Wrapping up: Further reading (all Free!)

- RStudio > Help > Cheatsheets > Data Visualization with ggplot2
- ggplot2 book v3 (Hadley Wickham, Danielle Navarro, and Thomas Lin Pedersen)
- `patchwork` website
- Cookbook for R (Winston Chang)
- R for Data Science (Hadley Wickham & Garrett Grolemund)
    - Chp 3 Data Visualization
- Data Visualization: A practical introduction (Kieran Healy)

Creating Figures as an Intro to R

Using the `ggplot2` package

steffilazerte
@steffilazerte@fosstodon.org
@steffilazerte
steffilazerte.ca

Preamble

Online workshops can be challenging

Consider keeping your video on (if possible)

- Kids? Pets? Spouses? No problem
- But ultimately, you need be comfortable! (and you absolutely have the right to privacy)

Interrupt me!

- Generally keep yourself muted but un-mute anytime to ask questions

Ask Questions!

- Group trouble-shooting is really valuable
- If you have a problem, others may also (or may have it in the future)

Screen-sharing

- I may ask you to share your screen with the group (feel free to decline)
- For privacy, close your email etc. Or just share your RStudio window

This is me and my creatures

This is my garden

What about you?

- Name
- Background (Role, Area of study, etc.)
- Familiarity with R or Programming
- Creatures (furry, feathery, scaley, green or otherwise)?

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 a 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?

```r
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?

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

```
[1] 2.5
```

R, save this value for later

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

R, multiply this value by 6

```r
steffis_mean * 6
```

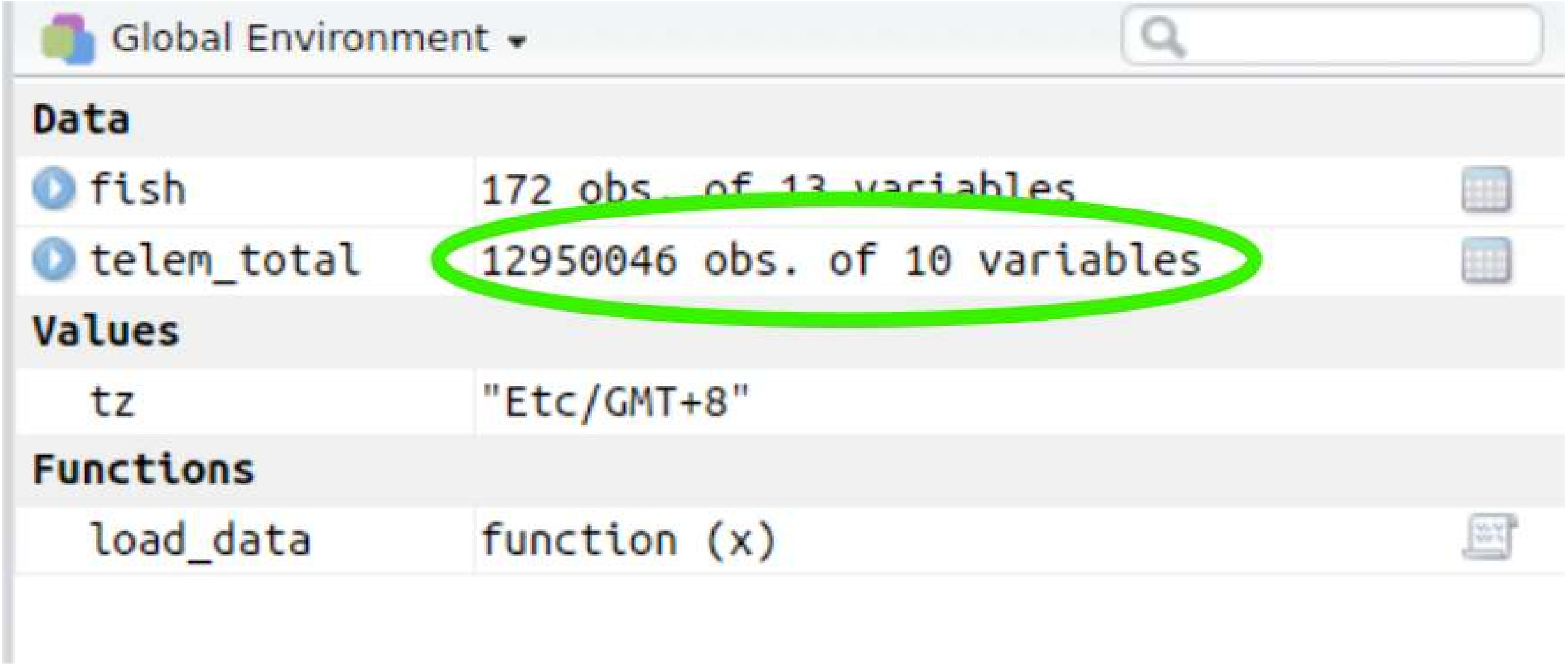
```
[1] 15
```

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],
```

But R is powerful (and reproducible)!



Data

```
fish          172 obs. of 13 variables
telem_total   12950046 obs. of 10 variables
```

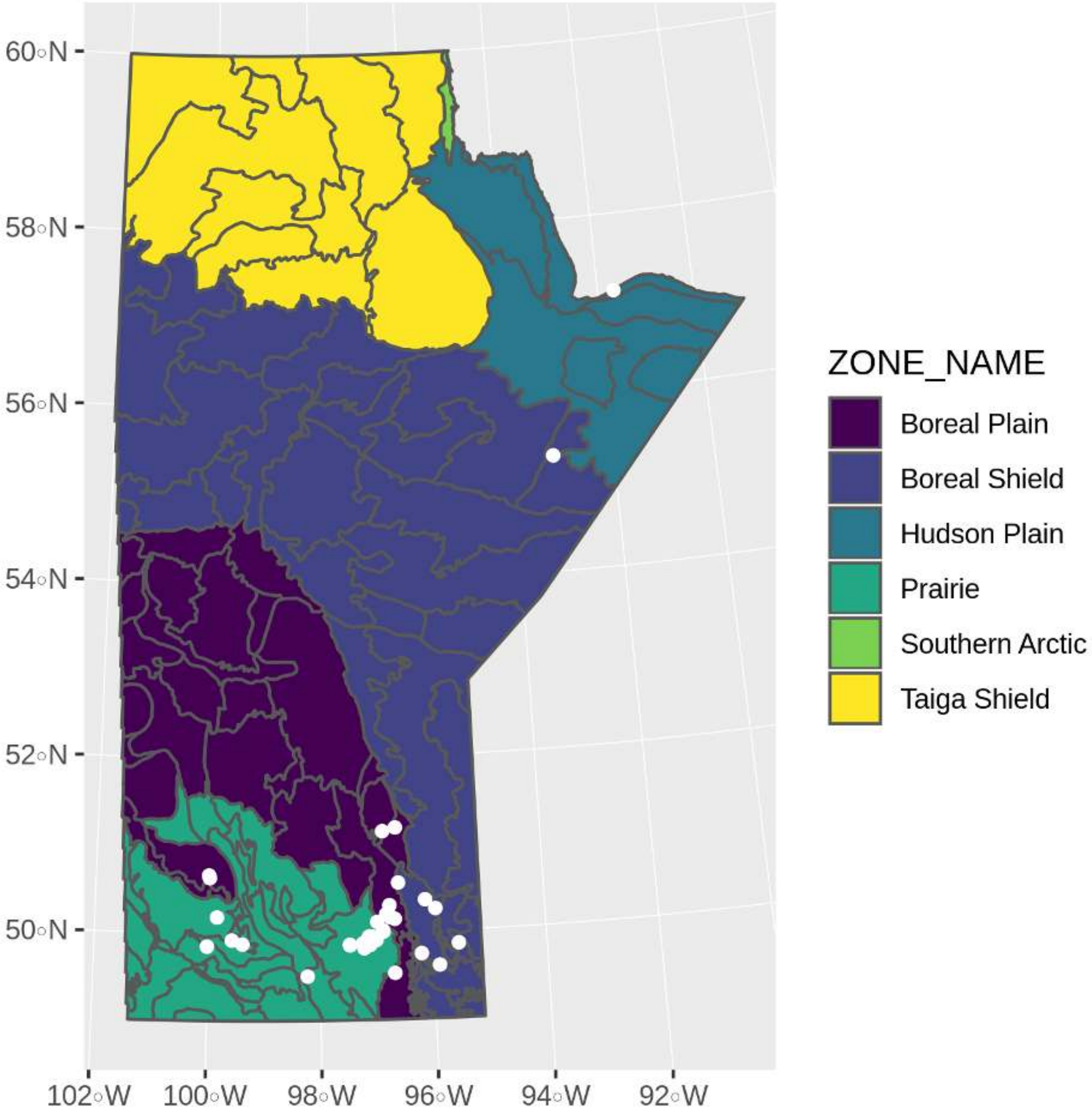
Values

```
tz            "Etc/GMT+8"
```

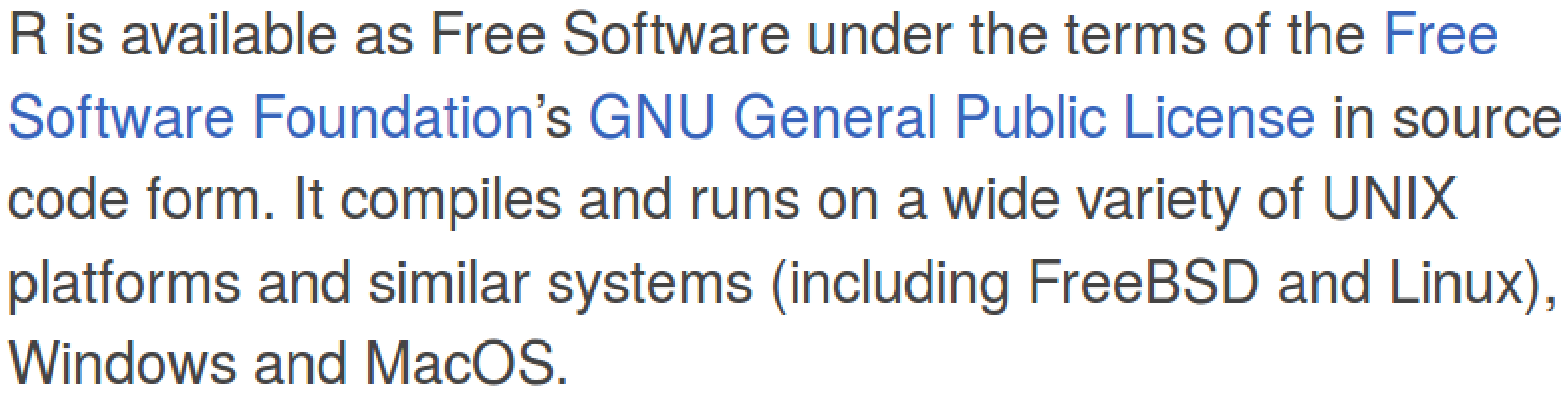
Functions

```
load_data     function (x)
```

R is also beautiful



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



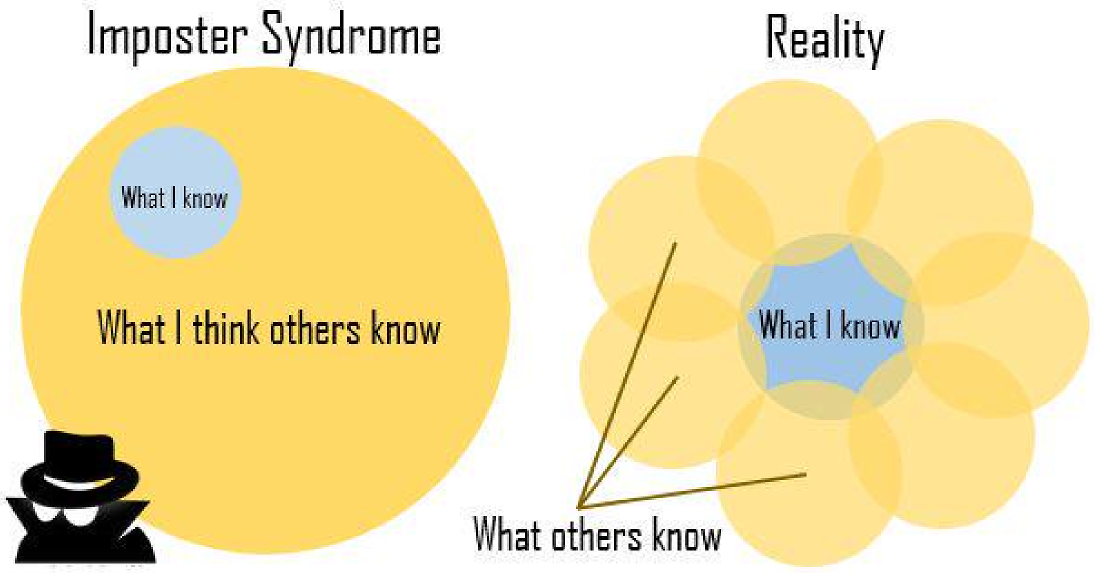
R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

ImpostR Syndrome

ImpostR
Syndrome

ImpostR Syndrome

Impost R
Syndrome



Moral of the story?
Make friends, code in groups, learn together and don’t beat yourself up

The Goal

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

For example:

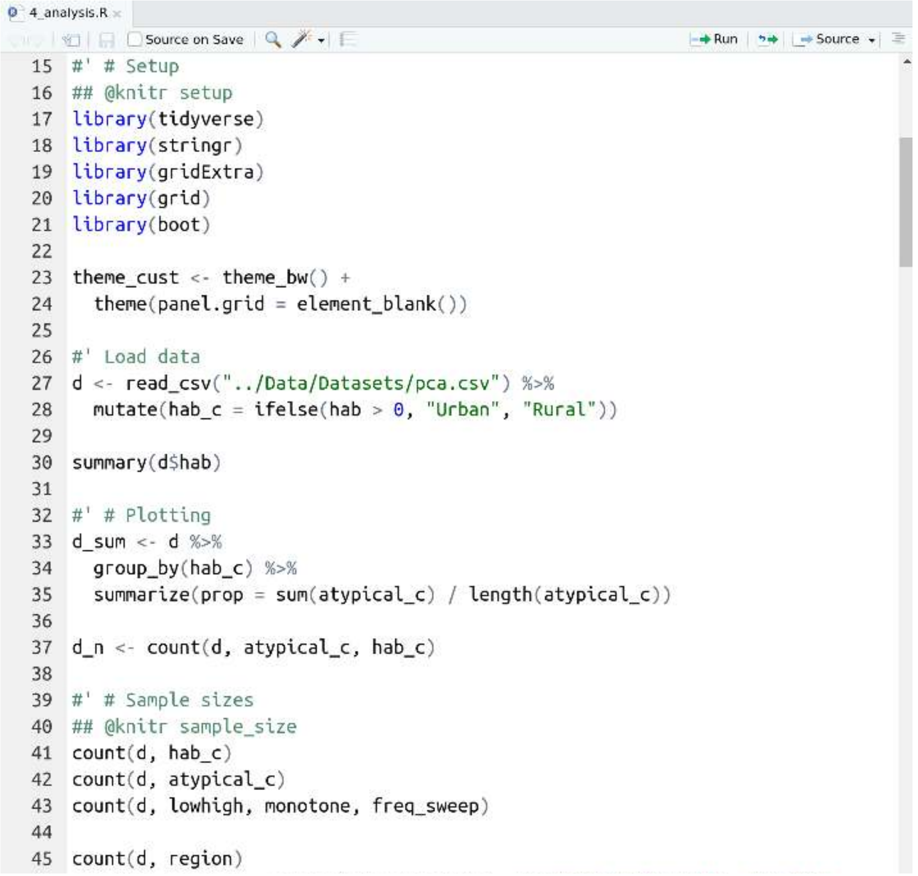
```
mean(c(1, 2, 3, 4))
```

Code

```
[1] 2.5
```

Output

Script



```
#' # Setup
## @knitr setup
library(tidyverse)
library(stringr)
library(gridExtra)
library(grid)
library(boot)

theme_cust <- theme_bw() +
  theme(panel.grid = element_blank())

#' Load data
d <- read_csv("../Data/Datasets/pca.csv") %>%
  mutate(hab_c = ifelse(hab > 0, "Urban", "Rural"))

summary(d$hab)

#' # Plotting
d_sum <- d %>%
  group_by(hab_c) %>%
  summarize(prop = sum(atypical_c) / length(atypical_c))

d_n <- count(d, atypical_c, hab_c)

#' # Sample sizes
## @knitr sample_size
count(d, hab_c)
count(d, atypical_c)
count(d, lowhigh, monotone, freq_sweep)

count(d, region)
```

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.

- 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
```

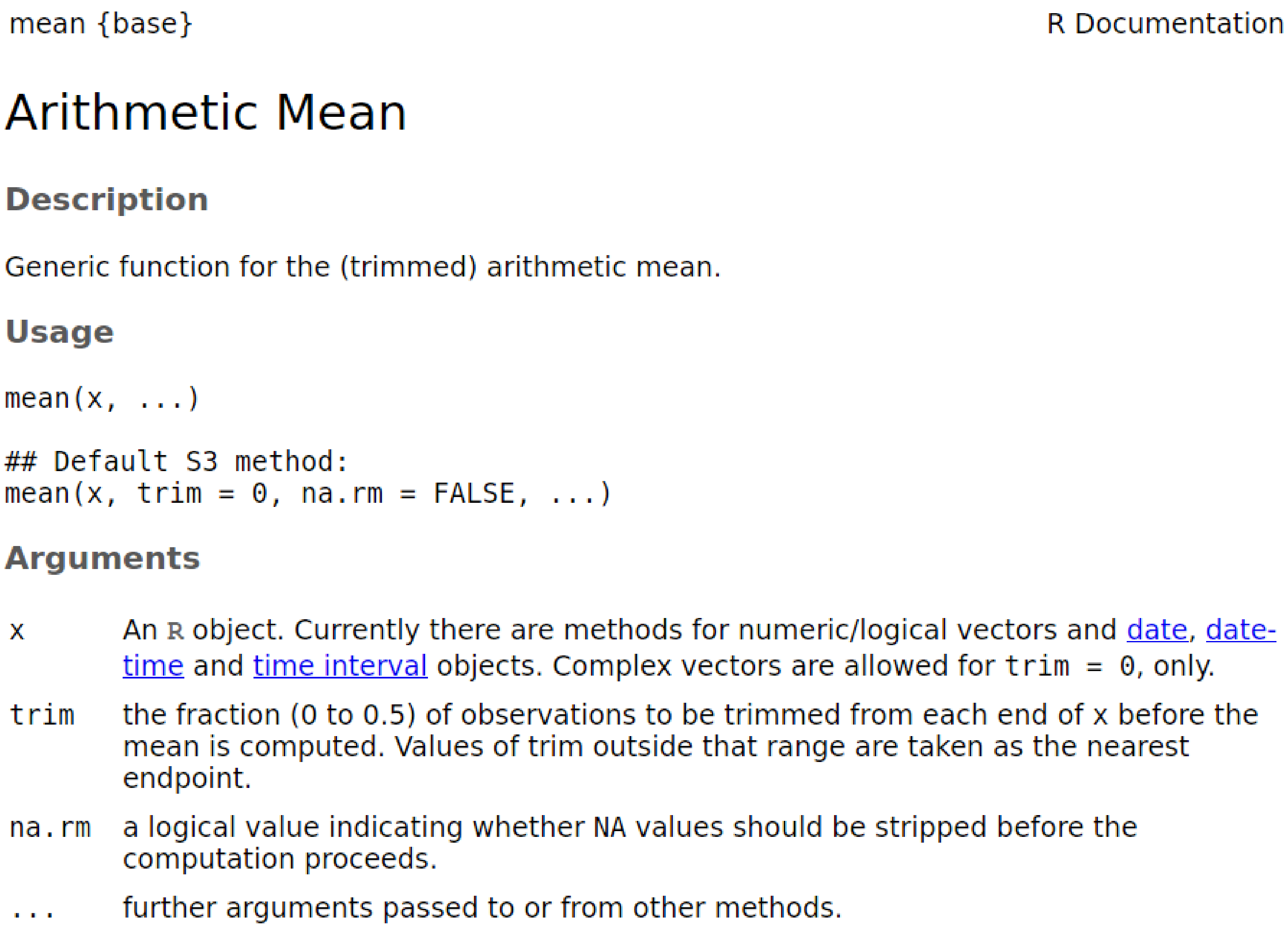
By position:

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

```
[1] 5.333333
```

R documentation

```r
?mean
```



mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```r
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)

```r
my_data <- c("a", 100, "c")
my_data
```

```
[1] "a"   "100" "c"
```

Data frame (2 dimensions)

```r
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

```r
# 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()
```

1. **Copy/paste or type** this into the **script** window in RStudio
    - You may have to go to File > New File > R Script

2. Click on the **first line of code**

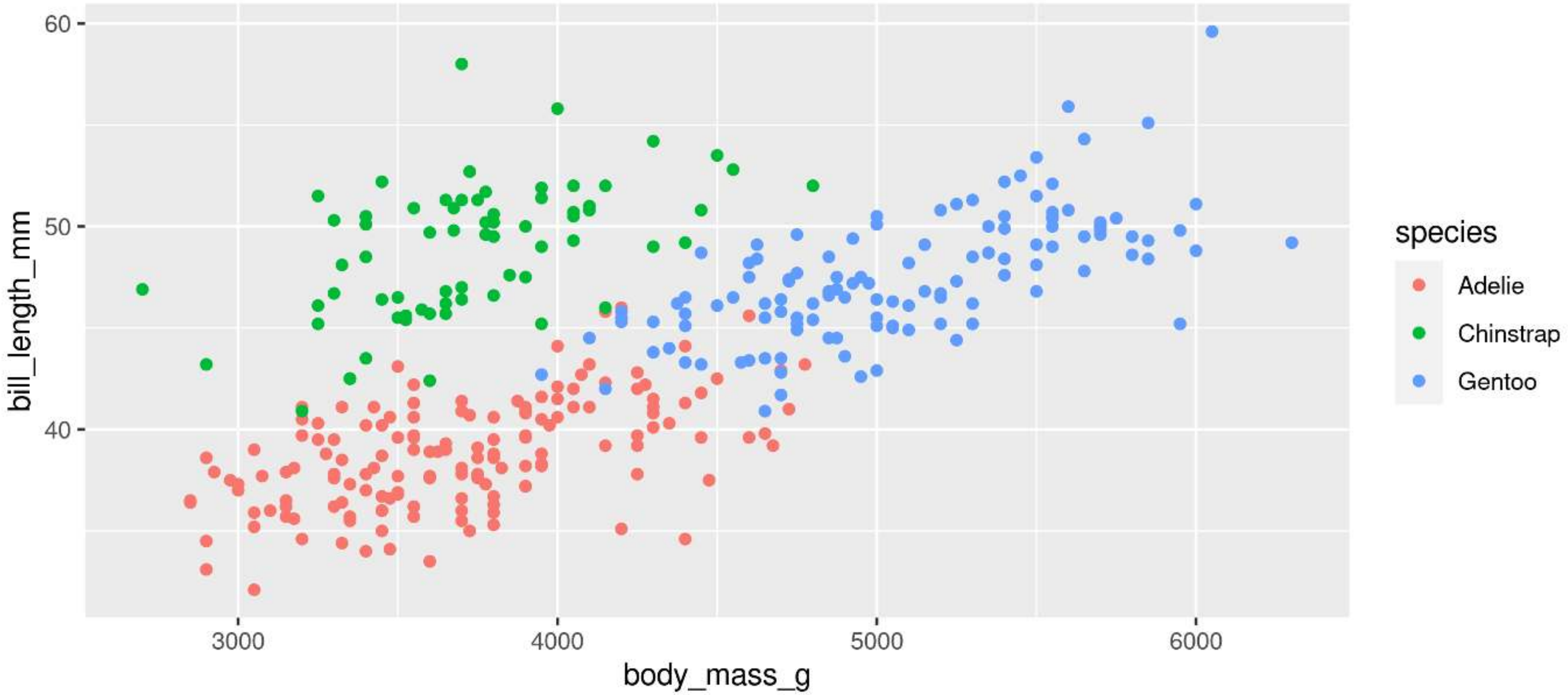
3. **Run the code**
    - Click ‘Run’ button (upper right) **or**
    - Use the short-cut `Ctrl-Enter`

4. **Repeat** until all the code has run

First Code

```r
# 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()`).
```

