Resampling Methods

Last time...

Review Lecture

- Weighted Least Squares (WLS)
- Worked on assignment

Today

- Start looking at topics related to machine learning
- We are going to start off with **Resampling methods**

First. Sampling

It is impossible to survey every person in the population we are interested in, so we often take a "random sample" from the population and calculate a **sample statistic** (e.g., mean, median).

A lot of our statistics follow well-defined distributions (e.g., normal distribution), and we use the properties of these distributions to estimate the population parameter.

Problems with Sampling

Single Estimate of the Population Parameter

- Estimate the population mean
- Sampling distribution of mean derived from sample statistics
- Built a 95% confidence interval around the sample mean
- We've been **relying on a single estimate** of the population parameter.

We make assumptions about the sample (e.g., representative sample, large sample size, normality), which may or may not be true.

Introduction to Resampling

A statistical technique that involves re-estimation of the population parameter by repeatedly drawing samples from **the original sample**

Reasons for Resampling

- Reduce bias of the estimate by using multiple samples instead of one
- Better sense of precision of the estimated population parameter
- We do not need to make assumptions about the population distribution (e.g., when we perform two samples t-test, for example, we make the assumption that the populations from which the samples are drawn are normally distributed)
- Sample does not have to be large

Types of Resampling

- Bootstrapping
- Jackknife method (just a glimpse)
- Permutation testing (just a glimpse; Research Methods will cover it)

pull yourself up by your bootstraps

Improve yourself/overcome (often impossible) obstacles without aid or assistance from anyone else. Use your own resources to improve your standing.

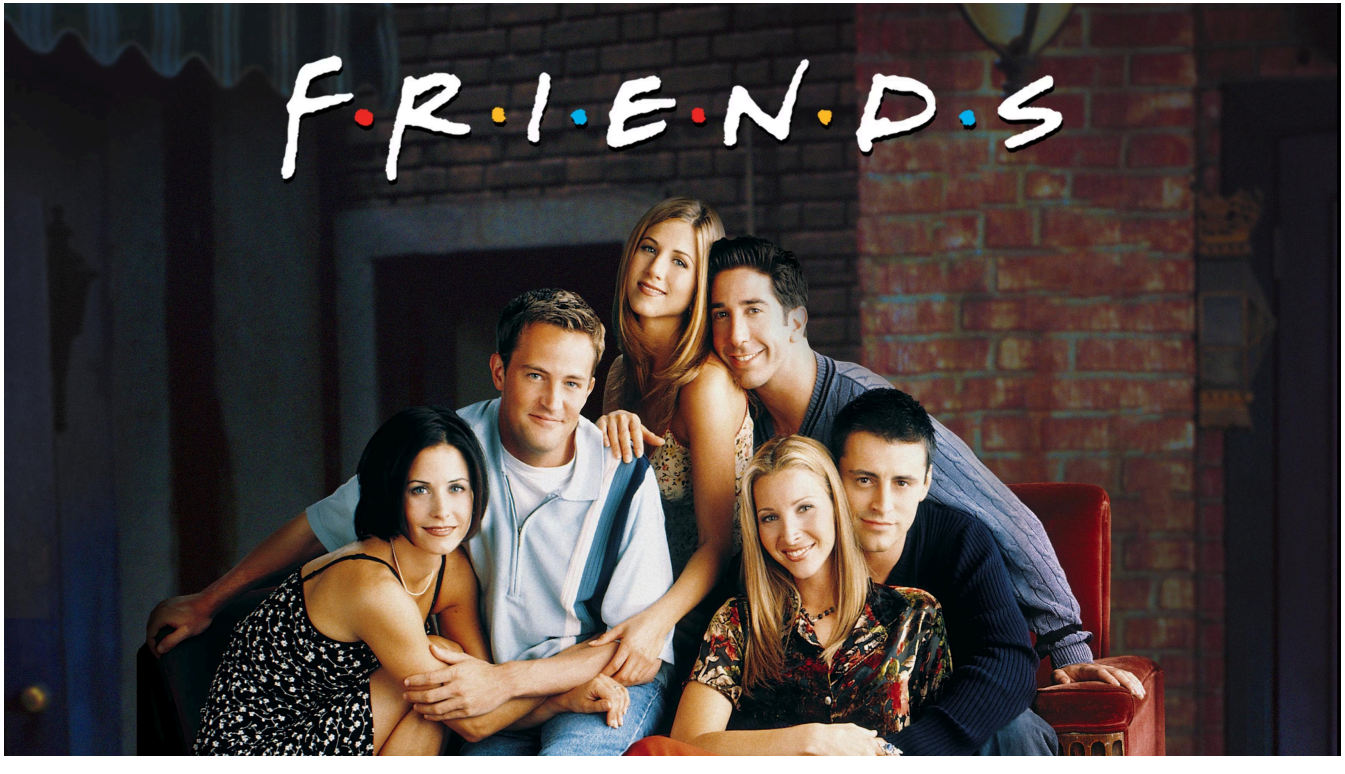
Bootstrapping

Bootstrapping is a method where we *rely entirely on the sample that we have at hand*.

- We randomly sample within the sample (with replacement)
- Compute the estimator of interest in order to...
- Build an empirical distribution of that test statistic.

Illustration of Bootstrapping

Imagine we are trying to estimate the height of a cohort of friends. But you only have 6. Specifically, these 6 friends.



To estimate their height, you decide to perform bootstrapping, meaning you draw many samples from this group of 6 people, *with replacement*, each time calculating the average height of the sample.

```
## [1] "Ross"     "Rachel"   "Chandler" "Monica"   "Monica"   "Rachel"
```

```
## [1] "Mean height of this sample: 168.5"
```

The number of friends that we randomly sample is 6, the same number of friends inside the initial sample.

Within the same sample, there can be duplicates. This is what it means to randomly sample **with replacement**.

