Big Data Tools For Managers

Unit-3 : Introduction to R & R Programming

Variables

- Variables are container for storing data value in memory.
- In R, Variable gets created as soon as it gets assign with some value to it
- R Supports, Left assignment (<-), Right Assignment (->) and Equal to (=) for assigning value to the Variable.
- Most prefer assignment operator is <- (Left assignment)

In [8]:

```
x <- 10
y = 100
200 -> z

a <- b <- c <- "Hello"   #Allows assigning same value to multiple variables
```

Print Function

- print() function used to display the value of variable in R

In [15]:

```
print(x) #Value of x
print(y) #Value of y
print(a)
print(b)
print(c)
```

```
[1] 10
[1] 100
[1] "Hello"
[1] "Hello"
[1] "Hello"
```

Working Directory

- getwd() function to get the project working directory in R
- setwd() function to change or modify the project working directory in R

In [11]:

```
getwd()
```

'/resources/labs/R101'

In [13]:

```
setwd("D:/R/Programs")
```

Comments

- R allows to annotate the code with comments.
- Comments line start with #(Hash) and anything after that will be ignored for execution in R

In [16]:

```
# This is comment lines
10 + 10 #Addition of two numbers
```

20

R Packages

- install.packages() functions to download and install packages from Internet.
- library() functions to import downloaded package in R code/program.

In [20]:

```
install.packages("car") #Dowload Regression packages
install.packages("tm")  #Text Mining Packages
```

In [21]:

```
library("car") # Importing car package in R Code
library("tm")  # Importing Text Mining package in R Code
```

Basic Data Types

- print function helps to display/get the value of variable
- class function helps to get the class for Data Type

In [23]:

```
# Logical

x <- TRUE
print(x)
```

```
[1] TRUE
```

In [25]:

```
class(x)
```

'logical'

In [ ]:

```

```

In [26]:

```
# Numeric
x <- 70.15
print(x)
```

```
[1] 70.15
```

In [27]:

```
class(x)
```

'numeric'

In [ ]:

```

```

In [28]:

```
# Integer
x <- 10L
print(x)
```

```
[1] 10
```

In [29]:

```
class(x)
```

'integer'

In [ ]:

```

```

In [30]:

```
# Complex Number
x <- 6+4i
print(x)
```

```
[1] 6+4i
```

In [31]:

```
class(x)
```

'complex'

In [ ]:

```

```

In [35]:

```
# Character

x <- "Hello World" #with Double qoute
y <- 'yes' #with single quote
```

In [36]:

```
print(x)
print(y)
```

```
[1] "Hello World"
[1] "yes"
```

In [37]:

```
class(x)
class(y)
```

'character'

'character'

In [ ]:

```

```

Vectors

- Vector is the essential building block for handling multiple items in R
- Combine/Concatenation function c() used to create vectors with the help of Basic data types

Vector Creation

In [138…

```
fruits <- c("Apple", "Banana", "Orange")  # Character type Vector
num <- c(1,2,3,4,5,6,100,400,500,11,12) # Numeric Vector
```

In [139…

```
print(fruits)
print(num)
```

```
[1] "Apple"  "Banana" "Orange"
 [1]   1   2   3   4   5   6 100 400 500  11  12
```

In [140…

```
class(fruits)
class(num)
```

'character'

'numeric'

Accessing Element from Vector

- We can access vector element by it's index, and index start from 1 to N.

vector_variable[index]

In [141…
```
fruits[1] # 1st Index
```

'Apple'

In [142…
```
num[3] #3rd index
```

3

In [149…
```
num[4:6] #Continoues index 4 to 6
```

4 · 5 · 6

In [150…
```
num[c(1,9,10)] #Index with discreate index
```

1 · 500 · 11

In [153…
```
num[-1]  # Negative index to exlude the element, display all the elemenet except 1st s
```