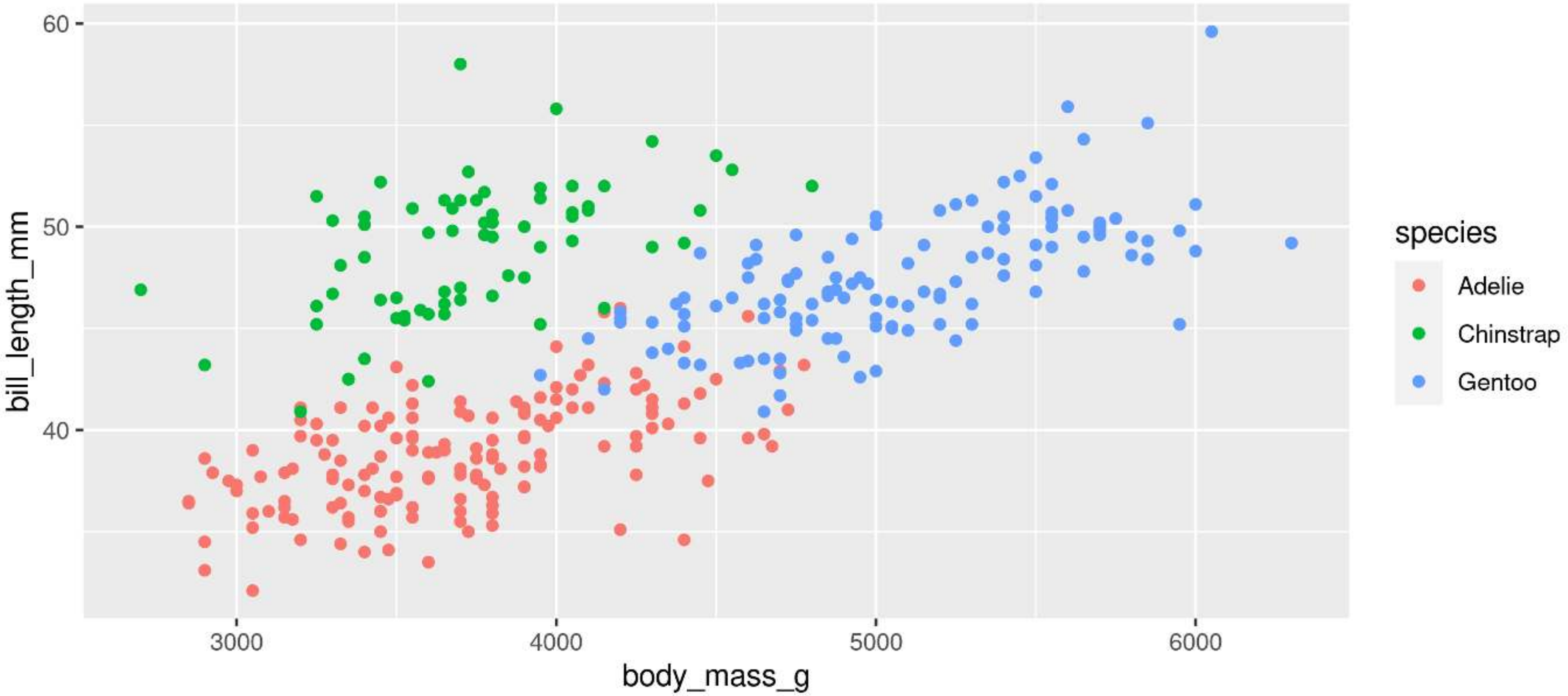


First Code

Packages
`ggplot2` and `palmerpenguins`

```r
# 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()`).
```



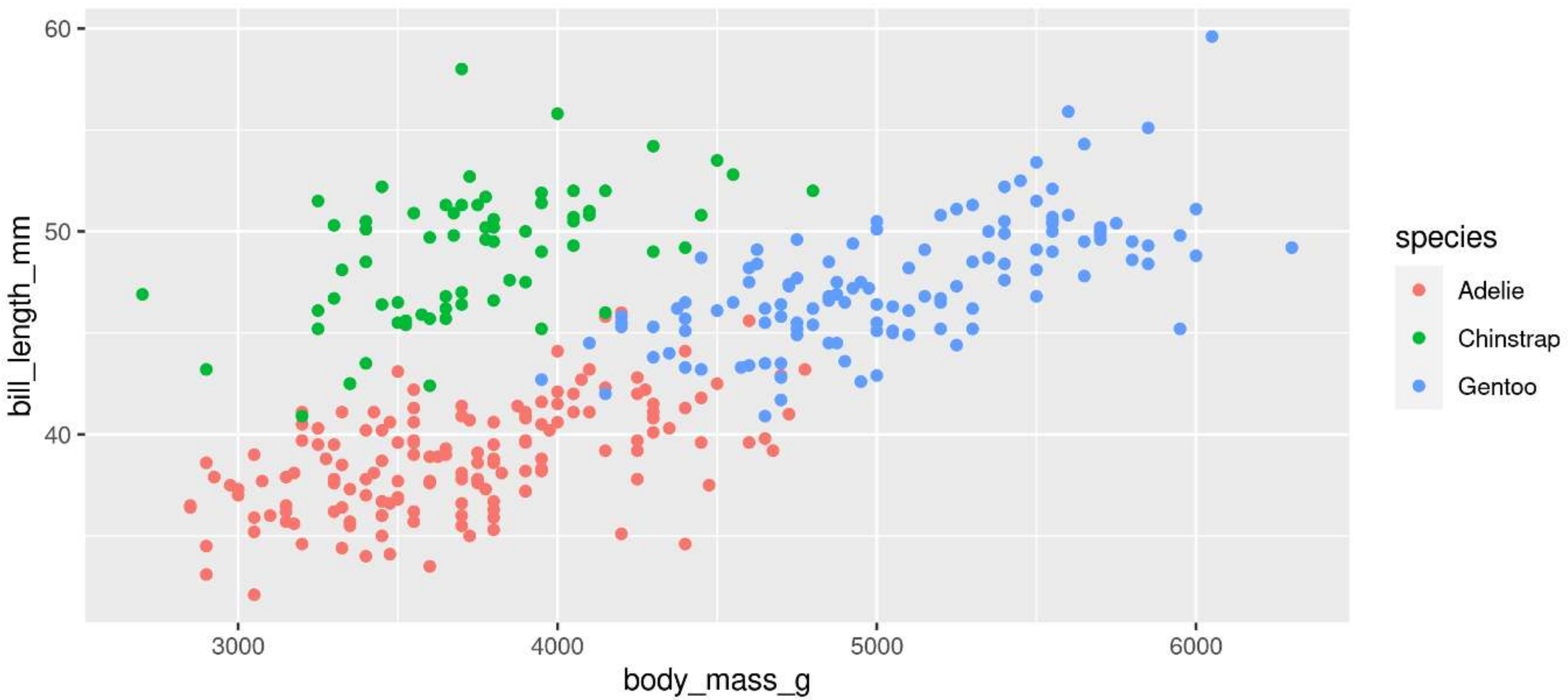
First Code

Functions

`library(),ggplot(),aes(),geom_point()`

```r
# 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()`).
```

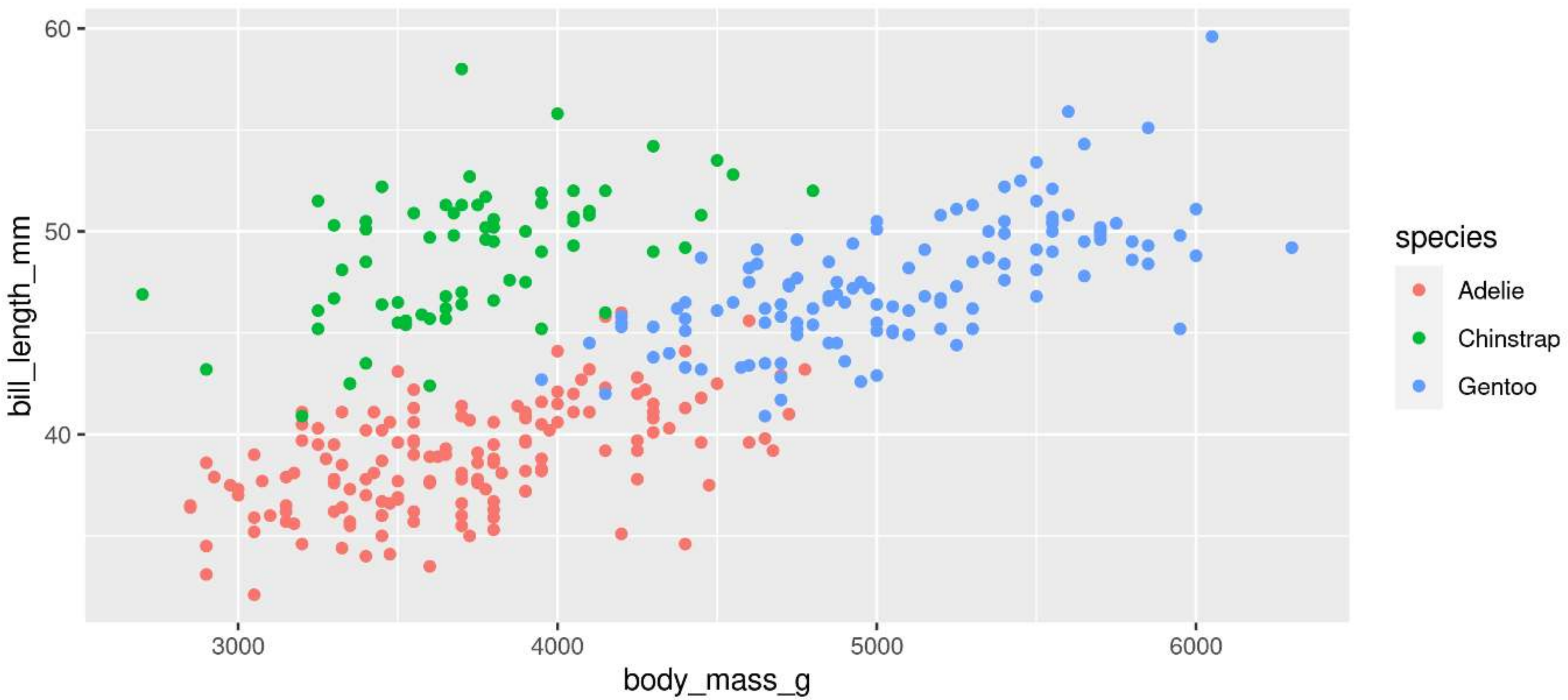


First Code

```r
# 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()
```

+
(Specific to `ggplot`)

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

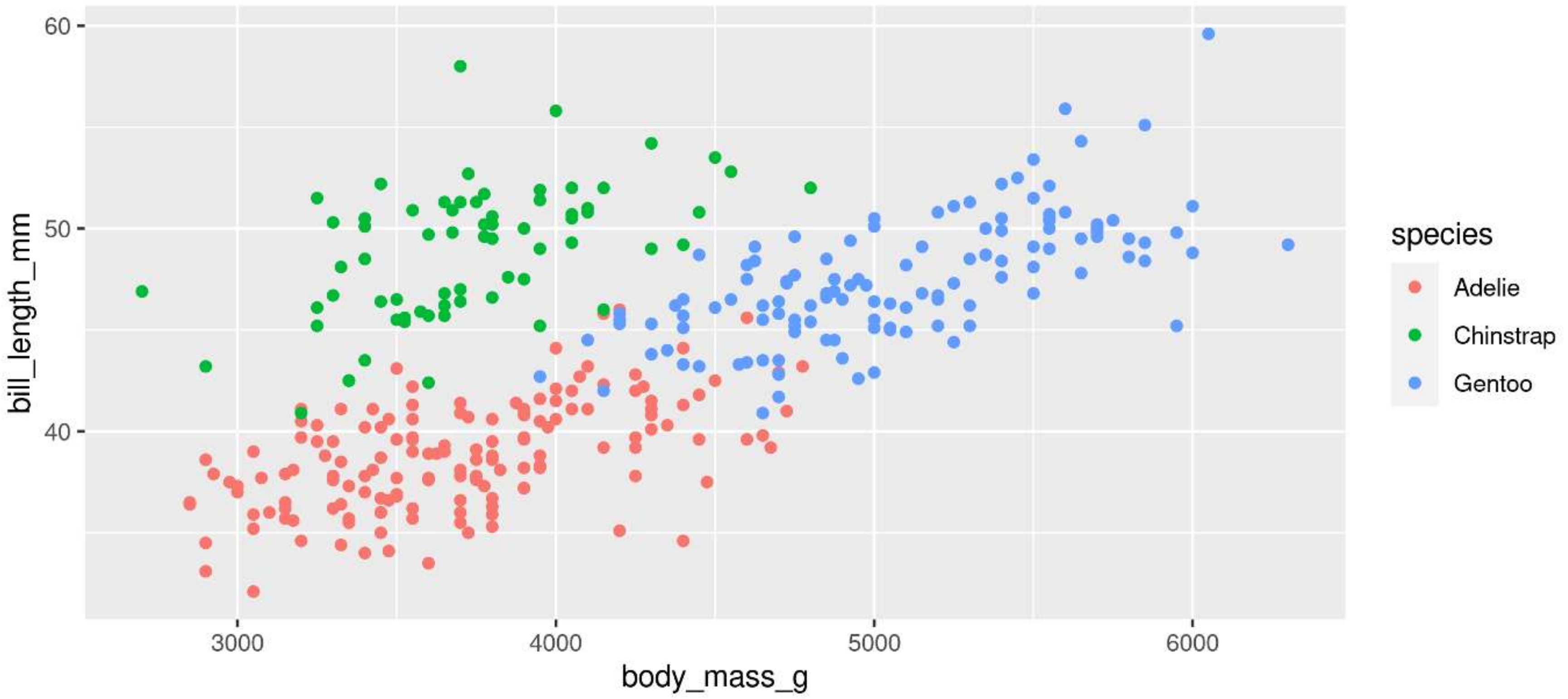


First Code

```r
# 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!

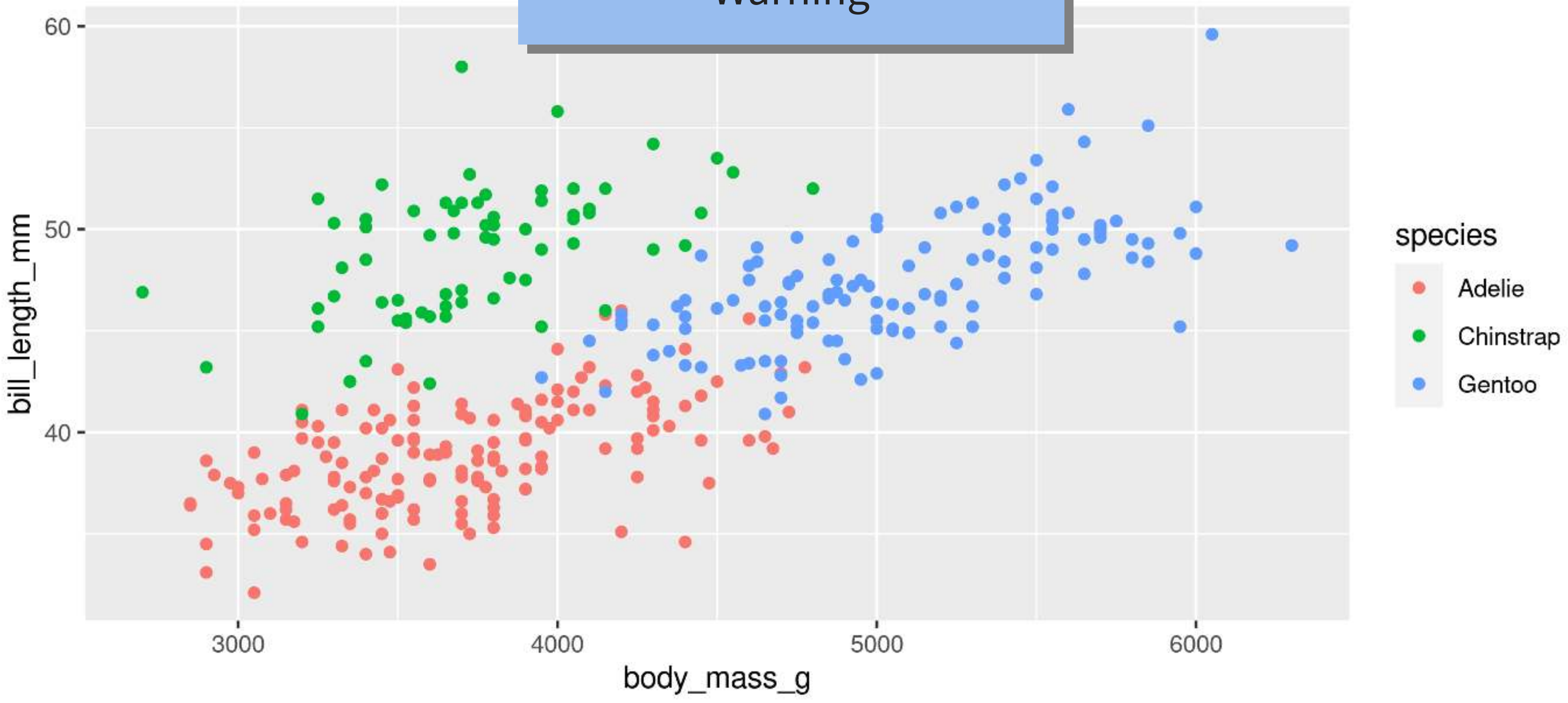


First Code

```r
# 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

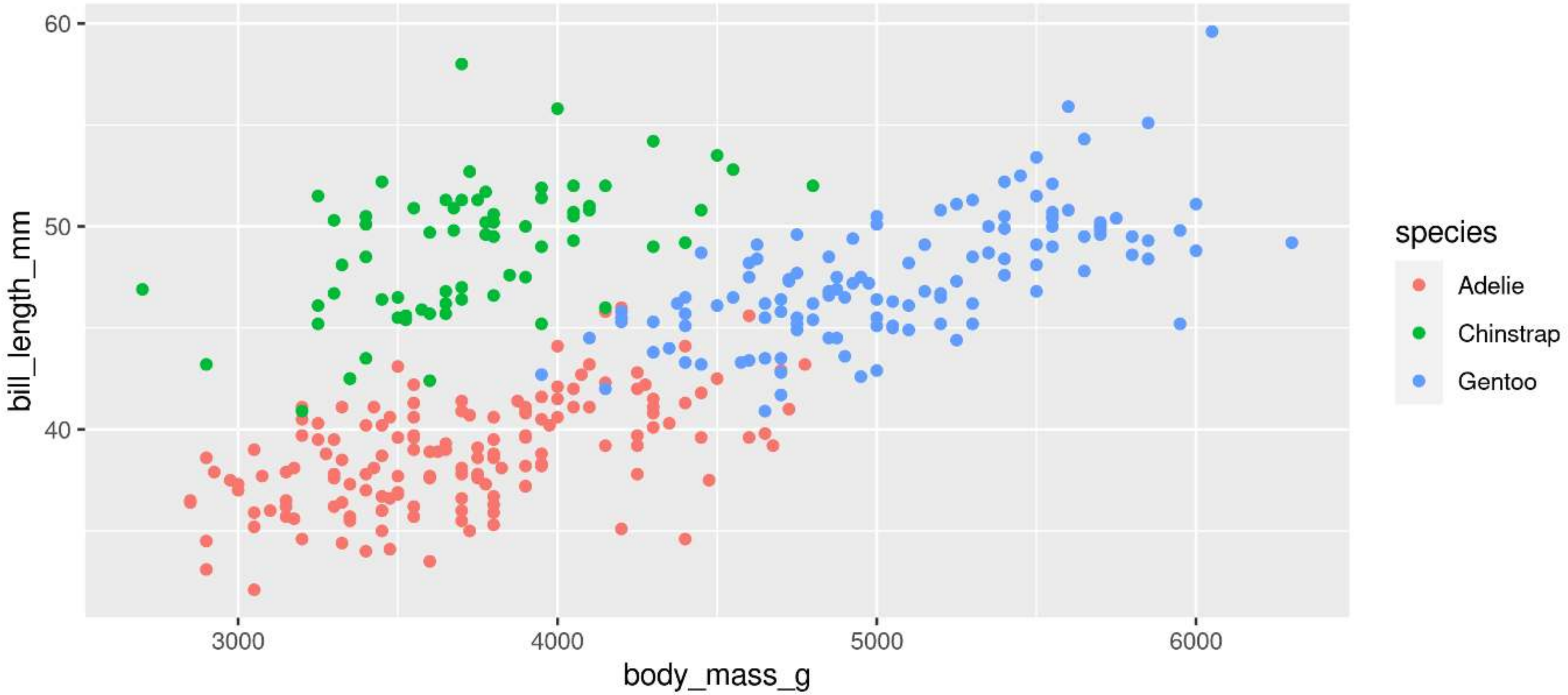


First Code

```r
# 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()
```

Comments

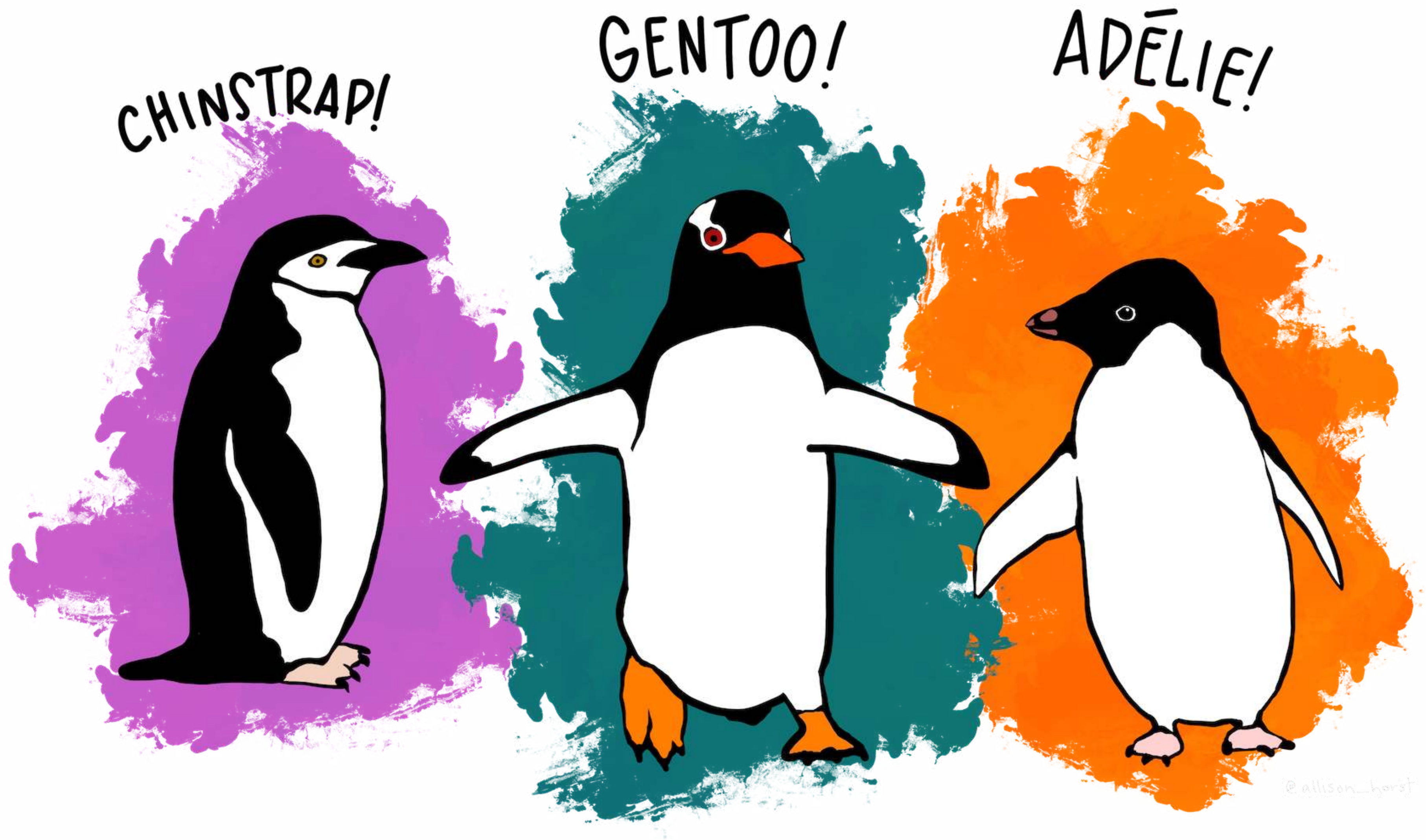
```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Now you know R!

Let’s get started

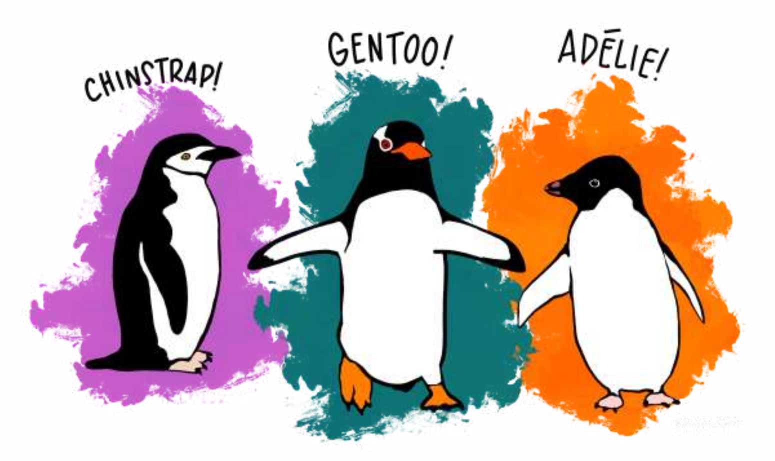
Our data set: Palmer Penguins!



Our data set: Palmer Penguins!

