Ashutosh Sharma & 200102014

DSA Individual Assignment

Submitted To:

Prof. (Dr.) Sridhar Vaithianathan,

Associate Professor (Analytics).

IMT - Hyderabad.

Date of submission : 19-12-2020

Table of Contents

1. Basic R codes and arithmetic operations

2. Data Types

3. Functions in R

4. Vectors

5. Data Structures

6. Factors

7. Basic info about Vectors, Arrays

8. Data Structures

9. Matrices

10. Arrays

11. Reading Data

12. Built-in Data Sets

13. Summary Statistics

14. Correlation

15. Hypothesis Testing

200102014

Ashutosh Sharma

17/12/2020

# Creating Simple Objects and Doing Mathematical Calculation  
v = 5  
# 1. Command Line Interface  
v

## [1] 5

z = 10  
z

## [1] 10

# 2. object need not be explicitly defined.  
v = 5  
class(v)

## [1] "numeric"

v = "Hello"  
class(v)

## [1] "character"

v = TRUE  
class(v)

## [1] "logical"

v = FALSE  
class (v)

## [1] "logical"

# Object Assignments and Simple Calculations  
v = 10  
w = 15  
v+w

## [1] 25

v-w

## [1] -5

v\*w

## [1] 150

v/w

## [1] 0.6666667

sqrt(v)

## [1] 3.162278

v^w

## [1] 1e+15

exp(v)

## [1] 22026.47

log(v, base=exp(1))

## [1] 2.302585

log10(v)

## [1] 1

factorial(v)

## [1] 3628800

cos(v)

## [1] -0.8390715

abs(v)

## [1] 10

There is no need for declaring the variables like in other languages

## TypesofData

# line by Line Execution of command - Compiler  
# Not explicitly declaring variables.  
  
#A = 10  
#Variable /Object -- > A (Case Sensitive)  
#Value = 10  
#Read from right to left.  
# <- or = # Assignment.  
# Simple Mathematical Operations.  
# Remove the objects or variables created.  
  
# DATA TYPES. (Nominal , Ordinal, Interval and Ratio)  
# Self (NOIR) and System (Numeric, Character, Logical, Date, Vector). (Two Brains).  
  
# DATA TYPES  
x = 10  
class(x)

## [1] "numeric"

# Numeric - Integer and Decimal - (R)- Integer (Whole Number) and Numeric (Float - Decimal)  
i = 5L # L - Integer  
class(i)

## [1] "integer"

is.integer(i)

## [1] TRUE

is.numeric(x)

## [1] TRUE

# Character - Categorical Variable - Words/String (Nominal), Classification (Gender - Male , Female)  
s = "R\_Studio"  
class(s)

## [1] "character"

# Levels of Classification - Factor --- Involves levels.(Ordinal)   
# Eg: Edu Quali - X, XII, Graduation, Post Graduation (4 Levels)  
  
# Logical - TRUE (1) and FALSE (0)  
TRUE \* 5

## [1] 5

FALSE \* 5

## [1] 0

K = TRUE  
class(K)

## [1] "logical"

is.logical(K)

## [1] TRUE

# Date - Starting Date (1970) - Numeric Value.  
# In R - 1 Jan 1970  
# Date - mm/dd/yyyy  
# POSIXct - Date plus Time.  
  
date1 = as.Date("2012-06-28")  
# as.Date()# Auto complete # How to enter  
# ? as.Date # help  
date1

## [1] "2012-06-28"

class (date1)

## [1] "Date"

as.numeric(date1)

## [1] 15519

#POSIXct - Date and Time  
date2 = as.POSIXct("2012-06-28 17:42")  
date2

## [1] "2012-06-28 17:42:00 IST"

class(date2)

## [1] "POSIXct" "POSIXt"

as.numeric(date2)

## [1] 1340885520

Main data types that we are going to use include Numeric and String

## IntrotoR

getwd()

## [1] "C:/Users/ash95/Desktop/Term 2/DSA/Assignment/R Code"

a=2  
a

## [1] 2

## Functions in R

# Functions in R  
divider = function(x,y) {  
 result = x/y  
 print(result)  
}  
divider(50,25)

## [1] 2

divider (100,25)

## [1] 4

# Multiplication  
multiply = function(a,b){  
 result = a \* b  
 print (result)  
}  
multiply(23,25)

## [1] 575

multiply (19,20)

## [1] 380

# Variables Names are CASE SENSITIVE  
A=10  
a=24  
  
# CONCATENATION AND ARRAYS  
f <- c(1,2,3,4,5)  
f = c(1,2,3,4,5)  
f

## [1] 1 2 3 4 5

f+4

## [1] 5 6 7 8 9

d = f / 4  
d

## [1] 0.25 0.50 0.75 1.00 1.25

f+d

## [1] 1.25 2.50 3.75 5.00 6.25

f = c(1,2,3,4,5)  
  
# Listing and Deleting Objects (Variables)  
ls()

## [1] "a" "A" "d" "date1" "date2" "divider"   
## [7] "f" "i" "K" "multiply" "s" "v"   
## [13] "w" "x" "z"

rm (a)  
rm (list = ls())

Functions are useful to create as we can again use the function by calling in different arguments into it.

## Vectors

# Vector - R is called as Vectorized language.  
  
v = c(1,2,3,4,5)  
s = v\*2  
s

## [1] 2 4 6 8 10

# A vector is collec tion of elements, all of same type.  
# A vector canot be of mixed type.  
  
# Vector Operation  
d = v-2  
d

## [1] -1 0 1 2 3

f = v /2  
f

## [1] 0.5 1.0 1.5 2.0 2.5

sqrt(f)

## [1] 0.7071068 1.0000000 1.2247449 1.4142136 1.5811388

## All about vectors

# A vector is collection of elements of same type.  
# (ie) A vector cannot be of mixed type.  
# R is a Vectorized Language. Thant means operations are applied to each element of the vector automatically,  
# .., without the need to loop through the vector.  
# This is a powerful concept and vector plays a crucial and significant role in R.  
  
