SIR C R REDDY COLLEGE OF ENGINEERING

ELURU

CERTIFICATE

This is to certify that this is the BONAFIDE RECORD of the work done in ***Data Mining Techniques With R (R2031122)*** *laboratory by Mr / Ms…….. ..... bearing Regd.No…….. of III/IV B.Tech I Semester course during 2023-24.*

Total number of experiments held …**14**

Total Number of experiments done...**14**

LAB-IN-CHARGE

HEAD OF THE DEPARTMENT

EXTERNAL EXAMINER

DATA MINING TECHNIQUES WITH R LAB

MANUAL

(R2031122)

III/IV B.Tech, I- Semester

Academic Year: 2023-24

DEPARTMENT OF INFORMATION TECHNOLOGY

SIR C.R.REDDY COLLEGE OF ENGINEERING
ELURU-534007, WEST GODAVARI DIST, A P., INDIA
(Approved by AICTE, New Delhi, Accredited by NBA & Affiliated to JNTU- KAKINADA)
Phone no: 08812-230840, 2300656 Fax: 08812-224193
Visit us at http://www.sircrrengg.ac.in

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION OF THE DEPARTMENT

To be a premier Department in the region in the field of Information Technology through academic excellence and research that enable graduates to meet the challenges of industry and society

MISSIONOF THE DEPARTMENT

- To Provide dynamic teaching-learning environment to make the students industry ready and advancement in career;
- To inculcate professional and leadership quality for better employability and entrepreneurship;
- To make high quality professional with moral and ethical values suitable for industry and society

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using the first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems and system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, society, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in society and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multi-disciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.

PO12: Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO No.	PSO Description
PSO 1	**Conceptual Skills:** Apply the core concepts of system architecture, information management, programming, networking for IT applications.
PSO 2	**Technical Skills:** Design and develop software by adapting emerging technologies for the need of IT industry.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO No.	PEO Description
PEO 1	Solve real world problems through effective professional skills in Information Technology industry and academic research.
PEO 2	Analyze and develop applications in Information Technology domain and adapt to changing technology trends with continuous learning.
PEO 3	Practice the profession in society with ethical and moral values.

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
    c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

SYLLABUS

List of Programs:

1. Implement all basic R commands.
2. Interact data through .csv files (Import from and export to .csv files).
3. Get and Clean data using swirl exercises.(Use 'swirl' package, library and install that topic from swirl).
4. Visualize all Statistical measures (Mean, Mode, Median, Range, Inter Quartile Range etc., using Histograms, Boxplots and Scatter Plots).
5. Create a data frame with the following structure.

EMP ID EMP	NAME	SALARY	START DATE
1	Satish	5000	01-11-2013
2	Vani	7500	05-06-2011
3	Ramesh	10000	21-09-1999
4	Praveen	9500	13-09-2005
5	Pallavi	4500	23-10-2000

   a. Extract two column names using column name.
   b. Extract the first two rows and then all columns.
   c. Extract 3rd and 5th row with 2nd and 4th column.

6. Write R Program using 'apply' group of functions to create and apply normalization function on each of the numeric variables/columns of iris dataset to transform them into
   i. 0 to 1 range with min-max normalization.
   ii. a value around 0 with z-score normalization.
7. Create a data frame with 10 observations and 3 variables and add new rows and columns to it using 'rbind' and 'cbind' function.
8. Write R program to implement linear and multiple regression on 'mtcars' dataset to estimate the value of 'mpg' variable, with best R2 and plot the original values in 'green' and predicted values in 'red'.
9. Write R Programs to implement k-means clustering.
10. Write R Programs to implement k-medoids clustering.

11. Write R Programs to implement density based clustering on iris dataset.
12. Write a R Program to implement decision trees using ‘reading Skills’ dataset.
13. Implement decision trees using ‘iris’ dataset using package party and ‘rpart’.
14. Use a Corpus() function to create a data corpus then Build a term Matrix and Reveal word frequencies.

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

COURSE OUTCOMES

CO Number	COs	Level
CO1	**Understand** the functionality of R by using add-on packages	L2
CO2	**Apply** various statistical functions and examine data from files and other sources and perform various data manipulation tasks on them.	L3
CO3	**Apply** preprocessing techniques and mining methods to extract knowledge using R Graphics and tables to visualize the results.	L3
CO4	**Analyze** the data for real life applications	L4

CO-PO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
CO1														
CO2		3												
CO3		3												
CO4			3		2								3	2
AVG		**3**	**3**	**-**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**3**	**2**

CO-PO MAPPING JUSTIFICATION

CO2	**PO2**	**3**	The student will be able to apply various statistical and other mathematical functions for flexible analysis of the data.
CO3	**PO2**	**3**	The student will learn and apply various preprocessing techniques and visualization techniques.
CO4	**PO3**	**3**	The student will be in a position to handle various types of data by examining, preprocessing and apply various mining techniques.
CO4	**PO5**	**2**	The student will be able to select, and apply appropriate techniques, resources and IT tools including prediction to extract knowledge.
CO4	**PSO1**	**3**	Design and develop methods and techniques in the area of relevance under realistic constraints.
CO4	**PSO2**	**2**	Adopt new and fast emerging technologies related to data warehousing and mining in the field of Information Technology.

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

INDEX

11	Write R Programs to implement density based clustering on iris dataset.	CO4		
12	Write a R Program to implement decision trees using ‘reading Skills’ dataset.	CO4		
13	Implement decision trees using ‘iris’ dataset using package party and ‘rpart’.	CO4		
14	Use a Corpus() function to create a data corpus then Build a term Matrix and Reveal word frequencies.	CO4		

SIR C.R.REDDY COLLEGE OF ENGINEERING

ELURU-534007, WEST GODAVARI DIST, A P., INDIA

(Approved by AICTE, New Delhi , Accredited by NBA & Affiliated to JNTU- KAKINADA)

Phone no: 08812-230840, 2300656 Fax: 08812-224193

Visit us at http://www.sircrrengg.ac.in

CONTENT BEYOND SYLLABUS

S.No	Name of Experiment	CO	Dates	Page No
1	Implementation of Principle Component Analysis in R	CO3		
2	Implementation of SVM in R	CO4		

Experiment 1: Implement all basic R commands.

Execution:

Input and display

1+1 #addition of 2 numbers

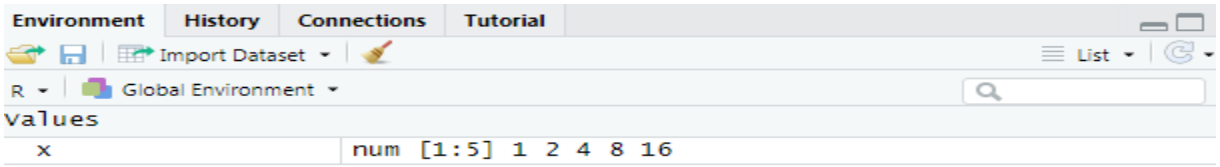
>1+1

[1] 2

x=c(1,2,4,8,16) #create a data vector with specified elements

> x=c(1,2,4,8,16)

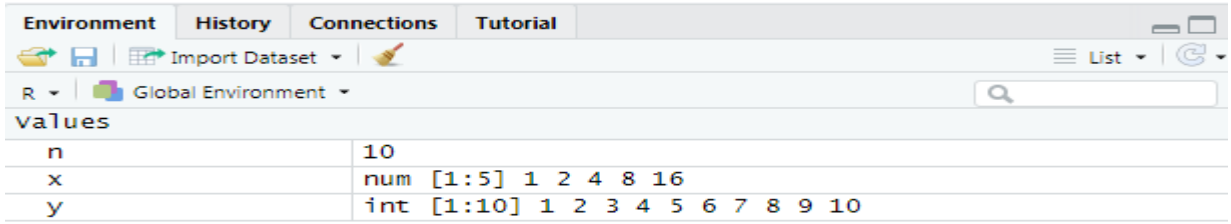
#create a data vector with elements 1-10



> y=c(1:10)

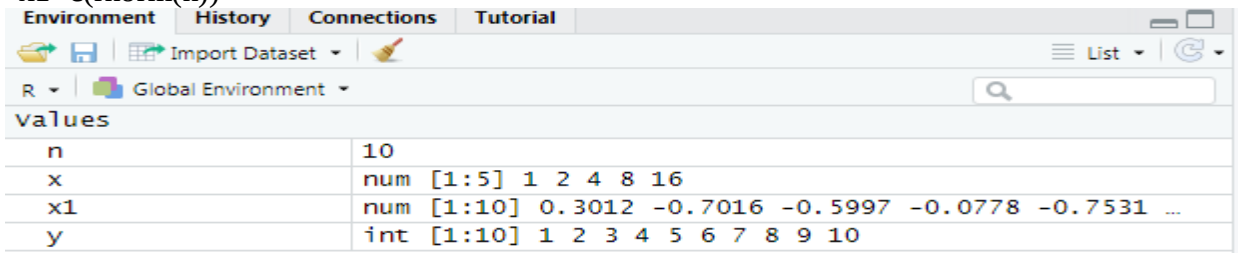
n=10

> n=10



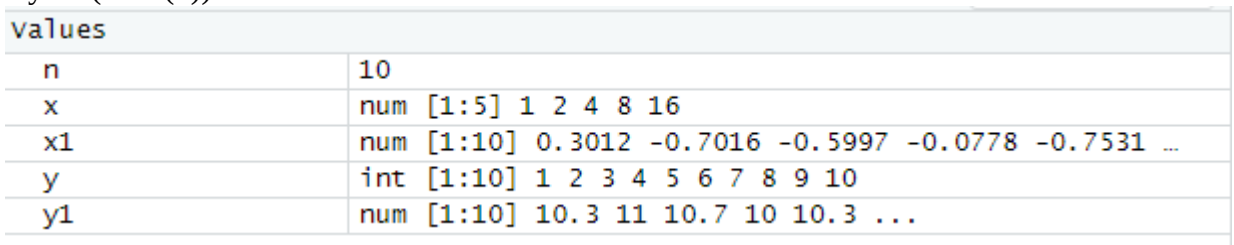
x1=c(rnorm(n)) #create a n item vector of random normal deviates

>x1=c(rnorm(n))



y1=c(runif(n))+n #create another n item vector that has n added to each random uniform distribution

> y1=c(runif(n))+n



vect=c(x,y) #combine them into one vector of length 2n

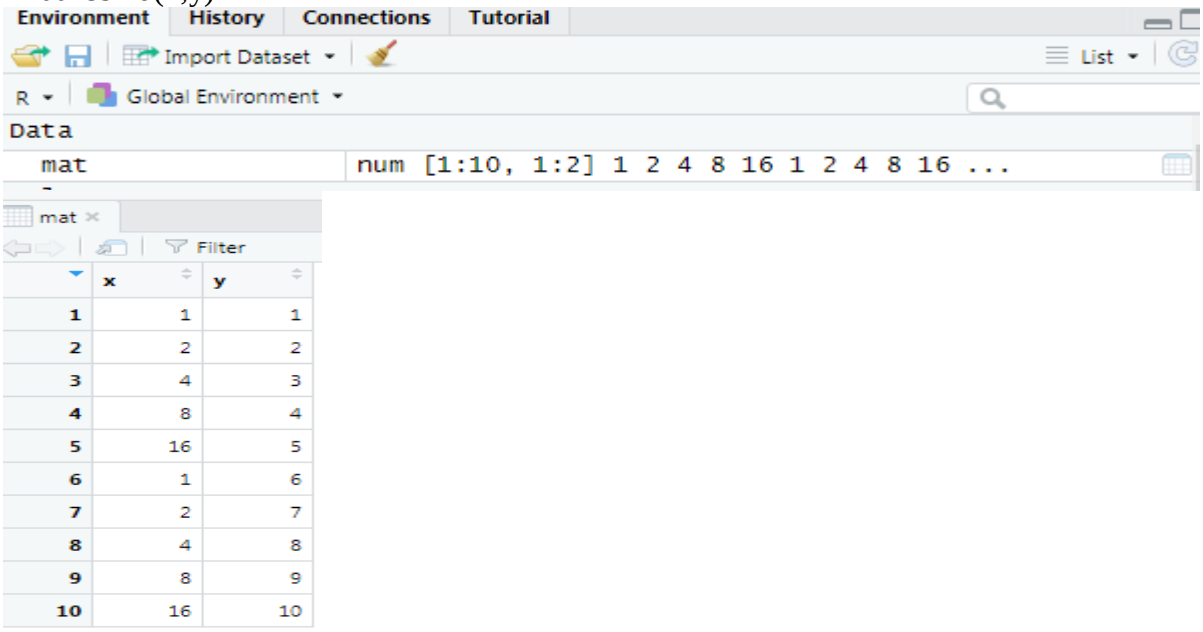
>vect=c(x,y)

Values

n	10
vect	num [1:15] 1 2 4 8 16 1 2 3 4 5 ...
x	num [1:5] 1 2 4 8 16
x1	num [1:10] 0.3012 -0.7016 -0.5997 -0.0778 -0.7531 ...
y	int [1:10] 1 2 3 4 5 6 7 8 9 10
y1	num [1:10] 10.3 11 10.7 10 10.3 ...

mat=cbind(x,y) #combine them into a n x 2 matrix

>mat=cbind(x,y)



	x	y
1	1	1
2	2	2
3	4	3
4	8	4
5	16	5
6	1	6
7	2	7
8	4	8
9	8	9
10	16	10

mat[4,2] #display the 4th row and the 2nd column

>mat[4,2]
y
4
mat[3,] #display the 3rd row

>mat[3,]
x y
4 3
mat[,2] #display the 2nd column

>mat[,2]
[1] 1 2 3 4 5 6 7 8 9 10
subset(mtcars, mpg>10) #those objects meeting a logical criterion

Moving Around

ls() #list the variables in the workspace
>ls()
[1] "mat" "n" "vect" "x" "x1" "y" "y1"
rm(x) #remove x from the workspace
>rm(x)

Data	
mat	num [1:10, 1:2] 1 2 4 8 16 1 2 4 8 16 ...
Values	
n	10
vect	num [1:15] 1 2 4 8 16 1 2 3 4 5 ...
x1	num [1:10] 0.3012 -0.7016 -0.5997 -0.0778 -0.7531 …
y	int [1:10] 1 2 3 4 5 6 7 8 9 10
y1	num [1:10] 10.3 11 10.7 10 10.3 ...

rm(list=ls()) #remove all the variables from the workspace

>rm(list=ls())

>ls()
character(0)
attach(mat) #make the names of the variables in the matrix or data frame available in the workspace
> data<-data.frame(x1 = c(9,8,3,4,8), x2=c(5,4,7,1,1), x3=c(1,2,3,4,5))

>x1
Error: object 'x1' not found
>attach(data)

>x1
[1] 9 8 3 4 8
detach(mat) #releases the names

>detach(data)

>x1
Error: object 'x1' not found
new=old[,-n] #copies the nth column

>new=MatrixA[,-5]

mat × MatrixA × new ×
Filter

	V1	V2	V3
1	1	4	7
2	2	5	8
3	3	6	9

mat × MatrixA × new ×
Filter

	V1	V2	V3
1	1	4	7
2	2	5	8
3	3	6	9

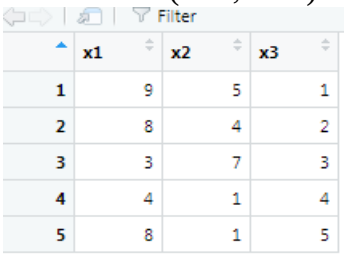
new=old[n,] #copies the nth row

>new=MatrixA[3,]

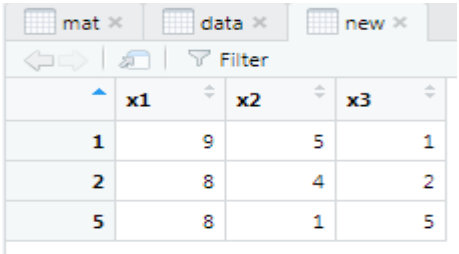
	V1
1	3
2	6
3	9

new=subset(oldgical) #select those cases that meet the logical condition

>new=subset(data,x1>6)



	x1	x2	x3
1	9	5	1
2	8	4	2
3	3	7	3
4	4	1	4
5	8	1	5



	x1	x2	x3
1	9	5	1
2	8	4	2
5	8	1	5

new=old[n1:n2,n3:n4] #select the n1 through n2 rows of variables n3 through n4)

Statistics and Transformations

max(x) #find the maximum
> x=c(1,2,4,8,16)
>max(x)
[1] 16

min(x) #find the minimum
>min(x)
[1] 1

mean(x) #find the mean value
>mean(x)
[1] 6.2

median(x) #find the median value
>median(x)
[1] 4

sum(x) #find the sum of value
>sum(x)
[1] 31

var(x) #produces the variance covariance matrix
>var(x)
[1] 37.2

sd(x) #standard deviation
>sd(x)
[1] 6.09918

mad(x) #(median absolute deviation)
>mad(x)
[1] 4.4478

Distributions

factorial(x)
> z=5
>factorial(z)
[1] 120

Math Functions

round(x,n) #rounds the values of x to n decimal places
> z=5.325

```
>round(z,2)
[1] 5.32

ceiling(x)                          #vector x of smallest integers > x
>ceiling(z)
[1] 6

floor(x)                            #vector x of largest interger< x
>floor(z)
[1] 5

as.integer(x)                       #truncates real x to integers (compare to round(x,0)
>as.integer(z)
[1] 5

x%in%y                              #tests each element of x for membership in y
>x%in%y
[1]  TRUETRUETRUETRUE FALSE.

y%in%x                               #tests each element of y for membership in x
>y%in%x
 [1]  TRUETRUE FALSE  TRUE FALSE FALSEFALSE  TRUE FALSE FAL

all(x%in%y)                         #true if x is a proper subset of y
>all(x%in%y)
[1] FALSE
```

Experiment 2: Interact data through .csv files (Import from and export to .csv files)

Procedure:

We will learn to read data from a .csv file and then write data into a .csv file.

How to ImportData from .csv files

1. To import data we need the file path and an extension. First of all, the path is the location where the data will be stored.
2. R allows the users to import the data into different types of files. Here we use .csv file
3. Check your working directory by **getwd()**
4. The file should be present in current working directory so that R can read it.
5. Create file with data.
6. Make sure the file should be saved in .csv (comma delimited) file.
7. Import the .csv file to your global environment by read.csv( )
8. Assign the .csv file in to new file.
9. View the file

Execution:

```
>getwd()
[1] "C:/Users/HOME/Documents"
>setwd("C:/Users/HOME/Desktop")
>output<-read.csv("input.csv", header = TRUE, sep = ",")
>View(output)
```

Output:

output

Filter

	id	name	salary	start_date	department
1	1	Rick	623.30	1/1/2012	it
2	2	Dan	515.20	9/23/2013	Operations
3	3	Michelle	611.00	11/15/2014	IT
4	4	Ryan	729.00	5/11/2014	HR
5	NA	Gary	843.25	3/27/2015	Finance
6	6	Nina	578.00	5/21/2013	IT
7	7	Simon	632.80	7/30/2013	Operations
8	8	Guru	722.50	6/17/2014	Finance

Procedure:

How to ExportData from.csv Files

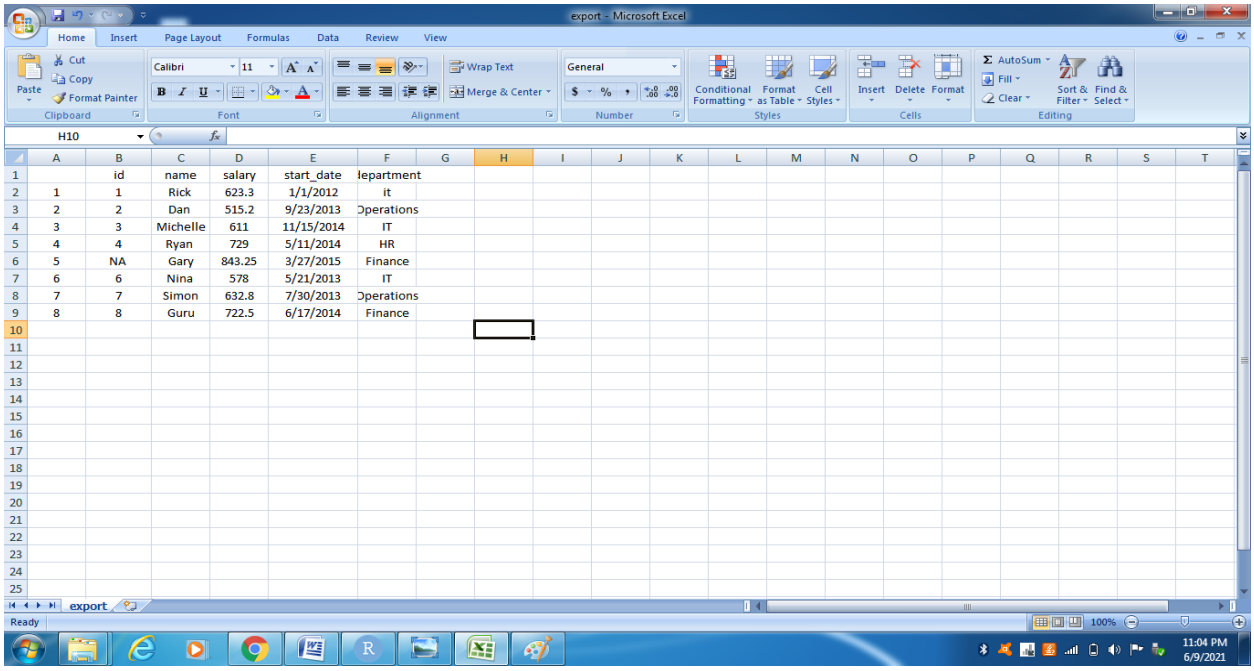
1. Export data from R to .csvfile is done by write.csv( ).
2. Set the directory where should the .csv file should export.

