Chapter 1: Introduction to R

Introduction to R

Question: What is R?

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It has several good features such as

1. an effective data handling and storage facility;
2. a suite of operators for calculations on arrays, in particular matrices;
3. a large, coherent, integrated collection of intermediate tools for data analysis;
4. graphical facilities for data analysis.

R and RStudio

R is a free program, which supports Windows, Mac OS and Linux. You can download from here (click me).

RStudio (IDE) provides a more friendly user interface (highly recommend), which supports Windows, Mac OS and Linux. You can download from here (click me).

Introduction to R - "Hello World"

Simply double click RStudio icon on the desktop to start up program, then choose File → New File → R Script to generate a blank R program.

Assignment is done using the assignment operator <- or =, such as

$$\textit{name} \quad \texttt{<- or =} \quad \textit{expression},$$

where *name* now refers to an object whose value is the result of evaluating *expression*.

Now let us write following two sentences.

```
1 str1 = "Hello World"
  str1
```

Let us click "Run" icon, and read the result from the console.

```
  > str1 = "Hello World"
2 > str1
  [1] "Hello World"
```

Introduction to R - Vectors

We can create vectors at the command prompt using the concatenation function c(...), such as

$$c(object1, object2, ...).$$

The following command instructs R to join together the numbers or people names, and to save them as a vector named numbers and people respectively. When we type number and people, which give us back the vectors.

```
> numbers <- c(1,2,3)
> people <- c("Ali","Bet","Cat")
> numbers
[1] 1 2 3
> people
[1] "Ali" "Bet" "Cat"
```

Tips: Hitting the up arrow multiple times will display the previous commands, which can then be edited. This is useful since one often wishes to repeat a similar command. In addition, typing ?funcname will always cause R to open a new help file window with additional information about the function funcname.

Introduction to R - Calculation

Simple arithmetic operations can be performed with vectors

```
> c(1,2,3)+c(4,5,6)
[1] 5 7 9
> c(1,2,4)*c(1,3,3)
[1]  1  6 12
```

Note in the above example that multiplication is done element by element.

If we try to add together vectors of different lengths, R uses a recycling rule: the smaller vector is repeated until the dimensions match.

```
> small=c(1,2)
> large=c(0,0,0,0,0,0)
> large+small
[1] 1 2 1 2 1 2
```

Question: Please try the case when the dimension of the larger vector is not a multiple of the dimension of the smaller vector.

Introduction to R - ls(), rm() and Clear Console Functions

We have now created a number of objects, like large, small, numbers and people.

The ls() function allows us to look at a list of all of the objects, such as data and functions, that we have saved so far.

The rm() function can be used to delete any that we do not want.

```
> ls()
[1] "large"   "numbers" "people"  "small"
> rm(numbers, people)
> ls()
[1] "large" "small"
```

It is also possible to remove all objects at once.

```
rm(list = ls()) # remove all objects from the memory
```

If we want to clear the console in RStudio, we have two ways: press "Ctrl + L".

Introduction to R - Data Frame

A data frame is an R object that can be thought of as representing a data set. A data frame consists of variables (columns vectors) of the same length with each row corresponding to an experimental unit. The general syntax for setting up a data frame is

$$name = data.frame(variable1,\ variable2,\ \ldots).$$

Individual variables in a data frame are accessed using the $ notation:

$$name\$variable.$$

Once a data frame has been created we can view it in a spreadsheet format using the command View(...). New variables can be added to an existing data frame by assignment. Now let us look at an example to illustrate a data frame function.

Introduction to R - Data Frame

We have taken a random sample of the weight of 5 sheep in the UK. The weights (kg) are

84.5 72.6 75.7 94.8 71.3.

We also measure the height at the shoulder. The heights (cm) are

86.5 71.8 77.2 84.9 75.4.

We will set up another variable for height. We would also like to have a single structure in which the association between weight and height (that is, that they are two measurements of the same sheep) is made explicit. This is done by adding each variable to a data frame. We will call the data frame sheep and view it using View(sheep).

```
> weight = c(84.5, 72.6, 75.7, 94.8, 71.3)
> height = c(86.5, 71.8, 77.2, 84.9, 75.4)
> sheep = data.frame(weight, height)
> mean(sheep$height)
[1] 79.16
> View(sheep) # view it in a spreadsheet
```

Introduction to R - Data Frame

Suppose that a third variable consisting of measurements of the length of the sheeps' backs becomes available. The values (in cm) are

130.4 100.2 109.4 140.6 101.4.

We can include a new variable in the data frame using assignment. Suppose we choose the identifier back length for this new variable:

```
> sheep$backlength = c(130.4, 100.2, 109.4, 140.6, 101.4)
```

Look at the data in spreadsheet format to check what has happened. The result should be the same as the following one.

```
> sheep
  weight height backlength
1   84.5   86.5      130.4
2   72.6   71.8      100.2
3   75.7   77.2      109.4
4   94.8   84.9      140.6
5   71.3   75.4      101.4
```

Introduction to R - Data Frame

The information in the variables height and weight is now encapsulated in the data frame sheep.

```
> rm(height,weight)
> weight
Error: object 'weight' not found
> sheep$weight
[1] 84.5 72.6 75.7 94.8 71.3
```

If we are going to be using the variables in the sheep data frame a lot, we can make them visible from the command line by using the attach(...) command. When we have finished using the data frame, it is good practice to use the detach() command (notice the empty () again) so the encapsulated variables are no longer visible.

```
> weight
Error: object 'weight' not found
> attach(sheep)
> weight
[1] 84.5 72.6 75.7 94.8 71.3
> detach()
> weight
Error: object 'weight' not found
```

Introduction to R - Working Directory

We can save the current workspace to file at any time. It is a good idea to create a folder for each new piece of work that you start. The correct R terminology for this folder is the working directory.

- Create a folder: create a new folder in your D: space and give your new folder the name BigData
- Set working directory:

```
> setwd("D:/BigData")
```

The command to save a workspace is

```
1 > save.image("Introduction.Rdata")
```

Introduction to R - Regular sequence

A regular sequence is a sequence of numbers or characters that follow a fixed pattern. We can use a number of different methods for generating sequences.

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
> 2*1:10
 [1]  2  4  6  8 10 12 14 16 18 20
> 1:10-1
 [1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
> seq(from=1,to=10,length=10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(from=1,to=10,by=1)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(from=1,by=1,length=10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1,10,1)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Introduction to R - array() Function

An array in R consists of a data vector that provides the contents of the matrix and a dimension vector. A matrix is a two dimensional array. The dimension vector has two elements the number of rows and the number of columns. A matrix can be generated by the array() function with the dim argument set to be a vector of length 2.

```
> A = array(data = 1:16, dim = c(4, 4))
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> A = array(data = 1:13, dim = c(4, 4)) # check it by yourself
> A = array(data = 1:20, dim = c(4, 4)) # check it by yourself
```

The penultimate line of the code block above will apply "recycling rule". The last line will discard extra values.

Introduction to R - matrix() Function

Alternatively the matrix() function can be used to create a matrix of numbers as well. Before matrix() we use the matrix() function, we can learn more about it:

```
> ?matrix # alternatively type ?"matrix" in the script
```

The help file reveals that the matrix() function takes a number of inputs, but for now we focus on the first three: the data (the entries in the matrix), the number of rows, and the number of columns. First, we create a simple matrix.

```
> x = matrix (data = c(1,2,3,4) , nrow = 2, ncol = 2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Note that we could just as well omit typing data =, nrow =, and ncol = in the matrix() command above: that is, we could just type

```
x = matrix (c(1,2,3,4), 2, 2)
```

and this would have the same effect.

Introduction to R - matrix() Function

As this example illustrates, by default R creates matrices by successively filling in columns. Alternatively, the byrow = TRUE option can be used to populate the matrix in order of the rows.

