Section-2 Programming in 

Surya Lamichhane

16 January 2024

Contents

1. Reading and writing data in R

Working directory

The working directory in R is the folder where you are working currently. It is the place where you are storing your current work or project unless specified any other location using the program.

```
getwd() # Find current directory
setwd("directory path") # can be used to to change the working directory.
```

A typical file path is of type "/Users/Surya/Desktop/Spring2023_STA4101L"

The path of the file can be found as

- Windows
  - In Windows from properties after right clicking the file, or
  - try drag and drop to R console.
  - File path can be defined in the form `"C:\\Users\\Surya\\Desktop\\STA4101L"`, because `"\"` has special meaning - it is used as escape
- In macOS
  - from get-info after right clicking the file, or
  - try drag and drop to R console

Exercise 1 (working directory)

- Find the current directory
- Create a directory STAT4101Lab and set that directory as current directory

Getting help in R

- help() function and ? help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.
- Examples
- Help documentation for linear regression function *lm*

```
help(lm) # or ?lm
```

Exercise 2 (Help in R)

a. Find the documentation for reading text data (hint: help for read.table)
b. Find the documentation for writing text data (hint: help for write.table)

Reading Data in R

R is capable of reading data from most formats, including files developed in other statistical programs such as using Excel (in CSV, XLSX, or TXT format), SAS, Stata, SPSS.

- R can read and load the data into memory.
- R also has two its native data formats: Rdata and Rds.

Here we only try three simple formats: .csv, .txt, .Rdat

```
# read csv file
my_data = read.csv("file_path/file name.csv",  header = TRUE/FALSE)
```

```
# read txt file
my_data = read.table("file_path/file_name.txt",  sep = " ", header = TRUE/FALSE)
```

```
# read Rdata
load("file_path/file_name.Rdata")
```

- We assign header = TRUE (default), that means the data has column names, if data has no column names specify header = FALSE.
- sep = '..', is specified based on how the text data-values are separated. We define sep = " " (default), if the columns are separated by 'white space', other options are sep = ',' (comma separated), sep = '\t' (tab separated)

Examples

```
cars_data = read.csv("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/Assi
str(cars_data)  # Lets find the structure of the dataset
```

```
## 'data.frame':    428 obs. of  15 variables:
##  $ Make       : chr  "Subaru" "Toyota" "Suzuki" "Dodge" ...
##  $ Model      : chr  " Forester X" " Camry Solara SE V6 2dr" " Aerio LX 4dr" " Dakota Club Cab" ...
##  $ Type       : chr  "Wagon" "Sedan" "Sedan" "Truck" ...
```

```
##  $ Origin     : chr  "" "Asia" "Asia" "USA" ...
##  $ DriveTrain : chr  "All" "Front" "Front" "Rear" ...
##  $ MSRP       : int  21445 21965 14500 20300 NA 29795 22450 25395 42735 52800 ...
##  $ Invoice    : int  19646 19819 14317 18670 15922 27536 20595 23043 37422 49104 ...
##  $ EngineSize : num  2.5 3.3 2.3 3.7 2.3 3.5 3.4 4.3 5.3 5 ...
##  $ Cylinders  : int  4 6 4 6 4 6 6 NA 8 8 ...
##  $ Horsepower : int  165 225 155 210 160 280 175 190 295 302 ...
##  $ MPG_City   : int  21 20 25 16 25 18 20 15 14 17 ...
##  $ MPG_Highway: int  28 29 31 22 31 26 29 19 18 22 ...
##  $ Weight     : int  3090 3417 2676 3829 2762 3416 3118 4083 4947 3585 ...
##  $ Wheelbase  : int  99 107 98 131 104 112 107 123 130 107 ...
##  $ Length     : int  175 193 171 219 179 182 186 208 219 183 ...
```

```
class_marks = read.csv("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/As
head(class_marks,2)  # Let's print few observations
```

```
##   Enrol.No. Maths Science English
## 1      A101    16      15      12
## 2      A102    16      17      11
```

```
load("/Users/suryalamichhane/Desktop/STAT4101L_all_files/Stat4101L-Rfiles/DataSets/Assisted Practice da
dim(class_info)  # Let's find number of rows and columns of the dataset
```

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

Exercise 2 (Reading data)

- Download above three datasets and read them on R
- use str( ), head( ) and tail( ) functions

