**Machine Learning Project – How to Detect Credit Card Fraud**

The aim of this R project is to build a classifier that can detect credit card fraudulent transactions. We will use a variety of [***machine learning algorithms***](https://data-flair.training/blogs/machine-learning-algorithm/) that will be able to discern fraudulent from non-fraudulent one. By the end of this machine learning project, you will learn how to implement machine learning algorithms to perform classification.

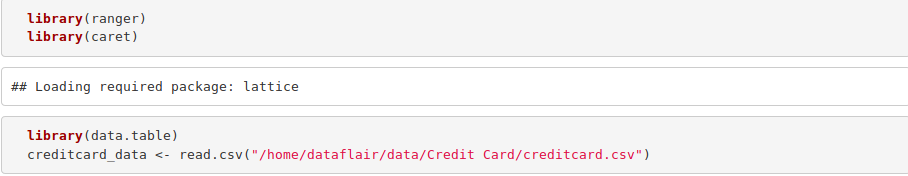
1. Importing the Datasets

We are importing the datasets that contain transactions made by credit cards-

**Code:**

1. **library**(ranger)
2. **library**(caret)
3. **library**(data.table)
4. creditcard\_data <- read.**csv**("/home/dataflair/data/Credit Card/creditcard.csv")

**Input Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Library.png)

***Before moving on, you must revise the***[***concepts of R Dataframes***](https://data-flair.training/blogs/r-data-frame-introduction-operations/)

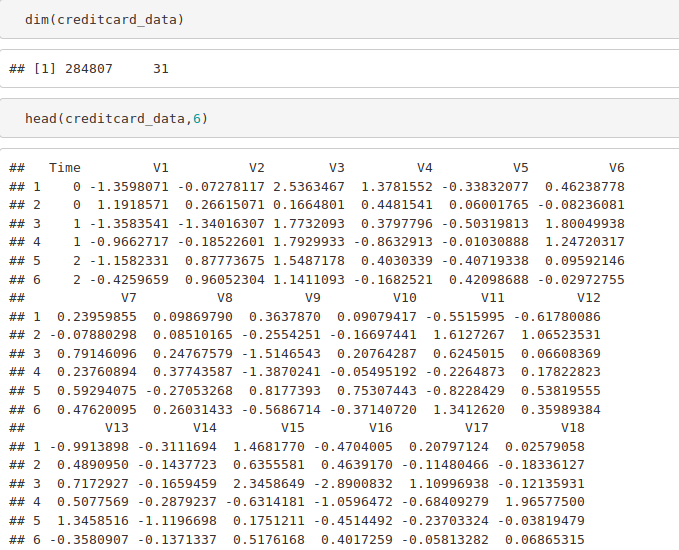
2. Data Exploration

In this section of the fraud detection ML project, we will explore the data that is contained in the creditcard\_data dataframe. We will proceed by displaying the creditcard\_data using the head() function as well as the tail() function. We will then proceed to explore the other components of this dataframe –

**Code:**

1. **dim**(creditcard\_data)
2. **head**(creditcard\_data,6)

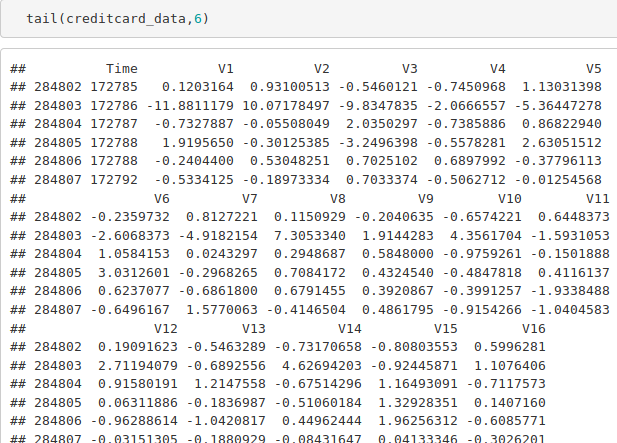
**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/data-Exploration-in-data-science-project.png)

**Code:**

1. **tail**(creditcard\_data,6)

