INTRODUCTION TO DATA SCIENCE WITH R

Session 3: Data exploration part 2

Ina Bornkessel-Schlesewsky

January 19, 2023

ROADMAP FOR TODAY

DISCUSSION OF THE EXERCISES

- Meet `select` and `mutate`
- What's up with the `size` aesthetic?

LET'S GET TO KNOW R A LITTLE BETTER!

- R programming basics
- R workflow basics

MORE ON DATA EXPLORATION

- A bit more on summarising data
- Plots galore
- Try your hand at a new data set

R PROGRAMMING BASICS

The following closely mirrors R4DS chapter 3

USING R AS A CALCULATOR

```r
3+4
```

```
[1] 7
```

```r
1 / 200 *30
```

```
[1] 0.15
```

```r
sin(pi / 2)
```

```
[1] 1
```

```r
sqrt(16)
```

```
[1] 4
```

Try out a few calculations in your RStudio console!

CREATING NEW OBJECTS

- Meet the assignment operator `<-`
- We can use this to create new objects using the general schema:

> `object_name <- value`

- RStudio shortcut: Option / Alt + - (i.e. the minus sign)
- Type an object's name to inspect it
- Or enclose the assignment in parentheses to show it automatically
- You can perform calculations using objects

- But make sure that they're of the right type for the operation that you want to perform!
- e.g. try running `b + c` in your console once you have created them

```r
a <- 1 + 2
b <- 3 + 4
c <- "objects can be of different types"
c
```

```
[1] "objects can be of different types"
```

```r
(d <- "another new string")
```

```
[1] "another new string"
```

```r
a + b
```

```
[1] 10
```

HOW TO NAME NEW OBJECTS

- Object names should be descriptive, i.e. allow you to quickly ascertain (or remember) what the object contains.
- They must start with a letter and can only contain letters, numbers, underscores and full stops
- You will need a way of naming objects using multiple words (no spaces allowed); here are a few options:
    - snake_case (used in most of the materials for this course)
    - camelCase (used in the `gapminder` examples that you will be working with as an exercise later)
    - you.can.also.use.full.stops
- Avoid spaces when naming variables (and when naming files) – it will make your life easier!

CODE COMPLETION

- RStudio’s code completion function is useful for selecting an object that you want to inspect
- Create the following object:

```
this_is_a_really_long_name <- 2.5
```

- To try out code completion, type `this`, press Tab and see what happens
- If the start of the object name is not unique, you will need to add more letters / select from the list
- Create another object called `this_is_too` and then try code completion again

GOING BACK TO PREVIOUS COMMANDS

- Let's say you want to change the value of `this_is_a_really_long_name` to `3.5`
- You can go back through the history of the commands typed into the console by pressing the "up" arrow key (Note: this only works with commands typed directly into the console)
- Select the command you want and then change it
- You can also search through the commands that you have run by typing `this` and pressing Cmd/Ctrl + up_arrow
- Use one of these methods to change the value of `this_is_a_really_long_name`

BE PRECISE WITH OBJECT NAMES

```r
r_rocks <- 2^3
colour <- "blue"
```

- If you try to inspect these objects but aren't completely precise in your spelling, you will get an error message
- For example, see what happens if you type in `r_rock` or `R_rocks` or if you use the American spelling of `color`
- Typos matter, case matters, spelling matters …
- This may seem complicated at first, but you will get the hang of it with a bit of practice!

CALLING FUNCTIONS

- R has many built-in functions (you have already seen a few, e.g. `ggplot`, `filter` etc.)
- Functions “do things” with objects; they are thus often verbs expressing the action
- Functions are called like this:

> function_name(arg1 = val1, arg2 = val2, …)

- The arguments that a function takes differ depending on the function

AN EXAMPLE: THE *SEQ()* FUNCTION

- Code completion is helpful here too: type `se` and hit Tab
- Select `seq()` by typing `q` or using the arrow keys
- Note that RStudio also gives you a bit of information about the function as you do this
- Once you have selected the right function, hit Tab again
- Note that RStudio already matches the opening and closing parentheses for you

```r
seq(1,10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Get help on this function by typing `?seq`

FURTHER ASSISTANCE WITH WRITING CODE

- Type the following:

```r
x <- "hello world"
```

- Note how RStudio again provides the matching second set of quotes for you
- You always need to have paired parentheses or quotes
- Otherwise, R will tell you that something is missing
    - try typing `x <- "hello`
    - the `+` shown in the console is the "continuation character", telling you that something is missing
    - this is usually a `"` or a `)`
    - add the missing character and hit Enter (or press Esc to exit)

FURTHER ASSISTANCE WITH WRITING CODE

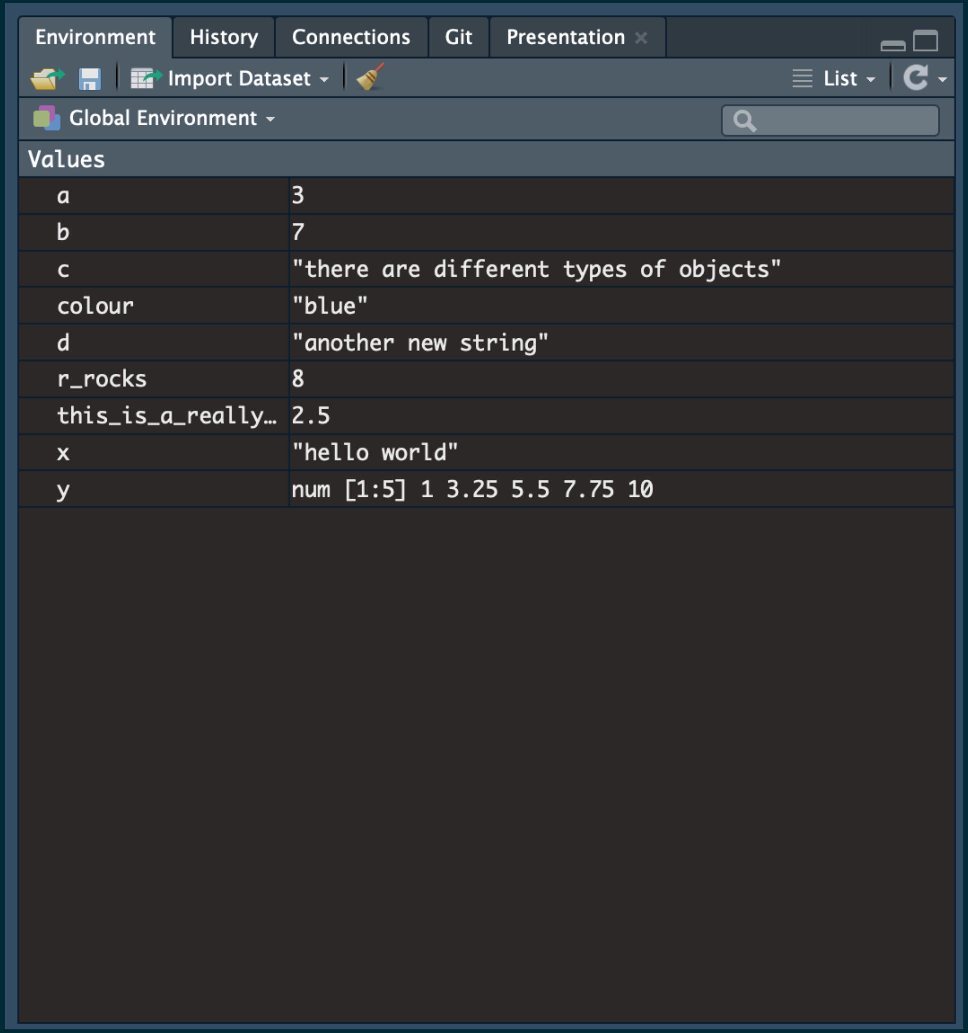
- Recall that you can enclose an assignment in parentheses to print to screen; this is particularly useful for functions, where you will usually want to check the result

```r
(y <- seq(1,10, length.out = 5))
```

```
[1]  1.00  3.25  5.50  7.75 10.00
```

INSPECTING OBJECTS IN YOUR WORKSPACE

The Environment panel shows all of the objects that you have created and their values. It is located in the upper-right pane of RStudio by default, but if you have changed your pane layout to match mine, it will be in the bottom-left pane.



