ANNAMACHARYA
INSTITUTE OF TECHNOLOGY AND SCIENCES
(AUTONOMOUS)

Approved by AICTE, New Delhi & Permanent Affiliation to JNTUA, Anantapur.
Three B. Tech Programmes (CSE , ECE & CE) are accredited by NBA, New Delhi,Accredited by NAAC with 'A' Grade , Bangalore.
A-grade awarded by AP Knowledge Mission. Recognized under sections 2(f) & 12(B) of UGC Act 1956.
Venkatapuram Village, Renigunta Mandal, Tirupati, Andhra Pradesh-517520.

Department of Computer Science and Engineering

Academic Year 2023-24

IV. B.Tech I Semster

Data Analytics (Common to CSE, CIC, AIDS) (20APE0514/20APE3613/20APE3016)

Prepared By

Mr. S. Prathap., M.Tech(Ph.D).
Mr. J Chandra Babu., M.Tech(Ph.D).
Mr. P. Bhanuprakash., M.Tech(Ph.D).
Assistant Professor
Department of CSE, AITS

DATA ANALYTICS

Unit 1: An overview of R, Vectors, factors, univariate time series, Data frames, matrices, Functions, operators, loops, Graphics, Revealing views of the data, Data summary, Statistical analysis questions, aims, and strategies; Statistical models, Distributions: models for the random component, Simulation of random numbers and random samples, Model assumptions

Text Books:
1. Data Analysis and Graphics Using R – an Example-Based Approach, John Maindonald, W. John Braun, Third Edition, 2010

An overview of R

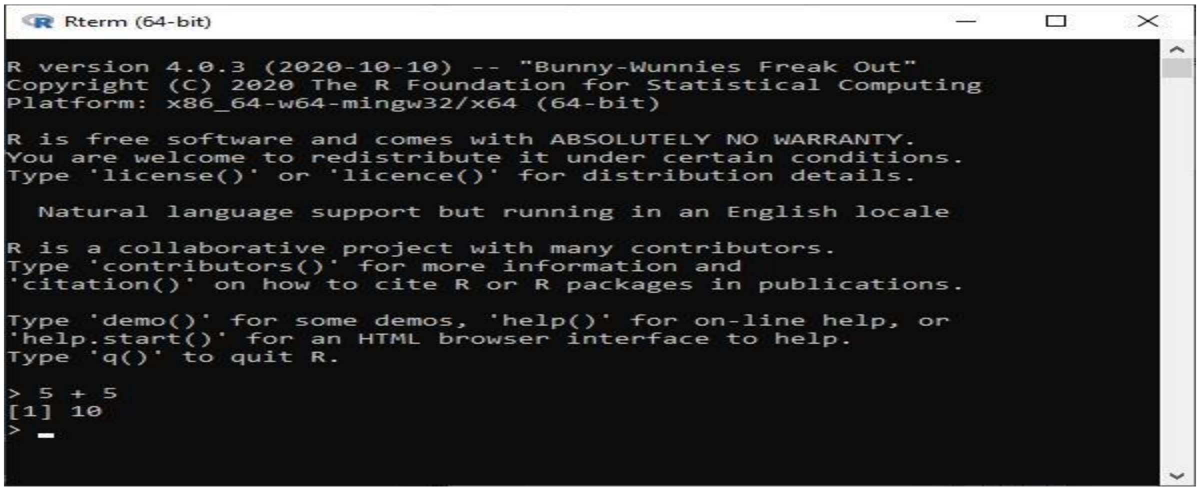
R is a programming language.
R is often used for statistical computing and graphical presentation to analyze and visualize data.

Why Use R?

- It is a great resource for data analysis, data visualization, data science and machine learning
- It provides many statistical techniques (such as statistical tests, classification, clustering and data reduction)
- It is easy to draw graphs in R, like pie charts, histograms, box plot, scatter plot, etc
- It works on different platforms (Windows, Mac, Linux)
- It is open-source and free
- It has a large community support
- It has many packages (libraries of functions) that can be used to solve different problems

How to Install R

To install R, go to https://cloud.r-project.org/ and download the latest version of R for Windows, Mac or Linux.
When you have downloaded and installed R, you can run R on your computer.
The screenshot below shows how it may look like when you run R on a Windows PC:



```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 5 + 5
[1] 10
> _
```

If you type 5 + 5, and press enter, you will see that R outputs 10.

The command line prompt (>) is an invitation to type commands or expressions. Once the command or expression is complete, and the Enter key is pressed, R evaluates and prints the result in the console window. This allows the use of R as a calculator.

For example, type 2+2 and press the Enter key. Here is what appears on the screen:

```
> 2+2
[1] 4
>
```

The first element is labeled [1] even when, as here, there is just one element! The final > prompt indicates that R is ready for another command.

```
> 2*3*4*5 # * denotes 'multiply'
[1] 120
> sqrt(10) # the square root of 10
[1] 3.162278
> pi # R knows about pi
[1] 3.141593
> 2*pi*6378 # Circumference of earth at equator (km)
            # (radius at equator is 6378 km)
[1] 40074.16
```

Anything that follows a # on the command line is taken as comment and ignored by R.
A continuation prompt, by default +, appears following a carriage return when the command is not yet complete. For example, an interruption of the calculation of 3*4^2 by a carriage return could appear as

```
> 3*4^
+ 2
[1] 48
```

Multiple commands may appear on one line, with a semicolon (;) as the separator. For example,

```
> 3*4^2; (3*4)^2
[1] 48
[1] 144
```

To output text in R, use single or double quotes:
Example

```
"Hello World!"
```

To output numbers, just type the number (without quotes):
Example

```
5
10
25
```

To do simple calculations, add numbers together:

Example

```
5 + 5
```

Print
Unlike many other programming languages, you can output code in R without using a print function:
Example

```
"Hello World!"
```

However, R does have a print() function available if you want to use it. This might be useful if you are familiar with other programming languages, such as Python, which often uses the print() function to output code.

Example

```
print("Hello World!")
```

And there are times you must use the print() function to output code, for example when working with for loops (which you will learn more about in a later chapter):

Example

```
for (x in 1:10)
{
  print(x)
}
```

Conclusion: It is up to you whether you want to use the print() function to output code. However, when your code is inside an R expression (for example inside curly braces {} like in the example above), use the print() function to output the result.

Creating Variables in R

Variables are containers for storing data values.

R does not have a command for declaring a variable. A variable is created the moment you first assign a value to it. To assign a value to a variable, use the <- sign. To output (or print) the variable value, just type the variable name:

Example

```
name <- "John"
age <- 40
name   # output "John"
age    # output 40
```

From the example above, name and age are **variables**, while "John" and 40 are **values**.

In other programming language, it is common to use = as an assignment operator. In R, we can use both = and <- as assignment operators.

However, <- is preferred in most cases because the = operator can be forbidden in some context in R.

Print / Output Variables

Compared to many other programming languages, you do not have to use a function to print/output variables in R. You can just type the name of the variable:

Example

```
name <- "John Doe"

name # auto-print the value of the name variable
```

However, R does have a print() function available if you want to use it. This might be useful if you are familiar with other programming languages, such as Python, which often use a print() function to output variables.

Example

```
name <- "John Doe"
print(name) # print the value of the name variable
```

And there are times you must use the print() function to output code, for example when working with for loops (which you will learn more about in a later chapter):

Example

```
for (x in 1:10)
{
  print(x)
}
```

Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

In R, variables do not need to be declared with any particular type, and can even change type after they have been set:

Example

my_var <- 30 # my_var is type of **numeric**

my_var <- "Sally" # my_var is now of type **character** (aka string)

R has a variety of data types and object classes.

Basic Data Types

Basic data types in R can be divided into the following types:

- numeric - (10.5, 55, 787)
- integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
- complex - (9 + 3i, where "i" is the imaginary part)
- character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- logical (a.k.a. boolean) - (TRUE or FALSE)

We can use the class() function to check the data type of a variable:

Example

```
# numeric
x <- 10.5
class(x)

# integer
x <- 1000L
class(x)

# complex
x <- 9i + 3
class(x)

# character/string
x <- "R is exciting"
class(x)

# logical/boolean
```

```
x <- TRUE
class(x)
```

Numbers

There are three number types in R:

- numeric
- integer
- complex

Variables of number types are created when you assign a value to them:

Example

```
x <- 10.5   # numeric
y <- 10L    # integer
z <- 1i     # complex
```

Numeric

A numeric data type is the most common type in R, and contains any number with or without a decimal, like: 10.5, 55, 787:

Example

```
x <- 10.5
y <- 55

# Print values of x and y
x
y

# Print the class name of x and y
class(x)
class(y)
```

Integer

Integers are numeric data without decimals. This is used when you are certain that you will never create a variable that should contain decimals. To create an integer variable, you must use the letter L after the integer value:

Example

```
x <- 1000L
y <- 55L

# Print values of x and y
x
y

# Print the class name of x and y
class(x)
class(y)
```

Complex

A complex number is written with an "i" as the imaginary part:

Example

```
x <- 3+5i
y <- 5i

# Print values of x and y
x
y

# Print the class name of x and y
class(x)
class(y)
```

Type Conversion

You can convert from one type to another with the following functions:

- as.numeric()
- as.integer()
- as.complex()

Example

```
x <- 1L # integer
y <- 2 # numeric

# convert from integer to numeric:
a <- as.numeric(x)

# convert from numeric to integer:
b <- as.integer(y)

# print values of x and y
x
y

# print the class name of a and b
class(a)
class(b)
```

Simple Math

In R, you can use **operators** to perform common mathematical operations on numbers.

The + operator is used to add together two values:

Example

```
10 + 5
```

And the - operator is used for subtraction:

Example

```
10 - 5
```

Built-in Math Functions

R also has many built-in math functions that allows you to perform mathematical tasks on numbers.

For example, the min() and max() functions can be used to find the lowest or highest number in a set:

Example

max(5, 10, 15)

min(5, 10, 15)

---

sqrt()
The sqrt() function returns the square root of a number:
Example
sqrt(16)

---

abs()
The abs() function returns the absolute (positive) value of a number:
Example
abs(-4.7)

---

ceiling() and floor()
The ceiling() function rounds a number upwards to its nearest integer, and the floor() function rounds a number downwards to its nearest integer, and returns the result:
Example
ceiling(1.4)
floor(1.4)

String Literals
A character, or strings, are used for storing text. A string is surrounded by either single quotation marks, or double quotation marks:
"hello" is the same as 'hello':
Example
"hello"
'hello'

Assign a String to a Variable
Assigning a string to a variable is done with the variable followed by the <- operator and the string:
Example
str <- "Hello"
str # print the value of str

Multiline Strings
You can assign a multiline string to a variable like this:
Example
str <- "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."

str # print the value of str
However, note that R will add a "**\n**" at the end of each line break. This is called an escape character, and the **n** character indicates a **new line**.

If you want the line breaks to be inserted at the same position as in the code, use the cat() function:

Example

```
str <- "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."

cat(str)
```

String Length

There are many usesful string functions in R.

For example, to find the number of characters in a string, use the nchar() function:

Example

```
str <- "Hello World!"

nchar(str)
```

Check a String

Use the grepl() function to check if a character or a sequence of characters are present in a string:

Example

```
str <- "Hello World!"

grepl("H", str)
grepl("Hello", str)
grepl("X", str)
```

Combine Two Strings

Use the paste() function to merge/concatenate two strings:

Example

```
str1 <- "Hello"
str2 <- "World"

paste(str1, str2)
```

Escape Characters

To insert characters that are illegal in a string, you must use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

```
str <- "We are the so-called "Vikings", from the north."

str
```

Result:

```
Error: unexpected symbol in "str <- "We are the so-called "Vikings"
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
str <- "We are the so-called \"Vikings\", from the north."

str
cat(str)
```

Note that auto-printing the **str** variable will print the backslash in the output. You can use the cat() function to print it without backslash.

Other escape characters in R:

Code	**Result**
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace

Booleans (Logical Values)

In programming, you often need to know if an expression is **true** or **false**.

You can evaluate any expression in R, and get one of two answers, TRUE or FALSE.

When you compare two values, the expression is evaluated and R returns the logical answer:

Example

```
10 > 9    # TRUE because 10 is greater than 9
10 == 9   # FALSE because 10 is not equal to 9
10 < 9    # FALSE because 10 is greater than 9
```

You can also compare two variables:

Example

```
a <- 10
b <- 9

a > b
```

You can also run a condition in an if statement, which you will learn much more about in the if..else chapter.

Example

```
a <- 200
b <- 33

if (b > a) {
  print ("b is greater than a")
} else {
  print("b is not greater than a")
}
```

Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

Example

10 + 5

R divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Miscellaneous operators

R Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
^	Exponent	x ^ y
%%	Modulus (Remainder from division)	x %% y
%/%	Integer Division	x%/%y

R Assignment Operators

Assignment operators are used to assign values to variables:

Example

my_var <- 3
my_var <<- 3
3 -> my_var
3 ->> my_var
my_var # print my_var

Note: <<- is a global assigner..

It is also possible to turn the direction of the assignment operator.
x <- 3 is equal to 3 -> x

R Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y

<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

R Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description
&	Element-wise Logical AND operator. It returns TRUE if both elements are TRUE
&&	Logical AND operator - Returns TRUE if both statements are TRUE
|	Elementwise- Logical OR operator. It returns TRUE if one of the statement is TRUE
||	Logical OR operator. It returns TRUE if one of the statement is TRUE.
!	Logical NOT - returns FALSE if statement is TRUE

R Miscellaneous Operators

Miscellaneous operators are used to manipulate data:

Operator	Description	Example
:	Creates a series of numbers in a sequence	x <- 1:10
%in%	Find out if an element belongs to a vector	x %in% y
%*%	Matrix Multiplication	x <- Matrix1 %*% Matrix2

Conditions and If Statements

R supports the usual logical conditions from mathematics:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written with the if keyword, and it is used to specify a block of code to be executed if a condition is TRUE:

Example

```
a <- 33
b <- 200

if (b > a) {
  print("b is greater than a")
}
```

In this example we use two variables, a and b, which are used as a part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".
R uses curly brackets { } to define the scope in the code.

Else If
The else if keyword is R's way of saying "if the previous conditions were not true, then try this condition":
Example

```
a <- 33
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print ("a and b are equal")
}
```

In this example a is equal to b, so the first condition is not true, but the else if condition is true, so we print to screen that "a and b are equal".
You can use as many else if statements as you want in R.

If Else
The else keyword catches anything which isn't caught by the preceding conditions:
Example

```
a <- 200
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater than b")
}
```

In this example, a is greater than b, so the first condition is not true, also the else if condition is not true, so we go to the else condition and print to screen that "a is greater than b".
You can also use else without else if:
Example

```
a <- 200
b <- 33
```

```
if (b > a) {
  print("b is greater than a")
} else {
  print("b is not greater than a")
}
```

Nested If Statements

You can also have if statements inside if statements, this is called *nested* if statements.

Example

```
x <- 41

if (x > 10) {
  print("Above ten")
  if (x > 20) {
    print("and also above 20!")
  } else {
    print("but not above 20.")
  }
} else {
  print("below 10.")
}
```

AND

The & symbol (and) is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a <- 200
b <- 33
c <- 500

if (a > b & c > a){
  print("Both conditions are true")
}
```

OR

The | symbol (or) is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, or if c is greater than a:

```
a <- 200
b <- 33
c <- 500

if (a > b | a > c){
  print("At least one of the conditions is true")
}
```

Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

R has two loop commands:

- while loops
- for loops

R While Loops

With the while loop we can execute a set of statements as long as a condition is TRUE:

Example

Print i as long as i is less than 6:

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

In the example above, the loop will continue to produce numbers ranging from 1 to 5. The loop will stop at 6 because 6 < 6 is FALSE.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

Note: remember to increment i, or else the loop will continue forever.

Break

With the break statement, we can stop the loop even if the while condition is TRUE:

Example

Exit the loop if i is equal to 4.

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
  if (i == 4) {
    break
  }
}
```

The loop will stop at 3 because we have chosen to finish the loop by using the break statement when i is equal to 4 (i == 4).

With the next statement, we can skip an iteration without terminating the loop:

Example

Skip the value of 3:

```
i <- 0
while (i < 6) {
  i <- i + 1
  if (i == 3) {
    next
  }
  print(i)
}
```

When the loop passes the value 3, it will skip it and continue to loop.

Yahtzee!

If .. Else Combined with a While Loop

To demonstrate a practical example, let us say we play a game of Yahtzee!

Example

Print "Yahtzee!" If the dice number is 6:

```
dice <- 1
while (dice <= 6) {
  if (dice < 6) {
    print("No Yahtzee")
  } else {
    print("Yahtzee!")
  }
  dice <- dice + 1
}
```

If the loop passes the values ranging from 1 to 5, it prints "No Yahtzee". Whenever it passes the value 6, it prints "Yahtzee!".

For Loops

A for loop is used for iterating over a sequence:

Example

```
for (x in 1:10) {
  print(x)
}
```

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a vector, array, list, etc..

You will learn about lists and vectors, etc in a later chapter.

Example

Print every item in a list:

```
fruits <- list("apple", "banana", "cherry")

for (x in fruits) {
  print(x)
}
```

Example

Print the number of dices:

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

for (x in dice) {
  print(x)
}
```

The for loop does not require an indexing variable to set beforehand, like with while loops.

Break

With the break statement, we can stop the loop before it has looped through all the items:

Example

Stop the loop at "cherry":

```
fruits <- list("apple", "banana", "cherry")

for (x in fruits) {
  if (x == "cherry") {
    break
  }
  print(x)
}
```

The loop will stop at "cherry" because we have chosen to finish the loop by using the break statement when x is equal to "cherry" (x == "cherry").

With the next statement, we can skip an iteration without terminating the loop:

Example

Skip "banana":

```
fruits <- list("apple", "banana", "cherry")

for (x in fruits) {
  if (x == "banana") {
    next
  }
  print(x)
}
```

When the loop passes "banana", it will skip it and continue to loop.

Yahtzee!

If .. Else Combined with a For Loop

To demonstrate a practical example, let us say we play a game of Yahtzee!

Example

Print "Yahtzee!" If the dice number is 6:

```
dice <- 1:6

for(x in dice) {
  if (x == 6) {
    print(paste("The dice number is", x, "Yahtzee!"))
  } else {
    print(paste("The dice number is", x, "Not Yahtzee"))
  }
}
```

If the loop reaches the values ranging from 1 to 5, it prints "No Yahtzee" and its number. When it reaches the value 6, it prints "Yahtzee!" and its number.

Nested Loops
You can also have a loop inside of a loop:

Example
Print the adjective of each fruit in a list:

```
adj <- list("red", "big", "tasty")

fruits <- list("apple", "banana", "cherry")
  for (x in adj) {
    for (y in fruits) {
      print(paste(x, y))
  }
}
```

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result.

Creating a Function
To create a function, use the function() keyword:

Example

```
my_function <- function() { # create a function with the name my_function
  print("Hello World!")
}
```

Call a Function
To call a function, use the function name followed by parenthesis, like my_function():

Example

```
my_function <- function() {
  print("Hello World!")
}

my_function() # call the function named my_function
```

Arguments
Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
my_function <- function(fname) {
  paste(fname, "Griffin")
}

my_function("Peter")
my_function("Lois")
my_function("Stewie")
```

Parameters or Arguments?

The terms "parameter" and "argument" can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less:

Example

This function expects 2 arguments, and gets 2 arguments:

```
my_function <- function(fname, lname) {
  paste(fname, lname)
}

my_function("Peter", "Griffin")
```

If you try to call the function with 1 or 3 arguments, you will get an error:

Example

This function expects 2 arguments, and gets 1 argument:

```
my_function <- function(fname, lname) {
  paste(fname, lname)
}

my_function("Peter")
```

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without an argument, it uses the default value:

Example

```
my_function <- function(country = "Norway") {
  paste("I am from", country)
}

my_function("Sweden")
my_function("India")
my_function() # will get the default value, which is Norway
my_function("USA")
```

Return Values
To let a function return a result, use the return() function:

Example

```
my_function <- function(x) {
  return (5 * x)
}

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

The output of the code above will be:

```
[1] 15
[1] 25
[1] 45
```

Nested Functions
There are two ways to create a nested function:

Call a function within another function.
Write a function within a function.
Example
Call a function within another function:

```
Nested_function <- function(x, y) {
  a <- x + y
  return(a)
}

Nested_function(Nested_function(2,2), Nested_function(3,3))
```

Example Explained
The function tells x to add y.

The first input Nested_function(2,2) is "x" of the main function.

The second input Nested_function(3,3) is "y" of the main function.

The output is therefore (2+2) + (3+3) = 10.

Example

Write a function within a function:

```
Outer_func <- function(x) {
  Inner_func <- function(y) {
    a <- x + y
    return(a)
  }
  return (Inner_func)
}
output <- Outer_func(3) # To call the Outer_func
output(5)
```

Example Explained

You cannot directly call the function because the Inner_func has been defined (nested) inside the Outer_func.

We need to call Outer_func first in order to call Inner_func as a second step.

We need to create a new variable called output and give it a value, which is 3 here.

We then print the output with the desired value of "y", which in this case is 5.

The output is therefore 8 (3 + 5).

Recursion

R also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly, recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Example

```
tri_recursion <- function(k) {
  if (k > 0) {
    result <- k + tri_recursion(k - 1)
    print(result)
  } else {
    result = 0
    return(result)
  }
}
tri_recursion(6)
```

Global Variables

Variables that are created outside of a function are known as **global** variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function and use it inside the function:

```
txt <- "awesome"
my_function <- function() {
  paste("R is", txt)
}

my_function()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside of a function with the same name as the global variable:

```
txt <- "global variable"
my_function <- function() {
  txt = "fantastic"
  paste("R is", txt)
}

my_function()

txt # print txt
```

If you try to print txt, it will return "**global variable**" because we are printing txt outside the function.

The Global Assignment Operator

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the **global assignment** operator <<-

Example

If you use the assignment operator <<-, the variable belongs to the global scope:

```
my_function <- function() {
txt <<- "fantastic"
  paste("R is", txt)
}

my_function()

print(txt)
```

Also, use the **global** assignment operator if you want to change a global variable inside a function:

Example

To change the value of a global variable inside a function, refer to the variable by using the global assignment operator <<-:

```
txt <- "awesome"
my_function <- function() {
  txt <<- "fantastic"
  paste("R is", txt)
}

my_function()

paste("R is", txt)
```

Vectors

A vector is simply a list of items that are of the same type.

To combine the list of items to a vector, use the c() function and separate the items by a comma.