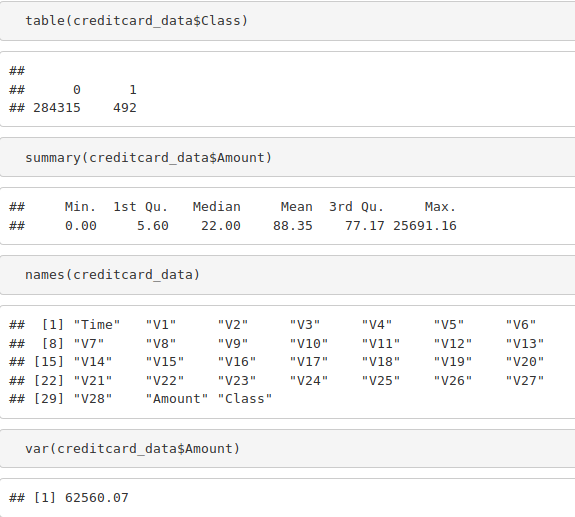
**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/data-exploration-1.png)

**Code:**

1. **table**(creditcard\_data$Class)
2. **summary**(creditcard\_data$Amount)
3. **names**(creditcard\_data)
4. **var**(creditcard\_data$Amount)

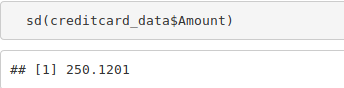
**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Data-Exploration-2.png)

**Code:**

1. **sd**(creditcard\_data$Amount)

**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/data-exploration-3.png)

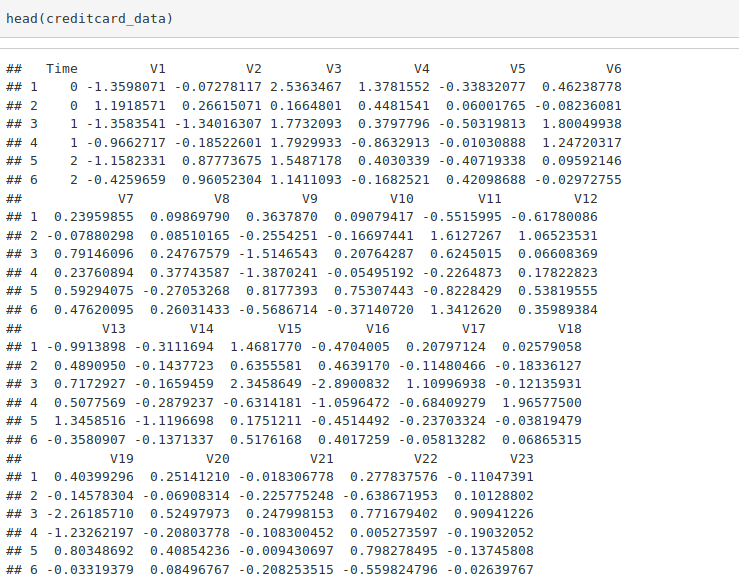
3. Data Manipulation

In this section of the R data science project, we will scale our data using the scale() function. We will apply this to the amount component of our creditcard\_data amount. Scaling is also known as feature standardization. With the help of scaling, the data is structured according to a specified range. Therefore, there are no extreme values in our dataset that might interfere with the functioning of our model. We will carry this out as follows:

**Code:**

1. **head**(creditcard\_data)

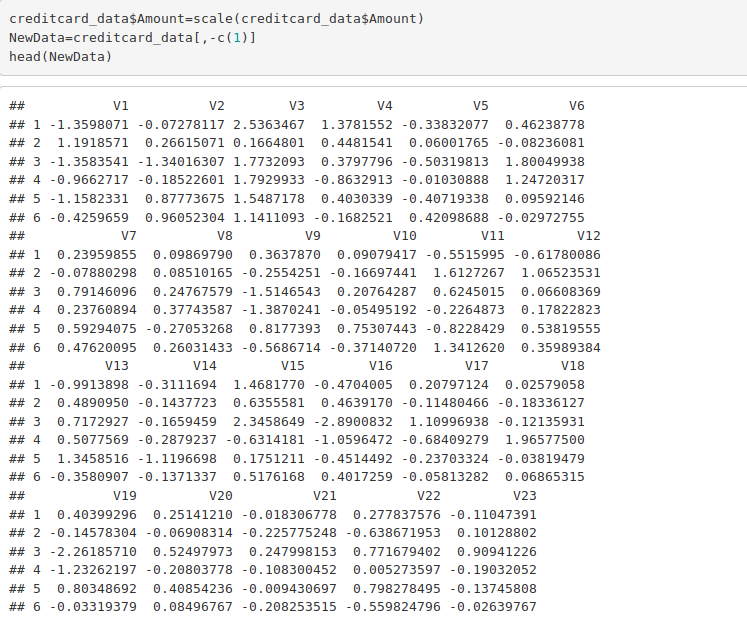
**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Data-Manipulation-1.png)

**Code:**

1. creditcard\_data$Amount=**scale**(creditcard\_data$Amount)
2. NewData=creditcard\_data[,-**c**(1)]
3. **head**(NewData)

**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Data-Manipulation-2.png)

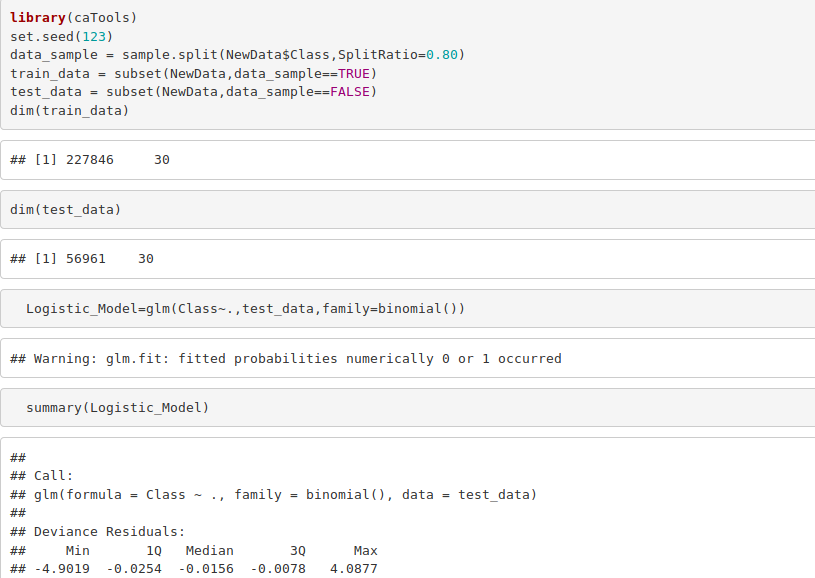
4. Data Modeling

After we have standardized our entire dataset, we will split our dataset into training set as well as test set with a split ratio of 0.80. This means that 80% of our data will be attributed to the train\_data whereas 20% will be attributed to the test data. We will then find the dimensions using the dim() function –

**Code:**

1. **library**(caTools)
2. set.**seed**(123)
3. data\_sample = sample.**split**(NewData$Class,SplitRatio=0.80)
4. train\_data = **subset**(NewData,data\_sample==TRUE)
5. test\_data = **subset**(NewData,data\_sample==FALSE)
6. **dim**(train\_data)
7. **dim**(test\_data)

**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Data-Modeling-1.png)

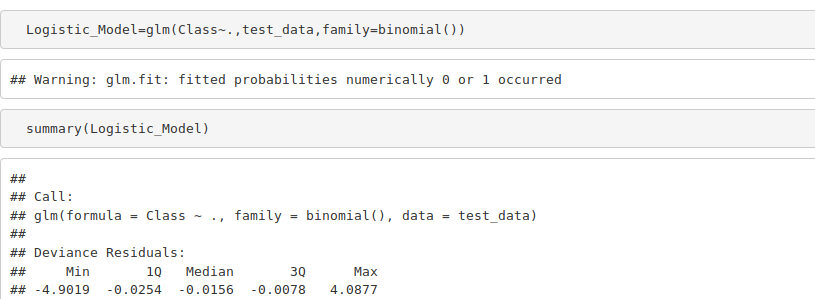
5. Fitting Logistic Regression Model

In this section of credit card fraud detection project, we will fit our first model. We will begin with logistic regression. A logistic regression is used for modeling the outcome probability of a class such as pass/fail, positive/negative and in our case – fraud/not fraud. We proceed to implement this model on our test data as follows –

