IMT 573: Lab 1 - Introduction to R and RStudio Server

Your name

Date you are uploading

Objectives

In this demo we will get a first look at writing R code for data science. We will review basic R syntax and take a look at the different R data structures we will use throughout the course. We will learn how to find and access existing datasets, and even make our first visualization of data!

This demonstration also give you a glimpse of writing reproducible data science reports using Rmarkdown. We will talk more about this next week!

A little background on R

Everything in **R** is an object - data, functions, everything! When you type in commands what happens:

- **R** tries to interpret what you've asked it to do (evaluation)
- If it understands what you've told it to do, no problem
- If it does not understand, it will likely give you an error or warning message

Some commands trigger **R** to print something to the screen, others don't.

If you type in an incomplete command, **R** will usually respond by changing the command prompt to the + character to demonstrate it is waiting for something more. A > indicated the beginning of a line. You shouldn't have to consider this too much because you should always write your code in a *script* rather than the Console.

Getting started

Welcome to RStudio! RStudio is an integrated development environment (IDE) for R. It comes with a lot of nifty functionality to make it easier for us to do data science! Take a tour of RStudio using the online learning center or just play around with it after class today.

First, let's consider setting up our environment. In this report, we will be able to write text, as we have done already, but we will also be able to write code! Code should be contained in a *code chunk*. Code chunks are marked as follows:

```
# Hello R!
# This is a code comment.
# Code comments help you document your coding proces!

#This is a code chunk named "our first code chunk", which is the name of the code chunk;
# Including code chunk name is optional, but this practice will help you create well documented code
```

Introduction to basic R syntax

Let's take a look at some basic R syntax. Remember everything in R is an object! We also want to follow the `tidyverse` style guide for writing code. Variable and function names should use only lowercase letters, numbers, and _.

```
1 + 3       # evaluation
```

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

```
a <- 3  # assignment. <- is the assignment symbol
a               # evaluation
```

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

```
a<-3        # spacing does not matter
a    <-    3        # spacing does not matter

sqrt(a)         # use the square root function
```

```
## [1] 1.732051
```

```
b <- sqrt(a)        # use function and save result
b
```

```
## [1] 1.732051
```

```
# x             # evaluate something that is not there

a == b          # is a equal to b?
```

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

```
a != b          # is a not equal to b?
```

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

```
# list objects in the R environment
# (same as viewing the "Workspace" pane in RStudio)
ls()
```

```
## [1] "a" "b"
```

```
rm(a)           # remove a single object
rm(list=ls())       # remove everything from the environment
```

Getting help in R

```
# get help with R generally
# (same as viewing the "Help" pane in RStudio)
help.start()
```

```
## starting httpd help server ... done
```

```
## If the browser launched by '/usr/bin/open' is already running, it is
##     *not* restarted, and you must switch to its window.
## Otherwise, be patient ...
```

```
# More targeted help
?sqrt           # get specific help for a function
```

```r
apropos("sq")       # regular expression match. What do you do when you can't really recall the exact f
```

```
## [1] "chisq.test" "dchisq"     "pchisq"     "qchisq"     "rchisq"
## [6] "sqrt"       "sQuote"
```

Data Types in R

There are numerous data types in R that store various kinds of data. The four main types of data most likely to be used are numeric, character (string), Date (time-based) and logical (TRUE/FALSE).

```r
# Check the type of data contained in a variable with the class function.
x <- 3
class(x)
```

```
## [1] "numeric"
```

```r
# Numeric data type -- Testing whether a variable is numeric
is.numeric(x)
```

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

```r
is.numeric(3.3)
```

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

```r
# Character data
x <- "hello"
class(x)
```

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

```r
nchar(x)
```

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

```r
# Date type -- Dealing with dates and times can be difficult in any language, and to further complicate
date1 <- as.Date('2020-01-01')
class(date1)
```

```
## [1] "Date"
```

```r
# Logical data type: True/False
k <- TRUE
class(k)
```

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

```r
2 == 2
```

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

```r
# Factor vectors: Ideal for representing categorical variables (More on that later)
```

Vectors and matrices in R

Vectors in R: A collection of elements all of the same data type

```r
# Creating vectors using c() function or the "combine" operator
a <- c(1,3,5,7)
a
```

```
## [1] 1 3 5 7
```

