R Programming

1. Comments within a code can be done using #

#This is a comment

log(1)

1. Always document your code.
2. Variable assignment

#declare a variable x and assign a value to it

x = 1

In the memory, the value of x gets stored as 1 and can be used for iterations.

1. Print multiple lines on the console at the same time

x;y

if x =1 and y =2

Output:

1

2

1. Object types in R:
   1. Characters: using single or double quotations
   2. Integers:
   3. Logical variables: TRUE or FALSE, T/F, 1/0
2. Concatenation

example\_vector\_char <- c('the','cat','in','the','hat')

example\_vector\_char <- c('the','cat','in',1,'the','hat') # will store it as ‘1’

1. Subset a vector and select a few elements by using a square bracket

example\_vector\_char <- c('the','cat','in','the','hat')

example\_vector\_char[1:3]

Output:

|  |
| --- |
| [1] "the" "cat" "in" |
|  |
| |  | | --- | |  | |

1. For matrix:

Rows, columns

example\_vector\_char[1:3,1:2]

1. Sort command

sort(unique(example\_vector\_char)) # sorts the string with only the unique values. In this case the word “the” is repeated and it gets sorted only once.

output:

|  |
| --- |
| [1] "cat" "hat" "in" "the" |
|  |
| |  | | --- | |  | |

sort(example\_vector\_char)

Output:

|  |
| --- |
| [1] "cat" "hat" "in" "the" "the" |
|  |
| |  | | --- | |  | |

1. Create a character string using the paste command

paste('file',1:5,sep='\_')

Output:

|  |
| --- |
| [1] "file\_1" "file\_2" "file\_3" "file\_4" "file\_5" |
|  |
| |  | | --- | |  | |

1. Number of characters in a string

nchar(example\_vector\_char) #number of characters

Output:

|  |
| --- |
| [1] 3 3 2 3 3 |
|  |
| |  | | --- | |  | |

1. R is a vectorized command language, it applies the nchar command to every output
2. Use the paste command to make the input into 1 string

paste(example\_vector\_char,collapse = ' ') #paste elements together

paste(example\_vector\_char,sep = ' ') #paste elements separately

rev(example\_vector\_char) #reverse ordering

Output:

|  |
| --- |
| [1] "the cat in the hat"  > paste(example\_vector\_char,sep = ' ') #paste elements together  [1] "the" "cat" "in" "the" "hat"  > rev(example\_vector\_char) #reverse ordering  [1] "hat" "the" "in" "cat" "the" |
|  |
| |  | | --- | |  | |

1. To replace a character from a string

x <- gsub('h','',example\_vector\_char) #replace h with nothing in all elements of example\_vector\_char

Output:



1. Everything is in the regular expression format. Finding the index of the

Output:

> grepl('h',example\_vector\_char) #index of those with h

[1] TRUE FALSE FALSE TRUE TRUE

Find the last h inside a string using the $ sign:

> grepl('$h',example\_vector\_char) #index of those with h

[1] FALSE FALSE FALSE FALSE FALSE

For more variations in the pattern: use Stackoverflow – Regex R

1. To know more about any command use the ?<command> on the console.

?grepl

1. Give the index value use the grep command:

> grep('h',example\_vector\_char) #index of those with h

[1] 1 4 5

1. Which command:

grep('h',example\_vector\_char) #index of those with h

which(grepl('h',example\_vector\_char)) #index of those with h

which(example\_vector\_char == "cat")

Output:

> which(grepl('h',example\_vector\_char)) #index of those with h)

[1] 1 4 5

> which(example\_vector\_char == "cat")

[1] 2

1. Substring

> substr(example\_vector\_char,1,1) #Substring from character 1 to 1 of the string

[1] "t" "c" "i" "t" "h"

1. Sequences with Numeric entries

Use seq – mention the starting point and the ending point

By – how do you want to make the string

example\_vector\_numeric <- seq(1,1000,by = 0.1)



1. View the first few rows using the head command

> head(example\_vector\_numeric,10)

[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9

> head(example\_vector\_numeric)

