Statistical basics in R

ANTH 572S

Laure Spake

BINGHAMTON
UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Brief review of last week

- We refreshed how to interact with data frames
- You learned how to produce data visualizations

Learning objectives

At the end of this lesson you will:

- Learn how to produce summary statistics
- Learn key functions for wrangling dataframes
- Refresh your understanding of the normal distribution
- Do more data visualizations

1. Choosing and producing summary statistics

What are descriptive statistics?

A set of numbers that describe the distribution of a variable

Why spend time on descriptive statistics?

1. Communicate information about the distribution of data

2. Form the basis for more further statistical concepts

Which descriptive statistics?

1. For numerical data: central tendency and dispersion

2. For categorial data: frequency/proportion tables

Descriptive statistics for numerical data

Measures of central tendency

Statistics that describe the middle of a distribution:

1. Mean
2. Median
3. Mode

Measures of central tendency convey the most typical value in the variable

The mean

The mean is the most commonly used measure of central tendency. It is defined as the sum of all the scores divided by the number of scores.

$$\overline{X} = \frac{\sum X}{N}$$

The mean

What is the mean of this vector?

```
1 p <- c(1, 2, 3, 4, 5) # same result with p <- c(1:5)
```

Calculate with the equation (sum of all the elements divided by the number of elements in the vector):

```
1 sum(p)/length(p)
```

```
[1] 3
```

Or, use R's built-in function:

```
1 mean(p)
```

```
[1] 3
```

The median

The median is defined as the middle value in a list of sorted numerical values.

Steps to calculating the median:

1. Order numbers
2. Exclude the first, then last, then first, then last (etc) numbers until you reach a single middle number.
3. If two middle numbers, take their means

The median

What is the median of this vector?

```
1 y <- sample(1:20, 5, replace = TRUE)
2 y
```

```
[1] 19 16  8 10  1
```

First, arrange in order:

```
1 sort(y)
```

```
[1]  1  8 10 16 19
```

Or, use R's built in function:

```
1 median(y)
```

```
[1] 10
```

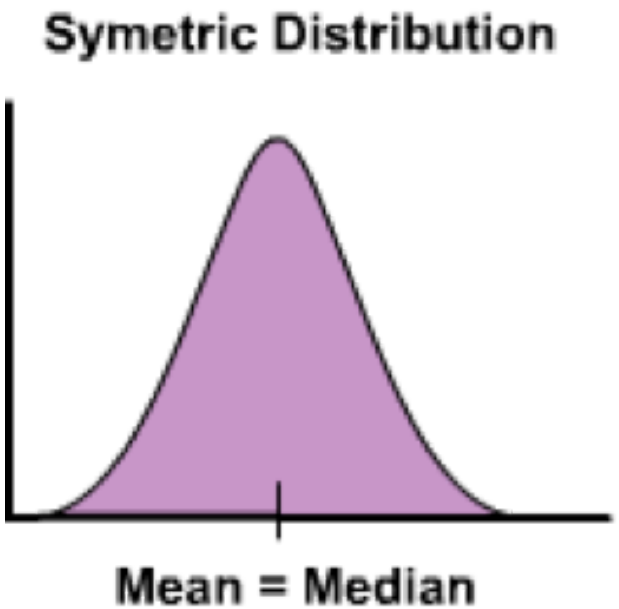
The median

The median is the second most commonly used statistic for describing the central tendency of a dataset.

It is more robust to extreme values, so it is preferred over the mean for datasets that either are skewed or are relatively small

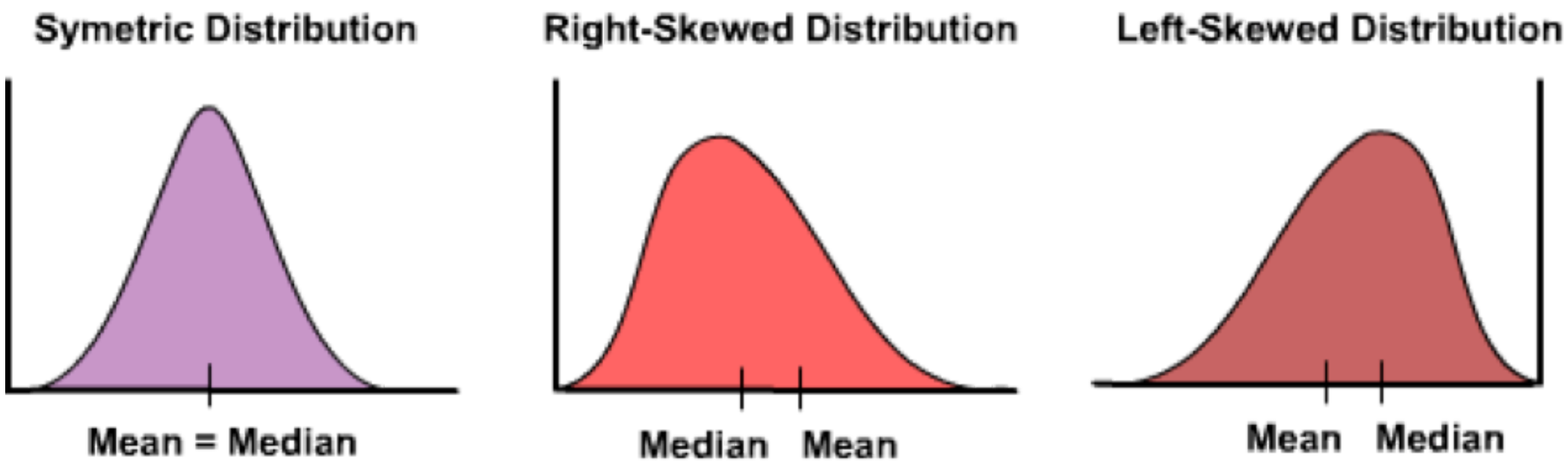
In practice, the mean is more commonly used.

Skewed distributions



Symmetric distributions (not skewed) are those where the distribution is roughly the same on each side of the most common value

Skewed distributions



Skewed distributions are those where values are more common on one side of the most common value and tend to have a longer maximum extension (tail)

The mean versus the median

What happens to our mean and medians when we add new values?

```
1 vector1 <- c( 4, 6, 7, 4, 6, 8, 5)
2 vector2 <- c( 4, 6, 7, 4, 6, 8, 5, 12, 10)
```

```
1 median(vector1)
```

```
[1] 6
```

```
1 median(vector2)
```

```
[1] 6
```

```
1 mean(vector1)
```

```
[1] 5.714286
```

```
1 mean(vector2)
```

```
[1] 6.888889
```

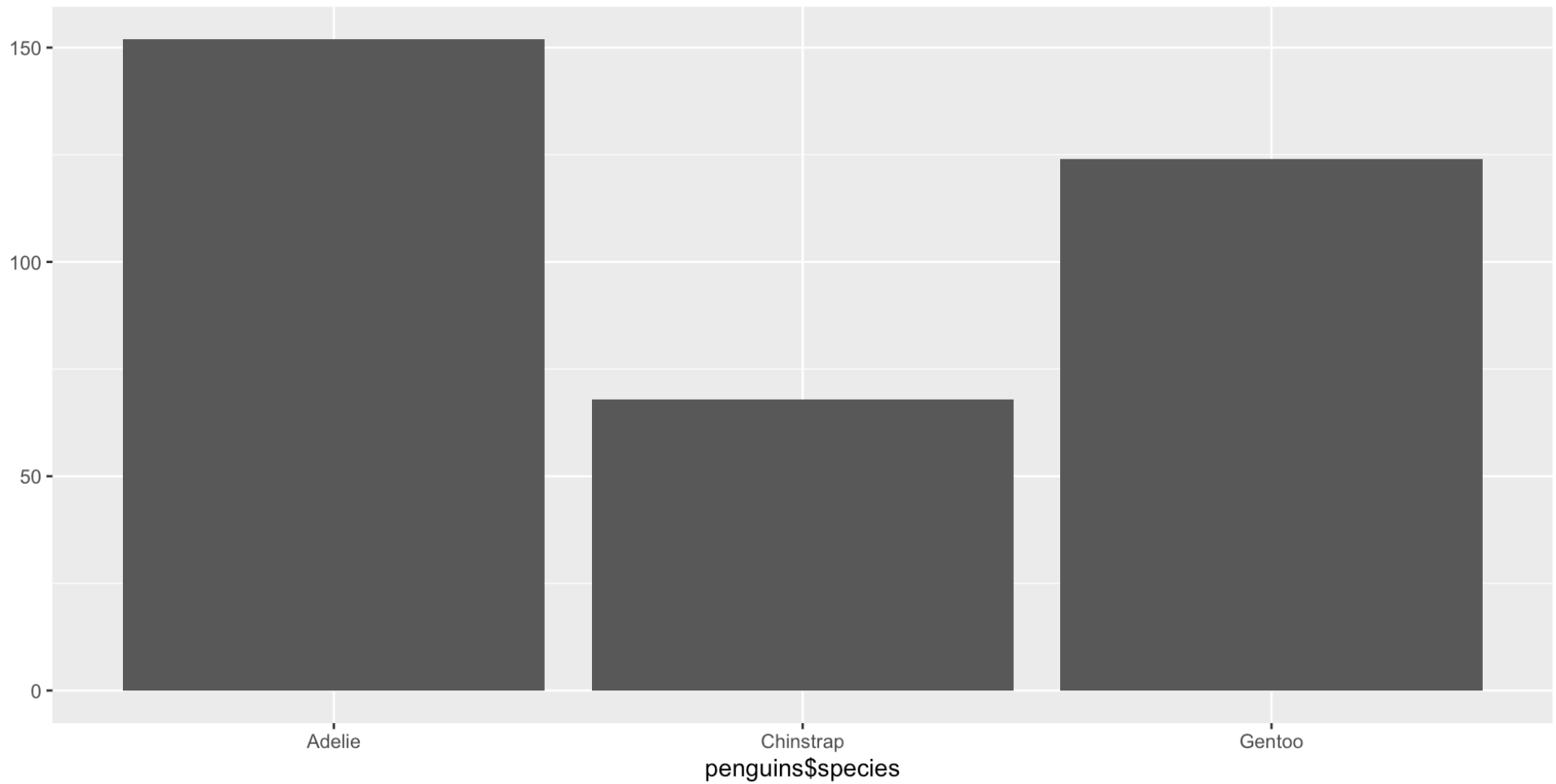
The mode

The mode is the most common value in a group of scores. It is typically determined by using a frequency table or a bar plot.

