Statistical basics and data visualization

ANTH 572S

Laure Spake

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Brief review of "last" week

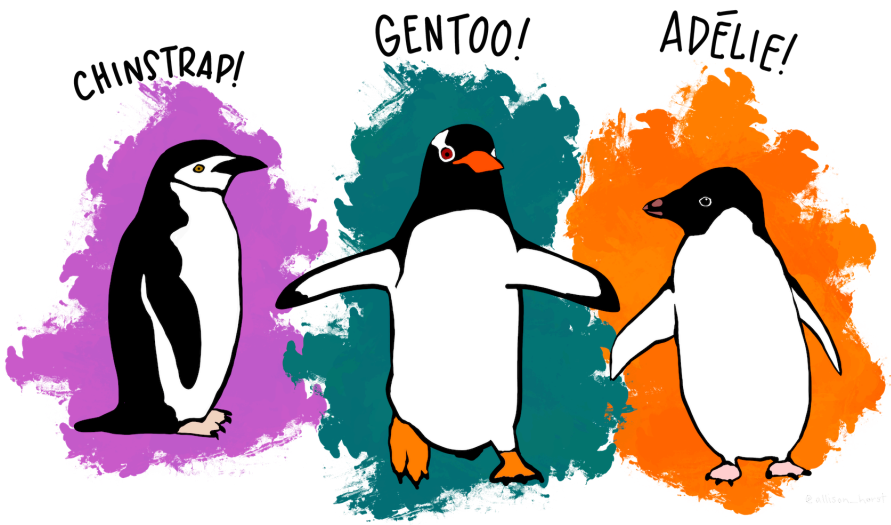
- You learned how to interact with R and RStudio
- You learned about vectors

Learning objectives

At the end of this lesson you will:

- Be able to use terminology that describes data and its components
- Explain why data visualization is important
- Produce data visualizations
- Produce summary statistics

Penguins from `palmerpenguins`



The dataset used to illustrate concepts today and through much of the course comes from the `palmerpenguins` library[1], and records observations about penguins made from three islands in the Palmer archipelago in Antarctica.

What does data look like?

A *dataframe* is a data structure that shapes data into a 2 dimensional table of rows and columns

The top row usually contains the names of the variables, and the data itself starts on the following row

```
# A tibble: 6 × 9
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
4 Adelie  Torgersen           NA            NA                  NA          NA
5 Adelie  Torgersen           36.7          19.3               193        3450
6 Adelie  Torgersen           39.3          20.6               190        3650
# ℹ 3 more variables: sex <fct>, year <int>, size <chr>
```

Terminology to describe data

Within a dataframe

Variable: A quantity, quality, or property that can be measured (typically columns)

Observation: A set of measurements made under similar conditions. Contains several values, each associated with a variable.

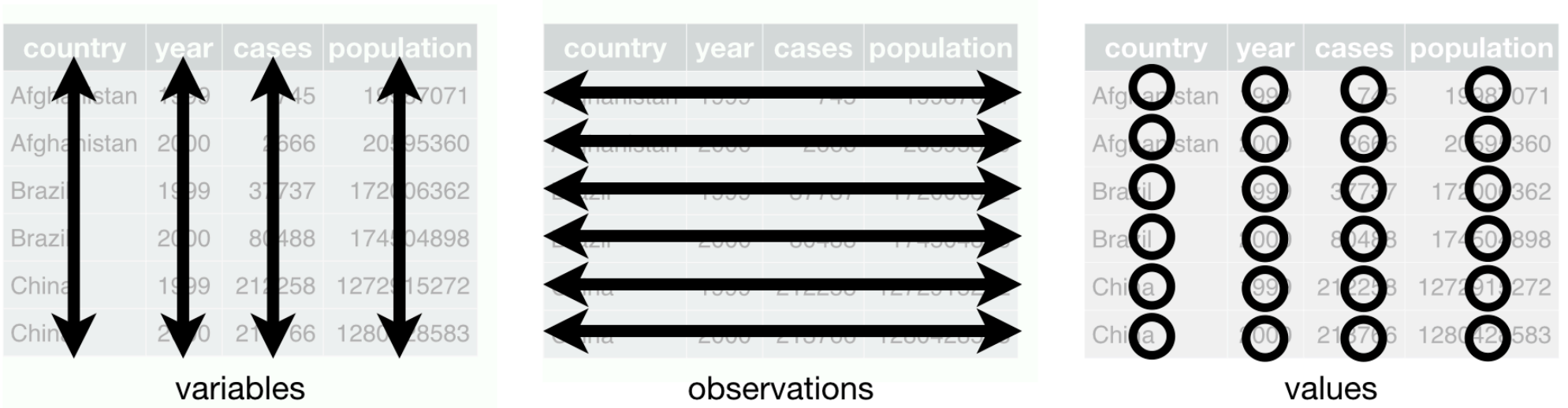
Value: The measurement that is recorded for a particular observation of a variable

Example

In the `penguins` dataframe, what are some of the variables? Where would you find an observation? What is one example of a value?

```
# A tibble: 4 × 9
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
4 Adelie  Torgersen           NA            NA                  NA          NA
# ℹ 3 more variables: sex <fct>, year <int>, size <chr>
```

The concept of tidy data



1. Each variable is a column
2. Each observation is a row
3. Each type of observational unit is a table/dataframe

Why you should record data in tidy formats

- Easy to share with others
- Easy to manipulate
- Easy to visualize and analyze

Data wrangling

Data wrangling is the process of both tidying data and pre-processing it (e.g. error checking, recoding) in order to make the analysis process easier.

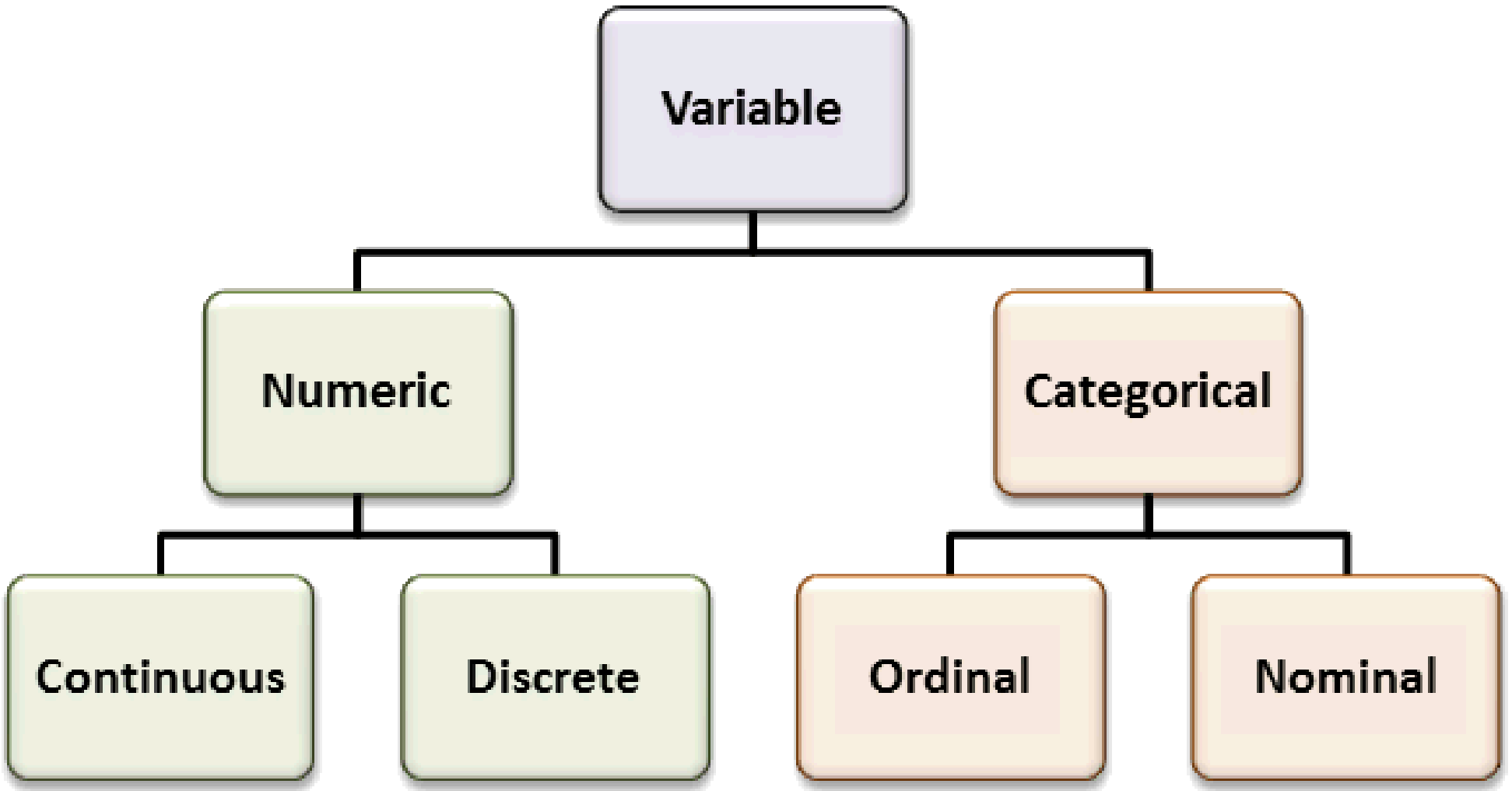
We will spend some time dealing with this topic using `dplyr` next week.

Dealing with variables

A note about understanding variable types

This is probably the most foundational concept in statistics, though it appears to be sometimes hard for students.

Types of variables



Numerical variables

Variables whose values are numbers. They can be either:

- Continuous: a numerical value that can take an infinite number of real values given an interval. For example, height.

- Discrete: a numerical value that can only take a finite number of real values within a given interval. For example, a count of students in this class.

Categorical variables

Variables whose values are characteristics that can't be measured by numbers. They can be either:

- Nominal: a categorical variable that describes a label or category without a natural order. For example, place of birth.

- Ordinal: a categorical variable that describes a label or category where there is a natural order between values. For example, educational achievement, or assessments of quality.

A note on variable types

```r
student <- c("A", "B", "C", "D", "E")
assigned.group <- c(1, 1, 2, 3, 2)

groups <- data.frame(student, assigned.group)
```

Be careful of variables coded with numbers! These can be categorical variables disguising as numerical variables.

Take the following example dataframe:

```r
head(groups)
```

```
  student assigned.group
1       A              1
2       B              1
3       C              2
4       D              3
5       E              2
```

Recognizing different variable types

Thesse variable are from the `penguins` dataframe. Turn to your neighbour and discuss - what type of variable is each of these?

```
# A tibble: 5 × 4
  species   sex    body_mass_g size
  <fct>     <fct>        <int> <chr>
1 Adelie    male          3750 small
2 Adelie    female        3150 x-small
3 Gentoo    female        5100 large
4 Chinstrap male          3725 small
5 Chinstrap female        3675 small
```

Recognizing different variable types

These variables are from the `ToothGrowth` dataframe. This is an experimental dataset that tests the effects of vitamin C supplementation delivered by two mechanisms on tooth length in guinea pigs. What type of variable is each of these?

```
  length delivery_type dose
1   21.5            VC  2.0
2   25.5            VC  2.0
3   26.7            VC  2.0
4   22.5            VC  1.0
5   11.2            VC  0.5
6   23.3            OJ  1.0
```

Some last thoughts on tidy data

Learning how to parse out a dataset is a crucial skill that can be developed over time

Tidy data makes it easier for us to understand the structure of a new dataset.

Untidy data has its uses, but is generally used as a tool after the data collection phase

Working with objects in R

How R works: Functions and arguments

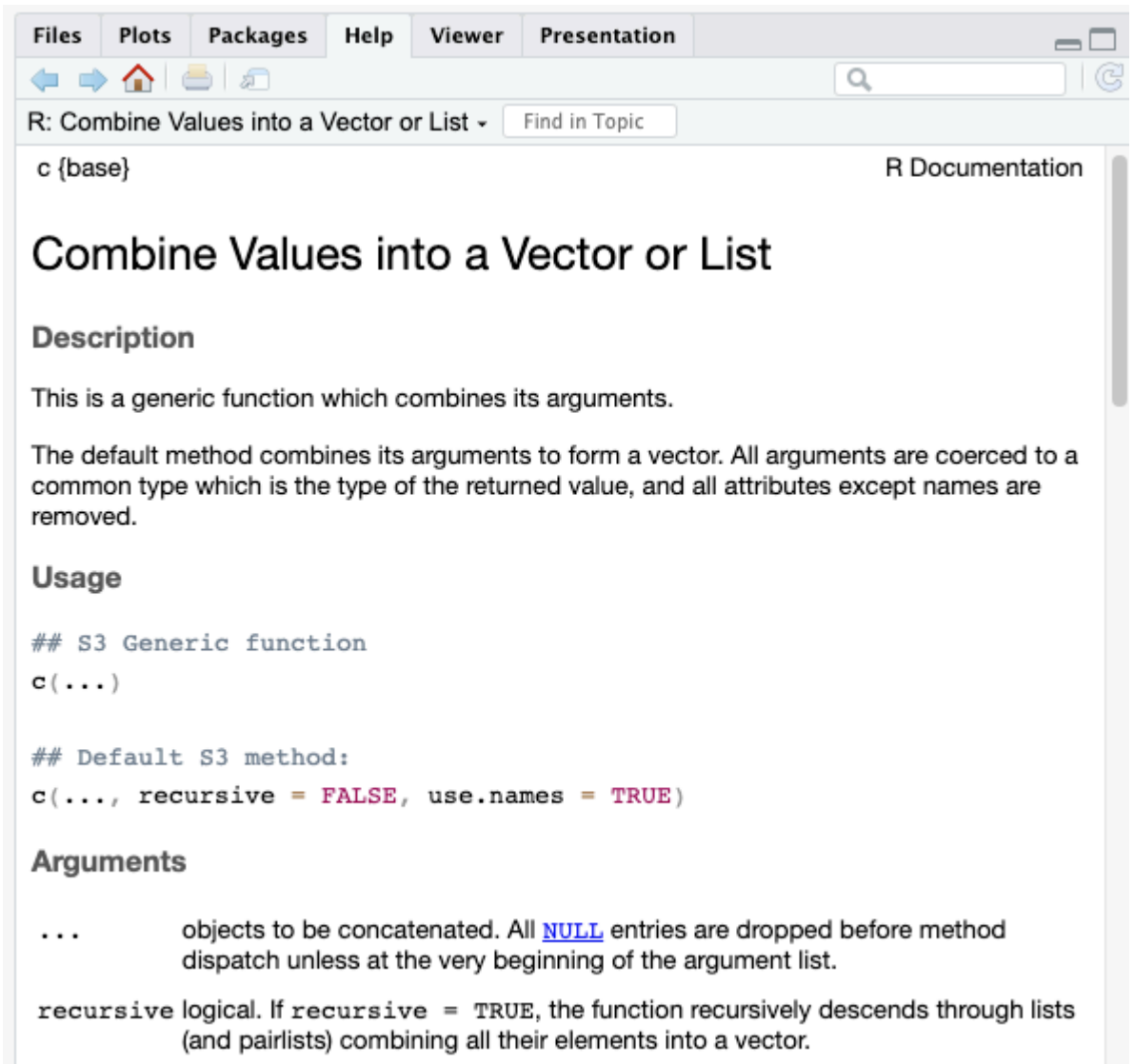
Any operation you perform in R takes place through the use of a function

A function takes the following form:
function(argument1, argument2, argument3)

You can learn about functions and the arguments they required by using the ? in the console, or searching the Help tab of the viewer.