```
a                  3
b                  7
c                  "there are different types of objects"
colour             "blue"
d                  "another new string"
r_rocks            8
this_is_a_really…  2.5
x                  "hello world"
y                  num [1:5] 1 3.25 5.5 7.75 10
```

REPLICABILITY VS. REPRODUCIBILITY …

… and consequences for workflow

REPLICABILITY

A study is replicable if the same results are obtained when the study is repeated.

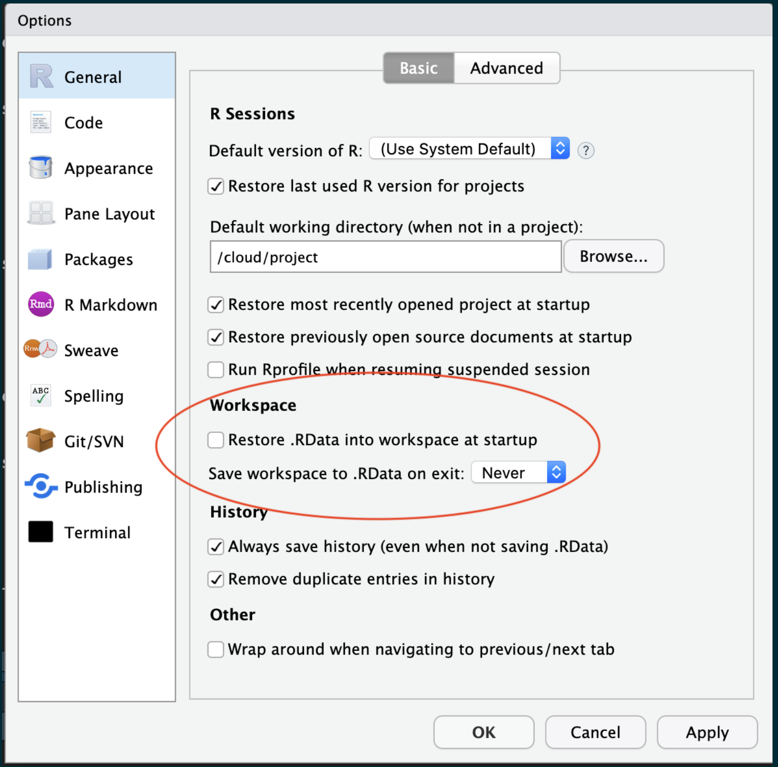
REPRODUCIBILITY

An analysis is reproducible if the same results can be obtained from the data

- By documenting your analysis in an R script, you have already taken an important first step towards reproducibility.
- While it is perfectly ok to try out different things on the console (and this helps while you're learning), get in the habit of pasting what works into a script to document your analysis.
- Better still: work directly from a Quarto document as we have been doing

WORKFLOW

- You should be able to reproduce the results that you got / the graphs that you generated etc. by simply running your script.
- You can try this by restarting R (Cmd/Ctrl-Shift-0 or Session > Restart R): this resets everything so that none of the products of your previous computations are available; then rerun the current script (Cmd/Ctrl-Shift-S)
- This may seem like extra work to start with, but in addition to being good practice, future you will thank you for it (believe me)!
- Make sure that RStudio doesn't save objects in your workspace from one session to the next



R Sessions

Default version of R: (Use System Default)

✓ Restore last used R version for projects

Default working directory (when not in a project): /cloud/project

✓ Restore most recently opened project at startup

✓ Restore previously open source documents at startup

☐ Run Rprofile when resuming suspended session

Workspace

☐ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

History

✓ Always save history (even when not saving .RData)

✓ Remove duplicate entries in history

Other

☐ Wrap around when navigating to previous/next tab

EXERCISES FOR YOU TO TRY

Complete exercises 1-3 in section 3.5 of R4DS

LIVE CODING EXERCISES 1: CREATING OBJECTS AND JUST A LITTLE MORE ON DATA MANIPULATION

LIVE CODING EXERCISES 2: A BIT MORE ON PLOTS

DIFFERENT TYPES OF GEOMS AND AESTHETICS

COMMON GEOMS

- `geom_point()` for scatterplots
- `geom_line()` for line graphs
- `geom_col()` for column or bar graphs
- `geom_histogram()` for histograms
- `geom_boxplot()` for boxplots

COMMON AESTHETICS

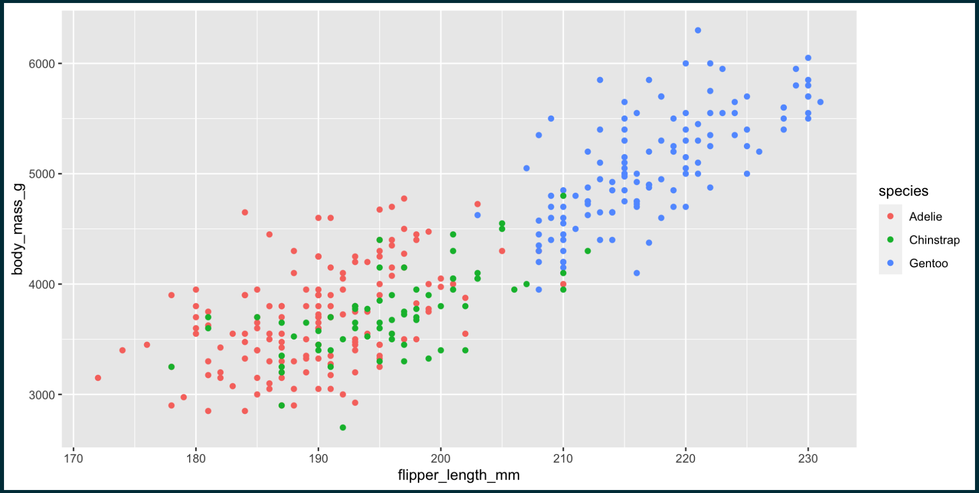
- *x*, *y* for axes
- *colour*, *fill* to add colour to lines and fill shapes with colour, respectively
- *shape*, *size* and *linetype* to vary pretty much what you would think given their names 😉

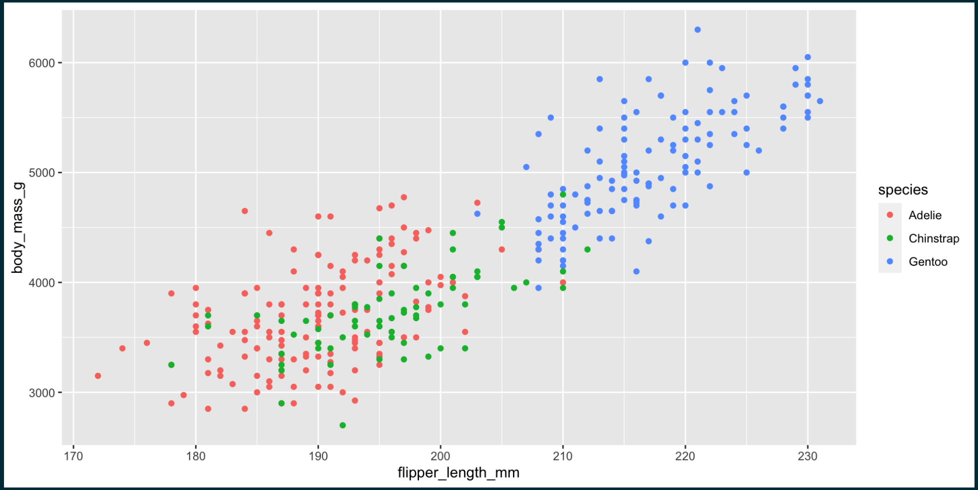
AESTHETICS

- Note that you can add aesthetics either to the call to ggplot (left example), in which case they apply to all applicable geoms or to the geom itself (right example), in which case they only apply to that geom (more on this later)

