Data Visualization with ggplot2

ODSC West

Outline

Part 1

- *Why do we need graphs?*
- *An exploratory mindset*
- *Surprise or confirm, then communicate*
- *The grammar of graphics*

- **Part 2**
- *RStudio Cloud*
- *Exercises & solutions*
- *Creating a graph (layer-by-layer)*
- *Applying the grammar*

Part 1

- Intros 👋
- Workshop materials ⬇️
- Basic understand of `ggplot2` syntax ✔️
- Build your first graph! ✔️

Why do we need graphs?

Raw data don’t communicate well

It’s hard to make sense of millions of rows and/or thousands of columns

Fortunately, we are **excellent at seeing patterns**:

“the human brain has a superior ability to mentally manipulate animate and inanimate patterns into a myriad of intangible symbols that can then be recombined to produce new images of the world;”

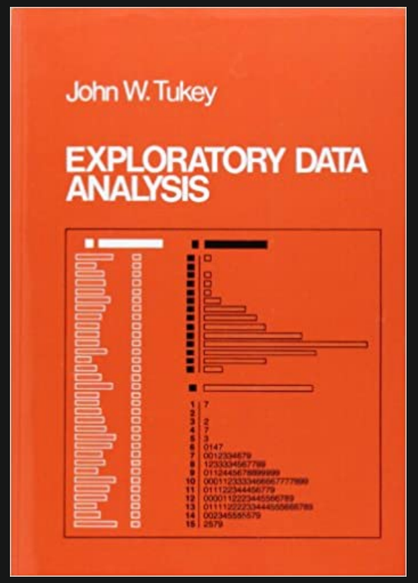
“we therefore live partly in worlds of our own mental creation, super-imposed upon or distinct from the natural world.”

- Source: Superior pattern processing is the essence of the evolved human brain

Graphs allow us to explore complexity with *symbols* and *images*

Exploratory Data Analysis

"Exploratory Data Analysis (EDA)" first coined by American mathematician John Tukey in 1977



"The greatest value of a picture is when it forces us to notice what we never expected to see."

- John Tukey, 1977

Exploration requires ‘listening’

“The role of the data analyst is to listen to the data in as many ways as possible until a plausible ‘story’ of the data is apparent”

- John T. Behrens, Principles and Procedures of Exploratory Data Analysis

Exploration is a ‘state of mind’

“More than anything, EDA is a state of mind. During the initial phases of EDA you should feel free to investigate every idea that occurs to you. Some of these ideas will pan out, and some will be dead ends…”

“As your exploration continues, you will hone in on a few particularly productive areas that you’ll eventually write up and communicate to others.”

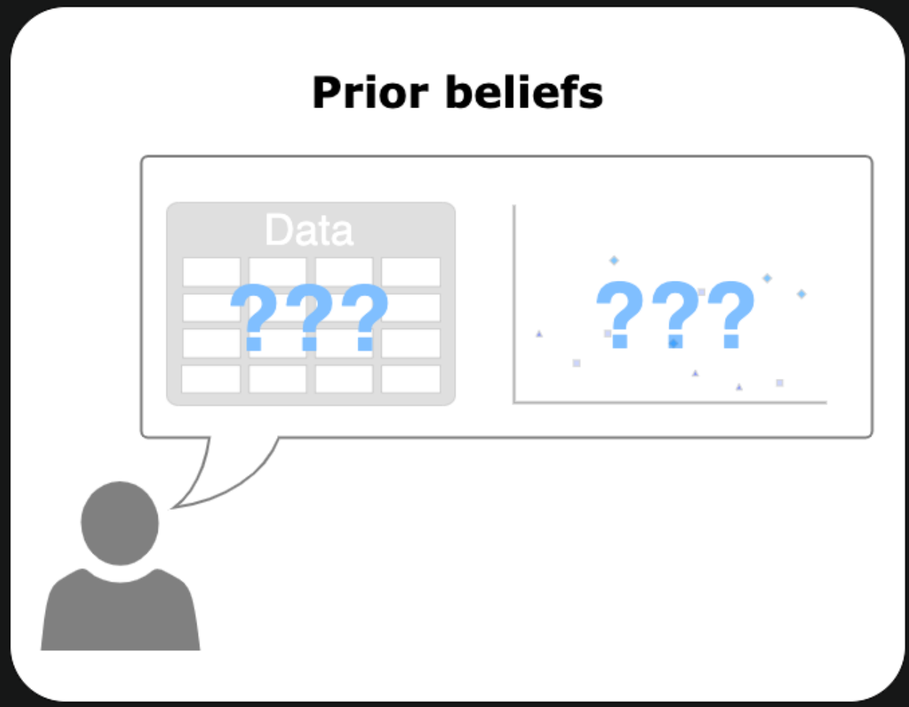
- Hadley Wickham, R for Data Science

An Exploratory Mindset

Exploration requires a Bayesian Mindset (1 of 3)

We all have implicit beliefs, or priors, about the world

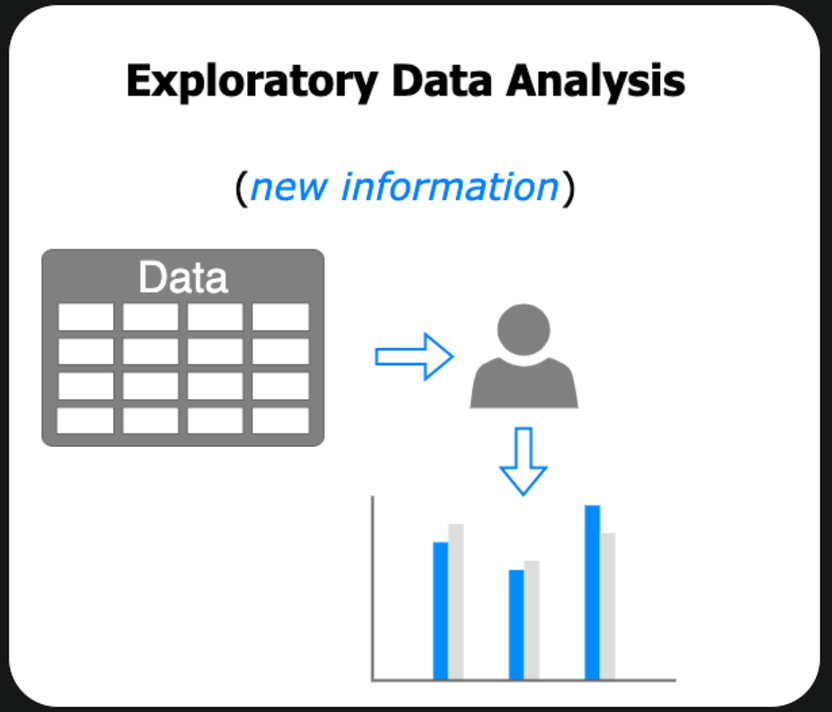
What we think we know (i.e.,our expectations)



Exploration requires a Bayesian Mindset (2 of 3)

When we encounter new information or data, our priors get updated

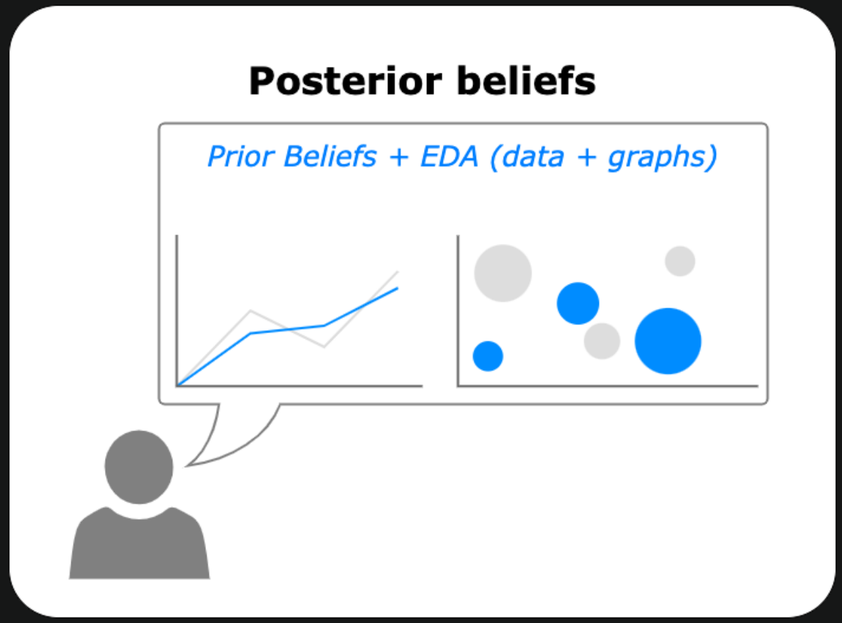
Our expectations + new data (i.e., what we see)



Exploration requires a Bayesian Mindset (3 of 3)

Our updated beliefs, or posteriors, depend on our priors ***and*** our perceptions of the new information

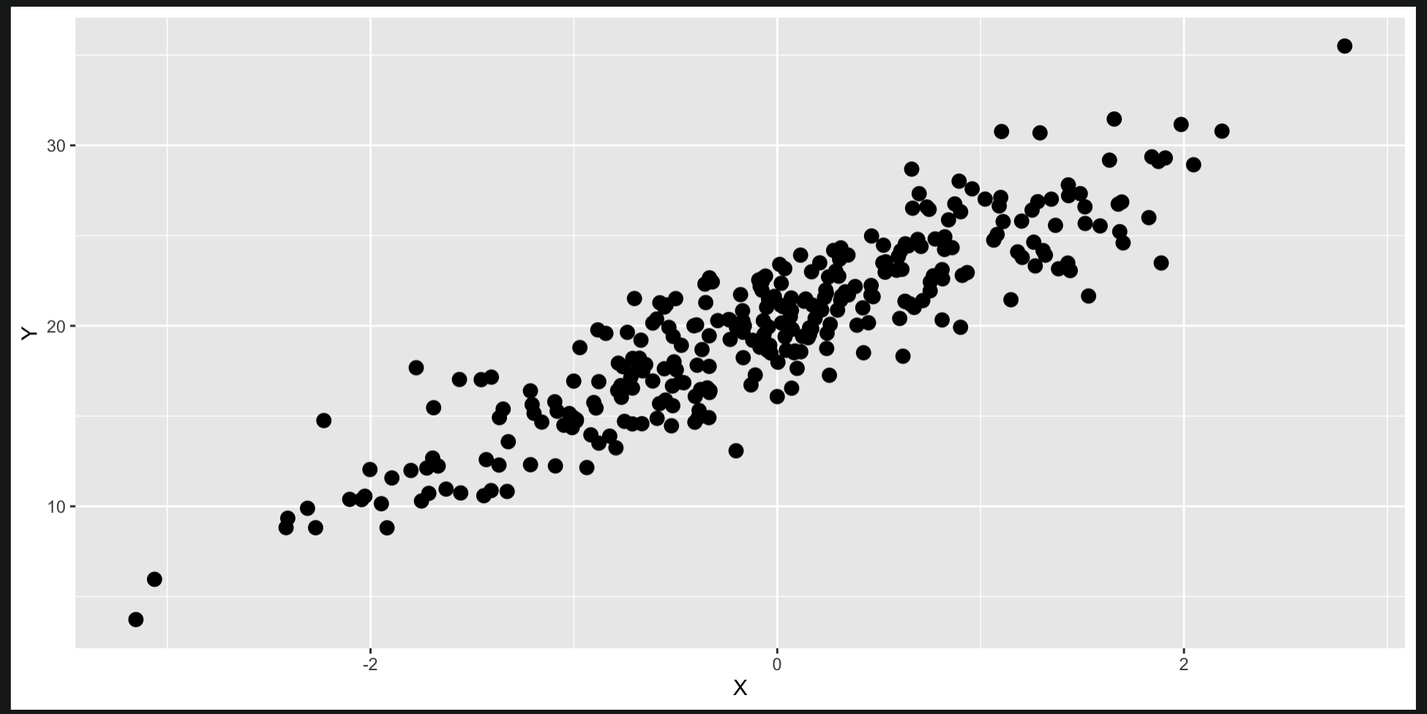
What we expect + what we see =
what we've learned



Graphs can confirm our expectations

*What if our expectation was that **X** is related to **Y**?*

...then we graphed the data...

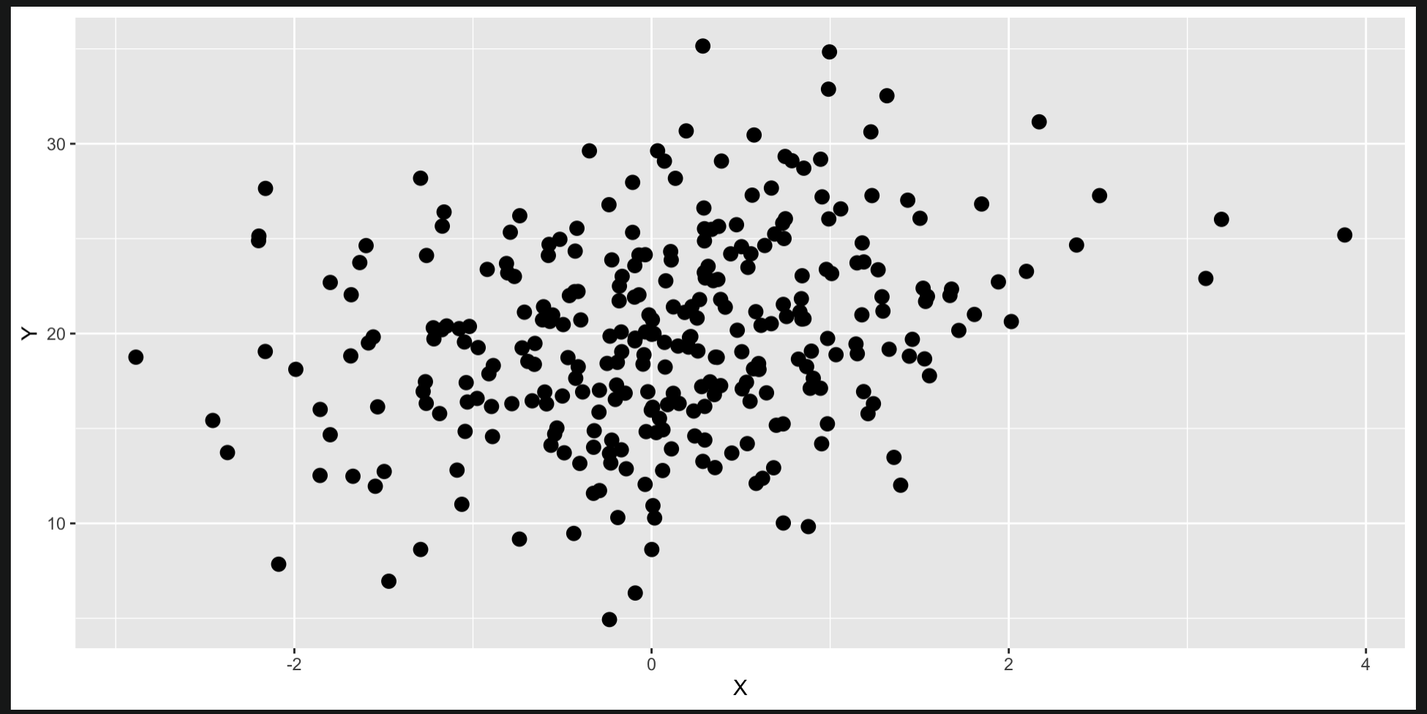


*We would say our expectations have been **confirmed***

Graphs can refute our expectations

*What if our expectation was that **X** is related to **Y**?*

...then we graphed the data...



