Set the Species column as the target/outcome and convert it to numeric.

```
colnames(scat)[1] = "outcome"
scat$outcome = as.numeric(as.factor(scat$outcome))
```

Remove the Month, Year, Site, Location features.

```
f = subset(scat, select = -c(Month,Year, Site, Location))
```

Check if any values are null. If there are, impute missing values using KNN.

```
sum(is.na(df))
impute <- preProcess(df, method = c("knnImpute","center","scale"))
newdf <- predict(impute, df)
sum(is.na(newdf))
```

Converting every categorical variable to numerical (if needed).

This is not needed as all the variables are numeric.

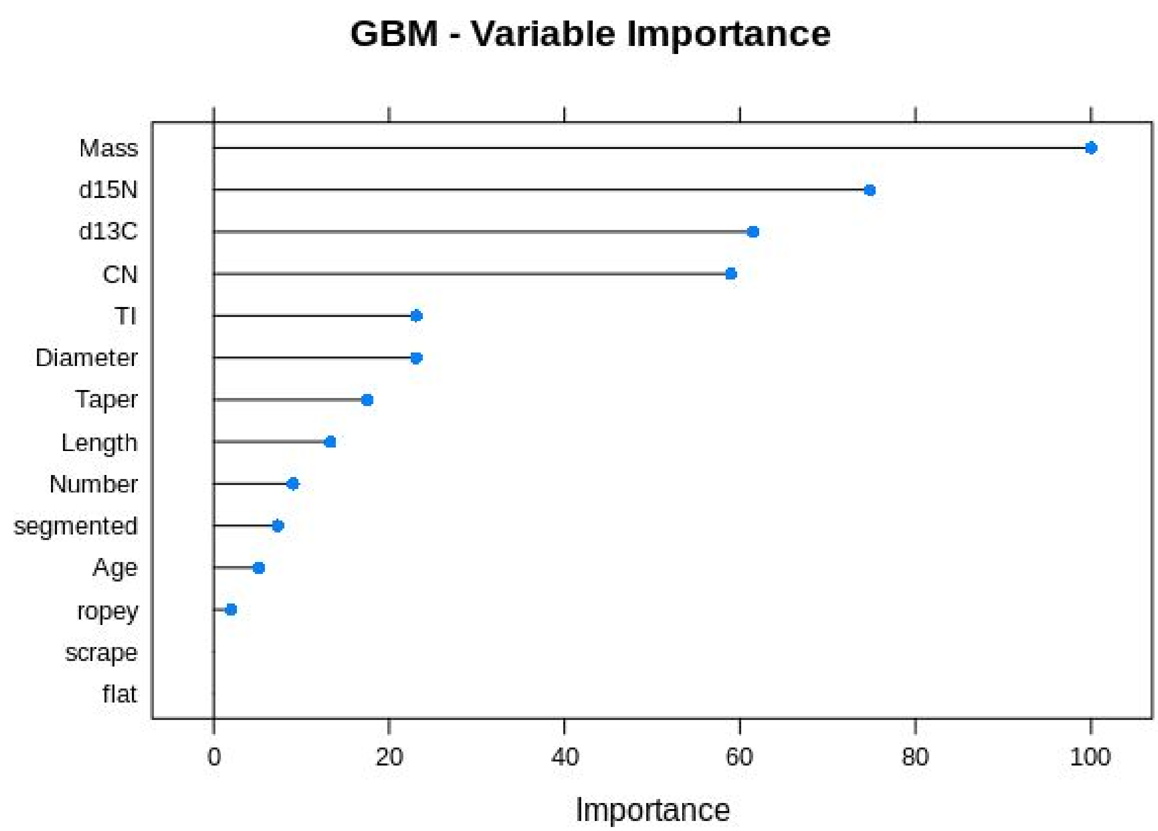
With a seed of 100, 75% training, 25% testing . Build the following models: randomforest, neural
net, naive bayes and GBM.

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

```
#Spliting training set into two parts based on outcome: 75% and 25%
set.seed(100)
index <- createDataPartition(newdf$outcome, p=0.75, list=FALSE)
trainSet <- newdf[ index,]
testSet <- newdf[-index,]

predictors<-c("Age", "Number", "Length", "Diameter", "Taper", "TI", "Mass", "d13C", "d15N",
"CN", "ropey", "segmented", "flat", "scrape")
outcomeName <- "outcome"

#####Models#####
##GBM##
#Model
model_gbm<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
#Summary
print(model_gbm)
#Plot
plot(varImp(object=model_gbm),main="GBM - Variable Importance")
#Predictions
predictions_gbm<-predict.train(object=model_gbm,testSet[,predictors],type="raw")
```

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

```
#Stochastic Gradient Boosting
#
#83 samples
#14 predictors
# 3 classes: '1', '2', '3'
#
#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      0.7009397  0.5036373
#  1                  100      0.6971025  0.4959233
#  1                  150      0.6998312  0.5025281
#  2                   50      0.6902376  0.4845022
#  2                  100      0.7034838  0.5079863
#  2                  150      0.6929869  0.4903189
#  3                   50      0.6927617  0.4879024
#  3                  100      0.6998958  0.5003638
#  3                  150      0.6886424  0.4800675
#
#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 = 2, shrinkage
=
# 0.1 and n.minobsinnode = 10.
```



