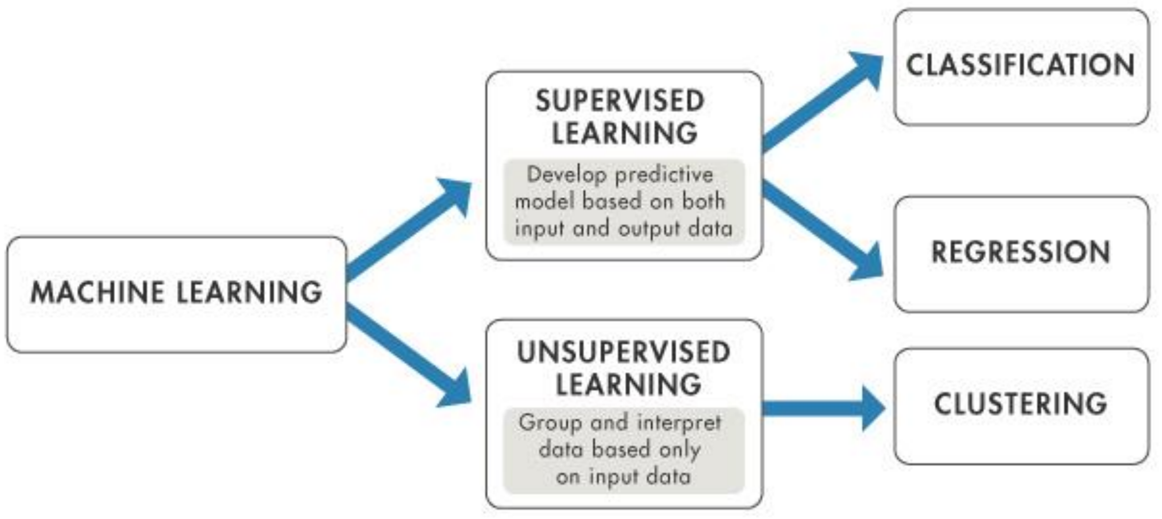
Adroitsoft
Enabling with great ingenuity

# Machine Learning - Technical Documentation

## Machine Learning

Machine learning is the science of getting computers to act without being explicitly programmed.


MACHINE LEARNING
SUPERVISED LEARNING
Develop predictive model based on both input and output data
UNSUPERVISED LEARNING
Group and interpret data based only on input data
CLASSIFICATION
REGRESSION
CLUSTERING

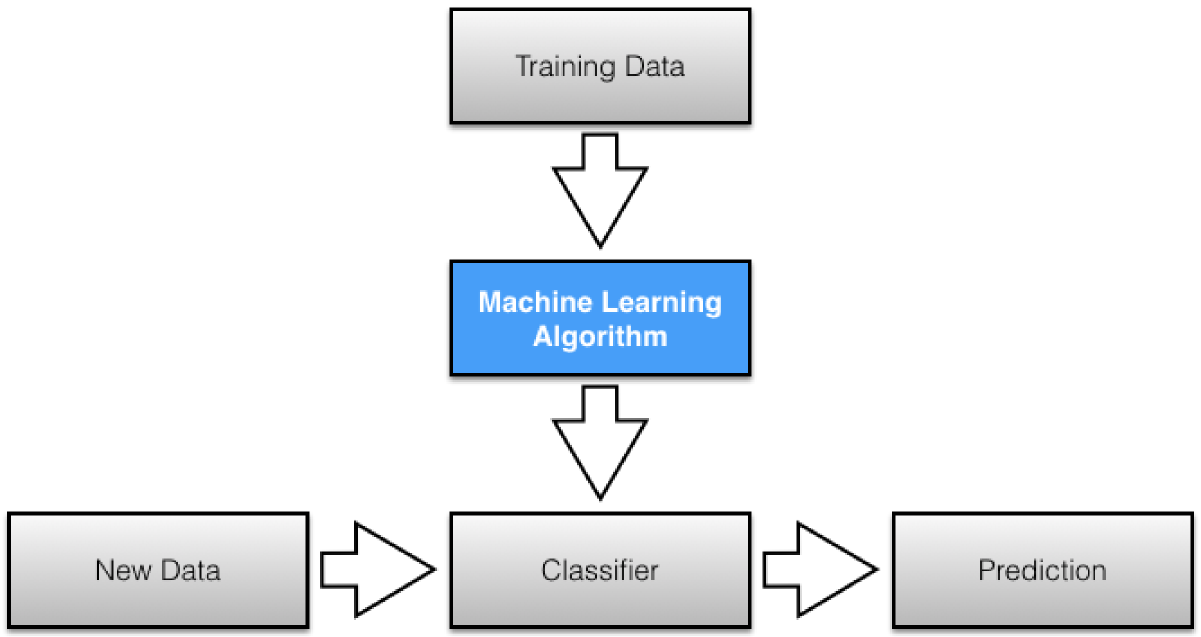
machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank, and computer vision.

## Problem Statement

We need to build the model through the initial data which is more accurate so that we can use in the production for prediction purpose.

Initial data will be divided into to training and testing . Training data will used to build the model and testing data will used to test the model built model.

**Predicting who has diabetes or not with the available initial womens data .**



## Success Criteria

Machine learning has different methods to do predictions and every algorithm has its own meaning and procedure. There are many algorithms available on internet which are defined and designed by illustrious R & D scientists so called as pre defined algorithms .

We can also make our user defined algorithms but it needs high exposure on science area  and more R &D exposure is required while making user defined algorithms.

We are using the pre defined  algorithms to get more accuracy in the built model and how we gonna use it ?

- Data Sampling
- Divide the data into training and testing
- Apply the algorithm such as random forest (randomForest) or decision tree (rpart)
- Build the model through training data
- Plot the model so that you get more accurate visualization
- Now test it through your testing data and check how good the built model has been fit
- Do prediction through the original data
- Print the table to check accuracy +

## Available Data -

**Predicting who has diabetes or not with the available initial womens data .**

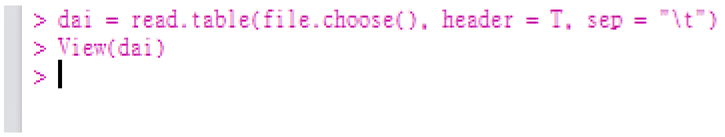
We have womens data with us which contains 768 observations and 9 variables .

Variables are -

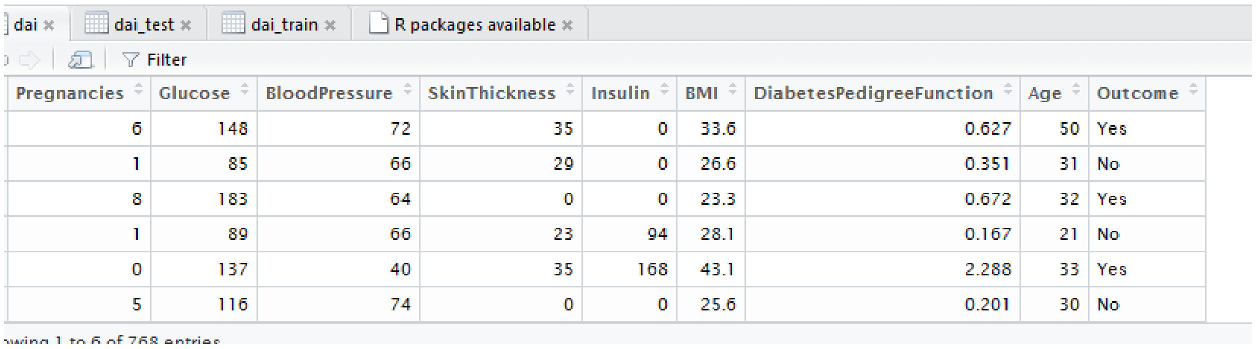
1. Pregnancies
2. Glucose
3. BLoodPressure
4. SkinThickness
5. Insulin
6. BMI
   DiabetesPedigreeFunction
7. Age
8. Outcome - Yes (1) or No (0)

## Technology and Tool - R language and R studio for statistical and visual analysis

**Load the data -**



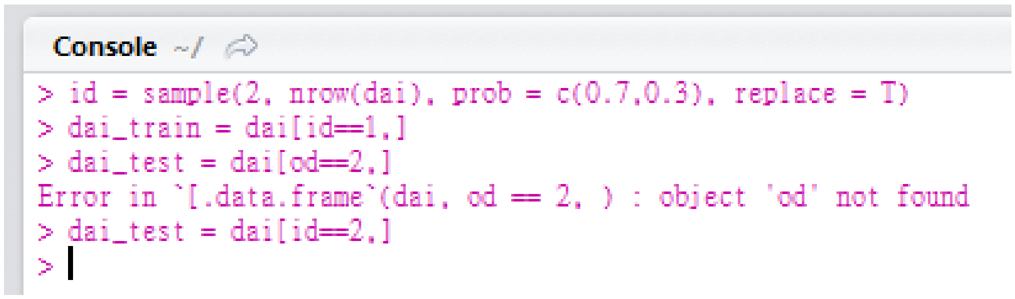
```
> dai = read.table(file.choose(), header = T, sep = "\t")
> View(dai)
> 
```



| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | Yes |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | No |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | Yes |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | No |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | Yes |
| 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | No |

owing 1 to 6 of 768 entries

**Dividing the data into 70 % and 30 %**



```
> id = sample(2, nrow(dai), prob = c(0.7,0.3), replace = T)
> dai_train = dai[id==1,]
> dai_test = dai[od==2,]
Error in `[.data.frame`(dai, od == 2, ) : object 'od' not found
> dai_test = dai[id==2,]
> 
```

# Approaches -

## Supervised learning -

This algorithm consist of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, Decision Tree, Random Forest,

### Classifications -

Popularly known as predictive analysis | identifying which set of category a new observation belongs. Supervised learning model as the classifier already has a classified examples

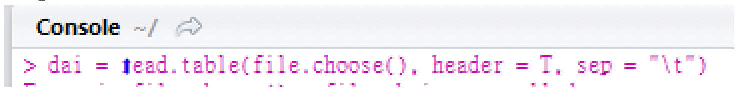
### Algorithm - Decision Tree :

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.

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.

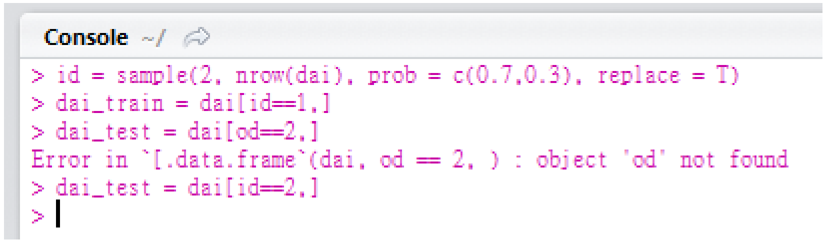
**Illustration with diabetes data -**

**Import the data in R studio**



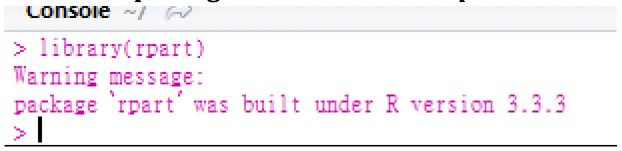
```
> dai = read.table(file.choose(), header = T, sep = "\t")
```

**Do the sampling and divide the data into aspected ration (70 % and 30 %)**



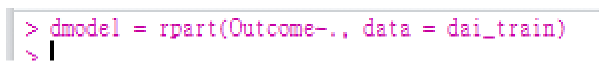
```
> id = sample(2, nrow(dai), prob = c(0.7,0.3), replace = T)
> dai_train = dai[id==1,]
> dai_test = dai[od==2,]
Error in `[.data.frame`(dai, od == 2, ) : object 'od' not found
> dai_test = dai[id==2,]
> 
```

**Install the package for decision tree *rpart***



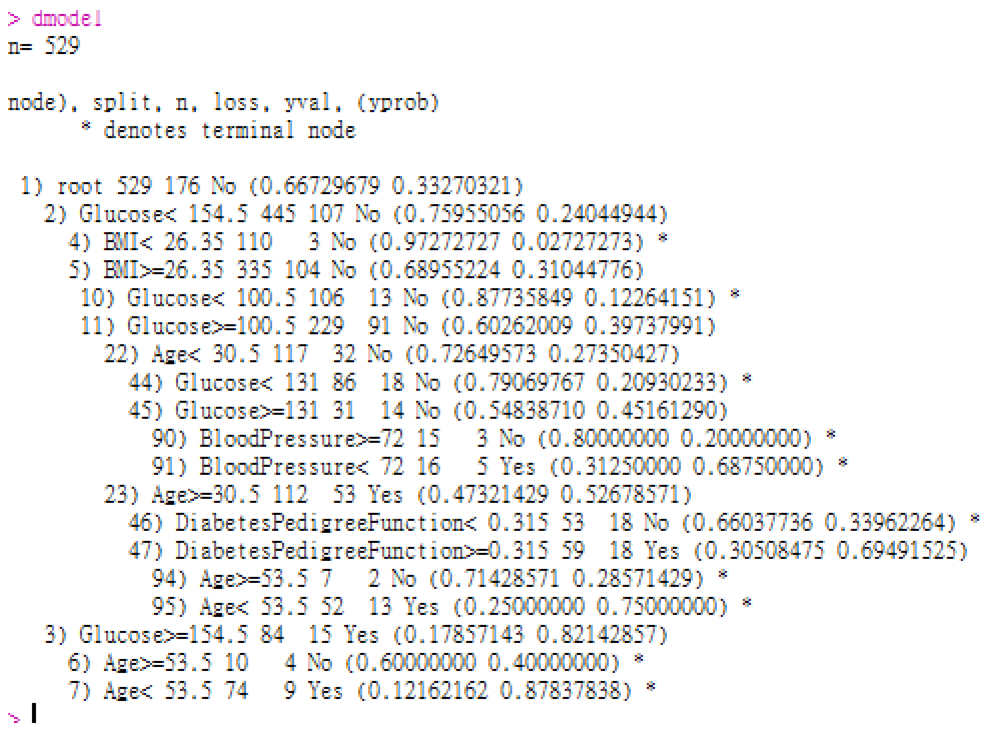
```
> library(rpart)
Warning message:
package 'rpart' was built under R version 3.3.3
> 
```

**Build the model with training data and based on the Outcome column you need to build the model**



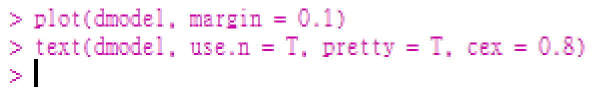
```
> dmodel = rpart(Outcome~., data = dai_train)
> 
```

**Review the model**

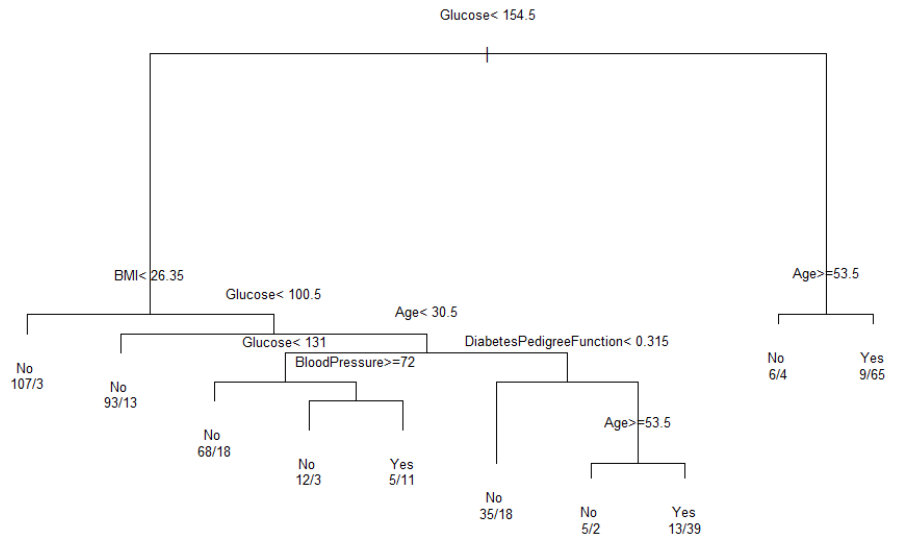


```
> dmodel
n= 529

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 529 176 No (0.66729679 0.33270321)
   2) Glucose< 154.5 445 107 No (0.75955056 0.24044944)
     4) BMI< 26.35 110   3 No (0.97272727 0.02727273) *
     5) BMI>=26.35 335 104 No (0.68955224 0.31044776)
      10) Glucose< 100.5 106  13 No (0.87735849 0.12264151) *
      11) Glucose>=100.5 229  91 No (0.60262009 0.39737991)
        22) Age< 30.5 117  32 No (0.72649573 0.27350427)
          44) Glucose< 131 86  18 No (0.79069767 0.20930233) *
          45) Glucose>=131 31  14 No (0.54838710 0.45161290)
            90) BloodPressure>=72 15   3 No (0.80000000 0.20000000) *
            91) BloodPressure< 72 16   5 Yes (0.31250000 0.68750000) *
        23) Age>=30.5 112  53 Yes (0.47321429 0.52678571)
          46) DiabetesPedigreeFunction< 0.315 53  18 No (0.66037736 0.33962264) *
          47) DiabetesPedigreeFunction>=0.315 59  18 Yes (0.30508475 0.69491525)
            94) Age>=53.5 7   2 No (0.71428571 0.28571429) *
            95) Age< 53.5 52  13 Yes (0.25000000 0.75000000) *
   3) Glucose>=154.5 84  15 Yes (0.17857143 0.82142857)
     6) Age>=53.5 10   4 No (0.60000000 0.40000000) *
     7) Age< 53.5 74   9 Yes (0.12162162 0.87837838) *
> 
```

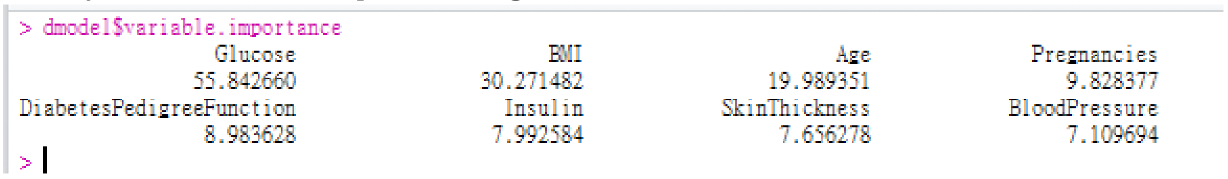
**Visualize the built model**



```
> plot(dmodel, margin = 0.1)
> text(dmodel, use.n = T, pretty = T, cex = 0.8)
> 
```



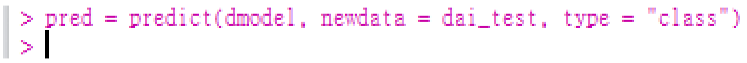
**Identify which is the most important or significant variable for decision tree**



```
> dmodel$variable.importance
                 Glucose                      BMI                      Age              Pregnancies 
               55.842660                30.271482                19.989351                 9.828377 
DiabetesPedigreeFunction                  Insulin            SkinThickness            BloodPressure 
                8.983628                 7.992584                 7.656278                 7.109694 
> 
```

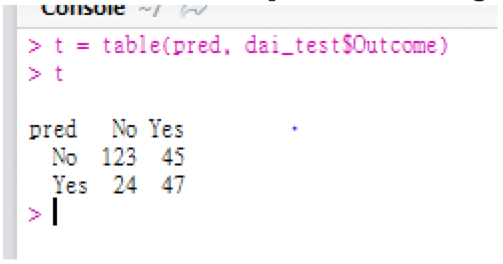
It indicates that glucose is most significant factor to evaluate who  has diabetes or not.
Glucose >154.5 then will check age and age >=53.5 then person has a high risk of diabetes and if age <53.5 then will check the body mass index which shows that person is pretty much healthier person.

**Test the model through  testing  data and validate how good it has been fit**



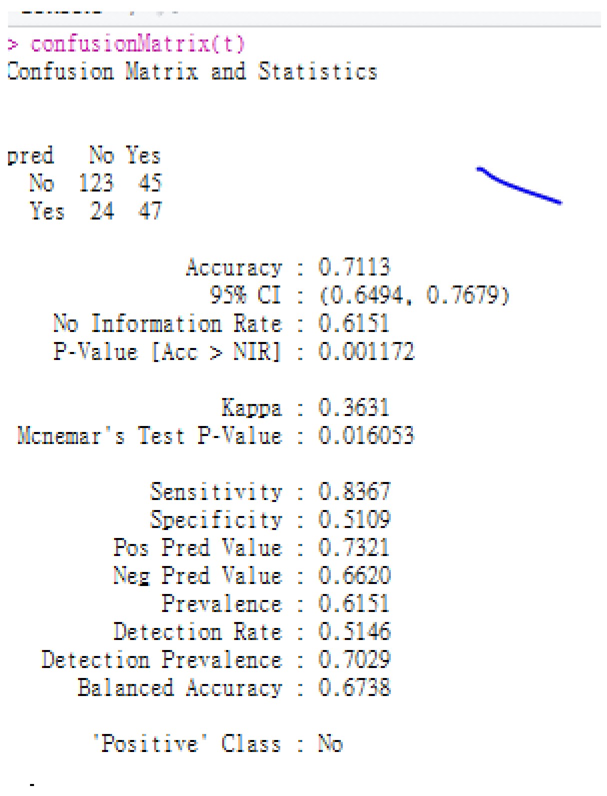
```
> pred = predict(dmodel, newdata = dai_test, type = "class")
> 
```

**Print the table to compare with the original outcome parameter**



```
> t = table(pred, dai_test$Outcome)
> t

pred   No Yes
  No  123  45
  Yes  24  47
> 
```

**Print confusion matrix  to  see the accuracy of the model**



```
> confusionMatrix(t)
Confusion Matrix and Statistics


pred   No Yes
  No  123  45
  Yes  24  47

               Accuracy : 0.7113
                 95% CI : (0.6494, 0.7679)
    No Information Rate : 0.6151
    P-Value [Acc > NIR] : 0.001172

                  Kappa : 0.3631
 Mcnemar's Test P-Value : 0.016053

            Sensitivity : 0.8367
            Specificity : 0.5109
         Pos Pred Value : 0.7321
         Neg Pred Value : 0.6620
             Prevalence : 0.6151
         Detection Rate : 0.5146
   Detection Prevalence : 0.7029
      Balanced Accuracy : 0.6738

       'Positive' Class : No
```

Clearly  this model has a accuracy of 71% which is not good enough for using in production.
This data can be used for other classification algorithms like Random Forest to find if that gives better accuracy.

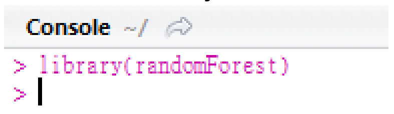
## Random Forest  -

In the Random Forest algorithm,  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.