3. Create name of the .csv file where exported data has to be stored using wrie.csv( ).
4. Finally the .csv file data from R will exported to the .csv file

Execution:

```
>setwd("C:/Users/HOME/Desktop")
>write.csv(output,"export.csv")
```

Output:



	A	B	C	D	E	F
1		id	name	salary	start_date	lepartment
2	1	1	Rick	623.3	1/1/2012	it
3	2	2	Dan	515.2	9/23/2013	Operations
4	3	3	Michelle	611	11/15/2014	IT
5	4	4	Ryan	729	5/11/2014	HR
6	5	NA	Gary	843.25	3/27/2015	Finance
7	6	6	Nina	578	5/21/2013	IT
8	7	7	Simon	632.8	7/30/2013	Operations
9	8	8	Guru	722.5	6/17/2014	Finance

Experiment 3: Get and Clean data using swirl exercises(Use 'swirl' package, library and install that topic from swirl).

```
>install.packages("swirl")
>library(swirl) // loads the swirl library Hi! Type swirl() when you are ready to begin.
> swirl() //opens the swirl lessons
| Welcome to swirl! Please sign in. If you've been here before, use the same name
| as you did then. If you are new, call yourself something unique.
What shall I call you?
Crrit Select 1, 2, or 3 and press Enter
1: Continue. 2: Proceed. 3: Let's get going!
Selection: 1 : R Programming: The basics of programming in R
2: Regression Models: The basics of regression modeling in R
 3: Statistical Inference: The basics of statistical inference in R
4: Exploratory Data Analysis: The basics of exploring data in R
5: Don't install anything for me. I'll do it myself.
 Enter an item from the menu, or 0 to exit Selection: 4
|======================================================================| 100%
| Course installed successfully!
 1: Exploratory Data Analysis
 2: Take me to the swirl course repository!
Selection:0
| Leaving swirl now. Type swirl() to resume.
>install_course("Getting_and_Cleaning_Data")
|======================================================================| 100%
| Please choose a lesson, or type 0 to return to course menu.
 1: Manipulating Data with dplyr
2: Grouping and Chaining with dplyr
3: Tidying Data with tidyr
4: Dates and Times with lubridate
Selection: 1
 | Attempting to load lesson dependencies...
| Package 'dplyr' loaded correctly! Error in yaml.load(readLines(con, warn = readLines.warn), error.label
= error.label, : (C:/Users/sircrrengg/AppData/Local/R/win-library/4.3/swirl/Courses/Get
ting_and_Cleaning_Data/Manipulating_Data_with_dplyr/lesson.yaml) Scanner error: while scanning a
```

tag at line 205, column 9 did not find expected whitespace or line break at line 205, column 19 | Leaving swirl now. Type swirl() to resume.

>path2csv

[1] "C:/Users/HOME/Documents/R/win-library/4.0/swirl/Courses/Getting_and_Cleaning_Data/Manipulating_Data_with_dplyr/2014-07-08.csv"

| That's not exactly what I'm looking for. Try again. Or, type info() for more options.

| Store the result of read.csv(path2csv, stringsAsFactors = FALSE) in a new variable called mydf.

> read.csv(path2csv, stringsAsFactors = FALSE)

	X	date	time	size	r_version	r_arch	r_os	package
1	1	2014-07-08	00:54:41	80589	3.1.0	x86_64	mingw32	htmltools
2	2	2014-07-08	00:59:53	321767	3.1.0	x86_64	mingw32	tseries
3	3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party
4	4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc
5	5	2014-07-08	00:46:50	79825	3.0.2	x86_64	linux-gnu	digest
6	6	2014-07-08	00:48:04	77681	3.1.0	x86_64	linux-gnu	randomForest
7	7	2014-07-08	00:48:35	393754	3.1.0	x86_64	linux-gnu	plyr
8	8	2014-07-08	00:47:30	28216	3.0.2	x86_64	linux-gnu	whisker
9	9	2014-07-08	00:54:58	5928	<NA>	<NA>	<NA>	Rcpp
10	10	2014-07-08	00:15:35	2206029	3.0.2	x86_64	linux-gnu	hflights
11	11	2014-07-08	00:15:25	526858	3.0.2	x86_64	linux-gnu	LPCM

| Store the result of read.csv(path2csv, stringsAsFactors = FALSE) in a new variable called mydf.

>mydf<- read.csv(path2csv,stringsAsFactors=FALSE)

| Your dedication is inspiring!

|===== | 7%

| Use dim() to look at the dimensions of mydf.

>?stringasfactors

No documentation for 'stringasfactors' in specified packages and libraries:

you could try '??stringasfactors'

> ?read.csv

> dim(mydf)

[1] 225468 11

| That's a job well done!

|====== | 8%

| Now use head() to preview the data.

> head(mydf)

```
  X       date     time   size r_version r_arch      r_os      package version
1 1 2014-07-08 00:54:41  80589     3.1.0 x86_64   mingw32    htmltools   0.2.4
2 2 2014-07-08 00:59:53 321767     3.1.0 x86_64   mingw32      tseries 0.10-32
3 3 2014-07-08 00:47:13 748063     3.1.0 x86_64 linux-gnu        party  1.0-15
4 4 2014-07-08 00:48:05 606104     3.1.0 x86_64 linux-gnu        Hmisc  3.14-4
5 5 2014-07-08 00:46:50  79825     3.0.2 x86_64 linux-gnu       digest   0.6.4
6 6 2014-07-08 00:48:04  77681     3.1.0 x86_64 linux-gnu randomForest   4.6-7
  country ip_id
1      US     1
2      US     2
3      US     3
4      US     3
5      CA     4
6      US     3
```

| All that hard work is paying off!

|======= | 10%

| The dplyr package was automatically installed (if necessary) and loaded at the

| beginning of this lesson. Normally, this is something you would have to do on

| your own. Just to build the habit, type library(dplyr) now to load the package

| again

> library(dplyr)

| Keep working like that and you'll get there!

|========= | 12%

| It's important that you have dplyr version 0.4.0 or later. To confirm this, type

| packageVersion("dplyr").

>packageVersion("dplyr")

[1] ‘1.0.7’

| That's correct!

|========== | 13%

| If your dplyr version is not at least 0.4.0, then you should hit the Esc key now,

| reinstall dplyr, then resume this lesson where you left off.

...

|=========== | 15%

| The first step of working with data in dplyr is to load the data into what the

| package authors call a 'data frame tbl' or 'tbl_df'. Use the following code to

| create a new tbl_df called cran:

|

| cran<- tbl_df(mydf).

>cran<- tbl_df(mydf)

| That's a job well done!

|========== | 17%

| To avoid confusion and keep things running smoothly, let's remove the original
| data frame from your workspace with rm("mydf").

>rm("mydf")

| Keep working like that and you'll get there!

|========== | 18%

| From ?tbl_df, "The main advantage to using a tbl_df over a regular data frame is
| the printing." Let's see what is meant by this. Type cran to print our tbl_df to
| the console.

>cran

```
# A tibble: 225,468 x 11
       X date    time     size r_version r_arch r_os  package  version country ip_id
   <int> <chr>   <chr>   <int> <chr>     <chr>  <chr> <chr>    <chr>   <chr>   <int>
 1     1 2014-0~ 00:54~ 8.06e4 3.1.0     x86_64 ming~ htmltoo~ 0.2.4   US          1
 2     2 2014-0~ 00:59~ 3.22e5 3.1.0     x86_64 ming~ tseries  0.10-32 US          2
 3     3 2014-0~ 00:47~ 7.48e5 3.1.0     x86_64 linu~ party    1.0-15  US          3
 4     4 2014-0~ 00:48~ 6.06e5 3.1.0     x86_64 linu~ Hmisc    3.14-4  US          3
 5     5 2014-0~ 00:46~ 7.98e4 3.0.2     x86_64 linu~ digest   0.6.4   CA          4
 6     6 2014-0~ 00:48~ 7.77e4 3.1.0     x86_64 linu~ randomF~ 4.6-7   US          3
 7     7 2014-0~ 00:48~ 3.94e5 3.1.0     x86_64 linu~ plyr     1.8.1   US          3
 8     8 2014-0~ 00:47~ 2.82e4 3.0.2     x86_64 linu~ whisker  0.3-2   US          5
 9     9 2014-0~ 00:54~ 5.93e3 NA        NA     NA    Rcpp     0.10.4  CN          6
10    10 2014-0~ 00:15~ 2.21e6 3.0.2     x86_64 linu~ hflights 0.1     US          7
# ... with 225,458 more rows
```

| Nice work!

|========== | 20%

| This output is much more informative and compact than what we would get if we
| printed the original data frame (mydf) to the console.

...

|========== | 22%

| First, we are shown the class and dimensions of the dataset. Just below that, we
| get a preview of the data. Instead of attempting to print the entire dataset,
| dplyr just shows us the first 10 rows of data and only as many columns as fit
| neatly in our console. At the bottom, we see the names and classes for any
| variables that didn't fit on our screen.

...

|========== | 23%

| According to the "Introduction to dplyr" vignette written by the package authors,
| "The dplyr philosophy is to have small functions that each do one thing well."
| Specifically, dplyr supplies five 'verbs' that cover most fundamental data
| manipulation tasks: select(), filter(), arrange(), mutate(), and summarize().

...

|================== | 25%

| Use ?select to pull up the documentation for the first of these core functions.

> ?select

| Great job!

|==================== | 27%

| Help files for the other functions are accessible in the same way.

...

|====================== | 28%

| As may often be the case, particularly with larger datasets, we are only

| interested in some of the variables. Use select(cran, ip_id, package, country) to

| select only the ip_id, package, and country variables from the cran dataset.

> select(cran, ip_id, package, country)

```
# A tibble: 225,468 x 3
   ip_id package      country
   <int> <chr>        <chr>
 1     1 htmltools    US
 2     2 tseries      US
 3     3 party        US
 4     3 Hmisc        US
 5     4 digest       CA
 6     3 randomForest US
 7     3 plyr         US
 8     5 whisker      US
 9     6 Rcpp         CN
10     7 hflights     US
# ... with 225,458 more rows
```

| You are quite good my friend!

|======================== | 30%

| The first thing to notice is that we don't have to type cran$ip_id, cran$package,

| and cran$country, as we normally would when referring to columns of a data frame.

| The select() function knows we are referring to columns of the cran dataset.

...

|========================= | 32%

| Also, note that the columns are returned to us in the order we specified, even

| though ip_id is the rightmost column in the original dataset.

...

|========================== | 33%

| Recall that in R, the `:` operator provides a compact notation for creating a

| sequence of numbers. For example, try 5:20.

> 5:20

[1] 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

| All that hard work is paying off!

|========== | 35%

| Normally, this notation is reserved for numbers, but select() allows you to

| specify a sequence of columns this way, which can save a bunch of typing. Use

| select(cran, r_arch:country) to select all columns starting from r_arch and

| ending with country.

> select(cran, r_arch:country)

```
# A tibble: 225,468 x 5
   r_arch r_os      package      version country
   <chr>  <chr>     <chr>        <chr>   <chr>
 1 x86_64 mingw32   htmltools    0.2.4   US
 2 x86_64 mingw32   tseries      0.10-32 US
 3 x86_64 linux-gnu party        1.0-15  US
 4 x86_64 linux-gnu Hmisc        3.14-4  US
 5 x86_64 linux-gnu digest       0.6.4   CA
 6 x86_64 linux-gnu randomForest 4.6-7   US
 7 x86_64 linux-gnu plyr         1.8.1   US
 8 x86_64 linux-gnu whisker      0.3-2   US
 9 NA     NA        Rcpp         0.10.4  CN
10 x86_64 linux-gnu hflights     0.1     US
# ... with 225,458 more rows
```

| Excellent work!

|========== | 37%

| We can also select the same columns in reverse order. Give it a try.

> select(cran,country:r_arch)

```
# A tibble: 225,468 x 5
   country version package      r_os      r_arch
   <chr>   <chr>   <chr>        <chr>     <chr>
 1 US      0.2.4   htmltools    mingw32   x86_64
 2 US      0.10-32 tseries      mingw32   x86_64
 3 US      1.0-15  party        linux-gnu x86_64
 4 US      3.14-4  Hmisc        linux-gnu x86_64
 5 CA      0.6.4   digest       linux-gnu x86_64
 6 US      4.6-7   randomForest linux-gnu x86_64
 7 US      1.8.1   plyr         linux-gnu x86_64
 8 US      0.3-2   whisker      linux-gnu x86_64
 9 CN      0.10.4  Rcpp         NA        NA
10 US      0.1     hflights     linux-gnu x86_64
# ... with 225,458 more rows
```

| You're the best!

|========== | 38%

| Print the entire dataset again, just to remind yourself of what it looks like.

| You can do this at anytime during the lesson.

>cran

```
# A tibble: 225,468 x 11
       X date    time     size r_version r_arch r_os  package  version country ip_id
   <int> <chr>   <chr>   <int> <chr>     <chr>  <chr> <chr>    <chr>   <chr>   <int>
 1     1 2014-0~ 00:54~ 8.06e4 3.1.0     x86_64 ming~ htmltoo~ 0.2.4   US          1
 2     2 2014-0~ 00:59~ 3.22e5 3.1.0     x86_64 ming~ tseries  0.10-32 US          2
 3     3 2014-0~ 00:47~ 7.48e5 3.1.0     x86_64 linu~ party    1.0-15  US          3
 4     4 2014-0~ 00:48~ 6.06e5 3.1.0     x86_64 linu~ Hmisc    3.14-4  US          3
 5     5 2014-0~ 00:46~ 7.98e4 3.0.2     x86_64 linu~ digest   0.6.4   CA          4
 6     6 2014-0~ 00:48~ 7.77e4 3.1.0     x86_64 linu~ randomF~ 4.6-7   US          3
 7     7 2014-0~ 00:48~ 3.94e5 3.1.0     x86_64 linu~ plyr     1.8.1   US          3
 8     8 2014-0~ 00:47~ 2.82e4 3.0.2     x86_64 linu~ whisker  0.3-2   US          5
 9     9 2014-0~ 00:54~ 5.93e3 NA        NA     NA    Rcpp     0.10.4  CN          6
10    10 2014-0~ 00:15~ 2.21e6 3.0.2     x86_64 linu~ hflights 0.1     US          7
# ... with 225,458 more rows
```

| That's a job well done!

|========== | 40%

| Instead of specifying the columns we want to keep, we can also specify the

| columns we want to throw away. To see how this works, do select(cran, -time) to

| omit the time column.

> **select(cran, -time)**

```
# A tibble: 225,468 x 10
       X date          size r_version r_arch r_os      package  version country ip_id
   <int> <chr>        <int> <chr>     <chr>  <chr>     <chr>    <chr>   <chr>   <int>
 1     1 2014-07-08   80589 3.1.0     x86_64 mingw32   htmlto~ 0.2.4   US          1
 2     2 2014-07-08  321767 3.1.0     x86_64 mingw32   tseries 0.10-32 US          2
 3     3 2014-07-08  748063 3.1.0     x86_64 linux-gnu party   1.0-15  US          3
 4     4 2014-07-08  606104 3.1.0     x86_64 linux-gnu Hmisc   3.14-4  US          3
 5     5 2014-07-08   79825 3.0.2     x86_64 linux-gnu digest  0.6.4   CA          4
 6     6 2014-07-08   77681 3.1.0     x86_64 linux-gnu random~ 4.6-7   US          3
 7     7 2014-07-08  393754 3.1.0     x86_64 linux-gnu plyr    1.8.1   US          3
 8     8 2014-07-08   28216 3.0.2     x86_64 linux-gnu whisker 0.3-2   US          5
 9     9 2014-07-08    5928 NA        NA     NA        Rcpp     0.10.4  CN          6
10    10 2014-07-08 2206029 3.0.2     x86_64 linux-gnu hfligh~ 0.1     US          7
# ... with 225,458 more rows
```

| You nailed it! Good job!

|========== | 42%

| The negative sign in front of time tells select() that we DON'T want the time

| column. Now, let's combine strategies to omit all columns from X through size

| (X:size). To see how this might work, let's look at a numerical example with

| -5:20.

> **-5:20**

[1] -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

| All that hard work is paying off!

|========== | 43%

| Oops! That gaves us a vector of numbers from -5 through 20, which is not what we

| want. Instead, we want to negate the entire sequence of numbers from 5 through

| 20, so that we get -5, -6, -7, ... , -18, -19, -20. Try the same thing, except

| surround 5:20 with parentheses so that R knows we want it to first come up with

| the sequence of numbers, then apply the negative sign to the whole thing.

> -(5:20)

```
 [1]  -5  -6  -7  -8  -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20
```

| That's correct!

```
  |=================================                                  | 45%
```

| Use this knowledge to omit all columns X:size using select().

> select(cran,-(X:size))

```
# A tibble: 225,468 x 7
   r_version r_arch r_os      package      version country ip_id
   <chr>     <chr>  <chr>     <chr>        <chr>   <chr>   <int>
 1 3.1.0     x86_64 mingw32   htmltools    0.2.4   US          1
 2 3.1.0     x86_64 mingw32   tseries      0.10-32 US          2
 3 3.1.0     x86_64 linux-gnu party        1.0-15  US          3
 4 3.1.0     x86_64 linux-gnu Hmisc        3.14-4  US          3
 5 3.0.2     x86_64 linux-gnu digest       0.6.4   CA          4
 6 3.1.0     x86_64 linux-gnu randomForest 4.6-7   US          3
 7 3.1.0     x86_64 linux-gnu plyr         1.8.1   US          3
 8 3.0.2     x86_64 linux-gnu whisker      0.3-2   US          5
 9 NA        NA     NA        Rcpp         0.10.4  CN          6
10 3.0.2     x86_64 linux-gnu hflights     0.1     US          7
# ... with 225,458 more rows
```

| You are doing so well!

```
  |==================================                                 | 47%
```

| Now that you know how to select a subset of columns using select(), a natural

| next question is "How do I select a subset of rows?" That's where the filter()

| function comes in.

...

```
  |===================================                                | 48%
```

| Use filter(cran, package == "swirl") to select all rows for which the package

| variable is equal to "swirl". Be sure to use two equals signs side-by-side!

> filter(cran, package == "swirl")

```
# A tibble: 820 x 11
      X date    time     size r_version r_arch r_os   package version country ip_id
  <int> <chr>   <chr>   <int> <chr>     <chr>  <chr>  <chr>   <chr>   <chr>   <int>
1    27 2014-0~ 00:17~ 105350 3.0.2     x86_64 mingw~ swirl   2.2.9   US         20
2   156 2014-0~ 00:22~  41261 3.1.0     x86_64 linux~ swirl   2.2.9   US         66
3   358 2014-0~ 00:13~ 105335 2.15.2    x86_64 mingw~ swirl   2.2.9   CA        115
4   593 2014-0~ 00:59~ 105465 3.1.0     x86_64 darwi~ swirl   2.2.9   MX        162
5   831 2014-0~ 00:55~ 105335 3.0.3     x86_64 mingw~ swirl   2.2.9   US         57
6   997 2014-0~ 00:33~  41261 3.1.0     x86_64 mingw~ swirl   2.2.9   US         70
7  1023 2014-0~ 00:35~ 106393 3.1.0     x86_64 mingw~ swirl   2.2.9   BR        248
8  1144 2014-0~ 00:00~ 106534 3.0.2     x86_64 linux~ swirl   2.2.9   US        261
9  1402 2014-0~ 00:41~  41261 3.1.0     i386   mingw~ swirl   2.2.9   US        234
10 1424 2014-0~ 00:44~ 106393 3.1.0     x86_64 linux~ swirl   2.2.9   US        301
# ... with 810 more rows
```

| Your dedication is inspiring!

```
  |====================================                               | 50%
```

| Again, note that filter() recognizes 'package' as a column of cran, without you

| having to explicitly specify cran$package.

