Lab 1: R Studio, knitr and review of R

YOUR NAME HERE

September 15, 2015

Resources

This website has good handout on how to install and use *knitr* and Rstudio for building reports.

`http://www.r-bloggers.com/using-r-in-latex-with-knitr-and-rstudio/`
`http://www.rstudio.com/`

On Moodle

This document will help you with typing up your homework. Latex is a good typesetting program which handles mathematical equations at arbitrary sophistication. Your homework and take home exam must be written in a combination of LaTeXand knitr/sweave (an add-on to which allows you inset your R code into the document).

> *The Not So Short Introduction to LaTeX* (lshort.pdf)

Lab # 1 Assignment: To be Turned in on 09/12/2014 via Moodle

It is expected that each individual will install `R` and run through a "LaTeXintroduction", "A Brief Reminder of Univariate Statistics with R", and "A Brief Introduction to R Commands" on their own machine. They will produce a .pdf using RStudio, which they can save and upload to Moodle under **Lab#1**.

What is R

`R` is a free software programming language and environment developed originally as the S programming language by Bell Labs. `R` is maintained by the R-Core team (for details see `http://www.r-project.org/`) and has in recent years become the preferred tool for statistical data analysis by statisticians.

Installing R

To install the latest version of R go to `http://www.r-project.org/` and select **Download R**. On the next page scroll down and select `http://cran.mtu.edu/` as your preferred download location (note you can of course go directly to aforementioned website). Next, select your operating system (e.g., OS X or Windows). For Windows, click on **install R for the first time** and then click on **Download R 3.0.1 for Windows**. For OS X, click the first link under the headline "Files:" (e.g., **R-3.0.1.pkg**). Remember If you have any trouble installing R on your laptop you can always come into office hours.

Installing RStudio

To install RStudio go to `http://www.rstudio.com/`. Click on the link below "If you run R on your desktop:" and then download the RStudio under "Recommended For Your System" (e.g., **RStudio 0.97.551 - Mac OS X 10.6+ (64-bit)**).

Installing Additional Packages in R

Here we will introduce you to installing additional packages for the R statistical language. The example here will be the required package `knitr`, which contains all the necessary tools to write up your document in a combination of *Sweave* and LaTeX. To install packages in R the command `install.packages` is used within the command line prompt >. To install the package `knitr` you type in:

```
install.packages("knitr")
```

`R` will then prompt you to choose your "download mirror." I recommend choosing USA (MI), which will facilitate downloading the package onto your system from the near by state of Michigan. As always, if you have trouble installing packages it is recommended that you bring your laptop to office hour.

LaTeXIntroduction

Installation

- MikTex `http://miktex.org/`
- MacTex `https://tug.org/mactex/`
  - TexShop `http://pages.uoregon.edu/koch/texshop/`

Problem Set

Writing LaTeXis fairly simple, but it is a sophisticated mark-up language like XML (in someways). Below are a series of exercises (using the Not To Short Introduction to LaTeX):

1. Write up the expression for the sample mean and sample variance.
2. Write up the expression for a discrete expectation ($E(X)$).
3. Write up the expression for the Binomial distribution.

A Brief Reminder of Univariate Statistics with R

Brief review of the standard sample statistics: mean, standard deviation, variance, and confidence intervals; and the Normal distribution. Given a that x_i, $i = 1, \ldots, n$, is drawn from a *simple random sample*. This lab will use an example from the `Sleuth3` library, *Darwin's Data* which is composed of a two-column matrix comprised of the height of Cross and Self fertilized plants. The first column is *Cross*, for cross-fertilized, and the second column is *Self* for self-fertilized.

First we will load the necessary libraries and data in `R`. Note that after loading the data, we will run `head` function to view a small portion of the matrix and then we will run `summary` to get a brief breakdown of the data.

```
install.packages("Sleuth3")
```

```
library(Sleuth3)
data(ex0428)  ## Load's Darwin's plant data
head(ex0428)
```

```
  Cross  Self
1 23.50 17.38
2 12.00 20.38
3 21.00 20.00
4 22.00 20.00
5 19.13 18.38
6 21.50 18.63
```

```
summary(ex0428)
```

```
     Cross            Self      
 Min.   :12.00   Min.   :12.75  
 1st Qu.:19.75   1st Qu.:16.38  
 Median :21.50   Median :18.00  
 Mean   :20.19   Mean   :17.58  
 3rd Qu.:22.13   3rd Qu.:18.63  
 Max.   :23.50   Max.   :20.38  
```

