Case Study: IRIS Classification

Hello World of Machine Learning

This is a good case study and dataset to begin an introduction to Machine Learning because it is so well understood.

- We will learn how to load and handle data.
- It is a classification problem, allowing us to experiment with various ML Algos
- It is a mutli-class classification problem (multi-nominal) that may require some specialized handling.
- It only has 4 attribute and 150 rows, meaning it is small and easily fits into memory.
- It gives an opportunity to apply many Machine Learning algorithms, to compare and contrast

---

Installing the required packages

We need to load a few important packages first to begin our analysis

CARET Package

The caret package provides a consistent interface into hundreds of machine learning algorithms and provides useful convenience methods for data visualization, data resampling, model tuning and model comparison, among other features. It's a must have tool for machine learning projects in R. Refer

TIDYR Package:

Is a new package that makes it easy to "tidy" your data. Tidy data is data that's easy to work with: it's easy to munge (*with dplyr*), visualise (*with ggplot2 or ggvis*) and model (with R's hundreds of modelling packages). The two most important properties of tidy data are:

- Each column is a variable.
- Each row is an observation.

Arranging your data in this way makes it easier to work with because you have a consistent way of referring to variables (as column names) and observations (as row indices). Refer

```
# install.packages("caret")
# install.packages("tidyr")
```

Get the Data

About the data set

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper. This is a very famous and widely used dataset by everyone trying to learn machine learning and statistics. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. The fifth column is the species of the flower observed. For more history of this dataset read here Wikipedia

---

Load the dataset

We are going to do the following

- Load the iris data directly from within R
- Download and Load the data from a file. (*This is usually what you have to do for other problems*)
- Split the data into a training dataset and a testing dataset to perform machine learning (*This is a standard 2 way split in machine learning. We could also do a 3 way split: training, validation and testing*)

Loading the data from within R

```
# Attach the dataset to the environment
data(iris)
# Get help on the data
help(iris)
# Rename the data
iris_dataset<-iris
# View the data
View(iris_dataset)
```

Loading the data from external source

In most cases you will work with, the data file is usually hosted somewhere on the internet which you might have to download and read the file into the R environment/memory before you can start working with it. It is also good practice to download the files from directly within your R script whenever possible because it makes your work more shareable and reproducible by others. The code below illustrates how this data is downloaded from the UCI Machine Learning repository

```
# set the url for download
url<-"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# set the filename and directory to download into
filename<-"./Data/iris.csv"
# Download the file
download.file(url=url, destfile = filename, method ="curl")
print("IRIS File downloaded")
```

```
## [1] "IRIS File downloaded"
```

Next we need to read the data from the file

```
# Read the file into the R environment
iris_filedata<-read.csv(file = filename,header = FALSE,sep = ",")
```

Viewing the data from within the R console - Useful when dealing with larger datasets

```
# View the top few rows of the data in R console
head(iris_filedata,7)
```

```
##    V1  V2  V3  V4          V5
## 1 5.1 3.5 1.4 0.2 Iris-setosa
## 2 4.9 3.0 1.4 0.2 Iris-setosa
## 3 4.7 3.2 1.3 0.2 Iris-setosa
## 4 4.6 3.1 1.5 0.2 Iris-setosa
## 5 5.0 3.6 1.4 0.2 Iris-setosa
```

```
## 6 5.4 3.9 1.7 0.4 Iris-setosa
## 7 4.6 3.4 1.4 0.3 Iris-setosa
```

We see here that the data has no names for the columns (they are assigned names such as V1, V2, V3 etc). So next, we will assign names to these columns

```
# Assigning meaningful column names
colnames(iris_filedata)<-c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species")
head(iris_filedata,5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width     Species
## 1          5.1         3.5          1.4         0.2 Iris-setosa
## 2          4.9         3.0          1.4         0.2 Iris-setosa
## 3          4.7         3.2          1.3         0.2 Iris-setosa
## 4          4.6         3.1          1.5         0.2 Iris-setosa
## 5          5.0         3.6          1.4         0.2 Iris-setosa
```

Splitting the Data into Training and Testing Sets

This is one of the most important steps and concepts in Machine Learning. Before we do any meaningful work and learn from the dataset available to us, we need to split the dataset into `training set` and `testing set` and sometimes into a third `validation set`.

TRAINING SET: Is the `SEEN DATA` which is used to build and train the model. In classification problems such as this, we train the model using the classfication error rate: the percentage of incorrectly/correctly classified instances. We use the training data set to help us understand the data, select the appropriate model and determine model parameters.

TESTING SET This is the `UNSEEN DATA`. We build a model because we want to classify `new data`. We are also chiefly interested in the model performance(error rate) on this new data as it is more realistic estimate of the model fit in the real world.

VALIDATION SET Sometimes a part of the training set is split into the Validation Set to help us tune and optimize our models. It can be thought of as a Practice Testing set before we actually test the model using the TESTING SET

A good discussion of this topic can be found here

```
#Load the Caret package which allows us to partition the data
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# We use the dataset to create a partition (80% training 20% testing)
index <- createDataPartition(iris_filedata$Species, p=0.80, list=FALSE)
# select 20% of the data for testing
testset <- iris_filedata[-index,]
# select 80% of data to train the models
trainset <- iris_filedata[index,]
```

Now that we have loaded and prepared our data for analysis, we are ready to move onto the next step which is `Exploring, Summarizing, Plotting and Understanding the Data`.

Explore the Data

Since we are dealing here with a clean and small dataset we are able to avoid the step of ***`Cleaning the data`*** *which typically consumes a majority of the data scientists' time.*

We explore the data to:

- Understand the data
- Summarize the data
- Clean and Prune the data
- Understand relationships between atrributes
- Think about and source other data which maybe useful in answering the question
- Get a preliminary feel for the types of models we think would best fit the data

Summarizing the Data

We begin by getting an idea of the attributes, dimensions and size of the data we are dealing with

```
# Dimensions of the data
dim(trainset)
```

```
## [1] 120   5
```

```
# Structure of the data
str(trainset)
```

```
## 'data.frame':    120 obs. of  5 variables:
##  $ Sepal.Length: num  4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 5.4 ...
##  $ Sepal.Width : num  3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 ...
##  $ Petal.Length: num  1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 ...
##  $ Species     : Factor w/ 3 levels "Iris-setosa",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# Summary of the data
summary(trainset)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.400   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.750   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.862   Mean   :3.063   Mean   :3.784   Mean   :1.188
##  3rd Qu.:6.500   3rd Qu.:3.300   3rd Qu.:5.125   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.700   Max.   :2.500
##             Species
##  Iris-setosa    :40
##  Iris-versicolor:40
##  Iris-virginica :40
##
##
##
```

```
# Levels of the prediction column
levels(trainset$Species)
```

```
## [1] "Iris-setosa"     "Iris-versicolor" "Iris-virginica"
```

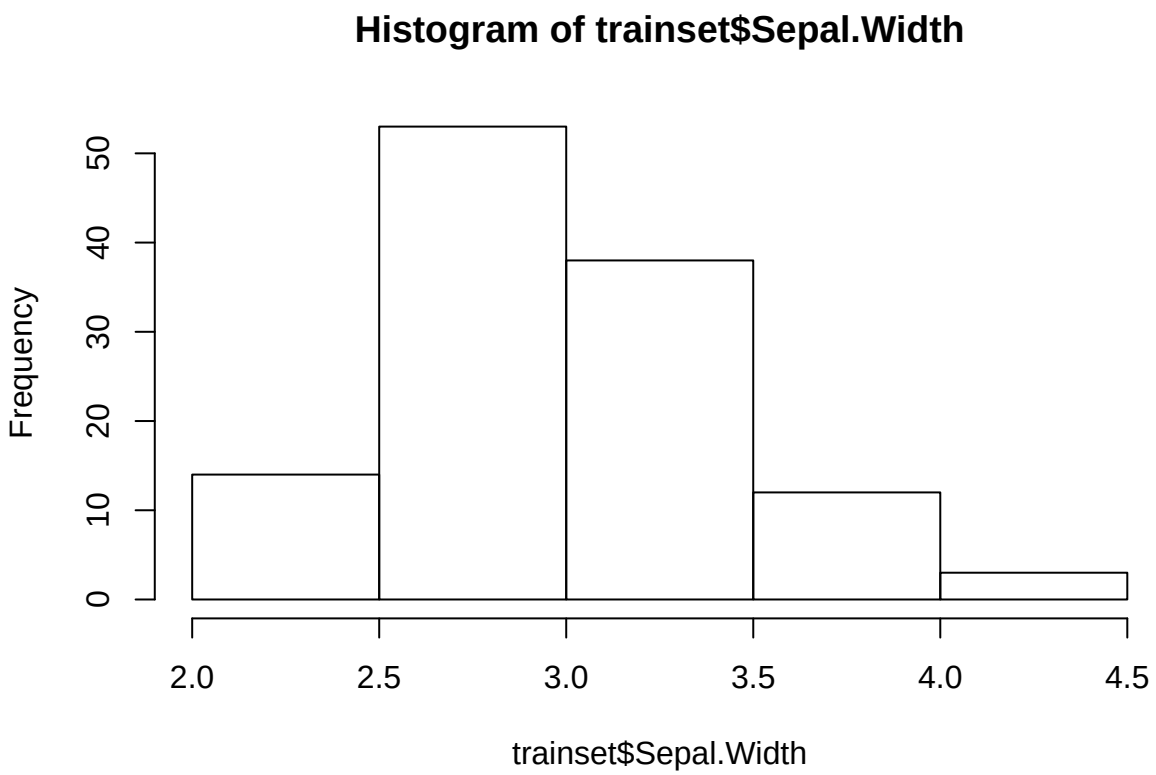
Visualization and Exploration

The advantage of a tool such as RStudio is that it allows a data scientist to creatively and quickly explore data via visualization. There are many powerful packages and tools in R such as `GGPLOT2`, `PLOTLY` etc which can be used to explore data as well as produce publishing quality plots to be used in reports and presentations. Here we will use some of these to explore and understand our dataset better

