```
# Libraries
library(caret)
library(gbm)
library(randomForest)
library(mlbench)
library(e1071)
library('RANN')
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 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.

```
########## Answer1
#Converting 'Species' column to numeric
outcomeName<-'Species'
scat$Species<-ifelse(scat$Species=='gray_fox',3,ifelse(scat$Species=='coyote',2,1))
str(scat)
View(scat)
```

```
'data.frame':   110 obs. of  19 variables:
 $ Species  : num  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 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 ...
```

	Species	Month	Year	Site	Location	Age	Number	Length	Diameter	Taper	TI	Mass	d13C	d15N
1	2	January	2012	YOLA	edge	5	2	9.5	25.7	41.9	1.63	15.89	-26.85	6.94
2	2	January	2012	YOLA	edge	3	2	14.0	25.4	37.1	1.46	17.61	-29.62	9.87
3	1	January	2012	YOLA	middle	3	2	9.0	18.8	16.5	0.88	8.40	-28.73	8.52
4	2	January	2012	YOLA	middle	5	2	8.5	18.1	24.7	1.36	7.40	-20.07	5.79
6	2	January	2012	YOLA	edge	5	4	8.0	20.7	20.1	0.97	25.45	-23.24	7.01
7	2	January	2012	YOLA	edge	5	3	9.0	21.2	28.5	1.34	14.14	-29.00	8.28
8	1	January	2012	ANNU	off_edge	1	5	6.0	15.7	8.2	0.52	14.82	-28.06	4.20
9	1	January	2012	ANNU	off_edge	3	7	5.5	21.9	19.3	0.88	26.41	-27.60	3.89
10	1	January	2012	ANNU	off_edge	5	2	11.0	17.5	29.1	1.66	16.24	-28.64	7.34
13	1	January	2012	ANNU	middle	5	1	20.5	18.0	21.4	1.19	11.22	-27.35	6.06
14	3	January	2012	ANNU	middle	3	1	8.0	NA	NA	NA	2.51	-25.79	7.83
15	3	January	2012	ANNU	middle	1	1	8.0	12.9	14.7	1.14	8.55	-25.71	8.47
16	3	January	2012	ANNU	middle	3	1	12.0	NA	NA	NA	18.14	-25.18	10.10
18	3	January	2012	ANNU	middle	3	1	11.5	NA	NA	NA	8.17	-25.73	9.72
19	3	January	2012	ANNU	middle	1	1	8.5	NA	NA	NA	3.43	-26.17	8.07
20	3	January	2012	ANNU	middle	5	1	10.5	12.1	11.9	0.98	3.10	-26.88	6.70
21	1	February	2013	ANNU	edge	5	7	5.0	13.0	37.6	2.89	9.75	-27.92	7.57
23	2	February	2013	ANNU	edge	5	6	6.5	24.0	23.1	0.96	33.00	-27.66	12.88
24	1	February	2013	ANNU	edge	5	4	10.5	15.5	38.2	2.46	12.76	-25.77	3.88
25	1	February	2013	ANNU	off_edge	3	3	11.0	16.5	25.8	1.56	18.75	-28.91	6.36
26	1	February	2013	ANNU	off_edge	5	4	11.5	17.5	18.9	1.08	14.08	-27.30	6.61

2. Remove the Month, Year, Site, Location features.

```
########## Answer2
#Dropping specified columns
drops <- c("Month","Year","Site","Location")
scat_processed<-scat[ , !(names(scat) %in% drops)]
View(scat_processed)

#As we dont have categorical value, we dont have to create dummy variables, and we can proceed to convert back our target feature to caterogical value
```

```
#Convering 'Species' column back to categorical
scat_processed$Species<-as.factor(scat_processed$Species)
str(scat_processed)
```

	Species	Age	Number	Length	Diameter	Taper	TI	Mass	d13C	d15N	CN	ropey	segmented	flat
1	2	5	2	9.5	25.7	41.9	1.63	15.89	-26.85	6.94	8.50	0	0	0
2	2	3	2	14.0	25.4	37.1	1.46	17.61	-29.62	9.87	11.30	0	0	0
3	1	3	2	9.0	18.8	16.5	0.88	8.40	-28.73	8.52	8.10	1	1	0
4	2	5	2	8.5	18.1	24.7	1.36	7.40	-20.07	5.79	11.50	1	0	0
6	2	5	4	8.0	20.7	20.1	0.97	25.45	-23.24	7.01	10.60	0	1	0
7	2	5	3	9.0	21.2	28.5	1.34	14.14	-29.00	8.28	9.00	1	0	0
8	1	1	5	6.0	15.7	8.2	0.52	14.82	-28.06	4.20	5.40	1	1	0
9	1	3	7	5.5	21.9	19.3	0.88	26.41	-27.60	3.89	5.60	0	1	0
10	1	5	2	11.0	17.5	29.1	1.66	16.24	-28.64	7.34	5.80	0	1	0
13	1	5	1	20.5	18.0	21.4	1.19	11.22	-27.35	6.06	7.70	1	1	0
14	3	3	1	8.0	NA	NA	NA	2.51	-25.79	7.83	20.50	0	0	1
15	3	1	1	8.0	12.9	14.7	1.14	8.55	-25.71	8.47	18.10	1	0	0
16	3	3	1	12.0	NA	NA	NA	18.14	-25.18	10.10	15.50	0	0	1
18	3	3	1	11.5	NA	NA	NA	8.17	-25.73	9.72	18.90	0	0	1
19	3	1	1	8.5	NA	NA	NA	3.43	-26.17	8.07	19.90	0	0	1
20	3	5	1	10.5	12.1	11.9	0.98	3.10	-26.88	6.70	7.00	1	1	0
21	1	5	7	5.0	13.0	37.6	2.89	9.75	-27.92	7.57	5.80	1	1	0
23	2	5	6	6.5	24.0	23.1	0.96	33.00	-27.66	12.88	7.70	1	1	0
24	1	5	4	10.5	15.5	38.2	2.46	12.76	-25.77	3.88	5.70	1	0	0
25	1	3	3	11.0	16.5	25.8	1.56	18.75	-28.91	6.36	6.00	1	1	0
26	1	5	4	11.5	17.5	18.9	1.08	14.08	-27.30	6.61	6.90	1	1	0

```
'data.frame':	110 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "1","2","3": 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.

```
########## Answer3
sum(is.na(scat_processed))
preProcValues <- preProcess(scat_processed, method = c("knnImpute","center","scale"))
```

```
scat_processed <- predict(preProcValues, scat_processed)
sum(is.na(scat_processed))
```

```
> ########## 3. Check if any values are null. If there are, impute missing values using KNN.
> ########## Answer3
> sum(is.na(scat_processed))
[1] 47
> preProcValues <- preProcess(scat_processed, method = c("knnImpute","center","scale"))
> scat_processed <- predict(preProcValues, scat_processed)
> sum(is.na(scat_processed))
[1] 0
```

4. Converting every categorical variable to numerical (if needed).

```
########## Answer4
#Not Needed as we dont have any categorical value feature (in predictors)
str(scat_processed)
```

```
'data.frame':	110 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "1","2","3": 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.

```
########## Answer5
#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,]
str(trainSet)
str(testSet)

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

#Building Models

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')
```

```
'data.frame':	83 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "1","2","3": 2 1 2 2 2 1 1 3 3 3 ...
 $ Age      : num  1.207 -0.252 1.207 1.207 1.207 ...
 $ Number   : num  -0.433 -0.433 -0.433 0.968 0.268 ...
 $ Length   : num  0.0587 -0.0867 -0.2322 -0.3777 -0.0867 ...
 $ Diameter : num  1.8396 0.0622 -0.1181 0.5516 0.6804 ...
 $ Taper    : num  0.9609 -0.7262 -0.1816 -0.4871 0.0709 ...
 $ TI       : num  0.0283 -0.7171 -0.24 -0.6277 -0.2599 ...
 $ Mass     : num  0.388 -0.458 -0.571 1.469 0.19 ...
 $ d13C     : num  0.00468 -0.85947 3.12113 1.66403 -0.98357 ...
 $ d15N     : num  -0.165 0.359 -0.546 -0.141 0.28 ...
 $ CN       : num  0.0276 -0.0816 0.8468 0.6011 0.1642 ...
 $ ropey    : num  -1.131 0.876 0.876 -1.131 0.876 ...
 $ segmented: num  -1.131 0.876 -1.131 0.876 -1.131 ...
 $ flat     : num  -0.239 -0.239 -0.239 -0.239 -0.239 ...
 $ scrape   : num  -0.217 4.562 -0.217 -0.217 -0.217 ...
```