```
> x = matrix (data = c(1,2,3,4) , nrow = 2, ncol = 2, byrow = TRUE)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

The sqrt() function returns the square root of each sqrt() element of a vector or matrix. The command x^2 raises each element of x to the power 2; any powers are possible, including fractional or negative powers.

```
> x = x^2
> x = sqrt(x)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
#calculate square and square root of x, the result is unchanged.
```

Introduction to R - Arithmetic Operation of Matrices

Using arithmetic operators on similar arrays will result in element by element calculations being performed. This may be what we want for matrix addition. However, it does not correspond to the usual mathematical definition of matrix multiplication. Conformal matrices can be multiplied using the operator $\%*\%$.

```
> A = array(1:16, dim = c(4, 4))
> B = array(2:17, dim = c(4, 4))
> A + B # check it by yourself
> A %*% B # check it by yourself
```

Matrix inversion and (equivalent) solution of linear systems of equations can be performed using the function solve(). For example, to solve the system $\boldsymbol{Ax} = \boldsymbol{b}$, we would use solve($\boldsymbol{A}, \boldsymbol{b}$). If a single matrix argument is passed to solve(), it will return the inverse of the matrix.

```
> A = array(c(1,3,2,1),c(2,2))
> b = array(c(1,0), c(2,1))
> solve(A, b)
     [,1]
[1,] -0.2
[2,]  0.6
> solve(A) # inverse of A, check it by yourself
```

Introduction to R - Outer Product

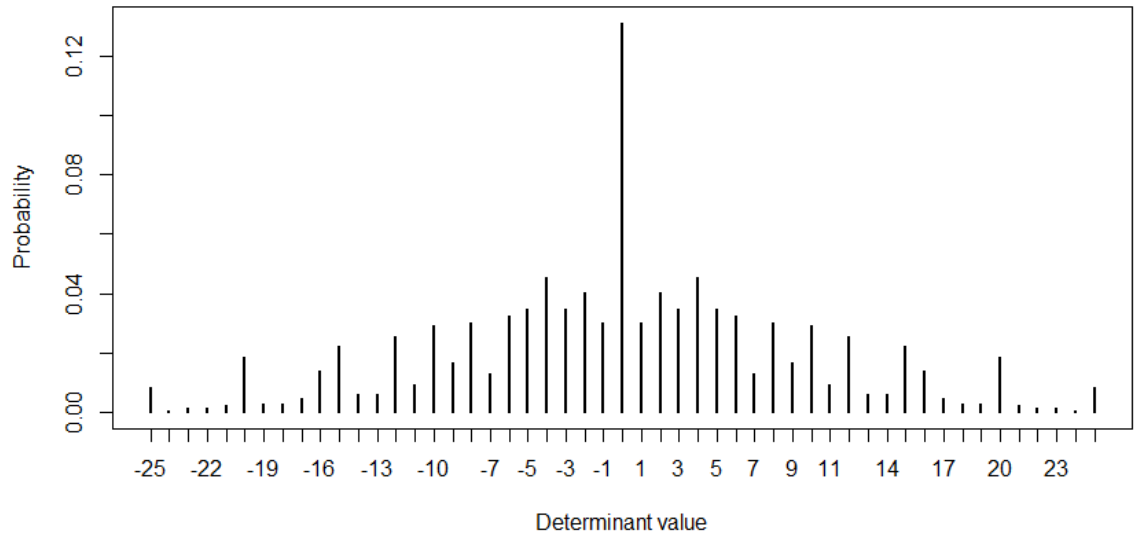
The outer product of two vectors is the matrix generated by looking at every possible combination of their elements. The first two arguments of the outer() function are the vectors. The third argument is the operation that we would like to perform (the default is multiplication).

The following example is taken from Venables et al. [2002]. The aim is to generate the mass function for the determinant of a 2×2 matrix whose elements are chosen from a discrete uniform distribution on the set $\{0, \ldots, 5\}$. The determinant is of the form $AD - BC$ where A, B, C and D are uniformly distributed. The mass function can be calculated by enumerating all possible outcomes. It exploits the table() function that forms a frequency table of its argument works for objects of class "table".

```
A = outer(0:5, 0:5) # every possible value of AD and BC
freq = table(outer(A, A, "-")) # frequency for all values of AD - BC
plot(freq/sum(freq), xlab="Determinant value", ylab = "Probability")
```

Introduction to R - Outer Product

After running the code in previous slide, the plot is as below.



Various other matrix operations are available in R: t() will take the transpose, nrow() will give the number of rows and ncol() the number of columns. The function rbind() (cbind()) will bind together the rows (columns) of matrices with the same number of columns (rows). We can get the eigenvectors and eigenvalues of a symmetric matrix using eigen() and perform singular value decomposition using svd(). Investigating the use of these commands is left to you.

Introduction to R - length() Function

We can tell R to add two sets of numbers together. It will then add the first number from x to the first number from y, and so on. However, x and y should be the same length. We can check their length using the length() function. Let us consider the following example.

```
x = c(1, 3, 2, 5)
y = c(2, 3, 5, 3)
length(x) # 4
length(y) # 4
x + y     # 3 6 7 8
```

Comments can be put almost anywhere, starting with a hash-mark "#", everything to the end of the line is a comment.

Tips: To comment a block of codes in RStudio, select (highlight) codes and press "Ctrl + Shift + C". To format codes in RStudio, select codes and press "Ctrl + Shift + A".

Introduction to R - Indexing Data

We often wish to examine part of a set of data. Suppose that our data is stored in the matrix A.

```
> A = matrix (data = 1:16, nrow = 4, ncol = 4)
# Here a:b is a sequence of integers between a and b.
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Then typing

```
> A[2,3]
[1] 10
```

will select the element corresponding to the second row and the third column. The first number after the open-bracket symbol [ ] always refers to the row, and the second number always refers to the column.

Introduction to R - Indexing Data and dim() Function

We can also select multiple rows and columns at a time, by providing vectors as the indices. The second example is no index for the columns, which indicates R to include all columns. There has a similar result for rows.

```
> A[c(1,3), c(2,4)]
     [,1] [,2]
[1,]    5   13
[2,]    7   15
> A[1:2, ]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
```

The use of a negative sign $-$ in the index tells R to keep all rows or columns except those indicated in the index.

```
> A[-c(1,3), -c(1,3,4)]
[1] 6 8
```

The dim() function outputs the number of rows followed by the number of columns of a given matrix.

```
> dim(A)
[1] 4 4
```

Introduction to R - List

A list is an ordered collection of objects. The objects in a list are referred to as components. The components may be of different modes. Lists can be generated using the function list(...); the return value is a list with components made up of the arguments passed to the function. Below we create a list, look at the intrinsic attributes then display the contents of the list.

```
> listEx = list("Test", c(2, 0, 6), sheep) # sheep is a data frame
> length(listEx)
[1] 3
> listEx[2]
[[1]]
[1] 2 0 6
```

We can use the indexing operator [ ] to access parts of the list. Note that the result of using indexing on a list is another list. If we want to add $c(1, 2, 4)$ to $c(2, 0, 6)$, we cannot do this using [ ] indexing since *listEx*[2] is not numeric. To avoid this problem we used a double square bracket subscripting operator [[ ]].

