First, let me be upfront with a bit of a warning. This lesson is a bit more technical than the previous. To boot there isn't much material in the main book. To compensate, I add much more background here in the notes than in the first three lessons.

The basic material in this lesson covers topics that have been opened up in recent years due to advances in computation, in particular the ease and ubiquity of computation now. "Ease," of course is relative, and the point here is to "ease" you in. The main thing is *simulation*. Something R is particularly helpful with. We will discuss how to do basic simulations and then apply what we see to two types of computer-based inference: permutation methods for doing significance tests and bootstrap methods for finding confidence intervals.

To do all this, we get introduced to the following R topics:

- Writing basic functions
- Looping or iterating over values. For this we discuss `for`, `replicate` and `sapply`

Statistical inference

The key idea of statistical inference is that a sample from a population somehow represents the population – called then a representative sample. How the sample is selected has a big effect on whether the sample is representative or not. For our discussions, statistics is based on the sample being chosen randomly from the population. [1] This does not ensure the sample is representative though! Rather, the random sampling and the laws of probability say that most of the time representative samples will be produced.

[1] Something devilishly hard to do in practice!

If a sample is representative, then a statistic formed from the sample then feels the influence of the population, hence a statistic may in turn give information about the population.

We have seen this in practice.

For example, in lesson 1 we considered the case when the *population is normally distributed* with mean μ. It was noted that the t-statistic $t = (\bar{x} - \mu)/(s/\sqrt{n})$ then has a t-distribution with $n - 1$ degrees of freedom. This sampling distribution can be used to to express a confidence interval for μ centered at $\bar{x}$: $(\bar{x} - t^* s/\sqrt{n}, \bar{x} + t^* s/\sqrt{n})$. The exact statement uses probability which is possible because the *sampling distribution* for the statistic in question is known:

> A $(1 - \alpha) \cdot 100\%$ confidence interval for μ based on $\bar{x}$ is given by
>
> $$(\bar{x} - t^* s/\sqrt{n}, \bar{x} + t^* s/\sqrt{n}).$$

It is well known, that the assumption of a normally distributed population can be relaxed if other things are known: if n, the sample size, is large (by the central limit theorem); or if the population is not too far from normal, by robustness of the underlying t-statistic.

What can be done though when the normal assumptions or the relaxations do not apply? In this case we do not know the underlying sampling distribution of $\bar{x}$ and hence can't make probability statements as that above.

One of the advances of statistics due to the computer age is that there are cases where unknown sampling distributions can be **simulated** to gain an understanding of their center, spread, shape and, most importantly, their distribution.

Prior to computers, simulations were still performed but were much more tedious. Gossett (aka Student), in his 1906 paper on the t-statistic, verified his calculations doing a simulation where he wrote down 3,000 measurements of the left middle finger of criminals (this data set had been previously used). He thoroughly shuffled the cards and then grouped them into 750 groups of 4. For each group he computed the mean and standard deviation. [2] Following these compuations, he plotted the theoretical t-distribution that he had computed and the empirical one computed from his simulations to investigate the match between theory and practice.

[2] The latter being a problem as sometimes all 4 values were identical due to the data being truncated.

We'll do the same, only R will make this *much* easier.

The basic idea of a simulation is contained in the above. We need:

- A means to generate random samples from a population is used. In the above this involved shuffling the 3,000 values and the picking groups of 4. These groups are a thought of as a random sample from the population.
- Statitics to summarize the random samples. The above mentions both the mean and standard deviation
- A means to investigate the population through the sample. Gossett did so graphically, as will we, but our graphs will be easy to do.

Simulations using R

Keeping in mind the desire to understand sampling distributions, we see how to perform Monte Carlo simulations to investigate sampling distributions and their properties.

R has the "r" functions to do basic simulations from *known* probability distributions. For example, 5 simulated normally distributed numbers are produced with this simple command:

```
rnorm(5)
```

```
[1]  0.31487804  0.24765885 -0.31540032 -0.03455825  0.43274838
```

The mean and standard deviation default to 0 and 1. Additional arguments to `rnorm` allow these to be adjusted.

What if we wanted to simulate from an unknown distribution, [3] say the median of 4 randomly generated normal numbers. A single value is found with:

```
median(rnorm(4))
```

```
[1] 0.8482783
```

Another with the same command

```
median(rnorm(4))
```

```
[1] 0.5872316
```

But how can we get *lots* of values easily (enough so that we can begin to see more or less what the actual distribution describing these values is)?

[3] Well, relatively unknown. With some work this one can be found explicitly

The replicate function to simplify simulations

To understand the distribution [4] of a statistic or random variable, many simluated values are needed.

There are many different ways to repeat some process in R. We mention 3 below. First, the `replicate` function can be called to replicate values `n` times.

An example usage would repeat the above 750 times with with: [5]

```
res <- replicate(750, median(rnorm(4)))
res[1:5]
```

```
[1] -0.03089782  0.60387400 -1.12534208  0.88634658 -0.97840332
```

New functions

`replicate`
`for`
`sapply`

[4] That is, what are the possible values and how likely these will be.

[5] Gosset would have loved R. It must have taken quite a long time to just write down the numbers involved, let alone compute the standard deviations of the 750 samples. Here we can do things in a snap.

The format is:

`replicate(` no. of times, block of commands to replicate `)`

Sometimes (usually!) using parameters in your code makes it easier to understand. Using `n` for the size of a sample and `m` for the number of simulations, we might see the above written as

```
m <- 750
n <- 4
res <- replicate(m, {
      median(rnorm(n))
  })
```

The braces

Composing functions, as we just done produces "one-liners." These are easy to write, but hard to read and debug. An alternate style is to write different commands on different lines. If the expression has more than 1 line, the braces must be used to enclose the expression. For example:

```
m <- 750
n <- 4
res <- replicate(m, {
      theData <- rnorm(n)
      median(theData)
  })
```

The value assigned to `res` is the last one in the block. In the above this is the median for the sample, and not the sample `theData` itself.

Using for to make simulations

A more familiar means to repeat some expression to those who have programmed before is the `for` loop. The basic steps here are to define a storage vector, `res`, define the values to loop over `1:1000` and then do the loop.

The following shows the basic steps:

```
res <- c()
m <- 750
n <- 4
for (i in 1:m) {
      x <- rnorm(n)
      res[i] <- median(x)
  }
```

The specification of what is being iterated over will commonly take the form we have above, but can be more general. For instance, it can be values in a data set:

```
for (i in letters[1:3]) print(i)
```

```
[1] "a"
[1] "b"
[1] "c"
```

(We skipped the braces here, as only one expression is being evaluated).

Many R programmers try to avoid `for` loops, thinking alternatives are faster. Indeed this is so when one can use vectorization

instead, but not necessarily the case. If you are familiar with using `for` loops there isn't too much reason to change your style. If you are learning, let's see some other alternatives.

Applying a function

Think about what is computed in

```
res <- c()
for (i in 1:3) res[i] <- median(rnorm(5))
```

There are 3 repeats of first generating 5 random numbers and summarizing. We could reverse, somewhat and compute 15 random numbers first, then summarize 3 times. But how? First, to make the 15 numbers we could just call `rnorm15`, but to get in a better data format we use a matrix with 3 columns:

```
m <- matrix(rnorm(15), ncol = 3)
m
```

```
           [,1]        [,2]       [,3]
[1,] -1.3804066 -1.62832286 -0.1528686
[2,] -0.8407022 -0.08419978 -0.1611478
[3,]  0.1721022  0.50120532 -0.2134632
[4,] -1.0657220  1.53696082  1.3682039
[5,]  0.5399167 -1.11045105 -0.1274351
```

Now we want to "apply" the median to each column. The `apply` function will do this. The `2` discusses what direction to apply over (`1` would be over the 5 rows:

```
apply(m, 2, median)
```

```
[1] -0.84070221 -0.08419978 -0.15286858
```

So no for loop. This separates out the data production from the summarizing. Sometimes a cleaner conceptual thing. [6]

Using `sapply` to iterate over values

The `apply` function works with matrices (and arrays). For vectors and lists, the `sapply` [7] function can be used. As data frames are lists with each column being a component, the above can be done through:

```
sapply(as.data.frame(m), median)
```

```
         V1          V2          V3
-0.84070221 -0.08419978 -0.15286858
```

[6] A not too distant discussion on the R mailing list spoke to the differences conceptually between a for loop and using apply like functions. The analogy of Barry Rowlingson is:

> To me, the world and how I interact with it is procedural. When I want to break six eggs I do 'get six eggs, repeat "break egg" until all eggs broken'. I don't apply an instance of the break egg function over a range of eggs. My world is not functional (just like me, some might say...). `https://stat.ethz.ch/pipermail/r-help/2009-May/198322.html`

He is arguing for a for loop – as that breaks the eggs one at a time. Where as replicate applies the egg breaking function n times. R gurus learn to appreciate the "applying" part of R, but when learning R, a for loop seems to be most natural.

