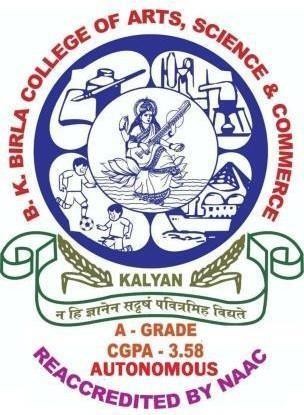
B. K. BIRLA COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS), KALYAN

DEPARTMENT OF INFORMATION TECHNOLOGY

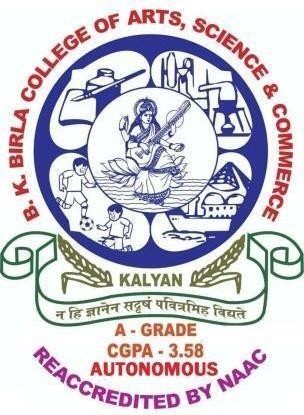


|  |  |
| --- | --- |
| **Student Name:** |  |
| **Student ID:** |  |
| **Class:** |  |
| **Subject:** |  |

**B. K. BIRLA COLLEGE OF ARTS, SCIENCE COMMERCE (AUTONOMOUS), KALYAN**

*(Affiliated to University of Mumbai)*

**KALYAN-MAHARASHTRA-421301**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**CERTIFICATE**

This is to certify that Mr/Ms bearing Seat. No: **( )**, in class

has successfully completed practical of the subject

# Teacher’s Signature:

**Place:**

# Date:

# 

# College Seal

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR. NO** | **PRACTICAL NAME** | **DATE** | **SIGNATURE** |
| 1. | Decision Tree Algorithm |  |  |
| 2. | Association Algorithm |  |  |
| 3. | K-Nearest Neighbor |  |  |
| 4. | Naive Bayes Algorithm |  |  |
| 5. | Time Series Analysis |  |  |
| 6. | Clustering Algorithm |  |  |

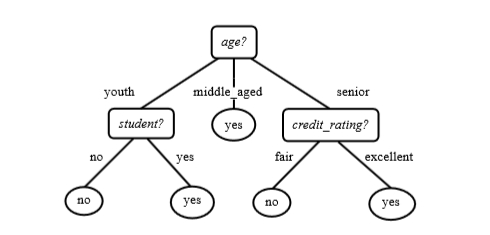
**PRACTICAL - 1**

**DECISION TREE ALGORITHM**

**AIM: SHOW THE IMPLEMENTATION OF DECISION TREE.**

**BACKGROUND INFORMATION:**

Decision tree induction is the learning of decision trees from class- labeled training tuples. A decision tree is a ﬂowchart- like tree structure,where each internal node (non-leaf node) denotes a test on an attribute ,each branch represents an outcome of the test ,and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.



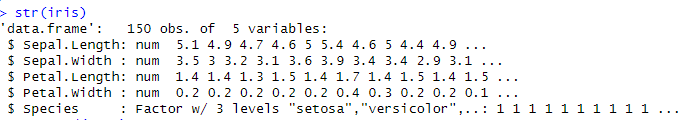
This practical will show the implementation of decision three using package party, rpart&random forest

**Decision Trees with Package party**

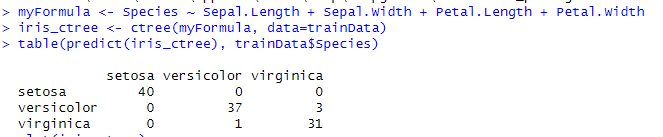
This section shows how to build a decision for the iris dataset with the function ctree() in package party. Iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.

