

Getting Started with R

Back to Basics

 [steffilazerte](#)
 @steffilazerte@fosstodon.org
 [@steffilazerte](#)
 [steffilazerte.ca](#)

Dr. Steffi LaZerte 
Analysis and Data Tools for Science



Compiled: 2024-02-13

These are me and my creatures 🐱



This is my garden 🌲



Introductions

Dr. Steffi LaZerte

- Background in Biology (Animal Behaviour)
- Working with R since 2007
- Professional R programmer/consultant since 2017
- [rOpenSci](#) Community Assistant



What about you?

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



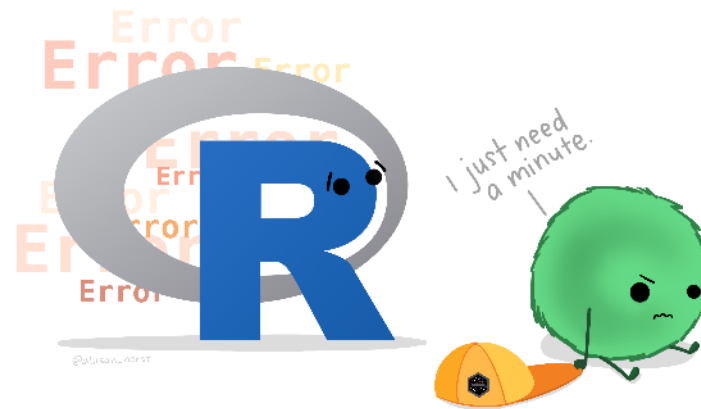
About this Workshop

Format

- I will provide you tools and workflow to get started with R
- We'll have hands-on activities, lectures, and demonstrations

R is hard: But have no fear!

- **Don't expect to remember everything!**
- Copy/Paste is your friend (never apologize for using it!)
- Consider this workshop a resource to return to



What is R?

RStudio vs. R



RStudio



R

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

Open RStudio

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?

```
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
```