...

|========== | 52%

| The == operator asks whether the thing on the left is equal to the thing on the
| right. If yes, then it returns TRUE. If no, then FALSE. In this case, package is
| an entire vector (column) of values, so package == "swirl" returns a vector of
| TRUEs and FALSEs. filter() then returns only the rows of cran corresponding to
| the TRUEs.

...

|========== | 53%

| You can specify as many conditions as you want, separated by commas. For example
| filter(cran, r_version == "3.1.1", country == "US") will return all rows of cran
| corresponding to downloads from users in the US running R version 3.1.1. Try it
| out.

> filter(cran, r_version == "3.1.1", country == "US")

```
# A tibble: 1,588 x 11
      X date   time      size r_version r_arch r_os   package version country ip_id
  <int> <chr>  <chr>    <int> <chr>     <chr>  <chr>  <chr>   <chr>   <chr>   <int>
1  2216 2014-~ 00:48~  385112 3.1.1     x86_64 darwi~ colors~ 1.2-4   US        191
2 17332 2014-~ 03:39~  197459 3.1.1     x86_64 darwi~ httr    0.3     US       1704
3 17465 2014-~ 03:25~   23259 3.1.1     x86_64 darwi~ snow    0.3-13  US         62
4 18844 2014-~ 03:59~  190594 3.1.1     x86_64 darwi~ maxLik  1.2-0   US       1533
5 30182 2014-~ 04:13~   77683 3.1.1     i386   mingw~ random~ 4.6-7   US        646
6 30193 2014-~ 04:06~ 2351969 3.1.1     i386   mingw~ ggplot2 1.0.0   US          8
7 30195 2014-~ 04:07~  299080 3.1.1     i386   mingw~ fExtre~ 3010.81 US       2010
8 30217 2014-~ 04:32~  568036 3.1.1     i386   mingw~ rJava   0.9-6   US         98
9 30245 2014-~ 04:10~  526858 3.1.1     i386   mingw~ LPCM    0.44-8  US          8
10 30354 2014-~ 04:32~ 1763717 3.1.1     i386   mingw~ mgcv    1.8-1   US       2122
# ... with 1,578 more rows
```

| Nice work!

|========== | 55%

| The conditions passed to filter() can make use of any of the standard comparison
| operators. Pull up the relevant documentation with ?Comparison (that's an
| uppercase C).

> ?Comparison

| Perseverance, that's the answer.

|========== | 57%

| Edit your previous call to filter() to instead return rows corresponding to users
| in "IN" (India) running an R version that is less than or equal to "3.0.2". The
| up arrow on your keyboard may come in handy here. Don't forget your double
| quotes!

> filter(cran, r_version == "3.0.2", country == "IN")

```
# A tibble: 2,915 x 11
       X date    time     size r_version r_arch r_os   package version country ip_id
   <int> <chr>   <chr>   <int> <chr>     <chr>  <chr>  <chr>   <chr>   <chr>   <int>
 1  9990 2014-0~ 02:11~ 3.97e5 3.0.2     x86_64 linux~ equate~ 1.1     IN       1054
 2  9991 2014-0~ 02:11~ 1.19e5 3.0.2     x86_64 linux~ ggdend~ 0.1-14  IN       1054
 3  9992 2014-0~ 02:11~ 8.18e4 3.0.2     x86_64 linux~ dfcrm   0.2-2   IN       1054
 4 10189 2014-0~ 02:38~ 9.09e5 3.0.2     x86_64 linux~ editru~ 2.7.2   IN       1054
 5 10199 2014-0~ 02:38~ 1.78e5 3.0.2     x86_64 linux~ energy  1.6.1   IN       1054
 6 10200 2014-0~ 02:38~ 5.18e4 3.0.2     x86_64 linux~ ENmisc  1.2-7   IN       1054
 7 10201 2014-0~ 02:38~ 6.52e4 3.0.2     x86_64 linux~ entropy 1.2.0   IN       1054
 8 10202 2014-0~ 02:38~ 1.64e6 3.0.2     x86_64 linux~ Epi     1.1.65  IN       1054
 9 10203 2014-0~ 02:38~ 4.55e5 3.0.2     i686   linux~ epicalc 2.15.1~ IN       1054
10 10205 2014-0~ 02:38~ 2.29e5 3.0.2     x86_64 linux~ epitoo~ 0.5-7   IN       1054
# ... with 2,905 more rows
```

| You are doing so well!

|=== | 58%

| Our last two calls to filter() requested all rows for which some condition AND

| another condition were TRUE. We can also request rows for which EITHER one

| condition OR another condition are TRUE. For example, filter(cran, country ==

| "US" | country == "IN") will gives us all rows for which the country variable

| equals either "US" or "IN". Give it a go.

> filter(cran, country =="US" | country == "IN")

```
# A tibble: 95,283 x 11
       X date    time     size r_version r_arch r_os  package  version country ip_id
   <int> <chr>   <chr>   <int> <chr>     <chr>  <chr> <chr>    <chr>   <chr>   <int>
 1     1 2014-0~ 00:54~ 8.06e4 3.1.0     x86_64 ming~ htmltoo~ 0.2.4   US          1
 2     2 2014-0~ 00:59~ 3.22e5 3.1.0     x86_64 ming~ tseries  0.10-32 US          2
 3     3 2014-0~ 00:47~ 7.48e5 3.1.0     x86_64 linu~ party    1.0-15  US          3
 4     4 2014-0~ 00:48~ 6.06e5 3.1.0     x86_64 linu~ Hmisc    3.14-4  US          3
 5     6 2014-0~ 00:48~ 7.77e4 3.1.0     x86_64 linu~ randomF~ 4.6-7   US          3
 6     7 2014-0~ 00:48~ 3.94e5 3.1.0     x86_64 linu~ plyr     1.8.1   US          3
 7     8 2014-0~ 00:47~ 2.82e4 3.0.2     x86_64 linu~ whisker  0.3-2   US          5
 8    10 2014-0~ 00:15~ 2.21e6 3.0.2     x86_64 linu~ hflights 0.1     US          7
 9    11 2014-0~ 00:15~ 5.27e5 3.0.2     x86_64 linu~ LPCM     0.44-8  US          8
10    12 2014-0~ 00:14~ 2.35e6 2.14.1    x86_64 linu~ ggplot2  1.0.0   US          8
# ... with 95,273 more rows
```

| All that practice is paying off!

|== | 60%

| Now, use filter() to fetch all rows for which size is strictly greater than (>)

| 100500 (no quotes, since size is numeric) AND r_os equals "linux-gnu". Hint: You

| are passing three arguments to filter(): the name of the dataset, the first

| condition, and the second condition.

> filter(cran, size>100500, r_os == "linux-gnu")

```
# A tibble: 33,683 x 11
       X date    time     size r_version r_arch r_os   package version country ip_id
   <int> <chr>   <chr>   <int> <chr>     <chr>  <chr>  <chr>   <chr>   <chr>   <int>
 1     3 2014-0~ 00:47~ 7.48e5 3.1.0     x86_64 linux~ party   1.0-15  US          3
 2     4 2014-0~ 00:48~ 6.06e5 3.1.0     x86_64 linux~ Hmisc   3.14-4  US          3
 3     7 2014-0~ 00:48~ 3.94e5 3.1.0     x86_64 linux~ plyr    1.8.1   US          3
 4    10 2014-0~ 00:15~ 2.21e6 3.0.2     x86_64 linux~ hfligh~ 0.1     US          7
 5    11 2014-0~ 00:15~ 5.27e5 3.0.2     x86_64 linux~ LPCM    0.44-8  US          8
 6    12 2014-0~ 00:14~ 2.35e6 2.14.1    x86_64 linux~ ggplot2 1.0.0   US          8
 7    14 2014-0~ 00:15~ 3.10e6 3.0.2     x86_64 linux~ Rcpp    0.9.7   VE         10
 8    15 2014-0~ 00:14~ 5.68e5 3.1.0     x86_64 linux~ rJava   0.9-6   US         11
 9    16 2014-0~ 00:15~ 1.60e6 3.1.0     x86_64 linux~ RSQLite 0.11.4  US          7
10    18 2014-0~ 00:26~ 1.87e5 3.1.0     x86_64 linux~ ipred   0.9-3   DE         13
# ... with 33,673 more rows
```

| That's a job well done!

|== | 62%

| Finally, we want to get only the rows for which the r_version is not missing. R
| represents missing values with NA and these missing values can be detected using
| the is.na() function.

...

|=== | 63%

| To see how this works, try is.na(c(3, 5, NA, 10)).

> is.na(c(3, 5, NA, 10))

[1] FALSE FALSE TRUE FALSE

| Perseverance, that's the answer.

|== | 65%

| Now, put an exclamation point (!) before is.na() to change all of the TRUEs to
| FALSEs and all of the FALSEs to TRUEs, thus telling us what is NOT NA:
| !is.na(c(3, 5, NA, 10)).

> !is.na(c(3, 5, NA, 10))

[1] TRUE TRUE FALSE TRUE

| Nice work!

|=== | 67%

| Okay, ready to put all of this together? Use filter() to return all rows of cran
| for which r_version is NOT NA. Hint: You will need to use !is.na() as part of
| your second argument to filter().

> filter(cran, !is.na(r_version))

```
# A tibble: 207,205 x 11
       X date    time     size r_version r_arch r_os  package  version country ip_id
   <int> <chr>   <chr>   <int> <chr>     <chr>  <chr> <chr>    <chr>   <chr>   <int>
 1     1 2014-0~ 00:54~ 8.06e4 3.1.0     x86_64 ming~ htmltoo~ 0.2.4   US          1
 2     2 2014-0~ 00:59~ 3.22e5 3.1.0     x86_64 ming~ tseries  0.10-32 US          2
 3     3 2014-0~ 00:47~ 7.48e5 3.1.0     x86_64 linu~ party    1.0-15  US          3
 4     4 2014-0~ 00:48~ 6.06e5 3.1.0     x86_64 linu~ Hmisc    3.14-4  US          3
 5     5 2014-0~ 00:46~ 7.98e4 3.0.2     x86_64 linu~ digest   0.6.4   CA          4
 6     6 2014-0~ 00:48~ 7.77e4 3.1.0     x86_64 linu~ randomF~ 4.6-7   US          3
 7     7 2014-0~ 00:48~ 3.94e5 3.1.0     x86_64 linu~ plyr     1.8.1   US          3
 8     8 2014-0~ 00:47~ 2.82e4 3.0.2     x86_64 linu~ whisker  0.3-2   US          5
 9    10 2014-0~ 00:15~ 2.21e6 3.0.2     x86_64 linu~ hflights 0.1     US          7
10    11 2014-0~ 00:15~ 5.27e5 3.0.2     x86_64 linu~ LPCM     0.44-8  US          8
# ... with 207,195 more rows
```

| Excellent job!

|== | 68%

| We've seen how to select a subset of columns and rows from our dataset using
| select() and filter(), respectively. Inherent in select() was also the ability to
| arrange our selected columns in any order we please.

...

|=== | 70%

| Sometimes we want to order the rows of a dataset according to the values of a

| particular variable. This is the job of arrange().

...

|== | 72%

| To see how arrange() works, let's first take a subset of cran. select() all

| columns from size through ip_id and store the result in cran2.

> select(cran, size:ip_id)

```
# A tibble: 225,468 x 8
      size r_version r_arch r_os      package      version country ip_id
     <int> <chr>     <chr>  <chr>     <chr>        <chr>   <chr>   <int>
 1   80589 3.1.0     x86_64 mingw32   htmltools    0.2.4   US          1
 2  321767 3.1.0     x86_64 mingw32   tseries      0.10-32 US          2
 3  748063 3.1.0     x86_64 linux-gnu party        1.0-15  US          3
 4  606104 3.1.0     x86_64 linux-gnu Hmisc        3.14-4  US          3
 5   79825 3.0.2     x86_64 linux-gnu digest       0.6.4   CA          4
 6   77681 3.1.0     x86_64 linux-gnu randomForest 4.6-7   US          3
 7  393754 3.1.0     x86_64 linux-gnu plyr         1.8.1   US          3
 8   28216 3.0.2     x86_64 linux-gnu whisker      0.3-2   US          5
 9    5928 NA        NA     NA        Rcpp         0.10.4  CN          6
10 2206029 3.0.2     x86_64 linux-gnu hflights     0.1     US          7
# ... with 225,458 more rows
```

| Not quite! Try again. Or, type info() for more options.

| Use select(cran, size:ip_id) to select all columns from size through ip_id. Store

| the result in cran2.

> cran2 <- select(cran,size:ip_id)

| Perseverance, that's the answer.

|== | 73%

| Now, to order the ROWS of cran2 so that ip_id is in ascending order (from small

| to large), type arrange(cran2, ip_id). You may want to make your console wide

| enough so that you can see ip_id, which is the last column.

> arrange(cran2,ip_id)

```
# A tibble: 225,468 x 8
     size r_version r_arch r_os         package     version country ip_id
    <int> <chr>     <chr>  <chr>        <chr>       <chr>   <chr>   <int>
 1  80589 3.1.0     x86_64 mingw32      htmltools   0.2.4   US          1
 2 180562 3.0.2     x86_64 mingw32      yaml        2.1.13  US          1
 3 190120 3.1.0     i386   mingw32      babel       0.2-6   US          1
 4 321767 3.1.0     x86_64 mingw32      tseries     0.10-32 US          2
 5  52281 3.0.3     x86_64 darwin10.8.0 quadprog    1.5-5   US          2
 6 876702 3.1.0     x86_64 linux-gnu    zoo         1.7-11  US          2
 7 321764 3.0.2     x86_64 linux-gnu    tseries     0.10-32 US          2
 8 876702 3.1.0     x86_64 linux-gnu    zoo         1.7-11  US          2
 9 321768 3.1.0     x86_64 mingw32      tseries     0.10-32 US          2
10 784093 3.1.0     x86_64 linux-gnu    strucchange 1.5-0   US          2
# ... with 225,458 more rows
```

| Excellent work!

|=== |75%

| To do the same, but in descending order, change the second argument to

| desc(ip_id), where desc() stands for 'descending'. Go ahead.

> arrange(cran2,desc(ip_id))

```
# A tibble: 225,468 x 8
      size r_version r_arch r_os         package      version country ip_id
     <int> <chr>     <chr>  <chr>        <chr>        <chr>   <chr>   <int>
 1    5933 NA        NA     NA           CPE          1.4.2   CN      13859
 2  569241 3.1.0     x86_64 mingw32      multcompView 0.1-5   US      13858
 3  228444 3.1.0     x86_64 mingw32      tourr        0.5.3   NZ      13857
 4  308962 3.1.0     x86_64 darwin13.1.0 ctv          0.7-9   CN      13856
 5  950964 3.0.3     i386   mingw32      knitr        1.6     CA      13855
 6   80185 3.0.3     i386   mingw32      htmltools    0.2.4   CA      13855
 7 1431750 3.0.3     i386   mingw32      shiny        0.10.0  CA      13855
 8 2189695 3.1.0     x86_64 mingw32      RMySQL       0.9-3   US      13854
 9 4818024 3.1.0     i386   mingw32      igraph       0.7.1   US      13853
10  197495 3.1.0     x86_64 mingw32      coda         0.16-1  US      13852
# ... with 225,458 more rows
```

| Your dedication is inspiring!

|== 77%

| We can also arrange the data according to the values of multiple variables. For

| example, arrange(cran2, package, ip_id) will first arrange by package names

| (ascending alphabetically), then by ip_id. This means that if there are multiple

| rows with the same value for package, they will be sorted by ip_id (ascending

| numerically). Try arrange(cran2, package, ip_id) now.

> arrange(cran2, package, ip_id)

```
# A tibble: 225,468 x 8
    size r_version r_arch r_os         package version country ip_id
   <int> <chr>     <chr>  <chr>        <chr>   <chr>   <chr>   <int>
 1 71677 3.0.3     x86_64 darwin10.8.0 A3      0.9.2   CN       1003
 2 71672 3.1.0     x86_64 linux-gnu    A3      0.9.2   US       1015
 3 71677 3.1.0     x86_64 mingw32      A3      0.9.2   IN       1054
 4 70438 3.0.1     x86_64 darwin10.8.0 A3      0.9.2   CN       1513
 5 71677 NA        NA     NA           A3      0.9.2   BR       1526
 6 71892 3.0.2     x86_64 linux-gnu    A3      0.9.2   IN       1542
 7 71677 3.1.0     x86_64 linux-gnu    A3      0.9.2   ZA       2925
 8 71672 3.1.0     x86_64 mingw32      A3      0.9.2   IL       3889
 9 71677 3.0.3     x86_64 mingw32      A3      0.9.2   DE       3917
10 71672 3.1.0     x86_64 mingw32      A3      0.9.2   US       4219
# ... with 225,458 more rows
```

| You got it right!

|== 78%

| Arrange cran2 by the following three variables, in this order: country

| (ascending), r_version (descending), and ip_id (ascending).

> arrange(cran2,country,desc(r_version),ip_id)

```
# A tibble: 225,468 x 8
      size r_version r_arch r_os      package       version   country ip_id
     <int> <chr>     <chr>  <chr>     <chr>         <chr>     <chr>   <int>
 1 1556858 3.1.1     i386   mingw32   RcppArmadillo 0.4.320.0 A1       2843
 2 1823512 3.1.0     x86_64 linux-gnu mgcv          1.8-1     A1       2843
 3   15732 3.1.0     i686   linux-gnu grnn          0.1.0     A1       3146
 4 3014840 3.1.0     x86_64 mingw32   Rcpp          0.11.2    A1       3146
 5  660087 3.1.0     i386   mingw32   xts           0.9-7     A1       3146
 6  522261 3.1.0     i386   mingw32   FNN           1.1       A1       3146
 7  522263 3.1.0     i386   mingw32   FNN           1.1       A1       3146
 8 1676627 3.1.0     x86_64 linux-gnu rgeos         0.3-5     A1       3146
 9 2118530 3.1.0     x86_64 linux-gnu spacetime     1.1-0     A1       3146
10 2217180 3.1.0     x86_64 mingw32   gstat         1.0-19    A1       3146
# ... with 225,458 more rows
```

| Keep working like that and you'll get there!

|== 80%

| To illustrate the next major function in dplyr, let's take another subset of our

| original data. Use select() to grab 3 columns from cran -- ip_id, package, and

| size (in that order) -- and store the result in a new variable called cran3.

> select(cran,ip_id,package,size)

```
# A tibble: 225,468 x 3
   ip_id package      size
   <int> <chr>       <int>
 1     1 htmltools   80589
 2     2 tseries    321767
 3     3 party      748063
 4     3 Hmisc      606104
 5     4 digest      79825
 6     3 randomForest 77681
 7     3 plyr       393754
 8     5 whisker     28216
 9     6 Rcpp         5928
10     7 hflights  2206029
# ... with 225,458 more rows
```

| Not quite, but you're learning! Try again. Or, type info() for more options.

| cran3 <- select(cran, ip_id, package, size) will store just these three columnsin a new variable called cran3.

> cran3 <- select(cran, ip_id, package, size)

| That's correct!

|== 82%

| Take a look at cran3 now.

> cran3

```
# A tibble: 225,468 x 3
   ip_id package      size
   <int> <chr>       <int>
 1     1 htmltools   80589
 2     2 tseries    321767
 3     3 party      748063
 4     3 Hmisc      606104
 5     4 digest      79825
 6     3 randomForest 77681
 7     3 plyr       393754
 8     5 whisker     28216
 9     6 Rcpp         5928
10     7 hflights  2206029
# ... with 225,458 more rows
```

| Excellent work!

|== 83%

| It's common to create a new variable based on the value of one or more variables

| already in a dataset. The mutate() function does exactly this.

...

|== 85%

| The size variable represents the download size in bytes, which are units of

| computer memory. These days, megabytes (MB) are a more common unit of

| measurement. One megabyte is equal to 2^20 bytes. That's 2 to the power of 20,

| which is approximately one million bytes!

...

|== 87%

| We want to add a column called size_mb that contains the download size in

| megabytes. Here's the code to do it:

|

| mutate(cran3, size_mb = size / 2^20)

> mutate(cran3, size_mb = size / 2^20)

```
# A tibble: 225,468 x 4
   ip_id package         size size_mb
   <int> <chr>          <int>   <dbl>
 1     1 htmltools      80589 0.0769
 2     2 tseries       321767 0.307
 3     3 party         748063 0.713
 4     3 Hmisc         606104 0.578
 5     4 digest         79825 0.0761
 6     3 randomForest   77681 0.0741
 7     3 plyr          393754 0.376
 8     5 whisker        28216 0.0269
 9     6 Rcpp            5928 0.00565
10     7 hflights     2206029 2.10
# ... with 225,458 more rows
```

| Perseverance, that's the answer.

|==88%

| An even larger unit of memory is a gigabyte (GB), which equals 2^10 megabytes. We

| might as well add another column for download size in gigabytes!