Functions and arguments

When you created vectors previously, you were actually using a function called c ( )



Functions and arguments

Other examples of functions we have used so far: `str()`, `data.frame()`

From now on we will be making extensive use of functions - remember that you can always look up "how they work" with the `?` call.

Types of objects in R

We will be working with three main types of objects in R:

1. Vectors
2. Dataframes
3. Models

Working with objects in R

How do we look at at vectors and learn their types? Let's say we have two vectors called `vec1` and `vec2`. We want to know what type of vectors they are. Let's start by printing them:

```
1 str(vec1)
```

```
 int [1:4] 1 2 3 4
```

```
1 typeof(vec1)
```

```
[1] "integer"
```

```
1 str(vec2)
```

```
 chr [1:4] "A" "B" "C" "D"
```

```
1 typeof(vec2)
```

```
[1] "character"
```

Working with objects in R

What type of vectors are they? We can use either the `typeof()` or `str()` functions to tell us:

```
1 str(vec1)
```

```
 int [1:4] 1 2 3 4
```

```
1 str(vec2)
```

```
 chr [1:4] "A" "B" "C" "D"
```

```
1 typeof(vec1)
```

```
[1] "integer"
```

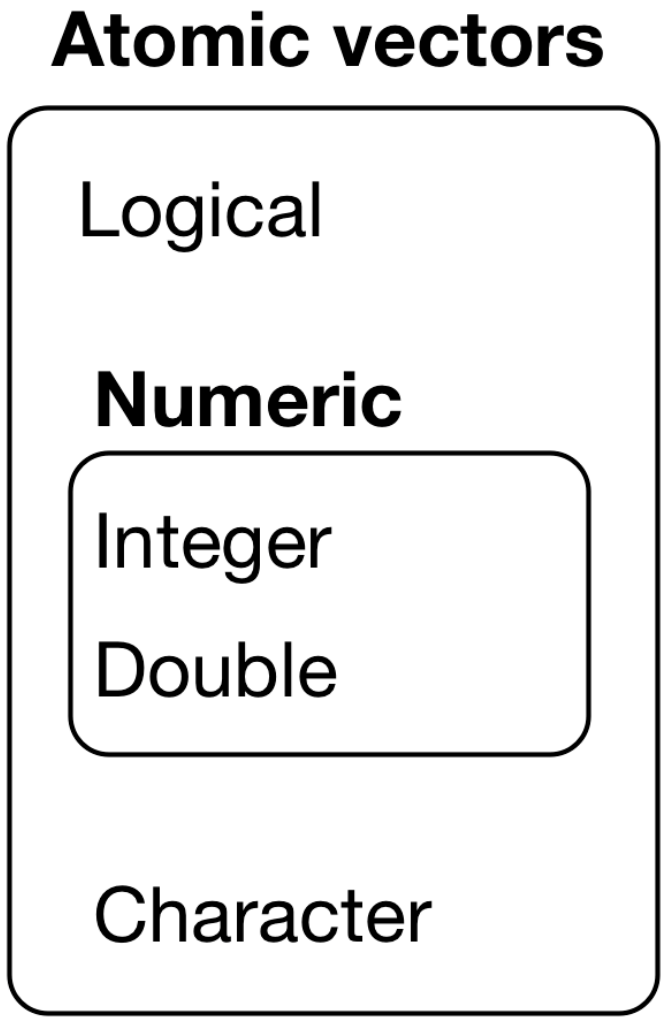
```
1 typeof(vec2)
```

```
[1] "character"
```

Converting between types of vectors

R handles 4 main types of vectors:

1. logical - `TRUE, FALSE`
2. integer - whole numbers
3. double/numeric - decimal numbers
4. character - most flexible type, alphanumeric



Converting between vector types

There are several functions for converting between types of vectors:

- `as.logical()`, can only take vectors of value 0, 1
- `as.integer()`, can only convert vectors and factors with whole number values
- `as.numeric()`, can take integer variables or logical vectors
- `as.character()`, can take vectors logical, integer, or double

Converting between variable types

`vec1` is of class integer. Let's say, however, that this variables really coding for some attribute that is measured in inches, and we want to be able to enter data that measured in half inch increments.

We could convert the vector to a numeric variable with the `as.numeric()` function

```
1 vec1numeric <- as.numeric(vec1)
```

Converting between variable types

In practice, R is pretty clever and will convert vectors when it needs to, so long as the vectors can be coerced to the new type. This is called implicit coercion.

For example:

```
1 typeof(vec1)
```

```
[1] "integer"
```

```
1 vec3 <- vec1 + 1.5
2 typeof(vec3)
```

```
[1] "double"
```

Converting between variable types

`vec1` is of class integer. Let's say, however, that this vector actually codes for five different study groups. What type of variable would this be?

We could convert it to a character vector:

```
1  vec1char <- as.character(vec1)
2  str(vec1char)
```

```
chr [1:4] "1" "2" "3" "4"
```

A note on categorical vectors and factors

Ordinal vectors, and categorical factors broadly, sometimes need to be handled as factors. Factors are coded as integers, with corresponding "levels" coding for the name of the category.

```
1 vec1factor <- as.factor(vec1)
2 str(vec1factor)
```

```
Factor w/ 4 levels "1","2","3","4": 1 2 3 4
```

A note on categorical vectors and factors

Factors are a bit painful to deal with, so we will avoid spending too much time working with them unless absolutely necessary.

In practice, many functions will implicitly coerce character vectors into factors when needed.

However, you should know what a factor is in case you get an error that mentions factors!

Dataframes

Dataframes are a series of vectors of the same length, bound together to create a two dimensional object. Each vector within the dataframe has a name.

A dataframe is similar to a standard Excel spreadsheet

Building a dataframe

Let's say we wanted to combine our two vectors into a dataframe. We can do this with the `data.frame()` function. Let's also add a third column to our dataframe, and call it "df"

```
1  df <- data.frame(column1 = vec1,
2                   column2 = vec2,
3                   column3 = c("blue", "green", "red", "yellow"))
```

```
  column1 column2 column3
1       1       A    blue
2       2       B   green
3       3       C     red
4       4       D  yellow
```

Learning about a dataframe

You often will want to quickly learn the structure of a dataframe. To do this, there are a few different options, e.g.:

```
1 str(df)
```

```
'data.frame':   4 obs. of  3 variables:
 $ column1: int  1 2 3 4
 $ column2: chr  "A" "B" "C" "D"
 $ column3: chr  "blue" "green" "red" "yellow"
```

```
1 head(df)
```

```
  column1 column2 column3
1       1       A    blue
2       2       B   green
3       3       C     red
4       4       D  yellow
```

Working with vectors within a dataframe

You can refer to individual vectors within a dataframe using this standard syntax: `df$column.name`.

```
1 df$column1
```

```
[1] 1 2 3 4
```

```
1 df$column1 + 1.5
```

```
[1] 2.5 3.5 4.5 5.5
```

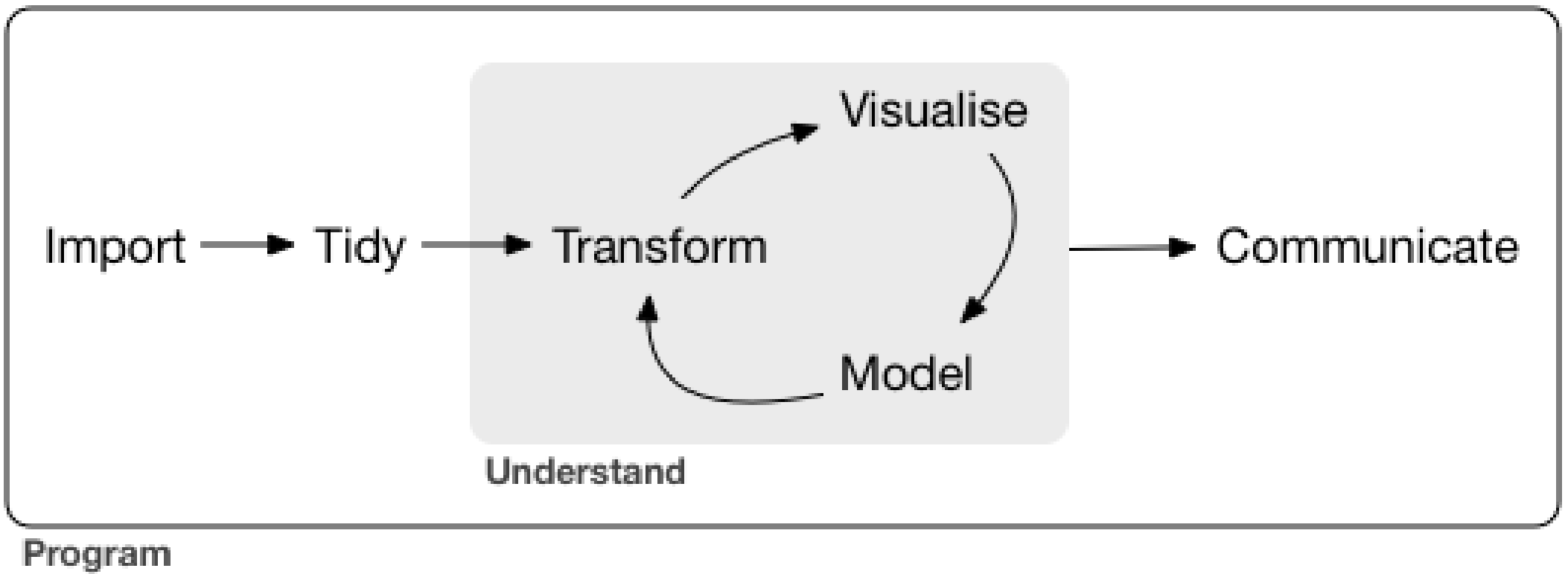
```
1 df$column3 <- df$column1 + 1.5
2 df
```

```
  column1 column2 column3
1       1       A     2.5
2       2       B     3.5
3       3       C     4.5
4       4       D     5.5
```

Your turn: Exercise 2 in Posit Cloud

Data visualization with `ggplot2`

The data science pipeline



The data science pipeline, Wickham and Grolemund (R4DS)

Why visualize data?

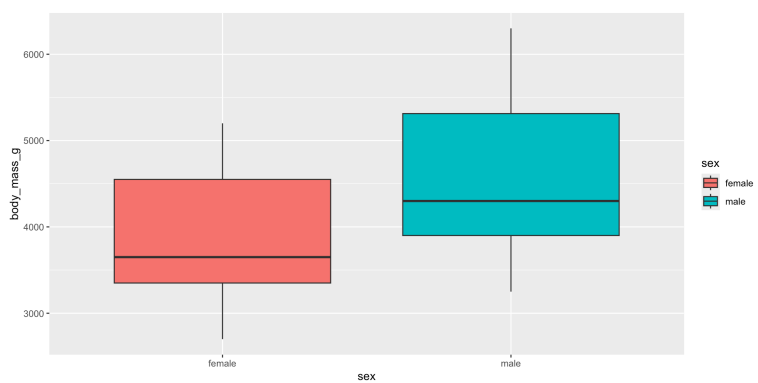
1. To better understand the shape of your data
2. To check and confirm that your tests are reliable
3. To communicate findings

Visualization to understand

Even people who are highly comfortable with numbers can be misled by summary statistics such as means, standard deviations, etc.

Distribution of body size of penguins between females and males

```
   Min. 1st Qu.  Median    Mean 3rd Qu.
Max.    NA's
   2700    3350    3650    3862    4550
5200      11
   Min. 1st Qu.  Median    Mean 3rd Qu.
Max.    NA's
   3250    3900    4300    4546    5312
6300      11
```



Anscombe's quartet

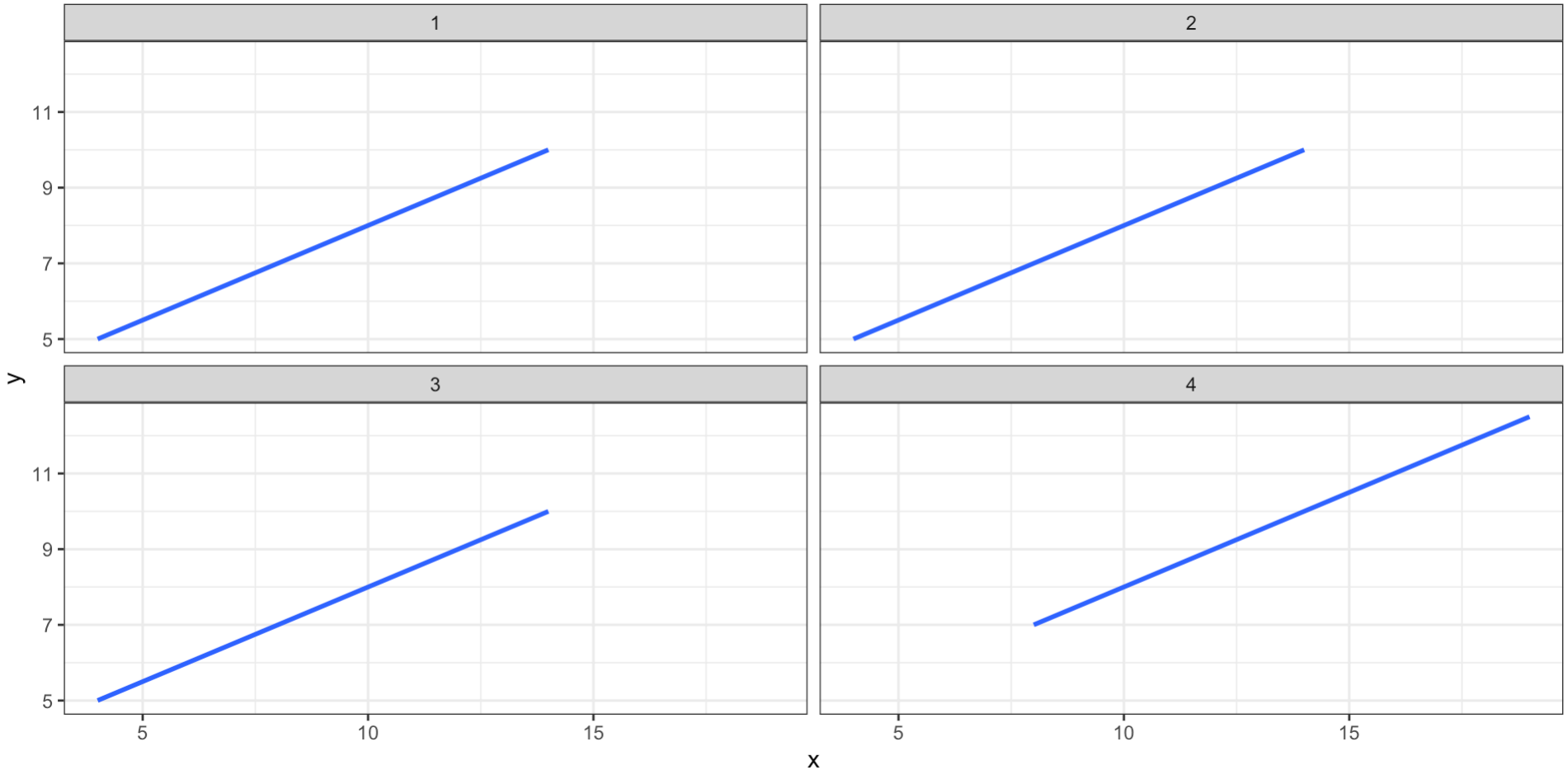
Anscombe's quartet is a famous illustration of why data visualization is so key when we are trying to understand our data or confirm our models.