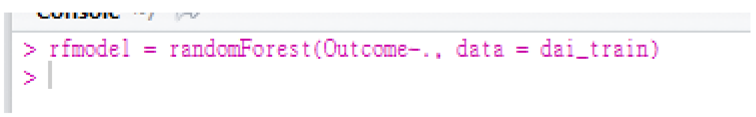
**Illustration with diabetes data -**

**Install the library**



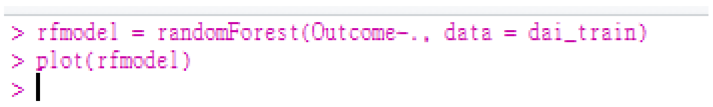
```
> library(randomForest)
>
```

**Create the model using the randomForest function in R**

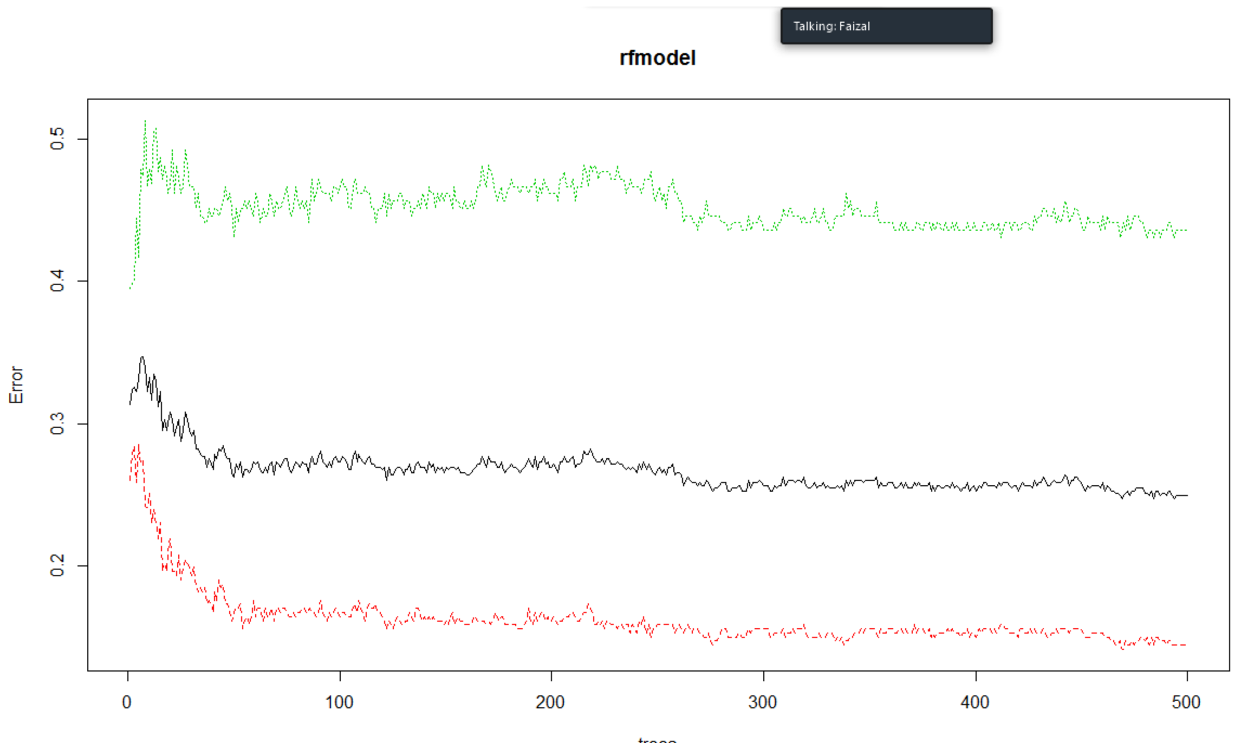


```
> rfmodel = randomForest(Outcome~., data = dai_train)
> 
```

**View the model**

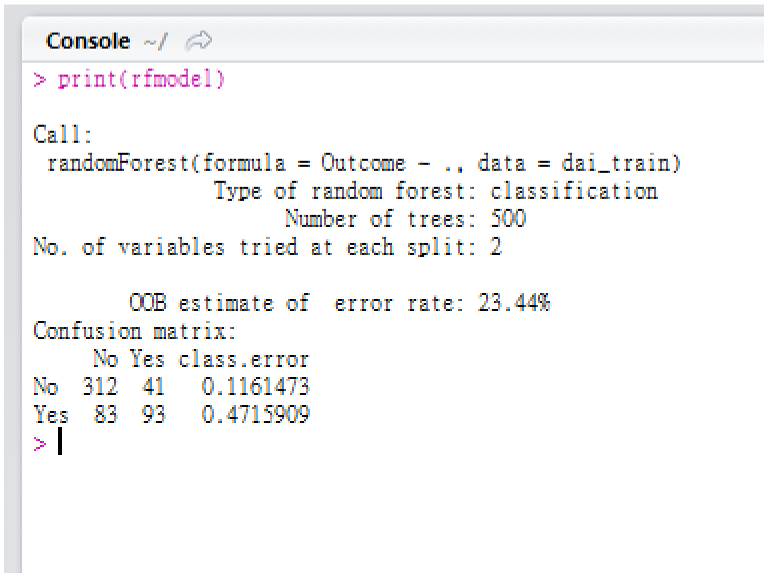


```
> rfmodel = randomForest(Outcome~., data = dai_train)
> plot(rfmodel)
> 
```



This graph is actually states the error rate here
As the number of trees grow, this error initially goes down and then it becomes more or less constant

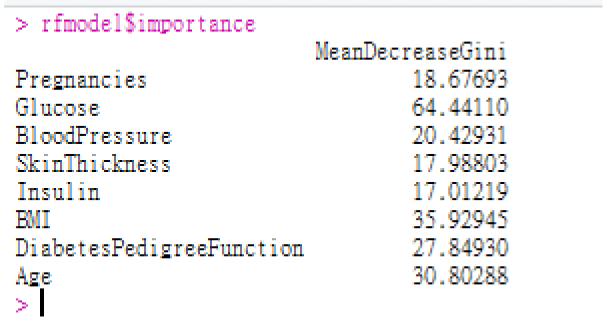
Reference - [https://www.youtube.com/watch?v=dJclNIN-TPo&t=602s](https://www.youtube.com/watch?v=dJclNIN-TPo&t=602s)



```
> print(rfmodel)

Call:
 randomForest(formula = Outcome ~ ., data = dai_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 23.44%
Confusion matrix:
     No Yes class.error
No  312  41   0.1161473
Yes  83  93   0.4715909
>
```

Here you can see 500 trees are generated and every node has 2 daughter nodes . Out of the bag error is 23.44 % shows the in accuracy of  Random Forest algorithm is slightly better than Decision Tree.

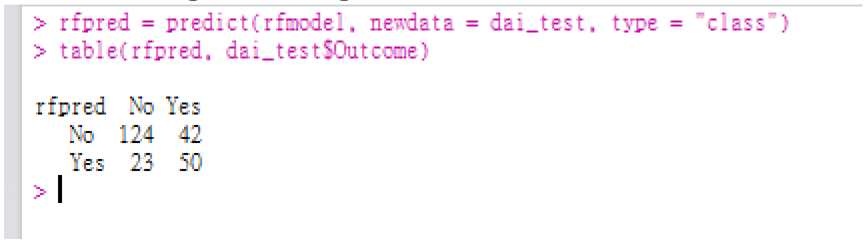
**Verify which is the most significant factor or variable for making decision.**



```
> rfmodel$importance
                         MeanDecreaseGini
Pregnancies                      18.67693
Glucose                          64.44110
BloodPressure                    20.42931
SkinThickness                    17.98803
Insulin                          17.01219
BMI                              35.92945
DiabetesPedigreeFunction         27.84930
Age                              30.80288
>
```

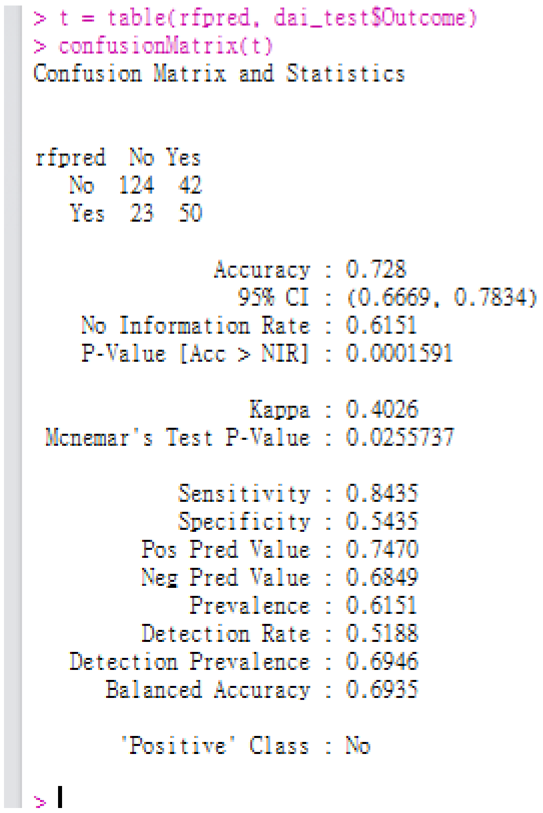
it demonstrate Glucose and BMI are the most important variable for making decision.

**Validate  through the testing data and view the table to check the error and accuracy of the model**



```
> rfpred = predict(rfmodel, newdata = dai_test, type = "class")
> table(rfpred, dai_test$Outcome)

rfpred  No Yes
   No  124  42
   Yes  23  50
>
```

**Build the confusion matrix**



```
> t = table(rfpred, dai_test$Outcome)
> confusionMatrix(t)
Confusion Matrix and Statistics


rfpred  No Yes
   No  124  42
   Yes  23  50

               Accuracy : 0.728
                 95% CI : (0.6669, 0.7834)
    No Information Rate : 0.6151
    P-Value [Acc > NIR] : 0.0001591

                  Kappa : 0.4026
 Mcnemar's Test P-Value : 0.0255737

            Sensitivity : 0.8435
            Specificity : 0.5435
         Pos Pred Value : 0.7470
         Neg Pred Value : 0.6849
             Prevalence : 0.6151
         Detection Rate : 0.5188
   Detection Prevalence : 0.6946
      Balanced Accuracy : 0.6935

       'Positive' Class : No

>
```

Here the model shows it has accuracy of 73 % based on the current data. Which is slightly better than classification algorithm outcome of the same data.

THis is not a good enough accuracy and need to continuously test the data through another algorithm like Logistic Regression.

## Logistic Regression -

- Regression analysis is a predictive modelling technique
- Estimates the relation between a dependent (target) and an independent variable (predictor)
- Logit model is a regression model where dependent variable is categorical.
  Categorical - variable that has a fix values such YES or NO , 0 or 1
  Dependent variable - Y= f(x) i.e Y is dependent on x .

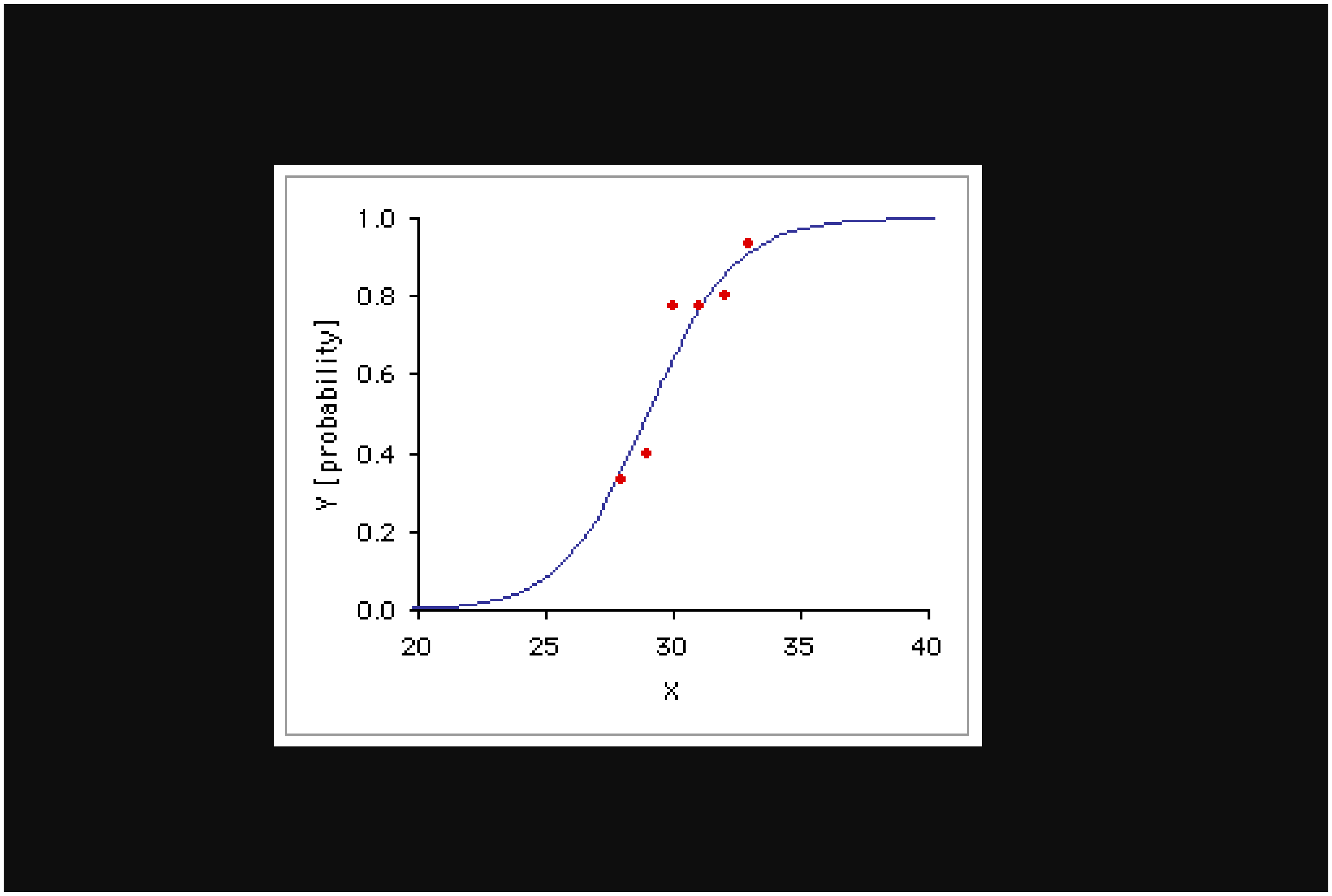
**What happens in logistic regression ?**

Calculate the probabilities , what is probability of Y being 1 based on that will take a call what will be the threshold and whenever the value is above that threshold we convert it to 1 and below threshold we convert it to 0.

**Why Logistic Regression ?**

Whenever the outcome of the dependent variable (Y) is  categorical /discrete  like 0  or 1, YEs or NO , we use this A,B or C regression. When we say discrete means the vale is fixed and can either be 0 or 1 or Yes or No or A,B or C.

**Basic graph of logistic regression -**



This graph is known as S curve or Sigmoid Curve , As we can see Y value is 0 (Y=0) for certain values of X  and our Y value is  also 1 (Y=1)for certain values of X also.  After 20, The values are becoming 1 i.e Y is becoming 1 and this transition has to be represented by a curve , Hence came up with this curve .

In this regression how do we decide whether the value will be 1 or 0  When we are predicting it through a model . So logistic regression will give us the probability so it will let us know what are the chances of Y being 1 .

Basically we have to come up with this kind of curve and having this kind of curve we should have the linear equation.

To come up with the linear equation  we compare our equation to the straight line linear equation .

***Y  = C + B1X1 + B2X2          Range of Y is  -(infinity) to (infinity)***

Let us try to reduce the Logistic Regression from this equation

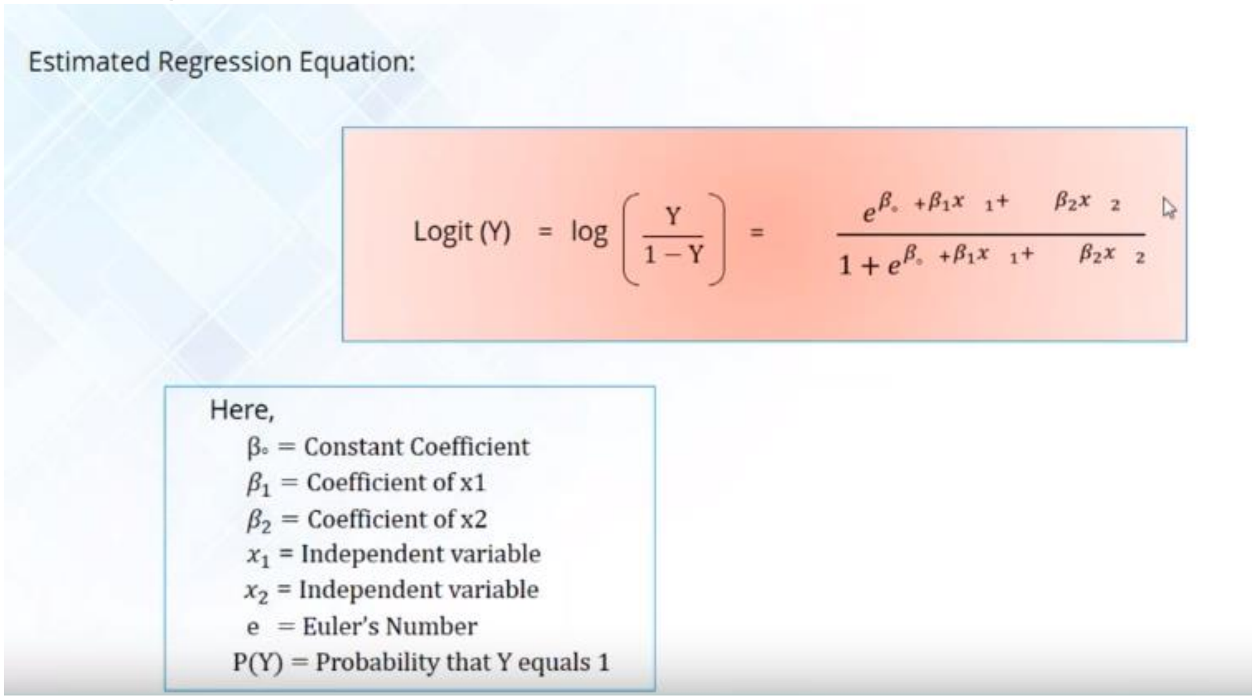
***Y  = C + B1X1 + B2X2          In Logistic Regression Y can only be 0 or 1***

Now get the range of Y between 0 and infinity, let's transform that

***Y/1-Y    i.e Y =0 | 0 and Y =1 | infinity               Now we have the range between 0 and infinity .***

Let us transform it further, to get the range between **-**(infinity) and (infinity)
***log(Y/1-Y) = log(Y/1-Y) = C +B1X1 + B2X2+ ……..***
Hence, it can be compared now with the straight line and hence it becomes a linear equation.

**Estimated Regression Equation -**

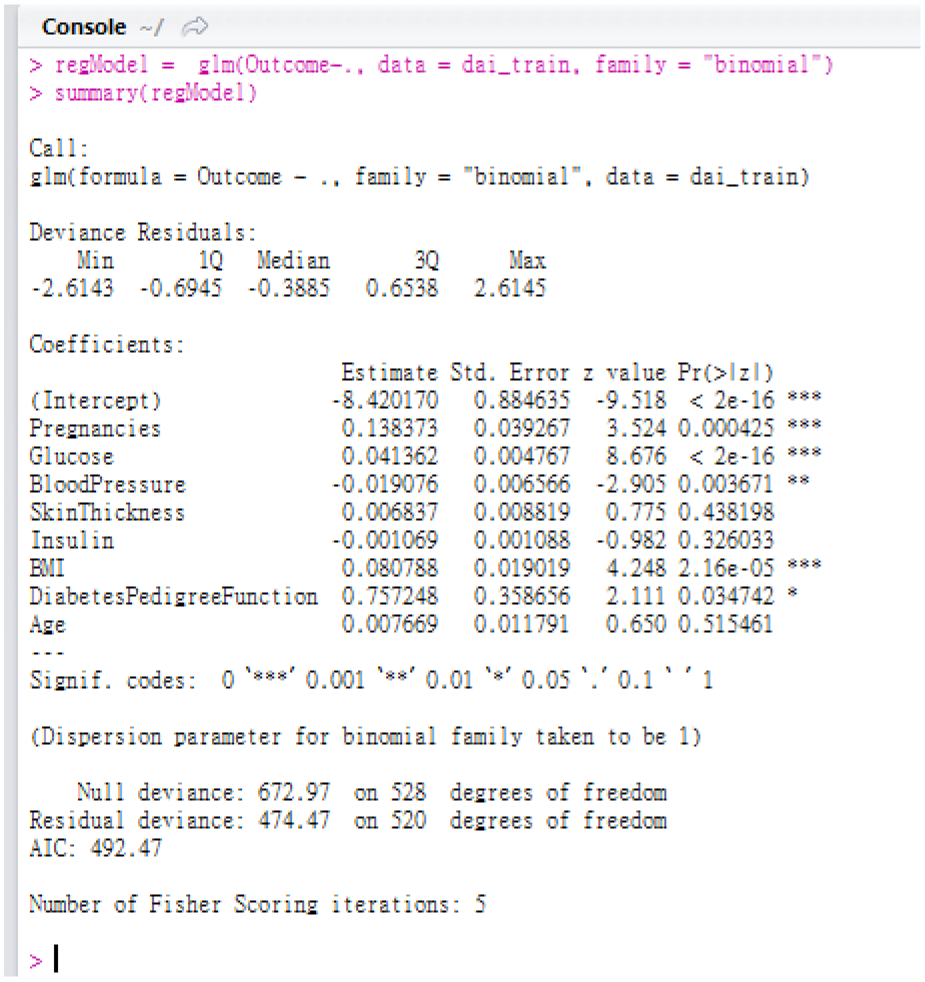


Estimated Regression Equation:

$$\text{Logit (Y)} = \log\left(\frac{Y}{1-Y}\right) = \frac{e^{\beta_\circ + \beta_1 x_1 + \beta_2 x_2}}{1 + e^{\beta_\circ + \beta_1 x_1 + \beta_2 x_2}}$$

Here,
$\beta_\circ$ = Constant Coefficient
$\beta_1$ = Coefficient of x1
$\beta_2$ = Coefficient of x2
$x_1$ = Independent variable
$x_2$ = Independent variable
e = Euler's Number
P(Y) = Probability that Y equals 1

**Illustration with the Diabetes Data**



```
Console ~/
> regModel =  glm(Outcome~., data = dai_train, family = "binomial")
> summary(regModel)

Call:
glm(formula = Outcome ~ ., family = "binomial", data = dai_train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.6143  -0.6945  -0.3885   0.6538   2.6145  

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)              -8.420170   0.884635  -9.518  < 2e-16 ***
Pregnancies               0.138373   0.039267   3.524 0.000425 ***
Glucose                   0.041362   0.004767   8.676  < 2e-16 ***
BloodPressure            -0.019076   0.006566  -2.905 0.003671 ** 
SkinThickness             0.006837   0.008819   0.775 0.438198    
Insulin                  -0.001069   0.001088  -0.982 0.326033    
BMI                       0.080788   0.019019   4.248 2.16e-05 ***
DiabetesPedigreeFunction  0.757248   0.358656   2.111 0.034742 *  
Age                       0.007669   0.011791   0.650 0.515461    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 672.97  on 528  degrees of freedom
Residual deviance: 474.47  on 520  degrees of freedom
AIC: 492.47

Number of Fisher Scoring iterations: 5

> 
```

Reference - **https://datascienceplus.com/perform-logistic-regression-in-r/**

So glm( Generalized Linear Model) function is used to build the regression model in R and family of Y is binomial i.e it can only two values such as Yes or No or 0 or 1 .
Summary of the model (regModel) -

The significant code that your R provides you basically it will tell us how much significant the particular independent variable is

*** 99.9 % confident - Your model is 99.9 % confident that your specific independent variable or value is significant to the model . It means it will add on to the accuracy of the model.

** 99 % confident - Shows your model is 99 % confident
* 95 % confident - Shows your model is 95 % confident
. 90 % confident - Shows your model is 90 % confident.
Basically we are optimizing our model - with the summary of regModel . If record is not significant we cant remove the record straight away.

Null Deviance - Deviance that you get from actual value of your data set so basically model is 672.97 units deviant when it is NULL.

