ISSR Short Course

Gregory J. Matthews [1]

[1]Department of Mathematics and Statistics
Loyola University Chicago

June 2015

Outline

What is R?

- R is a language and environment for statistical computing and graphics.
- Website: www.r-project.org

Pros

- R is free.
- There are many packages for R. Chances are someone has written it already.
- Many cutting edge techniques are available very quickly.

Cons

- R is free. This means there is no support for R.
- This also means that you use a package at your own risk and you trust the author wrote the code correctly.
- Takes a little while to learn.

R Studio

What is Rstudio^{TM}?

- RStudio? is a free and open source integrated development environment (IDE) for R.
- rstudio.org

Rstudio

- When you open up Rstudio the screen is divided into 4 parts
  - Source (Upper Left)
  - Console (Lower Left)
  - Workspace/History (Upper Right)
  - Files/Plots/Packages/**Help**
- 'Help' is your best friend. Every function in R has a help file.
- You can either search for a function (lm) or type: help(lm). Both take you to the same place. (lower right of R studio)
- ls() lists all the current object (Upper right of R studio)

Source

Upper Left box

- This is where we will type R code.
- To run R code click the "Run" button or type "Command-Enter"

Console

Lower Left box

- This displays the code that has run and the output.
- We can either run code from the source box or by typing directly into the console

Workspace/History

Upper Right box

- The workspace tab displays all of the R objects that have been created in this session and describes them
- we can view the data in the source window by clicking to the right of the appropriate row in the workspace window
- we can also view the data in the console by typing the name of the object that we are interested in viewing and hitting enter.
- the history tab displays all of the commands that have been run in the console

Files/Plots/Packages/Help

Lower Right box

- Files: Allows you to look for files. This is like Windows Explorer (or Finder, for Mac Users)
- Plots: Manages all of your plots that are created.
- Packages: Functions come in packages and you need to install and load the correct package to use a function. This tab allows you to manage which packages are loaded.
- **Help**: If you don't understand how to use a function, this allows you to get help (Can also get here by typing "help(lm)" which will give you the help file for the function "lm" used for linear modeling).

Some Notes

- R is case sensitive (R≠r)
- To comment out a command we use "#"
- If you can't figure out what something is *str* is a useful function
- Every function has a help page (?function or help(function))
- To install a package use install.package('package.name')
- To load a package use library(package.name)
- getwd() and setwd()

- Vectors consist of a series of elements
- To assign elements to a vector use:

  `<- or -> or =`
- Vectors have one index, which can be used to call specific elements from the vectors

```r
#Vectors
 #Assign like this
xVec<-c(4,1,3,8,6,7,5,3,0,9)
 #Or like this
 c(4,1,3,8,6,7,5,3,0,9)->xVec
#What is xVec?
 str(xVec)
```

```
##  num [1:10] 4 1 3 8 6 7 5 3 0 9
```

```r
 #I can look at specific element of the vector
 #only the first element
 xVec[1]
```

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

```r
 #the first five elements
 xVec[1:5]
```

```
## [1] 4 1 3 8 6
```

```r
 #Vectors can be added together
 #Vector Operations
 xVec
```

```
##  [1] 4 1 3 8 6 7 5 3 0 9
```

```r
 yVec<-c(1:10)
 yVec
```

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

```r
 #Add the two vectors together
 xVec+yVec
```

```
##  [1]  5  3  6 12 11 13 12 11  9 19
```

```r
 #Add 1 to every element
 xVec+1
```

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

```r
#Take the reciprocal of all elements
#Notice how R handles infinity
round(1/xVec,4)[5:10]
```

```
## [1] 0.1667 0.1429 0.2000 0.3333    Inf 0.1111
```

```r
#Vectors will repeat themselves
c(1:3)+c(1:6)
```

```
## [1] 2 4 6 5 7 9
```

```r
#However, they must be multiples
c(1:3)+c(1:5)
```

```
## Warning in c(1:3) + c(1:5): longer object length
is not a multiple of shorter object length
```

```
## [1] 2 4 6 5 7
```

- "=" assigns: x=3 means the values of "x" is 3
- "==" test: x==3 tests the elements of x to see if they meet this condition
- This will return TRUE or FALSE (or NA)
- Other logical symbols include:
  - "<" less than
  - ">" greater than
  - "<=" less than or equal
  - ">=" greater than or equal
  - "! =" not equal
- These can be combined by using "&" and "|" as "and" and "or", respectively.

```r
xVec<-c(4,1,3,8,6,7,5,3,0,9)
xVec==0
```

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

```r
(xVec==0)+0
```

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

```r
sum(xVec==0)
```

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

```r
xVec>5 | xVec==0
```

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

```r
#Pull out all of the elements that are greater than 5 or equal to 0.
xVec[xVec>5 | xVec==0]
```

```
## [1] 8 6 7 0 9
```

How does R deal with missing values?

- R uses "NA" to indicate missing values.
- R also uses "NaN" to indicate not a number.

```r
#Missing Values
xVec<-c(4,1,3,8,6)
xVec[1]<-NA
xVec
```

```
## [1] NA  1  3  8  6
```

```r
 ##Which values in xVec are missing?
 is.na(xVec)
```

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

```r
 ##Not the same as this:
 xVec==NA
```

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

```r
 ##There is also NaN
 xVec[2]<-0/0
 xVec
```

```
## [1]  NA NaN   3   8   6
```

```r
 ##NA and NaN are both treated as missing in is.na()
 is.na(xVec)
```

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

```r
 is.nan(xVec)
```

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

```r
#Is infinity missing?
 is.na(Inf)
```

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

```r
 ##Useful vector functions
 ##Creates a vector of numbers from 1 to 10 by 2.
 seq(1,10,2)
```

```
## [1] 1 3 5 7 9
```

```r
 ##Creates a vector of equally spaced numbers
 ##between 1 and 10 of length 5.
 seq(1,10,length=5)
```

```
## [1]  1.00  3.25  5.50  7.75 10.00
```

```r
 ##Creates a vector of zeroes with 5 elements
 rep(0,5)
```

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

```r
 ##Using both rep() and seq()
 c(rep(seq(1,10,2),2))
```

- So far, everything has been numeric in the vectors we have seen.
- However, other types of data are also important.
  - characters
  - factors
- Vectors can also consist of elements of both of these types
- Assign a character string to a vectors as before with the function c()
- Either single or double quotation marks are used to denote character strings

```r
 ##character vectors
 xChar<-c("A","A","B","C","A","B")
 #Still acts as a vector
 xChar[1]
```

```
## [1] "A"
```

```r
 xChar[1:4]
```

```
## [1] "A" "A" "B" "C"
```

```r
 #But I can't add
 xChar+1
```

```
## Error in xChar + 1:  non-numeric argument to
binary operator
```

```r
##character vectors
 xFact<-c(1,1,2,3,1,2)

 ##numeric
 str(xFact)

##  num [1:6] 1 1 2 3 1 2

 ##character
 str(xFact)

##  num [1:6] 1 1 2 3 1 2
```

```r
xChar<-c("1","1","2","3","1","2")
#Still can't add number to X even
#though its elements are "numbers"
 #X+1


 ##I can do this
 as.numeric(xChar)+1
```

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

```r
 ##Same as this
 xVec+1
```

```
## [1]  NA NaN   4   9   7
```

```r
 #Factors
 #Create a factor
 xFact<-factor(c("A","A","B","C","A","B"))
 xFact
```

```
## [1] A A B C A B
## Levels: A B C
```

```r
 #Display the factor levels
 levels(xFact)
```

```
## [1] "A" "B" "C"
```

```r
 #Coerced to numeric
 as.numeric(xFact)
```

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

```r
 #Create another factor
 yFact<-factor(c(100,100,200,300,100,200))
 yFact
```

```
## [1] 100 100 200 300 100 200
## Levels: 100 200 300
```

```r
 #Levels are different
 levels(yFact)
```

```
## [1] "100" "200" "300"
```

```r
 #Coerced to numeric
 as.numeric(yFact)
```

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

```r
#CAREFUL HERE!
#If you wanted yFact to be numeric, you won't
#get it by using as.numeric.
#You need to do:
 as.numeric(as.character(yFact))

## [1] 100 100 200 300 100 200
```

- paste() is a useful function when dealing with characters
- Use this function to join multiple character strings.

```r
 ##paste function
 ##Character vector
 xChar<-c("X1","X2","X3","X4","X5")
 xChar
```

```
## [1] "X1" "X2" "X3" "X4" "X5"
```

```r
 ##Same vector but simpler
 xVec<-paste("X",c(1:5),sep="")
 xVec
```

```
## [1] "X1" "X2" "X3" "X4" "X5"
```

```r
 ##paste function recycles values
 xVec<-paste("X",c(1:5),c(1:10),sep="-")
 xVec
```

```
##  [1] "X-1-1"  "X-2-2"  "X-3-3"  "X-4-4"  "X-5-5"  "X-1-6"  "X
##  [8] "X-3-8"  "X-4-9"  "X-5-10"
```

Some useful functions for test analysis.

- grep and grepl
- gregexpr
- gsub

```
(txtVec<-c("The","quick","brown","fox",
           "jumped","over","the","lazy","dog"))

## [1] "The"    "quick"  "brown"  "fox"    "jumped" "over"
## [9] "dog"

#returns index vector
grep("o",txtVec)

## [1] 3 4 6 9

#Returns logical vector
grepl("o",txtVec)

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

```r
(txtVec<-c("The","quick","brown","fox",
           "jumped","over","the","Lazy","dog"))
```

```
## [1] "The"    "quick"  "brown"  "fox"    "jumped" "over"
## [9] "dog"
```

```r
#Single letter searches
grep("[A-z]",txtVec)
```

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

```r
grep("[A,z]",txtVec)
```

```
## [1] 8
```

```r
grep("[A-Z,z]",txtVec)
```

```
## [1] 1 8
```

```
(txtVec<-c("The","quick","brown","fox",
           "jumped","over","the","Lazy","dog"))

## [1] "The"    "quick"  "brown"  "fox"    "jumped" "over"
## [9] "dog"

#Single letter searches
grep("The|the",txtVec)

## [1] 1 7

grep("(T|t)he",txtVec)

## [1] 1 7

grep("[aeiou][aeiou]",txtVec)

## [1] 2
```

- ?: zero or one of the preceding element
- +: one or more of the preceding element
- *: zero or more of the preceding element

```
(txtVec<-c("Greg","Gregg","Greory","Grigor",
           "Gregggory","Greggory","Gregory","Gregor"))
```

```
## [1] "Greg"      "Gregg"     "Greory"    "Grigor"    "Gre
## [7] "Gregory"   "Gregor"
```

```
grep("Greg",txtVec)
```

```
## [1] 1 2 5 6 7 8
```

```
grep("Gregg?ory",txtVec)
```

```
## [1] 6 7
```

```
grep("Greg*ory",txtVec)
```

```
## [1] 3 5 6 7
```

```
(txtVec<-c("Greg","Gregg","Greory","Grigor",
           "Gregggory","Greggory","Gregory","Gregor"))

## [1] "Greg"      "Gregg"     "Greory"    "Grigor"    "Gre
## [7] "Gregory"   "Gregor"

grep("Greg+ory",txtVec)

## [1] 5 6 7

grep("Gr.g",txtVec)

## [1] 1 2 4 5 6 7 8
```

```
(txtVec<-c("Greg","Gregg","Greory","Grigor",
           "Gregggory","Greggory","Gregory","Gregor"))

## [1] "Greg"      "Gregg"     "Greory"    "Grigor"    "Gre
## [7] "Gregory"   "Gregor"

grep(".*gg.*",txtVec)

## [1] 2 5 6

grep("Gr.*ory",txtVec)

## [1] 3 5 6 7

grep("Gr.*gg+.*r",txtVec)

## [1] 5 6
```

```
txtVec<-c("The","quick","brown","fox","jumped","over","the"
#gregexpr returns a list
gregexpr("o",txtVec)[[2]]

## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"useBytes")
## [1] TRUE

gregexpr("o",txtVec)[[3]]

## [1] 3
## attr(,"match.length")
## [1] 1
## attr(,"useBytes")
## [1] TRUE
```

```r
txtVec<-c("The","quick","brown","fox","jumped","over","the"
#gsub replaces a character string
gsub("o",":)",txtVec)

## [1] "The"    "quick"  "br:)wn" "f:)x"   "jumped" ":)ver"
## [9] "d:)g"
```

```
(txtVec<-c("Greg","Gregg","Greory","Grigor",
           "Gregggory","Greggory","Gregory","Gregor"))

## [1] "Greg"      "Gregg"     "Greory"    "Grigor"    "Gre
## [7] "Gregory"   "Gregor"

gsub("Greg+","Greg",txtVec)

## [1] "Greg"    "Greg"    "Greory"  "Grigor"  "Gregory" "(
## [8] "Gregor"

gsub("Gr.g+or.?","Gregory",txtVec)

## [1] "Greg"    "Gregg"   "Greory"  "Gregory" "Gregory" "(
## [8] "Gregory"
```

- How can we subset vectors?
- May want to remove certain elements from the vector.

