**Deccan Education Society’s**

**Kirti M. Doongursee College of Arts, Science and Commerce [NAAC Accredited: “A Grade”]**

**A picture containing text

Description automatically generated**

**T.Y.B.Sc. [Computer Science]**

**Practical Journal**

**USCSP602**

**Seat Number [681494]**

**Roll No:- 59**

**Department of Computer Science and Information Technology**

**Department of Computer Science and Information Technology**

**Deccan Education Society’s**

**Kirti M. Doongursee College of Arts, Science and Commerce[NAAC Accredited: “A Grade”]**

**C E R T I F I C A T E**

This is to certify that **Miss. Shamika Vinod Murkar** of T.Y.B.Sc. (Computer  Science) with seat no. **681494** & roll no. **59** has completed 12 practicals of Paper- USCSP602 under my supervision in this College during the year 2021-2022.

**Lecturer-In-Charge                                                          H.O.D.**

**Department of**

**Computer Science & IT**

Date: / /2022

**Examined by:                                                         Remarks:**

|  |  |  |
| --- | --- | --- |
| **DATE** | **TOPIC** | **SIGNATURE** |
| 03-01-2022 | **Practical 1 : Data types with r language(vectors)** |  |
| 05-01-2022 | **Practical 2: Data Cleaning and EDA** |  |
| 11-01-2022 | **Practical 3: Data Visualization** |  |
| 12-01-2022 | **Practical 4: Data Import with readr and parsing functions** |  |
| 17-01-2022 | **Practical 5: Data Modelling** |  |
| 18-01-2022 | **Practical 6: Regression Techniques and Normal Distribution** |  |
| 25-01-2022 | **Practical 7: Principal Component Analysis** |  |
| 25-01-2022 | **Practical 8: LASSO Regression** |  |
| 31-01-2022 | **Practical 9: Time Series Forecasting** |  |
| 02-02-2022 | **Practical 10: K-means clustering** |  |
| 09-02-2022 | **Practical 11: Hierarchical clustering** |  |
| 16-02-2022 | **Practical 12: Ensemble Method** |  |

**PRACTICAL 01**

Data Structures in R

Vectors

When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

# Create a vector.

apple <- c('red','green',"yellow")

print(apple)

# Get the class of the vector.

print(class(apple))

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

[1] "red" "green" "yellow"

[1] "character"

# Creating a sequence from 5 to 13.

v <- 5:13

print(v)

# Creating a sequence from 6.6 to 12.6.

v <- 6.6:12.6

print(v)

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

v <- 3.8:11.4

print(v)

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

[1] 5 6 7 8 9 10 11 12 13

[1] 6.6 7.6 8.6 9.6 10.6 11.6 12.6

[1] 3.8 4.8 5.8 6.8 7.8 8.8 9.8 10.8

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

# Create a list.

list1 <- list(c(2,5,3),21.3,sin)

# Print the list.

print(list1)

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

[[1]]

[1] 2 5 3

[[2]]

[1] 21.3

[[3]]

function (x) .Primitive("sin")

# values.

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

print(list\_data)

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

[[1]]

[1] "Red"

[[2]]

[1] "Green"

[[3]]

[1] 21 32 11

[[4]]

[1] TRUE

[[5]]

[1] 51.23

[[6]]

[1] 119.1

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

# Create a matrix.

M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)

print(M)

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

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

[1,] "a" "a" "b"

[2,] "c" "b" "a"

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

# Create an array.

a <- array(c('green','yellow'),dim = c(3,3,2))

print(a)

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

, , 1

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

[1,] "green" "yellow" "green"

[2,] "yellow" "green" "yellow"

[3,] "green" "yellow" "green"

, , 2

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

[1,] "yellow" "green" "yellow"

[2,] "green" "yellow" "green"

[3,] "yellow" "green" "yellow"

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor()** function. The **nlevels** functions gives the count of levels.

# Create a vector.

apple\_colors <- c('green','green','yellow','red','red','red','green')

# Create a factor object.

factor\_apple <- factor(apple\_colors)

# Print the factor.

print(factor\_apple)

print(nlevels(factor\_apple))

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

[1] green green yellow red red red green

Levels: green red yellow

[1] 3

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

# Create the data frame.

BMI <- data.frame(

gender = c("Male", "Male","Female"),

height = c(152, 171.5, 165),

weight = c(81,93, 78),

Age = c(42,38,26)

)

print(BMI)

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

gender height weight Age

1 Male 152.0 81 42

2 Male 171.5 93 38

3 Female 165.0 78 26

**PRACTICAL 02**

**Data Cleaning and EDA**

How to read different data?

A binary file is a file that contains information stored only in form of bits and bytes.(0’s and 1’s). They are not human readable as the bytes in it translate to characters and symbols which contain many other non-printable characters. Attempting to read a binary file using any text editor will show characters like Ø and ð.

The binary file has to be read by specific programs to be useable. For example, the binary file of a Microsoft Word program can be read to a human readable form only by the Word program. Which indicates that, besides the human readable text, there is a lot more information like formatting of characters and page numbers etc., which are also stored along with alphanumeric characters. And finally a binary file is a continuous sequence of bytes. The line break we see in a text file is a character joining first line to the next.

Sometimes, the data generated by other programs are required to be processed by R as a binary file. Also R is required to create binary files which can be shared with other programs.

R has two functions **WriteBin()** and **readBin()** to create and read binary files.

Syntax

writeBin(object, con)

readBin(con, what, n )

Following is the description of the parameters used −

* **con** is the connection object to read or write the binary file.
* **object** is the binary file which to be written.
* **what** is the mode like character, integer etc. representing the bytes to be read.
* **n** is the number of bytes to read from the binary file.

Example

We consider the R inbuilt data "mtcars". First we create a csv file from it and convert it to a binary file and store it as a OS file. Next we read this binary file created into R.

Writing the Binary File

We read the data frame "mtcars" as a csv file and then write it as a binary file to the OS.

# Read the "mtcars" data frame as a csv file and store only the columns

"cyl", "am" and "gear".

write.table(mtcars, file = "mtcars.csv",row.names = FALSE, na = "",

col.names = TRUE, sep = ",")

# Store 5 records from the csv file as a new data frame.

new.mtcars <- read.table("mtcars.csv",sep = ",",header = TRUE,nrows = 5)

# Create a connection object to write the binary file using mode "wb".

write.filename = file("/web/com/binmtcars.dat", "wb")

# Write the column names of the data frame to the connection object.

writeBin(colnames(new.mtcars), write.filename)

# Write the records in each of the column to the file.

writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)

# Close the file for writing so that it can be read by other program.

close(write.filename)

Reading the Binary File

The binary file created above stores all the data as continuous bytes. So we will read it by choosing appropriate values of column names as well as the column values.

# Create a connection object to read the file in binary mode using "rb".

read.filename <- file("/web/com/binmtcars.dat", "rb")

# First read the column names. n = 3 as we have 3 columns.

column.names <- readBin(read.filename, character(), n = 3)

# Next read the column values. n = 18 as we have 3 column names and 15 values.

read.filename <- file("/web/com/binmtcars.dat", "rb")

bindata <- readBin(read.filename, integer(), n = 18)

# Print the data.

print(bindata)

# Read the values from 4th byte to 8th byte which represents "cyl".

cyldata = bindata[4:8]

print(cyldata)

# Read the values form 9th byte to 13th byte which represents "am".

amdata = bindata[9:13]

print(amdata)

# Read the values form 9th byte to 13th byte which represents "gear".

geardata = bindata[14:18]

print(geardata)

# Combine all the read values to a dat frame.

finaldata = cbind(cyldata, amdata, geardata)

colnames(finaldata) = column.names

print(finaldata)

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

[1] 7108963 1728081249 7496037 6 6 4

[7] 6 8 1 1 1 0

[13] 0 4 4 4 3 3

[1] 6 6 4 6 8

[1] 1 1 1 0 0

[1] 4 4 4 3 3

cyl am gear

[1,] 6 1 4

[2,] 6 1 4

[3,] 4 1 4

[4,] 6 0 3

[5,] 8 0 3

As we can see, we got the original data back by reading the binary file in R.

XML is a file format which shares both the file format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It stands for Extensible Markup Language (XML). Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in xml the markup tags describe the meaning of the data contained into he file.

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>

## Reading XML File

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

### Get Number of Nodes Present in XML File

# Load the packages required to read XML files.

library("XML")

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.

rootnode <- xmlRoot(result)

# Find number of nodes in the root.

rootsize <- xmlSize(rootnode)

# Print the result.

print(rootsize)

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

output

[1] 8

## Details of the First Node

Let's look at the first record of the parsed file. It will give us an idea of the various elements present in the top level node.

# Load the packages required to read XML files.

library("XML")

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.

rootnode <- xmlRoot(result)

# Print the result.

print(rootnode[1])

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

$EMPLOYEE

1

Rick

623.3

1/1/2012

IT

attr(,"class")

[1] "XMLInternalNodeList" "XMLNodeList"

### Get Different Elements of a Node

# Load the packages required to read XML files.

library("XML")

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.

rootnode <- xmlRoot(result)

# Get the first element of the first node.

print(rootnode[[1]][[1]])

# Get the fifth element of the first node.

print(rootnode[[1]][[5]])

# Get the second element of the third node.

print(rootnode[[3]][[2]])

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

1

IT

Michelle

## XML to Data Frame

To handle the data effectively in large files we read the data in the xml file as a data frame. Then process the data frame for data analysis.

# Load the packages required to read XML files.

library("XML")

library("methods")

# Convert the input xml file to a data frame.

xmldataframe <- xmlToDataFrame("input.xml")

print(xmldataframe)

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

ID NAME SALARY STARTDATE 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

As the data is now available as a dataframe we can use data frame related function to read and manipulate the file.

JSON file stores data as text in human-readable format. Json stands for JavaScript Object Notation. R can read JSON files using the rjson package.

## Install rjson Package

In the R console, you can issue the following command to install the rjson package.

install.packages("rjson")

## Input Data

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

{

"ID":["1","2","3","4","5","6","7","8" ],

"Name":["Rick","Dan","Michelle","Ryan","Gary","Nina","Simon","Guru" ],

"Salary":["623.3","515.2","611","729","843.25","578","632.8","722.5" ],

"StartDate":[ "1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013",

"7/30/2013","6/17/2014"],

"Dept":[ "IT","Operations","IT","HR","Finance","IT","Operations","Finance"]

}

## Read the JSON File

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

# Load the package required to read JSON files.

library("rjson")

# Give the input file name to the function.

result <- fromJSON(file = "input.json")

# Print the result.

print(result)

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

$ID

[1] "1" "2" "3" "4" "5" "6" "7" "8"

$Name

[1] "Rick" "Dan" "Michelle" "Ryan" "Gary" "Nina" "Simon" "Guru"

$Salary

[1] "623.3" "515.2" "611" "729" "843.25" "578" "632.8" "722.5"

$StartDate

[1] "1/1/2012" "9/23/2013" "11/15/2014" "5/11/2014" "3/27/2015" "5/21/2013"

"7/30/2013" "6/17/2014"

$Dept

[1] "IT" "Operations" "IT" "HR" "Finance" "IT"

"Operations" "Finance"

## Convert JSON to a Data Frame

We can convert the extracted data above to a R data frame for further analysis using the **as.data.frame()** function.

# Load the package required to read JSON files.

library("rjson")

# Give the input file name to the function.

result <- fromJSON(file = "input.json")

# Convert JSON file to a data frame.

json\_data\_frame <- as.data.frame(result)

print(json\_data\_frame)

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

Many websites provide data for consumption by its users. For example the World Health Organization(WHO) provides reports on health and medical information in the form of CSV, txt and XML files. Using R programs, we can programmatically extract specific data from such websites. Some packages in R which are used to scrap data form the web are − "RCurl",XML", and "stringr". They are used to connect to the URL’s, identify required links for the files and download them to the local environment.

## Install R Packages

The following packages are required for processing the URL’s and links to the files. If they are not available in your R Environment, you can install them using following commands.

install.packages("RCurl")

install.packages("XML")

install.packages("stringr")

install.packages("plyr")

## Input Data

We will visit the URL [weather data](https://www.geos.ed.ac.uk/~weather/jcmb_ws/) and download the CSV files using R for the year 2015.

## Example

We will use the function **getHTMLLinks()** to gather the URLs of the files. Then we will use the function **download.file()** to save the files to the local system. As we will be applying the same code again and again for multiple files, we will create a function to be called multiple times. The filenames are passed as parameters in form of a R list object to this function.

# Read the URL.

url <- "http://www.geos.ed.ac.uk/~weather/jcmb\_ws/"

# Gather the html links present in the webpage.

links <- getHTMLLinks(url)

# Identify only the links which point to the JCMB 2015 files.

filenames <- links[str\_detect(links, "JCMB\_2015")]

# Store the file names as a list.

filenames\_list <- as.list(filenames)

# Create a function to download the files by passing the URL and filename list.

downloadcsv <- function (mainurl,filename) {

filedetails <- str\_c(mainurl,filename)

download.file(filedetails,filename)

}

# Now apply the l\_ply function and save the files into the current R working directory.

l\_ply(filenames,downloadcsv,mainurl = "http://www.geos.ed.ac.uk/~weather/jcmb\_ws/")

## Verify the File Download

After running the above code, you can locate the following files in the current R working directory.

"JCMB\_2015.csv" "JCMB\_2015\_Apr.csv" "JCMB\_2015\_Feb.csv" "JCMB\_2015\_Jan.csv"

"JCMB\_2015\_Mar.csv"

The data is Relational database systems are stored in a normalized format. So, to carry out statistical computing we will need very advanced and complex Sql queries. But R can connect easily to many relational databases like MySql, Oracle, Sql server etc. and fetch records from them as a data frame. Once the data is available in the R environment, it becomes a normal R data set and can be manipulated or analyzed using all the powerful packages and functions.

In this, we will be using MySql as our reference database for connecting to R.

## RMySQL Package

R has a built-in package named "RMySQL" which provides native connectivity between with MySql database. You can install this package in the R environment using the following command.

install.packages("RMySQL")

## Connecting R to MySql

Once the package is installed we create a connection object in R to connect to the database. It takes the username, password, database name and host name as input.

# Create a connection Object to MySQL database.

# We will connect to the sampel database named "sakila" that comes with MySql installation.

mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'sakila',

host = 'localhost')

# List the tables available in this database.

dbListTables(mysqlconnection)

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

[1] "actor" "actor\_info"

[3] "address" "category"

[5] "city" "country"

[7] "customer" "customer\_list"

[9] "film" "film\_actor"

[11] "film\_category" "film\_list"

[13] "film\_text" "inventory"

[15] "language" "nicer\_but\_slower\_film\_list"

[17] "payment" "rental"

[19] "sales\_by\_film\_category" "sales\_by\_store"

[21] "staff" "staff\_list"

[23] "store"

## Querying the Tables

We can query the database tables in MySql using the function **dbSendQuery()**. The query gets executed in MySql and the result set is returned using the R **fetch()** function. Finally it is stored as a data frame in R.

# Query the "actor" tables to get all the rows.

result = dbSendQuery(mysqlconnection, "select \* from actor")

# Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.

data.frame = fetch(result, n = 5)

print(data.fame)

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

actor\_id first\_name last\_name last\_update

1 1 PENELOPE GUINESS 2006-02-15 04:34:33

2 2 NICK WAHLBERG 2006-02-15 04:34:33

3 3 ED CHASE 2006-02-15 04:34:33

4 4 JENNIFER DAVIS 2006-02-15 04:34:33

5 5 JOHNNY LOLLOBRIGIDA 2006-02-15 04:34:33

## Query with Filter Clause

We can pass any valid select query to get the result.

result = dbSendQuery(mysqlconnection, "select \* from actor where last\_name = 'TORN'")

# Fetch all the records(with n = -1) and store it as a data frame.

data.frame = fetch(result, n = -1)

print(data)

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

actor\_id first\_name last\_name last\_update

1 18 DAN TORN 2006-02-15 04:34:33

2 94 KENNETH TORN 2006-02-15 04:34:33

3 102 WALTER TORN 2006-02-15 04:34:33

## Updating Rows in the Tables

We can update the rows in a Mysql table by passing the update query to the dbSendQuery() function.

dbSendQuery(mysqlconnection, "update mtcars set disp = 168.5 where hp = 110")

After executing the above code we can see the table updated in the MySql Environment.

## Inserting Data into the Tables

dbSendQuery(mysqlconnection,

"insert into mtcars(row\_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)

values('New Mazda RX4 Wag', 21, 6, 168.5, 110, 3.9, 2.875, 17.02, 0, 1, 4, 4)"

)

After executing the above code we can see the row inserted into the table in the MySql Environment.

## Creating Tables in MySql

We can create tables in the MySql using the function **dbWriteTable()**. It overwrites the table if it already exists and takes a data frame as input.

# Create the connection object to the database where we want to create the table.

mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'sakila',

host = 'localhost')

# Use the R data frame "mtcars" to create the table in MySql.

# All the rows of mtcars are taken inot MySql.

dbWriteTable(mysqlconnection, "mtcars", mtcars[, ], overwrite = TRUE)

After executing the above code we can see the table created in the MySql Environment.

## Dropping Tables in MySql

We can drop the tables in MySql database passing the drop table statement into the dbSendQuery() in the same way we used it for querying data from tables.

dbSendQuery(mysqlconnection, 'drop table if exists mtcars')

After executing the above code we can see the table is dropped in the MySql Environment.

**Data Management**

1. **Read CSV file**

Read.csv() is used for reading “comma separated value” files (“.csv”) . In this the data will be imported as data frame

**df <- read.csv('train.csv',encoding = "ISO-8859-1")**

Table

Description automatically generated

1. **dim() function**

In R Language is used to get or set the dimension of the specified matrix, array or data frame

**dim(df)**



**20580 rows (Observations) & 6 Columns (Attributes or variables)**

1. **head() function**

In R Language is used to get the first parts of a vector, matrix, table, data frame or function.

**Head(df)**

Text

Description automatically generated

1. **tail() function**

In R Language is used to get the last parts of a vector, matrix, table, data frame or function

**tail(df)**

**Text

Description automatically generated**

1. **ncol() function**

In R Language is used to return the number of columns of the specified matrix.

**ncol(df)**

****

1. **nrow() function**

In R Language is used to return the number of rows of the specified matrix.

**nrow(df)**

****

1. **str() function**

In R Language is used for compactly displaying the internal structure of a R object

**str(df)**

**Text

Description automatically generated**

**Data structure(character, Integer), total rows & columns, some records of attributes are defined**

1. **glimpse() function**

This is function of dplyr package in R language can be used to see the columns of the dataset and display some portion of the data with respect to each attribute that can fit on a single line.

**glimpse(df)**

**Same as str() function, Data structure(character, Integer), total rows & columns & records of attributes that can be fitted in one single line are defined.**

**Scatter chart

Description automatically generated**

1. **summary() function**

In R Language is a generic function used to produce result summaries of the results of various model fitting functions.

**summary(df)**

**Length, class, mode, minimum, maximum, mean, 1st quartile, 3rd quartile, NA’s are defined**

**Text

Description automatically generated**

1. **is.na() function**

To find missing values you check for NA in R using the is.na() function. This function returns a value of true and false for each value in a data set

1. **sum() function**

Sum function in R – sum(), is used to calculate the sum of vector elements.

**nmiss <- sum(is.na(df$date2020-03-24))**

**nmiss**

****

1. **names() function**

in R Language is used to get or set the name of an Object.

**names(df)[names(df)=="Country\_Region"] <- "Country"**

**head(df)**

Change the name of attribute “Country\_Region” to “Country”

Text

Description automatically generated

1. **select function()**

In R language is used to choose a subset of variables or columns from a data set.

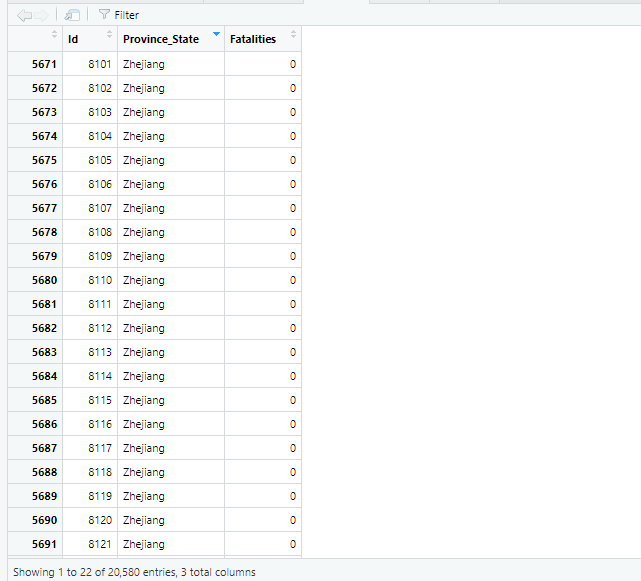
**df1 <- select(df,-(** **Country:ConfirmedCases))**

**Select columns exclude from area abbreviation to longitude**

1. **View function()**

In R Language the contents of a defined object can be viewed via the view function

**View(df1)**

****

**df1**

**Shape

Description automatically generated with medium confidence**

1. **colSums() function**

in R Language is used to compute the sums of matrix or array columns.