```r
penguins |>
  ggplot(aes(x = flipper_length_mm,
             y = body_mass_g,
             colour = species)) +
  geom_point()
```

```r
penguins |>
  ggplot(aes(x = flipper_length_mm,
             y = body_mass_g)) +
  geom_point(aes(colour = species))
```





SOOO MANY CHOICES ….!

THINGS TO CONSIDER WHEN CONSTRUCTING A PLOT

1. What question am I trying to answer?
2. Do I need to manipulate my data before plotting?
3. Which variables do I need to plot and what type are they? This will affect choice of axes, geoms and aesthetics

EXAMPLE (FROM DAY 1): WHICH OTHER ATTRIBUTES ARE PREDICTIVE OF PENGUINS' BODY MASS?

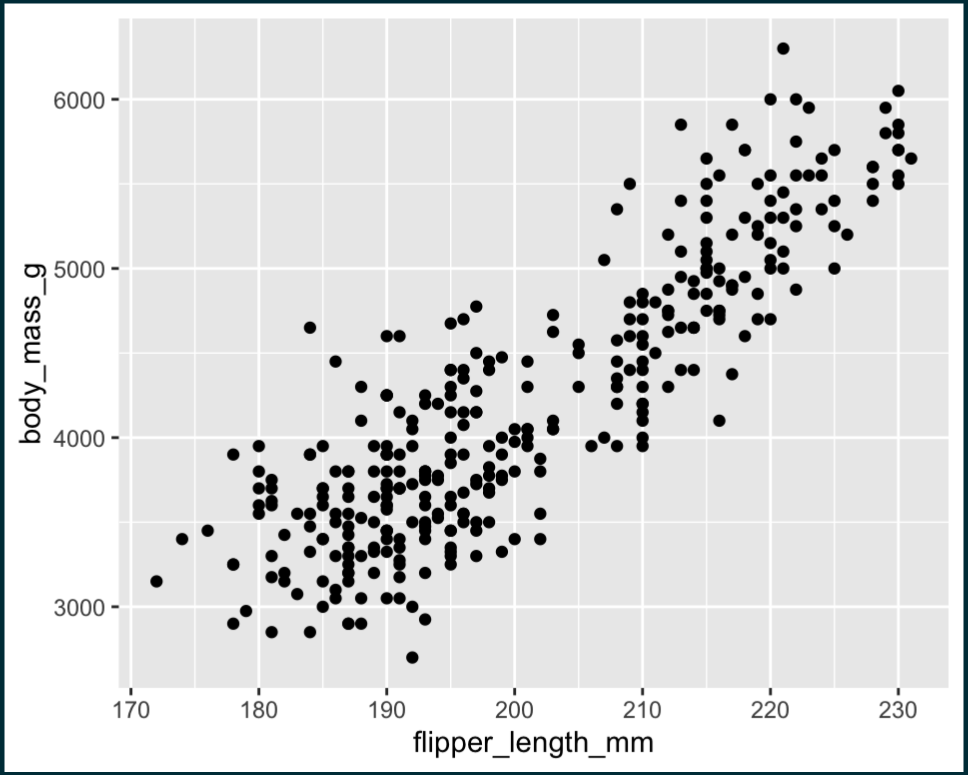
Let's start by looking at flipper length and go through our checklist.

1. Is flipper length predictive of body mass?
2. no
3. outcome variable = body_mass_g (numeric) -> y aes; predictor variable = flipper_length_mm (numeric) -> x aes; two numeric variables lend themselves to a scatterplot -> geom_point()

EXAMPLE 1: SCATTERPLOT

Is flipper length predictive of body mass?

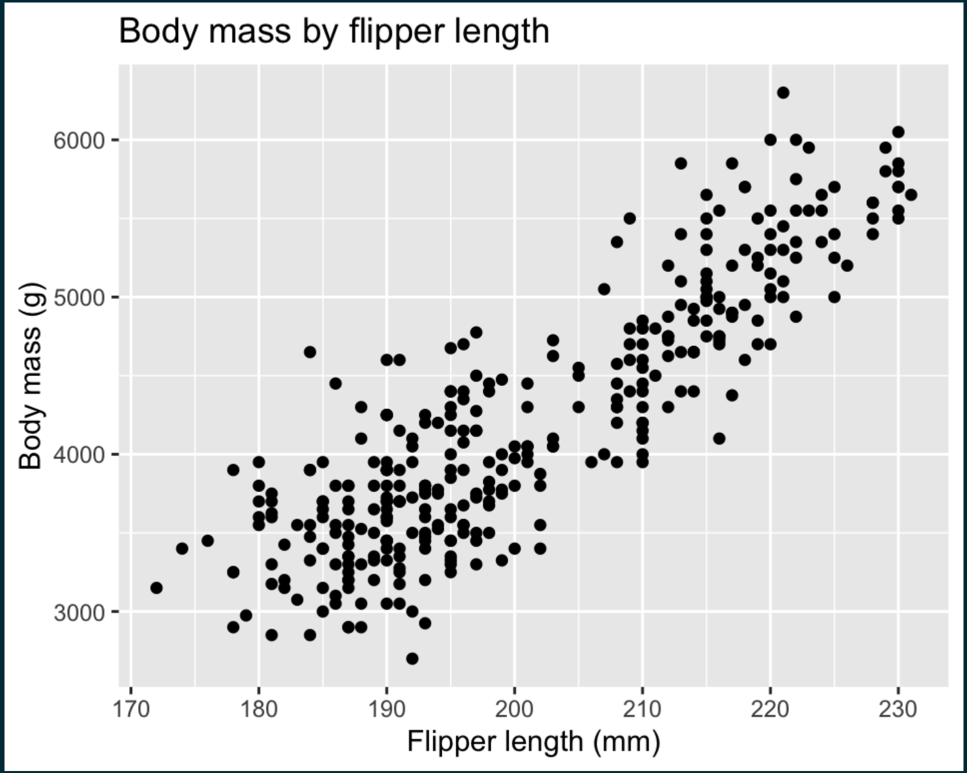
```
penguins |>
  ggplot(aes(x = flipper_length_mm,
             y = body_mass_g)) +
  geom_point()
```



EXAMPLE 1A: ADD TITLE AND AXIS LABELS

Is flipper length predictive of body mass?

