R Software

WHAT IS R?

R is an environment for data manipulation, statistical computing, graphics display and data analysis.

LEARNING R:

- If you need help with a function, then type question mark followed by the name of the function. For example, ?read.table
- Sometimes, you want to search by the subject on which we want help (e.g. data input). In such a case, type help.search("data input")
- 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.
- The find function tells us what package something is in.
  For example
  find("lowess")
- The apropos returns a character vector giving the names of all objects in the search list that match your enquiry. apropos("lm")
- To see a worked example just type the function name, e.g., lm
  example("lm")
- This can be useful for seeing the type of things that R can do.
  demo(graphics)

LIBRARYES IN R:

- R provides many functions and one can also write own. Functions and datasets are organised into libraries. To use a library, simply type the library function with the name of the library in brackets.
  For example, to load the spatial library type: library(spatial)
- The command packageDescription() provides the description file of a package. Here is how we find the description of the spatial library: packageDescription("spatial")
- It is easy to use the help function to discover the contents of library packages. Here is how we find out about the contents of the spatial library: library(help=spatial)
- To install any package, use the command install.packages() example: install.packages("boot").
- The command installed.packages() is used to see the installed packages on the computer.
- The command remove.packages("package") is used to see remove the installed packages on the computer.
- The command detach("package:cluster", unload=TRUE) is used to unload the installed packages on the computer.
- The command update.packages() is used to see update the installed packages on the computer. update.packages("cluster").

BASIC OPERATIONS_1:

- to see the contents in the working directory in R Type 'ls()'
- to see the location of the working directory in R Use 'getwd()'
- to set the working directory in R Use 'setwd()' to change the working directory in R.

example: setwd("c:/Rcourse")

- to interrupt a running computation in R press *esc*
- To clean the contents on the GUI window, press ctrl key and L
- To search the web for information and answers regarding R, use the RSiteSearch For example, to know about `mode', type RSiteSearch("mode")
- We assign names to variables when analyzing any data. It is good practice to remove the variable names given to any data frame at the end each session in R. This way, variables with same names but different properties will not get in each other's way in subsequent work. rm() command removes variable names. For example, rm(x,y,z) removes the variables x, y and z. use rm(list=ls()) to remove everything.
- Enter q() to quit.

BASIC OPERATIONS_2:

- To know if a value is a number, use is.numeric()
- To know if a value is a character, use is.character()

  Example:

```
x = 20
is.numeric(x)
[1] TRUE

is.character(x)
[1] FALSE

y = "apple"
is.character(y)
[1] TRUE

is.numeric(y)
[1] FALSE
```

- To convert a value as a number, use as.numeric()
- To convert a value as a character, use as.character()

  Example:

```
x = 20
is.numeric(x)
[1] TRUE

y = as.character(x)
is.numeric(y)
[1] FALSE

is.character(y)
[1] TRUE

y
[1] "20"
```

R – Software

- #: The character # marks the beginning of a comment.All characters until the end of the line are ignored. Example: # mu is the mean
- The command c(1,2,3,4,5) combines the numbers 1,2,3,4 and 5 to a vector.
- Command mode() explains the type or storage mode of an object. Command storage.mode() returns the storage mode of its argument.

  Example:

```
x = 6
x
[1] 6
mode(x)
[1] "numeric"

y = "apple"
y
[1] "apple"
mode(y)
[1] "character"
```

  Following modes are available:
  - "logical",
  - "integer",
  - "double",
  - "complex",
  - "raw",
  - "character",
  - "list",
  - "expression",
  - "name",
  - "symbol" and
  - "function".

- Some calculations yield result as infinity (∞). For example, 3/0. In R is.finite() and is.infinite() are used to know if an outcome is finite or infinite, respectively.

  Example:

```
x = 5+Inf
is.finite(x)
[1] FALSE

is.infinite(x)
[1] TRUE
```

R AS A CALCULATOR:

- R computations can be carried out with
  - Scalar versus scalar.
  - Scalar versus data vectors.
  - Data vectors versus data vectors.
- BODMAS rule is applicable. B-Brackets, O-Orders (powers), D-Division, M-Multiplication,

A-Addition, S-Subtraction. The mathematical expressions with multiple operators are solved from left to right in this order. Only ( ) brackets are used for BODMAS. No brackets { } and [ ] are used in BODMAS.

Example:

```
(2+3)*5 + 5 - 10 # Command for BODMAS
[1] 20 # Output
```

- When you add or subtract or divide or multiply a scalar with a value it is applied to all variables. (Vector with Scalar)

  Example:

```
c(12,13,15,17) / 10
[1] 1.2 1.3 1.5 1.7
```

- (Vector with Vector) Here is how you do it:

  Example:

```
c(2,3,5,7) + c(-2,-3, -5, 8)  # 2+(-2), 3+(-3), 5+(-5), 7+8
[1] 0 0 0 15

c(2,3,5,7) + c(-2,-3) # 2+(-2), 3+(-3), 5+(-2), 7+(-3)
[1] 0 0 3 4

c(2,3,5,7) + c(-2,-3, -5) # 2+(-2), 3+(-3), 5+(-5), 7+(-2)
[1] 0 0 3 4
```

- 2^3 or 2**3 will provide 2 powered by 3.

  Example:

```
2^3 # Command for power operator
[1] 8 # Output

2**3 # Command for power operator
[1] 8 # Output
```


- Power with scalar:

```
c(2,3,5,7)^2 # command: application to a vector
[1] 4 9 25 49 # output
```

  Power with vector:

```
c(2,3,5,7)^c(2,3) # !!ATTENTION! Observe the operation 2**2, 3**3, 5**2, 7**3
[1] 4 27 25 343 # output
```

  The scalar and vector operations work similar for all types of operations between operands

- Integer Division: Division in which the fractional part (remainder) is discarded

  Operator: %/%

```
2 %/% 2
[1] 1

5 %/% 2
[1] 2

7 %/% 3
[1] 2
```

- Modulo Division: modulo operation finds the remainder after division of one number by another.

  Operator: %%

  2 %% 2
  [1] 0
  3 %% 2
  [1] 1

  7 %% 3
  [1] 1

  7 %% 4
  [1] 3

BUILT IN FUNCTIONS AND ASSINEMENTS:

- Operator: max

  max(1.2, 3.4, -7.8)
  [1] 3.4

  max(c(1.2, 3.4, -7.8))
  [1] 3.4

- Operator: min

  min(1.2, 3.4, -7.8)
  [1] -7.8

  min(c(1.2, 3.4, -7.8))
  [1] -7.8

- Operator: mean

  mean(2, 3, 4)
  [1] 2

  mean(c(2, 3, 4))
  [1] 3

  See the difference in use of c command.

  Suggestion: Always use c command to input more than one data value.

Overview Over Further Functions

`abs()`	Absolute value
`sqrt()`	Square root
`round(), floor(), ceiling()`	Rounding, up and down
`sum(), prod()`	Sum and product
`log(), log10(), log2()`	Logarithms
`exp()`	Exponential function
`sin(), cos(), tan(), asin(), acos(), atan()`	Trigonometric functions
`sinh(), cosh(), tanh(), asinh(), acosh(), atanh()`	Hyperbolic functions

- Operator: abs for finding the absolute values

  abs(-4)

```
[1] 4

abs(c(-1,-2,-3,4,5))
[1] 1 2 3 4 5
```

- Operator: sqrt for finding the square root of values

```
sqrt(4)
[1] 2

sqrt(c(4,9,16,25))
[1] 2 3 4 5
```

- Operator: sum for finding the sum of values

```
sum(c(2,3,5,7))
[1] 17
```

- Operator: prod for finding the product of values

```
prod(c(2,3,5,7))
[1] 210
```

- Operator: round for finding the round off values

```
round(1.23)
[1] 1

round(1.83)
[1] 2
```

- Operator: log for finding the natural log (ln) of values

```
log(10)
[1] 2.302585

log(exp(1))
[1] 1

log(c(10, 100, 1000))
[1] 2.302585 4.605170 6.907755
```


MATRICES:

- In R, a 4 × 2-matrix X can be created with a following command:

```
x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8) ) #default column wise
x
     [,1] [,2]
[1,] 1    5
[2,] 2    6
[3,] 3    7
[4,] 4    8

OR
x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8), byrow = TRUE) # for row wise
x
     [,1]  [,2]
[1,]  1     2
```

```
[2,]  3    4
[3,]  5    6
[4,]  7    8
```

- One can access a single element of a matrix with x[i,j]:

```
x[3,2]
[1] 7
```

PROPERTYES OF A MATRIX:

- We can get specific properties of a matrix:

```
dim(x) # tells the dimension of matrix
[1] 4 2

nrow(x) # tells the number of rows
[1] 4

ncol(x) # tells the number of columns
[1] 2

mode(x) # Informs the type or storage mode of an object, e.g., numerical, logical etc.
[1] "numeric"
```

attributes provides all the attributes of an object

```
attributes(x) # Informs the dimension of matrix
$dim [1] 4 2
```


- 'matrix' creates a matrix from the given set of values.
  'as.matrix' attempts to turn its argument into a matrix.
  'is.matrix' tests if its argument is a (strict) matrix.
- Renaming the row and column names:
  rownames(x) Renames the row names.
  colnames(x) Renames the column names.
  Example:

```
x = matrix( nrow=4, ncol=3, data=c(1:12) )
x
[,1] [,2] [,3]
[1,] 1 5 9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
rownames(x) = c("r1", "r2", "r3", "r4")
x
[,1] [,2] [,3]
r1 1 5 9
r2 2 6 10
r3 3 7 11
r4 4 8 12

colnames(x) = c("c1", "c2", "c3")
```

```
x
c1 c2 c3
r1 1 5 9
r2 2 6 10
r3 3 7 11
r4 4 8 12
```

- Construction of a diagonal matrix, here the identity matrix of a dimension 3:
  Example:

```
d = diag(1, nrow=3, ncol=3)
d
[,1] [,2] [,3]
[1,] 1 0 0
[2,] 0 1 0
[3,] 0 0 1
```

- Transpose of a matrix X: X' (t())
  Example:

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T )
x
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
xt = t(x)
xt
[,1] [,2] [,3] [,4]
[1,] 1 3 5 7
[2,] 2 4 6 8
```


- Finding the row and column sums
  rowSums(x) Finds the sum of numbers in rows
  colSums(x) Finds the sum of numbers in columns
  Example:

```
x = matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))
x
[,1] [,2]
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
rowSums(x)
[1] 6 8 10 12
colSums(x)
[1] 10 26
```

- rowMeans(x) Finds the means of rows
  colMeans(x) Finds the means of columns
  Example:

```
x = matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))
```

```
x
[,1] [,2]
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
rowMeans(x)
[1] 3 4 5 6
colMeans(x)
[1] 2.5 6.5
```

- Matrix Addition, subtraction, multiplication, division with scalar
  Every element is added with the scalar
  Example:

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
x
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
x+5
[,1] [,2]
[1,] 6 7
[2,] 8 9
[3,] 10 11
[4,] 12 13
x-5
[,1] [,2]
[1,] -4 -3
[2,] -2 -1
[3,] 0 1
[4,] 2 3
x*5
[,1] [,2]
[1,] 5 10
[2,] 15 20
[3,] 25 30
[4,] 35 40
x/5
[,1] [,2]
[1,] 0.5 1
[2,] 1.5 2
[3,] 2.5 3
[4,] 3.5 4
```