*We would say our expectations have been **refuted***

ggplot2: grammar & syntax

Grammar

The system of rules for any given language

Includes:

1. Word meanings
2. Internal structure
3. Word arrangement

Syntax

The form, structure and order for constructing statements

Students Cook & Serve Grandparents

On Thursday, September 9, Gorman School hosted the first annual Grandparent's Day.

All Grandparents were invited to a school wide pancake breakfast. Upper grade students served as excellent chefs, as well as taking responsibility for serving the food and the clean up after-

```
[[students][[cook][and][serve grandparents]]]
```

```
[[students][[cook and serve][grandparents]]]
```

`ggplot2` : grammar & syntax

Built on top of the grammar & syntax of R

> *"In R, objects are like nouns, and functions (**fn**) are like verbs"*

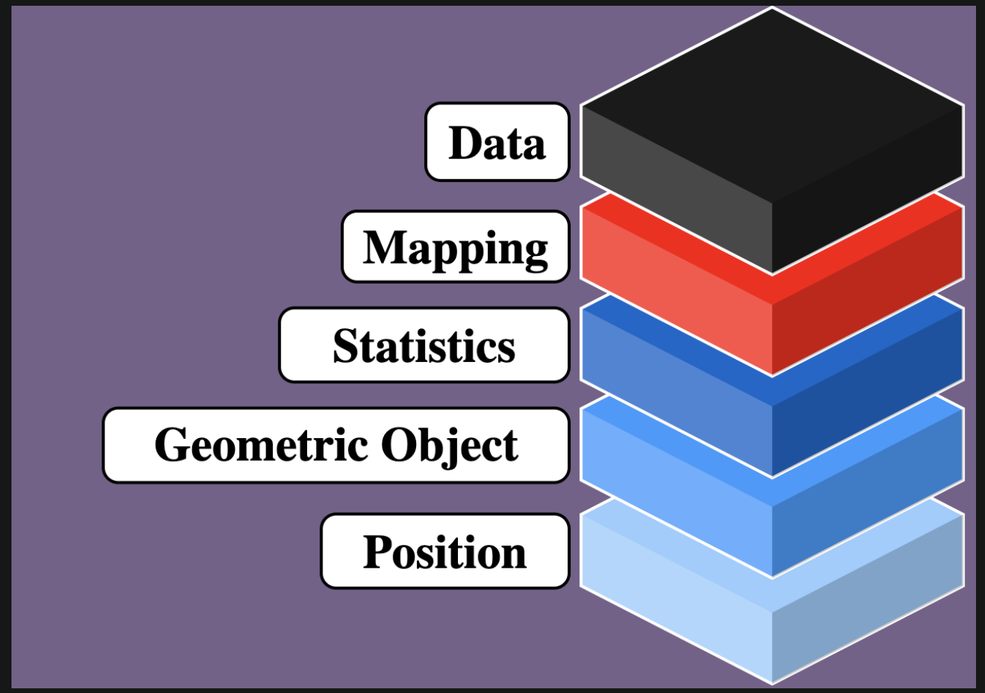
```
1  fn(object)
```

functions do things to *objects*

`ggplot2`: a layered language for graphs

`ggplot2` is comprised of layers

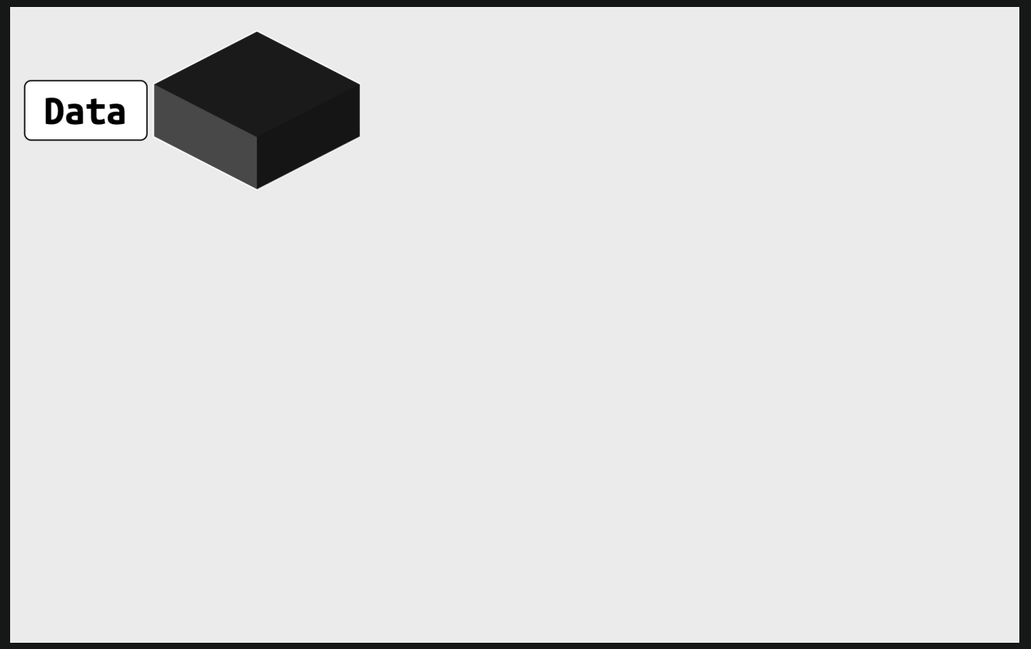
- Data
- Mapping
- Statistics
- Geometric objects
- Position adjustments



ggplot2: data

The **data layer** consists of a rectangular object (like a spreadsheet) with columns and rows

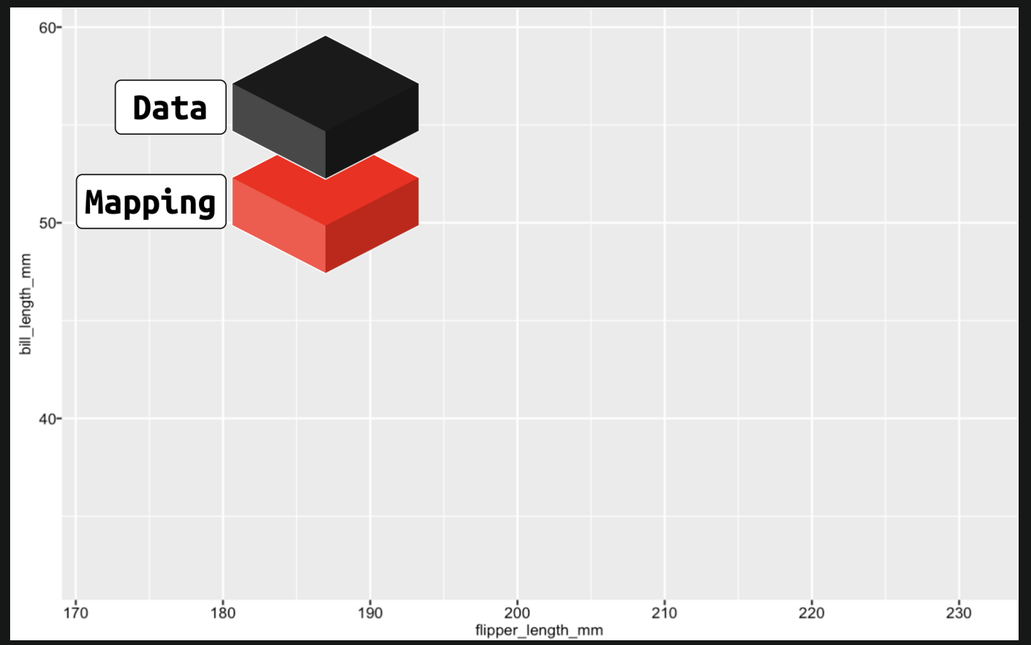
```
1  ggplot(data = penguins)
```



ggplot2: mapping

The **mapping layer** assigns columns (variables) from the data to a visual property (i.e. graph 'aes'thetic)

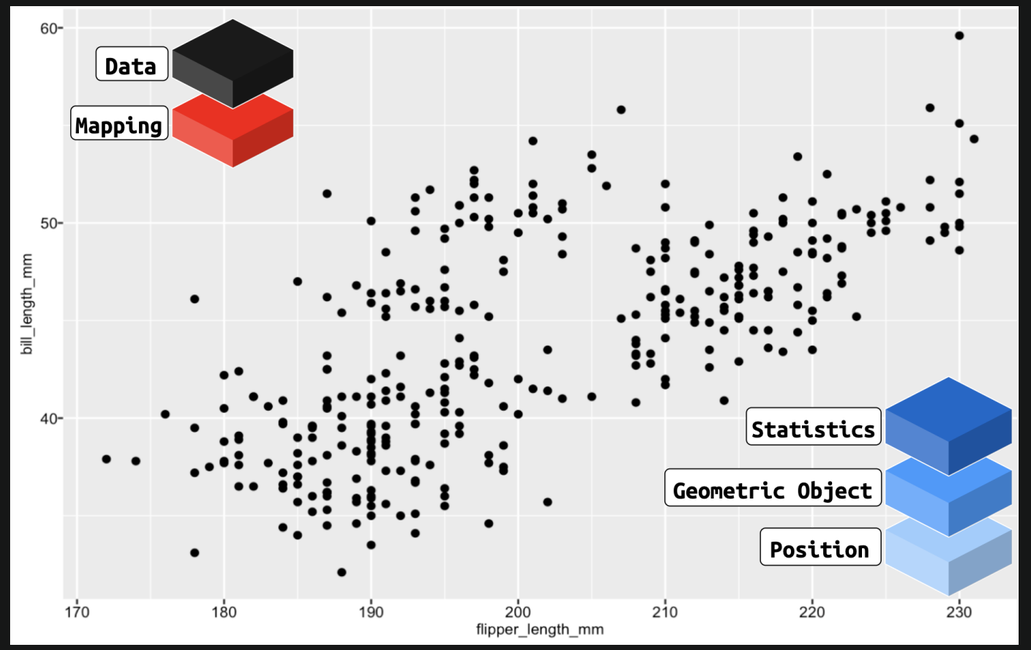
```
1 ggplot(data = penguins,
2   mapping = aes(x = flipper_length_mm
3                 y = bill_length_mm))
```



ggplot2: geoms

`geom_*()` functions include statistical transformations, shapes, and position adjustments for how to 'draw' the data on the graph

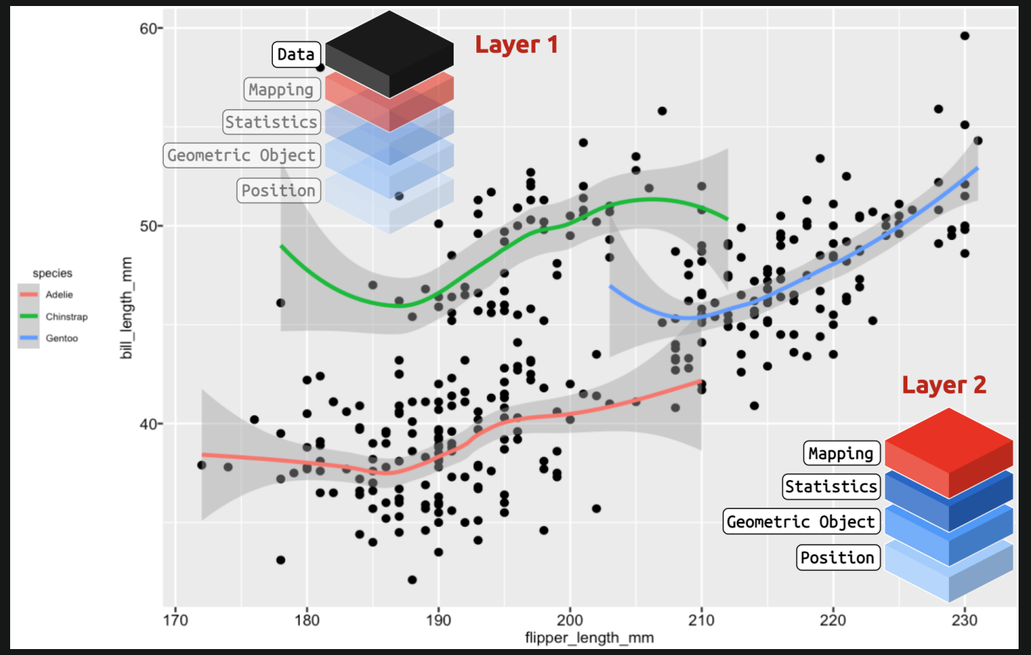
```
1  ggplot(data = penguins,
2    mapping = aes(
3      x = flipper_length_mm,
4      y = bill_length_mm)) +
5    geom_point()
```



ggplot2: layers

We can have multiple layers (data, mappings, geoms) in a single graph

```
ggplot(data = penguins,
 # layer 1
  mapping = aes(
    x = flipper_length_mm,
    y = bill_length_mm)) +
  geom_point()
# layer 2
  geom_smooth(
    mapping = aes(
      x = flipper_length_mm,
      y = bill_length_mm,
      color = species))
```