In the example below, we create a vector variable called **fruits**, that combine strings:

Example

```
# Vector of strings
fruits <- c("banana", "apple", "orange")
# Print fruits
fruits
```

In this example, we create a vector that combines numerical values:

Example

```
# Vector of numerical values
numbers <- c(1, 2, 3)

# Print numbers

numbers
```

To create a vector with numerical values in a sequence, use the : operator:

Example

```
# Vector with numerical values in a sequence
numbers <- 1:10

numbers
```

You can also create numerical values with decimals in a sequence, but note that if the last element does not belong to the sequence, it is not used:

Example

```
# Vector with numerical decimals in a sequence
numbers1 <- 1.5:6.5
numbers1

# Vector with numerical decimals in a sequence where the last element is not used
numbers2 <- 1.5:6.3
numbers2
```

Result:

```
[1] 1.5 2.5 3.5 4.5 5.5 6.5
[1] 1.5 2.5 3.5 4.5 5.5
```

In the example below, we create a vector of logical values:
Example

```
# Vector of logical values
log_values <- c(TRUE, FALSE, TRUE, FALSE)

log_values
```

Vector Length
To find out how many items a vector has, use the length() function:
Example

```
fruits <- c("banana", "apple", "orange")

length(fruits)
```

Sort a Vector
To sort items in a vector alphabetically or numerically, use the sort() function:
Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")
numbers <- c(13, 3, 5, 7, 20, 2)

sort(fruits)  # Sort a string
sort(numbers) # Sort numbers
```

Access Vectors
You can access the vector items by referring to its index number inside brackets []. The first item has index 1, the second item has index 2, and so on:
Example

```
fruits <- c("banana", "apple", "orange")

# Access the first item (banana)
fruits[1]
```

You can also access multiple elements by referring to different index positions with the c() function:
Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Access the first and third item (banana and orange)
fruits[c(1, 3)]
```

You can also use negative index numbers to access all items except the ones specified:
Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Access all items except for the first item
fruits[c(-1)]
```

Change an Item

To change the value of a specific item, refer to the index number:

Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Change "banana" to "pear"
fruits[1] <- "pear"

# Print fruits
fruits
```

Repeat Vectors

To repeat vectors, use the rep() function:

Example

Repeat each value:

```
repeat_each <- rep(c(1,2,3), each = 3)

repeat_each
```

Example

Repeat the sequence of the vector:

```
repeat_times <- rep(c(1,2,3), times = 3)

repeat_times
```

Example

Repeat each value independently:

```
repeat_indepent <- rep(c(1,2,3), times = c(5,2,1))

repeat_indepent
```

Generating Sequenced Vectors

One of the examples on top, showed you how to create a vector with numerical values in a sequence with the : operator:

Example

```
numbers <- 1:10

numbers
```

To make bigger or smaller steps in a sequence, use the seq() function:

Example

```
numbers <- seq(from = 0, to = 100, by = 20)
numbers
```

Note: The seq() function has three parameters: from is where the sequence starts, to is where the sequence stops, and by is the interval of the sequence.

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

Vector Creation

Single Element Vector

Even when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above vector types.

```
# Atomic vector of type character.
print("abc");

# Atomic vector of type double.
print(12.5)

# Atomic vector of type integer.
print(63L)

# Atomic vector of type logical.
print(TRUE)

# Atomic vector of type complex.
print(2+3i)

# Atomic vector of type raw.
print(charToRaw('hello'))
```

When we execute the above code, it produces the following result –

```
[1] "abc"
[1] 12.5
[1] 63
[1] TRUE
[1] 2+3i
[1] 68 65 6c 6c 6f
```

Multiple Elements Vector

Using colon operator with numeric data

```
# Creating a sequence from 5 to 13.
v <- 5:13
print(v)

# Creating a sequence from 6.6 to 12.6.
v <- 6.6:12.6
print(v)

# If the final element specified does not belong to the sequence then it is discarded.
v <- 3.8:11.4
print(v)
```

When we execute the above code, it produces the following result –

```
[1] 5 6 7 8 9 10 11 12 13
[1] 6.6 7.6 8.6 9.6 10.6 11.6 12.6
[1] 3.8 4.8 5.8 6.8 7.8 8.8 9.8 10.8
```

Using sequence (Seq.) operator

```
# Create vector with elements from 5 to 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```
When we execute the above code, it produces the following result –
```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Using the c() function

The non-character values are coerced to character type if one of the elements is a character.
```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5,TRUE)
print(s)
```
When we execute the above code, it produces the following result –
```
[1] "apple" "red" "5" "TRUE"
```

Accessing Vector Elements

Elements of a Vector are accessed using indexing. The **[ ] brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result.**TRUE**, **FALSE** or **0** and **1** can also be used for indexing.
```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)

# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)

# Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
print(x)

# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```
When we execute the above code, it produces the following result –
```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

Vector Manipulation

Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.
```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)

# Vector addition.
add.result <- v1+v2
```

```
print(add.result)

# Vector subtraction.
sub.result <- v1-v2
print(sub.result)

# Vector multiplication.
multi.result <- v1*v2
print(multi.result)

# Vector division.
divi.result <- v1/v2
print(divi.result)
```

When we execute the above code, it produces the following result –

```
[1] 7 19 4 13 1 13
[1] -1 -3 4 -3 -1 9
[1] 12 88 0 40 0 22
[1] 0.7500000 0.7272727 Inf 0.6250000 0.0000000 5.5000000
```

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)

add.result <- v1+v2
print(add.result)

sub.result <- v1-v2
print(sub.result)
```

When we execute the above code, it produces the following result –

```
[1] 7 19 8 16 4 22
[1] -1 -3 0 -6 -4 0
```

Vector Element Sorting

Elements in a vector can be sorted using the **sort()** function.

```
v <- c(3,8,4,5,0,11, -9, 304)

# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)

# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)

# Sorting character vectors.
v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
```

```
print(sort.result)

# Sorting character vectors in reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

When we execute the above code, it produces the following result −

```
[1] -9 0 3 4 5 8 11 304
[1] 304 11 8 5 4 3 0 -9
[1] "Blue" "Red" "violet" "yellow"
[1] "yellow" "violet" "Red" "Blue"
```

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Creating a List
Following is an example to create a list containing strings, numbers, vectors and a logical values.

```
# Create a list containing strings, numbers, vectors and a logical
# values.
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
[1] 119.1
```

Naming List Elements
The list elements can be given names and they can be accessed using these names.

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
  list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
```

```
# Show the list.
print(list_data)
```

When we execute the above code, it produces the following result −

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3
```

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example −

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Access the first element of the list.
print(list_data[1])

# Access the thrid element. As it is also a list, all its elements will be printed.
print(list_data[3])

# Access the list element using the name of the element.
print(list_data$A_Matrix)
```

When we execute the above code, it produces the following result −

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3
```

```
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8
```

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Add element at the end of the list.
list_data[4] <- "New element"
print(list_data[4])

# Remove the last element.
list_data[4] <- NULL

# Print the 4th Element.
print(list_data[4])

# Update the 3rd Element.
list_data[3] <- "updated element"
print(list_data[3])
```

When we execute the above code, it produces the following result –

```
[[1]]
[1] "New element"

$<NA>
NULL

$`A Inner list`
[1] "updated element"
```

Merging Lists

You can merge many lists into one list by placing all the lists inside one list() function.

```
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")

# Merge the two lists.
merged.list <- c(list1,list2)

# Print the merged list.
```

```
print(merged.list)
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

[[6]]
[1] "Tue"
```

Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

```
# Create lists.
list1 <- list(1:5)
print(list1)

list2 <-list(10:14)
print(list2)

# Convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)

print(v1)
print(v2)

# Now add the vectors
result <- v1+v2
print(result)
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] 1 2 3 4 5

[[1]]
[1] 10 11 12 13 14
```

```
[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 19
```

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.
A Matrix is created using the **matrix()** function.

Syntax
The basic syntax for creating a matrix in R is −
matrix(data, nrow, ncol, byrow, dimnames)
Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Example
Create a matrix taking a vector of numbers as input.

```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)

# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)

# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

When we execute the above code, it produces the following result −

```
[,1] [,2] [,3]
[1,] 3 4 5
[2,] 6 7 8
[3,] 9 10 11
[4,] 12 13 14
[,1] [,2] [,3]
[1,] 3 7 11
[2,] 4 8 12
[3,] 5 9 13
[4,] 6 10 14
```

```
col1 col2 col3
row1 3 4 5
row2 6 7 8
row3 9 10 11
row4 12 13 14
```

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# Create the matrix.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

# Access the element at 3rd column and 1st row.
print(P[1,3])

# Access the element at 2nd column and 4th row.
print(P[4,2])

# Access only the  2nd row.
print(P[2,])

# Access only the 3rd column.
print(P[,3])
```

When we execute the above code, it produces the following result –

```
[1] 5
[1] 13
col1 col2 col3
6 7 8
row1 row2 row3 row4
5 8 11 14
```

Matrix Computations

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.

The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

Matrix Addition & Subtraction

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)
```

```
# Add the matrices.
result <- matrix1 + matrix2
cat("Result of addition","\n")
print(result)

# Subtract the matrices
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

When we execute the above code, it produces the following result −

```
[,1] [,2] [,3]
[1,] 3 -1 2
[2,] 9 4 6
[,1] [,2] [,3]
[1,] 5 0 3
[2,] 2 9 4
Result of addition
[,1] [,2] [,3]
[1,] 8 -1 5
[2,] 11 13 10
Result of subtraction
[,1] [,2] [,3]
[1,] -2 -1 -1
[2,] 7 -5 2
```

Matrix Multiplication & Division

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# Multiply the matrices.
result <- matrix1 * matrix2
cat("Result of multiplication","\n")
print(result)

# Divide the matrices
result <- matrix1 / matrix2
cat("Result of division","\n")
print(result)
```

When we execute the above code, it produces the following result −

```
[,1] [,2] [,3]
[1,] 3 -1 2
[2,] 9 4 6
[,1] [,2] [,3]
[1,] 5 0 3
```

```
[2,] 2 9 4
Result of multiplication
[,1] [,2] [,3]
[1,] 15 0 6
[2,] 18 36 24
Result of division
[,1] [,2] [,3]
[1,] 0.6 -Inf 0.6666667
[2,] 4.5 0.4444444 1.5000000
```

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the **array()** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

Example

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

When we execute the above code, it produces the following result −

```
, , 1

[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15

, , 2

[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15
```

Naming Columns and Rows

We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
```

```
# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
   matrix.names))
print(result)
```

When we execute the above code, it produces the following result –

```
, , Matrix1

COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15

, , Matrix2

COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15
```

Accessing Array Elements

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,
   column.names, matrix.names))

# Print the third row of the second matrix of the array.
print(result[3,,2])

# Print the element in the 1st row and 3rd column of the 1st matrix.
print(result[1,3,1])

# Print the 2nd Matrix.
print(result[,,2])
```

When we execute the above code, it produces the following result –

```
COL1 COL2 COL3
3 12 15
[1] 13
COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15
```

Manipulating Array Elements

As array is made up matrices in multiple dimensions, the operations on elements of array are carried out by accessing elements of the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
array1 <- array(c(vector1,vector2),dim = c(3,3,2))

# Create two vectors of different lengths.
vector3 <- c(9,1,0)
vector4 <- c(6,0,11,3,14,1,2,6,9)
array2 <- array(c(vector1,vector2),dim = c(3,3,2))

# create matrices from these arrays.
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]

# Add the matrices.
result <- matrix1+matrix2
print(result)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]
[1,] 10 20 26
[2,] 18 22 28
[3,] 6 24 30
```

Calculations Across Array Elements

We can do calculations across the elements in an array using the **apply()** function.

Syntax

apply(x, margin, fun)

Following is the description of the parameters used –

- **x** is an array.
- **margin** is the name of the data set used.
- **fun** is the function to be applied across the elements of the array.

Example

We use the apply() function below to calculate the sum of the elements in the rows of an array across all the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
new.array <- array(c(vector1,vector2),dim = c(3,3,2))
print(new.array)

# Use apply to calculate the sum of the rows across all the matrices.
```

```
result <- apply(new.array, c(1), sum)
print(result)
```

When we execute the above code, it produces the following result –

```
, , 1

[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15

, , 2

[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15

[1] 56 68 60
```

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.
Factors are created using the **factor ()** function by taking a vector as input.
Example

```
# Create a vector as input.
data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)
print(is.factor(data))

# Apply the factor function.
factor_data <- factor(data)

print(factor_data)
print(is.factor(factor_data))
```

When we execute the above code, it produces the following result –

```
[1] "East" "West" "East" "North" "North" "East" "West" "West" "West" "East" "North"
[1] FALSE
[1] East West East North North East West West West East North
Levels: East North West
[1] TRUE
```

Factors in Data Frame
On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

```
# Create the vectors for data frame.
height <- c(132,151,162,139,166,147,122)
```

```
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","male")

# Create the data frame.
input_data <- data.frame(height,weight,gender)
print(input_data)

# Test if the gender column is a factor.
print(is.factor(input_data$gender))

# Print the gender column so see the levels.
print(input_data$gender)
```

When we execute the above code, it produces the following result –

```
height weight gender
1 132 48 male
2 151 49 male
3 162 66 female
4 139 53 female
5 166 67 male
6 147 52 female
7 122 40 male
[1] TRUE
[1] male male female female male female male
Levels: female male
```

Changing the Order of Levels

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.

```
data <- c("East","West","East","North","North","East","West",
  "West","West","East","North")
# Create the factors
factor_data <- factor(data)
print(factor_data)

# Apply the factor function with required order of the level.
new_order_data <- factor(factor_data,levels = c("East","West","North"))
print(new_order_data)
```

When we execute the above code, it produces the following result –

```
[1] East West East North North East West West West East North
Levels: East North West
[1] East West East North North East West West West East North
Levels: East West North
```

Generating Factor Levels

We can generate factor levels by using the **gl()** function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

```
gl(n, k, labels)
```

Following is the description of the parameters used –

- **n** is a integer giving the number of levels.

- **k** is a integer giving the number of replications.
- **labels** is a vector of labels for the resulting factor levels.

Example

```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

When we execute the above code, it produces the following result –

```
Tampa Tampa Tampa Tampa Seattle Seattle Seattle Seattle Boston
[10] Boston Boston Boston
Levels: Tampa Seattle Boston
```

A **data frame** is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result –

```
emp_id emp_name salary start_date
1 1 Rick 623.30 2012-01-01
2 2 Dan 515.20 2013-09-23
3 3 Michelle 611.00 2014-11-15
4 4 Ryan 729.00 2014-05-11
5 5 Gary 843.25 2015-03-27
```

Get the Structure of the Data Frame

The structure of the data frame can be seen by using **str()** function.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(emp.data)
```

When we execute the above code, it produces the following result –

```
'data.frame': 5 obs. of 4 variables:
$ emp_id : int 1 2 3 4 5
$ emp_name : chr "Rick" "Dan" "Michelle" "Ryan" ...
$ salary : num 623 515 611 729 843
$ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying **summary()** function.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

When we execute the above code, it produces the following result –

```
emp_id emp_name salary start_date
Min. :1 Length:5 Min. :515.2 Min. :2012-01-01
1st Qu.:2 Class :character 1st Qu.:611.0 1st Qu.:2013-09-23
Median :3 Mode :character Median :623.3 Median :2014-05-11
Mean :3 Mean :664.4 Mean :2014-01-14
3rd Qu.:4 3rd Qu.:729.0 3rd Qu.:2014-11-15
Max. :5 Max. :843.2 Max. :2015-03-27
```

Extract Data from Data Frame

Extract specific column from a data frame using column name.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
```

```
# Extract Specific columns.
result <- data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

When we execute the above code, it produces the following result −

```
emp.data.emp_name emp.data.salary
1 Rick 623.30
2 Dan 515.20
3 Michelle 611.00
4 Ryan 729.00
5 Gary 843.25
```

Extract the first two rows and then all columns

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Extract first two rows.
result <- emp.data[1:2,]
print(result)
```

When we execute the above code, it produces the following result −

```
emp_id emp_name salary start_date
1 1 Rick 623.3 2012-01-01
2 2 Dan 515.2 2013-09-23
```

Extract 3rd and 5th row with 2nd and 4th column

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

          start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)

# Extract 3rd and 5th row with 2nd and 4th column.
result <- emp.data[c(3,5),c(2,4)]
print(result)
```

When we execute the above code, it produces the following result −

```
emp_name start_date
3 Michelle 2014-11-15
```

```
5 Gary 2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)

# Add the "dept" coulmn.
emp.data$dept <- c("IT","Operations","IT","HR","Finance")
v <- emp.data
print(v)
```

When we execute the above code, it produces the following result –

```
emp_id emp_name salary start_date dept
1 1 Rick 623.30 2012-01-01 IT
2 2 Dan 515.20 2013-09-23 Operations
3 3 Michelle 611.00 2014-11-15 IT
4 4 Ryan 729.00 2014-05-11 HR
5 5 Gary 843.25 2015-03-27 Finance
```

Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the **rbind()** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

```
# Create the first data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  dept = c("IT","Operations","IT","HR","Finance"),
  stringsAsFactors = FALSE
)

# Create the second data frame
```

```
emp.newdata <-        data.frame(
  emp_id = c (6:8),
  emp_name = c("Rasmi","Pranab","Tusar"),
  salary = c(578.0,722.5,632.8),
  start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
  dept = c("IT","Operations","Fianance"),
  stringsAsFactors = FALSE
)

# Bind the two data frames.
emp.finaldata <- rbind(emp.data,emp.newdata)
print(emp.finaldata)
```

When we execute the above code, it produces the following result –

```
emp_id emp_name salary start_date dept
1 1 Rick 623.30 2012-01-01 IT
2 2 Dan 515.20 2013-09-23 Operations
3 3 Michelle 611.00 2014-11-15 IT
4 4 Ryan 729.00 2014-05-11 HR
5 5 Gary 843.25 2015-03-27 Finance
6 6 Rasmi 578.00 2013-05-21 IT
7 7 Pranab 722.50 2013-07-30 Operations
8 8 Tusar 632.80 2014-06-17 Finance
```

R **packages** are a collection of R functions, complied code and sample data. They are stored under a directory called **"library"** in the R environment. By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose. When we start the R console, only the default packages are available by default. Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.
All the packages available in R language are listed at R Packages.
Below is a list of commands to be used to check, verify and use the R packages.

Check Available R Packages
Get library locations containing R packages

```
.libPaths()
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.

```
[2] "C:/Program Files/R/R-3.2.2/library"
```

Get the list of all the packages installed

```
library()
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.
Install a New Package
There are two ways to add new R packages. One is installing directly from the CRAN directory and another is downloading the package to your local system and installing it manually.
Install directly from CRAN
The following command gets the packages directly from CRAN webpage and installs the package in the R environment. You may be prompted to choose a nearest mirror. Choose the one appropriate to your location.

```
install.packages("Package Name")

# Install the package named "XML".
install.packages("XML")
```

Install package manually

Go to the link R Packages to download the package needed. Save the package as a **.zip** file in a suitable location in the local system.

Now you can run the following command to install this package in the R environment.

```
install.packages(file_name_with_path, repos = NULL, type = "source")

# Install the package named "XML"
install.packages("E:/XML_3.98-1.3.zip", repos = NULL, type = "source")
```

Load Package to Library

Before a package can be used in the code, it must be loaded to the current R environment. You also need to load a package that is already installed previously but not available in the current environment.

A package is loaded using the following command −

```
library("package Name", lib.loc = "path to library")

# Load the package named "XML"
install.packages("E:/XML_3.98-1.3.zip", repos = NULL, type = "source")
```

Data Reshaping in R is about changing the way data is organized into rows and columns. Most of the time data processing in R is done by taking the input data as a data frame. It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from format in which we received it. R has many functions to split, merge and change the rows to columns and vice-versa in a data frame.

Joining Columns and Rows in a Data Frame

We can join multiple vectors to create a data frame using the **cbind()**function. Also we can merge two data frames using **rbind()** function.

```
# Create vector objects.
city <- c("Tampa","Seattle","Hartford","Denver")
state <- c("FL","WA","CT","CO")
zipcode <- c(33602,98104,06161,80294)

# Combine above three vectors into one data frame.
addresses <- cbind(city,state,zipcode)

# Print a header.
cat("# # # # The First data frame\n")

# Print the data frame.
print(addresses)

# Create another data frame with similar columns
new.address <- data.frame(
   city = c("Lowry","Charlotte"),
   state = c("CO","FL"),
  zipcode = c("80230","33949"),
  stringsAsFactors = FALSE
)

# Print a header.
cat("# # # The Second data frame\n")

# Print the data frame.
print(new.address)

# Combine rows form both the data frames.
all.addresses <- rbind(addresses,new.address)

# Print a header.
cat("# # # The combined data frame\n")

# Print the result.
print(all.addresses)
```

When we execute the above code, it produces the following result −

```
# # # # The First data frame
city state zipcode
[1,] "Tampa" "FL" "33602"
[2,] "Seattle" "WA" "98104"
[3,] "Hartford" "CT" "6161"
[4,] "Denver" "CO" "80294"

# # # The Second data frame
city state zipcode
1 Lowry CO 80230
2 Charlotte FL 33949

# # # The combined data frame
city state zipcode
1 Tampa FL 33602
2 Seattle WA 98104
3 Hartford CT 6161
4 Denver CO 80294
5 Lowry CO 80230
6 Charlotte FL 33949
```

Merging Data Frames

We can merge two data frames by using the **merge()** function. The data frames must have same column names on which the merging happens.

In the example below, we consider the data sets about Diabetes in Pima Indian Women available in the library names "MASS". we merge the two data sets based on the values of blood pressure("bp") and body mass index("bmi"). On choosing these two columns for merging, the records where values of these two variables match in both data sets are combined together to form a single data frame.

```
library(MASS)
```

```
merged.Pima <- merge(x = Pima.te, y = Pima.tr,
  by.x = c("bp", "bmi"),
  by.y = c("bp", "bmi")
)
print(merged.Pima)
nrow(merged.Pima)
```

When we execute the above code, it produces the following result −

```
bp bmi npreg.x glu.x skin.x ped.x age.x type.x npreg.y glu.y skin.y ped.y
1 60 33.8 1 117 23 0.466 27 No 2 125 20 0.088
2 64 29.7 2 75 24 0.370 33 No 2 100 23 0.368
3 64 31.2 5 189 33 0.583 29 Yes 3 158 13 0.295
4 64 33.2 4 117 27 0.230 24 No 1 96 27 0.289
5 66 38.1 3 115 39 0.150 28 No 1 114 36 0.289
6 68 38.5 2 100 25 0.324 26 No 7 129 49 0.439
7 70 27.4 1 116 28 0.204 21 No 0 124 20 0.254
8 70 33.1 4 91 32 0.446 22 No 9 123 44 0.374
9 70 35.4 9 124 33 0.282 34 No 6 134 23 0.542
10 72 25.6 1 157 21 0.123 24 No 4 99 17 0.294
11 72 37.7 5 95 33 0.370 27 No 6 103 32 0.324
12 74 25.9 9 134 33 0.460 81 No 8 126 38 0.162
13 74 25.9 1 95 21 0.673 36 No 8 126 38 0.162
14 78 27.6 5 88 30 0.258 37 No 6 125 31 0.565
15 78 27.6 10 122 31 0.512 45 No 6 125 31 0.565
16 78 39.4 2 112 50 0.175 24 No 4 112 40 0.236
17 88 34.5 1 117 24 0.403 40 Yes 4 127 11 0.598
age.y type.y
1 31 No
2 21 No
3 24 No
4 21 No
5 21 No
6 43 Yes
7 36 Yes
8 40 No
9 29 Yes
10 28 No
11 55 No
12 39 No
13 39 No
14 49 Yes
15 49 Yes
16 38 No
17 28 No
[1] 17
```

Melting and Casting

One of the most interesting aspects of R programming is about changing the shape of the data in multiple steps to get a desired shape. The functions used to do this are called **melt()** and **cast()**.

We consider the dataset called ships present in the library called "MASS".

