Data wrangling with `dplyr`

Week 4 (10/22/25)

Artwork by Allison Horst

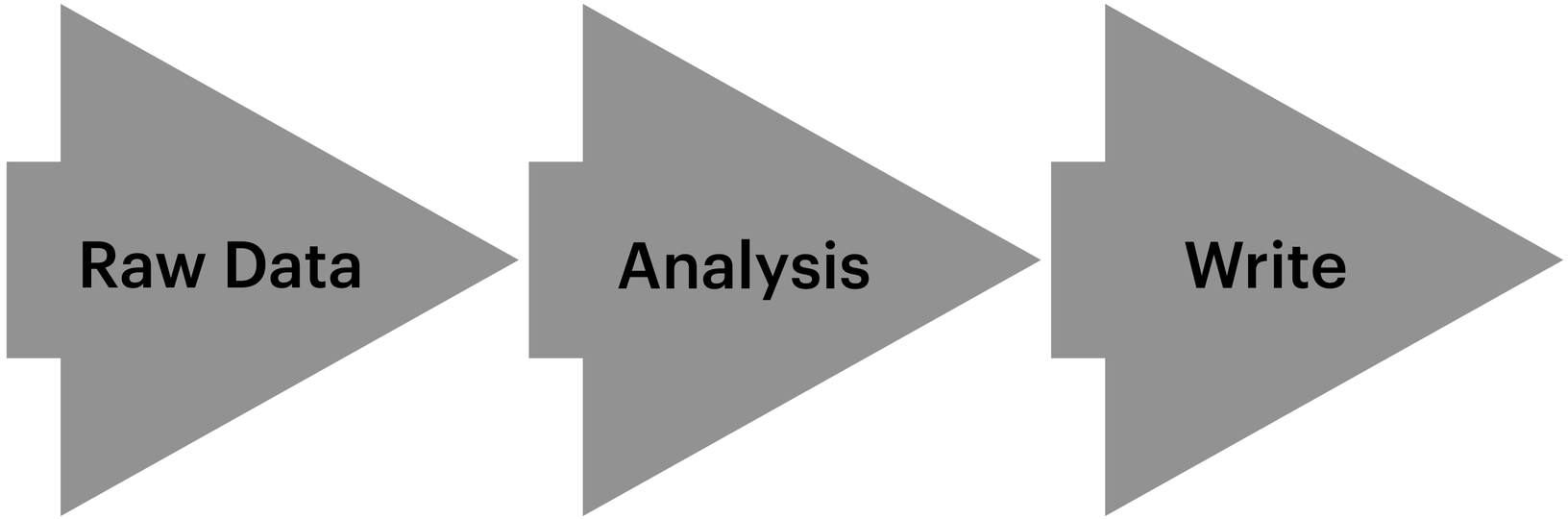
Stepfanie M. Aguillon

Outline of today's class

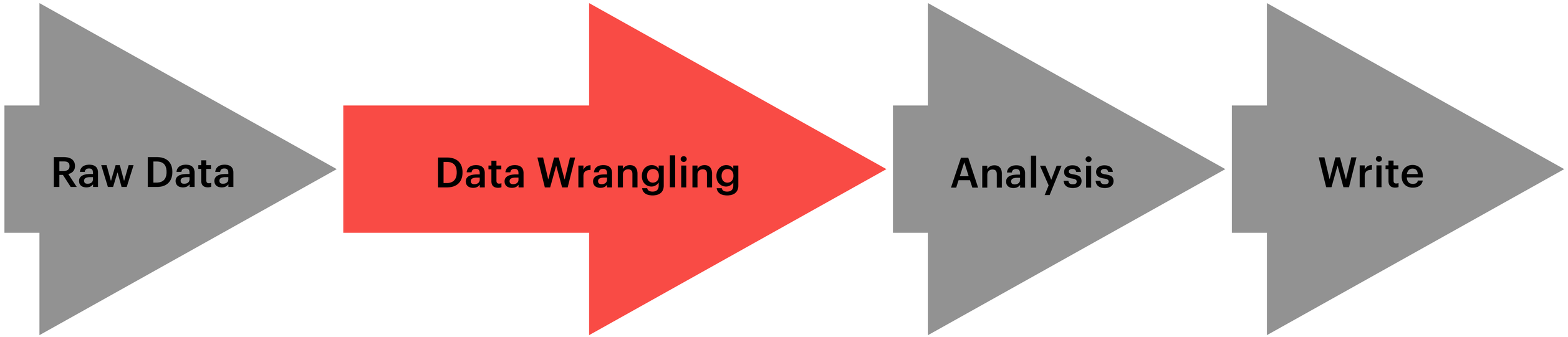
- What is data wrangling?
- Introduction to `dplyr`
- Continue practicing with RStudio/GitHub and Quarto

What is data wrangling?