Layers = infinitely extensible

`ggplot2` is a system for,

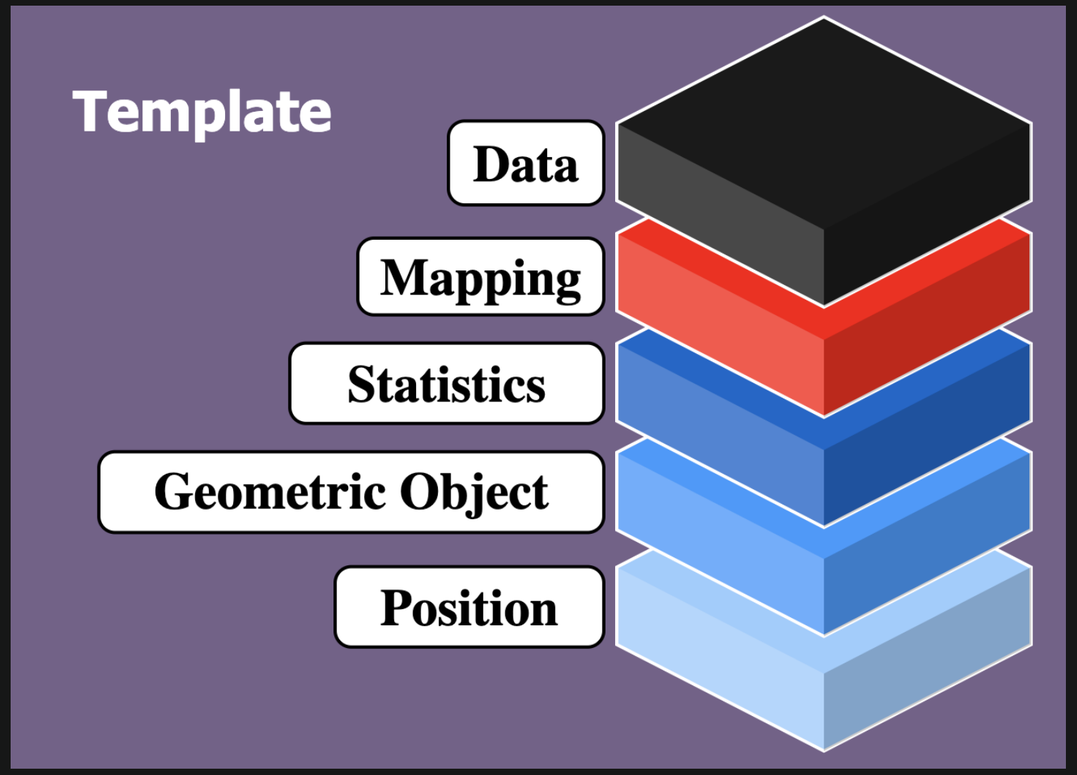
> *"making infinite use of finite means"* - Wilhelm von Humboldt

With a finite number of objects & functions, we can combine `ggplot2`'s grammar and syntax to create an infinite number of graphs!

ggplot2: templates

Basic Template: Data, aesthetic mappings, geom

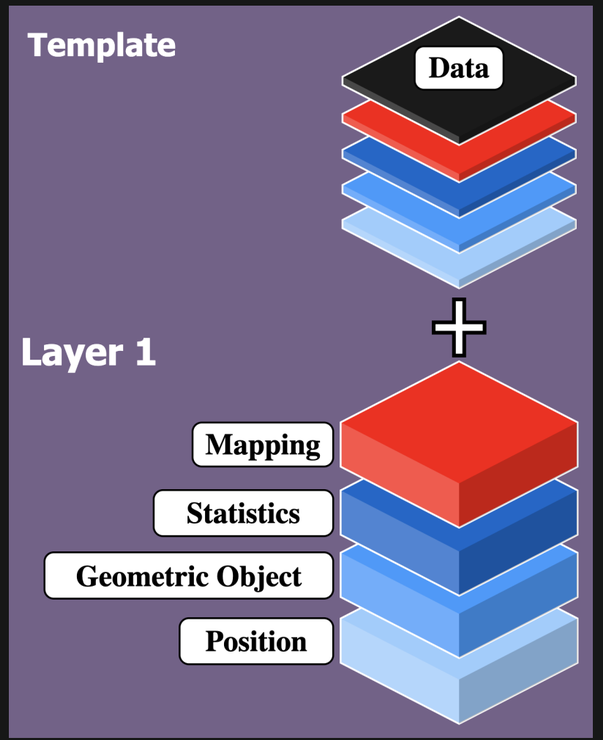
```
1 ggplot(data = <DATA>) +
2   geom_*(mapping = aes(<AESTHETIC MAPPINGS>))
```



ggplot2: more templates

Template + 1 Layer: more geoms and more aesthetic mappings

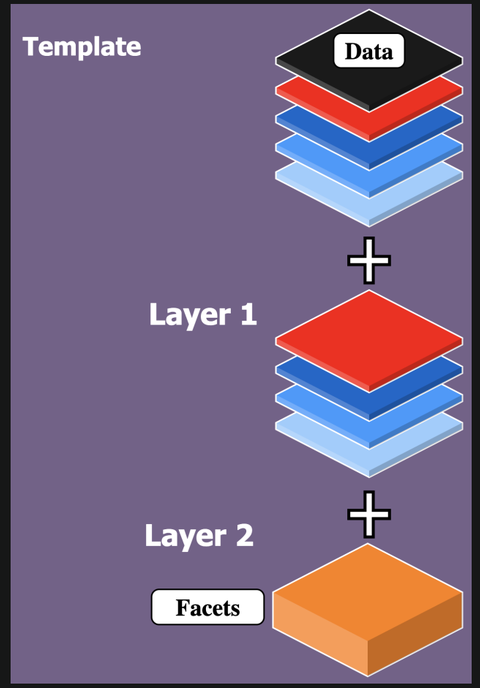
```
1  ggplot(data = <DATA>) +
2      geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
3      geom_*(mapping = aes(<AESTHETIC MAPPINGS>))
```



ggplot2: even more!

Template + 1 Layer + Facet Layer: template, more aesthetic mappings, and facets!

```
ggplot(data = <DATA>) +
    geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
    geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
    facet_*
```



Templates = infinitely extensible!

Themes

```
ggplot(data = <DATA>) +
    geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
    geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
    facet_* +
    theme_*
```

Don’t forget labels!

```
ggplot(data = <DATA>) +
    geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
    geom_*(mapping = aes(<AESTHETIC MAPPINGS>)) +
    facet_* +
    theme_* +
    labs()
```

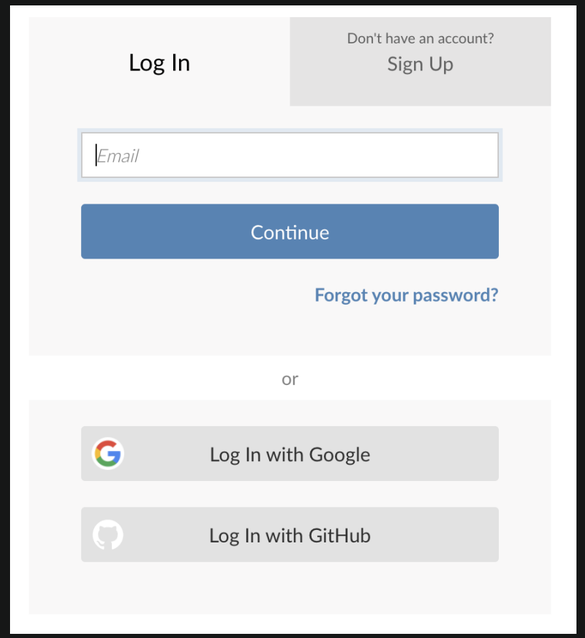
Part 2

- RStudio Cloud ✓
- Exercises & solutions ✓
- Creating a graph (layer-by-layer) ✓
- Applying the grammar ✓

RStudio.Cloud

RStudio.Cloud: Set up (1 of 4)

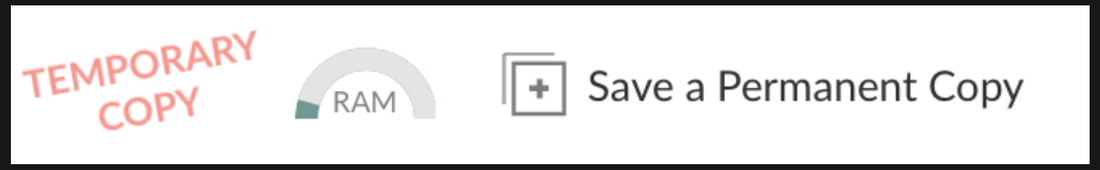
Head to RStudio.Cloud, you will see the following:



Log in with your GitHub credentials

RStudio.Cloud: Set up (2 of 4)

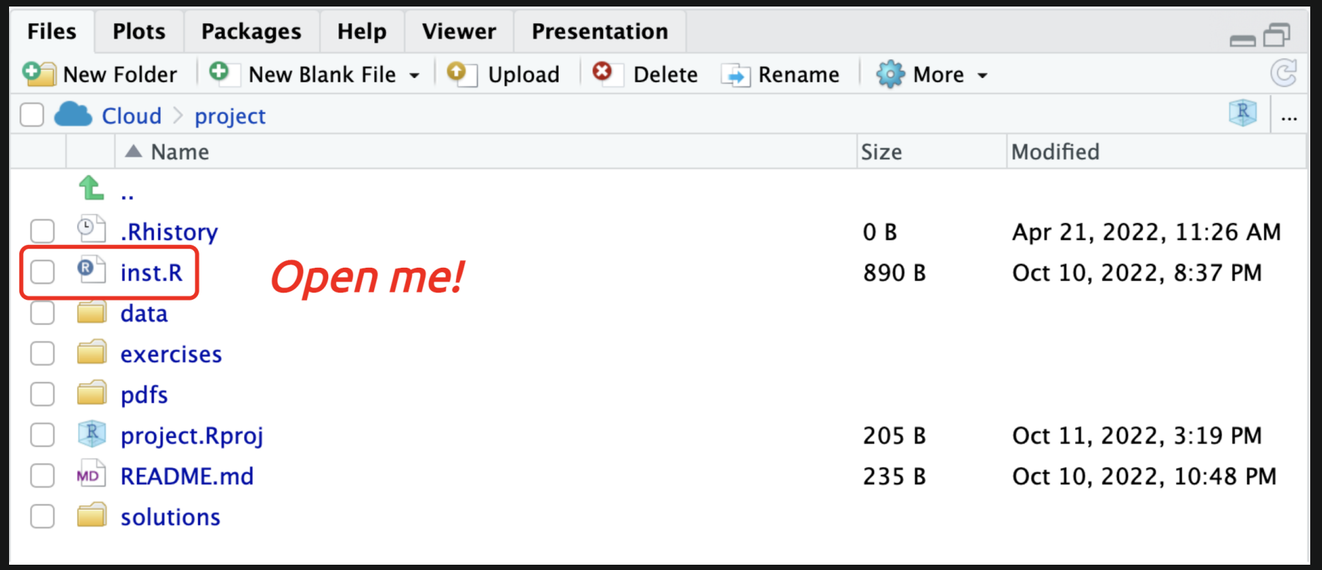
On the top of the RStudio IDE, you will see the following:



Click on ***Save a Permanent Copy*** to add this project to your workspace

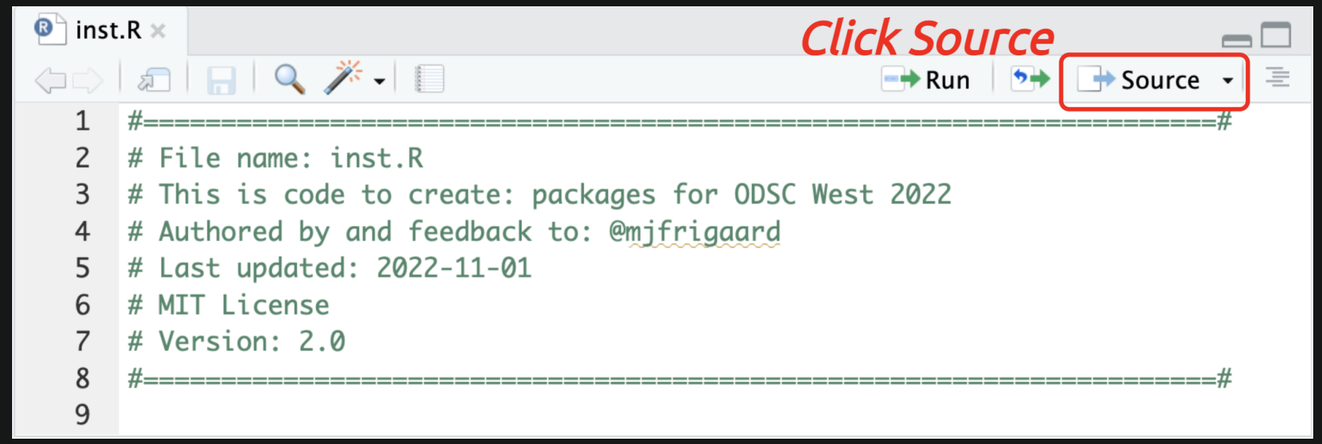
RStudio.Cloud: Set up (3 of 4)

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



RStudio.Cloud: Set up (4 of 4)

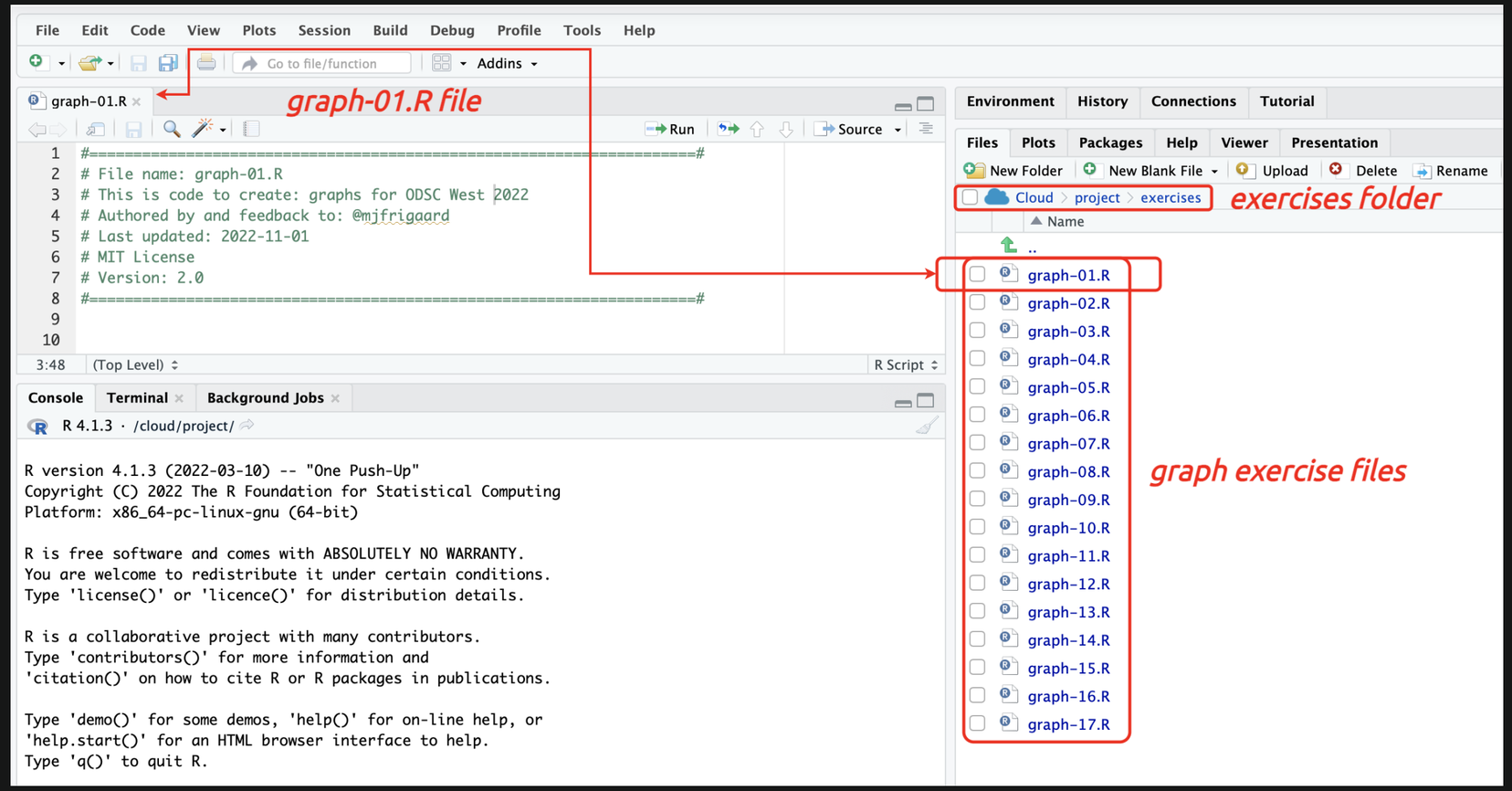
In the Source pane, click on the Source icon to run `inst.R`



