

ODSC: Data Visualization with ggplot2

Part 2: Creating graphs with ggplot2

by Martin Frigaard & Peter Spangler

Written: February 08 2022

Updated: April 21 2022

Resources

Links:

- [Conference Website](#)
- [Website](#)
- [Part 1](#)
- [Part 2](#)

Materials:

- [RStudio.Cloud](#)
- [Github Repo](#)

Outline

Part 1

Exploratory data analysis

- *What is it, who does it, and why it's important*

A Bayesian mindset

- *Priors → new information → posteriors*

The grammar of graphics

- *Layers, aesthetics, and geoms*

Part 2

Build labels first

- *Set expectations*

Exercises & solutions

- *RStudio.Cloud*

Creating graphs

- *Building graphs layer-by-layer, global vs. local mapping, visual encodings*

Applying the grammar

- *Mapping vs. setting aesthetics, combining layers, facets*

PART 2

Tip: writing code can be frustrating, especially in the beginning...

...it doesn't always produce a tangible result...

...but creating visualizations is rewarding!!!

ggplot2: before we start

Build the labels first!

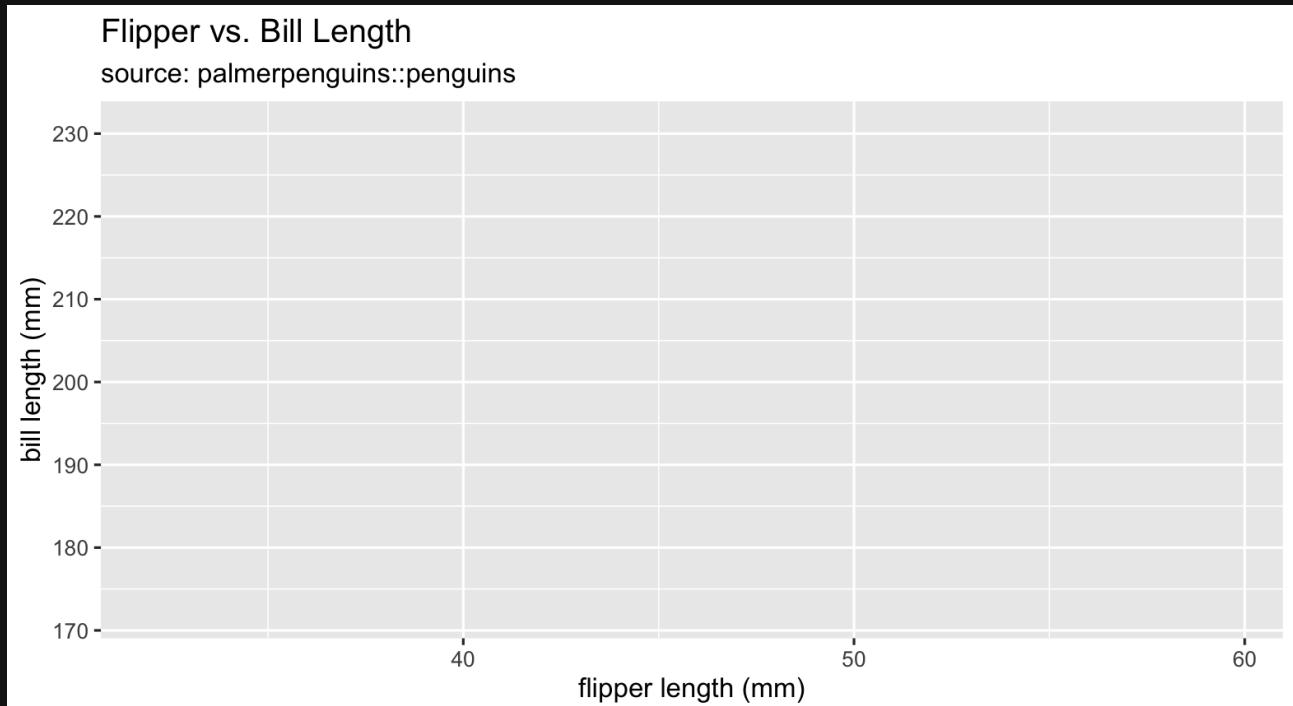
- Create a title, subtitle (with data source), and x/y axis labels

```
labs_penguins <- ggplot2::labs(  
  title = "Flipper vs. Bill Length",  
  subtitle = "source: palmerpenguins::penguins",  
  x = "flipper length (mm)",  
  y = "bill length (mm)")
```

← our expectations

ggplot2: build graph, check labels

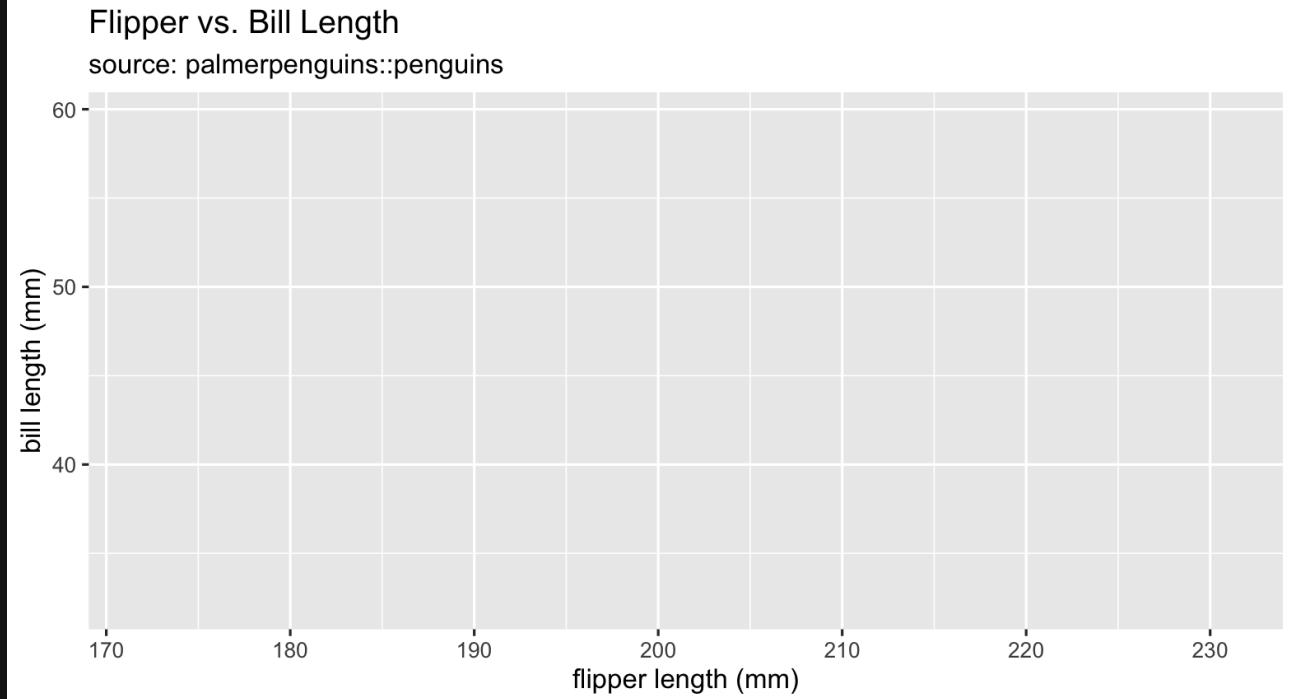
```
labs_penguins <- ggplot2::labs(  
  title = "Flipper vs. Bill Length",  
  subtitle = "source: palmerpenguins::penguins",  
  x = "flipper length (mm)",  
  y = "bill length (mm)")  
  
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      y = flipper_length_mm))  
  ) +  
  labs_penguins
```



What's wrong here?

ggplot2: build graph, check labels, revise

```
labs_penguins <- ggplot2::labs(  
  title = "Flipper vs. Bill Length",  
  subtitle = "source: palmerpenguins::penguins",  
  x = "flipper length (mm)",  
  y = "bill length (mm)")  
  
ggplot(data = penguins,  
       mapping = aes(x = flipper_length_mm,  
                      y = bill_length_mm))  
  ) +  
  labs_penguins
```



FIXED!!!

ggplot2: build graph, check labels, REVISE



Revision Sharpens Thinking: More particularly, rewriting is the key to improved thinking.

It demands a real open-mindedness and objectivity. It demands a willingness to cull verbiage so that ideas stand out clearly. And it demands a willingness to meet logical contradictions head on and trace them to the premises that have created them.

In short, it forces a writer to get up his courage and expose his thinking process to his own intelligence. — Marvin H. Swift, HBR [Clear Writing Means Clear Thinking Means...](#)

Exercises & Solutions

RStudio.Cloud: Set up

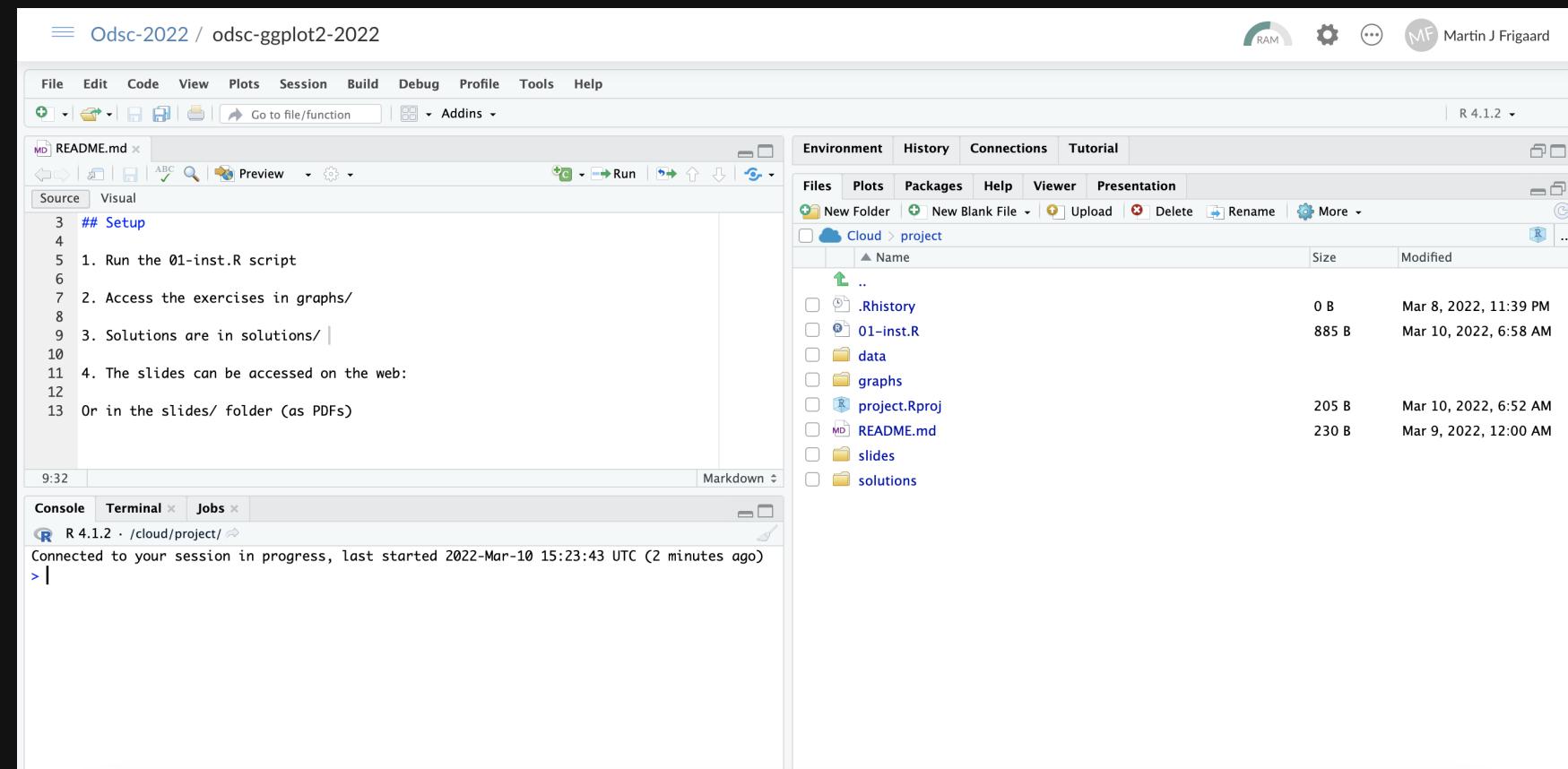
In your RStudio.Cloud, you will see the following:

The screenshot shows the RStudio.Cloud interface. At the top, there's a navigation bar with tabs for 'Projects' (which is selected), 'Members', 'Usage', and 'About'. On the far right of the top bar, there's a user profile for 'Martin J Frigaard' with initials 'MF'. Below the navigation bar, there's a sidebar on the left with three options: 'All Projects' (selected and highlighted in blue), 'Your Projects', and 'Trash'. The main area displays a list titled 'All Projects (1)'. A single project is listed: 'odsc-ggplot2-2022'. This project is highlighted with a red rounded rectangle. Below the project name, there are details: 'Martin J Frigaard' (user icon), 'RStudio Project' (project icon), 'Space members' (space members icon), and 'Created Mar 8, 2022 11:39 PM'. To the right of the project details are several icons: a trash can, a gear, a download arrow, and a three-dot menu. At the bottom of the page, there are global navigation links: 'New Project ▾', 'ACCESS * ▾', 'SORT A Z ▾', and a search bar.

