Experiment 1

week 1

Installing R and R Studio Basic functionality of R, variable, data types in R

Installing R and R Studio

Step – 1: Go to CRAN R Project Website.

Step – 2: Click on the Download for (Mac) OS X link.

Step – 3: Click on the link for the pkg file of the latest R version and save it.

Step – 4: Double click the downloaded file and follow installation instructions.

Local Environment Setup

If you are still willing to set up your environment for R, you can follow the steps given below.

Windows Installation

- You can download the Windows installer version of R from R-3.2.2 for Windows (32/64 bit) and save it in a local directory.
- As it is a Windows installer (.exe) with a name "R-version-win.exe". You can just double click and run the installer accepting the default settings. If your Windows is 32-bit version, it installs the 32-bit version. But if your windows is 64-bit, then it installs both the 32-bit and 64-bit versions.
- After installation you can locate the icon to run the Program in a directory structure "R\R3.2.2\bin\i386\Rgui.exe" under the Windows Program Files. Clicking this icon brings up the R-GUI which is the R console to do R Programming.

Linux Installation

- R is available as a binary for many versions of Linux at the location R Binaries.
- The instruction to install Linux varies from flavor to flavor. These steps are mentioned under each type of Linux version in the mentioned link. However, if you are in a hurry, then you can use yum command to install R as follows −
- $ yum install R
- Above command will install core functionality of R programming along with standard packages, still you need additional package, then you can launch R prompt as follows −
- $ R
- R version 3.2.0 (2015-04-16) -- "Full of Ingredients"

- Copyright (C) 2015 The R Foundation for Statistical Computing
- Platform: x86_64-redhat-linux-gnu (64-bit)
- R is free software and comes with ABSOLUTELY NO WARRANTY.
- You are welcome to redistribute it under certain conditions
- Type 'license()' or 'licence()' for distribution details
- R is a collaborative project with many contributors.
- Type 'contributors()' for more information and
- 'citation()' on how to cite R or R packages in publications.
- Type 'demo()' for some demos, 'help()' for on-line help, or
- 'help.start()' for an HTML browser interface to help.
- Type 'q()' to quit R.
- Now you can use install command at R prompt to install the required package. For example, the following command will install plotrix package which is required for 3D charts.
- > install.packages("plotrix")

Introduction to RStudio

1. Installation of RStudio

If you have used RStudio before, I recommend **uninstalling the old versions** and **installing the latest version of RStudio** since some libraries that we'll install might not be compatible with older versions of R.

2.2 Install the latest RStudio

RStudio provides a friendlier working environment for R

Download RStudio Desktop. Select an isntaller based on your OS and then install.

2.3 RStudio Layout

The RStudio interface consists of several windows

- Bottom left: **command window**. Here you can type simple commands after the ">" prompt and R will then execute your command. This is the most important window, because this is where R actually does stuff.
- Top left: **script window**. Collections of commands (scripts) can be edited and saved. When you don't get this window, you can open it with [File] – [New] – [R script]. Just typing a command in the editor window is not enough, it has to get into the command window before R executes the command. If you want to run a line from the script window (or the whole script), you can click Run or press CTRL+ENTER to send it to the command window.

- Top right: **workspace / history window**. In the workspace window you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been typed before.
- Bottom right: **files / plots / packages / help window**. Here you can open files, view plots (also previous plots), install and load packages or use the help function.

You can change the size of the windows by dragging the grey bars between the windows.

Variable, data types in R

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

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

Variable Assignment

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

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)

# Assignment using leftward operator.
var.2 <- c("learn","R")

# Assignment using rightward operator.
c(TRUE,1) -> var.3

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

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

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

Note − The vector c(TRUE,1) has a mix of logical and numeric class. So logical class is coerced to numeric class making TRUE as 1

Data Type of a Variable

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

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

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

var_x <- 27L
cat("     Next the class of var_x becomes ",class(var_x),"\n")
```

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

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

Finding Variables

To know all the variables currently available in the workspace we use the **ls()** function. Also the ls() function can use patterns to match the variable names.

```
print(ls())
```

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

```
[1] "my var"       "my_new_var" "my_var"       "var.1"
[5] "var.2"        "var.3"        "var.name"     "var_name2."
[9] "var_x"        "varname"
```

Note − It is a sample output depending on what variables are declared in your environment.

The ls() function can use patterns to match the variable names.

```
# List the variables starting with the pattern "var".
print(ls(pattern = "var"))
```

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

```
[1] "my var"       "my_new_var" "my_var"       "var.1"
[5] "var.2"        "var.3"        "var.name"     "var_name2."
[9] "var_x"        "varname"
```

The variables starting with **dot(.)** are hidden, they can be listed using "all.names = TRUE" argument to ls() function.

```
print(ls(all.name = TRUE))
```

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

```
[1] ".cars"          ".Random.seed" ".var_name"      ".varname"       ".varname2"
[6] "my var"          "my_new_var"     "my_var"          "var.1"          "var.2"
[11]"var.3"          "var.name"       "var_name2."     "var_x"
```

Deleting Variables

Variables can be deleted by using the **rm()** function. Below we delete the variable var.3. On printing the value of the variable error is thrown.

```
rm(var.3)
print(var.3)
```

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

```
[1] "var.3"
Error in print(var.3) : object 'var.3' not found
```

All the variables can be deleted by using the **rm()** and **ls()** function together.

```
rm(list = ls())
print(ls())
```

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

```
character(0)
```

Experiment 2

week 2

2a) Implement R script to show the usage of various operators available in R language

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

We have the following types of operators in R programming −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

```
# R program to illustrate the use of Arithmetic operators

vec1 <- c(0, 2)

vec2 <- c(2, 3)

# Performing operations on Operands

cat ("Addition of vectors :", vec1 + vec2, "\n")

cat ("Subtraction of vectors :", vec1 - vec2, "\n")

cat ("Multiplication of vectors :", vec1 * vec2, "\n")

cat ("Division of vectors :", vec1 / vec2, "\n")

cat ("Modulo of vectors :", vec1 %% vec2, "\n")

cat ("Power operator :", vec1 ^ vec2)
```

Output:

```
Addition of vectors : 2 5

Subtraction of vectors : -2 -1

Multiplication of vectors : 0 6

Division of vectors : 0 0.6666667

Modulo of vectors : 0 2

Power operator : 0 8
```

```
# R program to illustrate the use of Logical operators

vec1 <- c(0,2)

vec2 <- c(TRUE,FALSE)

# Performing operations on Operands

cat ("Element wise AND :", vec1 & vec2, "\n")

cat ("Element wise OR :", vec1 | vec2, "\n")

cat ("Logical AND :", vec1 && vec2, "\n")

cat ("Logical OR :", vec1 || vec2, "\n")

cat ("Negation :", !vec1)
```

Output:

```
Element wise AND : FALSE FALSE

Element wise OR : TRUE TRUE

Logical AND : FALSE

Logical OR : TRUE

Negation : TRUE FALSE
```

```
# R program to illustrate the use of Relational operators

vec1 <- c(0, 2)

vec2 <- c(2, 3)

# Performing operations on Operands

cat ("Vector1 less than Vector2 :", vec1 < vec2, "\n")

cat ("Vector1 less than equal to Vector2 :", vec1 <= vec2, "\n")

cat ("Vector1 greater than Vector2 :", vec1 > vec2, "\n")
```

```
cat ("Vector1 greater than equal to Vector2 :", vec1 >= vec2, "\n")

cat ("Vector1 not equal to Vector2 :", vec1 != vec2, "\n")
```

Output:

```
Vector1 less than Vector2 : TRUE TRUE

Vector1 less than equal to Vector2 : TRUE TRUE

Vector1 greater than Vector2 : FALSE FALSE

Vector1 greater than equal to Vector2 : FALSE FALSE

Vector1 not equal to Vector2 : TRUE TRUE
```

```
# R program to illustrate the use of Assignment operators

vec1 <- c(2:5)

c(2:5) ->> vec2

vec3 <<- c(2:5)

vec4 = c(2:5)

c(2:5) -> vec5

 # Performing operations on Operands

cat ("vector 1 :", vec1, "\n")

cat("vector 2 :", vec2, "\n")

cat ("vector 3 :", vec3, "\n")

cat("vector 4 :", vec4, "\n")

cat("vector 5 :", vec5)
```

Output:

```
vector 1 : 2 3 4 5

vector 2 : 2 3 4 5
```

vector 3 : 2 3 4 5

vector 4 : 2 3 4 5

vector 5 : 2 3 4 5

2b). Implement R script to read person‘s age from keyboard and display whether he is eligible for voting or not.

```
{
  age <- as.integer(readline(prompt = "Enter your age :"))

  if (age >= 18) {
    print(paste("You are valid for voting :", age))
  } else{
    print(paste("You are not valid for voting :", age))
  }
}
```

output:

```
 Enter your age :48
[1] "You are valid for voting : 48"
```

2c) Implement R script to find biggest number between two numbers.

```
{
    x <- as.integer(readline(prompt = "Enter first number :"))
    y <- as.integer(readline(prompt = "Enter second number :"))

    if (x > y) {
      print(paste("Greatest is :", x))
    } else{
      print(paste("Greatest is :", y))
    }
```

output :

Enter first number :2

Enter second number :22

[1] "Greatest is : 22"

2d) Implement R script to check the given year is leap year or not

```
year = as.integer(readline(prompt="Enter a year: "))
if((year %% 4) == 0) {
    if((year %% 100) == 0) {
        if((year %% 400) == 0) {
            print(paste(year,"is a leap year"))
        } else {
            print(paste(year,"is not a leap year"))
        }
    } else {
        print(paste(year,"is a leap year"))
    }
} else {
    print(paste(year,"is not a leap year"))
}
```

output:

Enter a year: 2011

[1] "2011 is not a leap year"

Enter a year: 2004

[1] "2004 is a leap year

Experiment 3

week3

3a)Implement R Script to create a list.

Creating a List

Lists in R can be created by placing the sequence inside the list() function.

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

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

print(list_data)

Output:

[[1]] [1] "Red", [[2]] [1] "Green", [[3]] [1] 21 32 11, [[4]] [1] TRUE, [[5]] [1], 51.23, [[6]], [1] 119.1

3b) Implement R Script to access elements in the list.

 # Create a list containing a vector, a matrix and a list.

list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),

list("green",12.3))

 # Give names to the elements in the list.

names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

 # Access the first element of the list.

print(list_data[1])

 # Access the thrid element. As it is also a list, all its elements will be printed.

print(list_data[3])

 # Access the list element using the name of the element.

print(list_data$A_Matrix)

Output:

$`1st_Quarter`,[1] "Jan" "Feb" "Mar", $A_Inner_list, $A_Inner_list[[1]], [1] "green", $A_Inner_list[[2]], [1] 12.3, [,1] [,2] [,3], [1,] 3 5 -2, [2,] 9 1 8

3c) Implement R Script to merge two or more lists. Implement R Script to perform matrix operation

```
# Create two lists.

list1 <- list(1,2,3)

list2 <- list("Sun","Mon","Tue")

# Merge the two lists.

merged.list <- c(list1,list2)

# Print the merged list.

print(merged.list)
```

Output:

[[1]], [1] 1, [[2]], [1] 2, [[3]], [1] 3, [[4]][1] "Sun", [[5]] [1] "Mon", [[6]] [1] "Tue"

Experiment 4

week4

4a) Implement R script to perform following operations

various operations on vectors

```
# Use of 'c' function

# to combine the values as a vector.

# by default the type will be double

X <- c(1, 4, 5, 2, 6, 7)

print('using c function')

print(X)

# using the seq() function to generate

# a sequence of continuous values

# with different step-size and length.

# length.out defines the length of vector.

Y <- seq(1, 10, length.out = 5)

print('using seq() function')

print(Y)

# using ':' operator to create

# a vector of continuous values.

Z <- 5:10

print('using colon')

print(Y)
```

Output:

using c function 1 4 5 2 6 7

using seq function 1.00   3.25   5.50   7.75 10.00

using colon 5   6   7   8   9 10

Accessing Vector Elements

```
# Accessing elements using the position number.

