theknowledgeacademy

# Data Science with R Training

Data science

# About The Knowledge Academy

The world's largest provider of classroom and online training courses

- ✓ World Class Training Solutions
- ✓ Subject Matter Experts
- ✓ Highest Quality Training Material
- ✓ Accelerated Learning Techniques
- ✓ Project, Programme, and Change Management, ITIL® Consultancy
- ✓ Bespoke Tailor Made Training Solutions
- ✓ PRINCE2®, MSP®, ITIL®, Soft Skills, and More

theknowledgeacademy

# Course Syllabus

- Module 1: Introduction to R
- Module 2: Data Structures in R
- Module 3: Working with Data in R
- Module 4: Data manipulation in R
- Module 5: Data visualisation in R
- Module 6: Statistics in R
- Module 7: Machine Learning with R

theknowledgeacademy

# Module 1


## Description

| | |
|---|---|
| 1 | What is R? |
| 2 | R Installation and Dashboard Overview |
| 3 | Variables |
| 4 | Data Types |
| 5 | Operators |
| 6 | Conditional Statements |

theknowledgeacademy

# Module 1

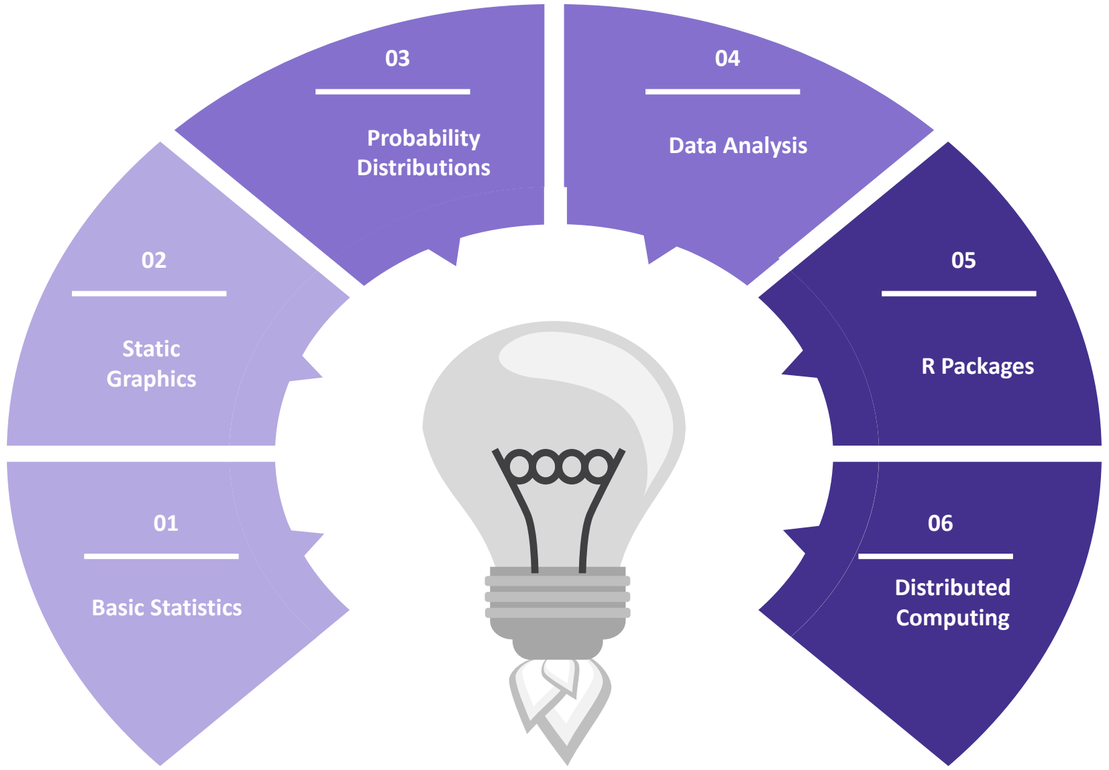

## Description

# What is R?

- ✓ R is a popular open-source programming language for statistical software and data analysis.
- ✓ R programming is widely used in machine learning, statistics, and data analysis. R makes it simple to create objects, functions, and packages.
- ✓ R is highly extensible and offers a wide range of statistical (linear and nonlinear modelling, classical statistical tests, classification, clustering,...) and graphical techniques.
- ✓ R's ease of producing well-designed publication-quality plots, including mathematical symbols and formulae where necessary, is one of its strengths.

# What is R?(Cont.)

**Features of R:**



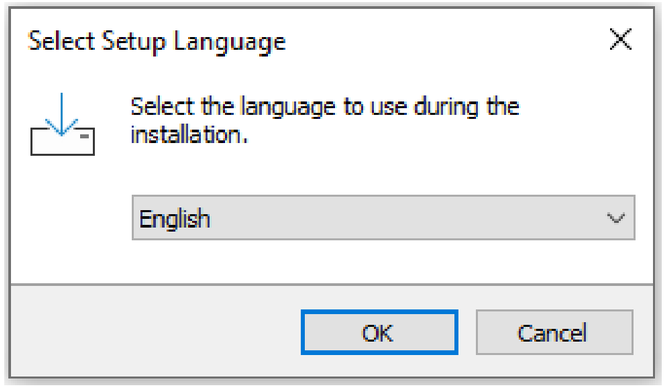
# R Installation

**Installation of R**

- ✓ RStudio is a R integrated development environment that makes it easier to use R. It comes with a console, a code editor, and plotting tools.
- ✓ Rstudio as well as R language both are required in order to work on R Language.

**Installing R Language**

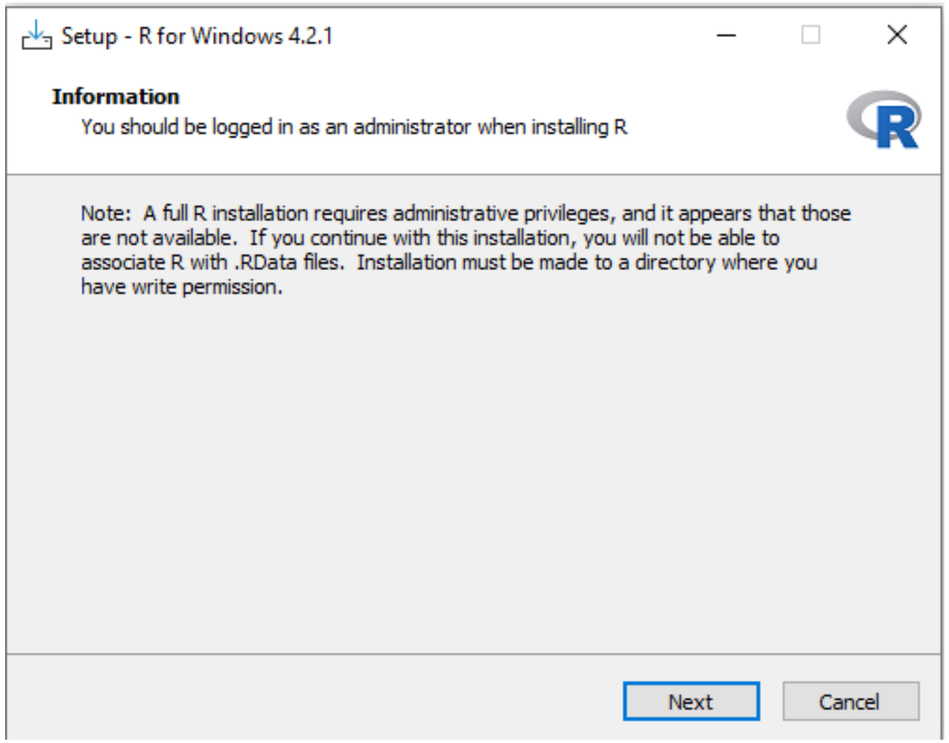
- ✓ In order to install the language setup, Nagivate to : https://cran.r-project.org/bin/windows/base/
- ✓ After that download and run the setup

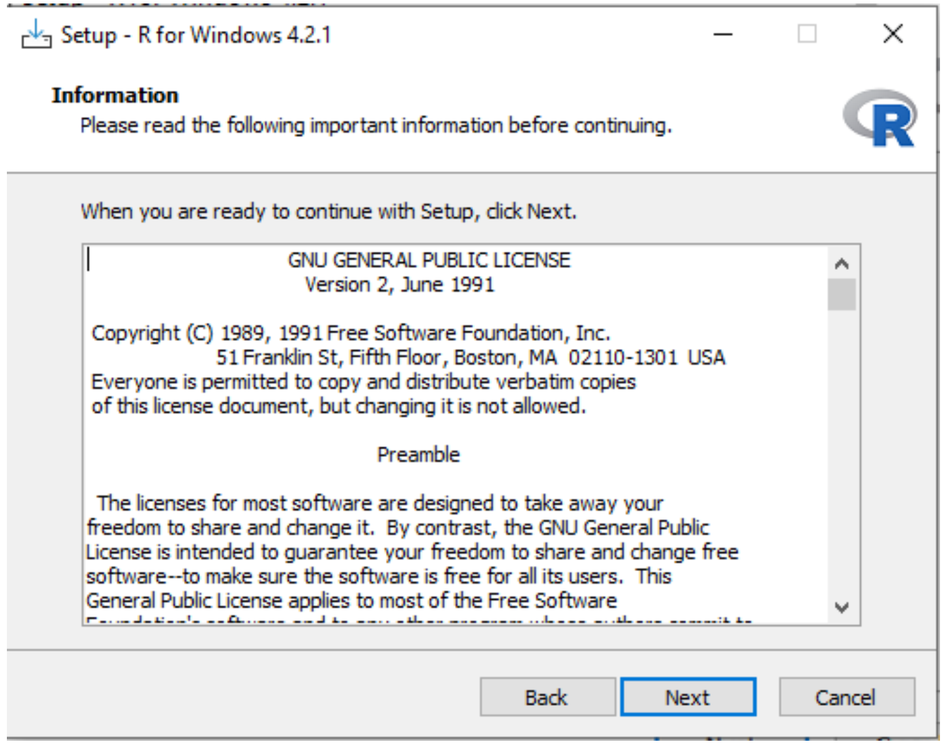


# R Installation

**Installation of R**

- ✓ After that, Click on Next to begin with the Installation

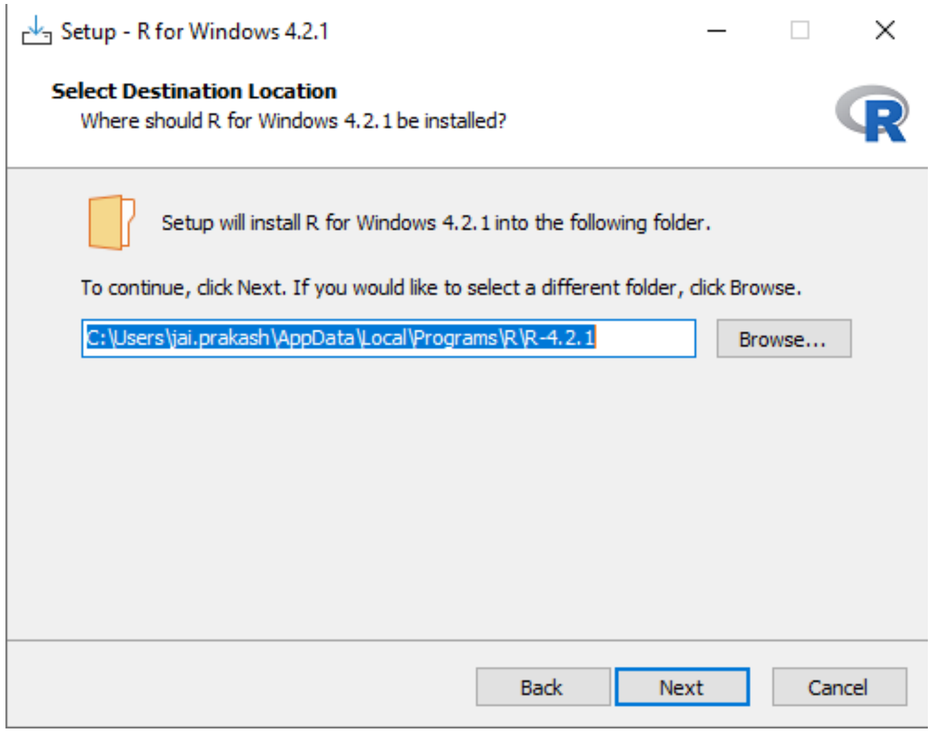


# R Installation

**Installation of R**

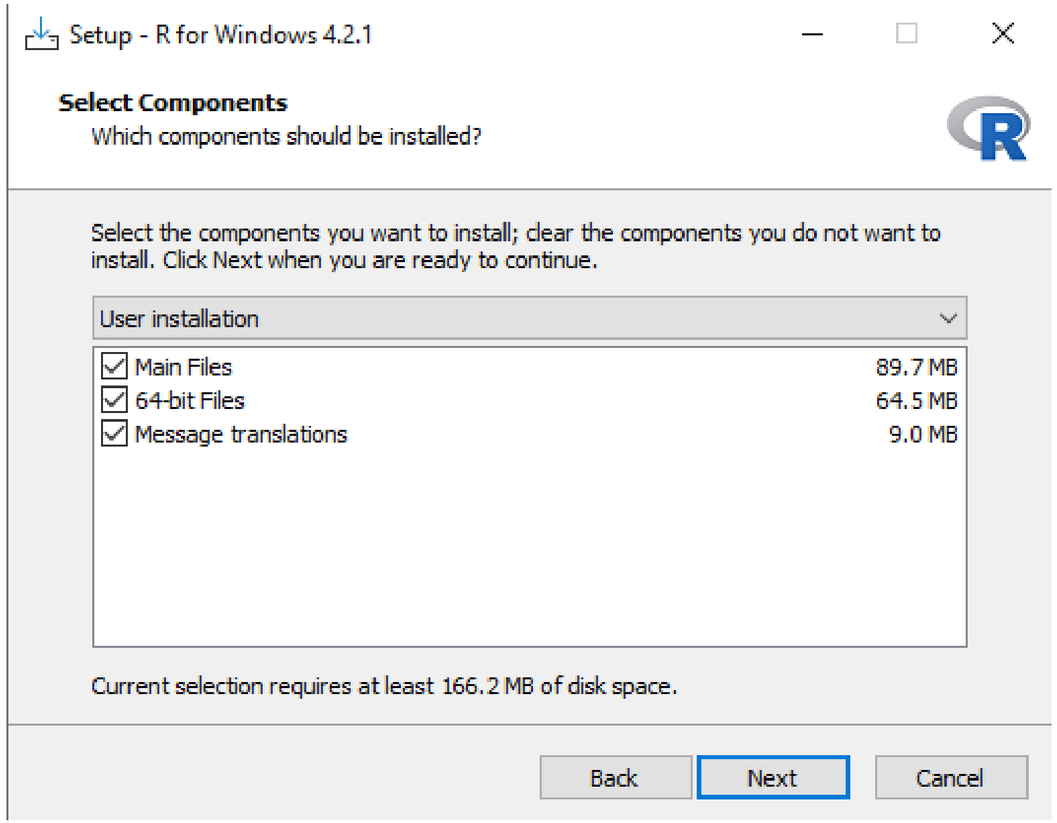
- ✓ After that, Select the folder or directory for installation and press Next.



# R Installation

**Installation of R**

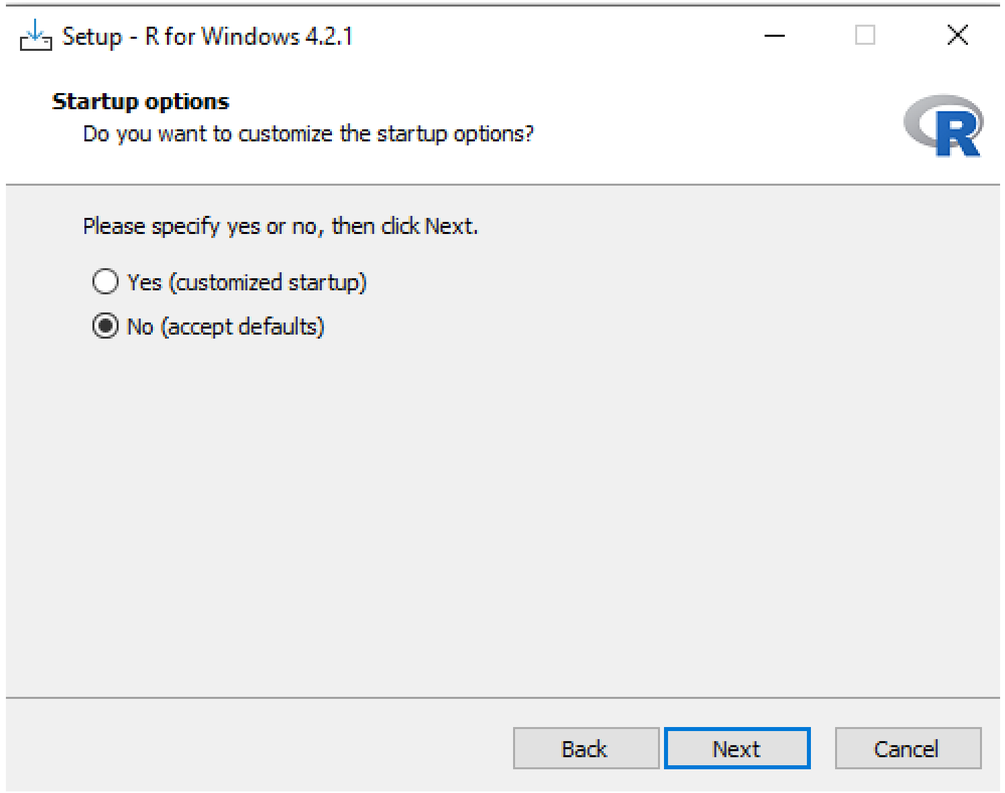
- ✓ After that, Select all of the components for Installation and click next



# R Installation

**Installation of R**

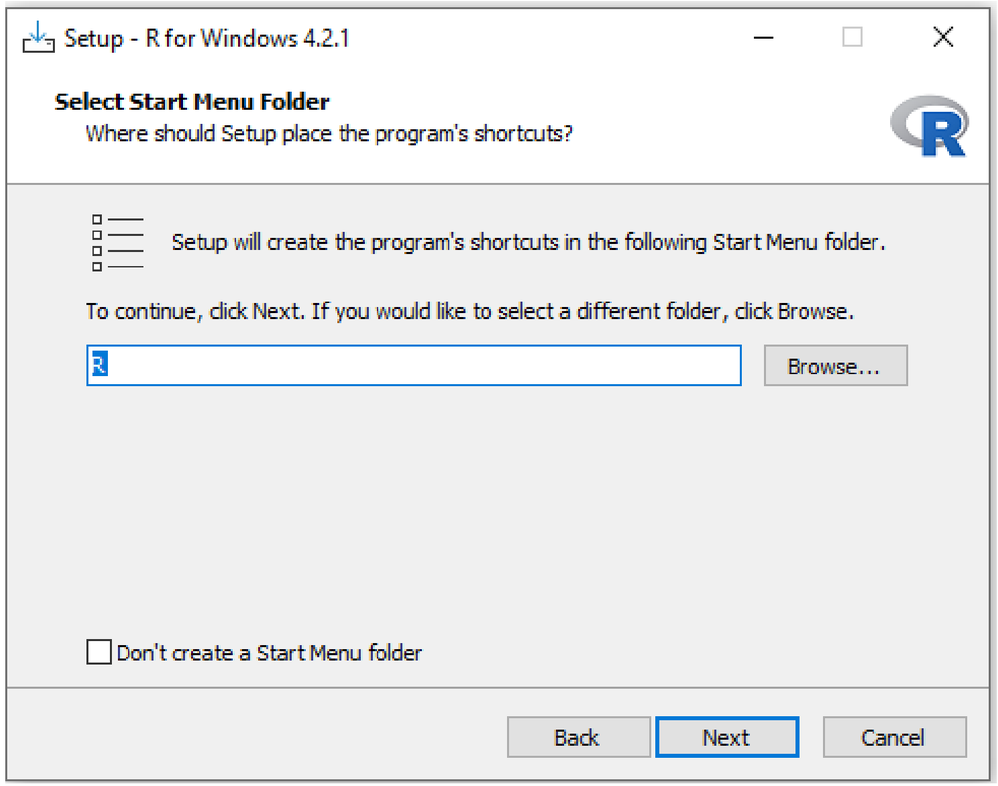
- ✓ After that, Select all the default settings option and Click on Next



# R Installation

**Installation of R**

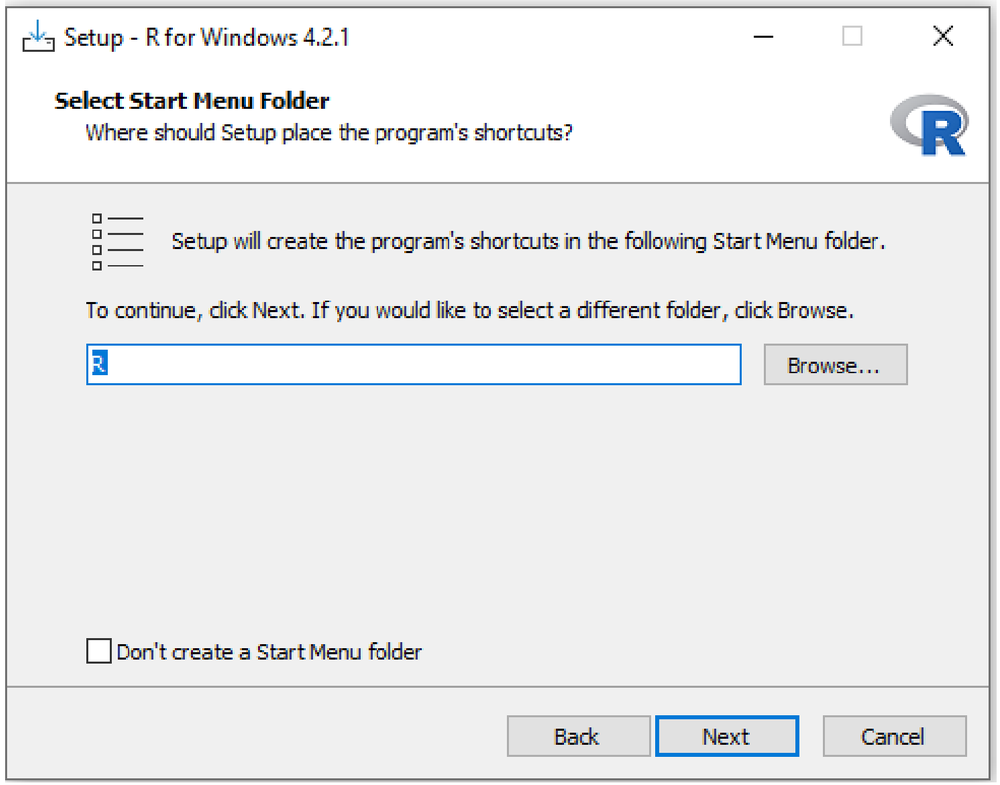
- ✓ After that, Select the Start Menu folder Name and Click on Next

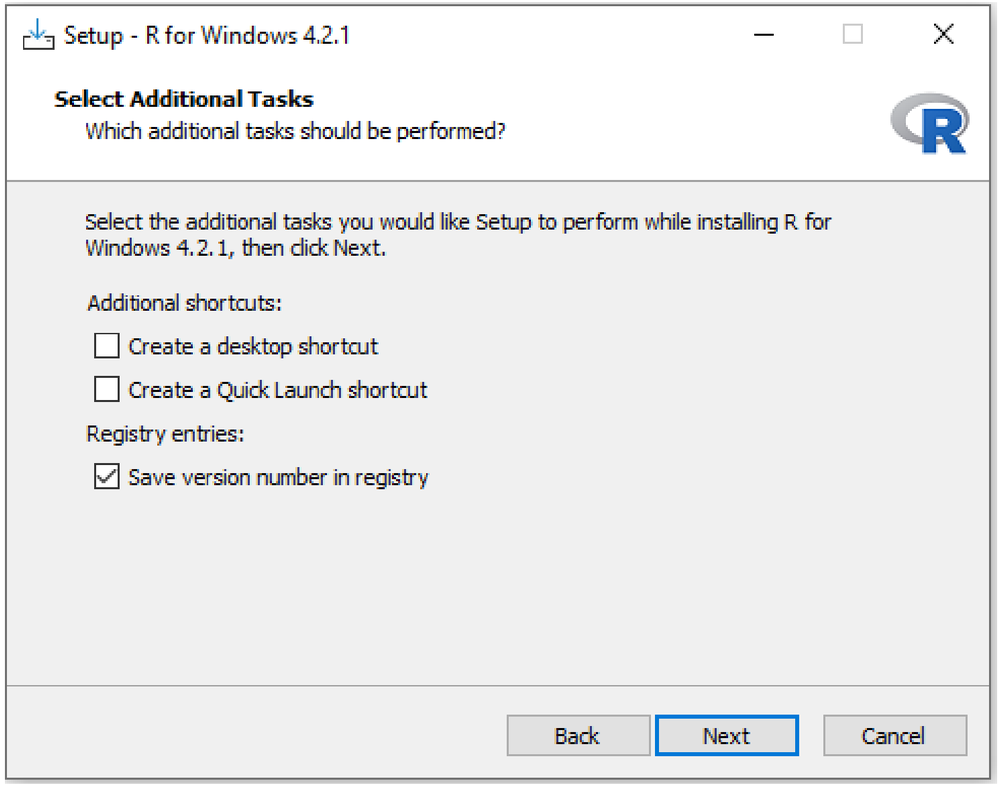


# R Installation

## Installation of R

- ✓ After that, Select the Start Menu folder Name and Additional Options and Click on Next

