Tutorial 4: Inference & Hypothesis testing

Johannes Karreth

RPOS 517, Day 4

This tutorial shows you:

- how to sample from a variable
- how to create a sampling distribution
- how to use for-loops in R
- how to calculate and plot confidence intervals

Note on copying & pasting code from the PDF version of this tutorial: Please note that you may run into trouble if you copy & paste code from the PDF version of this tutorial into your R script. When the PDF is created, some characters (for instance, quotation marks or indentations) are converted into non-text characters that R won't recognize. To use code from this tutorial, please type it yourself into your R script or you may copy & paste code from the *source file* for this tutorial which is posted on my website.

Saving your code (as usual)

If you are working with code and don't need to produce an assignment or a report, save your code as an R script rather than an R Markdown document. An R script contains only R commands and no descriptive text. It should contain comments on your code. Comments are preceded by the `#` sign. Any code in the line after a `#` sign is not executed by R.

To create a new script, click on File -> New -> R Script. This will open a blank document above the console. As you go along you can copy and paste your code here and save it. This is a good way to keep track of your code and be able to reuse it later. To run your code from this document, highlight the code and hit the Run button, or highlight the code and hit command+enter or a mac or control+enter on a PC.

You'll also want to save this script (code document). To do so click on the disk icon. The first time you hit save, RStudio will ask for a file name; you can name it anything you like. Once you hit save you'll see the file appear under the Files tab in the lower right panel. You can reopen this file anytime by simply clicking on it.

Background

In the first part of this tutorial, we investigate the ways in which the statistics from a random sample of data can serve as point estimates for population parameters. We're interested in formulating a *sampling distribution* of our estimate in order to learn about the properties of the estimate, such as its distribution.

The data

We consider real estate data from the city of Ames, Iowa. The details of every real estate transaction in Ames is recorded by the City Assessor's office. Our particular focus for this tutorial will be all residential home sales in Ames between 2006 and 2010. This collection represents our population of interest. In this tutorial we would like to learn about these home sales by taking smaller samples from the full population. Let's load the data.

```r
ames <- read.csv("http://www.jkarreth.net/files/ames.csv")
```

We see that there are quite a few variables in the data set, enough to do a very in-depth analysis. For this tutorial, we'll restrict our attention to just two of the variables: the above ground living area of the house in square feet (`Gr.Liv.Area`) and the sale price (`SalePrice`). To save some effort throughout the tutorial, create two variables with short names that represent these two variables.

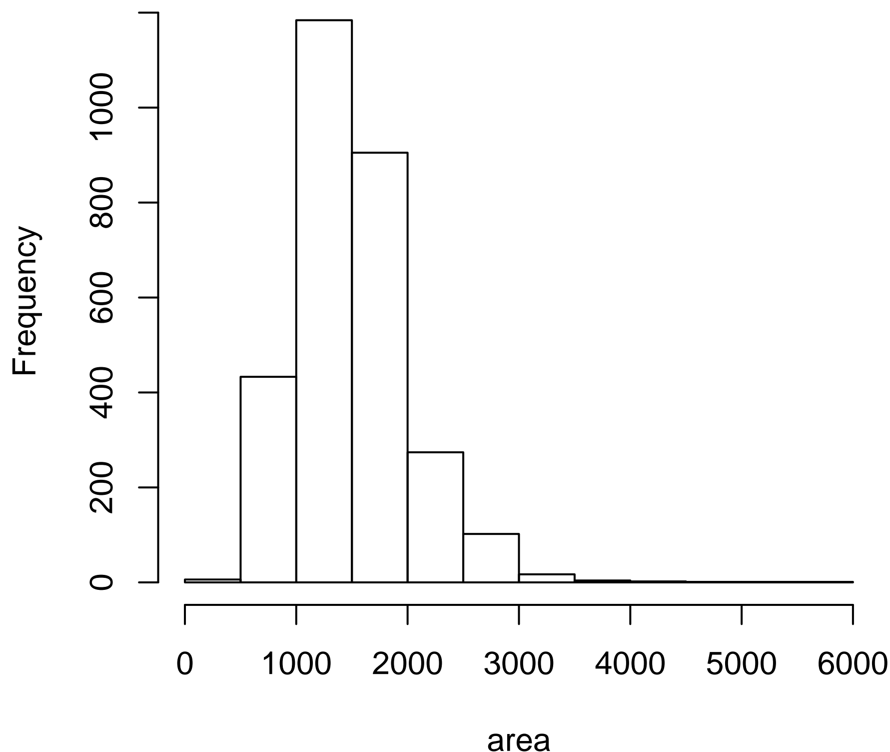
```r
area <- ames$Gr.Liv.Area
price <- ames$SalePrice
```