R, save this value for later

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

R, multiply this value by 6

```
steffis_mean * 6
```

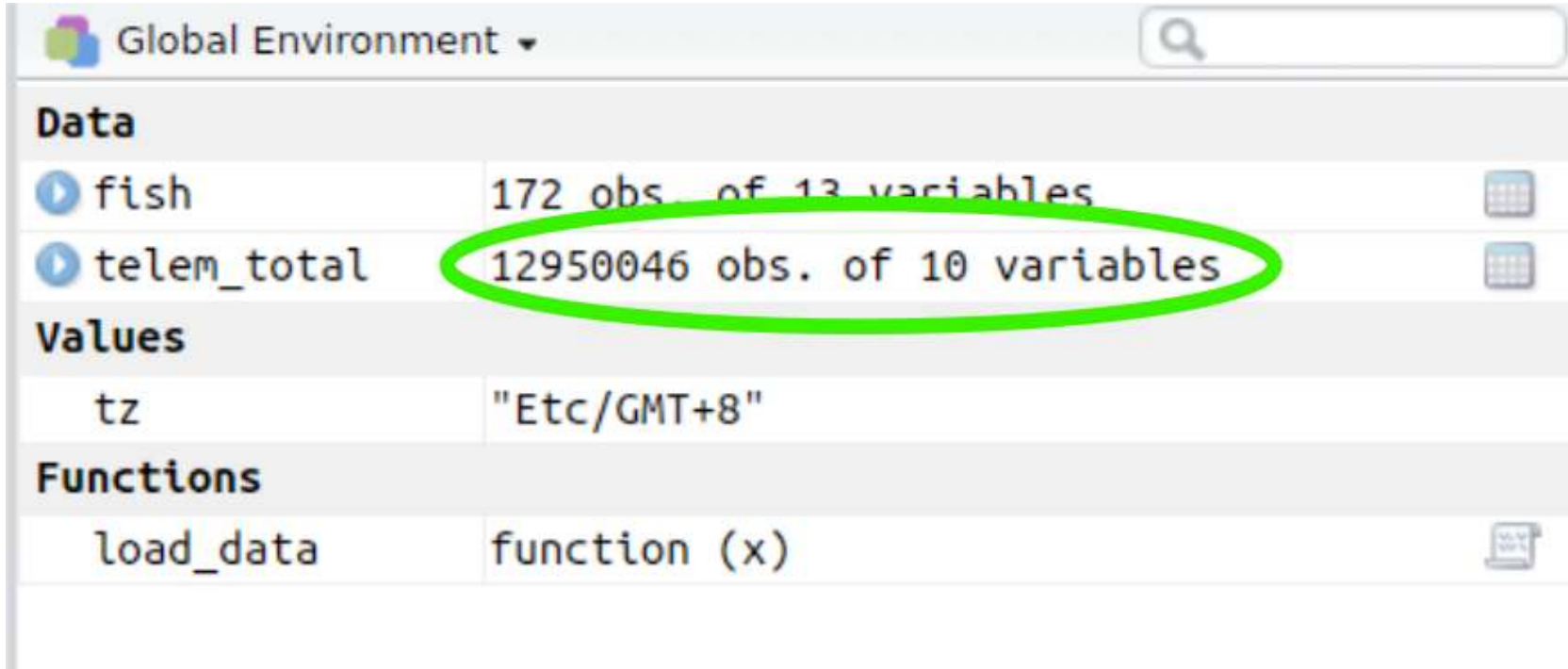
```
[1] 15
```

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],
```

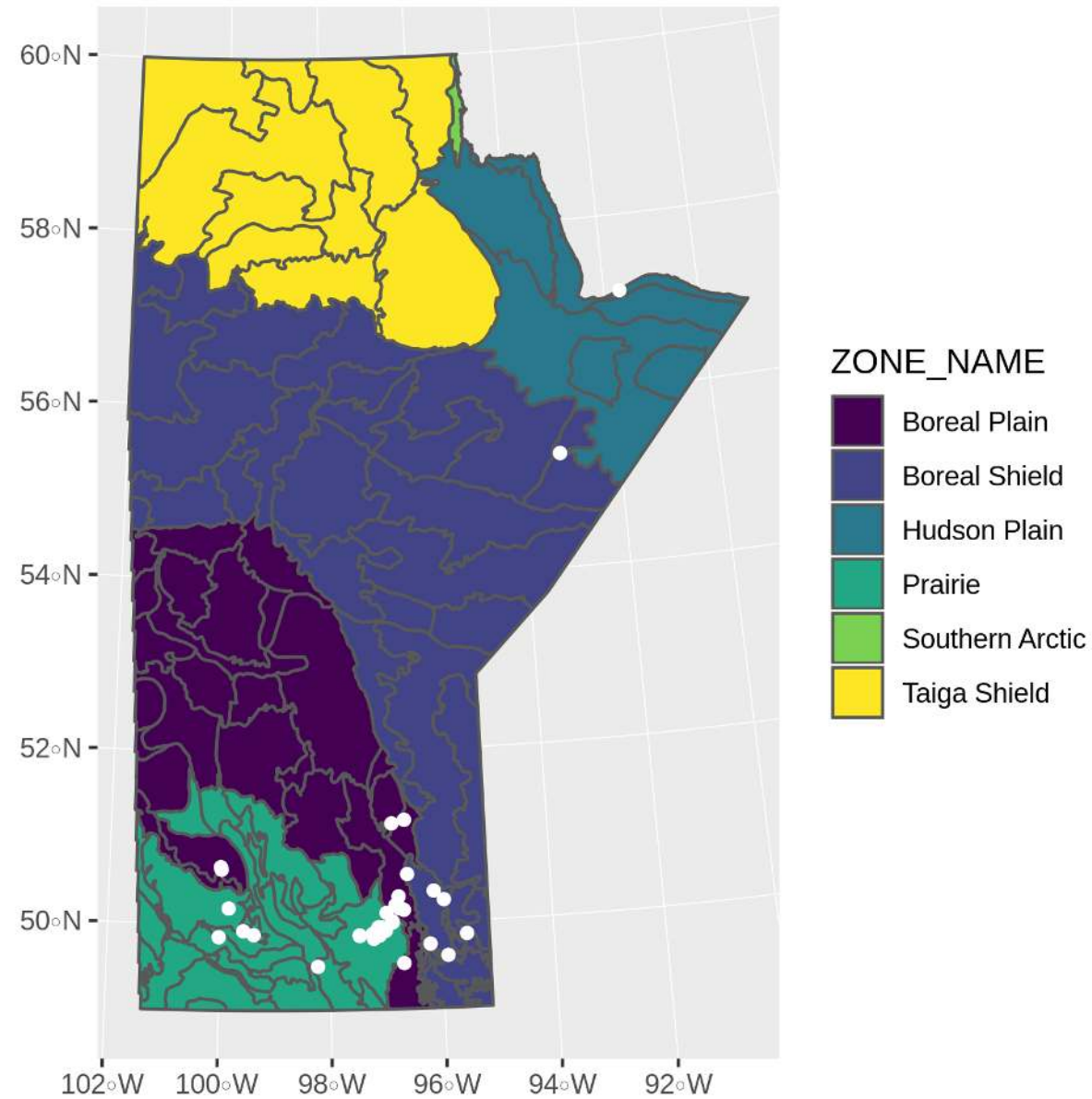
But R is powerful (and reproducible)!



The screenshot shows the R Global Environment window. At the top, there is a search bar and a dropdown menu labeled 'Global Environment'. Below this, the environment is organized into sections: 'Data', 'Values', and 'Functions'. In the 'Data' section, two objects are listed: 'fish' with 172 observations and 13 variables, and 'telem_total' with 12950046 observations and 10 variables. The 'telem_total' entry is circled in green. In the 'Values' section, the 'tz' variable is shown with the value 'Etc/GMT+8'. In the 'Functions' section, the 'load_data' function is shown as a function of 'x'.

Global Environment	
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



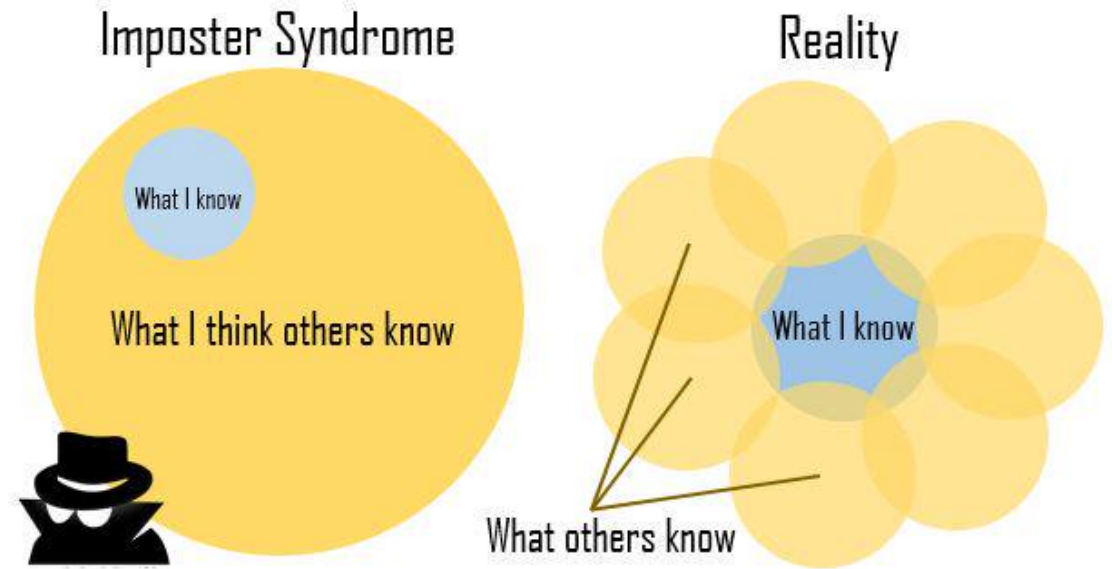
R is affordable (i.e., free!)

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.

Impostor Syndrome

A large, stylized logo of the letter 'R' in blue, with a gray oval shape behind it, positioned behind the word 'Impostor'.

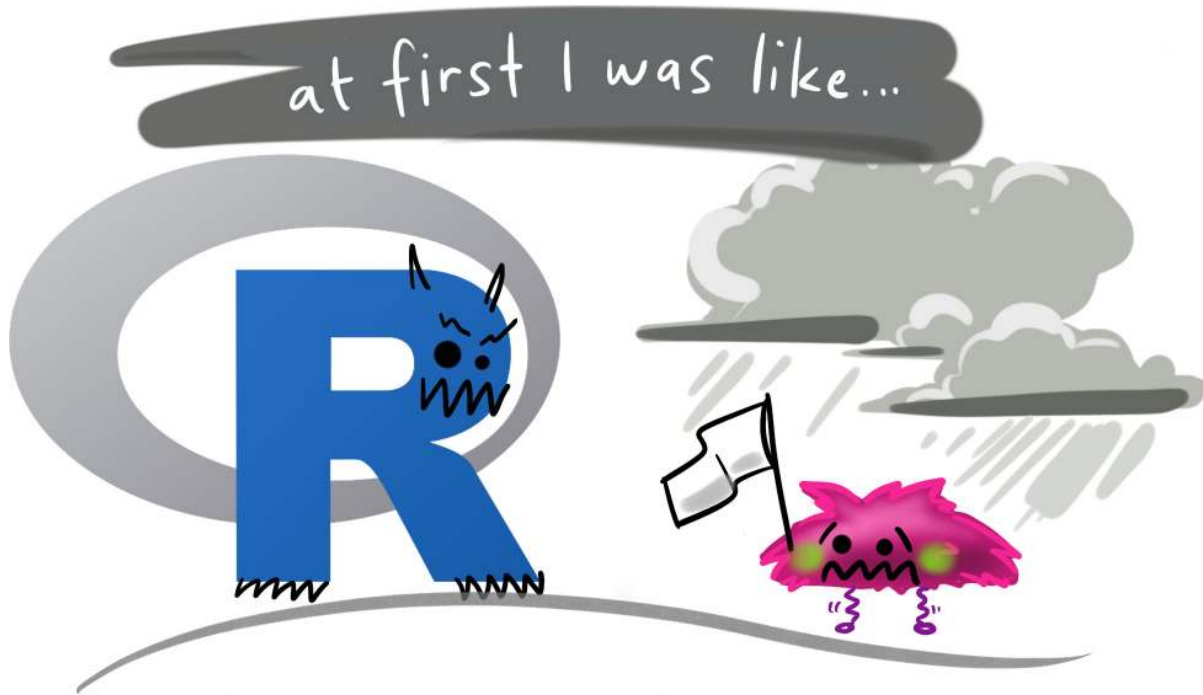
Impost R Syndrome



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:

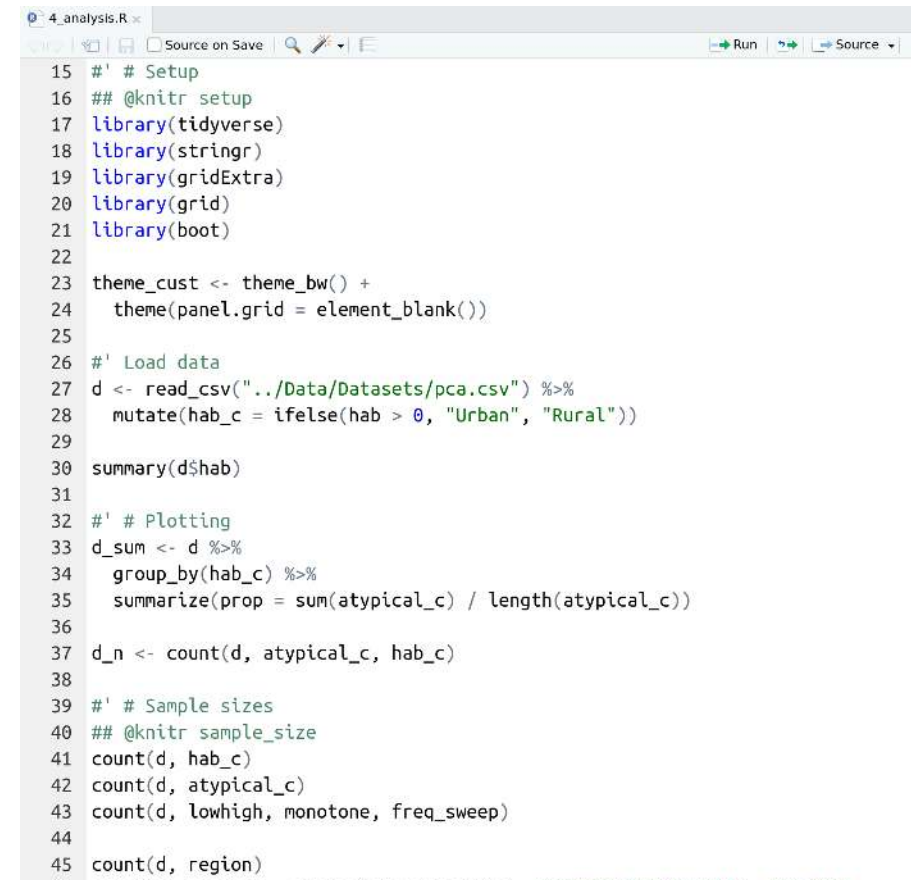
```
mean(c(1, 2, 3, 4))
```

Code

