BHARATI VIDYAPEETH DEEMED UNIVERSITY
COLLEGE OF ENGINEERING, PUNE - 43

DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual

Programming Lab 5

B.Tech. Computer Sem VII

Name of Faculty: Sheetal S.Patil.

DEPARTMENT OF COMPUTER ENGINEERING

VISION OF THE INSTITUTE

"To be World Class Institute for Social Transformation through Dynamic Education"

MISSION OF THE INSTITUTE

- To provide quality technical education with advanced equipment, qualified faculty members, infrastructure to meet needs of profession and society.
- To provide an environment conductive to innovation, creativity, research and entrepreneurial leadership.
- To practice and promote professional ethics, transparency and accountability for social community, economic and environmental conditions.

VISION OF THE DEPARTMENT

"To pursue and excel in the endeavour for creating globally recognized computer engineers through quality education".

MISSION OF THE DEPARTMENT

1. To impart fundamental knowledge and strong skills conforming to a dynamic curriculum leading to developing professional competencies and continuing intellectual growth.

2. To develop professional, entrepreneurial and research competencies encompassing continuing intellectual growth.

3. To strive for producing qualified graduates possessing proficient abilities and ethical responsibilities adapting to the demand of working environment.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates upon completion of B. Tech Computer Engineering Programme will able to**:**

1. Exhibit Competence and professional skills pertinent to the working environment.
2. Continue professional Development through available resources and avenues for career growth.
3. Act with ethical and societal awareness as expected from competent practicing professionals**.**

PROGRAM OUTCOMES

1. To apply knowledge of computing and mathematics appropriate to the domain.
2. To logically define, analyse and solve real world problems.
3. To apply design principles in developing hardware/software systems of varying complexity that meet the specified needs.
4. To interpret and analyse data for providing solutions to complex engineering problems.
5. To use and practise engineering and IT tools for professional growth.
6. To understand and respond to legal and ethical issues involving the use of technology for societal benefits.
7. To develop societal relevant projects using available resources.
8. To exhibit professional and ethical responsibilities.
9. To work effectively as an individual and a team member within the professional environment.
10. To prepare and present technical documents using effective communication skills.
11. To demonstrate effective leadership skills throughout the project management life cycle.
12. To understand the significance of lifelong learning for professional development.

GENRAL INSTUCTIONS:

- Equipment in the lab is meant for the use of students. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care.
- Students are required to carry their reference materials, files and records with completed assignment while entering the lab.
- Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.
- All the students should perform the given assignment individually.
- Lab can be used in free time/lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- All the Students are instructed to carry their identity cards when entering the lab.
- Lab files need to be submitted on or before date of submission.
- Students are not supposed to use pen drives, compact drives or any other storage devices in the lab.
- For Laboratory related updates and assignments students should refer to the notice board in the Lab.

COURSE NAME: Programming lab V

WEEKLY PLAN:

Week No.	Practical/Assignment Name	Problem Definition
1	Basics Of R	Introduction to R programming: How to Download, Install and set up R and R Studio.
2	Data structures in R	Demonstrate the concept of variables and data types in R and Create Vectors, Matrices, Lists, Arrays, using it.
3	Data import in various file formats & formatting	Import data, copy data from Excel to R, consider .csv, & .txt formats and performs Subset, Cbind, Rbind commands on it.
4	Data visualization	Write R script to make Bar chart(),Pie chart ()and Box Plot(titanic data set) on given datasets.
5	Data visualization	Produce stratified Boxplots, Histograms And Scatterplots using different datasets.
6	Binomial Distribution	Demonstrate & calculate Binomial Distribution, Poisson Distribution and Normal Distribution for given values.
7	Linear regression model	Write R script to build Linear Regression Model using given dataset.
8	Apriory Analysis	Perform Apriori Analysis using arules package.
9	K-means algorithm	Case Study: Implement K-Means Algorithm on worlddata dataset and Visualize the clusters.
10	Mini Project	Mini Project (group of 2 students) Create an Interactive Dashboard using shiny package.

EXAMINATION SCHEME

Practical Exam: 25 Marks
Term Work: 25 Marks
Total: 50 Marks
Minimum Marks required: 20 Marks

PROCEDURE OF EVALUATION

Each practical/assignment shall be assessed continuously on the scale of 25 marks. The distribution of marks as follows.

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	**Timely Submission**	**07**	➢ Punctuality reflects the work ethics. Students should reflect that work ethics by completing the lab assignments and reports in a timely manner without being reminded or warned.
2	**Presentation**	**06**	➢ Student are expected to write the technical document (lab report) in their own words. The presentation of the contents in the lab report should be complete, unambiguous, clear, understandable. The report should document approach/algorithm/design and code with proper explanation.
3	**Understanding**	**12**	➢ Correctness and Robustness of the code is expected. The Learners should have an in-depth knowledge of the practical assignment performed. The learner should be able to explain methodology used for designing and developing the program/solution. He/she should clearly understand the purpose of the assignment and its outcome.

LABORATORY USAGE

Students use R studio & R interface for executing the lab experiments, document the results and to prepare the technical documents for making the lab reports.

OBJECTIVES:-

- To solve the practical issues in statistical computing which includes programming in R, reading data into R, accessing R packages, writing R functions, debugging, profiling R code, and organizing and commenting R code.

PRACTICAL PRE-REQUISITE:-

1. Basics of Statistics
2. Data Analytics Algorithms

HARDWARE/ SOFTWARE REQUIREMENTS:-

1. R studio and R interface.

COURSE OUTCOMES

1. Recite R language fundamentals and basic syntax
2. Demonstrate how R is used to perform data analysis
3. Recite major R data structures
4. Illustrate visualizations using R
5. Implement various functions using R studio.
6. Design different Statistical models.

HOW OUTCOMES ARE ASSESSED?

Outcome	Assignment Number	Level	Proficiency evaluated by
Recite R language fundamentals and basic syntax.	**2 to 10**	**3**	Performing Practical and reporting results
Demonstrate how R is used to perform data analysis.	**1,2,4,6,7**	**3,3,2,2,2**	Problem definition &Performing Practical and reporting results
Recite major R data structures	**6-10**	**3**	Performing experiments and reporting results
Illustrate visualizations using R	**4,5,9,10**	**3,3,3,3**	Performing experiments and reporting results
Implement various functions using R studio.	**2 to 10**	**2**	Performing experiments and reporting results
Design different Statistical models.	**7,8**	**3,3**	Performing experiments and reporting results

CONTRIBUTION TO PROGRAM OUTCOMES

Program outcomes		1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2
Programming lab VCourse outcomes	**1**	3	2	2	2	2		1	2	3	3		2	3	
	2	3	3	2	3	2		1	3	3	3		3		3
	3	3	3	1	1	2		1	3	3	3		3	3	
	4	3	3	3	3	2		2	3	3	3		3		3
	5	3	2	3		2		2	3	3	3		2	3	
	6	3	3	3	3	2		3	3	2	2		2		2

DESIGN EXPERIENCE GAINED

The students learn algorithms of data analytics and using R convert that algorithm to executable code.

LEARNING EXPERIENCE GAINED

The students learn both soft skills and technical skills while they are undergoing the practical sessions. The soft skills gained in terms of communication, presentation and behavior. While technical skills they gained in terms of programming and efficient algorithm design using concepts of data analytics.

LIST OF PRACTICAL ASSIGNMENTS:

No	Detail of Assignment
1.	Introduction to R programming: How to Download, Install and set up R and R Studio.
2.	Demonstrate the concept of variables and data types in R and Create Vectors, Matrices, Lists, Arrays, using it.
3.	Import data, copy data from Excel to R, consider .csv, & .txt formats and performs Subset, Cbind, Rbind commands on it.
4.	Write R script to make Bar chart(),Pie chart ()and Box Plot(titanic data set) on given datasets.
5.	Produce stratified Boxplots, Histograms And Scatterplots using different datasets.
6.	Demonstrate & calculate Binomial Distribution, Poisson Distribution and Normal Distribution for given values.
7.	Write R script to build Linear Regression Model using given dataset.
8.	Perform Apriori Analysis using arules package.
9.	Case Study: Implement K-Means Algorithm on worlddata dataset and Visualize the clusters.
10.	Mini Project (group of 2 students) Create an Interactive Dashboard using shiny package.

EXPERIMENT NO: 1

Aim: Introduction to R programming: How to Download, Install and set up R and R Studio.

Objective : To download,install and set up R and R Studio.

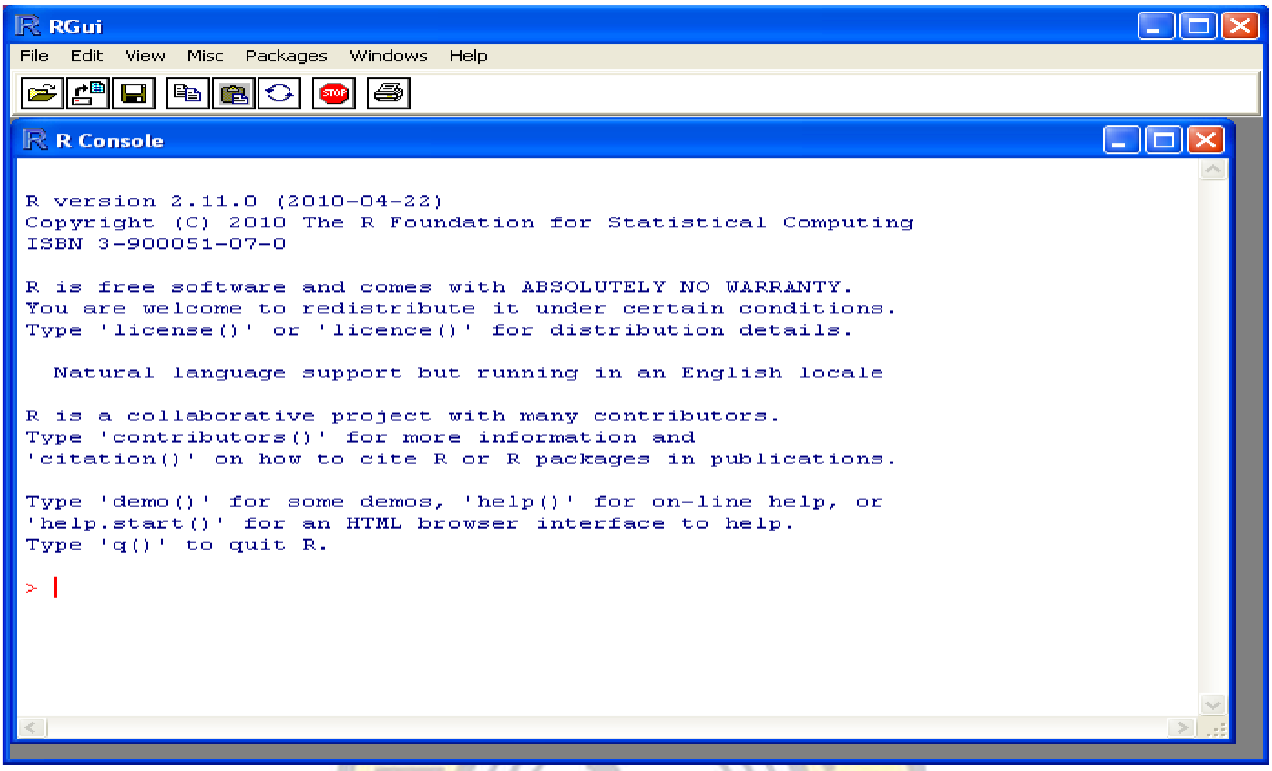
Theory:

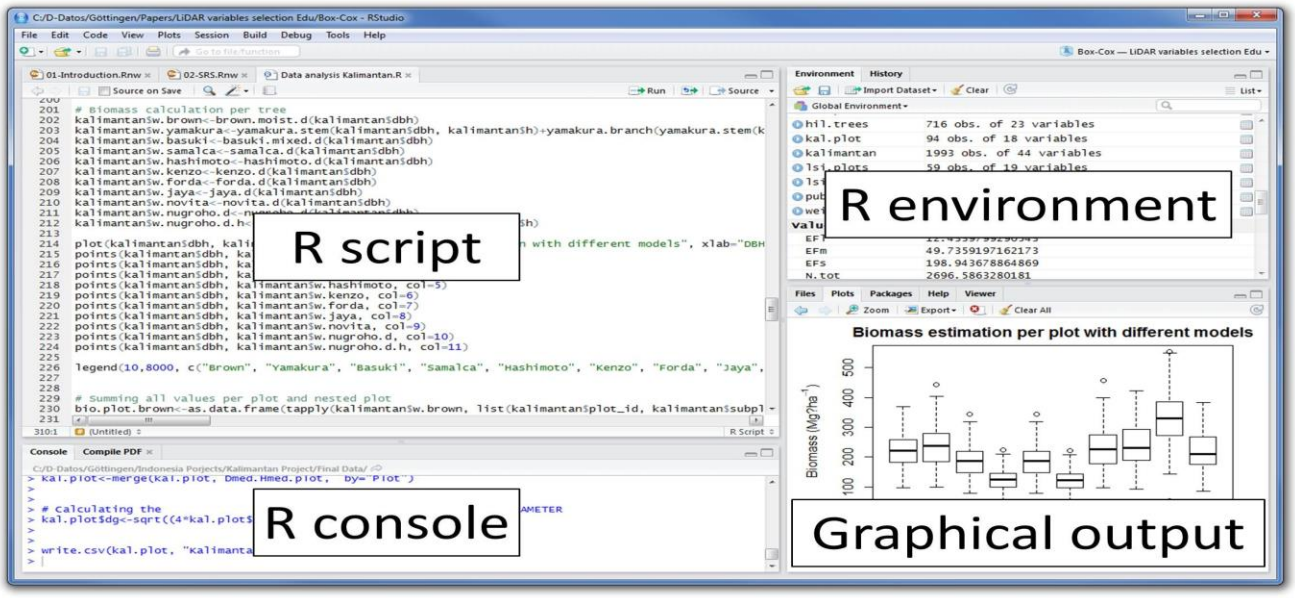
Installing R on a Windows PC

To install R on your Windows computer, follow these steps:

1. Go to http://ftp.heanet.ie/mirrors/cran.r-project.org.
2. Under "Download and Install R", click on the "Windows" link.
3. Under "Subdirectories", click on the "base" link.
4. On the next page, you should see a link saying something like "Download R 2.10.1 for Windows" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.11.1). Click on this link.
5. You may be asked if you want to save or run a file "R-2.10.1-win32.exe". Choose "Save" and save the file on the Desktop. Then double-click on the icon for the file to run it.
6. You will be asked what language to install it in - choose English.
7. The R Setup Wizard will appear in a window. Click "Next" at the bottom of the R Setup wizard window.
8. The next page says "Information" at the top. Click "Next" again.
9. The next page says "Information" at the top. Click "Next" again.
10. The next page says "Select Destination Location" at the top. By default, it will suggest to install R in "C:\Program Files" on your computer.
11. Click "Next" at the bottom of the R Setup wizard window.
12. The next page says "Select components" at the top. Click "Next" again.
13. The next page says "Startup options" at the top. Click "Next" again.
14. The next page says "Select start menu folder" at the top. Click "Next" again.
15. The next page says "Select additional tasks" at the top. Click "Next" again.
16. R should now be installed. This will take about a minute. When R has finished, you will see "Completing the R for Windows Setup Wizard" appear. Click "Finish".
17. To start R, you can either follow step 18, or 19:
18. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 19 instead.
19. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.