Click on **odsc-ggplot2-2022** project

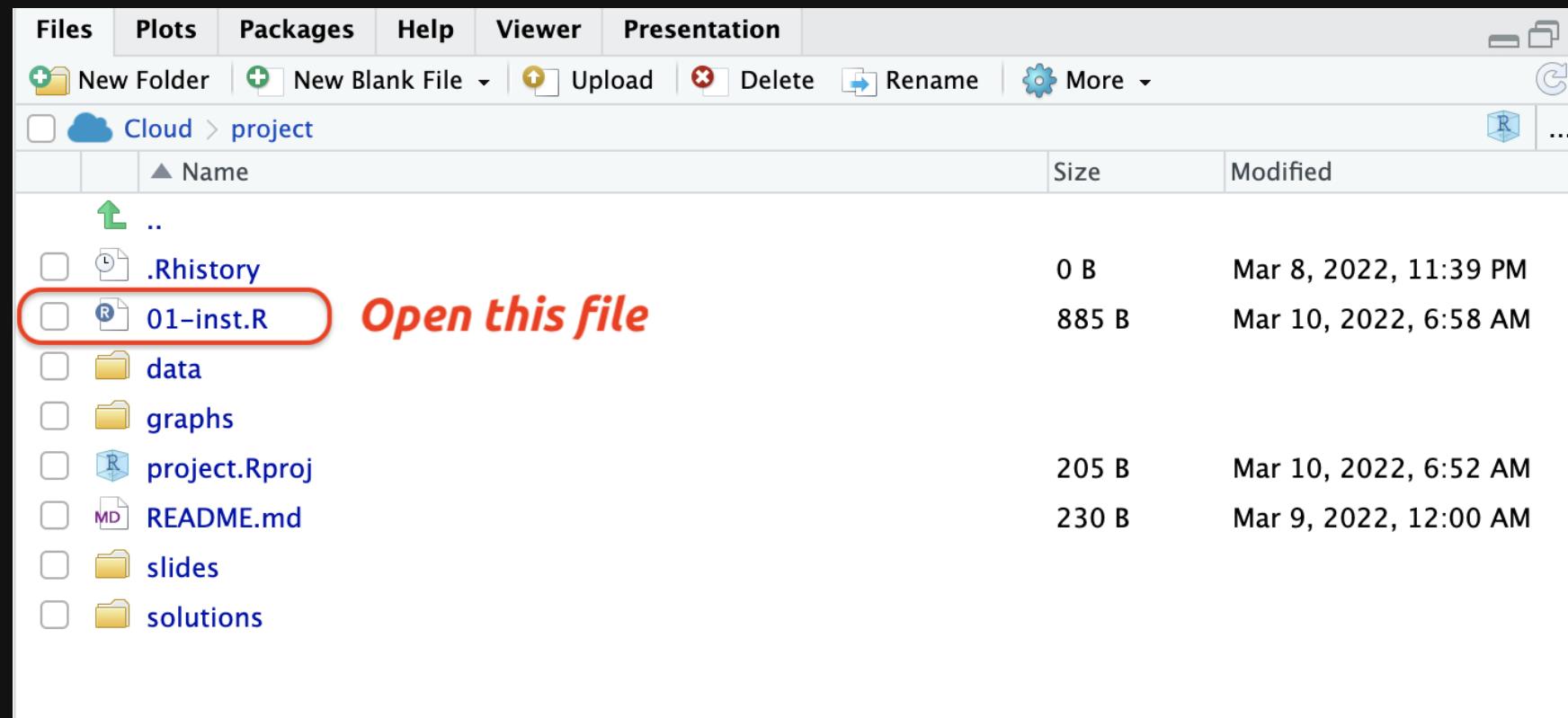
RStudio.Cloud: Set up

In your RStudio IDE, you will see the following:



RStudio.Cloud: Set up

In the **Files** pane, click on the **inst.R** file to open it



RStudio.Cloud: Set up

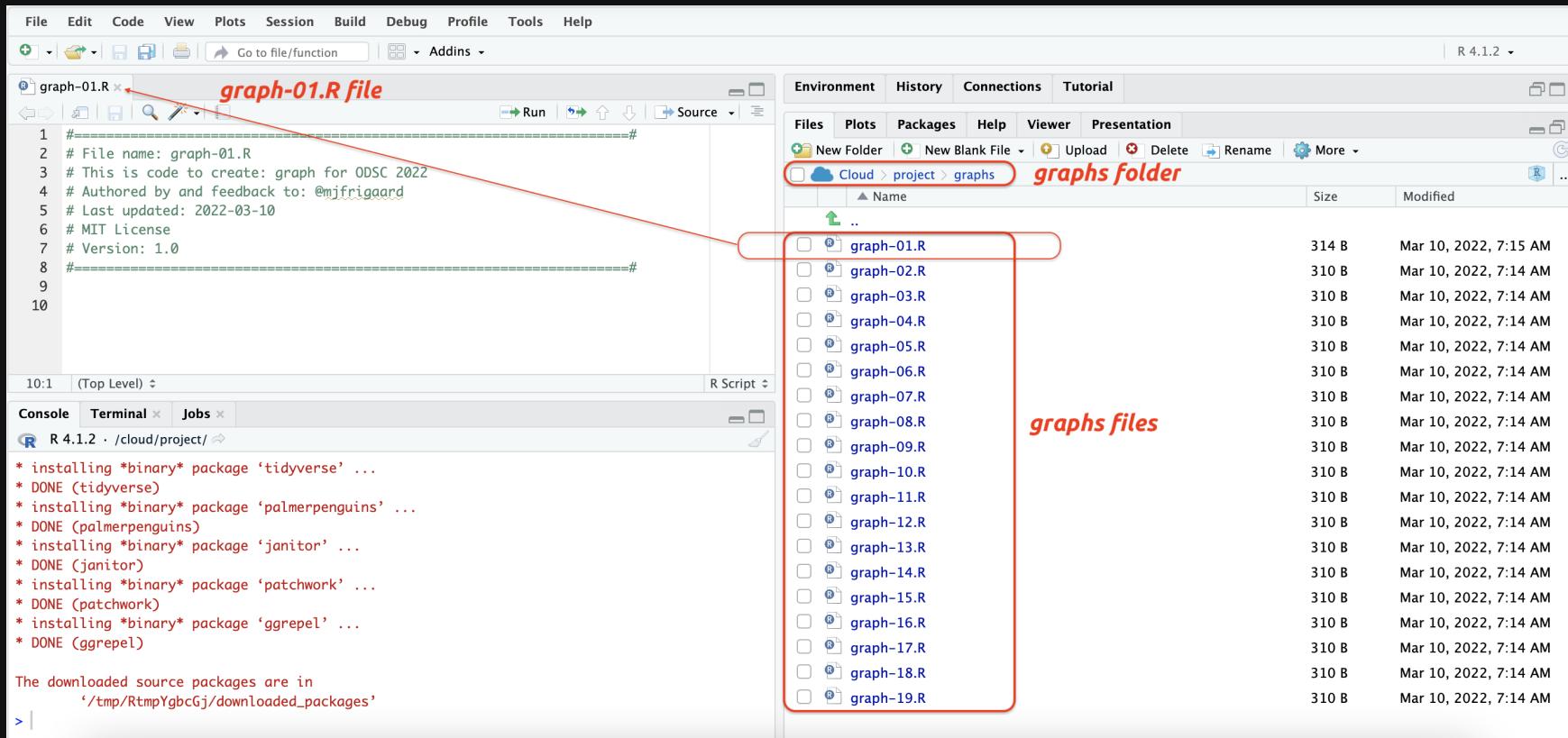
In the **Source** pane, click on the *Source* icon to run **inst.R**.



This sends the code in the **Source** pane to the **Console**

RStudio.Cloud: Exercises

The exercises are in the **graphs/** folder

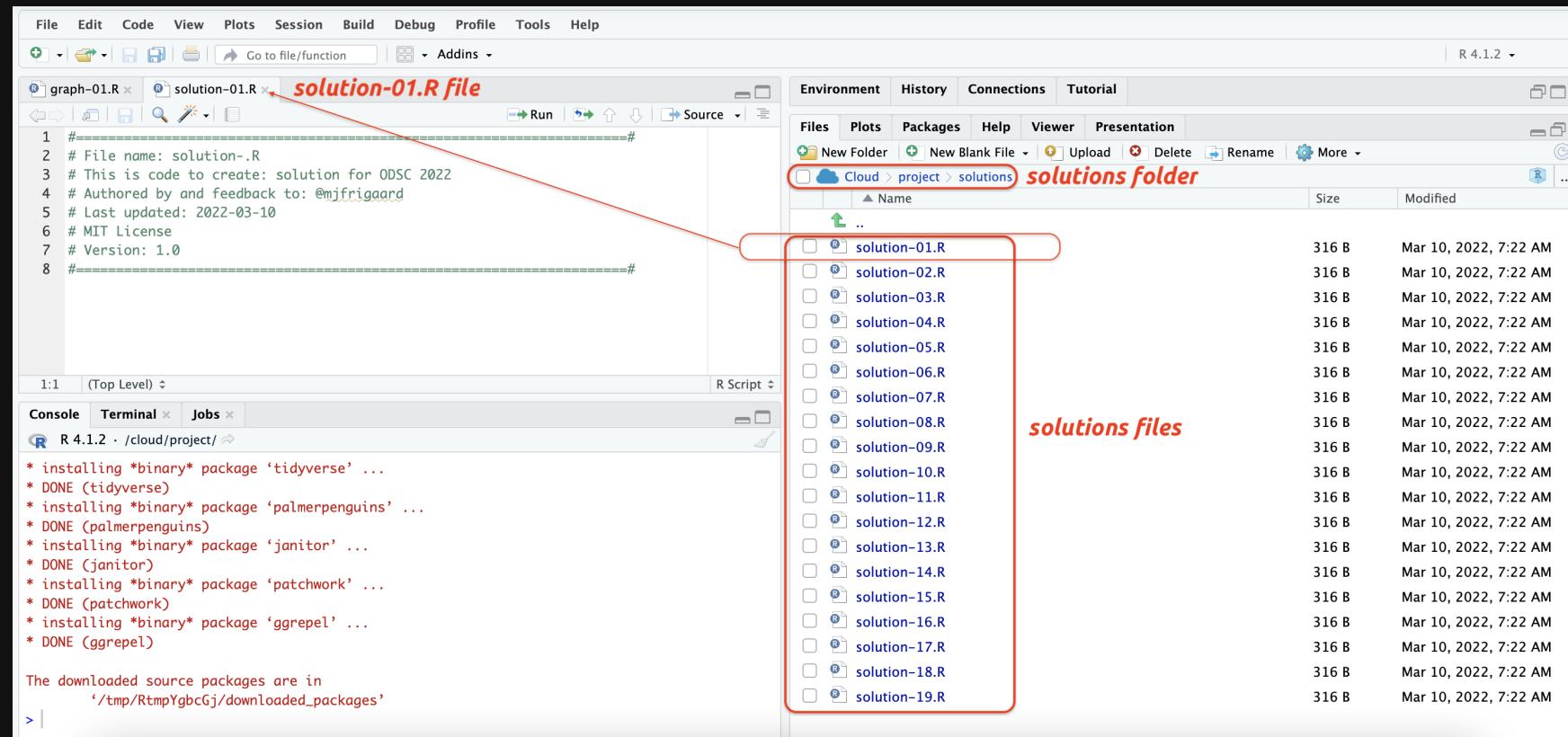


The screenshot shows the RStudio Cloud interface. On the left, the code editor displays a file named `graph-01.R`, which contains R code for creating a graph. A red arrow points from the title bar of this file to the file browser on the right. The file browser shows a list of files in a folder named `graphs`. A red box highlights this folder, and a red arrow points from it to the list of files. The list contains 19 files, each named `graph-n.R` where `n` is a number from 01 to 19. All files have a size of 310 B and were modified on Mar 10, 2022, at 7:14 AM.

