R Lesson One: Getting Started

Find R in a lab or install it and open it. I'll leave this part up to you! In what follows, the "carrot" at the beginning of some lines is a prompt given by R and not something to be typed.

Note: There are more clever ways to do many of the tasks in this tutorial. This is just a bare bones getting started document!

Defining Variables and Vectors

1. Assign a variable x to have the value 4.7:

   ```
   > x<-4.7
   ```

   Now to see the value you just assigned to x, you simply type "x".

   ```
   > x
   [1] 4.7
   ```

2. Define x to be the vector containing the elements 2, 1, and 5. To do this you use a "c". For this reason it is not a good idea to use "c" as a variable name.)

   ```
   > x<-c(2,1,5)
   ```

   ... and, take a look at it:

   ```
   > x
   [1] 2 1 5
   ```

   Access an element of it, say the second one by typing x[2]:

   ```
   > x[2]
   [1] 1
   ```

   Note that vector indices in R start from 1 as opposed to some languages where they start from 0.

3. Say you wanted to make a vector containing, in order, the numbers 2 through 8. Type

```
x<-c(2,3,4,5,6,7,8)
```

or, more simply

```
x<-2:8
```

or, use the "sequence function":

```
> x<-seq(2,8,1)
```

which tells you to go from 2 to 8 in increments of 1.

What do you expect to see from the following command? (Try it out!)

```
> x<-seq(2,8,2)
```

4. Sometimes you'll want to initialize a vector, filling it with all of one number. Let's make a vector of length 10 containing all zeros:

   ```
   > x<-rep(0,10)
   ```

   Now let's assign the value 3 to the 7th position:

   ```
   > x[7]<-3
   ```

Vector Operations

R has awesome vector/matrix manipulation capabilities!

1. Define two vectors x and y as follows:

   ```
   > x<-c(1,0,4,2,-1,0)
   > y<-c(2,7,1.3,5,1)
   ```

2. The purpose of this command should be obvious:

   ```
   > length(x)
   [1] 6
   ```

3. Now suppose we want to find out how many zeros are in the vector x.

   ```
   > length(x[x==0])
   [1] 2
   ```

or, the proportion of zeros:

```
> length(x[x==0])/length(x)
[1] 0.3333333
```

The "x[x==0]" can be read as "x such that x equals 0".

4. Check out the following commands:

```
> 2*x
[1]  2  0  8  4 -2  0
> x+3
[1] 4 3 7 5 2 3
```

Suppose you wanted to store the results in a new vector called z.

```
> z<-2*x
```

Do you remember how to look at z?

```
> z
[1]  2  0  8  4 -2  0
```

Let's turn those zeros into fives:

```
> z[z==0]<-5
```

Since there are now two 5's in the vector z, if we type

```
z[z==5]
```

we would get a vector of length 2. We could reassign those 5's, consecutively as, say, a 3 and 7 in one line by substituting in the vector containing 3 and 7:

```
> z[z==5]<-c(3,7)
> z
[1]  2  3  8  4 -2  7
```

Cool! "R" has amazing vector manipulation capabilities but is very slow to do "looping" operations. Part of the fun of using R is trying to figure out how to define things in a slick way in order to avoid loops. Did you notice how we replaced the 5's in z without looping through the elements one by one and checking for 5's?

5. Element-wise addition anyone?

```
> x+y
[1] 3.0 7.0 5.3 7.0 0.0 1.0
```

Assign the result to the vector z:

```
> z<-x+y
```

Element-wise subtraction, multiplication, and division also work the same way!

6. Element-wise squaring anyone?

```
> x^2
[1]  1  0 16  4  1  0
```

7. Sum all the elements of x:

```
> sum(x)
[1] 6
```

or the elements of x^2:

```
> sum(x^2)
[1] 22
```

8. I bet you can guess what the following commands do.

```
> mean(x)
[1] 1
> min(x)
[1] -1
> max(x)
[1] 4
```

There's even a variance command:

```
> var(x)
[1] 3.2
```

A standard deviation anyone?

```
> sqrt(var(x))
[1] 1.788854
```

Let's try that "from scratch". Recall that the sample variance is given by

$$S^2 = \frac{\sum X_i^2 - (\sum X_i)^2/n}{n-1}$$

and that the standard deviation is the square root. You could try this:

```
> sqrt((sum(x^2)-(sum(x)^2)/length(x))/(length(x)-1))
[1] 1.788854
```

Let's do that again, first storing the length n and then storing the final answer as s.

```
> n<-length(x)
> s<-sqrt((sum(x^2)-(sum(x)^2)/n)/(n-1))
> s
[1] 1.788854
```

What about the other definition of variance?

$$S^2 = \frac{\sum (X_i - \overline{X})^2}{n-1}$$

Since "mean(x)" gives a scalar, "x-mean(x)" would be a vector where the scalar mean was subtracted from each element of the vector x. So, we could do this to compute the variance:

```
> sum((x-mean(x))^2)/(n-1)
[1] 3.2
```

Matrices:

1. Let's define two 3×3 matrices

$$A = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0.3 & 0.3 & 0.4 \\ 0.1 & 0.6 & 0.3 \\ 0.5 & 0.2 & 0.3 \end{bmatrix}$$

In R set up the A matrix, initially filled with zeros.

```
> A<-matrix(0,3,3)
> A
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

2. We can access (for example)
   - the 1-3 entry by typing `A[1,3]`
   - the 2nd row by typing `A[2,]`
   - the 3rd column by typing `A[,3]`
3. Fill in the rows with the appropriate vectors as follows.

```
> A[1,]<-c(0.6,0.3,0.1)
> A[2,]<-c(0.2,0.5,0.3)
> A[3,]<-c(0.2,0.3,0.5)
```

   View the matrix by typing `A`.

   Go ahead and set up the `B` matrix on your own.
4. Note that, if we try to multiply the matrices with

```
> A*B
     [,1] [,2] [,3]
[1,] 0.18 0.09 0.04
[2,] 0.02 0.30 0.09
[3,] 0.10 0.06 0.15
```

   that you are only doing elementwise multiplication and not matrix multiplication.

   To do matrix multiplication, use `%*%`.

```
> A%*%B
     [,1] [,2] [,3]
[1,] 0.26 0.38 0.36
[2,] 0.26 0.42 0.32
[3,] 0.34 0.34 0.32
```

5. The inverse of matrix `A` is obtained using the "solve" command.

```
> solve(A)
     [,1] [,2] [,3]
[1,]  2.0 -1.5  0.5
[2,] -0.5  3.5 -2.0
[3,] -0.5 -1.5  3.0
```

6. When defining a vector by

```
> x<-c(2,1,5)
```

and using it in matrix operations, it will automatically be converted to a 3×1 matrix to look like a column vector and the outcome will be another "column vector matrix".

```
> A%*%x
     [,1]
[1,]  2.0
[2,]  2.4
[3,]  3.2
```

7. The transpose of a matrix:

```
> t(A)
     [,1] [,2] [,3]
[1,]  0.6  0.2  0.2
[2,]  0.3  0.5  0.3
[3,]  0.1  0.3  0.5
```

Histograms and Generating Values from the Uniform Distribution:

1. Assign `u` to be a value from the continuous uniform distribution over the interval $(0, 1)$ as follows.

```
> u<-runif(1)
> u
[1] 0.8032065
```

2. Assign `u` to be a vector of 10 values from the uniform $(0, 1)$ distribution as follows.

```
> u<-runif(10)
> u
 [1] 0.19812272 0.14867392 0.14788449 0.53694538 0.03466262 0.88710010
 [7] 0.68206183 0.43522225 0.97641464 0.64512447
```

3. Make a histogram of these values

```
> hist(u)
```

Note that the histogram is just using pure counts on the y-axis and has bin widths of 0.2. Let's make this look nicer.

4. Bin width: Define a sequence of "breaks" from 0 to 1 in increments of 0.1.

```
> br<-seq(0,1,0.1)
```

5. Density: Now let's remake the histogram with these breaks. Let's also make it so that the total area of each bar is the probability (and not count) of being in the bin by using the command "`prob=T`".

```
> hist(u,prob=T,breaks=br)
```

6. Let's superimpose the flat line uniform density. The following command will draw a line between 2 points. The points for the "line command" are not given as (x_1, y_1) and (x_2, y_2) but rather as (x_1, x_2) and (y_1, y_2).

```
> lines(c(0,1),c(1,1))
```

7. This histogram probably doesn't match the uniform density that well because of the small sample. Try again for a vector of 100,000 uniforms.

```
> u<-runif(100000)
> hist(u,prob=T,breaks=br)
> lines(c(0,1),c(1,1))
```

Simulating a Simple Markov Chain:

Suppose that we want to simulate a simple Markov chain on $S = \{1, 2\}$ with transition probability matrix

$$\mathbf{P} = \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \left\| \begin{array}{cc} 0.8 & 0.2 \\ 0.4 & 0.6 \end{array} \right\| \end{array}$$

1. Set up the matrix.

```
> P<-matrix(0,2,2)
> P[1,]<-c(0.8,0.2)
> P[2,]<-c(0.4,0.6)
```

2. Suppose that your chain is currently at state 1. The next state must be picked from $\{1, 2\}$ with probabilities 0.8 and 0.2, respectively.

3. Note that a uniform $(0, 1)$ random variable will be less than 0.8 with probability 0.8 and greater than 0.8 with probability 0.2. So, let's generate a value from the uniform distribution and assign the next value of our chain to be 1 or 2 depending on what we get.

   Try typing this

```
> x<-1
> u<-runif(1)
> if(u<=P[1,1]){x<-1}
```

   You'll notice that we didn't get a chance to assign `x` to be 2. By closing that final bracket and hitting enter, R thought that we were done with conditions we want to check.

   We need to make sure we start our "else" before completing that line:

```
> x<-1
> u<-runif(1)
> if(u<=P[1,1]){x<-1}else{
  x<-2}
```

   (Note that when R is waiting for you to finish a line after hiting enter, it will put in a prompt that looks like a plus sign. I didn't show that here.)

4. The full simulation of 1 value at the end of one long chain of length 10,000:

```
> x<-1
> for(i in 1:10000){
      u<-runif(1)
      if(u<=P[x,1]){x<-1}else{
      x<-2}
  }
```

   The output is

```
> x
[1] 2
```

   (Yours may be different!) Note that we here we could have put the "else" more neatly in the next line because we are still inside a larger loop and R isn't going to execute any commands until we close it.

5. Let's make a vector to store the output and run this many times.

```
> output<-rep(0,100000)
> for(j in 1:100000){
      x<-1
      for(i in 1:10000){
         u<-runif(1)
         if(u<=P[x,1]){x<-1}else{
         x<-2}
      }
      output[j]<-x
 }
```

This **will be very slow** but will get the job done! R is not very good at looping. (It's not helping that we are using overkill on the length of our chains!)

To get estimates for π_1 and π_2, type

```
> length(output[output==1])/length(output)
[1] 0.6665
> length(output[output==2])/length(output)
[1] 0.3335
```

6. Avoiding the loop.

   Coming soon..... (evening of 2/15)