- Matrix addition, subtraction with a matrix

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
y = matrix(nrow=4, ncol=2, data=11:18, byrow=T)
```

```
x
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
Y
[,1] [,2]
[1,] 11 12
[2,] 13 14
[3,] 15 16
[4,] 17 18
x + y
[,1] [,2]
[1,] 12 14
[2,] 16 18
[3,] 20 22
[4,] 24 26
x - y
[,1] [,2]
[1,] -10 -10
[2,] -10 -10
[3,] -10 -10
[4,] -10 -10
```


- Matrix multiplication Operator: %*%
  Example:

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
y = matrix(nrow=2, ncol=4, data=11:18, byrow=T)
x
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
y
[,1] [,2] [,3] [,4]
[1,] 11 12 13 14
[2,] 15 16 17 18
x%*%y
[,1] [,2] [,3] [,4]
[1,] 41 44 47 50
[2,] 93 100 107 114
[3,] 145 156 167 178
[4,] 197 212 227 242
```

- Cross Product of a matrix X, X'X, with crossprod()

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
x
```

```
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
crossprod(x)        #it is faster than t(x)%*%x
[,1] [,2]
[1,] 84 100
[2,] 100 120
```

- Concatenating matrices row wise: rbind(x, y)
  Concatenating matrices column wise: cbind(x, y)

```
x = matrix(nrow=3, ncol=2, data=1:6, byrow=T)
y = matrix(nrow=3, ncol=2, data=11:16, byrow=T)
x
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
y
[,1] [,2]
[1,] 11 12
[2,] 13 14
[3,] 15 16
rbind(x, y)
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 11 12
[5,] 13 14
[6,] 15 16
```


- solve() finds the inverse of a positive definite matrix

```
y = matrix( nrow = 2, ncol = 2, byrow = T,
data = c(84,100,100,120))
y
[,1] [,2]
[1,] 84 100
[2,] 100 120
solve(y)
[,1] [,2]
[1,] 1.50 -1.25
[2,] -1.25 1.05
```

- eigen() finds the eigen values and eigen vectors of a positive definite matrix

```
y = matrix( nrow = 2, ncol = 2, byrow = T, data = c(84,100,100,120))
y
[,1] [,2]
[1,] 84 100
```

[2,] 100 120
14
eigen(y)
$values
[1] 203.6070864 0.3929136
$vectors
[,1] [,2]
[1,] 0.6414230 -0.7671874
[2,] 0.7671874 0.6414230

LOGICAL OPERATORS:

The following table shows the operations and functions for logical comparisons (True or False).

Operator	Executions
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Exactly equal to
!=	Not equal to
!	Negation (not)
&, &&	and
|, ||	or

Operator	Executions
xor(x,y)	either... or (exclusive)
isTRUE(x)	test if x is TRUE
isFALSE(x)	test if x is FALSE
TRUE	true
FALSE	false

- || Operates only on the first element and not on remaining elements.
  | Operates on all the elements.
  x = c(8,18)
  (x < 10) || (x < 2)
  [1] TRUE
  (x < 10) | (x < 2)
  [1] TRUE FALSE
- x = 1:6 # Generates x=1,2,3,4,5,6
  (x > 2) & (x < 5) # Checks whether the values are greater than 2 and less than 5
  [1] FALSE FALSE TRUE TRUE FALSE FALSE
  x[(x > 2) & (x < 5)] # Finds which values are greater than 2 and smaller than 5
  [1] 3 4

MISSING DATA HANDLING:

- R represents missing observations through the data value NA We can detect missing values using is.na.
  Example:
  x = NA # assign NA to variable x
  is.na(x) # is it missing?
  [1] TRUE
  x = c(11, NA, 13, NA)
  is.na(x)
  [1] FALSE TRUE FALSE TRUE
- How to work with missing data
  x = c(11,NA,13, NA) # vector
  mean(x)

```
[1] NA

mean(x, na.rm = TRUE) # NAs can be removed
[1] 12
```

- NA is a placeholder for something that exists but is missing.
  NULL stands for something that never existed at all.
- To identify the location of NAs, use which() function as
  ```
  which(is.na( ))
  x = c(11,NA,13,NA)
  x
  [1] 11 NA 13 NA
  which(is.na(x))
  [1] 2 4
  ```
- To count of NAs the number of NAs , use sum() function as
  ```
  sum(is.na( ))
  x = c(11,NA,13,NA)
  x
  [1] 11 NA 13 NA
  sum(is.na(x))
  [1] 2
  ```
- To find complete cases, use complete.cases() function which returns a logical vector identifying rows which are complete cases.
  ```
  x = c(11,NA,13,NA)
  x                    #similar to is.na() but opposite
  [1] 11 NA 13 NA
  complete.cases(x)
  [1] TRUE FALSE TRUE FALSE
  ```
- The function na.omit() returns the object with listwise deletion of missing values. Drop out any rows with missing values anywhere in them and forgets them forever.
  ```
  x = c(11,NA,13,NA)
  y = na.omit(x)
  y
  [1] 11 13
  attr(,"na.action")
  [1] 2 4
  attr(,"class")
  [1] "omit"
  mean(x)
  [1] NA
  mean(y)
  [1] 12
  ```

CONDITIONAL EXICUTIONS:

- Conditional execution: if()
  Syntax:
  ```
  if (condition) {execute commands if condition is TRUE}
  ```

if() should not be applied when the condition being evaluated is a vector. It is best used only when meeting a single element condition.

Example:

```
x = 5
if (x > 4) x * 3
[1] 15
x = 3
if (x > 4) x * 3
```

- Conditional execution: if else()

Syntax:

```
if (condition) {executes commands if condition is TRUE}
else { executes commands if condition is FALSE }
```

Example:

```
x = 3
if ( x==3 ) { x = x-1 } else { x = 2*x }
x
[1] 2
x = 6

if(x > 3){
print("The value is more than 3")
} else {
print("The value is less than 3")
}
[1] "The value is more than 3"
```


- Conditional execution: Nested if else if()

Syntax:

```
if (condition1) {
executes commands if condition1 is TRUE
} else if (condition2) {
executes commands if condition2 is TRUE
} else if (condition3) {
executes commands if condition3 is TRUE
}
... ...
else {
executes commands if all conditions are FALSE
}
```

Example:

```
x = 5
if ( x==3 ) {
x = x-1
} else if ( x < 3 ) {
x = x+5
} else { x = 2*x }
x
[1] 10
```

R – Software

- Conditional execution: ifelse()
  Syntax:
  ifelse(test, yes, no)
  Example:
  x = 1:10
  x
  [1] 1 2 3 4 5 6 7 8 9 10
  ifelse(x<6, x^2, x+1)
  [1] 1 4 9 16 25 7 8 9 10 11
- Conditional execution: switch()
  Syntax:
  switch(expr, case1, case2,....)
  Example:
  switch("colour", "colour" = "blue", "gender" ="male", "volume" = 50)
  [1] "blue"
  switch(1,"apple", "banana", "orange")
  [1] "apple"
- Conditional execution: which() The which() function returns the position of the elements in a logical vector which are TRUE.
  Syntax:
  which(x, arr.ind, useNames)
  x: Specified input logical vector
  arr.ind: logical, returns the array indices if x is an array.
  useNames: logical, says the dimension names of an array.
  Example:
  x = c(10,15,8,14,6,12)
  x
  [1] 10 15 8 14 6 12
  which(x == 14)
  [1] 4
  which(x != 12)
  [1] 1 2 3 4 5

  x = matrix(nrow=3, ncol=3, data=1:9)
  x
  [,1] [,2] [,3]
  [1,] 1 4 7
  [2,] 2 5 8
  [3,] 3 6 9
  rownames(x)=c("row", "col")
  which(x %% 2 == 1)
  [1] 1 3 5 7 9
  which(x %% 2 == 1, arr.ind = TRUE)
  row col
  [1,] 1 1
  [2,] 3 1
  [3,] 2 2

```
[4,] 1 3
[5,] 3 3
```

LOOPS:

- for loop: Syntax for (name in vector) {commands to be executed}
  Example:
  ```
  for ( i in 1:5 ) { print( i^2 ) }
  [1] 1
  [1] 4
  [1] 9
  [1] 16
  [1] 25
  ```
  Note : print is a function to print the argument
  ```
  for ( i in c(2,4,6,7) ) { print( i^2 ) }
  [1] 4
  [1] 16
  [1] 36
  [1] 49
  ```
- break : Using the break command, we can stop the loop before it has looped through all the items.
  ```
  drink = c("coffee", "lemonade", "tea", "juice")
  for (x in drink) {
  if (x == "tea") {
  break
  }
  print(x)
  }
  [1] "coffee"
  [1] "lemonade"
  ```
- next : Using the next command, we can skip an iteration without terminating the loop.
  Suppose we want to skip lemonade.
  ```
  drink = c("coffee", "lemonade", "tea", "juice")
  for (x in drink) {
  if (x == "lemonade") {
  next
  }
  print(x)
  }
  [1] "coffee"
  [1] "tea"
  [1] "juice"
  ```
- while loop: Syntax while(condition){ commands to be executed as long as condition is TRUE }
  Example:
  ```
  i = 1
  while (i < 10) {
  print(i^2)
  i = i+2
  ```

```
}
[1] 1
[1] 9
[1] 25
[1] 49
[1] 81
```

- repeat loop : It executes for ever so it must be breaked.
  Syntax: repeat{ commands to be executed }
  Example:

```
i = 1
repeat{
print( i^2 )
i = i+2
if ( i > 10 )
break
}
[1] 1
[1] 9
[1] 25
[1] 49
[1] 81
```

FUNCTIONS:

- Syntax: Name <- function(Argument1, Argument2, ...)

```
{
expression(s)
}
```

Example:

```
abc = function(x,y){
x^2+y^2
}
abc(3,4)
[1] 25
abc(10, 10)
[1] 200
abc(-2, -3)
[1] 13
```

```
abc = function() {
for(i in 1:3) {
print(i^3)
}
}
abc()
[1] 1
[1] 8
```

[1] 27

SEQUENCE AND OTHER OPERATIONS:

- Syntax: seq()
  seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out = NULL, along.with = NULL, ...)
  Example:
  seq(from=20, to=10, by=-2)
  [1] 20 18 16 14 12 10

  seq(to=10, length=10)
  [1] 1 2 3 4 5 6 7 8 9 10

  seq(from=10, length=10, by=0.1)
  [1] 10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9
- Continuous sequences with constant unit increment and decrement
  Example:
  15:5
  [1] 15 14 13 12 11 10 9 8 7 6 5

  -1.23:-10
  [1] -1.23 -2.23 -3.23 -4.23 -5.23 -6.23 -7.23 -8.23 -9.23

  -5.23:6
  [1] -5.23 -4.23 -3.23 -2.23 -1.23 -0.23 0.77 1.77 2.77 3.77 4.77 5.77
- Assignment of an index-vector
  x = c(9,8,7,6) 9,8,7,6
  ind = seq(along=x)
  ind
  [1] 1 2 3 4
  Accessing a value in the vector through index vector
  Accessing an element of an index-vector
  x[ind[2]]
  [1] 8
- Sequences Of Dates
  Generating current time and date
  Sys.time() command provides the current time and date from the computer system.
  Sys.time()
  [1] "2021-11-29 21:23:57 IST"
  Sys.Date() command provides the current date from the computer system.
  Sys.Date()
  [1] "2021-11-29"
- Sequence of first day of years
  seq(as.Date("2010-01-01"), as.Date("2017-01-01"), by = "years")
  [1] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01"
  [5] "2014-01-01" "2015-01-01" "2016-01-01" "2017-01-01"
- Sequence of days

```
seq(as.Date("2017-01-01"), by = "days",length = 6)
[1] "2017-01-01" "2017-01-02" "2017-01-03" "2017-01-04"
[5] "2017-01-05" "2017-01-06"
```

- Sequence of months

```
seq(as.Date("2017-01-01"), by = "months",length = 6)
[1] "2017-01-01" "2017-02-01" "2017-03-01" "2017-04-01"
[5] "2017-05-01" "2017-06-01"
```

- Sequnce of years

```
seq(as.Date("2017-01-01"), by = "years",length = 6)
[1] "2017-01-01" "2018-01-01" "2019-01-01" "2020-01-01" #use -1 years to reverse order
[5] "2021-01-01" "2022-01-01"
```

- Sequene of alphabets

letters is used to find sequence of lowercase alphabets

```
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z“
```

letters[from_index:to_index] is used to find sequence of lowercase alphabets from a particular index to a specified index.