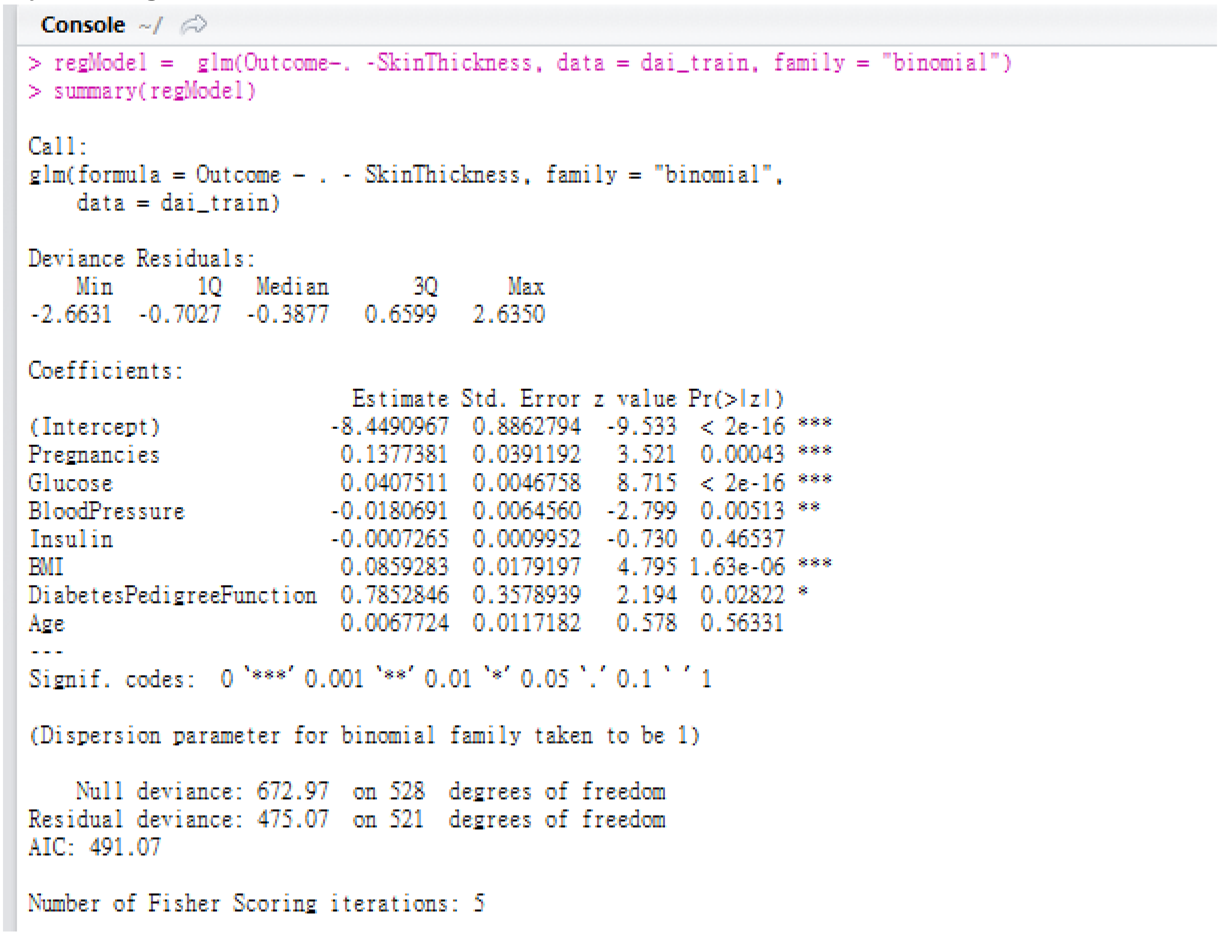
Null means when we are not using any independent  variable of the data set and only using the intercept.

Residual Deviance - when you include your independent variable in your model - if you using the independent variable that means that you are making your model accurate .

AIC - this value is as minimum as possible and this is helpful when you are actually removing not necessary independent variable .

**Optimization of Model (regModel) -**

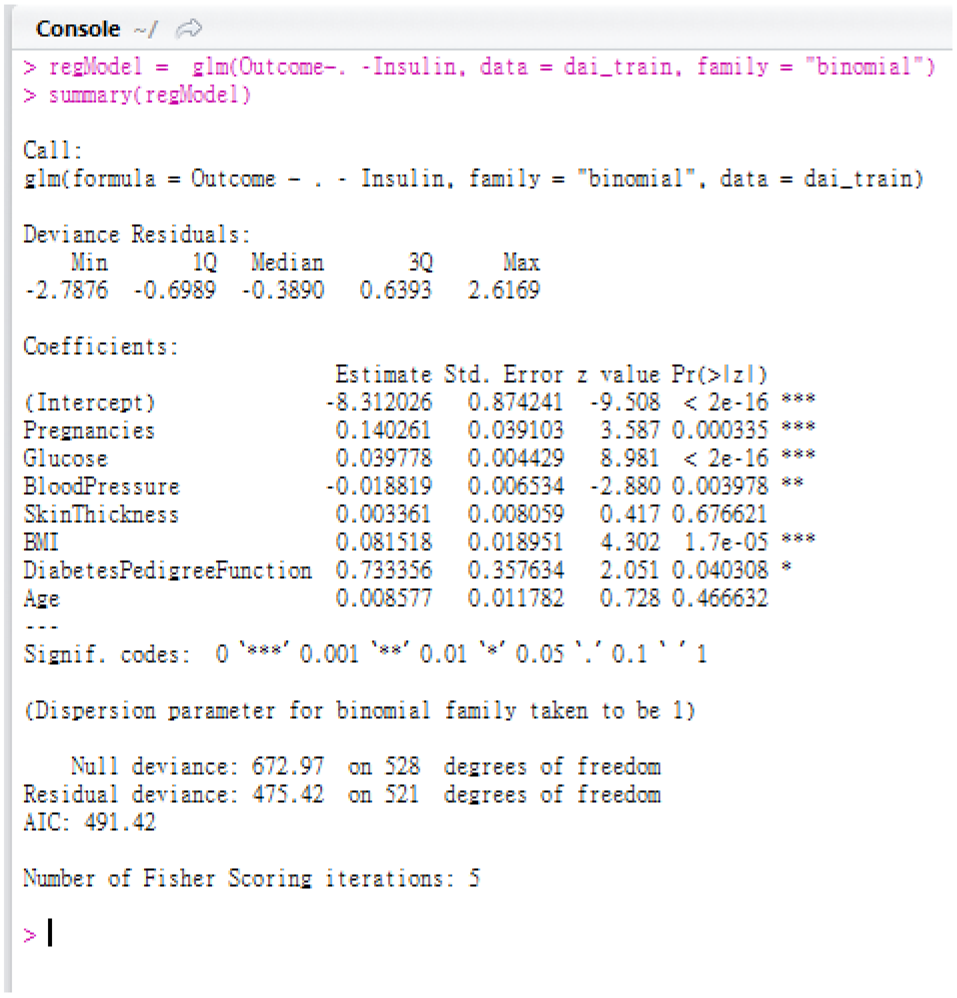
We will check which independent variable is not significant to our model and we found that SkinThickness, Insulin and Age are not the significant variables.
We will remove this independent variable one by one and check what are the difference will get in the values of Residual Deviance and AIC.
So basically our Residual Deviance will not increase and AIC should decrease.
Try removing the SkinThickness variable



```
> regModel =  glm(Outcome~. -SkinThickness, data = dai_train, family = "binomial")
> summary(regModel)

Call:
glm(formula = Outcome ~ . - SkinThickness, family = "binomial",
    data = dai_train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6631  -0.7027  -0.3877   0.6599   2.6350

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -8.4490967  0.8862794  -9.533  < 2e-16 ***
Pregnancies              0.1377381  0.0391192   3.521  0.00043 ***
Glucose                  0.0407511  0.0046758   8.715  < 2e-16 ***
BloodPressure           -0.0180691  0.0064560  -2.799  0.00513 **
Insulin                 -0.0007265  0.0009952  -0.730  0.46537
BMI                      0.0859283  0.0179197   4.795 1.63e-06 ***
DiabetesPedigreeFunction 0.7852846  0.3578939   2.194  0.02822 *
Age                      0.0067724  0.0117182   0.578  0.56331
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 672.97  on 528  degrees of freedom
Residual deviance: 475.07  on 521  degrees of freedom
AIC: 491.07

Number of Fisher Scoring iterations: 5
```

As per the summary we got the know that AIC has been increased which shouldn't be and Residual Deviance has been increased which clearly shows that SkinThickness is also  significant variable.

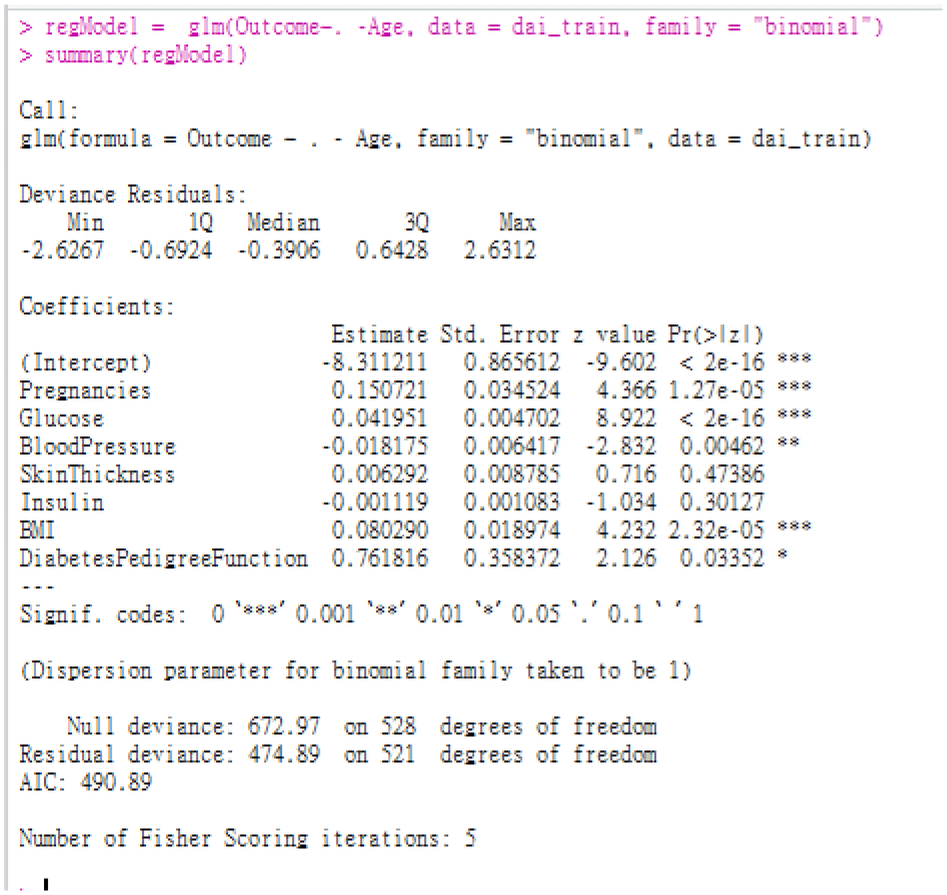
B1Let’s try to remove Insulin variable and view the summary again .



```
> regModel =  glm(Outcome~. -Insulin, data = dai_train, family = "binomial")
> summary(regModel)

Call:
glm(formula = Outcome ~ . - Insulin, family = "binomial", data = dai_train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.7876  -0.6989  -0.3890   0.6393   2.6169  

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)              -8.312026   0.874241  -9.508  < 2e-16 ***
Pregnancies               0.140261   0.039103   3.587 0.000335 ***
Glucose                   0.039778   0.004429   8.981  < 2e-16 ***
BloodPressure            -0.018819   0.006534  -2.880 0.003978 ** 
SkinThickness             0.003361   0.008059   0.417 0.676621    
BMI                       0.081518   0.018951   4.302  1.7e-05 ***
DiabetesPedigreeFunction  0.733356   0.357634   2.051 0.040308 *  
Age                       0.008577   0.011782   0.728 0.466632    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 672.97  on 528  degrees of freedom
Residual deviance: 475.42  on 521  degrees of freedom
AIC: 491.42

Number of Fisher Scoring iterations: 5

> 
```

We found that Residual Deviance has been increased and AIC has been decreased which concludes that Insulin is also an significant variable.

Lets try to remove Age variable and view the summary once gain

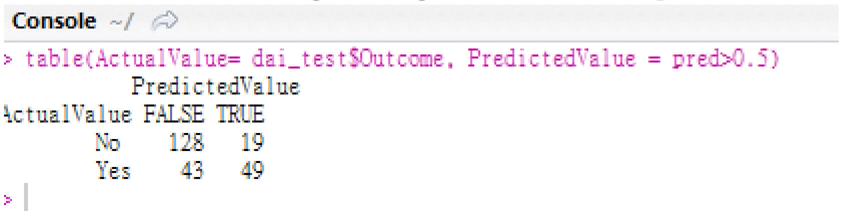


```
> regModel =  glm(Outcome~. -Age, data = dai_train, family = "binomial")
> summary(regModel)

Call:
glm(formula = Outcome ~ . - Age, family = "binomial", data = dai_train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.6267  -0.6924  -0.3906   0.6428   2.6312  

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)              -8.311211   0.865612  -9.602  < 2e-16 ***
Pregnancies               0.150721   0.034524   4.366 1.27e-05 ***
Glucose                   0.041951   0.004702   8.922  < 2e-16 ***
BloodPressure            -0.018175   0.006417  -2.832  0.00462 ** 
SkinThickness             0.006292   0.008785   0.716  0.47386    
Insulin                  -0.001119   0.001083  -1.034  0.30127    
BMI                       0.080290   0.018974   4.232 2.32e-05 ***
DiabetesPedigreeFunction  0.761816   0.358372   2.126  0.03352 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 672.97  on 528  degrees of freedom
Residual deviance: 474.89  on 521  degrees of freedom
AIC: 490.89

Number of Fisher Scoring iterations: 5
```

We found that AIC has been decreased and Residual Deviance has been increased by some .% not an significant change which can be considered . So clearly Age is not a significant variable , We can remove the Age variable here.

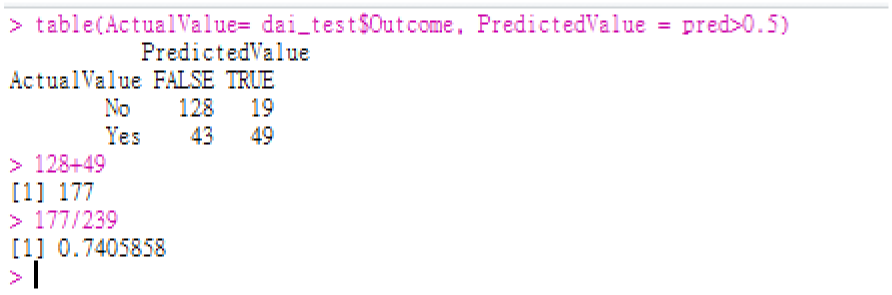
So Null Deviance shows how well the response variable is predicted by a model that includes only the intercept.
Residual Deviance shows that how well the response variable is predicted with the inclusion of independent variables.

**Validate the model through testing data and do some predictions.**



```
Console ~/
> table(ActualValue= dai_test$Outcome, PredictedValue = pred>0.5)
           PredictedValue
ActualValue FALSE TRUE
        No    128   19
        Yes    43   49
>
```

**Calculate the accuracy of the model**

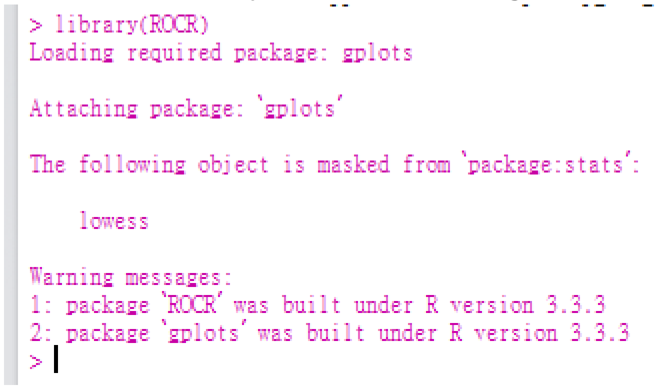


```
> table(ActualValue= dai_test$Outcome, PredictedValue = pred>0.5)
           PredictedValue
ActualValue FALSE TRUE
        No    128   19
        Yes    43   49
> 128+49
[1] 177
> 177/239
[1] 0.7405858
> 
```

So we found that our model is 74 % accurate model so we have optimized the model correctly . We assumed that our threshold is 0.5 and what if we increase our threshold and my accuracy actually increases and the wrong predicted values has to decreased.

**How to find the threshold ?**

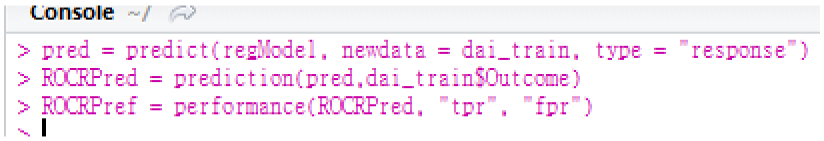
R has an inbuilt library called as ROCR through which we can find the threshold value.



```
> library(ROCR)
Loading required package: gplots

Attaching package: 'gplots'

The following object is masked from 'package:stats':

    lowess

Warning messages:
1: package 'ROCR' was built under R version 3.3.3
2: package 'gplots' was built under R version 3.3.3
> 
```

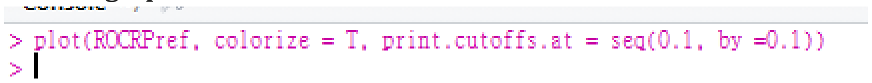
- Predicted values should be from training data set, basically calculating the threshold from training data set so that can be applied to whatever value will be passing to your model.
Threshold will be collected from your testing data set

Define the ROCRPred and ROCRPref variables . So prediction and performance are the parameters that the curve uses to actually plot the graph against the tpr (True Positive Rate) and FPR (False Positive Rate).

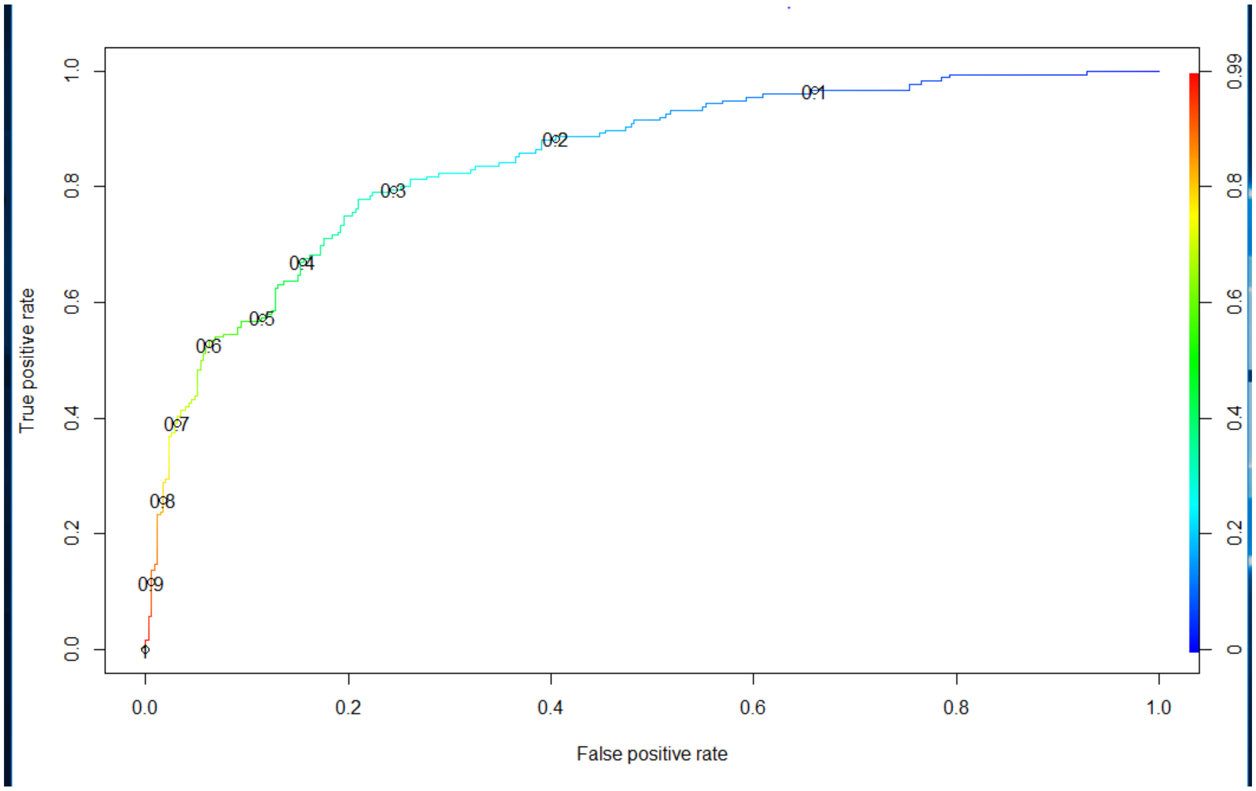


```
Console ~/
> pred = predict(regModel, newdata = dai_train, type = "response")
> ROCRPred = prediction(pred,dai_train$Outcome)
> ROCRPref = performance(ROCRPred, "tpr", "fpr")
> 
```

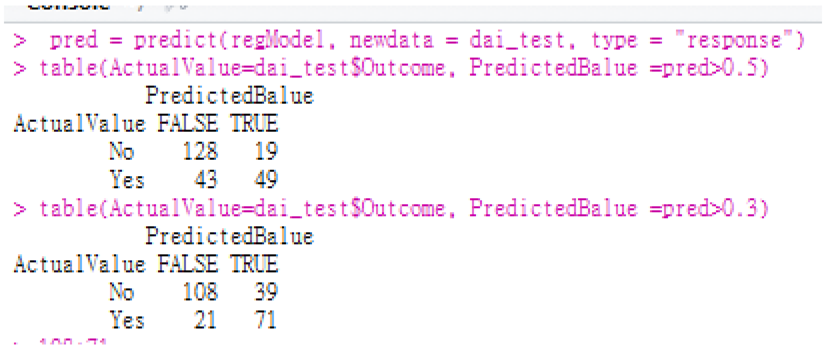
**Plot the graph**



```
> plot(ROCRPref, colorize = T, print.cutoffs.at = seq(0.1, by =0.1))
> 
```



So X axis is False Positive Rate and this should be minimum  and Y axis is True Positive Rate which should be maximum.
As we can see between 0. 3 and 0.2 there is a  lot of gap of false positive rate, so if give 0.2 as my threshold we'll be actually going towards the false positive rate and our model will go towards the lesser accuracy but 0.3 is going towards higher true positive rate at a same time it has least false positive rate also.
If you compare it with 0.5, It has least true positive rate though it has a less false positive rate as well .



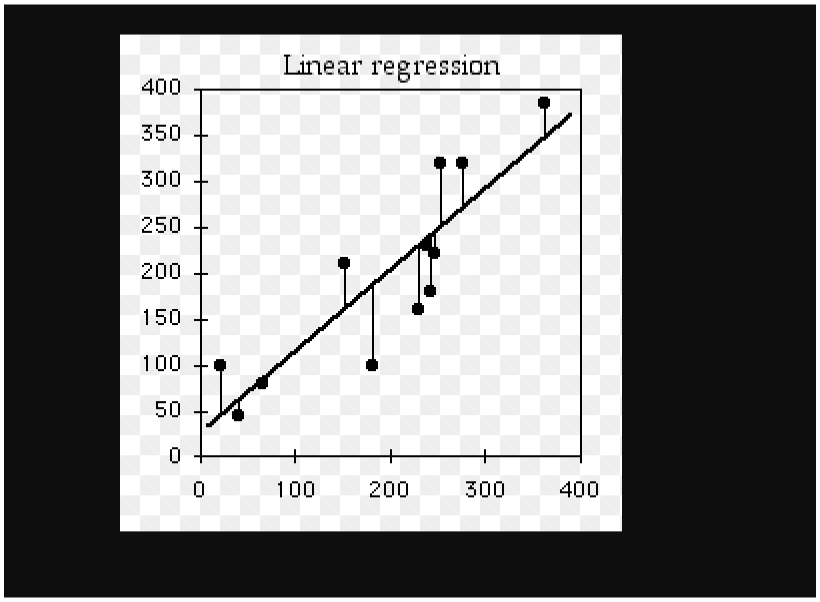
```
>  pred = predict(regModel, newdata = dai_test, type = "response")
> table(ActualValue=dai_test$Outcome, PredictedBalue =pred>0.5)
           PredictedBalue
ActualValue FALSE TRUE
        No    128   19
        Yes    43   49
> table(ActualValue=dai_test$Outcome, PredictedBalue =pred>0.3)
           PredictedBalue
ActualValue FALSE TRUE
        No    108   39
        Yes    21   71
```

Compare with the threshold 0.5 and 0.3 clearly we can see the there is no a huge change in the accuracy but at the same time my true negative value has also gone down. So whenever our threshold is 0.3 we will get the accuracy 0.748954 and at the same time our model has improved.

