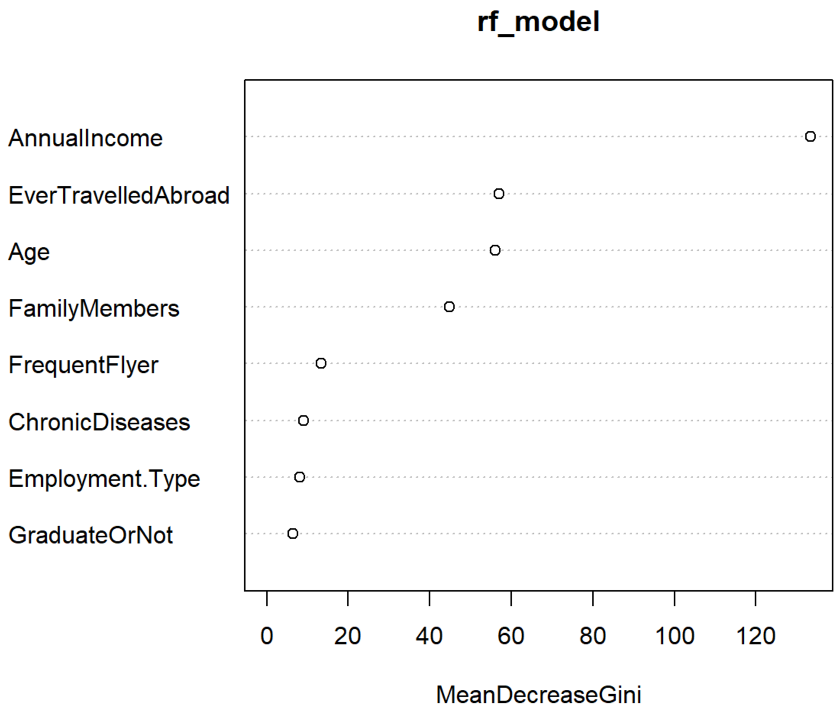
STAT562 PROJECT CODING

Random forest method

With out CV and tuning

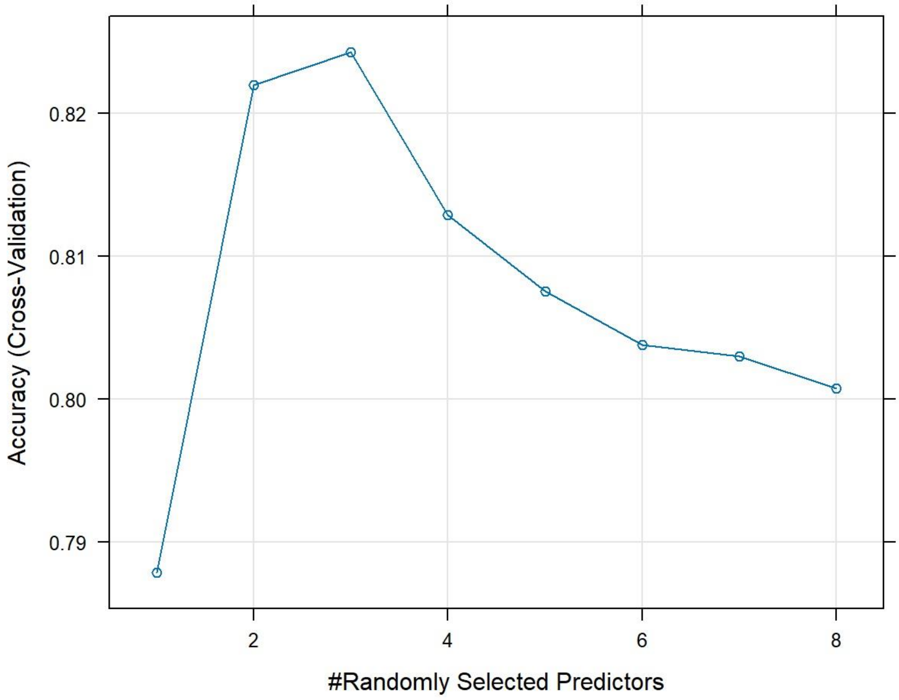
```
> library(readr)
> TravelInsuranceData <- read_csv("C:/Users/gutta/OneDrive/Desktop/SIUE/Fall
2023/STAT562 Machine Learning & Classification
methods/Project/TravelInsuranceData.csv")
> View(TravelInsuranceData)
> splitindex <- sample(1:nrow(TravelInsuranceData),
0.7*nrow(TravelInsuranceData))
> train_data <- TravelInsuranceData[splitindex, ]
> test_data <- TravelInsuranceData[-splitindex, ]
> set.seed(12)
> splitindex <- sample(1:nrow(TravelInsuranceData),
0.7*nrow(TravelInsuranceData))
> train_data <- TravelInsuranceData[splitindex, ]
> test_data <- TravelInsuranceData[-splitindex, ]
> train_data$Employment.Type <- as.factor(train_data$Employment.Type)
> test_data$Employment.Type <- as.factor(test_data$Employment.Type)
> train_data$GraduateOrNot <- as.factor(train_data$GraduateOrNot)
> test_data$GraduateOrNot <- as.factor(test_data$GraduateOrNot)
> train_data$ChronicDiseases <- as.factor(train_data$ChronicDiseases)
> test_data$ChronicDiseases <- as.factor(test_data$ChronicDiseases)
> train_data$FrequentFlyer <- as.factor(train_data$FrequentFlyer)
> test_data$FrequentFlyer <- as.factor(test_data$FrequentFlyer)
> train_data$EverTravelledAbroad <- as.factor(train_data$EverTravelledAbroad)
> test_data$EverTravelledAbroad <- as.factor(test_data$EverTravelledAbroad)
> train_data$TravelInsurance <- as.factor(train_data$TravelInsurance)
> X <- train_data[c("Age","Employment.Type", "GraduateOrNot", "AnnualIncome",
"FamilyMembers", "ChronicDiseases", "FrequentFlyer", "EverTravelledAbroad")]
> y <- train_data$TravelInsurance

> train_data <- train_data %>% select(-...1)
> test_data <- test_data %>% select(-...1)
> rf_model <- randomForest(TravelInsurance~., data = train_data, ntree = 1000)
> varImpPlot(rf_model)
```




```
> importance(rf_model)
                    MeanDecreaseGini
Age                        56.027789
Employment.Type             8.083341