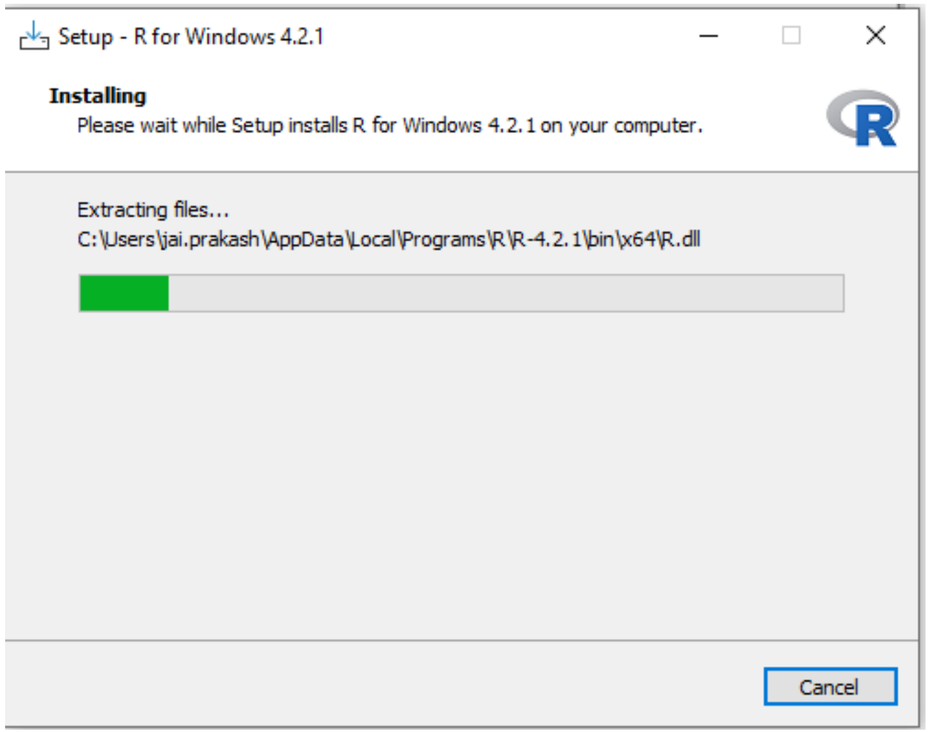


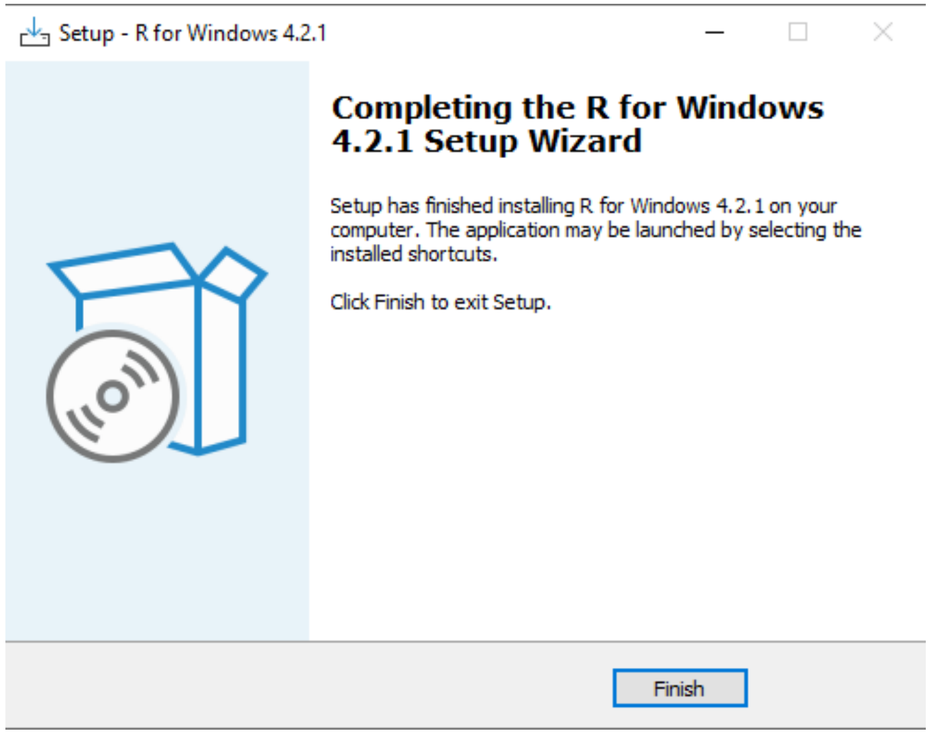



# R Installation

**Installation of R**

- ✓ After that, Let the Installation setup complete and finally click on finish to install the R Language.

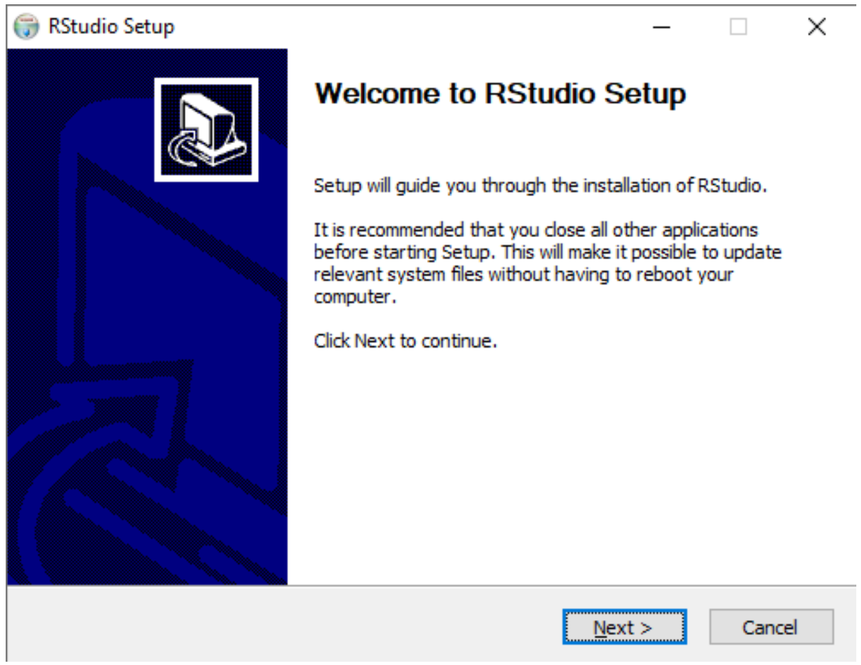


# R Installation

**Installation of R Studio**

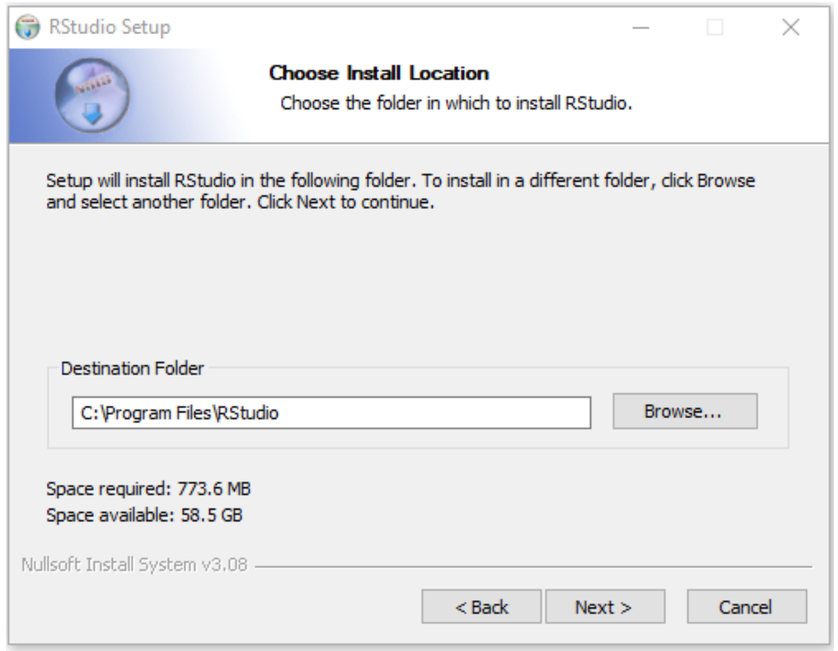
- ✓ After the Installation of R Language, RStudio Setup requires Installation
- ✓ In order to Install the Rstudio, Navigate to : https://www.rstudio.com/products/rstudio/download/ and select the package according the operating system and download it.
- ✓ After downloading the package, run it.

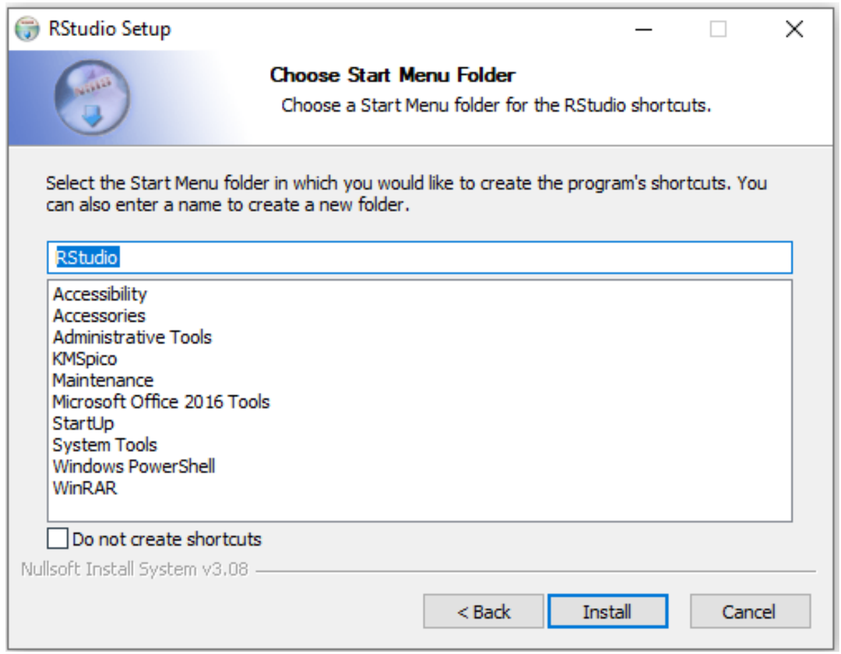


# R Installation

**Installation of R Studio**

- ✓ After Clicking on Next, Follow all of the necessary Steps by selecting the options and clicking on Next.

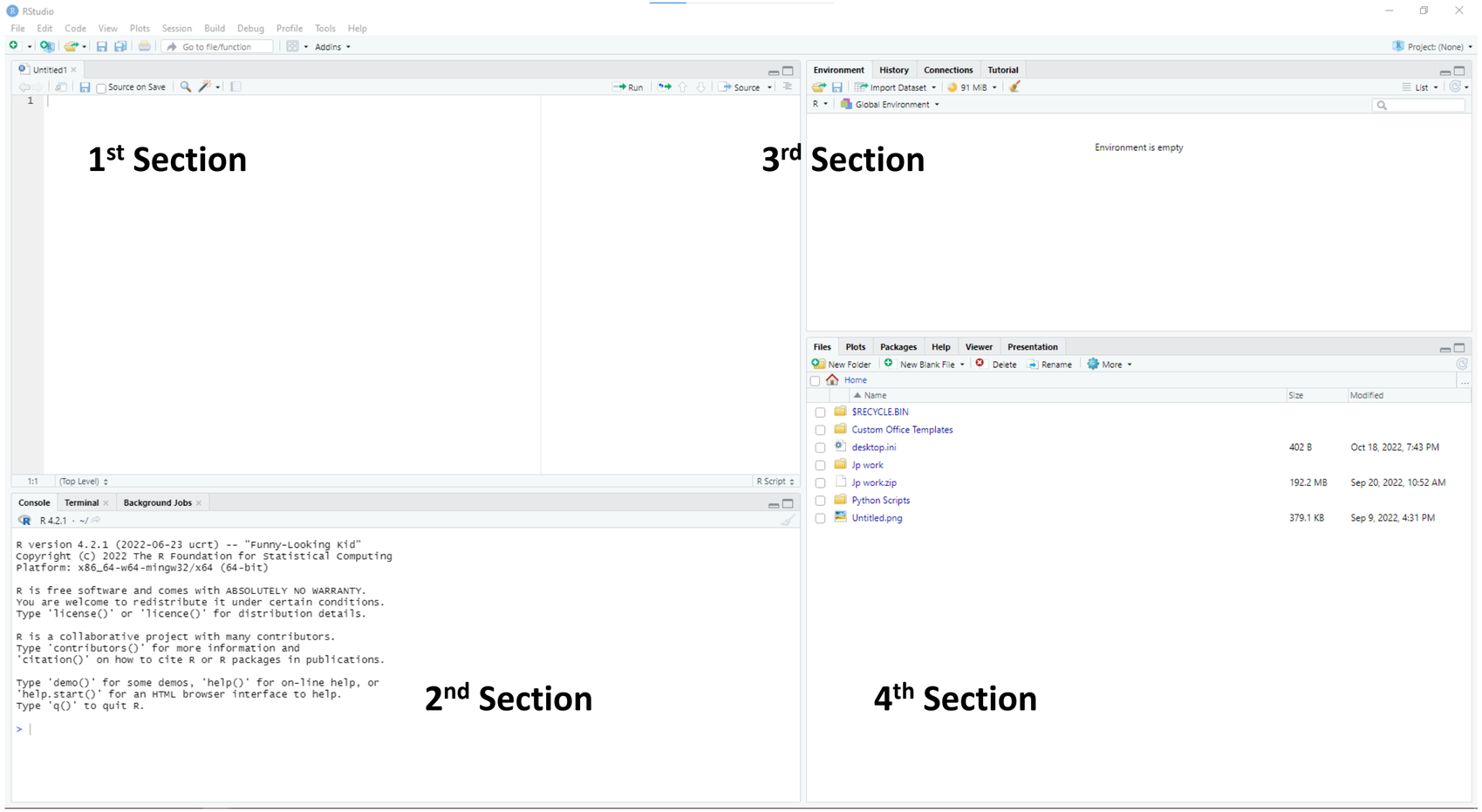


# R Dashboard Overview

**RStudio Dashboard Overview**

- ✓ Rstudio comes with multiple sections. The most important one's are:
  - ✓ File section
  - ✓ Console section
  - ✓ Environment section
  - ✓ Directory section



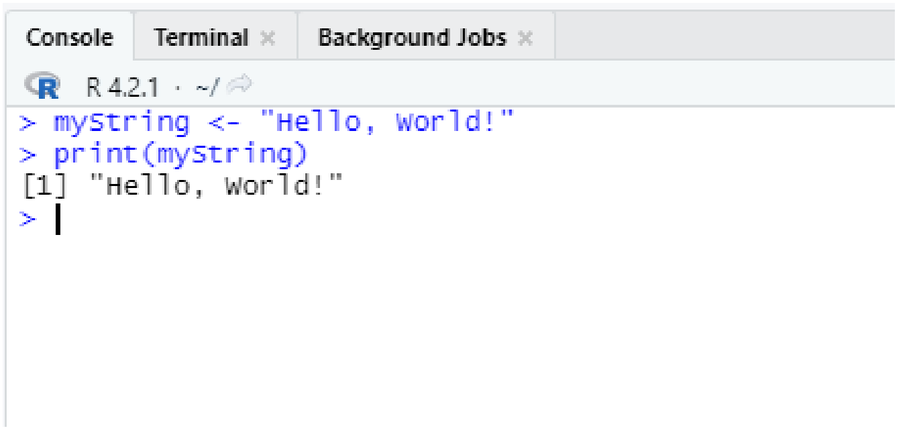
# R Fundamentals

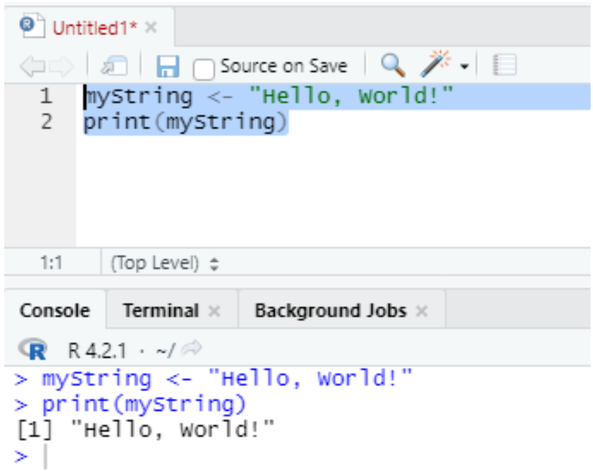
**Basic Syntax**

- ✓ To begin learning R programming, we will create a "Hello, World!" program.
- ✓ Depending on your needs, you can write your program at the R command prompt or in an R script file. Let's go over them one by one.

**Using R Prompt and R Script File**

- ✓ We can program the same code using the R console on Rstudio for smaller calculations but when it comes to work on Larger Programs, An R Script File is required.

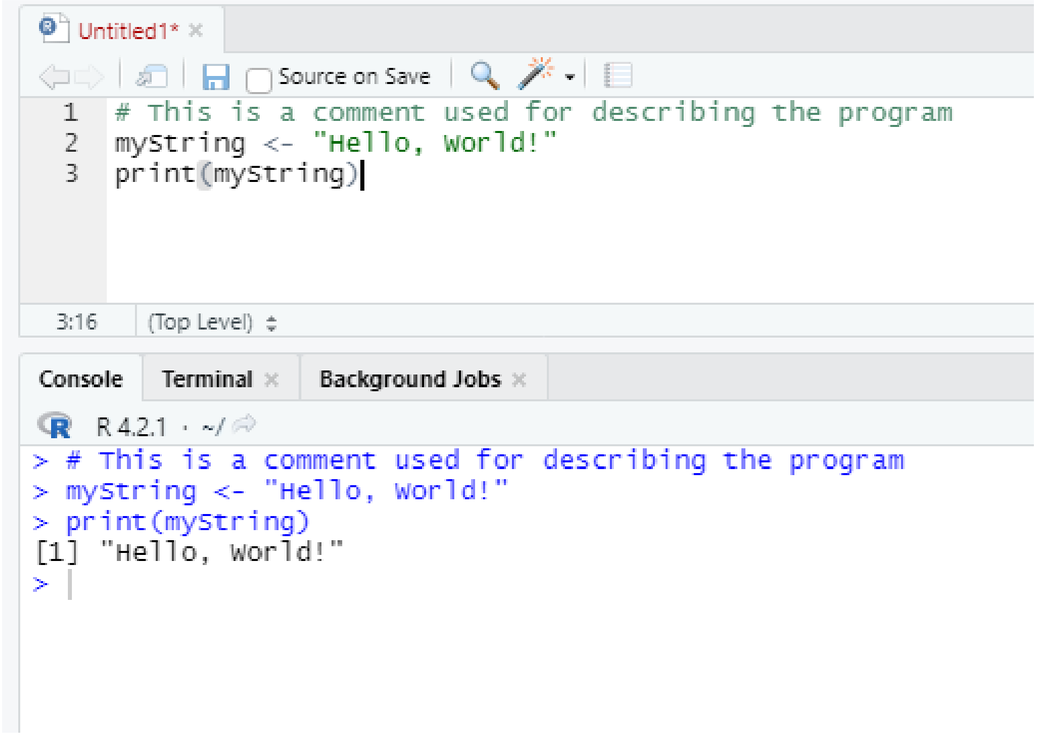


# R Fundamentals

**R Comments**

- ✓ Comments are similar to help text in your R programme, and they are ignored by the interpreter while your programme is being executed. As shown below, a single comment is written using # at the beginning of the statement.



# R Variables

**Introduction**

- ✓ Variables are used to store data or information in the R program. A vector, a matrix, or a combination of many R objects can be stored in the R variable. R is dynamically typed, which means it checks the type of data type when the statement is run.

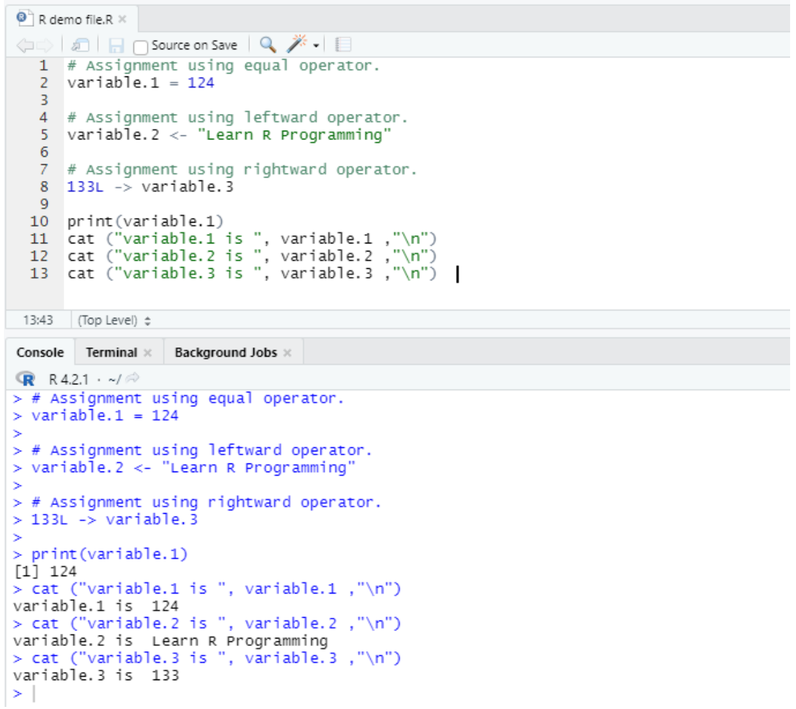
```
# myString is a variable here
myString <- "Hello, World!"
print(myString)
```

```
# "Hello, World!" is value
myString <- "Hello, World!"
print(myString)
```

# R Variables

**Value assignment to a variable**

- ✓ There are three operators in R programming that we can use to assign values to variables. For this, we can use the leftward, rightward, and equal to operators.
- ✓ Printing the value of a variable is accomplished using two functions: print() and cat (). The cat() function concatenates multiple values into a single print output.



# R Variables

**Rules to declare a variable**

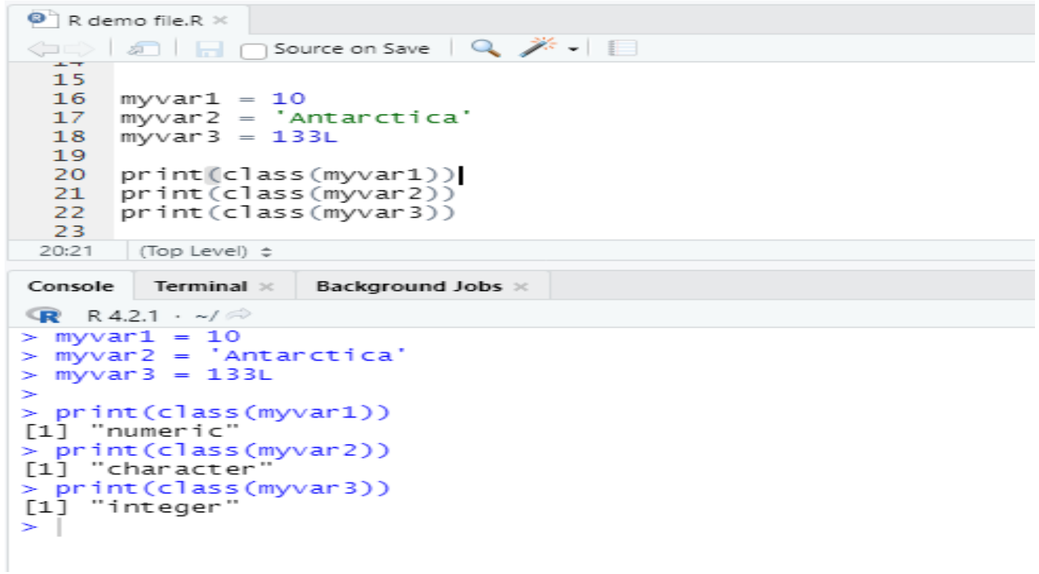
- ✓ A valid variable name includes letters, numbers, dots, and underlines.
- ✓ A variable name should begin with a letter or a dot which is not followed by a number.
- ✓ Some of the sample variable names are:

| Name of variable | Validity | Reason for valid and invalid |
|---|---|---|
| **_var_name** | Invalid | Variable name can't start with an underscore(_). |
| **var_name, var.name** | Valid | Variable can start with a dot, but dot should not be followed by a number. In this case, the variable will be invalid. |
| **var_name%** | Invalid | In R, we can't use any special character in the variable name except dot and underscore. |
| **2var_name** | Invalid | Variable name cant starts with a numeric digit. |
| **.2var_name** | Invalid | A variable name cannot start with a dot which is followed by a digit. |
| **var_name2** | Valid | The variable contains letter, number and underscore and starts with a letter. |

# R Variables

**Checking the Datatype of a variable**

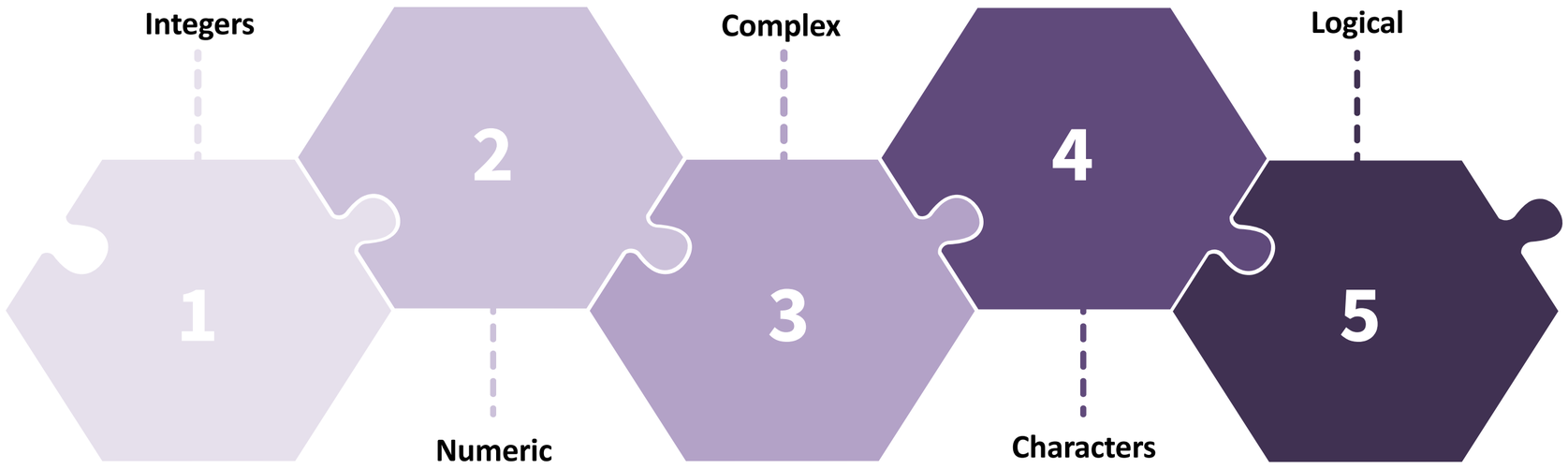
- ✓ R programming is a dynamically typed language, which means we can change the data type of the same variable in our program multiple times.
- ✓ A variable is not declared of any data type due to its dynamic nature. It obtains the data type to be assigned to the variable from the R-object.
- ✓ In order to check the datatype of a variable, R has a function named **class()** which provides information about the datatype.