```
> listEx[[2]] + c(1, 2, 4)
[1]  3  2 10
```

Introduction to R - List

The default labels for the components of a list (1, 2, 3,...) bear no association to what the list contains. In the example above, we have to remember that the date was the second component of our list. This is no great hardship for a list with three elements but becomes very inconvenient in larger lists. We can associate names with the components of a list.

```
> names(listEx) = c("String", "Vector", "DataFrame")
> df = as.data.frame(listEx[["DataFrame"]])
> View(df)
# Please check modes of listEx[["String"]] and listEx[["Vector"]]
```

In the second line of the code, the mode of listEx[["DataFrame"]] is list instead of data frame, so we apply the command as.data.frame to coerce it to be a data frame. Finally we can inspect this data frame by View(), which has been seen before.

Introduction to R - Loading Data

For most analyses, the first step involves importing a data set into R. The read.table() function is one of the primary ways to do this. The help file read.table() contains details about how to use this function. We can use the function write.table() to export data.

The following command will load the Auto.data file into R and store it as an object called Auto, in a format referred to as a data frame. Once the data has been loaded, the View() function can be used to view it in a spreadsheet like window.

```
> fileName = "Auto.data"
> defaultDataDir = "C:\Users\Cheng\Desktop\PUK–LSE summer School\R
    Code\Introduction to Statistical Learning\Dataset"
> fileLocation = file.path(defaultDataDir, fileName)
> Auto = read.table(file = fileLocation)
> View(Auto) # check it
```

Introduction to R - Loading Data

This particular data set has not been loaded correctly, because

- R has assumed that the variable names are part of the data and so has included them in the first row;
- the data set also includes a number of missing observations, indicated by a question mark ?.

Using the option header = T (or header = TRUE) in the read.table() function tells R that the first line of the file contains the variable names, and using the option na.strings tells R that any time it sees a particular character or set of characters (such as a question mark), it should be treated as a missing element of the data matrix.

```
> Auto = read.table(file = fileLocation, header = T, na.strings = "?
    ")
```

Introduction to R - Loading Data

Excel is a common-format data storage program. An easy way to load such data into R is to save it as a csv (comma separated value) file and then use the read.csv() function to load it in.

```
> fileName = "Auto.csv"
> fileLocation = file.path(defaultDataDir, fileName)
> Auto = read.csv(file = fileLocation, header = T, na.strings = "?")
> dim(Auto)
[1] 397   9
```

The dim() function tells us that the data has 397 observations, or rows, and dim() nine variables, or columns. There are various ways to deal with the missing data. In this case, only five of the rows contain missing observations, and so we choose to use the na.omit() function to simply remove these rows.

```
> Auto = na.omit(Auto)
> dim(Auto)
[1] 392   9
```

Introduction to R - Loading Data and summary() Function

Once the data are loaded correctly, we can use names() to check the variable names.

```
> names(Auto)
 [1] "mpg "  "cylinders "  "  displacement"  "horsepower "
 [5] "weight "  "  acceleration"  "year"  "origin "
 [9] "name"

#check following commands
> names(Auto)[1] = "New_Name_MPG"
> names(Auto)
```

The summary() function produces a numerical summary of each variable in a particular data set. For qualitative variables such as name, R will list the number of observations that fall in each category. We can also produce a summary of just a single quantitative variable.

```
> summary(Auto) # check it by yourself
> summary(Auto$mpg)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
    9.00    17.00    22.75    23.45    29.00    46.60
```

Introduction to R - Graphics - plot() Function

We can use the plot() function to produce scatterplots of the quantitative variables. However, simply typing the variable names will produce an error message, because R does not know to look in the Auto data set for those variables.

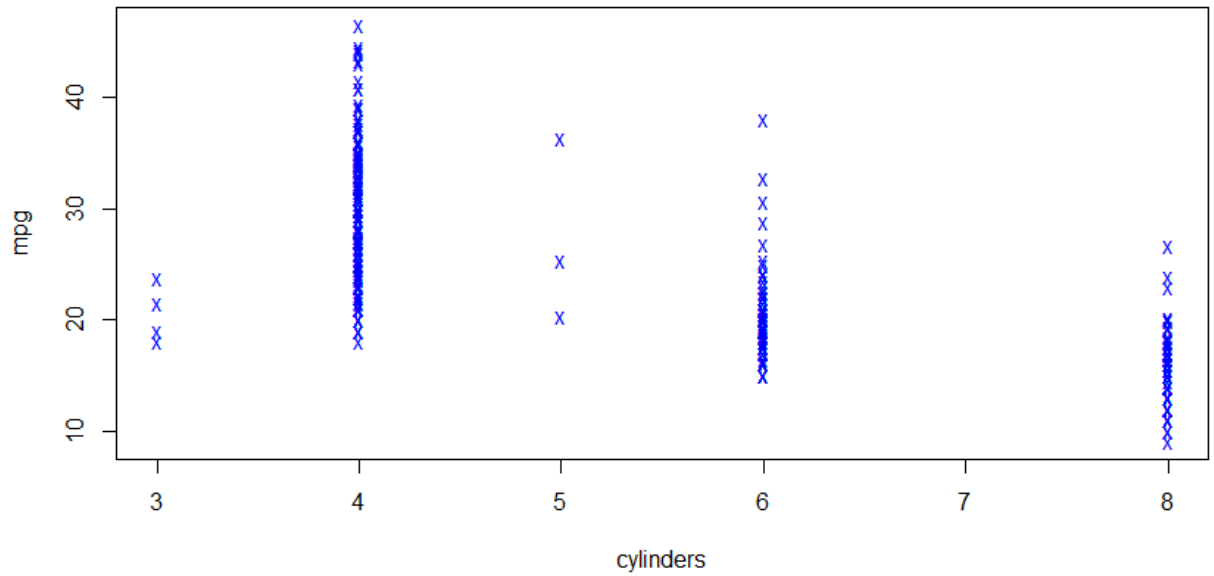
```
> plot(cylinders, mpg)
Error in plot(cylinders, mpg) : object 'cylinders' not found
```

To refer to a variable, we must type the data set and the variable name joined with a $ symbol. Alternatively, we can use the attach() function in order to tell R to make the variables in this data frame available by name.