```
> str(testSet)
'data.frame':	27 obs. of  15 variables:
 $ Species  : Factor w/ 3 levels "1","2","3": 2 1 1 3 3 3 2 1 1 1 ...
 $ Age      : num  -0.252 -0.252 1.207 -0.252 -1.711 ...
 $ Number   : num  -0.433 3.071 -1.134 -1.134 -1.134 ...
 $ Length   : num  1.368 -1.105 3.259 -0.378 -0.378 ...
 $ Diameter : num  1.762 0.861 -0.144 -0.628 -1.458 ...
 $ Taper    : num  0.6421 -0.5402 -0.4007 -0.0341 -0.8458 ...
 $ TI       : num  -0.141 -0.717 -0.409 -0.085 -0.459 ...
 $ Mass     : num  0.583 1.577 -0.14 -1.124 -0.441 ...
 $ d13C     : num  -1.269 -0.34 -0.225 0.492 0.529 ...
 $ d15N     : num  0.807 -1.176 -0.456 0.13 0.343 ...
 $ CN       : num  0.792 -0.764 -0.191 3.304 2.649 ...
```

```
 $ ropey    : num  -1.131 -1.131 0.876 -1.131 0.876 ...
 $ segmented: num  -1.131 0.876 0.876 -1.131 -1.131 ...
 $ flat     : num  -0.239 -0.239 -0.239 4.144 -0.239 ...
 $ scrape   : num  -0.217 -0.217 -0.217 -0.217 -0.217 ...
```

```
> model_nnet<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet')
# weights:  21
initial  value 96.518725
iter  10 value 65.027265
iter  20 value 62.389632
iter  30 value 61.678498
iter  40 value 61.330473
iter  50 value 60.247233
iter  60 value 60.218436
iter  70 value 60.094161
iter  80 value 59.800416
iter  90 value 59.644762
iter 100 value 59.380861
final  value 59.380861
stopped after 100 iterations
# weights:  57
initial  value 100.526310
iter  10 value 33.419895
iter  20 value 17.010371
iter  30 value 13.721868
iter  40 value 12.950765
iter  50 value 11.508013
iter  60 value 10.022844
iter  70 value 9.138934
iter  80 value 8.757278
iter  90 value 7.945041
iter 100 value 6.907337
final  value 6.907337
stopped after 100 iterations
# weights:  93
initial  value 111.258429
iter  10 value 12.746922
iter  20 value 3.744986
iter  30 value 3.214262
iter  40 value 3.113502
iter  50 value 2.182265
iter  60 value 2.153277
iter  70 value 2.133242
iter  80 value 1.912449
iter  90 value 0.355341
```

```
iter 100 value 0.269563
final  value 0.269563
stopped after 100 iterations
# weights:  21
initial  value 105.595080
iter  10 value 39.349392
iter  20 value 35.691139
iter  30 value 35.642093
final  value 35.642008
converged
# weights:  57
initial  value 118.314748
iter  10 value 31.508917
iter  20 value 10.118117
iter  30 value 7.838338
iter  40 value 7.653275
iter  50 value 7.613729
iter  60 value 7.597590
iter  70 value 7.587029
iter  80 value 7.581886
iter  90 value 7.564758
iter 100 value 7.467114
final  value 7.467114
stopped after 100 iterations
# weights:  93
initial  value 84.622193
iter  10 value 20.118290
iter  20 value 0.570974
iter  30 value 0.008286
iter  40 value 0.000376
final  value 0.000086
converged
```

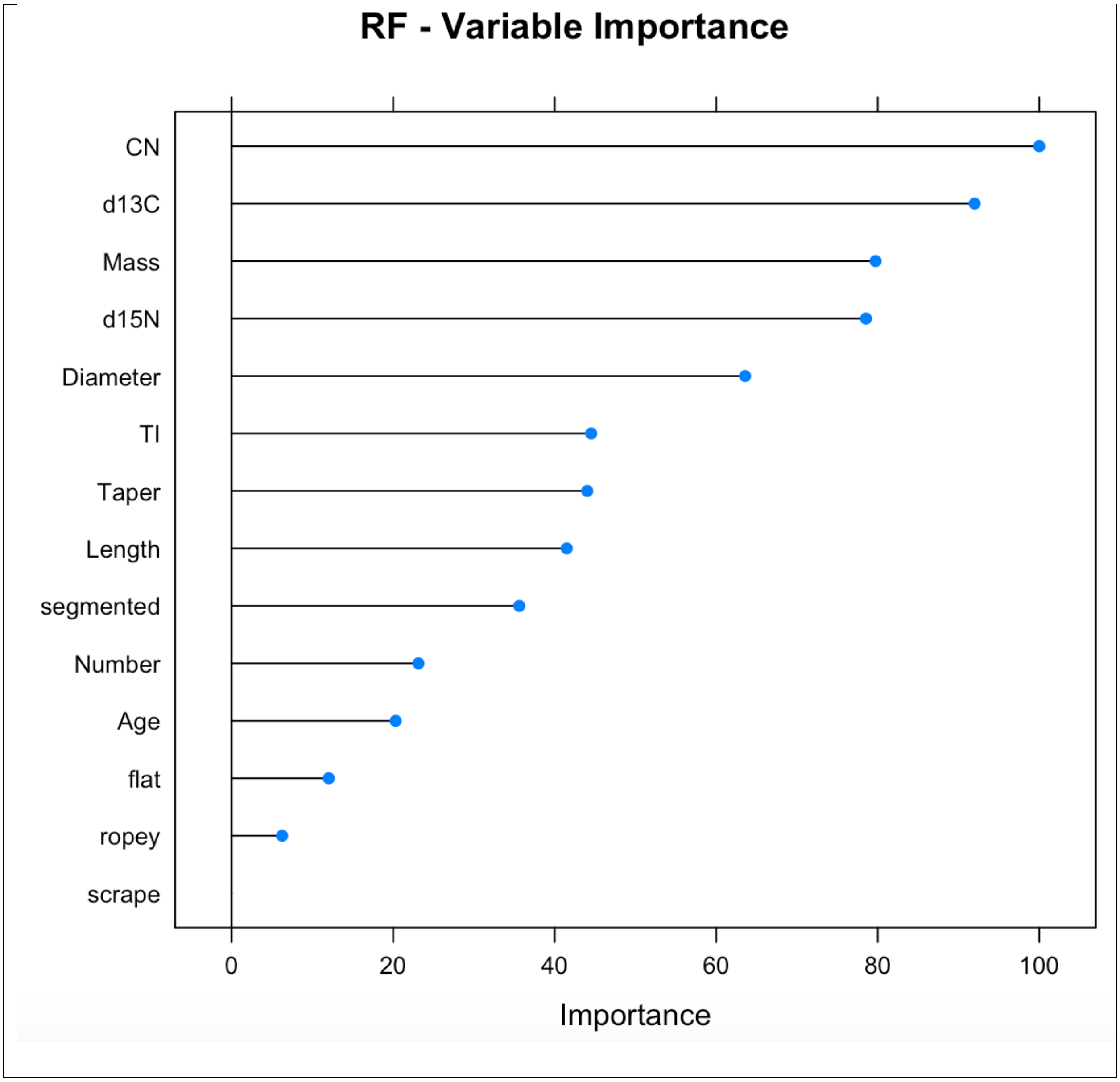
```
> model_gbm<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.0986             nan     0.1000    0.1862
     2        0.9554             nan     0.1000    0.1239
     3        0.8574             nan     0.1000    0.0704
     4        0.7918             nan     0.1000    0.0606
     5        0.7285             nan     0.1000    0.0544
     6        0.6699             nan     0.1000    0.0351
     7        0.6272             nan     0.1000    0.0400
     8        0.5799             nan     0.1000    0.0462
     9        0.5400             nan     0.1000    0.0003
    10        0.5192             nan     0.1000   -0.0134
```

```
    20       0.3335          nan     0.1000   -0.0106
    40       0.1740          nan     0.1000   -0.0101
    60       0.0891          nan     0.1000   -0.0125
    80       0.0463          nan     0.1000   -0.0107
   100       0.0269          nan     0.1000   -0.0012
   120       0.0177          nan     0.1000   -0.0062
   140       0.0109          nan     0.1000   -0.0034
   150       0.0108          nan     0.1000   -0.0007

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1       1.0986          nan     0.1000    0.1666
     2       0.9804          nan     0.1000    0.0524
     3       0.9077          nan     0.1000    0.0207
     4       0.8514          nan     0.1000    0.0441
     5       0.7902          nan     0.1000   -0.0118
     6       0.7472          nan     0.1000    0.0090
     7       0.7158          nan     0.1000    0.0103
     8       0.6859          nan     0.1000    0.0029
     9       0.6567          nan     0.1000    0.0045
    10       0.6242          nan     0.1000   -0.0372
    20       0.4560          nan     0.1000   -0.0528
    40       0.2623          nan     0.1000   -0.0138
    50       0.2109          nan     0.1000   -0.0150
```

```
#model_rf
# a) Randomforest - model summarization
print(model_rf)

# b) Randomforest - plot of variable of importance
varImp(object=model_rf)
plot(varImp(object=model_rf),main="RF - Variable Importance")

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

```
> #model_rf
> # a) Randomforest - model summarization
> print(model_rf)
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    0.6489581  0.3753569
   8    0.6447794  0.3948459
  14    0.6485858  0.4080342

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

```
> # b) Randomforest - plot of variable of importance
> varImp(object=model_rf)
rf variable importance

          Overall
CN        100.000
d13C       91.994
Mass       79.731
d15N       78.542
Diameter   63.578
TI         44.517
Taper      44.031
Length     41.497
segmented  35.612
Number     23.133
Age        20.309
flat       12.024
ropey       6.256
scrape      0.000
```

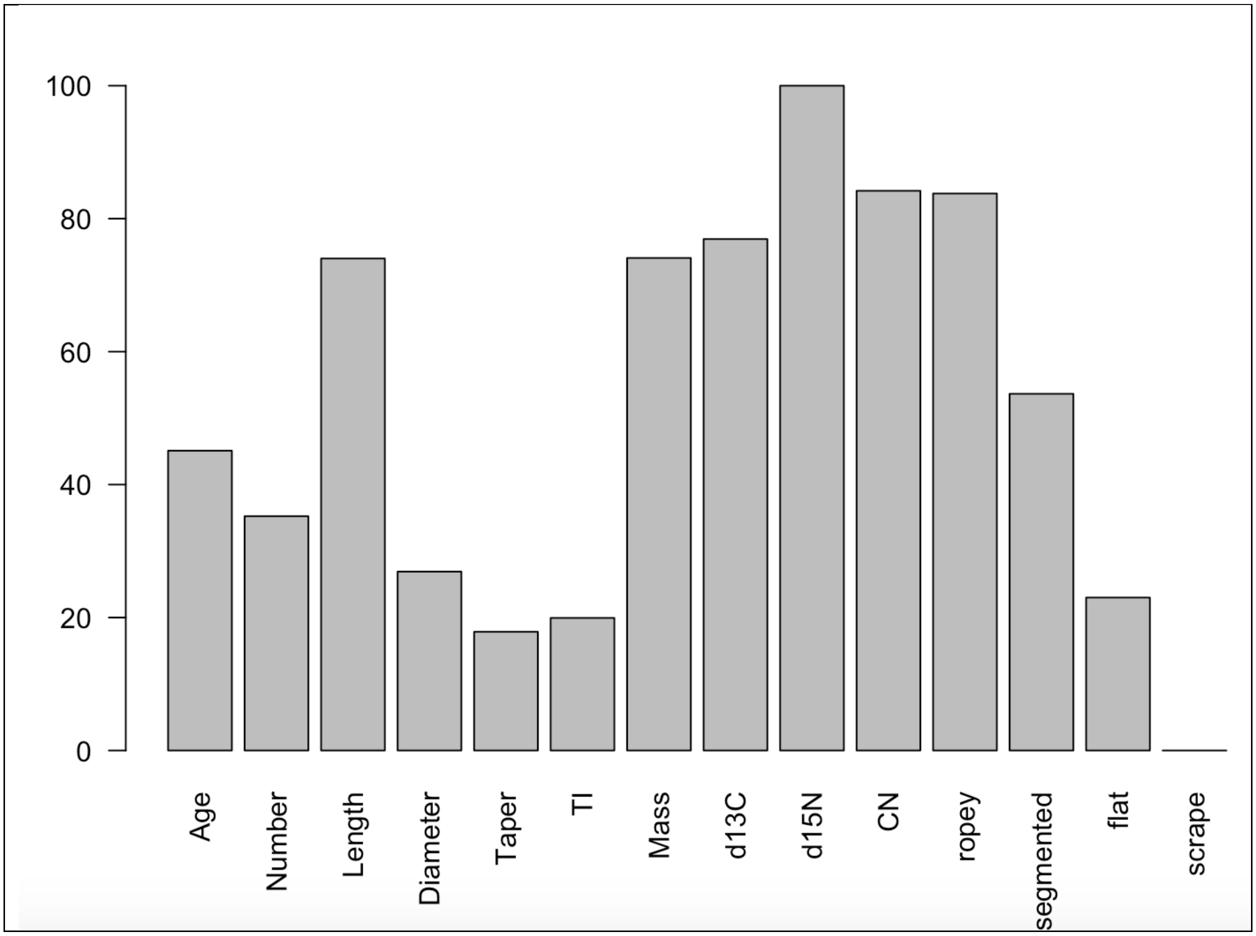

RF - Variable Importance
CN
d13C
Mass
d15N
Diameter
TI
Taper
Length
segmented
Number
Age
flat
ropey
scrape
0
20
40
60
80
100
Importance

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

          Reference
Prediction  1  2  3
         1 14  2  3
         2  0  5  0
         3  0  0  3

Overall Statistics

               Accuracy : 0.8148
                 95% CI : (0.6192, 0.937)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.001421

                  Kappa : 0.6707

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            1.0000   0.7143   0.5000
Specificity            0.6154   1.0000   1.0000
Pos Pred Value         0.7368   1.0000   1.0000
Neg Pred Value         1.0000   0.9091   0.8750
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.5185   0.1852   0.1111
Detection Prevalence   0.7037   0.1852   0.1111
Balanced Accuracy      0.8077   0.8571   0.7500
```

```
#model_nnet
# a) Neuralnet - model summarization
print(model_nnet)

# b) Neuralnet - plot of variable of importance
varimpnnetDF<-varImp(object=model_nnet)
varimpnnetDF_imp<-(varimpnnetDF$importance)
varimpnnetDF_impDF<-data.frame(varimpnnetDF_imp)
varimpnnetDF_impDF<-data.frame(varimpnnetDF_impDF$Overall)
row.names(varimpnnetDF_impDF)<-row.names(varimpnnetDF_imp)
varimpnnetDF_impDF$Variable<-row.names(varimpnnetDF_imp)
names(varimpnnetDF_impDF)<-c("Importance","Variables")
barplot(varimpnnetDF_impDF$Importance, names = varimpnnetDF_impDF$Variables,las=2)
```

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

```
> #model_nnet
> # a) Neuralnet - model summarization
> print(model_nnet)
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  0.5605160  0.2943533
  1     1e-04  0.5827323  0.3185248
  1     1e-01  0.5909552  0.3224521
  3     0e+00  0.6387795  0.4242100
  3     1e-04  0.6374897  0.4136697
  3     1e-01  0.6849031  0.4852554
  5     0e+00  0.6429729  0.4209983
  5     1e-04  0.6536084  0.4369658
  5     1e-01  0.6904109  0.4932604

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.
```

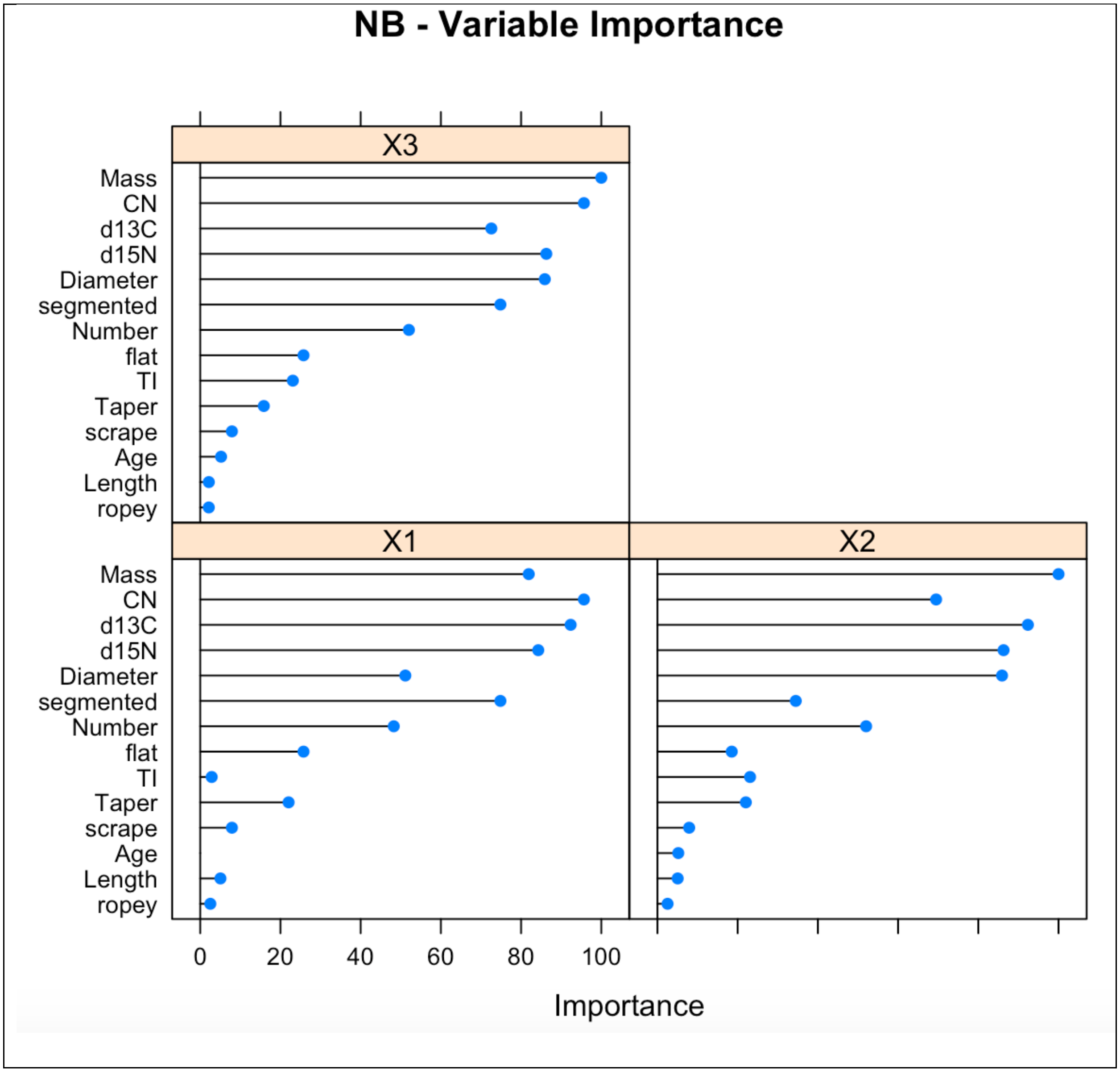

100
80
60
40
20
0
Age
Number
Length
Diameter
Taper
TI
Mass
d13C
d15N
CN
ropey
segmented
flat
scrape

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

          Reference
Prediction  1  2  3
         1 14  1  2
         2  0  5  1
         3  0  1  3

Overall Statistics

               Accuracy : 0.8148
                 95% CI : (0.6192, 0.937)
    No Information Rate : 0.5185
    P-Value [Acc > NIR] : 0.001421

                  Kappa : 0.6824

 Mcnemar's Test P-Value : 0.391625

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            1.0000   0.7143   0.5000
Specificity            0.7692   0.9500   0.9524
Pos Pred Value         0.8235   0.8333   0.7500
Neg Pred Value         1.0000   0.9048   0.8696
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.5185   0.1852   0.1111
Detection Prevalence   0.6296   0.2222   0.1481
Balanced Accuracy      0.8846   0.8321   0.7262
```

```
#model_nb
# a) Naivebayes - model summarization
print(model_nb)

# b) Naivebayes - plot of variable of importance
varImp(object=model_nb)
plot(varImp(object=model_nb),main="NB - Variable Importance")

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

```
> #model_nb
> # a) Naivebayes - model summarization
> print(model_nb)
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      0.6341237  0.4282647
   TRUE      0.6545369  0.4188071

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.
```

```
> # b) Naivebayes - plot of variable of importance
> varImp(object=model_nb)
ROC curve variable importance

  variables are sorted by maximum importance across the classes
              X1      X2      X3
Mass      81.911 100.000 100.000
CN        95.670  69.479  95.670
d13C      92.359  92.359  72.587
d15N      84.284  86.294  86.294
Diameter  51.142  85.913  85.913
segmented 74.845  34.487  74.845
Number    48.256  52.028  52.028
flat      25.757  18.523  25.757
TI         2.860  23.092  23.092
Taper     22.038  22.038  15.858
scrape     7.907   7.907   7.907
Age        0.000   5.197   5.197
Length     5.047   5.047   2.152
ropey      2.523   2.523   2.143
```


NB - Variable Importance
X3
X1
X2
Mass
CN
d13C
d15N
Diameter
segmented
Number
flat
TI
Taper
scrape
Age
Length
ropey
0
20
40
60
80
100
Importance

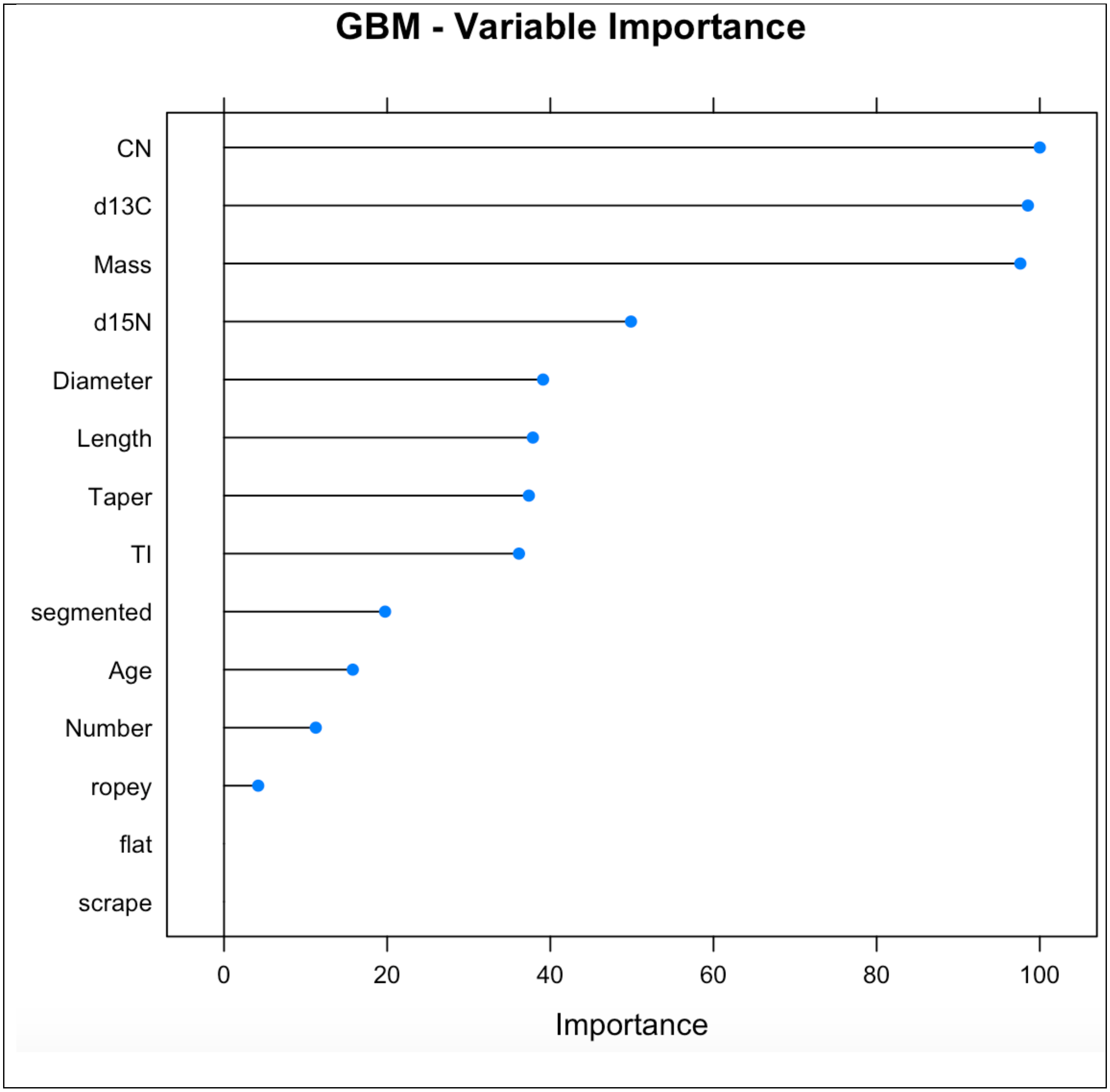
```
> # c) Naivebayes - confusion matrix
> predictions<-predict.train(object=model_nb,testSet[,predictors],type="ra
w")
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3
         1 14  2  2
         2  0  5  0
         3  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: 1 Class: 2 Class: 3
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_gbm
# a) GBM - model summarization
print(model_gbm)

# b) GBM - plot of variable of importance
varImp(object=model_gbm)
plot(varImp(object=model_gbm),main="GBM - Variable Importance")

# c) GBM - confusion matrix
```

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

```
> #model_gbm
> # a) GBM - model summarization
> print(model_gbm)
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      0.6118956  0.3610896
  1                  100      0.5828647  0.3178266
  1                  150      0.5777401  0.3092635
  2                   50      0.6080805  0.3579900
  2                  100      0.5968443  0.3420329
  2                  150      0.6012435  0.3524755
  3                   50      0.6156117  0.3669437
  3                  100      0.6003054  0.3483091
  3                  150      0.5991429  0.3463134

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 = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
> # b) GBM - plot of variable of importance
> varImp(object=model_gbm)
gbm variable importance

          Overall
CN        100.000
d13C       98.541
Mass       97.615
d15N       49.885
Diameter   39.108
Length     37.860
Taper      37.368
TI         36.152
segmented  19.742
Age        15.783
Number     11.247
ropey       4.187
flat        0.000
scrape      0.000
```


GBM - Variable Importance
CN
d13C
Mass
d15N
Diameter
Length
Taper
TI
segmented
Age
Number
ropey
flat
scrape
0
20
40
60
80
100
Importance

```
> # c) GBM - confusion matrix
> predictions<-predict.train(object=model_gbm,testSet[,predictors]
w")
> confusionMatrix(predictions,testSet[,outcomeName])
Confusion Matrix and Statistics

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

Overall Statistics

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

                  Kappa : 0.625

 Mcnemar's Test P-Value : 0.572407

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.9286   0.7143   0.5000
Specificity            0.7692   0.9000   0.9524
Pos Pred Value         0.8125   0.7143   0.7500
Neg Pred Value         0.9091   0.9000   0.8696
Prevalence             0.5185   0.2593   0.2222
Detection Rate         0.4815   0.1852   0.1111
Detection Prevalence   0.5926   0.2593   0.1481
Balanced Accuracy      0.8489   0.8071   0.7262
```

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.

```
########## Answer6

#Finding best model for Randomforest
model_rf_results_df=model_rf$results
model_rf_results_ordered_df=model_rf$results[order(-model_rf_results_df$Accuracy),]
model_rf_results_ordered_AK_df <-
data.frame(model_rf_results_ordered_df$Accuracy,model_rf_results_ordered_df$Kappa)
model_rf_results_ordered_AK_df_r1 <-
data.frame("model_rf",model_rf_results_ordered_AK_df[1,])
names(model_rf_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#Finding best model for Neuralnet
model_nnet_results_df=model_nnet$results
model_nnet_results_ordered_df=model_nnet$results[order(-
model_nnet_results_df$Accuracy),]
model_nnet_results_ordered_AK_df <-
data.frame(model_nnet_results_ordered_df$Accuracy,model_nnet_results_ordered_df$Kap
pa)
model_nnet_results_ordered_AK_df_r1 <-
data.frame("model_nnet",model_nnet_results_ordered_AK_df[1,])
names(model_nnet_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#Finding best model for Naivebayes
model_nb_results_df=model_nb$results
model_nb_results_ordered_df=model_nb$results[order(-model_nb_results_df$Accuracy),]
model_nb_results_ordered_AK_df <-
data.frame(model_nb_results_ordered_df$Accuracy,model_nb_results_ordered_df$Kappa)
model_nb_results_ordered_AK_df_r1 <-
data.frame("model_nb",model_nb_results_ordered_AK_df[1,])
names(model_nb_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#Finding best model for GradientBoosting
model_gbm_results_df=model_gbm$results
model_gbm_results_ordered_df=model_gbm$results[order(-
model_gbm_results_df$Accuracy),]
model_gbm_results_ordered_AK_df <-
data.frame(model_gbm_results_ordered_df$Accuracy,model_gbm_results_ordered_df$Kap
pa)
```

```
model_gbm_results_ordered_AK_df_r1 <-
data.frame("model_gbm",model_gbm_results_ordered_AK_df[1,])
names(model_gbm_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#Integrading best models
summarydf <- rbind(model_gbm_results_ordered_AK_df_r1,
model_rf_results_ordered_AK_df_r1,model_nnet_results_ordered_AK_df_r1,model_nb_res
ults_ordered_AK_df_r1)
summarydf=summarydf[order(-summarydf$Accuracy),]
rownames(summarydf) <- NULL
print(summarydf)
```

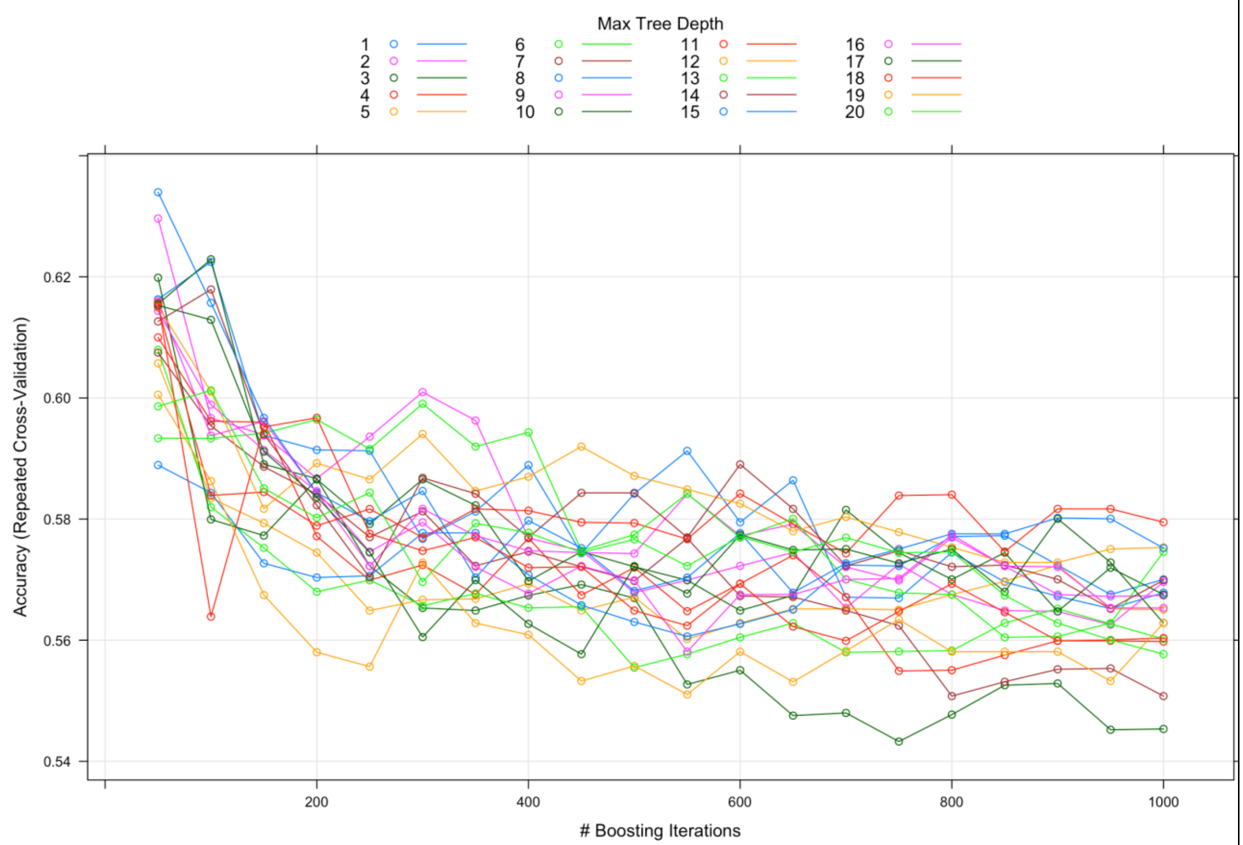
```
> print(summarydf)
       Model  Accuracy     Kappa
1 model_nnet 0.6904109 0.4932604
2   model_nb 0.6545369 0.4188071
3   model_rf 0.6489581 0.3753569
4  model_gbm 0.6156117 0.3669437
```

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

```
########## Answer7
#Tuning GBM - using tune length 20
fitControl <- trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 5)
model_gbm_tl<-
train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',trControl=fitControl,tuneL
ength=20)
print(model_gbm_tl)
plot(model_gbm_tl)

#Finding best model for GradientBoosting
model_gbm_tl_results_df=model_gbm_tl$results
model_gbm_tl_results_ordered_df=model_gbm_tl$results[order(-
model_gbm_tl_results_df$Accuracy),]
model_gbm_tl_results_ordered_AK_df <-
data.frame(model_gbm_tl_results_ordered_df$Accuracy,model_gbm_tl_results_ordered_df
$Kappa)
```

```
model_gbm_tl_results_ordered_AK_df_r1 <-
data.frame("model_gbm_tuneLength20",model_gbm_tl_results_ordered_AK_df[1,])
names(model_gbm_tl_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

summarydf <- rbind(model_gbm_results_ordered_AK_df_r1,
model_rf_results_ordered_AK_df_r1,model_nnet_results_ordered_AK_df_r1,model_nb_res
ults_ordered_AK_df_r1,model_gbm_tl_results_ordered_AK_df_r1)
summarydf=summarydf[order(-summarydf$Accuracy),]
rownames(summarydf) <- NULL
print(summarydf)
```

Iter	TrainDeviance	ValidDeviance	StepSize	Improve
1	1.0986	nan	0.1000	0.1108
2	0.9986	nan	0.1000	0.0881
3	0.9265	nan	0.1000	0.0062
4	0.8844	nan	0.1000	0.0726
5	0.8195	nan	0.1000	0.0321
6	0.7672	nan	0.1000	0.0397
7	0.7088	nan	0.1000	0.0147
8	0.6788	nan	0.1000	0.0244
9	0.6422	nan	0.1000	-0.0239
10	0.6257	nan	0.1000	-0.0113
20	0.4452	nan	0.1000	-0.0130
40	0.2695	nan	0.1000	-0.0226
60	0.1759	nan	0.1000	-0.0285
80	0.1223	nan	0.1000	-0.0082
100	0.0805	nan	0.1000	-0.0038
120	0.0573	nan	0.1000	-0.0109
140	0.0407	nan	0.1000	-0.0021
160	0.0295	nan	0.1000	0.0004
180	0.0214	nan	0.1000	-0.0018
200	0.0148	nan	0.1000	-0.0031
220	0.0112	nan	0.1000	-0.0013
240	0.0078	nan	0.1000	-0.0002
260	0.0051	nan	0.1000	-0.0005
280	0.0035	nan	0.1000	-0.0001
300	0.0023	nan	0.1000	-0.0002
320	0.0015	nan	0.1000	-0.0001
340	0.0011	nan	0.1000	-0.0000
360	0.0008	nan	0.1000	-0.0001
380	0.0005	nan	0.1000	-0.0001
400	0.0004	nan	0.1000	-0.0000
420	0.0003	nan	0.1000	-0.0000
440	0.0002	nan	0.1000	-0.0000
460	0.0001	nan	0.1000	-0.0000
480	0.0001	nan	0.1000	-0.0000
500	0.0001	nan	0.1000	-0.0000
520	0.0001	nan	0.1000	-0.0000
540	0.0000	nan	0.1000	-0.0000

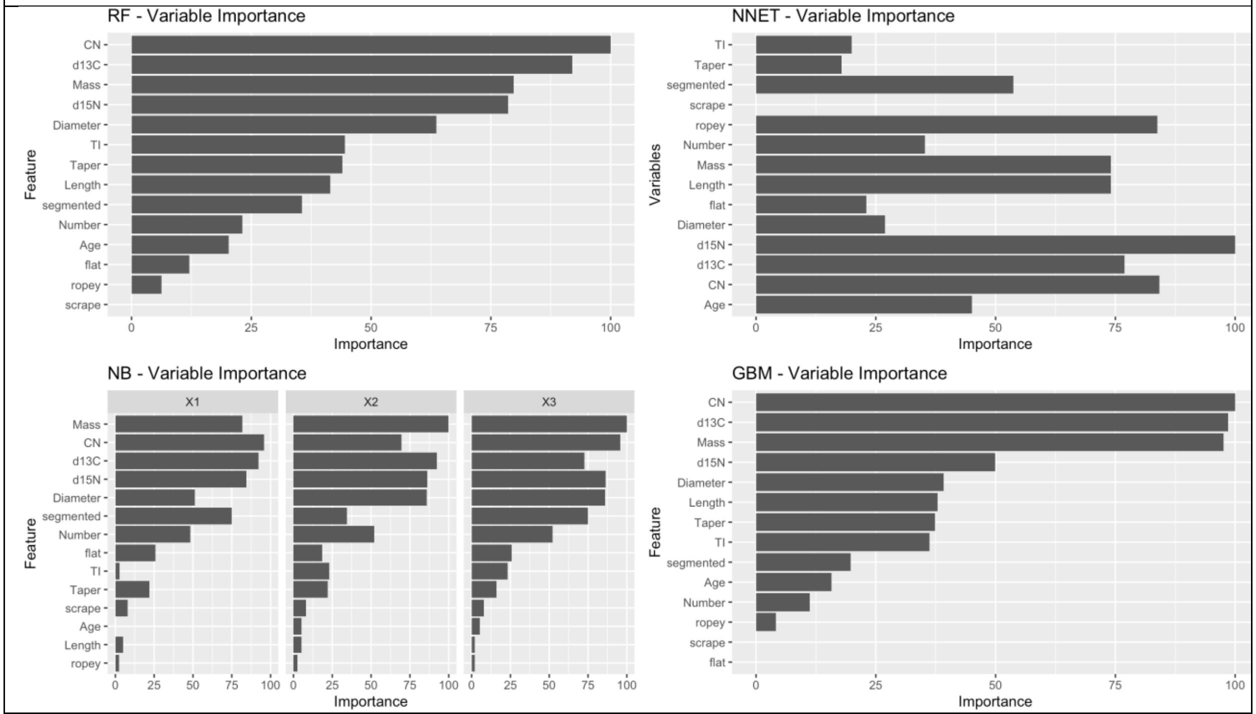



```
> print(summarydf)
                  Model  Accuracy     Kappa
1            model_nnet 0.6904109 0.4932604
2              model_nb 0.6545369 0.4188071
3              model_rf 0.6489581 0.3753569
4 model_gbm_tuneLength20 0.6339706 0.3945501
5             model_gbm 0.6156117 0.3669437
```

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

```
########## Answer8
#all variable of importance plots into one single plot
RF_gg<-ggplot(varImp(object=model_rf))+ggtitle("RF - Variable Importance")
GBM_gg<-ggplot(varImp(object=model_gbm))+ggtitle("GBM - Variable Importance")
NNET_gg<-ggplot(data=varimpnnetDF_impDF, aes(x=Variables, y=Importance)) +
ggtitle("NNET - Variable Importance")+
```

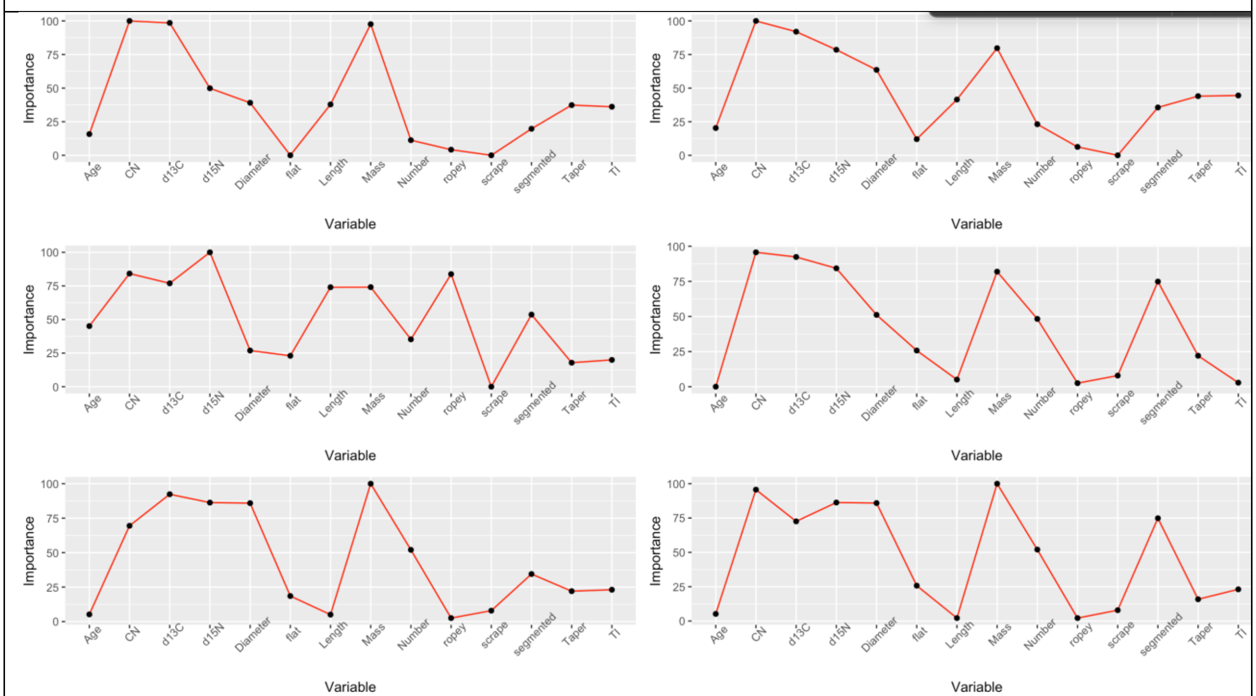
```
  geom_bar(stat="identity") + coord_flip()
NB_gg<-ggplot(varImp(object=model_nb))+ggtitle("NB - Variable Importance")
grid.arrange(RF_gg,NNET_gg,NB_gg,GBM_gg, ncol= 2 )
```




```
########## Answer8 Alternate way
#Plot importance variable
vimp_gbm<-varImp(object=model_gbm)
gbm_DF<-vimp_gbm$importance
gbm_DF$Variable <- rownames(gbm_DF)
gbm_DF$Model <- c("GBM")
names(gbm_DF)<-c("Importance","Variable","Model")

gbm_DF_plot<-ggplot(data=gbm_DF, aes(x=Variable, y=Importance, group=1)) +
  geom_line(color="red")+
  geom_point()+
  theme(axis.text.x = element_text(angle = 45))


vimp_rf<-varImp(object=model_rf)
rf_DF<-vimp_rf$importance
rf_DF$Variable <- rownames(rf_DF)
rf_DF$Model <- c("RF")
names(rf_DF)<-c("Importance","Variable","Model")
```

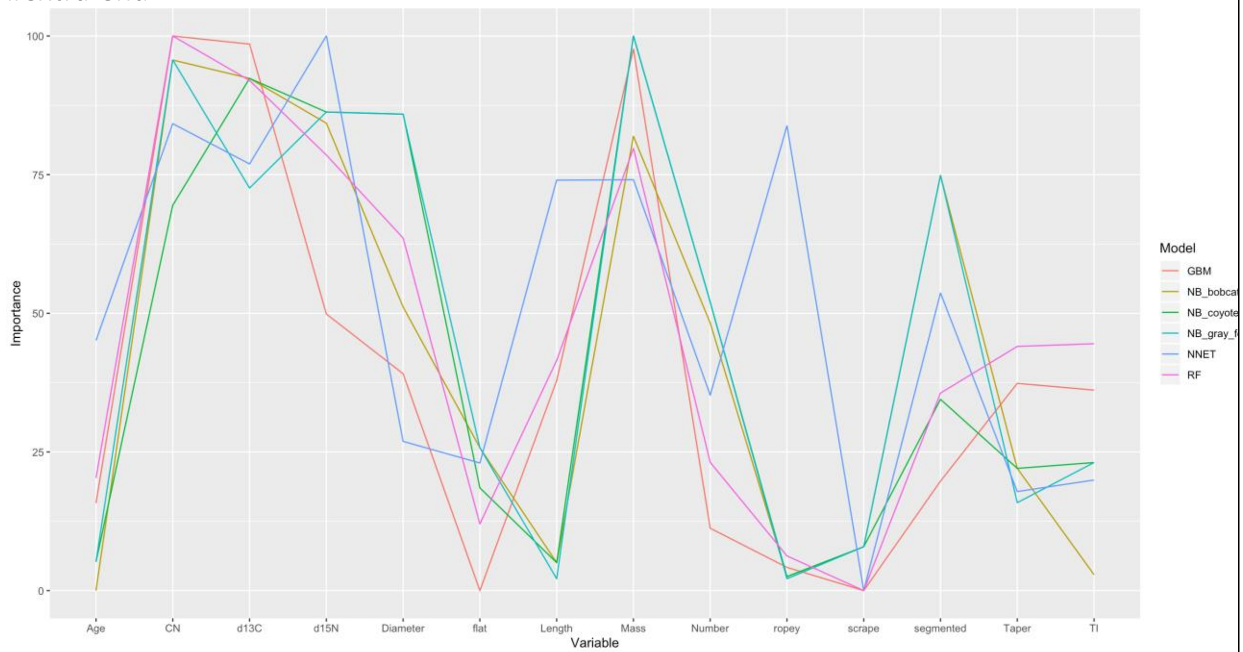
```
rf_DF_plot<-ggplot(data=rf_DF, aes(x=Variable, y=Importance, group=1)) +
  geom_line(color="red")+
  geom_point()+
  theme(axis.text.x = element_text(angle = 45))

vimp_nnet<-varImp(object=model_nnet)
nnet_DF<-vimp_nnet$importance
nnet_DF<-data.frame(nnet_DF)
nnet_DF <- data.frame(nnet_DF$Overall)
nnet_DF$Variable <- rownames(rf_DF)
nnet_DF$Model <- c("NNET")
names(nnet_DF)<-c("Importance","Variable","Model")

nnet_DF_plot<-ggplot(data=nnet_DF, aes(x=Variable, y=Importance, group=1)) +
  geom_line(color="red")+
  geom_point()+
  theme(axis.text.x = element_text(angle = 45))

vimp_nb<-varImp(object=model_nb)
nb_DF<-vimp_nb$importance
nb_bobcat_DF <- data.frame(nb_DF$X1)
nb_bobcat_DF$Variable <- rownames(rf_DF)
nb_bobcat_DF$Model <- c("NB_bobcat")
names(nb_bobcat_DF)<-c("Importance","Variable","Model")

nb_bobcat_DF_plot<-ggplot(data=nb_bobcat_DF, aes(x=Variable, y=Importance, group=1)) +
  geom_line(color="red")+
  geom_point()+
  theme(axis.text.x = element_text(angle = 45))

vimp_nb<-varImp(object=model_nb)
nb_DF<-vimp_nb$importance
nb_coyote_DF <- data.frame(nb_DF$X2)
nb_coyote_DF$Variable <- rownames(rf_DF)
nb_coyote_DF$Model <- c("NB_coyote")
names(nb_coyote_DF)<-c("Importance","Variable","Model")

nb_coyote_DF_plot<-ggplot(data=nb_coyote_DF, aes(x=Variable, y=Importance, group=1)) +
  geom_line(color="red")+
  geom_point()+
  theme(axis.text.x = element_text(angle = 45))

vimp_nb<-varImp(object=model_nb)
```

```
nb_DF<-vimp_nb$importance
nb_gray_fox_DF <- data.frame(nb_DF$X3)
nb_gray_fox_DF$Variable <- rownames(rf_DF)
nb_gray_fox_DF$Model <- c("NB_gray_fox")
names(nb_gray_fox_DF)<-c("Importance","Variable","Model")

nb_gray_fox_DF_plot<-ggplot(data=nb_gray_fox_DF, aes(x=Variable, y=Importance,
group=1)) +
  geom_line(color="red")+
  geom_point()+
  theme(axis.text.x = element_text(angle = 45))

grid.arrange(gbm_DF_plot,rf_DF_plot,nnet_DF_plot,nb_bobcat_DF_plot,nb_coyote_DF_plot,
nb_gray_fox_DF_plot, ncol= 2 )
```




```
#extra
varimpdf <- rbind(gbm_DF, rf_DF,nnet_DF,nb_bobcat_DF,nb_coyote_DF,nb_gray_fox_DF)
print(varimpdf)
varimpdf=varimpdf[order(-varimpdf$Importance),]

ggplot(data=varimpdf) +
  geom_line(aes(x=Variable, y=Importance, group=Model, color=Model ))
```

#extra-end



########## Answer9
With the below results:
Model Accuracy   Kappa
3   model_nnet 0.6908921 0.4788325
5 model_gbm_tuneLength20 0.6847222 0.4705875
4   model_nb 0.6624052 0.4391168
2   model_rf 0.6507490 0.4248715
1   model_gbm 0.6275335 0.3671868
We can say that best performing model is of Neural Net. As it has the highest accuracy in comparision to other model's accuracy when taken the best parameters of for the models.
Hence, we can say that we can predict Species with 69% accuracy.

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

########## 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

```
########## Answer10-a
#RFE feature selection
control <- rfeControl(functions = rfFuncs,
                method = "repeatedcv",
                repeats = 3,
                verbose = FALSE)
outcomeName<-'Species'
Loan_Pred_Profile <- rfe(trainSet[,predictors], trainSet[,outcomeName],rfeControl = control)
optVariables=Loan_Pred_Profile$optVariables
predictors<-head(optVariables,n=3)
print(predictors)
> print(predictors)
[1] "CN"   "d13C" "d15N"
```

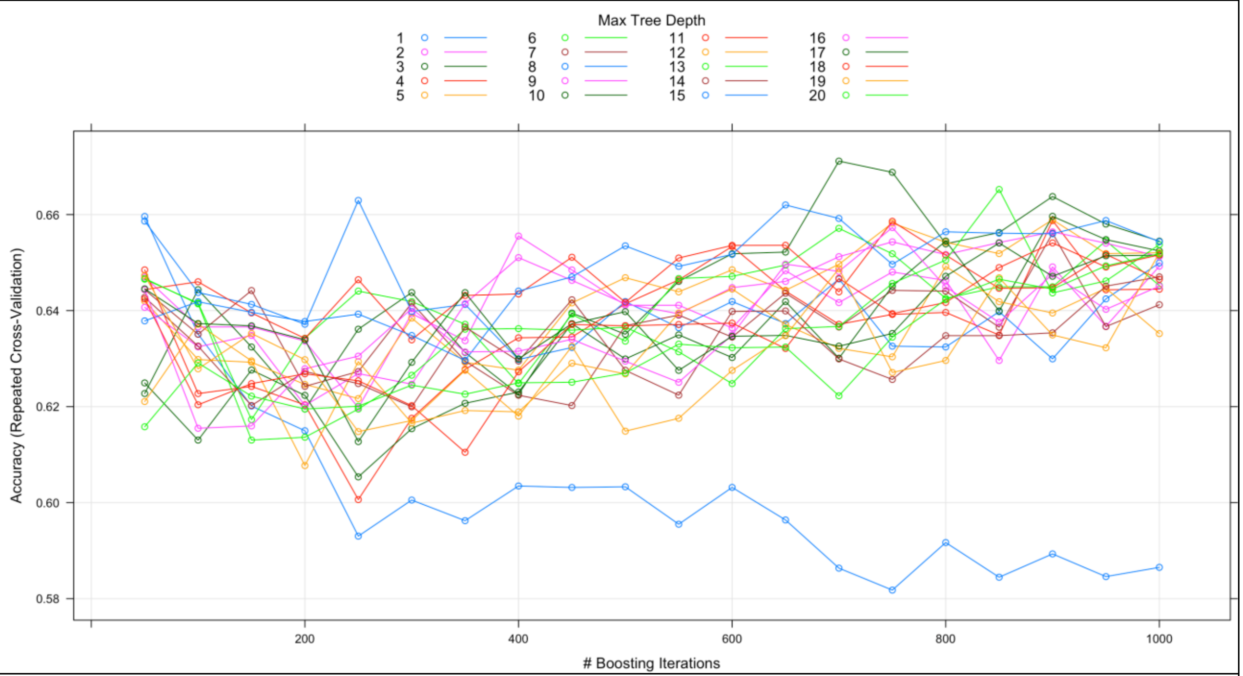
```
##Repeating 6 to 8

##### Repeating Answer6 with Feature Selection
#Creating Models with Feature Selection
fs_model_gbm<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
fs_model_rf<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf')
fs_model_nnet<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet')
fs_model_nb<-train(trainSet[,predictors],trainSet[,outcomeName],method='nb')

#model_gbm
fs_model_gbm_results_df<-fs_model_gbm$results
fs_model_gbm_results_ordered_df<-fs_model_gbm$results[order(-
fs_model_gbm_results_df$Accuracy),]
fs_model_gbm_results_ordered_AK_df <-
data.frame(fs_model_gbm_results_ordered_df$Accuracy,fs_model_gbm_results_ordered_df
$Kappa)
fs_model_gbm_results_ordered_AK_df_r1 <-
data.frame("fs_model_gbm",fs_model_gbm_results_ordered_AK_df[1,])
names(fs_model_gbm_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#model_rf
fs_model_rf_results_df=fs_model_rf$results
fs_model_rf_results_ordered_df=fs_model_rf$results[order(-
fs_model_rf_results_df$Accuracy),]
fs_model_rf_results_ordered_AK_df <-
data.frame(fs_model_rf_results_ordered_df$Accuracy,fs_model_rf_results_ordered_df$Kap
pa)
```

```
fs_model_rf_results_ordered_AK_df_r1 <-
data.frame("fs_model_rf",fs_model_rf_results_ordered_AK_df[1,])
names(fs_model_rf_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#model_nnet
fs_model_nnet_results_df=fs_model_nnet$results
fs_model_nnet_results_ordered_df=fs_model_nnet$results[order(-
fs_model_nnet_results_df$Accuracy),]
fs_model_nnet_results_ordered_AK_df <-
data.frame(fs_model_nnet_results_ordered_df$Accuracy,fs_model_nnet_results_ordered_d
f$Kappa)
fs_model_nnet_results_ordered_AK_df_r1 <-
data.frame("fs_model_nnet",fs_model_nnet_results_ordered_AK_df[1,])
names(fs_model_nnet_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")

#model_nb
fs_model_nb_results_df=fs_model_nb$results
fs_model_nb_results_ordered_df=fs_model_nb$results[order(-
fs_model_nb_results_df$Accuracy),]
fs_model_nb_results_ordered_AK_df <-
data.frame(fs_model_nb_results_ordered_df$Accuracy,fs_model_nb_results_ordered_df$Ka
ppa)
fs_model_nb_results_ordered_AK_df_r1 <-
data.frame("fs_model_nb",fs_model_nb_results_ordered_AK_df[1,])
names(fs_model_nb_results_ordered_AK_df_r1)<-c("Model","Accuracy","Kappa")


fs_summarydf <- rbind(fs_model_gbm_results_ordered_AK_df_r1,
fs_model_rf_results_ordered_AK_df_r1,fs_model_nnet_results_ordered_AK_df_r1,fs_model
_nb_results_ordered_AK_df_r1)
fs_summarydf=fs_summarydf[order(-fs_summarydf$Accuracy),]
print(fs_summarydf)
```

```
> print(fs_summarydf)
          Model  Accuracy     Kappa
1   fs_model_nb 0.7273329 0.5215849
2 fs_model_nnet 0.7190828 0.5213849
3   fs_model_rf 0.6825855 0.4629996
4  fs_model_gbm 0.6157889 0.3624927
```

