

## Week 1: Basic skills and estimation and confidence

*You should read Chapter 1 of Bolker's book, or a similar quick start guide to R, to accompany this week's exercises and lectures. The beginning of Bolker's Chapter 2 is also helpful. Interacting with R is a lot like programming, and if you have never programmed, much of what I will have to assume you know will be unknown. Ask me, if anything is unclear, and we'll sort out the confusion.*

### Exercise 1: Collections and data frames

Data is entered into R as “collections.” These “collections” are vectors of numbers or symbols that represent a single column of data. To make a collection, enter:

```
x <- c( 1 , 2 , 3 , 4 )
```

on the R command prompt and press ENTER. Now simply enter `x` on the command prompt and press ENTER, to see the values you just entered. You can now use the symbol `x` anywhere within R, to represent the collection of values. Using a different symbol makes a different collection that does not replace `x`:

```
y <- c( 4,3,2,1 )
```

Type `y` at the command prompt and press ENTER again, to see that the order of the values is preserved, so `x` and `y` now contain different lists of numbers.

(a) Execute the code below and explain the output.

```
c( x , x )
```

(b) Execute the code below. What happened, in both cases?

```
sort(x)
sort(x , decreasing=TRUE )
```

(c) You can use indexes to extract specific values from a collection. Execute the code below and explain the output.

```
y[1]
y[c(1,2)]
x[x<3]
y[y==2]
```

(d) A “data frame” is a table-shaped bundling of collections. That is, each collection is like a column of data. Put several of these together, and you have a set of data, which is called a “data frame” in R. To make a data frame named `d`, just use the code below:

```
d <- data.frame( a.column.name=x , another.column.name=y )
```

Then enter the symbol `d` at the prompt to see the table formatted data. To extract each column, use the code below.

```
d$a.column.name  
d$another.column.name
```

Indexing data frames is a bit more complicated than indexing collections, so we'll leave that for another time. The same tricks work, but since data frames are two dimensional, you have to be careful.

## Exercise 2: Plotting your data

R has some very powerful plotting functions that make very professional graphs. The trouble is that you have to execute code to produce these graphs, so it can seem intimidating. Here are some examples, with some modifications for you to try.

(a) We'll sample heights and weights from the class for this exercise. So first enter the heights and weights into a data frame named `exercise2data`, with columns named `height` and `weight`.

(b) Make a simple histogram of the heights:

```
hist( exercise2data$height , xlab="height" , main="Distribution of heights" )
```

Now make a histogram of weights, as well, by modifying the code above.

(c) Scatter plots are just as easy. The code below will produce a plot of height against weight.

```
plot( exercise2data$height ~ exercise2data$weight , xlab="weight" , ylab="height" )
```

Modify the code above to reverse which variable is on which axis.

(d) A very useful function in R is `curve()`, which plots a mathematical expression across a range of parameter values. The basic syntax is:

```
curve( your.function(x) , from=minimum.x.value , to=maximum.x.value )
```

Wherever the symbol `x` appears in the mathematical expression, R will replace it with the range of values you specify. For example:

```
curve( x^2 , from=0 , to=10 )  
curve( dnorm( x ) , from=-3 , to=3 )
```

We will use `curve()` over and over again, to visualize model predictions, as a function of specific parameters. You can make `curve()` superimpose a new function over a previously plotted one by adding `add=TRUE` to the function call. Execute both of the lines below to get two normal curves with different variances on the same plot:

```
curve( dnorm( x ) , from=-5 , to=5 )  
curve( dnorm( x , sd=2 ) , from=-5 , to=5 , add=TRUE )
```

## Exercise 3: Estimation

Now you'll use the class's height and weight data to explore how sample size affects estimates.

(a) Using the `exercise2data` data frame from Exercise 2, compute the mean height and mean weight in the population. Use the function `mean()` to do this.

(b) The code below will sample 3 heights from our population and then plot the likelihood function for the mean, given that sample. That is, we choose 3 heights at random. This is a new sample. We want to estimate the population mean, which you computed in (a). So we plot the likelihood of different hypothetical means, given our sample. The red line in the plot is the sample mean, and you'll see that it coincides with the high point of likelihood. What principle of likelihood does this illustrate?

```
n <- 3
a.sample <- sample( exercise2data$height , size=n )
likelihood.normal <- function( x ) sapply(x , function(z)
  prod( dnorm( a.sample , mean=z , sd=sd(a.sample) ) ))
curve( likelihood.normal(x) , from=60 , to=75 )
lines( c( mean(a.sample) , mean(a.sample) ) , c(0,1) , col="red" )
points( a.sample , rep(0,n) )
```

Repeat the code above and generate a number of samples. Notice how the sample mean varies with each sampling exercise.