X <- c(2, 5, 8, 1, 2)

print('using Subscript operator')

print(X[2])

# Accessing specific values by passing

# a vector inside another vector.

Y <- c(4, 5, 2, 1, 7)

print('using c function')

print(Y[c(4, 1)])

# Logical indexing

Z <- c(5, 2, 1, 4, 4, 3)

print('Logical indexing')

print(Z[Z>3])
```

Output:

using Subscript operator 5

using c function 1 4

Logical indexing 5 4 4

Vector Manipulation

Vectors can be modified using different indexing variations which are mentioned in the below code:

```
# Creating a vector

X <- c(2, 5, 1, 7, 8, 2)

# modify a specific element

X[3] <- 11

print('Using subscript operator')

print(X)

# Modify using different logics.
```

X[X>9] <- 0

print('Logical indexing')

print(X)

Modify by specifying the position or elements.

X <- X[c(5, 2, 1)]

print('using c function')

print(X)

Output:

Using subscript operator 2   5 11   7   8   2

Logical indexing 2 5 0 7 8 2

using c function 8 5 2

Deleting a vector

Vectors can be deleted by reassigning them as NULL. To delete a vector we use the NULL operator.

Creating a vector

X <- c(5, 2, 1, 6)

Deleting a vector

X <- NULL

print('Deleted vector')

print(X)

Deleted vector NULL

Sorting of Vectors

For sorting we use the sort() function which sorts the vector in ascending order by default.

Creating a Vector

X <- c(5, 2, 5, 1, 51, 2)

Sort in ascending order

A <- sort(X)

print('sorting done in ascending order')

```
print(A)

# sort in descending order.

B <- sort(X, decreasing = TRUE)

print('sorting done in descending order')

print(B)
```

Output:

sorting done in ascending order 1   2   2   5   5 51

sorting done in descending order 51   5   5   2   2   1

4b) Finding the sum and average of given numbers using arrays.

To find the sum of all array elements in R, we can use Reduce function with plus sign. For Example, if we have an array called ARRAY and we want to find the sum of all values in this array then we can use the command Reduce("+",ARRAY).

Example 1

To find the sum of all array elements in R use the snippet given below −

```
Array1<-array(1:100,c(5,4,5))
Array1
, , 1
```

If you execute the above given snippet, it generates the following Output −

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

To find the sum of all elements in Array1 on the above created data frame, add the following code to the above snippet −

```
Array1<-array(1:100,c(5,4,5))

Reduce("+",Array1)
```

Output

If you execute all the above given snippets as a single program, it generates the following Output −

[1] 5050

Average in R Programming

colMeans() function in R Language is used to compute the mean of each column of a matrix or array

Syntax: colMeans(x, dims = 1)

Parameters:

x: array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame

dims: integer value, which dimensions are regarded as 'columns' to sum over. It is over dimensions 1:dims.

```
# R program to illustrate

# colMeans function

# Initializing a 3D array

x <- array(1:12, c(2, 3, 3))

# Getting the array representation

X

# Calling the colMeans() function

# for dims = 1, x[, 1, 1], x[, 2, 1], x[, 3, 1],

# x[, 1, 2] ... are columns

colMeans(x, dims = 1)

# for dims = 2, x[,,1], x[,,2], x[,,3]

# are columns

colMeans(x, dims = 2)
```

Output:

```
,, 1

      [, 1] [, 2] [, 3]

[1, ]    1     3     5

[2, ]    2     4     6,, 2

      [, 1] [, 2] [, 3]

[1, ]    7     9    11
```

[2,]   8   10   12,, 3

[, 1] [, 2] [, 3]

[1,]   1   3   5

[2,]   2   4   6

[, 1] [, 2] [, 3]

[1,] 1.5 7.5 1.5

[2,] 3.5 9.5 3.5

[3,] 5.5 11.5 5.5

[1] 3.5 9.5 3.5

4c). To display elements of list in reverse order

```
x <- list(5, 25, 125)
result = rev(x)
print(result)
```

output:
[[1]]
[1]125
[[2]]
[1]25
[[3]]
[1]5

4d**) Finding the minimum and maximum elements in the array**

```
x = c(10, 20, 30, 25, 9, 26)
print("Original Vectors:")
print(x)
print("Maximum value of the above Vector:")
print(max(x))
print("Minimum value of the above Vector:")
print(min(x))
```

Output:

[1] "Original Vectors:"
[1] 10 20 30 25   9 26
[1] "Maximum value of the above Vector:"
[1] 30
[1] "Minimum value of the above Vector:"
[1] 9

EXPERIMENT 5

week5

5a) Implement R Script to perform various operations on matrices

```
# R program for matrix addition

# using '+' operator

# Creating 1st Matrix

B = matrix(c(1, 2 + 3i, 5.4, 3, 4, 5), nrow = 2, ncol = 3)

# Creating 2nd Matrix

C = matrix(c(2, 0i, 0.1, 3, 4, 5), nrow = 2, ncol = 3)

# Printing the resultant matrix

print(B + C)
```

Output:

```
      [,1]    [,2]   [,3]
[1,] 3+0i 5.5+0i    8+0i
[2,] 2+3i 6.0+0i 10+0i
```

```
# R program for matrix addition

# using '-' operator

# Creating 1st Matrix

B = matrix(c(1, 2 + 3i, 5.4, 3, 4, 5), nrow = 2, ncol = 3)

# Creating 2nd Matrix

C = matrix(c(2, 0i, 0.1, 3, 4, 5), nrow = 2, ncol = 3)

# Printing the resultant matrix

print(B - C)
```

Output:

```
        [,1]    [,2] [,3]
[1,] -1+0i 5.3+0i 0+0i
[2,]   2+3i 0.0+0i 0+0i
```

R program for matrix multiplication

using '*' operator

Creating 1st Matrix

B = matrix(c(1, 2 + 3i, 5.4), nrow = 1, ncol = 3)

Creating 2nd Matrix

C = matrix(c(2, 1i, 0.1), nrow = 1, ncol = 3)

Printing the resultant matrix

print (B * C)

Output:

```
     [,1]    [,2]       [,3]
[1,] 2+0i -3+2i 0.54+0i
```

R program for matrix division

using '/' operator

Creating 1st Matrix

B = matrix(c(4, 6i, -1), nrow = 1, ncol = 3)

Creating 2nd Matrix

C = matrix(c(2, 2i, 0), nrow = 1, ncol = 3)

Printing the resultant matrix

print (B / C)

Output:

```
     [,1] [,2]        [,3]
[1,] 2+0i 3+0i -Inf+NaNi
```

5b) Implement R Script to extract the data from dataframes.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
```

```
    salary = c(623.3,515.2,611.0,729.0,843.25),

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

    stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

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

```
  emp_id    emp_name     salary     start_date
1     1     Rick        623.30     2012-01-01
2     2     Dan         515.20     2013-09-23
3     3     Michelle    611.00     2014-11-15
4     4     Ryan        729.00     2014-05-11
5     5     Gary        843.25     2015-03-27
```

Get the Structure of the Data Frame

The structure of the data frame can be seen by using **str()** function.

```
# Create the data frame.
emp.data <- data.frame(
    emp_id = c (1:5),
    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),

    salary = c(623.3,515.2,611.0,729.0,843.25),

    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
        "2015-03-27")),
    stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(emp.data)
```

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

```
'data.frame':     5 obs. of   4 variables:
 $ emp_id      : int   1 2 3 4 5
 $ emp_name   : chr   "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary      : num   623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying **summary()** function.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

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

```
     emp_id      emp_name              salary         start_date
 Min.   :1    Length:5           Min.   :515.2   Min.   :2012-01-01
 1st Qu.:2    Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
 Median :3    Mode  :character   Median :623.3   Median :2014-05-11
 Mean   :3                       Mean   :664.4   Mean   :2014-01-14
 3rd Qu.:4                       3rd Qu.:729.0   3rd Qu.:2014-11-15
 Max.   :5                       Max.   :843.2   Max.   :2015-03-27
```

Extract Data from Data Frame

Extract specific column from a data frame using column name.

```
# Create the data frame.
emp.data <- data.frame(
    emp_id = c (1:5),
    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
    salary = c(623.3,515.2,611.0,729.0,843.25),
    start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
        "2015-03-27")),
    stringsAsFactors = FALSE
)
# Extract Specific columns.
result <- data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

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

```
  emp.data.emp_name emp.data.salary
1              Rick          623.30
2               Dan          515.20
3          Michelle          611.00
4              Ryan          729.00
5              Gary          843.25
```

Extract the first two rows and then all columns

```
# Create the data frame.
emp.data <- data.frame(
    emp_id = c (1:5),
    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
    salary = c(623.3,515.2,611.0,729.0,843.25),
    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
        "2015-03-27")),
    stringsAsFactors = FALSE
)
# Extract first two rows.
```

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

print(result)
```

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

```
  emp_id    emp_name   salary    start_date
      1      Rick        623.3      2012-01-01
      2      Dan          515.2      2013-09-23
```

5c) Write R script to display file contents.

R program to read one line at a time

Import the readr library

library(readr)

read_lines() to read one line at a time

myData = read_lines("geeksforgeeks.txt", n_max = 1)

print(myData)

read_lines() to read two line at a time

myData = read_lines("geeksforgeeks.txt", n_max = 2)

print(myData)

Output:

[1] "A computer science portal for geeks."

[1] "A computer science portal for geeks."

[2] "Geeksforgeeks is founded by Sandeep Jain Sir."

R program to read the whole file

Import the readr library

library(readr)

read_file() to read the whole file

myData = read_file("geeksforgeeks.txt")

print(myData)

Output:

[1] “A computer science portal for geeks.\r\nGeeksforgeeks is founded by Sandeep Jain Sir.\r\nI am an intern at this amazing platform.”

5d) Write R script to copy file contents from one file to another

Step 3: Copy a file from one folder to another.