# R Datatypes

**Introduction to Datatypes**

- ✓ In programming languages, we must use variables to store various types of information.
- ✓ Because information can be of various types, different data types are created to accommodate the various types of information.
- ✓ R consist of the following datatypes:



# R Datatypes

**Integer**

- ✓ Integers are numerical values without decimal.
- ✓ ‘L’ is used to describe R language that an integer is created.
- ✓ Integers declared without ‘L’ are considered as numerical datatype or floating point value.
- ✓ Syntax to declare an Integer:

```
> #Integer declaration
> num1 = 33L
> print(class(num1))
[1] "integer"
>
```

```
> num2 = 33.00L
Warning message:
integer literal 33.00L contains unnecessary decimal point
>
```

# R Datatypes

**Numeric**

- ✓ Numeric datatypes consist of numerical values with or without decimal.
- ✓ value with numbers is by default of numeric datatype in R
- ✓ Syntax to declare a numeric:

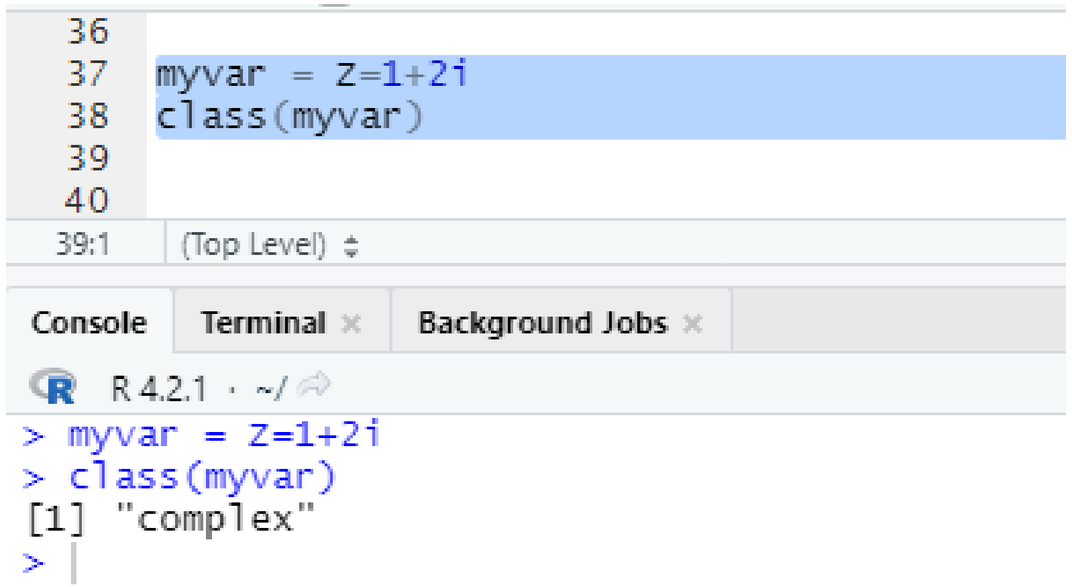
```
> nn = 2343
> class(nn)
[1] "numeric"
>
```

```
> nm = 7656.45
> class(nm)
[1] "numeric"
>
```

# R Datatypes

**Complex**

- ✓ Complex datatype consist of numerical values with **I** as an imaginary value.
- ✓ Complex datatypes are rarely used in R because of their Mathematical constraints.
- ✓ Syntax to declare a complex value:

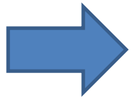


# R Datatypes

**Characters**

- ✓ Character datatype represent strings in R, whereas Strings are group of characters.
- ✓ A value of Character datatype is declared be enclosing under ‘’ or “” and characters can include alphanumeric values.
- ✓ Syntax to declare a character value:

```
a = 'a'
b = "good"
c = "TRUE"
d = '35.4'
print(class(a))
print(class(b))
print(class(c))
print(class(d))
```



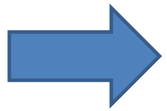
```
> print(class(a))
[1] "character"
> print(class(b))
[1] "character"
> print(class(c))
[1] "character"
> print(class(d))
[1] "character"
>
```

# R Datatypes

**Logical**

- ✓ Logical datatype is a special datatype with only two possible values i.e. True / False
- ✓ Logical datatype values are very important while working on the programming pathways, which means that while constructing the whole program, the True/False values play an important part.
- ✓ Syntax to declare a Logical value:

```
bool1 = TRUE
bool2 = FALSE
print(class(bool1))
print(class(bool2))
```

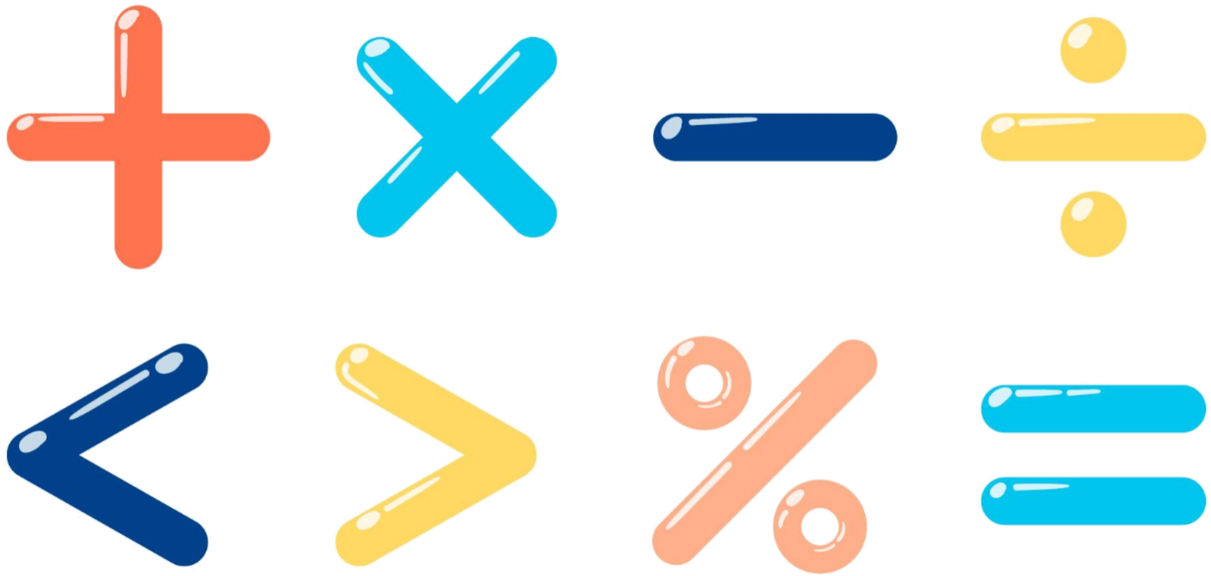


```
> print(class(bool1))
[1] "logical"
> print(class(bool2))
[1] "logical"
>
```

# R Operators

**Introduction**

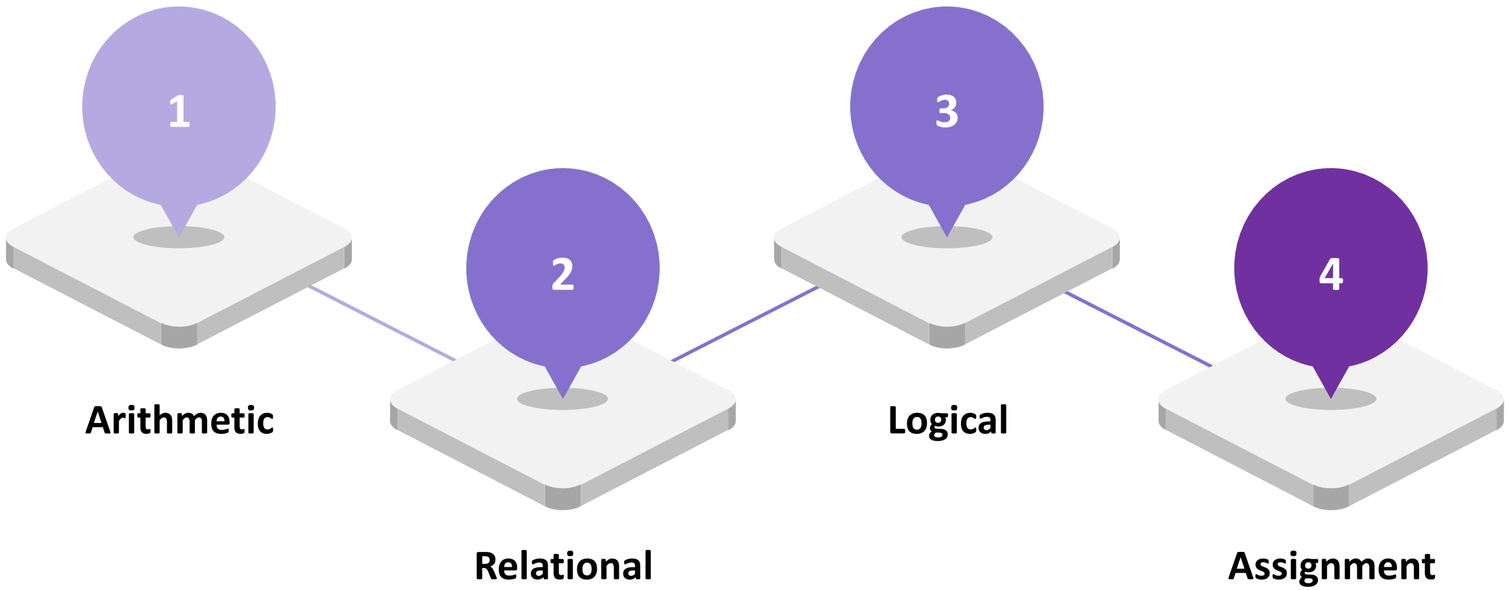
- ✓ An operator is a symbol in computer programming that represents an action. An operator is a symbol that instructs the compiler to perform certain logical or mathematical operations. R programming has a plethora of built-in operators.
- ✓ There are various types of operators in R programming, and each operator performs a different task. There are some advanced operators for data manipulation, such as model formula and list indexing.



# R Operators

**Operators in R**

- ✓ R covers a lot of operations on the basis of operators only and there are different categories of operators.
- ✓ R covers the following categories of operators:



# R Operators

**Arithmetic Operators**

- ✓ The symbols used to represent arithmetic math operations are known as arithmetic operators. Each element of the vector is affected by the operators.
- ✓ R provides support for a number of arithmetic operators.

| Operator | Working |
|---|---|
| + | Adds two values |
| - | Subtract two values |
| * | Multiply Two values |
| / | Divide two values |
| %% | Provides remainder of division |
| %/% | Provides Quotient of division |
| ^ | Exponential operator |

# R Operators

**Arithmetic Operators Examples**

```
a = 1 + 1
b = 34.55 - 23.54
c = 4L * 3L
d = 43/23
e = 32 %% 3
f = 234 %/% 11
g = 3^3

print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
> print(a)
[1] 2
> print(b)
[1] 11.01
> print(c)
[1] 12
> print(d)
[1] 1.869565
> print(e)
[1] 2
> print(f)
[1] 21
> print(g)
[1] 27
>
```

# R Operators

**Relational Operators**

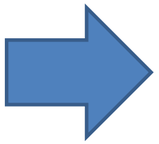
- A relational operator is a symbol that defines a relationship between two entities. These include both numerical equality and inequity. Each element of the first vector is compared to the corresponding element of the second vector by a relational operator. The comparison will produce a Boolean value. R provides support for the following relational operators:.

| Operator | Working |
|---|---|
| > | Return True when every value in the first vector is greater the second vector |
| < | Return True when every value in the first vector is smaller the second vector |
| <= | Return True when every value in the first vector is smaller or equal than the second vector |
| >= | Return True when every value in the first vector is greater or equal than the second vector |
| == | Return TRUE when both values are exactly equal |
| != | Return TRUE when both values are not exactly equal |

# R Operators

**Relational Operators Examples**

```
a = 1 > 1
b = 34.55 < 23.54
c = 4L >= 3L
d = 43 <= 23
e = 33  == 33
f = 234 != 11

print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
```



```
> print(a)
[1] FALSE
> print(b)
[1] FALSE
> print(c)
[1] TRUE
> print(d)
[1] FALSE
> print(e)
[1] TRUE
> print(f)
[1] TRUE
>
```

# R Operators

**Logical Operators**

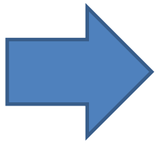
- ✓ The logical operators enable a program to make a decision based on a number of conditions. Each operand in the program is regarded as a condition that can be evaluated to a false or true value. The overall value of the op1 operator op2 is determined by the value of the conditions. Logical operators apply to vectors of the logical, numeric, or complex types.
- ✓ Each element of the first vector is compared to the corresponding element of the second vector by the logical operator. R provides support for the following types of operators:

| Operator | Meaning | Working |
|---|---|---|
| **&** | **Logical AND** | Return True if first value of both vectors are TRUE |
| **\|** | **Logical OR** | Return True if any from first value of both values is TRUE |
| **!** | **Logical NOT** | Return True when first value of a vector is FALSE and vice – versa |
| **&&** | **AND** | Return True as final result if first value of both vectors are TRUE |
| **\|\|** | **OR** | Return True as final result if any of the first value of both vectors are TRUE |

# R Operators

**Logical Operators Examples**

```
a <- c(3, 0, 2+2i)
b <- c(2, 4, 2+3i)

print(a&b)
print(a|b)
print(!a)
print(a&&b)
print(a||b)
```



```
> a <- c(3, 0, 2+2i)
> b <- c(2, 4, 2+3i)
> print(a&b)
[1]  TRUE FALSE  TRUE
> print(a|b)
[1] TRUE TRUE TRUE
> print(!a)
[1] FALSE  TRUE FALSE
> print(a&&b)
[1] TRUE

> print(a||b)
[1] TRUE
```

# R Operators

**Assignment Operators**

- ✓ An assignment operator is used to give a variable a new value. Vectors are assigned values using these operators in R.

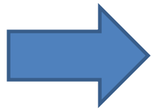
  There are several types of assignments.

| Operator | Meaning | Working |
|---|---|---|
| **<- or =** | **Left Assignment** | Assigns value on RHS to variable on LHS |
| **->** | **Right Assignment** | Assigns value on LHS to variable on RHS |

# R Operators

**Assignment Operators Examples**

```
a <- c(3, 0, 2+2i)
b = c(2, 4, 2+3i)
c(1,2,3,4,5) -> c

print(a)
print(b)
print(c)
```

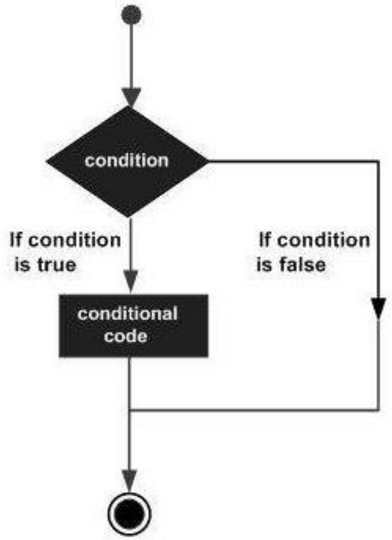


```
> print(a)
[1] 3+0i 0+0i 2+2i
> print(b)
[1] 2+0i 4+0i 2+3i
> print(c)
[1] 1 2 3 4 5
>
```

# R Conditional Statements

**Introduction**

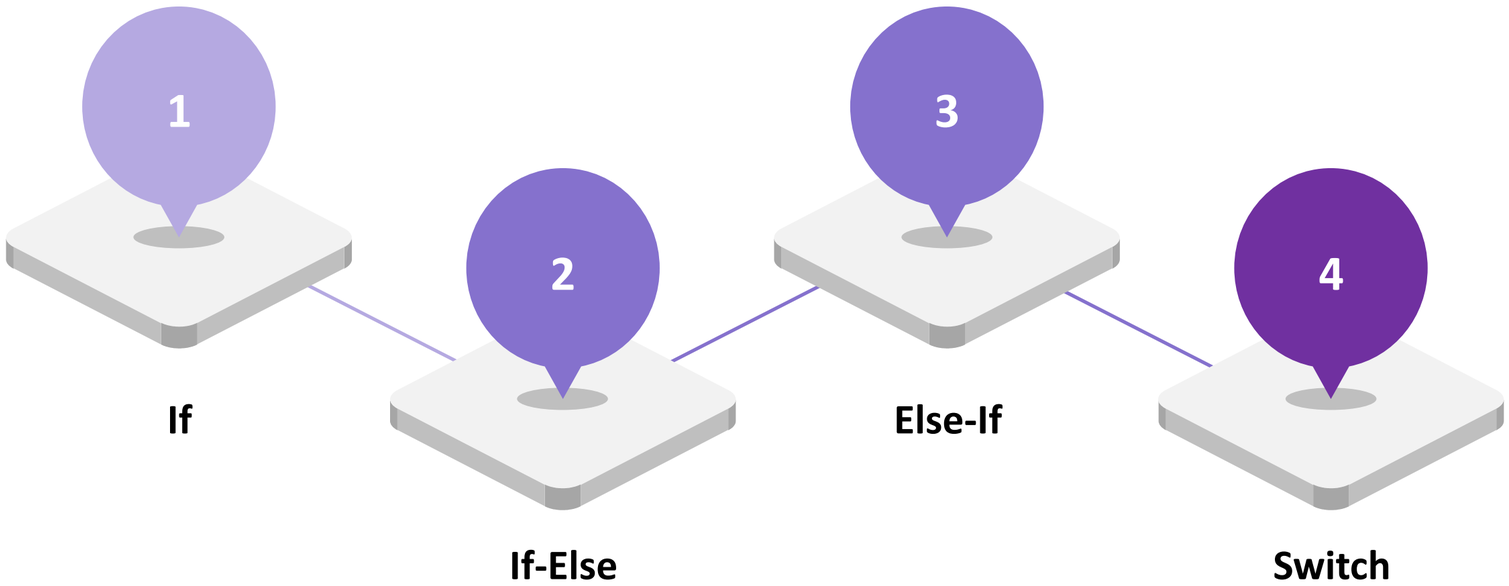
- ✓ Conditional statements are block of code that helps in creating a path in a program. These are generally a group of commands whose execution is based on the result of the provided condition and the result of condition is TRUE or FALSE in nature.
- ✓ The general form of a typical conditional structure found in most programming languages is shown below.



# R Conditional Statements

**Types of Conditional Statements**

- ✓ R consist of multiple conditional statements which accommodate a lot of scenarios in programming. Below is the list of conditional statements in R:

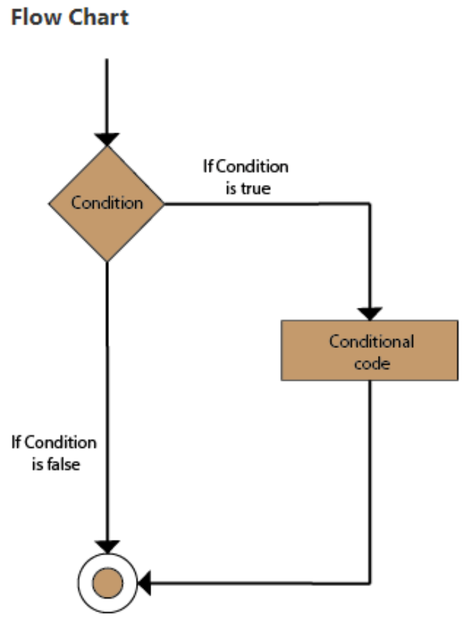


# R Conditional Statements

**If Statement**

- ✓ The Boolean expressions are followed by one or more statements in the if statement. The if statement is the most basic decision-making statement, allowing us to make a decision based on a condition.
- ✓ The if statement is a conditional programming statement that performs a function and displays information if the condition is met. The syntax and flow chart of If statement in R is:

```
if(boolean_expression) {
  // If the boolean expression is true, then statement(s) will be executed.
}
```

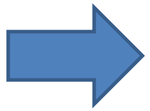


# R Conditional Statements

**If Statement example**

- ✓ The if statement is useful while working on a single condition. The following is an example of simple if statement:

```
x <-224L
y <- 34L
if(x > y)
{
  print("x is greater than y")
}
```



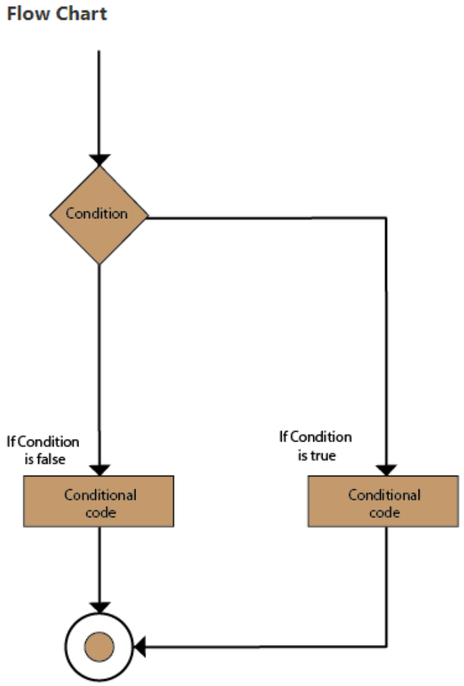
```
> x <-224L
> y <- 34L
> if(x > y)
+ {
+   print("x is greater than y")
+ }
[1] "x is greater than y"
>
```

# R Conditional Statements

**If - Else Statement**

- ✓ In other words, if a Boolean expression returns true, the if block is executed; otherwise, the else block is executed. R programming considers any non-zero and non-null values to be true, while zero and null values are considered false.
- ✓ The following is the basic syntax of an If-else statement:

```
if(boolean_expression) {
   // statement(s) will be executed if the boolean expression is true.
} else {
   // statement(s) will be executed if the boolean expression is false.
}
```

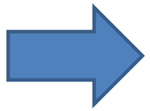


# R Conditional Statements

**If – Else Statement example**

- ✓ The if – else statement is useful while working on multiple conditions. The following is an example of if – Else statement:

```
a<- 100
#checking boolean condition
if(a<20){
  # if the condition is true then print the following
  cat("a is less than 20\n")
}else{
  # if the condition is false then print the following
  cat("a is not less than 20\n")
}
cat("The value of a is", a)
```



```
> a<- 100
> #checking boolean condition
> if(a<20){
+   # if the condition is true then print the following
+   cat("a is less than 20\n")
+ }else{
+   # if the condition is false then print the following
+   cat("a is not less than 20\n")
+ }
a is not less than 20
> cat("The value of a is", a)
The value of a is 100
>
```

# R Conditional Statements

**Else - If Statement**

- ✓ This is also known as a nested if-else statement. Following the if statement is an optional else if..... else statement. This statement tests multiple conditions in a single if.....else if statement. When using the if.....else if.....else statement, there are a few important points to remember. These are the following points:
  - ✓ If a statement has zero or one else statement, it must come after any other if statements.
  - ✓ The if statement can have multiple else if statements that come before the else statement.
  - ✓ When an else if statement succeeds, none of the remaining else if or else statements are tested.
- ✓ The following is the basic syntax of an If-else statement:

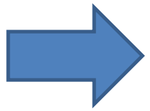
```
if(boolean_expression 1) {
   // This block executes when the boolean expression 1 is true.
} else if( boolean_expression 2) {
   // This block executes when the boolean expression 2 is true.
} else if( boolean_expression 3) {
   // This block executes when the boolean expression 3 is true.
} else {
   // This block executes when none of the above condition is true.
}
```

# R Conditional Statements

**Else – If Statement example**

- ✓ The if – else statement is useful while working on multiple conditions. The following is an example of if – Else statement:

```
age <- 35
if(age<18){
  print("You are child")
  }else if(age>30) {
  print("You are old guy")
  }else
  print("You are adult")
```



```
> age <- 35
> if(age<18){
+   print("You are child")
+   }else if(age>30) {
+   print("You are old guy")
+   }else
+   print("You are adult")
[1] "You are old guy"
>
```

# R Conditional Statements

**Switch Statement**

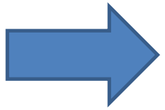
- ✓ A switch statement is a selection control mechanism that uses map and search to change the control flow of programme execution based on the value of an expression.
- ✓ The switch statement replaces long if statements that compare a variable to multiple integral values. It is a multi-way branch statement that allows you to easily dispatch execution for different sections of code. This code is based on the expression's value.
- ✓ The basic syntax of Switch statement is:

```
switch(expression, case1, case2, case3....)
```

# R Conditional Statements

**Switch Statement Example**

```
x <- switch(
   3,
   "Shubham",
   "Nishka",
   "Gunjan",
   "Sumit"
)
print(x)
```

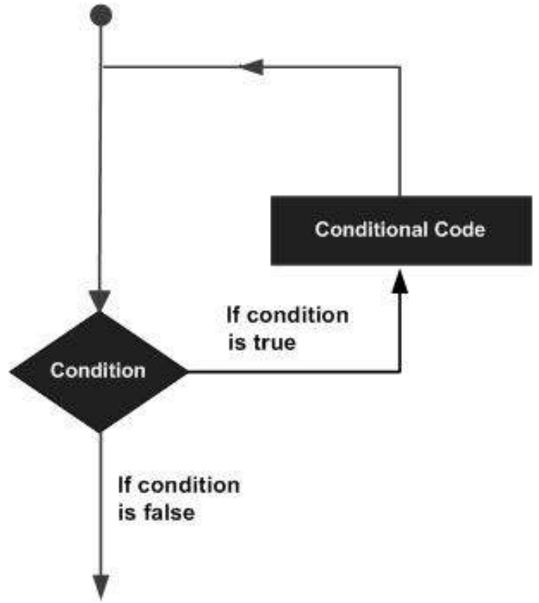


```
> x <- switch(
+    3,
+    "Shubham",
+    "Nishka",
+    "Gunjan",
+    "Sumit"
+ )
> print(x)
[1] "Gunjan"
>
```

# R Looping Statements

**Introduction**

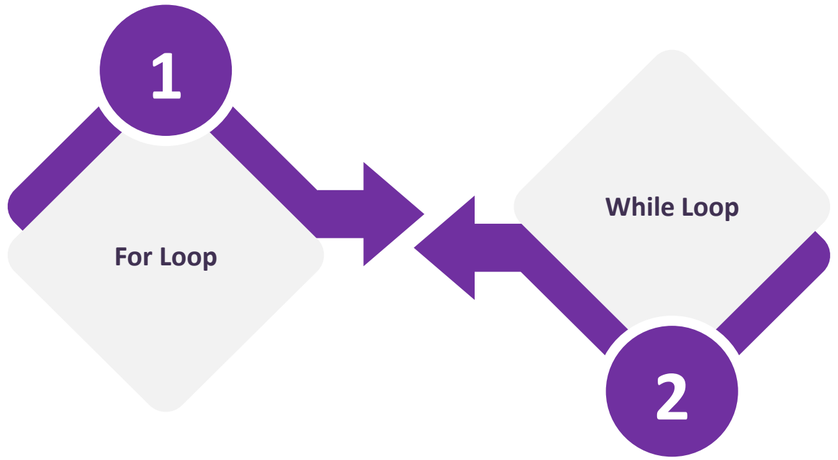
- ✓ Code Statements are typically executed sequentially but there may be times when you need to execute a block of code numerous times. A loop statement allows us to execute code statements multiple times, and the general form of a loop statement in most programming languages is as follows:



# R Looping Statements

**Types of Looping Statements**

- ✓ Loop control statements alter the execution sequence. All automatic objects created in that scope are destroyed when execution leaves that scope. The following control statements are supported by R.