GraduateOrNot               6.499292
AnnualIncome              133.489301
FamilyMembers              44.905043
ChronicDiseases             9.107968
FrequentFlyer              13.337995
EverTravelledAbroad        57.057060

> predictions_test <- predict(rf_model, newdata = test_data)
> conf_matrix_test <- table(predictions_test, test_data$TravelInsurance)
>
> accuracy_test <- sum(diag(conf_matrix_test)) / sum(conf_matrix_test)
> cat("Accuracy on Test Data:", accuracy_test, "\n")
Accuracy on Test Data: 0.8271605
> precision_test <- conf_matrix_test[2, 2] / sum(conf_matrix_test[, 2])
> recall_test <- conf_matrix_test[2, 2] / sum(conf_matrix_test[2, ])
> f1_score_test <- 2 * (precision_test * recall_test) / (precision_test +
recall_test)
>
> cat("Precision on Test Data:", precision_test, "\n")
Precision on Test Data: 0.5693069
> cat("Recall on Test Data:", recall_test, "\n")
Recall on Test Data: 0.9126984
> cat("F1 Score on Test Data:", f1_score_test, "\n")
F1 Score on Test Data: 0.7012195
> cat("Confusion Matrix on Test Data:\n")
Confusion Matrix on Test Data:
> print(conf_matrix_test)

predictions_test   0   1
               0 354  87
               1  11 115
```

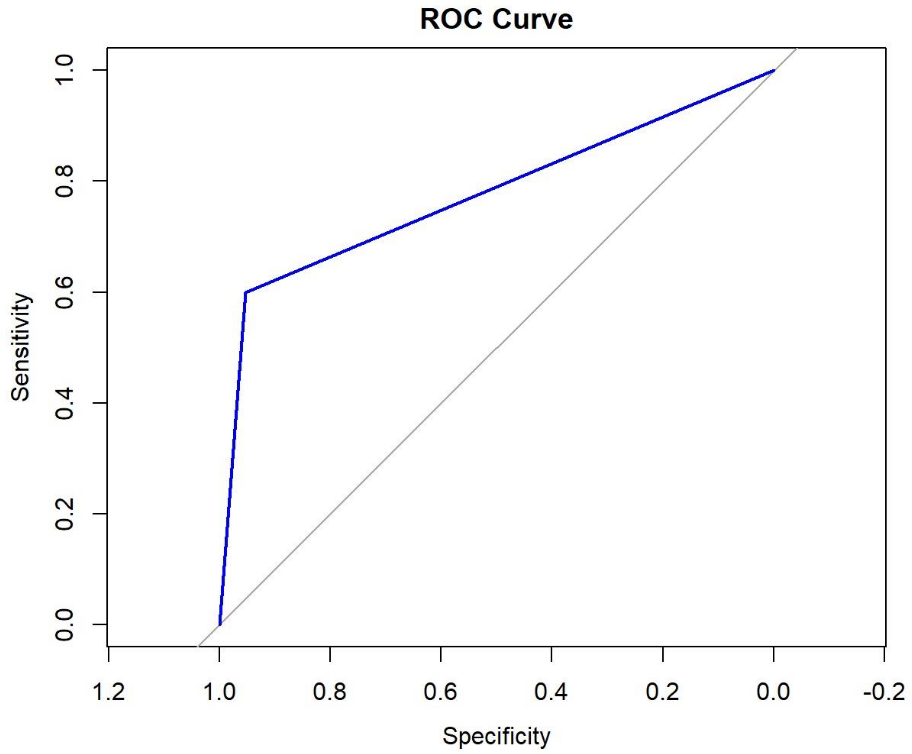
RF WITH CV AND TUNING :

```
> library(caret)
> ctrl <- trainControl(method = "cv", number = 10)
> param_grid <- expand.grid(mtry = c(1,2,3, 4,5, 6,7, 8))
> rf_model_tuned <- train(TravelInsurance ~ ., data = train_data, method = "rf",
+                         trControl = ctrl, tuneGrid = param_grid)
> print(rf_model_tuned)
Random Forest

1320 samples
   8 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1188, 1189, 1188, 1188, 1188, 1188, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  1     0.7878938  0.4800785
  2     0.8219736  0.5796921
  3     0.8242578  0.5907654
  4     0.8128939  0.5714048
  5     0.8075505  0.5650577
  6     0.8037797  0.5585706
  7     0.8030222  0.5570132
  8     0.8007494  0.5515060

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 3.
