Q1: Set the Species column as the target/outcome and convert it to numeric. (5 points)

Code: scat$Species<-ifelse(scat$Species=='coyote',0,ifelse(scat$Species=='bobcat',1,2))

Output:

	Species	Month	Year	Site	Location	Age	Number	Length	Diam
1	0	January	2012	YOLA	edge	5	2	9.5	25.7
2	0	January	2012	YOLA	edge	3	2	14.0	25.4
3	1	January	2012	YOLA	middle	3	2	9.0	18.8
4	0	January	2012	YOLA	middle	5	2	8.5	18.1
6	0	January	2012	YOLA	edge	5	4	8.0	20.7
7	0	January	2012	YOLA	edge	5	3	9.0	21.2
8	1	January	2012	ANNU	off_edge	1	5	6.0	15.7
9	1	January	2012	ANNU	off_edge	3	7	5.5	21.9
10	1	January	2012	ANNU	off_edge	5	2	11.0	17.5
13	1	January	2012	ANNU	middle	5	1	20.5	18.0
14	2	January	2012	ANNU	middle	3	1	8.0	NA
15	2	January	2012	ANNU	middle	1	1	8.0	12.9

Q2: Remove the Month, Year, Site, Location features. (5 points)

Code:

```
drop_features <- names(scat) %in% c("Month", "Year", "Site","Location")
descriptive_features <- scat[!drop_features]
```

Output:

```
descriptive_fe… 110 obs. of 15 variables
    Species : num 0 0 1 0 0 0 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…
    Taper : num 41.9 37.1 16.5 24.7 20.1 28.5 8.2 19.3 29.1 2…
    TI : num 1.63 1.46 0.88 1.36 0.97 1.34 0.52 0.88 1.66 1.1…
    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 [illegible] .89 7.34 6.…
    CN : num 8.5 11.3 8.1 11.5 10.6 [illegible] 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 ...
```

d13C : num -26.9 -29.6 -28.7 -20.1 -23.2 ...

Q3: Check if any values are null. If there are, impute missing values using KNN. (10 points)

Code :

```
target <- names(descriptive_features) %in% c("Species")
target_features <- descriptive_features[target]
descriptive_features <- descriptive_features[!target]

sum(is.na(descriptive_features))
preProcValues <- preProcess(descriptive_features, method = c("knnImpute","center","scale"))
train_processed <- predict(preProcValues, descriptive_features)

train_processed<-cbind(target_features, train_processed)

sum(is.na(train_processed))
```

Output:

```
Console   Terminal ×   Jobs ×
~/
> #3.Check if any values are null.
> #If there are, impute missing values using KNN. (10 points)
> target <- names(descriptive_features) %in% c("Species")
> target_features <- descriptive_features[target]
> descriptive_features <- descriptive_features[!target]
> sum(is.na(descriptive_features))
[1] 47
> preProcValues <- preProcess(descriptive_features, method = c("knnImpute","center","scale"))
> train_processed <- predict(preProcValues, descriptive_features)
> train_processed<-cbind(target_features, train_processed)
> sum(is.na(train_processed))
[1] 0
>
```

Q4 : Converting every categorical variable to numerical (if needed). (5 points)

Code :

```
dmy <- dummyVars(" ~ .", data = train_processed,fullRank = T)
train_transformed <- data.frame(predict(dmy, newdata = train_processed))

#Converting the dependent variable back to categorical
train_transformed$Species<-as.factor(train_transformed$Species)
```

Output:

```
> str(train_transformed)
'data.frame':   110 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "0","1","2": 1 1 2 1 1 1 2 2 2 2 ...
 $ 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 ...
```

Q5. With a seed of 100, 75% training, 25% testing.

Code :

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

Output:

testSet	27 obs. of 15 variables
train_process…	110 obs. of 15 variables
train_transfo…	110 obs. of 15 variables
trainSet	83 obs. of 15 variables

Q5.a. 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)

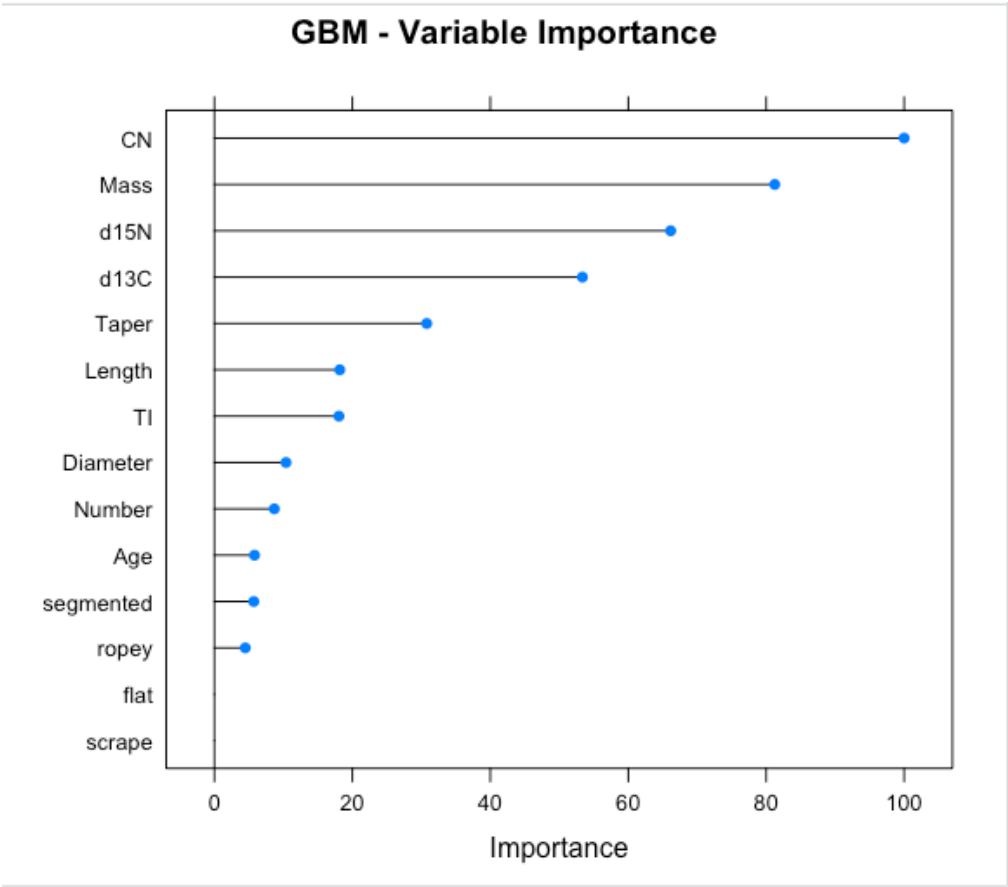
Code and Output

```
# Build the following models: randomforest, neural
# net, naive bayes and GBM.
outcomeName<-'Species'

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

model_gbm<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
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')
```