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

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

```r
# also works with strings. let's see how.
b <- c("red","green","blue","purple")
b
```

```
## [1] "red"    "green"  "blue"   "purple"
```

```r
# return the second element
b[2]
```

```
## [1] "green"
```

```r
# return all elements except the second element
b[-2]
```

```
## [1] "red"    "blue"   "purple"
```

```r
# return 1st and 2nd element
b[c(1,2)]
```

```
## [1] "red"   "green"
```

```r
# all colors except blue
b[b != 'blue']
```

```
## [1] "red"    "green"  "purple"
```

```r
#all numbers less than 5
a[a < 5]
```

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

```r
#add a new element
b[5] <- "yellow"

#change the first element
b[1] <- "gold"

# combine by applying recursively
a <- c(a,a)
a
```

```
## [1] 1 3 5 7 1 3 5 7
```

```r
# mixing types---what happens?
a <- c(a,b)
a            # all converted to the same type
```

```
##  [1] "1"      "3"      "5"      "7"      "1"      "3"      "5"      "7"
##  [9] "gold"   "green"  "blue"   "purple" "yellow"
```

```r
# Sequences and replication

#creating a vector using sequence. sequence from 1 to 5
a <- seq(from=1,to=5,by=1)
b <- 1:5                    # a shortcut!
```

```r
#creating a vector using sequence. sequence from 1 to 10, steps of 2
a <- seq(from=1,to=10,by=2)

# replicate elements of a vector
rep(1,times=5)      # a lot of ones
```

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

```r
rep(1:5,times=2)    # repeat sequence 1 to 5, twice
```

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

```r
rep(1:5,each=2) # same as above, 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
```

```r
# Any, all, and which (with vectors)
a <- 1:5        # create a vector (sequence from 1 to 5)
a>2         # some TRUE, some FALSE
```

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

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

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

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

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

```r
which(a>2)  # which indices are TRUE?
```

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

```r
# How long is the vector?
length(a)
```

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

Element-wise operations on vectors

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

vectorized functions. Transforming vectors by applying functions

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

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

```r
exp(b)              # exponential function
```

```
## [1]   2.718282   7.389056  20.085537  54.598150 148.413159
```

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

```
## [1] 1.414214 2.000000 2.449490 2.828427 3.162278
```

```r
log((sqrt(a+b)+a)*b)     # more nesting
```

```
## [1] 0.8813736 2.0794415 2.7941343 3.3073887 3.7089612
```

```r
#vector of numbers
nums <- c(3.98, 8.2, 10.5, 3.6, 5.5)
round(nums, 1) # round to nearest whole number or number of decimal places, if specified
```

```
## [1]  4.0  8.2 10.5  3.6  5.5
```

From vectors (1D collection of data) to matrices (2D collection of data)

```r
# create a matrix the "formal" way...
a <- matrix(data=1:25, nrow=5, ncol=5) #fill by column

a <- matrix(data=1:25, nrow=5, ncol=5, byrow = TRUE) # fill by column
a
```

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

```r
a[1,2]      # select an element (specifying two dimensions)
```

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

```r
a[1,]       # just the first row
```

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

```r
a[,2]       # just the second column
```

```
## [1]  2  7 12 17 22
```

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

```
##      [,1] [,2] [,3]
## [1,]    8    9   10
## [2,]   13   14   15
```

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

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

```r
a[-2,-2]        # get rid of row two and column two
```

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

```r
# another way to create matrices (bind together column-wise)
b <- cbind(1:5,1:5)
b
```

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

```r
# can perform with rows, too (bind together row-wise)
d <- rbind(1:5,1:5)
d
```

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

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

```
## [1] 5 2
```

```r
dim(d)
```

```
## [1] 2 5
```

```r
nrow(b)      # how many rows in b?
```

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

```r
ncol(b)      # how many columns in b?
```

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

```r
cbind(b,b)  # combining two matrices, column-wise
```

```
##      [,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
```

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

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

Element-wise operations on matrices (same principles as vectors)

```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
# Logical operators (generally) work like
# arithmetic ones:
a > 0   # each value greater than zero?
```

