# Analysis of Environmental Data Lab

## Lab 06: Introduction to Inference

## Learning objectives, technical skills and concepts:

- Sampling distribution: sample standard deviation vs standard error
- Building custom functions
- T-tests
- Resampling
- R formula notation
- R object structure

## Sampling Distribution

The building block of the sampling distribution is the Standard Error.

## Formula for the standard error of the mean

> The sample standard error of the mean is the sample standard deviation divided by the square root of the sample size

$SSE_m = \frac{s_x}{\sqrt{n}}$

R does not have a built-in function to calculate the standard error of the mean.

You should write one of your own!

`sse_mean = function(...` You can test it out on the penguin data:

```
require(palmerpenguins)
sse_mean(penguins$bill_depth_mm)
```

```
## [1] 0.1067846
```

Did you get the same result? If you got a NA instead, you may have had an issue with **missing data**.

Try out all of the intermediate steps you used in your calculation to make sure they can deal with NA values.

▶ Here are some hints (click to show/hide)

## Formula Notation

We've been using R's formula notation, but I haven't specifically talked about it

It's a powerful and expressive way to specify a statistical model. We'll be making use of it a lot in the rest of the course.

Here's a nice tutorial you can skip to get the basics

https://www.datacamp.com/community/tutorials/r-formula-tutorial (https://www.datacamp.com/community/tutorials/r-formula-tutorial)

We'll have plenty of opportunity for more discussion and practice.

## Classical p-value interpretation

In lecture, we've discussed a decision criterion in which we set a threshold false-positive rate that we are willing to accept. We then use a test statistic generated from the data to see if our data meet our criterion.

> Classical tests return a p-value

The p-value is what we compare against our decision criterion to decide if we can reject the null hypothesis.

## What does the p-value mean?

One way to interpret a p-value is:

> "How often would I expect to see a result as extreme, or more extreme, if the null hypothesis were true?"

It's a small rephrasing of the definition of a false-positive rate.

In the first t-test below you'll calculate a p-value for the difference in mean flipper length for two penguin species:

- The p-value from the t-test is tiny: 6.049e-08
- Let's just say it is less than 0.001
- The difference between mean flipper lengths for the two species was about 5.9mm

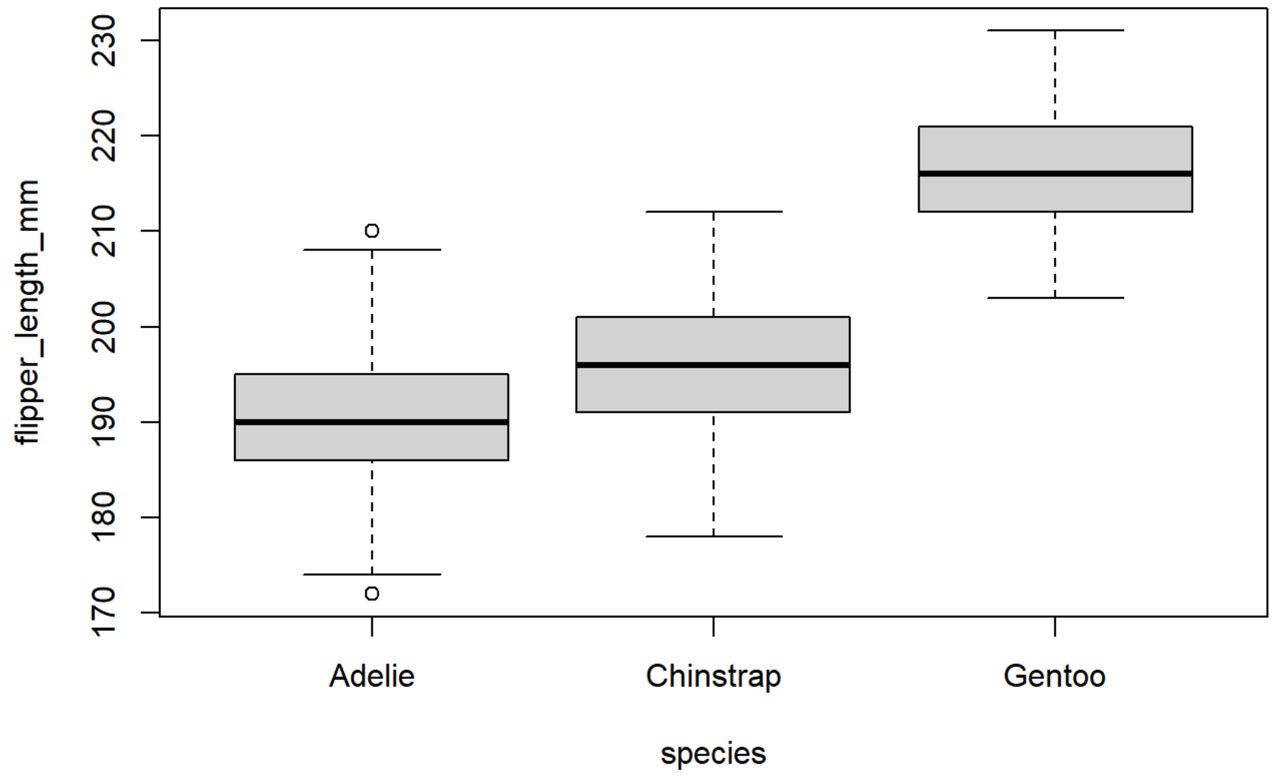
We can interpret that in words as:

> "If I repeatedly sampled flipper lengths for Adelie and Gentoo penguins randomly from a normally-distributed population of measurements, I would expect to observe a difference in mean flipper length of 5.9mm or greater less than 1 in 1000 experiments."

## The penguin data

Consider the flipper lengths of three species of penguins:

```
boxplot(flipper_length_mm ~ species, data = penguins)
```



Flipper lengths of the three penguin species

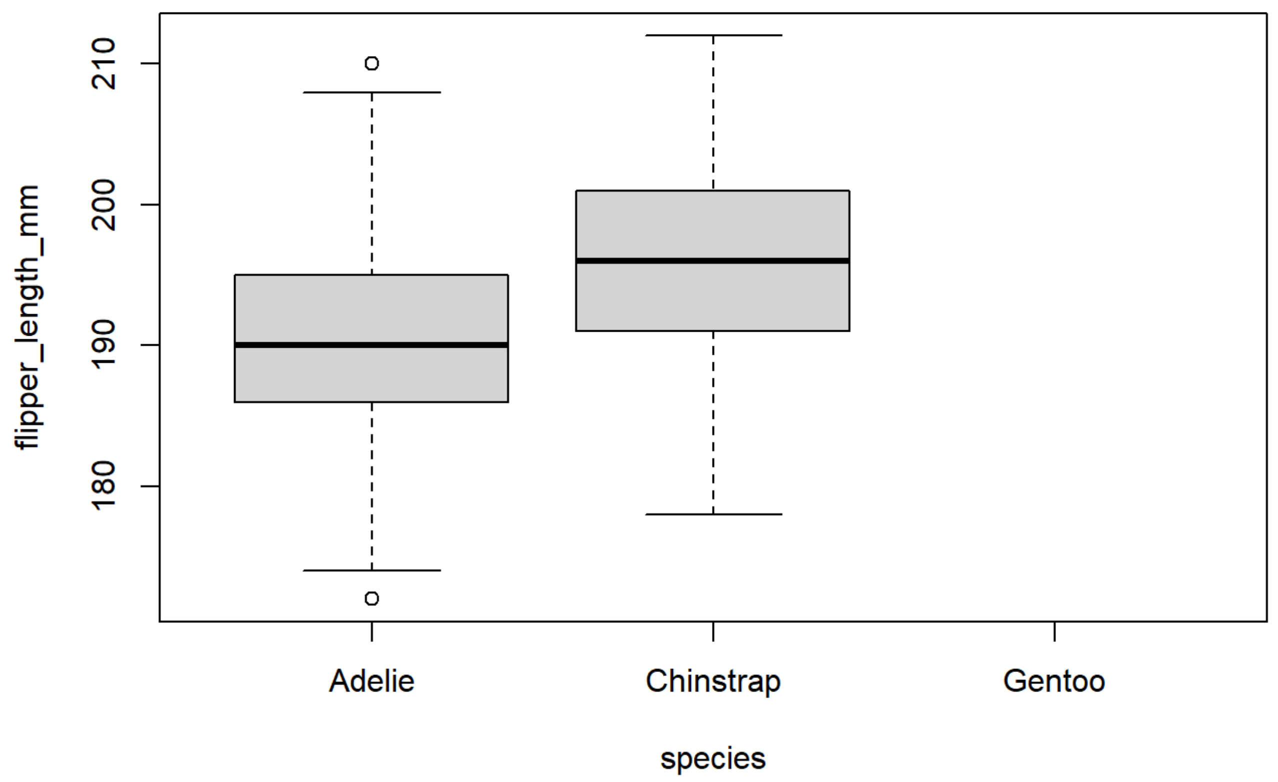
> They look different, but how could we back up our visual assessment?