################################## GBM ##################################
# model summarization
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      0.6277995  0.3620119
  1                  100      0.6123202  0.3374265
  1                  150      0.5904778  0.3018863
  2                   50      0.6116389  0.3357835
  2                  100      0.5923598  0.3132379
  2                  150      0.5879577  0.3045845
  3                   50      0.6227495  0.3570069
  3                  100      0.6016538  0.3262818
  3                  150      0.5970387  0.3179146

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 variable of importance, for the predictions (use the prediction set) display
plot(varImp(object=model_gbm),main="GBM - Variable Importance")



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

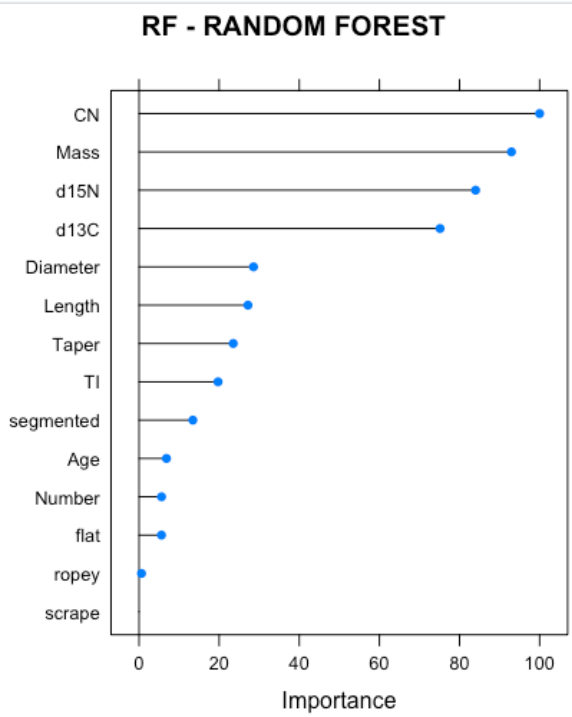
```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  5  1  0
         1  1 13  1
         2  1  0  5

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 : 0.5724067

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.7143   0.9286   0.8333
Specificity            0.9500   0.8462   0.9524
Pos Pred Value         0.8333   0.8667   0.8333
Neg Pred Value         0.9048   0.9167   0.9524
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1852   0.4815   0.1852
Detection Prevalence   0.2222   0.5556   0.2222
Balanced Accuracy      0.8321   0.8874   0.8929
>
```

#gbm_df<-data.frame(Experiment_name="GBM",Accuracy=postResample(pred = predictions, obs = testSet[,outcomeName])[1],Kappa=postResample(pred = predictions, obs = testSet[,outcomeName])[2])

################################## RANDOM FOREST ##############################
# model summarization
print(model_rf)

```
> ################################## RANDOM FOREST ##############################
> # model summarization
> 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    0.6478148  0.3908549
   8    0.6749243  0.4591244
  14    0.6545766  0.4318981

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 8.
>
```

# plot variable of importance, for the predictions (use the prediction set) display
plot(varImp(object=model_rf),main="RF - RANDOM FOREST")



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

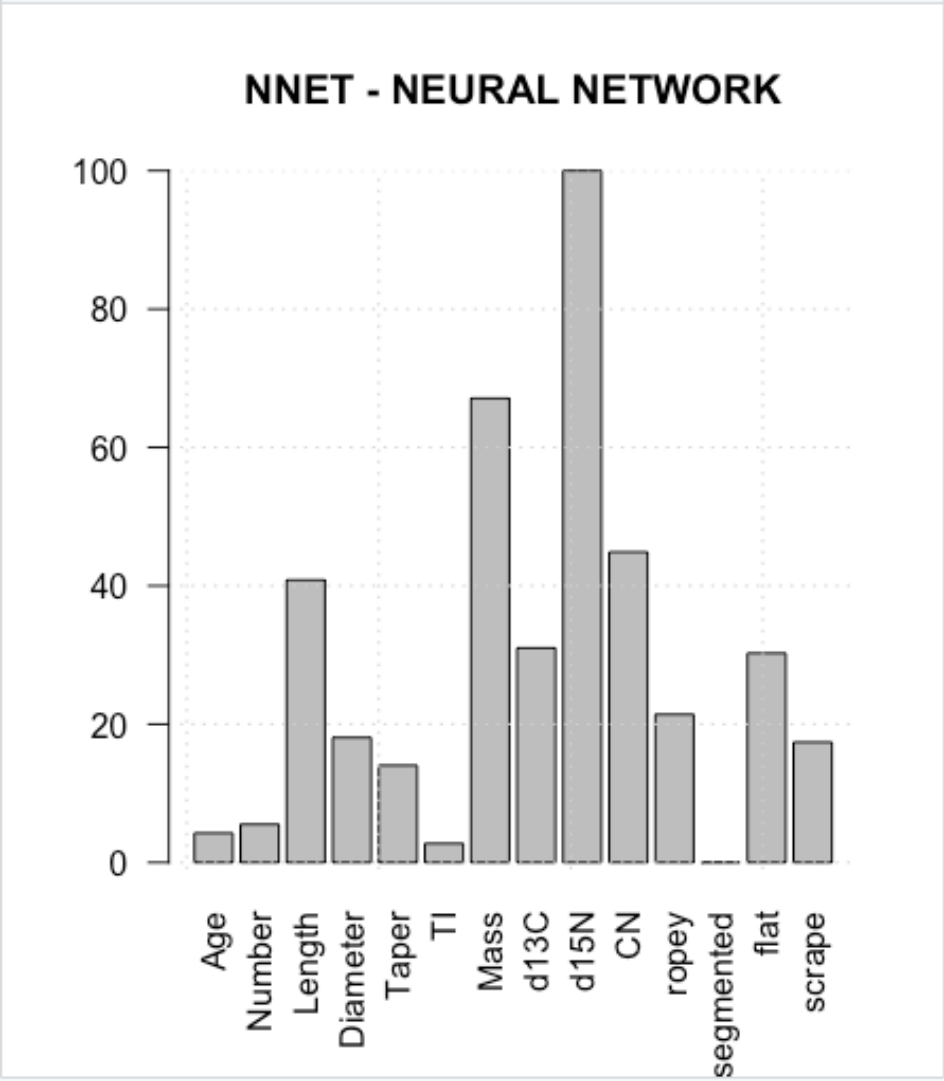
```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  5  1  0
         1  1 13  1
         2  1  0  5

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 : 0.5724067

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.7143   0.9286   0.8333
Specificity            0.9500   0.8462   0.9524
Pos Pred Value         0.8333   0.8667   0.8333
Neg Pred Value         0.9048   0.9167   0.9524
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1852   0.4815   0.1852
Detection Prevalence   0.2222   0.5556   0.2222
Balanced Accuracy      0.8321   0.8874   0.8929
>
```