The research pipeline

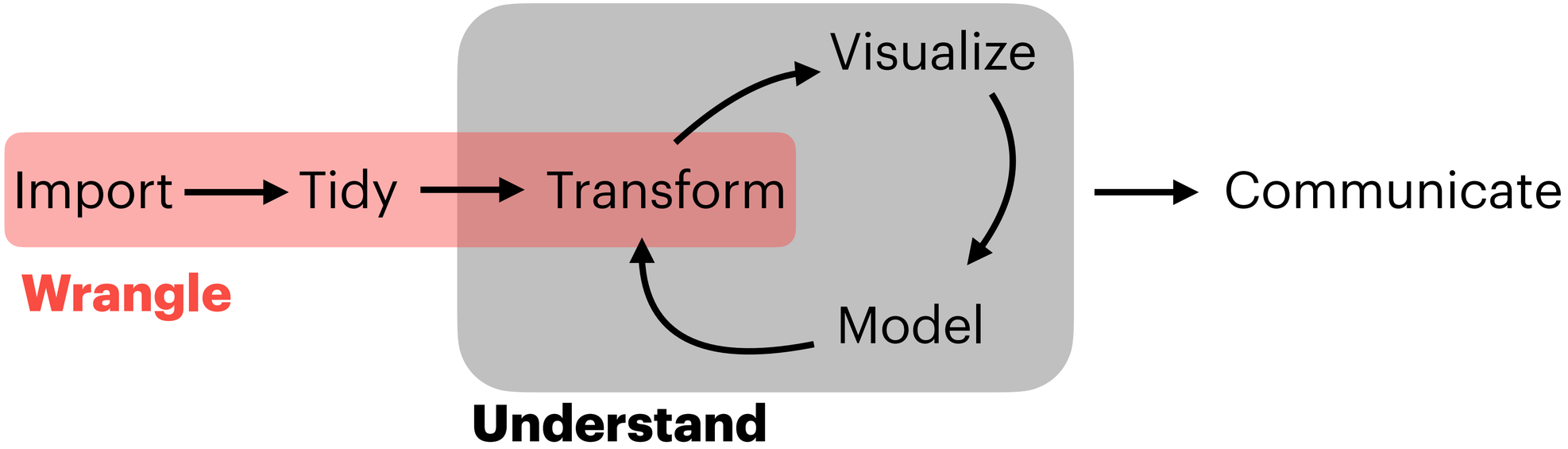


The research pipeline



Data wrangling = Getting your data into a useable form for downstream analyses and data visualization

Data wrangling



https://r4ds.had.co.nz/wrangle-intro.html

Introduction to `dplyr`

`tidyverse` package
for manipulating data

dplyr Cheatsheet

Data transformation with dplyr : : **CHEATSHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

pipes

x |> f(y) becomes **f(x, y)**

Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarize(.data, …**)** Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

Alternate grouping syntax with **.by** as an argument:

```
mtcars |>
  summarize(
    avg = mean(mpg), .by = cyl
  )
```

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars |>
  rowwise() |>
  mutate(film_count = length(films))
```

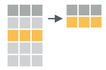
ungroup(x, …**)** Returns ungrouped copy of table.

```
g_mtcars <- mtcars |> group_by(cyl)
ungroup(g_mtcars)
```

Manipulate Cases

EXTRACT CASES

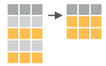
Row functions return a subset of rows as a new table.



filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)

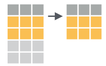
distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
mtcars |> distinct(gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
mtcars |> slice(10:15)



slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)



slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
mtcars |> arrange(mpg)
mtcars |> arrange(desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL**)** Add one or more rows to a table.
cars |> add_row(speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
mtcars |> pull(wt)

select(.data, …**)** Extract columns as a table.
mtcars |> select(mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. mtcars |> select(mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g., mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	**!**, e.g., !gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

across(.cols, .funs, …, .names = NULL**)** Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

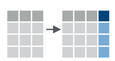
c_across(.cols**)** Compute across columns in row-wise data.

