



# An overview of dplyr

Daryn Ramsden

thisisdaryn at gmail dot com

last updated: 2021-03-06



# The data we will be using

```
#install.packages("palmerpenguins")  
library(palmerpenguins)
```

species <fct>	island <fct>	bill_length_mm <dbl>	bill_depth_mm <dbl>
Adelie	Biscoe	37.8	18.3
Adelie	Biscoe	37.7	18.7
Adelie	Biscoe	35.9	19.2
Adelie	Biscoe	38.2	18.1
Adelie	Biscoe	38.8	17.2
Adelie	Biscoe	35.3	18.9
Adelie	Biscoe	40.6	18.6
Adelie	Biscoe	40.5	17.9
Adelie	Biscoe	37.9	18.6
Adelie	Biscoe	40.5	18.9

21-30 of 344 rows | 1-4 of 8 columns

[Previous](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) ... [35](#) [Next](#)



# What do these variables represent?

Data were collected and made available by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER, a member of the Long Term Ecological Research Network.

- *species*: *Adelie*, *Chinstrap* or *Gentoo*
- *island*: *Biscoe*, *Dream* or *Torgersen* (factor)
- *bill\_length\_mm*: bill length mm (numeric)
- *bill\_depth\_mm*: bill depth in mm (numeric)
- *flipper\_length\_mm*: flipper length in mm (numeric)
- *body\_mass\_g*: body mass in grams (numeric)
- *sex*: *male* or *female* (factor)
- *year*: 2007, 2008 or 2009



# dplyr: a package for data manipulation

The data you get is almost in the form you want

**dplyr** is an R package that encapsulates many common data manipulation tasks

Sometimes you want to:

- keep only some of the rows
- keep only some of the columns
- adds new columns
- sort data
- provide summary statistics

**dplyr** has functions for each of these (and many others)



# Using **dplyr**

How do you install **dplyr**?

```
install.packages("dplyr")  
# or install.packages("tidyverse")
```

How do you use **dplyr**?

```
library(dplyr)  
# or library(tidyverse)
```



# Key single table verbs/functions

- Working with rows:
  - **filter**: keep only some of the rows based on column values
  - **slice**: keep some of the rows based on their location
  - **arrange**: sort data
- Working with columns:
  - **select**: keep only some of the columns
  - **mutate** adds new columns
  - **rename** change the name of specified columns
  - **relocate** changes the order of the columns
- Groups of rows:
- **summarise** (and **group\_by**): provide summary statistics



# filter

a function for specifying which rows to keep

Example 1: How do we get all penguins of the Chinstrap species?



# filter

a function for specifying which rows to keep

Example 1: How do we get all penguins of the Chinstrap species?

```
chinstrap <- filter(penguins, species == "Chinstrap")
```





# filter

a function for specifying which rows to keep

Example 1: How do we get all penguins of the Chinstrap species?

```
chinstrap <- filter(penguins, species == "Chinstrap")  
chinstrap
```

```
# A tibble: 68 x 8  
  species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g  
  <fct>   <fct>         <dbl>         <dbl>         <int>         <int>  
1 Chinstrap Dream         46.5           17.9           192           3500  
2 Chinstrap Dream         50             19.5           196           3900  
3 Chinstrap Dream         51.3           19.2           193           3650  
4 Chinstrap Dream         45.4           18.7           188           3525  
5 Chinstrap Dream         52.7           19.8           197           3725  
6 Chinstrap Dream         45.2           17.8           198           3950  
7 Chinstrap Dream         46.1           18.2           178           3250  
8 Chinstrap Dream         51.3           18.2           197           3750  
9 Chinstrap Dream         46             18.9           195           4150  
10 Chinstrap Dream        51.3           19.9           198           3700  
# ... with 58 more rows, and 2 more variables: sex <fct>, year <int>
```



# filter

a function for specifying which rows to keep

Example 2: How do we get penguins that are 4 kg or greater?