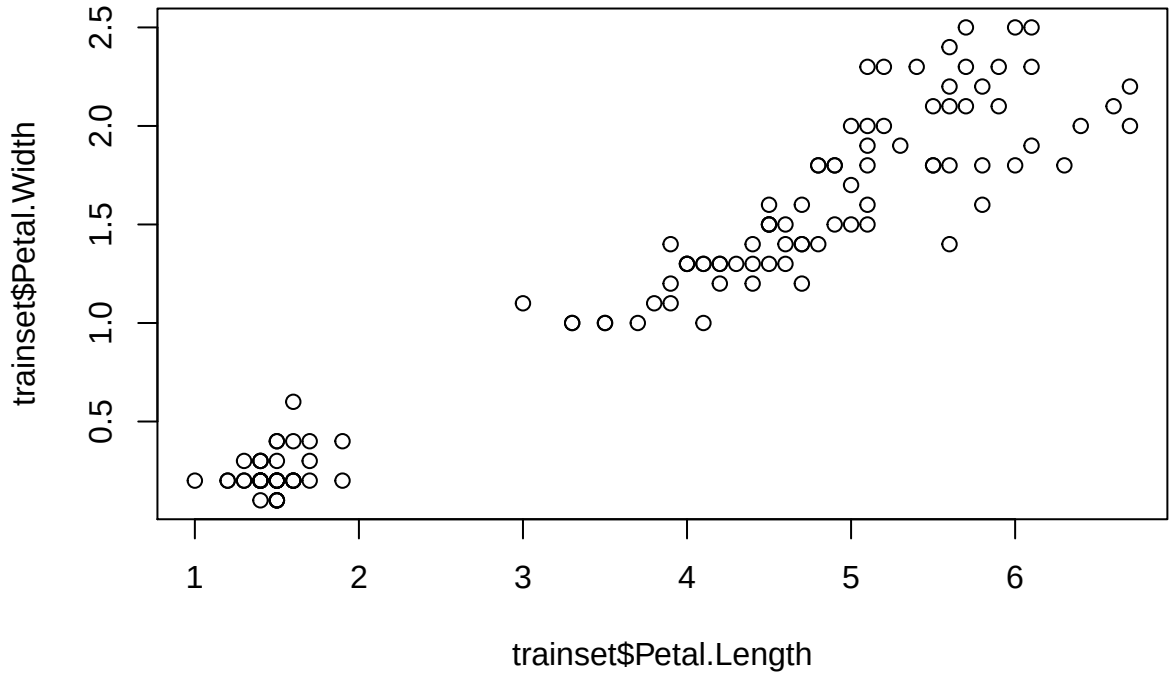
Base R plots

We begin by using some base R plots to understand the distribution and attributes

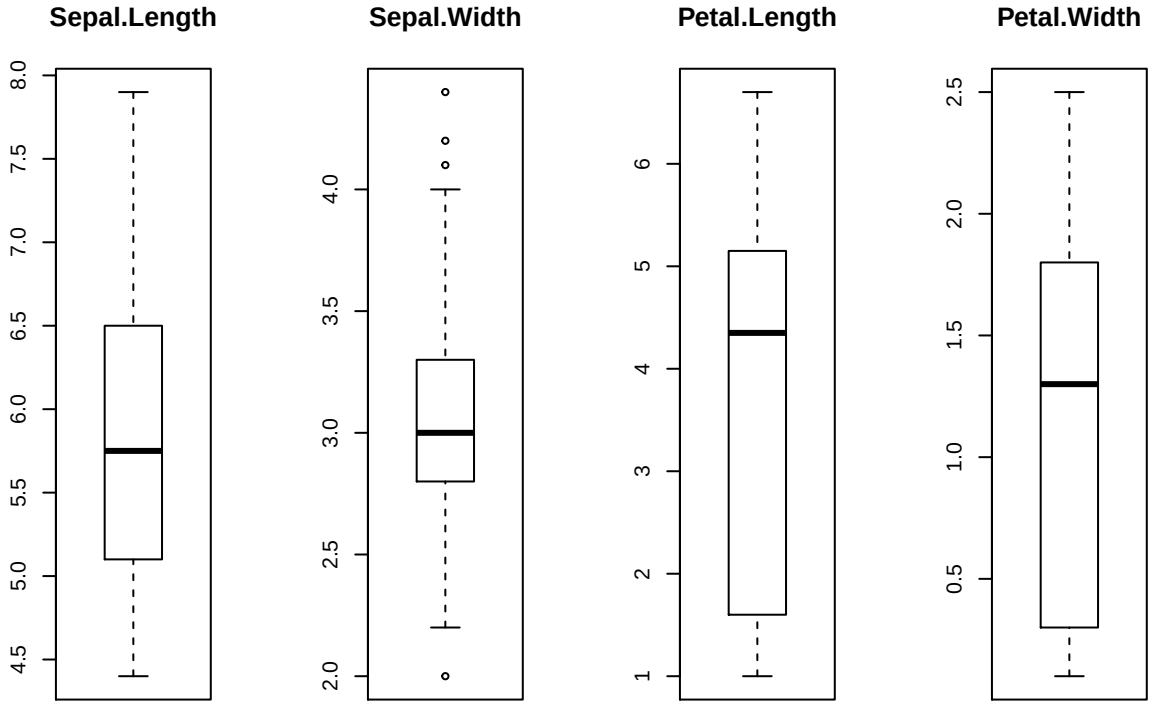
```
## Histogram
hist(trainset$Sepal.Width)
```



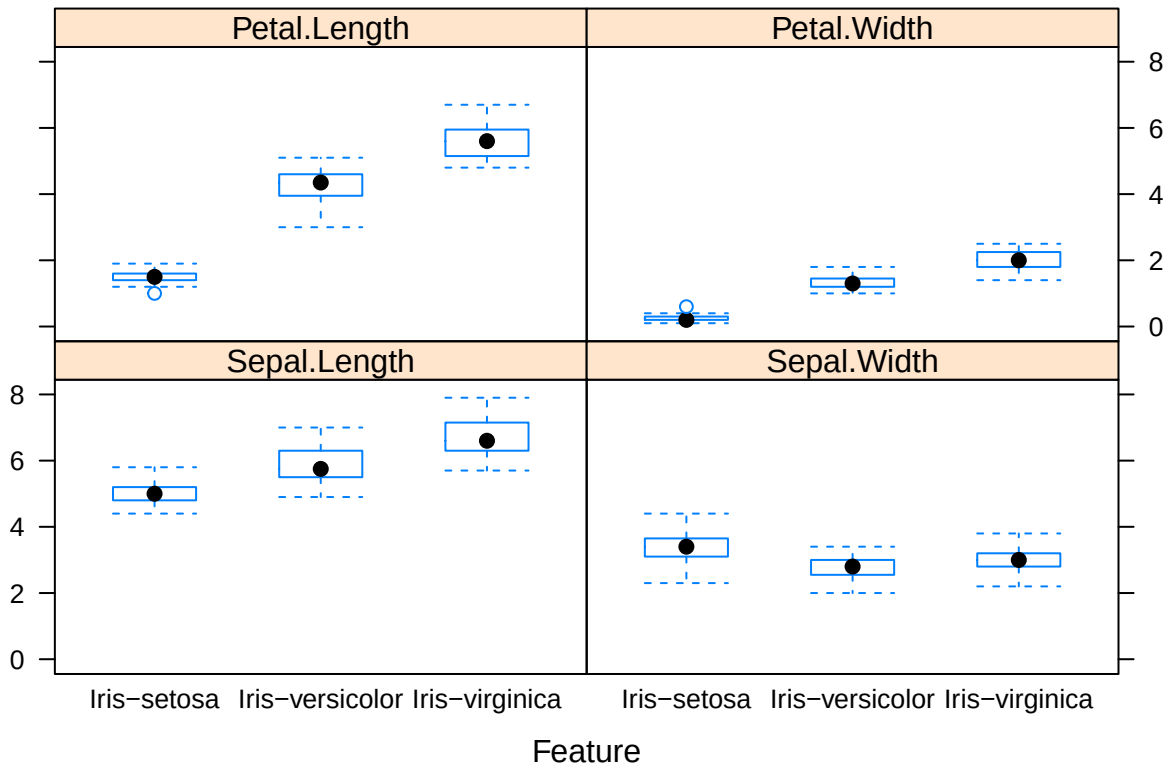

```
## Scatter plot to check relationship between two attributes
plot(trainset$Petal.Length,trainset$Petal.Width)
```



```
## Box plot to understand how the distribution varies for each attribute
par(mfrow=c(1,4))
  for(i in 1:4) {
  boxplot(trainset[,i], main=names(trainset)[i])
}
```




```
## Another Box plot to understand the distribution for each attribute broken down by class
featurePlot(x=trainset[,1:4], y=trainset[,5], plot="box")
```