[1] 1.0 1.1 1.2 1.3 1.4 1.5

1. Tail Command to see the last few entries

> tail(example\_vector\_numeric,10)

[1] 999.1 999.2 999.3 999.4 999.5 999.6 999.7 999.8 999.9 1000.0

1. Perform arithmetic operations
   1. Sum
   2. Cumsum
   3. Mean
   4. Median
   5. Quantile
   6. Summary

> sum(example\_vector\_numeric)

[1] 5000496

> tail(cumsum(example\_vector\_numeric))

[1] 4995497 4996496 4997496 4998496 4999496 5000496

> mean(example\_vector\_numeric)

[1] 500.5

> median(example\_vector\_numeric)

[1] 500.5

> quantile(example\_vector\_numeric,0.25)

25%

250.75

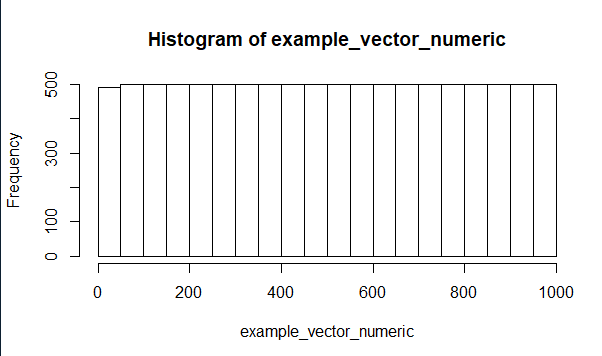
> summary(example\_vector\_numeric)

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.0 250.8 500.5 500.5 750.2 1000.0

1. Missing values will be shown as NA
2. Make a histogram using hist command

hist(example\_vector\_numeric)



1. Logical check using the which command

> which(example\_vector\_numeric == 434)# == for a logical check

[1] 4331

1. Use the ?plot command to get more plotting options
2. Plot

> plot(example\_vector\_numeric,typ = 'l',xlab = 'X-AXIS LfdasL',

+ ylab = 'Y-AXIS LABEL',main ='MAIN LABEL',col = 2)

Typ: l is for a line plot

Col=2 for red

3=green

4=blue

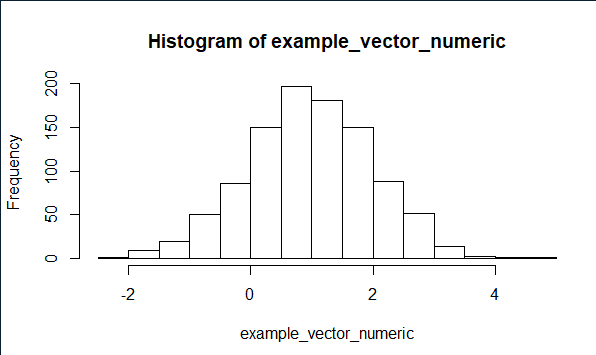
Use http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf for the various list of colors in R to create new color combinations for display.

1. Use random normal distribution to obtain randomly generated variables

example\_vector\_numeric <- rnorm(1000,mean = 1,sd = 1) #Thousand standard normal draws

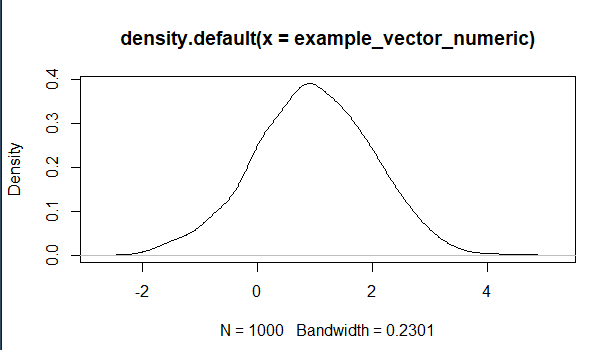


hist(example\_vector\_numeric)



1. Density plots

plot(density(example\_vector\_numeric))



1. Dot product

> c(1,2,3,4) %\*% matrix(c(1,2,3,4,5,6,7,8),4,2)

[,1] [,2]

