R - Next Statement

v <- LETTERS[1:6]

for ( i in v) {

if (i == "D") {

next

}

print(i)

}

[1] "A"

[1] "B"

[1] "C"

[1] "E"

[1] "F"

# R - Break Statement

v <- c("Hello","loop")

cnt <- 2

repeat {

print(v)

cnt <- cnt + 1

if(cnt > 5) {

break

}

}

[1] "Hello" "loop"

[1] "Hello" "loop"

[1] "Hello" "loop"

[1] "Hello" "loop"

# R - Bar Charts

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

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

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

Following is the description of the parameters used −

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

# R - Pie Charts

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

## **Syntax**

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

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

Following is the description of the parameters used −

* **x** is a vector containing the numeric values used in the pie chart.
* **labels** is used to give description to the slices.
* **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
* **main** indicates the title of the chart.
* **col** indicates the color palette.
* **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

# R - Histograms

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

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

## **Syntax**

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

Following is the description of the parameters used −

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

# R - Boxplots

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.

## **Syntax**

The basic syntax to create a boxplot in R is −

boxplot(x, data, notch, varwidth, names, main)

Following is the description of the parameters used −

* **x** is a vector or a formula.
* **data** is the data frame.
* **notch** is a logical value. Set as TRUE to draw a notch.
* **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
* **names** are the group labels which will be printed under each boxplot.
* **main** is used to give a title to the graph.

# R - Line Graphs

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

## **Syntax**

The basic syntax to create a line chart in R is −

plot(v,type,col,xlab,ylab)

Following is the description of the parameters used −

* **v** is a vector containing the numeric values.
* **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
* **xlab** is the label for x axis.
* **ylab** is the label for y axis.
* **main** is the Title of the chart.
* **col** is used to give colors to both the points and lines.

## **Multiple Lines in a Line Chart**

More than one line can be drawn on the same chart by using the **lines()**function.

lines(t, type = "o", col = "blue")

# R – Scatterplots

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

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

## **Syntax**

The basic syntax for creating scatterplot in R is −

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

Following is the description of the parameters used −

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

## **Scatterplot Matrices**

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

## **Scatterplot Matrices**

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

## **Mean**

It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function **mean()** is used to calculate this in R.

### **Syntax**

The basic syntax for calculating mean in R is −

mean(x, trim = 0, na.rm = FALSE, ...)

Following is the description of the parameters used −

* **x** is the input vector.
* **trim** is used to drop some observations from both end of the sorted vector.
* **na.rm** is used to remove the missing values from the input vector.

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.

## **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.

## **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)

### **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")

## **Creating a List**

Following is an example to create a list containing strings, numbers, vectors and a logical values

# 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)

## **Naming List Elements**

The list elements can be given names and they can be accessed using these names.

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

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

list("green",12.3))

# Give names to the elements in the list.

names(list\_data) <- c("1st Quarter", "A\_Matrix", "A Inner list")

# Show the list.

print(list\_data)

## **Merging Lists**

You can merge many lists into one list by placing all the lists inside one list() function.

# Create two lists.

list1 <- list(1,2,3)

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

# Merge the two lists.

merged.list <- c(list1,list2)

# Print the merged list.

print(merged.list)

## **Converting List to Vector**

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

# Create lists.

list1 <- list(1:5)

print(list1)

list2 <-list(10:14)

print(list2)

# Convert the lists to vectors.

v1 <- unlist(list1)

v2 <- unlist(list2)

print(v1)

print(v2)

# Now add the vectors

result <- v1+v2

print(result)

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

# R - Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

## **Syntax**

The basic syntax for creating a matrix in R is −

matrix(data, nrow, ncol, byrow, dimnames)

Following is the description of the parameters used −

* **data** is the input vector which becomes the data elements of the matrix.
* **nrow** is the number of rows to be created.
* **ncol** is the number of columns to be created.
* **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
* **dimname** is the names assigned to the rows and columns.

## **Example**

Create a matrix taking a vector of numbers as input

# Elements are arranged sequentially by row.

M <- matrix(c(3:14), nrow = 4, byrow = TRUE)

print(M)

# Elements are arranged sequentially by column.

N <- matrix(c(3:14), nrow = 4, byrow = FALSE)

print(N)

# Define the column and row names.

rownames = c("row1", "row2", "row3", "row4")

colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

print(P)

## **Matrix Computations**

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.

The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

### **Matrix Addition & Subtraction**

# Create two 2x3 matrices.

matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)

print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)

print(matrix2)

# Add the matrices.

result <- matrix1 + matrix2

cat("Result of addition","\n")

print(result)

# Subtract the matrices

result <- matrix1 - matrix2

cat("Result of subtraction","\n")

print(result)

### **Matrix Multiplication & Division**

# Create two 2x3 matrices.

matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)

print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)

print(matrix2)

# Multiply the matrices.

result <- matrix1 \* matrix2

cat("Result of multiplication","\n")

print(result)

# Divide the matrices

result <- matrix1 / matrix2

cat("Result of division","\n")

print(result)

# R - Arrays

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the **array()** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

## **Example**

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2))

print(result)

## **Calculations Across Array Elements**

We can do calculations across the elements in an array using the **apply()**function.

### **Syntax**

apply(x, margin, fun)

Following is the description of the parameters used −

* **x** is an array.
* **margin** is the name of the data set used.
* **fun** is the function to be applied across the elements of the array.

### **Example**

We use the apply() function below to calculate the sum of the elements in the rows of an array across all the matrices.

# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.

new.array <- array(c(vector1,vector2),dim = c(3,3,2))

print(new.array)

# Use apply to calculate the sum of the rows across all the matrices.

result <- apply(new.array, c(1), sum)

print(result)

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

, , 1

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

[1,] 5 10 13

[2,] 9 11 14

[3,] 3 12 15

, , 2

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

[1,] 5 10 13

[2,] 9 11 14

[3,] 3 12 15

[1] 56 68 60

# R - Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

Factors are created using the **factor ()** function by taking a vector as input.

# Create a vector as input.

data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)

print(is.factor(data))

# Apply the factor function.

factor\_data <- factor(data)

print(factor\_data)

print(is.factor(factor\_data))

## **Generating Factor Levels**

We can generate factor levels by using the **gl()** function. It takes two integers as input which indicates how many levels and how many times each level.

### **Syntax**

gl(n, k, labels)

Following is the description of the parameters used −

* **n** is a integer giving the number of levels.
* **k** is a integer giving the number of replications.
* **labels** is a vector of labels for the resulting factor levels.

### **Example**

v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))

print(v)

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

Tampa Tampa Tampa Tampa Seattle Seattle Seattle Seattle Boston

[10] Boston Boston Boston

Levels: Tampa Seattle Boston

# R - Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

* The column names should be non-empty.
* The row names should be unique.
* The data stored in a data frame can be of numeric, factor or character type.
* Each column should contain same number of data items.

## **Create Data Frame**

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

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

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

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

"2015-03-27")),

stringsAsFactors = FALSE

)

# Print the data frame.

print(emp.data)

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

emp\_id emp\_name salary start\_date

1 1 Rick 623.30 2012-01-01

2 2 Dan 515.20 2013-09-23

3 3 Michelle 611.00 2014-11-15

4 4 Ryan 729.00 2014-05-11

5 5 Gary 843.25 2015-03-27

## **Get the Structure of the Data Frame**

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

## **Summary of Data in Data Frame**

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

## **Extract Data from Data Frame**

Extract specific column from a data frame using column name.

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

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

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

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

"2015-03-27")),

stringsAsFactors = FALSE

)

# Extract Specific columns.

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

print(result)

## **Expand Data Frame**

A data frame can be expanded by adding columns and rows.

### **Add Column**

Just add the column vector using a new column name.

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

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

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

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

"2015-03-27")),

stringsAsFactors = FALSE

)

# Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance")

v <- emp.data

print(v)

### **Add Row**

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the **rbind()** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

# Create the first 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")),

dept = c("IT","Operations","IT","HR","Finance"),

stringsAsFactors = FALSE

)

# Create the second data frame

emp.newdata <- data.frame(

emp\_id = c (6:8),

emp\_name = c("Rasmi","Pranab","Tusar"),

salary = c(578.0,722.5,632.8),

start\_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),

dept = c("IT","Operations","Fianance"),

stringsAsFactors = FALSE

)

# Bind the two data frames.

emp.finaldata <- rbind(emp.data,emp.newdata)

print(emp.finaldata)

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

emp\_id emp\_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 5 Gary 843.25 2015-03-27 Finance

6 6 Rasmi 578.00 2013-05-21 IT

7 7 Pranab 722.50 2013-07-30 Operations

8 8 Tusar 632.80 2014-06-17 Fianance

Get all packages currently loaded in the R environment

search()

## **Get the list of all the packages installed**

library()

## **Merging Data Frames**

We can merge two data frames by using the **merge()** function. The data frames must have same column names on which the merging happens.