[7] And `lapply` and `vapply`

This function basically iterates over the values in its first argument and calls the function specified in the second argument on these values. Extra arguments may be given to pass into the function. For example, were there possible NA values, we could pass along an argument instructing R to ignore them:

```
sapply(as.data.frame(m), median, na.rm = TRUE)
```

```
         V1          V2          V3
-0.84070221 -0.08419978 -0.15286858
```

New functions

function keyword

Functions

R has tremendous flexibility. This is due in part to to its powerful subsetting syntax, but most importantly to the fact that it is easy to extend through user-written functions. Functions allow multi-step processes to be isolated off. This encapsulation makes thinking about large processes easier and also makes debugging easier.

An R function is similar to the abstract mathematical definition: a function is a rule assigning a value in the domain to a value in the range. The "domain" is specified by the function's arguments, the range is given by the function's return value and the "rule" determined by the body of the function.

For an example, the skew of a data set may be defined by

$$\frac{\frac{1}{n}\sum(x_i - \bar{x})^3}{(\frac{1}{n}\sum(x_i - \bar{x})^2)^{3/2}}.$$

This formula is a bit tedious to type into R, and were we to use it many times our fingers would likely rebel. Fortunately, if one were to write a function, the typing would only need to be done once. Here is how:

```
skew <- function(x) {
      n <- length(x)
      ds <- (x - mean(x))
      top <- (1/n) * sum(ds^3)
      bottom <- ((1/n) * sum(ds^2))^(3/2)
      top/bottom
  }
```

The three main pieces are:

The input variables Functions have arguments. The one above has a single argument, x. Arguments are specified just after the function keyword, which indicates to R a function is being defined. We have seen in examples that some functions have *named* arguments. For example, the arguments to the rnorm function are:

```
args(rnorm)
```

```
function (n, mean = 0, sd = 1)
NULL
```

the arguments are named `n`, `mean` and `sd`. The first one is not optional, the second two have *default values*. You can tell, as they are set with a value after the equals sign. (For `mean` it is 0 and for `sd` it is 1.) [8]

When calling a function, named arguments match by name, unnamed arguments match by position.

The body The body of a function is where the inputs get turned into the outputs. Typically [9] the body is separated off by curly braces. The values for the arguments are passed in by the user or the defaults are used. [10]

The output The output of a function is simply the result of last line evaluated. Common – but not required – is to call either `return`, to return the value; or `invisible`, to invisibly return the values (it won't print). In the above, the return values is just that found by computing `top` divided by `bottom`.

In the above, the `function` keyword returns a function object that gets assigned to `skew`. Calling `skew` on a dataset will cause this functions rule to be evaluted.

Functions too, can also be used as arguments of other functions.[11] In these cases, an *anonymous* function is often used, where an unnamed function is used for the argument.

Question 0.0.1 *The kurtosis of a distribution measures deviations from a normal distribution by longer tails and/or narrower shoulders. The t-distribution exhibits both.*

The kurtosis is defined by:

$$\frac{(1/n)\sum(x_i - \bar{x})^4}{((1/n)\sum(x_i - \bar{x})^2)^2} - 3$$

(The -3 makes the normal distribution have 0 kurtosis.).

Write an R function to compute this value.

Question 0.0.2 *Write a function to compute the t-statistic. It will need two arguments, `x` to pass in the data and `mu` to pass in the assumed mean.*

Question 0.0.3 *Define a function, center by:*

```
center <- function(x) x - mean(x)
```

[8] There is a special argument `...` that you will meet if you go far enough with R.

[9] The exception being functions with only one expression, as for these the braces are optional.

[10] When a variable is not passed in, but still used within the body of a function, then R has "scoping" rules to determine where a value is to be taken from. Scoping rules can get tricky.

[11] We did this with `apply`, say.

Verify that we can center the variables of a data frame by using `sapply`*:*

```
x <- data.frame(a = 1:3, b = 5:7)
sapply(x, center)
```

Assessing a simulation

We typically find ourselves simulating an unknown probability distribution. In the following example, however, we *know* the sampling distribution to be normal with mean 0 and standard deviation $1/\sqrt{10}$ (The key fact is that sums of independent normals are normal).

```
res <- replicate(1000, {
      x <- rnorm(1)
      mean(x)
  })
```

However, supposing we didn't know the distribution of our data, we could look at the simulation results either numerically with either of these

```
summary(res)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-2.744000 -0.654400  0.025290  0.008223  0.727900  3.052000
```

```
c(mean(res), sd(res))
```

```
[1] 0.008223174 1.005748390
```

Or graphically [12] with either a histogram, density plot, boxplot, or quantile plot. For instance a density plot and quantile plot using lattice graphics. (Figure 1.)

[12] As did Gosset

The following shows how to simulate the t-statistic when $n = 4$ and $\mu = 15.5$. We first define a function to find the t-statistic:

```
tstat <- function(x, mu = 0) (mean(x) - mu)/(sd(x)/sqrt(length(x)))
```

Then we can use this as follows

```
n <- 4
mu <- 15.5
res <- replicate(1000, {
      x <- rnorm(n, mean = mu)
      tstat(x, mu)
  })
```

We can compare this to the normal, and then the t-distribution with $n - 1 = 3$ degrees of freedom using a quantile plot (Figure 2).

```
library(lattice)
densityplot(res)

qqmath(~res)
```

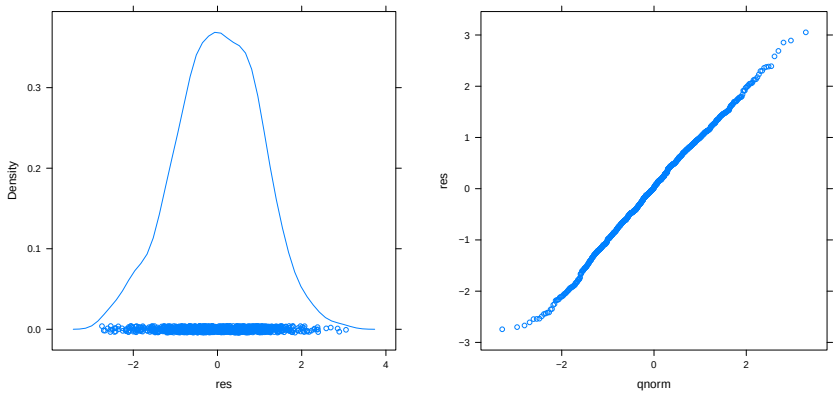


Figure 1: Density plot and quantile plot of simulation to find sampling distribution of $\bar{x}$ for a normal population. The theoretical distribution is normal with mean 0 and standard deviation $1/\sqrt{10}$.

```
qqnorm(res, main = "Is the sampling distribution normal?")
qqline(res)

qqplot(res, rt(1000, df = 4 - 1))
```

Question 0.0.4 *The central limit theorem basically says that the sample mean is approximately normal. More precisely, it says $\bar{x}_n$, in the limit, after rescaling, has the same quantiles as the standard normal. This means that a quantile-normal plot of $\bar{x}$ (which ignores scaling) should be approximately a straight line.*

Let the sample be given by an exponential with mean and standard deviation 1. For this we have for a given n, a single value of

```
n <- 5
replicate(4, mean(rexp(n)))
```

```
[1] 0.6073047 0.6710993 0.4965685 1.5745952
```

Change n and using 1,000 simulations for $n = 2, 5, 25$, and 50. Compare your simulations to the normal quantiles using `qqnorm`*. For which values of n is the quantile-normal plot essentially straight?*

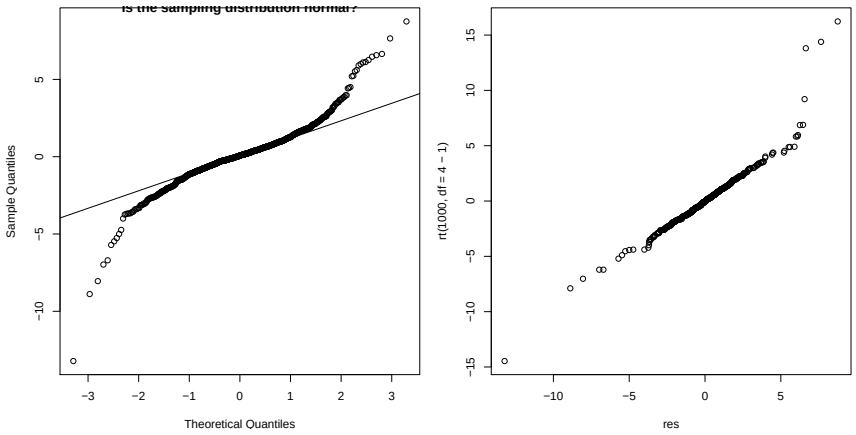


Figure 2: Quantile plots of the simulated sampling distribution of the t-statistic. The first plot compares the distribution to the normal distribution, the second to the t-distribution with 3 degrees of freedom.