20. The R console (a rectangle) should pop up:





EXPERIMENT NO: 2

Aim: Demonstrate the concept of variables and data types in R and Create Vectors, Matrices, Lists, Arrays, using it.

Objective: To implement the concept of variables and data types in R and Create Vectors, Matrices, Lists, Arrays, using R.

Theory:

R - Variables

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Variable Assignment

The variables can be assigned values using leftward, rightward and equal to operator. The values can be printed using print() or cat() function. The **cat()** function combines multiple items into a continuous print output.

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)
# Assignment using leftward operator.
var.2 <- c("learn","R")
# Assignment using rightward operator.
c(TRUE,1) -> var.3

print(var.1)
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

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

```
[1] 0 1 2 3
var.1 is 0 1 2 3
var.2 is learn R
var.3 is 1 1
```

Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language, which means that we can change a variable's data type of the same variable again and again when using it in a program.

```
var_x <- "Hello"

cat("The class of var_x is ",class(var_x),"\n")var_x <- 34.5

cat("  Now the class of var_x is ",class(var_x),"\n")

var_x <- 27L

cat("  Next the class of var_x becomes ",class(var_x),"\n")
```

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

```
The class of var_x is  character
  Now the class of var_x is  numeric
    Next the class of var_x becomes  integer
```

Vector Creation

Single Element Vector

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

```
# Atomic vector of type character.

print("abc");

# Atomic vector of type double.

print(12.5)

# Atomic vector of type integer.

print(63L)

# Atomic vector of type logical.

print(TRUE)

# Atomic vector of type complex.
```

```
print(2+3i)

# Atomic vector of type raw.

print(charToRaw('hello'))
```

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

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

Multiple Elements Vector

Using colon operator with numeric data

```
# Creating a sequence from 5 to 13.

v <- 5:13

print(v)

# Creating a sequence from 6.6 to 12.6.

v <- 6.6:12.6

print(v)

# If the final element specified does not belong to the sequence then it is discarded.

v <- 3.8:11.4

print(v)
```

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

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

Using sequence (Seq.) operator

```
# Create vector with elements from 5 to 9 incrementing by 0.4.

print(seq(5, 9, by = 0.4))
```

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

```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Using the c() function

The non-character values are coerced to character type if one of the elements is a character.

```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5,TRUE)
print(s)
```

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

```
[1] "apple" "red"   "5"     "TRUE"
```

R – Matrices

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

A Matrix is created using the **matrix()** function.

Syntax

The basic syntax for creating a matrix in R is −

```
matrix(data, nrow, ncol, byrow, dimnames)
```

Following is the description of the parameters used −

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

Example

Create a matrix taking a vector of numbers as input

```
# Elements are arranged sequentially by row.
```

```
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)

print(M)

# Elements are arranged sequentially by column.

N <- matrix(c(3:14), nrow = 4, byrow = FALSE)

print(N)

# Define the column and row names.

rownames = c("row1", "row2", "row3", "row4")

colnames = c("col1", "col2", "col3")


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

print(P)
```

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

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
     col1 col2 col3
row1 3 4 5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

R – Arrays

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

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

Example

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

```
# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2))

print(result)
```

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

```
, , 1

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

, , 2

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

R – Lists

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

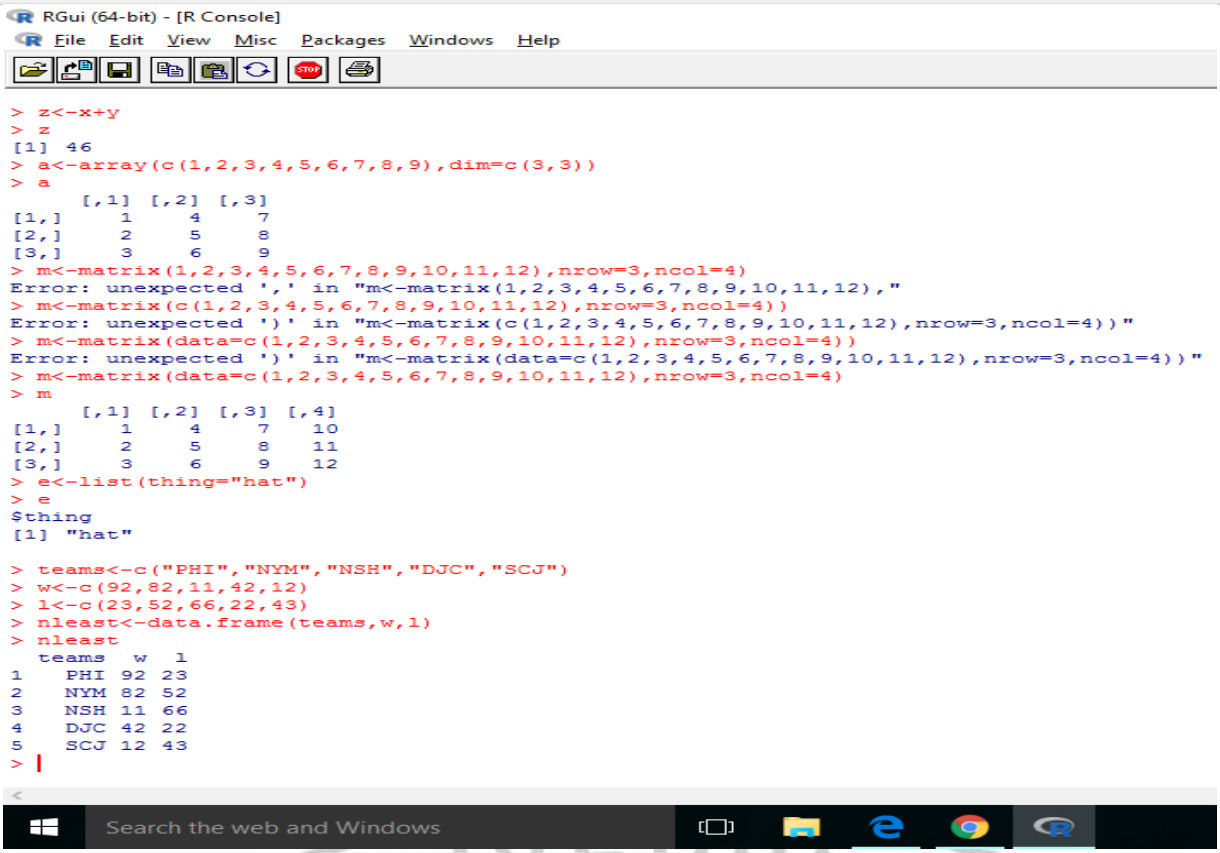
Creating a List

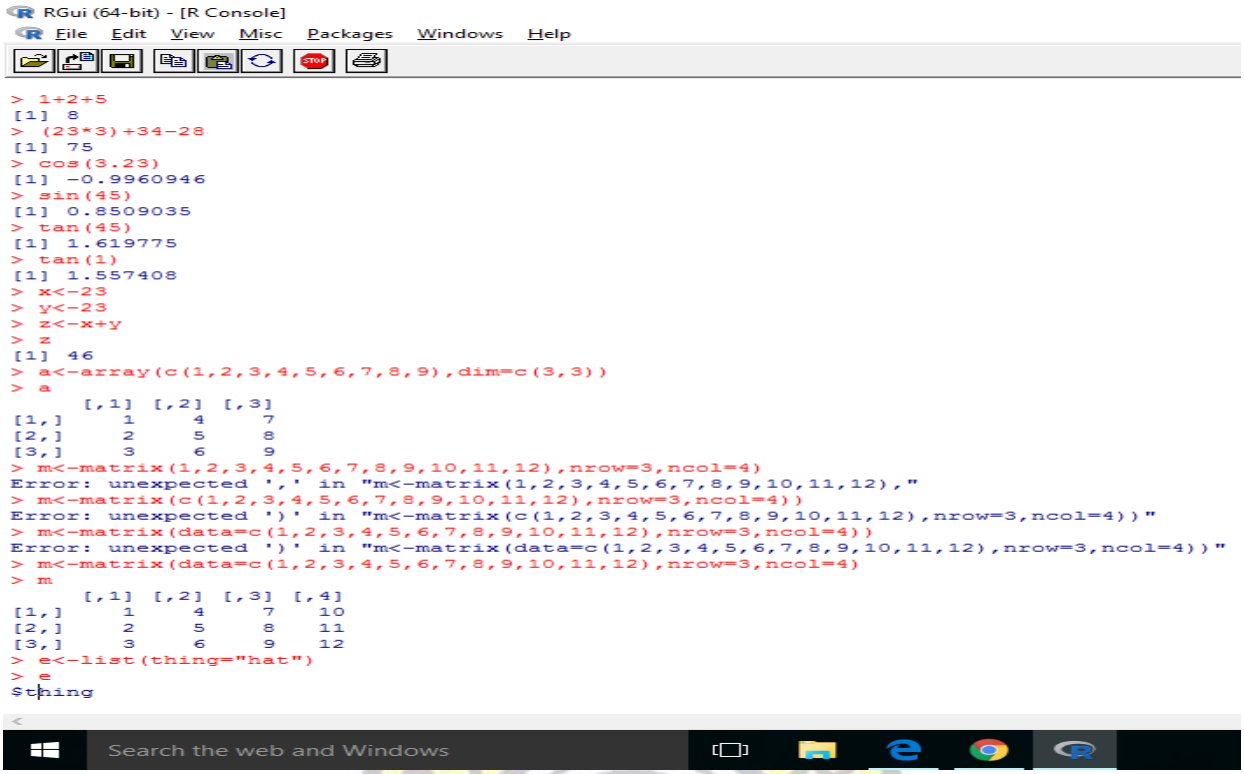
Following is an example to create a list containing strings, numbers, vectors and a logical values

```
# Create a list containing strings, numbers, vectors and a logical values.

list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)

print(list_data)
```

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

```
[[1]]
[1] "Red"
[[2]]
[1] "Green"
[[3]]
[1] 21 32 11
[[4]]
[1] TRUE
[[5]]
[1] 51.23
```





Questions:

1. **How will you check if an element 25 is present in a vector? Write R command for it.**
2. **Implement string operations in R language using package stringr.**
3. **Write R script to Find out Transpose of the matrix.**
4. **Sort the data using R. 60, 90, 30, 82,10,29,100.**
5. **Write R command that is used to store R objects in a file?**

EXPERIMENT NO: 3

Aim: Import data, copy data from Excel to R, consider .csv, & .txt formats and performs Subset, Cbind, Rbind commands on it.

Objective: To implement import data, copy data from Excel and perform subsets, Cbind, Rbind

Theory:

Different set of commands in R

1.read.table:-

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),

    row.names, col.names, as.is = !stringsAsFactors,
    na.strings = "NA", colClasses = NA, nrows = -1,
    skip = 0, check.names = TRUE, fill = !blank.lines.skip,
    strip.white = FALSE, blank.lines.skip = TRUE,
    comment.char = "#",
    allowEscapes = FALSE, flush = FALSE,
    stringsAsFactors = default.stringsAsFactors(),
    fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Arguments

file The name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, getwd(). Tilde-expansion is performed where supported. This can be a compressed file (see file).

Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback

on stdin()will be cleared before return.)

file can also be a complete URL. (For the supported URL schemes, see the 'URLs' section of the help for url.)

header a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.

sep the field separator character. Values on each line of the file are separated by this character. If sep = "" (the default for read.table) the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.

quote the set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless colClasses is specified.

2. head,tail :-

Description

Obtain the first several rows of a matrix or data frame using head, and use tail to obtain the last several rows. These functions may also be applied to obtain the first or last values in a vector.
head(x, n=6)

- x – A matrix, data frame, or vector.
- n – The first n rows (or values if x is a vector) will be returned.

tail(x, n=6)

- x – A matrix, data frame, or vector.
- n – The last n rows (or values if x is a vector) will be returned.

Example. Below are four examples of use of head and tail.

```
> library(MASS)
> data(Boston)
> head(Boston, 3)
    crim zn indus chas   nox   rm  age    dis rad tax
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242
  ptratio  black lstat medv
1    15.3 396.90  4.98 24.0
2    17.8 396.90  9.14 21.6
3    17.8 392.83  4.03 34.7
>
> tail(Boston)
      crim zn indus chas   nox   rm  age    dis rad tax
501 0.22438   0 9.69     0 0.585 6.027 79.7 2.4982   6 391
502 0.06263  0 11.93     0 0.573 6.593 69.1 2.4786   1 273
503 0.04527  0 11.93     0 0.573 6.120 76.7 2.2875   1 273
504 0.06076  0 11.93     0 0.573 6.976 91.0 2.1675   1 273
505 0.10959  0 11.93     0 0.573 6.794 89.3 2.3889   1 273
```

```
506 0.04741  0 11.93   0 0.573 6.030 80.8 2.5050    1 273
   ptratio  black lstat medv
501    19.2 396.90 14.33 16.8
502    21.0 391.99  9.67 22.4
503    21.0 396.90  9.08 20.6
504    21.0 396.90  5.64 23.9
505    21.0 393.45  6.48 22.0
506    21.0 396.90  7.88 11.9
>
> head(1:50, 10)
 [1]  1  2  3  4  5  6  7 8  9 10
>
> tail(1:50, 1)
[1] 50
```

3. numeric :-

Description

Numeric Vectors

Creates or coerces objects of type "numeric". is.numeric is a more general test of an object being interpretable as numbers.

