# Why focus on algorithms?

NRES 746

Fall 2021

**NOTE:** for those wishing to follow along with the R-based demo in class, click here for an R-script that contains all of the code blocks in this lecture.

## Algorithmic vs standard statistics: a brief demonstration

### Standard t-test

Here is a made-up data set.

Let's imagine we're interested in testing whether the mean body mass of farm-raised Atlantic salmon fed on a new all-vegetarian diet has a lower body mass than the 'typical' farm-raised fish raised on a conventional diet after one year of growth.

Let's assume that we know the following information:

First of all, we have been measuring the body mass of farm-raised salmon raised on a conventional diet for years, and we know that body mass for these individuals after one year closely follows a normal distribution with mean of 4.5 km and standard deviation of 0.9 km.

Secondly, we measured the body mass for 10 individuals raised on the new vegetarian diet after one year, and the measurements were as follows:

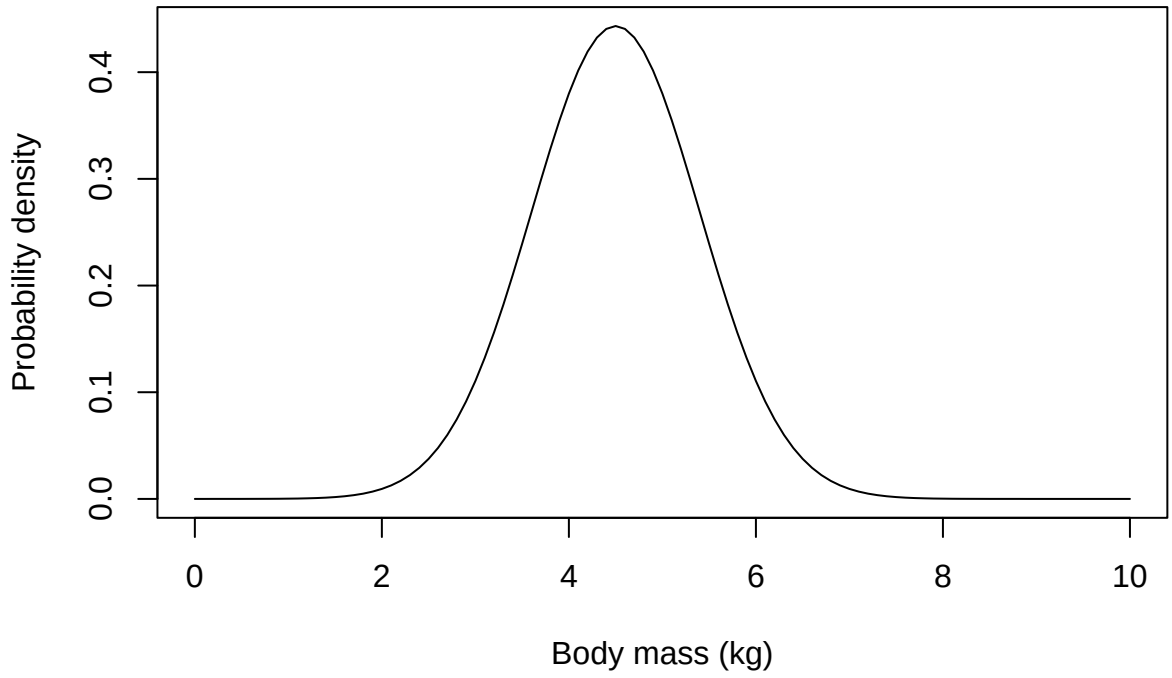
| Ind 1 | Ind 2 | Ind 3 | Ind 4 | Ind 5 | Ind 6 | Ind 7 | Ind 8 | Ind 9 | Ind 10 |
|---|---|---|---|---|---|---|---|---|---|
| 3.14 | 3.27 | 2.56 | 3.77 | 3.34 | 4.32 | 3.84 | 2.19 | 5.24 | 3.09 |

Finally, our alternative hypothesis is that the fish raised on the new diet will have lower body mass than fish raised on the conventional diet.

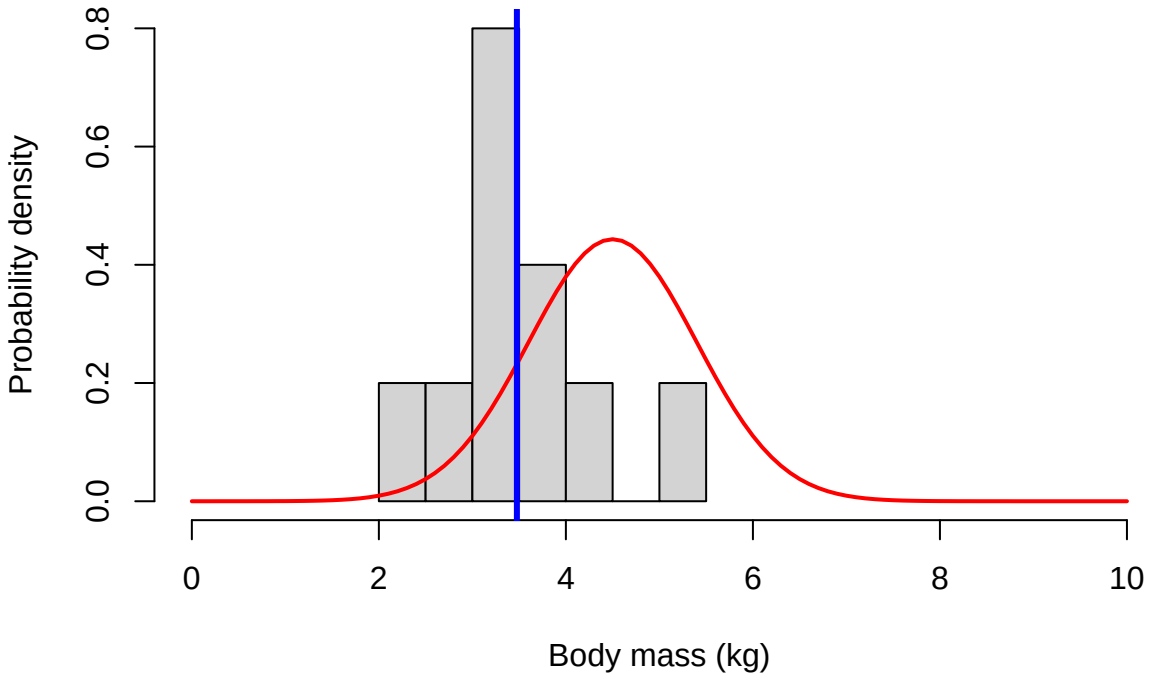
Let's read this information into R:

```
###################
# SALMON EXAMPLE (made-up!)

population.mean = 4.5
```

```r
population.sd = 0.9

my.sample = c(3.14,3.27,2.56,3.77,3.34,4.32,3.84,2.19,5.24,3.09)

sample.size <- length(my.sample)     # determine sample size

obs.samplemean = mean(my.sample)     # note the equal sign as assignment operator

## visualize the population of conventional-raised salmon

curve(dnorm(x,population.mean,population.sd),0,10,
      xlab="Body mass (kg)",ylab="Probability density")
```




```r
## now overlay this on the observed data

hist(my.sample,freq=F,
     xlab="Body mass (kg)",ylab="Probability density",main="",
     xlim=c(0,10))
curve(dnorm(x,population.mean,population.sd),0,10,
      col="red",lwd=2,add=T)
abline(v=obs.samplemean,col="blue",lwd=3)
```