In practice, the mode is not commonly used for numerical data, especially for continuous data. It is more commonly used for categorical data.

The mode

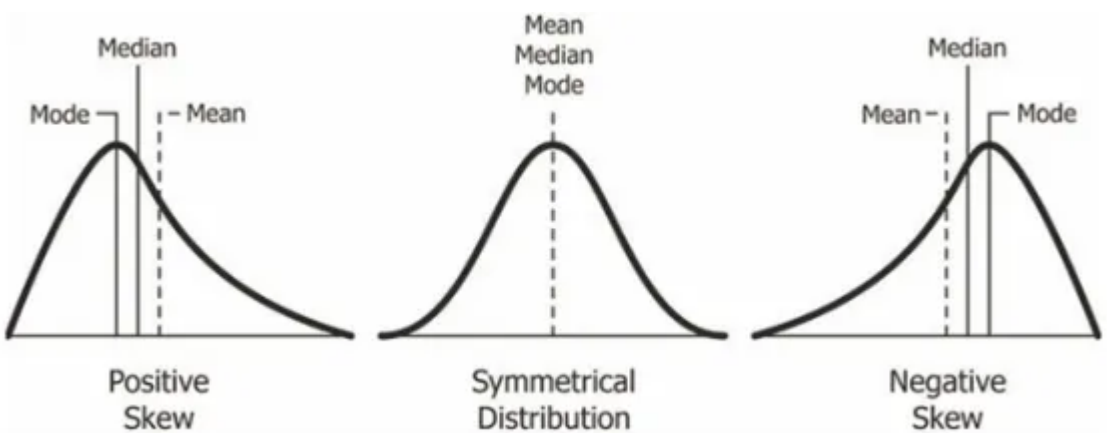
```
1 penguins <- penguins
2 qplot(penguins$species)
```



The mean, median and mode

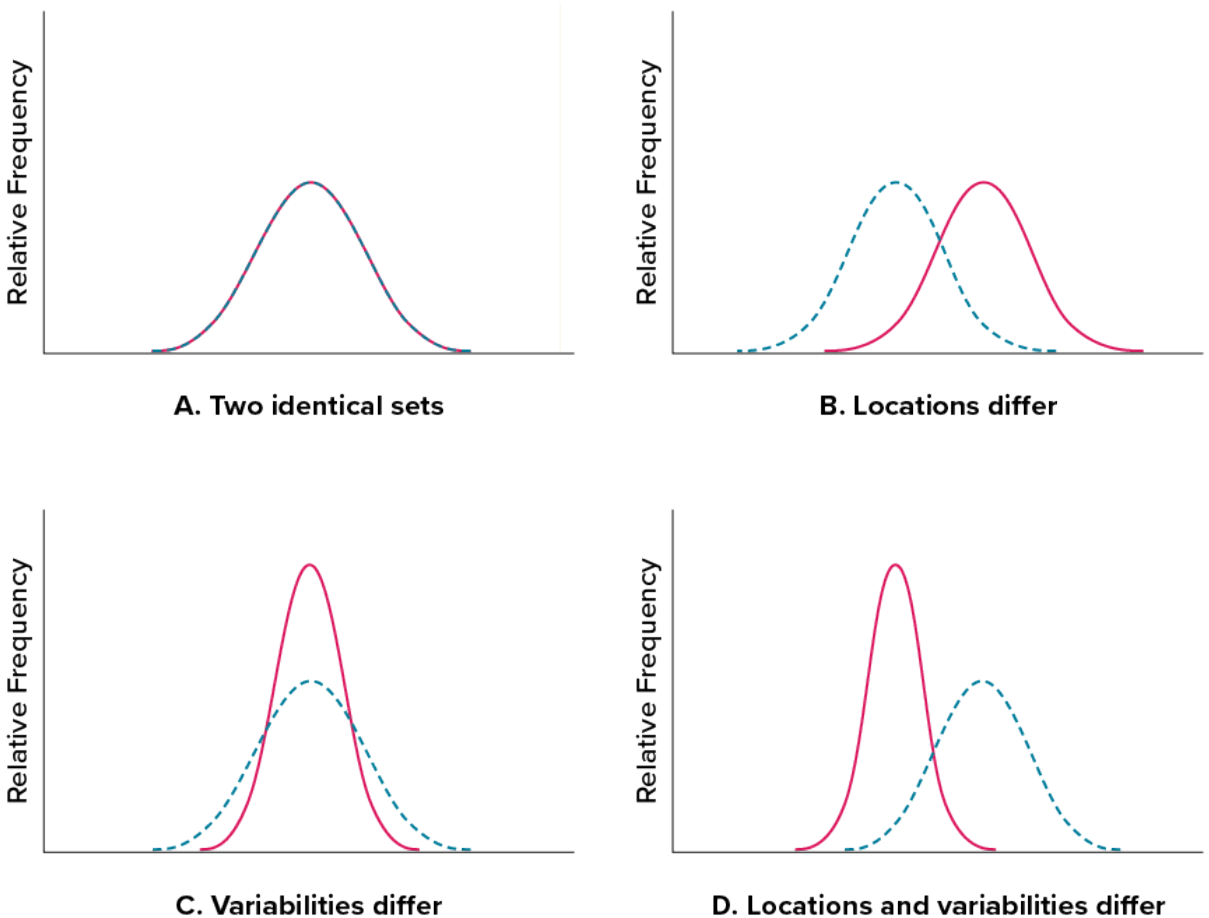
In perfectly symmetrical distributions, the mean, median, and mode are all the same. The difference between the mean and the median can be informative of the shape of the distribution

If the mean is greater than the median, then outlying values are likely to be large or the distribution has a long right tail (right skewed).



Measures of dispersion

Statistics that describe variability around a central tendency value.



Measures of dispersion

These come in a lot of flavours, e.g.:

1. Range
2. Variance
3. Standard deviation
4. Interquartile range (IQR)

The range

The range is a very simple metric that summarizes the maximal spread of the distribution, i.e. its smallest and largest value

```
penguins$bill_length_mm[c(1, 5, 29, 32, 57)] %>% sort()
```

```
[1] 36.7 37.2 37.9 39.0 39.1
```

```
max(penguins$bill_length_mm[c(1, 5, 29, 32, 57)])
```

```
[1] 39.1
```

```
min(penguins$bill_length_mm[c(1, 5, 29, 32, 57)])
```

```
[1] 36.7
```

The variance

The variance measures the spread of scores around the mean, and it is defined as the average of squared differences from the mean.

$$\sigma^2 = \frac{\sum (xi - \bar{x})^2}{N}$$

The variance

Steps for calculating the variance:

1. Subtract the mean from each score
2. Square each resulting deviation score
3. Add up the squared deviations
4. Divide this by the number of scores

$$\sigma^2 = \frac{\sum(xi - \bar{x})^2}{N}$$

The variance

In practice, the variance is not frequency given as a summary statistic because it's a squared value

That means it is not proportional to the mean, so it can be hard to interpret relative as a measure of dispersion

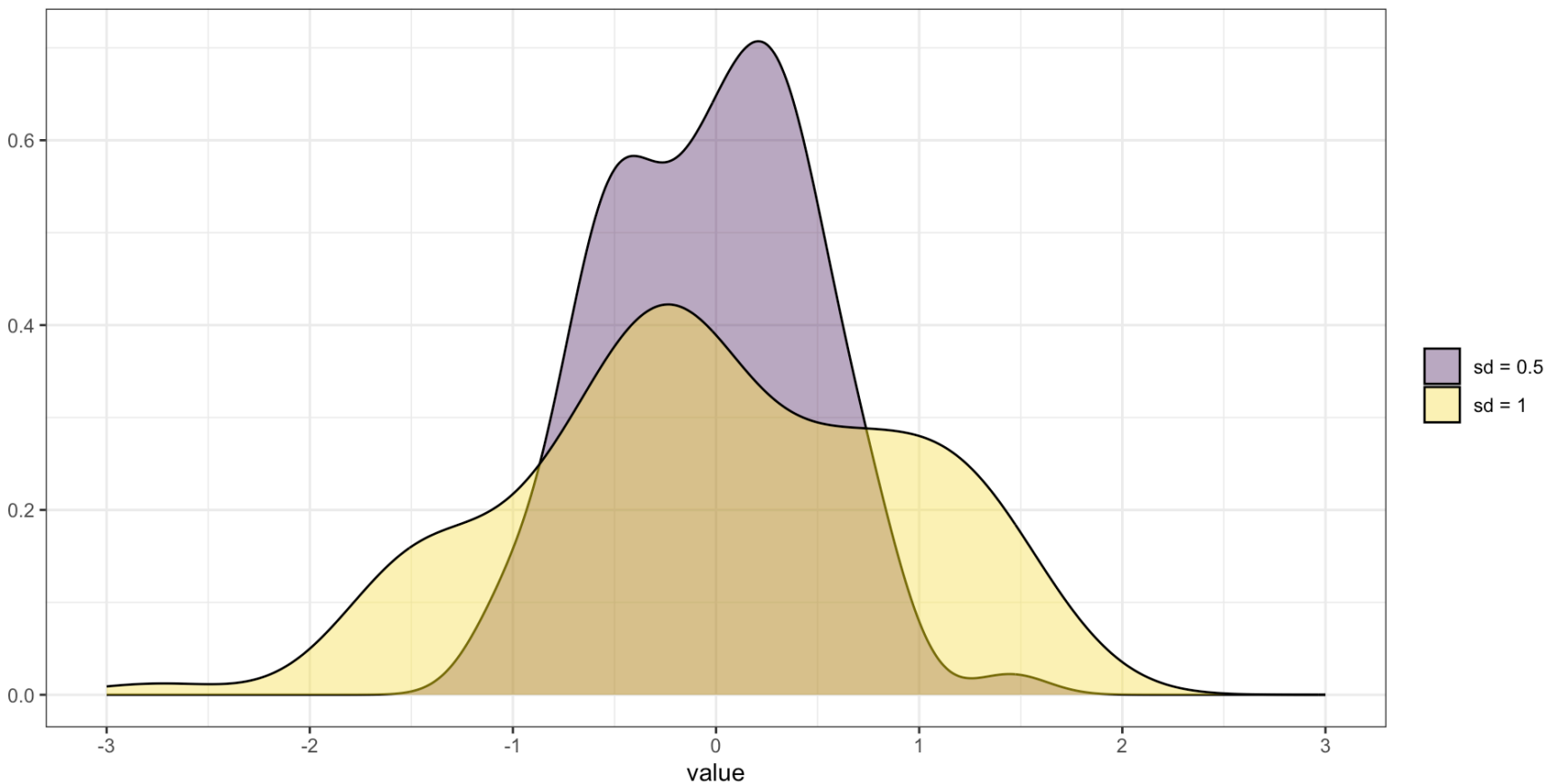
The standard deviation

The standard deviation is the most commonly used measure of spread. It is defined as the square root of the variance

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

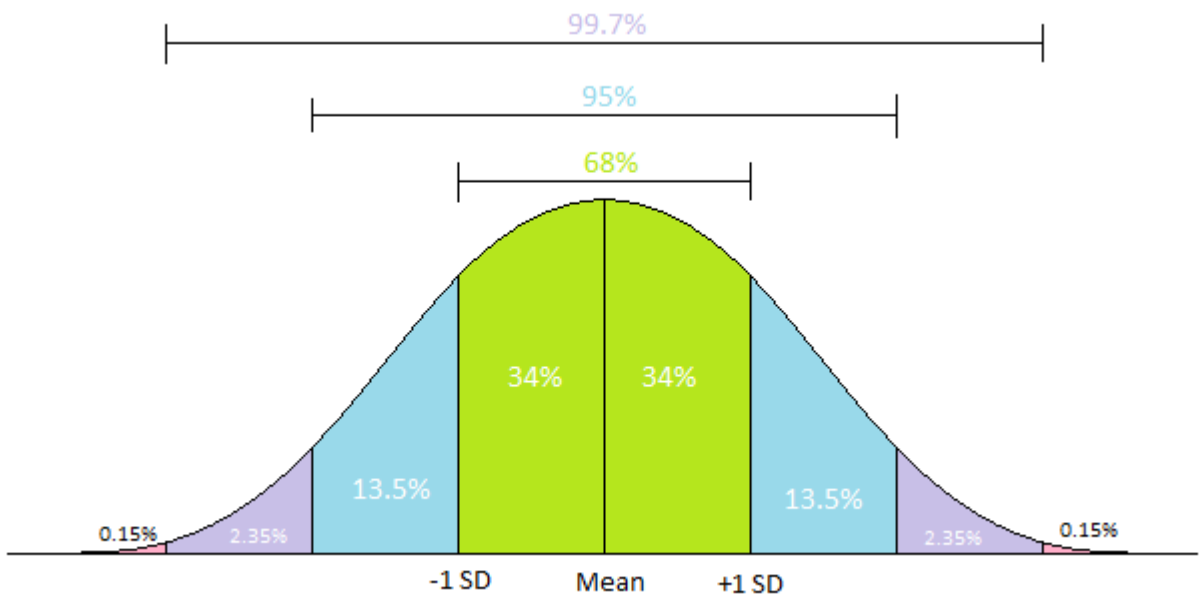
The standard deviation

As the standard deviation increases, so does spread of the data, and the distribution “flattens”



The standard deviation

By definition, the standard deviation is *more or less* the average amount by which the scores deviate from the mean



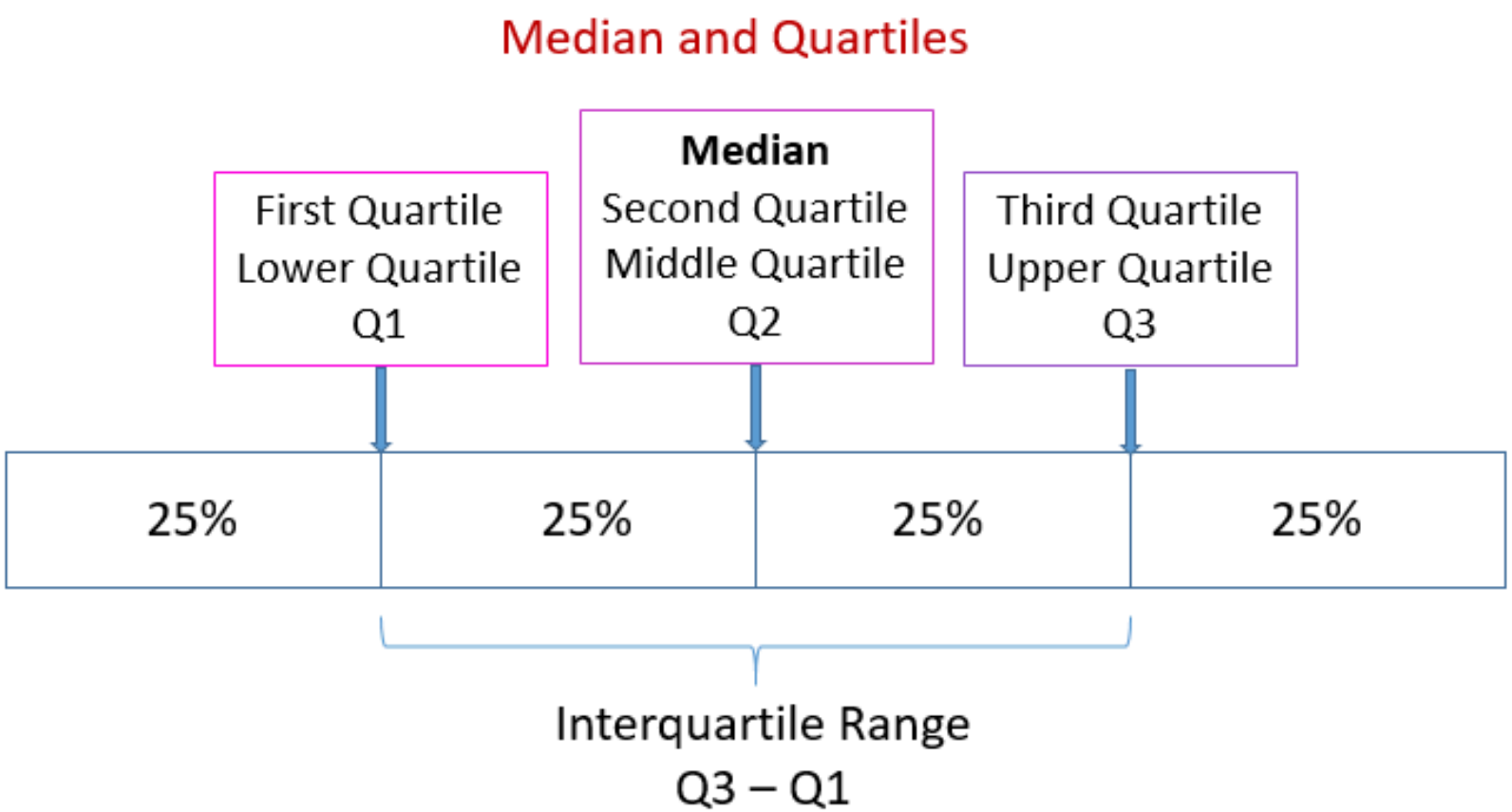
This only holds true if the distribution of the variable approximates a normal

The interquartile range

The interquartile range represents the values at the first and third quartile of the distribution, or in other words, more or less the 25th and 75th centiles

The interquartile range

The IQR is typically reported with the median.



