Introduction to R

Shirley Liao

6/13/2017

What is R?

R is a programming language designed for statistical computing. Notable characteristics include:

- Vast capabilities, wide range of statistical and graphical techniques
- Very popular in academia, growing popularity in business
- Written primarily by statisticians
- FREE (no cost, open source)
- Excellent community support: mailing list, blogs, tutorials
- Easy to extend by writing new functions

Previous coding experience?

- Stata: http://www.princeton.edu/~otorres/RStata.pdf
- SAS/SPSS: http://www.et.bs.ehu.es/~etptupaf/pub/R/RforSAS&SPSSusers.pdf
- matlab: http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf
- Python: http://mathesaurus.sourceforge.net/matlab-python-xref.pdf

R Studio orientation

Notes:

What can R do?

OK, it's free and popular, but what makes R worth learning? In a word, "packages". If you have a data manipulation, analysis or visualization task, chances are good that there is an R package for that. Lets install some packages and look at some examples.

R can predict the world population in 2020

```r
#install packages and call on them
install.packages(c("forecast", "plotly"),repos="https://cloud.r-project.org")
library(forecast)
library(plotly)

#input historical data
## from https://esa.un.org/unpd/wpp/Download/Standard/Population/
worldpop <- structure(c(2.525149312, 2.571867515, 2.617940399, 2.66402901,
                        2.710677773, 2.758314525, 2.807246148, 2.85766291, 2.909651396,
                        2.963216053, 3.018343828, 3.075073173, 3.133554362, 3.194075347,
                        3.256988501, 3.322495121, 3.390685523, 3.461343172, 3.533966901,
                        3.607865513, 3.682487691, 3.757734668, 3.833594894, 3.90972212,
                        3.985733775, 4.061399228, 4.13654207, 4.211322427, 4.286282447,
                        4.362189531, 4.439632465, 4.518602042, 4.599003374, 4.681210508,
                        4.765657562, 4.852540569, 4.942056118, 5.033804944, 5.126632694,
                        5.218978019, 5.309667699, 5.398328753, 5.485115276, 5.57004538,
                        5.653315893, 5.735123084, 5.815392305, 5.894155105, 5.971882825,
                        6.049205203, 6.126622121, 6.204310739, 6.282301767, 6.360764684,
                        6.439842408, 6.51963585, 6.600220247, 6.68160732, 6.763732879,
                        6.846479521, 6.92972504300001, 7.013427052, 7.097500453, 7.181715139,
                        7.265785946, 7.349472099), .Tsp = c(1950, 2015, 1), class = "ts")

# Plot Projected numbers (in billions) of humans living on earth
fit <- auto.arima(worldpop)
ggplotly(autoplot(forecast(fit)))
```

Try hovering over the graph with your mouse!

R can allow us to create beautiful 3D plots

```r
install.packages("plot3D",repos="https://cloud.r-project.org")
libary(plot3D)
example(surf3D)
```

Keep hitting return until you’ve gone through all the plots.

(My favorite) R can pull up XKCD comics by number

```r
install.packages("RXKCD",repos="https://cloud.r-project.org")
library(RXKCD)
getXKCD(539)
```

As you can see, you don’t need to be an expert coder in order to do some pretty cool things in R. The open-source, academic nature of this code allows you to build off the work of statisticians around the world. However, the downside of this is that there is no overarching entity performing quality control for these packages. Packages may be “abandoned” (not updated to keep up with updates in R), “broken” (are updated in such a way which breaks functionality), written in a poorly-functioning or ineffective way or extremely complicated to use.

Exercise 0:

1. Add 3932 and 458
2. Multiply 38 and -26
3. Find the square root of 23
4. Create a variable called 'x' and assign it a value of 2
5. Make R print "Hello World!" using the print function.
6. Install the package "cars"

If you have any questions, try asking R for help:

```
?mean #help on a package or function
??addition #help search using a key name
```

Google is also a valuable resource!

Lesson 1: Using packages & functions

The majority of R's capabilities involve using functions. These are commands which take input (data or specifications) and outputs answers. Here is an example of a simple function which only requires the input of a single number and outputs the absolute value of that number:

```
abs(1)
```

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

```
abs(-1)
```

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

There are functions which take inputs by way of multiple numbers (in this case, a numerical vector):

```
max(c(3,4,2,3,4,5,6,3))
```

```
## [1] 6
```

More complicated functions may require multiple inputs. To keep them straight, it is recommended to name inputs after the first. The quantile function calculates the quantiles of a vector of data, and we specify which quantiles we want by inputting a parameter called 'p':

```
quantile(c(3,4,4,3,5,7,23,5,45,75),p=0.5)
```

```
## 50%
##   5
```

Thus 5 is the number which is at the 50% quantile of our inputted set of numbers. If we do not specify a *p*, the quantile function has default values for *p*.

```
quantile(c(3,4,4,3,5,7,23,5,45,75))
```

```
##   0%  25%  50%  75% 100%
##    3    4    5   19   75
```

We can always examine the default values for inputs by querying the function:

```
?quantile
```

How many other default inputs does the quantile function contain?

R comes with many pre-installed functions, but we can extend R's capabilities by installing packages using the install.packages() function. The first and only required input is the name of the package:

```
install.packages("car",repos="https://cloud.r-project.org")
```

```
## Installing package into '/Users/shirleyliao/Library/R/3.3/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/96/7xw011zd5vs0f_80xf76pwrr0000gn/T//RtmpSvGcHr/downloaded_packages
```

Other function specifications include lib (location on computer you would like the package to be installed) and repos (where you would like to download the package from). Without a specification, R will create a default folder ('library') at the same source where R is downloaded and keep the package there.

Installing a package only puts a copy of it in your computer. Think of it as a library buying a copy of a book versus someone going to check it out. In order to use the package, you must call it using the library() function (this is also called "attaching" a package):

```
library(car)
```

You must call the package again every time you exit R.

Exercise 1: Using packages & functions

1. Install the package "fun" and attach it
2. Examine the random_password() function. What necessary inputs does it require? What optional inputs does it include and what are their defaults?
3. Generate a random password that is 8 characters long, with only with alphanumeric symbols and with no repeating letter/numbers

Lesson 2: Arithmetic & Assignment

R has the capabilities of a calculator: - '+', '-', '*'and'/'are symbols used for addition, subtraction, multiplication and division, respectively. These are called "operations". - sum() and prod() are used with vectors of numbers (if you want to use these with subtraction/division input negative numbers/fractions). - The functions sqrt(), exp(), abs() and log() square root, exponentiate (base e), finds the absolute value of and log numbers, respectively (can search for other functions which you need) -'^' is used to exponentiate

In R, we can save numbers as variables by using '=' or '<-'. For example:

```
x = 3
y <-3
```

We can check what number a variable represents by typing the variable into R:

```
x
```

```
## [1] 3
```

```
y
```

```
## [1] 3
```

We can also perform arithmetic functions using these variables:

```
x+y
```

```
## [1] 6
```

```
x-y
```

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

```
x*y
```

```
## [1] 9
```

```
x/y
```

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

Performing operations using assigned variables does NOT change the value of the variable itself unless you re-assign it.

Exercise 2: Arithmetic & Assignment

1. Assign the value of 2389 to 'x'
2. Multiply 'x' by 3 in a way which does not change the value of the variable 'x'
3. Multiply 'x' by 3 in a way which does change the value of the variable 'x'
4. Assign y the value of 'x' divided by 11
5. Subtract the square root of 'y' from the log-base-2 of 'x'
6. Find the sum of all numbers from 1 to x

Lesson 3: Data structures & types

Data types

In R, data exists in three "types": 1. Numerical (integer/double are technically two different types, but you don't have to worry about converting between them) 2. Character ("words or phrases", always written with quotations) 3. Boolean (TRUE/FALSE, no quotations)

A variable can be any of these data types. One advantage of R over other programming languages is that you do not have to declare a data type for your variable. R will automatically assign a data type after you define your variable:

```
a = 3
b = "letter"
c = TRUE
```

We can check which type of data a variable is using the typeof() function:

```
typeof(a)
```

```
## [1] "double"
```

```
typeof(b)
```

```
## [1] "character"
```

```r
typeof(c)
```

```
## [1] "logical"
```

You can convert between numeric and character variables using as.numeric() and as.character functions:

```r
d = 9
typeof(d)
```

```
## [1] "double"
```

```r
e = as.character(d)
typeof(e)
```

```
## [1] "character"
```

```r
e
```

```
## [1] "9"
```