```
library(MASS)
print(ships)
```

When we execute the above code, it produces the following result −

```
type year period service incidents
1 A 60 60 127 0
2 A 60 75 63 0
3 A 65 60 1095 3
4 A 65 75 1095 4
5 A 70 60 1512 6
............
............
8 A 75 75 2244 11
9 B 60 60 44882 39
10 B 60 75 17176 29
11 B 65 60 28609 58
...........
...........
17 C 60 60 1179 1
18 C 60 75 552 1
19 C 65 60 781 0
...........
...........
```

Melt the Data

Now we melt the data to organize it, converting all columns other than type and year into multiple rows.

```
molten.ships <- melt(ships, id = c("type","year"))
print(molten.ships)
```

When we execute the above code, it produces the following result −

```
type year variable value
1 A 60 period 60
2 A 60 period 75
3 A 65 period 60
4 A 65 period 75
...........
...........
9 B 60 period 60
10 B 60 period 75
11 B 65 period 60
12 B 65 period 75
13 B 70 period 60
..........
..........
41 A 60 service 127
42 A 60 service 63
43 A 65 service 1095
..........
..........
70 D 70 service 1208
```

```
71 D 75 service 0
72 D 75 service 2051
73 E 60 service 45
74 E 60 service 0
75 E 65 service 789
..........
..........
101 C 70 incidents 6
102 C 70 incidents 2
103 C 75 incidents 0
104 C 75 incidents 1
105 D 60 incidents 0
106 D 60 incidents 0
..........
..........
```

Cast the Molten Data

We can cast the molten data into a new form where the aggregate of each type of ship for each year is created. It is done using the **cast()** function.

```
recasted.ship <- cast(molten.ships, type+year~variable,sum)
print(recasted.ship)
```

When we execute the above code, it produces the following result −

```
type year period service incidents
1 A 60 135 190 0
2 A 65 135 2190 7
3 A 70 135 4865 24
4 A 75 135 2244 11
5 B 60 135 62058 68
6 B 65 135 48979 111
7 B 70 135 20163 56
8 B 75 135 7117 18
9 C 60 135 1731 2
10 C 65 135 1457 1
11 C 70 135 2731 8
12 C 75 135 274 1
13 D 60 135 356 0
14 D 65 135 480 0
15 D 70 135 1557 13
16 D 75 135 2051 4
17 E 60 135 45 0
18 E 65 135 1226 14
19 E 70 135 3318 17
20 E 75 135 542 1
```

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

Getting and Setting the Working Directory

You can check which directory the R workspace is pointing to using the getwd() function. You can also set a new working directory using setwd()function.

```
# Get and print current working directory.
print(getwd())

# Set current working directory.
setwd("/web/com")

# Get and print current working directory.
print(getwd())
```

When we execute the above code, it produces the following result –

```
[1] "/web/com/1441086124_2016"
[1] "/web/com"
```

This result depends on your OS and your current directory where you are working.

Input as CSV File

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named input.csv.

You can create this file using windows notepad by copying and pasting this data. Save the file as input.csv using the save As All files(*.*) option in notepad.

```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

Reading a CSV File

Following is a simple example of read.csv() function to read a CSV file available in your current working directory –

```
data <- read.csv("input.csv")
print(data)
```

When we execute the above code, it produces the following result –

```
id, name, salary, start_date, dept
1 1 Rick 623.30 2012-01-01 IT
2 2 Dan 515.20 2013-09-23 Operations
3 3 Michelle 611.00 2014-11-15 IT
4 4 Ryan 729.00 2014-05-11 HR
5 NA Gary 843.25 2015-03-27 Finance
```

```
6 6 Nina 578.00 2013-05-21 IT
7 7 Simon 632.80 2013-07-30 Operations
8 8 Guru 722.50 2014-06-17 Finance
```

Analyzing the CSV File

By default the read.csv() function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```
data <- read.csv("input.csv")

print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

When we execute the above code, it produces the following result –

```
[1] TRUE
[1] 5
[1] 8
```

Once we read data in a data frame, we can apply all the functions applicable to data frames as explained in subsequent section.

Get the maximum salary

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)
```

When we execute the above code, it produces the following result –

```
[1] 843.25
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)

# Get the person detail having max salary.
retval <- subset(data, salary == max(salary))
print(retval)
```

When we execute the above code, it produces the following result –

```
id name salary start_date dept
5 NA Gary 843.25 2015-03-27 Finance
```

Get all the people working in IT department

```
# Create a data frame.
```

```
data <- read.csv("input.csv")

retval <- subset( data, dept == "IT")
print(retval)
```

When we execute the above code, it produces the following result –

```
id name salary start_date dept
1 1 Rick 623.3 2012-01-01 IT
3 3 Michelle 611.0 2014-11-15 IT
6 6 Nina 578.0 2013-05-21 IT
```

Get the persons in IT department whose salary is greater than 600

```
# Create a data frame.
data <- read.csv("input.csv")

info <- subset(data, salary > 600 & dept == "IT")
print(info)
```

When we execute the above code, it produces the following result –

```
id name salary start_date dept
1 1 Rick 623.3 2012-01-01 IT
3 3 Michelle 611.0 2014-11-15 IT
```

Get the people who joined on or after 2014

```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
print(retval)
```

When we execute the above code, it produces the following result –

```
id name salary start_date dept
3 3 Michelle 611.00 2014-11-15 IT
4 4 Ryan 729.00 2014-05-11 HR
5 NA Gary 843.25 2015-03-27 Finance
8 8 Guru 722.50 2014-06-17 Finance
```

Writing into a CSV File

R can create csv file form existing data frame. The write.csv() function is used to create the csv file. This file gets created in the working directory.

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```

When we execute the above code, it produces the following result –

```
X id name salary start_date dept
1 3 3 Michelle 611.00 2014-11-15 IT
2 4 4 Ryan 729.00 2014-05-11 HR
3 5 NA Gary 843.25 2015-03-27 Finance
4 8 8 Guru 722.50 2014-06-17 Finance
```

Here the column X comes from the data set newper. This can be dropped using additional parameters while writing the file.

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv", row.names = FALSE)
newdata <- read.csv("output.csv")
print(newdata)
```

When we execute the above code, it produces the following result –

```
id name salary start_date dept
1 3 Michelle 611.00 2014-11-15 IT
2 4 Ryan 729.00 2014-05-11 HR
3 NA Gary 843.25 2015-03-27 Finance
4 8 Guru 722.50 2014-06-17 Finance
```

Unit 2:

Basic concepts of estimation, Confidence intervals and tests of hypotheses, Contingency tables, One-way unstructured comparisons, Response curves, Data with a nested variation structure, Resampling methods for standard errors, tests, and confidence intervals, Theories of inference, Regression with a single predictor, multiple linear regressions.

Basic concepts of estimation

We often have questions concerning large populations. Gathering information from the entire population is not always possible due to barriers such as time, accessibility, or cost. Instead of gathering information from the whole population, we often gather information from a smaller subset of the population, known as a sample.
Values concerning a sample are referred to as sample statistics while values concerning a population are referred to as population parameters.
Population: The entire set of possible cases
Sample: A subset of the population from which data are collected
Statistic: A measure concerning a sample (e.g., sample mean)
Parameter: A measure concerning a population (e.g., population mean)
The process of using sample statistics to make conclusions about population parameters is known as inferential statistics. In other words, data from a sample are used to make an inference about a population.
Inferential Statistics: Statistical procedures that use data from an observed sample to make a conclusion about a population

Example: Student Housing
A survey is carried out at Penn State Altoona to estimate the proportion of all undergraduate students living at home during the current term. Of the 3,838 undergraduate students enrolled at the campus, a random sample of 100 was surveyed.

- Population: All 3,838 undergraduate students at Penn State Altoona
- Sample: The 100 undergraduate students surveyed

We can use the data collected from the sample of 100 students to make inferences about the population of all 3,838 students.

Population parameters and sample statistics
Parameters, such as the mean (μ) or standard deviation (σ), numerically summarize various aspects of a population. Such parameters are usually unknown and are estimated using statistics calculated using a random sample taken from the population.
The sample mean is an example of a statistic, and it is used to estimate the population mean.
Other commonly used statistics are the proportion, standard deviation, variance, median, the quartiles, the slope of a regression line, and the correlation coefficient. Each may be used as an estimate of the corresponding population parameter.

Sampling distributions
The standard deviation (SD) measures the amount of variability, or dispersion, from the individual data values to the mean.
Standard error of the mean (SEM) measures how far the sample mean (average) of the data is likely to be from the true population mean. The SEM is always smaller than the SD. The SEM is a measure of the accuracy of the sample mean, as an estimate of the population mean. The SEM takes the SD and divides it by the square root of the sample size.
A sampling distribution is a probability distribution of a statistic that is obtained through repeated sampling of a specific population. It describes a range of possible outcomes for a statistic, such as the mean or mode of some variable, of a population. The majority of data analyzed by researchers are actually samples, not populations.

Calculating SD and SEM

The sampling distribution of the mean can, for a population with mean μ and standard deviation σ, often is well approximated by a normal distribution with mean μ and standard deviation $\sigma/\sqrt{n}$.
An estimate of the SEM is thus SEM = $s/\sqrt{n}$, where s is an estimator of the population standard deviation σ.

The formula for the SD requires a few steps:

1. First, take the square of the difference between each data point and the sample mean, finding the sum of those values.
2. Next, divide that sum by the sample size minus one, which is the variance.
3. Finally, take the square root of the variance to get the SD.

Standard Error of the Mean
SEM is calculated simply by taking the standard deviation and dividing it by the square root of the sample size.

Central Limit Theorem
The central limit theorem (CLT) states that the distribution of sample means approximates a normal distribution as the sample size gets larger, regardless of the population's distribution.
Sample sizes equal to or greater than 30 are often considered sufficient for the CLT to hold.
A key aspect of CLT is that the average of the sample means and standard deviations will equal the population mean and standard deviation.
A sufficiently large sample size can predict the characteristics of a population more accurately.

The selection of some of the employees/population from the whole employees/population list is known as **Sample**.
Let's say we have a company in which 30,000 employees are working. We want to find out the daily commute time of all the employees.

Understand the **Sampling Terminology**

- Total number of items/population**, Population Size** = **N**
- Mean of the population, **Population Mean(μ) = (Σ * X)/N**
- Variance of the population, **Population Variance(σ^2) = Σ(Xi — μ)^2/ N**
- Number of items/population, **Sample Size = n**
- Mean of the sample employees, **Sample Mean($\bar{x}$)** = **(Σ * x)/n**
- Variance of the sample, **Sample Variance(S^2) = Σ(xi — $\bar{x}$)^2/ n-1**

Assessing accuracy – the standard error
Eighteen elastic bands were divided into nine pairs, with bands of similar stretchiness placed in the same pair. One member of each pair was placed in hot water (60–65 ◦C) for four minutes, while the other was left at ambient temperature. After a wait of about 10 minutes, the amounts of stretch, under a 1.35 kg weight, were recorded.

Pair #	1	2	3	4	5	6	7	8	9
Heated (mm)	244	255	253	254	251	269	248	252	292
Ambient	225	247	249	253	245	259	242	255	286
Difference	19	8	4	1	6	10	6	−3	6

The mean is 6.33, the SD is s = 6.10, and SEM = $6.10/\sqrt{9}$ = 2.03.
"The mean change is 6.33 [SEM 2.03], based on n = 9 values", or "The mean change is 6.10/2.03 (= 3.11) times the standard error".

The standard error for the difference of means
Where there are two independent samples of size n1 and n2, the comparison is usually in the form of a difference:

$$\bar{x}_1 - \bar{x}_2$$

where $\bar{x}_1$ and $\bar{x}_2$ denote the respective sample means.
If the corresponding standard errors are denoted by SEM1 and SEM2, then the standard error of the difference (SED) is

SED = $\sqrt{SEM_1{}^2 + SEM_2{}^2}$

If all SEMs are the same, then for all comparisons, SED = $\sqrt{2}$ × SEM.
It is sometimes reasonable to assume equality of the standard deviations in the populations from which the samples are drawn.

Then SEM1 = $\frac{s}{\sqrt{n_1}}$, SEM2 = $\frac{s}{\sqrt{n_2}}$ and the formula can be written as SED = s $\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$
where s is the pooled standard deviation estimate.

Eg. Consider data from an experiment in which 21 elastic bands were randomly divided into two groups, one of 10 and one of 11. Bands in the first group were immediately tested for the amount that they stretched under a weight of 1.35 kg. The other group were dunked in hot water at 65◦C for four minutes, then left at air temperature for ten minutes, and then tested for the amount that they stretched under the same 1.35 kg weight as before. The results were:

Ambient:	254 252 239 240 250 256 267 249 259 269 (Mean = 253.5)
Heated:	233 252 237 246 255 244 248 242 217 257 254 (Mean = 244.1)

The pooled standard deviation estimate is s = 10.91, with 19 (= 10 + 11 − 2) degrees of freedom. Since the separate standard deviations (s1 = 9.92; s2 = 11.73) are similar, the pooled standard deviation estimate is an acceptable summary of the variation in the data
The pooled standard deviation estimate is 10.91. Hence, the SED is 10.91 × (1/10 + 1/11) = 4.77

The standard error of the median
For data from a normal distribution, there is a similarly simple formula for the standard error of the median. It is
$SE_{median} = \frac{\pi}{2}\frac{s}{\sqrt{n}} \approx 1.25\frac{s}{\sqrt{n}}$
The standard error of the median is thus about 25% greater than the standard error of the mean. For data from a normal distribution, the population mean can be estimated more precisely than can the population median.

The sampling distribution of the t-statistic
The formula t = $\frac{\bar{x} - \mu}{SEM}$ counts the number of standard error units between the true value μ and the sample estimate, $\bar{x}$. It can be thought of as a standardized distance between the true mean and the sample mean.
The variability in t has two sources: the sampling variability of $\bar{x}$ and the sampling variability of SEM.
The replacing of σ by s introduces an uncertainty that is larger as the degrees of freedom n − 1 in s are smaller. Hence the use of a t-distribution with n – 1 degrees of freedom (d.f.), where if σ was known precisely a normal distribution would be used. The t-statistic becomes more and more like a standard normal random variable as the sample size increases.

Confidence intervals and tests of hypotheses

Calculations with the t-distribution
A **t-test** is a statistical hypothesis test used to determine if there is a significant difference (differences are measured in means) between two groups and estimate the likelihood that this difference exists purely by chance (p-value). In a t-distribution, a test statistic called **t-score** or t-value is used to describe how far away an observation is from the mean. The t-score is used in t-tests**,** regression tests and to calculate confidence intervals.

Functions used:
To find the value of probability density function (pdf) of the Student's t-distribution given a random variable x, use the dt() function in R.
Syntax: dt(x, df)

Parameters:

- *x is the quantiles vector*
- *df is the degrees of freedom*

pt() function is used to get the cumulative distribution function (CDF) of a t-distribution

Syntax: pt(q, df, lower.tail = TRUE)

Parameter:

- *q is the quantiles vector*
- *df is the degrees of freedom*
- *lower.tail – if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X > x].*

The qt() function is used to get the quantile function or inverse cumulative density function of a t-distribution.

Syntax: qt(p, df, lower.tail = TRUE)

Parameter:

- *p is the vector of probabilities*
- *df is the degrees of freedom*
- *lower.tail – if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X > x].*

Approach

- Set degrees of freedom
- To plot the density function for student's t-distribution follow the given steps:
  - First create a vector of quantiles in R.
  - Next, use the dt function to find the values of a t-distribution given a random variable x and certain degrees of freedom.
  - Using these values plot the density function for student's t-distribution.
- Now, instead of the dt function, use the pt function to get the cumulative distribution function (CDF) of a t-distribution and the qt function to get the quantile function or inverse cumulative density function of a t-distribution. Put it simply, pt returns the area to the left of a given random variable q in the t-distribution and qt finds the t-score is of the pth quantile of the t-distribution.

Example: To find a value of t-distribution at x=1, having certain degrees of freedom, say D_f = 25,

```
# value of t-distribution pdf at
# x = 0 with 25 degrees of freedom
dt(x = 1, df = 25)
```

Output:

```
0.237211
```

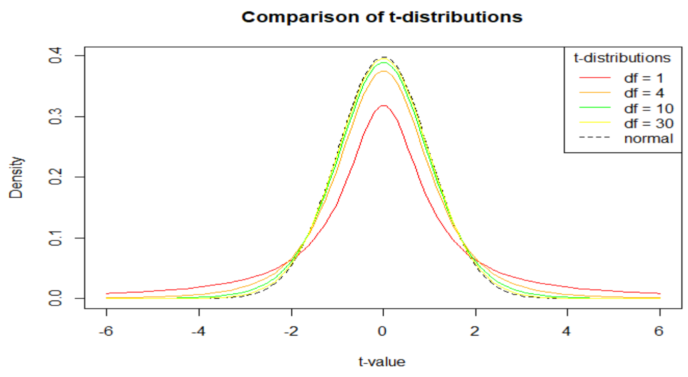
Example:

Code below shows a comparison of probability density functions having different degrees of freedom. It is observed as mentioned before, larger the sample size (degrees of freedom increasing), the closer the plot is to a normal distribution (dotted line in figure).

```
# Generate a vector of 100 values between -6 and 6
x <- seq(-6, 6, length = 100)
  # Degrees of freedom
df = c(1,4,10,30)
colour = c("red", "orange", "green", "yellow","black")
  # Plot a normal distribution
```

```
plot(x, dnorm(x), type = "l", lty = 2, xlab = "t-value", ylab = "Density",
     main = "Comparison of t-distributions", col = "black")
 # Add the t-distributions to the plot
for (i in 1:4){
  lines(x, dt(x, df[i]), col = colour[i])
}
  # Add a legend
legend("topright", c("df = 1", "df = 4", "df = 10", "df = 30", "normal"),
 col = colour, title = "t-distributions", lty = c(1,1,1,1,2))
```

Output:



Example: Finding p-value and confidence interval with t-distribution

```
# area to the right of a t-statistic with
# value of 2.1 and 14 degrees of freedom
pt(q = 2.1, df = 14, lower.tail = FALSE)
```

Output:

```
0.02716657
```

Essentially we found the one-sided p-value, P(t>2.1) as 2.7%. Now suppose we want to construct a two-sided 95% confidence interval. To do so, find the t-score or t-value for 95% confidence using the qt function or the quantile distribution.

Example:

```
# value in each tail is 2.5% as confidence is 95%
# find 2.5th percentile of t-distribution with
# 14 degrees of freedom
qt(p = 0.025, df = 14, lower.tail = TRUE)
```

Output:

```
-2.144787
```

So, a t-value of 2.14 will be used as the critical value for a confidence interval of 95%.

Tests of hypotheses

The four normal distribution functions are:

- dnorm: density function of the normal distribution
- pnorm: cumulative density function of the normal distribution
- qnorm: quantile function of the normal distribution
- rnorm: random sampling from the normal distribution

The probability density function: dnorm

The probability density function (PDF, in short: density) indicates the probability of observing a measurement with a specific value and thus the integral over the density is always

1. For a value x, the normal density is defined as

$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

where μ is the mean, σ is the standard deviation, and σ^2 is the variance.

Using the density, it is possible to determine the probabilities of events. For example, you may wonder: *What is the likelihood that a person has an IQ of exactly 140?*. In this case, you would need to retrieve the density of the IQ distribution at value 140. The IQ distribution can be modeled with a mean of 100 and a standard deviation of 15.

```
sample.range <- 50:150
iq.mean <- 100
iq.sd <- 15
iq.dist <- dnorm(sample.range, mean = iq.mean, sd = iq.sd)
iq.df <- data.frame("IQ" = sample.range, "Density" = iq.dist)
library(ggplot2)
ggplot(iq.df, aes(x = IQ, y = Density)) + geom_point()

pp <- function(x) {
   print(paste0(round(x * 100, 3), "%"))
}
# likelihood of IQ == 140?
pp(iq.df$Density[iq.df$IQ == 140])
## [1] "0.076%"

# likelihood of IQ >= 140?
pp(sum(iq.df$Density[iq.df$IQ >= 140]))
## [1] "0.384%"

# likelihood of 50 < IQ <= 90?
pp(sum(iq.df$Density[iq.df$IQ <= 90]))
## [1] "26.284%"
```

The cumulative density function: pnorm

The cumulative density (CDF) function is a monotonically increasing function as it integrates over densities via

$f(x|\mu, \sigma) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$

where $\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}}\int_{-x}^{x} e^{-t^2} dt$ is the error function.

```
cdf <- pnorm(sample.range, iq.mean, iq.sd)
iq.df <- cbind(iq.df, "CDF_LowerTail" = cdf)
```

```
ggplot(iq.df, aes(x = IQ, y = CDF_LowerTail)) + geom_point()
```

The depicted CDF shows the probability of having an IQ less or equal to a given value. This is because pnorm computes the lower tail by default, i.e. P[X<=x].

A hypothesis is made by the researchers about the data collected for any experiment or data set. A hypothesis is an assumption made by the researchers that are not mandatory true. In simple words, a hypothesis is a decision taken by the researchers based on the data of the population collected. Hypothesis Testing in R Programming is a process of testing the hypothesis made by the researcher or to validate the hypothesis. To perform hypothesis testing, a random sample of data from the population is taken and testing is performed. Based on the results of testing, the hypothesis is either selected or rejected. This concept is known as Statistical Inference.

Four Step Process of Hypothesis Testing

There are 4 major steps in hypothesis testing:

- State the hypothesis- This step is started by stating null and alternative hypothesis which is presumed as true.
- Formulate an analysis plan and set the criteria for decision- In this step, significance level of test is set. The significance level is the probability of a false rejection in a hypothesis test.
- Analyze sample data- In this, a test statistic is used to formulate the statistical comparison between the sample mean and the mean of the population or standard deviation of the sample and standard deviation of the population.
- Interpret decision- The value of the test statistic is used to make the decision based on the significance level. For example, if the significance level is set to 0.1 probability, then the sample mean less than 10% will be rejected. Otherwise, the hypothesis is retained to be true.

One Sample T-Testing

One sample T-Testing approach collects a huge amount of data and tests it on random samples. To perform T-Test in R, normally distributed data is required. This test is used to test the mean of the sample with the population. For example, the height of persons living in an area is different or identical to other persons living in other areas.

Syntax: t.test(x, mu)
Parameters:
x: represents numeric vector of data
mu: represents true value of the mean

To know about more optional parameters of t.test(), try below command:

```
help("t.test")
```

Example:

```
# Defining sample vector
x <- rnorm(100)

# One Sample T-Test
t.test(x, mu = 5)
```

Output:

```
    One Sample t-test

data:  x
t = -49.504, df = 99, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 5
```

```
95 percent confidence interval:
 -0.1910645  0.2090349
sample estimates:
  mean of x
0.008985172
```

Two Sample T-Testing

In two sample T-Testing, the sample vectors are compared. If var.equal = TRUE, the test assumes that the variances of both the samples are equal.

Syntax: t.test(x, y)

Parameters:

x and y: Numeric vectors

Example:

```
# Defining sample vector
x <- rnorm(100)
y <- rnorm(100)

# Two Sample T-Test
t.test(x, y)
```

Output:

```
        Welch Two Sample t-test

data:  x and y
t = -1.0601, df = 197.86, p-value = 0.2904
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.4362140  0.1311918
sample estimates:
  mean of x   mean of y
-0.05075633  0.10175478
```

Directional Hypothesis

Using the directional hypothesis, the direction of the hypothesis can be specified like, if the user wants to know the sample mean is lower or greater than another mean sample of the data.

Syntax: t.test(x, mu, alternative)

Parameters:

x: represents numeric vector data

mu: represents mean against which sample data has to be tested

alternative: sets the alternative hypothesis

Example:

```
# Defining sample vector
x <- rnorm(100)

# Directional hypothesis testing
t.test(x, mu = 2, alternative = 'greater')
```

Output:

```
        One Sample t-test
```

```
data:  x
t = -20.708, df = 99, p-value = 1
alternative hypothesis: true mean is greater than 2
95 percent confidence interval:
 -0.2307534        Inf
sample estimates:
 mean of x
-0.0651628
```

One Sample -Test
This type of test is used when comparison has to computed on one sample and the data is non-parametric. It is performed using wilcox.test() function in R programming.
Syntax: wilcox.test(x, y, exact = NULL)
Parameters:
x and y: represents numeric vector
exact: represents logical value which indicates whether p-value be computed
To know about more optional parameters of wilcox.test(), use below command:
help("wilcox.test")

Example:
```
# Define vector
x <- rnorm(100)

# one sample test
wilcox.test(x, exact = FALSE)
```

Output:
```
        Wilcoxon signed rank test with continuity correction

data:  x
V = 2555, p-value = 0.9192
alternative hypothesis: true location is not equal to 0
```

Two Sample -Test
This test is performed to compare two samples of data.

Example:
```
# Define vectors
x <- rnorm(100)
y <- rnorm(100)

# Two sample test
wilcox.test(x, y)
```

Output:
```
        Wilcoxon rank sum test with continuity correction

data:  x and y
```

```
W = 5300, p-value = 0.4643
alternative hypothesis: true location shift is not equal to 0
```

Correlation Test

This test is used to compare the correlation of the two vectors provided in the function call or to test for the association between the paired samples.

Syntax: cor.test(x, y)

Parameters:

x and y: represents numeric data vectors

To know about more optional parameters in cor.test() function, use below command:

```
help("cor.test")
```

Example:

```
# Using mtcars dataset in R
cor.test(mtcars$mpg, mtcars$hp)
```

Output:

```
        Pearson's product-moment correlation

data:  mtcars$mpg and mtcars$hp
t = -6.7424, df = 30, p-value = 1.788e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.8852686 -0.5860994
sample estimates:
       cor
-0.7761684
```

Contingency tables:

Contingency tables are very useful to condense a large number of observations into smaller to make it easier to maintain tables. A contingency table shows the distribution of a variable in the rows and another in its columns. Contingency tables are not only useful for condensing data, but they also show the relations between variables. They are a way of summarizing categorical variables. A contingency table that deals with a single table are called a **complex or a flat contingency table.**

Making Contingency tables

A contingency table is a way to redraw data and assemble it into a table. And, it shows the layout of the original data in a manner that allows the reader to gain an overall summary of the original data. The **table()** function is used in R to create a contingency table. The **table()** function is one of the most versatile functions in R. It can take any data structure as an argument and turn it into a table. The more complex the original data, the more complex is the resulting contingency table.

Creating contingency tables from Vectors

In R a vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures. It is the simplest data object from which you can create a contingency table.

Example:

```
# R program to illustrate
# Contingency Table

# Creating a vector
vec = c(2, 4, 3, 1, 6, 3, 2, 1, 4, 5)
```

```
# Creating contingency table from vec using table()
conTable = table(vec)
print(conTable)
```

Output:

```
vec
1 2 3 4 5 6
2 2 2 2 1 1
```

In the given program what happens is first when we execute table() command on the vector it sorts the vector value and also prints the frequencies of every element given in the vector.

Creating contingency tables from Data

Now we will see a simple example that provides a data frame containing character values in one column and also containing a factor in one of its columns. This one column of factors contains character variables. In order to create our contingency table from data, we will make use of the table(). In the following example, the table() function returns a contingency table. Basically, it returns a tabular result of the categorical variables.

Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Creating contingency table from data using table()
conTable = table(df)
print(conTable)
```

Output:

```
        Gender
 Name   Female  Male
 Amiya     0      1
 Asish     0      1
 Rosy      1      0
```

Creating custom contingency tables

The contingency table in R can be created using only a part of the data which is in contrast with collecting data from all the rows and columns. We can create a custom contingency table in R using the following ways:

- Using Columns of a Data Frame in a Contingency Table
- Using Rows of a Data Frame in a Contingency Table
- By Rotating Data Frames in R
- Creating Contingency Tables from Matrix Objects in R

- **Using Columns of a Data Frame in a Contingency Table:** With the help of table() command, we are able to specify the columns with which the contingency tables can be created. In order to do so, you only need to pass the name of vector objects in the parameter of table() command.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Creating contingency table by selecting a column
conTable = table(df$Name)
print(conTable)
```

Output:

```
Amiya Asish  Rosy
    1     1     1
```

From the output, you can notice that the table() command sorts the name in alphabetical order along with their frequencies of occurrence.

- **Using Rows of a Data Frame in a Contingency Table:** We can't create a contingency table using rows of a data frame directly as we did in "using column" part. With the help of the matrix, we can create a contingency table by looking at the rows of a data frame.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Creating contingency table by selecting rows
conTable = table(as.matrix(df[2:3, ]))
print(conTable)
```

Output:

```
Asish Female   Male   Rosy
  1      1      1      1
```

- **By Rotating Data Frames in R:** We can also create a contingency table by rotating a data frame in R. We can perform a rotation of the data, that is, transpose of the data using the **t()** command.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)
```

```
# Rotating the data frame
newDf = t(df)

# Creating contingency table by rotating data frame
conTable = table(newDf)
print(conTable)
```

Output:

```
newDf
Amiya  Asish Female   Male   Rosy
  1      1      1      2      1
```

- **Creating Contingency Tables from Matrix Objects in R** A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are two-dimensional, homogeneous data structures. We can create a contingency table by using this matrix object.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a matrix
A = matrix(
  c(1, 2, 4, 1, 5, 6, 2, 4, 7),
  nrow = 3,
  ncol = 3
)

# Creating contingency table using matrix object
conTable = table(A)
print(conTable)
```

Output:

```
A
1 2 4 5 6 7
2 2 2 1 1 1
```

Converting Objects into tables

As mentioned above, a table is a special type of data object which is similar to the matrix but also possesses several differences.

- **Converting Matrix Objects into tables:** We can directly convert a matrix object into table by using the **as.table()** command. Just pass the matrix object as a parameter to the **as.table()** command.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a matrix
A = matrix(
  c(1, 2, 4, 1, 5, 6, 2, 4, 7),
  nrow = 3,
  ncol = 3
)

# Converting Matrix Objects into tables
newTable = as.table(A)
print(newTable)
```

Output:

```
  A B C
A 1 1 2
B 2 5 4
C 4 6 7
```

- **Converting Data frame Objects into tables:** We can't directly convert a data frame object into table by using the **as.table()** command. In the case of a data frame, the object can be converted into the matrix, and then it can be converted into the table using **as.table()** command. **Example:**

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Converting data frame object to matrix object
modifiedDf = as.matrix(df)

# Converting Matrix Objects into tables
newTable = as.table(modifiedDf)
print(newTable)
```

Output:

```
  Name  Gender
A Amiya Male
B Rosy  Female
C Asish Male
```

One-way unstructured comparisons:

data from a one-way unstructured comparison between three treatments. The weights of the plants were measured after two months on respective treatments: water concentrated nutrient, and concentrated nutrient plus the selective herbicide 2,4-D.

```
Data are: tomato <- data.frame(weight= c(1.5, 1.9, 1.3, 1.5, 2.4, 1.5,      # water
                                          1.5, 1.2, 1.2, 2.1, 2.9, 1.6,     # Nutrient
                                         1.9, 1.6, 0.8, 1.15, 0.9, 1.6),    # Nutrient+24D
                                         trt = rep(c("water", "Nutrient", "Nutrient+24D"), c(6, 6, 6)))
```

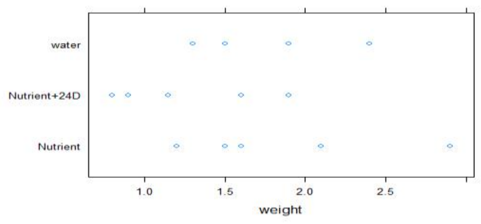


Figure 4.5 Weights of tomato plants, after two months of the three treatments.

## Make water the first level of trt. It will then appear as ## the initial level in the graphs. In aov or lm calculations, ## it will be the baseline or reference level.

tomato$trt <- relevel(tomato$trt, ref="water")

Figure 4.5 can be obtained with: stripplot(trt~weight, aspect=0.6, data=tomato)

The strip plots display "within-group" variability, as well as giving an indication of differ- ences among the group means. Variances seem similar for the three treatments.

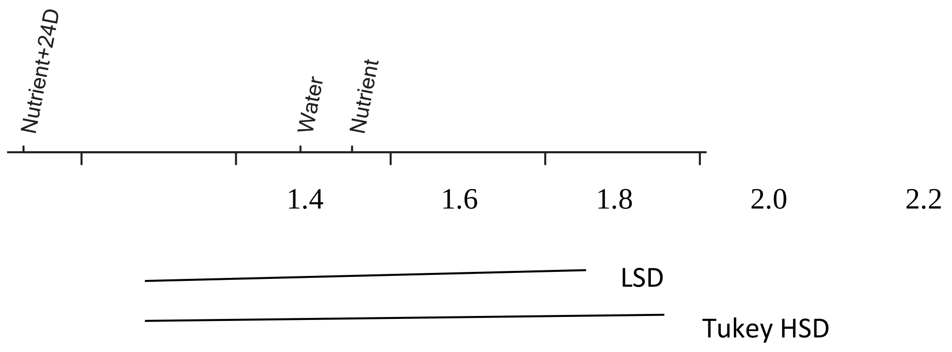
The tomato data–a visual summary

The function onewayPlot(), from the *DAAG* package, provides a convenient visual summary of results, shown in Figure 4.6. The code is:

## Do analysis of variance calculations tomato.aov <- aov(weight ˜ trt, data=tomato)##
Summarize results graphically oneway.plot(obj=tomato.aov)

Notice that the graph gives two different assessments of the least difference that should be treated as "significant". These use different criteria:

- The 5% least significant difference (LSD) is designed so that, under the null model (no differences), significant differences will be found in 5% of comparisons.
- The 5% honest significant difference (HSD) is designed so that, under the null model, the maximum difference will be significant in 5% of experiments.



The LSD is commonly regarded as overly lax, while the HSD may be overly conservative. There are a variety of alternatives to the HSD that are less conservative

The analysis of variance table

The analysis of variance table is given by the anova() function, thus:

```
> anova(tomato.aov)

 Analysis of Variance

 Table Response: weight

            Df       Sum Sq    Mean Sq    F value Pr(>F)

 trt        2 0.63    0.31       1.2        0.33

Residuals   15        3.91       0.26
```

Response curves:

data that are strongly structured. A model car was released three times at each of four different distances (starting.point) up a 20◦ ramp. The experimenter recorded distances traveled from the bottom of the ramp across a concrete floor.

Response curve analyses should be used whenever appropriate in preference to comparison of individual pairs of means. For these data, the physics can be used to suggest the likely form of response. Where no such help is available, careful examination of the graph, followed by systematic examination of plausible forms of response, may suggest a suitable form of response curve.

Table 4.8 Each tester made two firmness tests on each of five fruit.

Fruit	Tester	Firmness	Mean
1	1	6.8, 7.3	7.05
2	1	7.2, 7.3	7.25
3	1	7.4, 7.3	7.35
4	1	6.8, 7.6	7.2
5	1	7.2, 6.5	6.85
6	2	7.7, 7.7	7.7
7	2	7.4, 7.0	7.2
8	2	7.2, 7.6	7.4
9	2	6.7, 6.7	6.7
10	2	7.2, 6.8	7.0

Steps that are suitable for use with data that appear to follow a relatively simple form of response are:

1. Does a straight line explain the data better than assuming a random scatter about a horizontal line?

2. Does a quadratic response curve offer any improvement?

3. Would a cubic curve do better still?

A representation of the response curve in terms of coefficients of orthogonal polynomials provides information that makes it relatively easy to address questions 1–3. Consider, for example, a model that has terms in x and x2. Orthogonal polynomials re-express this combination of terms in such a way that the coefficient of the “linear” term is independent of the coefficient of the “quadratic” term. Higher-order (cubic, . . .) orthogonal polynomial terms can of course be fitted, and it remains the case that the coefficients are mutually independent.

Data with a nested variation structure:

Ten apples are taken from a box. A randomization procedure assigns five to one tester, and the other five to another tester. Each tester makes two firmness tests on each of their five fruit. Firmness is measured by the pressure needed to push the flat end of a piece of rod through the surface of the fruit.gives the results, in N/m2.

What happens if we ignore the data structure, and compare ten values for one tester with ten values for the other tester?

This pretends that we have ten experimental units for each tester. The analysis will suggest that the treatment means are more accurate than is really the case. We obtain a pretend standard error that is not the correct standard error of the mean. We are likely to under-estimate the standard error of the treatment difference.

Degrees of freedom considerations:

For comparison of two means when the sample sizes n1 and n2 are small, it is important to have as many degrees of freedom as possible for the denominator of the t-test. It is worth tolerating possible bias in some of the calculated SEDs in order to gain extra degrees of freedom.

The same considerations arise in the one-way analysis of variance, and we pursue the issue in that context. It is illuminating to plot out, side by side, say 10 SEDs based on randomly generated normal variates, first for a comparison based on 2 d.f., then 10 SEDs for a comparison based on 4 d.f., etc (d.f degree of freedom)

A formal statistical test is thus unlikely, unless the sample is large, to detect differences in variance that may have a large effect on the result of the test. It is therefore necessary to rely on judgment. Both past experience with similar data and subject area knowledge may be important.

In comparing two treatments that are qualitatively similar, differences in the population variance may be unlikely, unless the difference in means is at least of the same order of magnitude as the individual means.

If the means are not much different then it is reasonable, though this is by no means inevitable, to expect that the variances will not be much different.

If there do seem to be differences in variance, it may be possible to model the variance as a function of the mean. It may be possible to apply a variance-stabilizing transformation.

Otherwise, if there are just one or two degrees of freedom per mean, use a pooled estimate of variance unless the assumption of equal variance seems clearly unacceptable.

General multi-way analysis of variance designs:

Generalization to multi-way analysis of variance raises a variety of new issues. If each combination of factor levels has the same number of observations, and if there is no structure in the error (or noise), the extension is straightforward.

The extension is less straightforward when one or both of these conditions are not met. For unbalanced data from designs with a simple error structure, it is necessary to use the lm() (linear model) function.

The lme() function in the nlme package, or alternatively lmer() in the lme4 package, is able to handle problems where there is structure in the error term, including data from unbalanced designs.

Table 4.9 These are the same data as in Table 3.1.

	Pair #								
	1	2	3	4	5	6	7	8	9
Heated (mm)	244	255	253	254	251	269	248	252	292
Ambient	225	247	249	253	245	259	242	255	286
Difference	19	8	4	1	6	10	6	−3	6

Resampling methods for standard errors, tests, and confidence intervals:

Bootstrap resampling:

The bootstrap is mainly used in **estimation**. The method uses resampling with replacement to generates an approximate sampling distribution of an estimate. Standard errors and confidence intervals can be calculated using this bootstrap sampling distribution.

Resample data

Let’s assume that the data are a sample of measurements for a single variable stored in a vector x. The data may be numeric or categorical.

1. A single bootstrap replicate is obtained as follows. The replace option is used to indicate that sampling is carried out **with replacement**.

```
xboot <- sample(x, replace = TRUE)
```

2. Calculate the statistic of interest (for example, the mean) on the resampled data in xboot and store the result in a vector created for this purpose.

3.
```
z <- vector()       # initialize z (do only once)
```

```
z[1] <- mean(xboot)  # first bootstrap replicate estimate
```

4. Repeat steps (1) and (2) many times. The result will be a large collection of bootstrap replicate estimates for subsequent analysis.

In other cases, two or more variables are measured on individuals (e.g., stem height, leaf area, petal diameter, etc). Assume that each row of a data frame mydata is a different individual, and each column a different variable.

1. To resample individuals (i.e., rows),

2.
```
iboot <- sample(1:nrow(mydata), replace = TRUE))
```

```
bootdata <- mydata[iboot,]
```

The data frame bootdata will contain a single bootstrap replicate including all the variables.

3. Calculate the statistic of interest on the resampled data and store the result in vector created for this purpose. For example, to calculate the correlation between two variables x and y in bootdata,

```
4. z <- vector()        # initialize z (do only once)
```

```
z[1] <- cor(bootdata$x, bootdata$y)
```

5. Repeat steps (1) and (2) many times. The result will be a large collection of bootstrap replicate estimates for subsequent analysis.

Bootstrap standard error

Assume that the vector z contains a large number of bootstrap replicate estimates of a statistic of interest. The bootstrap standard error of the statistic is obtained as the **standard deviation** of the bootstrap replicate estimates. (The most common mistake at this stage is to calculate the standard error of the bootstrap replicates rather than the standard deviation.)

```
sd(z)
```

95% CI by percentile method

The percentile method is often used to provide an approximate 95% confidence interval for the population parameter. The percentile method is not as accurate as the BC_a method, but it is very quick and easy to calculate. Assume that the vector z contains a large number of bootstrap replicate estimates of the statistic of interest. By the percentile method, a 95% boostrap confidence interval for the parameter is obtained as

```
quantile(z, probs = c(0.025,0.975))
```

A large number of bootstrap replicate estimates is required for an accurate confidence interval.

Bootstrap estimate of bias

The bootstrap is often used to determine the bias of an estimate. Assume that the vector z contains a large number of bootstrap replicate estimates of a statistic of interest. Also, assume that the same statistic computed on the original data is stored in an object named “estimate”. The bootstrap estimate of bias is calculated as

```
mean(z) - estimate
```

Using the boot package

The boot library, included with the standard R installation, includes useful commands for bootstrapping. Notably, it will calculate confidence intervals using the BC_a method, which is more accurate than those produced by the percentile method. BC_a stands for “bias corrected and accelerated”. The method corrects for estimated bias and skewness. Consult Efron and Tibshirani (1998) for details.

To begin, load the boot library

```
library(boot)
```

Single variable

To use the boot package you will need to write a function to calculate the statistic of interest. The format is illustrated below for the sample mean, but any univariate function would be handled similarly. We'll call our function "boot.mean". When you have finished writing the script for a function you will need to cut and paste it into the command window so that R can access it (you'll need to do this just once in an R session). Here, x refers to the vector of data. i serves as a counter, as in your own for loop, but it must be included as an argument in your function as shown.

```
boot.mean <- function(x,i){boot.mean <- mean(x[i])}
```

The command boot will automatically carry out all the resampling and computations required. For this example, x is the vector of original data and boot.mean is the name of the function we created above to calculate the statistic of interest. R specifies the number of bootstrap replicate estimates desired.

```
z <- boot(x, boot.mean, R = 2000)
```

The resulting object (which here named z) is a **boot** object containing all the results. Use the following additional commands to pull out the results.

```
print(z)              # Bootstrap calculation of bias and SE
sd(z$t)               # Another way to get the standard error


hist(z$t)             # Histogram of boostrap replicate estimates
qqnorm(z$t)               # Normal quantiles of replicate estimates


boot.ci(z, type = "bca")  # 95% confidence interval using BCa
boot.ci(z, type = "perc") # Same using percentile method
```

Two or more variables

Here's how you would use the boot command If the statistic of interest must be calculated from two (or more) variables (for example, a correlation, regression slope, or odds ratio). Assume that there are two variables of interest, x and y. If not done already, put the two variables into a data frame (here called mydata),

```
mydata <- cbind.data.frame(x, y, stringsAsFactors = FALSE)
```

Then create a function to calculate the statistic of interest on the variables in mydata. For example, to create a function that calculates the correlation coefficient between the two variables x and y, use

```
boot.cor <- function(mydata,i){
  x <- mydata$x[i]
  y <- mydata$y[i]
  boot.cor <- cor(x,y)
  }
```

Here, i refers to a vector of indices, which must be included as an argument in the function and employed as shown.

Finally, pass your data frame and function to the boot command,

```
z <- boot(mydata, boot.cor, R = 2000)
```

See the previous section for a list of commands to pull results from the boot object (here named z).

Permutation test

A permutation test uses resampling and the computer to generate a null distribution for a test statistic. The test statistic is a measure of association between two variables or difference between groups, such as a slope, a correlation, or an odds ratio. Each permutation step involves randomly resampling **without replacement** the values of one of the two variables in the data and recalculating the test statistic in each permutation. The two variables may be categorical (character or factor), numeric, or one of each.

Categorical data

R has a built-in permutation procedure for a contingency test of association when both of two variables are categorical (call them A1 and A2). To apply it, execute the usual command for the χ^2 contingency test, but set the simulate.p.value option to TRUE. The number of replicates in the permutation is set by the option B (default is 2000). Each permutation rearranges the values in the contingency table while keeping all the row and column totals fixed to their observed values.

```
chisq.test(A1, A2, simulate.p.value = TRUE, B = 5000)
```

Numeric data

If one or both of the variables is numeric, then you will need to create a short loop to carry out the resampling necessary for the permutation test. Choose one of the two variables to resample (call it x). It doesn't matter which of the two variables you choose. Keep the other variable (call it y) unchanged (there is no benefit to resampling both variables).

1. Resample x without replacement to create a new vector (call it x1).

```
x1 <- sample(x, replace = FALSE)
```

2. Calculate the test statistic to measure association between y and the randomized variable x1. Store the result in a vector created for this purpose. For example, to calculate the correlation between the two variables,

3.
```
z <- vector()       # initialize z (do only once)
```

```
z[1] <- cor(x1, y)   # first permutation result
```

4. Repeat steps (1) and (2) many times. The result will be a large collection of replicates representing the null distribution of your test statistic.

Theories of inference:

Formal statistical methodologies are of two broad types:

frequentist and

Bayesian.

Frequentist: The frequentist approach is usually based on the concept of likelihood; given the model, what is the probability of obtaining a sample similar to that observed? Parameter values are 16

```
## Bootstrap estimate of 95% CI of cor(chest, belly):
 data frame possum (DAAG)
 possum.fun <- function(data, indices) {
chest <- data$chest[indices]
belly <- data$belly[indices]
cor(belly, chest)
}
possum.boot <- boot(possum, possum.fun, R=9999)
 boot.ci(possum.boot, type=c("perc", "bca"))
```

assumed to be unknown constants, and estimates are chosen to maximize this likelihood.

Another type of methodology, broadly known as "Bayesian" uses Bayes' theorem. The essential idea is that we might have prior information (knowledge or belief) about the distribution of a parameter value before taking a sample of observations. This prior information can be updated using the sample and the rules of probability.

Maximum likelihood estimation :
Consider the model
$y_i = \mu + \varepsilon_i, i = 1, 2,...,n$
where μ is an unknown constant,
and where the errors ε are assumed to be independent and normally distributed with mean 0 and variance σ2. The probability density for the ith y-value is normal with mean μ and variance σ2.
Because of the independence assumption, the probability density of the entire sample of ys is simply the product of these normal densities.
This product is the likelihood. The maximum likelihood estimates are the values of μ and σ which maximize this function. A calculus argument can be used to see that the estimates are $\bar{y}$ and $s \sqrt{(n-1)/n}$.
For an example,
consider the observed differences between heated and ambient,
assuming an independent normal errors model:

```
funlik <- function(mu, sigma, x=with(pair65, heated-ambient))
prod(dnorm(x, mu, sigma))
```

In practice, it is more convenient to work with the log likelihood, rather than the likelihood. Maximizing on the log scale leads to exactly the same estimates as on the original scale.
Try the following:

```
 > log(funlik(6.3, 6.1)) # Close to estimated mean and SD
 [1] -28.549
> log(funlik(6.33, 5.75)) # Close to the actual mle's
[1] -28.520
> log(funlik(7, 5.75))
[1] -28.580
```

<u>Bayesian estimation :</u>
the Bayesian methodology provides a way to update our prior information about the model parameters using sample information.
the prior information is summarized in the form of a probability law called the prior distribution of the model parameters. Interest usually centers on the posterior distribution of the parameters, which is proportional to the product of the likelihood and the prior distribution.
A simple application of Bayes' theorem is as follows. The incidence of HIV in adult Australian males (15–49 years) who do not have any known risk factor may be of the order of 1 in 100 000, i.e., the prior probability of infection is 0.00001. A person in this group has an initial test (for example, it may be required in order to obtain a US green card) that has a specificity of 0.01%, i.e., for every 10 000 people tested, there will on average be one false positive.

How should such an individual interpret the result? If 100 000 individuals take the test, one will on average have AIDS and will almost certainly return a positive test. On the other hand there will, on average, be close to 10 false positives (0.1% of 99 999).

Not infected	Infected
10000 × 0.001 = 10 (false) positives	1 true positive

The posterior odds that the person has AIDS are thus close to 1:10, certainly a narrowing from the prior odds of 1:99 999.
Note that, as often happens when Bayesian calculations are used, the prior information is not very precise. What we can say is that the prior probability is, in the case mentioned, very low.

Bayesian estimation with normal prior and normal likelihood:
A relatively simple example is that of a normal likelihood (as considered in the previous section) where the unobserved true mean is now also assumed to have a normal distribution, but this time with mean $\mu 0$ and variance $\sigma 2\ 0$.

The posterior density of the mean is then normal with
mean

$$\frac{n\bar{y} + \mu_0\sigma^2/\sigma^2_0}{n + \sigma^2/\sigma^2_0}$$

variance

$$\frac{\sigma^2}{n + \sigma^2/\sigma^2_0} .$$

This assumes that $\sigma 2$ is actually known; an estimate can be obtained using the sample variance. Alternatively, we could put a prior distribution on this parameter as well.
If there is strong prior information, use it:
Any methodology that ignores strong prior information is inappropriate, and may be highly misleading. Diagnostic testing (the AIDS test example mentioned above) and criminal investigations provide cogent examples.
Using the hypothesis testing framework, we take the null hypothesis H0, in the AIDS test example, to be the hypothesis that the individual does not have HIV.
Given this null hypothesis, the probability of a positive result is 0.0001. Therefore the null hypothesis is rejected.

Scrutiny of 10 000 potential perpetrators will on average net 10 suspects. Suppose one of these is later charged. The probability of such incriminating evidence, assuming that the defendant is innocent, is indeed 0.001.
The police screening will net around 10 innocent people along with, perhaps, the one perpetrator. The following summarizes the expected result of the police search for a suspect.
It is optimistic in its assumption that the perpetrator will be among those netted.

Not the perpetrator	The perpetrator
10000 × 0.001 = 10 (false) positives	1 true positive

This evidence leads to odds of 1:10 or worse, i.e., less than 10%, that the defendant is guilty.

Regression with a single predictor:
Linear regression (or **linear model**) is used to predict a quantitative outcome variable (y) on the basis of one or multiple predictor variables (x) (James et al. 2014,P. Bruce and Bruce (2017)).
The goal is to build a mathematical formula that defines y as a function of the x variable. Once, we built a statistically significant model, it's possible to use it for predicting future outcome on the basis of new x values.

When you build a regression model, you need to assess the performance of the predictive model. In other words, you need to evaluate how well the model is in predicting the outcome of a new test data that have not been used to build the model.

Two important metrics are commonly used to assess the performance of the predictive regression model:

- **Root Mean Squared Error**, which measures the model prediction error. It corresponds to the average difference between the observed known values of the outcome and the predicted value by the model. RMSE is computed as RMSE = mean((observeds - predicteds)^2) %>% sqrt(). The lower the RMSE, the better the model.
- **R-square**, representing the squared correlation between the observed known outcome values and the predicted values by the model. The higher the R2, the better the model.

A simple workflow to build to build a predictive regression model is as follow:

1. Randomly split your data into training set (80%) and test set (20%)
2. Build the regression model using the training set
3. Make predictions using the test set and compute the model accuracy metrics

Formula

The mathematical formula of the linear regression can be written as follow:

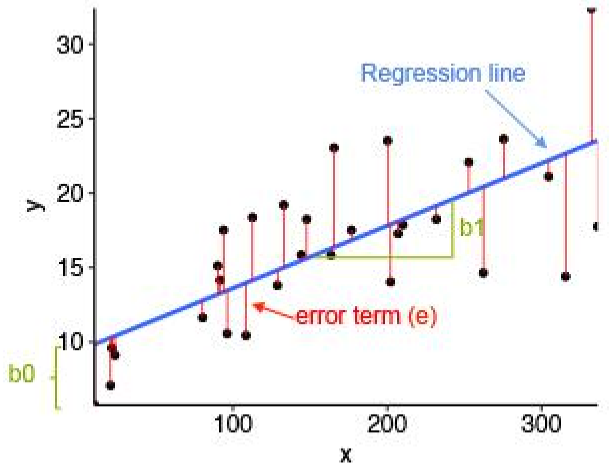
y = b0 + b1*x + e
We read this as “y is modeled as beta1 (b1) times x, plus a constant beta0 (b0), plus an error term e.”
When you have multiple predictor variables, the equation can be written as y = b0 + b1*x1 + b2*x2 + ... + bn*xn, where:

- b0 is the intercept,
- b1, b2, …, bn are the regression weights or coefficients associated with the predictors x1, x2, …, xn.
- e is the *error term* (also known as the *residual errors*), the part of y that can be explained by the regression model

Note that, b0, b1, b2, … and bn are known as the regression beta coefficients or parameters.

The figure below illustrates a simple linear regression model, where:

- the best-fit regression line is in blue
- the intercept (b0) and the slope (b1) are shown in green
- the error terms (e) are represented by vertical red lines



From the scatter plot above, it can be seen that not all the data points fall exactly on the fitted regression line. Some of the points are above the blue curve and some are below it; overall, the residual errors (e) have approximately mean zero.

The sum of the squares of the residual errors are called the **Residual Sum of Squares** or **RSS**.
The average variation of points around the fitted regression line is called the **Residual Standard Error** (**RSE**). This is one the metrics used to evaluate the overall quality of the fitted regression model. The lower the RSE, the better it is.
Since the mean error term is zero, the outcome variable y can be approximately estimated as follow:

y ~ b0 + b1*x
Mathematically, the beta coefficients (b0 and b1) are determined so that the RSS is as minimal as possible. This method of determining the beta coefficients is technically called **least squares** regression or **ordinary least squares** (OLS) regression.
Once, the beta coefficients are calculated, a t-test is performed to check whether or not these coefficients are significantly different from zero. A non-zero beta coefficients means that there is a significant relationship between the predictors (x) and the outcome variable (y).

Loading Required R packages

- tidyverse for easy data manipulation and visualization
- caret for easy machine learning workflow

```
library(tidyverse)
library(caret)
theme_set(theme_bw())
```

Preparing the data

We'll use the marketing data set, introduced in the Chapter @ref(regression-analysis), for predicting sales units on the basis of the amount of money spent in the three advertising medias (youtube, facebook and newspaper)
We'll randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). Make sure to set seed for reproducibility.

```
# Load the data
data("marketing", package = "datarium")
# Inspect the data
sample_n(marketing, 3)
##     youtube facebook newspaper sales
## 58    163.4     23.0      19.9  15.8
## 157   112.7     52.2      60.6  18.4
## 81     91.7     32.0      26.8  14.2
# Split the data into training and test set
set.seed(123)
training.samples <- marketing$sales %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- marketing[training.samples, ]
test.data <- marketing[-training.samples, ]
```

Computing linear regression

The R function lm() is used to compute linear regression model.

Quick start R code

```
# Build the model
model <- lm(sales ~., data = train.data)
# Summarize the model
summary(model)
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance
# (a) Prediction error, RMSE
RMSE(predictions, test.data$sales)
# (b) R-square
R2(predictions, test.data$sales)
```

Simple linear regression

The **simple linear regression** is used to predict a continuous outcome variable (y) based on one single predictor variable (x).
In the following example, we'll build a simple linear model to predict sales units based on the advertising budget spent on youtube. The regression equation can be written as sales = b0 + b1*youtube.
The R function lm() can be used to determine the beta coefficients of the linear model, as follow:

```
model <- lm(sales ~ youtube, data = train.data)
summary(model)$coef
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.3839    0.62442    13.4 5.22e-28
## youtube       0.0468    0.00301    15.6 7.84e-34
```

The output above shows the estimate of the regression beta coefficients (column Estimate) and their significance levels (column Pr(>|t|). The intercept (b0) is 8.38 and the coefficient of youtube variable is 0.046.

The estimated regression equation can be written as follow: sales = 8.38 + 0.046*youtube. Using this formula, for each new youtube advertising budget, you can predict the number of sale units.

For example:

- For a youtube advertising budget equal zero, we can expect a sale of 8.38 units.
- For a youtube advertising budget equal 1000, we can expect a sale of 8.38 + 0.046*1000 = 55 units.

Predictions can be easily made using the R function predict(). In the following example, we predict sales units for two youtube advertising budget: 0 and 1000.

```
newdata <- data.frame(youtube = c(0,  1000))
model %>% predict(newdata)
##     1     2
##  8.38 55.19
```

multiple linear regressions:

Multiple linear regression is an extension of simple linear regression for predicting an outcome variable (y) on the basis of multiple distinct predictor variables (x).

For example, with three predictor variables (x), the prediction of y is expressed by the following equation: y = b0 + b1*x1 + b2*x2 + b3*x3

The regression beta coefficients measure the association between each predictor variable and the outcome. “b_j” can be interpreted as the average effect on y of a one unit increase in “x_j”, holding all other predictors fixed.

In this section, we’ll build a multiple regression model to predict sales based on the budget invested in three advertising medias: youtube, facebook and newspaper. The formula is as follow: sales = b0 + b1*youtube + b2*facebook + b3*newspaper

You can compute the multiple regression model coefficients in R as follow:

```
model <- lm(sales ~ youtube + facebook + newspaper,
            data = train.data)
summary(model)$coef
```

Note that, if you have many predictor variables in your data, you can simply include all the available variables in the model using ~.:

```
model <- lm(sales ~., data = train.data)
summary(model)$coef
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.39188    0.44062   7.698 1.41e-12
## youtube      0.04557    0.00159  28.630 2.03e-64
## facebook     0.18694    0.00989  18.905 2.07e-42
## newspaper    0.00179    0.00677   0.264 7.92e-01
```

From the output above, the coefficients table shows the beta coefficient estimates and their significance levels. Columns are:

- Estimate: the intercept (b0) and the beta coefficient estimates associated to each predictor variable
- Std.Error: the standard error of the coefficient estimates. This represents the accuracy of the coefficients. The larger the standard error, the less confident we are about the estimate.
- t value: the t-statistic, which is the coefficient estimate (column 2) divided by the standard error of the estimate (column 3)
- Pr(>|t|): The p-value corresponding to the t-statistic. The smaller the p-value, the more significant the estimate is.

As previously described, you can easily make predictions using the R function predict():

```
# New advertising budgets
newdata <- data.frame(
  youtube = 2000, facebook = 1000,
  newspaper = 1000
)
# Predict sales values
```

```
model %>% predict(newdata)
##   1
## 283
```

Interpretation

Before using a model for predictions, you need to assess the statistical significance of the model. This can be easily checked by displaying the statistical summary of the model.

Model summary

Display the statistical summary of the model as follow:

```
summary(model)
##
## Call:
## lm(formula = sales ~ ., data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.412  -1.110   0.348   1.422   3.499
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.39188    0.44062    7.70  1.4e-12 ***
## youtube      0.04557    0.00159   28.63  < 2e-16 ***
## facebook     0.18694    0.00989   18.90  < 2e-16 ***
## newspaper    0.00179    0.00677    0.26     0.79
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.12 on 158 degrees of freedom
## Multiple R-squared:  0.89,   Adjusted R-squared:  0.888
## F-statistic:  427 on 3 and 158 DF,  p-value: <2e-16
```

The summary outputs shows 6 components, including:

- **Call**. Shows the function call used to compute the regression model.
- **Residuals**. Provide a quick view of the distribution of the residuals, which by definition have a mean zero. Therefore, the median should not be far from zero, and the minimum and maximum should be roughly equal in absolute value.
- **Coefficients**. Shows the regression beta coefficients and their statistical significance. Predictor variables, that are significantly associated to the outcome variable, are marked by stars.
- **Residual standard error** (RSE), **R-squared** (R2) and the **F-statistic** are metrics that are used to check how well the model fits to our data.

The first step in interpreting the multiple regression analysis is to examine the F-statistic and the associated p-value, at the bottom of model summary.

In our example, it can be seen that p-value of the F-statistic is < 2.2e-16, which is highly significant. This means that, at least, one of the predictor variables is significantly related to the outcome variable.

Coefficients significance

To see which predictor variables are significant, you can examine the coefficients table, which shows the estimate of regression beta coefficients and the associated t-statistic p-values.

```
summary(model)$coef
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.39188    0.44062   7.698 1.41e-12
## youtube      0.04557    0.00159  28.630 2.03e-64
## facebook     0.18694    0.00989  18.905 2.07e-42
## newspaper    0.00179    0.00677   0.264 7.92e-01
```

For a given the predictor, the t-statistic evaluates whether or not there is significant association between the predictor and the outcome variable, that is whether the beta coefficient of the predictor is significantly different from zero.

It can be seen that, changing in youtube and facebook advertising budget are significantly associated to changes in sales while changes in newspaper budget is not significantly associated with sales.

For a given predictor variable, the coefficient (b) can be interpreted as the average effect on y of a one unit increase in predictor, holding all other predictors fixed.

For example, for a fixed amount of youtube and newspaper advertising budget, spending an additional 1 000 dollars on facebook advertising leads to an increase in sales by approximately 0.1885*1000 = 189 sale units, on average.

The youtube coefficient suggests that for every 1 000 dollars increase in youtube advertising budget, holding all other predictors constant, we can expect an increase of 0.045*1000 = 45 sales units, on average.

We found that newspaper is not significant in the multiple regression model. This means that, for a fixed amount of youtube and newspaper advertising budget, changes in the newspaper advertising budget will not significantly affect sales units.

As the newspaper variable is not significant, it is possible to remove it from the model:

```
model <- lm(sales ~ youtube + facebook, data = train.data)
summary(model)
##
## Call:
## lm(formula = sales ~ youtube + facebook, data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.481  -1.104   0.349   1.423   3.486
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.43446    0.40877     8.4  2.3e-14 ***
## youtube      0.04558    0.00159    28.7  < 2e-16 ***
## facebook     0.18788    0.00920    20.4  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.11 on 159 degrees of freedom
## Multiple R-squared:  0.89,   Adjusted R-squared:  0.889
## F-statistic:  644 on 2 and 159 DF,  p-value: <2e-16
```

Finally, our model equation can be written as follow: sales = 3.43+ 0.045*youtube + 0.187*facebook.

Model accuracy

Once you identified that, at least, one predictor variable is significantly associated to the outcome, you should continue the diagnostic by checking how well the model fits the data. This process is also referred to as the *goodness-of-fit*
The overall quality of the linear regression fit can be assessed using the following three quantities, displayed in the model summary:

1. Residual Standard Error (RSE),

2. R-squared (R2) and adjusted R2,
3. F-statistic, which has been already described in the previous section

```
##    rse r.squared f.statistic  p.value
## 1 2.11      0.89         644 5.64e-77
```

1. **Residual standard error** (RSE).

The RSE (or model *sigma*), corresponding to the prediction error, represents roughly the average difference between the observed outcome values and the predicted values by the model. The lower the RSE the best the model fits to our data. Dividing the RSE by the average value of the outcome variable will give you the prediction error rate, which should be as small as possible.

In our example, using only youtube and facebook predictor variables, the RSE = 2.11, meaning that the observed sales values deviate from the predicted values by approximately 2.11 units in average.

This corresponds to an error rate of 2.11/mean(train.data$sales) = 2.11/16.77 = 13%, which is low.

2. **R-squared and Adjusted R-squared**:

The R-squared (R2) ranges from 0 to 1 and represents the proportion of variation in the outcome variable that can be explained by the model predictor variables.

For a simple linear regression, R2 is the square of the Pearson correlation coefficient between the outcome and the predictor variables. In multiple linear regression, the R2 represents the correlation coefficient between the observed outcome values and the predicted values.

The R2 measures, how well the model fits the data. The higher the R2, the better the model. However, a problem with the R2, is that, it will always increase when more variables are added to the model, even if those variables are only weakly associated with the outcome (James et al. 2014). A solution is to adjust the R2 by taking into account the number of predictor variables.
The adjustment in the "Adjusted R Square" value in the summary output is a correction for the number of x variables included in the predictive model.

So, you should mainly consider the adjusted R-squared, which is a penalized R2 for a higher number of predictors.

- An (adjusted) R2 that is close to 1 indicates that a large proportion of the variability in the outcome has been explained by the regression model.
- A number near 0 indicates that the regression model did not explain much of the variability in the outcome.

In our example, the adjusted R2 is 0.88, which is good.

3. **F-Statistic**:

Recall that, the F-statistic gives the overall significance of the model. It assess whether at least one predictor variable has a non-zero coefficient.

In a simple linear regression, this test is not really interesting since it just duplicates the information given by the t-test, available in the coefficient table.

The F-statistic becomes more important once we start using multiple predictors as in multiple linear regression.

A large F-statistic will corresponds to a statistically significant p-value ($p < 0.05$). In our example, the F-statistic equal 644 producing a p-value of 1.46e-42, which is highly significant.

Making predictions

We'll make predictions using the test data in order to evaluate the performance of our regression model.

The procedure is as follow:

1. Predict the sales values based on new advertising budgets in the test data
2. Assess the model performance by computing:
   - The prediction error RMSE (Root Mean Squared Error), representing the average difference between the observed known outcome values in the test data and the predicted outcome values by the model. The lower the RMSE, the better the model.

- The R-square (R2), representing the correlation between the observed outcome values and the predicted outcome values. The higher the R2, the better the model.

```
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance
# (a) Compute the prediction error, RMSE
RMSE(predictions, test.data$sales)
## [1] 1.58
# (b) Compute R-square
R2(predictions, test.data$sales)
## [1] 0.938
```

From the output above, the R2 is 0.93, meaning that the observed and the predicted outcome values are highly correlated, which is very good.

The prediction error RMSE is 1.58, representing an error rate of 1.58/mean(test.data$sales) = 1.58/17 = 9.2%, which is good.

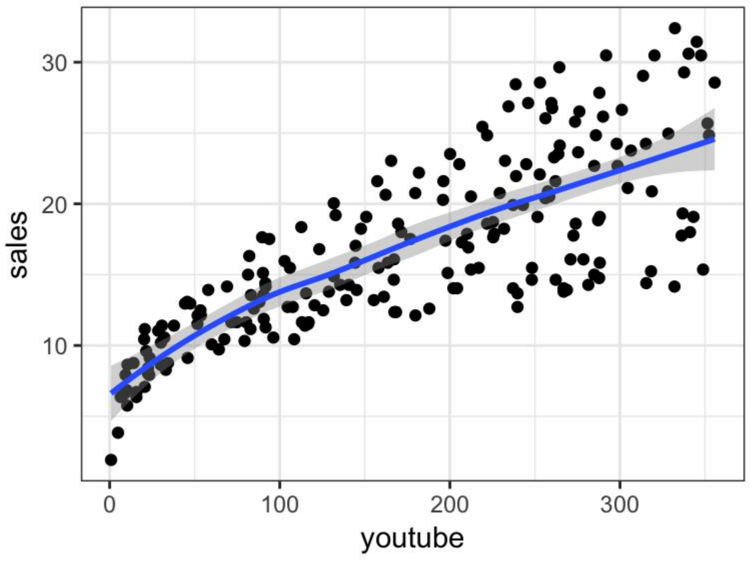
Discussion

This chapter describes the basics of linear regression and provides practical examples in R for computing simple and multiple linear regression models. We also described how to assess the performance of the model for predictions.

Note that, linear regression assumes a linear relationship between the outcome and the predictor variables. This can be easily checked by creating a scatter plot of the outcome variable vs the predictor variable.

For example, the following R code displays sales units versus youtube advertising budget. We'll also add a smoothed line:

```
ggplot(marketing, aes(x = youtube, y = sales)) +
  geom_point() +
  stat_smooth()
```



The graph above shows a linearly increasing relationship between the sales and the youtube variables, which is a good thing.

BLUE Property assumptions

- The Gauss Markov theorem tells us that if a certain set of assumptions are met, the ordinary least squares estimate for regression coefficients gives you the Best Linear Unbiased Estimate (BLUE) possible.

- There are five Gauss Markov assumptions (also called conditions):

- **Linearity**:
    - The parameters we are estimating using the OLS method must be themselves linear.
- **Random**:
    - Our data must have been randomly sampled from the population.
- **Non-Collinearity:**
    - The regressors being calculated aren’t perfectly correlated with each other.
- **Exogeneity**:
    - The regressors aren’t correlated with the error term.
- **Homoscedasticity:**
    - No matter what the values of our regressors might be, the error of the variance is constant.

Purpose of the Assumptions

- The Gauss Markov assumptions guarantee the validity of ordinary least squares for estimating regression coefficients.

- Checking how well our data matches these assumptions is an important part of estimating regression coefficients.

- When you know where these conditions are violated, you may be able to plan ways to change your experiment setup to help your situation fit the ideal Gauss Markov situation more closely.

- In practice, the Gauss Markov assumptions are rarely all met perfectly, but they are still useful as a benchmark, and because they show us what ‘ideal’ conditions would be.

- They also allow us to pinpoint problem areas that might cause our estimated regression coefficients to be inaccurate or even unusable.

The Gauss-Markov Assumptions in Algebra

- We can summarize the Gauss-Markov Assumptions succinctly in algebra, by saying that a linear regression model represented by

$$y_i = x_i{}' \beta + \varepsilon_i$$

- and generated by the ordinary least squares estimate is the best linear unbiased estimate (BLUE) possible if
    - $E\{\varepsilon_i\} = 0, i = 1, \ldots, N$
    - $\{\varepsilon_1 \ldots\ldots \varepsilon_n\}$ and $\{x_1 \ldots.., x_N\}$ are independent
    - $\text{cov}\{\varepsilon_i, \varepsilon_j\} = 0, i, j = 1, \ldots., N \; I \neq j.$
    - $V\{\varepsilon_1 = \sigma^2, i = 1, \ldots.N$

- The first of these assumptions can be read as “The expected value of the error term is zero.”. The second assumption is collinearity, the third is exogeneity, and the fourth is homoscedasticity.

Regression Concepts

Regression

- It is a Predictive modeling technique where the target variable to be estimated is continuous.

Examples of applications of regression

- predicting a stock market index using other economic indicators
- forecasting the amount of precipitation in a region based on characteristics of the jet stream
- projecting the total sales of a company based on the amount spent for advertising
- estimating the age of a fossil according to the amount of carbon-14 left in the organic material.

- Let D denote a data set that contains N observations,

$$D = \{(\mathbf{x}_i, y_i)|\ i = 1, 2, \ldots, N\}.$$

- Each x_i corresponds to the set of attributes of the i_{th} observation (known as explanatory variables) and y_i corresponds to the target (or response) variable.
- The explanatory attributes of a regression task can be either discrete or continuous.

Regression (Definition)

- Regression is the task of learning a target function *f* that maps each attribute set *x* into a continuous-valued output *y*.

The goal of regression

- To find a target function that can fit the input data with minimum error.
- The error function for a regression task can be expressed in terms of the sum of absolute or squared error:

$$\text{Absolute Error} = \sum_i |y_i - f(\mathbf{x}_i)| \qquad (D.1)$$

$$\text{Squared Error} = \sum_i (y_i - f(\mathbf{x}_i))^2 \qquad (D.2)$$

Simple Linear Regression

- Consider the physiological data shown in Figure D.1.
- The data corresponds to measurements of heat flux and skin temperature of a person during sleep.
- Suppose we are interested in predicting the skin temperature of a person based on the heat flux measurements generated by a heat sensor.
- The two-dimensional scatter plot shows that there is a strong linear relationship between the two variables.

Heat Flux	Skin Temperature
10.858	31.002
10.617	31.021
10.183	31.058
9.7003	31.095
9.652	31.133
10.086	31.188
9.459	31.226
8.3972	31.263
7.6251	31.319
7.1907	31.356
7.046	31.412
6.9494	31.468
6.7081	31.524
6.3221	31.581
6.0325	31.618
5.7429	31.674
5.5016	31.712
5.2603	31.768
5.1638	31.825
5.0673	31.862
4.9708	31.919
4.8743	31.975
4.7777	32.013
4.7295	32.07
4.633	32.126
4.4882	32.164
4.3917	32.221
4.2951	32.259
4.2469	32.296
4.0056	32.334
3.716	32.391
3.523	32.448
3.4265	32.505
3.3782	32.543
3.4265	32.6
3.3782	32.657
3.3299	32.696
3.3299	32.753
3.4265	32.791

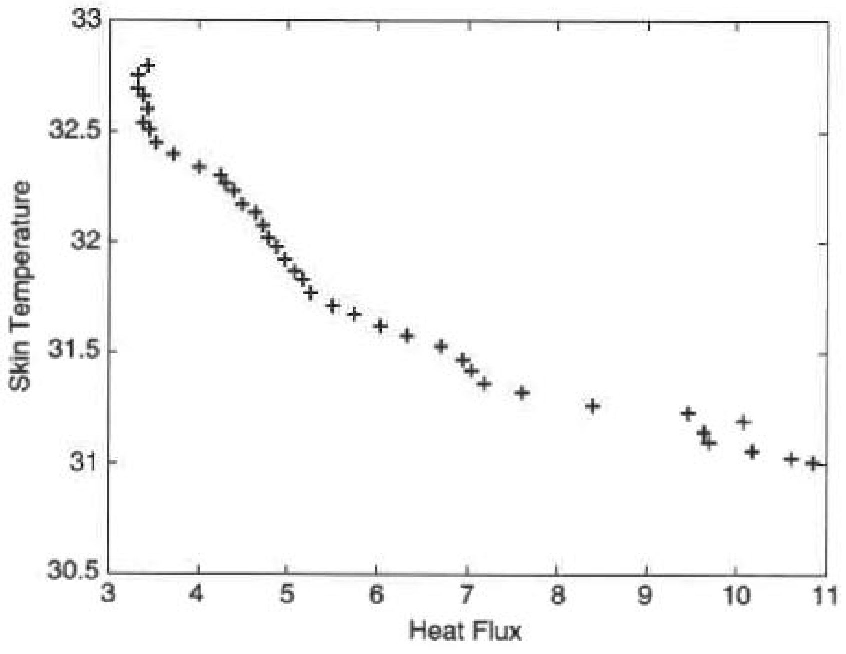


Figure D.1. Measurements of heat flux and skin temperature of a person.

Least Square Estimation or Least Square Method

- Suppose we wish to fit the following linear model to the observed data:

$$f(x) = \omega_1 x + \omega_0, \qquad \text{(D.3)}$$

- where w0 and w1 are parameters of the model and are called the regression coefficients.
- A standard approach for doing this is to apply the method of least squares, which attempts to find the parameters (w0,w1) that minimize the sum of the squared error

$$SSE = \sum_{i=1}^{N}[y_i - f(x_i)]^2 = \sum_{i=1}^{N}[y_i - \omega_1 x - \omega_0]^2, \qquad \text{(D.4)}$$

- which is also known as the residual sum of squares.
- This optimization problem can be solved by taking the partial derivative of E with respect to w0 and w1, setting them to zero, and solving the corresponding system of linear equations.

$$\begin{aligned}\frac{\partial E}{\partial \omega_0} &= -2\sum_{i=1}^{N}[y_i - \omega_1 x_i - \omega_0] = 0\\ \frac{\partial E}{\partial \omega_1} &= -2\sum_{i=1}^{N}[y_i - \omega_1 x_i - \omega_0]x_i = 0\end{aligned} \qquad \text{(D.5)}$$

- These equations can be summarized by the following matrix equation' which is also known as the normal equation:

$$\begin{pmatrix} N & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{pmatrix}\begin{pmatrix} \omega_0 \\ \omega_1 \end{pmatrix} = \begin{pmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{pmatrix}. \qquad \text{(D.6)}$$

- Since

$$\sum_i x_i = 229.9,\ \sum_i x_i^2 = 1569.2,\ \sum_i y_i = 1242.9, \text{ and } \sum_i x_i y_i = 7279.7,$$

- the normal equations can be solved to obtain the following estimates for the parameters.

$$\begin{aligned}\begin{pmatrix}\hat{\omega}_0\\ \hat{\omega}_1\end{pmatrix} &= \begin{pmatrix}39 & 229.9\\ 229.9 & 1569.2\end{pmatrix}^{-1}\begin{pmatrix}1242.9\\ 7279.7\end{pmatrix}\\ &= \begin{pmatrix}0.1881 & -0.0276\\ -0.0276 & 0.0047\end{pmatrix}\begin{pmatrix}1242.9\\ 7279.7\end{pmatrix}\\ &= \begin{pmatrix}33.1699\\ -0.2208\end{pmatrix}\end{aligned}$$

- Thus, the linear model that best fits the data in terms of minimizing the SSE is

$$f(x) = 33.17 - 0.22x.$$

- Figure D.2 shows the line corresponding to this model.

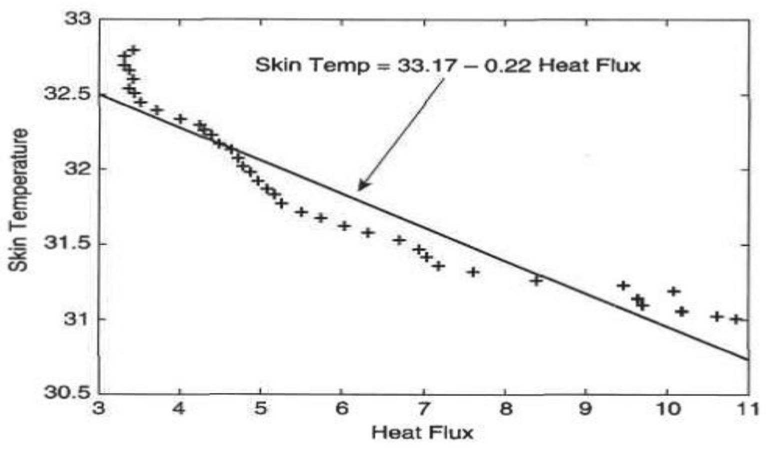


Figure D.2. A linear model that fits the data given in Figure D.1.

- We can show that the general solution to the normal equations given in D.6 can be expressed as follow:

$$\begin{aligned}\hat{\omega}_0 &= \overline{y} - \hat{\omega}_1\overline{x}\\ \hat{\omega}_1 &= \frac{\sigma_{xy}}{\sigma_{xx}}\end{aligned} \tag{D.7}$$

where $\overline{x} = \sum_i x_i/N$, $\overline{y} = \sum_i y_i/N$, and

$$\sigma_{xy} = \sum_i (x_i - \overline{x})(y_i - \overline{y}) \tag{D.8}$$

$$\sigma_{xx} = \sum_i (x_i - \overline{x})^2 \tag{D.9}$$

$$\sigma_{yy} = \sum_i (y_i - \overline{y})^2 \tag{D.10}$$

- Thus, linear model that results in the minimum squared error is given by

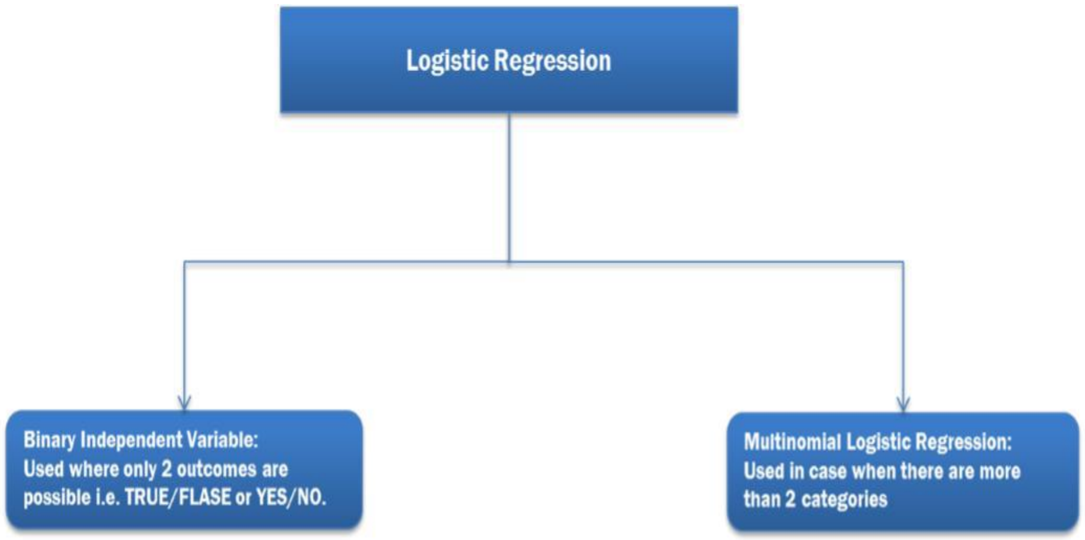
$$f(x) = \overline{y} + \frac{\sigma_{xy}}{\sigma_{xx}}[x - \overline{x}]. \qquad (D.11)$$

- In summary, the least squares method is a systematic approach to fit a linear model to the response variable g by minimizing the squared error between the true and estimated value of g.
- Although the model is relatively simple, it seems to provide a reasonably accurate approximation because a linear model is the first-order Taylor series approximation for any function with continuous derivatives.

Logistic Regression

Logistic regression, or Logit regression, or Logit model

- is a regression model where the dependent variable (DV) is categorical.
- was developed by statistician David Cox in 1958.


Logistic Regression
Binary Independent Variable: Used where only 2 outcomes are possible i.e. TRUE/FLASE or YES/NO.
Multinomial Logistic Regression: Used in case when there are more than 2 categories

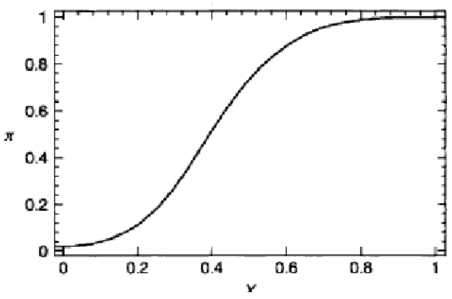
- The response variable Y has been regarded as a continuous quantitative variable.
- There are situations, however, where the response variable is qualitative.
- The predictor variables, however, have been both quantitative, as well as qualitative.
- Indicator variables fall into the second category.

- Consider a procedure in which individuals are selected on the basis of their scores in a battery of tests.
- After five years the candidates are classified as "good" or "poor.”
- We are interested in examining the ability of the tests to predict the job performance of the candidates.
- Here the response variable, performance, is dichotomous.
- We can code "good" as 1 and "poor" as 0, for example.
- The predictor variables are the scores in the tests.

- In a study to determine the risk factors for cancer, health records of several people were studied.
- Data were collected on several variables, such as age, gender, smoking, diet, and the family's medical history.
- The response variable was the person had cancer (Y = 1) or did not have cancer (Y = 0).

- The relationship between the probability π and X can often be represented by a logistic response function.
- It resembles an S-shaped curve.
- The probability π initially increases slowly with increase in X, and then the increase accelerates, finally stabilizes, but does not increase beyond 1.
- Intuitively this makes sense.
- Consider the probability of a questionnaire being returned as a function of cash reward, or the probability of passing a test as a function of the time put in studying for it.
- The shape of the S-curve can be reproduced if we model the probabilities as follows:

$$\pi = \Pr(Y = 1 | X = x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} ,$$



- A sigmoid function is a bounded differentiable real function that is defined for all real input values and has a positive derivative at each point.

- It has an “S” shape. It is defined by below function:

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

- The process of linearization of logistic regression function is called Logit Transformation.

$$\ln\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

- Modeling the response probabilities by the logistic distribution and estimating the parameters of the model given below constitutes fitting a logistic regression.
- In logistic regression the fitting is carried out by working with the logits.
- The Logit transformation produces a model that is linear in the parameters.
- The method of estimation used is the maximum likelihood method.
- The maximum likelihood estimates are obtained numerically, using an iterative procedure.

$$\begin{aligned}\pi &= \Pr(Y = 1|X_1 = x_1, \cdots, X_p = x_p)\\ &= \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_p x_p}}{1 + e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}}.\end{aligned}$$

OLS:

- The ordinary least squares, or OLS, can also be called the linear least squares.
- This is a method for approximately determining the unknown parameters located in a linear regression model.
- According to books of statistics and other online sources, the ordinary least squares is obtained by minimizing the total of squared vertical distances between the observed responses within the dataset and the responses predicted by the linear approximation.
- Through a simple formula, you can express the resulting estimator, especially the single regressor, located on the right-hand side of the linear regression model.
- For example, you have a set of equations which consists of several equations that have unknown parameters.
- You may use the ordinary least squares method because this is the most standard approach in finding the approximate solution to your overly determined systems.
- In other words, it is your overall solution in minimizing the sum of the squares of errors in your equation.
- Data fitting can be your most suited application. Online sources have stated that the data that best fits the ordinary least squares minimizes the sum of squared residuals.
- “Residual” is “the difference between an observed value and the fitted value provided by a model.”

Maximum likelihood estimation, or MLE,

- is a method used in estimating the parameters of a statistical model, and for fitting a statistical model to data.
- If you want to find the height measurement of every basketball player in a specific location, you can use the maximum likelihood estimation.
- Normally, you would encounter problems such as cost and time constraints.
- If you could not afford to measure all of the basketball players' heights, the maximum likelihood estimation would be very handy.
- Using the maximum likelihood estimation, you can estimate the mean and variance of the height of your subjects.
- The MLE would set the mean and variance as parameters in determining the specific parametric values in a given model.

Multinomial Logistic Regression

- We have n independent observations with p explanatory variables.
- The qualitative response variable has k categories.
- To construct the logits in the multinomial case one of the categories is considered the base level and all the logits are constructed relative to it. Any category can be taken as the base level.
- We will take category k as the base level in our description of the method.
- Since there is no ordering, it is apparent that any category may be labeled k. Let 7rj denote the multinomial probability of an observation falling in the jth category.
- We want to find the relationship between this probability and the p explanatory variables, Xl, X 2 , ... ,Xp. The multiple logistic regression model then is

$$\ln\left(\frac{\pi_j(x_i)}{\pi_k(x_i)}\right) = \beta_{0j} + \beta_{1j}x_{1i} + \beta_{2j}x_{2i} + \cdots + \beta_{pj}x_{pi}; \qquad \begin{array}{l} j = 1, 2, \cdots, (k-1), \\ i = 1, 2, \cdots, n. \end{array}$$

- Since all the 7r'S add to unity, this reduces to

$$\ln(\pi_j(x_i)) = \frac{\exp(\beta_{0j} + \beta_{1j}x_{1i} + \beta_{2j}x_{2i} + \cdots + \beta_{pj}x_{pi})}{1 + \sum_{j=1}^{k-1} \exp(\beta_{0j} + \beta_{1j}x_{1i} + \beta_{2j}x_{2i} + \cdots + \beta_{pj}x_{pi})},$$

- For j = 1,2,⋯, (k - 1). The model parameters are estimated by the method of maximum likelihood. Statistical software is available to do this fitting.

Regression vs. Segmentation

- **Regression analysis** focuses on finding a relationship between a dependent variable and one or more independent variables.
- Predicts the value of a dependent variable based on the value of at least one independent variable.
- Explains the impact of changes in an independent variable on the dependent variable.

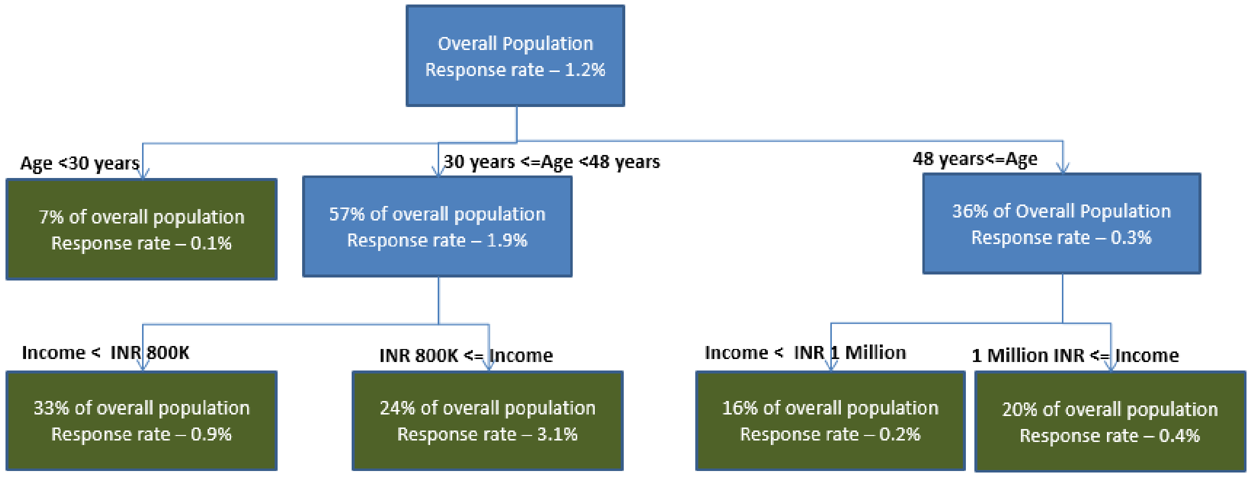
- We use linear or logistic regression technique for developing accurate models for predicting an outcome of interest.
- Often, we create separate models for separate segments.
- **Segmentation methods** such as CHAID or CRT is used to judge their effectiveness
- Creating separate model for separate segments may be time consuming and not worth the effort.
- But, creating separate model for separate segments may provide higher predictive power.

- **Market Segmentation**
- Dividing the target market or customers on the basis of some significant features which could help a company sell more products in less marketing expenses.

- Companies have limited marketing budgets. Yet, the marketing team is expected to makes large number of sales to ensure rising revenue & profits.
- A product is created in two ways:
  - Create a product after analyzing (research) the needs and wants of target market – For example: Computer. Companies like Dell, IBM, Microsoft entered this market after analyzing the enormous market which this product upholds.
  - Create a product which evokes the needs & wants in target market – For example: iPhone.
- Once the product is created, the ball shifts to the marketing team's court.
- As mentioned above, they make use of market segmentation techniques.
- This ensures the product is positioned to the right segment of customers with high propensity to buy.

- **How to create segments for model development?**
- Commonly adopted methodology
- Let us consider an example.
- Here we'll build a logistic regression model for predicting likelihood of a customer to respond to an offer.
- A very similar approach can also be used for developing a linear regression model.

- **Logistic regression** uses 1 or 0 indicator in the historical campaign data, which indicates whether the customer has responded to the offer or not.
- Usually, one uses the target (or 'Y' known as dependent variable) that has been identified for model development to undertake an objective segmentation.
- Remember, a separate model will be built for each segment.
- A segmentation scheme which provides the maximum difference between the segments with regards to the objective is usually selected.
- Below is a simple example of this approach.



- Fig: Sample segmentation for building a logistic regression – commonly adopted methodology
- The above segmentation scheme is the best possible objective segmentation developed, because the segments demonstrate the maximum separation with regards to the objectives (i.e. response rate).

Supervised and Unsupervised Learning

There are two broad set of methodologies for segmentation:

- Objective (supervised) segmentation
- Non-Objective (unsupervised) segmentation

Objective Segmentation

- Segmentation to identify the type of customers who would respond to a particular offer.
- Segmentation to identify high spenders among customers who will use the e-commerce channel for festive shopping.
- Segmentation to identify customers who will default on their credit obligation for a loan or credit card.

Non-Objective Segmentation

- Segmentation of the customer base to understand the specific profiles which exist within the customer base so that multiple marketing actions can be personalized for each segment
- Segmentation of geographies on the basis of affluence and lifestyle of people living in each geography so that sales and distribution strategies can be formulated accordingly.
- Segmentation of web site visitors on the basis of browsing behavior to understand the level of engagement and affinity towards the brand.
- Hence, it is critical that the segments created on the basis of an objective segmentation methodology must be different with respect to the stated objective (e.g. response to an offer).
- However, in case of a non-objective methodology, the segments are different with respect to the "generic profile" of observations belonging to each segment, but not with regards to any specific outcome of interest.
- The most common techniques for building non-objective segmentation are cluster analysis, K nearest neighbor techniques etc.
- Each of these techniques uses a distance measure (e.g. Euclidian distance, Manhattan distance, Mahalanobis distance etc.)
- This is done to maximize the distance between the two segments.
- This implies maximum difference between the segments with regards to a combination of all the variables (or factors).

Tree Building

- Decision tree learning
    - is a method commonly used in data mining.
    - is the construction of a decision tree from class-labeled training tuples.

- goal
    - to create a model that predicts the value of a target variable based on several input variables.

- Decision trees used in data mining are of two main types.
    - *Classification tree analysis*
    - *Regression tree analysis*

    - Classification tree analysis is when the predicted outcome is the class to which the data belongs.
    - Regression tree analysis is when the predicted outcome can be considered a real number. (e.g. the price of a house, or a patient's length of stay in a hospital).

- A decision tree
    - is a flow-chart-like structure
    - each internal (non-leaf) node denotes a test on an attribute
    - each branch represents the outcome of a test,
    - each leaf (or terminal) node holds a class label.
    - The topmost node in a tree is the root node.

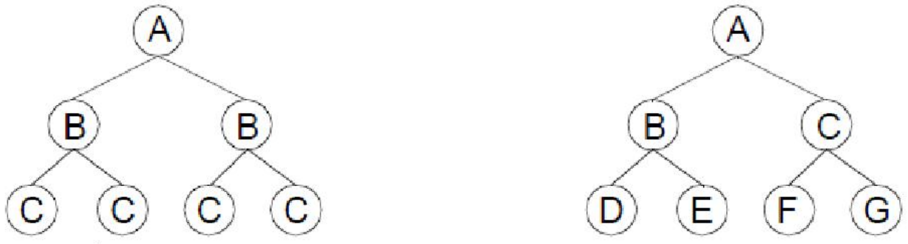
- Decision-tree algorithms:
    - ID3 (Iterative Dichotomiser 3)
    - C4.5 (successor of ID3)
    - CART (**Classification and Regression Tree**)
    - CHAID (CHI-squared Automatic Interaction Detector). Performs multi-level splits when computing **classification trees**.
    - MARS: extends decision trees to handle numerical data better. Conditional Inference Trees.

- Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid over fitting.
- This approach results in unbiased predictor selection and does not require pruning.
- ID3 and CART follow a similar approach for learning decision tree from training tuples.

CHAID (CHI-squared Automatic Interaction Detector)

- A simple method for fitting trees to predict a quantitative variable proposed by Morgan and Sonquist (1963).
- They called the method AID, for Automatic Interaction Detection.
- The algorithm performs stepwise splitting.
- It begins with a single cluster of cases and searches a candidate set of predictor variables for a way to split this cluster into two clusters.
- Each predictor is tested for splitting as follows:
    - Sort all the n cases on the predictor and examine all n-1 ways to split the cluster in two.
    - For each possible split, compute the within-cluster sum of squares about the mean of the cluster on the dependent variable.
    - Choose the best of the n-1 splits to represent the predictor's contribution. Now do this for every other predictor.
    - For the actual split, choose the predictor and its cut point which yields the smallest overall within-cluster sum of squares.
    - Categorical predictors require a different approach. Since categories are unordered, all possible splits between categories must be considered.
    - For deciding on one split of k categories into two groups, this means that 2k-1 possible splits must be considered.
    - Once a split is found, its suitability is measured on the same within-cluster sum of squares as for a quantitative predictor.
- Morgan and Sonquist called their algorithm AID because it naturally incorporates interaction among predictors. Interaction is not correlation.
- It has to do instead with conditional discrepancies.
- In the analysis of variance, interaction means that a trend within one level of a variable is not parallel to a trend within another level of the same variable.

- In the ANOVA model, interaction is represented by cross-products between predictors.
- In the tree model, it is represented by branches from the same nodes which have different splitting predictors further down the tree.



No interaction (left) and interaction (right) trees.

- Regression trees parallel regression/ANOVA modeling in which the dependent variable is quantitative.
- Classification trees parallel discriminant analysis and algebraic classification methods.
- Kass (1980) proposed a modification to AID called CHAID for categorized dependent and independent variables.
- His algorithm incorporated a sequential merge and split procedure based on a chi-square test statistic.
- Kass was concerned about computation time, so he decided to settle for a sub-optimal split on each predictor instead of searching for all possible combinations of the categories.
- Kass's algorithm is like sequential cross-tabulation.
    - For each predictor:
        1. cross tabulate the m categories of the predictor with the k categories of the dependent variable,
        2. find the pair of categories of the predictor whose 2xk sub-table is least significantly different on a chi-square test and merge these two categories;
        3. if the chi-square test statistic is not "significant" according to a preset critical value, repeat this merging process for the selected predictor until no non-significant chi-square is found for a sub-table, and pick the predictor variable whose chi-square is largest and split the sample into subsets, where l is the number of categories resulting from the merging process on that predictor;
        4. Continue splitting, as with AID, until no "significant" chi-squares result. The CHAID algorithm saves some computer time, but it is not guaranteed to find the splits which predict best at a given step. Only by searching all possible category subsets can we do that. CHAID is also limited to categorical predictors, so it cannot be used for quantitative or mixed categorical quantitative models.

CART (Classification And Regression Tree)

- CART algorithm was introduced in Breiman et al. (1986).
- A CART tree is a binary decision tree that is constructed by splitting a node into two child nodes repeatedly, beginning with the root node that contains the whole learning sample.
- The CART growing method attempts to maximize within-node homogeneity.
- The extent to which a node does not represent a homogenous subset of cases is an indication of impurity.
- For example, a terminal node in which all cases have the same value for the dependent variable is a homogenous node that requires no further splitting because it is "pure."
- For categorical (nominal, ordinal) dependent variables the common measure of impurity is Gini, which is based on squared probabilities of membership for each category.

- Splits are found that maximize the homogeneity of child nodes with respect to the value of the dependent variable.

- Impurity Measure:

- GINI Index Used by the CART (classification and regression tree) algorithm, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset.
- Gini impurity can be computed by summing the probability fi of each item being chosen times the probability 1-fi of a mistake in categorizing that item.
- It reaches its minimum (zero) when all cases in the node fall into a single target category.
- To compute Gini impurity for a set of items, suppose i ε {1, 2... m}, and let fi be the fraction of items labeled with value i in the set.

$$I_G(f) = \sum_{i=1}^{m} f_i(1-f_i) = \sum_{i=1}^{m}(f_i - f_i{}^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i{}^2 = 1 - \sum_{i=1}^{m} f_i{}^2 = \sum_{i \neq k} f_i f_k$$

Advantages of Decision Tree:

- *Simple to understand and interpret*. People are able to understand decision tree models after a brief explanation.
- *Requires little data preparation*. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- *Able to handle both numerical and categorical data*. Other techniques are usually specialized in analysing datasets that have only one type of variable.
- *Uses a white box model*. If a given situation is observable in a model the explanation for the condition is easily explained by Boolean logic.
- *Possible to validate a model using statistical tests*. That makes it possible to account for the reliability of the model.
- *Robust.* Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
- Performs well with large datasets. Large amounts of data can be analyzed using standard computing resources in reasonable time.

Tools used to make Decision Tree:

- Many data mining software packages provide implementations of one or more decision tree algorithms.
- Several examples include:
  - Salford Systems CART
  - IBM SPSS Modeler
  - Rapid Miner
  - SAS Enterprise Miner
  - Matlab
  - R (an open source software environment for statistical computing which includes several CART implementations such as rpart, party and random Forest packages)
  - Weka (a free and open-source data mining suite, contains many decision tree algorithms)
  - Orange (a free data mining software suite, which includes the tree module orngTree)
  - KNIME
  - Microsoft SQL Server
  - Scikit-learn (a free and open-source machine learning library for the Python programming language).

- <u>Pruning</u>
- After building the decision tree, a tree-pruning step can be performed to reduce the size of the decision tree.
- Pruning helps by trimming the branches of the initial tree in a way that improves the generalization capability of the decision tree.

- The errors committed by a classification model are generally divided into two types:
  - training errors
  - generalization errors.

- Training error
  - also known as resubstitution error or apparent error.
  - it is the number of misclassification errors committed on training records.
- generalization error
  - is the expected error of the model on previously unseen records.
  - A good classification model must not only fit the training data well, it must also accurately classify records it has never seen before.
- A good model must have low training error as well as low generalization error.

- <u>Model overfitting</u>
    - Decision trees that are too large are susceptible to a phenomenon known as overfitting.
    - A model that fits the training data too well can have a poorer generalization error than a model with a higher training error.
    - Such a situation is known as model overfitting.

- <u>Model underfitting</u>
    - The training and test error rates of the model are large when the size of the tree is very small.
    - This situation is known as model underfitting.
    - Underfitting occurs because the model has yet to learn the true structure of the data.

- <u>Model complexity</u>
    - To understand the overfitting phenomenon, the training error of a model can be reduced by increasing the model complexity.
    - Overfitting and underfitting are two pathologies that are related to the model complexity.

ARIMA (Autoregressive Integrated Moving Average)

- ARIMA model is a generalization of an autoregressive moving average (ARMA) model, in time series analysis,
- These models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting).
- They are applied in some cases where data show evidence of non-stationary, wherein initial differencing step (corresponding to the "integrated" part of the model) can be applied to reduce the non-stationary.

- *Non-seasonal ARIMA models*
    - These are generally denoted ARIMA(p, d, q) where parameters p, d, and q are non-negative integers, p is the order of the Autoregressive model, d is the degree of differencing, and q is the order of the Moving-average model.

- *Seasonal ARIMA models*
    - These are usually denoted ARIMA(p, d, q)(P, D, Q)_m, where m refers to the number of periods in each season, and the uppercase P, D, Q refer to the autoregressive, differencing, and moving average terms for the seasonal part of the ARIMA model.

- ARIMA models form an important part of the Box-Jenkins approach to time-series modeling.

- *Applications*
    - ARIMA models are important for generating forecasts and providing understanding in all kinds of time series problems from economics to health care applications.
    - In quality and reliability, they are important in process monitoring if observations are correlated.
    - designing schemes for process adjustment
    - monitoring a reliability system over time
    - forecasting time series
    - estimating missing values
    - finding outliers and atypical events
    - understanding the effects of changes in a system

Measure of Forecast Accuracy

- Forecast Accuracy can be defined as the deviation of Forecast or Prediction from the actual results.

Error = Actual demand – Forecast

OR

ei = At – Ft

- We measure Forecast Accuracy by 2 methods :
  - *Mean Forecast Error (MFE)*
    - For n time periods where we have actual demand and forecast values:

$$MFE = \frac{\sum_{i=1}^{n}(e_i)}{n}$$

    - Ideal value = 0;
    - MFE > 0, model tends to under-forecast
    - MFE < 0, model tends to over-forecast

  - *Mean Absolute Deviation (MAD)*
    - For n time periods where we have actual demand and forecast values:

$$MAD = \frac{\sum_{i=1}^{n}|e_i|}{n}$$

- While MFE is a measure of forecast model bias, MAD indicates the absolute size of the errors

Uses of Forecast error:

- Forecast model bias
- Absolute size of the forecast errors
- Compare alternative forecasting models
- Identify forecast models that need adjustment

ETL Approach

- Extract, Transform and Load (ETL) refers to a process in database usage and especially in data warehousing that:
  - Extracts data from homogeneous or heterogeneous data sources
  - Transforms the data for storing it in proper format or structure for querying and analysis purpose
  - Loads it into the final target (database, more specifically, operational data store, data mart, or data warehouse)
- Usually all the three phases execute in parallel since the data extraction takes time, so while the data is being pulled another transformation process executes, processing the already received data and prepares the data for loading and as soon as there is some data ready to be loaded into the target, the data loading kicks off without waiting for the completion of the previous phases.
- ETL systems commonly integrate data from multiple applications (systems), typically developed and supported by different vendors or hosted on separate computer hardware.
- The disparate systems containing the original data are frequently managed and operated by different employees.
- For example, a cost accounting system may combine data from payroll, sales, and purchasing.

- Commercially available ETL tools include:
  - Anatella
  - Alteryx
  - CampaignRunner
  - ESF Database Migration Toolkit
  - InformaticaPowerCenter
  - Talend
  - IBM InfoSphereDataStage
  - Ab Initio
  - Oracle Data Integrator (ODI)
  - Oracle Warehouse Builder (OWB)
  - Microsoft SQL Server Integration Services (SSIS)
  - Tomahawk Business Integrator by Novasoft Technologies.
  - Pentaho Data Integration (or Kettle) opensource data integration framework Stambia
  - Diyotta DI-SUITE for Modern Data Integration
  - FlyData
  - Rhino ETL
  - SAP Business Objects Data Services

  - SAS Data Integration Studio
  - SnapLogic
  - Clover ETL opensource engine supporting only basic partial functionality and not server
  - SQ-ALL - ETL with SQL queries from internet sources such as APIs
  - North Concepts Data Pipeline

- <u>Various steps involved in ETL.</u>
  - Extract
  - Transform
  - Load

  - <u>Extract</u>
    - The Extract step covers the data extraction from the source system and makes it accessible for further processing.
    - The main objective of the extract step is to retrieve all the required data from the source system with as little resources as possible.
    - The extract step should be designed in a way that it does not negatively affect the source system in terms or performance, response time or any kind of locking.

    - There are several ways to perform the extract:
      - Update notification - if the source system is able to provide a notification that a record has been changed and describe the change, this is the easiest way to get the data.
      - Incremental extract - some systems may not be able to provide notification that an update has occurred, but they are able to identify which records have been modified and provide an extract of such records. During further ETL steps, the system needs to identify changes and propagate it down. Note, that by using daily extract, we may not be able to handle deleted records properly.
      - Full extract - some systems are not able to identify which data has been changed at all, so a full extract is the only way one can get the data out of the system. The full extract requires keeping a copy of the last extract in the same format in order to be able to identify changes. Full extract handles deletions as well.
      - When using Incremental or Full extracts, the extract frequency is extremely important. Particularly for full extracts; the data volumes can be in tens of gigabytes.

- Clean - The cleaning step is one of the most important as it ensures the quality of the data in the data warehouse. Cleaning should perform basic data unification rules, such as:
- Making identifiers unique (sex categories Male/Female/Unknown, M/F/null, Man/Woman/Not Available are translated to standard Male/Female/Unknown)
- Convert null values into standardized Not Available/Not Provided value
- Convert phone numbers, ZIP codes to a standardized form
- Validate address fields, convert them into proper naming, e.g. Street/St/St./Str./Str
- Validate address fields against each other (State/Country, City/State, City/ZIP code, City/Street).

- <u>Transform</u>
  - The transform step applies a set of rules to transform the data from the source to the target.
  - This includes converting any measured data to the same dimension (i.e. conformed dimension) using the same units so that they can later be joined.
  - The transformation step also requires joining data from several sources, generating aggregates, generating surrogate keys, sorting, deriving new calculated values, and applying advanced validation rules.

- <u>Load</u>
  - During the load step, it is necessary to ensure that the load is performed correctly and with as little resources as possible.
  - The target of the Load process is often a database.
  - In order to make the load process efficient, it is helpful to disable any constraints and indexes before the load and enable them back only after the load completes.
  - The referential integrity needs to be maintained by ETL tool to ensure consistency.

- <u>Managing ETL Process</u>

  - The ETL process seems quite straight forward.
  - As with every application, there is a possibility that the ETL process fails.
  - This can be caused by missing extracts from one of the systems, missing values in one of the reference tables, or simply a connection or power outage.

- Therefore, it is necessary to design the ETL process keeping fail-recovery in mind.

- Staging
  - It should be possible to restart, at least, some of the phases independently from the others.
  - For example, if the transformation step fails, it should not be necessary to restart the Extract step.
  - We can ensure this by implementing proper staging. Staging means that the data is simply dumped to the location (called the Staging Area) so that it can then be read by the next processing phase.
  - The staging area is also used during ETL process to store intermediate results of processing.
  - This is ok for the ETL process which uses for this purpose.
  - However, the staging area should be accessed by the load ETL process only.
  - It should never be available to anyone else; particularly not to end users as it is not intended for data presentation to the end-user.
  - May contain incomplete or in-the-middle-of-the-processing data.

UNIT-V

1. RETAIL ANALYTICS:

WHAT IS RETAIL ANALYTICS: Retail analytics is the process of providing analytical data on inventory levels, supply chain movement, consumer demand, sales, etc. that are crucial for making marketing, and procurement decisions.

To understand the role analytics plays in retail, it is useful to break down the business decisions taken in retail into the following categories:

consumer,

product,

workforce, and

advertising.

1. Consumer: Personalization is a key consumer-level decision that retail firms make. Personalized pricing by offering discounts via coupons to select customers is one such decision. This approach uses data collection via loyalty cards to better understand a customer's purchase patterns and willingness to pay and uses that to offer personalized pricing. Such personalization can also be used as a customer retention strategy. Another example is to offer customers a personalized sales experience: in e-tail settings, this entails offering customers a unique browsing experience by modifying the products displayed and suggestions made based on the customer's historical information.

2. Product: Retail product decisions can be broken down into single product and group of product decisions. Single or individual product decisions are mostly inventory decisions: how much stock of the product to order, and when to place the order. At the group level, the decisions are typically related to pricing and assortment planning. That is, what price to set for each product in the group and how to place the products on the store-shelves, keeping in mind the variety of products, the number of each type of product, and location . To make these decisions, predictive modelling is called for to forecast the product demand and the price-response function, and essentially the decision-maker needs to understand how customer reacts to price changes. A fine understanding of consumer choice is also needed to understand how a customer chooses to buy a certain product from a group of products.

3. Human resources: The key decisions here are related to the number of employees needed in the store at various times of the day and how to schedule them. To make these decisions, the overall work to be completed by the employees needs to be estimated. Part of this is a function of other decisions, such as the effort involved in stocking shelves, taking deliveries, changing prices, etc. There is additional work that comes in as a function of the customer volume in the store. This includes answering customer questions and manning checkout counters.

4. Advertising: In the advertising sphere, companies deal with the typical decisions of finding the best medium to advertise on (online mediums such as Google Ad words, Facebook, Twitter, and/or traditional mediums such as print and newspaper inserts) and the best products to advertise. This may entail cultivating some “loss-leaders” that are priced low to entice customers into the store, so they may also purchase other items which have a greater margin.

Examples of Retail Analytics in Action:

• Analytics has revealed that a great number of customer visits to online stores fail to convert at the last minute, when the customer has the item in their shopping basket but does not go on to confirm the purchase. Theorizing that this was because customers often cannot find their credit or debit cards to confirm the details, Swedish e-commerce platform Klarna moved its clients (such as Vistaprint, Spotify, and 45,000 online stores) onto an invoicing model, where customers can pay after the product is delivered. Sophisticated fraud prevention analytics are used to make sure that the system cannot be manipulated by those with devious intent.

•Trend forecasting algorithms comb social media posts and Web browsing habits to elicit what products may be causing a buzz, and ad-buying data is analyzed to see what marketing departments will be pushing. Brands and marketers engage in “sentiment analysis,” using sophisticated machine learning-based algorithms to determine the context when a product is discussed. This data can be used to accurately predict what the top selling products in a category are likely to be.

•Amazon has proposed using predictive shipping analytics6 to ship products to customers before they even click “add to cart.” According to a recent trend report by DHL, over the next 5 years, this so-called psychic supply chain will have far reaching effects in nearly all industries, from automotive to consumer goods. It uses big data and advanced predictive algorithms to enhance planning and decision-making.

Complications in Retail Analytics:

There are various complications that arise in retail scenarios that need to be overcome for the successful use of retail analytics. These complications can be classified into

(a) those that affect predictive modelling and

(b) those that affect decision-making.

Some of the most common issues that affect predictive modelling are demand censoring and inventory inaccuracies (DeHoratius and Raman 2008). Typically**, retail firms only have access to sales information, not demand information,** and therefore need to account for the fact that when inventory runs out, actual demand is not observed.

Methodologies:

Product-Based Demand Modelling:

Typical forecasting methods consider the univariate time series of sales data and use time-series-based methods such as exponential smoothing and ARIMA models; These methods typically focus on forecasting sales and may require uncensoring to be used for decision-making. Recently, there have been advances that utilize statistical and machine learning approaches to deal with greater amounts of data.

Incorporating Consumer Choice in Demand Modeling:

This directly motivates modeling customer preferences over all the products carried by the retailer. One of the workhorse models for such consumer choice modeling is the multinomial logit (MNL);

$P\ (\text{Customer chooses product } j\ ) = \exp\ vj / 1 + k \in A \exp\ (vk)$

Business Challenges and Opportunities:

Omni-Channel Retail:

The tremendous success of e-commerce has led many retailers to augment their brick-and-mortar stores with an online presence, leading to the advent of multichannel retail. In this approach the retailer has access to multiple channels to engage with and sell to customer.

A good example of such an approach is the “buy online, pick up in store” (BOPS) approach that has become quite commonplace. This seamless approach inarguably improves the customer experience and overall sales;

Retail Startups:

In terms of data collection, there are many startups that cater to the range of retailers both small and large. Some illustrative examples here are Euclid Analytics, which uses in-store Wi-Fi to collect information on customers via their smartphones.

2. Marketing Analytics:

Marketing analytics can help firms realize the true potential of data and explore meaningful insights. Marketing analytics can be defined as a “high technology enabled and marketing science model-supported approach to harness the true values of the customer, market, and firm level data to enhance the effect of marketing strategies” Basically, marketing analytics is the creation and use of data to measure and optimize marketing decisions. Marketing analytics comprises tools and processes

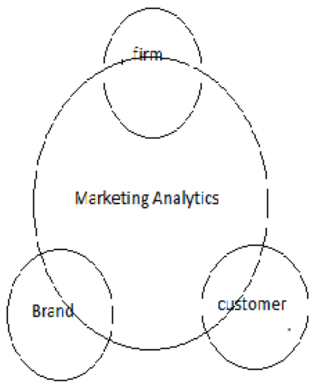
The processes and tools discussed in this chapter will help in various aspects of marketing such as target marketing and segmentation, price and promotion, customer valuation, resource allocation, response analysis, demand assessment, and new product development. These can be applied at the following levels:

• Firm: At this level, tools are applied to the firm as a whole. Instead of focusing on a particular product or brand, these can be used to decide and evaluate firm strategies. For example, data envelopment analysis (DEA) can be used for all the units (i.e., finance, marketing, HR, operation, etc.) within a firm to find the most efficient units and allocate resources accordingly.

• Brand/product: At the brand/product level, tools are applied to decide and evaluate strategies for a particular brand/product. For example, conjoint analysis can be conducted to find the product features preferred by customers or response analysis can be conducted to find how a particular brand advertisement will be received by the market.

• Customer: Tools applied at customer level provide insights that help in segmenting and targeting customers. For example, customer lifetime value is a forward-looking customer metric that helps assess the value provided by customers to the firm

Fig. 19.1 Levels of marketing analytics application



let us look at what constitutes marketing analytics. Though it is an ever-expanding field, for our purpose, we can segment marketing analytics into the following processes and tools:

1. Multivariate statistical analysis: It deals with the analysis of more than one outcome variable. Cluster analysis, factor analysis, perceptual maps, conjoint analysis, discriminant analysis, and MANOVA are a part of multivariate statistical analysis. These can help in target marketing and segmentation, optimizing product features, etc., among other applications.

2. Choice analytics: Choice modeling provides insights on how customers make decisions. Understanding customer decision-making process is critical as it can help to design and optimize various marketing mix strategies such as pricing and advertising. Largely, Logistic, Probit, and Tobit models .

3. Regression models: Regression modeling establishes relationships between dependent variables and independent variables. It can be used to understand outcomes such as sales and profitability, on the basis of different factors such as price and promotion. Univariate analysis, multivariate analysis, nonlinear analysis, and moderation and mediation analysis are covered in this section.

4. Time-series analytics: Models stated till now mainly deal with cross-sectional data (however, choice and regression models can be used for panel data as well). This section consists of auto-regressive models and vector auto-regressive models for time-series analysis. These can be used for forecasting sales, market share, etc.

5. Nonparametric tools: Non parametric tools are used when the data belongs to no particular distribution. Data envelopment analysis (DEA) and stochastic frontier analysis (SFA) are discussed in this section and can be used for benchmarking, resource allocation, and assessing efficiency.

6. Survival analysis: Survival analysis is used to determine the duration of time until an event such as purchase, attrition, and conversion happens. Baseline hazard model, proportional hazard model, and analysis with time varying covariates are covered in this section. 626 S. Arunachalam and A. Sharma

7. Sales force /sales analytics: This section covers analytics for sales, which includes forecasting potential sales, forecasting market share, and causal analysis. It comprises various methods such as chain ratio method, Delphi method, and product life cycle analysis.

8. Innovation analytics: Innovation analytics deals specifically with new products. New product analysis differs from existing product analysis as you may have little or no historical data either for product design or sales forecasting. Bass model, ASSESSOR model, conjoint analysis can be used for innovation analytics.

9. Conjoint analysis: This section covers one of the most widely used quantitative methods in marketing research. Conjoint (trade-off) analysis is a statistical technique to measure customer preferences for various attributes of a product or service. This is used in various stages of a new product design, segmenting customers and pricing.

10. Customer analytics: In this section, we probe customer metrics such as customer lifetime value, customer referral value, and RFM (recency, frequency, and monetary value) analysis. These can be used for segmenting customers and determining value provided by customers.

3.Financial Analytics:

Introduction: Data analytics in finance is a part of quantitative finance. Quantitative finance primarily consists of three sectors in finance—asset management, banking, and insurance. Across these three sectors, there are four tightly connected functions in which quantitative finance is used—valuation, risk management, portfolio management, and performance analysis. Data analytics in finance supports these four sequential building blocks of quantitative finance, especially the first three— valuation, risk management, and portfolio management.

1.2 Dichotomized World of Quant Finance

One can paraphrase Rudyard Kipling’s poem The Ballad of East and West and say, “Oh, the Q-world is the Q-world, the P-world is the P-world, and never the twain shall meet.”

Q-Quants:

In the Q-world, the objective is primarily to determine a fair price for a financial instrument, especially a derivative security, in terms of its underlying securities. The price of these underlying securities is determined by the market forces of demand and supply. The demand and supply forces come from a variety of sources in the financial markets, but they primarily originate from buy-side and sell-side financial institutions.

The Q-quants typically have deep knowledge about a specific product. So a Qquant who, for instance, trades credit derivatives for a living would have abundant knowledge about credit derivative products, but her know-how may not be very useful in, say, a domain like foreign exchange.

P-Quants:

We now discuss the P-world and their origins, tools, and techniques and contrast them with the Q-world. The P-world started with the mean–variance framework by Markowitz in 1952 (Markowitz 1952). Harry Markowitz showed that the conventional investment evaluation criteria of net present value (NPV) needs to be explicitly segregated in terms of risk and return. He defined risk as standard deviation of return distribution.

In reality, the probability distribution needs to be estimated from the available financial information. So a very large component of this so-called 20 Financial Analytics 663 information set, that is, the prices and other financial variables, is observed at discrete time intervals, forming a time series.

Methodology of Data Analysis in Finance: Three-Stage Process:

Methodology of three stage framework for data analysis in finance

Stage: I Asset Price Esmaon

• Step 1: Identification

• Step 2: I.I.D.

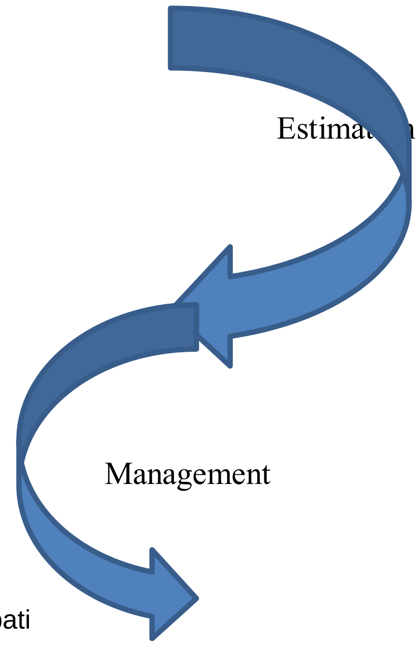
• Step 3: Inference

• Step 4: Projection

• Step 5: Pricing

Stage: II Risk Management

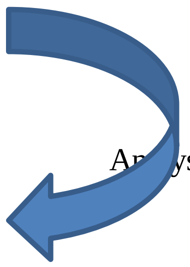
• Step 1: Aggregation



- Step 2: Assessment
- Step 3: Attribution

Stage: III Portfolio Analysis

- Step 1: Allocation
- Step 2: Execution



Analysis

Fig. 20.1 Methodology of the three-stage framework for data analysis in finance

The threestage methodology starting with variable identification for different asset classes such as equities, foreign exchange, fixed income, credit, and commodities.

1.3.1 Stage I: Asset Price Estimation The objective of the first stage is to estimate the price behavior of an asset. It starts with identification of the financial variable to model.

Step 1: Identification :

The first step of modeling in the P-world is to identify the appropriate variable which is different for distinct asset classes. The basic idea is to find a process for the financial variable where the residuals are essentially i.i.d. The most common process used for modeling a financial variable x is the random walk: $x_t = x_{t-1} + \varepsilon_t$

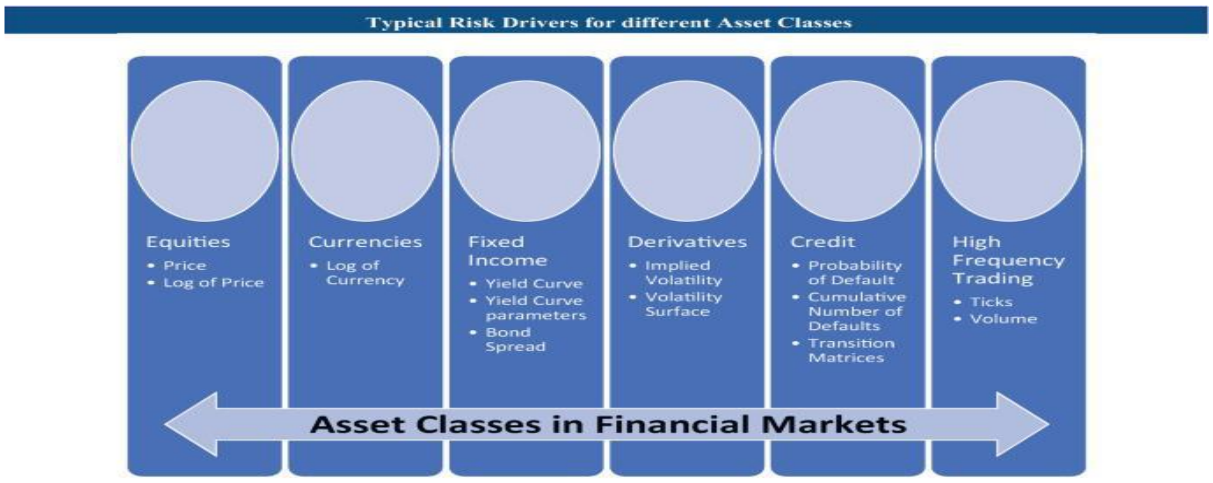


Fig:Typical risk drivers for different asset classes

Step 2: I.I.D.

Once the financial variable that is of interest is identified, the next step in data preparation is to obtain a time series of the variables that are of interest. These variables should display a homogenous behavior across time, For instance, in equities, currencies, or commodities, it is the log of the stock/currency/commodity price. For fixed income instruments, the variable of interest may not be the price or the log of price. The variable of interest would be the yield to maturity of the fixed income security.

Step 3: Inference

The third step in estimation after the financial variable is identified and after we have gotten to the point of i.i.d. shocks is to infer the joint behavior of i.i.d. shocks. In the estimation process, we typically determine those parameters in the model which gets us to an i.i.d. distribution.

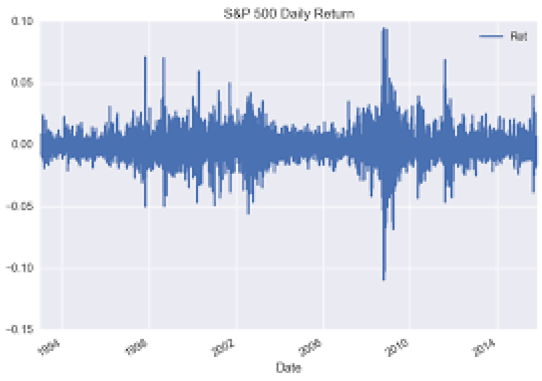


Fig: Daily returns of the S&P 500

absolute index levels of S&P 500 for reasons mentioned in Step 1. From the daily index levels, the 1-day returns are calculated as follows: rt = log (pt) − log (pt−1) This return rt itself is not distributed in an i.i.d. sense. Neither are the daily returns identical nor are they independent. We can infer from a quick look at the graph of the returns data.

The way we model variance σt 2 is: σt 2 = ω + α rt−1 2 + βσt−1 2 We have to estimate the parameters ω, α, β. The estimation technique we use is maximum likelihood estimation (MLE).

Using the Gaussian distribution, the likelihood or the probability of rt σt being normally distributed is given by: 1 / 2πσt 2 e^ – 1/ 2(rt/ σt) 2

Step 4: Projection

The fourth step is projection. We explain this step using a simple example from foreign exchange markets. Let us say that the financial variable is estimated using a technique such as MLE, GMM, or Bayesian estimation (Hansen 1982). The next step is to project the variable using the model. Say the horizon is 1 year, and we want to calculate the expected profit or loss of a certain portfolio.

Monte Carlo Simulation

We first pick a random number from the standard normal distribution say x. We then scale (multiply) x by standard deviation and add average return to get a random variable mapped to the exact normal distribution of returns.

R = x ∗ (10%) /sqrt(365) + (4.5%) /365

Note that the average return and standard deviation are adjusted for daily horizon by dividing with 365 and square root of 365, respectively

Year	INR/USD	% change	Year	INR/USD	% change	Year	INR/USD	% change	Year	INR/USD	% change
1973	7.66	3.86	1983	10.11	6.64	1993	31.26	11.08	2003	46.6	-4.15
1974	8.03	4.83	1984	11.36	12.36	1994	31.39	0.41	2004	45.28	-2.83
1975	8.41	4.73	1985	12.34	8.62	1995	32.43	3.31	2005	44.01	-2.80
1976	8.97	6.65	1986	12.6	2.10	1996	35.52	9.52	2006	45.17	2.63
1977	8.77	-2.22	1987	12.95	2.77	1997	36.36	2.36	2007	41.2	-8.78
1978	8.2	-6.49	1988	13.91	7.41	1998	41.33	13.66	2008	43.41	5.36
1979	8.16	-0.48	1989	16.21	16.53	1999	43.12	4.33	2009	48.32	11.31
1980	7.89	-3.30	1990	17.5	7.95	2000	45	4.36	2010	45.65	-5.52
1981	8.68	10.01	1991	22.72	29.82	2001	47.23	4.95	2011	46.61	2.10
1982	9.48	9.21	1992	28.14	23.85	2002	48.62	2.94	2012	53.34	14.43
Ave	**8.42**	**2.54**	Ave	**15.78**	**11.81**	Ave	**39.23**	**5.69**	Ave	**45.9**	**1.17**

Average of Annual Exchange Rate of Indian Rupee against US Dollar

Step 5: Pricing
The fifth step is pricing which logically follows from projection. The example that we used in Step 4 was projection of USD/INR for a horizon of 1 year. What pricing allows us to do is arrive at the ex-ante expected profit or loss of a specific instrument based on the projections done

Stage II: Risk Management The second stage of data analytics in finance concerns risk management. It involves analysis for risk aggregation, risk assessment, and risk attribution. The framework can be used for risk analysis of a portfolio or even for an entire financial institution.

Step 1: Aggregation The first of the three steps in risk management is risk aggregation. The aggregation step is crucial because all financial institutions need to know the value of the portfolio of their assets and also the aggregated risk exposures in their balance sheet.
We model this using a one-factor model. The single factor is assumed to be the state of the economy M, which is assumed to have a Gaussian distribution. To generate a one-factor model, we define random variables xi (1 ≤ i ≤ N): xi = ρiM + sqrt(1 − ρ 2 i Zi)

We model this using a one-factor model. The single factor is assumed to be the state of the economy M, which is assumed to have a Gaussian distribution. To generate a one-factor model, we define random variables xi (1 ≤ i ≤ N): xi = ρiM + sqrt(1 − ρ 2 i Zi)

Step 2: Assessment
We now move on to the second step of risk management which is assessment of the portfolio. Assessment of the portfolio is done by summarizing it according to a suitable statistical feature. More precisely, assessment is done by calculating the exante risk of the portfolio using metrics such as threshold persistence (TP) or value at risk (VaR) and sometimes sensitizing it using methods like stress-testing.
Step 3: Attribution
The third step in risk management analysis is attribution. Once we have assessed the risk of the portfolio in the previous step, we need to now attribute the risk to different risk factors.

4. Social media and web analytics:

Social media has created new opportunities to both consumers and companies. It has become one of the major drivers of consumer revolution. Companies can analyze data available from the web and social media to get valuable insights into what consumers want. Social media and web analytics can help companies measure the impact of their advertising and the effect of mode of message delivery on the consumers. Companies can also turn to social media analytics to learn more about their consumers.

4.1 Why Is It Different? What All Does It Cover?

Social media analytics involves gathering information from social networking sites such as Facebook, LinkedIn and Twitter in order to provide businesses with better understanding of customers. It helps in understanding customer sentiment, creating

customer profiles and evolving appropriate strategies for reaching the right customer at the right time.
It involves four basic activities, namely, listening (aggregating what is being said on social media),
analyzing (such as identifying trends, shifts in customer preferences and customer sentiments), understanding (gathering insights into customers, their interests and preferences and sentiments) and strategizing (creating appropriate strategies and campaigns to connect with customers with a view to encourage sharing and commenting as well as improving referrals).
One of the major advantages of social media analytics is that it enables businesses to identify and encourage different activities that drive revenues and profits and make real-time adjustments in the strategies and campaigns. Social media analytics can help businesses in targeting advertisements more effectively and thereby reduce the advertising cost while improving ROI

4.2 What Additional Information/Details Can It Provide?

Many companies have started leveraging the power of social media. A particular airline keeps the customers informed through social media about the delays and the causes for such delays. In the process, the airline is able to proactively communicate the causes for the delays and thereby minimize the negative sentiment arising out of the delays. In addition, the airline company is also able to save much of the time of its call centre employees, because many customers already knew about the delays as well as the reasons associated with the delays and hence do not make calls to the call centre.

Search engine optimization (SEO) is another technique to acquire customers when they are looking for a specific product or service or even an organization. For example, when a customer initiates a search for a product, say a smartphone, there is a possibility of getting overloaded and overwhelmed with the search results. These results contain both “paid listings” and “organic listings”.

Display Advertising in Real Time:

The Internet provides new scope for creative approaches to advertising. Advertising on the Internet is also called online advertising, and it encompasses display advertisements found on various websites and results pages of search queries and those placed in emails as well as social networks.

There are different types of display advertisements. The most popular one is the banner advertisement. This is usually a graphic image, with or without animation, displayed on a web page. These advertisements are usually in the GIF or JPEG images if they are static, but use Flash, JavaScript or video if there are animations involved.

How to Get the Advertisements Displayed?

There are many options for getting the advertisements displayed online. Some of these are discussed below. One of the most popular options is placing the advertisements on social media. You can get your ads displayed on social media such as Facebook, Twitter and LinkedIn. In general, Facebook offers standard advertisement space on the right-hand side bar. These advertisements can be placed based on demographic information as well as hobbies and interests which can make it easy to target the right audience.

Programmatic Display Advertising:

Programmatic advertising is “the automation of the buying and selling of desktop display, video, FBX, and mobile ads using real-time bidding. Programmatic describes how online campaigns are booked, flighted, analyzed and optimized via demand-side software (DSP) interfaces and algorithms”

The main components of programmatic display advertising are as follows:

(a) Supply-Side Platform (SSP) The SSP helps the publishers to better manage and optimize their online advertising space and advertising inventory. SSP constantly interacts with the ad exchange and the demand-side platform (DSP). Admeld (www.admeld.com) and Rubicon (https://rubiconproject.com/) are two examples of SSPs.

(b) Demand-Side Platform (DSP) The DSP enables the advertisers to set and apply various parameters and automate the buying of the displays. It also enables them to monitor the performance of their campaigns. Turn (now Amobee, https://www.amobee.com/), AppNexus (https://www.appnexus.com/) and Rocket Fuel (https://rocketfuel. com/) are some of the DSPs.

(c) Ad Exchange Ad exchanges such as Facebook Ad Exchange or DoubleClick Ad Exchange facilitate purchase of available display inventory through auctions. These auctions are automated and take place within milliseconds, before a web page loads on the consumer’s screen. These enable the publishers to optimize the price of their available inventory.

(d) Publisher Publishers are those who provide the display ad inventory.

(e) Advertiser The advertiser bids for the inventory in real time depending on the relevance of the inventory.

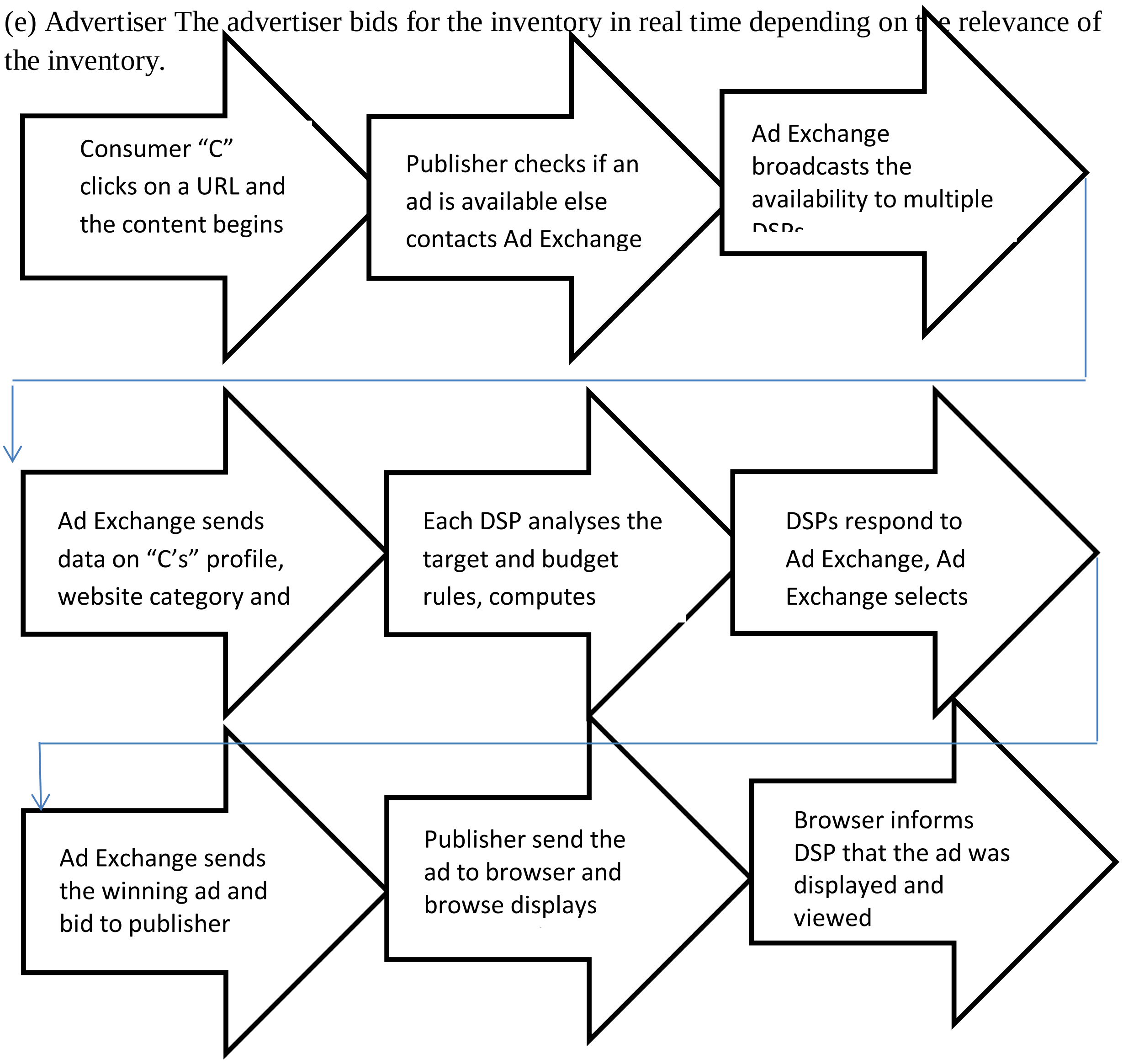

Consumer "C" clicks on a URL and the content begins
Publisher checks if an ad is available else contacts Ad Exchange
Ad Exchange broadcasts the availability to multiple DSPs
Ad Exchange sends data on "C's" profile, website category and
Each DSP analyses the target and budget rules, computes
DSPs respond to Ad Exchange, Ad Exchange selects
Ad Exchange sends the winning ad and bid to publisher
Publisher send the ad to browser and browse displays
Browser informs DSP that the ad was displayed and viewed

Fig. Flowchart of real-time bidding

The entire process described above takes less than half a second. In other words, the entire process is completed and the display ad is shown while the browser is loading the requested page on the consumer's screen.

This process of matching the right display to the right consumer is completely data driven. Data with respect to the context, who is to see the display, the profile of the consumer, who is a good target, etc. is part of the process. In order to complete the process within a very short time span, it is necessary to build all the rules in advance into the system.

5.Healthcare Analytics

What is health care analytics?

Health care analytics is a subset of data analytics that uses both historic and current data to produce actionable insights, improve decision making, and optimize outcomes within the health care industry. Health care analytics is not only used to benefit health care organizations but also to improve the patient experience and health outcomes.

Data analytics in health care

The health care industry is awash with valuable data in the form of detailed records. Industry regulations stipulate that health care providers must retain many of these records for a set period of time.

This means that health care has become a site of interest for those working with "big data," or large pools of unstructured data. As a still-developing field, big data analytics in health care offers the potential to reduce operation costs, improve efficiency, and treat patients.

Predictive analytics in health care

Predictive analytics is the use of historical data to identify past trends and project associated future outcomes. In the health care industry, predictive analytics has many impactful uses, such as identifying a patient's risk for developing a health condition, streamlining treatment courses, and reducing a hospital's number of 30-day readmissions (which can result in costly fines for the hospital).

A 2021 study conducted by a University of Michigan research team illustrates the positive impact that predictive analytics can have on patient treatment. During the study, researchers devised a sensitive blood test that predicted how well patients with HPV-positive throat cancer would respond to specific treatment courses. Overall, the researchers found that their method could predict treatment effectiveness many months earlier than traditional scans [1].

Prescriptive analytics in health care

Prescriptive analytics is the use of historical data to identify an appropriate course of action. In the health care industry, prescriptive analytics is used to both direct business decisions and to literally prescribe treatment plans for patients. As a result, some of the most common uses of prescriptive analytics in health care include identifying a patient's likelihood of developing diabetes, allocating ventilators for a hospital unit, and enhancing diagnostic imaging tools.

Benefits in health care analytics:

Health care analytics offers benefits to health businesses, hospital administrators, and patients. Although it can be tempting to imagine health care analysts working in a virtual data cloud, the reality is that their work has a tangible impact on how hospitals operate, treatment is provided, and medical research is conducted.

At a glance, some of the most common benefits of health care analytics include:

- Improved patient care, such as offering more effective courses of treatment
- Predictions for a patient’s vulnerability to a particular medical condition
- More accurate health insurance rates
- Improved scheduling for both patients and staff
- Optimized resource allocation
- More efficient decision-making at the business and patient care level