Writing Data in R

We may need to save our computed R output in our local folders

1. The simplest way we can save as R.data using

```
save(class_marks, file = "/Users/suryalamichhane/Desktop/student_class_marks.Rdata")
```

Not all other program can read Rdata, so we may need to save the output data in other format

2. Saving data into .csv extension file

```
write.csv(output, file = "file_path/file_name.csv", col.names = TRUE)
```

3. Saving data into .txt extension file

```
write.table(output, file = "file_path/file_name.txt", sep = " ",row.names = FALSE, col.names = TRUE)
# rnow.names/col.names = TRUE or FALSE is determined based on output data
# sep = " ", if data are sperated by white sapces
```

Exercise 3 (Writing data)

- Try to write the a dataset in above three extension format

2. Programming in R

Conditional Statements

- IF
- IF ELSE
- IF , ELSE IF, ELSE

```
if (Test condition ) {
   # statement to execute if condition is true
   # otherwise do not enter inside of this if block
}
```

If statement Examples

```
X = 7
Y = NA
if (X > 5 ) {
   Y = X + 2
}
print(Y)
```

```
## [1] 9
```

```
X = 3
Y = NA
if (X > 5 ) {
   Y = X + 2
}
print(Y)
```

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

If ELSE statement When we have two choices:

```
if (Test condition 1 ) {
   # statement to execute if condition is true
} else{
  # when first statement is false this statement executed
}
```

Examples

1. (If else statement)

```r
X = 7
if (X %%2 == 0 ) {
   print("X is even")
} else{
  print("X is odd")
}
```

```
## [1] "X is odd"
```

2. (If else statement)

```r
X = 24
if (X %%2 == 0 ) {
   print("X is even")
} else{
  print("X is odd")
}
```

```
## [1] "X is even"
```

ifelse is equivalent to if else

```r
X = 7
ifelse (X %%2 == 0, "X is even", "X is odd" )
```

```
## [1] "X is odd"
```

```r
# if the condition is true execute first, otherwise second.
```

IF, ELSE IF statement When we have multiple choices

```r
if (test condition 1 ) {
   # statement  executed if test condition 1 is true but not others
} else if (test condition 2 ){
   # statement  executed if test condition 2  is true but not others
} else if (test condition 3 ){
    # statement  executed if test condition 2  is true but not others
  }else{
    # statement  executed if all of above test conditions are false
  }
```

Example

```r
# A = 90 - 100, B = 80 - 89, C =  70 - 79, D = 60 - 69, F = 0 - 59
grade = 75
if (grade < 60) {
   print("letter grade is F")
} else if ( grade < 70 ){
   print("letter grade is D")
} else if (grade < 80 ){
```

```
    print("letter grade is C")
} else if ( grade < 90 ){
   print("letter grade is B")
} else{
   print("letter grade is A")
  }
```

```
## [1] "letter grade is C"
```

Exercise 4 (Conditions in R) Write a program to check whether a given year is a leap year or not If a year is divisible by 4, 100 and 400, it's a leap year. If a year is divisible by 4 and 100 but not divisible by 400, it's not a leap year. If a year is divisible by 4 but not divisible by 100, it's a leap year.

Counters in progamming language Counter is a numeric variable that tracks the number of times a process is repeated in our particular program. A counter is initialized with a numeric value, and it increases by 1 when the process continue. So, we usually use counter inside the loop to count the repeated process that satisfied the given condition.

```
counter = 1 # initialize the counter
loop starts{
  expression
  counter = counter + 1
}
```

Loops

If we want a set of operations to be repeated several times, we use usually use loops. When you create a loop, R will execute the instructions inside the loop for a specified number of times or until a specified condition is met. There are three main types of loop in R:

- While
- For
- Repeat

```
while(test condition) {
  expression
   # statement  executed until the condition is true
}
```

While loop Examples

1. Sum to 100

```
k = 0
sumk = k
while(k < 100) {
  k = k + 1 # statement  executed until the condition is true
```

```
  sumk = sumk + k
  # print(k)
}
print(sumk)
```

```
## [1] 5050
```

Loop Control Statements

- break statement

A break statement is used inside a loop (repeat, for, while) to stop the iterations and flow the control outside of the loop.

Example (break statement)

```
# Break Statement Example
a <- 1
while (a < 10)
{
    if(a==5)
      break
     #stops if condition is true
    a = a + 1
}
print(a)
```

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

- Next Statement

The next statement is used to skip the current iteration in the loop and move to the next iteration without exiting from the loop itself.

