Q1)Visualizing the data

```
> sum(is.na(pendigits_train))
[1] 0
>
```

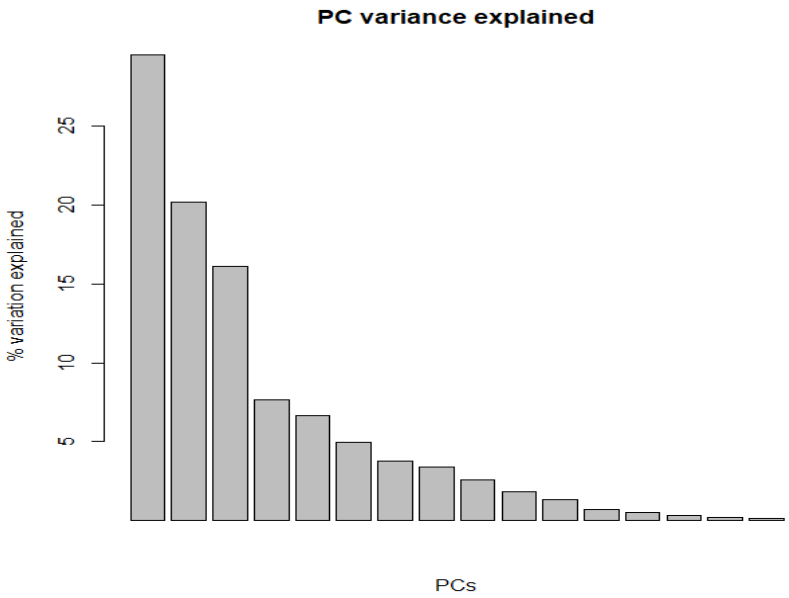
No missing data

```
> is.null(pendigits_train)
[1] FALSE
>
```

Part a)

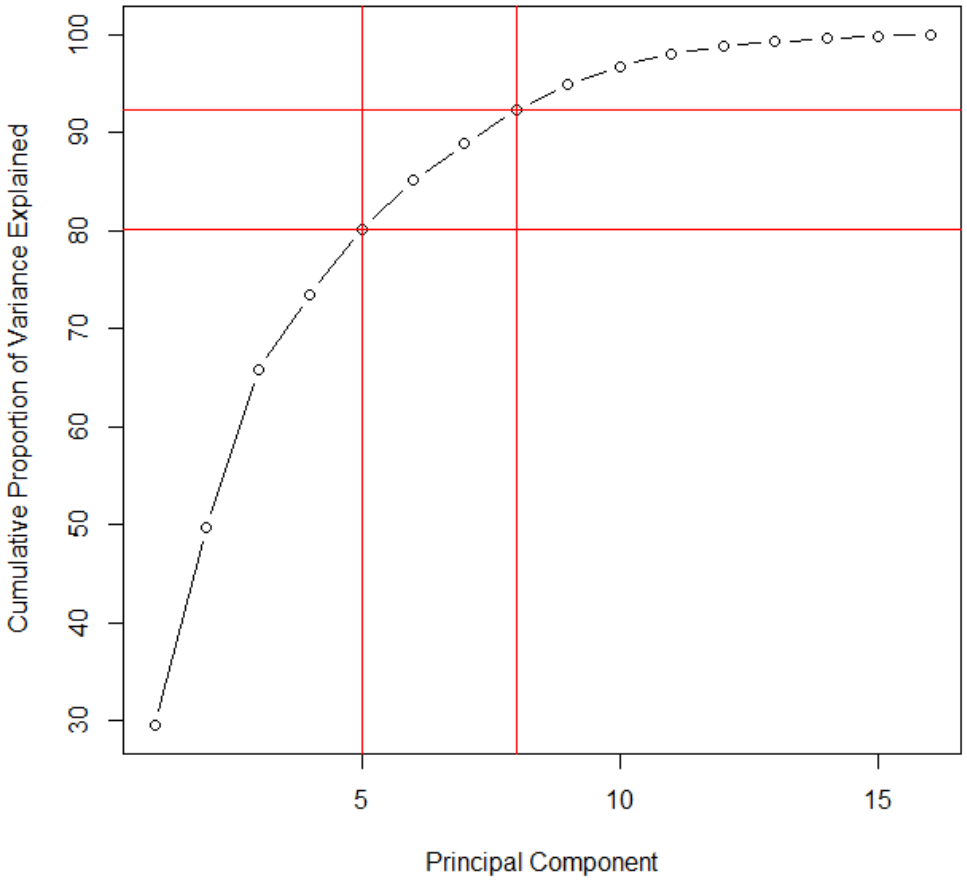
Percentage variances are shown below:

```
> percentage_var_exp
 [1] 29.4786648 20.1819506 16.1050394  7.6858863
 [5]  6.6443705  4.9833709  3.7915727  3.4265723
 [9]  2.5676505  1.8639429  1.3157041  0.7018557
[13]  0.5030066  0.3625133  0.2141531  0.1737463
>
```

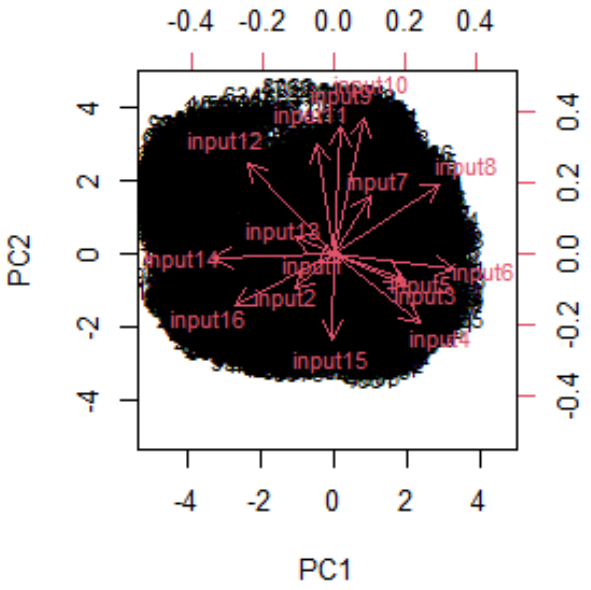


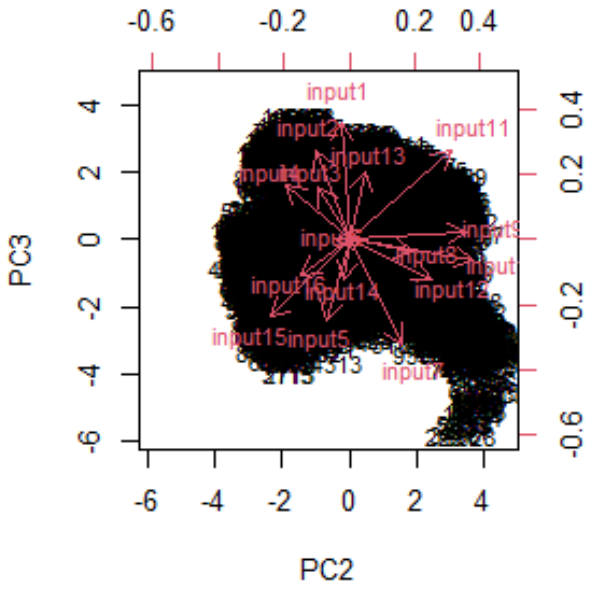
On adding up the variances explained above from PC1 till PC 5 we get a variation of 80%of the data and on further summing up till PC 8 we obtain a variation of 90% of the data. Also, this mathematical conclusion of above variations is supported by the data obtained under the **cumulative proportion** from the summary of prcomp fit which we see below.

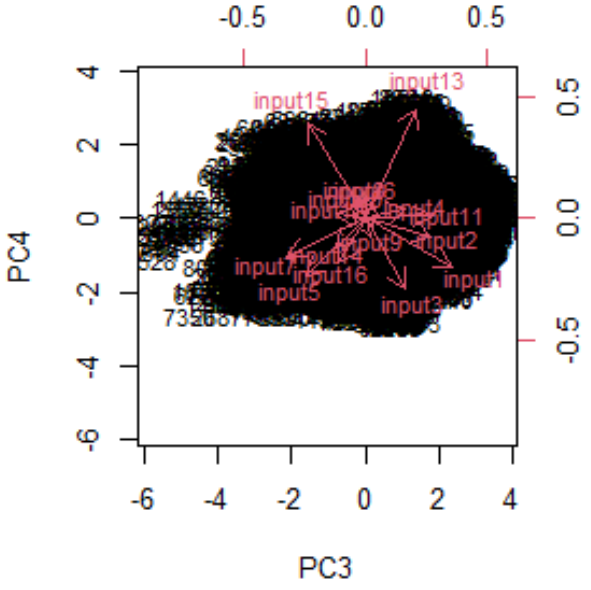
```
> summary(pendigits_prcomp)
Importance of components:
                          PC1    PC2    PC3
Standard deviation     2.1718 1.7970 1.6052
Proportion of Variance 0.2948 0.2018 0.1610
Cumulative Proportion  0.2948 0.4966 0.6577
                          PC4     PC5     PC6
Standard deviation     1.10894 1.03107 0.89294
Proportion of Variance 0.07686 0.06644 0.04983
Cumulative Proportion  0.73452 0.80096 0.85079
                          PC7     PC8     PC9
Standard deviation     0.77888 0.74044 0.64096
Proportion of Variance 0.03792 0.03427 0.02568
Cumulative Proportion  0.88871 0.922297 0.94865
                         PC10    PC11    PC12
Standard deviation     0.54611 0.45882 0.33511
Proportion of Variance 0.01864 0.01316 0.00702
Cumulative Proportion  0.96729 0.98045 0.98747
                         PC13    PC14    PC15
Standard deviation     0.28369 0.24084 0.18511
Proportion of Variance 0.00503 0.00363 0.00214
Cumulative Proportion  0.99250 0.99612 0.99826
```

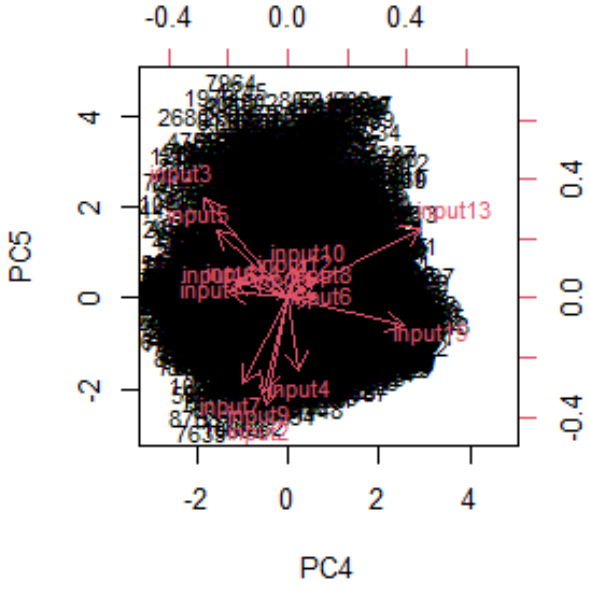


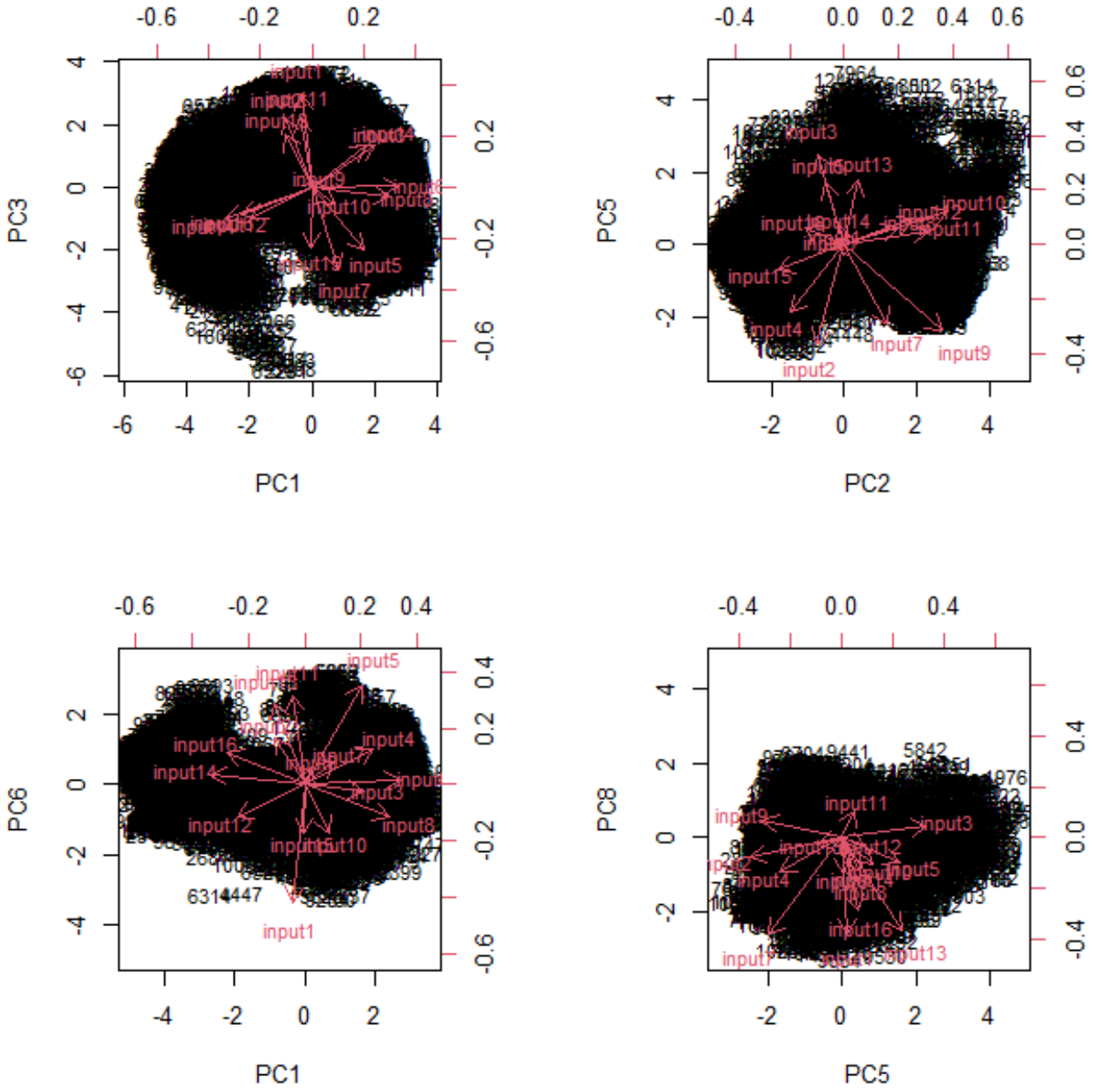
Biplots of PCs:


-0.4 -0.2 0.0 0.2 0.4
input10
input9
input11
input12
input8
input7
input13
input14
input1
input6
input3
input2
input16
input4
input15
-0.4 -0.2 0.0 0.2 0.4
-4 -2 0 2 4
PC2
PC1

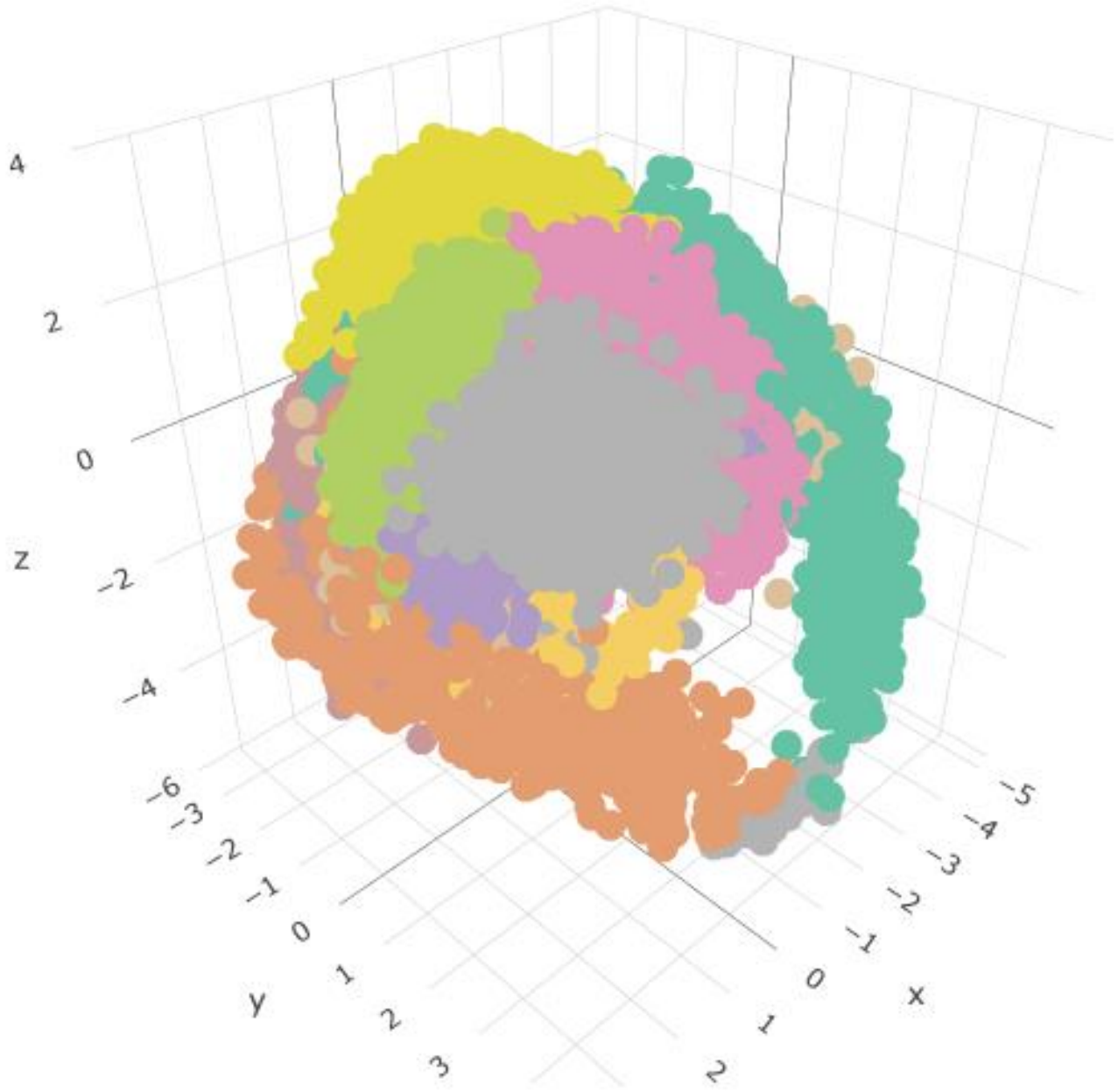

-0.6 -0.2 0.2 0.4
input1
input2
input11
input13
input3
input14
input12
input15
input5
-0.6 -0.2 0.2 0.4
-6 -4 -2 0 2 4
PC3
PC2


-0.5 0.0 0.5
input15
input13
input11
input2
input7
input16
input5
input3
-0.5 0.0 0.5
-6 -4 -2 0 2 4
PC4
PC3


-0.4 0.0 0.4
input3
input13
input10
input6
input4
-0.4 0.0 0.4
-2 0 2 4
PC5
PC4



3d score plot of first three principal components:



Partb)Fitting KNN with raw pendigits train data and predicting on test data:

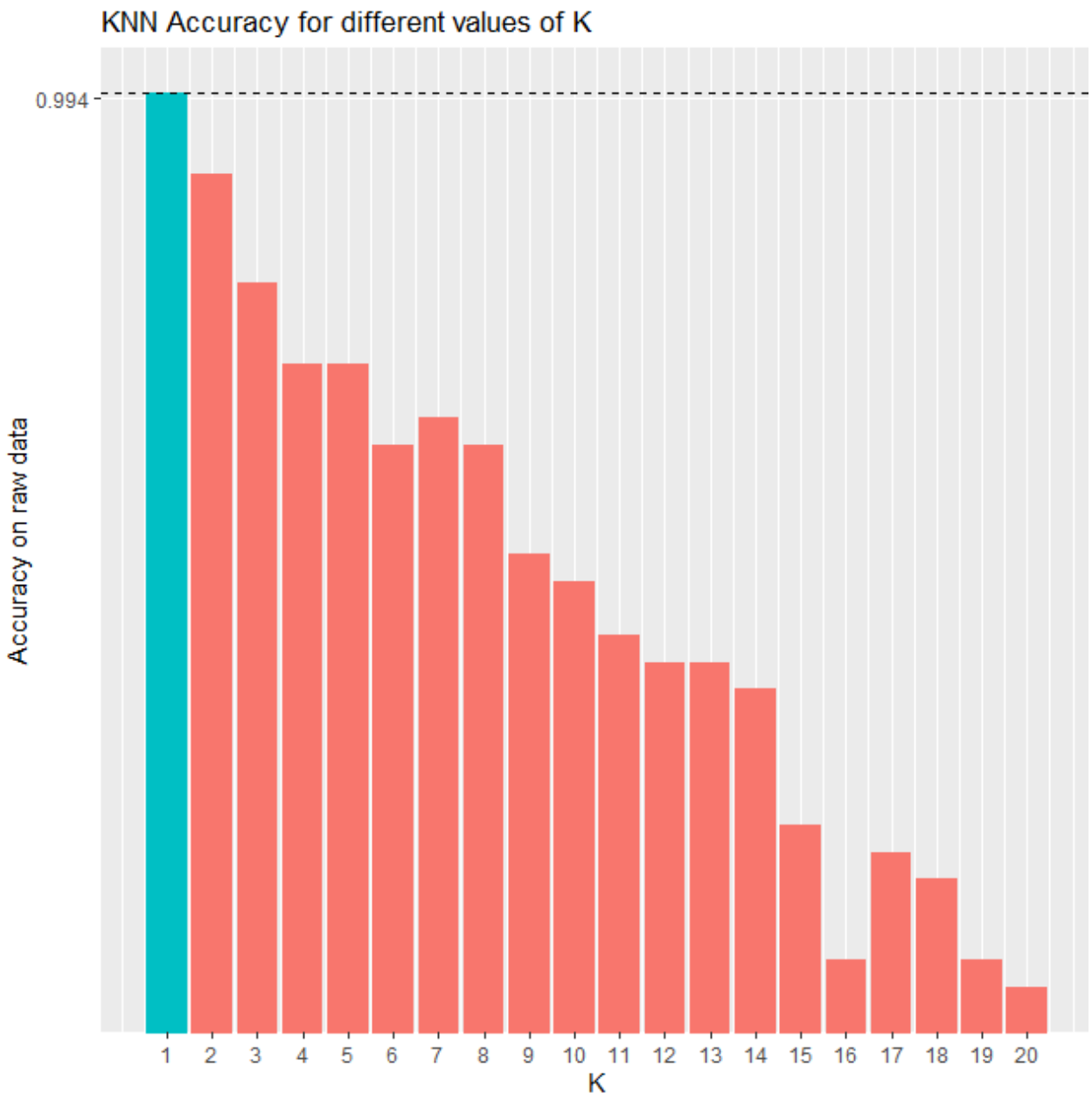
Confusion matrix:

```
> table(predicted, pendigits_test_class)
         pendigits_test_class
predicted   0   1   2   3   4
        0 232   0   0   0   0
        1   0 224   1   1   0
        2   0   1 237   0   0
        3   0   1   1 205   0
        4   1   0   0   0 210
        5   0   0   0   0   1
        6   0   0   0   0   0
        7   0   0   0   0   0
        8   0   0   0   0   0
        9   0   0   0   1   0
         pendigits_test_class
predicted   5   6   7   8   9
        0   0   0   0   0   0
        1   0   0   1   0   0
        2   0   0   1   0   0
        3   0   0   0   0   0
        4   0   0   0   0   1
        5 226   0   0   0   1
        6   0 197   0   0   0
        7   0   0 217   0   0
        8   1   0   0 225   0
        9   0   0   0   0 213
```

```
> mean(predicted == pendigits_test_class)
[1] 0.9940882
> 
```

We are getting an accuracy of 99percent with KNN over test data for k=1

Following is the list of accuracies we obtained for different K values on raw data.



With KNN we are able to get a great accuracy of 0.994 over test data for k=1 .

Now I would be fitting KNN on principal components obtained from part A. I took the first five principal components which explains 80% of the variation of data and tried to fit KNN on these principal components and following is my observation

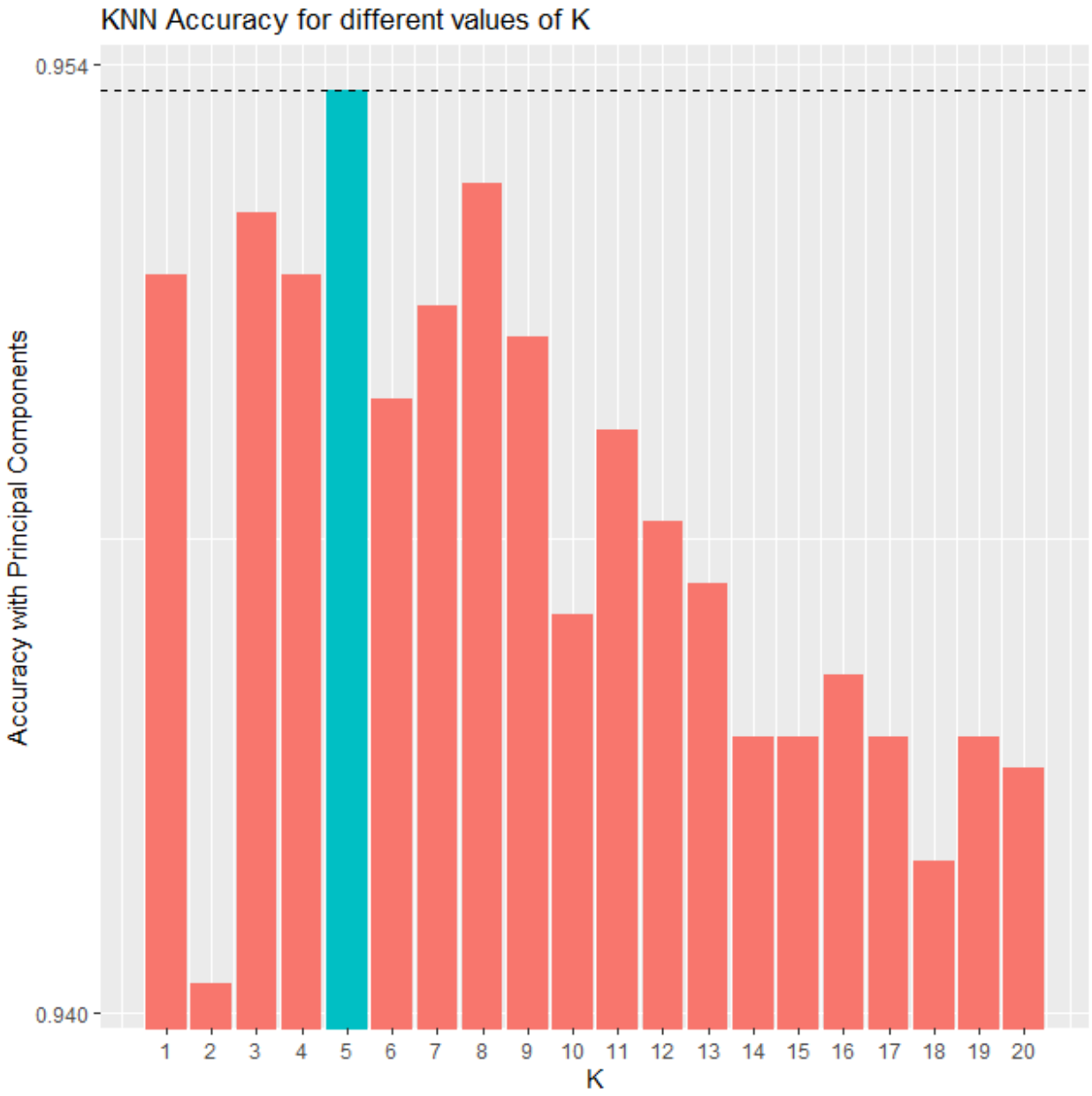
Confusion Matrix :

```
> table(predicted, pendigits_test_class)
         pendigits_test_class
predicted   0   1   2   3   4
        0 232   0   0   0   0
        1   0 188  27   1   1
        2   0  28 210   0   0
        3   0   5   0 206   0
        4   1   0   0   0 207
        5   0   0   0   0   2
        6   0   0   0   0   1
        7   0   5   2   0   0
        8   0   0   0   0   0
        9   0   0   0   0   0
         pendigits_test_class
predicted   5   6   7   8   9
        0   0   0   0   0   0
        1   0   1   3   3   0
        2   0   0   3   0   0
        3   0   0   0   0   3
        4   0   0   0   0   1
        5 221   0   0   0   9
        6   0 196   0   0   0
        7   0   0 212   2   2
        8   3   0   1 219   0
```

Accuracy:

```
> mean(predicted == pendigits_test_class)
[1] 0.9508868
```

With KNN we are able to get a great accuracy of 94..08% over test data with principal components for k=1 .Following is the list of accuracies we obtained for different K values with principal components.

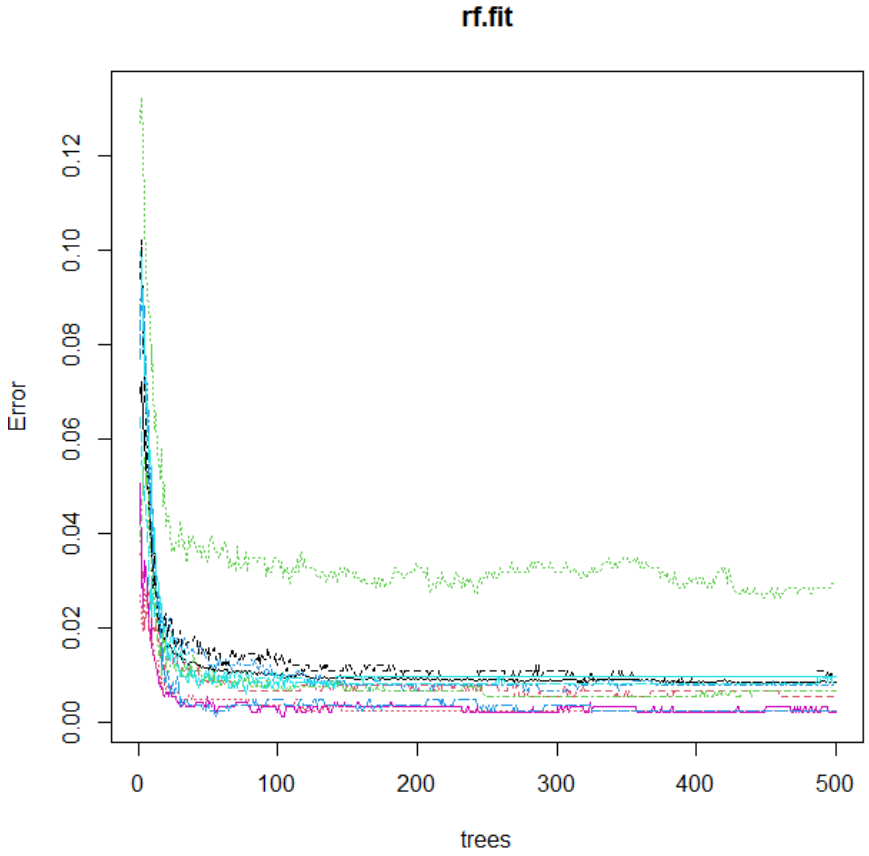



```
> pendigits_trainFinal = as.matrix(pendigits_train)%*%(rotate)
> pendigits_test_Final = as.matrix(pendigits_test) %*% (rotate)
> predicted = knn(pendigits_trainFinal, pendigits_test_Final, pendigits_train_class, k = 5)
> mean(predicted == pendigits_test_class)
[1] 0.952251
>
```

Thus with principal components with k=5 we are getting an accuracy of 95.4% which is good in the sense that when compared to KNN fit on raw data though the accuracy is high but that is for K=1 but with k=5 we would be able to generalize the model well with less complexity.

c)Fitting Pendigits Data with Random forest on raw data

```
> miss_rf
[1] 0.0004547522
> 1-miss_rf
[1] 0.9995452
>
```

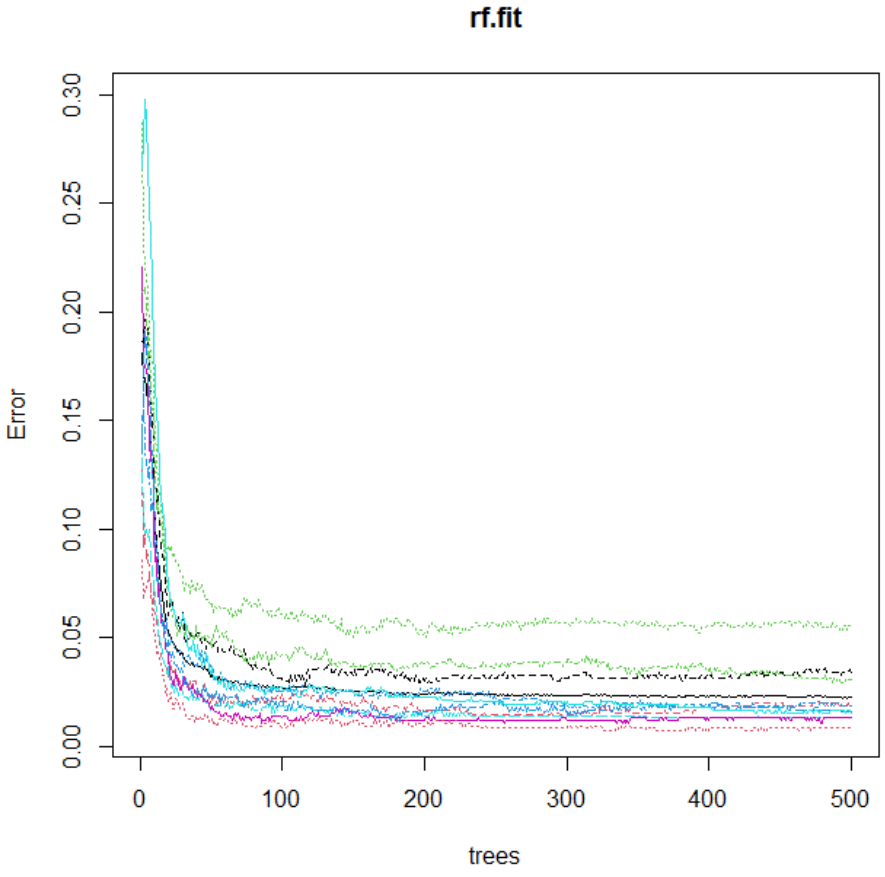


We are getting an accuracy of 99.9% with Random Forest Model over test data

)Fitting Pendigits Data with RandomForest have predictors as Principal Componenets

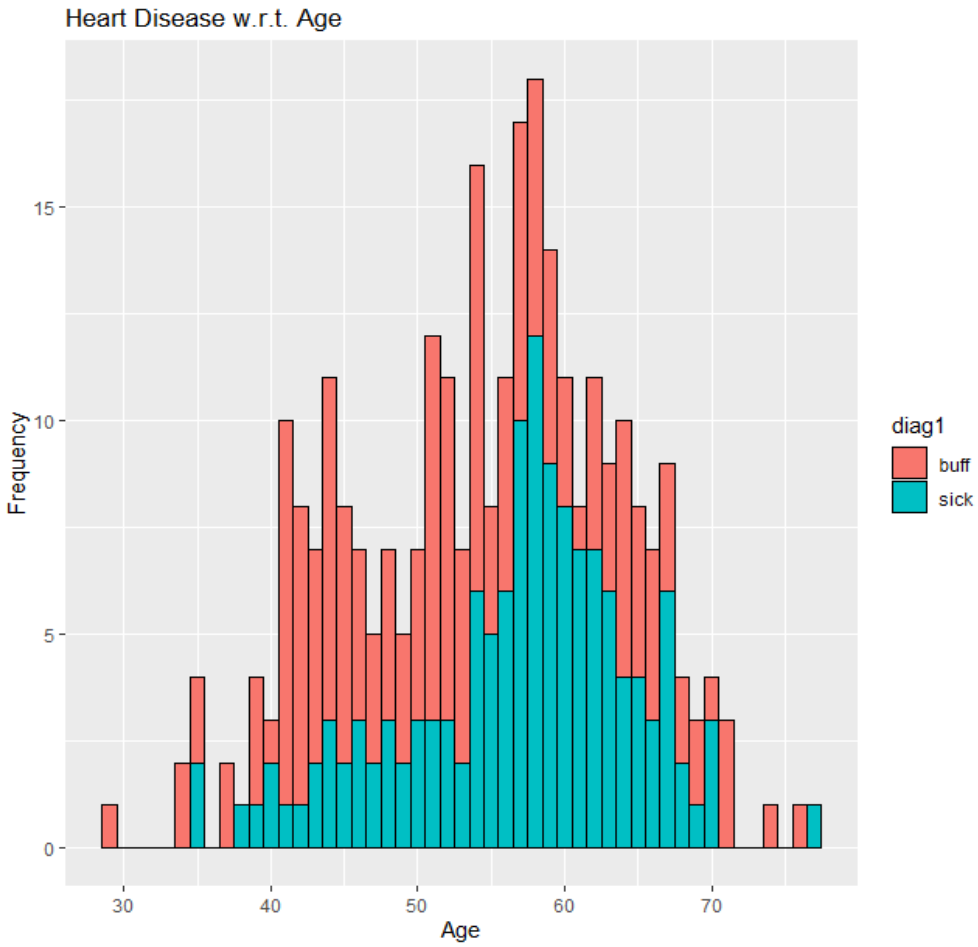
```
> length(y_true)
[1] 2199
> length(y_hat)
[1] 2199
> x<- which(y_hat!=y_true)
>
> miss_rf<-length(x)/length(y_true)
> miss_rf
[1] 0.05138699
> 1-miss_rf
[1] 0.948613
>
```

After reducing the dimensionality from 16 variables to 5 principal components we are getting a good accuracy of 94 percent with random forest



On comparing random forest and KNN, it seems both of have are at par both are having good performances and can act as good fit models for pen digits data .Also introducing principal component analysis would for both of them shows a good amount of accuracy with reduced dimensionality thus providing us a less complex model.

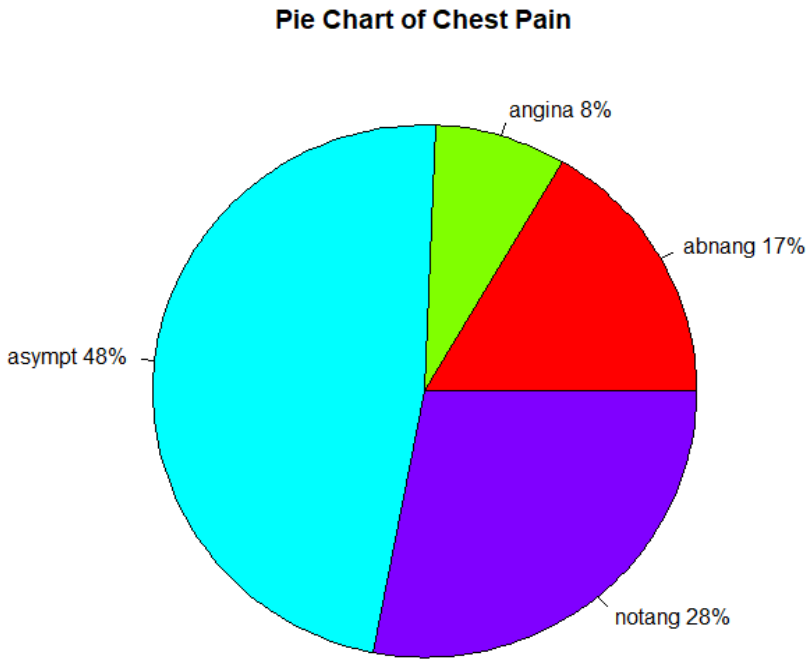
Task2:Graphical Exploration:



We can conclude that the age group of 40 to 60 has the highest probability of getting heart diseases compared to age above 60.

```
> sum(is.na(cleveland))
[1] 0
>
```

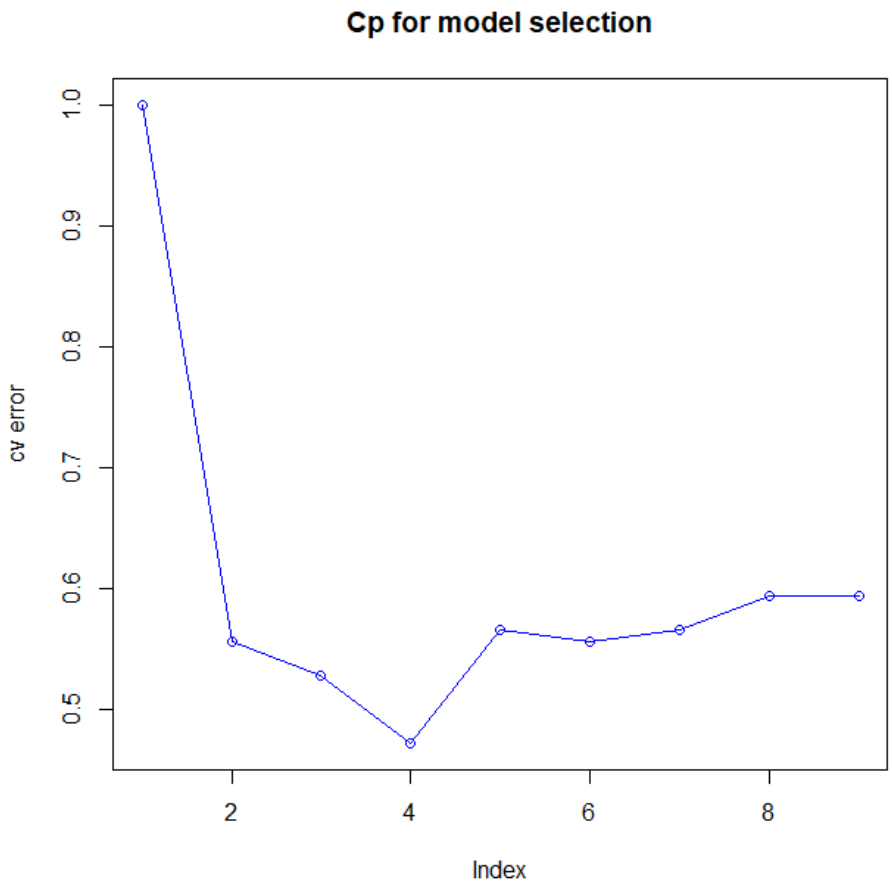
```
> is.null(cleveland)
[1] FALSE
>
```



we can conclude that out of all types of chest pain, most observed in the individual are aymptomatic type of chest pain, then comes the non-anginal.

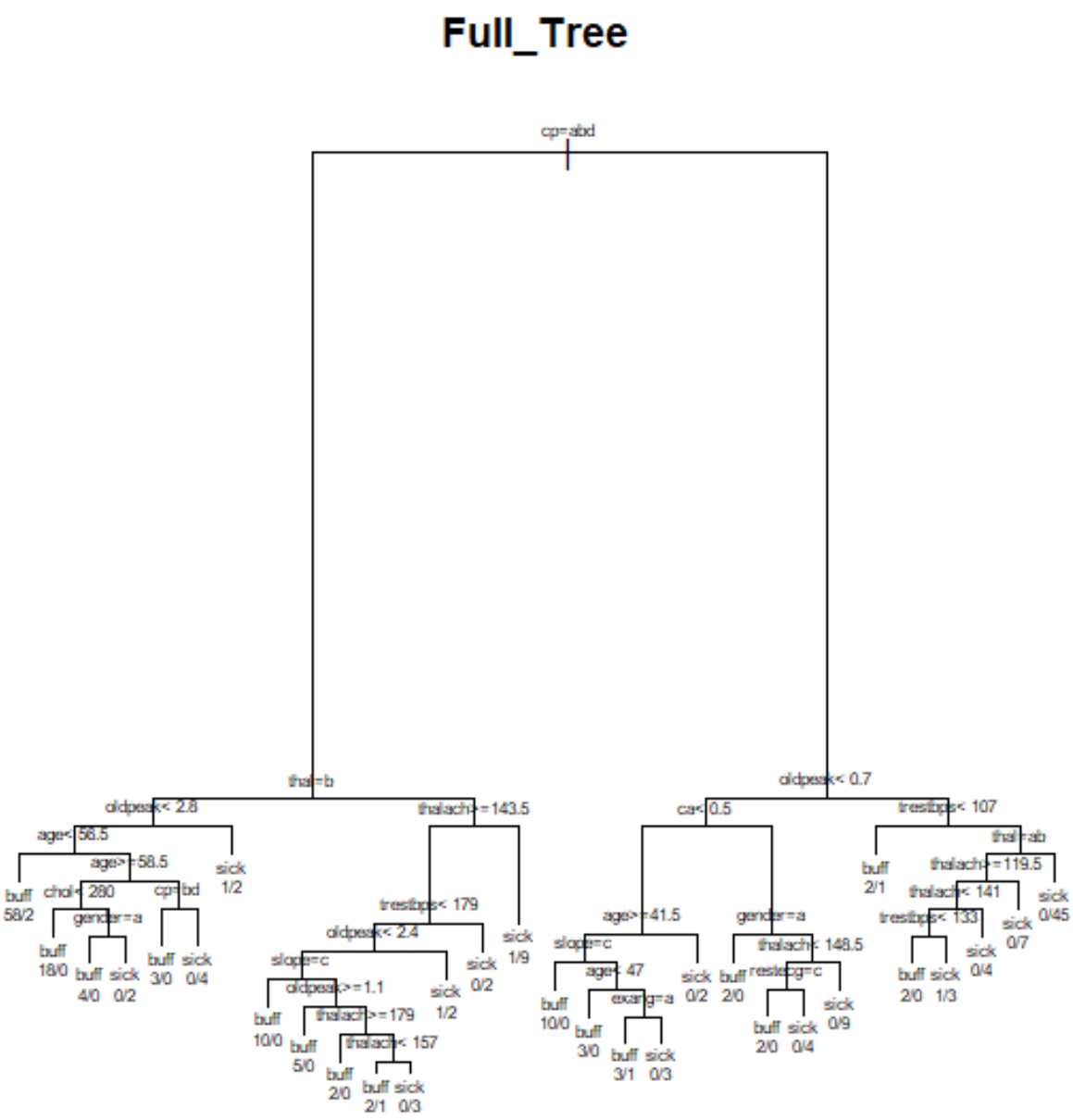
```
> fit_cleveland$cptable
           CP nsplit  rel error    xerror       xstd
1 0.509433962      0 1.00000000 1.0000000 0.07208796
2 0.047169811      1 0.49056604 0.5566038 0.06275535
3 0.037735849      3 0.39622642 0.5283019 0.06165501
4 0.018867925      5 0.32075472 0.4716981 0.05922153
5 0.011792453      8 0.26415094 0.5660377 0.06310592
6 0.009433962     12 0.21698113 0.5566038 0.06275535
7 0.006289308     21 0.13207547 0.5660377 0.06310592
8 0.004716981     25 0.10377358 0.5943396 0.06411084
9 0.000000000     29 0.08490566 0.5943396 0.06411084
> 
```

The CP table as well as the CP table plot below indicates that we can reduce the actual tree model complexity of nsplits=5 where the cross validation error is minimum.



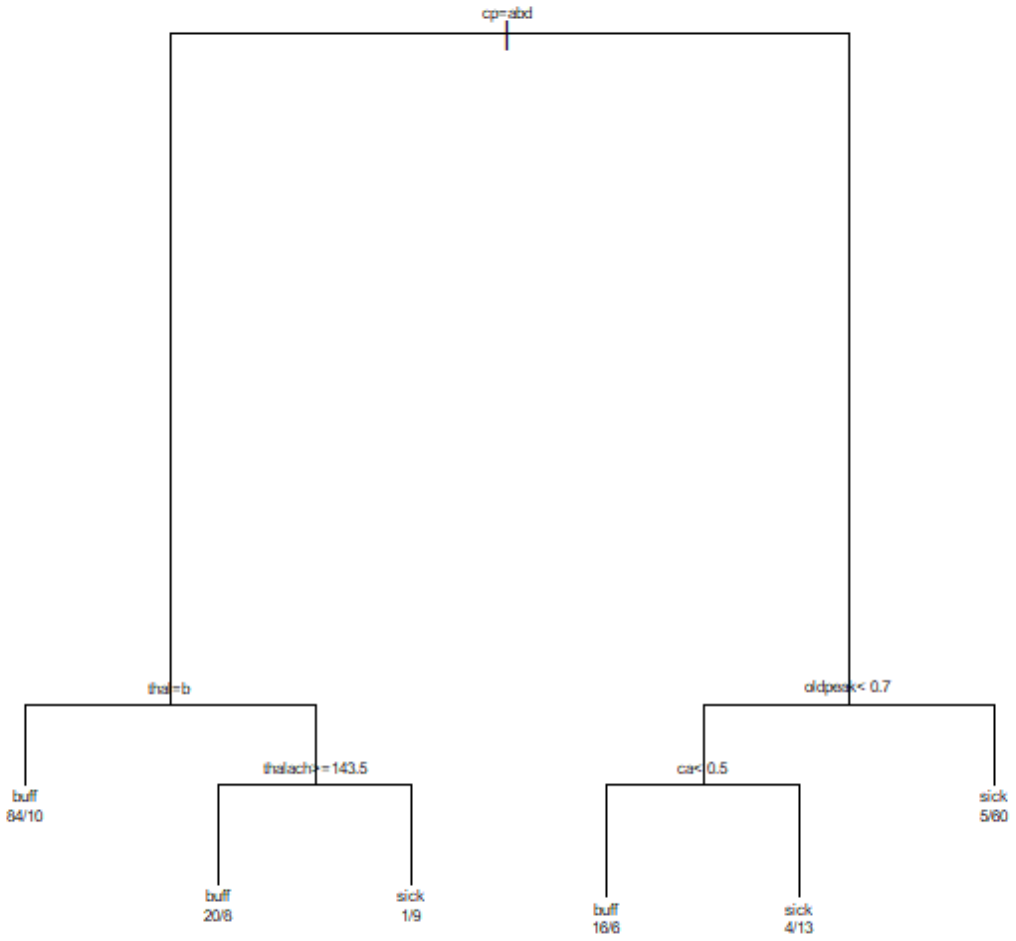
The above graph is a representation of the cp table which provides the cross-validation information of the fitted tree model on Boston Train Dataset, and the information that we get from the above graph is that we have a minimum cross validation error at index 4 which corresponds to the 5^{th} split in the cp table hence we can prune the model for a overall size of 5 splits.

Full Tree Model



Pruned Tree Model :

Pruned_Tree




```
> x<- which(predicted[,3]!=y_true_test)
> length(x)
[1] 13
> error<-length(x)/length(predicted[,3])
> predicted[,3]
 [1] "buff" "buff" "sick" "buff" "buff" "sick" "buff" "buff" "buff" "buff" "sick" "sick" "sick" "buff" "buff"
[16] "sick" "buff" "sick" "sick" "buff" "sick" "sick" "buff" "sick" "buff" "sick" "sick" "sick" "sick" "sick"
[31] "sick" "buff" "buff" "buff" "buff" "sick" "buff" "sick" "sick" "sick" "sick" "buff" "buff" "buff" "sick"
[46] "sick" "sick" "sick" "buff" "buff" "buff" "buff" "sick" "sick" "buff" "buff" "buff" "buff" "sick" "buff"
> y_true_test
 [1] buff buff buff buff buff sick buff sick buff buff sick sick sick buff buff sick buff sick sick buff buff sick
[23] buff sick buff sick sick sick sick sick buff sick buff sick buff buff buff sick sick sick sick buff sick buff
[45] sick sick buff sick buff sick buff sick sick buff sick buff buff buff sick buff
Levels: buff sick
> length(y_true_test)
[1] 60
> length(predicted[,3])
[1] 60
> error<-length(x)/length(predicted[,3])
> error
[1] 0.2166667
```

Explanation:

```
> predicted_prune_tree<-predict.pruned_tree_frame %>%
+     mutate(diag1 = case_when(
+         predict.pruned_tree_frame$buff > predict.pruned_tree_frame$sick ~ "buff",
+         predict.pruned_tree_frame$buff < predict.pruned_tree_frame$sick ~ "buff"
+     ))
> x<- which(predicted_prune_tree[,3]!=y_true_test)
> error<-length(x)/length(predicted_prune_tree[,3])
> error
[1] 0.5
>
```

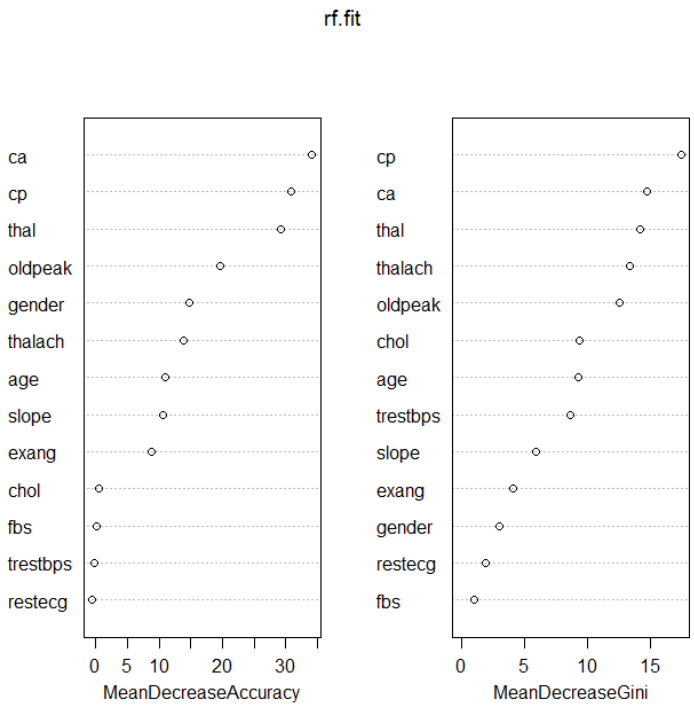
Error acquired for Pruned Tree is less than Full Tree model which is obvious as the Full Tree model being greedy tries to fit all the data. Thus we are getting an accuracy of 50% with pruned tree and 79% with full tree model

Training with random Forest

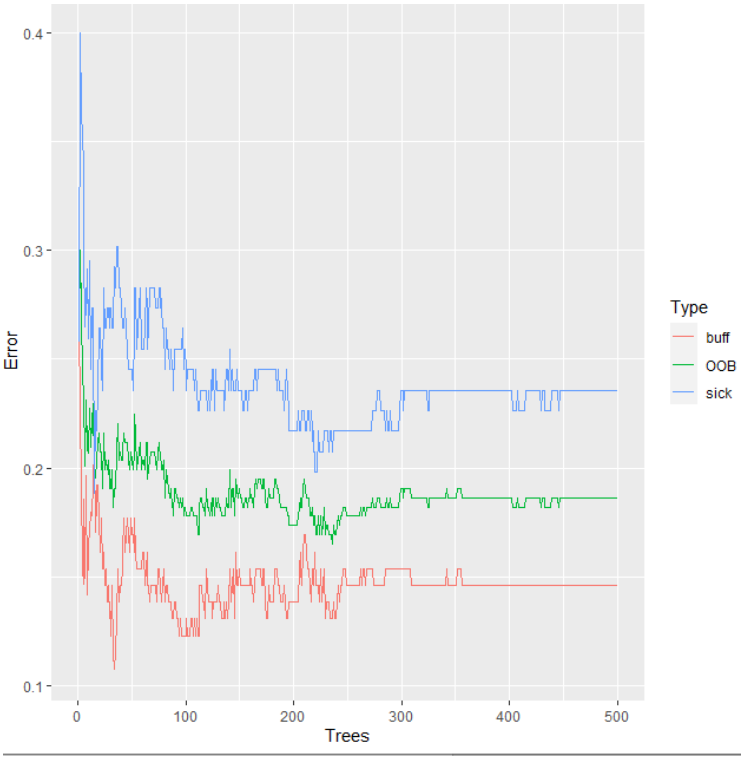
```
Call:
 randomForest(formula = cleveland_train_class ~ ., data = cleveland_train,      importance = TRUE, ntree = 1000)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 3

        OOB estimate of  error rate: 17.8%
Confusion matrix:
     buff sick class.error
buff  110   20   0.1538462
sick   22   84   0.2075472
> y_hat <- predict(rf.fit, newdata = cleveland_test, type = "response")
> y_hat <- as.numeric(y_hat)-1
> y_true <- as.numeric(cleveland_test_class)-1
> misclass_rf <- sum(abs(y_true- y_hat))/length(y_hat)
> misclass_rf
[1] 0.15
> 1-misclass_rf
[1] 0.85
>
```

Thus we are achieving an accuracy of 85percent on test data with 1000 trees which is better than both pruned and full tree

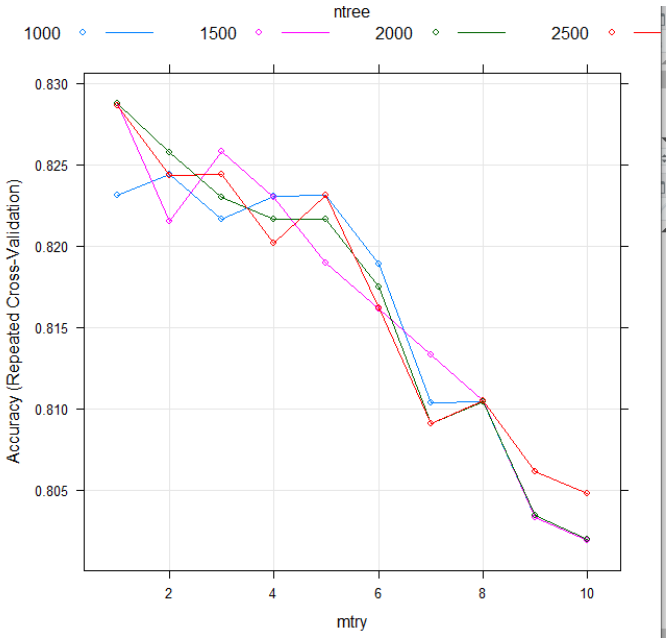


In Random Forest Each tree has its own out-of-bag sample of data that was not used during construction. This sample is used to calculate importance of a specific variable. It seems cp, thal and ca are very important variables because the mean Decrease Gini is very high suggesting that the difference between RSS before and after split at those variables is would have been very high although these are left out from splitting .Also from the mean decrease accuracy graph it seems that the exclusion of these same three variables reduces the accuracy between 15-20% .However these conclusions are based on on univariate distribution of each of this predictors rather we should look at the multivariate distribution of the data in order to decide which variables should be part of the model.

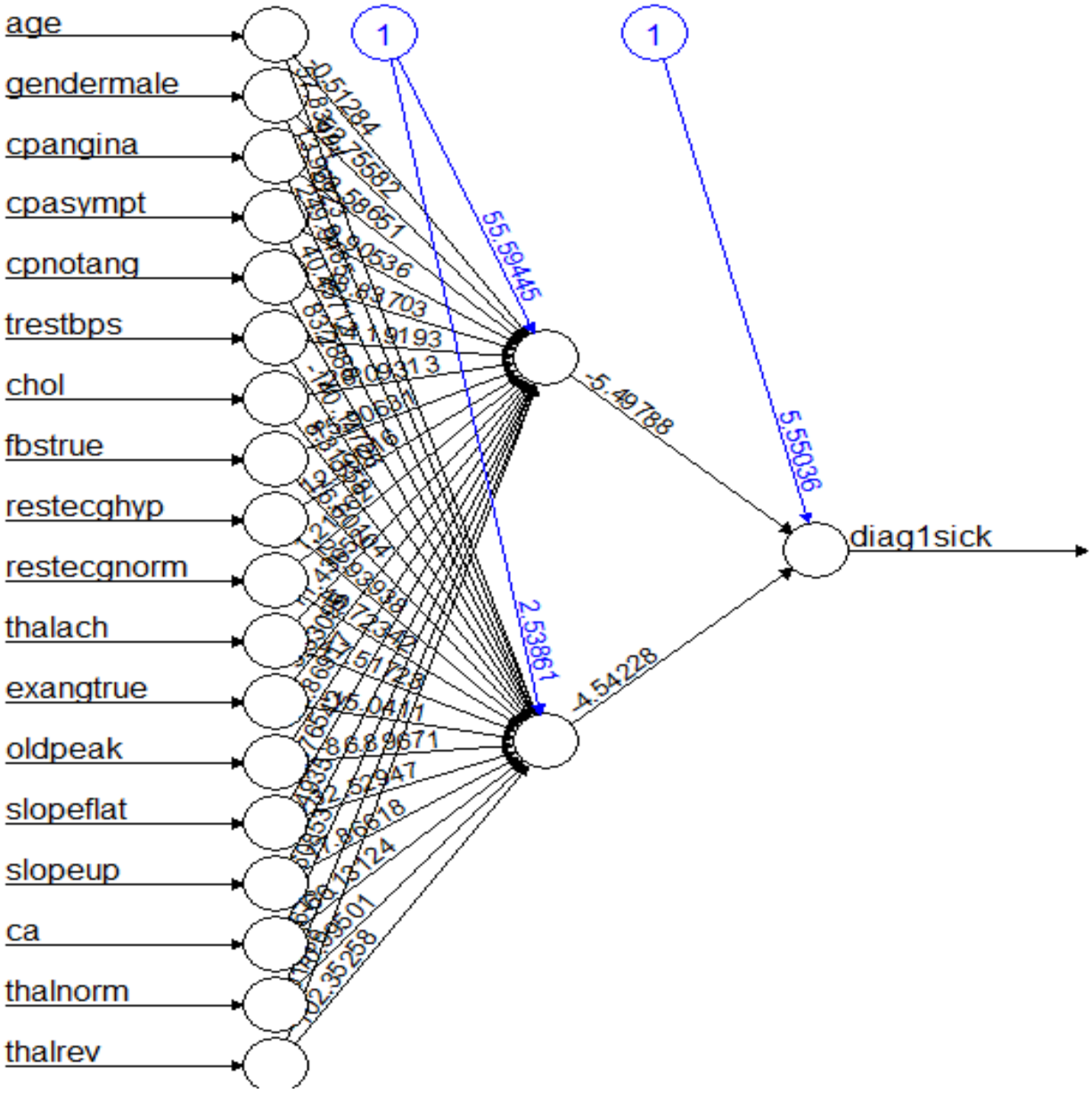


Red line represents MCR of class who are not having heart disease, blue line represents MCR of class having heart diseases and green line represents overall MCR or OOB error. Overall error rate is what we

Custom Tuning of Random Forest:



Thus, after tuning random forest it is observed that most accurate values for ntree and mtry were 2000 and 1 with an accuracy of 82.9%.




```
> train_err <- length(which(diag1 != y_hat_train))/length(y_hat_train)
> train_err
[1] 0.1610169
>
```

```
> table(actual,prediction)
      prediction
actual  0  1
     0 22  8
     1  5 25
> mean(actual==prediction)
[1] 0.7833333
>
```

```
> table(diag1_test,prediction_on_test)
          prediction_on_test
diag1_test  0  1
         0 19 11
         1  6 24
> mean(diag1_test==prediction_on_test)
[1] 0.7166667
```

With deep neural network we are getting an accuracy of 71 % with 1 hidden layer and two neurons on test data.

Tuning of the neural network

```
>
> train_err_store
[1] 0.44915254 0.44915254 0.15677966 0.09322034
> test_err_store
[1] 0.5000000 0.5000000 0.3166667 0.1333333
>
```

It seems there is sharp drop in test error as well as test error as well as training error when no.of neurons is 3 again there is great decrease in test error when the no.of neurons is 4 so optimal values for neurons for training Cleveland data would be is 4 with training error as 0.093 and test error as 0.133.

It seems out of the three models Random Forest is providing an accuracy of 85percent after tuning the model , tree model is giving an accuracy of 79 percent and neural network is providing an accuracy of 87 percent for four neurons after fine tuning it. So Neural Networks seems to lead ahead in terms of accuracy and performance compare to other models.

Task3 Training on OJ data

```
> sum(is.na("OJ"))
[1] 0
> 
```

No empty or missing values are found for this dataset

```
[1] 0
> is.null("OJ")
[1] FALSE
> 
```

a,b)Divided the data intro train and test and fitted a linear SVM with cost=0.01

```
Call:
svm(formula = Purchase ~ ., data = training, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  435

 ( 219 216 )


Number of Classes:  2

Levels:
 CH MM
> summary(svm_linear)

Call:
svm(formula = Purchase ~ ., data = OJ_train, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  452

 ( 226 226 )


Number of Classes:  2

Levels:
 CH MM
```

c)

```
> postResample(predict(svm_linear, training), training$Purchase)
 Accuracy     Kappa
0.8250000 0.6313971
>
```

Thus Training error when fitted with linear SVM model is 0.175

```
> postResample(predict(svm_linear, testing), testing$Purchase)
 Accuracy     Kappa
0.8222222 0.6082699
```

Thus test error when fitted with linear SVM model is 0.178

d)Tune the SVM model to select an optimal cost between 0.01 to 10.

```
Support Vector Machines with Linear Kernel

800 samples
 17 predictor
  2 classes: 'CH', 'MM'

Pre-processing: centered (17), scaled (17)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 721, 720, 720, 720, 721, 719, ...
Resampling results across tuning parameters:

  cost        Accuracy   Kappa
   0.0100000  0.8199215  0.6202565
   0.5357895  0.8273760  0.6360834
   1.0615789  0.8236101  0.6284665
   1.5873684  0.8261105  0.6333280
   2.1131579  0.8261105  0.6333280
   2.6389474  0.8273605  0.6362121
   3.1647368  0.8261105  0.6338114
   3.6905263  0.8248605  0.6309732
   4.2163158  0.8248605  0.6309732
   4.7421053  0.8261105  0.6338114
   5.2678947  0.8273605  0.6361662
   5.7936842  0.8273605  0.6361662
   6.3194737  0.8260947  0.6331693
   6.8452632  0.8260947  0.6331693
   7.3710526  0.8260947  0.6331693

   7.8968421  0.8273605  0.6361662
   8.4226316  0.8273605  0.6361662
   8.9484211  0.8273605  0.6361662
   9.4742105  0.8248447  0.6308145
  10.0000000  0.8248447  0.6308145

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cost = 0.5357895.
```

e)Training and test errors after tuning linear SVM model

```
> postResample(predict(svm_linear_tune, training), training$Purchase)
 Accuracy     Kappa
0.8350000 0.6524601
>
```

Training error after tuning the linear SVM model is 0.165

```
> postResample(predict(svm_linear_tune, testing), testing$Purchase)
 Accuracy     Kappa
0.8444444 0.6585983
>
```

The testing error after tuning linear SVM model is 0.156

f) Training with Radial Kernel

```
> summary(svm_radial)

Call:
svm(formula = Purchase ~ ., data = training, method = "radial", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01

Number of Support Vectors:  634

 ( 319 315 )


Number of Classes:  2

Levels:
 CH MM
```

```
> postResample(predict(svm_radial, training), training$Purchase)
Accuracy    Kappa
 0.60625  0.00000
>
>
```

The training error with SVM radial kernel is 0.394

```
> postResample(predict(svm_radial, testing), testing$Purchase)
 Accuracy     Kappa
0.6222222 0.0000000
>
```

The testing error with SVM radial kernel is 0.378

Tuning the SVM radial kernel

```
> svm_radial_tune
Support Vector Machines with Radial Basis Function Kernel

800 samples
 17 predictor
  2 classes: 'CH', 'MM'

Pre-processing: centered (17), scaled (17)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 720, 719, 719, 721, 719, 720, ...
Resampling results across tuning parameters:

  C          Accuracy   Kappa
   0.0100000  0.6062600  0.0000000
   0.5357895  0.8274369  0.6315267
   1.0615789  0.8249527  0.6267051
   1.5873684  0.8199986  0.6165675
   2.1131579  0.8174982  0.6105624
   2.6389474  0.8149824  0.6041027
   3.1647368  0.8112166  0.5964807
   3.6905263  0.8112166  0.5964807
   4.2163158  0.8124512  0.5993391
   4.7421053  0.8137170  0.6021336
   5.2678947  0.8137174  0.6017074
   5.7936842  0.8137174  0.6017074
   6.3194737  0.8124828  0.5988491
   6.8452632  0.8124828  0.5988491
   7.3710526  0.8137641  0.6020343
```

```
> postResample(predict(svm_radial_tune, training), training$Purchase)
Accuracy    Kappa
0.851250 0.684392
>
```

The training error after tuning SVM radial kernel is 0.149

```
> postResample(predict(svm_radial_tune, testing), testing$Purchase)
 Accuracy     Kappa
0.8185185 0.6040582
>
```

The testing error after tuning SVM radial kernel is 0.182

g)Fitting with polynomial kernel

```
> summary(svm_poly)

Call:
svm(formula = Purchase ~ ., data = training, method = "polynomial", degree = 2, cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01

Number of Support Vectors:  634

 ( 319 315 )


Number of Classes:  2

Levels:
 CH MM
```

```
> postResample(predict(svm_poly, training), training$Purchase)
Accuracy    Kappa
 0.60625  0.00000
>
```

Thus the training error with polynomial kernel with degree 2 is 0.394

```
> postResample(predict(svm_poly, testing), testing$Purchase)
 Accuracy     Kappa
0.6222222 0.0000000
>
```

Thus the test error with polynomial kernel with degree 2 is 0.378

After tuning Polynomial Kernel:

Training error is: 0.15

```
> postResample(predict(svm_poly_tune, training), training$Purchase)
Accuracy    Kappa
0.850000 0.678295
>
```

Test error is :0.186

```
> postResample(predict(svm_poly_tune, testing), testing$Purchase)
 Accuracy     Kappa
0.8148148 0.5886654
>
```

Overall the models are very similar, but the radial kernel does best by a small margin.