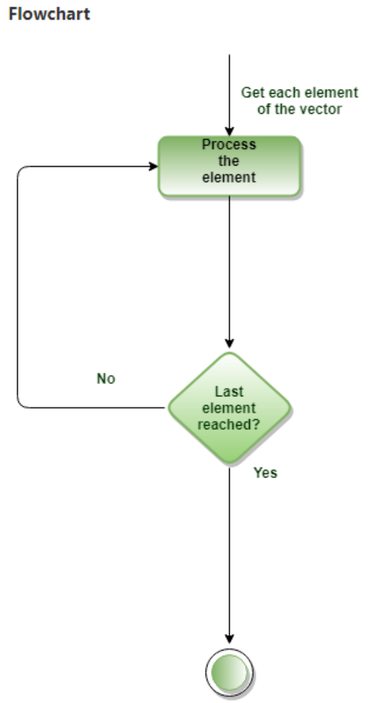


# R Looping Statements

**For Loop**

- ✓ A for loop in R is a method of repeating a sequence of instructions under certain conditions. It enables us to automate parts of our code that require repetition.
- ✓ A for loop is essentially a repetition control structure. It enables us to efficiently write the loop that must be executed a set number of times. The Syntax of For loop is given below:

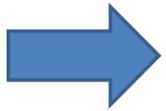
```
for (value in vector) {
   statements
}
```



# R Looping Statements

**For Loop example**

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



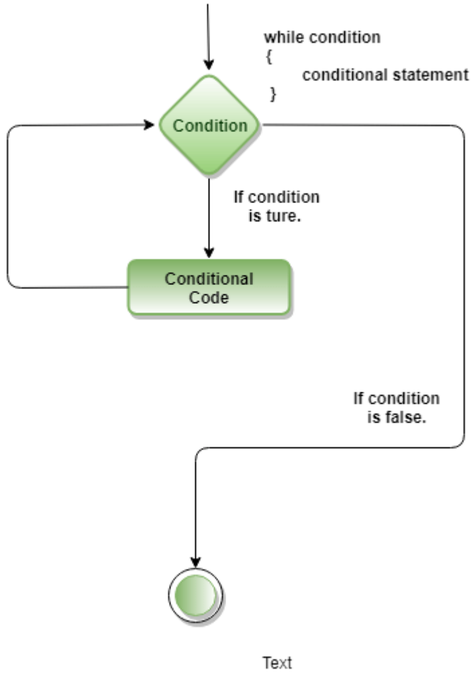
```
> for(i in 1:10){
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
>
```

# R Looping Statements

**While Loop**

- ✓ A while loop is a type of control flow statement that is used to repeatedly iterate a block of code. The while loop is terminated when the Boolean expression returns false.
- ✓ In a while loop, the condition is checked first, and then the body of the statement is executed. The condition will be checked n+1 times rather than n times in this statement.
- ✓ The while loop's basic syntax and flow chart is as follows:

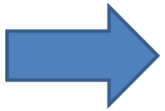
```
while (test_expression) {
   statement
}
```



# R Looping Statements

**While Loop example**

```
c = 1
while(c < 10){
   print(c)
   c = c + 1
}
```



```
> c = 1
> while(c < 10){
+   print(c)
+   c = c + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
>
```

# R Functions

**Introduction**

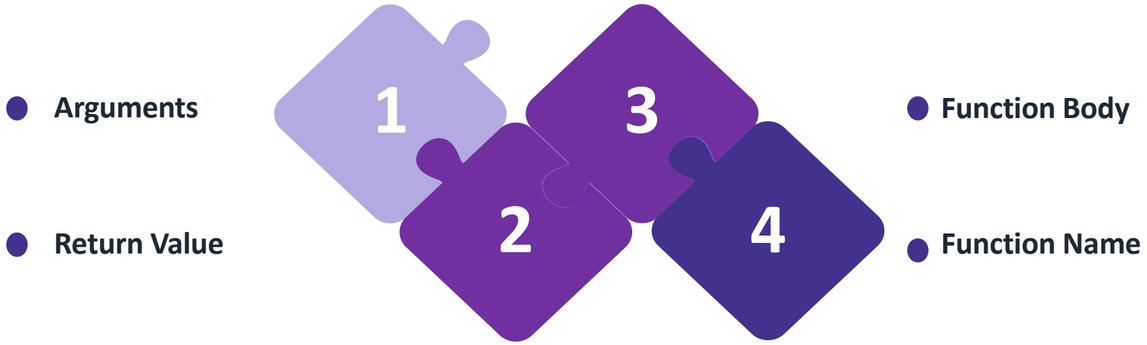
- ✓ A function is a collection of statements that are organised to perform a specific task. R includes a number of built-in functions and allows the user to create their own. In the modular approach, functions are used to carry out tasks.
- ✓ Functions are used to reduce complexity and avoid repeating the same task. To make our code easier to understand and maintain, we use the function to divide it into smaller chunks. A function should be:
  - ✓ Written to complete a specific task.
  - ✓ Arguments may or may not occur.
  - ✓ Consists of a body where our code is written.
  - ✓ It is possible that one or more output values will be returned.

```
func_name <- function(arg_1, arg_2, ...) {
  Function body
}
```

# R Functions

**Components of a function**

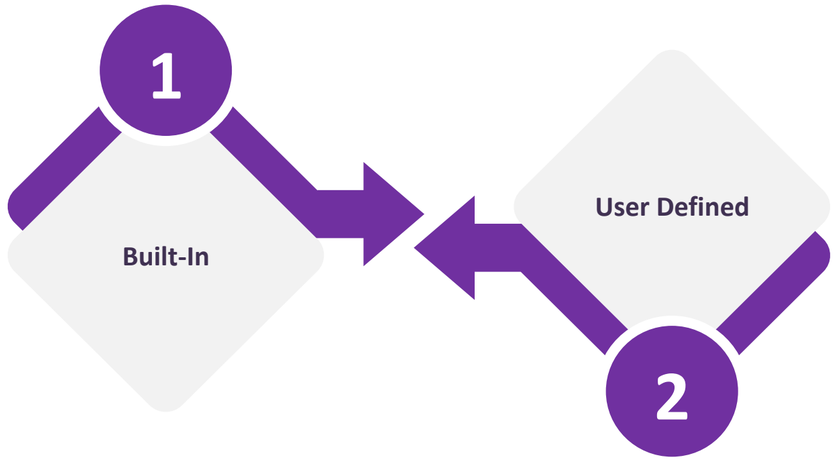
- ✓ There are 4 components of a function:
  - ✓ Function Arguments
  - ✓ Function Return Values
  - ✓ Function Body
  - ✓ Function Name



# R Functions

**Types of function**

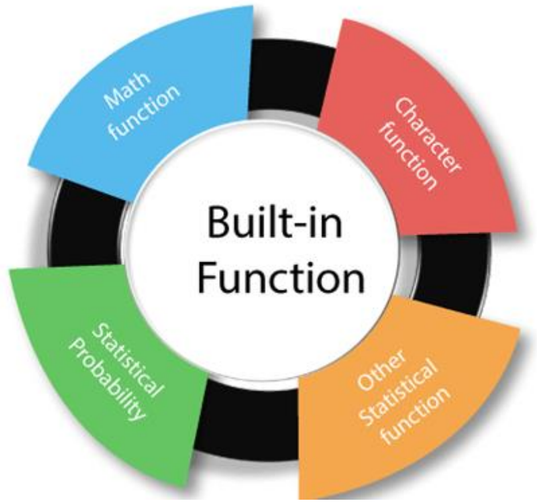
- ✓ There are 2 Types of function:
  - ✓ Built-In Functions
  - ✓ User Defined Functions



# R Functions

**Built-In Functions**

- ✓ Built-in functions are functions that are already created or defined in the programming framework. These functions are built into an application and do not require the user to create them.
- ✓ These functions are accessible to end users by simply calling them. R includes a variety of built-in functions such as seq(), mean(), max(), and sum(x), among others.

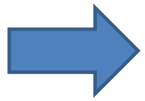


# R Functions

**Built-In Functions Example**

- ✓ R contains a lot of built-In functions and some of them are listed below:

```
x = 4
y = 4.5
print(abs(x))
print(sqrt(x))
print(ceiling(y))
print(floor(y))
print(round(y))
```



```
> x = 4
> y = 4.5
> print(abs(x))
[1] 4
> print(sqrt(x))
[1] 2
> print(ceiling(y))
[1] 5
> print(floor(y))
[1] 4
> print(round(y))
[1] 4
>
```

# R Functions

**User Defined Functions**

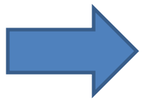
- ✓ In our program, R allows us to define our own function. To meet the user's needs, the user defines a user-define function. Once these functions are created, we can use them as if they were built-in.
- ✓ User defined functions have two parts :
  - ✓ Function body
  - ✓ Function Calling
- ✓ Both of the parts, user has to declare and use. The basic syntax for a function is:

```
func_name <- function(arg_1, arg_2, ...) {
   Function body
}
```

# R Functions

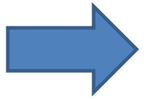
## User Defined Functions Example

```
# Creating a function without an argument.
new.function <- function() {
  for(i in 1:5) {
    print(i^2)
  }
}

new.function()
```



```
> new.function <- function() {
+   for(i in 1:5) {
+     print(i^2)
+   }
+ }
>
> new.function()
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
>
```

```
# Creating a function to print squares of numbers in sequence.
new.function <- function(a) {
  for(i in 1:a) {
    b <- i^2
    print(b)
  }
}
# Calling the function new.function supplying 10 as an argument.
new.function(10)
```



```
> # Creating a function to print squares of numbers in sequence.
> new.function <- function(a) {
+   for(i in 1:a) {
+     b <- i^2
+     print(b)
+   }
+ }
> # Calling the function new.function supplying 10 as an argument.
> new.function(10)
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
>
```

theknowledgeacademy

# Module 2

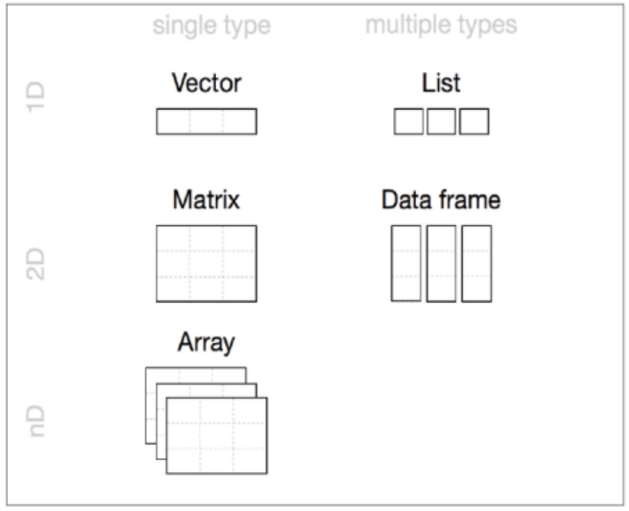

## Description

| | |
|---|---|
| 1 | What are Data Structures? |
| 2 | Vectors |
| 3 | Lists |
| 4 | Matrix |
| 5 | Arrays |
| 6 | DataFrames |

# Data Structures in R

**Introduction**

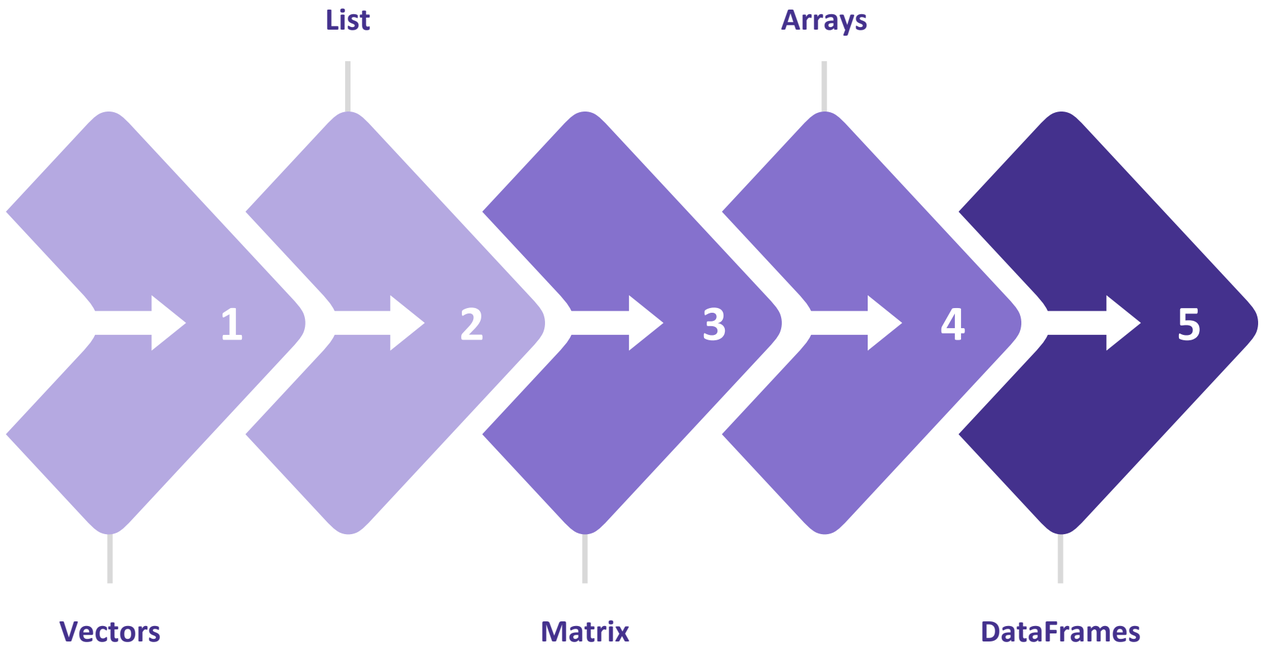
- ✓ A data structure is a method of organising data in a computer that allows it to be used effectively. The goal is to reduce the spatial and time complexities of various tasks. In R programming, data structures are tools for storing multiple values.
- ✓ R's base data structures are frequently organised by their dimensionality (1D, 2D, or nD) and whether they are homogeneous (all elements must be of the same type) or heterogeneous (all elements must be of different types) (the elements are often of various types). This gives rise to the 5 most commonly used data types in R:



# Data Structures in R

**Types of Data Structures in R**

- ✓ R consist of 5 main Data Structure, those are:

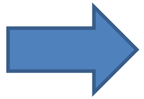


# Data Structures in R

**Vectors in R**

- ✓ A vector is a fixed-length ordered collection of basic data types. The only requirement here is that all elements of a vector be of the same data type, i.e. homogeneous data structures.
- ✓ Vectors are data structures with a single dimension.
- ✓ The syntax to declare a vector is:

```
# R program to illustrate Vector

# Vectors(ordered collection of same data type)
X = c(1, 3, 5, 7, 8)

# Printing those elements in console
print(X)
```



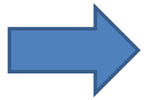
```
> # R program to illustrate Vector
>
> # Vectors(ordered collection of same data type)
> X = c(1, 3, 5, 7, 8)
>
> # Printing those elements in console
> print(X)
[1] 1 3 5 7 8
>
```

# Data Structures in R

**Lists in R**

- ✓ A list is a type of generic object that contains an ordered collection of objects. Lists are diverse data structures. These are one-dimensional data structures as well.
- ✓ A list can be a vector list, a matrix list, a character list, a function list, and so on.
- ✓ The syntax to write a list is:

```
# R program to illustrate a List

# The first attributes is a numeric vector
# containing the employee IDs which is
# created using the 'c' command here
empId = c(1, 2, 3, 4)

# The second attribute is the employee name
# which is created using this line of code here
# which is the character vector
empName = c("Debi", "Sandeep", "Subham", "Shiba")

# The third attribute is the number of employees
# which is a single numeric variable.
numberOfEmp = 4

# We can combine all these three different
# data types into a list
# containing the details of employees
# which can be done using a list command
empList = list(empId, empName, numberOfEmp)

print(empList)
```



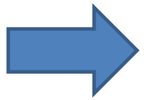
```
> # which is the character vector
> empName = c("Debi", "Sandeep", "Subham", "Shiba")
>
> # The third attribute is the number of employees
> # which is a single numeric variable.
> numberOfEmp = 4
>
> # We can combine all these three different
> # data types into a list
> # containing the details of employees
> # which can be done using a list command
> empList = list(empId, empName, numberOfEmp)
>
> print(empList)
[[1]]
[1] 1 2 3 4

[[2]]
[1] "Debi"    "Sandeep" "Subham"  "Shiba"

[[3]]
[1] 4
```

# Data Structures in R

**Matrix in R**

- ✓ A matrix is a rectangular array of numbers organised in rows and columns. Matrices are two-dimensional data structures that are homogeneous in nature.
- ✓ In R, you must use the matrix function to create a matrix. The vector's elements are passed as arguments to this matrix().
- ✓ You must specify how many rows and columns you want in your matrix, and keep in mind that matrices are always ordered in column-wise order by default.
- ✓ The syntax to write a Matrix is:

```
# R program to illustrate a matrix

A = matrix(
  # Taking sequence of elements
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),

  # No of rows and columns
  nrow = 3, ncol = 3,

  # By default matrices are
  # in column-wise order
  # So this parameter decides
  # how to arrange the matrix
  byrow = TRUE
)

print(A)
```



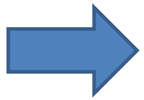
```
> # R program to illustrate a matrix
>
> A = matrix(
+   # Taking sequence of elements
+   c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+
+   # No of rows and columns
+   nrow = 3, ncol = 3,
+
+   # By default matrices are
+   # in column-wise order
+   # So this parameter decides
+   # how to arrange the matrix
+   byrow = TRUE
+ )
>
> print(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
>
```

# Data Structures in R

**Arrays in R**

- ✓ Arrays are R data objects that store information in more than two dimensions. Arrays are data structures with n dimensions. For example, if we create an array with dimensions (2, 3, 3), it will generate three rectangular matrices, each with two rows and three columns. They are data structures that are all the same.
- ✓ In R, you must use the array function to create an array (). The arguments to this array() are the set of elements in vectors, and you must pass a vector containing the array's dimensions.
- ✓ The syntax to declare Array in R is:

```
# R program to illustrate an array

A = array(
  # Taking sequence of elements
  c(1, 2, 3, 4, 5, 6, 7, 8),

  # Creating two rectangular matrices
  # each with two rows and two columns
  dim = c(2, 2, 2)
)

print(A)
```



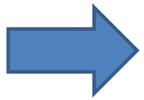
```
> print(A)
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

>
```

# Data Structures in R

**DataFrames in R**

- ✓ DataFrames are R's generic data objects that are used to store tabular data. They are heterogeneous two-dimensional data structures. These are lists of vectors with the same length.
- ✓ The following constraints are imposed on data frames:
  - ✓ A DataFrames must have column names, and each row must have a distinct name.
  - ✓ Each column in DataFrames must contain the same number of items.
  - ✓ A single column's items must all be of the same data type.
  - ✓ Different data types may exist in different columns.
  - ✓ The data.frame() function in R is used to create a data frame.

```r
# R program to illustrate dataframe

# A vector which is a character vector
Name = c("Amiya", "Raj", "Asish")

# A vector which is a character vector
Language = c("R", "Python", "Java")

# A vector which is a numeric vector
Age = c(22, 25, 45)

# To create dataframe use data.frame command
# and then pass each of the vectors
# we have created as arguments
# to the function data.frame()
df = data.frame(Name, Language, Age)

print(df)
```



```
> print(df)
   Name Language Age
1 Amiya        R  22
2   Raj   Python  25
3 Asish     Java  45
>
```

theknowledgeacademy

# Module 3

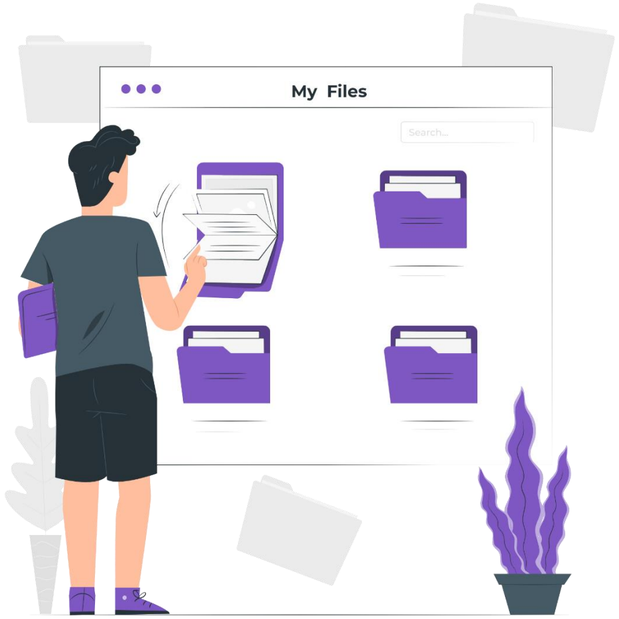

## Description

| | |
|---|---|
| 1 | Types of Files in R |
| 2 | Working with CSV Files |
| 3 | Working with Excel Files |
| 4 | Working with JSON Files |
| 5 | Working with XML Files |

# Working with Data in R

**Introduction**

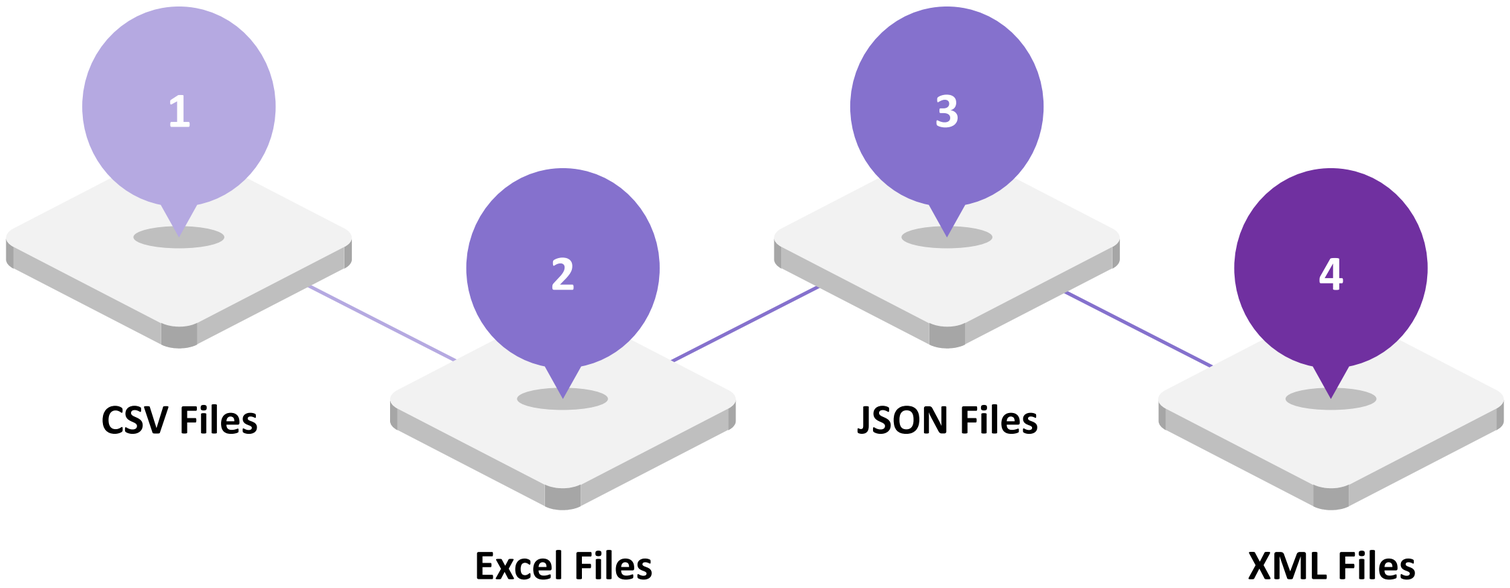
- ✓ We can read data from files outside the R environment and write data into files that will be stored and accessed by the operating system in R.
- ✓ In R, we can read and write to a variety of file formats, including csv, excel, and json.
- ✓ The getwd() function can be used to determine which directory the R workspace is pointing to. You can also use the setwd() function to create a new working directory.