To copy a file from one folder to another, use the file.copy() method. The complete code is below.

```
dir.create("newdir")

newDirPath <- "newdir"

files <- c("a.txt")

file.create(files)

newFilePath <- "a.txt"

file.copy(newFilePath, newDirPath)
```

Output

[1] TRUE
[1] TRUE
The first TRUE is for successfully creating a file and the second TRUE is for successfully copying the file.If it returns FALSE, that means there is s some problem while copying the files.R base functions for importing data

Experiment:- 6

week 6

6a) Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets.

```
>mtcars
mpg                     cyl     dis   php    drat   wtq    sec    vs   am          gear
Mazda RX4              21.0      6 160.0 110 3.90 2.620 16.46   0   1              4
4
Mazda RX4 Wag          21.0      6 160.0 110 3.90 2.875 17.02   0   1              4
4
Datsun 710             22.8      4 108.0   93 3.85 2.320 18.61   1   1              4
1
Hornet 4 Drive         21.4      6 258.0 110 3.08 3.215 19.44   1   0              3
1
Hornet Sportabout      18.7      8 360.0 175 3.15 3.440 17.02   0   0              3
2
Valiant                18.1      6 225.0 105 2.76 3.460 20.22   1   0              3
1
Duster 360             14.3      8 360.0 245 3.21 3.570 15.84   0   0              3
4
Merc 240D              24.4      4 146.7   62 3.69 3.190 20.00   1   0              4
2
Merc 230               22.8      4 140.8   95 3.92 3.150 22.90   1   0              4
2
Merc 280               19.2      6 167.6 123 3.92 3.440 18.30   1   0              4
4
Merc 280C              17.8      6 167.6 123 3.92 3.440 18.90   1   0              4
4
Merc 450SE             16.4      8 275.8 180 3.07 4.070 17.40   0   0              3
3
Merc 450SL             17.3      8 275.8 180 3.07 3.730 17.60   0   0              3
3
Merc 450SLC            15.2      8 275.8 180 3.07 3.780 18.00   0   0              3
3
Cadillac Fleetwood     10.4      8 472.0 205 2.93 5.250 17.98   0   0              3
4
Lincoln Continental    10.4      8 460.0 215 3.00 5.424 17.82   0   0              3
4
Chrysler Imperial      14.7      8 440.0 230 3.23 5.345 17.42   0   0              3
4
Fiat 128               32.4      4  78.7   66 4.08 2.200 19.47   1   1              4
1
Honda Civic            30.4      4  75.7   52 4.93 1.615 18.52   1   1              4
2
Toyota Corolla         33.9      4  71.1   65 4.22 1.835 19.90   1   1              4
1
Toyota Corona          21.5      4 120.1   97 3.70 2.465 20.01   1   0              3
```

1
Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3

2
AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30

```
13.3    8  350.0  245  3.73  3.840  15.41   0   0     3
19.2    8  400.0  175  3.08  3.845  17.05   0   0     3
27.3    4   79.0   66  4.08  1.935  18.90   1   1     4
26.0    4  120.3   91  4.43  2.140  16.70   0   1     5
30.4    4   95.1  113  3.77  1.513  16.90   1   1     5
15.8    8  351.0  264  4.22  3.170  14.50   0   1     5
19.7    6  145.0  175  3.62  2.770  15.50   0   1     5
15.0    8  301.0  335  3.54  3.570  14.60   0   1     5
21.4    4  121.0  109  4.11  2.780  18.60   1   1     4
```

Camaro Z284

Pontiac Firebird2

Fiat X1-91

Porsche 914-2

2

Lotus Europa2

Ford Pantera L4

Ferrari Dino6

Maserati Bora8

Volvo 142E2

>summary(mtcars)

mpgcyldisphp drat
Min.:10.40 Min. :4.000 Min. : 71.1 Min.: 52.0 Min.:2.760 1st
Qu.:15.43 1st Qu.:4.000 1st Qu.:120.8 1st Qu.: 96.5 1st
Qu.:3.080
Median :19.20 Median :6.000 Median :196.3 Median :123.0 Median
:3.695
Mean :20.09 Mean :6.188 Mean :230.7 Mean :146.7 Mean
:3.597
3rd Qu.:22.80 3rd Qu.:8.000 3rd Qu.:326.0 3rd Qu.:180.0 3rd

Qu.:3.920
Max. :33.90 Max. :8.000 Max. :472.0 Max. :335.0 Max. :4.930
wtqsec vs am gear
Min.:1.513 Min. :14.50 Min. :0.0000 Min. :0.0000 Min. :3.000
1st Qu.:2.581 1st Qu.:16.89 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:3.000
Median :3.325 Median :17.71 Median :0.0000 Median :0.0000 Median :4.000
Mean :3.217 Mean :17.85 Mean :0.4375 Mean :0.4062 Mean :3.688
3rd Qu.:3.610 3rd Qu.:18.90 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:4.000
Max. :5.424 Max. :22.90 Max. :1.0000 Max. :1.0000 Max. :5.000
carb
Min.:1.000
1st Qu.:2.000
Median :2.000
Mean :2.812 3rd Qu.:4.000 Max. :8.000

>str(mtcars)

‘data.frame': 32 obs. of 11 variables:
$ mpg :num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
$ cyl :num 6 6 4 6 8 6 8 4 4 6 ...
$ disp: num 160 160 108 258 360 ...
$ hp :num 110 110 93 110 175 105 245 62 95 123 ...
$ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
$ wt :num 2.62 2.88 2.32 3.21 3.44 ...
$ qsec: num 16.5 17 18.6 19.4 17 ...
$ vs :num 0 0 1 1 0 1 0 1 1 1 ...
$ am :num 1 1 1 0 0 0 0 0 0 0 ...
$ gear: num 4 4 4 3 3 3 3 4 4 4 ...
$ carb: num 4 4 1 1 2 1 4 2 2 4 ...

>quantile(mtcars$mpg)

0% 25% 50% 75% 100%
10.400 15.425 19.200 22.800 33.900

>cars speed

dist

1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10
7	10	18
8	10	26
9	10	34
10	11	17
11	11	28
12	12	14
13	12	20
14	12	24
15	12	28
16	13	26
17	13	34
18	13	34
19	13	46
20	14	26
21	14	36
22	14	60
23	14	80
24	15	20
25	15	26
26	15	54
27	16	32
28	16	40
29	17	32
30	17	40
31	17	50
32	18	42
33	18	56
34	18	76
35	18	84
36	19	36
37	19	46
38	19	68
39	20	32
40	20	48
41	20	52
42	20	56
43	20	64
44	22	66
45	23	54
46	24	70
47	24	92
48	24	93
49	24	120
50	25	85

```
>summary(cars)

speeddist
Min.: 4.0          Min.      :   2.00
  1st Qu.:12.0          1st Qu.: 26.00
Median :15.0          Median : 36.00
  Mean    :15.4    Mean        :   42.98  3rd
  Qu.:19.0   3rd   Qu.:   56.00  Max.    :25.0
  Max.   :120.00

>class(cars)
[1] "data.frame"

>dim(cars)

[1] 50   2

>str(cars)

'data.frame': 50 obs. of   2 variables:
  $ speed: num   4 4 7 7 8 9 10 10 10 11 ...
  $ dist :num   2 10 4 22 16 10 18 26 34 17 ..

>quantile(cars$speed)

     0    25%    50%   75%    100%
    %
     4      12    15          19      25
```

6b). Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

```
>aggregate(. ~ Species, data = iris, mean)
```

Output:

```
Species Sepal.LengthSepal.WidthPetal.LengthPetal.Width
 1           Setosa                 5.006            3.428              1.462             0.246
 2       Versicolor                 5.936            2.770              4.260             1.326
 3         Virginica                6.588            2.974              5.552             2.026
```

```
>subset(iris,iris$Sepal.Length==5.0)
```

Output:

```
Sepal.LengthSepal.WidthPetal.LengthPetal.WidthSpecies
5                    5          3.6            1.4           0.2       Setosa
8                    5          3.4            1.5           0.2       Setosa
26                   5          3.0            1.6           0.2       Setosa
27                   5          3.4            1.6           0.4       Setosa
36                   5          3.2            1.2           0.2       Setosa
```

41	5	3.5	1.3	0.3	Setosa
44	5	3.5	1.6	0.6	Setosa
50	5	3.3	1.4	0.2	Setosa
61	5	2.0	3.5	1.0	Versicolor
94	5	2.3	3.3	1.0	versicolor

Ways of Subsetting Data in R

R is capable of pulling the desired portion of data. Subsetting a data frame in R is the most essential part of data manipulation. We will go through subsetting data in detail.

In this part, we will use iris data set available in R. Firstly, let's know the iris data we will work on.

```
class(iris)
## [1] "data.frame"
dim(iris)
## [1] 150     5
```

1. Subset Using Brackets by Selecting Rows and Columns

In this part, we use brackets by selecting rows and colums. Firstly, we pull the first three rows of data. Then, we select the first three rows and the columns from third to fifth.

```
iris[c(1:3),]
##   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

iris[c(1:3),c(3:5)]


##   Petal.Length Petal.Width Species
## 1          1.4         0.2  setosa
## 2          1.4         0.2  setosa
## 3          1.3         0.2  setosa
```

2. Subset Using Brackets by Excluding Rows and Columns

Also, we can fin iris[-c(4:150),]

```
##   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
```

```
iris[-c(4:150),-c(1:2)]
##   Petal.Length Petal.Width Species
## 1          1.4         0.2  setosa
## 2          1.4         0.2  setosa
## 3          1.3         0.2  setosad same subset by excluding the rows and columns.
```

3. Subset Using Brackets with which() Function

We can select any subset of data in R based on condition with which() function. For example, let’s select setosa species with their sepal length larger than 5.6. Also, we can obtain the columns from third to fifth of the same subset.

```
iris[which(iris$Species=="setosa"&iris$Sepal.Length>5.6),]
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 19          5.7         3.8          1.7         0.3  setosa

iris[which(iris$Species=="setosa"&iris$Sepal.Length>5.6), 3:5]
##    Petal.Length Petal.Width Species
## 15          1.2         0.2  setosa
## 16          1.5         0.4  setosa
## 19          1.7         0.3  setosa
```

4. Subset Data with subset() Function

We can select same subsets with subset() fuction.

```
subset(iris, Species=="setosa"&Sepal.Length>5.6)
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 19          5.7         3.8          1.7         0.3  setosa

subset(iris, Species=="setosa"&Sepal.Length>5.6, 3:5)
##    Petal.Length Petal.Width Species
## 15          1.2         0.2  setosa

## 16          1.5         0.4  setosa
## 19          1.7         0.3  setosa
```

5. Subset Data in Combination of select() and filter() Functions

We can obtain same subsets using filter() and select() functions available in dplyr package (Wickham et al., 2020).

```
library(dplyr)
filter(iris, Species=="setosa"&Sepal.Length>5.6)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.8         4.0          1.2         0.2  setosa
## 2          5.7         4.4          1.5         0.4  setosa
## 3          5.7         3.8          1.7         0.3  setosa
```

```
select(filter(iris, Species=="setosa"&Sepal.Length>5.6), 3:5)
##     Petal.Length Petal.Width Species
## 1            1.2         0.2  setosa
## 2            1.5         0.4  setosa
## 3            1.7         0.3  setosa
```

6. Subset a Random Sample with sample() Function

Lastly, we will learn how to sample a subset randomly from a data frame with sample() function.

```
set.seed(123) # For reproducibility of same result
iris[sample(1:nrow(iris), 3, replace = FALSE),]
##       Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 44           5.0         3.5          1.6         0.6     setosa
## 118          7.7         3.8          6.7         2.2  virginica
## 61           5.0         2.0          3.5         1.0 versicolor

set.seed(123) # For reproducibility of same result
iris[sample(1:nrow(iris), 3, replace = FALSE), 3:5]
##       Petal.Length Petal.Width    Species
## 44           1.6         0.6     setosa
## 118          6.7         2.2  virginica
## 61           3.5         1.0 versicolor
```

Experiment 7

week7

7a) Reading different types of data sets (.txt, .csv) from Web or disk and writing in file inspecific disk location.