Grammer of Graphics GGPLOTs

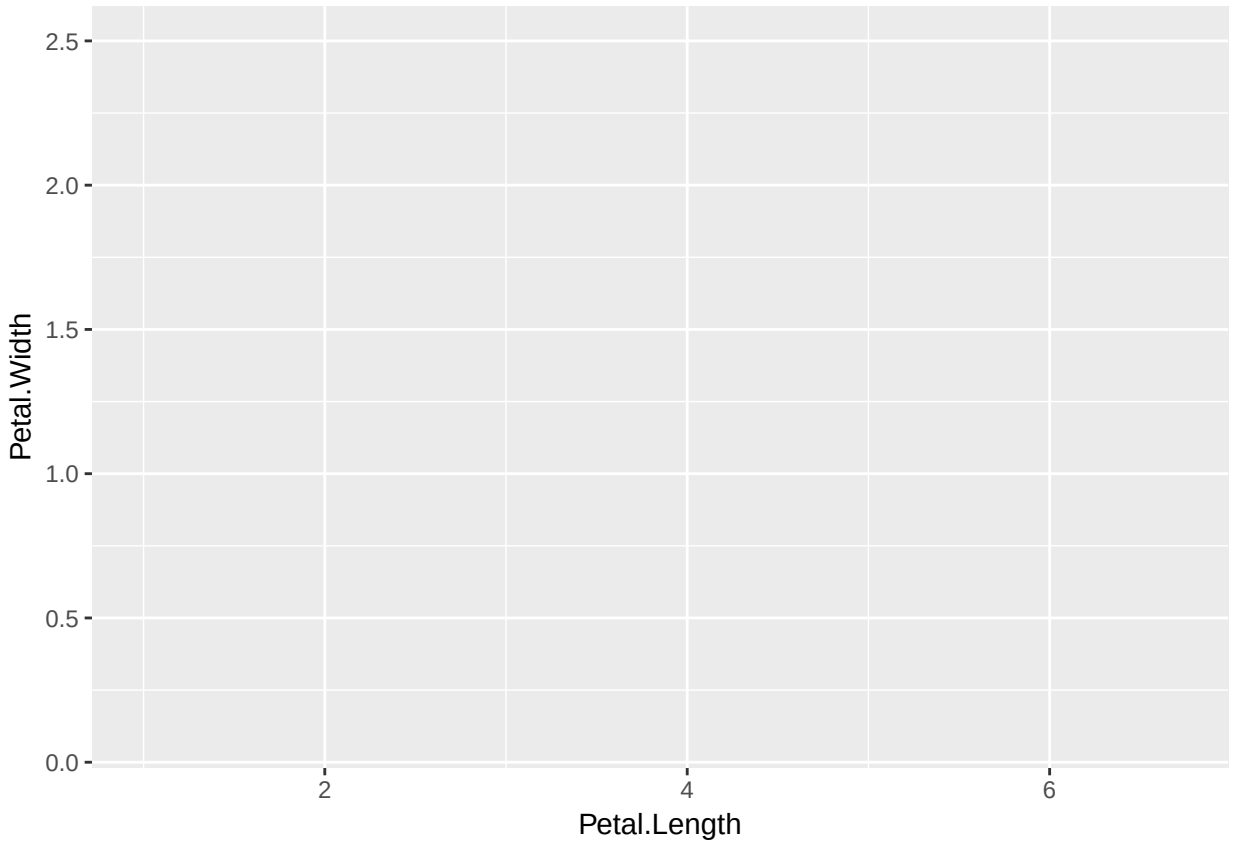
ggplot2 is a very powerful package built on the concept of `"Grammar of Graphics (gg)"`. It allows you to **build** a plot following a particular syntax. It produces more aesthetically pleasing, more powerful plots than base graphics using more compact code for plot of similar complexity. You can read and understand more about ggplot here and here. A cheatsheet for ggplot is available here

To understand the visualization techniques, best practices and how to build plots with ggplot using the Grammar of Graphics ideas, refer the link below which is an excellent tutorial on Visualization

Below we will plot some similar yet aesthetically pleasing plots using ggplot2

```
# begin by loading the library
library(ggplot2)
```

```
# Scatter plot
g <- ggplot(data=trainset, aes(x = Petal.Length, y = Petal.Width))
print(g)
```

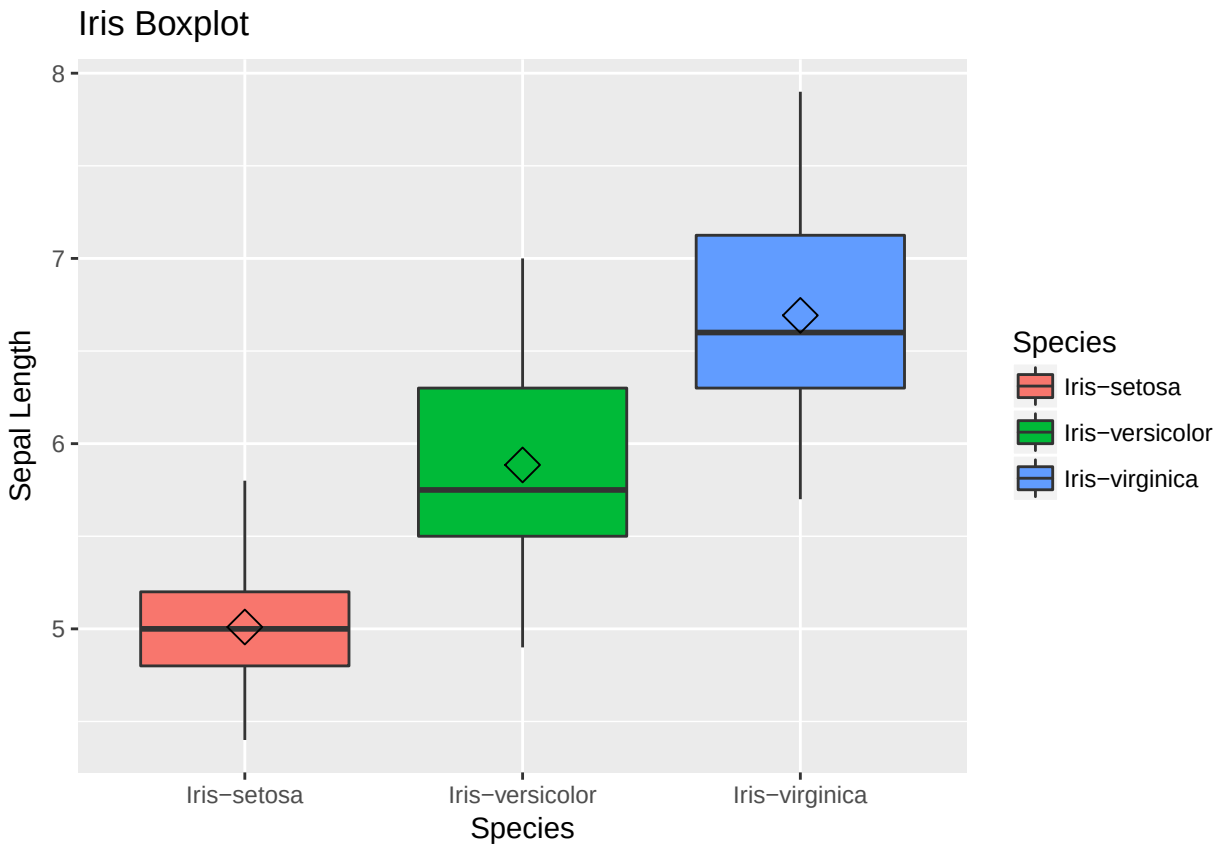


You see above that unlike regular R plot() function, calling `ggplot()` by itself produces no output. In ggplot if define the canvas first loading the data and then build layers of objects one by one. Below we add a `geom` point to plot the data points. We modify the axes labels and titles. We can also add a linear regression line through the points using yet another `geom` called `geom_smooth`. It computes the linear regression line and the errors before plotting it through a simple command

```
g <-g +
    geom_point(aes(color=Species, shape=Species)) +
    xlab("Petal Length") +
    ylab("Petal Width") +
    ggtitle("Petal Length-Width")+
    geom_smooth(method="lm")

print(g)
```