```
letters[1:3]
[1] "a" "b" "c"

letters[3:1]
[1] "c" "b" "a"

letters[21:23]
[1] "u" "v" "w“
```

LETTERS is used to find sequence of uppercase alphabets

```
LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```


REPEATES:

- The rep() function replicates numeric values, or text, or the values of a vector for a specific number of times. Syntax: rep(x) #replicates value in vector x or
  rep(x, times=n) #repeat x n times or rep(x, each=n) #repeat each cell n times
  Following commands repeat each cell for the desired length of the output vector
  rep(x, length.out=n) or rep(x, length=n) or rep_len(x, length.out)
  Example:

```
rep(3.5, times=10)
[1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5

rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4

x = 1:4
x
[1] 1 2 3 4
```

```
rep(x, times = 3)
[1] 1 2 3 4 1 2 3 4 1 2 3 4

rep(x, each = 3)
[1] 1 1 1 2 2 2 3 3 3 4 4 4
```

Every object is repeated a different number of times:

```
rep(1:4, 2:5)
[1] 1 1 2 2 2 3 3 3 3 4 4 4 4 4

ans = seq(from=2, to=8, by=2)
ans
[1] 2 4 6 8

rep(1:4, ans)
[1] 1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4

x = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
x
[,1] [,2]
[1,] 1 2
[2,] 3 4
rep(x, 3)
[1] 1 3 2 4 1 3 2 4 1 3 2 4

rep(2, length.out=5)
[1] 2 2 2 2 2

rep(2, length=5)
[1] 2 2 2 2 2

rep(c(2,3), length=5)
[1] 2 3 2 3 2

rep(c(2,3,4), length=5)
[1] 2 3 4 2 3
```


SORTING, ORDERING, MODE:

- Sort: sort function sorts the values of a vector in ascending order (by default) or descending order. Syntax: sort(x, decreasing = FALSE, ..,) or sort(x, decreasing = FALSE, na.last = NA, ...)
  Example:

```
y = c(8,5,7,6)
y
[1] 8 5 7 6
sort(y)
[1] 5 6 7 8
```

```
sort(y, decreasing = TRUE)
[1] 8 7 6 5
```

- Ordering: order function sorts a variable according to the order of variable.
  Syntax: order(x, decreasing = FALSE, ..,) or order(x, decreasing = FALSE, na.last = TRUE, ...)
  Example:

```
y
[1] 9 8 5 7 6
order(y)
[1] 3 5 4 2 1
order(y, decreasing = TRUE)
[1] 1 2 4 5 3
```

- Mode: Every object has a mode.
  The mode indicates how the object is stored in memory: as a
  number,
  character string,
  list of pointers to other objects,
  function etc.
  mode function gives us such information. Syntax: mode ()
  Example:

```
mode(factor(c("UP", "MP")) )
[1] "numeric"
mode(list("India", "USA"))
[1] "list"
mode(data.frame(x=1:2, y=c("India", "USA")))
[1] "list"
mode(print)
[1] "function"
```


LISTS:

- Lists can contain any kind of objects as well as objects of different types.
  Example:

```
x1 = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
x2 = matrix(nrow=2, ncol=2, data=5:8, byrow=T)
x1
[,1] [,2]
[1,] 1 2
[2,] 3 4
x2
[,1] [,2]
[1,] 5 6
[2,] 7 8
matlist = list(x1, x2)
matlist
[[1]]
[,1] [,2]
[1,] 1 2
[2,] 3 4
```

```
[[2]]
[,1] [,2]
[1,] 5 6
[2,] 7 8
matlist[1]
[[1]]
[,1] [,2]
[1,] 1 2
[2,] 3 4
matlist[2]
[[1]]
[,1] [,2]
[1,] 5 6
[2,] 7 8
```

An example of a list that contains different object types:

```
z1 = list( c("water", "juice", "lemonade"),rep(1:4, each=2), matrix(data=5:8, nrow=2,ncol=2, byrow=T) )
z1
[[1]]
[1] "water" "juice" "lemonade"
[[2]]
[1] 1 1 2 2 3 3 4 4
[[3]]
[,1] [,2]
[1,] 5 6
[2,] 7 8 10
```


- Access the elements of a list using the operator [[]]
  Following commands work.

```
z1[[1]]
[1] "water" "juice" "lemonade"
```

  Suppose we want to extract "juice". The command

```
z1[1][2] # Notice the positions of brackets
[[1]] NULL
```

  returns NULL instead of "juice", while

```
z1[[1]][2] # Notice the positions of brackets
[1] "juice"
```

  finally returns the desired result.
- Merging list:
  Example:

```
list12 = c(list1, list2)
list12
[[1]]
[1] 1
[[2]]
[1] 2
[[3]]
[1] 3
```

```
[[4]]
[1] "water"
[[5]]
[1] "juice"
[[6]]
[1] "lemonade"
```

- List To Vector: Converting list to vector. Use unlist() command.
  Example:

```
list1 = list(1,2,3)
list2 = list("water", "juice", "lemonade")
unlist(list1)
[1] 1 2 3
unlist(list2)
[1] "water" "juice" "lemonade"
mode(list1)
[1] "list"
mode(unlist(list1))
[1] "numeric"
```

- Appending: Appending list. Use append() command.
  Example:

```
list1 = list(1,2,3)
append(list1, 100)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 100
```


- Appending list after a position. Use append(, after) command.
  Example:

```
list1 = list(1,2,3)
append(list1, 100, after = 2)
[[1]]
[1] 1
[[2]]
[1] 2
[[3]]
[1] 100
[[4]]
[1] 3
```

- Removing: Removing from list at a position. Use listname[-position]
  Example:

```
list1 = list(1,2,3)
list2 = list("water", "juice", "lemonade")
list1[-2]
[[1]]
[1] 1
[[2]]
[1] 3
list2[-1]
[[1]]
[1] "juice"
[[2]]
[1] "lemonade"
```

- Accessing: Extracting from list. Use a range of indexes as listname [indexes]
  Example:

```
list1 = list(1,2,3,4,5,6)
list1[2:4]
[[1]]
[1] 2
[[2]]
[1] 3
[[3]]
[1] 4
list1[c(1,3,5)]
[[1]]
[1] 1
[[2]]
[1] 3
[[3]]
[1] 5
```


VECTOR INDEXING:

- The elements of a vector can be named. Using these names, we can access the vector elements.
  Example:

```
z = list(a1 = 1, a2 = "c", a3 = 1:3)
z
$a1
[1] 1
$a2
[1] "c"
$a3
[1] 1 2 3
names(z)
[1] "a1" "a2" "a3"
```

  Suppose want to change just the name of the third element.

```
z = list(a1 = 1, a2 = "c", a3 = 1:3)
names(z)[3] = "c2"
```

```
z
$a1
[1] 1
$a2
[1] "c"
$c2
[1] 1 2 3
```

- Empty Index:

```
x = 1:10
x
[1] 1 2 3 4 5 6 7 8 9 10
x[]
[1] 1 2 3 4 5 6 7 8 9 10
```

- Mixed Mode: List can be heterogeneous (mixed modes). We can start with a heterogeneous list, give it dimensions, and thus create a heterogeneous list that is a mixture of numeric and character data:

  Example:

```
ab = list(1, 2, 3, "X", "Y", "Z")
dim(ab) = c(2,3)
print(ab)
[,1] [,2] [,3]
[1,] 1 3 "Y"
[2,] 2 "X" "Z"
mode(print(ab))
[,1] [,2] [,3]
[1,] 1 3 "Y"
[2,] 2 "X" "Z"
[1] "list"
```


FACTORS:

- factor(x = character(), levels, labels = levels, exclude = NA, ...)

  levels : Determines the categories of the factor variable. Default is the sorted list of all the distinct values of x.

  labels : (Optional) Vector of values that will be the labels of the categories in the levels argument.

  exclude : (Optional) It defines which levels will be classified as NA in any output using the factor variable.

  Example:

  Suppose we roll a die seven times and observe the outcome in the vector y.

```
y = c(1, 4, 3, 5, 4, 2, 4)
```

  Possible values of upper face of die are 1 to 6 and we store them in a vector possible.dieface

```
possible.dieface = c(1, 2, 3, 4, 5, 6)
```

  We wish to label the rolls by the words “one”, “two”, ..., “six”. We put these labels in the vector labels.diefaces:

```
labels.dieface = c("one", "two", "three", "four", "five", "six")
```

  Construct the factor variable facy using the function factor:

```
facy = factor(y, levels = possible.dieface, labels = labels.dieface)
```

Observe the difference between a character vector and a factor.

```
facy
[1] one four three five four two four
Levels: one two three four five six
```

CLASS AND UNCLASS:

- A vector can be turned into a factor with the command as.factor:

```
x = c(3, 4, 5, 6, 1, 2, 3, 3, 4, 4, 5, 6)
x
[1] 3 4 5 6 1 2 3 3 4 4 5 6
y = as.factor(x)
y
[1] 3 4 5 6 1 2 3 3 4 4 5 6
Levels: 1 2 3 4 5 6
```

- class function : All objects in R have a class and function class reports it.

Example:

```
class function :
class(9)
[1] "numeric"
class("9")
[1] "character"
class(print)
[1] "function"
x = matrix(nrow=2, ncol=2, data=1:4)
class(x)
[1] "matrix" "array"
```


- unclass function: For example if an object has class "data.frame", it will be printed in a certain way, the plot() function will display it graphically in a certain way etc. unclass() is used to temporarily remove the effects of class.