**SOURCE CODE AND OUTPUT**

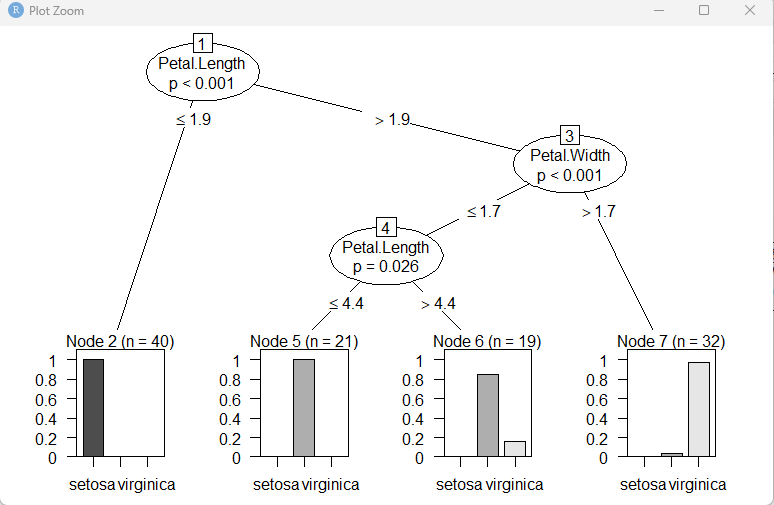
* **str(iris)**



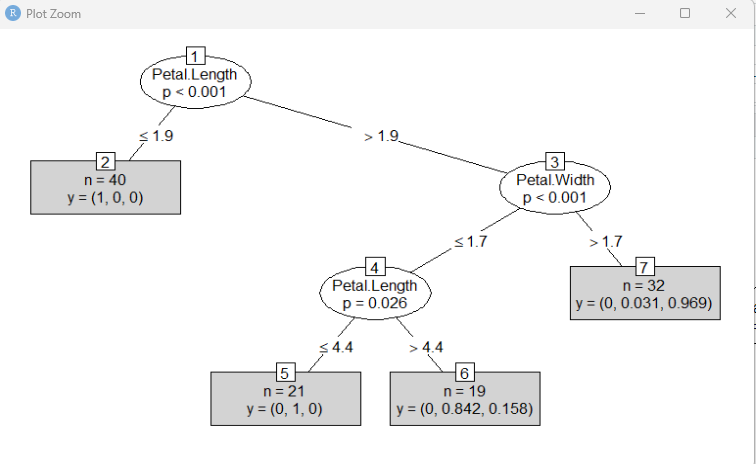
* **set.seed(1234)**
* **ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))**
* **trainData <- iris[ind==1,]**
* **testData <- iris[ind==2,]**
* **library(party) //for checking party package is installed or not**
* **install.packages("party") //for installing party package**
* **install.packages("partykit") //for installing partykit since ctree function is required further**
* **myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width**
* **iris\_ctree <- ctree(myFormula, data=trainData)**
* **table(predict(iris\_ctree), trainData$Species)**

****

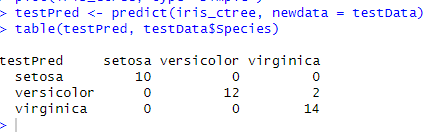
* **plot(iris\_ctree)**

****

* **plot(iris\_ctree, type="simple")**



* **testPred <- predict(iris\_ctree, newdata = testData)**
* **table(testPred, testData$Species)**



**PRACTICAL - 2**

**ASSOCIATION ALGORITHM**

**AIM: SHOW THE IMPLEMENTATION OF ASSOCIATION ALGORITHM.**

**BACKGROUND INFORMATION:**

**Association rule** learning is a method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness.

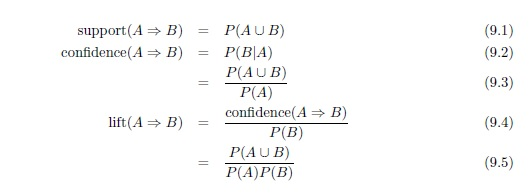
Association rules are rules presenting association or correlation between item sets. An association

rule is in the form of A ) B, where A and B are two disjoint item sets, referred to respectively

as the lhs (left-hand side) and rhs (right-hand side) of the rule.

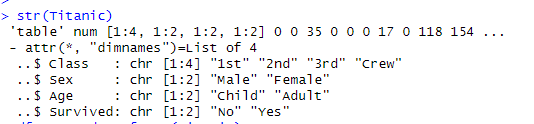
The three most widely-used measures for selecting interesting rules are support, confidence and lift. Support is the percentage of cases in the data that contains both A and B, coincedence is the percentage of cases containing

A that also contain B, and lift is the ratio of con\_dence to the percentage of cases containing B where P(A) is the percentage (or probability) of cases containing A.



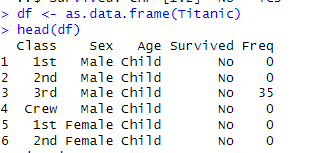
**SOURCE CODE AND OUTPUT**

* **str(Titanic)**

****

* **df <- as.data.frame(Titanic)**

**head(df)**



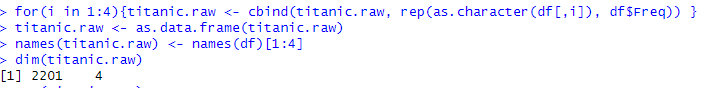
* **titanic.raw <- NULL**

**for(i in 1:4) { titanic.raw <- cbind(titanic.raw, rep(as.character(df[,i]), df$Freq))}**

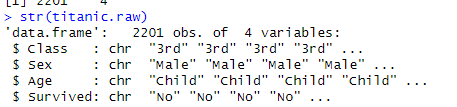
**> titanic.raw <- as.data.frame(titanic.raw)**

**> names(titanic.raw) <- names(df)[1:4]**

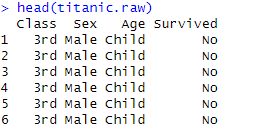
**> dim(titanic.raw)**



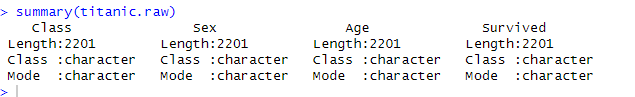
* **str(titanic.raw)**

****

* **head(titanic.raw)**

****

* summary(titanic.raw)



**PRACTICAL - 3**

**K-NEAREST NEIGHBOR**

**AIM: SHOW THE IMPLEMENTATION OF K-NEAREST NEIGHBOR.**

**BACKGROUND INFORMATION:**

The ***k*-Nearest Neighbors algorithm** (or ***k*-NN** for short) is a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics) method used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis). In both cases, the input consists of the *k* closest training examples in the [feature space](https://en.wikipedia.org/wiki/Feature_space). The output depends on whether *k*-NN is used for classification or regression:

* In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive [integer](https://en.wikipedia.org/wiki/Integer), typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
* In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

*k*-NN is a type of [instance-based learning](https://en.wikipedia.org/wiki/Instance-based_learning), or [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning), where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms.

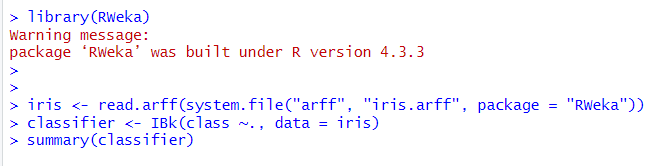
Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor.

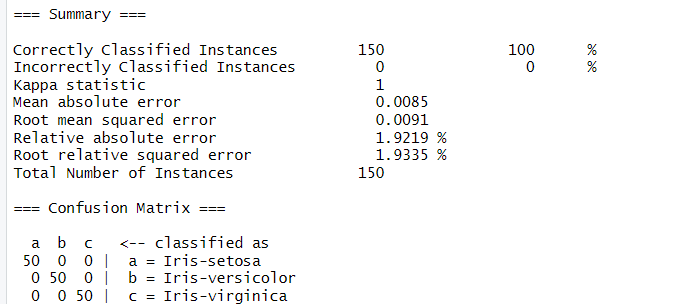
The neighbors are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A shortcoming of the *k*-NN algorithm is that it is sensitive to the local structure of the data.