Name	Size	Modified
graph-01.R	310 B	Mar 10, 2022, 7:14 AM
graph-02.R	310 B	Mar 10, 2022, 7:14 AM
graph-03.R	310 B	Mar 10, 2022, 7:14 AM
graph-04.R	310 B	Mar 10, 2022, 7:14 AM
graph-05.R	310 B	Mar 10, 2022, 7:14 AM
graph-06.R	310 B	Mar 10, 2022, 7:14 AM
graph-07.R	310 B	Mar 10, 2022, 7:14 AM
graph-08.R	310 B	Mar 10, 2022, 7:14 AM
graph-09.R	310 B	Mar 10, 2022, 7:14 AM
graph-10.R	310 B	Mar 10, 2022, 7:14 AM
graph-11.R	310 B	Mar 10, 2022, 7:14 AM
graph-12.R	310 B	Mar 10, 2022, 7:14 AM
graph-13.R	310 B	Mar 10, 2022, 7:14 AM
graph-14.R	310 B	Mar 10, 2022, 7:14 AM
graph-15.R	310 B	Mar 10, 2022, 7:14 AM
graph-16.R	310 B	Mar 10, 2022, 7:14 AM
graph-17.R	310 B	Mar 10, 2022, 7:14 AM
graph-18.R	310 B	Mar 10, 2022, 7:14 AM
graph-19.R	310 B	Mar 10, 2022, 7:14 AM

RStudio.Cloud: Solutions

Each exercise has a solution file in **solutions/** folder



The data

We're going to use the `palmerpenguins::penguins`

- Below are three options for viewing a dataset in RStudio:

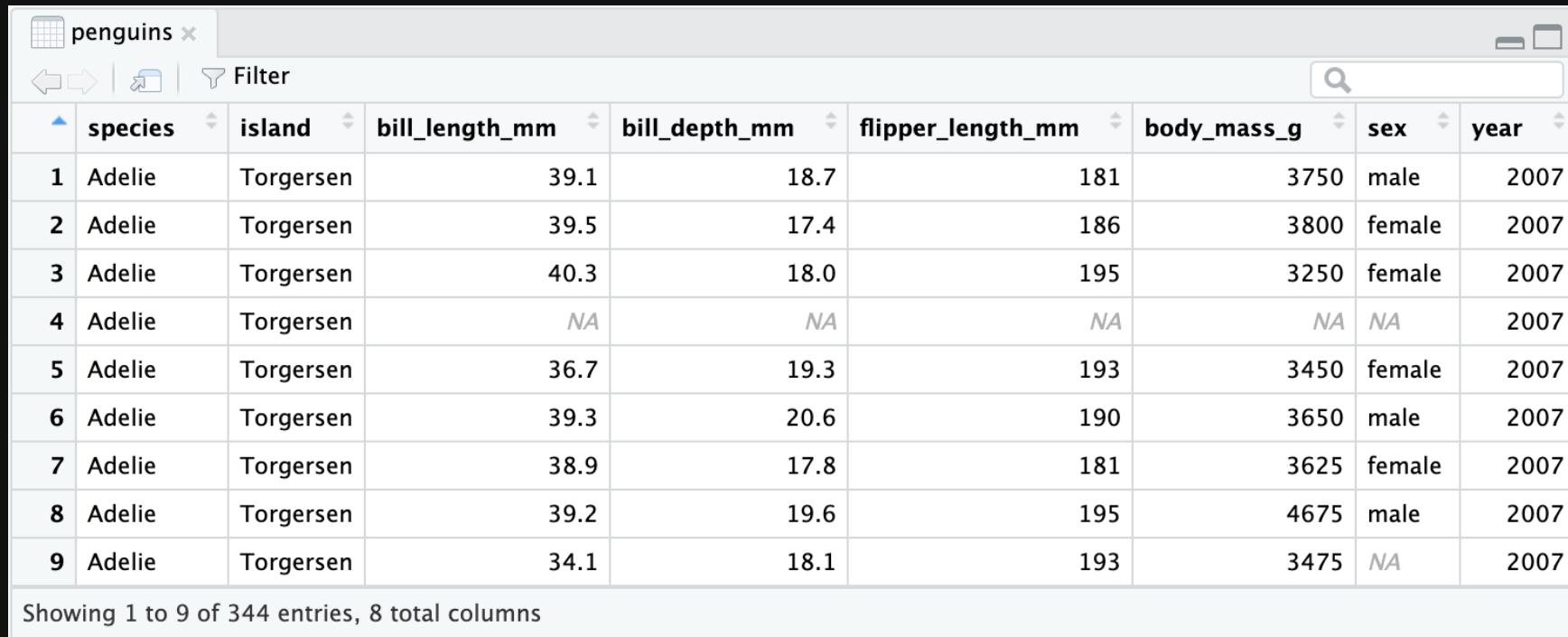
`View(penguins)`

`glimpse(penguins)`

`str(penguins)`

Viewing data

`View()` opens the RStudio data viewer



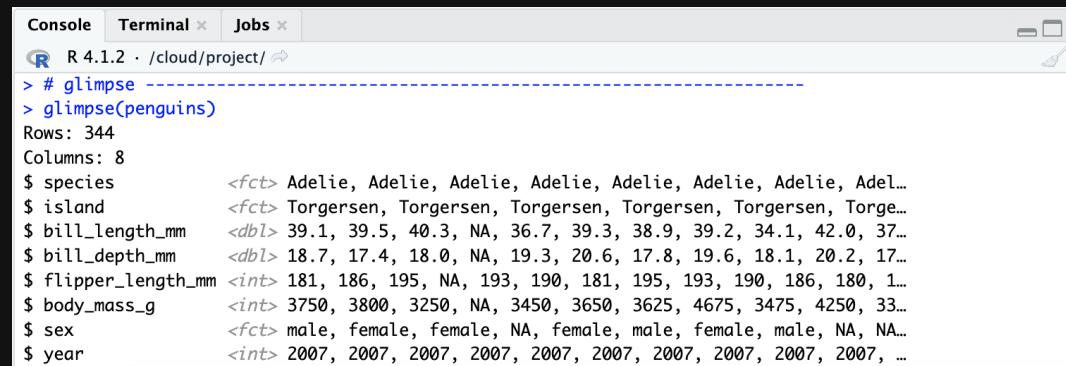
The screenshot shows the RStudio data viewer interface for the "penguins" dataset. The window title is "penguins". The viewer has a toolbar with icons for back/forward, refresh, and filter, along with a search bar. The main area displays a data frame with 9 rows and 8 columns. The columns are: species, island, bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g, sex, and year. The data consists of Adelie penguins from Torgersen island. Row 4 contains NA values for bill_length_mm, bill_depth_mm, and flipper_length_mm. The footer of the viewer indicates "Showing 1 to 9 of 344 entries, 8 total columns".

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
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	18.0	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

Showing 1 to 9 of 344 entries, 8 total columns

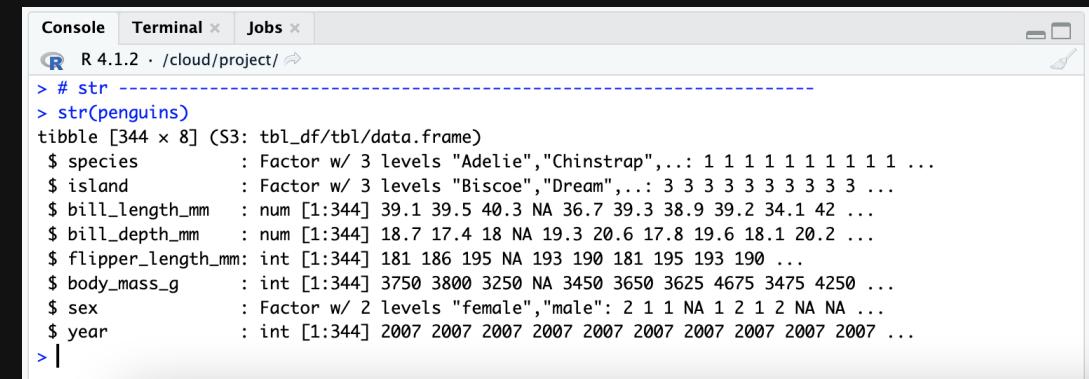
Viewing data

glimpse() and str() are displayed in the console



```
Console Terminal Jobs
R 4.1.2 · /cloud/project/ 

> # glimpse -----
> glimpse(penguins)
Rows: 344
Columns: 8
$ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel...
$ island       <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torg...
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.0, 37...
$ bill_depth_mm  <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, 20.2, 17...
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186, 180, 1...
$ body_mass_g    <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, 4250, 33...
$ sex           <fct> male, female, female, NA, female, male, female, male, NA, NA...
$ year          <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, ...
```



```
Console Terminal Jobs
R 4.1.2 · /cloud/project/ 

> # str -----
> str(penguins)
tibble [344 x 8] (S3:tbl_df/tbl/data.frame)
$ species      : Factor w/ 3 levels "Adelie", "Chinstrap", ...: 1 1 1 1 1 1 1 1 ...
$ island       : Factor w/ 3 levels "Biscoe", "Dream", ...: 3 3 3 3 3 3 3 3 ...
$ bill_length_mm : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
$ bill_depth_mm  : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
$ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
$ body_mass_g    : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
$ sex           : Factor w/ 2 levels "female", "male": 2 1 1 NA 1 2 1 2 NA NA ...
$ year          : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

Build from scratch, layer-by-layer

graph 01 Step 0 = LABELS!

We want to build the labels **first**:

- title = "Bill and flipper length of Palmer penguins"
- subtitle = "Size measurements for adult foraging penguins"
- x = "Bill length (mm)"
- y = "Flipper length (mm)"

```
# build labels
labs_bill_vs_flipper <- ggplot2::labs(
  title = "Bill and flipper length of Palmer penguins",
  subtitle = "Size measurements for adult foraging penguins",
  x = "Bill length (mm)",
  y = "Flipper length (mm)")
```

graph 01 Step 1: Initialize plot with data

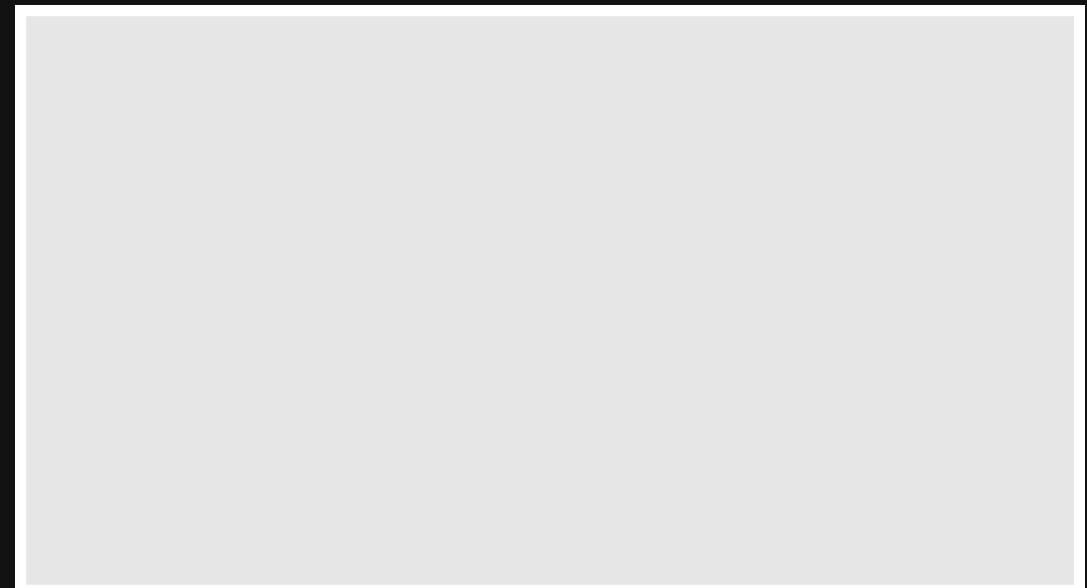
The `ggplot2::ggplot()` function initiates the plot