Usage

```
 numeric(length = 0)
 as.numeric(x, …)
is.numeric(x)
```

Arguments

length

A non-negative integer specifying the desired length. Double values will be coerced to integer: supplying an argument of length other than one is an error.

x

object to be coerced or tested.

…

further arguments passed to or from other methods.

4. Cbind, Rbind :-

Description

Take a sequence of vector, matrix or data frames arguments and combine by *c*olumns or *r*ows, respectively. There may be methods for other **R** classes.

Usage

cbind(...)
rbind(...)

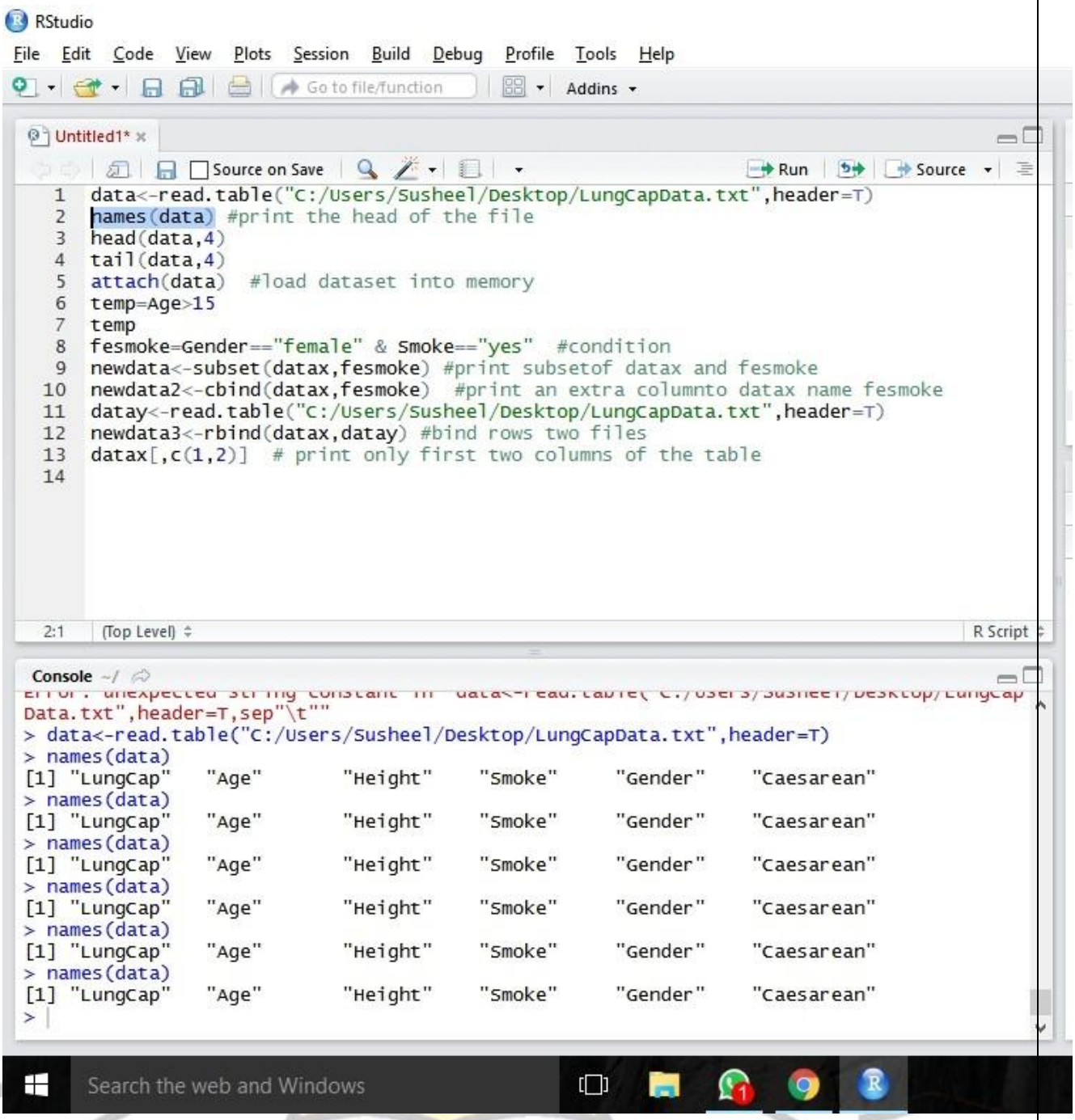
Details

The functions cbind and rbind are generic, with methods for data frames. The data frame method will be used if an argument is a data frame and the rest are vectors or matrices. There can be other methods, for example cbind.ts in package ts.

If there are several matrix arguments, they must all have the same number of columns (or rows) and this will be the number of columns (or rows) of the result. If all the arguments are vectors, the number of columns (rows) in the result is equal to the length of the longest vector. Values in shorter arguments are recycled to achieve this length (with a warning when they are recycled only *fractionally*).

Questions:

- Write R script for use of subset () and sample () functions in R?
- What is the function used for adding datasets in R? Add the dataset in any of the file.
- Find the covariance of eruption duration and waiting time in the data set faithful. Observe if there is any linear relationship between the two variables.
- Find the median of the eruption duration in the data set faithful.
- Find the quartiles of the eruption durations in the data set faithful.
- Find the 32^{nd}, 57^{th} and 98^{th} percentiles of the eruption durations in the data set faithful.
- Find the range of the eruption duration in the data set faithful.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1*
Source on Save
Run
Source
1 data<-read.table("C:/Users/Susheel/Desktop/LungCapData.txt",header=T)
2 names(data) #print the head of the file
3 head(data,4)
4 tail(data,4)
5 attach(data) #load dataset into memory
6 temp=Age>15
7 temp
8 fesmoke=Gender=="female" & Smoke=="yes" #condition
9 newdata<-subset(datax,fesmoke) #print subsetof datax and fesmoke
10 newdata2<-cbind(datax,fesmoke) #print an extra columnto datax name fesmoke
11 datay<-read.table("C:/Users/Susheel/Desktop/LungCapData.txt",header=T)
12 newdata3<-rbind(datax,datay) #bind rows two files
13 datax[,c(1,2)] # print only first two columns of the table
14
2:1 (Top Level) R Script
Console ~/
Data.txt",header=T,sep"\t""
> data<-read.table("C:/Users/Susheel/Desktop/LungCapData.txt",header=T)
> names(data)
[1] "LungCap" "Age" "Height" "Smoke" "Gender" "Caesarean"
> names(data)
[1] "LungCap" "Age" "Height" "Smoke" "Gender" "Caesarean"
> names(data)
[1] "LungCap" "Age" "Height" "Smoke" "Gender" "Caesarean"
> names(data)
[1] "LungCap" "Age" "Height" "Smoke" "Gender" "Caesarean"
> names(data)
[1] "LungCap" "Age" "Height" "Smoke" "Gender" "Caesarean"
> names(data)
[1] "LungCap" "Age" "Height" "Smoke" "Gender" "Caesarean"
>
Search the web and Windows

EXPERIMENT NO: 4

Aim: Write R script to make Bar chart (), Pie chart () and Box Plot (titanic data set) on given datasets.

Objective: Implement BarChart(), PieChart(), BoxPlot() on given datasets.

Theory:

Bar Charts

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is −

```
barplot(H, xlab, ylab, main, names.arg, col)
```

Following is the description of the parameters used −

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

Example

A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart.

H <- c(7,12,28,3,41)

# Give the chart file a name.
```

```
png(file = "barchart.png")

# Plot the bar chart.

barplot(H)

# Save the file.

dev.off()
```

Pie Charts

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is −

```
pie(x, labels, radius, main, col, clockwise)
```

Following is the description of the parameters used −

- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.

x <- c(21, 62, 10, 53)

labels <- c("London", "New York", "Singapore", "Mumbai")
```

```
# Give the chart file a name.

png(file = "city.jpg")

# Plot the chart.

pie(x,labels)

# Save the file.

dev.off()
```

Boxplots

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.

Syntax

The basic syntax to create a boxplot in R is −

```
boxplot(x, data, notch, varwidth, names, main)
```

Following is the description of the parameters used −

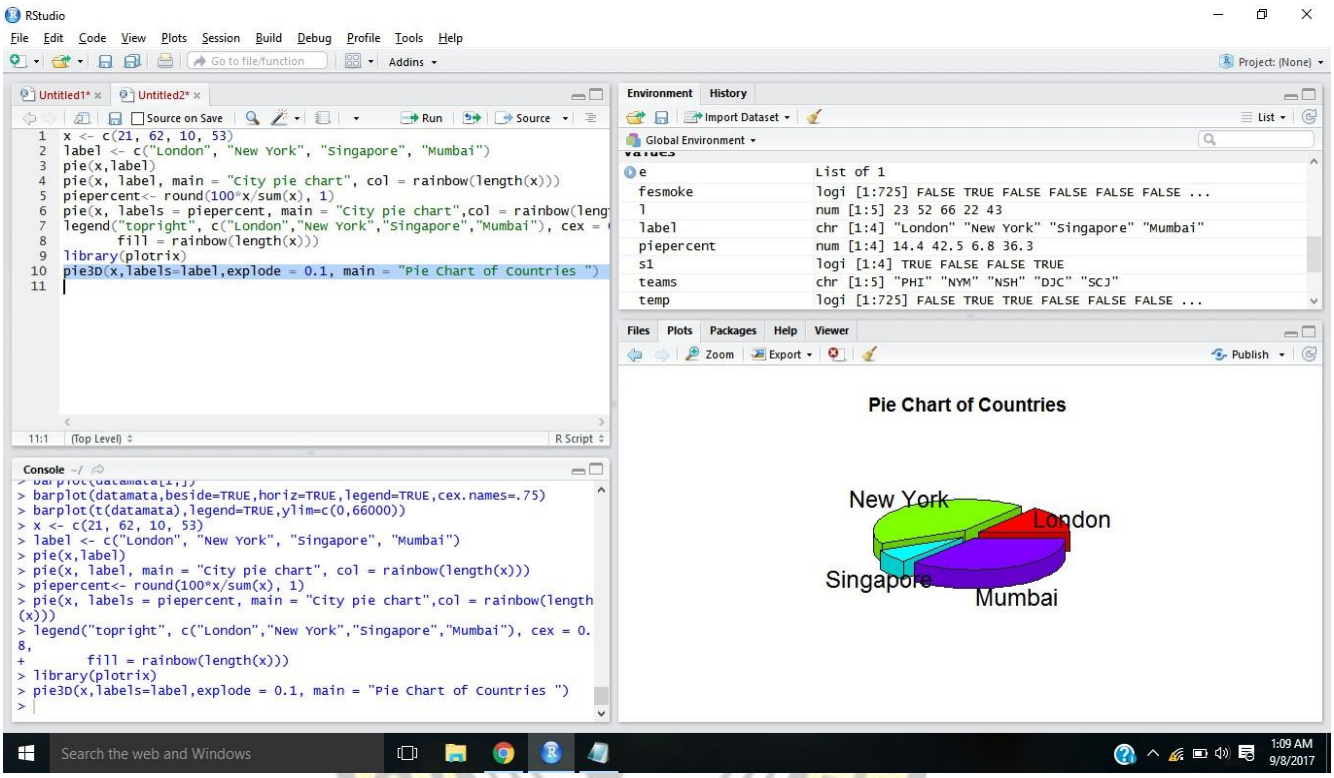
- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Example

The below script will create a boxplot graph for the relation between mpg (miles per gallon) and cyl (number of cylinders).

```
# Give the chart file a name.

png(file = "boxplot.png")
```

```
# Plot the chart.

boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",

  ylab = "Miles Per Gallon", main = "Mileage Data")
```



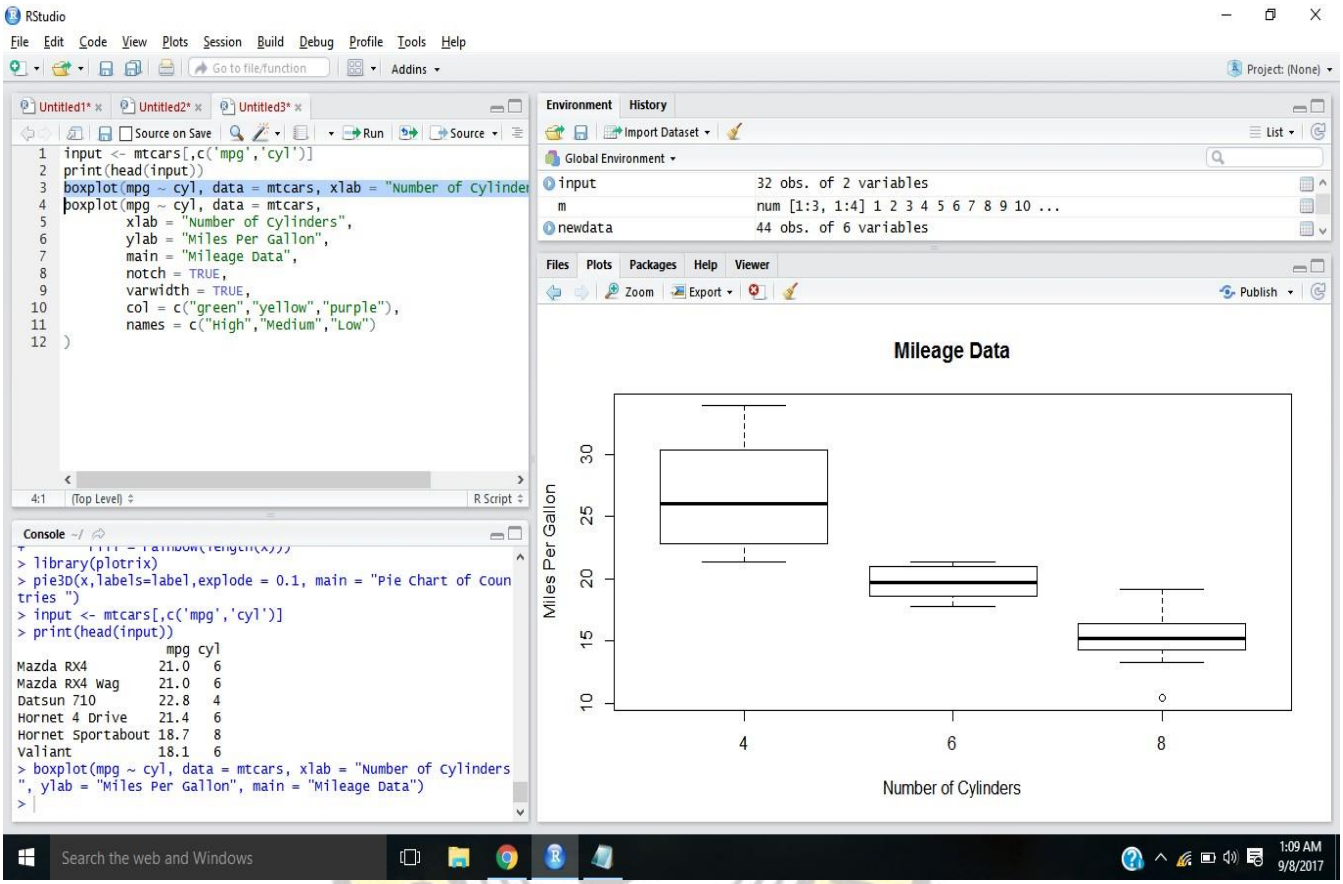
Q. Use the functions mean() and range()to find the mean and range of: (a) the numbers 1, 2, . . . , 21

