**Week-3**

**1**. . **Implement R Script to create a list.**

# Create a list containing strings, numbers, vectors and a logical

# values.

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

print(list\_data)

output:

[1] "Red"

[1] "Green"

[1] 21 32 11

[1] TRUE

[1] 51.23

[1] 119

**2. Implement R Script to access elements in the list.**

x <- list(TRUE, 25, "Apple")

print(x[1])

print(x[2])

print(x[3])

**output:**

|  |
| --- |
| [[1]]  [1] TRUE    [[1]]  [1] 25    [[1]]  [1] "Apple" |

**3** **. Implement R Script to perform matrix operation**.

B <- matrix( c(2, 4, 3, 1, 5, 7), nrow=3, ncol=2)

D <- matrix( c(1, 3, 2), nrow=3, ncol=1)

cbind(B,D)

**output:**

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

[1,] 2 1 1

[2,] 4 5 3

[3,] 3 7 2

**Week-4**

1. **Implement R script to perform following operations: various operations on vectors**

**program:-**

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

print('using c function')

print(x)

print('using seq() function')

print(seq(x))

print('using colon')

print(5:10)

print('using Subscript operator')

print(x[2])

print('using c function')

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

print('Logical indexing')

print(x[x>3])

X <- NULL

print('Deleted vector')

print(X)

y<- c(7, 9, 1, 5, 2, 1)

print('Addition')

print(x+y)

print('Subtraction')

print(x-y)

print('Multiplication')

print(x\*y)

print('Division')

print(x/y)

print('sorting done in ascending order')

print(sort(x))

print('sorting done in descending order')

print(sort(x, decreasing = TRUE))

**output:-**

"using c function"

[1] 1 4 5 2 6 7

"using seq() function"

[1] 1 2 3 4 5 6

"using colon"

[1] 5 6 7 8 9 10

"using Subscript operator"

[1] 4

"using c function"

[1] 2 1

"Logical indexing"

[1] 4 5 6 7

"Deleted vector"

NULL

"Addition"

[1] 8 13 6 7 8 8

"Subtraction"

[1] -6 -5 4 -3 4 6

"Multiplication"

[1] 7 36 5 10 12 7

"Division"

[1] 0.1428571 0.4444444 5.0000000 0.4000000 3.0000000 7.0000000

"sorting done in ascending order"

[1] 1 2 4 5 6 7

"sorting done in descending order"

[1] 7 6 5 4 2 1

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

We use the sum() and mean() function to find sum and average value respectively.

**Program:-**

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

print(sum(x))

print(mean(x))

**output:-**

[1] 25

[1] 4.166667

**3) To display elements of list in reverse order.**

by using rev function we can get list in reverse order

**program:-**

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

print(rev(1:5))

**output:-**

[1] 5 4 3 2 1

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

We use the min() and max() function to find minimum and maximum value respectively.

**program:-**

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

print(min(x))

print(max(x))

**output:**-

[1] 1

[1] 7

**Week 5**

* **Implement R Script to perform various operations on matrices**

**Program:-**

R **<-** matrix(c(5:16), nrow = 4,ncol=3)

S **<-** matrix(c(1:12), nrow = 4,ncol=3)

#Addition

sum**<-R**+S

print(sum)

#Subtraction

sub**<-R-S**

print(sub)

#Multiplication

mul**<-R**\*S

print(mul)

#Multiplication by constant

mul1**<-R**\*12

print(mul1)

#Division

div**<-R**/S

print(div)

**Output:-**

#addition

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

[1,] 6 14 22

[2,] 8 16 24

[3,] 10 18 26

[4,] 12 20 28

#subtraction

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

[1,] 4 4 4

[2,] 4 4 4

[3,] 4 4 4

[4,] 4 4 4

#Multiplication

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

[1,] 5 45 117

[2,] 12 60 140

[3,] 21 77 165

[4,] 32 96 192

#Multiplication by constant

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

[1,] 60 108 156

[2,] 72 120 168

[3,] 84 132 180

[4,] 96 144 192

#Division

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

[1,] 5.000000 1.800000 1.444444

[2,] 3.000000 1.666667 1.400000

[3,] 2.333333 1.571429 1.363636

[4,] 2.000000 1.500000 1.333333

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

Program :-

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

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

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

stringsAsFactors = FALSE

)

# Extract Specific columns.

result <- data.frame(emp.data$emp\_name,emp.data$salary)

print(result)

**Output:-**

Emp data.emp\_name emp.data.salary

1 Rick 623.30

2 Dan 515.20

3 Michelle 611.00

4 Ryan 729.00

5 Gary 843.25

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

**Program:-**

dir.create("newdir")

newDirPath <- "newdir"

files <- c("a.txt")

file.create(files)

newFilePath <- "a.txt"

file.copy(newFilePath, newDirPath)

**Output:-**

TRUE

TRUE

**WEEK-7**

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

**program:-**

# Read a txt file

my\_data <- read.delim(file.choose())

# Read a csv file

my\_data <- read.csv(file.choose())

output:-

source("C:/Users/CSM - LAB/Documents/7a.r")

7(b)Reading Excel data sheet in R.

program:-

library(readxl)

Data\_gfg <- read\_excel("Data\_gfg.xlsx")

Data\_gfg

output:-

>Data\_gfg

# A tibble: 10 x2

x y

<dbl> <dbl>

1 8 3

2 7 9

3 1 6

4 2 8

5 6 3

6 2 5

7 4 10

8 7 9

9 9 5

10 4 3

7(c)Reading XML dataset in R

program:-

# 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)

output:-

[1] 8

WEEK-8

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

ggplot2 is an R package which is designed especially for data visualization and providing best exploratory data analysis. It provides beautiful, hassle-free plots that take care of minute details like drawing legends and representing them. The plots can be created iteratively and edited later.

**Pie chart:**

Pie chart is a circular chart divided into different segments according to the ratio of data provided. The total value of the pie is 100 and the segments tell the fraction of the whole pie. It is another method to represent statistical data in graphical form and **pie()** function is used to perform the same.