```
> plot(Auto$cylinders, Auto$mpg, type = "p", pch = "x", col = "blue"
    , xlab = "cylinders", ylab = "mpg")
> attach(Auto)
> plot(cylinders, mpg, type = "p", pch = "x", col = "blue", xlab = "
    cylinders", ylab = "mpg")
```

Introduction to R - Graphics - as.factor() Function

After running the code in previous slide, the plot is as below.



The cylinders variable is stored as a numeric vector, so R has treated it as quantitative. However, since there are only a small number of possible values for cylinders, one may prefer to treat it as a qualitative variable. The as.factor() function converts quantitative variables into qualitative variables.

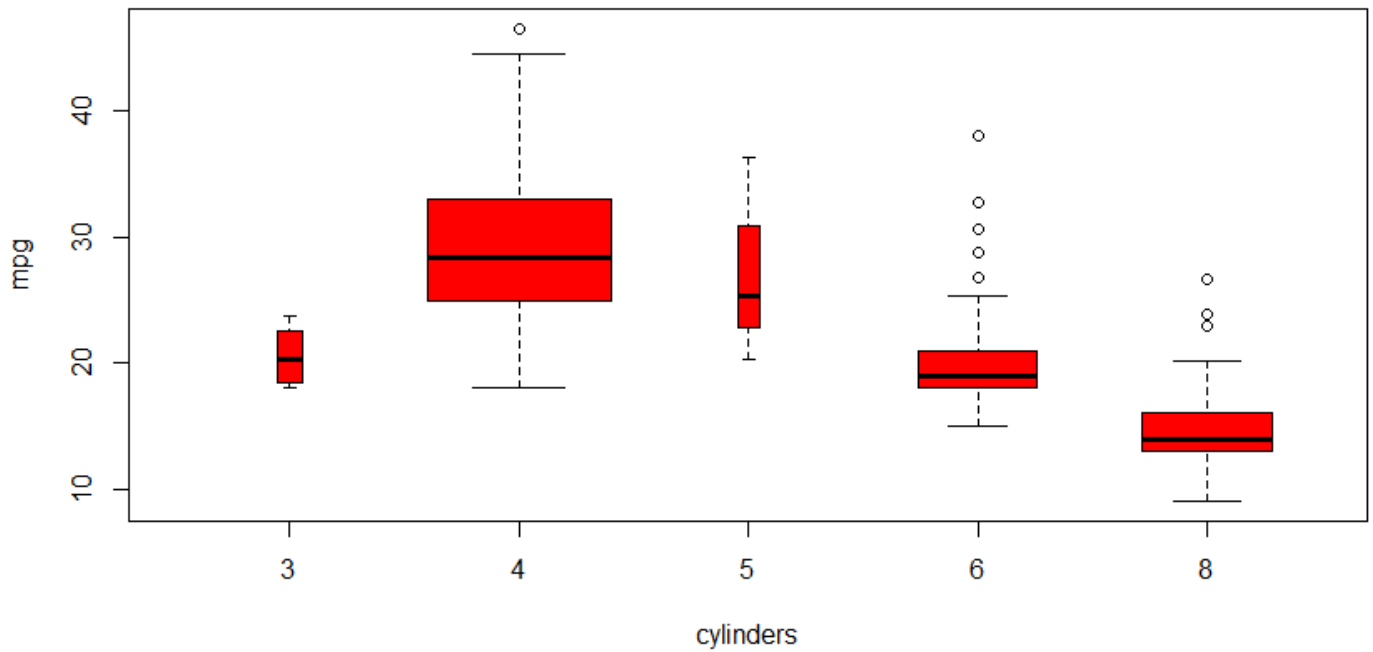
```
1 > cylinders = as.factor(cylinders)
```

Introduction to R - Graphics - Plot Categorical Data

If the variable plotted on the x-axis is categorical, then boxplots will automatically be produced by the plot() function. As usual, a number of options can be specified in order to customize the plots.

```
> plot(cylinders, mpg, col = "red", varwidth = T, xlab = "cylinders"
    , ylab = "mpg")
```

After running the code above, the plot is as below.

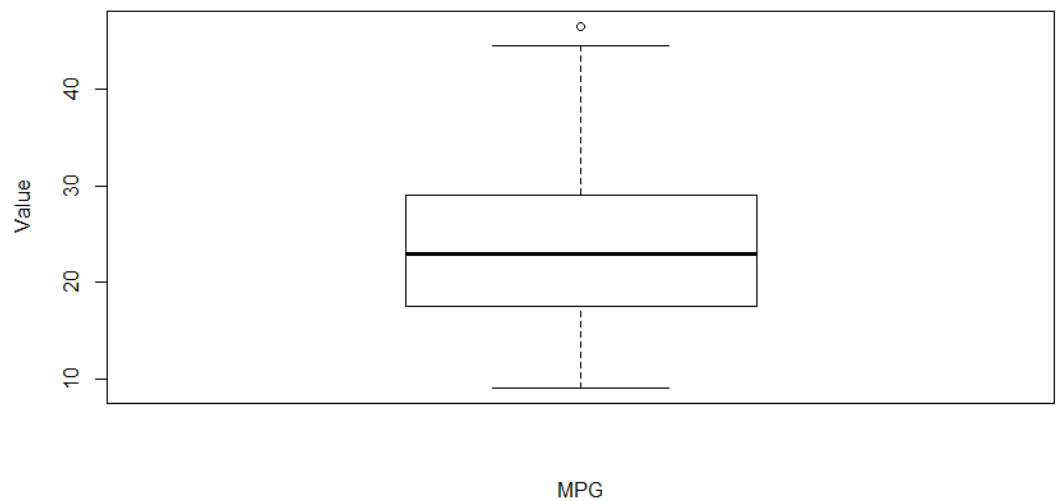


Introduction to R - Graphics - boxplot() Function

Boxplots provide another mechanism for getting a feel for the distribution of data. Parallel boxplots are useful for comparison. The full name is a box-and-whisker plot. The box is made up by connecting three horizontal lines: the lower quartile, median and upper quartile. In the default set up, the whiskers extend to any data points that are within 1.5 times the inter-quartile range of the edge of the box.

```
1 > boxplot(mpg, xlab = "MPG", ylab = "Value")
  > boxplot(acceleration, cylinders) # check it by your self
```

After running the code above, the plot is as below.

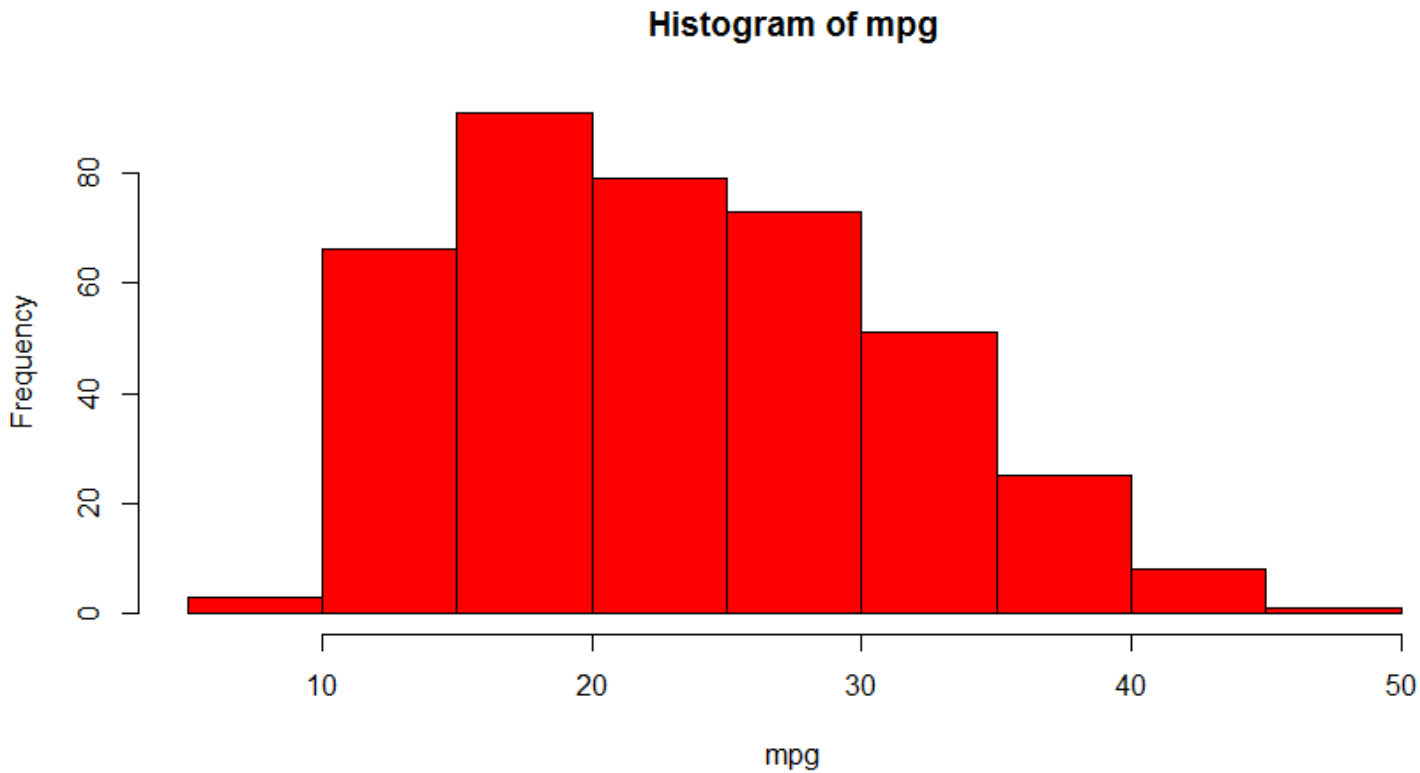


Introduction to R - Graphics - hist() Function

The hist() function can be used to plot a histogram. Note that col = 2 has the same effect as col = "red".