# Creating Vectors  
# The most common way to create a Vector is using 'c' [combine]  
x = c(1,2,3,4,5,6,7,8,9,10)  
x

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

# Vector Operations  
x\*3 # multiplies each element by 3; No loops necessary!

## [1] 3 6 9 12 15 18 21 24 27 30

x+2

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

x-3

## [1] -2 -1 0 1 2 3 4 5 6 7

x/4

## [1] 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50

x^2

## [1] 1 4 9 16 25 36 49 64 81 100

sqrt(x)

## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427  
## [9] 3.000000 3.162278

# colon (:) operation - Sequencing  
# Creates sequence of Numbers in either direction!  
1:10 #(: - Through)

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

10:1

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

-2:3

## [1] -2 -1 0 1 2 3

5:-7

## [1] 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7

# More on Vector Operations ... Two vectors  
# create two vectors of equal length  
x = 1:10  
y = -5:4  
x + y # Add

## [1] -4 -2 0 2 4 6 8 10 12 14

x-y

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

x\*y

## [1] -5 -8 -9 -8 -5 0 7 16 27 40

x/y

## [1] -0.2 -0.5 -1.0 -2.0 -5.0 Inf 7.0 4.0 3.0 2.5

x^y

## [1] 1.000000e+00 6.250000e-02 3.703704e-02 6.250000e-02 2.000000e-01  
## [6] 1.000000e+00 7.000000e+00 6.400000e+01 7.290000e+02 1.000000e+04

# check the length of each vector  
length(x)

## [1] 10

length(y)

## [1] 10

# Unequal length vectors  
x+c(1,2) # Shorter vector gets recycled!

## [1] 2 4 4 6 6 8 8 10 10 12

x+c (1,2,3)# If Longer vector is not "multiple" of shorter vector, a warning is given!

## Warning in x + c(1, 2, 3): longer object length is not a multiple of shorter  
## object length

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

# Comparison also work on vector!  
x <= 5

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

x<y

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

# Vector Comparison - "any" and "all"  
x = 10:1  
y = -4:5  
any(x<y)

## [1] TRUE

all(x<y)

## [1] FALSE

# The "nchar" function also acts on each element of vector.  
q = c("Hockey","Football","Baseball","Curlin","Rugby","Lacrosse",  
 "Basketball","Tennis","Cricket","Soccer")  
q

## [1] "Hockey" "Football" "Baseball" "Curlin" "Rugby"   
## [6] "Lacrosse" "Basketball" "Tennis" "Cricket" "Soccer"

nchar(q)

## [1] 6 8 8 6 5 8 10 6 7 6

nchar(y)

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

?nchar()

## starting httpd help server ... done

# Subscripting:Accessing "individual elements" in vector is done using square brackets []  
x[1]

## [1] 10

x[1:2]

## [1] 10 9

x[c(1:5,9)]

## [1] 10 9 8 7 6 2

# Give Names to Vector!  
c(One = "a", Two = "y", Last = "r") # Name-Value pair

## One Two Last   
## "a" "y" "r"

# You can Name the vector after creating vector as well!  
w = 1:3  
names(w) = c("a","b","c")  
w

## a b c   
## 1 2 3

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

The most common data structure is the one-dimensional vector

Vector forms the basis of everything in R.

A vector is collection of elements of same type.

(ie) A vector cannot be of mixed type.

R is a Vectorized Language. That means operations are applied to each element of the vector automatically,

.., without the need to loop through the vector.

This is a powerful concept and vector plays a crucial and significant role in R.

## Data Structures

# Sometimes data requires more complex storage than simple vectors.  
# Data Structures - Apart from Vectors, we have Data Frames, Matrix, List and Array.  
  
# Data Frames(DF) - Most useful features of R & also cited reason for R's ease of use.  
# In dataframe, each column is actually a vector, each of which has same length.  
# Each column can hold different type of data.  
# Also within each column, each element must be of same type, like vectors.  
  
# Creating a Dataframe from vectors  
  
x = 10:1  
y = -4:5  
q = c("Hockey","Football","Baseball","Curlin","Rugby","Lacrosse",  
 "Basketball","Tennis","Cricket","Soccer")  
theDF = data.frame(x,y,q) # this would create a 10x3 data.frame with x, y and q as variable names  
theDF

## x y q  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

# Assigning Names  
theDF = data.frame (First=x, Second =y, Sport = q)  
theDF

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

# Checking the dimensions of the DF.  
nrow(theDF)

## [1] 10

ncol(theDF)

## [1] 3

dim(theDF)

## [1] 10 3

names (theDF)

## [1] "First" "Second" "Sport"

names(theDF)[3]

## [1] "Sport"

rownames(theDF)

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

# Head and Tail  
head(theDF)

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse

head(theDF, n=7)

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball

tail(theDF)

## First Second Sport  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

class(theDF)

## [1] "data.frame"

# Accessing Individual Column using $  
theDF$Sport # gives the third column named Sport

## [1] "Hockey" "Football" "Baseball" "Curlin" "Rugby"   
## [6] "Lacrosse" "Basketball" "Tennis" "Cricket" "Soccer"

# Accessing Specific row and column  
theDF[3,2] # 3rd row and 2nd Column

## [1] -2

theDF[3,2:3] # 3rd Row and column 2 thru 3

## Second Sport  
## 3 -2 Baseball

theDF[c(3,5), 2]# Row 3&5 from Column 2;

## [1] -2 0

# since only one column was selected, it was returned as vector and hence no column names in output.  
  
# Rows 3&5 and Columns 2 through 3  
theDF[c(3,5), 2:3]

## Second Sport  
## 3 -2 Baseball  
## 5 0 Rugby

theDF[ ,3] # Access all Rows for column 3

## [1] "Hockey" "Football" "Baseball" "Curlin" "Rugby"   
## [6] "Lacrosse" "Basketball" "Tennis" "Cricket" "Soccer"

theDF[ , 2:3]

## Second Sport  
## 1 -4 Hockey  
## 2 -3 Football  
## 3 -2 Baseball  
## 4 -1 Curlin  
## 5 0 Rugby  
## 6 1 Lacrosse  
## 7 2 Basketball  
## 8 3 Tennis  
## 9 4 Cricket  
## 10 5 Soccer

theDF[2,]# Access all columns for Row 2

## First Second Sport  
## 2 9 -3 Football

theDF[2:4,]

## First Second Sport  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin

theDF[ , c("First", "Sport")]# access using Column Names

## First Sport  
## 1 10 Hockey  
## 2 9 Football  
## 3 8 Baseball  
## 4 7 Curlin  
## 5 6 Rugby  
## 6 5 Lacrosse  
## 7 4 Basketball  
## 8 3 Tennis  
## 9 2 Cricket  
## 10 1 Soccer

## Factors

# Factor Vectors - Ordinal data [Ordered Categorical]  
# Factors are important concept in R, esp. when building models  
  
q2 = c(q,"Hockey","Lacrosse","Hockey","Water Polo","Hockey","Lacrosse")  
q2

## [1] "Hockey" "Football" "Baseball" "Curlin" "Rugby"   
## [6] "Lacrosse" "Basketball" "Tennis" "Cricket" "Soccer"   
## [11] "Hockey" "Lacrosse" "Hockey" "Water Polo" "Hockey"   
## [16] "Lacrosse"

class(q2)

## [1] "character"

as.numeric(q2)

## Warning: NAs introduced by coercion

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

class(q2)

## [1] "character"

# Converting "q2" to factor!  
q2\_F = as.factor(q2)  
q2\_F # notice the "Levels" info in the output!

## [1] Hockey Football Baseball Curlin Rugby Lacrosse   
## [7] Basketball Tennis Cricket Soccer Hockey Lacrosse   
## [13] Hockey Water Polo Hockey Lacrosse   
## 11 Levels: Baseball Basketball Cricket Curlin Football Hockey ... Water Polo

# 11 Levels - 10 Distinct Names from "q" and one (Water polo) from "q2"   
# The "levels" of a factor are the unique values of that factor variable.  
# Technically R is giving "unique integer" to each distinct names, See below  
as.numeric(q2\_F)# IN the O/P --> Notice "6" = "Hockey"

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

# Ordered Levels and Un-ordered Levels  
# Factors can drastically reduce the size of the variable...  
# ... because they are storing only unique values!  
factor(x=c("High School","College","Masters","Doctrate"),  
 levels = c("High School","College","Masters","Doctrate"),  
 ordered = TRUE)

## [1] High School College Masters Doctrate   
## Levels: High School < College < Masters < Doctrate

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

## MissingData

# Missing data plays a crucial role in computing and Statistics  
# R has two types of missing data - NA and NULL  
# while they are similar, but they behave differently and hence needs attention!  
  