***Syntax:****pie(x, labels, col, main, radius)*

**Program:**

# defining vector x with number of articles

x <- c(210, 450, 250, 100, 50, 90)

# defining labels for each value in x

names(x) <- c("Algo", "DS", "Java", "C", "C++", "Python")

# output to be present as PNG file

png(file = "piechart.png")

# creating pie chart

pie(x, labels = names(x), col = "white",

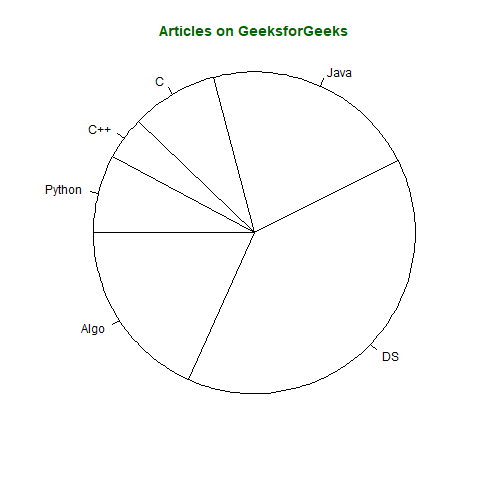
main = "Articles on GeeksforGeeks", radius = -1,

col.main = "darkgreen")

# saving the file

dev.off()

**output:**

**Bar Chart:** 

Bar plot or Bar Chart in R is used to represent the values in data vector as height of the bars. The data vector passed to the function is represented over y-axis of the graph. Bar chart can behave like histogram by using **table()** function instead of data vector.

***Syntax:****barplot(data, xlab, ylab)*

**Program:**

# defining vector

x <- c(7, 15, 23, 12, 44, 56, 32)

# output to be present as PNG file

png(file = "barplot.png")

# plotting vector

barplot(x, xlab = "GeeksforGeeks Audience",

        ylab = "Count", col = "white",

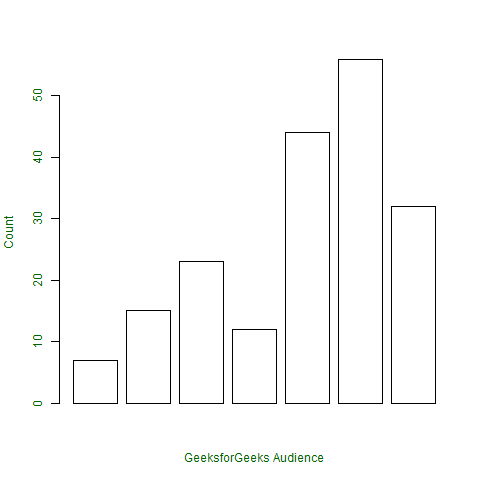
        col.axis = "darkgreen",

        col.lab = "darkgreen")

# saving the file

dev.off()

**Output:**



**scatter plot:**

A Scatter plot is another type of graphical representation used to plot the points to show relationship between two data vectors. One of the data vectors is represented on x-axis and another on y-axis.

***Syntax:****plot(x, y, type, xlab, ylab, main)*

***Where,***

* ***x****is the data vector represented on x-axis*
* ***y****is the data vector represented on y-axis*
* ***type****specifies the type of plot to be drawn. For example, “l” for lines, “p” for points, “s” for stair steps, etc.*
* ***xlab****specifies the label for x-axis*
* ***ylab****specifies the label for y-axis*
* ***main****specifies the title name of the graph*

**Program:** # taking input from dataset Orange already

# present in R

orange <- Orange[, c('age', 'circumference')]

# output to be present as PNG file

png(file = "plot.png")

# plotting

plot(x = orange$age, y = orange$circumference, xlab = "Age",

ylab = "Circumference", main = "Age VS Circumference",

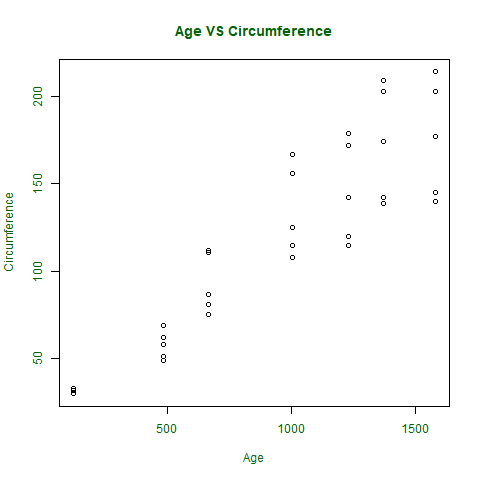
col.lab = "darkgreen", col.main = "darkgreen",

col.axis = "darkgreen")

# saving the file

dev.off()

**Output:**



**Histogram:**

Histogram is a graphical representation used to create a graph with bars representing the frequency of grouped data in vector. Histogram is same as bar chart but only difference between them is histogram represents frequency of grouped data rather than data itself.

***Syntax:****hist(x, col, border, main, xlab, ylab)*

***where:***

* ***x****is data vector*
* ***col****specifies the color of the bars to be filled*
* ***border****specifies the color of border of bars*
* ***main****specifies the title name of histogram*
* ***xlab****specifies the x-axis label*
* ***ylab****specifies the y-axis label*

**Program:**

# defining vector

x <- c(21, 23, 56, 90, 20, 7, 94, 12,

    57, 76, 69, 45, 34, 32, 49, 55, 57)

# output to be present as PNG file

png(file = "hist.png")

# hist(x, main = "Histogram of Vector x",

        xlab = "Values",

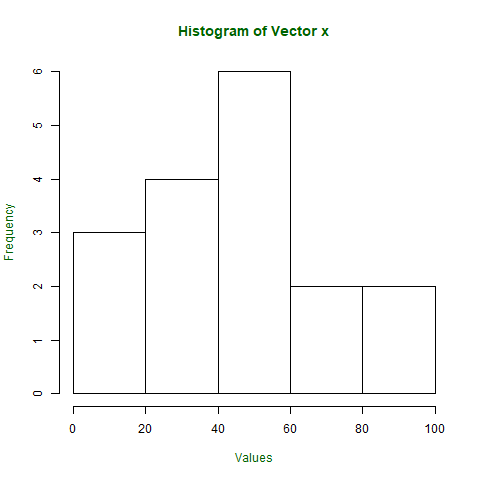
        col.lab = "darkgreen",

        col.main = "darkgreen")

# saving the file

dev.off()

**Output:**



**Week-9**

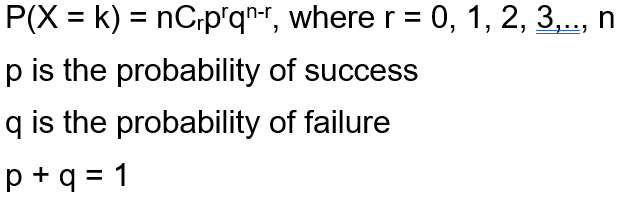
1. Implement R Script to perform Normal, Binomial distributions.

