# Week 2: Wrangling Basics – Select, Filter, Mutate

Ronnie Steinitz

2025-09-25

## Skills Learning – Lecture

Last week, we explored the structure of our dataset, checked data types, and built some basic plots.
This week, we'll begin *wrangling* data: learning how to **select**, **filter**, and **mutate** columns and rows using the `dplyr` package.

These are essential tools because:
- `select()` helps you *choose only the columns you need*
- `filter()` helps you *focus on specific rows of interest*
- `mutate()` helps you *add new variables or transform existing ones*

Together, they make it easy to organize your data into exactly what you need for analysis.

### 0. Load Required Packages

```
library(tidyverse)
library(janitor)
library(here)
```

### 1. Load Data

Use the same dataset we saved last week, in folder Week 1

We are using a function called `here()`. `here::here()` finds files based on the root of your R Project instead of wherever your .Rmd happens to live. This makes your code more reliable and portable — you can knit from any subfolder and still load data with the same path.

Think of your R Project as your "home address." If you use `here::here()`, you're saying "start from my house, then go to Week 1, then open this file."

Without it, R might start from whatever room you happen to be standing in (the folder where your script lives), and then it gets confused if the file isn't there.

So `here::here()` always starts from the same "home base," (your working directory) which keeps paths consistent when knitting or sharing code.

```
data_raw <- read_csv(here("Week 1/Palmer Penguins Raw.csv"))
```

```
## Rows: 344 Columns: 17
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## chr (10): studyName, Species, Region, Island, Stage, Individual ID, Clutch C...
## dbl  (7): Sample Number, Bill Length (mm), Bill Depth (mm), Flipper Length (...
##
## 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.
```

## 2. Pipes

The **pipe** (%>%) lets us *chain together commands* so they read like a sentence.
It allows us to write our code in a sequence that flows from left to right, instead of nesting functions inside one another.

```r
# Instead of nesting:
data <- clean_names(data_raw) # Clean column names

# We can write:
data <- data_raw %>%
  clean_names() # Clean column names using a pipe ("and then")

# Quick look
glimpse(data)
```

```
## Rows: 344
## Columns: 17
## $ study_name        <chr> "PAL0708", "PAL0708", "PAL0708", "PAL0708", "PAL0708~
## $ sample_number     <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ species           <chr> "Adelie Penguin (Pygoscelis adeliae)", "Adelie Pengu~
## $ region            <chr> "Anvers", "Anvers", "Anvers", "Anvers", "Anvers", "A~
## $ island            <chr> "Torgersen", "Torgersen", "Torgersen", "Torgersen", ~
## $ stage             <chr> "Adult, 1 Egg Stage", "Adult, 1 Egg Stage", "Adult, ~
## $ individual_id     <chr> "N1A1", "N1A2", "N2A1", "N2A2", "N3A1", "N3A2", "N4A~
## $ clutch_completion <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "No", "No"~
## $ date_egg          <chr> "11/11/07", "11/11/07", "11/16/07", "11/16/07", "11/~
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
## $ flipper_length_mm <dbl> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
## $ body_mass_g       <dbl> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
## $ sex               <chr> "MALE", "FEMALE", "FEMALE", NA, "FEMALE", "MALE", "F~
## $ delta_15_n_o_oo   <dbl> NA, 8.94956, 8.36821, NA, 8.76651, 8.66496, 9.18718,~
## $ delta_13_c_o_oo   <dbl> NA, -24.69454, -25.33302, NA, -25.32426, -25.29805, ~
## $ comments          <chr> "Not enough blood for isotopes.", NA, NA, "Adult not~
```

Read %>% as **"and then"**

> Shortcut for pipes: Ctrl + Shift + M

## 3. Select( ) – Choosing Columns

Sometimes we don't want all the variables.
For example, what if we only care about **species, island, and flipper_length_mm**?

```r
# let's see the dataset structure
head(data)
```

```
## # A tibble: 6 x 17
##   study_name sample_number species             region island stage individual_id
##   <chr>              <dbl> <chr>               <chr>  <chr>  <chr> <chr>
## 1 PAL0708                1 Adelie Penguin (Py~ Anvers Torge~ Adul~ N1A1
## 2 PAL0708                2 Adelie Penguin (Py~ Anvers Torge~ Adul~ N1A2
## 3 PAL0708                3 Adelie Penguin (Py~ Anvers Torge~ Adul~ N2A1
## 4 PAL0708                4 Adelie Penguin (Py~ Anvers Torge~ Adul~ N2A2
## 5 PAL0708                5 Adelie Penguin (Py~ Anvers Torge~ Adul~ N3A1
## 6 PAL0708                6 Adelie Penguin (Py~ Anvers Torge~ Adul~ N3A2
## # i 10 more variables: clutch_completion <chr>, date_egg <chr>,
## #   bill_length_mm <dbl>, bill_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>
```

```r
# we can't see all the columns, and we don't care about some of the columns...


# So, we can do the same thing, but limiting only to specific columns
data2 <- data %>%
  dplyr::select(species, island, flipper_length_mm)

# we made a new object here (a new dataset) copied from the old one, so that we are not over-writing 'd


# View(data2) # only includes columns 'species,' 'island,' and 'flipper_length_mm')



# We can also exclude columns using a `-` (minus) sign
data3 <- data %>%
  dplyr::select(-study_name, -comments)

# View(data3) # does not have columns 'study_name' and 'comments')
```

So now we know how to `select()` which columns we would like to focus on.
But what if we want exclude certain observations? For example, limiting to only a particular year or study group?

## 4. Filter( ) – Choosing Rows

Let's see how we can only target rows from Dream island

```r
data4 <- data %>%
  dplyr::filter(island == "Dream")

# only 124 observations!


# What about penguins with **flipper length greater than 200 mm**?
```

```r
# ============================================================
# Q: How many observations of flipper_length_mm > 200?
# A: 148
# ============================================================

data4 <- data %>%
  dplyr::filter(flipper_length_mm > 200)

# 148 observations


# you can also combine filter conditions:
data5 <- data %>% filter(flipper_length_mm > 200, island == "Dream")
```

**Logical operators you can use in filter():**
- == equal to
- != not equal to
- > greater than
- < less than
- & and
- | or

**Example - Select, Filter**

```r
# If I give you a line of code like this:
test1 <- arrange(select(filter(clean_names(data_raw), island == "Dream"), species, island, flipper_leng

# can you tell me what it does???
# IMPORTANT: the functions start from the inside and go outward! Find the most central parentheses and


# And, to write it, you would need to put lots of parentheses one inside the other which can lead to er
test1 <- clean_names(data_raw)
test1 <- filter(clean_names(data_raw), island == "Dream")
test1 <- select(filter(clean_names(data_raw), island == "Dream"), species, island, flipper_length_mm)
test1 <- arrange(select(filter(clean_names(data_raw), island == "Dream"), species, island, flipper_leng

# Nested functions (harder to read!)


# You can just use a pipe and read it like a sentence:
# Same thing, much clearer with a pipe
test2 <- data_raw %>%
  clean_names() %>%
  dplyr::filter(island == "Dream") %>%
  dplyr::select(species, island, flipper_length_mm) %>%
  dplyr::arrange(flipper_length_mm)

# functions start from top and move downward
```

## 5. Mutate( ) – Changing Variables

Let's create a new column for **body mass in kilograms**:
### 5.1 Create new variable

```
data2 <- data %>%
  mutate(body_mass_kg = body_mass_g / 1000) %>%
  select(species, body_mass_g, body_mass_kg)


data <- data %>%
  mutate(name_first = word(species, 1),
         name_second = word(species, 2))


data <- data %>%
  mutate(size_class = ifelse(flipper_length_mm>200, "Large", "Small"))

# notice that you are overwriting the previous "data" so you must be careful with that. If you mess up,
```

## 6. Visualizing Subsets

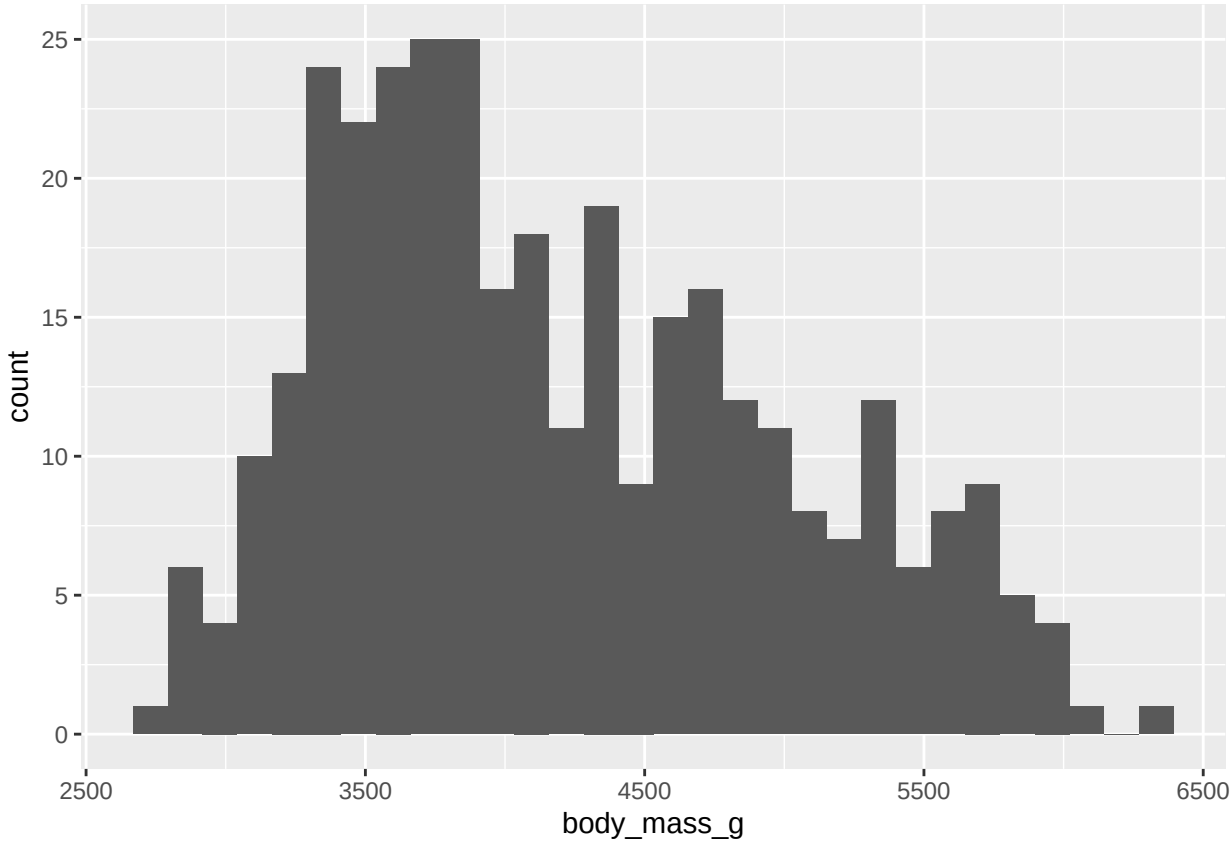
Now let's practice combining wrangling with visualization.

### Histogram of Large Penguins Only

```
# let's see the distribution of measurements
ggplot(data, aes(x = body_mass_g)) +
  geom_histogram()
```

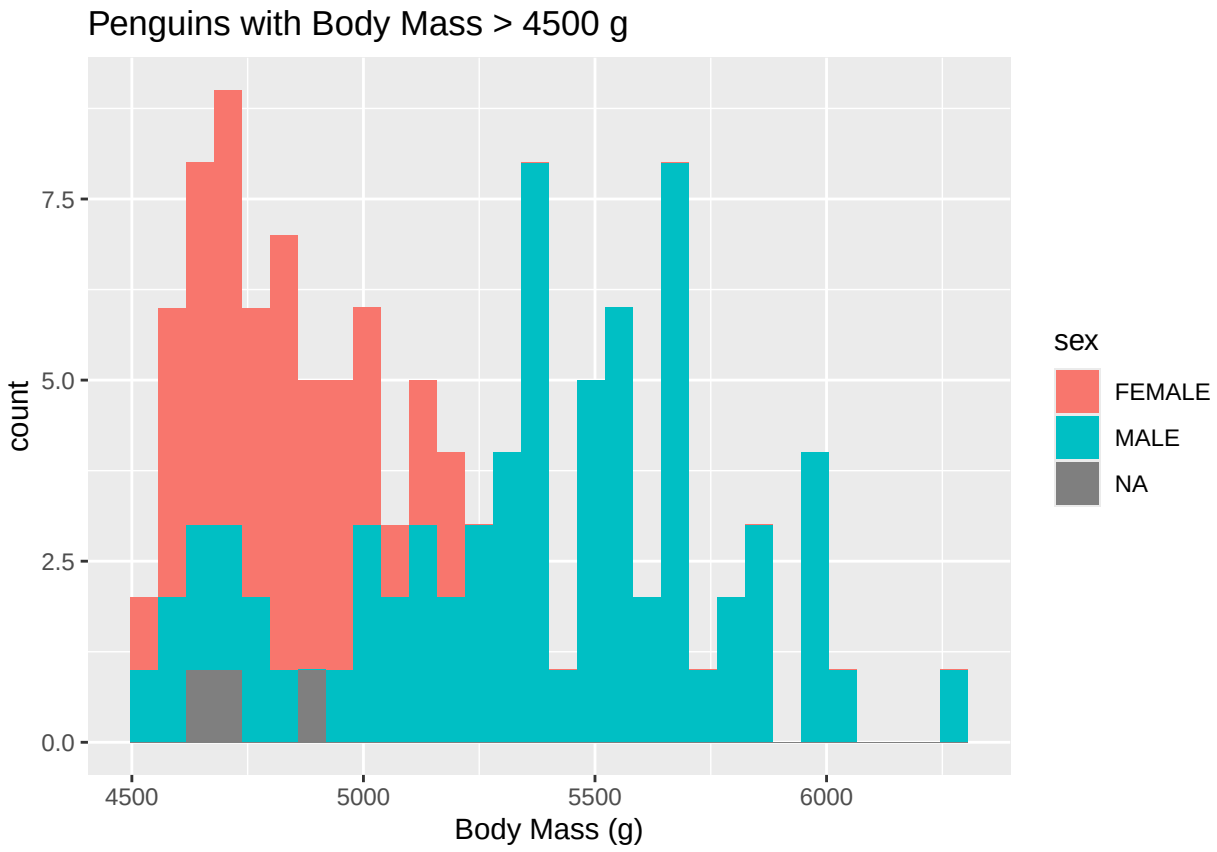
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



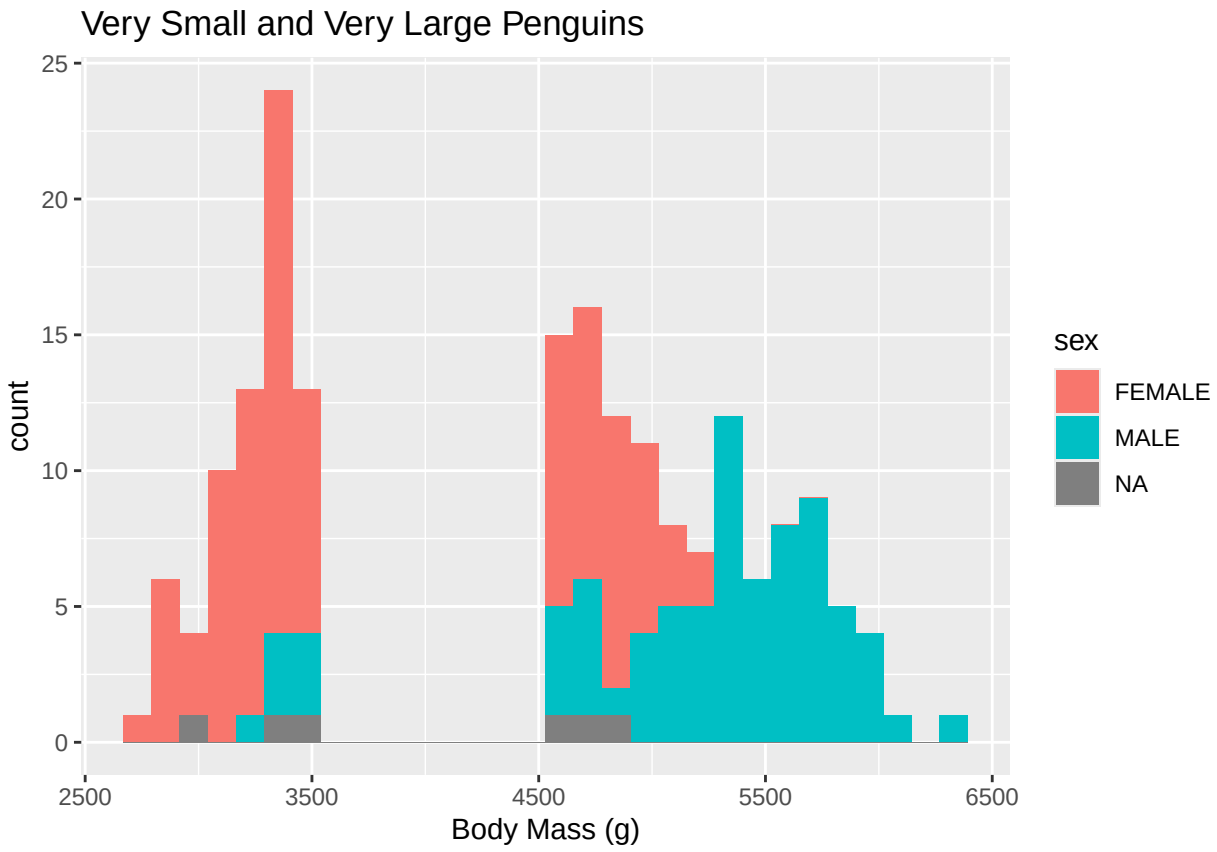

```
# ok, let's limit it to only large-flippered penguins
ggplot(data %>% filter(body_mass_g > 4500),
  aes(x = body_mass_g)) +
  #geom_histogram(fill = "orange", color = "firebrick") +
  geom_histogram(aes(fill = sex)) +
  labs(title = "Penguins with Body Mass > 4500 g", x = "Body Mass (g)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



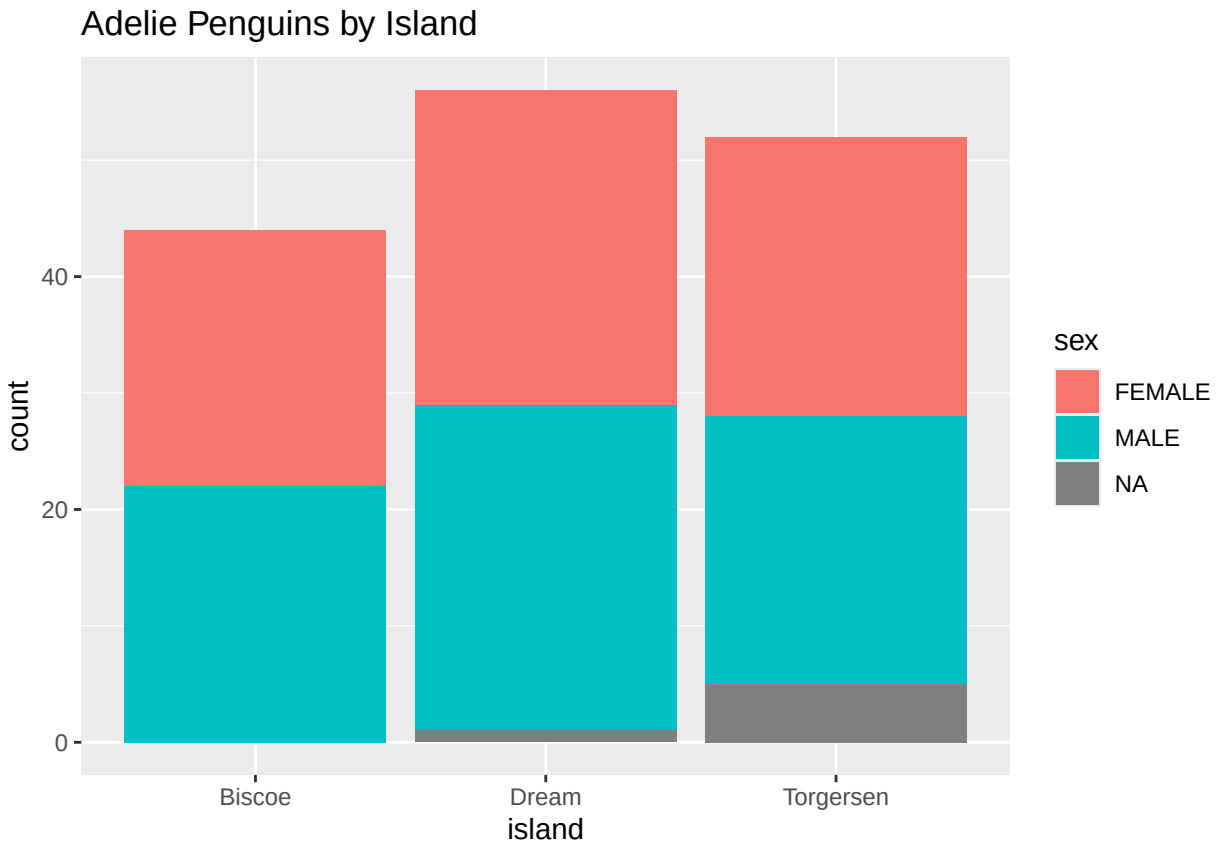

```
ggplot(data %>% filter(body_mass_g > 4500 | body_mass_g < 3500),
  aes(x = body_mass_g)) + # can also add 'filter !is.na(sex)
  #geom_histogram(fill = "orange", color = "firebrick") +
  geom_histogram(aes(fill = sex)) +
  labs(title = "Very Small and Very Large Penguins", x = "Body Mass (g)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

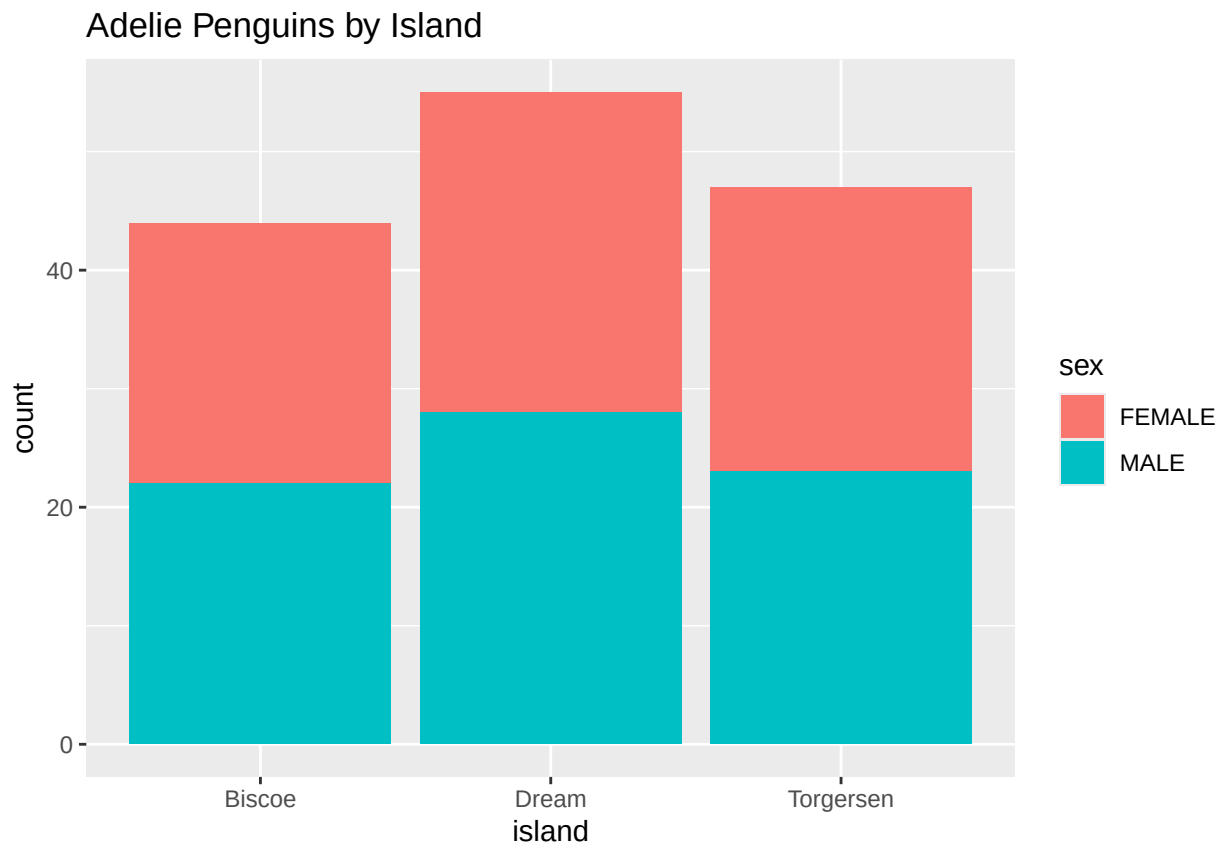



**Bar Plot by Island (Filtered to Adelie Only)**

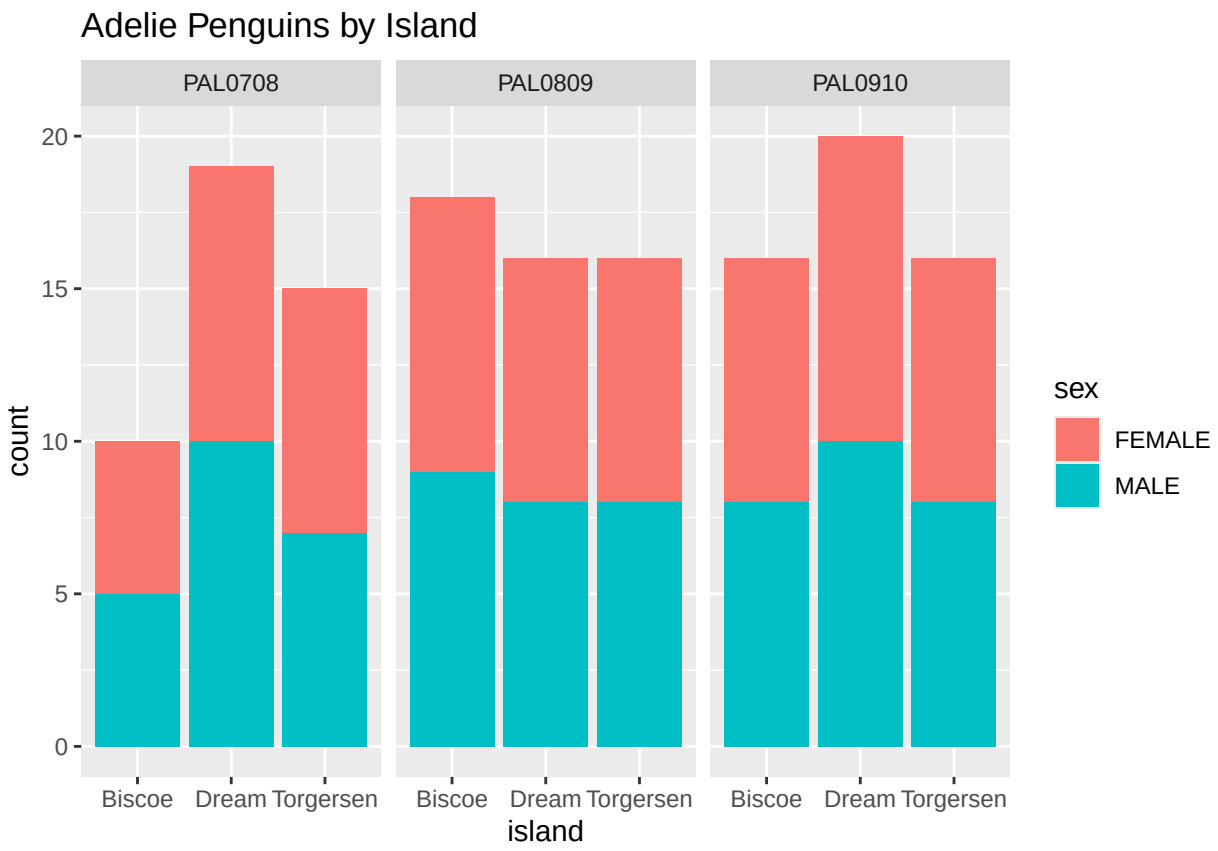
```
ggplot(data %>% filter(name_first == "Adelie"),
       aes(x = island, fill = sex)) +
  geom_bar() +
  labs(title = "Adelie Penguins by Island")
```




```
# what if we want to exclude NAs?
ggplot(data %>% filter(name_first == "Adelie" & !is.na(sex)),
       aes(x = island, fill = sex)) +
  geom_bar() +
  labs(title = "Adelie Penguins by Island")
```




```
# FACET_WRAP!!!
# FACET_WRAP!!!
# FACET_WRAP!!!
ggplot(data %>% filter(name_first == "Adelie" & !is.na(sex)),
       aes(x = island, fill = sex)) +
  geom_bar() +
  labs(title = "Adelie Penguins by Island") +
  facet_wrap(~study_name)
```




# Skills Application – Lab

## 7. Practice Prompts

Each student should now begin a **new R Markdown file** in the same project and try the following with either their own dataset or a provided one:

If you are working on **your own dataset**, then create a Markdown file to which you will be using and continuously adding.
- Name it: *Lastname_Firstname_Data*

If you are working on a **example dataset**, then create a Markdown file just for this week.
- Name it: *Lastname_Firstname_Week2*

0. Create new markdown file. Load packages. Import the dataset into the environment and save as an object (e.g., `data`).
1. Identify which variables have missing values.
2. Check for misclassified columns (e.g., text stored as factors, numbers stores as text). Get summary statistics for two variables of different classes.
3. Convert one column to the correct type.
4. Use `select()` to keep only 3 columns of your choice.
5. Use `filter()` to create a subset with one species of penguin.

6. Create a **new variable** using `mutate()` (for example, converting grams → kilograms, or categorize numeric values).
7. Make a histogram of a numeric variable from your subset. Add a title and axis labels to your plot.
8. Knit the script into an HTML file, which will be saved to your working directory.

## Optional – Look Ahead

Next week, we'll go deeper into wrangling (grouped summaries, joins). If you want to try early:

```
# Try grouping and summarizing!
data %>%
  group_by(species) %>%
  summarise(avg_mass = mean(body_mass_g, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   species                                   avg_mass
##   <chr>                                        <dbl>
## 1 Adelie Penguin (Pygoscelis adeliae)          3701.
## 2 Chinstrap penguin (Pygoscelis antarctica)    3733.
## 3 Gentoo penguin (Pygoscelis papua)            5076.
```

```
# Try arranging data (sorting)
data %>%
  arrange(desc(flipper_length_mm)) %>%
  head()
```

```
## # A tibble: 6 x 20
##   study_name sample_number species             region island stage individual_id
##   <chr>              <dbl> <chr>               <chr>  <chr>  <chr> <chr>
## 1 PAL0809               64 Gentoo penguin (Py~ Anvers Biscoe Adul~ N19A2
## 2 PAL0708                2 Gentoo penguin (Py~ Anvers Biscoe Adul~ N31A2
## 3 PAL0708               34 Gentoo penguin (Py~ Anvers Biscoe Adul~ N56A2
## 4 PAL0809               66 Gentoo penguin (Py~ Anvers Biscoe Adul~ N20A2
## 5 PAL0809               76 Gentoo penguin (Py~ Anvers Biscoe Adul~ N56A2
## 6 PAL0910               90 Gentoo penguin (Py~ Anvers Biscoe Adul~ N14A2
## # i 13 more variables: clutch_completion <chr>, date_egg <chr>,
## #   bill_length_mm <dbl>, bill_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>, name_first <chr>, name_second <chr>, size_class <chr>
```

This Week's Takeaway: We learned to use **pipes**, **select**, **filter**, **mutate**, and **case_when** to wrangle datasets into the form we need.