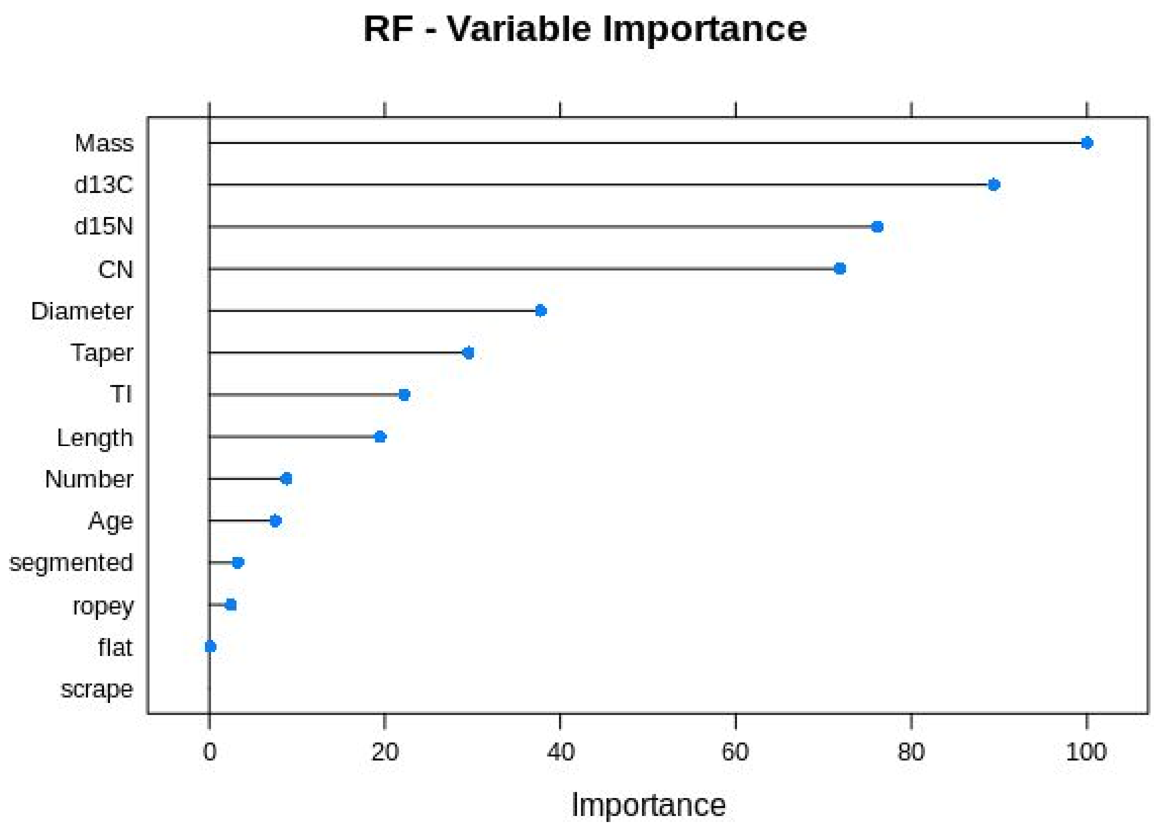
Confusion Matrix and Statistics

```
          Reference
Prediction 1 2 3
         1 9 2 1
         2 3 4 4
         3 2 1 1

Overall Statistics

               Accuracy : 0.5185
                 95% CI : (0.3195, 0.7133)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.5770

                  Kappa : 0.237

 Mcnemar's Test P-Value : 0.5062

Statistics by Class:
```

```
                    Class: 1 Class: 2 Class: 3
Sensitivity           0.6429   0.5714  0.16667
Specificity           0.7692   0.6500  0.85714
Pos Pred Value        0.7500   0.3636  0.25000
Neg Pred Value        0.6667   0.8125  0.78261
Prevalence            0.5185   0.2593  0.22222
Detection Rate        0.3333   0.1481  0.03704
Detection Prevalence  0.4444   0.4074  0.14815
Balanced Accuracy     0.7060   0.6107  0.51190
```

```
##Random Forest##
#Model
model_rf<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf')
#Summary
print(model_rf)
#Plot
plot(varImp(object=model_rf),main="RF - Variable Importance")
#Predictions
predictions_rf<-predict.train(object=model_rf,testSet[,predictors],type="raw")
table(predictions_rf)
#Confusion Matrix and Statistics
confusionMatrix(predictions_rf,testSet[,outcomeName])
```

```
#Random Forest

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

#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    0.6837039  0.4646791
#   8    0.6848865  0.4827960
#  14    0.6732672  0.4632360

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



Confusion Matrix and Statistics

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

Overall Statistics

```
               Accuracy : 0.7037
                 95% CI : (0.4982, 0.8625)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.04012

                  Kappa : 0.5168

 Mcnemar's Test P-Value : 0.44592
```

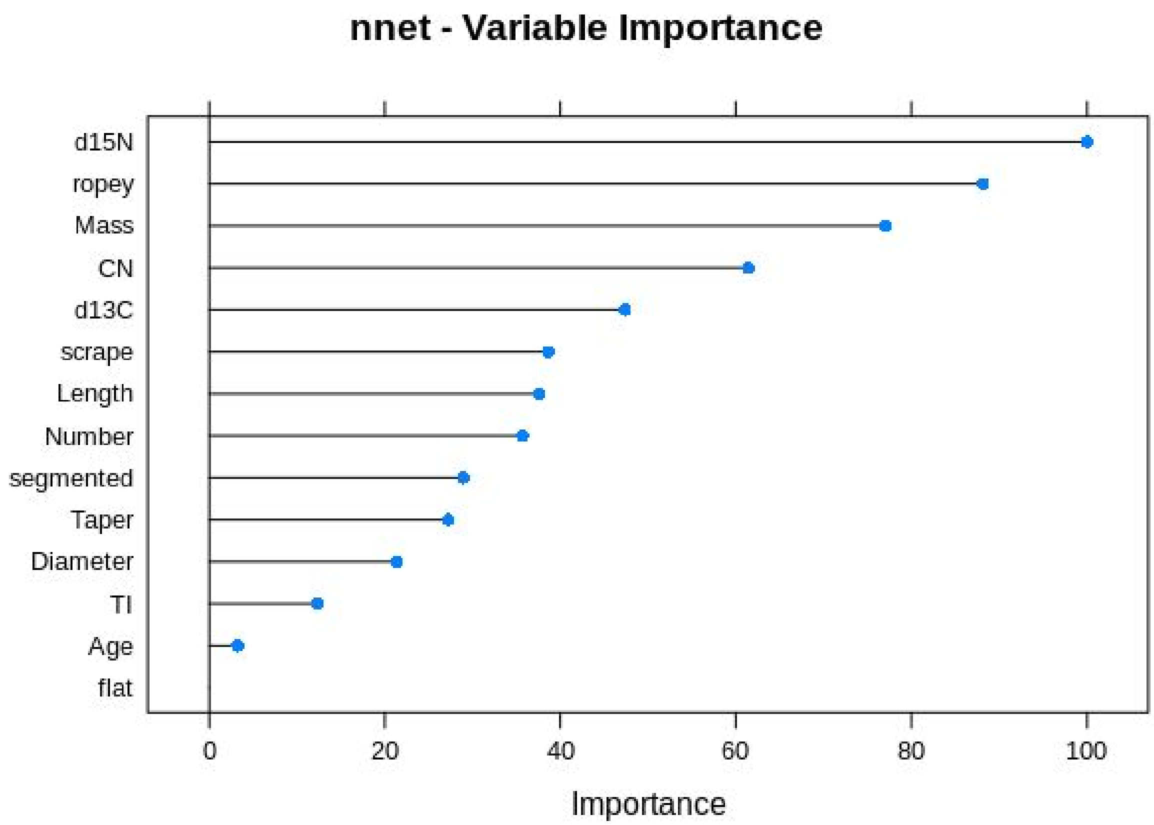
Statistics by Class:

```
               Class: 1 Class: 2 Class: 3
Sensitivity           0.9286   0.5714  0.33333
Specificity           0.9231   0.8000  0.85714
Pos Pred Value        0.9286   0.5000  0.40000
Neg Pred Value        0.9231   0.8421  0.81818
Prevalence            0.5185   0.2593  0.22222
Detection Rate        0.4815   0.1481  0.07407
Detection Prevalence  0.5185   0.2963  0.18519
Balanced Accuracy     0.9258   0.6857  0.59524
```

```
##Neural Net##
#Model
model_nnet<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet', importance=T)
#Summary
print(model_nnet)
#Plot
n <- varImp(object=model_nnet)
n$importance <- data.frame(n$importance[,1])
plot(n, main="nnet - Variable Importance")
#Predictions
predictions_nnet<-predict.train(object=model_nnet,testSet[,predictors],type="raw")
table(predictions_nnet)
#Confusion Matrix and Statistics
confusionMatrix(predictions_nnet,testSet[,outcomeName])
```

```
Neural Network

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

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  0.5531935  0.2587080
  1     1e-04  0.5686637  0.2833380
  1     1e-01  0.5782985  0.2860527
  3     0e+00  0.6582171  0.4492565
  3     1e-04  0.6419318  0.4359897
```

3   1e-01 0.6896808 0.4933364
5   0e+00 0.6776408 0.4751687
5   1e-04 0.6736580 0.4759998
5   1e-01 0.6853650 0.4842056



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.

Confusion Matrix and Statistics

Reference
Prediction 1 2 3
1 10 1 0
2 2 4 3
3 2 2 3

Overall Statistics

Accuracy : 0.6296
95% CI : (0.4237, 0.806)
No Information Rate : 0.5185

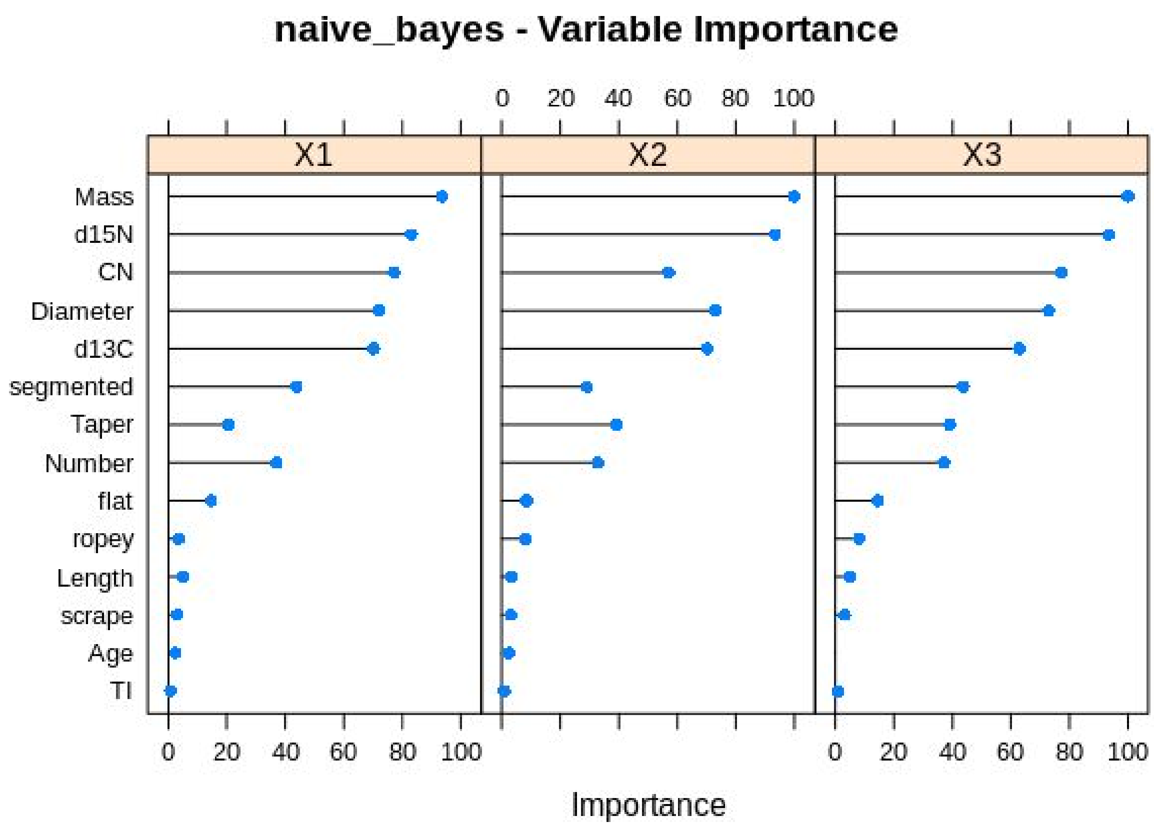
```
   P-Value [Acc > NIR] : 0.1679