Example:

```
brands = c("A","A","B","B","B","B","C")
brands
[1] "A" "A" "B" "B" "B" "B" "C"
brands_fac = factor(brands)
brands_fac
[1] A A B B B B C
Levels: A B C
unclass(brands_fac)
[1] 1 1 2 2 2 2 3
attr(,"levels")
[1] "A" "B" "C"
```

Suppose we have a vector of colors as

```
colours = c("blue", "green", "red")
colours
[1] "blue" "green" "red"
colours [unclass(brands_fac)]
[1] "blue" "blue" "green" "green" "green" "green" "red"
```

Recall brands = c("A","A","B","B","B","B","C")

- Example for an ordered factor:

```
income = ordered(c("high", "high", "low", "medium", "medium"), levels=c("low", "medium", "high") )
income
[1] high high low medium medium
Levels: low < medium < high
unclass(income)
[1] 3 3 1 2 2
attr(,"levels")
[1] "low" "medium" "high"
```

PRINT AND FORMAT:

- Try to understand from below examples

Example1:

```
print( sqrt(2) )
[1] 1.414214
print( sqrt(2), digits=5 )
[1] 1.4142
print( sqrt(2), digits=10 )
[1] 1.414213562
```

Example2:

```
print( format( 0.5, digits=10, nsmall=15 ) )
[1] "0.500000000000000"
```


Example3:

```
format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "centre")
[1] " A " " BB " " CCC " " DDDD "

format(c("A", "BB", "CCC", "DDDD"), width = 14, justify = "centre")
[1] " A " " BB " " CCC" " DDDD "
```

Example4:

```
format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "centre")
[1] " A " " BB " " CCC " " DDDD "
format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "left")
[1] "A " "BB " "CCC " "DDDD "
format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "right")
[1] " A" " BB" " CCC" " DDDD"
format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "none")
[1] "A" "BB" "CCC" "DDDD"
```

Example5:

```
format(1234567, big.mark = ",")
[1] "1,234,567"
format(123456789, big.mark = " ")
[1] "123 456 789"
```

DISPAY AND FORMATTING:

R – Software

- The print function has a significant limitation that it prints only one object at a time. The cat function is an alternative to print that lets you combine multiple items into a continuous output. Syntax: cat(... , file = "", sep = " ", fill = FALSE, labels = NULL, append = FALSE)
  Example:

```
print("The zero occurs at"); print(2*pi);
print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"

cat("The zero occurs at", 2*pi, "radians.", "\n")
The zero occurs at 6.283185 radians.

d = date()
cat("Today's date is:", d, "\n" )
Today's date is: Wed Dec 01 22:52:48 2021

x = 1:10
x
[1] 1 2 3 4 5 6 7 8 9 10
cat(x, sep =" ++ ")
1 ++ 2 ++ 3 ++ 4 ++ 5 ++ 6 ++ 7 ++ 8 ++ 9 ++ 10

cat(x, fill = 2, labels = paste("(", letters[1:10], "):"))

( a ): 1
( b ): 2
( c ): 3
( d ): 4
( e ): 5
( f ): 6
( g ): 7
( h ): 8
( i ): 9
( j ): 10
```

- paste(): The paste() function concatenates several strings together. It creates a new string by joining the given strings end to end. The result of paste() can be assigned to a variable. paste converts its arguments to character strings (via as.character) , and concatenates them (separating them by the string given by sep). Syntax: paste (..., sep = " ", collapse = NULL).
  Example:

```
paste(1:12)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"
as.character(1:12) #Alternative to paste
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"

paste("Everybody", "loves", "R Programming.", sep="===")
[1] "Everybody===loves===R Programming."
```

```
names = c("Prof. Singh", "Mr. Venkat", "Dr. Jha")
names
[1] "Prof. Singh" "Mr. Venkat" "Dr. Jha"
paste(names, "is", "a good", "person.")
[1] "Prof. Singh is a good person." "Mr. Venkat is a good person." "Dr. Jha is a good person."
```

- When we want to join even those combinations into one, big string. The collapse parameter defines a top-level separator and instructs paste to concatenate the generated strings using that separator:

```
names = c("Prof. Singh", "Mr. Venkat", "Dr. Jha")
paste(names, "is", "a good", "person.", collapse=", and ")
[1] "Prof. Singh is a good person., and Mr. Venkat is a good person., and Dr. Jha is a good
person."
```

```
x = paste("Ex", 1:5, sep="_")
x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
x[1]
[1] "Ex_1"
x[2]
[1] "Ex_2"
x[3]
[1] "Ex_3"
x[5]
[1] "Ex_5"
```


x is a vector of strings. If we use the parameter collapse, a single string, instead of a vector of strings, is created:

```
x = paste("Ex", 1:5, sep="_", collapse="")
x[1]
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
```

- When using a single vector, paste0 and paste have the same outcome and they work the same as.character.

  Example:

```
paste0(1:10)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"
paste(1:10)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"
```

  When we use more than one vectors to paste0 then they concatenate in a vectorized way.

  Example:

```
paste0(1:10, c("st", "nd", "rd", rep("th", 7)))
[1] "1st" "2nd" "3rd" "4th" "5th" "6th" "7th" "8th" "9th" "10th"
```

  Observe the role of blank space

```
paste(1:10, c("st", "nd", "rd", rep("th", 7)))
```

```
[1] "1 st" "2 nd" "3 rd" "4 th" "5 th" "6 th" "7 th" "8 th" "9 th" "10 th"
```

- Command strsplit, split the elements of a character vector. "Split" can be a single character, or a character string: Syntax: strsplit(x, split, fixed = FALSE, ...)
  With a command strsplit, we can split a string in pieces.

```
x = "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
x
[1] "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
strsplit(x, split="!")
[[1]]
[1] "The&" "syntax&" "of&"
[4] "paste&" "is" "&available"
[7] " &inthe online-help"

dates = c("2020-07-24", "2021-08-25", "2022-09-26", "2023-10-27")
```

Split the dates

```
datesplt = strsplit(dates, "-")
datesplt
[[1]]
[1] "2020" "07" "24"
[[2]]
[1] "2021" "08" "25"
[[3]]
[1] "2022" "09" "26"
[[4]]
[1] "2023" "10" "27"
```


Create matrix of dates and outcome is that the elements are character

```
datemat = matrix(unlist(datesplt), nrow = 4, ncol=3, byrow=TRUE)
datemat
[,1] [,2] [,3]
[1,] "2020" "07" "24"
[2,] "2021" "08" "25"
[3,] "2022" "09" "26"
[4,] "2023" "10" "27"
```

Create matrix of dates and outcome is that the elements are numbers

```
datematrix = matrix(as.numeric(unlist(datesplt)), nrow = 4, ncol=3, byrow=TRUE)
datematrix
[,1] [,2] [,3]
[1,] 2020 7 24
[2,] 2021 8 25
[3,] 2022 9 26
[4,] 2023 10 27
```

R – Software

- nchar takes a character vector as an argument and returns a vector whose elements contain the sizes of the corresponding elements of x. nzchar is a fast way to find out if elements of a character vector are non-empty strings. Syntax: nchar(x, type = "chars", allowNA = FALSE, keepNA = NA) Syntax: nzchar(x, keepNA = FALSE)
  Example:
  x = "R course 24.07.2022"
  y = "Number of participants: 25"
  nchar(x) #Count the Number of Characters in x
  [1] 19
  nchar(y) #Count the Number of Characters in y
  [1] 26

  The nzchar(x) function takes the character vector x as a parameter. Its output is either TRUE or FALSE.

  nzchar(x)
  [1] TRUE
  nzchar(y)
  [1] TRUE

  y = c("Apple", "", "Cake")
  y
  [1] "Apple" "" "Cake"
  nzchar(y)
  [1] TRUE FALSE TRUE

  z1 = c(1.1, 2.22, 3.333)
  nchar(z1)
  [1] 3 4 5
- tolower(x) and toupper(x) Functions: tolower(x) and toupper(x) convert upper-case characters in a character vector to lower-case, or vice versa. Non-alphabetic characters are left unchanged.
  Example:
  x = "R course will start from 24.07.2022"
  toupper(x)
  [1] "R COURSE WILL START FROM 24.07.2022"

  z = "INDIAN INSTITUTE OF TECHNOLOGY"
  tolower(z)
  [1] "indian institute of technology"

DISPLAY AND FORMATTING:

- Use sub and gsub to replace the first instance of a substring:
  Syntax: sub(old, new, string)

  The sub function finds the first instance of the old substring within string and replaces it with the new substring.

gsub does the same thing, but it replaces all instances of the substring (a global replace), not just the first.
Syntax: gsub(old, new, string)
Example:

```
y = "Mr. Singh is the smart one. Mr. Singh is funny, too."

gsub("Mr. Singh","Professor Jha", y)
[1] "Professor Jha is the smart one. Professor Jha is funny, too."

sub("Mr. Singh","Professor Jha", y)
[1] "Professor Jha is the smart one. Mr. Singh is funny, too."
```

- Some functions (e.g., grep, grepl, etc.) are used for searching for matches and functions whereas sub and gsub are used for performing replacement.

The grep function is used for searching the matches. Syntax: grep(pattern, x, ignore.case = FALSE) search for matches to argument pattern within each element of a character vector x. It returns an integer vector of the indices of the elements of x that yielded a match.

grep(pattern, x, value = TRUE) returns a character vector containing the selected elements of x.
Example:

```
str = c("R Course", "exercises", "include
examples of r language", "in R software.")

grep("R", str, ignore.case=F, value=T)
[1] "R Course" "in R software."

grep("R", str, ignore.case=T, value=T)
[1] "R Course" "exercises"
[3] "include examples of r language" "in R software."

grep("R", str, ignore.case=T, value=F)
[1] 1 2 3 4

grep("R", str, ignore.case=F, value=F)
[1] 1 4
```


grepl(pattern, x) returns a character vector containing the selected elements of x and the outcome is in terms of TRUE and FALSE. Indicating if the matching is available or not.
Example:

```
str = c("R Course", "exercises", "include examples of R language")
grepl("R", str)
[1] TRUE FALSE TRUE
```

DATA FRAMES:

R – Software

- The commands c, cbind, vector and matrix functions combine data. Another option is the data frame. In a data frame, we can combine variables of equal length, with each row in the data frame containing observations on the same unit. Hence, it is similar to the matrix or cbind functions. Advantage is that one can make changes to the data without affecting the original data.
  Example:
  An example data frame painters is available in the library. MASS
  library(MASS)
  painters
  Here, the names of the painters serve as row identifications, i.e., every row is assigned to the name of the corresponding painter.
  However, these names are not variables of the data set. Here a
  subset of these names:
  rownames(painters)
  [1] "Da Udine" "Da Vinci" "Del Piombo"
  [4] "Del Sarto" "Fr. Penni" "Guilio Romano"
  [7] "Michelangelo" "Perino del Vaga" "Perugino"
  [10] "Raphael" "F. Zucarro" "Fr. Salviata"
  [13] "Parmigiano" "Primaticcio" "T. Zucarro"
  [16] "Volterra" "Barocci" "Cortona"
  [19] "Josepin" "L. Jordaens" "Testa"
  [22] "Vanius" "Bassano" "Bellini"
  [25] "Giorgione" "Murillo" "Palma Giovane"
  [28] "Palma Vecchio" "Pordenone" "Tintoretto"
  [31] "Titian" "Veronese" "Albani"
  [34] "Caravaggio" "Corregio" "Domenichino"
  [37] "Guercino" "Lanfranco" "The Carraci"
  [40] "Durer" "Holbein" "Pourbus"
  [43] "Van Leyden" "Diepenbeck" "J. Jordaens"
  [46] "Otho Venius" "Rembrandt" "Rubens"
  [49] "Teniers" "Van Dyck" "Bourdon"
- The data set contains four numerical variables (Composition, Drawing, Colour and Expression), as well as one factor variable (School).
  is.numeric(painters$School)
  [1] FALSE
  Notice how we extract a variable (column) from data set.
  is.numeric(painters$Drawing)
  [1] TRUE
  is.factor(painters$School)
  [1] TRUE
  is.factor(painters$Drawing)
  [1] FALSE
  colnames(painters)
  [1] "Composition" "Drawing" "Colour" "Expression" "School"
- Using the summary function, we can get a quick overview of descriptive measures for each variable:
  summary(painters)