```r
library(palmerpenguins)
penguins
```

```
# A tibble: 344 × 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
# ℹ 334 more rows
```

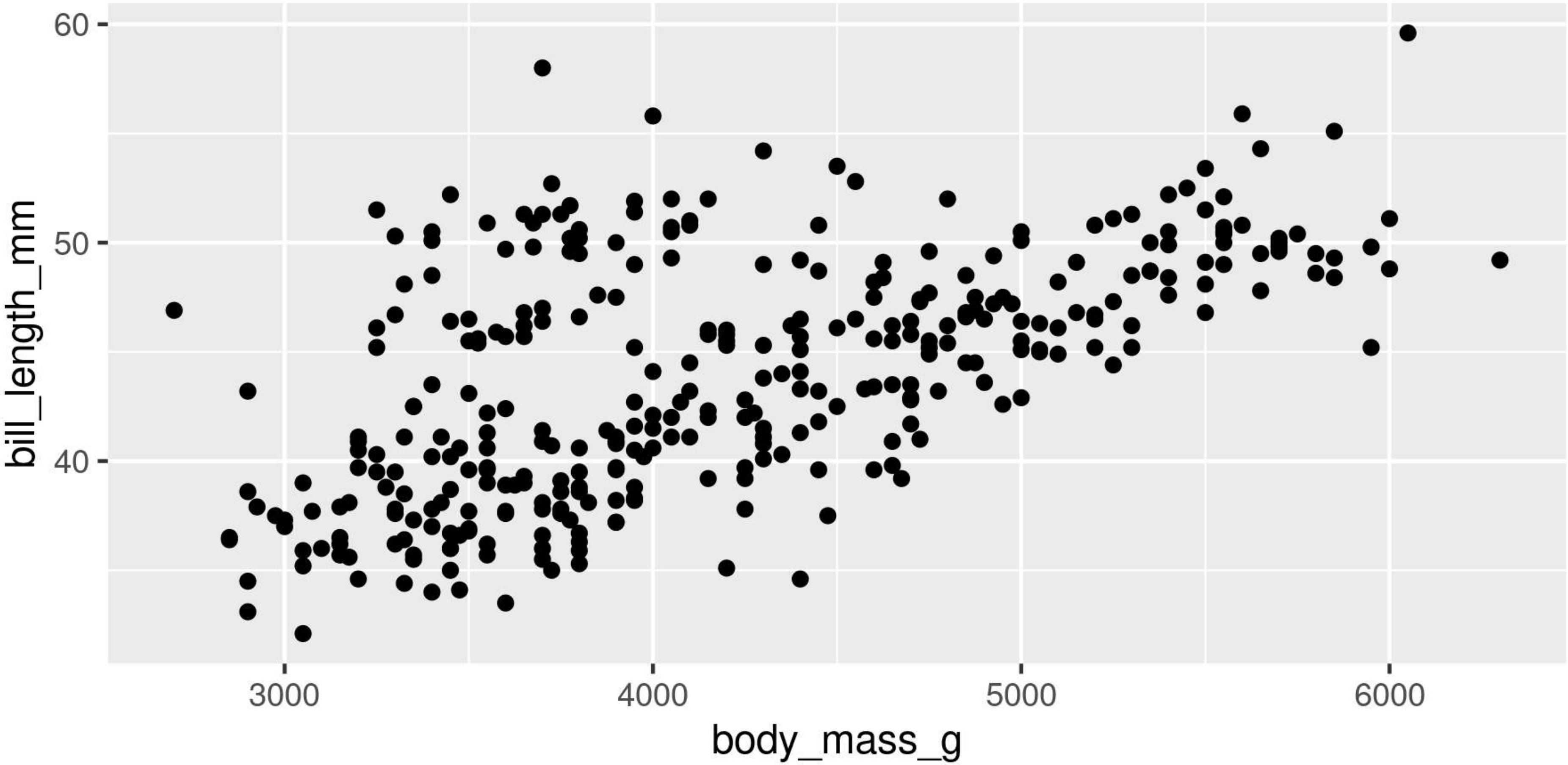


Your turn!

Run this code and look at the output in the console

A basic plot

```r
library(palmerpenguins)
library(ggplot2)

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

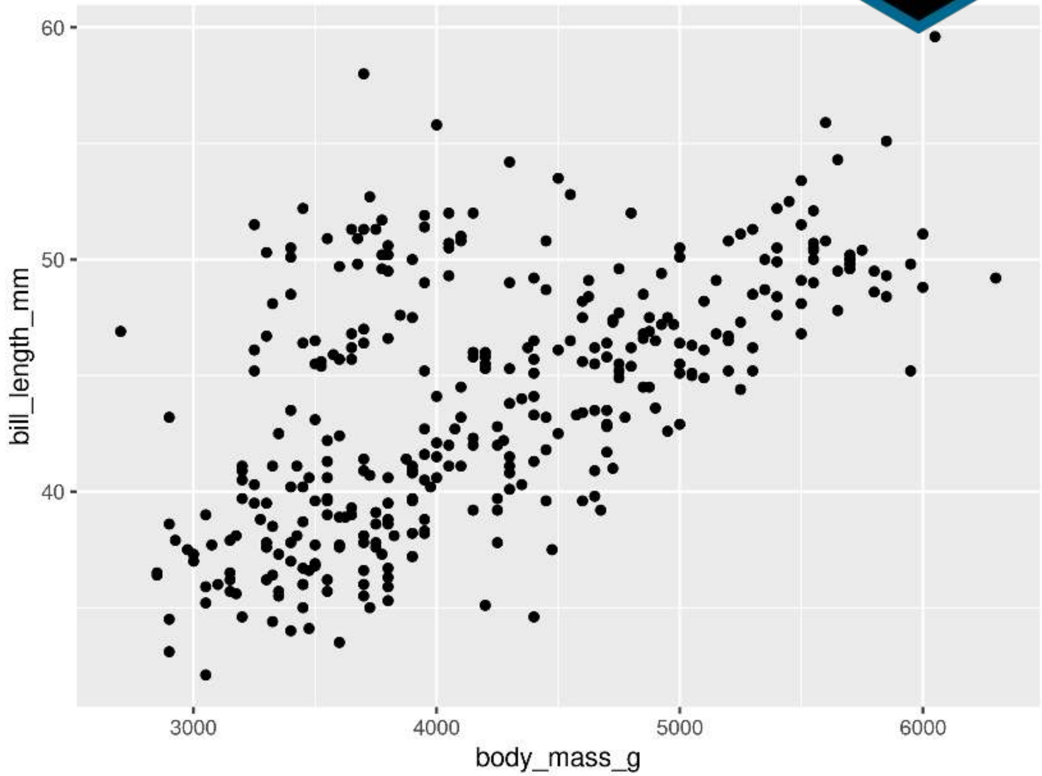


Break it down

```r
library(palmerpenguins)
library(ggplot2)

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

`library(palmerpenguins)`

- Load the `palmerguins` package
- Now we have access to `penguins` data

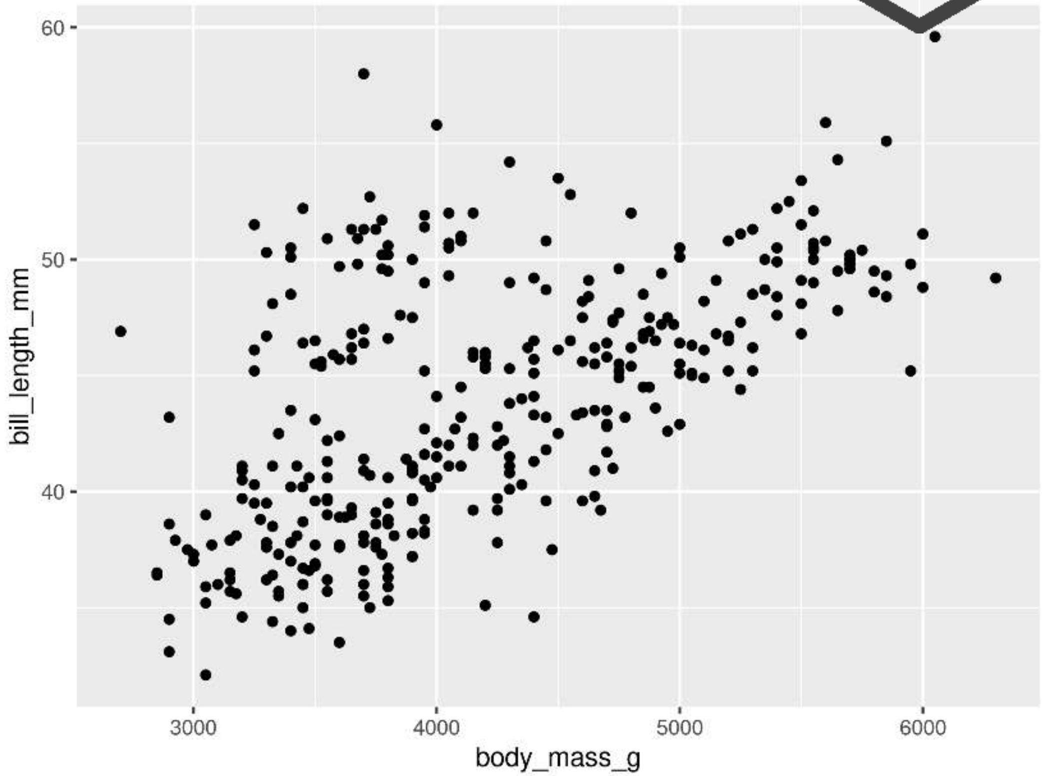


Break it down

```r
library(palmerpenguins)
library(ggplot2)

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

`library(ggplot2)`

- Load the `ggplot2` package
- Now we have access to the `ggplot()` function
    - (and `aes()` and `geom_point()` etc.)

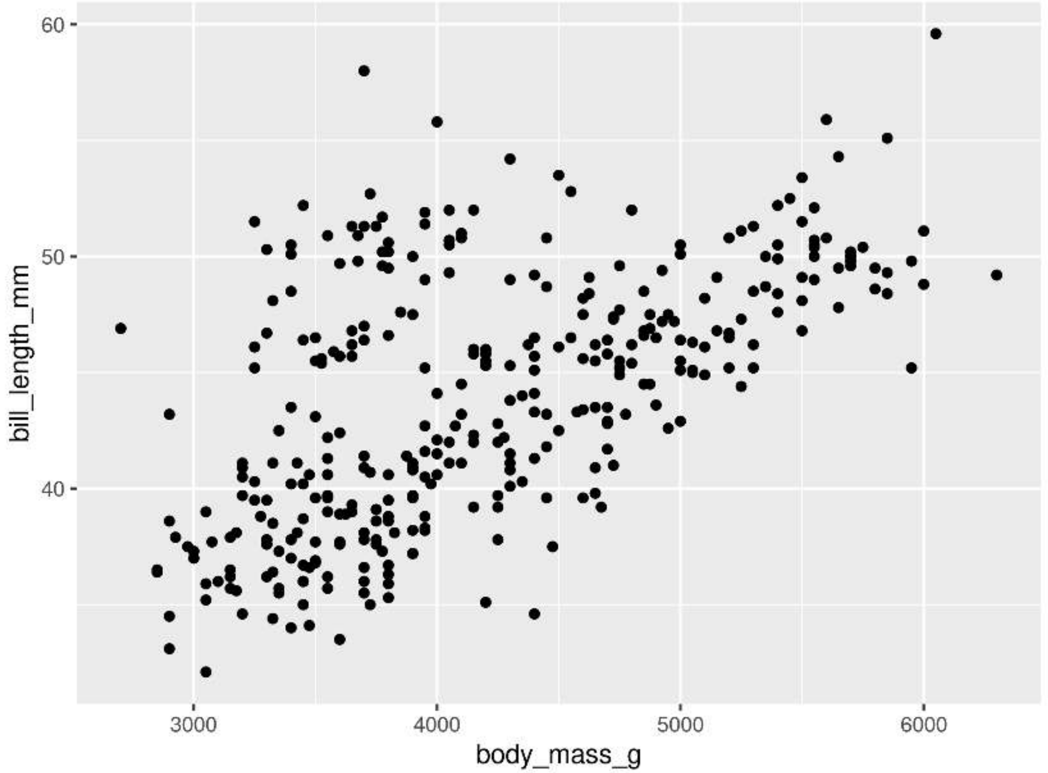


Break it down

```
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



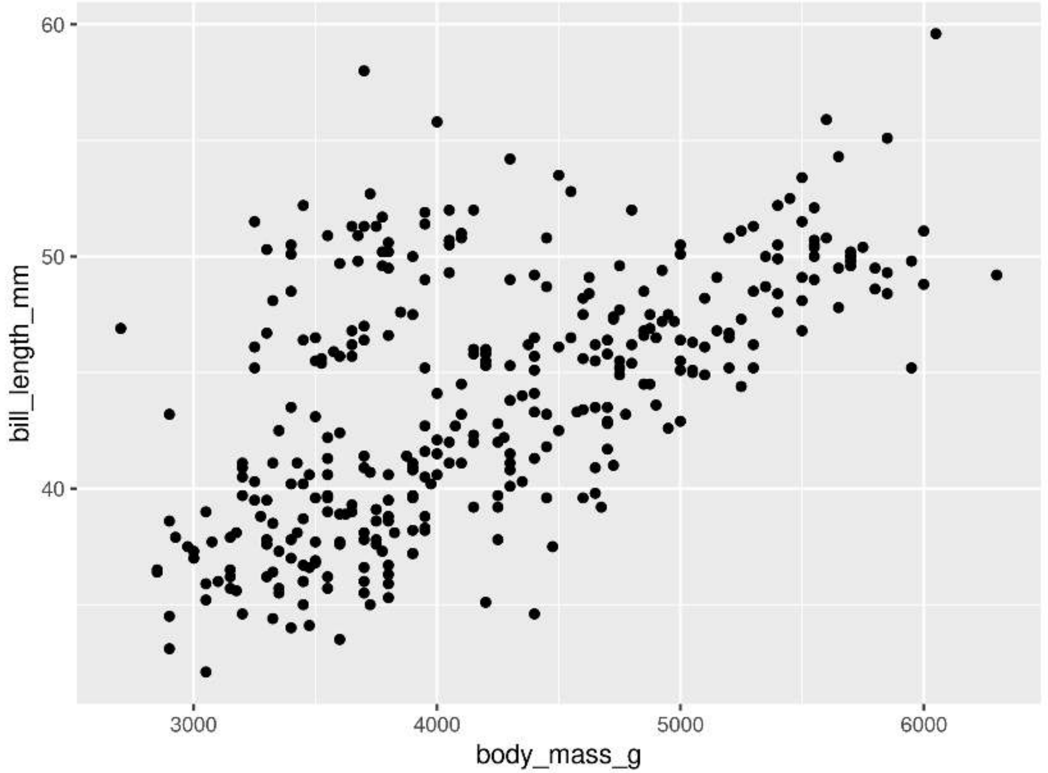
Break it down

```r
library(palmerpenguins)
library(ggplot2)

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

`geom_point()`

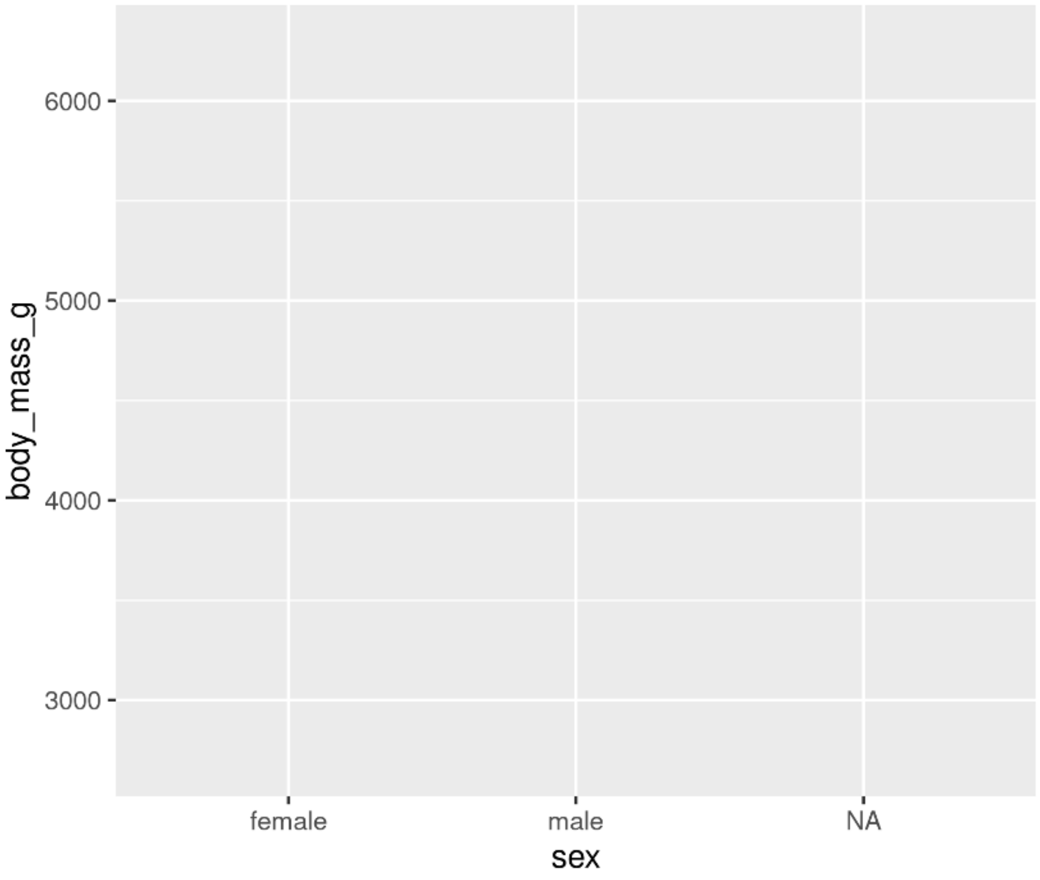
- Choose a `geom` function to display the data
- Always *added* to a `ggplot()` call with `+`

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

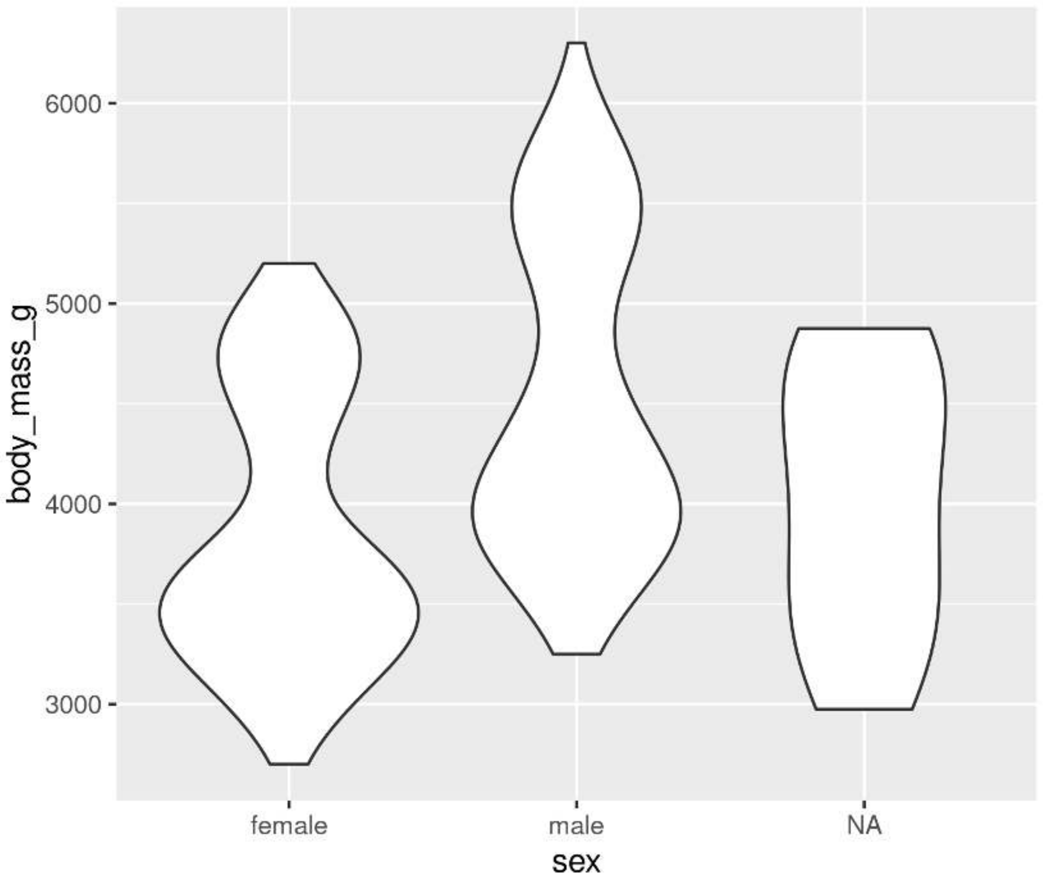


Plots are layered

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



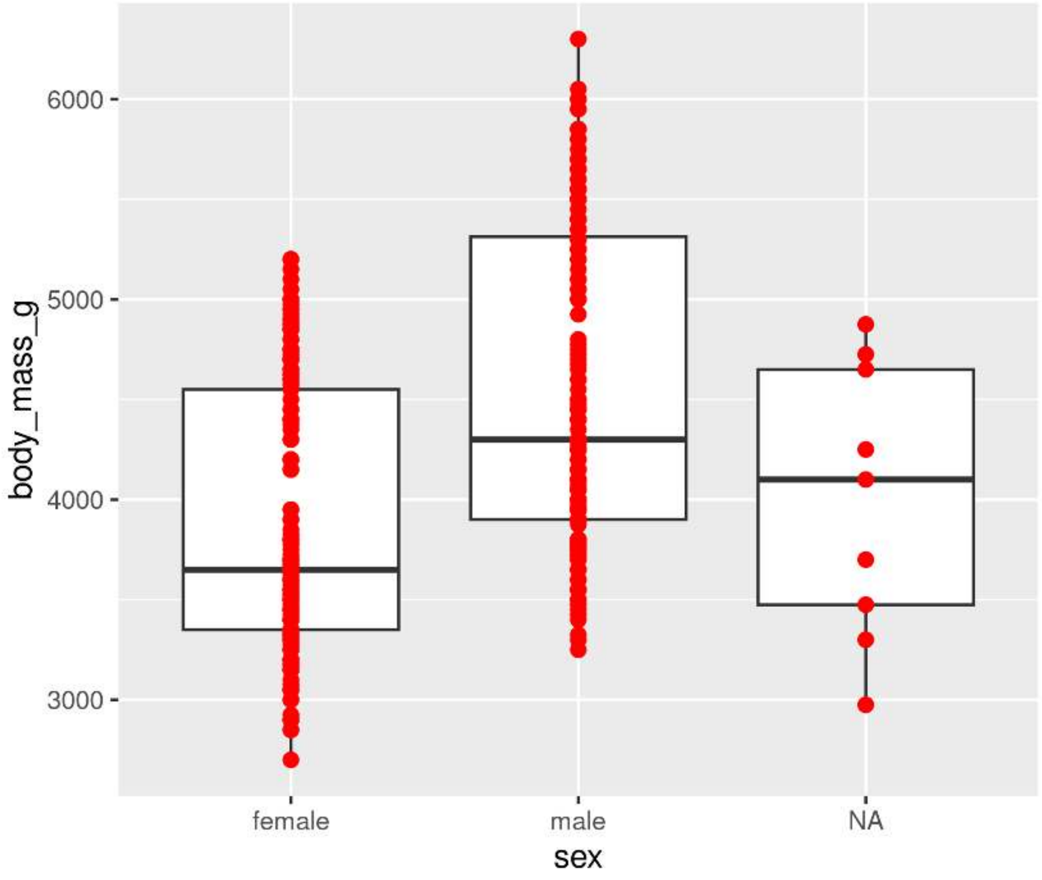
```r
ggplot(data = penguins, aes(x = sex, y = body_mass_g)) +
  geom_violin()
```



Plots are layered

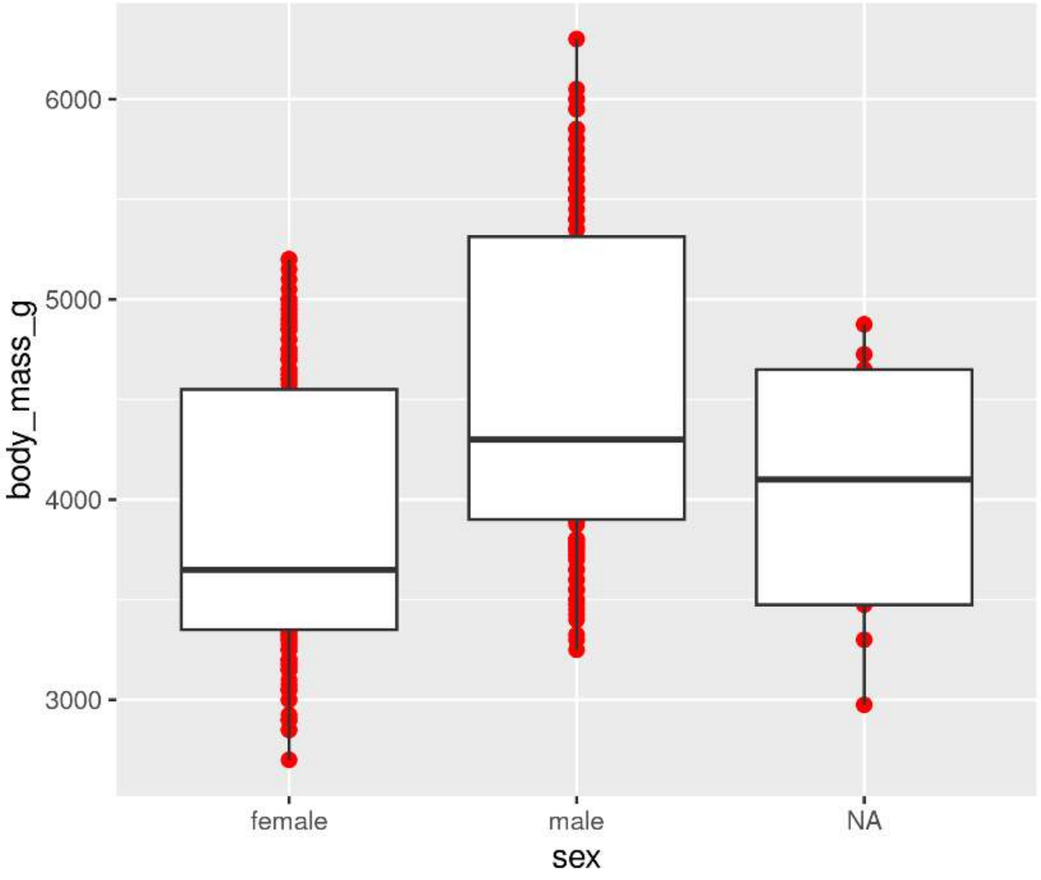
You can add multiple layers

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



Order matters

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

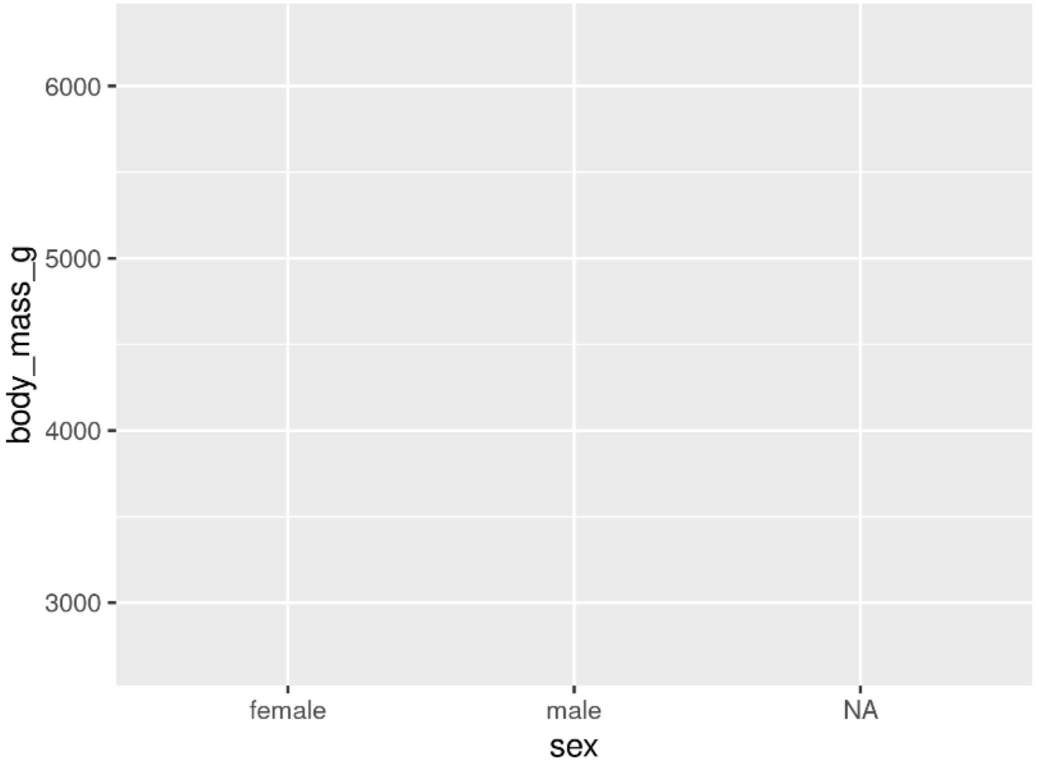


Plots are objects

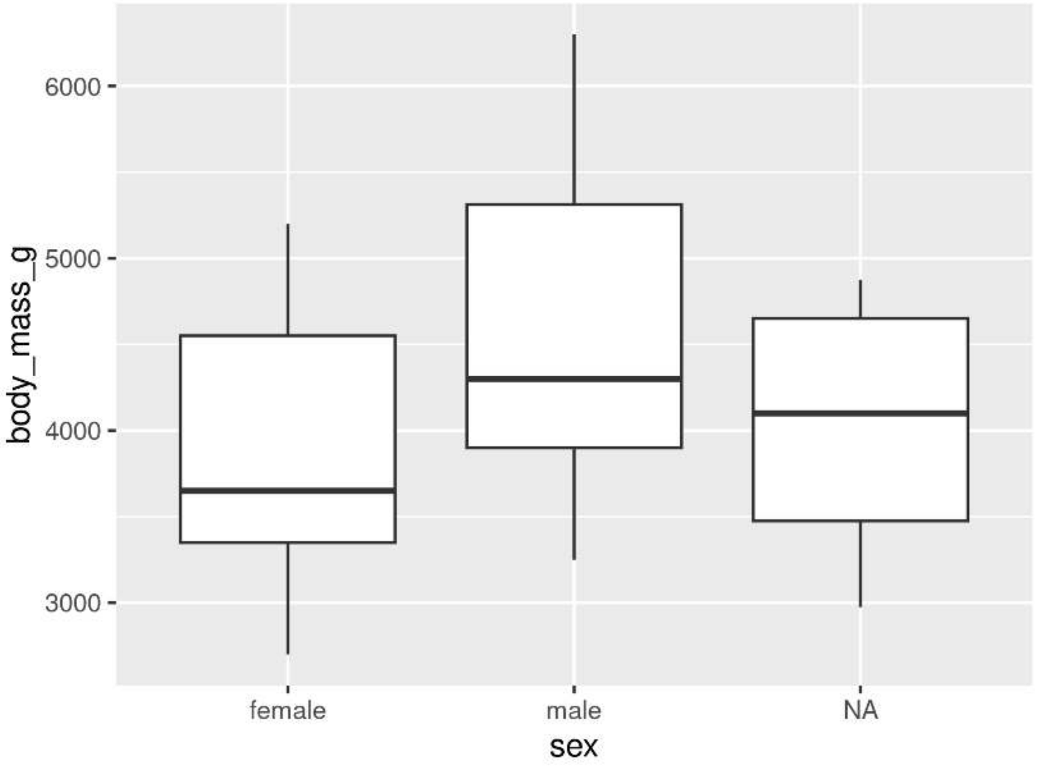
Any ggplot can be saved as an object

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

```r
g
```



```r
g + geom_boxplot()
```

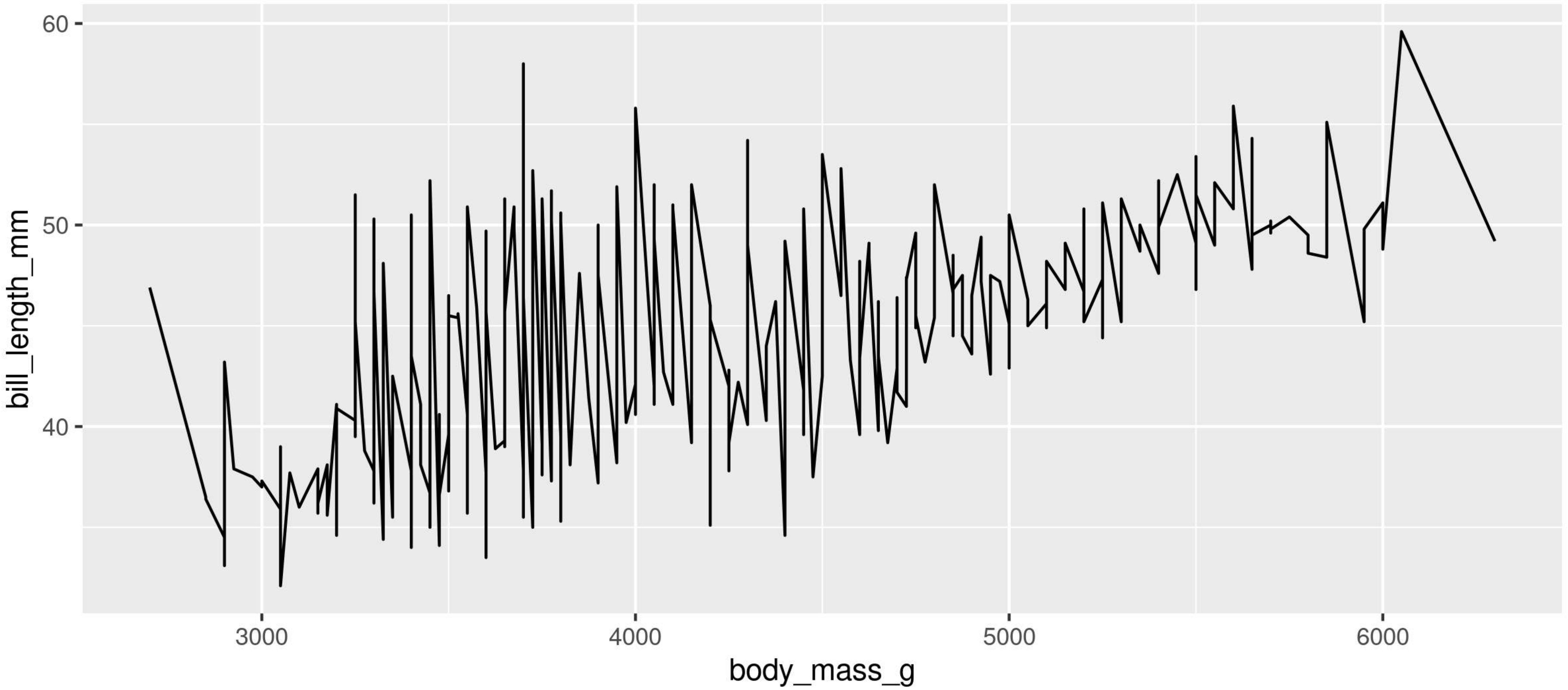


More Geoms

(Plot types)

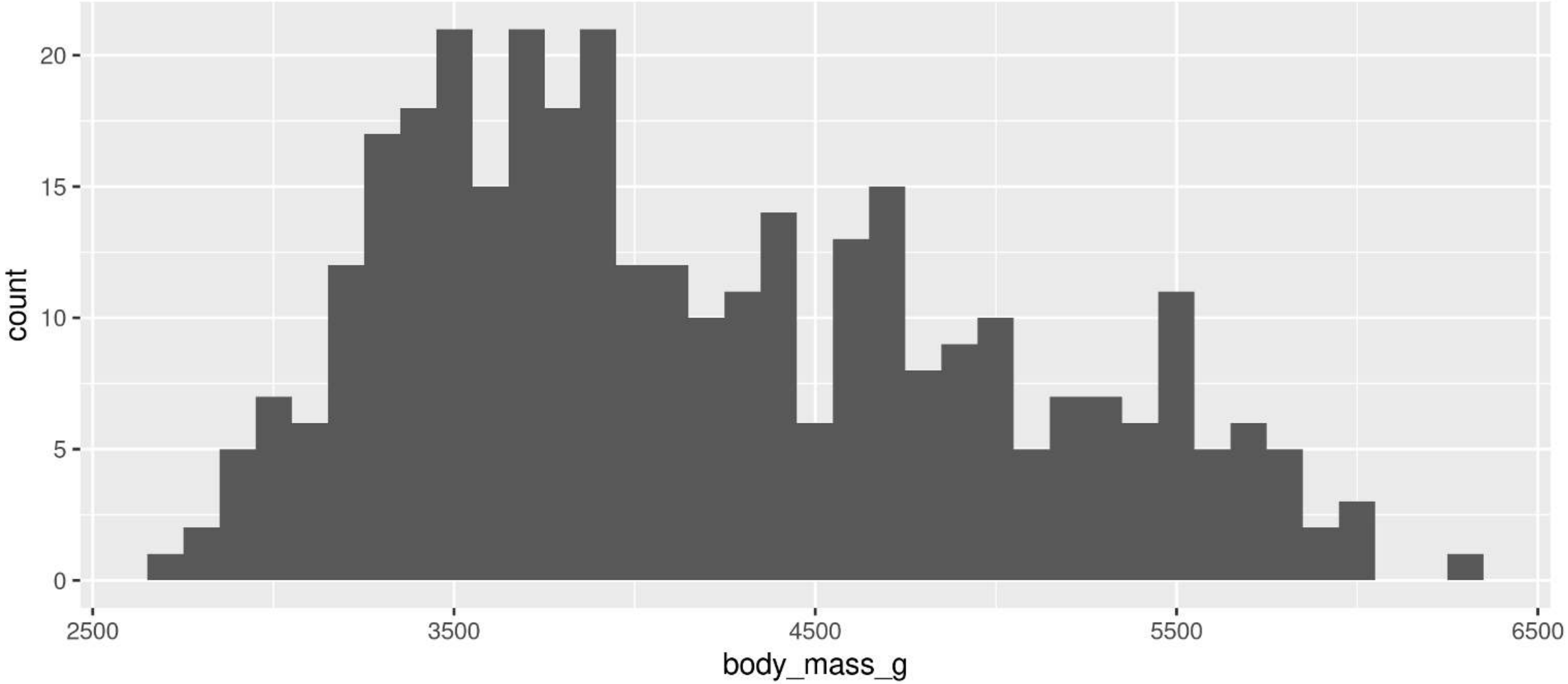
Geoms: Lines

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



Geoms: Histogram

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

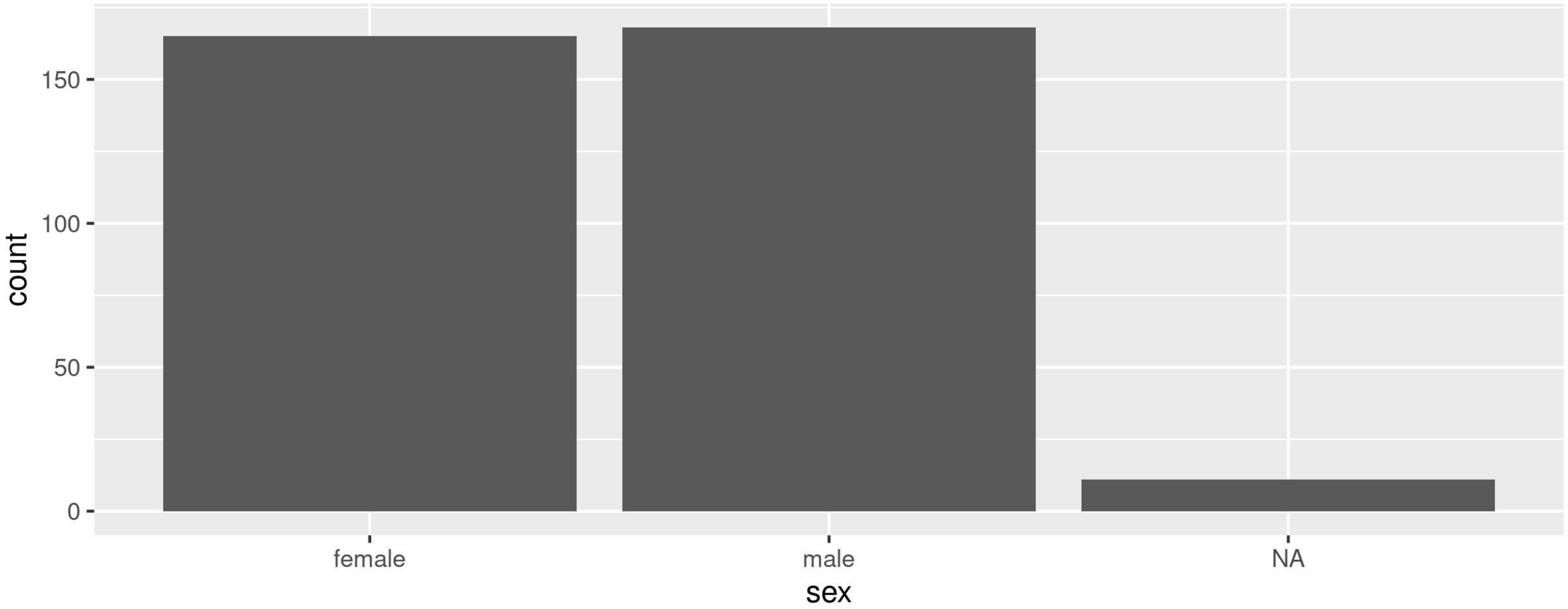


Note: We only need 1 aesthetic here

Geoms: Barplots

Let `ggplot` count your data

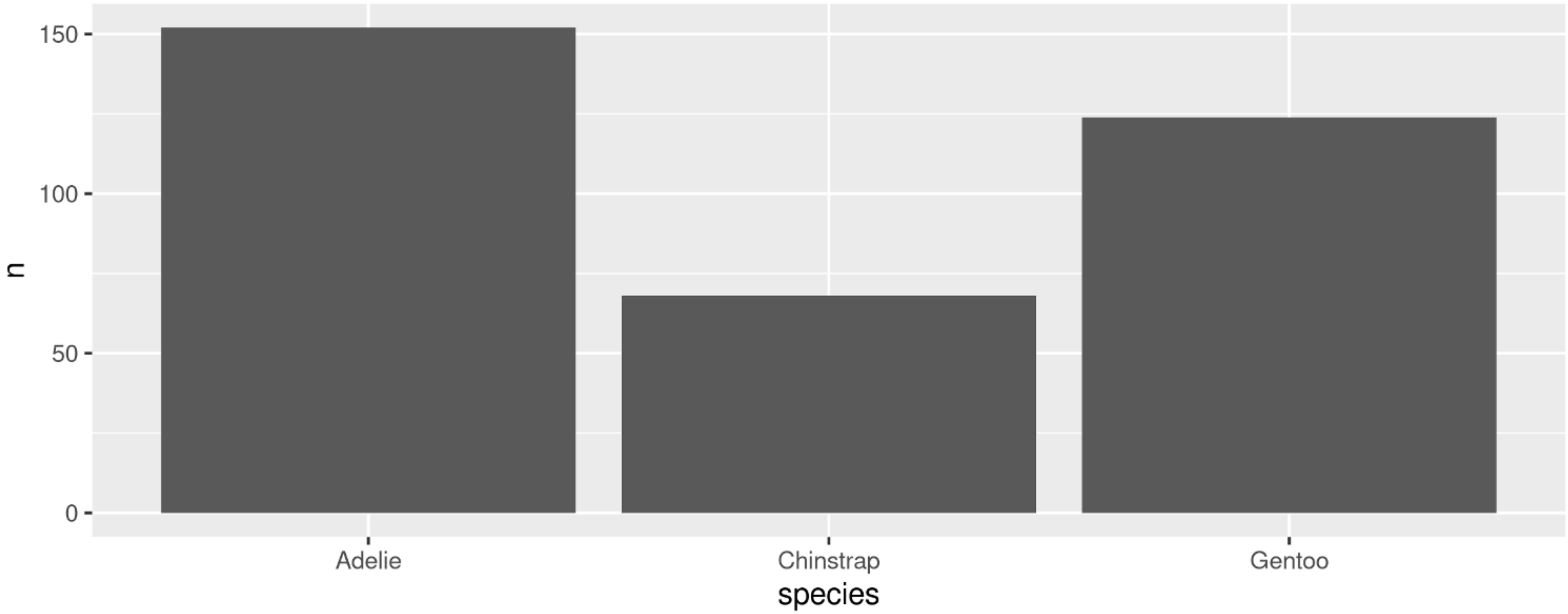
```r
ggplot(data = penguins, aes(x = sex)) +
  geom_bar()
```



Geoms: Barplots

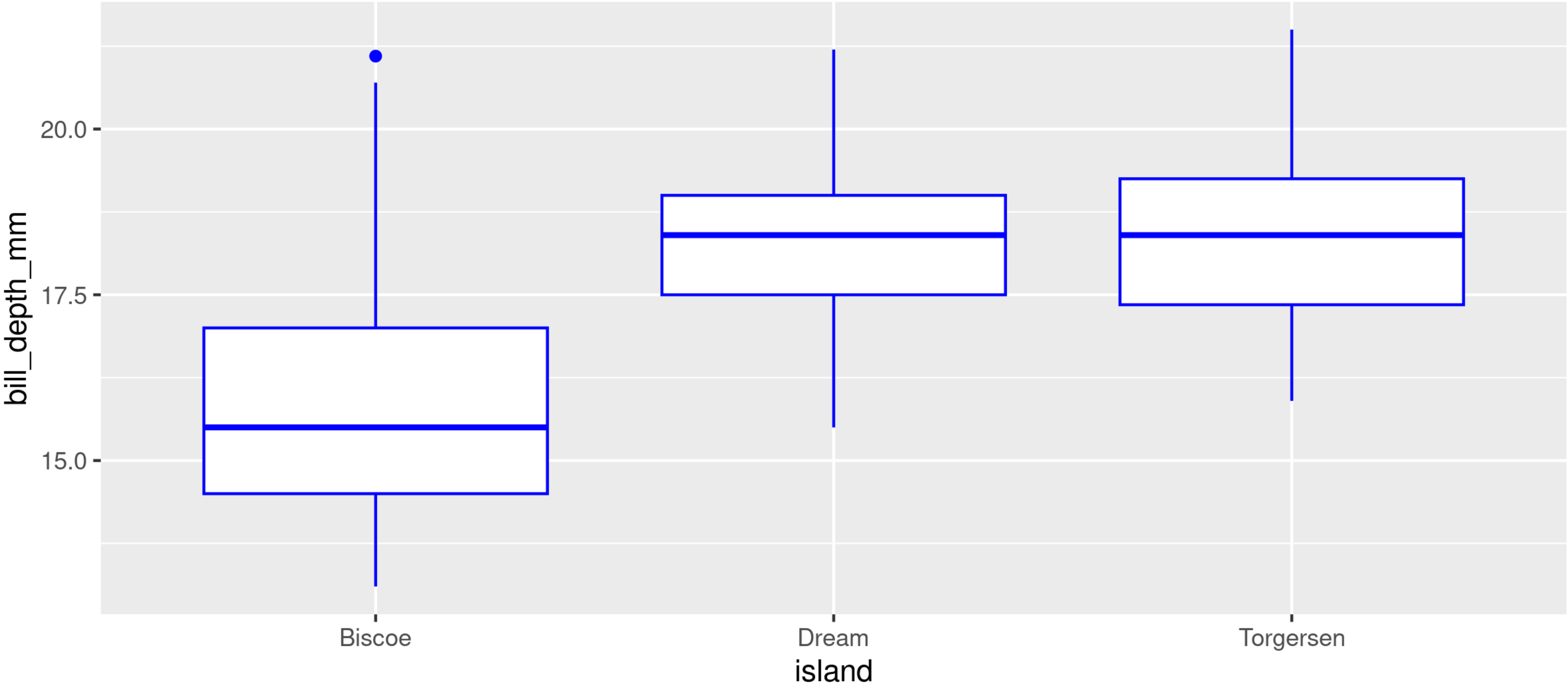
You can also provide the counts

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

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



Your Turn: Create this plot

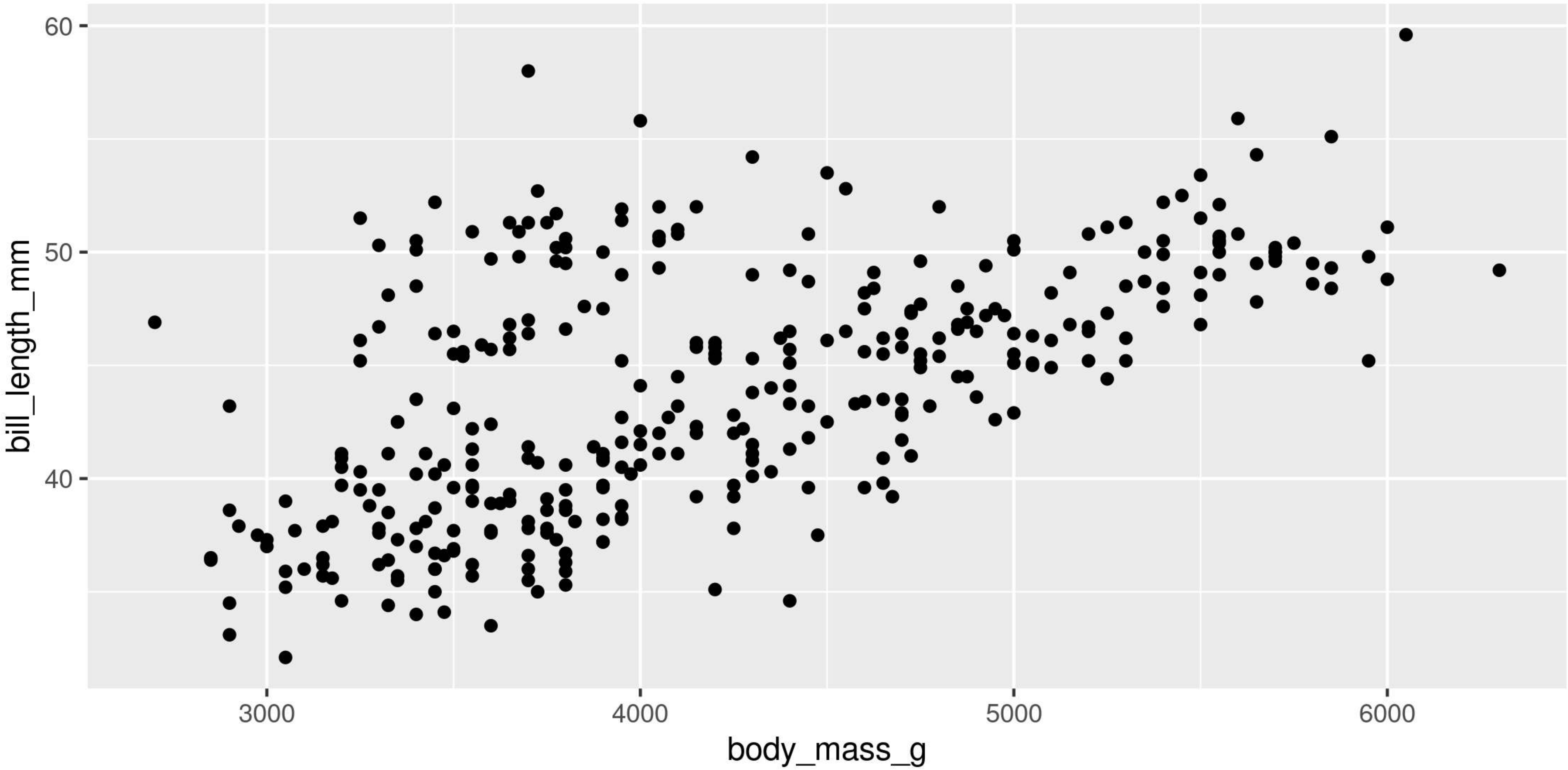
```r
library(ggplot2)

ggplot(data = ____, aes(x = ____, y = ____)) +
  geom_____(____)
```



Showing data by group

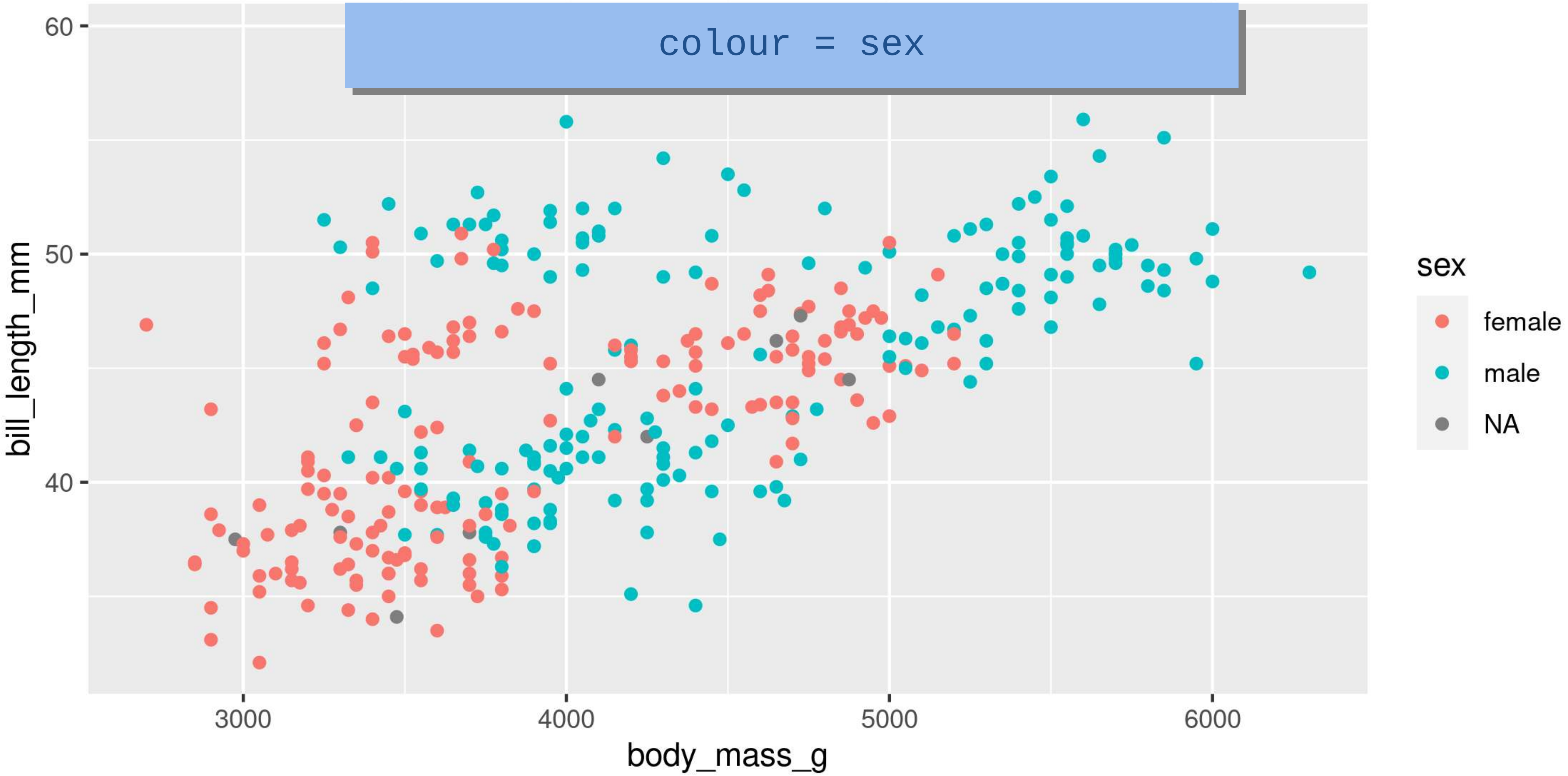
Mapping aesthetics

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



Mapping aesthetics

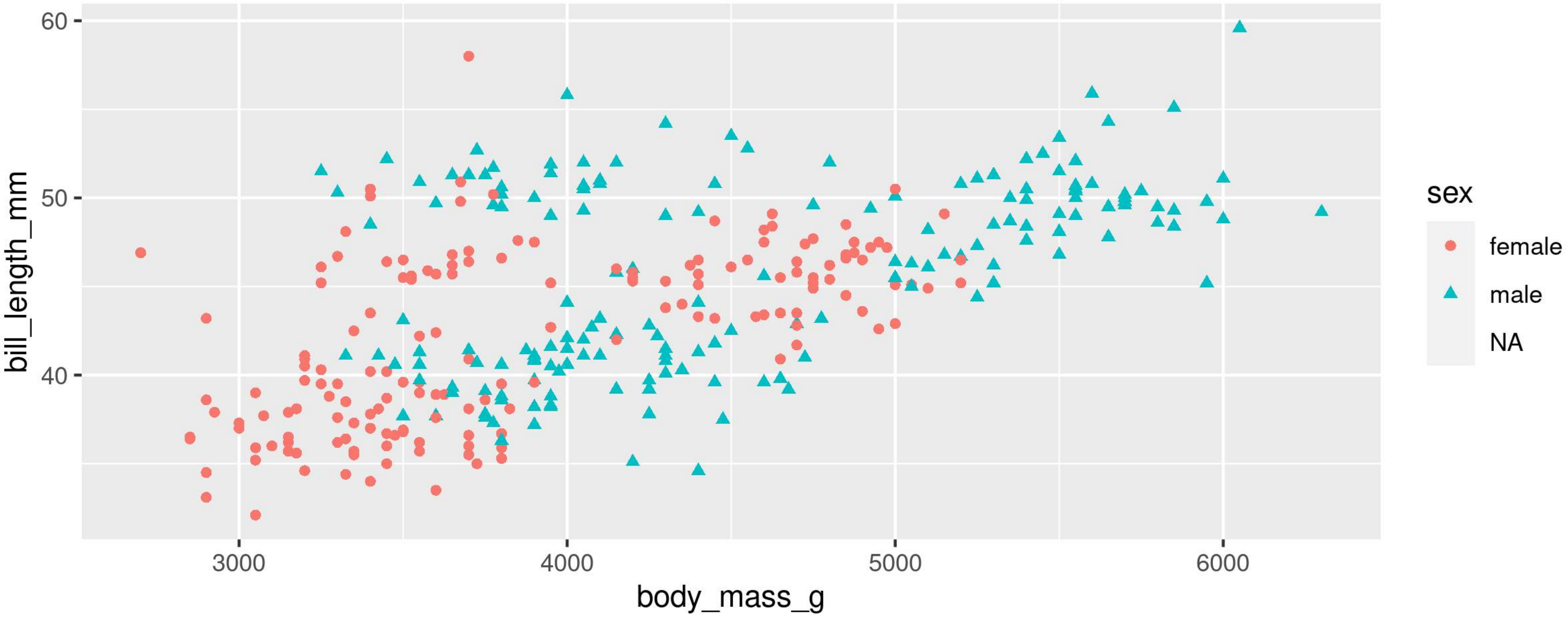
```
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +
  geom_point()
```



Mapping aesthetics

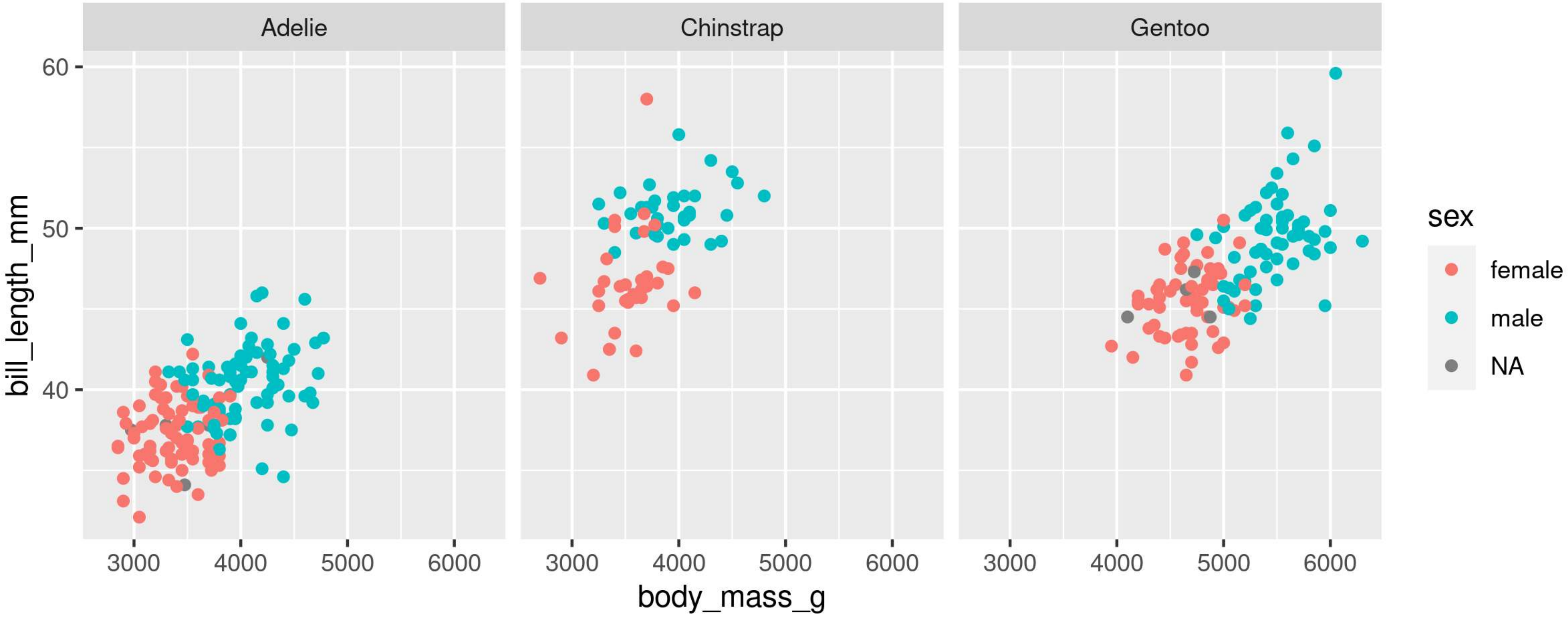
`ggplot` automatically populates the legends (combining where it can)

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



Faceting: `facet_wrap()`

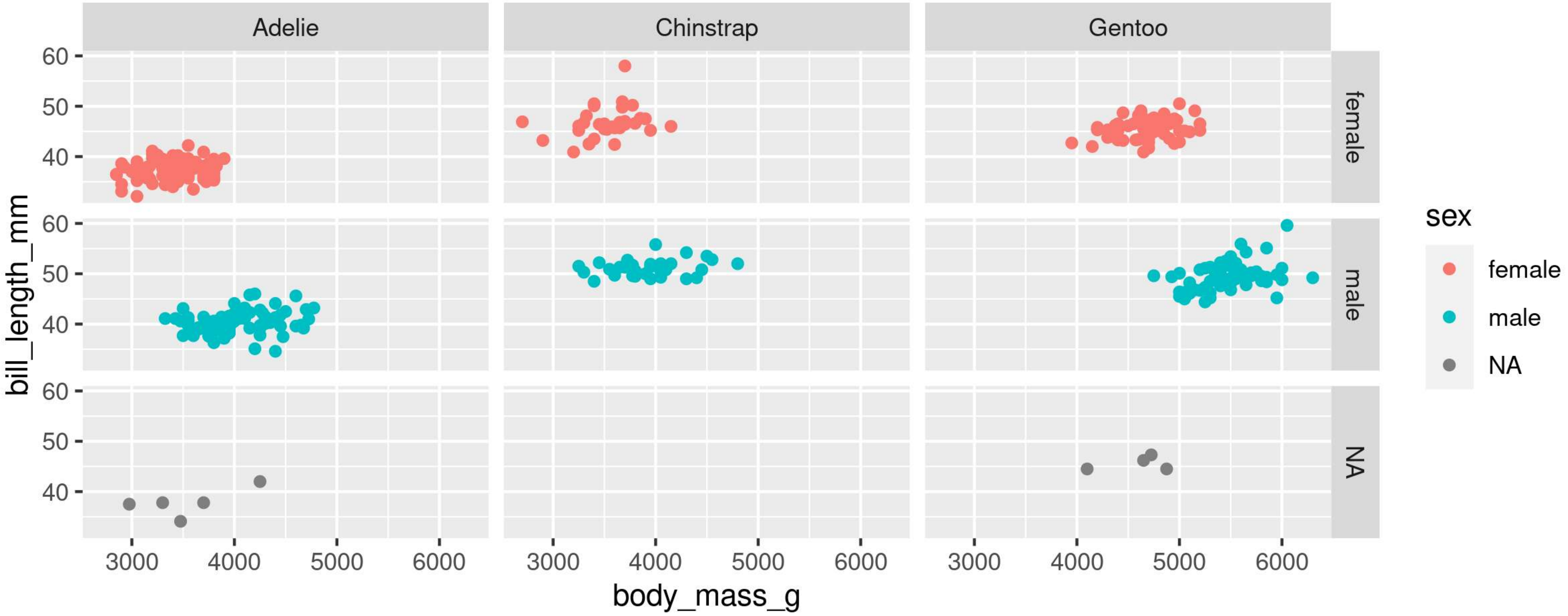
```r
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +
  geom_point() +
  facet_wrap(~ species)
```



Split plots by **one** grouping variable

Faceting: `facet_grid()`

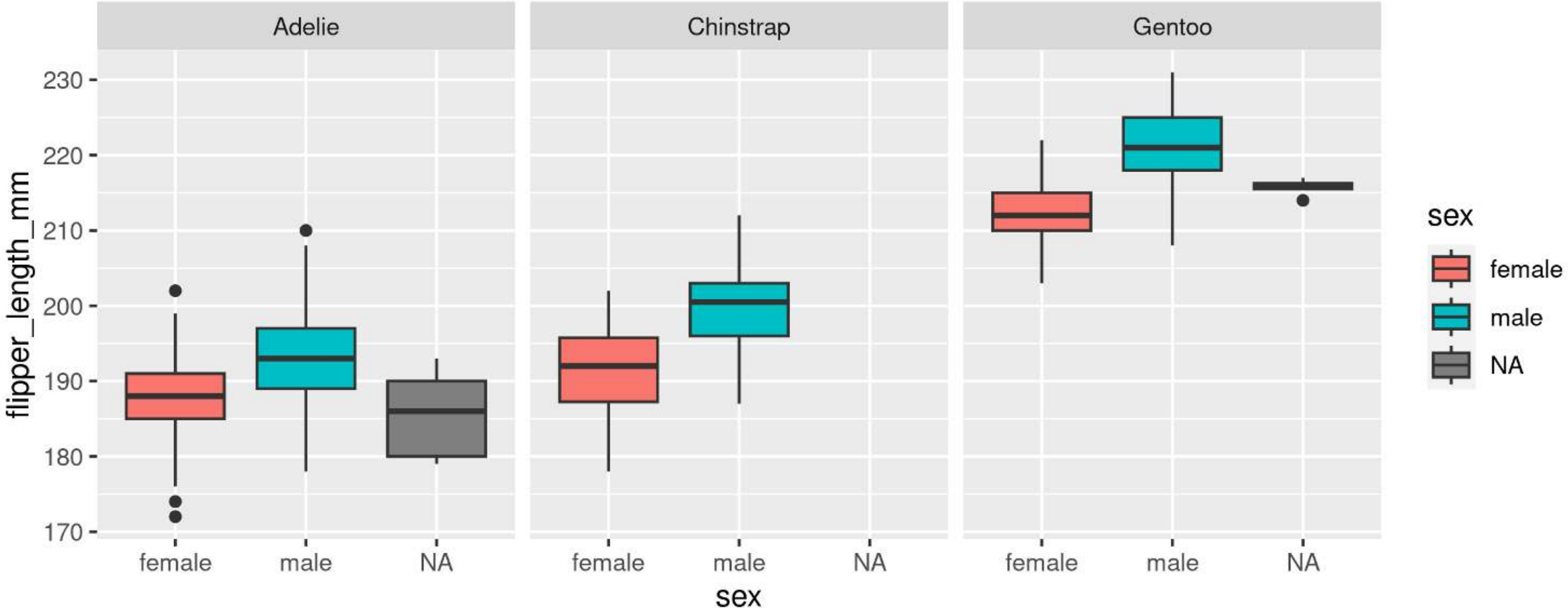
```r
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +
  geom_point() +
  facet_grid(sex ~ species)
```



Split plots by **two** grouping variables

Your Turn: Create this plot

```
ggplot(data = ____, aes(___________________________________)) +
  _____________ +
  _____________
```



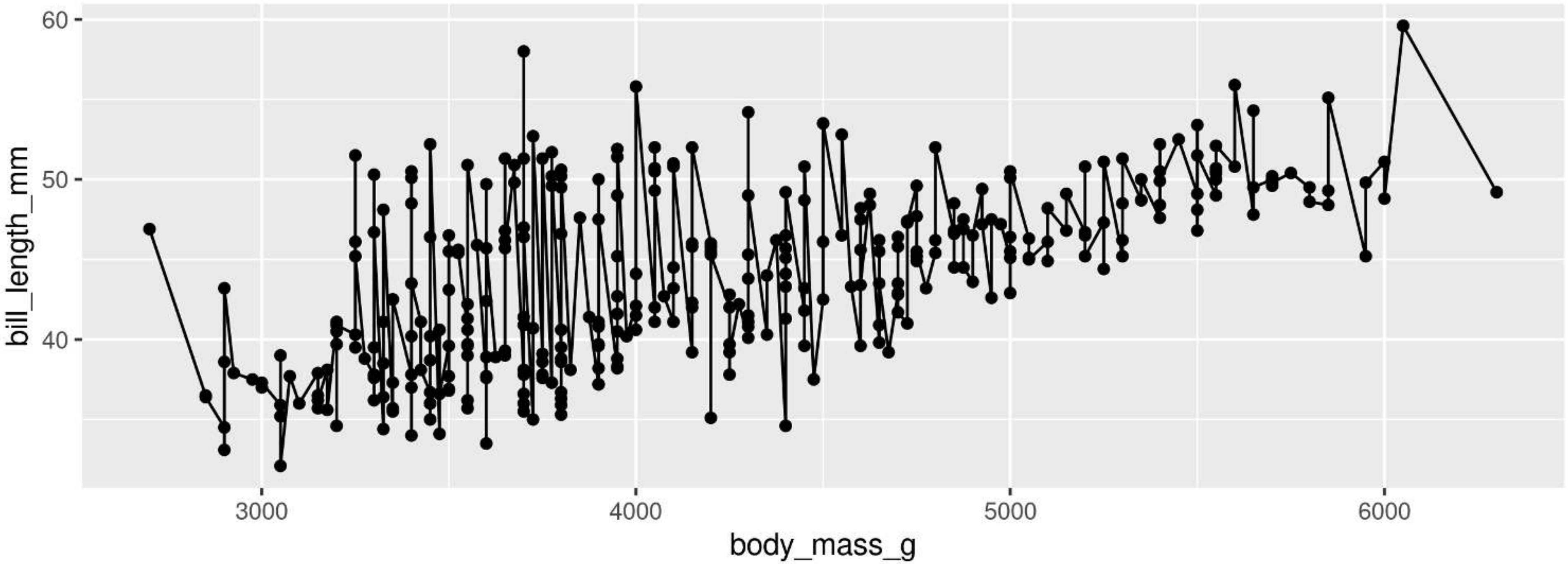
Hint: `colour` is for outlining with a colour, `fill` is for 'filling' with a colour
Too Easy? Split boxplots by sex **and** island

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()
```



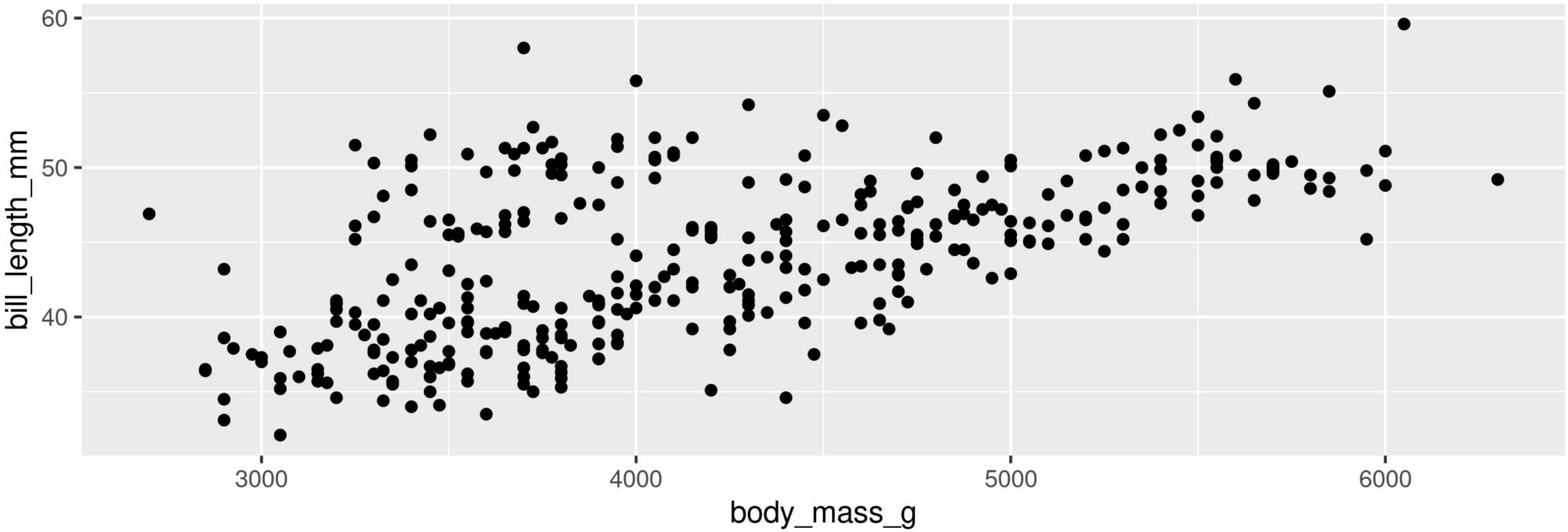
Not what we're looking for

Trendlines / Regression lines

Let’s add a trend line properly

Start with basic plot:

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

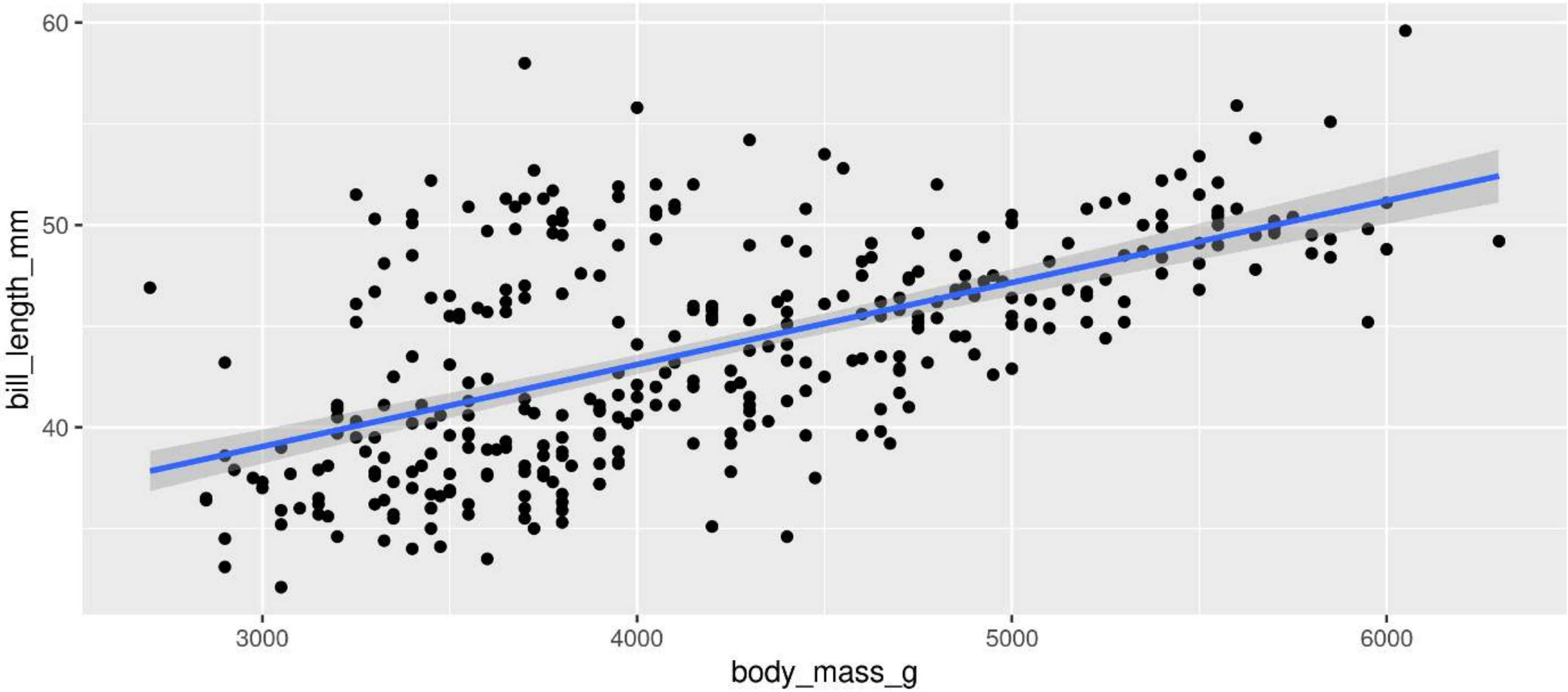


Trendlines / Regression lines

Add the `stat_smooth()`

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

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

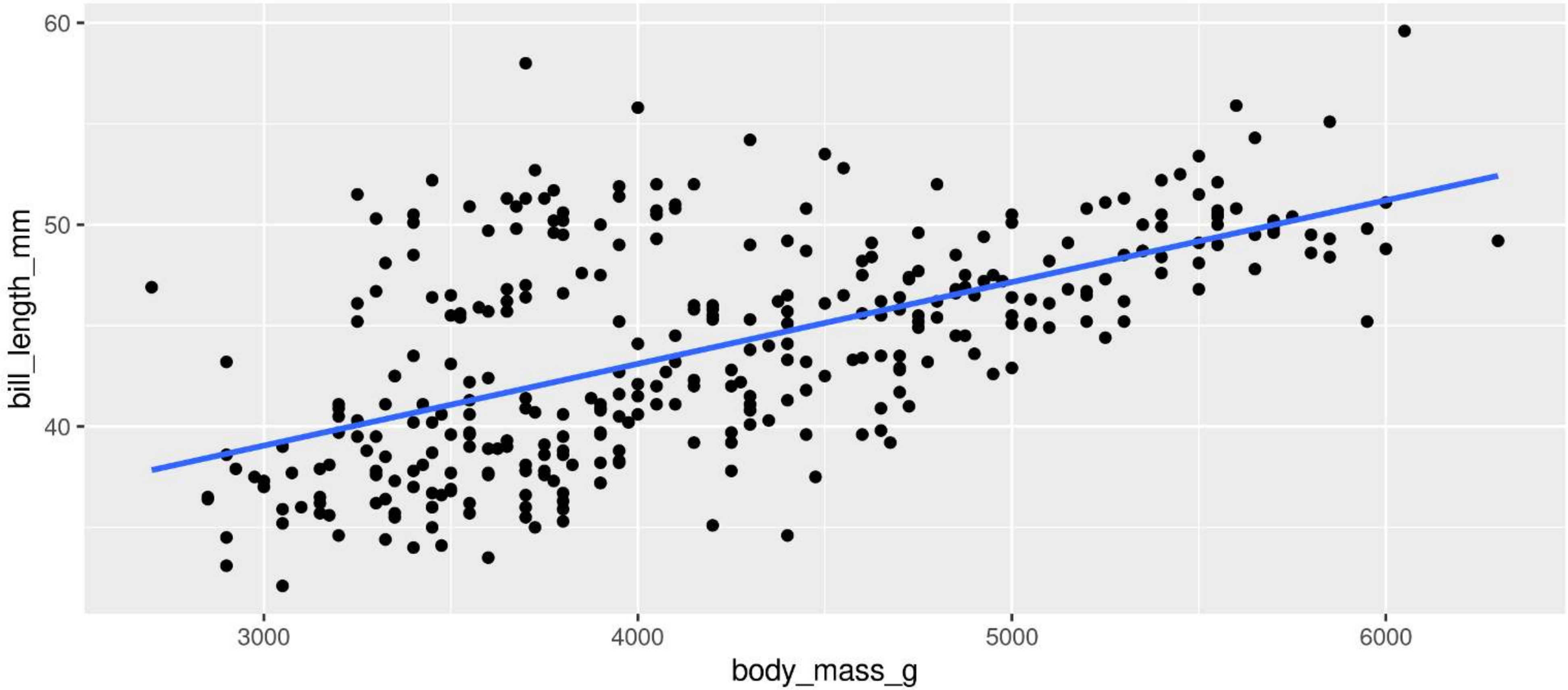


Trendlines / Regression lines

Add the `stat_smooth()`

- remove the grey ribbon `se = FALSE`

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

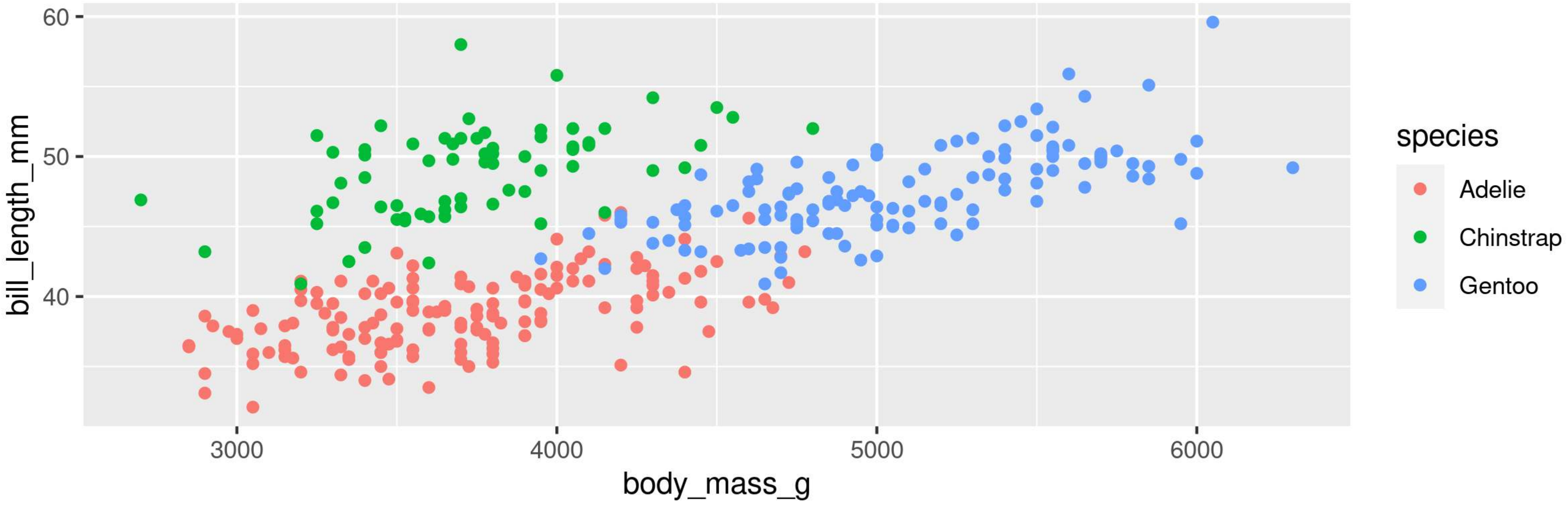


Trendlines / Regression lines

A line for each group

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

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

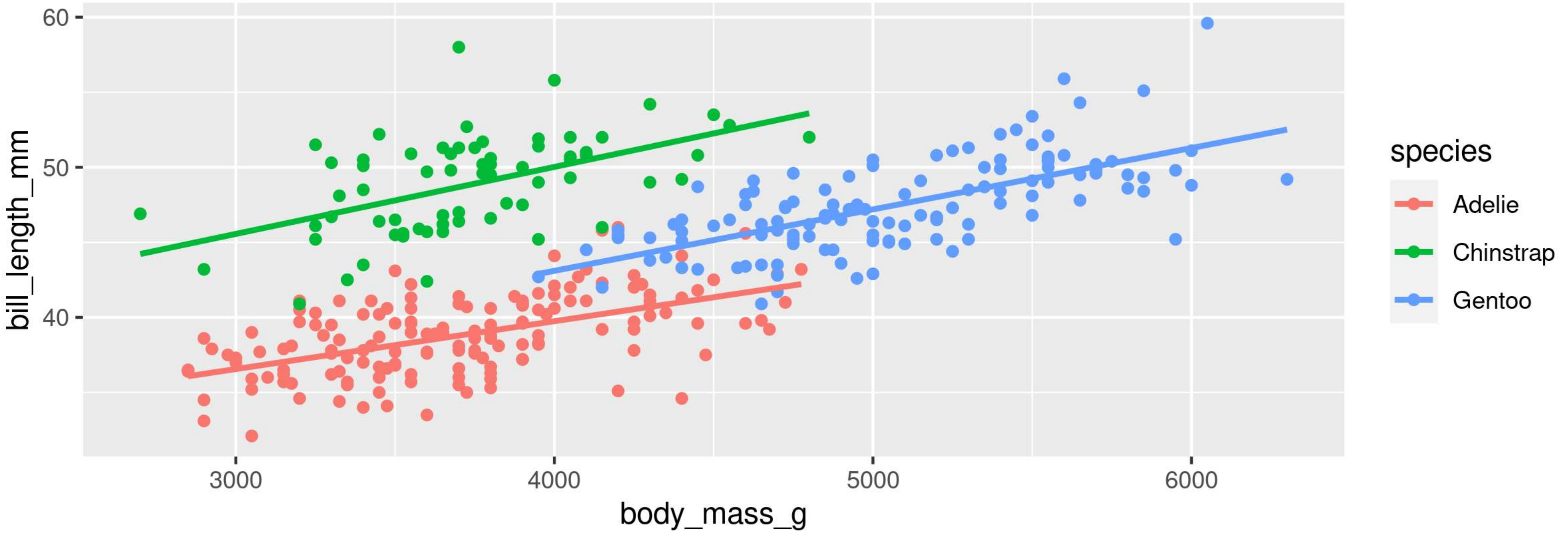


Trendlines / Regression lines

A line for each group

- `stat_smooth()` automatically uses the same grouping

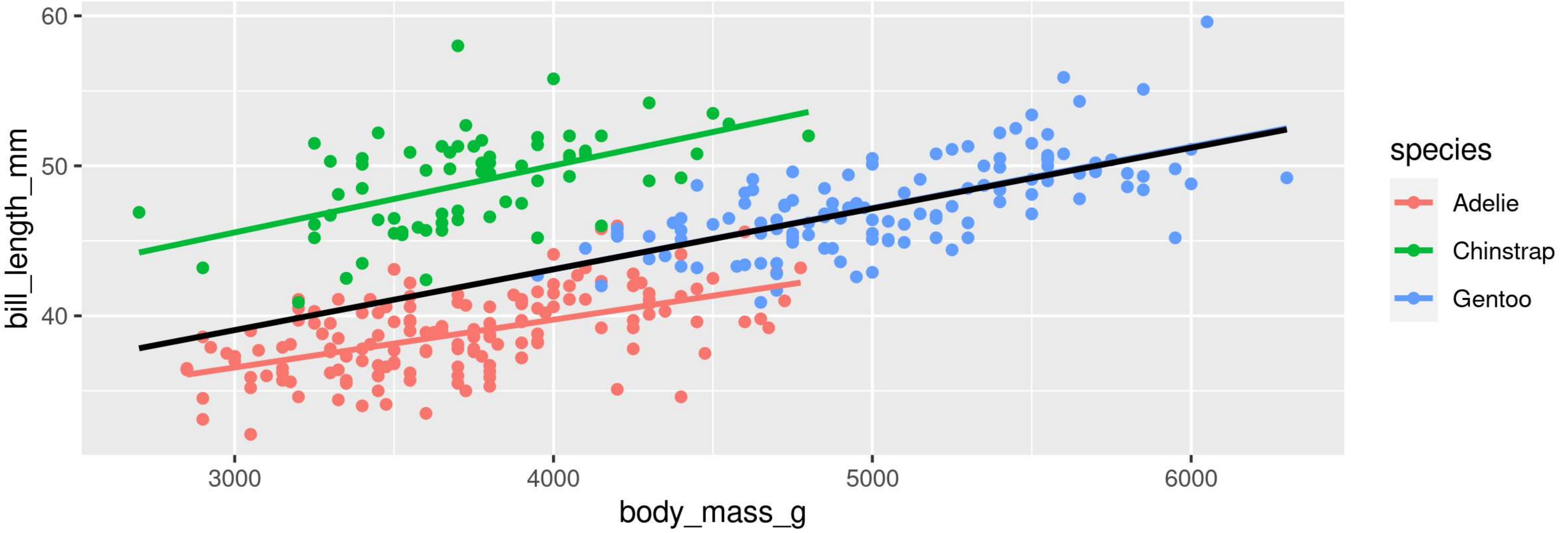
```
g + stat_smooth(method = "lm", se = FALSE)
```



Trendlines / Regression lines

A line for each group AND overall

```r
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*

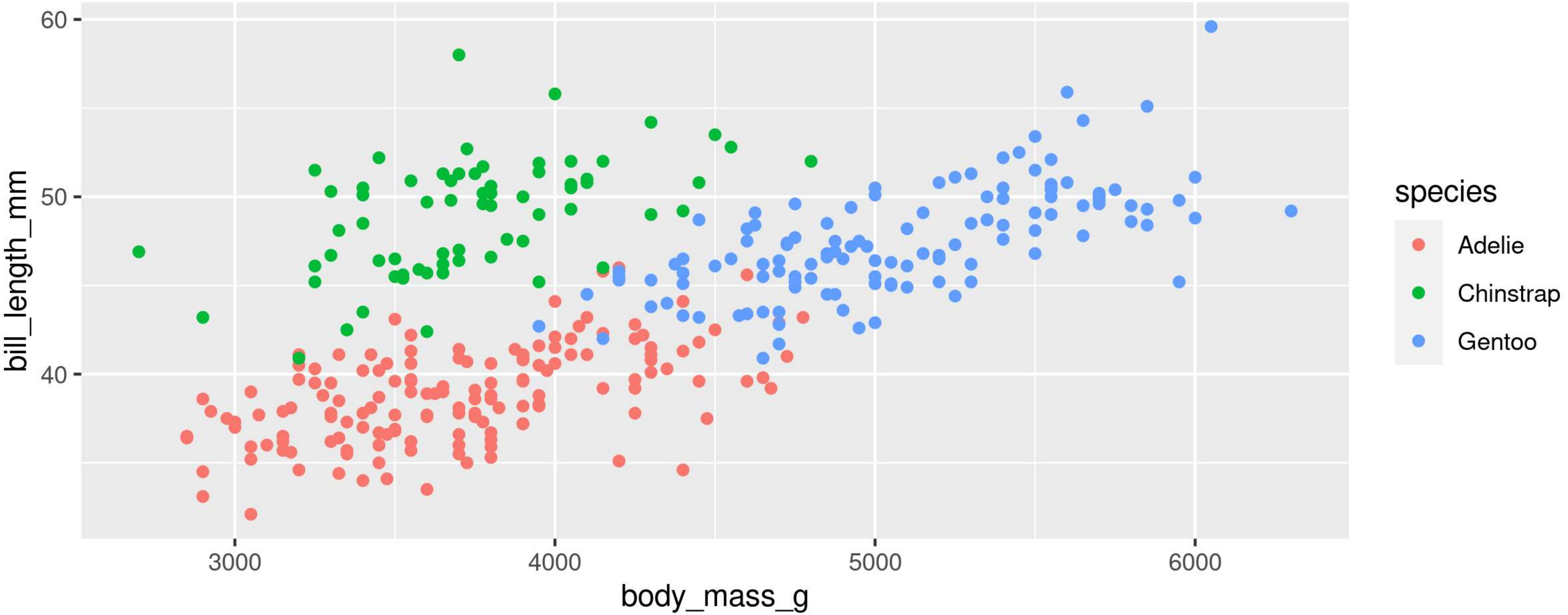
Too Easy? Create a separate plot for each sex as well

Customizing plots

Customizing: Starting plot

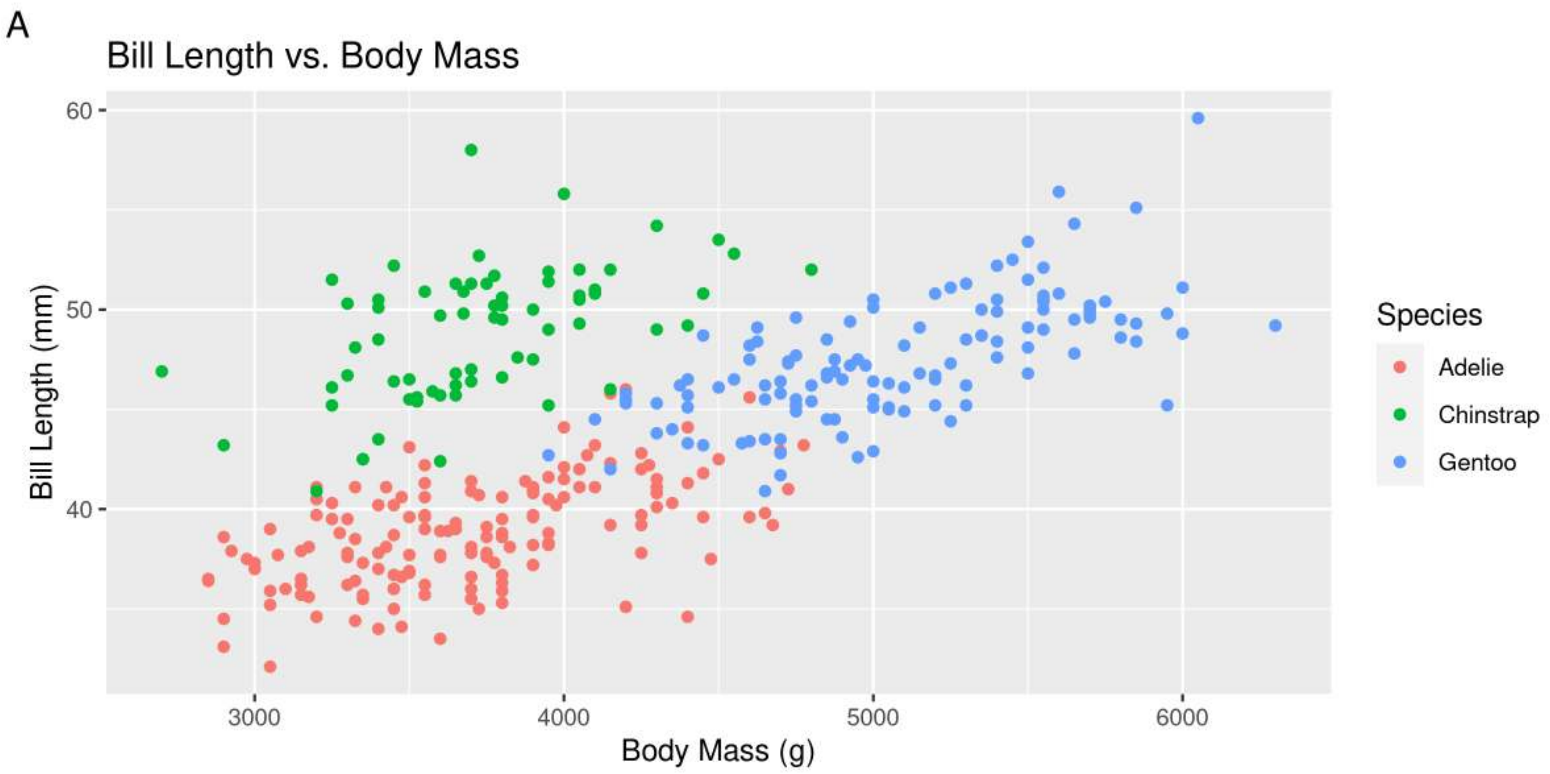
Let’s work with this plot

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



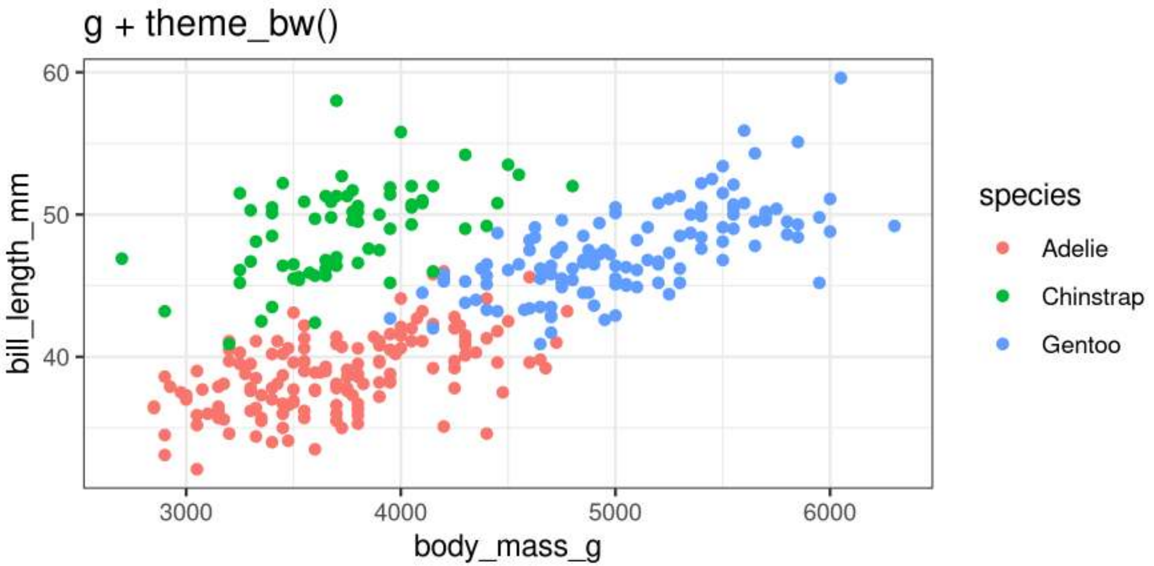
Customizing: Labels

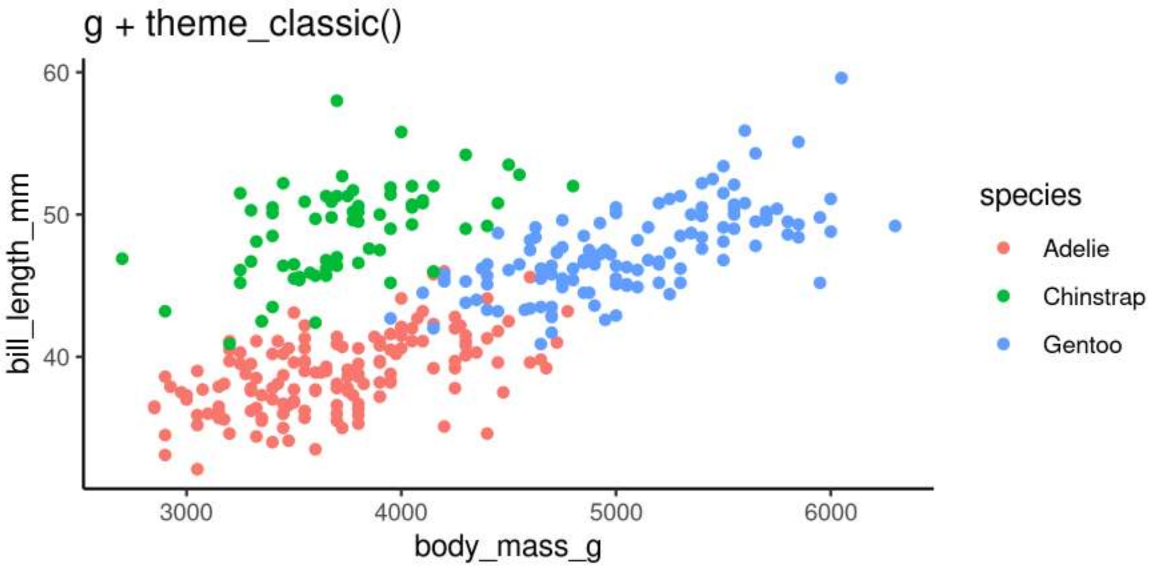
```r
g + labs(title = "Bill Length vs. Body Mass",
         x = "Body Mass (g)",
         y = "Bill Length (mm)",
         colour = "Species", tag = "A")
```

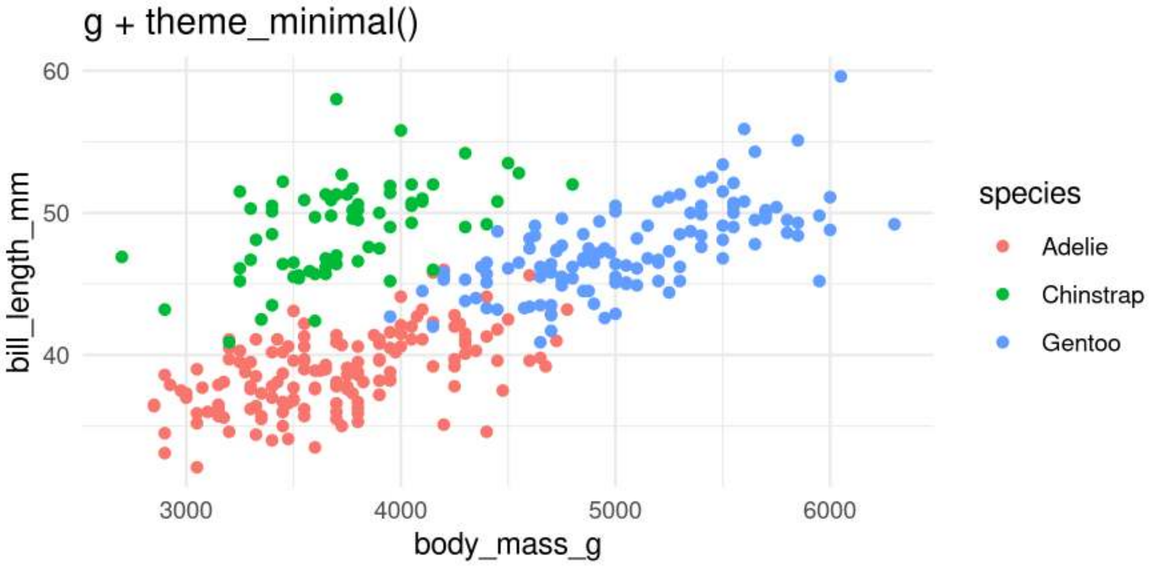


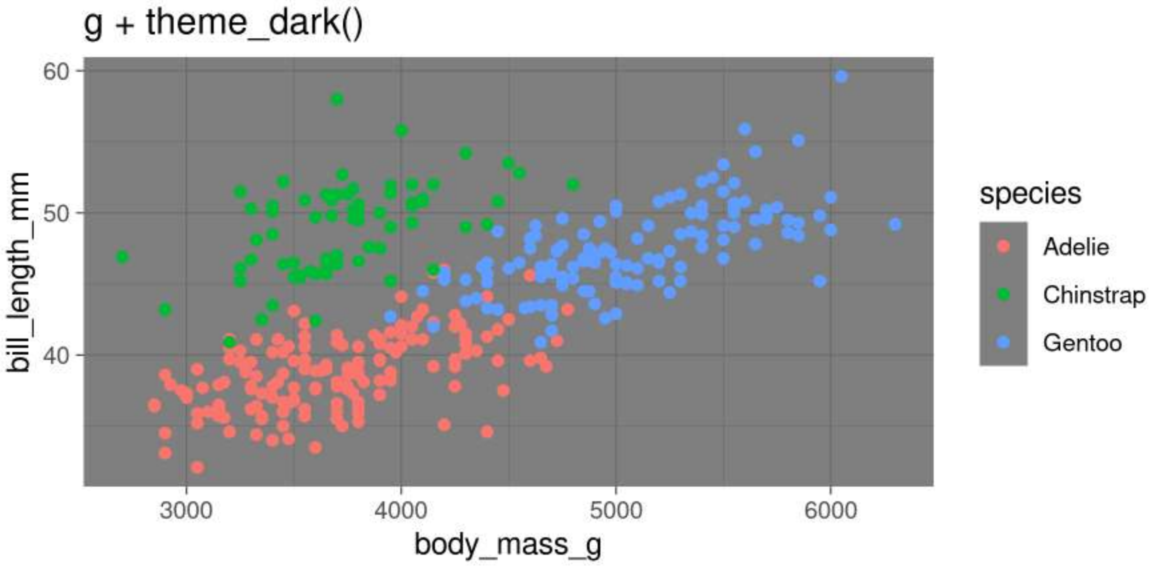
Practice for later: Add proper labels to some of your previous plots

Customizing: Built-in themes


g + theme_bw()


g + theme_classic()


g + theme_minimal()


g + theme_dark()

Customizing: Axes

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

- `scale_x_continuous()`
- `scale_y_discrete()`
- etc.