The quartet consists of four sets of data that when modeled with linear regression yield the same slope and the same R2 (a metric that quantifies goodness of fit).

Anscombe's quartet

The four similar linear regressions can be plotted:

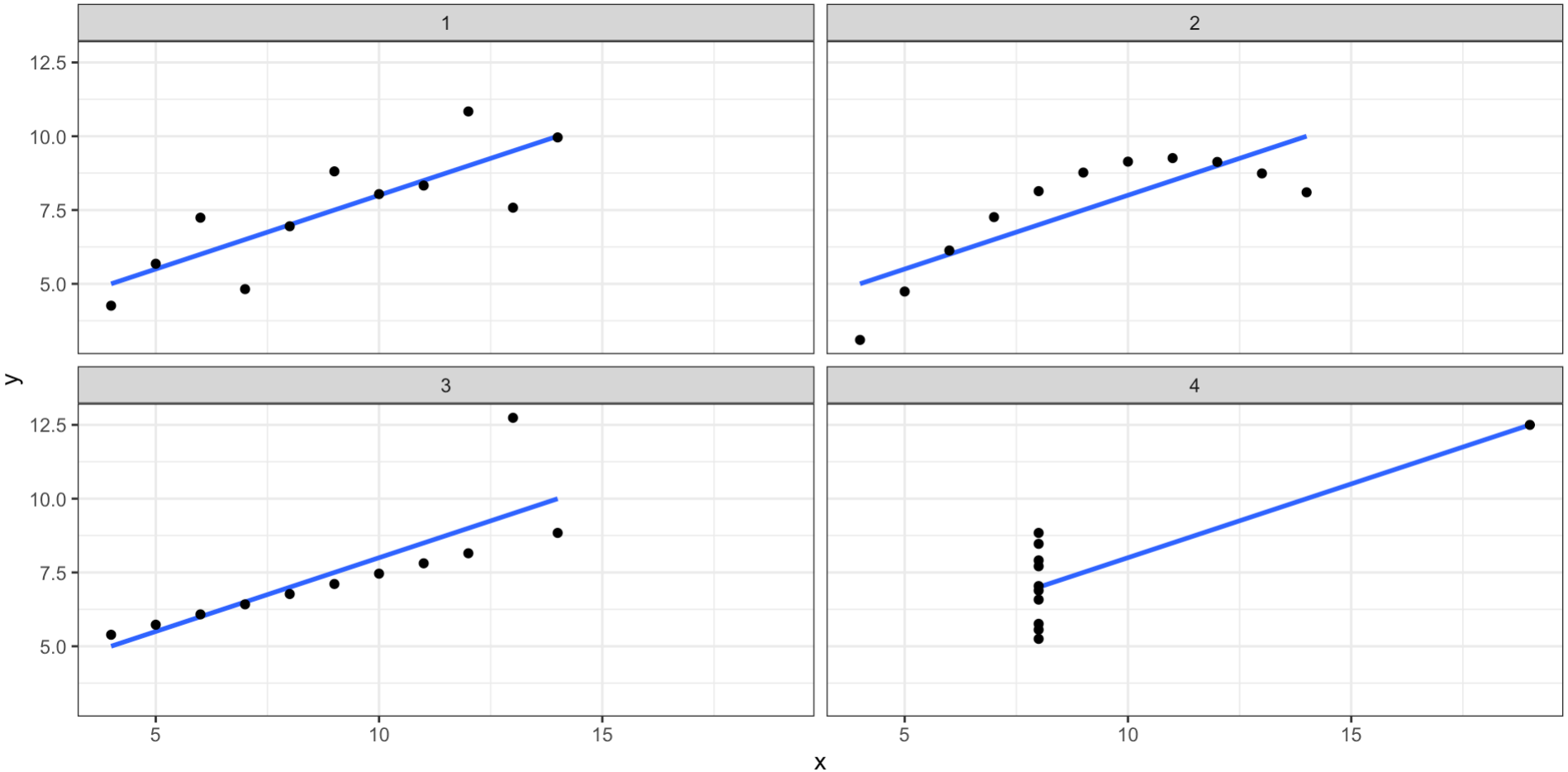
```
ggplot(anscombe_long, aes(x = x, y = y))+
  geom_smooth(method = "lm",  se = FALSE)+
  facet_wrap(set~.)+ theme_bw()
```



Anscombe's quartet

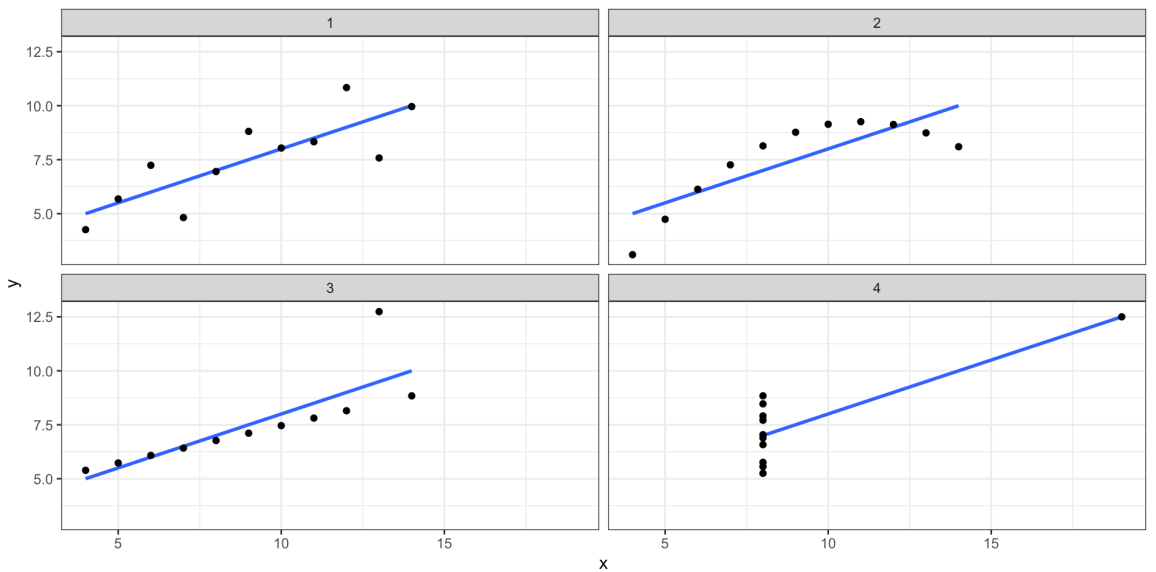
However, plotting the underlying data reveals a problem:

```
ggplot(anscombe_long, aes(x = x, y = y))+
  geom_smooth(method = "lm", se = FALSE)+  geom_point()+
  facet_wrap(set~.)+theme_bw()
```



Anscombe's quartet

Anscombe's quartet is a reminder that although statistics are useful for summarizing data and conveying results, they can also be misleading if we are not careful.



Choosing the correct visualization is important!

- Effective visualizations support claims in easy to understand ways
- Strong skills in selecting and producing visualizations make analysis easier
- Some students struggle with this, but it is super important
- Selecting the appropriate visualization has to do with variable types
- Making good graphics takes practice

Types of data visualization

Appropriate data visualizations vary based on the type of variables that you are trying to display

For a helpful cheatsheet on the types of plots you could use to display various types of data, see the “Cheatsheet” tab of Posit. I’ve uploaded this graphic to Brightspace as well.

The main visualization types we will use are: scatterplots, boxplots/violin plots, barplots, histograms, but there are lot of other options available to you.

Conceptualizing plots

Every plot needs an x and y axis, and you should know what you want to show in both axes

Many programming languages have defaults for some plots that mean you don't have to specify the y-axis for some types of plots

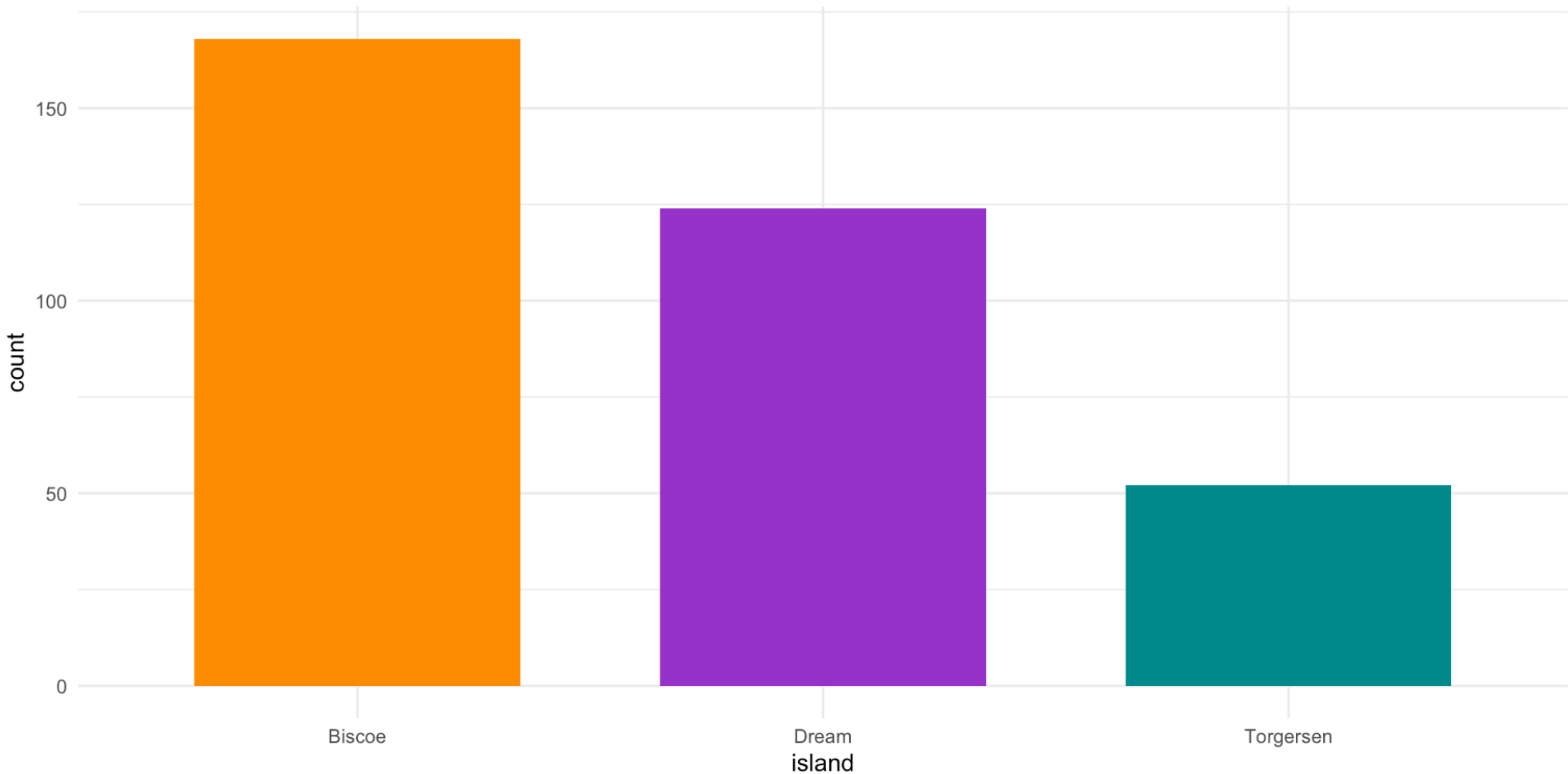
A reminder of the data structure

```
1 head(penguins)
```

```
# A tibble: 6 × 9
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
4 Adelie  Torgersen           NA            NA                  NA          NA
5 Adelie  Torgersen           36.7          19.3               193        3450
6 Adelie  Torgersen           39.3          20.6               190        3650
# ℹ 3 more variables: sex <fct>, year <int>, size <chr>
```

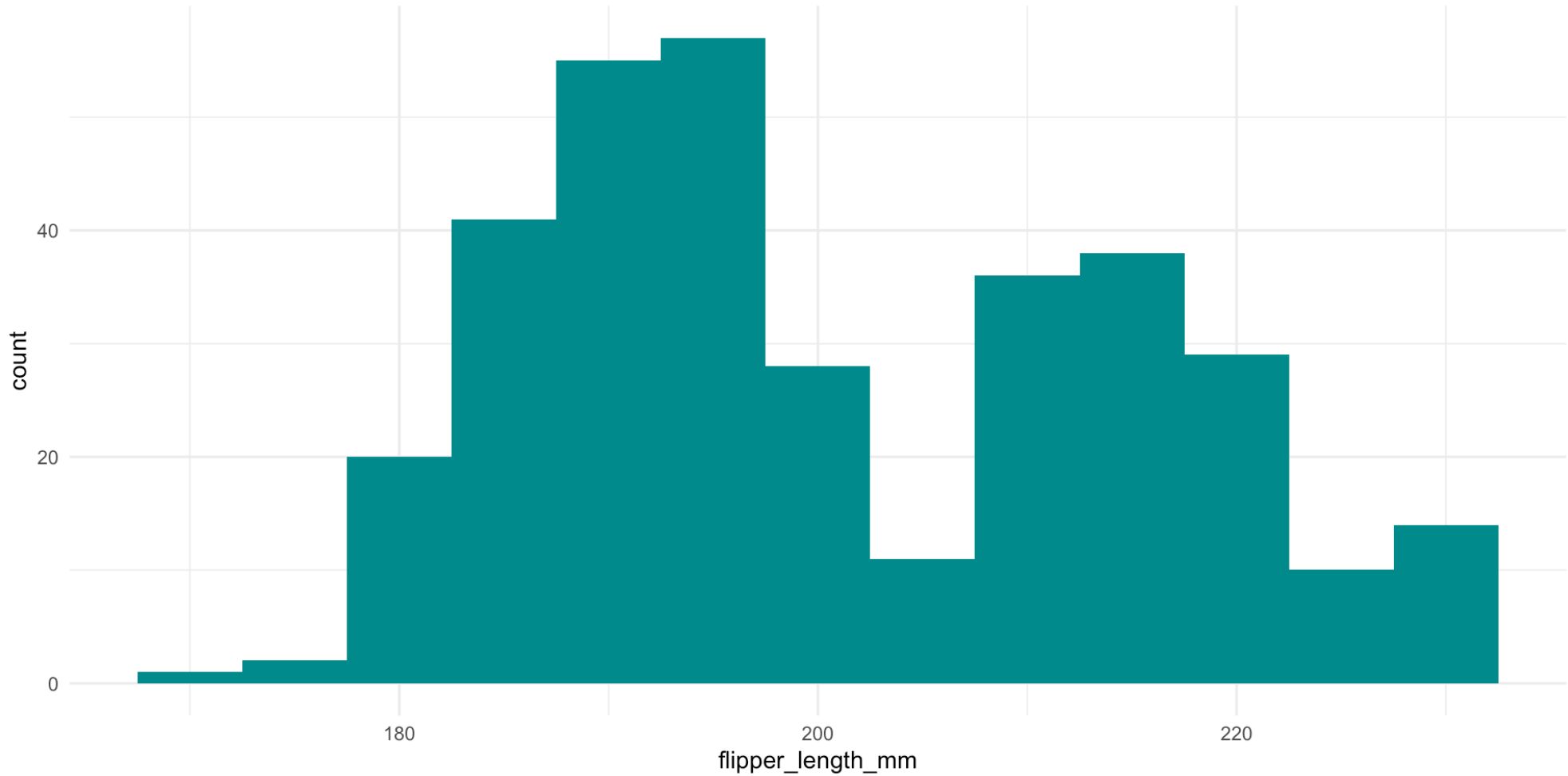
Bar plots: a single discrete variable

What is on the x axis? What is on the y axis?