**Reference**
https://www.youtube.com/watch?v=Z5WKQr4H4Xk&t=408s

## Linear Regression -

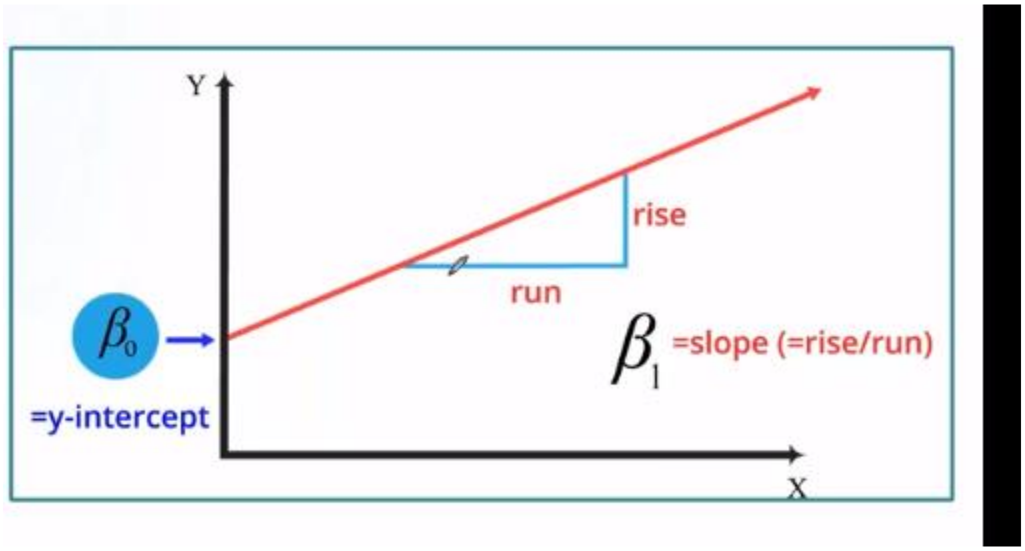
- Predictive modelling technique
- Estimates the relationship between the a dependent variable (target) and an independent variable predictor
- When there is a linear relationship between independent and dependent variables

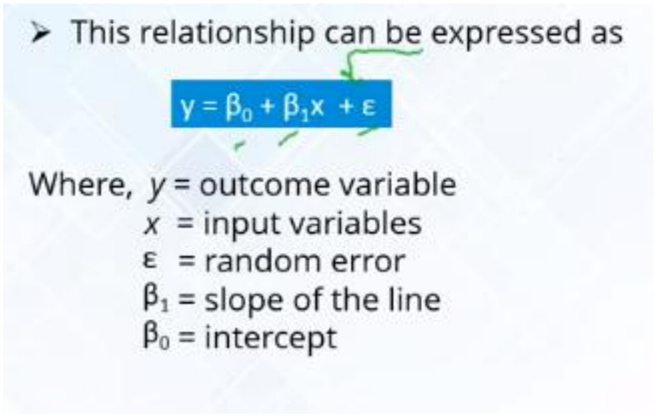


### Where to use linear regression

If the goal is prediction, forecasting or reduction , linear regression can be used to fit a predictive model to an observed data set of Y and X values. After developing such model , if the additional value of X is given without its accompanying value of Y , the fitted model can be used to make a prediction of the value of Y.

### Equation Linear Regression





➤ This relationship can be expressed as

$$y = \beta_0 + \beta_1 x + \varepsilon$$

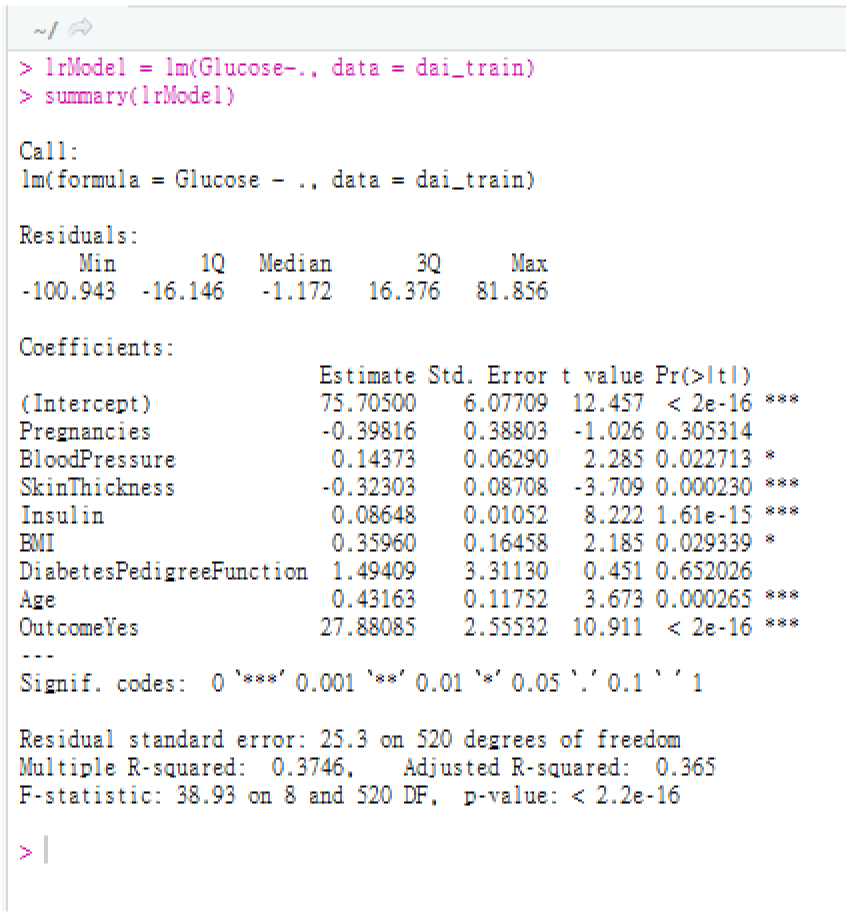
Where, $y$ = outcome variable
$x$ = input variables
$\varepsilon$ = random error
$\beta_1$ = slope of the line
$\beta_0$ = intercept

**Difference between Linear Regression and Logistic Regression**

- Linear regression requires the dependent variable to be continuous i.e. numeric values (no categories or groups) while on the other hand logistic regression requires the dependent variable to be binary - two categories only (0/1).
-

**Reference -**
http://scaryscientist.blogspot.in/2015/06/difference-between-linear-regression.html

**Illustration -**
Optimize the linear model with the lm() function



```
> lrModel = lm(Glucose~., data = dai_train)
> summary(lrModel)

Call:
lm(formula = Glucose ~ ., data = dai_train)

Residuals:
     Min       1Q   Median       3Q      Max
-100.943  -16.146   -1.172   16.376   81.856

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)              75.70500    6.07709  12.457  < 2e-16 ***
Pregnancies              -0.39816    0.38803  -1.026 0.305314
BloodPressure             0.14373    0.06290   2.285 0.022713 *
SkinThickness            -0.32303    0.08708  -3.709 0.000230 ***
Insulin                   0.08648    0.01052   8.222 1.61e-15 ***
BMI                       0.35960    0.16458   2.185 0.029339 *
DiabetesPedigreeFunction  1.49409    3.31130   0.451 0.652026
Age                       0.43163    0.11752   3.673 0.000265 ***
OutcomeYes               27.88085    2.55532  10.911  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 25.3 on 520 degrees of freedom
Multiple R-squared:  0.3746,	Adjusted R-squared:  0.365
F-statistic: 38.93 on 8 and 520 DF,  p-value: < 2.2e-16

>
```

Here we can see that Pregnancies, and DaibetesPedigreeFunction are not the significant variables for this linear model.

Residual Error - The difference between the observed value of the dependent variable ($y$) and the predicted value ($\hat{y}$) is called the **residual** ($e$).

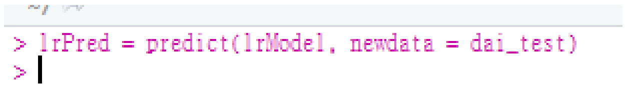
Residual = Observed value - Predicted value

$$e = y - \hat{y}$$

R - Squared Value - If your R squared value is closer to 1.0, then the linear model is best suited.

In general, the higher the R-squared, the better the model fits your data.

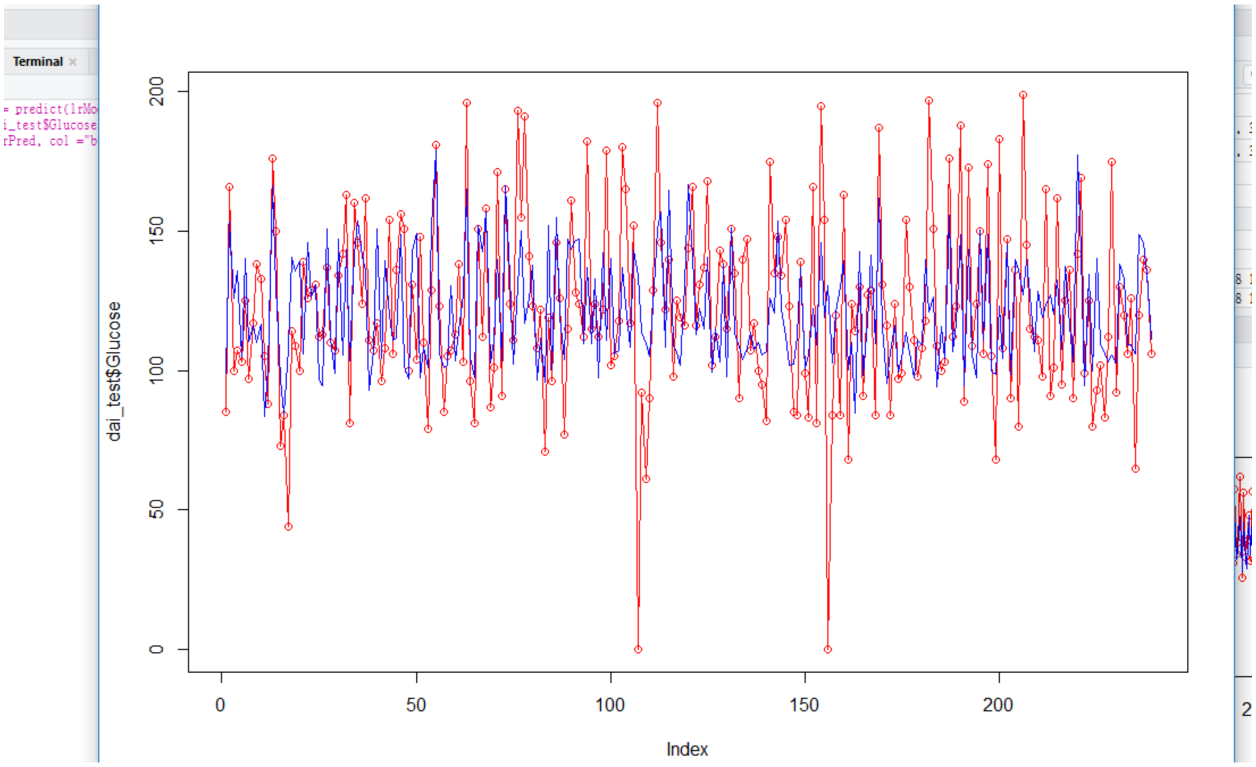
**Model Validation**



```
> lrPred = predict(lrModel, newdata = dai_test)
>
```

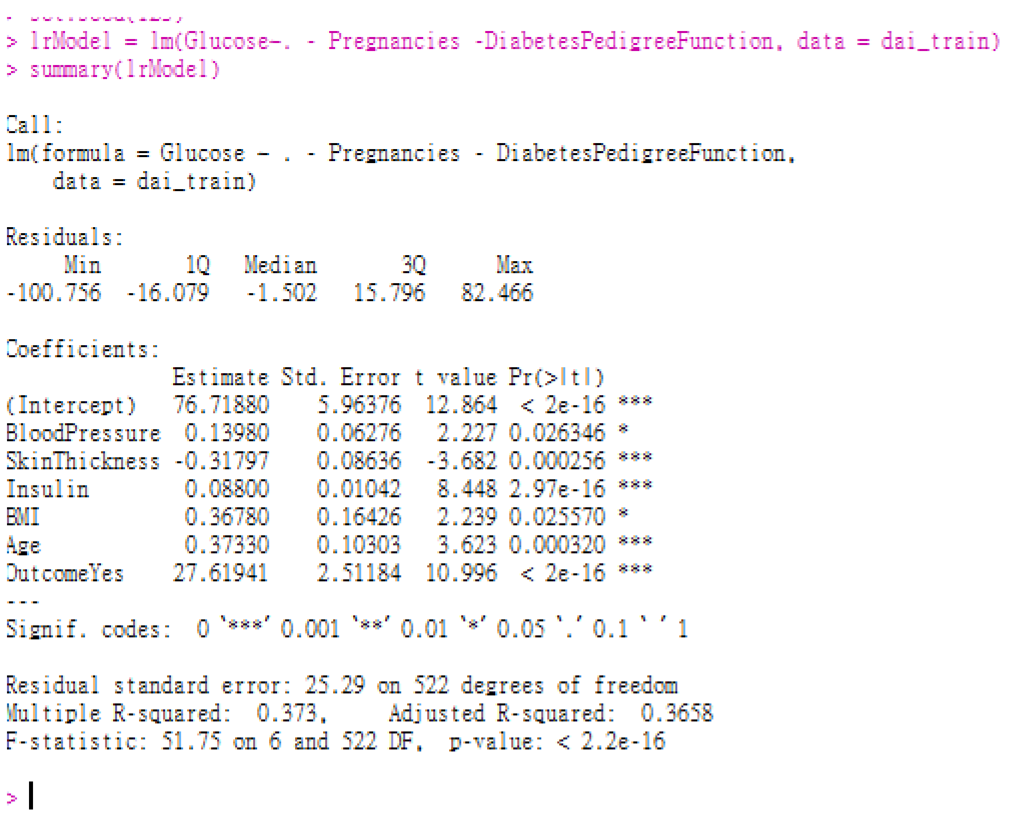
For comparing these values we can use plots

```
> plot(dai_test$Glucose,type ="o" ,lty=1.8, col ="red")
> lines(lrPred, col ="blue")
>
```



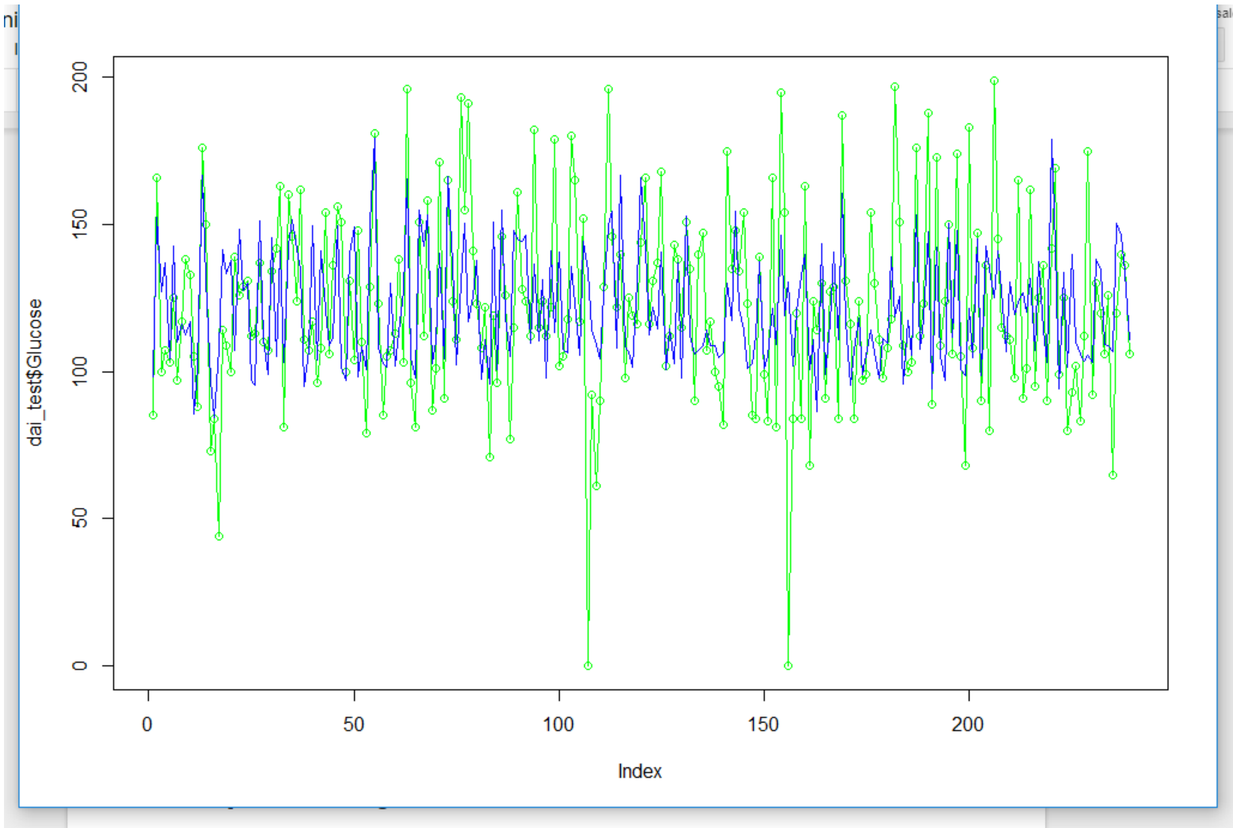
Here we have plotted the line graph where red line represent the actual values and the blue line represent the predicted model generated from the data.
As we can see from the graph most of the predictive values are overlapping the actual values.

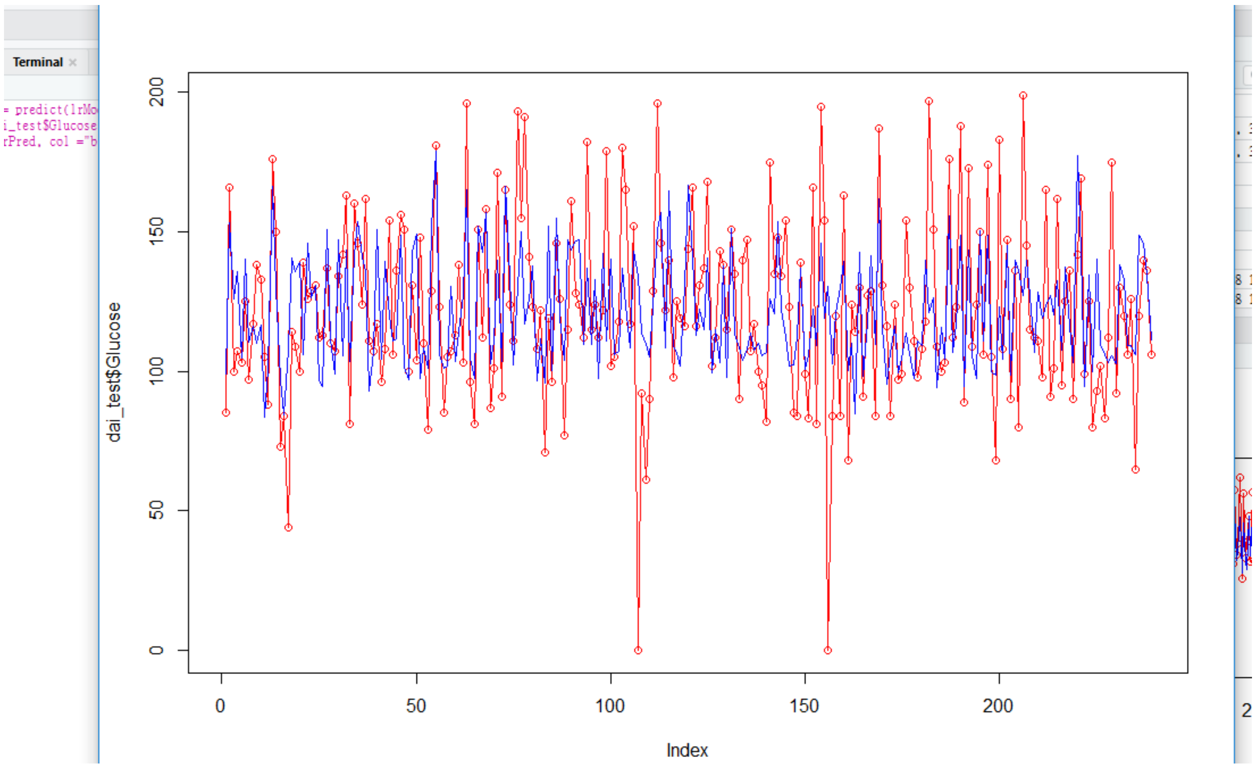
Lets try to remove those variables which are not significant to the model. As we have seen in or generated model lrModel that Pregnancies and DaibetesPedigreeFunction are not significant for this model, So will go ahead and remove that.



```
> lrModel = lm(Glucose~. - Pregnancies -DiabetesPedigreeFunction, data = dai_train)
> summary(lrModel)

Call:
lm(formula = Glucose ~ . - Pregnancies - DiabetesPedigreeFunction,
    data = dai_train)

Residuals:
     Min       1Q   Median       3Q      Max
-100.756  -16.079   -1.502   15.796   82.466

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   76.71880    5.96376  12.864  < 2e-16 ***
BloodPressure  0.13980    0.06276   2.227 0.026346 *
SkinThickness -0.31797    0.08636  -3.682 0.000256 ***
Insulin        0.08800    0.01042   8.448 2.97e-16 ***
BMI            0.36780    0.16426   2.239 0.025570 *
Age            0.37330    0.10303   3.623 0.000320 ***
OutcomeYes    27.61941    2.51184  10.996  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 25.29 on 522 degrees of freedom
Multiple R-squared:  0.373,	Adjusted R-squared:  0.3658
F-statistic: 51.75 on 6 and 522 DF,  p-value: < 2.2e-16

> |
```

**Validate and plot the model -**





```
= predict(lrMo
i_test$Glucose
rPred, col ="b
```

**References -**
[https://www.youtube.com/watch?v=1-URCcgTBf4](https://www.youtube.com/watch?v=1-URCcgTBf4)
[https://www.youtube.com/watch?v=hVMu3R5kwUo&t=1122s](https://www.youtube.com/watch?v=hVMu3R5kwUo&t=1122s)
[https://www.tutorialspoint.com/r/r_linear_regression.htm](https://www.tutorialspoint.com/r/r_linear_regression.htm)

## Unsupervised Learning

the machine learning task of inferring a function to describe hidden structure from "unlabeled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabeled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means. Tries to cluster the data based on the similarity and unsupervised means there is no outcome to be predicted finding out the similarities and differences in the data and group the similar items together and form clusters

**Clustering -**

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

**K means Algorithm -**

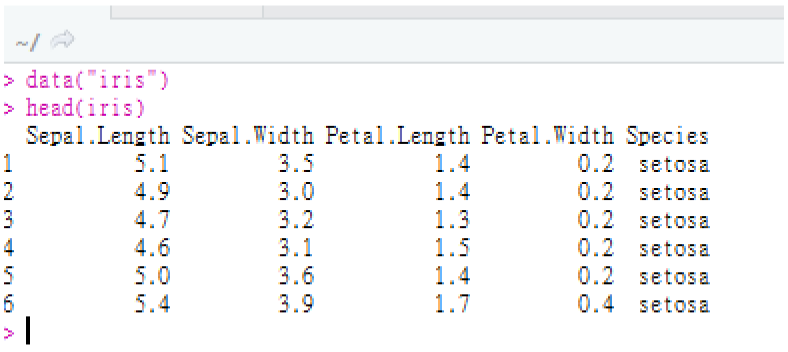
It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters).