```
ggplot(data = )
```

Place `penguins` in the data argument

```
ggplot(data = penguins)
```

This gives us a blank canvas!



graph 02 Step 2: Map variables to positions

We have our data and labels--we just need to add our variables

Map `bill_length_mm` to `x`

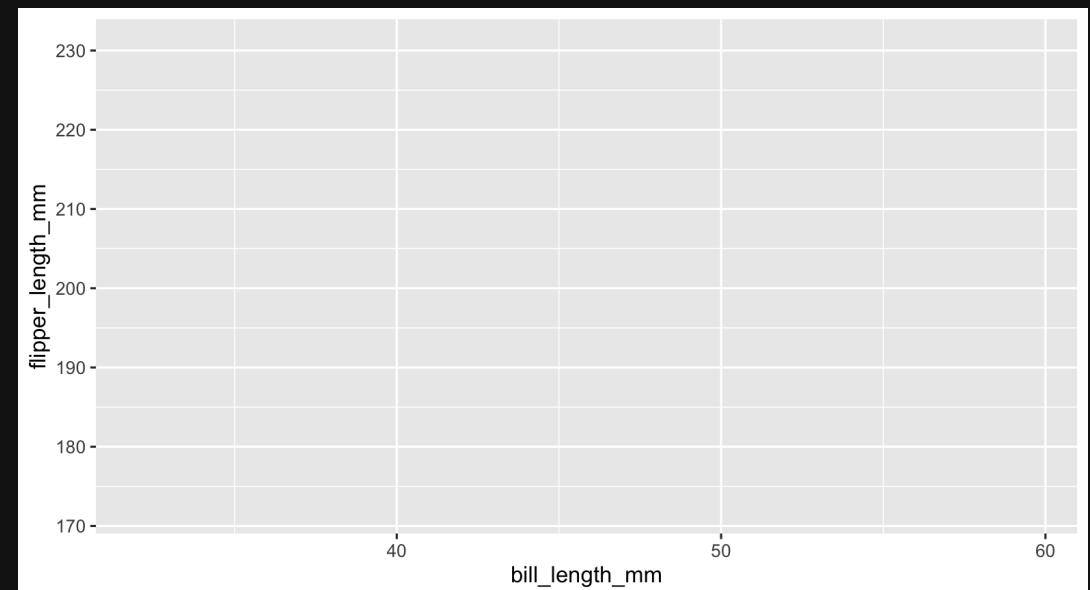
```
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      ))
```

Map `flipper_length_mm` to `y`

```
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      y = flipper_length_mm))
```

Now we have our variables on our graph!

Now our canvas has `x` and `y` axes

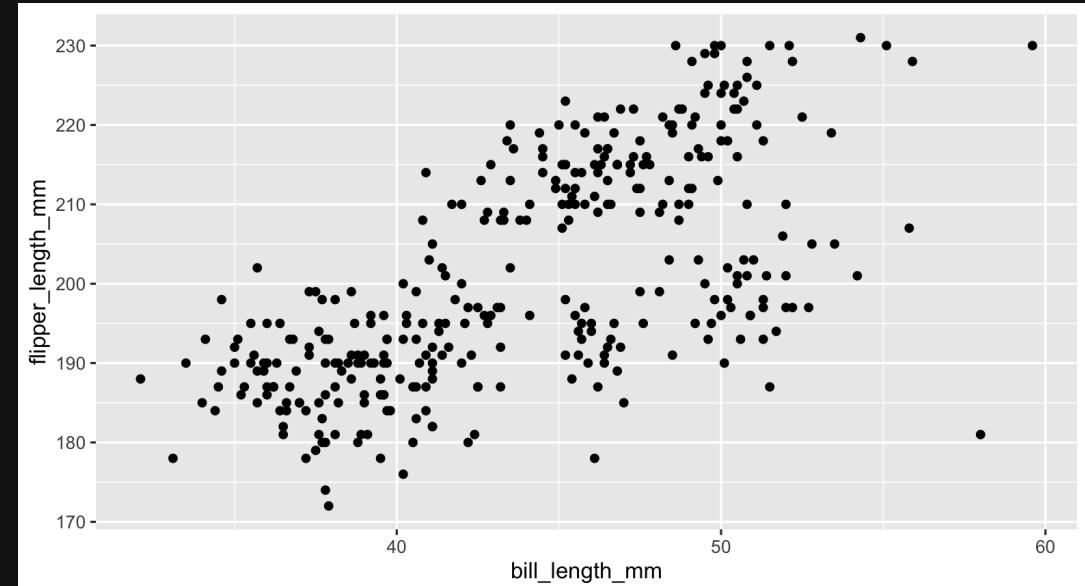


graph 03 Step 3: Adding geoms

Add the `geom_point()` function with the `+` symbol

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

The `geom_point()` function tells R we want to see the points (or dots) on our canvas

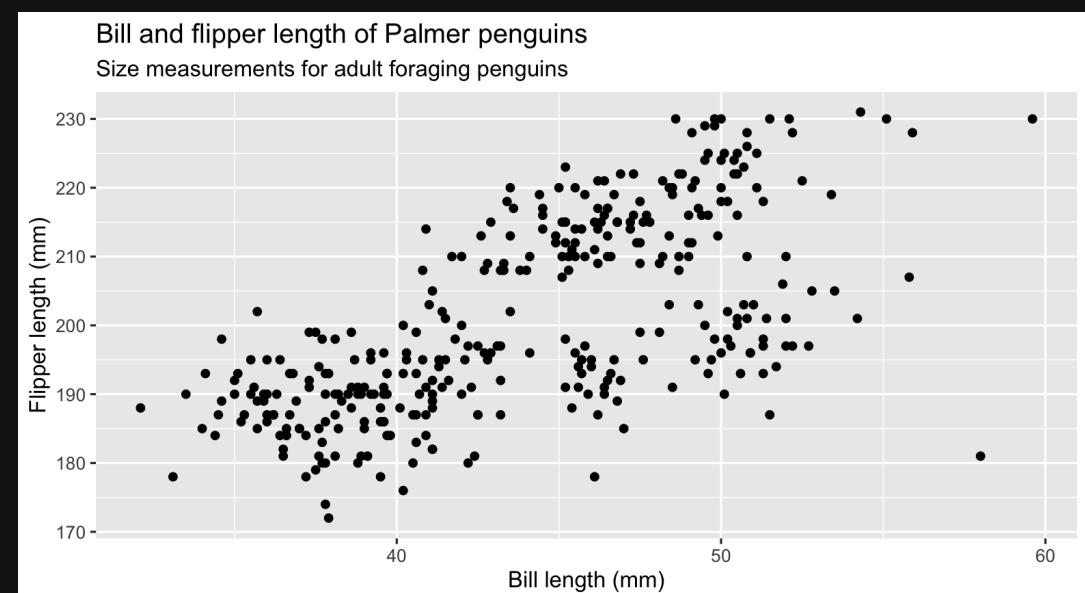


graph 04 Step 4: Don't forget the labels!

Finally, we want to add the labels we created
(`labs_bill_vs_flipper`)

```
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      y = flipper_length_mm)) +  
  geom_point() +  
  labs_bill_vs_flipper
```

And we have our first graph!



Global vs. local mapping

Global vs. local mapping

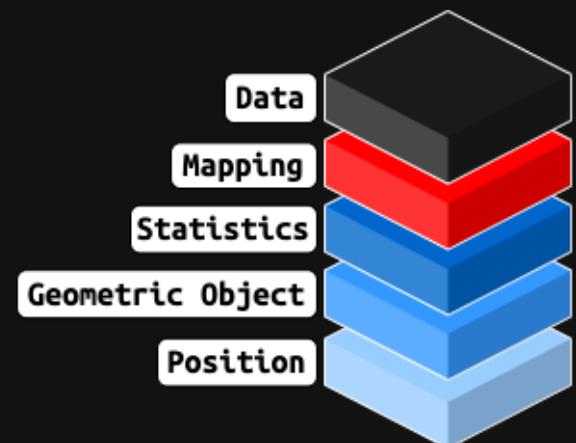
We've just created a graph by setting **global** aesthetics

Global means aesthetic mappings are set when the graph is initialized with the **ggplot()** function

```
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      y = flipper_length_mm)) +  
  geom_point() +  
  labs_bill_vs_flipper
```

If we map aesthetics **ggplot()**, all the following **geom_***() layers will inherit these aesthetics.

Recall what goes into each layer from Part 1

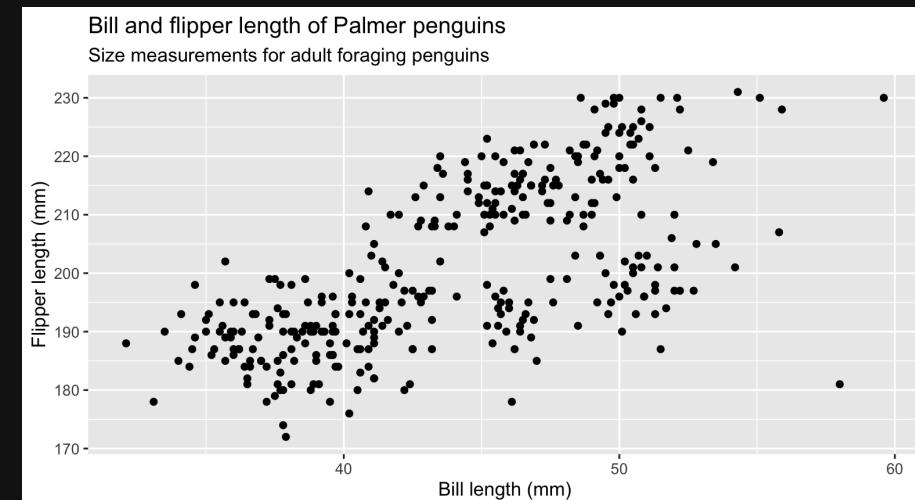
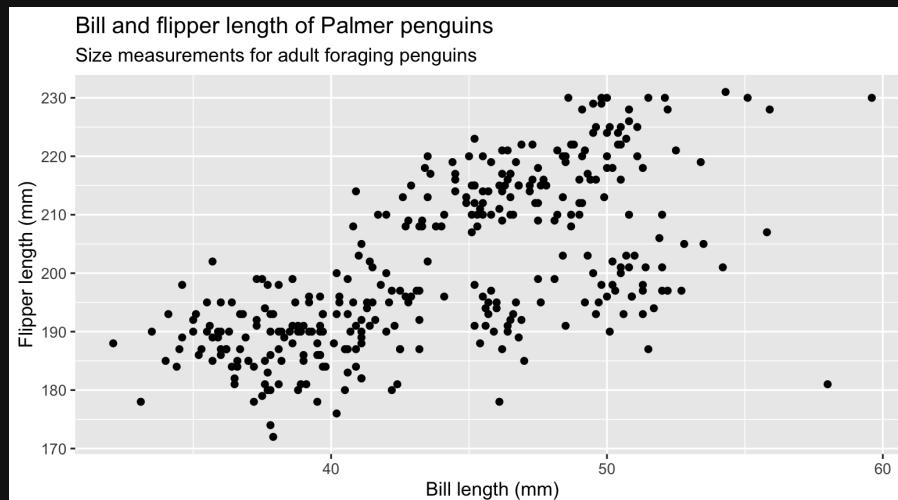


Global vs. local mapping

Mapping aesthetics **globally** and then adding the `geom_*()` function results in the same graph as when we map aesthetics **locally** (*inside the `geom_*()` function*)

```
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      y = flipper_length_mm)) +  
  geom_point() +  
  labs_bill_vs_flippper
```

```
ggplot(data = penguins) +  
  geom_point(  
    mapping = aes(x = bill_length_mm,  
                   y = flipper_length_mm)) +  
  labs_bill_vs_flippper
```



Our ggplot2 templates

The template from part 1 uses **local** mappings (i.e. aesthetic mappings are set *inside* the **geom_*** function).

