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

```
library(caret)
library(gbm)
data(scat)
data<-scat

#######Questions
#1
S<-as.factor(data[,'Species'])
data['Species']<-unclass(S)
###Bobcat=1, Coyote=2, Gray_Fox=3
#summary(data)
```

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

```
#2
data=data[,-(2:5)]
```

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

```
#3 & 4
sum(is.na(data))
Output:
[1] 47
#47 nulls
ma<-data.matrix(data)

preProcValues <- preProcess(ma, method = c("knnImpute"))#knnImpute forces scale and center
data_processed <- predict(preProcValues, ma)
sum(is.na(data_processed))
Output:
[1] 0
data_processed[,1]=as.factor(ma[,1])## setting target variable back to orginal values->1,2,3
```

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

```
#No Categorical Values
data_new<-data.frame(data_processed)
data_new$Species<-as.factor(data_new$Species)## keep target as categorical for classification
```

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

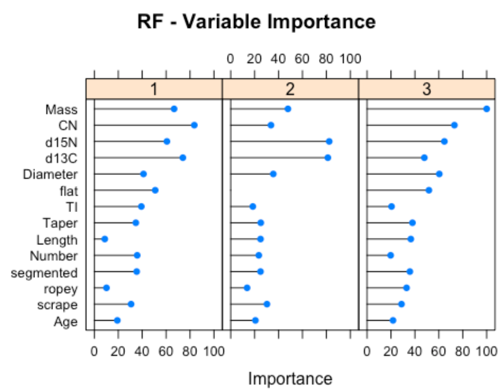
a. For these models display a)model summarization and b) plot variable of importance, for the predictions (use the prediction set) display c) confusion matrix (60 points)

```
set.seed(100)
```

```
index <- createDataPartition(data_new$Species, p=0.75, list=FALSE)
trainSet <- data_new[ index,]
testSet <- data_new[-index,]



############Random Forest
#Fit Model
model_rf<-train(trainSet[,-1],trainSet[,1],method='rf', importance=T)

#Model Summary
print(model_rf)

#Feature importance
plot(varImp(object=model_rf),main="RF - Variable Importance")

#Confusion Matrix
predictions<-predict.train(object=model_rf,testSet[,-1],type="raw")
table(predictions)

confusionMatrix(predictions,testSet[,1])$table
```

Output:

```
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    0.7034502  0.4870366
   8    0.6936636  0.4855754
  14    0.6811908  0.4724359

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




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

```
##################Neural Network
#Fit Model
model_nnet<-train(trainSet[,-1],trainSet[,1],method='nnet', importance=T)

#Model Summary
print(model_nnet)

#Feature importance
#reformat varImp Importance for plotting
c<-varImp(object=model_nnet)
v<-c$importance
c$importance<-as.data.frame(v)

plot(c,main="Neural Net - Variable Importance")

#Confusion Matrix
predictions<-predict.train(object=model_nnet,testSet[,-1],type="raw")
table(predictions)

confusionMatrix(predictions,testSet[,1])$table
```

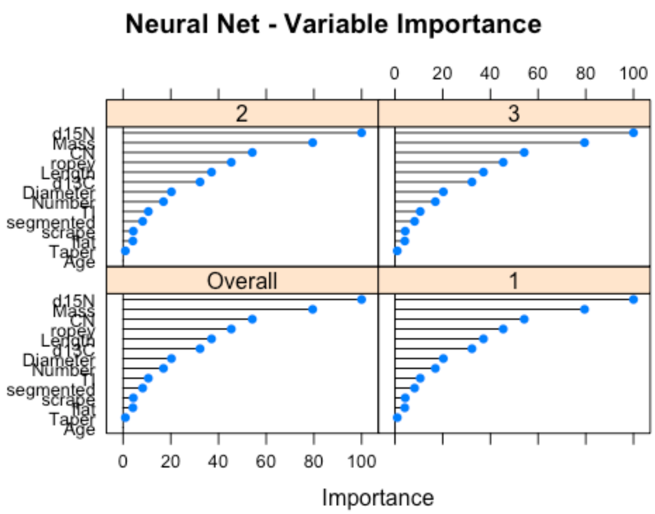
Output:

```
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	0.5488713	0.2627464
1	1e-04	0.5856552	0.3081288
1	1e-01	0.5853458	0.3162800
3	0e+00	0.6553875	0.4442270
3	1e-04	0.6796926	0.4910612
3	1e-01	0.7014925	0.5175975
5	0e+00	0.6575731	0.4595065
5	1e-04	0.6643261	0.4556073
5	1e-01	0.7089062	0.5246513

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.