Data structures

R is a “vectorized” language. That means that the simplest “building block” of data is a vector (in contrast, SAS and STATA are table-based and Matlab is matrix-based). A vector is defined as a collection of numbers/characters/booleans expressed in a row. Vectors may only contain one type of data.

```r
x <- c(10, 11, 12)
y <- c("10", "11", "12")
z <- c(TRUE, FALSE, TRUE, TRUE)

typeof(x)
```

```
## [1] "double"
```

```r
typeof(y)
```

```
## [1] "character"
```

```r
typeof(z)
```

```
## [1] "logical"
```

Data points in vectors are indexed and can be retrieved if we know which index to call:

```r
x[2]
```

```
## [1] 11
```

If we wish to retrieve something specific from a vector but do not know its index, we can submit the vector to a “logical test” using the which() function.

```r
which(y == "10")
```

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

```r
which(x < 12)
```

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

```r
which(z=="TRUE")
```

```
## [1] 1 3 4
```

Note that a single '=' is used for assignment while '==' is used to test for equality.

The most important point about vectors is that they can be used in any function/operation that single data points can. Try inputting vector 'x' into the abs() function and then into the max() function. What are the outputs of each? Why does one output an array and the other output a single number?

The second data structure you will be using the most often in this class is called a data frame. Data frames are created when vectors of data are inputted as columns. Data tables require mandatory column labels and optional row names. Unlike vectors, data frames may contain different data types (but must be the same type within column).

```
w = data.frame(x,y)
w
```

```
##    x  y
## 1 10 10
## 2 11 11
## 3 12 12
```

Data frames can only be created from vectors which all have the same length (which you can check using the length() function). Columns of a data frame can be extracted (as a vector) using the name of the column, and individual points can be extracted with a row and column number ([row,column]):

```
w$x
```

```
## [1] 10 11 12
```

```
w[2,1]
```

```
## [1] 11
```

Once columns of a data frame are extracted, they can be manipulated and (optionally) reassigned to the data frame:

```
w$x = w$x + 3
w
```

```
##    x  y
## 1 13 10
## 2 14 11
## 3 15 12
```

We can also add new columns to the data frame using the $ operator:

```
w$newcol = c(3,3,5)
w
```

```
##    x  y newcol
## 1 13 10      3
## 2 14 11      3
## 3 15 12      5
```

Data frames are commonly organized so that individual-level data is inputted by row under various variables, represented by columns. Let's look at the "sleep" dataset, which describes a study of two different drugs ("group") and how they impact the sleep ("extra", in hours) of 10 individuals ("ID") compared against control/no drugs. Each individual is given both drugs.

```
data(sleep)
sleep
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
```

If we wish to extract the "extra" column only for drug 1, we may use the code:

```
sleep$extra[sleep$group==1]
```

```
##  [1]  0.7 -1.6 -0.2 -1.2 -0.1  3.4  3.7  0.8  0.0  2.0
```

Which tells R that we want the values of the column "extra" which correspond to individuals in group 1.

Matricies are important structures for certain mathematical computations (i.e. linear algebra). Their rows/columns are unlabeled and they can be thought of as three dimentional vectors. Matricies can be created with the following command, where the data we wish to input in a matrix is the first argument and row/column number specifications follow:

```
matrix(sleep$extra,nrow=10,ncol=2)
```

```
##       [,1] [,2]
##  [1,]  0.7  1.9
##  [2,] -1.6  0.8
##  [3,] -0.2  1.1
##  [4,] -1.2  0.1
##  [5,] -0.1 -0.1
##  [6,]  3.4  4.4
##  [7,]  3.7  5.5
##  [8,]  0.8  1.6
##  [9,]  0.0  4.6
## [10,]  2.0  3.4
```

We can also make a matrix by binding two vectors together:

```r
rbind(x,y) #binds vectors as rows
```

```
##   [,1] [,2] [,3]
## x "10" "11" "12"
## y "10" "11" "12"
```

```r
cbind(x,y) #binds vectors as columns
```

```
##      x    y
## [1,] "10" "10"
## [2,] "11" "11"
## [3,] "12" "12"
```

Note that matrices, like vectors, may only be one data type. Binding two vectors together of different data types will coerce the entire matrix to the most "complex" type (booleans will be coerced to numeric, numeric will be coerced to character). Individual data points can be extracted using the [row, column] notation, but we can no longer extract columns via the $ operator. Matrices may use many of the functions which are available to vectors, and there are also a host of special functions and operations for matrices (determinants, matrix multiplication etc.).

Lists and arrays are other data structures available in R. They are used less often because R is ineffcient at storing them. Statisticians who work with large data sets in complicated structures often turn to Python.

Filling data structures

Somtimes you want to create an 'empty' data structure and fill it in later. In this case, we use placeholders called "NA", which stands for "not any":

```r
matrix(NA,nrow=2,ncol=2)
```

```
##      [,1] [,2]
## [1,]   NA   NA
## [2,]   NA   NA
```

We can create repetitions of NAs (or any number) to fill a vector:

```r
rep(NA,9)
```

```
## [1] NA NA NA NA NA NA NA NA NA
```

```r
rep(3,9)
```

```
## [1] 3 3 3 3 3 3 3 3 3
```

Another handy function is seq(), which outputs a sequence of numbers, allowing you to choose the increment:

```r
seq(1,4)
```

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

```r
seq(1,4,by=0.4)
```

```
## [1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8
```

Exercise 3: Data structures & types

1. Create 'a', a vector which contains the numbers 1 through 10 and 'b', a vector which contains 10 repetitions of 3.

2. Multiply ‘a’ and ‘b’ together, assigning this to a new variable ‘c’. How does R multiply vectors?
3. Add 8 to every number in ‘a’. Come up with two ways to do this.
4. Divide the 3rd number in ‘c’ by the 2nd number in ‘b’.
5. Create a 3 (rows) by 10 (columns) matrix called ‘d’ which is filled with vectors ‘a’, ‘b’ and ‘c’, in that order. Then create a data frame called ‘e’ with vectors ‘a’, ‘b’ and ‘c’ as columns.
6. What does d[3,3:7] and d[,3] extract?
7. Attempt to use the max() function on d and e. Does it work on both? Why/why not?
8. Input the vector seq(1,8) in a 2 by 4 matrix. In what order does R input the numbers in the matrix?

Lesson 4: Loading data

R can read in data files of various formats, including .csv, .xls etc. The R-specific format for data is .Rdata or .R (also used for R text files). We can load in data via the R console (with code) or through the RStudio environment.

Loading a CSV file via console: First, download the CSV file about esophageal cancer (esoph.csv) onto your desktop.

```
esoph = read.csv("~/Desktop/esoph.csv")
```

Loading a CSV file via environment: 1. download the CSV file about esophageal cancer (esoph.csv) onto your desktop 2. In the “environment” panel click Import Dataset 3. In the dropdown menu select From CSV 4. Using Browse, manuver to your desktop and choose esoph.csv 5. Make sure the data format looks correct (spacing should be “comma”) and load.

What data structure is esoph in?

Loading a .R file via console: First, download the .R file dat2.R onto your desktop.

```
load("~/Desktop/dat2.R")
```

Loading a .R file via environment: 1. Download the .R file dat2.R onto your desktop 2. In the environment panel, click the “open file” button 3. Maneuver to the file location of dat2.R 4. Choose it and open it

What data structure is dat2?

There are also many toy data sets already built into R if you would like to play around with them. You can see a list by typing this command (opens in new window):

```
library(help="datasets")
```

To load these datasets you simply use the function data():

```
data(sleep)
sleep
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
```

Exercise 4: Loading data

Load the babyNames.csv file onto your workspace.

Lesson 5: Selecting and manipulating data frames

Once we have read in a complicated data frame, we may wish to examine it. We can extract the names of a data frame using the names() function:

```
names(babyNames)
```

Perhaps we would like to know the number of rows and columns there are:

```
dim(babyNames)
```

And see a sample of the data:

```
head(babyNames)
```

Upon closer examination, we may see that Sex is a column comprised of FALSEs (F) and NAs (M). Perhaps something has gone wrong in the import. Let us create a new column: a numerical vector where "girl" names are assigned a 0 and "boy" names are assigned a 1. We will title this column "BoyNames" to avoid confusion. Try this yourself first:

```
babyNames$BoyNames = 0
babyNames$BoyNames[is.na(babyNames$Sex)] = 1
table(babyNames$BoyNames)
```

Say we wish to create a new data frame which only has information regarding girl names. We may extract column data one by one as we did in the previous section, but the subset() function is much faster and retains our data frame:

```
girlNames = subset(babyNames,BoyNames==0)
```

Remember that columns are not, themselves, data structures and R will not recognize them. Columns must always be preficed with the data frame they belong to:

```
BoyNames
```

We can also order the data according to year, ascending or descending:

```
babyNames[order(babyNames$Year),]
babyNames[order(-babyNames$Year),]
```

Exercise 5: Selecting and manipulating data frames

1. Examine the slice(), select() and filter() functions. Use slice() to extract rows 1:100 from babyNames, then perform it using indexing brackets. Use filter() to extract all data from "England and Wales", do the same using subset() and and finally by using indexing brackets. What are the advantages/disadvantages of each method?
2. Change the column "Percent" into decimal notation (Ex: 2.53 becomes 0.0253) and round to two spaces behind the decimal. Optional: attempt to do this using the within() function.
3. Remove the "Sex" column from the data frame.
4. Examine the location you were born, the date of your birth and your name. Can you find a row which corresponds to your name/date/location? (Note: this data set only has information 1996 to 2015, if you were born before 1996 you may use your discretion) If yes, how relatively popular was your name in the year you were born (in terms of 1st being the most popular name)? If not, insert a new data point (row) to babyNames: Location = (Country or state you were born), Year = (your DOB), Name = (your name), Count = 1, Percent = NA, Name.length = (number of letters in your name), then determine the most popular name in the year you were born.
5. Examine the apply() function and use it to find the mean and standard deviation of all numerical columns of babyNames.

Lesson 6: Basic statistical functions

Most summary statistics you could want can be found as built-in functions in R:

```
x = c(3,5,3,32,4,5,56,23,6,2)
mean(x)
```

```
## [1] 13.9
```

```
sd(x)
```

```
## [1] 17.89134
```

```
median(x)
```

```
## [1] 5
```

```r
var(x)
```

```
## [1] 320.1
```

Empirical quantiles can be found using the quantile() function:

```r
quantile(x)
```

```
##    0%   25%   50%   75%  100%
##  2.00  3.25  5.00 18.75 56.00
```

```r
quantile(x,p=c(0.025,0.975))
```

```
##   2.5%  97.5%
##  2.225 50.600
```

summary() is also a useful function that will print many summary statistics about the data at once:

```r
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.25    5.00   13.90   18.75   56.00
```

You can use the table() function in order to tabulate categorical data:

```r
r = c("a","a","b","c","b","a","b")
s = c("x","y","x","x","y","z","z")
table(r,s)
```

```
##    s
## r   x y z
##   a 1 1 1
##   b 1 1 1
##   c 1 0 0
```

For tables of more than two dimentions, look into the ftables() function.

Exercise 6: Basic statistical functions

1. Load the InsectSprays data set (counts of insects found in agricultural plots which were treated with different sprays) from base R. Find the mean, sd, empirical 95% interval (made with the quantiles() function) and asymptotic 95% interval (made assuming the data distribution is normal) of the "count". Do the empirical and asymptotic intervals match?
2. Find the mean, sd and asymptotic 95% interval of "count" for each spray separately. Do you notice a difference between sprays?
3. Use summary() on your InsectSprays data table. Why is the summary for "count" of a different format than the summary for "spray"?
4. Summarize the InsectSprays data in a table, namely, give the frequency under each spray that the "count" of found insects was over 10.

Lesson 7: Generating random data & probability distributions

The generation of random data has many uses in statistical analysis. You may want to evaluate methods in a simulation study, create a visual depiction of a probability distribution or to simply make some data to play around with!

Random sample of data

The function sample() allows you to create a random permutation of data and take samples with and without replacement.

```r
sample(seq(1,10)) #permute full data set
```

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

```r
sample(seq(1,10),replace=TRUE) #sample the same number of observations as was in the data set, with rep
```

```
## [1] 1 4 7 9 8 8 2 7 4 3
```

```r
sample(seq(1,10),size=2) #sample 2 observations from the data set, without replacement
```

```
## [1] 3 7
```

For more permutations/combinatorics, look up the choose() function.

Binomial distribution

Today you should have learned about probability distributions. R gives us the capabilities to draw random numbers from probability distributions, as well as finding their probability densities, inverse probability densities, and cumulative probability densities.

Let's begin in the discrete case: a binomial distribution with $n = 9$ and $p = 0.4$. The probability density function for this distribution looks like:

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} = \binom{9}{x} (0.4)^x (0.6)^{9-x}$$

We can find the probability that $x = 3$ by plugging this into our equation. Use R to calculate this probility manually now (HINT: the choose() function may come in handy):

R also has a one-line command to find this probability:

```r
dbinom(x = 3, size = 9, prob = 0.4)
```

Does this answer match your earlier calculations?

Maybe, instead, we are interested in the probability that $x \leq 3$. This means that $x = 0, 1, 2$ or 3 and is a cumulative probability. If we wish to do this manually we must sum up the probabilities of x being any of these numbers:

$$P(x \leq 3) = P(x = 0) + P(x = 1) + P(x = 2) + P(x = 3)$$

Find this sum, either by plugging in numbers manually into the probability density function or by using dbinom():

Then, compare this against the R function for cumulative probability:

```
pbinom(q=3, size = 9, prob=0.4)
```

Let us find the smallest value of x for which $F(x) \geq p = 0.8$. You can do this using a trial-and-error-method using pbinom():

Compare your answer to that generated by:

```
qbinom(p = 0.8, size = 9, prob = 0.4)
```

Finally, let us sample some random numbers from this binomial distribution:

```
rbinom(10,size=9,prob=0.4)
```

```
##  [1] 3 3 5 6 5 5 2 5 4 4
```

Do the sample numbers reflect what you'd expect given the n and p?

Commands for other distributions follow the same pattern (Ex: rnorm(), pnorm(), qnorm() and dnorm()).

Exercise 7: Generating random data & probability distributions

1. Randomly sample 10 numbers from 1 to 100 without replacement, ensuring that numbers 1-20 have double the chance of being sampled as numbers 21-100 (HINT: there are two ways to do this).
2. Brad was told that his test score was in the 94th percentile of test scores in the nation, and that scores were distributed according to a normal distribution with an average of 79 and a standard deviation of 7. What was Brad's test score?
3. Simulate and store random draws size 10, 20, 50 and 100 all from a normal distribution with a mean of 3 and a standard deviation of 8. Have R estimate the mean and standard deviation of these samples using the mean() and sd() functions. What do you notice about the quality of estimation? Why?
4. Simulate tossing 30 coins with rbinom() and sample().
5. A rule of thumb is that 5% of a normal distribution lies outside an interval approximately 2 standard deviations above or below the mean. To what extent is this true? How long are the intervals (in terms of standard deviations) which exclude 10%, 1% or 0.5% of the normal distribution? BONUS: How would you relate the relationship between interval length and "coverage" of the normal distribution to the concept of confidence intervals?
6. Diane is a surgeon testing a new procedure. If the procedure has a 20% failure rate, she will stop using it. She has thus far operated on 10 patients and there have been no failures. What is the probability of this occurence if the procedure actually does have a 20% failure rate?
7. Amy plays an unusual lottery where every number chosen is assigned a different binomial distribution. There are 8 numbers, and the first number has a binomial distribution with $n = 10$, $p = 0.1$, the second

number has a binomial distribution with $n = 20$, $p = 0.2$ and on until the 8th number has a binomial distribution with $n = 80$,$p = 0.8$. Amy's numbers this week are: 3 8 27 27 48 19 32 52. What is her probability of winning? BONUS: which numbers would give her the largest chance of winning and what is that chance?

Lesson 8: Saving & exporting data

Now that we've finished manipulating the babyNames dataset, we may want to save our changes:

```r
# write data to a .csv file
write_csv(babyNames, path = "~/Desktop/newbabyNames.csv")

# write data to an R file
save(babyNames, file = "~/Desktop/newbabyNames.Rdata")
```

Summary

- R works primarily using functions, which can be installed by the way of packages
- Vectors are the building blocks of all data storage within R, though most data you import will be in data tables
- When in doubt, look up help documents or google it!

Sources and further reading:

- http://tutorials.iq.harvard.edu/R/Rintro/Rintro.html
- Dalgaard, P. (2008). Introductory statistics with R.