```
# Recall our template from Part 1
ggplot(data = <DATA>) +
  geom_*(mapping = aes(<AESTHETIC MAPPINGS>))
```

We could adjust this template to include **global** mappings (and the option to include aesthetic mappings **locally**)

```
# Adjusted template
ggplot(data = <DATA>,
       mapping = aes(<AESTHETIC MAPPINGS>)) + # global mappings
       geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) # local mappings
```

Read more [here](#).

graph 05 Convert global to local mappings

For `graph 05.R`, convert the global aesthetics to local aesthetics inside the `geom_point()` function

Global

```
ggplot(data = penguins,  
       mapping = aes(x = bill_length_mm,  
                      y = flipper_length_mm)) +  
  geom_point() +  
  labs_bill_vs_flipper
```

Local

```
ggplot(data = penguins) +  
  geom_point(  
    mapping = aes(x = bill_length_mm,  
                  y = flipper_length_mm)) +  
  labs_bill_vs_flipper
```

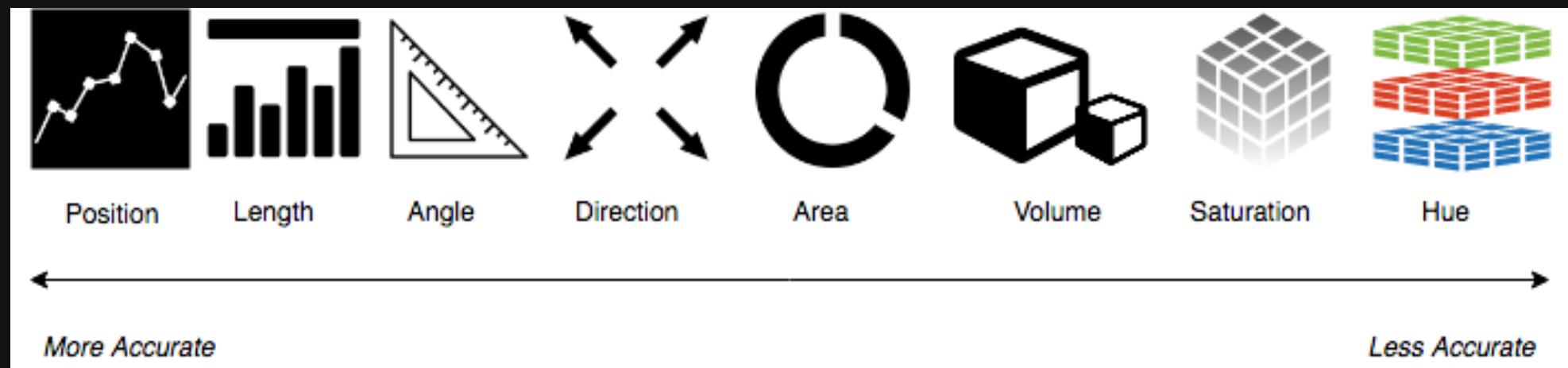
Visual encodings

What are visual encodings?

Visual encodings are what we see on the graph

Things like position, size, shape, color, etc.

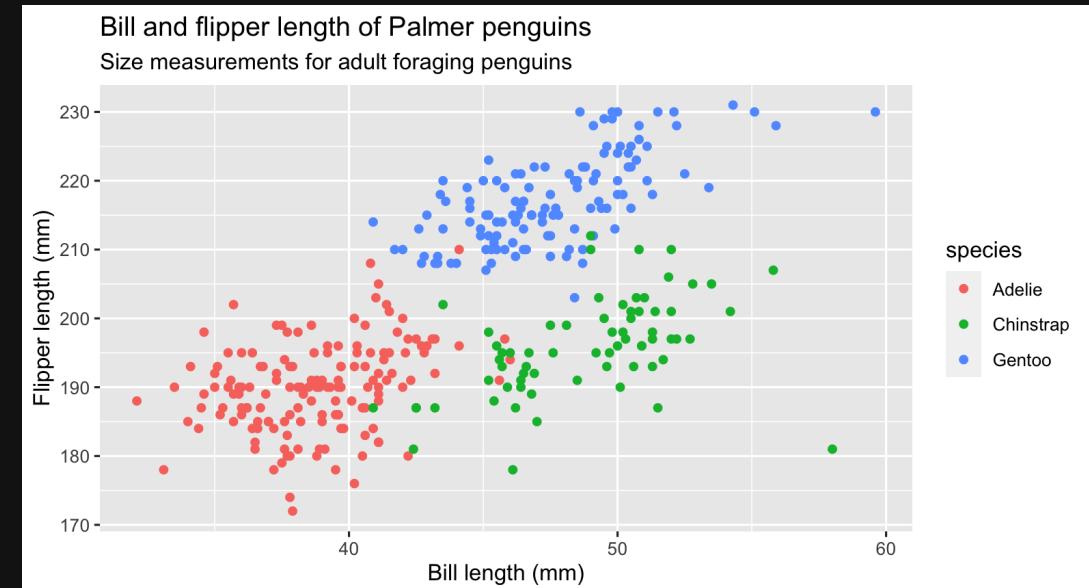
Ranked by accuracy ("generally speaking"):



graph 06 Adding color (global)

Map `color` to the `species` variable in the scatter plot using `global` aesthetic mapping

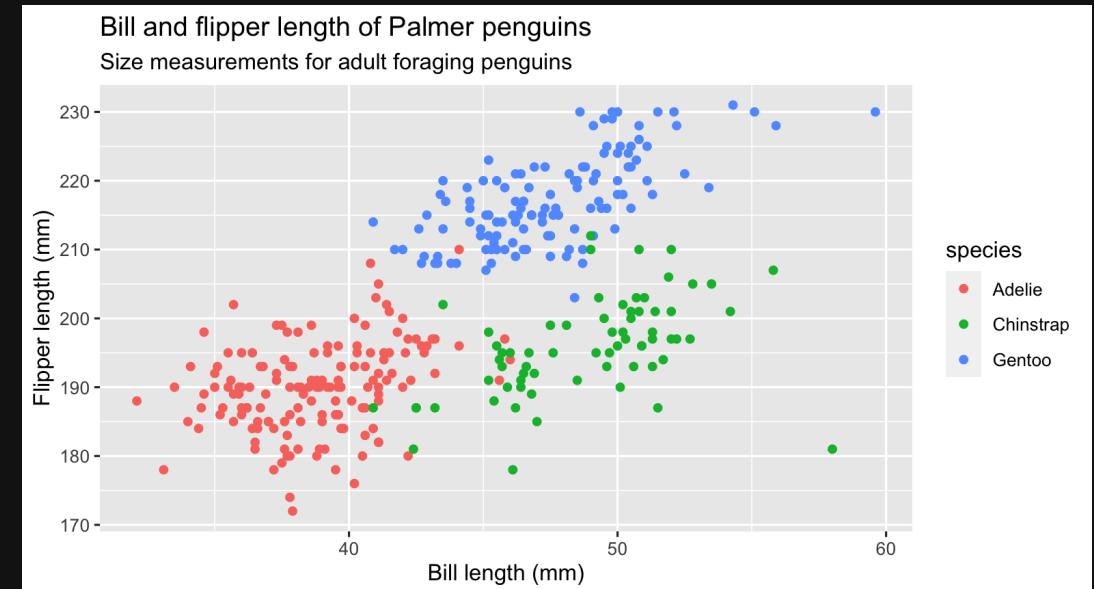
```
ggplot(data = penguins,  
       mapping =  
         aes(x = bill_length_mm,  
              y = flipper_length_mm,  
              color = species)) +  
  geom_point() +  
  labs_bill_vs_flipper
```



graph 07 Adding color (local)

Map `color` to the `species` variable in the scatter plot using `local` aesthetic mapping

```
ggplot(data = penguins,  
       mapping =  
         aes(x = bill_length_mm,  
             y = flipper_length_mm)) +  
  geom_point(aes(color = species)) +  
  labs_bill_vs_flipper
```



The `x` and `y` aesthetics are inherited from the `ggplot()` function, but the `color` aesthetic comes from the `geom_point()` layer

graph 08 Color vs. Fill

Below we'll look at the counts of `sex` vs. `species` of Palmer penguins

Create labels

```
labs_sex_vs_species <- ggplot2::labs(
  title = "Sex by species of Palmer penguins",
  subtitle = "Counts for adult foraging penguins",
  x = "Sex",
  fill = "Species")
```

Create `penguins_no_miss` by removing missing values

```
penguins_no_miss <- drop_na(data = penguins)
```

`penguins_no_miss`

species	island	bill_length_mm
<fct>	<fct>	<dbl>
Adelie	Torgersen	39.1
Adelie	Torgersen	39.5
Adelie	Torgersen	40.3
Adelie	Torgersen	36.7
Adelie	Torgersen	39.3
Adelie	Torgersen	38.9
Adelie	Torgersen	39.2
Adelie	Torgersen	41.1
Adelie	Torgersen	38.6
Adelie	Torgersen	34.6

1-10 of 333 row... [Previous](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) ... [34](#) [Next](#)

graph 08 Color vs. Fill

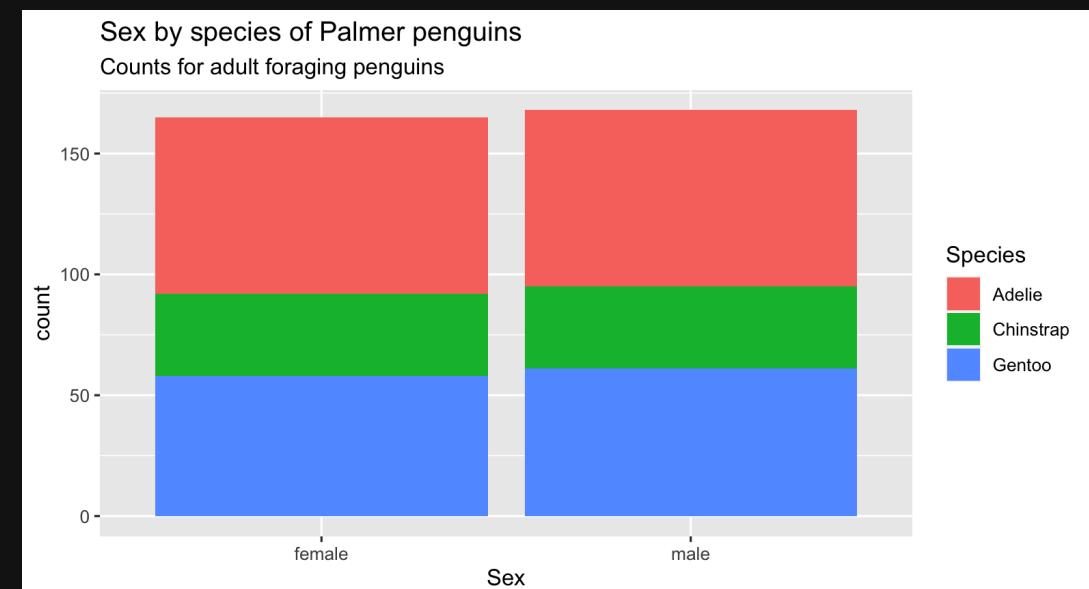
Some `geom_*` functions take the `fill` argument instead of `color`

Build a bar-graph using `geom_bar()` using `local` aesthetic mapping

Map `sex` to the `x` axis and `y` to `fill`