# Working with Data in R

**Types of Files in R**

- ✓ In R, we can read and write to a variety of file formats. Some of the most used files formats are:



# Working with Data in R

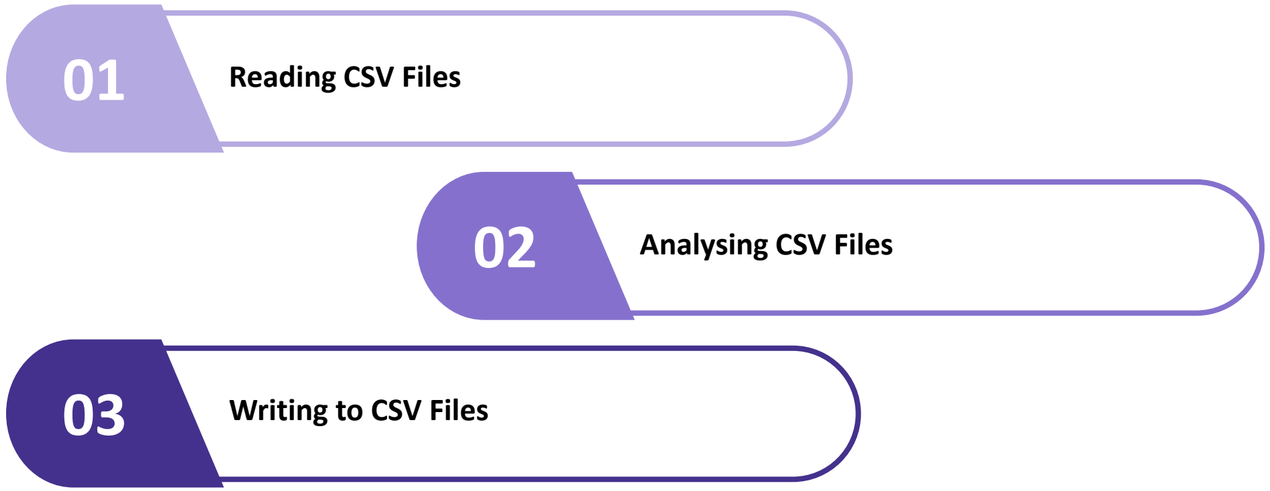
**Working with CSV Files**

- ✓ A CSV file is a plain text file that contains a list of data separated by commas. These files are frequently used for data exchange between applications. CSV files, for example, are commonly supported by databases and contact managers.
- ✓ These files are also known as character-separated values or comma-delimited files.
- ✓ There are numerous R packages available for accessing data from CSV files.
- ✓ R supports reading CSV file data from files stored outside of the R environment also.

# Working with Data in R

**Operation with CSV Files**

- ✓ R supports multiple operations for CSV files. Some of them are:



# Working with Data in R

**Reading CSV Files**

- ✓ R has a large number of functions. The read.csv() function in R allows us to read a CSV file from our current working directory. This function accepts a file name as an input and returns all of the records contained within it.
- ✓ Example :

```
#Program to demonstrate reading a csv file
x = "C:\\Users\\jai.prakash\\Anaconda3\\Lib\\site-packages\\statsmodels\\datasets\\ccard\\ccard.csv"
data = read.csv(x)
head(data)
```

```
> head(data)
  AVGEXP AGE INCOME INCOMESQ OWNRENT
1 124.98  38   4.52  20.4304       1
2   9.85  33   2.42   5.8564       0
3  15.00  34   4.50  20.2500       1
4 137.87  31   2.54   6.4516       0
5 546.50  32   9.79  95.8441       1
6  92.00  23   2.50   6.2500       0
>
```

# Working with Data in R

**Analysing CSV Files**

- ✓ Analysing a CSV file consists of numerous operations as per requirements. Some of the operations include file inspection such as checking the type of each and every column and row. Some operations can be related to data manipulations such as finding minimum or maximum values of a column.
- ✓ Example :

```
#Veiwing data
View(data)
```

| | AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|---|
| 1 | 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 2 | 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 3 | 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 4 | 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 5 | 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 6 | 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 7 | 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 8 | 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 9 | 777.82 | 37 | 3.80 | 14.4400 | 1 |

# Working with Data in R

**Analysing CSV Files**

✓ Example :

```
#Max value of a col
max(data$INCOME)
```

```
> max(data$INCOME)
[1] 10
>
```

```
#Min value of a col
min(data$AGE)
```

```
> min(data$AGE)
[1] 20
>
```

```
#Getting custom information
View(subset(data,data$OWNRENT == 1))
```

| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 256.66 | 31 | 3.95 | 15.6025 | 1 |
| 78.87 | 29 | 2.45 | 6.0025 | 1 |
| 42.62 | 35 | 1.91 | 3.6481 | 1 |
| 335.43 | 41 | 3.20 | 10.2400 | 1 |
| 248.72 | 40 | 4.00 | 16.0000 | 1 |
| 548.03 | 40 | 10.00 | 100.0000 | 1 |
| 43.34 | 35 | 2.35 | 5.5225 | 1 |

# Working with Data in R

**Writing to CSV Files**

- ✓ Writing into CSV files consist of operations like fetching data from custom command and inserting the data into a new CSV file.
- ✓ Example :

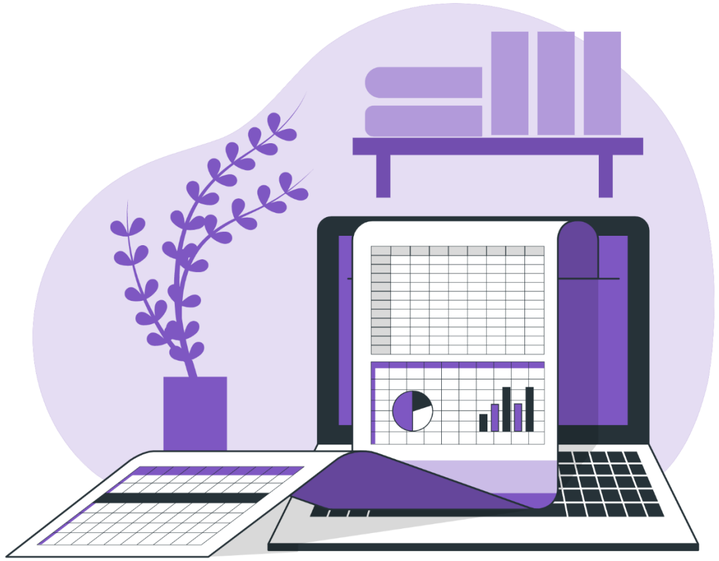
```
#Getting custom information
details = subset(data,data$OWNRENT == 1)


# Writing filtered data into a new file.
write.csv(details,"output.csv")
new_details<- read.csv("output.csv")
print(new_details)

#Veiwing New File
view(new_details)
```

| X | AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|---|
| 1 | 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 3 | 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 5 | 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 8 | 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 9 | 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 11 | 256.66 | 31 | 3.95 | 15.6025 | 1 |
| 12 | 78.87 | 29 | 2.45 | 6.0025 | 1 |
| 13 | 42.62 | 35 | 1.91 | 3.6481 | 1 |
| 14 | 335.43 | 41 | 3.20 | 10.2400 | 1 |
| 15 | 248.72 | 40 | 4.00 | 16.0000 | 1 |
| 16 | 548.03 | 40 | 10.00 | 100.0000 | 1 |
| 17 | 43.34 | 35 | 2.35 | 5.5225 | 1 |

# Working with Data in R

**Working with Excel Files**

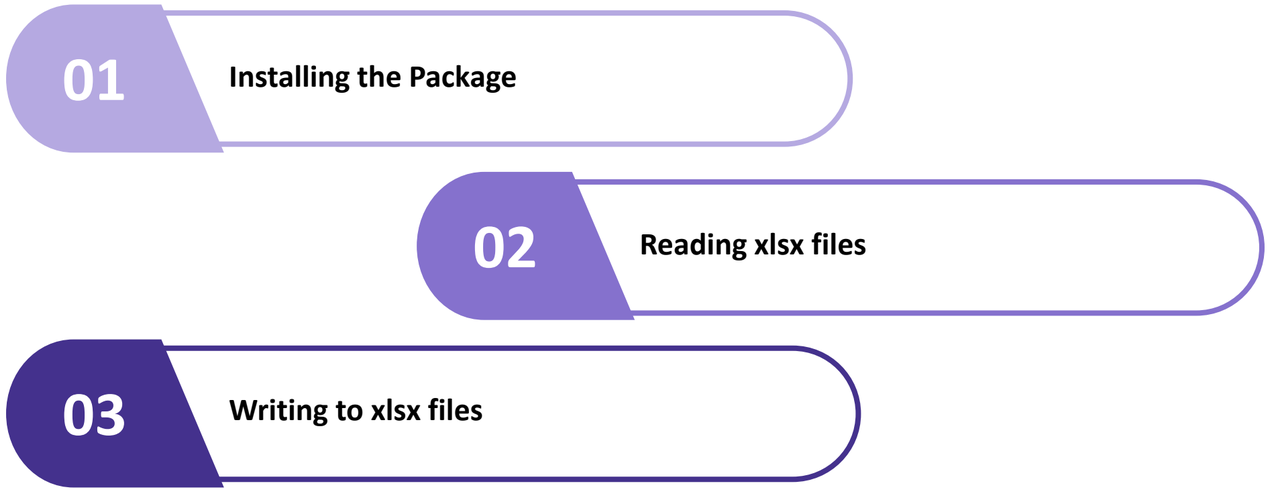
- ✓ The “xlsx” file extension is a spreadsheet file format developed by Microsoft for use with Microsoft Excel.
- ✓ Microsoft Excel is a popular spreadsheet program that accepts data in.xls or.xlsx format. R provides some excel-specific packages that allow us to read data directly from these files.
- ✓ So in order to work on Excel files, we first have to install the necessary package.



# Working with Data in R

**Operation with XLSX Files**

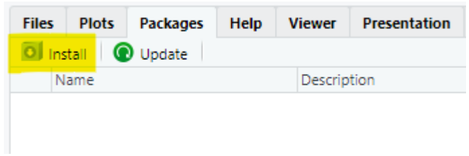
- ✓ R supports multiple operations for xlsx files. Some of them are:

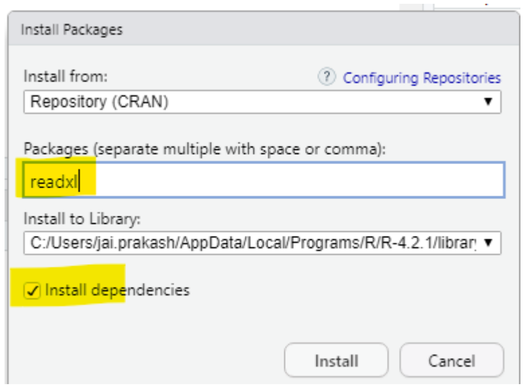


# Working with Data in R

**Installing the Package**

- ✓ In order to work on xlsx files, we first have to install the “readxl” package from the install packages dashboard.
- ✓ The procedure is as follows:




```
package 'readxl' successfully unpacked and MD5 sums checked
```

# Working with Data in R

**Reading xlsx files**

- ✓ In order to read an xlsx file, we first have to use the function **read_excel()** from the library readxl.
- ✓ The procedure is as follows:

```
#Loading library
library(readxl)

#Reading an xlsx file
ccxl <- read_excel("~/Jp work/ccxl.xlsx")
View(ccxl)
```

| customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup |
|---|---|---|---|---|---|---|---|---|---|---|
| 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | Yes |
| 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | No |
| 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | Yes |
| 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | No |
| 9237-HQITU | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | No |
| 9305-CDSKC | Female | 0 | No | No | 8 | Yes | Yes | Fiber optic | No | No |
| 1452-KIOVK | Male | 0 | No | Yes | 22 | Yes | Yes | Fiber optic | No | Yes |
| 6713-OKOMC | Female | 0 | No | No | 10 | No | No phone service | DSL | Yes | No |
| 7892-POOKP | Female | 0 | Yes | No | 28 | Yes | Yes | Fiber optic | No | No |
| 6388-TABGU | Male | 0 | No | Yes | 62 | Yes | No | DSL | Yes | Yes |
| 9763-GRSKD | Male | 0 | Yes | Yes | 13 | Yes | No | DSL | Yes | No |
| 7469-LKBCI | Male | 0 | No | No | 16 | Yes | No | No | No internet service | No internet service |
| 8091-TTVAX | Male | 0 | Yes | No | 58 | Yes | Yes | Fiber optic | No | No |
| 0280-XJGEX | Male | 0 | No | No | 49 | Yes | Yes | Fiber optic | No | Yes |

# Working with Data in R

**Writing to xlsx files**

- ✓ In order to write into an xlsx file, we first have to use the function **write_xlsx()** from the library **writexl**.
- ✓ The procedure is as follows:

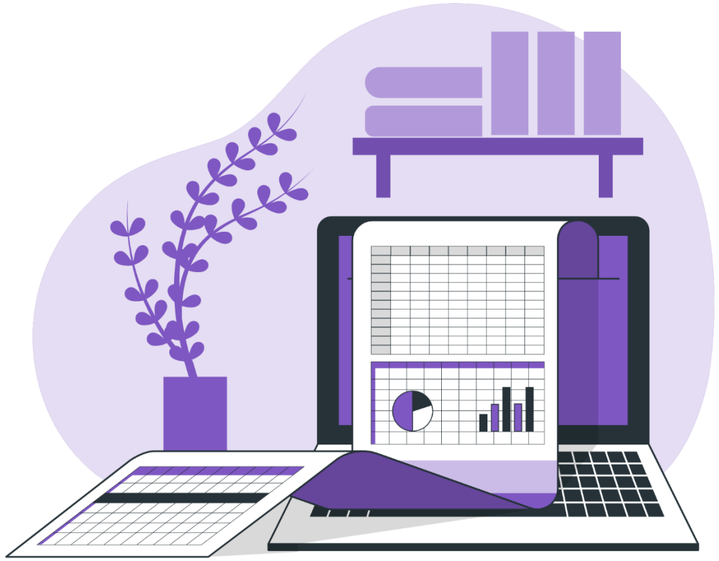
```
#Loading library
library(writexl)

#Writing data into an xlsx file
dd = data.frame(data)
write_xlsx(dd,'newxl.xlsx')

#reading data from new file
newdata = read_excel('newxl.xlsx')
View(newdata)
```

| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 52.58 | 28 | 3.20 | 10.2400 | 0 |
| 256.66 | 31 | 3.95 | 15.6025 | 1 |
| 78.87 | 29 | 2.45 | 6.0025 | 1 |

# Working with Data in R

**Working with JSON Files**

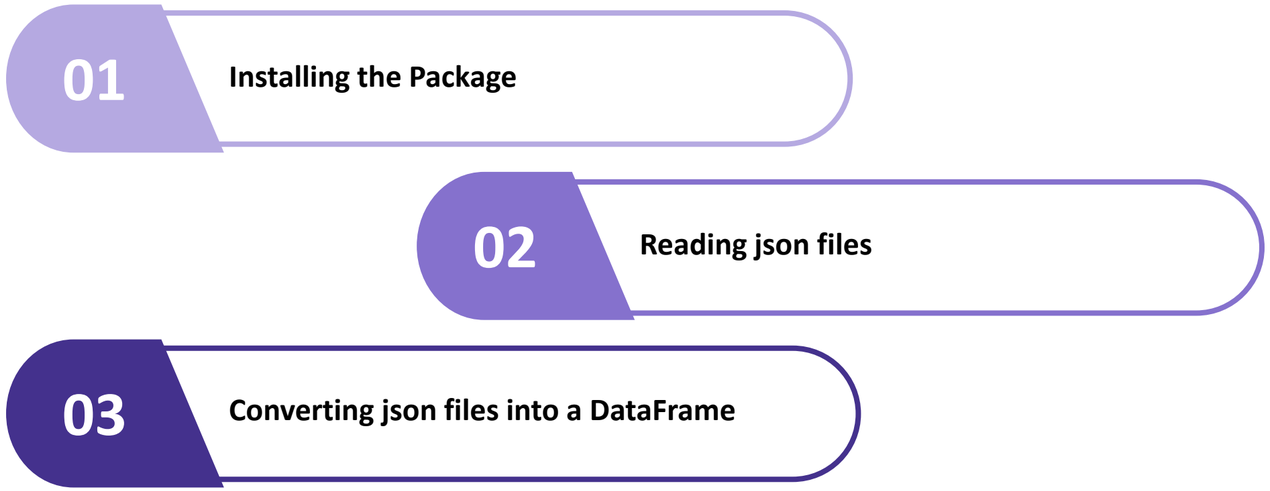
- ✓ JSON is an abbreviation for JavaScript Object Notation. The data is stored in the JSON file as text in a human-readable format. JSON files, like other files, can be read and written to.
- ✓ R provides a package called **rjson** for this purpose, which we must install using the Install packages prompt.



# Working with Data in R

**Operation with JSON Files**

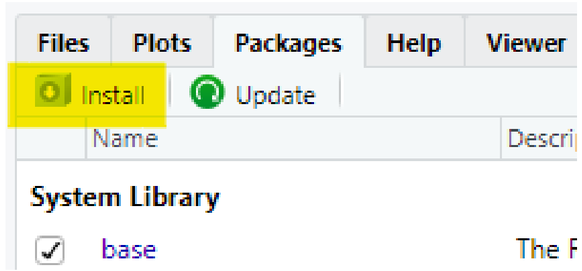
- ✓ R supports multiple operations for JSON files. Some of them are:

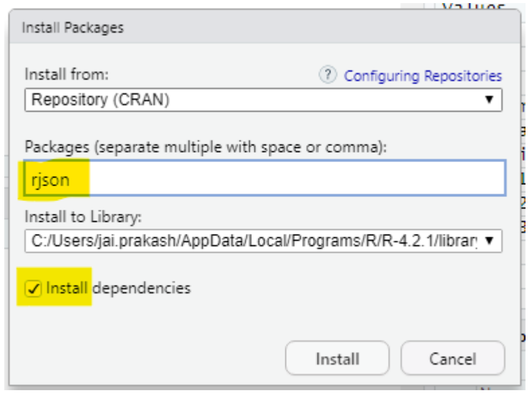


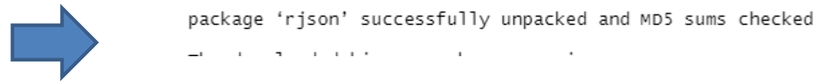
# Working with Data in R

**Installing the Package**

- ✓ In order to install rjson package, we have to use the install package option.
- ✓ The procedure is as follows:



# Working with Data in R

**Reading a JSON File**

- ✓ In order to read a json file, we have to use the function **fromJSON()** from the library **rjson.**
- ✓ The procedure is as follows:

```
#Loading Library
library(rjson)

#reading the file
result <- fromJSON(file = "info.json")

# Printing the result.
print(result)
```

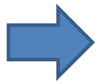
```
> print(result)
$id
[1] "1" "2" "3" "4" "5" "6" "7" "8"

$name
[1] "Shubham"  "Nishka"   "Gunjan"   "Sumit"    "Arpita"   "Vaishali" "Anisha"   "Ginni"

$salary
[1] "623" "552" "669" "825" "762" "882" "783" "964"

$start_date
[1] "1/1/2012"   "9/15/2013"  "11/23/2013" "5/11/2014"  "3/27/2015"  "5/21/2013"  "7/30/2013"  "6/17/2014"

$dept
[1] "IT"         "Operations" "Finance"    "HR"         "Finance"    "IT"         "Operations" "Finance"
```

# Working with Data in R

**Converting a JSON File into a DataFrame**

- ✓ In order to read a json file, we have to use the function **as.data.frame()**
- ✓ The procedure is as follows:

```
# Converting the JSON record to a data frame.
data_frame <- as.data.frame(result)

#Printing JSON data frame
print(data_frame)
```



```
> print(data_frame)
  id     name salary start_date       dept
1  1  Shubham    623   1/1/2012         IT
2  2   Nishka    552  9/15/2013 Operations
3  3   Gunjan    669 11/23/2013    Finance
4  4    Sumit    825  5/11/2014         HR
5  5   Arpita    762  3/27/2015    Finance
6  6 Vaishali    882  5/21/2013         IT
7  7   Anisha    783  7/30/2013 Operations
8  8    Ginni    964  6/17/2014    Finance
```

theknowledgeacademy

# Module 4


## Description

| | |
|---|---|
| 1 | What is Data Manipulation? |
| 2 | Installation of Dplyr Package |
| 3 | Data Manipulation Operations in R |

# Data Manipulation in R

**What is Data Manipulation?**

- ✓ Data manipulation is the process of modifying data to make it easier to read and more organised. We manipulate data in order to analyse and visualise it.
- ✓ It is also used in conjunction with the term 'data exploration,' which refers to the process of organising data using available sets of variables.
- ✓ Machine data collection can involve a lot of errors and inaccuracies in reading at times. Data manipulation is also used to remove inaccuracies and improve data accuracy and precision.

# Data Manipulation in R

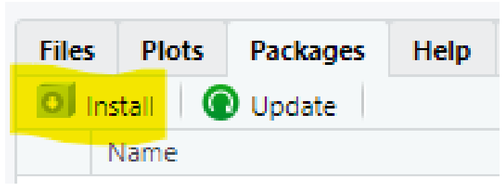
**Data Manipulation in R**

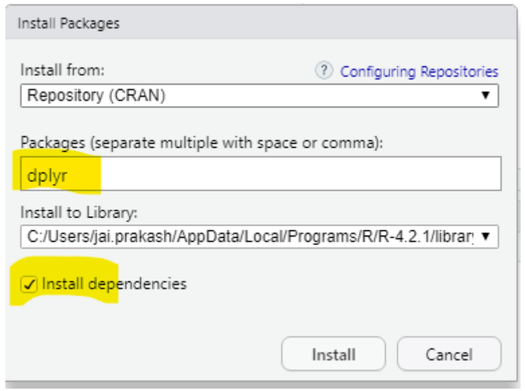
- ✓ We can represent data in the form of data analytics using data structures. Data manipulation in R is possible for further analysis and visualisation.
- ✓ In order to work on the concepts of Data Manipulation in R, we have to use the library "**Dplyr**".

# Data Manipulation in R

**Installation of dplyr Package**

- ✓ In order to install the package, we have the options to either user the command prompt or use the install package option.
- ✓ So, we'll be using the Install Package option for **dplyr** Package.