```
[1] 2.5
```

Output

Script



```
4_analysis.R x
Source on Save
Run
Source

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)
```

RStudio Features

Projects

- Handles working directories
- Organizes your work

Changing Options: Tools > Global Options

- General > Restore RData into workspace at startup (NO!)
- General > Save workspace to on exit (NEVER!)
- Code > Insert matching parens/quotes (Personal preference)

Let's change some options
in RStudio!

Packages

- Can use the package manager to install packages
- Can use the manager to load them as well, but not recommended

Getting Ready

 Open New File

(make sure you're in the RStudio Project)

 Write `library(tidyverse)` at the top

 Save this new script

(consider names like `intro.R` or `1_getting_started.R`)

Your first *real* code!

First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   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

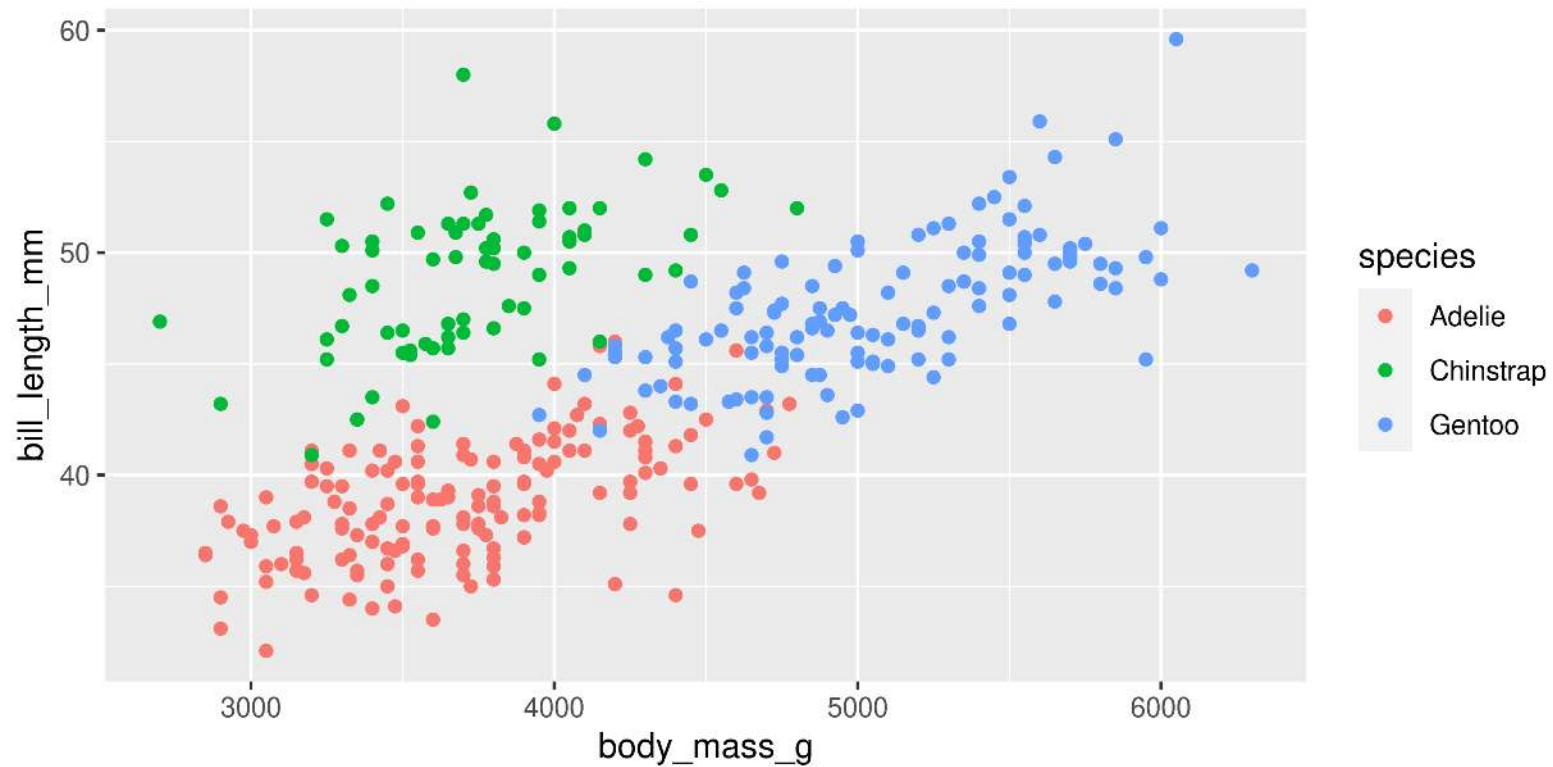
- Click 'Run' button (upper right) **or**
- Use the short-cut **Ctrl-Enter**

4. Repeat until all the code has run

First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

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

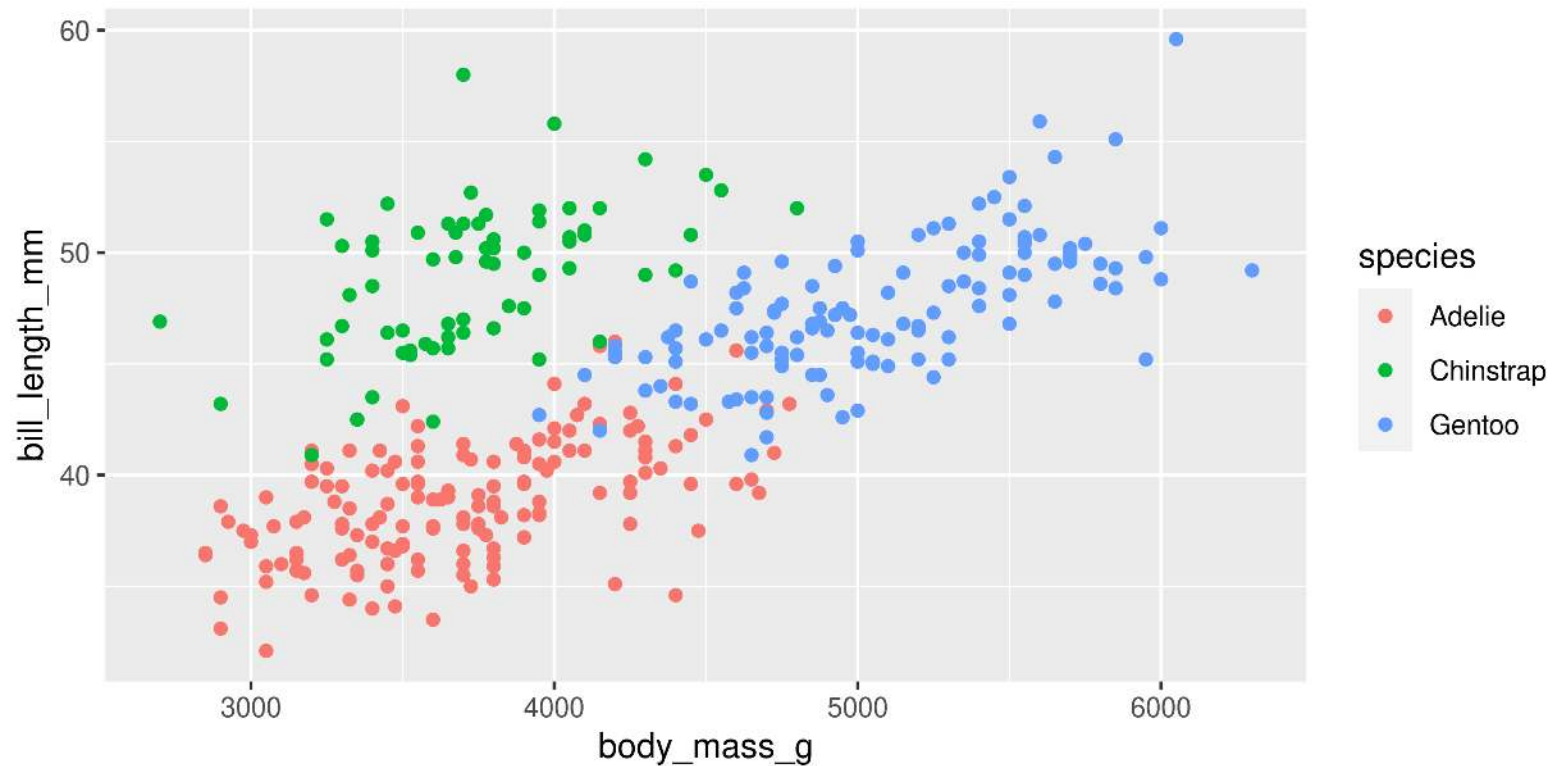


First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

Packages
ggplot2 and palmerpenguins

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

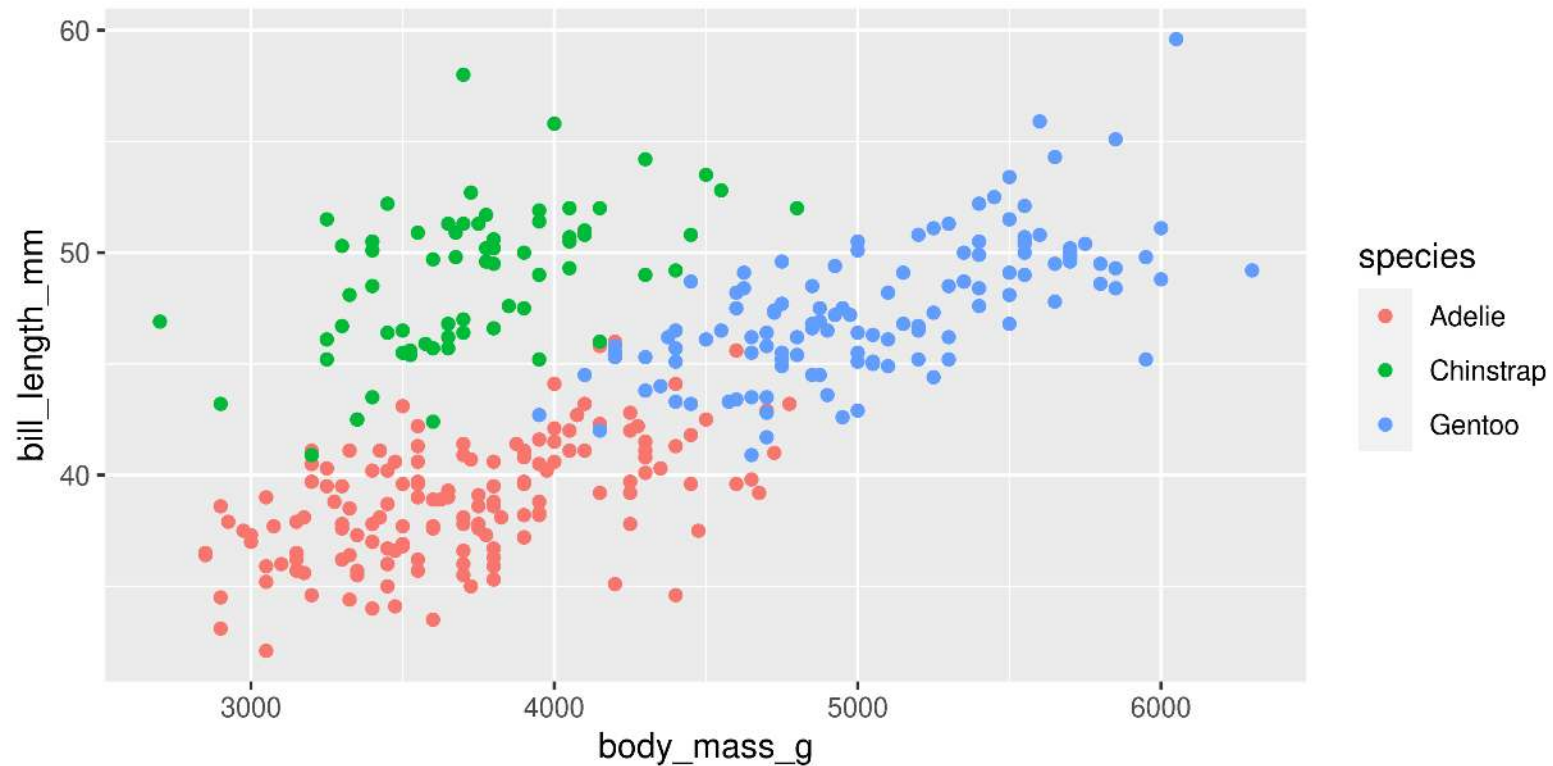