```
ggplot(data = penguins_no_miss) +
  geom_bar(mapping = aes(x = sex,
                        fill = species)) +
  labs_sex_vs_species
```

Don't forget the labels!

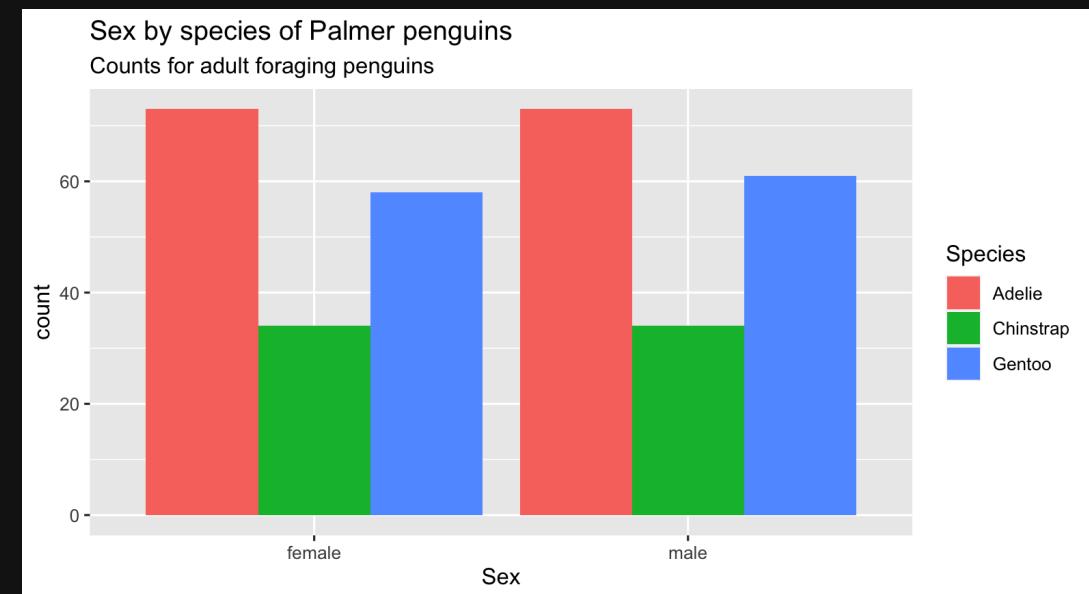


graph 09 Bar position

Stacked bar-graphs make it difficult to do side-by-side comparisons using the **y** axis

Using the same code as [graph 08](#), add the **position = "dodge"** argument *outside* the **aes()** function

```
ggplot(data = penguins_no_miss) +  
  geom_bar(mapping = aes(x = sex,  
                         fill = species),  
           position = "dodge") +  
  labs_sex_vs_species
```



graph 10 Histograms (special bar-graphs)

The `geom_histogram()` function uses 'bins' to represent counts for each value

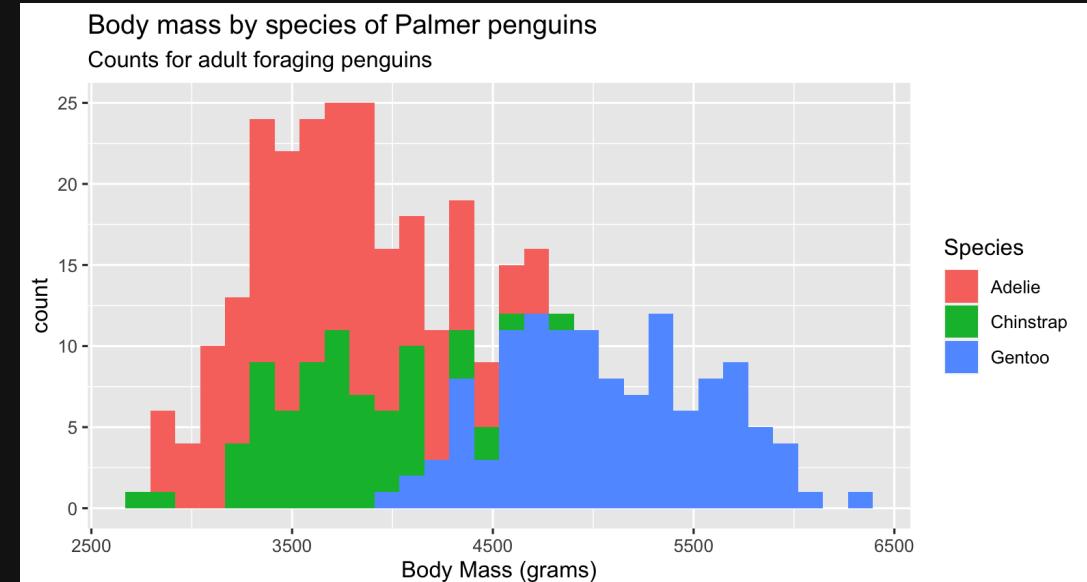
Create new labels

```
labs_bodymass_vs_species <- ggplot2::labs(
  title = "Body mass by species of Palmer penguins",
  subtitle = "Counts for adult foraging penguins",
  x = "Body Mass (grams)",
  fill = "Species")
```

Create a histogram of `body_mass_g`, colored (filled) by `species`

```
ggplot(data = penguins) +
  geom_histogram(
    mapping = aes(x = body_mass_g,
                  fill = species)) +
  labs_bodymass_vs_species
```

Notice the overlapping distributions of `body_mass_g`



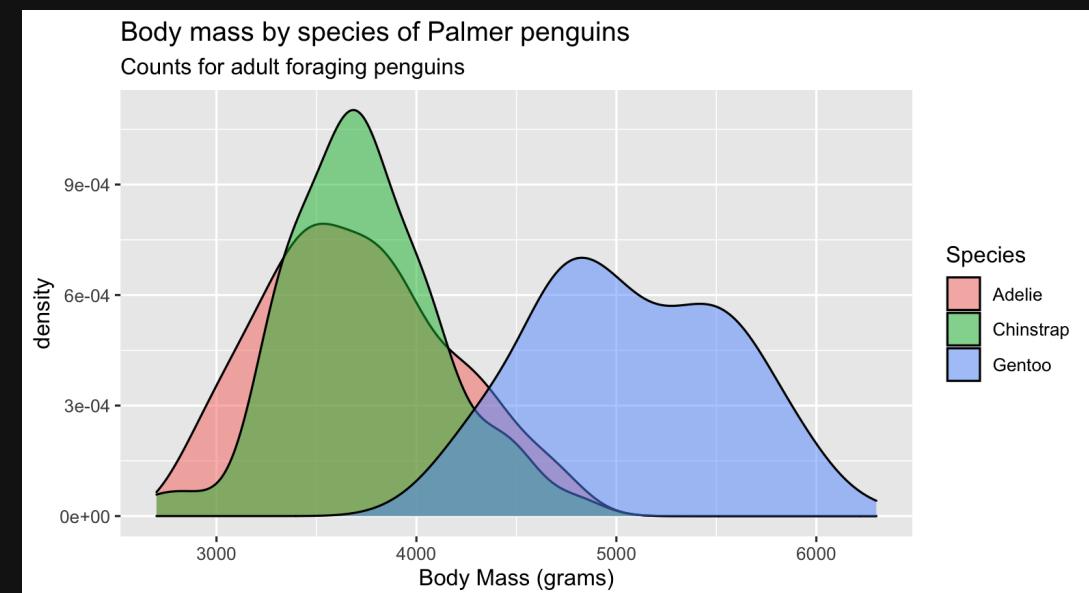
graph 11 Density plots

Density plots are also great for comparing overlapping distributions

Create a density plot with `geom_density()`

Change the color saturation by setting `alpha` to `1/2`

```
ggplot(data = penguins) +  
  geom_density(  
    mapping = aes(x = body_mass_g,  
                  fill = species),  
    alpha = 1/2) +  
  labs_bodymass_vs_species
```



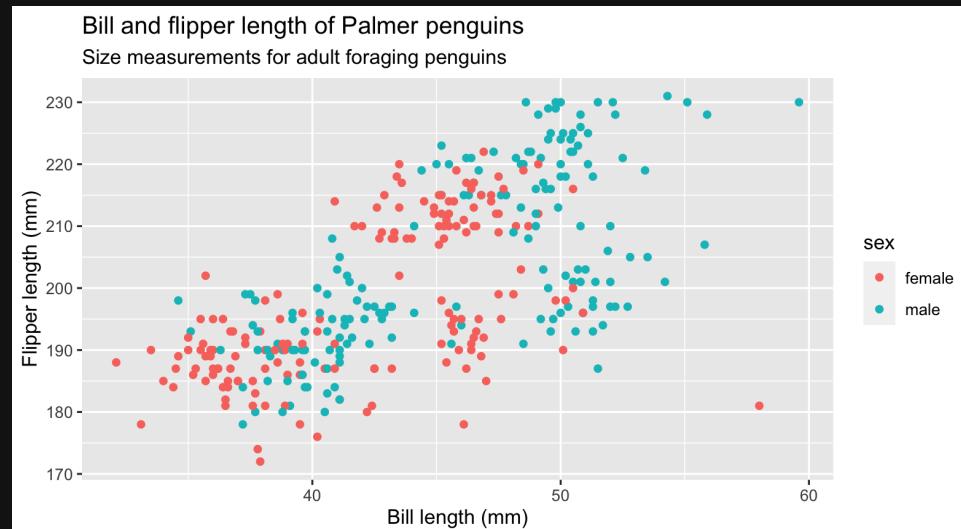
Also check out [ridgeline plots](#)

Mapping vs. setting aesthetics

Mapping vs. setting

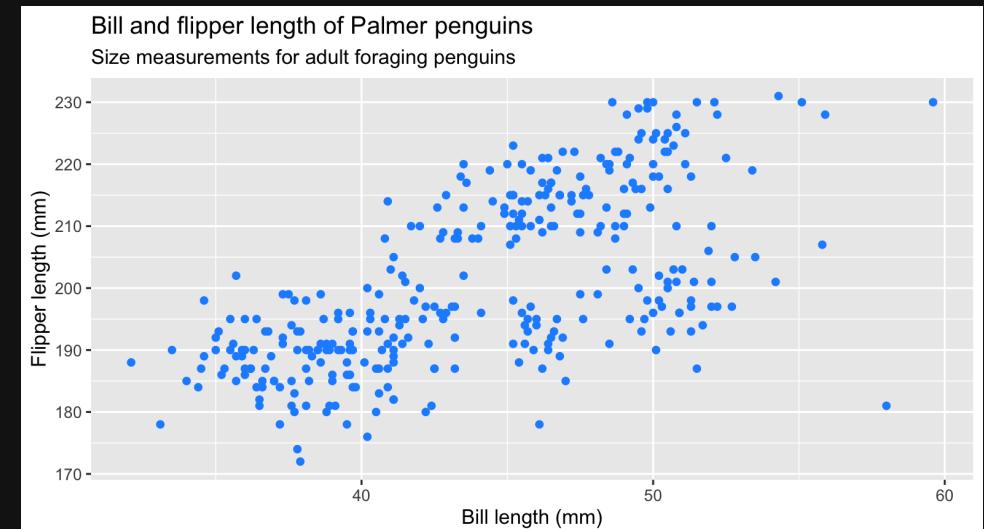
The last few graphs have mapped aesthetics inside and outside the `aes()` function

```
ggplot(data = penguins_no_miss) +
  geom_point(
    mapping = aes(x = bill_length_mm,
                  y = flipper_length_mm,
                  color = sex)) + # inside
  labs_bill_vs_flipper
```



Inside the `aes()` function is 'mapping', outside the `aes()` function is 'setting'

```
ggplot(data = penguins_no_miss) +
  geom_point(
    mapping = aes(x = bill_length_mm,
                  y = flipper_length_mm),
    color = "dodgerblue") + # outside
  labs_bill_vs_flipper
```



Mapping vs. setting

From [ggplot2 book](#)

If you want appearance to be governed by a variable, put the specification inside `aes()`; if you want override the default size or colour, put the value outside of `aes()`.

graph 12 Setting graph aesthetics

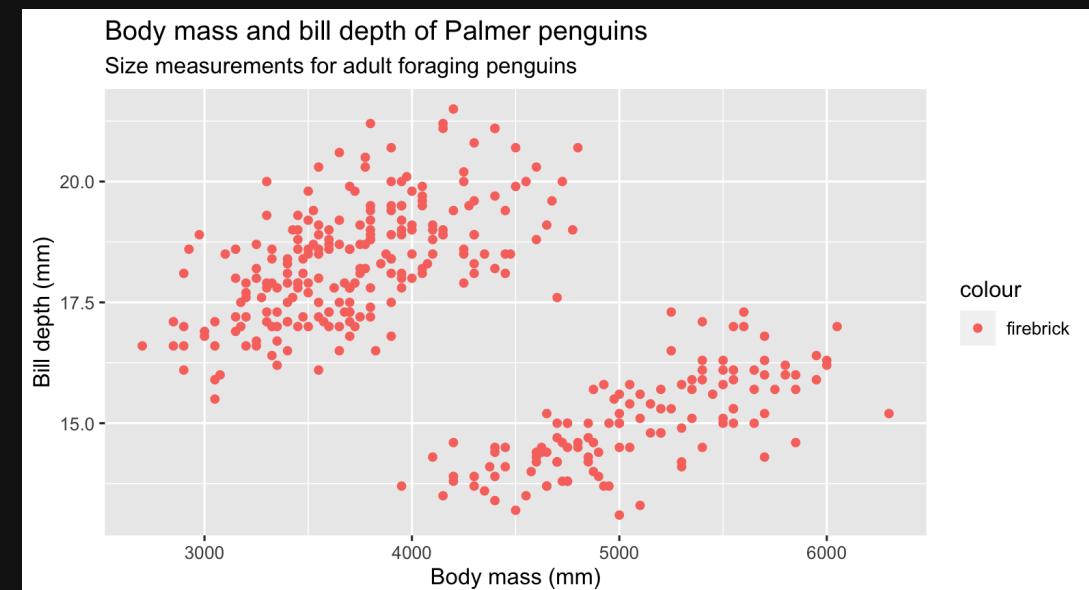
Change the code below to make the points "firebrick" red

Create labels

```
labs_body_mass_vs_bill_depth <- ggplot2::labs(
  title = "Body mass and bill depth of Palmer penguins",
  subtitle = "Size measurements for adult foraging penguins",
  x = "Body mass (mm)",
  y = "Bill depth (mm)")
```

What color will the points be on this graph?