```
penguins |>
  ggplot(aes(x = flipper_length_mm,
             y = body_mass_g)) +
  geom_point() +
  labs(
    title = "Body mass by flipper leng
    x = "Flipper length (mm)",
    y = "Body mass (g)"
  )
```

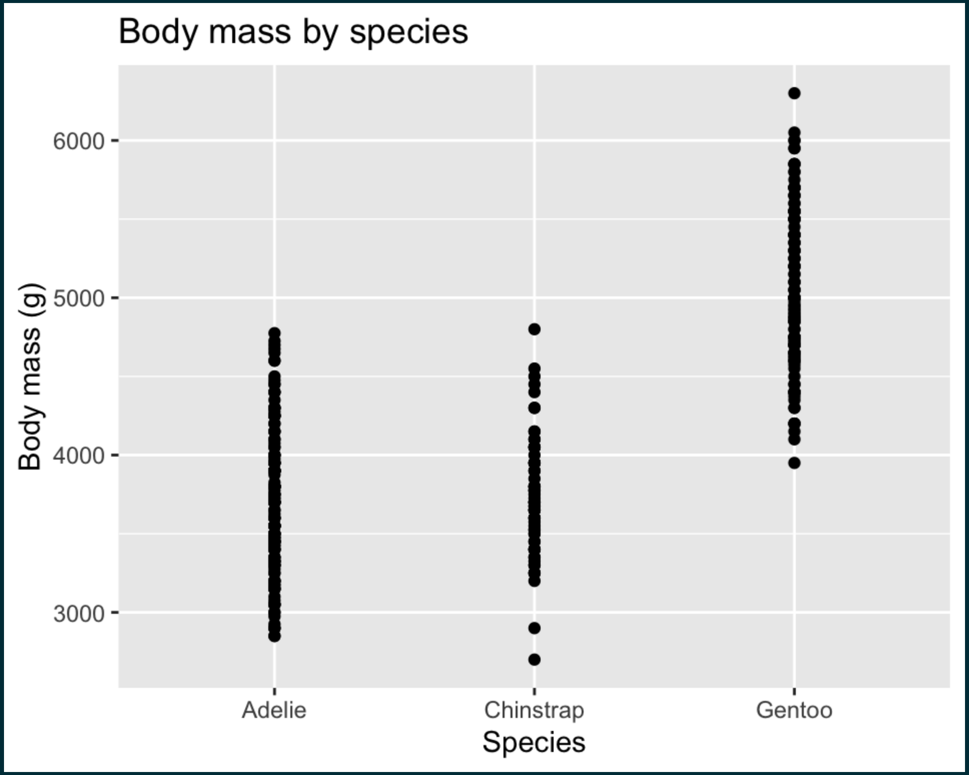


EXAMPLE 2: A CATEGORICAL PREDICTOR

Is species predictive of body mass?

No data manipulation needed, body_mass_g (outcome) -> y-axis, species (predictor) -> x-axis. Try using `geom_point()` again …

```r
penguins |>
  ggplot(aes(x = species,
             y = body_mass_g)) +
  geom_point() +
  labs(
    title = "Body mass by species",
    x = "Species",
    y = "Body mass (g)"
  )
```



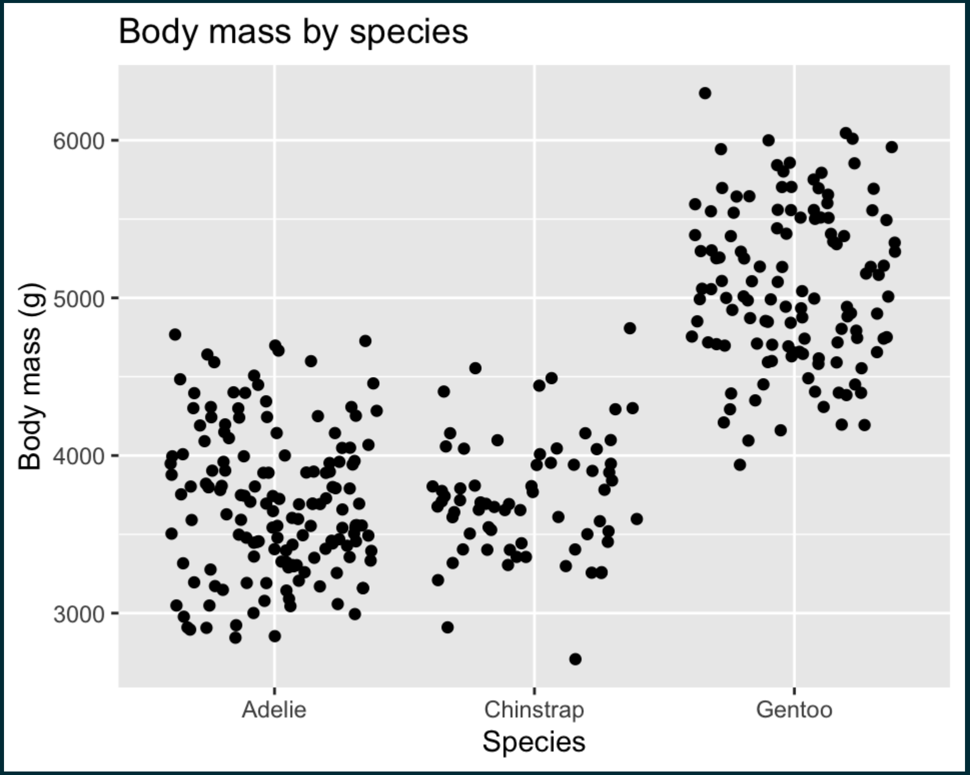
The output is not quite what we would hope for …

EXAMPLE 2A: A CATEGORICAL PREDICTOR

Is species predictive of body mass?

For this type of question, `geom_jitter()` gives a better result.

```r
penguins |>
  ggplot(aes(x = species,
             y = body_mass_g)) +
  geom_jitter() +
  labs(
    title = "Body mass by species",
    x = "Species",
    y = "Body mass (g)"
  )
```



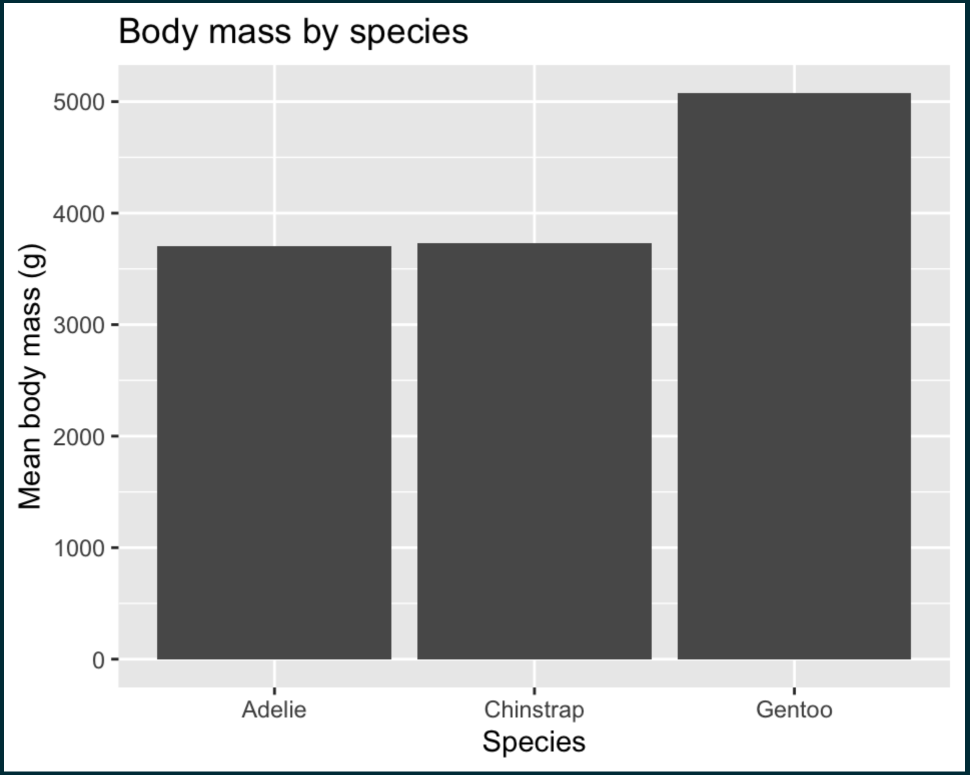
Much better!

EXAMPLE 2B: CAT. PREDICTOR OF MEANS

Is species predictive of mean body mass?

Here, we can use `geom_col()`, but we need to do some summarising first.