Sample Mean:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

```
sum(ex0428$Cross)/length(ex0428$Cross)
```

```
[1] 20.19333
```

```r
mean(ex0428$Cross)  ## built in function
```

```
[1] 20.19333
```

```r
mean(ex0428$Self)
```

```
[1] 17.57667
```

Sample Variance (unbiased):

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})^2$$

```r
sum((ex0428$Cross - mean(ex0428$Cross))^2)/(length(ex0428$Cross) -
    1)
```

```
[1] 13.08545
```

```r
s <- var(ex0428$Cross)  ## Built in function
s
```

```
[1] 13.08545
```

```r
var(ex0428$Self)
```

```
[1] 4.21331
```

Sample Standard Deviation:

$$s = \sqrt{s^2}$$

```r
sqrt(s)
```

```
[1] 3.617382
```

```r
sd(ex0428$Cross)
```

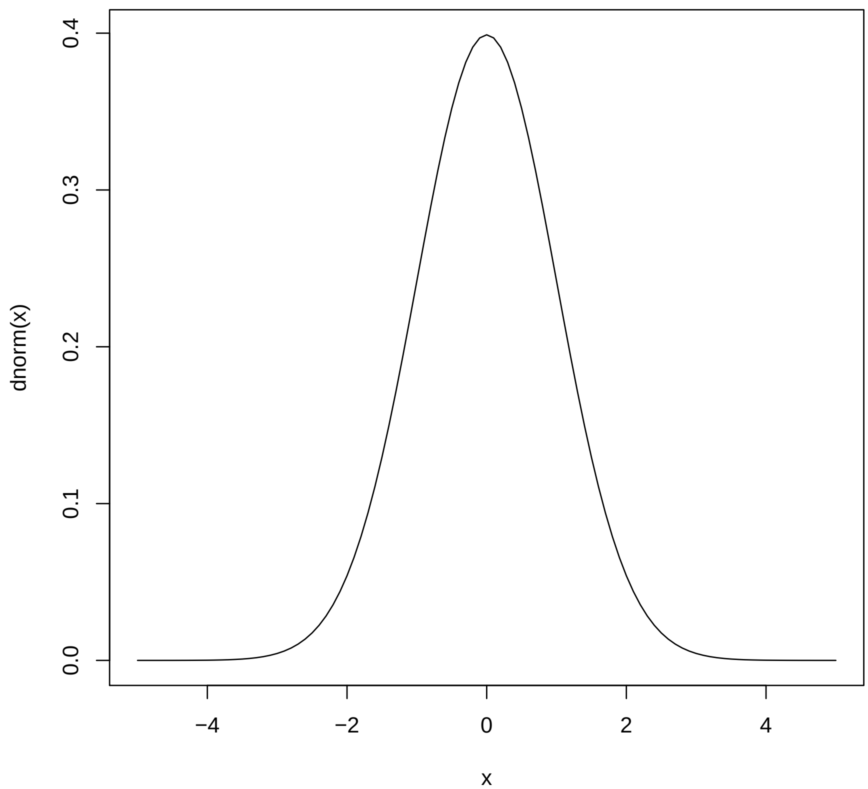
```
[1] 3.617382
```

```r
sd(ex0428$Self)
```

```
[1] 2.052635
```

Z Distribution (the Normal Distribution with mean 0 and standard deviation 1):

```r
curve(dnorm(x), -5, 5)
```



95% Confidence Interval:

```r
error <- qnorm(0.975) * sd(ex0428$Cross)/sqrt(length(ex0428$Cross))
left <- mean(ex0428$Cross) - error
left
```

```
[1] 18.36272
```

```r
right <- mean(ex0428$Cross) + error
right
```

```
[1] 22.02395
```

A Brief Introduction to R Commands

The following section is taken from "The World's Shortest R Tutorial"[1] and contains a brief introduction to `R` syntax and commands that you might find useful throughout the class. This is largely meant to be used as a resource to learn R and look back on through out the course.