You may recognize this as the kind of problem that you would address using a standard z-test; we are assuming for now that the samples are independently drawn from a normally distributed population with known variance. In the z-test, we are testing whether our sample could plausibly have been drawn from the **null distribution** (fish raised on conventional diet), which in this case has known variance. If we did not know the population variance (the population variance was uncertain) we would use a t-test instead of a z-test. We can run a x-test in R easily using the 'BSDA' package, using just one line of code!

```
################
# Perform standard z-test
################

library(BSDA)
z.test(x=my.sample,mu=population.mean, sigma.x=population.sd,alternative = "less")
```

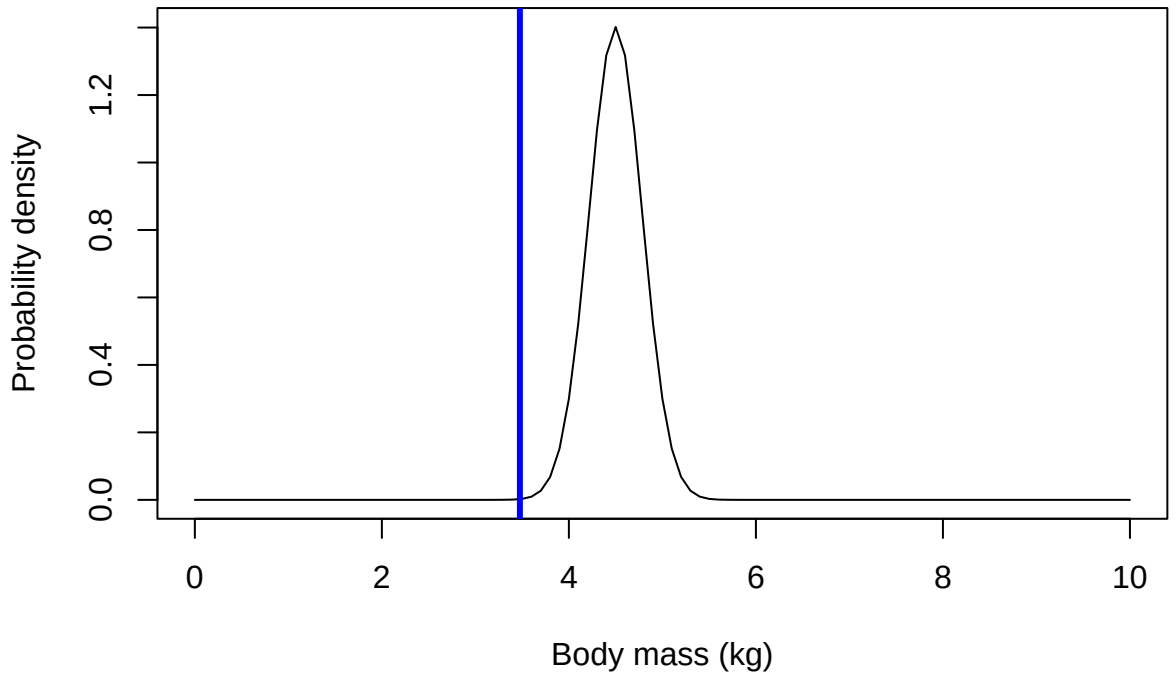
```
##
##  One-sample z-Test
##
## data:  my.sample
## z = -3.598, p-value = 0.0001604
## alternative hypothesis: true mean is less than 4.5
## 95 percent confidence interval:
##        NA 3.944134
## sample estimates:
## mean of x
##     3.476
```

...and we quickly see that our sample mean is much smaller than we could ever expect to sample under

the null hypothesis. Therefore we can conclude that the sample was NOT drawn from the null distribution, and that fish raised on the new diet tend to have lower body mass than other farm-raised fish raised on a conventional diet.

Here's an alternative z-test performed using base R:

```
#############
# alternative z-test

std.err = population.sd/sqrt(sample.size)

curve(dnorm(x,population.mean,std.err),0,10,     # visualize the sampling distribution under null hypot
      xlab="Body mass (kg)",ylab="Probability density")     # versus the observed sample mean
abline(v=obs.samplemean,col="blue",lwd=3)
```




```
p.val = pnorm(obs.samplemean,population.mean,std.err)
p.val     # this is the same as the p value from the z-test above...
```

```
## [1] 0.0001603558
```

**Brute-force z-test**

But imagine that we didn't know about the t-test. Let's build a solution to the same problem from the ground up, using our statistical intuition and R! Of course this is *totally unnecessary* in this case, but you will quickly run into problems with no simple, "canned" solution. That's where you might really need to develop an algorithm from scratch!

First, let's state the problem:

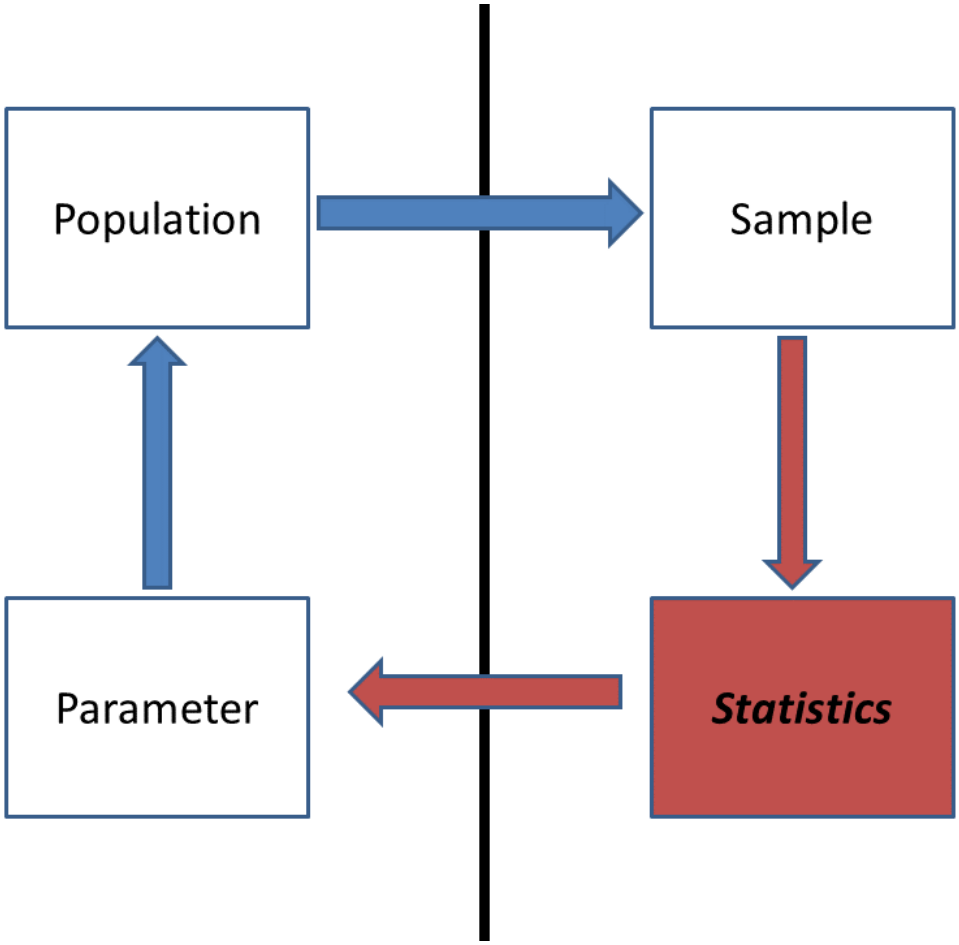
We want to know if the mean mass of 10 salmon raised on the new diet is less than the mean mass we would expect to observed if we sampled 10 random salmon raised on the conventional diet.

We assume that our sample of individuals raised on the new diet is identical to other farm-raised fish- the only difference is the diet.

We also assume that every fish in our sample is independently drawn from an infinitely large theoretical population of farm-raised fish raised on the new diet.

Let's build a simulation-based **algorithm** to generate a p-value!

Recall the difference between a "population" and a "sample" in statistics:




Ultimately, we want to make inference about a **population**, but all we have in hand is the **sample**. So we compute one or more **statistics** from our sample and use probabilistic reasoning to infer what our sample says about the population-level **parameters** we are interested in. Because we didn't observe the whole population (the sample typically represents only a small fraction of the total population), there's often substantial uncertainty about how well the sample statistic actually represents the population of interest- this is called **sampling uncertainty**.

Here, the *population* we are referring to is all farm-raised Atlantic salmon that are raised on the new vegetarian diet. The population *parameter* we are interested in is the mean body mass of these salmon after one year. The *sample* refers to all salmon actually measured as part of this study. Finally, the sample *statistic* is the observed mean body mass of all individuals in our sample.

Let's start by simulating a *statistical population* under the null hypothesis (no treatment effect):

```
######################
   # ALTERNATIVE ALGORITHMIC APPROACH!
######################

#############
# Simulate the STATISTICAL POPULATION under the null hypothesis
#############

infinity <- 1000000  # large number approximating infinity

popData_null <- rnorm(n=infinity,mean=population.mean,sd=population.sd)    # the statistical "populatio
```

Then we can draw a **sample** from the null distribution:

```
#############
# Draw a SAMPLE from that null data
#############

null.sample <- sample(popData_null,size=sample.size)    # use R's native "sample()" function to sample

round(null.sample,2)
```

```
##  [1] 3.27 4.70 2.70 3.30 4.69 5.01 3.32 3.50 4.11 5.76
```

```
null.samplemean <- mean(null.sample)
null.samplemean    # here is one sample mean that we can generate under the null hypothesis
```

```
## [1] 4.034775
```

Try it! What did you get? It may differ quite a bit from what I got!

This null sample mean represents the mean of 10 fish sampled under a *null hypothesis* where there is no underlying difference in body mass between the conventional diet and the new diet. The fact that this sample mean is not equal to 4.5 (the known population mean) represents **sampling error**.

Our ultimate goal is to determine how likely it is that the observed difference between the sample mean (the mean body mass of the fish we actually measured) and the known population mean is just a meaningless artifact of sampling error! This is exactly what the **p-value** tells us!

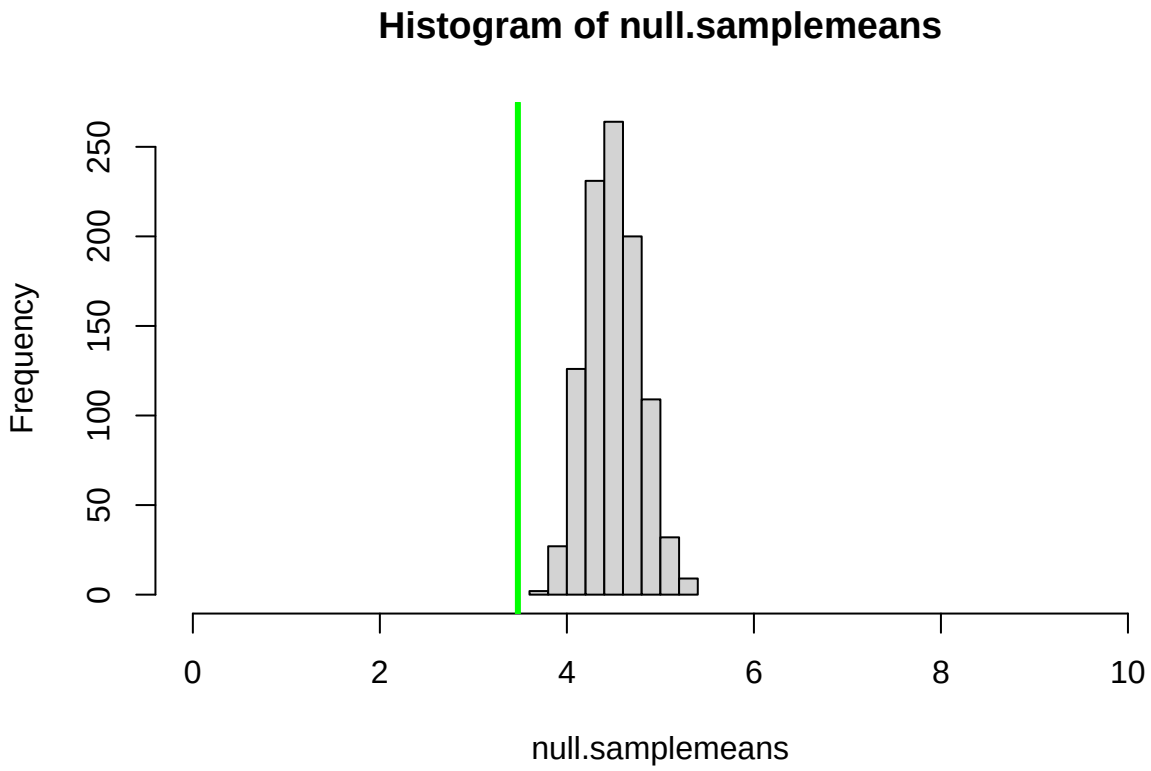
**Q** Given that our main goal is to **falsify** our null hypothesis about the population of interest, can you think of an algorithmic way to do this?

Let's generate a **sampling distribution** (distribution of sample means generated under the null hypothesis).

Here, we repeat this sampling process many times (using a "FOR" loop in R), each time drawing a different random sample of body masses from our statistical population.

```
#################
# Repeat this process using a FOR loop
#################

n.samples <- 1000                 # set the number of replicate samples to generate
null.samplemeans <- numeric(n.samples)       # initialize a storage vector for sample means under the n

for(i in 1:n.samples){            # for each replicate...
  this.nullsample <- sample(popData_null,size=sample.size)      # draw a sample of body masses assuming
  null.samplemeans[i] <- mean(this.nullsample)           # compute and store the sampling distribution
}
```

```r
hist(null.samplemeans,xlim=c(0,10))        # plot out the sampling distribution
abline(v=obs.samplemean,col="green",lwd=3)     # overlay the observed sample statistic.
```




Now, all we need to do to compute a p-value is to compare this vector of sampling errors with the observed statistic (between-group difference):

```r
#############
# Generate a p-value algorithmically!!
#############

ordered_means <- sort(null.samplemeans)       # sort the vector of null sample means
more_extreme <- length(which(ordered_means<=obs.samplemean))       # how many of these sampling errors
p_value <- more_extreme/n.samples       # compute a p-value!
p_value
```

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

Now, for convenience, let's collapse this all into a *function* for conducting our algorithmic t-test:

```r
##############
# Develop a function that wraps up all the above steps into one!
##############

z.test.algorithm <- function(sample, pop.mean, pop.sd){

  ##############
  # Compute the sample statistic
  ##############
```

```
  observed_mean <- mean(sample)

  sample.size <- length(observed_mean)   # compute sample size

  #################
  # Generate SAMPLING DISTRIBUTION
  #################

  reps <- 1000                 # set the number of replicate samples
  null_dist <- numeric(reps)       # initialize a storage structure for sampling distribution

  for(i in 1:reps){            # for each replicate...
    nullsamp <- rnorm(10,pop.mean,pop.sd)      # draw a sample assuming no treatment effect
    null_dist[i] <- mean(nullsamp)           # compute and store the sampling error produced under the
  }

  more.extreme <- length(which(null_dist<=observed_mean))       # how many of these sampling errors equ
  p_value <- more.extreme/reps

  to_return <- list()   # initialize object to return

  to_return$null_dist <- null_dist
  to_return$p_value <- p_value
  to_return$observed_mean <- observed_mean

  return(to_return)

}

ztest <- z.test.algorithm(sample = my.sample, pop.mean=population.mean, pop.sd=population.sd )   # try

ztest$p_value     # get the p_value

hist(ztest$null_dist)       # plot out all the sampling errors under the null hypothesis as a histogram
abline(v=ztest$observed_mean,col="green",lwd=3)     # indicate the observed sample statistic.
```

**Take-home message** The value of the algorithmic, brute-force approach to statistics is the flexibility! We have to be aware of assumptions in all of our analyses, but when we build our own computational algorithms, we can easily "relax" these assumptions! We only make the assumptions we are comfortable making. And we have to be totally explicit about our assumptions, because they are literally built into the code- we can't ignore any assumptions!