This sends the code in `inst.R` to the **Console**

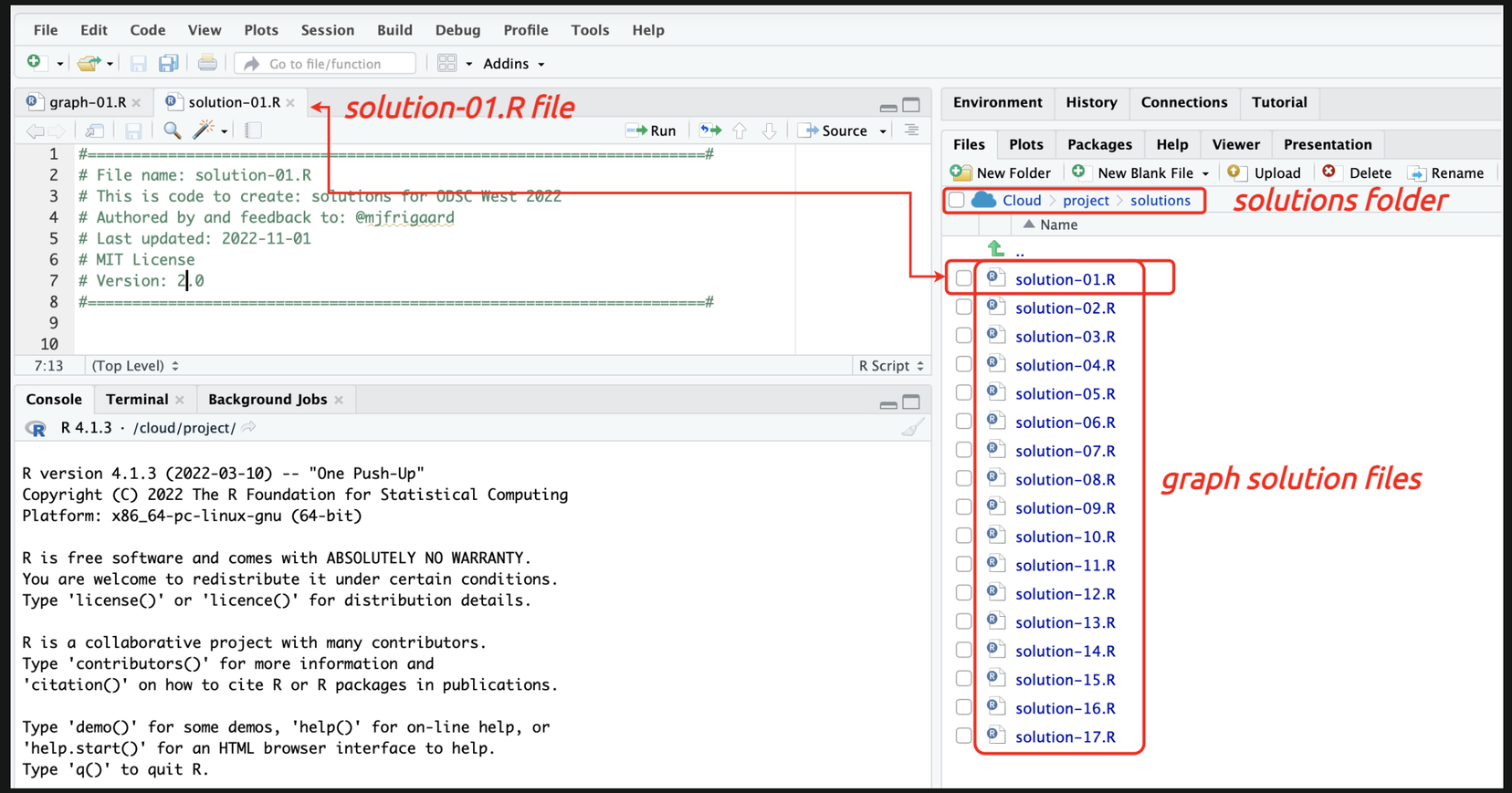
RStudio.Cloud: Exercises

The exercises are in the `exercises/` folder



RStudio.Cloud: Solutions

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



Quick Tip

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

…it doesn’t always produce a tangible result…

…but creating visualizations is rewarding!!!

ggplot2: build the labels first!

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

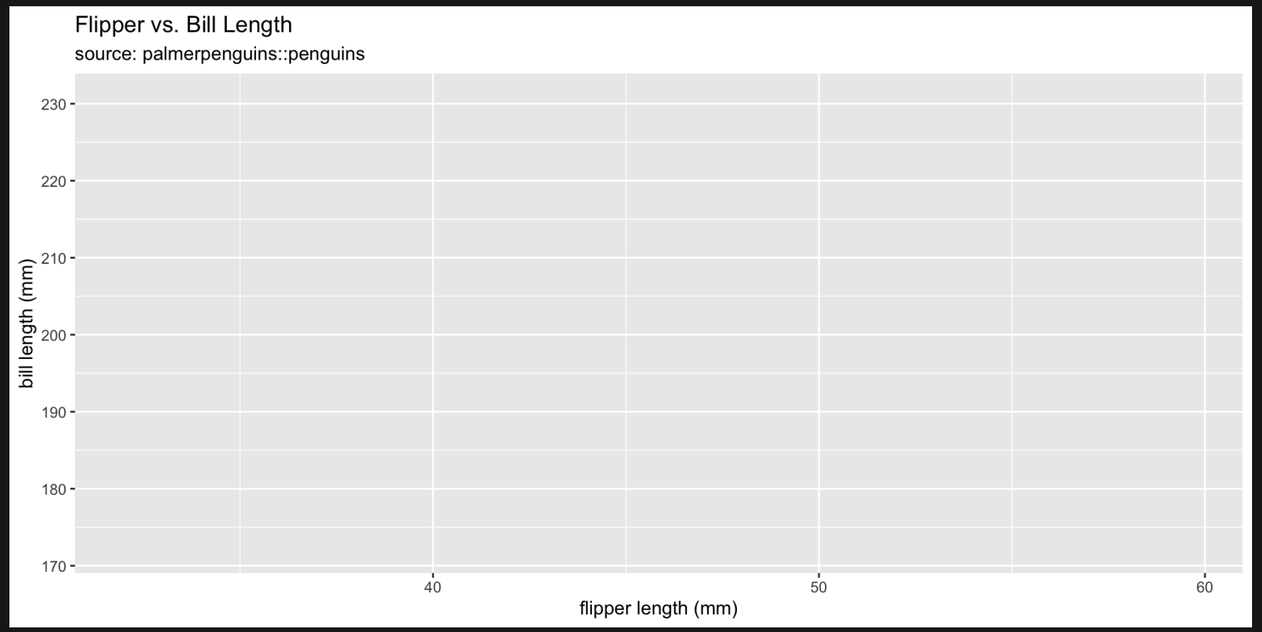
```
labs_pengiuns <- ggplot2::labs(
  title = "Flipper vs. Bill Length",
  subtitle = "source: palmerpenguins::penguins",
  x = "flipper length (mm)",
  y = "bill length (mm)")
```

<- expectations

ggplot2: build graph, check labels

Build labels, build graphs, then check labels!

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

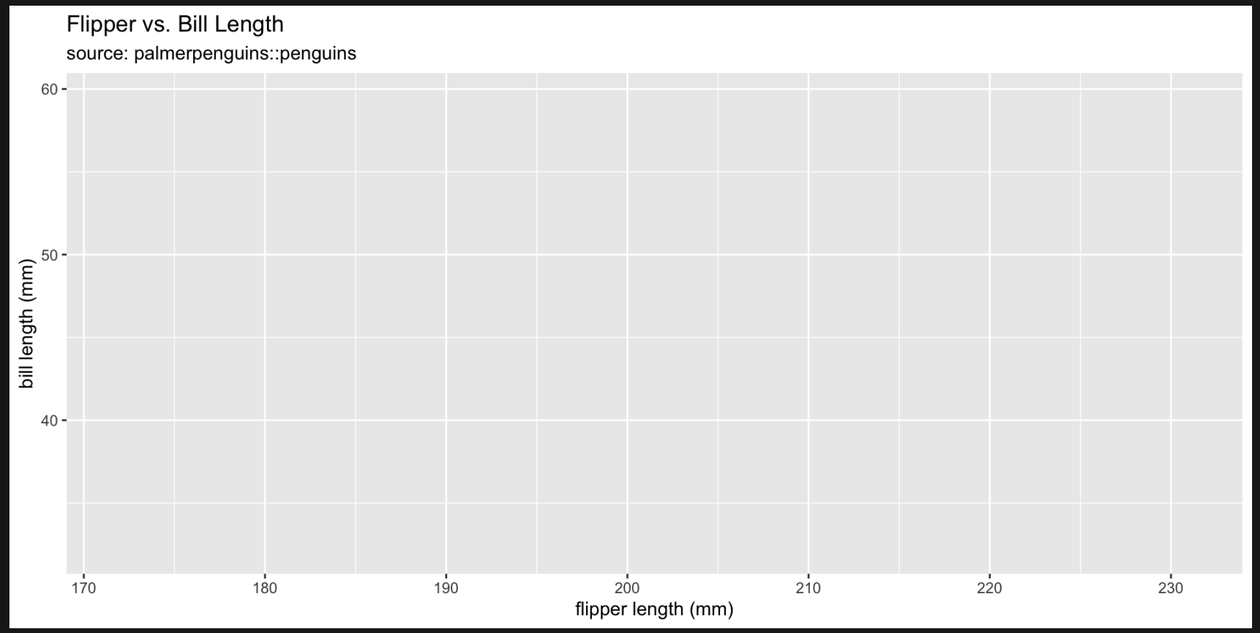


What's wrong here?

ggplot2: build graph, check labels, *revise*

x and y are flipped!

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



Fixed!

On the importance of revision:

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.”*

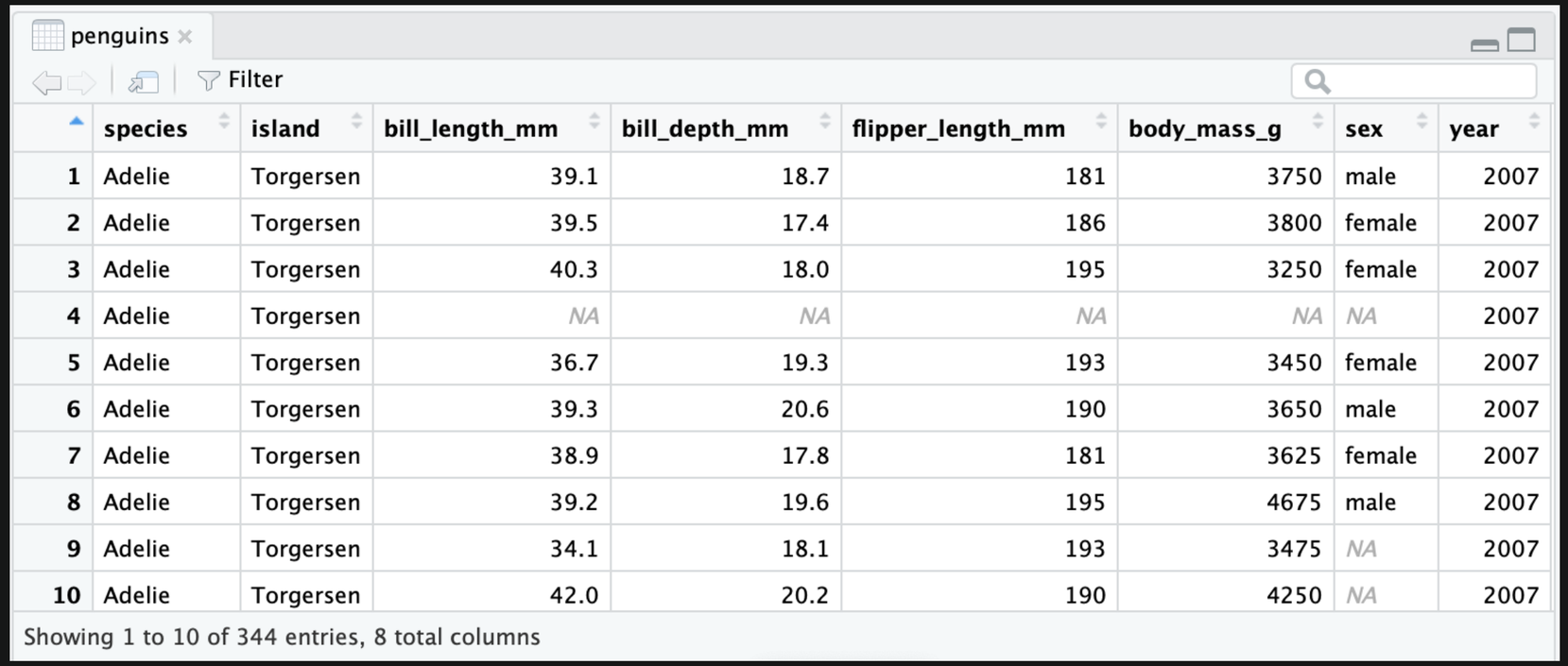
*“In short, it forces a writer to get up his courage and **expose his thinking process to his own intelligence**.”*

- Source: — Marvin H. Swift, HBR Clear Writing Means Clear Thinking Means...