```



Trained the model using the best feature size(mtry = 3)

```
> ctrl <- trainControl(method = "cv", number = 10)
>
> # Train the final random forest model with k-fold cross-validation
> final_rf_model_cv <- train(TravelInsurance ~ ., data = train_data, method =
"rf",
+                           trControl = ctrl, tuneGrid = data.frame(mtry = 3),
ntree = 2000)
>
> # Print the final model results
> print(final_rf_model_cv)
Random Forest

1320 samples
   8 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1188, 1188, 1188, 1188, 1189, 1188, ...
Resampling results:

  Accuracy   Kappa
  0.8250555  0.5939506

Tuning parameter 'mtry' was held constant at a value of 3

> predictions_final_cv <- predict(final_rf_model_cv, newdata = test_data)
> conf_matrix_final_cv <- table(predictions_final_cv,
test_data$TravelInsurance)
> accuracy_final_cv <- sum(diag(conf_matrix_final_cv)) /
sum(conf_matrix_final_cv)
> cat("Final Model Accuracy with Cross-Validation:", accuracy_final_cv, "\n")
Final Model Accuracy with Cross-Validation: 0.8271605
> print("Confusion Matrix:")
[1] "Confusion Matrix:"
> print(conf_matrix_final_cv)

predictions_final_cv   0   1
                   0 348  81
                   1  17 121

> precision_final_cv <- conf_matrix_final_cv[2, 2] /
sum(conf_matrix_final_cv[, 2])
> recall_final_cv <- conf_matrix_final_cv[2, 2] / sum(conf_matrix_final_cv[2,
])
> f1_score_final_cv <- 2 * (precision_final_cv * recall_final_cv) /
(precision_final_cv + recall_final_cv)
> roc_curve_final_cv <- roc(as.numeric(test_data$TravelInsurance == "1"),
as.numeric(predictions_final_cv == "1"))
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc_final_cv <- auc(roc_curve_final_cv)
> cat("Precision:", precision_final_cv, "\n")
Precision: 0.5990099
> cat("Recall:", recall_final_cv, "\n")
Recall: 0.8768116
> cat("F1 Score:", f1_score_final_cv, "\n")
F1 Score: 0.7117647
> cat("AUC:", auc_final_cv, "\n")
AUC: 0.7762173

> plot(roc_curve_final_cv, main = "ROC Curve", col = "blue", lwd = 2)
```



Gradient boosting method

GBM without CV AND SHRINKAGE

```
> set.seed(12)
> train_index <- sample(1:nrow(TravelInsuranceData), 0.7 *
nrow(TravelInsuranceData))
> train_data <- TravelInsuranceData[train_index, ]
> test_data <- TravelInsuranceData[-train_index, ]
> library(gbm)
>
> # Convert categorical variables to factors
> train_data$Employment.Type <- as.factor(train_data$Employment.Type)
> train_data$GraduateOrNot <- as.factor(train_data$GraduateOrNot)
> train_data$FrequentFlyer <- as.factor(train_data$FrequentFlyer)
> train_data$EverTravelledAbroad <- as.factor(train_data$EverTravelledAbroad)
> X_train <- train_data[, -which(names(train_data) == "TravelInsurance")]