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

```
## [1,] TRUE TRUE TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE TRUE TRUE
```

```r
a == b  # corresponding values equivalent?
```

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

```r
a != b  # corresponding values not equivalent?
```

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

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

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

Factor Variable

Factors consist of a finite set of categories (primarily used for categorical variables). Factors also optimize for space. Instead of storing each of the character strings, example 'small', 'medium', it will store a number and R will remember the relationship between the label and the string. Example: 1 for 'small', 2 for 'medium', etc. Let's see with an example

```r
# A character vector of shirt sizes
shirt_sizes <- c('small', 'medium', 'small', 'large', 'medium', 'medium')

# create a factor representation of the vector
ss <- as.factor(shirt_sizes)

# View factor and its levels
print(ss)
```

```
## [1] small  medium small  large  medium medium
## Levels: large medium small
```

```r
length(ss) #length of the factor is still the length of the vector
```

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

```r
length(shirt_sizes)
```

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

```r
# In ordinary factors, the order of the levels does not matter and one level is no different from anoth
# Sometimes, however, it is important to understand the order of a factor, such as when coding
ssf <- factor(shirt_sizes, ordered = TRUE)

# Another example of categorical variables where order is important
education <- factor(x = c("High School", "College", "Masters", "Doctorate"), ordered=TRUE)
education
```

```
## [1] High School College     Masters     Doctorate
## Levels: College < Doctorate < High School < Masters
```

R functions

```r
# call the sqrt() function, passing it an argument of 25
sqrt(25)
```

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

```r
# call the print function, passing it "IMT 573" as an argument
print("IMT 573")
```

```
## [1] "IMT 573"
```

```r
#printing using cat function
cat("value = ", a)
```

```
## value =  1 2 2 3 3 4 4 5 5 6
```

```r
#min and max function taking multiple arguments
min(5, 9, 2, 5)
```

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

```r
#function to return upper case
toupper('Seattle')
```

```
## [1] "SEATTLE"
```

```r
#Write a function of your name. Let's see how it works through an example
#write a function to combine first and last name

make_fullname <- function(firstname, lastname) {
  # function body
  fullname <- paste(firstname, lastname)

  #return the value
  return(fullname)
}

#call the function
some_name = make_fullname('John', 'Doe')
```

R functions: YOUR TURN - TODO

Write a function to calculate area of a rectangle

```r
rect_area <- function(width, height){
  return(width * height)
}

rect_area(10, 2)
```

```
## [1] 20
```

Data frames - act like tables where data is organized into rows and columns

```
# creating a dataframe by passing vectors to the `data.frame()` function

# a vector of names [INCLUDE CODE]
name <- c("Alice", "Bob", "Chris", "Diya", "Emma")
# A vector of heights
heights <- c(5.5, 6, 5.3, 5.8, 5.9)
weights <- c(100, 170, 150, 120, 155)

#Combine the vectors into a data frame
# Note the names of the variables become the names of the columns in the dataframe
people <- data.frame(name, heights, weights)

# to create row.names
people2 <- data.frame(name, heights, weights, row.names = 1)

# build an employee data frame of 5 employees with 3 columns: income, manager (T/F), empid
employee <- data.frame(income=101:105, manager=c(T,T,T,T,T),empid=LETTERS[1:5])

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

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

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

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

```
is.matrix(d)
```

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

```
# elements by row and column name
people2['Alice', 'heights']
```

```
## [1] 5.5
```

```
# elements by row and column indices
people2[1,1]
```

```
## [1] 5.5
```

```
people[1,1] # see the difference
```

```
## [1] "Alice"
```

```
#all elements in column or column names
people[, 2]
```

```
## [1] 5.5 6.0 5.3 5.8 5.9
```

```r
people[, 'heights']
```

```
## [1] 5.5 6.0 5.3 5.8 5.9
```

```r
#all elements in row or row names
people[1,]
```

```
##    name heights weights
## 1 Alice     5.5     100
```

```r
people2['Alice',]
```

```
##       heights weights
## Alice     5.5     100
```

```r
# can use dollar sign notation to extract columns
people$name
```

```
## [1] "Alice" "Bob"   "Chris" "Diya"  "Emma"
```

```r
# get 2 columns
people[,c('heights', 'weights')]
```

```
##   heights weights
## 1     5.5     100
## 2     6.0     170
## 3     5.3     150
## 4     5.8     120
## 5     5.9     155
```

