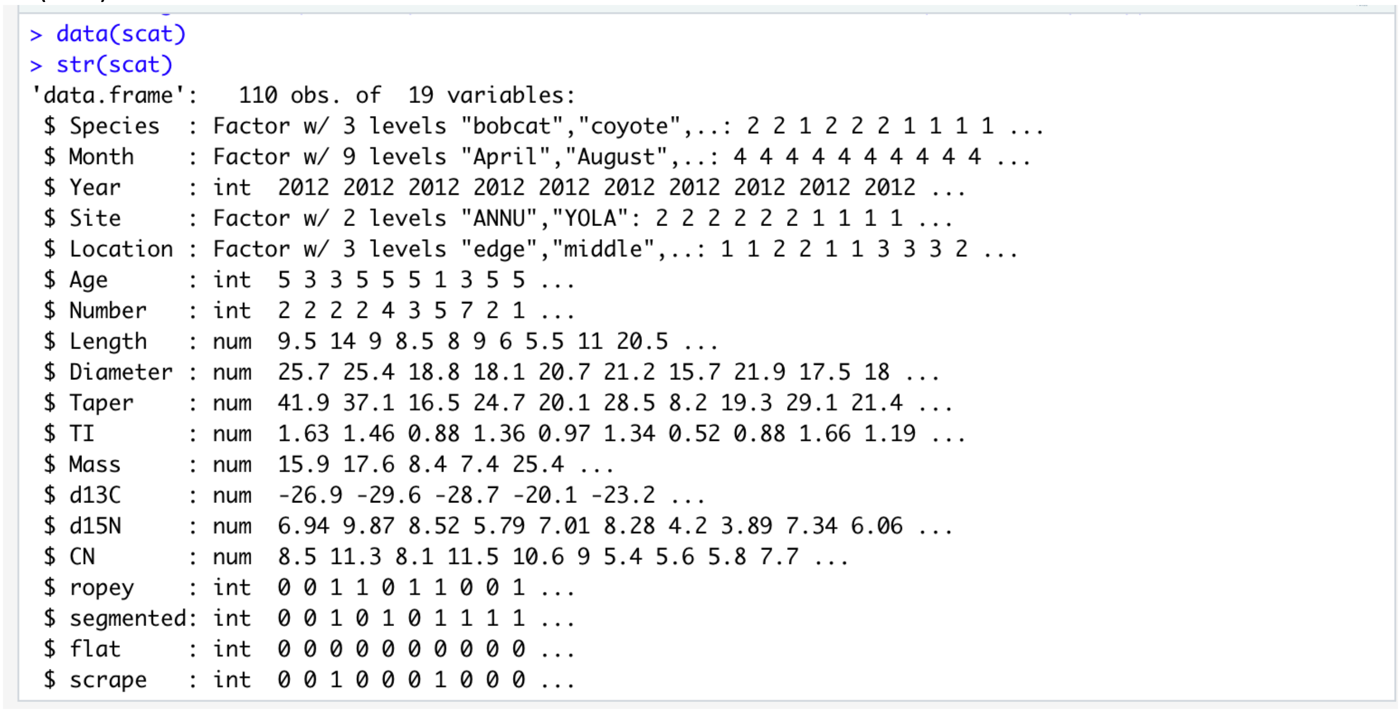
# Big Data Programming Assignment 5

## Susanth Dasari

### Big Data Programming class - Assignment 5
### Susanth Dasari

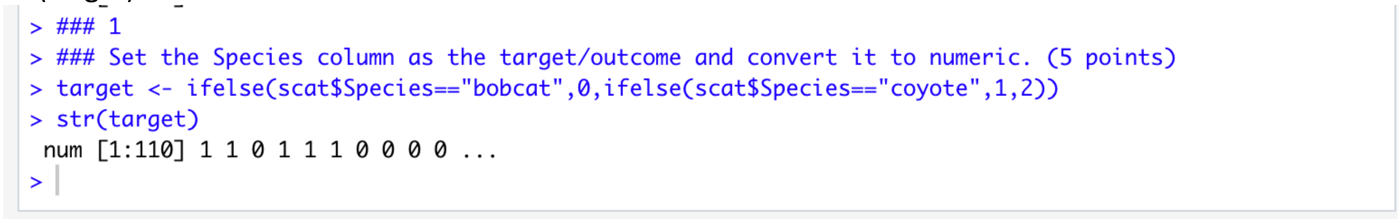
library(caret)
library(gbm)
library(RANN)
library(ggplot2)

### Using R 3.5.1 (or later) and caret's Animal Scat Data dataset (Hint: data(scat))
data(scat)
str(scat)



```
> data(scat)
> str(scat)
'data.frame':   110 obs. of  19 variables:
 $ Species  : Factor w/ 3 levels "bobcat","coyote",..: 2 2 1 2 2 2 1 1 1 1 ...
 $ Month    : Factor w/ 9 levels "April","August",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ Year     : int  2012 2012 2012 2012 2012 2012 2012 2012 2012 2012 ...
 $ Site     : Factor w/ 2 levels "ANNU","YOLA": 2 2 2 2 2 2 1 1 1 1 ...
 $ Location : Factor w/ 3 levels "edge","middle",..: 1 1 2 2 1 1 3 3 3 2 ...
 $ Age      : int  5 3 3 5 5 5 1 3 5 5 ...
 $ Number   : int  2 2 2 2 4 3 5 7 2 1 ...
 $ Length   : num  9.5 14 9 8.5 8 9 6 5.5 11 20.5 ...
 $ Diameter : num  25.7 25.4 18.8 18.1 20.7 21.2 15.7 21.9 17.5 18 ...
 $ Taper    : num  41.9 37.1 16.5 24.7 20.1 28.5 8.2 19.3 29.1 21.4 ...
 $ TI       : num  1.63 1.46 0.88 1.36 0.97 1.34 0.52 0.88 1.66 1.19 ...
 $ Mass     : num  15.9 17.6 8.4 7.4 25.4 ...
 $ d13C     : num  -26.9 -29.6 -28.7 -20.1 -23.2 ...
 $ d15N     : num  6.94 9.87 8.52 5.79 7.01 8.28 4.2 3.89 7.34 6.06 ...
 $ CN       : num  8.5 11.3 8.1 11.5 10.6 9 5.4 5.6 5.8 7.7 ...
 $ ropey    : int  0 0 1 1 0 1 1 0 0 1 ...
 $ segmented: int  0 0 1 0 1 0 1 1 1 1 ...
 $ flat     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ scrape   : int  0 0 1 0 0 0 1 0 0 0 ...
```

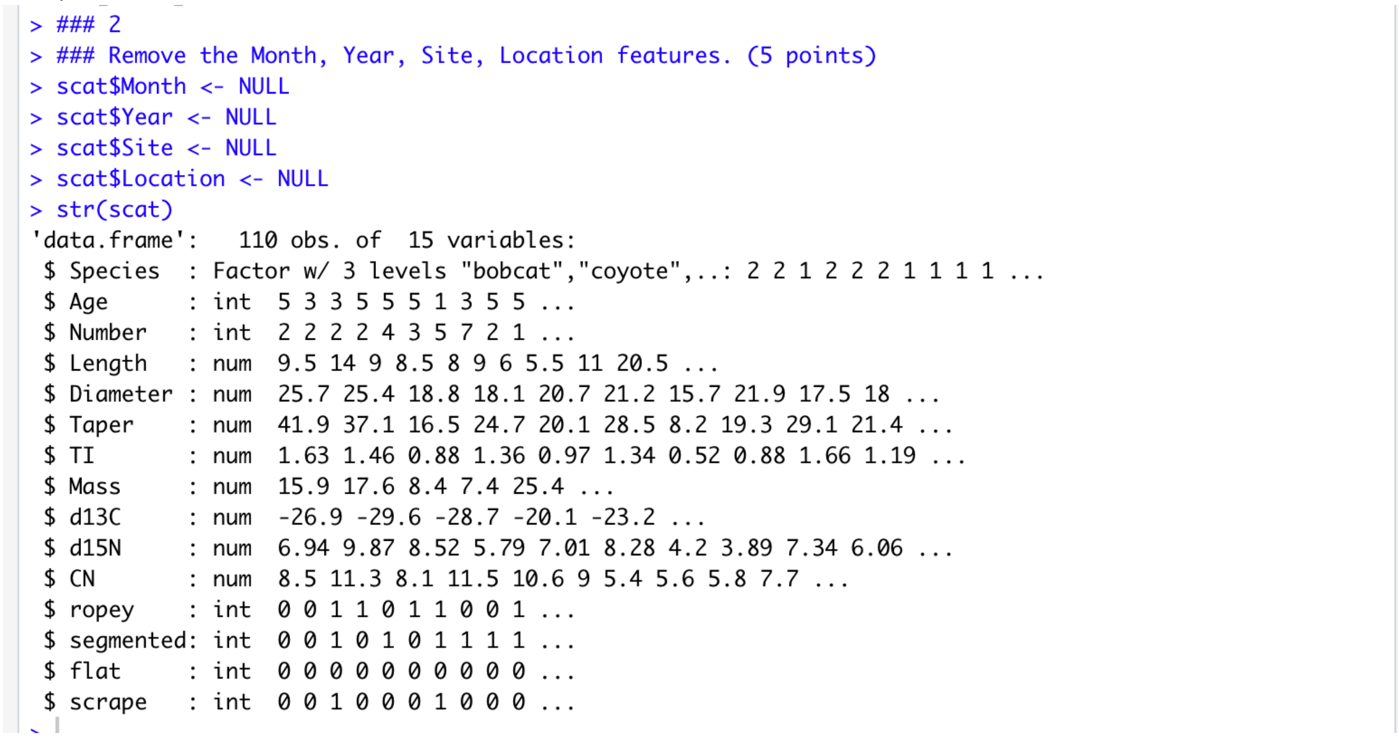
1
Set the Species column as the target/outcome and convert it to numeric. (5 points)
target <- ifelse(scat$Species=="bobcat",0,ifelse(scat$Species=="coyote",1,2))
str(target)



```
> ### 1
> ### Set the Species column as the target/outcome and convert it to numeric. (5 points)
> target <- ifelse(scat$Species=="bobcat",0,ifelse(scat$Species=="coyote",1,2))
> str(target)
 num [1:110] 1 1 0 1 1 1 0 0 0 0 ...
> 
```

2
### Remove the Month, Year, Site, Location features. (5 points)
scat$Month <- NULL
scat$Year <- NULL
scat$Site <- NULL

scat$Location <- NULL

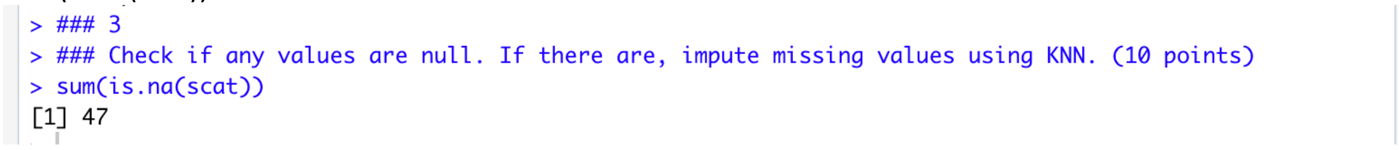


```
> ### 2
> ### Remove the Month, Year, Site, Location features. (5 points)
> scat$Month <- NULL
> scat$Year <- NULL
> scat$Site <- NULL
> scat$Location <- NULL
> str(scat)
'data.frame':	110 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "bobcat","coyote",..: 2 2 1 2 2 2 1 1 1 1 ...
 $ Age      : int  5 3 3 5 5 5 1 3 5 5 ...
 $ Number   : int  2 2 2 2 4 3 5 7 2 1 ...
 $ Length   : num  9.5 14 9 8.5 8 9 6 5.5 11 20.5 ...
 $ Diameter : num  25.7 25.4 18.8 18.1 20.7 21.2 15.7 21.9 17.5 18 ...
 $ Taper    : num  41.9 37.1 16.5 24.7 20.1 28.5 8.2 19.3 29.1 21.4 ...
 $ TI       : num  1.63 1.46 0.88 1.36 0.97 1.34 0.52 0.88 1.66 1.19 ...
 $ Mass     : num  15.9 17.6 8.4 7.4 25.4 ...
 $ d13C     : num  -26.9 -29.6 -28.7 -20.1 -23.2 ...
 $ d15N     : num  6.94 9.87 8.52 5.79 7.01 8.28 4.2 3.89 7.34 6.06 ...
 $ CN       : num  8.5 11.3 8.1 11.5 10.6 9 5.4 5.6 5.8 7.7 ...
 $ ropey    : int  0 0 1 1 0 1 1 0 0 1 ...
 $ segmented: int  0 0 1 0 1 0 1 1 1 1 ...
 $ flat     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ scrape   : int  0 0 1 0 0 0 1 0 0 0 ...
```

3

Check if any values are null. If there are, impute missing values using KNN. (10 points)

sum(is.na(scat))



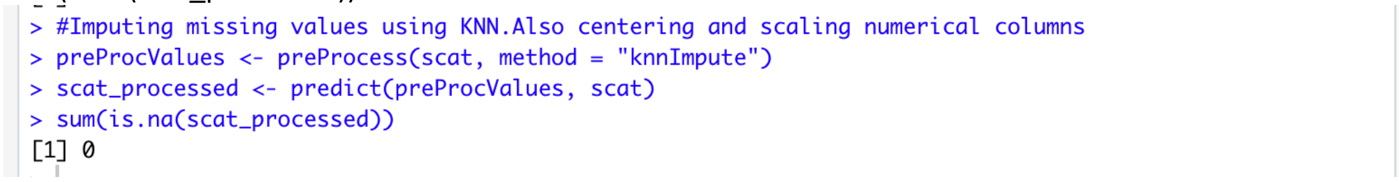
```
> ### 3
> ### Check if any values are null. If there are, impute missing values using KNN. (10 points)
> sum(is.na(scat))
[1] 47
```

#Imputing missing values using KNN.Also centering and scaling numerical columns

preProcValues <- preProcess(scat, method = "knnImpute")

scat_processed <- predict(preProcValues, scat)

sum(is.na(scat_processed))

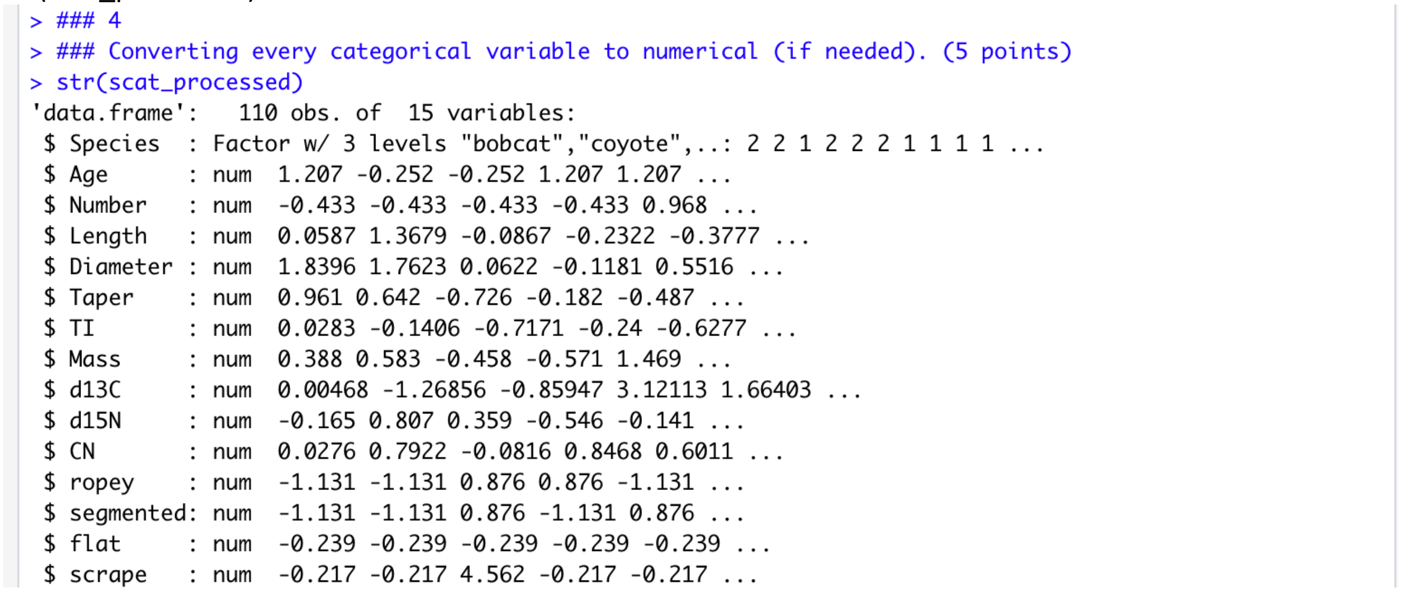


```
> #Imputing missing values using KNN.Also centering and scaling numerical columns
> preProcValues <- preProcess(scat, method = "knnImpute")
> scat_processed <- predict(preProcValues, scat)
> sum(is.na(scat_processed))
[1] 0
```

4

Converting every categorical variable to numerical (if needed). (5 points)

str(scat_processed)



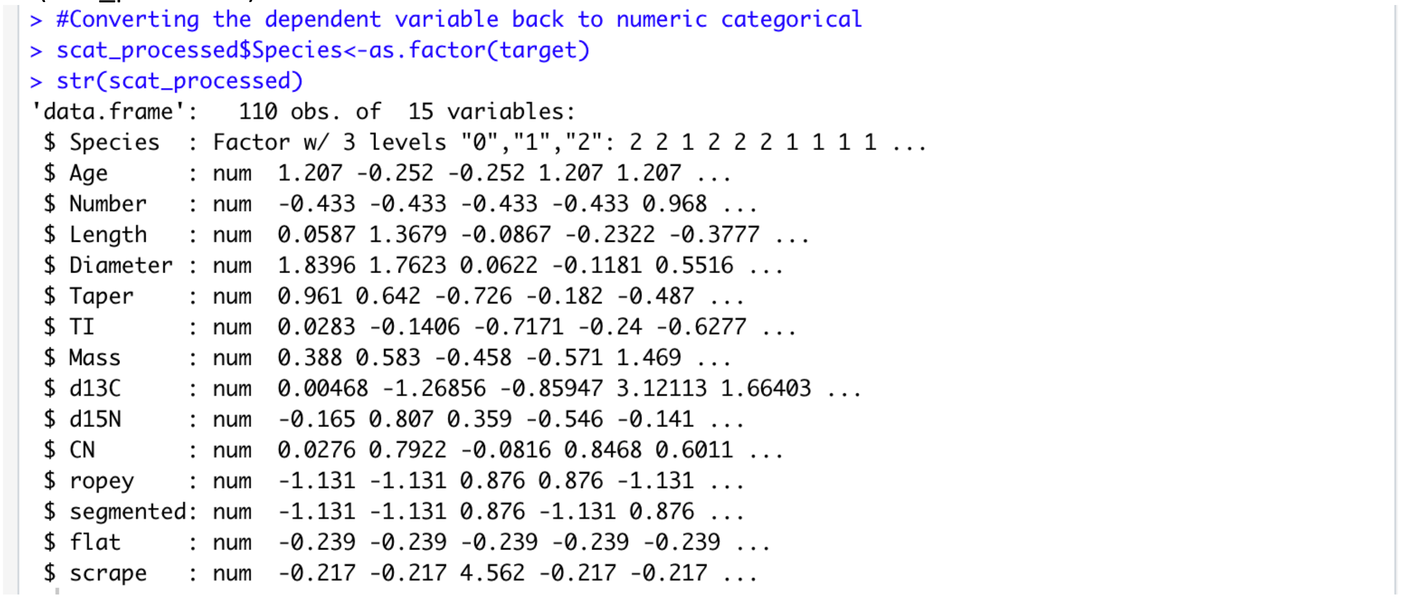
```
> ### 4
> ### Converting every categorical variable to numerical (if needed). (5 points)
> str(scat_processed)
'data.frame':	110 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "bobcat","coyote",..: 2 2 1 2 2 2 1 1 1 1 ...
 $ Age      : num  1.207 -0.252 -0.252 1.207 1.207 ...
 $ Number   : num  -0.433 -0.433 -0.433 -0.433 0.968 ...
 $ Length   : num  0.0587 1.3679 -0.0867 -0.2322 -0.3777 ...
 $ Diameter : num  1.8396 1.7623 0.0622 -0.1181 0.5516 ...
 $ Taper    : num  0.961 0.642 -0.726 -0.182 -0.487 ...
 $ TI       : num  0.0283 -0.1406 -0.7171 -0.24 -0.6277 ...
 $ Mass     : num  0.388 0.583 -0.458 -0.571 1.469 ...
 $ d13C     : num  0.00468 -1.26856 -0.85947 3.12113 1.66403 ...
 $ d15N     : num  -0.165 0.807 0.359 -0.546 -0.141 ...
 $ CN       : num  0.0276 0.7922 -0.0816 0.8468 0.6011 ...
 $ ropey    : num  -1.131 -1.131 0.876 0.876 -1.131 ...
 $ segmented: num  -1.131 -1.131 0.876 -1.131 0.876 ...
 $ flat     : num  -0.239 -0.239 -0.239 -0.239 -0.239 ...
 $ scrape   : num  -0.217 -0.217 4.562 -0.217 -0.217 ...
```

# There are no categorical variables remaining

#Converting the dependent variable back to numeric categorical

scat_processed$Species<-as.factor(target)

str(scat_processed)



```
> #Converting the dependent variable back to numeric categorical
> scat_processed$Species<-as.factor(target)
> str(scat_processed)
'data.frame':	110 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "0","1","2": 2 2 1 2 2 2 1 1 1 1 ...
 $ Age      : num  1.207 -0.252 -0.252 1.207 1.207 ...
 $ Number   : num  -0.433 -0.433 -0.433 -0.433 0.968 ...
 $ Length   : num  0.0587 1.3679 -0.0867 -0.2322 -0.3777 ...
 $ Diameter : num  1.8396 1.7623 0.0622 -0.1181 0.5516 ...
 $ Taper    : num  0.961 0.642 -0.726 -0.182 -0.487 ...
 $ TI       : num  0.0283 -0.1406 -0.7171 -0.24 -0.6277 ...
 $ Mass     : num  0.388 0.583 -0.458 -0.571 1.469 ...
 $ d13C     : num  0.00468 -1.26856 -0.85947 3.12113 1.66403 ...
 $ d15N     : num  -0.165 0.807 0.359 -0.546 -0.141 ...
 $ CN       : num  0.0276 0.7922 -0.0816 0.8468 0.6011 ...
 $ ropey    : num  -1.131 -1.131 0.876 0.876 -1.131 ...
 $ segmented: num  -1.131 -1.131 0.876 -1.131 0.876 ...
 $ flat     : num  -0.239 -0.239 -0.239 -0.239 -0.239 ...
 $ scrape   : num  -0.217 -0.217 4.562 -0.217 -0.217 ...
```

5

### With a seed of 100, 75% training, 25% testing.

### Build the following models: randomforest, neural net, naive bayes and GBM.

### For these models display

###   a) model summarization and

###   b) plot variable of importance, for the predictions (use the prediction set) display

###   c) confusion matrix (60 points)

########## Splitting Data Using Caret ##############

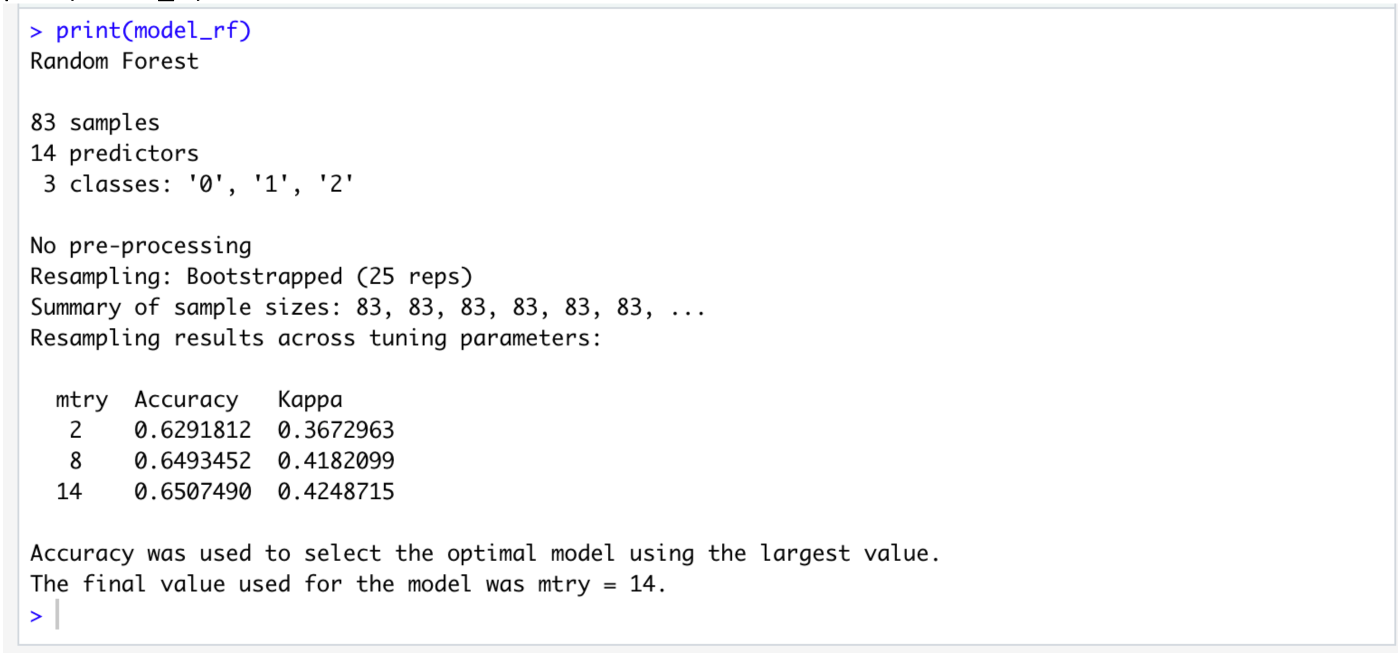
#Spliting training set into two parts based on outcome: 75% and 25%

set.seed(100)
index <- createDataPartition(scat_processed$Species, p=0.75, list=FALSE)
trainSet <- scat_processed[index,]
testSet <- scat_processed[-index,]

outcomeName<-'Species'
predictors<-names(trainSet)[!names(trainSet) %in% outcomeName]

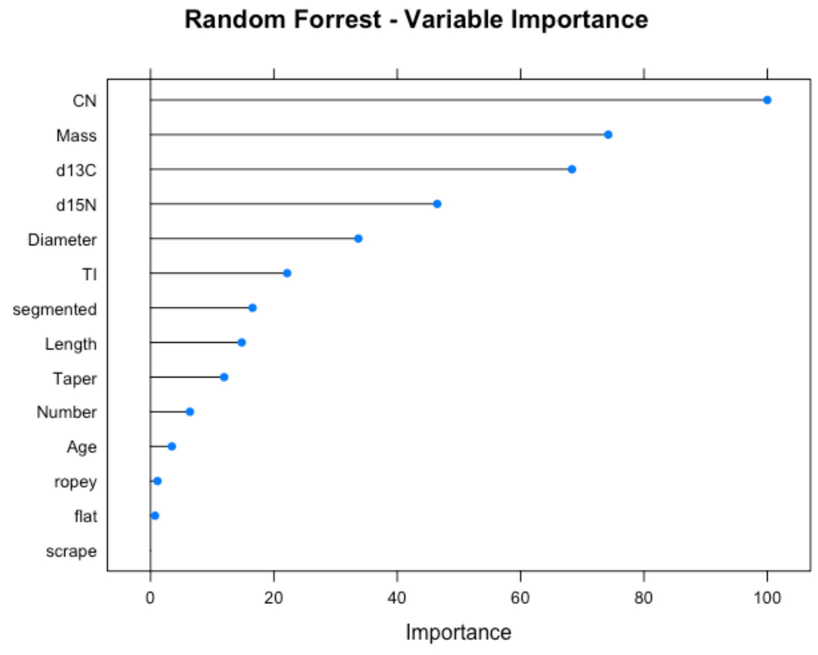
######## Training Models Using Caret ############
model_rf<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf')
model_nnet<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet')
model_nb<-train(trainSet[,predictors],trainSet[,outcomeName],method='naive_bayes')
model_gbm<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')

### Model Statistics for Random Forrest
print(model_rf)

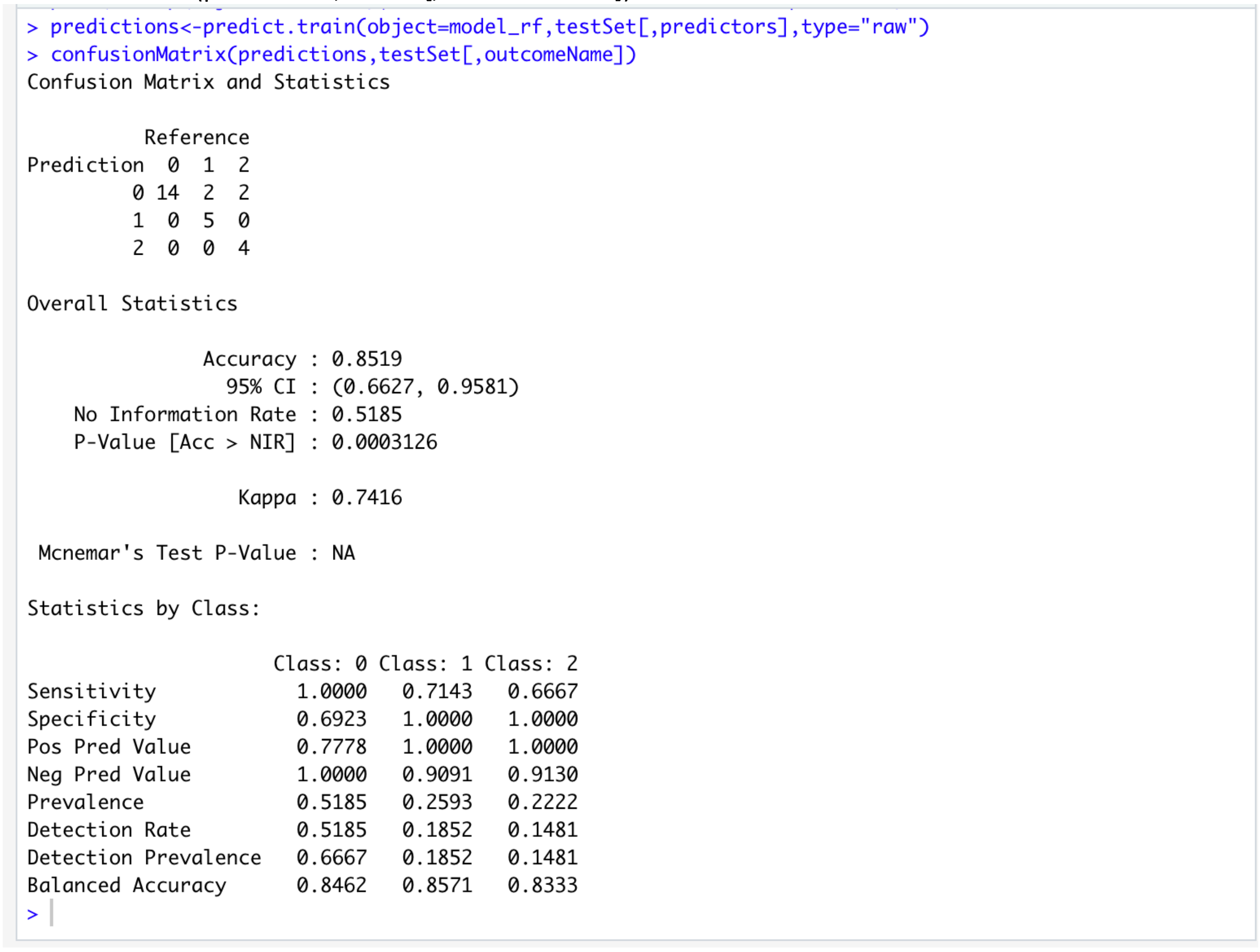


```
> print(model_rf)
Random Forest

83 samples
14 predictors
 3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.6291812  0.3672963
   8    0.6493452  0.4182099
  14    0.6507490  0.4248715

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 14.
>
```

plot(varImp(object=model_rf),main="Random Forrest - Variable Importance")



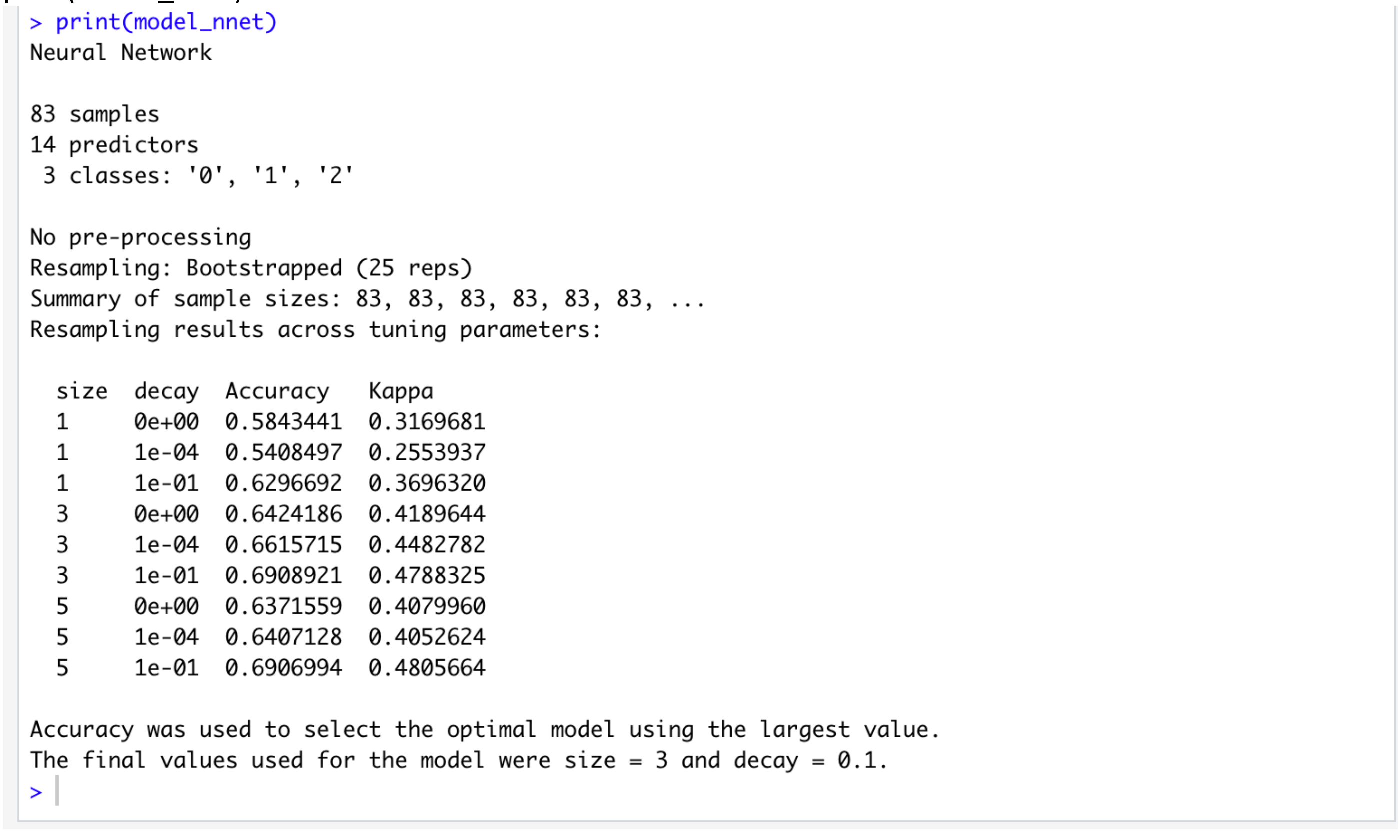
predictions<-predict.train(object=model_rf,testSet[,predictors],type="raw")
confusionMatrix(predictions,testSet[,outcomeName])



```
> predictions<-predict.train(object=model_rf,testSet[,predictors],type="raw")
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0 14  2  2
         1  0  5  0
         2  0  0  4

Overall Statistics

               Accuracy : 0.8519
                 95% CI : (0.6627, 0.9581)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.0003126

                  Kappa : 0.7416

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   0.7143   0.6667
Specificity            0.6923   1.0000   1.0000
Pos Pred Value         0.7778   1.0000   1.0000
Neg Pred Value         1.0000   0.9091   0.9130
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.5185   0.1852   0.1481
Detection Prevalence   0.6667   0.1852   0.1481
Balanced Accuracy      0.8462   0.8571   0.8333
>
```

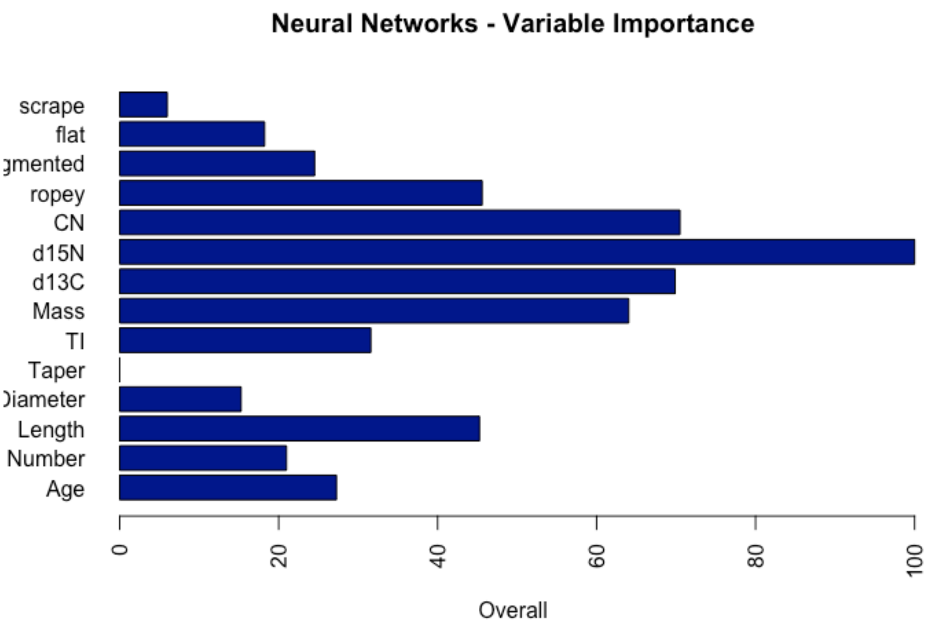
### Model Statistics for Neural Networks

print(model_nnet)



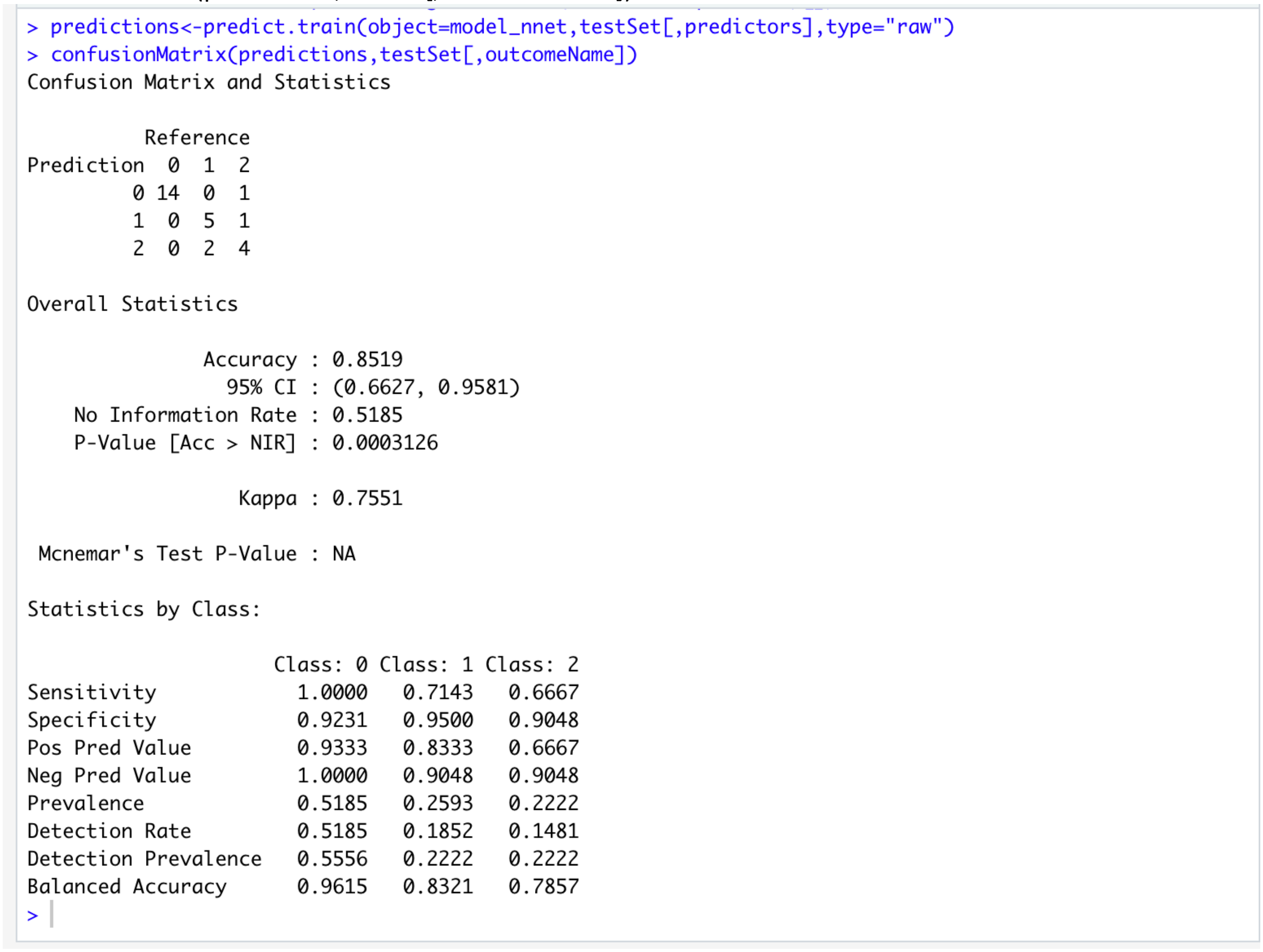
```
> print(model_nnet)
Neural Network

83 samples
14 predictors
 3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
Resampling results across tuning parameters:

  size  decay  Accuracy   Kappa
  1     0e+00  0.5843441  0.3169681
  1     1e-04  0.5408497  0.2553937
  1     1e-01  0.6296692  0.3696320
  3     0e+00  0.6424186  0.4189644
  3     1e-04  0.6615715  0.4482782
  3     1e-01  0.6908921  0.4788325
  5     0e+00  0.6371559  0.4079960
  5     1e-04  0.6407128  0.4052624
  5     1e-01  0.6906994  0.4805664

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were size = 3 and decay = 0.1.
> 
```

```
var_nnet <- varImp(object=model_nnet)
barplot(var_nnet$importance[,'Overall'], main="Neural Networks - Variable Importance",
     xlab='Overall', horiz=TRUE, las=2,
     col="darkblue", names.arg=row.names(var_nnet$importance)[])
```



```
predictions<-predict.train(object=model_nnet,testSet[,predictors],type="raw")
```

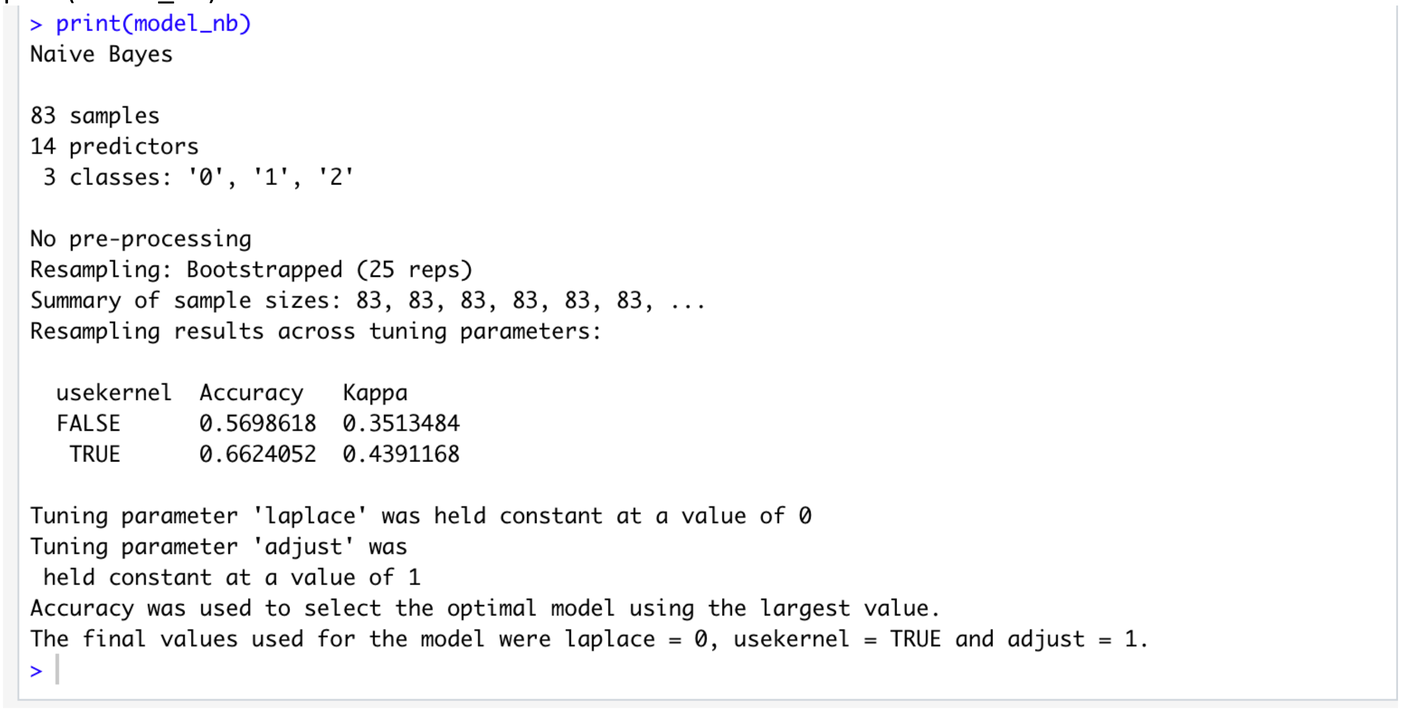
confusionMatrix(predictions,testSet[,outcomeName])



```
> predictions<-predict.train(object=model_nnet,testSet[,predictors],type="raw")
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0 14  0  1
         1  0  5  1
         2  0  2  4

Overall Statistics

               Accuracy : 0.8519
                 95% CI : (0.6627, 0.9581)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.0003126

                  Kappa : 0.7551

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   0.7143   0.6667
Specificity            0.9231   0.9500   0.9048
Pos Pred Value         0.9333   0.8333   0.6667
Neg Pred Value         1.0000   0.9048   0.9048
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.5185   0.1852   0.1481
Detection Prevalence   0.5556   0.2222   0.2222
Balanced Accuracy      0.9615   0.8321   0.7857
>
```

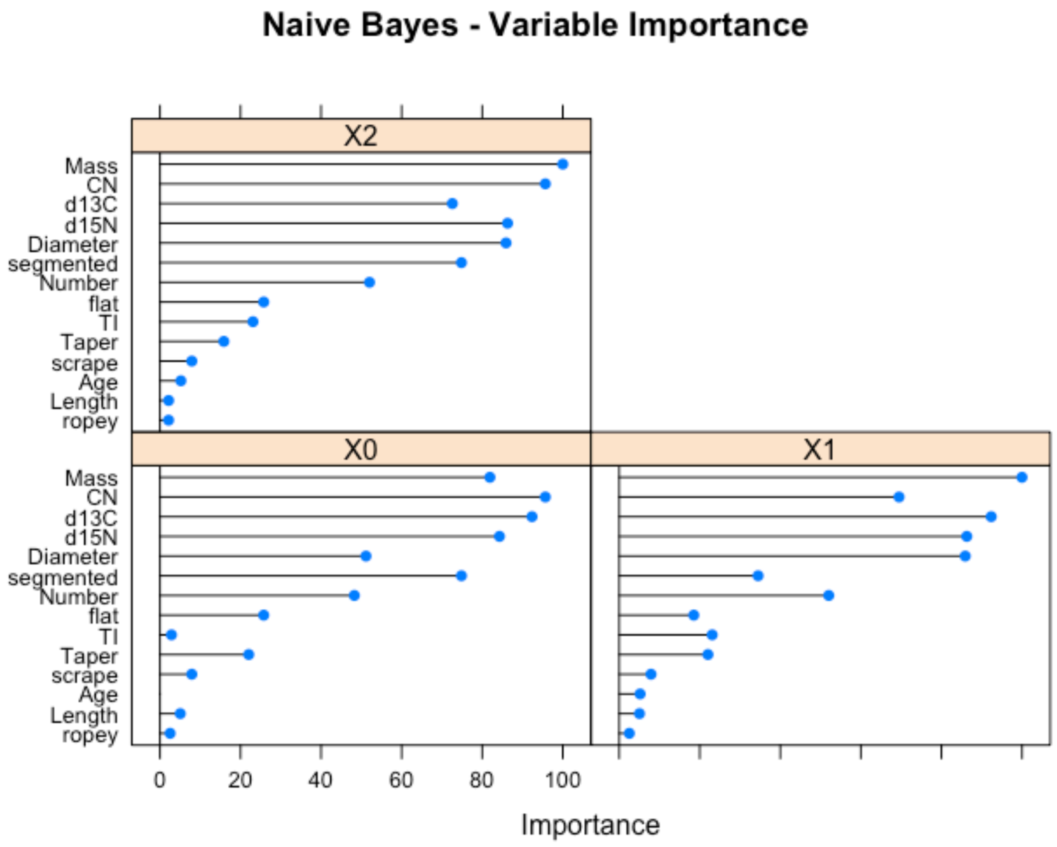
### Model Statistics for Naive Bayes

print(model_nb)

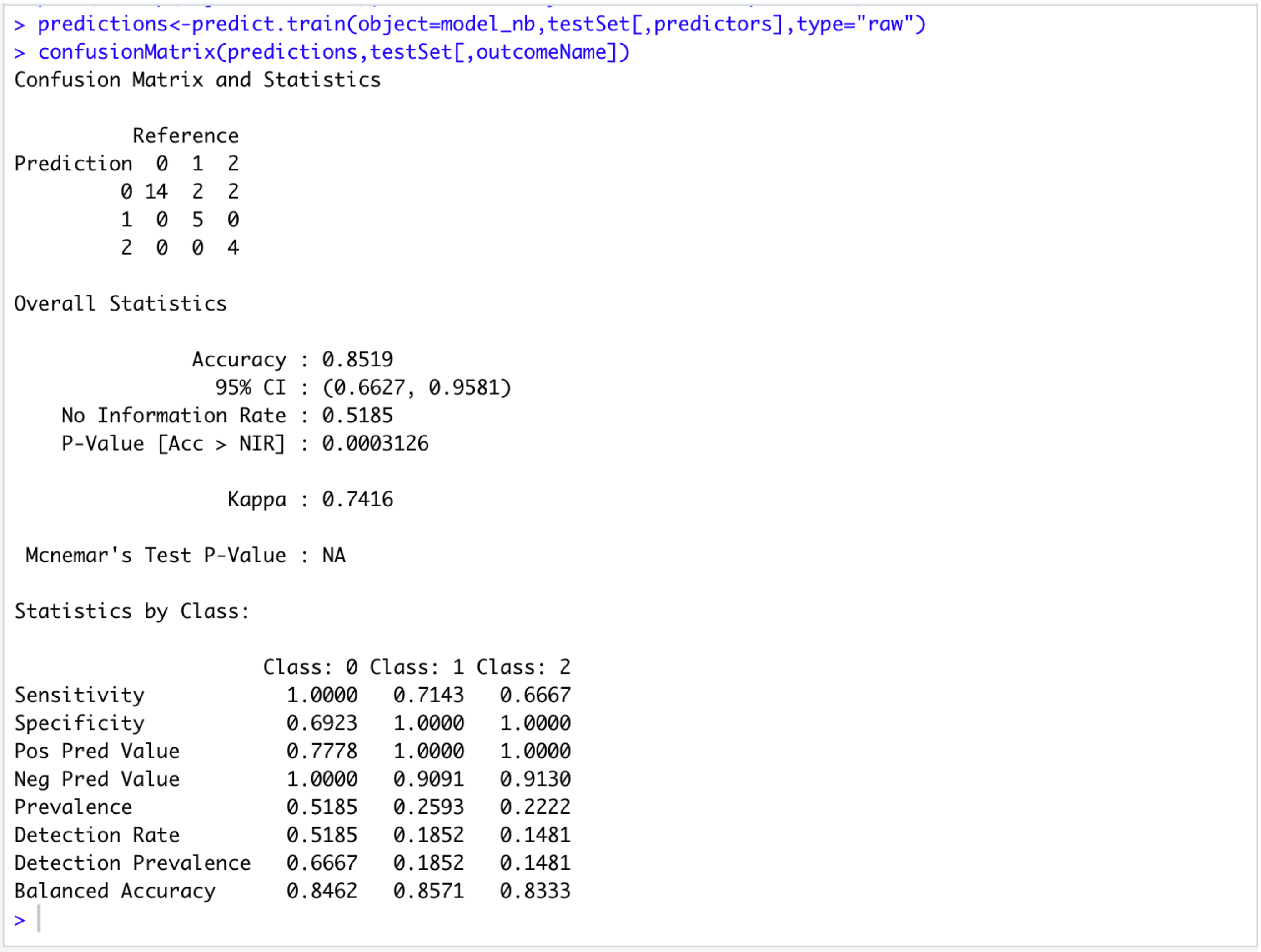


```
> print(model_nb)
Naive Bayes

83 samples
14 predictors
 3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.5698618  0.3513484
   TRUE      0.6624052  0.4391168

Tuning parameter 'laplace' was held constant at a value of 0
Tuning parameter 'adjust' was
 held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were laplace = 0, usekernel = TRUE and adjust = 1.
>
```

plot(varImp(object=model_nb),main="Naive Bayes - Variable Importance")

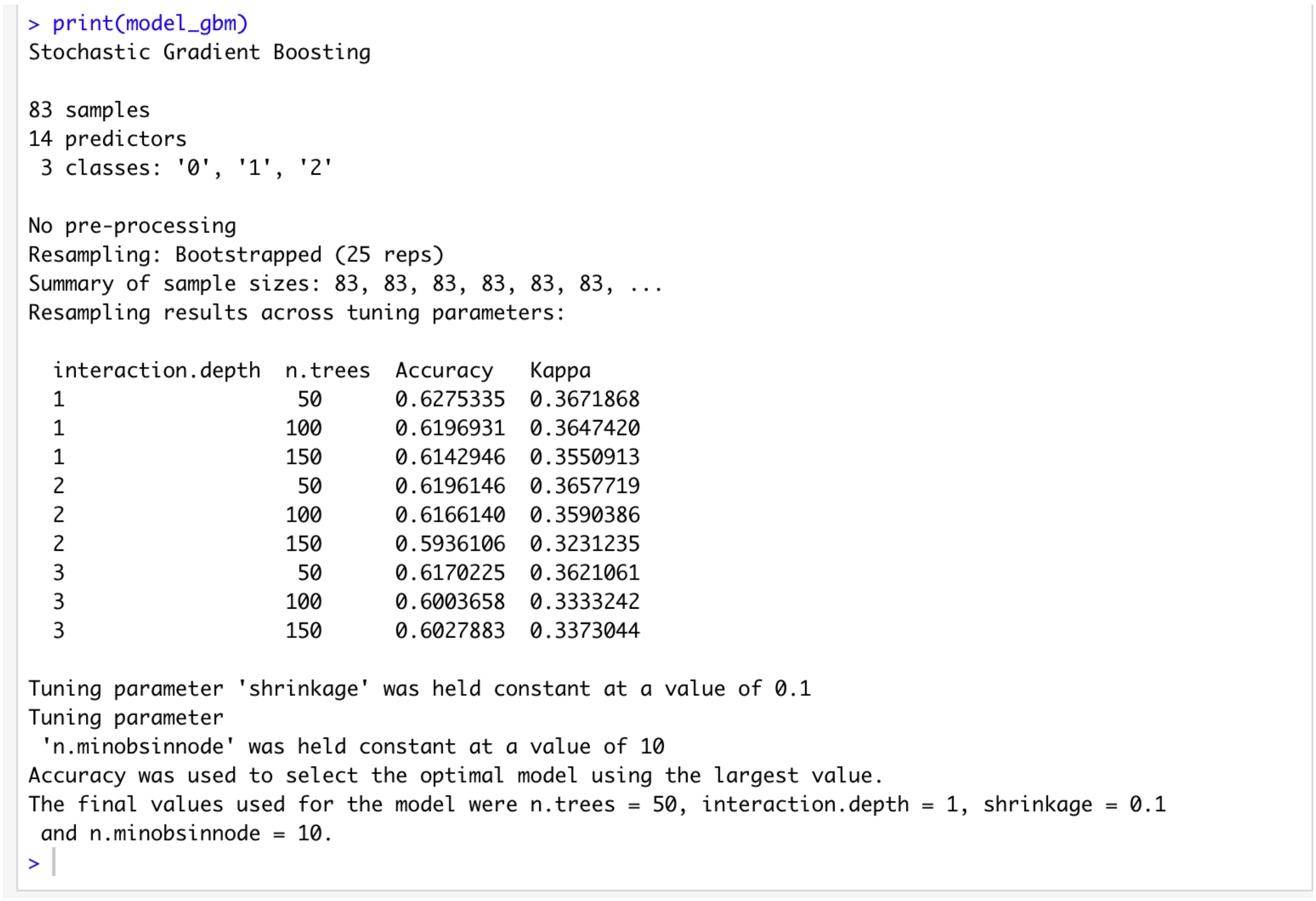


predictions<-predict.train(object=model_nb,testSet[,predictors],type="raw")
confusionMatrix(predictions,testSet[,outcomeName])



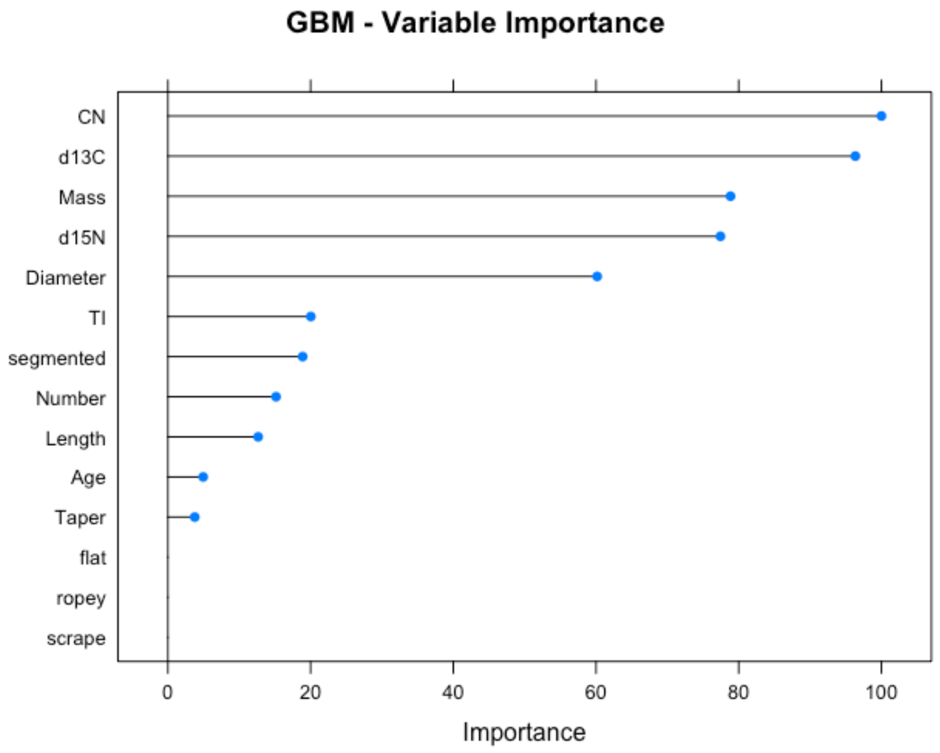
```
> predictions<-predict.train(object=model_nb,testSet[,predictors],type="raw")
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0 14  2  2
         1  0  5  0
         2  0  0  4

Overall Statistics

               Accuracy : 0.8519
                 95% CI : (0.6627, 0.9581)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.0003126

                  Kappa : 0.7416

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   0.7143   0.6667
Specificity            0.6923   1.0000   1.0000
Pos Pred Value         0.7778   1.0000   1.0000
Neg Pred Value         1.0000   0.9091   0.9130
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.5185   0.1852   0.1481
Detection Prevalence   0.6667   0.1852   0.1481
Balanced Accuracy      0.8462   0.8571   0.8333
> 
```

### Model Statistics for GBM
print(model_gbm)

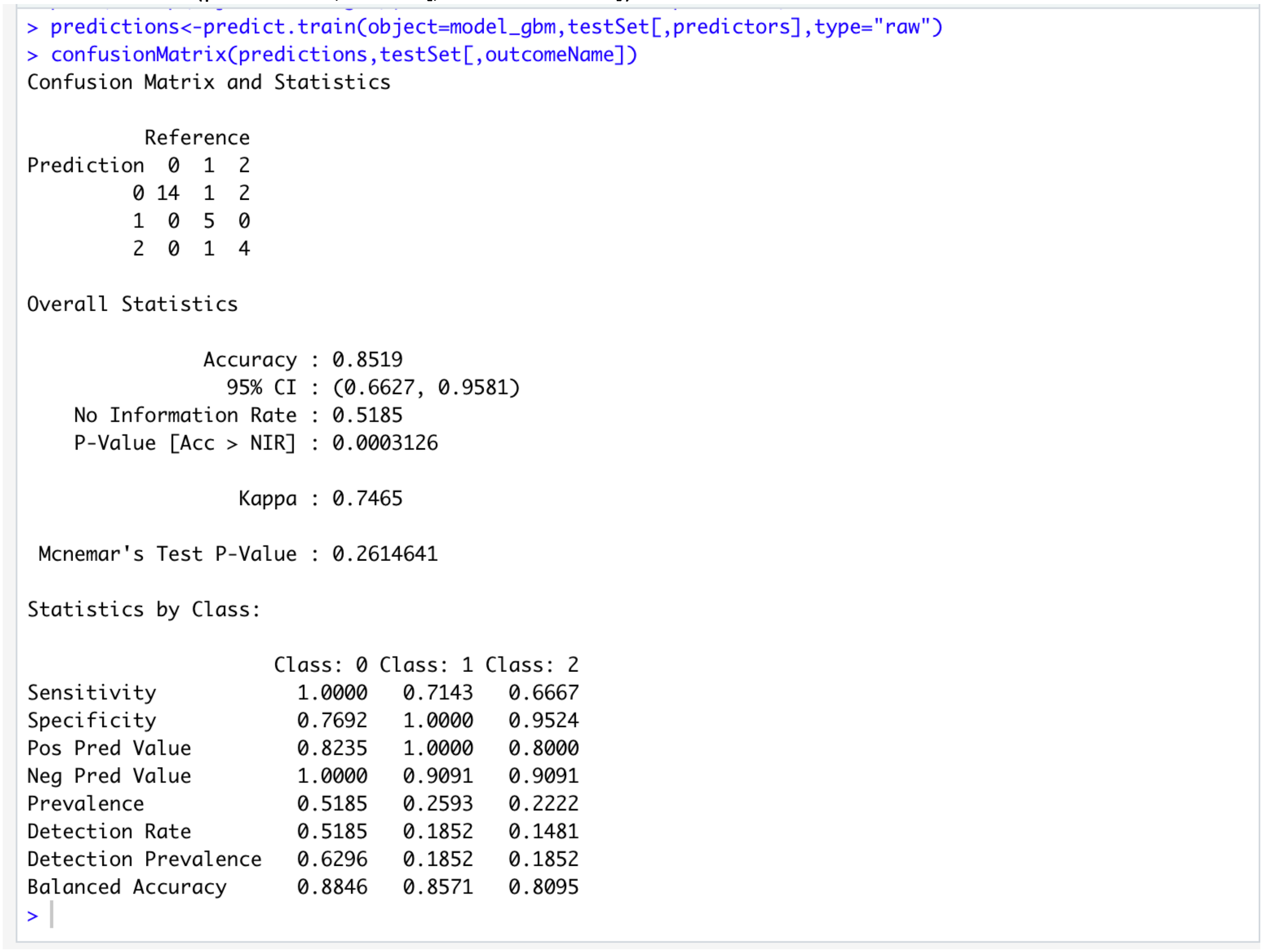


```
> print(model_gbm)
Stochastic Gradient Boosting

83 samples
14 predictors
 3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
Resampling results across tuning parameters:

  interaction.depth  n.trees  Accuracy   Kappa
  1                   50      0.6275335  0.3671868
  1                  100      0.6196931  0.3647420
  1                  150      0.6142946  0.3550913
  2                   50      0.6196146  0.3657719
  2                  100      0.6166140  0.3590386
  2                  150      0.5936106  0.3231235
  3                   50      0.6170225  0.3621061
  3                  100      0.6003658  0.3333242
  3                  150      0.6027883  0.3373044

Tuning parameter 'shrinkage' was held constant at a value of 0.1
Tuning parameter
 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 50, interaction.depth = 1, shrinkage = 0.1
 and n.minobsinnode = 10.
>
```

plot(varImp(object=model_gbm),main="GBM - Variable Importance")



predictions<-predict.train(object=model_gbm,testSet[,predictors],type="raw")
confusionMatrix(predictions,testSet[,outcomeName])

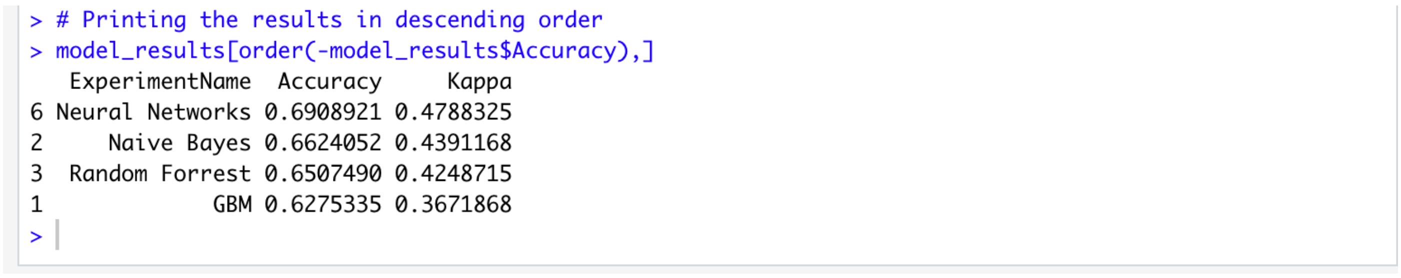


```
> predictions<-predict.train(object=model_gbm,testSet[,predictors],type="raw")
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0 14  1  2
         1  0  5  0
         2  0  1  4

Overall Statistics

               Accuracy : 0.8519
                 95% CI : (0.6627, 0.9581)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.0003126

                  Kappa : 0.7465

 Mcnemar's Test P-Value : 0.2614641

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   0.7143   0.6667
Specificity            0.7692   1.0000   0.9524
Pos Pred Value         0.8235   1.0000   0.8000
Neg Pred Value         1.0000   0.9091   0.9091
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.5185   0.1852   0.1481
Detection Prevalence   0.6296   0.1852   0.1852
Balanced Accuracy      0.8846   0.8571   0.8095
> 
```

6
### For the BEST performing models of each (randomforest, neural net, naive bayes and gbm)
### create and display a data frame that has the following columns:
###   ExperimentName, accuracy, kappa. Sort the data frame by accuracy. (15 points)

model_results <- NULL
model_results <- data.frame("ExperimentName" = "Random Forrest",model_rf$results[row.names(model_rf$bestTune),c("Accuracy","Kappa")])
newrow <- data.frame("ExperimentName" = "Neural Networks",model_nnet$results[row.names(model_nnet$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)
newrow <- data.frame("ExperimentName" = "Naive Bayes",model_nb$results[row.names(model_nb$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)
newrow <- data.frame("ExperimentName" = "GBM",model_gbm$results[row.names(model_gbm$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)

# Printing the results in descending order
model_results[order(-model_results$Accuracy),]

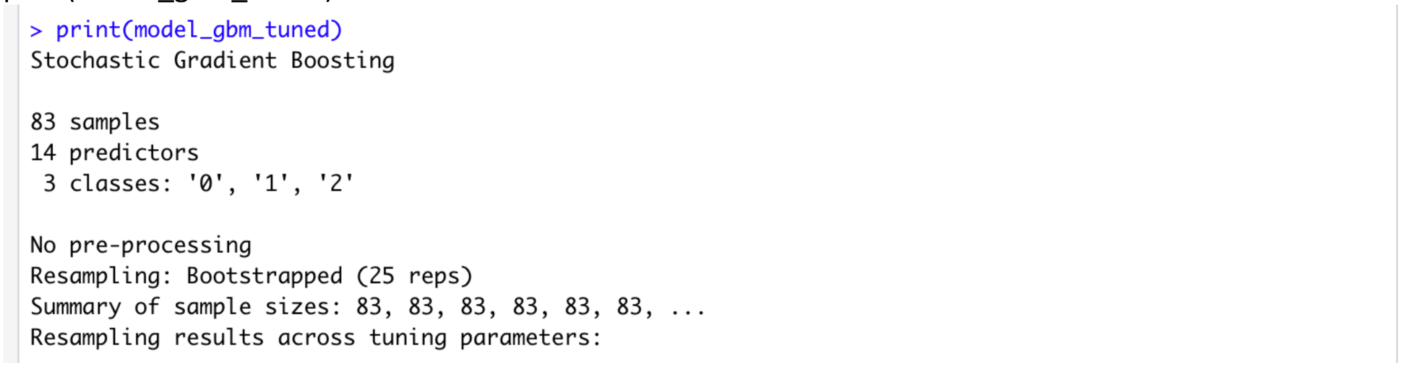


```
> # Printing the results in descending order
> model_results[order(-model_results$Accuracy),]
   ExperimentName  Accuracy     Kappa
6 Neural Networks 0.6908921 0.4788325
2     Naive Bayes 0.6624052 0.4391168
3  Random Forrest 0.6507490 0.4248715
1             GBM 0.6275335 0.3671868
> 
```

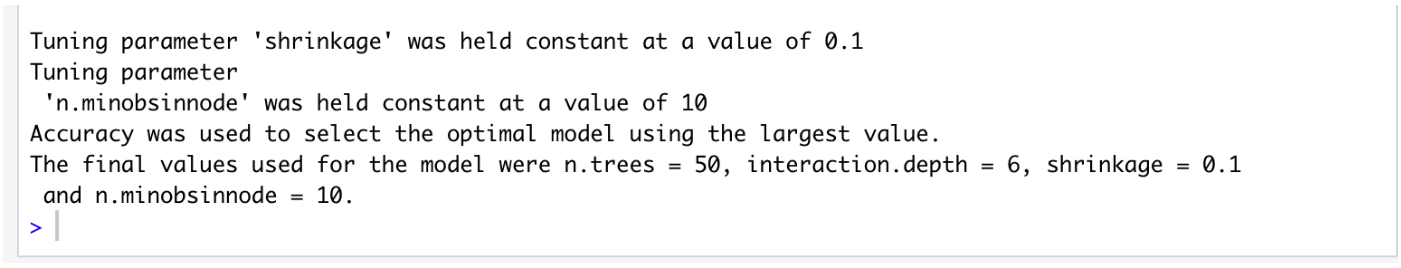
7
### Tune the GBM model using tune length = 20 and:
###   a) print the model summary and
###   b) plot the models. (20 points)

### Using tuneLength ###

model_gbm_tuned <- train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',tuneLength=20)
print(model_gbm_tuned)



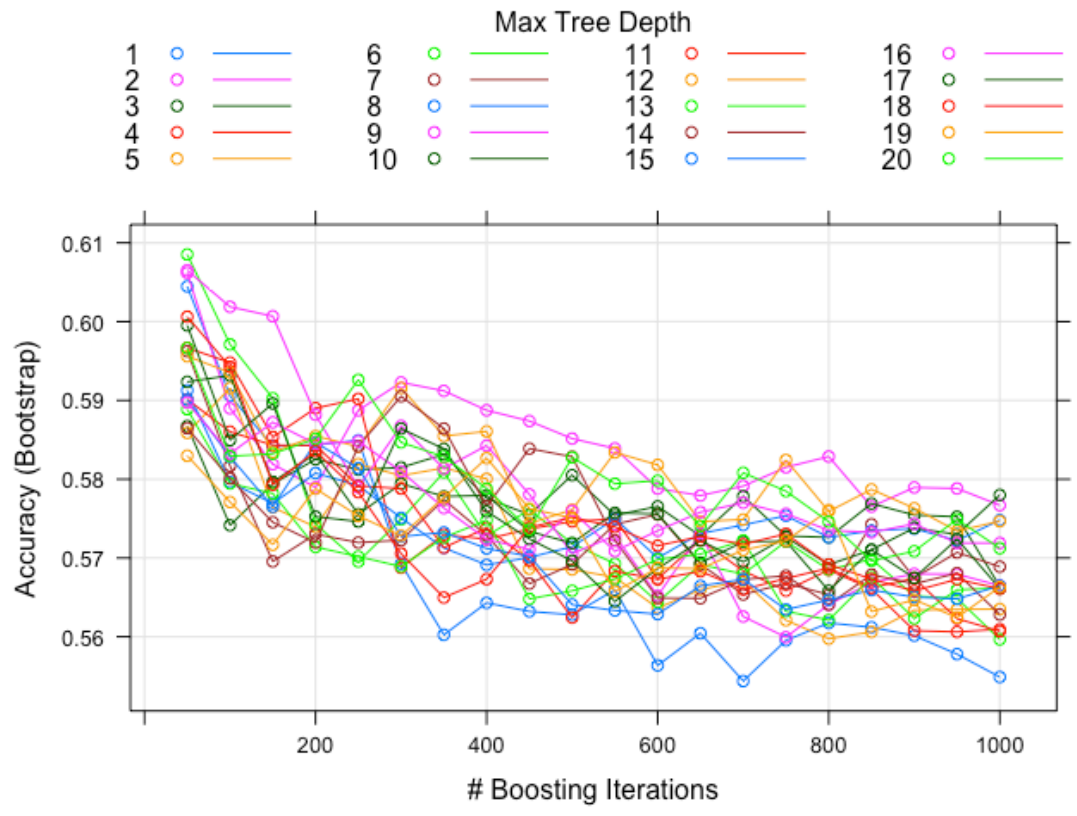
```
> print(model_gbm_tuned)
Stochastic Gradient Boosting

83 samples
14 predictors
 3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
Resampling results across tuning parameters:
```



```
Tuning parameter 'shrinkage' was held constant at a value of 0.1
Tuning parameter
 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 50, interaction.depth = 6, shrinkage = 0.1
 and n.minobsinnode = 10.
> 
```

# visualize the models

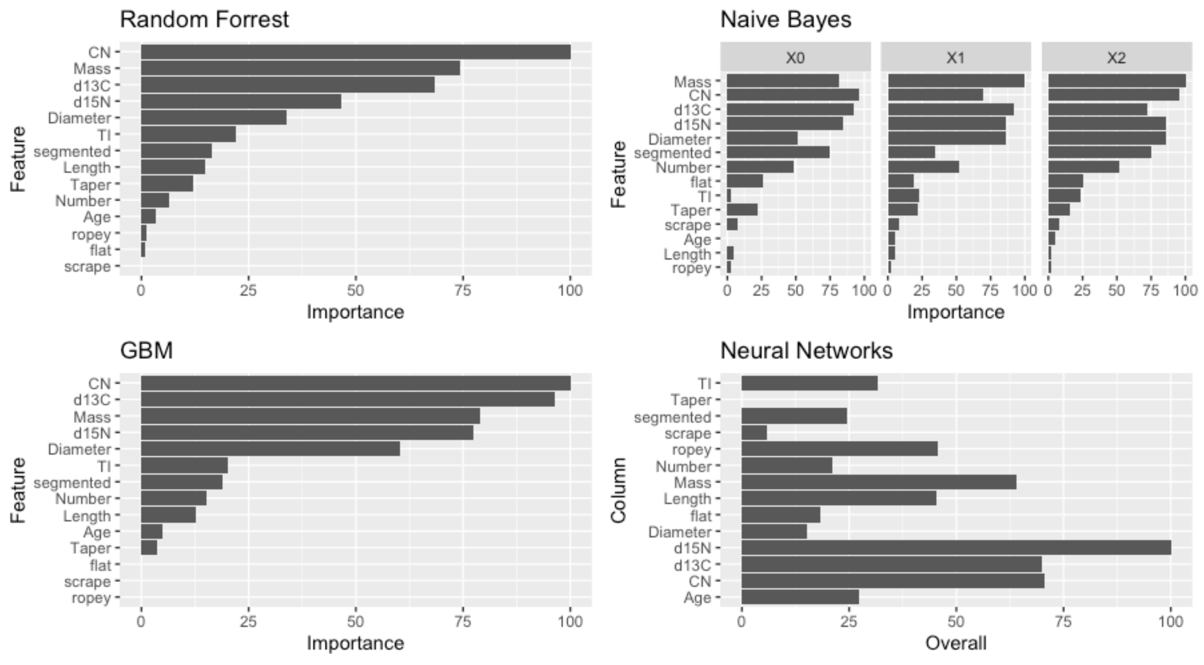
plot(model_gbm_tuned)



8
### Using GGplot and gridExtra to plot all variable of importance plots into one single plot. (10 points)

p1 <-ggplot(varImp(object=model_rf)) + ggtitle("Random Forrest")
p2 <-ggplot(varImp(object=model_nb)) + ggtitle("Naive Bayes")
p3 <-ggplot(varImp(object=model_gbm)) + ggtitle("GBM")
var_nnet_df <-
data.frame("Column"=row.names(var_nnet$importance)[],varImp(object=model_nnet)$importance)
p4 <- ggplot(var_nnet_df) + geom_col(aes(x=Column,y=Overall)) + coord_flip() + ggtitle("Neural Networks")

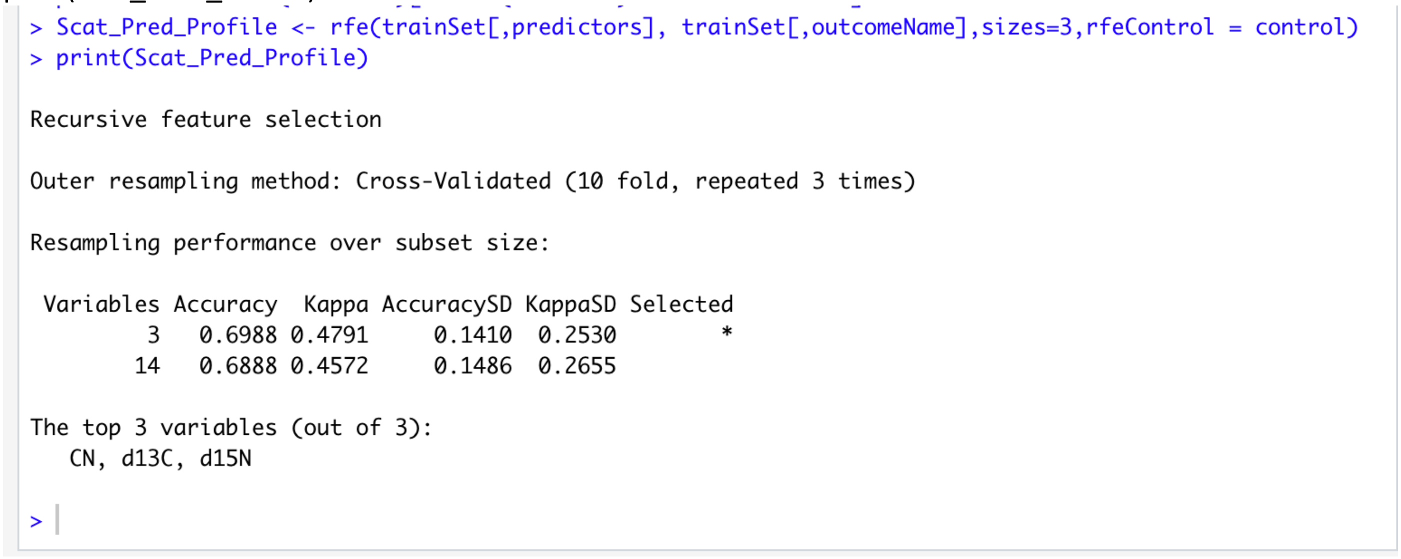
grid.arrange(p1,p2,p3,p4)



9
### Which model performs the best? and why do you think this is the case? Can we accurately predict species on this dataset? (10 points)

# After looking at the accuracies of the models, as of now Neural Network is performing the best.
# But since the accuracy is still 69%, I don't think we can predict the species accurately.
# It might because of the low size of data.

10
### Graduate Student Questions:
###   a.Using feature selection with rfe in caret and the repeatedcv method:
###   Find the top 3 predictors and build the same models as in 6 and 8 with the same parameters. (20 points)

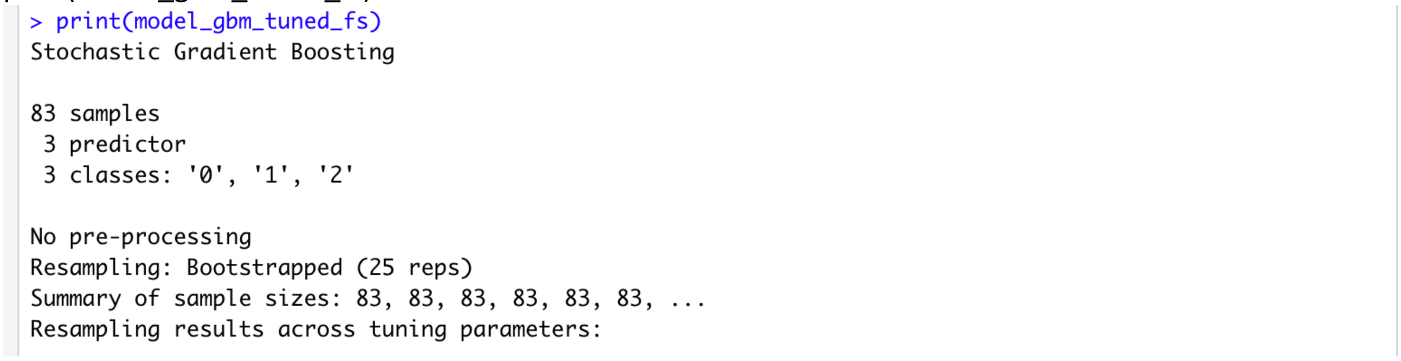
# Feature selection using rfe in caret
```
control <- rfeControl(functions = rfFuncs,
              method = "repeatedcv",
              repeats = 3,
              verbose = FALSE)
outcomeName<-'Species'
predictors<-names(trainSet)[!names(trainSet) %in% outcomeName]
Scat_Pred_Profile <- rfe(trainSet[,predictors], trainSet[,outcomeName],sizes=3,rfeControl = control)
print(Scat_Pred_Profile)
```

```
> Scat_Pred_Profile <- rfe(trainSet[,predictors], trainSet[,outcomeName],sizes=3,rfeControl = control)
> print(Scat_Pred_Profile)

Recursive feature selection

Outer resampling method: Cross-Validated (10 fold, repeated 3 times)

Resampling performance over subset size:

 Variables Accuracy  Kappa AccuracySD KappaSD Selected
         3   0.6988 0.4791     0.1410  0.2530        *
        14   0.6888 0.4572     0.1486  0.2655

The top 3 variables (out of 3):
   CN, d13C, d15N

> 
```

# Taking only the top 3 predictors
```
predictors<-c("CN", "d13C", "d15N")
```

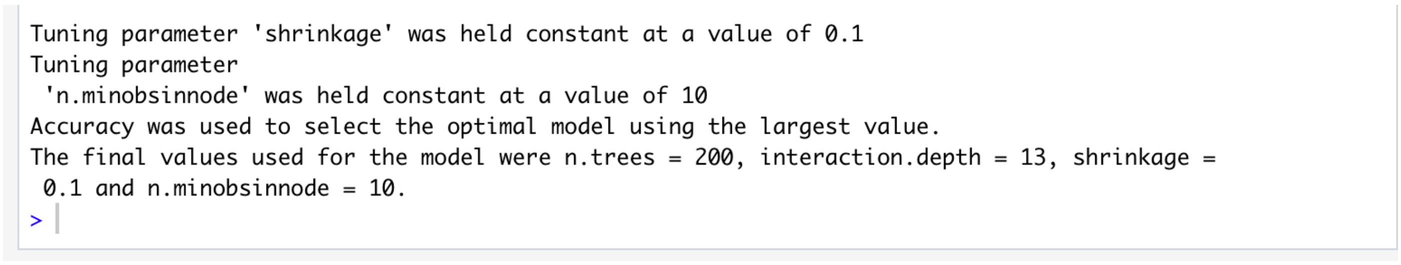
######## Training Models Using Caret ############
```
model_rf_fs<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf')
model_nnet_fs<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet')
model_nb_fs<-train(trainSet[,predictors],trainSet[,outcomeName],method='naive_bayes')
model_gbm_fs<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
```

### Tune the GBM model using tune length = 20 and:
### a) print the model summary and
### b) plot the models.
### Using tuneLength ###

#using tune length
model_gbm_tuned_fs <-
train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',tuneLength=20)
print(model_gbm_tuned_fs)

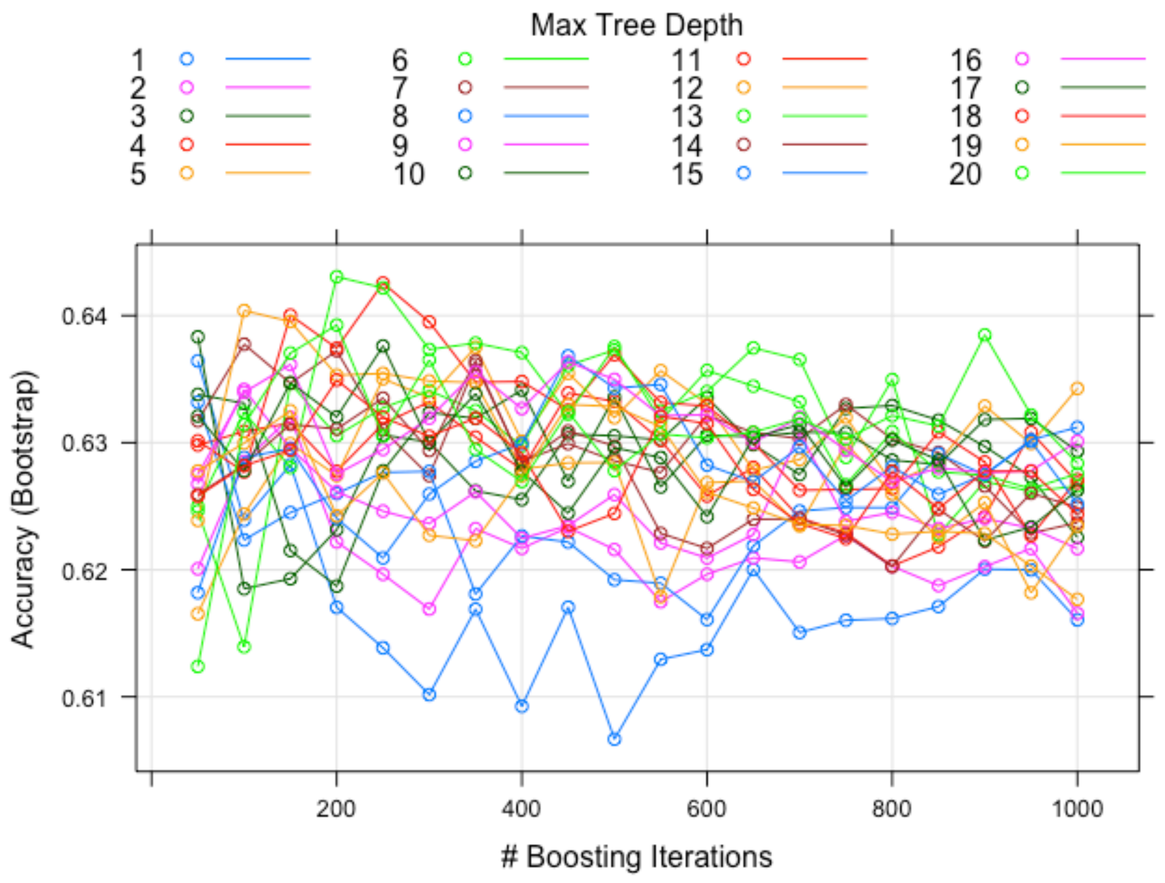


```
> print(model_gbm_tuned_fs)
Stochastic Gradient Boosting

83 samples
 3 predictor
 3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
Resampling results across tuning parameters:
```



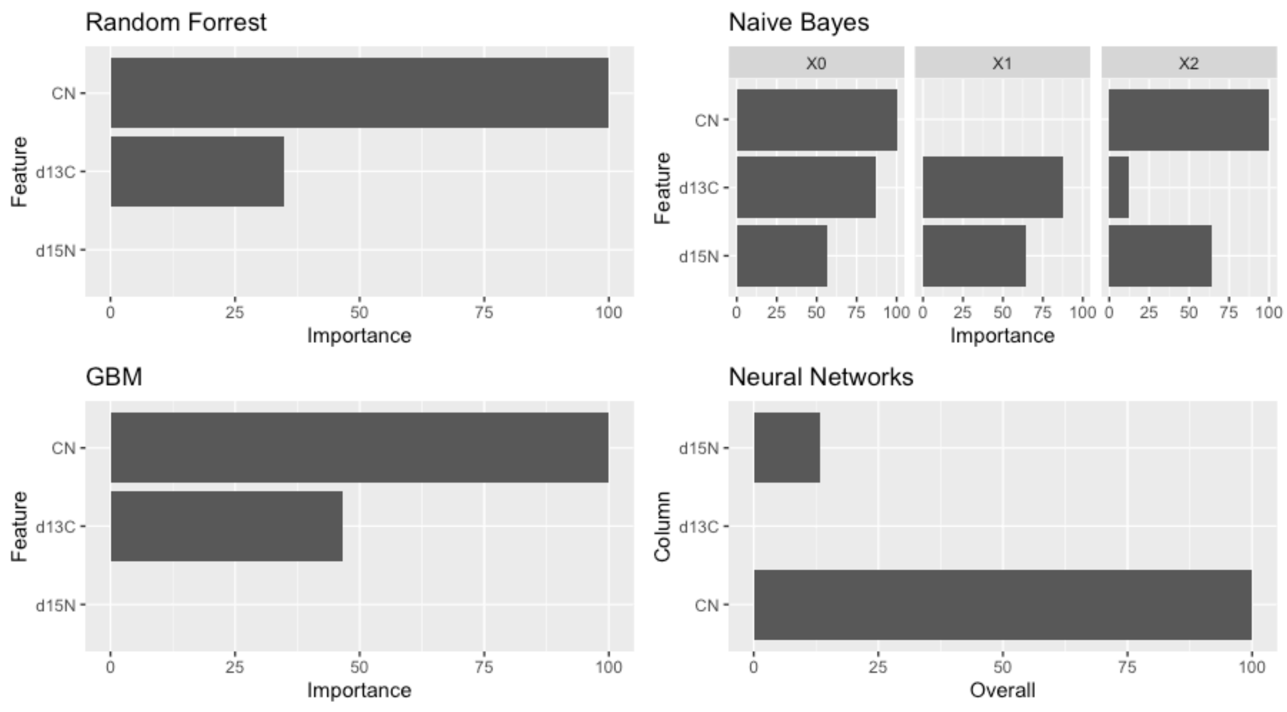
```
Tuning parameter 'shrinkage' was held constant at a value of 0.1
Tuning parameter
 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 200, interaction.depth = 13, shrinkage =
 0.1 and n.minobsinnode = 10.
>
```

# visualize the models
plot(model_gbm_tuned_fs)



### Using GGplot and gridExtra to plot all variable of importance plots into one single plot.

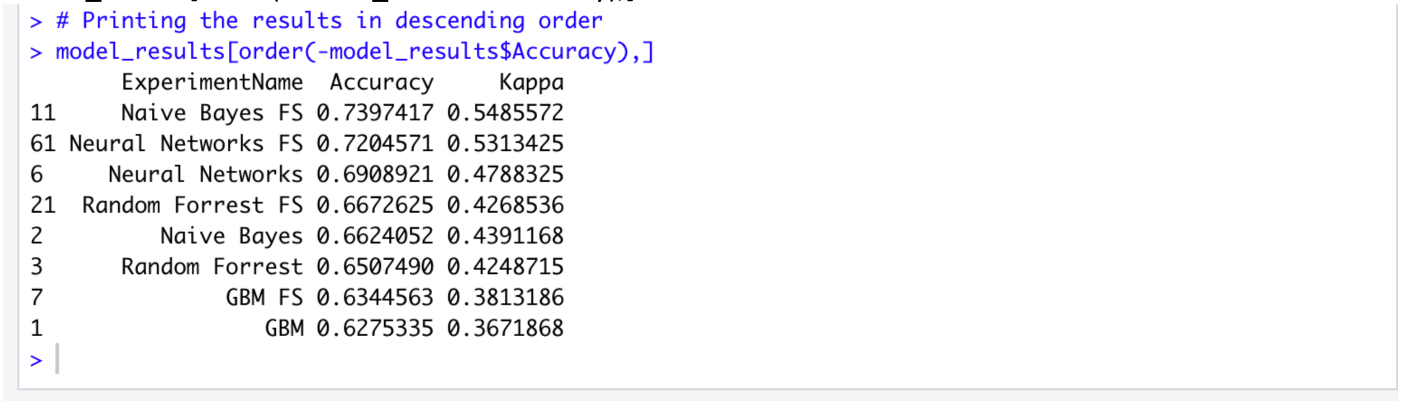
```
p1 <-ggplot(varImp(object=model_rf_fs)) + ggtitle("Random Forrest")
p2 <-ggplot(varImp(object=model_nb_fs)) + ggtitle("Naive Bayes")
p3 <-ggplot(varImp(object=model_gbm_fs)) + ggtitle("GBM")
var_nnet_df_fs <-
data.frame("Column"=row.names(var_nnet_fs$importance)[],varImp(object=model_nnet_fs)$importance)
p4 <- ggplot(var_nnet_df_fs) + geom_col(aes(x=Column,y=Overall)) +  coord_flip() + ggtitle("Neural
Networks")

grid.arrange(p1,p2,p3,p4)
```



10
### b. Create a dataframe that compares the non-feature selected models (the same as on 7)
### and add the best BEST performing models of each (randomforest, neural net, naive bayes and gbm)
### and display the data frame that has the following columns: ExperimentName, accuracy, kappa.
### Sort the data frame by accuracy.

```
newrow <- data.frame("ExperimentName" = "Random Forrest
FS",model_rf_fs$results[row.names(model_rf_fs$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)
newrow = data.frame("ExperimentName" = "Neural Networks
FS",model_nnet_fs$results[row.names(model_nnet_fs$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)
newrow = data.frame("ExperimentName" = "Naive Bayes
FS",model_nb_fs$results[row.names(model_nb_fs$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)
newrow = data.frame("ExperimentName" = "GBM
FS",model_gbm_fs$results[row.names(model_gbm_fs$bestTune),c("Accuracy","Kappa")])
model_results <- rbind(model_results,newrow)

# Printing the results in descending order
```

model_results[order(-model_results$Accuracy),]



```
> # Printing the results in descending order
> model_results[order(-model_results$Accuracy),]
       ExperimentName  Accuracy     Kappa
11     Naive Bayes FS 0.7397417 0.5485572
61 Neural Networks FS 0.7204571 0.5313425
6     Neural Networks 0.6908921 0.4788325
21  Random Forrest FS 0.6672625 0.4268536
2         Naive Bayes 0.6624052 0.4391168
3      Random Forrest 0.6507490 0.4248715
7              GBM FS 0.6344563 0.3813186
1                 GBM 0.6275335 0.3671868
>
```

10
###   c. Which model performs the best?
###   and why do you think this is the case?
###   Can we accurately predict species on this dataset? (10 points)

# After doing the feature selection, the best model now is Naive Bayes with Feature selection.
# Since the number of features has greatly reduced and the target variable has only 3 categories,
# It might be the case that Naive Bayes came first.
# But even here, the accuracy is only 73%, so we might still not be able to correctly predict species.
# With a bigger sample, with the same models we might have better results.