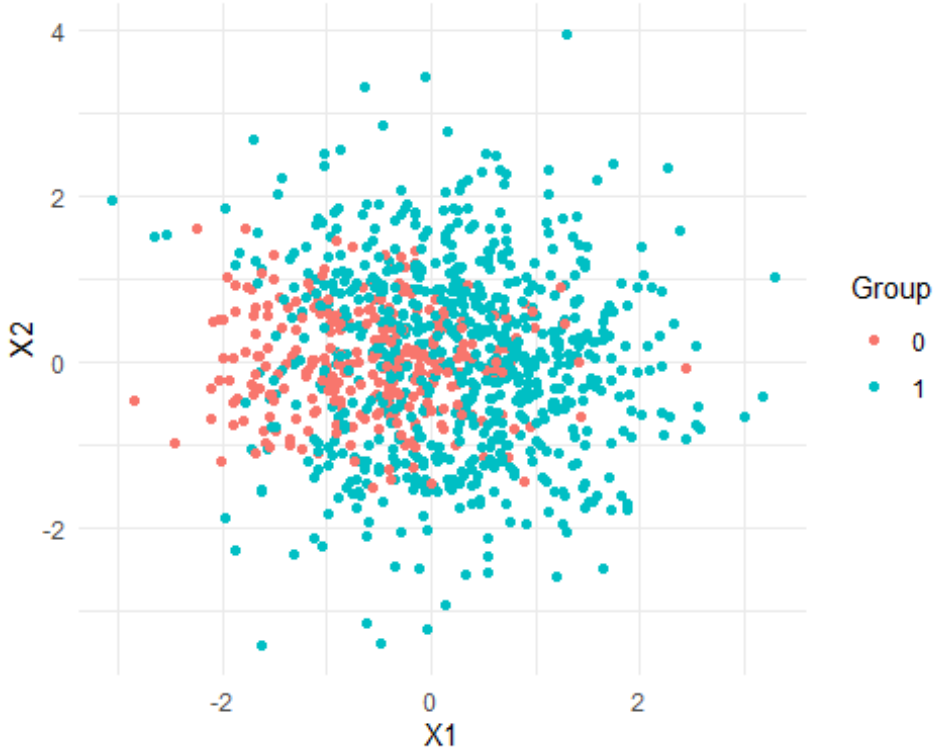
```
class_dataset = read.csv("Classification.csv")
class_dataset = class_dataset%>%mutate(Group = as.factor(Group))

summary(class_dataset)
```

```
##        X1                  X2           Group
##  Min.   :-3.055858   Min.   :-3.41429   0:293
##  1st Qu.:-0.679636   1st Qu.:-0.62566   1:707
##  Median :-0.038071   Median : 0.01472
##  Mean   : 0.002033   Mean   : 0.01631
##  3rd Qu.: 0.649037   3rd Qu.: 0.66946
##  Max.   : 3.285469   Max.   : 3.94396
```

```
ggplot(class_dataset) +
 aes(x = X1, y = X2, colour = Group) +
 geom_point(shape = "circle", size = 1.5) +
 scale_color_hue(direction = 1) +
 theme_minimal()
```



The dataset contains 1000 datapoints with no missing values. The data has two variables X1 and X2 these act as explanatory variables for two class Group 0 and group 1. On ploting the data points we observe that there is no proper separation between the two groups. Here we select 4 classifier models that perform well with this data KNN, QDA, SVM and Random forest. We are not using LDA as the data points are clustered together with low separation between the groups. This classification requires decision boundaries to be highly nonlinear. LDA uses linear decision boundaries and has high bias when the groups

are no separated clearly. Also to use LDA an assumption that data from each class should follow a normal distribution with equal variance has to be made. Therefore we are not using LAD to classify this problem.

Question 2

```
# CREATING TRAIN AND TEST DATA:
set.seed(12345)
partition = createDataPartition(class_dataset[,3], p =0.75,list = F)


# Train and test data:
exp_Train = class_dataset[partition,1:2]
resp_Train = class_dataset[partition,3]

exp_test = class_dataset[-partition, 1:2]
resp_test = class_dataset[-partition,3]

nrow(exp_Train)

## [1] 751

length(resp_Train)

## [1] 751

nrow(exp_test)

## [1] 249

length(resp_test)

## [1] 249
```

We divide the dataset into two parts train and test data. With 75% of data being train data and 25% being test data.

QUESTION 3

KNN Classifier:

KNN classifier is an flexible approach to estimate the bayes classifier. Here we find the neighbors of a point to check which class the point belongs to based on majority of the class of the neighbors. For this we have to mention the number of neighbours we are considering. This is given as a hyper parameter K. Then we find the probability of the point belonging to a class of the neighbouring points.

$$p(group0|X = x0) = 1/K(\sum_{i \in N_0} 1_{(y_1 = group0)}$$

From the equation we get the probability of a point belonging to each neighboring group. In binary case the probability of a point X0 belonging to group0 is the sum of number of neighboring group 0 points divided by the total number of neighbors. The same way we calculate the probability of getting group 1. Then we assign the point to a group with the highest probability. Here smaller the K value more flexible the model becomes the resulting boundary is overly flexible and will give low bias but high variance. Therefore we set an k = 3 as a initial hyper parameter.

```
# K - NEAREST NEIGHBOUR REGRESSION
fitted = knn(train = exp_Train,test = exp_test, cl = resp_Train, k=3)

#produce the confusion matrix
confusionMatrix(fitted,resp_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  37  26
##          1  36 150
##
##                Accuracy : 0.751
##                  95% CI : (0.6925, 0.8034)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.07026
##
##                   Kappa : 0.3741
##
##  Mcnemar's Test P-Value : 0.25304
##
##             Sensitivity : 0.5068
##             Specificity : 0.8523
##          Pos Pred Value : 0.5873
##          Neg Pred Value : 0.8065
##              Prevalence : 0.2932
##          Detection Rate : 0.1486
##    Detection Prevalence : 0.2530
##       Balanced Accuracy : 0.6796
##
##        'Positive' Class : 0
##
```

From the classification model, we get an accuracy of 75.1%. This shows the total number of correct classifications. This measure alone is not sufficient when selecting an ideal classifier. This is because it might predict bad for one class and good for another class. Therefore we check the sensitivity and specificity. We have a sensitivity of 50.6% this

shows the percentage of group 0 predicted correctly. The specificity is 85.23% this shows the percentage of group 1 predicted correctly. From this, it is clear that the model predicts group 1 better than group 0. Overall the model predicts well for k = 3. To check for other K values we perform K-fold Cross-validation.

```
#K- fold cross validation
opts <- trainControl(method='repeatedcv', number=10, repeats=5)

mdl <- train(x=exp_Train, y=resp_Train, # training data
             method='knn', # machine learning model
             trControl=opts, # training options
             tuneGrid=data.frame(k=seq(2, 15)))

print(mdl)
```

```
## k-Nearest Neighbors
##
## 751 samples
##   2 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 675, 676, 676, 676, 676, 676, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    2  0.6836211  0.2281735
##    3  0.7211614  0.3060428
##    4  0.7318070  0.3252220
##    5  0.7571404  0.3808296
##    6  0.7408912  0.3424460
##    7  0.7563333  0.3795568
##    8  0.7579404  0.3895478
##    9  0.7590070  0.3910466
##   10  0.7499474  0.3680402
##   11  0.7531474  0.3754219
##   12  0.7518140  0.3697808
##   13  0.7547474  0.3778937
##   14  0.7544842  0.3778810
##   15  0.7571474  0.3825703
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
# Test model on testing data
yTestPred <- predict(mdl, newdata=exp_test)
confusionMatrix(yTestPred, resp_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  38  20
##          1  35 156
##
##                Accuracy : 0.7791
##                  95% CI : (0.7224, 0.8291)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.006356
##
##                   Kappa : 0.4329
##
##  Mcnemar's Test P-Value : 0.059058
##
##             Sensitivity : 0.5205
##             Specificity : 0.8864
##          Pos Pred Value : 0.6552
##          Neg Pred Value : 0.8168
##              Prevalence : 0.2932
##          Detection Rate : 0.1526
##    Detection Prevalence : 0.2329
##       Balanced Accuracy : 0.7035
##
##        'Positive' Class : 0
##
```

Here we perform 10-fold cross-validation repeated 5 times. To reduce the effect of how the dataset is split on the outcome. For this, the function divides the dataset into 10 bits and trains on 9 bits and tests on 1 bit. For each split we consider k values from 2 to 15 and get the error rate then we calculate the error estimate for each k value.

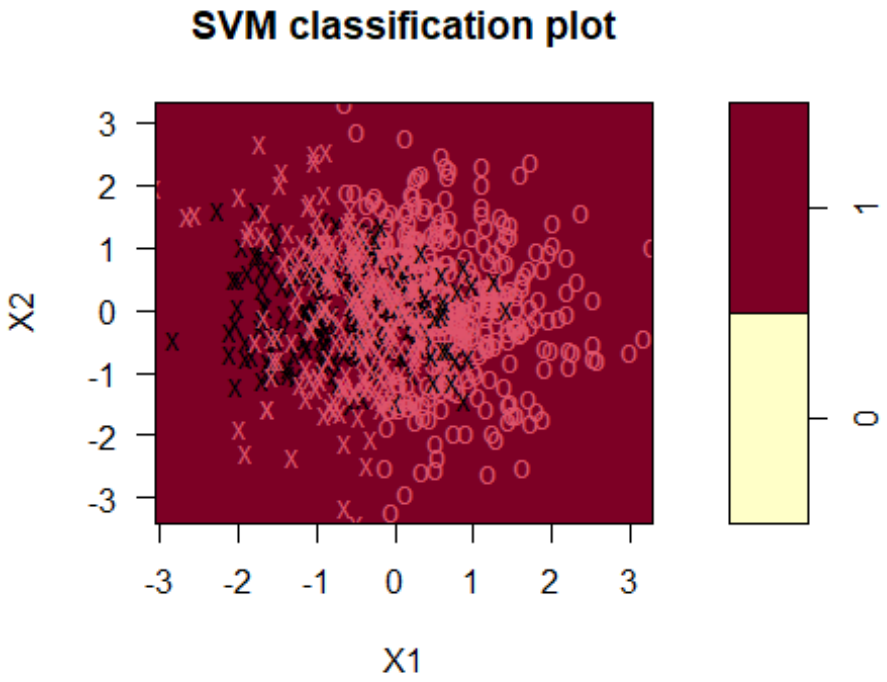
By performing cross-validation we got an ideal k value of 12 with increased accuracy, sensitivity and specificity. sensitivity and specificity increased to 54.79 and 86.93% respectively and the accuracy increased to 77.51%.

SVM:

Support vector machine classification is used for binary classification problems when two predictors give two classes. Here a hyperplane is used to separate a p-dimension plane containing the groups into two half. In 2D classification, the hyperplane is a line. We chose the hyperplane with the highest minimum distance between a point and the hyperplane called margin. After finding the optimal hyperplane with the maximum margin, we need to solve the problem of overfitting. Overfitting happens because the hyperplane acts as a line separating classes. For this, we provide each point with a slack variable. This allows observations to be on the wrong side of the margin. Slack becomes zero if an observation is on the correct side of the margin. if the observation is inside the margin but on the correct

side of the hyperplane slack is between 0 and 1. if a point is on the wrong side of the hyperplane slack is greater than 1. These slack points create a tuning parameter called constant by summing up all slack points and dividing it by the margin distance. This parameter is used to control the Bias- variance trade-off. Greater constant allows for more miss classification.

```
# SVM LINEAR:
df = data.frame(X2 = exp_Train$X2,X1 = exp_Train$X1,resp_Train)
mdl_linear_dif = svm(resp_Train~.,data = df, kernel= "linear")
plot(mdl_linear_dif,df)
```



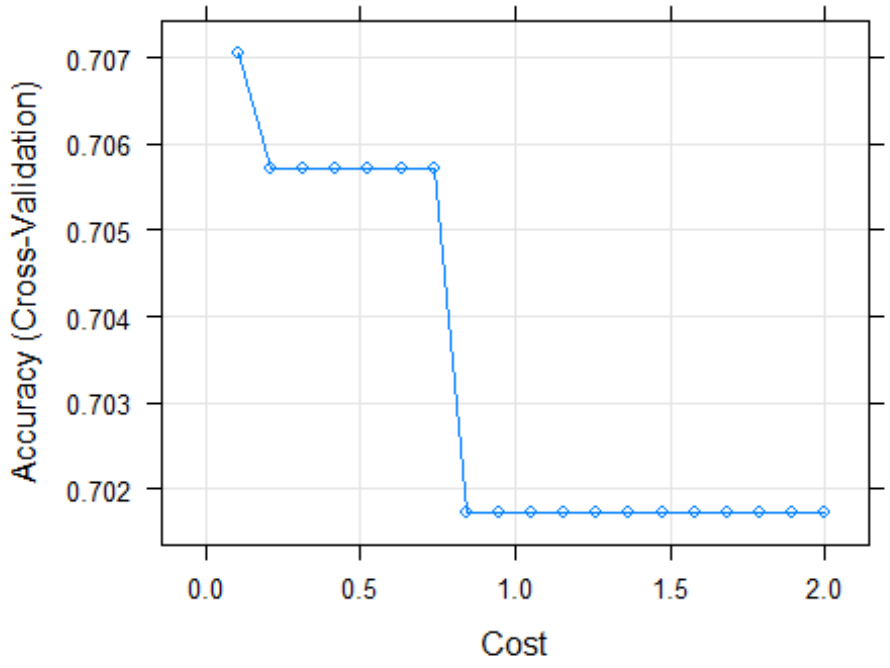

```
# Testing model
yTestPred <- predict(mdl_linear_dif, newdata = exp_test)
confusionMatrix(yTestPred, resp_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1  73 176
##
##                Accuracy : 0.7068
##                  95% CI : (0.646, 0.7626)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.5315
##
```

```
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7068
##              Prevalence : 0.2932
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

Here we get an accuracy of 70.68%. To find the optimal value of constant we perform cross-validation for constants between 0 to 2

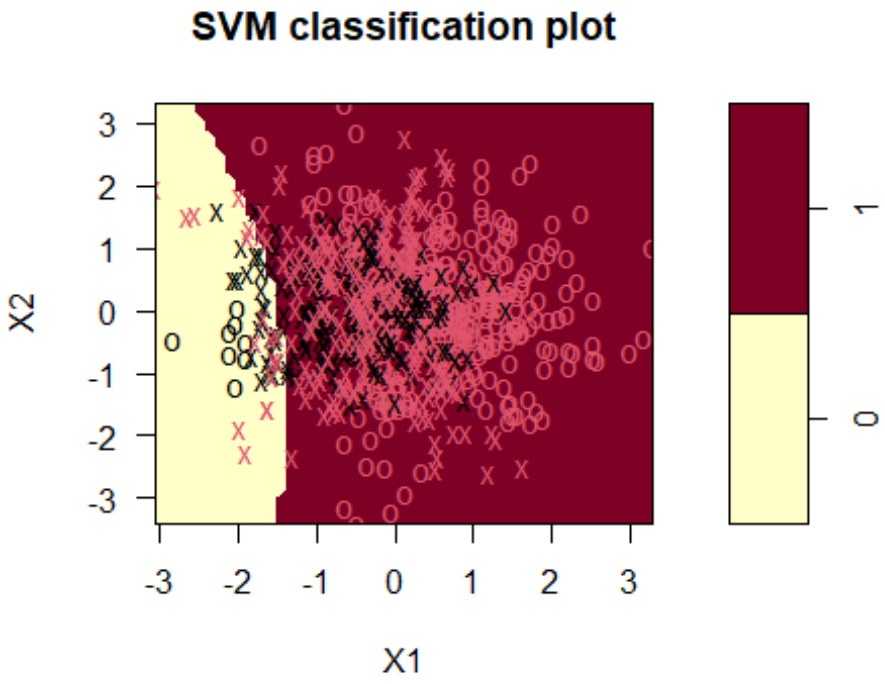
```
set.seed(123)
mdl <- train(x=exp_Train,y=resp_Train, method = "svmLinear",
             trControl = trainControl("cv", number = 5),
             tuneGrid = expand.grid(C = seq(0,2,length=20)))

plot(mdl)
```



```
# Best parameter C that maximises model accuracy:
mdl$bestTune

##           C
## 2 0.1052632

#summary of Test model on testing data
yTestPred <- predict(mdl, newdata=exp_test)
confusionMatrix(yTestPred, resp_test)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1  73 176
##
##                Accuracy : 0.7068
##                  95% CI : (0.646, 0.7626)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.5315
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7068
##              Prevalence : 0.2932
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

The optimal constant is 0.105. Here we see that the accuracy for the model is 70.68% but the sensitivity is 0. This means that the percentage of group 0 identified correctly is 0%. This is a problem with linearity in 'SVM' when the given data are not clearly separated. To relax the linearity we use the kernel function. There are 4 kernel functions linear, polynomial, radial and sigmoidal. To relax linearity here we are going to use polynomial and radial kernel.

```
# SVM POLY:

df = data.frame(X2 = exp_Train$X2,X1 = exp_Train$X1,resp_Train)
mdl_poly_dif = svm(resp_Train~.,data = df, kernel= "polynomial")
plot(mdl_poly_dif,df)
```



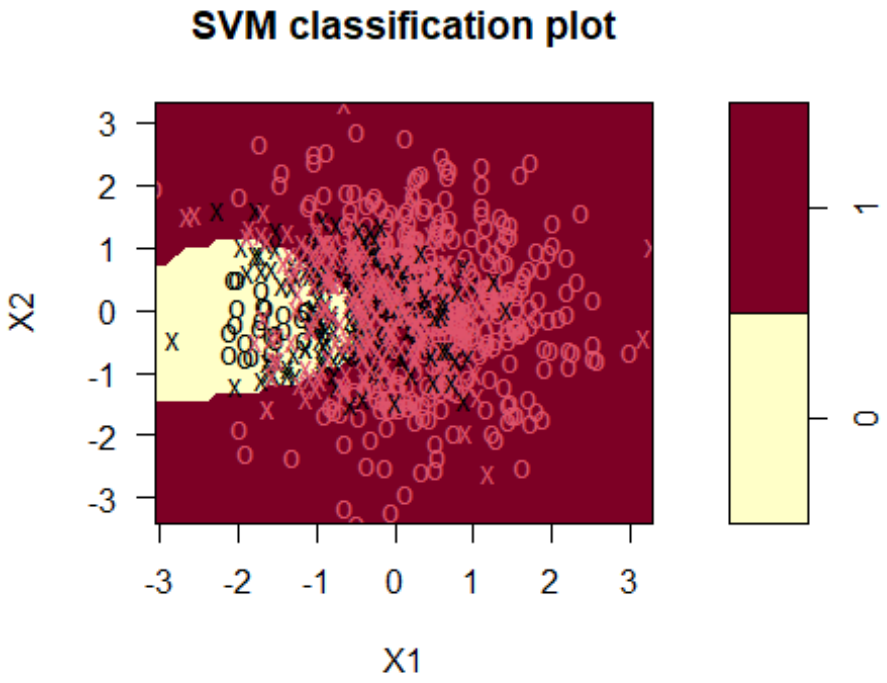

```
# summary stat on how model perform on Test
yTestPred <- predict(mdl_poly_dif, newdata = exp_test)
confusionMatrix(yTestPred, resp_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  11   2
##          1  62 174
##
##                Accuracy : 0.743
##                  95% CI : (0.684, 0.7961)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.1175
##
##                   Kappa : 0.1834
##
##  Mcnemar's Test P-Value : 1.643e-13
##
##             Sensitivity : 0.15068
##             Specificity : 0.98864
##          Pos Pred Value : 0.84615
##          Neg Pred Value : 0.73729
##              Prevalence : 0.29317
##          Detection Rate : 0.04418
##    Detection Prevalence : 0.05221
```

```
##       Balanced Accuracy : 0.56966
##
##        'Positive' Class : 0
##
```

From the confusion matrix, we observe a better classification result. Here Accuracy has increased to 74.3% and sensitivity has also increased to 15%.

```
# SVM POLY:
df = data.frame(X2 = exp_Train$X2,X1 = exp_Train$X1,resp_Train)
mdl_radial_dif = svm(resp_Train~.,data = df, kernel= "radial")
plot(mdl_radial_dif,df)
```




```
# summary stat on how model perform on Test
yTestPred_r <- predict(mdl_radial_dif, newdata = exp_test)
confusionMatrix(yTestPred_r, resp_test)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  31  16
##          1  42 160
##
##                Accuracy : 0.7671
##                  95% CI : (0.7095, 0.8181)
##     No Information Rate : 0.7068
```

```
##     P-Value [Acc > NIR] : 0.020069
##
##                   Kappa : 0.3726
##
##  Mcnemar's Test P-Value : 0.001028
##
##             Sensitivity : 0.4247
##             Specificity : 0.9091
##          Pos Pred Value : 0.6596
##          Neg Pred Value : 0.7921
##              Prevalence : 0.2932
##          Detection Rate : 0.1245
##    Detection Prevalence : 0.1888
##       Balanced Accuracy : 0.6669
##
##        'Positive' Class : 0
##
```

Here a better classification is observed with accuracy being 76.7% and sensitivity increased to 42% with a very low drop in the specificity of 90.91%. This seems to be a good classification model.

RANDOM FORESTS:

The random forest classification uses principles of decision tree but with less variance. Random forest produce uncorrelated decision trees from which we build separate models and we average the predictions we get from the model which gives us reduced variance. This is done by allowing each split to use only a small selection of variables which changes the inital split.

```
set.seed(123)

mdl_RF <- train(x=exp_Train,
             y=resp_Train,
             method='rf',
             ntree=200,
             tuneGrid=data.frame(mtry=2))

print(mdl)
```

```
## Support Vector Machines with Linear Kernel
##
## 751 samples
##   2 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 600, 601, 601, 601, 601
## Resampling results across tuning parameters:
```

```
##
##   C          Accuracy   Kappa
##   0.0000000        NaN         NaN
##   0.1052632  0.7070552  0.000000000
##   0.2105263  0.7057219  0.008075064
##   0.3157895  0.7057219  0.008075064
##   0.4210526  0.7057219  0.008075064
##   0.5263158  0.7057219  0.008075064
##   0.6315789  0.7057219  0.008075064
##   0.7368421  0.7057219  0.011399832
##   0.8421053  0.7017219  0.007177290
##   0.9473684  0.7017219  0.007177290
##   1.0526316  0.7017219  0.007177290
##   1.1578947  0.7017219  0.007177290
##   1.2631579  0.7017219  0.007177290
##   1.3684211  0.7017219  0.007177290
##   1.4736842  0.7017219  0.007177290
##   1.5789474  0.7017219  0.007177290
##   1.6842105  0.7017219  0.007177290
##   1.7894737  0.7017219  0.007177290
##   1.8947368  0.7017219  0.007177290
##   2.0000000  0.7017219  0.007177290
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.1052632.

# summary stat on how model perform on Test
test_Pred <- predict(mdl, newdata=exp_test)
confusionMatrix(test_Pred, resp_test)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1  73 176
##
##                Accuracy : 0.7068
##                  95% CI : (0.646, 0.7626)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.5315
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7068
```

```
##              Prevalence : 0.2932
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
varImp(mdl_RF$finalModel)
```

```
##     Overall
## X1 165.5558
## X2 144.4872
```

Here inside the train function we provide the model as random forest(rf).The variable ntree tell us how many bootstap dataset we want to create. Tune grid is used to specify the number of variables to consider at each split. From the statistical summary on how well the model fits the data we see that the accuracy is 77.51% and prediction made for group 0 is 54% (sensitivity) and for group 1 is 86.9% (specificity). Here we can see the importance of each variable using VarImp function. Here X1 is more important than X2 as it has higher overall value.

QDA:

QDA comes from LDA with some variations in the assumption. QDA has the underlying assumption that the classes come from a normal distribution. But unlike LDA it does not assume that the variance of each class has to be the same. QDA can use more than 2 response variables to classify a point. QDA uses an indirect method to calculate the probability of a point belonging to a class. This is done by first setting a prior probability of a point belonging to a class and then using Multivariate normal distribution (in case of more than 1 predictor) with a class-specific mean and class-specific covariate matrix to find the probability of an explanatory variable belonging to each class. We then use the Bayes theorem to flip the condition to find the probability of a point belonging to a class given an explanatory variable. The discriminate function of QDA is quadratic and produces quadratic decision boundaries.

```
# Data set for QDA:
data_train_qda = data.frame( X2 = exp_Train$X2,X1 = exp_Train$X1, resp_Train)

QDA_model = qda(resp_Train ~.,
                data = data_train_qda)

# summary stat on how model perform on Test
prediction_qda = predict(QDA_model, exp_test)
confusionMatrix(prediction_qda$class,resp_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0  36  22
##          1  37 154
##
##                Accuracy : 0.7631
##                  95% CI : (0.7053, 0.8145)
##     No Information Rate : 0.7068
##     P-Value [Acc > NIR] : 0.02826
##
##                   Kappa : 0.3917
##
##  Mcnemar's Test P-Value : 0.06836
##
##             Sensitivity : 0.4932
##             Specificity : 0.8750
##          Pos Pred Value : 0.6207
##          Neg Pred Value : 0.8063
##              Prevalence : 0.2932
##          Detection Rate : 0.1446
##    Detection Prevalence : 0.2329
##       Balanced Accuracy : 0.6841
##
##        'Positive' Class : 0
##
```

From the confusion matrix we get the accuracy as 76.31% and the percentage of group zero correctly predicted is 49.3% and the percentage of group one predicted correctly is 87.5%.

citation: https://www.datatechnotes.com/2019/04/qda-classification-with-r.html