# Assignment3

Dor Kolsky ID: 205687593 , Chemi Goldstein ID:206121717

28 12 2020

#Question 1 :

```
#The libraries required for the question are:
library(data.table)
library(ggplot2)
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

library(dplyr)

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following objects are masked from 'package:data.table':
##
##     between, first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

MSE <- function(x) x^2 %>% mean
```

First of all we will load 5000 random samples from the diamonds data set: A1:

```
diamonds <- as.data.table(diamonds)
set.seed(1)
```

```
d <- diamonds[sample(5000)]
folds <- 10
fold.assignment <- sample(1:folds, nrow(d), replace = TRUE)
errors <- NULL


for ( k in 1:folds){
  d_train <- d[fold.assignment!=k,] # train subset
  d_val <-  d[fold.assignment==k,] # validation subset
 randomTrain <- randomForest(price~., data = d_train, ntree=200, mtry
= ncol(d_train)/3) # train
  val_pred <- predict(randomTrain , newdata=d_val)
  folderrors <-  val_pred-d_val$price # save prediction errors in the
fold
  errors <- c(errors, folderrors)

}
```

A2: Aggregate the error on all the folds :

```
mseerror <- MSE(errors)
RMSE <- sqrt(mseerror)

cat("Our model RMSE is :" , RMSE)

## Our model RMSE is : 198.5886
```

B:

```
errors2 <- NULL


for ( k in 1:folds){
  d_train <- d[fold.assignment!=k,] # train subset
  d_val <-  d[fold.assignment==k,] # validation subset
 randomTrain <- randomForest(price~., data = d_train, ntree=300, mtry=
sqrt(ncol(d_train))) # train
  val_pred <- predict(randomTrain , newdata=d_val)
  folderrors <-  val_pred-d_val$price # save prediction errors in the
fold
  errors2 <- c(errors2, folderrors)

}
mseerror2 <- MSE(errors2)
RMSE2 <- sqrt(mseerror2)

cat("Our model RMSE is :" , RMSE2)

## Our model RMSE is : 198.5686
```

We increased the number of trees and changed the method of mtry and thus improved the RMSE .

C: The ntree parameter is the number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.

The mtry parameter is the number of variables randomly sampled as candidates at each split. We chose this parameter value depending on the kind of trial we are doing. For classification we will use the sqrt of the number of variables in the data-set and for regression we will use the number of variables divided by 3.

D: Leave-one-out cross validation:

Advantage: This cross validation method is much more effective. What it does in essence is divide the data into N subsets, and then cross validates K times (K=N), where for each cross validation, one of the subsets is left out in the training set, and this is the exact subset on which we make our prediction. This leads to a more precise model!

Disadvantage: One clear disadvantage is that it is much more time consuming and is more difficult for computer programs to compute this method.

#Question 2 :

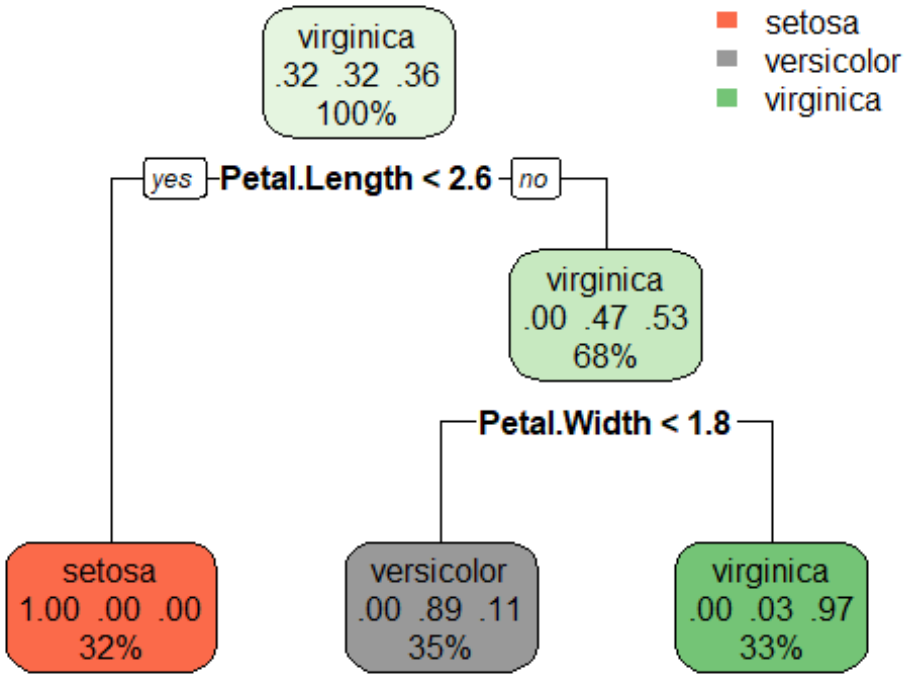
```
#The libraries required for the question are:
library(data.table)
library(rpart)
library(rpart.plot)
library(xgboost)

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

A:

```
iris <- as.data.table(iris)
set.seed(1)
ind<- sample(2, nrow(iris), replace= T , prob = c(0.7,0.3))
training<- iris[ind==1,]
test<- iris[ind==2,]

tree.1 <- rpart(Species~., data=training)
```

B:

```
rpart.plot::rpart.plot(tree.1)
```




This plot shows us a visualization of the decision tree which we have created. At the root, before we begin to 'grow' our tree, we can see that for our root, our model's species are distributed as follows: 32% - Setosa; 32% - Versicolor; and 36% - Virginica. Next, our tree divides the observations to those who's Petal_Length variable is less than 2.6. We now have two additional nodes (based on this condition), and now our model is a bit more detailed. Our new distribution based on this condition is as follows: Where the answer to our first condition was 'yes': 32% - Setosa [meaning that ALL of our Setosa species are separated using this condition]; Where the answer to our first condition was 'no: 68% of Versicolor and Virginica with the distribution of 47% and 53% respectfully.

Now, our tree will create an additional condition, to better divide our species distribution accurately. Our tree divides the observations to those who's Petal_Width variable is less than 1.8; only for the observations whose Petal_Length was greater than 2.6.

We now have two additional nodes (based on this second condition), and now our model is even more detailed. Our new distribution based on this condition is as follows: Still 32% of the observation are Setosa - unchanged Where the answer to our second condition was 'yes': 35% - Versicolor [where of these observations, 89% are accurate, and 11% are inaccurate - Virginica observation who also meet the two conditions]

Where the answer to our second condition was 'no': 33% - Virginica [where of these observations, 97% are accurate, and 3% are inaccurate - Versicolor observation who also meet the two conditions]

In summary, we have a decisive tree with two conditions. These conditions create the following distributions: Setosa – 32% (100% accuracy) Versicolor – 35% (3% above the true distribution) Virginica – 33% (3% below the true distribution)

C: Now we will predict our tree on the test set:

```
predictions.test <- predict(tree.1, newdata = test, type='class')
confusion.test <- table(prediction=predictions.test,
truth=test$Species)
confusion.test

##             truth
## prediction   setosa versicolor virginica
##   setosa         16          0         0
##   versicolor      0         16         1
##   virginica       0          0        11

accuracy <-
(confusion.test[1,1]+confusion.test[2,2]+confusion.test[3,3])/sum(confu
sion.test)
cat("Our tree accuracy prediction on the test set is ", accuracy)

## Our tree accuracy prediction on the test set is  0.9772727
```

D: We would consider using a decision tree for prediction in which the data of observations we possess has a significantly different variance depending on specific factors or variables. With that, using a decision tree to predict factors whose variables are random, will not be as effective – meaning we would not consider fitting to a decision tree.

For example, if we were to predict the height of a person using a large dataset, it would be wise to use a decision tree. One obvious condition, would be 'is the person a male or female?' This is intuitive since men are, by average, taller than women. And thus, using a decision tree with said condition would better divide our observations, and all-in-all give us a more precise prediction.

E: Now we will train xgb tree on the model :

```
# First of all we will make the model matrix of the training set and
cast it to matrix form
X_trn  <- data.matrix(training[, -5])
#Then we will extract the Species column from the training set and make
it 1 if it Setosa or 0 if its not
y_trn <- training[,5]
y_trn <- ifelse (y_trn=="setosa" , 1,0)

# We will do the same process to the test set
X_tst  <- data.matrix(test[, -5])
y_tst <- test[,5]
y_tst <- ifelse (y_tst=="setosa" , 1,0)
```

```r
# Now we will make an xgb model using default parameters (we did not
tune them)
params <- list( eta = 0.01, max_depth = 5, min_child_weight = 5,
subsample = 0.65, colsample_bytree = 1)

# Train final model
xgb.fit.final <- xgboost(
  params = params,
  data = X_trn,
  label = y_trn,
  nrounds = 50, # example for best nrounds found with cross validation
  objective = "binary:logistic",
  verbose = 0
)
```

```
## [15:50:32] WARNING: amalgamation/../src/learner.cc:1061: Starting in
XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
```

```r
# Now we will predict everything on the test set and calculate our
accuracy using a confusion matrix.
predxgb <- predict(xgb.fit.final,X_tst)
confusion.test.xgb <- table(prediction=predxgb, truth=y_tst)
confusion.test.xgb
```

```
##                    truth
## prediction           0  1
##   0.315809369087219 20  0
##   0.333517521619797  1  0
##   0.338067680597305  1  0
##   0.345219224691391  1  0
##   0.374915480613708  1  0
##   0.387742668390274  2  0
##   0.388432115316391  1  0
##   0.401163429021835  1  0
##   0.661567568778992  0 16
```

```r
accuracyxgb <-
(confusion.test.xgb[1,1]+confusion.test.xgb[9,2])/sum(confusion.test)
cat("Our tree accuracy prediction on the test set is ", accuracyxgb)
```

```
## Our tree accuracy prediction on the test set is  0.8181818
```

#Question 3 :

```r
#The libraries required for the question are:
library(data.table)
library(ggplot2)
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1

library(caret)

## Loading required package: lattice
```

A: L - is the empirical risk function Lambda - Controls the overall strength of the penalty in the model . Alpha - Controls the elastic net penalty , bridges the gap between lasso-ridge . ||B||2/2 - penalizing the l2 norm of the parameter , the sum of the squared coefficients . ||B||1 - penalizing the l1 norm of the parameter , the sum of the coefficient's in absolute terms.

```
diamonds <- as.data.table(diamonds)

d <- diamonds[1:10000,]
X_trn <- model.matrix(price~.-1, data=d[1:6000,]) %>% scale()
X_tst <- model.matrix(price~.-1,data=d[6001:10000,]) %>% scale()
y_trn <- scale(d$price[1:6000])
y_tst <- scale(d$price[6001:10000])
```

B1: We will run a glmnet model with all the parameters set to default (we assumed our Lambda is 0.01) on the train set and predict it on the test set:

```
glmnet_ridge <-
glmnet(x=X_trn,y=y_trn,family='gaussian',alpha=0,lambda=0.01)
ridge_prediction <- predict(glmnet_ridge,newx=X_tst)
```

B2: Now we will do the same thing with the lasso penalty :

```
glmnet_lasso <-
glmnet(x=X_trn,y=y_trn,family='gaussian',alpha=1,lambda=0.01)
lasso_prediction <- predict(glmnet_lasso,newx=X_tst)
```

C:

```
cbind(coef(glmnet_ridge, s = 0.01), coef(glmnet_lasso, s = 0.01))

## 25 x 2 sparse Matrix of class "dgCMatrix"
##                             1             1
## (Intercept)   3.266535e-15  2.280819e-15
## carat         9.449155e-02  .
## cutFair      -6.356443e-02 -3.291866e-02
## cutGood      -9.401225e-03  .
## cutVery Good  1.406395e-03  .
## cutPremium   -1.602480e-02  .
## cutIdeal      4.738134e-02  2.779295e-02
## color.L      -2.018777e-01 -1.787712e-01
## color.Q      -2.783098e-02 -1.042352e-02
## color.C      -4.374696e-03  .
## color^4       1.087838e-02  4.556123e-03
```

```
## color^5      -6.570310e-03  .
## color^6       1.052288e-02  8.251207e-04
## clarity.L     3.414428e-01  3.031721e-01
## clarity.Q    -9.788196e-02 -7.067872e-02
## clarity.C     7.028152e-02  4.338792e-02
## clarity^4    -7.372069e-02 -5.552009e-02
## clarity^5     2.582819e-02  1.190759e-02
## clarity^6    -6.229660e-03  .
## clarity^7     7.483374e-03  .
## depth         9.157647e-02  8.418813e-02
## table         2.322769e-02  .
## x             3.265659e-01  8.013730e-02
## y             5.807855e-01  9.129586e-01
## z             8.159288e-02  5.942416e-02
```

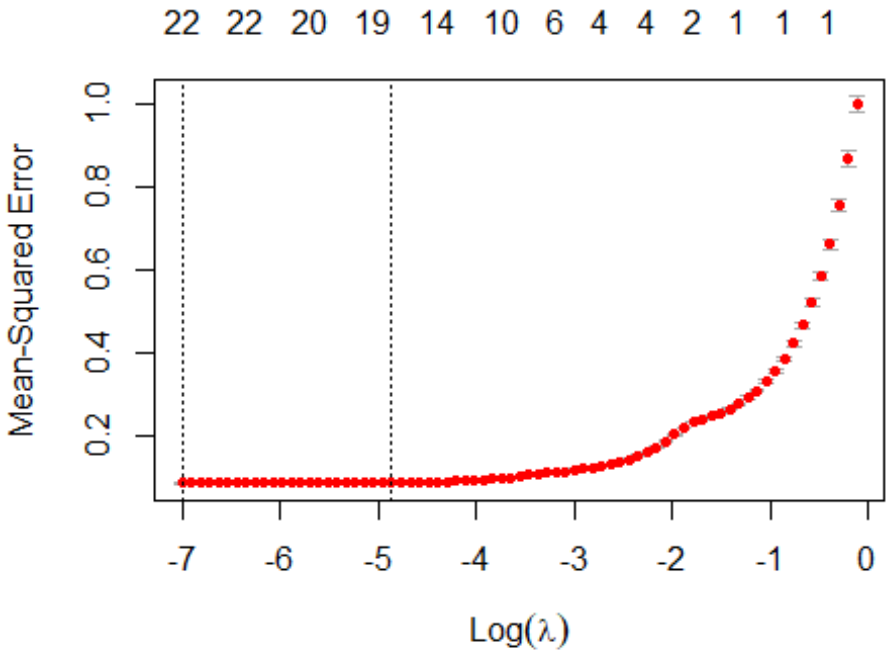
Both the ridge and lasso penalties can be used, and each has it's advantages and disadvantages.

As we can see from the results, some of our coefficients were set to 0 as a result of the lasso test. This is due to the fact that the lasso penalty constricts our coefficients region to a diamond which if the solution occurs at one of its corners, it can set a parameter to 0. The Ridge test constrains our coefficients to a disk region, where it cannot set any parameter to 0, but rather only make them smaller than their true value.

D: Using cross validation allows us to extract the Lambda which minimizes our model's MSE. This method is extremely useful. Cross validation uses a set number of Lambdas on our training_set, and afterwards tests out said Lambda on our test_set. It does this repeatedly and extracts the models MSE for each 'Lambda-run' In conclusion, we receive the Lambda which gives us the minimum MSE for our model, while taking into account overfitting (since we ran the Lambda on the test_set)

E: We will do a cross-validation to find the best Lambda for our test and run a glmm with this Lambda and with alpha=1 because we are doing a LASSO penalty :