```
ggplot(data = penguins) +
  geom_point(
    mapping = aes(x = body_mass_g,
                  y = bill_depth_mm,
                  color = "firebrick")) +
  labs_body_mass_vs_bill_depth
```



TIP: the legend tells us `geom_point()` is looking for a mapped variable in the penguins dataset named "firebrick"

Combining layers

graph 13 New layer, new data, no problem

Each `geom_*` function also has a `data` argument, so we can supply new data at each layer

Create a small dataset with max values from `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm` and `body_mass_g`

Create a dataset of the max body mass and bill depth
(`big_penguins`)

```
big_penguins <- bind_rows(  
  slice_max(.data = penguins, bill_length_mm, n = 1),  
  slice_max(.data = penguins, bill_depth_mm, n = 1),  
  slice_max(.data = penguins, flipper_length_mm, n = 1),  
  slice_max(.data = penguins, body_mass_g, n = 1))
```

Create data `label` and `source`

```
big_penguins <- mutate(.data = big_penguins,  
  label = case_when(  
    bill_length_mm == 59.6 ~ paste0("long bill = ", bill_length_mm),  
    bill_depth_mm == 21.5 ~ paste0("deep bill = ", bill_depth_mm),  
    flipper_length_mm == 231 ~ paste0("big flipper = ", flipper_length_mm),  
    body_mass_g == 6300 ~ paste0("big bird = ", body_mass_g)),  
  source = case_when(  
    bill_length_mm == 59.6 ~ "max bill length",  
    bill_depth_mm == 21.5 ~ "max bill depth",  
    flipper_length_mm == 231 ~ "max flipper length",  
    body_mass_g == 6300 ~ "max body mass"))
```

Our label dataset

View `label` and `source` from `big_penguins`

<code>label</code>	<code>source</code>
	<chr>
long bill = 59.6	max bill length
deep bill = 21.5	max bill depth
big flipper = 231	max flipper length
big bird = 6300	max body mass

4 rows

Objective: Create a scatter-plot to show the relationship between body mass, flipper length, and bill length.

graph 13 Layer 1

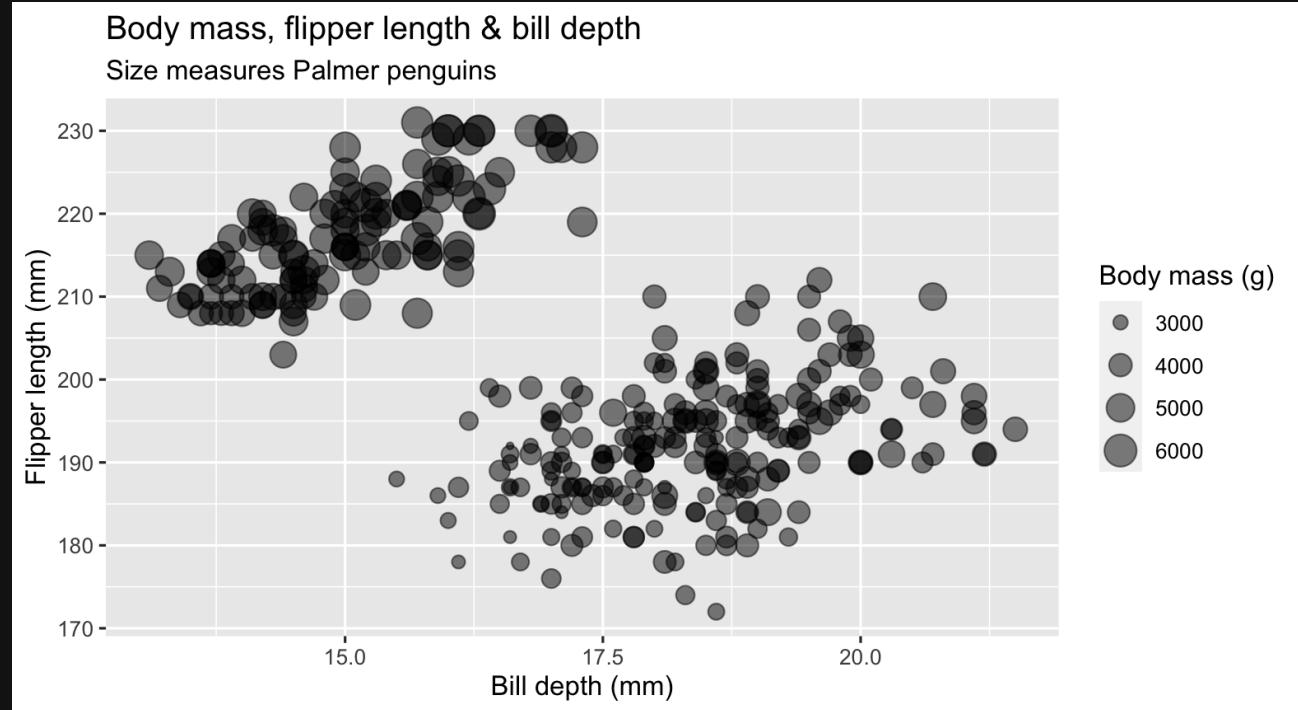
Create layer 1 with `geom_point()` using `size`

Create labels

```
labs_bodymass_bill_depth_flipper_length <- labs(
  title = "Body mass, flipper length & bill depth",
  subtitle = "Size measures Palmer penguins",
  x = "Bill depth (mm)",
  y = "Flipper length (mm)",
  size = "Body mass (g)")
```

Add new layer with new data

```
ggplot(data = penguins_no_miss) +
  # layer 1
  geom_point(
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  size = body_mass_g),
    alpha = 1/2) +
  # labels
  labs_bodymass_bill_depth_flipper_length
```



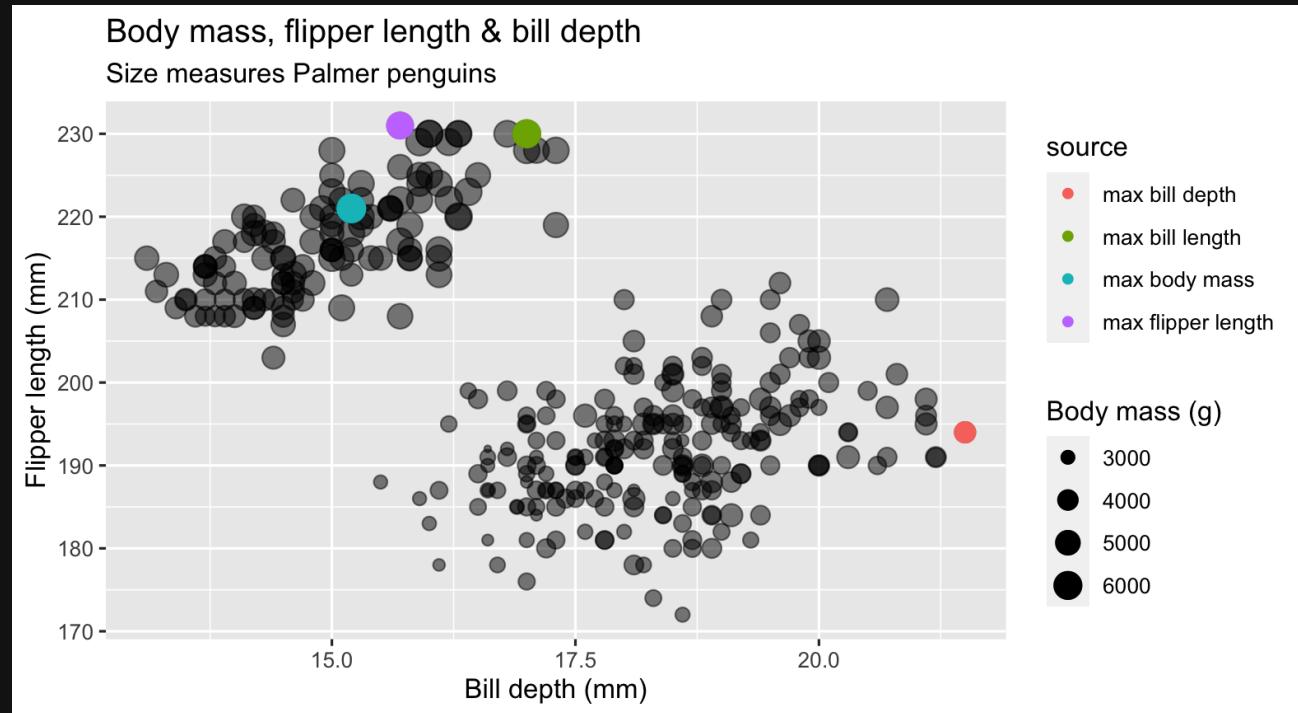
graph 14 Layer 2

Create layer 2 with another `geom_point()` using `color` and `size`

Set the `data` argument to `big_penguins`

Use `scale_size()` to adjust point scaling

```
ggplot(data = penguins_no_miss) +
  geom_point(
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  size = body_mass_g),
    alpha = 1/2) +
  # layer 2
  geom_point(data = big_penguins,
             mapping = aes(x = bill_depth_mm,
                           y = flipper_length_mm,
                           # color by source
                           color = source,
                           size = body_mass_g)) +
  # re-scale
  scale_size(range = c(1, 5)) +
  # labels
  labs_bodymass_bill_depth_flipper_length
```

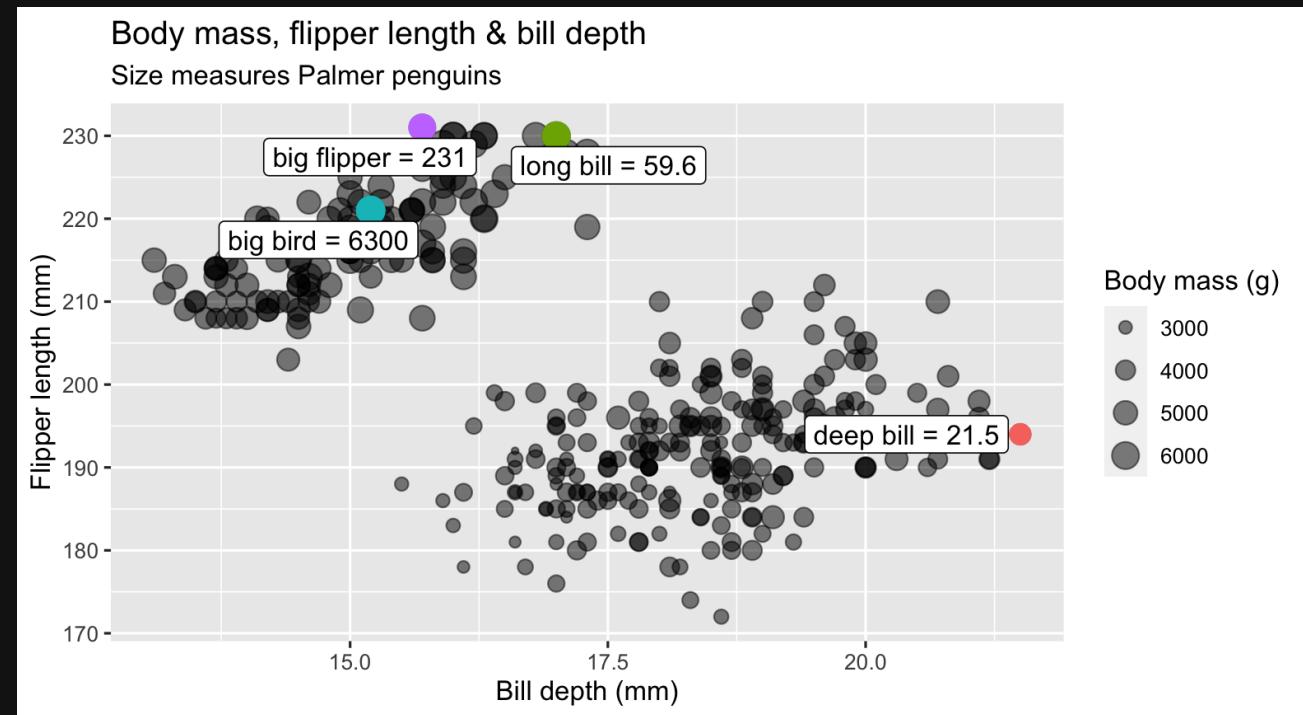


graph 15 Label 3 (max values)

Add layer 3 with the `geom_label_repel()` function from [ggrepel](#)

Add layer for `labels` in `big_penguins`