2 · 3 · 4 · 5 · 6 · 100 · 400 · 500 · 11 · 12

In [155…
```
num[c(-1,-5,-10)] # Exclude element index 1, 5 and 10
```

2 · 3 · 4 · 6 · 100 · 400 · 500 · 12

In [ ]:

Continuous Range

In [48]:
```
x <- 1 : 50
y <- 10.5 : 15.5
```

In [49]:
```
print(x)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

In [50]:
```
print(y)
```

```
[1] 10.5 11.5 12.5 13.5 14.5 15.5
```

In [ ]:

Sequence

- Common useful functions to create continuous number generation seq(from=, to=, by=)

seq(from=, to=, length.out=)

In [51]:
```
seq(from=1, to=10, by=1)
```

```
1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10
```

In [55]:
```
seq(from=10, to=100, by=10) # Increment by 10
```

```
10 · 20 · 30 · 40 · 50 · 60 · 70 · 80 · 90 · 100
```

In [56]:
```
seq(from=1, to=10, by=2) # Increment by 2
```

```
1 · 3 · 5 · 7 · 9
```

In [57]:
```
seq(from=1, to=10, length.out=20) # Generate 20 number in between 1 to 10
```

```
1 · 1.47368421052632 · 1.94736842105263 · 2.42105263157895 · 2.89473684210526 ·
3.36842105263158 · 3.84210526315789 · 4.31578947368421 · 4.78947368421053 ·
5.26315789473684 · 5.73684210526316 · 6.21052631578947 · 6.68421052631579 ·
7.15789473684211 · 7.63157894736842 · 8.10526315789474 · 8.57894736842105 ·
9.05263157894737 · 9.52631578947368 · 10
```

In [ ]:
```

```

Repeat

- Common useful functions to repeat the certain values in Vector

rep(x= ,times=) # Number of time X vectors gets repeated

rep(x= ,each=) # Number of time each X vectors get repeated

In [62]:
```
rep(x=1, times=4)
```

```
1 · 1 · 1 · 1
```

In [63]:
```
rep(x=c(1,2,3), times=4)
```

```
1 · 2 · 3 · 1 · 2 · 3 · 1 · 2 · 3 · 1 · 2 · 3
```

In [64]:
```
rep(x=c(1,2,3), each=2)
```

```
1 · 1 · 2 · 2 · 3 · 3
```

In [65]:
```
rep(x=c(1,2,3), each=2, times=4)
```

```
1 · 1 · 2 · 2 · 3 · 3 · 1 · 1 · 2 · 2 · 3 · 3 · 1 · 1 · 2 · 2 · 3 · 3 · 1 · 1 · 2 · 2 · 3 · 3
```

In [ ]:
```

```

Length

- length() function helps to find the number of elements/length of vector

In [67]:

```
num <- 1:500
length(num)
```

500

In [68]:

```
x <- c('A','B','C')
length(x)
```

3

In [ ]:

```
```

Sort

- sort() functions to sort the elements of Vectors.

sort(x= , decreasing=FALSE)

- x : is vectors
- decreasing: parameters to sort element in increasing/descrising order

In [71]:

```
data = c(2.5, -1, -10, 3.44, 55)
print(data)
```

```
[1]   2.50  -1.00 -10.00   3.44  55.00
```

In [76]:

```
sort(x=data, decreasing=FALSE) # Sort in increasing order, default if FALSE. means alw
```

-10 · -1 · 2.5 · 3.44 · 55

In [77]:

```
sort(x=data, decreasing=TRUE)
```

55 · 3.44 · 2.5 · -1 · -10

In [ ]:

```
```

In [ ]:

```
```

LETTERS or letters in-built variables used to create alphabets vectors directly.

A vectors of positive integer from letters return the 20 lower case and LETTERS returns the 26 upper case

In [78]:

```
letters
```

```
'a' · 'b' · 'c' · 'd' · 'e' · 'f' · 'g' · 'h' · 'i' · 'j' · 'k' · 'l' · 'm' · 'n' · 'o' · 'p' · 'q' · 'r' · 's' · 't' · 'u' · 'v' · 'w' · 'x' · 'y' ·
'z'
```

In [79]:

```
LETTERS
```

```
'A' · 'B' · 'C' · 'D' · 'E' · 'F' · 'G' · 'H' · 'I' · 'J' · 'K' · 'L' · 'M' · 'N' · 'O' · 'P' · 'Q' · 'R' · 'S' · 'T' · 'U' · 'V' · 'W' ·
'X' · 'Y' · 'Z'
```

In [80]:

```
letters[2:4]
```

```
'b' · 'c' · 'd'
```

In [81]:

```
LETTERS[2:4]
```

```
'B' · 'C' · 'D'
```

In [82]:

```
LETTERS[c(2,26)]
```

```
'B' · 'Z'
```

In [ ]:

Common Math Functions

R has verious in-built functions for the Maths and can be use as needed in R code

In [83]:

```
x <- c(11.2, 33, 55.10, -17.6)
```

In [86]:

```
max(x) # get the max number from x Vector
```

```
55.1
```

In [88]:

```
min(x) # get the min number from x Vector
```

```
-17.6
```

In [89]:

```
abs(x) # get the absolute values for each elements
```

```
11.2 · 33 · 55.1 · 17.6
```

In [92]:

```
sum(x) # sum of all the elements
```

```
81.7
```

In [93]:

```
prod(x) # Multiplication of each element
```

```
-358423.296
```

In [94]:

```
y <- c(4,9,16,25,36)
sqrt(y) # Get the square root
```

```
2 · 3 · 4 · 5 · 6
```

```
In [ ]:
```

Sample Function

- sample() functions helps to perfom randon selection of elements in Vectors
- Sample functions is mainly used to simulate the data

sample(x= ,size=, replace=TRUE)

- x : is vector
- size : number of sample to select randomly
- replace : (TRUE) sampling with Replacement or (FALSE) without replacement

```
In [100…  sample(1:20, size=10, replace=TRUE) #Sampling with replacement
```

```
13 · 1 · 4 · 19 · 8 · 11 · 15 · 15 · 20 · 5
```

```
In [102…  sample(1:20, size=10, replace=FALSE) #Sampling without replacement
```

```
11 · 6 · 14 · 18 · 20 · 9 · 7 · 15 · 5 · 12
```

```
In [ ]:
```

Operation on Vectors

```
In [103…  z <- 1:4
          y <- 6:9
```

```
In [104…  z+y # element wise addition of vectors
```

```
7 · 9 · 11 · 13
```

```
In [108…  z-y # Substraction
```

```
-5 · -5 · -5 · -5
```

```
In [109…  z*y # Multiplication
```

```
6 · 14 · 24 · 36
```

```
In [110…  z / y # Division
```

```
0.166666666666667 · 0.285714285714286 · 0.375 · 0.444444444444444
```

Special Operator

- mod (%%) # Modulo operator to get the reminder on division operator
- IN (%in%) IN operator to check the existence of element in vector
- Integer division (%/%) operator to perform integer division

In [111…

```
x <- 1:10

element_to_check <- 4
```

In [113…

```
element_to_check %in% x
```

TRUE

In [119…

```
## Modulo operator to ge the reminder
5 %% 3
```

2

In [120…

```
5/3 # Normal division
```

1.66666666666667

In [121…

```
5 %/% 3 #Integer Division
```

1

In [ ]:

```

```

Relation Operator

- To compare an element with vector. R has relational operator to compare the element and result you get is boolean (TRUE/FALSE)

> : Greater than

>= : Greater then or equal

\< : Less than

\<= : Less then or equal

\== : Equal to

!= : Not equal to

In [122…

```
x <- 1:10
```

In [123…

```
x>2
```

FALSE · FALSE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE

In [124…

```
x>=2
```

FALSE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE

In [125…

```
x<2
```

TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE

In [126…

```
x<=2
```

```
TRUE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE
```

In [127…

```
x==2
```

```
FALSE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE
```

In [128…

```
x!=2
```

```
TRUE · FALSE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE
```

In [ ]:

```

```

Display element based on condition, we can pass relation operator and condition as vector index

In [156…

```
x[x>2]
```

```
3 · 4 · 5 · 6 · 7 · 8 · 9 · 10
```

In [157…

```
x[x==2]
```

```
2
```

In [158…

```
x[x!=2]
```

```
1 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10
```

In [ ]:

```

```

Logical Operator

To combine multiple condition we have Logical AND OR NOT operator

- AND : if all the conditions are on vector meets then TRUE otherwise FALSE
- OR : if at-least one coditions are meets then TRUE otherwise FALSE
- NOT : Not used to do negation/reverses of vectors

In [159…

```
x <- 1:10
```

In [162…

```
(x>2) & (x<7)  # Condition with & AND
```

```
FALSE · FALSE · TRUE · TRUE · TRUE · TRUE · FALSE · FALSE · FALSE · FALSE
```

In [163…

```
(x>2) | (x<7)  # Condition with | OR
```

```
TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE · TRUE
```

In [171…

```
!(x>2) # Checking with NOT
```

```
TRUE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE
```

In [172...

```
x[(x>2) & (x<7)]
```

```
3 · 4 · 5 · 6
```

In [173...

```
x[(x>2) | (x<7)]
```

```
1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10
```

In [174...

```
x[!(x>2)]
```

```
1 · 2
```

In [ ]:

In [ ]:

Matrices

A Matrices is a simply several vectors stored together in form of Rows & Cols.

matrix() function used to create matrix.

matrix(data=, nrow=, ncol=, byrow=FALSE)

- byrow is matrix element arrangement. if byrow=TRUE means element is get arrange in row wise else column wise arragment for the matrix

In [186...

```
m <- matrix(data=c(1,2,3,4,5,6),
            nrow=3,
            ncol=2)
```

In [187...

```
m
```

```
   A
matrix:
 3 × 2
of type
  dbl
 1   4
 2   5
 3   6
```

In [188...

```
m <- matrix(data=c(1,2,3,4,5,6),
            nrow=3,
            ncol=2,
            byrow=FALSE)
```

In [189...

```
m
```

```
A
matrix:
3 × 2
of type
dbl
1   4
2   5
3   6
```

In [192…

```
m <- matrix(data=c(1,2,3,4,5,6),
            nrow=3,
            ncol=2,
            byrow=TRUE)
## Elemenet filling direction is byrow
```

In [193…

```
m
```

```
A
matrix:
3 × 2
of type
dbl
1   2
3   4
5   6
```

In [ ]:

```

```

Matrix using cbind & rbind function

- cbind() function to bind the vectors by columns
- rbind() function to bind the vectors by rows

In [194…

```
a <- c(1,2,3)
b <- c(4,5,6)

rbind(a,b)
```

```
A matrix: 2 ×
3 of type dbl
a   1   2   3
b   4   5   6
```

In [195…

```
cbind(a,b)
```

A matrix: 3 × 2 of type dbl

a	b
1	4
2	5
3	6

In [ ]:

```
```

Dimension of Matrix

dim() functions to get the dimension of matrix

- dim() functions gives no of rows & columns of matrix

In [197…

```
dim(m)
```

```
3 · 2
```

In [ ]:

```
```

Accessing Element of Matrix

matrix_variable[row_index, column_index]

In [201…

```
m[1:2,] #1 & 2 rows with all the columns
```

A matrix: 2 × 2 of type dbl

1	2
3	4

In [204…

```
m[1,1] #1st row & 1st column
```

```
1
```

In [205…

```
m[3,2] #3rd row & 2nd column
```

```
6
```

In [206…

```
dim(m)
```

3 · 2

```
In [ ]:
```

Matrix Operation

```
In [209…  a <- matrix(c(1,2,3,4,5,6), nrow=3, ncol=2)
          b <- matrix(c(11,22,33,44,55,66), nrow=3, ncol=2)
```

Transpose

- t() function to get the transpose of a matrix