# filter

a function for specifying which rows to keep

Example 2: How do we get penguins that are 4 kg or greater?

```
penguins_4k <- filter(penguins, body_mass_g >= 4000)
```



# filter

a function for specifying which rows to keep

Example 2: How do we get penguins that are 4 kg or greater?

```
penguins_4k <- filter(penguins, body_mass_g >= 4000)
penguins_4k
```

```
# A tibble: 177 x 8
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
	<fct>	<fct>	<dbl>	<dbl>	<int>	<int>
1	Adelie	Torgersen	39.2	19.6	195	4675
2	Adelie	Torgersen	42	20.2	190	4250
3	Adelie	Torgersen	34.6	21.1	198	4400
4	Adelie	Torgersen	42.5	20.7	197	4500
5	Adelie	Torgersen	46	21.5	194	4200
6	Adelie	Dream	39.2	21.1	196	4150
7	Adelie	Dream	39.8	19.1	184	4650
8	Adelie	Dream	44.1	19.7	196	4400
9	Adelie	Dream	39.6	18.8	190	4600
10	Adelie	Dream	42.3	21.2	191	4150

```
# ... with 167 more rows, and 2 more variables: sex <fct>, year <int>
```



# Assessment

How many penguins were found on Torgersen island (*Torgersen*)?



# Assessment

How many penguins were found on Torgersen island (*Torgersen*)?

```
torgersen<- filter(penguins, island == "Torgersen")  
dim(torgersen)
```

```
[1] 52  8
```

Also could have used:

```
torgersen<- penguins %>% filter(island == "Torgersen")  
dim(torgersen)
```

```
[1] 52  8
```



# select

A function/verb for specifying which columns to keep

As of dplyr 1.0 there are 5 ways to use select

1. By **position**
2. By **name**
3. by **function of name**
4. by **type**
5. by combination of the above using logical operators (**|**, **&**, **!**)



## select by position

Example: select columns 1, 3 and 5 from `penguins`

```
penguins %>% select(1, 3, 5)
```





# select by position

Example: select columns 1, 3 and 5 from `penguins`

```
penguins %>% select(1, 3, 5)
```

```
# A tibble: 344 x 3
  species bill_length_mm flipper_length_mm
  <fct>      <dbl>          <int>
1 Adelie    39.1             181
2 Adelie    39.5             186
3 Adelie    40.3             195
4 Adelie    NA                NA
5 Adelie    36.7             193
6 Adelie    39.3             190
7 Adelie    38.9             181
8 Adelie    39.2             195
9 Adelie    34.1             193
10 Adelie   42              190
# ... with 334 more rows
```



## select by name

Example: select *species*, *island* and *body\_mass\_g*

```
penguins %>% select(species, island, body_mass_g)
```



# select by name

Example: select *species*, *island* and *body\_mass\_g*

```
penguins %>% select(species, island, body_mass_g)
```

```
# A tibble: 344 x 3
  species island    body_mass_g
  <fct>   <fct>         <int>
1 Adelie  Torgersen     3750
2 Adelie  Torgersen     3800
3 Adelie  Torgersen     3250
4 Adelie  Torgersen      NA
5 Adelie  Torgersen     3450
6 Adelie  Torgersen     3650
7 Adelie  Torgersen     3625
8 Adelie  Torgersen     4675
9 Adelie  Torgersen     3475
10 Adelie Torgersen     4250
# ... with 334 more rows
```



# `select` by a function of column names

`select` can be used in conjunction with other useful functions such as:

- `starts_with`
- `ends_with`
- `contains`
- `matches`



## select by a function of column names

Example: Choose all columns that contain "mm":

```
penguins_mm <- penguins %>% select(contains("mm"))
```



# select by a function of column names

Example: Choose all columns that contain "mm":

```
penguins_mm <- penguins %>% select(contains("mm"))
```

```
penguins_mm
```

```
# A tibble: 344 x 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
# ... with 334 more rows
```



## select by a function of column names

Example: How to choose all columns starting with "bill":

```
bills_df <- penguins %>% select(starts_with("bill"))
```



# select by a function of column names

Example: How to choose all columns starting with "bill":

```
bills_df <- penguins %>% select(starts_with("bill"))
```

```
bills_df
```

```
# A tibble: 344 x 2
  bill_length_mm bill_depth_mm
      <dbl>         <dbl>
1           39.1           18.7
2           39.5           17.4
3           40.3            18
4            NA            NA
5           36.7           19.3
6           39.3           20.6
7           38.9           17.8
8           39.2           19.6
9           34.1           18.1
10          42            20.2
# ... with 334 more rows
```





## select by type

Example: choose all numeric columns:

```
penguins %>% select(where(is.numeric))
```



# select by type

Example: choose all numeric columns:

```
penguins %>% select(where(is.numeric))
```

```
# A tibble: 344 x 5
  bill_length_mm bill_depth_mm flipper_length_mm body_mass_g year
      <dbl>         <dbl>           <int>         <int> <int>
1         39.1         18.7             181          3750 2007
2         39.5         17.4             186          3800 2007
3         40.3          18              195          3250 2007
4          NA          NA                NA            NA 2007
5         36.7         19.3             193          3450 2007
6         39.3         20.6             190          3650 2007
7         38.9         17.8             181          3625 2007
8         39.2         19.6             195          4675 2007
9         34.1         18.1             193          3475 2007
10        42          20.2             190          4250 2007
# ... with 334 more rows
```



## select by logical combination

Example: choose all factor variables or variables containing the word "bill"

```
penguins %>% select(where(is.factor) | contains("bill"))
```



## select by logical combination

Example: choose all factor variables or variables containing the word "bill"

```
penguins %>% select(where(is.factor) | contains("bill"))
```

```
# A tibble: 344 x 5
  species island sex bill_length_mm bill_depth_mm
  <fct>   <fct>   <fct>   <dbl>         <dbl>
1 Adelie Torgersen male      39.1          18.7
2 Adelie Torgersen female    39.5          17.4
3 Adelie Torgersen female    40.3           18
4 Adelie Torgersen <NA>      NA            NA
5 Adelie Torgersen female    36.7          19.3
6 Adelie Torgersen male      39.3          20.6
7 Adelie Torgersen female    38.9          17.8
8 Adelie Torgersen male      39.2          19.6
9 Adelie Torgersen <NA>      34.1          18.1
10 Adelie Torgersen <NA>      42            20.2
# ... with 334 more rows
```



# mutate

## a function to add new columns

Example: Adding a column that indicates whether a penguin has a mass greater than 4 kg

```
penguin_extra <- penguins %>%  
  mutate(above_4kg= if_else(body_mass_g > 4000, TRUE, FALSE))
```



# mutate

## a function to add new columns

Example: Adding a column that indicates whether a penguin has a mass greater than 4 kg

```
penguin_extra <- penguins %>%  
  mutate(above_4kg= if_else(body_mass_g > 4000, TRUE, FALSE))  
  
head(penguin_extra)
```

```
# A tibble: 6 x 9  
  species island bill_length_mm bill_depth_mm flipper_length_... body_mass_g sex  
  <fct>   <fct>         <dbl>         <dbl>         <int>         <int> <fct>  
1 Adelie  Torge...         39.1          18.7           181          3750 male  
2 Adelie  Torge...         39.5          17.4           186          3800 fema...  
3 Adelie  Torge...         40.3          18             195          3250 fema...  
4 Adelie  Torge...         NA            NA             NA            NA <NA>  
5 Adelie  Torge...         36.7          19.3           193          3450 fema...  
6 Adelie  Torge...         39.3          20.6           190          3650 male  
# ... with 2 more variables: year <int>, above_4kg <lgl>
```



# arrange

## A function for sorting data

Example: Sort all penguins by body mass:

```
penguins_sorted <- penguins %>% arrange(body_mass_g)
```



# arrange

## A function for sorting data

Example: Sort all penguins by body mass:

```
penguins_sorted <- penguins %>%  
  arrange(body_mass_g)  
penguins_sorted
```

species <fct>	island <fct>	bill_length_... <dbl>	bill_depth_m... <dbl>	flipper_lengt... <int>
Chinstrap	Dream	46.9	16.6	192
Adelie	Biscoe	36.5	16.6	181
Adelie	Biscoe	36.4	17.1	184
Adelie	Biscoe	34.5	18.1	187
Adelie	Dream	33.1	16.1	178
Adelie	Torgersen	38.6	17.0	188
Chinstrap	Dream	43.2	16.6	187
Adelie	Biscoe	37.9	18.6	193
Adelie	Dream	37.5	18.9	179
Adelie	Dream	37.0	16.9	185

1-10 of 344 rows | 1-5 of 8 columns

Previous **1** 2 3 4 5 6 ... 35 Next 32 / 70





# sorting with multiple columns using **arrange**

Example sorting by species, then by descending order of mass:

```
penguins_sorted2 <- penguins %>%  
  arrange(species, desc(body_mass_g))  
penguins_sorted2
```

species <fct>	island <fct>	bill_length_... <dbl>	bill_depth_m... <dbl>	flipper_lengt... <int>
Adelie	Biscoe	43.2	19.0	197
Adelie	Biscoe	41.0	20.0	203
Adelie	Torgersen	42.9	17.6	196
Adelie	Torgersen	39.2	19.6	195
Adelie	Dream	39.8	19.1	184
Adelie	Dream	39.6	18.8	190
Adelie	Biscoe	45.6	20.3	191
Adelie	Torgersen	42.5	20.7	197
Adelie	Dream	37.5	18.5	199
Adelie	Torgersen	41.8	19.4	198

1-10 of 344 rows | 1-5 of 8 columns

Previous **1** 2 3 4 5 6 ... 35 Next



# summarise/summarize

A verb/function to get summary statistics.

Question: what's the mean flipper length and body mass among the Palmer penguins?

```
penguins %>%  
  summarise(num_penguins = n(),  
            avg_mass = mean(body_mass_g, na.rm = TRUE),  
            avg_fl_length = mean(flipper_length_mm, na.rm = TRUE))
```

```
# A tibble: 1 x 3  
  num_penguins avg_mass avg_fl_length  
    <int>      <dbl>      <dbl>  
1         344    4202.         201.
```



# group\_by

A function that makes **summarise** really powerful

**group\_by** creates a grouped data frame based on columns you specify

For example, grouping the penguins by island and species:

```
gr_penguins <- penguins %>% group_by(island, species)
```



# group\_by

A function that makes **summarise** really powerful

**group\_by** creates a grouped data frame based on columns you specify

For example, grouping the penguins by island and species:

```
gr_penguins <- penguins %>% group_by(island, species)
head(gr_penguins)
```

```
# A tibble: 6 x 8
# Groups:   island, species [1]
  species island bill_length_mm bill_depth_mm flipper_length_... body_mass_g sex
  <fct>   <fct>         <dbl>         <dbl>         <int>         <int> <fct>
1 Adelie Torge...         39.1          18.7           181           3750 male
2 Adelie Torge...         39.5          17.4           186           3800 fema...
3 Adelie Torge...         40.3          18             195           3250 fema...
4 Adelie Torge...         NA            NA             NA            NA <NA>
5 Adelie Torge...         36.7          19.3           193           3450 fema...
6 Adelie Torge...         39.3          20.6           190           3650 male
# ... with 1 more variable: year <int>
```



# How is the grouped data frame different?

- Extra information is added to the data frame
- rows that match on all the grouping variables will be in the same group
- rows that don't match on all the grouping variables will be in different groups



# group\_by and summarise together

Now let's do the same summary as before with the grouped data:

```
gr_penguins %>% summarise(num_penguins = n(),  
                           avg_mass = mean(body_mass_g, na.rm = TRUE),  
                           avg_fl_length = mean(flipper_length_mm,  
                                                na.rm = TRUE))
```

```
# A tibble: 5 x 5  
# Groups:   island [3]  
  island species num_penguins avg_mass avg_fl_length  
  <fct>   <fct>         <int>    <dbl>    <dbl>  
1 Biscoe  Adelie           44    3710.    189.  
2 Biscoe  Gentoo          124    5076.    217.  
3 Dream   Adelie           56    3688.    190.  
4 Dream   Chinstrap        68    3733.    196.  
5 Torgersen Adelie           52    3706.    191.
```



# New features of `summarise`

`dplyr` 1.0 has some new features of `summarise`:

- summaries that return multiple values
- summaries that return multiple columns



# Summaries with multiple values

Example: using `summarise` to get the range of bill lengths for each species of penguin:

```
penguins %>%  
  group_by(species) %>%  
  summarise(rng = range(bill_length_mm, na.rm = TRUE))
```