**Answer**:-

Binonal Distribution:

Binomial distribution in [R](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is a probability distribution used in statistics. The binomial distribution is a discrete distribution and has only two outcomes i.e. success or failure.

**Formula:**



**FUNCTIONS:-**

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():-

Program:

dbinom(3, size = 13, prob = 1 / 6)

probabilities <- dbinom(x = c(0:10), size = 10, prob = 1 / 6)

data.frame(x, probs)

plot(0:10, probabilities, type = "l")

Output:-



> dbinom(3, size = 13, prob = 1/6)

[1] 0.2138454

> probabilities = dbinom(x = c(0:10), size = 10, prob = 1/6)

> data.frame(probabilities)

probabilities

1 1.615056e-01

2 3.230112e-01

3 2.907100e-01

4 1.550454e-01

5 5.426588e-02

6 1.302381e-02

7 2.170635e-03

8 2.480726e-04

9 1.860544e-05

10 8.269086e-07

11 1.653817e-08

#### pbinom():-

**Program**:-

pbinom(3, size = 13, prob = 1 / 6)

plot(0:10, pbinom(0:10, size = 10, prob = 1 / 6), type = "l")

**Output**:-



> pbinom(3, size = 13, prob = 1/6)

[1] 0.8419226

#### qbinom():-

**Program:-**

qbinom(0.8419226, size = 13, prob = 1 / 6)

x <- seq(0, 1, by = 0.1)

y <- qbinom(x, size = 13, prob = 1 / 6)

plot(x, y, type = 'l')

**Output:-**

****

> qbinom(0.8419226, size = 13, prob = 1/6)

[1] 3

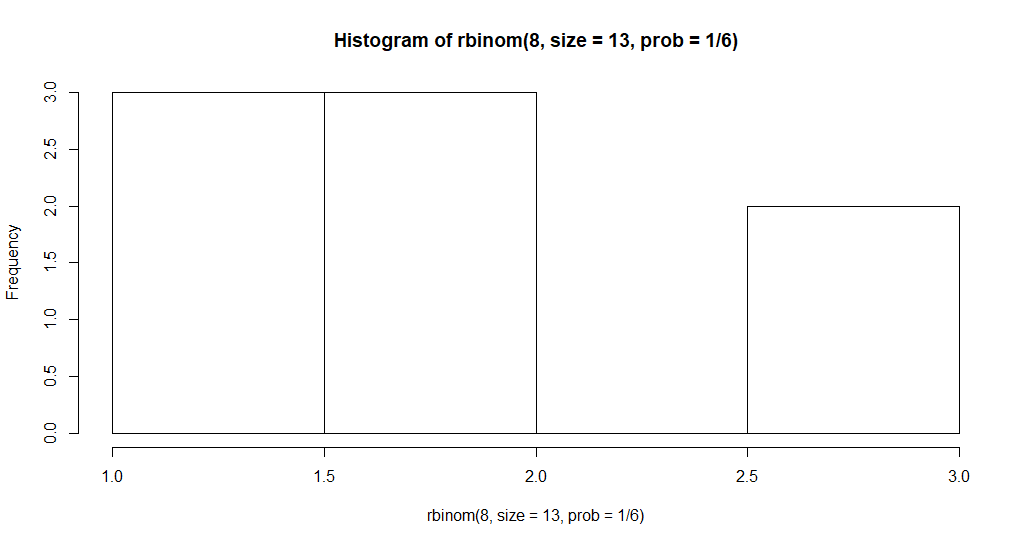
**rbinom():-**

**Program:-**

**rbinom(8, size = 13, prob = 1 / 6)**

**hist(rbinom(8, size = 13, prob = 1 / 6))**

**Output:-**



> rbinom(8, size = 13, prob = 1/6)

[1] 1 1 2 1 4 0 2 3

**Normal Distribution:**

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():-

**Program:-**

# creating a sequence of values

# between -15 to 15 with a difference of 0.1

x = seq(-15, 15, by=0.1)

y = dnorm(x, mean(x), sd(x))

# output to be present as PNG file

png(file="dnormExample.png")

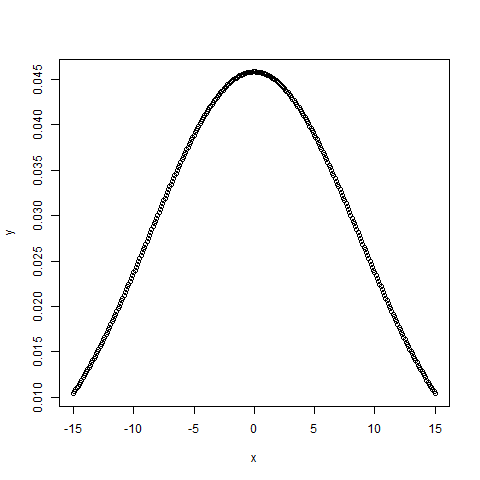
# Plot the graph.

plot(x, y)

# saving the file

dev.off()

Output:-



pnorm():-

Program:-

# creating a sequence of values

# between -10 to 10 with a difference of 0.1

x <- seq(-10, 10, by=0.1)

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

# output to be present as PNG file

png(file="pnormExample.png")

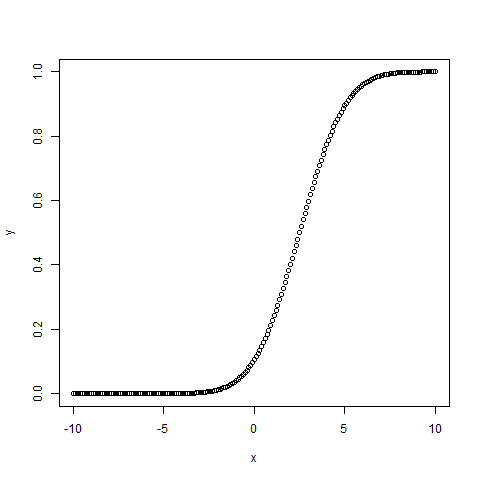
# Plot the graph.

plot(x, y)

# saving the file

dev.off()

Output:-



**rnorm():-**

Program:-

# Create a vector of 1000 random numbers

# with mean=90 and sd=5

x <- rnorm(10000, mean=90, sd=5)

# output to be present as PNG file

png(file = "rnormExample.png")