[1,] 30 70

> example\_vector\_numeric %\*% example\_vector\_numeric\_2

[,1]

[1,] 19470.49

1. Element wise multiplication across the two vectors

#Element wise operations

example\_vector\_numeric \* example\_vector\_numeric\_2

output:

|  |
| --- |
| [921] 5.068172e+01 1.877999e+01 1.193617e+01 1.498183e+01 1.978532e+01  [926] 1.157422e+01 4.847390e+01 2.010899e+01 2.584714e+01 3.339706e+01  [931] 3.146540e+01 2.073447e+01 -1.759905e+01 1.559350e+01 4.575517e+01  [936] 3.166519e+01 1.426238e+01 3.483007e+01 3.305784e+01 -1.829693e+01  [941] 4.587529e-01 1.988533e+01 1.513014e+00 -3.208682e+00 2.830000e+01  [946] 3.696610e+01 3.856930e+01 8.001866e+00 -3.138473e+01 2.685768e+01  [951] 1.243879e+01 3.691235e+01 4.447830e+01 -1.971026e-02 3.695823e+01  [956] 3.677376e+01 -1.485755e+00 3.827877e+01 3.568365e+01 4.529272e+00  [961] 4.517918e+00 1.809853e+01 2.899796e+01 2.632533e+01 -1.016059e+01  [966] 2.255278e+01 1.741878e+01 -3.077305e+00 3.865938e+01 1.023818e+01  [971] 5.538734e+01 5.695764e+00 2.084473e+01 2.096460e+01 2.556515e+01  [976] 5.901677e+00 2.965386e+01 3.513527e+01 -1.735030e+01 1.688698e+01  [981] 2.891953e+01 3.419222e+01 2.933557e+01 9.497775e+00 1.105078e+01  [986] 1.609793e+01 9.694464e+00 2.174661e+01 1.983511e+01 2.782573e+01  [991] 6.240284e+01 -8.649072e-01 2.389720e+01 3.815515e+01 3.886451e+01  [996] -1.041800e+01 1.729738e+01 6.281131e+00 1.711308e+01 1.412760e+01 |
|  |
| |  | | --- | |  | |

1. Logical operations

> T == TRUE

[1] TRUE

> T == F

[1] FALSE

> F == FALSE

[1] TRUE

> T != F # ! indicates Not

[1] TRUE

> F != F

[1] FALSE

> T + T #True is coded as a 1 in R and F is coded as a 0

[1] 2

> F + T

[1] 1

> F + F

[1] 0

> T\*T

[1] 1

> T | F# Vertical bar is an OR statement -> One needs to be true

[1] TRUE

> F | F

[1] FALSE

> T & F #& represents an AND statement -> both need to be true

[1] FALSE

> T & T

[1] TRUE

> F & F

[1] FALSE

1. Sample: shuffle the data
2. Rep: use to repeat a string
3. as.factor: convert a character to a factor.

> ##Factors: Reduce memory in your system

> example\_factor <- sample(rep(c('boy','girl'),100))

> head(example\_factor)

[1] "girl" "boy" "girl" "girl" "boy" "girl"

> class(example\_factor)

[1] "character"

> example\_factor <- as.factor(example\_factor)

> head(example\_factor)

[1] girl boy girl girl boy girl

Levels: boy girl

> class(example\_factor)

[1] "factor"

> levels(example\_factor)

[1] "boy" "girl"

> nlevels(example\_factor)

[1] 2

> as.numeric(example\_factor) #Will return the factor level

[1] 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 1 1 1 2 1 1 2 1 1 1 2 2 1 1 2 2 1 2 1 1 2 2 1 2

[41] 1 2 1 2 1 2 2 1 2 2 1 2 1 2 2 2 2 1 2 1 1 1 2 1 2 1 2 1 1 2 1 2 2 2 1 2 1 1 2 1

[81] 2 2 1 2 1 1 2 1 2 2 2 2 1 1 1 2 1 1 2 1 1 1 1 1 2 1 2 2 1 2 2 2 2 1 2 1 2 2 1 1

[121] 1 2 2 2 2 1 1 1 2 1 1 1 1 1 1 2 1 1 2 2 1 1 1 2 1 2 2 2 1 1 1 1 1 2 2 2 2 2 2 1

[161] 2 1 1 2 2 2 2 2 1 1 2 2 1 1 1 2 1 2 2 2 1 1 1 1 2 2 2 2 1 2 1 1 1 1 1 1 1 1 1 2

> as.numeric(as.factor(c('2','32','2','1','3')))#WRONG

[1] 2 4 2 1 3

> as.numeric(as.character(as.factor(c('2','32','2','1','3'))))#Hurray

[1] 2 32 2 1 3

1. Matrix:
2. Functions

my.function <- function(x,y){

(x + y)^2

}

my.function(1000,1000)

options(scipen = 10)

Output:

> my.function(1000,1000)

[1] 4e+06

> options(scipen = 10)

> my.function(1000,1000)

[1] 4000000

1. Scoping: create local variables that resides only inside the function. Use them for calculations and then return the value.

> andyFunction <- function(x,y){

+ gen\_dat\_1 <- rnorm(1000,x,x^2)

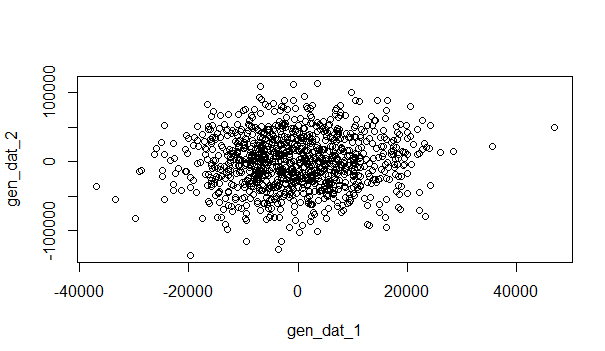
+ gen\_dat\_2 <- rnorm(1000,y,y^2)

+ plot(gen\_dat\_1,gen\_dat\_2)

+ return(sum(ifelse(gen\_dat\_1 > gen\_dat\_2,T,F)))

+ }

> res <- andyFunction(100,200)



> res

[1] 508

Mean of x and SD of X2, with 1000 values

How many elements of gen\_dat\_1 are greater than gen\_dat\_2

1. Conditional Statements

> ##Conditional Statment

> if((2+2) == 5){

+ #DO THIS

+ print('HI')

+ }else{

+ #DO SOMETHING ELSE

+ print('NO')

+ }

[1] "NO"

> #Can also be simplified quickly to

> ifelse(4> 3,'HI','NO')

[1] "HI"

1. For Loop: Create a temp variable till the condition is satisfied

> #####Loops

> #For

> for(cat in 1:4){

+ if(cat == 2){

+ print('YO')

+ }

else{

+ print(cat\*2)

+ }

+ }

[1] 2

[1] "YO"

[1] 6

[1] 8

1. A function to a loop

> #Apply family

> sapply(1:4,function(x)print(x)) #apply a function to a loop

[1] 1

[1] 2

[1] 3

[1] 4

[1] 1 2 3 4

1. Set your working directory

getwd()

setwd("C:/Users/Vidhi Patel/Desktop/workshop/Claremont-master/Claremont-master/workshop1"

)

install.packages("magrittr")

html <- read\_html("https://www.basketball-reference.com/teams/MIA/2019.html")

install.packages("rvest")

library("rvest")

1. Get a copy of your html file

> write\_xml(html,"bbref\_MIA\_2019.html")

> html %>%

+ write\_xml("bbref\_MIA\_2019.html")

The copy of the file will be saved to the working directory and you can open the html there.

This can help in obtaining stock data which can get refreshed eventually.

1. You can download all the team information by using the for loop.

teams = c("MIA", "BHK", "LAC")

years = c(2015:2019)

for (year in c(years)){

for (team in c(teams)){

filename = paste("bbref",team,"/",year,".html",sep = "")

paste("https://www.basketball-reference.com/teams/",team,"/",year,".html",sep="")%>%

read\_html()%>%

write\_xml(filename)

Sys.sleep(runif(1,4,6))

}

}