### A non-parametric t-test: permutation t-test

What if we don't want to make any assumptions about the process that generated the data? The normal distribution can arise in many different ways (according to the central limit theorem), but many data-generating processes **don't** result in a normal distribution!

We might be able to imagine which of the many alternative distributions makes the most sense for our data. But many times we can't do this with any level of certainty. What can we do in this case? A *permutation test* provides one answer.

This is a *nonparametric* test, meaning it does not make any assumptions about how the population parameter of interest is distributed.

**example: pygmy short-horned lizard**

Let's imagine we're interested in testing whether the expected mass of a study organism (let's say a pygmy short-horned lizard, *Phrynosoma douglasii*) in Treatment A (e.g., habitat restoration treatment) differs from Treatment B (e.g., no habitat restoration). In other words: does knowledge of an individuals treatment status contribute anything to understanding and/or predicting an individual's mass?

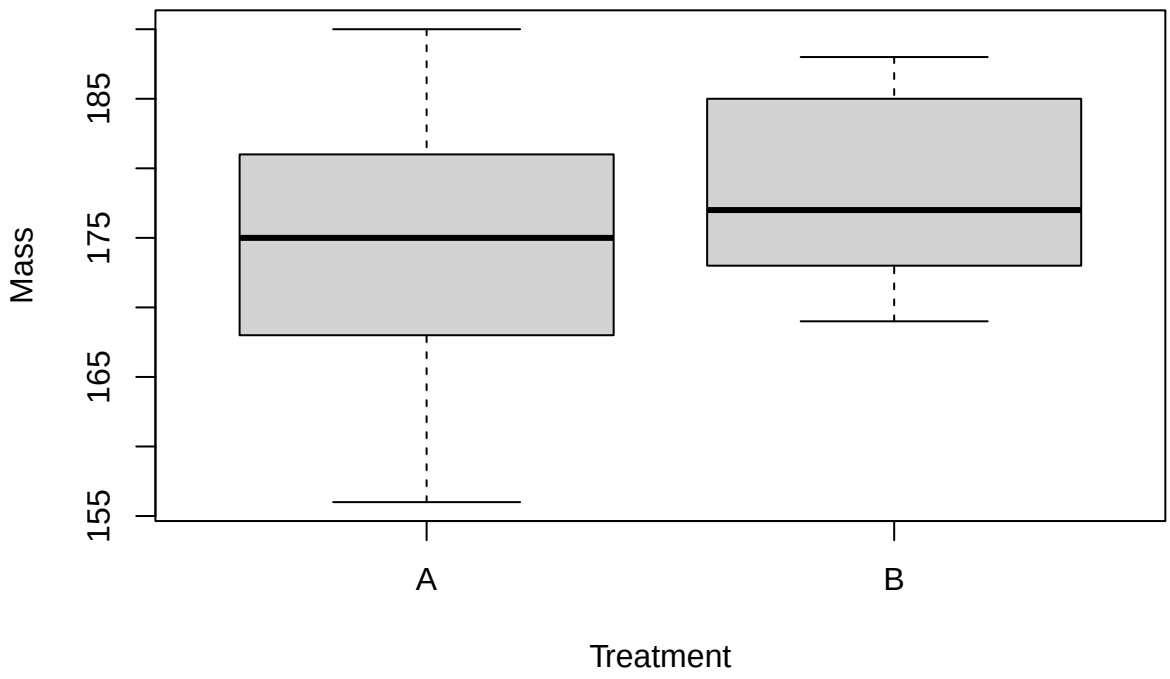
In this case, our alternative hypothesis is two-tailed: the treatment means are different, but treatment A mean could be larger or smaller than treatment B mean.

```
##############
# Start with a made-up data frame!
##############

df <- data.frame(
  A = c(175, 168, 168, 190, 156, 181, 182, 175, 174, 179),
  B = c(185, 169, 173, 173, 188, 186, 175, 174, 179, 180)
)

summary(df)    # summarize!
```

```
##        A               B
##  Min.   :156.0   Min.   :169.0
##  1st Qu.:169.5   1st Qu.:173.2
##  Median :175.0   Median :177.0
```

```
##  Mean   :174.8   Mean   :178.2
##  3rd Qu.:180.5   3rd Qu.:183.8
##  Max.   :190.0   Max.   :188.0
```

```
sample.size <- length(df$A)     # determine sample size

#######
# Get data in proper format

reshape_df <- data.frame(                # "reshape" the data frame so each observation gets its own ro
  Treatment = rep(c("A","B"),each=sample.size),
  Mass = c(df$A,df$B),
  stringsAsFactors = T
)


########
# Alternative (commented out)- using the 'tidyverse'

# library(tidyr)
# reshape_df <- pivot_longer(df,everything(),names_to = "Treatment",values_to="Mass")


plot(Mass~Treatment, data=reshape_df)    # explore/visualize the data
```




```
#######
# Compute the observed difference between group means
```

```r
observed_dif <- mean(reshape_df$Mass[reshape_df$Treatment=="A"])    - mean(reshape_df$Mass[reshape_df$T
```

Here, our goal will be to determine if the observed difference between the two group means could plausibly result from random sampling under the null hypothesis. This time, we want to generate a p-value that represents the probability that random sampling (under the null hypothesis) could result in a difference as or more extreme than the observed difference.