**Code:**

1. Logistic\_Model=**glm**(Class~.,test\_data,family=**binomial**())
2. **summary**(Logistic\_Model)

**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Logistic-Model.png)

After we have summarised our model, we will visual it through the following plots –

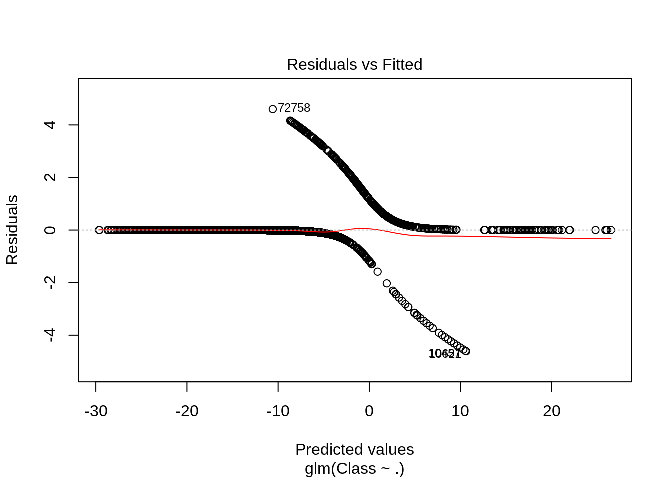
**Code:**

1. **plot**(Logistic\_Model)

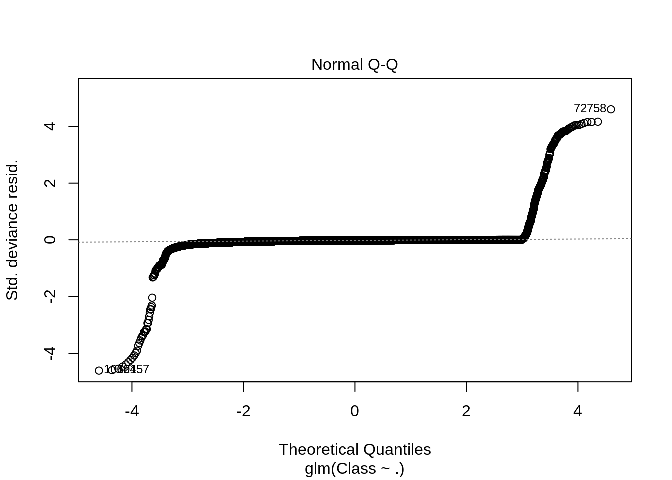
**Input Screenshot:**

[Data Modeling ](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Data-Modeling-2.png)

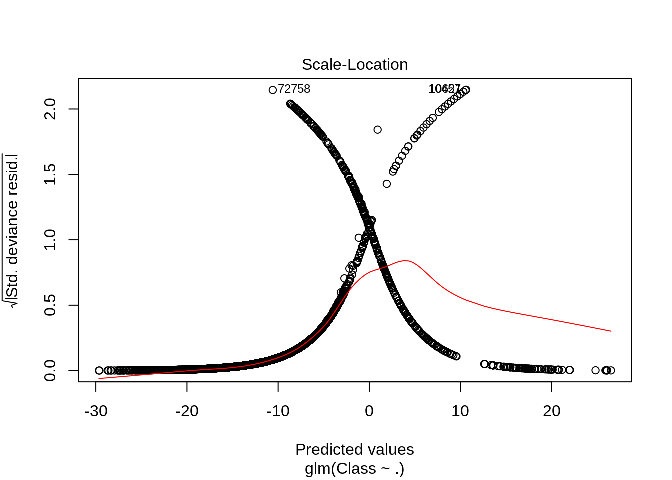
**Output:**



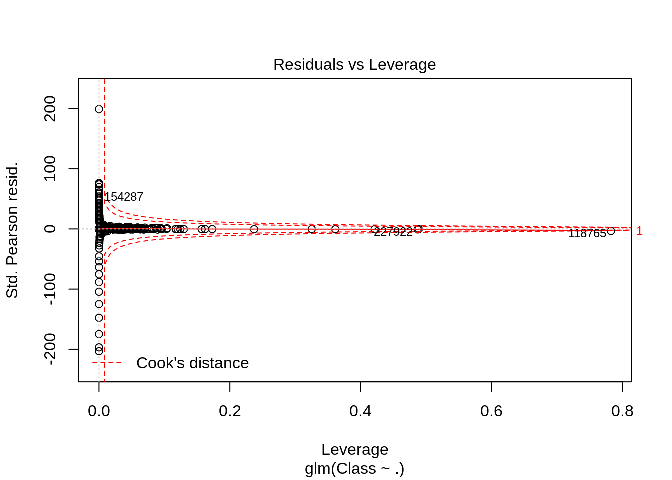
**Output:**



**Output:**



**Output:**

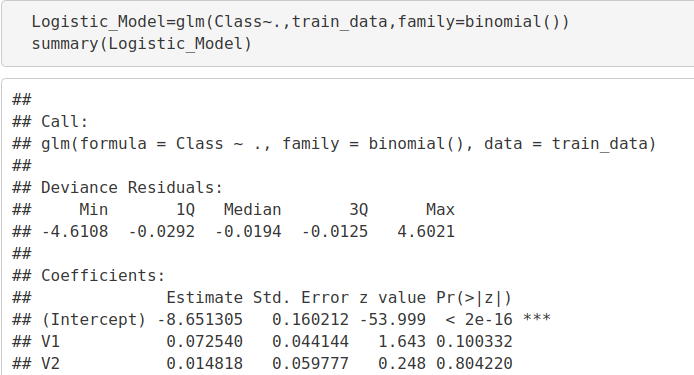


In order to assess the performance of our model, we will delineate the ROC curve. ROC is also known as Receiver Optimistic Characteristics. For this, we will first import the ROC package and then plot our [ROC](https://en.wikipedia.org/wiki/Receiver_operating_characteristic) curve to analyze its performance.

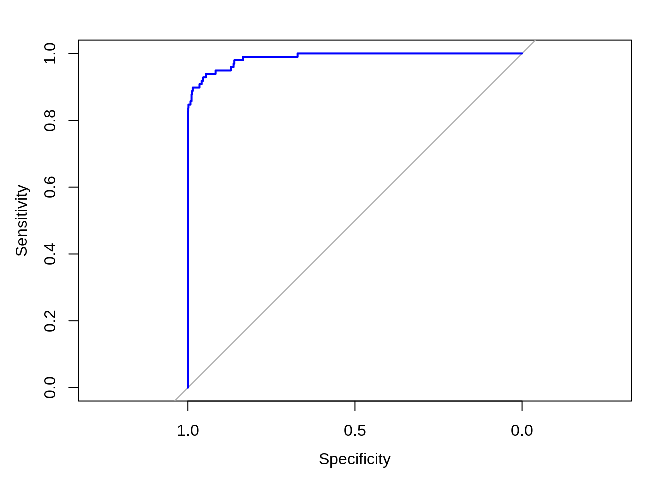
**Code:**

1. **library**(pROC)
2. lr.predict <- **predict**(Logistic\_Model,train\_data, probability = TRUE)
3. auc.gbm = **roc**(test\_data$Class, lr.predict, plot = TRUE, col = "blue")

**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/logistic-ROC.png)

**Output:**



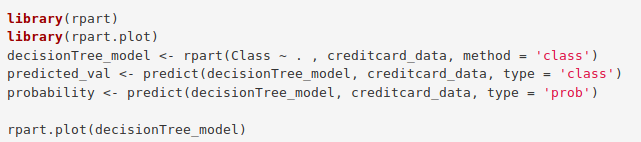
6. Fitting a Decision Tree Model

In this section, we will implement a decision tree algorithm. [***Decision Trees***](https://data-flair.training/blogs/r-decision-trees/) to plot the outcomes of a decision. These outcomes are basically a consequence through which we can conclude as to what class the object belongs to. We will now implement our decision tree model and will plot it using the rpart.plot() function. We will specifically use the recursive parting to plot the decision tree.

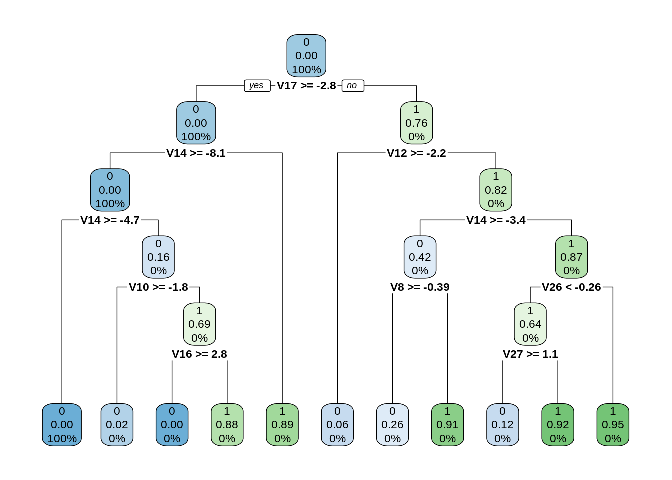
**Code:**

1. **library**(rpart)
2. **library**(rpart.plot)
3. decisionTree\_model <- **rpart**(Class ~ . , creditcard\_data, method = 'class')
4. predicted\_val <- **predict**(decisionTree\_model, creditcard\_data, type = 'class')
5. probability <- **predict**(decisionTree\_model, creditcard\_data, type = 'prob')
6. rpart.**plot**(decisionTree\_model)

**Input Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/DecisionTree-ML-Code.png)

**Output:**



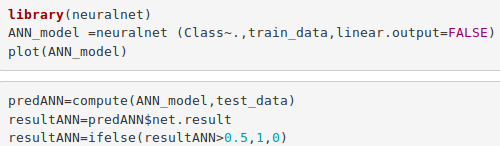
7. Artificial Neural Network

[***Artificial Neural Networks***](https://data-flair.training/blogs/artificial-neural-network/) are a type of machine learning algorithm that are modeled after the human nervous system. The ANN models are able to learn the patterns using the historical data and are able to perform classification on the input data. We import the neuralnet package that would allow us to implement our ANNs. Then we proceeded to plot it using the plot() function. Now, in the case of Artificial Neural Networks, there is a range of values that is between 1 and 0. We set a threshold as 0.5, that is, values above 0.5 will correspond to 1 and the rest will be 0. We implement this as follows –

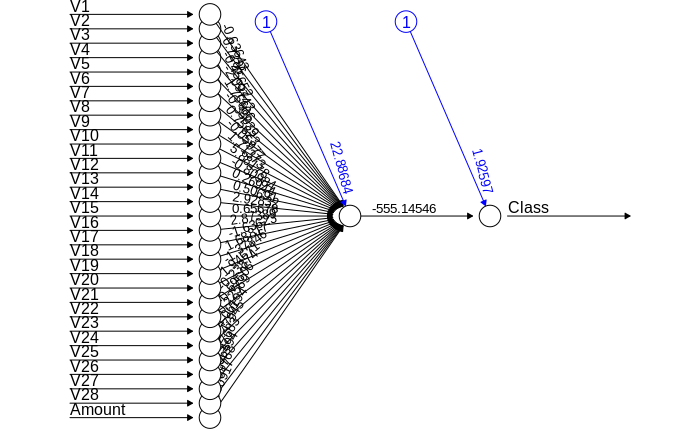
**Code:**

1. **library**(neuralnet)
2. ANN\_model =**neuralnet** (Class~.,train\_data,linear.output=FALSE)
3. **plot**(ANN\_model)
4. predANN=**compute**(ANN\_model,test\_data)
5. resultANN=predANN$net.result
6. resultANN=**ifelse**(resultANN>0.5,1,0)

**Input Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Neural-Networks.png)

**Output:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Neural-Network.png)

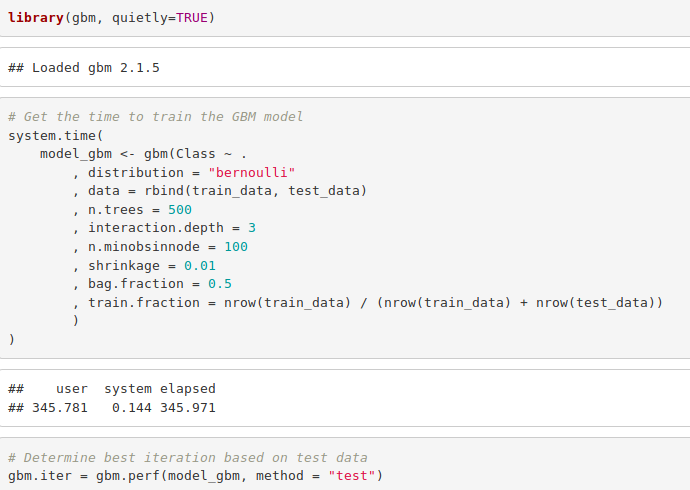
8. Gradient Boosting (GBM)

[***Gradient Boosting***](https://data-flair.training/blogs/gradient-boosting-algorithm/) is a popular machine learning algorithm that is used to perform classification and regression tasks. This model comprises of several underlying ensemble models like weak decision trees. These decision trees combine together to form a strong model of gradient boosting. We will implement gradient descent algorithm in our model as follows –

**Code:**

1. **library**(gbm, quietly=TRUE)
2. # Get the time to train the GBM model
3. system.**time**(
4. model\_gbm <- **gbm**(Class ~ .
5. , distribution = "bernoulli"
6. , data = **rbind**(train\_data, test\_data)
7. , n.trees = 500
8. , interaction.depth = 3
9. , n.minobsinnode = 100
10. , shrinkage = 0.01
11. , bag.fraction = 0.5
12. , train.fraction = **nrow**(train\_data) / (**nrow**(train\_data) + **nrow**(test\_data))
13. )
14. )
15. # Determine best iteration based on test data
16. gbm.iter = gbm.**perf**(model\_gbm, method = "test")

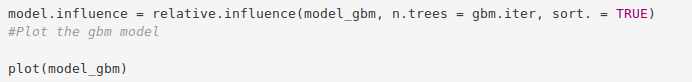
**Input Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/Gradient-boosting-in-R-project.png)

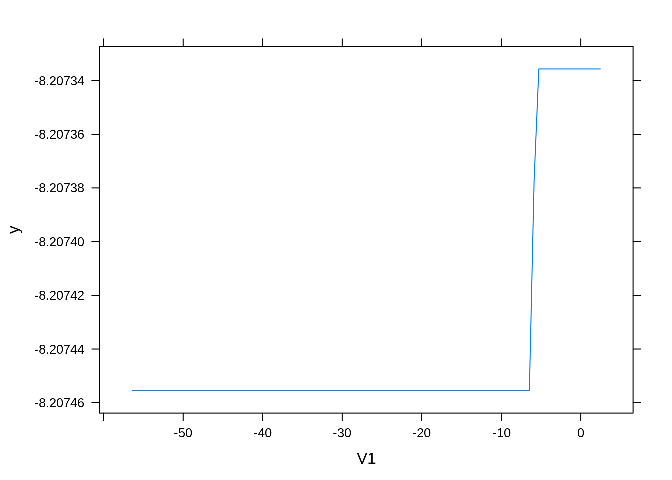
**Code:**

1. model.influence = relative.**influence**(model\_gbm, n.trees = gbm.iter, sort. = TRUE)
2. #Plot the gbm model
3. **plot**(model\_gbm)

**Input Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/gradient-boosting-plotting-in-R.png)

**Output:**

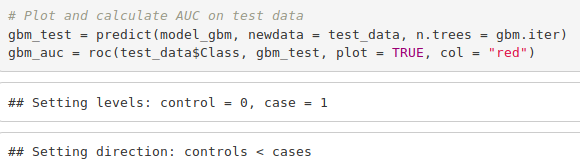


**Output:**

**Code:**

1. # Plot and calculate AUC on test data
2. gbm\_test = **predict**(model\_gbm, newdata = test\_data, n.trees = gbm.iter)
3. gbm\_auc = **roc**(test\_data$Class, gbm\_test, plot = TRUE, col = "red")

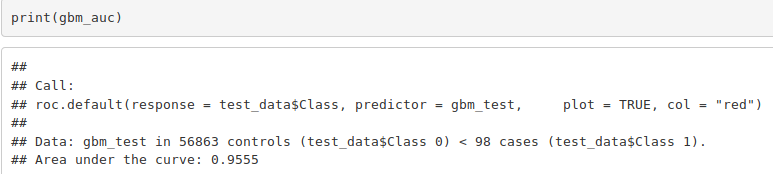
**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/gbm-input.png)

**Code:**

1. **print**(gbm\_auc)

**Output Screenshot:**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/07/gbm-auc.png)

Summary