#rf_df<-data.frame(Experiment_name="RF",Accuracy=postResample(pred = predictions, obs = testSet[,outcomeName])[1],Kappa=postResample(pred = predictions, obs = testSet[,outcomeName])[2])

################################## NEURAL NETWORK ##############################
# model summarization
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  0.5183520  0.2171698
  1     1e-04  0.5552636  0.2745087
  1     1e-01  0.5337311  0.2254746
  3     0e+00  0.6281257  0.3986831
  3     1e-04  0.6314094  0.4008951
  3     1e-01  0.6408351  0.4119614
  5     0e+00  0.6334838  0.4010480
  5     1e-04  0.6348526  0.4103639
  5     1e-01  0.6455855  0.4177539

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were size = 5 and decay = 0.1.
>
```

```
# plot variable of importance, for the predictions (use the prediction set) display
# Since variable importance of the neural network had outcomes for each class and overall
# outcome we are ploting variable importance for only overall outcome.
overall=varImp(model_nnet)
df<-(data.frame(values=overall$importance[1:14,1:0]))
df<-cbind(Predictor = rownames(df),df)
barplot(df$values, names = df$Predictor,las=2, main="NNET - NEURAL NETWORK")
grid(nx=NULL, ny=NULL)
```



```
# confusion matrix
predictions<-predict.train(object=model_nnet,testSet[,predictors],type="raw")
table(predictions)
confusionMatrix(predictions,testSet[,outcomeName])
```

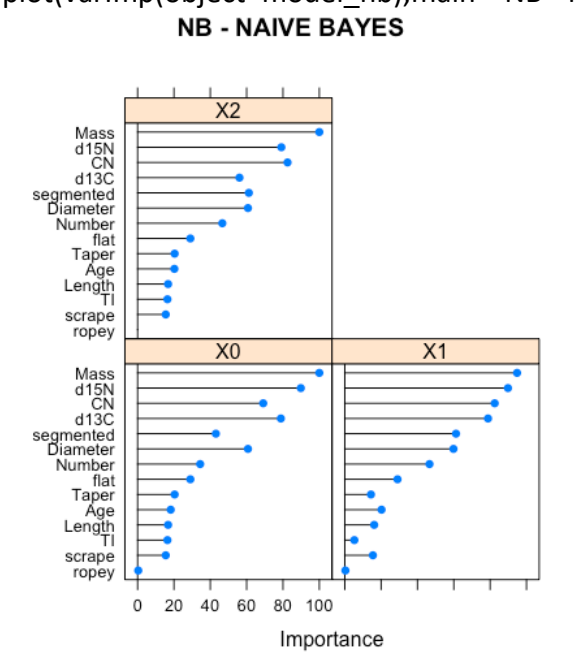
```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  5  1  0
         1  2 12  0
         2  0  1  6

Overall Statistics

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

                  Kappa : 0.7595

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.7143   0.8571   1.0000
Specificity            0.9500   0.8462   0.9524
Pos Pred Value         0.8333   0.8571   0.8571
Neg Pred Value         0.9048   0.8462   1.0000
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1852   0.4444   0.2222
Detection Prevalence   0.2222   0.5185   0.2593
Balanced Accuracy      0.8321   0.8516   0.9762
```

```
#NNET_df<-data.frame(Experiment_name="NNET",Accuracy=postResample(pred = predictions, obs = testSet[,outcomeName])[1],Kappa=postResample(pred = predictions, obs = testSet[,outcomeName])[2])
```

################################## NAIVE BAYES ##################################
# model summarization
print(model_nb)

```
> ################################ NAIVE BAYES ################################
> # model summarization
> 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      0.3969213  0.1448754
   TRUE      0.6154204  0.3762139

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 variable of importance, for the predictions (use the prediction set) display
plot(varImp(object=model_nb),main="NB - NAIVE BAYES")



```
# confusion matrix
predictions<-predict.train(object=model_nb,testSet[,predictors],type="raw")
table(predictions)
confusionMatrix(predictions,testSet[,outcomeName])
#postResample(pred = predictions, obs = testSet[,outcomeName])
```

```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  6  2  0
         1  0 12  1
         2  1  0  5

Overall Statistics

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

                  Kappa : 0.7626

 Mcnemar's Test P-Value : 0.2614641

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.8571   0.8571   0.8333
Specificity            0.9000   0.9231   0.9524
Pos Pred Value         0.7500   0.9231   0.8333
Neg Pred Value         0.9474   0.8571   0.9524
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.2222   0.4444   0.1852
Detection Prevalence   0.2963   0.4815   0.2222
Balanced Accuracy      0.8786   0.8901   0.8929
```

```
#NB_df<-data.frame(Experiment_name="NB",Accuracy=postResample(pred = predictions, obs
= testSet[,outcomeName])[1],Kappa=postResample(pred = predictions, obs =
testSet[,outcomeName])[2])
```

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)

Code:

```
gbm_df<data.frame(Experiment_name="GBM",Accuracy=max(model_gbm$results$Accuracy),
Kappa=max(model_gbm$results$Kappa))
rf_df<data.frame(Experiment_name="RF",Accuracy=max(model_rf$results$Accuracy),
Kappa=max(model_rf$results$Kappa))
NNET_df<data.frame(Experiment_name="NNET",
Accuracy=max(model_nnet$results$Accuracy),Kappa=max(model_nnet$results$Kappa))
NB_df<data.frame(Experiment_name="NB",
Accuracy=max(model_nb$results$Accuracy),Kappa=max(model_nb$results$Kappa))

Compare_models_df <- rbind(gbm_df,rf_df,NNET_df,NB_df)

Compare_models_df<-Compare_models_df[order(Compare_models_df$Accuracy),]
rownames(Compare_models_df) <- NULL
Compare_models_df
```

Output:

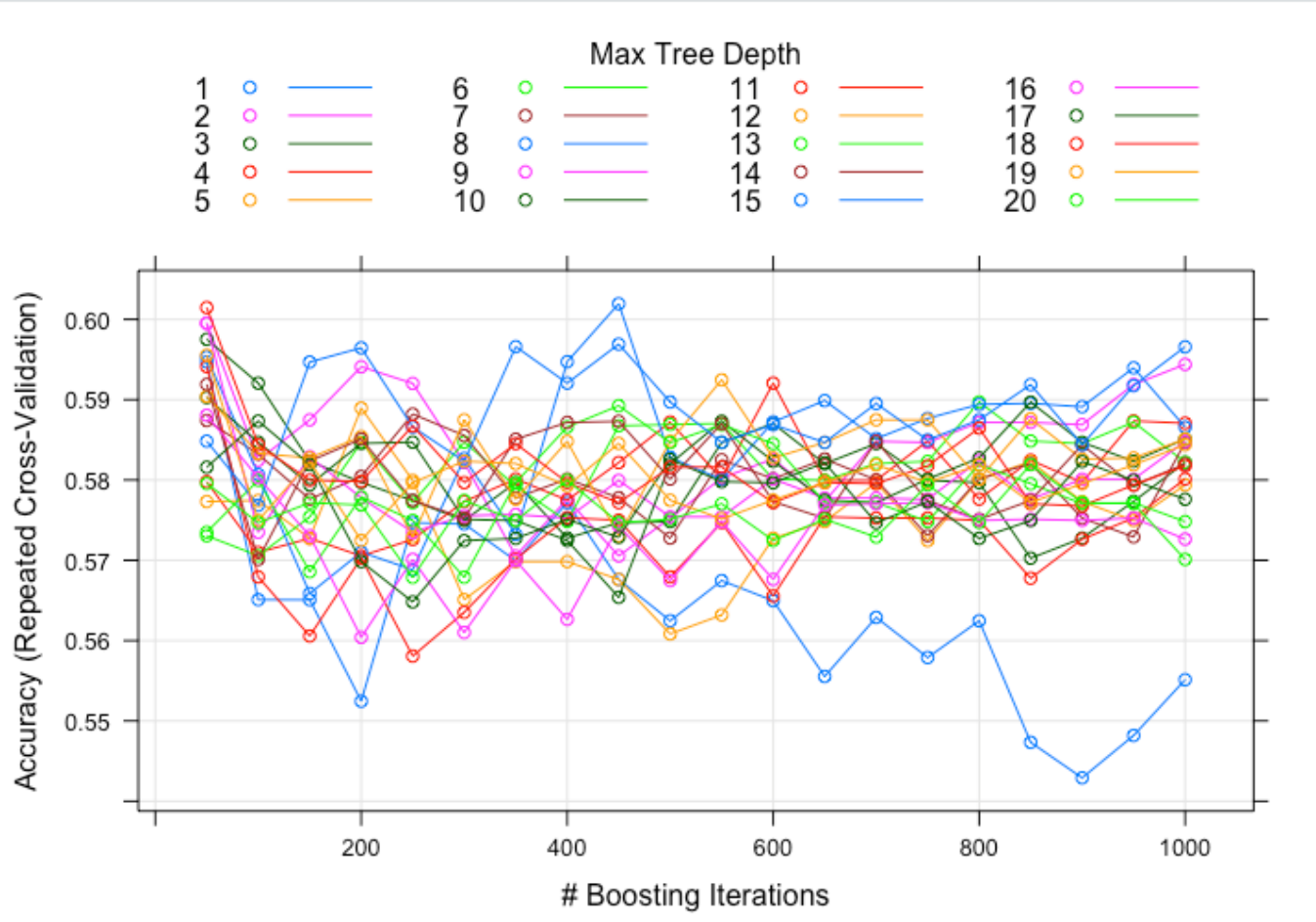
```
> Compare_models_df
  Experiment_name  Accuracy     Kappa
1              NB 0.6154204 0.3762139
2             GBM 0.6277995 0.3620119
3            NNET 0.6455855 0.4177539
4              RF 0.6749243 0.4591244
>
```

Q7.Tune the GBM model using tune length = 20 and: a) print the model summary and b) plot the models. (20 points)

Code:

```
fitControl <- trainControl(  method = "repeatedcv",  number = 5,  repeats = 5)
model_gbm_tuned<-
train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',trControl=fitControl,tuneLen
gth=20)
plot(model_gbm_tuned)
```

Output:



Q8. Using GGplot and gridExtra to plot all variable of importance plots into one single plot.

(10 points)

Code:

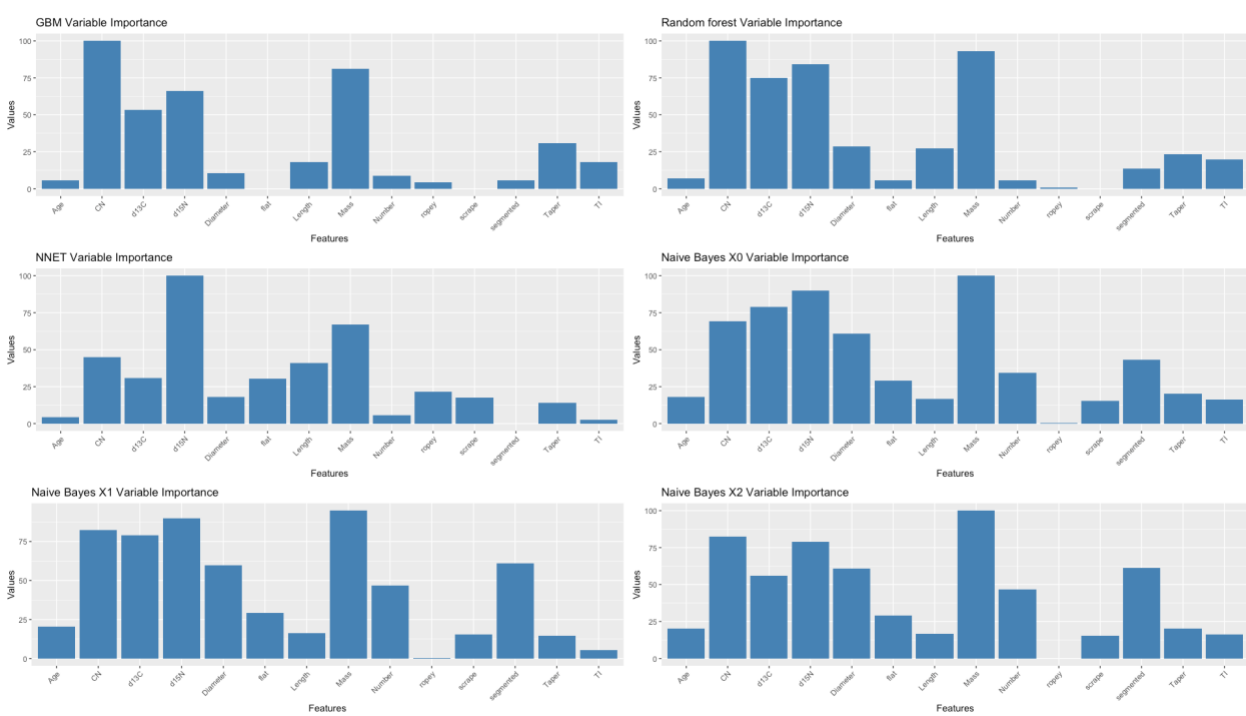
```
#creating DF for all the class of the Naive Bayes
nb_overall=varImp(object=model_nb)
nb_var_df=data.frame(features=rownames(nb_overall$importance),NB_X0=nb_overall$import
ance$X0,NB_X1=nb_overall$importance$X1,NB_X2=nb_overall$importance$X2)
print("Naive Bayes variable importance Data frame")
nb_var_df

#creating DF for all the class of the Naive Bayes
nnet_overall=varImp(model_nnet)
nnet_var_df<-(data.frame(NNET_values=nnet_overall$importance[1:14,1:0]))
nnet_var_df<-cbind(features = rownames(nnet_var_df),nnet_var_df)

print("Neural Network variable importance Data frame")
nnet_var_df

#Merging in All_model_varImp Data frame
All_model_varImp <- merge(nb_var_df, nnet_var_df, by.x = "features")
print("All Model variable importance Data frame")
All_model_varImp

#creating DF for all the class of the Random Forest
rf_overall=(varImp(object=model_rf))
rf_var_df=data.frame(rf_values=rf_overall$importance)
rf_var_df<-cbind(features = rownames(rf_var_df),RF_values=rf_var_df)

print("Randome Forest variable importance Data frame")
rf_var_df

#Merging in All_model_varImp Data frame
All_model_varImp <- merge(All_model_varImp, rf_var_df, by.x = "features")
colnames(All_model_varImp)[which(names(All_model_varImp) == "Overall")] <- "rf_values"
print("All Model variable importance Data frame")
All_model_varImp

#creating DF for all the class of the GBM
gbm_overall=(varImp(object=model_gbm))
```

```
gbm_var_df=data.frame(gbm_values=gbm_overall$importance)
gbm_var_df<-cbind(features = rownames(gbm_var_df),GBM_values=gbm_var_df)
print("GBM variable importance Data frame")
gbm_var_df

#Merging in All_model_varImp Data frame
All_model_varImp <- merge(All_model_varImp, gbm_var_df, by.x = "features")
colnames(All_model_varImp)[which(names(All_model_varImp) == "Overall")] <- "gbm_values"
print("All model variable importance Data frame")
All_model_varImp


# Plot for NNET
nnet_plot<-ggplot(data=All_model_varImp, aes(y=All_model_varImp$NNET_values,
x=All_model_varImp$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("NNET Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot for Naive Bayes class 0
NB_X0_plot<-ggplot(data=All_model_varImp, aes(y=All_model_varImp$NB_X0,
x=All_model_varImp$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Naive Bayes X0 Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot for Naive Bayes class 1
NB_X1_plot<-ggplot(data=All_model_varImp, aes(y=All_model_varImp$NB_X1,
x=All_model_varImp$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Naive Bayes X1 Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot for Naive Bayes class 2
NB_X2_plot<-ggplot(data=All_model_varImp, aes(y=All_model_varImp$NB_X2,
x=All_model_varImp$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Naive Bayes X2 Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot for Random Forest
rf_plot<-ggplot(data=All_model_varImp, aes(y=All_model_varImp$rf_values,
x=All_model_varImp$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Random forest Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot for GBM
```

```
gbm_plot<-ggplot(data=All_model_varImp, aes(y=All_model_varImp$gbm_values,
x=All_model_varImp$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("GBM Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Making a grid plot
gridPlot=grid.arrange(gbm_plot,rf_plot,nnet_plot,NB_X0_plot,NB_X1_plot,NB_X2_plot, ncol= 2
,nrow=3)

gridPlot
```

Output:

```
> gridPlot
TableGrob (3 x 2) "arrange": 6 grobs
  z     cells    name           grob
1 1 (1-1,1-1) arrange gtable[layout]
2 2 (1-1,2-2) arrange gtable[layout]
3 3 (2-2,1-1) arrange gtable[layout]
4 4 (2-2,2-2) arrange gtable[layout]
5 5 (3-3,1-1) arrange gtable[layout]
6 6 (3-3,2-2) arrange gtable[layout]
```



Q9.Which model performs the best? and why do you think this is the case? Can we accurately predict species on this dataset? (10 points)

Ans : From the above results of model comparison we can see random forest has maximum accuracy hence we can expect Random forest to perform better than others.RF performs best here becasue the provision of feature importance is much more reliable when compared to any other model. Also random forest deals with multiple decision tress making it more accurate.
It can predict Species column 7 out of 10 times approximately as it is evident from the accuracy

10.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)

Code & Output:

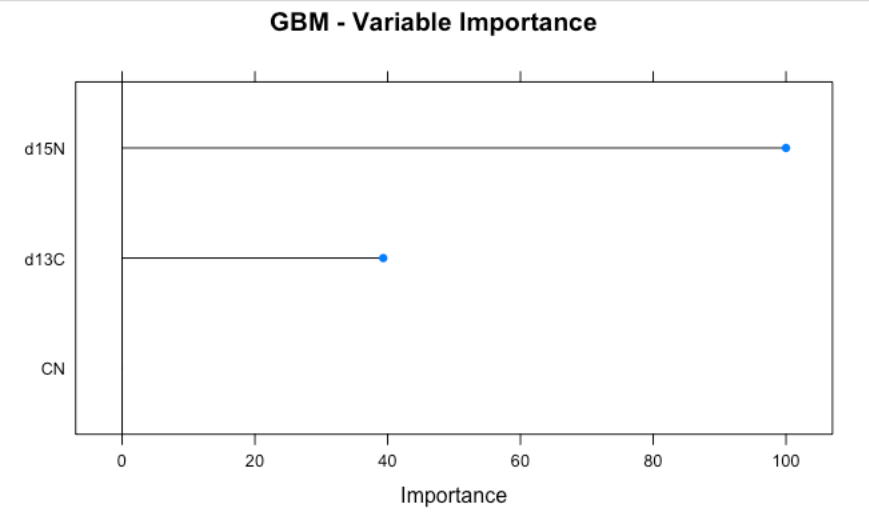
```
#Feature selection using rfe in caret
control <- rfeControl(functions = rfFuncs,
                      method = "repeatedcv",
                      repeats = 3,
                      verbose = FALSE)


Loan_Pred_Profile <- rfe(trainSet[,predictors], trainSet[,outcomeName],rfeControl = control)
Loan_Pred_Profile
```

```
> Loan_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
         4   0.7098 0.5041     0.1233  0.2325        *
         8   0.6934 0.4689     0.1432  0.2586
        14   0.6766 0.4365     0.1401  0.2566

The top 4 variables (out of 4):
   CN, d15N, d13C, Mass

>
```

```
predictors<-c("CN", "d15N", "d13C")

# For example, to apply, GBM, Random forest, Neural net:
model_gbm_featured<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
model_rf_featured<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf')
model_nnet_featured<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet')
model_nb_featured<-
train(trainSet[,predictors],trainSet[,outcomeName],method='naive_bayes')
################################## GBM ##################################

#model summarization
print(model_gbm_featured)
```

```
> print(model_gbm_featured)
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:

  interaction.depth  n.trees  Accuracy   Kappa
  1                   50      0.6224450  0.3564022
  1                  100      0.6035718  0.3277816
  1                  150      0.6109481  0.3383764
  2                   50      0.6242470  0.3632077
  2                  100      0.6173890  0.3545992
  2                  150      0.6089248  0.3406890
  3                   50      0.6155181  0.3467348
  3                  100      0.6332798  0.3759422
  3                  150      0.6127140  0.3467443

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 = 100, interaction.depth = 3,
 shrinkage = 0.1 and n.minobsinnode = 10.
>
```

```
#Plotting Varianle importance for GBM
plot(varImp(object=model_gbm_featured),main="GBM - Variable Importance")
predictions<-predict.train(object=model_gbm_featured,testSet[,predictors],type="raw")
table(predictions)
```



#Confusion Matrix and Statistics
confusionMatrix(predictions,testSet[,outcomeName])

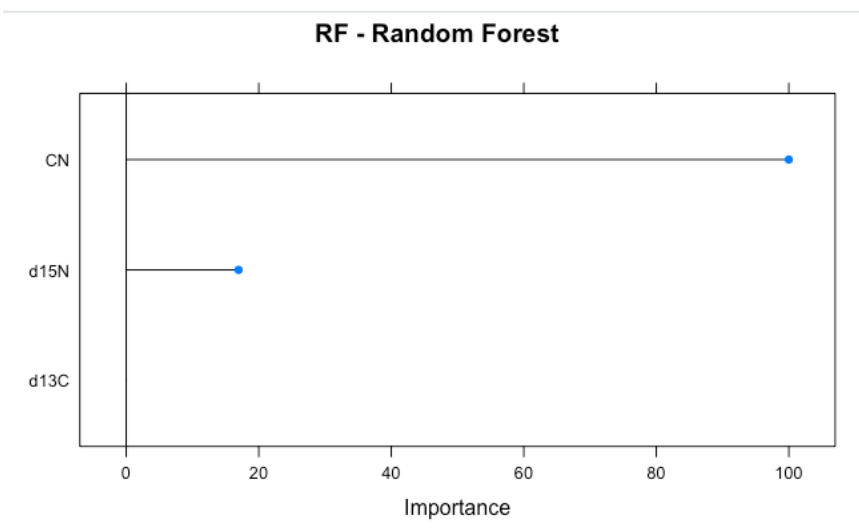
```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction 0 1 2
         0 4 3 2
         1 0 9 1
         2 3 2 3

Overall Statistics

               Accuracy : 0.5926
                 95% CI : (0.388, 0.7761)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.2827

                  Kappa : 0.3787

 Mcnemar's Test P-Value : 0.3165

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.5714   0.6429   0.5000
Specificity            0.7500   0.9231   0.7619
Pos Pred Value         0.4444   0.9000   0.3750
Neg Pred Value         0.8333   0.7059   0.8421
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1481   0.3333   0.1111
Detection Prevalence   0.3333   0.3704   0.2963
Balanced Accuracy      0.6607   0.7830   0.6310
```

################################## Random Forest ##################################

# model summarization
print(model_rf_featured)

```
> print(model_rf_featured)
Random Forest

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:

  mtry  Accuracy   Kappa
  2     0.6794414  0.4633116
  3     0.6764612  0.4596168

Accuracy was used to select the optimal model using
 the largest value.
The final value used for the model was mtry = 2.
>
```

# plot variable of importance, for the predictions (use the prediction set) display
plot(varImp(object=model_rf_featured),main="RF - Random Forest")



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

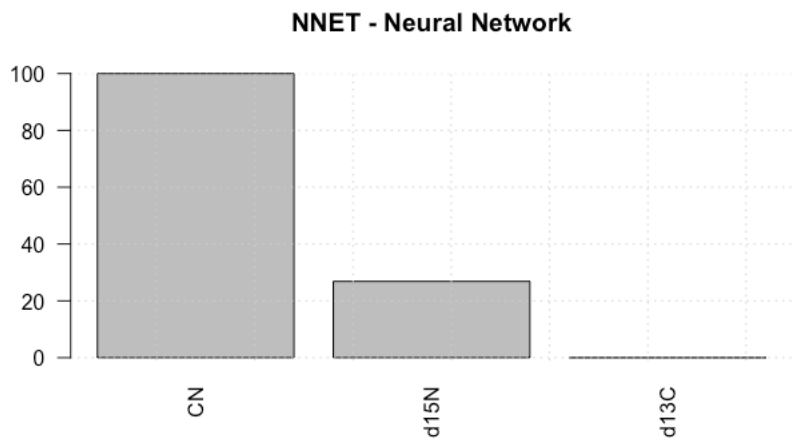
```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  5  3  0
         1  1 11  2
         2  1  0  4

Overall Statistics

               Accuracy : 0.7407
                 95% CI : (0.5372, 0.8889)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.01571

                  Kappa : 0.5772

 Mcnemar's Test P-Value : 0.26146

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.7143   0.7857   0.6667
Specificity            0.8500   0.7692   0.9524
Pos Pred Value         0.6250   0.7857   0.8000
Neg Pred Value         0.8947   0.7692   0.9091
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1852   0.4074   0.1481
Detection Prevalence   0.2963   0.5185   0.1852
Balanced Accuracy      0.7821   0.7775   0.8095
```

################################## Neural Network ##################################

# model summarization
print(model_nnet_featured)

```
> print(model_nnet_featured)
Neural Network

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:

  size  decay  Accuracy   Kappa
  1     0e+00  0.6133420  0.3258754
  1     1e-04  0.6507593  0.4007538
  1     1e-01  0.6493871  0.3942597
  3     0e+00  0.6359929  0.3937693
  3     1e-04  0.6638237  0.4349790
  3     1e-01  0.7122889  0.5080762
  5     0e+00  0.6375987  0.3951738
  5     1e-04  0.6398754  0.4053904
  5     1e-01  0.7080390  0.5023372

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.
>
```

```
# plot variable of importance, for the predictions (use the prediction set) display
overall=varImp(model_nnet_featured)
df_featured<-(data.frame(overall=overall$importance[1:3,1:0]))
df_featured<-cbind(Predictor = rownames(df_featured),df_featured)
barplot(df_featured$overall, names = df_featured$Predictor,las=2, main='NNET - Neural
Network')
grid(nx=NULL, ny=NULL)
```



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

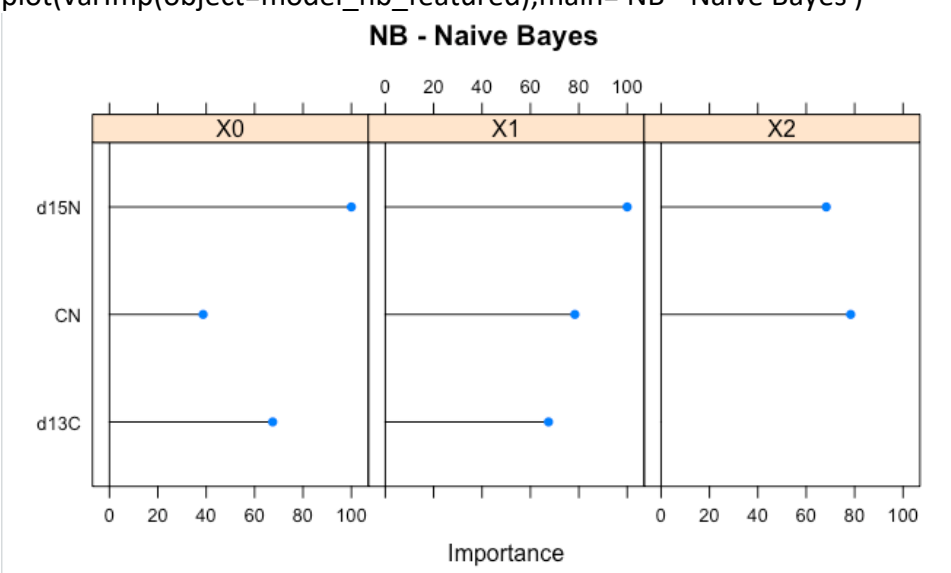
```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  5  1  1
         1  2 13  2
         2  0  0  3

Overall Statistics

               Accuracy : 0.7778
                 95% CI : (0.5774, 0.9138)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.005195

                  Kappa : 0.6179

 Mcnemar's Test P-Value : 0.343030

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.7143   0.9286   0.5000
Specificity            0.9000   0.6923   1.0000
Pos Pred Value         0.7143   0.7647   1.0000
Neg Pred Value         0.9000   0.9000   0.8750
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1852   0.4815   0.1111
Detection Prevalence   0.2593   0.6296   0.1111
Balanced Accuracy      0.8071   0.8104   0.7500
```

################################## Naive Bayes ##################################

# model summarization
print(model_nb_featured)

```
> print(model_nb_featured)
Naive Bayes

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:

  usekernel  Accuracy   Kappa
  FALSE      0.7516721  0.5506309
   TRUE      0.7202285  0.5128875

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 = FALSE and adjust = 1.
```

# plot variable of importance, for the predictions (use the prediction set) display
plot(varImp(object=model_nb_featured),main='NB - Naive Bayes')



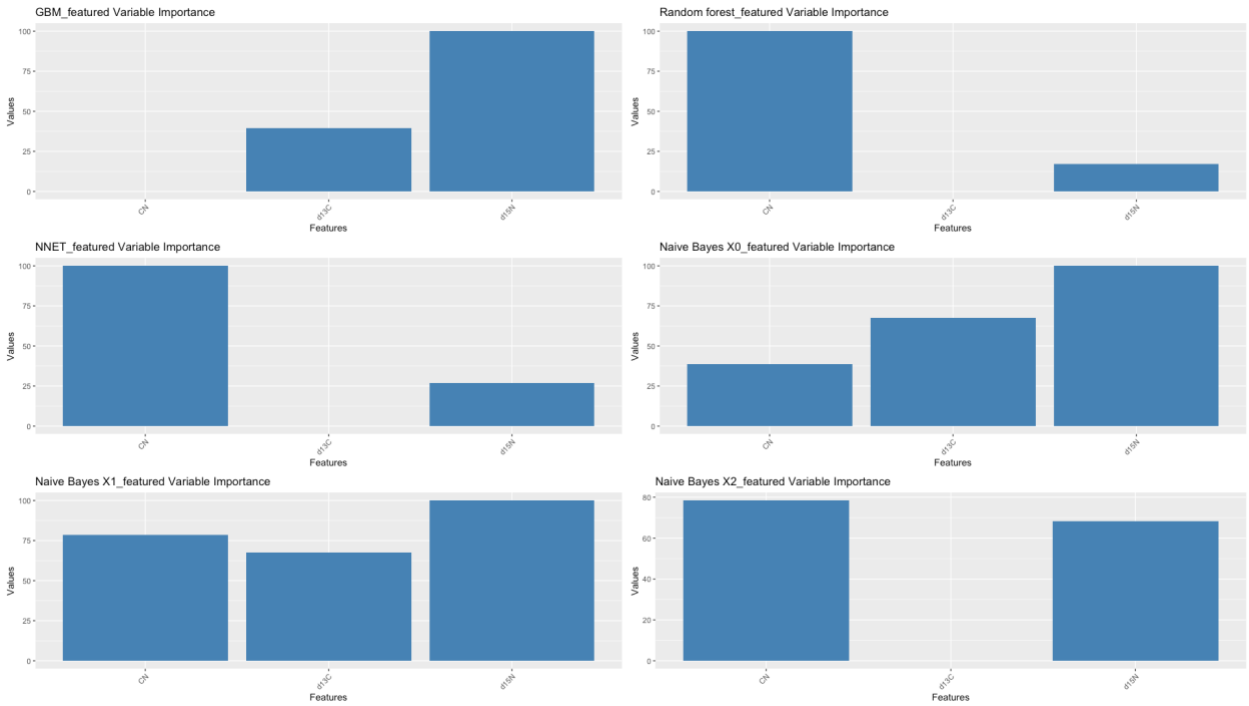
# Confusion Matrix and Statistics
predictions<-predict.train(object=model_nb_featured,testSet[,predictors],type="raw")
table(predictions)
confusionMatrix(predictions,testSet[,outcomeName])

```
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2
         0  4  1  0
         1  2 13  3
         2  1  0  3

Overall Statistics

               Accuracy : 0.7407
                 95% CI : (0.5372, 0.8889)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.01571

                  Kappa : 0.5478

 Mcnemar's Test P-Value : 0.22765

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.5714   0.9286   0.5000
Specificity            0.9500   0.6154   0.9524
Pos Pred Value         0.8000   0.7222   0.7500
Neg Pred Value         0.8636   0.8889   0.8696
Prevalence             0.2593   0.5185   0.2222
Detection Rate         0.1481   0.4815   0.1111
Detection Prevalence   0.1852   0.6667   0.1481
Balanced Accuracy      0.7607   0.7720   0.7262
>
```