1. Example of next statement

```
# Next Statement Example
x <- 0
while(x < 5)
{
  x <- x + 1;
    if (x == 3)
        next; #skips the line if the condition is true
    print(x);
}
```

```
## [1] 1
## [1] 2
## [1] 4
## [1] 5
```

2. Example of next statement (sum of even numbers)

```r
k = 0
sumk = k
while(k < 100) {
  k = k + 1 # statement  executed until the condition is true
  if (k %% 2 != 0) next  #skips the line if condition is true
   sumk = sumk + k

}
print(sumk)
```

```
## [1] 2550
```

For loop The mostly commonly used loop structure to compute repeated set of operations is for loop.

```r
for (val in squence)
{
   expression  # continue until the last value of sequence reached
}
```

```r
x = 1:10
y = rep(NA, 10)

for (i in 1:10){
  y[i] = x[i]^2 + 5
}
y
```

```
##  [1]   6   9  14  21  30  41  54  69  86 105
```

for loop Examples

```r
x = 1:10
y = c()

for (i in 1:10){
  temp = x[i]^2 + 5 # temporary variable
  y = c(y, temp)
}
y
```

```
##  [1]   6   9  14  21  30  41  54  69  86 105
```

break statement inside for loop

- mtcars data

```r
# load mtcars data from base library
for (i in 1: length(rownames(mtcars))){
  if (i == 5)
    break
  print(rownames(mtcars)[i])
}
```

```
## [1] "Mazda RX4"
## [1] "Mazda RX4 Wag"
## [1] "Datsun 710"
## [1] "Hornet 4 Drive"
```

```
# load mtcars data from base library
for (cars in rownames(mtcars)){
  if (cars == "Cadillac Fleetwood"){
    break
  }
  print(cars)

}
```

```
## [1] "Mazda RX4"
## [1] "Mazda RX4 Wag"
## [1] "Datsun 710"
## [1] "Hornet 4 Drive"
## [1] "Hornet Sportabout"
## [1] "Valiant"
## [1] "Duster 360"
## [1] "Merc 240D"
## [1] "Merc 230"
## [1] "Merc 280"
## [1] "Merc 280C"
## [1] "Merc 450SE"
## [1] "Merc 450SL"
## [1] "Merc 450SLC"
```

next statement inside for loop

```
# load mtcars data from base library
for (i in 1: nrow(mtcars)){
  car = rownames(mtcars[i,])
  if (car != "Cadillac Fleetwood"){
    next # skip the expression
  }

  print(paste(car, ": is", i, 'th car from the list', sep = " "))

}
```

```
## [1] "Cadillac Fleetwood : is 15 th car from the list"
```

- Count even numbers in a sequence

```
# vector of numbers
num = c(2, 3, 12, 14, 5, 19, 23, 64)

# variable to store the count of even numbers
count = 0
sum = 0
```

```r
# for loop to count even numbers
for (i in num) {
  # check if i is even
  if (i %% 2 == 0) {
    sum = sum + i
    count = count + 1
  }
}

print(count)
```

```
## [1] 4
```

Repeat loop Repeat loop in R is used to iterate over a block of code again and again until it reaches to the **break statement**.

```r
# Repeat loop syntax
repeat {
   expression
   if(condition) {
      break # same expression run again again until if(condition) does not hold.
   }
}
```

Repeat loop Example

1. Sum up to n

```r
k = 1
sumk = k
n = 10
# Repeat loop syntax
repeat {
  k = k + 1
   if(k > n) {
      break # same expression run again again until if condition does not hold.
   }
  sumk = sumk + k
  print(paste("Sum up to", k, 'is', sumk, sep = " "))
}
```

```
## [1] "Sum up to 2 is 3"
## [1] "Sum up to 3 is 6"
## [1] "Sum up to 4 is 10"
## [1] "Sum up to 5 is 15"
## [1] "Sum up to 6 is 21"
## [1] "Sum up to 7 is 28"
## [1] "Sum up to 8 is 36"
## [1] "Sum up to 9 is 45"
## [1] "Sum up to 10 is 55"
```

2. Sum of odd numbers

```
k = 1
sumk = 1
n = 10
# Repeat loop syntax
repeat {
  k = k + 1
   if(k %% 2 == 0) {
      next # skip if even
   }
  if (k > n ){
    break
  }
  sumk = sumk + k
  print(paste("Sum up to", k, 'is', sumk, sep = "_"))
}
print(sumk)
```