```r
penguins |>
  group_by(species) |>
  summarise(mean_body_mass_g =
              mean(body_mass_g, na.rm=TRUE)) |>
  ungroup() |>
  ggplot(aes(x = species,
             y = mean_body_mass_g)) +
  geom_col() +
  labs(
    title = "Body mass by species",
    x = "Species",
    y = "Mean body mass (g)"
  )
```



Don't forget to adjust your aesthetics to reflect the new variable names!

EXAMPLE 3

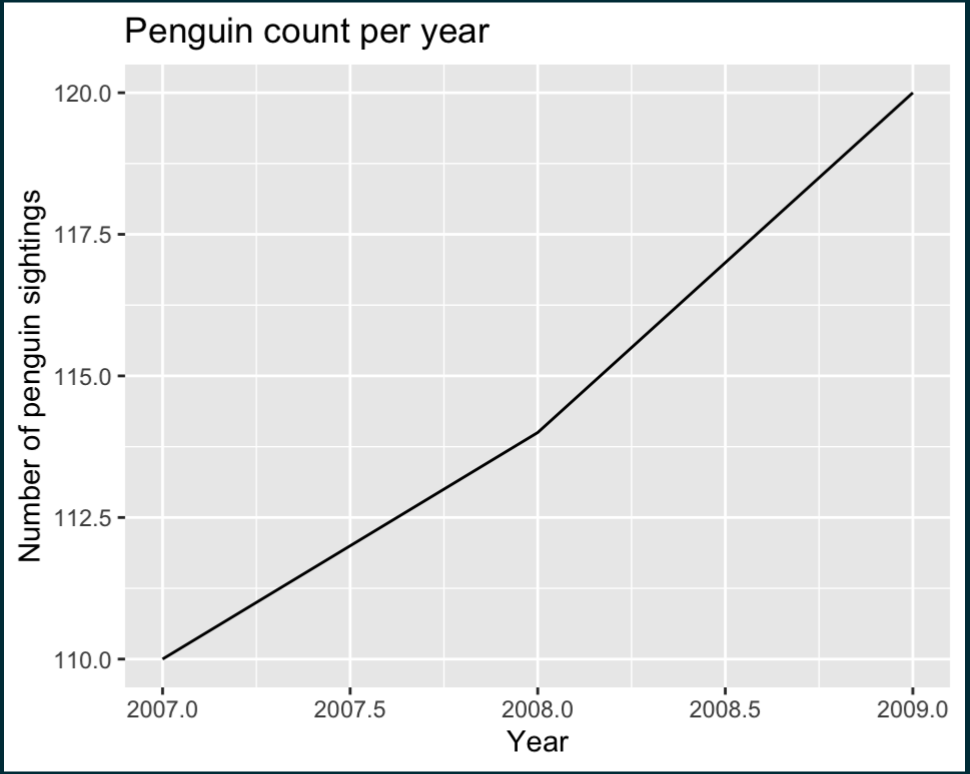
HOW MANY PENGUINS OF EACH SPECIES WERE SIGHTED ACROSS THE DIFFERENT YEARS?

1. Need to plot number of sightings (y-axis) by year (x-axis)
2. First, we need to produce some counts …
3. Finally, `geom_line()` is a good choice for this type of question: two numerical variables but only one y-value for each value of x

EXAMPLE 3A: GEOM_LINE WITHOUT GROUPS

How many penguins (of each species) were sighted across the different years?

```
penguins |>
  count(year) |>
  ggplot(aes(x = year,
             y = n)) +
  geom_line() +
  labs(
    title = "Penguin count per year",
    x = "Year",
    y = "Number of penguin sightings"
  )
```

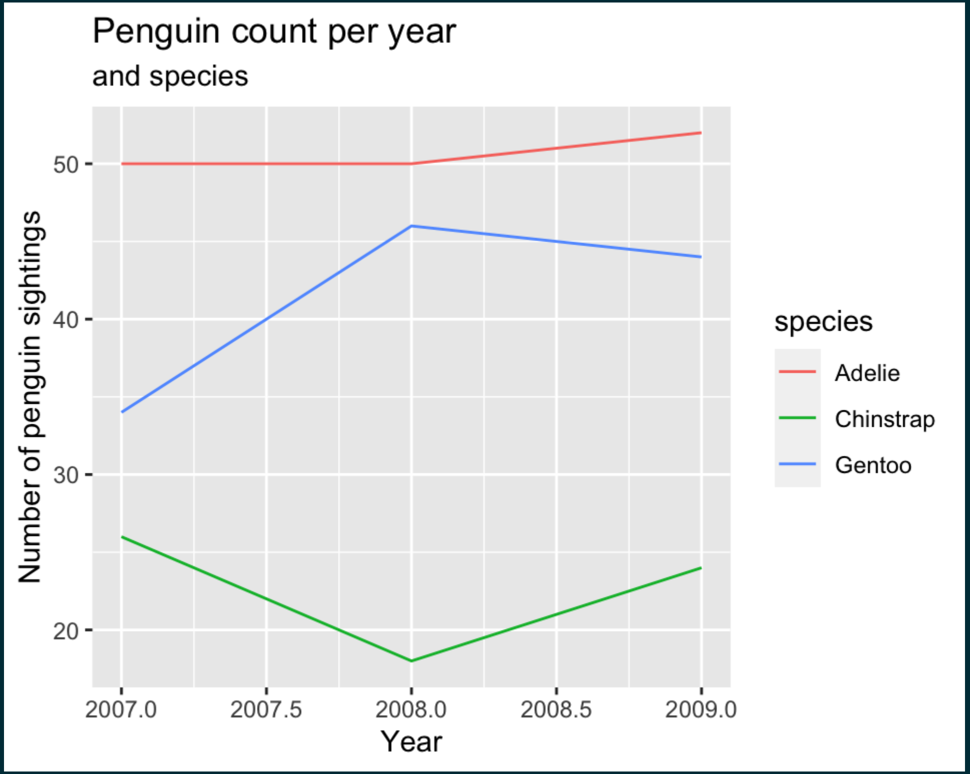


We will see how to fix the axis numbers later.

EXAMPLE 3B: GEOM_LINE WITH GROUPS

How many penguins of each species were sighted across the different years?

```r
penguins |>
  count(year,species) |>
  ggplot(aes(x = year,
             y = n,
             colour = species)) +
  geom_line() +
  labs(
    title = "Penguin count per year",
    subtitle = "and species",
    x = "Year",
    y = "Number of penguin sightings"
  )
```



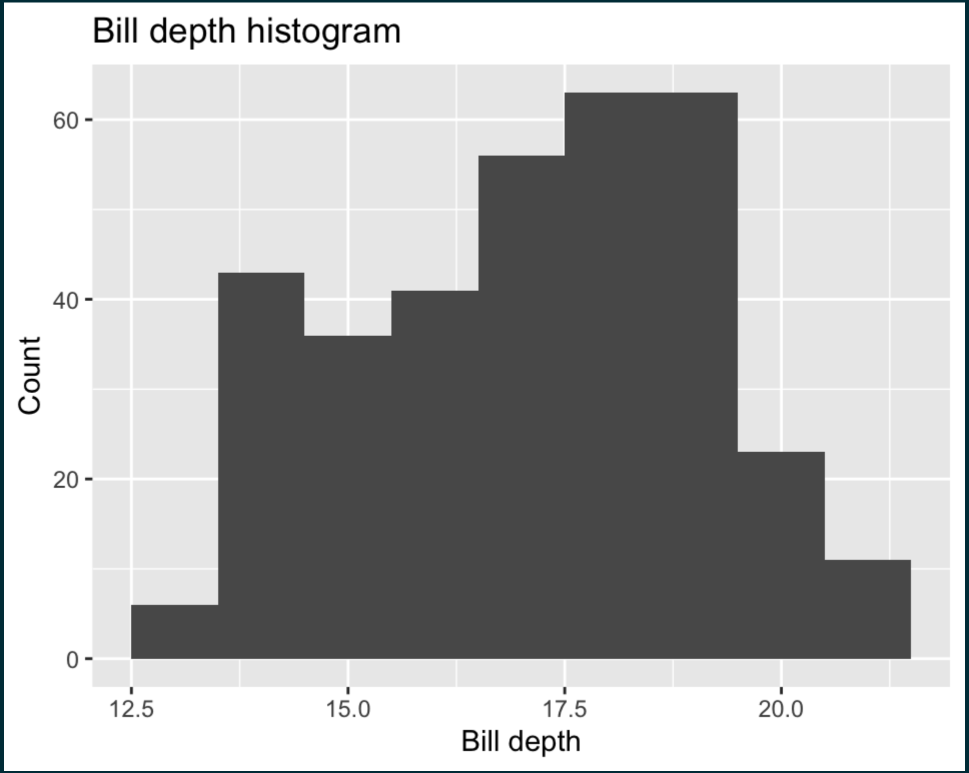
Note the use of colour to introduce groups and also how we can easily add a subtitle.

EXAMPLE 4: HISTOGRAMS

What is the distribution of bill depth?

We can use a histogram to answer this. Note that histograms only require an `x` aesthetic; the y-axis values are calculated automatically through the choice of geom. We also need to adapt the choice of `binwith` to suit the variable under consideration.