Introduction to basic R syntax

```
a <- 3  # assignment
a  # evaluation
```

```
[1] 3
```

```
sqrt(a)  # perform an operation
```

```
[1] 1.732051
```

```
b <- sqrt(a)  # perform operation and save
b
```

```
[1] 1.732051
```

```
a == a  # A is A?
```

```
[1] TRUE
```

```
a != b  # A is not B
```

```
[1] TRUE
```

```
ls()  # list objects in global environment
```

```
[1] "a"      "b"      "error"  "ex0428" "left"   "right"
[7] "s"
```

```
# help(sqrt) # help w/ functions ?sqrt # same thing help.start() #
# lots of help help.search('sqrt') # what am I looking for?
# apropos('sqr') # it's on the tip of my tongue...

rm(a)  # remove an object
```

Vectors and matrices in R

[1] Originally from "The World's Shortest R Tutorial", which can be found here `https://statnet.csde.washington.edu/trac/raw-attachment/wiki/Resources/statnet_polnet2011_ergm.pdf` and elsewhere.

```r
# Creating vectors using the concatenation operator
a <- c(1, 3, 5)  # create a vector by concatenation
a
```

```
[1] 1 3 5
```

```r
a[2]  # select the second element
```

```
[1] 3
```

```r
b <- c("one", "three", "five")  # also works with strings
b
```

```
[1] "one"   "three" "five"
```

```r
b[2]
```

```
[1] "three"
```

```r
a <- c(a, a)  # can apply recursively
a
```

```
[1] 1 3 5 1 3 5
```

```r
a <- c(a, b)  # mixing types - who will win?
a  # there can be only one!
```

```
[1] "1"     "3"     "5"     "1"     "3"     "5"     "one"
[8] "three" "five"
```

```r
# Sequences and replication
a <- seq(from = 1, to = 5, by = 1)  # from 1 to 5 the slow way
b <- 1:5  # a shortcut!
a == b  # all TRUE
```

```
[1] TRUE TRUE TRUE TRUE TRUE
```

```r
rep(1, 5)  # a lot of 1s
```

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

```r
rep(1:5, 2)  # repeat an entire sequence
```

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

```r
rep(1:5, each = 2)  # same, but element-wise
```

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

```r
rep(1:5, times = 5:1)  # can vary the count of each element
```

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

# Any and all (with vectors)
a <- 1:5  # create a vector
a > 2  # some TRUE, some FALSE

[1] FALSE FALSE  TRUE  TRUE  TRUE

any(a > 2)  # are any elements TRUE?

[1] TRUE

all(a > 2)  # are all elements TRUE?

[1] FALSE

# From vectors to matrices
a <- matrix(1:25, nr = 5, nc = 5)  # create a matrix the 'formal' way
a

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

a[1, 2]  # select a matrix element (two dimensions)

[1] 6

a[1, ]  # shortcut for ith row

[1]  1  6 11 16 21

all(a[1, ] == a[1, 1:5])  # show the equivalence

[1] TRUE

a[, 2]  # can also perform for columns

[1]  6  7  8  9 10

a[2:3, 3:5]  # select submatrices

     [,1] [,2] [,3]
[1,]   12   17   22
[2,]   13   18   23
```

```
a[-1, ]  # nice trick: negative numbers omit cells!
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    7   12   17   22
[2,]    3    8   13   18   23
[3,]    4    9   14   19   24
[4,]    5   10   15   20   25
```

```
a[-2, -2]  # get rid of number two
```

```
     [,1] [,2] [,3] [,4]
[1,]    1   11   16   21
[2,]    3   13   18   23
[3,]    4   14   19   24
[4,]    5   15   20   25
```

```
b <- cbind(1:5, 1:5)  # another way to create matrices
b
```

```
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
[4,]    4    4
[5,]    5    5
```

```
d <- rbind(1:5, 1:5)  # can perform with rows, too
d
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    2    3    4    5
```

```
try(cbind(b, d))  # no go: must have compatible dimensions!
dim(b)  # what were those dimensions, anyway?
```

```
[1] 5 2
```

```
dim(d)
```

```
[1] 2 5
```

```
NROW(b)
```

```
[1] 5
```

```
NCOL(b)
```

```
[1] 2
```

```r
cbind(b, b)  # here's a better example
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    2    2    2    2
[3,]    3    3    3    3
[4,]    4    4    4    4
[5,]    5    5    5    5
```

```r
t(b)  # can transpose b
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    2    3    4    5
```

```r
cbind(t(b), d)  # now it works
```

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

Element-wise operations

```r
# Most arithmetic operators are applied element-wise
a <- 1:5
a + 1  # addition
```

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

```r
a * 2  # multiplication
```

```
[1]  2  4  6  8 10
```

```r
a/3  # division
```

```
[1] 0.3333333 0.6666667 1.0000000 1.3333333 1.6666667
```

```r
a - 4  # subtraction
```

```
[1] -3 -2 -1  0  1
```

```r
a^5  # you get the idea...
```

```
[1]    1   32  243 1024 3125
```

```r
a + a  # also works on pairs of vectors
```

```
[1]  2  4  6  8 10
```

```r
a * a
```

```
[1]  1  4  9 16 25
```

```r
a %*% a  # note, not element-wise!
```

```
     [,1]
[1,]   55
```

```r
a + 1:6  # problem: need same length
```

```
Warning in a + 1:6:  longer object length is not a multiple of shorter
object length
```

