Introduction to R
(for Stat 102, 107, 111, 139)

W. Ryan Lee

This guide is designed for students of Statistics 102, 107, 111, and 139 at Harvard, and is intended to serve as a basic introduction to using R for data analysis. Please follow along on your own machine with the examples!

1 Math

Common Operations Anything that a calculator can do, R can do better:

```
> exp(2)
[1] 7.389056

> sin(pi/6)
[1] 0.5

> log(10); log(10, base = 10)
[1] 2.302585
[2] 1

> sqrt(16)
[1] 4

> abs(3-7)
[1] 4

> round(pi, 0); round(pi, 1); round(pi, 4)
[1] 3
[2] 3.1
[3] 3.1416
```

Matrices R works very well with matrix operations. The way you construct a matrix is to create a vector and then "shape it" into the dimensions you want. For example:

```
> matrix(c(1,2,3,4,5,6), byrow = T, nrow = 3)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

What the above code says is:

1. Form the vector $(1, 2, 3, 4, 5, 6)$
2. Create the matrix "by row"; that is, put the first element in a_{11}, then the second in a_{12} (not a_{21}), and so forth.
3. Use 3 rows. (R figures that it then needs 2 columns)

Matrix Operations Suppose we define the matrices:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, B = \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix}$$

(We'll figure out how to do this in the next section.) Then, R can do transposes, matrix multiplication, etc.:

```
> t(A)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> t(A) %*% B
     [,1] [,2]
[1,]   89   98
[2,]  116  128

> A * B
     [,1] [,2]
[1,]    7   16
[2,]   27   40
[3,]   55   72

> A[1:2,]
     [,1] [,2]
[1,]    1    2
[2,]    3    4

> solve(A[1:2,])
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

Note that * is element-wise multiplication, whereas %*% is matrix multiplication. “solve” yields the matrix inverse.

2 Syntax

Variables + Computation To define variables, you can use either ¡- or =:

```
> A <- matrix(c(1,2,3,4,5,6), byrow = T, nrow = 3)
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6

> x = c(1,0,0)
> x
[1] 1 0 0

> t(A) %*% x
     [,1]
[1,]    1
[2,]    2

> y <- c(2,3,3)
> x + y
[1] 3 3 3

> z = A[,1]; w = A[,2]
> z
[1] 1 3 5

> z + w
[1] 3 7 11
```

One particularly useful way to create sequence vectors in R is the following:

```
> x <- 1:10
[1]  1  2  3  4  5  6  7  8  9 10

> y <- seq(1,10)
[1]  1  2  3  4  5  6  7  8  9 10
```

Control Flow + Loops Since R is built on C, most of the control flow syntax is similar to that of C. However, for loops are particularly simple in R (i.e. like Python):

```
> for (i in 1:100) {
>     print(i*2)
> }
```

It used to be the case that for loops were frowned upon, and much less efficient, compared to using a more advanced method, *apply*. However, for most usages the differences are generally not significant enough to be too consequential.

Control flow tools include if, else, else if, and so forth:

```
> x <- 9001
> if (x > 9000) {
>     print("It's over 9,000!!!")
> } else {
>     print("Nope.")
> }
[1] "It's over 9,000!!!"

> if (x > 10000) {
>     x * 2
> } else if (x > 9000 & x <= 10000) {
>     print("But it's not over 10,000!")
> } else {
>     print("Weak!")
> }
[1] "But it's not over 10,000!"
```

3 Inputting Data

To input data:[1]

1. Switch into the directory that has the data set, in R. For example, if we want to go to ∼/Downloads:

   ```
   > setwd("~/Downloads")
   ```

2. Use a load function to load the data set, and assign it a variable. We will generally deal with two kinds of data sets:

   - **CSV files**: Download the *crabs.csv* dataset: `http://won1k.github.io/data/crabs.csv`
     We can input this dataset into R by using:

     ```
     > crabs <- read.csv("crabs.csv")

     > head(crabs)
       color spine width satell weight
     1     3     3  28.3      8   3050
     2     4     3  22.5      0   1550
     3     2     1  26.0      9   2300
     4     4     3  24.8      0   2100
     5     4     3  26.0      4   2600
     6     3     3  23.8      0   2100
     ```

   - **DAT files**: Sometimes, data files for R come in the *.dat* format. The same horseshoe crabs dataset can be found in this format at: `http://won1k.github.io/data/crabs.dat`.
     This can be inputted into R by using:

     ```
     > crabs <- read.table("crabs.dat", header = T)
     ```

     Note we had to use the "header = T" option because the dataset came with column names. Otherwise, R would just create new ones, i.e. "V1, V2, ..."