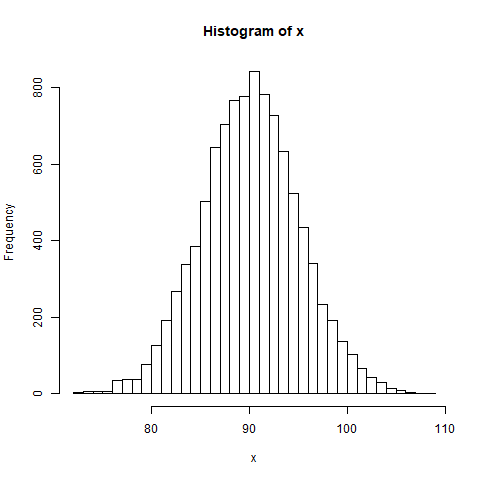
# Create the histogram with 50 bars

hist(x, breaks=50)

# Save the file.

dev.off()

Output:-



qnorm():-

Program:-

# Create a sequence of probability values

# incrementing by 0.02.

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

y <- qnorm(x, mean(x), sd(x))

# output to be present as PNG file

png(file = "qnormExample.png")

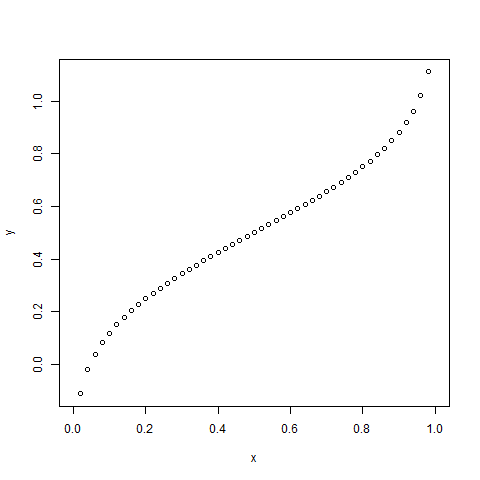
# Plot the graph.

plot(x, y)

# Save the file.

dev.off()

Output:-



1. Implement R Script to perform correlation, Linear and multiple regression:

**Pearson Correlation:-**

[**Correlation**](https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/) is a statistical measure that indicates how strongly two variables are related. It involves the relationship between multiple variables as well. For instance, if one is interested to know whether there is a relationship between the heights of fathers and sons, a correlation coefficient can be calculated to answer this question. Generally, it lies between -1 and +1. It is a scaled version of covariance and provides the direction and strength of a relationship.

## Pearson Correlation Testing in R

There are mainly two types of correlation:

* **Parametric Correlation** – [Pearson correlation(r)](https://www.geeksforgeeks.org/python-pearson-correlation-test-between-two-variables/): It measures a linear dependence between two variables (x and y) is known as a parametric correlation test because it depends on the distribution of the data.
* **Non-Parametric Correlation** – [Kendall(tau)](https://www.geeksforgeeks.org/python-kendall-rank-correlation-coefficient/?ref=rp) and [Spearman(rho)](https://www.geeksforgeeks.org/spearman-correlation-testing-in-r-programming/): They are rank-based correlation coefficients, are known as non-parametric correlation.

**Program:-**

**# R program to illustrate**

**# pearson Correlation Testing**

**# Using cor()**

**# Taking two numeric**

**# Vectors with same length**

**x = c(1, 2, 3, 4, 5, 6, 7)**

**y = c(1, 3, 6, 2, 7, 4, 5)**

**# Calculating**

**# Correlation coefficient**

**# Using cor() method**

**result = cor(x, y, method = "pearson")**

**# Print the result**

**cat("Pearson correlation coefficient is:", result)**

**output:-**

Pearson correlation coefficient is: 0.5357143

**Linear Regression:-**

#Creating input vector **for** lm() function

x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)

y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)

# Applying the lm() function.

relationship\_model<- lm(y~x)

#Printing the coefficient

print(relationship\_model)

Output:-

Call:

lm(formula = y ~ x)

Coefficients:

(Intercept) x

47.50833 0.07276

Multiple Regression:-

Program:-

#Creating input data.

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

# Creating the relationship model.

Model <- lm(mpg~wt+disp+hp, data = input)

# Showing the Model.

print(Model)

Output:-

Call:

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

Coefficients:

(Intercept) wt disp hp

37.105505 -3.800891 -0.000937 -0.031157

**WEEK 10**

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

**Time Series:**

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year.

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

**Program:**

stockrate <- c(480, 6813, 27466, 49287,  
7710, 96820, 96114, 236214,  
2088743, 381497, 927251,  
1407615, 1972113)  
stockrate.timeseries <- ts(stockrate,start = c(2019,1),frequency = 12)  
print(stockrate.timeseries)

output:  
Jan     Feb     Mar     Apr     May     Jun     Jul  
2019     480    6813   27466   49287    7710   96820   96114  
2020 1972113  
Aug     Sep     Oct     Nov     Dec  
2019  236214 2088743  381497  927251 1407615  
2020  
> plot(stockrate.timeseries)

output :

 **Spatial Data:** Spatial data is any type of data that directly or indirectly references a specific geographical area or location. Sometimes called geospatial data or geographic information, spatial data can also numerically represent a physical object in a geographic coordinate system.

**Program:**

name <- LETTERS[1:10]

longitude <- c(-116.7, -120.4, -116.7, -113.5, -115.5,

-120.8, -119.5, -113.7, -113.7, -110.7)

latitude <- c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9,

36.2, 39, 41.6, 36.9)

stations <- cbind(longitude, latitude)

*# Simulated rainfall data*

set.seed(0)

precip <- round((runif(length(latitude))\*10)^3)

psize <- 1 + precip/500

plot(stations, cex=psize, pch=20, col='red', main='Precipitation')

*# add names to plot*

text(stations, name, pos=4)

*# add a legend*

breaks <- c(100, 250, 500, 1000)

legend.psize <- 1+breaks/500

legend("topright", legend=breaks, pch=20, pt.cex=legend.psize, col='red', bg='gray')



**Network data**

A network is represented as a graph, which shows links (if any) between each vertex (or node) and its neighbors.

A line indicating a link between vertices is called an edge.

A group of vertices that are mutually reachable by following edges on the graph is called a component.

The edges followed from one vertex to another are called a path.

**Program:**

install.packages("igraph")

library(igraph)

g <- graph\_from\_literal(1-2, 1-3, 1-7, 3-4, 2-3, 2-4, 3-5, 4-5, 4-6, 4-7, 5-6, 5-8, 6-7, 7-8)

## Create directed graphs using addition or substraction operators

dg <- graph\_from\_literal(JFK-+PEK, JFK-+CDG, PEK++CDG)

plot(g)

**Output:**



#### Converting Numeric value to a Factor

For converting a numeric into factor we use **cut()** function. **cut()** divides the range of numeric vector(assume x) which is to be converted by cutting into intervals and codes its value (x) according to which interval they fall. Level one corresponds to the leftmost, level two corresponds to the next leftmost, and so on.

**Syntax:**cut. default(x, breaks, labels = NULL, include.lowest = FALSE, right = TRUE, dig.lab = 3)

Program:

# Creating vectors

age <- c(40, 49, 48, 40, 67, 52, 53)

salary <- c(103200, 106200, 150200, 10606, 10390, 14070, 10220)

gender <- c("male", "male", "transgender", "female", "male", "female", "transgender")

# Creating data frame named employee

employee<- data.frame(age, salary, gender)

# Creating a factor corresponding to age

# with three equally spaced levels

wfact = cut(employee$age, 3)

table(wfact)

**output:**

wfact

(40,49] (49,58] (58,67]

4 2 1

## Get Current System Date, and Time in R

In R, we use Sys.Date(),  Sys.time() to get the current date and time respectively based on the local system. For example,

# get current system date

Sys.Date()

# get current system time

Sys.time()

Output:

[1] "2023-03-15"

[1] "2023-03-15 15:15:19 IST"

**String Parsing:**

**strsplit() method in**R programing is used to split the string by using a delimiter.

**Syntax:** strsplit(string, split, fixed)

**Parameters:**

**string:**Input vector or string.

**split:**It is a character of string to being split.

**fixed:**Match the split or use the regular expression.

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

**Program:**

rp<-"R programming"

answer<-strsplit(rp, " ")

print(answer)

**Output:**

[[1]]

[1] "R" "programming"

**Geocoding:**

“**Geocoding** is the process of transforming a description of a location (such as an address, name of a place, or coordinates) to a location on the earth’s surface.”

“A **geocoder** is a piece of software or a (web) service that implements a geocoding process i.e. a set of inter-related components in the form of operations, algorithms, and data sources that work together to produce a spatial representation for descriptive locational references.”

“**Reverse geocoding** uses geographic coordinates to find a description of the location, most typically a postal address or place name.” (I rarely have needed to do this.)[1](https://cengel.github.io/rspatial/5_Geocoding.nb.html#fn1)

**Program:**

bee\_data <- data.frame(location = c("Alvin, TX, USA", "Alvin, TX, USA","Saratoga, CA, USA", "Athens, GA, USA", "Athens, GA, USA", "Athens, GA, USA","San Jose, CA, USA", "New Orleans, LA, USA", "Des Moines, IA, USA", "Keene, NH, USA","Keene, NH, USA"))

# get the distinct locations from the dataset and create a data frame

locations\_txt <- distinct(bee\_data, location)

locations\_txt <- as.data.frame(locations\_txt)

head(locations\_txt)

**Week 11**

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

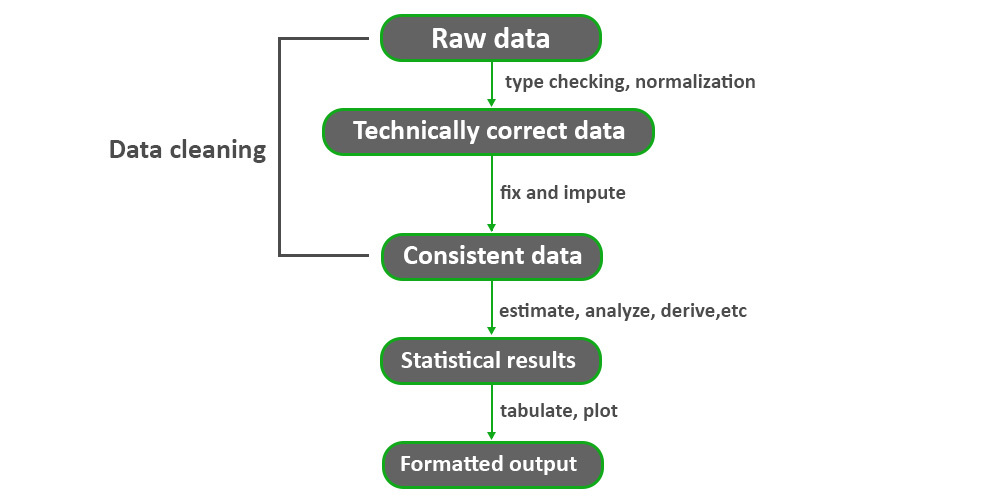
**Data Cleaning in R**

Data Cleaning is the process to transform raw data into consistent data that can be easily analyzed. It is aimed at filtering the content of statistical statements based on the data as well as their reliability. Moreover, it influences the statistical statements based on the data and improves your data quality and overall productivity.

**Purpose of Data Cleaning**

The following are the various purposes of data cleaning:

* Eliminate Errors
* Eliminate Redundancy
* Increase Data Reliability
* Delivery Accuracy
* Ensure Consistency
* Assure Completeness
* Standardize your approach



## Implementation of Data Cleaning in R

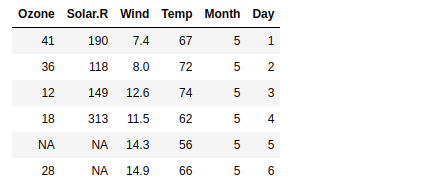
For this, we will use inbuilt datasets (air quality datasets) which are available in R.

https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781787284395/2/ch02lvl1sec22/using-air-quality-dataset

**Program:**

head(airquality)

**output:**

****

## Handling missing value in R:

To handle the missing value we will check the columns of the datasets, if we found some missing data inside the columns then this generates the NA values as an output, which can be not good for every model. So let’s check it using [mean()](https://www.geeksforgeeks.org/calculate-arithmetic-mean-in-r-programming-mean-function/) methods.

**Program:**

mean(airquality$Solar.R)

**Output:**<NA>

**Checking in another column:**

mean(airquality$Ozone)

**Output: <NA>**

**Checking in another column:**

mean(airquality$Wind)

**Output:** 9.95751633986928

**Data manipulation:**

In order to manipulate the data, R provides a library called dplyr which consists of many built-in methods to manipulate the data. So to use the data manipulation function, first need to import the dplyr package using ***library(dplyr)*** line of code. Below is the list of a few data manipulation functions present in dplyr package.

|  |  |
| --- | --- |
| filter() | Produces a subset of a Data Frame. |
| distinct() | Removes duplicate rows in a Data Frame |
| arrange() | Records the rows of a data frame |
| select() | Produces data in required columns of data frame |
| rename() | Renames the variables names |
| mutate() | Creates new variables without dropping old ones. |
| transmute() | Creates new variables by dropping the old. |
| summarize() | Gives summarized data like Average, Sum, etc. |

**Function Name: Description:**

## filter() method:

The filter() function is used to produce the subset of the data that satisfies the condition specified in the filter() method. In the condition, we can use conditional operators, logical operators, NA values, range operators etc. to filter out data.

**Syntax:**filter(dataframeName, condition)

## distinct() method:

The distinct() method removes duplicate rows from data frame or based on the specified columns.

**Syntax:**distinct(dataframeName, col1, col2,.., .keep\_all=TRUE)

## arrange() method:

In R, the arrange() method is used to order the rows based on a specified column.

**Syntax:**arrange(dataframeName, columnName)

## select() method:

The select() method is used to extract the required columns as a table by specifying the required column names in select() method.

**Syntax:**select(dataframeName, col1,col2,…)

## rename() method

The rename() function is used to change the column names.

**Syntax:**rename(dataframeName, newName=oldName)

## mutate() & transmute() methods

These methods are used to create new variables. The mutate() function creates new variables without dropping the old ones but transmute() function drops the old variables and creates new variables.

**Syntax:**mutate(dataframeName, newVariable=formula)

transmute(dataframeName, newVariable=formula)

## summarize() method:

Using the summarize method we can summarize the data in the data frame by using aggregate functions like sum(), mean(), etc.

**Syntax:**summarize(dataframeName, aggregate\_function(columnName))

**Program:**

library(dplyr)

stats <- data.frame(player=c('A', 'B', 'C', 'D'),

runs=c(100, 200, 408, 19),

wickets=c(17, 20, NA, 5))

#filter() method

filter(stats, runs>100)

#distinct() method

distinct(stats)

distinct(stats, player, .keep\_all = TRUE)

#arrange() method

arrange(stats, runs)

#select()

select(stats, player,wickets)

#rename() method

rename(stats, runs\_scored=runs)

#mutate() method

mutate(stats, avg=runs/4)

# transmute() method

transmute(stats, avg=runs/4)

# summarize() method

summarize(stats, sum(runs), mean(runs))

**Output:**

player runs wickets

1 B 200 20

2 C 408 NA

Player runswickets

1 A 100 17

2 B 200 20

3 C 408 NA

4 D 19 5

player runswickets

1 A 100 17

2 B 200 20

3 C 408 NA

4 D 19 5

Playerruns wickets

1 D 19 5

2 A 100 17

3 B 200 20

4 C408 NA

player wickets

1 A 17

2 B 20

3 C NA

4 D 5

player runs\_scoredwickets

1 A 100 17

2 B 200 20

3 C 408 NA

4 D 19 5

Player runs wickets avg

1 A 100 17 25.00

2 B 200 20 50.00

3 C 408 NA 102.00

4 D 19 5 4.75

avg

1 25.00

2 50.00

3 102.00

4 4.75

sum(runs) mean(runs)

1 727 181.75

**Duplicates:**

# Identify and Remove Duplicate Data in R

A dataset can have duplicate values and to keep it redundancy-free and accurate, duplicate rows need to be identified and removed. In this article, we are going to see how to identify and remove duplicate data in R. First we will check if duplicate data is present in our data, if yes then, we will remove it.

**Syntax:**duplicated(dataframe)

**Approach**:

* Create data frame
* Pass it to duplicated() function
* This function returns the rows which are duplicated in forms of boolean values
* Apply sum function to get the number

**Program:**

# Creating a sample data frame of students

# and their marks in respective subjects.

student\_result=data.frame(name=c("Ram","Geeta","John","Paul",

                                 "Cassie","Geeta","Paul"),

                          maths=c(7,8,8,9,10,8,9),

                          science=c(5,7,6,8,9,7,8),

                          history=c(7,7,7,7,7,7,7))

# Printing data

student\_result

duplicated(student\_result)

sum(duplicated(student\_result)

**Output:**

name mathsscience history

1 Ram 7 5 7

2 Geeta 8 7 7

3 John 8 6 7

4 Paul 9 8 7

5 Cassie 10 9 7

6 Geeta 8 7 7

7 Paul 9 8 7

[1] FALSE FALSEFALSEFALSEFALSE TRUETRUE

[1] 2

**Forms of data dates:**

R programming language provides several functions that deal with date and time. These functions are used to format and convert the date from one form to another form. R provides a format function that accepts the date objects and also format parameter that allows us to specify the format of the date we needed. R provides various format specifiers which are mentioned below in table-

| **Specifier** | **Description** |
| --- | --- |
| %a | Abbreviated weekday |
| %A | Full weekday |
| %b | Abbreviated month |
| %B | Full month |
| %C | Century |
| %y | Year without century |
| %Y | Year with century |
| %d | Day of month (01-31) |
| %j | Day in Year (001-366) |
| %m | Month of year (01-12) |
| %D | Data in %m/%d/%y format |
| %u | Weekday (01-07) Starts on Monday |
|  |  |

**Weekday:**

In this, we will look into the %a, %A, and %u specifiers which give the abbreviated weekday, full weekday, and numbered weekday starting from Monday.

**Program:**

|  |
| --- |
| # today date  date<-Sys.Date()  # abbreviated month  format(date,format="%a")    # fullmonth  format(date,format="%A")    # weekday  format(date,format="%u") |

**Output**

[1] "Sat"

[1] "Saturday"

[1] "6"

[Execution complete with exit code 0]

**Date:**

Let’s look into the day, month, and year format specifiers to represent dates in different formats.

**Program:**

|  |
| --- |
| # today date  date<-Sys.Date()    # default format yyyy-mm-dd  date    # day in month  format(date,format="%d")    # month in year  format(date,format="%m")    # abbreviated month  format(date,format="%b")    # full month  format(date,format="%B")    # Date  format(date,format="%D")  format(date,format="%d-%b-%y") |

**Output**

[1] "2022-04-02"

[1] "02"

[1] "04"

[1] "Apr"

[1] "April"

[1] "04/02/22"

[1] "02-Apr-22"

[Execution complete with exit code 0]

**Year:**

We can also able to format the year in different forms. %y, %Y, and %C are the few format specifiers that return the year without century, a year with century, and century of the given date respectively.

**Program:**

|  |
| --- |
| # today date  date<-Sys.Date()    # year without century  format(date,format="%y")    # year with century  format(date,format="%Y")    # century  format(date,format="%C") |

**Output**

[1] "22"

[1] "2022"

[1] "20"

[Execution complete with exit code 0]

**Outliers:**

Outliers are data points that don’t fit the pattern of the rest of the data set. The best way to detect the outliers in the given data set is to plot the boxplot of the data set and the point located outside the box in the boxplot are all the outliers in the data set. the removal of the outliers present in the given data sets.

**Syntax:**boxplot.stats(x, coef = 1.5, do.conf = TRUE, do.out = TRUE)

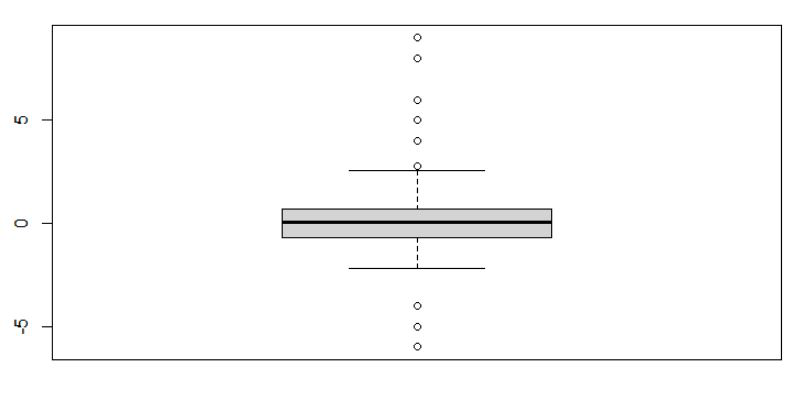
**Program:**Initial plot

gfg<-rnorm(500)

gfg[1:10]<-c(-4,2,5,6,4,1,-5,8,9,-6)

boxplot(gfg)

**Output:**

****

**Example:**Removing Outliers Using boxplot.stats() Function-

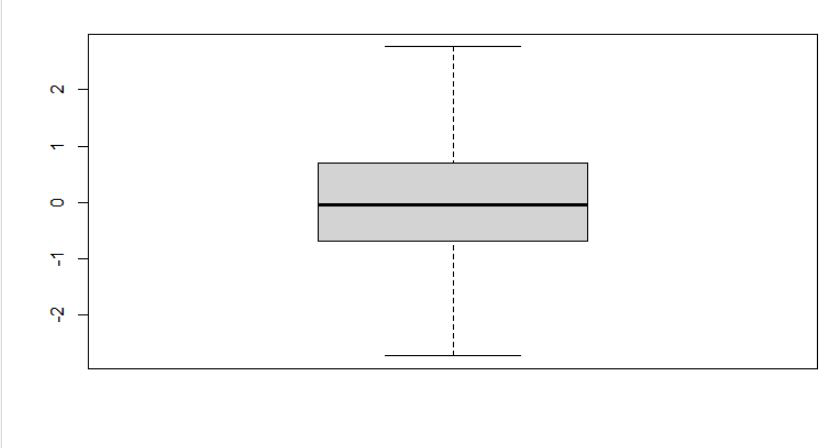
gfg<-rnorm(500)

gfg[1:10]<-c(-4,2,5,6,4,1,-5,8,9,-6)

gfg<- gfg[!gfg %in% boxplot.stats(gfg)$out]

boxplot(gfg)

**output:**



**Spell Checking Packages:**

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

spell\_check\_package()

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

The hunspell\_check and hunspell\_suggest functions can test individual words for correctness, and suggest similar (correct) words that look similar to the given (incorrect) word.

**Program:**

**library**(hunspell)

*# Check individual words*

words <- c("beer", "wiskey", "wine")

correct <- hunspell\_check(words)

print(correct)

**Output:**

[1] TRUE FALSE TRUE

**WEEK-12**

**(A)** SQLite is a relational database management system (RDBMS). It uses the same relational model that other popular DBMSs (such as [MySQL](https://www.quackit.com/mysql/), Oracle, [SQL Server](https://www.quackit.com/sql_server/), [MS Access](https://www.quackit.com/microsoft_access/)) use

* you can create multiple tables, then have them linking to each other via a relationship.
* A relationship is where you have multiple tables that contain related data, and the data is linked by a common value that is stored in both tables.

The following diagram illustrates this concept:

Diagram of a database relationship.

RSQLite is a DBI-compatible interface which means you primarily use functions defined in the DBI package, so you should always start by loading DBI, not RSQLite:

library(DBI)

**Creating a new database**

To create a new SQLite database, you simply supply the filename to dbConnect():

mydb <- dbConnect(RSQLite::SQLite(), "my-db.sqlite")

dbDisconnect(mydb)

If you just need a temporary database, use either "" (for an on-disk database) or ":memory:" or "file::memory:" (for a in-memory database). This database will be automatically deleted when you disconnect from it.

mydb <- dbConnect(RSQLite::SQLite(), "")

dbDisconnect(mydb)

**Loading data**

You can easily copy an R data frame into a SQLite database with dbWriteTable():

mydb <- dbConnect(RSQLite::SQLite(), "")

dbWriteTable(mydb, "mtcars", mtcars)

dbWriteTable(mydb, "iris", iris)

dbListTables(mydb)

*#> [1] "iris" "mtcars"*

To Create A Table :

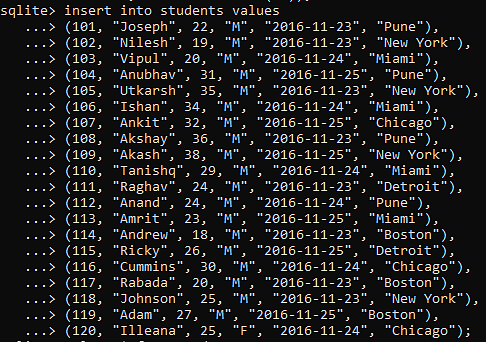
SQLiteTutorial_4.

Now, check the created table:

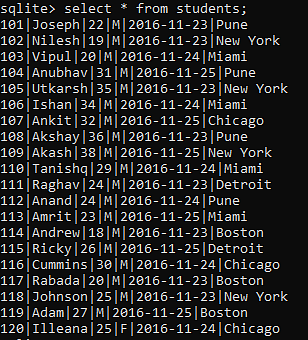
SQLiteTutorial_5.

### INSERT STATEMENT

In SQLite, the INSERT INTO statement is used to add new rows of data into a table. After you create the table, this command is used to insert records into the table.



A SELECT statement is used to fetch data from a given table.



## SQLite Commands

Now, you will look at some basic commands in this SQLite tutorial.

### CREATE TABLE STATEMENT

In SQLite, the CREATE TABLE statement is used to create a new table. While creating the table, you name that table and define its column and [data types](https://www.simplilearn.com/what-is-data-article) of each column.

Syntax:

CREATE TABLE database\_name.table\_name(

   col datatype  PRIMARY KEY(one or more columns),

   col datatype,

   col datatype,

   .....

   column datatype,

);

You will understand this command better with the help of an example.

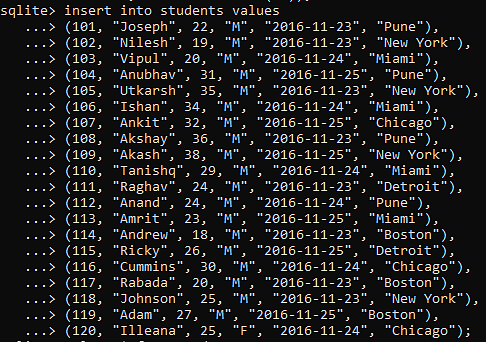
SQLiteTutorial_4.

Now, check the created table:

SQLiteTutorial_5.

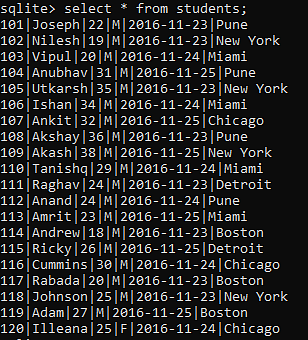
### INSERT STATEMENT

In SQLite, the INSERT INTO statement is used to add new rows of data into a table. After you create the table, this command is used to insert records into the table.



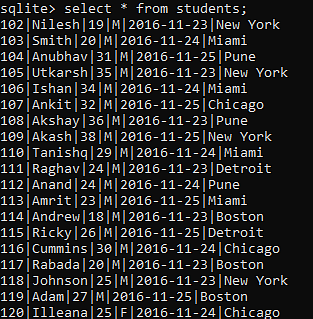
You can view the output by using the SELECT statement.

A SELECT statement is used to fetch data from a given table.

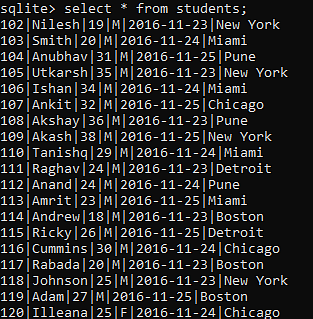


### UPDATE STATEMENT

In SQLite, the UPDATE query is used to modify the existing records in a table. It is used with the WHERE clause to select specific rows. Otherwise, it will update all the rows

The delete command is used to delete specific records (selected rows) from the table.SQLiteTutorial_10  


**DELETE STATEMENT :**

SQLiteTutorial_10

### AND OPERATIONS LIKE: AND STATEMENT, GROUP BY STATEMENT, HAVING STATEMENT, ORDER BY STATEMENT,MIN,MAX,AVF FUNCTIONS.

1. LOADING SAS:

In this script, we use the haven library to load a SAS file named data\_file.sas7bdat into R as a data frame. We use the read\_sas function to load the data, and then view the first few rows of the data using the head function and the structure of the data using the str function. Note that the haven library also provides functions for loading Stata and SPSS files in addition to SAS files.

# Load necessary library

library(haven) # for working with SAS files

# Load the SAS file

data\_sas <- read\_sas("data\_file.sas7bdat")

# View the first few rows of the data

head(data\_sas)

# View the structure of the data

str(data\_sas)

LOADING SPSS:

In this script, we use the foreign library to load an SPSS file named data\_file.sav into R as a data frame. We use the read.spss function and set the to.data.frame parameter to TRUE to ensure that the data is loaded as a data frame. We then view the first few rows of the data using the head function and the structure of the data using the str function.

# Load necessary library

library(foreign) # for working with SPSS files

# Load the SPSS file

data\_spss <- read.spss("data\_file.sav", to.data.frame = TRUE)

# View the first few rows of the data

head(data\_spss)

# View the structure of the data

str(data\_spss)

1. Reading from Google Spreadsheets, in r programming:

To read data from Google Spreadsheets in R programming, you can use the "googlesheets4" package, which is designed specifically for interacting with Google Sheets. Here are the steps to follow:

1. Install the "googlesheets4" package in R using the following command:

install.packages("googlesheets4")

1. Load the package using the following command:

library(googlesheets4)

1. Authenticate your Google account by running the following command:

gs4\_auth()

1. This will open a web page where you can authorize the package to access your Google account.
2. Get the URL or key of the Google Spreadsheet you want to read. You can do this by opening the spreadsheet in your web browser and copying the URL from the address bar, or by using the "File > Share" menu in Google Sheets to get the key.
3. Use the "read\_sheet()" function to read the data from the Google Spreadsheet into R. Here's an example:

# Replace "url\_or\_key" with the URL or key of your Google Spreadsheet

my\_data <- read\_sheet("url\_or\_key")

1. By default, this will read the first sheet in the spreadsheet. You can specify a different sheet using the "sheet" parameter:

# Replace "sheet\_name" with the name of the sheet you want to read

my\_data <- read\_sheet("url\_or\_key", sheet = "sheet\_name")

1. API and web scraping in r:

In this script, we use the httr library to make a GET request to the OpenWeatherMap API, extract the JSON response, and parse it into a list using the jsonlite library. We then print the weather data to the console.

Next, we use the rvest library to load a web page, extract the page title, and print it to the console. We then extract table data from the page using web scraping, and print the first table on the page to the console.

Program:

# Load necessary libraries

library(httr) # for making HTTP requests

library(rvest) # for web scraping

# Example of using an API

url <- "https://api.openweathermap.org/data/2.5/weather?q=London&appid=<your\_api\_key>"

response <- GET(url) # send a GET request to the API

data <- content(response, as = "text") # extract the response content

weather\_data <- jsonlite::fromJSON(data) # parse the JSON data into a list

print(weather\_data) # print the weather data

# Example of web scraping

url <- "https://en.wikipedia.org/wiki/R\_(programming\_language)"

page <- read\_html(url) # load the web page

title <- html\_text(html\_nodes(page, "title")) # extract the page title

print(title) # print the title

# Extracting table data using web scraping

table\_data <- page %>% html\_nodes("table") %>% html\_table(fill = TRUE)

print(table\_data[[1]]) # print the first table on the page