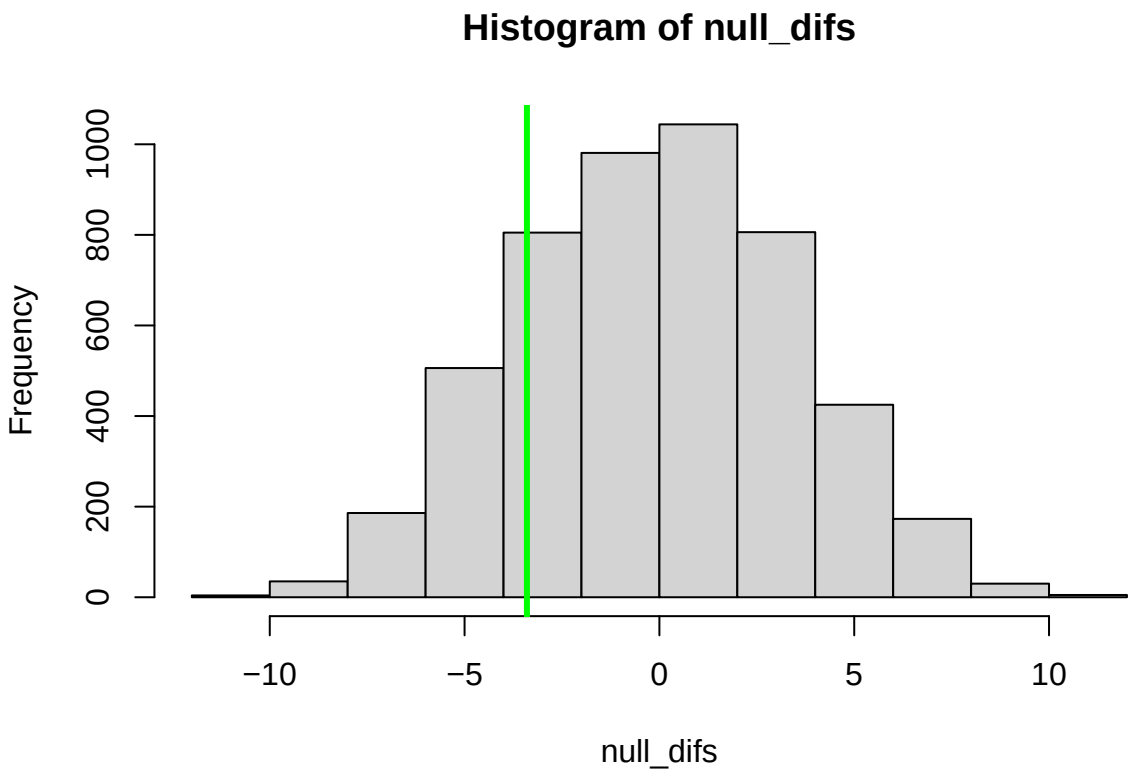
Let's build this permutation-test algorithm together.

Here is some **pseudocode**:

1. Define the number of permutations to run (number of replicate samples to generate)

2. Define a storage vector with the same number of elements as the number of samples to generate.
3. For each replicate sample:
    a. Assign each observation to a random treatment group (A or B)
    b. Compute the difference between the group means after assigning each observation to a random treatment group
    c. Store this value in the storage vector
4. Plot a histogram of the differences between group means under the null hypothesis (sampling distribution)
5. Add a vertical line to the plot to indicate the observed difference between group means

```r
##################
# NON-PARAMETRIC T-TEST -- PERMUTATION TEST
##################

reps <- 5000            # Define the number of permutations to run (number of replicates)
null_difs <- numeric(reps)   # initialize storage variable
for (i in 1:reps){          # For each replicate:
  newGroup <- reshape_df$Treatment[sample(c(1:nrow(reshape_df)))]              # randomly shuffle the o
    dif <- mean(reshape_df$Mass[newGroup=="A"]) - mean(reshape_df$Mass[newGroup=="B"])     #  compute t
    null_difs[i] <- dif     # store this value in a vector
}
hist(null_difs)    # Plot a histogram of null differences between group A and group B under the null hy
abline(v=observed_dif,col="green",lwd=3)   # Add a vertical line to the plot to indicate the observed d
```




Now we can compute a p-value, just as we did before:

```
########
# Compute a p-value based on the permutation test, just like we did before (except now 2-tailed)!
########

more_extreme <- length(which(abs(null_difs)>=abs(observed_dif)))
p_value <- more_extreme/reps
p_value
```

```
## [1] 0.3706
```

Again, for convenience, let's package this new permutation-based t test into an R function:

```
#############
# Develop a function that performs a permutation-t-test!
#############

t.test.permutation <- function(dat = reshape_df, group = "Treatment", value = "Mass" ){

  #############
  # Compute the sample statistic
  #############

  indexA <- which(dat[,group]=="A")      # rows representing treatment A
  indexB <- which(dat[,group]=="B")      # rows representing treatment B
  observed_dif <- mean(dat[indexA,value]) - mean(dat[indexB,value])
```

```
  reps <- 5000            # Define the number of permutations to run (number of replicates)
  null_difs <- numeric(reps)   # initialize storage variable
  for (i in 1:reps){            # For each replicate:
    newGroup <- reshape_df$Treatment[sample(c(1:nrow(reshape_df)))]           # randomly shuffle the o
    dif <- mean(reshape_df$Mass[newGroup=="A"]) - mean(reshape_df$Mass[newGroup=="B"])     #  compute t
    null_difs[i] <- dif     # store this value in a vector
  }

  more_extreme <- length(which(abs(null_difs)>=abs(observed_dif)))
  p_value <- more_extreme/reps

  to_return <- list()   # initialize object to return

  to_return$null_difs <- null_difs
  to_return$p_value <- p_value
  to_return$observed_dif <- observed_dif

  return(to_return)

}

my.ttest <- t.test.permutation()   # use default values for all function arguments

my.ttest$p_value

hist(my.ttest$null_difs)    # Plot a histogram of null differences between group A and group B under th
abline(v=my.ttest$observed_dif,col="green",lwd=3)   # Add a vertical line to the plot to indicate the o
```

## Bootstrapping a confidence interval

Let's imagine we want to compare different predictor variables in terms of how strong the relationship is with a response variable. In this case, we will use the coefficient of determination ($R^2$) as a measure of how good a predictor is. However, we want to be able to say that one predictor is definitively *better* than another one – for that, we would like a confidence interval around the $R^2$ value.