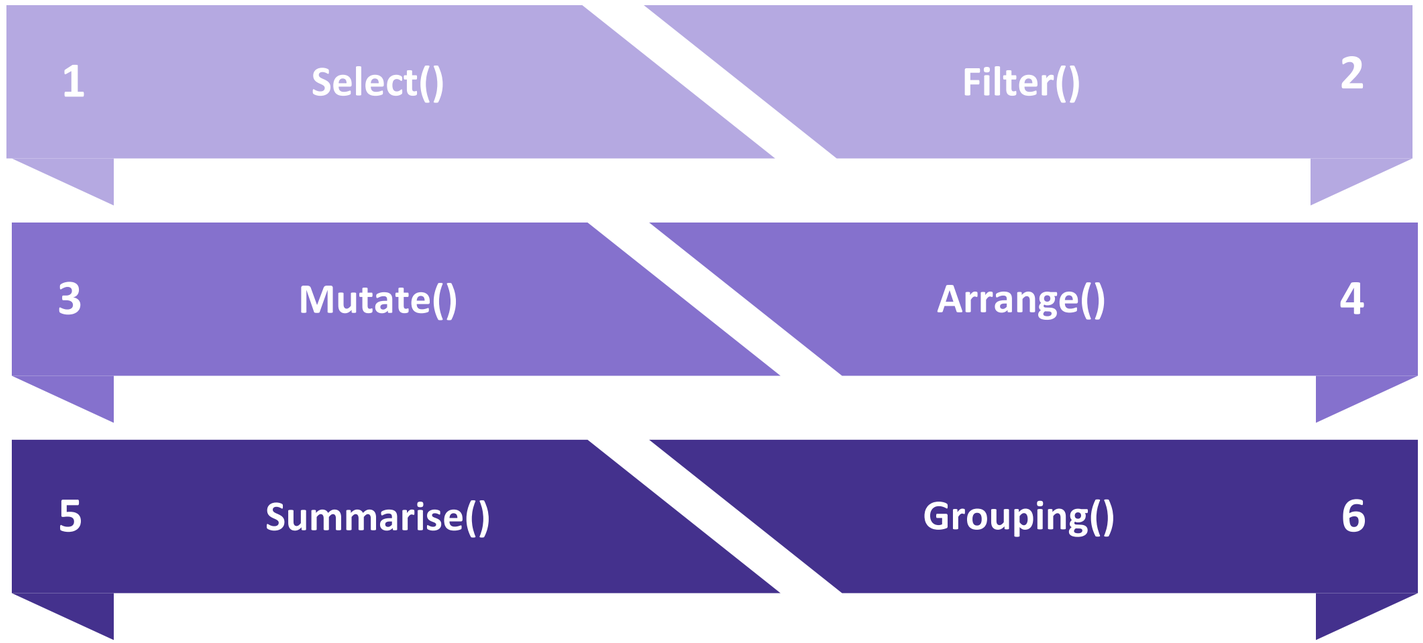




```
package 'generics' successfully unpacked and MD5 sums checked
package 'dplyr' successfully unpacked and MD5 sums checked
```

# Data Manipulation in R

**Data Manipulation Operations in R**

- ✓ R language support numerous operations of Data Manipulation with the help of dplyr package.
- ✓ We'll be using the most important functions of dplyr package. Those are as follows:

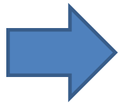


# Data Manipulation in R

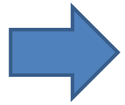
**Select()**

- ✓ The select() method is one of R's fundamental data manipulation functions.
- ✓ In R, this method is used to select columns. You can use this to select data by its column name.
- ✓ Syntax :

```
#Loading Library
library(dplyr)

#Veiwing Data
View(data)

#Selecting custom columns from data
View(select(data,AGE,INCOME))
```



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 52.58 | 28 | 3.20 | 10.2400 | 0 |
| 256.66 | 31 | 3.95 | 15.6025 | 1 |
| 78.87 | 29 | 2.45 | 6.0025 | 1 |
| 42.62 | 35 | 1.91 | 3.6481 | 1 |

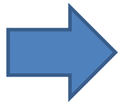


| AGE | INCOME |
|---|---|
| 38 | 4.52 |
| 33 | 2.42 |
| 34 | 4.50 |
| 31 | 2.54 |
| 32 | 9.79 |
| 23 | 2.50 |
| 28 | 3.96 |
| 29 | 2.37 |
| 37 | 3.80 |
| 28 | 3.20 |
| 31 | 3.95 |

# Data Manipulation in R

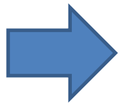
**Select() command Examples**

```
#Select columns with column Names
View(select(data,OWNRENT,INCOME))
```



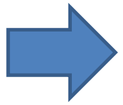
| OWNRENT | INCOME |
|---|---|
| 1 | 4.52 |
| 0 | 2.42 |
| 1 | 4.50 |
| 0 | 2.54 |
| 1 | 9.79 |
| 0 | 2.50 |

```
#Selecting multiple columns
View(select(data,AVGEXP:INCOME))
```



| AVGEXP | AGE | INCOME |
|---|---|---|
| 124.98 | 38 | 4.52 |
| 9.85 | 33 | 2.42 |
| 15.00 | 34 | 4.50 |
| 137.87 | 31 | 2.54 |
| 546.50 | 32 | 9.79 |
| 92.00 | 23 | 2.50 |
| 40.83 | 28 | 3.96 |

```
#Selecting columns with numeric index
View(select(data,c(1:4)))
```



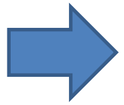
| AVGEXP | AGE | INCOME | INCOMESQ |
|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 |
| 9.85 | 33 | 2.42 | 5.8564 |
| 15.00 | 34 | 4.50 | 20.2500 |
| 137.87 | 31 | 2.54 | 6.4516 |
| 546.50 | 32 | 9.79 | 95.8441 |
| 92.00 | 23 | 2.50 | 6.2500 |
| 40.83 | 28 | 3.96 | 15.6816 |
| 150.79 | 29 | 2.37 | 5.6169 |

# Data Manipulation in R

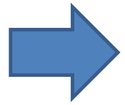
**Filter()**

- ✓ This method is used to find rows in a dataset that meet certain criteria.
- ✓ It works similarly to select() in that you pass the data frame first, followed by a condition separated by a comma.
- ✓ Syntax :

```
#Filter Command
view(filter(data,INCOME > 5.0))
```



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 52.58 | 28 | 3.20 | 10.2400 | 0 |

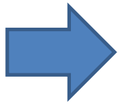


| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 548.03 | 40 | 10.00 | 100.0000 | 1 |
| 37.58 | 43 | 5.14 | 26.4196 | 1 |
| 1532.77 | 40 | 5.50 | 30.2500 | 1 |
| 568.77 | 37 | 5.70 | 32.4900 | 1 |
| 644.83 | 28 | 7.00 | 49.0000 | 1 |
| 306.03 | 41 | 6.00 | 36.0000 | 0 |
| 235.57 | 41 | 7.24 | 52.4176 | 1 |
| 474.15 | 33 | 6.00 | 36.0000 | 1 |
| 251.52 | 46 | 5.50 | 30.2500 | 1 |

# Data Manipulation in R

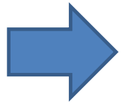
**Filter() command Examples**

```
#multiple filters in a single command
View(filter(data,INCOME > 5.0,AGE > 30))
```



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 548.03 | 40 | 10.00 | 100.0000 | 1 |
| 37.58 | 43 | 5.14 | 26.4196 | 1 |
| 1532.77 | 40 | 5.50 | 30.2500 | 1 |
| 568.77 | 37 | 5.70 | 32.4900 | 1 |
| 306.03 | 41 | 6.00 | 36.0000 | 0 |
| 235.57 | 41 | 7.24 | 52.4176 | 1 |
| 474.15 | 33 | 6.00 | 36.0000 | 1 |
| 251.52 | 46 | 5.50 | 30.2500 | 1 |

```
#different filters in single command
View(filter(data,AGE >40,INCOME > 5.0))
```



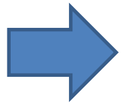
| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 37.58 | 43 | 5.14 | 26.4196 | 1 |
| 306.03 | 41 | 6.00 | 36.0000 | 0 |
| 235.57 | 41 | 7.24 | 52.4176 | 1 |
| 251.52 | 46 | 5.50 | 30.2500 | 1 |

# Data Manipulation in R

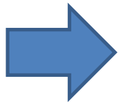
**Mutate()**

- ✓ You can use the mutate() method to add new columns to a dataset while keeping the existing ones. A condition can be used to create these columns..
- ✓ Syntax :

```
#Mutate command
col = mutate(data,newcol = INCOME > 6)

#veiwing result
view(filter(col,newcol == TRUE))
```



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 52.58 | 28 | 3.20 | 10.2400 | 0 |

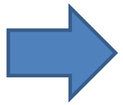


| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT | newcol |
|---|---|---|---|---|---|
| 546.50 | 32 | 9.79 | 95.8441 | 1 | TRUE |
| 548.03 | 40 | 10.00 | 100.0000 | 1 | TRUE |
| 644.83 | 28 | 7.00 | 49.0000 | 1 | TRUE |
| 235.57 | 41 | 7.24 | 52.4176 | 1 | TRUE |

# Data Manipulation in R

**Mutate() command Examples**

```
#Creating new col for Aged People
aged = mutate(data,oldage = AGE >= 40)

#veiwing result
View(filter(aged,oldage == TRUE))
```



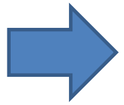
| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT | oldage |
|---|---|---|---|---|---|
| 335.43 | 41 | 3.20 | 10.2400 | 1 | TRUE |
| 248.72 | 40 | 4.00 | 16.0000 | 1 | TRUE |
| 548.03 | 40 | 10.00 | 100.0000 | 1 | TRUE |
| 37.58 | 43 | 5.14 | 26.4196 | 1 | TRUE |
| 1532.77 | 40 | 5.50 | 30.2500 | 1 | TRUE |
| 541.30 | 43 | 3.54 | 12.5316 | 1 | TRUE |
| 306.03 | 41 | 6.00 | 36.0000 | 0 | TRUE |
| 104.54 | 42 | 3.90 | 15.2100 | 0 | TRUE |
| 65.25 | 55 | 2.64 | 6.9696 | 1 | TRUE |
| 235.57 | 41 | 7.24 | 52.4176 | 1 | TRUE |
| 68.38 | 43 | 2.40 | 5.7600 | 0 | TRUE |
| 251.52 | 46 | 5.50 | 30.2500 | 1 | TRUE |

# Data Manipulation in R

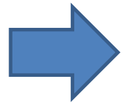
**Arrange()**

- ✓ Using one or more variables, this is used to sort rows in ascending or descending order. You can use a minus (-) symbol before the sorting variable instead of the desc() method.
- ✓ This will indicate the sorting order in descending order.
- ✓ Syntax:

```
#Arrange() command
View(arrange(data,INCOME))
```



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 52.58 | 28 | 3.20 | 10.2400 | 0 |



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 73.18 | 22 | 1.50 | 2.2500 | 0 |
| 32.78 | 21 | 1.50 | 2.2500 | 0 |
| 63.92 | 30 | 1.51 | 2.2801 | 0 |
| 108.61 | 20 | 1.65 | 2.7225 | 0 |
| 105.04 | 21 | 1.70 | 2.8900 | 0 |
| 165.65 | 30 | 1.85 | 3.4225 | 0 |
| 42.62 | 35 | 1.91 | 3.6481 | 1 |
| 218.52 | 34 | 2.00 | 4.0000 | 1 |
| 319.49 | 36 | 2.00 | 4.0000 | 0 |
| 93.20 | 24 | 2.00 | 4.0000 | 0 |

# Data Manipulation in R

**Arrange() command Examples**

```
#Ascending order
View(arrange(data,INCOME))

#Descending order
View(arrange(data,-(INCOME)))
```

| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 73.18 | 22 | 1.50 | 2.2500 | 0 |
| 32.78 | 21 | 1.50 | 2.2500 | 0 |
| 63.92 | 30 | 1.51 | 2.2801 | 0 |
| 108.61 | 20 | 1.65 | 2.7225 | 0 |
| 105.04 | 21 | 1.70 | 2.8900 | 0 |
| 165.85 | 30 | 1.85 | 3.4225 | 0 |
| 42.62 | 35 | 1.91 | 3.6481 | 1 |
| 218.52 | 34 | 2.00 | 4.0000 | 1 |

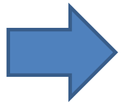
| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 548.03 | 40 | 10.00 | 100.0000 | 1 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 235.57 | 41 | 7.24 | 52.4176 | 1 |
| 644.83 | 28 | 7.00 | 49.0000 | 1 |
| 306.03 | 41 | 6.00 | 36.0000 | 0 |
| 474.15 | 33 | 6.00 | 36.0000 | 1 |
| 568.77 | 37 | 5.70 | 32.4900 | 1 |
| 1532.77 | 40 | 5.50 | 30.2500 | 1 |

# Data Manipulation in R

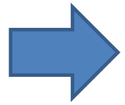
**Summarise()**

- ✓ The summarise() function is useful for calculating data insights like mean, median, and mode. It is used in conjunction with grouped data generated by another method, group by.
- ✓ summarise() is useful for condensing multiple values into a single one.
- ✓ Syntax:

```
#Summarise command
View(summarise(data,mean(INCOME)))
```



| AVGEXP | AGE | INCOME | INCOMESQ | OWNRENT |
|---|---|---|---|---|
| 124.98 | 38 | 4.52 | 20.4304 | 1 |
| 9.85 | 33 | 2.42 | 5.8564 | 0 |
| 15.00 | 34 | 4.50 | 20.2500 | 1 |
| 137.87 | 31 | 2.54 | 6.4516 | 0 |
| 546.50 | 32 | 9.79 | 95.8441 | 1 |
| 92.00 | 23 | 2.50 | 6.2500 | 0 |
| 40.83 | 28 | 3.96 | 15.6816 | 0 |
| 150.79 | 29 | 2.37 | 5.6169 | 1 |
| 777.82 | 37 | 3.80 | 14.4400 | 1 |
| 52.58 | 28 | 3.20 | 10.2400 | 0 |

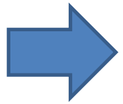


| mean(INCOME) |
|---|
| 3.437083 |

# Data Manipulation in R

**Summarise() command Examples**

```
#finding summary of columns
View(summarise(data,mean(INCOME),mean(AGE)))
```



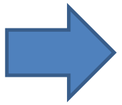
| mean(INCOME) | mean(AGE) |
|---|---|
| 3.437083 | 31.27778 |

# Data Manipulation in R

**group_by()**

- ✓ The group_by() method is used to group observations in a dataset according to one or more variables.
- ✓ Syntax:

```
#group_by command
grp = group_by(ccxl,InternetService)
View(summarise(grp,mean(MonthlyCharges)))
```



| InternetService | mean(MonthlyCharges) |
|---|---|
| DSL | 58.10217 |
| Fiber optic | 91.50013 |
| No | 21.07919 |

theknowledgeacademy

# Module 5


## Description

| | |
|---|---|
| 1 | What is Data visualisation? |
| 2 | Working with Graphs and Plots in R |

# Data visualisation in R

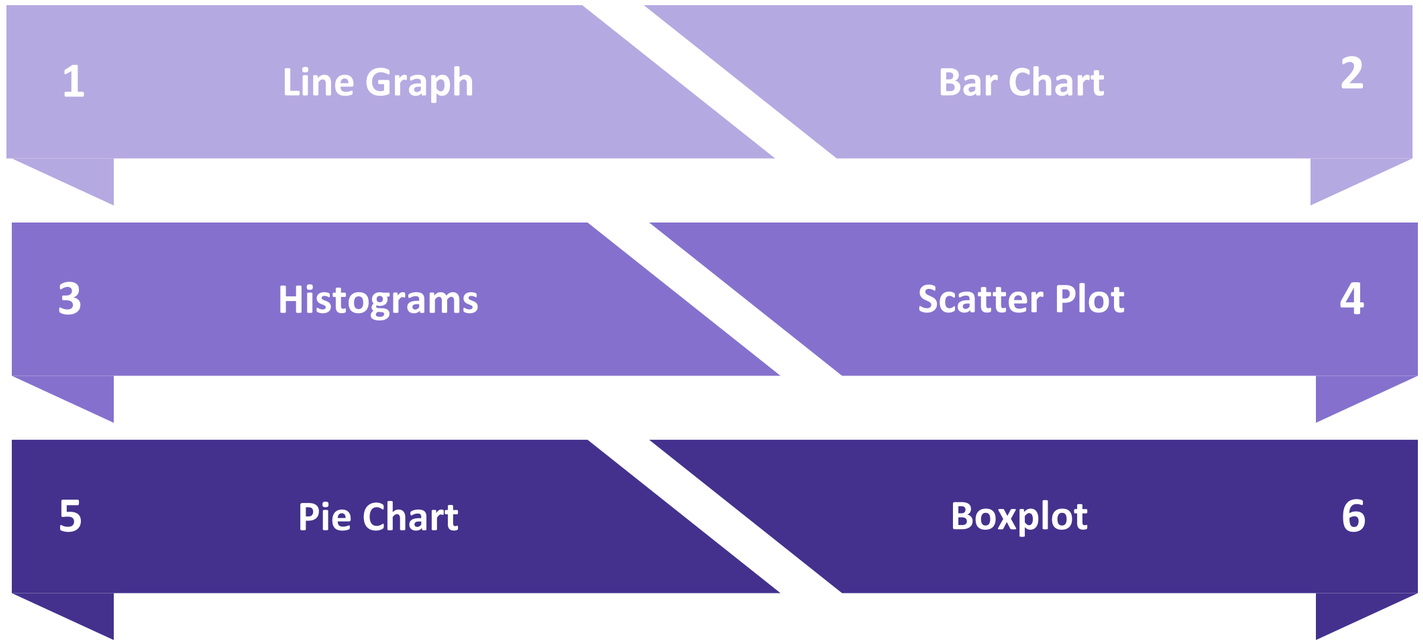
**What is Data visualisation?**

- ✓ Data visualisation is a technique for delivering data insights through visual cues such as graphs, charts, maps, and many others.
- ✓ This is beneficial because it allows for the intuitive and simple understanding of large amounts of data, allowing for better decision-making.
- ✓ R is a programming language created for statistical computing, graphical data analysis, and scientific research. It is commonly used for data visualisation because its packages provide flexibility and require little coding.

# Data visualisation in R

**Graphs and Plots in R?**

- ✓ R as a Data Favoured language provides a lot of graphs and plots for Data visualisation. Some of the most used graphs are listed below:

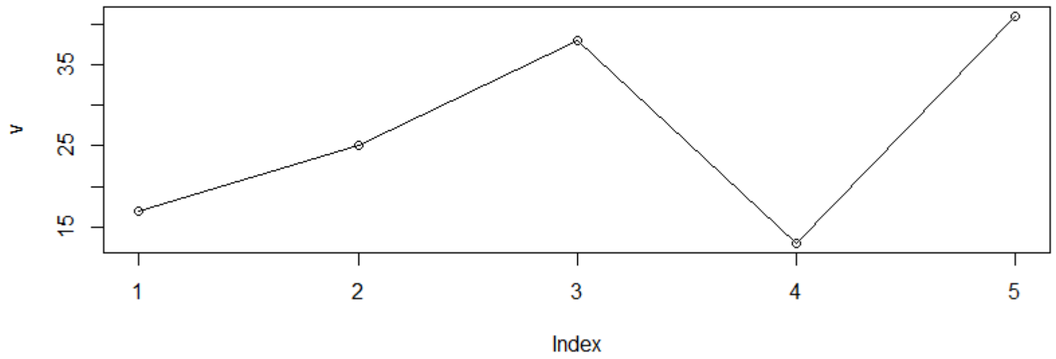


# Data visualisation in R

**Line Graph in R**

- ✓ A line graph in R is a type of chart that displays information as a series of data points. It employs points and lines to depict change over time.
- ✓ Line graphs are created by plotting different points on their X and Y coordinates, then connecting them with a line from beginning to end.
- ✓ The graph represents various values because it can move up and down depending on the appropriate variable.

```
# Create the data for the chart.
v <- c(17, 25, 38, 13, 41)

# Plot the bar chart.
plot(v, type = "o")
```

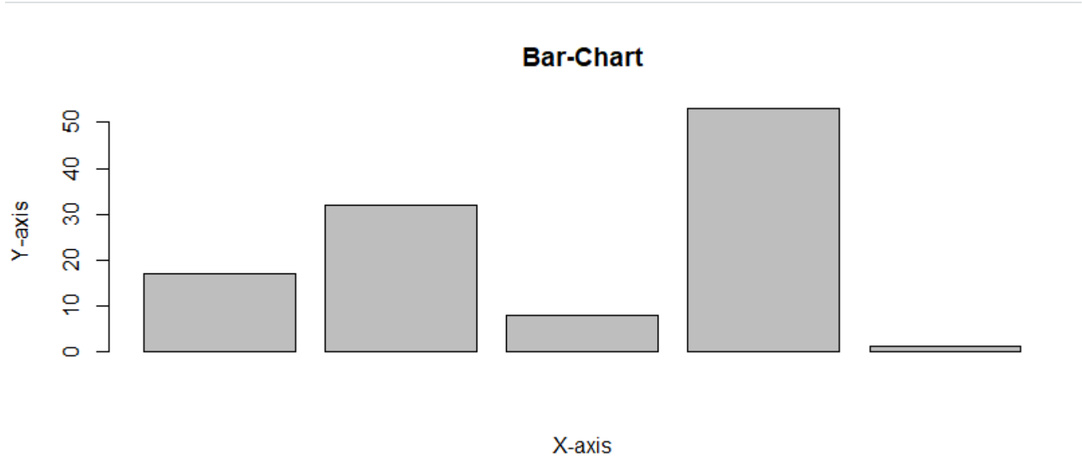


# Data visualisation in R

**Bar Chart in R**

- ✓ A bar chart is a graphical representation of categorical data that uses rectangular bars with heights or lengths proportional to the values they represent.
- ✓ In other words, it is a graphical representation of the dataset. The numerical values of variables that represent length or height are contained in these data sets.
- ✓ To generate bar charts, R employs the function barplot(). Vertical and horizontal bars can be drawn here.

```
# Create the data for the chart
A <- c(17, 32, 8, 53, 1)

# Plot the bar chart
barplot(A, xlab = "X-axis", ylab = "Y-axis", main ="Bar-Chart")
```

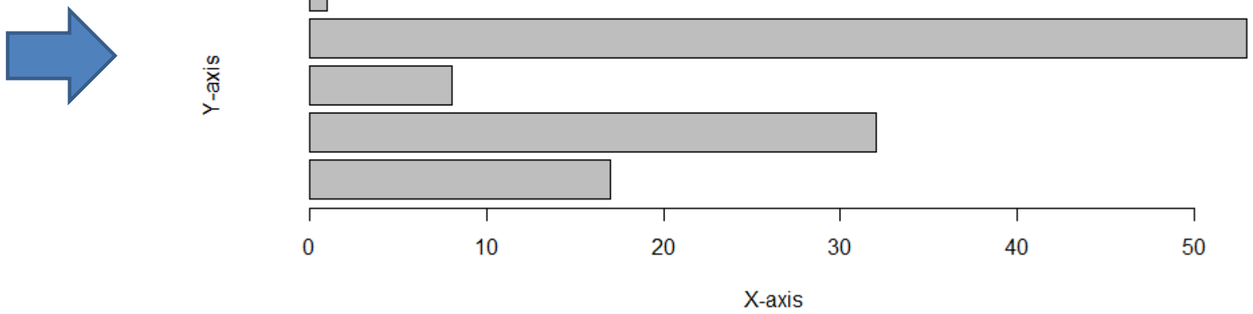


# Data visualisation in R

**Horizontal Bar Chart in R**

- ✓ In order to create a Horizontal Bar chart in R, Add horizontal parameter (horiz) == TRUE

```
# Create the data for the chart
A <- c(17, 32, 8, 53, 1)

# Plot the bar chart
barplot(A, horiz = TRUE, xlab = "X-axis",
        ylab = "Y-axis", main ="Bar-Chart")
```

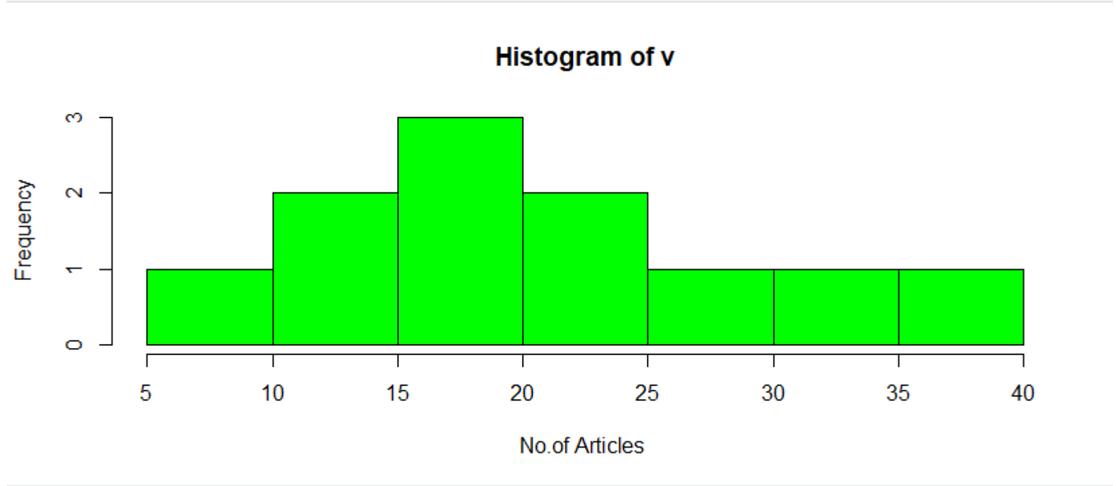


# Data visualisation in R

**Histograms in R**

- ✓ A histogram has a rectangular area to display statistical information that is proportional to a variable's frequency and width in successive numerical intervals.
- ✓ A graphical representation that divides a set of data points into different ranges. It has a unique feature that displays no gaps between the bars, similar to a vertical bar graph.

```
# Create data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32,
       14, 19, 27, 39)

# Create the histogram.
hist(v, xlab = "No.of Articles ",
     col = "green", border = "black")
```

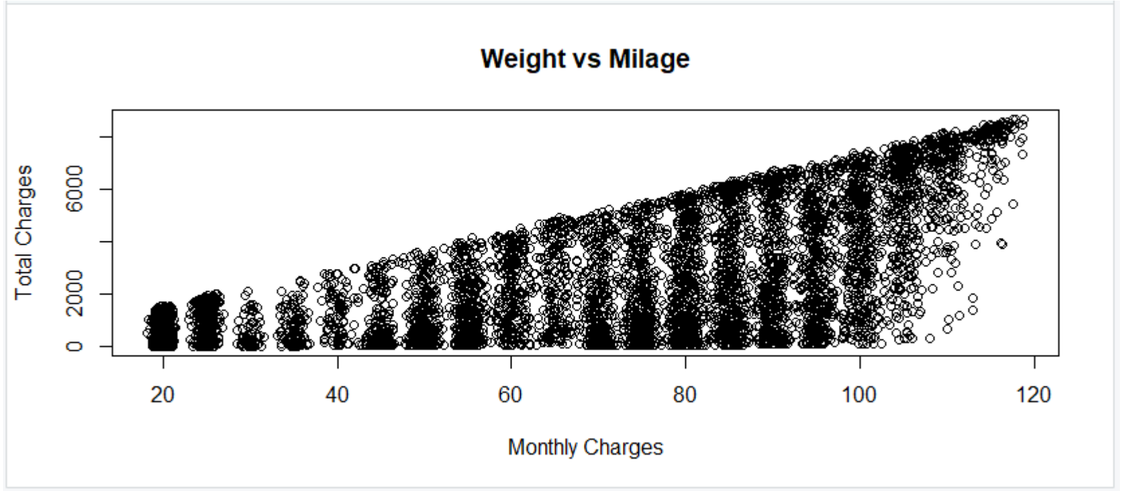


# Data visualisation in R

**Scatter Plot in R**

- ✓ A scatter plot is a set of dotted points in the horizontal and vertical axes that represent individual pieces of data.
- ✓ A graph in which the values of two variables are plotted along the X and Y axes, and the pattern of the resulting points reveals a relationship between them.

```
#Creating Data for the graph
input = ccxl[,c('MonthlyCharges','TotalCharges')]

#plotting the scatter plot
plot(x = input$MonthlyCharges, y = input$TotalCharges,
     xlab = "Monthly Charges",
     ylab = "Total Charges",
     main = "Monthly Charges vs Total Charges"
)
```

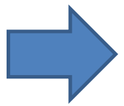


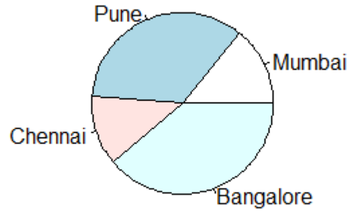
# Data visualisation in R

**Pie Chart in R**

- ✓ A pie chart is a circular statistical graphic divided into slices to show numerical proportions. It depicts a unique chart that employs "pie slices," with each sector displaying the relative sizes of data.
- ✓ A circular chart, also known as a circle graph, divides radii into segments that describe relative frequencies or magnitudes.

```
# Create data for the graph.
geeks<- c(23, 56, 20, 63)
labels <- c("Mumbai", "Pune", "Chennai", "Bangalore")

# Plot the chart.
pie(geeks, labels)
```



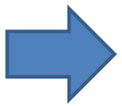


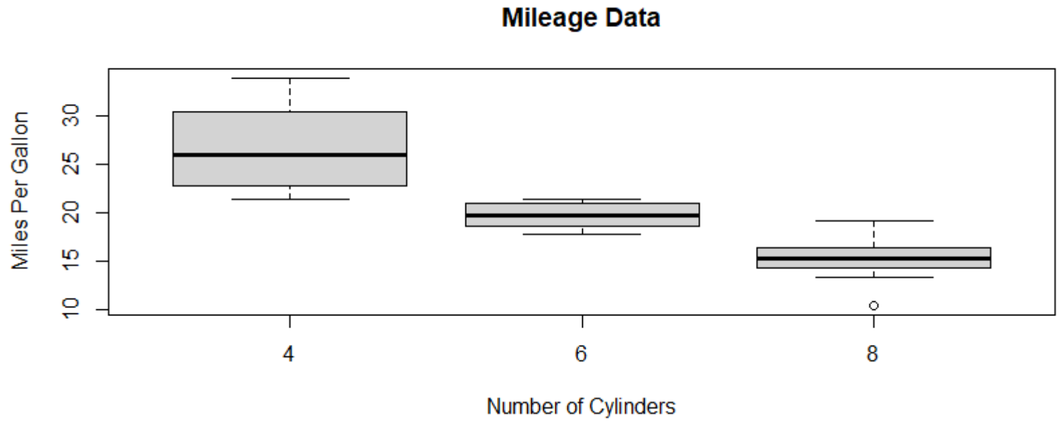
# Data visualisation in R

**Boxplot in R**

- ✓ A box graph is a type of chart that displays information in the form of distribution by drawing boxplots for each of them.
- ✓ This data distribution is based on five sets (minimum, first quartile, median, third quartile, maximum).
- ✓ In R, **boxplot()** function is used for creating boxplots.

```
# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars,
        xlab = "Number of Cylinders",
        ylab = "Miles Per Gallon",
        main = "Mileage Data")
```





theknowledgeacademy

# Module 6


## Description

| | |
|---|---|
| 1 | Introduction to Statistics in R |
| 2 | Introduction to Descriptive Statistics |
| 3 | Distributions in R |

# Statistics in R

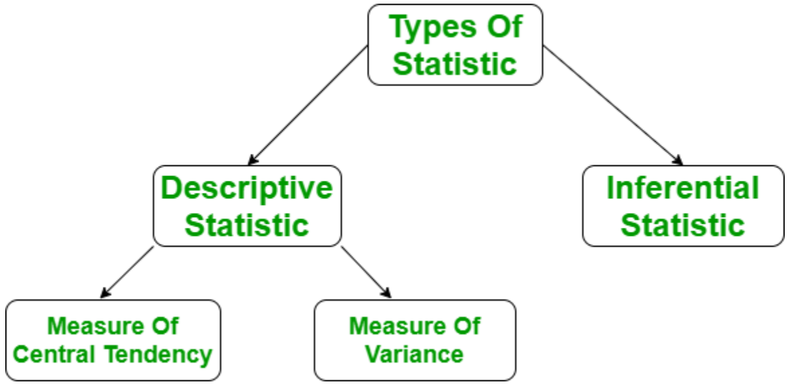
**Introduction to Statistics**

- ✓ Statistics is a branch of mathematics that deals with data collection, organisation, analysis, interpretation, and presentation. The statistical analysis aids in making the best use of the vast amounts of data available and improves the efficiency of solutions.
- ✓ R is a programming language that is used for statistical computing and graphics in the environment.