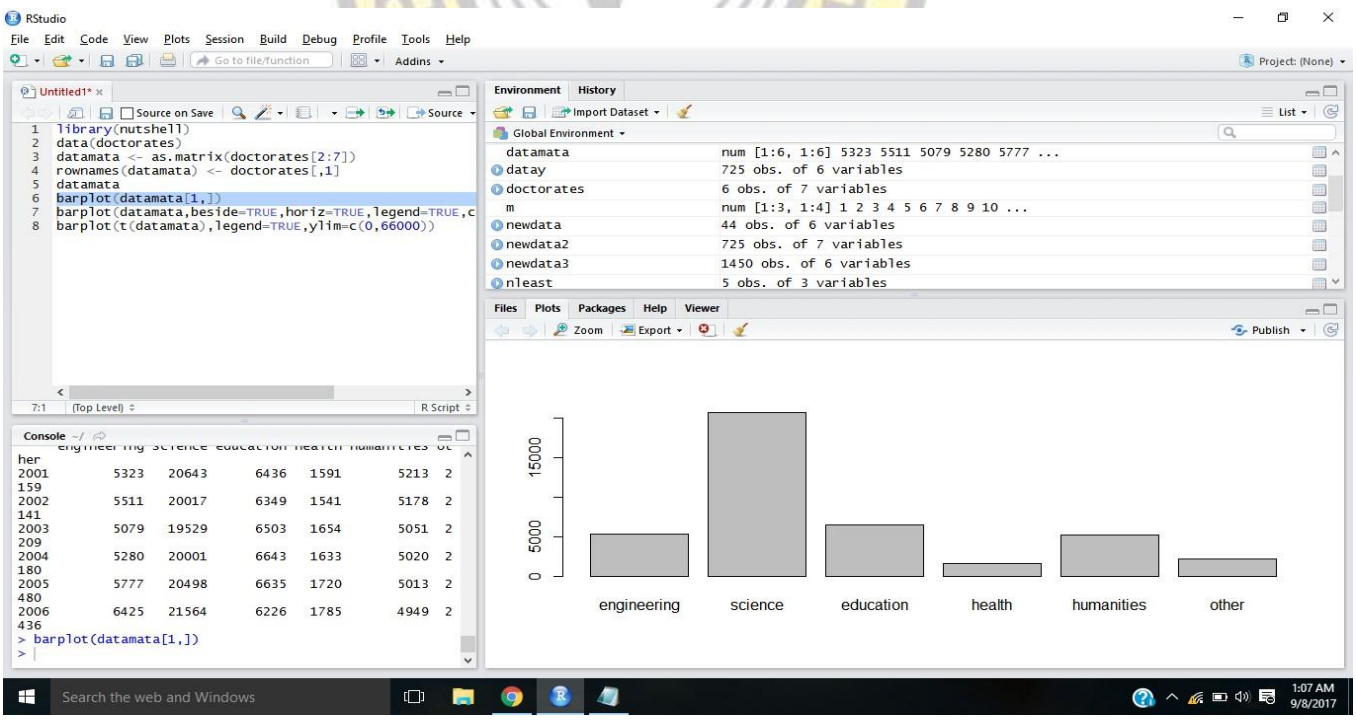
(b)the sample of 50 random normal values, that can be generated from a normal distribution with mean 0 and variance 1 using the assignment y <- rnorm(50).

Q. Use the functions mean() and range() to find the mean and range of: a)The sample of 50 random normal values, that can be generated from a normal distribution with mean 0 and variance 1 using the assignment y <- rnorm(50).

b)The columns height and weight in the data frame women [The datasets package that has this data frame is by default attached when R is started.]

Q. Use the functions median()and sum() to find the mean and range of: (a) the numbers 1, 2, . . . , 21 (b)the sample of 50 random normal values, that can be generated from a normal distribution with mean 0 and variance 1 using the assignment y <- r norm(50).


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1* Untitled2* Untitled3*
input <- mtcars[,c('mpg','cyl')]
print(head(input))
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinder
boxplot(mpg ~ cyl, data = mtcars,
xlab = "Number of Cylinders",
ylab = "Miles Per Gallon",
main = "Mileage Data",
notch = TRUE,
varwidth = TRUE,
col = c("green","yellow","purple"),
names = c("High","Medium","Low")
)
Environment History
Import Dataset
Global Environment
input 32 obs. of 2 variables
m num [1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
newdata 44 obs. of 6 variables
Files Plots Packages Help Viewer
Zoom Export Publish
Mileage Data
Miles Per Gallon
Number of Cylinders
Console
> library(plotrix)
> pie3D(x,labels=label,explode = 0.1, main = "Pie Chart of Countries ")
> input <- mtcars[,c('mpg','cyl')]
> print(head(input))
mpg cyl
Mazda RX4 21.0 6
Mazda RX4 Wag 21.0 6
Datsun 710 22.8 4
Hornet 4 Drive 21.4 6
Hornet Sportabout 18.7 8
Valiant 18.1 6
> boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders", ylab = "Miles Per Gallon", main = "Mileage Data")
Search the web and Windows
1:09 AM
9/8/2017


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1*
library(nutshell)
data(doctorates)
datamata <- as.matrix(doctorates[2:7])
rownames(datamata) <- doctorates[,1]
datamata
barplot(datamata[1,])
barplot(datamata,beside=TRUE,horiz=TRUE,legend=TRUE,c
barplot(t(datamata),legend=TRUE,ylim=c(0,66000))
Environment History
Import Dataset
Global Environment
datamata num [1:6, 1:6] 5323 5511 5079 5280 5777 ...
datay 725 obs. of 6 variables
doctorates 6 obs. of 7 variables
m num [1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
newdata 44 obs. of 6 variables
newdata2 725 obs. of 7 variables
newdata3 1450 obs. of 6 variables
nleast 5 obs. of 3 variables
Files Plots Packages Help Viewer
Zoom Export Publish
0 5000 15000
engineering science education health humanities other
Console
2001 5323 20643 6436 1591 5213 2159
2002 5511 20017 6349 1541 5178 2141
2003 5079 19529 6503 1654 5051 2209
2004 5280 20001 6643 1633 5020 2180
2005 5777 20498 6635 1720 5013 2480
2006 6425 21564 6226 1785 4949 2436
> barplot(datamata[1,])
Search the web and Windows
1:07 AM
9/8/2017

EXPERIMENT NO: 5

Aim: Produce stratified Boxplots, Histograms And Scatterplots using different datasets.

Objective: Implementing Boxplots, Histograms and Scatterplots.

Theory:

Histograms

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is −

```
hist(v,main,xlab,xlim,ylim,breaks,col,border)
```

Following is the description of the parameters used −

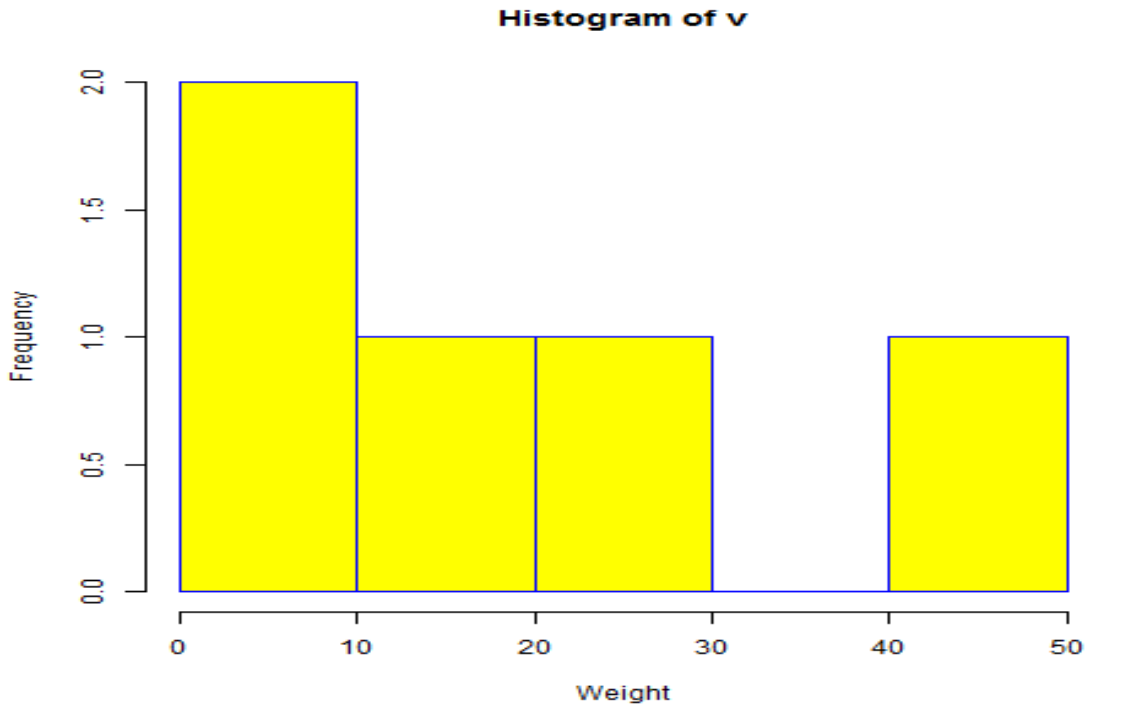
- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

Example

A simple histogram is created using input vector, label, col and border parameters. The script given below will create and save the histogram in the current R working directory.

```
# Create data for the graph.

v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.

png(file = "histogram.png")

# Create the histogram.

hist(v,xlab = "Weight",col = "yellow",border = "blue")

# Save the file.

dev.off()
```

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



Scatterplots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is −

```
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

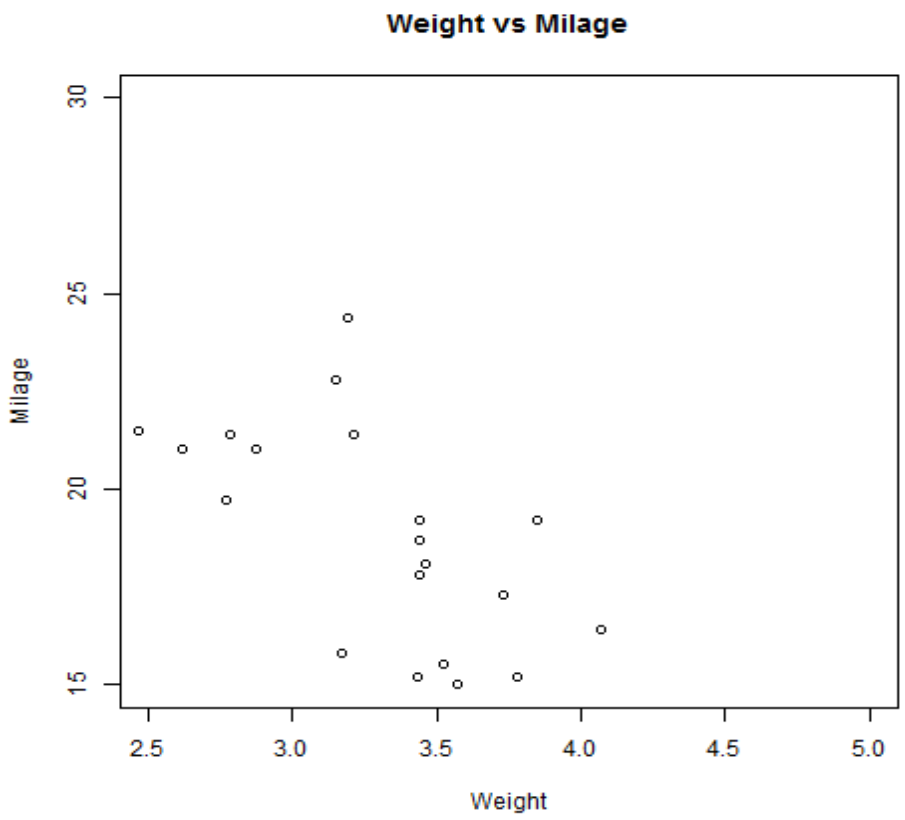
Example

The below script will create a scatterplot graph for the relation between wt(weight) and mpg(miles per gallon).