```
> hist(mpg, col = 2, breaks = 15)
```

The argument breaks in hist() is a single number giving the number of cells for the histogram. After running the code above, the plot is as below.

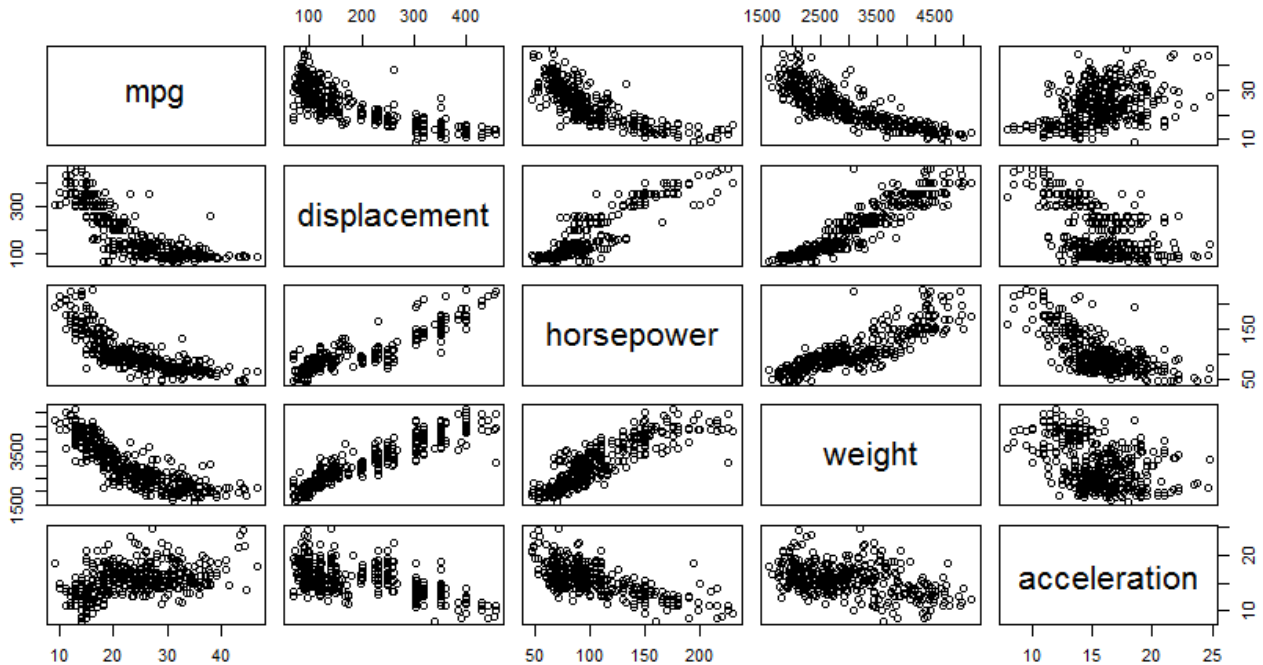


Introduction to R - Graphics - pairs() Function

The pairs() function creates a scatterplot matrix i.e. a scatterplot for every pair of variables for any given data set. We can also produce scatterplots for just a subset of the variables.

```
> pairs(Auto) # check it by yourself
> pairs(~mpg + displacement + horsepower + weight + acceleration ,
    Auto)
```

After running the second line of the code above, the plot is as below.

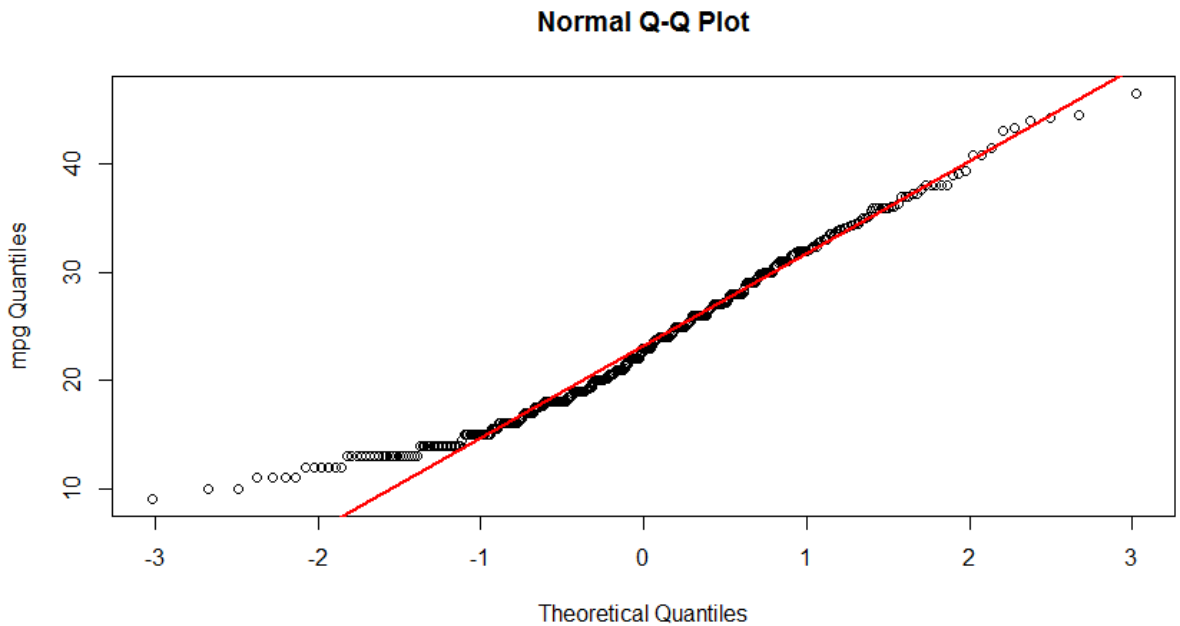


Introduction to R - Graphics - qqnorm() Function

It is often useful to compare a data set to the normal distribution. The qqnorm() command plots the sample quantiles against the quantiles from a normal distribution. A qqline() command after qqnorm(...) will draw a straight line through the coordinates corresponding to the first and third quartiles. We would expect a sample from a normal to yield points on the qq-plot that are close to this line.