R – Software

DATAFRAMES: CREATION AND OPERATIONS

- Test weather a data is a data frame or not
  Example:
  ```
  is.data.frame(painters)
  [1] TRUE
  ```
- Use data.frame to create a data frame
  Example:
  ```
  X=1:16
  Y=matrix(x,nrow=4,ncol=4)
  Z=letters[1:16]
  X
   [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
  Y
      [,1] [,2] [,3] [,4]
  [1,]    1    5    9   13
  [2,]    2    6   10   14
  [3,]    3    7   11   15
  [4,]    4    8   12   16
  Z
   [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
  datafr=data.frame(X,Y,Z)
  datafr
     X X1 X2 X3 X4 Z
  1   1  1  5  9 13 a
  2   2  2  6 10 14 b
  3   3  3  7 11 15 c
  4   4  4  8 12 16 d
  5   5  1  5  9 13 e
  6   6  2  6 10 14 f
  7   7  3  7 11 15 g
  8   8  4  8 12 16 h
  9   9  1  5  9 13 i
  10 10  2  6 10 14 j
  11 11  3  7 11 15 k
  12 12  4  8 12 16 l
  13 13  1  5  9 13 m
  14 14  2  6 10 14 n
  15 15  3  7 11 15 o
  16 16  4  8 12 16 p
  ```
- To display information about structure of data frame us str()
  Example:
  ```
  str(painters)
  'data.frame':   54 obs. of  5 variables:
   $ Composition: int  10 15 8 12 0 15 8 15 4 17 ...
   $ Drawing    : int  8 16 13 16 15 16 17 16 12 18 ...
   $ Colour     : int  16 4 16 9 8 4 4 7 10 12 ...
  ```

$ Expression : int 3 14 7 8 0 14 8 6 4 18 ...
$ School : Factor w/ 8 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 …

- Extract a variable from data frame using $.
  Example:
  painters$School
  [1] A A A A A A A A A A B B B B B B C C C C C C D D D D D D D D D D E E E E E E E E F F F F G G G G G G G H H H H
  Levels: A B C D E F G H
- Accessing data in dataframes use [row, column]
  Example:
  paitners[“Da Udine”, “Composition”]
  [1] 10
- Plot a graphs of data
  Example:
  barplot(table(painters$School))
  pie(table(painters$School))

DATA FRAMES: SOME MORE OPERATIONS

- Attaching a data frame With a command attach() over the data frame, the variables can be referenced directly by name. It can address the names of a data frame directly, without the prefix dollar sign operator, e.g. painters$.
  Example:
  attach(painters)
  Composition,
  Drawing,
  Colour,
  Expression,
  School
  summary(School) # Character variable See we used School instead of painters$School
  A B C D E F G H
  10 6 6 10 7 4 7 4
  The command detach() recovers the default setting and then we have to use painters$ again.
  detach(painters)
- Subsets of a data frame can be obtained with subset() or with the second equivalent command:
  subset(painters, School=='F') # == means logical equal sign

	Composition	Drawing	Colour	Expression	School
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F

  Similar outcome can be also obtained from
  painters[painters[["School"]] == "F",]

	Composition	Drawing	Colour	Expression	School
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F

```
Van Leyden    8        6        6              4          F
subset(painters, School='F', select=c(-3, -5))
          Composition Drawing Colour
Durer             8       10      10
Holbein           9       10      16
Pourbus           4       15       6
Van Leyden        8        6       6
```

- The command split partitions the data set by values of a specific variable. This should preferably be a factor variable.
  Example:

```
splitted = split(painters, painters$School)
splitted
$A
                Composition Drawing Colour Expression School
Da Udine             10      8     16        3      A
Da Vinci             15     16      4       14      A
Del Piombo            8     13     16        7      A
Del Sarto            12     16      9        8      A
Fr. Penni             0     15      8        0      A
Guilio Romano        15     16      4       14      A
Michelangelo          8     17      4        8      A
Perino del Vaga      15     16      7        6      A
Perugino              4     12     10        4      A
Raphael              17     18     12       18      A

$B
          Composition Drawing Colour Expression School
F. Zucarro          10     13      8        8      B
Fr. Salviata        13     15      8        8      B
Parmigiano          10     15      6        6      B
Primaticcio         15     14      7       10      B
T. Zucarro          13     14     10        9      B
Volterra            12     15      5        8      B

$C
          Composition Drawing Colour Expression School
Barocci             14     15      6       10      C
Cortona             16     14     12        6      C
Josepin             10     10      6        2      C
L. Jordaens         13     12      9        6      C
Testa               11     15      0        6      C
Vanius              15     15     12       13      C

$D
                Composition Drawing Colour Expression School
Bassano              6      8     17        0      D
Bellini              4      6     14        0      D
```

```
Giorgione          8     9    18       4     D
Murillo            6     8    15       4     D
Palma Giovane     12     9    14       6     D
Palma Vecchio      5     6    16       0     D
Pordenone          8    14    17       5     D
Tintoretto        15    14    16       4     D
Titian            12    15    18       6     D
Veronese          15    10    16       3     D

$E
          Composition Drawing Colour Expression School
Albani            14    14    10       6     E
Caravaggio         6     6    16       0     E
Corregio          13    13    15      12     E
Domenichino       15    17     9      17     E
Guercino          18    10    10       4     E
Lanfranco         14    13    10       5     E
The Carraci       15    17    13      13     E

$F
          Composition Drawing Colour Expression School
Durer             8     10    10       8     F
Holbein           9     10    16      13     F
Pourbus           4     15     6       6     F
Van Leyden        8      6     6       4     F

$G
          Composition Drawing Colour Expression School
Diepenbeck       11     10    14       6     G
J. Jordaens      10      8    16       6     G
Otho Venius      13     14    10      10     G
Rembrandt        15      6    17      12     G
Rubens           18     13    17      17     G
Teniers          15     12    13       6     G
Van Dyck         15     10    17      13     G

$H
        Composition Drawing Colour Expression School
Bourdon          10      8     8       4     H
Le Brun          16     16     8      16     H
Le Suer          15     15     4      15     H
Poussin          15     17     6      15     H
```

The objects splitted$A to splitted$H are themselves data frames:

```
is.data.frame(splitted$A)
[1] TRUE
```

DATA FRAMES: COMBINING AND MERGING

R – Software

- The command cbind horizontally merges two data frames side by side.
  Example: Create two data frames as follows:

```
df1=data.frame(state=c("UP", "MP", "AP", "JK"), popnsize=c(1000,2000,3000,4000))
df2=data.frame(state=c("UP", "MP", "AP", "JK"), samplesize=c(100,200,300,400),
surveycompleted=c("Yes", "No", "Yes", "No"))
df1
state popnsize
1 UP 1000
2 MP 2000
3 AP 3000
4 JK 4000

df2
state samplesize surveycompleted
1 UP 100 Yes
2 MP 200 No
3 AP 300 Yes
4 JK 400 No

cbind(df1,df2)
state popnsize state samplesize surveycompleted
1 UP 1000 UP 100 Yes
2 MP 2000 MP 200 No
3 AP 3000 AP 300 Yes
4    JK 4000 JK 400 No
```


- The command merge horizontally merges two data frames by common columns or row names.
  Example: Create two data frames as follows:

```
df1=data.frame(state=c("UP", "MP", "AP", "JK"), popnsize=c(1000,2000,3000,4000))

df2=data.frame(state=c("UP", "MP", "AP", "JK"), samplesize=c(100,200,300,400),
surveycompleted=c("Yes", "No", "Yes", "No"))
```

  Variable "state" is common between the two data frames and we want to merge the two data frames with respect to state.

```
df1
state popnsize
1 UP 1000
2 MP 2000
3 AP 3000
4 JK 4000

df2
state samplesize surveycompleted
1 UP 100 Yes
2 MP 200 No
3 AP 300 Yes
```

```
4 JK 400 No

merge(df1,df2,by="state")
state popnsize samplesize surveycompleted
1 AP 3000 300 Yes
2 JK 4000 400 No
3 MP 2000 200 No
4    UP 1000 100 Yes
```

- The command rbind stacks two data frames on top of each other, appending one to the other
  Example: Create two data frames as follows:

```
df11=data.frame(state=c("UP", "MP", "AP", "JK"), popnsize=c(1000,2000,3000,4000))
df22=data.frame(state=c("Bihar", "Delhi", "Punjab"), popnsize =c(100,200,300))
df11
state popnsize
1 UP 1000
2 MP 2000
3 AP 3000
4 JK 4000
df22
state popnsize
1 Bihar 100
2 Delhi 200
3 Punjab 300
```



```
rbind(df11,df22)
state popnsize
1 UP 1000
2 MP 2000
3 AP 3000
4 JK 4000
5 Bihar 100
6 Delhi 200
7 Punjab 300
```

CSV AND TABULAR DATA FILES

- One can also read or upload the file from Internet site. We can read the file containing rent index data from website:
  http://home.iitk.ac.in/~shalab/Rcourse/munichdata.asc
  as follows

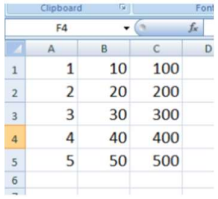
```
datamunich = read.table(file= "http://home.iitk.ac.in/~shalab/Rcourse/munichdata.asc", header=TRUE)
```

  Comma-separated values (CSV) files: First set the working directory where the CSV file is located. To read a CSV file Syntax: read.csv("filename.csv")
  Example:

```
data = read.csv("example1.csv")
data
```

R – Software

```
 X1 X10 X100
1 2 20 200
2 3 30 300
3 4 40 400
4 5 50 500
```



```
data = read.csv("datafile.csv", header=FALSE)
data
  V1 V2 V3
1 1 10 100
2 2 20 200
3 3 30 300
4 4 40 400
5 5 50 500

names(data) = c("Column1","Column2","Column3")

data
  Column1 Column2 Column3
1 1 10 100
2 2 20 200
3 3 30 300
4 4 40 400
5 5 50 500
```


Comma-separated values (CSV) files We can set the delimiter with sep.

If it is tab delimited, use sep="\t".

```
data = read.csv("datafile.csv", sep ="\t")
```

If it is space-delimited, use sep=" ".

```
data = read.csv("datafile.csv", sep =" ")
```

- Reading Tabular Data Files Tabular data files are text files with a simple format:
  • Each line contains one record.
  • Within each record, fields (items) are separated by a one- character delimiter, such as a space, tab, colon, or comma.
  • Each record contains the same number of fields.
  We want to read a text file that contains a table of data. read.table function is used and it returns a data frame.

```
read.table("FileName")
```

Example:

```
data = read.table("example3.txt", sep =" ")
data
  V1 V2 V3
1 1 10 100
2 2 20 200
3 3 30 300
```

4 4 40 400
5 5 50 500

EXCEL AND OTHER DATA FILES

- Spreadsheet (Excel) file data The readxl package has the function read_excel() for reading Excel files. This will read the first sheet of an Excel spreadsheet. To read Excel files, we first need to install the package

```
install.packages("readxl")
library(readxl)
```

Spreadsheet (Excel) file data

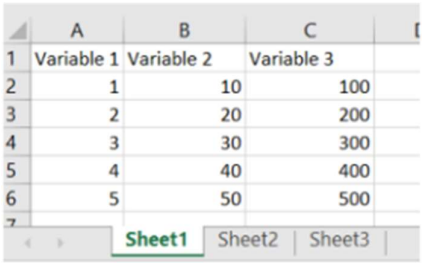
```
read_excel("datafile.xlsx")
read_excel("datafile.xls")

# Specify sheet either by position or by name
read_excel(datasets, sheet_number)
read_excel(datasets, "sheet_name")
```

To extract variable, write

```
object_name$Variable_name
```

Example:

```
dataspexcel = read_excel("spexcel.xlsx", sheet=1)
dataspexcel
# A tibble: 5 x 3
`Variable 1` `Variable 2` `Variable 3`
<dbl> <dbl> <dbl>
1 1 10 100
2 2 20 200
3 3 30 300
4 4 40 400
5 5 50 500
```

	A	B	C
1	Variable 1	Variable 2	Variable 3
2	1	10	100
3	2	20	200
4	3	30	300
5	4	40	400
6	5	50	500

Sheet1 Sheet2 Sheet3

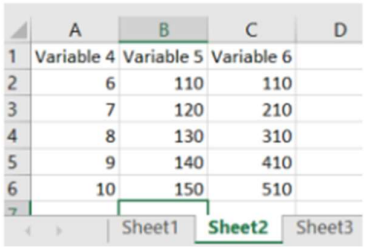
Excel data
Example: #Observe that this is similar to that of data.frame

```
dataspexcel$`Variable 1`
[1] 1 2 3 4 5
dataspexcel$`Variable 2`
[1] 10 20 30 40 50
mean(dataspexcel$`Variable 1`)
[1] 3
```

Excel data
Example:

```
dataspexcel2 = read_excel("spexcel.xlsx", sheet=2)
```

```
dataspexcel2
# A tibble: 5 x 3
`Variable 4` `Variable 5` `Variable 6`
<dbl> <dbl> <dbl>
1 6 110 110
2 7 120 210
3 8 130 310
4 9 140 410
5 10 150 510
```



	A	B	C	D
1	Variable 4	Variable 5	Variable 6	
2	6	110	110	
3	7	120	210	
4	8	130	310	
5	9	140	410	
6	10	150	510	

Sheet1 | Sheet2 | Sheet3

```
dataspexcel2$`Variable 4`
[1] 6 7 8 9 10
dataspexcel2$`Variable 5`
[1] 110 120 130 140 150
dataspexcel2$`Variable 6`
[1] 110 210 310 410 510
mean(dataspexcel2$`Variable 6`)
[1] 310
```

- Spreadsheet (Excel) file data

```
# Limit the number of data rows read
read_excel(datasets, n_max = 3)
# Read from an Excel range using A1 or R1C1 notation
read_excel(datasets, range = "C1:E7")
read_excel(datasets, range = "R1C2:R2C5")
```


R1C1 notation : Row-Column notation

R2C3 refers to the cell at the second row and third column

- # Limit the number of data rows read

```
dataspexcel4 = read_excel("spexcel.xlsx", n_max=3)
dataspexcel4
# A tibble: 3 x 3
`Variable 1` `Variable 2` `Variable 3`
<dbl> <dbl> <dbl>
1 1 10 100
2 2 20 200
3   3 30 300
```

- SPSS data file For reading SPSS data files, use foreign package and function read.spss()

```
#To read SPSS files, we first need to install the package
install.packages(" foreign ")
library(foreign)
data = read.spss("datafile.sav")
```

- HTML data file For reading HTML data files, use XML package and function readHTMLTable

```
#To read HTML data files, we first need to install the package
install.packages("XML")
library(XML)
data = readHTMLTable("filename")
```

R – Software

- Other data files The foreign package also includes functions to load from other formats, including:
  • read.octave("<Path to file>"): Octave and MATLAB
  • read.systat("<Path to file>"): SYSTAT
  • read.xport("<Path to file>"): SAS XPORT
  • read.dta("<Path to file>"): Stata

WRITING DATA INTO FILES

- The write function can write the data (usually a matrix) x are written to file file. If x is a two dimensional matrix you need to transpose it to get the columns in file the same as those in the internal representation.
  write(x, file = "data", ncolumns , append = FALSE, sep = " ")
  Arguments
  x the data to be written out, usually an atomic vector.
  file a connection, or a character string naming the file to write to. If "", print to the standard output connection.
  ncolumns the number of columns to write the data in.
  append if TRUE the data x are appended to the connection.
  sep a string used to separate columns. Using sep = "\t" gives tab delimited output; default is " ".
  Example:
  x=c(1:100)
  x
  [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
  [19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
  [37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
  [55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
  [73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
  [91] 91 92 93 94 95 96 97 98 99 100
  write(x, file="shalabh") #Written into a text file .txt

- The write.csv function can write tabular data to an ASCII file in CSV format. Each row of data creates one line in the file, with data items separated by commas (,):
  write.csv(x, file = "", append = FALSE)
  write.csv(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"), fileEncoding = "")
- The write.table prints its required argument x . write.table(x, file = "", append = FALSE)
  write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"), fileEncoding = "")
  quote a logical value (TRUE or FALSE) or a numeric vector. If TRUE, any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of columns to quote. If FALSE, nothing is quoted. sep the field separator string. Values within each row of x are separated by this string. eol the character(s) to print at the end of each line (row). na the string to use for missing values in the data.

R – Software

STATISTICAL FUNCTIONS

- First hand tools which gives first hand information.
  • Central tendency of data
  • Variation in data
  • Structure and shape of data tendency
  • Relationship study
  Graphical as well as analytical tools are used.
- Suppose there are 10 persons coded into two categories as male (M) and female (F).
  M, F, M, F, M, M, M, F, M, M.
  Use a1 and a2 to refer to male and female categories. There are 7 male and 3 female persons, denoted as n1 = 7 and n2 = 3 The number of observations in a particular category is called the absolute frequency.
- The relative frequencies of a1 and a2 are
  F1 = n1/(n1+n2) = 7/10 = 0.7 =70%
  F2 = n2/(n1+n2) = 3/10 = 0.3 =30%
  This gives us information about the proportions of male and female persons.
  table(variable) creates the absolute frequency of the variable of the data file. Enter data as x
  table(x) # absolute frequencies
  table(x)/length(x) # relative frequencies
  Example:
  Example: Code the 10 persons by using, say 1 for male (M) and 2 for female (F).
  M, F, M, F, M, M, M, F, M, M
  1, 2, 1, 2, 1, 1, 1, 2, 1, 1

  gender <- c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)
  gender
  [1] 1 2 1 2 1 1 1 2 1 1
  table(gender) # Absolute frequencies
  gender
  1 2
  7 3
  table(gender)/length(gender) #Relative freq.
  gender
  1 2
  0.7 0.3
- Such values divides the total frequency given data into required number of partitions.
  Quartile: Divides the data into 4 equal parts.
  Decile: Divides the data into 10 equal parts.
  Percentile: Divides the data into 100 equal parts.
- quantile function computes quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1.
  quantile(x, ...)
  quantile(x, probs = seq(0, 1, 0.25),...)
  Arguments
  x numeric vector whose sample quantiles are wanted,
  probs numeric vector of probabilities with values in [0, 1].

R – Software

Example:

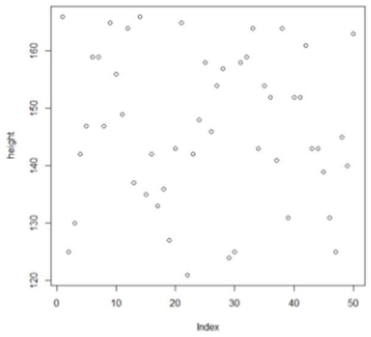
```
marks = c(68, 82, 63, 86, 34, 96, 41, 89, 29, 51, 75, 77, 56, 59, 42)

quantile(marks)
0% 25% 50% 75% 100%
29.0 46.5 63.0 79.5 96.0
```

Default values

```
quantile(marks, probs=c(0,0.25,0.5,0.75,1))
0% 25% 50% 75% 100%
29.0 46.5 63.0 79.5 96.0
```

SCATTER AND BAR PLOTS

- Graphical tools- various type of plots In R, Such graphics can be easily created and saved in various formats.
  - o Bar plot
  - o Pie chart
  - o Box plot
  - o Grouped box plot
  - o Scatter plot
  - o Histogram
  - o Various 3 dimensional plots
  - o ...
- Plot command for one variable:




```
x: Data vector
plot(x)
Example:
height = c(166,125,130,142,147,159,159,147,
165,156,149,164,137,166,135,142,133,136,127,143,
165,121,142,148,158,146,154,157,124,125,158,159,
164,143,154,152,141,164,131,152,152,161,143,143,
139,131,125,145,140,163)
plot(height)
plot(height, col = "red") #dots will appear in red color
```

- Bar Plots: Visualize the relative or absolute frequencies of observed values of a variable.

```
barplot(x, width = 1, space = NULL,...)
#Bar plot with absolute frequencies
barplot(table(x)) # Absolute frequencies
#Bar plot with relative frequencies
barplot(table(x)/length(x))  # you saw bar graph previously
barplot(table(direction), col=c("red", "green", "blue") )
barplot(table(direction), col=c("red", "green", "blue"), main="Directions of food delivery" )
#main gives the graph a name
barplot(table(direction), col=c("red", "green", "blue"), main="Directions of food delivery",
legend.text=c("dir1", "dir2", "dir3") ) # defines the plot
barplot(table(direction), col=c("red", "green", "blue"), main="Directions of food delivery",
legend.text=c("dir1", "dir2", "dir3"), sub="Three directions" ) # another name at bottom
```

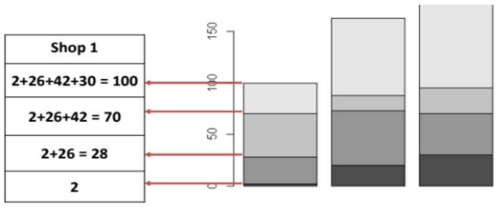
R – Software

barplot(table(direction), col=c("red", "green", "blue"), main="Directions of food delivery", legend.text=c("dir1", "dir2", "dir3"), sub="Three directions", xlab="Food Delivery Directions", ylab="Number of Deliveries") #labling x and y axis

SUB – DEVIDED BAR PLOTS AND PIE DIAGRAM

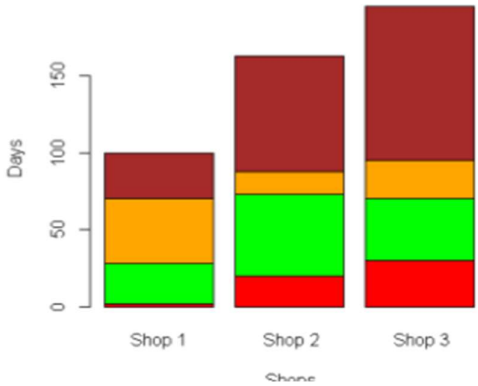
- Subdivided or component bar diagram divides the total magnitude of variables into various parts.
  Example:
  The data on the number of
  customers visiting 3 shops during
  10-11 AM on 4 consecutive days
  is as follows:

  cust = matrix(nrow=4, ncol=3, data =c(2,20, 30,26,53,40,42,15,25,30,75,100), byrow = T)
  cust
  [,1] [,2] [,3]
  [1,] 2 20 30
  [2,] 26 53 40
  [3,] 42 15 25
  [4,] 30 75 100