# NA - Missing data - Missing Value  
z = c(1,2,NA,8,3,NA,3)  
z = c(1,2,NA,8,3,NA,3)  
z

## [1] 1 2 NA 8 3 NA 3

# "is.na" tests each element of a vector for missingness  
is.na(z)

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

#Another example  
z\_char = c("Hockey", NA ,"Cricket")  
z\_char

## [1] "Hockey" NA "Cricket"

is.na(z\_char)

## [1] FALSE TRUE FALSE

# NULL - Absence of anything. It is not exactly missingness, but nothingness  
# Eg: Having Brain but thinking Nothing! - Makes Sense!!!  
# Functions can sometimes return NULL and their arguments can be NULL.  
# Important difference is, NULL is atomical and cannot exist within a vector...   
# ...If used inside a vector, it simply disappears! Let's see...  
z= c(1,NULL,3)  
z

## [1] 1 3

x = c(1,NA,3)  
x

## [1] 1 NA 3

# Notice, here the "NULL" didnot get stored in "z", infact "z" has only length of 2!  
length(z)

## [1] 2

length(x)

## [1] 3

# Assigning NULL and checking!  
d = NULL  
is.null(d)

## [1] TRUE

## DATA FRAME

# Data Frames(DF) - Most useful features of R & also cited reason for R’s ease of use.

# In dataframe, each column is actually a vector, each of which has same length.

# Each column can hold different type of data.

# Also within each column, each element must be of same type, like vectors.

# Refer the file : “4 DSA Data Structures\_Data.Frame - 4 Dec” (R File)

## MATRICES

# A matrix (plural matrices) is a rectangular array or table of numbers, symbols, or expressions…

#…, arranged in rows and columns.(i.e.) 2-Dimensional Array # Similar to data.frame(RxC) and also similar to Vector # Matrix - Element by element operations are possible. # Refer the file : “4 DSA Data Structures\_Matrices - 4 Dec” (R File)

## DataStructuresinR

# Sometimes data requires more complex storage than simple vectors.  
# Data Structures - Apart from Vectors, we have Data Frames, Matrix, List and Array.  
  
# Data Frames(DF) - Most useful features of R & also cited reason for R's ease of use.  
# In dataframe, each column is actually a vector, each of which has same length.  
# Each column can hold different type of data.  
# Also within each column, each element must be of same type, like vectors.  
  
# Creating a Dataframe from vectors  
  
x = 10:1  
y = -4:5  
q = c("Hockey","Football","Baseball","Curlin","Rugby","Lacrosse",  
 "Basketball","Tennis","Cricket","Soccer")  
theDF = data.frame(x,y,q) # this would create a 10x3 data.frame with x, y and q as variable names  
theDF

## x y q  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

str(theDF)# Very important - Str - Structure

## 'data.frame': 10 obs. of 3 variables:  
## $ x: int 10 9 8 7 6 5 4 3 2 1  
## $ y: int -4 -3 -2 -1 0 1 2 3 4 5  
## $ q: chr "Hockey" "Football" "Baseball" "Curlin" ...

q = as.factor(q)  
  
# Assigning Names  
theDF = data.frame (First=x, Second =y, Sport = q)  
theDF

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

# Checking the dimensions of the DF.  
nrow(theDF)

## [1] 10

ncol(theDF)

## [1] 3

dim(theDF)

## [1] 10 3

names (theDF)

## [1] "First" "Second" "Sport"

names(theDF)[3]

## [1] "Sport"

rownames(theDF)

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

# Head and Tail  
head(theDF)# First 6 rows with all columns

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse

head(theDF, n=10)

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

tail(theDF)# last six rows with all columns

## First Second Sport  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

class(theDF)

## [1] "data.frame"

# Accessing Individual Column using $  
theDF$Sport # gives the third column named Sport

## [1] Hockey Football Baseball Curlin Rugby Lacrosse   
## [7] Basketball Tennis Cricket Soccer   
## 10 Levels: Baseball Basketball Cricket Curlin Football Hockey ... Tennis

# Accessing Specific row and column  
theDF[3,2] # 3rd row and 2nd Column

## [1] -2

theDF[3,2:3] # 3rd Row and column 2 thru 3

## Second Sport  
## 3 -2 Baseball

theDF[c(3,5), 2]# Row 3&5 from Column 2;

## [1] -2 0

# since only one column was selected, it was returned as vector and hence no column names in output.  
  
# Rows 3&5 and Columns 2 through 3  
theDF[c(3,5), 2:3]

## Second Sport  
## 3 -2 Baseball  
## 5 0 Rugby

theDF[ ,3] # Access all Rows for column 3

## [1] Hockey Football Baseball Curlin Rugby Lacrosse   
## [7] Basketball Tennis Cricket Soccer   
## 10 Levels: Baseball Basketball Cricket Curlin Football Hockey ... Tennis

theDF[ , 2:3]

## Second Sport  
## 1 -4 Hockey  
## 2 -3 Football  
## 3 -2 Baseball  
## 4 -1 Curlin  
## 5 0 Rugby  
## 6 1 Lacrosse  
## 7 2 Basketball  
## 8 3 Tennis  
## 9 4 Cricket  
## 10 5 Soccer

theDF[2,]# Access all columns for Row 2

## First Second Sport  
## 2 9 -3 Football

theDF[2:4,]

## First Second Sport  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin

theDF[ , c("First", "Sport")]# access using Column Names

## First Sport  
## 1 10 Hockey  
## 2 9 Football  
## 3 8 Baseball  
## 4 7 Curlin  
## 5 6 Rugby  
## 6 5 Lacrosse  
## 7 4 Basketball  
## 8 3 Tennis  
## 9 2 Cricket  
## 10 1 Soccer

theDF[ ,"Sport"]# Access specific Column

## [1] Hockey Football Baseball Curlin Rugby Lacrosse   
## [7] Basketball Tennis Cricket Soccer   
## 10 Levels: Baseball Basketball Cricket Curlin Football Hockey ... Tennis

class(theDF[ ,"Sport"])