- We could use a simple Analysis of Variance (ANOVA) to perform a parametric, frequentist analysis.
- We could also do a resampling simulation to estimate how likely the observed differences in flipper length would be if there truly was no difference.

## 2-species data

For most of the lab walkthrough, you'll use a version of the penguins data with one of the species removed:

```
dat_pen = subset(penguins, species != "Gentoo")
boxplot(flipper_length_mm ~ species, data = dat_pen)
```



Flipper length of Chinstrap and Adelie penguins

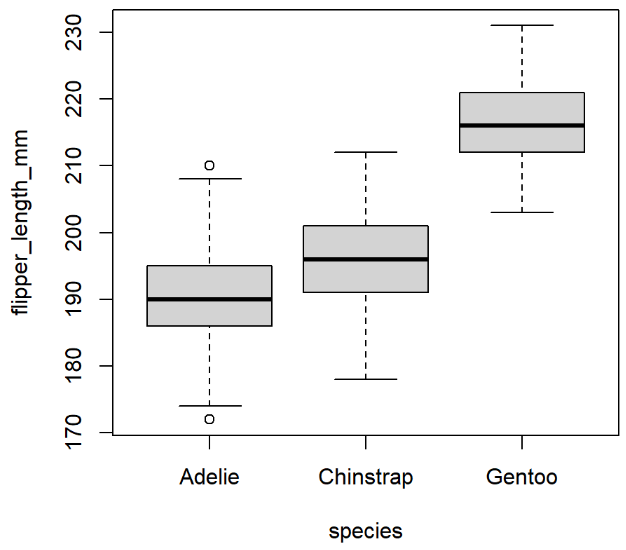
- That’s not what we wanted!

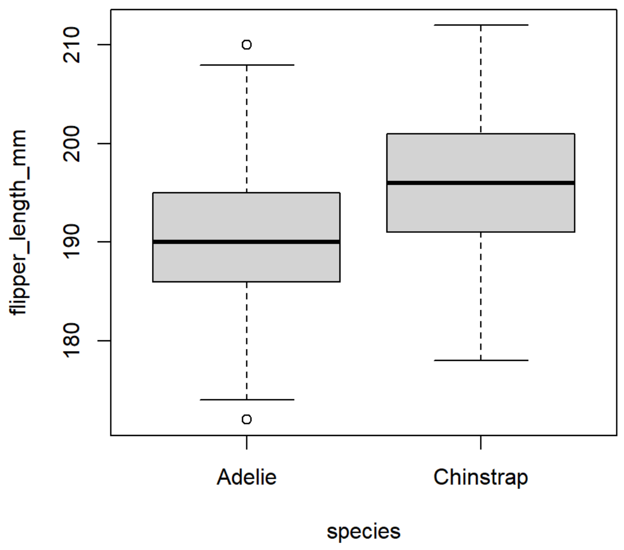
## `droplevels()`

Working with factors and factor levels can be frustrating.

- We can use `droplevels()` to remove unused factor levels from a data.frame:

```
dat_pen = droplevels(subset(penguins, species != "Gentoo"))
{
  par(mfrow = c(1, 2))
  boxplot(flipper_length_mm ~ species, data = penguins)
  boxplot(flipper_length_mm ~ species, data = dat_pen)
}
```





Flipper length of Chinstrap and Adelie penguins - fixed

> Much better!

# Resampling

Most of the lecture course focuses on inference with parametric distributions.

Resampling methods are also important, especially when some of the parametric inference assumptions are not amenable to your data.