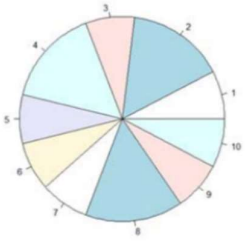


  Usage barplot(variable in matrix format) will create a subdivided or component bar diagram with columns of matrix as bars. Sections inside bars indicate the values in cumulative form.
  barplot(cust)
  barplot(cust, names.arg=c("Shop 1", " Shop 2", "Shop 3"), xlab = " Shops", ylab = "Days", col=c("red","green","orange","brown"))

No. of customers	Shop 1	Shop 2	Shop 3
Day 1	2	20	30
Day 2	26	53	40
Day 3	42	15	25
Day 4	30	75	100

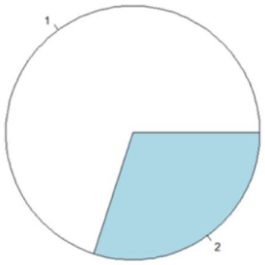


- Pie Diagram:
  Pie charts visualize the absolute and relative frequencies. A pie chart is a circle partitioned into segments where each of the segments represents a category. The size of each segment depends upon the relative frequency and is determined by the angle (frequency X 3600).
  pie(x, labels = names(x), ...)
  Example:
  Code the 10 persons by using, say 1 for male (M) and 2 for female (F).
  M, F, M, F, M, M, M, F, M, M
  1, 2, 1, 2, 1, 1, 1, 2, 1, 1
  gender = c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)

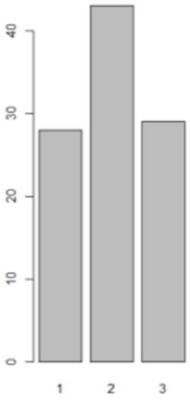


R – Software

gender
[1] 1 2 1 2 1 1 1 2 1 1
pie(gender)
pie(table(gender))

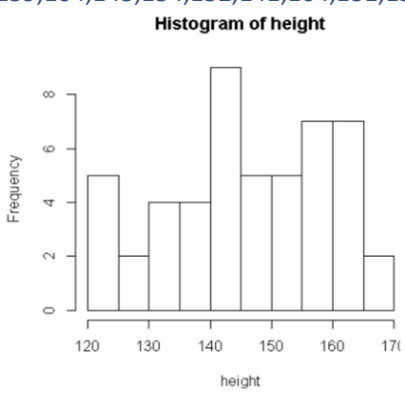


- Use command par() to put multiple graphs in a single plot. Adjust the graphical parameters with the help of function.
  par(mfrow=c(p,q)) # set the plotting area into a p*q array
  Example:
  direction=c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2,1,3,3,3,2,2,2,2,1,2,2,1,1,1,3,2,2,1,2,3,
  2,2,1,2,3,3,2,1,2,2,3,1,1,2,1,2,3,2,3,2,2,3,1,2,3,3,3,2,1,1,1,2,1,1,2,1,2,3,3,1,2,3,3,2,1,2,3,2,1,3
  ,2,2,2,2,3,2,2)
  par(mfrow=c(1,2)) # set the plotting area into a 1*2 array
  barplot(table(direction))
  pie(table(direction))



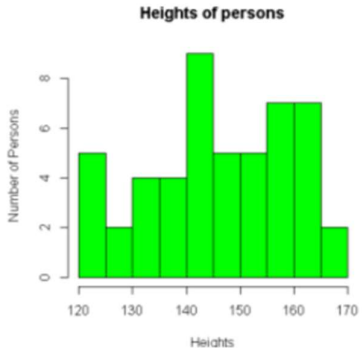
HISTOGRAMS:

- Histogram is based on the idea to categorize the data into different groups and plot the bars for each category with height. Data is continuous. The area of the bars (= height X width) is proportional to the frequency (or relative frequency). So the widths of the bars need not necessarily to be the same hist(x) # show absolute frequencies
  hist(x, freq=F) # show relative frequencies
  hist(x, main, col, xlab, xlim, ylim)
  x : Vector containing numeric values used in histogram.
  main : Title of the chart.
  col : Set colour of the bars.
  xlab : Description of x-axis.
  xlim : Specifies the range of values on x-axis.
  ylim : Specifies the range of values on y-axis.
  Example:
  Height of 50 persons in centimetres are recorded as follow
  166,125,130,142,147,159,159,147,165,156,149,164,137,166,135,142,133,136,127,143,165,
  121,142,148,158,146,154,157,124,125,158,159,164,143,154,152,141,164,131,152,152,161,
  143,143,139,131,125,145,140,163
  height=c(166,125,130,142,147,159,159,147,165,156,149,164,137,166,135,142,133,136,127,
  143,165,121,142,148,158,146,154,157,124,125,158,159,164,143,154,152,141,164,131,152,
  152,161,143,143,139,131,125,145,140,163)
  hist(height)



R – Software

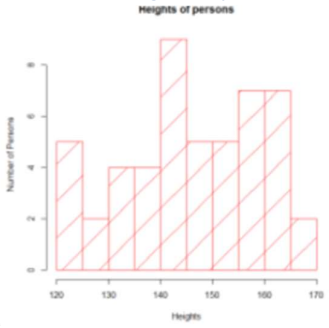
hist(height, main = "Heights of persons", col = "green", xlab = "Heights", ylab = "Number of Persons")



density : Density of shading lines, in lines per inch. Non- positive values of density also inhibit the drawing of shading lines.

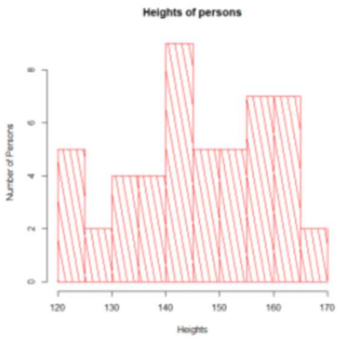
hist(height, main = "Heights of persons", col = "red", xlab = "Heights", ylab = "Number of Persons", density = 2)

Higher the density closer and more the lines(Shaded)



angle : the slope of shading lines, given as an angle in degrees (counter-clockwise).

hist(height, main = "Heights of persons", col = "red", xlab = "Heights", ylab = "Number of Persons", density = 8, angle=100)



BIVARIANT AND THREE-DIMENSIONAL SCATTER PLOTS

- In Association of two variables The observation on both the variables are related to each other. Question's:
  - How to know the variables are related?
  - How to know the degree of relationship between the two variables?

  Solution:

  - Graphical procedures – Two dimensional plots, three dimensional plots etc.

R – Software

- Quantitative procedures – Correlation coefficients, contingency tables, Chi-square statistic, linear regression, nonlinear regression etc.

- How to judge or graphically summarize the association of two variables?
  X, Y: Two variables
  n pairs of observations are available as (x1, y1), (x2, y2), ..., (xn, yn)
- Scatter Plots: Plot the paired observations in a single graph, called as scatter plot. Scatter plot reveals the nature and trend of possible relationship. Relationships : Linear or nonlinear.
  We will study about the direction and degree of linear relationships.
  Two aspects – graphical and quantitative
- Bivariate Plots:
  - Scatter plots:
    Plot command:
    x, y: Two data vectors
    plot(x, y)
    plot(x, y, type)

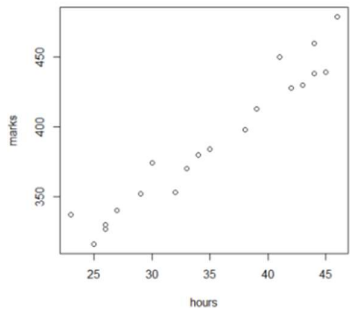
type	
"p" for points	"l" for lines
"b" for both	"c" for the lines part alone of "b"
"o" for both 'overplotted'	"s" for stair steps.
"h" for 'histogram' like (or 'high-density') vertical lines	

Example:
Data on marks obtained by 20 students out of 500 marks and the number of hours they studied per week are recorded as follows: We know from experience that marks obtained by students increase as the number of hours increase.

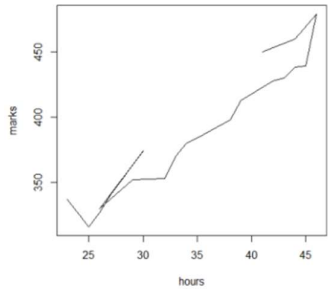
Marks	337	316	327	340	374	330	352	353	370	380
Number of hours per week	23	25	26	27	30	26	29	32	33	34

Marks	384	398	413	428	430	438	439	479	460	450
Number of hours per week	35	38	39	42	43	44	45	46	44	41

```
marks=c(337,316,327,340,374,330,352,353,370,380,384,398,413,428,430,438,439,479,460,450)
hours = c(23,25,26,27,30,26,29,32,33,34,35,38,39,42,43,44,45,46,44,41)
plot(hours, marks)
```

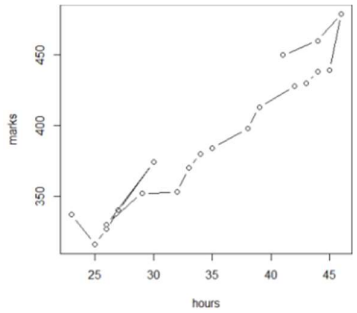


plot(hours, marks, "l") #"l" for lines

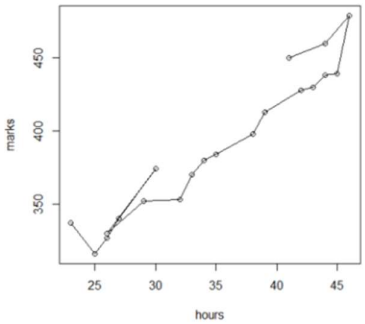


R – Software

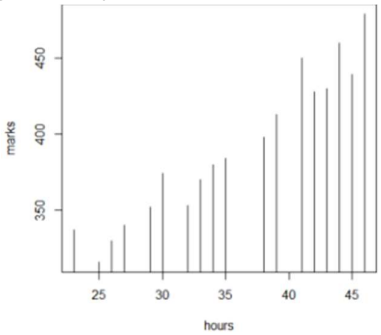
plot(hours, marks, "b") #“b" for both – line and point



plot(hours, marks, "o") #“o" for both ‘overplotted’

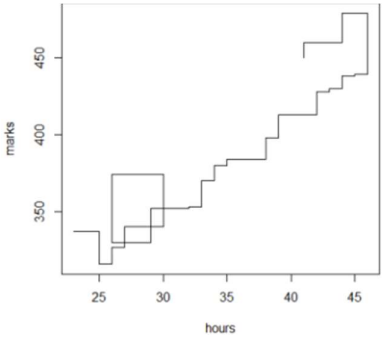


plot(hours, marks, "h") #“h" for ‘histogram’ like (or ‘high-density’) vertical lines

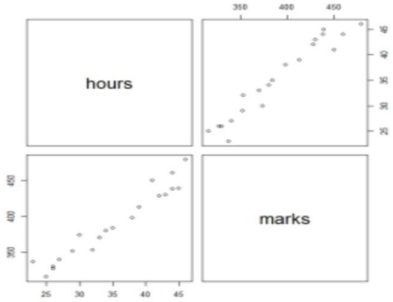


plot(hours, marks, "s") #“s" for stair steps.