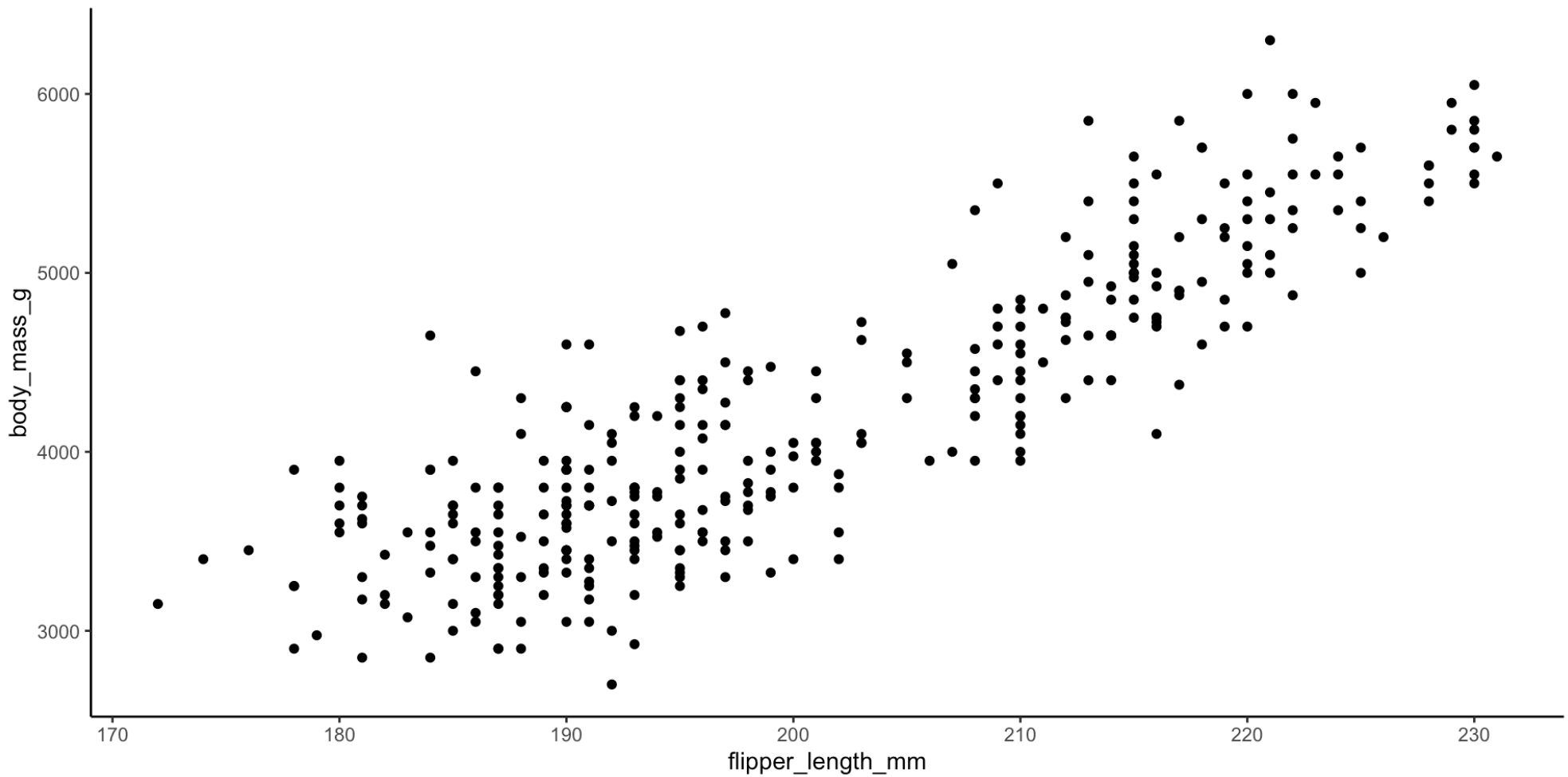
Histograms: a single continuous variable

What is on the x axis? What is on the y axis?



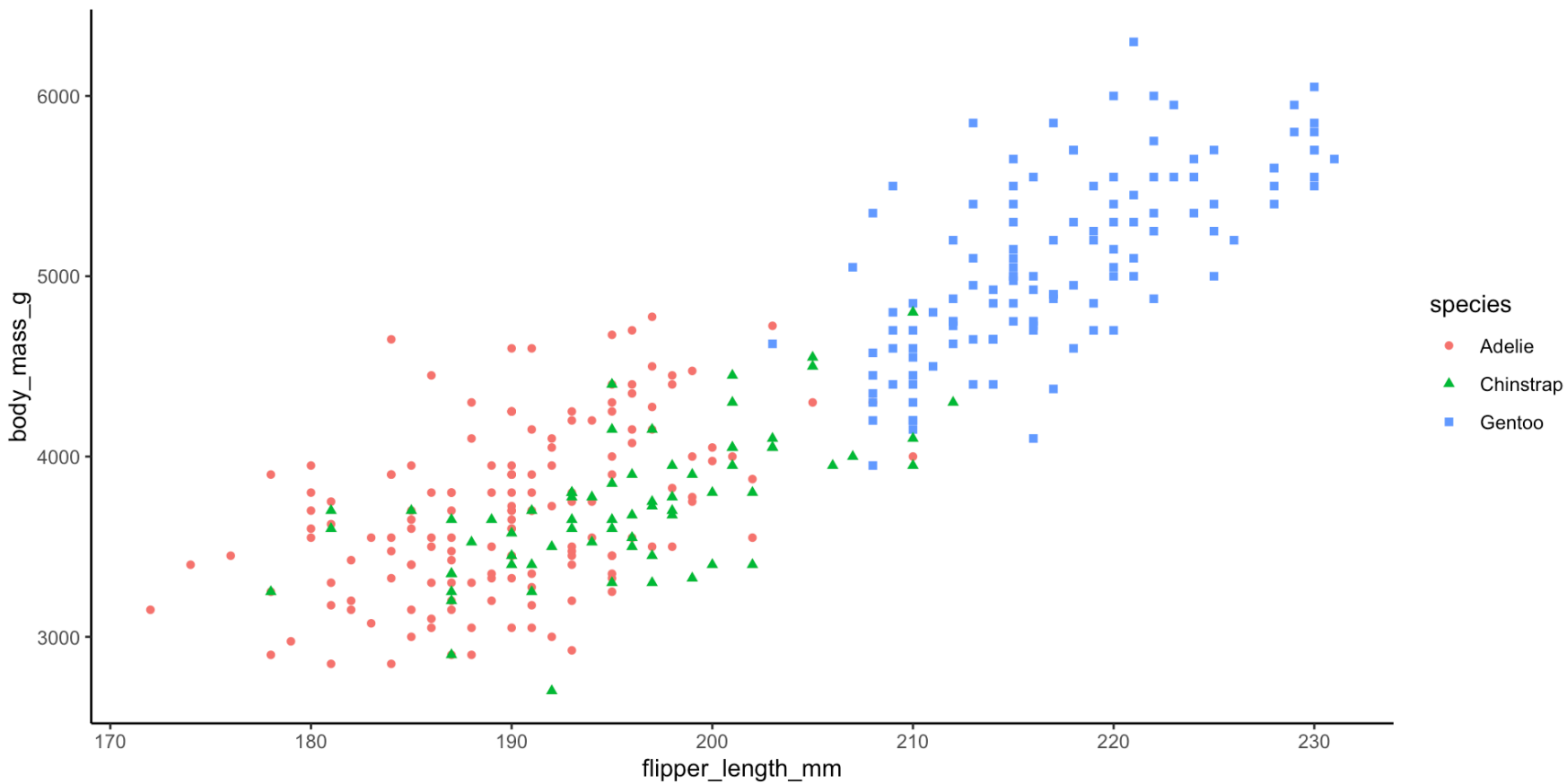
Scatterplots: two continuous variables

What is on the x axis? What is on the y axis?



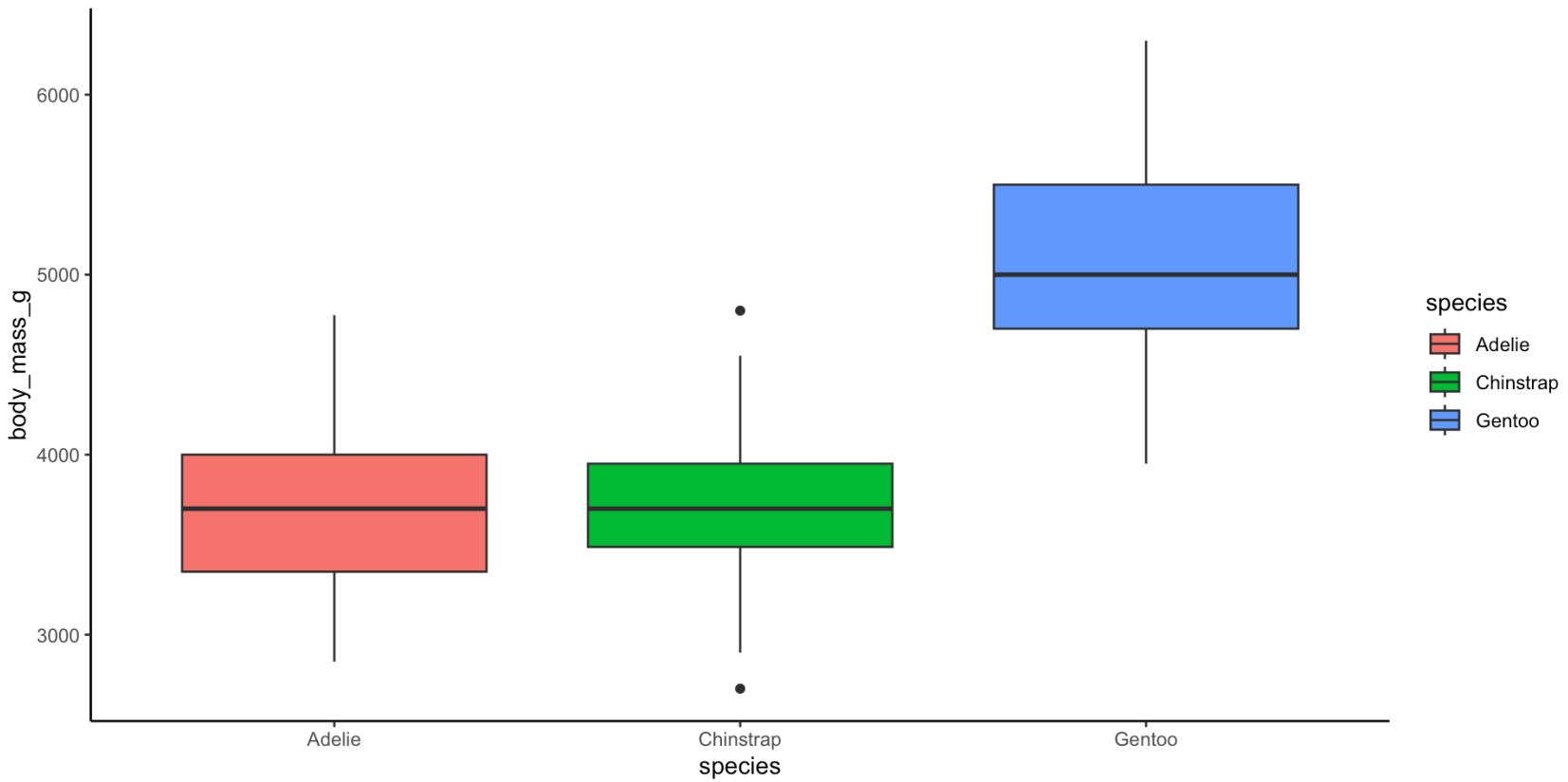
Scatterplots: two continuous variables

We can add a third grouping variable by assigning an attribute to the points themselves, e.g. shape or color



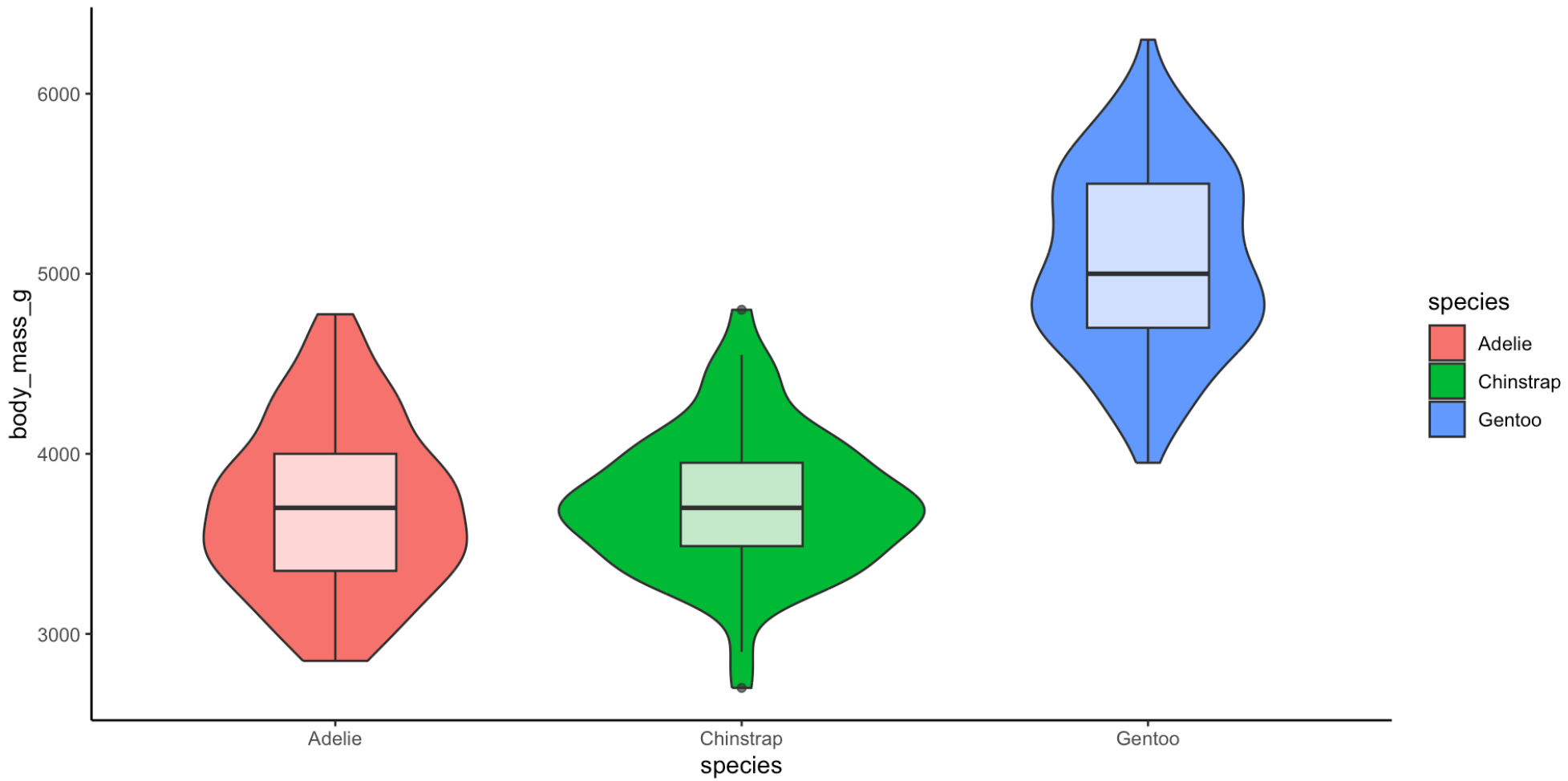
Box plots: one continuous and one discrete variable

What is on the x axis? What is on the y axis?