```
df |>
  rowwise() |>
  mutate(x_total = sum(c_across(1:2)))
```

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function



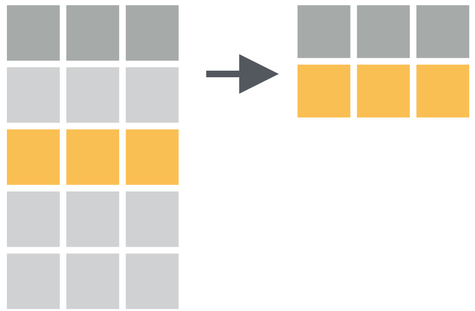
mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column()**.
mtcars |> mutate(gpm = 1 / mpg)
mtcars |> mutate(gpm = 1 / mpg, .keep = "none")

rename(.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
mtcars |> rename(miles_per_gallon = mpg)

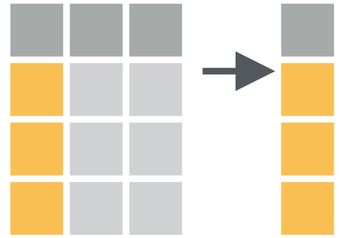
https://rstudio.github.io/cheatsheets/data-transformation.pdf

Four important `dplyr` functions

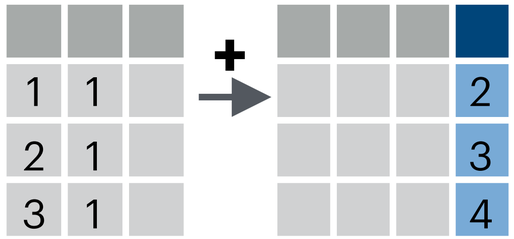
1. `filter()`



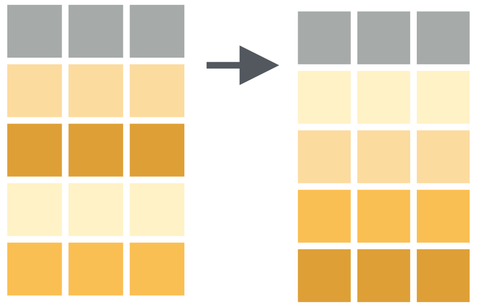
2. `select()`



3. `mutate()`



4. `arrange()`



Four important `dplyr` functions

1. `filter()`

filter observations by their values

2. `select()`

select variables by their column names

3. `mutate()`

mutate existing variables to create new ones

4. `arrange()`

arrange observations by their values

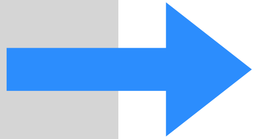
“The Pipe” is a connector

Allows you to combine a sequence of data wrangling steps

```
df_1 <- filter(df, <conditions>)
df_2 <- select(df_1, <conditions>)
df_3 <- mutate(df_2, <conditions>)
```