```
penguins |>
  ggplot(aes(x = bill_depth_mm)) +
  geom_histogram(binwidth = 1) +
  labs(
    title = "Bill depth histogram",
    x = "Bill depth",
    y = "Count"
  )
```

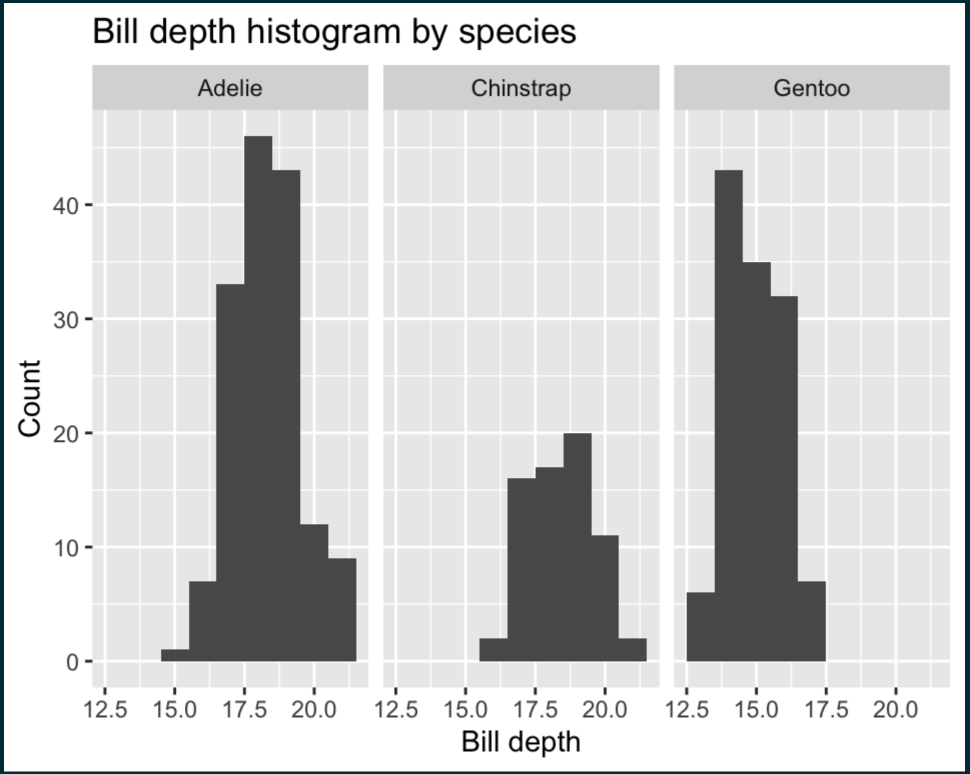


EXAMPLE 4A: HISTOGRAMS BY GROUP

What is the distribution of bill depth by species?

Pair a histogram with facets.

```r
penguins |>
  ggplot(aes(x = bill_depth_mm)) +
  geom_histogram(binwidth = 1) +
  labs(
    title = "Bill depth histogram by species",
    x = "Bill depth",
    y = "Count"
  ) +
  facet_wrap(~species)
```



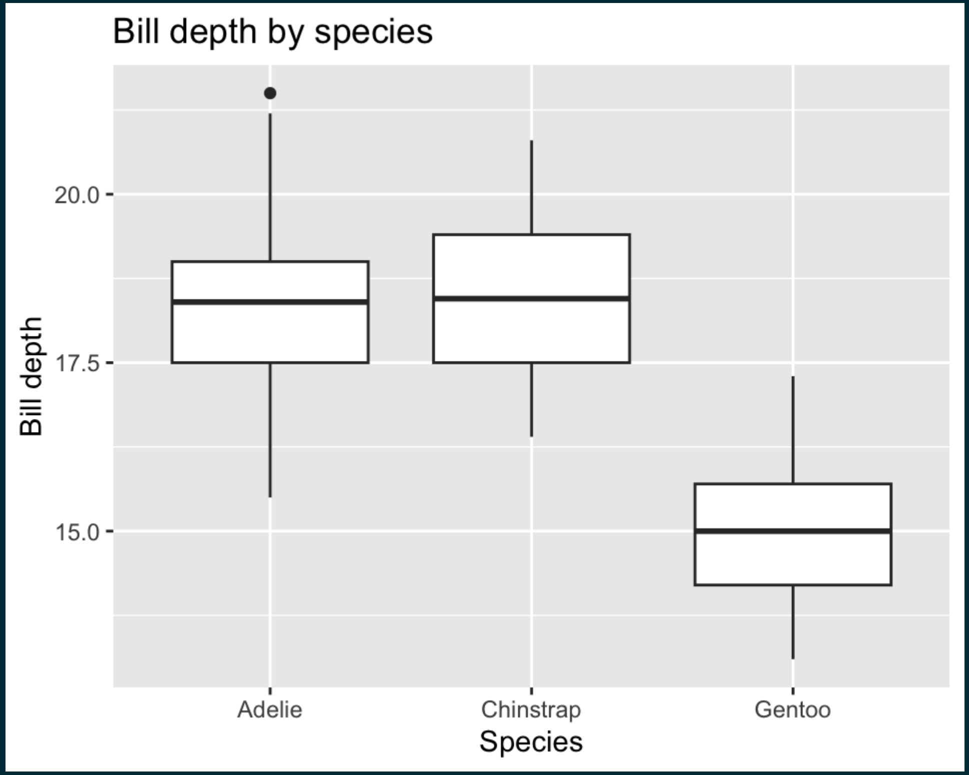
Note the use of `facet_wrap(~variable)` to split the plot into facets according to the variable.

EXAMPLE 5: BOXPLOTS

What is the distribution of bill depth by species?

Alternatively, we can use a boxplot.