1. **data.frame() function**

Data Frames in R Language are generic data objects of R which are used to store the tabular data. **Missing <- data.frame(colSums(is.na(df)))**

**View(Missing)**

**Graphical user interface, table

Description automatically generated with medium confidence**

1. **mean() function**

In R Language is used to calculate the arithmetic mean of the elements of the numeric vector passed to it as argument.

**mean(df1)**

**A picture containing diagram

Description automatically generated**

1. **c() function**

In R programming c stands for 'combine. ' This function is used to get the output by giving parameters in side the function.

**df1 <- df[,c(1,2,3,)]**

**View(df1)**

**Table

Description automatically generated**

Added new attributes continent & region using c function

1. **rowSums() function**

In R Language is used to compute the sum of rows of a matrix or an array

1. **subset()**

**#Create subsets using subset function**

**#Condition on observations**

**df12 <- subset(df, Country=="Albania" & ConfirmedCases > 50)**

**df12**

**Text, table

Description automatically generated with medium confidence**

**#Condition on variable names**

**#Only country, Item & Y2012 from previous data i.e.**

**df13 <- subset(df12,select=c(Country,ConfirmedCases))**

**head(df13)**

**Table

Description automatically generated with medium confidence**

**#Condition on observations & variable names**

**#Select production details of specific set of countries**

**df14 <- subset(df,Country=="Algeria" & ConfirmedCases > 50, select=c(ConfirmedCases,Country,Fatalities))**

**head(df14)**

**Text

Description automatically generated**

**Data Soring Functions**

**#Import & attach basic df\_Data**

**df\_Data <- read.csv(file.choose(),header=T)**

**attach(df\_Data)**

**head(df\_Data)**

**Text

Description automatically generated with low confidence**

**#Sort df\_Data by Province\_State in ascending order**

**Country\_sorted\_1 <-df\_Data[order(Province\_State),]**

**Country\_sorted\_1**

**Table

Description automatically generated**

**#Sort FAO\_Data by Fatalities in descending order**

**Country\_sorted\_2 <-df\_Data[order(-Fatalities),]**

**Country\_sorted\_2**

**Table

Description automatically generated**

**#Sort df\_Data by column with characters/factors**

**#Sort df\_Data by Fatalities**

**sorted\_ft <- df\_Data[order(Fatalities),]**

**sorted\_ft**

**Table

Description automatically generated**

**#Sort data by column with characters/factors in descending order**

**#Sort df\_Data by Fatalities in descending order**

**sorted\_ft2 <- df\_Data[order(Fatalities,decreasing=TRUE),]**

**sorted\_ft2**

**Table

Description automatically generated**

**#Sort data by giving multiple columns; one column with characters/ factors & one with numericals #Sort df\_Data by item & item code**

**sorted\_FtPs <- df\_Data[order(Fatalities,Province\_State),]**

**sorted\_FtPs**

**Table

Description automatically generated**

**Perform merging / joining operations using merge function.**

**Import following two data sets**

**Test <- read.csv('Test.csv')**

**View(Test)**

**df <- read.csv('train.csv',encoding = "ISO-8859-1")**

**View(df)**

**Graphical user interface, application, table

Description automatically generated**

Table

Description automatically generated

**#Outer join includes all latitude from both data sets**

**outerjoin<- merge(df,test,by=c("Date"), all=TRUE)**

**outerjoin**

**Table

Description automatically generated**

**Innerjoin**

**#Inner Join includes country code only if present in both data sets**

**innerjoin<- merge(df,test,by="Date")**

**innerjoin**

**Table

Description automatically generated**

**Leftjoin**

**# Right Join includes all country code from first data set**

**leftjoin<- merge(df,test,by=c("Date"), all.x=TRUE)**

**leftjoin**

**Table

Description automatically generated**

**Rightjoin**

**#Right Join includes all country code from second data set**

**rightjoin<- merge(df,test,by=c("Date"), all.y=TRUE)**

**rightjoin**

**Table

Description automatically generated**

**Aggregate() function**

**in R Splits the data into subsets, computes summary statistics for each subsets and returns the result in a group by form.**

**aggregate(ConfirmedCases~Fatalities,data=df,FUN=mean)**

**Table

Description automatically generated**

**PRACTICAL 03**

**PART A : DATA VISUALIZATION**

## Global growth of confirmed cases and fatalities.

library(stringr)

*#preparation for object*

total <- data.frame()

*#iterate through month and day*

for(month in 1:3) {

for(day in 1:31) {

reqDate <- paste("2020-0", month, "-", str\_pad(day, 2, pad = "0"), sep = "")

iter <- as.data.frame(colSums(df[as.character(df$Date) == reqDate,

c("ConfirmedCases", "Fatalities")]))

iter2 <- data.frame(Num = (month - 1) \* 31 + day,

Month = month, Day = day, ConfirmedCases = iter[1, ],

Fatalities = iter[2, ])

if(iter[1, ] != 0) total <- rbind(total, iter2)

}

}

*#create plot of cummulative confirmed cases*

plot(total$Num, total$ConfirmedCases,

type = "l", col = "blue",

xlab = "Day of (since January 22, 2020)",

ylab = "Number of Person",

main = "Cummulative Worldwide Confirmed Cases and Fatalities of Covid-19",

sub = "")

par(new = TRUE)

*#create plot of cummulative*

plot(total$Num, total$Fatalities,

type = "l", lty = 2, col = "red", xaxt = "n", yaxt = "n", xlab = "", ylab = "")

axis(side = 4)

legend("topleft", inset = .05,

legend = c("Confirmed Cases [Left axis]", "Fatalities [Right axis]"),

col = c("blue", "red"), bg = "gray",

lty = c(1, 2))

## Chart, line chart Description automatically generated

## By Visualizing the Cases and Fatalities Actual Data vs Model.

#set area

area <- "Indonesia-"

*#retrieve the data*

data <- as.data.frame(df2[[area]])

*#create plot*

plot(data$Day, data$ConfirmedCases,

type = "l", lty = 2,

col = "blue",

ylim = c(0, max(data$ConfirmedCases)),

xlab = "Day of (Since January 22, 2020)", ylab = "Number of People",

main = paste("Covid-19 Confirmed Cases in", area),

sub = "")

par(new = TRUE)

plot(data$Day, fitted(confirmed.cases.model[[area]]),

type = "l", lty = 3,

ylim = c(0, max(data$ConfirmedCases)),

col = "red",

xlab = "", ylab = "")

par(new = TRUE)

plot(data$Day, data$Fatalities,

type = "l", lty = 3,

col = "green",

ylim = c(0, max(data$Fatalities)),

xlab = "", ylab = "", xaxt = "n", yaxt = "n")

par(new = TRUE)

plot(data$Day, fitted(fatalities.model[[area]]),

type = "l", lty = 4,

ylim = c(0, max(data$Fatalities)),

col = "black",

xlab = "", ylab = "", xaxt = "n", yaxt = "n")

par(new = TRUE)

axis(side = 4)

legend("topleft", inset = .05,

legend = c("Confirmed Cases [Left Figure]",

"Estimated Cases (based on Model) [Left Figure]",

"Confirmed Fatality [Right Figure]",

"Estimated Fatality (based on Model) [Right Figure]"),

col = c("blue", "red", "green", "black"), bg = "gray",

lty = c(2, 3, 4, 5),

cex = 0.75)

**OUTPUT:**

Chart

Description automatically generated

print(paste("Worldwide Accuracy of Cases: ", mean(accuracy$AccuracyCases)))

print(paste("Worldwide Accuracy of Fatalities: ", mean(accuracy$AccuracyFatalities)))

**OUTPUT:**

[1] "Worldwide Accuracy of Cases: 0.915100576667328"

[1] "Worldwide Accuracy of Fatalities: 0.918521321360792"

**PART B: DATA VISUALIZATION**

library(tidyverse)

mpg

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy))

Graphical user interface

Description automatically generated with medium confidence

**Creating a ggplot**

Chart, scatter chart

Description automatically generated

**Aesthetic Mappings**

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, color = class))

Chart, scatter chart

Description automatically generated

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, size = class))

Chart, scatter chart

Description automatically generated

**# Top**

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, alpha = class))

Chart, scatter chart

Description automatically generated

**# Bottom**

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, shape = class))

Chart, scatter chart

Description automatically generated

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy), color = "blue")

Chart, scatter chart

Description automatically generated

**Facets**

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy)) + facet\_wrap(~ class, nrow = 2)

Chart, scatter chart

Description automatically generated

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy)) + facet\_grid(drv ~ cyl)

Chart, scatter chart

Description automatically generated

**Geometric Objects**

**# left**

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy))

Chart, scatter chart

Description automatically generated

**# right**

# ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy))

Chart, line chart, histogram

Description automatically generated

# ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))

Chart

Description automatically generated

# ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy))

Chart, line chart, histogram

Description automatically generated

ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy, group = drv))

Chart

Description automatically generated

# ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy)) + geom\_smooth(mapping = aes(x =

# displ, y = hwy))

Chart, scatter chart

Description automatically generated

# ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) + geom\_point() + geom\_smooth()

Chart, scatter chart

Description automatically generated

# ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) + geom\_point(mapping = aes(color = class)) +

# geom\_smooth()

Chart, scatter chart

Description automatically generated

# ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) + geom\_point(mapping = aes(color = class)) +

# geom\_smooth( data = filter(mpg, class == "subcompact"), se = FALSE )

Chart, scatter chart

Description automatically generated

**PRACTICAL 04**

## Data Import with readr

### Introduction

Working with data provided by R packages is a great way to learn the tools of data science, but at some point you want to stop learning and start working with your own data. In this chapter, you’ll learn how to read plain-text rectangular files into R. Here, we’ll only scratch the surface of data import, but many of the principles will translate to other forms of data. We’ll finish with a few pointers to packages that are useful for other types of data.

#### Prerequisites

In this chapter, you’ll learn how to load flat files in R with the **readr**

package, which is part of the core tidyverse.

library(tidyverse)

### Getting Started

Most of **readr**’s functions are concerned with turning flat files into data frames:

* read\_csv() reads comma-delimited files, read\_csv2() reads semicolon-separated files (common in countries where , is used as the decimal place), read\_tsv() reads tab-delimited files, and read\_delim() reads in files with any delimiter.
* read\_fwf() reads fixed-width files. You can specify fields either by their widths with fwf\_widths() or their position with fwf\_positions(). read\_table() reads a common variation of fixed-width files where columns are separated by white space.
* read\_log() reads Apache style log files. (But also check out [**webreadr**](https://github.com/Ironholds/webreadr), which is built on top of read\_log() and provides many more helpful tools.)

These functions all have similar syntax: once you’ve mastered one, you can use the others with ease. For the rest of this chapter we’ll focus on read\_csv(). Not only are CSV files one of the most com‐ mon forms of data storage, but once you understand read\_csv(), you can easily apply your knowledge to all the other functions in **readr**.

The first argument to read\_csv() is the most important; it’s the path to the file to read:

heights <- read\_csv("data/heights.csv")

*#> Parsed with column specification:*

*#> cols(*

*#> earn = col\_double(), #> height = col\_double(), #> sex = col\_character(), #> ed = col\_integer(),*

*#> age = col\_integer(), #> race = col\_character() #> )*

When you run read\_csv() it prints out a column specification that gives the name and type of each column. That’s an important part of **readr**, which we’ll come back to in [“Parsing a File” on page 137](#_bookmark274).

You can also supply an inline CSV file. This is useful for experi‐ menting with **readr** and for creating reproducible examples to share with others:

read\_csv("a,b,c 1,2,3

4,5,6")

*#> # A tibble: 2 × 3 #> a b c #> <int> <int> <int>*

|  |  |  |  |
| --- | --- | --- | --- |
| *#> 1* | *1* | *2* | *3* |
| *#> 2* | *4* | *5* | *6* |

In both cases read\_csv() uses the first line of the data for the col‐ umn names, which is a very common convention. There are two cases where you might want to tweak this behavior:

* Sometimes there are a few lines of metadata at the top of the file. You can use skip = n to skip the first n lines; or use comment = "#" to drop all lines that start with (e.g.) #:

read\_csv("The first line of metadata The second line of metadata

x,y,z

1,2,3", skip = 2)

*#> # A tibble: 1 × 3 #> x y z #> <int> <int> <int> #> 1 1 2 3*

read\_csv("# A comment I want to skip x,y,z

1,2,3", comment = "#")

*#> # A tibble: 1 × 3 #> x y z #> <int> <int> <int> #> 1 1 2 3*

* The data might not have column names. You can use col\_names

= FALSE to tell read\_csv() not to treat the first row as headings, and instead label them sequentially from X1 to Xn:

read\_csv("1,2,3\n4,5,6", col\_names = **FALSE**)

*#> # A tibble: 2 × 3 #> X1 X2 X3*

*#> <int> <int> <int>*

|  |  |  |  |
| --- | --- | --- | --- |
| *#> 1* | *1* | *2* | *3* |
| *#> 2* | *4* | *5* | *6* |

("\n" is a convenient shortcut for adding a new line. You’ll learn more about it and other types of string escape in [“String Basics”](#_bookmark371) [on page 195](#_bookmark371).)

Alternatively you can pass col\_names a character vector, which will be used as the column names:

read\_csv("1,2,3\n4,5,6", col\_names = c("x", "y", "z"))

*#> # A tibble: 2 × 3 #> x y z #> <int> <int> <int>*

|  |  |  |  |
| --- | --- | --- | --- |
| *#> 1* | *1* | *2* | *3* |
| *#> 2* | *4* | *5* | *6* |

Another option that commonly needs tweaking is na. This specifies the value (or values) that are used to represent missing values in your file:

read\_csv("a,b,c\n1,2,.", na = ".")

*#> # A tibble: 1 × 3 #> a b c #> <int> <int> <chr> #> 1 1 2 <NA>*

This is all you need to know to read ~75% of CSV files that you’ll encounter in practice. You can also easily adapt what you’ve learned to read tab-separated files with read\_tsv() and fixed-width files with read\_fwf(). To read in more challenging files, you’ll need to learn more about how **readr** parses each column, turning them into R vectors.

#### Compared to Base R

If you’ve used R before, you might wonder why we’re not using read.csv(). There are a few good reasons to favor **readr** functions over the base equivalents:

* They are typically much faster (~10x) than their base equiva‐ lents. Long-running jobs have a progress bar, so you can see what’s happening. If you’re looking for raw speed, try data.table::fread(). It doesn’t fit quite so well into the tidy‐ verse, but it can be quite a bit faster.
* They produce tibbles, and they don’t convert character vectors to factors, use row names, or munge the column names. These are common sources of frustration with the base R functions.
* They are more reproducible. Base R functions inherit some behavior from your operating system and environment vari‐ ables, so import code that works on your computer might not work on someone else’s.

#### Exercises

1. What function would you use to read a file where fields are sep‐ arated with “|”?
2. Apart from file, skip, and comment, what other arguments do

read\_csv() and read\_tsv() have in common?

1. What are the most important arguments to read\_fwf()?
2. Sometimes strings in a CSV file contain commas. To prevent them from causing problems they need to be surrounded by a quoting character, like " or '. By convention, read\_csv() assumes that the quoting character will be ", and if you want to change it you’ll need to use read\_delim() instead. What argu‐ ments do you need to specify to read the following text into a data frame?

"x,y\n1,'a,b'"

1. Identify what is wrong with each of the following inline CSV files. What happens when you run the code?

read\_csv("a,b\n1,2,3\n4,5,6")

read\_csv("a,b,c\n1,2\n1,2,3,4") read\_csv("a,b\n\"1") read\_csv("a,b\n1,2\na,b") read\_csv("a;b\n1;3")

### Parsing a Vector

Before we get into the details of how **readr** reads files from disk, we need to take a little detour to talk about the parse\_\*() functions. These functions take a character vector and return a more special‐ ized vector like a logical, integer, or date:

str(parse\_logical(c("TRUE", "FALSE", "NA")))

*#> logi [1:3] TRUE FALSE NA*

str(parse\_integer(c("1", "2", "3")))

*#> int [1:3] 1 2 3*

str(parse\_date(c("2010-01-01", "1979-10-14")))

*#> Date[1:2], format: "2010-01-01" "1979-10-14"*

These functions are useful in their own right, but are also an impor‐ tant building block for **readr**. Once you’ve learned how the individ‐ ual parsers work in this section, we’ll circle back and see how they fit together to parse a complete file in the next section.

Like all functions in the tidyverse, the parse\_\*() functions are uni‐ form; the first argument is a character vector to parse, and the na argument specifies which strings should be treated as missing:

parse\_integer(c("1", "231", ".", "456"), na = ".")

*#> [1] 1 231 NA 456*

If parsing fails, you’ll get a warning:

x <- parse\_integer(c("123", "345", "abc", "123.45"))

*#> Warning: 2 parsing failures.*

*#> row col expected actual #> 3 -- an integer abc #> 4 -- no trailing characters .45*

And the failures will be missing in the output:

x

*#> [1] 123 345 NA NA*

*#> attr(,"problems") #> # A tibble: 2 × 4*

*#> row col expected actual*

*#> <int> <int> <chr> <chr>*

*#> 1 3 NA an integer abc #> 2 4 NA no trailing characters .45*

If there are many parsing failures, you’ll need to use problems() to get the complete set. This returns a tibble, which you can then manipulate with **dplyr**:

problems(x)

*#> # A tibble: 2 × 4*

*#> row col expected actual*

*#> <int> <int> <chr> <chr>*

*#> 1 3 NA an integer abc #> 2 4 NA no trailing characters .45*

Using parsers is mostly a matter of understanding what’s available and how they deal with different types of input. There are eight par‐ ticularly important parsers:

* parse\_logical() and parse\_integer() parse logicals and inte‐ gers, respectively. There’s basically nothing that can go wrong with these parsers so I won’t describe them here further.
* parse\_double() is a strict numeric parser, and parse\_number() is a flexible numeric parser. These are more complicated than you might expect because different parts of the world write numbers in different ways.
* parse\_character() seems so simple that it shouldn’t be neces‐ sary. But one complication makes it quite important: character encodings.
* parse\_factor() creates factors, the data structure that R uses to represent categorical variables with fixed and known values.
* parse\_datetime(), parse\_date(), and parse\_time() allow you to parse various date and time specifications. These are the most complicated because there are so many different ways of writing dates.

The following sections describe these parsers in more detail.

#### Numbers

It seems like it should be straightforward to parse a number, but three problems make it tricky:

* People write numbers differently in different parts of the world. For example, some countries use . in between the integer and fractional parts of a real number, while others use ,.
* Numbers are often surrounded by other characters that provide some context, like “$1000” or “10%”.
* Numbers often contain “grouping” characters to make them easier to read, like “1,000,000”, and these grouping characters vary around the world.

To address the first problem, **readr** has the notion of a “locale,” an object that specifies parsing options that differ from place to place. When parsing numbers, the most important option is the character you use for the decimal mark. You can override the default value of . by creating a new locale and setting the decimal\_mark argu‐ ment:

parse\_double("1.23")

*#> [1] 1.23*

parse\_double("1,23", locale = locale(decimal\_mark = ","))

*#> [1] 1.23*

**readr**’s default locale is US-centric, because generally R is US-centric (i.e., the documentation of base R is written in American English). An alternative approach would be to try and guess the defaults from your operating system. This is hard to do well, and, more impor‐ tantly, makes your code fragile: even if it works on your computer, it might fail when you email it to a colleague in another country.

parse\_number() addresses the second problem: it ignores non- numeric characters before and after the number. This is particularly useful for currencies and percentages, but also works to extract numbers embedded in text:

parse\_number("$100")

*#> [1] 100*

parse\_number("20%")

*#> [1] 20*

parse\_number("It cost $123.45")

*#> [1] 123*

The final problem is addressed by the combination of parse\_num ber() and the locale as parse\_number() will ignore the “grouping mark”:

*# Used in America* parse\_number("$123,456,789") *#> [1] 1.23e+08*

*# Used in many parts of Europe*

parse\_number( "123.456.789",

locale = locale(grouping\_mark = ".")

)

*#> [1] 1.23e+08*

*# Used in Switzerland*

parse\_number( "123'456'789",

locale = locale(grouping\_mark = "'")

)

*#> [1] 1.23e+08*

#### Strings

It seems like parse\_character() should be really simple—it could just return its input. Unfortunately life isn’t so simple, as there are multiple ways to represent the same string. To understand what’s going on, we need to dive into the details of how computers repre‐ sent strings. In R, we can get at the underlying representation of a string using charToRaw():

charToRaw("Hadley")

*#> [1] 48 61 64 6c 65 79*

Each hexadecimal number represents a byte of information: 48 is H, 61 is a, and so on. The mapping from hexadecimal number to char‐ acter is called the encoding, and in this case the encoding is called

ASCII. ASCII does a great job of representing English characters, because it’s the *American* Standard Code for Information Inter‐ change.

Things get more complicated for languages other than English. In the early days of computing there were many competing standards for encoding non-English characters, and to correctly interpret a string you needed to know both the values and the encoding. For example, two common encodings are Latin1 (aka ISO-8859-1, used for Western European languages) and Latin2 (aka ISO-8859-2, used for Eastern European languages). In Latin1, the byte b1 is “±”, but in Latin2, it’s “ą”! Fortunately, today there is one standard that is sup‐ ported almost everywhere: UTF-8. UTF-8 can encode just about every character used by humans today, as well as many extra sym‐ bols (like emoji!).

**readr** uses UTF-8 everywhere: it assumes your data is UTF-8 enco‐ ded when you read it, and always uses it when writing. This is a good default, but will fail for data produced by older systems that don’t understand UTF-8. If this happens to you, your strings will look weird when you print them. Sometimes just one or two charac‐ ters might be messed up; other times you’ll get complete gibberish. For example:

x1 <- "El Ni\xf1o was particularly bad this year" x2 <- "\x82\xb1\x82\xf1\x82\xc9\x82\xbf\x82\xcd"

To fix the problem you need to specify the encoding in parse\_char acter():

parse\_character(x1, locale = locale(encoding = "Latin1"))

*#> [1] "El Niño was particularly bad this year"* parse\_character(x2, locale = locale(encoding = "Shift-JIS")) *#> [1] "*こんにちは*"*

How do you find the correct encoding? If you’re lucky, it’ll be included somewhere in the data documentation. Unfortunately, that’s rarely the case, so **readr** provides guess\_encoding() to help you figure it out. It’s not foolproof, and it works better when you have lots of text (unlike here), but it’s a reasonable place to start. Expect to try a few different encodings before you find the right one:

guess\_encoding(charToRaw(x1))

*#> encoding confidence #> 1 ISO-8859-1 0.46*

*#> 2 ISO-8859-9 0.23*

guess\_encoding(charToRaw(x2))

*#> encoding confidence #> 1 KOI8-R 0.42*

The first argument to guess\_encoding() can either be a path to a file, or, as in this case, a raw vector (useful if the strings are already in R).

Encodings are a rich and complex topic, and I’ve only scratched the surface here. If you’d like to learn more I’d recommend reading the detailed explanation at [*http://kunststube.net/encoding/*](http://kunststube.net/encoding/).

#### Factors

R uses factors to represent categorical variables that have a known set of possible values. Give parse\_factor() a vector of known levels to generate a warning whenever an unexpected value is present:

fruit <- c("apple", "banana")

parse\_factor(c("apple", "banana", "bananana"), levels = fruit)

*#> Warning: 1 parsing failure.*

*#> row col expected actual #> 3 -- value in level set bananana #> [1] apple banana <NA>*

*#> attr(,"problems") #> # A tibble: 1 × 4*

*#> row col expected actual*

*#> <int> <int> <chr> <chr> #> 1 3 NA value in level set bananana #> Levels: apple banana*

But if you have many problematic entries, it’s often easier to leave them as character vectors and then use the tools you’ll learn about in [Chapter 11](#_bookmark366) and [Chapter 12](#_bookmark413) to clean them up.

#### Dates, Date-Times, and Times

You pick between three parsers depending on whether you want a date (the number of days since 1970-01-01), a date-time (the num‐ ber of seconds since midnight 1970-01-01), or a time (the number of seconds since midnight). When called without any additional argu‐ ments:

* parse\_datetime() expects an ISO8601 date-time. ISO8601 is an international standard in which the components of a date are organized from biggest to smallest: year, month, day, hour, minute, second:

parse\_datetime("2010-10-01T2010")

*#> [1] "2010-10-01 20:10:00 UTC"*

*# If time is omitted, it will be set to midnight*

parse\_datetime("20101010")

*#> [1] "2010-10-10 UTC"*

This is the most important date/time standard, and if you work with dates and times frequently, I recommend reading [*https://*](https://en.wikipedia.org/wiki/ISO_8601)[*en.wikipedia.org/wiki/ISO\_8601*](https://en.wikipedia.org/wiki/ISO_8601).

* parse\_date() expects a four-digit year, a - or /, the month, a -

or /, then the day:

parse\_date("2010-10-01")

*#> [1] "2010-10-01"*

* parse\_time() expects the hour, :, minutes, optionally : and seconds, and an optional a.m./p.m. specifier:

library(hms) parse\_time("01:10 am") *#> 01:10:00*

parse\_time("20:10:01")

*#> 20:10:01*

Base R doesn’t have a great built-in class for time data, so we use the one provided in the **hms** package.

If these defaults don’t work for your data you can supply your own date-time format, built up of the following pieces:

*Year*

%Y (4 digits).

%y (2 digits; 00-69 → 2000-2069, 70-99 → 1970-1999).

*Month*

%m (2 digits).

%b (abbreviated name, like “Jan”).

%B (full name, “January”).

*Day*

%d (2 digits).

%e (optional leading space).

*Time*

%H (0-23 hour format).

%I (0-12, must be used with %p).

%p (a.m./p.m. indicator).

%M (minutes).

%S (integer seconds).

%OS (real seconds).

%Z (time zone [a name, e.g., America/Chicago]). Note: beware of abbreviations. If you’re American, note that “EST” is a Cana‐ dian time zone that does not have daylight saving time. It is Eastern Standard Time! We’ll come back to this in [“Time](#_bookmark462) [Zones” on page 254](#_bookmark462).

%z (as offset from UTC, e.g., +0800).

*Nondigits*

%. (skips one nondigit character).

%\* (skips any number of nondigits).

The best way to figure out the correct format is to create a few examples in a character vector, and test with one of the parsing functions. For example:

parse\_date("01/02/15", "%m/%d/%y")

*#> [1] "2015-01-02"*

parse\_date("01/02/15", "%d/%m/%y")

*#> [1] "2015-02-01"*

parse\_date("01/02/15", "%y/%m/%d")

*#> [1] "2001-02-15"*

If you’re using %b or %B with non-English month names, you’ll need to set the lang argument to locale(). See the list of built-in lan‐ guages in date\_names\_langs(), or if your language is not already included, create your own with date\_names():

parse\_date("1 janvier 2015", "%d %B %Y", locale = locale("fr"))

*#> [1] "2015-01-01"*

**PRACTICAL 05**

**Part A : Data Modeling**

#set area

area <- "Indonesia-"

*#retrieve the data*

data <- as.data.frame(df2[[area]])

*#create plot*

plot(data$Day, data$ConfirmedCases,

type = "l", lty = 2,

col = "blue",

ylim = c(0, max(data$ConfirmedCases)),

xlab = "Day of (Since January 22, 2020)", ylab = "Number of People",

main = paste("Covid-19 Confirmed Cases in", area),

sub = "")

par(new = TRUE)

plot(data$Day, fitted(confirmed.cases.model[[area]]),

type = "l", lty = 3,

ylim = c(0, max(data$ConfirmedCases)),

col = "red",

xlab = "", ylab = "")

par(new = TRUE)

plot(data$Day, data$Fatalities,

type = "l", lty = 3,

col = "green",

ylim = c(0, max(data$Fatalities)),

xlab = "", ylab = "", xaxt = "n", yaxt = "n")

par(new = TRUE)

plot(data$Day, fitted(fatalities.model[[area]]),

type = "l", lty = 4,

ylim = c(0, max(data$Fatalities)),

col = "black",

xlab = "", ylab = "", xaxt = "n", yaxt = "n")

par(new = TRUE)

axis(side = 4)

legend("topleft", inset = .05,

legend = c("Confirmed Cases [Left Figure]",

"Estimated Cases (based on Model) [Left Figure]",

"Confirmed Fatality [Right Figure]",

"Estimated Fatality (based on Model) [Right Figure]"),

col = c("blue", "red", "green", "black"), bg = "gray",

lty = c(2, 3, 4, 5),

cex = 0.75)

**Chart

Description automatically generated**

**Part B : Data Modeling**

# library(tidyverse)

# library(modelr)

# options(na.action = na.warn)

# df <- tribble( ~y, ~x1, ~x2, 4, 2, 5, 5, 1, 6 )

# model\_matrix(df, y ~ x1)

**OUTPUT:**

> df <- tribble( ~y, ~x1, ~x2, 4, 2, 5, 5, 1, 6 )

> model\_matrix(df, y ~ x1)

# A tibble: 2 x 2

`(Intercept)` x1

<dbl> <dbl>

1 1 2

2 1 1

## model\_matrix(df, y ~ x1 - 1)

**OUTPUT:**

model\_matrix(df, y ~ x1 - 1)

# A tibble: 2 x 1

x1

<dbl>

1 2

2 1

## model\_matrix(df, y ~ x1 + x2)

**OUTPUT:**

model\_matrix(df, y ~ x1 + x2)

# A tibble: 2 x 3

`(Intercept)` x1 x2

<dbl> <dbl> <dbl>

1 1 2 5

2 1 1 6

## df <- tribble( ~ sex, ~ response, "male", 1, "female", 2, "male", 1 ) model\_matrix(df, response ~ sex)

**OUTPUT:**

df <- tribble( ~ sex, ~ response, "male", 1, "female", 2, "male", 1 )

> model\_matrix(df, response ~ sex)

# A tibble: 3 x 2

`(Intercept)` sexmale

<dbl> <dbl>

1 1 1

2 1 0

3 1 1

## ggplot(sim2) + geom\_point(aes(x, y))

**OUTPUT:**

Chart, bubble chart

Description automatically generated

## mod2 <- lm(y ~ x, data = sim2)

## grid <- sim2 %>% data\_grid(x) %>% add\_predictions(mod2)

## grid

**OUTPUT:**

mod2 <- lm(y ~ x, data = sim2)

> grid <- sim2 %>% data\_grid(x) %>% add\_predictions(mod2)

> grid

# A tibble: 4 x 2

x pred

<chr> <dbl>

1 a 1.15

2 b 8.12

3 c 6.13

4 d 1.91

## ggplot(sim2, aes(x)) +

## geom\_point(aes(y = y)) +

## geom\_point( data = grid, aes(y = pred), color = "red", size = 4 )

**OUTPUT:**

Chart

Description automatically generated

**PRACTICAL 06**

**Linear Regression**

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

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

The general mathematical equation for a linear regression is −

y = ax + b

Following is the description of the parameters used −

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

Steps to Establish a Regression

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

The steps to create the relationship is −

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

Input Data

Below is the sample data representing the observations −

# Values of height

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

# Values of weight.

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

lm() Function

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

Syntax

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

lm(formula,data)

Following is the description of the parameters used −

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

Create Relationship Model & get the Coefficients

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

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

# Apply the lm() function.

relation <- lm(y~x)

print(relation)

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

Call:

lm(formula = y ~ x)

Coefficients:

(Intercept) x

-38.4551 0.6746

Get the Summary of the Relationship

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

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

# Apply the lm() function.

relation <- lm(y~x)

print(summary(relation))

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

Call:

lm(formula = y ~ x)

Residuals:

Min 1Q Median 3Q Max

-6.3002 -1.6629 0.0412 1.8944 3.9775

Coefficients:

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

(Intercept) -38.45509 8.04901 -4.778 0.00139 \*\*

x 0.67461 0.05191 12.997 1.16e-06 \*\*\*

---

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

Residual standard error: 3.253 on 8 degrees of freedom

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

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

predict() Function

Syntax

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

predict(object, newdata)

Following is the description of the parameters used −

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

Predict the weight of new persons

# The predictor vector.

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

# The resposne vector.

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

# Apply the lm() function.

relation <- lm(y~x)

# Find weight of a person with height 170.

a <- data.frame(x = 170)

result <- predict(relation,a)

print(result)

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

1

76.22869

Visualize the Regression Graphically

# Create the predictor and response variable.

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

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

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

# Plot the chart.

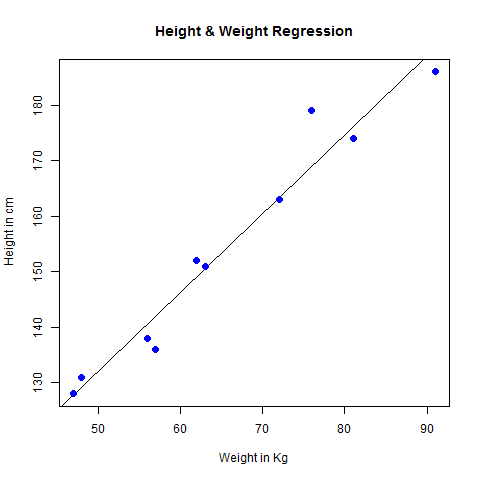
plot(y,x,col = "blue",main = "Height & Weight Regression",

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

# Save the file.

dev.off()

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



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.

The general mathematical equation for multiple regression is −

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

Following is the description of the parameters used −

* **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

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

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

Following is the description of the parameters used −

* **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

or

Y = 37.15+(-0.000937)\*x1+(-0.0311)\*x2+(-3.8008)\*x3

Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horse power and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is −

Y = 37.15+(-0.000937)\*221+(-0.0311)\*102+(-3.8008)\*2.91 = 22.7104

Logistic Regression:

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −

y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))

Following is the description of the parameters used −

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

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

Syntax

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

glm(formula,data,family)

Following is the description of the parameters used −

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

Example

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

# Select some columns form mtcars.

input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))

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

am cyl hp wt

Mazda RX4 1 6 110 2.620

Mazda RX4 Wag 1 6 110 2.875

Datsun 710 1 4 93 2.320

Hornet 4 Drive 0 6 110 3.215

Hornet Sportabout 0 8 175 3.440

Valiant 0 6 105 3.460

Create Regression Model

We use the **glm()** function to create the regression model and get its summary for analysis.

input <- mtcars[,c("am","cyl","hp","wt")]

am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)

print(summary(am.data))

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

Call:

glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:

Min 1Q Median 3Q Max

-2.17272 -0.14907 -0.01464 0.14116 1.27641

Coefficients:

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

(Intercept) 19.70288 8.11637 2.428 0.0152 \*

cyl 0.48760 1.07162 0.455 0.6491

hp 0.03259 0.01886 1.728 0.0840 .

wt -9.14947 4.15332 -2.203 0.0276 \*

---

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

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 43.2297 on 31 degrees of freedom

Residual deviance: 9.8415 on 28 degrees of freedom

AIC: 17.841

Number of Fisher Scoring iterations: 8

Conclusion

In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

Normal distribution

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

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

dnorm(x, mean, sd)

pnorm(x, mean, sd)

qnorm(p, mean, sd)

rnorm(n, mean, sd)

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

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

dnorm()

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

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

x <- seq(-10, 10, by = .1)

# Choose the mean as 2.5 and standard deviation as 0.5.

y <- dnorm(x, mean = 2.5, sd = 0.5)

# Give the chart file a name.

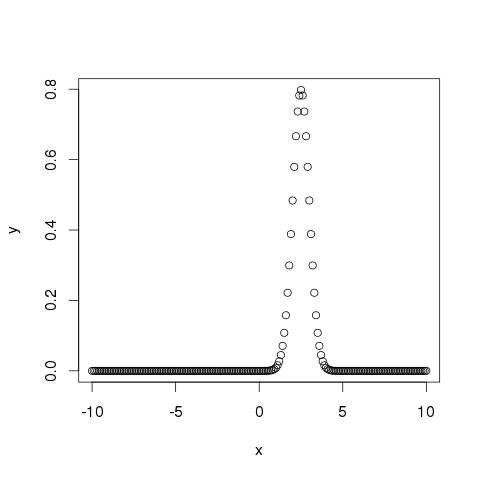
png(file = "dnorm.png")

plot(x,y)

# Save the file.

dev.off()

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



pnorm()

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

# Create a sequence of numbers between -10 and 10 incrementing by 0.2.

x <- seq(-10,10,by = .2)

# Choose the mean as 2.5 and standard deviation as 2.

y <- pnorm(x, mean = 2.5, sd = 2)

# Give the chart file a name.

png(file = "pnorm.png")

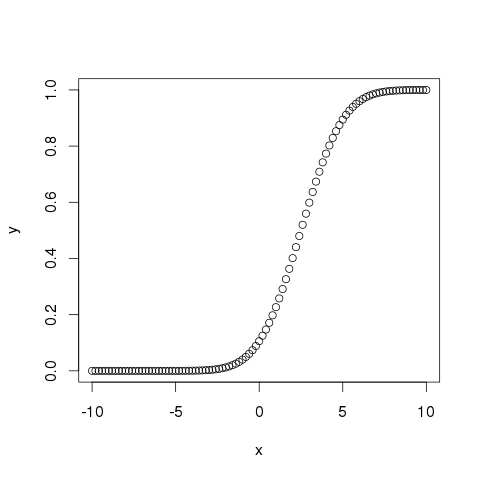
# Plot the graph.

plot(x,y)

# Save the file.

dev.off()

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



qnorm()

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

# Create a sequence of probability values incrementing by 0.02.

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

# Choose the mean as 2 and standard deviation as 3.

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

# Give the chart file a name.

png(file = "qnorm.png")

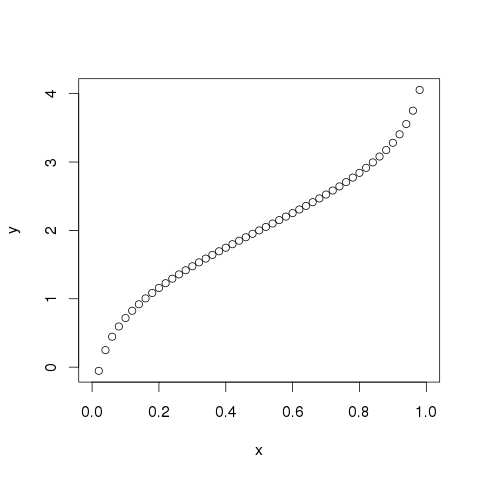
# Plot the graph.

plot(x,y)

# Save the file.

dev.off()

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



rnorm()

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

# Create a sample of 50 numbers which are normally distributed.

y <- rnorm(50)

# Give the chart file a name.

png(file = "rnorm.png")

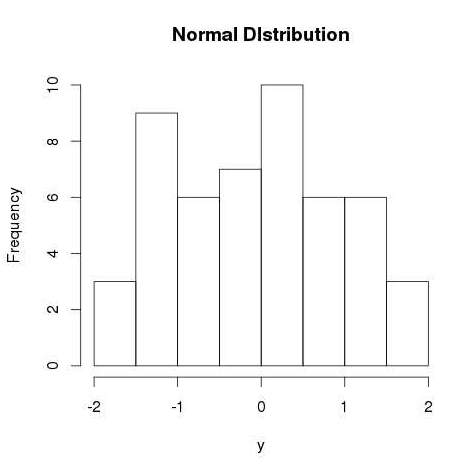
# Plot the histogram for this sample.

hist(y, main = "Normal DIstribution")

# Save the file.

dev.off()

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



Bionomial distribution:

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

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

dbinom(x, size, prob)

pbinom(x, size, prob)

qbinom(p, size, prob)

rbinom(n, size, prob)

Following is the description of the parameters used −

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

## dbinom()

This function gives the probability density distribution at each point.

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

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

# Create the binomial distribution.

y <- dbinom(x,50,0.5)

# Give the chart file a name.

png(file = "dbinom.png")

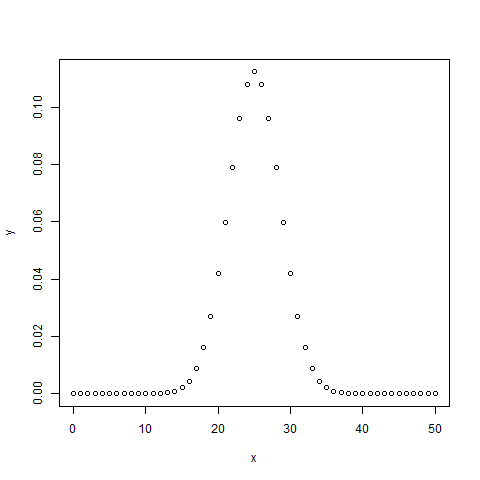
# Plot the graph for this sample.

plot(x,y)

# Save the file.

dev.off()

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



## pbinom()

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

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

x <- pbinom(26,51,0.5)

print(x)

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

[1] 0.610116

## qbinom()

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

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

# is tossed 51 times.

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

print(x)

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

[1] 23

## rbinom()

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

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

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

print(x)

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

[1] 58 61 59 66 55 60 61 67

Poisson Regression

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

The general mathematical equation for Poisson regression is −

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

Following is the description of the parameters used −

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

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

### Syntax

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

glm(formula,data,family)

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

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

### Example

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

**Input Data**

input <- warpbreaks

print(head(input))

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

breaks wool tension

1 26 A L

2 30 A L

3 54 A L

4 25 A L

5 70 A L

6 52 A L

## Create Regression Model

output <-glm(formula = breaks ~ wool+tension, data = warpbreaks,

family = poisson)

print(summary(output))

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

Call:

glm(formula = breaks ~ wool + tension, family = poisson, data = warpbreaks)

Deviance Residuals:

Min 1Q Median 3Q Max

-3.6871 -1.6503 -0.4269 1.1902 4.2616

Coefficients:

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

(Intercept) 3.69196 0.04541 81.302 < 2e-16 \*\*\*

woolB -0.20599 0.05157 -3.994 6.49e-05 \*\*\*

tensionM -0.32132 0.06027 -5.332 9.73e-08 \*\*\*

tensionH -0.51849 0.06396 -8.107 5.21e-16 \*\*\*

---

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

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 297.37 on 53 degrees of freedom

Residual deviance: 210.39 on 50 degrees of freedom

AIC: 493.06

Number of Fisher Scoring iterations: 4

In the summary we look for the p-value in the last column to be less than 0.05 to consider an impact of the predictor variable on the response variable. As seen the wool type B having tension type M and H have impact on the count of breaks.