**Use case | Problem statement**

We have the dataset called Iris dataset and in this we have the few species of flowers and the dependent variable is species which the unit belongs to.

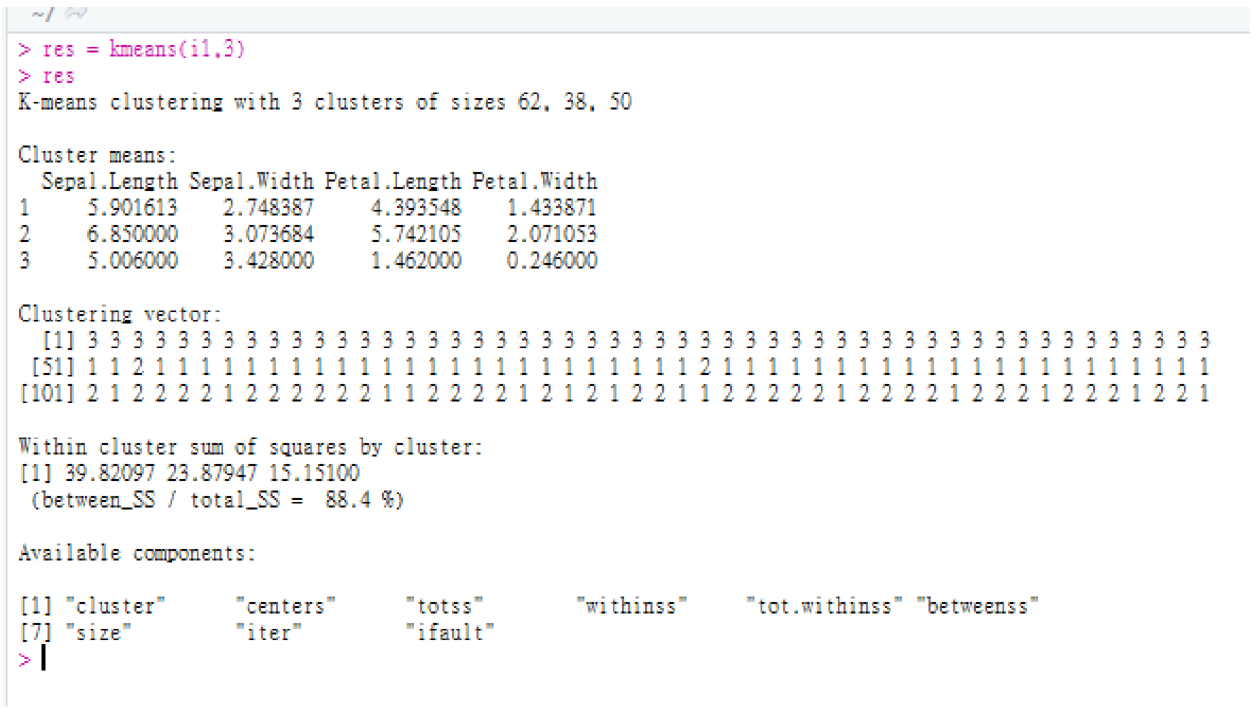
Classify each of the units into which of the species they fall into.

**Illustration**



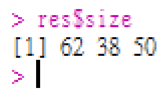
```
~/
> data("iris")
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> 
```

So using the Kmeans Algorithm, We will be clustering the species of the flower.



```
> res = kmeans(i1,3)
> res
K-means clustering with 3 clusters of sizes 62, 38, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.901613    2.748387     4.393548    1.433871
2     6.850000    3.073684     5.742105    2.071053
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 3 3 3 3 3 3 3 3 3 3 3 3 3
 [51] 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[101] 2 1 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1

Within cluster sum of squares by cluster:
[1] 39.82097 23.87947 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>
```

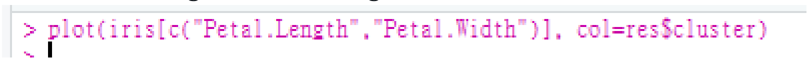
Lets check the size of each cluster



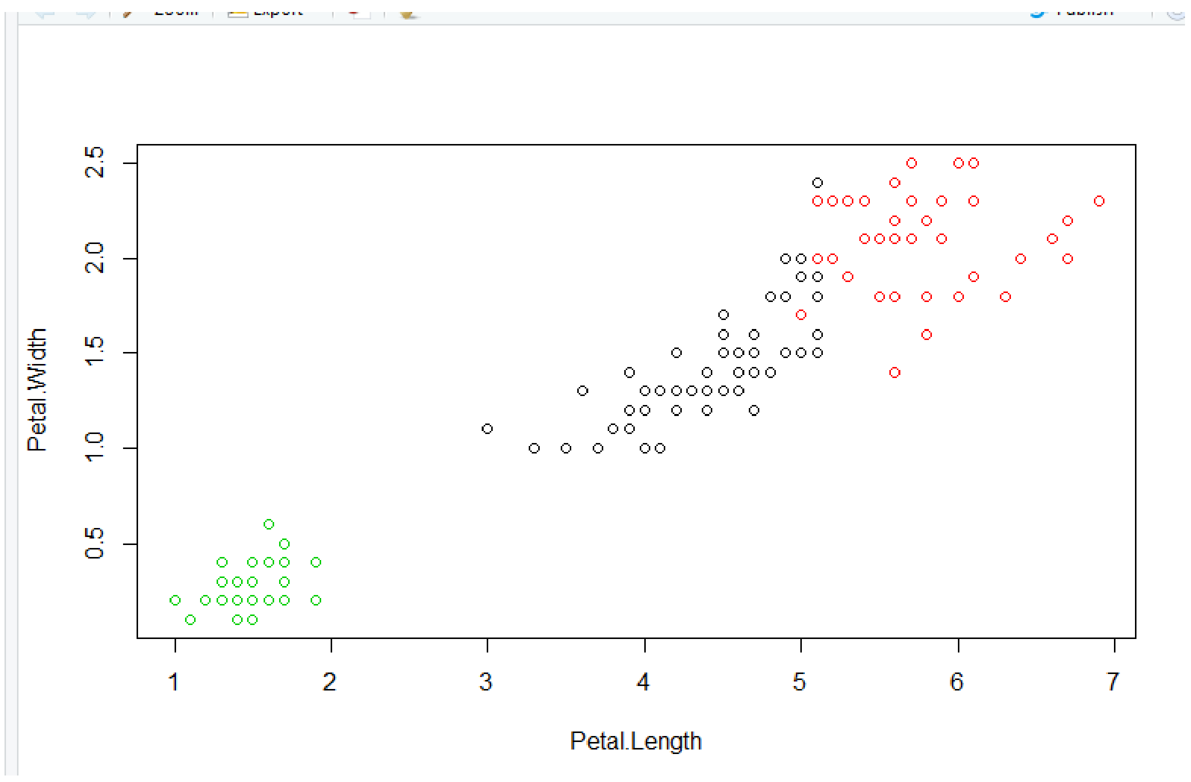
```
> res$size
[1] 62 38 50
>
```

**Check how good it has been fit**

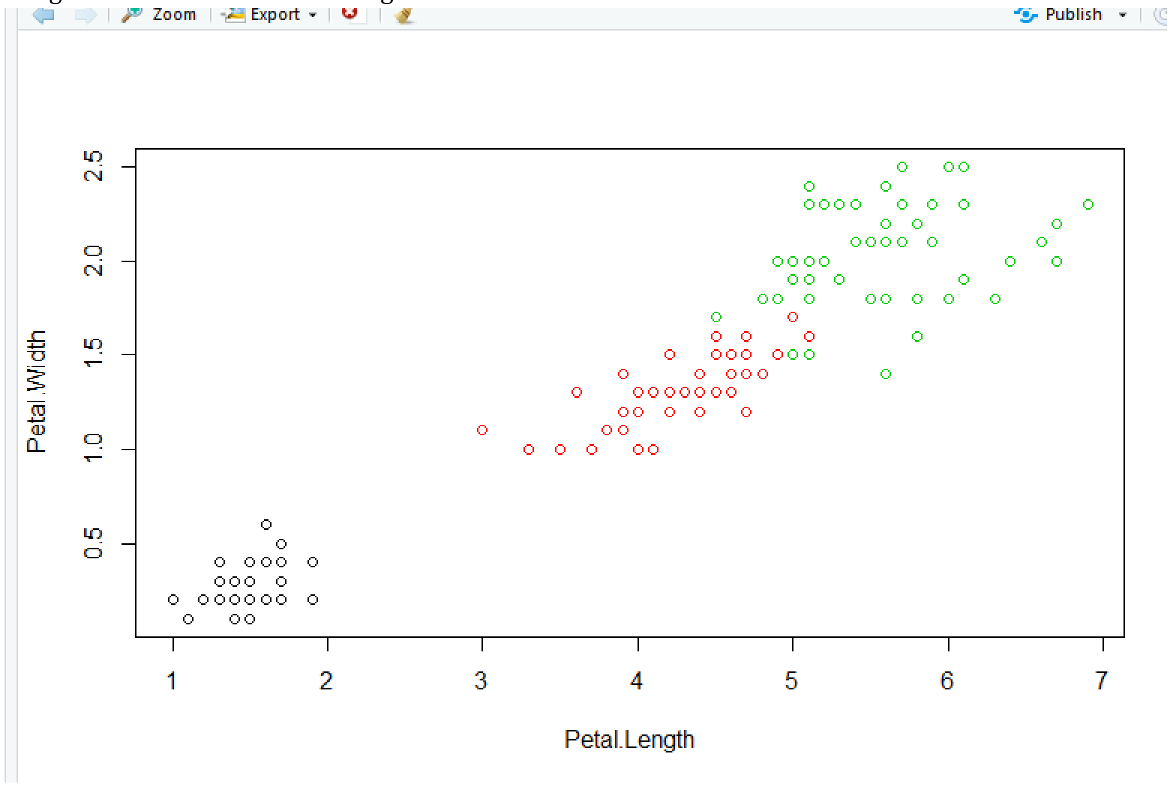
Plot the  result against Petal.Length and Petal.Width and color them according to their clusters



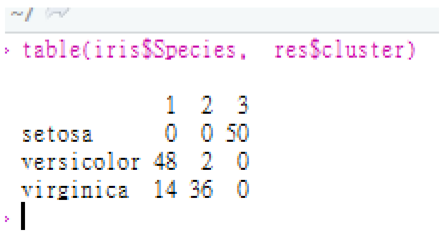
```
> plot(iris[c("Petal.Length","Petal.Width")], col=res$cluster)
```



Check how good has been fit with the original dataset



**Print the table against the Original Dataset**

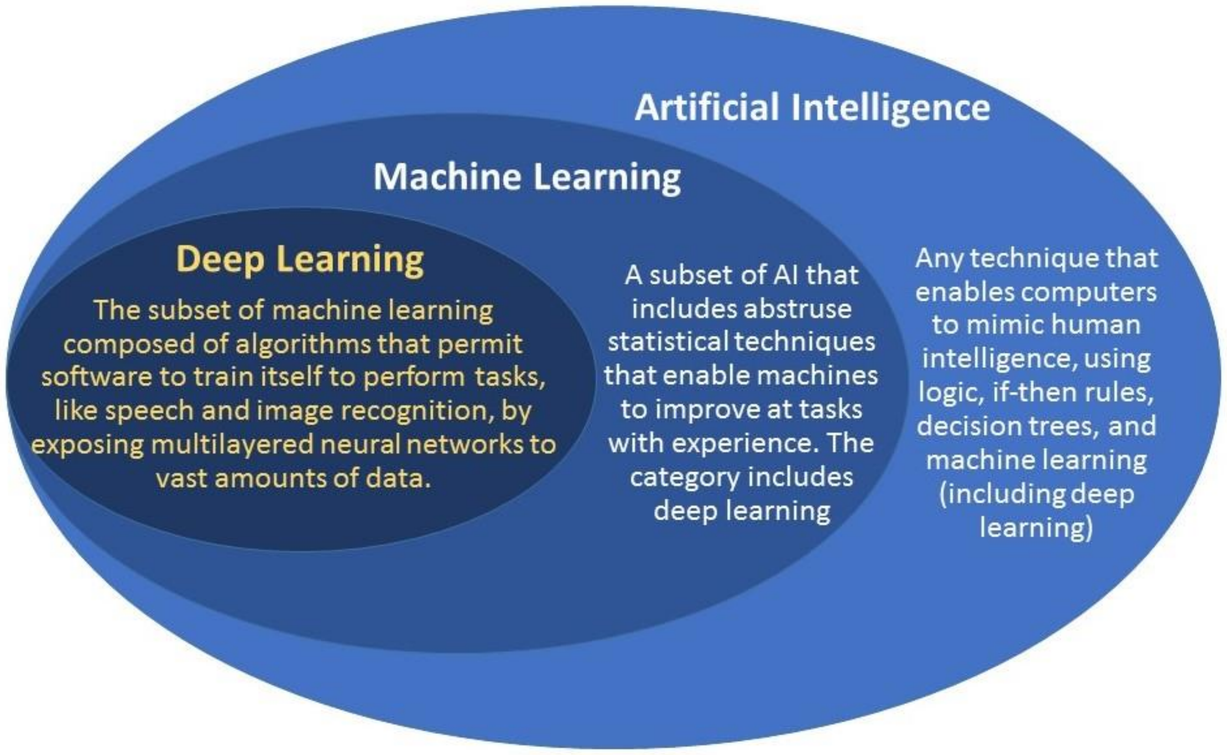


```
> table(iris$Species,  res$cluster)

             1  2  3
 setosa      0  0 50
 versicolor 48  2  0
 virginica  14 36  0
>
```

Seems that the performance has been pretty good  and the error isnt too high.

**Limitations of Machine Learning -**

- Not useful while working with high dimensional data ,that is  where we have a large number of inputs and outputs.
- Cannot solve the crucial AI problems like NLP, Image Recognition etc.
- One of the big challenges with the machine learning is feature extraction. So in statistic we consider features as variable but when we talk about AI these variable are nothing but  the features and because of that complex problems like object recognition or handwriting recognition becomes the huge challenge.


Artificial Intelligence
Machine Learning
Deep Learning
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.
A subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning
Any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning)

## Deep Learning | Neural Network

- **Why Neural Network**

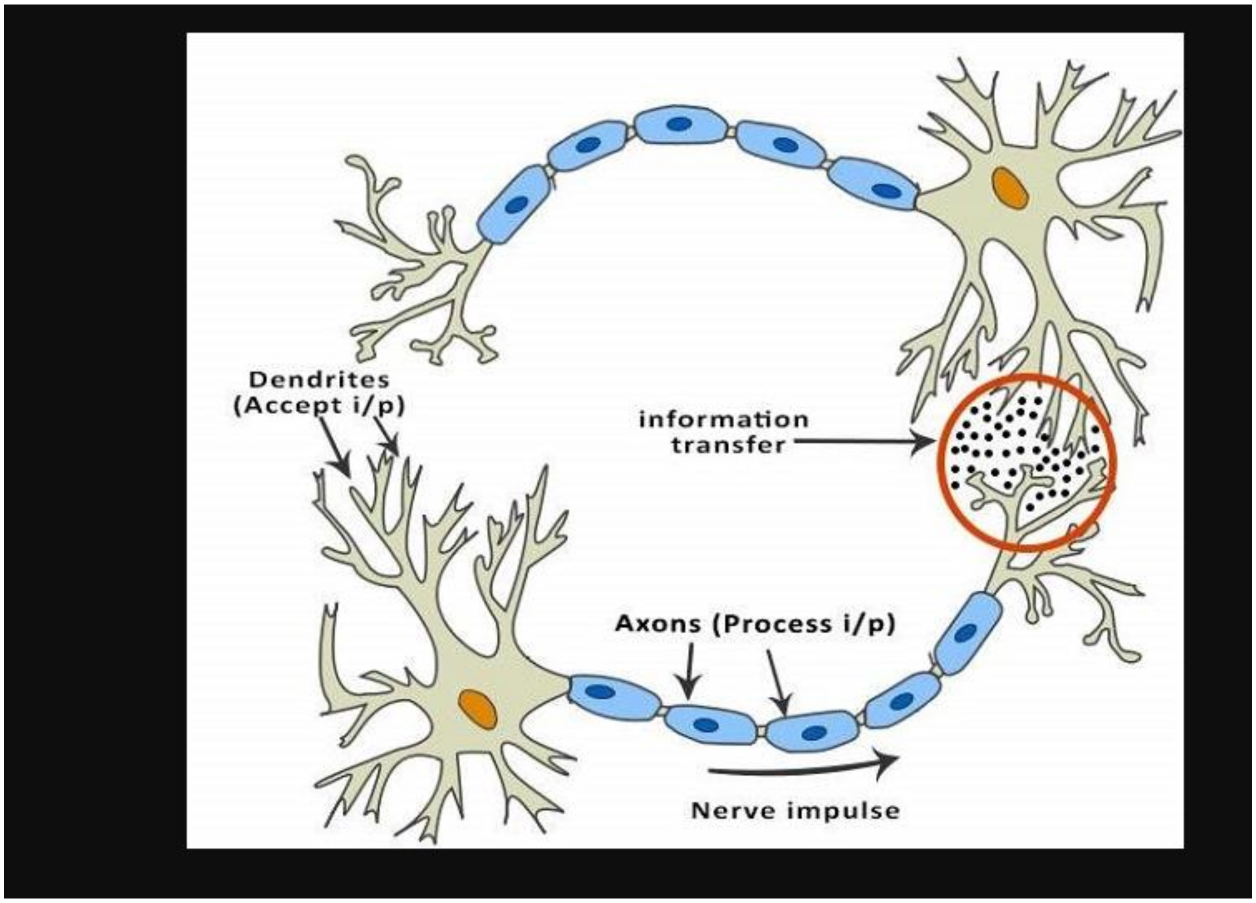
Earlier the conventional computers use an algorithmic approach that is computer follows set of instructions in order to solve a problem and unless the specific steps that the computer needs to follow are known the computer cannot solve the problem.
So this restricts the problem solving capability of conventional computers.

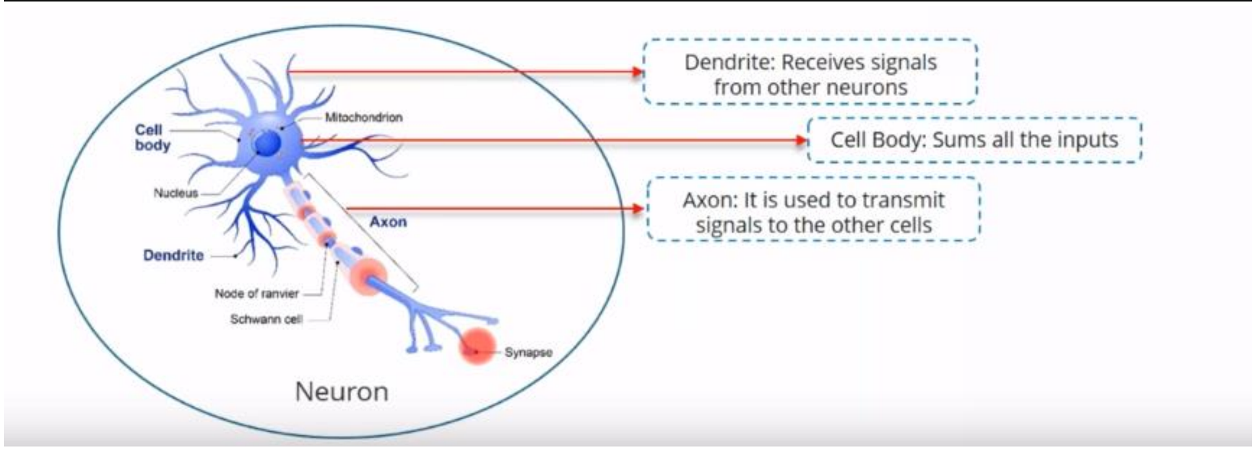
- **After Neural Network**

Basically process the information in as similar way the human brain does and these networks actually learn from examples.

- **Motivation Behind the neural Networks**

Basically inspired by the neurons which are nothing but our brain cells and exact working of human brains is still a mystery though.

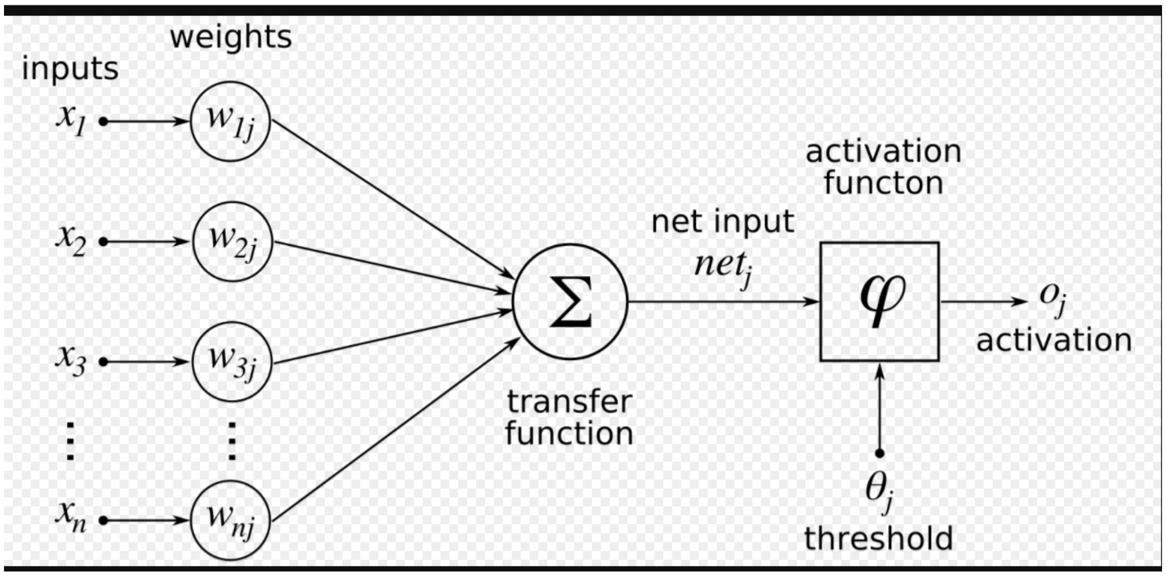



Dendrite: Receives signals
from other neurons
Cell Body: Sums all the inputs
Axon: It is used to transmit
signals to the other cells
Cell
body
Mitochondrion
Nucleus
Axon
Dendrite
Node of ranvier
Schwann cell
Synapse
Neuron

**What are Artificial Neural Networks ?**
Computing system that is designed to stimulate the way human brain analyses and process the information . AI has self learning capabilities that enable it to produce the better results as more data becomes available . With this new technologies have evolved like translating the web pages and to conversing with the chat box. In a nutshell, Artificial Neural Network is network of artificial neurons .

**How Artificial Neural Network Works ?**

Let’s start with single artificial neuron called as perceptron



This is an example of perceptron, Over here we have multiple x1, x2….xn and we have corresponding weight as well w1 for x1 , w2 for x2 and wn for xn then we calculated the weightage sum of these inputs after that we pass it through the activation function.
This activation function provides the threshold value so above that value my neuron will fire.