```r
#get 2nd through 4th column
people[2:4, ]
```

```
##    name heights weights
## 2   Bob     6.0     170
## 3 Chris     5.3     150
## 4  Diya     5.8     120
```

```r
# get all rows where people height is > 5.5 and all columns
people[people$heights > 5.5, ]
```

```
##   name heights weights
## 2  Bob     6.0     170
## 4 Diya     5.8     120
## 5 Emma     5.9     155
```

```r
# make changes by extracting values. Chris's height is actually 6ft. and not 5.3
people$heights[3] <- 6
```

Read and Write data

Working Directory When working with .csv files, the read.csv() function takes as an argument a path to a file. You need to make sure you have the correct path. To check your current working directory using the R function **getwd()**

```r
#read titanic.csv data. Download data from Canvas
titanic_dataset <- read.csv("../../data/titanic.csv")
```

```r
#check the type of data
class(titanic_dataset)
```

```
## [1] "data.frame"
```

```r
#check additional structure and type in the data
str(titanic_dataset) #str display the internal structure of the data allows you to check the classes of
```

```
## 'data.frame':    1309 obs. of  14 variables:
##  $ pclass   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ survived : int  1 1 0 0 0 1 1 0 1 0 ...
##  $ name     : chr  "Allen, Miss. Elisabeth Walton" "Allison, Master. Hudson Trevor" "Allison, Miss. 
##  $ sex      : chr  "female" "male" "female" "male" ...
##  $ age      : num  29 0.917 2 30 25 ...
##  $ sibsp    : int  0 1 1 1 1 0 1 0 2 0 ...
##  $ parch    : int  0 2 2 2 2 0 0 0 0 0 ...
##  $ ticket   : chr  "24160" "113781" "113781" "113781" ...
##  $ fare     : num  211 152 152 152 152 ...
##  $ cabin    : chr  "B5" "C22 C26" "C22 C26" "C22 C26" ...
##  $ embarked : chr  "S" "S" "S" "S" ...
##  $ boat     : chr  "2" "11" "" "" ...
##  $ body     : int  NA NA NA 135 NA NA NA NA NA 22 ...
##  $ home.dest: chr  "St Louis, MO" "Montreal, PQ / Chesterville, ON" "Montreal, PQ / Chesterville, ON
```

```r
# inspect the data - look at top and bottom
head(titanic_dataset)
```

```
##   pclass survived                                            name    sex
## 1      1        1                   Allen, Miss. Elisabeth Walton female
## 2      1        1                  Allison, Master. Hudson Trevor   male
## 3      1        0                    Allison, Miss. Helen Loraine female
## 4      1        0            Allison, Mr. Hudson Joshua Creighton   male
## 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6      1        1                             Anderson, Mr. Harry   male
##       age sibsp parch ticket     fare   cabin embarked boat body
## 1 29.0000     0     0  24160 211.3375      B5        S    2   NA
## 2  0.9167     1     2 113781 151.5500 C22 C26        S   11   NA
## 3  2.0000     1     2 113781 151.5500 C22 C26        S        NA
## 4 30.0000     1     2 113781 151.5500 C22 C26        S       135
## 5 25.0000     1     2 113781 151.5500 C22 C26        S        NA
## 6 48.0000     0     0  19952  26.5500     E12        S    3   NA
##                         home.dest
## 1                    St Louis, MO
## 2 Montreal, PQ / Chesterville, ON
## 3 Montreal, PQ / Chesterville, ON
## 4 Montreal, PQ / Chesterville, ON
## 5 Montreal, PQ / Chesterville, ON
## 6                    New York, NY
```

```r
tail(titanic_dataset)
```

```
##      pclass survived                      name    sex  age sibsp parch ticket
## 1304      3        0     Youssef, Mr. Gerious   male   NA     0     0   2627
## 1305      3        0      Zabour, Miss. Hileni female 14.5     1     0   2665
## 1306      3        0     Zabour, Miss. Thamine female   NA     1     0   2665
## 1307      3        0 Zakarian, Mr. Mapriededer   male 26.5     0     0   2656
```

