Mean

### **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.
* # Create a vector.
* x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
* # Find Mean.
* result.mean <- mean(x)
* print(result.mean)
* When we execute the above code, it produces the following result −
* [1] 8.22

## Applying Trim Option

* When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.
* When trim = 0.3, 3 values from each end will be dropped from the calculations to find mean.
* In this case the sorted vector is (−21, −5, 2, 3, 4.2, 7, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (−21,−5,2) from left and (12,18,54) from right.
* # Create a vector.
* x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
* # Find Mean.
* result.mean <- mean(x,trim = 0.3)
* print(result.mean)

## Applying NA Option

If there are missing values, then the mean function returns NA.

To drop the missing values from the calculation use na.rm = TRUE. which means remove the NA values.

# Create a vector.

x <- c(12,7,3,4.2,18,2,54,-21,8,-5,NA)

# Find mean.

result.mean <- mean(x)

print(result.mean)

# Find mean dropping NA values.

result.mean <- mean(x,na.rm = TRUE)

print(result.mean)

## Median

The middle most value in a data series is called the median. The **median()**function is used in R to calculate this value.

### **Syntax**

The basic syntax for calculating median in R is −

median(x, na.rm = FALSE)

Following is the description of the parameters used −

* **x** is the input vector.
* **na.rm** is used to remove the missing values from the input vector.

## Mode

The mode is the value that has highest number of occurrences in a set of data. Unike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

# Create the function.

getmode <- function(v) {

uniqv <- unique(v)

uniqv[which.max(tabulate(match(v, uniqv)))]

}

# Create the vector with numbers.

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

# Calculate the mode using the user function.

result <- getmode(v)

print(result)

# Create the vector with characters.

charv <- c("o","it","the","it","it")

# Calculate the mode using the user function.

result <- getmode(charv)

print(result)

# **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))

## 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))

# **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))

## Create Regression Model

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

family = poisson)

print(summary(output))

# **Analysis of Covariance**

We use Regression analysis to create models which describe the effect of variation in predictor variables on the response variable. Sometimes, if we have a categorical variable with values like Yes/No or Male/Female etc. The simple regression analysis gives multiple results for each value of the categorical variable. In such scenario, we can study the effect of the categorical variable by using it along with the predictor variable and comparing the regression lines for each level of the categorical variable. Such an analysis is termed as **Analysis of Covariance** also called as **ANCOVA**.

## Example

Consider the R built in data set mtcars. In it we observer that the field "am" represents the type of transmission (auto or manual). It is a categorical variable with values 0 and 1. The miles per gallon value(mpg) of a car can also depend on it besides the value of horse power("hp").

We study the effect of the value of "am" on the regression between "mpg" and "hp". It is done by using the **aov()** function followed by the **anova()** function to compare the multiple regressions.

## Input Data

Create a data frame containing the fields "mpg", "hp" and "am" from the data set mtcars. Here we take "mpg" as the response variable, "hp" as the predictor variable and "am" as the categorical variable.

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

print(head(input))

## ANCOVA Analysis

We create a regression model taking "hp" as the predictor variable and "mpg" as the response variable taking into account the interaction between "am" and "hp".

### **Model with interaction between categorical variable and predictor variable**

# Get the dataset.

input <- mtcars

# Create the regression model.

result <- aov(mpg~hp\*am,data = input)

print(summary(result))

### **Model without interaction between categorical variable and predictor variable**

# Get the dataset.

input <- mtcars

# Create the regression model.

result <- aov(mpg~hp+am,data = input)

print(summary(result))

## Comparing Two Models

Now we can compare the two models to conclude if the interaction of the variables is truly in-significant. For this we use the **anova()** function.

# Get the dataset.

input <- mtcars

# Create the regression models.

result1 <- aov(mpg~hp\*am,data = input)

result2 <- aov(mpg~hp+am,data = input)

# Compare the two models.

print(anova(result1,result2))

# **Decision Tree**

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions. It is mostly used in Machine Learning and Data Mining applications using R.

Examples of use of decision tress is − predicting an email as spam or not spam, predicting of a tumor is cancerous or predicting a loan as a good or bad credit risk based on the factors in each of these. Generally, a model is created with observed data also called training data. Then a set of validation data is used to verify and improve the model. R has packages which are used to create and visualize decision trees. For new set of predictor variable, we use this model to arrive at a decision on the category (yes/No, spam/not spam) of the data.

The R package **"party"** is used to create decision trees.

## Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

install.packages("party")

### **Syntax**

The basic syntax for creating a decision tree in R is −

ctree(formula, data)

Following is the description of the parameters used −

* **formula** is a formula describing the predictor and response variables.
* **data** is the name of the data set used.

### **Input Data**

We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

Here is the sample data.

# Load the party package. It will automatically load other

# dependent packages.

library(party)

# Print some records from data set readingSkills.

print(head(readingSkills))

# Load the party package. It will automatically load other

# dependent packages.

library(party)

# Create the input data frame.

input.dat <- readingSkills[c(1:105),]

# Give the chart file a name.

png(file = "decision\_tree.png")

# Create the tree.

output.tree <- ctree(

nativeSpeaker ~ age + shoeSize + score,

data = input.dat)

# Plot the tree.

plot(output.tree)

# Save the file.

dev.off()

# **Nonlinear Least Square**

When modeling real world data for regression analysis, we observe that it is rarely the case that the equation of the model is a linear equation giving a linear graph. Most of the time, the equation of the model of real world data involves mathematical functions of higher degree like an exponent of 3 or a sin function. In such a scenario, the plot of the model gives a curve rather than a line. The goal of both linear and non-linear regression is to adjust the values of the model's parameters to find the line or curve that comes closest to your data. On finding these values we will be able to estimate the response variable with good accuracy.

In Least Square regression, we establish a regression model in which the sum of the squares of the vertical distances of different points from the regression curve is minimized. We generally start with a defined model and assume some values for the coefficients. We then apply the **nls()** function of R to get the more accurate values along with the confidence intervals.

## Syntax

The basic syntax for creating a nonlinear least square test in R is −

nls(formula, data, start)

Following is the description of the parameters used −

* **formula** is a nonlinear model formula including variables and parameters.
* **data** is a data frame used to evaluate the variables in the formula.
* **start** is a named list or named numeric vector of starting estimates.

## Example

We will consider a nonlinear model with assumption of initial values of its coefficients. Next we will see what is the confidence intervals of these assumed values so that we can judge how well these values fir into the model.

So let's consider the below equation for this purpose −

a = b1\*x^2+b2

xvalues <- c(1.6,2.1,2,2.23,3.71,3.25,3.4,3.86,1.19,2.21)

yvalues <- c(5.19,7.43,6.94,8.11,18.75,14.88,16.06,19.12,3.21,7.58)

# Give the chart file a name.

png(file = "nls.png")

# Plot these values.

plot(xvalues,yvalues)

# Take the assumed values and fit into the model.

model <- nls(yvalues ~ b1\*xvalues^2+b2,start = list(b1 = 1,b2 = 3))

# Plot the chart with new data by fitting it to a prediction from 100 data points.

new.data <- data.frame(xvalues = seq(min(xvalues),max(xvalues),len = 100))

lines(new.data$xvalues,predict(model,newdata = new.data))

# Save the file.

dev.off()

# Get the sum of the squared residuals.

print(sum(resid(model)^2))

# Get the confidence intervals on the chosen values of the coefficients.

print(confint(model))

# **Time Series Analysis**

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

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

### **Syntax**

The basic syntax for **ts()** function in time series analysis is −

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

Following is the description of the parameters used −

* **data** is a vector or matrix containing the values used in the time series.
* **start** specifies the start time for the first observation in time series.
* **end** specifies the end time for the last observation in time series.
* **frequency** specifies the number of observations per unit time.

# Get the data points in form of a R vector.

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

# Convert it to a time series object.

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

# Print the timeseries data.

print(rainfall.timeseries)

# Give the chart file a name.

png(file = "rainfall.png")

# Plot a graph of the time series.

plot(rainfall.timeseries)

# Save the file.

dev.off()

## Different Time Intervals

The value of the **frequency** parameter in the ts() function decides the time intervals at which the data points are measured. A value of 12 indicates that the time series is for 12 months. Other values and its meaning is as below −

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

## Multiple Time Series

We can plot multiple time series in one chart by combining both the series into a matrix.

# Get the data points in form of a R vector.

rainfall1 <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

rainfall2 <-

c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.8)

# Convert them to a matrix.

combined.rainfall <- matrix(c(rainfall1,rainfall2),nrow = 12)

# Convert it to a time series object.

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

# Print the timeseries data.

print(rainfall.timeseries)

# Give the chart file a name.

png(file = "rainfall\_combined.png")

# Plot a graph of the time series.

plot(rainfall.timeseries, main = "Multiple Time Series")

# Save the file.

dev.off()

# **Random Forest**

the random forest approach, a large number of decision trees are created. Every observation is fed into every decision tree. The most common outcome for each observation is used as the final output. A new observation is fed into all the trees and taking a majority vote for each classification model.

An error estimate is made for the cases which were not used while building the tree. That is called an **OOB (Out-of-bag)** error estimate which is mentioned as a percentage.

The R package **"randomForest"** is used to create random forests.

## Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

install.packages("randomForest)

### **Syntax**

The basic syntax for creating a random forest in R is −

randomForest(formula, data)

Following is the description of the parameters used −

* **formula** is a formula describing the predictor and response variables.
* **data** is the name of the data set used.

### **Input Data**

We will use the R in-built data set named readingSkills to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker.

Here is the sample data.

# Load the party package. It will automatically load other

# required packages.

library(party)

# Print some records from data set readingSkills.

print(head(readingSkills))

### **Example**

We will use the **randomForest()** function to create the decision tree and see it's graph.

# Load the party package. It will automatically load other

# required packages.

library(party)

library(randomForest)

# Create the forest.

output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,

data = readingSkills)

# View the forest results.

print(output.forest)

# Importance of each predictor.

print(importance(fit,type = 2))

# **Survival Analysis**

Survival analysis deals with predicting the time when a specific event is going to occur. It is also known as failure time analysis or analysis of time to death. For example predicting the number of days a person with cancer will survive or predicting the time when a mechanical system is going to fail.

The R package named **survival** is used to carry out survival analysis. This package contains the function **Surv()** which takes the input data as a R formula and creates a survival object among the chosen variables for analysis. Then we use the function **survfit()** to create a plot for the analysis.

## Install Package

install.packages("survival")

### **Syntax**

The basic syntax for creating survival analysis in R is −

Surv(time,event)

survfit(formula)

Following is the description of the parameters used −

* **time** is the follow up time until the event occurs.
* **event** indicates the status of occurrence of the expected event.
* **formula** is the relationship between the predictor variables.

### **Example**

We will consider the data set named "pbc" present in the survival packages installed above. It describes the survival data points about people affected with primary biliary cirrhosis (PBC) of the liver. Among the many columns present in the data set we are primarily concerned with the fields "time" and "status". Time represents the number of days between registration of the patient and earlier of the event between the patient receiving a liver transplant or death of the patient.

# Load the library.

library("survival")

# Print first few rows.

print(head(pbc))

### **Applying Surv() and survfit() Function**

Now we proceed to apply the **Surv()** function to the above data set and create a plot that will show the trend.

# Load the library.

library("survival")

# Create the survival object.

survfit(Surv(pbc$time,pbc$status == 2)~1)

# Give the chart file a name.

png(file = "survival.png")

# Plot the graph.

plot(survfit(Surv(pbc$time,pbc$status == 2)~1))

# Save the file.

dev.off()

# **Chi Square Test**

**Chi-Square test** is a statistical method to determine if two categorical variables have a significant correlation between them. Both those variables should be from same population and they should be categorical like − Yes/No, Male/Female, Red/Green etc.

For example, we can build a data set with observations on people's ice-cream buying pattern and try to correlate the gender of a person with the flavor of the ice-cream they prefer. If a correlation is found we can plan for appropriate stock of flavors by knowing the number of gender of people visiting.

## Syntax

The function used for performing chi-Square test is **chisq.test()**.

The basic syntax for creating a chi-square test in R is −

chisq.test(data)

library("MASS")

print(str(Cars93))

The above result shows the dataset has many Factor variables which can be considered as categorical variables. For our model we will consider the variables "AirBags" and "Type". Here we aim to find out any significant correlation between the types of car sold and the type of Air bags it has. If correlation is observed we can estimate which types of cars can sell better with what types of air bags.

# Load the library.

library("MASS")

# Create a data frame from the main data set.

car.data <- data.frame(Cars93$AirBags, Cars93$Type)

# Create a table with the needed variables.

car.data = table(Cars93$AirBags, Cars93$Type)

print(car.data)

# Perform the Chi-Square test.

print(chisq.test(car.data))

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

Compact Large Midsize Small Sporty Van

Driver & Passenger 2 4 7 0 3 0

Driver only 9 7 11 5 8 3

None 5 0 4 16 3 6

Pearson's Chi-squared test

data: car.data

X-squared = 33.001, df = 10, p-value = 0.0002723

Warning message:

In chisq.test(car.data) : Chi-squared approximation may be incorrect

## Conclusion

The result shows the p-value of less than 0.05 which indicates a string correlation.