3. How many times you need to draw a card from a pack of 52 playing cards with replacement to get a king first time.

```
kings = rep("King", 4)
others = rep("noKing", 48)
packOfcards = sample(c(kings, others)) # shuffle all the cards
# sample(packOfcards, 1) # a randomly selected card
print(paste("one randomly selected card is:", sample(packOfcards, 1)))
```

```
## [1] "one randomly selected card is: noKing"
```

- How many draws you required to get first king

```
k = 0
# Repeat loop syntax
repeat {
  k = k + 1
  kth_card = sample(packOfcards, 1)
  print(paste(k, "th card is", kth_card, sep = " " ))

  if(kth_card == "King") {
    break
    }
  }
```

Exercise 5 (Loops in R)

- Write a program to create a vector of the first 10 prime numbers
- An integer n >1 is prime if it is only divisible by 1 and itself
- i.e, n is prime if n %% c(2: n-1) !=0,

```
prime = 2
n = 2
while(length(prime) < 10){
```

```
  n = n + 1
  check_remainder = n%% (2:(n-1))
  if (any(check_remainder == 0 ))
    {
    next # not a prime number, We don't want it
  }else {
    prime = c(prime, n)
  }
}
prime
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29
```

```
prime_numbers = 2
n = 1
repeat{
  n = n + 1
  check_remainder = n%% (2:(n-1))
  if (any(check_remainder == 0 ))
  {
    next
    }
  else {prime_numbers = c(prime_numbers, n)
  }
  if (length(prime_numbers) == 10)
    {
    break
  }
}
prime
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29
```

Build your own function in R

Type of functions

Inbuilt functions in R: We have already used several inbuilt functions in R such as mean, list, matrix, sum. Function always takes input and computes the results as output. We can get help to know about inbuilt functions in R to find the more details about those functions. Type **help(function_name)** or **?function_name()** to get the help in R.

```
# get help about mean function
help(mean)
?mean()
```

```
sum(...) - returns sum of a numerical objects, sum without NA values: sum(X, na.rm = TRUE)
colSums(...) - returns column sum of a matrix
rowSums(...) - returns row sum of a matrix
cumsum(...) - retruns cumulative sum of the numerical varaible
rm(...) - removes a varaible from R enviroment
```

```
help(...) - gives details of the function
na.omit(...) - removes  NA values
mean(...) - returns mean of a numerical variable, mean without NA values: mean(X, na.rm = TRUE)
log(...) - computes the natural logarithm
exp(...) - computes exponential
head(X, n)  - returns top n observations of matrix or vector X
tail(X, n)  - returns buttom n observations of matrix or vector X
typeof(...) - returns the type of data
str(...) - returns structure of the data
summary(...) - returns six point summary of a numerical data object Minimum, 1st Quartile, Median, Mean
sort() - sorts a vector in ascending or descending (decreasing=TRUE) order
round(...) - rounds number with decimial, for specific k digits use round(x, digits = k)
floor(...) - returns the greatest integer smaller than x
ceiling(...) - returns the smallest integer greater than x
min(), max() - returns the minimum / maximum / mean / median value of a numeric vector, correspondingly
range() - returns the minimum and maximum values of a numeric vector
abs() - returns the absulute value of a number
print() - displays an R object on the console
nchar() - returns the number of characters in a character object
exists() - returns TRUE or FALSE depending on whether or not a variable is defined in the R
           environment
na.omit() - remove na values
```

User defined functions: The user defined functions are useful in R when same block of code can be used for similar type of tasks or project, so that we can call that function to perform the same operations again just like inbuilt function. Functions are created in R by using the command function(), and the general structure of the function is as follows:

```
# general function structure in R
function_name = function(arg1, arg2, ... ){
  expression
  return(output)
}
```

- Arguments are passed as inputs to the function that may be necessary to accomplish the task.
- A return statement is optional in R, but if present, it returns the control to the main program along with the results.
- Any objects defined inside the functions are local (i.e. they are not stored in the R environment), and they can be any data type.