```
## 1308      3        0       Zakarian, Mr. Ortin   male 27.0     0     0   2670
## 1309      3        0        Zimmerman, Mr. Leo   male 29.0     0     0 315082
##         fare cabin embarked boat body home.dest
## 1304 14.4583              C        NA
## 1305 14.4542              C       328
## 1306 14.4542              C        NA
## 1307  7.2250              C       304
## 1308  7.2250              C        NA
## 1309  7.8750              S        NA
```

Finding built-in data sets

```r
# Many packages have built-in data for testing
# and educational purposes:
data()          # lists them all
?USArrests      # get help on a data set
data(USArrests) # load the data set
USArrests           # view the object
```

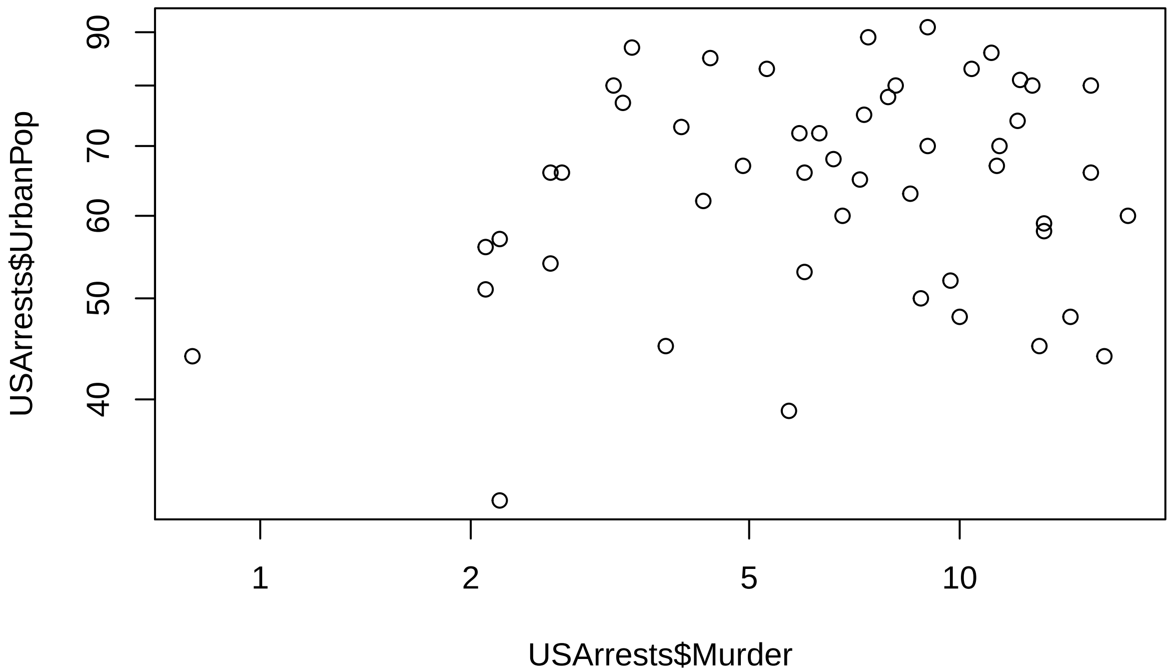
```
##                Murder Assault UrbanPop Rape
## Alabama          13.2     236       58 21.2
## Alaska           10.0     263       48 44.5
## Arizona           8.1     294       80 31.0
## Arkansas          8.8     190       50 19.5
## California        9.0     276       91 40.6
## Colorado          7.9     204       78 38.7
## Connecticut       3.3     110       77 11.1
## Delaware          5.9     238       72 15.8
## Florida          15.4     335       80 31.9
## Georgia          17.4     211       60 25.8
## Hawaii            5.3      46       83 20.2
## Idaho             2.6     120       54 14.2
## Illinois         10.4     249       83 24.0
## Indiana           7.2     113       65 21.0
## Iowa              2.2      56       57 11.3
## Kansas            6.0     115       66 18.0
## Kentucky          9.7     109       52 16.3
## Louisiana        15.4     249       66 22.2
## Maine             2.1      83       51  7.8
## Maryland         11.3     300       67 27.8
## Massachusetts     4.4     149       85 16.3
## Michigan         12.1     255       74 35.1
## Minnesota         2.7      72       66 14.9
## Mississippi      16.1     259       44 17.1
## Missouri          9.0     178       70 28.2
## Montana           6.0     109       53 16.4
## Nebraska          4.3     102       62 16.5
## Nevada           12.2     252       81 46.0
## New Hampshire     2.1      57       56  9.5
## New Jersey        7.4     159       89 18.8
## New Mexico       11.4     285       70 32.1
## New York         11.1     254       86 26.1
## North Carolina   13.0     337       45 16.1
```