Let's look at the distribution of area in our population of home sales by calculating a few summary statistics and making a histogram.

```r
summary(area)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     334    1126    1442    1500    1743    5642
```

```r
hist(area)
```



The unknown sampling distribution

In this tutorial we have access to the entire population, but this is rarely the case in real life. Gathering information on an entire population is often extremely costly or impossible. Because of this, we often take a sample of the population and use that to understand the properties of the population.

If we were interested in estimating the mean living area in Ames based on a sample, we can use the following command to survey the population.

```
samp1 <- sample(area, 50)
```

This command collects a simple random sample of size 50 from the vector `area`, which is assigned to `samp1`. This is like going into the City Assessor's database and pulling up the files on 50 random home sales. Working with these 50 files would be considerably simpler than working with all 2930 home sales.

If we're interested in estimating the average living area in homes in Ames using the sample, our best single guess is the sample mean.

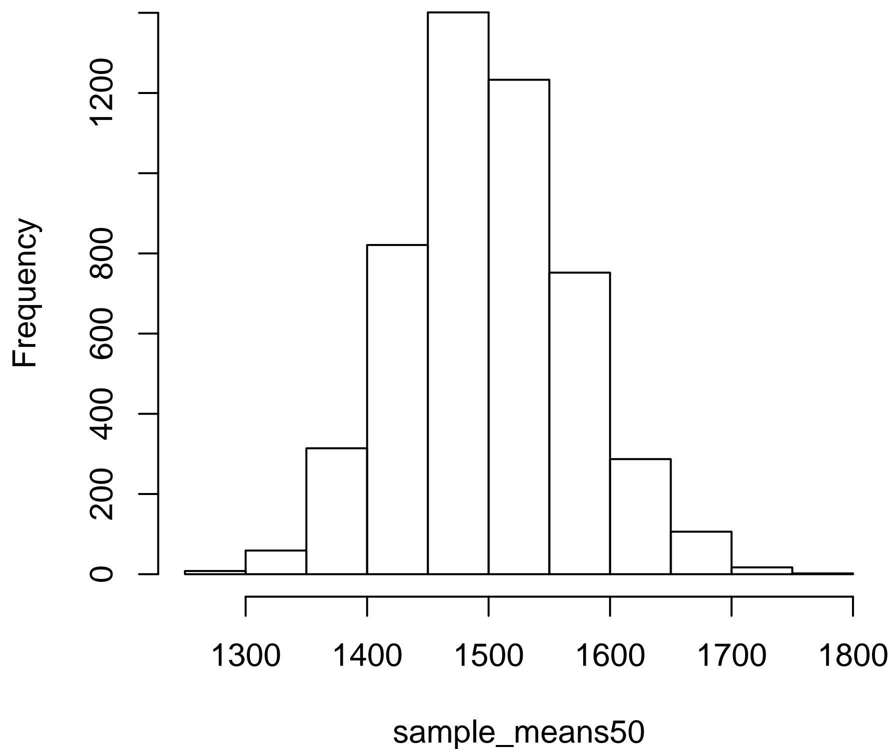
```
mean(samp1)
```

```
## [1] 1561.28
```

Depending on which 50 homes you selected, your estimate could be a bit above or a bit below the true population mean of 1499.69 square feet. In general, though, the sample mean turns out to be a pretty good estimate of the average living area, and we were able to get it by sampling less than 3% of the population.