```
In [213…  t(a)
```

```
A matrix: 2
  × 3 of
 type dbl
 1   2   3
 4   5   6
```

```
In [214…  # Scaller multiplication
          a * 2
```

```
   A
 matrix:
3 × 2 of
type dbl
 2    8
 4   10
 6   12
```

Arithmatic Operation

Arithmatic operation on matrix takes place on element wise. 1st element of matrix A get add/sub/mul/div with 1st element of matrix B...same for all the elements of matrix

```
In [219…  a + b
```

```
A matrix:
 3 × 2 of
type dbl
 12   48
 24   60
 36   72
```

```
In [220…  a - b
```

```
A matrix: 3
× 2 of type
    dbl

-10   -40

-20   -50

-30   -60
```

```
In [221…  a * b
```

```
A matrix: 3
  × 2 of
type dbl

11   176

44   275

99   396
```

```
In [222…  a / b
```

```
A matrix: 3 × 2 of type dbl

0.09090909   0.09090909

0.09090909   0.09090909

0.09090909   0.09090909
```

```
In [ ]:
```

Matrix Multiplication in General Mathematics

we used special multiplication operation to perform matrix multiplication.

\%*% Special Multiplication operator

```
In [230…  a <- matrix(c(2,6,5,1,2,4), nrow=2, ncol=3)
          b <- matrix(c(5,-1,1,-3,1,5), nrow=3, ncol=2)
```

```
In [231…  a
```

```
A matrix: 2
  × 3 of
type dbl

2   5   2

6   1   4
```

```
In [232…  b
```

A matrix: 3 × 2 of type dbl

5	-3
-1	1
1	5

In [233…

```
#Matrix Mutliplication

a %*% b
```

A matrix: 2 × 2 of type dbl

7	9
33	3

In [ ]:

```

```

DataFrame

- A dataframe is R's most natural ways of presenting a dataset, and it's a collection of recorded observation for one or more variables.
- Data Frame contains data in two dimensional (Row X Col) format with row index and column index

data.frame() functions helps to create dataframe

In [14]:

```
name <- c("John", "Nick", "Dom")
age <- c(35,25,40)
country <- c("USA", "UK", "USA")

df <- data.frame(name, age, country)
```

In [15]:

```
print(df)
```

```
  name age country
1 John  35     USA
2 Nick  25      UK
3  Dom  40     USA
```

In [17]:

```
View(df)
```

Accessing Element from DataFrame

dataframe_var[row-index , col-index]

- If user is not passing either row or columns index then it display all the rows/columns

In [18]:
```
df[] #Display all the observation with all columns, No row index & column index so it
```

```
A data.frame: 3 × 3
name    age    country
<fct>   <dbl>  <fct>
John    35     USA
Nick    25     UK
Dom     40     USA
```

In [19]:
```
df[1,] #Display first observation with all columns
```

```
  A data.frame: 1 × 3
  name    age    country
  <fct>   <dbl>  <fct>
1 John    35     USA
```

In [20]:
```
df[1, 3] #Display first observation with only 3rd column
```

```
USA
▶ Levels:
```

In [21]:
```
df[2, 2:3] #Display second observation with 2 to 3 columns
```

```
  A data.frame: 1 × 2
  age    country
  <dbl>  <fct>
2 25     UK
```

In [22]:
```
df[1:2, 1] #Display 1 to 2 observation with 1st column
```

```
John · Nick

▶ Levels:
```

In [23]:
```
df[c(1,3),] #Display 1 & 3rd observation with all the columns , Column index is blank
```

```
  A data.frame: 2 × 3
  name    age    country
  <fct>   <dbl>  <fct>
1 John    35     USA
3 Dom     40     USA
```

In [ ]:
```

```

Accessing Variable/Columns in Dataframe using columns name

- $(dollor) symbol used to access the variables from Dataframe

dataframe-var$col-name

In [25]:

```
df$name
```

John · Nick · Dom

▶ **Levels**:

In [26]:

```
df$age
```

35 · 25 · 40

In [27]:

```
print(df$name)
```

```
[1] John Nick Dom
Levels: Dom John Nick
```

View the dimension & structure of Dataframe

- dim() functions to get the dimension of Dataframe
- str() functions to get the structure of dataframe which gives information about the no of rows & columns along with data types of each variable

In [30]:

```
dim(df)
```

3 · 3

In [31]:

```
str(df)
```

```
'data.frame':	3 obs. of  3 variables:
 $ name   : Factor w/ 3 levels "Dom","John","Nick": 2 3 1
 $ age    : num  35 25 40
 $ country: Factor w/ 2 levels "UK","USA": 2 1 2
```

In [ ]:

```

```

List

- List is powerful data strucutres. it can be used to group together any kind of data type & structures in R. i.e List can be created using combining Vectors, Matrix, Logical type, Complex type, String, Dataframe and list itself

list() function used to create list in R

In [36]:

```
name <- c("John", "Nick", "Dom")
age <- c(35,25,40)
country <- c("USA", "UK", "USA")

df <- data.frame(name, age, country)
```

```
m <- matrix(c(1,2,3,4,5,6), nrow=3, ncol=2)

logical_val <- c(TRUE, TRUE, FALSE)

char_val <- "Tumkur"

num <- 199.12

c_num <- 6+2i
```

In [37]:
```
my_list <- list(df, m, logical_val, char_val, num, c_num)
```

In [38]:
```
my_list
```

```
        A data.frame: 3 × 3
1.  name     age   country
    <fct>   <dbl>   <fct>
    John     35      USA
    Nick     25       UK
    Dom      40      USA

       A
    matrix:
     3 × 2
    of type
      dbl
2.  1   4
    2   5
    3   6

3. TRUE · TRUE · FALSE
4. 'Tumkur'
5. 199.12
6. 6+2i
```

In [ ]:

Accessing Element in List

in list we need to use double square brackets to access the elements

[[index]]

In [40]:
```
my_list[[1]] # Access the first element from a list and in above example 1st element i
```

A data.frame: 3 × 3

name	age	country
<fct>	<dbl>	<fct>
John	35	USA
Nick	25	UK
Dom	40	USA

In [42]:

```
my_list[[5]] # Accessing 5th element from a list and 5th element is Number in above ex
```

199.12

In [44]:

```
length(my_list) # No of element List has
```

6

In [ ]:

```

```

Factor

- Factor represent categorical variable and are used as grouping indicator. It stores the vector along with the distinct values of the elements in the vectors as lables.
- Factors can be useful in the columns which has a limited number of unique values. ie. Male/Female, True/False, High/Medium/Low..etc
- Factor are useful in data analysis for statistical modeling . Factors are self describing hence using Factor with label is better than using integer. ie. Having variable values "yes" and "no" ,is better than a variable that has values 1 and 2

factor(v=, levels=, labels=, exclude=NA) function used to create a Factors in R

In [45]:

```
data <- c("Birla", "Tata", "Ambani", "Ambani", "Tata", "Birla", "Ambani", "Infy" ,"TCS
```

In [46]:

```
data
```

'Birla' · 'Tata' · 'Ambani' · 'Ambani' · 'Tata' · 'Birla' · 'Ambani' · 'Infy' · 'TCS'

In [47]:

```
factor_data <- factor(data)
```

In [48]:

```
factor_data
```

Birla · Tata · Ambani · Ambani · Tata · Birla · Ambani · Infy · TCS

▶ **Levels**:

In [52]:

```
nlevels(factor_data)
```

5

In [54]:

```
levels(factor_data)
```

```
'Ambani' · 'Birla' · 'Infy' · 'Tata' · 'TCS'
```

In [57]:
```
a= seq(from=5, to=-11, by= -0.3)
```

In [58]:
```
sort(a)
```