## [1] "factor"

theDF["Sport"]# This returns the one column data.frame

## Sport  
## 1 Hockey  
## 2 Football  
## 3 Baseball  
## 4 Curlin  
## 5 Rugby  
## 6 Lacrosse  
## 7 Basketball  
## 8 Tennis  
## 9 Cricket  
## 10 Soccer

class(theDF["Sport"]) # Data.Frame

## [1] "data.frame"

theDF[["Sport"]]#To access Specific column using Double Square Brackets

## [1] Hockey Football Baseball Curlin Rugby Lacrosse   
## [7] Basketball Tennis Cricket Soccer   
## 10 Levels: Baseball Basketball Cricket Curlin Football Hockey ... Tennis

class(theDF[["Sport"]]) # Factor

## [1] "factor"

theDF[ ,"Sport", drop = FALSE]# Use "Drop=FALSE" to get data.fame with single sqaure bracket.

## Sport  
## 1 Hockey  
## 2 Football  
## 3 Baseball  
## 4 Curlin  
## 5 Rugby  
## 6 Lacrosse  
## 7 Basketball  
## 8 Tennis  
## 9 Cricket  
## 10 Soccer

class(theDF[ ,"Sport", drop = FALSE]) # data.frame

## [1] "data.frame"

theDF[ ,3, drop = FALSE]

## Sport  
## 1 Hockey  
## 2 Football  
## 3 Baseball  
## 4 Curlin  
## 5 Rugby  
## 6 Lacrosse  
## 7 Basketball  
## 8 Tennis  
## 9 Cricket  
## 10 Soccer

class(theDF[ ,3, drop = FALSE]) # data.frame

## [1] "data.frame"

# To see how factor is stored in data.frame  
newFactor = factor(c("Pennsylvania","New York","New Jersey","New York","Tennessee","Massachusetts","Pennsylvania","New York"))  
newFactor

## [1] Pennsylvania New York New Jersey New York Tennessee   
## [6] Massachusetts Pennsylvania New York   
## Levels: Massachusetts New Jersey New York Pennsylvania Tennessee

# model.matrix(~newFactor -1)  
# ? model.matrix()

## DataStructure-Matrices

# A matrix (plural matrices) is a rectangular array or table of numbers, symbols, or expressions...  
#..., arranged in rows and columns.(i.e.) 2-Dimensional Array  
  
# Similar to data.frame(RxC) and also similar to Vector  
# Matrix - Element by element operations are possible  
  
A = matrix(1:10, nrow=5)# Create a 5x2 matrix  
B = matrix(21:30, nrow=5)#Create another 5x2 matrix   
C = matrix (21:40, nrow=2)#Create another 2x10 matrix  
  
A

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

B

## [,1] [,2]  
## [1,] 21 26  
## [2,] 22 27  
## [3,] 23 28  
## [4,] 24 29  
## [5,] 25 30

C

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 21 23 25 27 29 31 33 35 37 39  
## [2,] 22 24 26 28 30 32 34 36 38 40

nrow(A)

## [1] 5

ncol(A)

## [1] 2

dim(A)

## [1] 5 2

# Add Them  
A+B

## [,1] [,2]  
## [1,] 22 32  
## [2,] 24 34  
## [3,] 26 36  
## [4,] 28 38  
## [5,] 30 40

# Multiply Them (Vector Multiplication!)  
A

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

B

## [,1] [,2]  
## [1,] 21 26  
## [2,] 22 27  
## [3,] 23 28  
## [4,] 24 29  
## [5,] 25 30

A\*B # A = 5x2 and B = 5x2

## [,1] [,2]  
## [1,] 21 156  
## [2,] 44 189  
## [3,] 69 224  
## [4,] 96 261  
## [5,] 125 300

#See if the elements are equal  
A == B

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

# Matrix Multiplication(MM. A is 5x2. B is 5x2. B-transpose is 2x5  
A %\*% t(B)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 177 184 191 198 205  
## [2,] 224 233 242 251 260  
## [3,] 271 282 293 304 315  
## [4,] 318 331 344 357 370  
## [5,] 365 380 395 410 425

# Naming the Columns and Rows   
colnames(A)

## NULL

rownames(A)

## NULL

colnames(A)= c("Left","Right")  
rownames(A)= c("1st","2nd","3rd","4th","5th")  
colnames(B)

## NULL

rownames(B)

## NULL

colnames(B)= c("First","Second")  
rownames(B)= c("One","Two","Three","Four","Five")  
colnames(C)

## NULL

rownames(C)

## NULL

colnames(C) = LETTERS [1:10]  
rownames(C) = c("Top", "Bottom")  
  
# Matrix Multiplication. A is 5x2 and C is 2x10  
dim(A)

## [1] 5 2

dim(C)

## [1] 2 10

t(A)

## 1st 2nd 3rd 4th 5th  
## Left 1 2 3 4 5  
## Right 6 7 8 9 10

A %\*% C

## A B C D E F G H I J  
## 1st 153 167 181 195 209 223 237 251 265 279  
## 2nd 196 214 232 250 268 286 304 322 340 358  
## 3rd 239 261 283 305 327 349 371 393 415 437  
## 4th 282 308 334 360 386 412 438 464 490 516  
## 5th 325 355 385 415 445 475 505 535 565 595

## ARRAYS

# Arrays - An array is essentially a multidimensional vector.

# It must all be of the same type and

# …individual elements are accessed using Square Brackets.

# First element is Row(R) Index, Second Element is Column(C) Index and

# the remaining elements are for Outer Dimensions (OD).

## Arrays

# Arrays - An array is essentially a multidimensional vector.  
# It must all be of the same type and   
# ...individual elements are accessed using Square Brackets.  
# First element is Row(R) Index, Second Element is Column(C) Index and   
# the remaining elements are for Outer Dimensions (OD).  
  
theArray = array(1:12, dim=c(2,3,2))# Total Elements = R x C x OD  
theArray

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

theArray [1, ,]# Accessing all elements from Row 1, all columns, all outer dimensions & build C x OD (R x C)

## [,1] [,2]  
## [1,] 1 7  
## [2,] 3 9  
## [3,] 5 11

theArray[1, ,1]# Accessing all elements from Row 1, all columns, first outer dimension

## [1] 1 3 5

theArray[, ,1]# Accessing all rows, all columns, first outer dimension

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

# Array with Four Outer Dimensions (OD)  
theArray\_4D = array(1:32, dim=c(2,4,4))  
theArray\_4D

## , , 1  
##   
## [,1] [,2] [,3] [,4]  
## [1,] 1 3 5 7  
## [2,] 2 4 6 8  
##   
## , , 2  
##   
## [,1] [,2] [,3] [,4]  
## [1,] 9 11 13 15  
## [2,] 10 12 14 16  
##   
## , , 3  
##   
## [,1] [,2] [,3] [,4]  
## [1,] 17 19 21 23  
## [2,] 18 20 22 24  
##   
## , , 4  
##   
## [,1] [,2] [,3] [,4]  
## [1,] 25 27 29 31  
## [2,] 26 28 30 32

theArray\_4D [1, ,]

## [,1] [,2] [,3] [,4]  
## [1,] 1 9 17 25  
## [2,] 3 11 19 27  
## [3,] 5 13 21 29  
## [4,] 7 15 23 31

theArray\_4D[1, ,1]

## [1] 1 3 5 7

theArray[, ,1]

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

## LIST

# Lists - Stores any number of items of any type.

# List can contain all numerics or characters or…

#…a mix of the two or data.frames or recursively other lists.