# Statistics in R

**Descriptive Statistics**

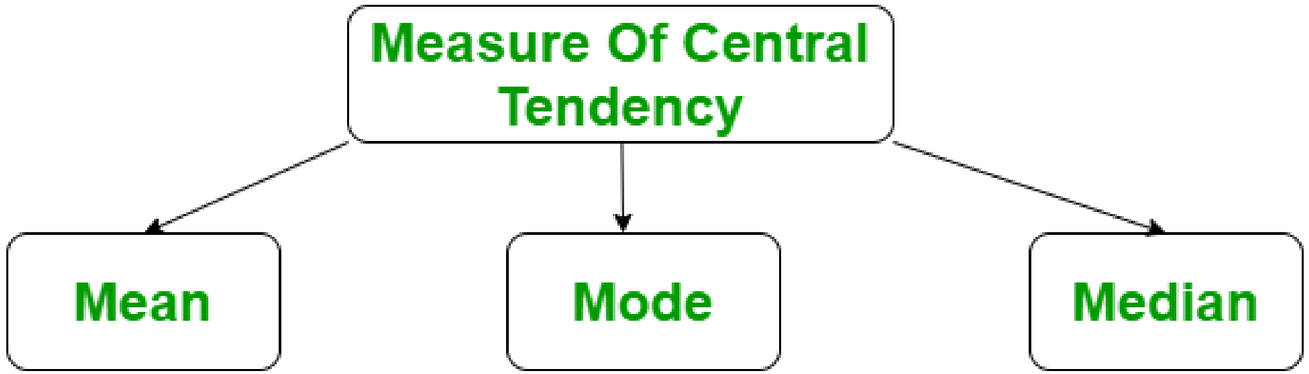
- ✓ When you receive a new data set to analyse, one of the first tasks is to find ways to summarise the data in a compact, easily understood manner. This is what descriptive statistics are all about. In other words,
- ✓ In descriptive analysis, we describe our data using various representative methods such as charts, graphs, tables, excel files, and so on. We describe our data in some way and present it in a meaningful way so that it can be easily understood in the descriptive analysis. Most of the time, it is performed on small data sets, and this analysis greatly aids us in forecasting future trends based on current findings.



# Statistics in R

**Measure of Central Tendency**

- A single value represents the entire set of data. It tells us where the central points are. There are three majorly types of central tendency measures:
  - Mean
  - Mode
  - Median



# Statistics in R

**Mean**

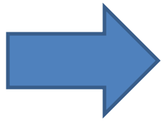
- ✓ It is calculated by dividing the total number of observations by the sum of the observations. It can also be defined as the sum divided by the count.
- ✓ Formula of Mean is:

$$\text{Mean } (\bar{x}) = \frac{\sum x}{n}$$

n = Number of observations

- ✓ Example :

```
# R program to illustrate
# Descriptive Analysis

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors = F)

# Compute the mean value
mean = mean(myData$Age)
print(mean)
```



```
[1] 28.78889
```

# Statistics in R

**Median**

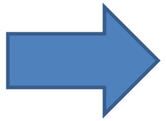
- ✓ It is the data set's median value. It divides the data into two parts. If the number of elements in the data set is odd, the center element is the median; otherwise, the median is the average of two central elements.
- ✓ Formula of Median is:

| Odd | Even |
|---|---|
| $\frac{n+1}{2}$ | $\frac{n}{2}, \frac{n}{2}+1$ |

n = Number of observations

- ✓ Example :

```
# R program to illustrate
# Descriptive Analysis

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors = F)

# Compute the median value
median = median(myData$Age)
print(median)
```



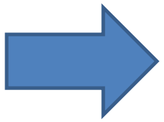
```
[1] 26
```

# Statistics in R

**Mode**

- ✓ It is the most frequently occurring value in the given data set. If the frequency of all data points is the same, the data set may not have a mode. We can also have multiple modes if we encounter two or more data points with the same frequency.
- ✓ Example :

```
# R program to illustrate
# Descriptive Analysis

# Import the library
library(modeest)

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors = F)

# Compute the mode value
mode = mfv(myData$Age)
print(mode)
```

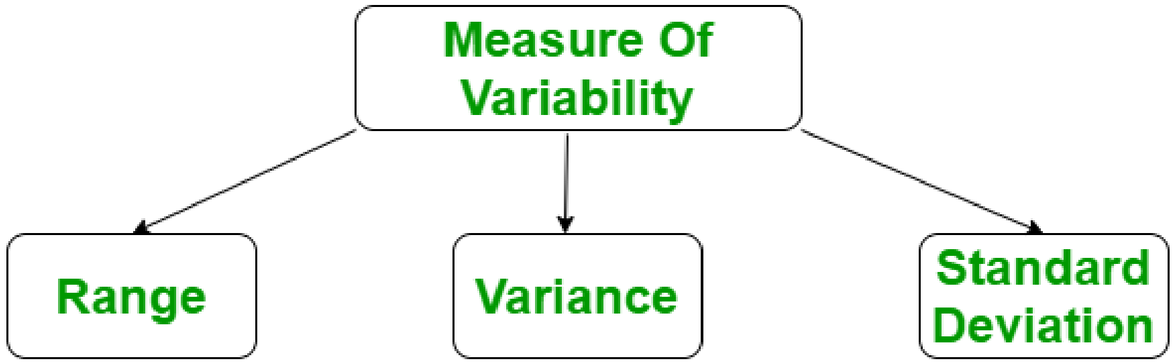


```
[1] 25
```

# Statistics in R

**Measure of Variability**

- The spread of data, or how well our data is distributed, is a measure of variability. The most common measures of variability are:
  - Range
  - Variance
  - Standard Deviation

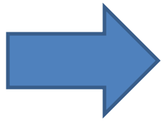


# Statistics in R

**Range**

- ✓ The range describes the difference between our data set's largest and smallest data points. The greater the range, the greater the data spread, and vice versa.
- ✓ Formula of Range is:
  - ✓ Range = greatest data value - the smallest data value
- ✓ Example :

```
# R program to illustrate
# Descriptive Analysis

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors = F)

# Calculate the maximum
max = max(myData$Age)
# Calculate the minimum
min = min(myData$Age)
# Calculate the range
range = max - min

cat("Range is:\n")
print(range)

# Alternate method to get min and max
r = range(myData$Age)
print(r)
```



```
Range is:
[1] 32


[1] 18 50
```

# Statistics in R

**Variance**

- ✓ It is defined as a squared deviation from the mean on average. It is calculated by squaring the difference between each data point and the average, also known as the mean, adding all of them, and then dividing by the number of data points in our data set.
- ✓ Formula of Variance is:

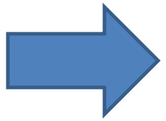
$$\sigma^2 = \frac{\sum(\chi - \mu)^2}{N}$$

Where, N = Number of terms

u = Mean

- ✓ Example :

```
# R program to illustrate
# Descriptive Analysis

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors = F)

# Calculating variance
variance = var(myData$Age)
print(variance)
```



```
[1] 48.21217
```

# Statistics in R

**Standard Deviation**

- ✓ It is calculated by taking square root of the variance. It is calculated by first determining the Mean, subtracting each number from the Mean, also known as the average, and squaring the result. Adding all of the values, dividing by the number of terms, and finally taking the square root.
- ✓ Formula of Standard Deviation is:

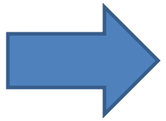
$$\sigma = \sqrt{\frac{\Sigma (x - u)^2}{N}}$$

Where, N = Number of terms

u = Mean

- ✓ Example :

```
# R program to illustrate
# Descriptive Analysis

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv", stringsAsFactors = F)

# Calculating Standard deviation
std = sd(myData$Age)
print(std)
```



```
[1] 6.943498
```

# Statistics in R

**Descriptive Analysis in R**

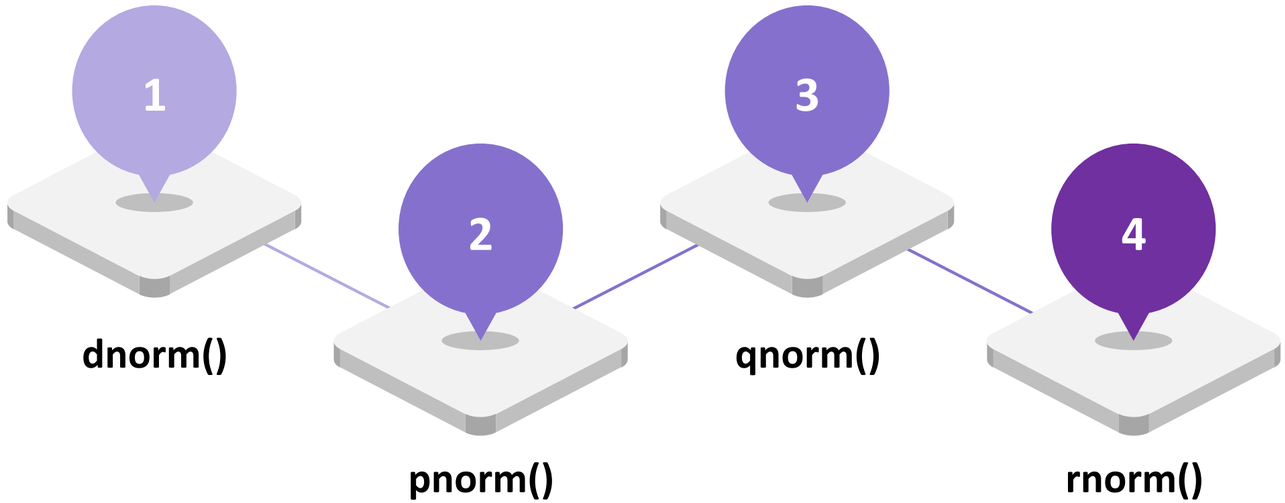
- ✓ Descriptive analyses involve simply describing the data using summary statistics and graphics.
- ✓ Before performing any computation, we must first prepare our data by saving it in external.txt or.csv files, and it is best practise to save the file in the current directory..
- ✓ Function in R that provides Descriptive Analysis :

| Analysis | R Function |
|---|---|
| Mean | mean() |
| Median | median() |
| Mode | mfv() [modeest] |
| Range of values (minimum and maximum) | range() |
| Minimum | min() |
| Maximum | maximum() |
| Variance | var() |
| Standrad Deviation | sd() |
| Sample quantiles | quantile() |
| Interquartile range | IQR() |
| Generic function | summary() |

# Statistics in R

**Normal Distribution in R**

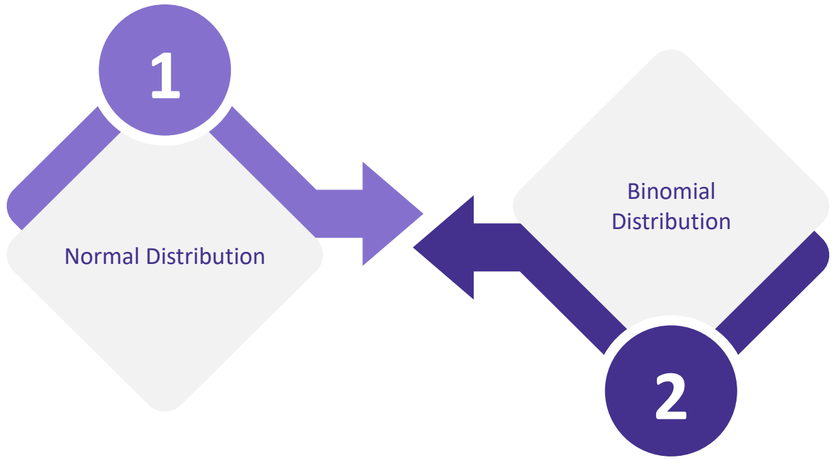
- ✓ The Normal Distribution is a statistical probability function that describes how data values are distributed. Because of its advantages in real-world scenarios.
- ✓ It is the most important probability distribution function used in statistics. For example, population height, shoe size, IQ level, rolling a dice, and many more.
- ✓ R includes four built-in functions for generating normal distributions:



# Statistics in R

**Distribution in R**

- ✓ The distribution provides a mathematical function that can be used to calculate the probability of any individual observation from the sample space. The probability density function, which describes the grouping or density of the observations, is described by this distribution. We can also compute the probability of an observation having a value equal to or less than a given value. A cumulative density function is a summary of these observations' relationships. There are 2 major distributions in R:



# Statistics in R

**dnorm()**

- ✓ In R programming, the dnorm() function calculates the density function of a distribution. In statistics, it is calculated using the following formula and syntax:
- ✓ Syntax : dnorm(x,mean,sd)

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

**pnorm()**

- ✓ The cumulative distribution function, pnorm(), measures the probability that a random number X has a value less than or equal to x; in statistics, it is denoted by-
- ✓ Syntax : pnorm(x,mean,sd)

$$F_X(x) = Pr[X \leq x] = \alpha$$

# Statistics in R

**qnorm()**

- ✓ The inverse of the pnorm() function is the qnorm() function. It takes the probability value and returns the probability value as output. It can be used to calculate the percentiles of a normal distribution.
- ✓ Syntax: qnorm(p, mean, sd)

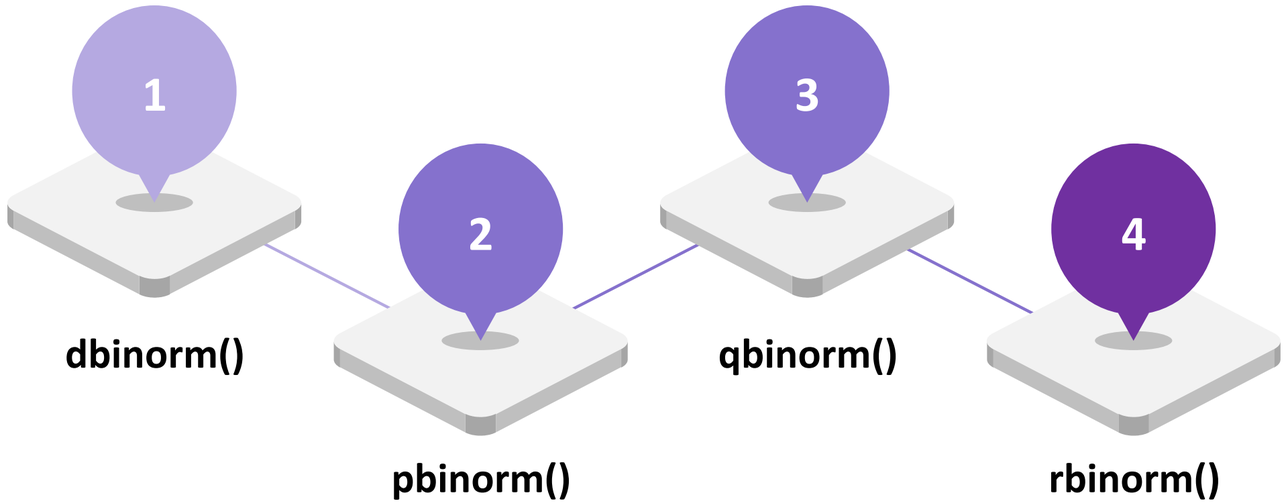
**rnorm()**

- ✓ In R programming, the rnorm() function is used to generate a vector of normally distributed random numbers.
- ✓ Syntax : rnorm(x,mean,sd)

# Statistics in R

**Normal Distribution in R**

- ✓ The binomial distribution in R is a statistical probability distribution. The binomial distribution is a discrete distribution with only two possible outcomes: success or failure. All of its trials are independent, the probability of success remains constant, and the previous outcome has no bearing on the next. The outcomes of different trials are distinct. The binomial distribution allows us to calculate individual and cumulative probabilities over a given range. Functions for handling binomial distribution in R :



# Statistics in R

**dbinorm()**

- ✓ This function is used to find the probability at a given value for data with a binomial distribution, i.e. it finds:

  P(X=k)

- ✓ Syntax : dbinorm(k, n, p)

**pbinorm()**

- ✓ The function pbinom() is used to calculate the cumulative probability of a data set following a binomial distribution up to a given value, i.e.

  P(X <= k)

- ✓ Syntax : pbinorm(k, n, p)

# Statistics in R

**dbinorm()**

- ✓ This function is used to find the nth quantile, that is, it finds k if P(x <= k) is given.
- ✓ Syntax : qbinorm(P, n, p)

**pbinorm()**

- ✓ This function creates n random variables with a given probability.
- ✓ Syntax : rbinorm(n,N, p)

theknowledgeacademy

# Module 7

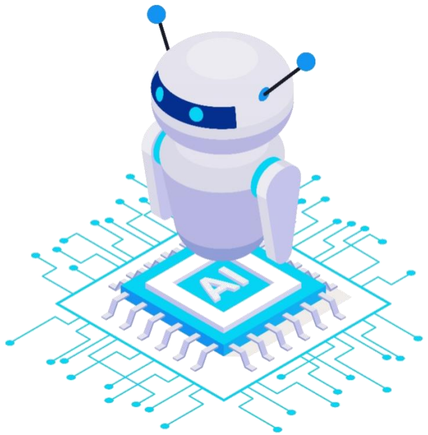

## Description

| | |
|---|---|
| 1 | Introduction to Machine Learning in R |
| 2 | Types of Machine Learning in R |
| 3 | Introduction to Supervised Learning in R |
| 4 | Introduction to Unsupervised Learning in R |

# Machine Learning in R

**Introduction**

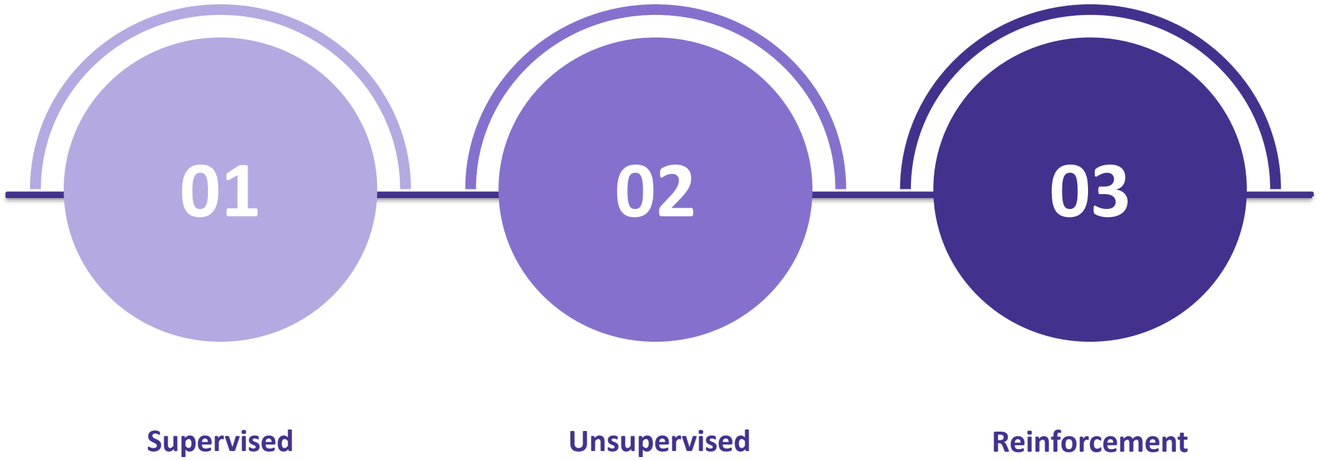
- ✓ Machine Learning is a subset of Artificial Intelligence (AI) that is used to create intelligent systems that can learn without being explicitly programmed.
- ✓ In machine learning, we develop algorithms and models that are used by an intelligent system to predict outcomes based on specific patterns or trends observed in the data.
- ✓ Machine learning is based on the unique principle of using data and the results of data to predict the rules that are stored in a model.
- ✓ This model is then used to forecast outcomes based on new data. The environment for machine learning in R programming can be easily configured using RStudio.



# Machine Learning in R

**Types of Machine Learning**

- ✓ Machine Learning is divided into 3 Major sections based on the problems and data. Those are:
  - ✓ Supervised Learning
  - ✓ Unsupervised Learning
  - ✓ Reinforcement Learning



# Machine Learning in R

**Supervised Machine Learning**

- ✓ Supervised learning, as the name implies, occurs in the presence of supervision. In short, supervised learning attempts to teach the machine using labels that already contain the correct answer.
- ✓ Following that, the machine will generate an example set of data so that the supervised algorithm can analyse the training data and produce the correct labelled data output.
- ✓ There are two major types of supervised machine-learning algorithms:
- ✓ **Classification:** A classification problem occurs when the output variable is a category, such as "Red," "Orange," "countable," or "not countable."
- ✓ **Regression:** When the output variable is a real value, such as "rupees" or "height," a regression is used.

# Machine Learning in R

**Regression in Machine Learning**

- ✓ Regression analysis is a statistical technique used to determine the relationship between two or more variables. One response variable and one or more predictor variables are always present.
- ✓ Regression analysis is commonly used to fit the data and predict the data for forecasting.
- ✓ It assists businesses and organisations in learning about the market behaviour of their product by utilising the dependent/response variable and the independent/predictor variable.
- ✓ Linear Regression Algorithm is used for solving Regression Problems.