```
# Get the input values.
input <- mtcars[,c('wt','mpg')]
# Give the chart file a name.
png(file = "scatterplot.png")
# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
   xlab = "Weight",
   ylab = "Milage",
   xlim = c(2.5,5),
   ylim = c(15,30),
   main = "Weight vs Milage"
```

```
)

# Save the file.

dev.off()
```

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



Q.Extract the following subsets from the data frame ais (DAAG):

(a) Extract the data for the rowers.

(b)Extract the data for the rowers, the netballers and the tennis players.

Q. Extract the following subsets from the data frame ais (DAAG):

(b)Extract the data for the rowers, the netballers and the tennis players.

(c)Extract the data for the female basket ballers and rowers.

Q.Using the Acmena data from the data frame rainforest , plot wood (wood biomass) vs Dbh (diameter at breast height), trying both untransformed scales and logarithmic scales.

Q. Using the Acmena data from the data frame rainforest, Sort the rows in the data frame Acmena in order of increasing values of dbh.

EXPERIMENT NO: 6

Aim: Demonstrate & calculate Binomial Distribution, Poisson Distribution and Normal Distribution for given values.

Objective: Implementation of Binomial Distribution, Poisson Distribution and Normal Distribution.

Theory:

Binomial Distribution

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

R has four in-built functions to generate binomial distribution. They are described below.

```
dbinom(x, size, prob)
pbinom(x, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)
```

Following is the description of the parameters used −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations.
- **size** is the number of trials.
- **prob** is the probability of success of each trial.

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

R has four in-built functions to generate binomial distribution. They are described below.

```
dbinom(x, size, prob)
pbinom(x, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)
```

Following is the description of the parameters used −

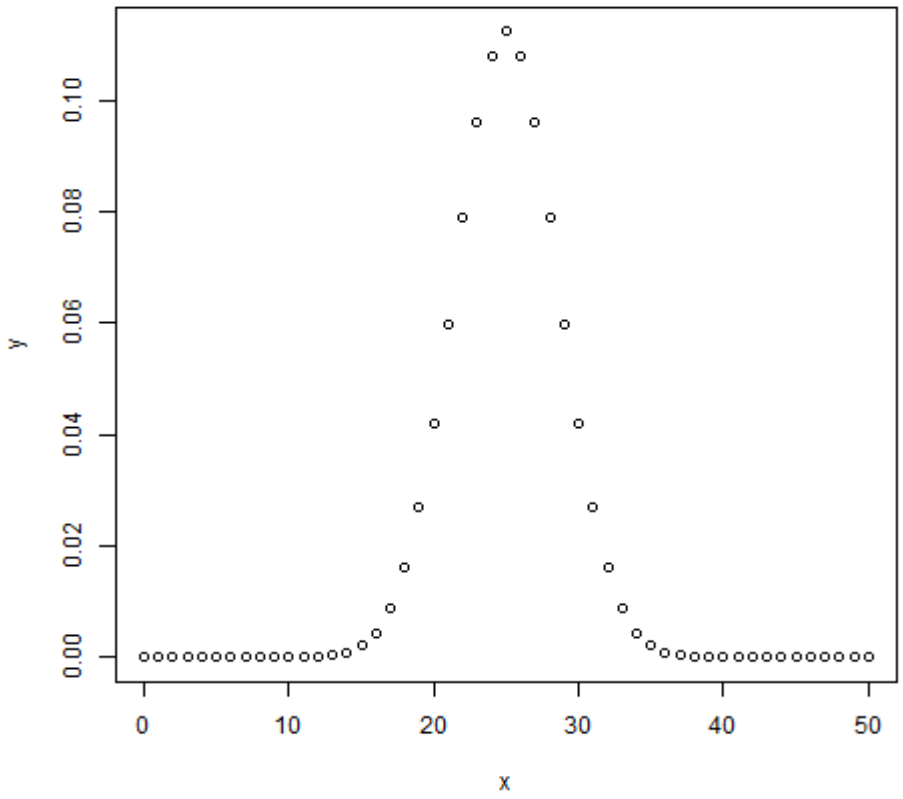
- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations.
- **size** is the number of trials.
- **prob** is the probability of success of each trial.

dbinom()

This function gives the probability density distribution at each point.

```
# Create a sample of 50 numbers which are incremented by 1.

x <- seq(0,50,by = 1)

# Create the binomial distribution.

y <- dbinom(x,50,0.5)

# Give the chart file a name.

png(file = "dbinom.png")

# Plot the graph for this sample.

plot(x,y)

# Save the file.

dev.off()
```

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



pbinom()

This function gives the cumulative probability of an event. It is a single value representing the probability.

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)
print(x)
```

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

```
[1] 0.610116
```

qbinom()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# How many heads will have a probability of 0.25 will come out when a coin is tossed 51 times.

x <- qbinom(0.25,51,1/2)

print(x)
```

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

```
[1] 23
```

rbinom()

This function generates required number of random values of given probability from a given sample.

```
# Find 8 random values from a sample of 150 with probability of 0.4.

x <- rbinom(8,150,.4)

print(x)
```

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

```
[1] 58 61 59 66 55 60 61 67
```

Poisson Regression

Poisson Regression involves regression models in which the response variable is in the form of counts and not fractional numbers. For example, the count of number of births or number of wins in a football match series. Also the values of the response variables follow a Poisson distribution.

The general mathematical equation for Poisson regression is −

```
log(y) = a + b1x1 + b2x2 + bnxn.....
```

Following is the description of the parameters used −

- **y** is the response variable.
- **a** and **b** are the numeric coefficients.
- **x** is the predictor variable.

The function used to create the Poisson regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in Poisson regression is −

```
glm(formula,data,family)
```

Following is the description of the parameters used in above functions −

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is 'Poisson' for Logistic Regression.

Example

We have the in-built data set "warpbreaks" which describes the effect of wool type (A or B) and tension (low, medium or high) on the number of warp breaks per loom. Let's consider "breaks" as the response variable which is a count of number of breaks. The wool "type" and "tension" are taken as predictor variables.

Input Data

```
input <- warpbreaks

print(head(input))
```

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

```
   breaks  wool  tension
1   26      A     L
2   30      A     L
3   54      A     L
4   25      A     L
5   70      A     L
6   52      A     L
```

Normal Distribution

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve represents the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph. This is referred as normal distribution in statistics.

R has four in built functions to generate normal distribution. They are described below.

```
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

Following is the description of the parameters used in above functions −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations(sample size).
- **mean** is the mean value of the sample data. It's default value is zero.
- **sd** is the standard deviation. It's default value is 1.

dnorm()

This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.
x <- seq(-10, 10, by = .1)
# Choose the mean as 2.5 and standard deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)
# Give the chart file a name.
png(file = "dnorm.png")
plot(x,y)
# Save the file.
dev.off()
```

pnorm()

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.
x <- seq(-10,10,by = .2)
 # Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)
```

```
# Give the chart file a name.
png(file = "pnorm.png")
# Plot the graph.
plot(x,y)
# Save the file.
dev.off()
```

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

qnorm()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

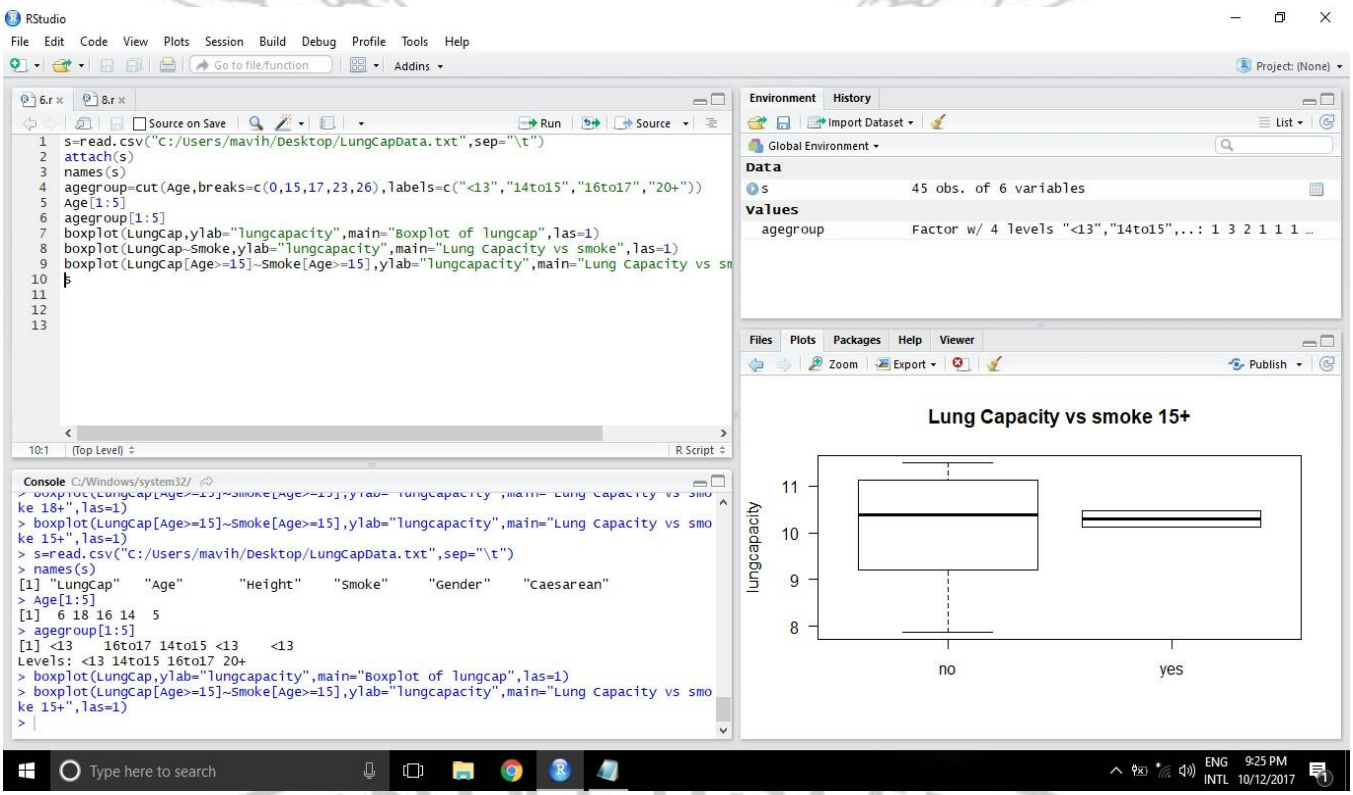
```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)
# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)
# Give the chart file a name.
png(file = "qnorm.png")
# Plot the graph.
plot(x,y)
# Save the file.
dev.off()
```

rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)
```

```
# Give the chart file a name.

png(file = "rnorm.png")

# Plot the histogram for this sample.

hist(y, main = "Normal Distribution")

# Save the file.

dev.off()
```



EXPERIMENT NO: 7

Aim: Write R script to build Linear Regression Model using given dataset.

Objective: To implement Linear Regression Model in R.

Theory:

Linear Regression

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is −

```
y = ax + b
```

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship are −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.

- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()**function in R.

Input Data

Below is the sample data representing the observations −

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is −

```
lm(formula,data)
```

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

print(relation)
```

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

```
Call:
lm(formula = y ~ x)
```

```
Coefficients:
(Intercept)          x
  -38.4551        0.6746
```

Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

print(summary(relation))
```

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

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q    Median      3Q     Max
-6.3002  -1.6629  0.0412   1.8944  3.9775

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509  8.04901  -4.778  0.00139 **
x             0.67461   0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,  Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict() Function

Syntax

The basic syntax for predict() in linear regression is −

```
predict(object, newdata)
```

Following is the description of the parameters used −

- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
```

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

# Find weight of a person with height 170.

a <- data.frame(x = 170)

result <-  predict(relation,a)