Common arguments

```r
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

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

Customizing: Axes

Space between origin and axis start

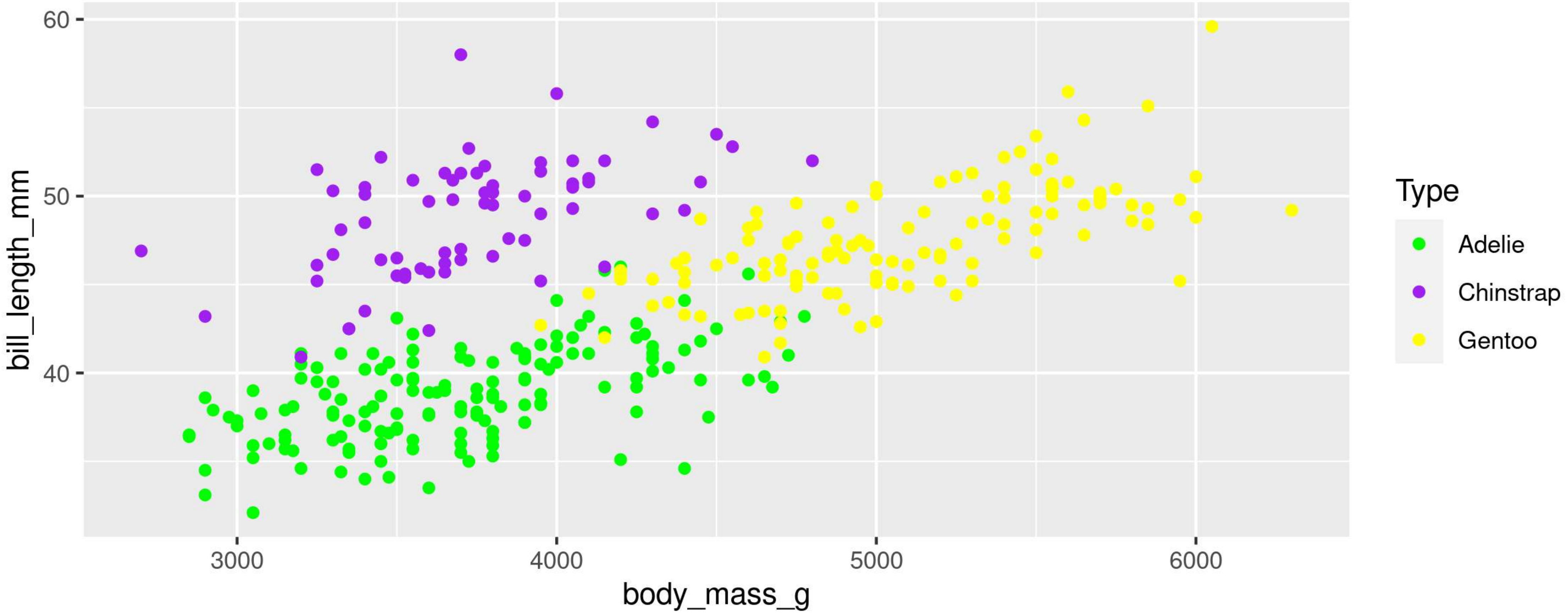
```r
g + scale_x_continuous(expand = c(0, 0))
```

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"))
```

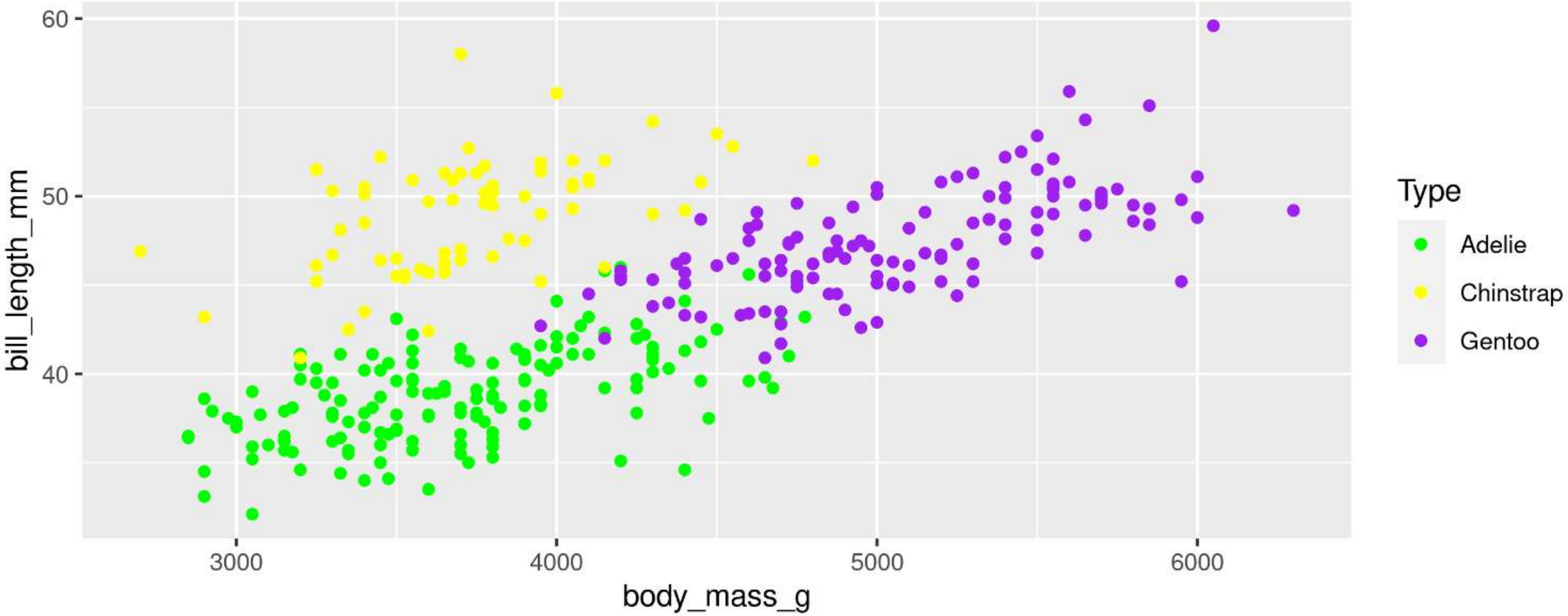


Customizing: Aesthetics

Using scales

Or be very explicit:

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

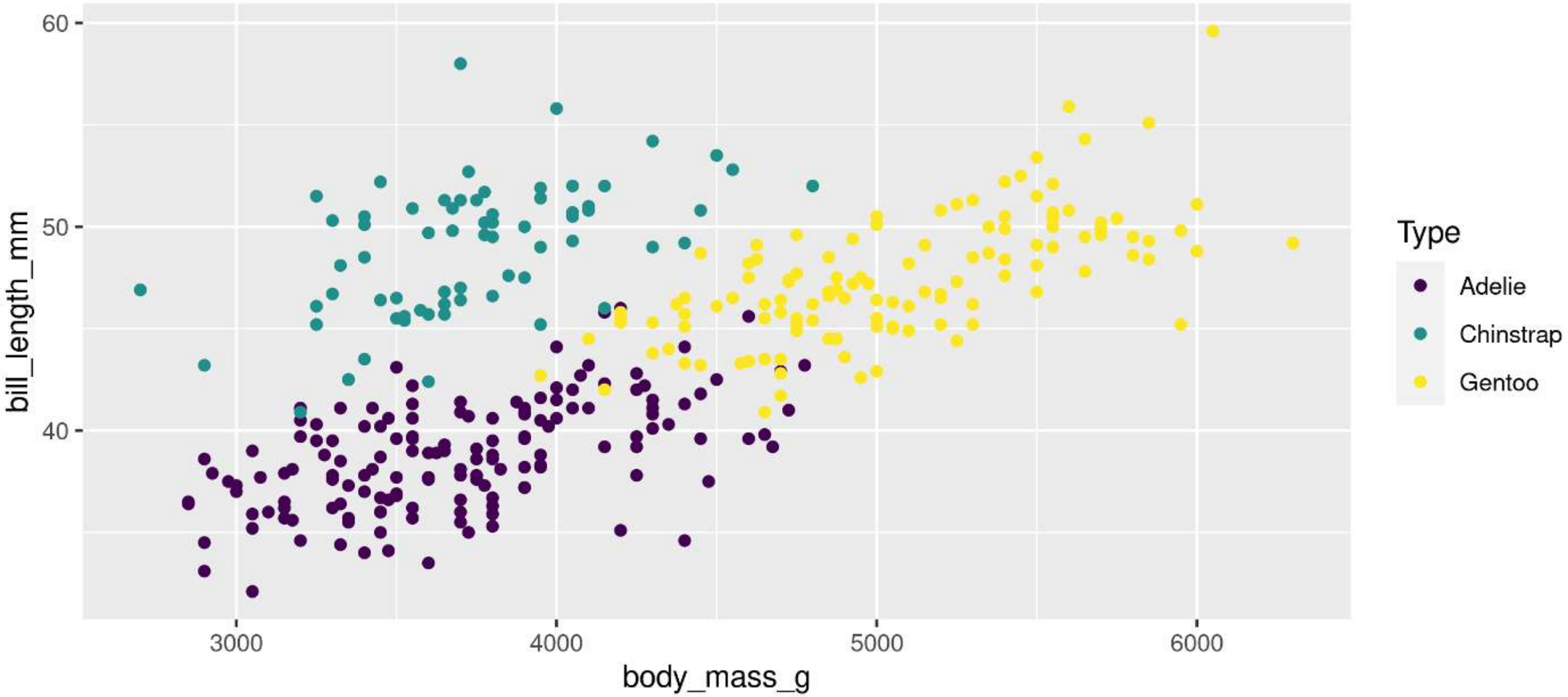


Customizing: Aesthetics

For colours, consider colour-blind-friendly scale

`viridis_d` for “discrete” data

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

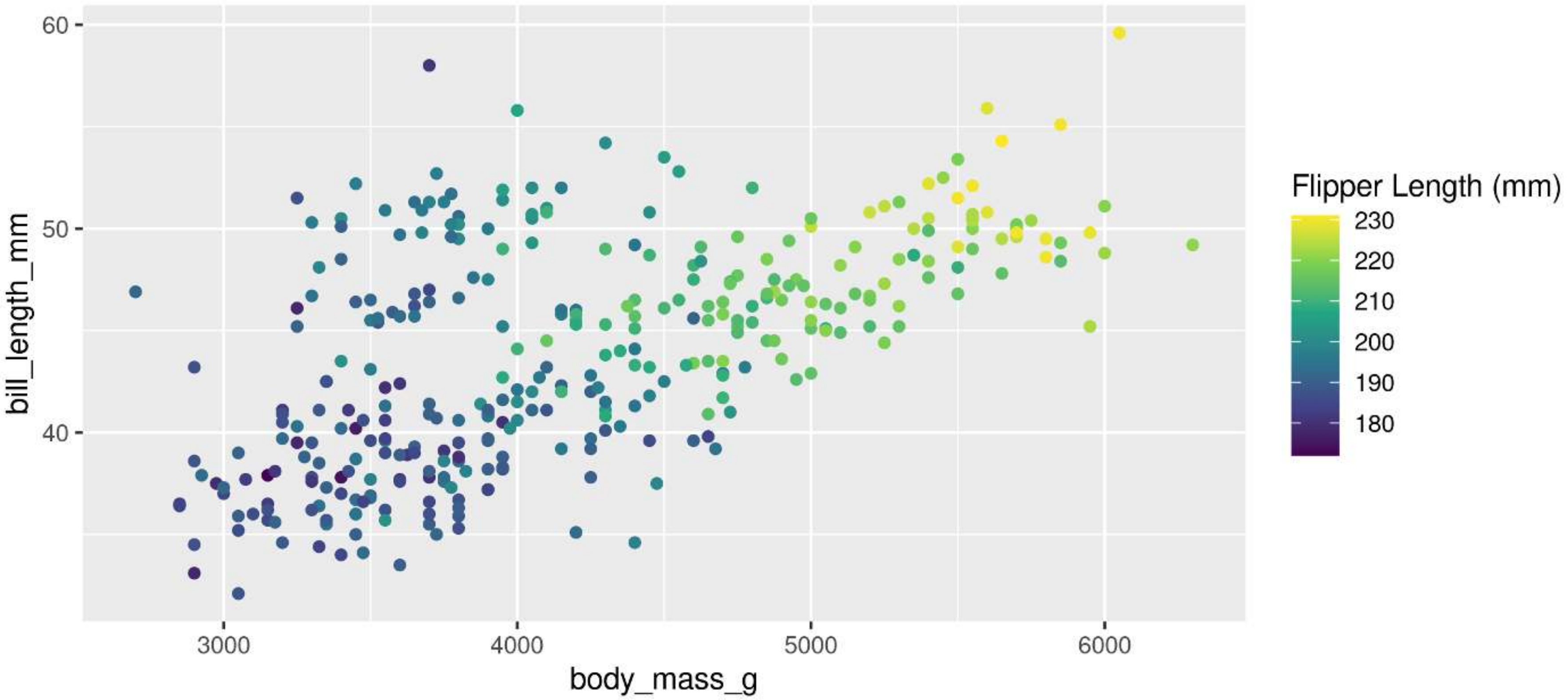


Customizing: Aesthetics

For colours, consider colour-blind-friendly scale

`viridis_c` for "continuous" data

```r
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)")
```

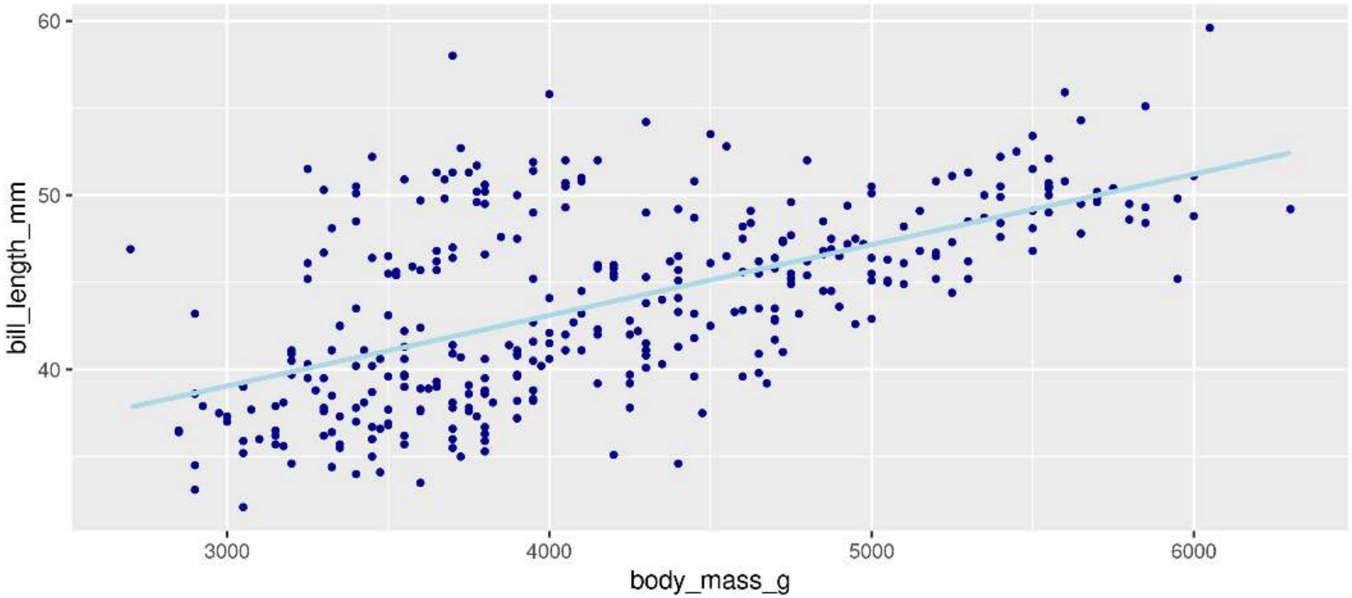


Customizing: Aesthetics

Forcing

Remove the association between a variable and an aesthetic

```r
ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = sex)) +
  geom_point(colour = "darkblue", size = 1) +
  stat_smooth(method = "lm", se = FALSE, colour = "lightblue")
```

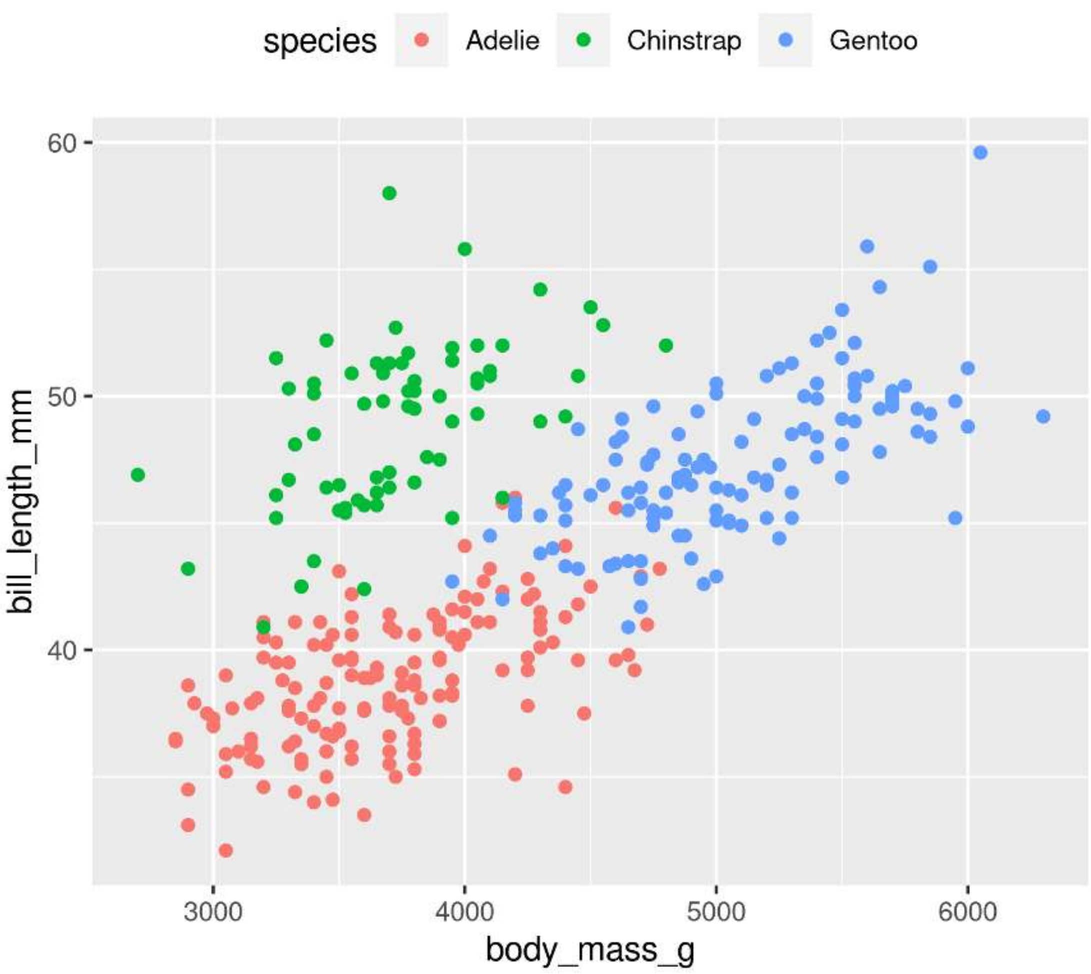


Note: When forcing, aesthetic is not inside `aes()`

Customizing: Legends placement

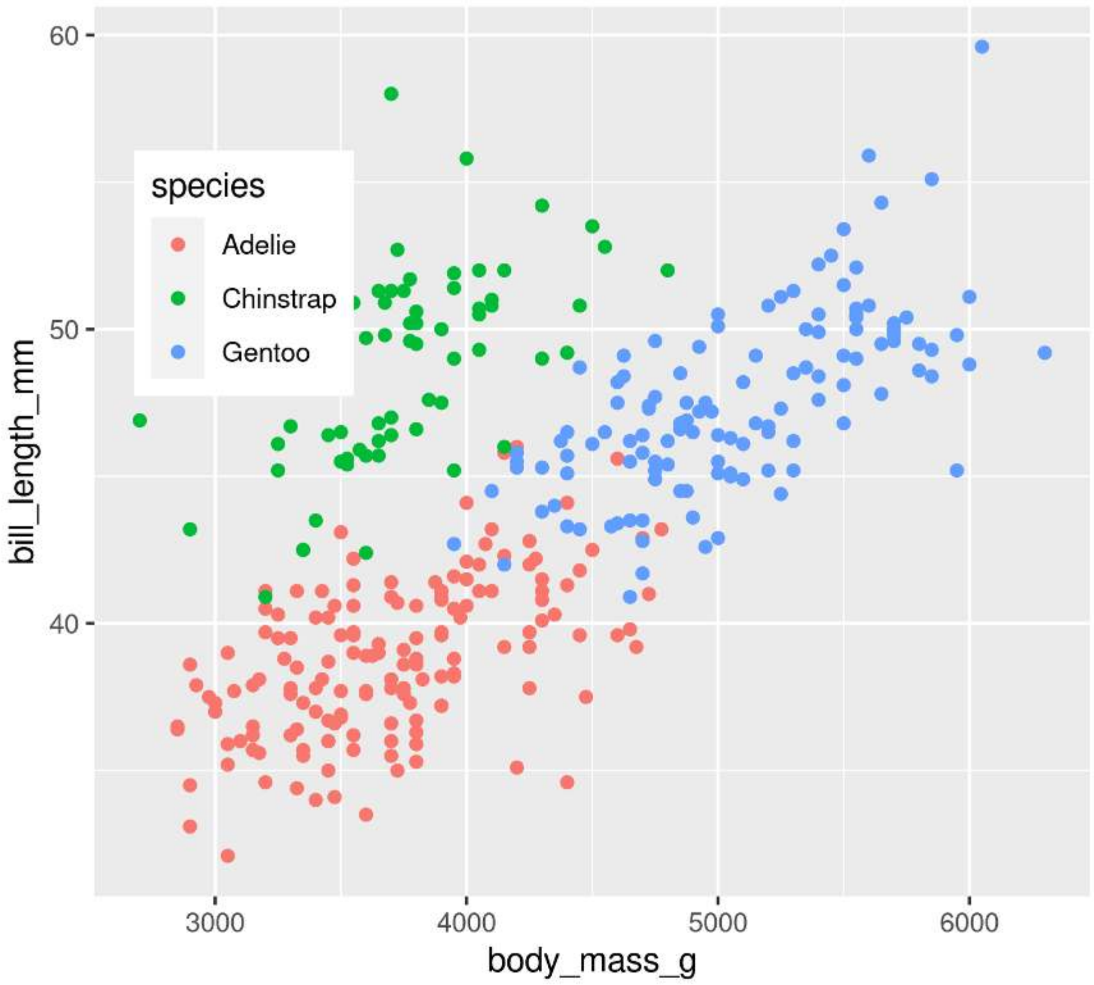
At the: top, bottom, left, right

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



Exactly here

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



Combining plots

Combining plots with `patchwork`

Setup

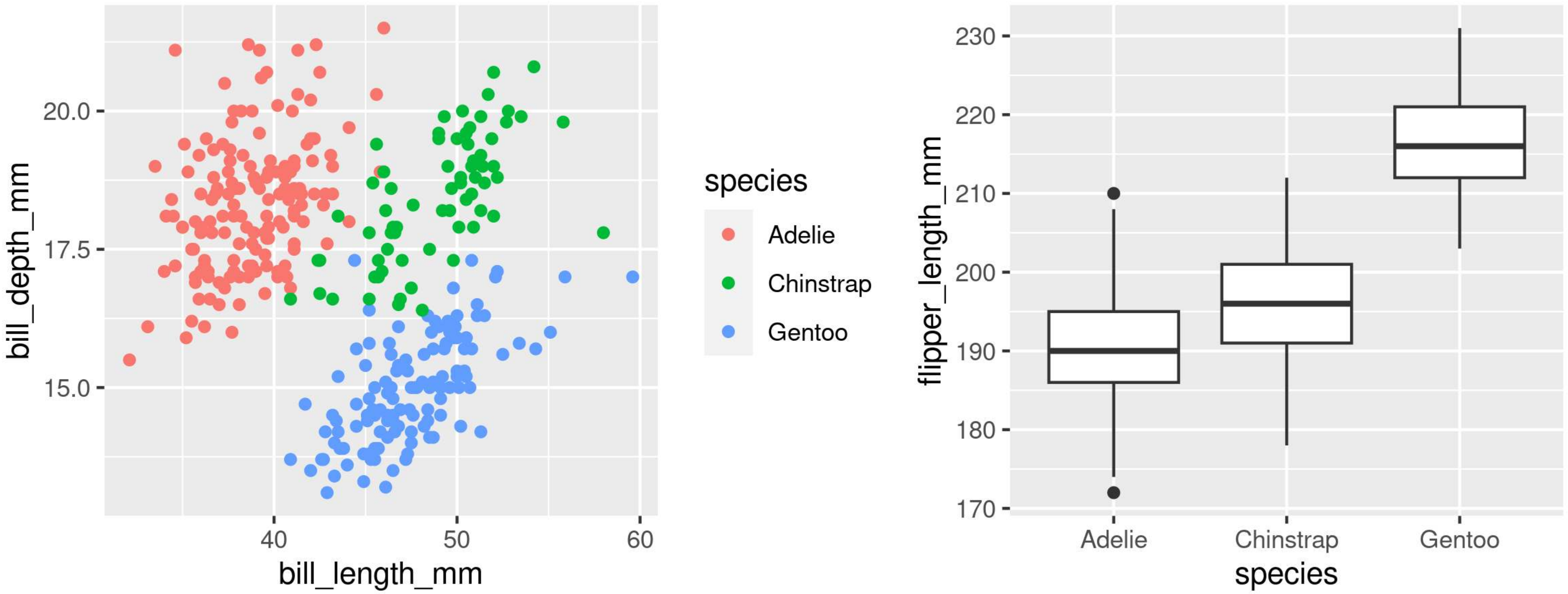
- Load `patchwork`
- Create a couple of different plots

```r
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

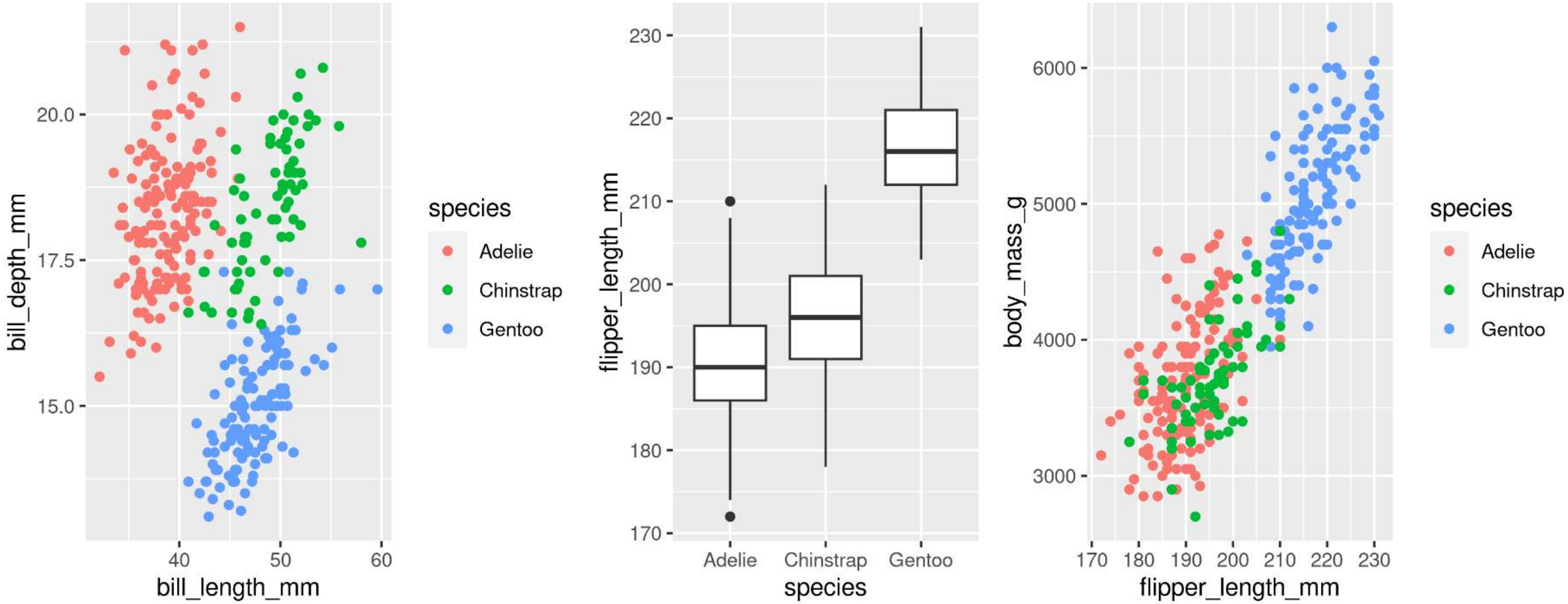
```r
g1 + g2
```



Combining plots with `patchwork`

Side-by-Side 3 plots

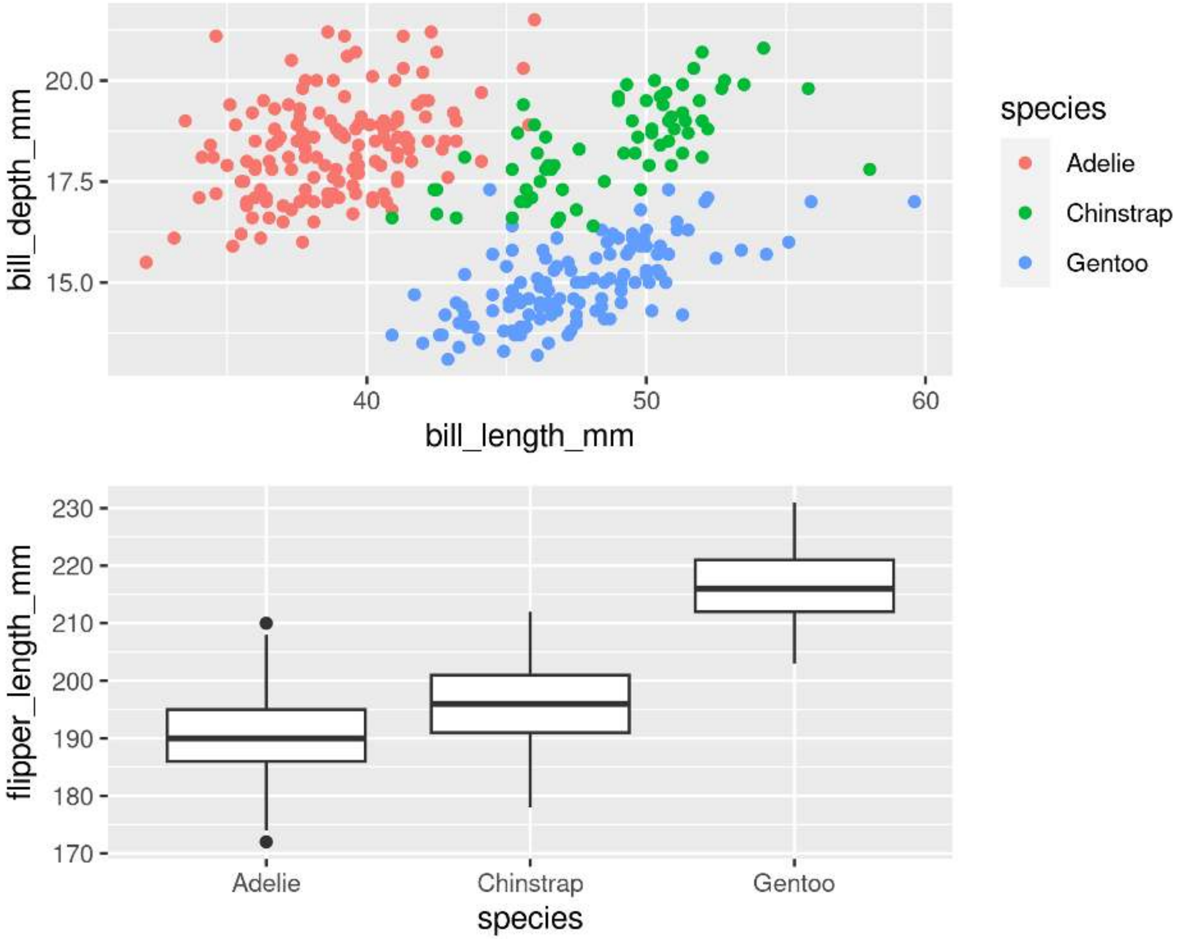
```
g1 + g2 + g3
```



Combining plots with `patchwork`

Stacked 2 plots

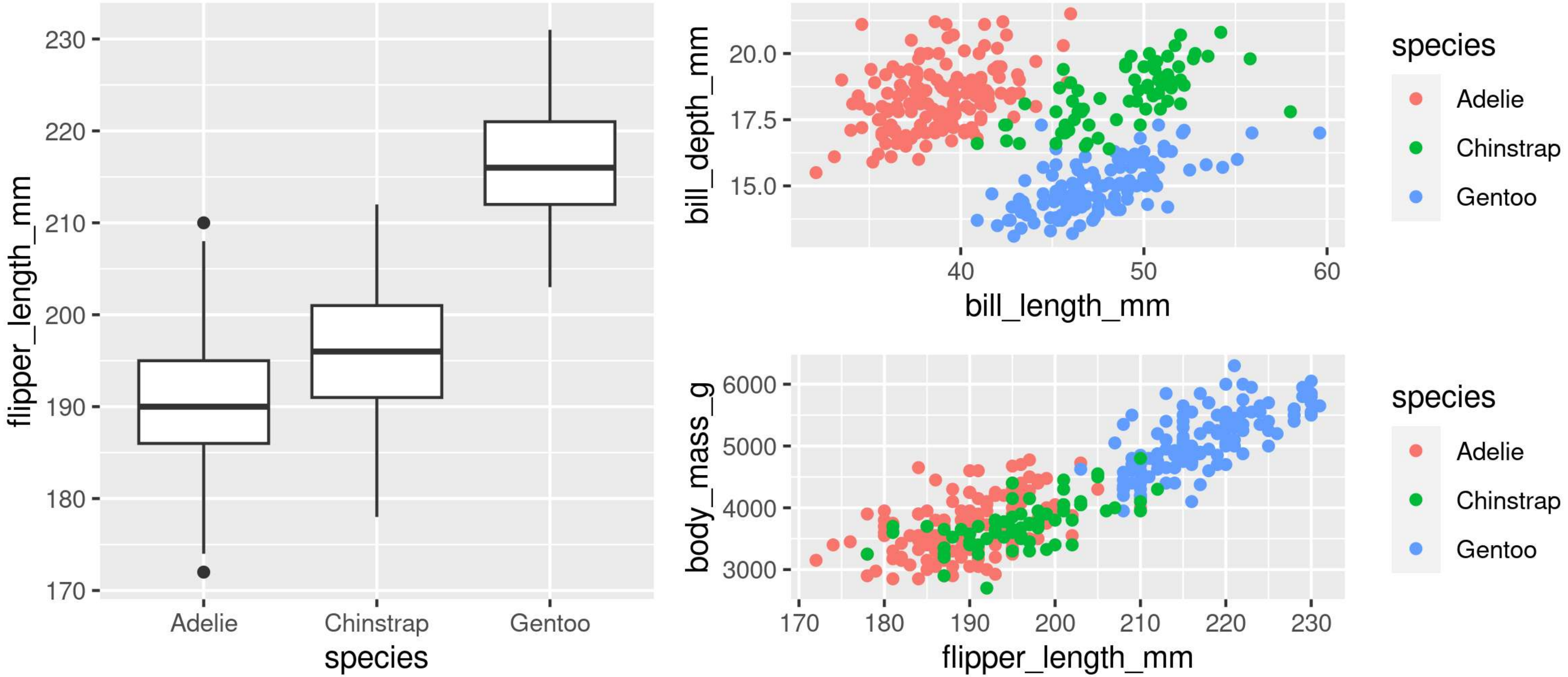
```
g1 / g2
```



Combining plots with `patchwork`

More complex arrangements

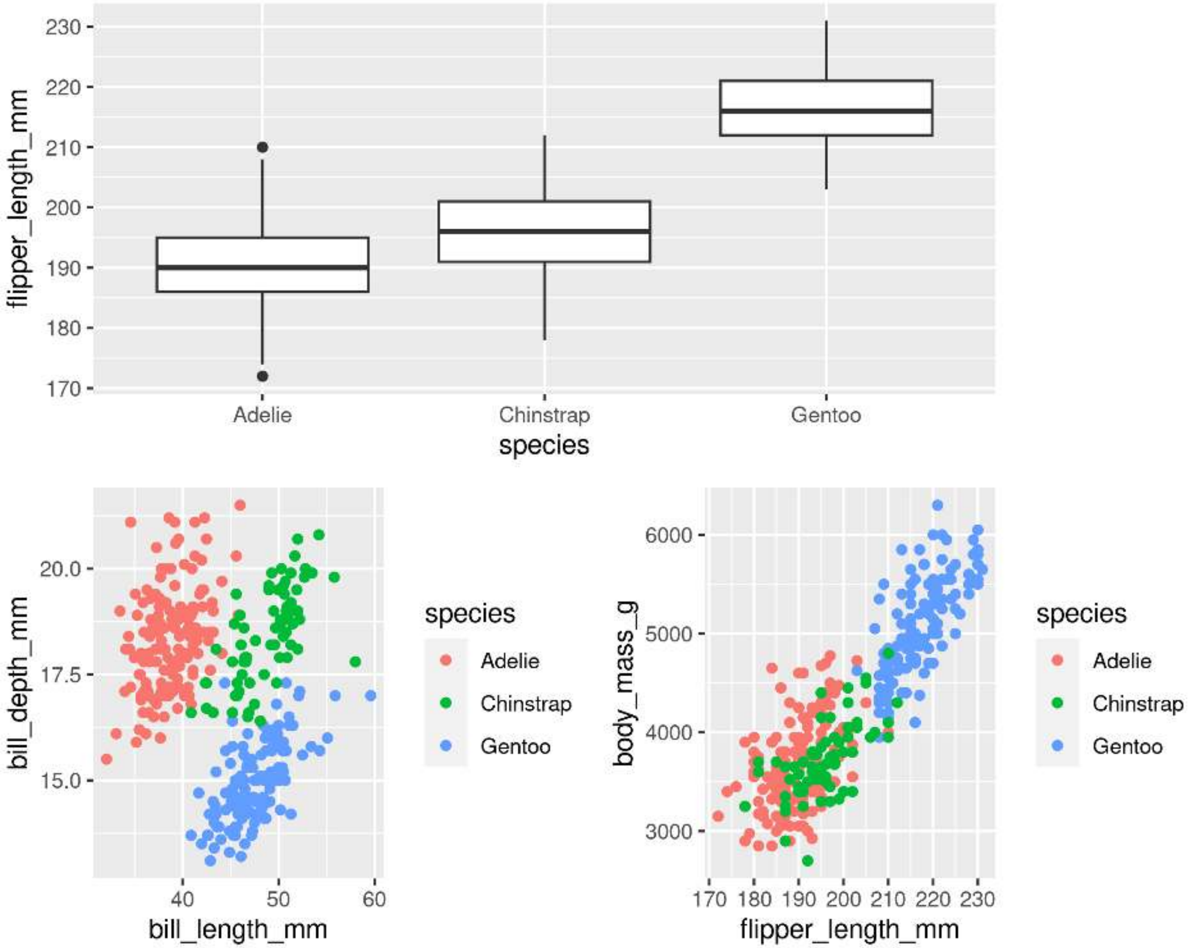
```
g2 + (g1 / g3)
```



Combining plots with `patchwork`

More complex arrangements

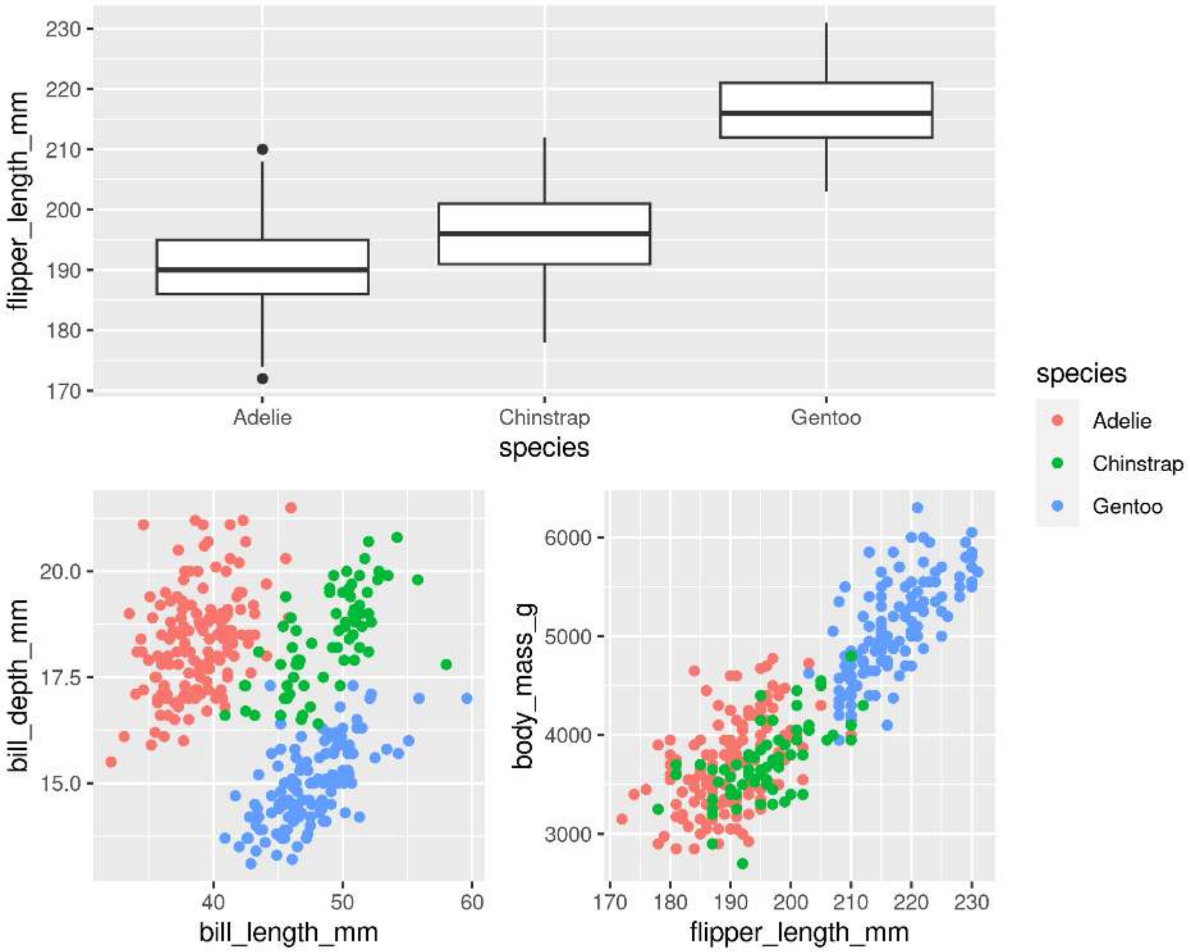
```r
g2 / (g1 + g3)
```



Combining plots with `patchwork`

“collect” common legends

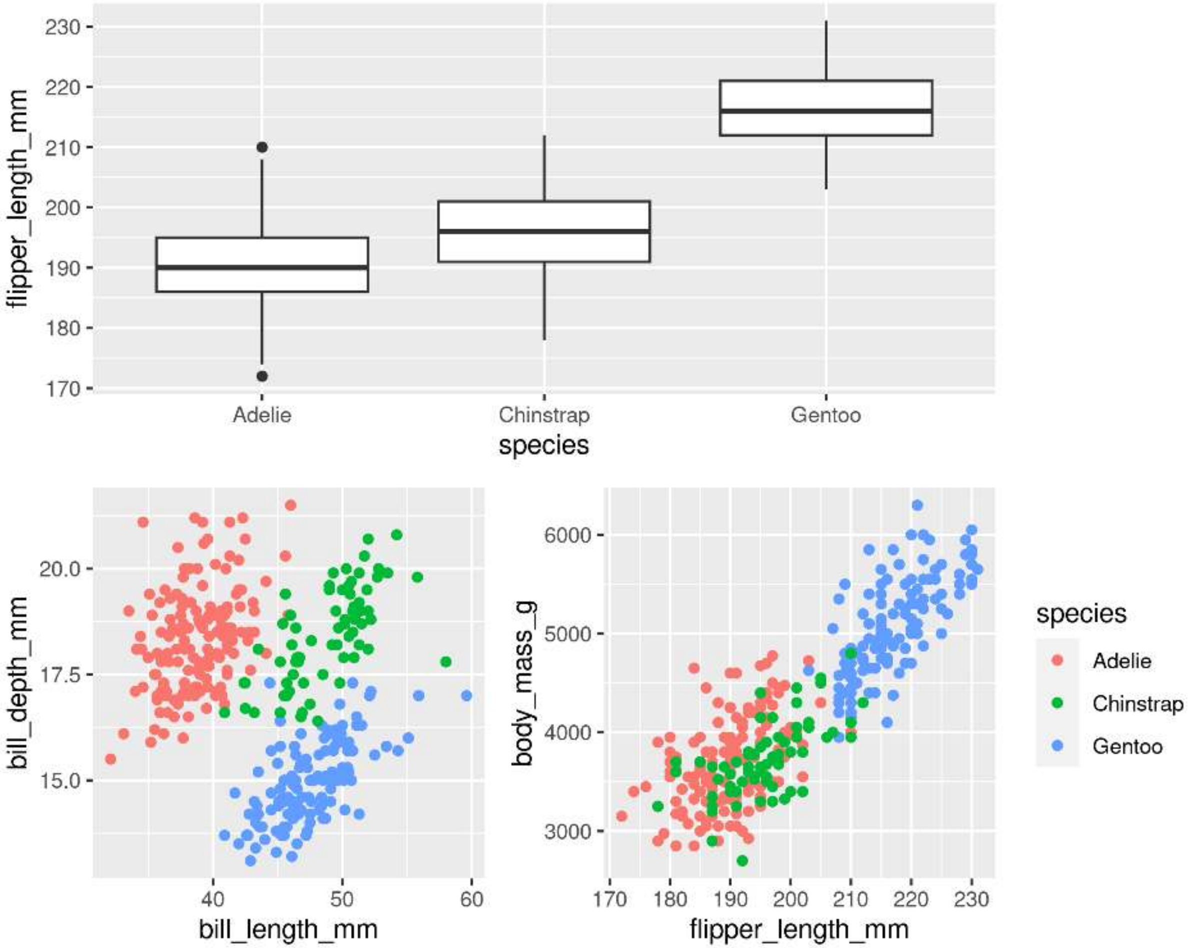
```
g2 / (g1 + g3) + plot_layout(guides = "collect")
```



Combining plots with `patchwork`

"collect" common legends

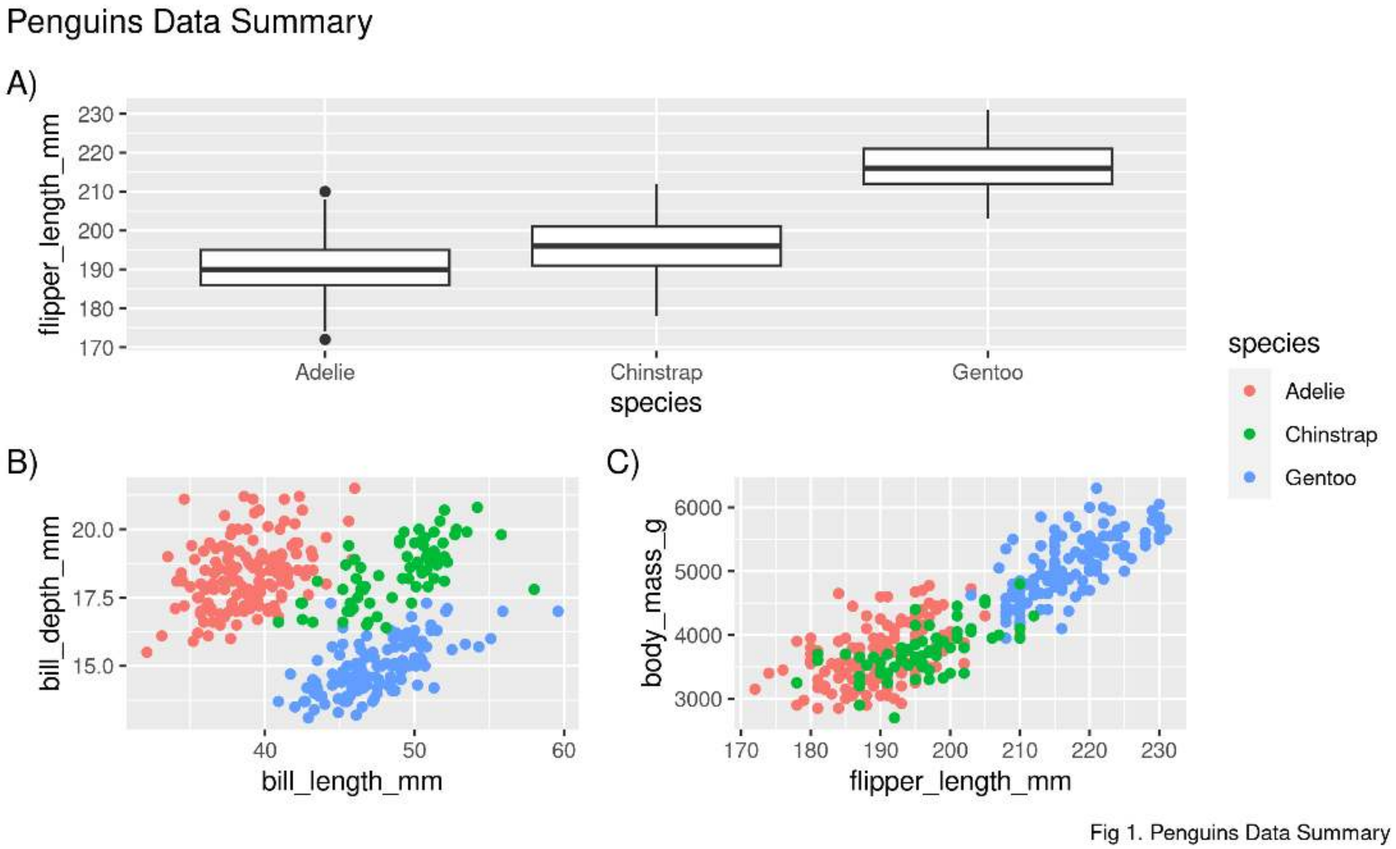
```r
g2 / (g1 + g3 + plot_layout(guides = "collect"))
```



Combining plots with `patchwork`

Annotate

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



Saving plots

Saving plots

RStudio Export

Demo

`ggsave()`

```r
g <- ggplot(penguins, aes(x = sex, y = bill_length_mm, fill = year)) +
  geom_boxplot()

ggsave(filename = "penguins_mass.png", plot = g)
```

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

```r
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

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()`
  - e.g. `ggplot(data = penguins, aes(x = factor(year), y = body_mass_g))`

Thank you!

Dr. Steffi LaZerte

Analysis and Data Tools for Science

🌐 steffilazerte.ca

✉ sel@steffilazerte.ca

Extra

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

😁

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