```
cvfit <- cv.glmnet(X_trn,y_trn,family='gaussian',alpha=1)
plot(cvfit)
```




This plot shows us a visualization of our models MSE (with cross-validation), for a continuous value of Lambda The first vertical line, left, points to the Lambda which returns the models with the minimums MSE. The second vertical line, right, shows the Lambda which deviates 1 standard-deviation from our min-MSE-Lambda Both lines are informative and will result in very similar models.

As we can see, the is a clear trade-off between the size of the Lambda, and our models MSE. Our MSE is reduced as we increase our Lambda, up to a certain point in which it begins to rise. (our minimum MSE point)

The second vertical line, right, sums up our trade-off and is used frequently to avoid overfitting of our model. We 'give-up' some of our model's precision in exchange for being cautious to the fact that the lowest MSE might be too specific to our model, and thus would not be informative in explaining our model more generally. And as we mentioned, it gives us a model which is very similar to our MIN-MSE model.

F:

```
prediction_lasso_cv <- predict(cvfit,X_tst,s="lambda.min")
```

After extracting the Lambda which gave us our MIN-MSE model, we used said Lambda in our prediction of our test_set to have the most accurate predictions.

G: We will now run another model with the model matrix of all the interactions , we will use Default penalty Alpha:

```
X_interactions_train <- model.matrix(price~(.)^2-1,  data=d[1:6000,])
%>% scale()
X_interactions_test <- model.matrix(price~(.)^2-1,data=d[6001:10000,])
%>% scale()

cvfit_interactions <-
cv.glmnet(X_interactions_train,y_trn,family='gaussian')
```

Defining a new MSE function:

```
MSE <- function(x) x^2 %>% mean
```

Predicting on the Test set and finding the MSE:

```
prediction_lasso_cv_interactions<-
predict(cvfit_interactions,X_interactions_test,s="lambda.min")
MSE_CV <- MSE(prediction_lasso_cv_interactions)
cat("Our MSE is: ", MSE_CV)

## Our MSE is:  0.9339875
```

H:

```
y_trn_binar=y_trn>0
y_tst_binar=y_tst>0
glmnet_positive_Y <-
glmnet(x=X_trn,y=y_trn_binar,family='binomial',lambda=0.1)
prediction_positive_Y<- predict(glmnet_positive_Y,X_tst, type =
'class')
confusion.positive_Y <- table(predection = prediction_positive_Y,truth
= y_tst_binar)
```

To measure accuracy we will use the formula: Accuracy = (TP+TN)/(TP+TN+FP+FN)

```
TruePredctions <- confusion.positive_Y[1]+confusion.positive_Y[4]
TotalPredictions <- sum(confusion.positive_Y)
Accuracy <- TruePredctions/TotalPredictions
cat("Our accuracy level is ", Accuracy)

## Our accuracy level is  0.981
```

I:

```
X_trn_color <- model.matrix(color~.-1, data=d[1:6000,]) %>% scale()
X_tst_color <- model.matrix(color~.-1,data=d[6001:10000,]) %>% scale()
y_trn_color <- d$color[1:6000]
y_tst_color <- d$color[6001:10000]
```

We used the family of multi-nomial since our “Y” is factorial with many levels.

```
glmnet_color <-
glmnet(x=X_trn_color,y=y_trn_color,family='multinomial',alpha=0.5,lambd
```

```
a=0.01)
color_prediction <- predict(glmnet_color,newx=X_tst_color,type='class')

confusion.color <- table(predection = color_prediction ,truth =
y_tst_color)
TruePredctions_color <-
confusion.color[1]+confusion.color[9]+confusion.color[17]+confusion.col
or[25]+confusion.color[33]+confusion.color[41]+confusion.color[49]
TotalPredictions_color <- sum(confusion.color)
Accuracy_color <- TruePredctions_color/TotalPredictions_color
cat("Our accuracy level is ", Accuracy_color)

## Our accuracy level is  0.2145

caret::confusionMatrix(confusion.color)

## Confusion Matrix and Statistics
##
##           truth
## predection   D   E   F   G   H   I   J
##          D  93  60  34  14  13  15   1
##          E 249 327 306 294 310 148  25
##          F 131 196 292 281 354 134  58
##          G   8  32  31  30  74  79  43
##          H  11  13  23  38  32  20  34
##          I   0   1   4  10  24  49  66
##          J   0   0   0   0   0   8  35
##
## Overall Statistics
##
##                Accuracy : 0.2145
##                  95% CI : (0.2019, 0.2276)
##     No Information Rate : 0.2018
##     P-Value [Acc > NIR] : 0.02394
##
##                   Kappa : 0.0641
##
##  Mcnemar's Test P-Value : < 2e-16
##
## Statistics by Class:
##
##                      Class: D Class: E Class: F Class: G Class: H
Class: I
## Sensitivity          0.18902  0.51987   0.4232  0.04498  0.03965
0.10817
## Specificity          0.96095  0.60487   0.6514  0.91989  0.95647
0.97040
## Pos Pred Value       0.40435  0.19711   0.2019  0.10101  0.18713
0.31818
## Neg Pred Value       0.89416  0.87100   0.8442  0.82798  0.79760
0.89496
```

```
## Prevalence             0.12300  0.15725   0.1725  0.16675  0.20175
0.11325
## Detection Rate         0.02325  0.08175   0.0730  0.00750  0.00800
0.01225
## Detection Prevalence   0.05750  0.41475   0.3615  0.07425  0.04275
0.03850
## Balanced Accuracy      0.57499  0.56237   0.5373  0.48243  0.49806
0.53928
##                       Class: J
## Sensitivity            0.13359
## Specificity            0.99786
## Pos Pred Value         0.81395
## Neg Pred Value         0.94263
## Prevalence             0.06550
## Detection Rate         0.00875
## Detection Prevalence   0.01075
## Balanced Accuracy      0.56572
```

With our model, the color which was predicted with the highest accuracy was 'J'. We can see this by looking at the "Pos pred value' – which gives the percentage of positive predictions out of total predictions for said color. J's – Pos pred value: 81.3%

Base on our results, the colors which tend to be confused with each other using our model are: E-D, E-F, and F-H.

#Question 4 :

```
#The libraries required for the question are:
library(data.table)
library(ggplot2)
library(dplyr)
library(e1071)
library(caret)
```

A:

```
diamonds <- as.data.table(diamonds)
set.seed(1)
d <- diamonds[sample(4000)]
d$price <- cut(d$price,breaks=10 , labels = LETTERS[1:10])
ind<- sample(3, nrow(d), replace= T , prob = c(0.5,0.2,0.3))
d <- d %>% rename(price_cat=price)
training<- d[ind==1,]
validation<- d[ind==2,]
test <-d[ind==3,]
```

B:

```
X_trn <- model.matrix(price_cat~.-1, data=training)
X_val <- model.matrix(price_cat~.-1, data=validation)
X_tst <- model.matrix(price_cat~.-1, data=test)
```

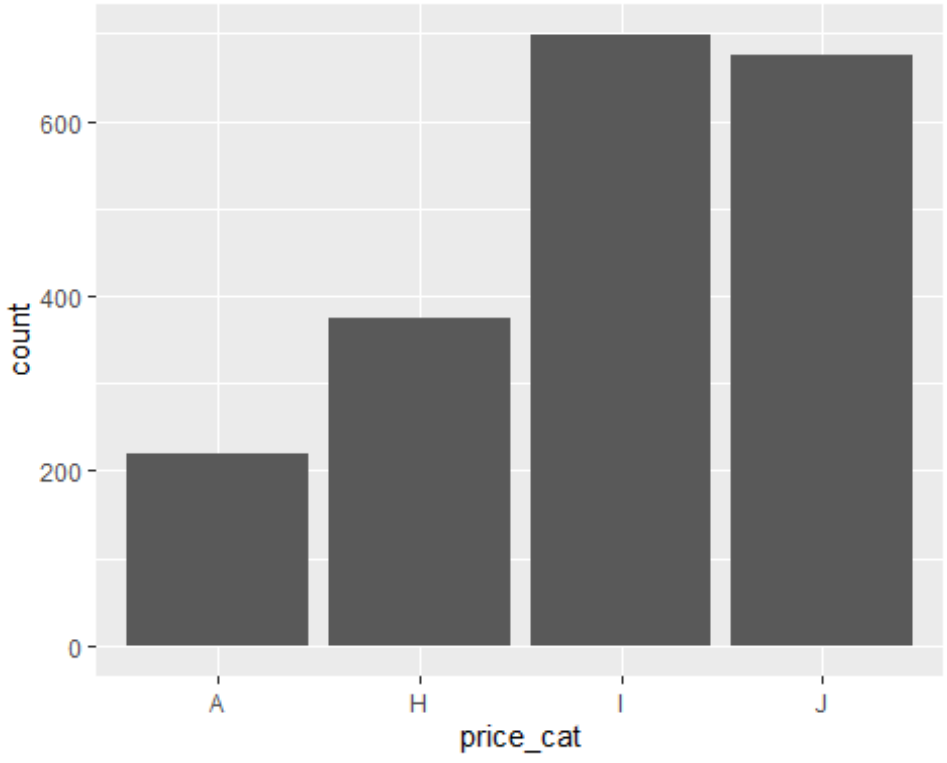
C:

```
means <- apply(X_trn , 2 , mean )
sds <- apply(X_trn , 2 , sd )
X_trn <- X_trn %>% sweep(MARGIN = 2, STATS = means, FUN = `-`) %>%
  sweep(MARGIN = 2, STATS = sds, FUN = `/`)

X_val <- X_val %>% sweep(MARGIN = 2, STATS = means, FUN = `-`) %>%
  sweep(MARGIN = 2, STATS = sds, FUN = `/`)

X_tst <- X_tst %>% sweep(MARGIN = 2, STATS = means, FUN = `-`) %>%
  sweep(MARGIN = 2, STATS = sds, FUN = `/`)
```

Scaling aka Standardization, is a technique in which all the features are centered around zero and have roughly unit variance. We do this in order to be able to use the data more efficiently we scale based on x_trn as to not contaminate the "test" data.

D:

```
ggplot(training, aes(x = price_cat)) +
    geom_bar()
```




As we can see, the price distribution of the training set is not balanced. Most of the diamonds

are priced above 2500; although there are a lot of diamonds which cost around 500 (outliers of sorts).

Additionally, the prediction type is not linear (as we can see from our distribution histogram), but rather radial.

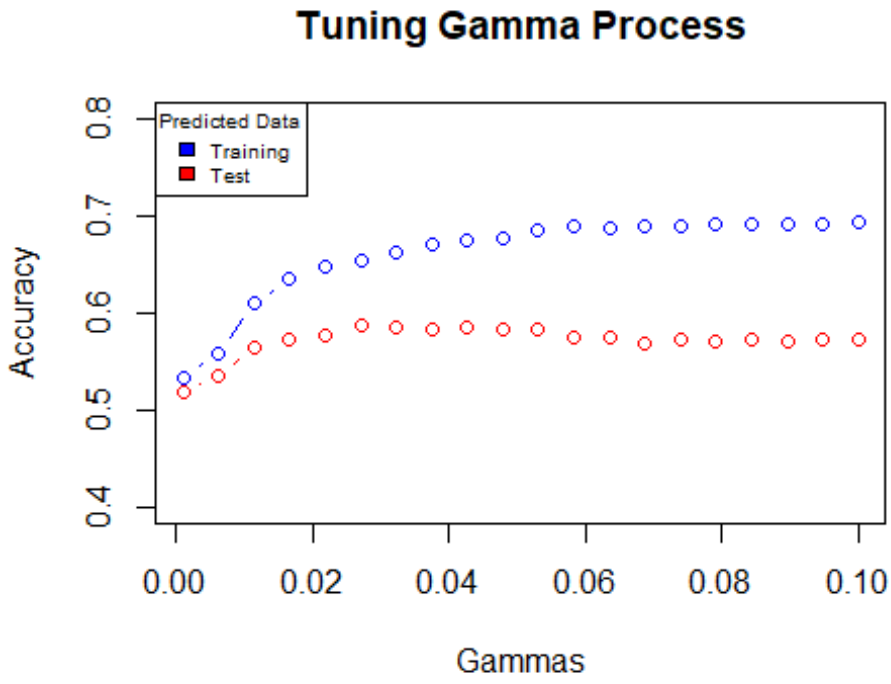
E: As requested - fit an SVM classification algorithm on the training set :

```
svm.1 <- svm(price_cat~X_trn, data = training)
```

Tuning the Gama on the validation set:

```
gamma_vector <- seq(0.001,0.1,length=20)
Accuracy_Vector_Training <- vector()
Accuracy_Vector_Validation <- vector()
Accuracy_Vector_Test <- vector()



for ( i in 1:20){
# SVM LEARNING ON THE TRAINING WITH DIFFRENT GAMA ALL THE TIME
svm.temp <- svm(x=X_trn , y=training$price_cat,type="C-
classification",kernel = "radial",gamma=gamma_vector[i])

#SVM PREDICTING ON THE Training AND MAKING CONFUSION MATRIX REGARDING
THE REAL Training PRICE_CAT
svm_temp_pred_trn <-predict(svm.temp,X_trn)
confusion_temp_trn <- confusionMatrix(data
=svm_temp_pred_trn,reference=training$price_cat)
Accuracy_Temp_trn <- confusion_temp_trn$overall[1]
Accuracy_Vector_Training[i] <-Accuracy_Temp_trn


#SVM PREDICTING ON THE Validation AND MAKING CONFUSION MATRIX REGARDING
THE REAL Training PRICE_CAT
svm_temp_pred_val <-predict(svm.temp,X_val)
confusion_temp_val <- confusionMatrix(data
=svm_temp_pred_val,reference=validation$price_cat)
Accuracy_Temp_val <- confusion_temp_val$overall[1]
Accuracy_Vector_Validation[i] <-Accuracy_Temp_val

#SVM PREDICTING ON THE Test AND MAKING CONFUSION MATRIX REGARDING THE
REAL Training PRICE_CAT
svm_temp_pred_test <-predict(svm.temp,X_tst)
confusion_temp_tst <- confusionMatrix(data
=svm_temp_pred_test,reference=test$price_cat)
Accuracy_Temp_test <- confusion_temp_tst$overall[1]
Accuracy_Vector_Test[i] <-Accuracy_Temp_test
```

```
}
{
plot(Accuracy_Vector_Test~gamma_vector,type="b",col="red",ylab="Accurac
y",xlab="Gammas",main="Tuning Gamma Process",ylim=c(0.4,0.8))
points(Accuracy_Vector_Training~gamma_vector , type="b", col="blue")
legend("topleft", legend=c("Training","Test"),fill=c("blue", "red"
),title="Predicted Data",cex=0.6)
}
```




```
#Find the best accuracy for the VALIDATION Data
Maximum_Accurate_Gama_Index <-which.max(Accuracy_Vector_Validation)
Maximum_Accurate_Gama <- gamma_vector[Maximum_Accurate_Gama_Index]
cat("The Best gama which give me the best accuracy is:
",Maximum_Accurate_Gama)

## The Best gama which give me the best accuracy is:  0.02705263
```

F: We found the most optimal Gamma (one which leads to the highest Accuracy-level), by tuning Gamma to the VALIDATION-SET, as not to have a case of overfitting. If we were to tune gamma with a loop using our original training set, we would obviously have a higher accuracy level, since our model was built using the SAME data-set. The purpose of using the validation set to tune gamma, is exactly to stray away from a case of overfitting, where our accuracy level would indeed be the highest, but would be a BIASED accuracy level.

G:

```r
svm.temp <- svm(x=X_trn , y=training$price_cat,type="C-
classification",kernel = "radial",gamma= 0.02705263)
#Training Confusion
svm_pred_training <-predict(svm.temp,X_trn)
confusion_training <- table(predection = svm_pred_training ,truth =
training$price_cat)
print ( " The confusion matrix for the training data is : ")
```

```
## [1] " The confusion matrix for the training data is : "
```

```r
confusion_training
```

```
##           truth
## predection   A   B   C   D   E   F   G   H   I   J
##          A 219   0   0   0   0   0   0   0   0   0
##          B   0   0   0   0   0   0   0   0   0   0
##          C   0   0   0   0   0   0   0   0   0   0
##          D   0   0   0   0   0   0   0   0   0   0
##          E   0   0   0   0   0   0   0   0   0   0
##          F   0   0   0   0   0   0   0   0   0   0
##          G   0   0   0   0   0   0   0   0   0   0
##          H   0   0   0   0   0   0   0  97  39   6
##          I   0   0   0   0   0   0   0 229 476 176
##          J   0   0   0   0   0   0   0  48 184 494
```

```r
diag_training <- sum(diag(confusion_training))
accuracy_training <- diag_training/sum(confusion_training)
cat("The accuracy in the training set is :", accuracy_training , "\n")
```

```
## The accuracy in the training set is : 0.6534553
```

```r
#Validation Confusion :
svm_pred_validation <-predict(svm.temp,X_val)
confusion_validation <- table(predection = svm_pred_validation ,truth =
validation$price_cat)
print ( " The confusion matrix for the validation data is : ")
```

```
## [1] " The confusion matrix for the validation data is : "
```

```r
confusion_validation
```

```
##           truth
## predection   A   B   C   D   E   F   G   H   I   J
##          A  92   0   0   0   0   0   0   0   0   0
##          B   0   0   0   0   0   0   0   0   0   0
##          C   0   0   0   0   0   0   0   0   0   0
##          D   0   0   0   0   0   0   0   0   0   0
##          E   0   0   0   0   0   0   0   0   0   0
##          F   0   0   0   0   0   0   0   0   0   0
##          G   0   0   0   0   0   0   0   0   0   0
##          H   1   0   0   0   0   0   0  27  16   6
```

```
##          I   0   0   0   0   0   0   0 109 197  84
##          J   0   0   0   0   0   0   0  31  92 166
```

```
diag_validation <- sum(diag(confusion_validation))
accuracy_validation <- diag_validation/sum(confusion_validation)
cat("The accuracy in the validation set is :",
accuracy_validation,"\n")
```

```
## The accuracy in the validation set is : 0.5870889
```

```
#Test Confusion
svm_pred_test <-predict(svm.temp,X_tst)
confusion_test <- table(predection = svm_pred_test ,truth =
test$price_cat)
print ( " The confusion matrix for the test data is : ")
```

```
## [1] " The confusion matrix for the test data is : "
```

```
confusion_test
```

```
##           truth
## predection   A   B   C   D   E   F   G   H   I   J
##          A 138   0   0   0   0   0   0   0   0   0
##          B   0   0   0   0   0   0   0   0   0   0
##          C   0   0   0   0   0   0   0   0   0   0
##          D   0   0   0   0   0   0   0   0   0   0
##          E   0   0   0   0   0   0   0   0   0   0
##          F   0   0   0   0   0   0   0   0   0   0
##          G   0   0   0   0   0   0   0   0   0   0
##          H   0   0   0   0   0   0   0  40  16   7
##          I   0   0   0   0   0   0   0 139 273 154
##          J   0   0   0   0   0   0   0  42 143 259
```

```
diag_test <- sum(diag(confusion_test))
accuracy_test <- diag_test/sum(confusion_test)
cat("The accuracy in the test set is :", accuracy_test)
```

```
## The accuracy in the test set is : 0.5862923
```

Our confusion matrices and accuracy levels differ from each of our sub-data sets, since for each set, our model is performed on a completely different data set. That being said, we can confidently assume that our accuracy level of the training-set will be significantly higher than our other two sets, since our model is built upon the same data set. Additionally, we can also assume that our accuracy levels for the Validation-set and the Test-sets ( the sets which were not used to build our model, and where separated randomly) will be rather close, for each Gamma.

H: It is important to find the optimal Gamma on the validation test, and only then use our optimal gamma on our test-set, because doing this accounts for overfitting (using a separate data-set than the one we built our model), and finally we can use

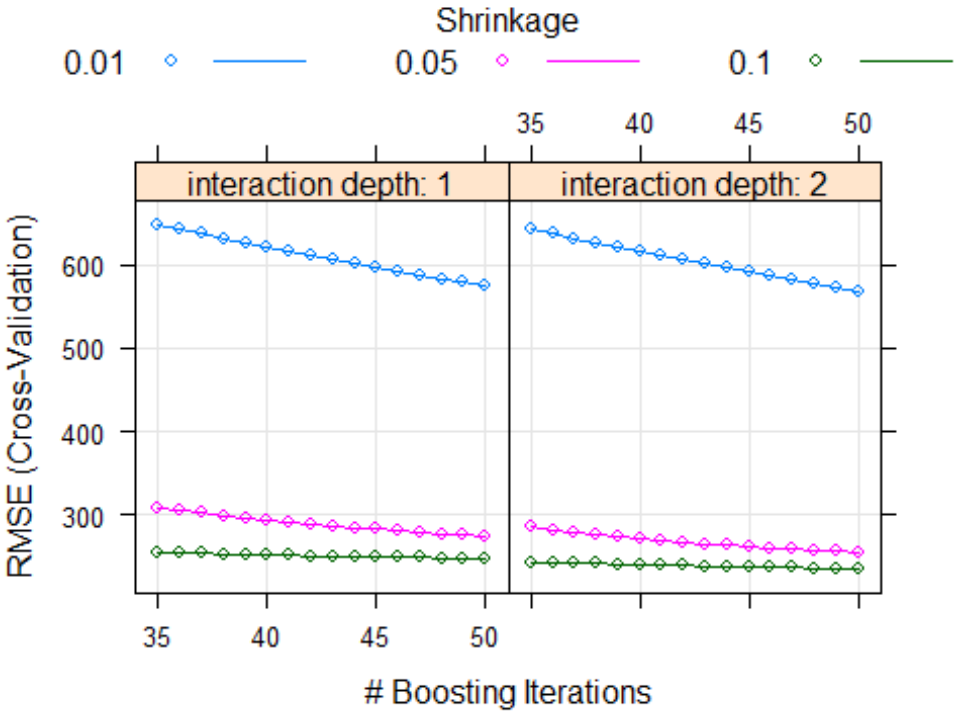
our original model, with our most accurate hyper-parameter, in order to predict or estimate performance on an uncontaminated and clean data-set, our test-set.

#Question 5:

```
#The libraries required for the question are:
library(data.table)
library(ggplot2)
library(caret)
library(gbm)

## Loaded gbm 2.1.8

trn <- diamonds[1:5000, ]
```

A:

```
gbmGrid <- expand.grid(interaction.depth =c(1,2) ,
 n.trees = c(35:50) , shrinkage =c(0.01,0.05,0.1),n.minobsinnode = 20 )



trcontrol <- trainControl(method = "cv",
                          number = 5,
                          search = "grid")

gbmFit <- train(price ~ .,
                data = trn,
                method = "gbm",
                trControl = trcontrol,
                verbose = FALSE,
                tuneGrid = gbmGrid)

plot(gbmFit)
```




B: We chose to do the tuning on the glmnet model we learned in the past :

```
glmnet_grid <- expand.grid(alpha = c(0,1) ,lambda = seq(0.0001, 1,
length = 50))
trcontrol <- trainControl(method = "cv",
                          number = 5,
                          search = "grid")

glmnet_model <- train(price ~ ., data = trn, method = "glmnet",
  tuneGrid = glmnet_grid, trControl = trcontrol
)

#In order to show our hyper-parameter tuning we will show the results
which contain our best hyper-parameters.
glmnet_model

## glmnet
##
## 5000 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3999, 4000, 4001, 4000, 4000
## Resampling results across tuning parameters:
##
##   alpha  lambda      RMSE      Rsquared   MAE
```

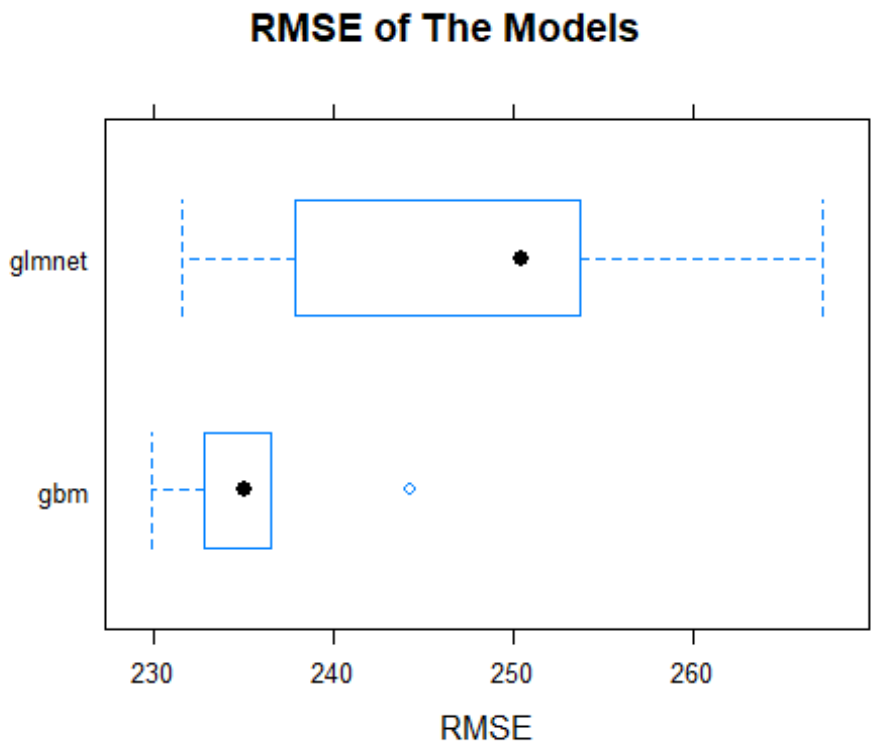
```
##   0      0.00010000  260.0796  0.9137964  201.5443
##   0      0.02050612  260.0796  0.9137964  201.5443
##   0      0.04091224  260.0796  0.9137964  201.5443
##   0      0.06131837  260.0796  0.9137964  201.5443
##   0      0.08172449  260.0796  0.9137964  201.5443
##   0      0.10213061  260.0796  0.9137964  201.5443
##   0      0.12253673  260.0796  0.9137964  201.5443
##   0      0.14294286  260.0796  0.9137964  201.5443
##   0      0.16334898  260.0796  0.9137964  201.5443
##   0      0.18375510  260.0796  0.9137964  201.5443
##   0      0.20416122  260.0796  0.9137964  201.5443
##   0      0.22456735  260.0796  0.9137964  201.5443
##   0      0.24497347  260.0796  0.9137964  201.5443
##   0      0.26537959  260.0796  0.9137964  201.5443
##   0      0.28578571  260.0796  0.9137964  201.5443
##   0      0.30619184  260.0796  0.9137964  201.5443
##   0      0.32659796  260.0796  0.9137964  201.5443
##   0      0.34700408  260.0796  0.9137964  201.5443
##   0      0.36741020  260.0796  0.9137964  201.5443
##   0      0.38781633  260.0796  0.9137964  201.5443
##   0      0.40822245  260.0796  0.9137964  201.5443
##   0      0.42862857  260.0796  0.9137964  201.5443
##   0      0.44903469  260.0796  0.9137964  201.5443
##   0      0.46944082  260.0796  0.9137964  201.5443
##   0      0.48984694  260.0796  0.9137964  201.5443
##   0      0.51025306  260.0796  0.9137964  201.5443
##   0      0.53065918  260.0796  0.9137964  201.5443
##   0      0.55106531  260.0796  0.9137964  201.5443
##   0      0.57147143  260.0796  0.9137964  201.5443
##   0      0.59187755  260.0796  0.9137964  201.5443
##   0      0.61228367  260.0796  0.9137964  201.5443
##   0      0.63268980  260.0796  0.9137964  201.5443
##   0      0.65309592  260.0796  0.9137964  201.5443
##   0      0.67350204  260.0796  0.9137964  201.5443
##   0      0.69390816  260.0796  0.9137964  201.5443
##   0      0.71431429  260.0796  0.9137964  201.5443
##   0      0.73472041  260.0796  0.9137964  201.5443
##   0      0.75512653  260.0796  0.9137964  201.5443
##   0      0.77553265  260.0796  0.9137964  201.5443
##   0      0.79593878  260.0796  0.9137964  201.5443
##   0      0.81634490  260.0796  0.9137964  201.5443
##   0      0.83675102  260.0796  0.9137964  201.5443
##   0      0.85715714  260.0796  0.9137964  201.5443
##   0      0.87756327  260.0796  0.9137964  201.5443
##   0      0.89796939  260.0796  0.9137964  201.5443
##   0      0.91837551  260.0796  0.9137964  201.5443
##   0      0.93878163  260.0796  0.9137964  201.5443
##   0      0.95918776  260.0796  0.9137964  201.5443
##   0      0.97959388  260.0796  0.9137964  201.5443
##   0      1.00000000  260.0796  0.9137964  201.5443
```

```
##   1      0.00010000  248.9054  0.9192507  193.8454
##   1      0.02050612  248.9054  0.9192507  193.8454
##   1      0.04091224  248.9054  0.9192507  193.8454
##   1      0.06131837  248.9054  0.9192507  193.8454
##   1      0.08172449  248.9054  0.9192507  193.8454
##   1      0.10213061  248.9054  0.9192507  193.8454
##   1      0.12253673  248.9054  0.9192507  193.8454
##   1      0.14294286  248.9054  0.9192507  193.8454
##   1      0.16334898  248.9054  0.9192507  193.8454
##   1      0.18375510  248.9054  0.9192507  193.8454
##   1      0.20416122  248.9054  0.9192507  193.8454
##   1      0.22456735  248.9054  0.9192507  193.8454
##   1      0.24497347  248.9054  0.9192507  193.8454
##   1      0.26537959  248.9043  0.9192514  193.8459
##   1      0.28578571  248.9001  0.9192540  193.8475
##   1      0.30619184  248.8941  0.9192580  193.8474
##   1      0.32659796  248.8876  0.9192625  193.8465
##   1      0.34700408  248.8779  0.9192690  193.8453
##   1      0.36741020  248.8522  0.9192860  193.8427
##   1      0.38781633  248.8258  0.9193034  193.8403
##   1      0.40822245  248.7989  0.9193211  193.8379
##   1      0.42862857  248.7734  0.9193379  193.8355
##   1      0.44903469  248.7486  0.9193543  193.8330
##   1      0.46944082  248.7232  0.9193711  193.8305
##   1      0.48984694  248.6979  0.9193877  193.8282
##   1      0.51025306  248.6731  0.9194042  193.8261
##   1      0.53065918  248.6485  0.9194204  193.8241
##   1      0.55106531  248.6244  0.9194363  193.8223
##   1      0.57147143  248.6008  0.9194519  193.8205
##   1      0.59187755  248.5776  0.9194672  193.8187
##   1      0.61228367  248.5543  0.9194826  193.8170
##   1      0.63268980  248.5310  0.9194980  193.8155
##   1      0.65309592  248.5080  0.9195132  193.8140
##   1      0.67350204  248.4854  0.9195281  193.8125
##   1      0.69390816  248.4632  0.9195429  193.8109
##   1      0.71431429  248.4412  0.9195574  193.8095
##   1      0.73472041  248.4192  0.9195720  193.8080
##   1      0.75512653  248.3972  0.9195865  193.8063
##   1      0.77553265  248.3754  0.9196009  193.8047
##   1      0.79593878  248.3539  0.9196153  193.8027
##   1      0.81634490  248.3333  0.9196292  193.8008
##   1      0.83675102  248.3131  0.9196428  193.7992
##   1      0.85715714  248.2933  0.9196562  193.7978
##   1      0.87756327  248.2733  0.9196697  193.7963
##   1      0.89796939  248.2523  0.9196838  193.7941
##   1      0.91837551  248.2315  0.9196979  193.7919
##   1      0.93878163  248.2110  0.9197117  193.7897
##   1      0.95918776  248.1912  0.9197251  193.7880
##   1      0.97959388  248.1739  0.9197370  193.7870
##   1      1.00000000  248.1561  0.9197491  193.7860
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 1.
```

C: We will now compare the previous models:

```
resamps <- resamples(list(glmnet=glmnet_model , gbm =gbmFit  ))
summary(resamps)

##
## Call:
## summary.resamples(object = resamps)
##
## Models: glmnet, gbm
## Number of resamples: 5
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## glmnet 187.7665 188.3650 193.6925 193.7860 197.7870 201.3188    0
## gbm    187.9219 190.3675 194.4966 192.9361 194.6959 197.1986    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## glmnet 231.5180 237.9103 250.3976 248.1561 253.7405 267.2142    0
## gbm    229.8624 232.8280 235.0123 235.6957 236.5404 244.2357    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
NA's
## glmnet 0.9099042 0.9181048 0.9189328 0.9197491 0.9195084 0.9322950
0
## gbm    0.9202360 0.9227983 0.9274251 0.9280657 0.9343661 0.9355029
0

bwplot(resamps, metric = "RMSE", main = "RMSE of The Models")
```




#Question 6 :

```
#Libraries Required for the question:
library(dplyr)
library(data.table)
library(devtools)

## Loading required package: usethis

#install_github("vqv/ggbiplot")
library(ggbiplot)

## Loading required package: plyr

## ------------------------------------------------------------------------
----------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr
first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------
----------

##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## Loading required package: scales

## Loading required package: grid
```

A:

```
data("spam", package = 'kernlab')
X <- spam[,1:50]
variance <- lapply(X,var )
Total_Variance <- sum(unlist(lapply(variance,sum)))
cat ("The sum of total variances (column-wise) is  " , Total_Variance)

## The sum of total variances (column-wise) is   34.40472
```

B:

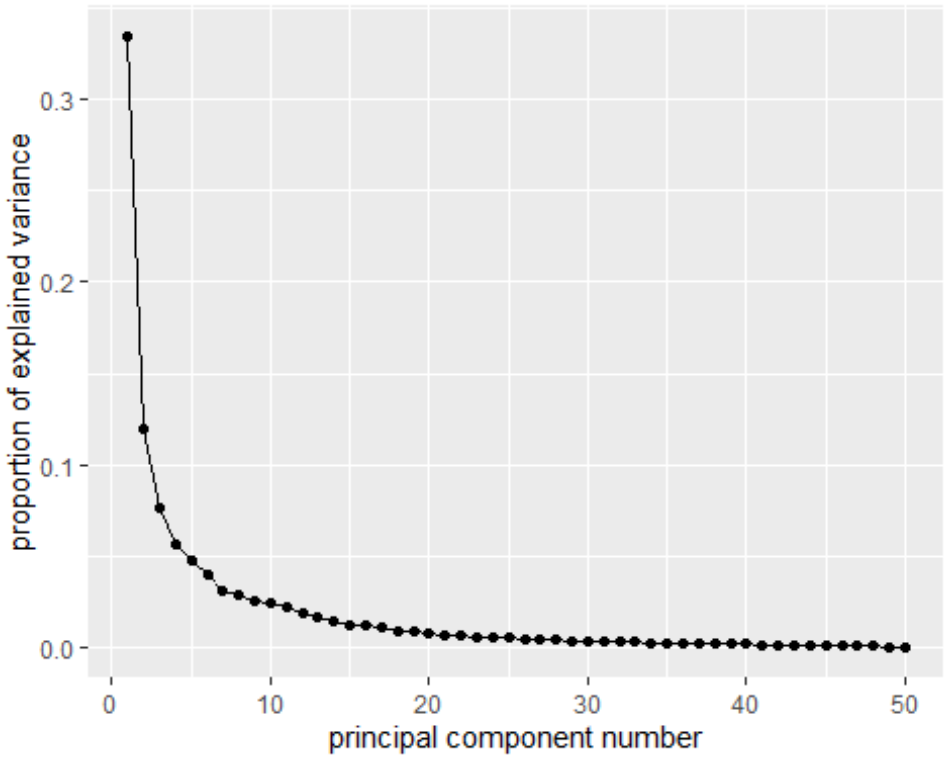
```
# First we will scale the data:
 X<- X
pca.1 <- prcomp(X , scale= FALSE)
SumVaration <- sum(pca.1$sdev^2)
cat("The total PC'S variance is ", SumVaration)

## The total PC'S variance is  34.40472
```

The variance we received is the same as in Q1. This is obviously the case, since the PCA model divides the total variance, into a certain amount of PCA's. Where the divided and newly built PCAs are ordered by the percent of the total variance in which they explain. PCA1-explains the most of our total variance, PCA2-explains the second most of our total variance, and so on… That being said, the total variance of all our PCAs explain 100% of our original variance, or in other words: The sum-variance of all PCAs, is equal to the the total variance.

C:

```
fifthpcsvar <- sum(pca.1$sdev[1:5])
cat ("The explained variance share of the first 5 PCS is ",fifthpcsvar
)

## The explained variance share of the first 5 PCS is  9.725481

precentage <- fifthpcsvar/SumVaration
cat ("     The 5 first pc's explain ", precentage*100 , "precent of the
total varaiance")

##      The 5 first pc's explain  28.26787 precent of the total
varaiance
```

```
# We created a screeplot:

ggbiplot::ggscreeplot(pca.1)
```




D:

```
AccumulativeVariance <- 0
NumofPcs <- 0
for ( i in 1:50){
  TempVar <- pca.1$sdev[i]
  AccumulativeVariance <- AccumulativeVariance +TempVar
  if (AccumulativeVariance/SumVaration >= 0.8){
    NumofPcs <- i
    break
  }
}
cat("The minimum number of PC's required to explain 80 precent of the
total variation is, " ,NumofPcs)

## The minimum number of PC's required to explain 80 precent of the
total variation is,  37
```

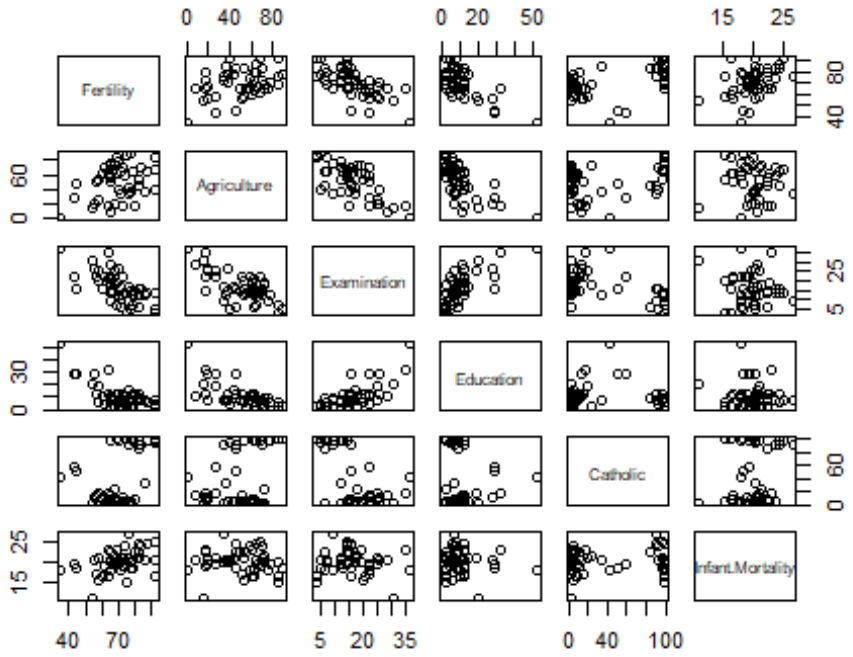
E:

```
X_SCALED<- X
pca.2 <- prcomp(X_SCALED , scale= TRUE)
ggbiplot::ggbiplot(pca.2 , groups = spam$type , var.axes = F)
```

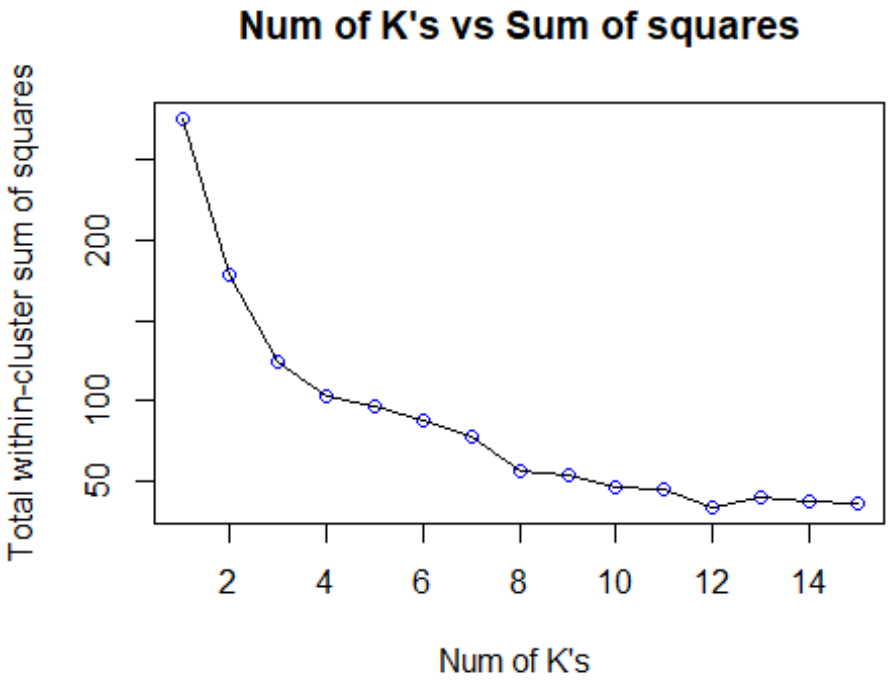

Clearly Spam/non-spam emails have different representations in the two-dimension graph plotting PC1~PC2. The non-spam emails (red) are effected by PC1 much more than that of spam emails. Spam emails are almost unaffected by PC1 as we can see that they are clustered at PC1=0, much unlike non-spam emails.

#Question 7 :

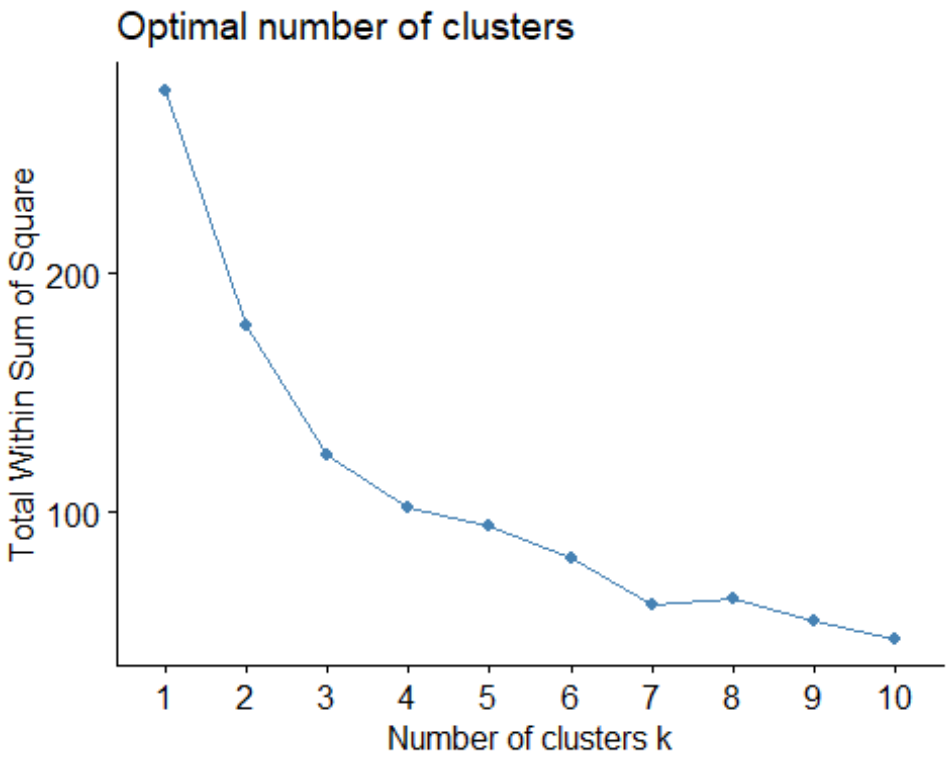
```
#Libraries Required for the question:
library(factoextra)

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

plot(swiss)
```




```
Total_Within_Clusters_SOS <- vector()
k_vector <- c(1:15)
swiss <- scale(swiss)

for (k in 1 :15 ){
kmeansswiss <- kmeans(swiss,centers=k)
Total_Within_Clusters_SOS_temp <- kmeansswiss$tot.withinss
Total_Within_Clusters_SOS[k] <-  Total_Within_Clusters_SOS_temp
}

{
plot(Total_Within_Clusters_SOS~k_vector , col = "blue" , xlab="Num of
K's " , ylab= "Total within-cluster sum of squares" , main = "Num of
K's vs Sum of squares" )
lines(Total_Within_Clusters_SOS~k_vector )
}
```




```
# We can see from the visualization that the elbow curve happens on K=7
and that is the optimal K for K-means. In order to concur our
hypothesis, we will use a function which returns a graph meant for K-
means which will help us again find the optimal K-mean cluster:

fviz_nbclust(swiss , kmeans , method="wss")
```




```
# As we can see here as well, the optimal K is 7 .
```

#Question 8 :

```
#Libraries Required for the question:
library(data.table)
library(shiny)
library(dplyr)

# # #First of all we will deal with the server side of the shiny app:

my_min <- 2



# server.R:
server <- function (input, output,session){
           swiss <- as.data.table(swiss)
           swiss_sub <- reactive({swiss[,input$variables, with =
                                 FALSE]%>%scale})
            pca.1 <- reactive({prcomp(swiss_sub() , scale.= TRUE )})
            distancesa <- reactive({dist(swiss_sub())})
            HCPC.swiss <-reactive({hclust(distancesa(),
method='single')})
            cut <- reactive({cutree(HCPC.swiss(), k=input$clusters)})
            output$main_plot<-renderPlot({
                plot(pca.1()$x[,1], pca.1()$x[,2], xlab="PC 1",
```

```
ylab="PC 2", type                ='n', lwd=2 , main = "Clustering
Observations over PC'S")
               text(pca.1()$x[,1], pca.1()$x[,2] , cex=0.7, lwd=2 ,
col=cut())})

 observe({
    if(length(input$variables) < my_min){
      updateCheckboxGroupInput(session, "variables", selected=
c("Fertility","Agriculture","Examination","Education" ,"Catholic"
,"Infant.Mortality"))
    }
  })



}




# UI
ui <- {
shinyUI(fluidPage(
  checkboxGroupInput(inputId = "variables",
  label = "Variables within the swiss data :",
  choices =
   c("Fertility","Agriculture","Examination","Education"
     ,"Catholic" ,"Infant.Mortality") , selected =
c("Fertility","Agriculture","Examination","Education"
     ,"Catholic" ,"Infant.Mortality")) ,

  numericInput  (inputId = "clusters",
  label = strong("Choose the number of clusters you want"),
  min = 1 , max = 9 , value = 5 ),

   plotOutput(outputId = "main_plot", height = "300px")

)
)
}
shinyApp(ui = ui,server = server)


# The link for the app is :
# https://dorchemi.shinyapps.io/DorChemi/
```