## List

# Lists - Stores any number of items of any type.  
# List can contain all numerics or characters or...  
#...a mix of the two or data.frames or recursively other lists.  
  
# Lists are created with the "list" function.  
# Each argument in "list" becomes an element of the list.  
  
list(1,2,3)# creates a three element list

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

list(c(1,2,3))# creates a single element(vector with three elements)

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

list3 = list(c(1,2,3), 3:7)# create two element list  
# first is three elements vector, next is five element vector.  
list3

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

# The same can be written as  
(list3 = list(c(1,2,3), 3:7))

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

# Two Element list  
# First element is data.frame and next is 10 element vector  
list(theDF, 1:10)# theDF is already created in previous exercise!

## [[1]]  
## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer  
##   
## [[2]]  
## [1] 1 2 3 4 5 6 7 8 9 10

# Three element list  
list5 = list(theDF, 1:10, list3)  
list5

## [[1]]  
## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer  
##   
## [[2]]  
## [1] 1 2 3 4 5 6 7 8 9 10  
##   
## [[3]]  
## [[3]][[1]]  
## [1] 1 2 3  
##   
## [[3]][[2]]  
## [1] 3 4 5 6 7

#Naming List (similar to column name in data.frame)   
names(list5)= c("data.frame", "vector","list")  
names(list5)

## [1] "data.frame" "vector" "list"

list5

## $data.frame  
## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer  
##   
## $vector  
## [1] 1 2 3 4 5 6 7 8 9 10  
##   
## $list  
## $list[[1]]  
## [1] 1 2 3  
##   
## $list[[2]]  
## [1] 3 4 5 6 7

#Naming using "Name-Value" pair  
list6 = list(TheDataFrame = theDF, TheVector = 1:10, TheList = list3)  
names(list6)

## [1] "TheDataFrame" "TheVector" "TheList"

list6

## $TheDataFrame  
## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer  
##   
## $TheVector  
## [1] 1 2 3 4 5 6 7 8 9 10  
##   
## $TheList  
## $TheList[[1]]  
## [1] 1 2 3  
##   
## $TheList[[2]]  
## [1] 3 4 5 6 7

# Creating an empty list  
(emptylist = vector(mode="list", length =4))

## [[1]]  
## NULL  
##   
## [[2]]  
## NULL  
##   
## [[3]]  
## NULL  
##   
## [[4]]  
## NULL

# Accessing individual element of a list - Double Square Brackets  
# specify either element number or name  
list5[[1]]

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

list5[["data.frame"]]

## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer

list5[[1]]$Sport

## [1] Hockey Football Baseball Curlin Rugby Lacrosse   
## [7] Basketball Tennis Cricket Soccer   
## 10 Levels: Baseball Basketball Cricket Curlin Football Hockey ... Tennis

list5[[1]][,"Second"]

## [1] -4 -3 -2 -1 0 1 2 3 4 5

list5[[1]][,"Second", drop = FALSE]

## Second  
## 1 -4  
## 2 -3  
## 3 -2  
## 4 -1  
## 5 0  
## 6 1  
## 7 2  
## 8 3  
## 9 4  
## 10 5

# LENGTH OF LIST  
length(list5)

## [1] 3

names(list5)

## [1] "data.frame" "vector" "list"

list5

## $data.frame  
## First Second Sport  
## 1 10 -4 Hockey  
## 2 9 -3 Football  
## 3 8 -2 Baseball  
## 4 7 -1 Curlin  
## 5 6 0 Rugby  
## 6 5 1 Lacrosse  
## 7 4 2 Basketball  
## 8 3 3 Tennis  
## 9 2 4 Cricket  
## 10 1 5 Soccer  
##   
## $vector  
## [1] 1 2 3 4 5 6 7 8 9 10  
##   
## $list  
## $list[[1]]  
## [1] 1 2 3  
##   
## $list[[2]]  
## [1] 3 4 5 6 7

## Reading data in R

# Its time that we load data in R.  
# Most common way to get data is reading comma separated values(CSV)  
  
# Reading CSVs  
theUrl = "http://www.jaredlander.com/data/Tomato%20First.csv"  
# visit https://www.jaredlander.com/data/ for other Datasets  
tomato = read.table(file=theUrl, header=TRUE, sep =",")  
head(tomato)

## Round Tomato Price Source Sweet Acid Color Texture Overall  
## 1 1 Simpson SM 3.99 Whole Foods 2.8 2.8 3.7 3.4 3.4  
## 2 1 Tuttorosso (blue) 2.99 Pioneer 3.3 2.8 3.4 3.0 2.9  
## 3 1 Tuttorosso (green) 0.99 Pioneer 2.8 2.6 3.3 2.8 2.9  
## 4 1 La Fede SM DOP 3.99 Shop Rite 2.6 2.8 3.0 2.3 2.8  
## 5 2 Cento SM DOP 5.49 D Agostino 3.3 3.1 2.9 2.8 3.1  
## 6 2 Cento Organic 4.99 D Agostino 3.2 2.9 2.9 3.1 2.9  
## Avg.of.Totals Total.of.Avg  
## 1 16.1 16.1  
## 2 15.3 15.3  
## 3 14.3 14.3  
## 4 13.4 13.4  
## 5 14.4 15.2  
## 6 15.5 15.1

#It might be tempting to use read.csv but that is more trouble than it is worth,  
#...and all it does is call read.table with some arguments preset.  
  
# Sometimes CSVs(or tab delimited files) are poorly built,   
# where the cell separator has been used inside a cell.  
# In this case read.csv2(or read.delim2)should be used instead of read.table.  
  
# Reading Excel Data - Not worth the Effort.  
# Unfortunately, it is difficult to read Excel data into R - Requires additional packages to be installed.  
# Convert into CSV and read.  
  