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

```
#################Naive Bayes
model_nb<-train(trainSet[,-1],trainSet[,1],method='naive_bayes', importance=T)

#Model Summary
print(model_nb)

#Feature importance
plot(varImp(object=model_nb),main="Naive Bayes - Variable Importance")
```

```
#Confusion Matrix
predictions<-predict.train(object=model_nb,testSet[,-1],type="raw")
table(predictions)

confusionMatrix(predictions,testSet[,1])$table
```

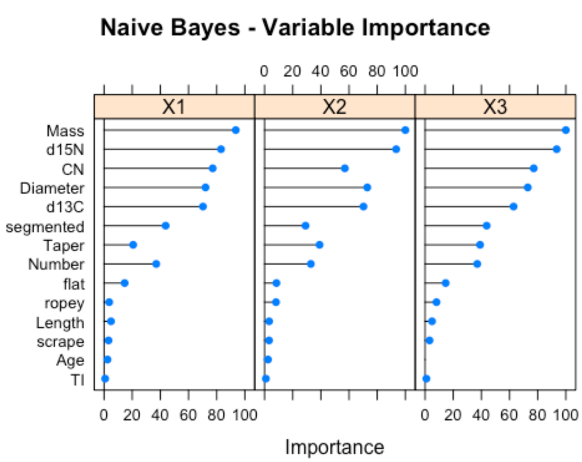
Output:
Naive Bayes

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	0.6312607	0.4317705
TRUE	0.6743543	0.4366177

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.

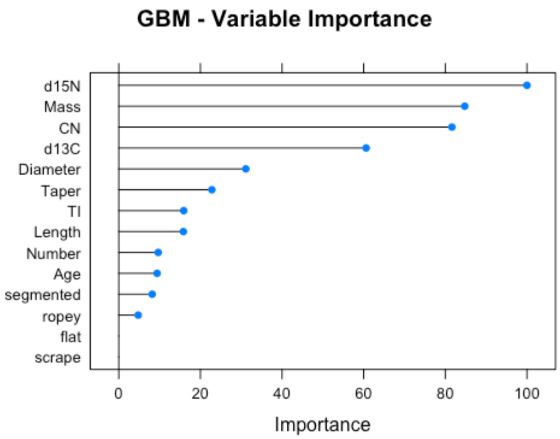


	Reference		
Prediction	1	2	3
1	13	4	0
2	0	3	2
3	1	0	4

```
################GBM
model_gbm<-train(trainSet[,-1],trainSet[,1],method='gbm',distribution = "multinomial")

#Model Summary
print(model_gbm)

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

#Confusion Matrix
predictions<-predict.train(object=model_gbm,testSet[,-1],type="raw")
table(predictions)

confusionMatrix(predictions,testSet[,1])$table
```

Output:

```
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      0.6957848  0.4885577
  1                  100      0.6960712  0.4909677
  1                  150      0.6860903  0.4722038
  2                   50      0.6821648  0.4682668
  2                  100      0.6819343  0.4695772
  2                  150      0.6863378  0.4781445
  3                   50      0.6964489  0.4913461
  3                  100      0.6900058  0.4799270
  3                  150      0.6913272  0.4845239

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.
```




```
        Reference
Prediction  1  2  3
         1 10  2  1
         2  2  5  3
         3  2  0  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)