...

|==90%

| One very nice feature of mutate() is that you can use the value computed for your

| second column (size_mb) to create a third column, all in the same line of code.

| To see this in action, repeat the exact same command as above, except add a third

| argument creating a column that is named size_gb and equal to size_mb / 2^10.

> mutate(cran3,size_gb=size/2^10)

```
# A tibble: 225,468 x 4
   ip_id package         size size_gb
   <int> <chr>          <int>   <dbl>
 1     1 htmltools      80589   78.7
 2     2 tseries       321767  314.
 3     3 party         748063  731.
 4     3 Hmisc         606104  592.
 5     4 digest         79825   78.0
 6     3 randomForest   77681   75.9
 7     3 plyr          393754  385.
 8     5 whisker        28216   27.6
 9     6 Rcpp            5928    5.79
10     7 hflights     2206029 2154.
# ... with 225,458 more rows
```

| Not quite, but you're learning! Try again. Or, type info() for more options.

| Use mutate(cran3, size_mb = size / 2^20, size_gb = size_mb / 2^10) to create two

| new columns called size_mb and size_gb, respectively.

> mutate(cran3, size_mb = size / 2^20, size_gb = size_mb / 2^10)

```
# A tibble: 225,468 x 5
   ip_id package         size size_mb    size_gb
   <int> <chr>          <int>   <dbl>      <dbl>
 1     1 htmltools      80589 0.0769  0.0000751
 2     2 tseries       321767 0.307   0.000300
 3     3 party         748063 0.713   0.000697
 4     3 Hmisc         606104 0.578   0.000564
 5     4 digest         79825 0.0761  0.0000743
 6     3 randomForest   77681 0.0741  0.0000723
 7     3 plyr          393754 0.376   0.000367
 8     5 whisker        28216 0.0269  0.0000263
 9     6 Rcpp            5928 0.00565 0.00000552
10     7 hflights     2206029 2.10    0.00205
# ... with 225,458 more rows
```

| You got it!

|==========92%

| Let's try one more for practice. Pretend we discovered a glitch in the system

| that provided the original values for the size variable. All of the values in

| cran3 are 1000 bytes less than they should be. Using cran3, create just one new

| column called correct_size that contains the correct size.

> mutate(cran3,correct_size=size+1000)

```
# A tibble: 225,468 x 5
   ip_id package         size size_mb    size_gb
   <int> <chr>          <int>   <dbl>      <dbl>
 1     1 htmltools      80589 0.0769  0.0000751
 2     2 tseries       321767 0.307   0.000300
 3     3 party         748063 0.713   0.000697
 4     3 Hmisc         606104 0.578   0.000564
 5     4 digest         79825 0.0761  0.0000743
 6     3 randomForest   77681 0.0741  0.0000723
 7     3 plyr          393754 0.376   0.000367
 8     5 whisker        28216 0.0269  0.0000263
 9     6 Rcpp            5928 0.00565 0.00000552
10     7 hflights     2206029 2.10    0.00205
# ... with 225,458 more rows
```

| You nailed it! Good job!

|==========| 93%

| The last of the five core dplyr verbs, summarize(), collapses the dataset to a

| single row. Let's say we're interested in knowing the average download size.

| summarize(cran, avg_bytes = mean(size)) will yield the mean value of the size

| variable. Here we've chosen to label the result 'avg_bytes', but we could have

| named it anything. Give it a try.

> **summarize(cran,avg_bytes=mean(size))**

```
# A tibble: 1 x 1
  avg_bytes
      <dbl>
1   844086.
```

Experiment 4: Visualize all Statistical measures (Mean,Mode, Median, Range, Inter Quartile Range etc., using Histograms, Boxplots and Scatter Plots).

We use the dataset iris. This dataset is imported by default in R

Execution:

>dat<- iris # load the iris dataset and renamed it dat

> View(dat)

> head(dat) # first 6 observations

```
  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
```

>str(dat)

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Mean

> mean(dat$Sepal.Length)

[1] 5.843333

Median

> median(dat$Sepal.Length)

[1] 5.8

Range

The range can then be easily computed, by subtracting the minimum from the maximum:

> min(dat$Sepal.Length)

[1] 4.3

> max(dat$Sepal.Length)

[1] 7.9

> max(dat$Sepal.Length) - min(dat$Sepal.Length)

[1] 3.6

Interquartile range

The interquartile range (i.e., the difference between the first and third quartile) can be computed with the IQR() function

>quantile(dat$Sepal.Length, 0.25)

25%

5.1

>quantile(dat$Sepal.Length, 0.75)

75%

6.4

>quantile(dat$Sepal.Length, 0.75) - quantile(dat$Sepal.Length, 0.25)

75%

1.3

(or)

> IQR(dat$Sepal.Length)

[1] 1.3

Mode

There is no function to find the mode of a variable. We can find it by the functions table() and sort():

> tab <- table(dat$Sepal.Length) # number of occurrences for each unique value

> sort(tab, decreasing = TRUE) # sort highest to lowest

```
  5 5.1 6.3 5.7 6.7 5.5 5.8 6.4 4.9 5.4 5.6   6 6.1 4.8 6.5 4.6 5.2 6.2 6.9 7.7 4.4
 10   9   9   8   8   7   7   7   6   6   6   6   6   5   5   4   4   4   4   4   3
5.9 6.8 7.2 4.7 6.6 4.3 4.5 5.3   7 7.1 7.3 7.4 7.6 7.9
  3   3   3   2   2   1   1   1   1   1   1   1   1   1
```

The mode of the variable Sepal.Length is thus 5.

To find the mode can also be applied to qualitative variables such as Species:

> sort(table(dat$Species), decreasing = TRUE)

(or)

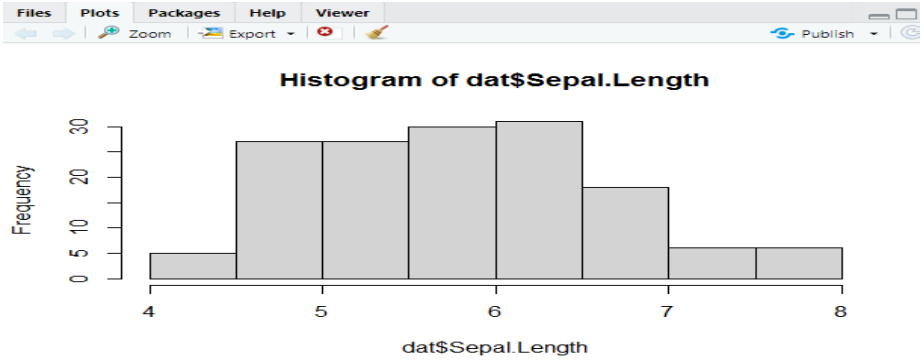
>summary(dat$Species)

```
setosa versicolor  virginica
    50         50         50
```

Histogram

A histogram gives an idea about the distribution of a quantitative variable. The idea is to break the range of values into intervals and count how many observations fall into each interval. To draw a histogram in R, use hist():

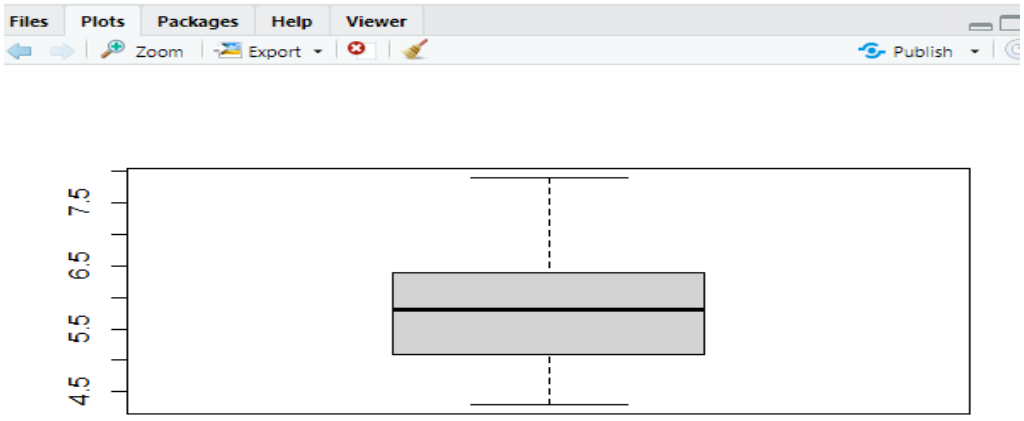
>hist(dat$Sepal.Length)



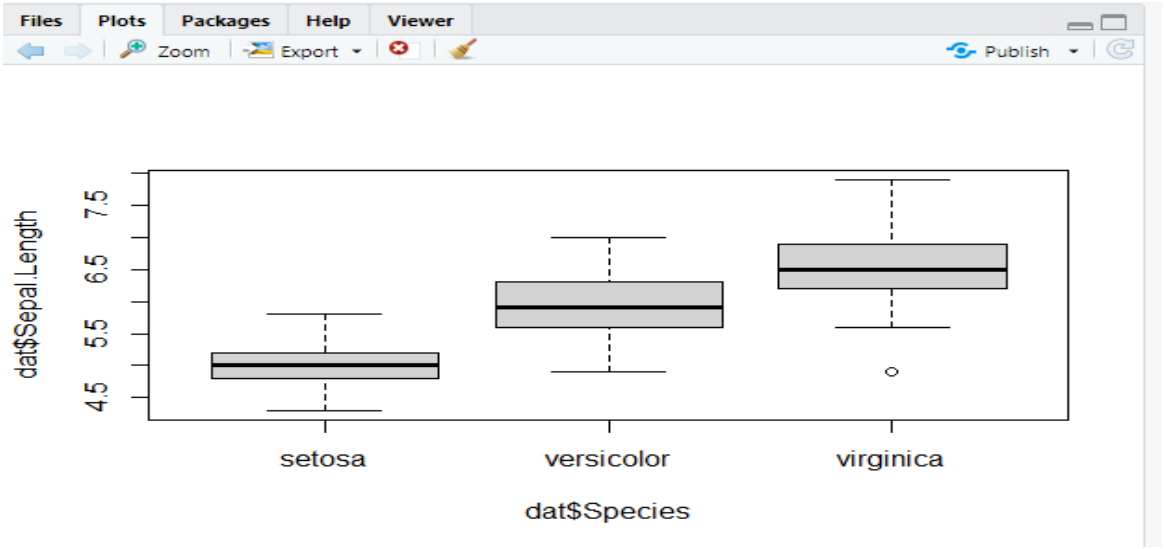
Boxplot

A boxplot graphically represents the distribution of a quantitative variable by visually displaying five common location summary (minimum, median, first/third quartiles and maximum) and any observation that was classified as a suspected outlier using the interquartile range (IQR) criterion.

> boxplot(dat$Sepal.Length)



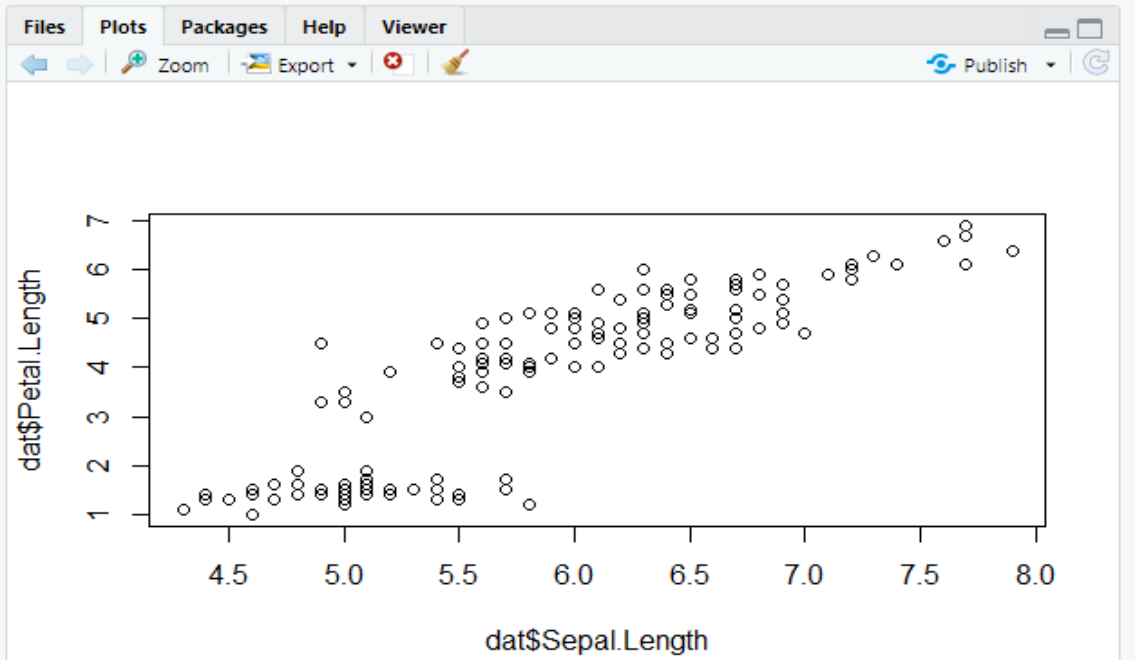
Boxplots can present side-by-side for comparing and contrasting distributions from two or more groups. For instance, we compare the length of the sepal across the different species:



Scatterplot

Scatterplots allow to check whether there is a potential link between two quantitative variables. For this reason, scatterplots are often used to visualize a potential correlation between two variables. For instance, when drawing a scatterplot of the length of the sepal and the length of the petal:

> plot(dat$Sepal.Length, dat$Petal.Length)


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
dat$Petal.Length
1
2
3
4
5
6
7
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
dat$Sepal.Length

Experiment 5: Create a data frame with the following structure.

EMP ID	EMP NAME	SALARY	START DATE
1	Satish	5000	01-11-2013
2	Vani	7500	05-06-2011
3	Ramesh	10000	21-09-1999
4	Praveen	9500	13-09-2005
5	Pallavi	4500	23-10-2000

Creation of table:

```
>employee<-
matrix(c(1,2,3,4,5,'Satish','Vani','Ramesh','Praveen','Pallavi',5000,7500,10000,9500,4500,"01-11-
2013","05-06-2011","21-09-1999","13-09-2005","23-10-2000"), ncol=4)
>colnames(employee) <- c('EMPID', 'EMPNAME','SALARY','STARTDATE')
```

cran | cran2 | cran3 | dat | employee

Filter

	EMPID	EMPNAME	SALARY	STARTDATE
1	1	Satish	5000	01-11-2013
2	2	Vani	7500	05-06-2011
3	3	Ramesh	10000	21-09-1999
4	4	Praveen	9500	13-09-2005
5	5	Pallavi	4500	23-10-2000

a. Extract two column names using column name.

```
>colnames(employee)[1:2]
[1] "EMPID"   "EMPNAME"
```

b. Extract the first two rows and then all columns.

```
> employee[1:2,]
     EMPID EMPNAME  SALARY STARTDATE
[1,] "1"       "Satish"     "5000" "01-11-2013"
[2,] "2"      "Vani"         "7500" "05-06-2011"
```

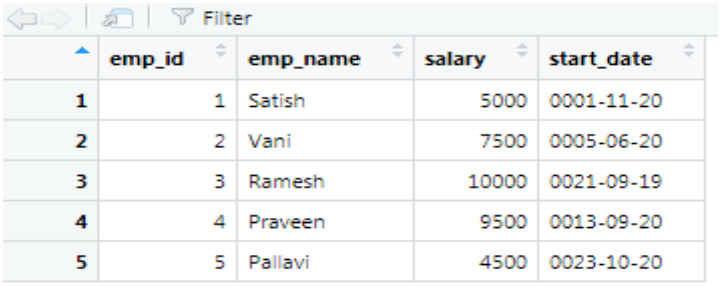
c. Extract 3rd and 5th row with 2nd and 4th column.

```
> employee[c(3,5),c(2,4)]
   EMPNAME   STARTDATE
[1,] "Ramesh"  "21-09-1999"
[2,] "Pallavi" "23-10-2000
```

Creation of data frame:

```
>emp.data<- data.frame(
+     emp_id = c (1:5),
+     emp_name = c("Satish","Vani","Ramesh","Praveen","Pallavi"),
+     salary = c(5000,7500,10000,9500,4500),
+     start_date = as.Date(c("01-11-2013", "05-06-2011", "21-09-1999", "13-09-2005",
+                            "23-10-2000")),
+     stringsAsFactors = FALSE
+ )
> # Print the data frame.
```

> print(emp.data)
Output:

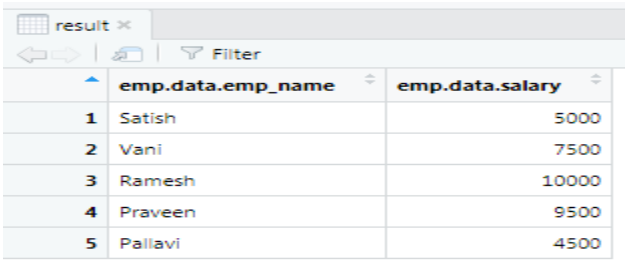


	emp_id	emp_name	salary	start_date
1	1	Satish	5000	0001-11-20
2	2	Vani	7500	0005-06-20
3	3	Ramesh	10000	0021-09-19
4	4	Praveen	9500	0013-09-20
5	5	Pallavi	4500	0023-10-20

>str(emp.data)
'data.frame': 5 obs. of 4 variables:
$ emp_id : int 1 2 3 4 5
$ emp_name : chr "Satish" "Vani" "Ramesh" "Praveen" ...
$ salary : num 5000 7500 10000 9500 4500
$ start_date: Date, format: "0001-11-20" "0005-06-20" ...

a. Extract two column names using column name.

> result <- data.frame(emp.data$emp_name,emp.data$salary)
> print(result)

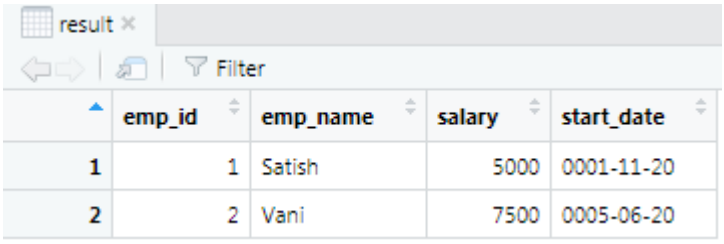


	emp.data.emp_name	emp.data.salary
1	Satish	5000
2	Vani	7500
3	Ramesh	10000
4	Praveen	9500
5	Pallavi	4500

b. Extract the first two rows and then all columns.

> result <- emp.data[1:2,]

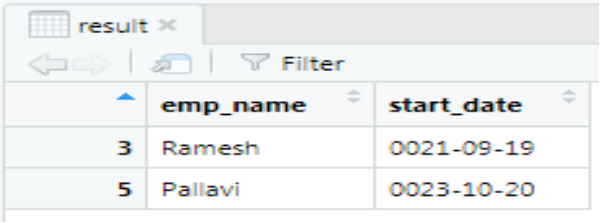
> print(result)



	emp_id	emp_name	salary	start_date
1	1	Satish	5000	0001-11-20
2	2	Vani	7500	0005-06-20

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

> print(result)



	emp_name	start_date
3	Ramesh	0021-09-19
5	Pallavi	0023-10-20

Experiment 6: Write R Program using 'apply' group of functions to create and apply normalization function on each of the numeric variables/columns of iris dataset to transform them into i. 0 to 1 range with min-max normalization. ii. a value around 0 with z-score normalization.