```
> qqnorm(mpg, ylab = "mpg Quantiles")
> qqline(mpg, col = 2, lwd = 2)
```

After running the codes above, the plot is as below.



Introduction to R - Graphics - 3D Plot

We can generate various types of three dimensional plot in R. The input for this type of plotting function is two vectors (values from the x and y axes) and a matrix (values from the surface that is to be plotted). We are going to generate 3-D plots for the surface

$$f(x, y) = x^2 y^3$$

for $x \in [-1, 2]$ and $y \in [1, 4]$. In order to do this we first generate regular sequence of 50 points on the interval $[-1, 1]$ and then use the outer function to give the 50×50 matrix (outer product) of points on the surface.

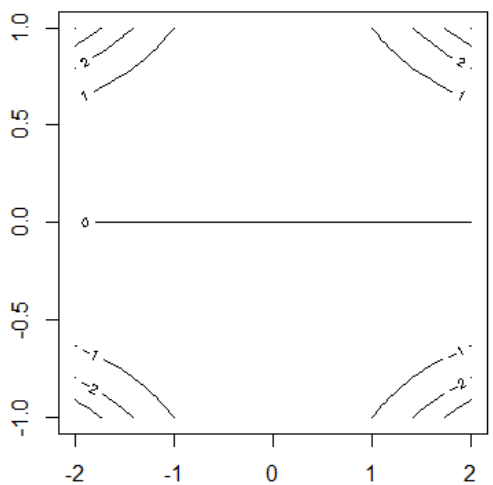
```
> x = seq(from = -2, to = 2, length=50)
> y = seq(from =-1, to =1, length=50)
```

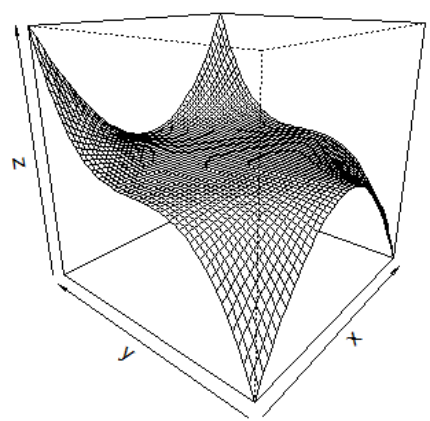
The function seq() can be used to create a sequence of numbers. For seq() instance, seq(a, b) makes a vector of integers between *a* and *b*. For instance, seq(-2, 2, length = 50) makes a sequence of 50 numbers that are equally spaced between -2 and 2.

Introduction to R - Graphics - 3D Plot

The par function can be used to set or query graphical parameters. $mfrow = c(1, 2)$ makes the output of images in 1×2 format. The contour() function produces a contour plot in order to represent three-dimensional data. The persp() function draws perspective plots of a surface over the xy plane. The arguments theta and phi control the angles at which the plot is viewed.

```
> par(mfrow = c(1, 2))
> z = outer(x^2,y^3)
> contour(x, y, z)
> persp(x, y, z, theta = -50, phi = 20)
```





Introduction to R - Logical vectors

The logical values in R are TRUE and FALSE.

```
> x = 10
> y = 20
> z = 10
> x == y
[1] FALSE
> x == z
[1] TRUE
> x != y
[1] TRUE
> x > y
[1] FALSE
> x <= y
[1] TRUE
> x == z & x != y
[1] TRUE
> x == z | x == y
[1] TRUE
> ! x == z
[1] FALSE
```

Introduction to R - More R Essentials - Loops

A for loop often provides the most obvious implementation.

$$for(loop\ variable\ \ in\ \ sequence)\ expr1$$

Here sequence is actually any vector expression but usually takes the form of a regular sequence such as 1:5. The statements of expr1 are executed for each value of the loop variable in the sequence.

```
for (i in 1:5) print(i)
```

We can achieve the same result as a for loop using either a while or repeat loop. A while loop continues execution of the expression while the condition holds true. A repeat loop repeated executes the expression until explicitly terminated.

$$while\ (condition)\ expr\quad or\quad repeat\ expr$$

```
i = 1; # try to write repeat loop by yourself
while(i < 6){
  print(i)
  i = i + 1}
```

Introduction to R - More R Essentials - Conditional Statements

The if statement in R follows the standard syntax such as

$$if\,(condition)\ if\_expr \quad or \quad if\,(condition)\ if\_expr\ else\ else\_expr.$$

Here the condition is an expression that yields a logical value (TRUE or FALSE) when evaluated.

```
> y = rep("NA", times = length(x))
> for(i in 1:length(x)){
+   if(x[i] >= 0)
+     y[i] = "non-negative"
+   else
+     y[i] = "negative"
+ }
```

→ use for loop cuz we need to check one by one.

Loops are not efficiently implemented in R. One way of avoiding the use of loops is to use commands that operate on whole objects. For example, ifelse() is a conditional statement that works on whole vectors (rather than requiring a loop to go through the elements).

```
x = seq(-5, 10, by = 1.5)
y = ifelse(x < 0, "negative", "non-negative")
```

expression for otherwise

Introduction to R - More R Essentials - Writing Your Own Functions

We have seen that there are a large number of useful function built into R; these include mean(), plot() and outer(), etc. Furthermore, one of the most powerful features of R is that the user can write their own functions. This allows complicated procedures to be built with relative ease. The general syntax for defining a function is

$$functionName = function(arg1, arg2, ...)\ \{expr1\}$$

The function is called by using functionName(). When the function is called the statements that make up expr1 are executed. The final line of expr1 gives the return value.

```
> fn = function(arg1){
+   if(arg1 <= 0){
+     100}
+   else{
+     2 + fn(arg1 - 1)}
+ }
> m = fn(5) # please check fn(-5). Guess it first : )
> m
[1] 110
```

We can exploit the ideas of named arguments and default values in functions. Consider a function to draw the binomial mass function for some values of number of trial n (size) and probability of success p (prob).

The argument colour is a number to specify the colour of the plotted lines while outputvals is a logical value; if outputvals is TRUE, R will print out the values used to generate the plot. In general, we may want green lines and not printing out all of the values used to generate the plot. If this is the case, we can specify default values for these arguments.

```
> binomplot = function(size, prob=0.5, colour=3, outputvals=FALSE)
+ { x = 0:size
+   y = dbinom(x, size, prob)
+   plot(x, y, type="h", col=colour)
+   if (outputvals) y
+ }
> binomplot(50)
> binomplot(55, outputvals = TRUE, colour = 1)
```

For a discrete distribution there is a countable set of possible values of the variable, $\{y_1, y_2, \dots\}$ (the points at which the discontinuities in F_Y occur). The probability mass function is defined by

$$f_Y(y) = \mathbb{P}(Y = y)$$

and takes non-zero values on $\{y_1, y_2, \dots\}$. The distribution function can then be written as

$$F_Y(y) = \sum_{i:y_i \leq y} f_Y(y).$$

Some commonly used mass functions are given below:

- Binomial(n, p): $f_Y(y) = \begin{pmatrix} n \\ p \end{pmatrix} p^y (1-p)^{n-y}$ for $y = 0, \dots, n$;
- Poisson(λ): $f_Y(y) = \lambda^y e^{-\lambda}/y!$ for $y = 0, 1, \dots$;
- Discrete Uniform $\{1, \dots, n\}$: $f_Y(y) = 1/n$ for $y = 1, \dots, n$.

For a continuous distribution the density function, f_Y, is a positive real-valued function satisfying

$$F_Y(y) = \int_{-\infty}^{y} f_Y(u)du.$$

The density at a point is not a probability. Probabilities can be calculated from the density function by integration:

$$\mathbb{P}(a < Y < b) = \int_{a}^{b} f_Y(y)dy.$$

Some commonly used densities are given below:

- normal(μ, σ^2): $f_Y(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/(2\sigma^2)}$ for $y \in \mathbb{R}$;
- exponential(θ): $f_Y(y) = \theta e^{-\theta y}$ for $y > 0$;
- continuous uniform$[a, b]$: $f_Y(y) = 1/(b-a)$ for $y \in [a, b]$.

Introduction to R - Probability and Quantile Calculations

R can be used like a set of statistical tables, that is, for a random variable Y , work out the values of p and q associated with expressions of the form

$$\mathbb{P}(Y \leq q) \geq p.$$

In order to calculate probabilities from points on a distribution, we put a p in front of the distribution name. The function *pdistributionname*$(q, \ldots)$ will return the probability that the distribution named takes a value less than or equal to q. The follow simple example illustrates. Suppose $X \sim Bin(85, 0.6)$ and $Y \sim N(12, 9)$ and that X and Y are independent. Calculate $\mathbb{P}(X \leq 60)$, $\mathbb{P}(Y > 15)$ and $\mathbb{P}(X < 60, Y > 15)$.

```
> pbinom(q = 60, size = 85, prob = 0.6)
[1] 0.9837345
> pnorm(q = 15, mean = 12, sd = 3, lower.tail = FALSE) # 1 - pnorm(q
      = 15, mean = 12, sd = 3)
[1] 0.1586553
> pbinom(q = 59, size = 85, prob = 0.6) * pnorm(q = 15, mean = 12,
     sd = 3, lower.tail = FALSE)
[1] 0.1541691
```

Introduction to R - Density and Cumulative Distribution

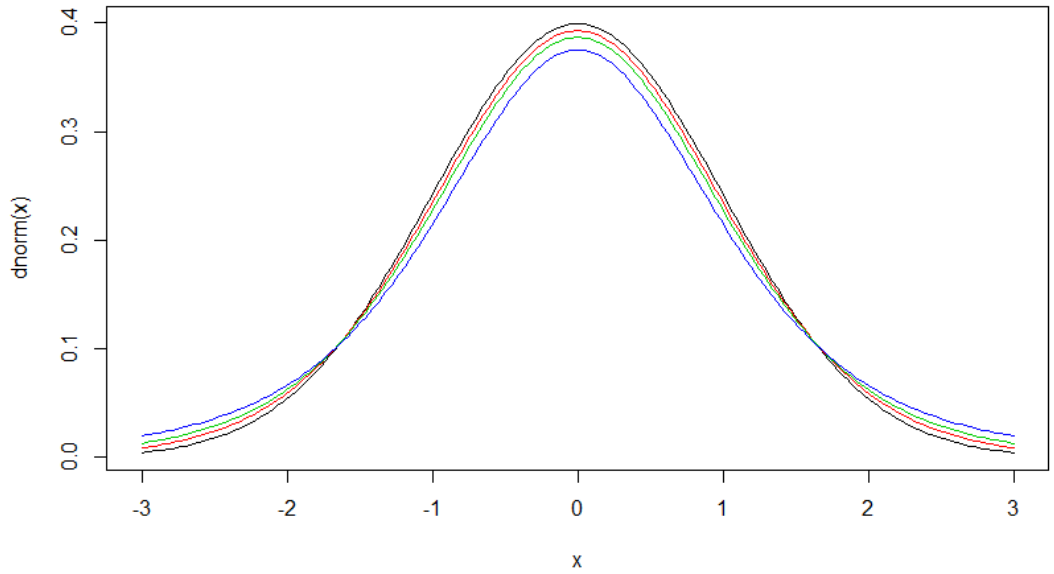
R allows us to calculate density function values (probability mass in the discrete case). The function ddistributionname (x, ...) will generate the value of the density function (probability mass function) for the named distribution at x. These values can be used to density function plots.; We use this function to investigate the effect of the parameter values on the t-distribution below.

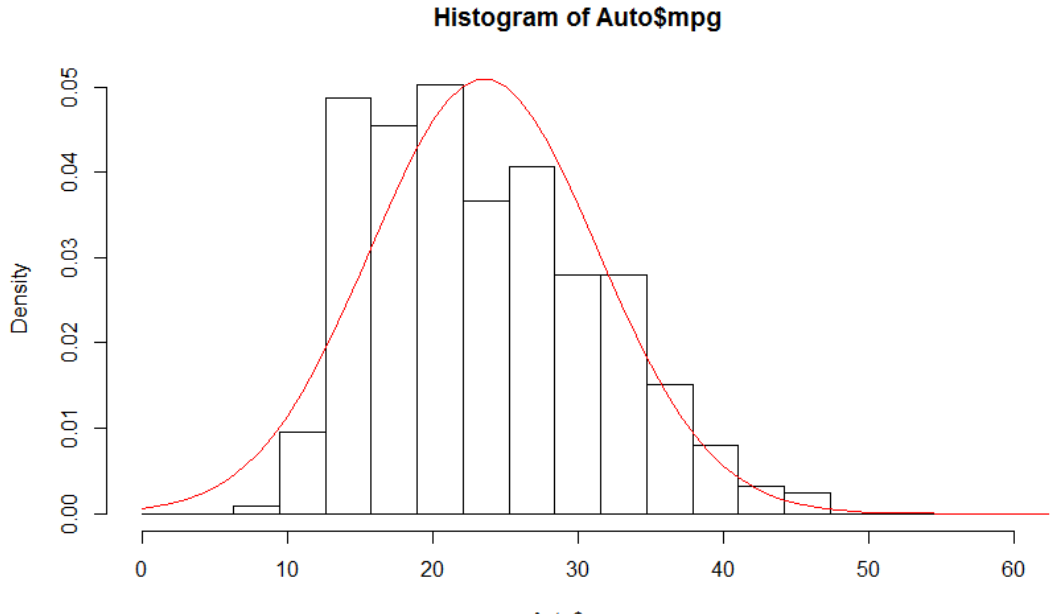
```
> x = seq(from = -3, to = 3, length=200)
> plot(x, dnorm(x), type="l")
> lines(x, dt(x, df = 16), col = 2)
> lines(x, dt(x, df = 8), col = 3)
> lines(x, dt(x, df = 4), col = 4)
```

In determining which distribution is appropriate for a set of data, a plot comparing the empirical density (or empirical cumulative distribution) with the theoretical density is useful. This applies to data from simulation experiments as well as observed data.

```
> hist(Auto$mpg, breaks=seq(0,60,length = 20), probability = TRUE)
> lines(x, dnorm(x, mean = mean(Auto$mpg), sd = sd(Auto$mpg)), col
    =2)
```

Introduction to R - Density and Cumulative Distribution





Introduction to R - Samples and Statistics

An observed sample, $y_1, \ldots, y_n$, from a population is a collection of numbers. These numbers are often referred to as the data. The data can be thought of as one instance of a collections of IID random variables $Y_1, \ldots, Y_n$. The value of a statistic is a function of the data; to represent this general relationship we use the notation

$$u = h(y_1, \ldots, y_n).$$

The term statistic refers to the same function h applied to the random variables $Y_1, \ldots, Y_n$,

$$U = h(Y_1, \ldots, Y_n).$$

Just as the data $y_1, \ldots, y_n$ are thought of as one instance of the IID r.v.s $Y_1, \ldots, Y_n$, the value u is taken to be one instance of the statistic U.

The Central Limit Theorem: Let $Y_1, \ldots, Y_n$ be independent, identically distributed random variables with mean $E(Y) = \mu_Y$ and finite variance $Var(Y) = \sigma_Y^2 < \infty$. Let

$$Z_n = \frac{\bar{Y} - \mu_Y}{\sigma_Y / \sqrt{n}}$$

then Z_n converges in distribution to a standard normal as $n \to \infty$.

Introduction to R - Generating (pseudo-) Random Samples

Using R we can generate random instances from any commonly used distribution. The function rdistributionname (n, ...) will return a random sample of size n from the named distribution. At the heart of this function, R uses some of the most recent innovations in random number generation. We illustrate by sampling from Poisson and normal distributions.