```
-10.9 · -10.6 · -10.3 · -10 · -9.7 · -9.4 · -9.1 · -8.8 · -8.5 · -8.2 · -7.9 · -7.6 · -7.3 · -7 · -6.7 · -6.4 · -6.1 ·
-5.8 · -5.5 · -5.2 · -4.9 · -4.6 · -4.3 · -4 · -3.7 · -3.4 · -3.1 · -2.8 · -2.5 · -2.2 · -1.9 · -1.6 · -1.3 · -1 · -0.7 ·
-0.399999999999999 · -0.0999999999999996 · 0.2 · 0.5 · 0.8 · 1.1 · 1.4 · 1.7 · 2 · 2.3 · 2.6 · 2.9 · 3.2 ·
3.5 · 3.8 · 4.1 · 4.4 · 4.7 · 5
```

Exercise

1. Install following packages in R
   jsonify, RMySQL, haven, XML,readxl

1. Create and store a sequence of values from 5 to -11 that progresses in step of 0.3

1. Display the sequence element which created in point#2, sort in increasing order

1. Repeat vector c(-1,3,-5,7,-9) twice with each element repeated 10 times and store the result in a variable

1. Find the lenght of vector which is created in point#4

1. Construct and store a 4 x 2 matrix that’s filled row-wise with the values 4.3, 3.1, 8.2, 8.2, 3.2, 0.9, 1.6, and 6.5, in that order.

1. Store the bottom four elements of point#6 as a new 2 x 2 matrix.w

1. Calculate the following:

$$\frac{2}{7}\left(\begin{bmatrix}1 & 2\\ 2 & 4\\ 7 & 6\end{bmatrix} - \begin{bmatrix}10 & 20\\ 30 & 40\\ 50 & 60\end{bmatrix}\right)$$

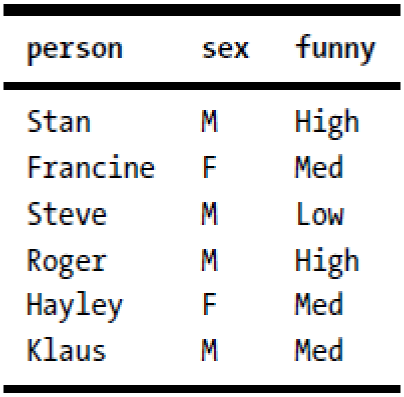
1. Store the following vector of 15 values as an object in your workspace:
   c(6,9,7,3,6,7,9,6,3,6,6,7,1,9,1). Identify the following elements:

- Those equal to 6
- Those greater than or equal to 6
- Those less than 6 + 2
- Those not equal to 6

1. Store the vector c(7,1,7,10,5,9,10,3,10,8) as foo. Identify the elements greater than 5 OR equal to 2.

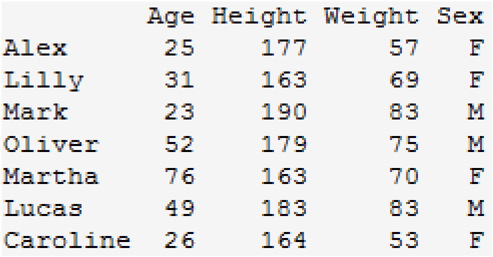
1. Store the vector c(8,8,4,4,5,1,5,6,6,8) as bar. Identify the elements less than or equal to 6 AND not equal to 4.

1. Create and store this data frame as df in your R workspace, and display all the person names



person	sex	funny
Stan	M	High
Francine	F	Med
Steve	M	Low
Roger	M	High
Hayley	F	Med
Klaus	M	Med

1. Create and store this data frame as b in R workspce and display the dimentions and structure of b



```
          Age Height Weight Sex
Alex       25    177     57   F
Lilly      31    163     69   F
Mark       23    190     83   M
Oliver     52    179     75   M
Martha     76    163     70   F
Lucas      49    183     83   M
Caroline   26    164     53   F
```