```
#6
##Extract metrics from models
results<-data.frame(ExperimentName=c('RF','NNet', 'NB','GBM'),
                Accuracy=c(model_rf$results[order(model_rf$results$Accuracy,decreasing = T),]['Accuracy'][1,1],model_nnet$results[order(model_nnet$results$Accuracy,decreasing = T),]['Accuracy'][1,1],
                         model_nb$results[order(model_nb$results$Accuracy,decreasing = T),]['Accuracy'][1,1],model_gbm$results[order(model_gbm$results$Accuracy,decreasing = T),]['Accuracy'][1,1]),
                Kappa=c(model_rf$results[order(model_rf$results$Accuracy,decreasing = T),]['Kappa'][1,1],model_nnet$results[order(model_nnet$results$Accuracy,decreasing = T),]['Kappa'][1,1],
                      model_nb$results[order(model_nb$results$Accuracy,decreasing = T),]['Kappa'][1,1],model_gbm$results[order(model_gbm$results$Accuracy,decreasing = T),]['Kappa'][1,1]))
##Sort on Accuracy
results[order(results$Accuracy,decreasing = T),]
```

Output:

	ExperimentName	Accuracy	Kappa
2	NNet	0.7089062	0.5246513
1	RF	0.7034502	0.4870366
4	GBM	0.6964489	0.4913461
3	NB	0.6743543	0.4366177

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

```
fitControl <- trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 5)