```
df_new <- df %>%
  filter(<conditions>) %>%
  select(<conditions>) %>%
  mutate(<conditions>)
```

the pipe!



original pipe

%>%

newer pipe

|>

You've already seen a type of connector

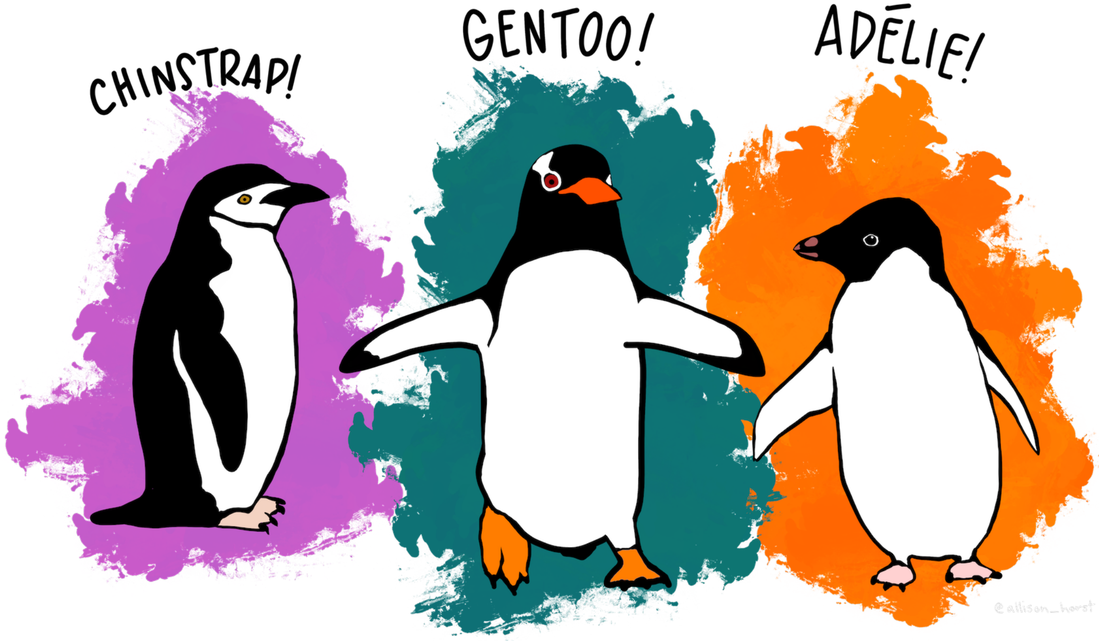
```
ggplot(data = <DATA>) +
 geom_<TYPE>(mapping = aes(<MAPPINGS>))
```

necessary in `ggplot2` to tell R this is all one plot

Practice, practice, practice...

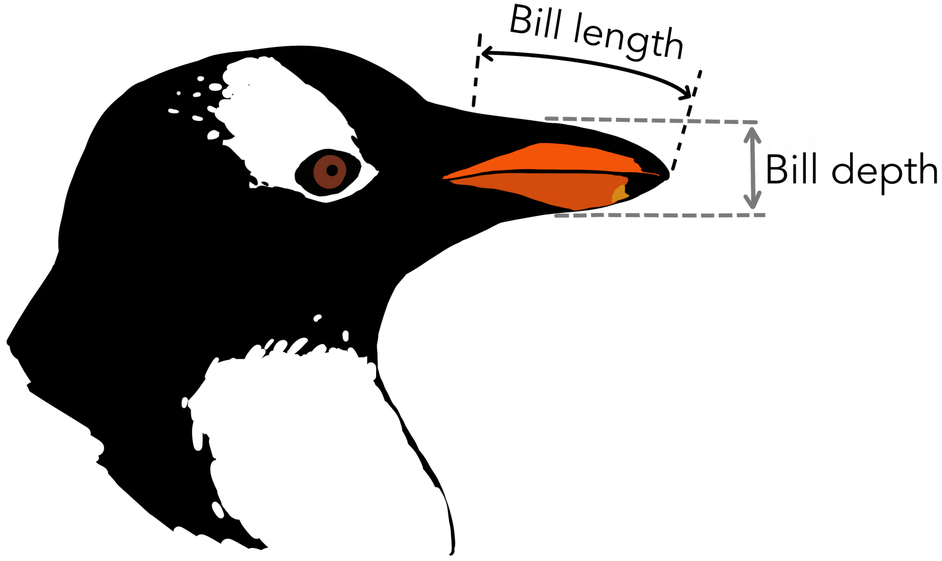
The `penguins` dataset

3 penguin species



Artwork by Allison Horst

Morphological measures



https://allisonhorst.github.io/palmerpenguins/

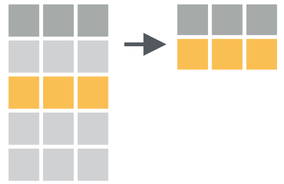
The `penguins` dataset

Snippet of the dataset:

species	island	bill_len	bill_dep	flipper_len	body_mass	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	*NA*	*NA*	*NA*	*NA*	*NA*	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007

https://allisonhorst.github.io/palmerpenguins/

`filter()`

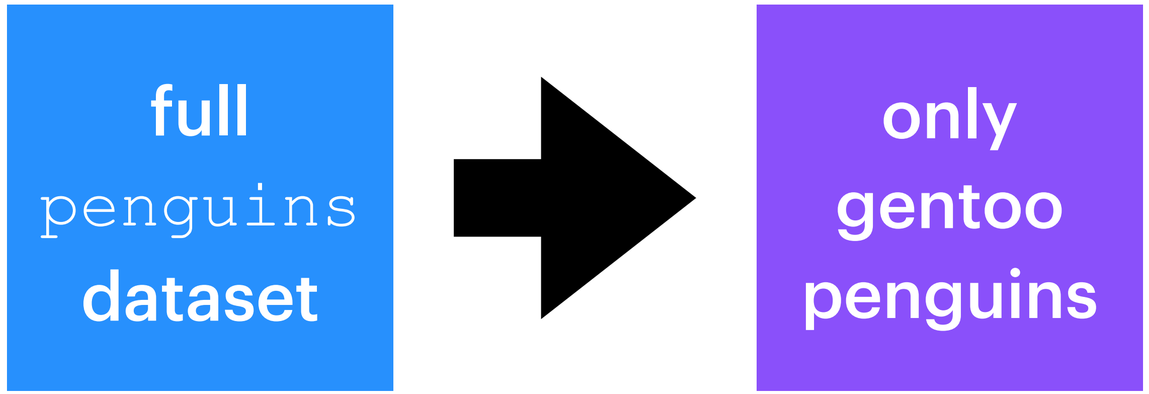


filter observations by their values

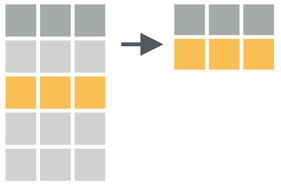
```
gentoo <- penguins %>%
  filter(species == "Gentoo")
```

two equals signs for "equals to"

Goal:



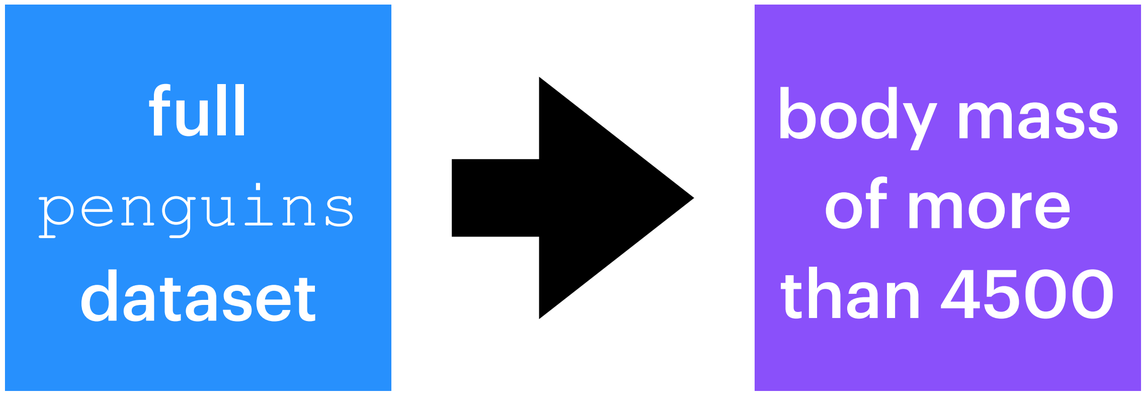
`filter()`



filter observations by their values