First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

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

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

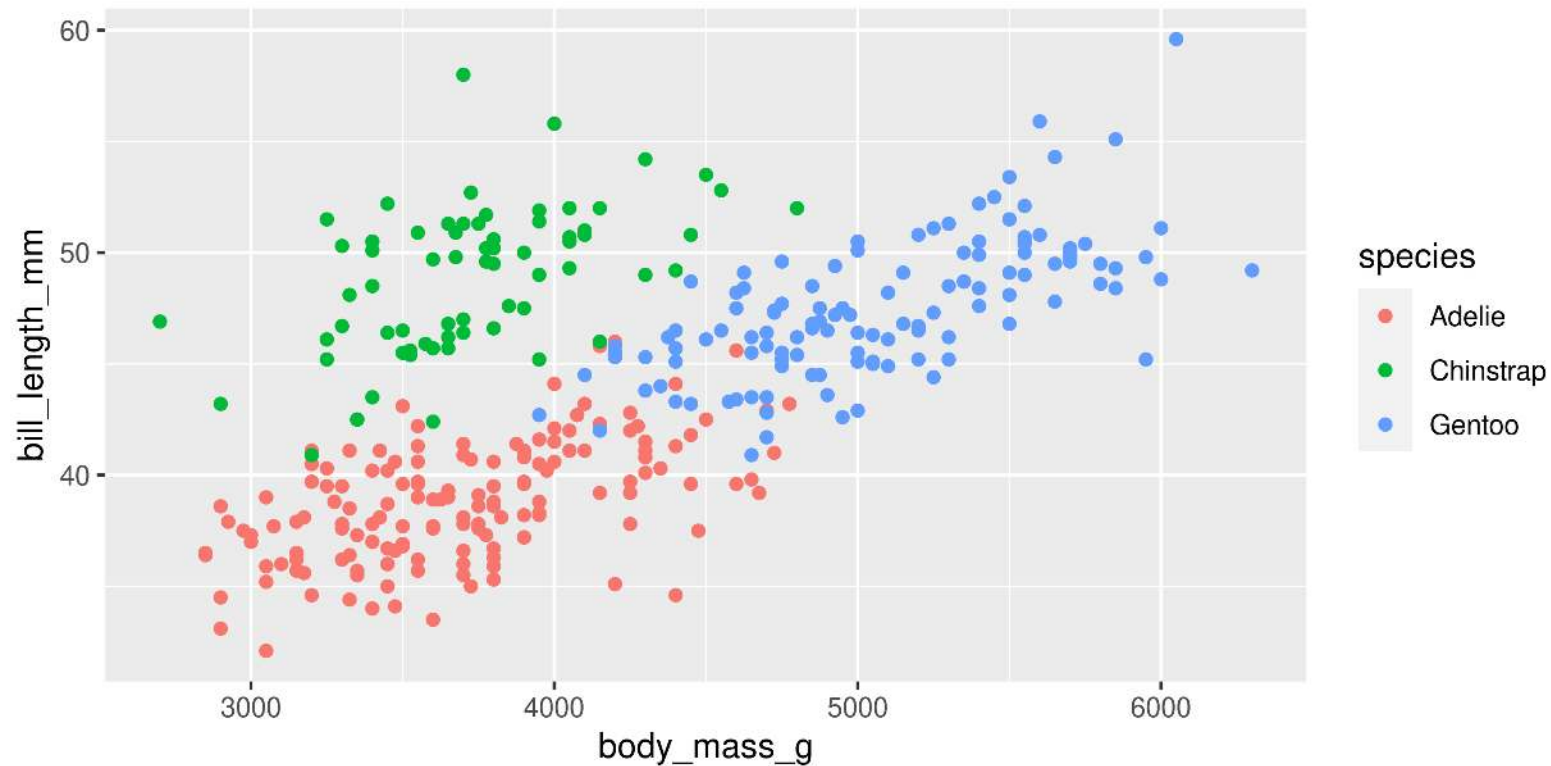


First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

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

+
(Specific to
ggplot)

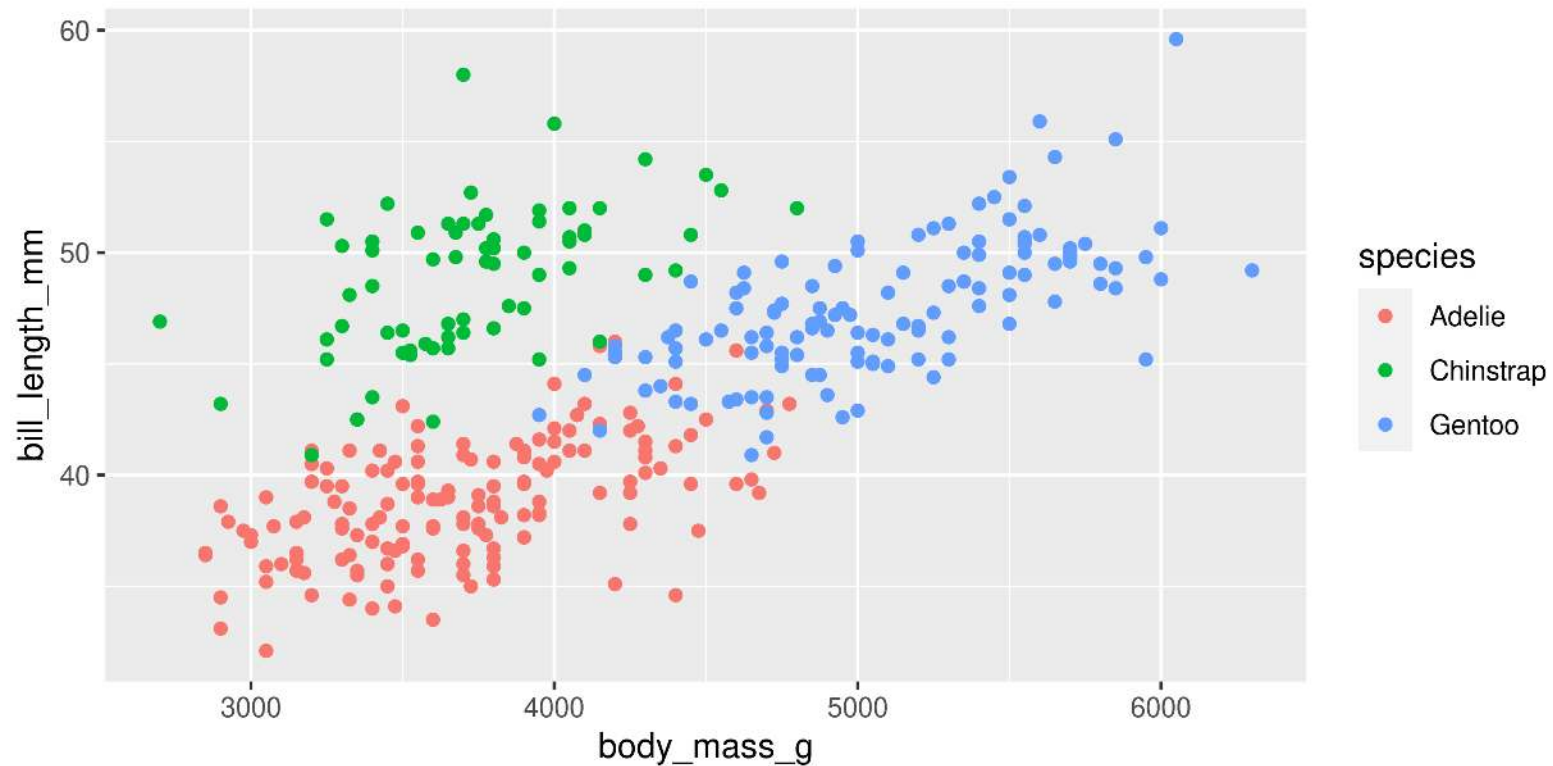


First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

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

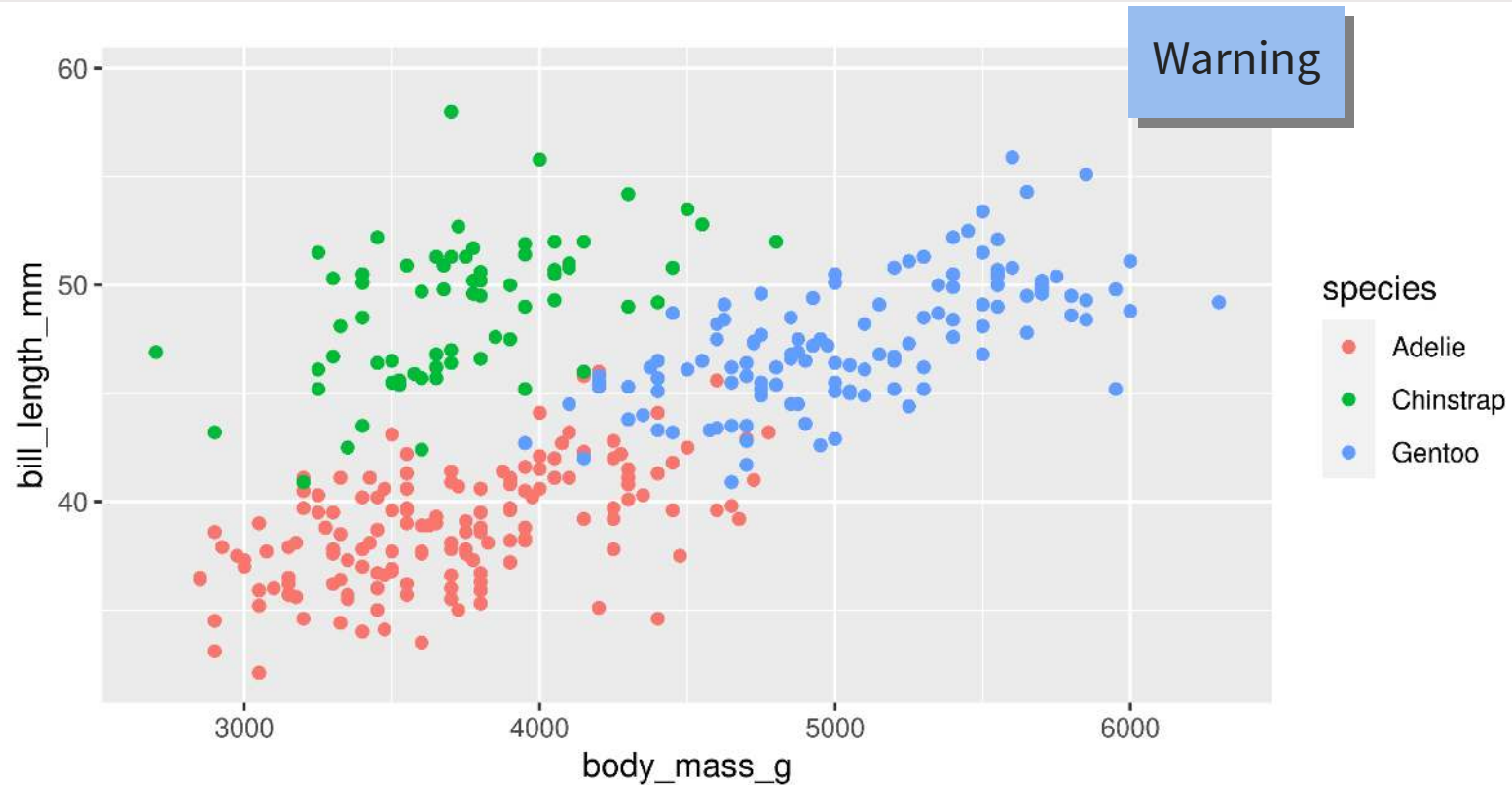
Figure!



First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

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

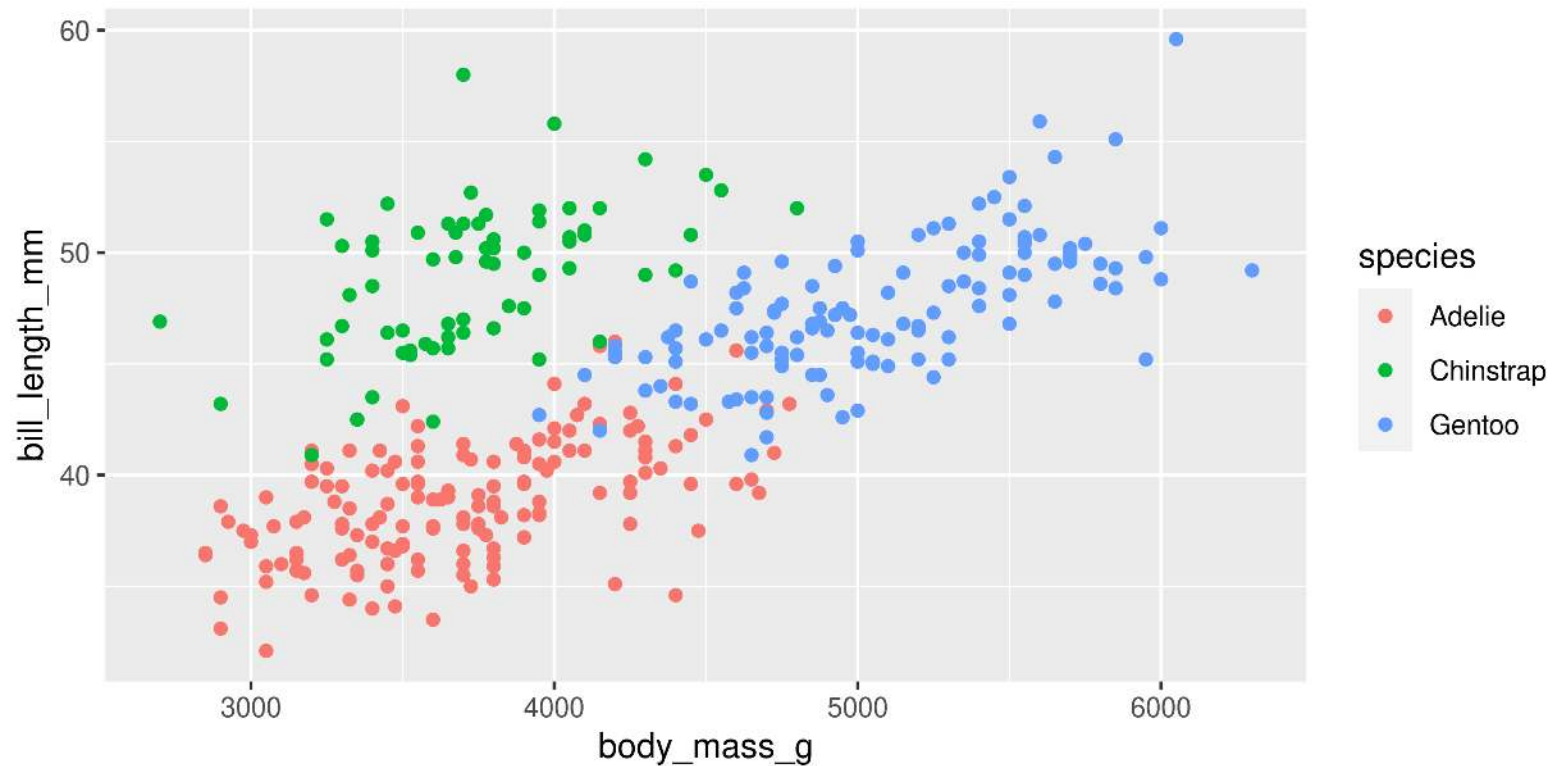


First Code