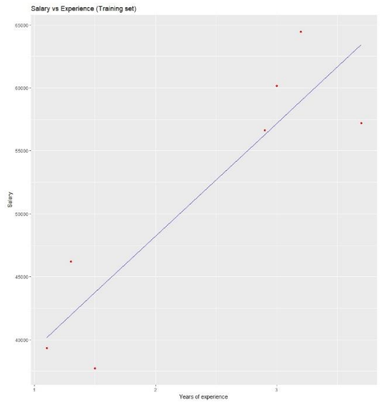
# Machine Learning in R

**Linear Regression in Machine Learning**

- ✓ It is a common type of predictive analysis. It is a statistical method for modelling the relationship between a dependent variable and a set of independent variables.
- ✓ In other words, It is a statistical method for summarising and studying relationships between two continuous (quantitative) variables. One variable denoted x, is an independent variable, while the other, denoted y, is a dependent variable. It is assumed that the two variables are linearly related. As a result, we seek a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable (x). The line that best fits the model is know as Regression Line.
- ✓ The equation of regression line is:

  **y = mx + c**

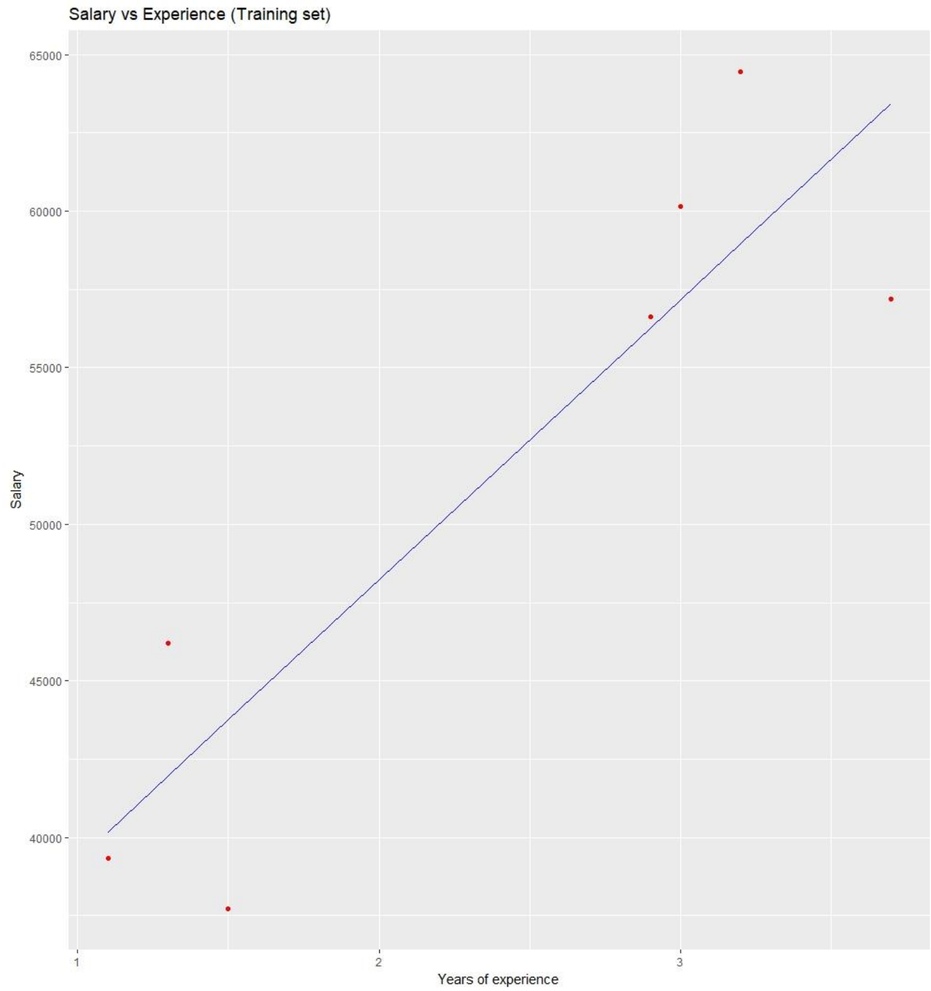
- ✓ Where y = predicted response
- ✓ c = intercept
- ✓ x = feature value
- ✓ m = slope



# Machine Learning in R

## Implementing Linear Regression in Machine Learning

```
# Simple Linear Regression
# Importing the dataset
dataset = read.csv('salary.csv')

# Splitting the dataset into the
# Training set and Test set
install.packages('caTools')
library(caTools)
split = sample.split(dataset$Salary, SplitRatio = 0.7)
trainingset = subset(dataset, split == TRUE)
testset = subset(dataset, split == FALSE)

# Fitting Simple Linear Regression to the Training set
lm.r= lm(formula = Salary ~ YearsExperience,
                        data = trainingset)
coef(lm.r)

# Predicting the Test set results
ypred = predict(lm.r, newdata = testset)

install.packages("ggplot2")
library(ggplot2)

# Visualising the Training set results
ggplot() + geom_point(aes(x = trainingset$YearsExperience,
                 y = trainingset$Salary), colour = 'red') +
geom_line(aes(x = trainingset$YearsExperience,
y = predict(lm.r, newdata = trainingset)), colour = 'blue') +

ggtitle('Salary vs Experience (Training set)') +
xlab('Years of experience') +
ylab('Salary')

# Visualising the Test set results
  ggplot() +
  geom_point(aes(x = testset$YearsExperience, y = testset$Salary),
             colour = 'red') +
  geom_line(aes(x = trainingset$YearsExperience,
            y = predict(lm.r, newdata = trainingset)),
            colour = 'blue') +
  ggtitle('Salary vs Experience (Test set)') +
  xlab('Years of experience') +
  ylab('Salary')
```



# Machine Learning in R

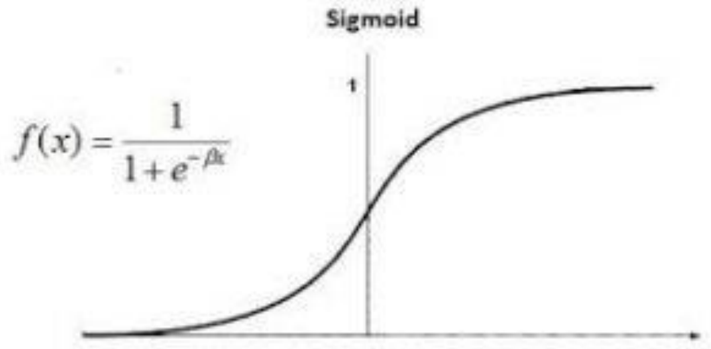
**Classification in Machine Learning**

- ✓ Classifiers in R are typically used to predict specific category-related information such as reviews or ratings such as good, best, or worst. There are 6 types of classification algorithms in R:

# Machine Learning in R

**Logistic Regression in Machine Learning**

- ✓ In R Programming, logistic regression is a classification algorithm used to determine the likelihood of event success and failure.
- ✓ When the dependent variable is binary (0/1, True/False, Yes/No), logistic regression is used. In a binomial distribution, the logit function is used as a link function.
- ✓ Binomial logistic regression is another name for logistic regression. It is based on the sigmoid function, with an output of probability and an input range ranging from -infinity to +infinity.



# Machine Learning in R

## Implementing Logistic Regression

```
# Installing the package
install.packages("caTools")    # For Logistic regression
install.packages("ROCR")       # For ROC curve to evaluate model

# Loading package
library(caTools)
library(ROCR)

# Splitting dataset
split <- sample.split(mtcars, SplitRatio = 0.8)
split

train_reg <- subset(mtcars, split == "TRUE")
test_reg <- subset(mtcars, split == "FALSE")

# Training model
logistic_model <- glm(vs ~ wt + disp,
                      data = train_reg,
                      family = "binomial")
logistic_model

# Summary
summary(logistic_model)

# Predict test data based on model
predict_reg <- predict(logistic_model,
                       test_reg, type = "response")
predict_reg

# Changing probabilities
predict_reg <- ifelse(predict_reg >0.5, 1, 0)

# Evaluating model accuracy
# using confusion matrix
table(test_reg$vs, predict_reg)

missing_classerr <- mean(predict_reg != test_reg$vs)
print(paste('Accuracy =', 1 - missing_classerr))
```

```
  predict_reg
   0 1
0  3 0
1  3 3
```

# Machine Learning in R

**Decision Trees in Machine Learning**

- ✓ It is essentially a graph that represents options. The graph's nodes or vertices represent events, and the graph's edges represent decision conditions. Its most common application is in Machine Learning and Data Mining.
- ✓ **Applications:**
- ✓ Email spam/non-spam classification, predicting whether a tumour is cancerous or not Typically, a model is built using known data, also known as the training dataset. The model is then validated and improved using a set of validation data. R has packages for creating and visualising decision trees.
- ✓ To create decision trees, the R package "party" is used.

# Machine Learning in R

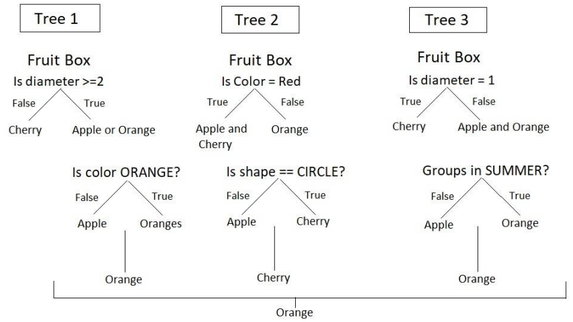
**Implementing Decision Trees**

```
# Load the party package. It will automatically load other
# dependent packages.
library(party)

# Create the input data frame.
input.data <- readingSkills[c(1:105), ]

# Give the chart file a name.
png(file = "decision_tree.png")

# Create the tree.
  output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.dat)

# Plot the tree.
plot(output.tree)

# Save the file.
dev.off()
```

# Machine Learning in R

**Random Forest in Machine Learning**

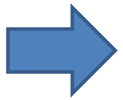
- ✓ Random Forest is a decision tree ensemble in R programming. It constructs and combines multiple decision trees in order to make more accurate predictions.
- ✓ It's a classification algorithm that isn't linear. When used alone, each decision tree model is used. Cases are estimated with an error that is not used when building the tree. This is known as an out of bag error estimate, and it is expressed as a percentage.
- ✓ They are called random because predictors are chosen at random during training. They are called forests because they make decisions based on the output of multiple trees.
- ✓ Random forest outperforms decision trees because a large number of uncorrelated trees (models) working together always outperform the individual constituent models.



# Machine Learning in R

## Implementing Random Forest

```
# Installing package
install.packages("caTools")       # For sampling the dataset
install.packages("randomForest")  # For implementing random forest algorithm

# Loading package
library(caTools)
library(randomForest)

# Splitting data in train and test data
split <- sample.split(iris, SplitRatio = 0.7)
split

train <- subset(iris, split == "TRUE")
test <- subset(iris, split == "FALSE")

# Fitting Random Forest to the train dataset
set.seed(120)  # Setting seed
classifier_RF = randomForest(x = train[-5],
                             y = train$Species,
                             ntree = 500)

classifier_RF

# Predicting the Test set results
y_pred = predict(classifier_RF, newdata = test[-5])

# Confusion Matrix
confusion_mtx = table(test[, 5], y_pred)
confusion_mtx

# Plotting model
plot(classifier_RF)

# Importance plot
importance(classifier_RF)

# Variable importance plot
varImpPlot(classifier_RF)
```



- **Model classifier_RF:**

```
> classifier_RF

Call:
 randomForest(x = train[-5], y = train$Species, ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.33%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         30          0         0  0.00000000
versicolor      0         29         1  0.03333333
virginica       0          2        28  0.06666667
```

The number of trees is 500 in the model and no. of variable tried at each split are 2. Classification error in setosa is 0.000 i.e 0%, Versicolor is 0.033 i.e 3.3% and virginica is 0.066 i.e 6.6% .

- **Confusion matrix:**

```
           setosa versicolor virginica
setosa         20          0         0
versicolor      0         20         0
virginica       0          3        17
```

# Machine Learning in R

**Naïve Bayes in Machine Learning**

- ✓ The naive Bayes classification method is a general classification method that employs a probability approach, also known as a probabilistic approach, and is based on Bayes' theorem with the assumption of feature independence. The model is trained on a training dataset before being used by the predict() function to make predictions.
- ✓ Formula: **P(A|B)=P(B|A)×P(A)P(B)**
- ✓ It is a sample method in machine learning methods, but it can be useful in certain situations. The training is simple and quick, requiring only that each predictor in each class be considered separately.
- ✓ **Application:**
  - ✓ It is commonly used in sentimental analysis.

# Machine Learning in R

## Implementing Naïve Bayes

```
library(caret)
## Warning: package 'caret' was built under R version 3.4.3
set.seed(7267166)
trainIndex = createDataPartition(mydata$prog, p = 0.7)$Resample1
train = mydata[trainIndex, ]
test = mydata[-trainIndex, ]

## check the balance
print(table(mydata$prog))
##
## academic    general vocational
## 105         45         50
print(table(train$prog))
```

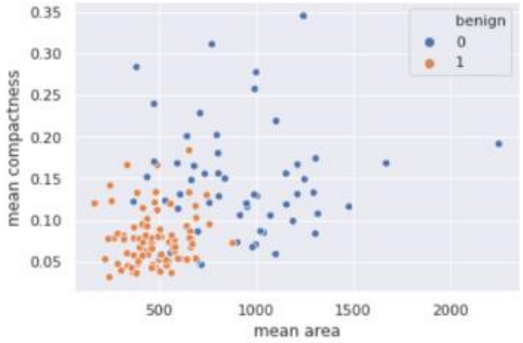
```
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   academic    general vocational
##  0.5248227  0.2269504  0.2482270
##
## Conditional probabilities:
##             science
## Y                [, 1]     [, 2]
##   academic   54.21622 9.360761
##   general    52.18750 8.847954
##   vocational 47.31429 9.969871
##
##             socst
## Y                [, 1]      [, 2]
##   academic   56.58108  9.635845
##   general    51.12500  8.377196
##   vocational 44.82857 10.279865
```

# Machine Learning in R

**K-NN Algorithm in Machine Learning**

- ✓ The K-NN classifier is another popular classifier. The k-nearest neighbor's algorithm (k-NN) is a non-parametric pattern recognition method that is commonly used for classification and regression. The input in both cases consists of the k closest training examples in the feature space. The output of k-NN classification is a class membership.
- ✓ **Applications:**
- ✓ Economic forecasting, data compression, and genetics are just a few of the applications.

```
# Write Python3 code here
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set()
breast_cancer = load_breast_cancer()
X = pd.DataFrame(breast_cancer.data, columns = breast_cancer.feature_names)
X = X[['mean area', 'mean compactness']]
y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
y = pd.get_dummies(y, drop_first = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)
sns.scatterplot(
    x ='mean area',
    y ='mean compactness',
    hue ='benign',
    data = X_test.join(y_test, how ='outer')
)
```

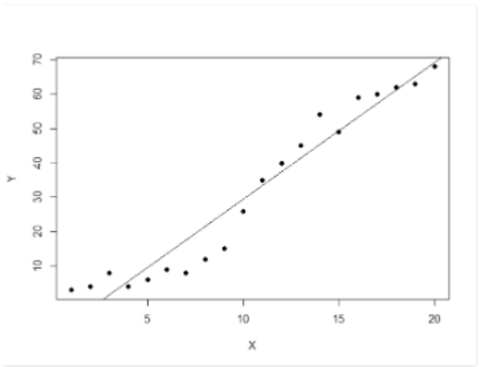


# Machine Learning in R

**SVM Algorithm in Machine Learning**

- ✓ A support vector machine (SVM) is a supervised binary machine learning algorithm that solves two-group classification problems using classification algorithms. They can categorise new text after providing an SVM model with sets of labelled training data for each category.
- ✓ SVM is primarily used for text classification problems. It categorises the unseen data. It is more commonly used than Naive Bayes. SVM is a fast and dependable classification algorithm that works well with small amounts of data.
- ✓ SVMs have a variety of applications, including bioinformatics, gene classification, and so on.

```
# Load the data from the csv file
dataDirectory <- "D:/" # put your own folder here
data <- read.csv(paste(dataDirectory, 'regression.csv', sep =""), header = TRUE)

# Plot the data
plot(data, pch = 16)

# Create a linear regression model
model <- lm(Y ~ X, data)

# Add the fitted line
abline(model)
```



# Machine Learning in R

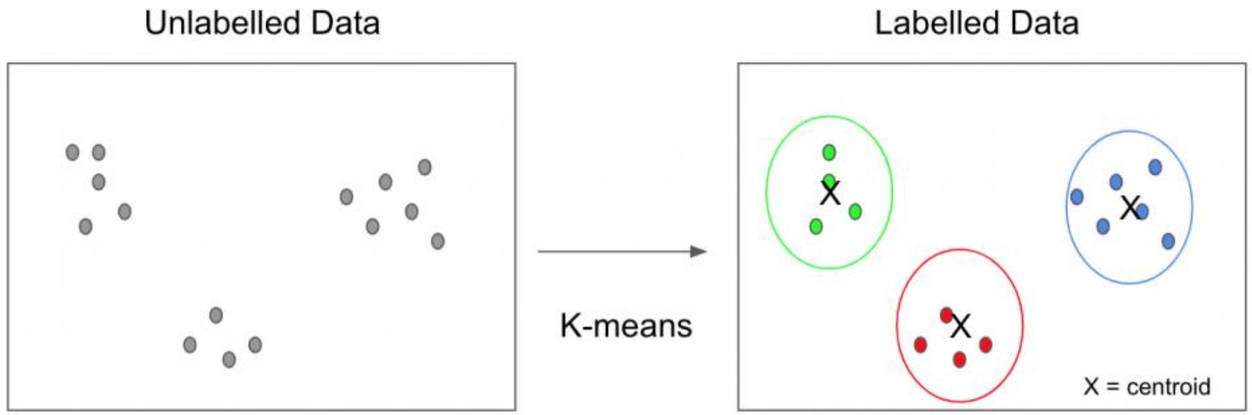
**Unsupervised Machine Learning**

- ✓ Unsupervised learning is the training of machines using unlabeled data that works without supervision. The machine's main task here is to separate the data based on similarities, differences, and patterns without any prior supervision. As a result, the machine is limited to discovering the hidden structure in unlabeled data on its own. For example, suppose we provide a group of previously unseen cats and dogs. The machine will then classify the cats and dogs based on their behaviour and nature. Now, when we provide images of dogs and cats, the machine will provide the results based on the classification. Unsupervised Learning is divided into two categories, as shown below:
- ✓ **Clustering:** A clustering problem is one in which the machine identifies the inherent groupings in the data, such as grouping customers based on shop visits.
- ✓ **Association:** An association problem is one in which we can discover the relationship between two events or items, such as people who buy item A also tend to buy item B.

# Machine Learning in R

**Clustering in Unsupervised Machine Learning**

- ✓ Clustering in R Programming Language is an unsupervised learning technique that divides a data set into several groups called clusters based on their similarity.
- ✓ Following data segmentation, several data clusters are produced. A cluster's objects all have similar characteristics.
- ✓ Clustering is used in data mining and analysis to find similar datasets.
- ✓ K-Means Algorithm is used to solve problems regarding clustering in Machine Learning .



# Machine Learning in R

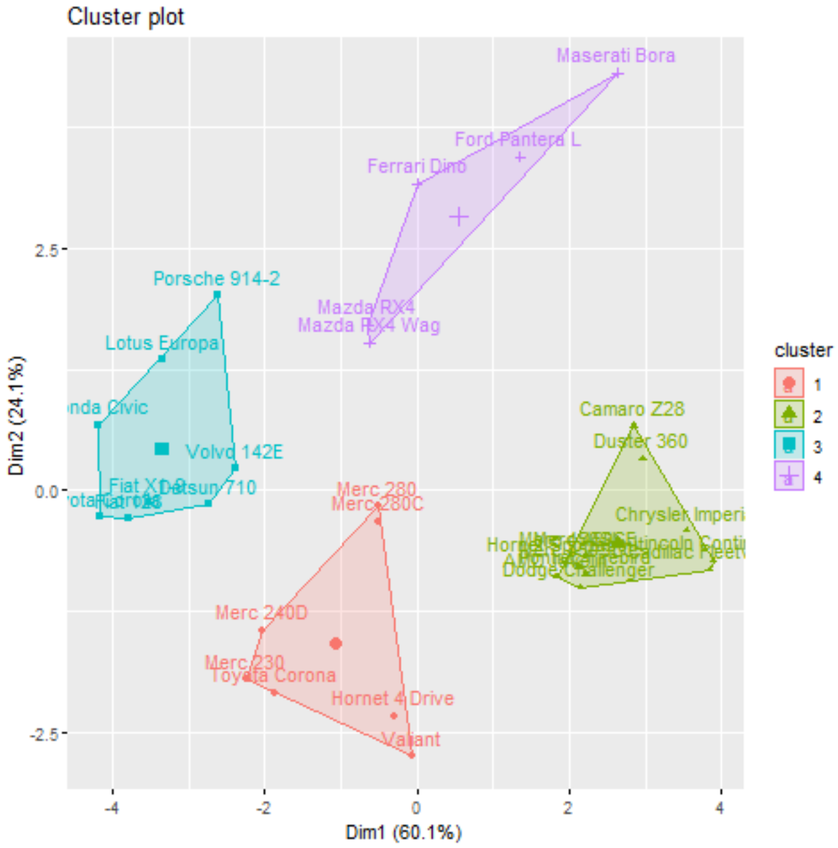
**K-Means Algorithm**

- K-Means is an unsupervised learning algorithm-based iterative hard clustering technique. The total number of clusters is pre-defined by the user, and the data points are clustered based on their similarity. This algorithm also determines the cluster's centroid.
- **Algorithm**:
  - Enter the number of clusters (K): Consider the case of k = 2 and 5 data points.
  - Assign each data point to a cluster at random: The red and green colours in the example below represent two clusters, each with its own set of random data points.
  - Calculate cluster centroids as follows: The cross mark represents the corresponding cluster's centroid.
  - Re-assign each data point to the cluster centroid closest to it: Because it is close to the centroid of the red cluster, the green data point is assigned to it.
  - Reposition the cluster centroid

# Machine Learning in R

## Implementing K-Means Algorithm

```
# Library required for fviz_cluster function
install.packages("factoextra")
library(factoextra)

# Loading dataset
df <- mtcars

# Omitting any NA values
df <- na.omit(df)

# Scaling dataset
df <- scale(df)

# output to be present as PNG file
png(file = "KMeansExample.png")

km <- kmeans(df, centers = 4, nstart = 25)

# Visualize the clusters
fviz_cluster(km, data = df)

# saving the file
dev.off()

# output to be present as PNG file
png(file = "KMeansExample2.png")

km <- kmeans(df, centers = 5, nstart = 25)

# Visualize the clusters
fviz_cluster(km, data = df)

# saving the file
dev.off()
```



# Machine Learning in R

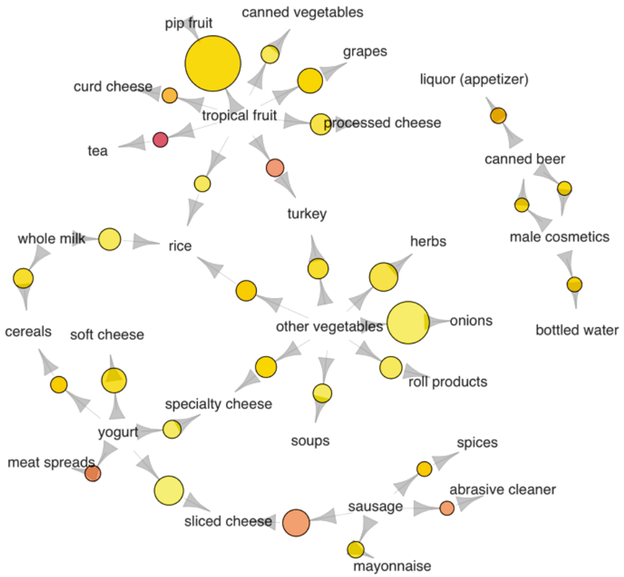
**K-Means Algorithm**

- K-Means is an unsupervised learning algorithm-based iterative hard clustering technique. The total number of clusters is pre-defined by the user, and the data points are clustered based on their similarity. This algorithm also determines the cluster's centroid.
- **Algorithm**:
  - Enter the number of clusters (K): Consider the case of k = 2 and 5 data points.
  - Assign each data point to a cluster at random: The red and green colours in the example below represent two clusters, each with its own set of random data points.
  - Calculate cluster centroids as follows: The cross mark represents the corresponding cluster's centroid.
  - Re-assign each data point to the cluster centroid closest to it: Because it is close to the centroid of the red cluster, the green data point is assigned to it.
  - Reposition the cluster centroid

# Machine Learning in R

**Association Rules Mining in Unsupervised Machine Learning**

- Association Rule Mining in R is an unsupervised non-linear algorithm used to discover how items are related to one another. Its frequent Mining feature displays which items appear together in a transaction or relation. Retailers, grocery stores, and an online marketplace with a large transactional database primarily use it. Similarly, any online social media, marketplace, or e-commerce website can predict what you will buy next by using recommendation engines. The recommendations you receive on items or variables while checking out your order are the result of Association rule mining based on previous customer data. Association can be measured in three ways:
  - Support
  - Confidence
  - Lift



# Machine Learning in R

**Support**

- ✓ The percentage of transactions in which an item set appears indicates how popular an item is.

**Confidence**

- ✓ Confidence expresses how likely it is that item Y will be purchased when item X is purchased, expressed as X -> Y.
- ✓ Thus, it is determined by the proportion of transactions that include item X and item Y. Confidence may overestimate the significance of association.

**Lift**

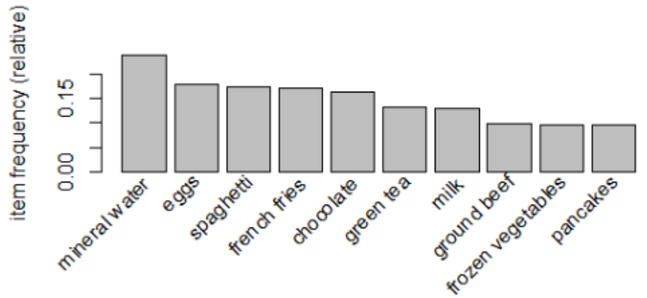
- ✓ Lift expresses how likely item Y is to be purchased when item X is purchased while controlling for the popularity of item Y.

# Machine Learning in R

## Implementing Association Rule Mining

```
# Installing Packages
install.packages("arules")
install.packages("arulesViz")

# Loading package
library(arules)
library(arulesViz)

# Fitting model
# Training Apriori on the dataset
set.seed = 220 # Setting seed
associa_rules = apriori(data = dataset,
                        parameter = list(support = 0.004,
                                         confidence = 0.2))

# Plot
itemFrequencyPlot(dataset, topN = 10)

# Visualising the results
inspect(sort(associa_rules, by = 'lift')[1:10])
plot(associa_rules, method = "graph",
     measure = "confidence", shading = "lift")
```

```
> associa_rules = apriori(data = dataset, parameter = list(support = 0.004, confidence = 0.2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target
        0.2    0.1    1 none FALSE            TRUE       5   0.004      1     10  rules
  ext
 TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 30

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7501 transaction(s)] done [0.01s].
sorting and recoding items ... [114 item(s)] done [0.05s].
creating transaction tree ... done [0.01s].
checking subsets of size 1 2 3 4 done [0.14s].
writing ... [811 rule(s)] done [0.31s].
creating S4 object  ... done [0.01s].
```



# Machine Learning in R

**Reinforcement Machine Learning**

- ✓ The reinforcement learning method is all about taking appropriate action in order to maximise reward in a specific situation. It is guided by various machines to find the best solution to a problem in a given situation. The difference between supervised learning and reinforcement learning is that in supervised learning, the data contains a key to the correct answer, which the agent uses to find the answer, whereas in reinforcement, the agent decides what to do to complete the given task. For example, when travelling from one location to another, we always consider the shortest and best route to our destination. The following are some key points in reinforcement learning:
- ✓ **Input:** The input should come from the initial stage where the model is created.
- ✓ **Output:** Every problem has multiple outputs.
- ✓ **Training:** Because training is input-dependent, the model will return the state, and the user will decide whether to reward or reject the model based on its output.

# Congratulations

Congratulations on completing this module!

# Contact Us

info@theknowledgeacademy.com

www.theknowledgeacademy.com/tickets

https://uk.trustpilot.com/review/theknowledgeacademy.com

theknowledgeacademy