```
ggplot(data = penguins_no_miss) +
  geom_point(
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  size = body_mass_g),
    alpha = 1/2) +
  geom_point(data = big_penguins,
             mapping = aes(x = bill_depth_mm,
                           y = flipper_length_mm,
                           color = source,
                           size = body_mass_g),
             # remove legend
             show.legend = FALSE) +
  # rescale
  scale_size(range = c(1, 5)) +
  # layer 3
  ggrepel::geom_label_repel(
    data = big_penguins,
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  label = label)) +
  # labels
  labs_bodymass_bill_depth_flipper_length
```



Facets

Facets

From [ggplot2 book](#)

Small multiples are a powerful tool for exploratory data analysis: you can rapidly compare patterns in different parts of the data and see whether they are the same or different.

Facets = small multiples

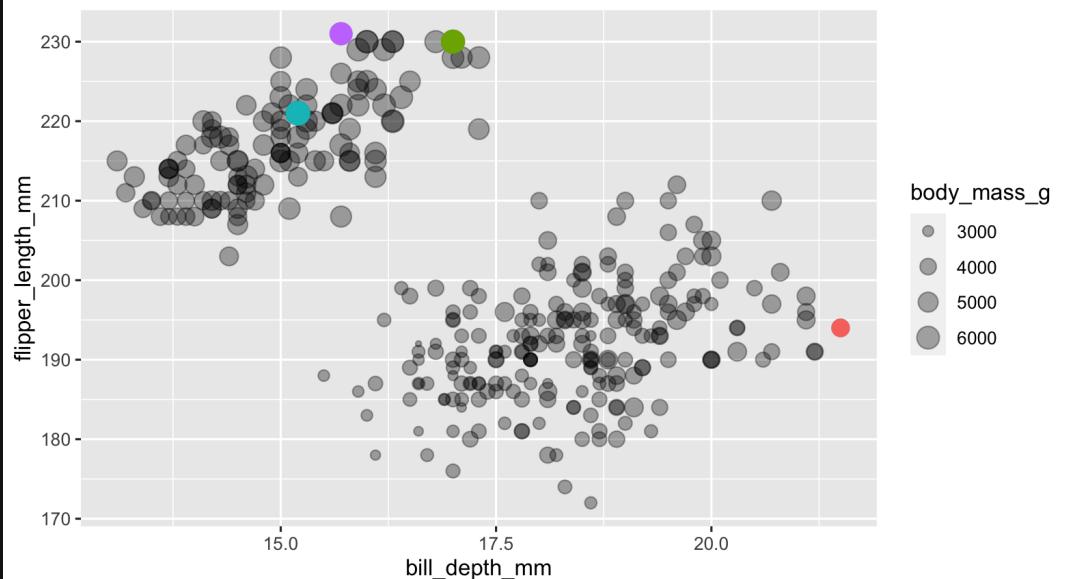
In the previous graph, we used multiple aesthetics (`color`, `size`, `shape`)

Can we explore these relationships by `sex` or `species`?

Store graph 15 in `gpp_penguin_measures`

```
gpp_penguin_measures <- ggplot(data =
penguins_no_miss) +
  geom_point(
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  size = body_mass_g),
    alpha = 1/3) +
  geom_point(data = big_penguins,
             mapping = aes(x = bill_depth_mm,
                           y = flipper_length_mm,
                           color = source,
                           size = body_mass_g),
             show.legend = FALSE) +
  scale_size(range = c(1, 5))
```

gpp_penguin_measures

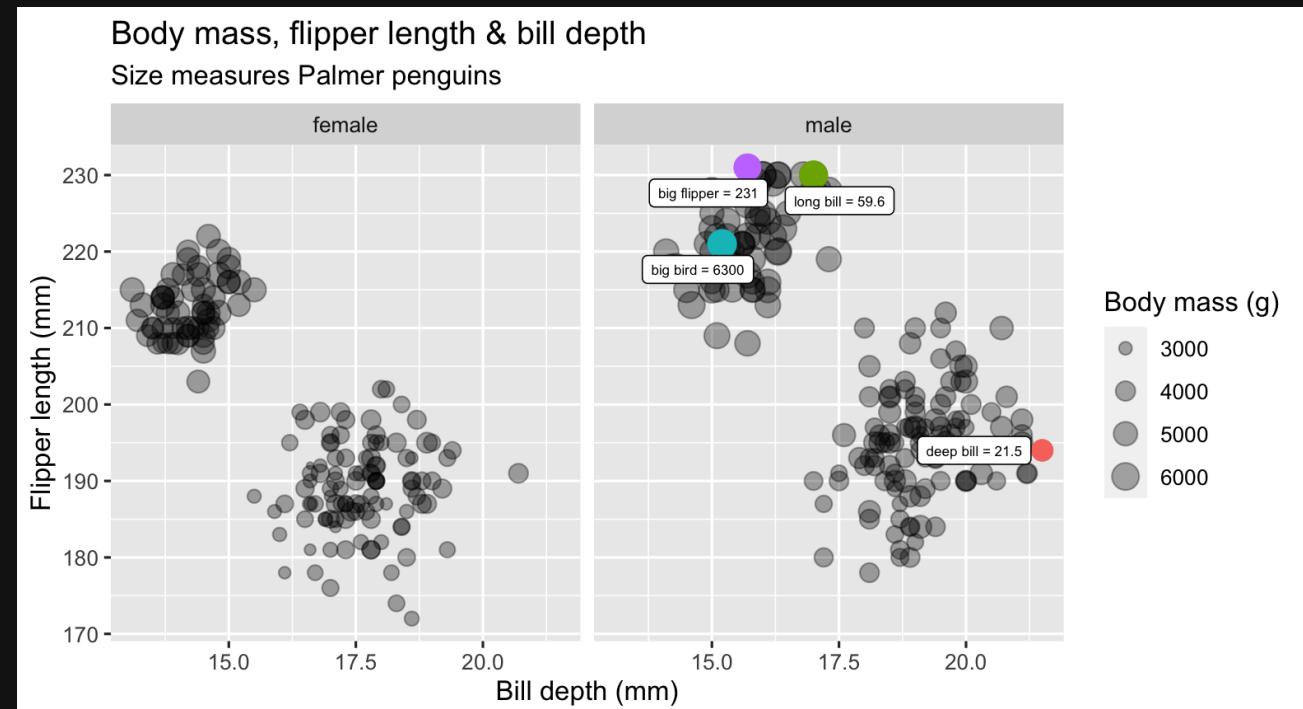


graph 16 Facet by sex

Use `facet_wrap()` to view our previous graph by `sex`

`facet_wrap()` uses `~ [var]`

```
ggp_penguin_measures +
  ggrepel::geom_label_repel(
    data = big_penguins,
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  label = label),
    # adjust size
    size = 2) +
  # facet by sex
  facet_wrap(~ sex) +
  # labels
  labs_bodymass_bill_depth_flipper_length
```



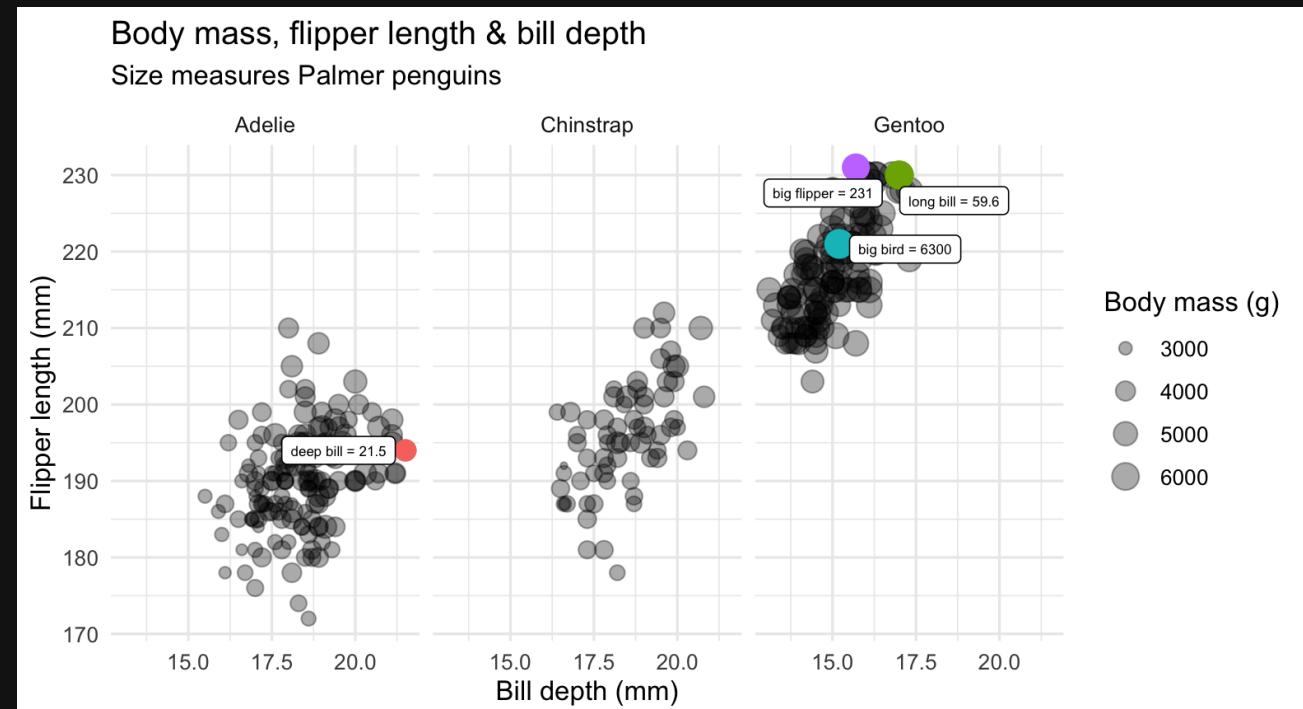
graph 17 Facet by species

Change `facet_wrap()` to build graphs by `species` and add theme

Change `facet_wrap()` to ~ `species`

Add `theme_minimal()`

```
ggp_penguin_measures +
  ggrepel::geom_label_repel(
    data = big_penguins,
    mapping = aes(x = bill_depth_mm,
                  y = flipper_length_mm,
                  label = label),
    size = 2) +
  # change to species
  facet_wrap(~ species) +
  # add theme
  theme_minimal() +
  # labels
  labs_bodymass_bill_depth_flipper_length
```



Recap

What we've covered

1. Build labels (set your expectations)
2. View data before building any graphs
3. Building graphs layer-by-layer (data, mapping, geom)
4. Mapping variables to graph elements (color, position, size, etc)
5. Extending graphs by combining layers
6. Using facets to explore relationships

Thanks!

[@mjfrigaard](https://twitter.com/mjfrigaard) 

[@mjfrigaard](https://mastodon.social/@mjfrigaard) 

[mjfrigaard@pm.e](mailto:mjfrigaard@pm.me) 

[What does "λέξις" mean?](#)