```
## North Dakota       0.8      45        44  7.3
## Ohio               7.3     120        75 21.4
## Oklahoma           6.6     151        68 20.0
## Oregon             4.9     159        67 29.3
## Pennsylvania       6.3     106        72 14.9
## Rhode Island       3.4     174        87  8.3
## South Carolina    14.4     279        48 22.5
## South Dakota       3.8      86        45 12.8
## Tennessee         13.2     188        59 26.9
## Texas             12.7     201        80 25.5
## Utah               3.2     120        80 22.9
## Vermont            2.2      48        32 11.2
## Virginia           8.5     156        63 20.7
## Washington         4.0     145        73 26.2
## West Virginia      5.7      81        39  9.3
## Wisconsin          2.6      53        66 10.8
## Wyoming            6.8     161        60 15.6
```

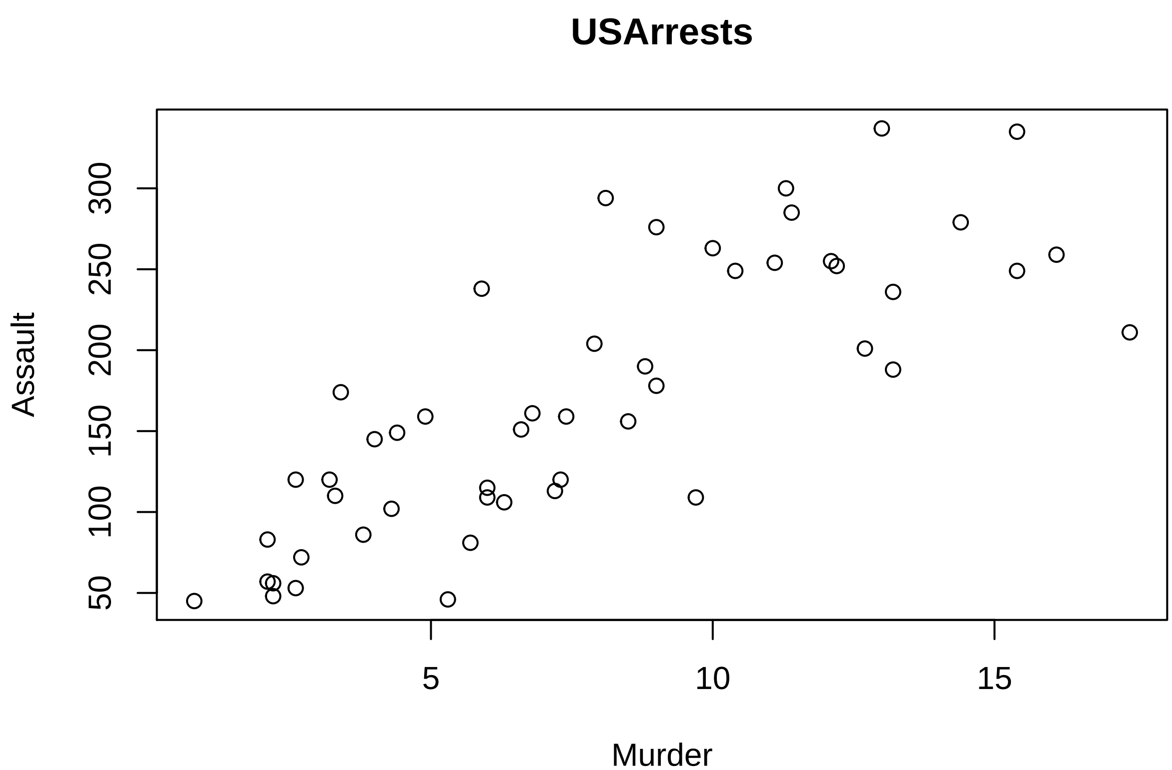
Elementary visualization