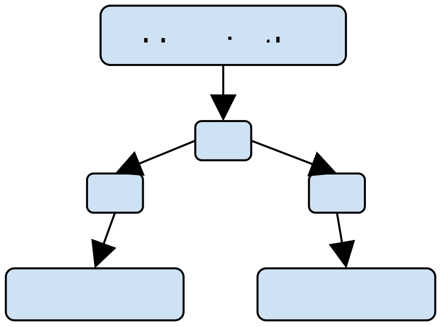
**Perceptron Training with Analogy -**

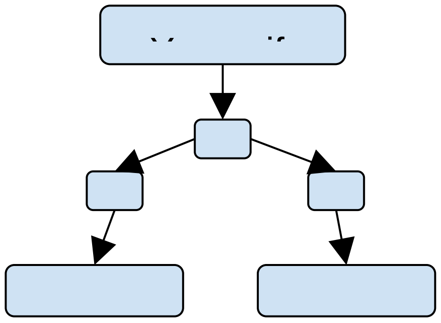
Suppose you want to go to a festival happening near your house. So your decision will depend on multiple factors.

- How is the weather ?
- Your wife is going with you ?
- Any public transport is available ?

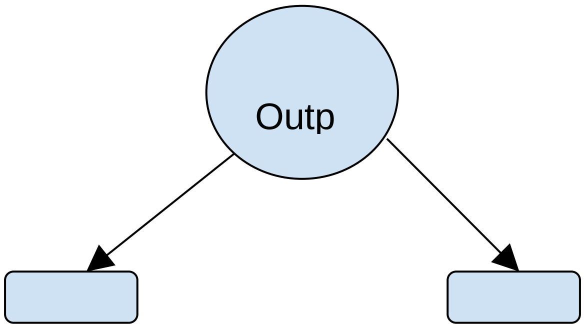
So will consider these three factors as inputs to our perceptron and will consider output for going or not going in the festival.

**Inputs -**





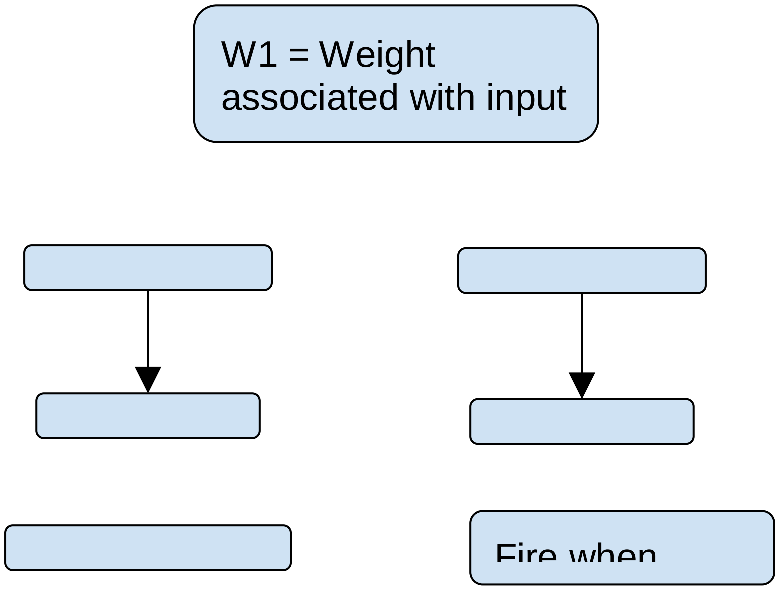
**Output -**



**Prioritize our factors -**

Here our most important factor is weather i.e if x1= 1 then the output o =1 , We will be assigning high weights to our high priority factor and low weights to less priority factors

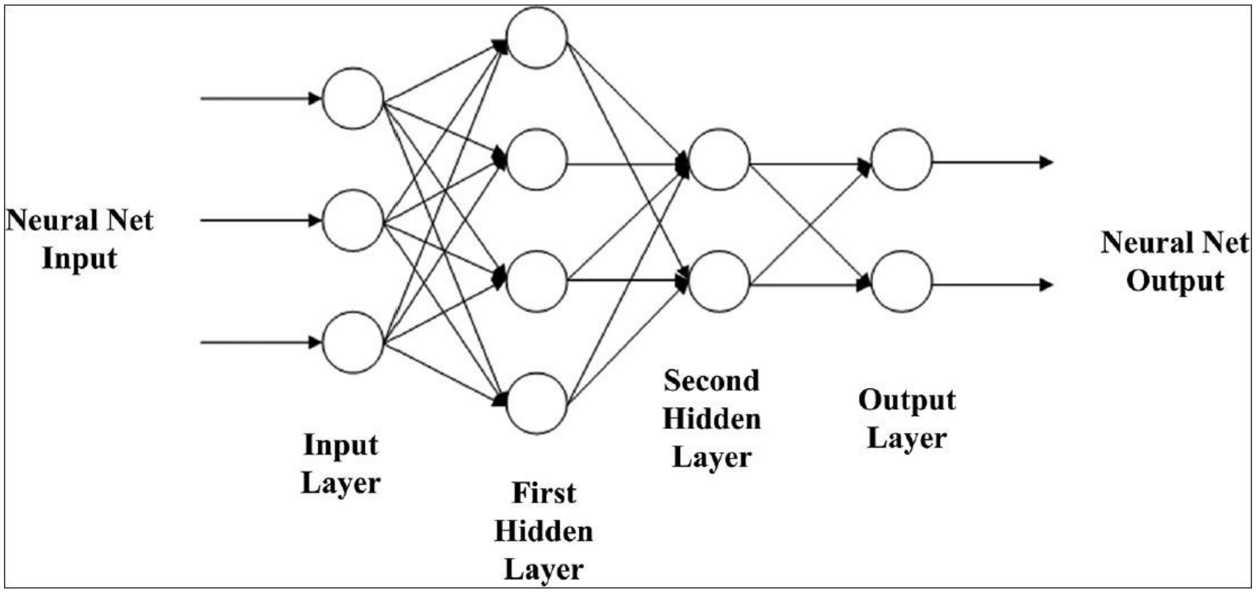
**Assign Weights -**



**x1*w1+x2*w3+x3*w3**
**1*6+0*2+0*3 = 6**

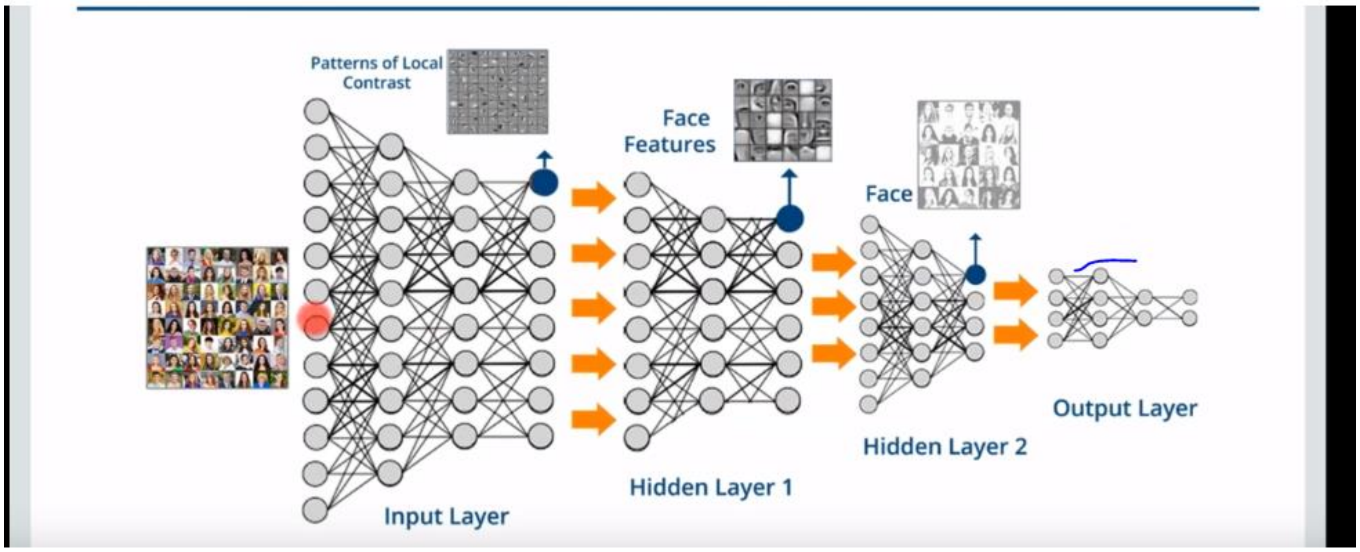
**1*6 +0*2+0*3 = 6**

**Multilayer Perceptron - Artificial Neural Network -**



First layer is always a input layer this is where we actually feeding all our inputs then we have the first hidden layer and second hidden layer  after that we have the output .
Although your hidden layers are actually depends upon what application we are working on or what is our problem so that actually determines how many hidden layers will have.

**Image Recognition - Neural Network**

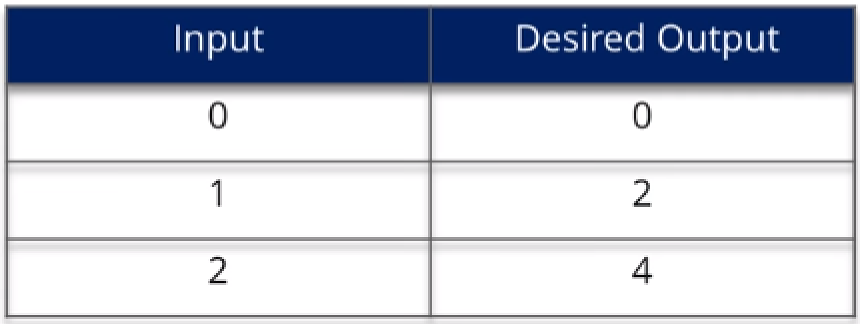


Over here we are feeding lot of images to the input layer , so this layer will actually detect the patterns of local contrast here and then will feed that into  our next layer that is hidden layer in this face features will be recognized  and then i will be again fed into the layer which second hidden layer this layer will actually assemble the features and  try to make the face and then will get the output.

**Training a neural network -**

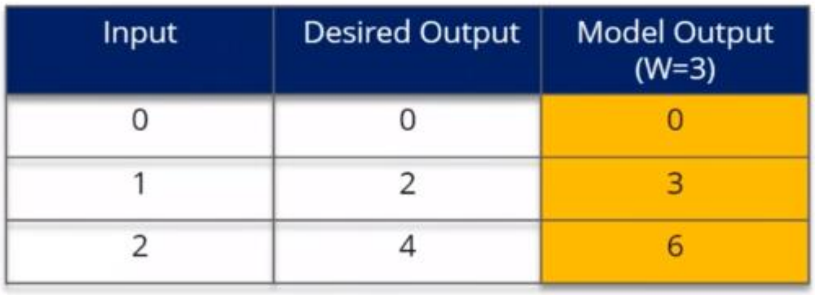
So basically the most common algorithm for training a network is back propagation
After the weightage sum of all inputs and passing it through the activation function and getting the output, we will compare the output with the actual output that we already know and we figure out what is difference, calculate the error and based on that error we propagate backwards .
We will check what happens when we change the weight , will the error increase or decrease.

**Example -**



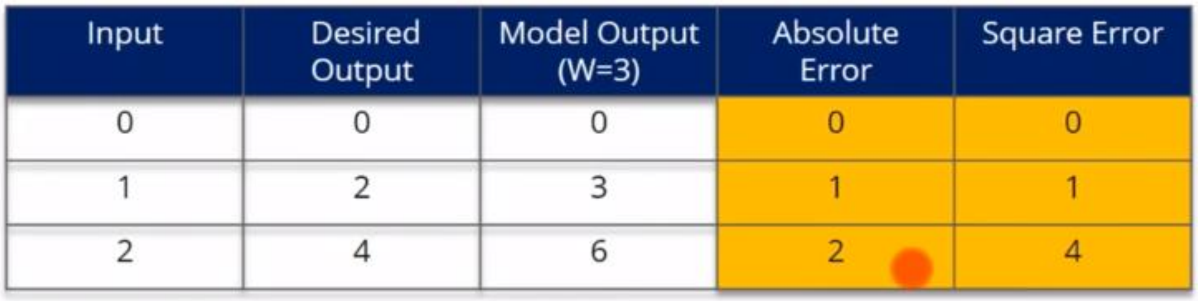
| Input | Desired Output |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |

Consider the initial value of weight as 3 W = 3,



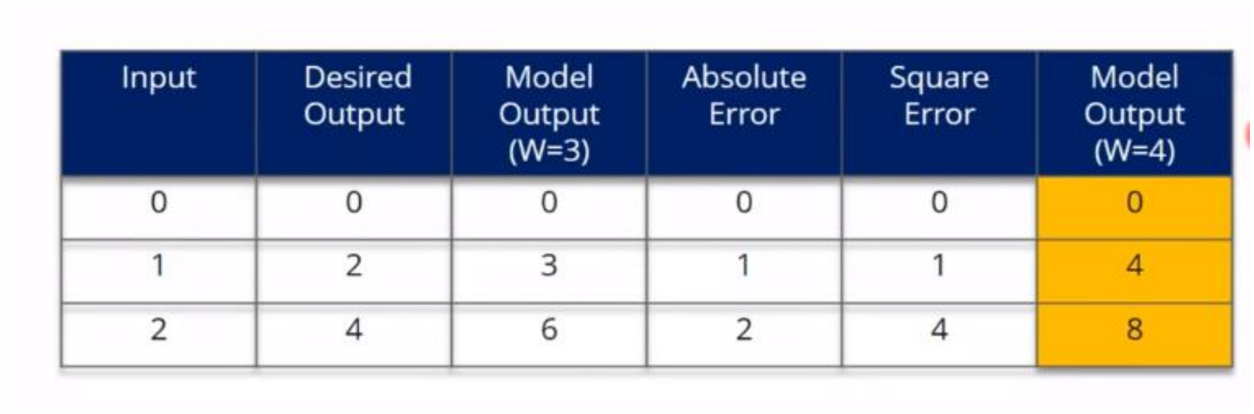
| Input | Desired Output | Model Output (W=3) |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 2 | 3 |
| 2 | 4 | 6 |

Obviously the output is not equals to your desired output so we check the error (Absolute and Square).



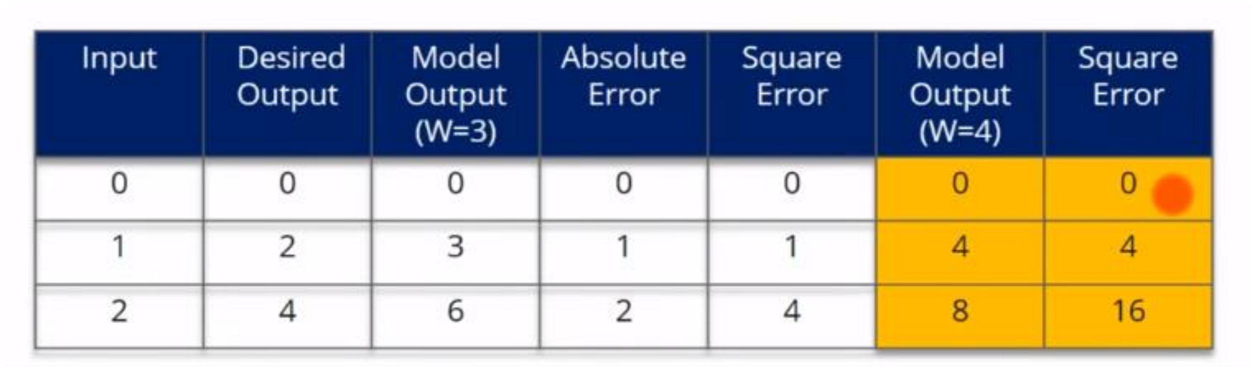
| Input | Desired Output | Model Output (W=3) | Absolute Error | Square Error |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 1 | 1 |
| 2 | 4 | 6 | 2 | 4 |

Now we need to update the value of the weight  W = 4,



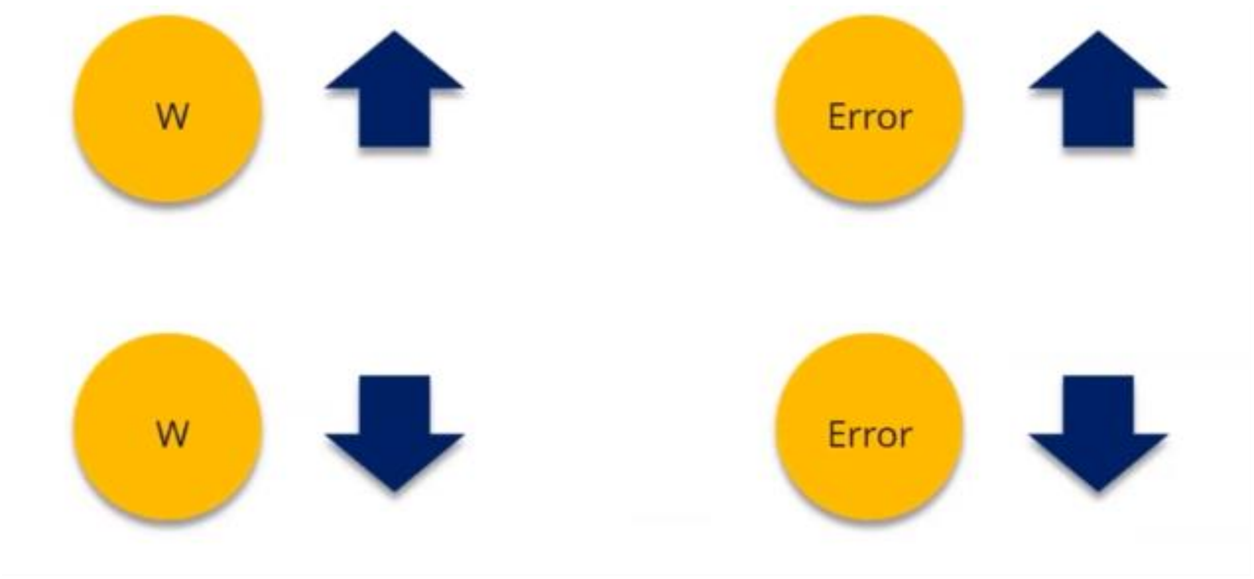
| Input | Desired Output | Model Output (W=3) | Absolute Error | Square Error | Model Output (W=4) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 1 | 1 | 4 |
| 2 | 4 | 6 | 2 | 4 | 8 |

And then w'll check the error and found that error has actually increased instead of decreasing , So after increasing the variable error is increasing.



| Input | Desired Output | Model Output (W=3) | Absolute Error | Square Error | Model Output (W=4) | Square Error |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 1 | 1 | 4 | 4 |
| 2 | 4 | 6 | 2 | 4 | 8 | 16 |

I.e we can not increase our variable value but if we decrease the variable value , W= 2, will get the desired output.



**How we determine whether we have to increase or decrease the weight.**

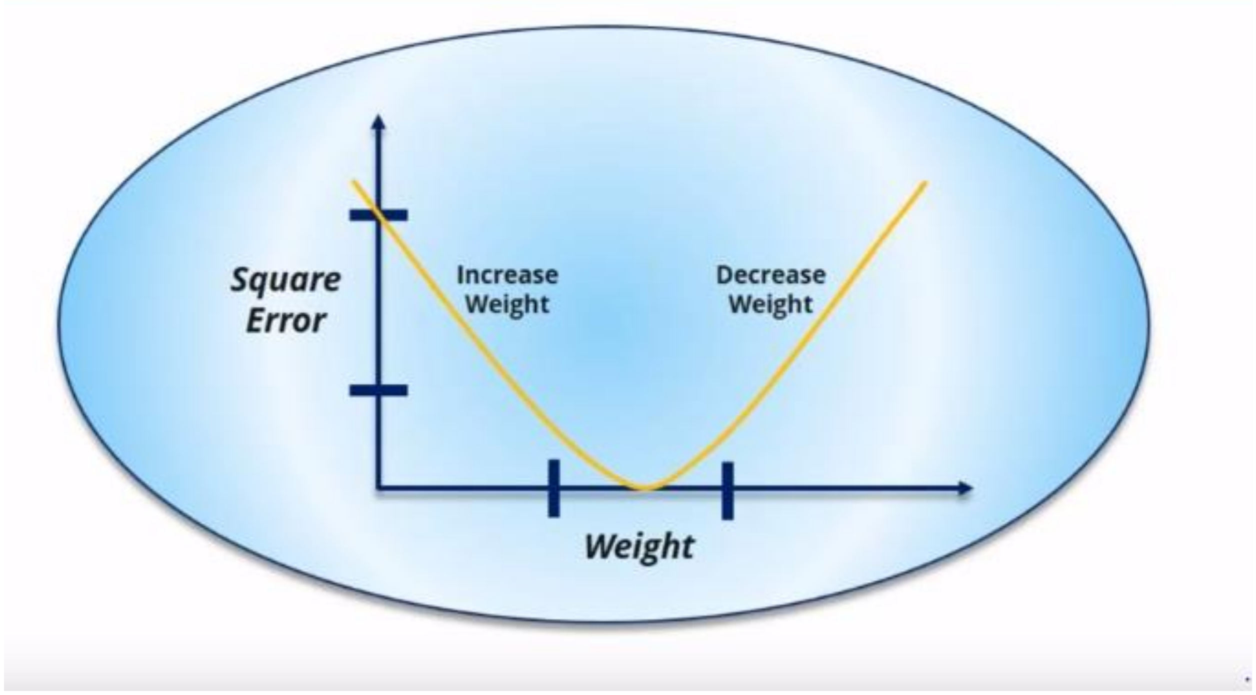
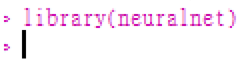


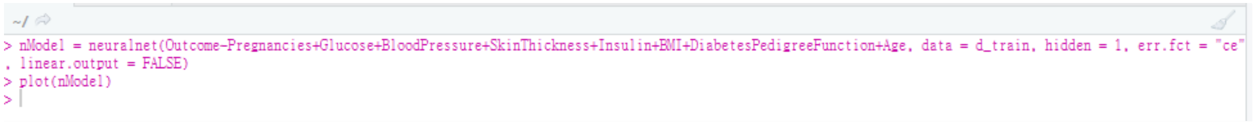
**Illustration -**

Install the library neuralnet



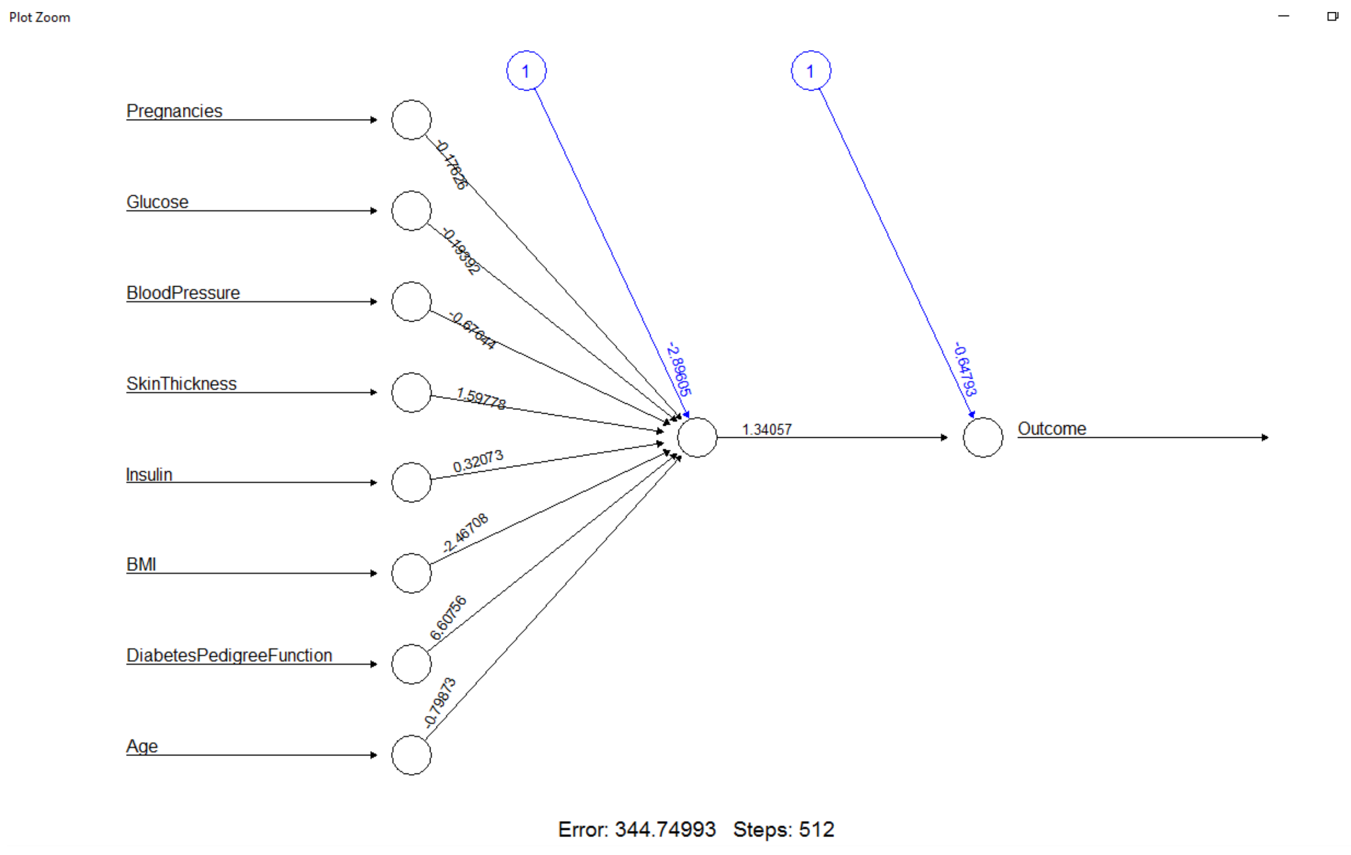
```
> library(neuralnet)
> 
```

**Fit the model**



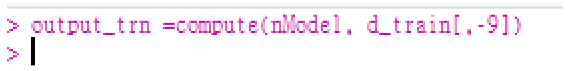
```
> nModel = neuralnet(Outcome-Pregnancies+Glucose+BloodPressure+SkinThickness+Insulin+BMI+DiabetesPedigreeFunction+Age, data = d_train, hidden = 1, err.fct = "ce"
, linear.output = FALSE)
> plot(nModel)
> 
```

Err.fct = Differential function used for calculation of the error.
Linear. Output = logical function, if act.fct is not applied to the output neurons  then set linear.output to TRUE otherwise FALSE
Act.fct = Differential function used for smoothing the result of the cross product of the neurons and weights .