Violin plots: improving on the box plot

Violin plots show the density distribution of the underlying data



Raincloud plots: improving on the violin plot

Raincloud plots show the density distribution of the underlying data and the underlying data itself!

`ggplot2` - a grammar of graphics

We're going to use the library `ggplot2` to do all of our plotting in this course.

`ggplot2` is written following data visualization principles laid out in *The Grammar of Graphics*

What this means for you is that `ggplot2` and all its extensions use internally consistent syntax, and you only need to do a few things to get a great plot.

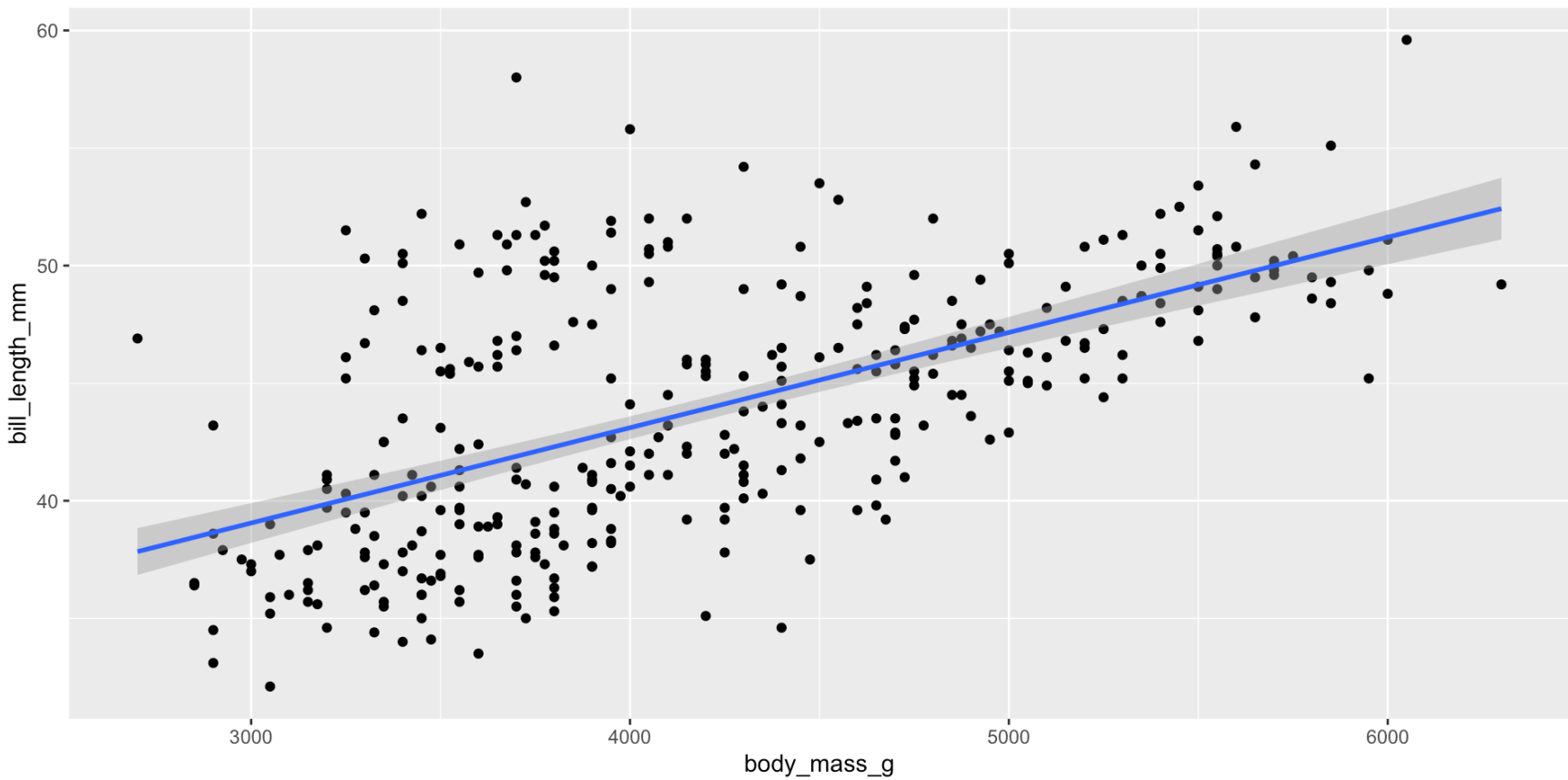
`ggplot2` - a grammar of graphics

Steps to creating a plot using `ggplot2`:

1. Supply the data
2. Supply the aesthetics (which variables, which groups)
3. Specify the type(s) of visualization (geometries)
4. Additional functions to modify and adjust the plot

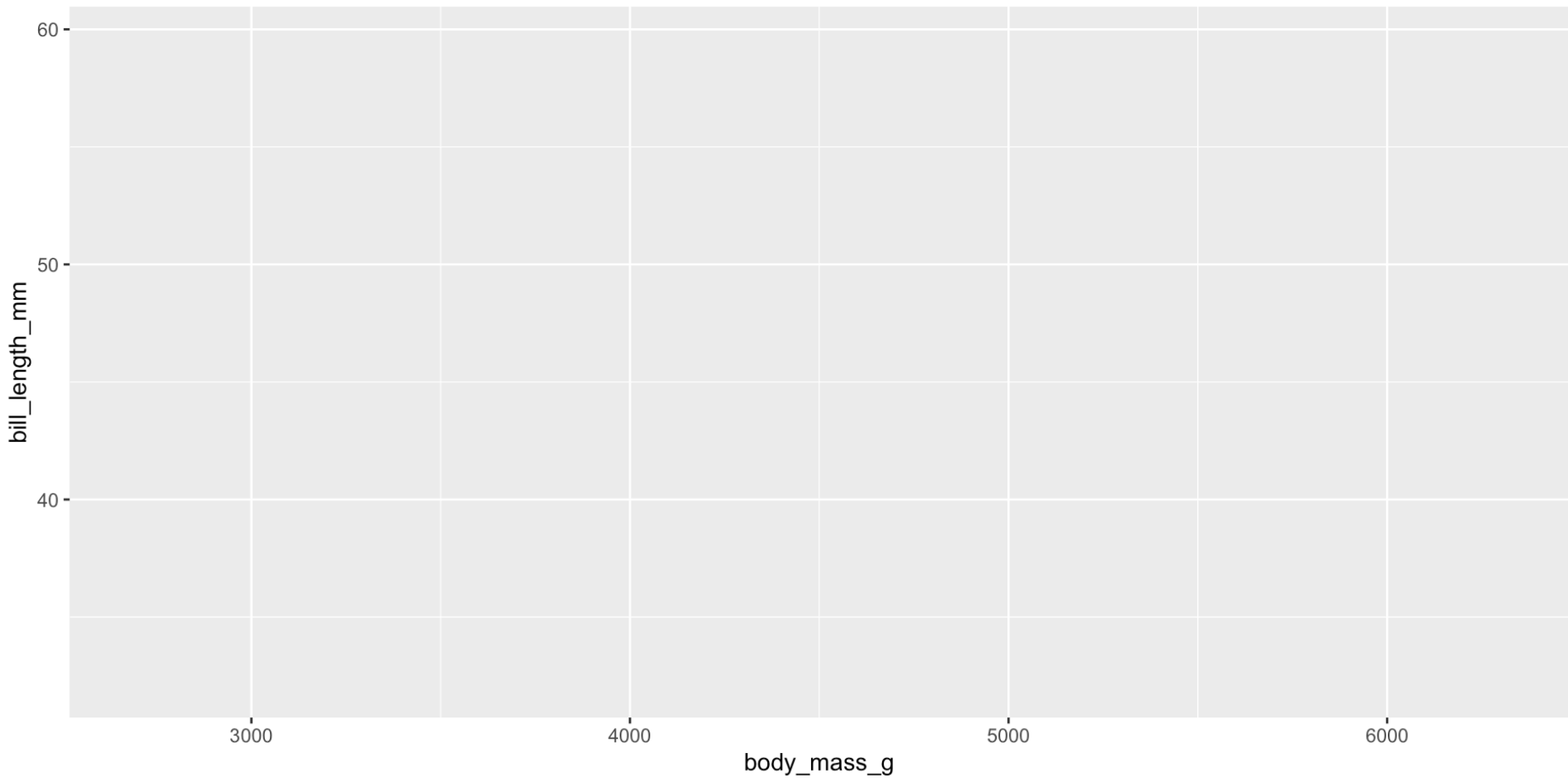
Example

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm))+
  geom_point()+
  geom_smooth(method = "lm")
```



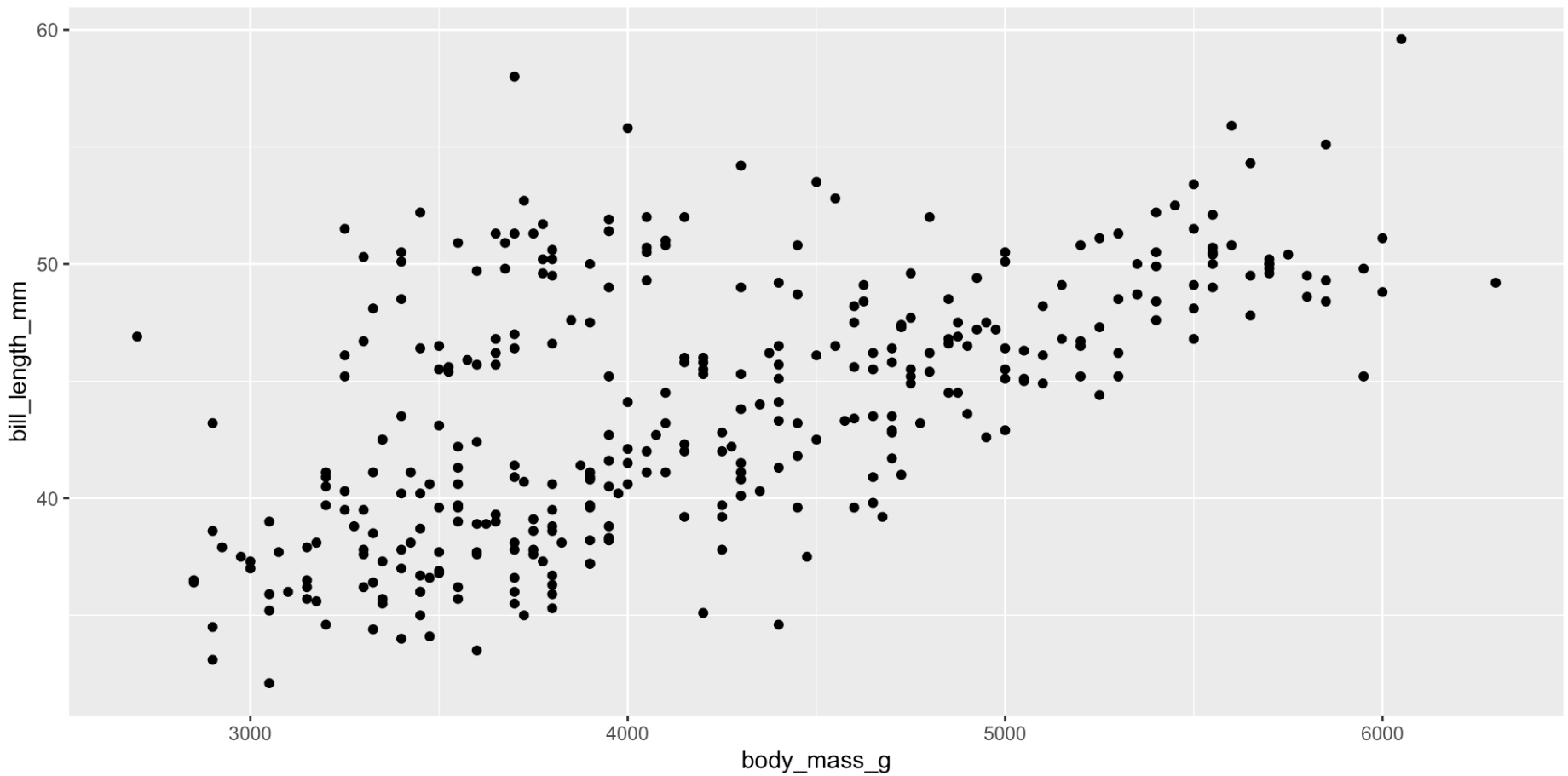
Example

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



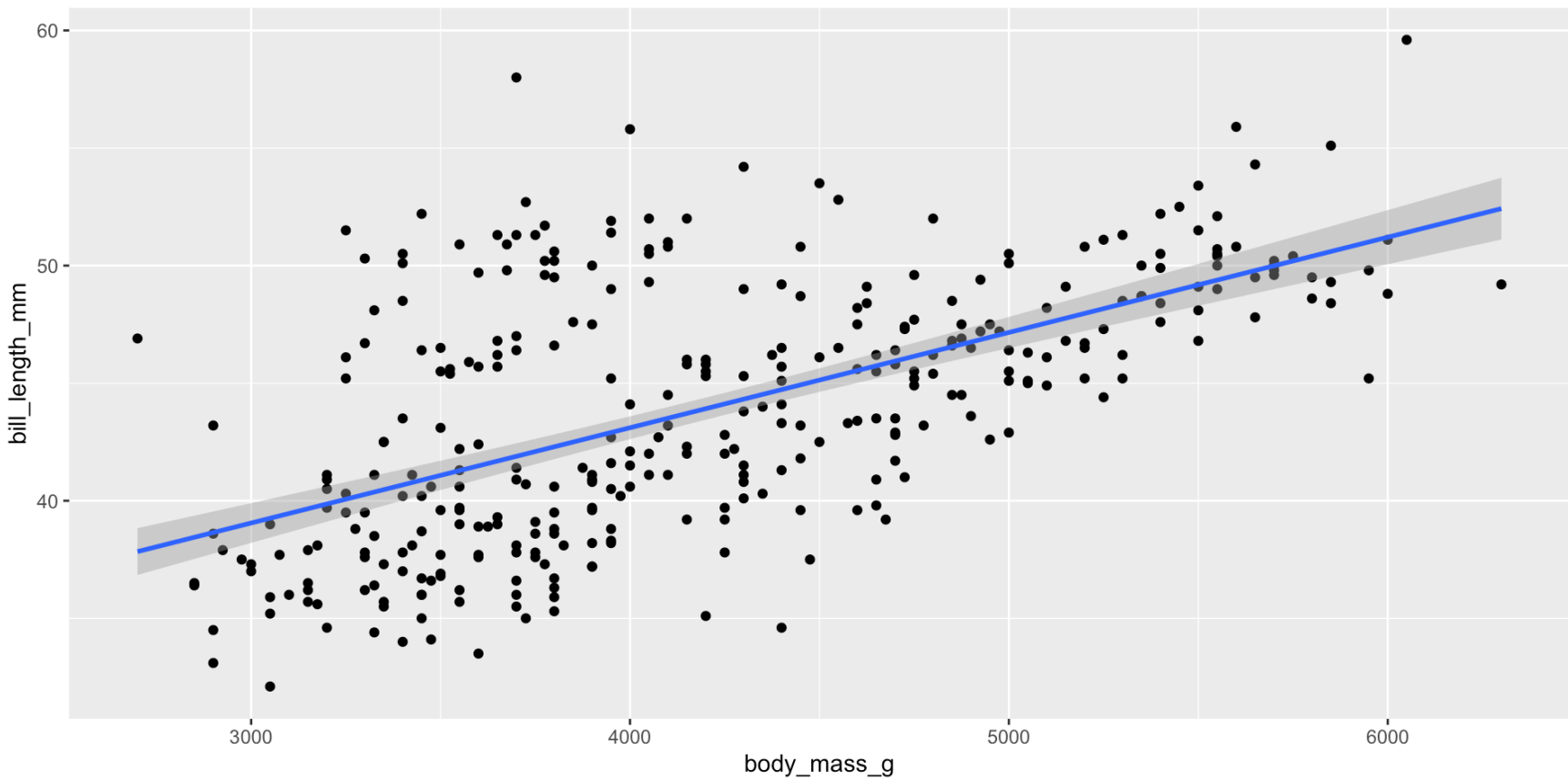
Example

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



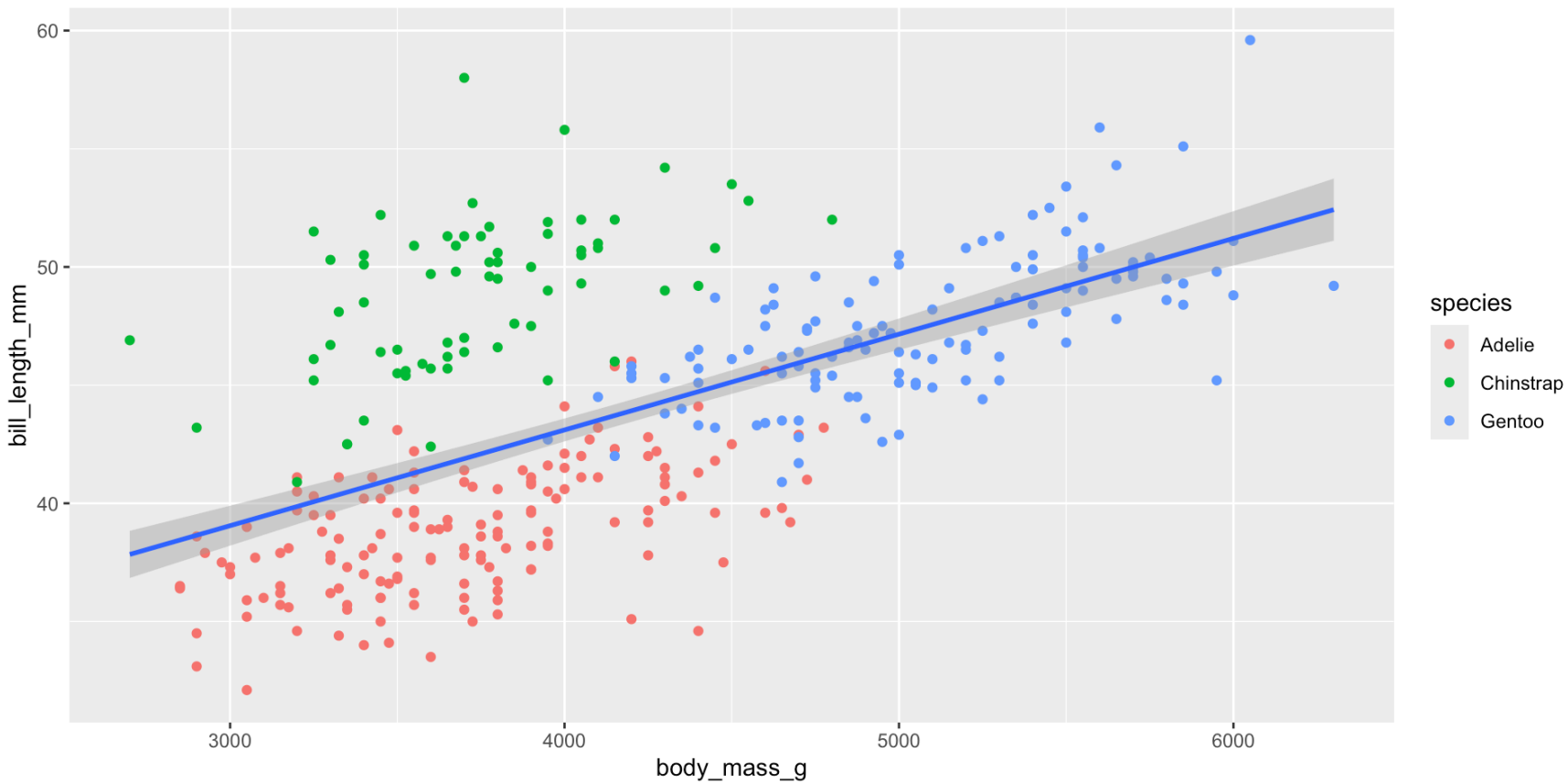
Example

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm))+
  geom_point()+
  geom_smooth(method = "lm")
```



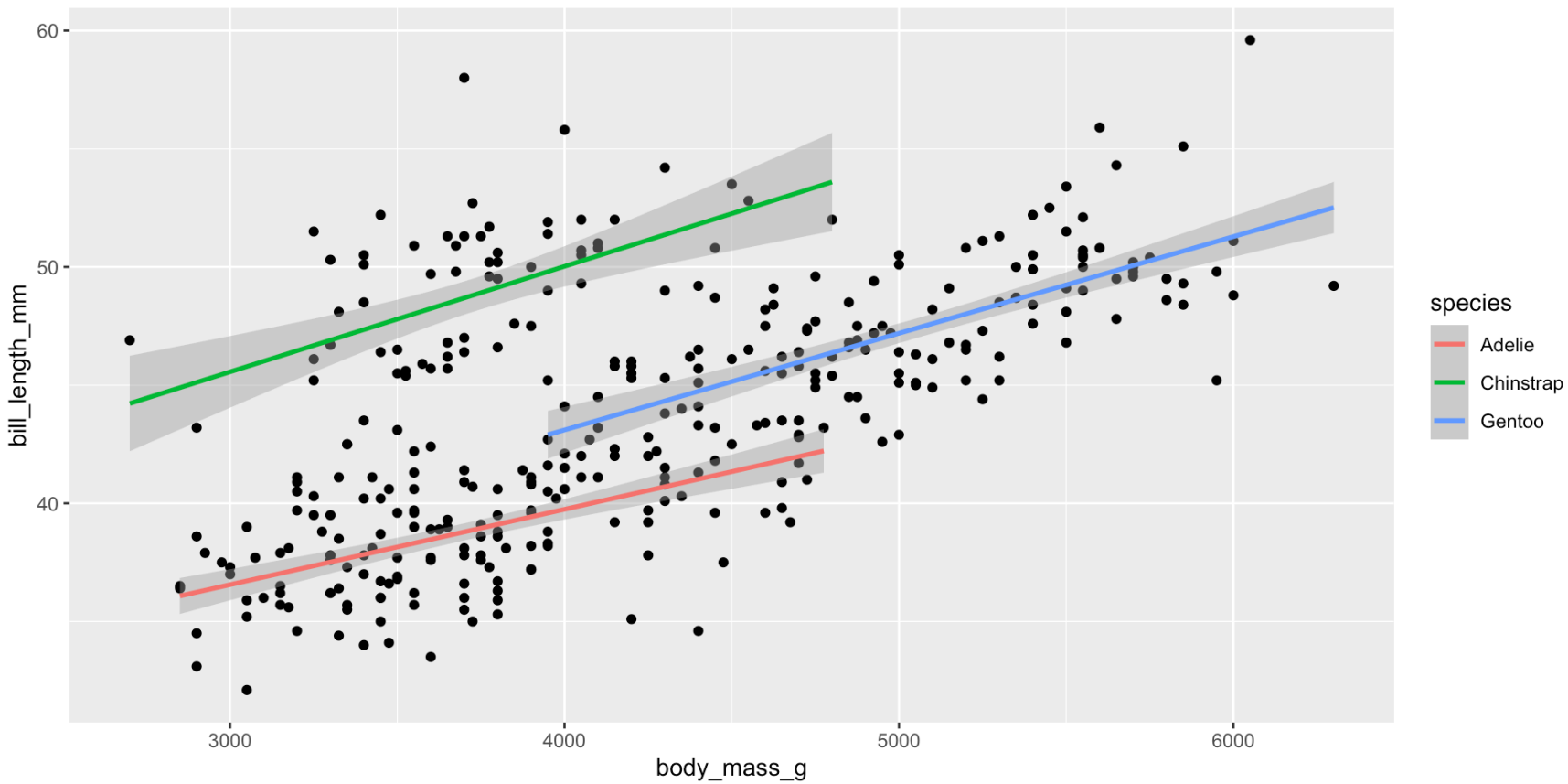
Specifying groups in different geometries

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm))+
  geom_point(aes(color = species))+
  geom_smooth(method = "lm")
```



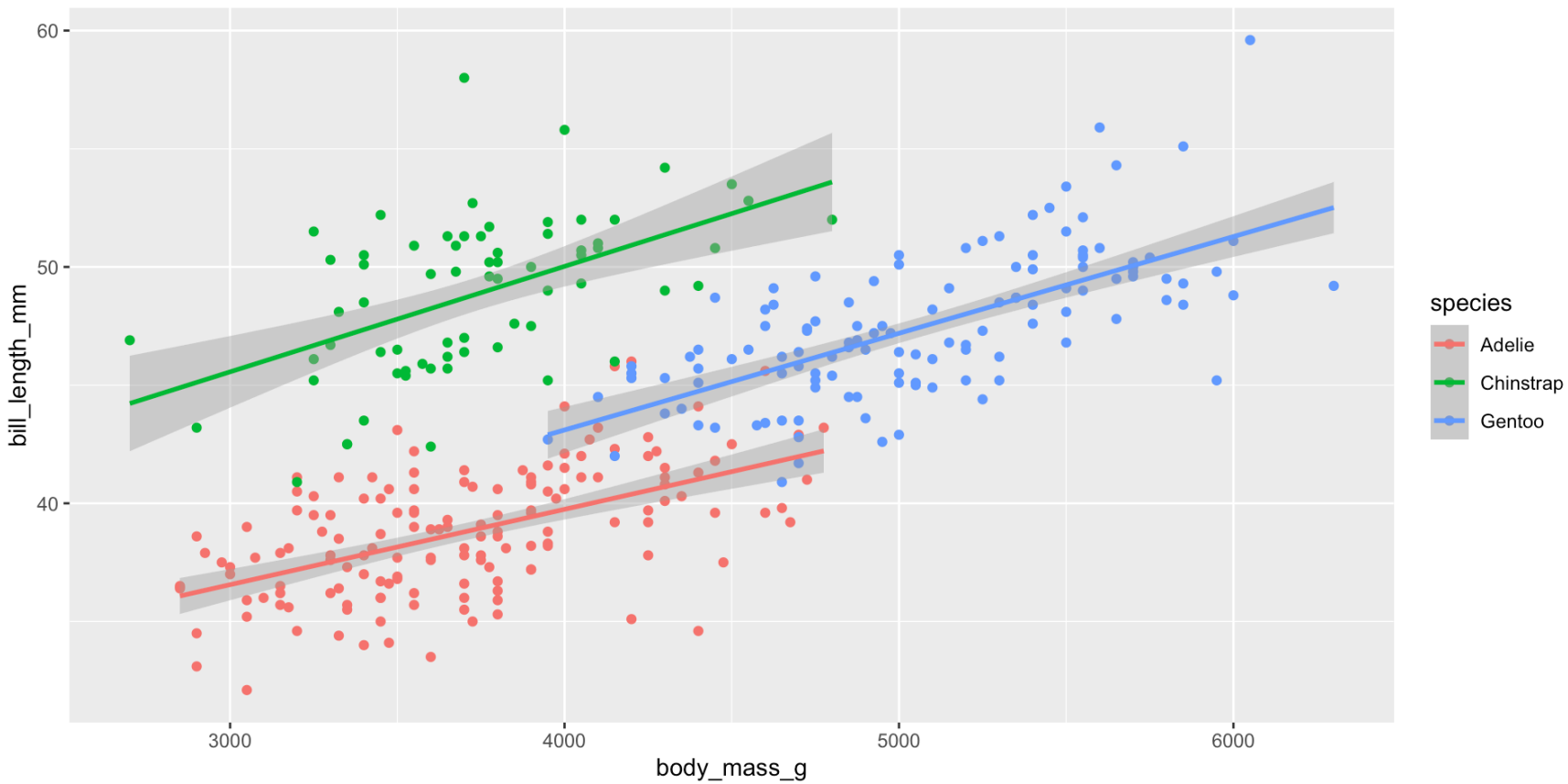
Specifying groups in different geometries

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm))+
  geom_point()+
  geom_smooth(aes(color = species), method = "lm")
```



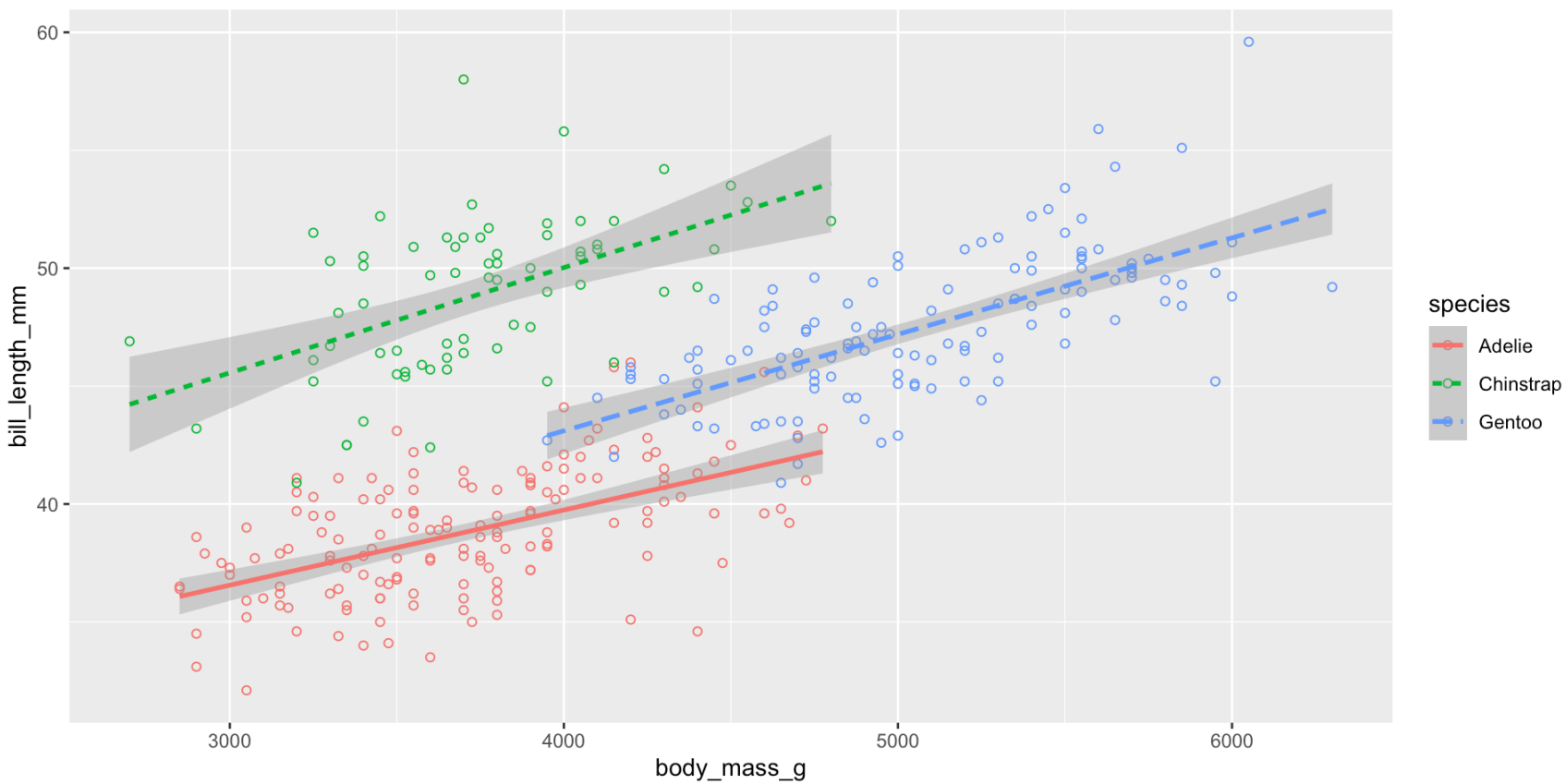
Specifying groups across all geometries

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm, color = species))
  geom_point()+
  geom_smooth(method = "lm")
```



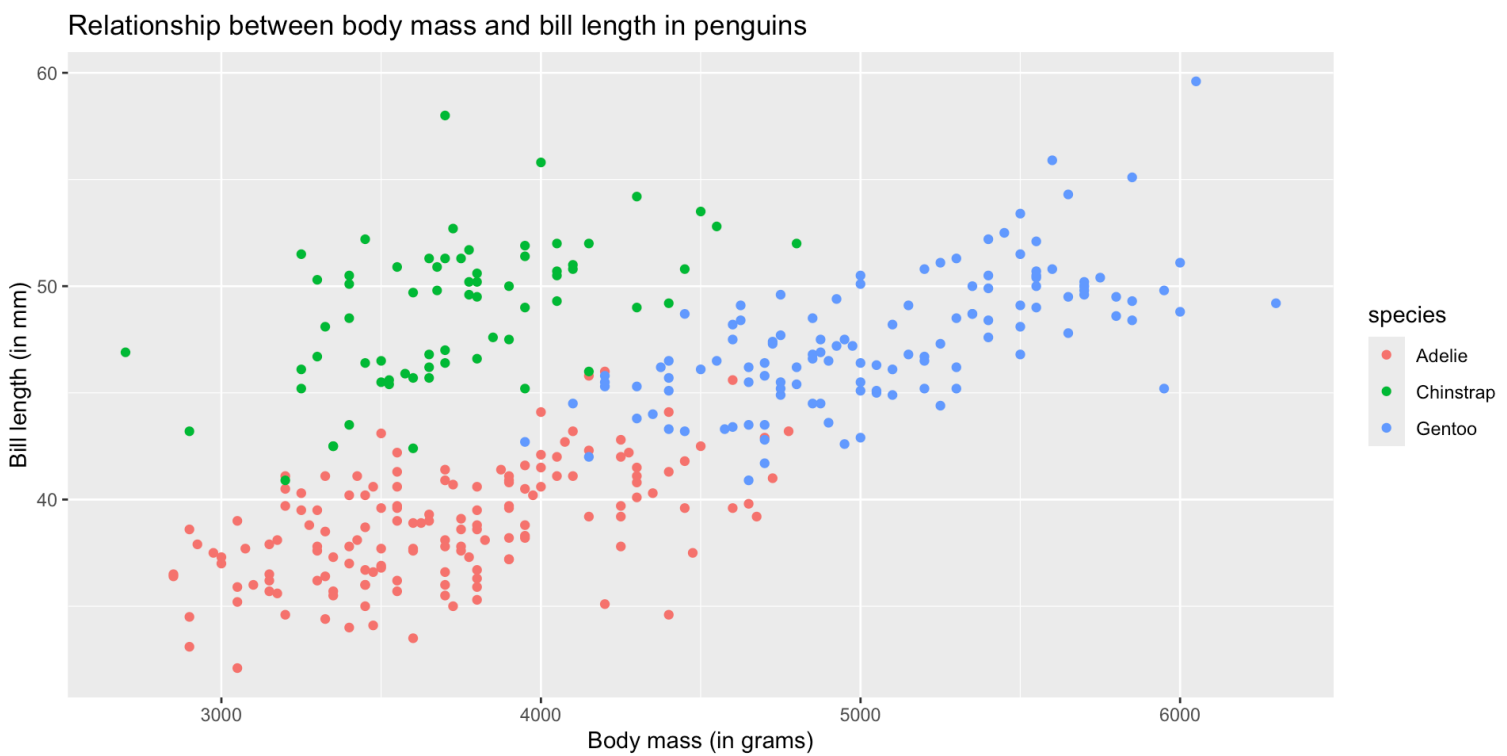
Setting options in/out of `aes()`

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm, color = species))
  geom_point(shape = 1)+
  geom_smooth(aes(lty = species), method = "lm")
```



Adding titles and axis labels

```
ggplot(data = penguins,
       mapping = aes(x = body_mass_g, y = bill_length_mm))+
  geom_point(aes(col = species))+
  labs(title = "Relationship between body mass and bill length in penguins",
       x = "Body mass (in grams)",
       y = "Bill length (in mm)")
```



Commonly used geometries

Here are the ones you most commonly will use:

- Bar plots: `geom_bar()`, needs an x variable
- Histograms: `geom_histogram()`, needs an x variable
- Scatter plots: `geom_point()`, needs x and y variables
- Box plots: `geom_boxplot()`, needs x and y variables

You can endlessly customize your plots, refer to cheatsheets or to the cook book for help!

Your turn - Exercise 2

1. Working with data frames

Introducing `dplyr`

`dplyr` is a package within the `tidyverse` whose purpose is to help us wrangle data - that is reshape and manipulate it

Introducing `dplyr`

The main functions we will introduce today from `dplyr` are:

- `filter()`
- `select()`
- `mutate()`
- `rename()`

Introducing `dplyr`

All of the functions in this package share a few commonalities:

- The first argument is a data frame type object (data frame or tibble)
- Subsequent arguments describe what to do with the data frame. You can refer to columns directly by name without $ operator
- The result returns a new data frame
- Data frames must be in tidy format to take full advantage of dplyr

Accessing dataframe elements with $

Each dataframe is composed of vectors. These are technically elements of the dataframe.

```
1 str(penguins)
```

```
tibble [344 × 9] (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
```

Accessing dataframe elements with $

In order to access an element of a dataframe (a vector), you can use the $ operator to specify it by name

```
1 penguins$bill_length_mm %>% head(30)
```

```
 [1] 39.1 39.5 40.3   NA 36.7 39.3 38.9 39.2 34.1 42.0 37.8 37.8 41.1 38.6
34.6
[16] 36.6 38.7 42.5 34.4 46.0 37.8 37.7 35.9 38.2 38.8 35.3 40.6 40.5 37.9
40.5
```

```
1 penguins$species %>% head(30)
```

```
 [1] Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie
[11] Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie
[21] Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie Adelie
Levels: Adelie Chinstrap Gentoo
```

Accessing dataframe elements with $

You can perform operations on a vector from a dataframe just as you would on any other vector:

```
1  bill_length_cm <- penguins$bill_length_mm/10
2
3  head(bill_length_cm, 30)
```

```
 [1] 3.91 3.95 4.03   NA 3.67 3.93 3.89 3.92 3.41 4.20 3.78 3.78 4.11 3.86
3.46
[16] 3.66 3.87 4.25 3.44 4.60 3.78 3.77 3.59 3.82 3.88 3.53 4.06 4.05 3.79
4.05
```

Accessing dataframe elements with $

You can also define a new variable in a dataframe using the $ and <- operators

```
1 penguins$bill_length_cm <- penguins$bill_length_mm/10
2
3 colnames(penguins)
```

```
 [1] "species"           "island"            "bill_length_mm"
 [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
 [7] "sex"               "year"              "size"
[10] "bill_length_cm"
```

Logical operators for subsetting

Here are the logical operators you should commit to memory:

- `==` - Equals
- `!=` - Does not equal
- `>` and `>=` - Greater than and Greater than or equal to
- `<` and `<=` - Less than and Less than or equal to
- `%in%` - Included in (followed by a vector)
- `is.na()` - Is a missing value
- `&` and `|` - And ; Or, for stringing multiple criteria

Selecting columns with `select()`

Sometimes with large datasets, or when manipulating columns you want to select only certain columns to keep. `select()` is a very useful way to do this.

```
1 select(penguins, species, year)
```

```
# A tibble: 344 × 2
   species  year
   <fct>   <int>
 1 Adelie   2007
 2 Adelie   2007
 3 Adelie   2007
 4 Adelie   2007
 5 Adelie   2007
 6 Adelie   2007
 7 Adelie   2007
 8 Adelie   2007
 9 Adelie   2007
10 Adelie   2007
# ℹ 334 more rows
```

Selecting columns with `select()`

We can select multiple consecutive columns with `:`, which will keep all columns starting with the first specified and ending with the second specified.

```
1 select(penguins, species:bill_length_mm, year)
```

```
# A tibble: 344 × 4
   species island    bill_length_mm  year
   <fct>   <fct>              <dbl> <int>
 1 Adelie  Torgersen           39.1  2007
 2 Adelie  Torgersen           39.5  2007
 3 Adelie  Torgersen           40.3  2007
 4 Adelie  Torgersen           NA    2007
 5 Adelie  Torgersen           36.7  2007
 6 Adelie  Torgersen           39.3  2007
 7 Adelie  Torgersen           38.9  2007
 8 Adelie  Torgersen           39.2  2007
 9 Adelie  Torgersen           34.1  2007
10 Adelie  Torgersen           42    2007
# ℹ 334 more rows
```

Selecting columns with `select()`

We can select also pair select with helper functions such as `starts_with()`, `ends_with()`, `contains()`, `matches()`

```
1 select(penguins, ends_with("_mm"))
```

```
# A tibble: 344 × 3
   bill_length_mm bill_depth_mm flipper_length_mm
            <dbl>         <dbl>             <int>
 1           39.1          18.7               181
 2           39.5          17.4               186
 3           40.3          18                 195
 4           NA            NA                  NA
 5           36.7          19.3               193
 6           39.3          20.6               190
 7           38.9          17.8               181
 8           39.2          19.6               195
 9           34.1          18.1               193
10           42            20.2               190
# ℹ 334 more rows
```

Selecting columns with `select()`

We can also "negatively select," in the sense that we can get rid of columns while keeping the rest by prefacing our selection arguments with a – sign:

```
1 select(penguins, -ends_with("_mm"))
```

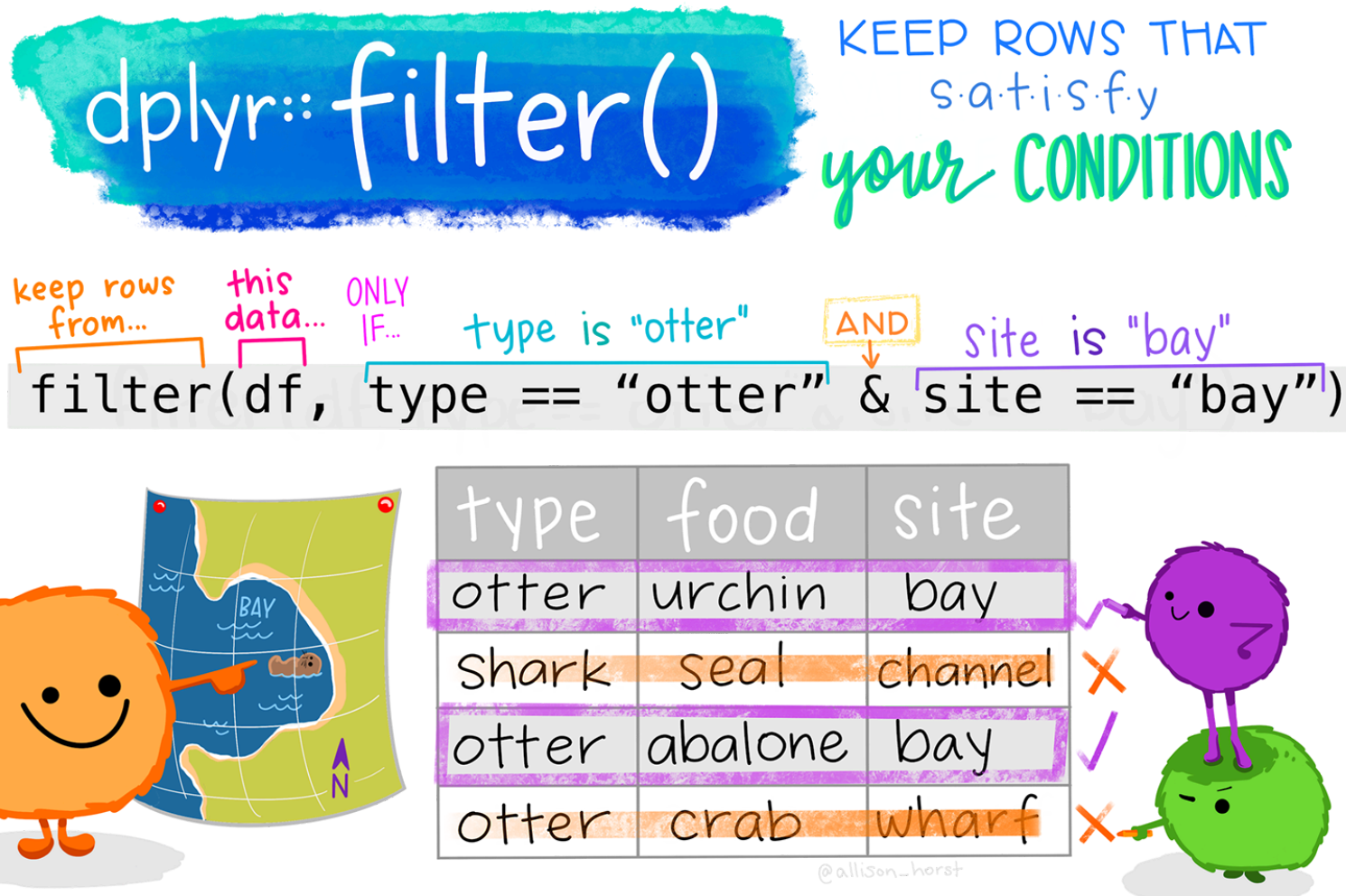
```
# A tibble: 344 × 7
   species island    body_mass_g sex      year size    bill_length_cm
   <fct>   <fct>           <int> <fct>   <int> <chr>              <dbl>
 1 Adelie  Torgersen        3750 male     2007 small              3.91
 2 Adelie  Torgersen        3800 female   2007 small              3.95
 3 Adelie  Torgersen        3250 female   2007 x-small            4.03
 4 Adelie  Torgersen          NA <NA>     2007 <NA>                NA
 5 Adelie  Torgersen        3450 female   2007 x-small            3.67
 6 Adelie  Torgersen        3650 male     2007 small              3.93
 7 Adelie  Torgersen        3625 female   2007 small              3.89
 8 Adelie  Torgersen        4675 male     2007 medium             3.92
 9 Adelie  Torgersen        3475 <NA>     2007 x-small            3.41
10 Adelie  Torgersen        4250 <NA>     2007 medium             4.2
# ℹ 334 more rows
```

Subsetting columns with `select()` rather than indexing

It is possible to select columns with indexing, as we did with vectors e.g. `penguins[,1:3]` returns columns 1-3.