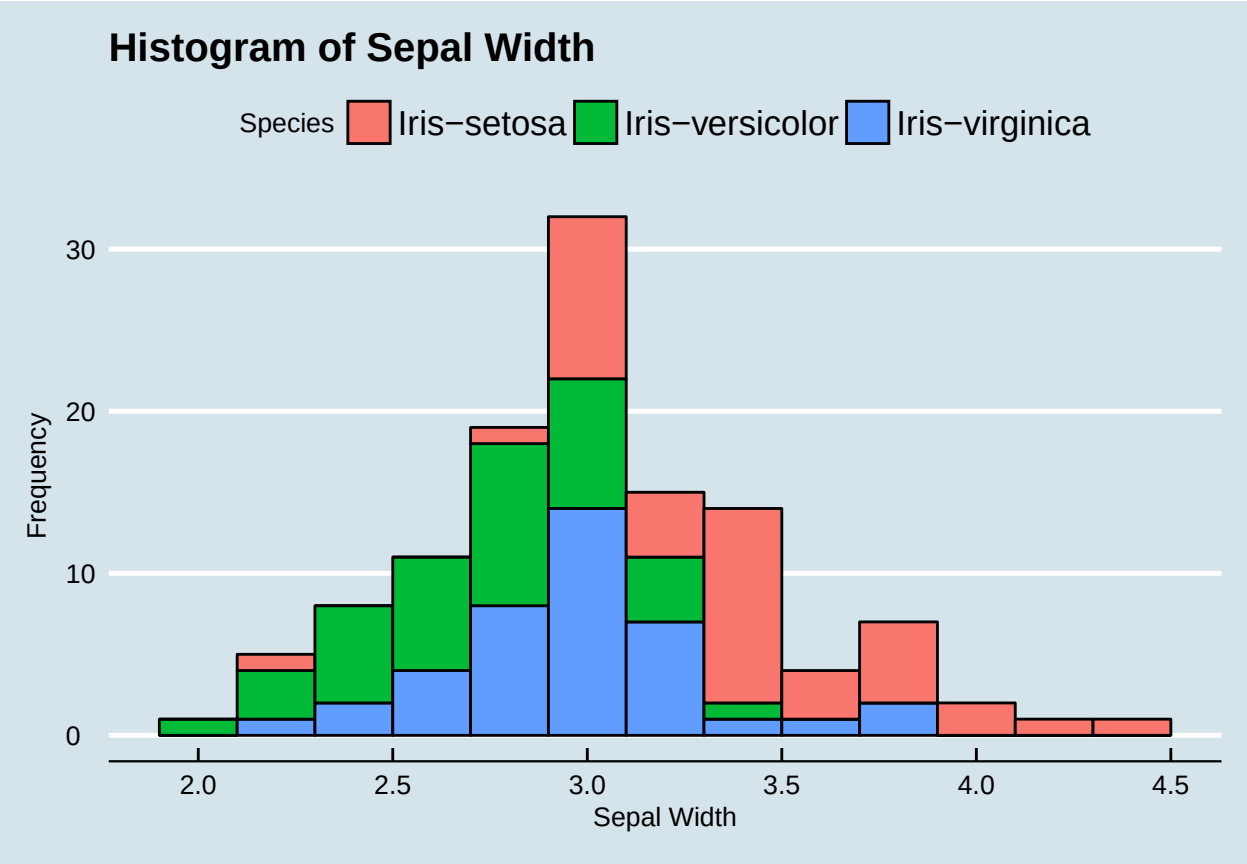
In the next chart we use a `boxplot` geomand summarize the statistic `mean` to display it on top the box plot.

```
## Box Plot
box <- ggplot(data=trainset, aes(x=Species, y=Sepal.Length)) +
    geom_boxplot(aes(fill=Species)) +
    ylab("Sepal Length") +
    ggtitle("Iris Boxplot") +
    stat_summary(fun.y=mean, geom="point", shape=5, size=4)

print(box)
```

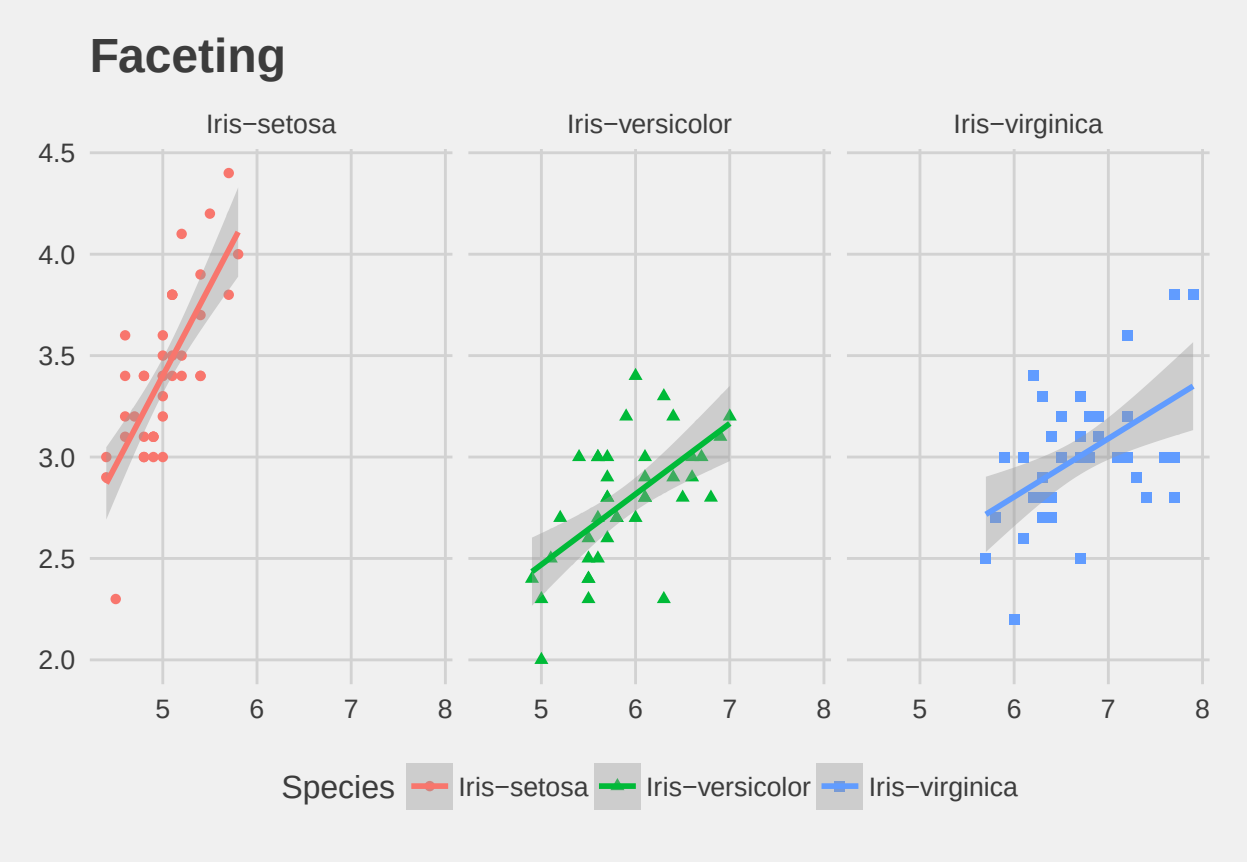


We can also use themes to make our plots look professional

```
library(ggthemes)
## Histogram
histogram <- ggplot(data=trainset, aes(x=Sepal.Width)) +
    geom_histogram(binwidth=0.2, color="black", aes(fill=Species)) +
    xlab("Sepal Width") +
    ylab("Frequency") +
    ggtitle("Histogram of Sepal Width")+
    theme_economist()

print(histogram)
```




```
## Faceting: Producing multiple charts in one plot
library(ggthemes)
facet <- ggplot(data=trainset, aes(Sepal.Length, y=Sepal.Width, color=Species))+
    geom_point(aes(shape=Species), size=1.5) +
    geom_smooth(method="lm") +
    xlab("Sepal Length") +
    ylab("Sepal Width") +
    ggtitle("Faceting") +
    theme_fivethirtyeight() +
    facet_grid(. ~ Species) # Along rows

print(facet)
```



Getting Started with Machine Learning

Now that we have loaded the data and explored it to get a basic understanding of the relationships between the attributes, we can move to the next step which is **`Model Building`**.

> **The Problem:** *Given a set of data about the flowers (column 1 through 4) can we predict which of the 3 classes of flowers it belongs to.*

We will build and fit a few different models to the training set and try to learn from the **`trainset`** data. This is machine learning. We will later use the best model selected (or even a combination of models) to predict the classification for the **`testset`**. We will then measure how well our model is expected to perform in the real world.

Decision Tree Classifiers

Decision Trees are a widely used set of algorithms used in Classification as well as Regression Problems in Data mining. Decision Trees classify observations by sorting them down the tree from the root node to the leaf node which provides the classification for the observation. Each node specifies a test on a particular attribute and each branch from that node represents one of the possible values for that test. These represent a form of supervised learning as trees can be first learnt using training observations and then be used to predict on the test set.