As per the graph we can see the first layer is the input layer and the second layer is the hidden layer and the last layer is output layer.
With in these layers some weights are estimated and the blue connected to the node is similar to the constant

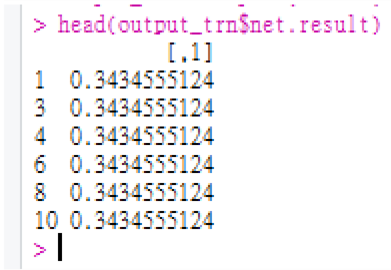
## Validating the model



```
> output_trn =compute(nModel, d_train[,-9])
> 
```

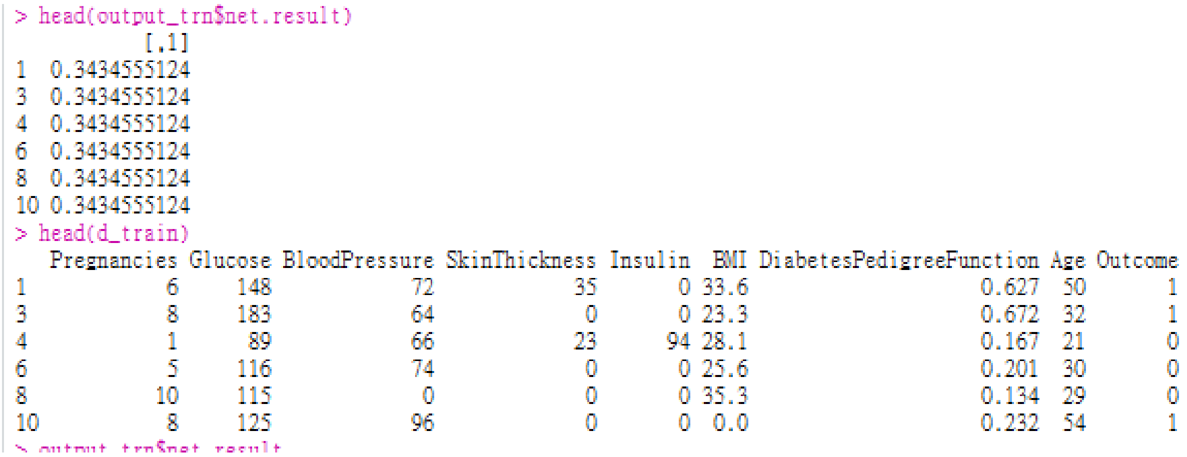
For prediction we are using the compute function here and make sure that while using the this function you have to exclude the ninth  colom when you are using the neural network .

Lets see the first few rows in the output and look at net.result



```
> head(output_trn$net.result)
           [,1]
1  0.3434555124
3  0.3434555124
4  0.3434555124
6  0.3434555124
8  0.3434555124
10 0.3434555124
> 
```

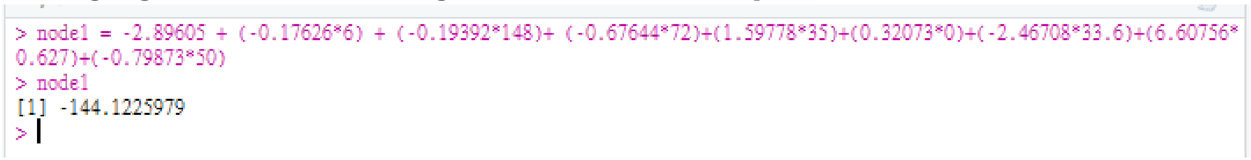
Compare it with the training data that we have .



```
> head(output_trn$net.result)
           [,1]
1  0.3434555124
3  0.3434555124
4  0.3434555124
6  0.3434555124
8  0.3434555124
10 0.3434555124
> head(d_train)
   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI DiabetesPedigreeFunction Age Outcome
1            6     148            72            35       0 33.6                    0.627  50       1
3            8     183            64             0       0 23.3                    0.672  32       1
4            1      89            66            23      94 28.1                    0.167  21       0
6            5     116            74             0       0 25.6                    0.201  30       0
8           10     115             0             0       0 35.3                    0.134  29       0
10           8     125            96             0       0  0.0                    0.232  54       1
```

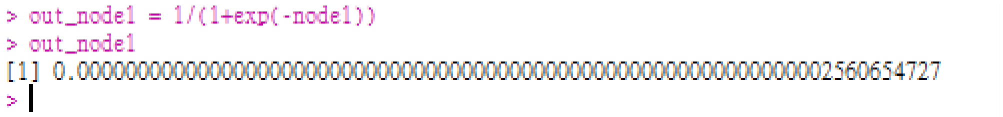
**Node output calculations with the Sigmoid Activation Function.**

We are going to make use of these weights associated with each inputs .



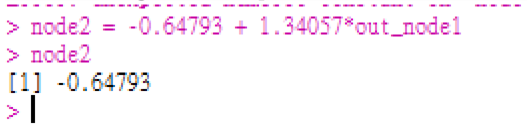
```
> node1 = -2.89605 + (-0.17626*6) + (-0.19392*148)+ (-0.67644*72)+(1.59778*35)+(0.32073*0)+(-2.46708*33.6)+(6.60756*
0.627)+(-0.79873*50)
> node1
[1] -144.1225979
>
```

So the output of this node will be based on the sigmoid function



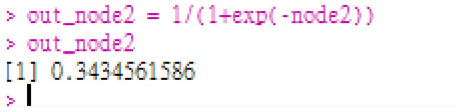
```
> out_node1 = 1/(1+exp(-node1))
> out_node1
[1] 0.0000000000000000000000000000000000000000000000000000000000000002560654727
>
```

So here we got the output of node1 and will be calculate the output of the next node .



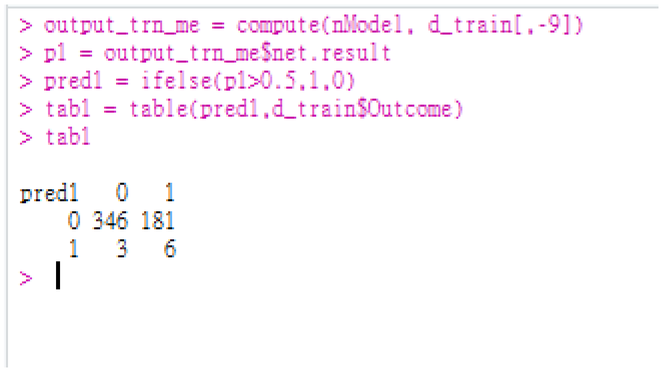
```
> node2 = -0.64793 + 1.34057*out_node1
> node2
[1] -0.64793
>
```

Apply the sigmoid function once again



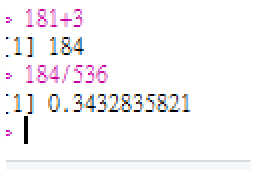
```
> out_node2 = 1/(1+exp(-node2))
> out_node2
[1] 0.3434561586
>
```

**Confusion Matrix and Misclassification Error - Training Data**



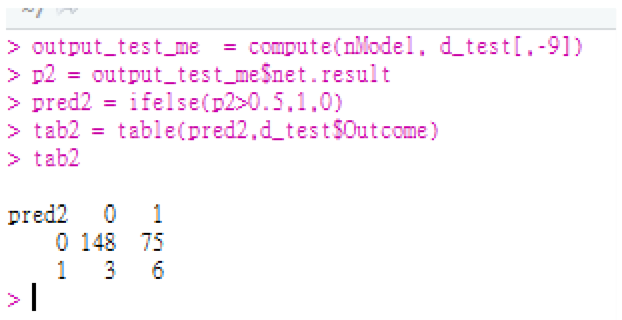
```
> output_trn_me = compute(nModel, d_train[,-9])
> p1 = output_trn_me$net.result
> pred1 = ifelse(p1>0.5,1,0)
> tab1 = table(pred1,d_train$Outcome)
> tab1

pred1   0   1
    0 346 181
    1   3   6
> 
```

**Misclassification error -**



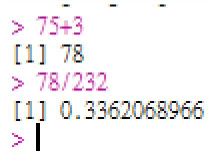
```
> 181+3
[1] 184
> 184/536
[1] 0.3432835821
> 
```

**Confusion Matrix and Misclassification Error - Testing Data**



```
> output_test_me  = compute(nModel, d_test[,-9])
> p2 = output_test_me$net.result
> pred2 = ifelse(p2>0.5,1,0)
> tab2 = table(pred2,d_test$Outcome)
> tab2

pred2   0   1
    0 148  75
    1   3   6
> 
```

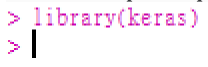
**Misclassification Error -**



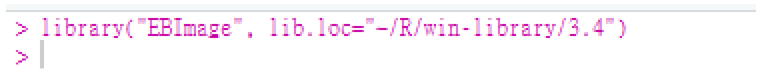
```
> 75+3
[1] 78
> 78/232
[1] 0.3362068966
> 
```

**Image recognition -**

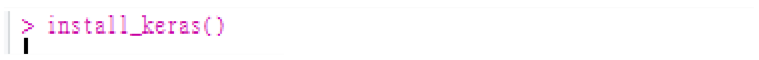
Install the prerequisite libraries



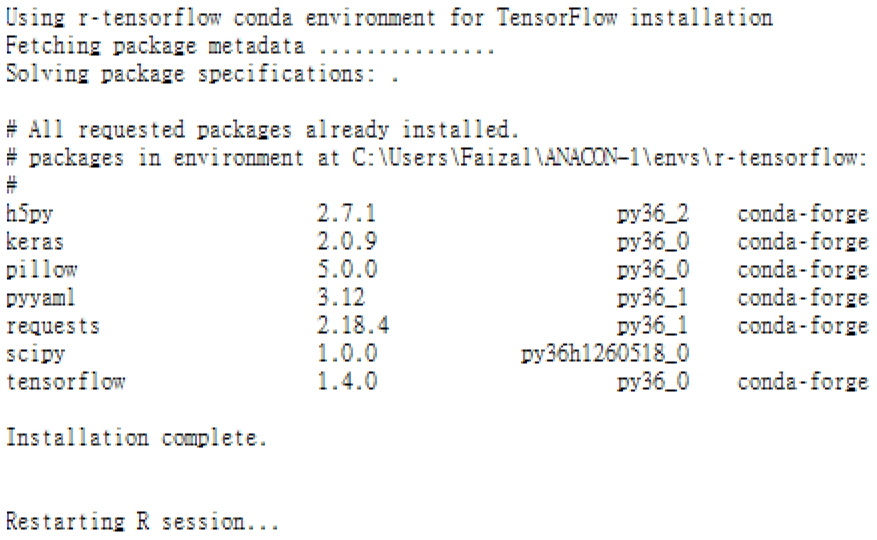
```
> library(keras)
> 
```



```
> library("EBImage", lib.loc="~/R/win-library/3.4")
> 
```

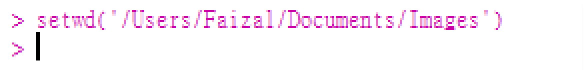


```
> install_keras()
```



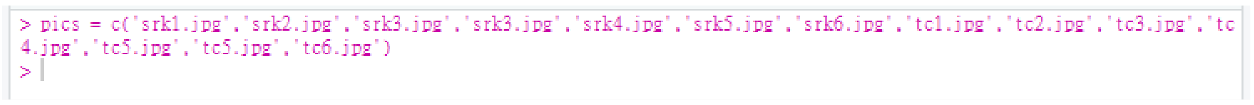
```
Using r-tensorflow conda environment for TensorFlow installation
Fetching package metadata ...............
Solving package specifications: .

# All requested packages already installed.
# packages in environment at C:\Users\Faizal\ANACON~1\envs\r-tensorflow:
#
h5py                      2.7.1                    py36_2    conda-forge
keras                     2.0.9                    py36_0    conda-forge
pillow                    5.0.0                    py36_0    conda-forge
pyyaml                    3.12                     py36_1    conda-forge
requests                  2.18.4                   py36_1    conda-forge
scipy                     1.0.0            py36h1260518_0
tensorflow                1.4.0                    py36_0    conda-forge

Installation complete.


Restarting R session...
```

**Read the Images -**
setwd('/Users/Faizal/Documents/Images')



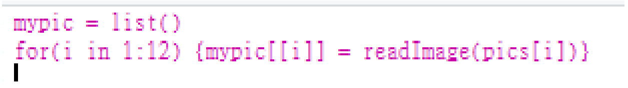
```
> setwd('/Users/Faizal/Documents/Images')
> 
```

> pics =
c('srk1.jpg','srk2.jpg','srk3.jpg','srk3.jpg','srk4.jpg','srk5.jpg','srk6.jpg','tc1.jpg','tc2.jpg','tc3.jpg','tc4.jpg','tc5.jp
g','tc5.jpg','tc6.jpg')



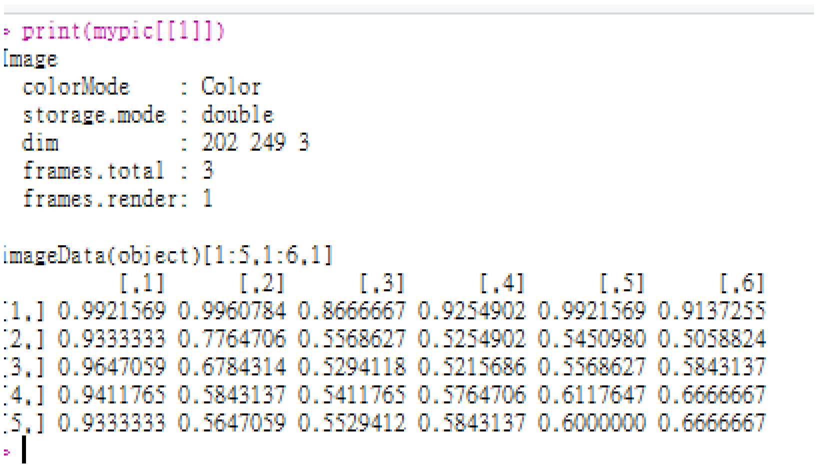
```
> pics = c('srk1.jpg','srk2.jpg','srk3.jpg','srk3.jpg','srk4.jpg','srk5.jpg','srk6.jpg','tc1.jpg','tc2.jpg','tc3.jpg','tc
4.jpg','tc5.jpg','tc5.jpg','tc6.jpg')
> 
```

> mypic = list()
> for(i in 1:12) {mypic[[i]] = readImage(pics[i])}



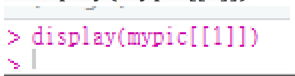
```
mypic = list()
for(i in 1:12) {mypic[[i]] = readImage(pics[i])}
```

**Explore -**
> print(mypic[[1]])



```
> print(mypic[[1]])
Image
  colorMode    : Color
  storage.mode : double
  dim          : 202 249 3
  frames.total : 3
  frames.render: 1

imageData(object)[1:5,1:6,1]
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.9921569 0.9960784 0.8666667 0.9254902 0.9921569 0.9137255
[2,] 0.9333333 0.7764706 0.5568627 0.5254902 0.5450980 0.5058824
[3,] 0.9647059 0.6784314 0.5294118 0.5215686 0.5568627 0.5843137
[4,] 0.9411765 0.5843137 0.5411765 0.5764706 0.6117647 0.6666667
[5,] 0.9333333 0.5647059 0.5529412 0.5843137 0.6000000 0.6666667
> 
```

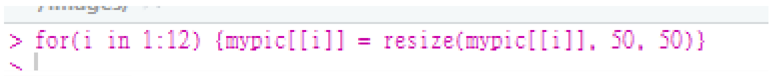
> display(mypic[[1]])



```
> display(mypic[[1]])
> 
```

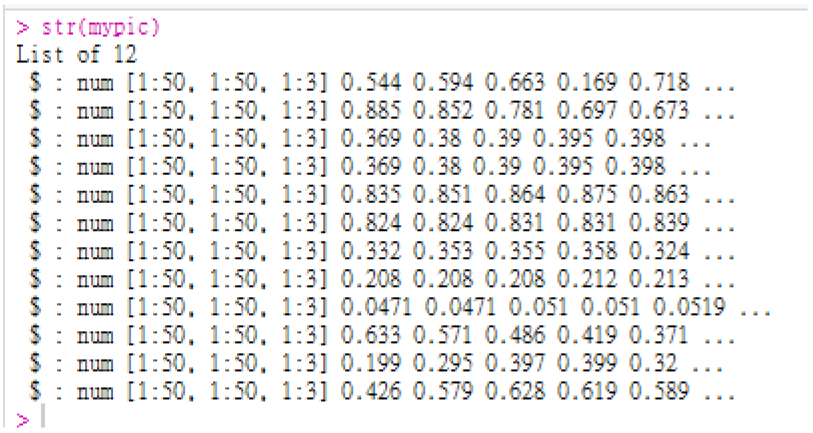
**Resize the images -**

> for(i in 1:12) {mypic[[i]] = resize(mypic[[i]], 50, 50)}



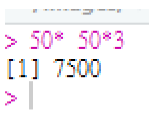
```
> for(i in 1:12) {mypic[[i]] = resize(mypic[[i]], 50, 50)}
> 
```

str(mypic)



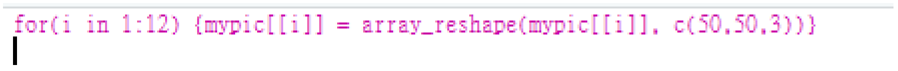
```
> str(mypic)
List of 12
 $ : num [1:50, 1:50, 1:3] 0.544 0.594 0.663 0.169 0.718 ...
 $ : num [1:50, 1:50, 1:3] 0.885 0.852 0.781 0.697 0.673 ...
 $ : num [1:50, 1:50, 1:3] 0.369 0.38 0.39 0.395 0.398 ...
 $ : num [1:50, 1:50, 1:3] 0.369 0.38 0.39 0.395 0.398 ...
 $ : num [1:50, 1:50, 1:3] 0.835 0.851 0.864 0.875 0.863 ...
 $ : num [1:50, 1:50, 1:3] 0.824 0.824 0.831 0.831 0.839 ...
 $ : num [1:50, 1:50, 1:3] 0.332 0.353 0.355 0.358 0.324 ...
 $ : num [1:50, 1:50, 1:3] 0.208 0.208 0.208 0.212 0.213 ...
 $ : num [1:50, 1:50, 1:3] 0.0471 0.0471 0.051 0.051 0.0519 ...
 $ : num [1:50, 1:50, 1:3] 0.633 0.571 0.486 0.419 0.371 ...
 $ : num [1:50, 1:50, 1:3] 0.199 0.295 0.397 0.399 0.32 ...
 $ : num [1:50, 1:50, 1:3] 0.426 0.579 0.628 0.619 0.589 ...
>
```

**Reshape -**
**> 50* 50*3**
**[1] 7500**



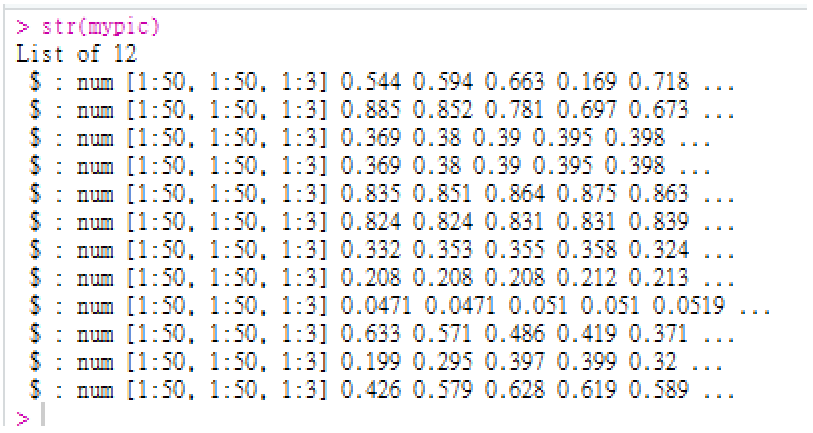
```
> 50* 50*3
[1] 7500
>
```

> for(i in 1:12) {mypic[[i]] = array_reshape(mypic[[i]], c(50,50,3))}



```
for(i in 1:12) {mypic[[i]] = array_reshape(mypic[[i]], c(50,50,3))}
```

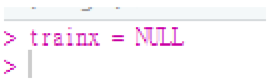
> str(mypic)



```
> str(mypic)
List of 12
 $ : num [1:50, 1:50, 1:3] 0.544 0.594 0.663 0.169 0.718 ...
 $ : num [1:50, 1:50, 1:3] 0.885 0.852 0.781 0.697 0.673 ...
 $ : num [1:50, 1:50, 1:3] 0.369 0.38 0.39 0.395 0.398 ...
 $ : num [1:50, 1:50, 1:3] 0.369 0.38 0.39 0.395 0.398 ...
 $ : num [1:50, 1:50, 1:3] 0.835 0.851 0.864 0.875 0.863 ...
 $ : num [1:50, 1:50, 1:3] 0.824 0.824 0.831 0.831 0.839 ...
 $ : num [1:50, 1:50, 1:3] 0.332 0.353 0.355 0.358 0.324 ...
 $ : num [1:50, 1:50, 1:3] 0.208 0.208 0.208 0.212 0.213 ...
 $ : num [1:50, 1:50, 1:3] 0.0471 0.0471 0.051 0.051 0.0519 ...
 $ : num [1:50, 1:50, 1:3] 0.633 0.571 0.486 0.419 0.371 ...
 $ : num [1:50, 1:50, 1:3] 0.199 0.295 0.397 0.399 0.32 ...
 $ : num [1:50, 1:50, 1:3] 0.426 0.579 0.628 0.619 0.589 ...
>
```