print(result)
```

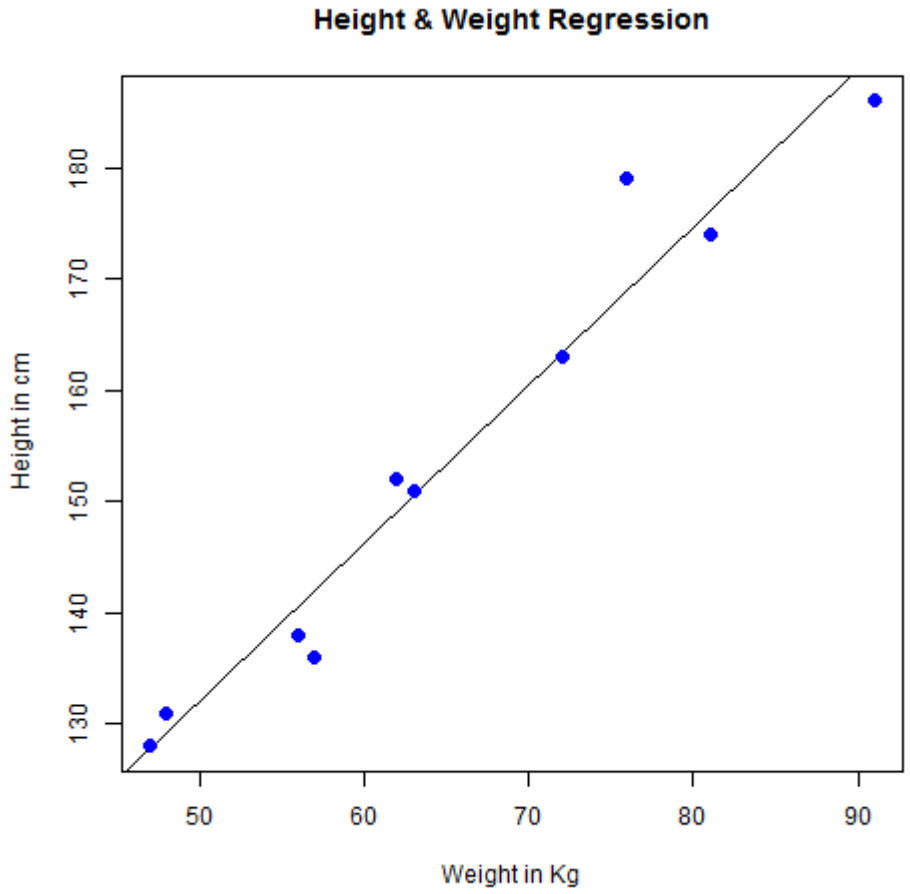
When we execute the above code, it produces the following result −

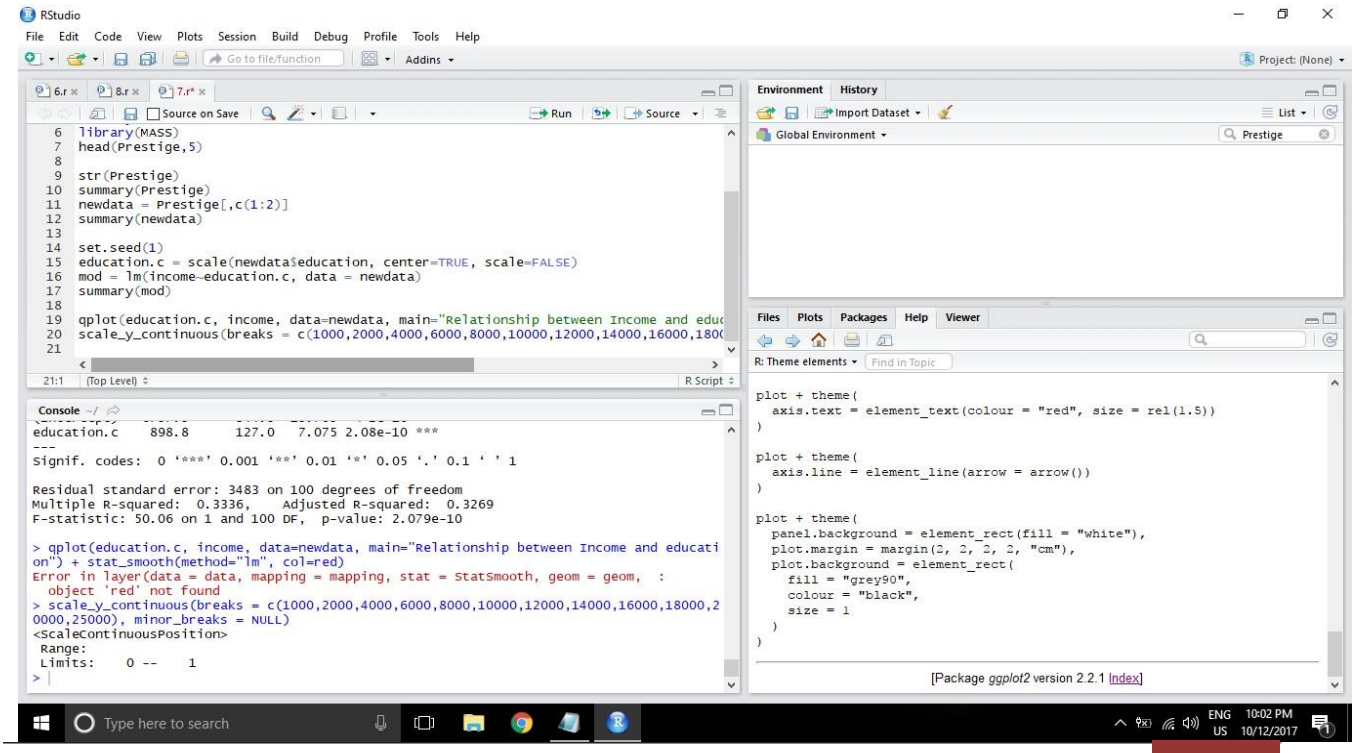
```
     1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

# Plot the chart.

plot(y,x,col = "blue",main = "Height & Weight Regression",

abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.

dev.off()
```

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


Height & Weight Regression
Height in cm
Weight in Kg
130
140
150
160
170
180
50
60
70
80
90


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
6.r 8.r 7.r*
Source on Save
Run
Source
library(MASS)
head(Prestige,5)
str(Prestige)
summary(Prestige)
newdata = Prestige[,c(1:2)]
summary(newdata)
set.seed(1)
education.c = scale(newdata$education, center=TRUE, scale=FALSE)
mod = lm(income~education.c, data = newdata)
summary(mod)
qplot(education.c, income, data=newdata, main="Relationship between Income and educ
scale_y_continuous(breaks = c(1000,2000,4000,6000,8000,10000,12000,14000,16000,180
Environment History
Import Dataset
Global Environment
Prestige
Files Plots Packages Help Viewer
R: Theme elements
plot + theme(
axis.text = element_text(colour = "red", size = rel(1.5))
)
plot + theme(
axis.line = element_line(arrow = arrow())
)
plot + theme(
panel.background = element_rect(fill = "white"),
plot.margin = margin(2, 2, 2, 2, "cm"),
plot.background = element_rect(
fill = "grey90",
colour = "black",
size = 1
)
)
[Package ggplot2 version 2.2.1 Index]
Console
education.c 898.8 127.0 7.075 2.08e-10 ***
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 3483 on 100 degrees of freedom
Multiple R-squared: 0.3336, Adjusted R-squared: 0.3269
F-statistic: 50.06 on 1 and 100 DF, p-value: 2.079e-10
> qplot(education.c, income, data=newdata, main="Relationship between Income and educati
on") + stat_smooth(method="lm", col=red)
Error in layer(data = data, mapping = mapping, stat = StatSmooth, geom = geom, :
object 'red' not found
> scale_y_continuous(breaks = c(1000,2000,4000,6000,8000,10000,12000,14000,16000,18000,2
0000,25000), minor_breaks = NULL)
<ScaleContinuousPosition>
Range:
Limits: 0 -- 1
Type here to search
ENG US 10:02 PM 10/12/2017

EXPERIMENT NO: 8

Aim: Perform Apriori Analysis using arules package.

Objective: To implement Apriori Analysis using arules package.

Theory:

Apriori Algorithm:

With the quick growth in e-commerce applications, there is an accumulation vast quantity of data in months not in years. Data Mining, also known as Knowledge Discovery in Databases(KDD), to find anomalies, correlations, patterns, and trends to predict outcomes.

Apriori algorithm is a classical algorithm in data mining. It is used for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store.

It is very important for effective Market Basket Analysis and it helps the customers in purchasing their items with more ease which increases the sales of the markets. It has also been used in the field of healthcare for the detection of adverse drug reactions. It produces association rules that indicates what all combinations of medications and patient characteristics lead to ADRs.

Association rules

Association rule learning is a prominent and a well-explored method for determining relations among variables in large databases. Let us take a look at the formal definition of the problem of association rules given by Rakesh Agrawal, the President and Founder of the Data Insights Laboratories.

Let $I = \{i_1, i_2, i_3, ..., i_n\}$ be a set of n attributes called items and $D = \{t_1, t_2, ..., t_n\}$ be the set of transactions. It is called database. Every transaction, t_i in D has a unique transaction ID, and it consists of a subset of itemsets in I. A rule can be defined as an implication, $X \longrightarrow Y$ where X and Y are subsets of $I(X, Y \subseteq I)$, and they have no element in common, i.e., $X \cap Y$. X and Y are the antecedent and the consequent of the rule, respectively.

Let's take an easy example from the supermarket sphere. The example that we are considering is quite small and in practical situations, datasets contain millions or billions of transactions. The set of itemsets, I= {Onion, Burger, Potato, Milk, Beer} and a database consisting of six transactions. Each transaction is a tuple of 0's and 1's where 0 represents the absence of an item and 1 the presence.

An example for a rule in this scenario would be {Onion, Potato} => {Burger}, which means that if onion and potato are bought, customers also buy a burger.

Transaction ID	Onion	Potato	Burger	Milk	Beer
t_1	1	1	1	0	0
t_2	0	1	1	1	0
t_3	0	0	0	1	1
t_4	1	1	0	1	0
t_5	1	1	1	0	1
t_6	1	1	1	1	1

There are multiple rules possible even from a very small database, so in order to select the interesting ones, we use constraints on various measures of interest and significance. We will look at some of these useful measures such as support, confidence, lift and conviction.

Support

The support of an itemset X, $supp(X)$ is the proportion of transaction in the database in which the item X appears. It signifies the popularity of an itemset.

$$supp(X) = \frac{\text{Number of transaction in which } X \text{ appears}}{\text{Total number of transactions}}.$$

In the example above, $supp(Onion) = \frac{4}{6} = 0.66667$.

If the sales of a particular product (item) above a certain proportion have a meaningful effect on profits, that proportion can be considered as the support threshold. Furthermore, we can identify itemsets that have support values beyond this threshold as significant itemsets.

Confidence

Confidence of a rule is defined as follows:

$$conf(X \longrightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

It signifies the likelihood of item Y being purchased when item X is purchased. So, for the rule {Onion, Potato} => {Burger},

This implies that for 75% of the transactions containing onion and potatoes, the rule is correct. It can also be interpreted as the conditional probability $P(Y|X)$, i.e, the probability of finding the itemset in Y transactions given the transaction already contains . X

It can give some important insights, but it also has a major drawback. It only takes into account the popularity of the itemset X and not the popularity of Y. If Y is equally popular as X then there will be a higher probability that a transaction containing X will also contain Y thus increasing the confidence. To overcome this drawback there is another measure called lift.

Lift

The lift of a rule is defined as:

$$lift(X \longrightarrow Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)}$$

This signifies the likelihood of the itemset Y being purchased when item X is purchased while taking into account the popularity of Y.

In our example above,

If the value of lift is greater than 1, it means that the itemset Y is likely to be bought with itemset X, while a value less than 1 implies that itemset Y is unlikely to be bought if the itemset X is bought.

Conviction

The conviction of a rule can be defined as:

$$conv(X \longrightarrow Y) = \frac{1 - supp(Y)}{1 - conf(X \longrightarrow Y)}$$

For the rule {onion, potato}=>{burger}

The conviction value of 1.32 means that the rule {onion,potato}=>{burger} would be incorrect 32% more often if the association between X and Y was an accidental chance.

How does Apriori algorithm work?

So far, we learned what the Apriori algorithm is and why is important to learn it.

A key concept in Apriori algorithm is the anti-monotonicity of the support measure. It assumes that

1. All subsets of a frequent itemset must be frequent
2. Similarly, for any infrequent itemset, all its supersets must be infrequent too

Let us now look at the intuitive explanation of the algorithm with the help of the example we used above. Before beginning the process, let us set the support threshold to 50%, i.e. only those items are significant for which support is more than 50%.

Step 1: Create a frequency table of all the items that occur in all the transactions. For our case:

Item	Frequency (No. of transactions)
Onion(O)	4

Potato(P)	5
Burger(B)	4
Milk(M)	4
Beer(Be)	2

Step 2: We know that only those elements are significant for which the support is greater than or equal to the threshold support. Here, support threshold is 50%, hence only those items are significant which occur in more than three transactions and such items are Onion(O), Potato(P), Burger(B), and Milk(M). Therefore, we are left with:

Item	**Frequency (No. of transactions)**
Onion(O)	4
Potato(P)	5
Burger(B)	4
Milk(M)	4

The table above represents the single items that are purchased by the customers frequently.

Step 3: The next step is to make all the possible pairs of the significant items keeping in mind that the order doesn't matter, i.e., AB is same as BA. To do this, take the first item and

pair it with all the others such as OP, OB, OM. Similarly, consider the second item and pair it with preceding items, i.e., PB, PM. We are only considering the preceding items because PO (same as OP) already exists. So, all the pairs in our example are OP, OB, OM, PB, PM, BM.

Step 4: We will now count the occurrences of each pair in all the transactions.

Itemset	Frequency (No. of transactions)
OP	4
OB	3
OM	2
PB	4
PM	3
BM	2

Step 5: Again only those itemsets are significant which cross the support threshold, and those are OP, OB, PB, and PM.

Step 6: Now let's say we would like to look for a set of three items that are purchased together. We will use the itemsets found in step 5 and create a set of 3 items.

To create a set of 3 items another rule, called self-join is required. It says that from the item pairs OP, OB, PB and PM we look for two pairs with the identical first letter and so we get

- **O**P and **O**B, this gives OPB
- **P**B and **P**M, this gives PBM

Next, we find the frequency for these two itemsets.

Itemset	Frequency (No. of transactions)
OPB	4
PBM	3

Applying the threshold rule again, we find that OPB is the only significant itemset.

Therefore, the set of 3 items that was purchased most frequently is OPB.

The example that we considered was a fairly simple one and mining the frequent itemsets stopped at 3 items but in practice, there are dozens of items and this process could continue to many items. Suppose we got the significant sets with 3 items as OPQ, OPR, OQR, OQS and PQR and now we want to generate the set of 4 items. For this, we will look at the sets which have first two alphabets common, i.e,

- **OP**Q and **OP**R gives OPQR
- **OQ**R and **OQ**S gives OQRS

In general, we have to look for sets which only differ in their last letter/item.

Now that we have looked at an example of the functionality of Apriori Algorithm, let us formulate the general process.

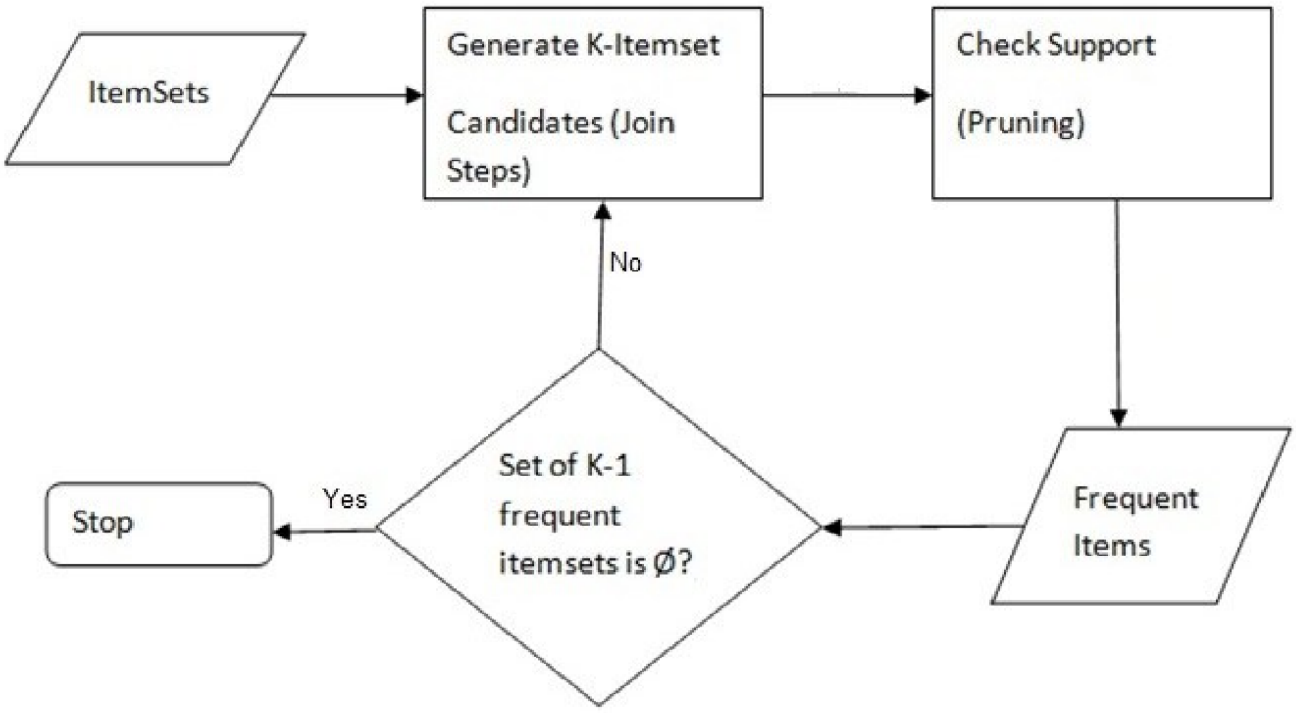
General Process of the Apriori algorithm

The entire algorithm can be divided into two steps:

Step 1: Apply minimum support to find all the frequent sets with k items in a database.

Step 2: Use the self-join rule to find the frequent sets with k+1 items with the help of frequent k-itemsets. Repeat this process from k=1 to the point when we are unable to apply the self-join rule.

This approach of extending a frequent itemset one at a time is called the “bottom up” approach.