Working with CSV files in R Programming

input.csv

id,name,salary,start_date,dept

1,Rick,623.3,2012-01-01,IT

2,Dan,515.2,2013-09-23,Operations

3,Michelle,611,2014-11-15,IT

4,Ryan,729,2014-05-11,HR

5,Gary,843.25,2015-03-27,Finance

6,Nina,578,2013-05-21,IT

7,Simon,632.8,2013-07-30,Operations

8,Guru,722.5,2014-06-17,Finance

Reading a CSV File: Following is a simple example of read.csv() function to read a CSV file available in your current working directory –

```
data <- read.csv("input.csv")
print(data)
```

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

id, name, salary, start_date, dept

1 1 Rick 623.30 2012-01-01 IT

2 2 Dan 515.20 2013-09-23 Operations

3 3 Michelle 611.00 2014-11-15 IT

4 4 Ryan 729.00 2014-05-11 HR

5 NA Gary 843.25 2015-03-27 Finance

6 6 Nina 578.00 2013-05-21 IT

7 7 Simon 632.80 2013-07-30 Operations

8 8 Guru 722.50 2014-06-17 Finance

Analyzing the CSV File

By default the read.csv() function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```
data <- read.csv("input.csv")
```

print(is.data.frame(data))

print(ncol(data))

print(nrow(data))

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

[1] TRUE

[1] 5

[1] 8

Writing into a CSV File

R can create csv file form existing data frame. The write.csv() function is used to create the csv file. This file gets created in the working directory.

Create a data frame.

data <- read.csv("input.csv")

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

Write filtered data into a new file.

write.csv(retval,"output.csv")

newdata <- read.csv("output.csv")

print(newdata)

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

X id name salary start_date dept

1 3 3 Michelle 611.00 2014-11-15 IT

2 4 4 Ryan 729.00 2014-05-11 HR

3 5 NA Gary 843.25 2015-03-27 Finance

4 8 8 Guru 722.50 2014-06-17 Finance

7b) Reading Excel data sheet in R.

Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with required package name to install the additional packages.

install.packages("xlsx")

Verify and Load the "xlsx" Package

Use the following command to verify and load the "xlsx**" package.**

Verify the package is installed.

any(grepl("xlsx",installed.packages()))

Load the library into R workspace.

library("xlsx")

When the script is run we get the following output.

[1] TRUE
Loading required package: rJava
Loading required package: methods
Loading required package: xlsxjars
Input as xlsx File
Open Microsoft excel. Copy and paste the following data in the work sheet named as sheet1.

id name salary start_date dept
1 Rick 623.3 1/1/2012 IT
2 Dan 515.2 9/23/2013 Operations
3 Michelle 611 11/15/2014 IT
4 Ryan 729 5/11/2014 HR
5 Gary 43.25 3/27/2015 Finance
6 Nina 578 5/21/2013 IT
7 Simon 632.8 7/30/2013 Operations
8 Guru 722.5 6/17/2014 Finance
Also copy and paste the following data to another worksheet and rename this worksheet to "city".
name city
Rick Seattle
Dan Tampa
Michelle Chicago
Ryan Seattle
Gary Houston
Nina Boston
Simon Mumbai
Guru Dallas
Save the Excel file as "input.xlsx". You should save it in the current working directory of the R workspace.
Reading the Excel File
The input.xlsx is read by using the read.xlsx() function as shown below. The result is stored as a data frame in the R environment.

```
# Read the first worksheet in the file input.xlsx.
data <- read.xlsx("input.xlsx", sheetIndex = 1)
print(data)
```

When we execute the above code, it produces the following result −
id, name, salary, start_date, dept
1 1 Rick 623.30 2012-01-01 IT
2 2 Dan 515.20 2013-09-23 Operations
3 3 Michelle 611.00 2014-11-15 IT
4 4 Ryan 729.00 2014-05-11 HR
5 NA Gary 843.25 2015-03-27 Finance
6 6 Nina 578.00 2013-05-21 IT
7 7 Simon 632.80 2013-07-30 Operations
8 8 Guru 722.50 2014-06-17 Finance

7c) Reading XML dataset in R

You can read a xml file in R using the "XML" package. This package can be installed using following command.

```
install.packages("XML")
```

Input Data

Create a XMl file by copying the below data into a text editor like notepad. Save the file with a **.xml** extension and choosing the file type as **all files(*.*)**.


```
<RECORDS>
   <EMPLOYEE>
      <ID>1</ID>
      <NAME>Rick</NAME>
      <SALARY>623.3</SALARY>
      <STARTDATE>1/1/2012</STARTDATE>
      <DEPT>IT</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>2</ID>
      <NAME>Dan</NAME>
      <SALARY>515.2</SALARY>
      <STARTDATE>9/23/2013</STARTDATE>
      <DEPT>Operations</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>3</ID>
      <NAME>Michelle</NAME>
      <SALARY>611</SALARY>
      <STARTDATE>11/15/2014</STARTDATE>
      <DEPT>IT</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>4</ID>
      <NAME>Ryan</NAME>
      <SALARY>729</SALARY>
      <STARTDATE>5/11/2014</STARTDATE>
      <DEPT>HR</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>5</ID>
      <NAME>Gary</NAME>
      <SALARY>843.25</SALARY>
      <STARTDATE>3/27/2015</STARTDATE>
      <DEPT>Finance</DEPT>
   </EMPLOYEE>
```

```
<EMPLOYEE>
    <ID>6</ID>
    <NAME>Nina</NAME>
    <SALARY>578</SALARY>
    <STARTDATE>5/21/2013</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>7</ID>
    <NAME>Simon</NAME>
    <SALARY>632.8</SALARY>
    <STARTDATE>7/30/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>8</ID>
    <NAME>Guru</NAME>
    <SALARY>722.5</SALARY>
    <STARTDATE>6/17/2014</STARTDATE>
    <DEPT>Finance</DEPT>
  </EMPLOYEE>

</RECORDS>
```

The xml file is read by R using the function **xmlParse()**. It is stored as a list in R.

```
# Load the package required to read XML files.
library("XML")

# Also load the other required package.
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Print the result.
print(result)
```

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

```
1
Rick
623.3
1/1/2012
```

IT
2
Dan
515.2
9/23/2013
Operations
3
Michelle
611
11/15/2014
IT
4
Ryan
729
5/11/2014
HR
5
Gary
843.25
3/27/2015
Finance
6
Nina
578
5/21/2013
IT
7
Simon
632.8
7/30/2013
Operations
8
Guru
722.5
6/17/2014
Finance

Experiment 8

week8

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

Bar Chart

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

Syntax

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

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

Following is the description of the parameters used −

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

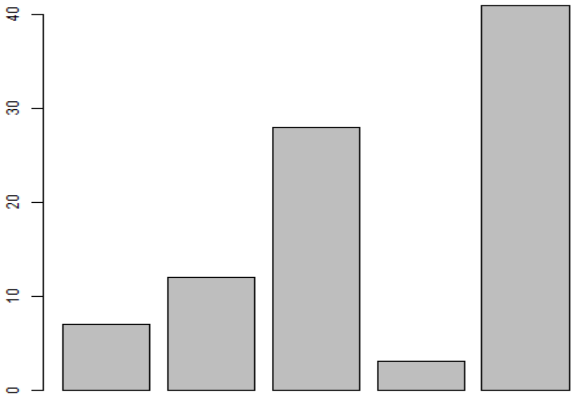
Example

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

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

```
# Create the data for the chart
H <- c(7,12,28,3,41)

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

# Plot the bar chart
barplot(H)

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

When we execute above code, it produces following result −



Pie chart

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

Syntax

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

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

Following is the description of the parameters used −

x is a vector containing the numeric values used in the pie chart.

labels is used to give description to the slices.

radius indicates the radius of the circle of the pie chart.(value between −1 and +1).

main indicates the title of the chart.

col indicates the color palette.

clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise.

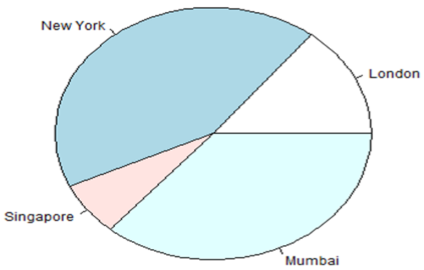
Example

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

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")
```

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

# Plot the chart.
pie(x,labels)

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

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



Scatterplots

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

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

Syntax

The basic syntax for creating scatterplot in R is −

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

Following is the description of the parameters used −

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

Example: We use the data set **"mtcars"** available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

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

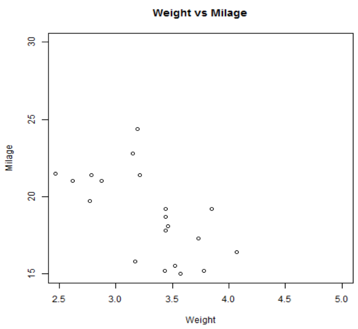
```
                     wt      mpg
Mazda RX4           2.620    21.0
Mazda RX4 Wag       2.875    21.0
Datsun 710          2.320    22.8
Hornet 4 Drive      3.215    21.4
Hornet Sportabout   3.440    18.7
Valiant             3.460    18.1
```

Creating the Scatterplot

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

```
# Get the input values.
input <- mtcars[,c('wt','mpg')]

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

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

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



R - Histograms

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

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

Syntax

The basic syntax for creating a histogram using R is −

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

Following is the description of the parameters used −

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

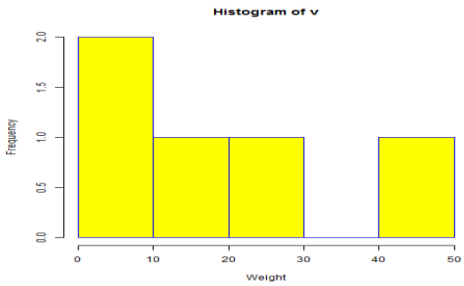
Example

A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.