**PRACTICAL 07**

Practical of Principal Component Analysis

**Principal Component Analysis**:- PCA can be considered as a rotation of the axes of the original variable coordinate system to new orthogonal axes, called as the principal axes such that new axes coincide with directions of maximum variation of the original observations. PCA is also called as Karhunen-Loeve of K-L Method. This method of dimensionality reduction searches for k n-dimensional orthogonal vectors that can best used to represent the data. The original data are thus projected onto a much smaller space which results to dimensionally reduction.

**Code:-**

**df <- read.csv('train.csv',encoding = "ISO-8859-1")**

**head(df)**

**summary(df)**

**myPrc<-prcomp(df[5:6],scale=TRUE)**

**plot(df$ConfirmedCases,df$Fatalities)**

**plot(scale(df$ConfirmedCase),scale(df$Fatalities))**

**myPrc**

**summary(myPrc)**

**plot(myPrc,scale='1')**

**biplot(myPrc,scale=0)**

**str(myPrc)**

**myPrc$x**

**df2<-cbind(df,myPrc$x[,1:2])**

**head(df2)**

**library(ggplot2)**

**ggplot(data = df2) + geom\_point(mapping = aes(x = PC1, y = PC2))**

**stat\_ellipse(geom="polygon",col="red",alpha=0.5)**

**geom\_point(shape=21,col="black")**

**cor(df[5:6],df[5:6])**

**Output :-**

|  |
| --- |
|  |

**df <- read.csv('train.csv',encoding = "ISO-8859-1")**

**> head(df)**

**Id Province\_State Country\_Region Date ConfirmedCases Fatalities**

**1 1 Afghanistan 2020-01-22 0 0**

**2 2 Afghanistan 2020-01-23 0 0**

**3 3 Afghanistan 2020-01-24 0 0**

**4 4 Afghanistan 2020-01-25 0 0**

**5 5 Afghanistan 2020-01-26 0 0**

**6 6 Afghanistan 2020-01-27 0 0**

**> summary(df)**

**Id Province\_State Country\_Region Date ConfirmedCases**

**Min. : 1 Length:20580 Length:20580 Length:20580 Min. : 0.0**

**1st Qu.: 7336 Class :character Class :character Class :character 1st Qu.: 0.0**

**Median :14686 Mode :character Mode :character Mode :character Median : 0.0**

**Mean :14686 Mean : 514.9**

**3rd Qu.:22035 3rd Qu.: 35.0**

**Max. :29370 Max. :105792.0**

**Fatalities**

**Min. : 0.00**

**1st Qu.: 0.00**

**Median : 0.00**

**Mean : 21.08**

**3rd Qu.: 0.00**

**Max. :12428.00**

**> myPrc<-prcomp(df[5:6],scale=TRUE)**

**> plot(df$ConfirmedCases,df$Fatalities)**

**> plot(scale(df$ConfirmedCase),scale(df$Fatalities))**

**> myPrc**

**Standard deviations (1, .., p=2):**

**[1] 1.3689540 0.3549153**

**Rotation (n x k) = (2 x 2):**

**PC1 PC2**

**ConfirmedCases 0.7071068 0.7071068**

**Fatalities 0.7071068 -0.7071068**

**> summary(myPrc)**

**Importance of components:**

**PC1 PC2**

**Standard deviation 1.369 0.35492**

**Proportion of Variance 0.937 0.06298**

**Cumulative Proportion 0.937 1.00000**

**> plot(myPrc,scale='1')**

**Warning messages:**

**1: In plot.window(xlim, ylim, log = log, ...) :**

**"scale" is not a graphical parameter**

**2: In title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...) :**

**"scale" is not a graphical parameter**

**3: In axis(if (horiz) 1 else 2, cex.axis = cex.axis, ...) :**

**"scale" is not a graphical parameter**

**> biplot(myPrc,scale=0)**

**Warning messages:**

**1: In doTryCatch(return(expr), name, parentenv, handler) :**

**"scale" is not a graphical parameter**

**2: In doTryCatch(return(expr), name, parentenv, handler) :**

**"scale" is not a graphical parameter**

**3: In doTryCatch(return(expr), name, parentenv, handler) :**

**"scale" is not a graphical parameter**

**4: In doTryCatch(return(expr), name, parentenv, handler) :**

**"scale" is not a graphical parameter**

**5: In doTryCatch(return(expr), name, parentenv, handler) :**

**"scale" is not a graphical parameter**

**6: In doTryCatch(return(expr), name, parentenv, handler) :**

**"scale" is not a graphical parameter**

**> str(myPrc)**

**List of 5**

**$ sdev : num [1:2] 1.369 0.355**

**$ rotation: num [1:2, 1:2] 0.707 0.707 0.707 -0.707**

**..- attr(\*, "dimnames")=List of 2**

**.. ..$ : chr [1:2] "ConfirmedCases" "Fatalities"**

**.. ..$ : chr [1:2] "PC1" "PC2"**

**$ center : Named num [1:2] 514.9 21.1**

**..- attr(\*, "names")= chr [1:2] "ConfirmedCases" "Fatalities"**

**$ scale : Named num [1:2] 4541 288**

**..- attr(\*, "names")= chr [1:2] "ConfirmedCases" "Fatalities"**

**$ x : num [1:20580, 1:2] -0.132 -0.132 -0.132 -0.132 -0.132 ...**

**..- attr(\*, "dimnames")=List of 2**

**.. ..$ : NULL**

**.. ..$ : chr [1:2] "PC1" "PC2"**

**- attr(\*, "class")= chr "prcomp"**

**> myPrc$x**

**PC1 PC2**

**[1,] -1.319818e-01 -2.837757e-02**

**[2,] -1.319818e-01 -2.837757e-02**

**[3,] -1.319818e-01 -2.837757e-02**

**[4,] -1.319818e-01 -2.837757e-02**

**[5,] -1.319818e-01 -2.837757e-02**

**[6,] -1.319818e-01 -2.837757e-02**

**[7,] -1.319818e-01 -2.837757e-02**

**[8,] -1.319818e-01 -2.837757e-02**

**[9,] -1.319818e-01 -2.837757e-02**

**[10,] -1.319818e-01 -2.837757e-02**

**[11,] -1.319818e-01 -2.837757e-02**

**[12,] -1.319818e-01 -2.837757e-02**

**[13,] -1.319818e-01 -2.837757e-02**

**[14,] -1.319818e-01 -2.837757e-02**

**[15,] -1.319818e-01 -2.837757e-02**

**[16,] -1.319818e-01 -2.837757e-02**

**[17,] -1.319818e-01 -2.837757e-02**

**[18,] -1.319818e-01 -2.837757e-02**

**[19,] -1.319818e-01 -2.837757e-02**

**[20,] -1.319818e-01 -2.837757e-02**

**[21,] -1.319818e-01 -2.837757e-02**

**[22,] -1.319818e-01 -2.837757e-02**

**[23,] -1.319818e-01 -2.837757e-02**

**[24,] -1.319818e-01 -2.837757e-02**

**[25,] -1.319818e-01 -2.837757e-02**

**[26,] -1.319818e-01 -2.837757e-02**

**[27,] -1.319818e-01 -2.837757e-02**

**[28,] -1.319818e-01 -2.837757e-02**

**[29,] -1.319818e-01 -2.837757e-02**

**[30,] -1.319818e-01 -2.837757e-02**

**[31,] -1.319818e-01 -2.837757e-02**

**[32,] -1.319818e-01 -2.837757e-02**

**[33,] -1.319818e-01 -2.837757e-02**

**[34,] -1.318261e-01 -2.822186e-02**

**[35,] -1.318261e-01 -2.822186e-02**

**[36,] -1.318261e-01 -2.822186e-02**

**[37,] -1.318261e-01 -2.822186e-02**

**[38,] -1.318261e-01 -2.822186e-02**

**[39,] -1.318261e-01 -2.822186e-02**

**[40,] -1.318261e-01 -2.822186e-02**

**[41,] -1.318261e-01 -2.822186e-02**

**[42,] -1.318261e-01 -2.822186e-02**

**[43,] -1.318261e-01 -2.822186e-02**

**[44,] -1.318261e-01 -2.822186e-02**

**[45,] -1.318261e-01 -2.822186e-02**

**[46,] -1.318261e-01 -2.822186e-02**

**[47,] -1.313590e-01 -2.775474e-02**

**[48,] -1.313590e-01 -2.775474e-02**

**[49,] -1.312033e-01 -2.759903e-02**

**[50,] -1.308919e-01 -2.728762e-02**

**[51,] -1.308919e-01 -2.728762e-02**

**[52,] -1.308919e-01 -2.728762e-02**

**[53,] -1.302690e-01 -2.666479e-02**

**[54,] -1.294905e-01 -2.588625e-02**

**[55,] -1.287120e-01 -2.510772e-02**

**[56,] -1.285562e-01 -2.495201e-02**

**[57,] -1.285562e-01 -2.495201e-02**

**[58,] -1.285562e-01 -2.495201e-02**

**[59,] -1.282448e-01 -2.464060e-02**

**[60,] -1.282448e-01 -2.464060e-02**

**[61,] -1.232962e-01 -2.460658e-02**

**[62,] -1.232962e-01 -2.460658e-02**

**[63,] -1.180022e-01 -1.931253e-02**

**[64,] -1.139878e-01 -2.021276e-02**

**[65,] -1.075162e-01 -2.357028e-02**

**[66,] -1.050248e-01 -2.107896e-02**

**[67,] -1.050248e-01 -2.107896e-02**

**[68,] -1.034678e-01 -1.952189e-02**

**[69,] -9.568241e-02 -1.173654e-02**

**[70,] -9.505959e-02 -1.111371e-02**

**[71,] -1.319818e-01 -2.837757e-02**

**[72,] -1.319818e-01 -2.837757e-02**

**[73,] -1.319818e-01 -2.837757e-02**

**[74,] -1.319818e-01 -2.837757e-02**

**[75,] -1.319818e-01 -2.837757e-02**

**[76,] -1.319818e-01 -2.837757e-02**

**[77,] -1.319818e-01 -2.837757e-02**

**[78,] -1.319818e-01 -2.837757e-02**

**[79,] -1.319818e-01 -2.837757e-02**

**[80,] -1.319818e-01 -2.837757e-02**

**[81,] -1.319818e-01 -2.837757e-02**

**[82,] -1.319818e-01 -2.837757e-02**

**[83,] -1.319818e-01 -2.837757e-02**

**[84,] -1.319818e-01 -2.837757e-02**

**[85,] -1.319818e-01 -2.837757e-02**

**[86,] -1.319818e-01 -2.837757e-02**

**[87,] -1.319818e-01 -2.837757e-02**

**[88,] -1.319818e-01 -2.837757e-02**

**[89,] -1.319818e-01 -2.837757e-02**

**[90,] -1.319818e-01 -2.837757e-02**

**[91,] -1.319818e-01 -2.837757e-02**

**[92,] -1.319818e-01 -2.837757e-02**

**[93,] -1.319818e-01 -2.837757e-02**

**[94,] -1.319818e-01 -2.837757e-02**

**[95,] -1.319818e-01 -2.837757e-02**

**[96,] -1.319818e-01 -2.837757e-02**

**[97,] -1.319818e-01 -2.837757e-02**

**[98,] -1.319818e-01 -2.837757e-02**

**[99,] -1.319818e-01 -2.837757e-02**

**[100,] -1.319818e-01 -2.837757e-02**

**[101,] -1.319818e-01 -2.837757e-02**

**[102,] -1.319818e-01 -2.837757e-02**

**[103,] -1.319818e-01 -2.837757e-02**

**[104,] -1.319818e-01 -2.837757e-02**

**[105,] -1.319818e-01 -2.837757e-02**

**[106,] -1.319818e-01 -2.837757e-02**

**[107,] -1.319818e-01 -2.837757e-02**

**[108,] -1.319818e-01 -2.837757e-02**

**[109,] -1.319818e-01 -2.837757e-02**

**[110,] -1.319818e-01 -2.837757e-02**

**[111,] -1.319818e-01 -2.837757e-02**

**[112,] -1.319818e-01 -2.837757e-02**

**[113,] -1.319818e-01 -2.837757e-02**

**[114,] -1.319818e-01 -2.837757e-02**

**[115,] -1.319818e-01 -2.837757e-02**

**[116,] -1.319818e-01 -2.837757e-02**

**[117,] -1.319818e-01 -2.837757e-02**

**[118,] -1.316704e-01 -2.806615e-02**

**[119,] -1.304247e-01 -2.682050e-02**

**[120,] -1.276560e-01 -2.896638e-02**

**[121,] -1.259432e-01 -2.725360e-02**

**[122,] -1.243862e-01 -2.569653e-02**

**[123,] -1.236076e-01 -2.491799e-02**

**[124,] -1.229848e-01 -2.429516e-02**

**[125,] -1.215834e-01 -2.289380e-02**

**[126,] -1.209606e-01 -2.227097e-02**

**[127,] -1.178805e-01 -2.410544e-02**

**[128,] -1.171020e-01 -2.332690e-02**

**[129,] -1.161677e-01 -2.239266e-02**

**[130,] -1.152335e-01 -2.145842e-02**

**[131,] -1.132093e-01 -1.943422e-02**

**[132,] -1.059591e-01 -2.201321e-02**

**[133,] -1.005434e-01 -2.151207e-02**

**[134,] -9.696209e-02 -1.793080e-02**

**[135,] -9.014500e-02 -1.602830e-02**

**[136,] -8.336192e-02 -1.907440e-02**

**[137,] -7.673455e-02 -2.227621e-02**

**[138,] -7.439895e-02 -1.994061e-02**

**[139,] -7.022887e-02 -2.068512e-02**

**[140,] -5.728555e-02 -2.740016e-02**

**[141,] -1.319818e-01 -2.837757e-02**

**[142,] -1.319818e-01 -2.837757e-02**

**[143,] -1.319818e-01 -2.837757e-02**

**[144,] -1.319818e-01 -2.837757e-02**

**[145,] -1.319818e-01 -2.837757e-02**

**[146,] -1.319818e-01 -2.837757e-02**

**[147,] -1.319818e-01 -2.837757e-02**

**[148,] -1.319818e-01 -2.837757e-02**

**[149,] -1.319818e-01 -2.837757e-02**

**[150,] -1.319818e-01 -2.837757e-02**

**[151,] -1.319818e-01 -2.837757e-02**

**[152,] -1.319818e-01 -2.837757e-02**

**[153,] -1.319818e-01 -2.837757e-02**

**[154,] -1.319818e-01 -2.837757e-02**

**[155,] -1.319818e-01 -2.837757e-02**

**[156,] -1.319818e-01 -2.837757e-02**

**[157,] -1.319818e-01 -2.837757e-02**

**[158,] -1.319818e-01 -2.837757e-02**

**[159,] -1.319818e-01 -2.837757e-02**

**[160,] -1.319818e-01 -2.837757e-02**

**[161,] -1.319818e-01 -2.837757e-02**

**[162,] -1.319818e-01 -2.837757e-02**

**[163,] -1.319818e-01 -2.837757e-02**

**[164,] -1.319818e-01 -2.837757e-02**

**[165,] -1.319818e-01 -2.837757e-02**

**[166,] -1.319818e-01 -2.837757e-02**

**[167,] -1.319818e-01 -2.837757e-02**

**[168,] -1.319818e-01 -2.837757e-02**

**[169,] -1.319818e-01 -2.837757e-02**

**[170,] -1.319818e-01 -2.837757e-02**

**[171,] -1.319818e-01 -2.837757e-02**

**[172,] -1.319818e-01 -2.837757e-02**

**[173,] -1.319818e-01 -2.837757e-02**

**[174,] -1.319818e-01 -2.837757e-02**

**[175,] -1.318261e-01 -2.822186e-02**

**[176,] -1.318261e-01 -2.822186e-02**

**[177,] -1.318261e-01 -2.822186e-02**

**[178,] -1.318261e-01 -2.822186e-02**

**[179,] -1.318261e-01 -2.822186e-02**

**[180,] -1.318261e-01 -2.822186e-02**

**[181,] -1.315147e-01 -2.791045e-02**

**[182,] -1.312033e-01 -2.759903e-02**

**[183,] -1.301133e-01 -2.650908e-02**

**[184,] -1.301133e-01 -2.650908e-02**

**[185,] -1.293348e-01 -2.573055e-02**

**[186,] -1.293348e-01 -2.573055e-02**

**[187,] -1.290234e-01 -2.541913e-02**

**[188,] -1.288677e-01 -2.526342e-02**

**[189,] -1.288677e-01 -2.526342e-02**

**[190,] -1.288677e-01 -2.526342e-02**

**[191,] -1.257875e-01 -2.709789e-02**

**[192,] -1.230188e-01 -2.924377e-02**

**[193,] -1.188488e-01 -2.998829e-02**

**[194,] -1.146787e-01 -3.073280e-02**

**[195,] -1.137444e-01 -2.979856e-02**

**[196,] -1.128102e-01 -2.886432e-02**

**[197,] -1.032584e-01 -3.405630e-02**

**[198,] -9.631963e-02 -3.694670e-02**

**[199,] -9.093792e-02 -4.139417e-02**

**[200,] -7.347909e-02 -4.359370e-02**

**[201,] -5.891066e-02 -3.885445e-02**

**[202,] -5.439515e-02 -3.433894e-02**

**[203,] -4.418652e-02 -3.395949e-02**

**[204,] -3.335506e-02 -3.295721e-02**

**[205,] -1.340492e-02 -3.266543e-02**

**[206,] -4.407923e-03 -2.858303e-02**

**[207,] 9.970782e-03 -2.894809e-02**

**[208,] 2.376068e-02 -2.498737e-02**

**[209,] 4.495648e-02 -2.344994e-02**

**[210,] 8.762547e-02 -2.501225e-02**

**[211,] -1.319818e-01 -2.837757e-02**

**[212,] -1.319818e-01 -2.837757e-02**

**[213,] -1.319818e-01 -2.837757e-02**

**[214,] -1.319818e-01 -2.837757e-02**

**[215,] -1.319818e-01 -2.837757e-02**

**[216,] -1.319818e-01 -2.837757e-02**

**[217,] -1.319818e-01 -2.837757e-02**

**[218,] -1.319818e-01 -2.837757e-02**

**[219,] -1.319818e-01 -2.837757e-02**

**[220,] -1.319818e-01 -2.837757e-02**

**[221,] -1.319818e-01 -2.837757e-02**

**[222,] -1.319818e-01 -2.837757e-02**

**[223,] -1.319818e-01 -2.837757e-02**

**[224,] -1.319818e-01 -2.837757e-02**

**[225,] -1.319818e-01 -2.837757e-02**

**[226,] -1.319818e-01 -2.837757e-02**

**[227,] -1.319818e-01 -2.837757e-02**

**[228,] -1.319818e-01 -2.837757e-02**

**[229,] -1.319818e-01 -2.837757e-02**

**[230,] -1.319818e-01 -2.837757e-02**

**[231,] -1.319818e-01 -2.837757e-02**

**[232,] -1.319818e-01 -2.837757e-02**

**[233,] -1.319818e-01 -2.837757e-02**

**[234,] -1.319818e-01 -2.837757e-02**

**[235,] -1.319818e-01 -2.837757e-02**

**[236,] -1.319818e-01 -2.837757e-02**

**[237,] -1.319818e-01 -2.837757e-02**

**[238,] -1.319818e-01 -2.837757e-02**

**[239,] -1.319818e-01 -2.837757e-02**

**[240,] -1.319818e-01 -2.837757e-02**

**[241,] -1.319818e-01 -2.837757e-02**

**[242,] -1.319818e-01 -2.837757e-02**

**[243,] -1.319818e-01 -2.837757e-02**

**[244,] -1.319818e-01 -2.837757e-02**

**[245,] -1.319818e-01 -2.837757e-02**

**[246,] -1.319818e-01 -2.837757e-02**

**[247,] -1.319818e-01 -2.837757e-02**

**[248,] -1.319818e-01 -2.837757e-02**

**[249,] -1.319818e-01 -2.837757e-02**

**[250,] -1.319818e-01 -2.837757e-02**

**[251,] -1.318261e-01 -2.822186e-02**

**[252,] -1.318261e-01 -2.822186e-02**

**[253,] -1.318261e-01 -2.822186e-02**

**[254,] -1.318261e-01 -2.822186e-02**

**[255,] -1.318261e-01 -2.822186e-02**

**[256,] -1.318261e-01 -2.822186e-02**

**[257,] -1.318261e-01 -2.822186e-02**

**[258,] -1.318261e-01 -2.822186e-02**

**[259,] -1.318261e-01 -2.822186e-02**

**[260,] -1.318261e-01 -2.822186e-02**

**[261,] -1.318261e-01 -2.822186e-02**

**[262,] -1.318261e-01 -2.822186e-02**

**[263,] -1.318261e-01 -2.822186e-02**

**[264,] -1.318261e-01 -2.822186e-02**

**[265,] -1.316704e-01 -2.806615e-02**

**[266,] -1.259092e-01 -2.230499e-02**

**[267,] -1.259092e-01 -2.230499e-02**

**[268,] -1.237293e-01 -2.012509e-02**

**[269,] -1.203038e-01 -1.669953e-02**

**[270,] -1.182796e-01 -1.467534e-02**

**[271,] -1.119296e-01 -1.323996e-02**

**[272,] -1.088155e-01 -1.012581e-02**

**[273,] -1.039885e-01 -5.298894e-03**

**[274,] -1.002516e-01 -1.561923e-03**

**[275,] -8.973153e-02 -8.710571e-04**

**[276,] -8.303612e-02 5.824349e-03**

**[277,] -7.665213e-02 1.220834e-02**

**[278,] -6.523186e-02 8.884841e-03**

**[279,] -5.471181e-02 9.575707e-03**

**[280,] -4.394839e-02 6.807697e-04**

**[281,] -1.319818e-01 -2.837757e-02**

**[282,] -1.319818e-01 -2.837757e-02**

**[283,] -1.319818e-01 -2.837757e-02**

**[284,] -1.319818e-01 -2.837757e-02**

**[285,] -1.319818e-01 -2.837757e-02**

**[286,] -1.319818e-01 -2.837757e-02**

**[287,] -1.319818e-01 -2.837757e-02**

**[288,] -1.319818e-01 -2.837757e-02**

**[289,] -1.319818e-01 -2.837757e-02**

**[290,] -1.319818e-01 -2.837757e-02**

**[291,] -1.319818e-01 -2.837757e-02**

**[292,] -1.319818e-01 -2.837757e-02**

**[293,] -1.319818e-01 -2.837757e-02**

**[294,] -1.319818e-01 -2.837757e-02**

**[295,] -1.319818e-01 -2.837757e-02**

**[296,] -1.319818e-01 -2.837757e-02**

**[297,] -1.319818e-01 -2.837757e-02**

**[298,] -1.319818e-01 -2.837757e-02**

**[299,] -1.319818e-01 -2.837757e-02**

**[300,] -1.319818e-01 -2.837757e-02**

**[301,] -1.319818e-01 -2.837757e-02**

**[302,] -1.319818e-01 -2.837757e-02**

**[303,] -1.319818e-01 -2.837757e-02**

**[304,] -1.319818e-01 -2.837757e-02**

**[305,] -1.319818e-01 -2.837757e-02**

**[306,] -1.319818e-01 -2.837757e-02**

**[307,] -1.319818e-01 -2.837757e-02**

**[308,] -1.319818e-01 -2.837757e-02**

**[309,] -1.319818e-01 -2.837757e-02**

**[310,] -1.319818e-01 -2.837757e-02**

**[311,] -1.319818e-01 -2.837757e-02**

**[312,] -1.319818e-01 -2.837757e-02**

**[313,] -1.319818e-01 -2.837757e-02**

**[314,] -1.319818e-01 -2.837757e-02**

**[315,] -1.319818e-01 -2.837757e-02**

**[316,] -1.319818e-01 -2.837757e-02**

**[317,] -1.319818e-01 -2.837757e-02**

**[318,] -1.319818e-01 -2.837757e-02**

**[319,] -1.319818e-01 -2.837757e-02**

**[320,] -1.319818e-01 -2.837757e-02**

**[321,] -1.319818e-01 -2.837757e-02**

**[322,] -1.319818e-01 -2.837757e-02**

**[323,] -1.319818e-01 -2.837757e-02**

**[324,] -1.319818e-01 -2.837757e-02**

**[325,] -1.319818e-01 -2.837757e-02**

**[326,] -1.319818e-01 -2.837757e-02**

**[327,] -1.319818e-01 -2.837757e-02**

**[328,] -1.319818e-01 -2.837757e-02**

**[329,] -1.319818e-01 -2.837757e-02**

**[330,] -1.319818e-01 -2.837757e-02**

**[331,] -1.319818e-01 -2.837757e-02**

**[332,] -1.319818e-01 -2.837757e-02**

**[333,] -1.319818e-01 -2.837757e-02**

**[334,] -1.319818e-01 -2.837757e-02**

**[335,] -1.319818e-01 -2.837757e-02**

**[336,] -1.319818e-01 -2.837757e-02**

**[337,] -1.319818e-01 -2.837757e-02**

**[338,] -1.319818e-01 -2.837757e-02**

**[339,] -1.318261e-01 -2.822186e-02**

**[340,] -1.316704e-01 -2.806615e-02**

**[341,] -1.316704e-01 -2.806615e-02**

**[342,] -1.315147e-01 -2.791045e-02**

**[343,] -1.315147e-01 -2.791045e-02**

**[344,] -1.315147e-01 -2.791045e-02**

**[345,] -1.313590e-01 -2.775474e-02**

**[346,] -1.313590e-01 -2.775474e-02**

**[347,] -1.312033e-01 -2.759903e-02**

**[348,] -1.259773e-01 -3.220221e-02**

**[349,] -1.259773e-01 -3.220221e-02**

**[350,] -1.259773e-01 -3.220221e-02**

**[351,] -1.319818e-01 -2.837757e-02**

**[352,] -1.319818e-01 -2.837757e-02**

**[353,] -1.319818e-01 -2.837757e-02**

**[354,] -1.319818e-01 -2.837757e-02**

**[355,] -1.319818e-01 -2.837757e-02**

**[356,] -1.319818e-01 -2.837757e-02**

**[357,] -1.319818e-01 -2.837757e-02**

**[358,] -1.319818e-01 -2.837757e-02**

**[359,] -1.319818e-01 -2.837757e-02**

**[360,] -1.319818e-01 -2.837757e-02**

**[361,] -1.319818e-01 -2.837757e-02**

**[362,] -1.319818e-01 -2.837757e-02**

**[363,] -1.319818e-01 -2.837757e-02**

**[364,] -1.319818e-01 -2.837757e-02**

**[365,] -1.319818e-01 -2.837757e-02**

**[366,] -1.319818e-01 -2.837757e-02**

**[367,] -1.319818e-01 -2.837757e-02**

**[368,] -1.319818e-01 -2.837757e-02**

**[369,] -1.319818e-01 -2.837757e-02**

**[370,] -1.319818e-01 -2.837757e-02**

**[371,] -1.319818e-01 -2.837757e-02**

**[372,] -1.319818e-01 -2.837757e-02**

**[373,] -1.319818e-01 -2.837757e-02**

**[374,] -1.319818e-01 -2.837757e-02**

**[375,] -1.319818e-01 -2.837757e-02**

**[376,] -1.319818e-01 -2.837757e-02**

**[377,] -1.319818e-01 -2.837757e-02**

**[378,] -1.319818e-01 -2.837757e-02**

**[379,] -1.319818e-01 -2.837757e-02**

**[380,] -1.319818e-01 -2.837757e-02**

**[381,] -1.319818e-01 -2.837757e-02**

**[382,] -1.319818e-01 -2.837757e-02**

**[383,] -1.319818e-01 -2.837757e-02**

**[384,] -1.319818e-01 -2.837757e-02**

**[385,] -1.319818e-01 -2.837757e-02**

**[386,] -1.319818e-01 -2.837757e-02**

**[387,] -1.319818e-01 -2.837757e-02**

**[388,] -1.319818e-01 -2.837757e-02**

**[389,] -1.319818e-01 -2.837757e-02**

**[390,] -1.319818e-01 -2.837757e-02**

**[391,] -1.319818e-01 -2.837757e-02**

**[392,] -1.319818e-01 -2.837757e-02**

**[393,] -1.319818e-01 -2.837757e-02**

**[394,] -1.319818e-01 -2.837757e-02**

**[395,] -1.319818e-01 -2.837757e-02**

**[396,] -1.319818e-01 -2.837757e-02**

**[397,] -1.319818e-01 -2.837757e-02**

**[398,] -1.319818e-01 -2.837757e-02**

**[399,] -1.319818e-01 -2.837757e-02**

**[400,] -1.319818e-01 -2.837757e-02**

**[401,] -1.319818e-01 -2.837757e-02**

**[402,] -1.318261e-01 -2.822186e-02**

**[403,] -1.318261e-01 -2.822186e-02**

**[404,] -1.318261e-01 -2.822186e-02**

**[405,] -1.318261e-01 -2.822186e-02**

**[406,] -1.318261e-01 -2.822186e-02**

**[407,] -1.318261e-01 -2.822186e-02**

**[408,] -1.318261e-01 -2.822186e-02**

**[409,] -1.318261e-01 -2.822186e-02**

**[410,] -1.318261e-01 -2.822186e-02**

**[411,] -1.318261e-01 -2.822186e-02**

**[412,] -1.315147e-01 -2.791045e-02**

**[413,] -1.315147e-01 -2.791045e-02**

**[414,] -1.315147e-01 -2.791045e-02**

**[415,] -1.308919e-01 -2.728762e-02**

**[416,] -1.308919e-01 -2.728762e-02**

**[417,] -1.308919e-01 -2.728762e-02**

**[418,] -1.308919e-01 -2.728762e-02**

**[419,] -1.308919e-01 -2.728762e-02**

**[420,] -1.308919e-01 -2.728762e-02**

**[421,] -1.319818e-01 -2.837757e-02**

**[422,] -1.319818e-01 -2.837757e-02**

**[423,] -1.319818e-01 -2.837757e-02**

**[424,] -1.319818e-01 -2.837757e-02**

**[425,] -1.319818e-01 -2.837757e-02**

**[426,] -1.319818e-01 -2.837757e-02**

**[427,] -1.319818e-01 -2.837757e-02**

**[428,] -1.319818e-01 -2.837757e-02**

**[429,] -1.319818e-01 -2.837757e-02**

**[430,] -1.319818e-01 -2.837757e-02**

**[431,] -1.319818e-01 -2.837757e-02**

**[432,] -1.319818e-01 -2.837757e-02**

**[433,] -1.319818e-01 -2.837757e-02**

**[434,] -1.319818e-01 -2.837757e-02**

**[435,] -1.319818e-01 -2.837757e-02**

**[436,] -1.319818e-01 -2.837757e-02**

**[437,] -1.319818e-01 -2.837757e-02**

**[438,] -1.319818e-01 -2.837757e-02**

**[439,] -1.319818e-01 -2.837757e-02**

**[440,] -1.319818e-01 -2.837757e-02**

**[441,] -1.319818e-01 -2.837757e-02**

**[442,] -1.319818e-01 -2.837757e-02**

**[443,] -1.319818e-01 -2.837757e-02**

**[444,] -1.319818e-01 -2.837757e-02**

**[445,] -1.319818e-01 -2.837757e-02**

**[446,] -1.319818e-01 -2.837757e-02**

**[447,] -1.319818e-01 -2.837757e-02**

**[448,] -1.319818e-01 -2.837757e-02**

**[449,] -1.319818e-01 -2.837757e-02**

**[450,] -1.319818e-01 -2.837757e-02**

**[451,] -1.319818e-01 -2.837757e-02**

**[452,] -1.319818e-01 -2.837757e-02**

**[453,] -1.319818e-01 -2.837757e-02**

**[454,] -1.319818e-01 -2.837757e-02**

**[455,] -1.319818e-01 -2.837757e-02**

**[456,] -1.319818e-01 -2.837757e-02**

**[457,] -1.319818e-01 -2.837757e-02**

**[458,] -1.319818e-01 -2.837757e-02**

**[459,] -1.319818e-01 -2.837757e-02**

**[460,] -1.319818e-01 -2.837757e-02**

**[461,] -1.319818e-01 -2.837757e-02**

**[462,] -1.318261e-01 -2.822186e-02**

**[463,] -1.318261e-01 -2.822186e-02**

**[464,] -1.318261e-01 -2.822186e-02**

**[465,] -1.316704e-01 -2.806615e-02**

**[466,] -1.307361e-01 -2.713191e-02**

**[467,] -1.276560e-01 -2.896638e-02**

**[468,] -1.276560e-01 -2.896638e-02**

**[469,] -1.268775e-01 -2.818784e-02**

**[470,] -1.265661e-01 -2.787643e-02**

**[471,] -1.265661e-01 -2.787643e-02**

**[472,] -1.222403e-01 -2.846524e-02**

**[473,] -1.217732e-01 -2.799811e-02**

**[474,] -1.200604e-01 -2.628534e-02**

**[475,] -1.183476e-01 -2.457256e-02**

**[476,] -1.164791e-01 -2.270407e-02**

**[477,] -1.147664e-01 -2.099129e-02**

**[478,] -1.095063e-01 -2.064586e-02**

**[479,] -1.046794e-01 -1.581894e-02**

**[480,] -9.755090e-02 -1.360502e-02**

**[481,] -8.073453e-02 3.211347e-03**

**[482,] -7.528478e-02 8.661096e-03**

**[483,] -5.697938e-02 1.713732e-02**

**[484,] -5.206479e-02 1.222273e-02**

**[485,] -3.170118e-02 2.767175e-02**

**[486,] -8.325477e-03 3.138909e-02**

**[487,] 1.968742e-02 3.482903e-02**

**[488,] 3.070860e-02 4.093563e-02**

**[489,] 5.221582e-02 4.278448e-02**

**[490,] 9.848046e-02 6.939077e-02**

**[491,] -1.319818e-01 -2.837757e-02**

**[492,] -1.319818e-01 -2.837757e-02**

**[493,] -1.319818e-01 -2.837757e-02**

**[494,] -1.319818e-01 -2.837757e-02**

**[495,] -1.319818e-01 -2.837757e-02**

**[496,] -1.319818e-01 -2.837757e-02**

**[497,] -1.319818e-01 -2.837757e-02**

**[498,] -1.319818e-01 -2.837757e-02**

**[499,] -1.319818e-01 -2.837757e-02**

**[500,] -1.319818e-01 -2.837757e-02**

**[ reached getOption("max.print") -- omitted 20080 rows ]**

**> df2<-cbind(df,myPrc$x[,1:2])**

**> head(df2)**

**Id Province\_State Country\_Region Date ConfirmedCases Fatalities PC1 PC2**

**1 1 Afghanistan 2020-01-22 0 0 -0.1319818 -0.02837757**

**2 2 Afghanistan 2020-01-23 0 0 -0.1319818 -0.02837757**

**3 3 Afghanistan 2020-01-24 0 0 -0.1319818 -0.02837757**

**4 4 Afghanistan 2020-01-25 0 0 -0.1319818 -0.02837757**

**5 5 Afghanistan 2020-01-26 0 0 -0.1319818 -0.02837757**

**6 6 Afghanistan 2020-01-27 0 0 -0.1319818 -0.02837757**

**> library(ggplot2)**

**> ggplot(data = df2) + geom\_point(mapping = aes(x = PC1, y = PC2))**

**> stat\_ellipse(geom="polygon",col="red",alpha=0.5)**

**geom\_polygon: na.rm = FALSE**

**stat\_ellipse: type = t, level = 0.95, segments = 51, na.rm = FALSE**

**position\_identity**

**> geom\_point(shape=21,col="black")**

**geom\_point: na.rm = FALSE**

**stat\_identity: na.rm = FALSE**

**position\_identity**

**> cor(df[5:6],df[5:6])**

**ConfirmedCases Fatalities**

**ConfirmedCases 1.0000000 0.8740351**

**Fatalities 0.8740351 1.0000000**

**>**

Chart

Description automatically generated

Shape, rectangle

Description automatically generated**Chart, scatter chart

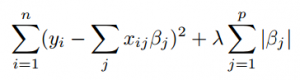
Description automatically generated**

**PRACTICAL 08**

**Lasso Regression in R Programming**

Lasso regression is a classification algorithm that uses shrinkage in simple and sparse models(i.e model with fewer parameters). In Shrinkage, data values are shrunk towards a central point like the mean. Lasso regression is a regularized regression algorithm that performs L1 regularization which adds penalty equal to the absolute value of the magnitude of coefficients.

**“LASSO” stands for Least Absolute Shrinkage and Selection Operator**. Lasso regression is good for models showing high levels of multicollinearity or when you want to automate certain parts of model selection i.e variable selection or parameter elimination. Lasso regression solutions are quadratic programming problems that can best solve with software like [RStudio](https://www.geeksforgeeks.org/introduction-to-r-studio/), Matlab, etc. It has the ability to select predictors.



The algorithm minimizes the sum of squares with constraint. Some **Beta** are shrunk to zero that results in a regression model. A tuning parameter **lambda** controls the strength of the L1 regularization penalty. **lambda** is basically the amount of shrinkage:

* When **lambda** = 0, no parameters are eliminated.
* As **lambda** increases, more and more coefficients are set to zero and eliminated & bias increases.
* When **lambda** = infinity, all coefficients are eliminated.
* As **lambda** decreases, variance increases.

Also, If an intercept is included in the model, it is left unchanged.

Now let’s implementing Lasso regression in [R programming](https://www.geeksforgeeks.org/introduction-to-r-programming-language/).

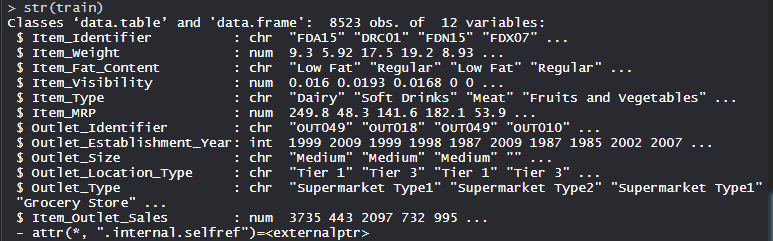
**Implementation in R**

The Dataset

[Big Mart](https://www.kaggle.com/brijbhushannanda1979/bigmart-sales-data/data) dataset consists of 1559 products across 10 stores in different cities. Certain attributes of each product and store have been defined. It consists of 12 features i.e Item\_Identifier( is a unique product ID assigned to every distinct item), Item\_Weight(includes the weight of the product), Item\_Fat\_Content(describes whether the product is low fat or not), Item\_Visibility(mentions the percentage of the total display area of all products in a store allocated to the particular product), Item\_Type(describes the food category to which the item belongs), Item\_MRP(Maximum Retail Price (list price) of the product), Outlet\_Identifier(unique store ID assigned. It consists of an alphanumeric string of length 6), Outlet\_Establishment\_Year(mentions the year in which store was established), Outlet\_Size(tells the size of the store in terms of ground area covered), Outlet\_Location\_Type(tells about the size of the city in which the store is located), Outlet\_Type(tells whether the outlet is just a grocery store or some sort of supermarket) and Item\_Outlet\_Sales( sales of the product in the particular store).

|  |
| --- |
| # Loading data  train = fread("Train\_UWu5bXk.csv")  test = fread("Test\_u94Q5KV.csv")    # Structure  str(train) |

**Output:**



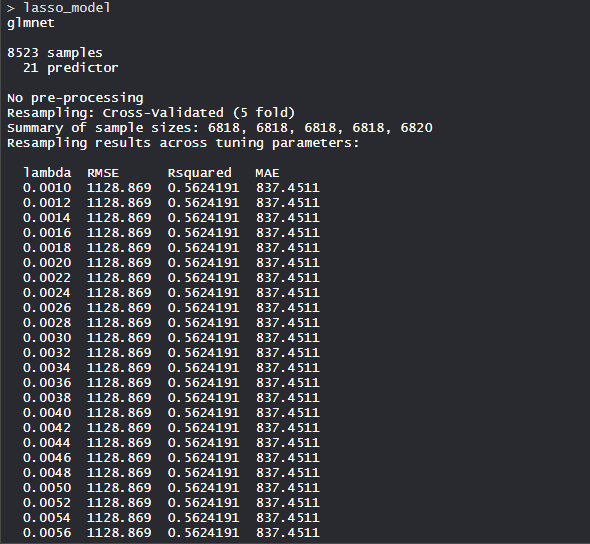
Performing Lasso Regression on Dataset

Using the Lasso regression algorithm on the dataset which includes 12 features with 1559 products across 10 stores in different cities.

|  |
| --- |
| # Installing Packages  install.packages("data.table")  install.packages("dplyr")  install.packages("glmnet")  install.packages("ggplot2")  install.packages("caret")  install.packages("xgboost")  install.packages("e1071")  install.packages("cowplot")    # load packages  library(data.table) # used for reading and manipulation of data  library(dplyr)      # used for data manipulation and joining  library(glmnet)     # used for regression  library(ggplot2)    # used for ploting  library(caret)      # used for modeling  library(xgboost)    # used for building XGBoost model  library(e1071)      # used for skewness  library(cowplot)    # used for combining multiple plots    # Loading datasets  train = fread("Train\_UWu5bXk.csv")  test = fread("Test\_u94Q5KV.csv")    # Setting test dataset  # Combining datasets  # add Item\_Outlet\_Sales to test data  test[, Item\_Outlet\_Sales := NA]    combi = rbind(train, test)    # Missing Value Treatment  missing\_index = which(is.na(combi$Item\_Weight))  for(i in missing\_index)  {    item = combi$Item\_Identifier[i]    combi$Item\_Weight[i] =        mean(combi$Item\_Weight[combi$Item\_Identifier == item],                                                   na.rm = T)  }    # Replacing 0 in Item\_Visibility with mean  zero\_index = which(combi$Item\_Visibility == 0)  for(i in zero\_index)  {    item = combi$Item\_Identifier[i]    combi$Item\_Visibility[i] =        mean(combi$Item\_Visibility[combi$Item\_Identifier == item],                                                       na.rm = T)  }    # Label Encoding  # To convert categorical in numerical  combi[, Outlet\_Size\_num := ifelse(Outlet\_Size == "Small", 0,                                   ifelse(Outlet\_Size == "Medium",                                                          1, 2))]    combi[, Outlet\_Location\_Type\_num :=           ifelse(Outlet\_Location\_Type == "Tier 3", 0,                  ifelse(Outlet\_Location\_Type == "Tier 2", 1, 2))]    combi[, c("Outlet\_Size", "Outlet\_Location\_Type") := NULL]    # One Hot Encoding  # To convert categorical in numerical  ohe\_1 = dummyVars("~.", data = combi[, -c("Item\_Identifier",                                            "Outlet\_Establishment\_Year",                                            "Item\_Type")], fullRank = T)  ohe\_df = data.table(predict(ohe\_1, combi[, -c("Item\_Identifier",                                             "Outlet\_Establishment\_Year",                                             "Item\_Type")]))    combi = cbind(combi[, "Item\_Identifier"], ohe\_df)    # Remove skewness  skewness(combi$Item\_Visibility)  skewness(combi$price\_per\_unit\_wt)    # log + 1 to avoid division by zero  combi[, Item\_Visibility := log(Item\_Visibility + 1)]    # Scaling and Centering data  num\_vars = which(sapply(combi, is.numeric)) # index of numeric features  num\_vars\_names = names(num\_vars)    combi\_numeric = combi[, setdiff(num\_vars\_names,                                "Item\_Outlet\_Sales"),                                 with = F]    prep\_num = preProcess(combi\_numeric,                        method=c("center", "scale"))  combi\_numeric\_norm = predict(prep\_num, combi\_numeric)    # removing numeric independent variables  combi[, setdiff(num\_vars\_names,                  "Item\_Outlet\_Sales") := NULL]  combi = cbind(combi, combi\_numeric\_norm)    # splitting data back to train and test  train = combi[1:nrow(train)]  test = combi[(nrow(train) + 1):nrow(combi)]    # Removing Item\_Outlet\_Sales  test[, Item\_Outlet\_Sales := NULL]    # Model Building :Lasso Regression  set.seed(123)  control = trainControl(method ="cv", number = 5)  Grid\_la\_reg = expand.grid(alpha = 1,                lambda = seq(0.001, 0.1, by = 0.0002))    # Training lasso regression model  lasso\_model = train(x = train[, -c("Item\_Identifier",                                 "Item\_Outlet\_Sales")],                      y = train$Item\_Outlet\_Sales,                      method = "glmnet",                      trControl = control,                      tuneGrid = Grid\_reg                      )  lasso\_model    # mean validation score  mean(lasso\_model$resample$RMSE)    # Plot  plot(lasso\_model, main = "Lasso Regression") |

**Output:**

* **Model lasso\_model:**



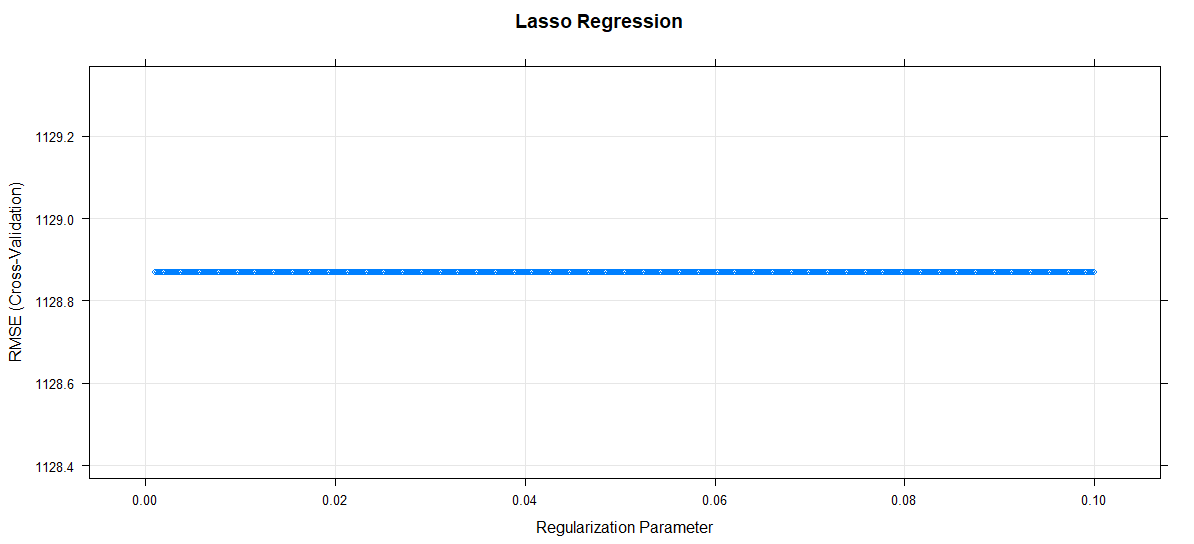
The Lasso regression model uses the alpha value as 1 and lambda value as 0.1. RMSE was used to select the optimal model using the smallest value.

* **Mean validation score:**

output

The mean validation score of the model is 1128.869.

* **Plot:**



The regularization parameter increases, RMSE remains constant.

**PRACTICAL 09**

Practical of Time Series Forecasting

**Time Series Forecasting:-** It is a technique for the prediction of events through a sequence of time. The technique is used across many fields of study, from the geology to behavior to economics. The techniques predict future events by [analyzing](https://searchbusinessanalytics.techtarget.com/definition/predictive-analytics) the trends of the past, on the assumption that future trends will hold similar to historical trends. Time series forecasting is sometimes just the analysis of experts studying a field and offering their predictions. In many modern applications, however, time series forecasting uses computer technologies.

**Code:-**

data("AirPassengers")

class(AirPassengers)

start(AirPassengers)

end(AirPassengers)

frequency(AirPassengers)

summary(AirPassengers)

plot(AirPassengers)

abline(reg=lm(AirPassengers~time(AirPassengers)))

cycle(AirPassengers)

plot(aggregate(AirPassengers,FUN=mean))

boxplot(AirPassengers~cycle(AirPassengers))

**Output:-**

> class(AirPassengers)

[1] "ts"

> start(AirPassengers)

[1] 1949 1

> end(AirPassengers)

[1] 1960 12

> frequency(AirPassengers)

[1] 12

> summary(AirPassengers)

Min. 1st Qu. Median Mean 3rd Qu. Max.

104.0 180.0 265.5 280.3 360.5 622.0

> plot(AirPassengers)

> abline(reg=lm(AirPassengers~time(AirPassengers)))

> cycle(AirPassengers)

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

1949 1 2 3 4 5 6 7 8 9 10 11 12

1950 1 2 3 4 5 6 7 8 9 10 11 12

1951 1 2 3 4 5 6 7 8 9 10 11 12

1952 1 2 3 4 5 6 7 8 9 10 11 12

1953 1 2 3 4 5 6 7 8 9 10 11 12

1954 1 2 3 4 5 6 7 8 9 10 11 12

1955 1 2 3 4 5 6 7 8 9 10 11 12

1956 1 2 3 4 5 6 7 8 9 10 11 12

1957 1 2 3 4 5 6 7 8 9 10 11 12

1958 1 2 3 4 5 6 7 8 9 10 11 12

1959 1 2 3 4 5 6 7 8 9 10 11 12

1960 1 2 3 4 5 6 7 8 9 10 11 12

Chart, histogram

Description automatically generated Chart, box and whisker chart

Description automatically generated

**PRACTICAL 10**

**K-Means Clustering in R:**

Clustering is a technique in machine learning that attempts to find *clusters* of observations within a dataset.

The goal is to find clusters such that the observations within each cluster are quite similar to each other, while observations in different clusters are quite different from each other.

Clustering is a form of unsupervised learning because we’re simply attempting to find structure within a dataset rather than predicting the value of some response variable.

Clustering is often used in marketing when companies have access to information like:

* Household income
* Household size
* Head of household Occupation
* Distance from nearest urban area

When this information is available, clustering can be used to

identify households that are similar and may be more likely to purchase certain products or respond better to a certain type of advertising.

One of the most common forms of clustering is known as **k-means clustering**.

**What is K-Means Clustering?**

K-means clustering is a technique in which we place each observation in a dataset into one of *K* clusters.

The end goal is to have *K*clusters in which the observations

within each cluster are quite similar to each other while the observations in different clusters are quite different from each other.

In practice, we use the following steps to perform K-means clustering:

**1. Choose a value for *K*.**

* First, we must decide how many clusters we’d like to identify in the data. Often we have to simply test several different values for *K* and analyze the results to see which number of clusters seems to make the most sense for a given problem.

**2. Randomly assign each observation to an initial cluster, from 1 to *K.***

**3. Perform the following procedure until the cluster assignments stop changing.**

* For each of the *K*clusters, compute the cluster *centroid.* This is simply the vector of the *p* feature means for the observations in the *k*th cluster.
* Assign each observation to the cluster whose centroid is closest. Here, *closest* is defined using Euclidean distance.

**K-Means Clustering in R**

**Step 1: Load the Necessary Packages**

First, we’ll load two packages that contain several useful functions for k-means clustering in R.

**library(factoextra)**

**library(cluster)**

**Step 2: Load and Prep the Data**

For this example we’ll use the *USArrests* dataset built into R, which contains the number of arrests per 100,000 residents in each U.S. state in 1973 for *Murder*, *Assault*, and *Rape*along with the percentage of the population in each state living in urban areas, *UrbanPop*.

The following code shows how to do the following:

* Load the *USArrests* dataset
* Remove any rows with missing values
* Scale each variable in the dataset to have a mean of 0 and a standard deviation of 1

**#load data**

**df <- USArrests**

**#remove rows with missing values**

**df <- na.omit(df)**

**#scale each variable to have a mean of 0 and sd of 1**

**df <- scale(df)**

**#view first six rows of dataset**

**head(df)**

**Murder Assault UrbanPop Rape**

**Alabama 1.24256408 0.7828393 -0.5209066 -0.003416473**

**Alaska 0.50786248 1.1068225 -1.2117642 2.484202941**

**Arizona 0.07163341 1.4788032 0.9989801 1.042878388**

**Arkansas 0.23234938 0.2308680 -1.0735927 -0.184916602**

**California 0.27826823 1.2628144 1.7589234 2.067820292**

**Colorado 0.02571456 0.3988593 0.8608085 1.864967207**

**Step 3: Find the Optimal Number of Clusters**

To perform k-means clustering in R we can use the built-in **kmeans()** function, which uses the following syntax:

**kmeans(data, centers, nstart)**

where:

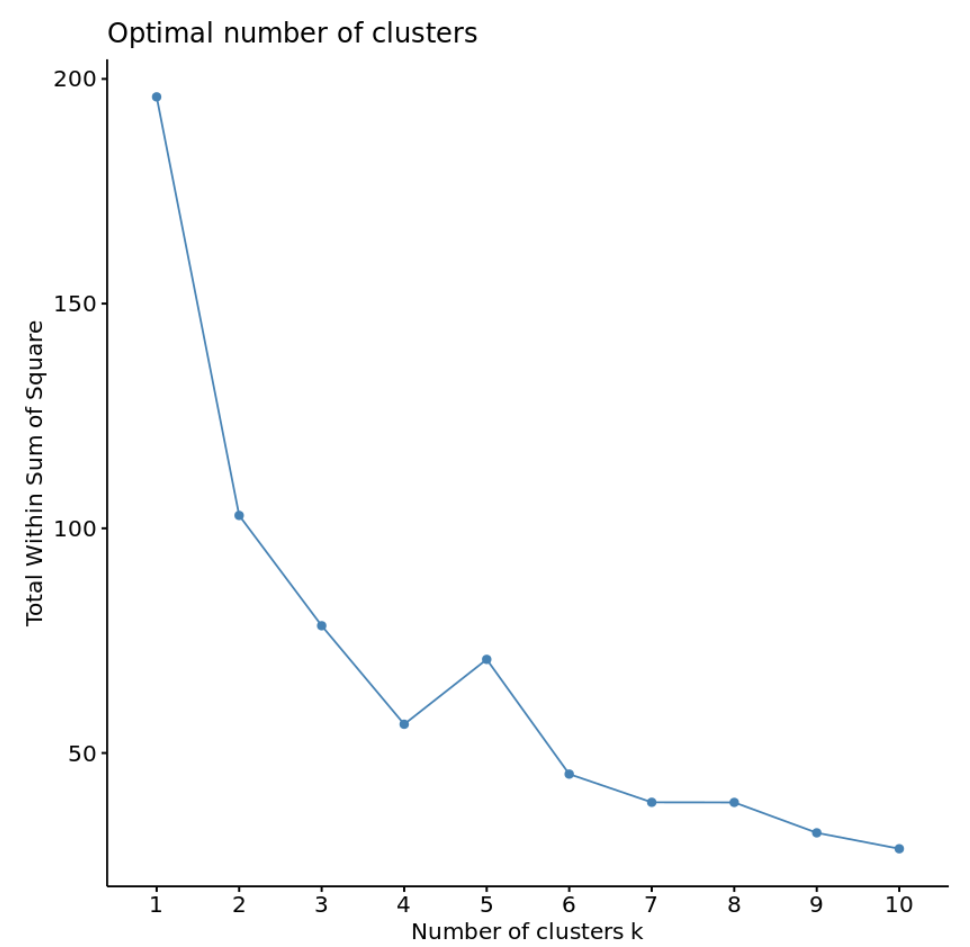
* **data:** Name of the dataset.
* **centers:** The number of clusters, denoted *k*.
* **nstart:** The number of initial configurations. Because it’s possible that different initial starting clusters can lead to different results, it’s recommended to use several different initial configurations. The k-means algorithm will find the initial configurations that lead to the smallest within-cluster variation.

Since we don’t know beforehand how many clusters is optimal, we’ll create two different plots that can help us decide:

**1. Number of Clusters vs. the Total Within Sum of Squares**

First, we’ll use the **fviz\_nbclust()** function to create a plot of the number of clusters vs. the total within sum of squares:

**fviz\_nbclust(df, kmeans, method = "wss")**



Typically when we create this type of plot we look for an “elbow” where the sum of squares begins to “bend” or level off. This is typically the optimal number of clusters.

For this plot it appear that there is a bit of an elbow or “bend” at k = 4 clusters.

**2. Number of Clusters vs. Gap Statistic**

Another way to determine the optimal number of clusters is to use a metric known as the gap statistic, which compares the total intra-cluster variation for different values of k with their expected values for a distribution with no clustering.

We can calculate the gap statistic for each number of clusters using the **clusGap()** function from the *cluster* package along with a plot of clusters vs. gap statistic using the **fviz\_gap\_stat()** function:

**#calculate gap statistic based on number of clusters**

**gap\_stat <- clusGap(df,**

**FUN = kmeans,**

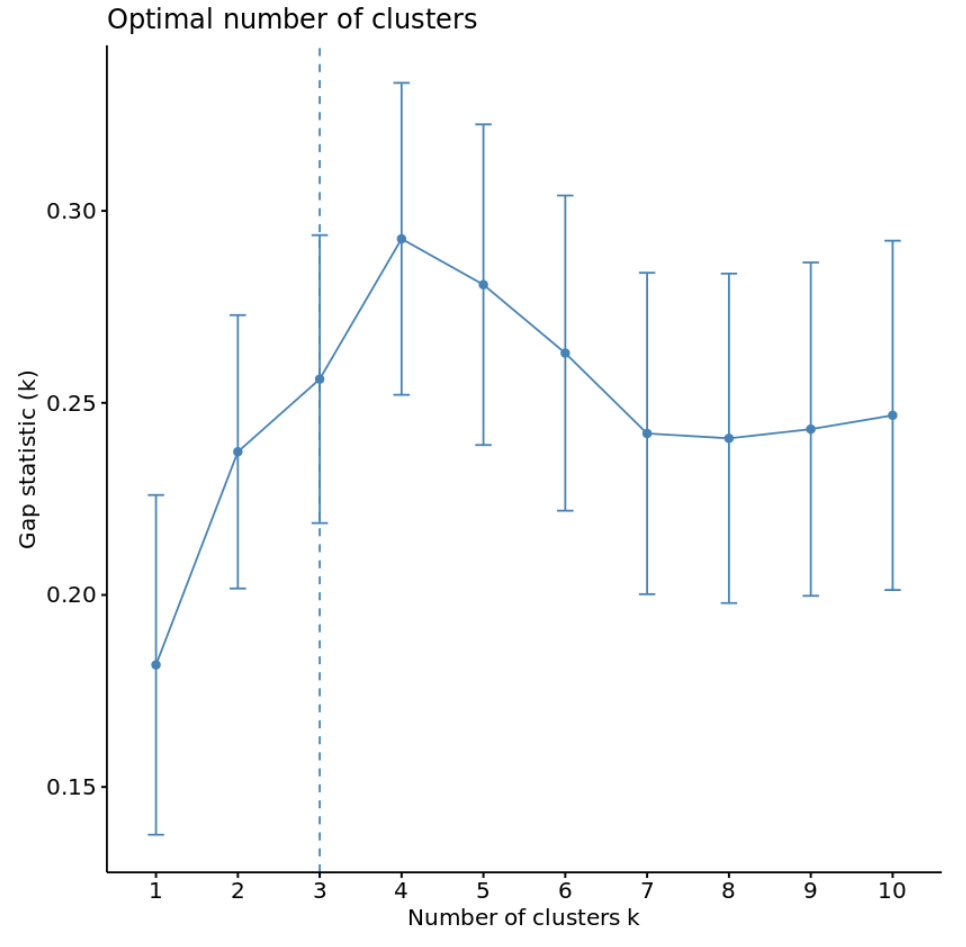
**nstart = 25,**

**K.max = 10,**

**B = 50)**

**#plot number of clusters vs. gap statistic**

**fviz\_gap\_stat(gap\_stat)**



From the plot we can see that gap statistic is highest at k = 4 clusters, which matches the elbow method we used earlier.

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

Lastly, we can perform k-means clustering on the dataset using the optimal value for *k* of 4:

**#make this example reproducible**

**set.seed(1)**

**#perform k-means clustering with k = 4 clusters**

**km <- kmeans(df, centers = 4, nstart = 25)**

**#view results**

**km**

**K-means clustering with 4 clusters of sizes 16, 13, 13, 8**

**Cluster means:**

**Murder Assault UrbanPop Rape**

**1 -0.4894375 -0.3826001 0.5758298 -0.26165379**

**2 -0.9615407 -1.1066010 -0.9301069 -0.96676331**

**3 0.6950701 1.0394414 0.7226370 1.27693964**

**4 1.4118898 0.8743346 -0.8145211 0.01927104**

**Clustering vector:**

**Alabama Alaska Arizona Arkansas California Colorado**

**4 3 3 4 3 3**

**Connecticut Delaware Florida Georgia Hawaii Idaho**

**1 1 3 4 1 2**

**Illinois Indiana Iowa Kansas Kentucky Louisiana**

**3 1 2 1 2 4**

**Maine Maryland Massachusetts Michigan Minnesota Mississippi**

**2 3 1 3 2 4**

**Missouri Montana Nebraska Nevada New Hampshire New Jersey**

**3 2 2 3 2 1**

**New Mexico New York North Carolina North Dakota Ohio Oklahoma**

**3 3 4 2 1 1**

**Oregon Pennsylvania Rhode Island South Carolina South Dakota Tennessee**

**1 1 1 4 2 4**

**Texas Utah Vermont Virginia Washington West Virginia**

**3 1 2 1 1 2**

**Wisconsin Wyoming**

**2 1**

**Within cluster sum of squares by cluster:**

**[1] 16.212213 11.952463 19.922437 8.316061**

**(between\_SS / total\_SS = 71.2 %)**

**Available components:**

**[1] "cluster" "centers" "totss" "withinss" "tot.withinss" "betweenss"**

**[7] "size" "iter" "ifault"**

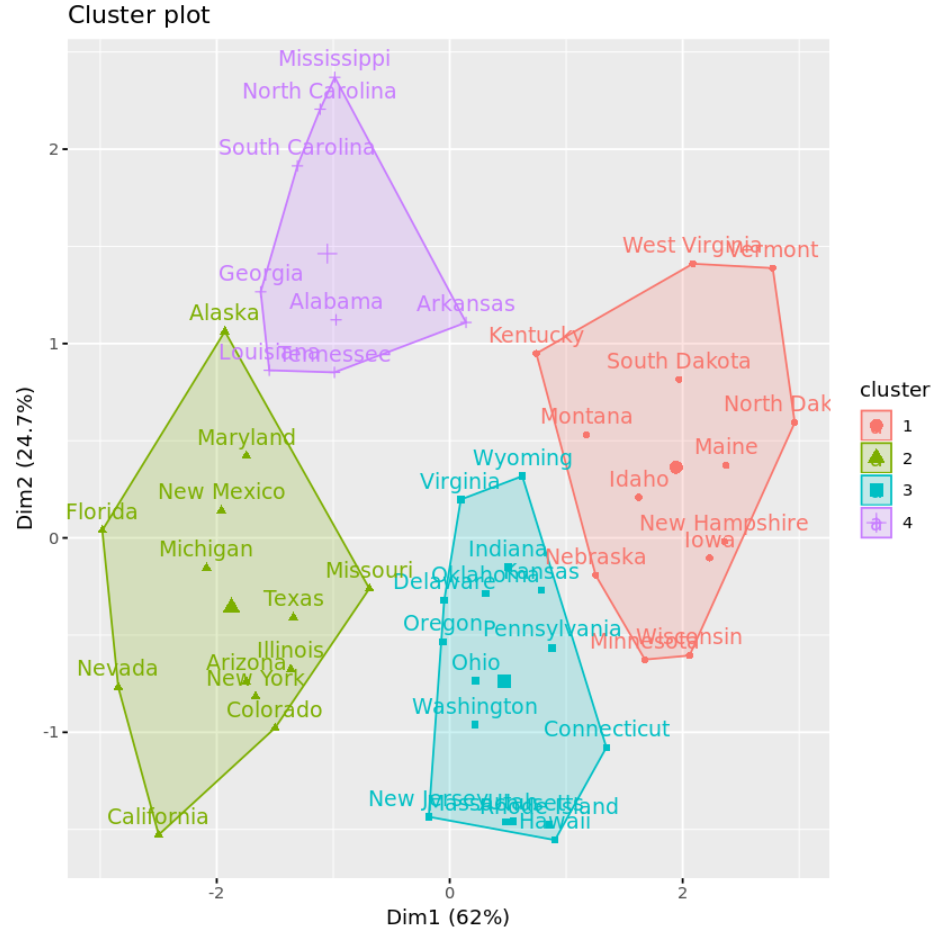
From the results we can see that:

* **16**states were assigned to the first cluster
* **13** states were assigned to the second cluster
* **13** states were assigned to the third cluster
* **8**states were assigned to the fourth cluster

We can visualize the clusters on a scatterplot that displays the first two principal components on the axes using the **fivz\_cluster()** function:

**#plot results of final k-means model**

**fviz\_cluster(km, data = df)**



We can also use the **aggregate()** function to find the mean of the variables in each cluster:

**#find means of each cluster**

**aggregate(USArrests, by=list(cluster=km$cluster), mean)**

**cluster Murder Assault UrbanPop Rape**

**1 3.60000 78.53846 52.07692 12.17692**

**2 10.81538 257.38462 76.00000 33.19231**

**3 5.65625 138.87500 73.87500 18.78125**

**4 13.93750 243.62500 53.75000 21.41250**

We interpret this output is as follows:

* The mean number of murders per 100,000 citizens among the states in cluster 1 is **3.6**.
* The mean number of assaults per 100,000 citizens among the states in cluster 1 is **78.5**.
* The mean percentage of residents living in an urban area among the states in cluster 1 is **52.1%**.
* The mean number of rapes per 100,000 citizens among the states in cluster 1 is **12.2.**

And so on.

We can also append the cluster assignments of each state back to the original dataset:

**#add cluster assigment to original data**

**final\_data <- cbind(USArrests, cluster = km$cluster)**

**#view final data**

**head(final\_data)**

**Murder Assault UrbanPop Rape cluster**

**Alabama 13.2 236 58 21.2 4**

**Alaska 10.0 263 48 44.5 2**

**Arizona 8.1 294 80 31.0 2**

**Arkansas 8.8 190 50 19.5 4**

**California 9.0 276 91 40.6 2**

**Colorado 7.9 204 78 38.7 2**

**Pros & Cons of K-Means Clustering**

K-means clustering offers the following benefits:

* It is a fast algorithm.
* It can handle large datasets well.

However, it comes with the following potential drawbacks:

* It requires us to specify the number of clusters before performing the algorithm.
* It’s sensitive to outliers.

R-Code

library(factoextra)

library(cluster)

#########################################

#LOAD DATA

#########################################

#load and prep data

df <- USArrests

df <- na.omit(df)

df <- scale(df)

#########################################

#DETERMINE HOW MANY CLUSTERS IS OPTIMAL

#########################################

#plot number of clusters vs. total within sum of squares

fviz\_nbclust(df, kmeans, method = "wss")

#calculate gap statistic based on number of clusters

gap\_stat <- clusGap(df,

FUN = kmeans,

nstart = 25,

K.max = 10,

B = 50)

#plot number of clusters vs. gap statistic

fviz\_gap\_stat(gap\_stat)

##########################################

#PERFORM K-MEANS CLUSTERING WITH OPTIMAL K

##########################################

#make this example reproducible

set.seed(1)

#perform k-means clustering with k = 4 clusters

km <- kmeans(df, centers = 4, nstart = 25)

#view results

km

#plot results of final k-means model

fviz\_cluster(km, data = df)

#find mean of each cluster

aggregate(USArrests, by=list(cluster=km$cluster), mean)

#add cluster assigment to original data

final\_data <- cbind(USArrests, cluster = km$cluster)

#view final data

head(final\_data)

**PRACTICAL 11**

Clustering:

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***

It does it by finding some similar patterns in the unlabeled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

The clustering technique is commonly used for **statistical data analysis.**

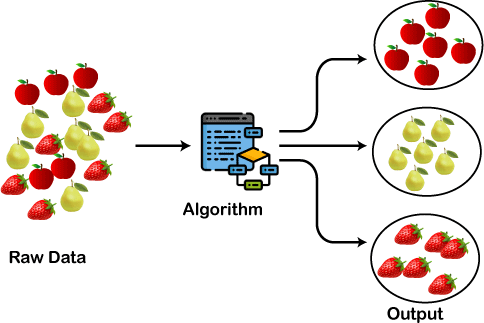
**Example**: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.

The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

* Market Segmentation
* Statistical data analysis
* Social network analysis
* Image segmentation
* Anomaly detection, etc.

Apart from these general usages, it is used by the **Amazon** in its recommendation system to provide the recommendations as per the past search of products. **Netflix** also uses this technique to recommend the movies and web-series to its users as per the watch history.

The below diagram explains the working of the clustering algorithm. We can see the different fruits are divided into several groups with similar properties.



Types of Clustering Methods

The clustering methods are broadly divided into **Hard clustering** (datapoint belongs to only one group) and **Soft Clustering** (data points can belong to another group also). But there are also other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

#### Partitioning Clustering

1. **Density-Based Clustering**

#### Distribution Model-Based Clustering

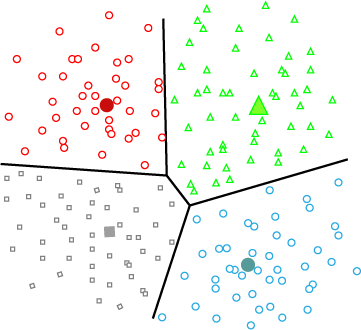
1. **Hierarchical Clustering**

#### Fuzzy Clustering

Partitioning Clustering

It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the **centroid-based method**. The most common example of partitioning clustering is the [**K-**](https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning)[**Means Clustering algorithm**.](https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning)

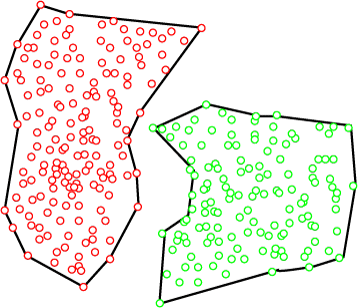
In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.



Density-Based Clustering

The density-based clustering method connects the highly-dense areas into clusters, and the arbitrarily shaped distributions are formed as long as the dense region can be connected. This algorithm does it by identifying different clusters in the dataset and connects the areas of high densities into clusters. The dense areas in data space are divided from each other by sparser areas.

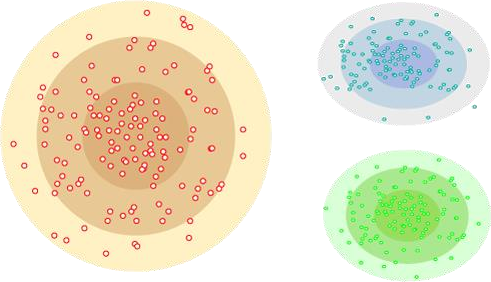
These algorithms can face difficulty in clustering the data points if the dataset has varying densities and high dimensions.



Distribution Model-Based Clustering

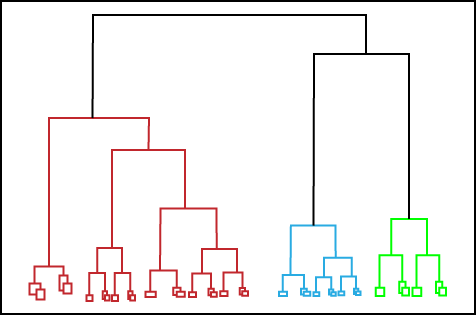
In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions commonly **Gaussian Distribution**.

The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



Hierarchical Clustering

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**. The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm**.



Fuzzy Clustering

[Fuzzy](https://www.javatpoint.com/fuzzy-logic) clustering is a type of soft method in which a data object may belong to more than one group or cluster. Each dataset has a set of membership coefficients, which depend on the degree of membership to be in a cluster. **Fuzzy C-means algorithm** is the example of this type of clustering; it is sometimes also known as the Fuzzy k-means algorithm.

Clustering Algorithms

The Clustering algorithms can be divided based on their models that are explained above. There are different types of clustering algorithms published, but only a few are commonly used. The clustering algorithm is based on the kind of data that we are using. Such as, some algorithms need to guess the number of clusters in the given dataset, whereas some are required to find the minimum distance between the observation of the dataset.

Here we are discussing mainly popular Clustering algorithms that are widely used in machine learning:

1. **K-Means algorithm:** The k-means algorithm is one of the most popular clustering algorithms. It classifies the dataset by dividing the samples into different clusters of equal variances. The number of clusters must be specified in this algorithm. It is fast with fewer computations required, with the linear complexity of **O(n).**
2. **Mean-shift algorithm:** Mean-shift algorithm tries to find the dense areas in the smooth density of data points. It is an example of a centroid-based model, that works on updating the candidates for centroid to be the center of the points within a given region.
3. **DBSCAN Algorithm:** It stands **for Density-Based Spatial Clustering of Applications with Noise**. It is an example of a density-based model similar to the mean-shift, but with

some remarkable advantages. In this algorithm, the areas of high density are separated by the areas of low density. Because of this, the clusters can be found in any arbitrary shape.

1. **Expectation-Maximization Clustering using GMM:** This algorithm can be used as an alternative for the k-means algorithm or for those cases where K-means can be failed. In GMM, it is assumed that the data points are Gaussian distributed.
2. **Agglomerative Hierarchical algorithm:** The Agglomerative hierarchical algorithm performs the bottom-up hierarchical clustering. In this, each data point is treated as a single cluster at the outset and then successively merged. The cluster hierarchy can be represented as a tree-structure.
3. **Affinity Propagation:** It is different from other clustering algorithms as it does not require to specify the number of clusters. In this, each data point sends a message between the pair of data points until convergence. It has O(N2T) time complexity, which is the main drawback of this algorithm.

Applications of Clustering

Below are some commonly known applications of clustering technique in Machine Learning:

* **In Identification of Cancer Cells:** The clustering algorithms are widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups.
* **In Search Engines:** Search engines also work on the clustering technique. The search result appears based on the closest object to the search query. It does it by grouping similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.
* **Customer Segmentation:** It is used in market research to segment the customers based on their choice and preferences.
* **In Biology:** It is used in the biology stream to classify different species of plants and animals using the image recognition technique.
* **In Land Use:** The clustering technique is used in identifying the area of similar lands use in the GIS database. This can be very useful to find that for what purpose the particular land should be used, that means for which purpose it is more suitable.

## Hierarchical Clustering in R Programming

**Hierarchical clustering in R Programming Language** is an Unsupervised non-linear algorithm in which clusters are created such that they have a hierarchy (or a pre- determined ordering). For example, consider a family of up to three generations. A grandfather and mother have their children that become father and mother of their children. So, they all are grouped together to the same family i.e they form a hierarchy.

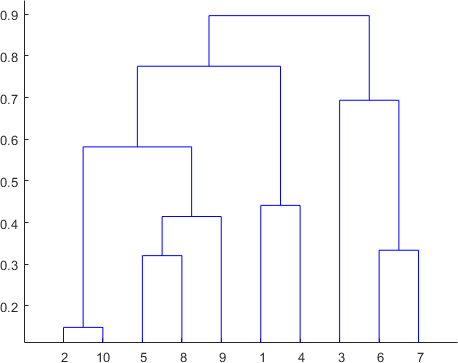
Hierarchical clustering is of two types:

* **Agglomerative Hierarchical clustering:** It starts at individual leaves and successfully merges clusters together. Its a Bottom-up approach.
* **Divisive Hierarchical clustering:** It starts at the root and recursively split the clusters. It’s a top-down approach.

## Theory:

In hierarchical clustering, Objects are categorized into a hierarchy similar to a tree- shaped structure which is used to interpret hierarchical clustering models. The algorithm is as follows:

1. Make each data point in a single point cluster that forms **N** clusters.
2. Take the two closest data points and make them one cluster that forms **N-1** clusters.
3. Take the two closest clusters and make them one cluster that forms **N-2** clusters.
4. Repeat steps 3 until there is only one cluster.



Dendrogram is a hierarchy of clusters in which distances are converted into heights. It clusters **n** units or objects each with **p** feature into smaller groups. Units in the same cluster are joined by a horizontal line. The leaves at the bottom represent individual units. It provides a visual representation of clusters. **Thumb Rule:** Largest vertical distance which doesn’t cut any horizontal line defines the optimal number of clusters.

# The Dataset

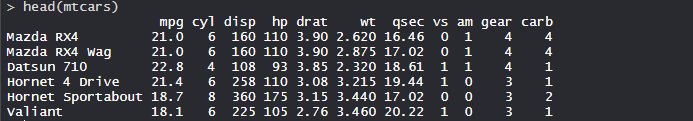
**mtcars**(motor trend car road test) comprise fuel consumption, performance, and 10 aspects of automobile design for 32 automobiles. It comes pre-installed with dplyr package in R.

# Installing the package install.packages("dplyr")

# Loading package library(dplyr)

# Summary of dataset in package head(mtcars)

**Output:**



# Performing Hierarchical clustering on Dataset

Using Hierarchical Clustering algorithm on the dataset using **hclust()** which is pre- installed in stats package when R is installed.

# Finding distance matrix

distance\_mat <- dist(mtcars, method = 'euclidean') distance\_mat

# Fitting Hierarchical clustering Model # to training dataset

set.seed(240) # Setting seed

Hierar\_cl <- hclust(distance\_mat, method = "average")

Hierar\_cl

# Plotting dendrogram plot(Hierar\_cl)

# Choosing no. of clusters # Cutting tree by height

abline(h = 110, col = "green")

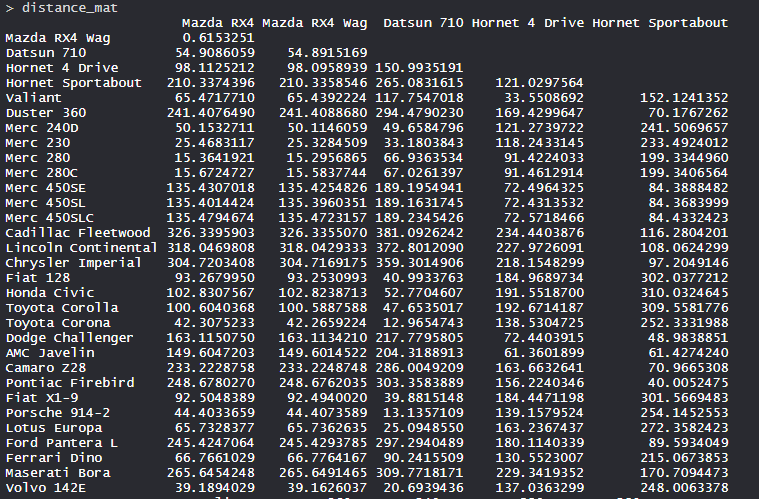
# Cutting tree by no. of clusters fit <- cutree(Hierar\_cl, k = 3 ) fit

table(fit)

rect.hclust(Hierar\_cl, k = 3, border = "green")

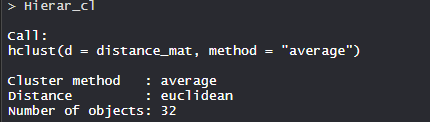
**Output:**

* **Distance matrix:**



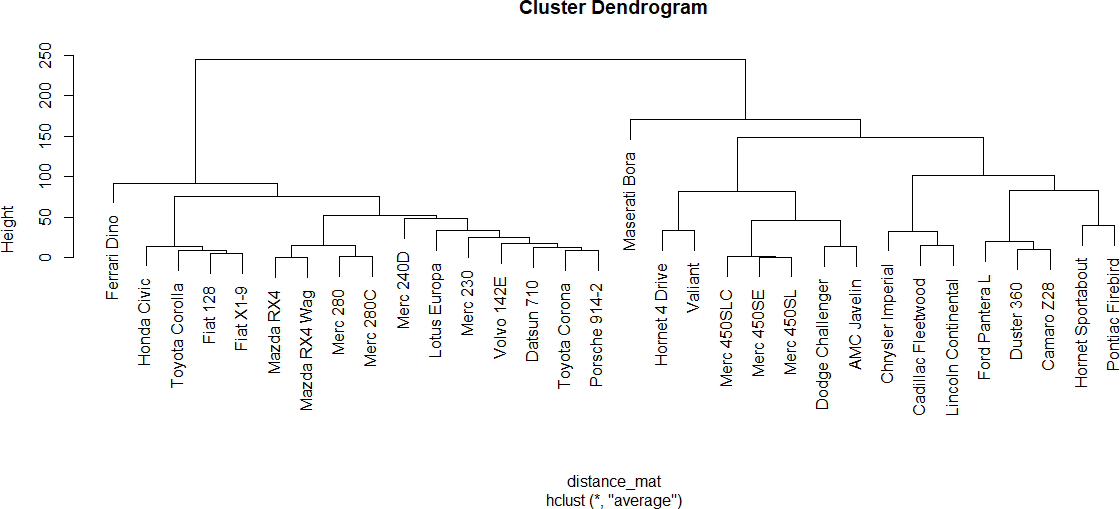
* The values are shown as per the distance matrix calculation with the method as euclidean.

### Model Hierar\_cl:



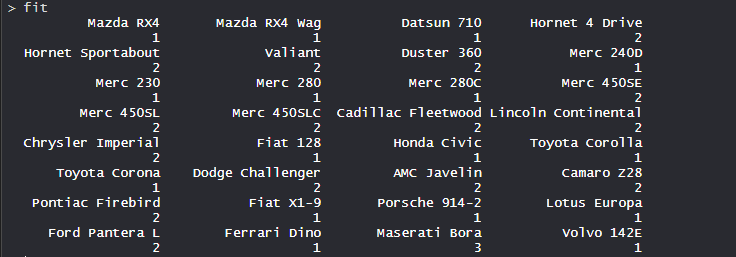
* In the model, the cluster method is average, distance is euclidean and no. of objects are 32.

### Plot dendrogram:



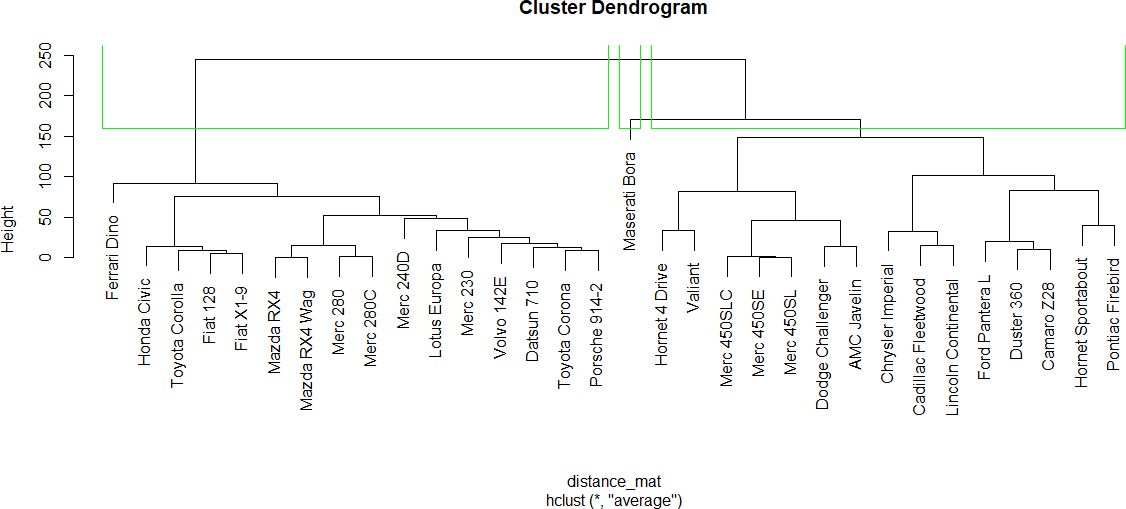
* The plot dendrogram is shown with x-axis as distance matrix and y-axis as height.

### Cutted tree:



* So, Tree is cut where k = 3 and each category represents its number of clusters.

### Plotting dendrogram after cutting:



* The plot denotes dendrogram after being cut. The green lines show the number of clusters as per the thumb rule.

**PRACTICAL 12**

**Ensemble Machine Learning in R**

You can create ensembles of machine learning algorithms in R.

There are three main techniques that you can create an ensemble of machine learning algorithms in R: Boosting, Bagging and Stacking. In this section, we will look at each in turn.Before we start building ensembles, let’s define our test set-up.

**Test Dataset**

All of the examples of ensemble predictions in this case study will use the ionosphere dataset.

This is a dataset available from the UCI Machine Learning Repository. This dataset describes high-frequency antenna returns from high energy particles in the atmosphere and whether the return shows structure or not. The problem is a binary classification that contains 351 instances and 35 numerical attributes.

Let’s load the libraries and the dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # Load libraries  library(mlbench)  library(caret)  library(caretEnsemble)    # Load the dataset  data(Ionosphere)  dataset <- Ionosphere  dataset <- dataset[,-2]  dataset$V1 <- as.numeric(as.character(dataset$V1)) |

Note that the first attribute was a factor (0,1) and has been transformed to be numeric for consistency with all of the other numeric attributes. Also note that the second attribute is a constant and has been removed.

Here is a sneak-peek at the first few rows of the ionosphere dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | > head(dataset)    V1      V3       V4       V5       V6       V7       V8      V9      V10     V11      V12     V13      V14      V15  1  1 0.99539 -0.05889  0.85243  0.02306  0.83398 -0.37708 1.00000  0.03760 0.85243 -0.17755 0.59755 -0.44945  0.60536  2  1 1.00000 -0.18829  0.93035 -0.36156 -0.10868 -0.93597 1.00000 -0.04549 0.50874 -0.67743 0.34432 -0.69707 -0.51685  3  1 1.00000 -0.03365  1.00000  0.00485  1.00000 -0.12062 0.88965  0.01198 0.73082  0.05346 0.85443  0.00827  0.54591  4  1 1.00000 -0.45161  1.00000  1.00000  0.71216 -1.00000 0.00000  0.00000 0.00000  0.00000 0.00000  0.00000 -1.00000  5  1 1.00000 -0.02401  0.94140  0.06531  0.92106 -0.23255 0.77152 -0.16399 0.52798 -0.20275 0.56409 -0.00712  0.34395  6  1 0.02337 -0.00592 -0.09924 -0.11949 -0.00763 -0.11824 0.14706  0.06637 0.03786 -0.06302 0.00000  0.00000 -0.04572         V16      V17      V18      V19      V20      V21      V22      V23      V24      V25      V26      V27      V28  1 -0.38223  0.84356 -0.38542  0.58212 -0.32192  0.56971 -0.29674  0.36946 -0.47357  0.56811 -0.51171  0.41078 -0.46168  2 -0.97515  0.05499 -0.62237  0.33109 -1.00000 -0.13151 -0.45300 -0.18056 -0.35734 -0.20332 -0.26569 -0.20468 -0.18401  3  0.00299  0.83775 -0.13644  0.75535 -0.08540  0.70887 -0.27502  0.43385 -0.12062  0.57528 -0.40220  0.58984 -0.22145  4  0.14516  0.54094 -0.39330 -1.00000 -0.54467 -0.69975  1.00000  0.00000  0.00000  1.00000  0.90695  0.51613  1.00000  5 -0.27457  0.52940 -0.21780  0.45107 -0.17813  0.05982 -0.35575  0.02309 -0.52879  0.03286 -0.65158  0.13290 -0.53206  6 -0.15540 -0.00343 -0.10196 -0.11575 -0.05414  0.01838  0.03669  0.01519  0.00888  0.03513 -0.01535 -0.03240  0.09223         V29      V30      V31      V32      V33      V34 Class  1  0.21266 -0.34090  0.42267 -0.54487  0.18641 -0.45300  good  2 -0.19040 -0.11593 -0.16626 -0.06288 -0.13738 -0.02447   bad  3  0.43100 -0.17365  0.60436 -0.24180  0.56045 -0.38238  good  4  1.00000 -0.20099  0.25682  1.00000 -0.32382  1.00000   bad  5  0.02431 -0.62197 -0.05707 -0.59573 -0.04608 -0.65697  good  6 -0.07859  0.00732  0.00000  0.00000 -0.00039  0.12011   bad |

**1. Boosting Algorithms**

We can look at two of the most popular boosting machine learning algorithms:

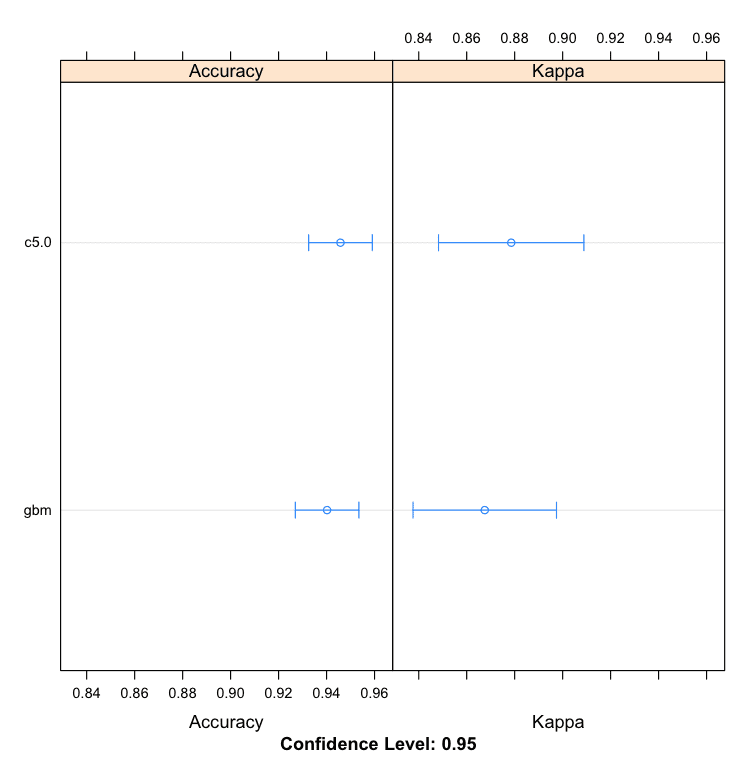
* C5.0
* Stochastic Gradient Boosting

Below is an example of the C5.0 and Stochastic Gradient Boosting (using the Gradient Boosting Modeling implementation) algorithms in R. Both algorithms include parameters that are not tuned in this example.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # Example of Boosting Algorithms  control <- trainControl(method="repeatedcv", number=10, repeats=3)  seed <- 7  metric <- "Accuracy"  # C5.0  set.seed(seed)  fit.c50 <- train(Class~., data=dataset, method="C5.0", metric=metric, trControl=control)  # Stochastic Gradient Boosting  set.seed(seed)  fit.gbm <- train(Class~., data=dataset, method="gbm", metric=metric, trControl=control, verbose=FALSE)  # summarize results  boosting\_results <- resamples(list(c5.0=fit.c50, gbm=fit.gbm))  summary(boosting\_results)  dotplot(boosting\_results) |

We can see that the C5.0 algorithm produces a more accurate model with an accuracy of 94.58%.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Models: c5.0, gbm  Number of resamples: 30    Accuracy         Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's  c5.0 0.8824  0.9143 0.9437 0.9458  0.9714    1    0  gbm  0.8824  0.9143 0.9429 0.9402  0.9641    1    0 |



Boosting Machine Learning Algorithms in R

**2. Bagging Algorithms**

Let’s look at two of the most popular bagging machine learning algorithms:

Bagging, or bootstrap aggregation, is an ensemble method that involves training the same algorithm many times by using different subsets sampled from the training data. The final output prediction is then averaged across the predictions of all the sub-models. The two most popular bagging ensemble techniques are Bagged Decision Trees and Random Forest.

* Bagged CART

This method performs best with algorithms that have high variance, for example, decision trees.

* Random Forest

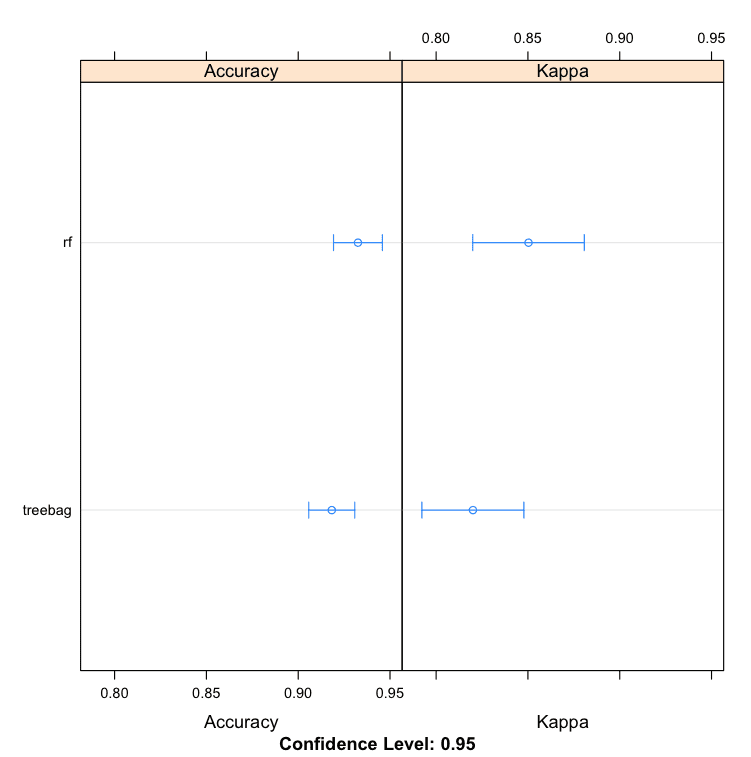
Random Forest is an extension of bagged decision trees, where the samples of the training dataset are taken with replacement. The trees are constructed with the objective of reducing the correlation between the individual decision trees.

Below is an example of the Bagged CART and Random Forest algorithms in R. Both algorithms include parameters that are not tuned in this example.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # Example of Bagging algorithms  control <- trainControl(method="repeatedcv", number=10, repeats=3)  seed <- 7  metric <- "Accuracy"  # Bagged CART  set.seed(seed)  fit.treebag <- train(Class~., data=dataset, method="treebag", metric=metric, trControl=control)  # Random Forest  set.seed(seed)  fit.rf <- train(Class~., data=dataset, method="rf", metric=metric, trControl=control)  # summarize results  bagging\_results <- resamples(list(treebag=fit.treebag, rf=fit.rf))  summary(bagging\_results)  dotplot(bagging\_results) |

We can see that random forest produces a more accurate model with an accuracy of 93.25%.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Models: treebag, rf  Number of resamples: 30    Accuracy            Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's  treebag 0.8529  0.8946 0.9143 0.9183  0.9440    1    0  rf      0.8571  0.9143 0.9420 0.9325  0.9444    1    0 |



Bagging Machine Learning Algorithms in R

**3. Stacking Algorithms**

You can combine the predictions of multiple *caret* models using the *caretEnsemble* package.

Given a list of caret models, the *caretStack()* function can be used to specify a higher-order model to learn how to best combine the predictions of sub-models together.

Let’s first look at creating 5 sub-models for the ionosphere dataset, specifically:

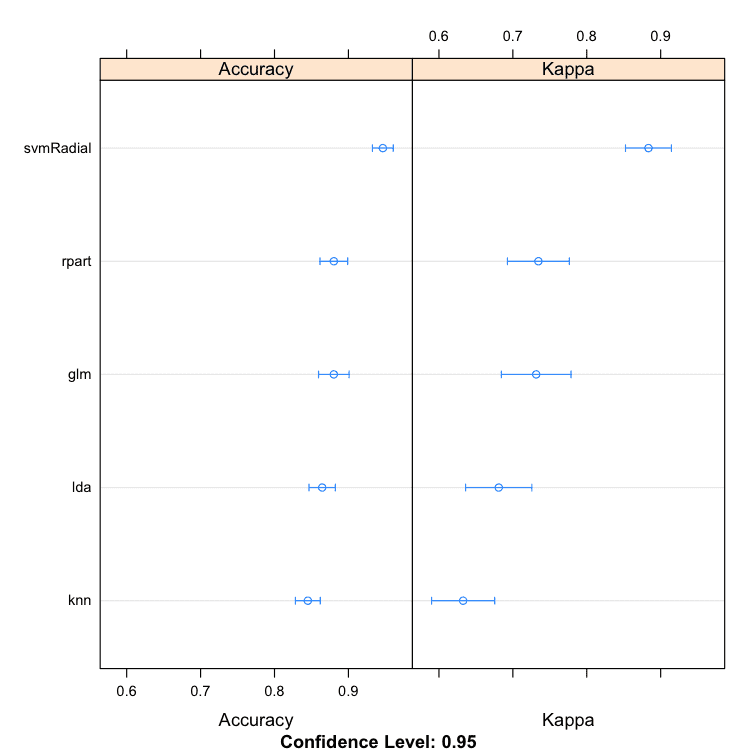
* Linear Discriminate Analysis (LDA)
* Classification and Regression Trees (CART)
* Logistic Regression (via Generalized Linear Model or GLM)
* k-Nearest Neighbors (kNN)
* Support Vector Machine with a Radial Basis Kernel Function (SVM)

Below is an example that creates these 5 sub-models. Note the new helpful *caretList()* function provided by the *caretEnsemble* package for creating a list of standard caret models.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # Example of Stacking algorithms  # create submodels  control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)  algorithmList <- c('lda', 'rpart', 'glm', 'knn', 'svmRadial')  set.seed(seed)  models <- caretList(Class~., data=dataset, trControl=control, methodList=algorithmList)  results <- resamples(models)  summary(results)  dotplot(results) |

We can see that the SVM creates the most accurate model with an accuracy of 94.66%.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Models: lda, rpart, glm, knn, svmRadial  Number of resamples: 30    Accuracy              Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's  lda       0.7714  0.8286 0.8611 0.8645  0.9060 0.9429    0  rpart     0.7714  0.8540 0.8873 0.8803  0.9143 0.9714    0  glm       0.7778  0.8286 0.8873 0.8803  0.9167 0.9722    0  knn       0.7647  0.8056 0.8431 0.8451  0.8857 0.9167    0  svmRadial 0.8824  0.9143 0.9429 0.9466  0.9722 1.0000    0 |



Comparison of Sub-Models for Stacking Ensemble in R

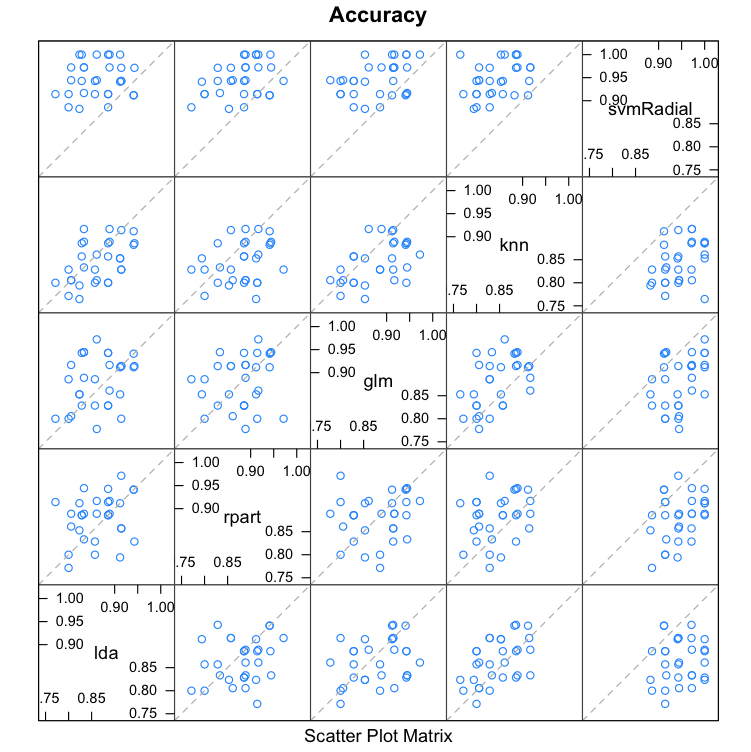
When we combine the predictions of different models using stacking, it is desirable that the predictions made by the sub-models have low correlation. This would suggest that the models are skillful but in different ways, allowing a new classifier to figure out how to get the best from each model for an improved score.

If the predictions for the sub-models were highly corrected (>0.75) then they would be making the same or very similar predictions most of the time reducing the benefit of combining the predictions.

|  |  |
| --- | --- |
| 1  2  3 | # correlation between results  modelCor(results)  splom(results) |

We can see that all pairs of predictions have generally low correlation. The two methods with the highest correlation between their predictions are Logistic Regression (GLM) and kNN at 0.517 correlation which is not considered high (>0.75).

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | lda     rpart       glm       knn svmRadial  lda       1.0000000 0.2515454 0.2970731 0.5013524 0.1126050  rpart     0.2515454 1.0000000 0.1749923 0.2823324 0.3465532  glm       0.2970731 0.1749923 1.0000000 0.5172239 0.3788275  knn       0.5013524 0.2823324 0.5172239 1.0000000 0.3512242  svmRadial 0.1126050 0.3465532 0.3788275 0.3512242 1.0000000 |



Correlations Between Predictions Made By Sub-Models in Stacking Ensemble

Let’s combine the predictions of the classifiers using a simple linear model.

|  |  |
| --- | --- |
| 1  2  3  4  5 | # stack using glm  stackControl <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)  set.seed(seed)  stack.glm <- caretStack(models, method="glm", metric="Accuracy", trControl=stackControl)  print(stack.glm) |

We can see that we have lifted the accuracy to 94.99% which is a small improvement over using SVM alone. This is also an improvement over using random forest alone on the dataset, as observed above.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | A glm ensemble of 2 base models: lda, rpart, glm, knn, svmRadial    Ensemble results:  Generalized Linear Model    1053 samples     5 predictor     2 classes: 'bad', 'good'    No pre-processing  Resampling: Cross-Validated (10 fold, repeated 3 times)  Summary of sample sizes: 948, 947, 948, 947, 949, 948, ...  Resampling results      Accuracy  Kappa     Accuracy SD  Kappa SD    0.949996  0.891494  0.02121303   0.04600482 |

We can also use more sophisticated algorithms to combine predictions in an effort to tease out when best to use the different methods. In this case, we can use the random forest algorithm to combine the predictions.

|  |  |
| --- | --- |
| 1  2  3  4 | # stack using random forest  set.seed(seed)  stack.rf <- caretStack(models, method="rf", metric="Accuracy", trControl=stackControl)  print(stack.rf) |

We can see that this has lifted the accuracy to 96.26% an impressive improvement on SVM alone.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | A rf ensemble of 2 base models: lda, rpart, glm, knn, svmRadial    Ensemble results:  Random Forest    1053 samples     5 predictor     2 classes: 'bad', 'good'    No pre-processing  Resampling: Cross-Validated (10 fold, repeated 3 times)  Summary of sample sizes: 948, 947, 948, 947, 949, 948, ...  Resampling results across tuning parameters:      mtry  Accuracy   Kappa      Accuracy SD  Kappa SD    2     0.9626439  0.9179410  0.01777927   0.03936882    3     0.9623205  0.9172689  0.01858314   0.04115226    5     0.9591459  0.9106736  0.01938769   0.04260672    Accuracy was used to select the optimal model using  the largest value.  The final value used for the model was mtry = 2. |