The interquartile range

In fact, the IQR is reported alongside the median in the boxplot

What measures of central tendency and spread to report?

You should always report one of each.

Typically, researchers report the mean and the standard deviation

In some situations, for example with a skewed distribution or a small sample size, you may want to report a median, in which case you report the IQR alongside it.

Descriptive statistics for categorical data

Frequency and proportion tables

When summarizing categorical data, numerical summaries don’t make much sense

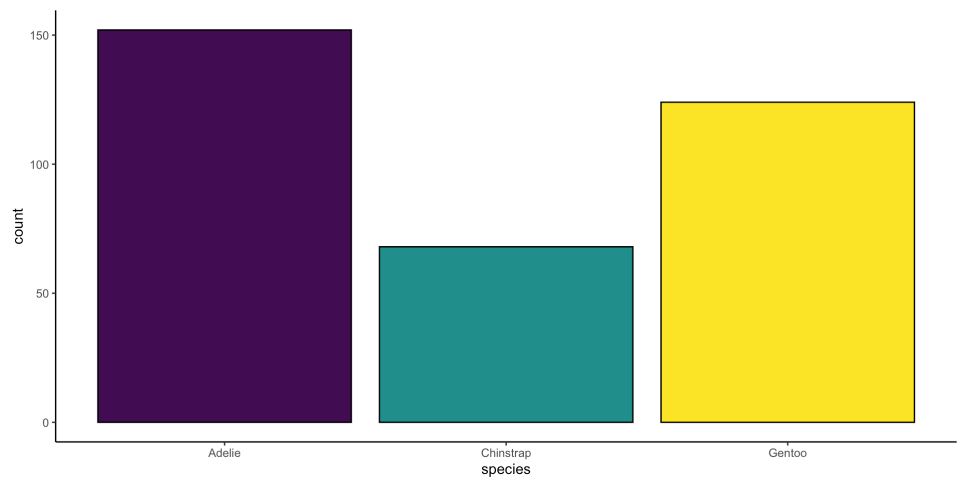
Instead, we summarize them using frequency and proportion tables

Frequency tables

Frequency tables summarize counts. They are quite similar to bar plots, but in a numerical form.

```
1  table(penguins$species)
```

```
   Adelie Chinstrap    Gentoo
      152        68       124
```



Frequency tables

You can also summarize by multiple variables, e.g.

```
1 table(penguins$species, penguins$island)
```

```
           Biscoe Dream Torgersen
Adelie         44    56        52
Chinstrap       0    68         0
Gentoo        124     0         0
```

Proportion tables

Instead of counts, you can also summarize by proportion (or percentage).

```
1 proportions(table(penguins$species, penguins$island))
```

```
             Biscoe     Dream Torgersen
Adelie    0.1279070 0.1627907 0.1511628
Chinstrap 0.0000000 0.1976744 0.0000000
Gentoo    0.3604651 0.0000000 0.0000000
```

```
1 proportions(table(penguins$species, penguins$island))*100
```

```
             Biscoe    Dream Torgersen
Adelie    12.79070 16.27907  15.11628
Chinstrap  0.00000 19.76744   0.00000
Gentoo    36.04651  0.00000   0.00000
```

2. Manipulating data frames with `dplyr`

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 × 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
```

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"              "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"))
```

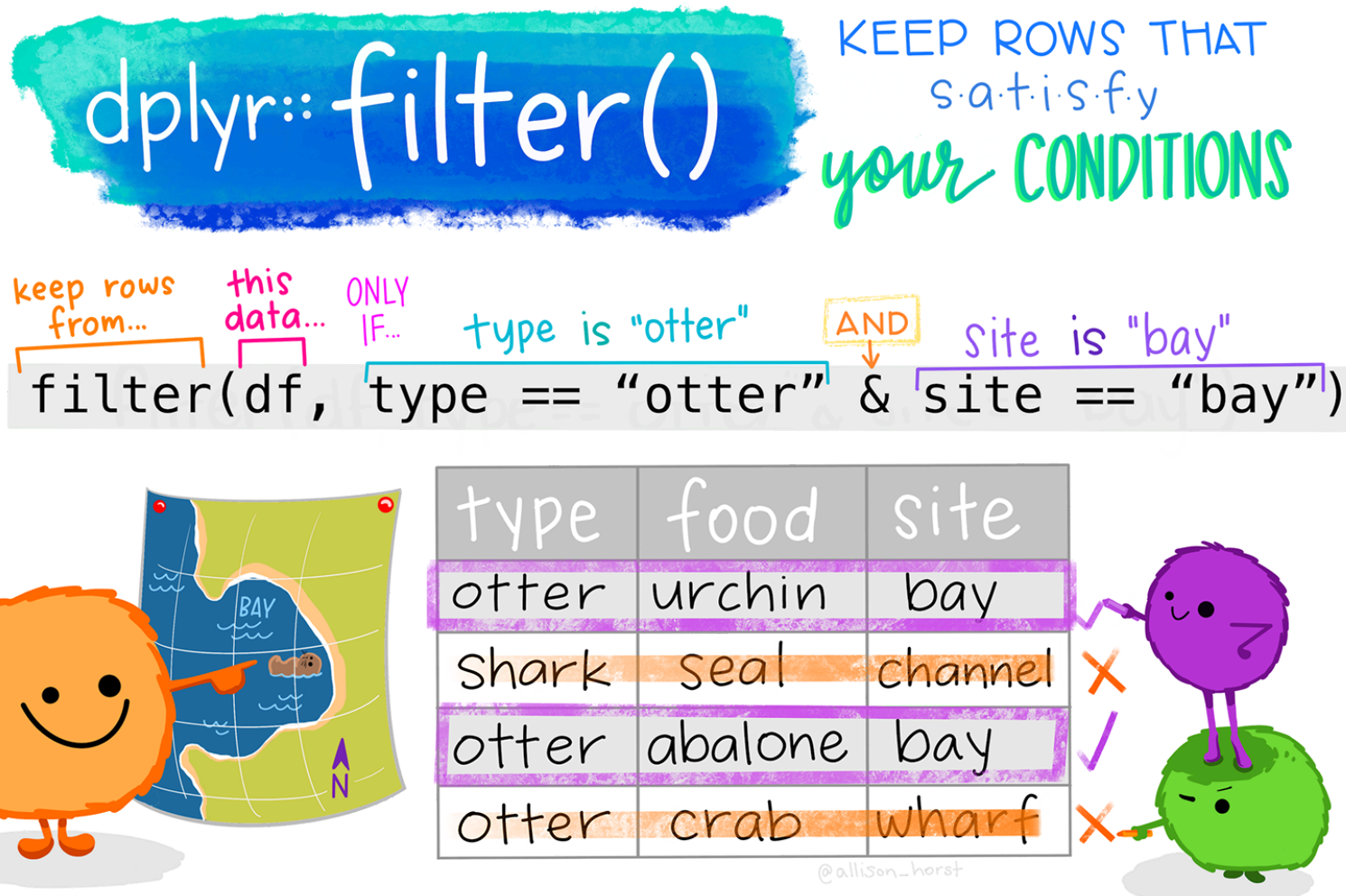
```
# A tibble: 344 × 6
   species island    body_mass_g sex      year bill_length_cm
   <fct>   <fct>           <int> <fct>   <int>          <dbl>
 1 Adelie  Torgersen        3750 male     2007           3.91
 2 Adelie  Torgersen        3800 female   2007           3.95
 3 Adelie  Torgersen        3250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

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 × 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
```

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 × 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
```

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 × 9
   species island    bill.l bill_depth_mm flip.l body_mass_g sex     year
   <fct>   <fct>      <dbl>         <dbl>  <int>       <int> <fct>  <int>
 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      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            20.2    190        4250 <NA>    2007
# ℹ 334 more rows
# ℹ 1 more variable: bill_length_cm <dbl>
```