(c) Now increase the sample size from (b) to  $n = 15$ . How does this change our estimates and the shape of the likelihood function? Why?

(d) The estimates of the population mean that you generated in (b) and (c) themselves have distributions. That is, there is a theoretical distribution from which we randomly draw each sample mean. This is called the mean's *sampling distribution*. I want you to simulate this sampling distribution, in order to intuit an important fact about all such sampling distributions.

Run the code below. It generates 10,000 samples and computes their means. It then plots these 10,000 sample means in a histogram. It also draws a red line where the true population mean lies.

```
n <- 10
sample.means <- sapply( rep(n,10000) , function(x)
  mean( sample( exercise2data$height , size=x ) ) )
hist( sample.means , breaks=50 )
lines( c( mean(exercise2data$height) , mean(exercise2data$height) ) , c(0,5000) ,
  col="red" )
```

Do you think this distribution looks like any particular type of distribution? Where is the true population mean located in this distribution? What happens when you change the sample size from 10 to a larger or smaller value?

### Exercise 3: Confidence

All estimates need a measure of confidence to accompany them. So now you'll explore confidence estimates for the height data. Some of the code I'll provide for you here is complicated (at least it may look so now), but don't worry—you just to be able to run it in R and answer questions from the output it produces.

(a) The code below will sample 100 different samples of 5 data points each from our population of heights. It will then compute the 95% confidence interval for each sample, saving the upper and lower bounds in `sample.ci.hi` and `sample.ci.lo`, respectively. The confidence interval in this case is defined as *the range of values we expect the true mean of the population to lie within, 95% of the time*. Each sample suggests a different interval, but we expect 95% of these intervals to contain the true mean.

The code below will plot the sample means for you, each on a different row, as well as horizontal lines for the confidence intervals. The vertical red line it shows will be the true population mean. Confidence intervals in red do not contain the true mean.

```
# sample
sample.means <- {}
sample.ci.hi <- {}
sample.ci.lo <- {}
n <- 100
for ( i in 1:n ) {
  s <- sample( exercise2data$h , size=5 )
  sample.means[i] <- mean(s)
  sample.ci <- confint.default( lm( s ~ 1 ) , level=0.95 )
  sample.ci.hi[i] <- sample.ci[2]
  sample.ci.lo[i] <- sample.ci[1]
}
# count instances of true mean being inside intervals
is.inside <- function( true.mean , lo , hi ) {
  ifelse( true.mean>lo & true.mean<hi , TRUE , FALSE )
}
count.inside <- 0
for ( i in 1:n ) {
  if ( is.inside( mean(h) , sample.ci.lo[i] , sample.ci.hi[i] ) ) {
    count.inside <- count.inside + 1
  }
}
p.inside <- count.inside / n
# plot sample means and confidence intervals
plot( 1:n ~ sample.means , xlim=c( min( sample.ci.lo ) , max( sample.ci.hi ) ) ,
      ylab="sample number" , xlab="sample mean" ,
      main=paste(p.inside,"of confidence intervals contain true mean") )
for ( i in 1:n ) {
  lines( c( sample.ci.hi[i] , sample.ci.lo[i] ) , c( i , i ) ,
        col=ifelse( is.inside( mean(h) , sample.ci.lo[i] , sample.ci.hi[i] ) ,
                    "black" , "red" ) )
}
# plot true mean
lines( c( mean(h) , mean(h) ) , c(1,n) , col="red" )
```

Execute this code several times and get a feel for the parts of the graph. Do you see how the

intent of the confidence interval is to quantify our uncertainty about estimates? Can you explain the definition of the confidence interval in your own words?

**(b)** Now modify the code above to increase the number of heights in each sample to 10. What happens to the confidence intervals? Now reduce the sample size to 3. Can you summarize how sample size affects the quality of the confidence intervals and our uncertainty in the estimate?

**(c)** Finally, modify the code above to compute 80% confidence intervals, rather than 95% intervals. Again vary the sample sizes from 3 to 5 to 10. How have the intervals changed? What about 50% confidence intervals? Can you connect the definition of the confidence interval to how its width responds to changes in the specified percent coverage (50% versus 80% versus 95%)?