# Reading Text Files  
myPeople = read.table("C:/Users/ash95/Desktop/Term 2/DSA/Assignment/New folder/Azutoz.txt",  
header=T, sep=" ",  
na.strings="`",  
stringsAsFactors=F)  
myPeople

## name roll sex  
## 1 ashutosh 20080 m  
## 2 aman 1213 m  
## 3 Rahul 4567567 M  
## 4 Rahul 4567567 M  
## 5 Rahul 4567567 M  
## 6 Rahul 4567567 M  
## 7 Rahul 4567567 M  
## 8 Rahul 4567567 M  
## 9 Rahul 4567567 M  
## 10 Rahul 4567567 M

#Reading the files  
  
  
  
# Add another person  
addname = data.frame(name="Rahul",  
roll="4567567",  
sex="M")  
myPeople = rbind(myPeople, addname)  
myPeople

## name roll sex  
## 1 ashutosh 20080 m  
## 2 aman 1213 m  
## 3 Rahul 4567567 M  
## 4 Rahul 4567567 M  
## 5 Rahul 4567567 M  
## 6 Rahul 4567567 M  
## 7 Rahul 4567567 M  
## 8 Rahul 4567567 M  
## 9 Rahul 4567567 M  
## 10 Rahul 4567567 M  
## 11 Rahul 4567567 M

# Update a record  
myPeople[2,2] = "1213"  
myPeople

## name roll sex  
## 1 ashutosh 20080 m  
## 2 aman 1213 m  
## 3 Rahul 4567567 M  
## 4 Rahul 4567567 M  
## 5 Rahul 4567567 M  
## 6 Rahul 4567567 M  
## 7 Rahul 4567567 M  
## 8 Rahul 4567567 M  
## 9 Rahul 4567567 M  
## 10 Rahul 4567567 M  
## 11 Rahul 4567567 M

# Update the file by supplying the data.frame,  
# the file to write, seperator, na, whether to  
# quote strings, whether to include row numbers  
write.table(x=myPeople, "C:/Users/ash95/Desktop/Term 2/DSA/Assignment/New folder/Azutoz.txt",  
sep=" ", na="`",  
quote=F, row.names=F)  
  
  
  
# Get 1st 3 records  
head(myPeople, 3)

## name roll sex  
## 1 ashutosh 20080 m  
## 2 aman 1213 m  
## 3 Rahul 4567567 M

# Get remaining records  
tail(myPeople, 3)

## name roll sex  
## 9 Rahul 4567567 M  
## 10 Rahul 4567567 M  
## 11 Rahul 4567567 M

Any kind of file (csv or txt) can be read into R and data manipulation and presentation can be done with the help of it.

## builtindatasets

# Built-in datasets in R  
data()# List of built-in Datasets in R  
  
# Loading  
data(mtcars)  
# Print the first 6 rows  
head(mtcars, 6)

## mpg cyl disp hp drat wt qsec vs am gear carb  
## Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4  
## Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
## Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1  
## Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
## Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
## Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

## Summary Statistics

# Basic Statistics - Mean, Variances,Correlations and T-tests  
  
# Generate a random sample of 100 numbers between 1 and 100  
x = sample(x=1:100, size = 100, replace = TRUE)  
x # the output of "x" is a vector of data

## [1] 29 91 74 55 45 51 87 45 40 45 49 54 5 55 28 35 14 70 38 83 3 65 86 83 23  
## [26] 24 15 55 66 79 81 99 3 39 89 74 13 39 28 23 92 84 77 86 76 41 17 69 7 31  
## [51] 70 80 89 84 52 56 70 94 65 54 65 57 83 26 83 22 58 6 72 30 46 92 75 48 1  
## [76] 98 97 11 92 13 59 41 68 47 37 17 51 44 77 92 69 99 47 4 85 75 90 43 42 60

# Simple Arithmetic Mean  
mean(x)

## [1] 54.96

# Calculate Mean when Missing Data is found  
y = x # copy x to y  
y[sample(x=1:100, size = 20, replace = FALSE)] = NA  
y

## [1] 29 91 NA NA 45 51 NA 45 40 45 49 54 5 NA 28 35 14 70 NA 83 3 65 86 83 23  
## [26] 24 15 55 NA 79 81 99 NA 39 89 74 13 39 28 NA 92 NA 77 86 76 NA 17 69 7 31  
## [51] 70 80 NA 84 52 56 70 94 65 54 65 57 83 26 NA 22 58 6 NA 30 46 92 75 48 1  
## [76] 98 97 11 92 NA 59 41 68 47 NA NA 51 44 77 92 69 99 47 NA NA NA 90 43 NA 60

mean(y)# Will give NA!

## [1] NA

# Remove missing value(s)and calculate mean  
mean(y, na.rm=TRUE) # Now, it will give the mean value

## [1] 55.6625

# Weighted Mean  
Grades = c(95,72,87,66)  
Weights = c(1/2, 1/4, 1/8, 1/8)  
mean(Grades)# Simple Arithmetic mean

## [1] 80

weighted.mean(x = Grades, w = Weights)# Weighted Mean

## [1] 84.625

#Variance  
var(x)

## [1] 768.059

#Calculating Variance using formula!  
sum((x-mean(x))^2)/ (length(x)-1)

## [1] 768.059

# Standard Deviation  
sqrt(var(x))

## [1] 27.71388

sd(x)

## [1] 27.71388

sd(y)

## [1] NA

sd(y, na.rm=TRUE)

## [1] 27.46964

# Other Commonly Used Functions  
min(x)

## [1] 1

max(x)

## [1] 99

median(x)

## [1] 55

min(y)

## [1] NA

min(y, na.rm=TRUE)

## [1] 1

# Summary Statistics  
summary(x)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.00 36.50 55.00 54.96 79.25 99.00

summary(y)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 1.00 38.00 55.50 55.66 79.25 99.00 20

# Quantiles  
quantile(x, probs = c(0.25, 0.75)) # Calculate 25th and 75th Quantile

## 25% 75%   
## 36.50 79.25

quantile(x, probs = c(0.1,0.25,0.5, 0.75,0.99))

## 10% 25% 50% 75% 99%   
## 13.90 36.50 55.00 79.25 99.00

quantile(y, probs = c(0.25, 0.75), na.rm=TRUE)

## 25% 75%   
## 38.00 79.25

All the descriptive statistics parameters like mean, median and mode can be found out and not only this quantiles and percentiles could be found out using simple formulae

## Correlation

# Correlation  
  
# Prepare the Data  
mydata <- mtcars[, c(1,3,4,5,6,7)]  
head(mydata)

## mpg disp hp drat wt qsec  
## Mazda RX4 21.0 160 110 3.90 2.620 16.46  
## Mazda RX4 Wag 21.0 160 110 3.90 2.875 17.02  
## Datsun 710 22.8 108 93 3.85 2.320 18.61  
## Hornet 4 Drive 21.4 258 110 3.08 3.215 19.44  
## Hornet Sportabout 18.7 360 175 3.15 3.440 17.02  
## Valiant 18.1 225 105 2.76 3.460 20.22