```
# R's graphics workhorse is the "plot" command:
plot(x=USArrests$Murder,y=USArrests$UrbanPop)
```


```
# Same as above, but now on log-log scale
plot(x=USArrests$Murder,y=USArrests$UrbanPop,log="xy")
```

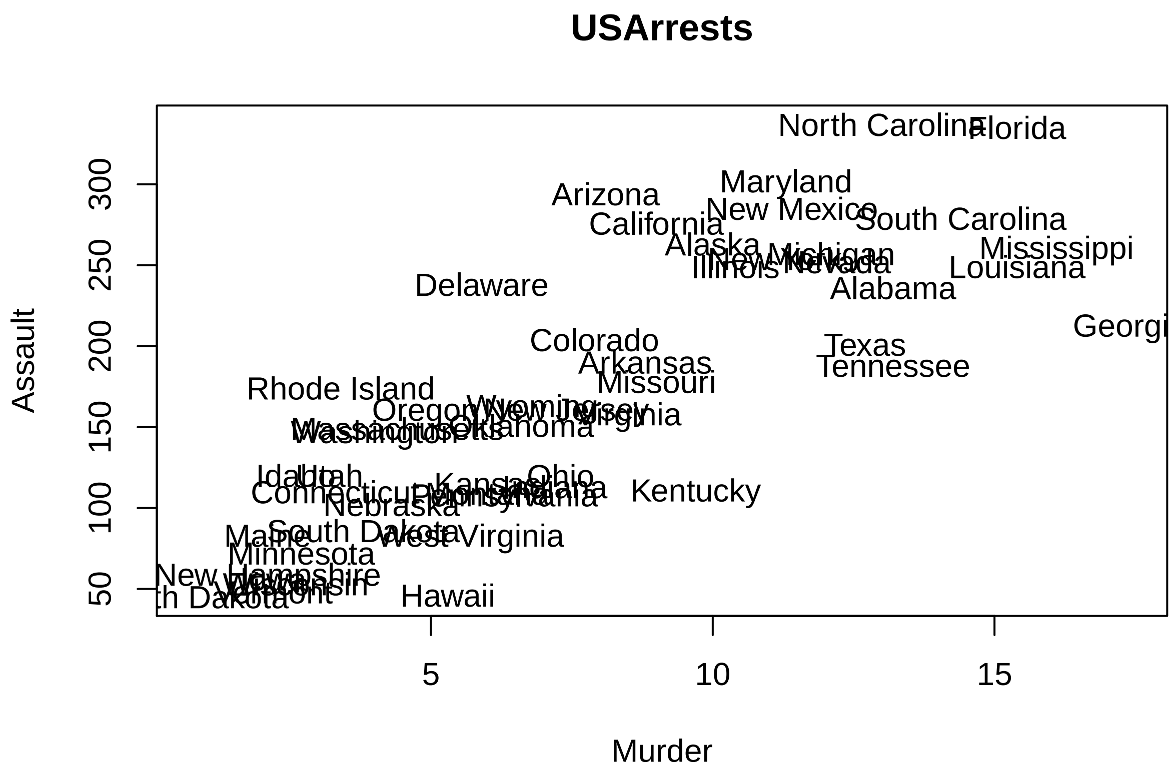



```
# Adding plot title and clean up axis labels
plot(x=USArrests$Murder,y=USArrests$Assault,xlab="Murder",ylab="Assault",main="USArrests")
```




```
# Can also add text to a plot:

# Step 1: set up a "blank" plot window
# Step 2: add in text
plot(x=USArrests$Murder,y=USArrests$Assault,xlab="Murder",ylab="Assault", main="USArrests",type="n")
text(x=USArrests$Murder,y=USArrests$Assault,labels=rownames(USArrests))
```




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

```r
boxplot(USArrests)
```