# 02_DataWrangling

**Professor Becca Selden**

**2025-01-29**

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(here)
```

```
## Warning in readLines(f, n): line 1 appears to contain an embedded nul
```

```
## here() starts at /Volumes/Selden_SSD/Wellesley Courses/BISC198_S25/BISC198_S25_R
```

You see that in the first code chunk above we loaded the libraries we'll use for this exercise.

## Load data

We are going to load in our cleaned penguins dataset from from last time using `read_csv()`. Remember that this dataset is in the "Output" folder. The syntax is `data <- read_csv(here::here("path_to_file"))`

```
penguins <- read_csv(here::here("Output/clean_penguin.csv"))
```

```
## Rows: 344 Columns: 17
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (9): study_name, species, region, island, stage, individual_id, clutch_...
## dbl  (7): sample_number, culmen_length_mm, culmen_depth_mm, flipper_length_m...
## date (1): date_egg
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

## Adding columns (Driver 1)

We are going to create a short species name `spp` from the existing `species` column using `mutate()` and the function `case_when()`. We add a column using the "pipe" `%>%` by taking the original dataset "then" adding a column using mutate, and we overwrite the original with the version with the new column name by assigning it to the same name. The syntax is `dataset <- dataset %>% mutate(new_col=some_function(some_column_in_data))`

To use `case_when` we use a conditional statement. Conditionals include: - `==` is equal to
- `<=` less than or equal to
- `>=` greater than or equal to

- `%in%` is in the set of

We are going to use `==` to assign this new variable to a certain value when the old variable is equal to X. We are going to do this for each species so we need to know what the original `species` variable calls each.

```
unique(penguins$species)
```

```
## [1] "Adelie Penguin (Pygoscelis adeliae)"
## [2] "Gentoo penguin (Pygoscelis papua)"
## [3] "Chinstrap penguin (Pygoscelis antarctica)"
```

You can see that there are three species with both their common name and their scientific name in parentheses. We will shorten this to just "Adelie" "Gentoo" and "Chinstrap". We can copy the output for species into the code below that we'll use to reclassify with `case_when()`

The syntax for using `case_when()` is `case_when(first_condition_is_true ~ "new category1", second_conditions_is_true ~ "new category2")`

```
penguins <- penguins %>%
  mutate(spp=case_when(species=="Adelie Penguin (Pygoscelis adeliae)" ~ "Adelie",
                       species=="Gentoo penguin (Pygoscelis papua)" ~ "Gentoo",
                       species=="Chinstrap penguin (Pygoscelis antarctica)" ~ "Chinstrap"))
```

We can also create a new column using `ifelse()`. This syntax is helpful if you want to rename only some levels of the variable and everything other than that condition will get some other name. We'll create a new column that classifies the penguins as female or not. We could use this later to subset by this variable using `filter()`. With `ifelse()` the syntax is `ifelse(condition, value_if_true, value_if_false)`

```
penguins <- penguins %>%
  mutate(female_penguin=ifelse(sex=="FEMALE", "yes", "no"))
```

## Practice: Add a new column for month and year from the `date_egg` column

1. Load the lubridate package using `library()`.
2. Add a new column for month using the functions `month()` and year using `year()` in one mutate. The syntax for making multiple variables in one mutate is to separate them with a comma: `data <- data %>% mutate(variable1=some_function(col_of_interest), variable2=some_function(col_of_interest))`. For this case `col_of_interest` is `date_egg`

# Create subsets of data using `filter()`

We will subset the data to just Adelie penguins using `filter(condition_that_is_true)`. Then print out the first few rows of the adelie dataset by printing the name of the subsetted data.

```
adelie <- penguins %>%
  filter(spp=="Adelie")
adelie
```

```
## # A tibble: 152 × 19
##    study_name sample_number species            region island stage individual_id
##    <chr>              <dbl> <chr>              <chr>  <chr>  <chr> <chr>
##  1 PAL0708                1 Adelie Penguin (P… Anvers Torge… Adul… N1A1
##  2 PAL0708                2 Adelie Penguin (P… Anvers Torge… Adul… N1A2
##  3 PAL0708                3 Adelie Penguin (P… Anvers Torge… Adul… N2A1
##  4 PAL0708                4 Adelie Penguin (P… Anvers Torge… Adul… N2A2
##  5 PAL0708                5 Adelie Penguin (P… Anvers Torge… Adul… N3A1
##  6 PAL0708                6 Adelie Penguin (P… Anvers Torge… Adul… N3A2
##  7 PAL0708                7 Adelie Penguin (P… Anvers Torge… Adul… N4A1
##  8 PAL0708                8 Adelie Penguin (P… Anvers Torge… Adul… N4A2
##  9 PAL0708                9 Adelie Penguin (P… Anvers Torge… Adul… N5A1
## 10 PAL0708               10 Adelie Penguin (P… Anvers Torge… Adul… N5A2
## # ℹ 142 more rows
## # ℹ 12 more variables: clutch_completion <chr>, date_egg <date>,
## #   culmen_length_mm <dbl>, culmen_depth_mm <dbl>, flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, delta_15_n_o_oo <dbl>, delta_13_c_o_oo <dbl>,
## #   comments <chr>, spp <chr>, female_penguin <chr>
```

### Practice: Filter to Gentoo

## Summarizing Data (Driver 2)

We will get the mean, standard deviation, and the number of observations of flipper length by the new shortened species name `spp`, and then calculate the standard error of the mean using `group_by()` and `summarize()`. We will save this summmary as a new object called `summ_flipp`. We will use the following functions:
- `mean(variable, na.rm=T)` we use na.rm=T to remove any NAs that are in our variable that would otherwise make our mean `NA`
- `sd(variable, na.rm=T)` calculate standard deviation of our variable.
- `n()` count the number of observations in our group.
The variable of interest here is `flipper_length_mm`.

```
summ_flipp <- penguins %>%
  group_by(spp) %>%
  summarize(mean_flipp=mean(flipper_length_mm, na.rm=T),
            sd_flipp=sd(flipper_length_mm, na.rm=T),
            n_penguin=n())
```

Then to calculate the standard error we will manually divide the standard deviation by the square root of the number of observations. The syntax is `se_var=sd_var/sqrt(n_obs)`

```
summ_flipp <- summ_flipp %>%
  mutate(se_flipp=sd_flipp/sqrt(n_penguin))
```

Finally we'll print out the summary object so we can see the results. We do so just by typing the name of the summary object

```
summ_flipp
```

```
## # A tibble: 3 × 5
##   spp       mean_flipp sd_flipp n_penguin se_flipp
##   <chr>          <dbl>    <dbl>     <int>    <dbl>
## 1 Adelie          190.     6.54       152    0.530
## 2 Chinstrap       196.     7.13        68    0.865
## 3 Gentoo          217.     6.48       124    0.582
```

### Add summary stats into raw dataset using `mutate()`

If we want to add a summary stat into the raw dataset we can use similar syntax, but using `mutate()` rather than `summarize()`

```
penguins <- penguins %>%
  group_by(spp) %>%
  mutate(mean_flipp=mean(flipper_length_mm, na.rm=T))
```

## Filter data to those greater than the mean

We will then filter to those penguins that had a larger flipper size than the mean for their species, and we'll save this new object as `large_flipp`

```
large_flipp <- penguins %>%
  filter(flipper_length_mm >= mean_flipp)
```

## Practice: Summarize body mass by species

1. Summarize body mass by species using `group_by()` and `summarize()` with `mean()` , `sd()` and `n()` .
2. Add the standard error using `mutate()` .
3. Print out the summary object

## Practice: Add mean body mass by species to the full dataset

1. Add the mean body mass by species using `group_by()` and `mutate()` with `mean()`

## Practice: Filter to penguins with larger than average body mass

1. Use `filter()` with a conditional that `body_mass_g` is greater than the average for each speices that you calculated

# Application: Summarize E. coli by beach & filter data to worst beaches (Driver 3)

1. Read in the clean_sydneybeaches.csv file from the "Output" folder.
2. Calculate the mean, standard deviation, and maximum E. coli levels in the column `enterococci_cfu_100ml` and the number of observations by `site` using `group_by()` and `summarize()` and save the summary object as `summ_beaches` .
3. Add a column to the summary object with the standard error of the E. coli levels called `se_ecoli` using `mutate()` by dividing the standard deviation by the square root of the number of observations. Print out the summary object
4. In the printout of the summary object, examine the column you created for the maximum E. coli levels. Identify the name of the beach that has the highest maximum E.coli levels. Create an object `worst` that is the `site` name of this beach using the syntax `worst <- c("name_of_worst_beach")`
5. Filter the full dataset to just the data from the "worst" beach and save this object as `worst_sydney_beach` by filtering with the condition `site == worst` . Print the first few rows of the data for the worst Sydney beach.