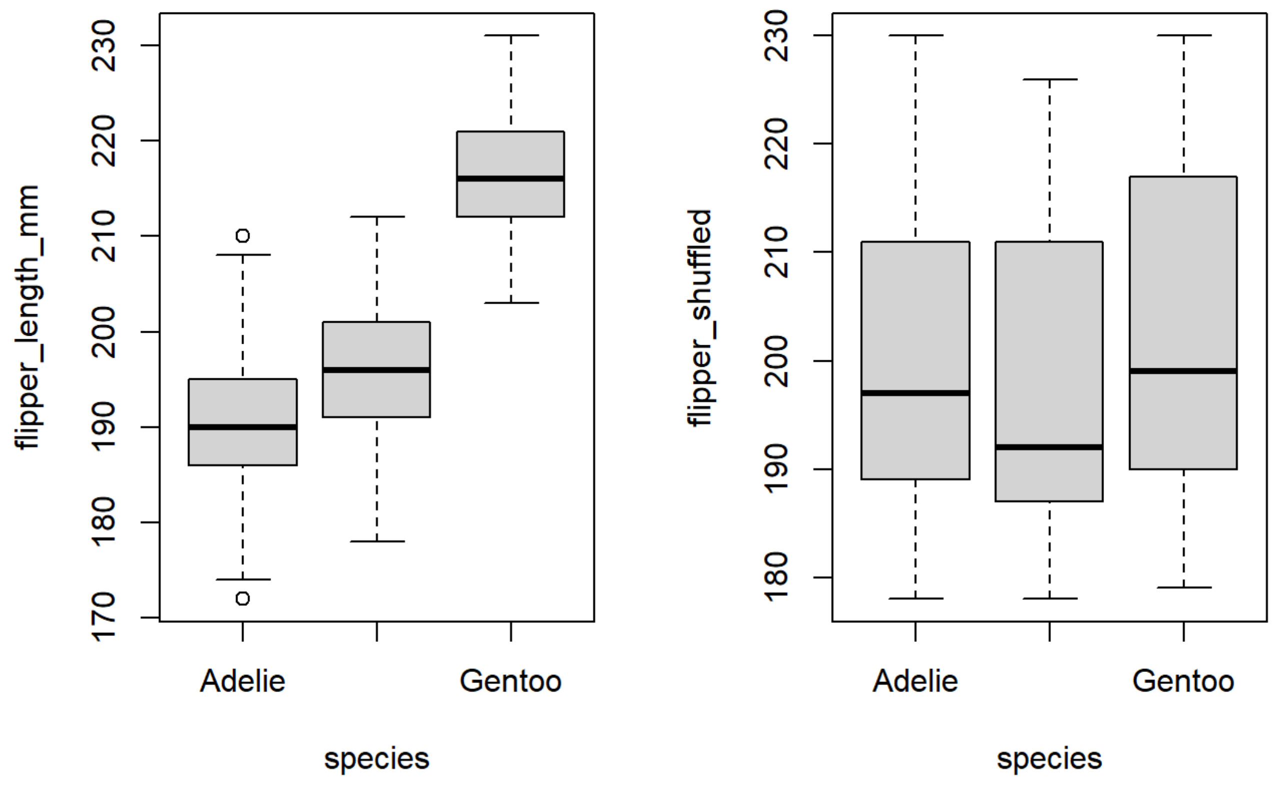
Although resampling methods aren't *parametric* in the sense of a linear regression or t-test, they personify the Frequentist ideal of repeated sampling.

# Resampling with replacement

What if we randomly shuffled the data and made another boxplot?

```
# for reproducibility
set.seed(123)

flipper_shuffled = sample(penguins$flipper_length_mm, replace = TRUE)
par(mfrow = c(1, 2))
boxplot(flipper_length_mm ~ species, data = penguins)
boxplot(flipper_shuffled ~ penguins$species, xlab = "species")
```



Flipper length: original and resampled data

## Monte Carlo Resampling and Null Hypotheses

Note that in the previous resampling, we shuffled the flipper length separately from the species labels. This effectively *breaks the structure* in the data. By that, I mean it destroys any association between flipper length and the species label. This is exactly the spirit of a Frequentist null hypothesis in which:

> The flipper lengths for the penguin species are drawn from the same population of all flipper lengths.

In plain English: There's no difference in flipper length between the species.

If the resampling process reminds you of picking up acorns or writing a book for the Library of Babel, you're on the right track!

