Data Analytics using R

UNIT-I

Introduction to R-Features of R, Installation of R, Getting Started, Variables in R, Mathematical Operators and Vectors, Assigning Variables, Special Numbers, classes, different types of numbers, changing classes, examining variables, the workplace, library of package, getting to know a package. Input of Data, Output in R, In-Built Functions in R, Packages in R

Features of R

R allows branching and looping as well as modular programming using functions.

R allows integration with different programming languages like C, C++, .Net, Python etc.

R has an extensive community of contributors.

R has an effective data handling and storage facility for numeric and textual data.

R provides a collection of operators for calculations on arrays, lists, factor, vectors, data frame and matrices.

R provides large and integrated collection of tools for data analysis and statistical functions.

R provides graphical facilities for data analysis and can show result both in soft and hard copies.

R is an integrated suite of software facilities for data manipulation, calculation and graphical facilities for data analysis and display

Installation of R

R can be installed from R-3.2.2 for Windows (32/64bit) and save it in a local directory. In windows, installer (.exe) with a name "R-version-win.exe" will be downloaded. Double click and run the installer accepting the default settings. R is available for both the versions of windows (32-bit/ 64-bit.)

Getting Started

R Studio has four main window sections:

Top-Left Section: To write and save R code (Script section)

Bottom-Left Section: To execute R code and doing calculations. The nature and values of all variables and objects appear here (Console section)

Top-Right Section: To manage datasets and variables (Data section)

Bottom-Right Section: To display plots ,installed packages and seek help on R functions (Plot, Packages and Help Section)

Variables in R

Naming Variables: A variable in R can store any object in R including atomic vector, list, matrix, array, factor and data frame. A valid variable name consists of letters, numbers and the dot or underline characters.

Assigning Values to Variables: In R, an assignment to a variable can be done in three ways = , <- and -> sign.

Finding Variables: To know all the variables currently available in the workspace we use the ls() function.

Removing Variables: Variable can be deleted by using the rm() function along with variable name.

Arithmetic Operations in R

Addition & Subtraction

```
2+2  # Addition

## [1] 4

2-2  # Substraction

## [1] 0
```

Multiplication & Division

```
2*2 # Multiplication

## [1] 4

2/2 # Division

## [1] 1
```

Squaring a number

```
2**2

## [1] 4
```

Another way to square a number is

```
3^2 # Square of three

## [1] 9
```

Square Root

```
2^(1/2)
## [1] 1.414214
```

Alternatively you can use sqrt function

```
sqrt(2)
## [1] 1.414214
```

Integer Division- Returns Integer after division

```
5%/%2
## [1] 2
```

Modulus- Returns Remainder after division

```
5%%2
## [1] 1
```

Relational Operators in R

```
x=5 # we have assigned x value of 5
y=7  # we have assigned y value of 7
# To check value of X just type x in console
X
## [1] 5
y
## [1] 7
#The '>' operator is caled relational operation and mean greater than
x>y
## [1] FALSE
y>x
## [1] TRUE
```

Logical Operations in R

```
#Character "!" is Logical NOT
# so a not TRUE will evaluate to FALSE
!TRUE
## [1] FALSE
```

```
# Symbol "&" is used for elementwise Logical AND for example TRUE & TRUE
will evaluate to TRUE, while TRUE and FALSE will evaluate to FALSE

TRUE & TRUE

## [1] TRUE

TRUE & FALSE

## [1] FALSE

TRUE | TRUE

## [1] TRUE

TRUE | FALSE

## [1] TRUE

FALSE | FALSE

## [1] FALSE
```

The ‘&’ and ‘|’ symbol perform logical operations elementwise.

```
TRUE && TRUE

## [1] TRUE

TRUE & FALSE

## [1] FALSE

TRUE & FALSE

## [1] FALSE

FALSE && FALSE

## [1] FALSE
```

Vector

Lets create a vector containing three numeric values. This will be an example of Numeric Vector

```
c(1,4,7)

## [1] 1 4 7

# To check length of the vector we can use length function.

length(c(1,4,7))

## [1] 3

#Lets assign this to a variable for future use

Num_variable<-c(1,4,7)
```

Let’s create a vector “Names” that contain names of person appearing in an Exam. This will be a kind of character vector

```
Names<-c("Arvind", "Krishna", "Rahul", "Saurabh", "Venkat","Sucharitha")
```

```
# Lets extract first element of Names vector. To extract first element we need to use "[" and 1 value
```

```
Names[1]
```

```
## [1] "Arvind"
```

```
Names[3]
```

```
## [1] "Rahul"
```

Class of Vector

We can also check the types of vector using different functions such as

```
class(Names)
```

```
## [1] "character"
```

```
class(Num_variable)
```

```
## [1] "numeric"
```

```
typeof(Num_variable)
```

```
## [1] "double"
```

Adding Vectors of different type

Let’s add character and numeric vector and check the result

```
mix_vector<- c(1,2,"Amish")
```

```
class(mix_vector)
```

```
## [1] "character"
```

The mix_vector we created had multiple data types-Numeric and Character, however R converts the multiple data type to a single data type through a process called coercion. Logical values are converted to numbers: TRUE is converted to 1 and FALSE to 0.