There are many decision tree algorithms available and vary by the method the trees are constructed and grown. Here we will use the simple `rpart` algorithm to classify our data set and predict

```
library(caret)
set.seed(1000)
?rpart
```

```
## No documentation for 'rpart' in specified packages and libraries:
## you could try '??rpart'
```

```
# Fit the model
model.rpart<-train(x = trainset[,1:4],y = trainset[,5], method = "rpart",metric = "Accuracy")
```

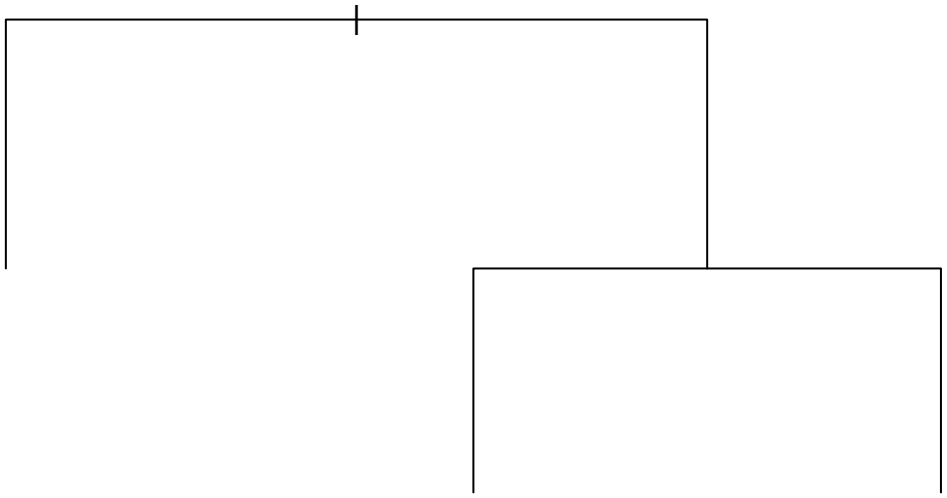
```
## Loading required package: rpart
```

```
print(model.rpart)
```

```
## CART
##
## 120 samples
##   4 predictor
##   3 classes: 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 120, 120, 120, 120, 120, 120, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.00  0.9484135  0.9217799
##   0.45  0.6836306  0.5447280
##   0.50  0.5280749  0.3206051
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.
```

Let us now plot the tree and see how the classification tree looks

```
plot(model.rpart$finalModel)
```

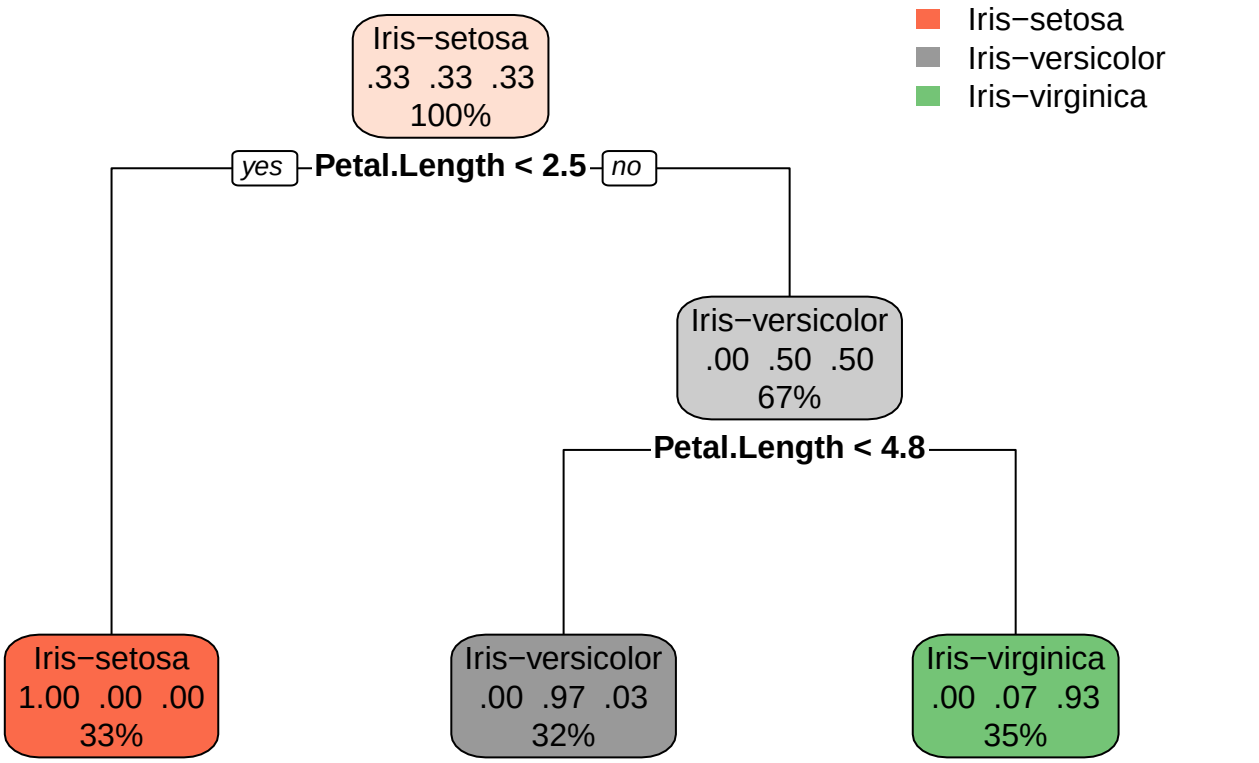


```
model.rpart$finalModel
```

```
## n= 120
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 120 80 Iris-setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 40  0 Iris-setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 80 40 Iris-versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Length< 4.85 38  1 Iris-versicolor (0.00000000 0.97368421 0.02631579) *
##     7) Petal.Length>=4.85 42  3 Iris-virginica (0.00000000 0.07142857 0.92857143) *
```

Let us produce a prettier tree

```
## We use the Rattle package to produce some pretty tree plots
# install.packages("rattle")
# #library(rattle)
library(rpart.plot)
# install.packages("rpart.plot")
#fancyRpartPlot(model.rpart$finalModel)
rpart.plot(model.rpart$finalModel)
```



Now we can check how the tree performs on our training data

```
## Predictions on train dataset
pred<-table(predict(object = model.rpart$finalModel,newdata = trainset[,1:4],type="class"))
pred
```

```
##
##     Iris-setosa Iris-versicolor  Iris-virginica
##              40              38              42
```

```
## Checking the accuracy using a confusion matrix by comparing predictions to actual classifications
confusionMatrix(predict(object = model.rpart$finalModel,newdata = trainset[,1:4],type="class"),trainset$
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              40               0              0
##   Iris-versicolor           0              37              1
##   Iris-virginica            0               3             39
##
## Overall Statistics
##
##                Accuracy : 0.9667
##                  95% CI : (0.9169, 0.9908)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.95
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 0.9250
## Specificity                      1.0000                 0.9875
## Pos Pred Value                   1.0000                 0.9737
## Neg Pred Value                   1.0000                 0.9634
## Prevalence                       0.3333                 0.3333
## Detection Rate                   0.3333                 0.3083
## Detection Prevalence             0.3333                 0.3167
## Balanced Accuracy                1.0000                 0.9563
##                      Class: Iris-virginica
## Sensitivity                         0.9750
## Specificity                         0.9625
## Pos Pred Value                      0.9286
## Neg Pred Value                      0.9872
## Prevalence                          0.3333
## Detection Rate                      0.3250
## Detection Prevalence                0.3500
## Balanced Accuracy                   0.9688
```

Let us predict on the test data set and see how the `rpart` model does

```
## Checking accuracy on the testdata set we created initially
pred_test1<-predict(object = model.rpart$finalModel,
                    newdata = testset[,1:4],
                    type="class")

confusionMatrix(pred_test1,testset$Species)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
```

```
##   Iris-setosa              10               0              0
##   Iris-versicolor           0               9              2
##   Iris-virginica            0               1              8
##
## Overall Statistics
##
##                Accuracy : 0.9
##                  95% CI : (0.7347, 0.9789)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 1.665e-10
##
##                   Kappa : 0.85
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 0.9000
## Specificity                      1.0000                 0.9000
## Pos Pred Value                   1.0000                 0.8182
## Neg Pred Value                   1.0000                 0.9474
## Prevalence                       0.3333                 0.3333
## Detection Rate                   0.3333                 0.3000
## Detection Prevalence             0.3333                 0.3667
## Balanced Accuracy                1.0000                 0.9000
##                      Class: Iris-virginica
## Sensitivity                         0.8000
## Specificity                         0.9500
## Pos Pred Value                      0.8889
## Neg Pred Value                      0.9048
## Prevalence                          0.3333
## Detection Rate                      0.2667
## Detection Prevalence                0.3000
## Balanced Accuracy                   0.8750
```

Our accuracy on the test set is worse than our accuracy on the training set. This is usually to be expected. If the training set accuracy differs from the test set accuracy by a lot, usually it is an indication of model overfitting to the training set.