                  Kappa : 0.4255

 Mcnemar's Test P-Value : 0.4693

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.7143   0.5714   0.5000
Specificity            0.9231   0.7500   0.8095
Pos Pred Value         0.9091   0.4444   0.4286
Neg Pred Value         0.7500   0.8333   0.8500
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.3704   0.1481   0.1111
Detection Prevalence   0.4074   0.3333   0.2593
Balanced Accuracy      0.8187   0.6607   0.6548
```

```
##Naive Bayes##
#Model
model_naive_bayes<-train(trainSet[,predictors],trainSet[,outcomeName],method='naive_bayes')
#Summary
print(model_naive_bayes)
#Plot
plot(varImp(object=model_naive_bayes),main="naive_bayes - Variable Importance")
#Predictions
predictions_naive_bayes<-predict.train(object=model_naive_bayes,testSet[,predictors],type="ra
w")
table(predictions_naive_bayes)
#Confusion Matrix and Statistics
confusionMatrix(predictions_naive_bayes,testSet[,outcomeName])
```

```
Naive Bayes

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

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	0.665980	0.4730763
TRUE	0.662076	0.4216043

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.



Confusion Matrix and Statistics

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

Overall Statistics

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

                Kappa : 0.7595

 Mcnemar's Test P-Value : 0.5724067

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.9286   0.7143   0.8333
Specificity            0.9231   0.9500   0.9048
Pos Pred Value         0.9286   0.8333   0.7143
Neg Pred Value         0.9231   0.9048   0.9500
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.4815   0.1852   0.1852
Detection Prevalence   0.5185   0.2222   0.2593
Balanced Accuracy      0.9258   0.8321   0.8690
```

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.

```
top_gbm <- c(model_gbm$method, model_gbm$results[1, "Accuracy"], model_gbm$results[1,
"Kappa"])
top_rf <- c(model_rf$method, model_rf$results[1, "Accuracy"], model_rf$results[1, "Kappa"])
top_nnet <- c(model_nnet$method, model_nnet$results[1, "Accuracy"], model_nnet$results[1,
"Kappa"])
top_naive_bayes <- c(model_naive_bayes$method, model_naive_bayes$results[1, "Accuracy"],
model_naive_bayes$results[1, "Kappa"])

consolidated = list(top_gbm, top_rf, top_nnet, top_naive_bayes)
top <- data.frame(matrix(unlist(consolidated), nrow=4, byrow=T))
colnames(top)[1] = "ExperimentName"
colnames(top)[2] = "accuracy"
colnames(top)[3] = "kappa"
top <-top[order(top$accuracy, decreasing = TRUE), ]
```

```
ExperimentName          accuracy             kappa
1            gbm 0.700939704222433 0.503637345892972
2             rf 0.683703864254885 0.464679095516671
4    naive_bayes 0.665980025859402 0.473076325742013
3           nnet 0.553193471165891 0.258707983200176
```

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

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

# visualize the models
plot(model_gbm_tune_length)
```

```
Stochastic Gradient Boosting

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

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     0.7058926  0.5119664
   1                 100     0.7013061  0.5023579
   1                 150     0.6908826  0.4838082
   1                 200     0.7005013  0.5006235
   1                 250     0.7017829  0.5032722
   1                 300     0.6904022  0.4861549
   1                 350     0.6909806  0.4855379
   1                 400     0.6890571  0.4836859
   1                 450     0.6826057  0.4732562
   1                 500     0.6821205  0.4715135
   1                 550     0.6845853  0.4790040
   1                 600     0.6851118  0.4761895
   1                 650     0.6916364  0.4884416
```

1	*700*	*0.6906028*	*0.4868970*
1	*750*	*0.6905370*	*0.4859113*
1	*800*	*0.6839720*	*0.4752836*
1	*850*	*0.6900155*	*0.4855909*
1	*900*	*0.6903862*	*0.4863054*
1	*950*	*0.6892497*	*0.4843916*
1	*1000*	*0.6864182*	*0.4789454*
2	*50*	*0.6924605*	*0.4915177*
2	*100*	*0.6925072*	*0.4887965*
2	*150*	*0.6853469*	*0.4785266*
2	*200*	*0.6827984*	*0.4736630*
2	*250*	*0.6916979*	*0.4898064*
2	*300*	*0.6888802*	*0.4872207*
2	*350*	*0.6864475*	*0.4811046*
2	*400*	*0.6819573*	*0.4757979*
2	*450*	*0.6795060*	*0.4723020*
2	*500*	*0.6908538*	*0.4929096*
2	*550*	*0.6882749*	*0.4875546*
2	*600*	*0.6871967*	*0.4836320*
2	*650*	*0.6880291*	*0.4846220*
2	*700*	*0.6867008*	*0.4826675*
2	*750*	*0.6860742*	*0.4816689*
2	*800*	*0.6861659*	*0.4820927*
2	*850*	*0.6864583*	*0.4823198*
2	*900*	*0.6861244*	*0.4820596*
2	*950*	*0.6860814*	*0.4809598*
2	*1000*	*0.6909971*	*0.4896526*
3	*50*	*0.6926648*	*0.4851403*
3	*100*	*0.6907529*	*0.4870326*
3	*150*	*0.6956084*	*0.4935919*
3	*200*	*0.6985081*	*0.4997790*
3	*250*	*0.6984481*	*0.4998629*
3	*300*	*0.6991359*	*0.5009636*
3	*350*	*0.6969534*	*0.4980927*
3	*400*	*0.6941533*	*0.4937045*
3	*450*	*0.6973117*	*0.4975679*
3	*500*	*0.7036698*	*0.5072951*
3	*550*	*0.7005630*	*0.5044610*
3	*600*	*0.6997877*	*0.5029525*
3	*650*	*0.6888052*	*0.4852868*
3	*700*	*0.6920387*	*0.4887244*
3	*750*	*0.6890248*	*0.4844853*
3	*800*	*0.6917717*	*0.4898983*