Values are converted to the simplest type required to represent all information.

The ordering is roughly logical < integer < numeric < complex < character < list.

Accessing elements of vectors

Let’s try to access elements of vector using negative indexing.

Names[-1] # All Elements except the first element will be printed

```
## [1] "Krishna"    "Rahul"      "Saurabh"    "Venkat"     "Sucharitha"
## [6] "Ajay"       "Amit"
```

Names[c(-1,-2)] # All Elements except the first and second elements will be printed

```
## [1] "Rahul"      "Saurabh"    "Venkat"     "Sucharitha" "Ajay"
## [6] "Amit"
```

Names[-c(1,2)] # All Elements except the first and second elements will be printed, we just used negative at the start of vector , rather than using negatives in front of multiple numbers.

```
## [1] "Rahul"      "Saurabh"    "Venkat"     "Sucharitha" "Ajay"
## [6] "Amit"
```

Creating Vector using In Built Functions

seq(1,10, by = 1) # Create a sequential vector from 1 to 10 which increases by 1 in length.

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

seq(1,10, by = 0.5)

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0
```

seq(1,10, by = 0.5)

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0
```

Sorting a Vector

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

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

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

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

Change an Item

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

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

# Print fruits
fruits
```

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

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

numbers
```

Arithmetic Operator in Vectors in R

We have performed arithmetic operations in previous chapters. Let's see how can we perform arithmetic operations in Vectors.

```
V1<- c(1,2,3,4) # created a vector V1

V2<- c(5,6,7,8) # created a vector V2
```

Lets perform addition

```
V12<-V1+V2

V12 #Print result of V12, which is sum of V1 and V2 vector.

## [1]  6  8 10 12

V12_mult<-V1*V2

V12_mult

## [1]  5 12 21 32

V12_division <-V2/V1

V12_division

## [1] 5.000000 3.000000 2.333333 2.000000
```

Lets convert the double to numeric

```
V12_division<-as.integer(V12_division)
```

V12_division

Vectors Vs List

Vector:	List:
Vector: In many programming languages, a vector is a dynamic array that can resize itself automatically. Vectors typically provide constant-time access to individual elements and amortized constant-time for appending new elements.	List: A list is a more generic term and can refer to various data structures. In some languages, a list is synonymous with an array or a vector, while in others, it might refer to a linked list. Linked lists have nodes where each node contains a value and a reference (or link) to the next node.
Vectors often allocate contiguous memory, making them efficient for random access to elements.	Lists can use contiguous or non-contiguous memory depending on the type. Linked lists, for example, use non-contiguous memory as each element points to the next one.
Vectors are generally more efficient when it comes to random access to elements due to their contiguous memory allocation.	Lists, especially linked lists, can be more efficient for inserting or deleting elements in the middle, as it doesn't require shifting elements like in a vector.
Vectors are often more rigid in size, but some languages provide resizable vectors. Once a vector reaches its capacity, it may need to allocate a new, larger chunk of memory and copy elements over.	Lists, especially linked lists, can easily grow or shrink without the need for reallocation. Each element points to the next one, so inserting or removing elements is more straightforward.
In many languages, vectors have specific syntax and methods/functions for operations like appending, accessing elements, and resizing.	Lists can be more generic, and the term might be used interchangeably with arrays or other data structures. Syntax and methods can vary widely depending on the language and the specific type of list (e.g., linked list, array list).

Assigning Variables

R, you can assign variables using the assignment operator, which is <- or =. Here's an example of how to assign a value to a variable:

```
# Using the assignment operator <-

x <- 10

# Assigning values to multiple variables

a <- 3

b <- 7

c <- 11
```

Both <- and = can be used for variable assignment, but conventionally, <- is more commonly used. It's a good practice to stick to one style consistently in your code.

Additionally, you can use the assign() function to assign values to variables programmatically:

```
# Using assign() function

variable_name <- "z"

assign(variable_name, 15)


# Using the equal sign =

y = 5
```

Special Numbers

Inf (Infinity):

•Represented by Inf or 1/0.

•Used to represent positive infinity.

```
x <- Inf
```

-Inf (Negative Infinity):

•Represented by -Inf or -1/0.

•Used to represent negative infin

```
y <- -Inf
```

NaN (Not a Number):

•Represented by NaN.

•Used to represent undefined or unrepresentable value resulting from an undefined operation.

```
z <- 0/0
```

NA (Not Available):

•Represented by NA.

•Used to represent missing or undefined data.

```
missing_data <- NA
```

Changing classes in R Programming

as. Functions: The **as.** functions are used to coerce or convert objects from one class to another. For example:

```
# Convert numeric to character

x <- 123

x_char <- as.character(x)


# Convert character to numeric

y <- "456"

y_num <- as.numeric(y)
```

as() Function: The **as()** function is a more general way to coerce objects to a specified class:

```
# Convert numeric to character

x <- 123

x_char <- as(x, "character")


# Convert character to numeric

y <- "456"

y_num <- as(y, "numeric")

# Convert numeric to character

x <- 123

x_char <- as(x, "character")


# Convert character to numeric

y <- "456"

y_num <- as(y, "numeric")
```

factor() Function: If you want to convert a vector to a factor, you can use the **factor()** function:

```
# Convert a character vector to a factor

colors <- c("red", "green", "blue")

factor_colors <- factor(colors)
```

as.Date() Function: To convert a character or numeric vector to a Date object, you can use **as.Date()**:

```
# Convert character to Date

date_str <- "2024-03-06"

date_obj <- as.Date(date_str)
```

matrix() Function: If you want to convert a vector to a matrix, you can use the **matrix()** function:

```
# Convert a numeric vector to a matrix

vec <- 1:6

mat <- matrix(vec, nrow = 2, ncol = 3)
```

R Packages

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

All the packages available in R language are listed at R Packages.

Below is a list of commands to be used to check, verify and use the R packages.

Get the list of all the packages installed

```
library()
```

Install directly from CRAN

The following command gets the packages directly from CRAN webpage and installs the package in the R environment. You may be prompted to choose a nearest mirror. Choose the one appropriate to your location.

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

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

Install package manually

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

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

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

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

Input in R

Input of Data from Terminal: The scan function is used to take data from the user at the terminal.

Input of Data through R Objects: There are many types of R-objects including Vectors, Lists, Matrices, Arrays, Factors and Data Frames.

Output in R

print() Function: Print cannot combine two or more strings, variables, a string and a variable.

cat() Function: The cat() function is an alternative to print that lets you combine multiple items into a continuous output.

Inbuilt Functions in R

Mathematical Functions: R can also be used as a calculator along with facility to use many mathematical functions. Ex: sqrt, abs, floor, ceiling etc.

Trigonometric Functions: R provides the user an ability to compute the result using different trigonometric functions. Ex: sin, cos, tan etc.

Logarithmic Functions: R has an extensive facility to provide log of a number with proper specification of the base . Ex : log with base 10 and natural base

Date and Time Functions: Dates and times have special classes in R that allow for numerical and statistical calculations.

Sequence Function: A sequence is a set of related numbers, events, date etc. that follow each other in a particular order. R has a number of facilities for generating commonly used sequences of numbers.

Repeat Function: Function rep is used to replicates the values in a vector. It is a very powerful feature in R which helps the user to create a set of values in an easy manner

Strings

Creating a String: String in R is written within a pair of single quote or double quotes. Concatenating Strings: The paste() function concatenates several strings together. It creates a new string by joining the given strings end to end.

Formatting of Strings: Strings can be formatted to a specific style according to the requirement of the user using format() function.

Counting number of character: nchar() function is used to count the number of characters including spaces in a string.

Change case: The functions toupper() and tolower() functions are used to change the case of characters of a string.

Extracting parts of a string: The substring() or substr() function extracts parts of a string depending on the index position of the string.

Searching Matches: The grep() function is used for searching the matches.

Changing String to expression: The eval() function evaluates an expression only and not a string..

Split the Elements of Vector: The function strsplit() is used to split the elements of a character vector into substrings according to the matches to substring split within them.

Replace substring with other: sub (substring) and gsub (global substring) functions are used if we want to replace one substring with another with in a string.

Packages in R

Standard Packages: R packages are a collection of R functions, complied code and sample data. They are stored under a directory called "library" in the R environment.

Packages in R

packages in library ‘C:/Program Files/R/R-3.2.2/library’:

base	The R Base Package
boot	Bootstrap Functions (Originally by Angelo Canty for S)
class	Functions for Classification
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.
codetools	Code Analysis Tools for R
compiler	The R Compiler Package

datasets	The R Datasets Package
foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...
graphics	The R Graphics Package
grDevices	The R Graphics Devices and Support for Colours and Fonts
grid	The Grid Graphics Package
KernSmooth	Functions for Kernel Smoothing Supporting Wand & Jones (1995)
lattice	Trellis Graphics for R
MASS	Support Functions and Datasets for Venables and Ripley's MASS
Matrix	Sparse and Dense Matrix Classes and Methods
methods	Formal Methods and Classes
mgcv	Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation
nlme	Linear and Nonlinear Mixed Effects Models
nnet	Feed-Forward Neural Networks and Multinomial Log-Linear Models
parallel	Support for Parallel computation in R
rpart	Recursive Partitioning and Regression Trees
spatial	Functions for Kriging and Point Pattern Analysis
splines	Regression Spline Functions and Classes
stats	The R Stats Package
stats4	Statistical Functions using S4 Classes
survival	Survival Analysis
tcltk	Tcl/Tk Interface
tools	Tools for Package Development
utils	The R Utils Package

Contributed Packages: There are thousands of contributed packages for R, written by many authors. Some of these packages implement specialized statistical methods, others give access to data or hardware, and others are designed to complement textbooks.