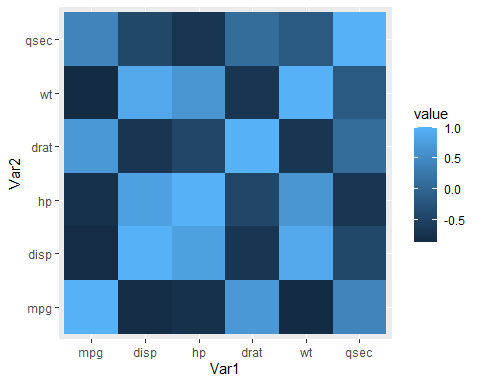
# Compute the correlation matrix - cor()  
cormat <- round(cor(mydata),2)  
head(cormat)

## mpg disp hp drat wt qsec  
## mpg 1.00 -0.85 -0.78 0.68 -0.87 0.42  
## disp -0.85 1.00 0.79 -0.71 0.89 -0.43  
## hp -0.78 0.79 1.00 -0.45 0.66 -0.71  
## drat 0.68 -0.71 -0.45 1.00 -0.71 0.09  
## wt -0.87 0.89 0.66 -0.71 1.00 -0.17  
## qsec 0.42 -0.43 -0.71 0.09 -0.17 1.00

# Create the correlation heatmap with ggplot2  
# The package reshape is required to melt the correlation matrix.  
library(reshape2)  
melted\_cormat <- melt(cormat)  
head(melted\_cormat)

## Var1 Var2 value  
## 1 mpg mpg 1.00  
## 2 disp mpg -0.85  
## 3 hp mpg -0.78  
## 4 drat mpg 0.68  
## 5 wt mpg -0.87  
## 6 qsec mpg 0.42

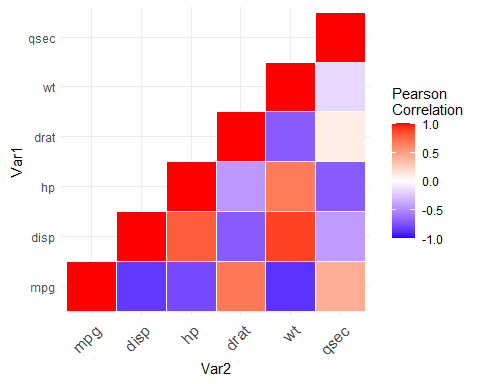
#The function geom\_tile()[ggplot2 package] is used to visualize the correlation matrix :  
library(ggplot2)  
ggplot(data = melted\_cormat, aes(x=Var1, y=Var2, fill=value)) +   
 geom\_tile()



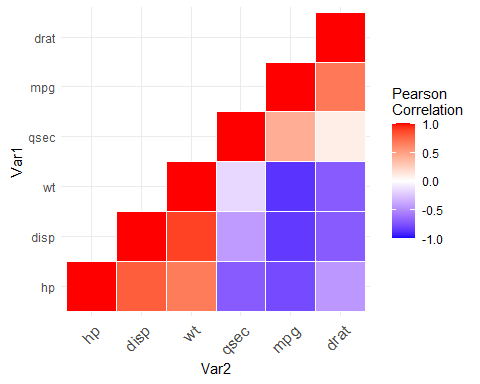
#Doesnot Look Great.. Let's Enhance the viz!  
  
#Get the lower and upper triangles of the correlation matrix  
## a correlation matrix has redundant information. We'll use the functions below to set half of it to NA.  
  
# Get lower triangle of the correlation matrix  
get\_lower\_tri<-function(cormat){  
 cormat[upper.tri(cormat)] <- NA  
 return(cormat)  
}  
# Get upper triangle of the correlation matrix  
get\_upper\_tri <- function(cormat){  
 cormat[lower.tri(cormat)]<- NA  
 return(cormat)  
}  
  
upper\_tri <- get\_upper\_tri(cormat)  
upper\_tri

## mpg disp hp drat wt qsec  
## mpg 1 -0.85 -0.78 0.68 -0.87 0.42  
## disp NA 1.00 0.79 -0.71 0.89 -0.43  
## hp NA NA 1.00 -0.45 0.66 -0.71  
## drat NA NA NA 1.00 -0.71 0.09  
## wt NA NA NA NA 1.00 -0.17  
## qsec NA NA NA NA NA 1.00

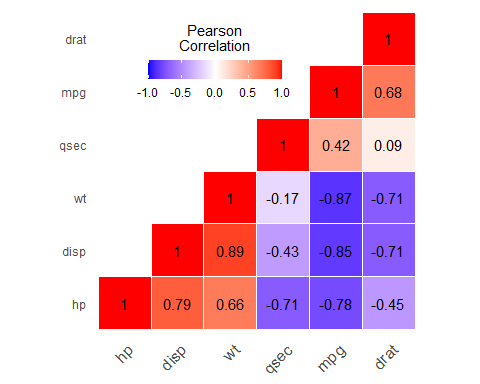
# Finished correlation matrix heatmap  
## Melt the correlation data and drop the rows with NA values  
# Melt the correlation matrix  
library(reshape2)  
melted\_cormat <- melt(upper\_tri, na.rm = TRUE)  
# Heatmap  
library(ggplot2)  
ggplot(data = melted\_cormat, aes(Var2, Var1, fill = value))+  
 geom\_tile(color = "white")+  
 scale\_fill\_gradient2(low = "blue", high = "red", mid = "white",   
 midpoint = 0, limit = c(-1,1), space = "Lab",   
 name="Pearson\nCorrelation") +  
 theme\_minimal()+   
 theme(axis.text.x = element\_text(angle = 45, vjust = 1,   
 size = 12, hjust = 1))+  
 coord\_fixed()



# negative correlations are in blue color and positive correlations in red.   
# The function scale\_fill\_gradient2 is used with the argument limit = c(-1,1) as correlation coefficients range from -1 to 1.  
# coord\_fixed() : this function ensures that one unit on the x-axis is the same length as one unit on the y-axis.  
  