gbm_tuned<-train(trainSet[,-1],trainSet[,1],method='gbm',distribution =
"multinomial",trControl=fitControl, tuneLength = 20)
print(gbm_tuned)
plot(gbm_tuned)
```

output:
Stochastic Gradient Boosting

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

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 5 times)
Summary of sample sizes: 66, 67, 65, 66, 68, 66, ...
Resampling results across tuning parameters:

interaction.depth	n.trees	Accuracy	Kappa
1	50	0.6975065	0.4947665
1	100	0.7090098	0.5192320
1	150	0.7067876	0.5108496
1	200	0.7228301	0.5387546
1	250	0.7165294	0.5309002
1	300	0.7141797	0.5235982
1	350	0.7141961	0.5258301
1	400	0.7137516	0.5243096
1	450	0.7067876	0.5141162
1	500	0.7044542	0.5136034
1	550	0.7022484	0.5087906
1	600	0.7071013	0.5134934
1	650	0.7116765	0.5214893
1	700	0.7069706	0.5131773
1	750	0.7116765	0.5219700

1	800	0.7116765	0.5216205
1	850	0.7069706	0.5153552
1	900	0.7094542	0.5204611
1	950	0.7045817	0.5106522
1	1000	0.6970980	0.4984337
2	50	0.7042876	0.5066513
2	100	0.6941569	0.4940327
2	150	0.6944510	0.4980906
2	200	0.6996176	0.5040357
2	250	0.7039935	0.5102322
2	300	0.7014935	0.5131776
2	350	0.6990098	0.5091299
2	400	0.7010654	0.5069559
2	450	0.7018235	0.5104905
2	500	0.7063824	0.5175341
2	550	0.6914739	0.4931444
2	600	0.6960490	0.5010905
2	650	0.6888235	0.4867113
2	700	0.6913235	0.4909149
2	750	0.6966569	0.5014496
2	800	0.6969706	0.5032946
2	850	0.6963627	0.5060253
2	900	0.6891373	0.4920029
2	950	0.6891373	0.4903242
2	1000	0.6985490	0.5077055
3	50	0.6928007	0.4948914
3	100	0.7020850	0.5137833
3	150	0.7088627	0.5233912
3	200	0.7065098	0.5191514
3	250	0.6963595	0.5028934
3	300	0.7013595	0.5099349
3	350	0.7062157	0.5192199
3	400	0.7087320	0.5245505
3	450	0.7208105	0.5470349
3	500	0.7206634	0.5468241
3	550	0.7104771	0.5290506
3	600	0.7159575	0.5386503
3	650	0.7035490	0.5176661
3	700	0.6917680	0.4975468
3	750	0.6991405	0.5104612
3	800	0.7038627	0.5177501
3	850	0.7038627	0.5164525
3	900	0.7018072	0.5147507
3	950	0.6991405	0.5086673
3	1000	0.6916373	0.4947966
4	50	0.6803333	0.4694878
4	100	0.6878007	0.4846554
4	150	0.6948431	0.5006913

4	200	0.6995654	0.5050337
4	250	0.6945817	0.4982783
4	300	0.7011961	0.5091187
4	350	0.6988595	0.5048635
4	400	0.7010654	0.5114764
4	450	0.7032712	0.5157545
4	500	0.6988431	0.5041491
4	550	0.6940065	0.4965678
4	600	0.6935817	0.4971281
4	650	0.7032712	0.5106350
4	700	0.7029935	0.5134694
4	750	0.6960654	0.5016287
4	800	0.6960654	0.4996695
4	850	0.7009183	0.5081195