```
# A tibble: 6 x 2  
# Groups:   species [3]  
  species    rng  
  <fct>    <dbl>  
1 Adelie   32.1  
2 Adelie   46  
3 Chinstrap 40.9  
4 Chinstrap 58  
5 Gentoo   40.9  
6 Gentoo   59.6
```





# Summaries with multiple columns

Example: using `summarise` to find the minimum and maximum mass penguin on each island:

```
penguins %>%  
  group_by(island) %>%  
  summarise(tibble(min_mass = min(body_mass_g, na.rm = TRUE),  
                    max_mass = max(body_mass_g, na.rm = TRUE)))
```

```
# A tibble: 3 x 3  
  island      min_mass max_mass  
  <fct>      <int>    <int>  
1 Biscoe      2850     6300  
2 Dream       2700     4800  
3 Torgersen   2900     4700
```



## So ... a couple other things about groups

- default behavior is to remove the last level of grouping after a call to **summarise**
- grouped data can be used with other **dplyr** verbs e.g. **mutate**
- you can ungroup data using **ungroup**



## Example using `group_by` with `mutate`

What if we wanted to give each penguin a number within its species?

```
numbered_penguins <- penguins %>%  
  group_by(species) %>%  
  mutate(penguin_num = 1:n())
```



## Example using `group_by` with `mutate`

What if we wanted to give each penguin a number within its species?

```
numbered_penguins <- penguins %>%  
  group_by(species) %>%  
  mutate(penguin_num = 1:n())
```

```
numbered_penguins
```

```
# A tibble: 344 x 9
```

```
# Groups:   species [3]
```

	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
7	Adelie	Torgersen	38.9	17.8	181	3625
8	Adelie	Torgersen	39.2	19.6	195	4675
9	Adelie	Torgersen	34.1	18.1	193	3475
10	Adelie	Torgersen	42	20.2	190	4250

```
# ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
```

```
#   penguin_num <int>
```



# rename

A function/verb to rename columns

Works like `select`

Example: renaming by position

```
penguins_different <- penguins %>% rename(bill_length = 3,  
                                          bill_depth = 4)
```

```
penguins_different
```

```
# A tibble: 344 x 8
```

	species	island	bill_length	bill_depth	flipper_length_mm	body_mass_g	sex
	<fct>	<fct>	<dbl>	<dbl>	<int>	<int>	<fct>
1	Adelie	Torgersen	39.1	18.7	181	3750	male
2	Adelie	Torgersen	39.5	17.4	186	3800	female
3	Adelie	Torgersen	40.3	18	195	3250	female
4	Adelie	Torgersen	NA	NA	NA	NA	<NA>
5	Adelie	Torgersen	36.7	19.3	193	3450	female
6	Adelie	Torgersen	39.3	20.6	190	3650	male
7	Adelie	Torgersen	38.9	17.8	181	3625	female
8	Adelie	Torgersen	39.2	19.6	195	4675	male



# rename\_with

`rename_with` can be used with a specified transformation (and optionally with a column selection).

Example: rename all columns to be uppercase

```
penguins %>% rename_with(toupper)
```

```
# A tibble: 344 x 8
  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
7 Adelie Torgersen      38.9           17.8           181           3625
8 Adelie Torgersen      39.2           19.6           195           4675
9 Adelie Torgersen      34.1           18.1           193           3475
10 Adelie Torgersen      42             20.2           190           4250
# ... with 334 more rows, and 2 more variables: SEX <fct>, YEAR <int>
```



# rename\_with

```
penguins %>% rename_with(toupper, where(is.numeric))
```

```
# A tibble: 344 x 8
  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
7 Adelie  Torgersen      38.9          17.8           181           3625
8 Adelie  Torgersen      39.2          19.6           195           4675
9 Adelie  Torgersen      34.1          18.1           193           3475
10 Adelie Torgersen      42            20.2           190           4250
# ... with 334 more rows, and 2 more variables: sex <fct>, YEAR <int>
```



# relocate

## A function

- (**default**) move selected variables to the front
- move selected columns before a specified location
- move selected columns after a specified location





# relocate examples

Example: bring all the factor variables to the front

```
penguins %>% relocate(where(is.factor))
```



# relocate examples

Example: bring all the factor variables to the front