Repeat

```
## [1] "Rachel"   "Phoebe"   "Ross"     "Monica"   "Monica"   "Chandler"
```

```
## [1] "Mean height of this sample: 169.666666666667"
```

```
## [1] "Monica"   "Phoebe"   "Chandler" "Chandler" "Rachel"   "Chandler"
```

```
## [1] "Mean height of this sample: 170.166666666667"
```

```
## [1] "Chandler" "Rachel"   "Chandler" "Chandler" "Joey"     "Rachel"
```

```
## [1] "Mean height of this sample: 169.833333333333"
```

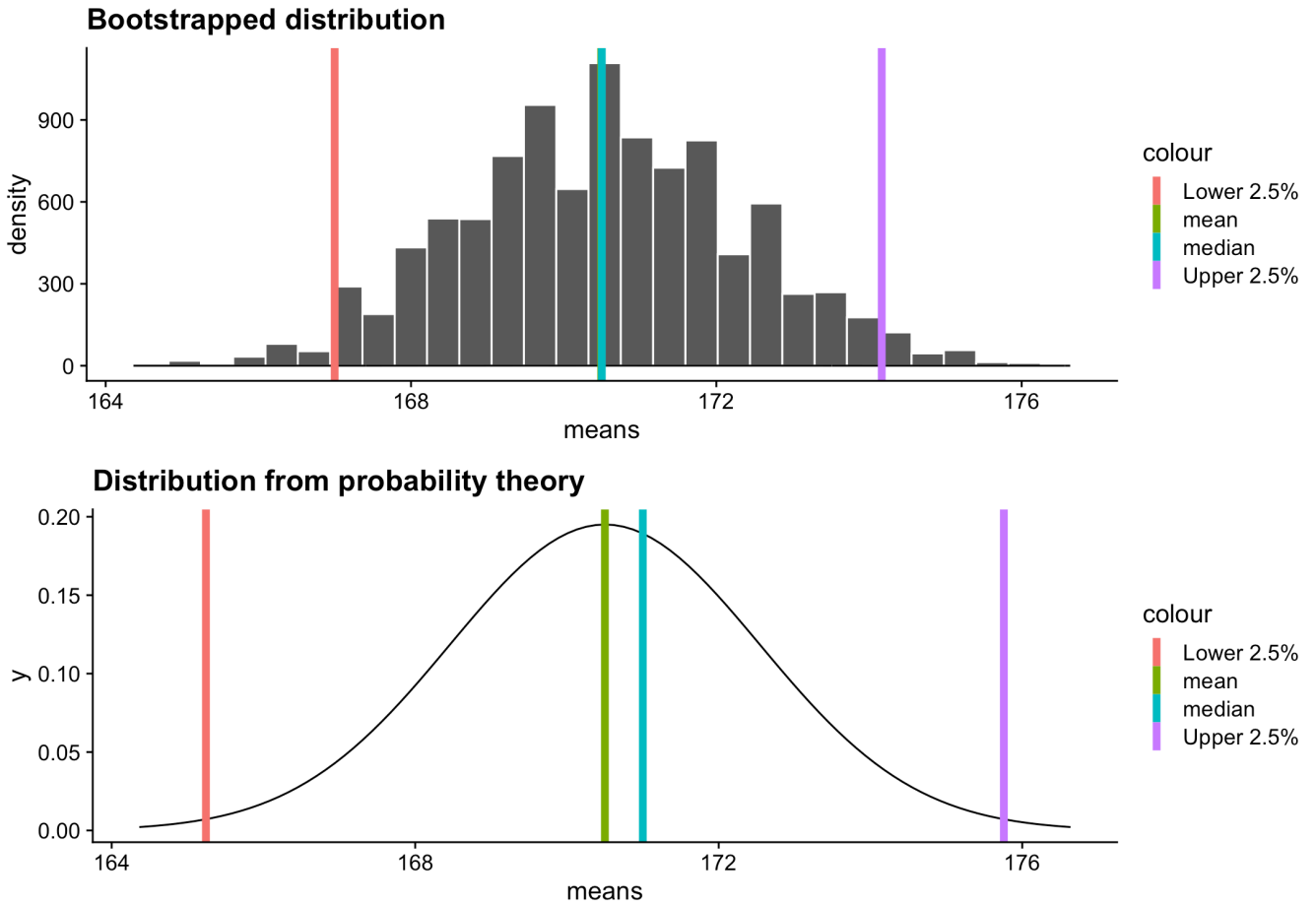
Bootstrap 10,000 Times

```r
# When resampling, it is generally a good practice to set ran
# for full reproducibility of the resampling process
set.seed(5067)

boot <- 10000 # Set number of bootstrap samples

friends = c('Monica', 'Rachel', 'Ross', 'Joey', 'Phoebe', 'Ch
heights <- c(165, 165, 178, 170, 172, 173)

sample_means <- NULL # Initialize list to store sample means
```

Bootstrap 10,000 Times

```r
# When resampling, it is generally a good practice to set ran
# for full reproducibility of the resampling process
set.seed(5067)

boot <- 10000 # Set number of bootstrap samples

friends = c('Monica', 'Rachel', 'Ross', 'Joey', 'Phoebe', 'Ch
heights <- c(165, 165, 178, 170, 172, 173)

sample_means <- NULL # Initialize list to store sample means

# Append the mean of bootstrap sample heights to *sample_mean
for(i in 1:boot){
  this_sample <- sample(heights, size = length(heights), repl
  sample_means <- c(sample_means, mean(this_sample))
}
```

Comparison


Bootstrapped distribution
density
900
600
300
0
164
168
172
176
means
colour
Lower 2.5%
mean
median
Upper 2.5%
Distribution from probability theory
y
0.20
0.15
0.10
0.05
0.00
164
168
172
176
means
colour
Lower 2.5%
mean
median
Upper 2.5%

There are several candidates for central tendency (e.g., mean, median) and for variability (e.g., standard deviation, interquartile range). **Some of these do not have well understood theoretical sampling distributions.**