The data

Viewing data (1 of 3)

`View()` opens the RStudio data viewer



penguins

Filter

	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
10	Adelie	Torgersen	42.0	20.2	190	4250	*NA*	2007

Showing 1 to 10 of 344 entries, 8 total columns

Viewing data (2 of 3)

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

```
1  glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel…
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse…
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, …
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, …
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186…
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, …
$ sex               <fct> male, female, female, NA, female, male, female, male…
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007…
```

Viewing data (3 of 3)

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

```
1 str(penguins)
```

```
tibble [344 × 8] (S3: tbl_df/tbl/data.frame)
 $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 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: 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_flippper <- 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`: Initialize plot with data

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

Place `penguins` in the `data` argument

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

This gives us a blank canvas!

graph 02: Map variables to positions

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

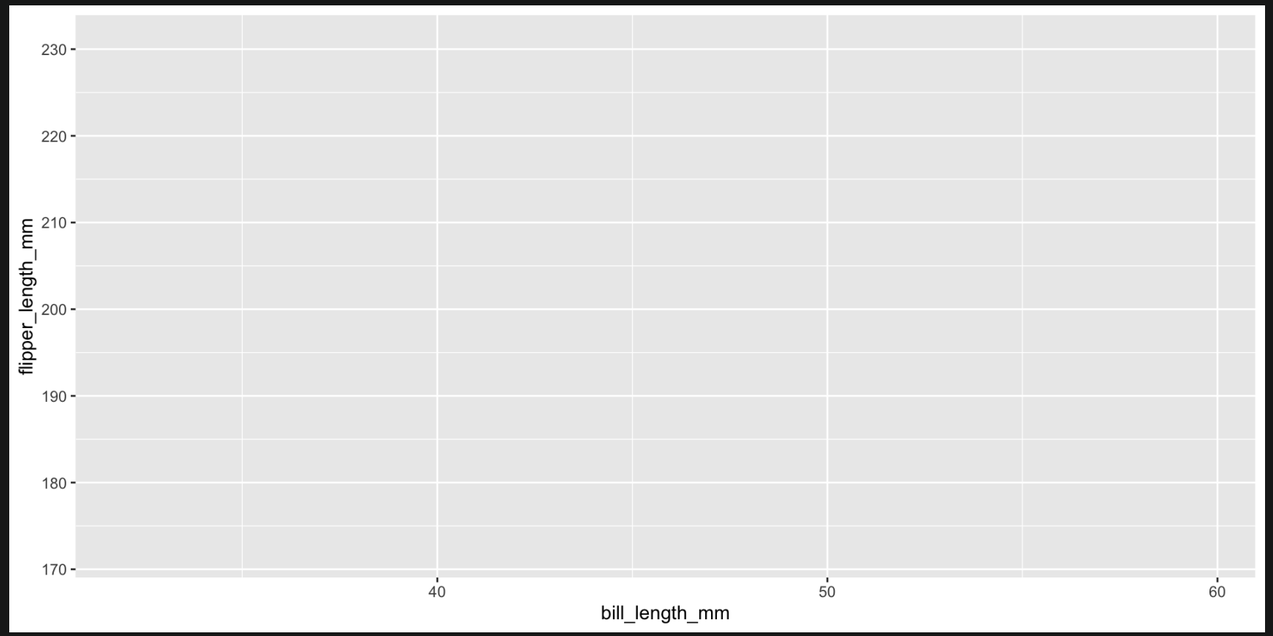
Map `bill_length_mm` to `x`

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



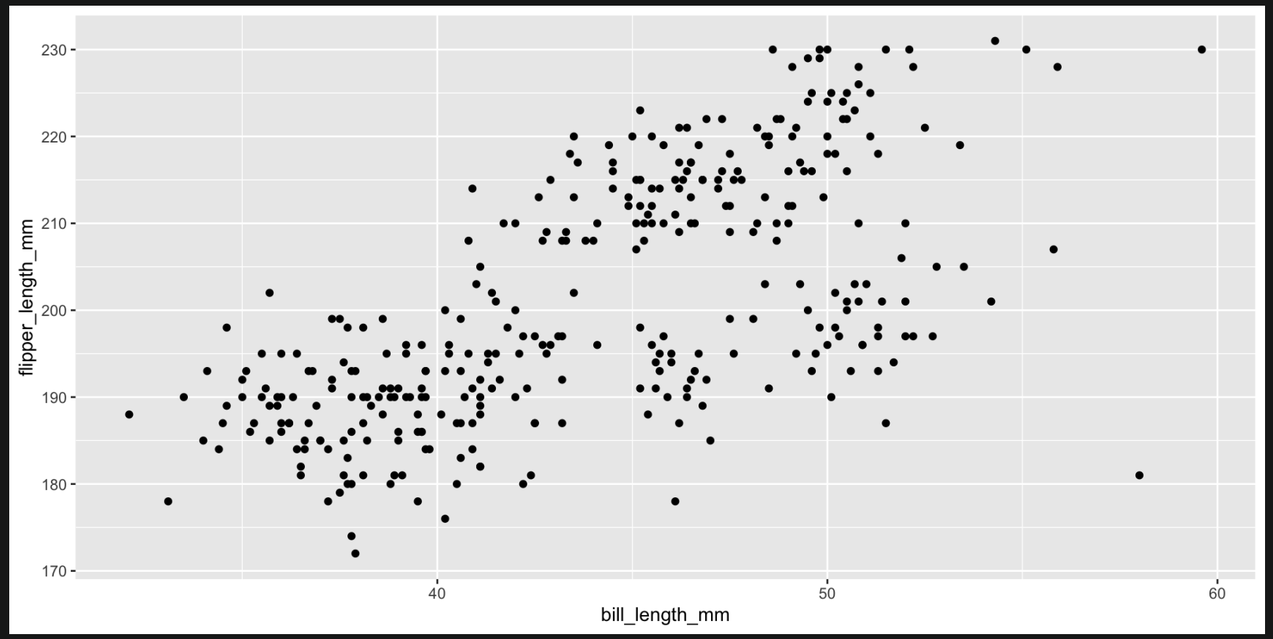
graph 03: Adding geoms

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

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

Don't confuse this with the pipes (`|>` or `%>%`)

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



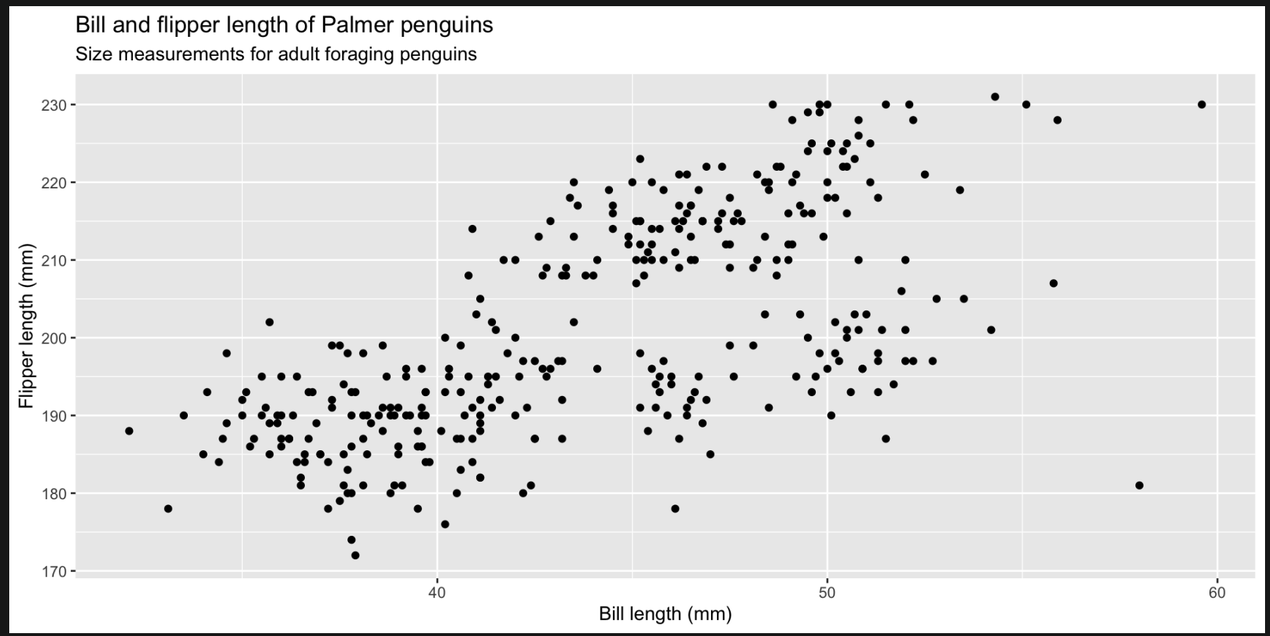
graph 04: Don't forget the labels!

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

Add labels with `+`

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

And we have our first graph!



Global vs. local mapping

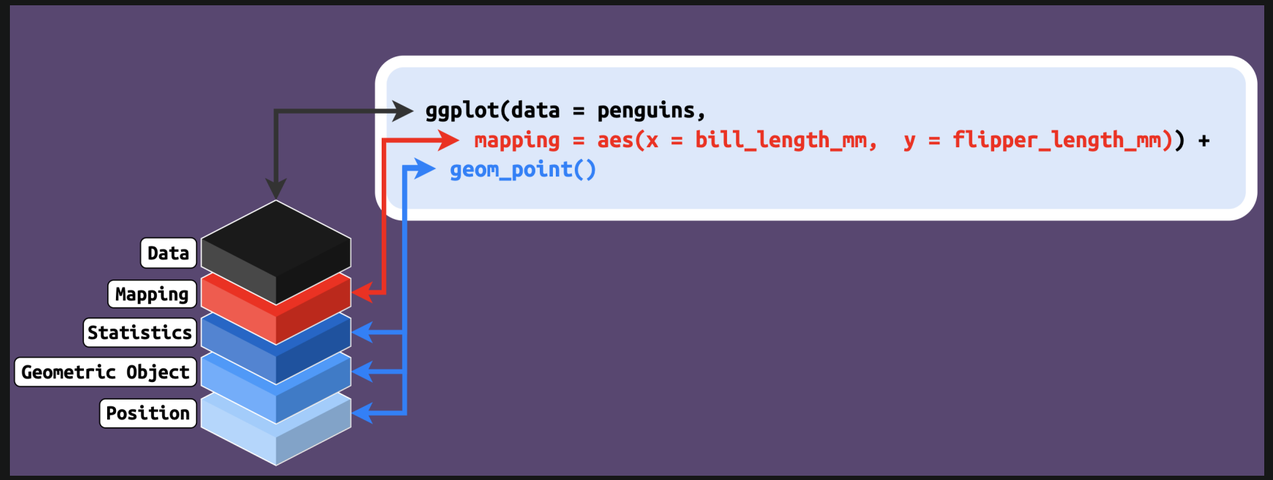
Global mapping

The previous graphs mapped aesthetics globally

Global = aesthetics are mapped when the graph is initialized with `ggplot()`:

```
1  ggplot(data = penguins,
2    mapping = aes(
3      x = bill_length_mm,
4      y = flipper_length_mm))
```

Recall the layers from Part 1:

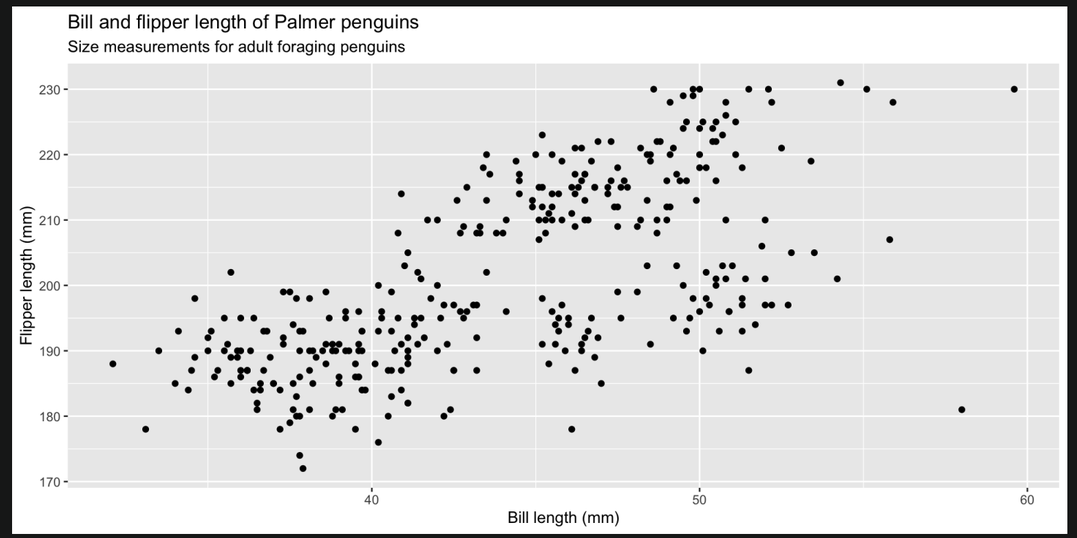


If we map aesthetics in `ggplot()`, all the following `geom_*()` layers will inherit these aesthetics

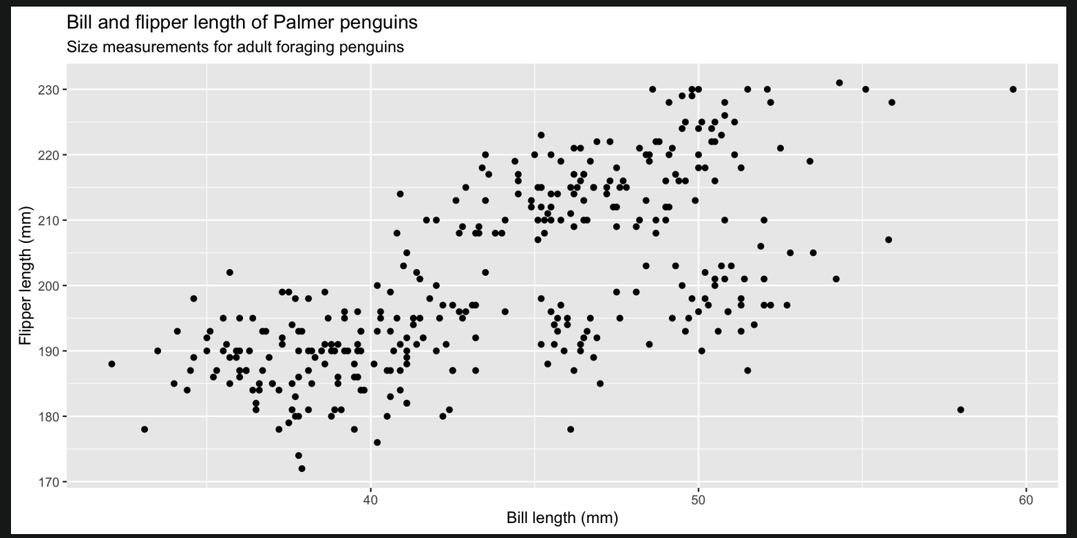
Local mapping

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