Function examples

1. Sum of two integeres

```
sum_of_two_numbers <- function(x, y){
  z = x+y
  K = x - y
  return(z)
}
sum_of_two_numbers(12, 5)
```

```
## [1] 17
```

2. Check fuel economy

```
 isMPGEconomy <- function(mpg, threshold = 25){
    #This function returns "NO" if the input value is less than 25. Otherwise it returns "YES".
    if(mpg < threshold ){
        return("NO") # return NO if the mpg is less than 25

    }else{
        return("YES") # otherwise return YES
    }
}

isMPGEconomy(22)
```

```
## [1] "NO"
```

```
isMPGEconomy(33)
```

```
## [1] "YES"
```

Check on mtcars data

Find economy cars from *mtcars* data

```
FindEconomyCar <- function(data){
    mpg <- data$mpg
    for (i in 1: nrow(data)){
      if (isMPGEconomy(mpg[i]) == "YES"){
        print(rownames(data[i, ]))
      }

    }

}

FindEconomyCar(mtcars)
```

```
## [1] "Fiat 128"
## [1] "Honda Civic"
## [1] "Toyota Corolla"
## [1] "Fiat X1-9"
## [1] "Porsche 914-2"
## [1] "Lotus Europa"
```

3. Formula for computing sample variance

$$\text{Sample variance} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2.$$

```r
compute_sample_var = function(X_vec){
  n = length(X_vec)
  X_bar = mean(X_vec)
  dx = X_vec - X_bar
  samp_var = 1/(n-1) * sum(dx^2)
  return(samp_var)
}

X = runif(20, 50, 90)
compute_sample_var(X)
```

```
## [1] 140.4296
```

Exercise 6 (Functions in R) Mathematical form of sigmoid function

$$S(x) = \frac{1}{1 + e^{-x}}$$

- Sigmoid function or logistic function is useful in modeling probability in Statistics
- It takes any real number as its input and outputs as a positive value between 0 and 1.
- Write a program to compute the sigmoid function in R

Packages in R

Packages are collections of functions and data along with their documentation that can be shared among all kind of R-users.

- CRAN repository has more than 18000 packages for different applications of data science, machine learning, and deep learning.
- The list of installed packages in R can be retrieved by using the function:

```r
# check installed pacakges
installed.packages()
```

Installing and loading new packages

```r
# If a given package is already installed, it will be checked for an update.
install.packages('dplyr')
```

- To access functions of any package in R, the package must be loaded and attached in the current R session.
- To load any installed packages, use *library(packagename)* function. This will allow us to use any function from the loaded package.

R Packages for Data Analysis

```
dplyr: used in data manipulation
ggplot2: creates elegant data visualizations
caret: Contains classification and regression techniques
tidyr: various tools for data cleaning
forecast: has time series forecasting functions
xts:  offers a number of great tools for data manipulation and aggregation of timeseries data.
lubridate: makes easier to work with dates and times
arules: Contains association rules mining functions
readr: used to read tabular data
stringr: has functions for various string operations
```

String functions in R

```
nchar(x): Returns the count of characters
toupper(x) and tolower(x): Changes the case of a string to uppercase or lowercase
substr (x, start, stop): Returns a part of a string
paste(...): Concatenates multiple string objects
strsplit(x, split): Splits a string or each string in a vector at given character
sub(pattern, replacement, x): Replaces a first occurrence of a substring with another
format(): Formats number and strings to a specific style
grep(pattern, x): Searches for a pattern in a vector of strings and returns the matching indices.
```

Basic Statistical functions in R

```
mean(x ): Returns the average of the object
median(x): Returns the median value of the given object
sd(x): Returns the standard deviation of the object
var(x): Returns the variance of the object
quantile(x): Returns the quartile values of x
range(x): Returns the maximum and minimum value in the object x
scale(x): Returns the standardized scores of the object x
summary(x): Shows the statistical summary of the object x and given mean, median,minimum value, maximum
```

Miscelleneous functions

```
seq(): Generates a sequence with the given start and end
rep(): Repeats a vector for a given no. of times
cut():Divides a continuous variable in factor with given levels
sample(): Shuffle the sequence and return desired samples
```

Apply Family Functions

The apply() family function comes with R base package. The family has functions which are generally used to manipulate the data to avoid explicit use of loop constructs. They inputs list, matrix or array and apply a named function with one or several optional arguments. Most common function of this family are:

- apply() : apply(X , MARGIN, FUN)
- lapply(): lapply(X, FUN, ...)
- sapply(): sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
- tapply(): tapply(X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)

apply( ) function It takes a matrix or an array as input and returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

```
# apply syntax
syntax: apply(X , MARGIN, FUN)
X     : an array, including a matrix
MARGIN: a vector giving the subscripts which the function will be applied over
FUN   : the function to be applied
...   : optional arguments to FUN.
}
```