> y_train <- train_data$TravelInsurance
> model <- gbm(y_train ~ ., data = X_train, distribution = "bernoulli",
n.trees = 100)
>
> test_data$Employment.Type <- as.factor(test_data$Employment.Type)
> test_data$GraduateOrNot <- as.factor(test_data$GraduateOrNot)
> test_data$FrequentFlyer <- as.factor(test_data$FrequentFlyer)
> test_data$EverTravelledAbroad <- as.factor(test_data$EverTravelledAbroad)
> X_test <- test_data[, -which(names(test_data) == "TravelInsurance")]
> pred_probs <- predict(model, newdata = X_test, type = "response", n.trees =
100)
> predictions <- ifelse(pred_probs > 0.455, 1, 0)
> conf_matrix <- table(predictions, test_data$TravelInsurance)
```

```
> accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
> cat("Accuracy on the test set:", accuracy, "\n")
Accuracy on the test set: 0.8289242
> # Calculate Precision, Recall, and F1 Score
> precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
> recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
> f1_score <- 2 * (precision * recall) / (precision + recall)
> cat("Precision:", precision, "\n")
Precision: 0.5841584
> cat("Recall:", recall, "\n")
Recall: 0.9007634
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7087087
> library(pROC)
> roc_curve <- roc(test_data$TravelInsurance, pred_probs)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc_value <- auc(roc_curve)
> cat("AUC:", auc_value, "\n")
AUC: 0.789231
> plot(roc_curve, main = "ROC Curve", col = "blue", lwd = 2)
> abline(a = 0, b = 1, lty = 2, col = "red")  # Diagonal line for reference
> print(conf_matrix)

predictions   0   1
          0 352  84
          1  13 118
```

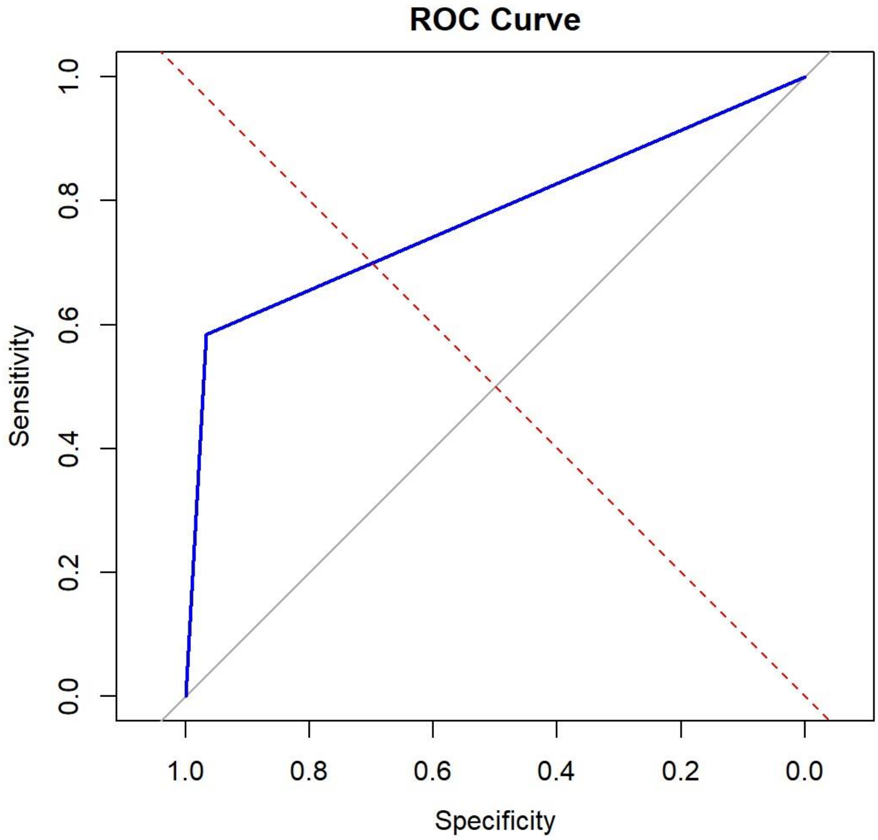
GBM WITH CV AND SHRINKAGE

```
> train_data$Employment.Type <- as.factor(train_data$Employment.Type)
> train_data$GraduateOrNot <- as.factor(train_data$GraduateOrNot)
> train_data$FrequentFlyer <- as.factor(train_data$FrequentFlyer)
> train_data$EverTravelledAbroad <- as.factor(train_data$EverTravelledAbroad)
> ctrl <- trainControl(method = "cv", number = 10)
> X_train <- train_data[, -which(names(train_data) == "TravelInsurance")]
> y_train <- as.factor(train_data$TravelInsurance)

> tuneGrid <- expand.grid(n.trees = c(150, 200, 300),interaction.depth =
c(3,4,5),shrinkage = c(0.001, 0.005,0.01, 0.1),n.minobsinnode = c(5,10,15))
> gbm_cv_model <- train(x = X_train, y = y_train, method = "gbm", trControl =
ctrl, tuneGrid = tuneGrid, verbose = FALSE )
There were 50 or more warnings (use warnings() to see the first 50)
> print(gbm_cv_model)
Stochastic Gradient Boosting

1320 samples
   8 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1188, 1189, 1187, 1188, 1188, 1188, ...
Resampling results across tuning parameters:

  shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy   Kappa
  0.001      3                   5              150      0.6371228  0.0000000
  0.001      3                   5              200      0.6371228  0.0000000
  0.001      3                   5              300      0.7803177  0.4614652
  0.001      3                  10              150      0.6371228  0.0000000
  0.001      3                  10              200      0.6371228  0.0000000
  0.001      3                  10              300      0.7825962  0.4705822
  0.001      3                  15              150      0.6371228  0.0000000
  0.001      3                  15              200      0.6371228  0.0000000
  0.001      3                  15              300      0.7825962  0.4705822
  0.001      4                   5              150      0.6371228  0.0000000
```

0.001	4	5	200	0.6371228	0.0000000
0.001	4	5	300	0.7818328	0.4651225
0.001	4	10	150	0.6371228	0.0000000
0.001	4	10	200	0.6371228	0.0000000
0.001	4	10	300	0.7689309	0.4273069
0.001	4	15	150	0.6371228	0.0000000
0.001	4	15	200	0.6371228	0.0000000
0.001	4	15	300	0.7674215	0.4236248
0.001	5	5	150	0.6371228	0.0000000
0.001	5	5	200	0.6371228	0.0000000
0.001	5	5	300	0.7833596	0.4694716
0.001	5	10	150	0.6371228	0.0000000
0.001	5	10	200	0.6371228	0.0000000
0.001	5	10	300	0.7696942	0.4281669
0.001	5	15	150	0.6371228	0.0000000
0.001	5	15	200	0.6371228	0.0000000
0.001	5	15	300	0.7689366	0.4254708
0.005	3	5	150	0.8280340	0.5981137
0.005	3	5	200	0.8280340	0.5981137
0.005	3	5	300	0.8280340	0.5981137
0.005	3	10	150	0.8280340	0.5981137
0.005	3	10	200	0.8280340	0.5981137
0.005	3	10	300	0.8280340	0.5981137
0.005	3	15	150	0.8280340	0.5981137
0.005	3	15	200	0.8280340	0.5981137
0.005	3	15	300	0.8280340	0.5981137
0.005	4	5	150	0.8250037	0.5900906
0.005	4	5	200	0.8257613	0.5921224
0.005	4	5	300	0.8280340	0.5981137
0.005	4	10	150	0.8280340	0.5981137
0.005	4	10	200	0.8280340	0.5981137
0.005	4	10	300	0.8280340	0.5981137
0.005	4	15	150	0.8280340	0.5981137
0.005	4	15	200	0.8280340	0.5981137
0.005	4	15	300	0.8280340	0.5981137
0.005	5	5	150	0.8234828	0.5859918
0.005	5	5	200	0.8242461	0.5880390
0.005	5	5	300	0.8280340	0.5981137
0.005	5	10	150	0.8280340	0.5981137
0.005	5	10	200	0.8280340	0.5981137
0.005	5	10	300	0.8280340	0.5981137
0.005	5	15	150	0.8280340	0.5981137
0.005	5	15	200	0.8280340	0.5981137
0.005	5	15	300	0.8280340	0.5981137
0.010	3	5	150	0.8280340	0.5981137
0.010	3	5	200	0.8280340	0.5981137
0.010	3	5	300	0.8280340	0.5981137
0.010	3	10	150	0.8280340	0.5981137
0.010	3	10	200	0.8280340	0.5981137
0.010	3	10	300	0.8280340	0.5981137
0.010	3	15	150	0.8280340	0.5981137
0.010	3	15	200	0.8280340	0.5981137
0.010	3	15	300	0.8280340	0.5981137
0.010	4	5	150	0.8280340	0.5981137
0.010	4	5	200	0.8280340	0.5981137
0.010	4	5	300	0.8280340	0.5981137
0.010	4	10	150	0.8280340	0.5981137
0.010	4	10	200	0.8280340	0.5981137
0.010	4	10	300	0.8280340	0.5981137
0.010	4	15	150	0.8280340	0.5981137
0.010	4	15	200	0.8280340	0.5981137
0.010	4	15	300	0.8280340	0.5981137
0.010	5	5	150	0.8280340	0.5981137
0.010	5	5	200	0.8280340	0.5981137
0.010	5	5	300	0.8280340	0.5981137
0.010	5	10	150	0.8280340	0.5981137
0.010	5	10	200	0.8280340	0.5981137
0.010	5	10	300	0.8280340	0.5981137

```
  0.010      5               15               150        0.8280340  0.5981137
  0.010      5               15               200        0.8280340  0.5981137
  0.010      5               15               300        0.8280340  0.5981137
  0.100      3                5               150        0.8212101  0.5850909
  0.100      3                5               200        0.8159185  0.5720436
  0.100      3                5               300        0.8060983  0.5530402
  0.100      3               10               150        0.8219848  0.5861617
  0.100      3               10               200        0.8235056  0.5896827
  0.100      3               10               300        0.8189488  0.5799479
  0.100      3               15               150        0.8197121  0.5819823
  0.100      3               15               200        0.8151551  0.5729927
  0.100      3               15               300        0.8143975  0.5716049
  0.100      4                5               150        0.8174393  0.5769764
  0.100      4                5               200        0.8174451  0.5793464
  0.100      4                5               300        0.8030512  0.5494138
  0.100      4               10               150        0.8235056  0.5906380
  0.100      4               10               200        0.8121419  0.5660034
  0.100      4               10               300        0.8045662  0.5507005
  0.100      4               15               150        0.8181912  0.5784374
  0.100      4               15               200        0.8159127  0.5741610
  0.100      4               15               300        0.8068388  0.5566704
  0.100      5                5               150        0.8068274  0.5539539
  0.100      5                5               200        0.8000091  0.5419552
  0.100      5                5               300        0.7977708  0.5360100
  0.100      5               10               150        0.8182142  0.5789142
  0.100      5               10               200        0.8091060  0.5611459
  0.100      5               10               300        0.8030453  0.5471333
  0.100      5               15               150        0.8113729  0.5629662
  0.100      5               15               200        0.8121190  0.5659409
  0.100      5               15               300        0.8053007  0.5526422

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 150, interaction.depth = 3,
shrinkage
 = 0.005 and n.minobsinnode = 5.

> X_test <- test_data[, -which(names(test_data) == "TravelInsurance")]


> predictions <- predict(gbm_cv_model, newdata = X_test)
> conf_matrix <- table(predictions, test_data$TravelInsurance)
> print(conf_matrix)

predictions   0   1
          0 353  84
          1  12 118
> precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
> precision
[1] 0.5841584
> accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
> accuracy
[1] 0.8306878
> recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
> recall
[1] 0.9076923
> f1_score <- 2 * (precision * recall) / (precision + recall)
> f1_score
[1] 0.7108434
> library(pROC)
>
> X_test <- test_data[, -which(names(test_data) == "TravelInsurance")]
> pred_probs <- predict(gbm_cv_model, newdata = X_test, type = "raw")
>
> # Calculate AUC
> roc_curve <- roc(test_data$TravelInsurance, as.numeric(pred_probs))
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
> auc_value <- auc(roc_curve)
> cat("AUC:", auc_value, "\n")
AUC: 0.7756409
>
> # Plot ROC Curve
> plot(roc_curve, main = "ROC Curve", col = "blue", lwd = 2)
> abline(a = 0, b = 1, lty = 2, col = "red")  # Diagonal line for reference
```




```
> summary(gbm_cv_model)
                                   var    rel.inf
AnnualIncome               AnnualIncome 71.7385309
FamilyMembers             FamilyMembers 16.9126674
Age                                 Age 10.5625873
EverTravelledAbroad EverTravelledAbroad  0.7862144
Employment.Type         Employment.Type  0.0000000
GraduateOrNot             GraduateOrNot  0.0000000
ChronicDiseases         ChronicDiseases  0.0000000
FrequentFlyer             FrequentFlyer  0.0000000
```

SVM METHOD

```
> set.seed(12)
>
> train_indices <- sample(1:nrow(TravelInsuranceData), 1320)

> train_set <- TravelInsuranceData[train_indices, ]
> test_set <- TravelInsuranceData[-train_indices, ]
```

```
>
> # Ensure that TravelInsurance is a factor
> train_set$TravelInsurance <- as.factor(train_set$TravelInsurance)
> test_set$TravelInsurance <- as.factor(test_set$TravelInsurance)
>
> # Train SVM for classification
> svm_model <- svm(TravelInsurance ~ ., data = train_set, kernel = "linear",
cost = 0.01)
> summary(svm_model)

Call:
svm(formula = TravelInsurance ~ ., data = train_set, kernel = "linear",
    cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  898

 ( 449 449 )


Number of Classes:  2

Levels:
 0 1

> prediction <- predict(svm_model, newdata = test_set)
> test_error_rate <- mean(prediction != test_set$TravelInsurance)
> test_error_rate
[1] 0.2627866

> conf_matrix <- table(Actual = test_set$TravelInsurance, Predicted =
+                          prediction)
>
> print(conf_matrix)
      Predicted
Actual   0   1
     0 332  33
     1 116  86
```

SVM WITH COST VALUES

```
> cost_values <- c(0.001, 0.01, 0.1, 1, 10)

> tune_result <- tune(svm, TravelInsurance ~ ., data = train_set, kernel =
+ "linear", ranges = list(cost = cost_values), tunecontrol =
+                         tune.control(sampling = "cross", cross = 10))

> cross_val_errors <- tune_result$performances
> best_cost <- tune_result$best.parameters$cost
> print(cross_val_errors)
   cost     error dispersion
1 1e-03 0.3628788 0.05730154
2 1e-02 0.2378788 0.04332851
3 1e-01 0.2469697 0.04228573
4 1e+00 0.2469697 0.04228573
5 1e+01 0.2469697 0.04228573
>
> print(best_cost)
[1] 0.01
> print(tune_result)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.2378788

> optimal_cost <- 0.01
>
> svm_model_optimal <- svm(TravelInsurance~., data = train_set, kernel =
+                             "linear", cost = optimal_cost)
>
> test_svm_optimal_prediction <- predict(svm_model_optimal, newdata
+                                       = test_set)
>
> test_error_rate_optimal <- mean(test_svm_optimal_prediction !=
+                                    test_set$TravelInsurance)
>
> test_error_rate_optimal
[1] 0.2627866

> cost_values <- c(0.0001, 0.0005,0.001,0.005,0.01)
> tune_result <- tune(svm, TravelInsurance ~ ., data = train_set, kernel =
"linear", ranges = list(cost = cost_values), tunecontrol =
tune.control(sampling = "cross", cross = 10))
> cross_val_errors <- tune_result$performances
> best_cost <- tune_result$best.parameters$cost
> print(cross_val_errors)
   cost     error dispersion
1 1e-04 0.3628788 0.05302429
2 5e-04 0.3628788 0.05302429
3 1e-03 0.3628788 0.05302429
4 5e-03 0.2272727 0.04573427
5 1e-02 0.2363636 0.05009938
> print(best_cost)
[1] 0.005
> print(tune_result)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
  cost
 0.005

- best performance: 0.2272727

> optimal_cost <- 0.005
> svm_model_optimal <- svm(TravelInsurance~., data = train_set, kernel =
"linear", cost = optimal_cost)
> test_svm_optimal_prediction <- predict(svm_model_optimal, newdata =
test_set)
> test_error_rate_optimal <- mean(test_svm_optimal_prediction !=
test_set$TravelInsurance)
> test_error_rate_optimal
[1] 0.2222222

> cost_values <- c(0.001,0.004,0.005,0.006, 0.007)
> tune_result <- tune(svm, TravelInsurance ~ ., data = train_set, kernel =
"linear", ranges = list(cost = cost_values), tunecontrol =
```

```
tune.control(sampling = "cross", cross = 10))
> cross_val_errors <- tune_result$performances
> best_cost <- tune_result$best.parameters$cost
> print(cross_val_errors)
   cost     error dispersion
1 0.001 0.3628788 0.05444833
2 0.004 0.2462121 0.04137871
3 0.005 0.2310606 0.03964727
4 0.006 0.2250000 0.04256380
5 0.007 0.2257576 0.04192223
> print(best_cost)
[1] 0.006
> print(tune_result)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
  cost
 0.006

- best performance: 0.225

> optimal_cost <- 0.006
> svm_model_optimal <- svm(TravelInsurance~., data = train_set, kernel =
"linear", cost = optimal_cost)
> test_svm_optimal_prediction <- predict(svm_model_optimal, newdata =
test_set)
> test_error_rate_optimal <- mean(test_svm_optimal_prediction !=
test_set$TravelInsurance)
> test_error_rate_optimal
[1] 0.2222222

> conf_matrix <- table(Actual = test_set$TravelInsurance, Predicted =
test_svm_optimal_prediction)
> print(conf_matrix)
      Predicted
Actual   0   1
     0 344  21
     1 105  97

> optimal_cost <- 0.006
> svm_model_optimal <- svm(TravelInsurance~., data = train_set, kernel =
"linear", cost = optimal_cost)
> test_svm_optimal_prediction <- predict(svm_model_optimal, newdata =
test_set)
> test_error_rate_optimal <- mean(test_svm_optimal_prediction !=
test_set$TravelInsurance)
> test_error_rate_optimal
[1] 0.2222222
> conf_matrix <- table(Actual = test_set$TravelInsurance, Predicted =
test_svm_optimal_prediction)
> print(conf_matrix)
      Predicted
Actual   0   1
     0 344  21
     1 105  97
> precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
> cat("Precision:", precision, "\n")
Precision: 0.8220339
>
> recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
> cat("Recall:", recall, "\n")
Recall: 0.480198
> f1_score <- 2 * (precision * recall) / (precision + recall)
```

```
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.60625
> Accuracy <- 1 - test_error_rate_optimal
> Accuracy
[1] 0.7777778
```

Svm with polynomial ad cost

```
> degree_values <- c(1, 2, 3, 4, 5, 6)
>
> cost_values <- c(0.1, 1, 10, 100, 1000)
> tune_result_poly <- tune(svm, TravelInsurance ~ ., data = train_set,
+                          kernel = "polynomial", ranges = list(degree =
degree_values, cost =
+
cost_values), tunecontrol = tune.control(sampling = "cross", cross =
+
10))
>
> cross_val_errors_poly <- tune_result_poly$performances
>
> best_degree_poly <- tune_result_poly$best.parameters$degree
> best_cost_poly <- tune_result_poly$best.parameters$cost
>
> print(cross_val_errors_poly)
   degree  cost     error dispersion
1       1 1e-01 0.2393939 0.03410494
2       2 1e-01 0.2212121 0.02715087
3       3 1e-01 0.2295455 0.02475520
4       4 1e-01 0.2484848 0.03249638
5       5 1e-01 0.2598485 0.03114364
6       6 1e-01 0.2750000 0.03500004
7       1 1e+00 0.2469697 0.03237843
8       2 1e+00 0.2196970 0.02923191
9       3 1e+00 0.2022727 0.02526515