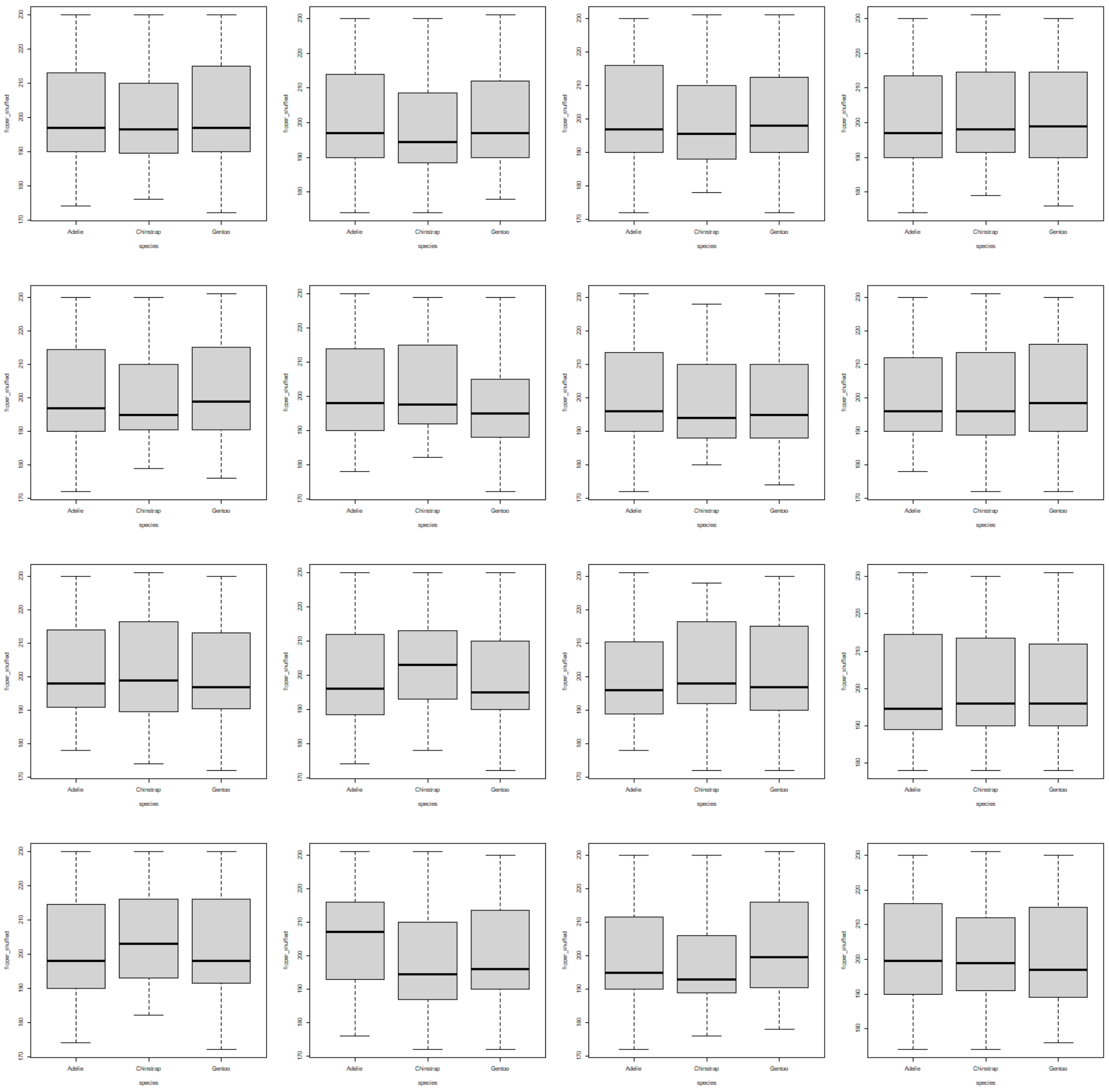
Monte Carlo resampling breaks up associations, therefore it's a great way to simulate what would happen if the null hypothesis were true. In other words, it's a great way to simulate a *null distribution*.

## Bootstrap Resampling and Alternative Hypotheses

Another class of resampling methods is *bootstrapping*. The primary difference between bootstrap and Monte Carlo resampling is that bootstrapping does not destroy the associations in the data. In other words, it samples entire rows of your data. In yet other words, it shuffles rows (with replacement), but does not shuffle the values in columns.

## Repeated Resampling

- What if we repeated this process many times?
    - How often do you think we'd see a pattern like the one in the real data?



Resampled flipper length data

## T-tests - A Frequentist Approach

## Classical t-test: Adelie and Chinstrap penguins

The two-sample t-test is a great starting point to compare the mean values of two groups:

```
t.test(dat_pen$flipper_length_mm ~ dat_pen$species)
```

```
## 
##  Welch Two Sample t-test
## 
## data:  dat_pen$flipper_length_mm by dat_pen$species
## t = -5.7804, df = 119.68, p-value = 6.049e-08
## alternative hypothesis: true difference in means between group Adelie and group
Chinstrap is not equal to 0
## 95 percent confidence interval:
##  -7.880530 -3.859244
## sample estimates:
##    mean in group Adelie mean in group Chinstrap 
##                189.9536                195.8235
```

- The t-test suggests that there is good evidence that the flipper length is different between the two species.

## Two-sample resampling

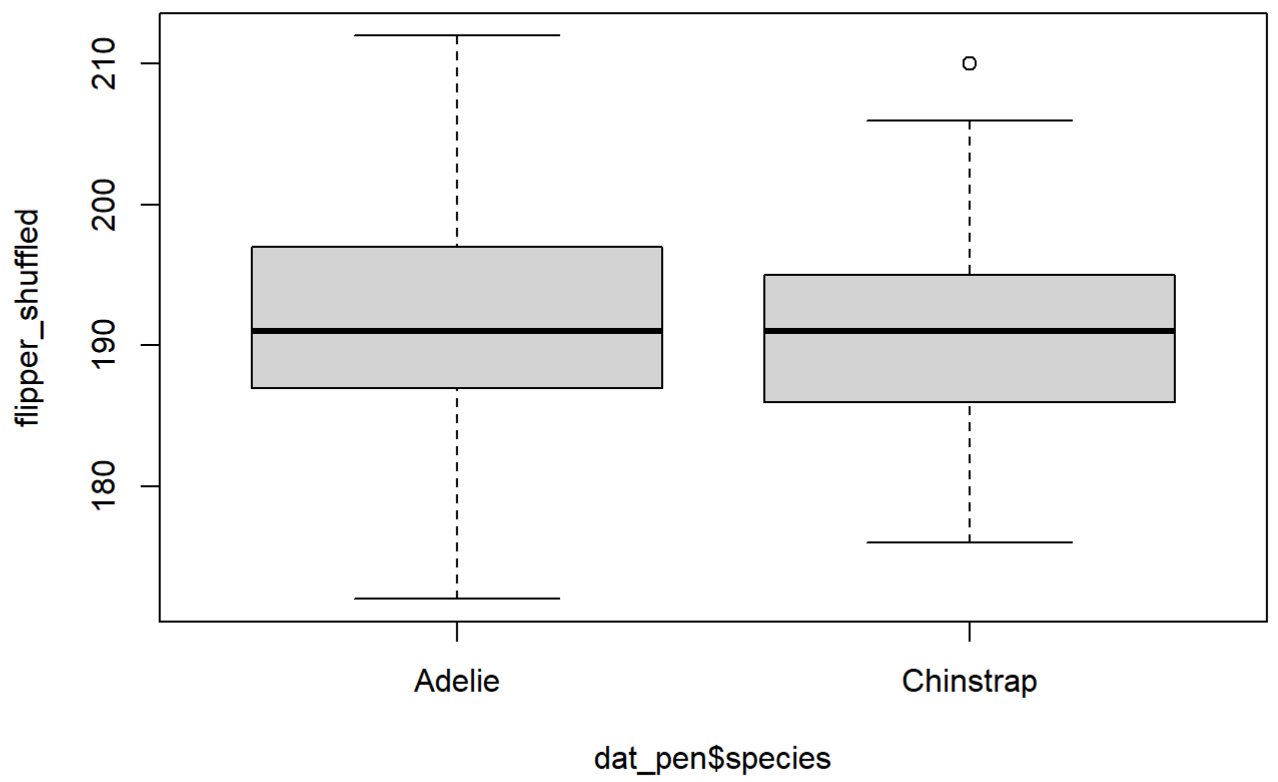
You're going to perform a resampling version of a t-test on penguin flipper length.

> We can resample the flipper lengths *with replacement* using the `sample()` function:

```
# Reset the random number generator state for reproduceablility
set.seed(1)
flipper_shuffled = sample(dat_pen$flipper_length_mm)
```

- Now our data look like this:

```
boxplot(flipper_shuffled ~ dat_pen$species)
```



Shuffled flipper length data

- Visually it seems the difference has disappeared…

## Classical test on resampled data

> What if we reshuffled the data and re-ran the t-test?

```
t_test_1 = t.test(flipper_shuffled ~ dat_pen$species)
t_test_1
```

```
##
##  Welch Two Sample t-test
##
## data:  flipper_shuffled by dat_pen$species
## t = 1.3204, df = 131.52, p-value = 0.189
## alternative hypothesis: true difference in means between group Adelie and group
Chinstrap is not equal to 0
## 95 percent confidence interval:
##  -0.6857072  3.4386530
## sample estimates:
##    mean in group Adelie mean in group Chinstrap
##                192.1974                190.8209
```

The t-test output does not support rejecting a null hypothesis that the two flipper lengths are different between the two species.

## Difference of means

The t-test great for comparing the means of two groups. We can see the group means in the t-test output:

```
t_test = t.test(dat_pen$flipper_length_mm ~ dat_pen$species)
t_test
```

```
##
##  Welch Two Sample t-test
##
## data:  dat_pen$flipper_length_mm by dat_pen$species
## t = -5.7804, df = 119.68, p-value = 6.049e-08
## alternative hypothesis: true difference in means between group Adelie and group
Chinstrap is not equal to 0
## 95 percent confidence interval:
##  -7.880530 -3.859244
## sample estimates:
##    mean in group Adelie mean in group Chinstrap
##                189.9536                195.8235
```

```
t_test$estimate
```

```
##    mean in group Adelie mean in group Chinstrap
##                189.9536                195.8235
```

The difference in means is:

```
diff_observed = round(diff(t_test$estimate), digits = 3)
print(diff_observed, digits = 3)
```

```
## mean in group Chinstrap
##                    5.87
```

- You can check out the `Retrieving named elements` section below to understand the dollar sign usage here.