Examples

1. Find the average test score of the each of the students form *class_marks* dataset

```
# remove enrol.nu which is not a numerical value
# find row average, i.e. MARGIN = 1, and FUN = mean
class_marks_only = class_marks[, -1]
Students_average = apply(class_marks_only, MARGIN = 1, FUN = mean)
print(Students_average)
```

```
##  [1] 14.33333 14.66667 15.66667 13.33333 13.00000 15.33333 14.33333 15.66667
##  [9] 14.00000 17.33333 14.00000 16.00000 18.00000 12.66667 15.33333
```

2. Find the class average in each of the courses form *class_marks* dataset

```
# remove enrol.nu which is not a numerical value
# find column average, i.e. MARGIN = 2, and FUN = mean
class_marks_only = class_marks[, -1]
course_average = apply(class_marks_only, MARGIN = 2, FUN = mean)
print(course_average)
```

```
##    Maths  Science  English
## 14.86667 15.93333 13.93333
```

lapply( ) function lapply() applies any given function to each element of a vector or a list and returns a list. It is best for working with a list where apply() cannot be used

Examples

1. Find the range of the class mark using lapply

```
range_class_mark = lapply(class_marks, range)
range_class_mark
```

```
## $Enrol.No.
## [1] "A101" "A119"
##
## $Maths
## [1] 11 19
##
```

```
## $Science
## [1] 11 19
##
## $English
## [1] 10 19
```

2. Working with list:

```
class_grade_list = list(letter_grades = c("F","D", "C", "B", "A"), class1 = c(65, 70, 90), class2 = c(95

lapply(class_grade_list, range)
```

```
## $letter_grades
## [1] "A" "F"
##
## $class1
## [1] 65 90
##
## $class2
## [1] 55 95
##
## $class3
## [1] 42 94
```

sapply() Function sapply() simplifies the output of lapply() by coercing it into a simpler data structure.

- If the output of lapply() is a list where each element has a length of 1, then it is converted to a vector in sapply().
- If the output of lapply() is a list where each element has the same length, then it is converted to a matrix in sapply().
- Else, the output is a list.

1. Find the range of class_mark using sapply()

```
range_class_mark = sapply(class_grade_list, range)
range_class_mark
```

```
##      letter_grades class1 class2 class3
## [1,] "A"           "65"   "55"   "42"
## [2,] "F"           "90"   "95"   "94"
```

2. Find the minimum grade of class_grade_list using sapply()

```
class_grade_list = list(letter_grades = c("F","D", "C", "B", "A"), class1 = c(65, 70, 90), class2 = c(95

sapply(class_grade_list, min)
```

```
## letter_grades        class1        class2        class3
##           "A"          "65"          "55"          "42"
```

tapply() Function tapply() applies a function to a vector for each group in the factor vector, it works just like table function.

```
 # get mean of scores by gender
df <- data.frame(score = c(95, 78, 90, 13, 74, 83, 81, 34, 20, 15),
                 gender = c('F', 'M', 'F', 'F', 'F', 'M', 'M', 'F', 'M', 'M'))
df
```

```
##    score gender
## 1     95      F
## 2     78      M
## 3     90      F
## 4     13      F
## 5     74      F
## 6     83      M
## 7     81      M
## 8     34      F
## 9     20      M
## 10    15      M
```

```
tapply(df$score, df$gender, mean)
```

```
##    F    M
## 61.2 55.4
```

Other apply() functions are

- vapply()
- mapply()
- rapply()

Miscellaneous function

which function which() function in R Programming Language is used to return the position or index of the specified values where the given conditions are true.

```
X = c(1, 3, 7, 2, 8, 12, 6, 4, 3)
#find the positions where X has even integer
even_index = which(X %% 2 == 0)
#cat("positions where X has even values:", even_index)
X[even_index]
```

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

Order function order() function in R Programming Language is used to return the position or index of the entries of a vector into ascending or descending order.

```
order(..., na.last = TRUE, decreasing = FALSE,
      method = c("auto", "shell", "radix"))
```

```r
X = c(1, 3, 7, 2, 8, 12, 6, 4, 3)
order(X) # returns the indices of the values in ascending order
```

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

```r
X[order(X)]
```

```
## [1]  1  2  3  3  4  6  7  8 12
```