```
install.packages("arules")
library(arules)
data("AdultUCI")
class(AdultUCI)
head(AdultUCI)
AdultUCI[["fnlwgt"]]=NULL
AdultUCI[["education-num"]]=NULL
AdultUCI[["age"]]=ordered(cut(AdultUCI[["age"]],c(15,25,45,65,100)),labels=c("young
AdultUCI[["hours-per-week"]]=ordered(cut(AdultUCI[["hours-per-week"]],c(0,25,40,60,
                                labels=c("part-time","full-time","overtime","v
```

```
[1] "data.frame"
> head(AdultUCI)
  age         workclass fnlwgt education education-num      marital-status
1  39         State-gov  77516 Bachelors            13       Never-married
2  50 Self-emp-not-inc  83311 Bachelors            13  Married-civ-spouse
3  38           Private 215646   HS-grad             9            Divorced
4  53           Private 234721      11th             7  Married-civ-spouse
5  28           Private 338409 Bachelors            13  Married-civ-spouse
6  37           Private 284582   Masters            14  Married-civ-spouse
        occupation  relationship  race    sex capital-gain capital-loss
1     Adm-clerical Not-in-family White   Male         2174            0
2  Exec-managerial       Husband White   Male            0            0
3 Handlers-cleaners Not-in-family White   Male            0            0
4 Handlers-cleaners       Husband Black   Male            0            0
5    Prof-specialty          Wife Black Female            0            0
6  Exec-managerial          Wife White Female            0            0
  hours-per-week native-country income
1             40  United-States  small
2             13  United-States  small
3             40  United-States  small
4             40  United-States  small
5             40           Cuba  small
6             40  United-States  small
>
```

```
plot + theme(
  axis.text = element_text(colour = "red", size = rel(1.5))
)

plot + theme(
  axis.line = element_line(arrow = arrow())
)

plot + theme(
  panel.background = element_rect(fill = "white"),
  plot.margin = margin(2, 2, 2, 2, "cm"),
  plot.background = element_rect(
    fill = "grey90",
    colour = "black",
    size = 1
  )
)
```

EXPERIMENT NO: 9

Aim: Case Study: Implement K-Means Algorithm on worlddata dataset and Visualize the clusters.

Objective: Implementing K-Means Algorithm on given dataset and visualizing the Clusters.

Theory:

K Means

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have the specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps:

- Reassign data points to the cluster whose centroid is closest.
- Calculate new centroid of each cluster.

These two steps are repeated till the within cluster variation cannot be reduced any further. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

Exploring the data

The iris dataset contains data about sepal length, sepal width, petal length, and petal width of flowers of different species. Let us see what it looks like:

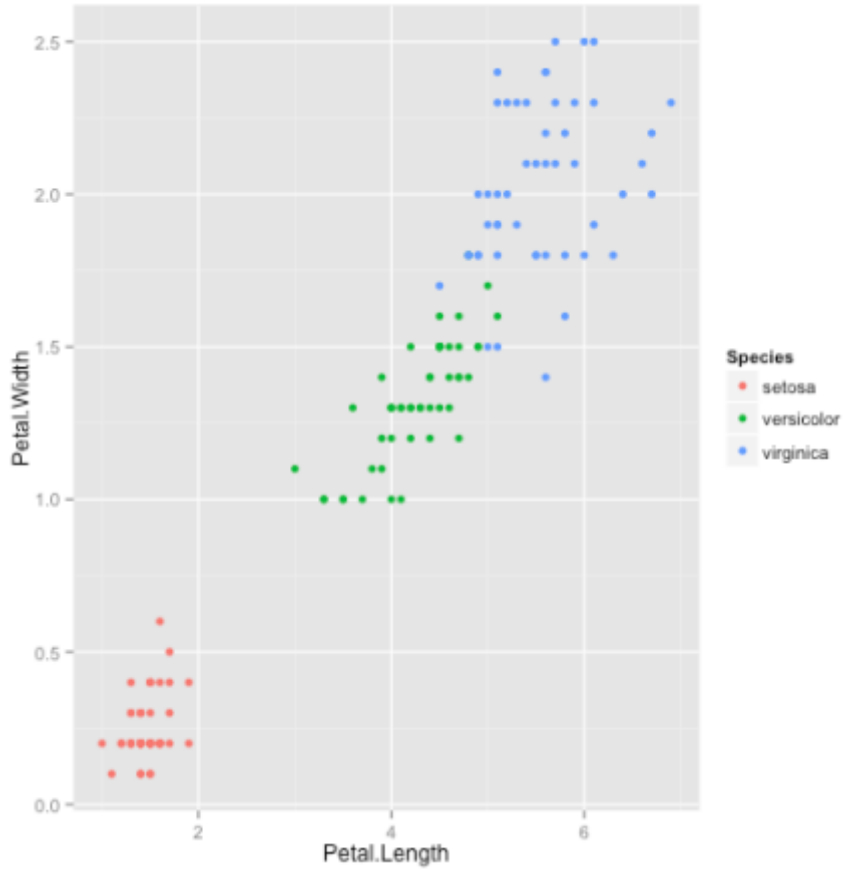
```
library(datasets)
head(iris)
```

	Sepal.Length	*Sepal.Width*	*Petal.Length*	*Petal.Width*	*Species*
1	*5.1*	*3.5*	*1.4*	*0.2*	*setosa*
2	*4.9*	*3.0*	*1.4*	*0.2*	*setosa*
3	*4.7*	*3.2*	*1.3*	*0.2*	*setosa*
4	*4.6*	*3.1*	*1.5*	*0.2*	*setosa*
5	*5.0*	*3.6*	*1.4*	*0.2*	*setosa*
6	*5.4*	*3.9*	*1.7*	*0.4*	*setosa*

After a little bit of exploration, I found that Petal.Length and Petal.Widthwere similar among the same species but varied considerably between different species, as demonstrated below:

```
library(ggplot2)
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
```

Here is the plot:



pamk

Partitioning Around Medoids With Estimation Of Number Of Clusters

This calls the function pam or clara to perform a partitioning around medoids clustering with the number of clusters estimated by optimum average silhouette width (see pam.object) or Calinski-Harabasz index (calinhara). The Duda-Hart test (dudahart2) is applied to decide whether there should be more than one cluster (unless 1 is excluded as number of clusters or data are dissimilarities).

Usage