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 × 9
   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 × 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
```

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:

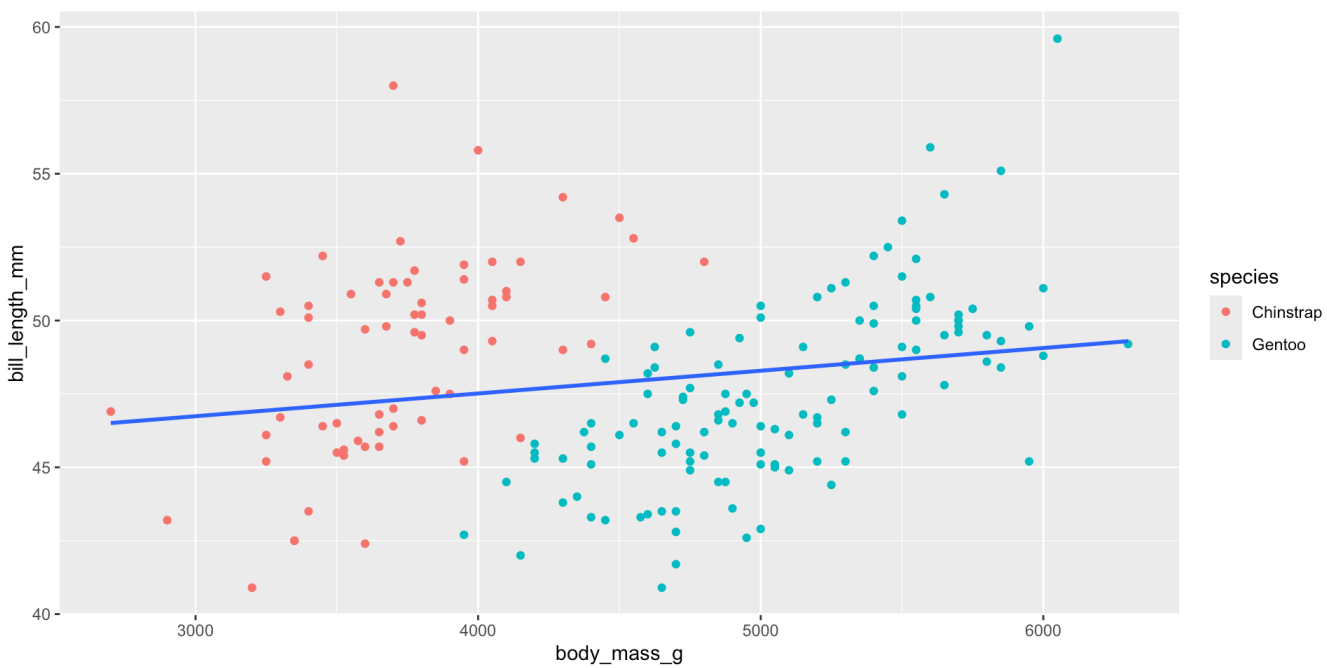
```
penguins %>%
  mutate(bill_length_cm = bill_length_mm/10)%>%
  filter(species == "Gentoo")%>%
  select(island, sex, bill_length_cm)
```

```
# 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`

```
penguins %>%
  filter(species %in% c("Gentoo", "Chinstrap"))%>%
  ggplot(aes(x = body_mass_g, y = bill_length_mm))+ # note we use +
    geom_point(aes(col = species))+
    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:

```r
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
```

3. Statistical basics

The normal curve

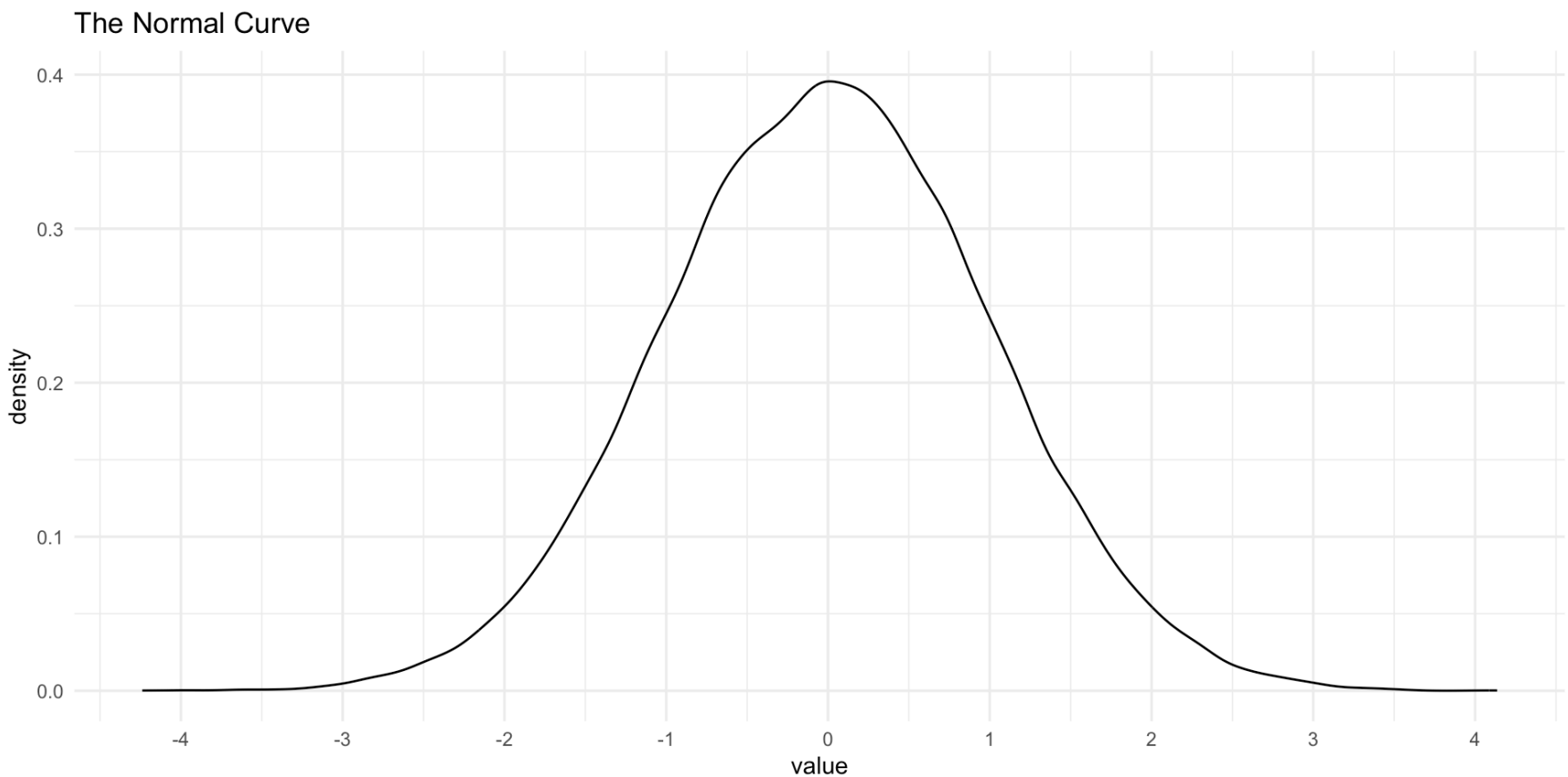
The normal curve is a theoretical distribution which we often use as the model for many of our variables.

Its features:

- most of the scores are near the middle
- distribution is symmetrical about the mean
- the mean, mode, and median are the same
- it is described by mean = 0 and sd = 1

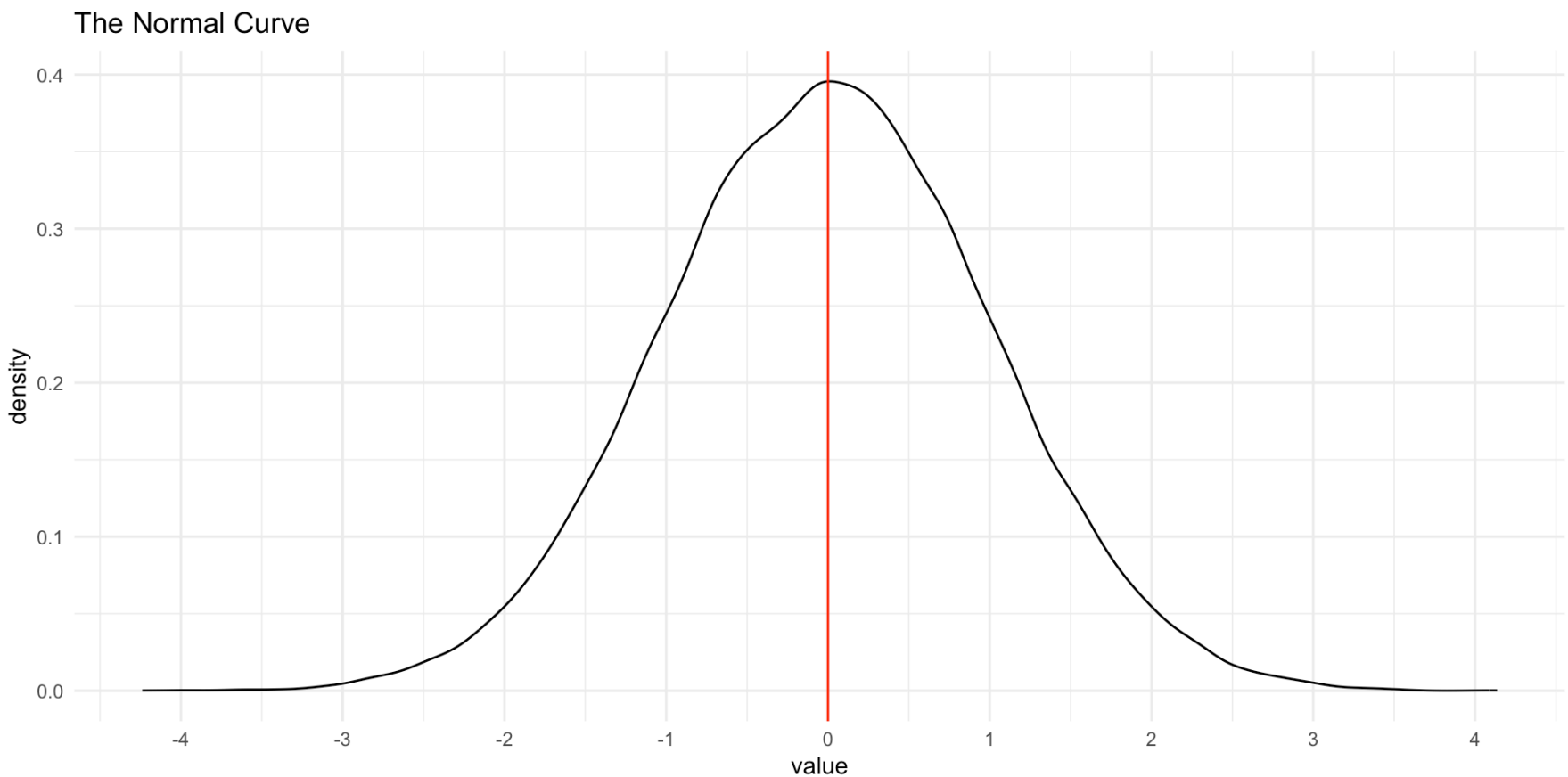
The normal curve

Here is what the normal curve looks like - the data producing these results is simulated.



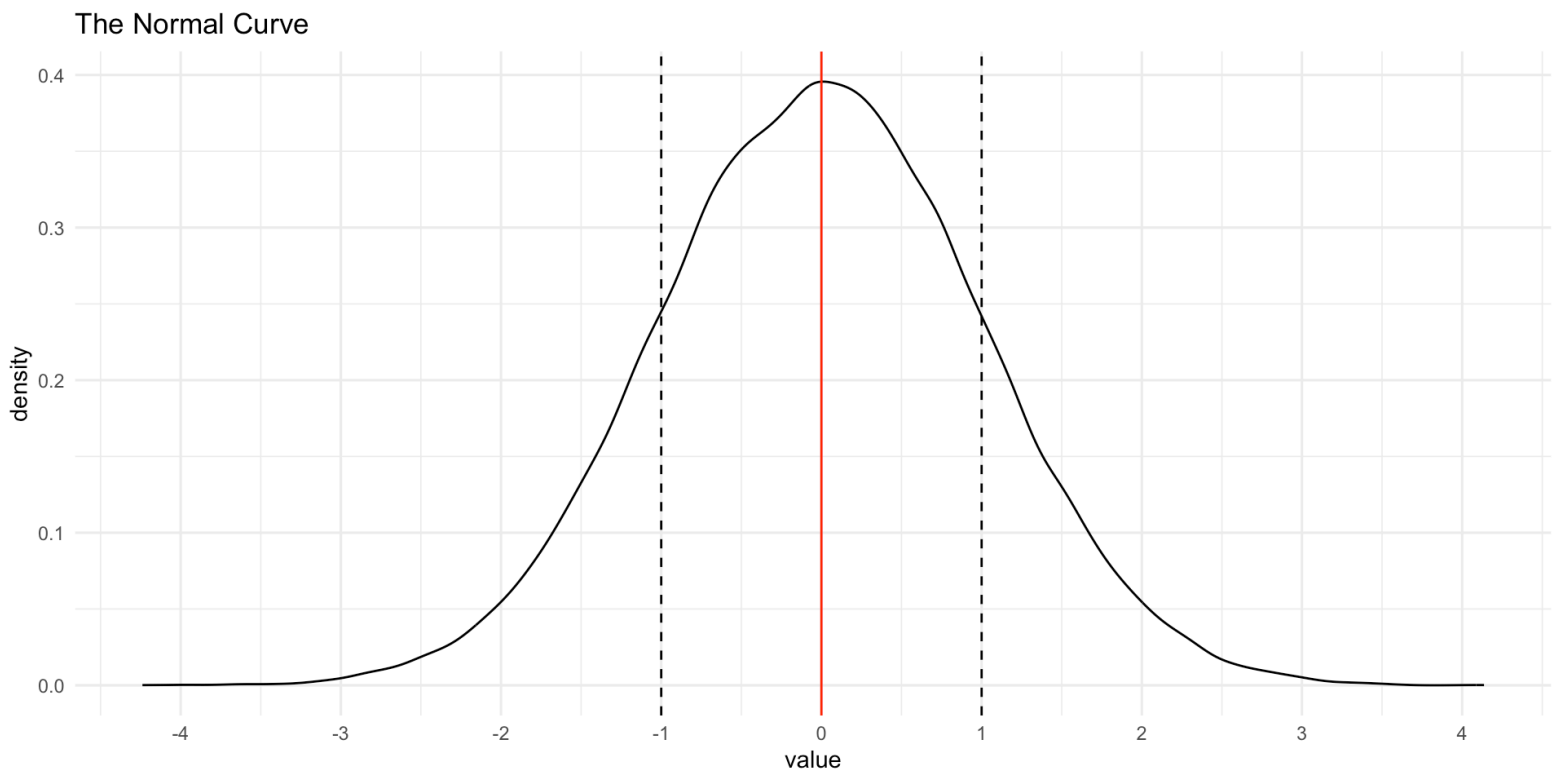
The normal curve

Here is what the normal curve looks like - the data producing these results is simulated.



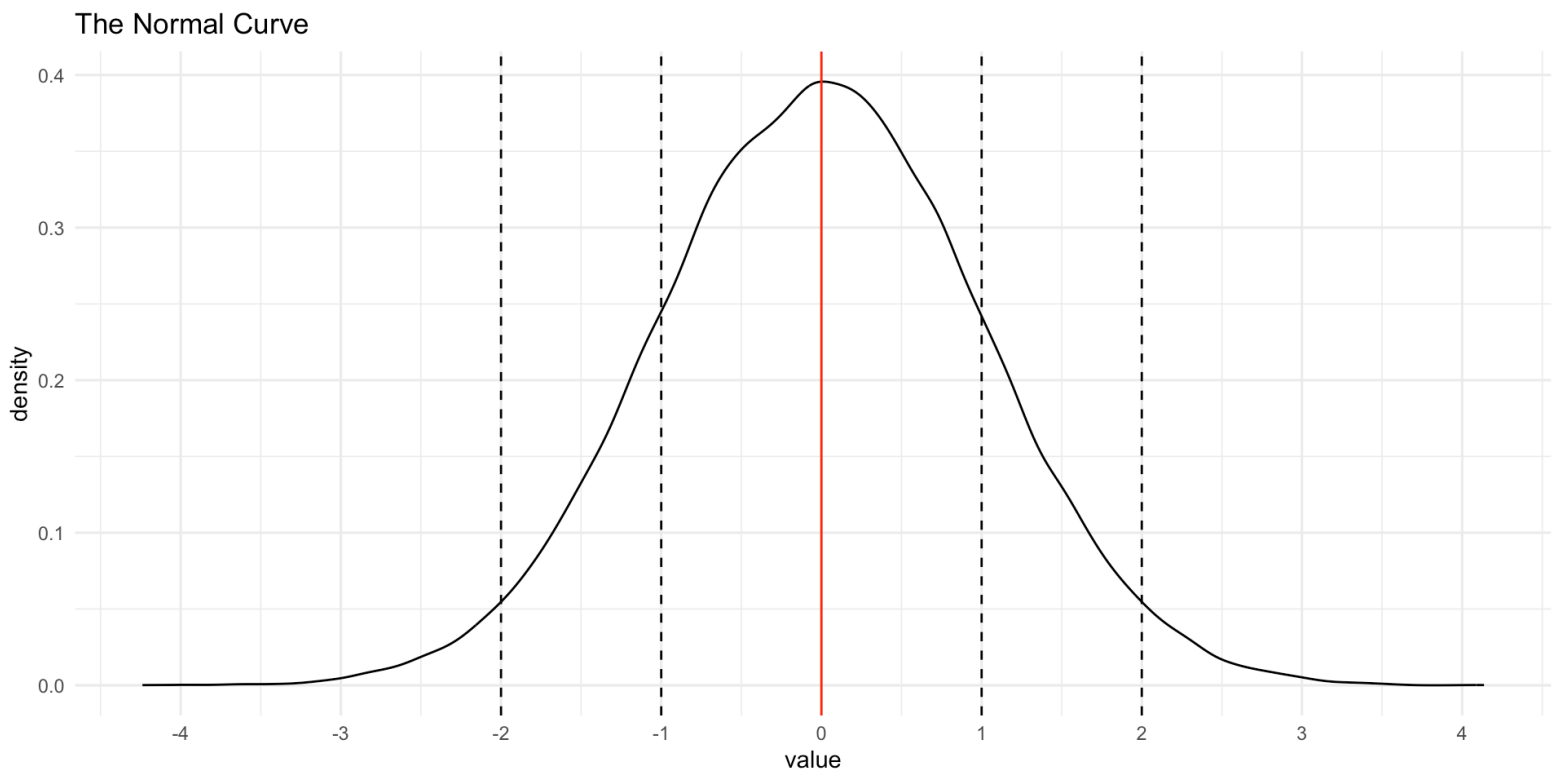
The normal curve

The first standard deviation from the mean covers ~34% of the data on either side, for a total of 68% of the data in the range from -1 sd to +1 sd



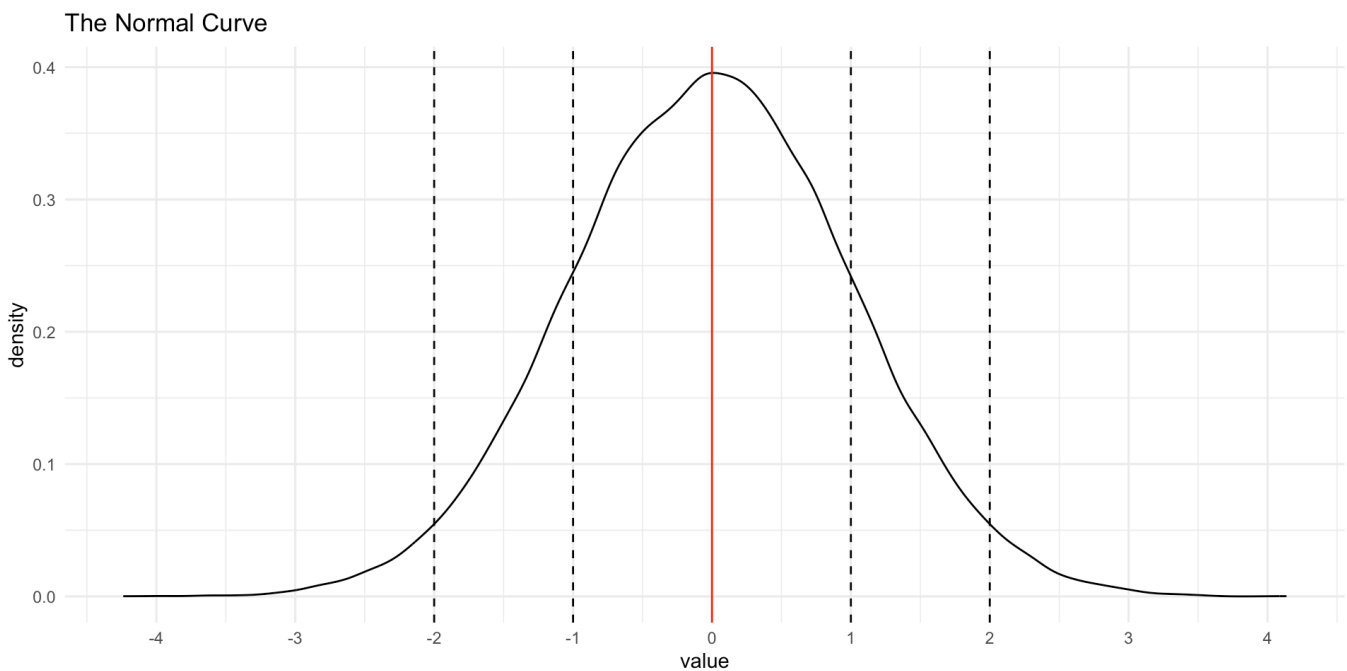
The normal curve

The second standard deviation from the mean covers an additional ~13.5% of the data on either side, for a total of approx. 95% of the data in the range from -2 sd to +2 sd