```
> poissamp = rpois(n = 400, lambda = 2)
> hist(poissamp, breaks = 0:10, probability = TRUE)
> normsamp = rnorm(n = 250, mean = 10, sd = 5)
> hist(normsamp, breaks = seq(-10, 30, length = 15), probability =
    TRUE)
> x = seq(-10,30,length = 200)
> lines(x, dnorm(x, mean = 10, sd = 5), col = 2)
```

The command set.seed() allows us to determine the starting point of the iterative process and thus ensure identical output from the random number generator. This is useful when developing the code for a simulation experiment.

Introduction to R - Generating (pseudo-) Random Samples

```
> set.seed(1)
> rnorm(5)
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

The functions sample() can be used to generate random permutations and random samples from a data vector. The arguments to the function are the vector that we would like to sample from and the size of the vector (if the size is excluded a permutation of the vector is generated). Sampling with replacement is also possible using this command.

```
> nvec = 1:6
> sample(nvec)
[1] 2 1 3 4 6 5
> sample(nvec, size = 3)
[1] 3 6 4
> sample(nvec, replace = TRUE)
[1] 4 4 3 2 6 4
> sample(nvec, size = 10, replace = TRUE)
[1] 2 1 3 6 4 6 5 3 3 1
```

Introduction to R - Simulation Experiments

To demonstrate how a simulation experiment works, we are going to look at the small sample properties of the sample mean for a Poisson population, a statistic whose asymptotic properties are well known. Consider a population that has a Poisson distribution with mean λ, so $Y \sim Pois(\lambda)$. We take a sample of size n. We would like to know whether the normal distribution will provide a reasonable approximation to the distribution of the sample mean.

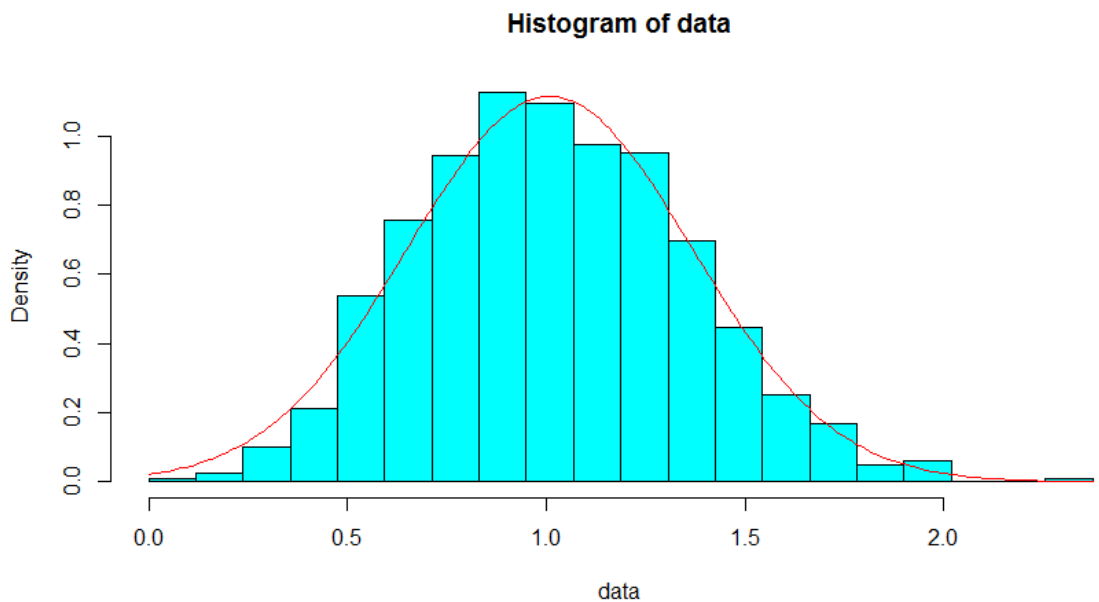
We will use the computer to generate simulated samples $(y_1^{(1)}, \ldots, y_n^{(1)})$, $(y_1^{(2)}, \ldots, y_n^{(2)})$, $\ldots$, $(y_1^{(r)}, \ldots, y_n^{(r)})$ where r is the number of simulated replications. For each of these simulates samples, we evaluate the sample mean to give a sequence $\bar{y}^{(1)}, \ldots, \bar{y}^{(r)}$ that can be viewed as instances of the statistic of interest $\bar{Y}$.

```
> poisSampMean = function(n, lambda, r){
+   simvals = rpois(n * r, lambda)
+   simvals = matrix(simvals, n, r)
+   colMeans(simvals)
+ }
> set.seed(1)
> poisSampMean(10, 3, 6)
[1] 3.3 3.4 2.6 3.0 3.3 2.6
```

Introduction to R - Simulation Experiments

To get a visual impression of the simulated sample means we write a function to draw a histogram and plot a normal distribution with the same mean and standard deviation.

```
> histNorm = function(data, nbins = 21){
+   hist(data, breaks = seq(min(data), max(data), length=nbins),
       probability = TRUE, col = 5)
+   x = seq(min(data), max(data), length = 200)
+   lines(x, dnorm(x, mean = mean(data), sd = sd(data)), col = 2)
+ }
> histNorm(poisSampMean(8, 1, 1000))
```



Introduction to R - Coding Practice

Q1. (if condition) Write a function named calculate(x, y, operator), where x and y are two numeric values and operator is an character like "+", "-", "*" and "/". The return value is x plus y when operator is "+", x minus y when operator is "-", x times y when operator is "*" and x divide y when operator is "/". Note: think about how to set the default value for operator to be "+".

Q2. (indexing and for loop) Write a function named Vector2Matrix(x), where the only argument is a vector x whose length is p. The return is a p by p matrix and the ith row jth column is ith element in x minus jth element in x. Note: use for loop first and then think about how to avoid using for loop for this.

Q3. (while loop) Write a function named oddSum(x), where the x is a numeric number. The output of this function is the sum of all odd numbers between 0 and x inclusively. Note: use both while loop and for loop and combining the Q2 think about the pros and cons of while and for loops respectively.

Introduction to R - Coding Practice

Q4. (Simulation 1) Compare two estimators for mean value. The first estimator: sum of all values and then divided by the number of values. The second estimator: sum of the largest two numbers and the smallest two numbers and then divided by 4. Note: first, generate the sample by using normal distribution with mean 0 and variance 1. Second, use boxplot to show the results including title, x label and y label. Third, try different sample sizes, like 10, 100, 1000. (We repeat 100 times for this simulation)

Q5. (Simulation 2) In a primitive society, every couple prefers to have a baby girl. There is a 50% chance that each child they have is a girl, and the genders of their children are mutually independent. If each couple insists on having more children until they get a girl and once they have a girl they will stop having more kids. Use R to caculate the expected value of the number of babies in a family in this society and the expected value of the number of girls in a family in this society.

Introduction to R - Optional question

Q6. (Google interview) You are given a vector of characters like ["A","B","C","D","A"], ["B","C","B","A","C"] or ["C","B","C"] and you should write a function to return the first recurring character in that vector. For example, ["A","B","C","D","A"] should return A, ["B","C","B","A","C"] should retrun B and ["C","B","C"] should return C. If there is no recurring character, then return NA.

Q7. (Amazon interview) You are given a sequence of numbers, like [1,2,3,9] or [1,2,4,4] and you are also given a number, like 8. You are asked to find out whether there is a pair that the sum of them is the number you are given, like 8 in this question.

References

William N Venables, David M Smith, R Development Core Team, et al. An introduction to r, 2002.