```
[1]  2  4  6  8 10  7
```

```r
a <- rbind(1:5, 2:6)  # same principles apply to matrices
b <- rbind(3:7, 4:8)
a + b
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    4    6    8   10   12
[2,]    6    8   10   12   14
```

```r
a/b
```

```
          [,1] [,2]      [,3]      [,4]      [,5]
[1,] 0.3333333  0.5 0.6000000 0.6666667 0.7142857
[2,] 0.5000000  0.6 0.6666667 0.7142857 0.7500000
```

```r
a %*% t(b)  # matrix multiplication
```

```
     [,1] [,2]
[1,]   85  100
[2,]  110  130
```

```r
# Logical operators (generally) work like arithmetic ones
a > 0
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] TRUE TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE TRUE
```

```r
a == b
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE
```

```r
a != b
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] TRUE TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE TRUE
```

```r
!(a == b)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] TRUE TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE TRUE
```

```r
(a > 2) | (b > 4)
```

```
      [,1]  [,2] [,3] [,4] [,5]
[1,] FALSE FALSE TRUE TRUE TRUE
[2,] FALSE  TRUE TRUE TRUE TRUE
```

```r
(a > 2) & (b > 4)
```

```
      [,1]  [,2] [,3] [,4] [,5]
[1,] FALSE FALSE TRUE TRUE TRUE
[2,] FALSE  TRUE TRUE TRUE TRUE
```

```r
(a > 2) || (b > 4)  # beware the 'double-pipe'!
```

```
[1] FALSE
```

```r
(a > 2) && (b > 4)  # (and the 'double-ampersand'!)
```

```
[1] FALSE
```

```r
# Ditto for many other basic transformations
log(a)
```

```
          [,1]      [,2]     [,3]     [,4]     [,5]
[1,] 0.0000000 0.6931472 1.098612 1.386294 1.609438
[2,] 0.6931472 1.0986123 1.386294 1.609438 1.791759
```

```r
exp(b)
```

```
         [,1]      [,2]     [,3]      [,4]     [,5]
[1,] 20.08554  54.59815 148.4132  403.4288 1096.633
[2,] 54.59815 148.41316 403.4288 1096.6332 2980.958
```

```
sqrt(a + b)  # note that we can nest statements!
```

```
        [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 2.00000 2.449490 2.828427 3.162278 3.464102
[2,] 2.44949 2.828427 3.162278 3.464102 3.741657
```

```
log((sqrt(a + b) + a) * b)  # as recursive as we wanna be
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 2.197225 2.879084 3.372185 3.760588 4.081744
[2,] 2.879084 3.372185 3.760588 4.081744 4.355853
```

Lists, data frames, and arrays

```
# R has many other data types.  One important type is the list.
a <- list(1:5)
a  # not an ordinary vector...
```

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

```
a <- list(1:5, letters[1:3])  # can we mix types and lengths?
a  # yes!
```

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

[[2]]
[1] "a" "b" "c"
```

```
b <- matrix(1:3, 3, 3)
a <- list(1:5, letters[1:3], b)  # anything can be stuffed in here
a
```

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

[[2]]
[1] "a" "b" "c"

[[3]]
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
```

```
a[[1]]  # retrieve the first item
```

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

```
a[[2]][3]  # the letter 'c'
```

```
[1] "c"
```

```
(a[[3]])[1, 3]  # it's really just recursion again
```

```
[1] 1
```

```
a <- list(boo = 1:4, hoo = 5)  # list elements are often named
names(a)  # get the element names
```

```
[1] "boo" "hoo"
```

```
a[["boo"]]  # ask for it by name
```

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

```
a$hoo  # use '£' to get what you want
```

```
[1] 5
```

```
# a+3 # whoops - not a vector!
a[[1]] + 3  # that works
```

```
[1] 4 5 6 7
```

```
a[[2]] <- a[[2]] * 4  # can also perform assignment
a$woo <- "glorp"  # works with '£'
a[["foo"]] <- "shazam"  # prolonging the magic
a
```

```
$boo
[1] 1 2 3 4

$hoo
[1] 20

$woo
[1] "glorp"

$foo
[1] "shazam"
```

```
# Another useful generalization: the data frame
d <- data.frame(income = 1:5, sane = c(T, T, T, T, F), name = LETTERS[1:5])  # Store multip
d
```