```
1 # First load the packages
2 library(palmerpenguins)
3 library(ggplot2)
4
5 # Now create the figure
6 ggplot(data = penguins, aes(x = body_mass_g, y = bill_length_mm, colour = species)) +
7   geom_point()
```

Comments

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



R Basics: Objects

Objects are *things* in the environment
(Check out the **Environment** pane in RStudio)

functions()

Do things, Return things

Does something but returns nothing

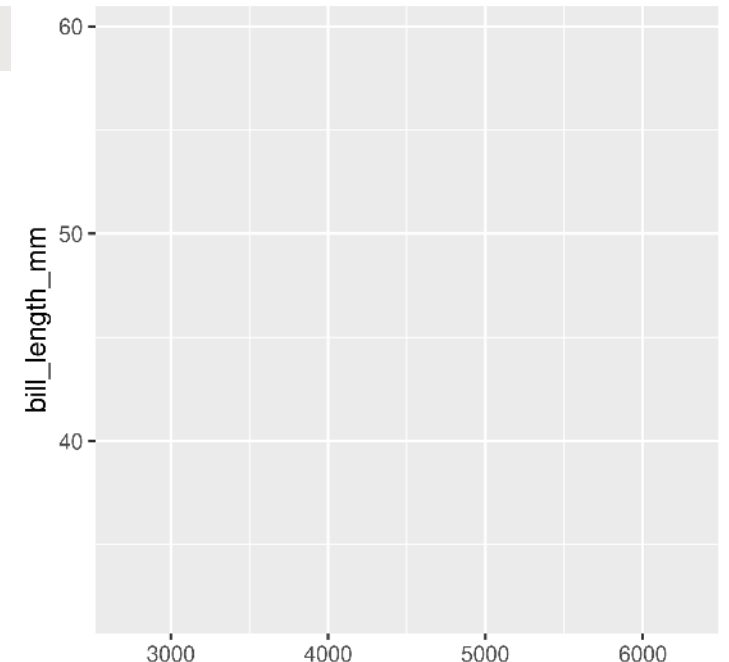
e.g., `library()` - Loads an R package so we can use its functions and other objects it supplies

```
1 library(palmerpenguins)
```

Does something and returns something

e.g., `ggplot()` - Creates and returns a basic plot

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



functions()

- Functions can take **arguments** (think ‘options’)
- **data, x, y, colour**

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

- Arguments defined by **name** or by **position**
- With correct position, do not need to specify by name

By name:

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

```
[1] 5.333333
```

By order:

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

```
[1] 5.333333
```

functions()

Watch out for 'hidden' arguments

By name:

```
1 mean(x = c(1, 5, 10, NA),  
2      na.rm = TRUE)
```

```
[1] 5.333333
```

By order:

```
1 mean(c(1, 5, 10, NA),  
2      TRUE)
```

```
Error in mean.default(c(1, 5, 10, NA), TRUE): 'trim' must be  
numeric of length one
```

This error states that we've assigned the argument `trim` to a non-valid argument

Where did `trim` come from?

R documentation

1 ?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

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

Data

- Generally kept in `vectors` or `data.frames` (also `tibbles`)
- These are objects with names (like functions)
- Here are two **built-in** examples (part of R)

Vector (1 dimension)

1	month.name
[1]	"January" "February" "March"
[4]	"April" "May" "June"
[7]	"July" "August" "September"
[10]	"October" "November" "December"

Data frame (2 dimensions)

1	mtcars
	mpg cyl disp hp drat wt qsec vs
	Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0
	Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0
	Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1
	Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1
	Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0
	Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1
	Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0
	Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1
	Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1
	Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1
	Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1
	Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0
	Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0
	Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0
	Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0
	Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0
	Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0
	Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1
	Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1
	Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1
	Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1
	Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0
	AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0
	Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0
	Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0
	Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1
	Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0
	Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1
	Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.50 0
	Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0

- Columns have different types of variables

Your Turn: Vectors and Data frames

Try out the following code...

- Here we will make a vector and a data frame
- What is the output in your console?
- How does your `environment` change (upper right panel)?

Vectors