[1] There is one true dataset you need to know, and it concerns the mating patterns of horseshoe crabs. Dataset was provided by A. Agresti, *Foundations of Linear and Generalized Linear Models*, 2015.

4 Summary Statistics + Regression

Where R really shines is letting you calculate statistics in single-line commands. For example, let's return to our beloved *crabs* dataset.

Selecting Variables The first thing to note is that we can select individual columns in a dataset in one of two ways:

```
> crabs$width
[1] 28.3 22.5 26.0 24.8 26.0 23.8 26.5 ...

> crabs[,3]
[1] 28.3 22.5 26.0 24.8 26.0 23.8 26.5 ...
```

Summaries for Individual Variables Let's let x be the column with the shell widths of these crabs. Then we can find summary statistics:

```
> x <- crabs$width

> mean(x)
[1] 26.29884
> sd(x)
[1] 2.109061
> var(x)
[1] 4.448138
> length(x)
[1] 173
> median(x)
[1] 26.1
```

We might also be interested in how two variables are correlated. Perhaps girth-ier crabs are associated with more satellites?

```
> y <- crabs$satell

> cor(x,y)
[1] 0.3398903
```

Selecting Subsets One important technique for data analysis in R is to be able to select subsets of your data that meet a certain criteria. For example, we may only be concerned with horseshoe crabs that have color 3. In that case, we can do one of two things:

```
> crabs3 <- subset(crabs, crabs$color == 3)
> head(crabs3)
   color spine width satell weight
1      3     3  28.3      8   3050
6      3     3  23.8      0   2100
9      3     1  23.7      0   1950
12     3     3  25.8      0   2650
13     3     3  28.2     11   3050
15     3     1  26.0     14   2300

> crabs3.2 <- crabs[crabs$color == 3,]
> head(crabs3.2)
   color spine width satell weight
1      3     3  28.3      8   3050
6      3     3  23.8      0   2100
9      3     1  23.7      0   1950
12     3     3  25.8      0   2650
13     3     3  28.2     11   3050
15     3     1  26.0     14   2300
```

Note that in either case, we have ended up with the subset of the data that contains precisely the rows for which the color is 3. We can use identical methods for conditions such as $\leq, \geq$.

Overall Summary One useful command is to run a "summary" on the entire dataset, which yields max, min, mean, etc. for every variable in the dataset:

```
> summary(crabs)
    color           spine           width          satell           weight
Min.   :2.000   Min.   :1.000   Min.   :21.0   Min.   : 0.000   Min.   :1200
```

```
1st Qu.:3.000   1st Qu.:2.000   1st Qu.:24.9   1st Qu.: 0.000   1st Qu.:2000
Median :3.000   Median :3.000   Median :26.1   Median : 2.000   Median :2350
Mean   :3.439   Mean   :2.486   Mean   :26.3   Mean   : 2.919   Mean   :2437
3rd Qu.:4.000   3rd Qu.:3.000   3rd Qu.:27.7   3rd Qu.: 5.000   3rd Qu.:2850
Max.   :5.000   Max.   :3.000   Max.   :33.5   Max.   :15.000   Max.   :5200
```

Regression We can also run a regression between two or more variables. We use the "lm" (for "linear model", i.e. the subject of Stat 139) command:

```
> fit <- lm(y ~ x)

> summary(fit)
Call:
lm(formula = crabs$satell ~ crabs$width)

Residuals:
Min      1Q  Median      3Q     Max
-4.1374 -2.2093 -0.7379  1.8921 11.2326

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -10.4244     2.8324  -3.680 0.000312 ***
crabs$width   0.5074     0.1074   4.726 4.76e-06 ***
...
```

Note that we use "lm" with the formula $y \sim x$ to say that we want to regress y on x, that is fit a line $y = a \cdot x + b$. The "(Intercept)" estimate yields the estimated value of b, while the coefficient of "crabs$width" is the estimated value of a.

5 Graphics + Plotting

Of course, statistics is no fun without the pretty pictures. Fortunately, R knows all about making plots and graphs!

Histograms One of the quickest ways to get a sense of a dataset is through a histogram:

```
> hist(crabs$width)

> hist(crabs$width, breaks = 25)
```

The "breaks" option is often quite useful to get more fine-grained observations.

Plot For plotting 2 variables, the simplest (and often most useful) way is to use the "plot" command:

```
> plot(crabs$width, crabs$satell)

> plot(crabs$satell ~ crabs$width)
```

Both of the above commands, as you can test, yield the exact same plot. Thus, you can use:

```
> plot(x,y)

> plot(y ~ x)
```

as you prefer! (Try plotting other relationships between the variables in the dataset.)

Adding Lines We can also add fitted lines to our plot:

```
> fit <- lm(crabs$satell ~ crabs$width)
> plot(crabs$width, crabs$satell)
> abline(fit)
```

Moreover, we can also add vertical bars at designated x values, for example at the mean and 2 standard deviations:

```
> plot(crabs$width, crabs$satell)
> abline(v = mean(crabs$width), lty = 2)
> abline(v = mean(crabs$width) + 2 * sd(crabs$width), lty = 3)
> abline(v = mean(crabs$width) - 2 * sd(crabs$width), lty = 3)
```

As you can guess, the “lty” option allows us to change the style of the lines that we plot.

Tables + Barplots Another useful command is *table*, which shows the frequency of particular values in the dataset. For example:

```
> table(crabs$color)
 2  3  4  5
12 95 44 22

> table(crabs$satell)
 0  1  2  3  4  5  6  7  8  9 10 11 12 14 15
62 16  9 19 19 15 13  4  6  3  3  1  1  1  1
```

This might look an awful lot like a barplot, and indeed we can easily make a barplot from a table:

```
> barplot(table(crabs$satell), name = "# of Satellites")
```

Contingency Tables One very nice use of the “table” command is to build a *contingency table* between two variables. for example:

```
> table(crabs$satell, crabs$color)
     2  3  4  5
0    3 26 18 15
1    0 11  5  0
2    1  6  2  0
3    1  9  7  2
4    1 12  5  1
5    1 11  2  1
6    3  8  2  0
7    0  3  1  0
8    1  4  0  1
...
```

This tells us, for example, that the number of crabs in the dataset with color 3 and 4 satellites is 12. Similarly, there are 7 crabs in the dataset with color 4 and 3 satellites.

Scatterplot + Correlation Matrix Another fun technique for exploratory data analysis is to create a *scatterplot matrix* in R, which visualizes correlations between every pair of variables in the dataset (or the subset you choose).

```
> attach(crabs)
> pairs(~ width + satell + weight)
```

which yields scatterplots between width and weight, width and satell, and so forth.

We can also construct a correlation matrix for all the variables in the dataset, or a subset, as follows:

```
> cor(crabs)
            color       spine      width      satell     weight
color   1.0000000  0.37850163 -0.2643863 -0.19078455 -0.2507772
spine   0.3785016  1.00000000 -0.1218946 -0.08993242 -0.1664817
width  -0.2643863 -0.12189458  1.0000000  0.33989033  0.8868715
satell -0.1907846 -0.08993242  0.3398903  1.00000000  0.3692474
weight -0.2507772 -0.16648173  0.8868715  0.36924744  1.0000000

> cor(crabs[,3:5])
           width    satell    weight
width  1.0000000 0.3398903 0.8868715
satell 0.3398903 1.0000000 0.3692474
weight 0.8868715 0.3692474 1.0000000
```

A fun way to visualize this correlation matrix is to use the *corrplot* library:

```
> install.packages('corrplot')
> library('corrplot')

> M <- cor(crabs[,3:5])
> corrplot(M)
```