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

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

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

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

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



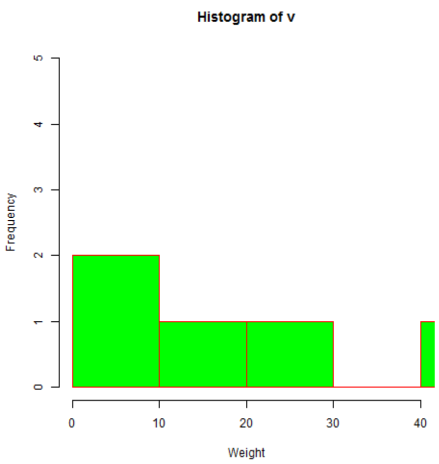
Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

The width of each of the bar can be decided by using breaks.

```
# 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_lim_breaks.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
    breaks = 5)

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

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



8b) Implement R Script to perform mean, median, mode, range, summary, variance,

standard deviation operations.

Mean

CardioGoodFitness.csv

```
# R program to illustrate

# Descriptive Analysis

# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                  stringsAsFactors=F)

# Compute the mean value

mean = mean(myData$Age)

print(mean)
```

Output:

```
[1] 28.78889
```

```
# R program to illustrate

# Descriptive Analysis

# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                  stringsAsFactors=F)

# Compute the median value

median = median(myData$Age)

print(median)
```

Output:

```
[1] 26
```

```
# R program to illustrate

# Descriptive Analysis

# Import the library
```

```
library(modest)
# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors=F)
# Compute the mode value
mode = mfv(myData$Age)
print(mode)
```

Output:

[1] 25

R program to get average of a list

```
# Taking a list of elements
list = c(2, 4, 4, 4, 5, 5, 7, 9)
# Calculating average using mean()
print(mean(list))
```

Output:

[1] 5

```
# R program to get variance of a list
# Taking a list of elements
list = c(2, 4, 4, 4, 5, 5, 7, 9)
# Calculating variance using var()
print(var(list))
```

Output:

[1] 4.571429

```
# R program to get
# standard deviation of a list
# Taking a list of elements
list = c(2, 4, 4, 4, 5, 5, 7, 9)
```

Calculating standard

deviation using sd()

print(sd(list))

Output:

[1] 2.13809

create vector

data = c(12, 45, NA, NA, 67, 23, 45, 78, NA, 89)

display

print(data)

find range

print(max(data, na.rm=TRUE)-min(data, na.rm=TRUE))

Output:

[1] 12 45 NA NA 67 23 45 78 NA 89

[1] 77

create a vector wit 10 elements

data = c(1: 5, 56, 43, 56, 78, 51)

display

print(data)

get summary

print(summary(data))

```
[1]  1  2  3  4  5 56 43 56 78 51
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25   24.00   29.90   54.75   78.00
```

EXPERIMENT 9

week9

9a) Implement R Script to perform Normal, Binomial distributions.

Functions to Generate Normal Distribution in R
Below are the different functions to generate normal distribution in R programming:

1. dnorm()
Syntax: dnorm(x, mean, sd)

For example:

Create a sequence of numbers between -10 and 10 incrementing by 0.1.

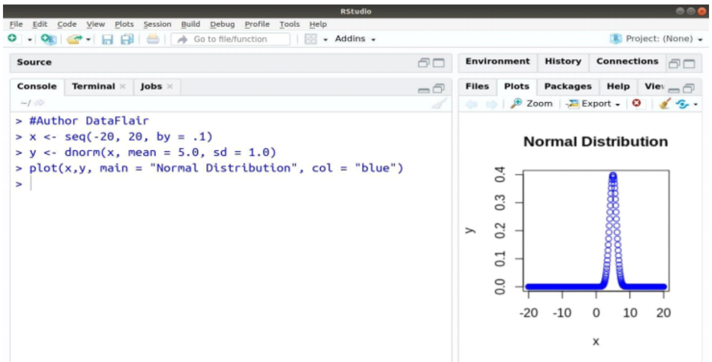
> #Author DataFlair

> x <- seq(-20, 20, by = .1)

> y <- dnorm(x, mean = 5.0, sd = 1.0)

> plot(x,y, main = "Normal Distribution", col = "blue")

Output:



2. pnorm()

Syntax: pnorm(x,mean,sd)

For example:

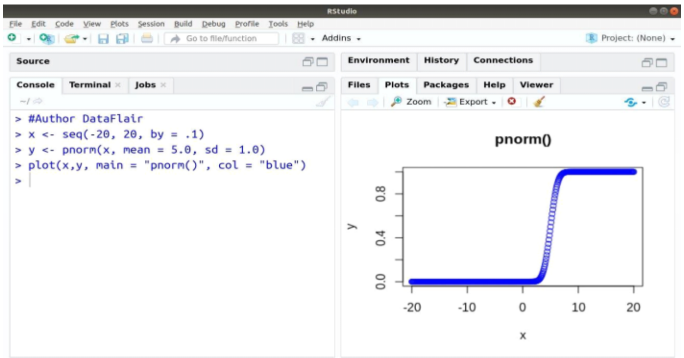
> #Author DataFlair

> x <- seq(-20, 20, by = .1)

> y <- pnorm(x, mean = 5.0, sd = 1.0)

> plot(x,y, main = "pnorm()", col = "blue")

Output:



You must definitely check the [Numeric and Character Functions in R]

3. qnorm()

Syntax: qnorm(x,mean,sd)

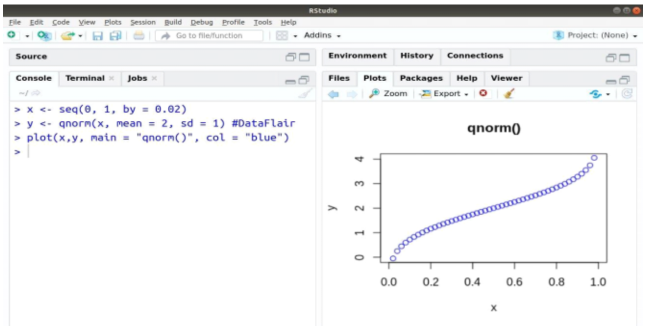
For example:

> x <- seq(0, 1, by = 0.02)

> y <- qnorm(x, mean = 2, sd = 1) #DataFlair

> plot(x,y, main = "qnorm()", col = "blue

Output:



4. rnorm()

Syntax: rnorm(n, mean, sd)

For example:

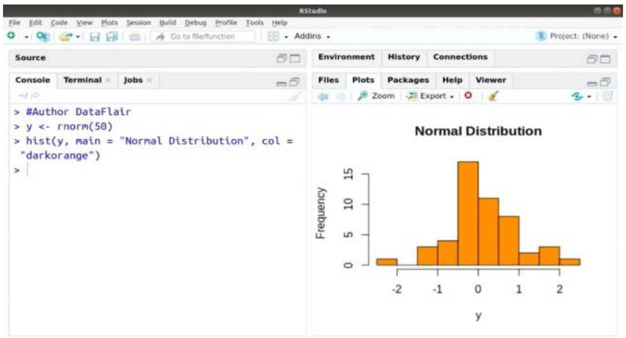
Create a sample of 50 numbers which are normally distributed.

#Author Dataflair

y <- rnorm(50)

hist(y, main = "Normal Distribution", col = "darkorange")

Output:



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.

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

dbinom(x, size, prob)

pbinom(x, size, prob)

qbinom(p, size, prob)

rbinom(n, size, prob)

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

dbinom()

This function gives the probability density distribution at each point.

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

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

Create the binomial distribution.

y <- dbinom(x,50,0.5)

Give the chart file a name.

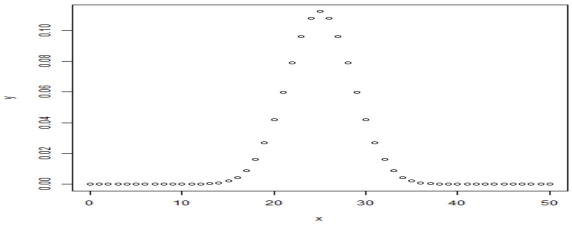
png(file = "dbinom.png")

Plot the graph for this sample.

plot(x,y)

Save the file.

dev.off()



pbinom()

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

Probability of getting 26 or less heads from a 51 tosses of a coin.

x <- pbinom(26,51,0.5)

print(x)

output

[1] 0.610116

qbinom()

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

How many heads will have a probability of 0.25 will come out when a coin

is tossed 51 times.

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

print(x)

output

[1] 23

rbinom()

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

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

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

print(x)
```

output

[1] 58 61 59 66 55 60 61 67

9b) Implement R Script to perform correlation, Linear and multiple regression

Correlation in R Programming Language

cor() function in R programming measures the correlation coefficient value. Correlation is a relationship term in statistics that uses the covariance method to measure how strong the vectors are related.

Correlation in R

Syntax: cor(x, y, method)

where,

x and y represents the data vectors

method defines the type of method to be used to compute covariance.

```
# Data vectors

x <- c(1, 3, 5, 10)

y <- c(2, 4, 6, 20)

# Print correlation using different methods

print(cor(x, y))

print(cor(x, y, method = "pearson"))

print(cor(x, y, method = "kendall"))

print(cor(x, y, method = "spearman"))
```

Output:

[1] 0.9724702

[1] 0.9724702

[1] 1

[1] 1

LINEAR REGRESSION

It is a commonly used type of predictive analysis. It is a statistical approach for modeling the relationship between a dependent variable and a given set of independent variables. y = ax + b

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

Input Data
Values of height

151, 174, 138, 186, 128, 136, 179, 163, 152, 131

Values of weight.

63, 81, 56, 91, 47, 57, 76, 72, 62, 48

lm() Function
This function creates the relationship model between the predictor and the response variable.

Syntax
lm(formula,data)

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

Create Relationship Model & get the Coefficients
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

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

Apply the lm() function.

relation <- lm(y~x)

print(relation)

Call:

lm(formula = y ~ x)

Coefficients:

(Intercept) x

-38.4551 0.6746

Get the Summary of the Relationship
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

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

Apply the lm() function.

relation <- lm(y~x)

print(summary(relation))

Call:

lm(formula = y ~ x)

Residuals:

Min 1Q Median 3Q Max

-6.3002 -1.6629 0.0412 1.8944 3.9775

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) -38.45509 8.04901 -4.778 0.00139 **

x 0.67461 0.05191 12.997 1.16e-06 ***

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom

Multiple R-squared: 0.9548, Adjusted R-squared: 0.9491

F-statistic: 168.9 on 1 and 8 DF, p-value: 1.164e-06

predict() Function

Syntax
predict(object, newdata)

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

Predict the weight of new persons

```
# The predictor vector.

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

# The resposne vector.

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

# Apply the lm() function.

relation <- lm(y~x)

# Find weight of a person with height 170.

a <- data.frame(x = 170)

result <-   predict(relation,a)

print(result)
```

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

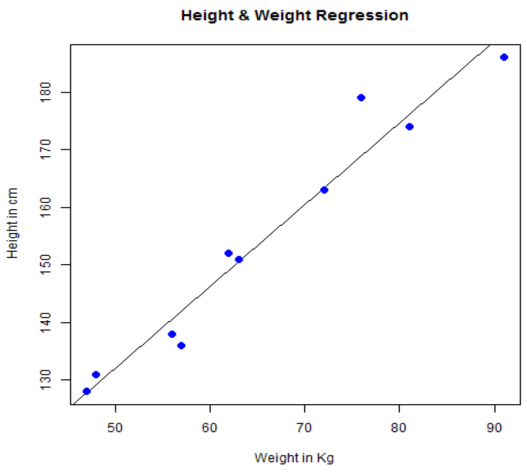
1

76.22869

Visualize the Regression Graphically

```
# Create the predictor and response variable.

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

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

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

# Plot the chart.

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

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

# Save the file.

dev.off()
```

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



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.

y = a + b1x1 + b2x2 +...bnxn

- y is the response variable.
- a, b1, b2...bn are the coefficients.
- x1, x2, ...xn are the predictor variables.

We create the regression model using the lm() function in R. The model determines the value of the coefficients using the input data. Next we can predict the value of the response variable for a given set of predictor variables using these coefficients.

lm() Function

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

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

- formula is a symbol presenting the relation between the response variable and predictor variables.
- data is the vector on which the formula will be applied.

Example

Input Data
Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.

The goal of the model is to establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

input <- mtcars[,c("mpg","disp","hp","wt")]

print(head(input))

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

mpg disp hp wt

Mazda RX4 21.0 160 110 2.620

Mazda RX4 Wag 21.0 160 110 2.875

Datsun 710 22.8 108 93 2.320

Hornet 4 Drive 21.4 258 110 3.215

Hornet Sportabout 18.7 360 175 3.440

Valiant 18.1 225 105 3.460

Create Relationship Model & get the Coefficients
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)

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

Call:

lm(formula = mpg ~ disp + hp + wt, data = input)

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

Create Equation for Regression Model
Based on the above intercept and coefficient values, we create the mathematical equation.

Y = a+Xdisp.x1+Xhp.x2+Xwt.x3

EXPERIMENT 10

week10

10a) Introduction to Non-Tabular Data Types: Time series, spatial data, Network data

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame.

The time series object is created by using the ts() function.

Syntax

timeseries.object.name <- ts(data, start, end, frequency)

- 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.

Except the parameter "data" all other parameters are optional.

Example

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.

Get the data points in form of a R vector.

rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

Convert it to a time series object.

rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)

Print the timeseries data.

print(rainfall.timeseries)

Give the chart file a name.

png(file = "rainfall.png")

Plot a graph of the time series.

plot(rainfall.timeseries)

Save the file.

dev.off()

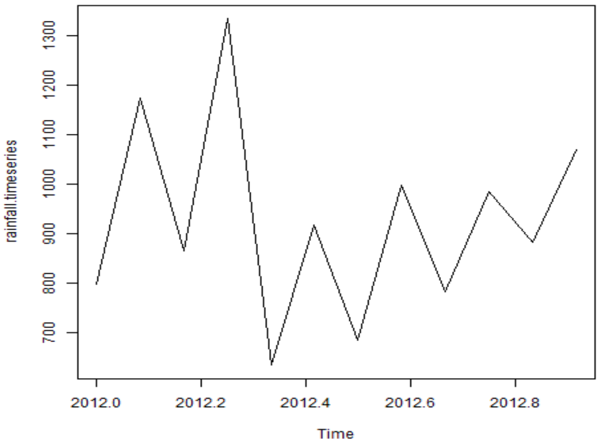
output

Jan Feb Mar Apr May Jun Jul Aug Sep

2012 799.0 1174.8 865.1 1334.6 635.4 918.5 685.5 998.6 784.2

Oct Nov Dec

2012 985.0 882.8 1071.0



Different Time Intervals

- frequency = 12 pegs the data points for every month of a year.
- frequency = 4 pegs the data points for every quarter of a year.
- frequency = 6 pegs the data points for every 10 minutes of an hour.
- frequency = 24*6 pegs the data points for every 10 minutes of a day.

SPATIAL DATA

Over time, there was an increasing number of contributed packages for handling and analyzing

General advantages of Command Line Interface (CLI) software include:

- Automation—Doing otherwise unfeasible repetitive tasks
- Reproducibility—Precise control of instructions to the computer

Moreover, specific strengths of R as a GIS are:

- R capabilities in data processing and visualization, combined with dedicated packages for spatial data
- A single environment encompassing all analysis aspects—acquiring data, computation, statistics, visualization, Web, etc.

Nevertheless, there are situations when other tools are needed:

- Interactive editing or georeferencing (but see mapedit package)
- Unique GIS algorithms (3D analysis, label placement)
- Data that cannot fit in RAM (but R can connect to spatial databases[4] and other software for working with big data)

Input and output of spatial data

Package sf combined with RPostgreSQL can be used to read from, and write to, a PostGIS spatial database:

```
library(sf)

library(RPostgreSQL)

con = dbConnect(

  PostgreSQL(),

  dbname = "gisdb",

  host = "159.89.13.241",

  port = 5432,

  user = "geobgu",

  password = "*******"

)

dat = st_read(con, query = "SELECT name_lat, geometry FROM plants LIMIT 5;")

dat

## Simple feature collection with 5 features and 1 field
```

```
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: 35.1397 ymin: 31.44711 xmax: 35.67976 ymax: 32.77013
## old-style crs object detected; please recreate object with a recent sf::st_crs()
## Geodetic CRS:  WGS 84
##          name_lat                  geometry
## 1     Iris haynei POINT (35.67976 32.77013)
## 2     Iris haynei    POINT (35.654 32.74137)
## 3 Iris atrofusca POINT (35.19337 31.44711)
## 4 Iris atrofusca POINT (35.18914 31.51475)
## 5   Iris vartanii  POINT (35.1397 31.47415)
```

Data Transformations: Converting Numeric Variables into Factors, Date Operations, String Parsing, Geocoding

DATE OPERATIONS

Dates are represented by the Date class and can be coerced from a character string using the as.Date() function. This is a common way to end up with a Date object in R.

```
> ## Coerce a 'Date' object from character
> x <- as.Date("1970-01-01")
> x
[1] "1970-01-01"
```

You can see the internal representation of a Date object by using the unclass() function.

```
> unclass(x)
[1] 0
> unclass(as.Date("1970-01-02"))
```

output

```
[1] 1
```

You can use mathematical operations on dates and times. Well, really just + and -. You can do comparisons too (i.e. ==, <=)

```
> x <- as.Date("2012-01-01")

> y <- strptime("9 Jan 2011 11:34:21", "%d %b %Y %H:%M:%S")

> x-y

Warning: Incompatible methods ("-.Date", "-.POSIXt") for "-"

Error in x - y: non-numeric argument to binary operator

> x <- as.POSIXlt(x)

> x-y

Time difference of 356.3095 days
```

The nice thing about the date/time classes is that they keep track of all the annoying things about dates and times, like leap years, leap seconds, daylight savings, and time zones.

Here's an example where a leap year gets involved.

```
> x <- as.Date("2012-03-01")

> y <- as.Date("2012-02-28")

> x-y
```

output

Time difference of 2 day

R : CONVERTING MULTIPLE NUMERIC VARIABLES TO FACTOR

In R, you can convert multiple numeric variables to factor using lapply function. The lapply function is a part of apply family of functions. They perform multiple iterations (loops) in R. In R, categorical variables need to be set as factor variables. Some of the numeric variables which are categorical in nature need to be transformed to factor so that R treats them as a grouping variable.

Converting Numeric Variables to Factor

1. Using Column Index Numbers

In this case, we are converting first, second, third and fifth numeric variables to factor variables. mydata is a data frame.

```
names <- c(1:3,5)
mydata[,names] <- lapply(mydata[,names] , factor)
str(mydata)
```

2. Using Column Names

In this case, we are converting two variables 'Credit' and 'Balance' to factor variables.

```
names <- c('Credit' ,'Balance')
mydata[,names] <- lapply(mydata[,names] , factor)
str(mydata)
```

3. Converting all variables

```
col_names <- names(mydata)
mydata[,col_names] <- lapply(mydata[,col_names] , factor)
```

4. Converting all numeric variables

```
mydata[sapply(mydata, is.numeric)] <- lapply(mydata[sapply(mydata, is.numeric)], as.factor)
```

5. Checking unique values in a variable and convert to factor only those variables having unique count less than 4

```
col_names <- sapply(mydata, function(col) length(unique(col)) < 4)
mydata[ , col_names] <- lapply(mydata[ , col_names] , factor)
```

Splitting Strings in R programming – strsplit() method

strsplit() method in [R Programming Language]() is used to split the string by using a delimiter.

strsplit() Syntax:

Syntax: strsplit(string, split, fixed)

Parameters:

- string: Input vector or string.
- split: It is a character of string to being split.
- fixed: Match the split or use the regular expression.

Return: Returns the list of words or sentences after split.

Splitting Strings in R language Example

Example 1: Using strsplit() function with delimiter

Here, we are using strsplit() along with delimiter, delimiter is a character of an existing string to being removed from the string and display out.

```
# R program to split a string

# Given String
```

```
gfg < - "cmrtc For cmrtc"
# Using strsplit() method
answer < - strsplit(gfg, " ")

print(answer)
```

Output:

[1] "cmrtc" "For"   "cmrtc"

Example 2: Splitting the dates using strsplit() function in R

We can also manipulate with date using strsplit(), only we need to understand the date formatting, for example in this date(2-07-2020) following the same pattern (-), so we can remove them using delimiter along with “-“.

Output:

```
 [[1]]

[1] "2"     "07"    "2020"

 [[2]]

[1] "5"     "07"    "2020"

 [[3]]

[1] "6"     "07"    "2020"

 [[4]]

[1] "7"     "07"    "2020"

 [[5]]

[1] "8"     "07"    "2020"
```

EXPERIMENT 11

week11

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

R – handling Missing Values

Missing values are practical in life. For example, some cells in spreadsheets are empty. If an insensible or impossible arithmetic operation is tried then NAs occur.

Dealing Missing Values in R

Missing Values in R, are handled with the use of some pre-defined functions:

is.na() Function for Finding Missing values:
A logical vector is returned by this function that indicates all the NA values present. It returns a Boolean value. If NA is present in a vector it returns TRUE else FALSE.

```
x<- c(NA, 3, 4, NA, NA, NA)

is.na(x)
```

Output:

```
[1]   TRUE FALSE FALSE   TRUE   TRUE   TRUE
```

is.nan() Function for Finding Missing values:
A logical vector is returned by this function that indicates all the NaN values present. It returns a Boolean value. If NaN is present in a vector it returns TRUE else FALSE.

```
x<- c(NA, 3, 4, NA, NA, 0 / 0, 0 / 0)

is.nan(x)
```

Output:

```
[1] FALSE FALSE FALSE FALSE FALSE   TRUE   TRUE
```

Removing NA or NaN values

There are two ways to remove missing values:

Extracting values except for NA or NaN values:
Example :

```
x <- c(1, 2, 0 / 0, 3, NA, 4, 0 / 0)
x
x[! is.na(x)]
```

Output:

```
[1]    1    2 NaN    3   NA    4 NaN
```

```
[1] 1 2 3 4
```

Data Manipulation

Data manipulation involves modifying data to make it easier to read and to be more organized. We manipulate data for analysis and visualization. It is also used with the term 'data exploration' which involves organizing data using available sets of variables.

Data Manipulation in R With dplyr Package

There are different ways to perform data manipulation in R, such as using Base R functions like subset(), with(), within(), etc., Packages like data.table, ggplot2, reshape2, readr, etc., and different Machine Learning algorithms.

Following are some of the important functions included in the dplyr package

select() :- To select columns (variables)

filter() :-To filter (subset) rows.

mutate() :-To create new variables

summarise() :- To summarize (or aggregate) data

arrange() :- To sort data

To install the dplyr package, run the following command:

```
install.packages("dplyr")
#To load dplyr package

library("dplyr")

#To load datasets package

library("datasets")

#To load iris dataset

data(iris)
```

summary(iris)

Output:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa: 50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	versicolor:0.300	versicolor:50
Median: 5.800	Median: 3.000	Median: 4.350	Median: 1.300	virginica: 50
Mean: 5.843	Mean: 3.057	Mean: 3.758	Mean: 1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

It contains 150 samples of three plant species (setosa, virginica, and versicolor) and four features measured for each sample.

Filter()

It is used to find rows with matching criteria. It also works like the select() function, i.e., we pass a data frame along with a condition separated by a comma.
For example:

#To select the first 3 rows with Species as setosa

filtered <- filter(iris, Species == "setosa")

head(filtered,3)

Output:

Sl. No.	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

Mutate()

It creates new columns and preserves the existing columns in a dataset.

For example:

#To create a column “Greater.Half” which stores TRUE if given condition

is TRUE

col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)

tail(col1)

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
145	6.7	3.3	5.7	2.5	Virginica	FALSE
146	6.7	3.0	5.2	2.3	Virginica	FALSE
147	6.3	2.5	5.0	1.9	Virginica	FALSE
148	6.5	3.0	5.2	2.0	Virginica	FALSE
149	6.2	3.4	5.4	2.3	Virginica	TRUE
150	5.9	3.0	5.1	1.8	Virginica	TRUE

Arrange()

It is used to sort rows by variables in both an ascending and descending order.