> library(RWeka)

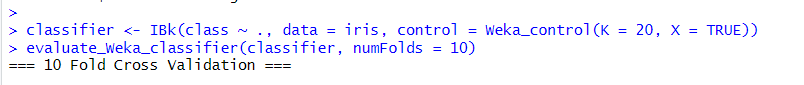
It comes with several well-known datasets, which can be loaded in as ARFF files (Weka's default file format). We now load a sample dataset, the famous Iris dataset [3], and learn a kNN classifier for it, using default parameters:

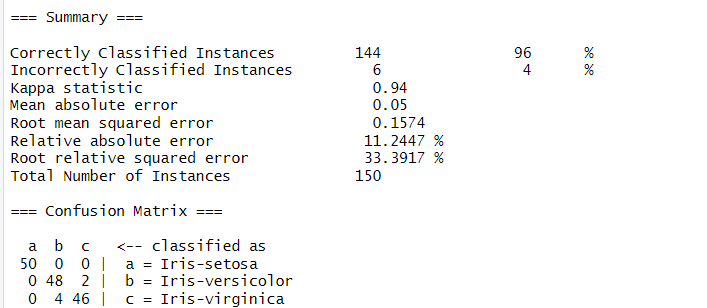




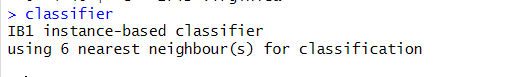
### Execution and Results

We will generate a kNN classifier, but we'll let RWeka automatically find the best value for k, between 1 and 20. We'll also use 10-fold cross validation to evaluate our classifier:





> classifier



**PRACTICAL - 4**

**NAIVE BAYES ALGORITHM**

**AIM: SHOW THE IMPLEMENTATION OF NAIVE BAYES.**

**BACKGROUND INFORMATION:**

In [machine learning](https://en.wikipedia.org/wiki/Machine_learning), naive Bayes classifiers are a family of simple [probabilistic classifiers](https://en.wikipedia.org/wiki/Probabilistic_classifier) based on applying [Bayes' theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence) assumptions between the features.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. [Maximum-likelihood](https://en.wikipedia.org/wiki/Maximum-likelihood_estimation) training can be done by evaluating a [closed-form expression](https://en.wikipedia.org/wiki/Closed-form_expression), which takes [linear time](https://en.wikipedia.org/wiki/Linear_time), rather than by expensive [iterative approximation](https://en.wikipedia.org/wiki/Iterative_method) as used for many other types of classifiers.

In the [statistics](https://en.wikipedia.org/wiki/Statistics) and [computer science](https://en.wikipedia.org/wiki/Computer_science) literature, Naive Bayes models are known under a variety of names, including simple Bayes and independence Bayes. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not (necessarily) a [Bayesian](https://en.wikipedia.org/wiki/Bayesian_probability) method.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of [feature](https://en.wikipedia.org/wiki/Feature_vector) values, where the class labels are drawn from some finite set. It is not a single [algorithm](https://en.wikipedia.org/wiki/Algorithm) for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is [independent](https://en.wikipedia.org/wiki/Independence_%28probability_theory%29) of the value of any other feature, given the class variable.

**SOURCE CODE AND OUTPUT**

> library(e1071)

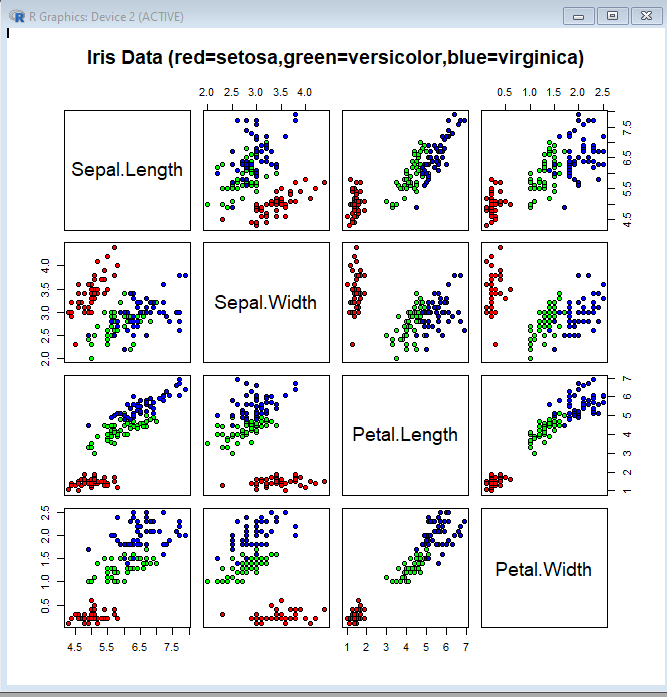
It comes with several well-known datasets, which can be loaded in as ARFF files (Weka's default file format). We now load a sample dataset, the famous Iris dataset [[1]](https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/Na%C3%AFve_Bayes#endnote_Fisher) and learn a Naïve Bayes classifier for it, using default parameters. First, let us take a look at the Iris dataset.

Each instance of datasets contains four attributes:sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. The next picture shows each attribute plotted against the others, with the different classes in color.

> library(e1071)

> pairs(iris[1:4], main = "Iris Data (red=setosa,green=versicolor,blue=virginica)",

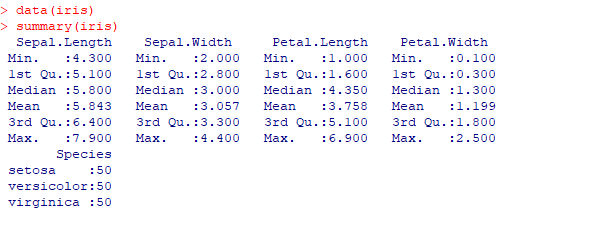
pch = 21, bg = c("red", "green", "blue")[unclass(iris$Species)])



First of all, we need to specify which base we are going to use:

> data(iris)

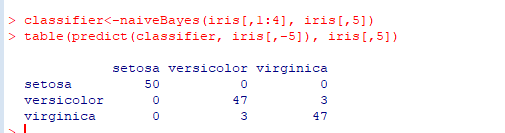
> summary(iris)



After that, we are ready to create a Naïve Bayes model to the dataset using the first 4 columns to predict the fifth. (Factor the target column by so: dataset$col <- factor(dataset$col) )

> classifier<-naiveBayes(iris[,1:4], iris[,5])

> table(predict(classifier, iris[,-5]), iris[,5])