### Using `aggregate()`

We can also calculate the difference in means using the `aggregate()` function:

```
agg_means = aggregate(
  flipper_length_mm ~ species,
  data = dat_pen,
  FUN = "mean",
  na.rm = TRUE)
diff_observed = diff(agg_means[, 2])

agg_means
diff_observed
```

```
##     species flipper_length_mm
## 1    Adelie          189.9536
## 2 Chinstrap          195.8235
## [1] 5.869887
```

## Sample sizes

The number of individuals of each species in the data are:

```
table(dat_pen$species)
```

```
##
##    Adelie Chinstrap
##       152        68
```

Resampling with replacement is the same thing as randomly sampling 68 flipper lengths in one group and 152 in another.

```
n_1 = 68
n_2 = 152

dat_1 = sample(dat_pen$flipper_length_mm, n_1, replace = TRUE)
dat_2 = sample(dat_pen$flipper_length_mm, n_2, replace = TRUE)

diff_simulated =
  mean(dat_1, na.rm = TRUE) - mean(dat_2, na.rm = TRUE)
```

What was the difference in means for the resampled data?

```
print(c(observed = diff_observed, simulated = diff_simulated))
```

```
##  observed simulated
##  5.869887 -1.334632
```

## Simulation function

I could modify the code I used to resample and wrap it into a function:

```
x = dat_pen$flipper_length_mm
n_1 = 68
n_2 = 152

dat_1 = sample(x, n_1, replace = TRUE)
dat_2 = sample(x, n_2, replace = TRUE)

diff_simulated =
  mean(dat_1, na.rm = TRUE) - mean(dat_2, na.rm = TRUE)
```

My function might look something like this:

`two_group_resample = function(x, n_1, n_2) {...}`

- I'll let you fill in the code. See the second lab question for a hint.

Here's a typical output of my implementation:

```
set.seed(54321)
two_group_resample(dat_pen$flipper_length_mm, 68, 152)
```