```
1 a <- c("apples", 12, "pears", 5, 8)
2 a
```

Data frames

```
1 my_data <- data.frame(x = c("s1", "s2", "s3", "s4"),
2                       y = c(101, 102, 103, 104),
3                       z = c("a", "b", "c", "d"))
4 my_data
```

Your Turn: Vectors and Data frames

Try out the following code...

- What does `:` do?
- What does `c()` do?
- Why use a comma with data frames?

Vectors

- Use `[index]` to access part of a vector
- Can access multiple parts at once

```
1 a[2]
2 a[2:5]      # What does : do?
3 a[c(1, 3)] # What does c() do?
```

Data frames

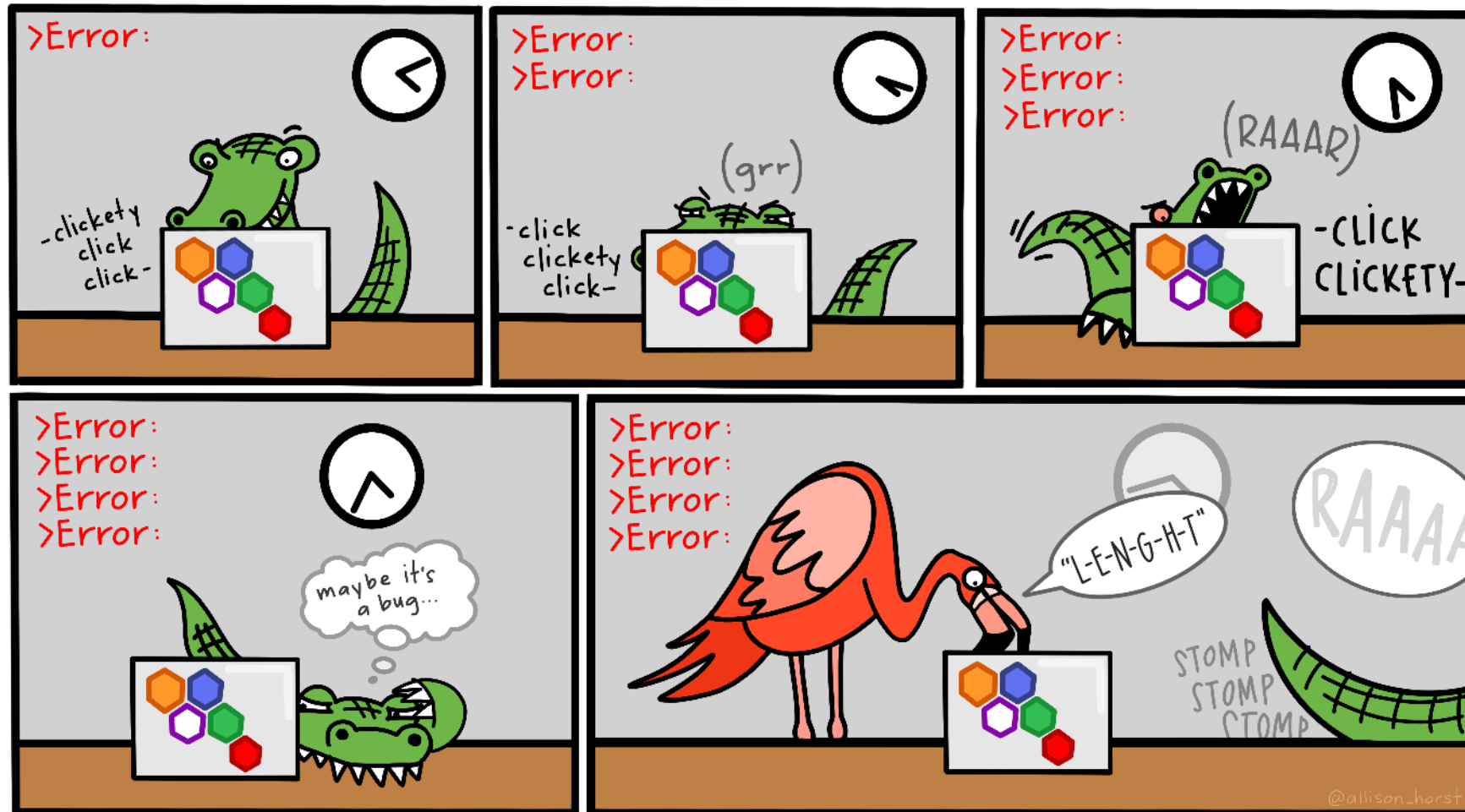
- `x$colname` to pull columns out as vector
- `x[row, col]` to access rows/columns

```
1 my_data[3, ] # Why the comma?
2 my_data[3, 1]
3 my_data[, 1:2]
```


Miscellaneous

R has spelling and punctuation

- R cares about spelling
- R is also case sensitive! (AppLe is not the same as apple)



R has spelling and punctuation

- Commas are used to separate arguments in functions

This is correct:

```
1 mean(c(5, 7, 10)) # [1] 7.333333
```

This is **not** correct:

```
1 mean(c(5 7 10))
```

>80% of learning R is learning to **troubleshoot!**

R has spelling and punctuation

Spaces usually don't matter unless they change meanings

```
1 5>=6      # [1] FALSE
2 5 >=6      # [1] FALSE
3 5 >= 6     # [1] FALSE
4 5 > = 6    # Error: unexpected '=' in "5 > ="
```

Periods don't matter either, but can be used in the same way as letters

(But don't)

```
1 apple.oranges <- "fruit"
```

Assignments and Equal signs

Use **<-** to assign values to objects

```
1 a <- "hello"
```

Use **=** to set function arguments

```
1 mean(x = c(4, 9, 10))
```

Use **==** to determine equivalence (logical)

```
1 10 == 10 # [1] TRUE
2 10 == 9  # [1] FALSE
```


Braces/Brackets

Round brackets: ()

- Identify functions (even if there are no arguments)

```
1 Sys.Date() # Get the Current Date
```

```
[1] "2024-02-13"
```

- Without the (), R spits out information on the function:

```
1 Sys.Date
```

```
function ()  
as.Date(as.POSIXlt(Sys.time()))  
<bytecode: 0x55b55841d340>  
<environment: namespace:base>
```

() must be associated with a **function** (Well, *almost* always)

Square brackets: []

- Extract parts of objects

```
1 LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
1 LETTERS[1]
```

```
[1] "A"
```

```
1 LETTERS[26]
```

```
[1] "Z"
```

[] have to be associated with an **object** that has dimensions (Always!)

Improving code readability

Use spaces like you would in sentences:

```
1 a <- mean(c(4, 10, 13))
```

is easier to read than

```
1 a<-mean(c(4,10,13))
```

(But the same, coding-wise)

Improving code readability

Don't be afraid to use line breaks ('Enters') to make the code more readable

Hard to read

```
1 a <- data.frame(exp = c("A", "B", "A", "B", "A", "B"), sub = c("A1", "A1", "A2", "A2", "A3", "A3"), res = c(10,
```

Easier to read

```
1 a <- data.frame(exp = c("A", "B", "A", "B", "A", "B"),  
2               sub = c("A1", "A1", "A2", "A2", "A3", "A3"),  
3               res = c(10, 12, 45, 12, 12, 13))
```

(But the same, coding-wise)

Let's go!