```
big_penguins <- penguins %>%
  filter(body_mass > 4500)
```

Goal:



Comparison “operators” in R

Operator	Description
`>`	Greater than
`<`	Less than
`==`	Equals to
`!=`	Not equal to
`>=`	Greater than or equal to
`<=`	Less than or equal to

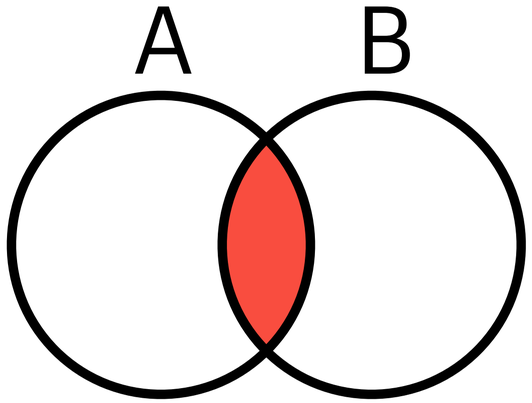
Combine multiple statements

`&` “and”

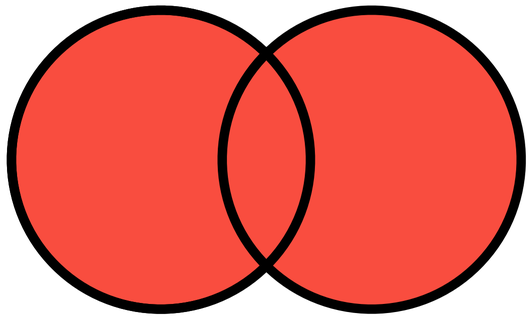
`|` “or”

https://www.r-bloggers.com/2021/09/r-booleans-comparison-and-logical-operators/

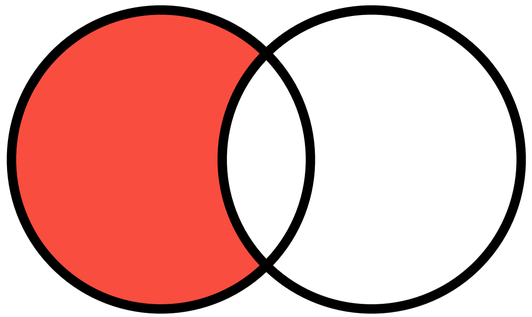
Comparison “operators” in R



A *and* B



A *or* B



A *and* not B

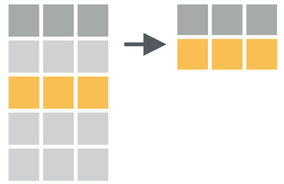
Combine multiple statements

&	“and”
|	“or”

https://www.r-bloggers.com/2021/09/r-booleans-comparison-and-logical-operators/

live coding

`filter()`



filter observations by their values

Goal:

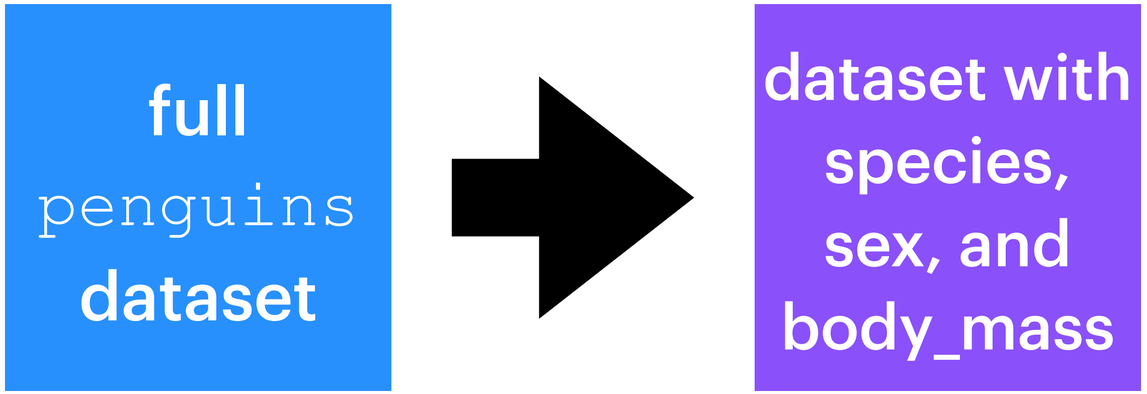
full `penguins` **dataset** → **body mass of more than 4500** → **only gentoo penguins**

`select()` 

select variables by their column names