4	900	0.6888595	0.4885959
4	950	0.6937124	0.4975422
4	1000	0.7035850	0.5151427
5	50	0.7036601	0.5086148
5	100	0.7036961	0.5096124
5	150	0.7034183	0.5068474
5	200	0.7040458	0.5108802
5	250	0.7012320	0.5051177
5	300	0.6996176	0.5057487
5	350	0.6919510	0.4951562
5	400	0.6965261	0.5031792
5	450	0.7015458	0.5104641
5	500	0.7115131	0.5266616
5	550	0.6996176	0.5096908
5	600	0.6993039	0.5081604
5	650	0.6922288	0.4955037
5	700	0.6992876	0.5050766
5	750	0.6992876	0.5063032
5	800	0.6968039	0.5028438
5	850	0.6944510	0.4994791
5	900	0.7037320	0.5134073
5	950	0.6990261	0.5067532
5	1000	0.6940065	0.4979000
6	50	0.6707222	0.4522033
6	100	0.6969510	0.4936042
6	150	0.6919314	0.4876464
6	200	0.6798725	0.4694237
6	250	0.6869314	0.4804099
6	300	0.6963431	0.4980799
6	350	0.6941569	0.4945216
6	400	0.6894510	0.4904441
6	450	0.6918039	0.4926699
6	500	0.6919510	0.4930107
6	550	0.6991569	0.5062446

6	600	0.6944510	0.4982665
6	650	0.6944510	0.4977665
6	700	0.6872451	0.4862205
6	750	0.6947647	0.4990608
6	800	0.6966569	0.5016959
6	850	0.6921176	0.4960918
6	900	0.6972843	0.5048541
6	950	0.6968235	0.5053021
6	1000	0.6993235	0.5083790
7	50	0.6851699	0.4783565
7	100	0.6909804	0.4854733
7	150	0.6933007	0.4936439
7	200	0.6901895	0.4853590
7	250	0.6900425	0.4915977
7	300	0.6903366	0.4904795
7	350	0.7019542	0.5112397
7	400	0.6996536	0.5066485
7	450	0.7071373	0.5198884
7	500	0.7090294	0.5216619
7	550	0.7069902	0.5192847
7	600	0.7068431	0.5183221
7	650	0.6925425	0.4942657
7	700	0.6950425	0.4989552
7	750	0.6996176	0.5052452
7	800	0.6950425	0.4969074
7	850	0.6925588	0.4922630
7	900	0.6949118	0.4965350
7	950	0.6973954	0.5015124
7	1000	0.6947647	0.4981656
8	50	0.6975065	0.5008439
8	100	0.6959150	0.5020423
8	150	0.6993039	0.5056872
8	200	0.6993399	0.5054318
8	250	0.7018758	0.5106111
8	300	0.7018758	0.5116394
8	350	0.6970229	0.5037228
8	400	0.7062876	0.5205859
8	450	0.6990621	0.5074573
8	500	0.6964150	0.5040688
8	550	0.7014150	0.5113329
8	600	0.6919869	0.4959288
8	650	0.6990621	0.5086380
8	700	0.6919869	0.4972091
8	750	0.6844673	0.4858802
8	800	0.6918562	0.4975561
8	850	0.6871144	0.4897239
8	900	0.6968399	0.5071572
8	950	0.6921340	0.5000724

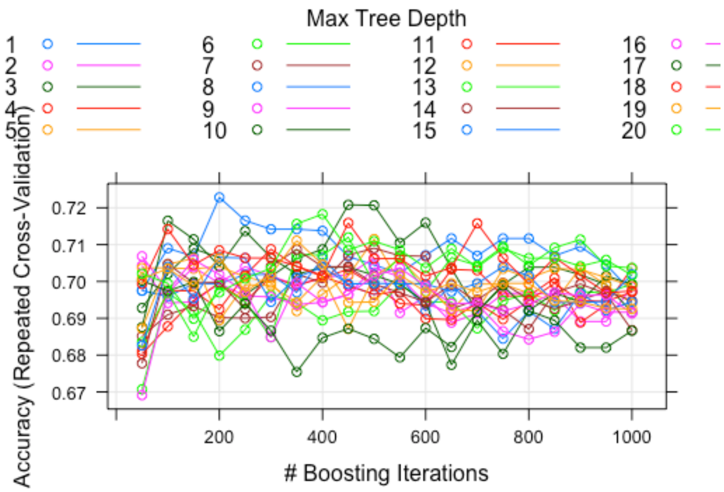
8	1000	0.6944869	0.5051209
9	50	0.6691242	0.4494846
9	100	0.6969542	0.4997450
9	150	0.7038627	0.5126336
9	200	0.6994706	0.5056790
9	250	0.6960654	0.5023587
9	300	0.6957876	0.5012162
9	350	0.6937320	0.5001092
9	400	0.6943235	0.4988684
9	450	0.6963791	0.5015879