```r
penguins |>
  ggplot(aes(x = species,
             y = bill_depth_mm)) +
  geom_boxplot() +
  labs(
    title = "Bill depth by species",
    x = "Species",
    y = "Bill depth"
  )
```

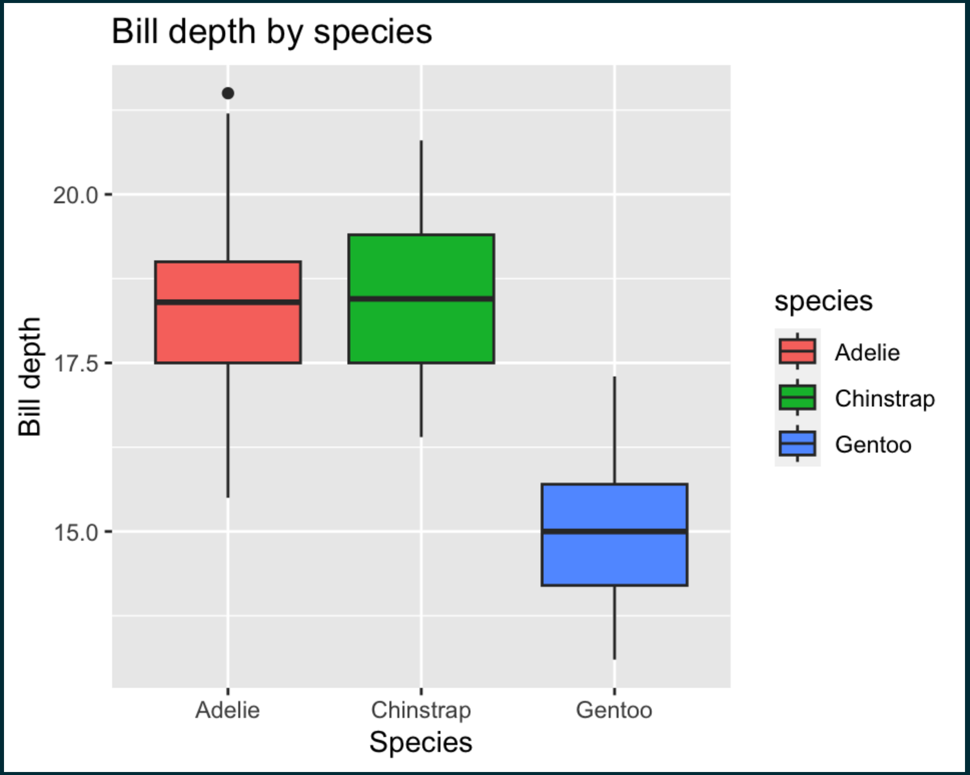


EXAMPLE 5A: BOXPLOTS WITH COLOUR FILL

What is the distribution of bill depth by species?

Note that `colour` is used for points, lines and outlines, while `fill` is used to fill shapes.

```r
penguins |>
  ggplot(aes(x = species,
             y = bill_depth_mm,
             fill = species)) +
  geom_boxplot() +
  labs(
    title = "Bill depth by species",
    x = "Species",
    y = "Bill depth"
  )
```

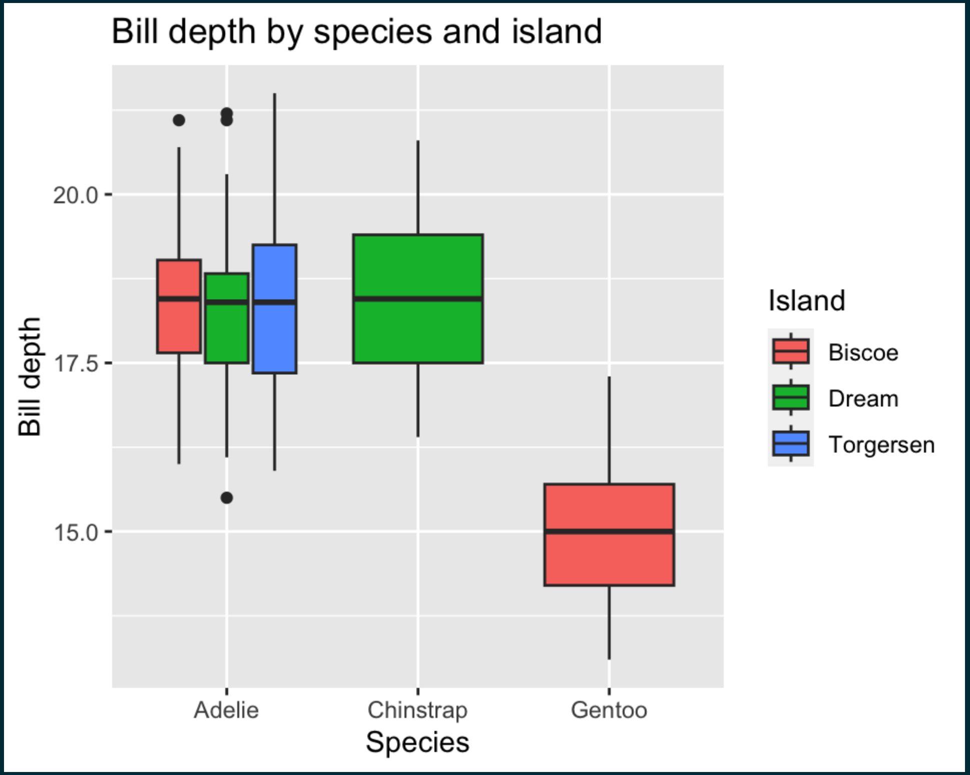


EXAMPLE 5B: BOXPLOTS WITH COLOUR FILL

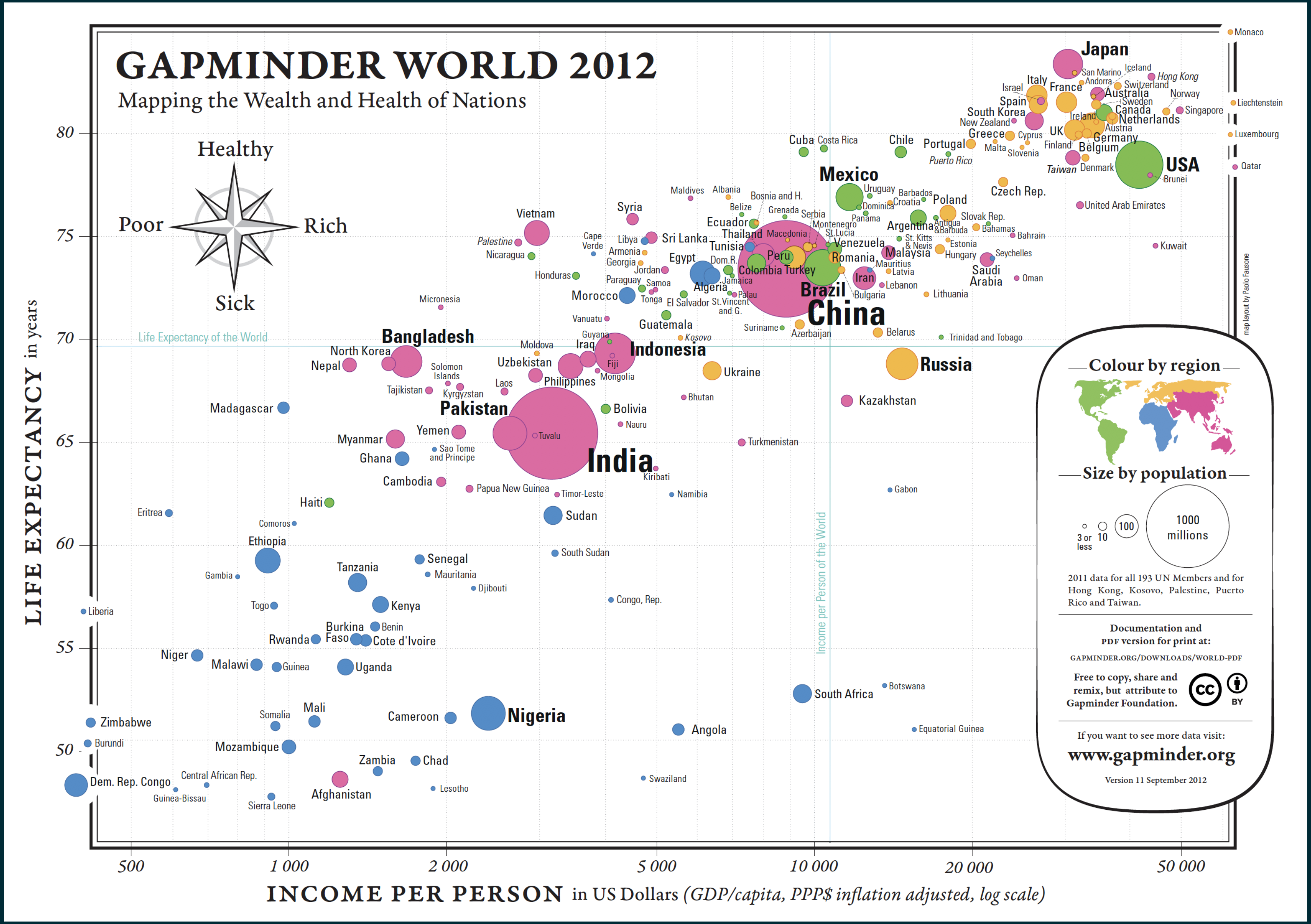
What is the distribution of bill depth by species and island?

You can use `fill` to introduce an additional set of groups. Note also how you can change the label for a particular aesthetic using labs.