```
new_penguins <- penguins %>%
  select(species, sex, body_mass)
```

Goal:

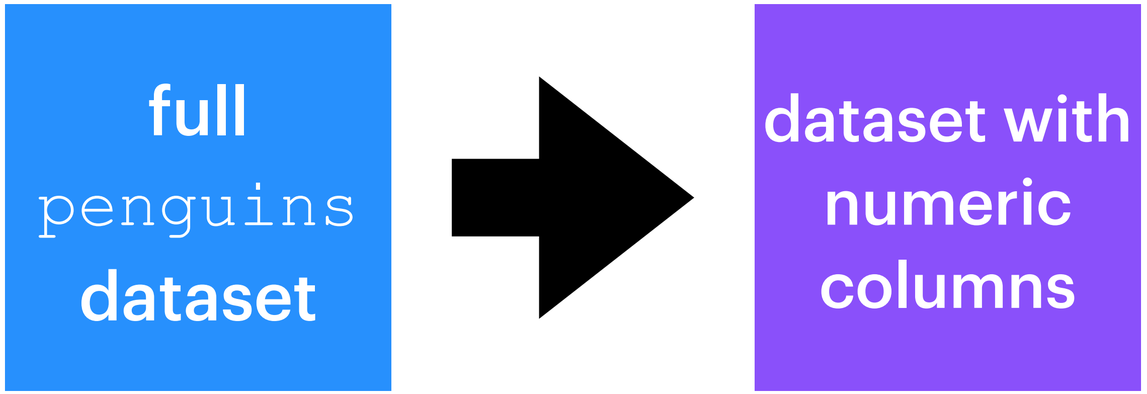


`select()` 

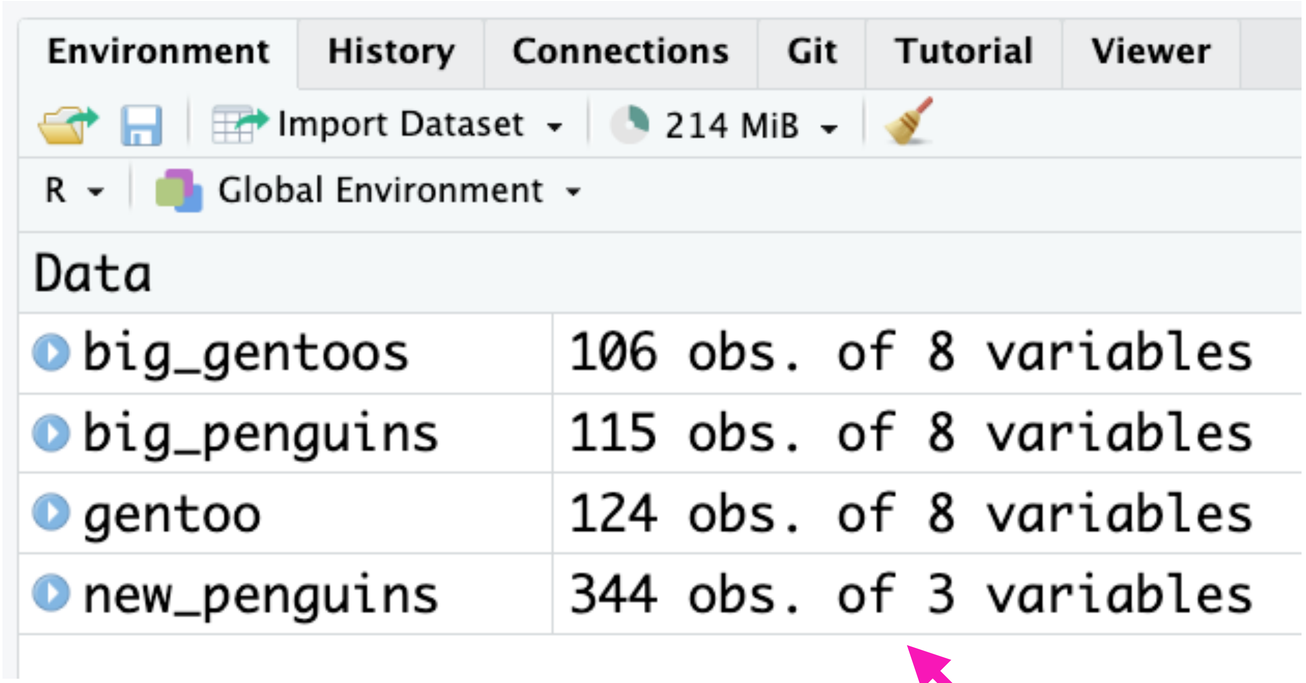
select variables by their column names

```
numeric_penguins <- penguins %>%
  select(where(is.numeric))
```

Goal:



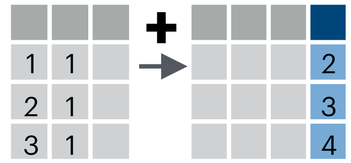
How does your R environment look now?



more dataframes

all of different dimensions

mutate()



mutate (or change) existing variables to create new ones

add to our original dataset