This is not great practice. If we were to change the ordering of the columns, either through uploading a new dataset or by changing a data management step, we would have to revisit our indexing.

Filtering a dataset with `filter()`



Filtering a dataset with `filter()`

We can filter for multiple values in one call

```
1 filter(penguins, species == "Adelie" & year != 2007)
```

```
# A tibble: 102 × 10
   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
 1 Adelie  Biscoe            39.6          17.7               186        3500
 2 Adelie  Biscoe            40.1          18.9               188        4300
 3 Adelie  Biscoe            35            17.9               190        3450
 4 Adelie  Biscoe            42            19.5               200        4050
 5 Adelie  Biscoe            34.5          18.1               187        2900
 6 Adelie  Biscoe            41.4          18.6               191        3700
 7 Adelie  Biscoe            39            17.5               186        3550
 8 Adelie  Biscoe            40.6          18.8               193        3800
 9 Adelie  Biscoe            36.5          16.6               181        2850
10 Adelie  Biscoe            37.6          19.1               194        3750
# ℹ 92 more rows
# ℹ 4 more variables: sex <fct>, year <int>, size <chr>, bill_length_cm <dbl>
```

Filtering a dataset with `filter()`

To filter for missing values (NA) use `is.na()` or `!is.na()`

```
1 filter(penguins, is.na(flipper_length_mm))
```

```
# A tibble: 2 × 10
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen             NA            NA                NA          NA
2 Gentoo  Biscoe               NA            NA                NA          NA
# ℹ 4 more variables: sex <fct>, year <int>, size <chr>, bill_length_cm <dbl>
```

Filtering a dataset with `filter()`

To filter for multiple values of a variable, use `|` or `%in%`

```
1 filter(penguins, species == "Adelie" | species == "Gentoo")
```

```
# A tibble: 276 × 10
   species island    bill_length_mm bill_depth_mm flipper_length_mm
body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>
<int>
 1 Adelie  Torgersen           39.1          18.7               181
3750
 2 Adelie  Torgersen           39.5          17.4               186
3800
 3 Adelie  Torgersen           40.3          18                 195
3250
 4 Adelie  Torgersen           NA            NA                  NA
NA
 5 Adelie  Torgersen           36.7          19.3               193
3450
```

Filtering a dataset with `filter()`

To filter for multiple values of a variable, use `|` or `%in%`

```
1 filter(penguins, species %in% c("Adelie", "Gentoo"))
```

```
# A tibble: 276 × 10
   species island    bill_length_mm bill_depth_mm flipper_length_mm
body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>
<int>
 1 Adelie  Torgersen           39.1          18.7               181
3750
 2 Adelie  Torgersen           39.5          17.4               186
3800
 3 Adelie  Torgersen           40.3          18                 195
3250
 4 Adelie  Torgersen           NA            NA                  NA
NA
 5 Adelie  Torgersen           36.7          19.3               193
3450
```

Filtering a dataset with `filter()`

To filter for multiple values of a variable, use `|` or `%in%`

```
1 targetspecies <- c("Adelie", "Gentoo")
2 filter(penguins, species %in% targetspecies)
```

```
# A tibble: 276 × 10
   species island    bill_length_mm bill_depth_mm flipper_length_mm
body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>
<int>
 1 Adelie  Torgersen           39.1          18.7               181
3750
 2 Adelie  Torgersen           39.5          17.4               186
3800
 3 Adelie  Torgersen           40.3          18                 195
3250
 4 Adelie  Torgersen           NA            NA                  NA
NA
 5 Adelie  Torgersen           36.7          19.3               193
3450
```

Renaming columns with `rename()`

Renaming columns is a breeze, using the sytax `new = old`. This is so much better than having to use `colnames()` and indexing!

```
1 rename(penguins, bill.l = bill_length_mm, flip.l = flipper_length_mm)
```

```
# A tibble: 344 × 10
   species island    bill.l bill_depth_mm flip.l body_mass_g sex     year size
   <fct>   <fct>      <dbl>         <dbl>  <int>       <int> <fct>  <int>
<chr>
 1 Adelie  Torgersen   39.1          18.7    181        3750 male    2007
small
 2 Adelie  Torgersen   39.5          17.4    186        3800 female  2007
small
 3 Adelie  Torgersen   40.3          18      195        3250 female  2007 x-
sma…
 4 Adelie  Torgersen   NA            NA       NA          NA <NA>    2007 <NA>
 5 Adelie  Torgersen   36.7          19.3    193        3450 female  2007 x-
sma…
 6 Adelie  Torgersen   39.3          20.6    190        3650 male    2007
small
```

Adding new columns with `mutate()`

`mutate()` allows us to add new columns to our dataset. It's especially useful for defining new columns as a computation of other columns.

Adding new columns with `mutate()`

```
1 mutate(penguins, bill_length_mm = bill_length_cm/10)
```

```
# A tibble: 344 × 10
   species island    bill_length_mm bill_depth_mm flipper_length_mm
body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>
<int>
 1 Adelie  Torgersen          0.391          18.7               181
3750
 2 Adelie  Torgersen          0.395          17.4               186
3800
 3 Adelie  Torgersen          0.403          18                 195
3250
 4 Adelie  Torgersen         NA              NA                  NA
NA
 5 Adelie  Torgersen          0.367          19.3               193
3450
```

Adding new columns with `mutate()`

Let's add a column called `cute`, which is coded "yes" if bill length < 35mm, and "no" if bill length >= 35mm

```
1 mutate(penguins, cute = ifelse(bill_length_mm < 35, "Yes", "No"))
```

```
# A tibble: 344 × 11
   species island    bill_length_mm bill_depth_mm flipper_length_mm
body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>
<int>
 1 Adelie  Torgersen           39.1          18.7               181
3750
 2 Adelie  Torgersen           39.5          17.4               186
3800
 3 Adelie  Torgersen           40.3          18                 195
3250
 4 Adelie  Torgersen           NA            NA                  NA
NA
 5 Adelie  Torgersen           36.7          19.3               193
3450
```

Adding new columns with `mutate()`

Let's add a column that calculates the z-score for bill length - this can be done in one line, but let's do it in two to illustrate that we can call on new variables within a mutate call:

```
1 mutate(penguins,
2        bill_length_z = bill_length_mm - mean(bill_length_mm, na.rm = TRUE),
3        bill_length_z = bill_length_z/ sd(bill_length_mm, na.rm = TRUE))   %
```

```
# A tibble: 344 × 4
   species island    bill_length_mm bill_length_z
   <fct>   <fct>              <dbl>         <dbl>
 1 Adelie  Torgersen           39.1        -0.883
 2 Adelie  Torgersen           39.5        -0.810
 3 Adelie  Torgersen           40.3        -0.663
 4 Adelie  Torgersen           NA          NA
 5 Adelie  Torgersen           36.7        -1.32
 6 Adelie  Torgersen           39.3        -0.847
 7 Adelie  Torgersen           38.9        -0.920
 8 Adelie  Torgersen           39.2        -0.865
 9 Adelie  Torgersen           34.1        -1.80
```

The pipe operator %>%

Often times, we want to string together multiple operations. The pipe operator allows us to do that.

Kind of like a `ggplot2` layers, the pipe inherits the data from the previous operation and passes it on to the next.

A few %>% example

When we want to use a pipe, we move the name of the dataframe out of the function:

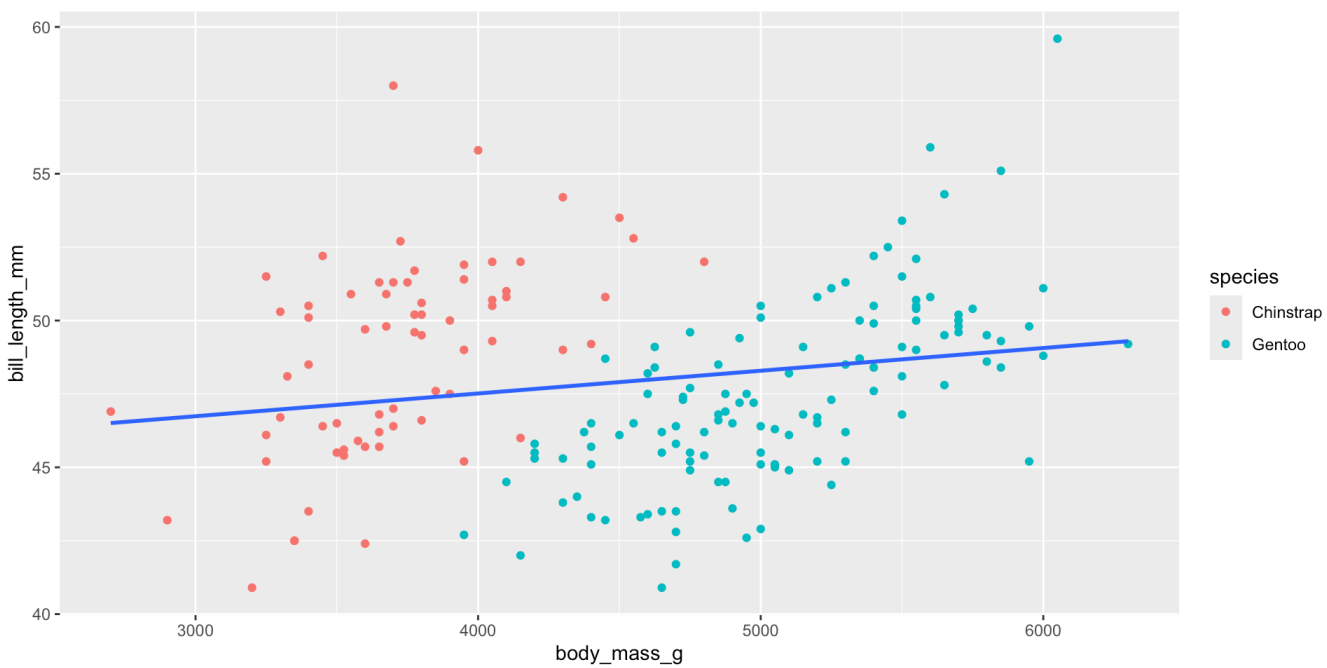
```
penguins %>%
  mutate(bill_length_cm = bill_length_mm/10)%>%
  filter(species == "Gentoo")%>%
  select(island, sex, bill_length_cm)
```

```
# A tibble: 124 × 3
   island sex    bill_length_cm
   <fct>  <fct>           <dbl>
 1 Biscoe female            4.61
 2 Biscoe male              5
 3 Biscoe female            4.87
 4 Biscoe male              5
 5 Biscoe male              4.76
 6 Biscoe female            4.65
 7 Biscoe female            4.54
 8 Biscoe male              4.67
 9 Biscoe female            4.33
10 Biscoe male              4.68
# ℹ 114 more rows
```

A few %>% example

We can also string dplyr pipes with ggplot2

```
1  penguins %>%
2    filter(species %in% c("Gentoo", "Chinstrap"))%>%
3    ggplot(aes(x = body_mass_g, y = bill_length_mm))+ # note we use +
4      geom_point(aes(col = species))+
5      geom_smooth(method = "lm", se = FALSE)
```



Other useful functions

We're not going to cover these in detail, but they are pretty useful and you may need to refer to them later:

- `relocate()` for moving columns relative to others within a dataframe
- `arrange()` for sorting by values in a column

Summarizing

Last week, we produced summaries with base R functions for quick assessment of individual groups. We also used `table1` to make nicely formatted tables summarizing multiple variables.

There is one more use for summarizing data that is super useful for data analysis: summarizing across groups.

Summarizing across groups

This can be very useful when we want to produce summary statistics for different groups, or even better, when we want to derive new variables for use in future analyses.

For example, in this dataset, you might want to count the number of children born to each woman

	id	child	age	sex	coreside	biodad
1	5f2cc1361992661d1aae04a8	child1	9.500000	Female	Yes	Yes
2	5f2cc1361992661d1aae04a8	child2	5.666667	Female	Yes	Yes
3	5f2cc1361992661d1aae04a8	child3	2.833333	Male	Yes	Yes
4	5f0a30f39111162e746ff6da	child1	4.750000	Female	Yes	<NA>
5	5f0a30f39111162e746ff6da	child2	0.750000	Female	Yes	<NA>
6	5dbcab7226c04c250046cab5	child1	8.416667	Female	Yes	Yes

Summarizing across groups

You can achieve this with `group_by()` and `summarize()` like so:

```
                           id  child
1   5f2cc1361992661d1aae04a8 child1
2   5f2cc1361992661d1aae04a8 child2
3   5f2cc1361992661d1aae04a8 child3
4   5f0a30f39111162e746ff6da child1
5   5f0a30f39111162e746ff6da child2
6   5dbcab7226c04c250046cab5 child1
7   5dbcab7226c04c250046cab5 child2
8   5cdd6db6c07328001a4bf67e child1
9   5cdd6db6c07328001a4bf67e child2
10  5cdd6db6c07328001a4bf67e child3
```

```r
child %>%
  filter(!is.na(age))%>%
  group_by(id)%>%
  summarize(n_kids = n())
```

```
# A tibble: 1,532 × 2
   id                       n_kids
   <chr>                     <int>
 1 544fe86dfdf99b56bcf1f8ce      1
 2 54847013fdf99b0379939c8a      2
 3 5532496efdf99b1fccde4347      1
 4 559ae660fdf99b361ac4662c      2
 5 55b249bdfdf99b28e3c14b05      3
 6 55cc0397fe3304000562553b      2
 7 55ce148b34e9060012e56279      2
 8 55d0d4b334e9060005e57470      2
 9 55d35a63da14d7001295328e      3
10 560e5dc454221f001035e2f4      1
# ℹ 1,522 more rows
```

Summarizing across groups

You can also produce lots of other summary statistics:

```
penguins %>%
  group_by(species)%>%
  summarize(n = n(),
            mean_bill_l = mean(bill_length_mm, na.rm = TRUE),
            sd_bill_l = sd(bill_length_mm, na.rm = TRUE))
```

```
# A tibble: 3 × 4
  species       n mean_bill_l sd_bill_l
  <fct>     <int>       <dbl>     <dbl>
1 Adelie      152        38.8      2.66
2 Chinstrap    68        48.8      3.34
3 Gentoo      124        47.5      3.08
```