**PRACTICAL - 5**

**TIME SERIES ANALYSIS**

**AIM: SHOW THE IMPLEMENTATION OF TIME SERIES ANALYSIS.**

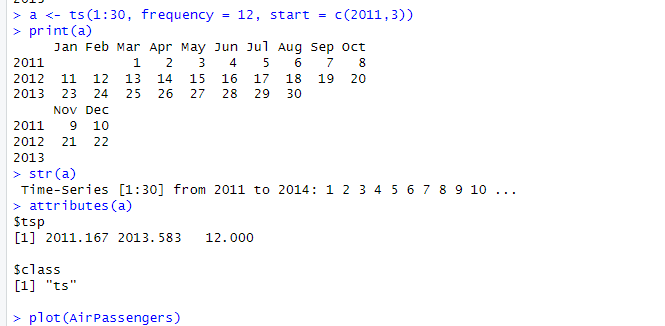
**BACKGROUND INFORMATION:**

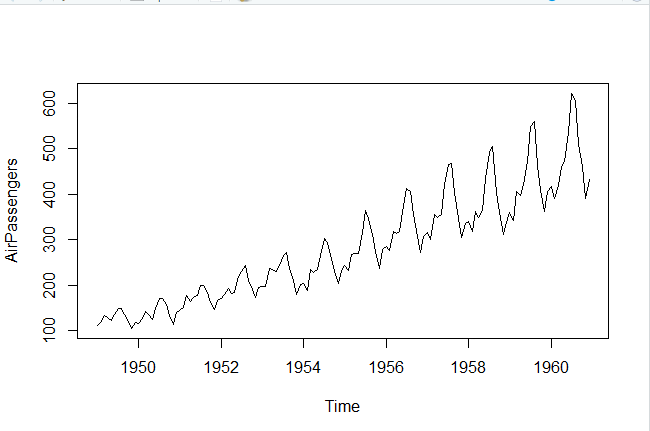
A **time series** is a sequence of [data points](https://en.wikipedia.org/wiki/Data_point), typically consisting of successive measurements made over a time interval. Examples of time series are ocean [tides](https://en.wikipedia.org/wiki/Tides), counts of [sunspots](https://en.wikipedia.org/wiki/Sunspots). Time series are very frequently plotted via [line charts](https://en.wikipedia.org/wiki/Line_chart). Time series are used in [statistics](https://en.wikipedia.org/wiki/Statistics), [signal processing](https://en.wikipedia.org/wiki/Signal_processing), [pattern recognition](https://en.wikipedia.org/wiki/Pattern_recognition), [econometrics](https://en.wikipedia.org/wiki/Econometrics), [mathematical finance](https://en.wikipedia.org/wiki/Mathematical_finance), [weather forecasting](https://en.wikipedia.org/wiki/Weather_forecasting), intelligent transport and trajectory forecasting,[earthquake prediction](https://en.wikipedia.org/wiki/Earthquake_prediction), [electroencephalography](https://en.wikipedia.org/wiki/Electroencephalography), [control engineering](https://en.wikipedia.org/wiki/Control_engineering), [astronomy](https://en.wikipedia.org/wiki/Astronomy), [communications engineering](https://en.wikipedia.org/wiki/Communications_engineering), and largely in any domain of applied [science](https://en.wikipedia.org/wiki/Applied_science) and [engineering](https://en.wikipedia.org/wiki/Engineering) which involves [temporal](https://en.wikipedia.org/wiki/Time) measurements.

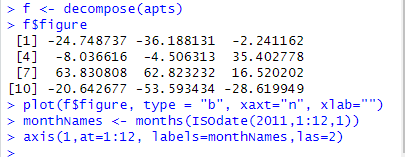
**Time series *analysis*** comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. **Time series *forecasting*** is the use of a [model](https://en.wikipedia.org/wiki/Model_%28abstract%29) to predict future values based on previously observed values. While [regression analysis](https://en.wikipedia.org/wiki/Regression_analysis) is often employed in such a way as to test theories that the current values of one or more independent time series affect the current value of another time series, this type of analysis of time series is not called "time series analysis", which focuses on comparing values of a single time series or multiple dependent time series at different points in time. Time series analysis can be applied to [real-valued](https://en.wikipedia.org/wiki/Real_number), continuous data, [discrete](https://en.wiktionary.org/wiki/discrete) [numeric](https://en.wikipedia.org/wiki/Data_type#Numeric_types) data, or discrete symbolic data.

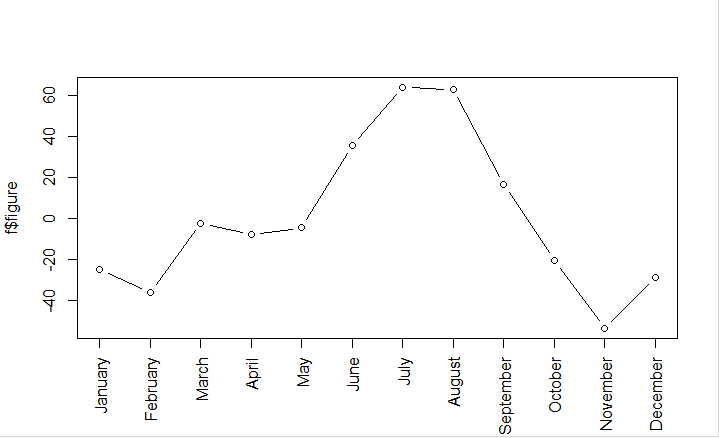
When information is transferred across time, often to specific points in time, the process is known as [forecasting](https://en.wikipedia.org/wiki/Forecasting).

**SOURCE CODE AND OUTPUT :**

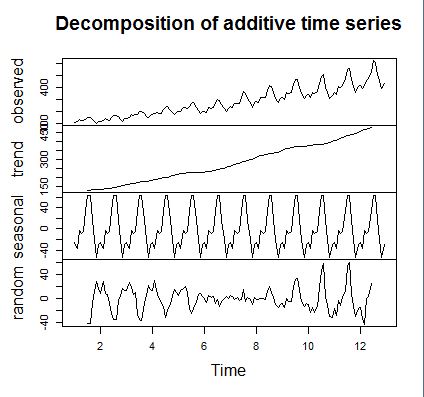


****

****

****

**Seasonal Components**

****

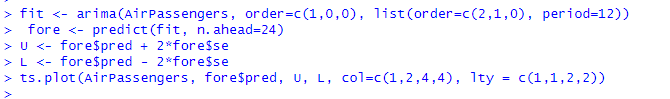
**Time Series Forecasting**

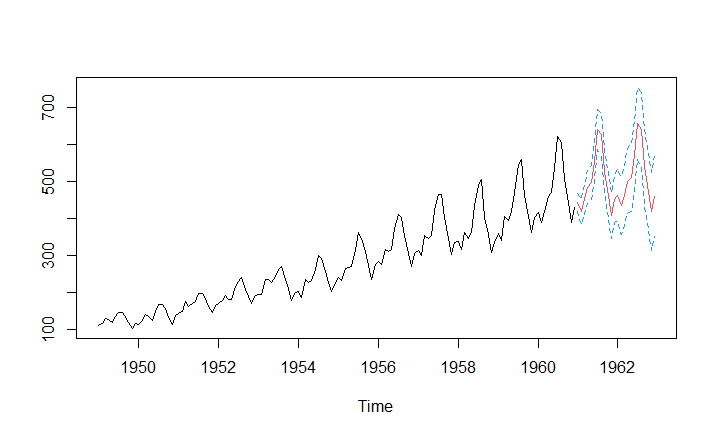
Time series forecasting is to forecast future events based on historical data. One example is

to predict the opening price of a stock based on its past performance. Two popular models for

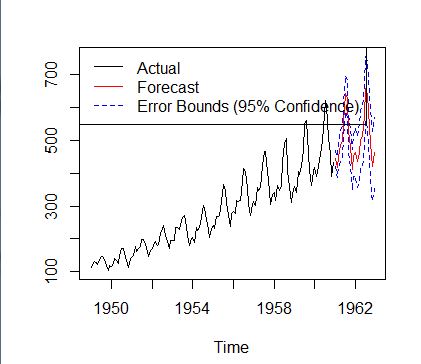
time series forecasting are autoregressive moving average (ARMA) and autoregressive integrated

moving average (ARIMA)



****

****

****

**PRACTICAL - 6**

**CLUSTERING ALGORITHM**

**AIM: SHOW THE IMPLEMENTATION OF CLUSTERING ALGORITHM.**

**BACKGROUND INFORMATION:**

**Cluster analysis** or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory [data mining](https://en.wikipedia.org/wiki/Data_mining), and a common technique for [statistical](https://en.wikipedia.org/wiki/Statistics) [data analysis](https://en.wikipedia.org/wiki/Data_analysis)

***k*-means clustering** is a method of [vector quantization](https://en.wikipedia.org/wiki/Vector_quantization), originally from signal processing, that is popular for [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis) in [data mining](https://en.wikipedia.org/wiki/Data_mining). *k*-means clustering aims to [partition](https://en.wikipedia.org/wiki/Partition_of_a_set) *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest [mean](https://en.wikipedia.org/wiki/Mean), serving as a [prototype](https://en.wikipedia.org/wiki/Prototype) of the cluster.

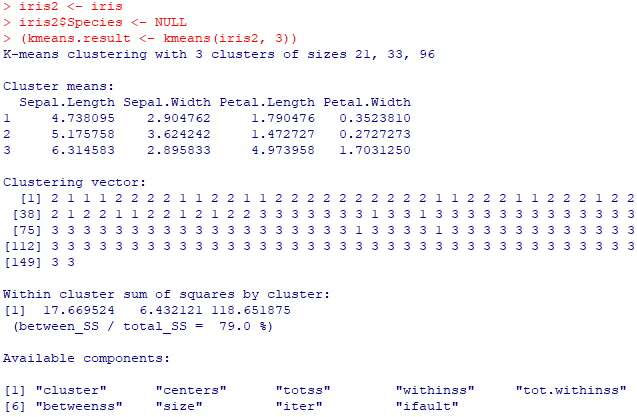
**Hierarchical clustering** involves creating clusters that have a predetermined ordering from top to bottom. For example, all files and folders on the hard disk are organized in a hierarchy. There are two types of hierarchical clustering, *Divisive* and *Agglomerative*.

**SOURCE CODE AND OUTPUT**

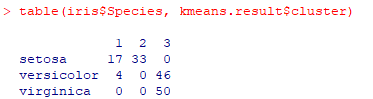
**> iris2 <- iris**

**> iris2$Species <- NULL**

**> (kmeans.result <- kmeans(iris2, 3))**

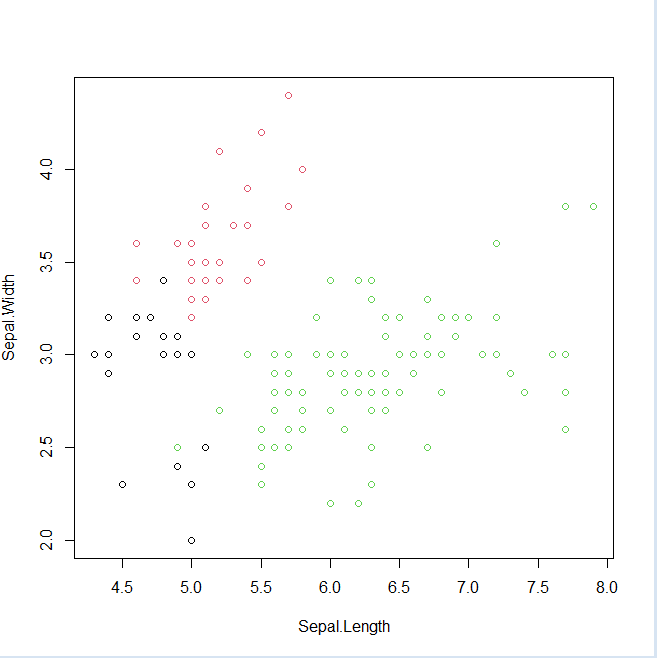


**>table(iris$Species, kmeans.result$cluster )**

****

**> plot(iris2[c("Sepal.Length", "Sepal.Width")], col = kmeans.result$cluster)**

**> points(kmeans.result$centers[,c("Sepal.Length", "Sepal.Width")], col = 1:3, pch = 8, cex=2)**



**Hierarchical**

**> idx <- sample(1:dim(iris)[1], 40)**

**> irisSample <- iris[idx,]**

**> irisSample$Species <- NULL**

**> hc <- hclust(dist(irisSample), method="ave")**

**> plot(hc, hang = -1, labels=iris$Species[idx])**

**> rect.hclust(hc, k=3)**

**> groups <- cutree(hc, k=3)**