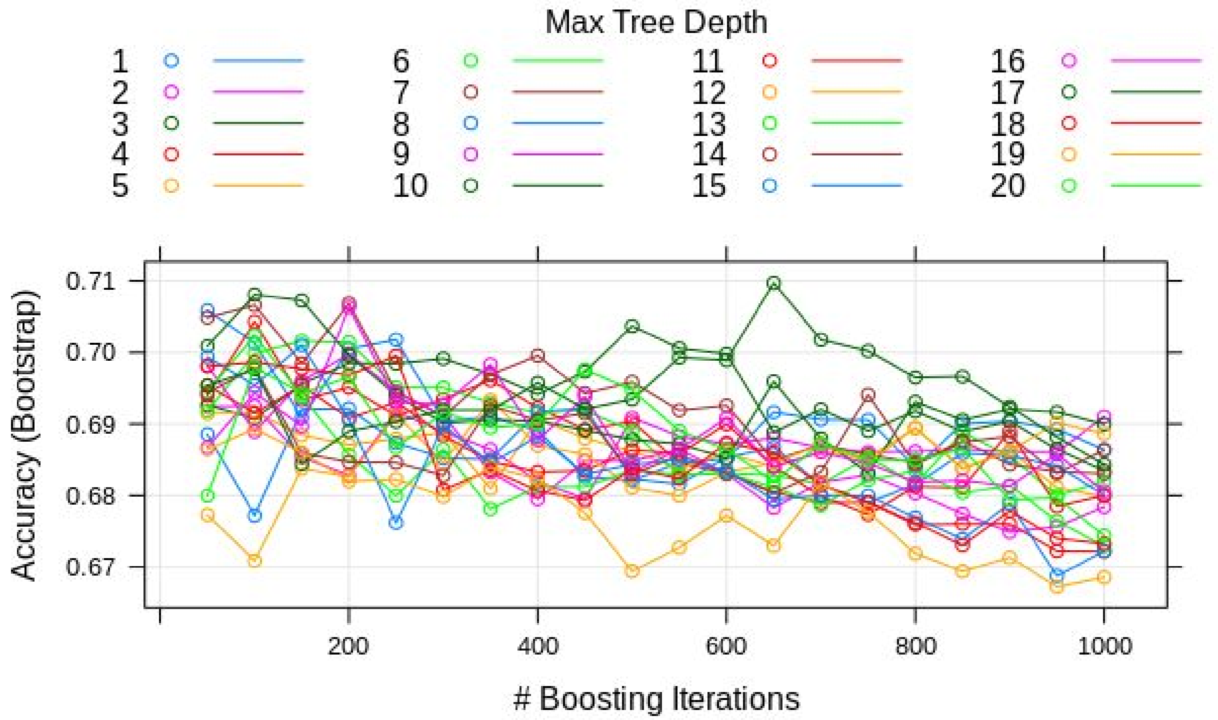
3	*850*	*0.6889359*	*0.4859331*
3	*900*	*0.6903503*	*0.4881572*
3	*950*	*0.6864505*	*0.4828508*
3	*1000*	*0.6834505*	*0.4783156*
4	*50*	*0.6939892*	*0.4906512*
4	*100*	*0.7042999*	*0.5052648*
4	*150*	*0.6933737*	*0.4907262*
4	*200*	*0.6951153*	*0.4944502*
4	*250*	*0.6911266*	*0.4884588*
4	*300*	*0.6934637*	*0.4922822*
4	*350*	*0.6961008*	*0.4956613*
4	*400*	*0.6922803*	*0.4899888*
4	*450*	*0.6891841*	*0.4858723*
4	*500*	*0.6903000*	*0.4869612*
4	*550*	*0.6845381*	*0.4782006*
4	*600*	*0.6831096*	*0.4764523*
4	*650*	*0.6804689*	*0.4721616*
4	*700*	*0.6789787*	*0.4698453*
4	*750*	*0.6790270*	*0.4699219*
4	*800*	*0.6759380*	*0.4653981*
4	*850*	*0.6760270*	*0.4647434*
4	*900*	*0.6760270*	*0.4647955*
4	*950*	*0.6722343*	*0.4582109*
4	*1000*	*0.6721912*	*0.4575923*
5	*50*	*0.6914278*	*0.4879221*
5	*100*	*0.6916427*	*0.4897480*
5	*150*	*0.6884411*	*0.4860060*
5	*200*	*0.6870999*	*0.4842092*
5	*250*	*0.6879297*	*0.4840991*
5	*300*	*0.6887704*	*0.4838920*
5	*350*	*0.6809647*	*0.4720695*
5	*400*	*0.6906839*	*0.4863111*
5	*450*	*0.6879885*	*0.4839642*
5	*500*	*0.6862123*	*0.4802341*
5	*550*	*0.6837457*	*0.4779671*
5	*600*	*0.6859026*	*0.4804036*
5	*650*	*0.6840293*	*0.4778379*
5	*700*	*0.6861065*	*0.4822491*
5	*750*	*0.6853456*	*0.4810586*
5	*800*	*0.6893719*	*0.4870555*
5	*850*	*0.6838446*	*0.4799249*
5	*900*	*0.6862087*	*0.4830573*
5	*950*	*0.6811257*	*0.4735024*