# Reorder the correlation matrix  
  
# This section describes how to reorder the correlation matrix according to the correlation coefficient.   
# This is useful to identify the hidden pattern in the matrix.   
# hclust for hierarchical clustering order is used in the example below.  
  
reorder\_cormat <- function(cormat){  
 # Use correlation between variables as distance  
 dd <- as.dist((1-cormat)/2)  
 hc <- hclust(dd)  
 cormat <-cormat[hc$order, hc$order]  
}  
  
# Reorder the correlation matrix  
cormat <- reorder\_cormat(cormat)  
upper\_tri <- get\_upper\_tri(cormat)  
# Melt the correlation matrix  
melted\_cormat <- melt(upper\_tri, na.rm = TRUE)  
# Create a ggheatmap  
ggheatmap <- ggplot(melted\_cormat, aes(Var2, Var1, fill = value))+  
 geom\_tile(color = "white")+  
 scale\_fill\_gradient2(low = "blue", high = "red", mid = "white",   
 midpoint = 0, limit = c(-1,1), space = "Lab",   
 name="Pearson\nCorrelation") +  
 theme\_minimal()+ # minimal theme  
 theme(axis.text.x = element\_text(angle = 45, vjust = 1,   
 size = 12, hjust = 1))+  
 coord\_fixed()  
# Print the heatmap  
print(ggheatmap)



#Add correlation coefficients on the heatmap  
  
## Use geom\_text() to add the correlation coefficients on the graph  
## Use a blank theme (remove axis labels, panel grids and background, and axis ticks)  
## Use guides() to change the position of the legend title  
  
ggheatmap +   
 geom\_text(aes(Var2, Var1, label = value), color = "black", size = 4) +  
 theme(  
 axis.title.x = element\_blank(),  
 axis.title.y = element\_blank(),  
 panel.grid.major = element\_blank(),  
 panel.border = element\_blank(),  
 panel.background = element\_blank(),  
 axis.ticks = element\_blank(),  
 legend.justification = c(1, 0),  
 legend.position = c(0.6, 0.7),  
 legend.direction = "horizontal")+  
 guides(fill = guide\_colorbar(barwidth = 7, barheight = 1,  
 title.position = "top", title.hjust = 0.5))



The heatmap() function is natively provided in R. It produces high quality matrix and offers statistical tools to normalize input data, run clustering algorithm and visualize the result with dendrograms.

ggplot2 also allows to build heatmaps thanks to geom\_tile()

Three options exist to build an interactive heatmap from R:

plotly: as described above, plotly allows to turn any heatmap made with ggplot2 interactive.

d3heatmap: a package that uses the same syntax as the base R heatmap() function to make interactive version.

heatmaply: the most flexible option, allowing many different kind of customization. See the code of the chart beside here.

## Hypothesis Testing

# T-tests  
# Dataset: Tips dependents on...  
data(tips, package = "reshape2")  
head(tips)

## total\_bill tip sex smoker day time size  
## 1 16.99 1.01 Female No Sun Dinner 2  
## 2 10.34 1.66 Male No Sun Dinner 3  
## 3 21.01 3.50 Male No Sun Dinner 3  
## 4 23.68 3.31 Male No Sun Dinner 2  
## 5 24.59 3.61 Female No Sun Dinner 4  
## 6 25.29 4.71 Male No Sun Dinner 4

str(tips)

## 'data.frame': 244 obs. of 7 variables:  
## $ total\_bill: num 17 10.3 21 23.7 24.6 ...  
## $ tip : num 1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...  
## $ sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2 2 ...  
## $ smoker : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ day : Factor w/ 4 levels "Fri","Sat","Sun",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ time : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1 1 1 ...  
## $ size : int 2 3 3 2 4 4 2 4 2 2 ...

write.csv(tips, "C:/Users/ash95/Desktop/Term 2/DSA/Assignment/R Code/tips.csv", row.names = FALSE)  
  
# Gender  
unique(tips$sex)

## [1] Female Male   
## Levels: Female Male

#Day of the week  
unique(tips$day)

## [1] Sun Sat Thur Fri   
## Levels: Fri Sat Sun Thur

#One Sample t-test - ONE GROUP [Two Tail. Ho:Mean = 2.5]  
t.test(tips$tip, alternative = "two.sided", mu=2.5)

##   
## One Sample t-test  
##   
## data: tips$tip  
## t = 5.6253, df = 243, p-value = 5.08e-08  
## alternative hypothesis: true mean is not equal to 2.5  
## 95 percent confidence interval:  
## 2.823799 3.172758  
## sample estimates:  
## mean of x   
## 2.998279

#One Sample t-test - Upper Tail. Ho:Mean LE 2.5  
t.test(tips$tip, alternative = "greater", mu=2.5)

##   
## One Sample t-test  
##   
## data: tips$tip  
## t = 5.6253, df = 243, p-value = 2.54e-08  
## alternative hypothesis: true mean is greater than 2.5  
## 95 percent confidence interval:  
## 2.852023 Inf  
## sample estimates:  
## mean of x   
## 2.998279

# Two Sample T-test - TWO GROUP  
t.test(tip ~ sex, data = tips, var.equal = TRUE)

##   
## Two Sample t-test  
##   
## data: tip by sex  
## t = -1.3879, df = 242, p-value = 0.1665  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.6197558 0.1074167  
## sample estimates:  
## mean in group Female mean in group Male   
## 2.833448 3.089618

#Paired Two-Sample T-Test   
# Dataset: Heights of Father and Son (Package:UsingR)  
install.packages("UsingR", repo= "https://cran.us.r-project.org")

## Installing package into 'C:/Users/ash95/Documents/R/win-library/4.0'  
## (as 'lib' is unspecified)

## package 'UsingR' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\ash95\AppData\Local\Temp\RtmpGSvHxq\downloaded\_packages

require(UsingR)

## Loading required package: UsingR

## Loading required package: MASS

## Loading required package: HistData

## Loading required package: Hmisc

## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

##   
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':  
##   
## format.pval, units

##   
## Attaching package: 'UsingR'

## The following object is masked from 'package:survival':  
##   
## cancer

head(father.son)

## fheight sheight  
## 1 65.04851 59.77827  
## 2 63.25094 63.21404  
## 3 64.95532 63.34242  
## 4 65.75250 62.79238  
## 5 61.13723 64.28113  
## 6 63.02254 64.24221

write.csv(father.son, "C:/Users/ash95/Desktop/Term 2/DSA/Assignment/R Code/father\_son.csv", row.names = FALSE)  
  
#ANOVA - Comparing Multiple Groups  
# Tip by the Day of the Week  
str(tips)

## 'data.frame': 244 obs. of 7 variables:  
## $ total\_bill: num 17 10.3 21 23.7 24.6 ...  
## $ tip : num 1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...  
## $ sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2 2 ...  
## $ smoker : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ day : Factor w/ 4 levels "Fri","Sat","Sun",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ time : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1 1 1 ...  
## $ size : int 2 3 3 2 4 4 2 4 2 2 ...

tipAnova = aov(tip ~ day, tips)  
summary(tipAnova)

## Df Sum Sq Mean Sq F value Pr(>F)  
## day 3 9.5 3.175 1.672 0.174  
## Residuals 240 455.7 1.899

* **t.test(data.1, data.2)** – The basic method of applying a t-test is to compare two vectors of numeric data.
* **alternative = “two.sided”** – It sets the alternative hypothesis. The default value for this is “two.sided” but a greater or lesser value can also be assigned. You can abbreviate the instruction.
* **conf.level = 0.95** – It sets the confidence level of the interval (default = 0.95).
* **paired = FALSE** – If set to TRUE, a matched pair T-test is carried out.

The one-way analysis of variance (ANOVA), also known as one-factor ANOVA, is an extension of independent two-samples t-test for comparing means in a situation where there are more than two groups

This could also be performed in R using aov and summary