Question 0.0.5 *The t-statistic has the t-distribution if the parent population is normal. What if the parent population has short tails?*

Generate random samples with the uniform on $[-1, 1]$ (which has mean 0) as follows (when $n = 5$) (you replace m with 1000)

```
m <- 4
n <- 5
replicate(m, {
      x <- runif(n, -1, 1)
      tstat(x, 0)
  })
```

```
[1] -0.1989756  1.1409462  0.3019271 -0.6737141
```

For $n = 5, 10$ and 50 compare the result of a simulation with the t-distribution with $n - 1$ degrees of freedom.

Is there a difference in the distributions?

Question 0.0.6 *The t-statistic has the t-distribution if the parent population is normal. What if the parent population has long tails?*

Generate random samples with the t-distribution with 3 degrees of freedom (which has mean 0) as follows (when $n = 5$) (replace `m` with `1000`):

```
m <- 4
n <- 5
replicate(m, {
      x <- rt(n, df = 3)
      tstat(x, 0)
  })
```

```
[1]  0.9076513 -2.2809001 -0.4400149  0.4782340
```

For $n = 5, 10$ and 50 compare the result of a simulation with the t-distribution with $n - 1$ degrees of freedom.

Is there a difference in the distributions?

Interactive graphics in RStudio

If you are using R Studio, the `manipulate` package can be used to animate graphics. Copy and past the following into the R session then adjust the slider to see the central limit theorem:

```
m <- 100
n <- 10
tdist3 <- function(n) rt(n, df = 3)
tdist10 <- function(n) rt(n, df = 10)
tdist50 <- function(n) rt(n, df = 50)
require(manipulate)
manipulate({
      res <- replicate(m, {
          x <- do.call(dist, list(n = n))
          SE <- sd(x)/sqrt(n)
          mean(x)/SE
      })
      qqnorm(res)
  }, dist = picker("rnorm", "rexp", "runif", "tdist3", "tdist10",
      "tdist50"), n = slider(2, 100))
```

Permutation methods

Some statistical inference can be carried out with reduced population assumptions because certain sampling distributions can be computed by considering permutations of the data. The term permutation methods covers two cases:

randomization tests where the random assignment of subjects to treatment groups is exploited

permutation tests where there is an invariance to permutations

Historically there were few permutation methods where the sampling distributions could be worked out *by hand*, but recent advancements in theory and in computational power have changed that. In particular, simulations can be done to investigate previously unattainable sampling distributions, as will be seen in this section.

[Quite a bit of this section is optional, and is here to give the interested student some extra material.]

The outcome of a significance test is the production of a p-value. In order to do so with a t-test, there are necessary assumptions made about the population that allow one to describe the sampling distribution of the test statistic, in this case the t-statistic. Some times these assumptions about the population are valid, but of course, sometimes they are not. When they are not, different test statistics are needed. One class of tests with known sampling distributions are *permutation methods*. The sampling distributions related to these, are computed by considering possible permuations of the data. At one point you likely learned formulas for permuations, and you may even recall that there are **lots** of permuations, even when there are not many objects. As such, explicit enumeration of permuations can be quite complicated – although we'll see that the computer is quite happy to do the work for us, up to a point

So, in one light permuation methods may be seen as a way of trading off stronger assumptions about a population for more complicated means to compute p-values of a test statistic.

However, permuation methods have a deeper role in the understanding of the use of statistical tests. In an interesting article by George Cobb The Introductory Statistics Course: A Ptolemaic Curriculum published at Technology Innovations in Statistics Education the author argues for the centrality of permutation methods in the teaching of statistics, rather than tests based on the normal distribution, such as the t-test. This is because the underlying reasoning of the randomization test matches the production of the data.

In the article, the example of a randomized controlled experiment is used. As learned in most introductory statistics courses, this experiment involves a treatment, for instance some kind of care after a knee surgery and some measure of the outcome, say time to recovery.

The basic idea is we administer the treatment and then record the outcome. How do we draw inference about population parameters from the outcomes in a sample? Significance tests are one way.

Now a typical problem with the above experimental design is the *placebo effect* – people report getting better faster, as they believe they are supposed to. To avoid this, a *control group* is often created. That is the group of people in the study are split into two: a control group which may get a placebo and a treatment group which gets the actual treatment. Then differences in the outcome may be attributed to differences in the treatment, as the placebo effect may be expected to equally alter both groups.

However, even this has issues: first, the people in the sample may need to somehow represent a larger population that we wish to

draw generalizations about and second, the difference in the outcome measure between the two groups should not be attributable to the composition of the groups or other confounding variables.

The first case ("selection of units") is addressed by randomly selecting participants for the study from a wider population. This may not always be possible in practice, but when it is, it allows the researcher to use the rules of probability to control the samples that are somehow not representative of the population.

The second case ("allocation of units to groups") is also addressed by randomization. Consider the case of the "crooked researcher" who puts those he think will score low in the control group and those who will score high in the treatment group. Then difference in the outcome measure may be attributed to difference in the groups. By randomizing, you ensure that such cases are controlled by the laws of probability. This may allow the researcher to draw causal inferences.

This second randomization gives rise to a randomization test alternative to the two-sample t-test which would often be suggested when comparing two independent samples.

A randomization test

For example, suppose 12 identical knee surgeries were performed. Through randomization, 6 subjects were given the usual protocol for rehabilitation (the control group) and the other 6 were given a different set of instructions that the researcher thought might offer an improvement. The time to achieve 90-degree range of motion was used to measure the outcome. Our data may look like

```
control:   26 27 22 26 27 22
treatment: 18 16 17 25 18 24
```

Note that these people do not represent the wider population, nor are they likely to be randomly chosen from some population. What is random here is the allocation of the patient to the treatment group.

In R we could use the t-test to analyze this significance test as follows:

H_0 : no difference in means H_A : control mean is greater

```
ctrl <- c(26, 27, 22, 26, 27, 22)
treat <- c(18, 16, 17, 25, 18, 24)
t.test(ctrl, treat, alt = "greater")
```

```
        Welch Two Sample t-test
```

```
data:  ctrl and treat
t = 2.9019, df = 8.332, p-value = 0.009499
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.93331     Inf
sample estimates:
mean of x mean of y
 25.00000  19.66667
```

This produces a small p-value indicating a a poor fit between the data and the null hypothesis of no difference.

Of course the t-test has assumptions – normality of the populations and independence of the samples – that may not be appropriate. By randomization, we have independence, but the normal population is suspect. What we do know, however, is that under the assumption of no difference between the populations *and* under the assumption of random allocation of units to treatment groups that it shouldn't matter what unit got assigned to which group.

That is, we shouldn't be able to distinguish differences in the outcomes regardless of which 6 people we could have assigned to the control group *assuming* the null hypothesis is true.

Our observed difference in means is

```
mean(ctrl) - mean(treat)
```

```
[1] 5.333333
```

How unlikely is that? Well *were* we to have chosen the 6 for each group differently we would likely have seen some different value. Looking at all such possible values allows us to see if our one observed one is unusual. We can use the `sample` function to choose 6 different values from all the data to compare. Below we take advantage of negative subscripting [13] to split the data into two groups.

[13] cf. `?"["` to review R's subscripting conventions.

```
alldata <- c(ctrl, treat)
ind <- sample(1:length(alldata), 6)
mean(alldata[ind]) - mean(alldata[-ind])
```

```
[1] -2.666667
```

That is one possible other value assuming there is no difference and we had chosen a different randomization. Our value of 5.33 can be seen as being a big value or not so big value by comparing it to *all* possible different randomizations.

The set of all randomizations is found by listing all possible ways to pick 6 items from 12 where order is unimportant. You may know there are 12 choose 6 ways to do this:

```
choose(12, 6)
```

```
[1] 924
```

But how do we get a listing of these? The `combn` function will do this. It returns a matrix where each column is a possible choice for randomization. For example, a smaller set of numbers produces: [14]

[14] Five choose 2 counts all combinations of size 2 from 5 items

```
combn(5, 2)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    2    2    2    3    3     4
[2,]    2    3    4    5    3    4    5    4    5     5
```

So we simply loop over these possible randomizations and store the computed difference of means:

```
allRandomizations <- combn(12, 6)
n <- choose(12, 6)
res <- numeric(n)
for (j in 1:n) {
      ind <- allRandomizations[, j]
      res[j] <- mean(alldata[ind]) - mean(alldata[-ind])
  }
```

How unusual (large) was our observed difference of 5.33?

```
sum(res > (mean(ctrl) - mean(treat)))/n
```

```
[1] 0.00974026
```

As an aside, 0ne can use `apply` as an alternate to `for`:

```
allRandomizations <- combn(12, 6)
n <- choose(12, 6)
res <- apply(allRandomizations, 2, function(ind) {
      mean(alldata[ind]) - mean(alldata[-ind])
  })
```

The above computes a p-value, showing that 5.33 is unusually large indicating that the null hypothesis does a poor job of explaining the data. This, of course, is similar to before and was more work. But there with fewer assumptions about the populations the data come from. All we assumed was that randomization was used to assign units to treatment groups and that under the null the two populations were identical. [15]

This specific examples shows naturally how randomization tests can be used in place of other significance tests. In the following we illustrate other typical uses and their implementation in R.

[15] Same data, different p-values. At first glance this may seem odd, but keep in mind the assumptions about the data lead to knowledge about the sampling distribution of the test statistic. Generally, fewer assumptions means less power.

The rank-sum test

Suppose a food writer was interested in testing whether a change of ingredient made a noticeable difference in a finished product. To test, the food writer made two pies, one with the change of ingredient and one without. Then she randomly assigned 10 people to test the pies and rank on a scale of 1 to 10. The data recorded is

```
taster | 1  2  3  4  5 | n xbar   s
-------------------------------------
pie 1  | 4  5  4  6  4 | 5  4.6  0.89
pie 2  | 5  6  6  7  5 | 5  5.8  0.83
```

Is there a noticeable difference in pie 1? A test of $H_0 : \mu_1 = \mu_2$ against an alternative $H_A : \mu_1 < \mu_2$ yields a small p-value:

```
pie1 <- c(4, 5, 4, 6, 4)
pie2 <- c(5, 6, 6, 7, 5)
t.test(pie1, pie2)

        Welch Two Sample t-test

data:  pie1 and pie2
t = -2.1909, df = 7.965, p-value = 0.05999
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.46402827  0.06402827
sample estimates:
mean of x mean of y
      4.6       5.8
```

But does the assumption of normality apply here? The data is not continuous and with small samples gauging normality always requires an act of faith.

Although a t-test is fairly robust to such deviations from normality, we can try a different test statistic for which a sampling distribution can be computed.

This sampling distribution will be computed by looking at all possible permutations of the underlying data.

It appears that the data for the first pie is "less" than that for the second pie. Why? If the pies were equivalent, we would expect that the number of 4's would be more or less equally spread out amongst both pies, as with the number of 7's etc. This is because the distribution of pie-ratings should be identical for each pie. Why equally? Because a 4 would be as likely to be assigned to pie 1 as pie 2. So here equally means on average. Technically, each arrangement of the ranks should be equally likely, its just that most arrangements have a more or less "equal" distribution. A measure,

or statistic, of how the ranks are given might be to add up all the ranks associated with one of the pies. For this problem, we first arrange the ratings from smallest to largest. Where there are ties, we take an average of the ranks.

```
value | 4 4 4 5 5 5 6 6 6 7
rank  | 2 2 2 5 5 5 8 8 8 10
pie   | 1 1 1 1 2 2 1 2 2 2
```

The ranks should add to $55 = 1 + 2 + ... + 10$ and they do. The ranks for the first pie are only $19 = 2 + 2 + 2 + 5 + 8$. Is this an unusually small number? How could we tell? In order to do so, we need to know what the distribution is for the sum of the ranks under the assumption that the two populations are identical.

This distribution can be computed exactly, with the aid of the computer as the assumptions allow it to be computed using permutations. If we assume that each rank is equally likely to have come from pie 1 or pie 2, then the distribution of the rankings is found by looking at all the possible assignments of 5 of the ranks to pie 1. There are 252 (10 choose 5) equally likely ways to assign the 5 ranks to the first pie. Of these how many create a value of 19 or less?

R can happily compute a problem of this size. In this instance the details are a bit tricky, but we'll show them below. First, the combinations are created by the function `combn`. This function returns a matrix with 5 rows and 252 columns, each column being one of the combinations. For each column then we want to sum the ranks corresponding to these values. This can be achieved with indexing and the `sum` function. To do all 252 combinations at once, the `sapply` function can be used. This will apply a function to each component of a list (columns of a data frame) and return a data vector or list as appropriate. To use it on the output of `combn` we coerce the matrix to a data frame below.

```
theRanks <- c(2, 2, 2, 5, 5, 5, 8, 8, 8, 10)
theCmbns <- as.data.frame(combn(10, 5))
vals <- sapply(theCmbns, function(i) sum(theRanks[i]))
```

We can visualize the distribution with a table

```
table(vals)
```

```
vals
16 19 21 22 24 25 27 28 30 31 33 34 36 39
 3 12  3 30 12 36 30 30 36 12 30  3 12  3
```

We are interested in how likely it is to get a 19 or less. This is computed with

```
sum(vals <= 19)/length(vals)
```

```
[1] 0.05952381
```

We get a p-value of 0.0595 for this test.

For the above test we could do this calculation exactly using R. For that particular problem, the test is called the Mann-Whitney test or the Wilcoxon Rank Sum test and is one of the standard non-parametric significance tests. The only assumption about the populations is that the two are identical up to a possible shift in center. (Technically we assume the p.d.f.'s satistify $f_1(x) = f_2(x - c)$ where c is the possible shift. In the null hypothesis c is assumed to be 0.)

The Wilcoxon-Mann-Whitney test is implemented in the `wilcox.test` function of R (see Section 8.6.3 of UsingR), although that implementation covers the case where the data is continuous so there are no ties. (The help page of `wilcox.test` refers the reader to the `exactRankTests` package for the case with ties.)

The `wilcox.test` function is used in a manner similar to `t.test`. For a two sample test it can take either two data vectors or a formula interface. As our data is stored in two data vectors we simply have

```
wilcox.test(pie1, pie2, alternative = "less")
```

```
        Wilcoxon rank sum test with continuity correction

data:  pie1 and pie2
W = 4, p-value = 0.04133
alternative hypothesis: true location shift is less than 0
```

The p value is not the same as computed above due to the ties in the data and the continuity correction employed. Later, in our brief description of the `coin` package we illustrate how its `wilcox_test` function can produce the correct p-value.

Before leaving this example, it should be noted that although a two-sample t-test or the Mann-Whitney test test similar things there are reasons to use one over the other. At first glance it would appear that the Mann-Whitney test, being more general, should be the preferred test. But there are issues, primarily:

- The Mann-Whitney test has far less power than the t-test. Practically speaking, this means you need larger samples or bigger differences for the test to reject when the null is not actually true
- Secondly, as per nih.gov the permutation methods do poorly when the assumption of identical populations is not met.

This is different from power, which computes the probability of accepting when c is not 0 where $f(x)$ and $f(x-c)$ are the two p.d.f.'s for the distributions. (Same f means same shape!) The abstract indicates that you can have inflated type I errors (rejecting H_0 erroneously) when the means are the same, but the shapes are not.

Test of means

Suppose a test is designed to see if online-instruction has a similar performance as face-to-face instruction. Two classes are run with each cohort being randomly assigned from a homogeneous pool of students. At the end of the course their scores on a similar exam are analyzed. Suppose the collected data is

```
online | 61 59 43 56 33 71 49 54 43 53 24
-----------------------------------------------------------------------
face2  | 33 46 10 37 42 37 52 60 28
```

First we compute a t-test, assuming equal variances:

```
online <- c(61, 59, 43, 56, 33, 71, 49, 54, 43, 53, 24)
face2 <- c(33, 46, 10, 37, 42, 37, 52, 60, 28)
t.test(online, face2, var.equal = TRUE)
```

```
        Two Sample t-test

data:  online and face2
t = 1.82, df = 18, p-value = 0.08543
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.744615 24.350675
sample estimates:
mean of x mean of y
 49.63636  38.33333
```

At first glance, the two means appear different but the p-value indicates only borderline significance. The assumptions for this t-test are two normally distributed populations with equal variances. We are interested in detecting a change in center.

What if normality is not a valid assumption? Can we compute the p-value a different way? Under the null hypothesis (equal means, and the underlying assumption that the two populations differ by at most a shift) each score is equally likely to have come from either group. So we could see how likely our difference in sample means is for all possible permutations of the scores which assign 11 scores to the online group and 9 to the face-to-face group.

There are 167,960 ways (`choose(20,11)`) ways to assign the scores. This is a big number, with some time we can compute the

exact distribution of the test statistic. We do so below, but will next show how to approximate this work using a random sample.

```
scores <- c(online, face2)
teststat <- function(ind) mean(scores[ind]) - mean(scores[-ind])
allCombns <- data.frame(combn(20, 11))
res <- sapply(allCombns, teststat)
```

That takes awhile to compute, but once computed we can compute a p-value for the probability our test statistic is more than the observed value by looking at the proportion of the combinations that produce a greater value:

```
sum(res >= (mean(online) - mean(face2)))/length(res)
```

```
[1] 0.04497499
```

The two-sided test would be basically double this as the sampling distribution is nearly symmetric about 0.

Using a simulation to see the sampling distribution

If the time involved is too long to generate all the permutations, then a simulation can be used. To generate a single random combination the `sample` command without replacement can be used, as in

```
sample(1:20, 11)
```

```
 [1] 16  2  4 19 18  6  9  5  3 20 13
```

The first argument is what we sample from, the second the number of samples. By default sampling *without replacement* is used. If the second argument is omitted, then one of the 20! rearrangements of the 20 terms is returned (known as a permuation):

```
sample(1:20)
```

```
 [1] 20  8  4  5 19 16 15 10  1  9 12 11  7  6  3  2 13 18 17 14
```

So the exact p-value above is approximated by the randomly generated one below:

```
res <- replicate(1000, teststat(sample(1:20, 11)))
sum(res > (mean(online) - mean(face2)))/length(res)
```

```
[1] 0.047
```

Correlation test

The `cor.test` for whether two samples are correlated is based on an assumption of normal populations. The tested hypotheses are

$$H_0 : r = 0, \quad H_A : r >, <, \text{ or } \neq 0$$

where r is the population correlation (`cor(x,y)` returns the *sample* correlation).

Under the null hypothesis, if the values are uncorrelated then permuting the y values shouldn't change the correlation. This is not the case for correlated variables.

This example might illustrate this.

```
x = rnorm(100)
y1 = rnorm(100)  # uncorrelated
cor(x,y1)
```

```
[1] -0.04829928
```

```
y2 = rnorm(100, mean=x) # correlated
cor(x,y2)
```

```
[1] 0.7488316
```

But notice when we shuffle up the y-values we get different results.

```
cor(x, sample(y1))
```

```
[1] 0.1710046
```

```
cor(x, sample(y2))
```

```
[1] -0.08917016
```

If our test statistic is simply `cor(x,y)` then a significance test can be made by considering how likely it is that a different permutation would produce a value as large as the given correlation. This is the Fisher-Pitman correlation test and requires no assumptions on the underlying populations.

Let's look at some sample data. If we record a random sample of ages and weights from a population of 6-foot males we have

```
age    |  25  50  24  60  35  42  52
-----------------------------------------------
weight| 180 175 200 210 185 185 195
```

Is there evidence that the population correlation is 0?

If we assumed that the populations were normally distributed we've seen that the `cor.test` gives a two sided p-value of

```
age <- c(25, 50, 24, 60, 35, 42, 52)
weight <- c(180, 175, 200, 210, 185, 185, 195)
cor.test(age, weight)
```

```
        Pearson's product-moment correlation

data:  age and weight
t = 0.7299, df = 5, p-value = 0.4982
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.5777505  0.8619507
sample estimates:
      cor
0.3103228
```

If we were worried about the normality assumption, we can do an exact test based on permutations. As there are 7 cases, there are $7! = 5{,}046$ possible permutations. [16]

We do this compuations using simulation. The `sample` function simply permutes the values in `weight` below so that each time the `replicate` function call `cor` a newly shuffled `weight` variable is used:

[16] This is small enough to do without simulation, but we won't pursue that. Those who are interested might find the `permutations` function from the `e1071` package to be useful to generate the possible permuations.

```
res <- replicate(1000, {
      cor(age, sample(weight))
  })
t.obs <- cor(age, weight)
sum(res <= -abs(t.obs) | res >= abs(t.obs))/length(res)
```

```
[1] 0.51
```

Question 0.0.7 *Use a permutation test to test if these two independent samples have the same median*

```
x |  8  5  3  1  2 12  0  3 12 21 20 13
--------------------------
y |  1  7 20 17  1  8  1  5  9  1
```

You may assume the populations are identical up to, perhaps, a shift in their center.

Which test do you use? Is it already implemented, or do you have to find all the terms yourself?

Question 0.0.8 *The `home` data set in `UsingR` has data on old and new assessed values for 15 homes randomly sampled from a towns housing stock. Use a permutation test to test*

$$H_0 : r = 0 \quad H_A : r > 0$$

where r is the population correlation coefficient.

The coin package

[This is only for the interested reader. It discusses a package in R giving a general framework for permutation tests.]

A general framework for many more permutation tests is provided by the coin package. This implementation is beyond the scope of this introductory class, but we sketch out briefly what the package provides below for the interested reader.

The coin package is used to handle the following general class of problems, which we describe briefly here only to give a flavor. There are two samples, x and y whose sampling distributions are not known. The interest is in a test whose null hypothesis is that the two populations are independent. One way to write this is to express the conditional distribution of one on the other, say y on x. Then a certain type of test statistic, such as the rank sum, is considered. The actual class of statistics is multivariate linear statistics of a certain form which are then turned into univariate statistics in one of two ways. The distribution of these test statistics, in general, will not be known. However, under the null hypothesis, as the values of x do not influence the values of the y, one can fix these values of x. The special form of the test statitistic then ensures that the sampling distribution does not depend on a permutation of the y variables. In this case, the probability the test statistic is less (more) than a certain value is given by the proportion of all permutations which produce a value less (more) than the certain value.

The proper setup of the above allows for analysis in a number of ways.

- If the problem is small enough all the possible permutations can be considered. Such tests are a class of *exact tests*. However, in general:
- If the number of permutations is too large to make their enumeration feasible then a simulation can be run to estimate the p-value.
- If the problem size is large enough, then there are asymptotic results about the problem that allow the normal distribution to be used. This is similar to what occurs with the Wilcoxon rank-sum test, where for large n a normal approximation is used for the sampling distribution.

The coin package implements a wide range of permutation tests, and is flexible enough to allow the user to implement new tests with out the need of complicated coding. However, defining the statistic and the problem is more complicated than the level of this course. In the following we mention just a single application. The package itself contains two vignettes for the interested reader.

The rank-sum test

As mentioned the Wilcoxon rank-sum test performed in the first example is implemented in the `wilcox.test` function. This function is called in a manner similar to the t-test. Here is the test that

$$H_0 : \mu_1 = \mu_2, \quad H_A : \mu_1 < \mu_2$$

The test is non-parametric, but there is an assumption on the population, namely that the two populations are symmetric. This ensures that under H_0 that a given rank is equally likely to have come from either population.

To illustrate, we revisit the pie-rater problem, where we saw:

```
wilcox.test(pie1, pie2, alt = "less")
```

```
        Wilcoxon rank sum test with continuity correction

data:  pie1 and pie2
W = 4, p-value = 0.04133
alternative hypothesis: true location shift is less than 0
```

The small p-value is a bit misleading, as the warning message indicates that exact p-values are not computed.

The `wilcox_test` function in the `coin` package can compute these. First, we show how to install the coin package if it isn't already:

```
> install.packages("coin", dep=TRUE)
```

Now we load the package.

```
library(coin)
```

The function has a similar name with the dot "`.`" replaced by an underscore "`_`". The function needs to be called using R's formula notation. We first stack the data to do so.

```
d <- stack(data.frame(pie1, pie2))
wilcox_test(values ~ ind, data = d, alt = "less")
```

```
        Asymptotic Wilcoxon Mann-Whitney Rank Sum Test

data:  values by ind (pie1, pie2)
Z = -1.8439, p-value = 0.0326
alternative hypothesis: true mu is less than 0
```

The key above is the term *Asymptotic*.

The returned p-value is not computed with the exact distribution. The default is to compute using an assumption of large samples. This isn't the case here, so we have to ask for the "exact" distribution to be used.

```
wilcox_test(values ~ ind, data = d, alt = "less", distribution = "exact")
```

```
        Exact Wilcoxon Mann-Whitney Rank Sum Test

data:  values by ind (pie1, pie2)
Z = -1.8439, p-value = 0.05952
alternative hypothesis: true mu is less than 0
```

Here the p-value previously computed the hard way is returned.

As an illustration, the argument `distribution = "approximate"` will compute the p-value using a simulation.

```
wilcox_test(values ~ ind, data = d, alt = "less", distribution = "approximate")
```

```
        Approximative Wilcoxon Mann-Whitney Rank Sum Test

data:  values by ind (pie1, pie2)
Z = -1.8439, p-value = 0.062
alternative hypothesis: true mu is less than 0
```

This p value will vary from run-to-run, but will be close to the exact one.

The Bootstrap

The bootstrap method is another computer intensive means to investigate distributions. Rather than rely on permutations, the boostrap method simulates an approximate distribution. For the boostrap we assume only that we have a random sample from a population and hence a single value for the statistic. From this, we "bootstrap" up to a understanding of the sampling distribution of the statistic.

Some background reading on the bootstrap method is contained in these resources:

- wikipedia has a nice description of the topic: `http://en.wikipedia.org/wiki/Resampling_(statistics)`
- A free chapter from the Moore and McCabe book, `http://bcs.whfreeman.com/ips5e/content/cat_080/pdf/moore14.pdf`, although a bit long, this provides a more complete exposition than what follows.
- Finally, if you have the book, then, as usual, Venables' and Ripley's *Modern Applied Statistics with S-Plus* has an authoritative, succinct treatment on pages 143-146.

The basic idea of the bootstrap

To investigate the bootstrap, we first fix notation. Let X denote a random sample and F the population. We are interested in using a statistic R to estimate or make inference about a parameter θ. We write the statistic $R(X, F)$ to emphasize it is based on a sample and reflects the distribution. The sampling distribution of $R - \theta$ is of interest as it may reflect information about F or θ. The idea of the bootstrap method is to understand the sampling distribution of $R(X, F) - \theta$ using the data that has been previously observed.

To make this less abstract, for the t-test we have $R(X, F)$ is basically the sample mean, $\bar{x}$ and θ is the population mean μ. We know the sampling distribution of $\bar{x} - \mu$: it is normal with mean 0 and standard deviation $\sigma/\sqrt{n}$, when F is the normal distribution.

The main idea of the bootstrap is a sample $X = \{X_1, X_2, \ldots, X_n\}$ from a population gives rise to $\hat{F}$, the empirical distribution from which we subsequently sample. [17]

From this sample, let $X_1^*, X_2^*, \ldots, X_n^*$ be a *resample* of size n *with replacement*. This resample gives rise to a new random variable $R(X^*, \hat{F})$ where we simply compute the statistic for the new data. As the resample reflects $\hat{F}$ which in turn reflects F, the distribution of this new random variable should give insight into the distribution of the original. What kind of insight? We may be interested in the center, as there may be a bias introduced by the process; the spread, to understand the inherent variability; and the shape.

Now, why do we know anymore about the distribution of $R(X^*, \hat{F})$? Because we can *simulate* this random variable. [18]

First let us consider the concrete example above where the underlying population is known. This allows us to see what issues may be involved.

Let F be the normal distribution and with $\mu = 5$ and $\sigma = 2$ and let $n = 10$ (our sample is $X_1, X_2, \ldots, X_{10}$). Suppose we want to know about $\bar{X} - \mu$. (R is the mean.) In this case, we know the answer: the distribution is normal with mean 0 (unbiased) and variance $\sigma^2/10$.

Although in this case we don't need the bootstrap, if we used it then we would consider a sample with replacement of size 10 from the original sample of size 10, call this X^*. Then this new sample produces a mean $\bar{X}^*$. The bootstrap method looks at the distribution of $\bar{X}^*$.

The sample function

We will use the `sample` function to take a sample from a set of numbers. By default this function randomly samples a certain number of values from a specified set *without* replacement:

[17] The empirical distribution assigns probability i/n to an interval A if A contains exactly i of the n data points.

[18] There are cases where a theoretical treatment is possible, but that's not our interest here.

```
items <- 1:5
sample(items, size = 3)
```

```
[1] 5 2 3
```

The `size` argument is optional, it defaults to the size of `items`:

```
sample(items)
```

```
[1] 3 4 1 5 2
```

In this case we get a rearrangement, as we sample without replacement. For the bootstrap method, we sample *with* replacement. This requires the `replace=TRUE` argument (abbreviated to `rep=TRUE` below)

```
sample(items, rep = TRUE)
```

```
[1] 2 3 1 3 4
```

That is we can resample from a data set as follows:

```
theOriginalSample <- rnorm(5)
theOriginalSample
```

```
[1]  0.17435367 -0.03117268 -0.98778796  0.29159359 -1.37368906
```

```
sample(theOriginalSample, rep = TRUE)
```

```
[1] -0.98778796  0.17435367 -1.37368906 -0.03117268 -0.98778796
```

Question 0.0.9 *Just for practice – or convenience – can you define a `resample` function?*

A bootstrap example

With the `sample` function we can do the simulation above to find the bootstrap distibution. First we define the original sample:

```
X <- rnorm(10, mean = 5, sd = 2)
```

Now we use `replicate` to compute 1,000 bootstrap samples

```
res <- replicate(1000, {
    Xs <- sample(X, rep=TRUE)
    mean(Xs) -5
  })
```

These resulting values are termed the *replicates* to distinguish them from the original sample.

How do we interpret this output?

In this case the statistic $\bar{x}$ is an unbiased estimator for the mean, μ. In general we don't know this for the population. In the notation above, the bias is the expected value of $R(X, F) - \theta$. This is approximated by the bias in $R(X^*, F) - R(X, F)$, that is the difference between the value of the statistic and the center of the bootstrap simulation. For our data this bias is estimated by

```
mean(res) - (mean(X) - 5)
```

```
[1] 0.01839669
```

and the spread by either the IQR or standard deviation

```
c(IQR = IQR(res), sd = sd(res))
```

```
     IQR       sd
0.854959 0.607704
```

This indicates that the bootstrap sample is unbiased in agreement with the fact that $\bar{x}$ is an unbiased estimator for μ.

We could graphically assess this with a boxplot (Fig.3).

For this example, we know that the standard deviation of $\bar{x}$ is $\sigma/\sqrt{10}$ and this compares with the result above:

```
sd(res) - 2/sqrt(10)
```

```
[1] -0.02475152
```

The shape of our bootstrap sample is bell shaped, as can be seen from the histogram, say:

```
hist(res - (mean(X) - 5))
abline(v = 0, col = "red", lwd = 2)
```

Assuming normality we can use the known percentiles of the normal distribution to compute confidence intervals. In this case we would get the following 95% confidence interval:

```
mean(res) + c(-1, 1) * 1.96 * sd(res)
```

```
[1] -0.618728  1.763472
```

The center (`mean(res)`) does not reflect the possible bias. This is corrected for when we use the `boot` package.

The shape need not be normally distributed. In that case, confidence intervals are often computed from the percentiles of the bootstrap simulation. These are termed *percentile confidence intervals*. From our sample we have

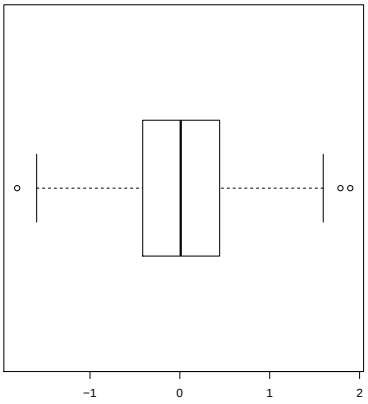


Figure 3: Boxplot of bootstrap samples showing the unbiasedness of the statistic.

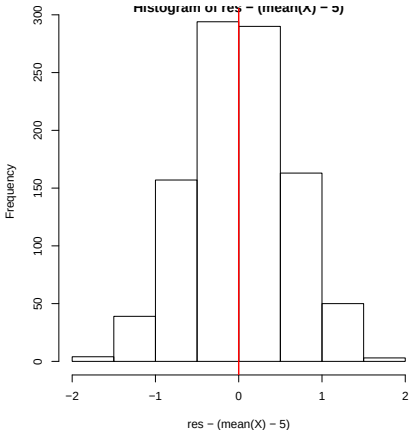


Figure 4: Histogram of the data from our simulation of $\bar{X}^* - \mu$.

```
alpha <- 0.05
quantile(res, c(alpha/2, 1 - alpha/2))
```

```
     2.5%      97.5%
-0.5840449  1.7545572
```

Both the above suffer from a bias. The more difficult to explain *BCa* intervals attempt to correct this bias. These will be shown when the `boot` package is introduced.

Bootstrapping the median

Our next example uses the `faithful` data set in the `MASS` package. First we assign the data to the variable `erupt`.

```
library(MASS)
erupt <- faithful$eruptions
```

This is a bimodal data set containing the time between eruptions of the Old Faithful Geyser. Our goal is to make inference on the median of the population.

Of course the sample median is found with

```
median(erupt)
```

```
[1] 4
```

but how can we relate this to the population median?

We first perform a bootstrap simulation

```
res <- replicate(1000, median(sample(erupt, rep = TRUE)))
```

We look at the possible bias of using the median for the estimator

```
mean(res) - median(erupt)
```

```
[1] -0.0174125
```

There appears to be none, although we need to consider the variability to be certain.

The shape of the bootstrap distribution does not appear to be normal, so we use the percentile confidence intervals to find a 95% confidence interval

```
alpha <- 0.05
quantile(res, c(alpha/2, 1 - alpha/2))
```

```
 2.5% 97.5%
3.833 4.100
```

The trimmed mean

The trimmed mean, $\bar{x}_t$ is often suggested to estimate a population center when the data is long tailed. The trimming lessens the effects of the outliers, but unlike the median, combines information from typical points near the center.

In this example, we look into using the bootstrap method to find a confidence interval for the population mean of the `erupt` data using the 25% trimmed mean.

First, we create the bootstrap sample. We use the `trim=0.25` argument for `mean()` to get a trimmed mean.

```
tmean <- function(x) mean(x, trim = 0.25)
res <- replicate(1000, tmean(sample(erupt, rep = TRUE)))

opar <- par(mar = c(3.5, 4.1, 0.1, 0.1))
boxplot(res - mean(erupt, trim = 0.25), horizontal = TRUE)
```

There appears (Figure 5) to be little bias between $\bar{x}_t^*$ and $\bar{x}_t$:

```
mean(res) - mean(erupt, trim = 0.25)
```

```
[1] 0.001325441
```

In fact the sampling distribution of the bootstrap statistic appears normal, so we use the normal quantiles to find a 95% confidence interval.

```
mean(res) + c(-1, 1) * 1.96 * sd(res)
```

```
[1] 3.450083 3.922641
```

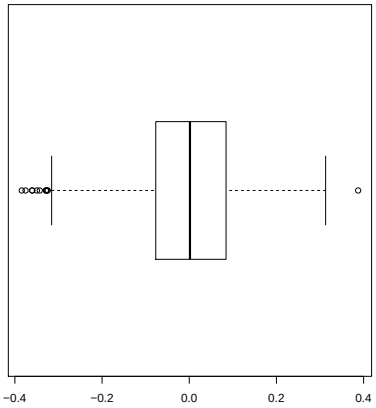


Figure 5: Boxplot of bootstrap replicates for trimmed mean. They show little bias.

The `boot` package

The `boot` package will automate the finding of bootstrap samples and compute confidence intervals in one of several ways. We show how to do the two of the previous examples using this package.

First you may need to install the package. It is available from CRAN, and may be installed from the menu bar or with a command like

```
> install.packages("boot")
```

If the package is installed, we can load the package with

```
library(boot)
```

The main function is `boot()`. To use it one must specify, at a minimum the data, the statistic and the number of replicates. The `boot` function does not use a `for` loop (for speed). So, one must specify the statistic a bit differently. In the upcoming example, we use a function of two variables.

To create 1,000 bootstrap replicates for the median we have

```
erupt.median <- boot(erupt, function(x, i) median(x[i]), R = 1000)
erupt.median
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = erupt, statistic = function(x, i) median(x[i]), R = 1000)


Bootstrap Statistics :
    original    bias    std. error
t1*        4 -0.015099  0.07868858
```

The printed summary shows the original value of the statistic for the data, the bias of the bootstrap data and the standard error of the bootstrap data.

As with linear models, the return value contains more information. The return value is a list. The `t` component contains the bootstrapped data. However, you may not need to access this. For example, to plot the data, the generic `plot` function is used (Figure 6.)

```
plot(erupt.median)
```

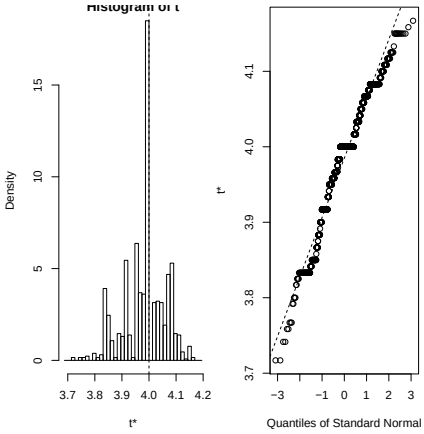


Figure 6: Plot of the bootstrap sample produced by `plot` showing non-normality

Confidence intervals are computed by the `boot.ci` function. The default shows 95% confidence intervals.

```
boot.ci(erupt.median)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = erupt.median)

Intervals :
Level      Normal              Basic
95%   ( 3.861,  4.169 )   ( 3.883,  4.167 )

Level     Percentile            BCa
95%   ( 3.833,  4.117 )   ( 3.781,  4.083 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

As noted, these confidence intervals are on the original scale. The "Normal" confidence intervals use the normal distribution to estimate the width and adjust for the bias. The "basic" intervals

are similar to the percentile confidence intervals only they are centered by the original statistic. The "BCa" are bias-corrected.

We repeat the above with the trimmed mean, only this time finding 90% confidence intervals to indicate how that parameter is changed.

```
erupt.trim <- boot(erupt, function(x, i) tmean(x[i]), R = 1000)
boot.ci(erupt.trim, conf = 0.9)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = erupt.trim, conf = 0.9)

Intervals :
Level      Normal              Basic
90%   ( 3.484,  3.877 )   ( 3.482,  3.885 )

Level     Percentile            BCa
90%   ( 3.485,  3.888 )   ( 3.479,  3.882 )
Calculations and Intervals on Original Scale
```

Extensions to the basic bootstrap

The basic bootstrap described above is only a start on the possibilities that have been explored. Some other possibilities are to bootstrap from a density estimate not from $\hat{F}$, or to bootstrap more complicated statistics such as the regression line.

Question 0.0.10 *Again for the* `erupt` *data, find confidence intervals using the bootstrap method for the standard deviation.*

Do the bootstrap replicates appear to be normally distributed?

Question 0.0.11 *Again for the* `erupt` *data, find confidence intervals using the bootstrap method for the* `mad`*. (The* `mad` *is the median of the deviations from the median and measures the center of a distribution in a more robust way than the mean.)*

Do the bootstrap replicates appear to be normally distributed?

Question 0.0.12 *Again for the* `erupt` *data, find confidence intervals using the bootstrap method for the skewness. The skewness is defined by*

```
skewness <- function(x) sum((x - mean(x))^3/sqrt(var(x))^3)/length(x)
```

It measures non-symmetry, or skew, in a data set. Do the bootstrap replicates appear to be normally distributed?

Answers to questions

Answer: 0.0.1 We have the following:

```
kurtosis <- function(x) {
      n <- length(x)
      z <- x - mean(x)
      ((1/n) * sum(z^4))/((1/n) * sum(z^2))^2 - 3
  }
```

Answer: 0.0.2 Try this:

```
tstat <- function(x, mu) {
      n <- length(x)
      SE <- sd(x)/sqrt(n)
      (mean(x) - mu)/SE
  }
```

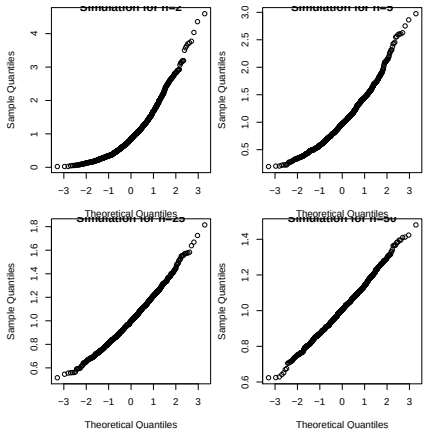
Answer: 0.0.3 Just type in to see:

```
center <- function(x) x - mean(x)
x <- data.frame(a = 1:3, b = 5:7)
sapply(x, center)
```

```
     a  b
[1,] -1 -1
[2,]  0  0
[3,]  1  1
```

Answer: 0.0.4 We use the `par` function with the `mfrow` argument to specify 4 graphics. Then we simulate

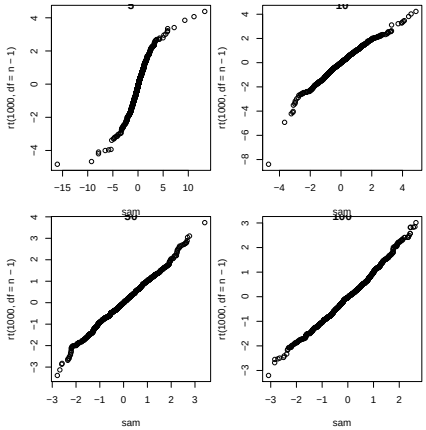
```
set.seed(10)
par(mfrow = c(2, 2))
for (n in c(2, 5, 25, 50)) {
      x <- replicate(1000, mean(rexp(n)))
      qqnorm(x, main = paste("Simulation for n=", n, sep = ""))
  }
```



In my simulation (with the random seed set), you can see a pronounced curve until $n = 50$.

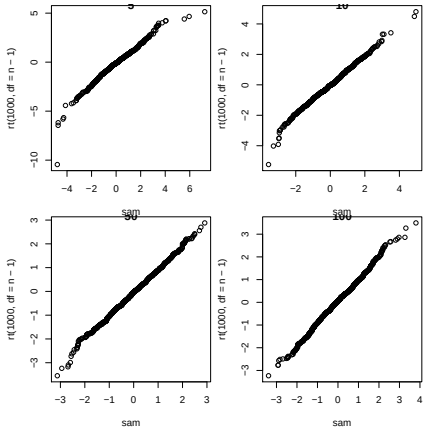
Answer: 0.0.5 We look at some quantile graphs as follows

```
par(mfrow = c(2, 2))
m <- 1000
for (n in c(5, 10, 50, 100)) {
      sam <- replicate(m, {
          x <- runif(n, -1, 1)
          tstat(x, 0)
      })
      qqplot(sam, rt(1000, df = n - 1), main = n)
  }
```



By $n = 10$, the graphs seem to have straightened out. Answer: 0.0.6 The solution is nearly identical to the previous.

```
par(mfrow = c(2, 2))
m <- 1000
for (n in c(5, 10, 50, 100)) {
      sam <- replicate(m, {
          x <- rt(n, df = n - 1)
          tstat(x, 0)
      })
      qqplot(sam, rt(1000, df = n - 1), main = n)
  }
```



By $n = 50$ things seem to be okay, but there is still a possible issue with the tail even with $n = 100$. Answer: 0.0.7 The test of median is implemented by the `wilcox.test` function. So we only need to enter the data and apply the function.

```
x <- c(8, 5, 3, 1, 2, 12, 0, 3, 12, 21, 20, 13)
y <- c(1, 7, 20, 17, 1, 8, 1, 5, 9, 1)
wilcox.test(x, y)
```

```
        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 68.5, p-value = 0.5952
alternative hypothesis: true location shift is not equal to 0
```

Answer: 0.0.8 Following the example, we simulate to mix around the y values:

```
library(UsingR)
r.obs <- with(home, cor(old, new))
res <- with(home, replicate(m, cor(old, sample(new))))
sum(abs(res) > abs(r.obs))/m
```

```
[1] 0
```

The 0 p-value is consistent with the fact that old and new home values are obviously correlated – an expensive house stays expensive after a new reassessment.
Answer: 0.0.9 The `resample` function just needs to have a new argument:

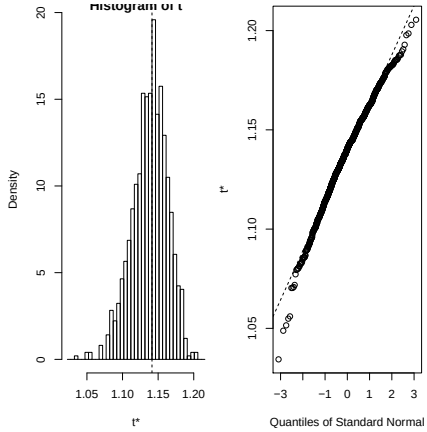
```
resample <- function(..., replace = FALSE) sample(..., replace = FALSE)
```

Answer: 0.0.10 We just need to modify the function in our call to `boot` to do this:

```
erupt <- faithful$eruptions
erupt.sd <- boot(erupt, function(x, i) sd(x[i]), R = 1000)
boot.ci(erupt.sd)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = erupt.sd)

Intervals :
Level      Normal              Basic
95%   ( 1.097,  1.192 )   ( 1.098,  1.195 )

Level     Percentile            BCa
95%   ( 1.088,  1.185 )   ( 1.091,  1.187 )
Calculations and Intervals on Original Scale
```

```
plot(erupt.sd)
```



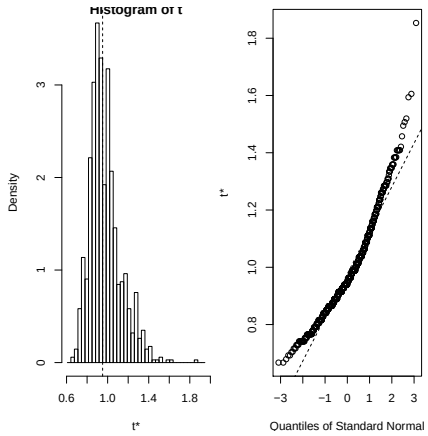
Answer: 0.0.11 We just need to modify the function in our call to boot to do this:

```
erupt <- faithful$eruptions
erupt.mad <- boot(erupt, function(x, i) mad(x[i]), R = 1000)
boot.ci(erupt.mad)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = erupt.mad)

Intervals :
```

```
Level      Normal              Basic
95%   ( 0.6178,  1.2150 )   ( 0.5427,  1.1490 )

Level     Percentile            BCa
95%   ( 0.7532,  1.3595 )   ( 0.7413,  1.2979 )
Calculations and Intervals on Original Scale
```

```
plot(erupt.mad)
```



Answer: 0.0.12 Again, the computation just involves modifying the previous work.

```
erupt <- faithful$eruptions
erupt.skew <- boot(erupt, function(x, i) skewness(x[i]), R = 1000)
boot.ci(erupt.skew)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = erupt.skew)

Intervals :
Level      Normal              Basic
95%   (-0.6263, -0.2065 )   (-0.6194, -0.2025 )

Level     Percentile            BCa
95%   (-0.6246, -0.2077 )   (-0.6417, -0.2135 )
Calculations and Intervals on Original Scale
```

```
plot(erupt.skew)
```