UNIT - 1

Descriptive statistics, Introduction to Analytics, Business Understanding, Introduction to R- Basics, The R Environment, Inductive and Deductive Logic, Installation of R

R. K. Sharma, CSED, TIET Patiala

Covered in this Presentation:

Introduction to R- Basics, The R Environment, Inductive and Deductive Logic, Installation of R

What is R

R is a free, open-source language used as a statistical and visualization software. It can handle structured as well as semi-structured data.

Features of R

- Is available across all platforms, such as Linux, Mac, and Windows
- Has the ability to integrate with the procedures written in the C, C++, .Net, Python, or FORTRAN languages
- Has an effective data handling and storage facility
- Provides a wide variety of integrated collection of tools for data analytics

R has a worldwide repository system –CRAN. The Comprehensive R Archive Network (CRAN) is a network of sites that acts as the primary web service distributing R sources and binaries, extension packages, and documentation.

R and Data Analytics

- Open Source
  - R is a free tool
- Statistical Algorithms
  - R is a statistical software where complex stats models like linear regression, logistic regression, hypothesis testing, ANOVA(Analysis Of Variance), GLM(Generalized Linear Model), etc., can be run.
- Visualization
  - R has some great tools to aid data visualization to create graphs, bar charts, multi-panel lattice charts, scatter plots, and new custom-designed graphics.

R and Data Analytics …

- ML Algorithms
  - ML algorithms like SVM, NaivesBayes theorem, XGboost, Decision tree, and Random forest are available in R readily. These algorithms have proven to be better over time and provide good accuracy of results.
- Recognized by other Software
  - One can now write R codes in SAS as R codes are widely used, and programmers are getting familiar with these.
  - R can handle semi-structured data and has algorithms built.
- Customized Algorithms
  - Programmers can define their customized algorithms in R and develop their own algorithms and packages.

Data Types in R

- Consider the statement: int a = 5
  - Data type is int here, a is the variable and 5 is the value

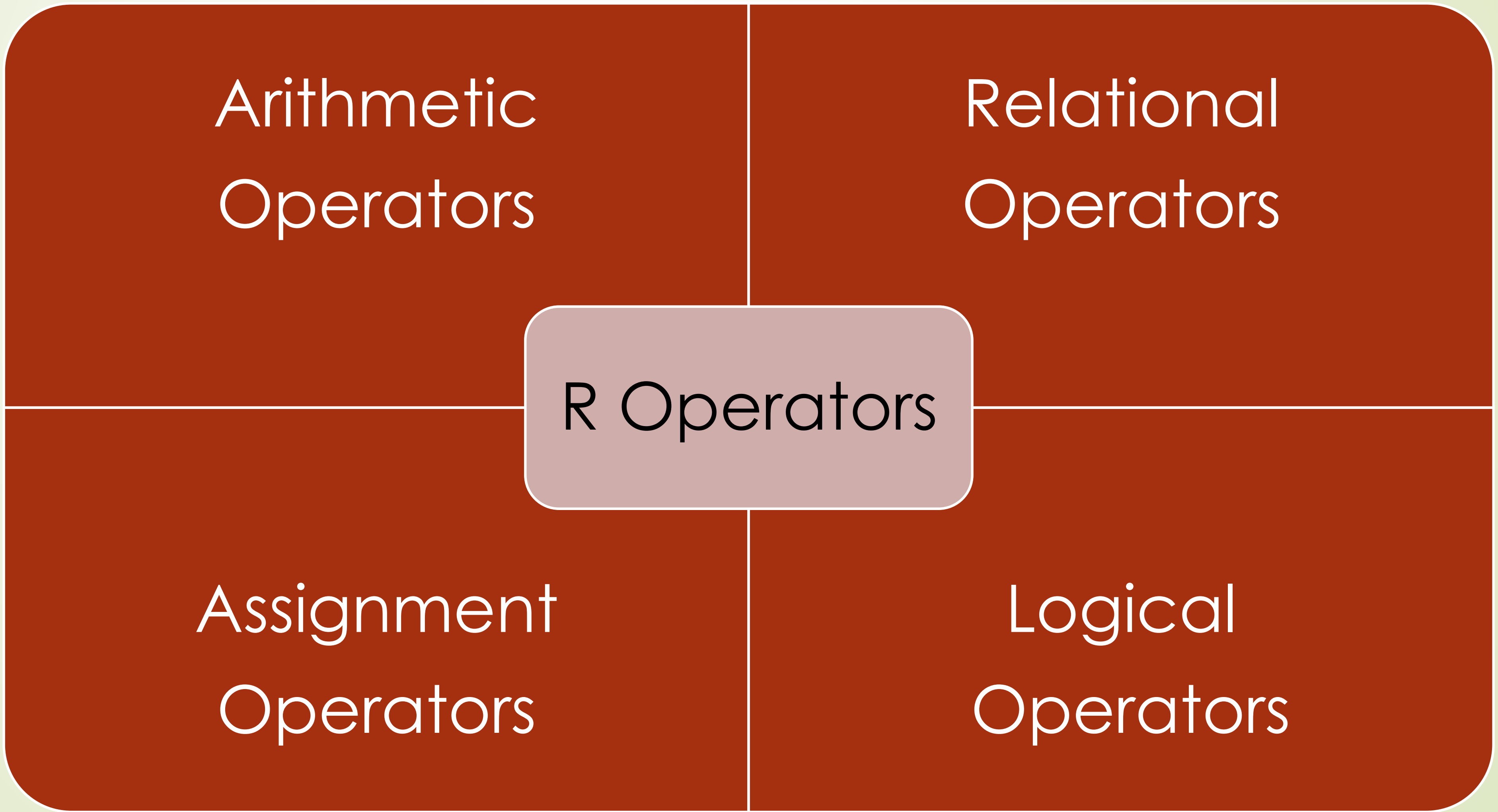
Data Type	Description	Example
Logical	Boolean Variables	TRUE, FALSE
Numeric	Number of all kinds	6, 18.5, 825
Integer	Explicit integers	8L, 45L
Complex	A + B i	7 + 5 i
Character	Characters and Strings	‘r’, “RKS”
Raw	Any data is stored as raw bytes	“Hello” is stored as 68 65 6c 6c 6f

Variables in R

- Variables are logical names for pieces of data. They are used to store data, and their value can be changed.
- The unique name given to a variable is called an identifier.
- Rules for defining variable names are as follows:
  - Variable names can consist of a combination of letters, numbers, underscore, and period.
  - It should begin with a letter or a period. If it starts with a period, then it cannot be followed by a digit.
  - Reserved words in R cannot be used as identifiers.

Operators in R

R supports these operators



Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus
%/%	Integer Division

Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

Assignment Operator: age <- 20
20 -> age (This also works)

Conditional Statements in R

```
age <- 30
if (age > 18) { print ( “Major”) }
else
{ print (“Minor”)
```

```
age <-30
ifelse(age>=18, “Major”, ”Minor”)
```

Conditional Statements in R

```
age <- "Major"
switch(age,
Major = { print("Age is
greater than 18") },
Minor = { print("Age is
less than 18") }
)
```

```
a <- 1; b <- -2; c <- 1
d <- b*b-4*a*c
if(d>=0) {x<-"1"} else {x
<- "2"}
switch(x,
"1" = {print("Roots are
Real")},
"2" = {print("Roots are
Complex")}     )
```

Loop Statements in R

A loop is a control flow statement that executes a statement or a group of statements many times.

Loops in R

For Loop

```
vec<-c(1,2,3,4,5)
for (val in vec)
{
print(val)
}
```

While Loop

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

Repeat Loop

```
x <-1
repeat
{
print(x)
x = x+1
if (x == 6){
break
} }
```

Control Statements in R: break and next

```
num <-1:5
for (val in num)
{
if (val == 3)
{
break
}
print(val)
}
```

```
num <-1:5
for (val in num)
{
if (val == 3)
{
next
}
print(val)
}
```

R Script

R script is a text file that contains a set of commands to be executed in R.

- The commands are executed in the console.
- Script files allow us to execute long and multiple commands easily at once.
- The file is saved with an extension ".R"

Running an R Script

```
source("myScript.R")
```

- The “source()” function instructs R to read the text file and execute its contents.
- It executes the script from the R console.

```
Rscript myScript.R
```

- “Rscript()” command can be invoked to run an R script from the command line.

R Functions

A function is a code used to execute a specific task

- Functions are stored as R objects.
- There are over 1000 functions at the core of R, and new R functions are created all the time.
- Each R function comes with its own help page. To access a function's help page, type a question mark followed by the function's name in the console.

R Functions: Examples

Function	Description
append()	Add elements to a vector
c()	Combine values into a vector or a list
identical()	Test if 2 objects are exactly equal
ls()	List objects in the current environment
range()	Returns the minimum and maximum of a vector
rep(x,n)	Repeat the number x n times
length()	Return the length of R object
rev(x)	Provide the reverse version of an argument
unique(x)	Remove duplicate entries from the vector
seq(x, y, n)	Generate regular sequences from x to y with spacing of n

R Functions: Examples

Function	Description
tolower()	Convert a string to lower case letters
toupper()	Convert a string to upper case letters
grep()	Use for regular expressions
summary()	Returns object summary
str()	Compactly displays the structure of an R object
class()	Returns the class of an object
mode()	Get or set the type or object mode of an object
…	…

Data Structures in R

Data Structure	Type	Dimensionality
Atomic vectors	Homogeneous	1
List	Heterogeneous	1
Matrix	Homogeneous	2
Array	Homogeneous	N
Factor	Homogeneous	1
Data Frame	Heterogeneous	2

Atomic Vectors in R

An atomic vector is a one-dimensional object and is the simplest data structure.

- It is called an atomic vector as all elements in it are of the same type.
- The data types in atomic vectors:
    - Numeric Data Type
    - Integer Data Type
    - Character Data Type
    - Logical Data Type

Examples:

- a <-c(1, 2, 5, 3, 6, -2, 4)
- b <-c("one", "two", "three")
- c <-c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)

Atomic Vectors in R

The : operator

- x <- 1:5
- y <- 1:-1

- vec <-c(“a”, “b”, “c”, “d”, “e”, ”f”)
- vec[1] # will return the first element in the vector
- vec[c(2,4)] # will return the second and fourth elements in the vector

Matrices in R

- vector <-c(1,2,3,4)
- m1 <-matrix(vector, nrow=2, ncol=2)
  - Elements in a matrix must be of the same type, whether a number, character, or Boolean.

- matrix(1:9, nrow= 3, ncol= 3, byrow=TRUE) #Matrix is filled column-wise by default

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

- a <- x[c(1,2),c(2,3)]   # select rows 1 & 2 and columns 2 & 3, this is how we can extract the sub-matrix

Array in R

Arrays are similar to a matrix but can have more than two dimensions

- A <-array(1: 24, dim= c(3, 4, 2))

- Example

```
vector1 <-c(1, 2, 3)
vector2 <-c(22,33,44,55,66,77)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
result <-array(c(vector1,vector2),dim = c(3,3,2), dimnames =
list(row.names,column.names,matrix.names))
print(result)


print(result[2,,1])
# Prints the second row of the first matrix of the array
```

Factors in R

Factors take only a predefined, finite number of categorical values

- Factors are created using the factor() function.
- These are built using two attributes: class and levels.

Example

```
x <- factor(c("male", "female", "female", "male"))
x
[1] male   female female male
Levels: female male

x <- factor(c("male", "female", "female", "male"), levels = c("male",
"female"))

x[3]   #Accessing elements of factors is similar to accessing
elements of an atomic vector
```

Data Frames in R

- Data frames are the most commonly used data structures in R.
- A data frame is similar to a general matrix, but its columns can contain different modes of data, such as a number and character.

Example:

```
name <-c( "Ram" , "Shyam" , "Sita")
sex <-c("M", "M", "F")
age <-c(27,26,26)
df <-data.frame(name, sex, age)
```

Other way of defining:

```
df <- data.frame (
name <-c( "Ram" , "Shyam" , "Sita"),
sex <-c("M", "M", "F"),
age <-c(27,26,26),
StringsAsFactors = FALSE)
```

Data Frames in R

- The data can be accessed using column names

Example:

```
result <- data.frame (df$age,df$sex)
print(result)
```

Lists in R

- Lists are the most complex data structures.
- A list is a vector that has elements of different types.
- A list may contain a combination of vectors, matrices, data frames, and even other lists.

- List elements can be accessed by indexing.
- The vector can be an integer, character, or logical vector.

Example:

```
vec <- c(1,2,3,4)
mat <- matrix(vec,2,2)
List_data <- list(vec, mat)
print(List_data)
```

```
print(list_data[1])

print(list_data[2])
```

Importing and Exporting Data in R

- We can import data from four types of files in R: Excel, Minitab, Table, and CSV

Example:

```
library(gdata ) #load gdata package
help(read.xls) #documentation
mydata = read.xls("mydata.xls") #read from first sheet

CrimeData <- read.csv("D:/Crimedata.xls")


x <- runif (100)
write.csv(x, "d:/runif.csv")
```

Data Manipulation

Function apply(): This is applied on a matrix row or column and returns a vector, array, or list.

- Syntax : apply(x, margin, function)
- x is the matrix, margin =1 when function is to be applied to a row, and 2 if it is to be applied to a column.
- Function can be any function such as mean, sum, or average
  - m <-matrix(c(1,2,3,4),2,2)
  - apply(m,1,sum)
  - apply(m,2,sum)

Data Manipulation

Function lapply(): This takes a list as its argument and works by looping through each element in the list. The output of the function is a list.

- Syntax : lapply(list, function)
- x is the matrix, margin =1 when function is to be applied to a row, and 2 if it is to be applied to a column.
- Function can be any function such as mean, sum, or average

  - list <-list(a=c(1,1), b=c(2,2), c=c(3,3))
  - lapply(list,sum)
  - lapply(list,mean)

Data Manipulation

- Function sapply(): sapply() is similar to lapply(), except that it simplifies the result so that: If the result is a list and every element in the list is of size 1, then a vector is returned. If the result is a list and every element in the list is of the same size (>1), then a matrix is returned. Otherwise, the result is returned as a list itself.

  - list <-list(a = c(1,1), b=c(2,2), c=c(3,3))
  - X<-sapply(list, sum); class(X)
  - list <-list(a = c(1,2), b=c(1,2,3), c=c(1,2,3,4))
  - Y<-sapply(list, range); class(Y)

The dplyr package: select statement

- library(dplyr)
- select(CrimeData, ID)
- select(CrimeData, ID: Beat)

- select(iris, starts_with("Petal"))
- select(iris, ends_with("Width"))
- select(iris, contains("etal"))
- select(iris, matches(".t."))

- data()
- View(iris)
- View(mtcars)
- summary(iris)
- summary(iris)

The dplyr package: filter, mutate, and summarize statements

- filter(CrimeData, Arrest == TRUE)
- filter(CrimeData, ID < “1410022”)
  - We can use a valid logical expression here

- The mutate() function helps in adding a new variable to the existing dataset.
  - mutate(mtcars, my_custom_d = disp / 1.0237)

- The summarise() function summarizes multiple values to a single value in a dataset.
  - summarise(group_by(mtcars, cyl), mean(hp))
  - summarise(group_by(mtcars, cyl), m = mean(hp), sd= sd(hp))

Inductive and Deductive Logic

Inductive and Deductive Logic

- Reasoning in artificial intelligence has two important forms, Inductive reasoning, and Deductive reasoning.
  - Both reasoning forms have premises and conclusions.

- Inductive reasoning involves drawing conclusions from facts, using logic. We draw these kinds of conclusions all the time.
  - For example, if someone we know to have good literary taste recommends a book, we may assume that we will enjoy the book.
  - Induction can be strong or weak. If an inductive argument is strong, the truth of the premise would mean the conclusion is likely. If an inductive argument is weak, the logic connecting the premise and conclusion is incorrect.

Inductive and Deductive Logic …

- Deductive Reasoning
  - Deductive reasoning is the form of valid reasoning, to deduce new information or conclusion from known related facts and information.
- Inductive Reasoning
  - Inductive reasoning arrives at a conclusion by the process of generalization using specific facts or data.
  - A fundamental problem in ML is *generalization*.

Inductive and Deductive Logic …

- The main difference between inductive and deductive reasoning is that inductive reasoning aims at developing a theory while deductive reasoning aims at testing an existing theory.

- Induction is going from examples to a more general rule: Specifics to generality.

- Deduction is going from general rule to specific instances.

Inductive and Deductive Logic …

- AI initially was deductive reasoning (say, logic programming wherein we have certain rules and we deduce only those things that immediately follow these rules).
  - A → B; If we have A, it means we have B.
  - Assumptions → Conclusion, the conclusion follows with certainty from the premises.
  - Sunny days are warmer; Day is sunny; Day is warmer.
  - All humans are mortal; Krishna was human; Krishna was mortal.

Inductive and Deductive Logic …

- All Machine Learning problems and certainly the supervised learning problems are induction, as opposed to deduction.
  - Supervised learning is nothing but function approximation, we may say that supervised learning is approximate function induction?
  - The conclusion follows not with certainty, but only with some probability (A risky inference).
  - 90% of human are right-handed; John is human; So, John is right-handed (Logically strong inference).
  - 1% of emails are spams; The mail received just now is not a spam.
  - 50% of human are female; The newly born baby is female (Logically weak inference)
- Problem of Induction logic: How strong is the inference?
  - How high does the probability have to be before it's rational to accept the conclusion? This leads to the concept of "Threshold Value" in ML.

Thank You !