```
penguins <- penguins %>%
  mutate(bill_shape = bill_len/bill_dep)
```

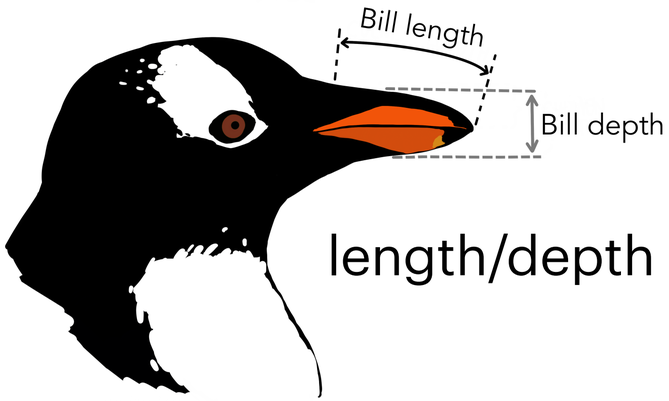
new variable name

the calculation

Goal:

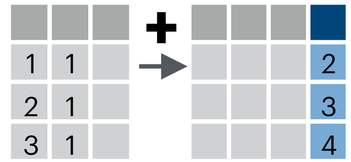
full `penguins` **dataset**

new variable for bill shape



length/depth

`mutate()`



mutate (or change) existing variables to create new ones

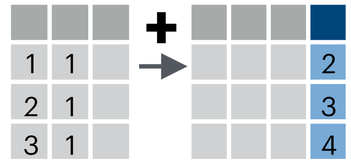
```
df <- df %>%
  mutate(<NEW_VARIABLE> = <EQUATION>)
```

instead create a new data frame

```
df_new <- df %>%
  mutate(<NEW_VARIABLE> = <EQUATION>)
```

live coding

`mutate()`



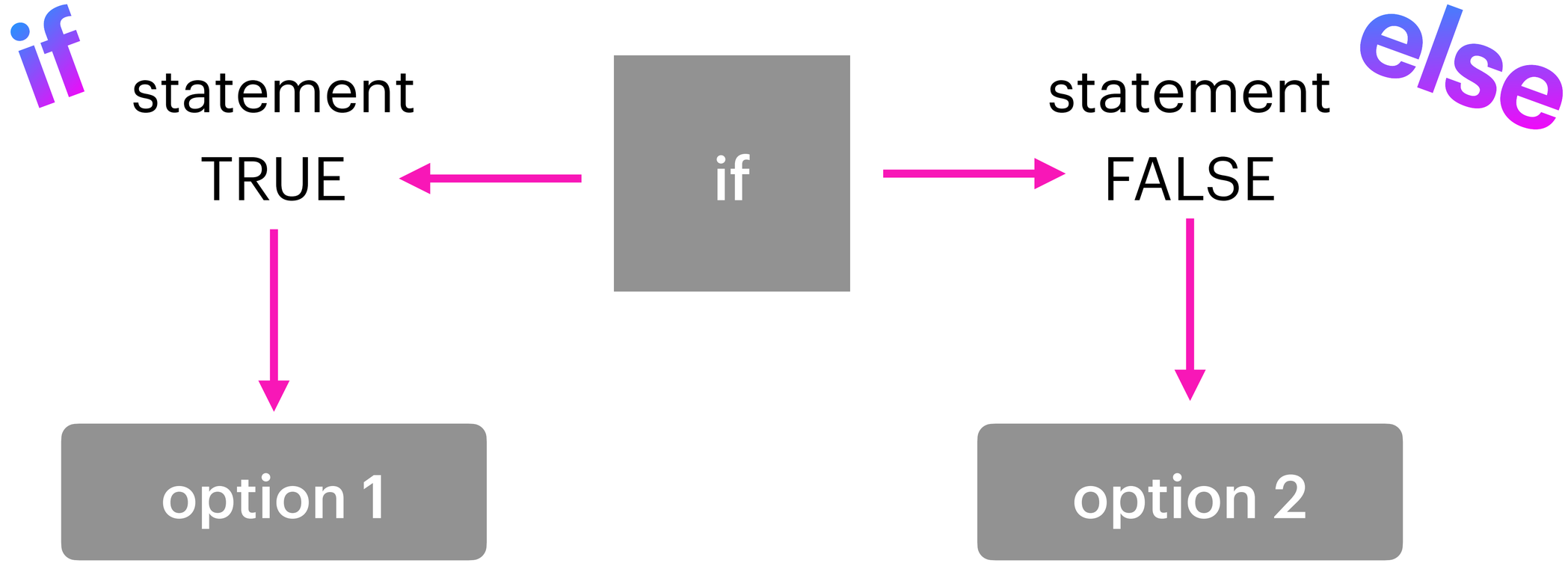
mutate (or change) existing variables to create new ones

Goal:

full `penguins` **dataset**

new variable year times body mass

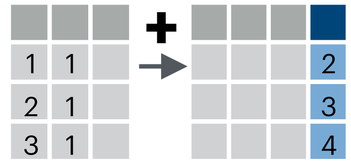
`ifelse()` function