Random Forest Algorithm

Produce a model which averages the predictions from many such classification trees. This class of algo's uses a variation on the technique called Bagging using different attributes for growing each tree.

From Wikipedia

> Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Because of the standardization of the `CARET` package constructing fitting and verifying the performance of a `random forest` algorithm is very similar to the `rpart` algorithm we just saw

```
library(caret)
## install.packages("rf")
set.seed(1000)
?randomForest
```

```
## No documentation for 'randomForest' in specified packages and libraries:
## you could try '??randomForest'
```

```
# Fit the model
model.rf<-train(x = trainset[,1:4],y = trainset[,5], method = "rf",metric = "Accuracy")
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# Print the model
print(model.rf)
```

```
## Random Forest
##
## 120 samples
##   4 predictor
##   3 classes: 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 120, 120, 120, 120, 120, 120, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9501940  0.9243896
##   3     0.9547815  0.9313248
##   4     0.9549063  0.9315307
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 4.
```

```
## Verify the accuracy on the training set
pred<-predict(object = model.rf$finalModel,newdata = trainset[,1:4],type="class")
confusionMatrix(pred,trainset$Species)
```

```
## Confusion Matrix and Statistics
##
##                 Reference
## Prediction       Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              40               0              0
##   Iris-versicolor           0              40              0
##   Iris-virginica            0               0             40
##
```

```
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9697, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 1.0000
## Specificity                      1.0000                 1.0000
## Pos Pred Value                   1.0000                 1.0000
## Neg Pred Value                   1.0000                 1.0000
## Prevalence                       0.3333                 0.3333
## Detection Rate                   0.3333                 0.3333
## Detection Prevalence             0.3333                 0.3333
## Balanced Accuracy                1.0000                 1.0000
##                      Class: Iris-virginica
## Sensitivity                         1.0000
## Specificity                         1.0000
## Pos Pred Value                      1.0000
## Neg Pred Value                      1.0000
## Prevalence                          0.3333
## Detection Rate                      0.3333
## Detection Prevalence                0.3333
## Balanced Accuracy                   1.0000
```

We see the power of the random forest algorithm with the perfect accuracy of classification on the training set.

We now verify how it performs on the `testset` data in the real world

```
## Performance on the test set
pred_test<-predict(object = model.rf$finalModel,newdata = testset[,1:4],type="class")
confusionMatrix(pred_test,testset$Species)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              10               0              0
##   Iris-versicolor           0              10              1
##   Iris-virginica            0               0              9
##
## Overall Statistics
##
##                Accuracy : 0.9667
##                  95% CI : (0.8278, 0.9992)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 2.963e-13
##
##                   Kappa : 0.95
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 1.0000
## Specificity                      1.0000                 0.9500
## Pos Pred Value                   1.0000                 0.9091
## Neg Pred Value                   1.0000                 1.0000
## Prevalence                       0.3333                 0.3333
## Detection Rate                   0.3333                 0.3333
## Detection Prevalence             0.3333                 0.3667
## Balanced Accuracy                1.0000                 0.9750
##                      Class: Iris-virginica
## Sensitivity                         0.9000
## Specificity                         1.0000
## Pos Pred Value                      1.0000
## Neg Pred Value                      0.9524
## Prevalence                          0.3333
## Detection Rate                      0.3000
## Detection Prevalence                0.3000
## Balanced Accuracy                   0.9500
```

Even though we were able to improve our accuracy on the training set, on the test set we still are misclassifying two observations. We will next consider another class of Algorithm which we hope would improve our testset accuracy

Gradient Boosting Method

This class of ML Algorithms uses a technique called `BOOSTING` where we still grow decision classification trees, but each successive tree is grown with an intent to classify the missclassified data from the previous tree correctly.

A very good discussion of this method is available here on AnalyticsVidhya

As before the method to build a model using this algorithm is very similar to the ones we used before. As in the previous cases a lot of the difference in actual implementation of the algo's comes from the various parameters used to optimize the algo's. `CARET` package also helps standardize the optimization steps such as `CROSS VALIDATION` ,`HYPER PARAMETER TURNING`,`GRID SEARCH` etc.

```
library(caret)
## install.packages("gbm")
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

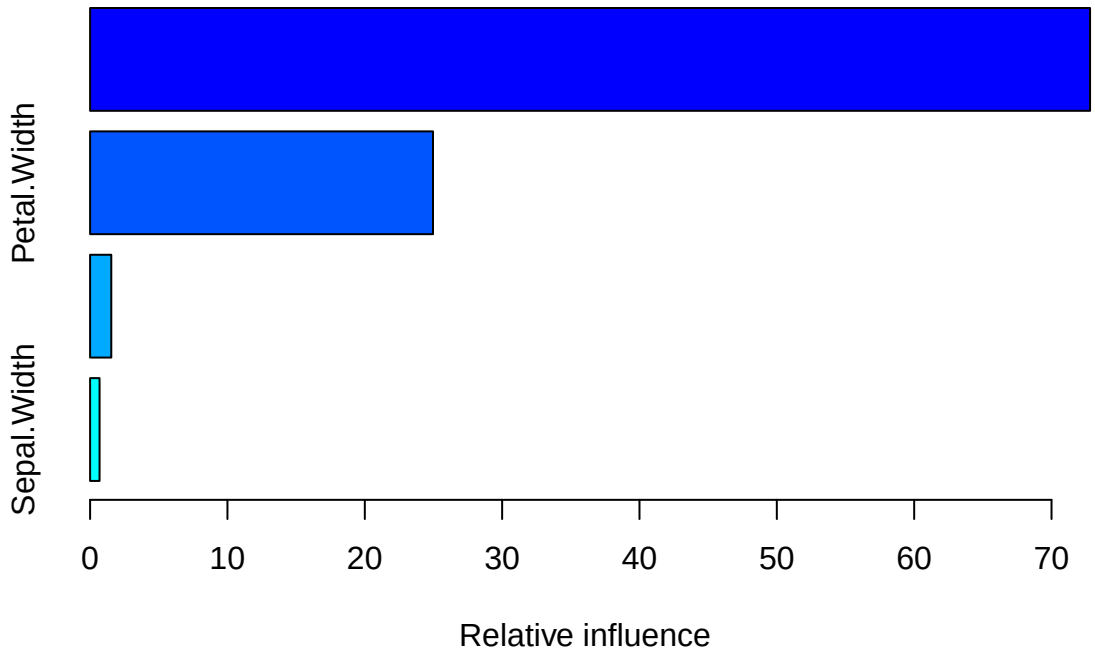
```r
set.seed(1000)

# Fit the model
model.gbm<-train(x = trainset[,1:4],y = trainset[,5], method = "gbm",metric = "Accuracy",verbose=FALSE)
```

```
## Loading required package: plyr
```

```r
# Print the model
print(model.gbm)
```

```
## Stochastic Gradient Boosting
##
## 120 samples
##   4 predictor
##   3 classes: 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 120, 120, 120, 120, 120, 120, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9547488  0.9313289
##   1                  100      0.9519537  0.9270553
##   1                  150      0.9536550  0.9296157
##   2                   50      0.9574368  0.9354323
##   2                  100      0.9565160  0.9340487
##   2                  150      0.9574251  0.9354312
##   3                   50      0.9544762  0.9308559
##   3                  100      0.9564035  0.9337737
##   3                  150      0.9581539  0.9364286
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```r
summary(model.gbm)
```




```
##                       var    rel.inf
## Petal.Length Petal.Length 72.8055964
## Petal.Width   Petal.Width 24.9648539
## Sepal.Length Sepal.Length  1.5423194
## Sepal.Width   Sepal.Width  0.6872303
```

```
## Verify the accuracy on the training set

pred<-predict(object = model.gbm,newdata = trainset[,1:4])
confusionMatrix(pred,trainset$Species)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              40               0              0
##   Iris-versicolor           0              40              0
##   Iris-virginica            0               0             40
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9697, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 1.0000
## Specificity                      1.0000                 1.0000
```

```
## Pos Pred Value                    1.0000                 1.0000
## Neg Pred Value                    1.0000                 1.0000
## Prevalence                        0.3333                 0.3333
## Detection Rate                    0.3333                 0.3333
## Detection Prevalence              0.3333                 0.3333
## Balanced Accuracy                 1.0000                 1.0000
##                       Class: Iris-virginica
## Sensitivity                          1.0000
## Specificity                          1.0000
## Pos Pred Value                       1.0000
## Neg Pred Value                       1.0000
## Prevalence                           0.3333
## Detection Rate                       0.3333
## Detection Prevalence                 0.3333
## Balanced Accuracy                    1.0000
```

Let us now verify how it performs on the `testset` data in the real world