But... none of the standard R packages provides a confidence interval for the $R^2$ value... What do do???

With an algorithmic approach to statistics, getting stuck is not an option. We can just write some code!

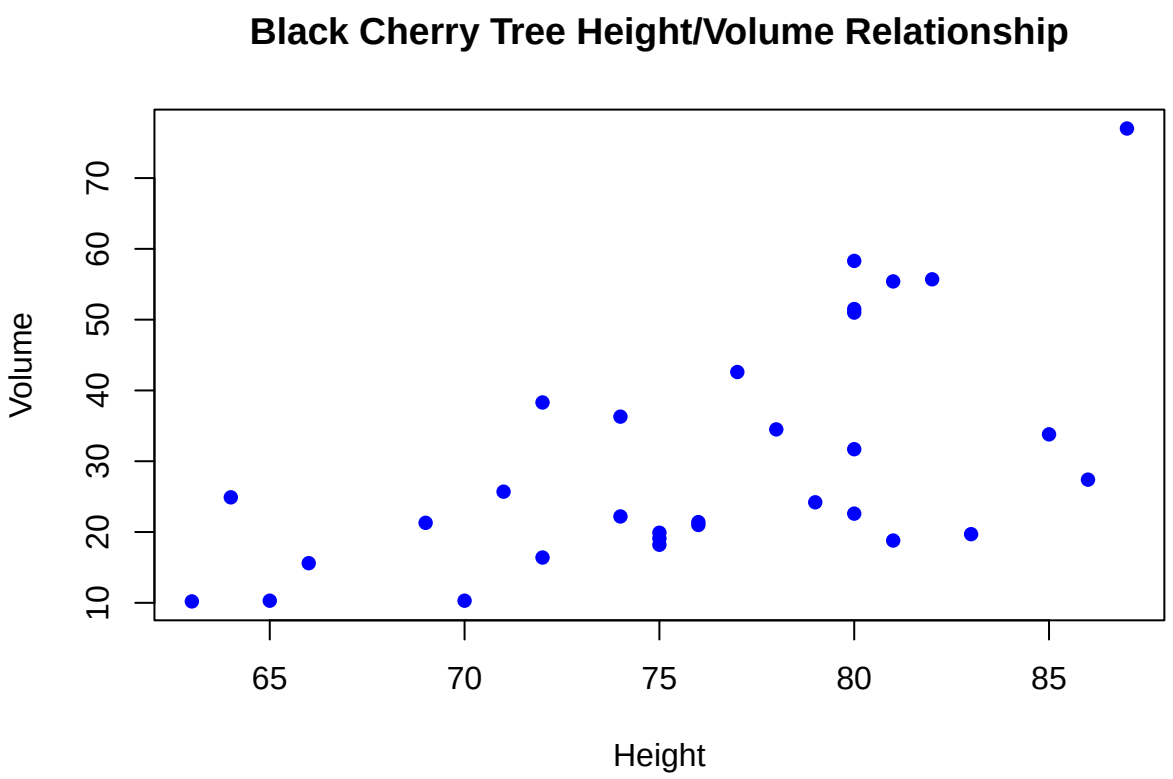
Let's use the "trees" dataset provided in base R:

```
##############
# Demonstration: bootstrapping a confidence interval!

## use the "trees" dataset in R:

head(trees)   # use help(trees) for more information
```

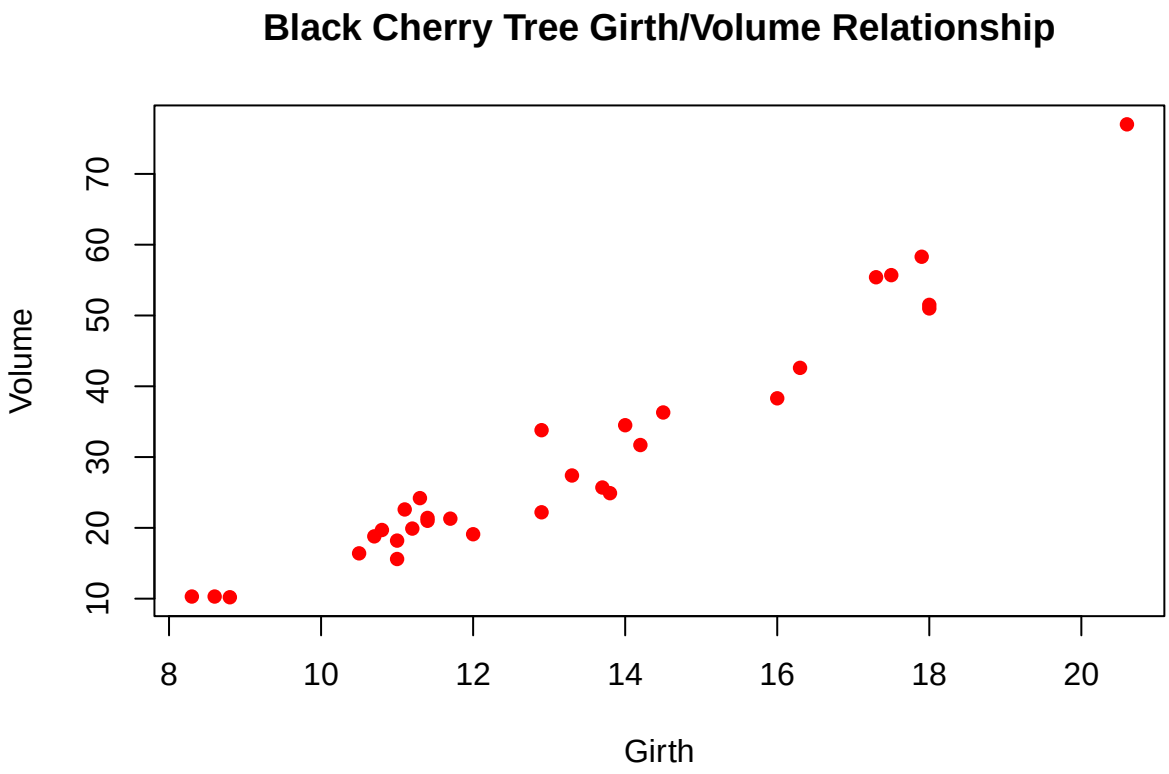
Tree volume is our response variable. We want to test whether girth or height are better predictors of tree volume.