```
ifelse(<test>, <true option>, <false option>)
```

live coding

`mutate()`



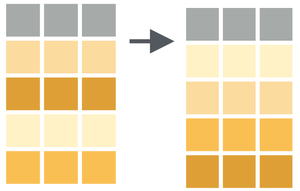
mutate (or change) existing variables to create new ones

Goal:

full `penguins` **dataset**

new variable “Adelie” (yes or no)

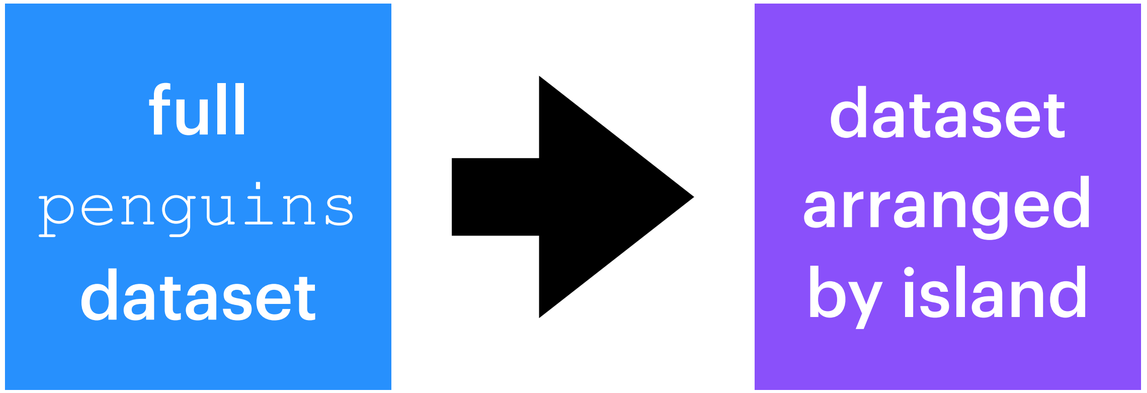
arrange()



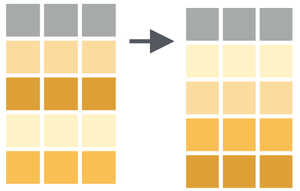
arrange observations by their values

```
penguins <- penguins %>%
  arrange(island)
```

Goal:



arrange()

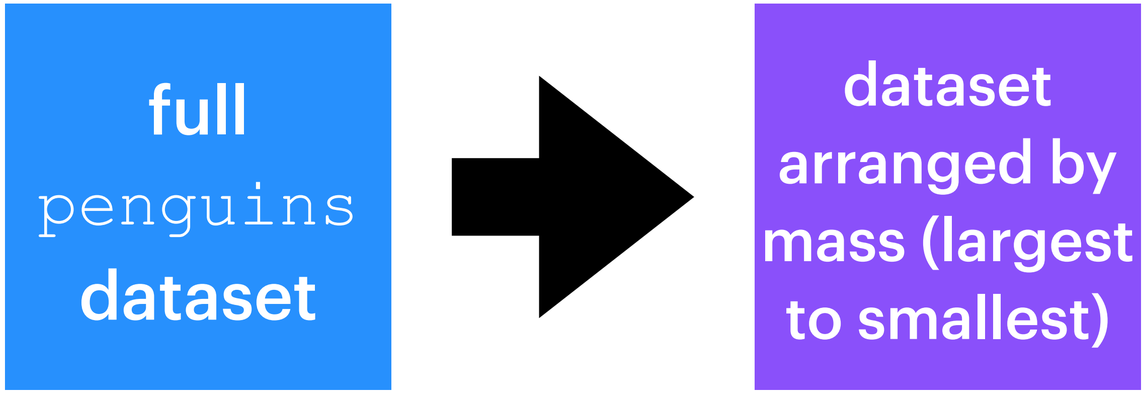


arrange observations by their values

```
penguins <- penguins %>%
  arrange(desc(body_mass))
```

"descending"

Goal:

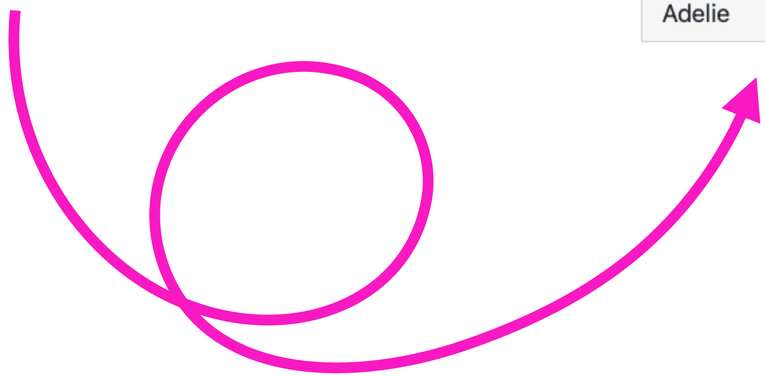


Make nice tables in Quarto

Load (or install + load) the R Package `knitr` to use the **`kable()`** function

```
head(penguins)
```

```
  species    island bill_len bill_dep flipper_len body_mass    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
```



```
kable(head(penguins))
```

species	island	bill_len	bill_dep	flipper_len	body_mass	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

Further reading

- Data Transformation with `dplyr` Cheatsheet
- Boehmke 2016, *Data Wrangling with R*
- Alston & Rick 2021. A Beginner's Guide to Conducting Reproducible Research. *The Bulletin of the Ecological Society of America*