Introduction to R: Data Structures and Functions

Machine Learning

AY 2025-2026

Contents

What is R?

- R is a free, open-source software and programming language
- Developed as an environment for statistical computing and graphics
- Now one of the dominant software environments for data analysis
- Used by a variety of scientific disciplines, including economics, econometrics, finance etc
- R: a language to explore, summarize, and model data
    - functions = verbs
    - objects = nouns
- R: encourages publications of "Reproducible Research"
    - integrate data, code, text into one document
    - Sweave and knitr

Installing R and RStudio

- Download and install the latest R: http://www.r-project.org/
- Download and Install RStudio: https://rstudio.com/
- R is the underlying engine that powers RStudio's computations
- RStudio will provide sample data, command autocompletion, help files, and an effective interface for getting things done quickly
- There are four main parts of RStudio
    - The Editor: the top left quadrant - where you write code for later use

  - The Console: the lower left quadrant
  - History / Environment: the top right quadrant
  - Misc: the bottom right panel

Launching RStudio and Installing Packages

- R provides several in-house and user contributed packages
- The easiest way to install packages is to do it via R console.
- The command *install.packages("package name")* installs R packages directly from internet.

```
# the help window in RStudio
    # help(install.packages)

    # Example
    # Install package Quantreg
    # install.packages("quantreg")

    # to activate the package
   # library(quantreg)
```

Updating R

To update R to the newest version, using the following code

```
# Open R and type the following

    # install.packages("installr")
    #
    # library(installr)
    #
    # updateR()
```

Data Types in R

- To make the best of the R language, you will need a strong understanding of the basic data types and data structures and how to operate on them.
- Everything in R is an object.
- R has many data types. These include
  - character
  - numeric (real or decimal)
  - integer
  - logical
  - complex
  - factors
  - date and time

Characters

```
The character class is the typical string, a set of one or more letters.
```

```r
# Example
x <- "dataset"
class(x)
```

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

```r
# Example
Names <- c("Mario", "Federico", "Gloria", "Manuel", "Marco")
Names
```

```
## [1] "Mario"    "Federico" "Gloria"   "Manuel"   "Marco"
```

```r
## Note: c()    Combine values or characters into a vector
class(Names)
```

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

```r
# Example
Years <- c("1999.5", "2000.1", "2001.3", "2002.4", "2003.6")
class(Years)
```

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

Numeric

The numeric class is the typical real or decimal numbers.

```r
# Example
Height <- c(58.5, 44.6, 67.2, 61.3, 59.7)
class(Height)
```

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

You can change numeric string values into numeric types

```r
# Example
Years <- c("1999.5", "2000.1", "2001.3", "2002.4", "2003.6")
class(Years)
```

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

```r
Y <- as.numeric(Years)
class(Y)
```

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

Integers

The integer class are whole numbers, though they get changed into numerics when saved into variables:

```
# Example
Age <- c(24L, 22L, 23L, 24L, 25L)
Age
```

```
## [1] 24 22 23 24 25
```

```
### the L tells R to store this as an integer
Age.n <- as.numeric(Age)
class(Age.n)
```

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

You can change numeric (string) values into integer types

```
# Example
Years <- c("1999.5", "2000.1", "2001.3", "2002.4", "2003.6")
Years
```

```
## [1] "1999.5" "2000.1" "2001.3" "2002.4" "2003.6"
```

```
class(Years)
```

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

```
W <- as.integer(Years)
W
```

```
## [1] 1999 2000 2001 2002 2003
```

```
class(W)
```

```
## [1] "integer"
```

Logical

The logical class is the typical either true, or false statements. True can be represented with TRUE or *T* while false is FALSE or *F*.

```
# Example
Smoker <- c(FALSE, FALSE, TRUE, TRUE, FALSE)
Smoker
```

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

```
class(Smoker)
```

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

```r
# Example
S <- c(F, F, T, T, F)
S
```

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

```r
class(S)
```

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

Complex

The complex class is the typical numbers with real and imaginary parts.

```r
# Example
C <- c( 3+4i, 5+12i, 9+12i )
C
```

```
## [1] 3+ 4i 5+12i 9+12i
```

```r
class(C)
```

```
## [1] "complex"
```

Factors

The factor class is an important data type to represent categorical data. Factor objects can be created from character object or from numeric object.

```r
# Example of factors are Blood type (A , B, AB, O)
b.type = c("A", "AB", "B", "O")     # character object

# use factor Function to convert to factor object
b.type2 = factor(b.type)
b.type2
```

```
## [1] A  AB B  O
## Levels: A AB B O
```

```r
class(b.type2)
```

```
## [1] "factor"
```

Date and Time

R is capable of dealing with calendar dates and times. This is important when dealing with time series models. The function *as.Date* can be used to create an object of class Date.

```r
# Example
date1 = "05-11-2019"
date2 = as.Date(date1, "%d-%m-%Y")
date1
```

```
## [1] "05-11-2019"
```

```r
date2
```

```
## [1] "2019-11-05"
```

```r
class(date1)
```

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

```r
class(date2)
```

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

Data Structures in R

R has many data structures. These include - vectors - matrices - arrays - data frames - lists

Vectors

Vectors are group of values having same data types.

There can be numeric vectors, character vector etc.

```r
# Example
Names <- c("Mario", "Federico", "Gloria", "Manuel", "Marco")
Names
```

```
## [1] "Mario"    "Federico" "Gloria"   "Manuel"   "Marco"
```

```r
X <- 1:10
X
```

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

```r
Y <- seq(from = 1, to = 10, by = 0.1)
Y
```

```
##  [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4
## [16]  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9
## [31]  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1  5.2  5.3  5.4
## [46]  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9
## [61]  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.0  8.1  8.2  8.3  8.4
## [76]  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9
## [91] 10.0
```

Matrices

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.

Like vectors all the elements in a matrix are of same data type.

```
# Example
M1 <- matrix(c("A", "B", "C", "D", "E", "F"), nrow = 3,
             ncol = 2, byrow = T)
M1
```

```
##      [,1] [,2]
## [1,] "A"  "B"
## [2,] "C"  "D"
## [3,] "E"  "F"
```

```
M2 <- matrix(c("A", "B", "C", "D", "E", "F"), nrow = 3,
              ncol = 2, byrow = F)
M2
```

```
##      [,1] [,2]
## [1,] "A"  "D"
## [2,] "B"  "E"
## [3,] "C"  "F"
```

```
M3 <- matrix(nrow = 4, ncol = 5, 1:20)
M3
```

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

Alternative ways of creating matrices

```
# Example
v <- 1:5
x <- 6:10
vx1 <- cbind(v, x)  # combine by columns
vx1
```

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

```
vx2 <- rbind(v, x)  # combine by rows
vx2
```

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

Arrays

Arrays are the generalisation of vectors and matrices.

- A vector in R is a one dimensional array.
- A matrix is a two dimensional array.

```
# Example
z = c(1:24)      #   vector of length 24

# constructing a 3 by 4 by 2 array
a1 = array(z, dim = c(3, 4, 2))
a1
```

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

Data Frames

Data frame is the most convenient data structures in R for tabular data.

Data frames have multiple rows and multiple columns with different data types.

```
# Example
num1 = seq(1:5)
ch1 = c("A", "B", "C", "D", "E")
df1 = data.frame(ch1, num1)
df1
```

```
##   ch1 num1
## 1   A    1
## 2   B    2
## 3   C    3
## 4   D    4
## 5   E    5
```

```
colnames(df1) <- c('Character','Number')
df1
```

```
##   Character Number
## 1         A      1
## 2         B      2
## 3         C      3
## 4         D      4
## 5         E      5
```

Lists

A list is like generic vector containing other objects. They can have numerous elements any type and structure they can also be of different lengths

```
# Example
e1 = c(2, 3, 5) #element-1

e2 = c("aa", "bb", "cc", "dd", "ee")  #element-2

e3 = c(TRUE, FALSE, TRUE, FALSE, FALSE)#element-3

e4 = df1 #element-4 (previously constructed data frame)
lst1 = list(e1,e2,e3, e4)
str(lst1) # show the structure of lst1
```

```
## List of 4
##  $ : num [1:3] 2 3 5
##  $ : chr [1:5] "aa" "bb" "cc" "dd" ...
##  $ : logi [1:5] TRUE FALSE TRUE FALSE FALSE
##  $ :'data.frame':    5 obs. of  2 variables:
##   ..$ Character: chr [1:5] "A" "B" "C" "D" ...
##   ..$ Number   : int [1:5] 1 2 3 4 5
```

```
lst1[[1]]
```

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

Data Manipulation Functions

```
# dim (x)       dimensions of x
# str (x)       Structure of an object
# list (..)         create a list
# colnames (x)  set or find column names
# rownames (x)  set or find row names
# ncol(x)       number of columns
# nrow(z)       number of row
# rbind (..)        combine by rows
# cbind (..)        combine by columns
# is.na (x)         also is.null(x), is...
# na.omit (x)   ignore missing data
# rm ()             remove variables from workspace
# mean (x)
# sum (x)
# min (x)
```

```
    # max (x)
    # range (x)
    # summary (x)   depends upon x
    # sd (x)            standard deviation
    # cor (x)       correlation
    # cov (x)       covariance
    # solve (x)         inverse of x
```

In R, c() is used to both define a vector and a matrix. For instance, the following command returns a row vector with 5 elements:

```
y <- c( 5, exp(2), sign(-5), sqrt(9), pi)
y
```

```
## [1]  5.000000  7.389056 -1.000000  3.000000  3.141593
```

In the above example

- exp(.) is the exponential,
- sign(.) returns the sign,
- sqrt(.) is the square root ($\sqrt{}$) and
- pi represents π

This are build-in functions of R

The length of the above vector is obtained via the following command:

```
  length(y)
```

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

Vector Algebra

- An element-by-element multiplication of vector **a** by 5:

```
a <- c(-1, 2, -3); b <- c(2, 1, 2)
a
```

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

```
b
```

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

```
c=5*a
c
```

```
## [1]  -5  10 -15
```

- An element-by-element addition of vector **b** with 5:

```
c1=5+b
c1
```

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

- An element-by-element addition of two equal length vectors:

```
d=a+b
d
```

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

An element-by-element multiplication of vector **a** and **b**:

```
e=a*b
e
```

```
## [1] -2  2 -6
```

An element-by-element division of vector **a** over **b**:

```
f=a/b
f
```

```
## [1] -0.5  2.0 -1.5
```

The Colon/Sequence Notation

The colon notation **:** can be used to pick out selected rows, columns and elements of vectors, matrices, and arrays. It is a shortcut that is used to create row vectors.

```
v1=c(1:8); v1
```

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

```
v2=seq(-4, 2, 2); v2
```

```
## [1] -4 -2  0  2
```

```
v3=seq(0.1,0.6, 0.2); v3
```

```
## [1] 0.1 0.3 0.5
```

Extracting the 2nd, 3rd and 4th elements of v1

```
v4 = v1[c(2:4)]; v4
```

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

Extracting the 2nd, 5th and 8th elements of v1:

```
v1[seq(2,8,3)]
```

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

```
v1[c(2, 5, 8)]
```

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

Matrices

cbind rbind

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

```
M <- rbind ( c(1,2,3,4), c(5,6,7,8) )
M

colSums(M)
rowSums(M)
apply(M, 1, sum)
apply(M, 2, sum)

M2 <- cbind ( c(1,5), c(2,6), c(3,7), c(4,8))
M2

M3 <- matrix( c(1:8), nrow=2, ncol=4, byrow=T )
M3

M4 <- matrix( c(1:8), nrow=2, ncol=4, byrow=F )
M4
```

Create a (3×3) - matrix consisting solely of zeros

```
Z = matrix(0, ncol=3, nrow=3)
Z
```

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

Create a (3×3) - matrix consisting solely of ones

```
M5 = matrix(1, ncol=3,nrow=3)
M5
```

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

Create a four-dimensional identity matrix

```
K = diag(4)
K
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```
diag(K)
```

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

```
diag(diag(K))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```
n=4
A = diag(n)
noff = n*(n-1)/2

# runif - random draws from a uniform distribution
# rnorm - random draws from a normal distribution
# rexp - random draws from an exponential distribution
# rgeom - random draws from a geometric distribution
# rbeta - random draws from a beta distribution
# rt  - random draws from a t-distribution
# rbinom - random draws from a binomial distribution
# rchisq - random draws from a chi-square distribution

A[lower.tri(A, diag = FALSE)] = runif(noff, -1,1)
R = t(A)%*%A   # A'*A

C <- cov2cor(R)  ## Convert Covariance to Correlations
C
```

```
##             [,1]        [,2]       [,3]        [,4]
## [1,]  1.00000000 -0.00828893 -0.3380459 -0.67631522
## [2,] -0.00828893  1.00000000  0.3997856  0.01759952
## [3,] -0.33804590  0.39978564  1.0000000  0.48177242
## [4,] -0.67631522  0.01759952  0.4817724  1.00000000
```

Matrix Algebra

Matrix Addition

```
A <- matrix( c(1:16), nrow = 4)
B <- matrix( rep(2^c(1:4), 4), nrow = 4)

diag(A)
```

```
## [1]  1  6 11 16
```

```
diag(diag(A))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    6    0    0
## [3,]    0    0   11    0
## [4,]    0    0    0   16
```

```
C <- A + B; C
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3    7   11   15
## [2,]    6   10   14   18
## [3,]   11   15   19   23
## [4,]   20   24   28   32
```

Matrix Subtraction

```
D <- A - B; D
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -1    3    7   11
## [2,]   -2    2    6   10
## [3,]   -5   -1    3    7
## [4,]  -12   -8   -4    0
```

Matrix Multiplication (element-by-element multiplication - dot-product)

```
E <- A*B; E
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2   10   18   26
## [2,]    8   24   40   56
## [3,]   24   56   88  120
## [4,]   64  128  192  256
```

Matrix Multiplication

```
F <- A %*% B; F
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  302  302  302  302
## [2,]  332  332  332  332
## [3,]  362  362  362  362
## [4,]  392  392  392  392
```

Determinant of Matrix

```r
G <- matrix(c(1:4), nrow = 2); G
```

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

```r
det(G)
```

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

Inverse Of A Matrix

```r
solve(G)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

Extract Rows of a Matrix

```r
# Extract the first row of A
A[1,]
```

```
## [1]  1  5  9 13
```

```r
# Extract the second and fourth rows of A
A[c(2,4), ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    6   10   14
## [2,]    4    8   12   16
```

Extract Columns of a Matrix

```r
# Extract the second and third columns of A
A[, c(2,3)]
```

```
##      [,1] [,2]
## [1,]    5    9
## [2,]    6   10
## [3,]    7   11
## [4,]    8   12
```

Extract Rows and Columns

```
# Extract the 2nd and 3rd row with the 1st and 4th column
A[ c(2,3), c(1,4) ]
```

```
##      [,1] [,2]
## [1,]    2   14
## [2,]    3   15
```

Replicate Columns

```
# Replicate the first column of A 4 times
A1 <- A[, rep(1,4)]
A1
```

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

Matrix Representation: Example

Airfares in dollars for direct flights between the cities.

```
A <- matrix(c(NA,80,NA,30,NA,80,
              80,NA,80,NA,NA,NA,
              NA,80,NA,NA,40,50,
              30,NA,NA,NA,80,60,
              NA,NA,40,80,NA,50,
              80,NA,50,60,50,NA), nrow = 6)
rownames(A) <- colnames(A) <- c("T", "U", "V", "W", "X", "Y")
A
```

```
##    T  U  V  W  X  Y
## T NA 80 NA 30 NA 80
## U 80 NA 80 NA NA NA
## V NA 80 NA NA 40 50
## W 30 NA NA NA 80 60
## X NA NA 40 80 NA 50
## Y 80 NA 50 60 50 NA
```

```
colSums(A, na.rm = T)
```

```
##   T   U   V   W   X   Y
## 190 160 170 170 170 240
```

```
rowSums(A, na.rm = T)
```

```
##   T   U   V   W   X   Y
## 190 160 170 170 170 240
```

```
apply(A, 1, sum, na.rm=T)
```

```
##   T   U   V   W   X   Y
## 190 160 170 170 170 240
```

```
apply(A, 2, sum, na.rm=T)
```

```
##   T   U   V   W   X   Y
## 190 160 170 170 170 240
```

```
France <- c(0, 62254, 55196, 190160, 255147)
Germany <- c(0, 0, 0, 0, 0)
Italy <- c(0, 0, 0, 0, 0)
UK <- c(101341, 110244, 22573, 0, 144163)
US <- c(20686, 31449, 5850, 74733, 0)

Flows <- rbind( France, Germany, Italy, UK, US)
Flows
```

```
##           [,1]   [,2]  [,3]   [,4]   [,5]
## France       0  62254 55196 190160 255147
## Germany      0      0     0      0      0
## Italy        0      0     0      0      0
## UK      101341 110244 22573      0 144163
## US       20686  31449  5850  74733      0
```

```
# Change the column names
colnames(Flows) <- rownames(Flows)
```

Find 1. Top lender 2. Top Borrower 3. Net-lending = Lending - Borrowing

```
Lending <- rowSums(Flows)
Lending
```

```
##  France Germany   Italy      UK      US
##  562757       0       0  378321  132718
```

```
## Alternative
Lending.1 <- apply(Flows, 1, sum)
Lending.1
```

```
##  France Germany   Italy      UK      US
##  562757       0       0  378321  132718
```

```
Borrowing <- colSums(Flows)
Borrowing
```

```
##  France Germany   Italy      UK      US
##  122027  203947   83619  264893  399310
```

```r
## Alternative
Borrowing.1 <- apply(Flows, 2, sum)
Borrowing.1
```

```
##  France Germany   Italy      UK      US
##  122027  203947   83619  264893  399310
```

```r
Net.Lending <- Lending - Borrowing
Net.Lending
```

```
##  France Germany   Italy      UK      US
##  440730 -203947  -83619  113428 -266592
```

R Datasets

RStudio comes with some datasets for new users

```r
    # Example. To see all the available built-in data sets, use
    data()


    rm(list = ls(all = TRUE))
    graphics.off()
    cat("\014")
```

```
data(women)
women
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

```
nrow(women)     # number of rows
```

```
## [1] 15
```

```
ncol(women)     # number of columns
```

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

```
summary(women)  # summary of data
```

```
##      height         weight
##  Min.   :58.0   Min.   :115.0
##  1st Qu.:61.5   1st Qu.:124.5
##  Median :65.0   Median :135.0
##  Mean   :65.0   Mean   :136.7
##  3rd Qu.:68.5   3rd Qu.:148.0
##  Max.   :72.0   Max.   :164.0
```

```
str(women)      # structure of data
```

```
## 'data.frame':    15 obs. of  2 variables:
##  $ height: num  58 59 60 61 62 63 64 65 66 67 ...
##  $ weight: num  115 117 120 123 126 129 132 135 139 142 ...
```

```
dim(women)      # dimensions of a data set
```

```
## [1] 15  2
```

```
head(women)     # display first n rows (elements)
```

```
##   height weight
## 1     58    115
## 2     59    117
## 3     60    120
## 4     61    123
## 5     62    126
## 6     63    129
```

```
tail(women)     # display last n rows (elements)
```

```
##    height weight
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

```
str(women)      # summarize structure of an object
```

```
## 'data.frame':    15 obs. of  2 variables:
##  $ height: num  58 59 60 61 62 63 64 65 66 67 ...
##  $ weight: num  115 117 120 123 126 129 132 135 139 142 ...
```

```
# swiss dataframe has standardized fertility measure and
# socio-economic indicators for each of 47 French-speaking
# provinces of Switzerland at about 1888.

data(swiss)
str(swiss)
```

```
## 'data.frame':    47 obs. of  6 variables:
##  $ Fertility       : num  80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 ...
##  $ Agriculture     : num  17 45.1 39.7 36.5 43.5 35.3 70.2 67.8 53.3 45.2 ...
##  $ Examination     : int  15 6 5 12 17 9 16 14 12 16 ...
##  $ Education       : int  12 9 5 7 15 7 7 8 7 13 ...
##  $ Catholic        : num  9.96 84.84 93.4 33.77 5.16 ...
##  $ Infant.Mortality: num  22.2 22.2 20.2 20.3 20.6 26.6 23.6 24.9 21 24.4 ...
```

```
#using names and row.names
names(swiss)  #name of the columns (can also use colnames)
```

```
## [1] "Fertility"        "Agriculture"      "Examination"      "Education"
## [5] "Catholic"         "Infant.Mortality"
```

```
colnames(swiss)
```

```
## [1] "Fertility"        "Agriculture"      "Examination"      "Education"
## [5] "Catholic"         "Infant.Mortality"
```

```
    row.names(swiss)  #name of the rows
```

```
##  [1] "Courtelary"   "Delemont"     "Franches-Mnt" "Moutier"      "Neuveville"
##  [6] "Porrentruy"   "Broye"        "Glane"        "Gruyere"      "Sarine"
## [11] "Veveyse"      "Aigle"        "Aubonne"      "Avenches"     "Cossonay"
## [16] "Echallens"    "Grandson"     "Lausanne"     "La Vallee"    "Lavaux"
## [21] "Morges"       "Moudon"       "Nyone"        "Orbe"         "Oron"
## [26] "Payerne"      "Paysd'enhaut" "Rolle"        "Vevey"        "Yverdon"
## [31] "Conthey"      "Entremont"    "Herens"       "Martigwy"     "Monthey"
## [36] "St Maurice"   "Sierre"       "Sion"         "Boudry"       "La Chauxdfnd"
## [41] "Le Locle"     "Neuchatel"    "Val de Ruz"   "ValdeTravers" "V. De Geneve"
## [46] "Rive Droite"  "Rive Gauche"
```

```
    swiss$Fertility   #returns the data in the column Fertility
```

```
##  [1] 80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 87.1 64.1 66.9 68.9 61.7
## [16] 68.3 71.7 55.7 54.3 65.1 65.5 65.0 56.6 57.4 72.5 74.2 72.0 60.5 58.3 65.4
## [31] 75.5 69.3 77.3 70.5 79.4 65.0 92.2 79.3 70.4 65.7 72.7 64.4 77.6 67.6 35.0
## [46] 44.7 42.8
```

Creating Functions in R

The advantage of a programming language like R is that you have the flexibility to write your own functions. For instance

```
myfunction <- function(inputs){
  #
  operations
  #
  return(output)
}
```

- The **function()** can include several arguments
- The **return()** is the output of the function that can include only one object, although that object might include several elements
- To call the function you type in R the name of the function with the appropriate arguments: **myfunction()**

Example:

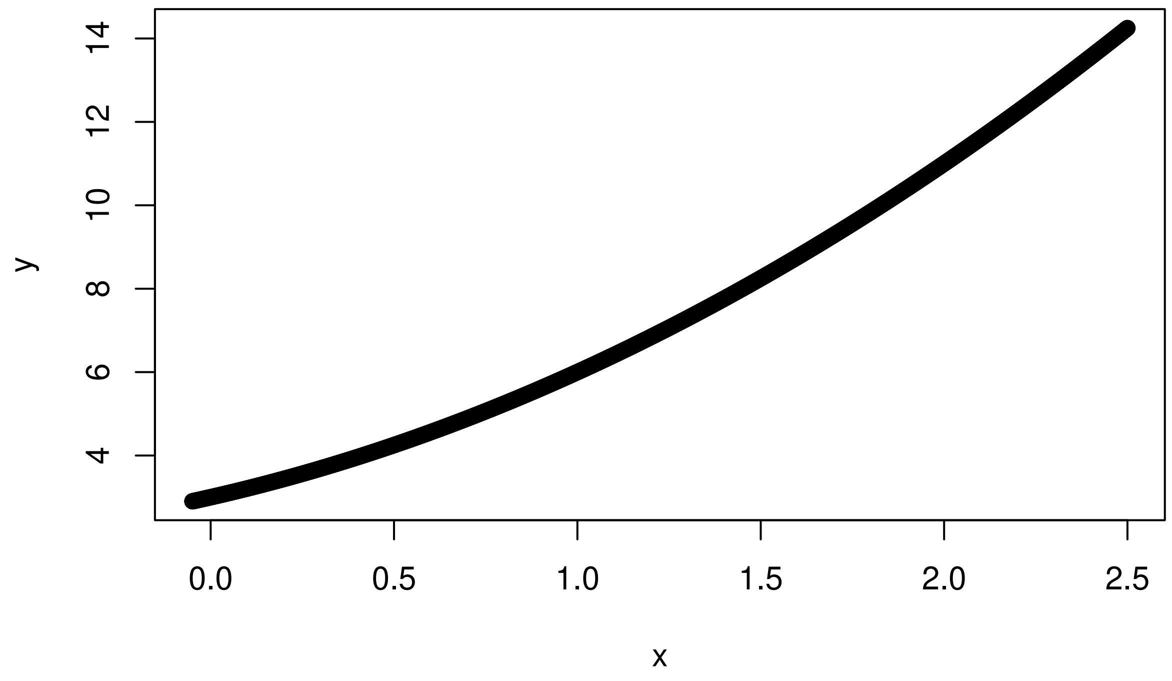
Suppose we want to write a function for the equation of a line:

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

```
function.name <- function(inputs){
  #' Type in your function
  # Example : y = f(x) = x^2 + 2x + 3
}

Yfun <- function(x){
  y <- x^2 + 2*x + 3
  return(y)
}

x <- seq(-0.05,2.5, 0.001)
y <- Yfun(x)
plot(x, y)
```



Example:

Write a function that computes the mean of a vector

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

```
X.bar <- function(x){
  n <- length(x)
  sum.x <- sum(x)
  xbar <- (1/n)*sum.x
  return(xbar)
}

set.seed(123)
x <- rnorm(1000, mean = 10, sd=10)

X.bar(x)
```

```
## [1] 10.16128
```

```r
mean(x)
```

```
## [1] 10.16128
```

Example:

Write a function that computes the Variance of a vector

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

```r
VarX <- function(x){
  x <- na.omit(x)
  n <- length(x)
  xbar <- X.bar(x)
  sq.dev <- (x - xbar)^2
  vx <- (1/(n-1))*sum(sq.dev)
  return(vx)
}

VarX.P <- function(x){
  x <- na.omit(x)
  n <- length(x)
  xbar <- X.bar(x)
  sq.dev <- (x - xbar)^2
  vx <- (1/(n))*sum(sq.dev)
  return(vx)
}
```

Using Expectations

$$Var(X) = E[(X - \mu)^2]$$

```r
Var.Exp <- function(x){
  x <- na.omit(x)
  mu = X.bar(x)
  SD = (x-mu)^2
  Vx = X.bar(SD)
  return(Vx)
}

var(x)
```

```
## [1] 98.34589
```

```r
VarX(x)
```

```
## [1] 98.34589
```

```
Var.Exp(x)
```

```
## [1] 98.24755
```

```
VarX.P(x)
```

```
## [1] 98.24755
```

Skewness is defined as

$$Skew(X) = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$$

Skew(X) can be expressed in terms of the first three moments of X

$$Skew(X) = \frac{E(X^3) - 3\mu E(X^2) + 2\mu^3}{\sigma^3}$$

```
Skew.Ex <- function(x)
{
  x = na.omit(x)
  mu = X.bar(x)
  sig = sqrt(VarX(x))
  Sx = (x - mu)/sig
  Sk = X.bar(Sx^3)
  return(Sk)
}


Skew.Ex(x)
```

```
## [1] 0.065196
```

```
skew(x)
```

```
## [1] 0.065196
```

The Kurtosis

$$kurt(X) = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$$

Kurtosis can be expressed in terms of the first four moments of X

$$kurt(X) = \frac{E(X^4) - 4\mu E(X^3) + 6\mu^2 E(X^2) - 3\mu^4}{\sigma^4}$$

```r
Kurt.Ex <- function(x)
{
  x = na.omit(x)
  mu = X.bar(x)
  sig = sqrt(VarX(x))
  Sx = (x - mu)/sig
  Kt = X.bar(Sx^4)
  return(Kt)
}


kurtosi(x) + 3 ## Excess kurtosis,  Ex.kurt = kurtosis - 3
```

```
## [1] 2.919898
```

```r
Kurt.Ex(x)
```

```
## [1] 2.919898
```

For-Loops in R

The syntax in *R* to implement a *for* loop is as follows:

```r
# for (i in 1:N)
# {
# ## write your commands here
# }
```

Example:

Let's consider an example of the use of the for loop that demonstrates the validity of the Central Limit Theorem (CLT). We are going to do this by simulation, which means that we simulate data and calculate some statistic of interest and repeat these operations a large number of times.

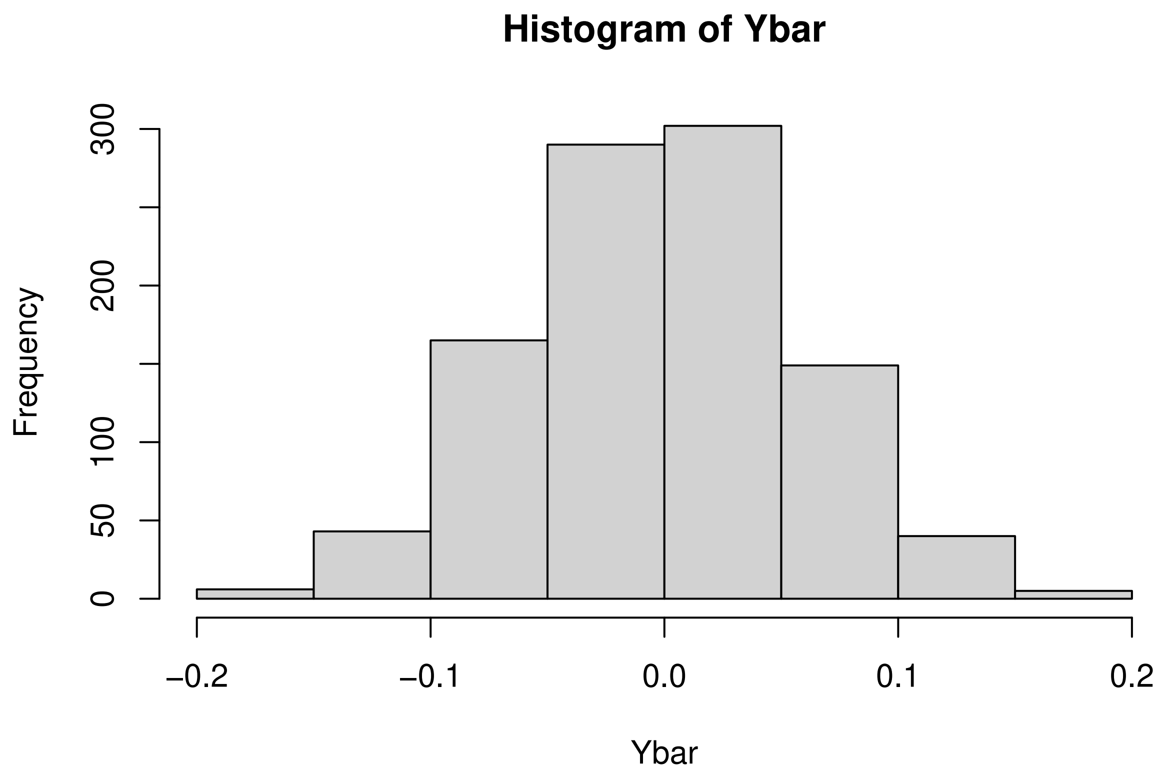
In particular, we want to demonstrate that, no matter how the data are distributed, the sample mean is normally distributed with mean the population mean and variance given by σ^2/N, where σ^2 is the population variance and N is the sample size. We assume that the population distribution is $\mathcal{N}(0,4)$ and we want to repeat a large number of times the following operations:

1. Generate a sample of length N
2. Calculate the sample mean
3. Repeat 1-2 S times

```r
S = 1000 # set the number of simulations
N = 1000 # set the length of the sample
mu = 0 # population mean
sigma = 2 # population standard deviation
Ybar = vector('numeric', S) # create an empty vector of S elements
# to store the t-stat of each simulation
for (i in 1:S)
{
```

```r
Y = rnorm(N, mu, sigma) # Generate a sample of length N
Ybar[i] = mean(Y) # store the t-stat
}


hist(Ybar)
```



```r
c(mean(Ybar), sd(Ybar))
```

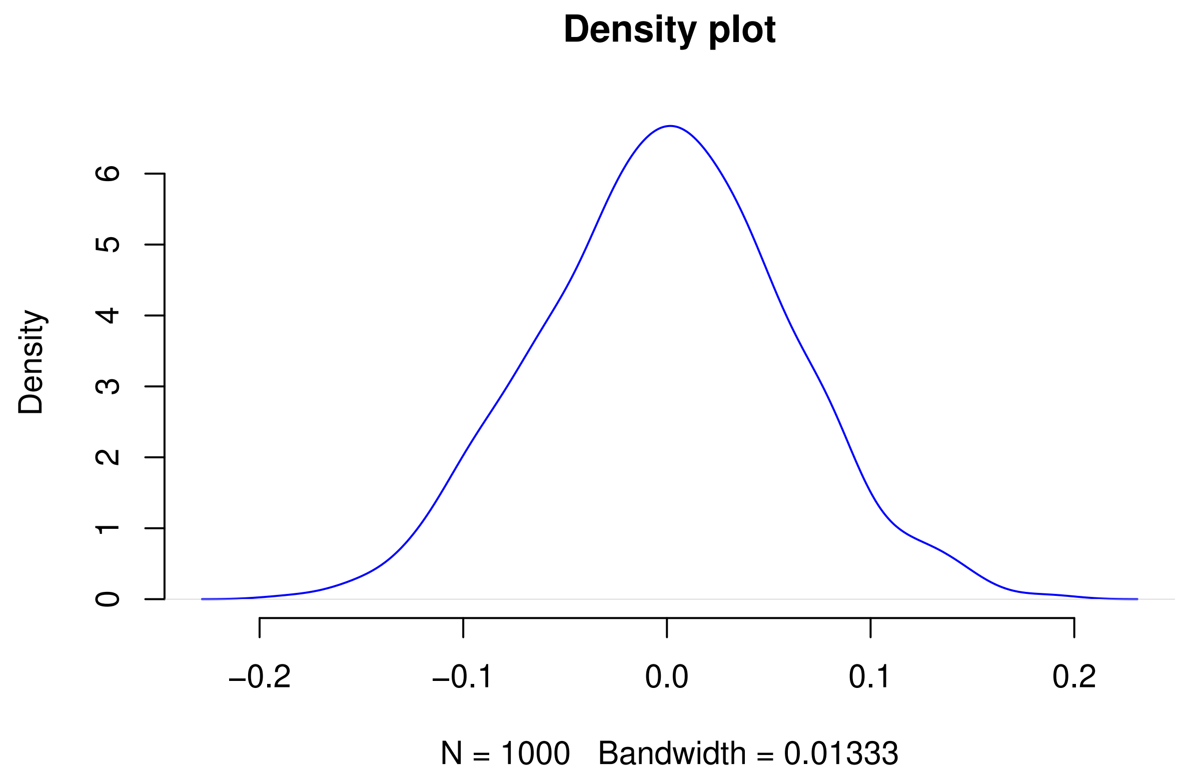
```
## [1] -0.001006151  0.060170076
```

```r
sqrt(sigma^2/N)
```

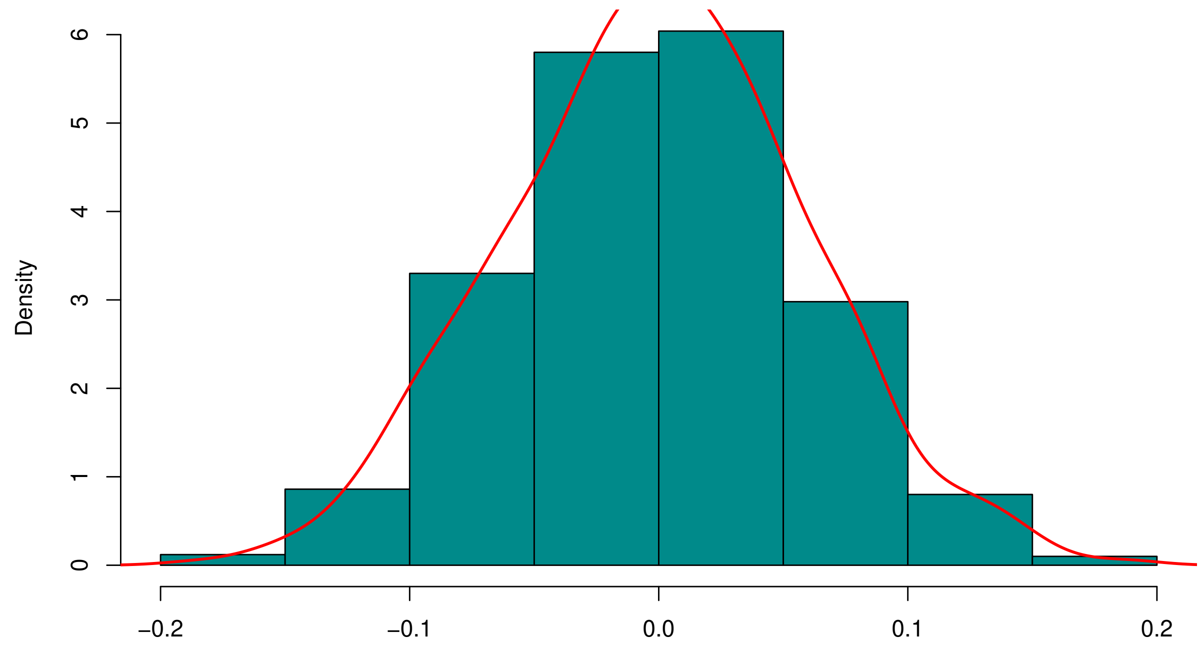
```
## [1] 0.06324555
```

```r
den <- density(Ybar)

plot(den, frame = FALSE, col = "blue",main = "Density plot")
```



```
hist(Ybar, col="darkcyan", border="black", prob=TRUE, xlab = NA,  main = NA)
lines(density(Ybar), lwd = 2, col = "red")
```



Functions with conditional execution

Conditional Statements

1. **if**, **if else**
2. **while**

An **if** statement allows you to conditionally execute code. For instance

```
# if (condition) {
#   # code executed when condition is TRUE
# } else {
#   # code executed when condition is FALSE
# }
```

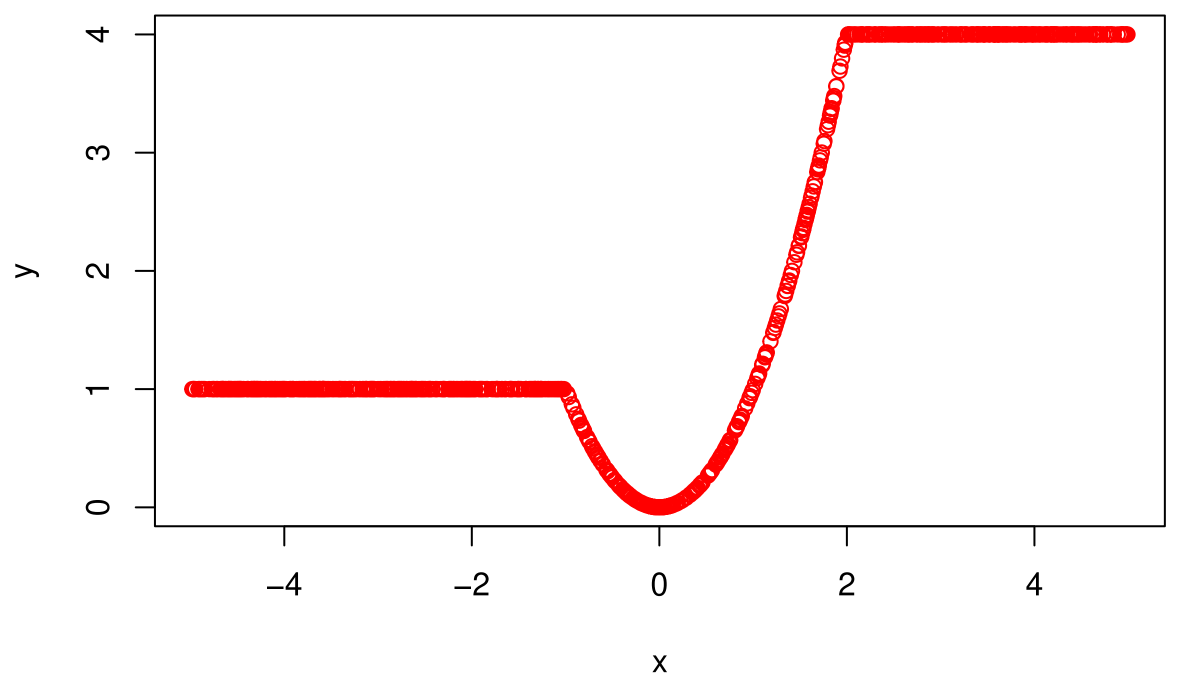
If-Else Statements in R

Example

$$y = \begin{cases} 1 & \text{if} \quad x < -1 \\ x^2 & \text{if} \quad -1 \le x \le 2 \\ 4 & \text{if} \quad x > 2 \end{cases}$$

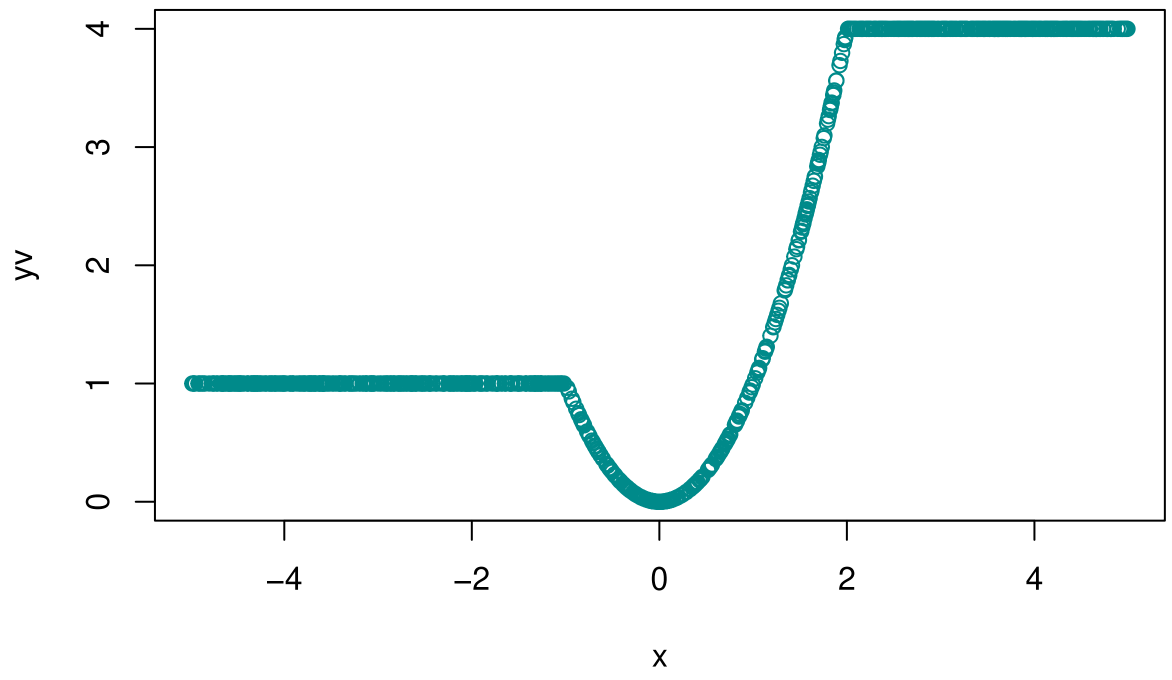
```r
Fx <- function(x)
{
  if(x < -1){
    y <- 1
  } else if(x >= -1 && x<= 2){
    y <- x^2
  } else if (x > 2){
    y <- 4
  }
  return(y)
}

x <- as.matrix(runif(1000, -5, 5))
y <- as.matrix(apply(x, 1, Fx))
plot(x,y, col="red")
```



For-Loop

```r
## Alternative (For-Loop)
yv <- 0*x
for (i in 1:length(x)) {
  yv[i] <- Fx(x[i])
}
plot(x,yv, col="darkcyan")
```



While Loop in R

A loop is a statement that keeps running until a condition is satisfied. The syntax for a while loop is the following:

```
#  while (condition) {
#     Exp
# }
```

```
Fxn <- function(x) x^3 - 3*x + 5
Fxn(x=-5)
```

```
## [1] -105
```

```
x<- -5
count <- 0
while(Fxn(x) < 0){
  x <- x + 0.0001
  count <- count + 1
}
x
```

```
## [1] -2.279
```

```
Fxn(x)
```

```
## [1] 0.000236361
```