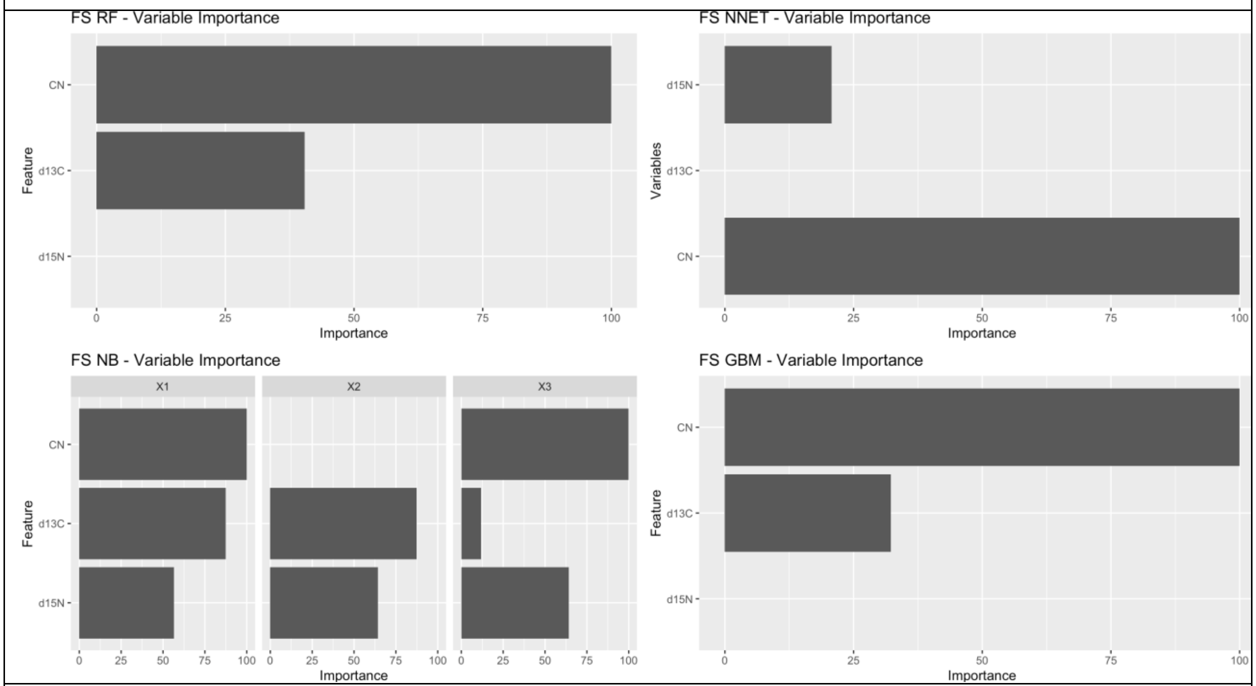
Repeating Answer7 with Feature Selection

```
#Tuning GBM - using tune length 20
fitControl <- trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 5)
model_gbm_tl<-
train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',trControl=fitControl,tuneL
ength=20)
print(model_gbm_tl)
plot(model_gbm_tl)
```




```
##### Repeating Answer8 with Feature Selection

fs_varimpnnetDF<-varImp(object=fs_model_nnet)
fs_varimpnnetDF_imp<-(fs_varimpnnetDF$importance)
fs_varimpnnetDF_impDF<-data.frame(fs_varimpnnetDF_imp)
fs_varimpnnetDF_impDF<-data.frame(fs_varimpnnetDF_impDF$Overall)
row.names(fs_varimpnnetDF_impDF)<-row.names(fs_varimpnnetDF_imp)
fs_varimpnnetDF_impDF$Variable<-row.names(fs_varimpnnetDF_imp)
names(fs_varimpnnetDF_impDF)<-c("Importance","Variables")
fs_varimpnnetDF_impDF

RF_gg<-ggplot(varImp(object=fs_model_rf))+ggtitle("FS RF - Variable Importance")
GBM_gg<-ggplot(varImp(object=fs_model_gbm))+ggtitle("FS GBM - Variable Importance")
```

```
NNET_gg<-ggplot(data=fs_varimpnnetDF_impDF, aes(x=Variables, y=Importance)) +
ggtitle("FS NNET - Variable Importance")+
  geom_bar(stat="identity") + coord_flip()
NB_gg<-ggplot(varImp(object=fs_model_nb))+ggtitle("FS NB - Variable Importance")
grid.arrange(RF_gg,NNET_gg,NB_gg,GBM_gg, ncol= 2 )

##Repeating 6 to 8 - end
```



########## 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.

```
########## Answer10-b
comb_summaryDF <- rbind(fs_summarydf, summarydf)
comb_summaryDF=comb_summaryDF[order(-comb_summaryDF$Accuracy),]
print(comb_summaryDF)
```

```
> print(comb_summaryDF)
                 Model  Accuracy     Kappa
1          fs_model_nb 0.7273329 0.5215849
2        fs_model_nnet 0.7190828 0.5213849
3           model_nnet 0.6904109 0.4932604
4          fs_model_rf 0.6825855 0.4629996
5             model_nb 0.6545369 0.4188071
6             model_rf 0.6489581 0.3753569
7 model_gbm_tuneLength20 0.6339706 0.3945501
8         fs_model_gbm 0.6157889 0.3624927
9            model_gbm 0.6156117 0.3669437
```

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

########## Answer10-c
With the below results:
Model Accuracy   Kappa
1   fs_model_nb 0.7273329 0.5215849
2   fs_model_nnet 0.7190828 0.5213849
3   model_nnet 0.6904109 0.4932604
4   fs_model_rf 0.6825855 0.4629996
5   model_nb 0.6545369 0.4188071
6   model_rf 0.6489581 0.3753569
7 model_gbm_tuneLength20 0.6339706 0.3945501
8   fs_model_gbm 0.6157889 0.3624927
9   model_gbm 0.6156117 0.3669437
We can say that best performing model is of Naive Bayes model build with top 3 predictors. As it has the highest accuracy in comparision to other model's accuracy .
Hence, we can say that we can predict Species with 75.76% accuracy.