5	*1000*	*0.6798109*	*0.4700945*
6	*50*	*0.6918512*	*0.4857085*
6	*100*	*0.6998491*	*0.5009198*
6	*150*	*0.7016206*	*0.5082473*
6	*200*	*0.7014162*	*0.5085316*
6	*250*	*0.6951008*	*0.4957551*
6	*300*	*0.6950611*	*0.4953349*
6	*350*	*0.6929597*	*0.4920357*
6	*400*	*0.6915472*	*0.4898795*
6	*450*	*0.6918755*	*0.4895853*
6	*500*	*0.6889935*	*0.4861803*
6	*550*	*0.6828627*	*0.4773260*
6	*600*	*0.6830963*	*0.4777254*
6	*650*	*0.6828339*	*0.4769649*
6	*700*	*0.6816047*	*0.4749107*
6	*750*	*0.6853193*	*0.4816357*
6	*800*	*0.6815974*	*0.4761750*
6	*850*	*0.6888011*	*0.4869582*
6	*900*	*0.6854726*	*0.4800774*
6	*950*	*0.6800240*	*0.4706386*
6	*1000*	*0.6819684*	*0.4762914*
7	*50*	*0.7048559*	*0.5077815*
7	*100*	*0.7066248*	*0.5118528*
7	*150*	*0.6984496*	*0.5001795*
7	*200*	*0.7068512*	*0.5141186*
7	*250*	*0.6944563*	*0.4944450*
7	*300*	*0.6910664*	*0.4883884*
7	*350*	*0.6969252*	*0.4970345*
7	*400*	*0.6995535*	*0.5020569*
7	*450*	*0.6942895*	*0.4924842*
7	*500*	*0.6958498*	*0.4964839*
7	*550*	*0.6919017*	*0.4899174*
7	*600*	*0.6925838*	*0.4909791*
7	*650*	*0.6849042*	*0.4791015*
7	*700*	*0.6871102*	*0.4817607*
7	*750*	*0.6856826*	*0.4800773*
7	*800*	*0.6845790*	*0.4780266*
7	*850*	*0.6876250*	*0.4838510*
7	*900*	*0.6843409*	*0.4772241*
7	*950*	*0.6833123*	*0.4767466*
7	*1000*	*0.6830742*	*0.4757773*
8	*50*	*0.6885490*	*0.4846552*
8	*100*	*0.6771905*	*0.4646982*

8	150	0.6920568	0.4907372
8	200	0.6921502	0.4905909
8	250	0.6873735	0.4817361
8	300	0.6852081	0.4791497
8	350	0.6850628	0.4774671
8	400	0.6917201	0.4889099
8	450	0.6921961	0.4900628
8	500	0.6823091	0.4746569
8	550	0.6816241	0.4758124
8	600	0.6852815	0.4830483
8	650	0.6867457	0.4829871
8	700	0.6815037	0.4760285
8	750	0.6789877	0.4718992
8	800	0.6815653	0.4759200
8	850	0.6858963	0.4828249
8	900	0.6855683	0.4827818
8	950	0.6841211	0.4805008
8	1000	0.6800937	0.4745660
9	50	0.6981679	0.5000707
9	100	0.6889126	0.4863497
9	150	0.6959703	0.4944156
9	200	0.6997936	0.5016447
9	250	0.6944821	0.4929356
9	300	0.6911041	0.4884020
9	350	0.6832623	0.4755256
9	400	0.6794889	0.4687519
9	450	0.6848292	0.4787400
9	500	0.6848292	0.4783083
9	550	0.6864648	0.4797645
9	600	0.6830123	0.4764063
9	650	0.6782745	0.4672898
9	700	0.6818654	0.4739593
9	750	0.6828866	0.4749167
9	800	0.6802925	0.4710351
9	850	0.6774488	0.4642258
9	900	0.6748728	0.4609719
9	950	0.6756683	0.4612204
9	1000	0.6783954	0.4655456
10	50	0.7008772	0.5022990
10	100	0.7080799	0.5170387
10	150	0.7072665	0.5154254
10	200	0.6995885	0.5014298
10	250	0.6940051	0.4916336

10	300	0.6900889	0.4865164
10	350	0.6905850	0.4900065
10	400	0.6903844	0.4885779
10	450	0.6891112	0.4863116
10	500	0.6878037	0.4846395
10	550	0.6872353	0.4831970
10	600	0.6851563	0.4797513
10	650	0.6959172	0.4985950
10	700	0.6879588	0.4853197
10	750	0.6831154	0.4775642
10	800	0.6930977	0.4957555
10	850	0.6905261	0.4927456
10	900	0.6920686	0.4943470
10	950	0.6916565	0.4933103
10	1000	0.6899521	0.4895809
11	50	0.6953916	0.4871349
11	100	0.6915040	0.4881147
11	150	0.6953804	0.4942970
11	200	0.6907483	0.4883844
11	250	0.6930863	0.4912792
11	300	0.6883701	0.4868326
11	350	0.6847373	0.4809097
11	400	0.6832881	0.4781705
11	450	0.6834732	0.4782593
11	500	0.6863759	0.4833053
11	550	0.6858690	0.4826173
11	600	0.6899625	0.4884005
11	650	0.6840351	0.4804563
11	700	0.6799120	0.4740008
11	750	0.6773378	0.4702610
11	800	0.6812574	0.4769946
11	850	0.6811037	0.4752353
11	900	0.6889706	0.4880188
11	950	0.6784898	0.4711364
11	1000	0.6799800	0.4729047
12	50	0.6772937	0.4639104
12	100	0.6708584	0.4564109
12	150	0.6837624	0.4772286
12	200	0.6826489	0.4754625
12	250	0.6912913	0.4886887
12	300	0.6845587	0.4758348
12	350	0.6933582	0.4938948
12	400	0.6869390	0.4827456