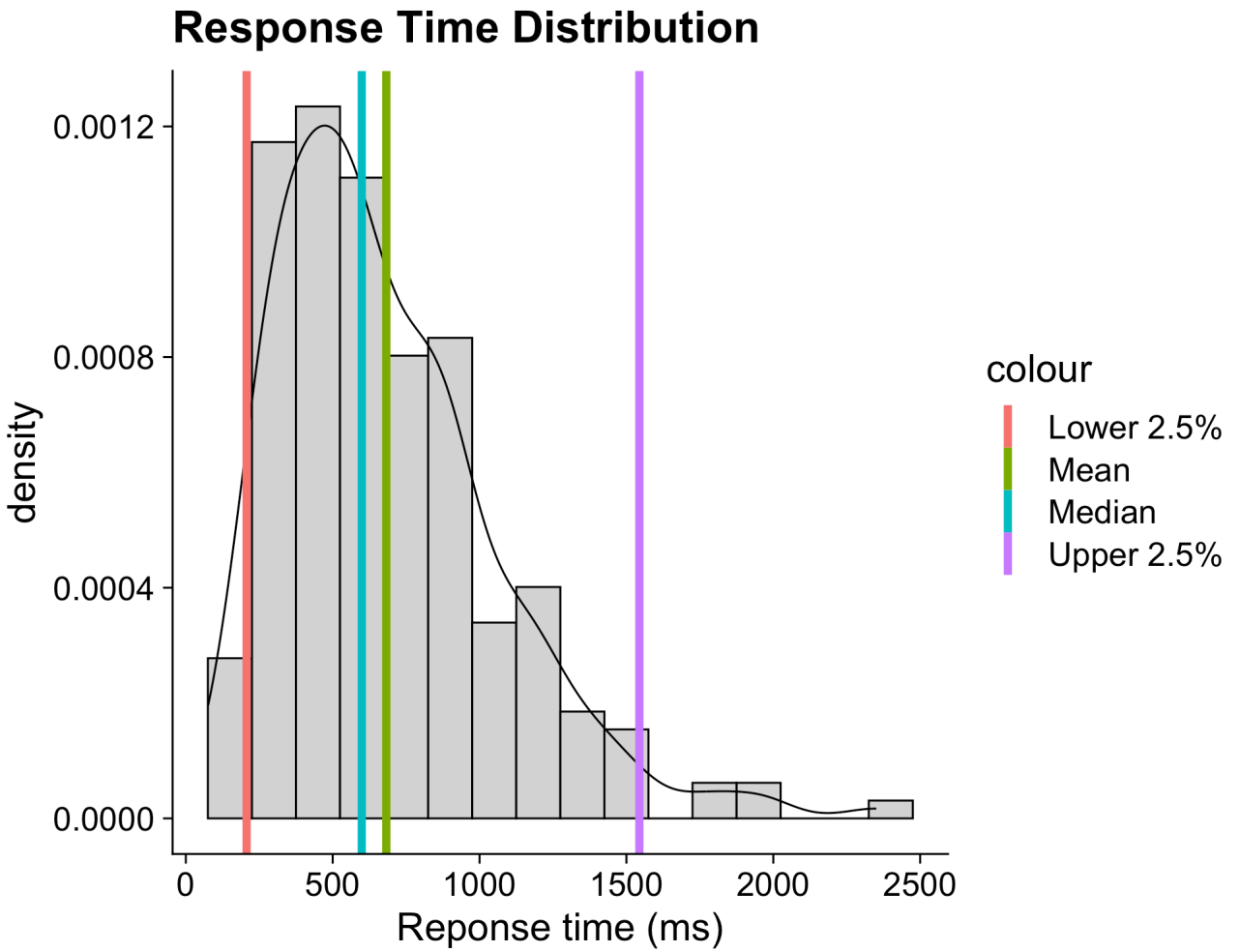
For the mean and standard deviation, we have theoretical sampling distributions to help us, provided we think the mean and standard deviation are the best indices. For the others, we can use bootstrapping.

Example 2

Central tendency and variability of 216 reaction times

```
# Set random seed before generating data
set.seed(5067)

# The observations generally follow the F Distribution + rand
response = rf(n = 216, 3, 50)
response = response * 500 + rnorm(n = 216, mean = 200, sd = 1
```

Visualize Data



```r
set.seed(5067) # Set random seed
boot <- 10000 # Set number of bootstrap samples

response_means <- NULL # Initialize list of mean values

for(i in 1:boot){
  sample_response <- sample(response, size = 216, replace = T
  response_means <- c(response_means, mean(sample_response))
}
```

What is the bootstrap mean and its 95% CI?