The normal curve

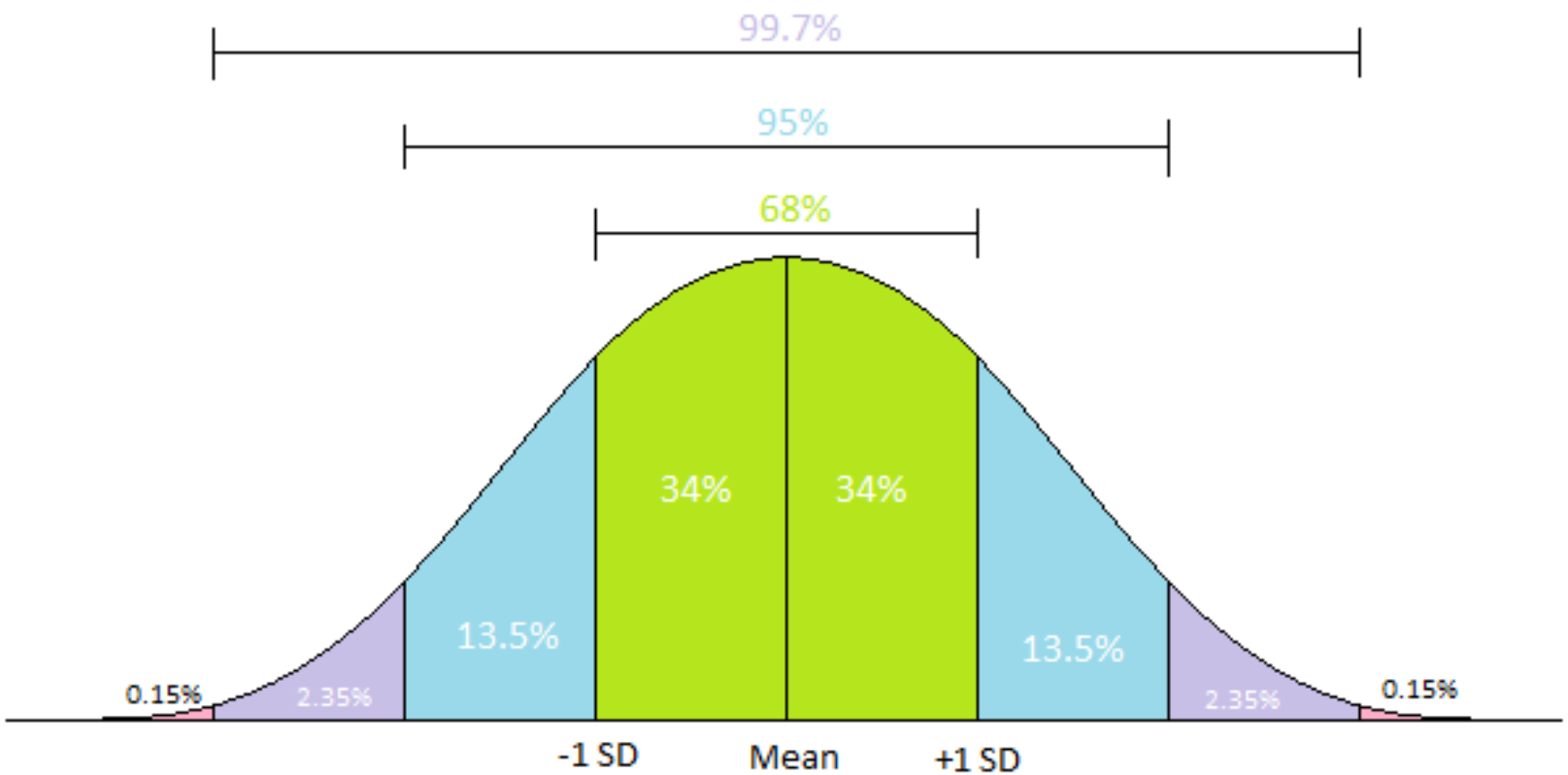
The third standard deviation from the mean covers an additional ~2% of the data on either side, for a total of approx. 99.7 % of the data in the range from -3 sd to +3 sd. 0.15% of the data are greater than 3 sd away from the mean.



The normal curve

Coverage of the normal curve by standard deviations

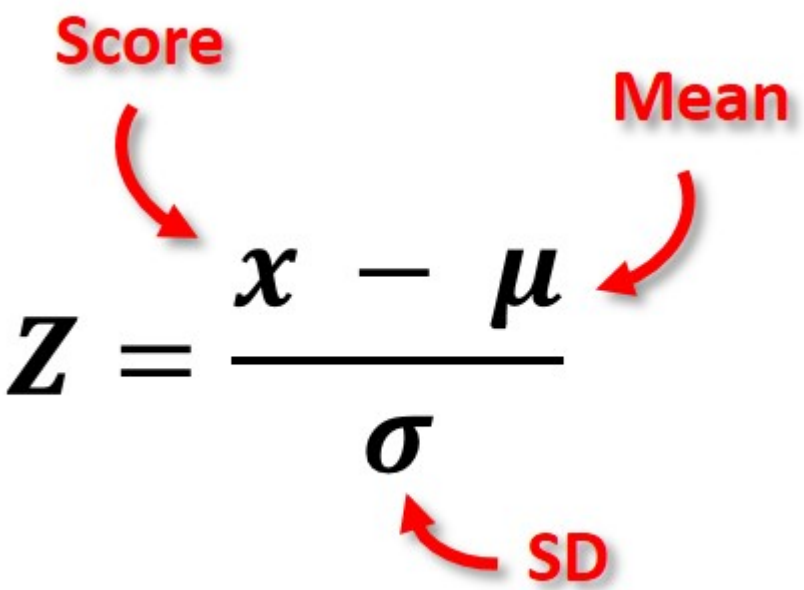
Hint: This is important, commit this to memory.



Z-scores: deviation for a single individual

The z-score quantifies the standardized deviation from the mean.

It is calculated as the difference between the score and the mean, divided by the standard deviation



Z-scores: deviation for a single individual

Things to know about z-scores:

- They express deviation from the mean in terms of standard deviations (they are standardized)
- You can calculate the percentage of the distribution that falls above/below a z-score or between two z-scores

Z-scores: deviation for a single individual

For example, if you had someone whose height fell at the z-score 2.10, you could calculate how tall they are relative to the rest of the population