9	500	0.7037320	0.5144565
9	550	0.7012320	0.5107880
9	600	0.6940261	0.4983683
9	650	0.6913595	0.4955290
9	700	0.6938431	0.4995719
9	750	0.6864902	0.4910197
9	800	0.6842647	0.4832556
9	850	0.6863235	0.4870945
9	900	0.6939902	0.5029951
9	950	0.6916373	0.4995276
9	1000	0.6916373	0.4942346
10	50	0.6875980	0.4814446
10	100	0.7164804	0.5313400
10	150	0.7114608	0.5243784
10	200	0.7041242	0.5120924
10	250	0.7136830	0.5264590
10	300	0.7064771	0.5156320
10	350	0.7019379	0.5117235
10	400	0.7038464	0.5158154
10	450	0.7038824	0.5166122
10	500	0.7016765	0.5104352
10	550	0.6993235	0.5096980
10	600	0.6941569	0.4998785
10	650	0.6773562	0.4722087
10	700	0.6894346	0.4942235
10	750	0.6803366	0.4763402
10	800	0.6921176	0.4966824
10	850	0.6968235	0.5069912
10	900	0.6944706	0.5027025
10	950	0.7015294	0.5158790
10	1000	0.6991765	0.5124078
11	50	0.6811667	0.4701098
11	100	0.6975588	0.5030369
11	150	0.6974118	0.5007676
11	200	0.6924281	0.4951149
11	250	0.7016732	0.5121296
11	300	0.7087680	0.5199058
11	350	0.7040098	0.5172635

11	400	0.7013595	0.5097393
11	450	0.6988595	0.5056806
11	500	0.6965065	0.5027812
11	550	0.6972647	0.5059763
11	600	0.6898922	0.4925139
11	650	0.6895980	0.4907782
11	700	0.6941732	0.4980474
11	750	0.6897451	0.4917802
11	800	0.6947810	0.4985452
11	850	0.6994869	0.5073165
11	900	0.6947647	0.5001832
11	950	0.6946176	0.5003092
11	1000	0.6971176	0.5051140
12	50	0.6872255	0.4787915
12	100	0.7046373	0.5085955
12	150	0.7062680	0.5167919
12	200	0.6894673	0.4864209
12	250	0.6991569	0.5028748
12	300	0.6988431	0.5007498
12	350	0.6943203	0.4952910
12	400	0.7013791	0.5099900
12	450	0.6871144	0.4836384
12	500	0.7012320	0.5081947
12	550	0.7060850	0.5155898
12	600	0.6990098	0.5022634
12	650	0.6922451	0.4909388
12	700	0.6920980	0.4935097
12	750	0.7034183	0.5116965
12	800	0.7009183	0.5062372
12	850	0.7034379	0.5117087
12	900	0.7032712	0.5099226
12	950	0.7010490	0.5055384
12	1000	0.6987124	0.5005694
13	50	0.6835196	0.4713577
13	100	0.6969510	0.4967528
13	150	0.6850033	0.4795503
13	200	0.6989902	0.5051886
13	250	0.6944510	0.4960639
13	300	0.6848725	0.4821562
13	350	0.7072843	0.5194727
13	400	0.7071373	0.5169885
13	450	0.7119739	0.5286457
13	500	0.6995817	0.5068866

[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

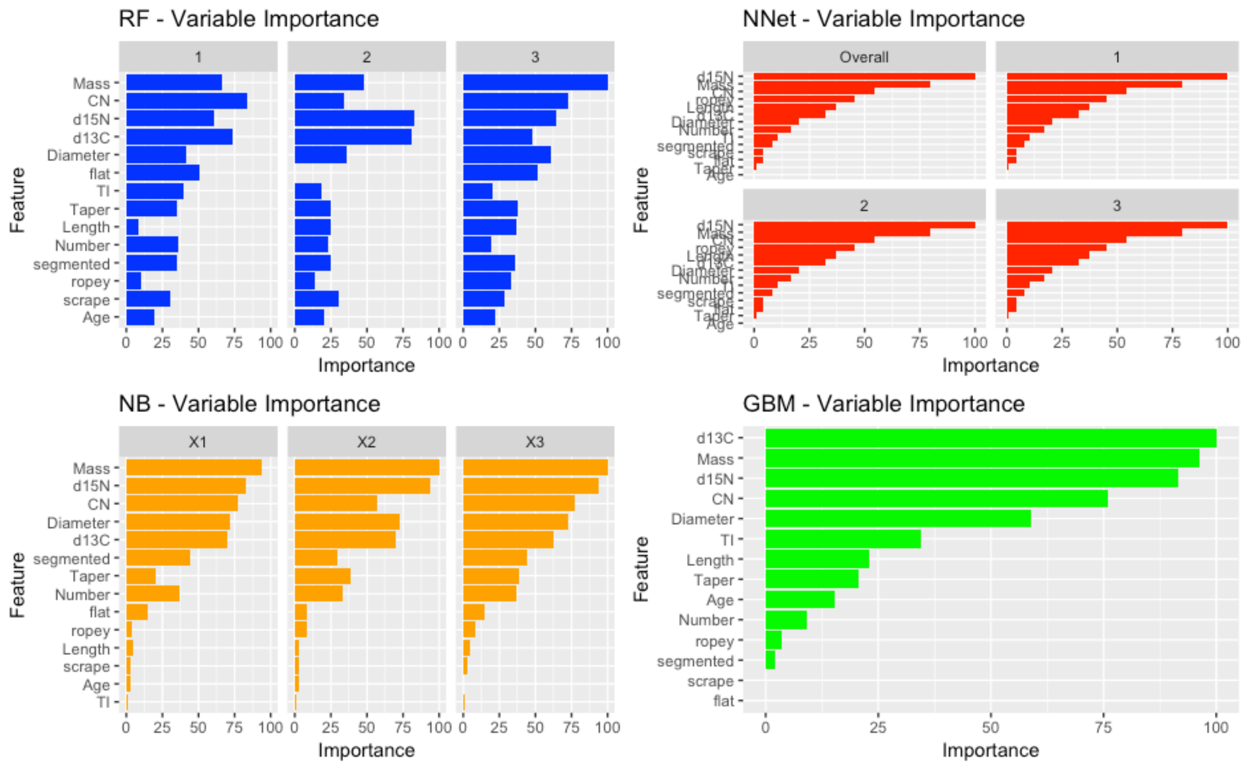
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 200, interaction.depth = 1,
shrinkage = 0.1 and n.minobsinnode = 10.



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

```
library(ggplot2)
library(gridExtra)
p1<-ggplot(varImp(object=model_rf))+ggtitle("RF - Variable Importance")+geom_col(fill='blue')
p2<-ggplot(c)+ggtitle("NNet - Variable Importance")+geom_col(fill='red')
p3<-ggplot(varImp(object=model_nb))+ggtitle("NB - Variable Importance")+geom_col(fill='orange')
p4<-ggplot(varImp(object=gbm_tuned))+ggtitle("GBM - Variable Importance")+geom_col(fill='green')

grid.arrange(p1, p2, p3, p4, ncol=2)
```

Output:



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

```
#9
new.row<-data.frame(ExperimentName=c('GBM Tuned'),
            Accuracy=c(gbm_tuned$results[order(gbm_tuned$results$Accuracy,decreasing =
T),]['Accuracy'][1,1]),
            Kappa=c(gbm_tuned$results[order(gbm_tuned$results$Accuracy,decreasing =
T),]['Kappa'][1,1]))
results<-rbind(results,new.row)
results[order(results$Accuracy,decreasing = T),]
```

	ExperimentName	Accuracy	Kappa
5	GBM Tuned	0.7228301	0.5387546
2	NNet	0.7089062	0.5246513
1	RF	0.7034502	0.4870366
4	GBM	0.6964489	0.4913461
3	NB	0.6743543	0.4366177

Answer:
##Based on accuracy, GBM Tuned performs the best on this data. Boosting algorithm like Random Forest
#is an ensemble method, the difference being that predictors are made sequentially not independently.
#Random Forest can outperform GBM but its random nature does not guarentee it. Our accuracy ~64-73% suggests

#our models can predict better than a 1 in 3 guess. However,in my opinion, 70% might be too low to be #reliable and more models should be explored to find a more accurate model.

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

```
#a
control <- rfeControl(functions = rfFuncs,
                  method = "repeatedcv",
                  repeats = 3,
                  verbose = FALSE)
outcomeName<-'Species'
predictors<-names(trainSet)[!names(trainSet) %in% outcomeName]
spec_Pred_Profile <- rfe(trainSet[,predictors], trainSet[,outcomeName],rfeControl = control)
spec_Pred_Profile

#Top 3 predictors
predictors<-c('d15N', 'Mass', 'd13C')

###Fitting Models
model_rf2<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf', importance=T)
model_nnet2<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet',
importance=T)
model_nb2<-train(trainSet[,predictors],trainSet[,outcomeName],method='naive_bayes',
importance=T)
model_gbm2<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',distribution =
"multinomial")
gbm_tuned2<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm',distribution =
"multinomial",trControl=fitControl, tuneLength = 20)

#b
##Extract metrics from models
results2<-data.frame(ExperimentName=c('RF','NNet', 'NB','GBM', 'GBM Tuned'),
                Accuracy=c(model_rf2$results[order(model_rf2$results$Accuracy,decreasing =
T),]['Accuracy'][1,1],model_nnet2$results[order(model_nnet2$results$Accuracy,decreasing =
T),]['Accuracy'][1,1],
```

```
                         model_nb2$results[order(model_nb2$results$Accuracy,decreasing = 
T),]['Accuracy'][1,1],model_gbm2$results[order(model_gbm2$results$Accuracy,decreasing = 
T),]['Accuracy'][1,1],
                         gbm_tuned2$results[order(gbm_tuned2$results$Accuracy,decreasing = 
T),]['Accuracy'][1,1]),
                Kappa=c(model_rf2$results[order(model_rf2$results$Accuracy,decreasing = 
T),]['Kappa'][1,1],model_nnet2$results[order(model_nnet2$results$Accuracy,decreasing = 
T),]['Kappa'][1,1],
                       model_nb2$results[order(model_nb2$results$Accuracy,decreasing = 
T),]['Kappa'][1,1],model_gbm2$results[order(model_gbm2$results$Accuracy,decreasing = 
T),]['Kappa'][1,1],
                       gbm_tuned2$results[order(gbm_tuned2$results$Accuracy,decreasing = 
T),]['Kappa'][1,1]))
##Sort on Accuracy
results2[order(results2$Accuracy,decreasing = T),]
```

Output:

```
  ExperimentName   Accuracy    Kappa
3           NB          0.7173333  0.5318312
2         NNet          0.6941420  0.4922918
4          GBM          0.6497462  0.4261104
5     GBM Tuned      0.6470033  0.4199928
1           RF          0.6432586  0.4175706
```

#c
##The best performing model is Naive Bayes. Now that we eliminated most features, ensemble
#methods like Random Forest and Gradient Boosting become less effective having less possible features
#to predict on. Neural Networks are probably too complex of a model for this data with only 110
#observations, so Naive Bayes performs the best on only a few feaures. Again we get accuracies better than
#random guessing but not close to an acceptable accuray.