```
1  ggplot(data = penguins,
2    mapping = aes(
3      x = bill_length_mm,
4      y = flipper_length_mm)) +
5    geom_point() +
6      labs_bill_vs_flippper
```




```
1  ggplot(data = penguins) +
2    geom_point(mapping =
3        aes(x = bill_length_mm,
4            y = flipper_length_mm)) +
5    labs_bill_vs_flippper
```



The ggplot2 templates (refresher)

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>))
```

Below we've adjusted the 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_flippper
```

Local

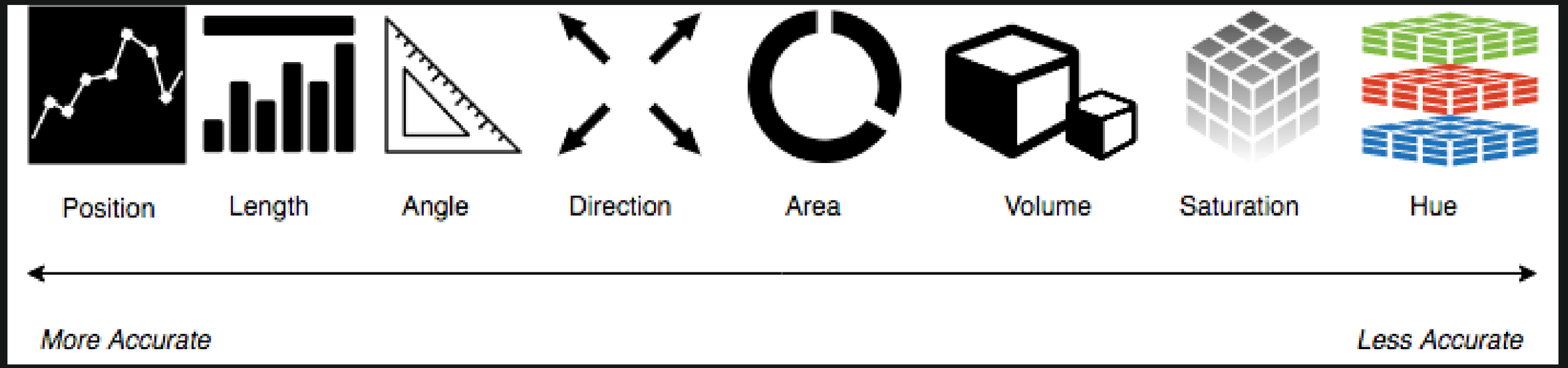
```
ggplot(data = penguins) +
geom_point(mapping =
    aes(x = bill_length_mm,
        y = flipper_length_mm)) +
      labs_bill_vs_flippper
```

Visual encodings

What are visual encodings?

Visual encodings are what we see on the graph

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



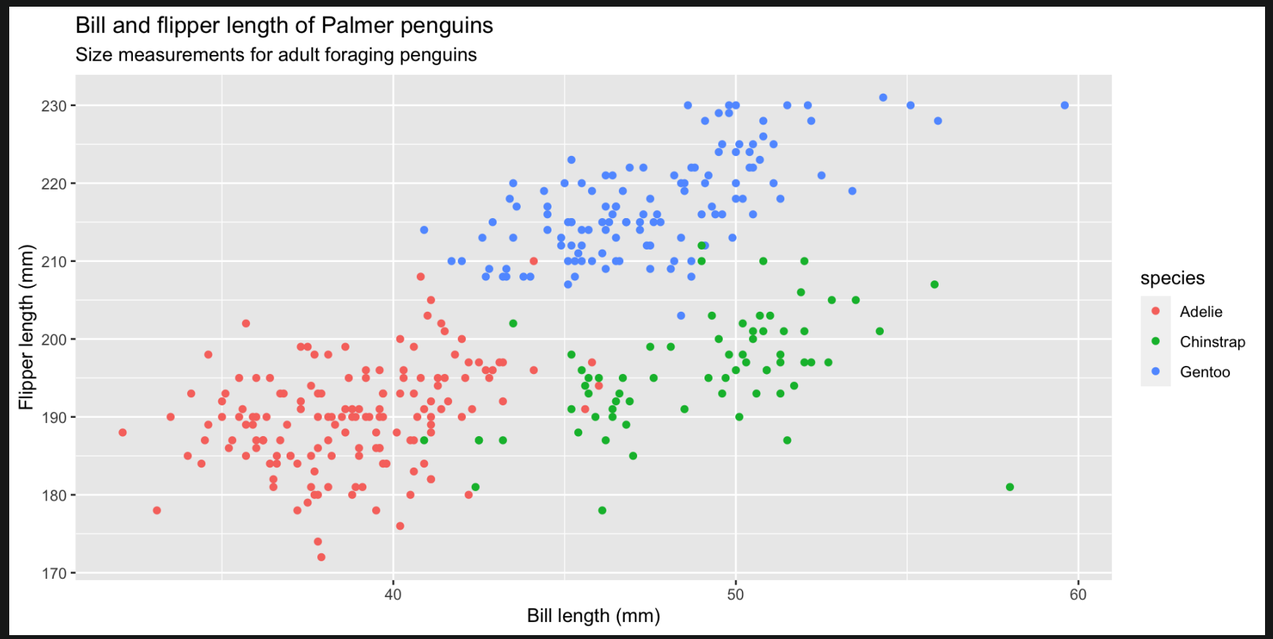
Ranked by accuracy

graph 06: Adding color (global)

Map `color` to the `species` variable using **global** aesthetic mapping:

Inside the `aes()` function:

```
1  ggplot(data = penguins,
2    mapping =
3      aes(x = bill_length_mm,
4          y = flipper_length_mm,
5          color = species)) +
6    geom_point() +
7    labs_bill_vs_flippper
```



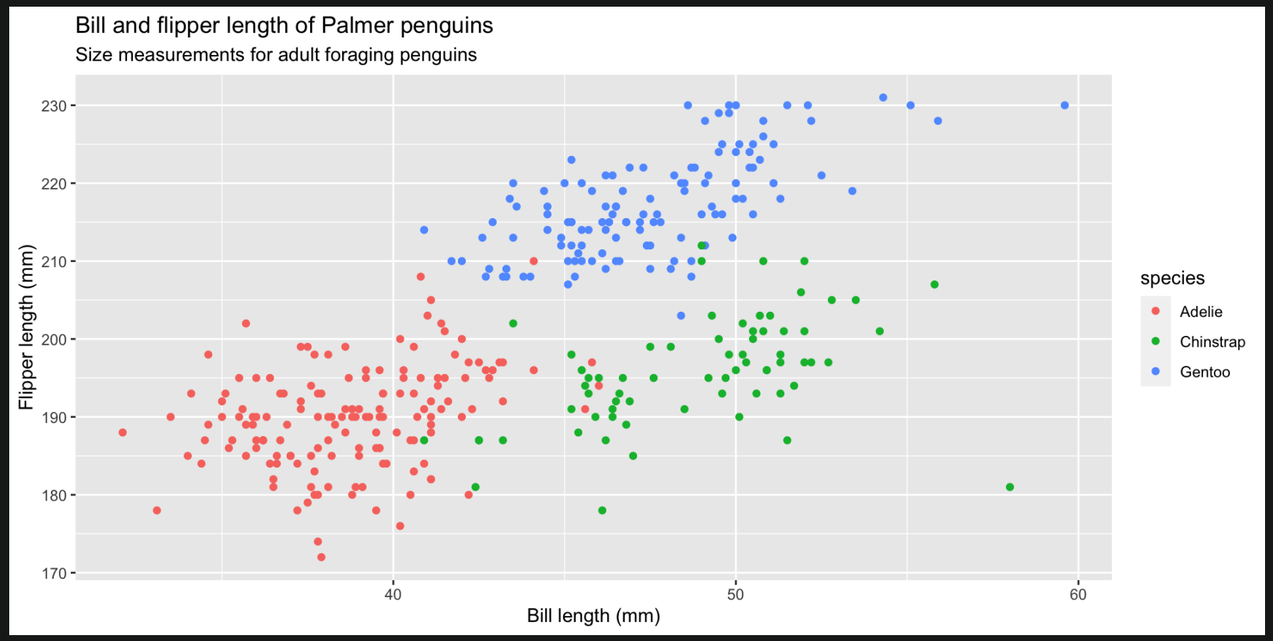
`ggplot2` includes a legend by default

graph 07: Adding color (local)

Map `color` to the `species` variable using ***local*** aesthetic mapping

The `x` and `y` aesthetics are inherited from the `ggplot()` function...

```
1  ggplot(data = penguins,
2    mapping =
3      aes(x = bill_length_mm,
4          y = flipper_length_mm)) +
5    geom_point(
6      aes(color = species)) +
7    labs_bill_vs_flippper
```



...but the `color` aesthetic comes from the `geom_point()` layer

graph 08: Color vs. Fill (1 of 2)

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

First 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)
```

View our data:

```
glimpse(penguins_no_miss, 50)
```

```
Rows: 333
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie…
$ island            <fct> Torgersen, Torgersen, …
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, 36.7…
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, 19.3…
$ flipper_length_mm <int> 181, 186, 195, 193, 19…
$ body_mass_g       <int> 3750, 3800, 3250, 3450…
$ sex               <fct> male, female, female, …
$ year              <int> 2007, 2007, 2007, 2007…
```

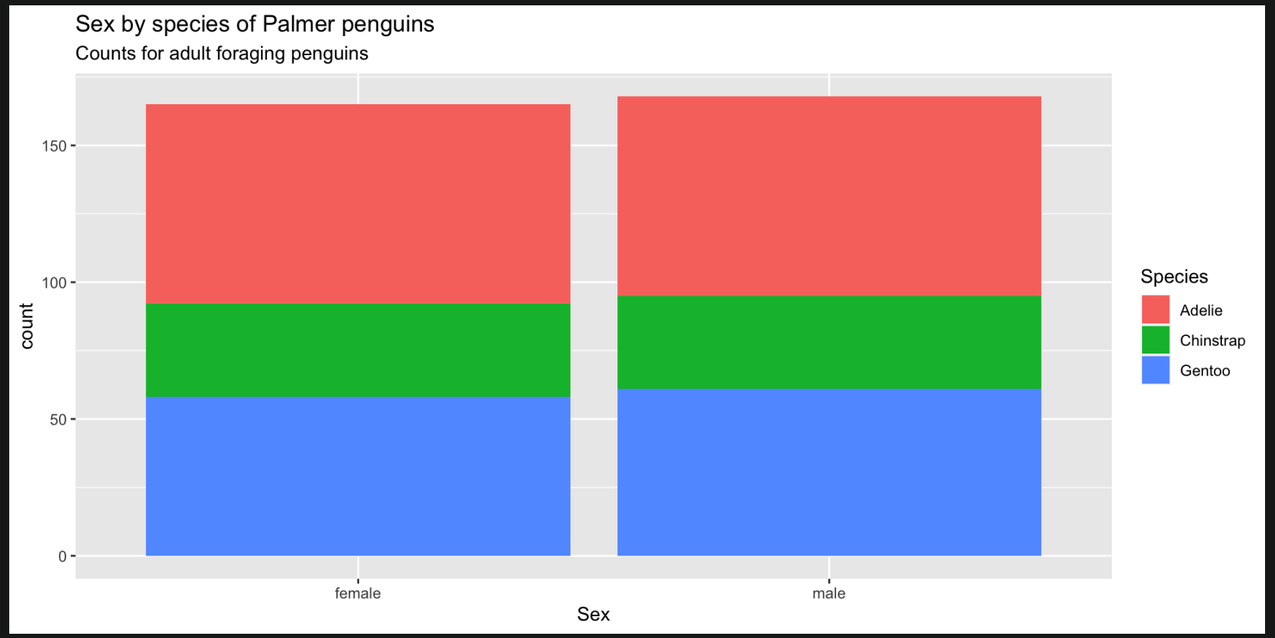
graph 08: Color vs. Fill (2 of 2)

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

Build a bar-graph using `geom_bar()` by locally mapping `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!

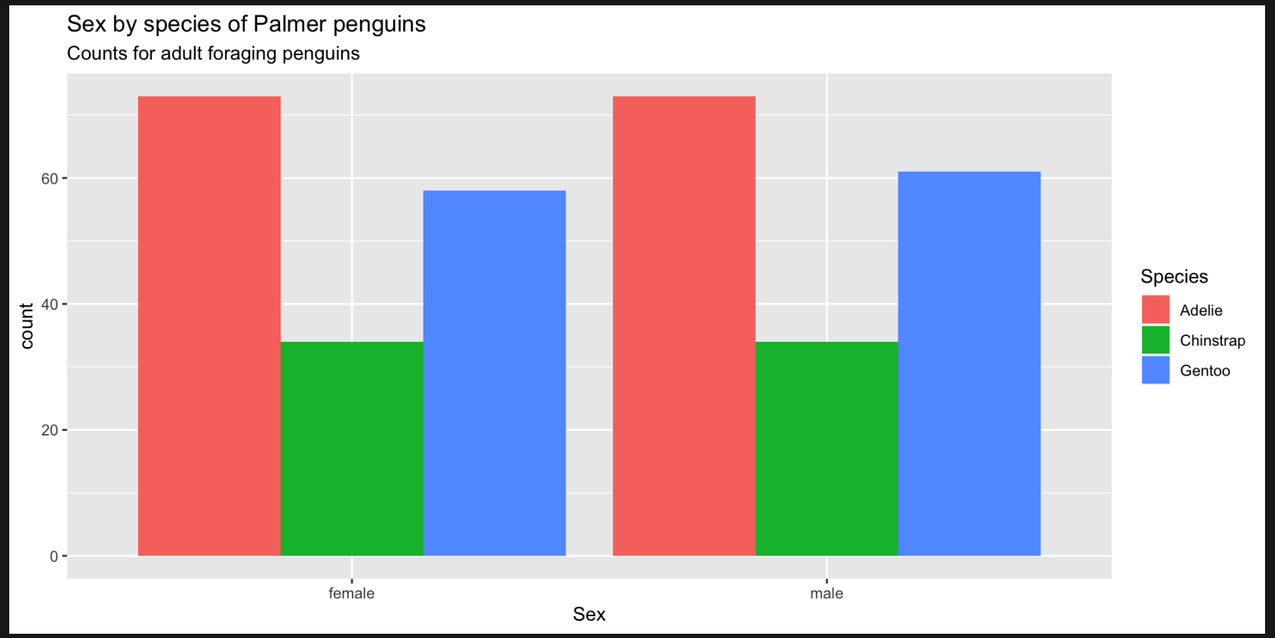


graph 09: Bar position

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

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



graph 10: Histograms (special bar-graphs)