```r
##Subsetting a vector
 X<-c(1:10,NA,NA)
 X
```

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

```r
 #Pull out first element
 X[1]
```

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

```r
 #Use logical statements to subset the vectors
 #Pull out elements that are <3 or >8
 X[X<3 | X>8]
```

```
## [1]  1  2  9 10 NA NA
```

```r
 #pull out only the non-missing values
 X[!is.na(X)]
```

```
(txtVec<-c("The","quick","brown","fox","jumped","over","the

## [1] "The"    "quick"  "brown"  "fox"    "jumped" "over"
## [9] "dog"

#Both work here
txtVec[grep("o",txtVec)]

## [1] "brown" "fox"   "over"  "dog"

txtVec[grepl("o",txtVec)]

## [1] "brown" "fox"   "over"  "dog"
```

- Matrices have two indices
  - First index is the row index
  - Second index is the column index
- Arrays are more general and can have many indices
- So matrices are really two dimensional arrays

```r
##Matrices
##By default matrices are populated by row
xMat<-matrix(c(1:9),ncol=3)
xMat
```

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

```r
##To input by row
xMat<-matrix(c(1:9),ncol=3,byrow=TRUE)
xMat
```

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

```r
 ##Pull out elements in same way as with vectors
 #Now we need to specify a row and and column
 xMat[1,1]
```

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

```r
 ##All of the first row
 xMat[1,]
```

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

```r
 ##All of the first column
 xMat[,1]
```

```
## [1] 1 4 7
```

```r
##First and second row with the first and second column
 xMat[c(1,2),1:2]

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

##First and third row with the second column removed
 xMat[c(1,3),-c(2)]

##      [,1] [,2]
## [1,]    1    3
## [2,]    7    9
```

cbind() and rbind()

- cbind allows us to join two matrices by placing them next to one another.
- rbind allows us to join two matrices together by placing one on top of the other.

```r
mat1<-matrix(1,ncol=2,nrow=2)
mat2<-matrix(2,ncol=2,nrow=2)
mat3<-matrix(3,ncol=2,nrow=2)
#cbind
 cbind(mat1,mat2,mat3)
```

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

```r
 rbind(mat1,mat2,mat3)
```

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

```
 wide<-cbind(mat1,mat2,mat3)
 tall<-rbind(mat1,mat2,mat3)
 rbind(wide,cbind(tall,tall,tall))
```

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

Lists

- Lists are very flexible R objects
- The items of a list can be of any class including many different classes within one R list
- Lists are said to be recursive because an element of a list could potentially be a list itself
- Lists are indexed with double brackets "[[]]" or with a "$name"

```r
##Lists
##Create a list
xList<-list(3,rep(0,3),matrix(c(1:4),
        ncol=2),paste("X",c(1:5),sep=""))
xList
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 0 0 0
##
## [[3]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## [[4]]
## [1] "X1" "X2" "X3" "X4" "X5"
```

```r
##Create a list
xList<-list()
xList[[1]]<-3
xList[[2]]<-rep(0,3)
xList$three<-matrix(c(1:4),ncol=2)
xList$four<-paste("X",c(1:5),sep="")
```

```
 xList

## [[1]]
## [1] 3
##
## [[2]]
## [1] 0 0 0
##
## $three
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $four
## [1] "X1" "X2" "X3" "X4" "X5"
```

```r
 names(xList)
```

```
## [1] ""      ""      "three" "four"
```

```r
 ##I want to assign names to the first two elements
 names(xList)[1:2]<-c("one","two")
 names(xList)
```

```
## [1] "one"   "two"   "three" "four"
```

```r
 str(xList)
```

```
## List of 4
##  $ one  : num 3
##  $ two  : num [1:3] 0 0 0
##  $ three: int [1:2, 1:2] 1 2 3 4
##  $ four : chr [1:5] "X1" "X2" "X3" "X4" ...
```

```r
 length(xList)
```

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

```r
##Call the third element is two ways:
##By index
xList[[3]]

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

##By name
xList$three

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

```r
##Lists lists
xList<-list(list(3,rep(0,3)),list(matrix(c(1:4),
  ncol=2),paste("X",c(1:5),sep="")))
xList[[1]]

## [[1]]
## [1] 3
##
## [[2]]
## [1] 0 0 0
```

```
xList[[2]]

## [[1]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## [[2]]
## [1] "X1" "X2" "X3" "X4" "X5"
```

R: data frame

- Data frames are just lists where each element has to be a vector of the same length.
- Much of the data that we are interested in can be easily analyzed as a data frame.
- Data frames have many of the properties of matrices.
  - Subset like a matrix
    - X[,1] - Display first column
    - X[1,] - Display first row
  - Or by variable name
    - X$V1 - Display variable "V1"

```r
#data.frames
#Create a simple data frame
xDF<-data.frame(V1=c(1:10),V2=rep(1,10),V3=seq(1,20,2),
V4=c(rep("A",3),rep("B",7)),V5=rnorm(10,0,5),V6=xVec)
str(xDF)

## 'data.frame': 10 obs. of  6 variables:
##  $ V1: int  1 2 3 4 5 6 7 8 9 10
##  $ V2: num  1 1 1 1 1 1 1 1 1 1
##  $ V3: num  1 3 5 7 9 11 13 15 17 19
##  $ V4: Factor w/ 2 levels "A","B": 1 1 1 2 2 2 2 2 2 2
##  $ V5: num  4.74 3.08 3.71 -10.16 4.63 ...
##  $ V6: Factor w/ 10 levels "X-1-1","X-1-6",..: 1 3 5 7 1
```

```
names(xDF)

## [1] "V1" "V2" "V3" "V4" "V5" "V6"

colnames(xDF)

## [1] "V1" "V2" "V3" "V4" "V5" "V6"

rownames(xDF)

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

```r
##Dataframes have many of the properties of a matrix.
#Only First column
xDF[,1]

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

#Only first row
xDF[1,]

##   V1 V2 V3 V4       V5     V6
## 1  1  1  1  A 4.735175 X-1-1
```

```r
#everything but the first three rows
xDF[-c(1:3),]
```

```
##    V1 V2 V3 V4          V5     V6
## 4   4  1  7  B -10.164791  X-4-4
## 5   5  1  9  B   4.630954  X-5-5
## 6   6  1 11  B  -3.435143  X-1-6
## 7   7  1 13  B   8.328872  X-2-7
## 8   8  1 15  B  -4.581561  X-3-8
## 9   9  1 17  B   2.758271  X-4-9
## 10 10  1 19  B  -3.973927 X-5-10
```

```r
#Rows 3,6, and 8, columns 2, 4
xDF[c(3,6,8),c(2,4)]
```

```
##   V2 V4
## 3  1  A
## 6  1  B
## 8  1  B
```

```r
#Can't look at V1 directly
#V1
#Doesn't work, but.....this will
xDF$V1
```

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

```r
#I could also use attach() so I don't
#have to type xDF£V1 every single time
attach(xDF)

 #And now
 V1
```

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

```r
 xDF$V1
```

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

R: data frame

- Observations (rows) can be added to a data frame by using rbind()
- Variables (columns) can be added to a data frame by using cbind()
- Columns can also be added as follows:

```
x$three<-x$y+x$z
```

- This creates a new variable three which is the sum of variables y and z which are contained in the data frame x

```r
#Add a row
rbind(xDF,c(11,1,21,'B',6.5,'X-1-1'))
```

```
##     V1 V2 V3 V4                  V5      V6
## 1    1  1  1  A 0.558761966919819  X-1-1
## 2    2  1  3  A 0.816434873119438  X-2-2
## 3    3  1  5  A -4.94461566404576  X-3-3
## 4    4  1  7  B  1.75711244993745  X-4-4
## 5    5  1  9  B -7.03035199251439  X-5-5
## 6    6  1 11  B -2.11248941373554  X-1-6
## 7    7  1 13  B 0.157241524888664  X-2-7
## 8    8  1 15  B -2.38863817110877  X-3-8
## 9    9  1 17  B  -1.0575768718952  X-4-9
## 10 10  1 19  B  -1.0221420115029 X-5-10
## 11 11  1 21  B                6.5  X-1-1
```

```r
#Add one column
 cbind(xDF,NEW=-c(1:10))
```

```
##     V1 V2 V3 V4          V5      V6 NEW
## 1    1  1  1  A  0.5587620  X-1-1  -1
## 2    2  1  3  A  0.8164349  X-2-2  -2
## 3    3  1  5  A -4.9446157  X-3-3  -3
## 4    4  1  7  B  1.7571124  X-4-4  -4
## 5    5  1  9  B -7.0303520  X-5-5  -5
## 6    6  1 11  B -2.1124894  X-1-6  -6
## 7    7  1 13  B  0.1572415  X-2-7  -7
## 8    8  1 15  B -2.3886382  X-3-8  -8
## 9    9  1 17  B -1.0575769  X-4-9  -9
## 10 10  1 19  B -1.0221420 X-5-10 -10
```

```r
##Add two colums
mat=matrix(NA,ncol=2,nrow=10)
colnames(mat)<-c("V7","V8")
 cbind(xDF,mat)
```

```
##     V1 V2 V3 V4          V5      V6 V7 V8
## 1    1  1  1  A  0.5587620  X-1-1 NA NA
## 2    2  1  3  A  0.8164349  X-2-2 NA NA
## 3    3  1  5  A -4.9446157  X-3-3 NA NA
## 4    4  1  7  B  1.7571124  X-4-4 NA NA
## 5    5  1  9  B -7.0303520  X-5-5 NA NA
## 6    6  1 11  B -2.1124894  X-1-6 NA NA
## 7    7  1 13  B  0.1572415  X-2-7 NA NA
## 8    8  1 15  B -2.3886382  X-3-8 NA NA
## 9    9  1 17  B -1.0575769  X-4-9 NA NA
## 10 10  1 19  B -1.0221420 X-5-10 NA NA
```

```r
##Add a variable
xDF$new.var<-2*xDF$V1+1
xDF
```

```
##     V1 V2 V3 V4          V5      V6 new.var
## 1    1  1  1  A  0.5587620  X-1-1       3
## 2    2  1  3  A  0.8164349  X-2-2       5
## 3    3  1  5  A -4.9446157  X-3-3       7
## 4    4  1  7  B  1.7571124  X-4-4       9
## 5    5  1  9  B -7.0303520  X-5-5      11
## 6    6  1 11  B -2.1124894  X-1-6      13
## 7    7  1 13  B  0.1572415  X-2-7      15
## 8    8  1 15  B -2.3886382  X-3-8      17
## 9    9  1 17  B -1.0575769  X-4-9      19
## 10 10  1 19  B -1.0221420 X-5-10      21
```

- summary() gives a five number of continuous variables and a frequency table for character of factor type variables.
- table() returns a frequency table of the observations

```r
#Five number summary of V5
 summary(xDF$V5)

##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -7.0300 -2.3200 -1.0400 -1.5270  0.4584  1.7570

 #Frequency table of V4
 table(xDF$V4)

##
## A B
## 3 7
```

```r
#two by two table of V4 by V6
table(xDF$V4,xDF$V6)
```

```
##
##     X-1-1 X-1-6 X-2-2 X-2-7 X-3-3 X-3-8 X-4-4 X-4-9 X-5-10 X-5-5
##   A     1     0     1     0     1     0     0     0      0     0
##   B     0     1     0     1     0     1     1     1      1     1
```

```r
 #all variables at once
summary(xDF)
```

```
##        V1              V2          V3       V4          V5
##  Min.   : 1.00   Min.   :1   Min.   : 1.0   A:3   Min.   :-7.0304
##  1st Qu.: 3.25   1st Qu.:1   1st Qu.: 5.5   B:7   1st Qu.:-2.3196
##  Median : 5.50   Median :1   Median :10.0         Median :-1.0399
##  Mean   : 5.50   Mean   :1   Mean   :10.0         Mean   :-1.5266
##  3rd Qu.: 7.75   3rd Qu.:1   3rd Qu.:14.5         3rd Qu.: 0.4584
##  Max.   :10.00   Max.   :1   Max.   :19.0         Max.   : 1.7571
##
##        V6       new.var
##  X-1-1  :1   Min.   : 3.0
##  X-1-6  :1   1st Qu.: 7.5
##  X-2-2  :1   Median :12.0
##  X-2-7  :1   Mean   :12.0
##  X-3-3  :1   3rd Qu.:16.5
##  X-3-8  :1   Max.   :21.0
##  (Other):4
```