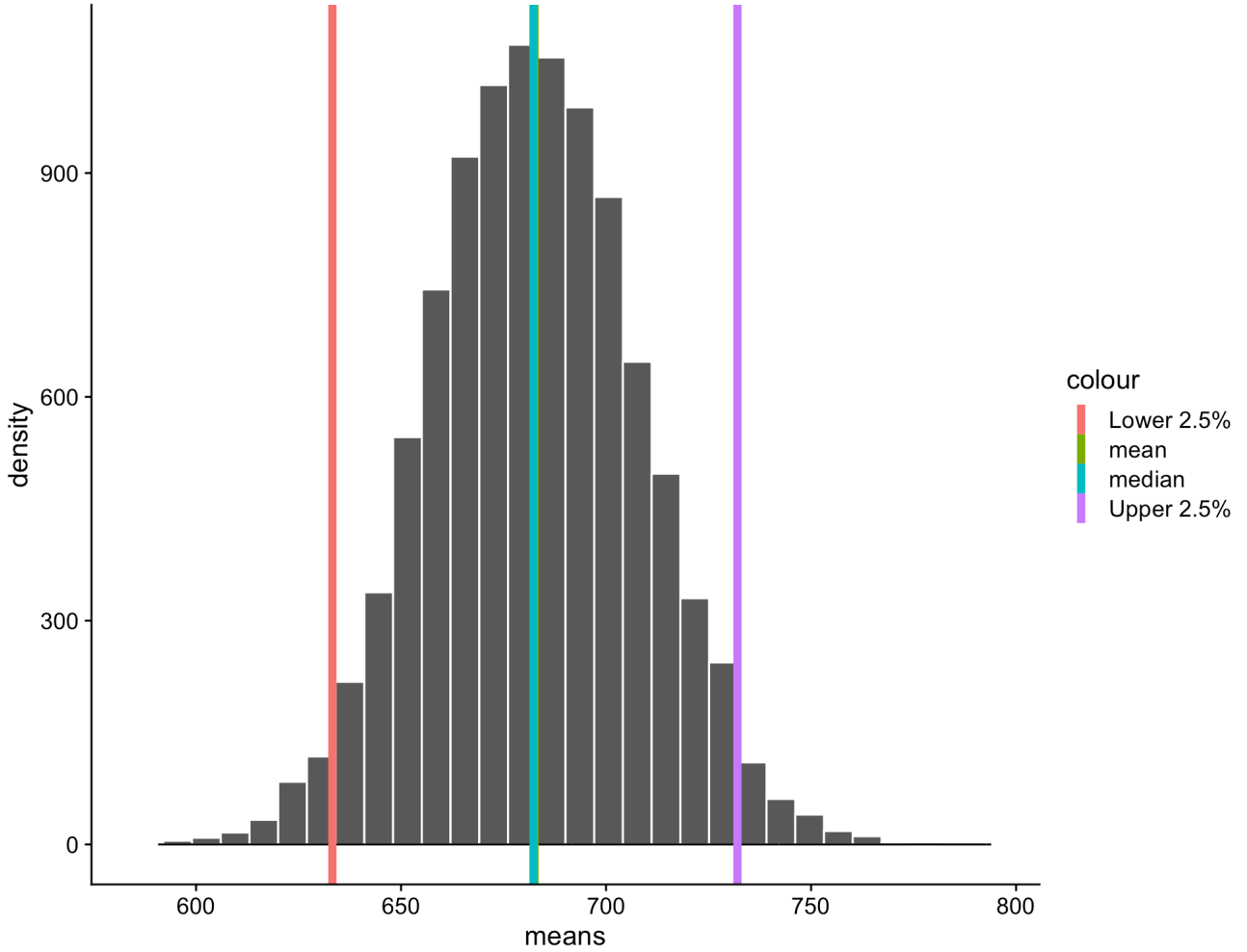
```r
mean(response_means)
```

```
## [1] 682.5499
```

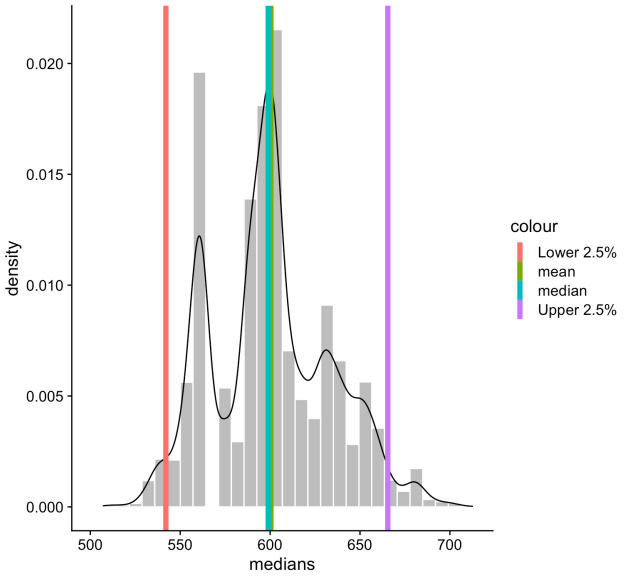
```r
quantile(response_means, probs = c(.025, .975))
```

```
##     2.5%    97.5%
## 633.2482 732.0461
```

Distribution of Means



```
set.seed(5067)
boot <- 10000
response_med <- NULL

for(i in 1:boot){
  sample_response <- sample(response, size = 216, replace = T
  response_med <- c(response_med, median(sample_response))}
```




```
mean(response_med)
```

```
## [1] 600.4248
```

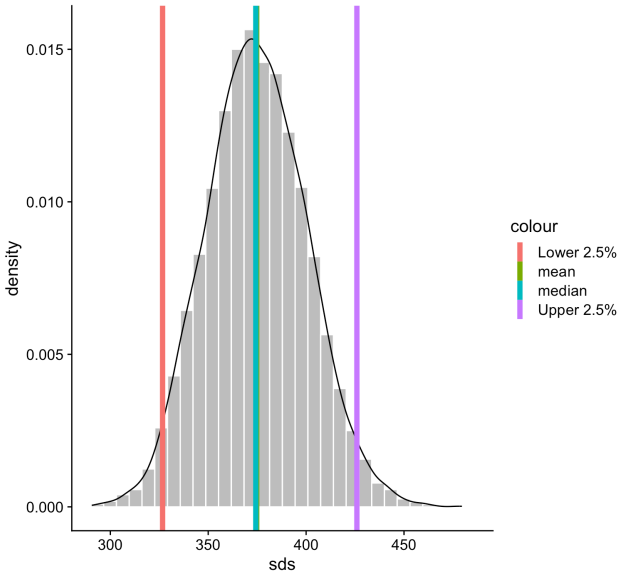
```
median(response_med)
```

```
## [1] 598.9464
```

```
quantile(response_med,
         probs = c(.025, .97
```

```
##     2.5%    97.5%
## 542.0469 665.4593
```

```
set.seed(5067)
boot <- 10000
response_sd <- NULL

for(i in 1:boot){
  sample_response <- sample(response, size = 216, replace = T
  response_sd <- c(response_sd, sd(sample_response))}
```




```
mean(response_sd)
```

```
## [1] 374.7614
```

```
median(response_sd)
```

```
## [1] 374.299
```

```
quantile(response_sd,
         probs = c(.025, .975
```

```
##     2.5%    97.5%
## 326.6762 425.9399
```

Other Estimators?

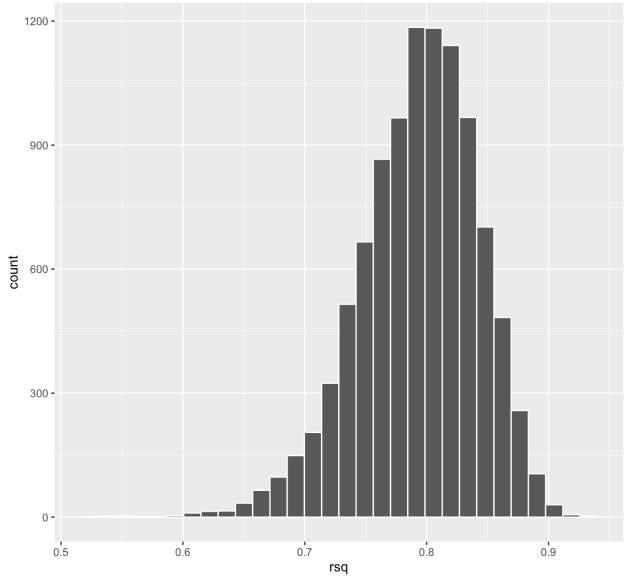
You can bootstrap estimates and 95% confidence intervals for *any* statistics you'll need to estimate.

Things you should learn how to do in R:

- learn to read a `for loop`
- learn to write your own function

The `boot` package and function provides some help to speed this process along. Things you should learn how to do in R:

```
library(boot)

# function to obtain R-Squared from the data
rsq <- function(data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(mpg ~ wt + disp, data = d) # this is the code you
  return(summary(fit)$r.square)
}

results <- boot(data = mtcars, statistic = rsq, R = 10000)
```




```
median(results$t)
```

```
## [1] 0.7972849
```

```
boot.ci(results, type = "per
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULA
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "p
##
## Intervals :
## Level     Percentile
## 95%   ( 0.6862,  0.8771 )
## Calculations and Intervals on Origina
```

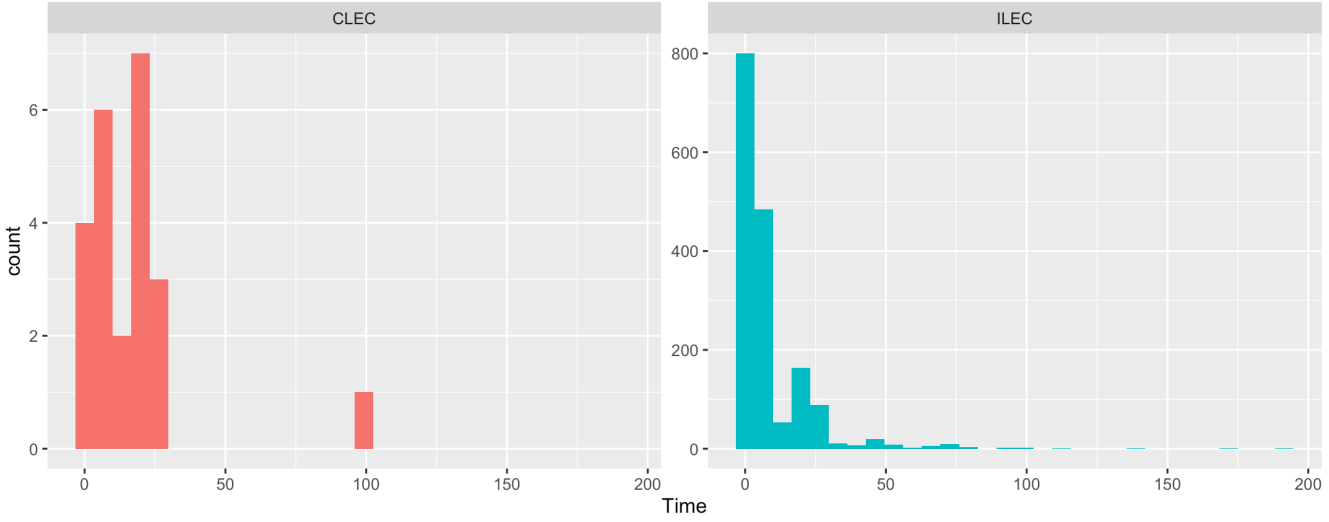
Another Example

In this district, Verizon provides line service to both Verizon and non-Verizon customers. Here, we are going to look at a dataset containing service waiting times for Verizon customers (ILEC) and non-Verizon customers (CLEC). We are interested in **whether waiting time of non-Verizon customers is longer than that of Verizon customers**.

```
head(Verizon, 3)
```

```
##   Time Group
## 1 17.5  ILEC
## 2  2.4  ILEC
## 3  0.0  ILEC
```

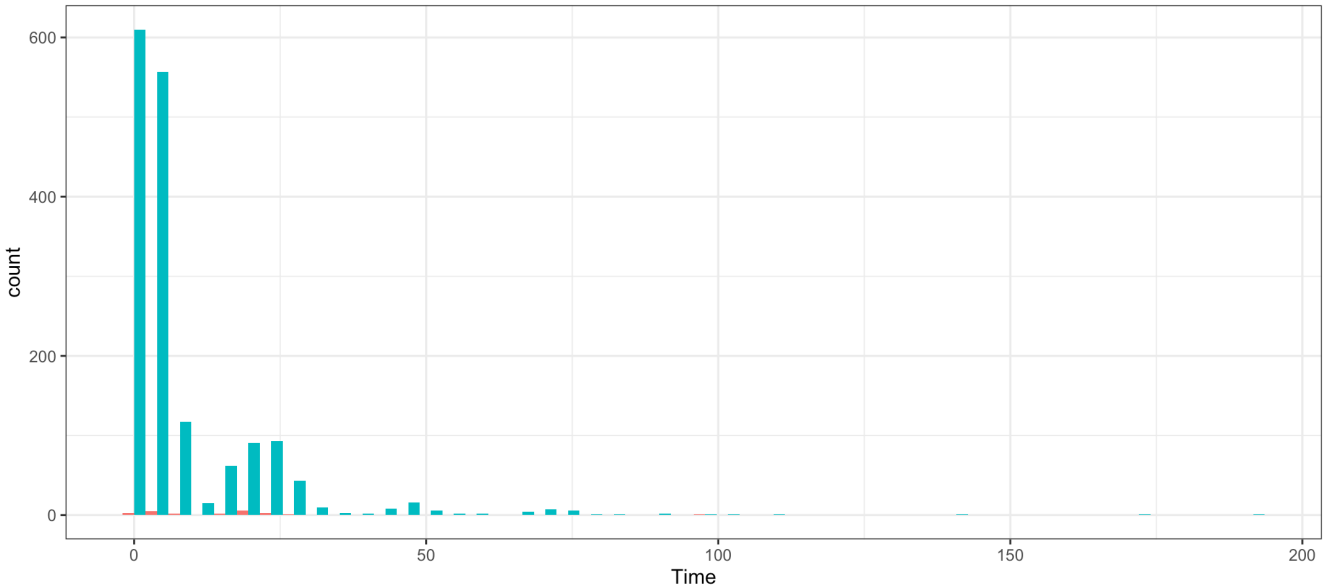
Inspect Verizon Data 1



- Left is the distribution of waiting times of Non-Verizon (CLEC) customers; and
- Right is the distribution of waiting times of Verizon (ILEC) customers.

Both distributions are not normal.

Inspect Verizon Data 2




```
##
## CLEC ILEC
##   23 1664
```

The groups are (very) unbalanced.

Analysis Plan and Justification

It seems that the data do not meet the typical assumptions of an independent samples t-test.

In this case, to estimate mean differences we can use bootstrapping.

Here, we'll resample with replacement separately from the two samples and calculate their difference in means.

```
# Set random seed and number of bootstrap samples
set.seed(5067)
boot <- 10000

response_means <- NULL

for(i in 1:boot){
  sample_response <- sample(response, size = 216, replace = T
  response_means <- c(response_means,mean(sample_response))
}
```

Inside the For Loop

- Sample (with replacement) Verizon group (ILEC) customers
- Sample (with replacement) non-Verizon group (CLEC) customers
- Calculate the difference in means between the two groups
- Append the difference value to a list

One Solution

```
set.seed(5067)
boot <- 10000
difference <- NULL

subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")
```

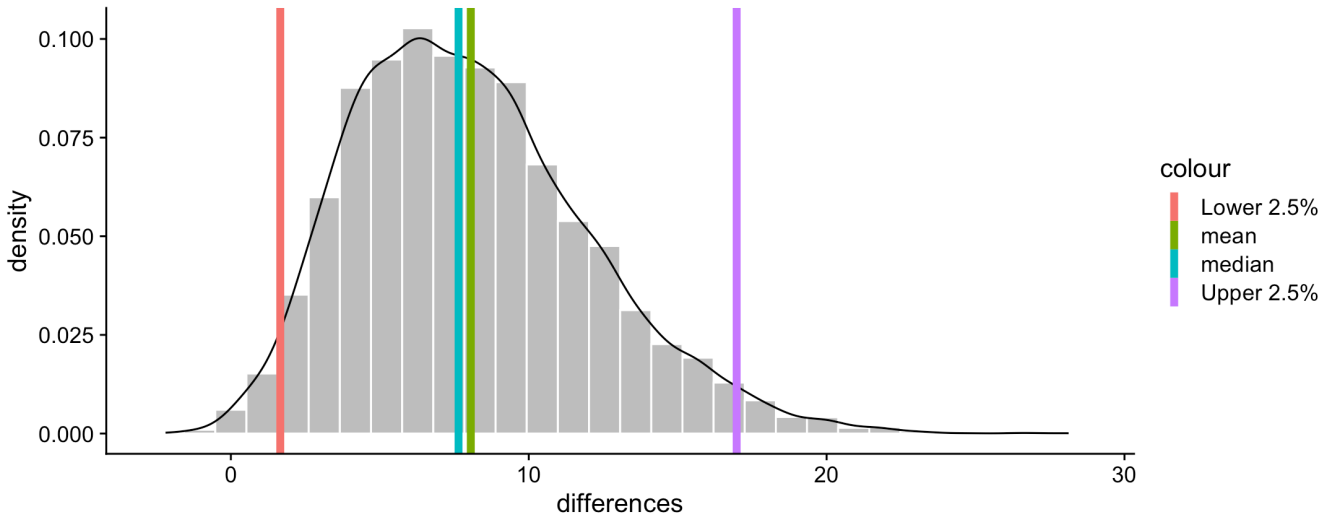
One Solution

```
set.seed(5067)
boot <- 10000
difference <- NULL

subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")

for(i in 1:boot){
  # Sample (with replacement) Verizon group (ILEC) customers
  sample_CLEC = sample(subsample_CLEC$Time,
                       size = nrow(subsample_CLEC),
                       replace = T)
  # Sample (with replacement) Non-Verizon group (CLEC) custom
  sample_ILEC = sample(subsample_ILEC$Time,
                       size = nrow(subsample_ILEC),
                       replace = T)
}
```

One Solution

```
set.seed(5067)
boot <- 10000
difference <- NULL

subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")

for(i in 1:boot){
  # Sample (with replacement) Verizon group (ILEC) customers
  sample_CLEC = sample(subsample_CLEC$Time,
                       size = nrow(subsample_CLEC),
                       replace = T)
  # Sample (with replacement) Non-Verizon group (CLEC) custom
  sample_ILEC = sample(subsample_ILEC$Time,
                       size = nrow(subsample_ILEC),
                       replace = T)

  # Calculate the difference in means between the two groups
  # Append the difference value to a list
  difference <- c(difference, mean(sample_CLEC) - mean(sample
}
```

Bootstrap Distribution of Differences



The difference in means is 7.64 $[1.66, 16.98]$. What would this mean?

Bootstrap CI does not include 0. There is a difference in the waiting time of Verizon and non-Verizon customers in the expected

Bootstrapping is useful when:

1. Violated assumptions of the test (e.g., normality)

2. You have good reason to believe the sampling distribution is not normal, but don't know what it is (e.g., median)

3. Oddities in your data, like very unbalanced samples

This allows you to create a confidence interval around any statistic you want. You can test whether these statistics are significantly different from any other value.

Bootstrapping will **NOT** help you deal with:

- Dependence between observations -- for this, you'll need to explicitly model dependence
- Improperly specified models or forms -- use theory to guide you here
- Measurement error -- why bother?
- Caveats: representativeness of the sample, outliers

Jackknife Resampling

Jackknife Resampling is a method where researchers generate n sub-samples, each leaving out one observation. The method is very similar to bootstrapping except **the way that we create the sub-samples**.

Friends Example. If you decide to jackknife their heights, you would draw six sub-samples and calculate their respective mean height.

```
friends = c('Monica', 'Rachel', 'Ross', 'Joey', 'Phoebe', 'Ch
heights = c(165, 165, 178, 170, 172, 173)
names(heights) = friends
```

```
## [1] "First Sub-sample:  Rachel, Ross, Joey, Phoebe, Chandler"

## [1] "Second Sub-sample:  Monica, Ross, Joey, Phoebe, Chandler"

## [1] "Third Sub-sample:  Monica, Rachel, Joey, Phoebe, Chandler"

## [1] "Fourth Sub-sample:  Monica, Rachel, Ross, Phoebe, Chandler"

## [1] "Fifth Sub-sample:  Monica, Rachel, Ross, Joey, Chandler"

## [1] "Sixth Sub-sample:  Monica, Rachel, Ross, Joey, Phoebe"
```

Notice that there are **6** sub-samples of **size 5**.

Jack of all trades, master of none - Jackknife

1. Can assess the accuracy of a statistical estimator without making assumptions about the underlying distribution

2. Computationally efficient bc only *n* sub-samples are generated (compared to 10,000 for example)

3. Not used as much these days

4. Sensitive to sample size: If sample size is small, it can result in inaccurate estimates of the bias (e.g., sample size of 6 means six jackknife samples).

Permutation Testing

A resampling method that involves **randomly shuffling the labels** (e.g., conditions) across the data and recomputing the test statistic of interest, thereby deriving a null distribution of the test statistic.

Permutation Example: Restaurants

We are interested in the difference in the rating of group A and group B restaurants.

```
##                      name group rating
## 1    Pappy's Smokehouse     A      7
## 2                Mai Lee     A      8
## 3 Adriana's on the Hill     A      8
## 4           Salt & Smoke     B      4
## 5            Chilli Spot     B      6
## 6                BLK MKT     B      7
```

Permutation Example: Restaurants

The observed difference is...

```
##                     name group rating
## 1    Pappy's Smokehouse     A      7
## 2                Mai Lee     A      8
## 3 Adriana's on the Hill     A      8
```

```
## [1] 7.666667
```

```
##           name group rating
## 1 Salt & Smoke     B      4
## 2  Chilli Spot     B      6
## 3      BLK MKT     B      7
```

```
## [1] 5.666667
```

```
perm_df %>% filter(group == 'A') %>% pull(rating) %>% mean()
```

```
## [1] 2
```

Initially, the three restaurants in group A were:

```
##                     name group rating
## 1    Pappy's Smokehouse     A      7
## 2                Mai Lee     A      8
## 3 Adriana's on the Hill     A      8
```

Now, we are going to randomly assign restaurants to group A and group B, and calculate the mean difference in ratings.

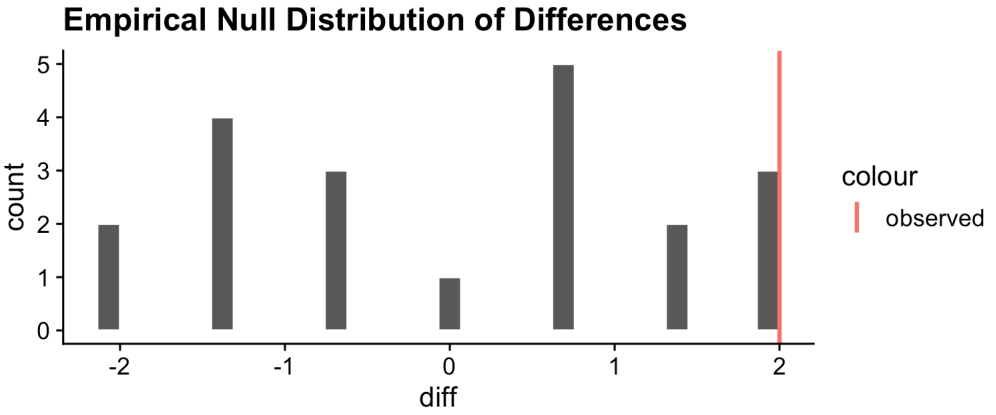
```
random_indices <- sample.int(nrow(perm_df), 3)
random_A <- perm_df[random_indices,]
random_B <- perm_df[-random_indices,]

random_A %>% pull(name)
```

```
## [1] "Salt & Smoke" "Mai Lee"      "Chilli Spot"
```

```
random_A %>% pull(rating) %>% mean() - random_B %>% pull(rati
```

```
## [1] -1.333333
```

Repeat the previous step numerous times to impose a null distribution of mean differences



We could count the number of permutations that have larger test statistic value than the observed difference, which would be equivalent to the p-value (or the probability that a test statistic is greater than or equal to the observed value, **under the null**).

Let's Try

Initially, we had 23 CLEC and 1,664 ILEC customers. **First**, we are going to calculate the mean difference in waiting time between the two groups using the observed data.

```
subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")

(mean(subsample_CLEC$Time) - mean(subsample_ILEC$Time))
```

```
## [1] 8.09752
```

Second Step

We are going to randomly shuffle the labels of groups (previous CLEC and ILEC labels don't matter!). Label 23 random customers as CLEC and 1,664 random customers as ILEC 1,000 times, and store the mean difference in waiting time between the two groups inside a list.

One iteration would look something like this:

```
random_indices <- sample.int(nrow(Verizon), 23)
random_CLEC <- Verizon[random_indices,]
random_ILEC <- Verizon[-random_indices,]

(mean(random_CLEC$Time) - mean(random_ILEC$Time))
```

```
## [1] 2.394109
```

Repeat 1,000 times to derive the *p*-value

One iteration would look something like this:

```
random_indices <- sample.int(nrow(Verizon), 23)
random_CLEC <- Verizon[random_indices,]
random_ILEC <- Verizon[-random_indices,]

(mean(random_CLEC$Time) - mean(random_ILEC$Time))
```

```
## [1] -0.3775922
```

Analysis Plan

1. Put this in a for loop
2. Construct a list of mean differences
3. Determine the number of (random) mean differences greater than the observed difference in means (8.10).

One Way to do this

```
subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")
observed_diff <- mean(subsample_CLEC$Time) - mean(subsample_I

set.seed(1048596) # Set random seed
perm = 1000 # Set number of permutations

differences <- NULL
```

One Way to do this

```
subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")
observed_diff <- mean(subsample_CLEC$Time) - mean(subsample_I

set.seed(1048596) # Set random seed
perm = 1000 # Set number of permutations

differences <- NULL

for (i in 1:perm){
  random_indices <- sample.int(nrow(Verizon), 23)
  random_CLEC <- Verizon[random_indices,]
  random_ILEC <- Verizon[-random_indices,]

  differences <- c(differences,
                   mean(random_CLEC$Time) - mean(random_ILEC$
}
```

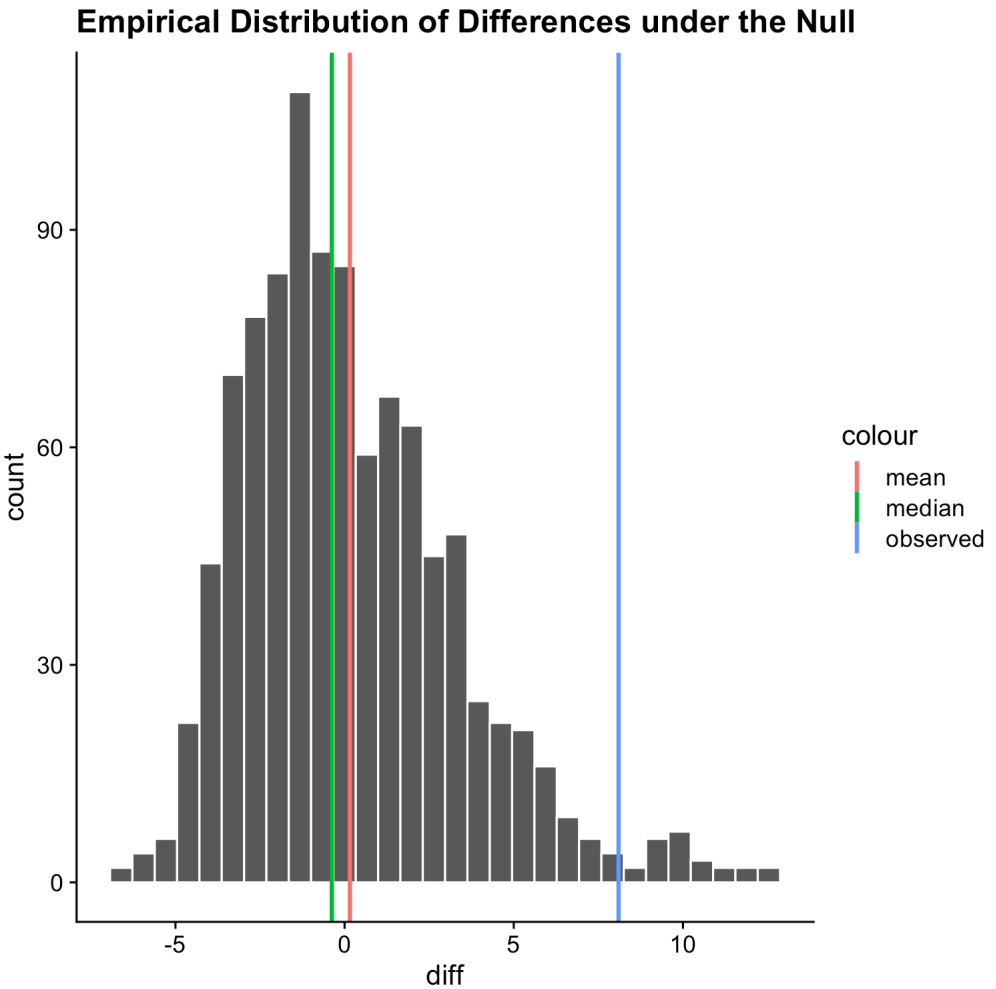
```
subsample_CLEC = Verizon %>% filter(Group == "CLEC")
subsample_ILEC = Verizon %>% filter(Group == "ILEC")
observed_diff <- mean(subsample_CLEC$Time) - mean(subsample_I

set.seed(1048596) # Set random seed
perm = 1000 # Set number of permutations

differences <- NULL

for (i in 1:perm){
  random_indices <- sample.int(nrow(Verizon), 23)
  random_CLEC <- Verizon[random_indices,]
  random_ILEC <- Verizon[-random_indices,]

  differences <- c(differences,
                   mean(random_CLEC$Time) - mean(random_ILEC$
}

paste("Number of mean differences greater than observed diffe
```

```
## [1] "Number of mean differences greater than observed difference:  24"
```

```
paste("p-value: ", sum(differences > observed_diff)/length(di
```

```
## [1] "p-value:  0.024"
```

Visualization



Permutation Testing Summary

- Permutation testing is useful for hypothesis testing because we can easily derive a p-value
- Can be used when data violates common assumptions about the data (i.e., homogeneity of variance and normality)
- Assumption that the observations need to be exchangeable. Some observations may not be exchangeable (e.g., time series data - data collected at different time points)
- Sample size needs to be large. No point randomly shuffling 1,000 times when the possible permutation is less than 1000.

Next time ...

- Thursday: One more machine learning lecture
- Don't forget to read Yarkoni & Westfall, 2017!
- Next Tuesday: Final review session