Execution:

```
>head(iris)      #view first six rows of iris dataset
```

Output:

```
#  Sepal.LengthSepal.WidthPetal.LengthPetal.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
```

```
>min_max_norm<- function(x) {
    (x - min(x)) / (max(x) - min(x))
 }                #define Min-Max normalization function
>iris_norm<- as.data.frame(lapply(iris[1:4], min_max_norm))#apply Min-Max normalization to first four columns in iris dataset
```

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	0.22222222	0.62500000	0.06779661	0.04166667	setosa
2	0.16666667	0.41666667	0.06779661	0.04166667	setosa
3	0.11111111	0.50000000	0.05084746	0.04166667	setosa
4	0.08333333	0.45833333	0.08474576	0.04166667	setosa
5	0.19444444	0.66666667	0.06779661	0.04166667	setosa
6	0.30555556	0.79166667	0.11864407	0.12500000	setosa
7	0.08333333	0.58333333	0.06779661	0.08333333	setosa
8	0.19444444	0.58333333	0.08474576	0.04166667	setosa
9	0.02777778	0.37500000	0.06779661	0.04166667	setosa
10	0.16666667	0.45833333	0.08474576	0.00000000	setosa
11	0.30555556	0.70833333	0.08474576	0.04166667	setosa

Showing 1 to 12 of 150 entries, 5 total columns

```
>head(iris_norm)      #view first six rows of normalized iris dataset
```

Output:

```
#  Sepal.LengthSepal.WidthPetal.LengthPetal.Width
#1   0.22222222   0.6250000   0.06779661  0.04166667
#2   0.16666667   0.4166667   0.06779661  0.04166667
#3   0.11111111   0.5000000   0.05084746  0.04166667
#4   0.08333333   0.4583333   0.08474576  0.04166667
#5   0.19444444   0.6666667   0.06779661  0.04166667
```

#6 0.30555556 0.7916667 0.11864407 0.12500000

>iris_norm$Species<- iris$Species #add back Species column

>head(iris_norm) #view first six rows of iris_norm

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
#1	0.22222222	0.6250000	0.06779661	0.04166667	setosa
#2	0.16666667	0.4166667	0.06779661	0.04166667	setosa
#3	0.11111111	0.5000000	0.05084746	0.04166667	setosa
#4	0.08333333	0.4583333	0.08474576	0.04166667	setosa
#5	0.19444444	0.6666667	0.06779661	0.04166667	setosa
#6	0.30555556	0.7916667	0.11864407	0.12500000	setosa

Z-Score Standardization

1. Standardize one variable

>iris$Sepal.Width<- (iris$Sepal.Width - mean(iris$Sepal.Width)) / sd(iris$Sepal.Width)#standardize Sepal.Width

>head(iris)

Sepal.LengthSepal.WidthPetal.LengthPetal.Width Species

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
#1	5.1	1.01560199	1.4	0.2	setosa
#2	4.9	-0.13153881	1.4	0.2	setosa
#3	4.7	0.32731751	1.3	0.2	setosa
#4	4.6	0.09788935	1.5	0.2	setosa
#5	5.0	1.24503015	1.4	0.2	setosa
#6	5.4	1.93331463	1.7	0.4	setosa

The values of *Sepal.Width* are now scaled such that the mean is 0 and the standard deviation is 1.

>mean(iris$Sepal.Width) #find mean of Sepal.Width

Output: #basically zero

[1] 2.034094e-16

>sd(iris$Sepal.Width) #find standard deviation of Sepal.Width

Output:

[1] 1

2. Standardize several variables using the scale function

```
>iris_standardize<- as.data.frame(scale(iris[1:4]))    #standardize first four columns of iris dataset
>head(iris_standardize)          #view first six rows of standardized dataset
```

Output:

```
#  Sepal.LengthSepal.WidthPetal.LengthPetal.Width
#1  -0.8976739  1.01560199   -1.335752  -1.311052
#2  -1.1392005 -0.13153881   -1.335752  -1.311052
#3  -1.3807271  0.32731751   -1.392399  -1.311052
#4  -1.5014904  0.09788935   -1.279104  -1.311052
#5  -1.0184372  1.24503015   -1.335752  -1.311052
#6  -0.5353840  1.93331463   -1.165809  -1.048667
```

```
>library(dplyr)          #load dplyr package
>iris_new<- iris %>%mutate_each_(list(~scale(.) %>% as.vector),
+  vars = c("Sepal.Width","Sepal.Length"))  #standardize Sepal.Width and Sepal.Length
>head(iris_new)          #view first six rows of new data frame
```

Output:

```
#  Sepal.LengthSepal.WidthPetal.LengthPetal.Width Species
#1  -0.8976739  1.01560199          1.4         0.2  setosa
#2  -1.1392005 -0.13153881          1.4         0.2  setosa
#3  -1.3807271  0.32731751          1.3         0.2  setosa
#4  -1.5014904  0.09788935          1.5         0.2  setosa
#5  -1.0184372  1.24503015          1.4         0.2  setosa
#6  -0.5353840  1.93331463          1.7         0.4  setosa
```

Experiment 7: Create a data frame with 10 observations and 3 variables and add new rows andcolumns to it using 'rbind' and 'cbind' function.

Execution:

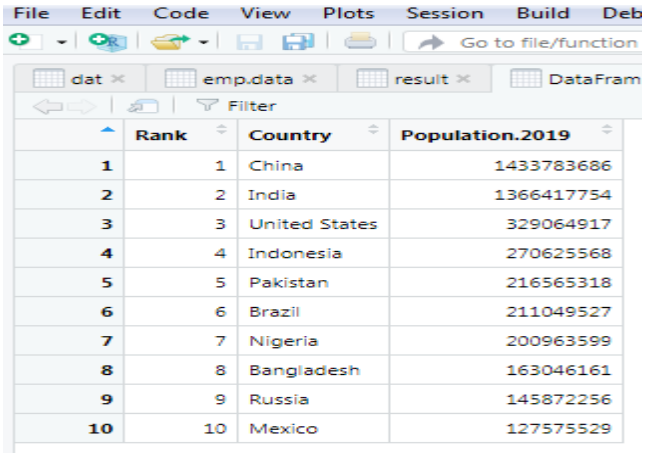
> Rank <- 1:10 #Creating Rank Vector column

>Country <- c("China", "India", "United States", "Indonesia", "Pakistan", "Brazil", "Nigeria", "Bangladesh", "Russia", "Mexico") #Creating Country Vector column

> Population.2019 <- c(1433783686, 1366417754, 329064917, 270625568, 216565318, 211049527, 200963599, 163046161, 145872256, 127575529) #Creating 2019 Population Vector

>DataFrame.WorldPopulation<-data.frame(Rank, Country, Population.2019) #Creating Data Frame using above vectors

Output:



	Rank	Country	Population.2019
1	1	China	1433783686
2	2	India	1366417754
3	3	United States	329064917
4	4	Indonesia	270625568
5	5	Pakistan	216565318
6	6	Brazil	211049527
7	7	Nigeria	200963599
8	8	Bangladesh	163046161
9	9	Russia	145872256
10	10	Mexico	127575529

#Printing Structure / Dimension of DataFrame.WorldPopulation Before Row Addition

>str(DataFrame.WorldPopulation)

Output:

'data.frame': 10 obs. of 3 variables:

$ Rank : int 1 2 3 4 5 6 7 8 9 10

$ Country : chr "China" "India" "United States" "Indonesia" ...

$ Population.2019: num 1.43e+09 1.37e+09 3.29e+08 2.71e+08 2.17e+08 ...

Adding Observation/Row to Data Frame

- To add or insert observation/row to an existing Data Frame in R, we use **rbind()** function.
- We can add single or multiple observations/rows to a Data Frame.
- The basic syntax of rbind() is as shown below.

rbind(<old existing object>, <new object to be added>)

To add a single observation at a time to an existing data frame we will use the following steps.

1. Create a new Data Frame of the same number of variables/columns.

2. Name the newly created Data Frame variable as of old Data Frame in which you want to add this observation.
3. Use the rbind() function to add a new observation.

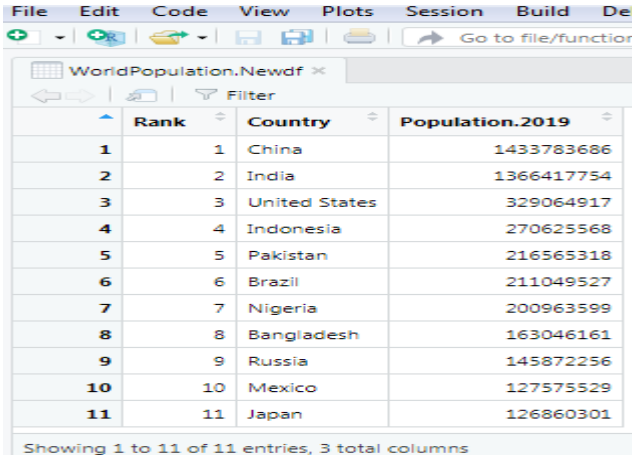
>Japan.Population<- data.frame(11, "Japan", 126860301) #Japan Population Data Frame - Step 1

>names(Japan.Population)<-c("Rank","Country", "Population.2019") #Naming the Data Frame - Step 2

>WorldPopulation.Newdf<-rbind(DataFrame.WorldPopulation,Japan.Population)

#Using rbind() function to insert above observation

Output:



	Rank	Country	Population.2019
1	1	China	1433783686
2	2	India	1366417754
3	3	United States	329064917
4	4	Indonesia	270625568
5	5	Pakistan	216565318
6	6	Brazil	211049527
7	7	Nigeria	200963599
8	8	Bangladesh	163046161
9	9	Russia	145872256
10	10	Mexico	127575529
11	11	Japan	126860301

Showing 1 to 11 of 11 entries, 3 total columns

Adding Multiple Observations/Rows to Data Frame

>NewCountryPopulation<-data.frame(c(12, 13),

c("Ethiopia", "Philippines"),

c(10, 10)) #New Country Population

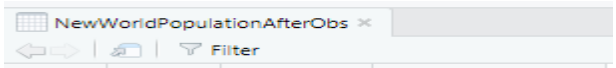
>names(NewCountryPopulation) <- c("Rank", "Country", "Population.2019")

#Naming the above Data Frame

>NewWorldPopulationAfterObs<- rbind(DataFrame.WorldPopulation,NewCountryPopulation)

#Adding observations using rbind() function

Output:



	Rank	Country	Population.2019
3	3	United States	329064917
4	4	Indonesia	270625568
5	5	Pakistan	216565318
6	6	Brazil	211049527
7	7	Nigeria	200963599
8	8	Bangladesh	163046161
9	9	Russia	145872256
10	10	Mexico	127575529
11	11	Japan	10
12	12	Ethiopia	10
13	13	Philippines	10

Showing 3 to 13 of 13 entries, 3 total columns

Adding Variable/Column to Data Frame

The basic syntax for adding a variable or a column to an existing data frame is very simple which is as shown below.

<Existing Data Frame>$ <Variable/Column Name> <- <Object/Value of Variable>

>DataFrame.WorldPopulation$Area.kmSquare<- c(9706961, 3287590, 9372610, 1904569, 881912, 8515767, 923768, 147570, 17098242, 1964375)#Adding New Variable/Column - Area.km²

Output:

NewWorldPopulationAfterObs | DataFrame.WorldPopulation

Filter

	Rank	Country	Population.2019	Area.kmSquare
1	1	China	1433783686	9706961
2	2	India	1366417754	3287590
3	3	United States	329064917	9372610
4	4	Indonesia	270625568	1904569
5	5	Pakistan	216565318	881912
6	6	Brazil	211049527	8515767
7	7	Nigeria	200963599	923768
8	8	Bangladesh	163046161	147570
9	9	Russia	145872256	17098242
10	10	Mexico	127575529	1964375

>str(DataFrame.WorldPopulation)

Output:

'data.frame': 10 obs. of **4** variables:

$ Rank : int 1 2 3 4 5 6 7 8 9 10

$ Country : chr "China" "India" "United States" "Indonesia" ...

$ Population.2019: num 1.43e+09 1.37e+09 3.29e+08 2.71e+08 2.17e+08 ...

$ Area.kmSquare : num 9706961 3287590 9372610 1904569 881912 ...

Adding Multiple Variables/Columns to Data Frame

- We can add multiple variables/columns to a data frame using **cbind()** function.
- To add the multiple columns to a data frame we need to Create a new Data Frame with an individual column using vector c() function.
- Use the cbind() function to add a new data frame as the variables.

#Creating new data frame - name: NewCountry.Df

#Contains two columns DensityPerKmSquare and GDP2019.PPP.InBillions

>NewCountry.Df<- data.frame(

DensityPerKmSquare = c(148, 416, 35, 142, 246, 25, 218, 1105, 9, 65),

GDP2019.PPP.InBillions = c(14172.20, 2957.72, 21482.41, 1066.84, 298.31, 1929.71, 447.01, 313.51, 4345.36, 2696.45))

Output:

DataFrame.WorldPopulation | NewCountry.Df

Filter

	DensityPerKmSquare	GDP2019.PPP.InBillions
1	148	14172.20
2	416	2957.72
3	35	21482.41
4	142	1066.84
5	246	298.31
6	25	1929.71
7	218	447.01
8	1105	313.51
9	9	4345.36
10	65	2696.45

>DataFrame.WorldPopulation<- cbind(DataFrame.WorldPopulation, NewCountry.Df)

Output:

	Rank	Country	Population.2019	Area.kmSquare	DensityPerKmSquare	GDP2019.PPP.InBillions
1	1	China	1433783686	9706961	148	14172.20
2	2	India	1366417754	3287590	416	2957.72
3	3	United States	329064917	9372610	35	21482.41
4	4	Indonesia	270625568	1904569	142	1066.84
5	5	Pakistan	216565318	881912	246	298.31
6	6	Brazil	211049527	8515767	25	1929.71
7	7	Nigeria	200963599	923768	218	447.01
8	8	Bangladesh	163046161	147570	1105	313.51
9	9	Russia	145872256	17098242	9	4345.36
10	10	Mexico	127575529	1964375	65	2696.45

Showing 1 to 10 of 10 entries, 6 total columns

Experiment 8: Write R program to implement linear and multiple regression on 'mtcars' dataset to estimate the value of 'mpg' variable, with best R2 and plot the original values in 'green' and predicted values in 'red'.

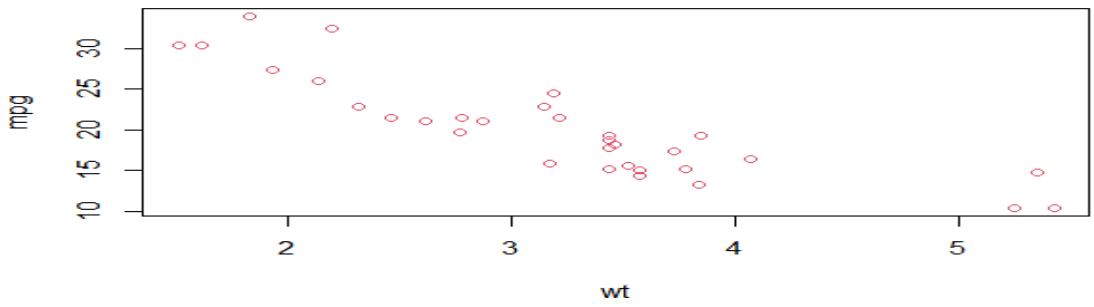
Execution:

```
>library(datasets);                #Getting the dataset
>data(mtcars)
> fit1 <- lm(mpg ~ factor(am), data=mtcars);
> summary(fit1)$coef #simple linear regression
                 Estimate Std. Error   t value     Pr(>|t|)
(Intercept)      17.147368   1.124603 15.247492 1.133983e-15
factor(am)1       7.244939   1.764422  4.106127 2.850207e-04
>library(car);
>fit <- lm(mpg ~ . ,data=mtcars);
>sqrt(vif(fit));cor(mtcars)[1,]          #determine variation inflation factors and select variables for building this linear model
```

Output:

```
##   cyldisphp  drat   wtqsecvs   am  gear  carb
## 3.921 4.650 3.136 1.837 3.894 2.744 2.228 2.156 2.315 2.812
##    mpg    cyldisphp   drat    wtqsecvs    am
##  1.0000 -0.8522 -0.8476 -0.7762  0.6812 -0.8677  0.4187  0.6640  0.5998
##    gear    carb
##  0.4803 -0.5509
>plot(mpg ~ wt, data = mtcars, col=2)          # plotting those variables
```

Output:

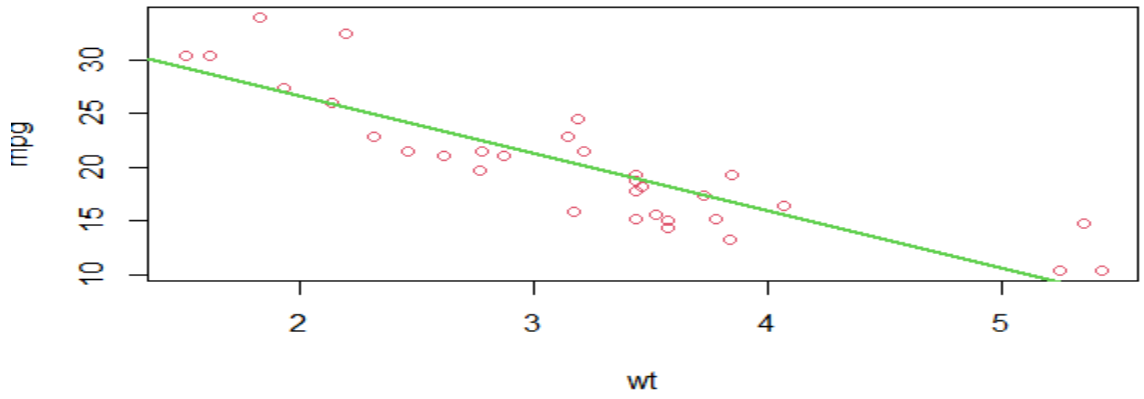



```
> fit <- lm(mpg ~ wt, data = mtcars)
>summary(fit)  # predicting mpg as explained by wt.
Call:
lm(formula = mpg ~ wt, data = mtcars)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858< 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif.codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,  Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

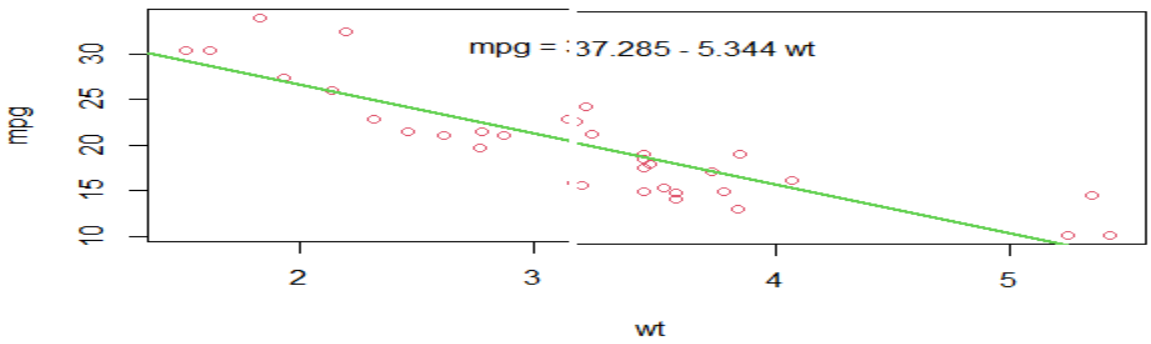
```
>abline(fit,col=3,lwd=2)
```

Output:




```
>bs<-round(coef(fit),3)
>lmlab<- paste0("mpg = ", bs[1],
ifelse(sign(bs[2])==1, " + ", " - "), abs(bs[2]), " wt ")
>mtext(lmlab, 3, line=-2)
```

Output:



Experiment 9: Write R Programs to implement k-means clustering.

K Means Clustering in R Programming is an Unsupervised Non-linear algorithm that cluster data based on similarity or similar groups. It seeks to partition the observations into a pre-specified number of clusters. Segmentation of data takes place to assign each training example to a segment called a cluster. In the unsupervised algorithm, high reliance on raw data is given with large expenditure on manual review for review of relevance is given. It is used in a variety of fields like Banking, healthcare, retail, Media, etc.

Theory

K-Means clustering groups the data on similar groups. The algorithm is as follows:

1. Choose the number **K** clusters.
2. Select at random K points, the centroids(Not necessarily from the given data).
3. Assign each data point to closest centroid that forms K clusters.
4. Compute and place the new centroid of each centroid.
5. Reassign each data point to new cluster.

Procedure:

Step 1: Install pre-requisite packages.

Step 2: Load required libraries.

Step 3: Unsupervised learning – Convert iris data to unlabelled.

Step 4: wws plot to choose maximum number of clusters.

Step 5: Spotting the link in the curve in order to choose the optimal number of clusters.

Step 6: Evaluating Cluster Analysis and plot cluster.

Execution:

```
>install.packages("stats")
>install.packages("dplyr")
>install.packages("ggplot2")
>install.packages("ggfortify")
>library(stats)
>library(dplyr)
>library(ggplot2)
>library(ggfortify)
>View(iris)
>mydata=select(iris,c(1,2,3,4)
```

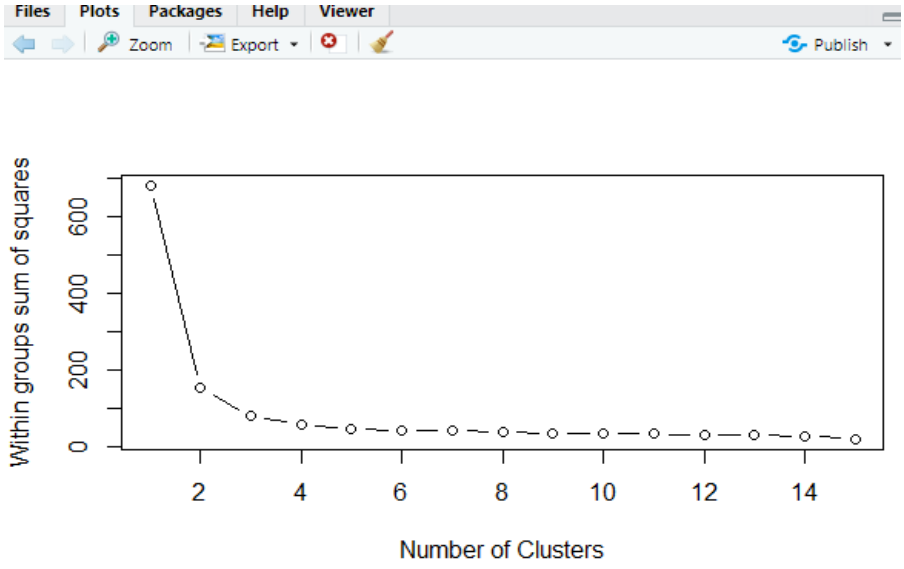
Output:

	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
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa

Showing 1 to 12 of 150 entries, 5 total columns

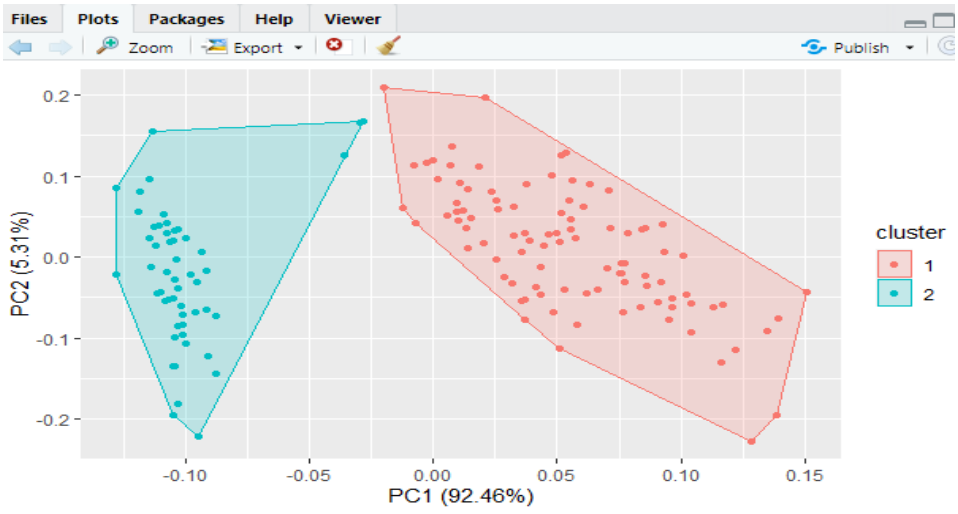
```
>wssplot<- function(data, nc=15, seed=1234){
        wss<- (nrow(data)-1)*sum(apply(data,2,var))
        for (i in 2:nc){
        set.seed(seed)
        wss[i] <- sum(kmeans(data, centers=i)$withinss)}
        plot(1:nc, wss, type="b", xlab="Number of Clusters",
        ylab="Within groups sum of squares")
        wss
}
>wssplot(mydata)
```

Output:




```
>km=kmeans(mydata,2)
>autoplot(km,mydata,frame=TRUE)
```

Output:



>km$centers

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	6.301031	2.886598	4.958763	1.695876
2	5.005660	3.369811	1.560377	0.290566

Experiment 10: Write R Programs to implement k-medoids clustering.

Medoid: A Medoid is a point in the cluster from which the sum of distances to other data points is minimal.**(or)**

A Medoid is a point in the cluster from which dissimilarities with all the other points in the clusters are minimal.

Instead of centroids as reference points in K-Means algorithms, the K-Medoids algorithm takes a Medoid as a reference point.

There are three types of algorithms for K-Medoids Clustering:

1. **PAM (Partitioning Around Clustering)**
2. **CLARA (Clustering Large Applications)**
3. **CLARANS (Randomized Clustering Large Applications)**

PAM is the most powerful algorithm of the three algorithms but has the disadvantage of time complexity. The following K-Medoids are performed using PAM.

Algorithm:

Given the value of k and unlabelled data:

1. Choose k number of random points from the data and assign these k points to k number of clusters. These are the initial medoids.
2. For all the remaining data points, calculate the distance from each medoid and assign it to the cluster with the nearest medoid.
3. Calculate the total cost (Sum of all the distances from all the data points to the medoids)
4. Select a random point as the new medoid and swap it with the previous medoid. Repeat 2 and 3 steps.
5. If the total cost of the new medoid is less than that of the previous medoid, make the new medoid permanent and repeat step 4.
6. If the total cost of the new medoid is greater than the cost of the previous medoid, undo the swap and repeat step 4.
7. The Repetitions have to continue until no change is encountered with new medoids to classify data points.

K-Medoids Clustering in R

Step 1: Load the Necessary Packages

Step 2: Load and Prep the Data

Step 3: Find the Optimal Number of Clusters **using p**am() **function**

Step 4: Perform K-Medoids Clustering with Optimal ***K***

Execution:

```
>data("USArrests")          # Load the data set
>df<- scale(USArrests)      # Scale the data
>head(df, n = 3)            # View the firt 3 rows of the data
```

Output:

	Murder	Assault	UrbanPop	Rape
Alabama	1.24256408	0.7828393	-0.5209066	-0.003416473
Alaska	0.50786248	1.1068225	-1.2117642	2.484202941
Arizona	0.07163341	1.4788032	0.9989801	1.042878388

```
>install.packages(c("cluster", "factoextra"))          #installing required packages
>library(cluster)
>library(factoextra)          #loading the packages.
> pam.res <- pam(df, 2 )      # Computing PAM clustering with k=2
>print(pam.res)
```

```
Medoids:
              ID     Murder    Assault   UrbanPop       Rape
New Mexico 31  0.8292944  1.3708088  0.3081225  1.1603196
Nebraska   27 -0.8008247 -0.8250772 -0.2445636 -0.5052109
Clustering vector:
       Alabama         Alaska        Arizona       Arkansas     California
             1              1              1              2              1
      Colorado    Connecticut       Delaware        Florida        Georgia
             1              2              2              1              1
        Hawaii          Idaho       Illinois        Indiana           Iowa
             2              2              1              2              2
        Kansas       Kentucky      Louisiana          Maine       Maryland
             2              2              1              2              1
 Massachusetts       Michigan      Minnesota    Mississippi       Missouri
             2              1              2              1              1
       Montana       Nebraska         Nevada  New Hampshire     New Jersey
             2              2              1              2              2
    New Mexico       New York North Carolina   North Dakota           Ohio
             1              1              1              2              2
      Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
             2              2              2              2              1
  South Dakota      Tennessee          Texas           Utah        Vermont
             2              1              1              2              2
      Virginia     Washington  West Virginia      Wisconsin        Wyoming
             2              2              2              2              2
Objective function:
   build     swap
1.441358 1.368969

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"  "clusinfo"
 [7] "silinfo"    "diss"       "call"       "data"
```

Output:

```
>dd<- cbind(USArrests, cluster = pam.res$cluster)
>head(dd, n = 3)
```

Output:

	Murder	Assault	UrbanPop	Rape	cluster
Alabama	13.2	236	58	21.2	1
Alaska	10.0	263	48	44.5	1

Arizona 8.1 294 80 31.0 1

>pam.res$medoids # Cluster medoids: New Mexico, Nebraska

Output:

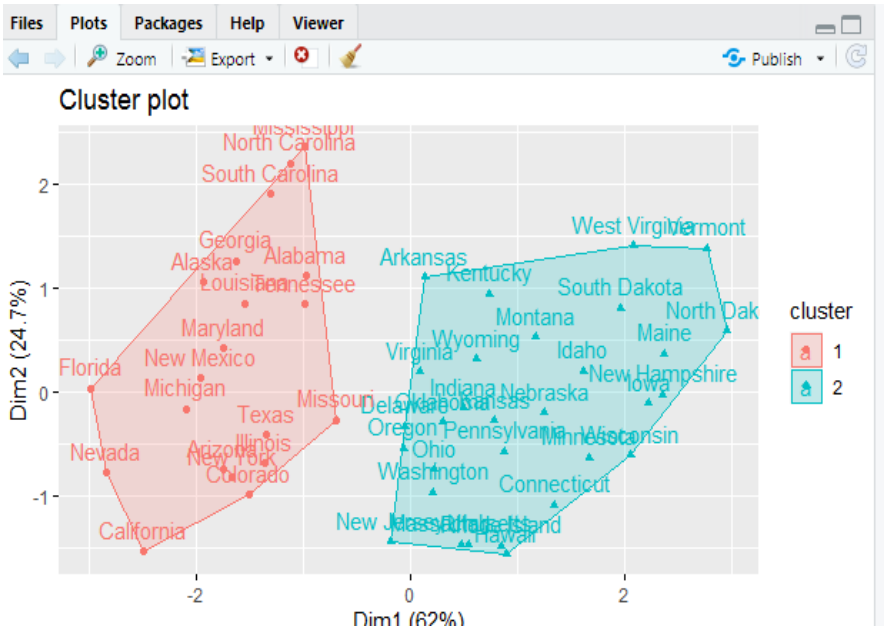
	Murder	Assault	UrbanPop	Rape
New Mexico	0.8292944	1.3708088	0.3081225	1.1603196
Nebraska	-0.8008247	-0.8250772	-0.2445636	-0.5052109

> head(pam.res$clustering) # Cluster numbers

Alabama	Alaska	Arizona	Arkansas	California	Colorado
1	1	1	2	1	1

>fviz_cluster(pam.res, data = df) #plot results of final k medoids model

Output:



Experiment 11: Write R Programs to implement density based clustering on iris dataset.

Execution:

```
>data(iris)                # Loading data
>str(iris)                 # Structure
```

Output:

```
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
>install.packages("fpc")        # Installing Packages
>library(fpc)             # Loading package
> iris_1 <- iris[-5]      # Remove label form dataset
>set.seed(220)# Setting seed
>Dbscan_cl<- dbscan(iris_1, eps = 0.45, MinPts = 5)
>Dbscan_cl
```

Output:

```
dbscanPts=150 MinPts=5 eps=0.45
          0  1  2
border  24  4 13
seed     0 44 65
total   24 48 78
>Dbscan_cl$cluster              # Checking cluster
```

Output:

```
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [40] 1 1 0 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 2 2 0 2 0 2 2 2 2 2 0 2 2 2 2 2 2 2 2
 [79] 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 2 2 0 0 0 0 0 2 2 2 2 0 2 2
[118] 0 0 2 2 2 0 2 2 0 2 2 2 0 0 0 2 2 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2
```

>table(Dbscan_cl$cluster, iris$Species) # Table

Output:

setosaversicolorvirginica

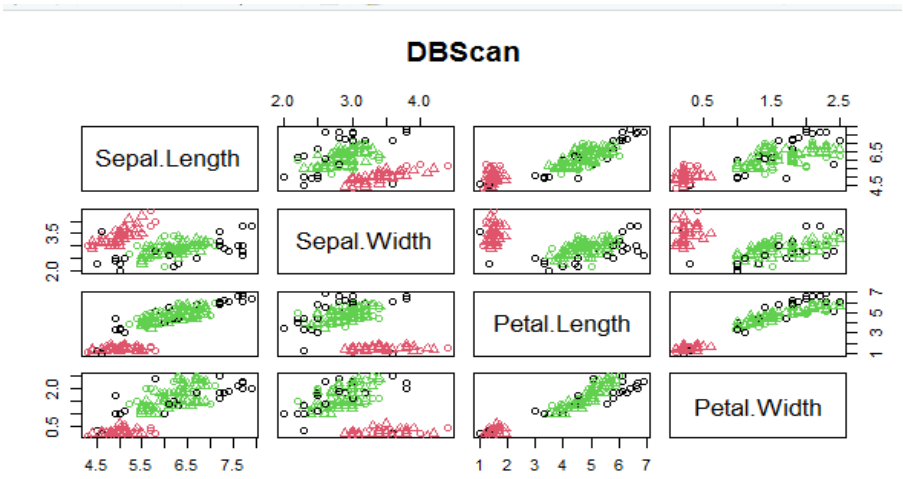
0 2 7 15

1 48 0 0

2 0 43 35

>plot(Dbscan_cl, iris_1, main = "DBScan")

Output:



Experiment 12: Write a R Program to implement decision trees using 'readingSkills' dataset.

Procedure:

Step 1: Run the required libraries

Step 2: Load the dataset reading Skills and execute head(readingSkills)

Step 3: Splitting dataset into 4:1 ratio for train and test data

Step 4: Create the decision tree model using ctree and plot the model

Execution:

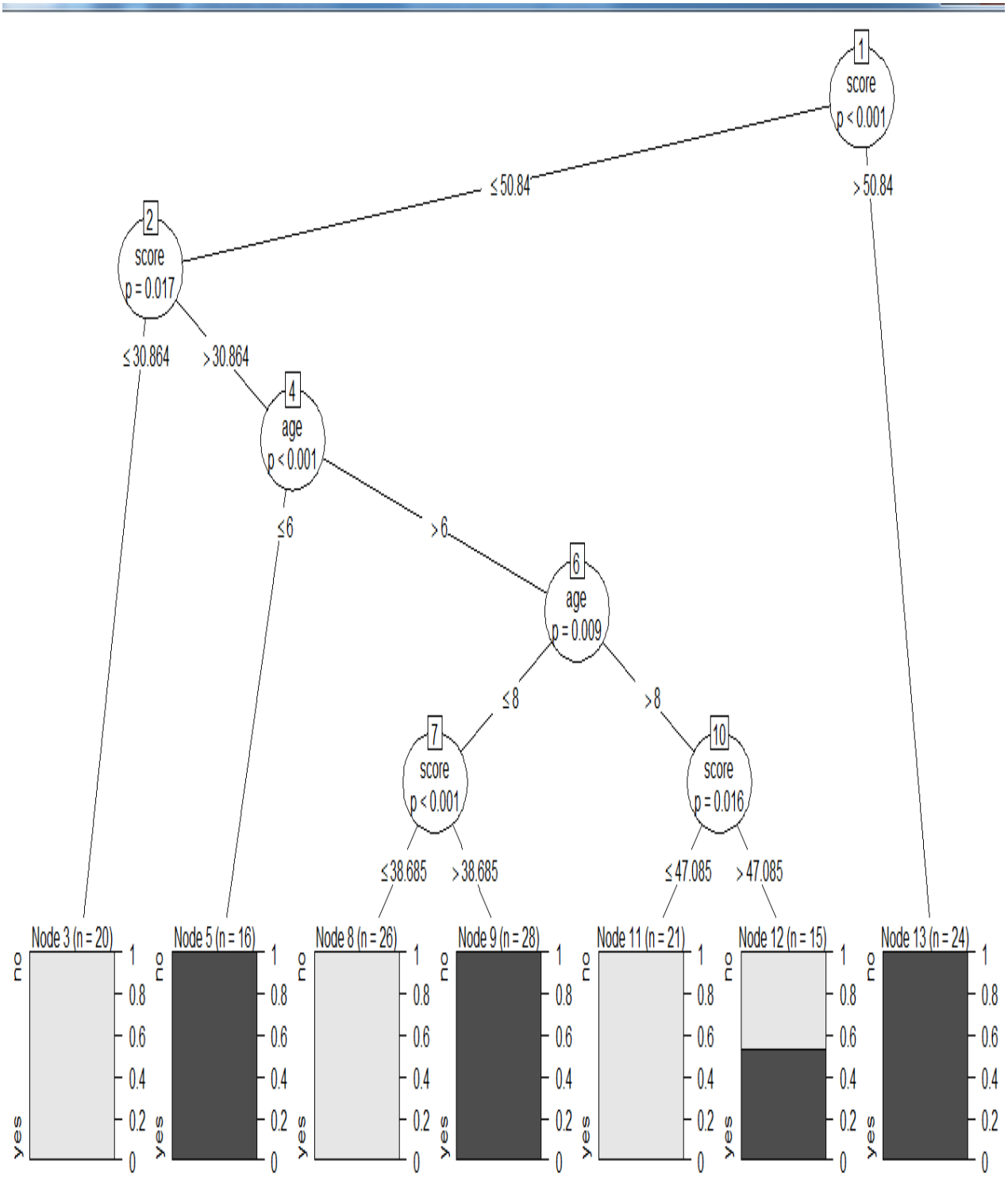
```
>library(datasets)
>library(caTools)
>library(party)
>library(dplyr)
>library(magrittr)
>data("readingSkills")
>head(readingSkills)
```

Output:

	nativeSpeaker	age	shoeSize	score
1	yes	5	24.83189	32.29385
2	yes	6	25.95238	36.63105
3	no	11	30.42170	49.60593
4	yes	7	28.66450	40.28456
5	yes	11	31.88207	55.46085
6	yes	10	30.07843	52.83124

```
>sample_data = sample.split(readingSkills, SplitRatio = 0.8)
>train_data<- subset(readingSkills, sample_data == TRUE)
>test_data<- subset(readingSkills, sample_data == FALSE)
>model<- ctree(nativeSpeaker ~ ., train_data)
>plot(model)
```

Output:


1
score
p < 0.001
≤ 50.84
> 50.84
2
score
p = 0.017
≤ 30.864
> 30.864
4
age
p < 0.001
≤ 6
> 6
6
age
p = 0.009
≤ 8
> 8
7
score
p < 0.001
10
score
p = 0.016
≤ 38.685
> 38.685
≤ 47.085
> 47.085
Node 3 (n = 20)
Node 5 (n = 16)
Node 8 (n = 26)
Node 9 (n = 28)
Node 11 (n = 21)
Node 12 (n = 15)
Node 13 (n = 24)
no
yes
0
0.2
0.4
0.6
0.8
1

Experiment 13: Implement decision trees using 'iris' dataset using package party and 'rpart'.

Decision Trees with Package party

Code - Split Iris data

```
>str(iris)
```

Output:

```
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
>set.seed(1234)
>ind<- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
>trainData<- iris[ind==1,]
```

Output:

trainData × testData ×

Filter

	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
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa

Showing 1 to 12 of 112 entries, 5 total columns

```
>testData<- iris[ind==2,]
```

Output:

trainData × testData ×

Filter

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5	5.0	3.6	1.4	0.2	setosa
14	4.3	3.0	1.1	0.1	setosa
16	5.7	4.4	1.5	0.4	setosa
26	5.0	3.0	1.6	0.2	setosa
28	5.2	3.5	1.5	0.2	setosa
29	5.2	3.4	1.4	0.2	setosa
36	5.0	3.2	1.2	0.2	setosa
39	4.4	3.0	1.3	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
50	5.0	3.3	1.4	0.2	setosa
53	6.9	3.1	4.9	1.5	versicolor

Showing 1 to 12 of 38 entries, 5 total columns

Build a Decision Tree

>library(party)

>myFormula<- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width

>iris_ctree<- ctree(myFormula, data=trainData)

>table(predict(iris_ctree), trainData$Species)

setosa versicolorvirginica

setosa 40 0 0

versicolor 0 37 3

virginica 0 1 31

Print rules and plot

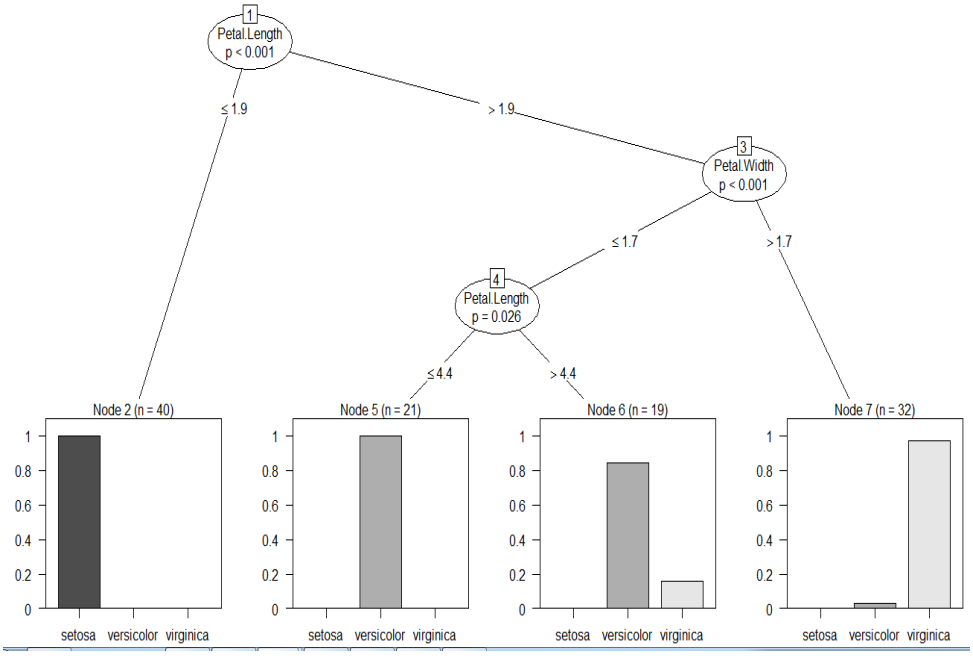
>print(iris_ctree)

Output:

```
##
##   Conditional inference tree with 4 terminal nodes
##
## Response:  Species
## Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.W
idth
## Number of observations:  112
##
## 1) Petal.Length <= 1.9; criterion = 1, statistic = 104.64
3
##   2)*  weights = 40
## 1) Petal.Length > 1.9
##   3) Petal.Width <= 1.7; criterion = 1, statistic = 48.93
9
##     4) Petal.Length <= 4.4; criterion = 0.974, statistic
= 7.397
##       5)*  weights = 21
##     4) Petal.Length > 4.4
##       6)*  weights = 19
##   3) Petal.Width > 1.7
##     7)*  weights = 32
```

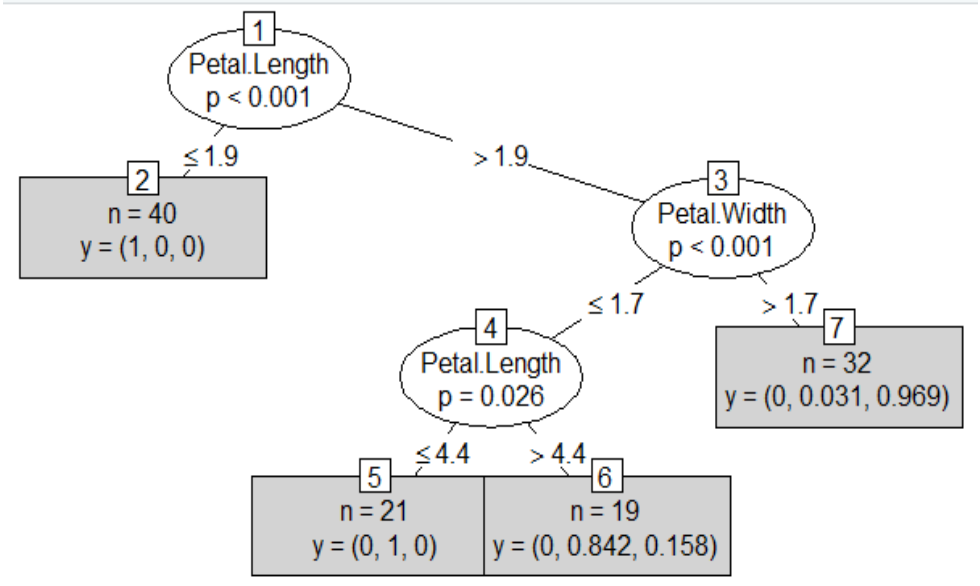
>plot(iris_ctree)

Output:



>plot(iris_ctree, type="simple")

Output:



Decision Trees with Package rpart

The algorithm of decision tree models works by repeatedly partitioning the data into multiple sub-spaces, so that the outcomes in each final sub-space is as homogeneous as possible. This approach is technically called *recursive partitioning*.

>library(rpart)

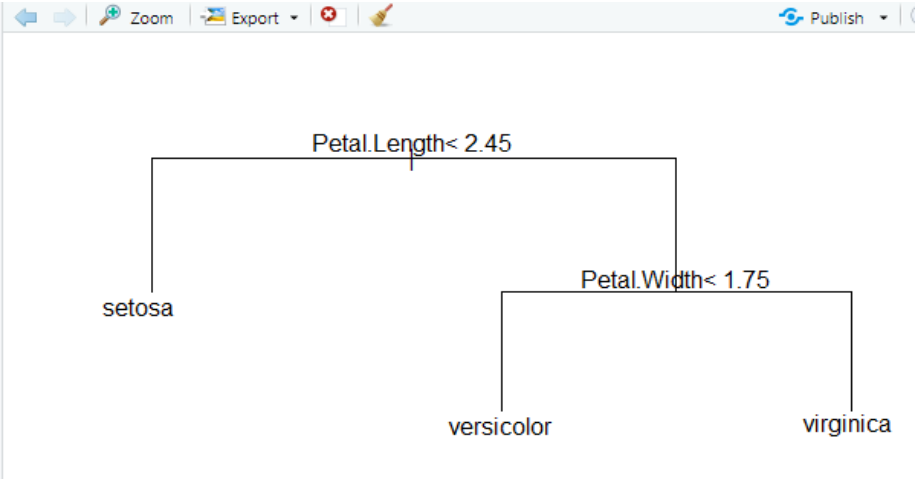
>model<- rpart(Species ~., data = iris)

>par(xpd = NA)

>plot(model)

>text(model, digits = 3)

Output:



>newdata<- data.frame(

+ Sepal.Length = 6.5, Sepal.Width = 3.0,

+ Petal.Length = 5.2, Petal.Width = 2.0)

Output:

>model |> predict(newdata, "class")

Output:

iris_ctree | iris_ctree | iris | newdata

Filter

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	6.5	3	5.2	2

>library(mlbench)
>data("PimaIndiansDiabetes2", package = "mlbench")# Load the data and remove NAs

> PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)

Output:

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
4	1	89	66	23	94	28.1	0.167	21	neg
5	0	137	40	35	168	43.1	2.288	33	pos
7	3	78	50	32	88	31.0	0.248	26	pos
9	2	197	70	45	543	30.5	0.158	53	pos
14	1	189	60	23	846	30.1	0.398	59	pos
15	5	166	72	19	175	25.8	0.587	51	pos
17	0	118	84	47	230	45.8	0.551	31	pos
19	1	103	30	38	83	43.3	0.183	33	neg
20	1	115	70	30	96	34.6	0.529	32	pos
21	3	126	88	41	235	39.3	0.704	27	neg
25	11	143	94	33	146	36.6	0.254	51	pos

Showing 1 to 12 of 392 entries, 9 total columns

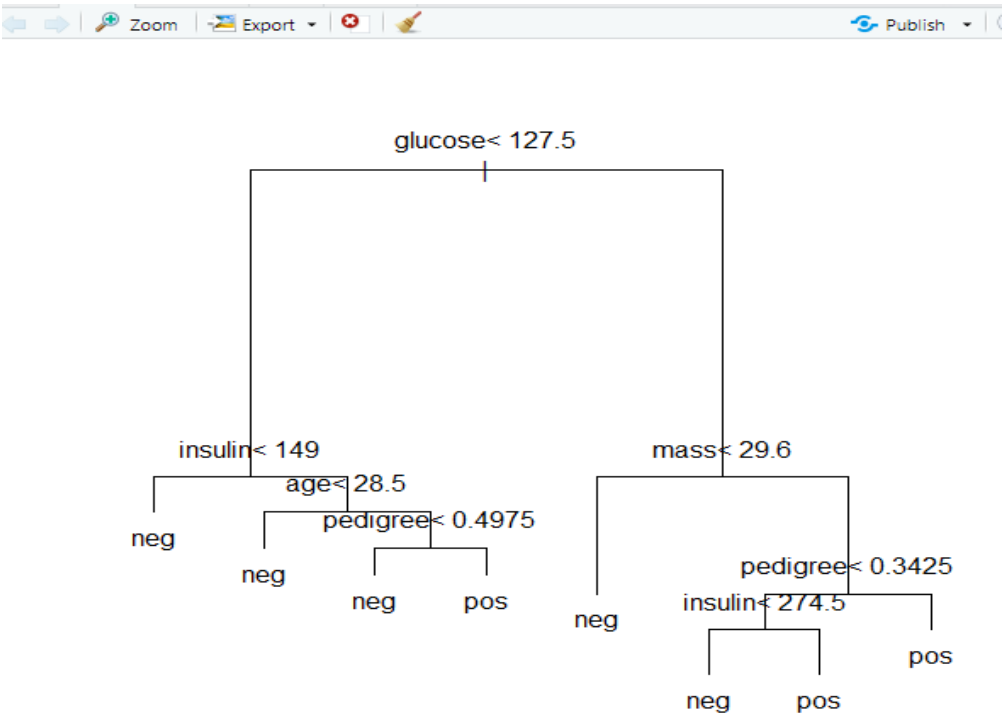
>sample(PimaIndiansDiabetes2, 3)

Output:

```
     pregnant glucose pressure triceps insulin mass pedigree age diabetes
414        1     143       74      22      61 26.2    0.256  21      neg
390        3     100       68      23      81 31.6    0.949  28      neg
383        1     109       60       8     182 25.4    0.947  21      neg
```

```
>set.seed(123) # Split the data into training and test set
>library(caret)
>training.samples<- PimaIndiansDiabetes2$diabetes |>
+     createDataPartition(p = 0.8, list = FALSE)
>train.data<- PimaIndiansDiabetes2[training.samples, ]
>test.data<- PimaIndiansDiabetes2[-training.samples, ]
>set.seed(123)          # Build the model
> model1 <- rpart(diabetes ~., data = train.data, method = "class")
>par(xpd = NA)          # Avoid clipping the text in some device
>plot(model1) # Plot the trees
>text(model1, digits = 3)
```

Output:



Experiment 14: Use a Corpus() function to create a data corpus then Build a term Matrix and Reveal word frequencies.

Procedure:

Step 1: Create a text file

Step 2 : Install and load the required packages

Step 3 : Text mining

Step 4 : Build a term matrix

Step 5 : Generate the Word cloud

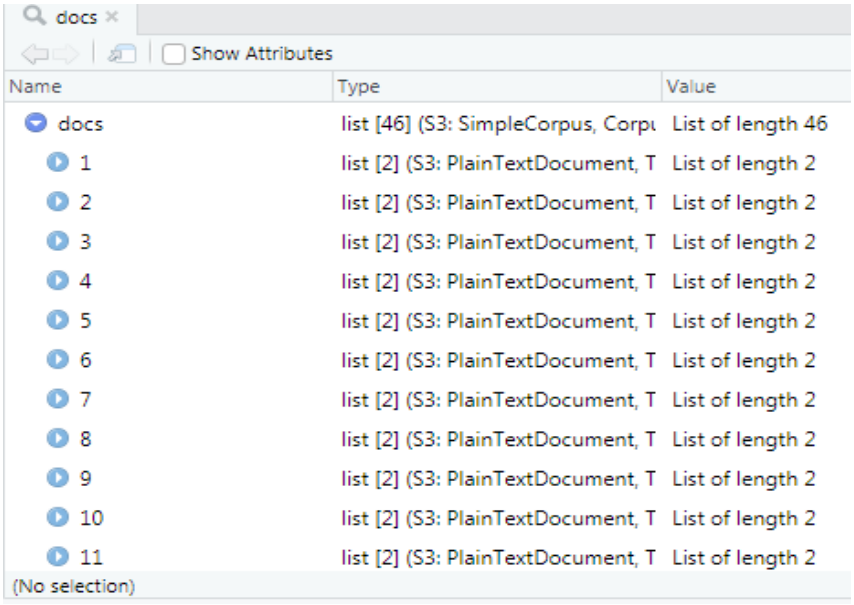
Step 6 : Explore word frequencies

Step 7 : Plot word frequencies

Execution:

```
# Install
>install.packages("tm")                # for text mining
>install.packages("SnowballC")         # for text stemming
>install.packages("wordcloud")         # word-cloud generator
>install.packages("RColorBrewer")      # color palettes
# Load
>library("tm")
>library("SnowballC")
>library("wordcloud")
>library("RColorBrewer")
>filePath<- "http://www.sthda.com/sthda/RDoc/example-files/martin-luther-king-i-have-a-dream-speech.txt"                                    # Read the text file
> text <- readLines(filePath)
> docs <- Corpus(VectorSource(text))          # Load the data as a corpus
>toSpace<- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
>docs <- tm_map(docs, toSpace, "/")          #Text transformation
>docs <- tm_map(docs, toSpace, "@")
>docs <- tm_map(docs, toSpace, "\\|")
```

Output:



Cleaning the text

```
>docs <- tm_map(docs, content_transformer(tolower))        # Convert the text to lower case
>docs <- tm_map(docs, removeNumbers)     # Remove numbers
>docs <- tm_map(docs, removeWords, stopwords("english"))        # Remove english common stopwords

>docs <- tm_map(docs, removeWords, c("blabla1", "blabla2")) # Remove your own stop word
                                        # specify your stopwords as a character vector

>docs <- tm_map(docs, removePunctuation) # Remove punctuations
>docs <- tm_map(docs, stripWhitespace)              # Eliminate extra white spaces
>docs <- tm_map(docs, stemDocument)                 # Text stemming
>dtm<- TermDocumentMatrix(docs)
>dtm
```

Output:

```
<< Term document matrix (terms:162, document:46)>>
Non-/sparse entries: 277/7175
Sparcity: 96%
Maximum term length: 12
Weighting: term frequency(tf)
```

Build a term-document matrix

```
> m <- as.matrix(dtm)
> v <- sort(rowSums(m),decreasing=TRUE)
```

> d <- data.frame(word = names(v),freq=v)

> head(d, 10)

Output:

```
            word freq
the          the   31
and          and   26
will        will   17
freedom freedom   13
from        from   12
let          let   11
that        that   10
this        this   10
with        with   10
dream      dream    9
```

>set.seed(1234)

wordcloud(words = d$word, freq = d$freq, min.freq = 1,

max.words=200, random.order=FALSE, rot.per=0.35,

colors=brewer.pal(8, "Dark2"))

Explore word frequencies

>findFreqTerms(dtm, lowfreq = 4)

Output:

[1] "and" "dream" "have" "the" "day" "one"

[7] "that" "this" "will" "all" "able" "freedom"

[13] "with" "every" "mountain" "shall" "free" "together,"

[19] "when" "from" "let" "ring"

>findAssocs(dtm, terms = "freedom", corlimit = 0.3)

Output:

let	from	ring	into	stone	mountainside,
0.89	0.86	0.79	0.34	0.34	0.34

state	every	mountain
0.32	0.32	0.32

The frequency table of words

> head(d, 10)

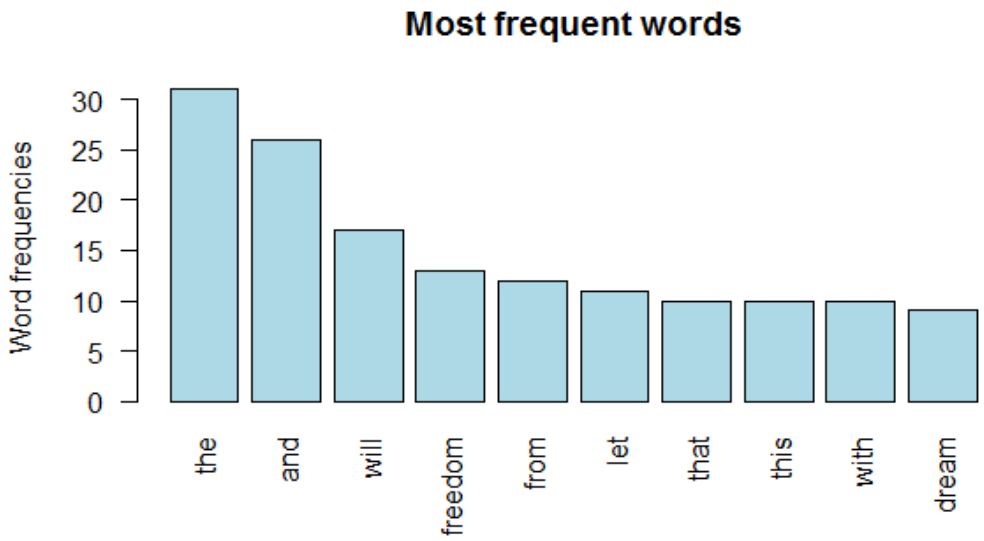
Output:

```
           word freq
the         the   31
and         and   26
will       will   17
freedom freedom   13
from       from   12
let         let   11
that       that   10
this       this   10
with       with   10
dream     dream    9
```

Plot word frequencies

```
>barplot(d[1:10,]$freq, las = 2, names.arg = d[1:10,]$word,
+         col ="lightblue", main ="Most frequent words",
+         ylab = "Word frequencies")
```

Output:



CONTENT BEYOND SYLLABUS

Program 1: Principle Component Analysis

>install.packages("stats")

>install.packages("dplyr")

>library(stats)

>library(dplyr)

>View(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

>mydata=select(iris,c(1,2,3,4))

>cor(mydata)

```
             Sepal.Lengt  Sepal.Widt  Petal.Lengt  Petal.Widt
             h            h           h            h
Sepal.Length 1.0000000    -0.1175698  0.8717538    0.8179411
Sepal.Width  -0.1175698   1.0000000   -0.4284401   -0.3661259
Petal.Length 0.8717538    -0.4284401  1.0000000    0.9628654
Petal.Width  0.8179411    -0.3661259  0.9628654    1.0000000
```

```
>mean(cor(mydata))
[1] 0.4675531
```

>PCA=princomp(mydata)

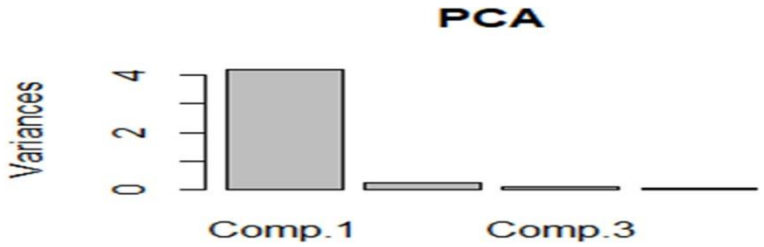
```
>PCA$loadings


Loadings:

             Comp.1 Comp.2 Comp.3 Comp.4
Sepal.Length  0.361  0.657  0.582  0.315

Sepal.Width          0.730 0.598 0.320
Proportion Var   0.25
                        1.00   1.00   1.00
                       0.25   0.25   0.25
```

>PC=PCA$scores

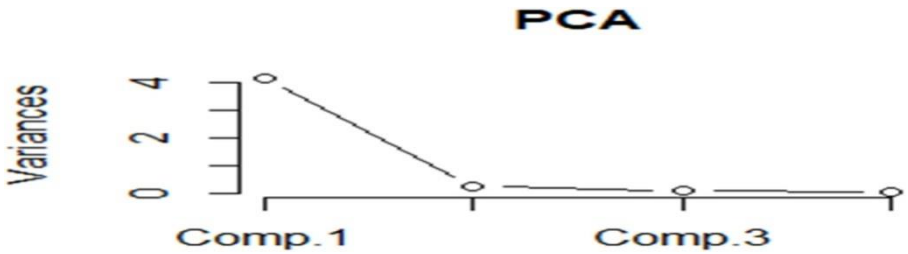
>View(PC)

	Comp.1	Comp.2	Comp.3	Comp.4
1	-2.684125626	0.31939725	0.02791483	0.0022624371
2	-2.714141687	-0.17700123	0.21046427	0.0990265503
3	-2.888990569	-0.14494943	-0.01790026	0.0199683897
4	-2.745342856	-0.31829898	-0.03155937	-0.0755758166
5	-2.728716537	0.32675451	-0.09007924	-0.0612585926
6	-2.280859633	0.74133045	-0.16867766	-0.0242008576

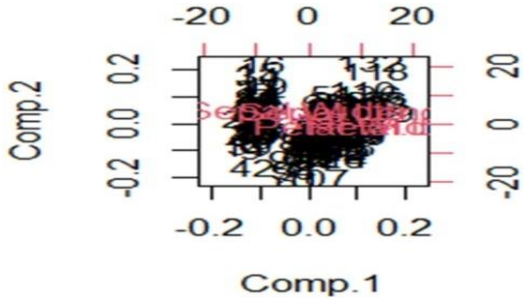
>plot(PCA)



>plot(PCA,type="l")



>biplot(PCA)

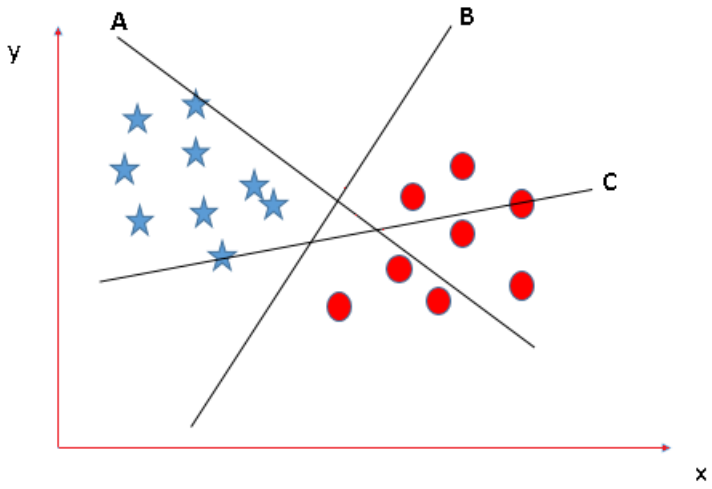


Program 2: Implementation of SVM in R

How SVM works

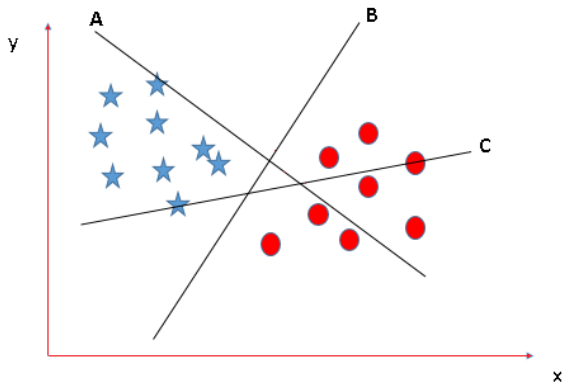
A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyper plane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyper plane that categorizes new examples.
The most important question that arises while using SVM is how to decide the right hyperplane. Consider the following scenarios:

Scenario 1:
In this scenario, there are three hyper planes called A, B, C. Now the problem is to identify the right hyper-plane which best differentiates the stars and the circles.



- The thumb rule to be known, before finding the right hyper plane, to classify star and circle is that the hyper plane should be selected which segregate two classes better.
  In this case, B classifies star and circle better, hence it is a right hyper plane.

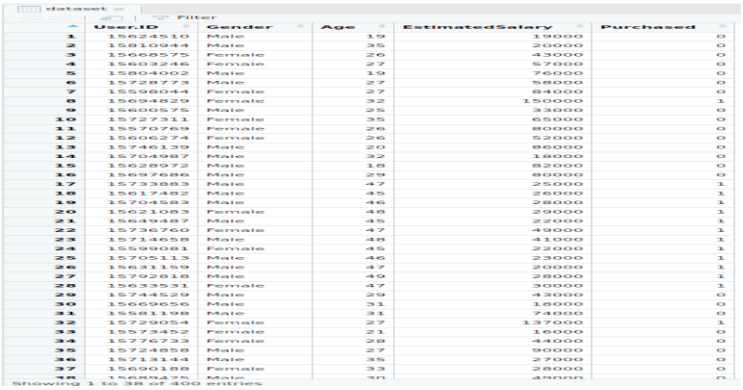
- **Scenario 2:**
  Now take another Scenario where all three planes are segregating classes well. Now the question arises of how to identify the right plane in this situation.



Importing the dataset

```
# Importing the dataset
dataset = read.csv('Social_Network_Ads.csv')
dataset = dataset[3:5]
```

Output:



	User.ID	Gender	Age	EstimatedSalary	Purchased
1	15624510	Male	19	19000	0
2	15810944	Male	35	20000	0
3	15668575	Female	26	43000	0
4	15603246	Female	27	57000	0
5	15804002	Male	19	76000	0
6	15728773	Male	27	58000	0
7	15598044	Female	27	84000	0
8	15694829	Female	32	150000	1
9	15600575	Male	25	33000	0
10	15727311	Female	35	65000	0
11	15570769	Female	26	80000	0
12	15606274	Female	26	52000	0
13	15746139	Male	20	86000	0
14	15704987	Male	32	18000	0
15	15628972	Male	18	82000	0
16	15697686	Male	29	80000	0
17	15733883	Male	47	25000	1
18	15617482	Male	45	26000	1
19	15704583	Male	46	28000	1
20	15621083	Female	48	29000	1
21	15649487	Male	45	22000	1
22	15736760	Female	47	49000	1
23	15714658	Male	48	41000	1
24	15599081	Female	45	22000	1
25	15705113	Male	46	23000	1
26	15631159	Male	47	20000	1
27	15792818	Male	49	28000	1
28	15633531	Female	47	30000	1
29	15744529	Male	29	43000	0
30	15669656	Male	31	18000	0
31	15581198	Male	31	74000	0
32	15729054	Female	27	137000	1
33	15573452	Female	21	16000	0
34	15776733	Female	28	44000	0
35	15724858	Male	27	90000	0
36	15713144	Male	35	27000	0
37	15690188	Female	33	28000	0
38	15689425	Male	30	49000	0

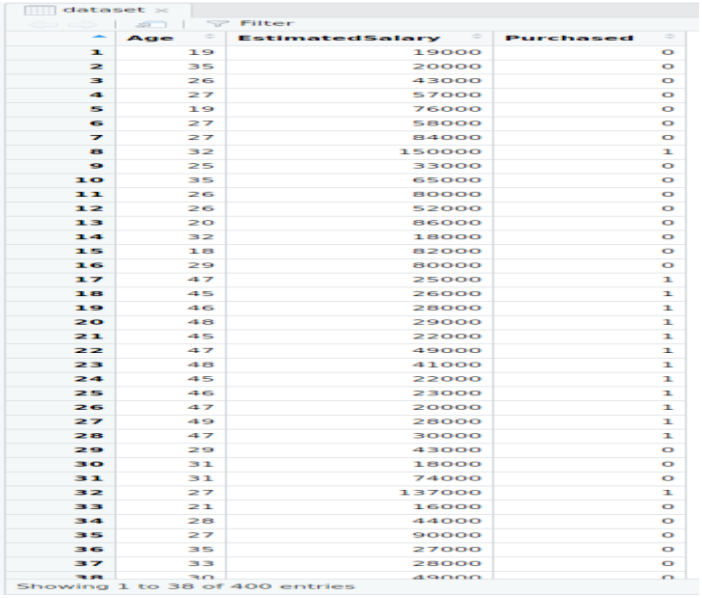
Showing 1 to 38 of 400 entries

Selecting columns 3-5

This is done for ease of computation and implementation (to keep the example simple).

```
# Taking columns 3-5

dataset = dataset[3:5]
```

Output:



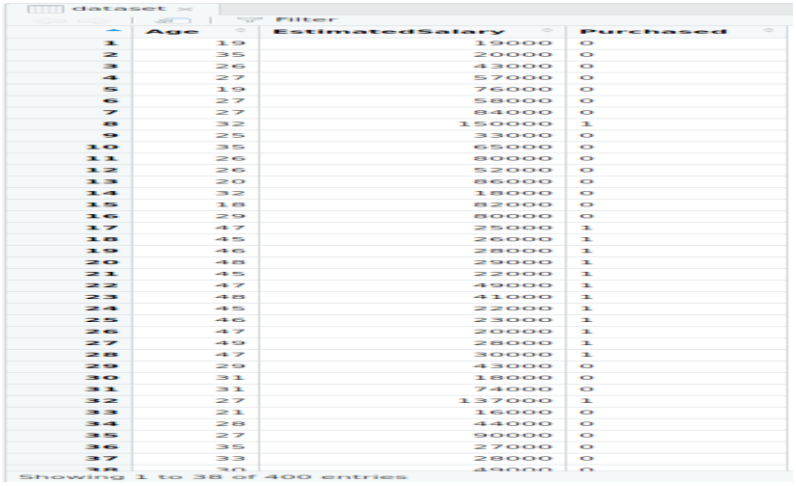
	Age	EstimatedSalary	Purchased
1	19	19000	0
2	35	20000	0
3	26	43000	0
4	27	57000	0
5	19	76000	0
6	27	58000	0
7	27	84000	0
8	32	150000	1
9	25	33000	0
10	35	65000	0
11	26	80000	0
12	26	52000	0
13	20	86000	0
14	32	18000	0
15	18	82000	0
16	29	80000	0
17	47	25000	1
18	45	26000	1
19	46	28000	1
20	48	29000	1
21	45	22000	1
22	47	49000	1
23	48	41000	1
24	45	22000	1
25	46	23000	1
26	47	20000	1
27	49	28000	1
28	47	30000	1
29	29	43000	0
30	31	18000	0
31	31	74000	0
32	27	137000	1
33	21	16000	0
34	28	44000	0
35	27	90000	0
36	35	27000	0
37	33	28000	0
38	30	49000	0

Showing 1 to 38 of 400 entries

Encoding the target feature

```
# Encoding the target feature as factor

dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
```

Output:



dataset

Filter

	Age	EstimatedSalary	Purchased
1	19	19000	0
2	35	20000	0
3	26	43000	0
4	27	57000	0
5	19	76000	0
6	27	58000	0
7	27	84000	0
8	32	150000	1
9	25	33000	0
10	35	65000	0
11	26	80000	0
12	26	52000	0
13	20	86000	0
14	32	18000	0
15	18	82000	0
16	29	80000	0
17	47	25000	1
18	45	26000	1
19	46	28000	1
20	48	29000	1
21	45	22000	1
22	47	49000	1
23	48	41000	1
24	45	22000	1
25	46	23000	1
26	47	20000	1
27	49	28000	1
28	47	30000	1
29	29	43000	0
30	31	18000	0
31	31	74000	0
32	27	137000	1
33	21	16000	0
34	28	44000	0
35	27	90000	0
36	35	27000	0
37	33	28000	0
38	30	49000	0

Showing 1 to 38 of 400 entries

Splitting the dataset

Splitting the dataset into the Training set and Test set

install.packages('caTools')

library(caTools)

set.seed(123)

split = sample.split(dataset$Purchased, SplitRatio = 0.75)

training_set = subset(dataset, split == TRUE)

test_set = subset(dataset, split == FALSE)

Output:

```
> split
  [1]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
 [12] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
 [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
 [34] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [45] FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
 [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
 [67]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
 [78]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
 [89] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[100]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
[111]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[122]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
[133]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[144]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[155]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
[166]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
[177]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[188]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[199] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[210]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[221]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE
[232]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[243]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[254]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[265] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
[276]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
[287]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[298]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
[309]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[320]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[331]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
[342]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[353] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[364] FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
[375]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
[386]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
[397]  TRUE  TRUE  TRUE FALSE
```

Feature Scaling

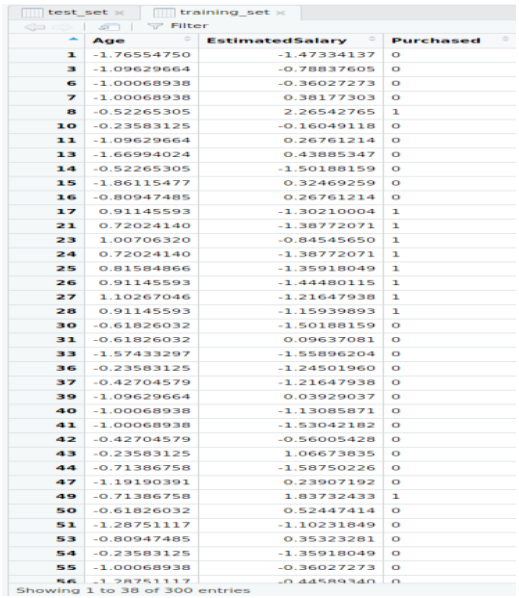
Feature Scaling

training_set[-3] = scale(training_set[-3])

test_set[-3] = scale(test_set[-3])

Output:

Feature scaled training dataset & Feature scaled test dataset

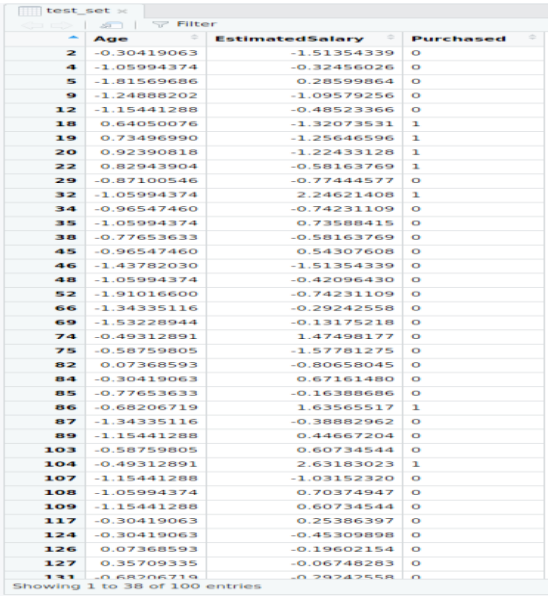


test_set | training_set

Filter

	Age	EstimatedSalary	Purchased
1	-1.76554750	-1.47334137	0
3	-1.09629664	-0.78837605	0
6	-1.00068938	-0.36027273	0
7	-1.00068938	0.38177303	0
8	-0.52265305	2.26542765	1
10	-0.23583125	-0.16049118	0
11	-1.09629664	0.26761214	0
13	-1.66994024	0.43885347	0
14	-0.52265305	-1.50188159	0
15	-1.86115477	0.32469259	0
16	-0.80947485	0.26761214	0
17	0.91145593	-1.30210004	1
21	0.72024140	-1.38772071	1
23	1.00706320	-0.84545650	1
24	0.72024140	-1.38772071	1
25	0.81584866	-1.35918049	1
26	0.91145593	-1.44480115	1
27	1.10267046	-1.21647938	1
28	0.91145593	-1.15939893	1
30	-0.61826032	-1.50188159	0
31	-0.61826032	0.09637081	0
33	-1.57433297	-1.55896204	0
36	-0.23583125	-1.24501960	0
37	-0.42704579	-1.21647938	0
39	-1.09629664	0.03929037	0
40	-1.00068938	-1.13085871	0
41	-1.00068938	-1.53042182	0
42	-0.42704579	-0.56005428	0
43	-0.23583125	1.06673835	0
44	-0.71386758	-1.58750226	0
47	-1.19190391	0.23907192	0
49	-0.71386758	1.83732433	1
50	-0.61826032	0.52447414	0
51	-1.28751117	-1.10231849	0
53	-0.80947485	0.35323281	0
54	-0.23583125	-1.35918049	0
55	-1.00068938	-0.36027273	0
56	-1.28751117	-0.44589340	0

Showing 1 to 38 of 300 entries



test_set

Filter

	Age	EstimatedSalary	Purchased
2	-0.30419063	-1.51354339	0
4	-1.05994374	-0.32456026	0
5	-1.81569686	0.28599864	0
9	-1.24888202	-1.09579256	0
12	-1.15441288	-0.48523366	0
18	0.64050076	-1.32073531	1
19	0.73496990	-1.25646596	1
20	0.92390818	-1.22433128	1
22	0.82943904	-0.58163769	1
29	-0.87100546	-0.77444577	0
32	-1.05994374	2.24621408	1
34	-0.96547460	-0.74231109	0
35	-1.05994374	0.73588415	0
38	-0.77653633	-0.58163769	0
45	-0.96547460	0.54307608	0
46	-1.43782030	-1.51354339	0
48	-1.05994374	-0.42096430	0
52	-1.91016600	-0.74231109	0
66	-1.34335116	-0.29242558	0
69	-1.53228944	-0.13175218	0
74	-0.49312891	1.47498177	0
75	-0.58759805	-1.57781275	0
82	0.07368593	-0.80658045	0
84	-0.30419063	0.67161480	0
85	-0.77653633	-0.16388686	0
86	-0.68206719	1.63565517	1
87	-1.34335116	-0.38882962	0
89	-1.15441288	0.44667204	0
103	-0.58759805	0.60734544	0
104	-0.49312891	2.63183023	1
107	-1.15441288	-1.03152320	0
108	-1.05994374	0.70374947	0
109	-1.15441288	0.60734544	0
117	-0.30419063	0.25386397	0
124	-0.30419063	-0.45309898	0
126	0.07368593	-0.19602154	0
127	0.35709335	-0.06748283	0
131	-0.68206719	-0.29242558	0

Showing 1 to 38 of 100 entries

Fitting SVM to the training set

Fitting SVM to the Training set

install.packages('e1071')

library(e1071)

classifier = svm(formula = Purchased ~ .,

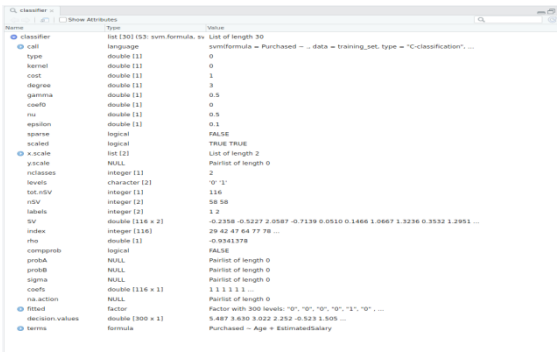
data = training_set,

type = 'C-classification',

kernel = 'linear')

Output:

- Classifier detailed



classifier

Show Attributes

Name	Type	Value
classifier	list [30] (S3: svm.formula, sv	List of length 30
call	language	svm(formula = Purchased ~ ., data = training_set, type = "C-classification", ...
type	double [1]	0
kernel	double [1]	0
cost	double [1]	1
degree	double [1]	3
gamma	double [1]	0.5
coef0	double [1]	0
nu	double [1]	0.5
epsilon	double [1]	0.1
sparse	logical	FALSE
scaled	logical	TRUE TRUE
x.scale	list [2]	List of length 2
y.scale	NULL	Pairlist of length 0
nclasses	integer [1]	2
levels	character [2]	'0' '1'
tot.nSV	integer [1]	116
nSV	integer [2]	58 58
labels	integer [2]	1 2
SV	double [116 x 2]	-0.2358 -0.5227 2.0587 -0.7139 0.0510 0.1466 1.0667 1.3236 0.3532 1.2951 ...
index	integer [116]	29 42 47 64 77 78 ...
rho	double [1]	-0.9341378
compprob	logical	FALSE
probA	NULL	Pairlist of length 0
probB	NULL	Pairlist of length 0
sigma	NULL	Pairlist of length 0
coefs	double [116 x 1]	1 1 1 1 1 1 ...
na.action	NULL	Pairlist of length 0
fitted	factor	Factor with 300 levels: "0", "0", "0", "0", "1", "0" , ...
decision.values	double [300 x 1]	5.487 3.630 3.022 2.252 -0.523 1.505 ...
terms	formula	Purchased ~ Age + EstimatedSalary

Classifier in nutshell

```
> classifier

Call:
svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  116
```

Predicting the test set result

Predicting the Test set results

y_pred = predict(classifier, newdata = test_set[-3])

```
> y_pred
  2   4   5   9  12  18  19  20  22  29  32  34  35  38  45  46  48
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 52  66  69  74  75  82  84  85  86  87  89 103 104 107 108 109 117
  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
124 126 127 131 134 139 148 154 156 159 162 163 170 175 176 193 199
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
200 208 213 224 226 228 229 230 234 236 237 239 241 255 264 265 266
  0   1   1   1   0   1   0   1   1   1   0   1   1   1   0   1   1
273 274 281 286 292 299 302 305 307 310 316 324 326 332 339 341 343
  1   1   1   0   1   1   1   0   1   0   0   0   0   1   0   1   0
347 353 363 364 367 368 369 372 373 380 383 389 392 395 400
  1   1   0   1   1   1   0   1   0   1   1   0   0   0   0
Levels: 0 1
> |
```

Making Confusion Matrix

Making the Confusion Matrix

cm = table(test_set[, 3], y_pred)

```
> CM
   y_pred
     0  1
  0 57  7
  1 13 23
```

Visualizing the Training set results

installing library ElemStatLearn

library(ElemStatLearn)

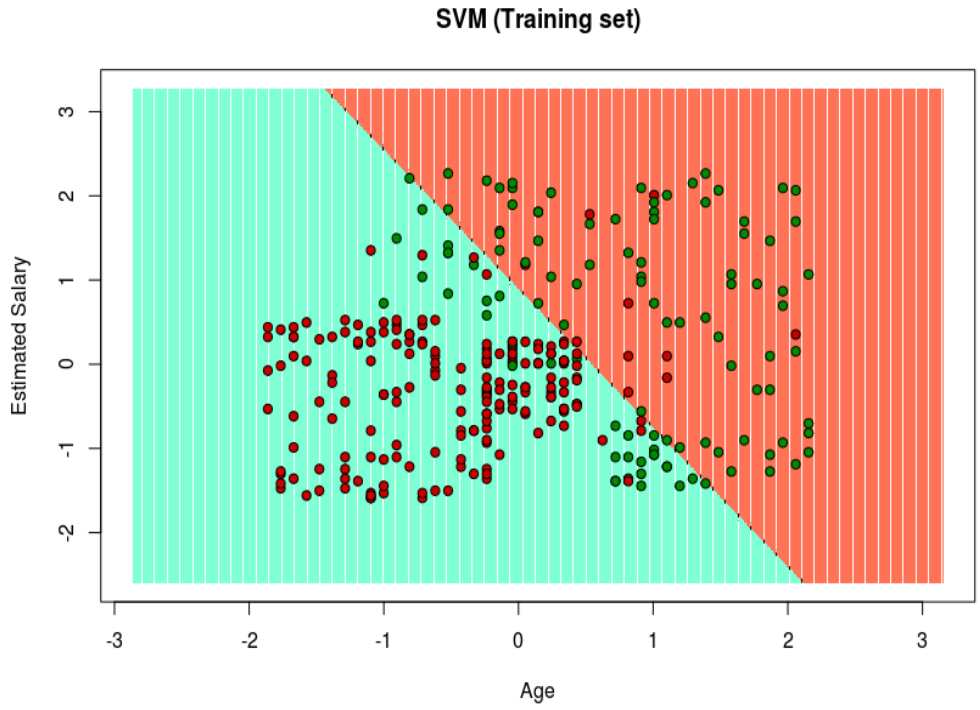
Plotting the training data set results

set = training_set

X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)

```
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Output:



Visualizing the Test set results

```
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
```

```
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = 'SVM (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Output: