Statistical Data Mining 1

Assignment 5

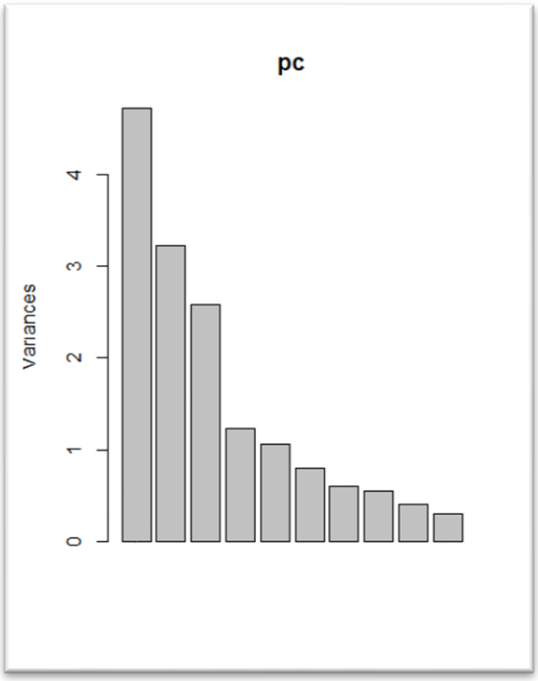
Name: Shreeram G S

UBId: 50413349

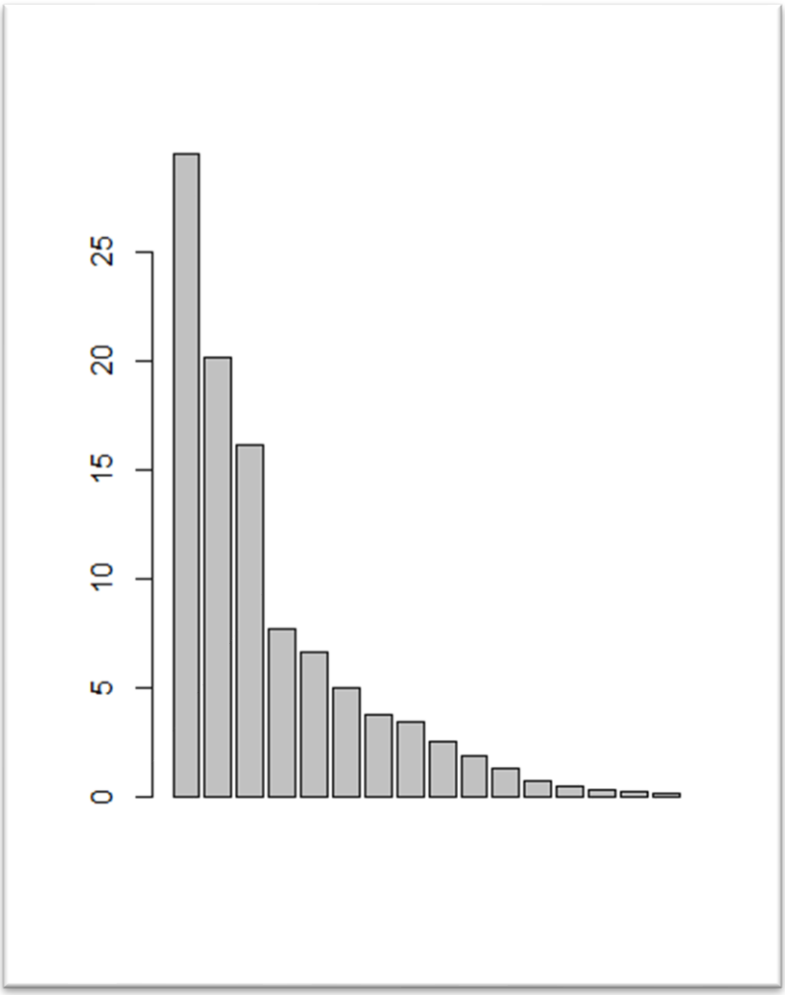
Q1) a)

```
a <- get(load("C:/Users/Sriram/Desktop/R projects/hw5/pendigits.RData"))


Inputs = a[,1:16]
label = a[,17]
Inputs = scale(Inputs)
pc = prcomp(Inputs,center = FALSE,scale. = FALSE)
plot(pc)

var = (pc$sdev)^2
var_per = (var/sum(var))*100
barplot(var_per)
var_per
 [1] 29.4786648 20.1819506 16.1050394  7.6858863  6.6443705  4.9833709  3.7915727  3.4265723  2.5676505  1.8639429  1.3157041  0.7018557  0.5030066  0.3625133  0.2141531
[16]  0.1737463

biplot(pc)

library(plotly)
plot_ly(x=pc$x[,1], y=pc$x[,2], z=pc$x[,3],mode = "markers",type = "scatter3d",color = label)
```

In this question we first load the pendigits data set to a variable a. Then we take inputs from 1 to 16 and the rest labels. We then use the scale function to scale the inputs. Later we use prcomp, which takes the input data, and standardize the input data, so that it could have zero mean and variance one before doing PCA.Then once we do this we plot the resultant pc. The plot is shown below:

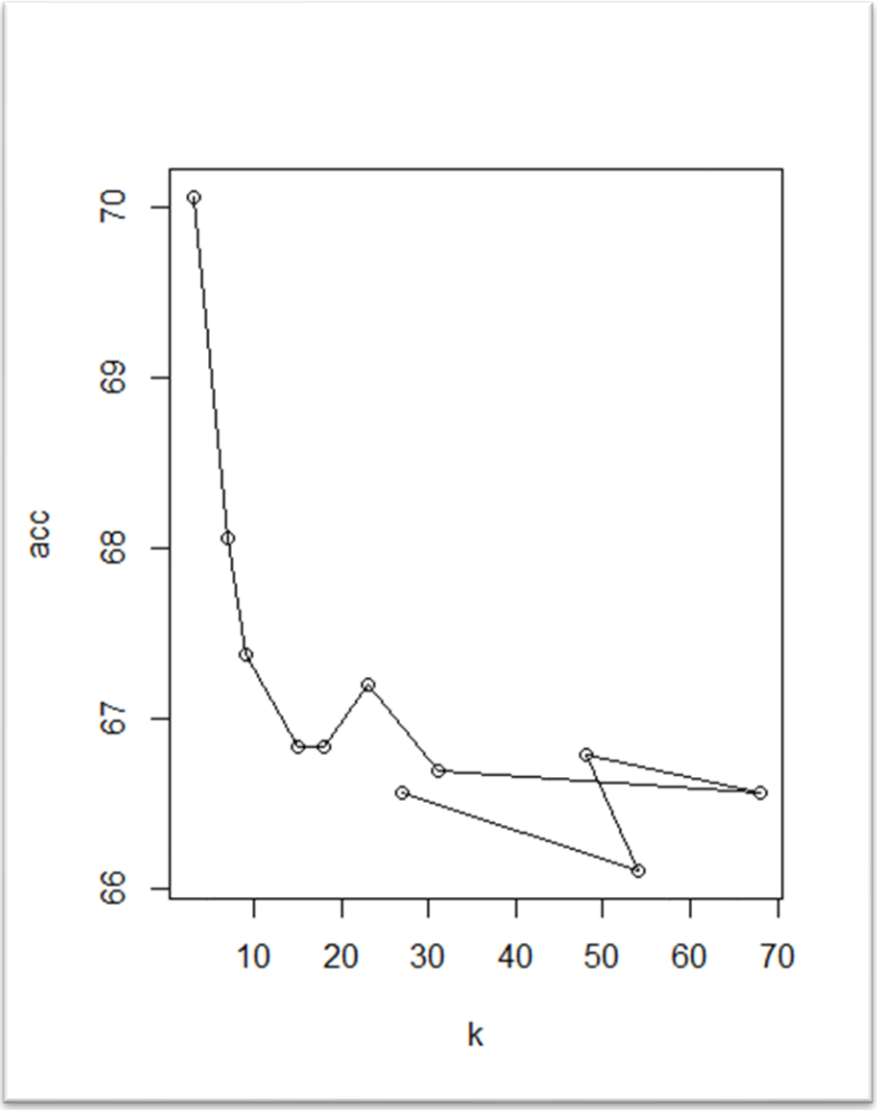


Later we use another variable called var, where the dev col of the pc is squared. Later we perform the above operations to plot a barplot inorder to gain better insights from the ne visual data representation. The project barplot is as follows:



b) Next for K nearest neighbours we first create a confusion matrix c. The basic libraries required for this part is caret (Classification and REgression Training) which has the confusion matrix function. So first we create a variable called split to perform a split limit and use this limit to partition the data set into test and train. Next we use the knn function and fit the model on just the raw data and consequently plot respective graph. Once we generate the confusion-matrix we further apply min function to get the min value which is nothing but he test error value.

```
> library(class)
> library(caret)
> set.seed(1)
> split = sample(1:nrow(a), 0.2*nrow(a))
> k_test = a[split,]
> k_train = a[-split,]
> k = c(3,7,9,15,18,23,31,68,48,54,27)
> acc = c()
> for(i in 1:length(k)){
+   model = knn(k_train[,-c(1)],k_test[,-c(1)],as.factor(k_train[,1]),k =k[i]
+   c =confusionMatrix(data=model,reference=as.factor(k_test[,1]))
+   acc[i]=((1-c$overall[1])*100)
+ }
> min(acc)
[1] 66.10555
> plot(k,acc,type='o')
```

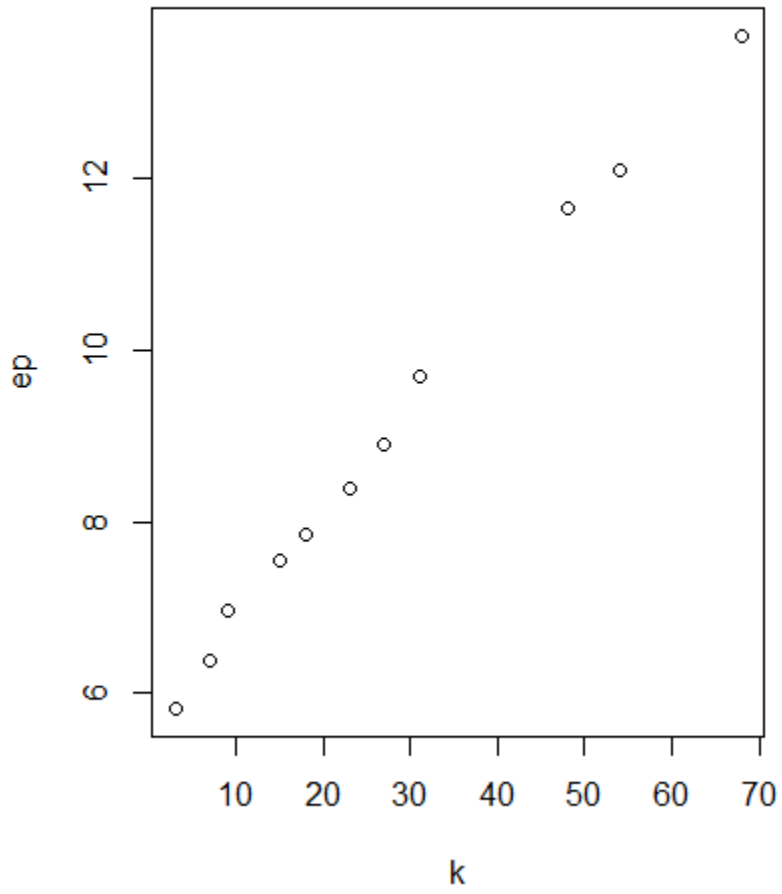


- PCs from part (A) that capture at least 80% of the variation.

```
> dats=data.frame(pc$x[,1:5],a$class)
> dats$a.class=as.factor(dats$a.class)
>
> trainl=dats$a.class[split]
> testl=dats$a.class[-split]
> train_pc=dats[split,]
> test_pc=dats[-split,]
>
> ep=c()
>
> for(i in 1:length(k)){
+   knn_model1<-knn(train_pc[,-c(6)],test_pc[,-c(6)],train_pc[,6],k =k[i])
+   c1 =confusionMatrix(data=knn_model1,reference=testl)
+   ep[i]=(1-c1$overall[1])*100
+ }
> min(ep)
[1] 5.822151
> plot(k,ep)
```

Here we see after changing parameters inn kn function while making a model, the new and improvised test error rate has drastically reduced to 5.822%, and we now get much smoother linear graph when plotted between the k values and the test error values.

The plot for the above is as follows:



c) We consider randomforrest as another classifier in order to compare with the above classifier.

```
> k_train$class = as.factor(k_train$class)
> r.model = randomForest(class~., data=k_train,ntree=1000)
> ran_pred= predict(r.model, k_test, type = "response")
> ran_errorrate= mean(ran_pred != k_test$class)
> ran_errorrate
[1] 0.004549591
> acc1<-length(which(ran_pred==k_test$class))/length(k_test$class)
> rand_test_error<-(1-acc1)*100
> rand_test_error
[1] 0.4549591
>
```

Q2)

First before we start, we require few essential packages to be loaded into our R Script for further processing. These packages are:

1) **randomForest**: The R package "randomForest" is used to create random forests.
2) **Neuralnet**: A neural network is a computational system that creates predictions based on existing data.
3) **ggplot2**: ggplot2 is a R package dedicated to data visualization
4) **rpart**: Rpart is a powerful machine learning library in R that is used for building classification and regression trees
5) **GGally**: the R package 'ggplot2' is a plotting system based on the grammar of graphics
6) **Caret**: The R package 'ggplot2' is a plotting system based on the grammar of graphics
7) **nnet**: Software for feed-forward neural networks with a single hidden layer, and for multinomial log-linear models

```
> cl <- get(load( cleveland.RData ))
> str(cl)
'data.frame':   296 obs. of  15 variables:
 $ age     : int  63 67 67 37 41 56 62 57 63 53 ...
 $ gender  : Factor w/ 2 levels "fem","male": 2 2 2 2 1 2 1 1 2 2 ...
 $ cp      : Factor w/ 4 levels "abnang","angina",..: 2 3 3 4 1 1 3 3 3 3 ...
 $ trestbps: int  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : int  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : Factor w/ 2 levels "fal","true": 2 1 1 1 1 1 1 1 1 2 ...
 $ restecg : Factor w/ 3 levels "abn","hyp","norm": 2 2 2 3 2 3 2 3 2 2 ...
 $ thalach : int  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : Factor w/ 2 levels "fal","true": 1 2 2 1 1 1 1 2 1 2 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : Factor w/ 3 levels "down","flat",..: 1 2 2 1 3 3 1 3 2 1 ...
 $ ca      : int  0 3 2 0 0 0 2 0 1 0 ...
 $ thal    : Factor w/ 3 levels "fix","norm","rev": 1 2 3 2 2 2 2 2 3 3 ...
 $ diag1   : Factor w/ 2 levels "buff","sick": 1 2 2 1 1 1 2 1 2 2 ...
 $ diag2   : Factor w/ 5 levels "H","S1","S2",..: 1 3 2 1 1 1 4 1 3 2 ...
```

```
> dim(cl)
[1] 296  15
> summary(cl)
      age        gender          cp        trestbps         chol         fbs        restecg      thalach        exang       oldpeak        slope          ca           thal
 Min.   :29.00   fem : 95   abnang: 49   Min.   : 94.0   Min.   :126.0   fal :253   abn :  4   Min.   : 71.0   fal :199   Min.   :0.000   down: 21   Min.   :0.0000   fix : 18
 1st Qu.:48.00   male:201   angina: 23   1st Qu.:120.0   1st Qu.:211.0   true: 43   hyp :145   1st Qu.:133.0   true: 97   1st Qu.:0.000   flat:137   1st Qu.:0.0000   norm:163
 Median :56.00              asympt:141   Median :130.0   Median :242.5              norm:147   Median :152.5              Median :0.800   up  :138   Median :0.0000   rev :115
 Mean   :54.52              notang: 83   Mean   :131.6   Mean   :247.2                         Mean   :149.6              Mean   :1.059              Mean   :0.6791
 3rd Qu.:61.00                           3rd Qu.:140.0   3rd Qu.:275.2                         3rd Qu.:166.0              3rd Qu.:1.650              3rd Qu.:1.0000
 Max.   :77.00                           Max.   :200.0   Max.   :564.0                         Max.   :202.0              Max.   :6.200              Max.   :3.0000
  diag1     diag2
 buff:160   H :160
 sick:136   S1: 53
            S2: 35
            S3: 35
            S4: 13

>
```

The above image shows us that the cleveland data has a dimensionality of [1] 296 14, through the summary function we get to know about the various dataframes involved in this data set.

```
> y$diag2 <- NULL
> str(y)
'data.frame':   296 obs. of  14 variables:
 $ age     : int  63 67 67 37 41 56 62 57 63 53 ...
 $ gender  : Factor w/ 2 levels "fem","male": 2 2 2 2 1 2 1 1 2 2 ...
 $ cp      : Factor w/ 4 levels "abnang","angina",..: 2 3 3 4 1 1 3 3 3 3 ...
 $ trestbps: int  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : int  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : Factor w/ 2 levels "fal","true": 2 1 1 1 1 1 1 1 1 2 ...
 $ restecg : Factor w/ 3 levels "abn","hyp","norm": 2 2 2 3 2 3 2 3 2 2 ...
 $ thalach : int  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : Factor w/ 2 levels "fal","true": 1 2 2 1 1 1 1 2 1 2 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : Factor w/ 3 levels "down","flat",..: 1 2 2 1 3 3 1 3 2 1 ...
 $ ca      : int  0 3 2 0 0 0 2 0 1 0 ...
 $ thal    : Factor w/ 3 levels "fix","norm","rev": 1 2 3 2 2 2 2 2 3 3 ...
 $ diag1   : Factor w/ 2 levels "buff","sick": 1 2 2 1 1 1 2 1 2 2 ...
> training <- createDataPartition(y$diag1, p=0.70, list=FALSE)
> y_train <- y[ training,]
> y_test <- y[-training,]
> str(y_train)
'data.frame':   208 obs. of  14 variables:
 $ age     : int  63 37 41 56 62 57 57 56 44 64 ...
 $ gender  : Factor w/ 2 levels "fem","male": 2 2 1 2 1 1 2 2 2 2 ...
 $ cp      : Factor w/ 4 levels "abnang","angina",..: 2 4 1 1 3 3 3 4 1 2 ...
 $ trestbps: int  145 130 130 120 140 120 140 130 120 110 ...
 $ chol    : int  233 250 204 236 268 354 192 256 263 211 ...
 $ fbs     : Factor w/ 2 levels "fal","true": 2 1 1 1 1 1 1 2 1 1 ...
 $ restecg : Factor w/ 3 levels "abn","hyp","norm": 2 3 2 3 2 3 3 2 3 2 ...
 $ thalach : int  150 187 172 178 160 163 148 142 173 144 ...
 $ exang   : Factor w/ 2 levels "fal","true": 1 1 1 1 1 2 1 2 1 2 ...
 $ oldpeak : num  2.3 3.5 1.4 0.8 3.6 0.6 0.4 0.6 0 1.8 ...
 $ slope   : Factor w/ 3 levels "down","flat",..: 1 1 3 3 1 3 2 2 3 2 ...
 $ ca      : int  0 0 0 0 2 0 0 1 0 0 ...
 $ thal    : Factor w/ 3 levels "fix","norm","rev": 1 2 2 2 2 2 1 1 3 2 ...
 $ diag1   : Factor w/ 2 levels "buff","sick": 1 1 1 1 2 1 1 2 1 1 ...
> str(y_test)
'data.frame':   88 obs. of  14 variables:
 $ age     : int  67 67 63 53 56 49 58 66 60 59 ...
 $ gender  : Factor w/ 2 levels "fem","male": 2 2 2 2 1 2 2 1 2 2 ...
 $ cp      : Factor w/ 4 levels "abnang","angina",..: 3 3 3 3 1 1 4 2 3 3 ...
 $ trestbps: int  160 120 130 140 140 130 132 150 117 135 ...
 $ chol    : int  286 229 254 203 294 266 224 226 230 234 ...
 $ fbs     : Factor w/ 2 levels "fal","true": 1 1 1 2 1 1 1 1 2 1 ...
 $ restecg : Factor w/ 3 levels "abn","hyp","norm": 2 2 2 2 2 3 2 3 3 3 ...
 $ thalach : int  108 129 147 155 153 171 173 114 160 161 ...
 $ exang   : Factor w/ 2 levels "fal","true": 2 2 1 2 1 1 1 1 2 1 ...
 $ oldpeak : num  1.5 2.6 1.4 3.1 1.3 0.6 3.2 2.6 1.4 0.5 ...
 $ slope   : Factor w/ 3 levels "down","flat",..: 2 2 2 1 2 3 3 1 3 2 ...
 $ ca      : int  3 2 1 0 0 0 2 0 2 0 ...
 $ thal    : Factor w/ 3 levels "fix","norm","rev": 2 3 3 3 2 2 3 2 3 3 ...
 $ diag1   : Factor w/ 2 levels "buff","sick": 2 2 2 2 1 1 2 1 2 1 ...
```

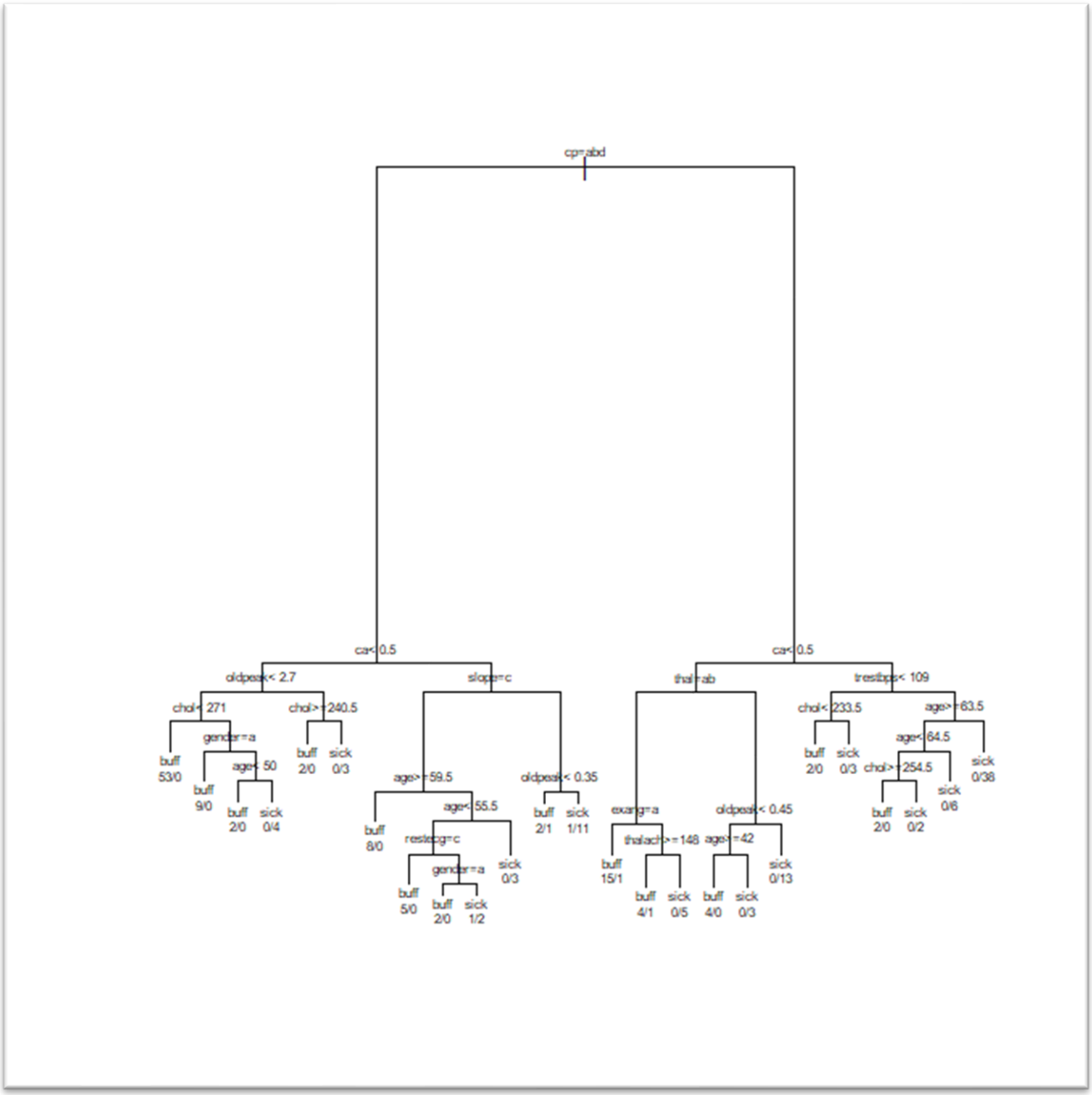
Here first show all the variables involved in the Cleveland data set and then proceed to divide it into 2 partitions of training and test data set. This gives us a general idea of what the data hold, such as it represents of is the age group this data set has, the genders respectively.

Next we shall fit the cart model.

CART MODEL

```
> ctrl <- rpart.control(minbucket = 2, minsplit = 4, xval = 10, cp = 0)
> y.cart <- rpart(diag1~., data = y_train, control = ctrl)
> x11()
> plot(y.cart)
> text(y.cart, use.n = TRUE, cex = .5)
```

The x11() renders and displays us the graph. The graph generated is as follows:



The graph gives all the weighted relation with respect to age, gender, and weather they are buff or sick.

```
> p1 <- predict(y.cart, newdata = y_test, type = "class")
> y_true <- y_test$diag1
> table(p1, y_true)
      y_true
p1     buff sick
  buff   38    8
  sick   10   32
> mean(p1 != y_true)
[1] 0.2045455
```

In the next step, we assign p1 as a predictor value, and we fit the Cart model with test diag1, where diag1 is the response variable. Then by using the mean function we calculate the accuracy of the above test set and we get an accuracy of 20.45%.

Now as we had installed the above packages, we shall first start off with the randomforest.

Randomforest:

```
> RF <- randomForest(diag1~., data = y_train, n.tree = 10000)
> p1 <- predict(RF, newdata = y_test, type = "class")
> y_true = y_test$diag1
>
> table(p1, y_true)
      y_true
p1     buff sick
  buff   43    8
  sick    5   32
> mean(p1 != y_true)
[1] 0.1477273
>
> training_nnet <- data.frame(y_train[,-14])
> testing_nnet <- data.frame(y_test[,-14])
>
> dummy <- dummyVars(" ~ .", data = training_nnet)
> training_nnet <- data.frame(predict(dummy, newdata = training_nnet))
>
> dummy <- dummyVars(" ~ .", data = testing_nnet)
> testing_nnet <- data.frame(predict(dummy, newdata = testing_nnet))
>
> training_nnet$diag1 <- ifelse(y_train$diag1=="sick",1,0)
> testing_nnet$diag1 <- ifelse(y_test$diag1=="sick",1,0)
```

Here RF variable is the random forest variable and the Randomforest function implements Breiman's random forest algorithm or classification and regression. We use randomforest as it is used build decision trees on different samples and then take its majority vote for classification and average in case of regression. Here we get the test error rate as 14.77%

```
> str(training_nnet)
'data.frame':   208 obs. of  26 variables:
 $ age          : num  63 37 41 56 62 57 57 56 44 64 ...
 $ gender.fem   : num  0 0 1 0 1 1 0 0 0 0 ...
 $ gender.male  : num  1 1 0 1 0 0 1 1 1 1 ...
 $ cp.abnang    : num  0 0 1 1 0 0 0 0 1 0 ...
 $ cp.angina    : num  1 0 0 0 0 0 0 0 0 1 ...
 $ cp.asympt    : num  0 0 0 0 1 1 1 0 0 0 ...
 $ cp.notang    : num  0 1 0 0 0 0 0 1 0 0 ...
 $ trestbps     : num  145 130 130 120 140 120 140 130 120 110 ...
 $ chol         : num  233 250 204 236 268 354 192 256 263 211 ...
 $ fbs.fal      : num  0 1 1 1 1 1 1 0 1 1 ...
 $ fbs.true     : num  1 0 0 0 0 0 0 1 0 0 ...
 $ restecg.abn  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg.hyp  : num  1 0 1 0 1 0 0 1 0 1 ...
 $ restecg.norm: num  0 1 0 1 0 1 1 0 1 0 ...
 $ thalach      : num  150 187 172 178 160 163 148 142 173 144 ...
 $ exang.fal    : num  1 1 1 1 1 0 1 0 1 0 ...
 $ exang.true   : num  0 0 0 0 0 1 0 1 0 1 ...
 $ oldpeak      : num  2.3 3.5 1.4 0.8 3.6 0.6 0.4 0.6 0 1.8 ...
 $ slope.down   : num  1 1 0 0 1 0 0 0 0 0 ...
 $ slope.flat   : num  0 0 0 0 0 0 1 1 0 1 ...
 $ slope.up     : num  0 0 1 1 0 1 0 0 1 0 ...
 $ ca           : num  0 0 0 0 2 0 0 1 0 0 ...
 $ thal.fix     : num  1 0 0 0 0 0 1 1 0 0 ...
 $ thal.norm    : num  0 1 1 1 1 1 0 0 0 1 ...
 $ thal.rev     : num  0 0 0 0 0 0 0 0 1 0 ...
 $ diag1        : num  0 0 0 0 1 0 0 1 0 0 ...
> str(testing_nnet)
'data.frame':   88 obs. of  26 variables:
 $ age          : num  67 67 63 53 56 49 58 66 60 59 ...
 $ gender.fem   : num  0 0 0 0 1 0 0 1 0 0 ...
 $ gender.male  : num  1 1 1 1 0 1 1 0 1 1 ...
 $ cp.abnang    : num  0 0 0 0 1 1 0 0 0 0 ...
 $ cp.angina    : num  0 0 0 0 0 0 0 1 0 0 ...
 $ cp.asympt    : num  1 1 1 1 0 0 0 0 1 1 ...
 $ cp.notang    : num  0 0 0 0 0 0 1 0 0 0 ...
 $ trestbps     : num  160 120 130 140 140 130 132 150 117 135 ...
 $ chol         : num  286 229 254 203 294 266 224 226 230 234 ...
 $ fbs.fal      : num  1 1 1 0 1 1 1 1 0 1 ...
 $ fbs.true     : num  0 0 0 1 0 0 0 0 1 0 ...
 $ restecg.abn  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg.hyp  : num  1 1 1 1 1 0 1 0 0 0 ...
 $ restecg.norm: num  0 0 0 0 0 1 0 1 1 1 ...
 $ thalach      : num  108 129 147 155 153 171 173 114 160 161 ...
 $ exang.fal    : num  0 0 1 0 1 1 1 1 0 1 ...
 $ exang.true   : num  1 1 0 1 0 0 0 0 1 0 ...
 $ oldpeak      : num  1.5 2.6 1.4 3.1 1.3 0.6 3.2 2.6 1.4 0.5 ...
 $ slope.down   : num  0 0 0 1 0 0 0 1 0 0 ...
 $ slope.flat   : num  1 1 1 0 1 0 0 0 0 1 ...
 $ slope.up     : num  0 0 0 0 0 1 1 0 1 0 ...
 $ ca           : num  3 2 1 0 0 0 2 0 2 0 ...
 $ thal.fix     : num  0 0 0 0 0 0 0 0 0 0 ...
 $ thal.norm    : num  1 0 0 0 1 1 0 1 0 0 ...
 $ thal.rev     : num  0 1 1 1 0 0 1 0 1 1 ...
 $ diag1        : num  1 1 1 1 0 0 1 0 1 0 ...
>
```

Next we shall look into neural network.

NEURAL NETWORK

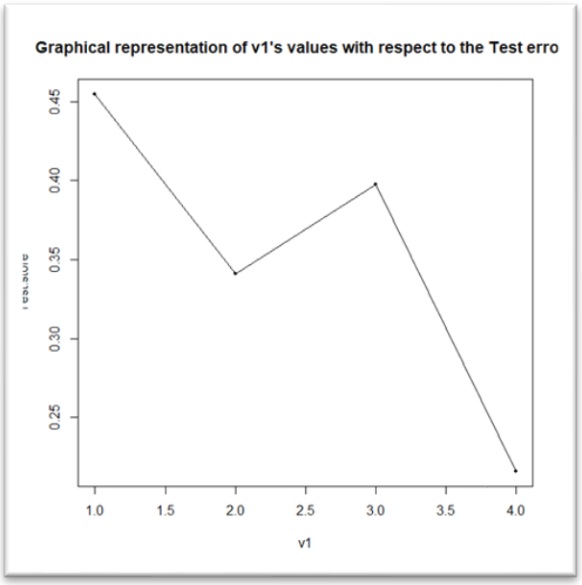
```
> #The Neural_Networks
> Test.store <- c()
> for (i in 1:4){
+   data.neuralnet <- neuralnet(diag1~., data = training_nnet, hidden = i, stepmax = 10^9, err.fct = "ce", linear.output = FALSE)
+   p1 <- predict(data.neuralnet, newdata = testing_nnet)
+   y.h <- ifelse(p1>0.5, 1, 0)
+   y.true <- testing_nnet$diag1
+   m <- mean(y.h != y.true) #where m is the error variable
+   Test.store <- c(Test.store, m)
+ }
> Test.store
[1] 0.4545455 0.3409091 0.3977273 0.2159091
```

Here we have p1 as the predictor variable upon which the predicted value from the neural net shall be stored. Then we assign y.h and y.true variables to assign parameters from the prector which will later be used to compare values from the predictor value to testing_nnet value from diag1 correspondingly. The compared values are further put into mean function which gives us the mean error values. Now we shall later storing these test errors in a temporary variable test.store which will have a set of test errors.

Now in the next screen snippet, we shall show as to how we would use the test.store's value and use min function to calculate minimum test error rate and then compare the results with the original error rates, hence subsequently enabling us to plot respective graphs to gain better understanding of the nnet algorithm.

```
> min_error_rate = min(Test.store)
> min_error_rate
[1] 0.1931818
>
> i = which(Test.store == min_error_rate)
> i
[1] 3
> v1 <- c(1:4) # where v1 is the values stored from c (i.e, from 1 to 4th position)
> x11()
> plot(v1,Test.store, type="o", pch=19, cex=0.5, main="Graphical representation of v1's values with respect to the Test errors")
```

We get a result stating that we get an approx. lowest test error value of 19.3%.



Now by comparing all the three model, i.e, the Cart model, the neural network and the random forest we gain the conclusion that the Randomforest is the best fit as it produces a test error rate of only 14.77%.

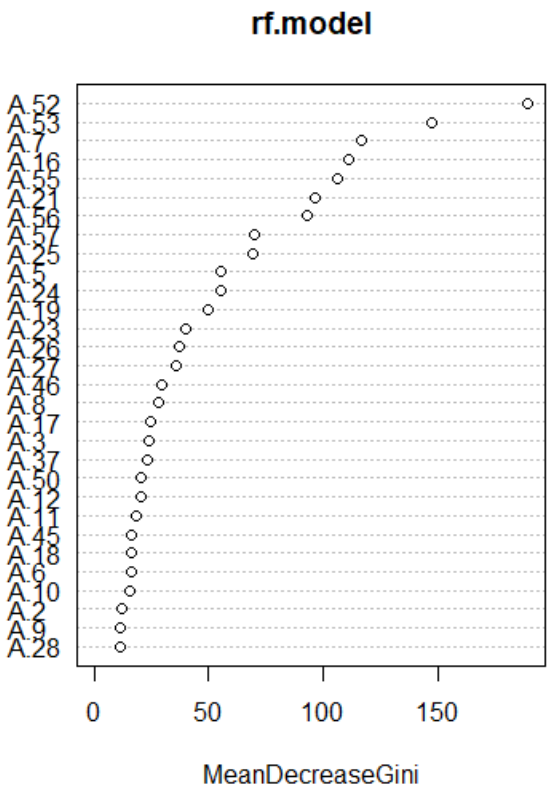
Q5) First we shall install and call the necessary packages such as glmnet, leaps, and random forest. Next, we shall use load the spam data and check the summary to see what the data set is composed of.

```
> library(glmnet)
> library(pls)
> library(leaps)
> library(randomForest)
> spam.dataset <- get(load("C:/Users/Sriram/Desktop/R projects/hw5/spam.RData"))
> summary(spam.dataset)
      A.1               A.2               A.3               A.4               A.5               A.6               A.7               A.8               A.9         
 Min.   :0.0000   Min.   : 0.000   Min.   :0.0000   Min.   : 0.00000   Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000   Min.   : 0.0000   Min.   :0.00000  
 1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.00000  
 Median :0.0000   Median : 0.000   Median :0.0000   Median : 0.00000   Median : 0.0000   Median :0.0000   Median :0.0000   Median : 0.0000   Median :0.00000  
 Mean   :0.1046   Mean   : 0.213   Mean   :0.2807   Mean   : 0.06542   Mean   : 0.3122   Mean   :0.0959   Mean   :0.1142   Mean   : 0.1053   Mean   :0.09007  
 3rd Qu.:0.0000   3rd Qu.: 0.000   3rd Qu.:0.4200   3rd Qu.: 0.00000   3rd Qu.: 0.3800   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000   3rd Qu.:0.00000  
 Max.   :4.5400   Max.   :14.280   Max.   :5.1000   Max.   :42.81000   Max.   :10.0000   Max.   :5.8800   Max.   :7.2700   Max.   :11.1100   Max.   :5.26000  
      A.10              A.11              A.12              A.13              A.14              A.15              A.16              A.17              A.18        
 Min.   : 0.0000   Min.   :0.00000   Min.   :0.0000   Min.   :0.00000   Min.   : 0.00000   Min.   :0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.: 0.0000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
 Median : 0.0000   Median :0.00000   Median :0.1000   Median :0.00000   Median : 0.00000   Median :0.0000   Median : 0.0000   Median :0.0000   Median :0.0000  
 Mean   : 0.2394   Mean   :0.05982   Mean   :0.5417   Mean   :0.09393   Mean   : 0.05863   Mean   :0.0492   Mean   : 0.2488   Mean   :0.1426   Mean   :0.1847  
 3rd Qu.: 0.1600   3rd Qu.:0.00000   3rd Qu.:0.8000   3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.1000   3rd Qu.:0.0000   3rd Qu.:0.0000  
 Max.   :18.1800   Max.   :2.61000   Max.   :9.6700   Max.   :5.55000   Max.   :10.00000   Max.   :4.4100   Max.   :20.0000   Max.   :7.1400   Max.   :9.0900  
      A.19              A.20              A.21              A.22              A.23              A.24              A.25              A.26              A.27        
 Min.   : 0.000   Min.   : 0.00000   Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.00000   Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000  
 1st Qu.: 0.000   1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000  
 Median : 1.310   Median : 0.00000   Median : 0.2200   Median : 0.0000   Median :0.0000   Median : 0.00000   Median : 0.0000   Median : 0.0000   Median : 0.0000  
 Mean   : 1.662   Mean   : 0.08558   Mean   : 0.8098   Mean   : 0.1212   Mean   :0.1016   Mean   : 0.09427   Mean   : 0.5495   Mean   : 0.2654   Mean   : 0.7673  
 3rd Qu.: 2.640   3rd Qu.: 0.00000   3rd Qu.: 1.2700   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000  
 Max.   :18.750   Max.   :18.18000   Max.   :11.1100   Max.   :17.1000   Max.   :5.4500   Max.   :12.50000   Max.   :20.8300   Max.   :16.6600   Max.   :33.3300  
      A.28              A.29              A.30              A.31              A.32              A.33              A.34              A.35              A.36        
 Min.   :0.0000   Min.   : 0.00000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000   Min.   : 0.00000   Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000  
 1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000  
 Median :0.0000   Median : 0.00000   Median :0.0000   Median : 0.00000   Median :0.00000   Median : 0.00000   Median :0.00000   Median : 0.0000   Median :0.00000  
 Mean   :0.1248   Mean   : 0.09892   Mean   :0.1029   Mean   : 0.06475   Mean   :0.04705   Mean   : 0.09723   Mean   :0.04784   Mean   : 0.1054   Mean   :0.09748  
 3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.00000  
 Max.   :9.0900   Max.   :14.28000   Max.   :5.8800   Max.   :12.50000   Max.   :4.76000   Max.   :18.18000   Max.   :4.76000   Max.   :20.0000   Max.   :7.69000  
      A.37              A.38              A.39              A.40              A.41              A.42              A.43              A.44              A.45        
 Min.   :0.000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000   Min.   :0.00000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.0000   Min.   : 0.0000  
 1st Qu.:0.000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000  
 Median :0.000   Median :0.0000   Median : 0.00000   Median :0.00000   Median :0.00000   Median : 0.0000   Median :0.0000   Median : 0.0000   Median : 0.0000  
 Mean   :0.137   Mean   :0.0132   Mean   : 0.07863   Mean   :0.06483   Mean   :0.04367   Mean   : 0.1323   Mean   :0.0461   Mean   : 0.0792   Mean   : 0.3012  
 3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.1100  
 Max.   :6.890   Max.   :8.3300   Max.   :11.11000   Max.   :4.76000   Max.   :7.14000   Max.   :14.2800   Max.   :3.5700   Max.   :20.0000   Max.   :21.4200  
      A.46              A.47              A.48              A.49              A.50              A.51              A.52              A.53              A.54        
 Min.   : 0.0000   Min.   :0.000000   Min.   : 0.00000   Min.   :0.00000   Min.   :0.000   Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000   Min.   : 0.00000  
 1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000   1st Qu.: 0.00000  
 Median : 0.0000   Median :0.000000   Median : 0.00000   Median :0.00000   Median :0.065   Median :0.00000   Median : 0.0000   Median :0.00000   Median : 0.00000  
 Mean   : 0.1798   Mean   :0.005444   Mean   : 0.03187   Mean   :0.03857   Mean   :0.139   Mean   :0.01698   Mean   : 0.2691   Mean   :0.07581   Mean   : 0.04424  
 3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.:0.188   3rd Qu.:0.00000   3rd Qu.: 0.3150   3rd Qu.:0.05200   3rd Qu.: 0.00000  
 Max.   :22.0500   Max.   :2.170000   Max.   :10.00000   Max.   :4.38500   Max.   :9.752   Max.   :4.08100   Max.   :32.4780   Max.   :6.00300   Max.   :19.82900  
      A.55              A.56              A.57            spam        
 Min.   :   1.000   Min.   :   1.00   Min.   :   1.0   email:2788  
 1st Qu.:   1.588   1st Qu.:   6.00   1st Qu.:  35.0   spam :1813  
 Median :   2.276   Median :  15.00   Median :  95.0               
 Mean   :   5.191   Mean   :  52.17   Mean   : 283.3               
 3rd Qu.:   3.706   3rd Qu.:  43.00   3rd Qu.: 266.0               
 Max.   :1102.500   Max.   :9989.00   Max.   :15841.0              
> str(spam.dataset)
'data.frame':	4601 obs. of  58 variables:
 $ A.1 : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
 $ A.2 : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
 $ A.3 : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
 $ A.4 : num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.5 : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
 $ A.6 : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
 $ A.7 : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
 $ A.8 : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
 $ A.9 : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
 $ A.10: num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
 $ A.11: num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
 $ A.12: num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
 $ A.13: num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
 $ A.14: num  0 0.21 0 0 0 0 0 0 0 0 ...
 $ A.15: num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
 $ A.16: num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
 $ A.17: num  0 0.07 0.06 0 0 0 0 0 0 0 ...
 $ A.18: num  1.29 0.28 1.03 0 0 0 0 0.32 0 0.15 0.12 ...
 $ A.19: num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
 $ A.20: num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
 $ A.21: num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
 $ A.22: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.23: num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
 $ A.24: num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
 $ A.25: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.26: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.27: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.28: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.29: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.30: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.31: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.32: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.33: num  0 0 0 0 0 0 0 0 0.15 0 ...
 $ A.34: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.35: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.36: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.37: num  0 0.07 0 0 0 0 0 0 0 0 ...
 $ A.38: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.39: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.40: num  0 0 0.06 0 0 0 0 0 0 0 ...
 $ A.41: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.42: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.43: num  0 0 0.12 0 0 0 0 0 0.3 0 ...
 $ A.44: num  0 0 0 0 0 0 0 0 0 0.06 ...
 $ A.45: num  0 0 0.06 0 0 0 0 0 0 0 ...
 $ A.46: num  0 0 0.06 0 0 0 0 0 0 0 ...
 $ A.47: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.48: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.49: num  0 0 0.01 0 0 0 0 0 0 0.04 ...
 $ A.50: num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.271 0.03 ...
 $ A.51: num  0 0 0 0 0 0 0 0 0 0 ...
 $ A.52: num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0.244 ...
 $ A.53: num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
 $ A.54: num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
```

Here we divide the spam data set into 2 parts of test and training data set and store in its respective variables. Then we call the randomforest function to use the randomforest classifier with m value, i.e., the number of randomly selected inputs for each tree as 5. The resultant confusion matrix is as follows.

```
> x.train.set <- sample(1:nrow(spam.dataset), nrow(spam.dataset)*0.80)
> x.test.set <- -x.train.set
> training.data <- spam.dataset[x.train.set, ]
> testing.data.set <- spam.dataset[x.test.set, ]
>
> rf.model <- randomForest(spam~., data=training.data,mtry=5,ntree=2500)
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 5,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 4.46%
Confusion matrix:
      email spam class.error
email  2183   58  0.02588130
spam    106 1333  0.07366227
```

Followed by the above code, we use varImpPlot () which gives us the Dotchart of variable importance that is measured by a Random Forest classifier and importance()extractor function where a variable importance is measured as produced by randomForest. The plot generated varIMP() is as follows:




```
> varImpPlot(rf.model)
> importance(rf.model)
      MeanDecreaseGini
A.1           7.4241505
A.2          12.3080758
A.3          23.7801360
A.4           1.7275054
A.5          55.4881815
A.6          15.9392614
A.7         117.1472591
A.8          27.9569410
A.9          11.3704772
A.10         15.6114807
A.11         18.4390557
A.12         20.4026109
A.13          6.8810114
A.14          4.3541469
A.15          3.4288645
A.16        111.3810024
A.17         24.7876641
A.18         15.9886831
A.19         50.0617205
A.20         10.5185793
A.21         96.6570402
A.22          5.3502182
A.23         40.3236312
A.24         55.0995655
A.25         69.5917941
A.26         36.8739310
A.27         36.1444731
A.28         11.2339585
A.29          4.9251465
A.30          7.7964524
A.31          3.6264859
A.32          1.2403962
A.33          5.5930697
A.34          1.4424825
A.35          7.2189823
A.36          6.0683606
A.37         23.2955368
A.38          0.6067407
A.39          6.2420190
A.40          2.4574650
A.41          1.6910724
A.42         10.2369619
A.43          3.1364025
A.44          3.4780366
A.45         16.0005460
A.46         29.7527917
A.47          0.4529441
A.48          2.1007280
A.49          9.1689032
A.50         20.7416287
A.51          4.9371289
A.52        189.6201990
A.53        147.3754806
A.54          8.6310887
```

Here we get the predictors which we later use to predict the values by using predict function and by fitting randomforest classifier to testing data set and store these values in ran_pred variable. Then we compare these predicted values with the spam's testing data set and generate the mean of the compared values. In this scenario we obtain 5.53% as mean value which is nothing but the testing error accuracy percentage.

```
> y.true <- as.numeric(testing.data.set$spam)-1
> y.h <- predict(rf.model, newdata = testing.data.set, type = "response")
> y.h <- as.numeric(y.h)-1
> misclass_rf <- sum(abs(y.true- y.h))/length(y.h)
> misclass_rf
[1] 0.05537459
>
>
> ran_pred <- predict(rf.model, testing.data.set, type = "class")
> mean(ran_pred != testing.data.set$spam)
[1] 0.05537459
```

Comparing generic Rf model and Rf model with value as 10

```
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 5,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 4.46%
Confusion matrix:
      email spam class.error
email  2183   58  0.02588130
spam    106 1333  0.07366227
> #Using 10 inputs:
> rf.model <- randomForest(spam~., data=training.data,mtry=10,ntree=2500)
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 10,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 10

        OOB estimate of  error rate: 4.46%
Confusion matrix:
      email spam class.error
email  2179   62  0.02766622
spam    102 1337  0.07088256
>
> ran_pred <- predict(rf.model, testing.data.set, type = "class")
> mean(ran_pred != testing.data.set$spam)
[1] 0.06297503
>
>
```

In Rfmodel with 10 inputs we get test error accuracy as 6.29%, and the confusion matrix for its respective m value is shown as above.

For Rf model with input 20 we get test error accuracy of 6.18% and its corresponding confusion matrix is shown below.

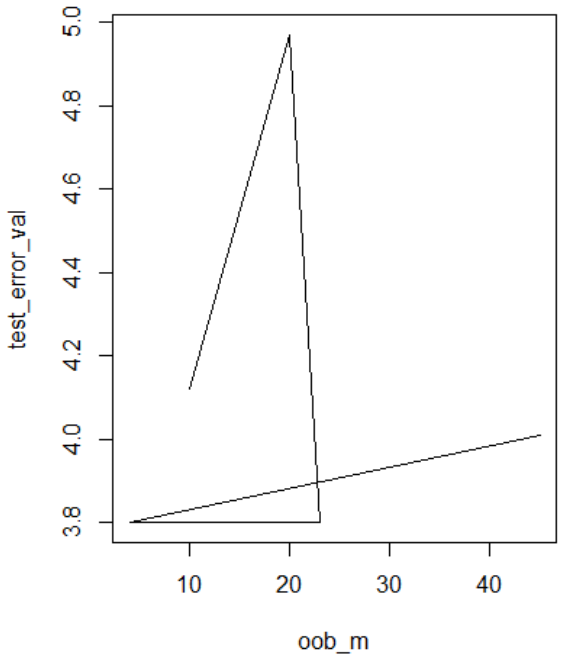
```
> #Using 20 inputs:
> rf.model <- randomForest(spam~., data=training.data,mtry=20,ntree=2500)
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 20,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 20

        OOB estimate of  error rate: 4.78%
Confusion matrix:
      email spam class.error
email  2168   73  0.03257474
spam    103 1336  0.07157748
>
> ran_pred <- predict(rf.model, testing.data.set, type = "class")
> mean(ran_pred != testing.data.set$spam)
[1] 0.06188925
```

Similarly we apply for input values of 23,4,45 and we test error accuracy of 6.4%, 5.3% and 6.9 % respectively.

```
> #Using 23 inputs:
> rf.model <- randomForest(spam~., data=training.data,mtry=23,ntree=2500)
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 23,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 23

        OOB estimate of  error rate: 4.84%
Confusion matrix:
      email spam class.error
email  2168   73  0.03257474
spam    105 1334  0.07296734
>
> ran_pred <- predict(rf.model, testing.data.set, type = "class")
> mean(ran_pred != testing.data.set$spam)
[1] 0.0640608
>
> #Using 4 inputs:
> rf.model <- randomForest(spam~., data=training.data,mtry=4,ntree=2500)
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 4,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 4.48%
Confusion matrix:
      email spam class.error
email  2185   56  0.02498884
spam    109 1330  0.07574705
>
> ran_pred <- predict(rf.model, testing.data.set, type = "class")
> mean(ran_pred != testing.data.set$spam)
[1] 0.05320304
>
> #using 45:
> rf.model <- randomForest(spam~., data=training.data,mtry=45,ntree=2500)
> rf.model

Call:
 randomForest(formula = spam ~ ., data = training.data, mtry = 45,      ntree = 2500)
               Type of random forest: classification
                     Number of trees: 2500
No. of variables tried at each split: 45

        OOB estimate of  error rate: 5.19%
Confusion matrix:
      email spam class.error
email  2161   80  0.03569835
spam    111 1328  0.07713690
>
> ran_pred <- predict(rf.model, testing.data.set, type = "class")

> mean(ran_pred != testing.data.set$spam)
[1] 0.06948969
```

Finally we Plot both the OOB error as well as the test error against a suitably chosen range of values for m.

```
> oob_m <- c(10,20,23,4,45)
> oob_val <- c(4.86,4.97,5.17,5.08,5.3)
> test_error_val <- c(4.12,4.97,3.80,3.80,4.01)
>
> plot(oob_m, oob_val, type="l")
> plot(oob_m,test_error_val, type = "l")
```




```
> x11()
> plot(oob_m, oob_val, type="o", col="blue", ylim=c(3.5,6), ylab = "Error", xlab = "M values")
> points(oob_m, test_error_val, col="orange")
> lines(oob_m, test_error_val, col="orange")
> legend(x="topleft",legend=c("OOB","Test"),col=c("blue","orange"), lty=1)
```