#plot(hours, marks, xlab="Number of weekly hours", ylab="Marks obtained", main="Marks obtained versus Number of hours per week")

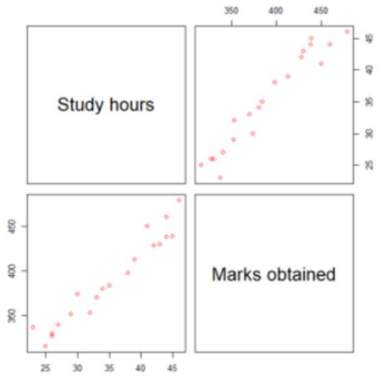


- Matrix Scatter Plot: The command pairs() allows the simple creation of a matrix of scatter plots.
  Example:
  pairs(cbind(hours, marks)) #Continution to above code

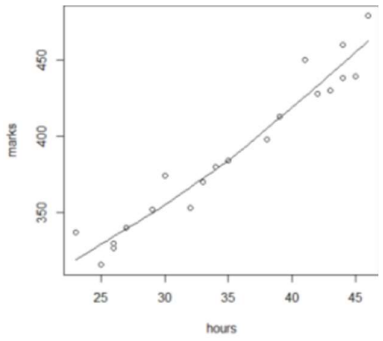


R – Software

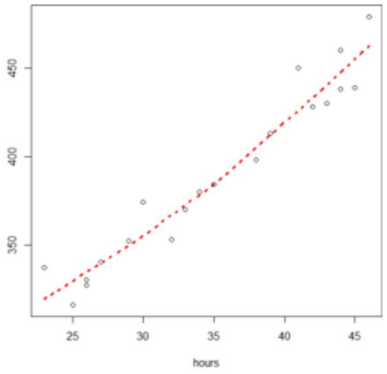
pairs(cbind(hours, marks), labels=c("Study hours", "Marks obtained"), col="red")



- Scatter Plot with Smooth Curve: scatter.smooth is based on the concept of LOESS which is a locally weighted scatterplot smoothing method. LOESS is used for local polynomial regression fitting. Fit a polynomial surface determined by one or more numerical predictors, using local fitting.
  scatter.smooth(x, y = NULL, span = 2/3, degree = 1, family = c("symmetric","gaussian"), xlab = NULL, ylab = NULL, ylim = range(y, pred$y, na.rm = TRUE),...)
  Example:
  scatter.smooth(hours, marks) #continution to above code



scatter.smooth(hours, marks, lpars = list(col = "red", lwd = 3, lty = 3))



- Three Dimensional Scatter Plot: scatterplot3d(x,y,z)
  Plots a three dimensional (3D) point cloud of the data in x,y and z Need a package scatterplot3d
  install.packages("scatterplot3d")
  library(scatterplot3d)

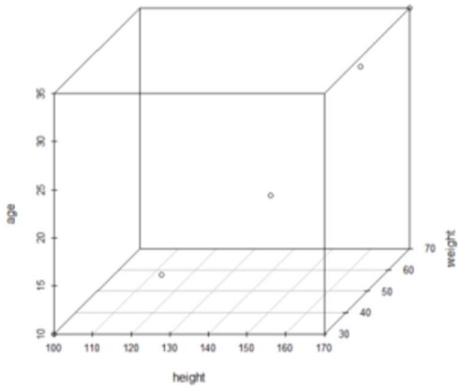
R – Software

Example:

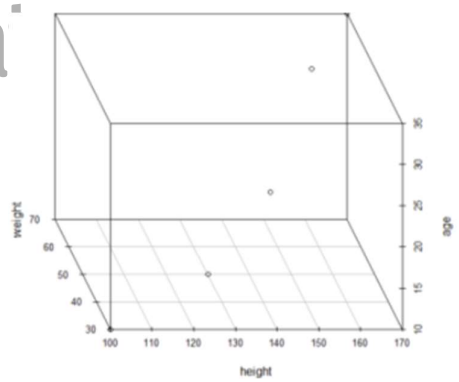
The data on height (in cms.), weight (in kg.) and age (in years) of 5persons are recorded as follows. We would like to create a 3 dimensional plot for this data.

Person No.	Height (Cms.)	Weight (Kg.)	Age (Years)
1	100	30	10
2	125	35	15
3	145	50	20
4	160	65	30
5	170	70	35

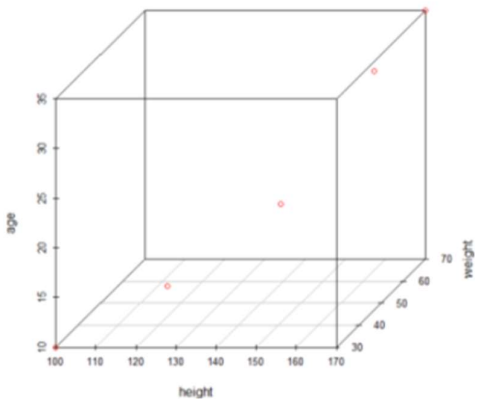
```
library(scatterplot3d)
height = c(100, 125, 145, 160, 170)
weight = c(30, 35, 50, 65, 70)
age = c(10, 15, 20, 30, 35)
scatterplot3d(height, weight, age)
```




```
scatterplot3d(height, weight, age, angle = 120)
```

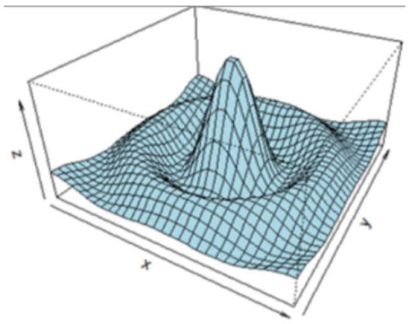




```
scatterplot3d(height, weight, age, color="red")
```

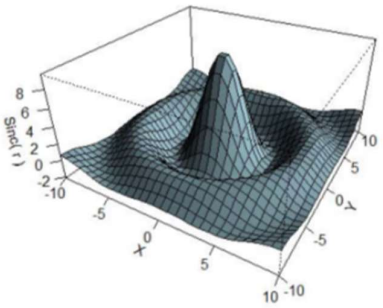


R – Software

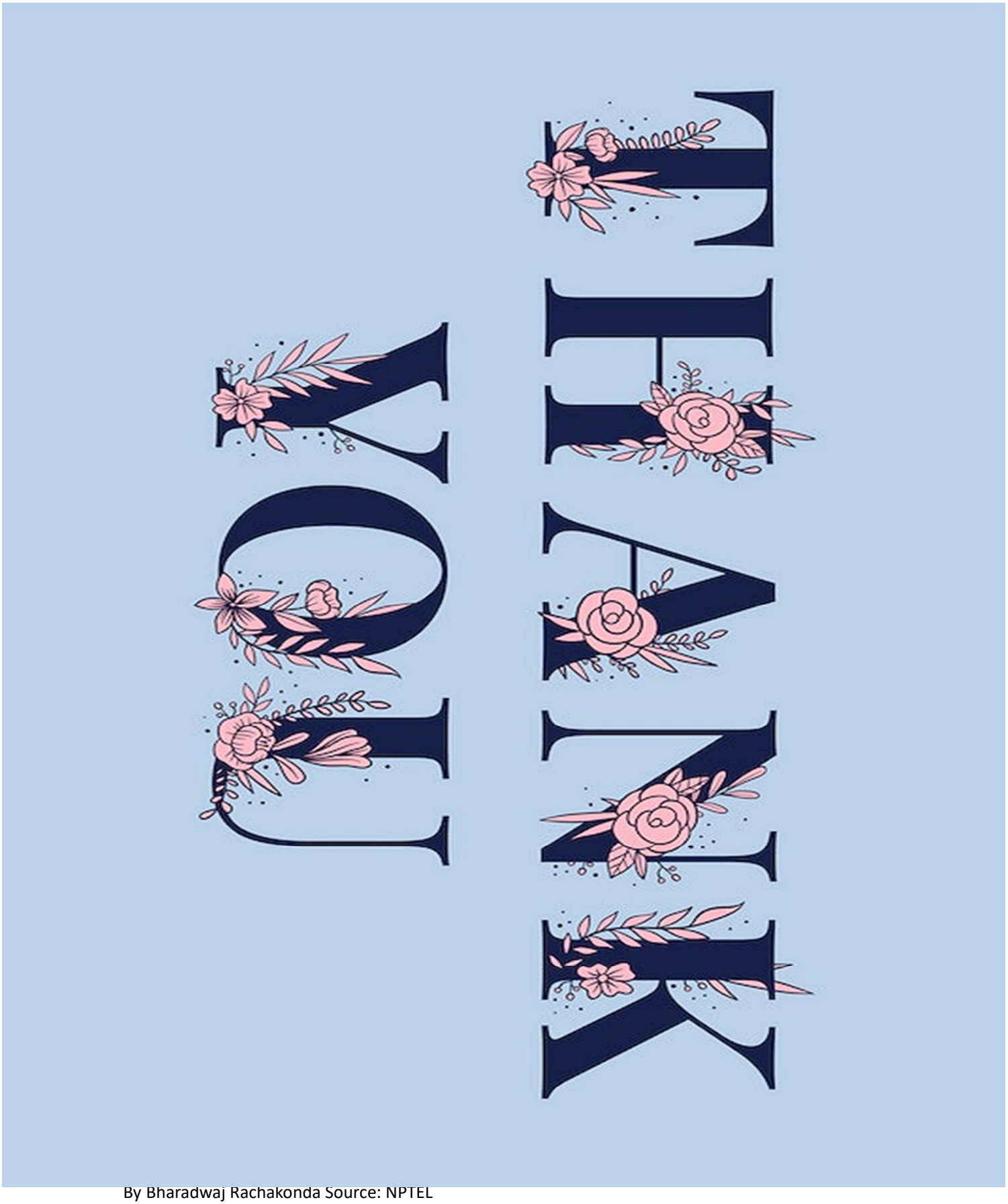
- More Functions:
  - contour() for contour lines
  - dotchart() for dot charts (replacement for bar charts)
  - image() pictures with colors as third dimension
  - mosaicplot() mosaic plot for (multidimensional) diagrams of categorical variables (contingency tables)
  - persp() perspective surfaces over the x–y plane
- persp() perspective surfaces over the x–y plane




```
x = seq(-10, 10, length= 30)
y=x
f =function(x,y){r = sqrt(x^2+y^2);10*sin(r)/r}
z = outer(x, y, f)
z[is.na(z)] = 1
op = par(bg = "white")
persp(x,y,z, theta=30, phi=30, expand=0.5, col= "lightblue")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue", ltheta = 120, shade = 0.75,
ticktype = "detailed", xlab = "X", ylab = "Y", zlab = "Sinc( r )")
```



R – Software



By Bharadwaj Rachakonda Source: NPTEL
Pg No: