Aggregating

Emily Malcolm-White

```
#LOAD PACKAGES
library(tidyverse)
```

`palmerpenguins` dataset

Size measurements, clutch observations, and blood isotope ratios for adult foraging Adélie, Chinstrap, and Gentoo penguins observed on islands in the Palmer Archipelago near Palmer Station, Antarctica. Data were collected and made available by Dr. Kristen Gorman and the Palmer Station Long Term Ecological Research (LTER) Program.

```
#LOAD DATA
library(palmerpenguins) #<1>
data(penguins)          #<2>
```

① Load the `palmerpenguins` package
② Display the penguins dataset in the environment

Remove rows with missing data with drop_na()

```
penguins <- penguins %>%    #<2>
  drop_na()                 #<1>
```

① Drops all the rows in the penguins dataset which has missing data (NA values)
② overwrite the penguins dataset with the penguins dataset without the missing rows

Warning

Is it appropriate to remove rows with missing data? How many rows have missing data? Do the missing rows have something in common?
Removing rows can affect the validity and generalizability of your analysis!

`count()`

`count()` lets you quickly count the unique values of one or more variables. Suppose you want the number of penguins on each island.

```
penguins %>%
    count(island)
```

```
# A tibble: 3 x 2
  island        n
  <fct>     <int>
1 Biscoe      163
2 Dream       123
3 Torgersen    47
```

`summarize()` or `summarise()` (either works)

Suppose we are interested in the average bill length of all Adelie penguins:

```
penguins %>%
  filter(species == "Adelie") %>%                       #<1>
  summarize(average_bill_length = mean(bill_length_mm))  #<2>
```

① only include the rows where the species is Adelie
② calculate the average bill length; save this as `average_bill_length`

```
# A tibble: 1 x 1
  average_bill_length
                <dbl>
1                38.8
```

Suppose we are interested in the average bill length AND average bill depth of all Adelie penguins:

```
penguins %>%
  filter(species == "Adelie") %>%
  summarize(average_bill_lenth = mean(bill_length_mm),  #<1>
            average_bill_depth = mean(bill_depth_mm))   #<2>
```

① calculate the average bill length; save this as `average_bill_length`
② calculate the average bill depth; save this as `average_bill_depth`

```
# A tibble: 1 x 2
  average_bill_lenth average_bill_depth
               <dbl>              <dbl>
1               38.8               18.3
```

Typically, we seperate each calculation with a new line to keep things pretty. These new values will print out on the same table.

There are lots of other functions available:

- `min`: minimum value
- `max`: maximum value
- `mean`: average or mean value
- `median`: median value
- `var`: variance
- `sd`: standard deviation
- `n`: count or number of values
- `n_distinct`: counts number of distinct values

Suppose we are interested in the average bill length AND the median bill length of all Adelie penguins:

```
penguins %>%
  filter(species == "Adelie") %>%
  summarise(average_bill_lenth = mean(bill_length_mm),
            median_bill_length = median(bill_length_mm))
```

```
# A tibble: 1 x 2
  average_bill_lenth median_bill_length
               <dbl>              <dbl>
1               38.8               38.8
```

`group_by()`

Let's say we were interested in the average bill length and bill depth of all penguin species in this dataset. We could repeat this for the other species (Gentoo and Chinstrap). This would be a fair amount of work AND the results would not end up in the same table.

OR we could use the `group_by` command!

```
penguins %>%
  group_by(species) %>% #<1>
  summarise(average_bill_lenth = mean(bill_length_mm),
            average_bill_depth = mean(bill_depth_mm))
```

① Repeats the calculate below for each different species.

```
# A tibble: 3 x 3
  species   average_bill_lenth average_bill_depth
  <fct>                  <dbl>              <dbl>
1 Adelie                  38.8               18.3
2 Chinstrap               48.8               18.4
3 Gentoo                  47.6               15.0
```

Multiple Groups

Suppose we wish to have the average bill length and average bill depth broken down by sex AND species:

```
penguins %>%
  group_by(species, sex) %>%
  summarise(average_bill_length = mean(bill_length_mm),
            average_bill_depth = mean(bill_depth_mm))
```

```
# A tibble: 6 x 4
# Groups:   species [3]
  species   sex    average_bill_length average_bill_depth
  <fct>     <fct>                <dbl>              <dbl>
1 Adelie    female                37.3               17.6
2 Adelie    male                  40.4               19.1
3 Chinstrap female                46.6               17.6
4 Chinstrap male                  51.1               19.3
5 Gentoo    female                45.6               14.2
6 Gentoo    male                  49.5               15.7
```

`across()` (Optional)

If you wish to apply the same calculation to many columns, you may wish to check out the `across` function.

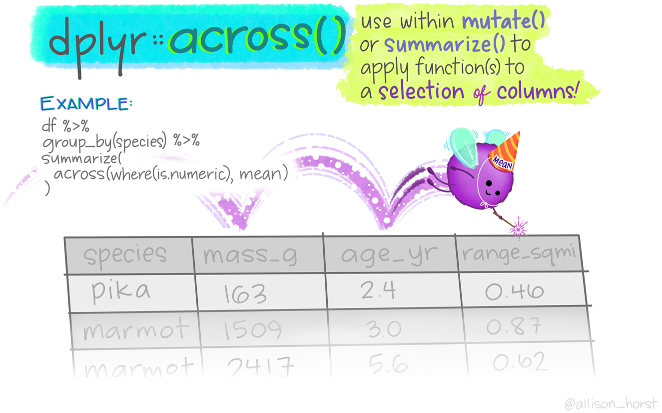

Figure 1: Artwork by @allisonhorst

```r
penguins %>%
  group_by(species, sex) %>%
  summarise(across(where(is.numeric), mean))
```

```
# A tibble: 6 x 7
# Groups:   species [3]
  species sex    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g  year
  <fct>   <fct>           <dbl>         <dbl>             <dbl>       <dbl> <dbl>
1 Adelie  fema~            37.3          17.6              188.       3369. 2008.
2 Adelie  male             40.4          19.1              192.       4043. 2008.
3 Chinst~ fema~            46.6          17.6              192.       3527. 2008.
4 Chinst~ male             51.1          19.3              200.       3939. 2008.
5 Gentoo  fema~            45.6          14.2              213.       4680. 2008.
6 Gentoo  male             49.5          15.7              222.       5485. 2008.
```

Recall: `mutate()`

The mutate function allows you create a new column which is a function of other columns. This can be useful to converting units.

For example, let's calculate create a new column which displays the body length weight in pounds (lbs) instead of grams. Recall: to convert from grams to pounds we need to multiply by 0.00220462

```
penguins <- penguins %>%
  mutate(body_mass_lbs = body_mass_g*0.00220462) #<1>
```

① Creates a new column in the penguins dataset called **`body_mass_lbs`** calculated by taking the value of the body mass (in g) and multiplying by 0.00220562.

`case_when()`

Case when can be used in combination with `mutate` to create a new column with a categorical variable conditional on the values in another column.

```
df %>%
  mutate(danger = case_when(type == "kraken" ~ "extreme!",
                            TRUE ~ "high"))
```

type	age	danger
kraken	baby	extreme!
dragon	adult	high
cyclops	teen	high
kraken	adult	extreme!
dragon	teen	high

Figure 2: Artwork by @allisonhorst

For example:

```
penguins <- penguins %>%
  mutate(penguin_length_cat = case_when(bill_length_mm > 50 ~ 'whoa! huge bill!', TRUE ~ '
```

Tip

For those of you who have taken a computer science class before, you may notice that `case_when` is similar to using an `ifelse` statement. You can also use `ifelse` in R if you'd prefer!

```
penguins <- penguins %>%
  mutate(penguin_length_cat = ifelse(bill_length_mm > 50, 'whoa! huge bill!', '--' ))
```

Brain Break: Jingjing!

https://youtu.be/oks2R4LqWtE