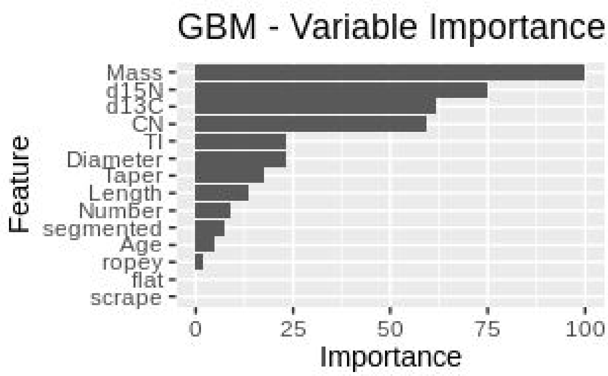
```
  12           450    0.6857645  0.4808286
  12           500    0.6810990  0.4740225
  12           550    0.6799512  0.4713372
  12           600    0.6832418  0.4791665
  12           650    0.6846103  0.4807624
  12           700    0.6870439  0.4857020
  12           750    0.6846704  0.4807957
  12           800    0.6894455  0.4875881
  12           850    0.6836910  0.4790702
  12           900    0.6862364  0.4830184
  12           950    0.6901442  0.4911616
  12          1000    0.6886266  0.4884098
  13            50    0.6945348  0.4952526
  13           100    0.7023882  0.5068107
  13           150    0.6939072  0.4951320
  13           200    0.6856806  0.4824237
  13           250    0.6799563  0.4713191
  13           300    0.6864795  0.4831196
  13           350    0.6781101  0.4689937
  13           400    0.6812267  0.4730588
  13           450    0.6812203  0.4733059
  13           500    0.6835812  0.4779453
 [ reached getOption("max.print") -- omitted 150 rows ]

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 = 650, interaction.depth = 17, shrinkage
 = 0.1 and n.minobsinnode = 10.
```

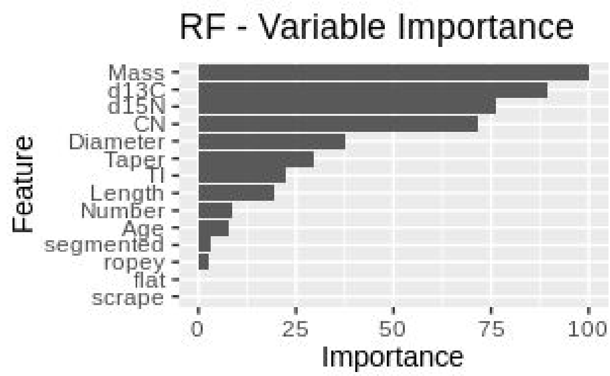


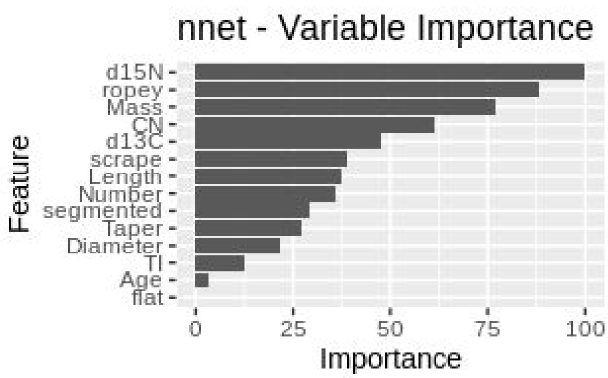
Using GGplot and gridExtra to plot all variable of importance plots into one single plot.

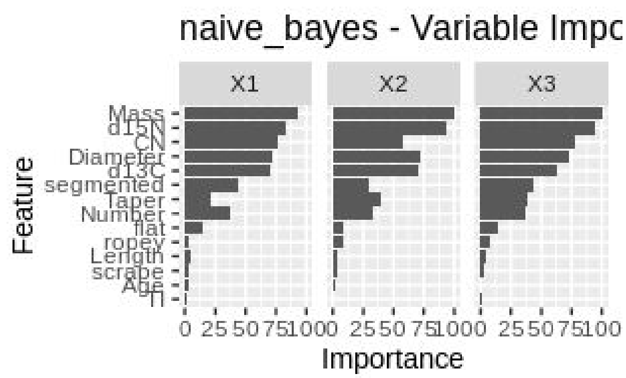
one <- ggplot(varImp(object=model_gbm)) + ggtitle("GBM - Variable Importance")
two <- ggplot(varImp(object=model_rf)) + ggtitle("RF - Variable Importance")
three <- ggplot(n) + ggtitle("nnet - Variable Importance")
four <- ggplot(varImp(object=model_naive_bayes)) + ggtitle("naive_bayes - Variable Importance")

grid.arrange(one,two, three, four, five, nrow = 2, ncol = 2)









Which model performs the best? and why do you think this is the case? Can we accurately predict species on this dataset?

Out of the non-tuned models, gradient boosting performed the best with accuracy near to 70%. I think this is the case due to its capability to convert weak learners into strong learners. We can predict the species with 70% accuracy.