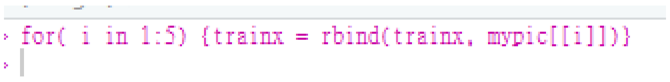
**Combine this data into one using row bind -**

> trainx = NULL
Data that will used for training trainx , x for independent variable

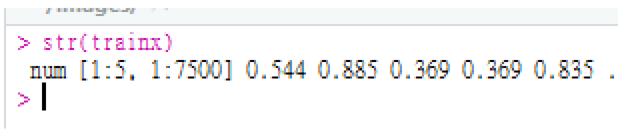


```
> trainx = NULL
>
```

For training will use first five images of SRK and sixth one for testing and similarly will use the first five images of TC and sixth one for testing.
> for( i in 1:5) {trainx = rbind(trainx, mypic[[i]])}

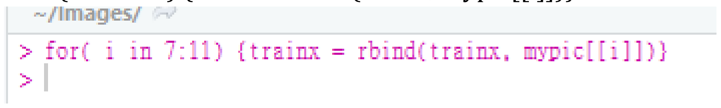


```
> for( i in 1:5) {trainx = rbind(trainx, mypic[[i]])}
>
```

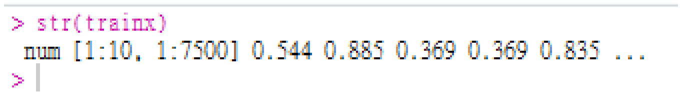


```
> str(trainx)
 num [1:5, 1:7500] 0.544 0.885 0.369 0.369 0.835 .
>
```

> for( i in 7:11) {trainx = rbind(trainx, mypic[[i]])}

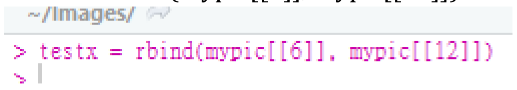


```
~/Images/
> for( i in 7:11) {trainx = rbind(trainx, mypic[[i]])}
>
```



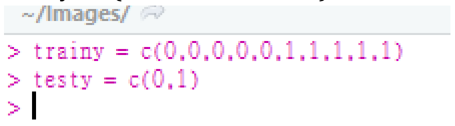
```
> str(trainx)
 num [1:10, 1:7500] 0.544 0.885 0.369 0.369 0.835 ...
>
```

For testing for SRK and TC
> testx = rbind(mypic[[6]], mypic[[12]])



```
~/Images/
> testx = rbind(mypic[[6]], mypic[[12]])
>
```

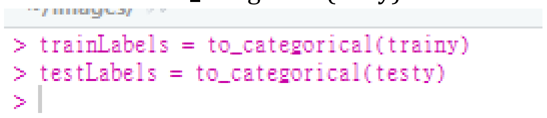
So here we are representing SRK by 0 and TC by 1
trainy = c(0,0,0,0,0,1,1,1,1,1)



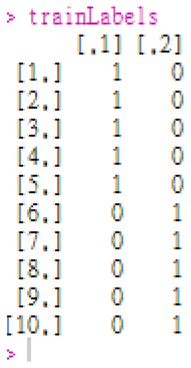
```
~/Images/
> trainy = c(0,0,0,0,0,1,1,1,1,1)
> testy = c(0,1)
>
```

**One Hot Encoding -**
One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

> trainLabels = to_categorical(trainy)
> testLabels = to_categorical(testy)



```
> trainLabels = to_categorical(trainy)
> testLabels = to_categorical(testy)
>
```

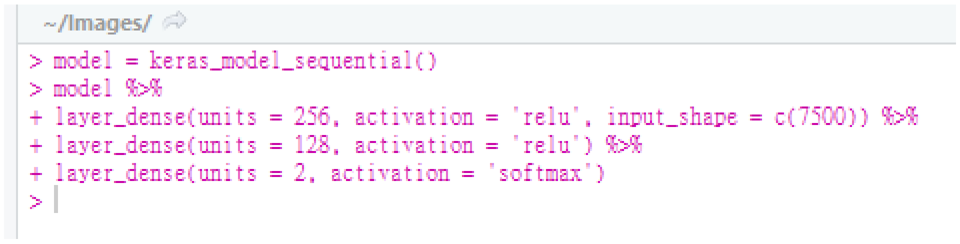


```
> trainLabels
      [,1] [,2]
 [1,]    1    0
 [2,]    1    0
 [3,]    1    0
 [4,]    1    0
 [5,]    1    0
 [6,]    0    1
 [7,]    0    1
 [8,]    0    1
 [9,]    0    1
[10,]    0    1
>
```

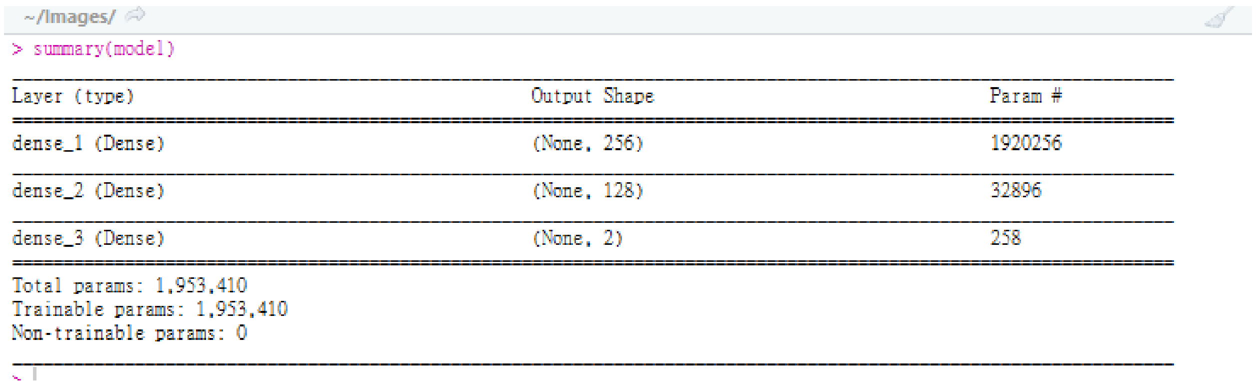
Created two dummy variables, whenever it indicates 1 that means thats the first class means thats a SRK and for the second coloum when we have 1 thats second category thats a TC.

**Create the Model -**
Layer_dense = fully connected neural network
Units = number of neurons in the hidden layer
Activation = defines the output of that node given an input or set of inputs
Input_shape = how many neurons are there in input layer
Softmax = output of the softmax function can be used to represent a categorical distribution
Relu = rectified linear unit (ReLU)

```
> model = keras_model_sequential()
> model %>%
+ layer_dense(units = 256, activation = 'relu', input_shape = c(7500)) %>%
+ layer_dense(units = 128, activation = 'relu') %>%
+ layer_dense(units = 2, activation = 'softmax')
```

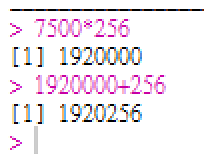


```
> model = keras_model_sequential()
> model %>%
+ layer_dense(units = 256, activation = 'relu', input_shape = c(7500)) %>%
+ layer_dense(units = 128, activation = 'relu') %>%
+ layer_dense(units = 2, activation = 'softmax')
>
```



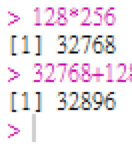
```
~/Images/
> summary(model)
________________________________________________________________________________________
Layer (type)                        Output Shape                      Param #
========================================================================================
dense_1 (Dense)                     (None, 256)                       1920256
________________________________________________________________________________________
dense_2 (Dense)                     (None, 128)                       32896
________________________________________________________________________________________
dense_3 (Dense)                     (None, 2)                         258
========================================================================================
Total params: 1,953,410
Trainable params: 1,953,410
Non-trainable params: 0
________________________________________________________________________________________
>
```

So total number of parameters based on first hidden layer is



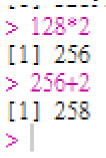
```
> 7500*256
[1] 1920000
> 1920000+256
[1] 1920256
>
```

Similarly for second hidden layer, total number of parameters based on first hidden layer is



```
> 128*256
[1] 32768
> 32768+128
[1] 32896
>
```

Similarly for third hidden layer, total number of parameters based on first hidden layer is

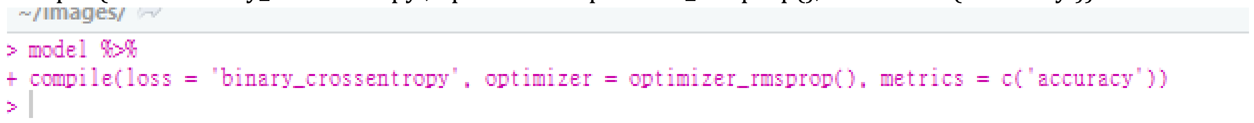


```
> 128*2
[1] 256
> 256+2
[1] 258
>
```

**Compile the model -**

Loss = Binary_crossentropy (because response variable has only two values )

Model %>%
compile(loss = 'binary_crossentropy', optimizer = optimizer_rmsprop(), metrics = c('accuracy'))

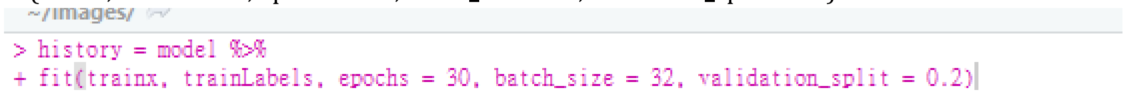


```
~/Images/
> model %>%
+ compile(loss = 'binary_crossentropy', optimizer = optimizer_rmsprop(), metrics = c('accuracy'))
>
```

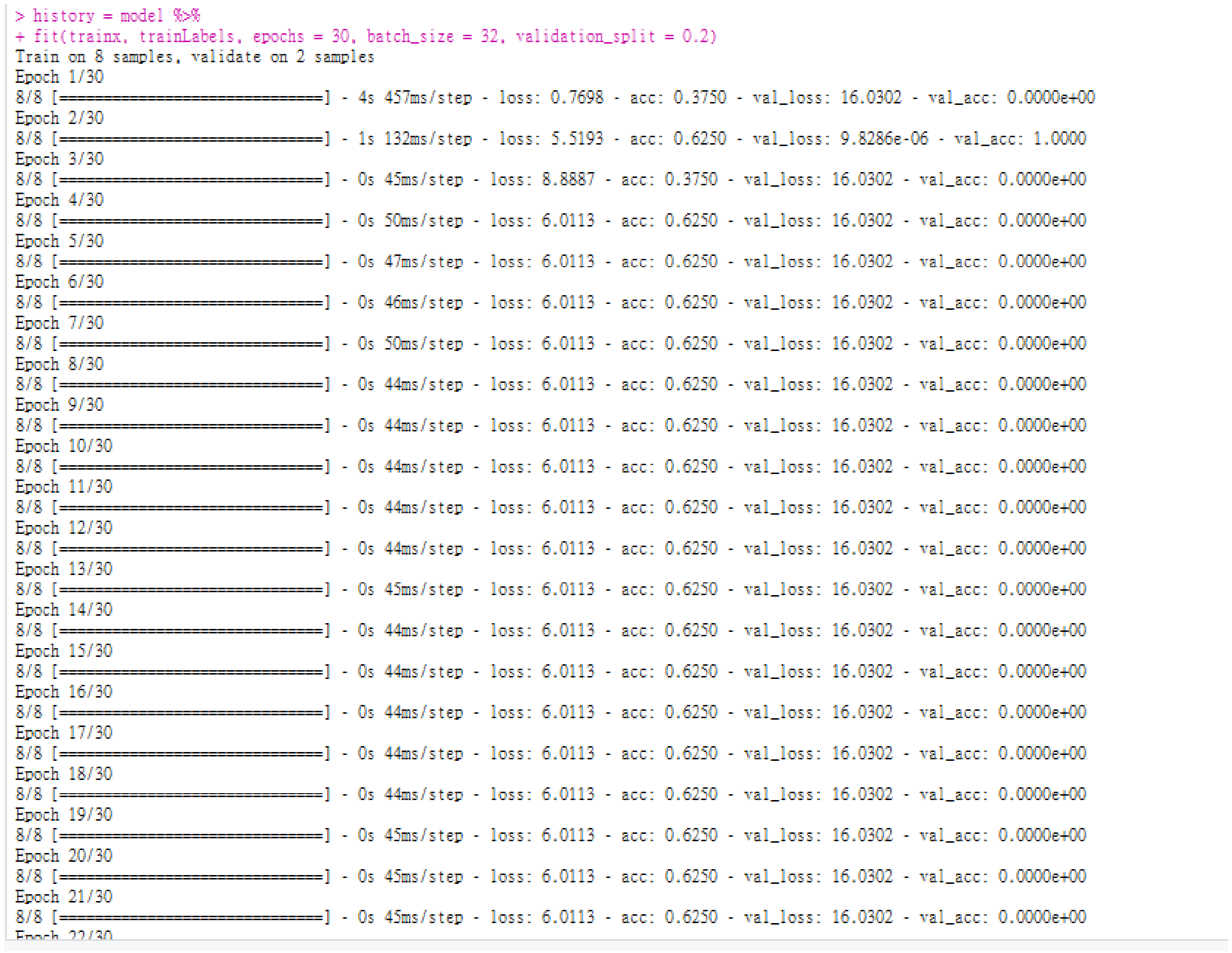
**Fit the model -**

Trainx = for supplying the independent variables
TrainLables = for training related response that we have created through one hot encoding
Epochs = no of Iteration is 30
Validation_split = 20 %

Model %>%
fit(trainx, trainLabels, epochs = 30, batch_size = 32, validation_split = 0.2)



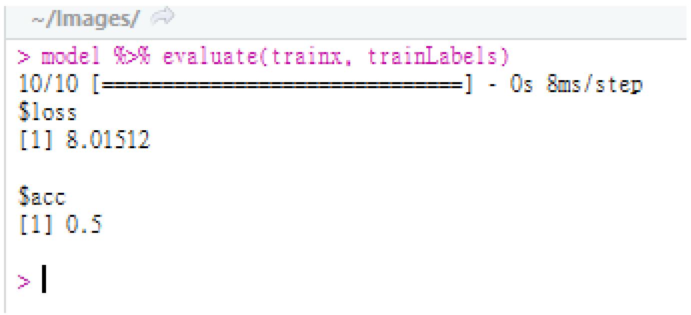
```
~/Images/
> history = model %>%
+ fit(trainx, trainLabels, epochs = 30, batch_size = 32, validation_split = 0.2)
```



```
> history = model %>%
+ fit(trainx, trainLabels, epochs = 30, batch_size = 32, validation_split = 0.2)
Train on 8 samples, validate on 2 samples
Epoch 1/30
8/8 [==============================] - 4s 457ms/step - loss: 0.7698 - acc: 0.3750 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 2/30
8/8 [==============================] - 1s 132ms/step - loss: 5.5193 - acc: 0.6250 - val_loss: 9.8286e-06 - val_acc: 1.0000
Epoch 3/30
8/8 [==============================] - 0s 45ms/step - loss: 8.8887 - acc: 0.3750 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 4/30
8/8 [==============================] - 0s 50ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 5/30
8/8 [==============================] - 0s 47ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 6/30
8/8 [==============================] - 0s 46ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 7/30
8/8 [==============================] - 0s 50ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 8/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 9/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 10/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 11/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 12/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 13/30
8/8 [==============================] - 0s 45ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 14/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 15/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 16/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 17/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 18/30
8/8 [==============================] - 0s 44ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 19/30
8/8 [==============================] - 0s 45ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 20/30
8/8 [==============================] - 0s 45ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 21/30
8/8 [==============================] - 0s 45ms/step - loss: 6.0113 - acc: 0.6250 - val_loss: 16.0302 - val_acc: 0.0000e+00
Epoch 22/30
```

**Evaluation and Prediction - Train Data**

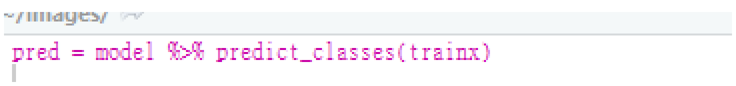
model %>% evaluate(trainx, trainLabels)



```
~/Images/
> model %>% evaluate(trainx, trainLabels)
10/10 [==============================] - 0s 8ms/step
$loss
[1] 8.01512

$acc
[1] 0.5

>
```

So the 50 % of the classifications are correct .

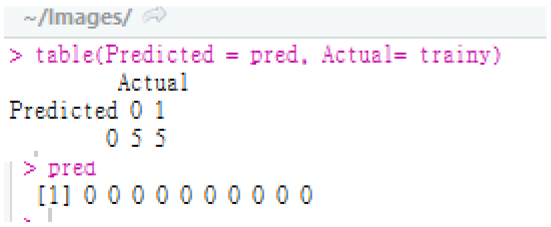
> pred = model %>% predict_classes(trainx)



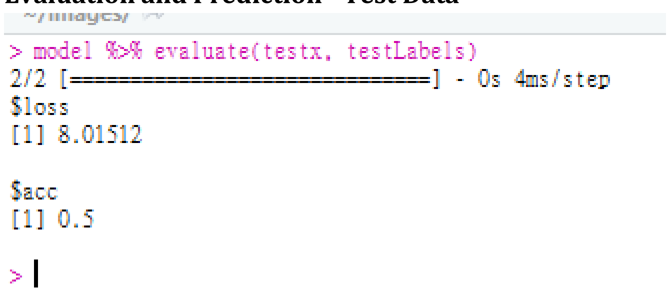
```
~/Images/
pred = model %>% predict_classes(trainx)
```

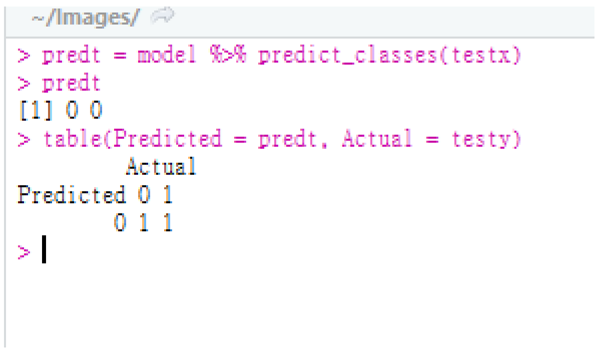
Print the table -

> table(Predicted = pred, Actual= trainy)


```
~/Images/
> table(Predicted = pred, Actual= trainy)
         Actual
Predicted 0 1
        0 5 5
> pred
 [1] 0 0 0 0 0 0 0 0 0 0
```

**Evaluation and Prediction - Test Data**


```
> model %>% evaluate(testx, testLabels)
2/2 [==============================] - 0s 4ms/step
$loss
[1] 8.01512

$acc
[1] 0.5

>
```


```
~/Images/
> predt = model %>% predict_classes(testx)
> predt
[1] 0 0
> table(Predicted = predt, Actual = testy)
         Actual
Predicted 0 1
        0 1 1
>
```

Loss - Loss value implies how well or poorly a certain model behaves after each iteration of optimization

**Trend analysis and forecasting -**

Forecasting is the process of making predictions of the future based on past and present data and most commonly by analysis of trends.

**Why do we need Time Series Analysis ?**

Comparing to other forcast algorithms, like logistic or linear , they deal with two variables , but with this tme series we deal with a single variable which is dependent on time.
For example , I own a car company and i say that last month i sold around 100 cars and this month i sold 70 cars , so what is the sale that i will achieve in the next month.

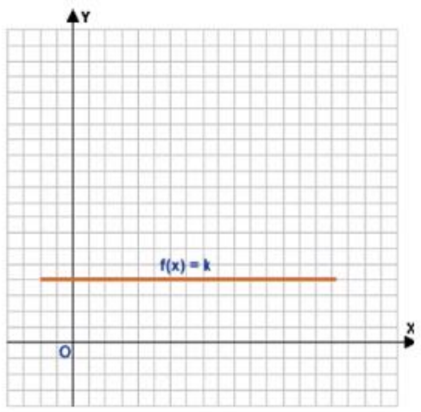
**What is Time Series Analysis ?**

A time series is a series of data points indexed in time order. Time series forecasting is the use of a model to predict future values based on previous observed values .

We use time series algorithm to create a model and use that model to predict the future values.

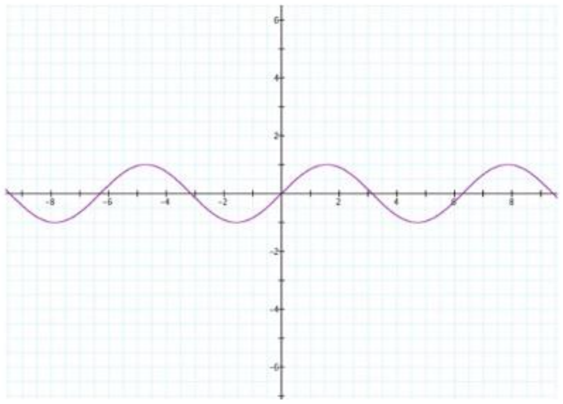
**When not to use the time series analysis ?**

- When Values are constant.



When the values are constant, for example my company sold 25 cars in previous month and 25 cars in current month , so what is the sale is going to be in the next month ?
So in this case number of cars in the previous month and the current month is same so there is no need to do TS analysis .
Even if you want to do the analysis then there i something that is going to be in the next month may be there is discount in the next month.

- When Values can be represented using known functions such as sin x.



IF your values can be representing a function for example sinx function , So if you have the x value you can get it by putting it to the function and you can apply TS analysis in this as well because you could have get that value by putting in the function itself.

**What are the cases in which you cant use the TS analysis ?**

So if your data is not stationary, For series to be stationary there are three conditions -

- Mean(average) should be constant according to the time.

- Variance should be equal at different time interval from the mean.
- Covariance should be equal .

Variance - Basically distance, each point distance from the mean should be equal
At equal interval of time . How far each number is from the mean.

**Components of TS Analysis -**

General Trend - Direction in which something is developing or changing
Seasonal - A peak or dip which is seen in a time interval.
Irregular Fluctuation - Uncontrolled situations which arises due to which the value changes.

An script has been attached TSA.R in the Machine Learning folder - Please import in the RStudio to see how it is done .

**Dimensionality Reduction -**

dimensionality reduction or dimension reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

**Why Dimension Reduction is important in machine learning & predictive modeling?**

The problem of unwanted increase in dimension is closely related to fixation of measuring / recording data at a far granular level then it was done in past. This is no way suggesting that this is a recent problem. It has started gaining more importance lately due to surge in data.

Lately, there has been a tremendous increase in the way sensors are being used in the industry. These sensors continuously record data and store it for analysis at a later point. In the way data gets captured, there can be a lot of redundancy. This is the problem of high unwanted dimensions and needs a treatment of dimension reduction.

Let’s look at other examples of new ways of data collection:

- Casinos are capturing data using cameras and tracking each and every move of their customers.
- Political parties are capturing data by expanding their reach on field
- Your smart phone apps collects a lot of personal details about you
- Your set top box collects data about which programs preferences and timings
- Organizations are evaluating their brand value by social media engagements (comments, likes), followers, positive and negative sentiments

With more variables, comes more trouble and to avoid this trouble, dimension reduction techniques comes to the rescue.

**What is Dimension Reduction**

Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely. These techniques are typically used while solving machine learning problems to obtain better features for a classification or regression task.

**PCA Advantages -**

- Useful for dimension reduction for high dimensional data analysis
- Helps reduce the number of predictor items using principal components
- Helps to make predictor items independent & avoid multicollinearity problem.
- Allows interpretation of many variables using a 2 - dimensional biplot.
- Can be used for developing the predictions model.

**PCA Disadvantages -**

- Only numeric variables can be used
- Prediction models are less interpretable

**Please use the uploaded R script (D_Reduction.R) for coding and analysis purpose in the drive folder.**