For example:

#To arrange Sepal Width in ascending order

arranged <- arrange(col1, Sepal.Width)

head(arranged)

#To arrange Sepal Width in descending order

arranged <- arrange(col1, desc(Sepal.Width))

head(arranged)

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
1	5.0	2.0	3.5	1.0	Versicolor	FALSE
2	6.0	2.2	4.0	1.0	Versicolor	FALSE
3	6.2	2.2	4.5	1.5	Versicolor	FALSE
4	6.0	2.2	5.0	1.5	Virginica	FALSE
5	4.5	2.3	1.3	0.3	Setosa	TRUE
6	5.5	2.3	4.0	1.3	Versicolor	FALSE

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
1	5.7	4.4	1.5	0.4	Setosa	TRUE
2	5.5	4.2	1.4	0.2	Setosa	TRUE
3	5.2	4.1	1.5	0.1	Setosa	TRUE
4	5.8	4.0	1.2	0.2	Setosa	TRUE
5	5.4	3.9	1.7	0.4	Setosa	TRUE
6	5.4	3.9	1.3	0.4	Setosa	TRUE

Summarise()

It is used to find insights(mean, median, mode, etc.) from a dataset. It reduces multiple values down to a single value.

For example:

summarised <- summarise(arranged, Mean.Width = mean(Sepal.Width))

head(summarised)

Output:

Mean.Width

1   3.057333

Installing packages in R:

install.packages("dplyr")

Creating a data frame:

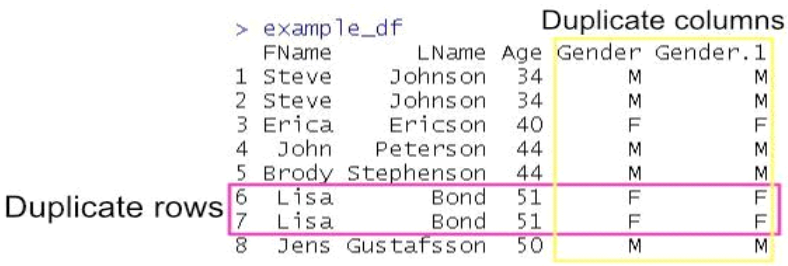
example_df <- data.frame(FName = c('Steve', 'Steve', 'Erica', 'John', 'Brody', 'Lisa', 'Lisa', 'Jens'),

LName = c('Johnson', 'Johnson', 'Ericson','Peterson', 'Stephenson', 'Bond', 'Bond', 'Gustafsson'),

Age = c(34, 34, 40, 44, 44, 51, 51, 50),

Gender = c('M', 'M', 'F', 'M', 'M', 'F', 'F', 'M'),

Gender = c('M', 'M', 'F', 'M', 'M', 'F', 'F', 'M')



```
> example_df
  FName      LName Age Gender Gender.1
1 Steve    Johnson  34      M        M
2 Steve    Johnson  34      M        M
3 Erica    Ericson  40      F        F
4  John   Peterson  44      M        M
5 Brody Stephenson  44      M        M
6  Lisa       Bond  51      F        F
7  Lisa       Bond  51      F        F
8  Jens Gustafsson  50      M        M
```


R Data Frame with duplicate rows and columns

Example 1: Delete Duplicates in R using dplyr’s distinct() Function

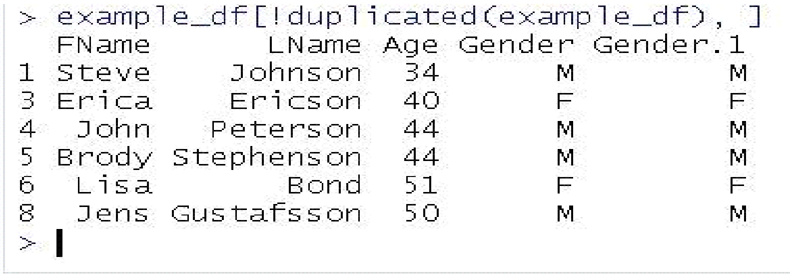
Here’s how to drop duplicates in R with the distinct() function:

Deleting duplicates with dplyr

ex_df.un <- example_df %>%

distinct()

Code language: R (r)



```
> example_df[!duplicated(example_df), ]
  FName       LName Age Gender Gender.1
1 Steve     Johnson  34      M        M
3 Erica     Ericson  40      F        F
4  John    Peterson  44      M        M
5 Brody Stephenson  44      M        M
6  Lisa        Bond  51      F        F
8  Jens  Gustafsson  50      M        M
>
```

In the image above, we can see that two columns has been removed.

Outliers

Max and min can also be used to detect outliers. An outlier is a data point that differs from rest of the observations.

Example

- # Create a matrix
  thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

  # Print the matrix
  thismatrix

    You can also create a matrix with strings:

- Example
- thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

- Access Matrix Items
- You can access the items by using [ ] brackets. The first number "1" in the bracket specifies the row-position, while the second number "2" specifies the column-position:
- Example
- thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

  thismatrix[1, 2]
- The whole row can be accessed if you specify a comma after the number in the bracket:
- Example

- thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

  thismatrix[2,]

- The whole column can be accessed if you specify a comma before the number in the bracket:

  Example

- thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

  thismatrix[,2]

Example

- thismatrix <-
  matrix(c("apple", "banana", "cherry", "orange", "mango", "pineapple"), nrow = 3, ncol =2)

  #Remove the first row and the first column
  thismatrix <- thismatrix[-c(1), -c(1)]

  thismatrix

- Check if an Item Exists

- To find out if a specified item is present in a matrix, use the %in% operator:

  Example

- Check if "apple" is present in the matrix:

- thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

  "apple" %in% thismatri

- Amount of Rows and Columns

- Use the dim() function to find the amount of rows and columns in a Matrix:

  Example

- thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

  dim(thismatrix)

- Matrix Length

- Use the length() function to find the dimension of a Matrix:

Spell Checking Packages

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:

> spelling::spell_check_package("~/workspace/writexl")

WORD FOUND **IN**

booleans write_xlsx.Rd:21

xlsx write_xlsx.Rd:6,18

> spelling::update_wordlist("~/workspace/writexl")

The following words will be added to the wordlist:

- booleans

- xlsx

Words added to this file are ignored in the spell check, making it easier to catch actual spelling errors:

> spell_check_package("~/workspace/writexl")

No spelling errors found.

The package also includes a cool function

spell_check_setup() which adds a unit test to your package that automatically runs the spell check.

> spelling::spell_check_setup("~/workspace/writexl")

No changes required to /Users/jeroen/workspace/writexl/inst/WORDLIST

Updated /Users/jeroen/workspace/writexl/tests/spelling.R

EXPERIMENT 12

week12

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

Working with Databases in R Programming

In R programming Language, a number of datasets are passed to the functions to visualize them using statistical computing. So, rather than creating datasets again and again in the console, we can pass those normalized datasets from relational databases.

Databases in R Programming Language

R can be connected to many relational databases such as Oracle, MySQL, SQL Server, etc, and fetches the result as a data frame. Once the result set is fetched into data frame, it becomes very easy to visualize and manipulate them. In this article, we'll discuss MySQl as reference to connect with R, creating, dropping, inserting, updating, and querying the table using R Language.

RMySQL Package

It is a built-in package in R and Its provides connectivity between the R and MySql databases.
It can be installed with the following commands:
install.packages("RMySQL")
Connecting MySQL with R Programming Language
R requires RMySQL package to create a connection object which takes username, password, hostname and database name while calling the function. dbConnect() function is used to create the connection object in R.
Syntax: dbConnect(drv, user, password, dbname, host)
Parameter values:
drv represents Database Driver
user represents username
password represents password value assigned to Database server
dbname represents name of the database

host represents host name

Example:

Install package

install.packages("RMySQL")

loading library

library("RMySQL")

Create connection

mysqlconn = dbConnect(MySQL(), user = 'root', password = 'welcome',

dbname = 'GFG', host = 'localhost')

Show tables in database

dbListTables(mysqlconn)

Tables present in the given database:

```
mysql> USE GFG;
Database changed
mysql> SHOW TABLES;
+----------------+
| Tables_in_gfg |
+----------------+
| articles      |
+----------------+
1 row in set (0.00 sec)

mysql>
```

Output:

Loading required package: DBI

[1] "articles"

How to Import SPSS Files into R

In this article, we are going to see how to import SPSS Files(.sav files) into R Programming Language.

Method 1: Using haven Package

Here we will use the haven package to import the SAS files.

To install the package:

```
install.packages('haven')
```

To import the SAV file read_sav() methods are capable to read the file.

Syntax:

read_sav('file')

Example: Reading SPSS file

```
# import lib
library(haven)
data <- read_sav("airline_passengers.sav")
head(data)
```

Output:

number

112

118

132

129

121

135

SAS

In this article, we are going to see how to import SAS files(.sas7bdat) into R Programming Language.

SAS stands for Statistical Analysis Software, it contains SAS program code saved in standard ASCII text format.

Method 1: Using haven Package

Here we will use the **haven** package to import the SAS files.

To install the package:

```
install.packages('haven')
```

To import the SAS file read_sas() methods are capable to read the file.

Syntax:

read_sas('file')

Example: Reading SAS file

import lib

library(haven)

import data

data <- read_sas('lond_small.sas7bdat')

display data

Data

Output:

```
# A tibble: 500 x 10
   WFOOD  WFUEL  WCLOTH    WALC   WTRANS WOTHER TOTEXP INCOME   AGE    NK
   <dbl>  <dbl>   <dbl>   <dbl>    <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
 1 0.412 0.244  0.0219  0.0220 0.191     0.109     70    120    36     2
 2 0.502 0.177  0.00690 0.0180 0.00800   0.287     60     90    39     2
 3 0.349 0.169  0.128   0.0319 0.0517    0.270     70     90    29     2
 4 0.423 0.169  0.00400 0.0715 0.0549    0.277     90    140    42     2
 5 0.245 0.0214 0.00170 0.0391 0.464     0.229     90    120    38     2
 6 0.271 0.108  0.302   0.0265 0.153     0.139    150    150    32     2
 7 0.345 0.0384 0.0645  0.0255 0.313     0.213    110    190    39     2
 8 0.424 0.0868 0.0681  0.0299 0.106     0.285     90     90    25     2
 9 0.367 0.112  0.128   0       0.156     0.237    110    140    57     2
10 0.423 0.0298 0.0886  0.0388 0.257     0.163    160    120    46     2
# ... with 490 more rows
```

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)

That's good. Our **'googlesheets4'** library is now ready to pull the data from google sheets.

1. Setup the Authorization

#Read google sheets data into R

x <-

read_sheet('https://docs.google.com/spreadsheets/d/1J9-ZpmQT_oxLZ4kfe5gRvBs7vZhEGhSCIpNS78XOQUE/edit?usp=sharing')

Is it OK to cache OAuth access credentials in the folder

1: Yes

2: No

You have to select option 1: YES to continue to the authorization process.

As a first step, if you are having multiple G accounts logged in, it will ask you to continue with your account as shown below.

Account Sign In

- You have to select your account to authorise R to access the G sheets. This process is followed by multiple authorizations. You have to allow R to in all those steps.

Access

- In the below picture, you will be shown the permissions you are giving to the Tidyverse API. Click **“Allow”** and you are done.

Access Authorization

- After the successful authorization, you can see the completion message.

Authorization Success

- After this, you will see a successful authorization message in the R studio as shown be

Implementation of Web Scraping using R

Web Scraping in R with rvest

rvest maintained by the legendary Hadley Wickham. We can easily scrape data from webpage from this library.

Import rvest libraries

Before starting we will import the rvest library

library(rvest)

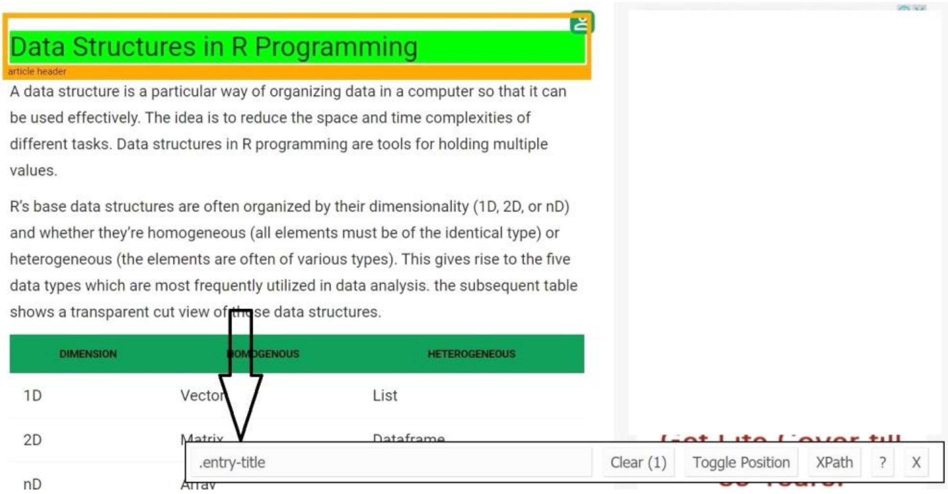
Read HTML Code

Read the HTML code from the webpage using **read_html()**

webpage = read_html("https://www.google.co.in//\
data-structures-**in**-r-programming")

Scrape Data From HTML Code

Now, let’s start by scraping the heading field. For that, use the selector gadget to get the specific CSS selectors that enclose the heading. One can click on the extension in his/her browser and select the heading field with the cursor.



Once one knows the CSS selector that contains the heading, he/she can use this simple R code to get the heading

Using CSS selectors to scrape the heading section

heading = html_node(webpage, '.entry-title')

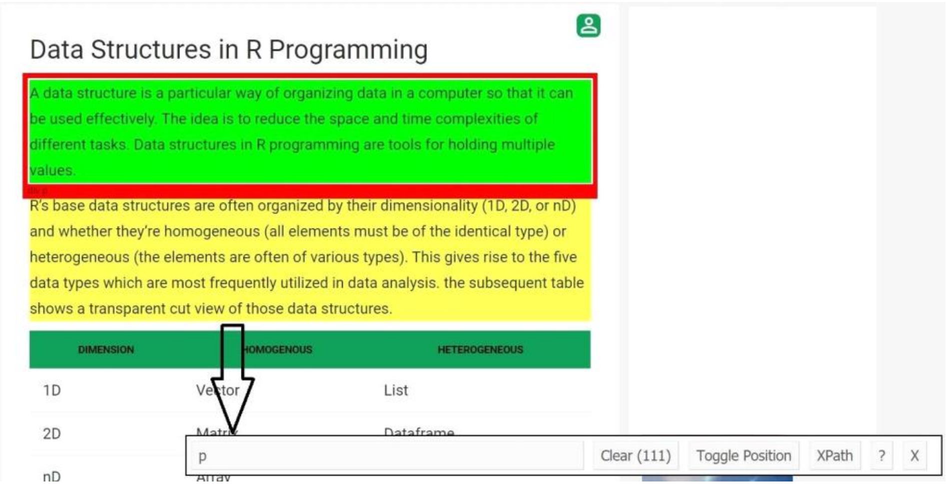
Converting the heading data to text

text = html_text(heading)

print(text)

Output:

[1] "Data Structures in R Programming



Once one knows the CSS selector that contains the paragraphs, he/she can use this simple R code to get all the paragraphs.

Using CSS selectors to scrape

all the paragraph section

Note that we use html_nodes() here

paragraph = html_nodes(webpage, 'p')

```
# Converting the heading data to text
pText = html_text(paragraph)
# Print the top 6 data
print(head(pText))
```

Output:

[1] "A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values. "

[2] "R's base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the five data types which are most frequently utilized in data analysis. the subsequent table shows a transparent cut view of those data structures."

[3] "The most essential data structures used in R include:"

[4] "A vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures.

The complete code for Web Scraping using R Language

```
# R program to illustrate
# Web Scraping
 # Import rvest library
library(rvest)
# Reading the HTML code from the website
webpage = read_html("https://www.google.co.in/ /
data-structures-in-r-programming")
# Using CSS selectors to scrape the heading section
heading = html_node(webpage, '.entry-title')
# Converting the heading data to text
text = html_text(heading)
print(text)
# Using CSS selectors to scrape
# all the paragraph section
```

# Note that we use html_nodes() here
paragraph = html_nodes(webpage, 'p')

Accessing REST API using R Programming

REST(Representational state transfer) API is an architectural style that includes specific constraints for building APIs to ensure that they are consistent, efficient, and scalable. REST API is a collection of syntax and constraints which are used in the development and operation of web services that include sending & receiving information through their **endpoint** i.e a URL providing an interface to the external environment.

REST was first presented by Roy Fielding in 2000. The abstraction of information in REST is termed as a resource. REST uses a resource identifier to identify the particular resource in an interaction between components. It allows an application to access resources or functionality available on another server that is remote to that application's architectural and security domain.

Implementation in R

The API if of movie **Guardians of Galaxy Vol. 2** was released in the year 2017 on 05th May of runtime 136 minutes directed by James Gun. It was Action, Comedy, Adventure, and Sci-fi movie. The API is of OMDb which is an open web service that hosts movie information.

Installing the packages

install.packages("httr")

install.packages("jsonlite")

Loading packages

library(httr)

library(jsonlite)

Initializing API Call

call <- "http://www.omdbapi.com/?i=tt3896198&apikey=948d3551&plot=full&r=json"

Accessing movie API using packages

Accessing the movie API using the httr package and jsonlite package in R.

Output:

```
# Getting details in API

get_movie_details <- GET(url = call)

 # Getting status of HTTP Call

status_code(get_movie_details)

 # Content in the API

str(content(get_movie_details))

 # Converting content to text

get_movie_text <- content(get_movie_details,"text", encoding = "UTF-8")

get_movie_text

 # Parsing data in JSON

get_movie_json <- fromJSON(get_movie_text,flatten = TRUE)

get_movie_json

# Converting into dataframe

get_movie_dataframe <- as.data.frame(get_movie_json)
```

- **Status_code:**

```
> status_code(get_movie_details)
[1] 200
```

Status code 200 shows that data of API is successfully requested, responded, and received.

- **Content in the API:**

```
> str(content(get_movie_details))
List of 25
 $ Title     : chr "Guardians of the Galaxy Vol. 2"
 $ Year      : chr "2017"
 $ Rated     : chr "PG-13"
 $ Released  : chr "05 May 2017"
 $ Runtime   : chr "136 min"
 $ Genre     : chr "Action, Adventure, Comedy, Sci-Fi"
 $ Director  : chr "James Gunn"
 $ Writer    : chr "James Gunn, Dan Abnett (based on the Marvel comics by), Andy Lanning (bas
ed on the Marvel comics by), Steve Eng"| __truncated__
 $ Actors    : chr "Chris Pratt, Zoe Saldana, Dave Bautista, Vin Diesel"
 $ Plot      : chr "After saving Xandar from Ronan's wrath, the Guardians are now recognized
 as heroes. Now the team must help thei"| __truncated__
 $ Language  : chr "English"
 $ Country   : chr "USA"
 $ Awards    : chr "Nominated for 1 Oscar. Another 15 wins & 56 nominations."
 $ Poster    : chr "https://m.media-amazon.com/images/M/MV5BNjM0NTc0NzItM2FlYS00YzEwLWE0YmUtN
TA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzI"| __truncated__
 $ Ratings   :List of 3
  ..$ :List of 2
  .. ..$ Source: chr "Internet Movie Database"
  .. ..$ Value : chr "7.6/10"
  ..$ :List of 2
  .. ..$ Source: chr "Rotten Tomatoes"
  .. ..$ Value : chr "85%"
  ..$ :List of 2
  .. ..$ Source: chr "Metacritic"
  .. ..$ Value : chr "67/100"
 $ Metascore : chr "67"
 $ imdbRating: chr "7.6"
 $ imdbVotes : chr "548,890"
 $ imdbID    : chr "tt3896198"
 $ Type      : chr "movie"
 $ DVD       : chr "22 Aug 2017"
 $ BoxOffice : chr "$389,804,217"
 $ Production: chr "Walt Disney Pictures"
```

The requested API data is displayed using the **content()** function.

- **get_movie_text:**

```
> get_movie_text
[1] "{\"Title\":\"Guardians of the Galaxy Vol. 2\",\"Year\":\"2017\",\"Rated\":\"PG-13\",\"Re
leased\":\"05 May 2017\",\"Runtime\":\"136 min\",\"Genre\":\"Action, Adventure, Comedy, Sci-F
i\",\"Director\":\"James Gunn\",\"Writer\":\"James Gunn, Dan Abnett (based on the Marvel comi
cs by), Andy Lanning (based on the Marvel comics by), Steve Englehart (Star-Lord created by),
 Steve Gan (Star-Lord created by), Jim Starlin (Gamora and Drax created by), Stan Lee (Groot
 created by), Larry Lieber (Groot created by), Jack Kirby (Groot created by), Bill Mantlo (Ro
cket Raccoon created by), Keith Giffen (Rocket Raccoon created by), Steve Gerber (Howard the
 Duck created by), Val Mayerik (Howard the Duck created by)\",\"Actors\":\"Chris Pratt, Zoe S
aldana, Dave Bautista, Vin Diesel\",\"Plot\":\"After saving Xandar from Ronan's wrath, the Gu
ardians are now recognized as heroes. Now the team must help their leader Star Lord (Chris Pr
att) uncover the truth behind his true heritage. Along the way, old foes turn to allies and b
etrayal is blooming. And the Guardians find that they are up against a devastating new menace
 who is out to rule the galaxy.\",\"Language\":\"English\",\"Country\":\"USA\",\"Awards\":\"N
ominated for 1 Oscar. Another 15 wins & 56 nominations.\",\"Poster\":\"https://m.media-amazo
n.com/images/M/MV5BNjM0NTc0NzItM2FlYS00YzEwLWE0YmUtNTA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzIyNz
g@._V1_SX300.jpg\",\"Ratings\":[{\"Source\":\"Internet Movie Database\",\"Value\":\"7.6/10
\"},{\"Source\":\"Rotten Tomatoes\",\"Value\":\"85%\"},{\"Source\":\"Metacritic\",\"Value
\":\"67/100\"}],\"Metascore\":\"67\",\"imdbRating\":\"7.6\",\"imdbVotes\":\"548,890\",\"imdbI
D\":\"tt3896198\",\"Type\":\"movie\",\"DVD\":\"22 Aug 2017\",\"BoxOffice\":\"$389,804,217
\",\"Production\":\"Walt Disney Pictures\",\"Website\":\"N/A\",\"Response\":\"True\"}"
```