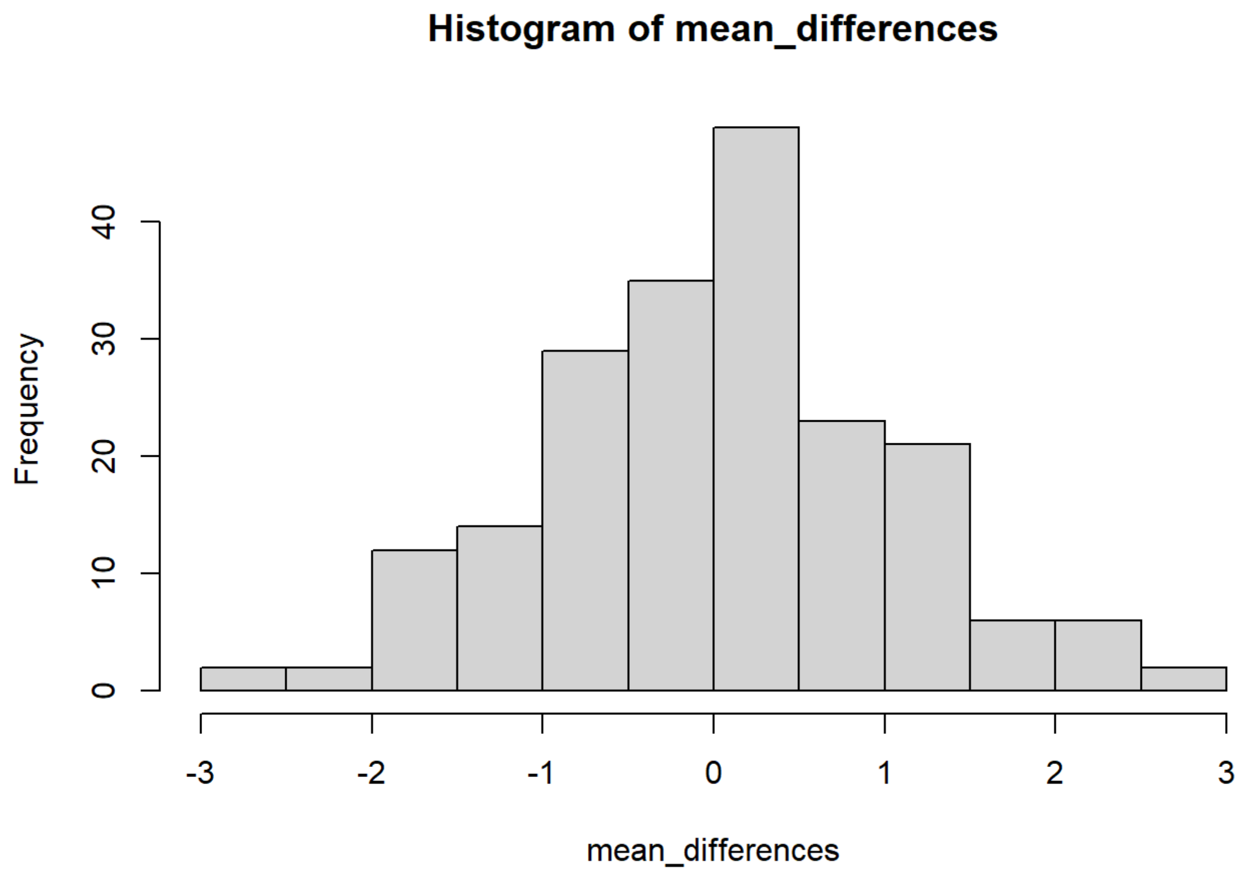
```
## [1] 2.123452
```

## Resampling experiment

If I ran this function many times, how often would I see a mean difference greater than `diff_observed` ?

I'll try it 200 times:

```
n = 200
mean_differences = c()
for (i in 1:n)
{
  mean_differences = c(
    mean_differences,
    two_group_resample(dat_pen$flipper_length_mm, 68, 152)
  )
}
hist(mean_differences)
```



```
sum(abs(mean_differences) >= diff_observed)
```

```
## [1] 0
```

I didn't see any differences as extreme as what was in the real data. You should try to re-run the code with 2000 simulations to see if you observe any differences greater than 5.8.

- Are these simulation results consistent with the t-test?

## Retrieving named elements

We've used the dollar sign to retrieve subsets of data.frames.

It can also be useful with different types of objects.

We saved the t-test output to an object:

```
t_test = t.test(flipper_shuffled ~ dat_pen$species)
```

Then we can use `str()` to examine what the object contains:

```
str(t_test)
```

```
## List of 10
##  $ statistic  : Named num 1.32
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 132
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 0.189
##  $ conf.int   : num [1:2] -0.686 3.439
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num [1:2] 192 191
##   ..- attr(*, "names")= chr [1:2] "mean in group Adelie" "mean in group Chinstra
p"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "difference in means between group Adelie and group
Chinstrap"
##  $ stderr     : num 1.04
##  $ alternative: chr "two.sided"
##  $ method     : chr "Welch Two Sample t-test"
##  $ data.name  : chr "flipper_shuffled by dat_pen$species"
##  - attr(*, "class")= chr "htest"
```

There's an item called `estimate`. We can access elements of most objects with the dollar sign:

```
t_test$estimate
```

```
##    mean in group Adelie mean in group Chinstrap
##                192.1974                190.8209
```

This method doesn't work for all objects in R, but it is often helpful to use the `str()` function to see what an object contains and the dollar sign to retrieve items of interest.

The structure of objects in R is complicated, but if you are interested here are a couple of resources

https://techvidvan.com/tutorials/r-object-oriented-programming/ (https://techvidvan.com/tutorials/r-object-oriented-programming/)

http://adv-r.had.co.nz/S4.html (http://adv-r.had.co.nz/S4.html)

# Lab Questions

Q1 Sample Standard Error Function    Q2-3 Two Group Resampling

Q4-5 Resampled Flipper Lengths    Q6 Resampling and p-values

Q7-11 Resampling a different variable

Recall your `sse_mean()` function. Your function should accept a single numeric vector and return a numeric value:

```
# Function template
sse_mean = function(x)
{

  # ... your code here ...

  return(sse)
}
```

You can compare your results to these to make sure that your standard error function is working correctly:

```
sse_mean(penguins$body_mass_g)
sse_mean(mtcars$mpg)
```

```
## [1] 43.36473
## [1] 1.065424
```

- **Q1 (3 pts.):** Show the R code you used to define your `sse_mean()` function. Include the following line before your function definition:

```
rm(list = ls())
```

and the following two test lines after your function:

```
sse_mean(penguins$body_mass_g)
sse_mean(mtcars$mpg)
```

## Report

Compile your answers into a pdf document and submit via Moodle.

- You may also do your work in an R Notebook and submit a rendered *html* file.