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

#Tuning GBM
fitControl <- trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 5)
model_gbm_tuned<-
train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',trControl=fitControl,tuneLen
gth=20)
plot(model_gbm_tuned)

#creating DF for all the class of the Naive Bayes
nb_overall_featured=varImp(object=model_nb_featured)
nb_var_df_featured=data.frame(features=rownames(nb_overall_featured$importance),NB_X0
_featured=nb_overall_featured$importance$X0,NB_X1_featured=nb_overall_featured$import
ance$X1,NB_X2_featured=nb_overall_featured$importance$X2)
print("Neural Network variable importance Data frame")
nb_var_df_featured

#creating DF for all the class of the Neural Network
nnet_overall_featured=varImp(model_nnet_featured)
nnet_var_df_featured<-
(data.frame(NNET_values_featured=nnet_overall_featured$importance[1:3,1:0]))
nnet_var_df_featured<-cbind(features =
rownames(nnet_var_df_featured),nnet_var_df_featured)
print("Neural Network variable importance Data frame")
nnet_var_df_featured

All_model_varImp_featured <- merge(nb_var_df_featured, nnet_var_df_featured, by.x =
"features")
All_model_varImp_featured

#creating DF for all the class of the Random forest
rf_overall_featured=(varImp(object=model_rf_featured))
rf_var_df_featured=data.frame(rf_values_featured=rf_overall_featured$importance)
rf_var_df_featured<-cbind(features =
rownames(rf_var_df_featured),RF_values=rf_var_df_featured)
print("Random forest variable importance Data frame")
rf_var_df_featured

All_model_varImp_featured <- merge(All_model_varImp_featured, rf_var_df_featured, by.x =
"features")
```

```
colnames(All_model_varImp_featured)[which(names(All_model_varImp_featured) ==
"Overall")] <- "rf_values_featured"

All_model_varImp_featured
#creating DF for all the class of the GBM
gbm_overall_featured=(varImp(object=model_gbm_featured))
gbm_var_df_featured=data.frame(gbm_values_featured=gbm_overall_featured$importance)
gbm_var_df_featured<-cbind(features =
rownames(gbm_var_df_featured),GBM_values=gbm_var_df_featured)
print("Random forest variable importance Data frame")
gbm_var_df_featured


All_model_varImp_featured <- merge(All_model_varImp_featured, gbm_var_df_featured, by.x
= "features")
colnames(All_model_varImp_featured)[which(names(All_model_varImp_featured) ==
"Overall")] <- "gbm_values_featured"

All_model_varImp_featured

#preparing grid plots
nnet_plot_featured<-ggplot(data=All_model_varImp_featured,
aes(y=All_model_varImp_featured$NNET_values_featured,
x=All_model_varImp_featured$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("NNET_featured Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

NB_X0_plot_featured<-ggplot(data=All_model_varImp_featured,
aes(y=All_model_varImp_featured$NB_X0_featured, x=All_model_varImp_featured$features))
+
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Naive Bayes X0_featured Variable
Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

NB_X1_plot_featured<-ggplot(data=All_model_varImp_featured,
aes(y=All_model_varImp_featured$NB_X1_featured, x=All_model_varImp_featured$features))
+
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Naive Bayes X1_featured Variable
Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

NB_X2_plot_featured<-ggplot(data=All_model_varImp_featured,
aes(y=All_model_varImp_featured$NB_X2_featured, x=All_model_varImp_featured$features))
+
```

```
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Naive Bayes X2_featured Variable
Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

rf_plot_featured<-ggplot(data=All_model_varImp_featured,
aes(y=All_model_varImp_featured$rf_values_featured,
x=All_model_varImp_featured$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("Random forest_featured Variable
Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

gbm_plot_featured<-ggplot(data=All_model_varImp_featured,
aes(y=All_model_varImp_featured$gbm_values_featured,
x=All_model_varImp_featured$features)) +
  geom_bar(stat="identity",fill="steelblue")+ggtitle("GBM_featured Variable Importance") +
  xlab("Features") + ylab("Values") + theme(axis.text.x = element_text(angle = 45, hjust = 1))

gridPlot_featured=grid.arrange(gbm_plot_featured,rf_plot_featured,nnet_plot_featured,NB_X
0_plot_featured,NB_X1_plot_featured,NB_X2_plot_featured, ncol= 2 ,nrow=3)


gridPlot_featured
```



Q10.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. (40 points)

Code :

```
gbm_df_featured<-
data.frame(Experiment_name="GBM_featured",Accuracy=max(model_gbm_featured$results$
Accuracy),Kappa=max(model_gbm_featured$results$Kappa))
rf_df_featured<-
data.frame(Experiment_name="RF_featured",Accuracy=max(model_rf_featured$results$Accur
acy),Kappa=max(model_rf_featured$results$Kappa))
NNET_df_featured<-
data.frame(Experiment_name="NNET_featured",Accuracy=max(model_nnet_featured$results$
Accuracy),Kappa=max(model_nnet_featured$results$Kappa))
NB_df_featured<-
data.frame(Experiment_name="NB_featured",Accuracy=max(model_nb_featured$results$Acc
uracy),Kappa=max(model_nb_featured$results$Kappa))

Compare_models_df_featured <-
rbind(gbm_df_featured,rf_df_featured,NNET_df_featured,NB_df_featured)

Compare_models_df_featured<-
Compare_models_df_featured[order(Compare_models_df_featured$Accuracy),]

Compare_models_df_featured

Compare_models_all<-rbind(Compare_models_df_featured,Compare_models_df)
Compare_models_all

Compare_models_all<-Compare_models_all[order(Compare_models_all$Accuracy),]
rownames(Compare_models_all) <- NULL
Compare_models_all
```

Output:

```
> Compare_models_all
  Experiment_name  Accuracy     Kappa
1              NB 0.6154204 0.3762139
2             GBM 0.6277995 0.3620119
3    GBM_featured 0.6332798 0.3759422
4            NNET 0.6455855 0.4177539
5              RF 0.6749243 0.4591244
6     RF_featured 0.6794414 0.4633116
7   NNET_featured 0.7122889 0.5080762
8     NB_featured 0.7516721 0.5506309
```

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)

Ans : From the above results of model comparison we can see Naïve Bayes Feature selected has maximum accuracy hence we can expect Naïve Bayes Feature selected to perform better than others model. Naïve Bayes Feature selected performs best here because Naïve Bayes works best with continuous data and when the data is small. So if you need something fast we can easily apply and get better results.
It can predict Species column 8 out of 10 times approximately as it is evident from the accuracy