Index

By K.Sravanthi

21761A4233

Week 1:

a) Installing R and RStudio

Install R on windows

Step – 1: Go to CRAN R project HYPERLINK "https://cran.r-project.org/" website.
Step – 2: Click on the **Download R for Windows** link. **Step – 3**: Click on the **base** subdirectory link or **install R for the first time** link.
Step – 4: Click **Download R X.X.X for Windows** (X.X.X stand for the latest version of R. eg: 3.6.1) and save the executable .exe file.
Step – 5: Run the **.exe** file and follow the installation instructions.
5.a. Select the desired language and then click

Next. **5.b.** Read the license agreement and click

Next.

5.c. Select the components you wish to install (it is recommended to install all the components). Click **Next**.

5.d. Enter/browse the folder/path you wish to install R into and then confirm by clicking **Next**

5.e. Select additional tasks like creating desktop shortcuts etc. then click **Next**.
5.f. Wait for the installation process to complete.

5.g. Click on **Finish** to complete the installation.

Install RStudio on Windows

Step – 1: With R-base installed, let's move on to installing RStudio. To begin, go to download RStudio and click on the download button for **RStudio desktop**.
Step – 2: Click on the link for the windows version of RStudio and save the .exe file.
Step – 3: Run the .exe and follow the installation instructions.
3.a. Click **Next** on the welcome window.

3.b. Enter/browse the path to the installation folder and click **Next** to proceed.
3.c. Select the folder for the start menu shortcut or click on do not create shortcuts and then click **Next**.
3.d. Wait for the installation process to complete.
3.e. Click **Finish** to end the installation.

b) Basic functionality of R, variable, data types in R

Variables in R Programming

A variable is a name given to a memory location, which is used to store values in a computer program. Variables in R programming can be used to store numbers (real and complex), words, matrices, and even tables. R is a dynamically programmed language which means that unlike other programming languages, we do not have to declare the data type of a variable before we can use it in our program.

R does not have a command for declaring a variable. A variable is created the moment you first assign a value to it. To assign a value to a variable, use the <- sign. To output (or print) the variable value, just type the variable name:

```
name <- " SRAVATHI"

name # output "SRAVANTHI"
```

Different Types of Data Types

In R, there are 6 basic data types:

- logical

- numeric
- integer
- complex
- character
- raw

1. Logical Data Type

The logical data type in R is also known as **boolean** data type. It can only have two values: TRUE and FALSE. For example,

```
bool1 <- TRUE

print(bool1)

print(class(bool1))

bool2 <- FALSE

print(bool2)

print(class(bool2))
```

Output

```
[1] TRUE

[1] "logical"
[1] FALSE

[1] "logical"
```

In the above example,

- *bool1* has the value TRUE,
- *bool2* has the value FALSE.

Here, we get "logical" when we check the type of both variables.

Note: You can also define logical variables with a single letter - T for TRUE or F for FALSE. For example,

```
is_weekend <- F

print(class(is_weekend)) # "logical
```

Data Type

In R, the numeric data type represents all real numbers with or without decimal values. For example,

```
# floating point values

weight <- 63.5

print(weight)

print(class(weight))

# real numbers

height <- 182

print(height)
print(class(height))
```

Output:

```
[1] 63.5

[1] "numeric"

[1] 182
```

[1] "numeric"

Here, both *weight* and *height* are variables of numeric type.

3. Integer Data Type

The integer data type specifies real values without decimal points. We use the suffix L to specify integer data. For example,

```
integer_variable <- 186L

print(class(integer_variable))
```

Output:

[1] "integer"

Here, 186L is an integer data. So we get "integer" when we print the class of *integer_variable*.

4. Complex Data Type

The complex data type is used to specify purely imaginary values in R. We use the suffix i to specify the imaginary part. For example,

```
# 2i represents imaginary part
complex_value <- 3 + 2i

# print class of complex_value

print(class(complex_value))
```

Output:

[1] "complex"

Here, 3 + 2i is of complex data type because it has an imaginary part 2i.

5. Character Data Type

The character data type is used to specify character or string values in a variable.

In programming, a string is a set of characters. For example, 'A' is a single character and "Apple" is a string.

You can use single quotes '' or double quotes "" to represent strings. In general, we use:

- '' for character variables
- "" for string variables

For example,

```
# create a string variable

fruit <- "Apple"

print(class(fruit))

# create a character variable

my_char <- 'A'
print(class(my_char))
```

Output

```
[1] "character"

[1] "character"
```

Here, both the variables - *fruit* and *my_char* - are of character data type.

6. Raw Data Type

A raw data type specifies values as raw bytes. You can use the following methods to convert character data types to a raw data type and vice-versa: • charToRaw() - converts character data to raw data

- rawToChar() - converts raw data to character data

For example,

```
# convert character to raw

raw_variable <- charToRaw("Welcome to LBRCE")

print(raw_variable)

print(class(raw_variable))

# convert raw to character

char_variable <- rawToChar(raw_variable)

print(char_variable)
print(class(char_variable))
```

Output

```
[1] 57 65 6c 63 6f 6d 65 20 74 6f 20 50 72 6f 67 72 61 6d 69

7a [1] "raw"

[1] "Welcome to LBRCE"

[1] "character"
```

In this program,

- We have first used the charToRaw() function to convert the string "Welcome to

Programiz" to raw bytes.

This is why we get "raw" as output when we print the class of *raw_variable*.

Week 2

- **Implement R script to show the usage of various operators available in R language**.

Types of the operator in R language

- Arithmetic Operators
- Logical Operators
- Relational Operators
- Assignment Operators
- Miscellaneous Operator

- **Arithmetic:** arithmetic operators are used with numeric values to perform

common mathematical operators.
Ex: +, -, *, /, ^,%%,%/%.
Program:

```
a<-10
b<-13
add=a+b
print(add)
sub=a-b
print(sub)
muL=10*13
print(muL)
div=a/b
print(div)
exponent=a^b
print(exponent)
```

output:

```
[1] 23
[1] -3
[1] 130
[1] 0.7692308
[1] 1e+13
```

- **Assignment:** assignment operators are used to assign values to variables.

```
my_var<-3
```

```
my_var<<-3
3->my_var
3->>my_var
my_var
0utput:[1] 3
```

- **Comparison :** comparison operators are used to compare two values.
  Ex: ==,!=,>,<,>=,<=.
  Program:

```
a<-15
b<-12
print (paste ("output 0f 15>12:", a>b))
print (paste ("output 0f 15<12:", a<b))
print (paste ("output 0f 15<=12:", a<=b))
print (paste ("output 0f 15>=12:", a>=b))
print (paste ("output 0f 15==12:", a==b))
print (paste ("output 0f 15! =12:", a! =b))
output:
[1] "output 0f 15>12: TRUE"
[1] "output 0f 15<12: FALSE"
[1] "output 0f 15<=12: FALSE"
[1] "output 0f 15>=12: TRUE"
[1] "output 0f 15==12: FALSE"
[1] "output 0f 15! =12: TRUE"
```

- Logical: logical operators are used to compare conditional statements.
  Ex: -: &, &&, |, ||,!
  Program:

```
num1<-c(0,1,23,24)
num2<-c(0,1,1,0)
num1 & num2
num1 && num2
num1 | num2
! num1
!!num2
!!num1
Output:
[1] FALSE TRUE TRUE TRUE
[1] TRUE FALSE FALSE FALSE
[1] FALSE TRUE TRUE FALSE
[1] FALSE TRUE TRUE TRUE
```

- **Miscellaneous**: Miscellaneous operators are used to manipulate data.

Ex: :,%in%,%*%.

Program:

```
print("Miscellaneous Operators")
print("Colon operator")
print(2:8)
}
```

OUTPUT:

"Miscellaneous Operators"

[1] "Colon operator"

[1] 2 3 4 5 6 7 8

b) Implement R script to read person‘s age from keyboard and display whether he is eligible for voting or not.

Program:

```
{
 age<-as.integer(readline(prompt = "Enter your age: "))
if (age>=18)
{
 print(paste("You are Eligible for voting:", age))
}
else
{
 print(paste("You are not Eligible for voting:", age))
}
}
```

Output:

Enter your age: 19
[1] "You are Eligible for voting: 15"

Enter your age: 15
[1] "You are not Eliginble for voting: 15"

c) Implement R script to find biggest number between two numbers.

Program:

```
{
x <-as. integer(readline(prompt = "Enter first number:"))
y <-as.integer(readline(prompt = "Enter second number:"))
z <-as.integer(readline(prompt = "Enter third number:"))
if (x > y && x > z)
 {
print (paste("Greatest is:", x))
}
else if (y > z)
{
 print(paste("Greatest is:", y))
}
 else
 {
print(paste("Greatest is:", z))
}
 }
```

Output:

Enter first number:55

Enter second number:10

Enter third number:69

[1] "Greatest is: 69"

d) Implement R script to check the given year is leap year or not.

```
y = as.numeric(readline(prompt = 'year:'))
if ((y %% 4) == 0){
 if ((y %% 100) == 0){
```

```
if ((y%% 400) ==0){
print(paste(y,'is a leap year'))
}
else {
print(paste(y,"is not a leap year"))
}
}
else{
print(paste(y,"is a leap year"))
}
}else{
print(paste(y,"is not a leap year"))
}
```

OUTPUT:

Year:2000

[1] "2000 is a leap year"

Year:1999

[1] "1999 is not a leap year"

Week 3

a) Implement R Script to create a list

A list in R is a generic object consisting of an ordered collection of objects. Lists are one-dimensional, heterogeneous data structures. Procedure:

```
list_data<-list(c("Jan","Feb","Mar"),matrix(c(3,9,5,1,-2,8), nrow=
2),list("green",12.3))
names(list_data) <-c ("1st Quarter", "A_Matrix", "A Inner list")
print(list_data)
```

output:

$`1st Quarter`

[1] "Jan" "Feb" "Mar"
$A_Matrix
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8

$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3

b) Implement R Script to access elements in the list.

list_data<-list(c("Jan","Feb","Mar"),matrix(c(3,9,5,1,
- 2,8),nrow=2),list("green",12.3))
names(list_data)<-c("1stQuarter","A_Matrix","AInnerlist"
) print(list_data[1])
print(list_data[3])
print(list_data$A_Matrix)

output:

$`1stQuarter`
[1] "Jan" "Feb" "Mar"
print(list_data[3])
$AInnerlist
$AInnerlist[[1]]
[1] "green"
$AInnerlist[[2]]
[1] 12.3
print(list_data$A_Matrix)
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8

Week 4

Implement R script to perform following operations:
a) various operations on vectors

Vector:

Vectors are the most basic data types in R. Even a single object created is also stored in the form of a vector

Program:

```
x2<-c("red","green","yellow","blue")
class(x2)
```

output:

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

Program:

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

Output:

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

Program:

```
t<-c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u<-t[c(2,3,6)]
print(u)
```

output:

```
[1] "Mon" "Tue" "Fri"
```

Program:

```
v1 <-c(3,8,4,5,0,11)
v2 <-c(4,11,0,8,1,2)
add.result<-v1+v2
print(add.result)
sub.result<-v1-v2
print(sub.result)
multi.result<-v1*v2
print(multi.result)
divi.result<-v1/v2
print(divi.result)
```

output:

```
[1] 7 19 4 13 1 13
[1] -1 -3 4 -3 -1 9
[1] 12 88 0 40 0 22
[1] 0.7500000 0.7272727 Inf 0.6250000 0.0000000 5.5000000
```

b) Finding the sum and average of given numbers using arrays.

To compute the average of values, R provides a pre-defined

function mean (), sum ().

Program:

```
vec=c(10,7,5,8,4)
print("sum of the vector")
print(sum(vec))
print("mean of the vector")
print(mean(vec))
```

Output:

```
> vec=c(10,7,5,8,4)
> print("sum of the vector")
[1] "sum of the vector"
> print(sum(vec))
[1] 34
> print("mean of the vector")
[1] "mean of the vector"
> print(mean(vec))
[1] 6.8
```

c) To display elements of list in reverse order.

we use rev () function in this program to reverse the lists.

Program:

```
x<-list ("a","b","c","d")
result=rev(x)
print(result)
l=c(1:5)
rev.default(l)
rev(l)
```

output:

```
[[1]]
[1] "d"

[[2]]
[1] "c"

[[3]]
[1] "b"
```

[[4]]
[1] 5 4 3 2 1
[1] 5 4 3 2 1

d) Finding the minimum and maximum elements in the array.

```
numbers <- c(2,4,6,8,10)

 max(numbers)

characters <- c("s", "a", "p", "b")

max(characters)
```

OUTPUT:

[1] 10

[1] “s”

[1] 2

[1] “a”

Week 5

a) Implement R Script to perform various operations on matrices

```
data<-c(1,2,3,4,5,6,7,8,9)
a<-matrix(data,nrow=3,ncol=3)
data<-c(10,11,12,13,14,15,16,17,18)
b<-matrix(data,nrow=3,ncol=3)
c<-a+b
d<-a-b
e<-a %*% b
print("Matrix a")
print(a)
print ("Matric b")
print(b)
print(“Addition of a Matrix”)
print(c)
print(“Subtraction of a
Matrix”) print(d)
print ("Matrix multiplcation
result") print(e)
```

Output:

```
[1] "Matrix a"
 [,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
[1] "Matric b"
 [,1] [,2] [,3]
[1,] 10 13 16
[2,] 11 14 17
[3,] 12 15 18
[1] "Addition of a Matrix"
 [,1] [,2] [,3]
[1,] 11 17 23
[2,] 13 19 25
[3,] 15 21 27
[1] "Subtraction of a Matrix"
 [,1] [,2] [,3]
[1,] -9 -9 -9
[2,] -9 -9 -9
[3,] -9 -9 -9
[1] "Matrix multiplcation result"
> print(c)
 [,1] [,2] [,3]
[1,] 11 17 23
[2,] 13 19 25
[3,] 15 21 27
```

b) Implement R Script to extract the data from dataframes.

Program:

```
emp_data<-
data.frame(emp_id=c(1:5),emp_name=c("Ayra","Revanth","Priya","
Padma","Sasii"),salary=c(900.56,589.0,712.5,642.32,895.52)

,start_date=as.Date(c("2012-01-01","2013-09-23","2014-11-
15","2014-05-11","2015-03-27"))

,stringsAsFactors = FALSE )

print(emp_data)

str(emp_data)

result<-data.frame(emp_data$emp_name,emp_data$salary)

print(result)

result<-emp_data[1:2,]

print(result)

rbind(emp_data,list(1,"Paul",120,"2013-09-23"))
emp_data$state<-c("Vij","Hyd","VZA","FL","NY");emp_data
```

OUTPUT:

```
print(emp_data)

 emp_id emp_name salary start_date

1 1 Ayra 900.56 2012-01-01

2 2 Revanth 589.00 2013-09-23

3 3 Priya712.50 2014-11-15

4 4 Padma642.32 2014-05-11

5 5 Sasi 895.52 2015-03-27

> str(emp_data)

'data.frame': 5 obs. of 4 variables:

 $ emp_id : int 1 2 3 4 5
```

$ emp_name : chr "Ayra" "Revanth" "Priya" "Padma" ,”Sasi” $ salary : num

501 589 712 642 896

$ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014- 05-11" ...

>

result<-data.frame(emp_dataemp_name,emp_datasalary)

> print(result)

emp_data.emp_name emp_data.salary

1 Ayra900.56

2 Revanth 589.00

3 Priya 712.00

4 Padma642.32

5 Sasi895.52

> result<-emp_data[1:2,]

> print(result)

emp_id emp_name salary start_date

1 1 Ayra900.56 2012-01-01

2 2 Revanth 589.00 2013-09-23

>

rbind(emp_data,list(1,"Paul",120,"2013-09-23"))

emp_id emp_name salary start_date

1 1 Ayra 900.56 2012-01-01

2 2 Revanth 589.00 2013-09-23

3 3 Priya712.50 2014-11-15

4 4 Padma 642.32 2014-05-11

5 5 Sasi 895.52 2015-03-27

6 1 Paul 120.00 2013-09-23

> emp_data$state<-c("Vij","Hyd","VZA","FL","NY");emp_data

emp_id emp_name salary start_date state

1 1 Ayra 900.56 2012-01-01 Vij

2 2 Revanth 589.00 2013-09-23 Hyd

3 3 Priya712.50 2014-11-15 VZA

4 4 Padma 642.32 2014-05-11 FL

5 5 Sasi 895.52 2015-03-27 NY

c) Write R script to display file contents.

```
# R program reading a text file

# Read a text file using read.delim()

myData = read.delim("P:/oops/example.txt", header = FALSE)

print(myData)
```

output:

Hi, this is r programming language.

d) Write R script to copy file contents from one file to another.

```
library(readr)

f1=read_file("P:/oops/example.txt")

print(f1)

 write_file(f1,"f2.txt")

res=read_file("f2.txt")

 print(res)
```

output:

Hi, this is r programming language.

Week 6

a) Write an R script to find basic descriptive statistics using summary, str, quartile function onmtcars& cars datasets

```
data("mtcars")

head(mtcars,10)

summary(mtcars)
```

```
str(mtcars)

colMeans(mtcars)

colSums(mtcars)

rowSums(mtcars[2,])

quantile(mtcars $ mpg)
```

output:

b)Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset

```
data("iris")
head(iris)
nrow(iris)
ncol(iris)
iris[1,2:5]
iris[,c(2,5)]
colnames(iris)
rownames(iris)
m=aggregate(iris[,1:4],by=list(iris $
Species),FUN=mean,na.rm=TRUE) p=aggregate(iris[,1:4],by=list(iris $
Species),FUN=sum,na.rm=TRUE) q=aggregate(iris[,1:4],by=list(iris $
Species),FUN=length) r=aggregate(iris[,1:4],by=list(iris $
Species),FUN=max,na.rm=TRUE) s=aggregate(iris[,1:4],by=list(iris $
Species),FUN=min,na.rm=TRUE) m
p
q
r
```

output:

Week 7

a) Reading different types of data sets (.txt, .csv) from Web or disk and writing in file in specific disk location.

```
Data1 = read.delim("C.txt", header = TRUE)
print(Data1)
```

Output:

```
 print(Data1)
 outlook.temperature.humidity.windy.play
1 1 overcast hot high FALSE yes
2 2 overcast cool normal TRUE yes
3 3 overcast mild high TRUE yes
4 4 overcast hot normal FALSE yes
```

5 5 rainy mild high FALSE yes

6 THIS IS EXTRACTED FROM .TXT FILE

Data3<- read.csv(“weather.csv”, header=TRUE)

Data3

OUTPUT:

outlook temperature humidity windy play

1 overcast hot high FALSE yes

2 overcast cool normal TRUE yes

3 overcast mild high TRUE yes

4 overcast hot normal FALSE yes

5 rainy mild high FALSE yes

6 rainy cool normal FALSE yes

7 rainy cool normal TRUE no

8 rainy mild normal FALSE yes

9 rainy mild high TRUE no

10 sunny hot high FALSE no

11 sunny hot high TRUE no

12 sunny mild high FALSE no

13 sunny cool normal FALSE yes

14 sunny mild normal TRUE yes

#writing data to a file

write.table(weather, file="New.txt", quote=F)

output:

outlook temperature humidity windy play

1 overcast hot high FALSE yes

2 overcast cool normal TRUE yes

3 overcast mild high TRUE yes

4 overcast hot normal FALSE yes

5 rainy mild high FALSE yes

6 rainy cool normal FALSE yes

7 rainy cool normal TRUE no

8 rainy mild normal FALSE yes

9 rainy mild high TRUE no

10 sunny hot high FALSE no

11 sunny hot high TRUE no

12 sunny mild high FALSE no

13 sunny cool normal FALSE yes

14 sunny mild normal TRUE yes

b) Reading Excel data sheet in R.

```
library(readxl)

Data <- read_excel("studentMarks.xlsx")

Data
```

number_courses	time_study	Marks
3	4.508	19.202
4	0.096	7.734
4	3.133	13.811
6	7.909	53.018
8	7.811	55.299
6	3.211	17.822
3	6.063	29.889
5	3.413	17.264
4	4.41	20.348
3	6.173	30.862

3	7.353	42.036
7	0.423	12.132
7	4.218	24.318
3	4.274	17.672
3	2.908	11.397
4	4.26	19.466
5	5.719	30.548
8	6.08	38.49
6	7.711	50.986
8	3.977	25.133
4	4.733	22.073
6	6.126	35.939
5	2.051	12.209
7	4.875	28.043

c) Reading XML dataset in R

```
#program to read XML file.

library("XML")
library("methods")

data <- xmlParse(file = "sample.xml")

print(data)
```

output:

```
1

kavyasamitha

620
```

IT

2

avramya

440

Commerce

3

bindhu

600

Humanities

4

haritha

660

IT

5

sandhya

560

IT

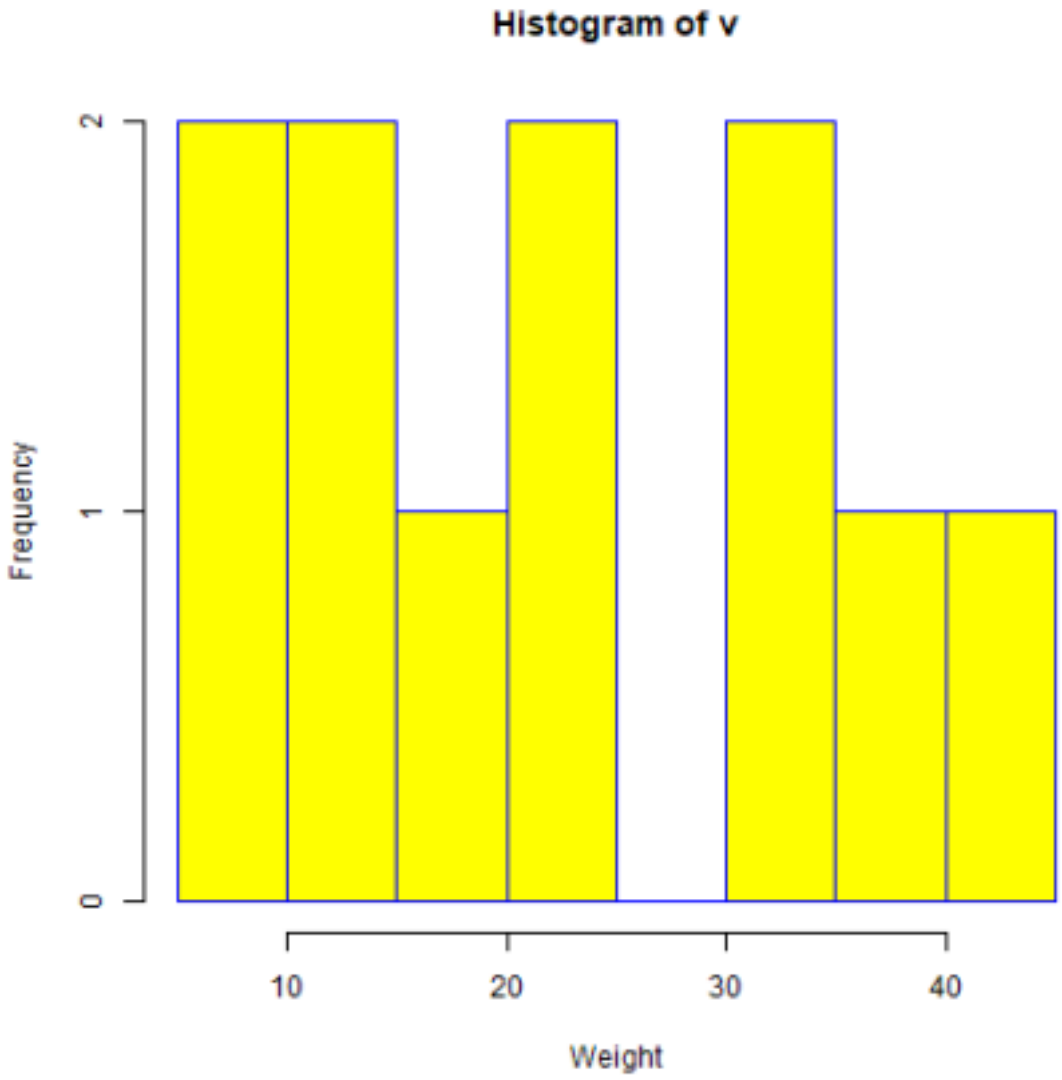
Week 8

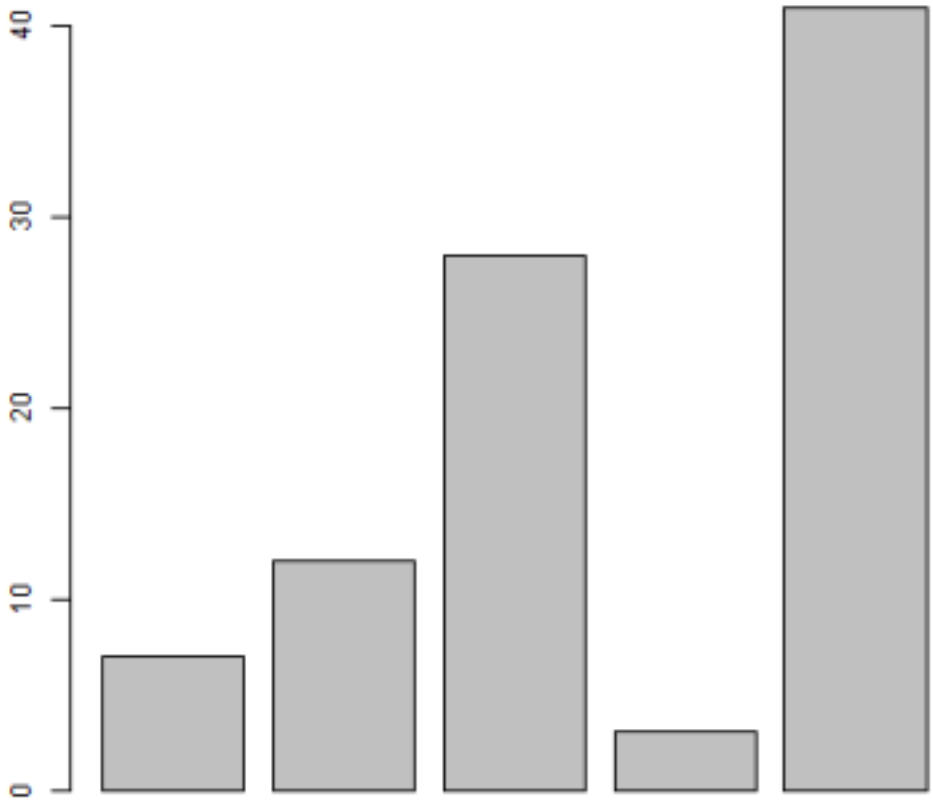
a) Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram (Introduction to ggplot2 graphics)

```
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

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

# Plot the chart.
pie(x,labels)

# Save the file.
dev.off()
# Create the data for the chart
H <- c(7,12,28,3,41)
```

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

# Plot the bar chart
barplot(H)

# Save the file
dev.off()
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

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

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

# Save the file.
dev.off()
# Get the input values.
input <- mtcars[,c('wt','mpg')]

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

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and
30.
plot(x = input$wt,y = input$mpg,
    xlab = "Weight",
    ylab = "Milage",
    xlim = c(2.5,5),
    ylim = c(15,30),
    main = "Weight vs Milage"
)

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

Output:


Histogram of v
Frequency
0
1
2
10
20
30
40
Weight



b) Implement R Script to perform mean, median, mode, range, summary, variance, standard deviation operations. Median: The middle value of the data set.

Mean: Calculate sum of all the values and divide it with the total number of values in the data set.
Median: The middle value of the data set.

```
x<-c(1,2,3,4,5,6,1,2,3,4,9,2,6,1,2,7,7,5,6,7)

#Mean

mean.result=mean(x)

print(mean.result)

#Median

median.result=median(x)

print(median.result)
```

OUTPUT:

[1] 4.15

[1] 4

Mode:

The most occurring number in the data set. For calculating mode, there is no default function in R. So, we have to create our own custom function.

```
mode <- function(x) {
 ux <- unique(x)
 ux[which.max(tabulate(match(x, ux)))]
 }
x<-c(1,2,3,4,5,6,1,2,3,4,9,2,6,1,2,7,7,5,6,7)
 mode.result = mode(x) # calculate mode (with our custom function named 'mode')
 print (mode.result)
```

Variance:

How far a set of data values are spread out from their

mean x<-c(1,2,3,4,5,6,1,2,3,4,9,2,6,1,2,7,7,5,6,7)

```
variance.result = var(x)
 print (variance.result)
```

OUTPUT:

print (variance.result)

[1] 5.818421

Standard Deviation:

A measure that is used to quantify the amount of variation or dispersion of a set of data values.

```
x<-c(1,2,3,4,5,6,1,2,3,4,9,2,6,1,2,7,7,5,6,7)
sd.result = sqrt(var(x)) # calculate standard deviation
print (sd.result)
```

OUTPUT:

print (sd.result)

[1] 2.41214

Range: The result is that you have range chart of values covered by the dataset

x<-c(1,2,3,4,5,6,1,2,3,4,9,2,6,1,2,7,7,5,6,7)

range(x)

OUTPUT:

range(x)

[1] 1 9

Week 9

a) Implement R Script to perform Normal, Binomial distributions.

Normal Distribution :

Normal Distribution is a probability function used in statistics that tells about how the data values are distributed. It is the most important probability distribution function used in statistics because of its advantages in real case scenarios

In R, there are 4 built-in functions to generate normal distribution:

· **dnorm()**

dnorm(x, mean, sd)

· **pnorm()**

pnorm(x, mean, sd)

· **qnorm()**

qnorm(p, mean, sd)

· **rnorm()**

rnorm(n, mean, sd)

Normal Distribution :

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.
x <- seq(-10, 10, by = .1)
# Choose the mean as 2.5 and standard deviation as 0.5.
```

```
y <- dnorm(x, mean = 2.5, sd = 0.5)
# Give the chart file a name.
png(file = "dnorm.png")

plot(x,y)
# Save the file.
dev.off()
```

O/p:

Binomial Distribution :

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments

Binomial Distribution :

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)
# Create the binomial distribution.
y <- dbinom(x,50,0.5)
# Give the chart file a name.
png(file = "dbinom.png")
# Plot the graph for this sample.
plot(x,y)
# Save the file.
dev.off()
```

O/p:

b) Implement R Script to perform correlation, Linear and multiple regression.

Linear Regression:

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

lm() Function

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

Syntax

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

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

Liner Regression :

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
print(relation)
```

O/p:

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

Multiple Regression:

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

lm() Function

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

Syntax

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

```
lm(y ~ x1+x2+x3...,data)
```

Multiple Regression :

```
input <- mtcars[,c("mpg","disp","hp","wt")]
# Create the relationship model.
model <- lm(mpg~disp+hp+wt, data = input)
# Show the model.print(model)
# Get the Intercept and coefficients as vector elements.
cat("# # # # The Coefficient Values # # # ","\n")

a <- coef(model)[1]print(a)
Xdisp <- coef(model)[2]Xhp <- coef(model)[3]Xwt <- coef(model)[4]
print(Xdisp)print(Xhp)print(Xwt)
```

O/p:

```
Coefficients:
(Intercept) disp hp wt
  37.105505 -0.000937 -0.031157 -3.800891

# # # # The Coefficient Values # # #
(Intercept)
   37.10551
       disp
-0.0009370091
         hp
-0.03115655
         wt
-3.800891
```

Week 10

Introduction to Non-Tabular Data Types: Time series, spatial data, Network data. Data Transformations: Converting Numeric Variables into Factors, Date Operations, String Parsing, Geocoding.

Introduction to Non-Tabular Data Types:

Time Series :

Time series in R is defined as a series of values, each associated with the timestamp also measured over regular intervals (monthly, daily) like weather forecasting and sales analysis. The R stores the time series data in the time series object and is created using the ts() function as a base distribution.

The basic syntax for **ts()** function in time series analysis is −
timeseries.object.name <- ts(data, start, end, frequency)
Following is the description of the parameters used −
• **data** is a vector or matrix containing the values used in the time series. • **start** specifies the start time for the first observation in time series. • **end** specifies the end time for the last observation in time series. • **frequency** specifies the number of observations per unit time

Spatial data:

For **spatial** data, they enable R to process spatial data formats, carry out analysis tasks and create some of the maps that follow. Bascially, without packages, R would be very limited.

Network data:

```
install.packages("igraph")
```

```
## Load package
library(igraph)
## Create undirected graphs
g <- graph_from_literal(1-2, 1-3, 1-7, 3-4, 2-3, 2-4, 3-5, 4-5, 4-6, 4-7, 5-6, 5-8,
6- 7, 7-8)
## Create directed graphs using addition or substraction operators
dg <- graph_from_literal(JFK-+PEK, JFK-+CDG, PEK++CDG) ## check
out the vertices and the order
V(g)
vcount(g)
## check out the edges and the size
E(g)
ecount(g)
## summary
g$name <- "undirected graph"
print_all(g)
plot(g)
dg$name <- "directed graph"
print_all(dg)
plot(dg)
V(g)$name <-c("Adam","Judy","Bobby","Sam","Frank","Jay","Tom","Jerry")
plot(g)
```

Data Transformations:

Converting Numeric Variables into Factors:

```
# Data Vector 'V'
V = c("North", "South", "East", "East")
# Convert vector 'V' into a factor
drn <- factor(V)
# Converting a factor into a numeric vector
as.numeric(drn)# Creating a Factor
soap_cost <- factor(c(29, 28, 210, 28, 29))
# Converting Factor to Numeric
as.numeric(soap_cost)
```

O/p:
[1] 2 3 1 1
[1] 2 1 3 1 2

Date Operations :

```
xd <- as.Date("2012-07-27")
```

```
> xd
[1] "2012-07-27"
> str(xd)
Date[1:1], format: "2012-07-27"
> weekdays(xd)
[1] "Friday"
> xd + 7
[1] "2012-08-03"
> xd + 0:6
[1] "2012-07-27" "2012-07-28" "2012-07-29"
"2012-07-30" [5] "2012-07-31" "2012-08-01"
"2012-08-02" weekdays(xd + 0:6)
[1] "Friday" "Saturday" "Sunday" "Monday"
[5] "Tuesday" "Wednesday" "Thursday"
startDate <- as.Date("2012-01-01")
> xm <- seq(startDate, by="2 months", length.out=6)
> xm
[1] "2012-01-01" "2012-03-01" "2012-05-01"
"2012-07-01" [5] "2012-09-01" "2012-11-01"
months(xm)
[1] "January" "March" "May" "July"
[5] "September" "November"
> quarters(xm)
[1] "Q1" "Q1" "Q2" "Q3" "Q3" "Q4"
```

String Parsing

```
fruits <- c(
"apples and oranges and pears and
bananas", "pineapples and mangos and
guavas"
)
str_split(fruits, " and ")
#> [[1]]
#> [1] "apples" "oranges" "pears" "bananas"
#>
#> [[2]]
#> [1] "pineapples" "mangos" "guavas"
#>
str_split(fruits, " and ", simplify = TRUE)
#> [,1] [,2] [,3] [,4]
#> [1,] "apples" "oranges" "pears" "bananas"
#> [2,] "pineapples" "mangos" "guavas" ""
# Specify n to restrict the number of possible matches
str_split(fruits, " and ", n = 3)
```

```
#> [[1]]
#> [1] "apples" "oranges" "pears and bananas"
#>
#> [[2]]
#> [1] "pineapples" "mangos" "guavas"
#>
str_split(fruits, " and ", n = 2)
#> [[1]]
#> [1] "apples" "oranges and pears and bananas"
#>
#> [[2]]
#> [1] "pineapples" "mangos and guavas"
#>
# If n greater than number of pieces, no padding occurs
str_split(fruits, " and ", n = 5)
#> [[1]]
#> [1] "apples" "oranges" "pears" "bananas"
```

Geocoding :

Geocoding can be simply achieved in R using the geocode() function from the ggmap library. The geocode function uses Googles Geocoding API to turn addresses from text to latitude and longitude pairs very simply.

There is a usage limit on the geocoding service for free users of 2,500 addresses per IP address per day.

Week 11

Introduction Dirty data problems: Missing values, data manipulation, duplicates, forms of data dates, outliers, spelling

Introduction Dirty data problems:

Missing values:
A missing value is one whose value is unknown. Missing values are represented in R by the NA symbol. NA is a special value whose properties are different from other values. NA is one of the very few reserved words in R: you cannot give anything this name.

Data Manipulation:

Data Manipulation in R In a data analysis process, the data has to be altered, sampled, reduced or elaborated. Such actions are called data manipulation. Data has to be manipulated many times during any kind of analysis process.

Duplicated() in R:

The duplicated() is a built-in R function that determines which elements of a vector or data frame are duplicates of elements with smaller subscripts and returns a logical vector indicating which elements (rows) are duplicates.

Syntax

duplicated(data, incomparables = FALSE, fromLast = FALSE, nmax = NA, …)

Parameters

data: It is a vector or a data frame or an array or NULL.

incomparables: It is a vector of values that cannot be compared. FALSE is a special value, meaning that all values can be compared and maybe the only value accepted for methods other than the default. It will be coerced internally to the same type as data.

fromLast: It is the logical argument that indicates if duplication should be considered from the reverse side; for example, the last (or rightmost) of identical elements would correspond to duplicated = FALSE.

Forms of data dates :

To change the date formats in R, use the format() function. For example, if you want to get the date than the standard %Y-%m-%d, use the format() function from the base package.

```
dates <- c("11/20/80", "11/20/91", "11/20/1993", "09/10/93")
dt <- as.Date(dates, "%m/%d/%y")
```

dt

```
cat("After formatting a date in other format", "\n")

fmt <- format(dt, "%a %b %d")

print(fmt)
```

O/p:

```
[1] "1980-11-20" "1991-11-20" "2019-11-20" "1993-09-10"
```

After formatting a date in other format[1] "Thu Nov 20" "Wed Nov 20" "Wed Nov 20" "Fri Sep 10"

Outliers:

An outlier is an observation that lies abnormally far away from other values in a dataset. Outliers can be problematic because they can affect the results of an analysis. This tutorial explains how to identify and remove outliers in R.

Spelling:

The main purpose of this package is to quickly find spelling errors in R packages. The

spell_check_package()

function extracts all text from your package manual pages and vignettes, compares it against a language (e.g. en_US or en_GB), and lists potential errors in a nice tidy format:

Week 12

Data sources: SQLite examples for relational databases, Loading SPSS and SAS files, Reading from Google Spreadsheets, API and web scraping examples

Data sources:

SQLite examples for relational databases :

```
# Create toy data frames
car <- c('Camaro', 'California', 'Mustang', 'Explorer')

make <- c('Chevrolet','Ferrari','Ford','Ford')

df1 <- data.frame(car,make)

car <- c('Corolla', 'Lancer', 'Sportage', 'XE')

make <- c('Toyota','Mitsubishi','Kia','Jaguar')

df2 <- data.frame(car,make)

# Add them to a list

dfList <- list(df1,df2)

# Write a table by appending the data frames inside the list
```

```
for(k in 1:length(dfList)){
        dbWriteTable(conn,"Cars_and_Makes", dfList[[k]], append = TRUE)
}
# List all the Tables
dbListTables(conn)
```

Loading SPSS Files in R:

The easiest way to import SPSS files into R is to use the **read_sav()** function from the haven library.

This function uses the following basic syntax:

```
data <- read_sav('C:/Users/User_Name/file_name.sav')
```

Loading SAS Files in R:

The easiest way to import SAS files into R is to use the **read_sas()** function from the haven library.

This function uses the following basic syntax:

```
data <- read_sas('C:/Users/User_Name/file_name.sas7bdat')
```

Reading Google Sheets in R :

You can read google sheets data in R using the package **'googlesheets4'**. This package will allow you to get into sheets using R.

First you need to install the **'googlesheets4'** package in R and then you have to load the library to proceed further.

```
#Install the required package
install.packages('googlesheets4')
#Load the required library
library(googlesheets4)
```

API:

An API is a set of methods and tools that allow one's to Query and retrieve

data dynamically.Spotify,twitter,facebook and many other companies provide free API'S that enable developers to access the information they store on their services,other change for access to their API'S.

Web Scrapping:

A Lot of data isn't accessible through datasets or api's but rather exists on the internet as web pages.

So, through web scrapping one can access the data without waiting for the provider to create an API.