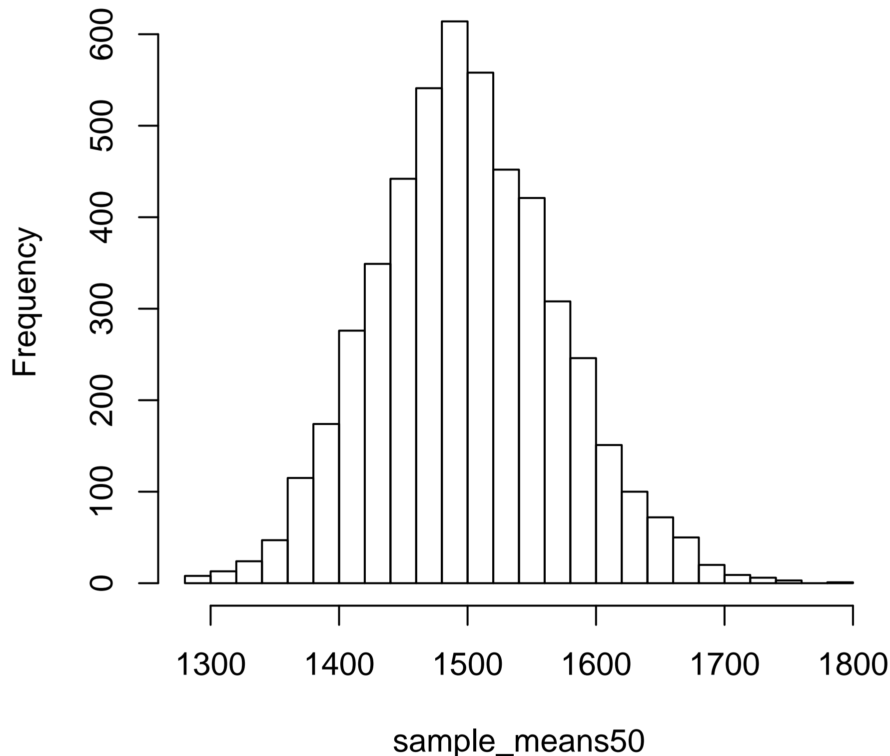
Not surprisingly, every time we take another random sample, we get a different sample mean. It's useful to get a sense of just how much variability we should expect when estimating the population mean this way. The distribution of sample means, called the *sampling distribution*, can help us understand this variability. In this tutorial, because we have access to the population, we can build up the sampling distribution for the sample mean by repeating the above steps many times. Here we will generate 5000 samples and compute the sample mean of each.

```
sample_means50 <- rep(NA, 5000)

for(i in 1:5000){
   samp <- sample(area, 50)
   sample_means50[i] <- mean(samp)
   }

hist(sample_means50)
```



If you would like to adjust the bin width of your histogram to show a little more detail, you can do so by changing the `breaks` argument.

```
hist(sample_means50, breaks = 25)
```



Here we use R to take 5000 samples of size 50 from the population, calculate the mean of each sample, and store each result in a vector called `sample_means50`. On the next page, we'll review how this set of code works.

Interlude: The `for` loop

Let's take a break from the statistics for a moment to let that last block of code sink in. You have just run your first `for` loop, a cornerstone of computer programming. The idea behind the for loop is *iteration*: it allows you to execute code as many times as you want without having to type out every iteration. In the case above, we wanted to iterate the two lines of code inside the curly braces that take a random sample of size 50 from `area` then save the mean of that sample into the `sample_means50` vector. Without the `for` loop, this would be painful:

```
sample_means50 <- rep(NA, 5000)

samp <- sample(area, 50)
sample_means50[1] <- mean(samp)

samp <- sample(area, 50)
sample_means50[2] <- mean(samp)

samp <- sample(area, 50)
sample_means50[3] <- mean(samp)

samp <- sample(area, 50)
sample_means50[4] <- mean(samp)
```

and so on…

With the for loop, these thousands of lines of code are compressed into a handful of lines. We've added one extra line to the code below, which prints the variable `i` during each iteration of the `for` loop. Run this code.

```
sample_means50 <- rep(NA, 5000)

for(i in 1:5000){
   samp <- sample(area, 50)
   sample_means50[i] <- mean(samp)
   print(i)
   }
```

Let's consider this code line by line to figure out what it does. In the first line we *initialized a vector*. In this case, we created a vector of 5000 zeros called `sample_means50`. This vector will will store values generated within the `for` loop.

The second line calls the `for` loop itself. The syntax can be loosely read as, "for every element `i` from 1 to 5000, run the following lines of code". You can think of `i` as the counter that keeps track of which loop you're on. Therefore, more precisely, the loop will run once when `i = 1`, then once when `i = 2`, and so on up to `i = 5000`.

The body of the `for` loop is the part inside the curly braces, and this set of code is run for each value of `i`. Here, on every loop, we take a random sample of size 50 from `area`, take its mean, and store it as the ith element of `sample_means50`.

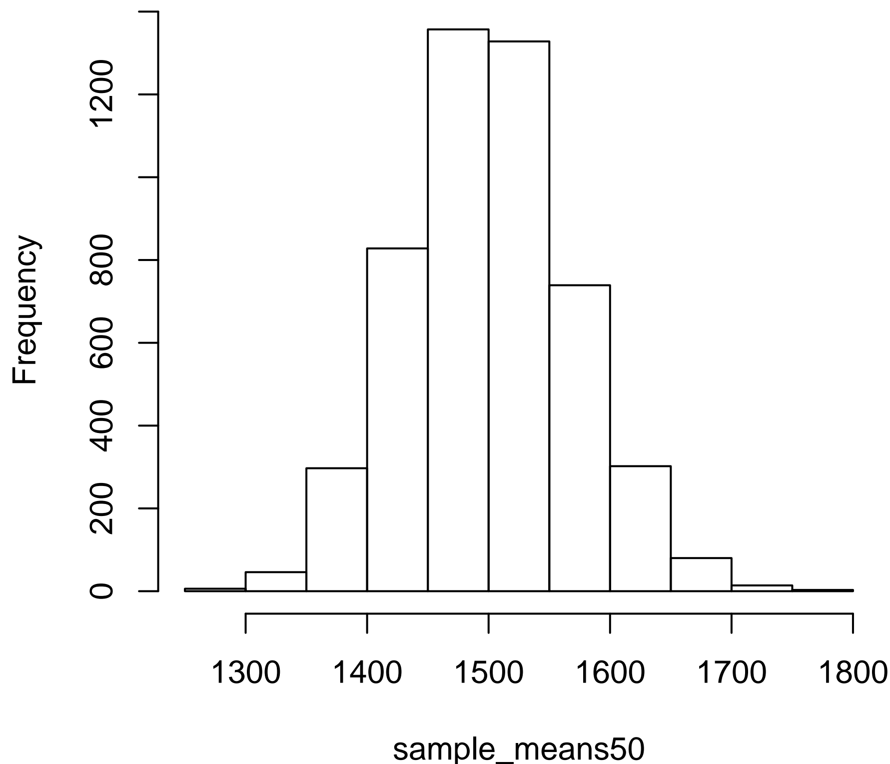
In order to display that this is really happening, we asked R to print `i` at each iteration. This line of code is optional and is only used for displaying what's going on while the `for` loop is running.

The `for` loop allows us to not just run the code 5000 times, but to neatly package the results, element by element, into the empty vector that we initialized at the outset.

Sample size and the sampling distribution

Mechanics aside, let's return to the reason we used a `for` loop: to compute a sampling distribution, specifically, this one.

```
hist(sample_means50)
```



The sampling distribution that we computed tells us much about estimating the average living area in homes in Ames. Because the sample mean is an unbiased estimator, the sampling distribution is centered at the true average living area of the the population, and the spread of the distribution indicates how much variability is induced by sampling only 50 home sales.

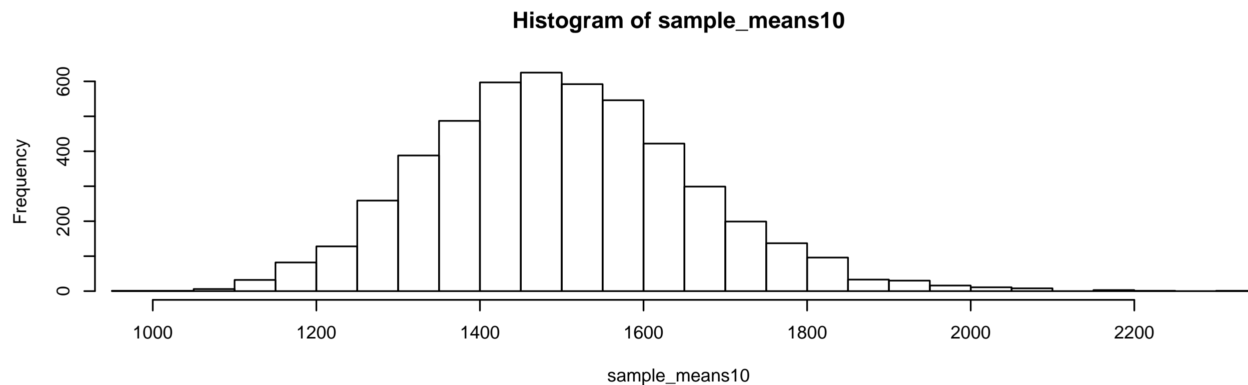
To get a sense of the effect that sample size has on our distribution, let's build up two more sampling distributions: one based on a sample size of 10 and another based on a sample size of 100.

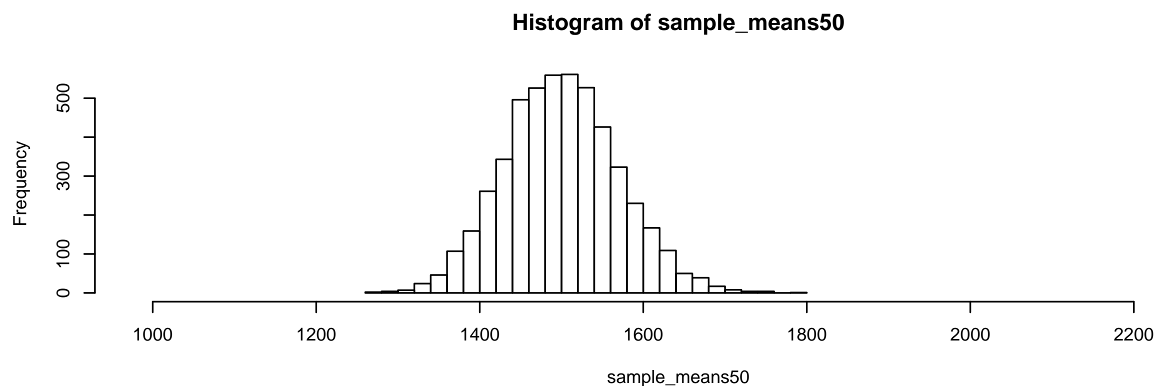
```
sample_means10 <- rep(NA, 5000)
sample_means100 <- rep(NA, 5000)

for(i in 1:5000){
  samp <- sample(area, 10)
  sample_means10[i] <- mean(samp)
  samp <- sample(area, 100)
  sample_means100[i] <- mean(samp)
}
```

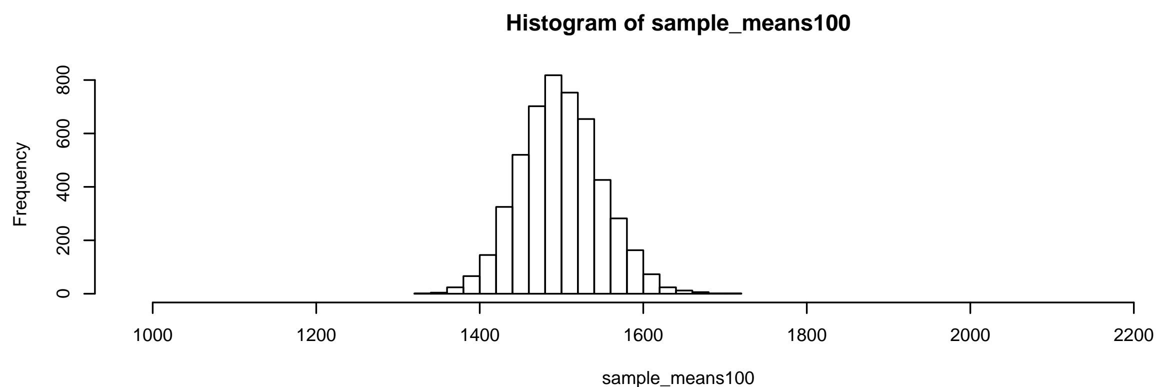
Here we're able to use a single `for` loop to build two distributions by adding additional lines inside the curly braces. Don't worry about the fact that `samp` is used for the name of two different objects. In the second command of the `for` loop, the mean of `samp` is saved to the relevant place in the vector `sample_means10`. With the mean saved, we're now free to overwrite the object `samp` with a new sample, this time of size 100. In general, anytime you create an object using a name that is already in use, the old object will get replaced with the new one.

To see the effect that different sample sizes have on the sampling distribution, plot the three distributions on top of one another.

```r
par(mfrow = c(3, 1))

xlimits <- range(sample_means10)

hist(sample_means10, breaks = 20, xlim = xlimits)
hist(sample_means50, breaks = 20, xlim = xlimits)
hist(sample_means100, breaks = 20, xlim = xlimits)
```








```r
par(mfrow = c(1, 1))
```

The first command specifies that you'd like to divide the plotting area into 3 rows and 1 column of plots (to return to the default setting of plotting one at a time, use `par(mfrow = c(1, 1))`). The `breaks` argument specifies the number of bins used in constructing the histogram. The `xlim` argument specifies the range of

the x-axis of the histogram, and by setting it equal to `xlimits` for each histogram, we ensure that all three histograms will be plotted with the same limits on the x-axis.

Sampling from Ames, Iowa

If you have access to data on an entire population, say the size of every house in Ames, Iowa, it's straight forward to answer questions like, "How big is the typical house in Ames?" and "How much variation is there in sizes of houses?". If you have access to only a sample of the population, as is often the case, the task becomes more complicated. What is your best guess for the typical size if you only know the sizes of several dozen houses? This sort of situation requires that you use your sample to make inference on what your population looks like.

The data

In the first part of this tutorial, we looked at the population data of houses from Ames, Iowa. Let's start by loading that data set anew.

```
ames <- read.csv("http://www.jkarreth.net/files/ames.csv")
```

In this second part of the tutorial, we'll start with a simple random sample of size 60 from the population. Specifically, this is a simple random sample of size 60. Note that the data set has information on many housing variables, but for the first portion of the tutorial we'll focus on the size of the house, represented by the variable `Gr.Liv.Area`.

```
population <- ames$Gr.Liv.Area
samp <- sample(population, 60)
```

Confidence intervals

One of the most common ways to describe the typical or central value of a distribution is to use the mean. In this case we can calculate the mean of the sample using,

```
sample_mean <- mean(samp)
```

Return for a moment to the question that first motivated this tutorial: based on this sample, what can we infer about the population? Based only on this single sample, the best estimate of the average living area of houses sold in Ames would be the sample mean, usually denoted as $\bar{x}$ (here we're calling it `sample_mean`). That serves as a good *point estimate* but it would be useful to also communicate how uncertain we are of that estimate. This can be captured by using a *confidence interval*.

We can calculate a 95% confidence interval for a sample mean by adding and subtracting 1.96 standard errors to the point estimate (See Section 4.2.3 of the textbook if you are unfamiliar with this formula).

```
se <- sd(samp) / sqrt(60)
lower <- sample_mean - 1.96 * se
upper <- sample_mean + 1.96 * se
c(lower, upper)
```

```
## [1] 1327.416 1498.451
```

This is an important inference that we've just made: even though we don't know what the full population looks like, we're 95% confident that the true average size of houses in Ames lies between the values *lower* and *upper*. There are a few conditions that must be met for this interval to be valid.

Confidence levels

In this case we have the luxury of knowing the true population mean since we have data on the entire population. This value can be calculated using the following command:

```
mean(population)
```

```
## [1] 1499.69
```

Each student in your class should have gotten a slightly different confidence interval. What proportion of those intervals would you expect to capture the true population mean? Why? If you are working in this tutorial in a classroom, collect data on the intervals created by other students in the class and calculate the proportion of intervals that capture the true population mean.

Using R, we're going to recreate many samples to learn more about how sample means and confidence intervals vary from one sample to another. *Loops* come in handy here (If you are unfamiliar with loops, review the first part of this tutorial).

Here is the rough outline:

- Obtain a random sample.
- Calculate and store the sample's mean and standard deviation.
- Repeat steps (1) and (2) 50 times.
- Use these stored statistics to calculate many confidence intervals.

But before we do all of this, we need to first create empty vectors where we can save the means and standard deviations that will be calculated from each sample. And while we're at it, let's also store the desired sample size as `n`. I choose n = 75 (75 draws from the 2930 homes in Iowa) but could pick any number below 2930 here.

```
samp_mean <- rep(NA, 50)
samp_sd <- rep(NA, 50)
n <- 75
```

Now we're ready for the loop where we calculate the means and standard deviations of 50 random samples. Note that I am setting a seed here to make sure that you obtain the exact same results when you work through this tutorial.

```
set.seed(123)
for(i in 1:50){
  samp <- sample(population, n) # obtain a sample of size n = 60 from the population
  samp_mean[i] <- mean(samp)     # save sample mean in ith element of samp_mean
  samp_sd[i] <- sd(samp)         # save sample sd in ith element of samp_sd
}
```

Lastly, we construct the confidence intervals.

```
lower_vector <- samp_mean - 1.96 * samp_sd / sqrt(n)
upper_vector <- samp_mean + 1.96 * samp_sd / sqrt(n)
```

Lower bounds of these 50 confidence intervals are stored in `lower_vector`, and the upper bounds are in `upper_vector`. Let's view the first interval.

```
c(lower_vector[1], upper_vector[1])
```

```
## [1] 1428.928 1684.646
```

Now, you can plot these 50 confidence intervals. The authors of the OpenIntro project provide a handy function for this. You can read it in from my website http://www.jkarreth.net/files/plot_ci.R using the `source()` function:

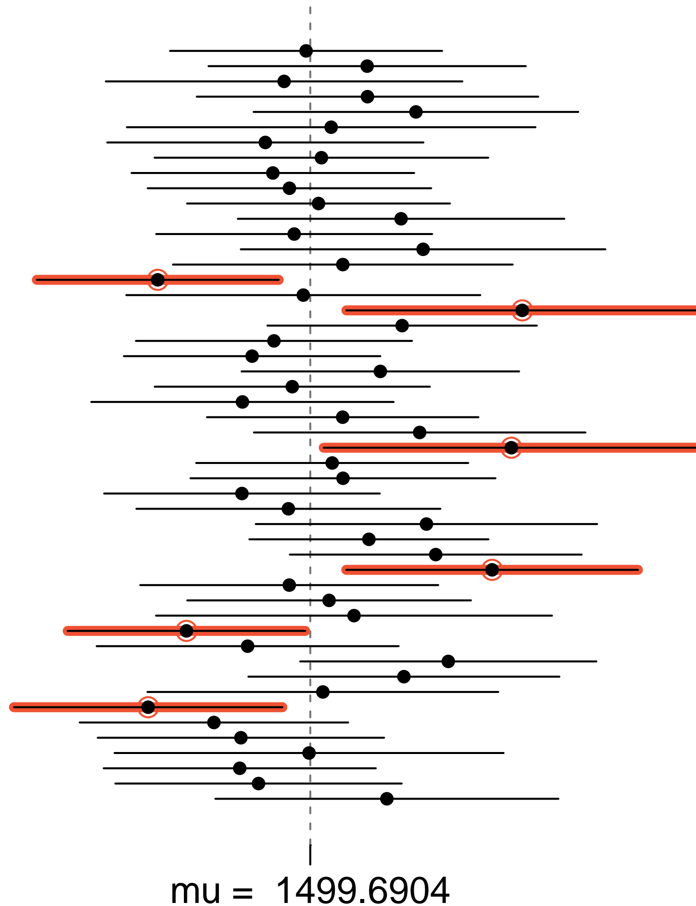
```
source("http://www.jkarreth.net/files/plot_ci.R")
```

Or you can copy and paste the functions below in your R script (avoid copying & pasting from the PDF version of this tutorial, as indentations might get lost):

```
contains <- function(lo,hi,m){
  if(m>= lo & m <= hi) return(TRUE)
  else return(FALSE)
}

plot_ci <- function(lo, hi, m){
  par(mar=c(2, 1, 1, 1), mgp=c(2.7, 0.7, 0))
  k <- 50
  ci.max <- max(rowSums(matrix(c(-1*lo,hi),ncol=2)))

  xR <- m + ci.max*c(-1, 1)
  yR <- c(0, 41*k/40)

  plot(xR, yR, type='n', xlab='', ylab='', axes=FALSE)
  abline(v=m, lty=2, col='#00000088')
  axis(1, at=m, paste("mu = ",round(m,4)), cex.axis=1.15)
  #axis(2)
  for(i in 1:k){
    x <- mean(c(hi[i],lo[i]))
    ci <- c(lo[i],hi[i])
   if(contains(lo[i],hi[i],m)==FALSE){
      col <- "#F05133"
        points(x, i, cex=1.4, col=col)
      #  points(x, i, pch=20, cex=1.2, col=col)
        lines(ci, rep(i, 2), col=col, lwd=5)
       }
     col <- 1
   points(x, i, pch=20, cex=1.2, col=col)
     lines(ci, rep(i, 2), col=col)
    }
  par(mfrow = c(1, 1))
}
```

Using this function yields the plot below. Note that it automatically highlights in red the confidence intervals that do not contain the population mean.

```
plot_ci(lower_vector, upper_vector, mean(population))
```



[**Advanced content**] If you wanted to plot confidence intervals yourself using ggplot2, you can use the following code without having to use the above function.