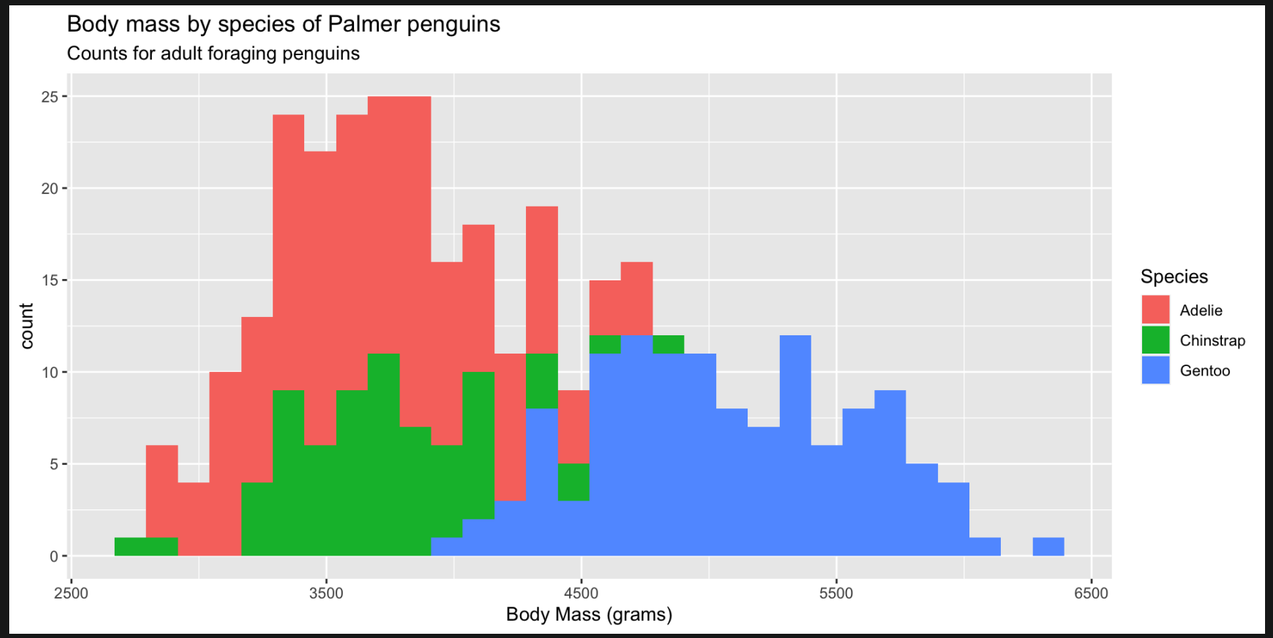
The `geom_histogram()` function uses ‘bins’ to represent counts for each value

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
```



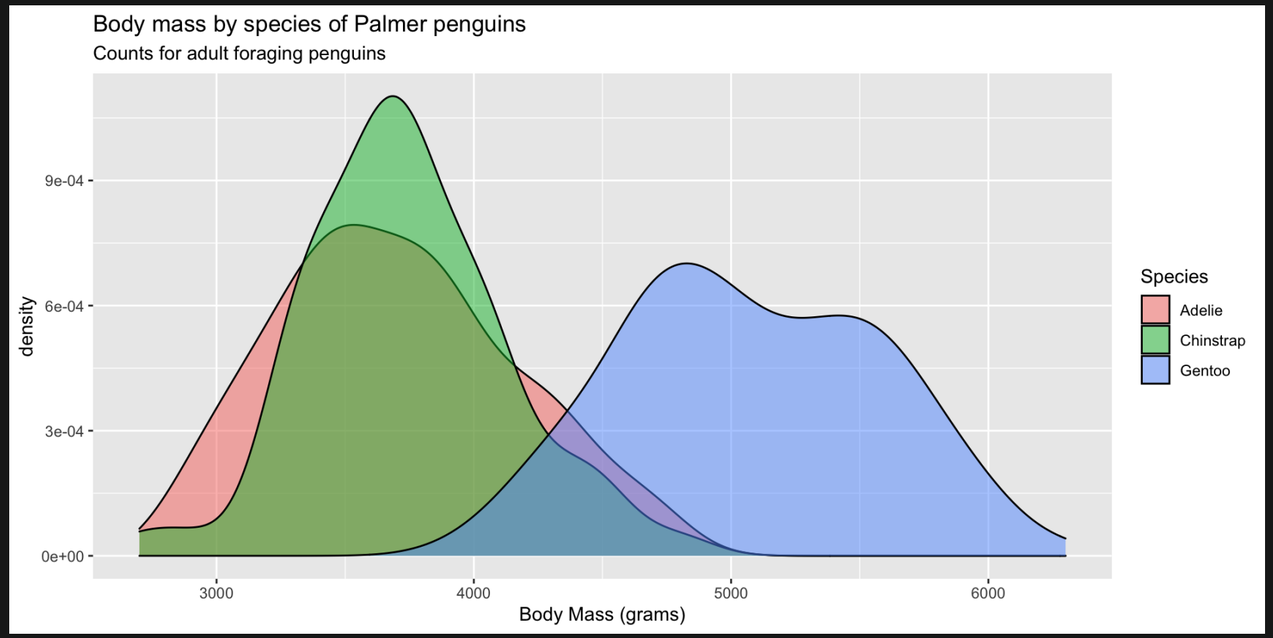
graph 11: Density plots

Density plots are also great for comparing overlapping distributions

Create a density plot with `geom_density()`

Set the `alpha` (color saturation) to `1/2`

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



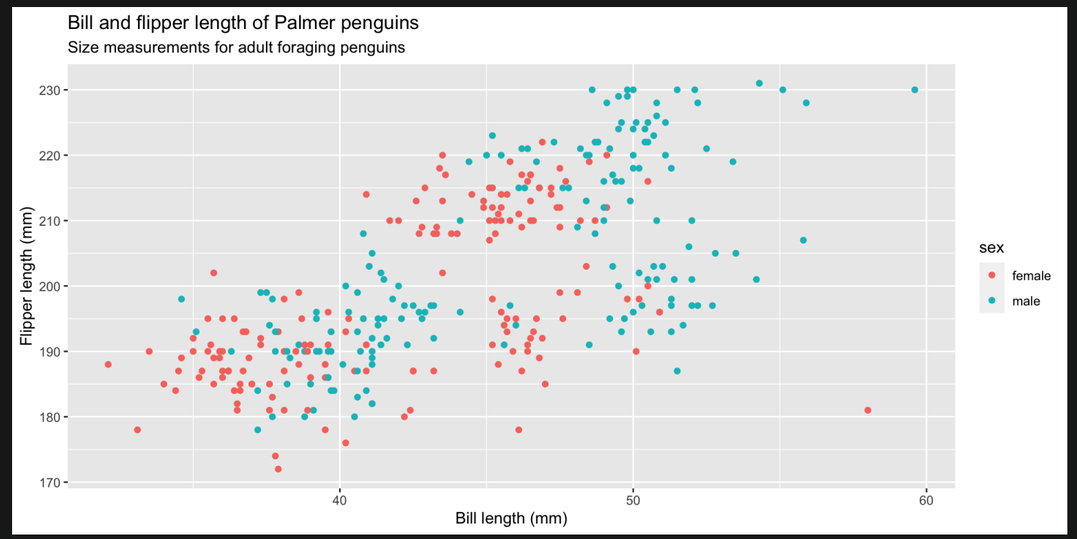
Also check out ridgeline plots

Mapping vs. setting aesthetics

Mapping vs. setting (1 of 2)

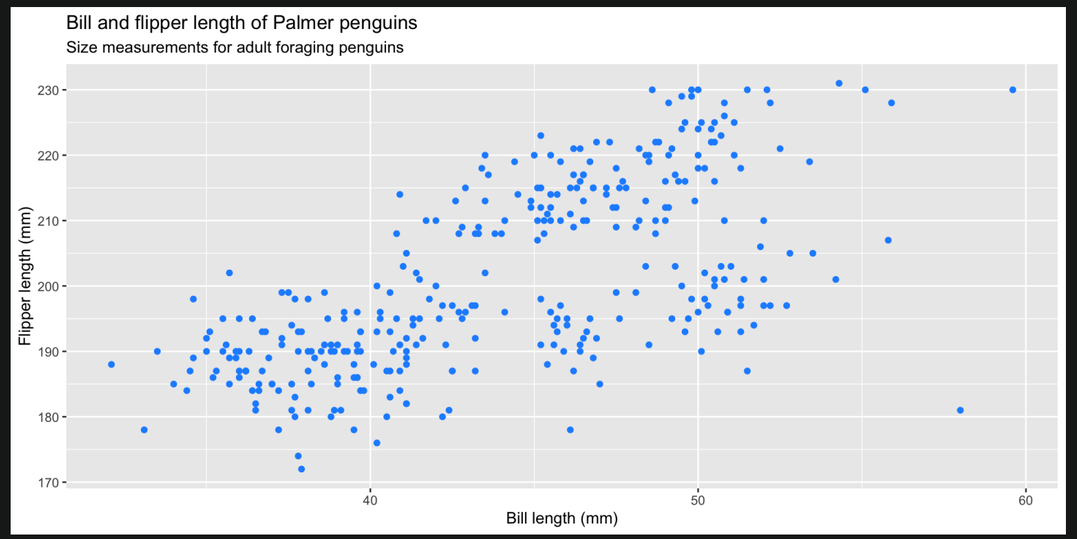
Variables are mapped to aesthetics *inside* `aes()`

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



Values are set outside the `aes()` function

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



Mapping vs. setting (2 of 2)

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

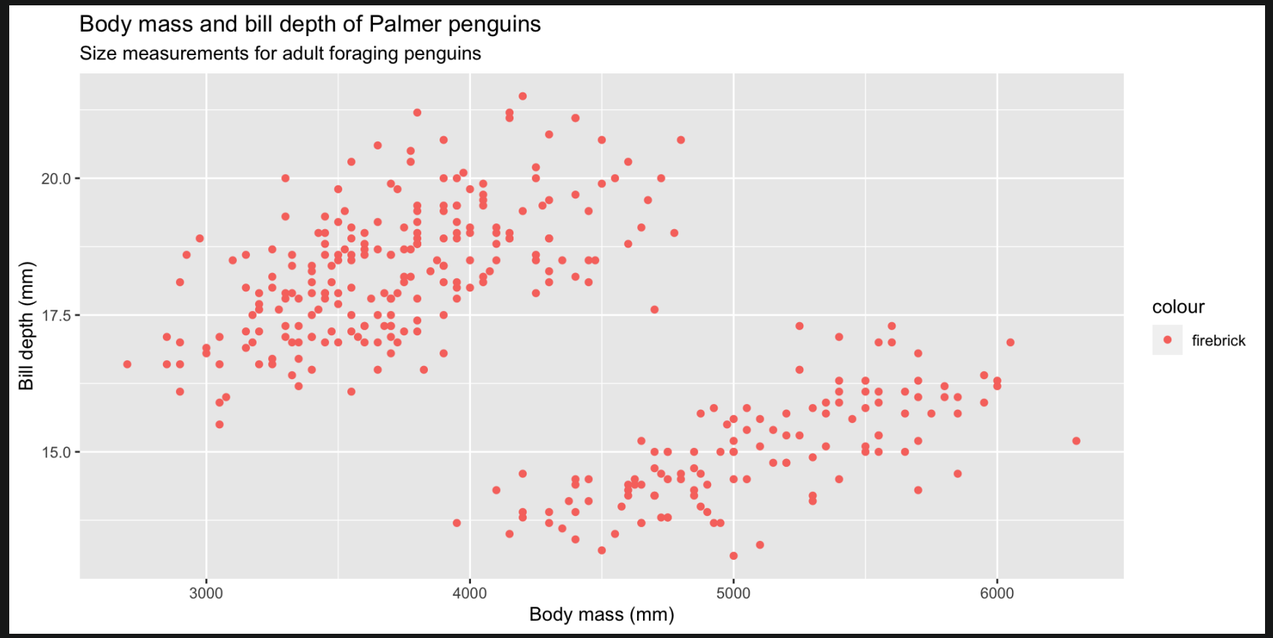
Change the code below to make the points "firebrick" red

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 dataset of the max bill length and depth, body mass and flipper length (`big_penguins`):

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

Create data `label` and `source`

```
big_penguins <- mutate(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

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

label	source
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

graph 13: Layer 1

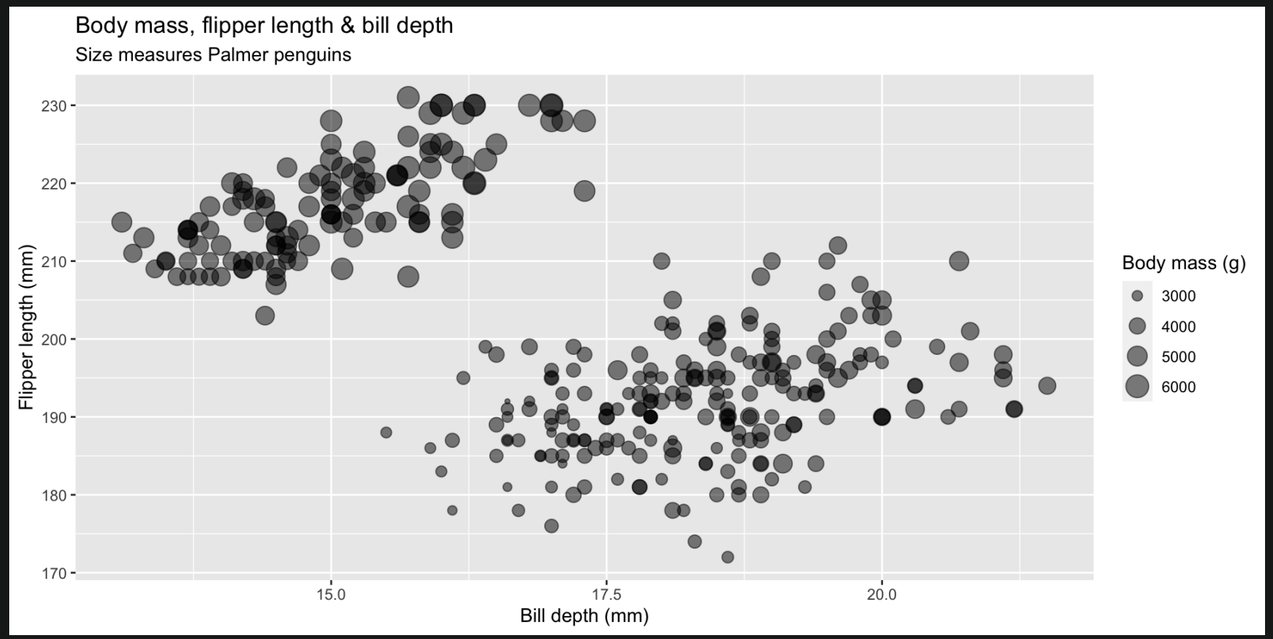
Create layer 1 with penguins_no_miss data and geom_point()

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)")
```

Assign x, y, size, and alpha