```
penguins %>% relocate(where(is.factor))
```

```
# A tibble: 344 x 8
  species island sex bill_length_mm bill_depth_mm flipper_length_mm
  <fct> <fct> <fct> <dbl> <dbl> <int>
1 Adelie Torgersen male 39.1 18.7 181
2 Adelie Torgersen female 39.5 17.4 186
3 Adelie Torgersen female 40.3 18 195
4 Adelie Torgersen <NA> NA NA NA
5 Adelie Torgersen female 36.7 19.3 193
6 Adelie Torgersen male 39.3 20.6 190
7 Adelie Torgersen female 38.9 17.8 181
8 Adelie Torgersen male 39.2 19.6 195
9 Adelie Torgersen <NA> 34.1 18.1 193
10 Adelie Torgersen <NA> 42 20.2 190
# ... with 334 more rows, and 2 more variables: body_mass_g <int>, year <int>
```



# relocate examples

Example: relocate all factor variables after *body\_mass\_g*

```
penguins %>% relocate(contains("bill"), .after = body_mass_g)
```

```
# A tibble: 344 x 8
  species island flipper_length_mm body_mass_g bill_length_mm bill_depth_mm
  <fct>   <fct>         <int>         <int>         <dbl>         <dbl>
1 Adelie Torgersen         181         3750         39.1         18.7
2 Adelie Torgersen         186         3800         39.5         17.4
3 Adelie Torgersen         195         3250         40.3          18
4 Adelie Torgersen          NA          NA          NA          NA
5 Adelie Torgersen         193         3450         36.7         19.3
6 Adelie Torgersen         190         3650         39.3         20.6
7 Adelie Torgersen         181         3625         38.9         17.8
8 Adelie Torgersen         195         4675         39.2         19.6
9 Adelie Torgersen         193         3475         34.1         18.1
10 Adelie Torgersen         190         4250          42         20.2
# ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```



## across: a really useful new function

What if you wanted the average value - per group - of each numeric column?

Annoying way:

```
penguins %>% group_by(species) %>%  
  summarise(avg_bill_length = mean(bill_length_mm, na.rm = TRUE),  
            avg_bill_depth = mean(bill_depth_mm, na.rm = TRUE),  
            avg_fl_length_mm = mean(flipper_length_mm, na.rm = TRUE),  
            avg_body_mass_g = mean(body_mass_g, na.rm = TRUE))
```



## across: a really useful new function

What if you wanted the average value - per group - of each numeric column?

Annoying way:

```
penguins %>% group_by(species) %>%  
  summarise(avg_bill_length = mean(bill_length_mm, na.rm = TRUE),  
            avg_bill_depth = mean(bill_depth_mm, na.rm = TRUE),  
            avg_fl_length_mm = mean(flipper_length_mm, na.rm = TRUE),  
            avg_body_mass_g = mean(body_mass_g, na.rm = TRUE))
```

# A tibble: 3 x 5

	species	avg_bill_length	avg_bill_depth	avg_fl_length_mm	avg_body_mass_g
	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	Adelie	38.8	18.3	190.	3701.
2	Chinstrap	48.8	18.4	196.	3733.
3	Gentoo	47.5	15.0	217.	5076.



## **across:** a really useful new function

What if you wanted the average value - per group - of each numeric column?

Neater/better way:

```
penguins %>% group_by(species) %>%  
  summarise(across(where(is.numeric) & !contains("year"),  
                    mean, na.rm = TRUE))
```



## across: a really useful new function

What if you wanted the average value - per group - of each numeric column?

Neater/better way:

```
penguins %>% group_by(species) %>%  
  summarise(across(where(is.numeric) & !contains("year"),  
                    mean, na.rm = TRUE))
```

```
# A tibble: 3 x 5
```

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	Adelie	38.8	18.3	190.	3701.
2	Chinstrap	48.8	18.4	196.	3733.
3	Gentoo	47.5	15.0	217.	5076.



## across: a closer look

`across` has two primary arguments:

- `.cols` selects the columns you want to operate on
- `.fns` is a function or list of functions that you want to apply
  - can be a `purrr` style formula





## multiple summaries with **across**

Example: For each island, what is the average of all numeric variables and the count of all factor variables?

```
penguins %>%  
  group_by(island) %>%  
  summarise(  
    across(where(is.numeric), mean, na.rm = TRUE),  
    across(where(is.factor), n_distinct),  
    n = n(),  
  )
```

```
# A tibble: 3 x 9  
  island bill_length_mm bill_depth_mm flipper_length_... body_mass_g year species  
  <fct>         <dbl>         <dbl>         <dbl>         <dbl> <dbl> <int>  
1 Biscoe         45.3           15.9           210.         4716. 2008.     2  
2 Dream          44.2           18.3           193.         3713. 2008.     2  
3 Torge...      39.0           18.4           191.         3706. 2008.     1  
# ... with 2 more variables: sex <int>, n <int>
```



## across example with filter

Example: get all rows without missing values:

```
penguins_complete <- penguins %>%  
  filter(across(everything(), ~ !is.na(.x)))
```

Is that any different to?

```
penguins_complete2 <- penguins %>%  
  filter(across(everything(), complete.cases))
```



# across example with distinct

All combinations of variables meeting specified criteria using `distinct`

```
penguins %>% distinct(across(is.factor, sort = TRUE))
```

```
# A tibble: 13 x 3
  species island sex
  <fct>   <fct> <fct>
1 Adelie  Torgersen male
2 Adelie  Torgersen female
3 Adelie  Torgersen <NA>
4 Adelie  Biscoe female
5 Adelie  Biscoe male
6 Adelie  Dream female
7 Adelie  Dream male
8 Adelie  Dream <NA>
9 Gentoo  Biscoe female
10 Gentoo Biscoe male
11 Gentoo Biscoe <NA>
12 Chinstrap Dream female
13 Chinstrap Dream male
```



## across example with count

Counts of all combinations of variables meeting specified criteria using `count`

```
penguins %>% count(across(is.factor, sort = TRUE))
```

```
# A tibble: 13 x 4
  species island sex      n
  <fct>   <fct>   <fct> <int>
1 Adelie  Biscoe  female 22
2 Adelie  Biscoe  male   22
3 Adelie  Dream   female 27
4 Adelie  Dream   male   28
5 Adelie  Dream   <NA>    1
6 Adelie  Torgersen female 24
7 Adelie  Torgersen male   23
8 Adelie  Torgersen <NA>    5
9 Chinstrap Dream   female 34
10 Chinstrap Dream   male   34
11 Gentoo  Biscoe  female 58
12 Gentoo  Biscoe  male   61
13 Gentoo  Biscoe  <NA>    5
```



# across example with mutate

Using **across** with **mutate** to rescale all numeric variables between 0 and 1

```
rescale01 <- function(x) {  
  rng <- range(x, na.rm = TRUE)  
  (x - rng[1]) / (rng[2] - rng[1])  
}  
  
penguins_rescaled <- penguins %>%  
  mutate(across(where(is.numeric), rescale01))  
  
penguins_rescaled
```

# A tibble: 344 x 8

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
	<fct>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	Adelie	Torgersen	0.255	0.667	0.153	0.292
2	Adelie	Torgersen	0.269	0.512	0.237	0.306
3	Adelie	Torgersen	0.298	0.583	0.390	0.153
4	Adelie	Torgersen	NA	NA	NA	NA
5	Adelie	Torgersen	0.167	0.738	0.356	0.208
6	Adelie	Torgersen	0.262	0.893	0.305	0.264
7	Adelie	Torgersen	0.247	0.560	0.153	0.257
8	Adelie	Torgersen	0.258	0.774	0.390	0.549
9	Adelie	Torgersen	0.0727	0.595	0.356	0.215



# Row-wise operations

Question: what if we wanted to create a new column that was the average of the *bill\_depth\_mm* and *bill\_length\_mm* variables?

You might try:

```
penguins %>% select(contains("bill")) %>%  
  mutate(avg = mean(c(bill_length_mm, bill_depth_mm), na.rm = TRUE))
```



# Row-wise operations

Question: what if we wanted to create a new column that was the average of the *bill\_depth\_mm* and *bill\_length\_mm* variables?

You might try:

```
penguins %>% select(contains("bill")) %>%  
  mutate(avg = mean(c(bill_length_mm, bill_depth_mm), na.rm = TRUE))
```

```
# A tibble: 344 x 3  
  bill_length_mm bill_depth_mm  avg  
    <dbl>         <dbl> <dbl>  
1         39.1         18.7  30.5  
2         39.5         17.4  30.5  
3         40.3          18   30.5  
4          NA          NA   30.5  
5         36.7         19.3  30.5  
6         39.3         20.6  30.5  
7         38.9         17.8  30.5  
8         39.2         19.6  30.5  
9         34.1         18.1  30.5  
10        42          20.2  30.5  
# ... with 334 more rows
```



# Using `rowwise`

We can use `rowwise` prior to mutate instead

```
penguins %>%  
  select(contains("bill")) %>%  
  rowwise() %>%  
  mutate(avg = mean(c(bill_length_mm, bill_depth_mm), na.rm = TRUE))
```





# Using `rowwise`

We can use `rowwise` prior to mutate instead

```
penguins %>%  
  select(contains("bill")) %>%  
  rowwise() %>%  
  mutate(avg = mean(c(bill_length_mm, bill_depth_mm), na.rm = TRUE))
```

```
# A tibble: 344 x 3
```

```
# Rowwise:
```

	bill_length_mm	bill_depth_mm	avg
	<dbl>	<dbl>	<dbl>
1	39.1	18.7	28.9
2	39.5	17.4	28.4
3	40.3	18	29.2
4	NA	NA	NaN
5	36.7	19.3	28
6	39.3	20.6	30.0
7	38.9	17.8	28.4
8	39.2	19.6	29.4
9	34.1	18.1	26.1
10	42	20.2	31.1

```
# ... with 334 more rows
```



# Joins

To illustrate the join functions, we will use two small data sets

First, a data frame containing the populations of 8 countries (via census.gov):

```
populations <- readr::read_csv("data/populations.csv")
populations
```

```
# A tibble: 8 x 2
  Country      Population
  <chr>         <dbl>
1 India        1326093247
2 United States 329877505
3 Indonesia    267026366
4 Pakistan     233500636
5 Nigeria      214028302
6 Bangladesh   162650853
7 Russia       141722205
8 Mexico       128649565
```



# Joins

Next, a data frame containing the land areas of some countries (via wikipedia)

```
areas <- readr::read_csv("data/areas.csv")
areas
```

```
# A tibble: 7 x 2
  Country      Area
  <chr>      <dbl>
1 Russia    16377742
2 China      9326410
3 United States 9147593
4 Brazil     8460415
5 India      2973190
6 Indonesia  1811569
7 Nigeria     910768
```

Note that some countries are in both data frames while others are only in one.



## Inner joins with `inner_join`

Inner joins combine tables, taking only entries that are in both:

```
inner_join(populations, areas)
```

```
# A tibble: 5 x 3
  Country      Population      Area
  <chr>         <dbl>     <dbl>
1 India         1326093247 2973190
2 United States 329877505  9147593
3 Indonesia     267026366 1811569
4 Nigeria       214028302  910768
5 Russia        141722205 16377742
```



# Full joins with `full_join`

Full joins combine tables, taking all entries from either:

```
full_join(populations, areas)
```

```
# A tibble: 10 x 3
  Country      Population      Area
  <chr>         <dbl>     <dbl>
1 India         1326093247 2973190
2 United States 329877505 9147593
3 Indonesia     267026366 1811569
4 Pakistan      233500636      NA
5 Nigeria       214028302 910768
6 Bangladesh    162650853      NA
7 Russia        141722205 16377742
8 Mexico        128649565      NA
9 China          NA 9326410
10 Brazil        NA 8460415
```



## Left (or right) joins with `left_join` (or `right_join`)

Left joins take all the rows in the first table along with any rows in the second table that match

```
left_join(populations, areas)
```

```
# A tibble: 8 x 3
  Country      Population      Area
  <chr>          <dbl>    <dbl>
1 India      1326093247 2973190
2 United States 329877505 9147593
3 Indonesia   267026366 1811569
4 Pakistan    233500636      NA
5 Nigeria     214028302  910768
6 Bangladesh  162650853      NA
7 Russia      141722205 16377742
8 Mexico      128649565      NA
```