First, create a data frame with the sample means and lower and upper ends of the confidence intervals.

```
library(ggplot2)
confint.dat <- data.frame(samp_mean, lower_vector, upper_vector,
  i = 1:length(samp_mean))
```

Next, create a variable that indicates whether the confidence interval covers the mean. This variable is only created for coloring the confidence intervals in the resulting plot. It uses the `ifelse` function, which is very handy for creating variables based on conditions. Its basic logic:

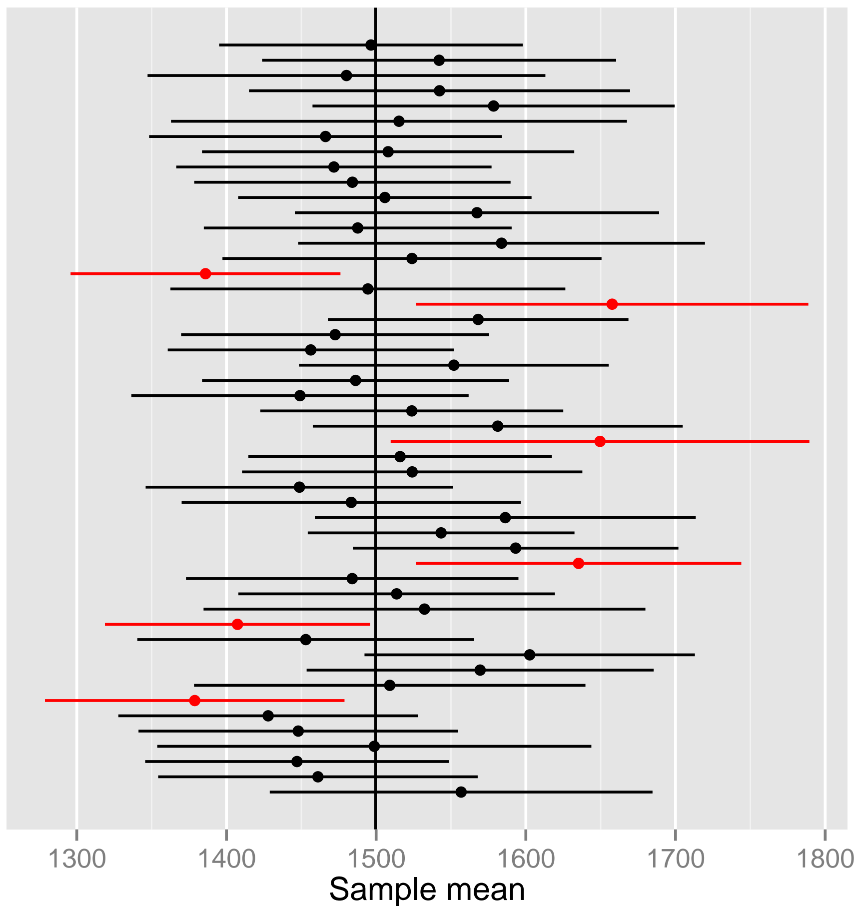
```
newvariable <- ifelse([condition on oldvariable],
  [value of newvar if condition is met],
  [value of newvar if condition is not met])
```

See `?ifelse` or http://www.ats.ucla.edu/stat/r/library/intro_function.htm for more information.

```
confint.dat$excl0 <- ifelse(confint.dat$upper_vector < mean(population) |
  mean(population) < confint.dat$lower_vector, 1, 0)
```

Lastly, we construct the plot through a series of `ggplot` commands:

```
p <- ggplot(data = confint.dat, aes(x = i, y = samp_mean, ymin = lower_vector,
      ymax = upper_vector, colour = factor(excl0))) + geom_pointrange() +
      scale_x_continuous(breaks = NULL) + geom_hline(yintercept = mean(population)) +
      scale_colour_manual(values = c("black", "red")) + guides(colour = FALSE) +
      ylab("Sample mean") + xlab("") + coord_flip()
p
```



This tutorial is based on a product of OpenIntro that was released under a Creative Commons Attribution-ShareAlike 3.0 Unported license. The original tutorial was adapted for OpenIntro by Andrew Bray and Mine Cetinkaya-Rundel from a tutorial written by Mark Hansen of UCLA Statistics. It was modified by Johannes Karreth for use in RPOS/RPAD 517 at the University at Albany, State University of New York.