```
ggp_13 <- 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)
ggp_13 +
    # labels
    labs_bodymass_bill_depth_flipper_length
```

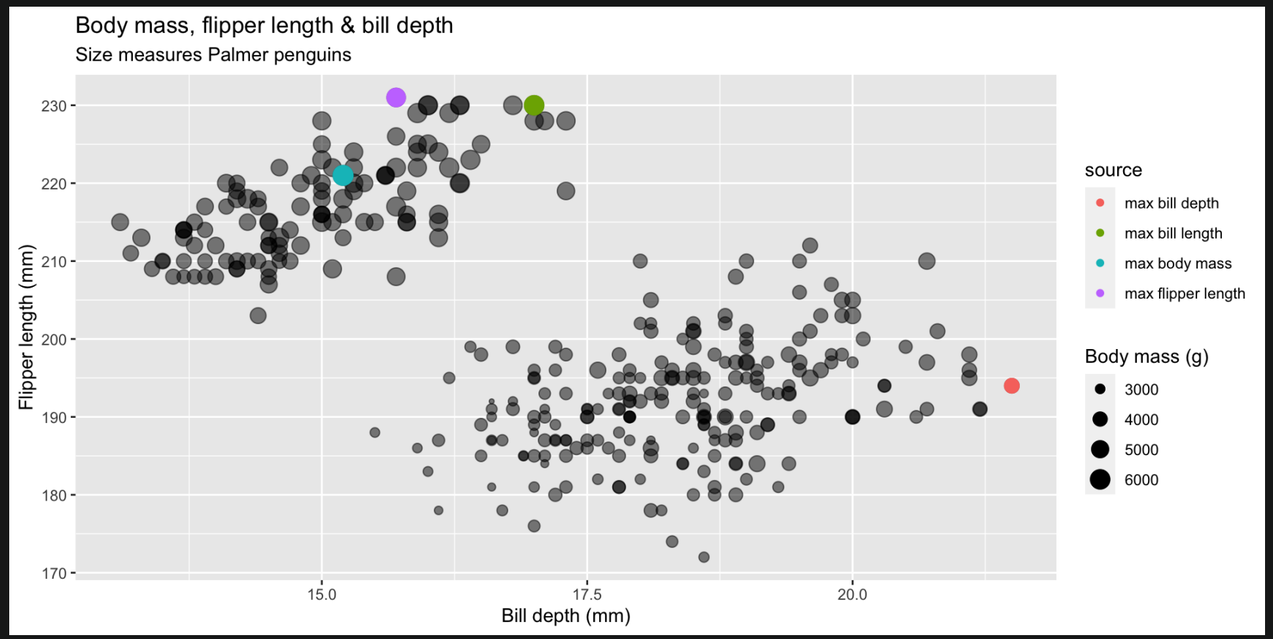


graph 14: Layer 2

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

Use `scale_size()` to adjust point scaling

```
ggp14 <- ggp_13 +
  # 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))
ggp14 +
  # labels
  labs_bodymass_bill_depth_flipper_length
```

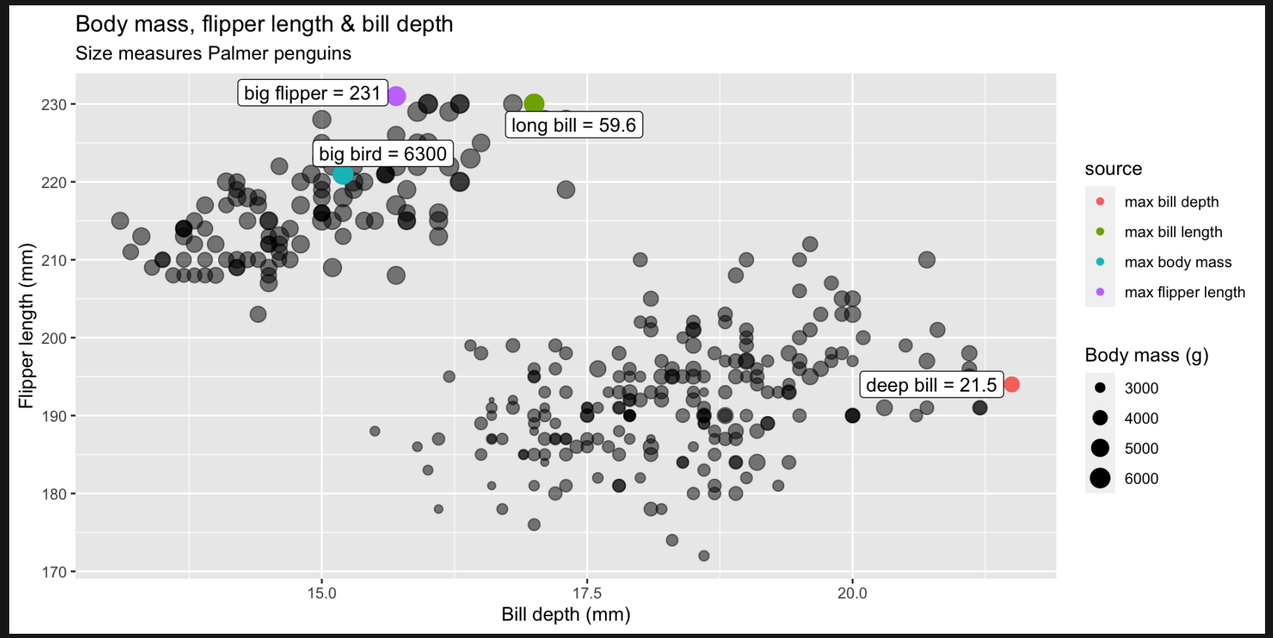


graph 15: Label 3 (max values)

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

Add layer for `labels` in `big_penguins`

```
 1  library(ggrepel)
 2  ggp15 <- ggp14 +
 3    # layer 3
 4    ggrepel::geom_label_repel(
 5      data = big_penguins,
 6      mapping = aes(x = bill_depth_mm,
 7        y = flipper_length_mm,
 8        label = label))
 9  ggp15 +
10    # labels
11    labs_bodymass_bill_depth_flipper_length
```



Facets

Small multiples

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

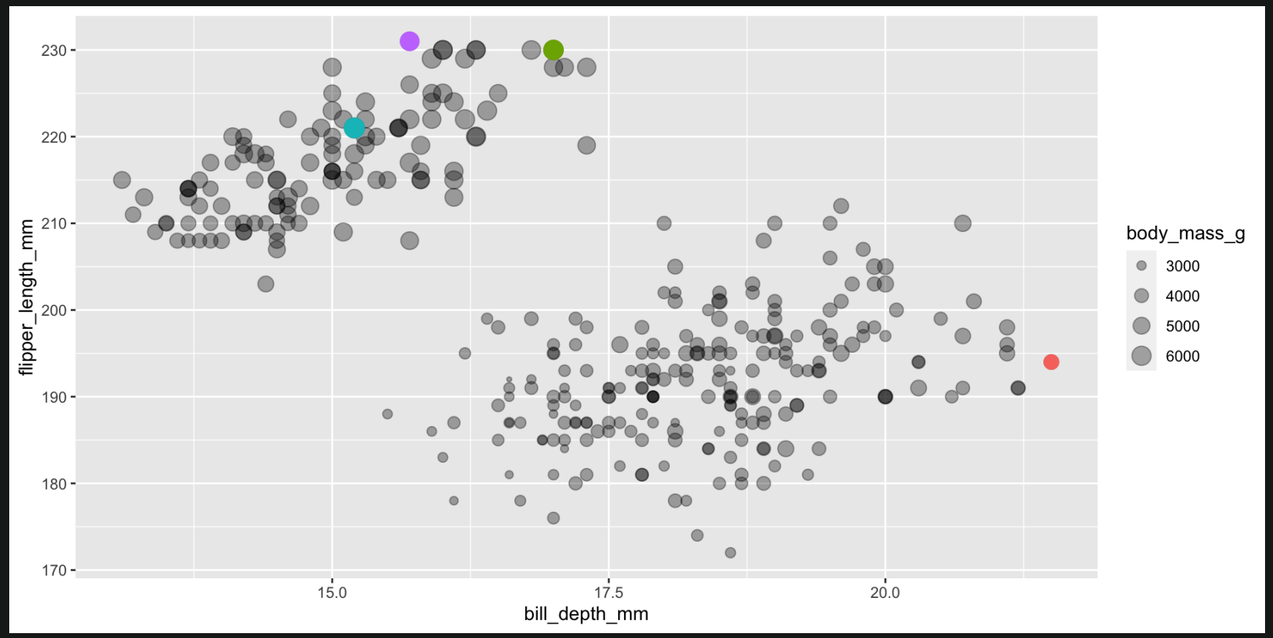
In the previous graph, we used multiple aesthetics (color, size, shape)

Can we explore these relationships by sex or species?

Store graph 15 in `ggp_penguin_measures`

```
ggp15_l1 <- ggplot(data = penguins_no_miss) +
  geom_point(
    mapping = aes(x = bill_depth_mm,
      y = flipper_length_mm,
      size = body_mass_g),
    alpha = 1 / 3)
```

```
ggp_penguin_measures <- ggp15_l1 +
  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))
```

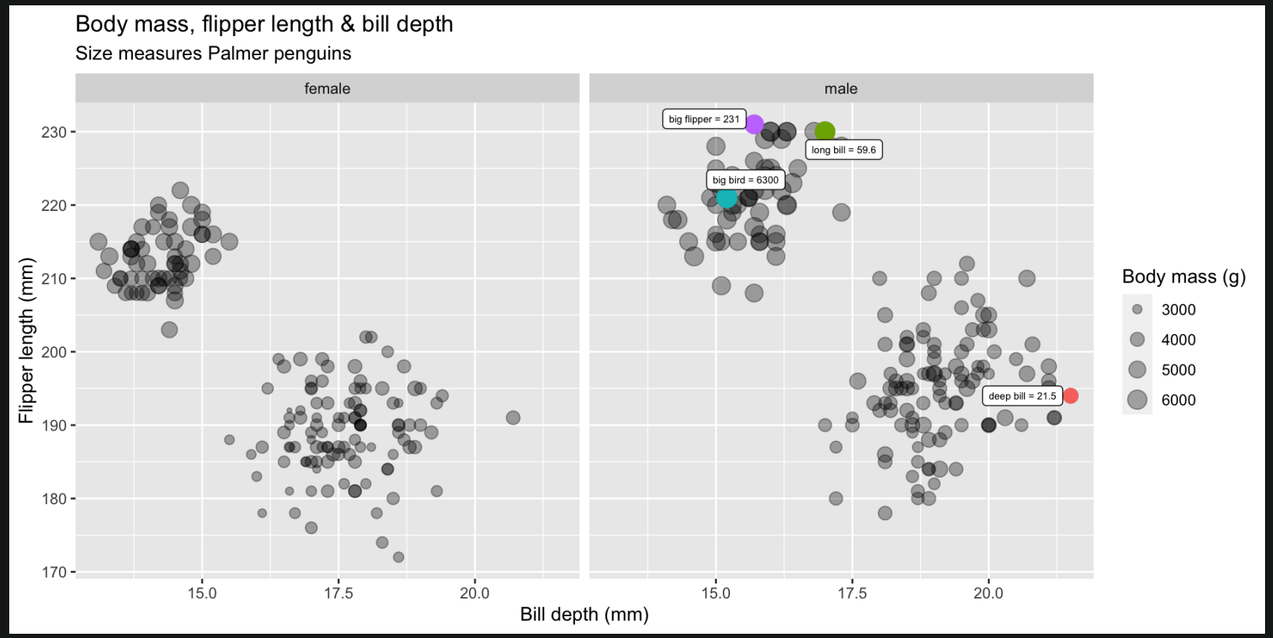


graph 16: Facet by sex

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),
    size = 2) + # adjust size
  facet_wrap(. ~ sex) + # facet by sex
  # labels
  labs_bodymass_bill_depth_flipper_length
```

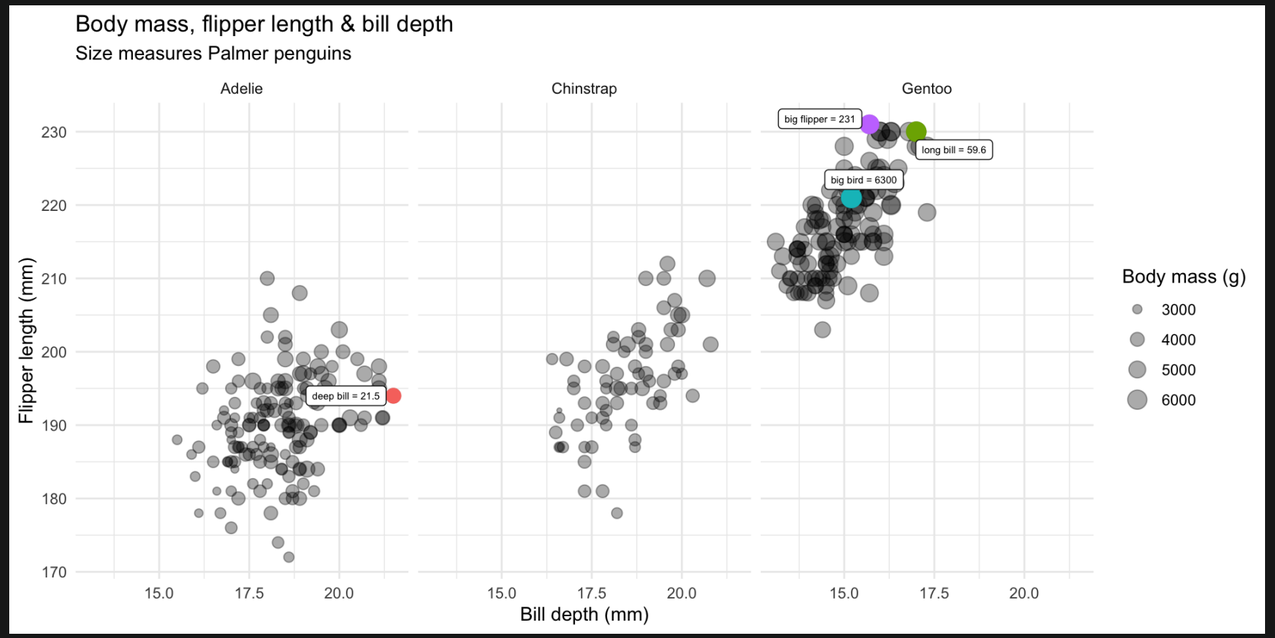


graph 17: Facet by species

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

Change `facet_wrap()` to `~ species`
Add `theme_minimal()` and labels

```
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*, *geoms*)
4. Mapping variables to graph elements (*color*, *position*, *size*, *etc.*)
5. Extending graphs by combining layers
6. Using facets to explore relationships

Thanks!

Twitter @mjfrigaard

GitHub @mjfrigaard

Email @mjfrigaard