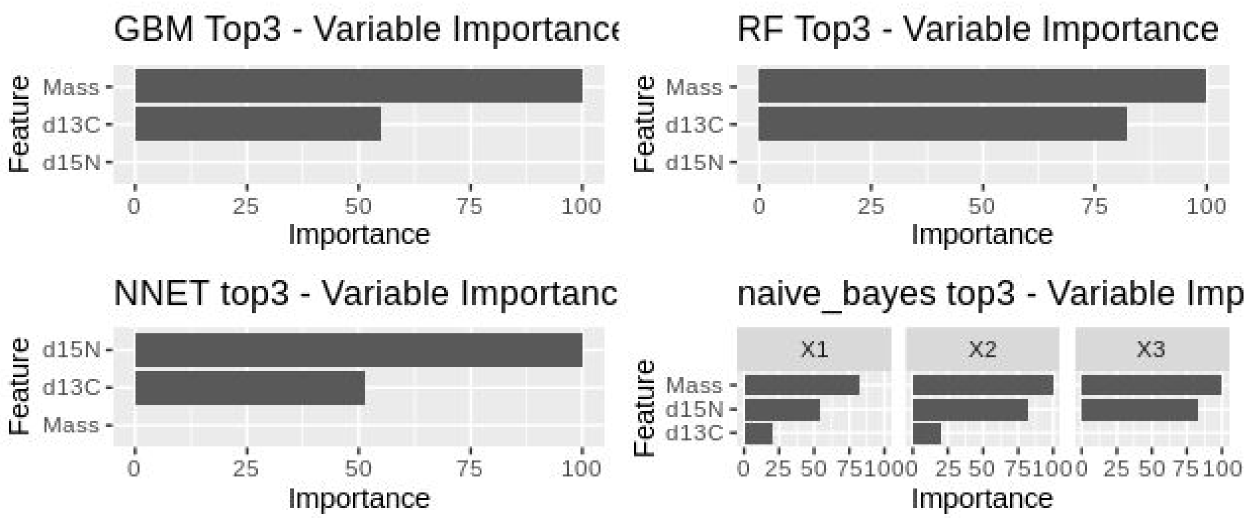
Using feature selection with rfe in caret and the repeatedcv method: Find the top 3 predictors and build the same models as in 6 and 8 with the same parameters.

```
control <- rfeControl(functions = rfFuncs,
                      method = "repeatedcv",
                      repeats = 3,
                      verbose = FALSE)
top3<-names(trainSet)[!names(trainSet) %in% outcomeName]
columns <- rfe(trainSet[,top3], trainSet[,outcomeName],
               rfeControl = control)
columns

top3 <- c("d15N", "Mass", "d13C")

#####Models#####
##GBM##
```

```
#Model
model_top3_gbm<-train(trainSet[,top3],trainSet[,outcomeName],method='gbm')
#Summary
print(model_top3_gbm)
#Plot
plot(varImp(object=model_top3_gbm),main="GBM - Variable Importance")
#Predictions
predictions_gbm_top3<-predict.train(object=model_top3_gbm,testSet[,top3],type="raw")
table(predictions_gbm_top3)
#Confusion Matrix and Statistics
confusionMatrix(predictions_gbm_top3,testSet[,outcomeName])


##Random Forest##
#Model
model_top3_rf<-train(trainSet[,top3],trainSet[,outcomeName],method='rf')
#Summary
print(model_top3_rf)
#Plot
plot(varImp(object=model_top3_rf),main="RF - Variable Importance")
#Predictions
predictions_rf_top3<-predict.train(object=model_top3_rf,testSet[,top3],type="raw")
table(predictions_rf_top3)
#Confusion Matrix and Statistics
confusionMatrix(predictions_rf_top3,testSet[,outcomeName])


##Neural Net##
#Model
model_top3_nnet<-train(trainSet[,top3],trainSet[,outcomeName],method='nnet', importance=T)
#Summary
print(model_top3_nnet)
#Plot
nt <- varImp(object=model_top3_nnet)
nt$importance <- data.frame(nt$importance[,1])
plot(nt, main="nnet - Variable Importance")
#Predictions
predictions_nnet_top3<-predict.train(object=model_top3_nnet,testSet[,top3],type="raw")
table(predictions_nnet_top3)
#Confusion Matrix and Statistics
confusionMatrix(predictions_nnet_top3,testSet[,outcomeName])
```

```
##Naive Bayes##
#Model
model_top3_naive_bayes<-train(trainSet[,top3],trainSet[,outcomeName],method='naive_bayes')
#Summary
print(model_top3_naive_bayes)
#Plot
plot(varImp(object=model_top3_naive_bayes),main="naive_bayes - Variable Importance")
#Predictions
predictions_naive_bayes_top3<-predict.train(object=model_top3_naive_bayes,testSet[,top3],typ
e="raw")
table(predictions_naive_bayes_top3)
#Confusion Matrix and Statistics
confusionMatrix(predictions_naive_bayes_top3,testSet[,outcomeName])

#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.

top3_gbm <- c(model_top3_gbm$method, model_top3_gbm$results[1, "Accuracy"],
model_top3_gbm$results[1, "Kappa"])
top3_rf <- c(model_top3_rf$method, model_top3_rf$results[1, "Accuracy"],
model_top3_rf$results[1, "Kappa"])
top3_nnet <- c(model_top3_nnet$method, model_top3_nnet$results[1, "Accuracy"],
model_top3_nnet$results[1, "Kappa"])
top3_naive_bayes <- c(model_top3_naive_bayes$method, model_top3_naive_bayes$results[1,
"Accuracy"], model_top3_naive_bayes$results[1, "Kappa"])

top3_naive_bayes[1] <- "Naive Bayes Top3"
top3_nnet[1] <- "Neural-Net Top3"
top3_rf[1] <- "Random Forest Top3"
top3_gbm[1] <- "GBM Tuned Top3"

consolidated3 = list(top3_gbm, top3_rf, top3_nnet, top3_naive_bayes)
top3df <- data.frame(matrix(unlist(consolidated3), nrow=4, byrow=T))
colnames(top3df)[1] = "ExperimentName"
colnames(top3df)[2] = "accuracy"
colnames(top3df)[3] = "kappa"
top3df <- top3df[order(top3df$accuracy, decreasing = TRUE), ]
top3df

#8. Using GGplot and gridExtra to plot all variable of importance plots into one single plot.
a <- ggplot(varImp(object=model_top3_gbm)) + ggtitle("GBM Top3 - Variable Importance")
```

```
b <- ggplot(varImp(object=model_top3_rf)) + ggtitle("RF Top3 - Variable Importance")
c <- ggplot(nt) + ggtitle("NNET top3 - Variable Importance")
d <- ggplot(varImp(object=model_top3_naive_bayes)) + ggtitle("naive_bayes top3 - Variable Importance")

grid.arrange(a,b, c, d, nrow = 2, ncol = 2)
```



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.

```
ExperimentName          accuracy            kappa
1          GBM Tuned 0.705892562260533 0.511966356735086
4 Random Forest Tuned 0.688035304351064 0.436413134411252
2   Naive Bayes Tuned  0.63860889423292 0.436413134411252
3    Neural-Net Tuned 0.566483019648683 0.436413134411252
```

Which model performs the best? and why do you think this is the case? Can we accurately predict species on this dataset?

The GBM tuned model performs the best among all. I think the reason for this is ‘Boosting’. The GBM model converts the weak learners into strong learners. We can 70% accurately predict the species on this dataset.