Let's first do some basic data exploration:

```
#########
# Basic data exploration

plot(trees$Volume~trees$Height, main = 'Black Cherry Tree Height/Volume Relationship', xlab = 'Height',
```




```
plot(trees$Volume~trees$Girth, main = 'Black Cherry Tree Girth/Volume Relationship', xlab = 'Girth', yl
```




Let's write a simple function that generates coefficients of determination given a response and some predictor variables:

```
#########
# Function for returning a vector of R-squared statistics from models regressing a response variable on
   # here we assume that all columns in the input data frame that are NOT the response variable are pot

Rsquared <- function(df,responsevar="Volume"){    # univariate models only- interaction and multiple re
  response <- df[,responsevar]       # extract the response variable
  names <- names(df)
  rsq <- numeric(length(names))        # named storage vector
  names(rsq) <- names(df)
  rsq <- rsq[names(rsq)!=responsevar]           # assume that all columms that are not the response var
  for(i in names(rsq)){         # loop through predictors
      predictor <- df[,i]                  # extract this predictor
      model <- lm(response~predictor)       # regress response on predictor
      rsq[i] <- summary(model)$r.square       # extract R-squared statistic
  }
  return(rsq)
}
```

Let's first compute the $R^2$ values for all predictor variables:

```
#########
# test the function to see if it works!

stat <- Rsquared(trees,"Volume")
stat
```

```
##     Girth    Height
## 0.9353199 0.3579026
```

Now we can use a "bootstrapping" procedure to generate a confidence interval around these values, to see how certain we can be about the strength of the linear relationship between the response and the predictor variable in general (the population-level *parameter*) on the basis of the computed R-squared value (a sample *statistic*)

Let's first write a function to generate bootstrap samples from a data set:

```
#############
# new function to generate "bootstrap" samples from a data frame

boot_sample <- function(df,statfunc,n_samples,n_stats,responsevar="Volume"){
  indices <- c(1:nrow(df))
  output <- matrix(NA,nrow=n_samples,ncol=n_stats)        # storage object- to store a single bootstrap

  for(i in 1:n_samples){              # for each bootstrap replicate:
    boot_rows <- sample(indices,size=nrow(df),replace=T)         # randomly sample observations with re
    newdf <- df[boot_rows,]                          # dataframe of bootstrapped observations
    output[i,] <- statfunc(newdf,responsevar)                  # generate statistics from the bootstrapp
  }
  return(output)
}
```

Now we can generate a bunch of "bootstrapped" statistics to compare with the ones we calculated from the full dataset. Here, the values represent R-squared values from alternative bootstrapped samples. Each row is a different bootstrapped sample, and each column is a different predictor variable.

```
##########
# Generate a few bootstrapped samples!

boot <- boot_sample(df=trees,statfunc=Rsquared,n_samples=10,n_stats=2)       # generate test stats from
colnames(boot) <- names(stat)         # name the columns to recall which predictor variables they repre

boot
```

```
##           Girth    Height
##  [1,] 0.9466058 0.5104277
##  [2,] 0.9406089 0.2165671
##  [3,] 0.9430564 0.2689090
##  [4,] 0.8974317 0.2920579
##  [5,] 0.9451512 0.1826786
##  [6,] 0.8788925 0.1728614
##  [7,] 0.9407895 0.4464610
##  [8,] 0.9299656 0.3718730
##  [9,] 0.9500993 0.2592902
## [10,] 0.9511894 0.5384229
```

```
stat
```

```
##     Girth    Height
## 0.9353199 0.3579026
```

Finally, we can use the quantiles of the bootstrap samples to generate bootstrap confidence intervals.

```
#############
# use bootstrapping to generate confidence intervals for R-squared statistic!
```

```
boot <- boot_sample(df=trees,statfunc=Rsquared,n_samples=1000,n_stats=2)    # generate test statistics (
confint <- apply(boot,2,function(t)  quantile(t,c(0.025,0.5,0.975)))       # summarize the quantiles to
colnames(confint) <- names(stat)
t(confint)
```

```
##             2.5%       50%     97.5%
## Girth  0.8977934 0.9381074 0.9626512
## Height 0.1316745 0.3623297 0.6076836
```

Again, don't feel bad if you don't understand all the code yet. At this point, I just want you to understand the value of being able to program your own data analysis algorithms.

- It allows you to run custom analyses that you can't run any other way.
- It allows you to 'relax' assumptions that standard analyses may make
- It allows you to formalize your understanding of how your data were generated, and use this understanding to make the most of your data
- It's fun!

–go to next lecture–