```
pamk(data,krange=2:10,criterion="asw", usepam=TRUE, scaling=FALSE,
alpha=0.001, diss=inherits(data, "dist"), critout=FALSE, ns=10, seed=NULL,
...)
```

Arguments

data

a data matrix or data frame or something that can be coerced into a matrix, or dissimilarity matrix or object. See pam for more information.

krange

integer vector. Numbers of clusters which are to be compared by the average silhouette width criterion. Note: average silhouette width and Calinski-Harabasz can't estimate number of clusters nc=1. If 1 is included, a Duda-Hart test is applied and 1 is estimated if this is not significant.

criterion

one of "asw", "multiasw" or "ch". Determines whether average silhouette width (as given out by pam/clara, or as computed by distcritmulti if "multiasw" is specified; recommended for large data sets with usepam=FALSE) or Calinski-Harabasz is applied. Note that the original Calinski-Harabasz index is not defined for dissimilarities; if dissimilarity data is run with criterion="ch", the dissimilarity-based generalisation in Hennig and Liao (2013) is used.

usepam

logical. If TRUE, pam is used, otherwise clara (recommended for large datasets with 2,000 or more observations; dissimilarity matrices can not be used with clara).

scaling

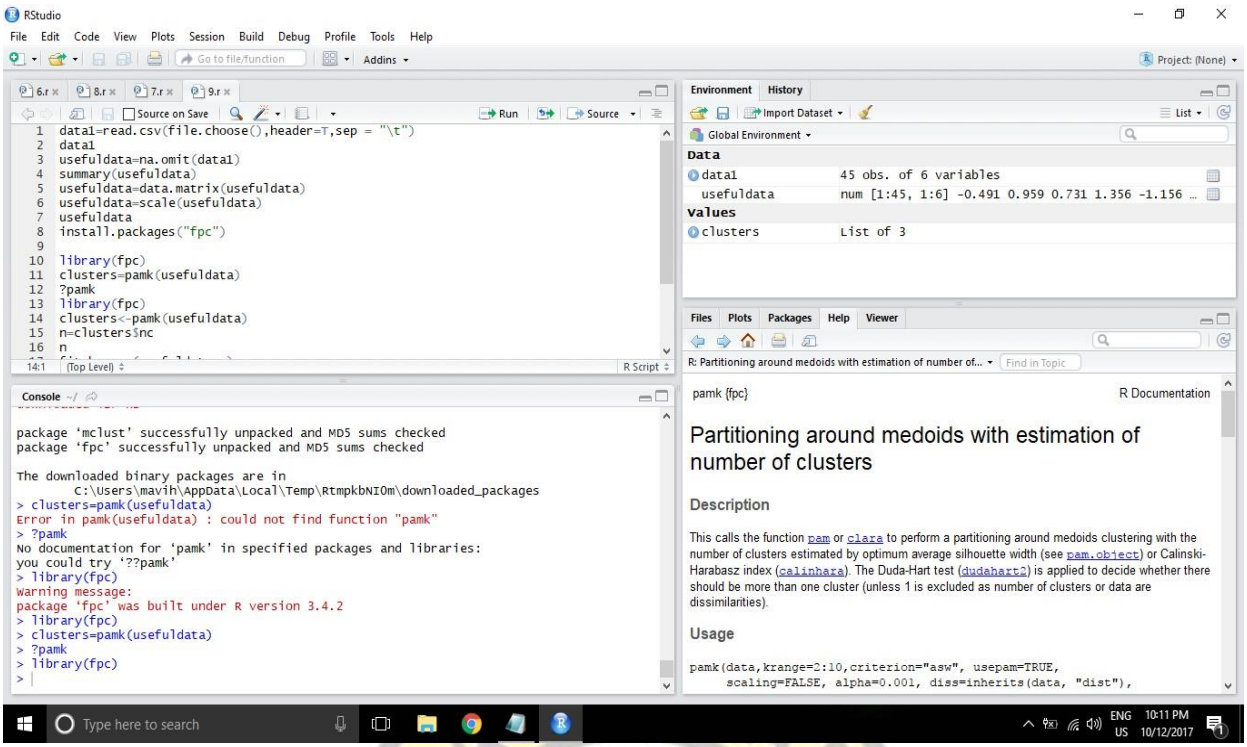
either a logical value or a numeric vector of length equal to the number of variables. If scaling is a numeric vector with length equal to the number of variables, then each variable is divided by the corresponding value from scaling. If scaling is TRUE then scaling is done by dividing the (centered) variables by their root-mean-square, and if scaling is FALSE, no scaling is done.

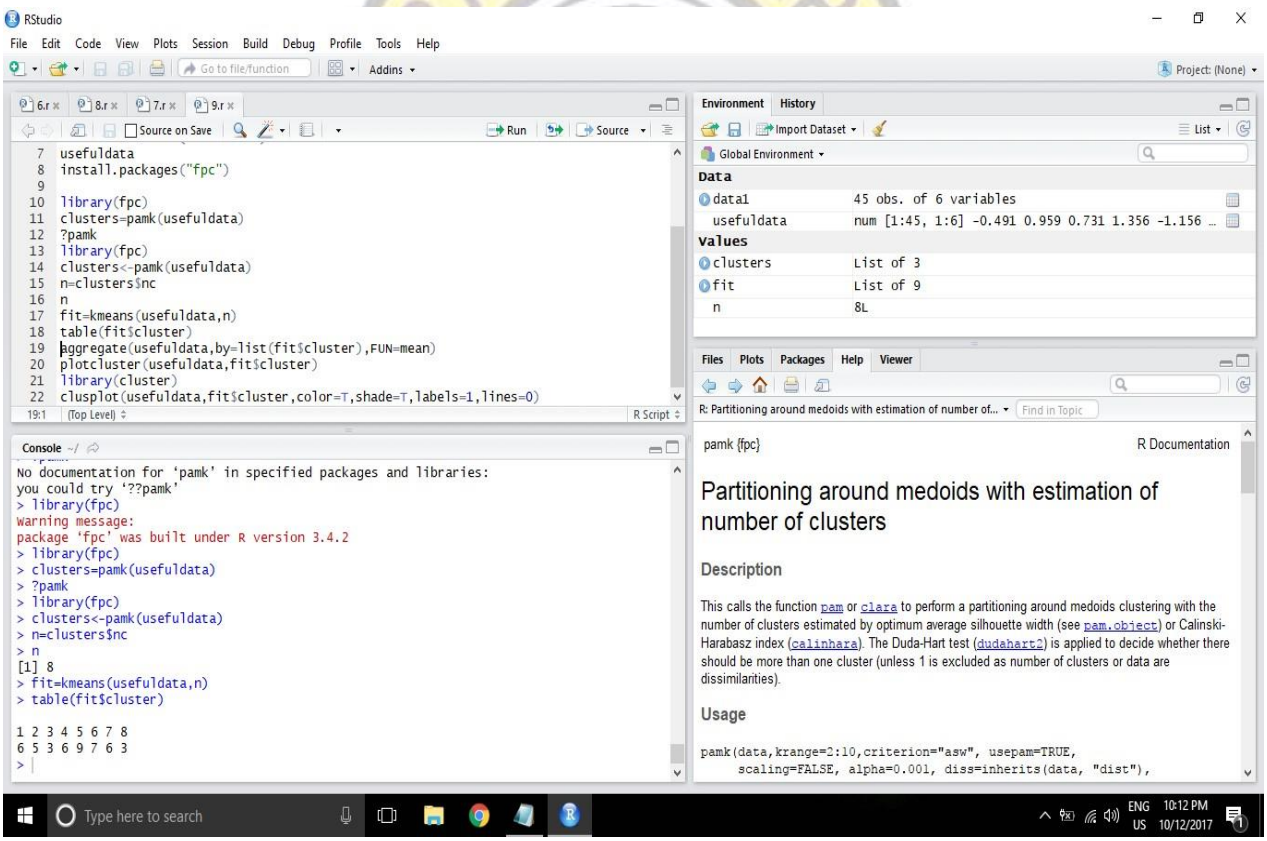
alpha

numeric between 0 and 1, tuning constant for dudahart2 (only used for 1-cluster test).

diss

logical flag: if TRUE (default for dist or dissimilarity-objects), then data will be considered as a dissimilarity matrix (and the potential number of clusters 1 will be ignored). If FALSE, then data will be considered as a matrix of observations by variables.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
6.r 8.r 7.r 9.r
1 data1=read.csv(file.choose(),header=T,sep = "\t")
2 data1
3 usefuldata=na.omit(data1)
4 summary(usefuldata)
5 usefuldata=data.matrix(usefuldata)
6 usefuldata=scale(usefuldata)
7 usefuldata
8 install.packages("fpc")
10 library(fpc)
11 clusters=pamk(usefuldata)
12 ?pamk
13 library(fpc)
14 clusters<-pamk(usefuldata)
15 n=clusters$nc
16 n
Environment History
Data
data1 45 obs. of 6 variables
usefuldata num [1:45, 1:6] -0.491 0.959 0.731 1.356 -1.156 ...
values
clusters List of 3
Files Plots Packages Help Viewer
R: Partitioning around medoids with estimation of number of...
pamk {fpc} R Documentation
Partitioning around medoids with estimation of number of clusters
Description
This calls the function pam or clara to perform a partitioning around medoids clustering with the number of clusters estimated by optimum average silhouette width (see pam.object) or Calinski-Harabasz index (calinhara). The Duda-Hart test (dudahart2) is applied to decide whether there should be more than one cluster (unless 1 is excluded as number of clusters or data are dissimilarities).
Usage
pamk(data,krange=2:10,criterion="asw", usepam=TRUE,
scaling=FALSE, alpha=0.001, diss=inherits(data, "dist"),
Console
package 'mclust' successfully unpacked and MD5 sums checked
package 'fpc' successfully unpacked and MD5 sums checked
The downloaded binary packages are in
C:\Users\mavih\AppData\Local\Temp\RtmpkbNIOm\downloaded_packages
> clusters=pamk(usefuldata)
Error in pamk(usefuldata) : could not find function "pamk"
> ?pamk
No documentation for 'pamk' in specified packages and libraries:
you could try '??pamk'
> library(fpc)
Warning message:
package 'fpc' was built under R version 3.4.2
> library(fpc)
> clusters=pamk(usefuldata)
> ?pamk
> library(fpc)
Type here to search
ENG US 10:11 PM 10/12/2017


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
7 usefuldata
8 install.packages("fpc")
10 library(fpc)
11 clusters=pamk(usefuldata)
12 ?pamk
13 library(fpc)
14 clusters<-pamk(usefuldata)
15 n=clusters$nc
16 n
17 fit=kmeans(usefuldata,n)
18 table(fit$cluster)
19 aggregate(usefuldata,by=list(fit$cluster),FUN=mean)
20 plotcluster(usefuldata,fit$cluster)
21 library(cluster)
22 clusplot(usefuldata,fit$cluster,color=T,shade=T,labels=1,lines=0)
Environment History
Data
data1 45 obs. of 6 variables
usefuldata num [1:45, 1:6] -0.491 0.959 0.731 1.356 -1.156 ...
values
clusters List of 3
fit List of 9
n 8L
Files Plots Packages Help Viewer
R: Partitioning around medoids with estimation of number of...
pamk {fpc} R Documentation
Partitioning around medoids with estimation of number of clusters
Description
This calls the function pam or clara to perform a partitioning around medoids clustering with the number of clusters estimated by optimum average silhouette width (see pam.object) or Calinski-Harabasz index (calinhara). The Duda-Hart test (dudahart2) is applied to decide whether there should be more than one cluster (unless 1 is excluded as number of clusters or data are dissimilarities).
Usage
pamk(data,krange=2:10,criterion="asw", usepam=TRUE,
scaling=FALSE, alpha=0.001, diss=inherits(data, "dist"),
Console
No documentation for 'pamk' in specified packages and libraries:
you could try '??pamk'
> library(fpc)
Warning message:
package 'fpc' was built under R version 3.4.2
> library(fpc)
> clusters=pamk(usefuldata)
> ?pamk
> library(fpc)
> clusters<-pamk(usefuldata)
> n=clusters$nc
> n
[1] 8
> fit=kmeans(usefuldata,n)
> table(fit$cluster)
1 2 3 4 5 6 7 8
6 5 3 6 9 7 6 3
Type here to search
ENG US 10:12 PM 10/12/2017

EXPERIMENT NO: 10

Aim: Mini Project (group of 2 students) Create an Interactive Dashboard using shiny package.

Objective: To implement a Mini Project using Shiny Package.

Theory:

Introducing Shiny

Shiny is a new package from RStudio that makes it incredibly easy to build interactive web applications with R.

For an introduction and live examples, visit the Shiny homepage. Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions.

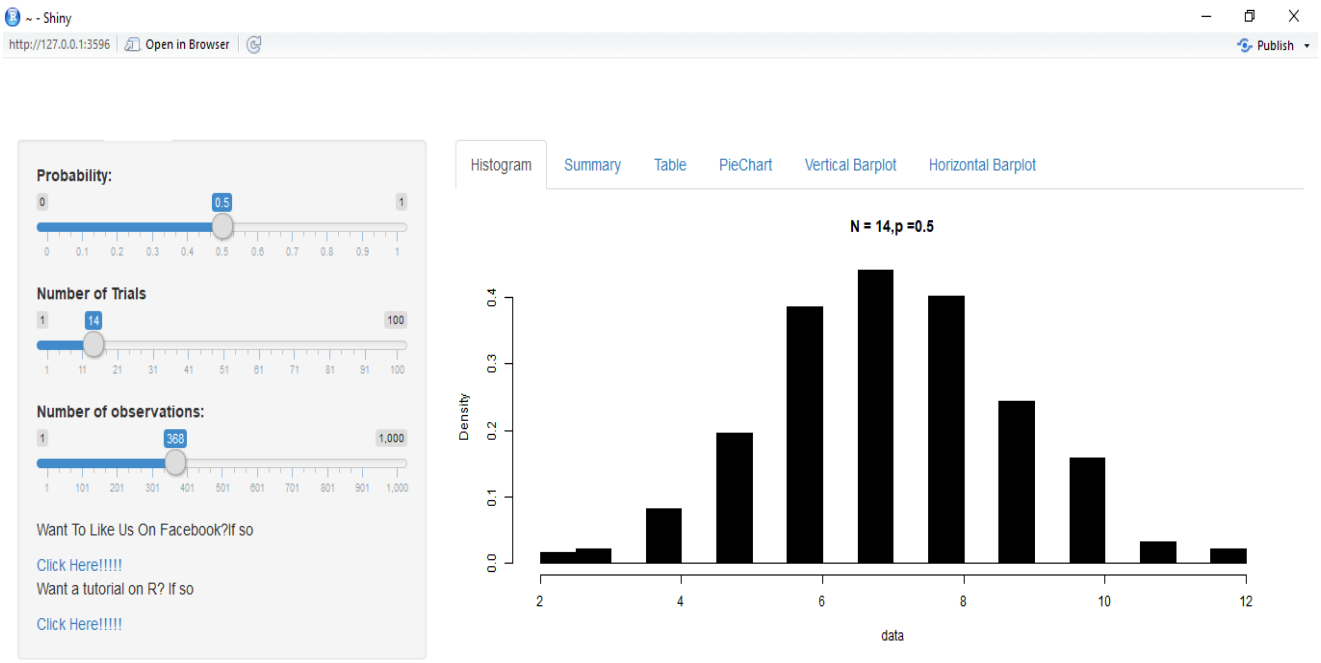
Features:

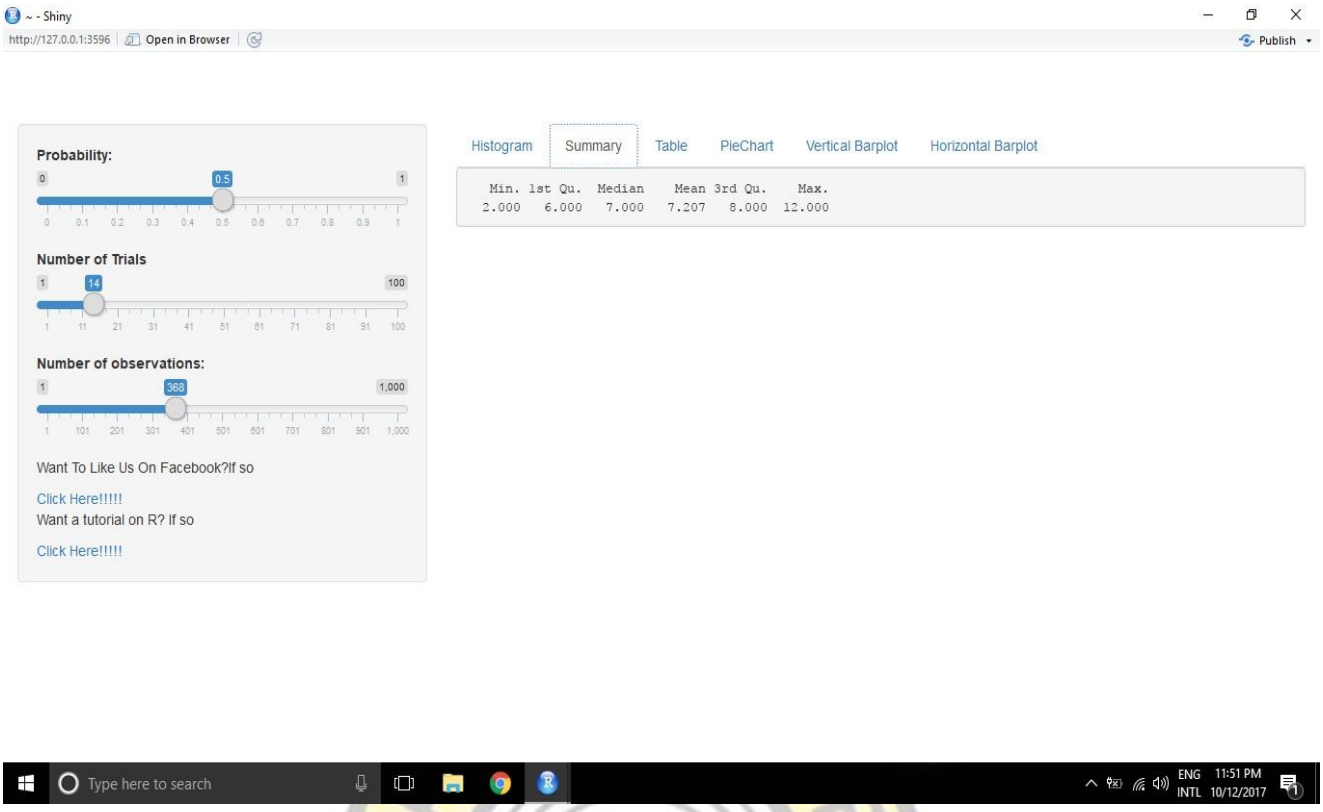
- Build useful web applications with only a few lines of code—no JavaScript required.
- Shiny applications are automatically "live" in the same way that spreadsheets are live. Outputs change instantly as users modify inputs, without requiring a reload of the browser.
- Shiny user interfaces can be built entirely using R, or can be written directly in HTML, CSS, and JavaScript for more flexibility.
- Works in any R environment (Console R, Rgui for Windows or Mac, ESS, StatET, RStudio, etc.)
- Attractive default UI theme based on Twitter Bootstrap.
- A highly customizable slider widget with built-in support for animation.
- Pre-built output widgets for displaying plots, tables, and printed output of R objects.
- Fast bidirectional communication between the web browser and R using the websockets package.
- Uses a reactive programming model that eliminates messy event handling code, so you can focus on the code that really matters.
- Develop and redistribute your own Shiny widgets that other developers can easily drop into their own applications (coming soon!).

Installation

Shiny is available on CRAN, so you can install it in the usual way from your R console:

```
install.packages("shiny")
```




~ - Shiny
http://127.0.0.1:3596
Open in Browser
Publish
Probability:
0
0.5
1
Number of Trials
1
14
100
Number of observations:
1
368
1,000
Want To Like Us On Facebook?If so
Click Here!!!!!
Want a tutorial on R? If so
Click Here!!!!!
Histogram
Summary
Table
PieChart
Vertical Barplot
Horizontal Barplot
Min. 1st Qu. Median Mean 3rd Qu. Max.
2.000 6.000 7.000 7.207 8.000 12.000

Type here to search
ENG
INTL
11:51 PM
10/12/2017