```
  income  sane name
1      1  TRUE    A
2      2  TRUE    B
3      3  TRUE    C
4      4  TRUE    D
5      5 FALSE    E
```

```
d[1, 2]  # acts a lot like a matrix!
```

```
[1] TRUE
```

```
d[, 1] * 5
```

```
[1]  5 10 15 20 25
```

```
d[-1, ]
```

```
  income  sane name
2      2  TRUE    B
3      3  TRUE    C
4      4  TRUE    D
5      5 FALSE    E
```

```
names(d)  # also acts like a list
```

```
[1] "income" "sane"   "name"
```

```
d[[2]]
```

```
[1]  TRUE  TRUE  TRUE  TRUE FALSE
```

```
d$sane[3] <- FALSE
d
```

```
  income  sane name
1      1  TRUE    A
2      2  TRUE    B
3      3 FALSE    C
4      4  TRUE    D
5      5 FALSE    E
```

```
d[2, 3]  # hmm - our data got factorized!
```

```
[1] B
Levels: A B C D E
```

```
d$name <- LETTERS[1:5]  # eliminate evil factors by overwriting
d[2, 3]
```

```
[1] "B"
```

```r
d <- data.frame(income = 1:5, sane = c(T, T, T, T, F), name = LETTERS[1:5],
    stringsAsFactors = FALSE)
d  # another way to fix it
```

```
  income  sane name
1      1  TRUE    A
2      2  TRUE    B
3      3  TRUE    C
4      4  TRUE    D
5      5 FALSE    E
```

```r
d <- as.data.frame(cbind(1:5, 2:6))  # can create from matrices
d
```

```
  V1 V2
1  1  2
2  2  3
3  3  4
4  4  5
5  5  6
```

```r
is.data.frame(d)  # how can we tell it's not a matrix?
```

```
[1] TRUE
```

```r
is.matrix(d)  # the truth comes out
```

```
[1] FALSE
```

```r
# When two dimensions are not enough: arrays
a <- array(1:18, dim = c(2, 3, 3))  # now in 3D
a
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

```
, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18
```

```r
a[1, 2, 3]  # selection works like a matrix
```

```
[1] 15
```

```r
a[1, 2, ]
```

```
[1]  3  9 15
```

```r
a[1, , ]
```

```
     [,1] [,2] [,3]
[1,]    1    7   13
[2,]    3    9   15
[3,]    5   11   17
```

```r
a[-1, 2:3, 1:2]
```

```
     [,1] [,2]
[1,]    4   10
[2,]    6   12
```

```r
a * 5  # ditto for element-wise operations
```

```
, , 1

     [,1] [,2] [,3]
[1,]    5   15   25
[2,]   10   20   30

, , 2

     [,1] [,2] [,3]
[1,]   35   45   55
[2,]   40   50   60

, , 3

     [,1] [,2] [,3]
[1,]   65   75   85
[2,]   70   80   90
```

```r
a <- array(dim = c(2, 3, 2, 5, 6))  # can have any number of dimensions
dim(a)  # find out what we've allocated
```

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

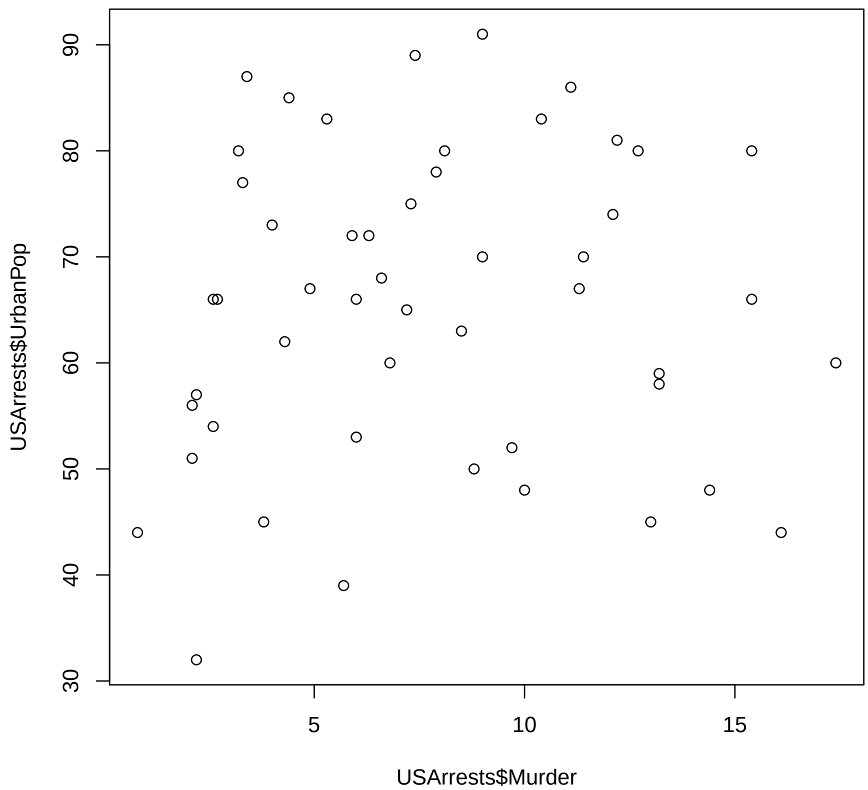
Finding built-in data sets

```r
# Many packages have built-in data for testing and educational
# purposes data() # list them all data(package='base') # all base
# package ?USArrests # get help on a data set
data(USArrests)  # load the data set
head(USArrests)  # view the object
```

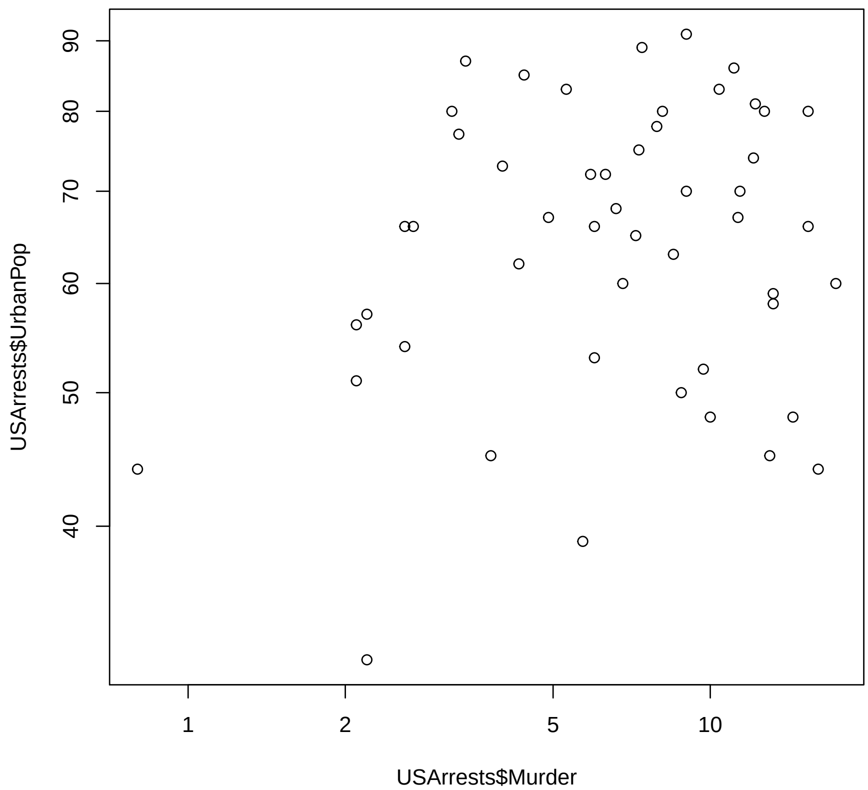
```
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

Elementary visualization

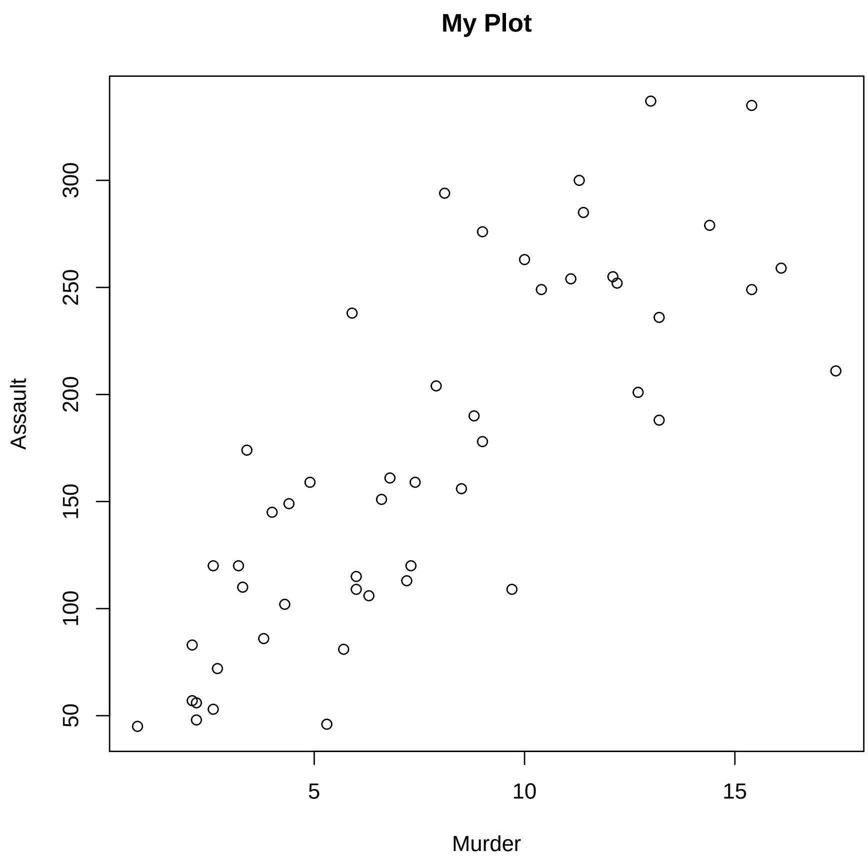
```r
# R's workhorse is the 'plot' command
plot(USArrests$Murder, USArrests$UrbanPop)
```



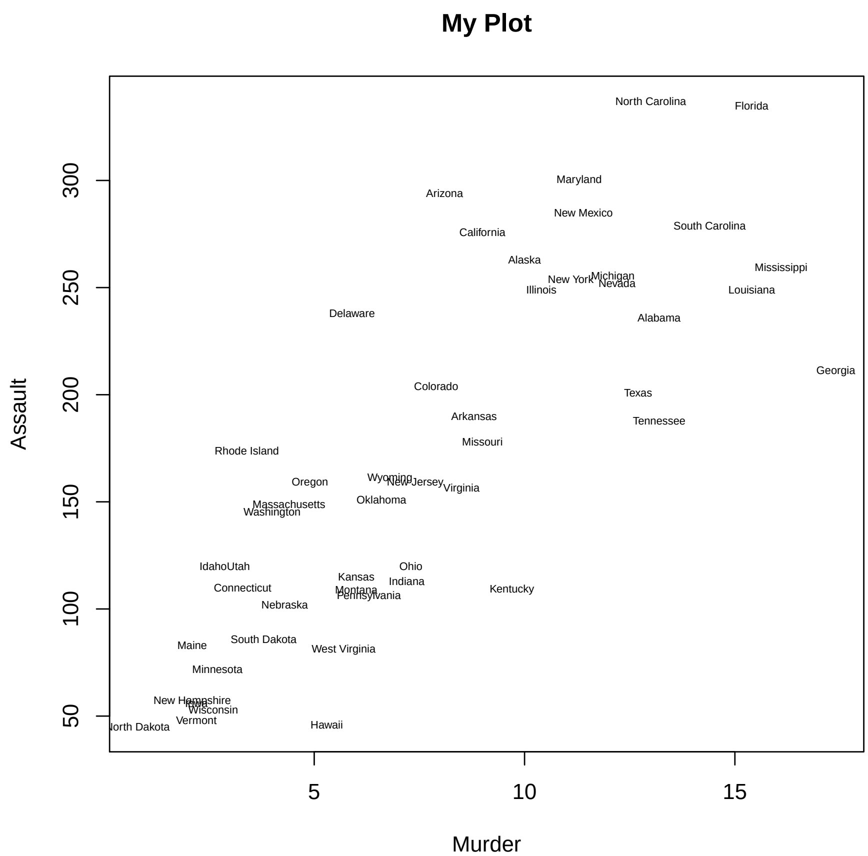

```r
plot(USArrests$Murder, USArrests$UrbanPop, log = "xy")  # log-log scale
```



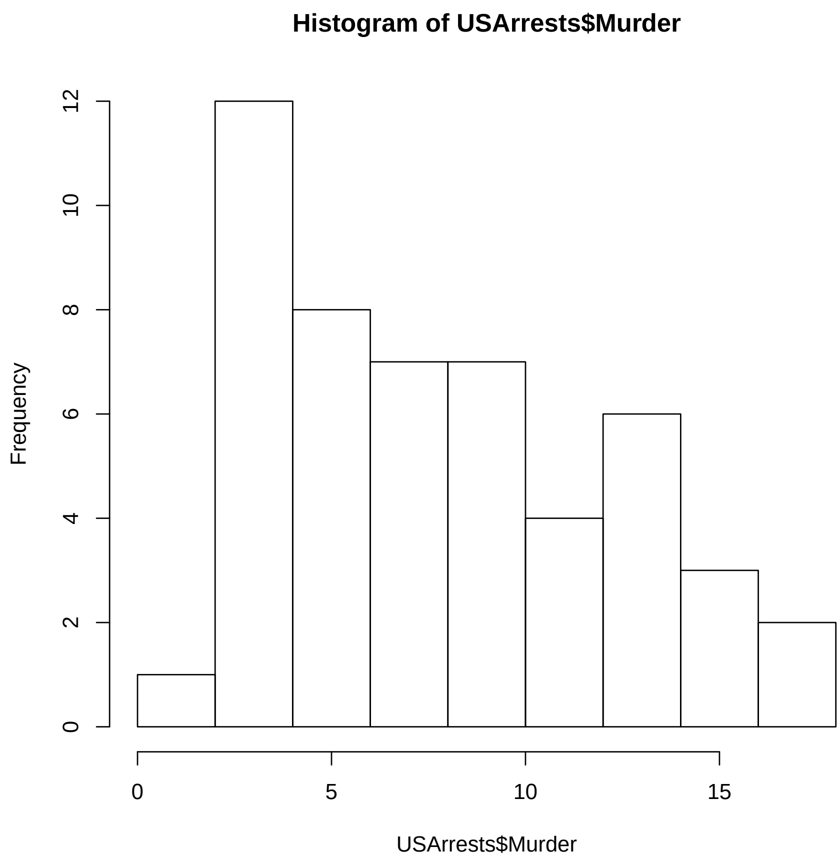

```r
plot(USArrests$Murder, USArrests$Assault, xlab = "Murder", ylab = "Assault",
    main = "My Plot")
```



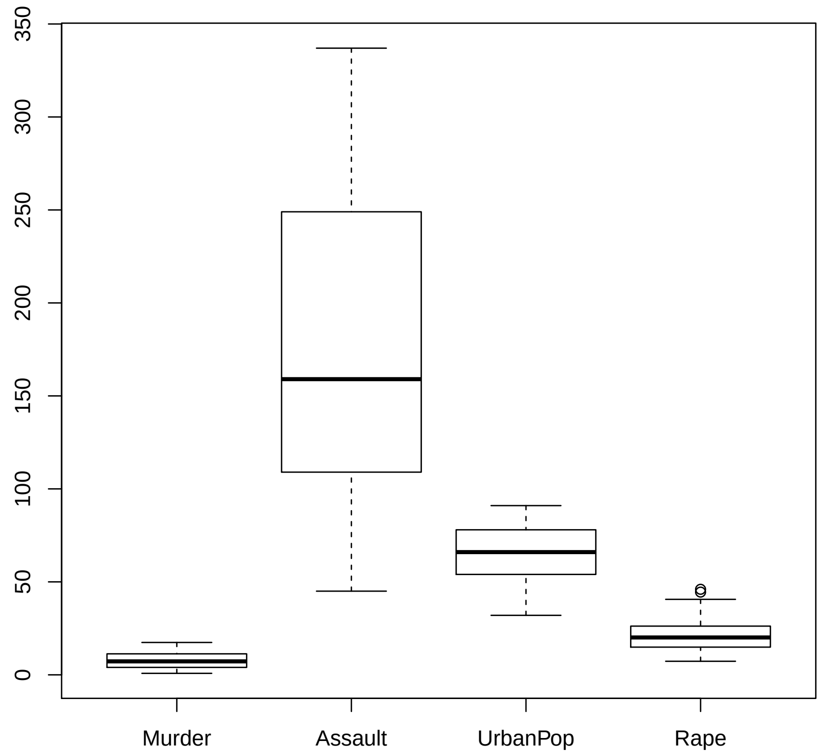

```r
# Can also add text
plot(USArrests$Murder, USArrests$Assault, xlab = "Murder", ylab = "Assault",
    main = "My Plot", type = "n")
text(USArrests$Murder, USArrests$Assault, rownames(USArrests), cex = 0.5)
```




```r
# Histograms and boxplots are often helpful
hist(USArrests$Murder)
```




```r
boxplot(USArrests)
```



Reading in data (and writing to disk) ** Not required for Lab 1**

```
# We won't use them right now, but here are some useful commands:
?read.table # a workhorse routine
?read.csv # a specialized CSV version
?scan # a more low-level variant
apropos("read") # list various "read" commands
?load # loads objects in native R format
?save # saves objects in native R format
?write.table # counterpart to read.table
apropos("write") # various "write" functions
```