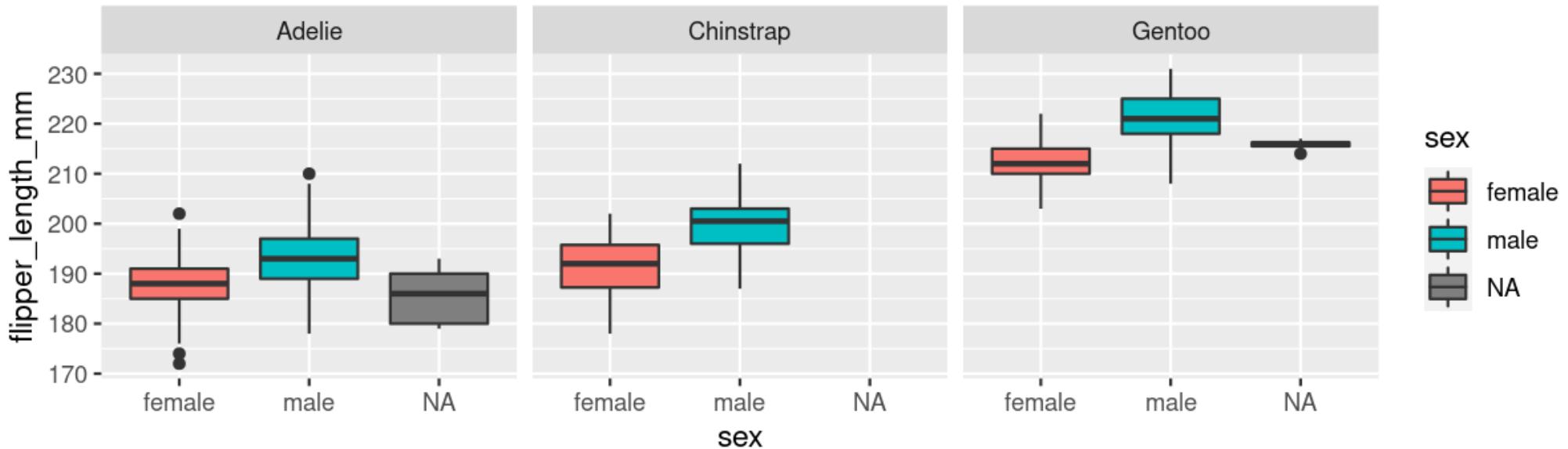
```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +  
  geom_point() +  
  facet_grid(sex ~ species)
```

Split plots by **two** grouping variables



Your Turn: Create this plot

```
ggplot(data = [REDACTED], aes([REDACTED])) +  
  [REDACTED] +  
  [REDACTED]
```

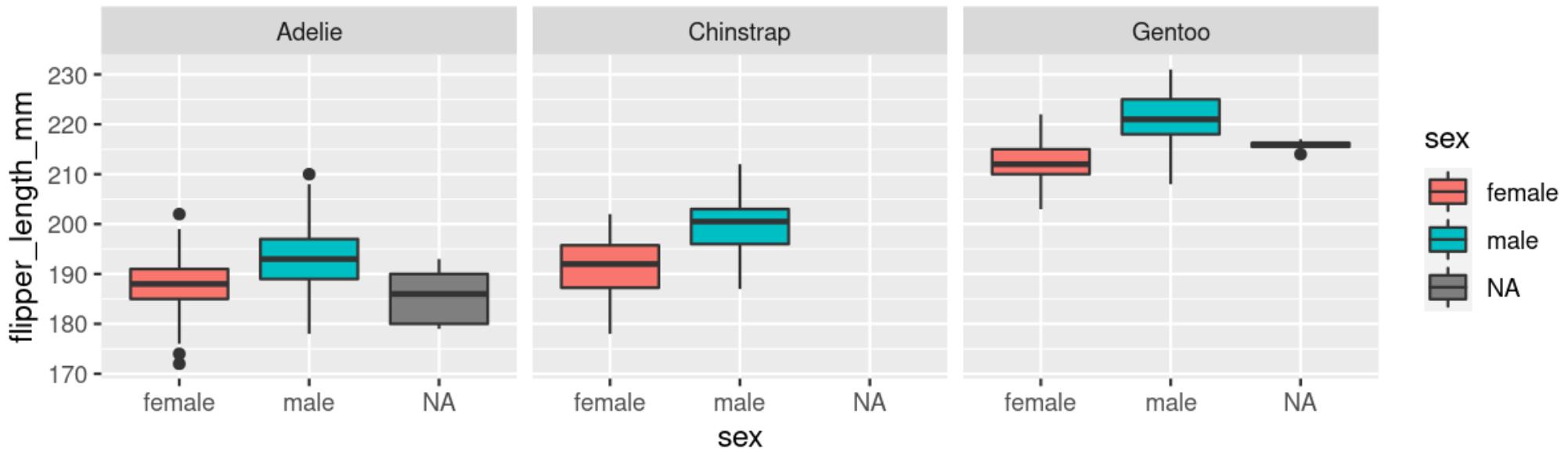


Hint: **colour** is for outlining with a colour, **fill** is for 'filling' with a colour

Too Easy? Split boxplots by sex **and** island

Your Turn: Create this plot

```
ggplot(data = penguins, aes(x = sex, y = flipper_length_mm, fill = sex)) +  
  geom_boxplot() +  
  facet_wrap(~ species)
```

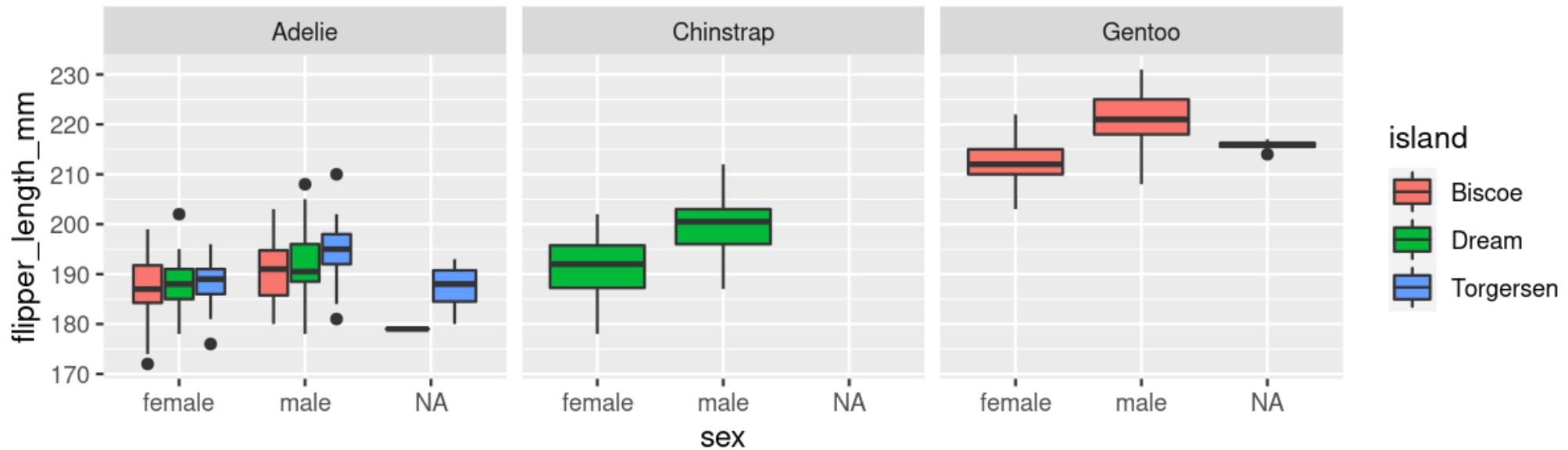


Hint: **colour** is for outlining with a colour, **fill** is for 'filling' with a colour
Too Easy? Split boxplots by sex **and** island

Your Turn: Create this plot

Too Easy?

```
ggplot(data = penguins, aes(x = sex, y = flipper_length_mm, fill = island)) +  
  geom_boxplot() +  
  facet_wrap(~ species)
```



Small change (**fill = sex** to **fill = island**) results in completely different plot

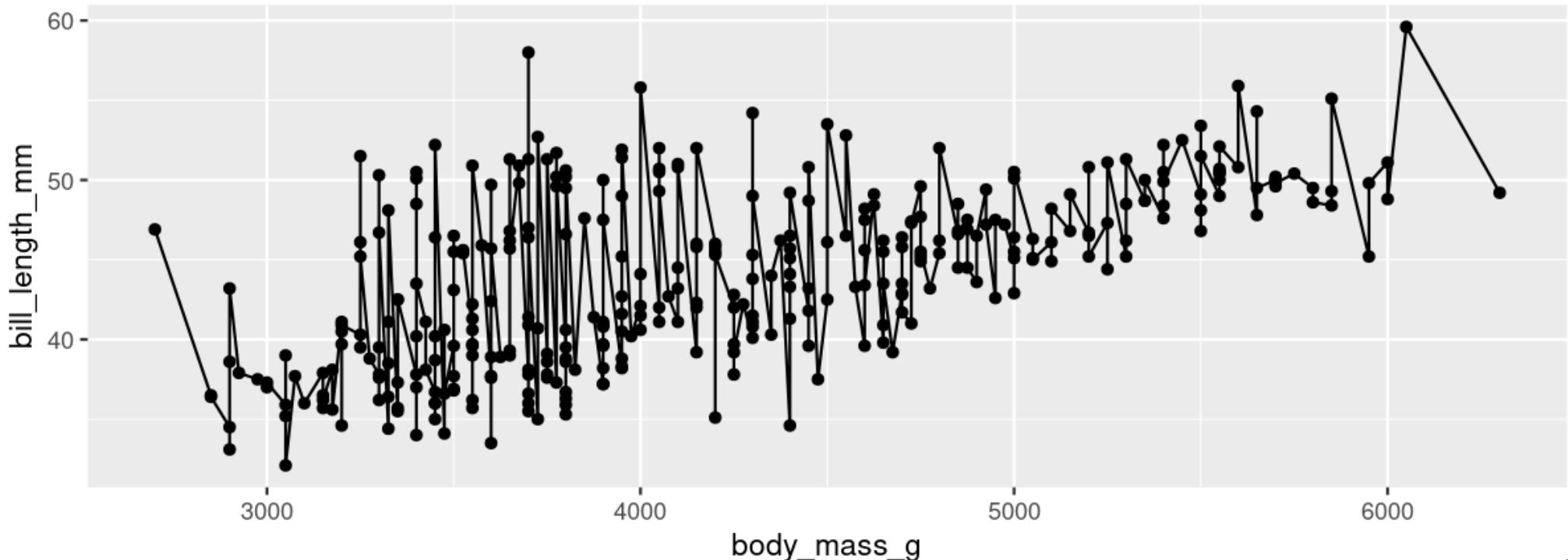
60 / 101

Trendlines / Regression Lines

Trendlines / Regression lines

geom_line() is connect-the-dots, not a trend or linear model

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +  
  geom_point() +  
  geom_line()
```

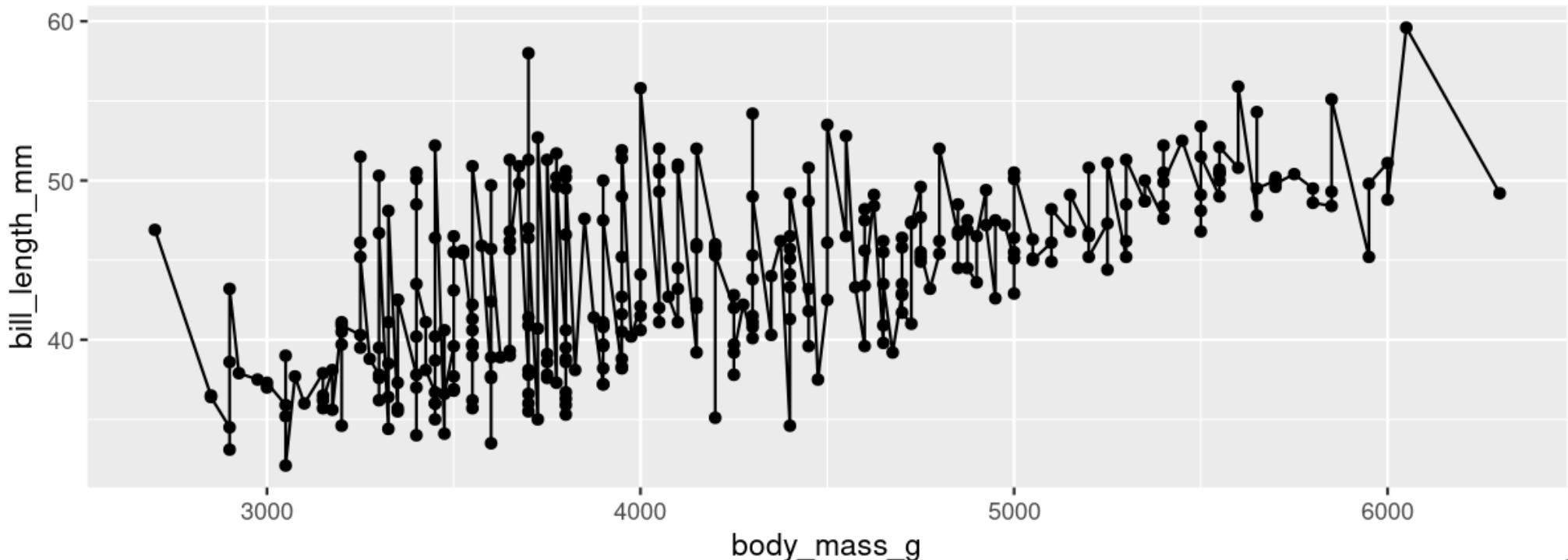


Trendlines / Regression lines

geom_line() is connect-the-dots, not a trend or linear model

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +  
  geom_point() +  
  geom_line()
```

Not what we're
looking for

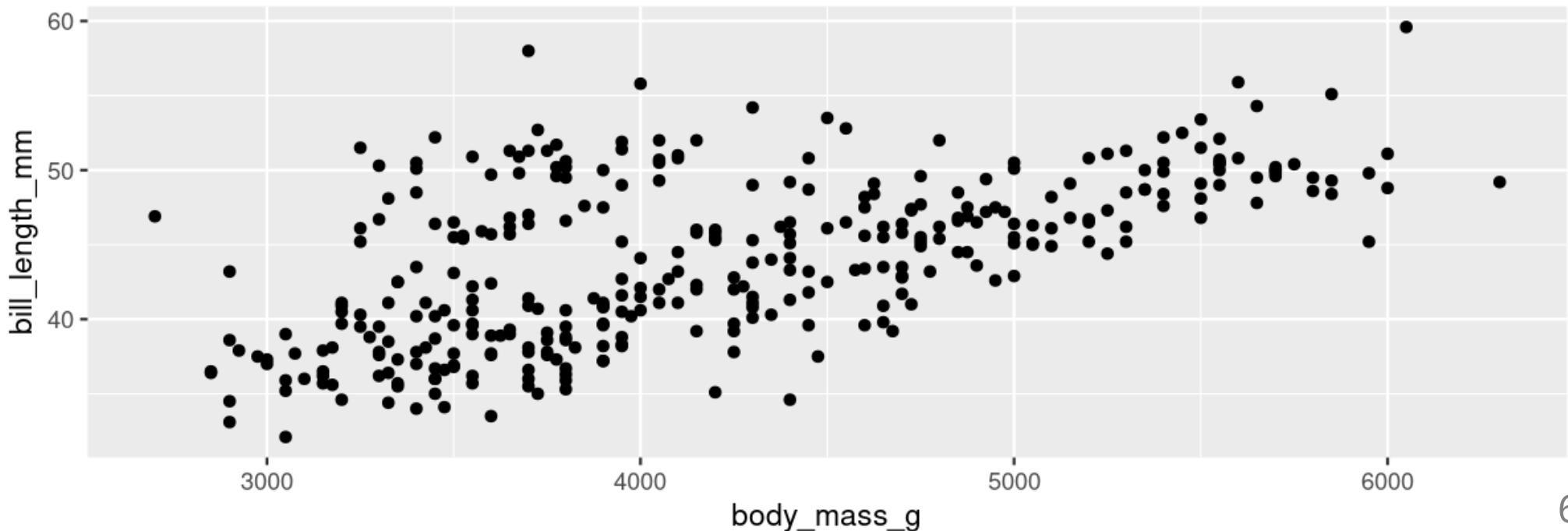


Trendlines / Regression lines

Let's add a trend line properly

Start with basic plot:

```
g <- ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm)) +  
  geom_point()  
g
```

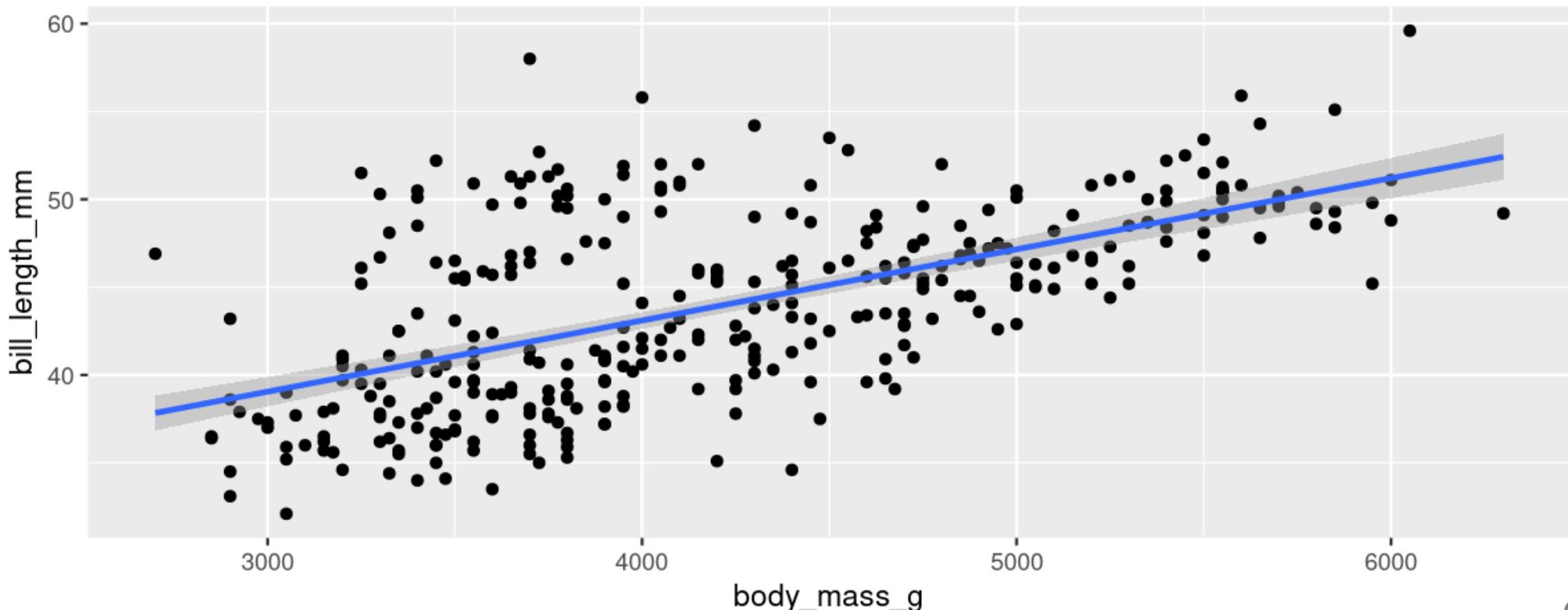


Trendlines / Regression lines

Add the `stat_smooth()`

```
g + stat_smooth(method = "lm")
```

- `lm` is for "linear model" (i.e. trendline)
- grey ribbon = standard error

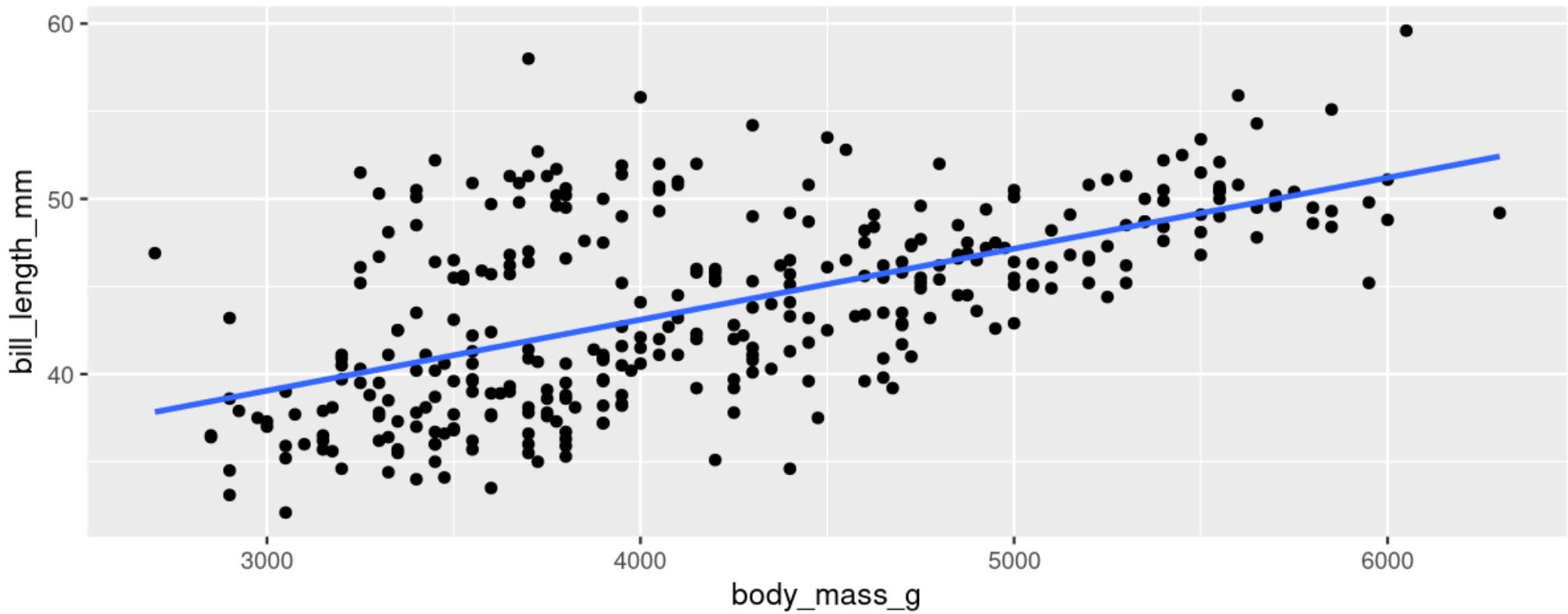


Trendlines / Regression lines

Add the `stat_smooth()`

```
g + stat_smooth(method = "lm", se = FALSE)
```

- remove the grey ribbon `se = FALSE`

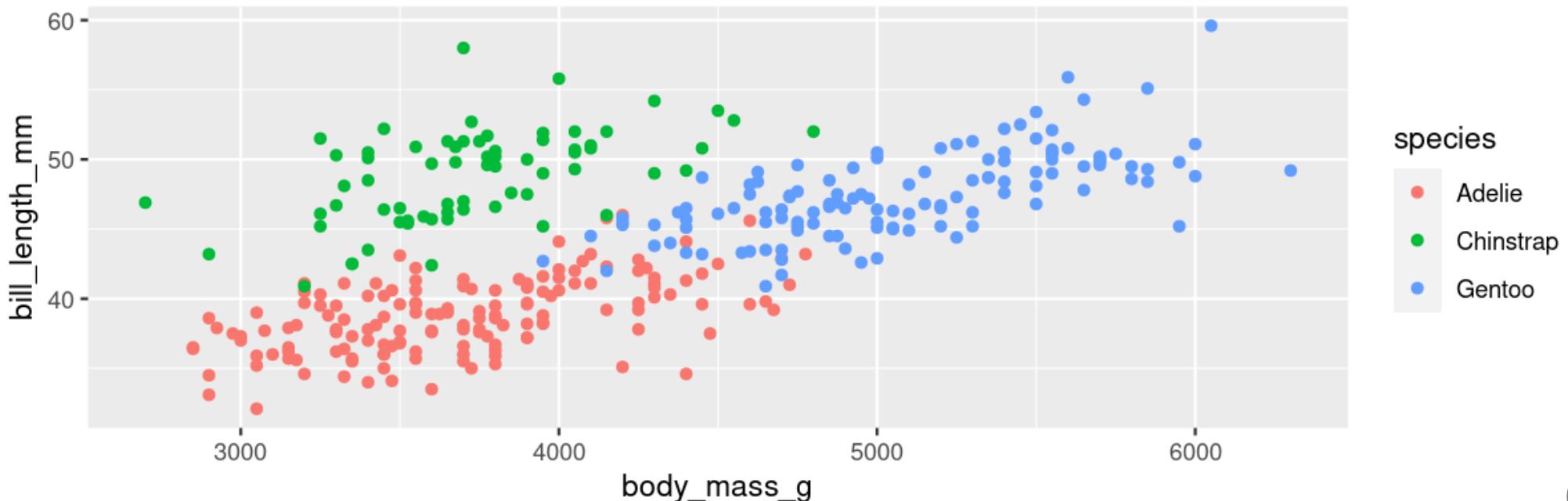


Trendlines / Regression lines

A line for each group

- Specify group (here we use **colour** to specify **species**)

```
g <- ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()  
  
g
```

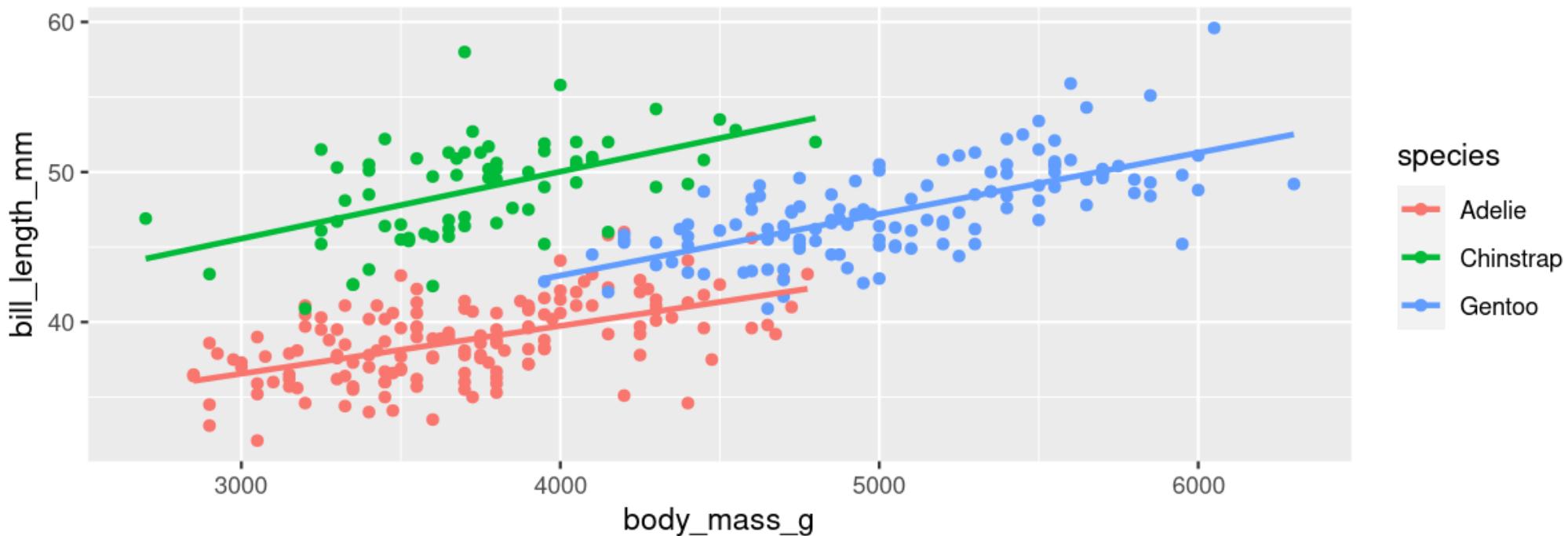


Trendlines / Regression lines

A line for each group

- `stat_smooth()` automatically uses the same grouping

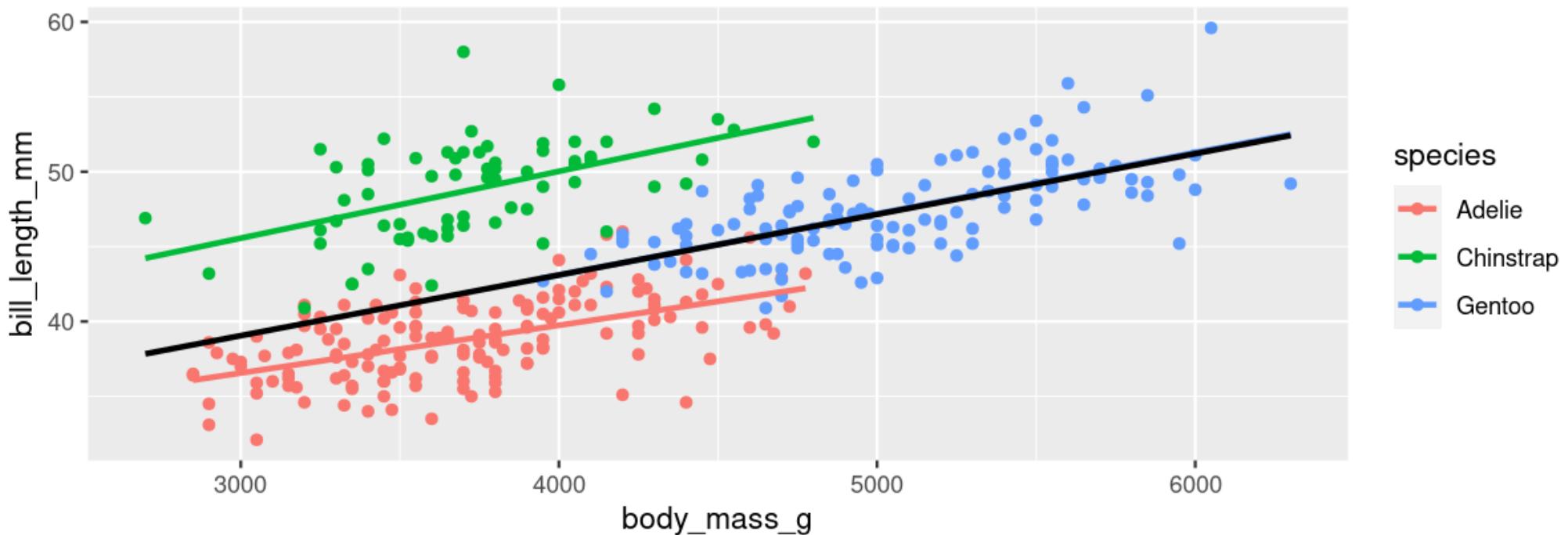
```
g + stat_smooth(method = "lm", se = FALSE)
```



Trendlines / Regression lines

A line for each group AND overall

```
g +
  stat_smooth(method = "lm", se = FALSE) +
  stat_smooth(method = "lm", se = FALSE, colour = "black")
```



Your Turn: Create this plot

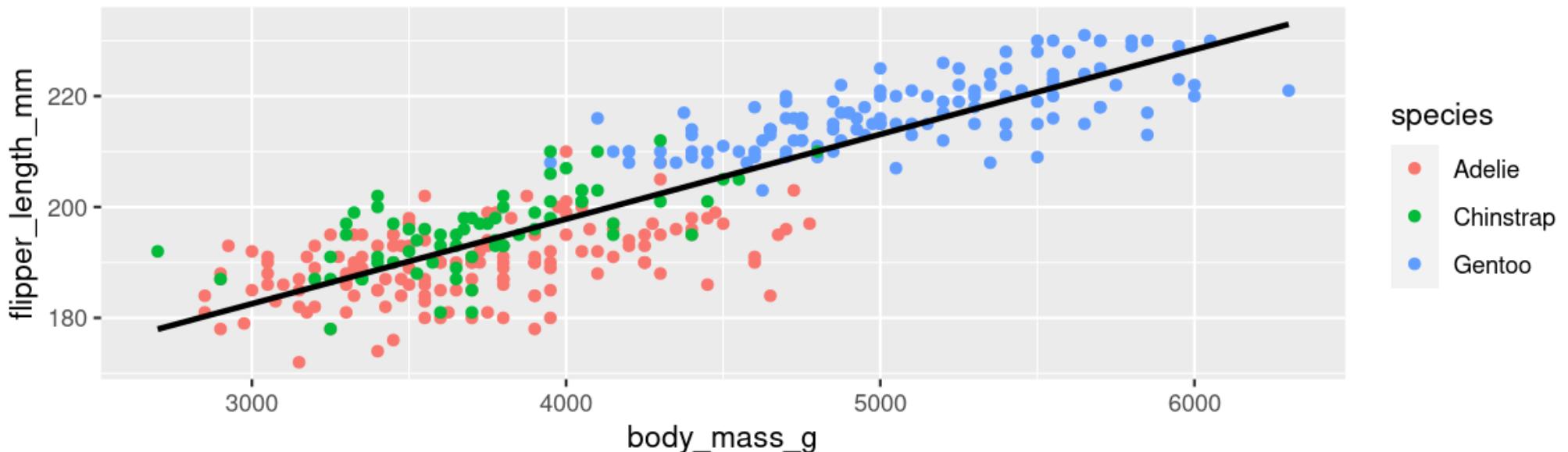
- A scatter plot: Flipper Length by Body Mass grouped by Species
- With *a single regression line for the overall trend*

Too Easy? Create a separate plot for each sex as well

Your Turn: Create this plot

- A scatter plot: Flipper Length by Body Mass grouped by Species
- With *a single regression line for the overall trend*

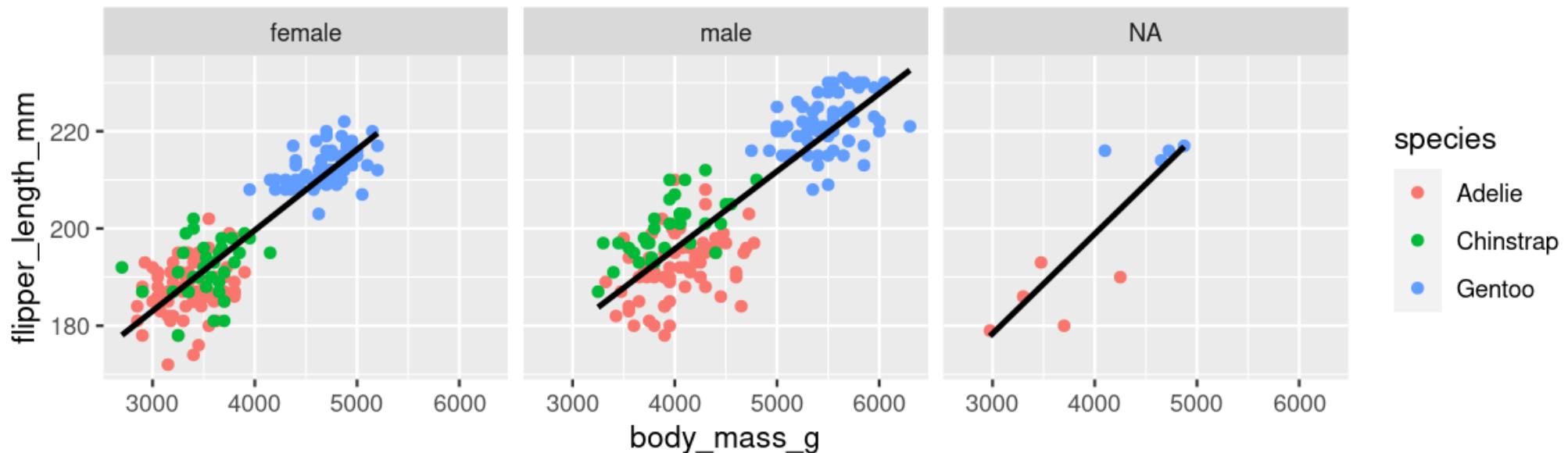
```
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +  
  geom_point() +  
  stat_smooth(se = FALSE, colour = "black", method = "lm")
```



Your Turn: Create this plot

Too Easy?

```
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +  
  geom_point() +  
  stat_smooth(se = FALSE, colour = "black", method = "lm") +  
  facet_wrap(~sex)
```

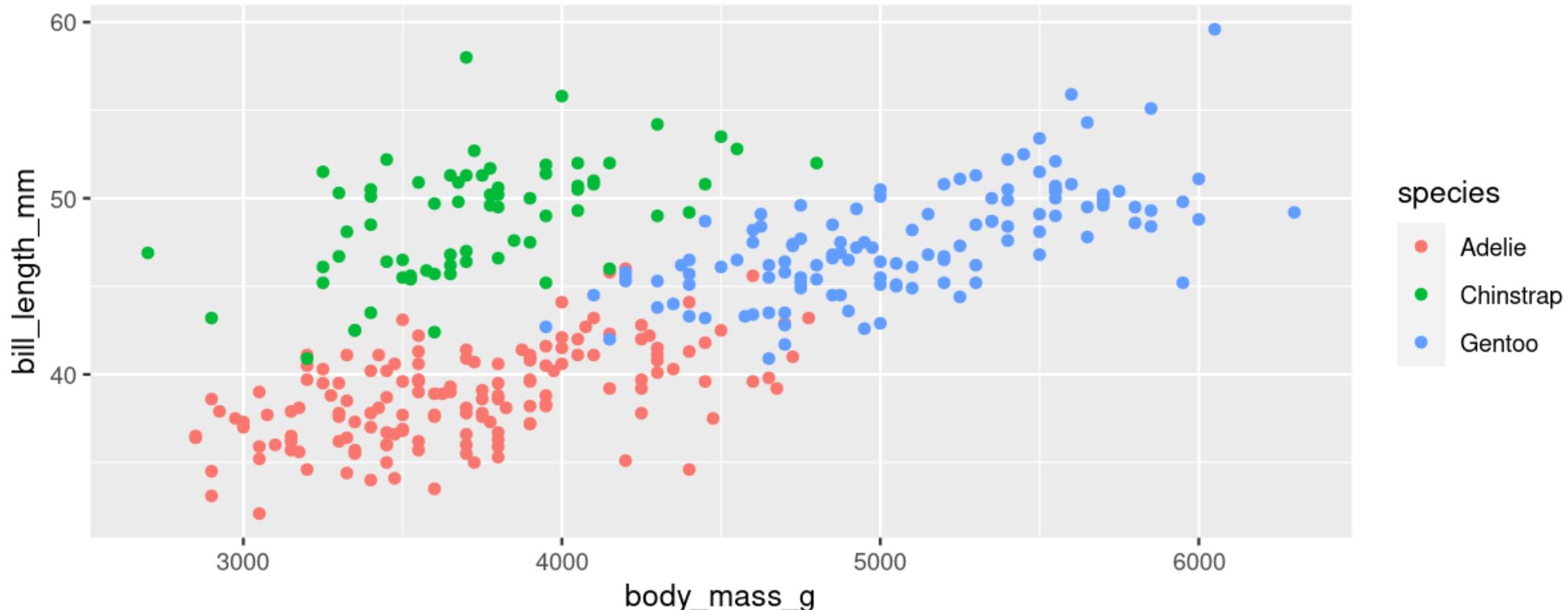


Customizing plots

Customizing: Starting plot

Let's work with this plot

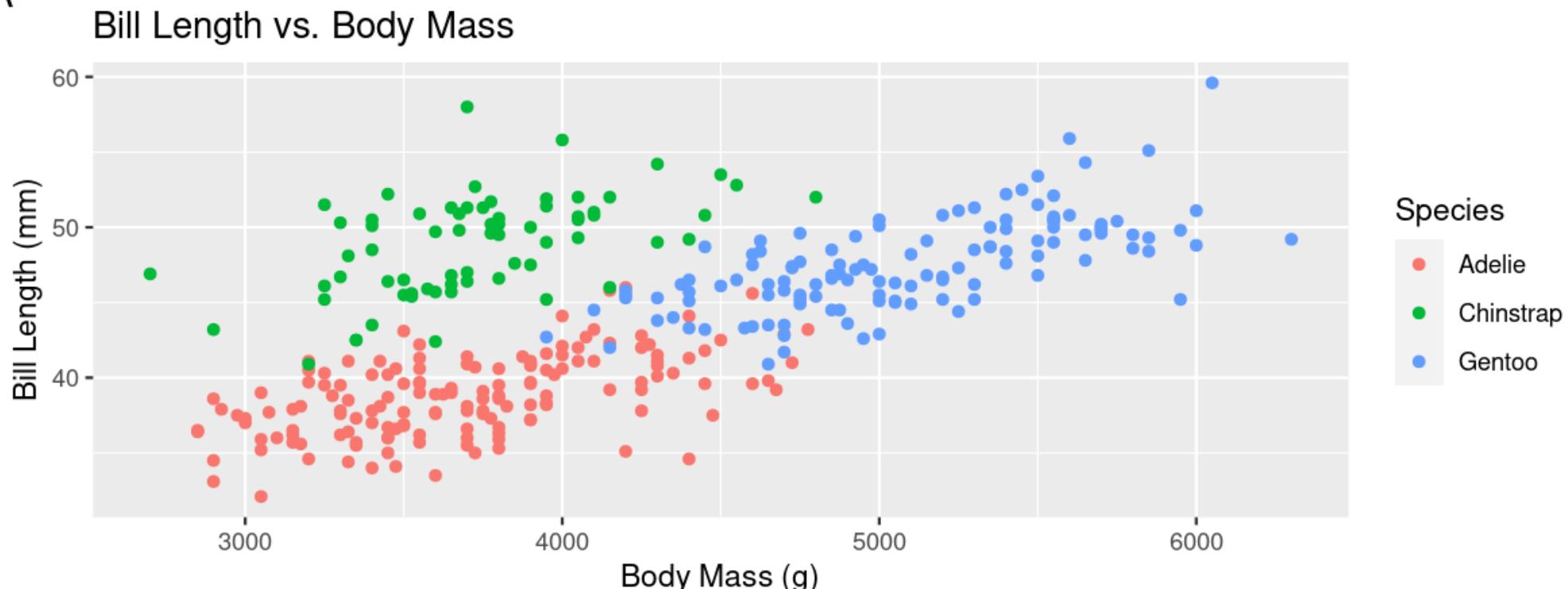
```
g <- ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point()
```



Customizing: Labels

```
g + labs(title = "Bill Length vs. Body Mass",  
         x = "Body Mass (g)",  
         y = "Bill Length (mm)",  
         colour = "Species", tag = "A")
```

A



Customizing: Labels

```
g + labs(title = "Bill Length vs. Body Mass",  
        x = "Body Mass (g)",  
        y = "Bill Length (mm)",  
        colour = "Species", tag = "A")
```

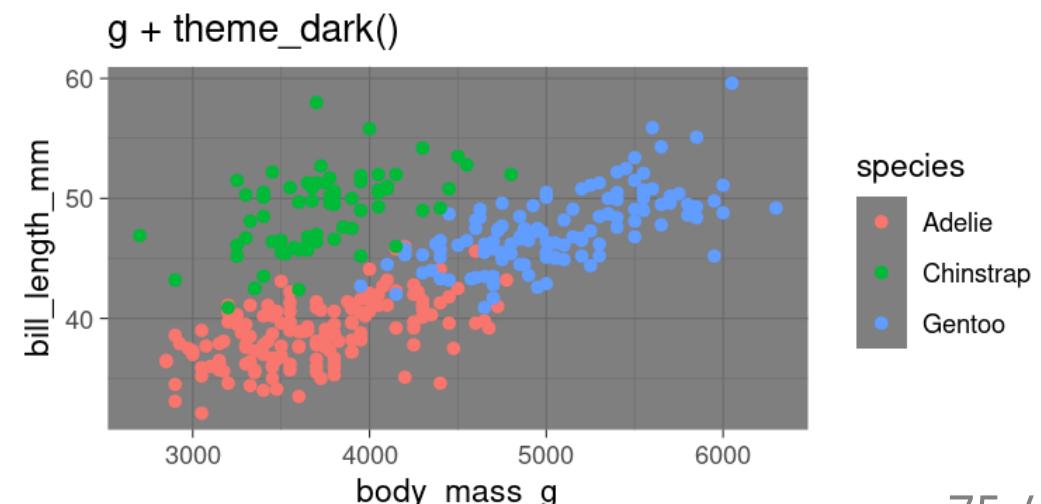
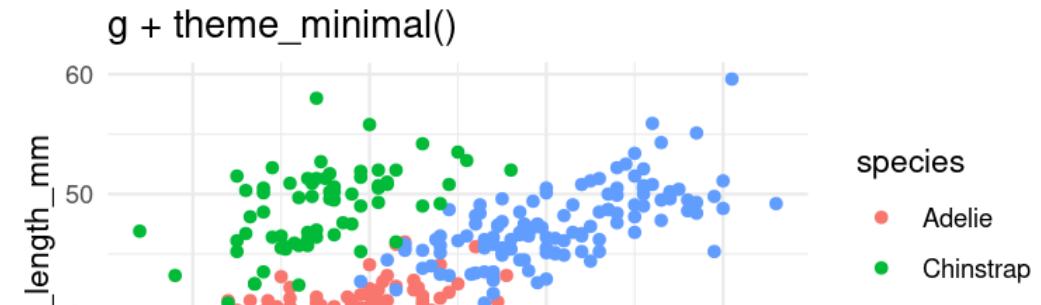
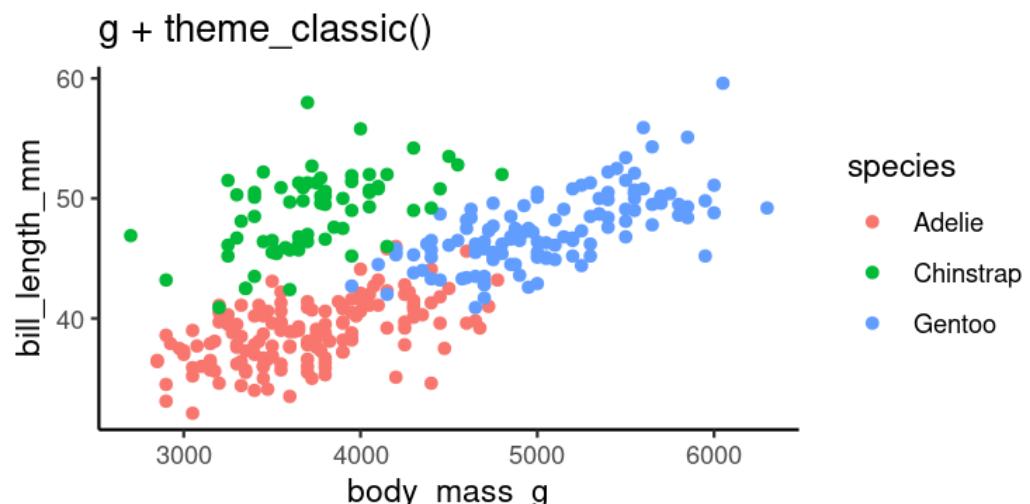
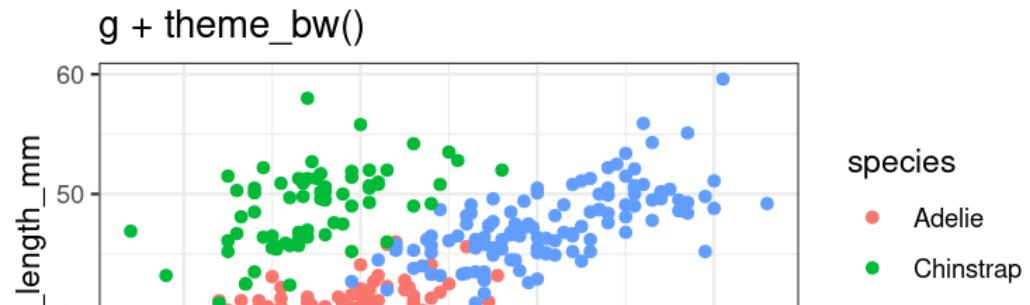
Practice for later

Add proper labels to some of your previous plots

A



Customizing: Built-in themes



Customizing: Axes

scale_ + (x or y) + type (**continuous**, **discrete**, **date**, **datetime**)

- **scale_x_continuous()**
- **scale_y_discrete()**
- etc.

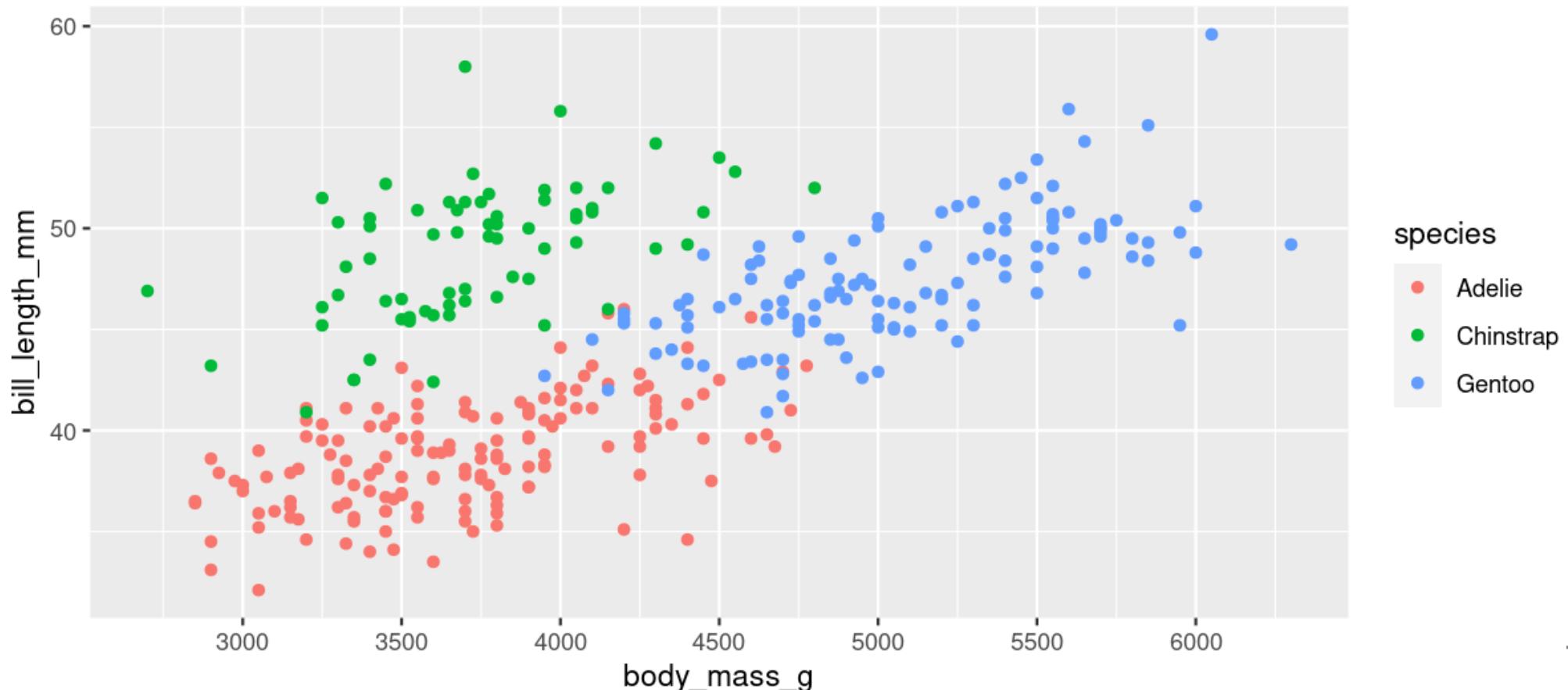
Common arguments

```
g + scale_x_continuous(breaks = seq(0, 20, 10)) # Tick breaks
g + scale_x_continuous(limits = c(0, 15))        # xlim() is a shortcut for this
g + scale_x_continuous(expand = c(0, 0))          # Space between axis and data
```

Customizing: Axes

Breaks

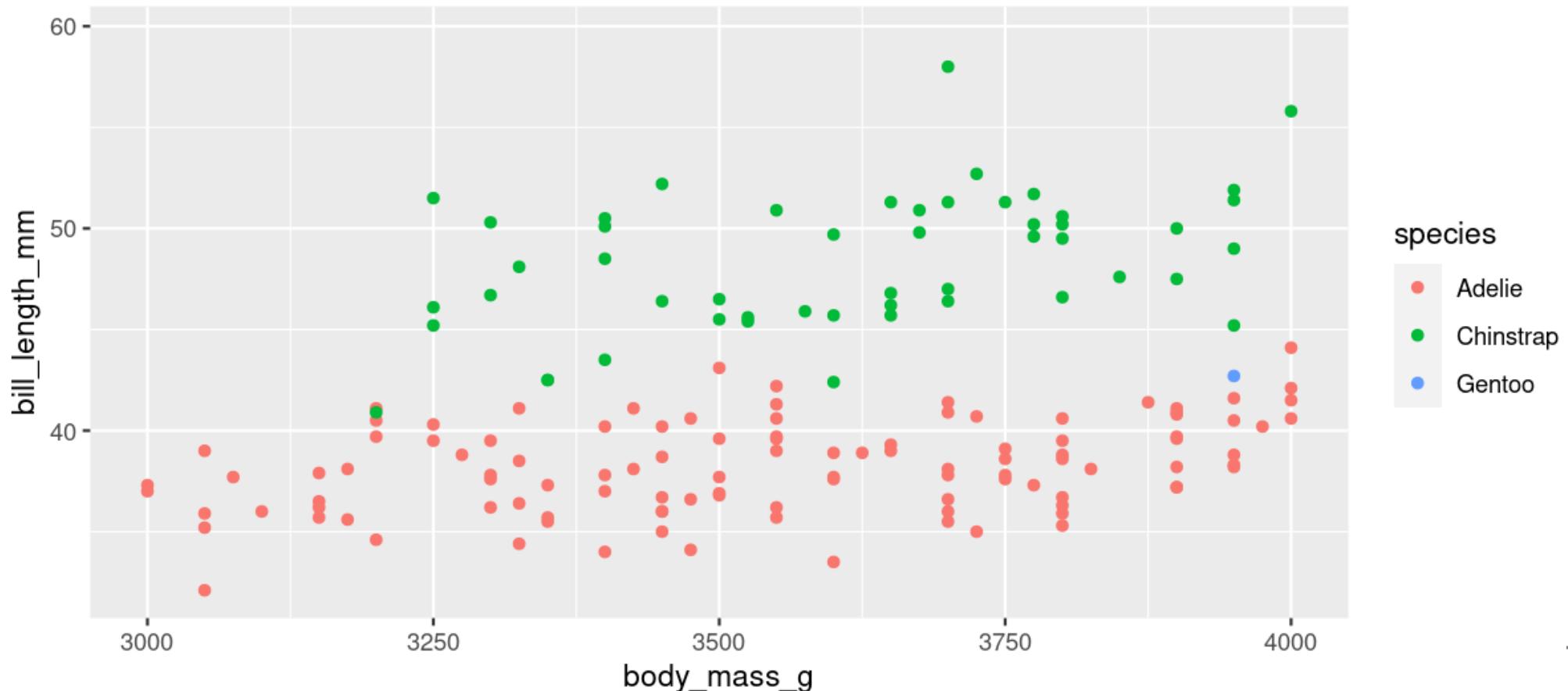
```
g + scale_x_continuous(breaks = seq(2500, 6500, 500))
```



Customizing: Axes

Limits

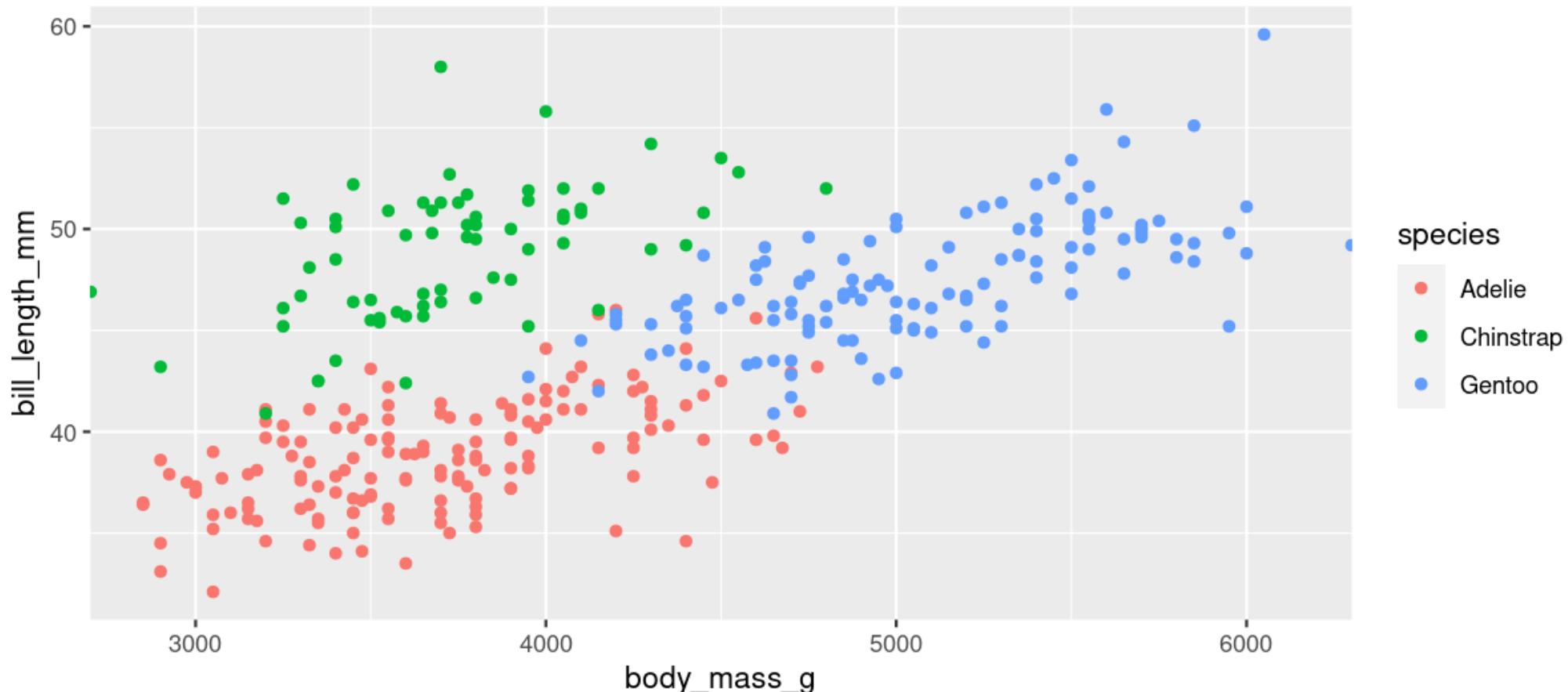
```
g + scale_x_continuous(limits = c(3000, 4000))
```



Customizing: Axes

Space between origin and axis start

```
g + scale_x_continuous(expand = c(0, 0))
```

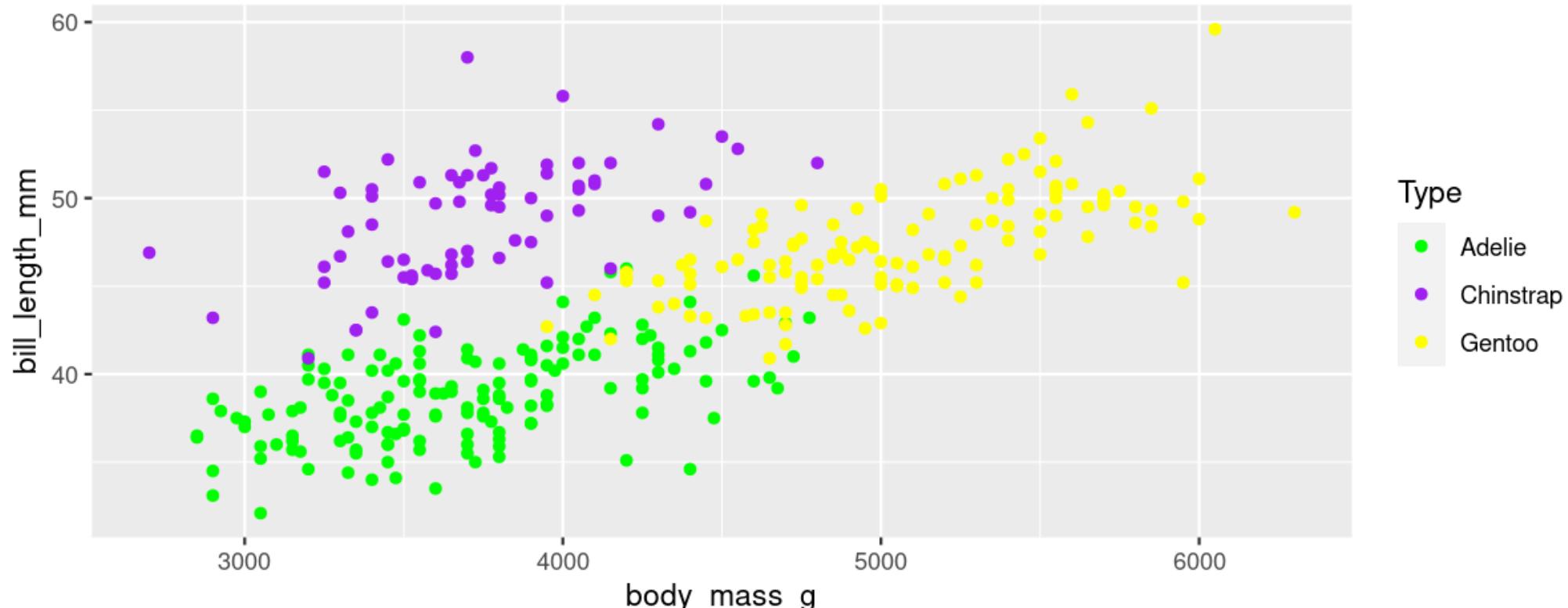


Customizing: Aesthetics

Using scales

`scale_` + aesthetic (**colour**, **fill**, **size**, etc.) + type (**manual**, **continuous**, **datetime**, etc.)

```
g + scale_colour_manual(name = "Type", values = c("green", "purple", "yellow"))
```

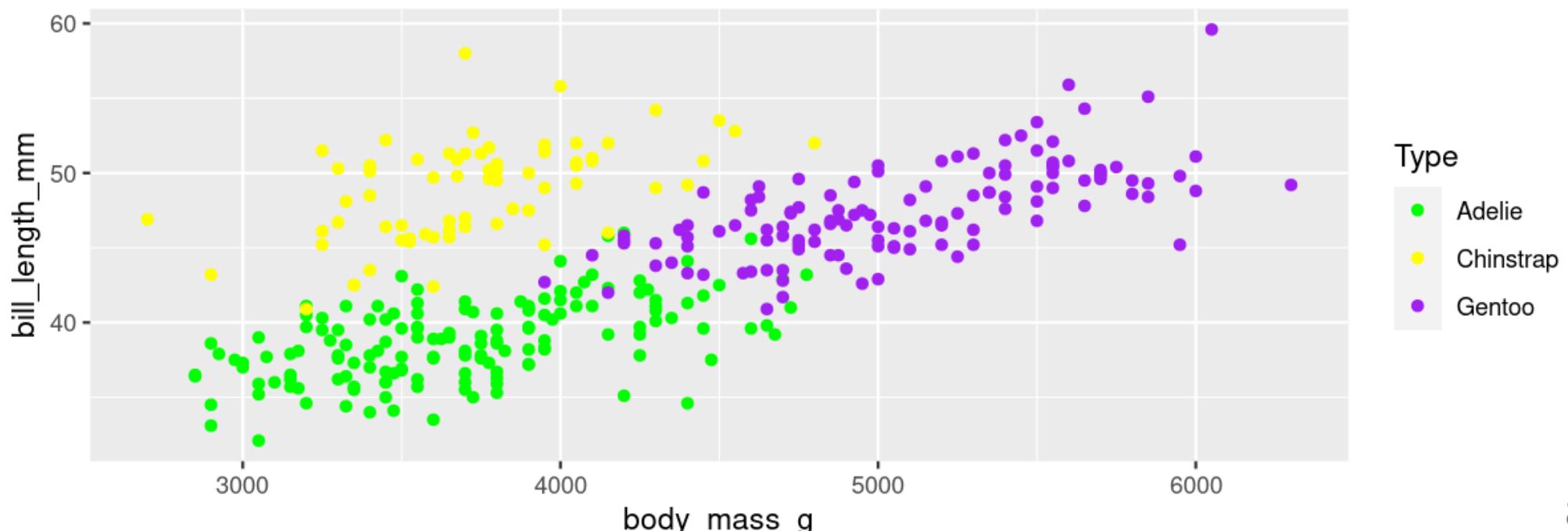


Customizing: Aesthetics

Using scales

Or be very explicit:

```
g + scale_colour_manual(name = "Type",
                         values = c("Adelie" = "green", "Gentoo" = "purple", "Chinstrap" = "yellow"),
                         na.value = "black")
```

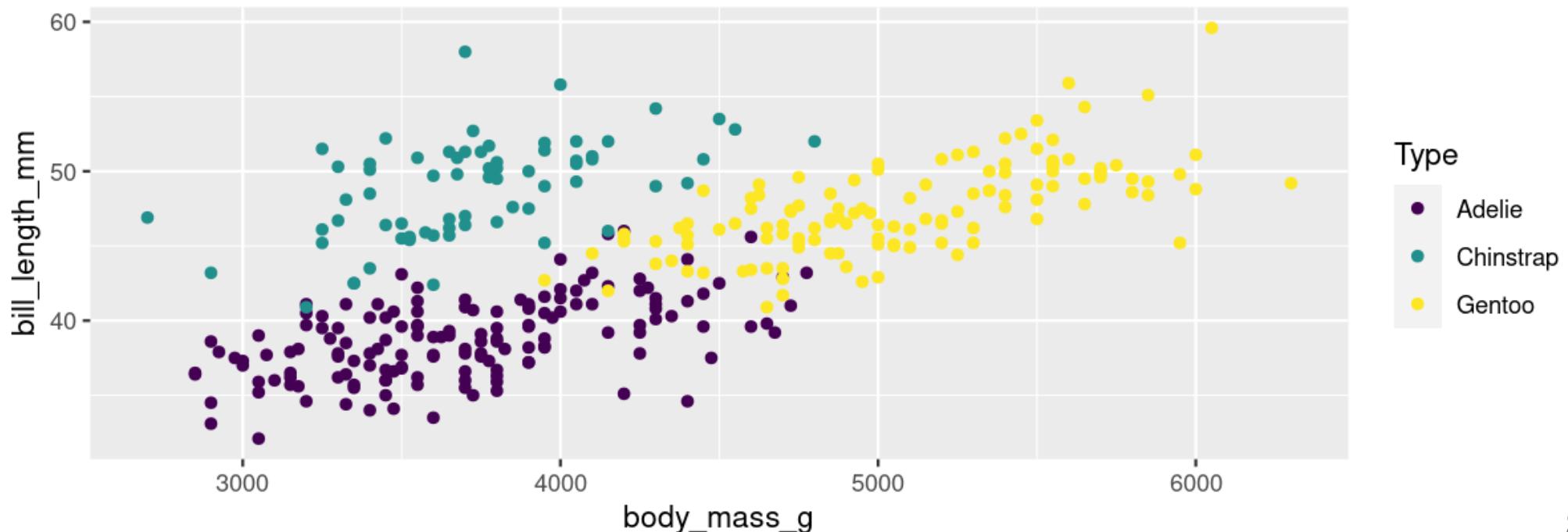


Customizing: Aesthetics

For colours, consider colour-blind-friendly scale

`viridis_d` for "discrete" data

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +  
  geom_point() +  
  scale_colour_viridis_d(name = "Type")
```

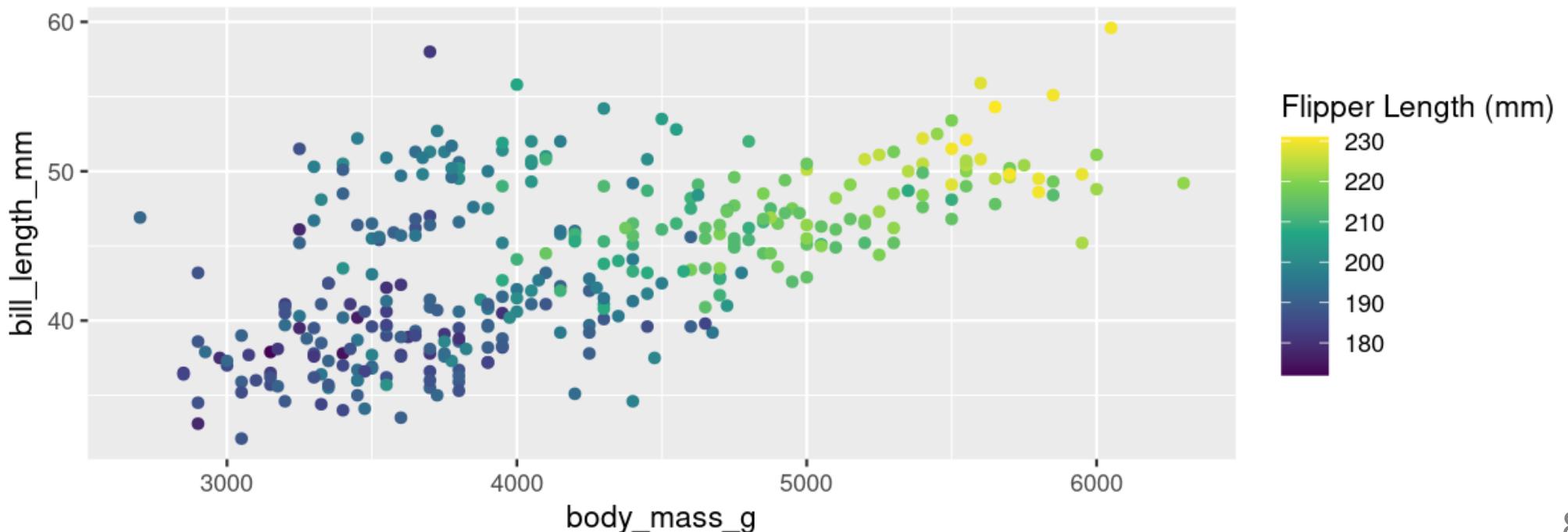


Customizing: Aesthetics

For colours, consider colour-blind-friendly scale

`viridis_c` for "continuous" data

```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = flipper_length_mm)) +  
  geom_point() +  
  scale_colour_viridis_c(name = "Flipper Length (mm)")
```

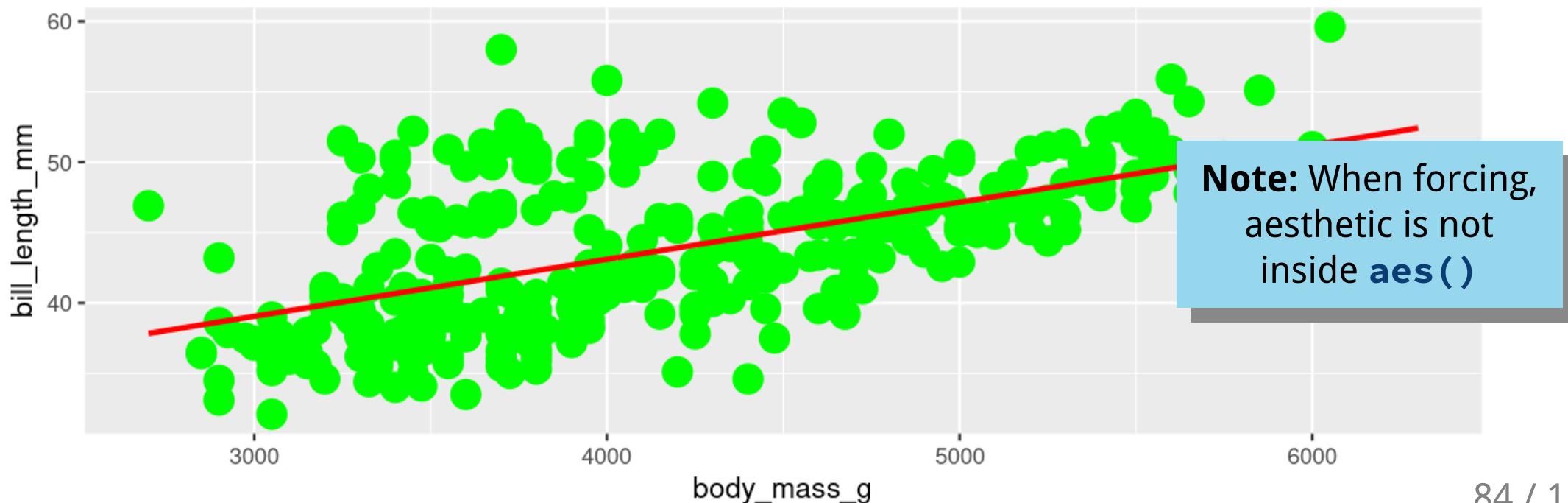


Customizing: Aesthetics

Forcing

Remove the association between a variable and an aesthetic

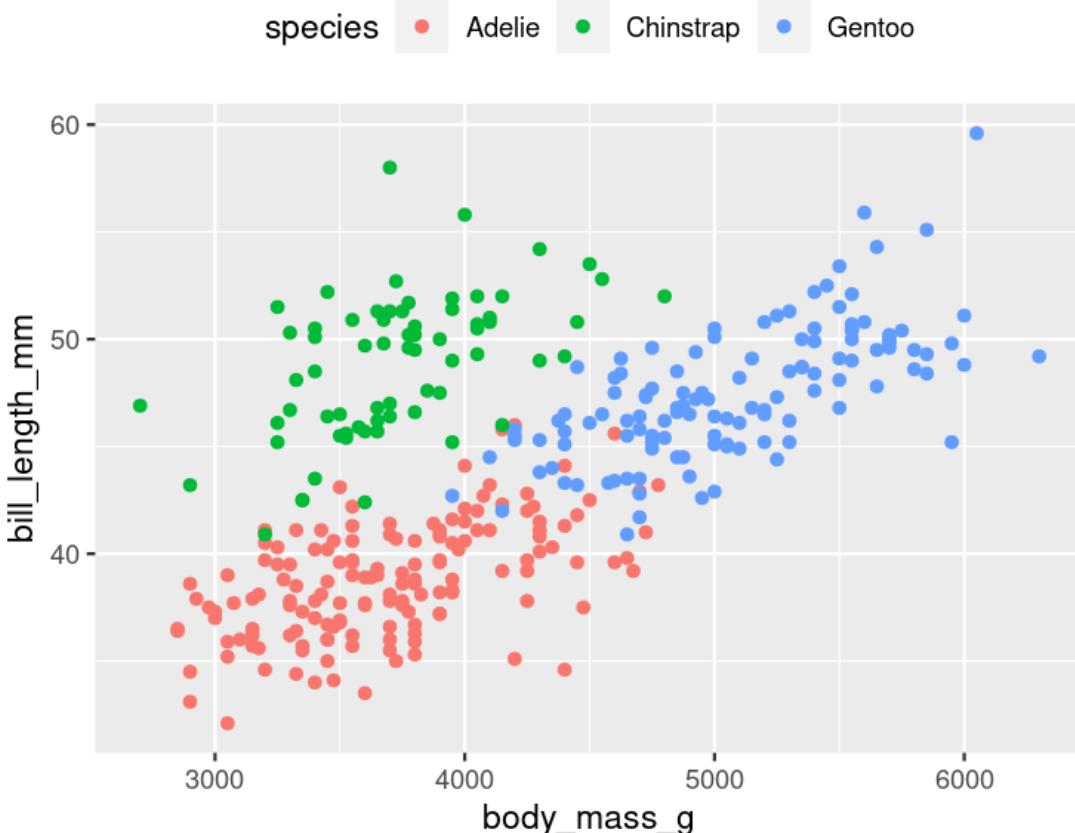
```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +  
  geom_point(colour = "green", size = 5) +  
  stat_smooth(method = "lm", se = FALSE, colour = "red")
```



Customizing: Legends placement

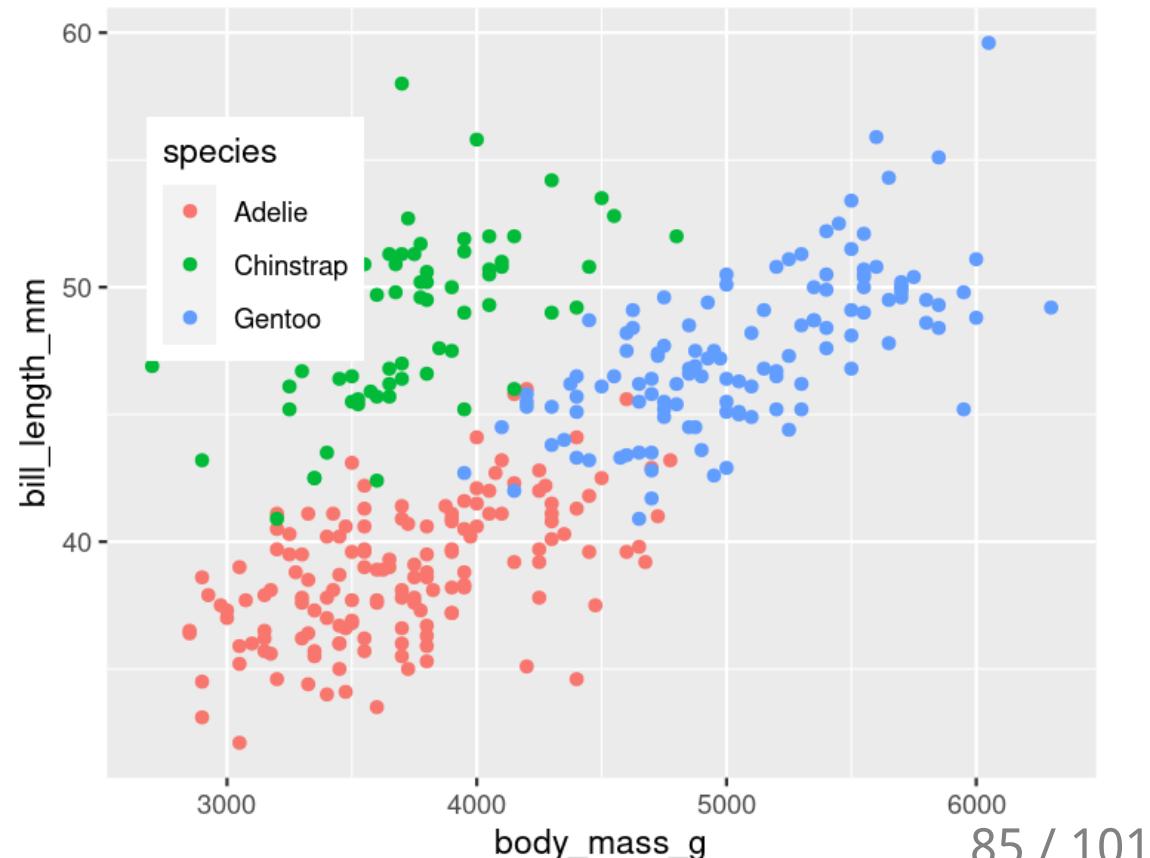
At the: top, bottom, left, right

```
g + theme(legend.position = "top")
```



Exactly here

```
g + theme(legend.position = c(0.15, 0.7))
```



Combining plots

Combining plots with **patchwork**

Setup

- Load **patchwork**
- Create a couple of different plots

```
library(patchwork)

g1 <- ggplot(data = penguins, aes(x = bill_length_mm, y = bill_depth_mm, colour = species)) +
  geom_point()

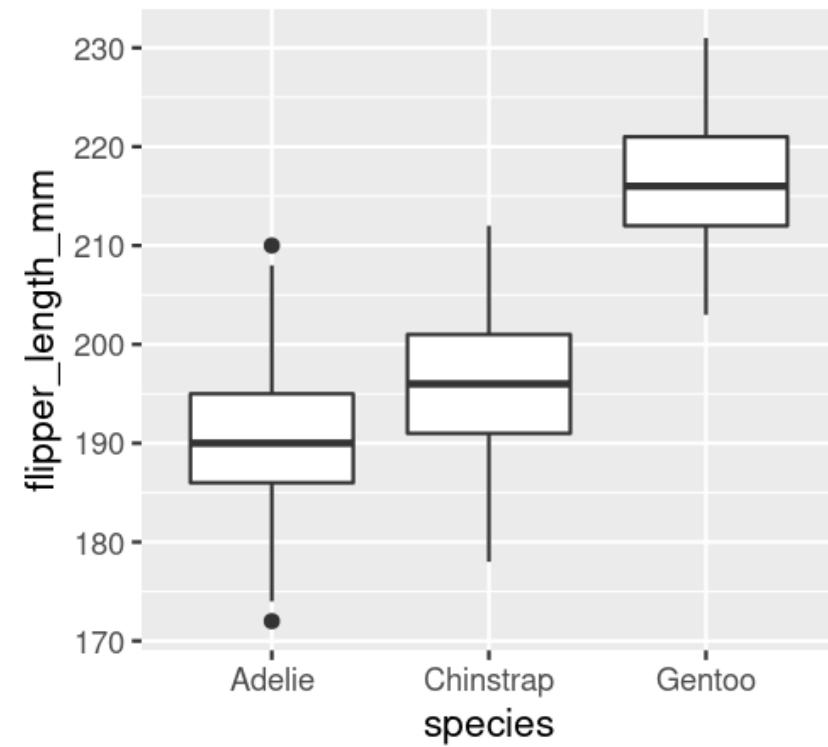
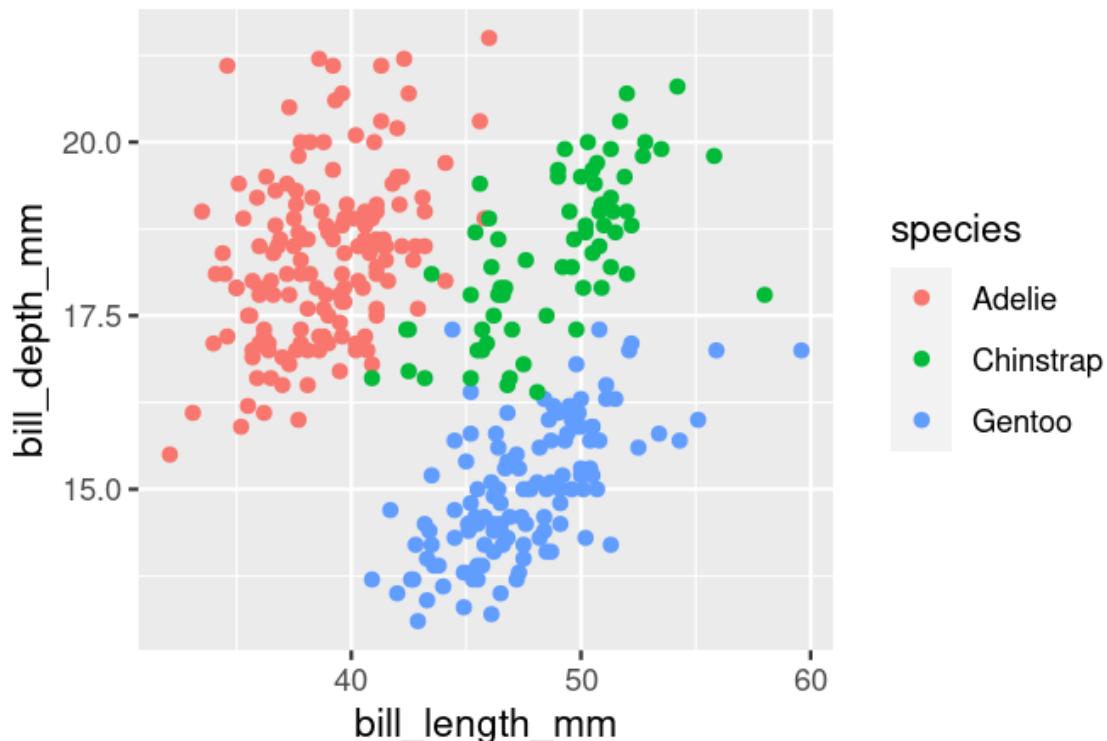
g2 <- ggplot(data = penguins, aes(x = species, y = flipper_length_mm)) +
  geom_boxplot()

g3 <- ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g, colour = species)) +
  geom_point()
```

Combining plots with patchwork

Side-by-Side 2 plots

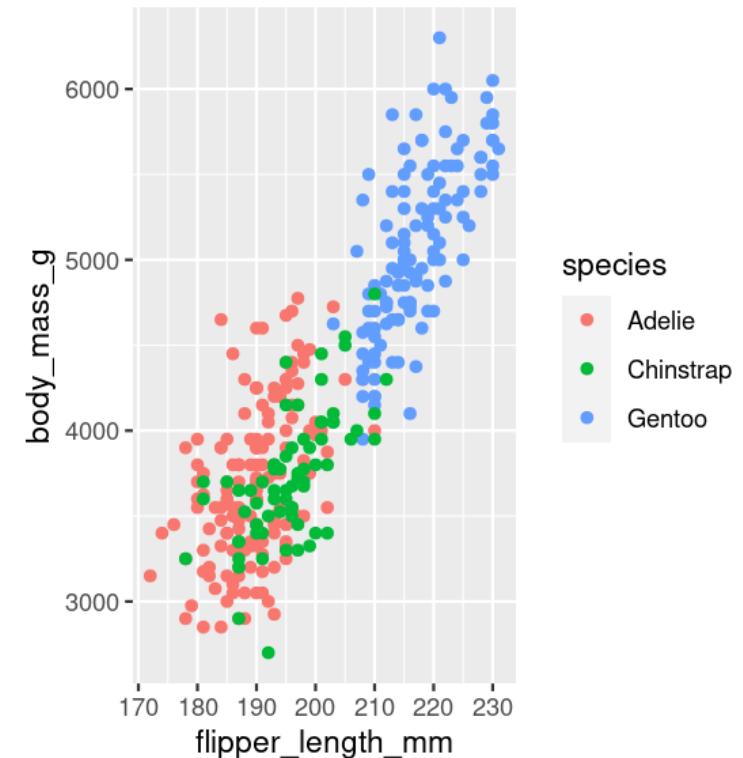
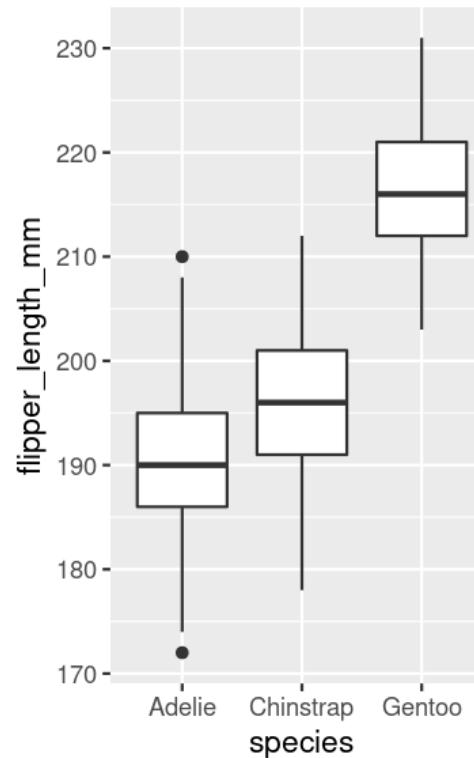
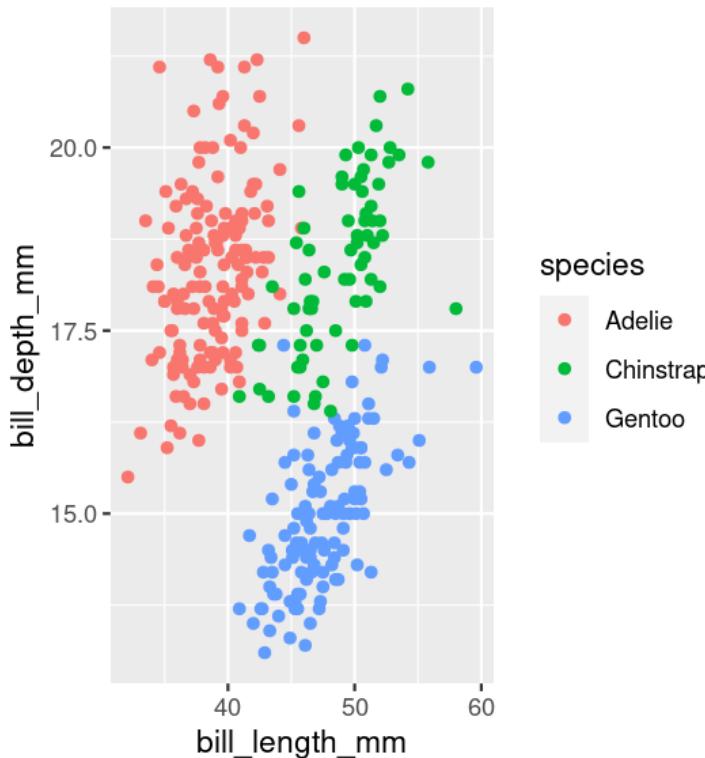
g1 + g2



Combining plots with patchwork

Side-by-Side 3 plots

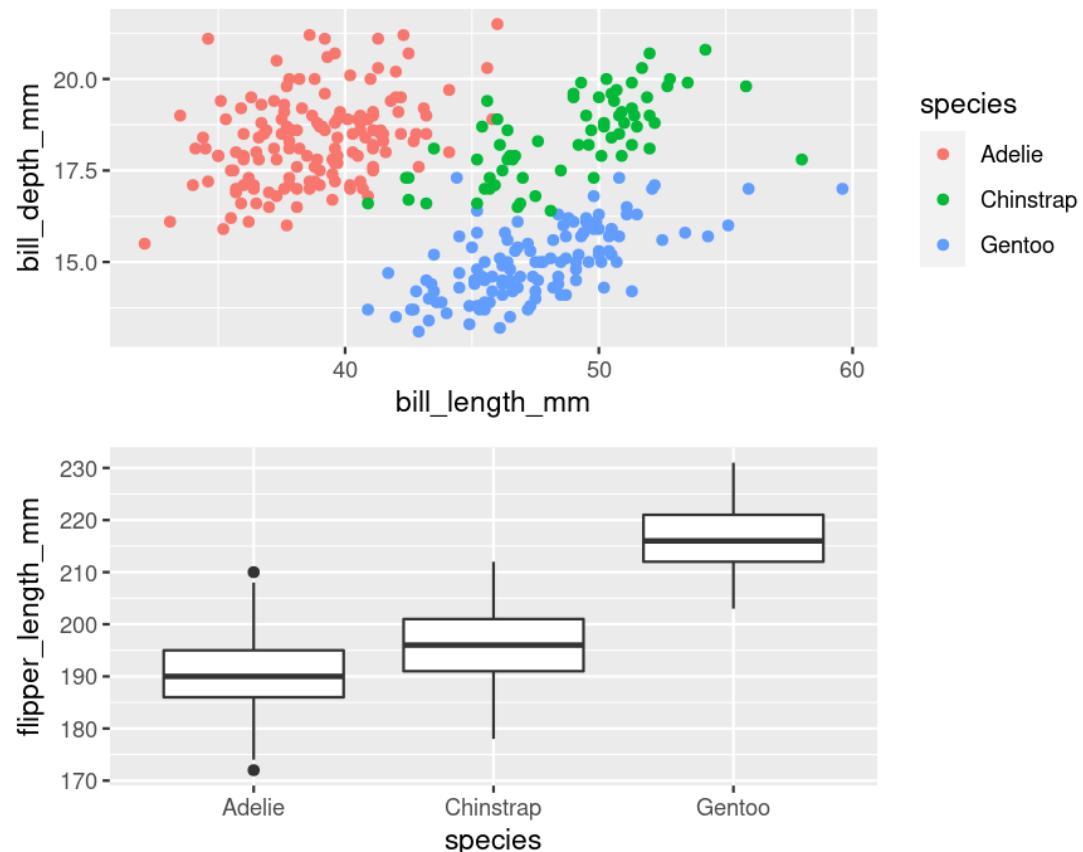
g1 + g2 + g3



Combining plots with patchwork

Stacked 2 plots

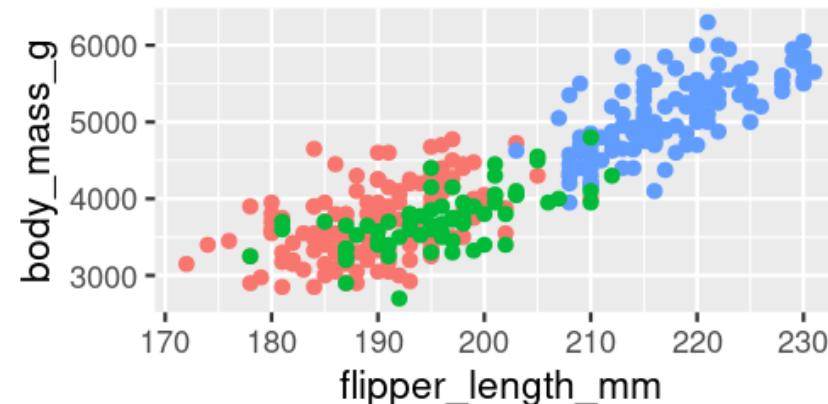
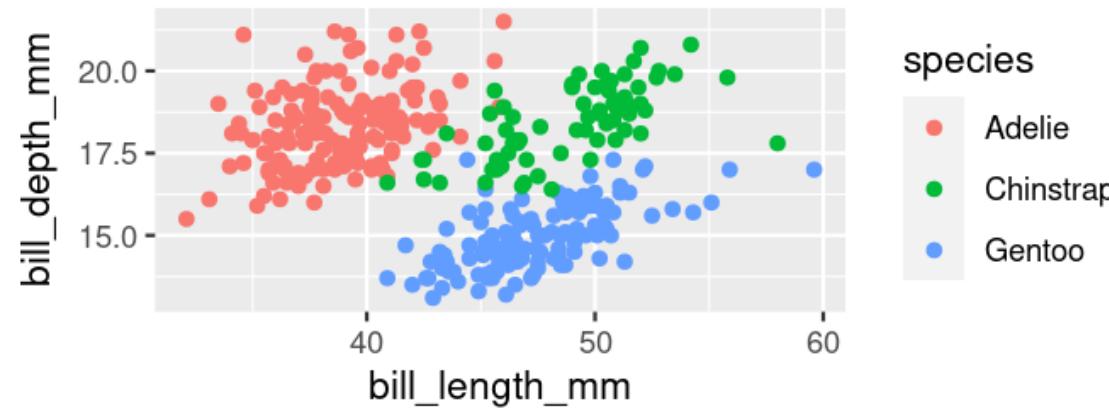
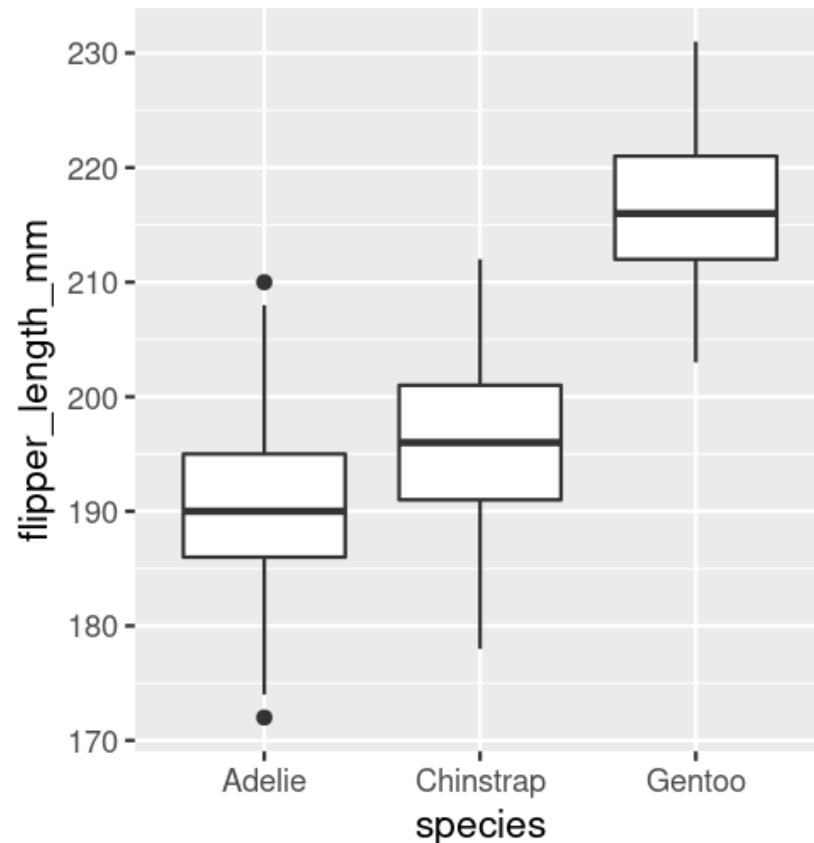
g1 / g2



Combining plots with patchwork

More complex arrangements

```
g2 + (g1 / g3)
```



species

- Adelie
- Chinstrap
- Gentoo

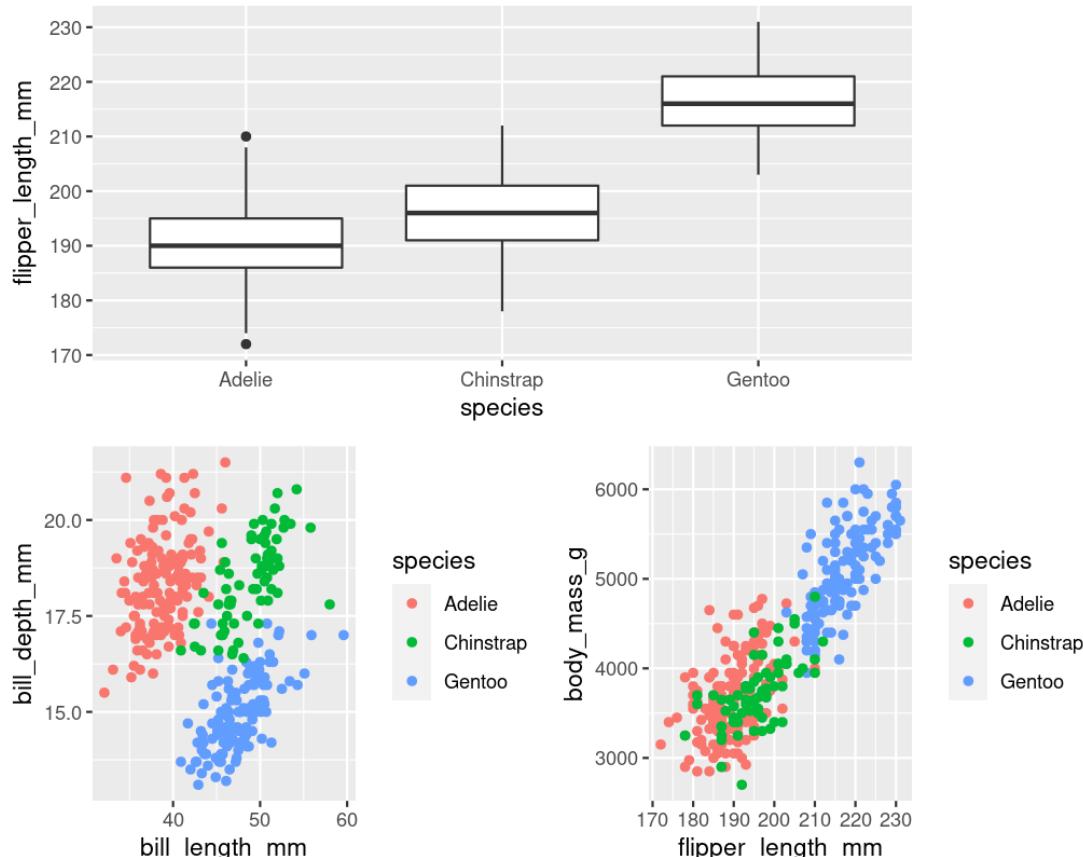
species

- Adelie
- Chinstrap
- Gentoo

Combining plots with patchwork

More complex arrangements

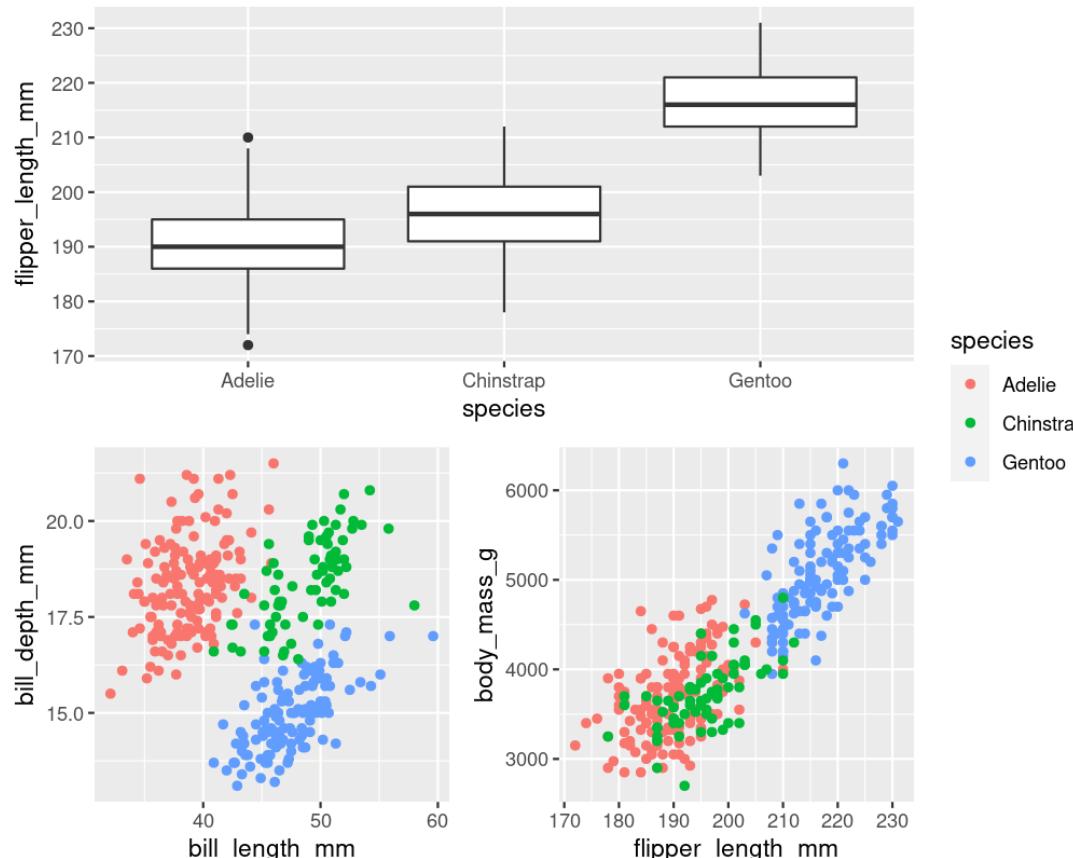
```
g2 / (g1 + g3)
```



Combining plots with patchwork

"collect" common legends

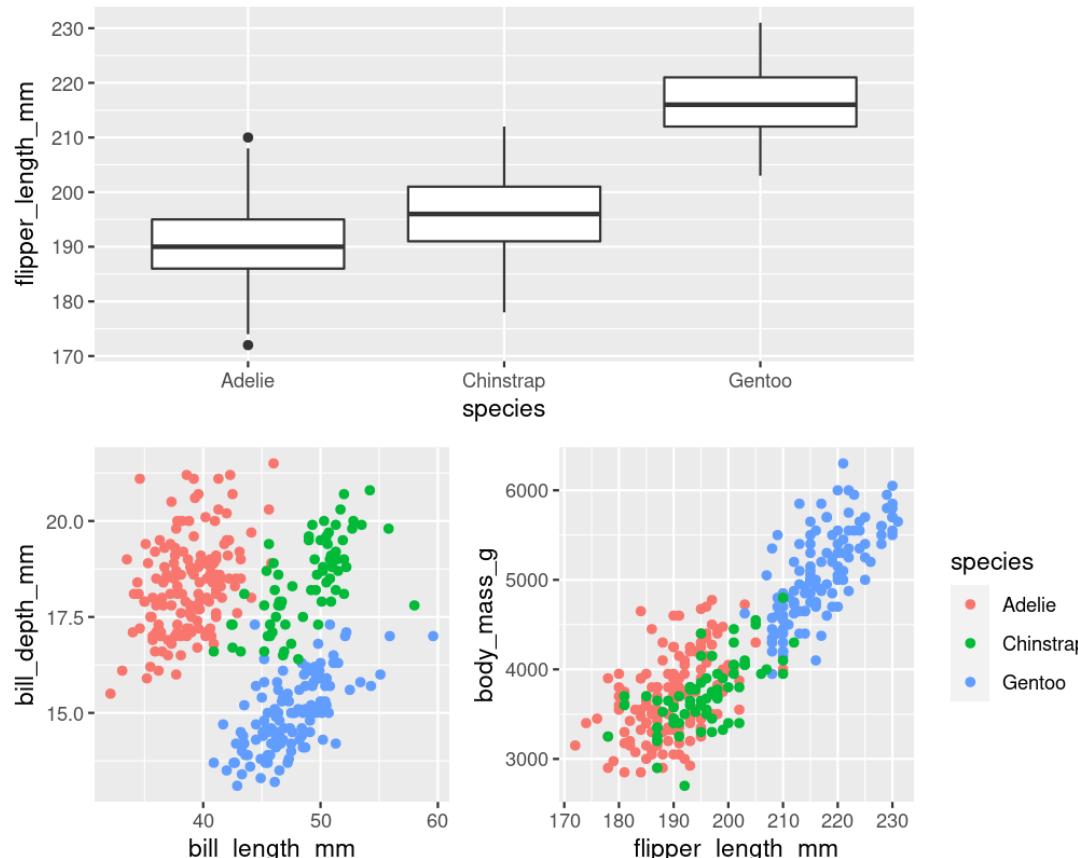
```
g2 / (g1 + g3) + plot_layout(guides = "collect")
```



Combining plots with patchwork

"collect" common legends

```
g2 / (g1 + g3 + plot_layout(guides = "collect"))
```



Combining plots with patchwork

Annotate

```
g2 / (g1 + g3) +
  plot_layout(guides = "collect") +
  plot_annotation(title = "Penguins Data Summary",
                  caption = "Fig 1. Penguins Data Summary",
                  tag_levels = "A",
                  tag_suffix = ")")
```

Penguins Data Summary

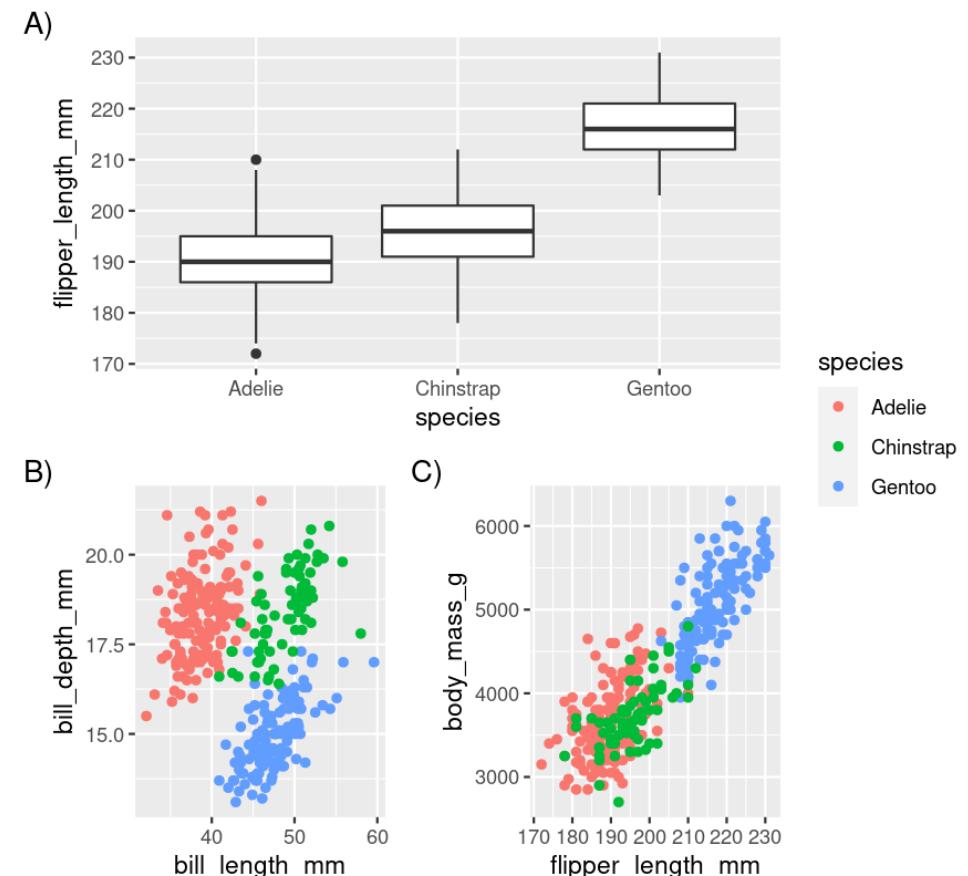


Fig 1. Penguins Data Summary

Saving plots

Saving plots

RStudio Export

Demo

Saving plots

RStudio Export

Demo

ggsave()

```
g <- ggplot(penguins, aes(x = sex, y = bill_length_mm, fill = year)) +  
  geom_boxplot()  
  
ggsave(filename = "penguins_mass.png", plot = g)
```

```
## Saving 8 x 3.6 in image
```

Saving plots

Publication quality plots

- Many publications require 'lossless' (pdf, svg, eps, ps) or high quality formats (tiff, png)
- Specific sizes corresponding to columns widths
- Minimum resolutions

```
g <- ggplot(penguins, aes(x = sex, y = body_mass_g)) +  
  geom_boxplot() +  
  labs(x = "Sex", y = "Body Mass (g)") +  
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  
  
ggsave(filename = "penguins_mass.pdf", plot = g, dpi = 300,  
       height = 80, width = 129, units = "mm")
```

Wrapping up: Common mistakes

- The **package** is `ggplot2`, the function is just `ggplot()`
- Did you remember to put the `+` at the **end** of the line?
- Order matters! If you're using custom `theme()`'s, make sure you put these lines **after** bundled themes like `theme_bw()`, or they will be overwritten
- Variables like 'year' are treated as continuous, but are really categories
 - Wrap them in `factor()`, i.e. `ggplot(data = penguins, aes(x = factor(year), y = body_mass_g))`

Wrapping up: Further reading (all Free!)

- RStudio > Help > Cheatsheets > Data Visualization with ggplot2
- [ggplot2 book v3](#)
 - By Hadley Wickham, Danielle Navarro, and Thomas Lin Pedersen
- [Cookbook for R](#) - by Winston Chang
 - See also R Graphics Cookbook by Winston Chang
- [R for Data Science](#)
 - [Data Visualization](#)

Your Turn!

Create a figure with...

- Custom colour mapping (i.e. `scales_....`)
- Clear, human-readable labels
- More than one graph, each one tagged (e.g., A) or B))
- With more than one geom type
- At least one scatterplot with regression line

:D

OR... Load your own data and create a figure of your own!