```
1 pnorm(2.10, mean = 0, sd = 1)
```

```
[1] 0.9821356
```

Z-scores: deviation for a single individual

For example, if you had someone whose height fell at the z-score 2.10, you could calculate how tall they are relative to the rest of the population

```
pnorm(2.10, mean = 0, sd = 1)
```

```
[1] 0.9821356
```

We can also generate the likelihood of finding someone that tall, or taller:

```
1 - pnorm(2.10, mean = 0, sd = 1)
```

```
[1] 0.01786442
```

```
pnorm(2.10, mean = 0, sd = 1, lower.tail = FALSE)
```

```
[1] 0.01786442
```

The normal distribution

The normal distribution is what makes statistics so powerful.

We know that our samples are not perfectly normal, but if they approximate the normal distribution closely enough, we can use the normal distribution to study our samples.

The normal distribution

You can think of the normal distribution almost like a probability distribution.

The probability of a score landing between the mean and +1 sd is 0.34 or 34%

```
mean <- pnorm(0, mean = 0, sd = 1)
plusone <- pnorm(1, mean = 0, sd = 1)
```

The normal distribution

You can think of the normal distribution almost like a probability distribution.

The probability of a score landing between the mean and +1 sd is 0.34 or 34%

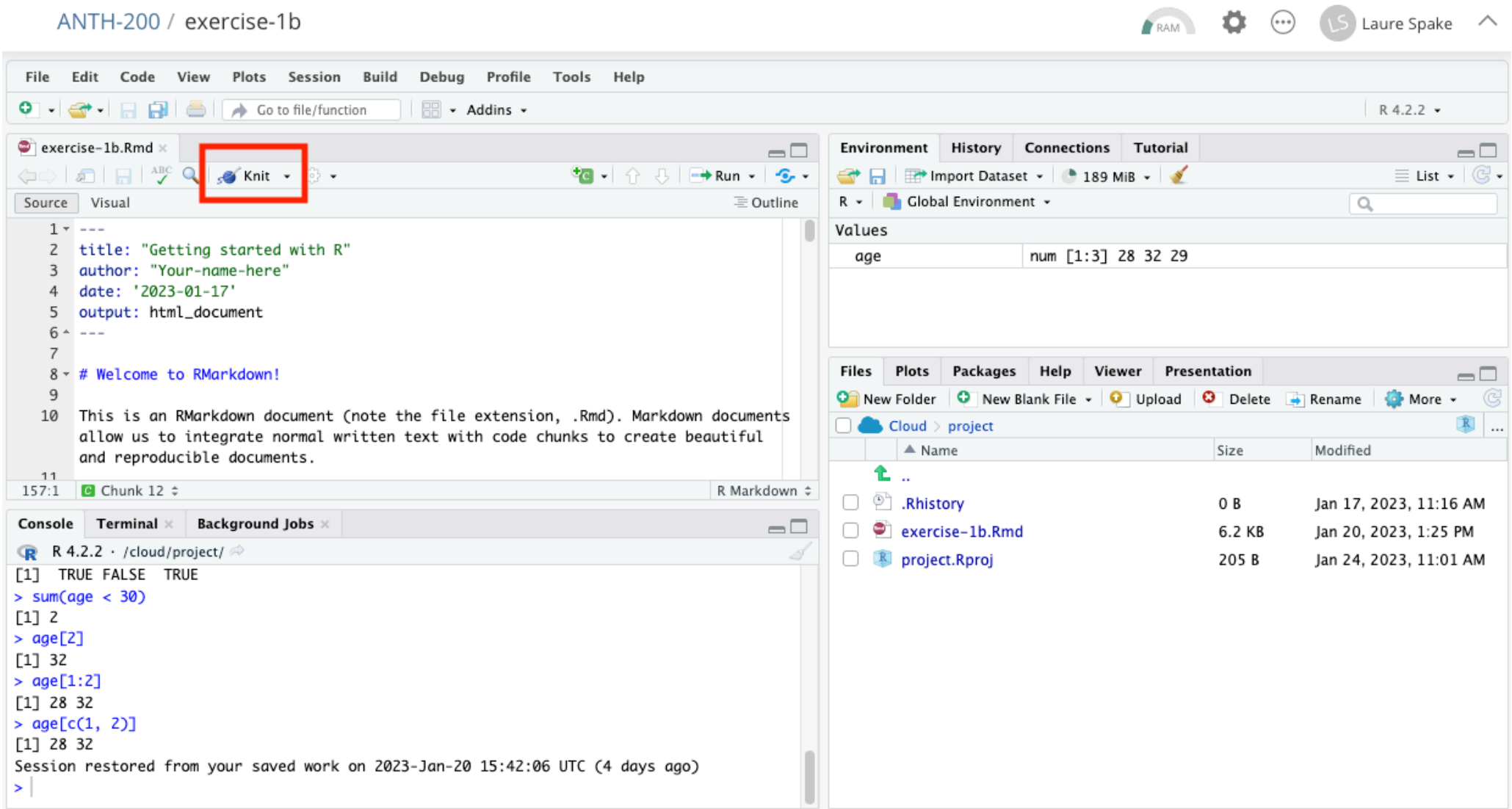
```
mean <- pnorm(0, mean = 0, sd = 1)
plusone <- pnorm(1, mean = 0, sd = 1)
```

```
plusone - mean
```

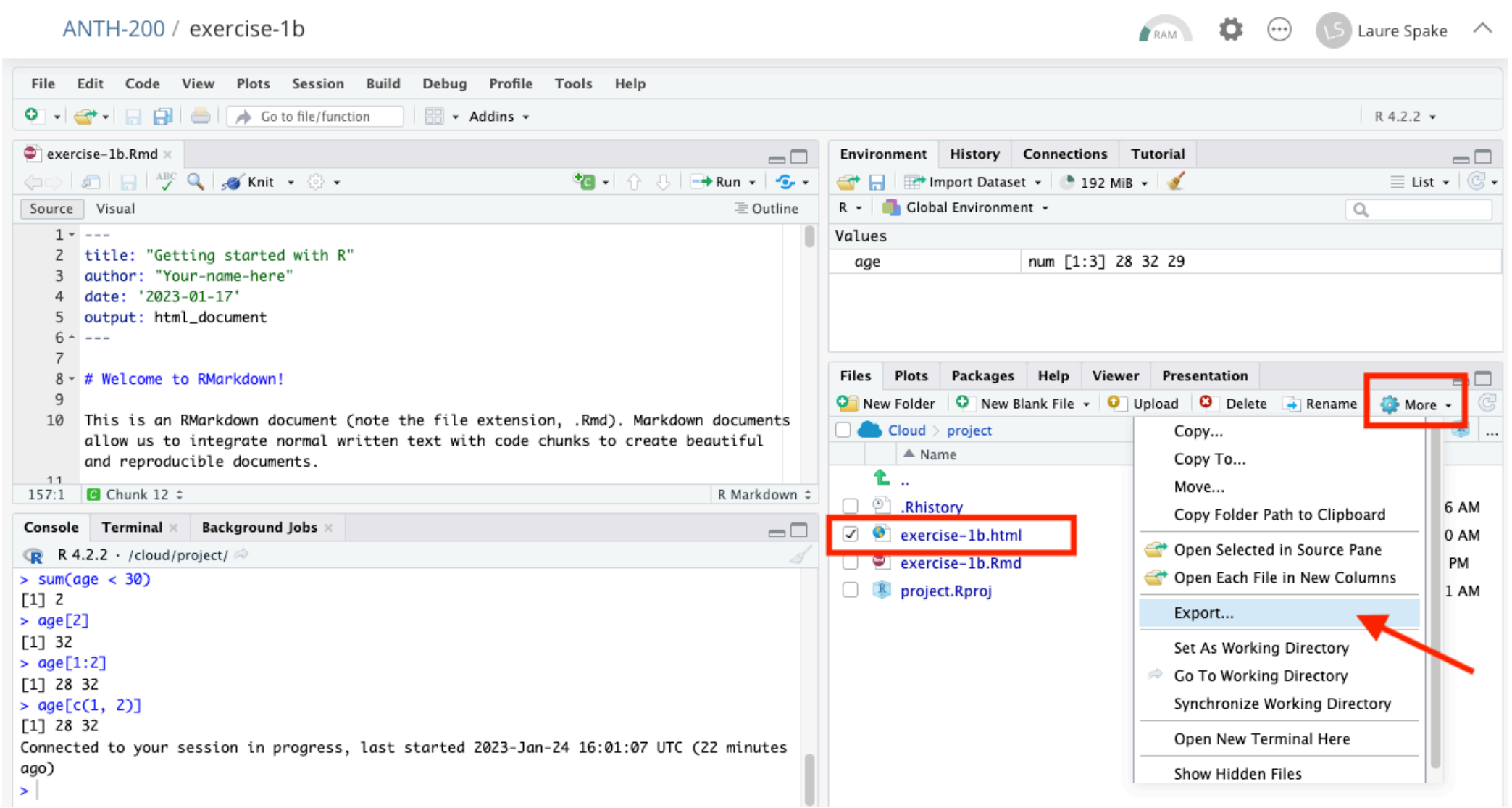
```
[1] 0.3413447
```

Quick note - submitting your assignments

Creating a knitted html document



Creating a knitted html document



Your turn: Posit Cloud

Summary

Learning objectives

In this lesson you:

- Learned how to produce summary statistics
- Learned how to wrangle dataframes
- Refreshed your understanding of the normal distribution

Attribution

- The `pivot` section of this lecture is inspired by: R4DS
- Also inspired by Data Carpentries' R for Ecology Course