```
## Performance on the test set
pred_test<-predict(object = model.gbm,newdata = testset[,1:4])
confusionMatrix(pred_test,testset$Species)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              10               0              0
##   Iris-versicolor           0               9              2
##   Iris-virginica            0               1              8
##
## Overall Statistics
##
##                Accuracy : 0.9
##                  95% CI : (0.7347, 0.9789)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 1.665e-10
##
##                   Kappa : 0.85
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                       Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                       1.0000                 0.9000
## Specificity                       1.0000                 0.9000
## Pos Pred Value                    1.0000                 0.8182
## Neg Pred Value                    1.0000                 0.9474
## Prevalence                        0.3333                 0.3333
## Detection Rate                    0.3333                 0.3000
## Detection Prevalence              0.3333                 0.3667
## Balanced Accuracy                 1.0000                 0.9000
##                       Class: Iris-virginica
## Sensitivity                          0.8000
## Specificity                          0.9500
## Pos Pred Value                       0.8889
```

```
## Neg Pred Value                0.9048
## Prevalence                    0.3333
## Detection Rate                0.2667
## Detection Prevalence          0.3000
## Balanced Accuracy             0.8750
```

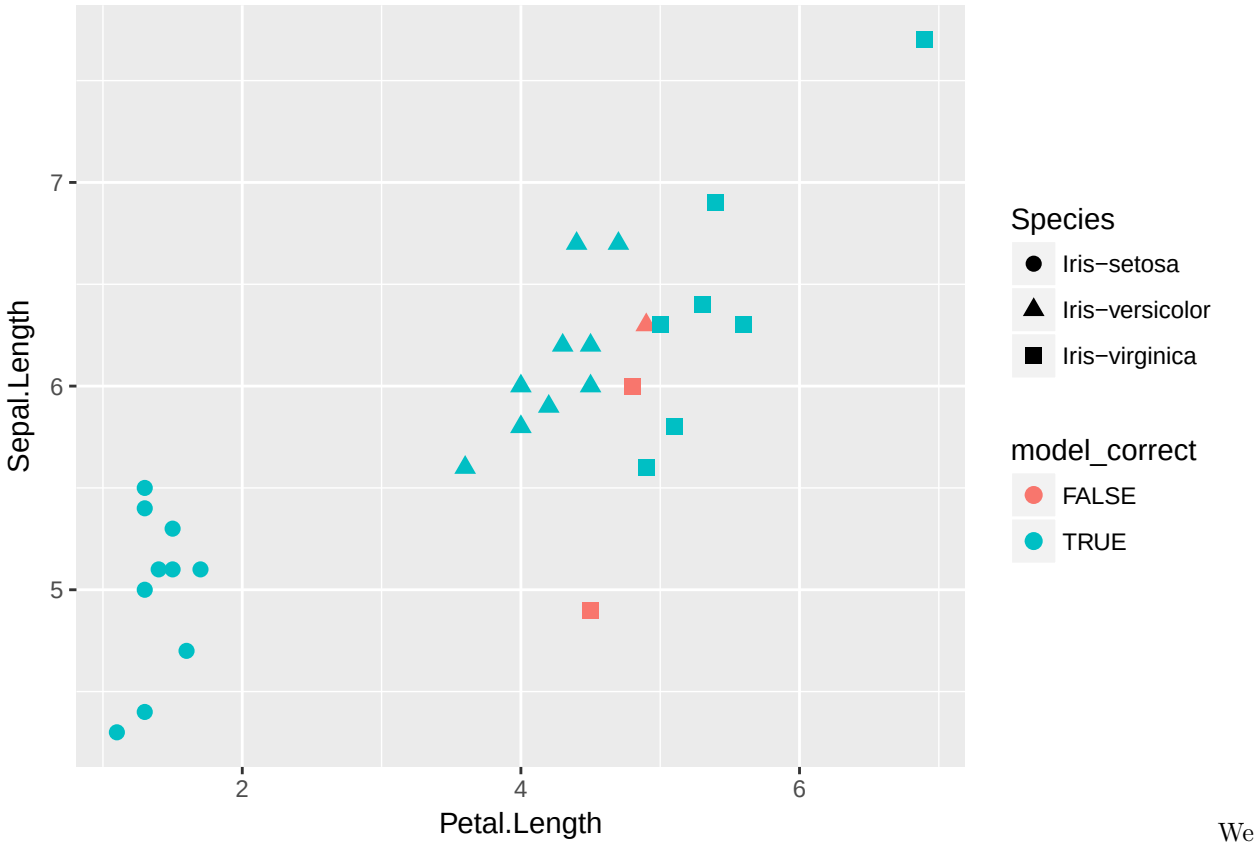
We still end up with the same testset accuracy despite using more and more sophisticated Algos!

Plotting Misclassified Observations

Let us now plot the observations and see which flowers were actually misclassified. We can then figure out if there is a reason why all these models misclassify those two flowers.

Please note that changing the model now after we have had it look at the test data isn't recommended or proper datascience

```
model_correct <- testset$Species==pred_test1

misclassify<- ggplot(testset) +
    geom_point(aes(Petal.Length, Sepal.Length, colour = model_correct, shape = Species), size = 2.5) +
    labs(x = "Petal.Length", y = "Sepal.Length")

print(misclassify)
```



We can see they are misclassified because the `Petal.Length` attributes for the two species `versicolor` and `virginica` are very close to each other and not clearly separable. This is why most of our models got them wrong.

Let us now try another totally different class of model and see if we can improve on our results

K Means Clustering Model

This model belongs to the super class of models which follow a machine learning technique called Unsupervised Learning.

Machine learning (ML) itself can broadly be divided into two different fields:

- *Supervised ML* defined as a set of tools used for prediction (linear model, logistic regression, linear discriminant analysis, classification trees, support vector machines and more)
- *Unsupervised ML*, also known as `clustering`, is an exploratory data analysis technique used for identifying groups (i.e clusters) in the data set of interest. Each group contains observations with similar profile according to a specific criteria. Similarity between observations is defined using some inter-observation distance measures including Euclidean and correlation-based distance measures.

We will also not use our favorite `CARET` package here, but we will come back to it in our final model later.

```
# Since Kmeans is a random start algo, we need to set the seed to ensure reproduceability
set.seed(20)
```

Since we know there are 3 classes, we begin with 3 centers. Typically when we don't know the classification of the dataset, we begin with a best estimate of number of classes we think it contains.

Also since k-means assigns the centroids randomly we specify `nstart` as 20 to run the algo 20 times with 20 random starting sets of centroids and then pick the best of those 20

```
irisCluster <- kmeans(iris[, 1:4], centers = 3, nstart = 20)
irisCluster
```

```
## K-means clustering with 3 clusters of sizes 38, 62, 50
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     6.850000    3.073684     5.742105    2.071053
## 2     5.901613    2.748387     4.393548    1.433871
## 3     5.006000    3.428000     1.462000    0.246000
##
## Clustering vector:
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1
## [106] 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1
## [141] 1 1 2 1 1 1 2 1 1 2
##
## Within cluster sum of squares by cluster:
## [1] 23.87947 39.82097 15.15100
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

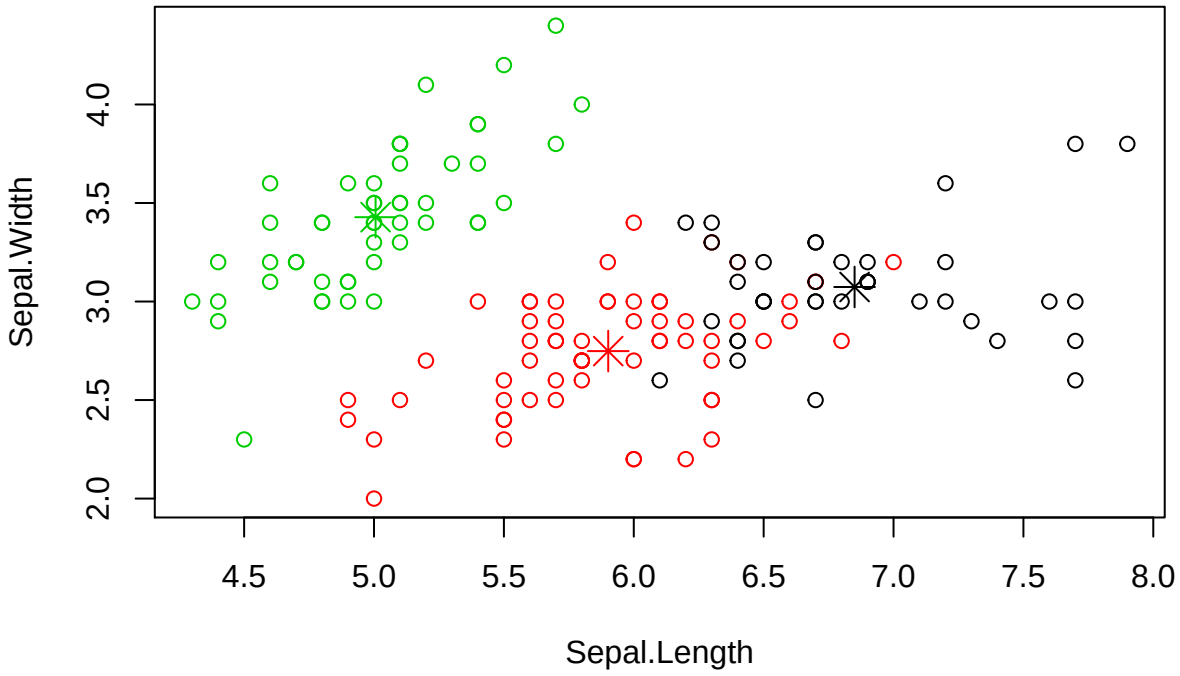
```
# Check the classification accuracy
table(irisCluster$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          2        36
```

```
##    2      0         48         14
##    3     50          0          0
```

We can also plot the clusters and their centroids to see how the algo clustered the observations

```
plot(iris[c("Sepal.Length", "Sepal.Width")], col=irisCluster$cluster)
points(irisCluster$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=8, cex=2)
```



Linear Discriminant Analysis

Our last model will be the model to explain which Ronald Fisher originally used the Iris dataset. More information about this algorithm can be obtained here.