```r
penguins |>
  ggplot(aes(x = species,
             y = bill_depth_mm,
             fill = island)) +
  geom_boxplot() +
  labs(
    title = "Bill depth by species and island",
    x = "Species",
    y = "Bill depth",
    fill = "Island"
  )
```



A NEW DATA SET



THE GAPMINDER DATA

- Data from the Gapminder Foundation.
- Their mission is "to fight devastating ignorance with a fact-based worldview that everyone can understand" (from https://www.gapminder.org).
- They provide access to a wide variety of data about the world, including information on income, population and life expectancy across many countries.
- The `gapminder` package in R provides convenient access to an excerpt from the Gapminder data, focusing on life expectancy, GDP per capita and population by country.

- It should already be installed in your cloud workspace, but if typing `gapminder` into the console yields an error, run `install.packages("gapminder")`

THE GAPMINDER DATA

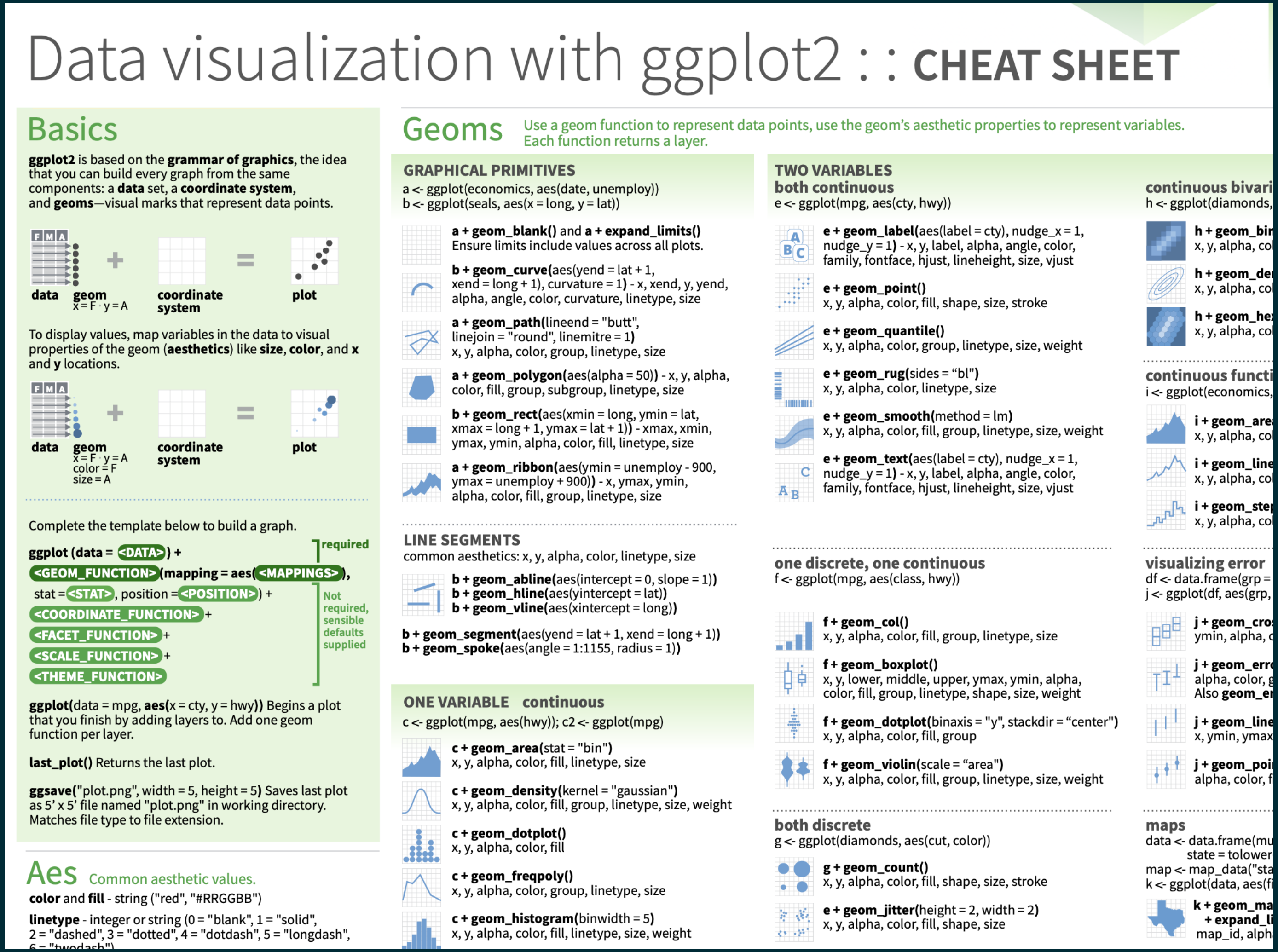
```r
head(gapminder)
```

```
# A tibble: 6 × 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.
```

DO SOME SIMPLE EXPLORATION

- Inspect the data
- Ask some simple questions and create plots to answer them (keeping in mind that you may need to manipulate the data first)
- For example:
  - How has life expectancy in Australia changed over time?
  - What was the relationship between life expectancy and how wealthy a country is (gdpPercap) in 1952? In 2007?
  - How did the mean life expectancy per continent change over time?
  - Which continent had the highest variability in population across countries? Pick two years of your choice to examine. Convert population to millions before plotting.

FURTHER VISUALISATION RESOURCES



Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

data
geom x = F y = A
coordinate system
plot

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA> ) +
  <GEOM_FUNCTION>(mapping = aes( <MAPPINGS> ),
  stat = <STAT> , position = <POSITION> ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and **a + expand_limits()**
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)) - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight

TWO VARIABLES

both continuous
e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl")
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm)
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group

f + geom_violin(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight

both discrete
g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

continuous bivari
h <- ggplot(diamonds,

h + geom_bin
x, y, alpha, co

h + geom_de
x, y, alpha, co

h + geom_he
x, y, alpha, co

continuous functi
i <- ggplot(economics,

i + geom_are
x, y, alpha, co

i + geom_line
x, y, alpha, co

i + geom_ste
x, y, alpha, co

visualizing error
df <- data.frame(grp =
j <- ggplot(df, aes(grp,

j + geom_cro
ymin, alpha, c

j + geom_erro
alpha, color, g
Also **geom_er**

j + geom_line
x, ymin, ymax

j + geom_poi
alpha, color, f

maps
data <- data.frame(mu
state = tolower
map <- map_data("sta
k <- ggplot(data, aes(fi

k + geom_ma
+ expand_li
map_id, alph

The Posit data visualisation cheatsheet includes a lot of useful information, e.g. which geoms to use for which type of question