Linear Discriminant Analysis model is generally used for small data sets which would otherwise suffer from small sample bias in other models. Other classes of models might not be able to pick the trends in the data correctly, so for smaller datasets, a probability based classifier such as bayesian classifiers and lda are more suitable

Linear Discriminant Analysis (LDA) is also most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting ("curse of dimensionality") and also reduce computational costs. A good discussion of this (although in Python) can be found here

To use this model on our dataset we can go back to using the `CARET` package

```
library(caret)
#install.packages("MASS")
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.4.3
```

```
set.seed(1000)

# Fit the model
model.lda<-train(x = trainset[,1:4],y = trainset[,5], method = "lda",metric = "Accuracy")
```

```r
# Print the model
print(model.lda)
```

```
## Linear Discriminant Analysis
##
## 120 samples
##   4 predictor
##   3 classes: 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 120, 120, 120, 120, 120, 120, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9735939  0.9598427
```

```r
## Verify the accuracy on the training set

pred<-predict(object = model.lda,newdata = trainset[,1:4])
confusionMatrix(pred,trainset$Species)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              40               0              0
##   Iris-versicolor           0              39              1
##   Iris-virginica            0               1             39
##
## Overall Statistics
##
##                Accuracy : 0.9833
##                  95% CI : (0.9411, 0.998)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.975
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 0.9750
## Specificity                      1.0000                 0.9875
## Pos Pred Value                   1.0000                 0.9750
## Neg Pred Value                   1.0000                 0.9875
## Prevalence                       0.3333                 0.3333
## Detection Rate                   0.3333                 0.3250
## Detection Prevalence             0.3333                 0.3333
## Balanced Accuracy                1.0000                 0.9812
##                      Class: Iris-virginica
## Sensitivity                         0.9750
## Specificity                         0.9875
## Pos Pred Value                      0.9750
```

```
## Neg Pred Value                      0.9875
## Prevalence                          0.3333
## Detection Rate                      0.3250
## Detection Prevalence                0.3333
## Balanced Accuracy                   0.9812
```

Let us now verify how it performs on the `testset` data in the real world

```
## Performance on the test set
pred_test<-predict(object = model.lda,newdata = testset[,1:4])
confusionMatrix(pred_test,testset$Species)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Iris-setosa Iris-versicolor Iris-virginica
##   Iris-setosa              10               0              0
##   Iris-versicolor           0              10              0
##   Iris-virginica            0               0             10
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.8843, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 4.857e-15
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Iris-setosa Class: Iris-versicolor
## Sensitivity                      1.0000                 1.0000
## Specificity                      1.0000                 1.0000
## Pos Pred Value                   1.0000                 1.0000
## Neg Pred Value                   1.0000                 1.0000
## Prevalence                       0.3333                 0.3333
## Detection Rate                   0.3333                 0.3333
## Detection Prevalence             0.3333                 0.3333
## Balanced Accuracy                1.0000                 1.0000
##                      Class: Iris-virginica
## Sensitivity                         1.0000
## Specificity                         1.0000
## Pos Pred Value                      1.0000
## Neg Pred Value                      1.0000
## Prevalence                          0.3333
## Detection Rate                      0.3333
## Detection Prevalence                0.3333
## Balanced Accuracy                   1.0000
```

Finally we get a better classification than what we had previously!

Summarizing the Models

We have tried a few models on the Iris dataset which hopefully gives a broad overview of the variety of algorithms and models possible in R. As a final step we can summarize the results of our analysis by presenting the training set results for the models we employed

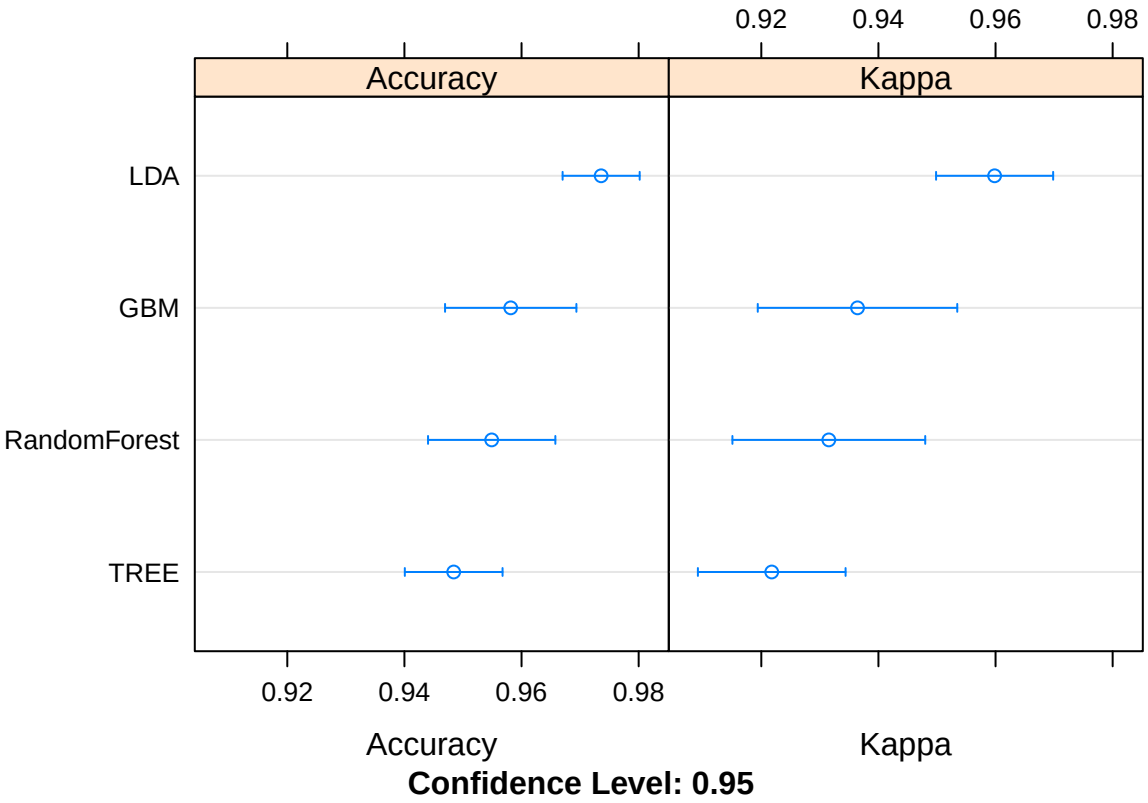
This sort of summary can be used to select the model just based on the training set data

```
# summarize accuracy of models
results <- resamples(list(TREE=model.rpart, RandomForest=model.rf, GBM=model.gbm, LDA=model.lda))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: TREE, RandomForest, GBM, LDA
## Number of resamples: 25
##
## Accuracy
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## TREE         0.9090909 0.9333333 0.9545455 0.9484135 0.9574468 0.9791667
## RandomForest 0.9000000 0.9361702 0.9565217 0.9549063 0.9772727 1.0000000
## GBM          0.9111111 0.9347826 0.9565217 0.9581539 0.9777778 1.0000000
## LDA          0.9318182 0.9574468 0.9777778 0.9735939 0.9800000 1.0000000
##              NA's
## TREE            0
## RandomForest    0
## GBM             0
## LDA             0
##
## Kappa
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## TREE         0.8620690 0.8998516 0.9283388 0.9217799 0.9358799 0.9686888
## RandomForest 0.8510131 0.9027586 0.9346591 0.9315307 0.9652510 1.0000000
## GBM          0.8668639 0.9020582 0.9341917 0.9364286 0.9665179 1.0000000
## LDA          0.8967944 0.9349030 0.9665179 0.9598427 0.9695493 1.0000000
##              NA's
## TREE            0
## RandomForest    0
## GBM             0
## LDA             0
```

Plotting the results

```
dotplot(results)
```



Further Reading Assignment

A technique being widely employed in Machine Learning to improve the accuracy of models is Ensembling or Stacking of different models to produce a super model. *Random Forest and Gradient Boosted Method use a similar such technique employed on similar kind of trees.*

Ensembling is a very popular and effective technique that is very frequently used by data scientists for beating the accuracy benchmark of even the best of individual algorithms. More often than not it's the winning recipe in hackathons.

- A starting point for learning about **Model Ensembling** can be found here
- A package which makes this convenient in R while using `CARET` can be found here