10      4 1e+00 0.2015152 0.02207250
11      5 1e+00 0.2068182 0.02021780
12      6 1e+00 0.2212121 0.01667432
13      1 1e+01 0.2469697 0.03237843
14      2 1e+01 0.2174242 0.02858113
15      3 1e+01 0.1901515 0.02844694
16      4 1e+01 0.1871212 0.02813136
17      5 1e+01 0.1893939 0.03425419
18      6 1e+01 0.1893939 0.03589058
19      1 1e+02 0.2469697 0.03237843
20      2 1e+02 0.2159091 0.02772033
21      3 1e+02 0.1909091 0.03603244
22      4 1e+02 0.2060606 0.03908843
23      5 1e+02 0.2000000 0.03538959
24      6 1e+02 0.2000000 0.03484482
25      1 1e+03 0.2469697 0.03237843
26      2 1e+03 0.2143939 0.02673669
27      3 1e+03 0.1946970 0.03312800
28      4 1e+03 0.2128788 0.03701886
29      5 1e+03 0.2037879 0.03489055
30      6 1e+03 0.2143939 0.02423453
>
> print(best_degree_poly)
[1] 4
>
> print(best_cost_poly)
[1] 10
> optimal_degree_poly <- 4
> optimal_cost_poly <- 10
> svm_model_optimal_poly <- svm(TravelInsurance ~ ., data = train_set,
+                               kernel = "polynomial", degree =
optimal_degree_poly, cost =
+                                  optimal_cost_poly)
```

```
>
> test_predictions_optimal_poly <- predict(svm_model_optimal_poly,
+                                          newdata = test_set)
>
> test_error_rate_optimal_poly <- mean(test_predictions_optimal_poly
+                                      != test_set$TravelInsurance)
>
> print(test_error_rate_optimal_poly)
[1] 0.1869489
>
> conf_matrix <- table(Actual = test_set$TravelInsurance, Predicted =
test_predictions_optimal_poly)
> print(conf_matrix)
      Predicted
Actual   0   1
     0 346  19
     1  87 115
> accuracy <- 1 - test_error_rate_optimal_poly
> accuracy
[1] 0.8130511

> precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])

> precision
[1] 0.858209
> recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
> recall
[1] 0.5693069
> f1_score <- 2 * (precision * recall) / (precision + recall)
> f1_score
[1] 0.6845238
```

SVM WITH RADIAL AND COST

```
> gamma_values <- c(0.1, 1, 10, 100, 1000)
> cost_values <- c(0.1, 1, 10, 100, 1000)
> tune_result_radial <- tune(svm, TravelInsurance ~ ., data = train_set,
+                           kernel = "radial", ranges = list(gamma =
gamma_values, cost =
+                                                             cost_values),
tunecontrol = tune.control(sampling = "cross", cross =
+
10))
>
> cross_val_errors_radial <- tune_result_radial$performances
> best_gamma_radial <- tune_result_radial$best.parameters$gamma
> best_cost_radial <- tune_result_radial$best.parameters$cost
> print("Cross-Validation Errors (Radial Kernel):")
[1] "Cross-Validation Errors (Radial Kernel):"
> print(cross_val_errors_radial)
   gamma  cost     error dispersion
1  1e-01 1e-01 0.2196970 0.03829735
2  1e+00 1e-01 0.2818182 0.04176984
3  1e+01 1e-01 0.3628788 0.04292187
4  1e+02 1e-01 0.3628788 0.04292187
5  1e+03 1e-01 0.3628788 0.04292187
6  1e-01 1e+00 0.2060606 0.03725068
7  1e+00 1e+00 0.1924242 0.04520131
8  1e+01 1e+00 0.2590909 0.03567673
9  1e+02 1e+00 0.2765152 0.03867018
10 1e+03 1e+00 0.2765152 0.03867018
11 1e-01 1e+01 0.1886364 0.03903129
12 1e+00 1e+01 0.2257576 0.04099940
13 1e+01 1e+01 0.2689394 0.04229327
14 1e+02 1e+01 0.2772727 0.03882653
15 1e+03 1e+01 0.2765152 0.03867018
16 1e-01 1e+02 0.1916667 0.04271336
```

```
17 1e+00 1e+02 0.2250000 0.04301091
18 1e+01 1e+02 0.2719697 0.03967942
19 1e+02 1e+02 0.2772727 0.03882653
20 1e+03 1e+02 0.2765152 0.03867018
21 1e-01 1e+03 0.1969697 0.03912104
22 1e+00 1e+03 0.2454545 0.04846906
23 1e+01 1e+03 0.2719697 0.03967942
24 1e+02 1e+03 0.2772727 0.03882653
25 1e+03 1e+03 0.2765152 0.03867018
> print(best_gamma_radial)
[1] 0.1
> print(best_cost_radial)
[1] 10
>
> optimal_gamma <- 0.1
> optimal_cost <- 10
> svm_model_optimal_radial <- svm(TravelInsurance ~ ., data = train_set,
+                                  kernel = "radial", gamma = optimal_gamma,
cost = optimal_cost)
> test_predictions_optimal_radial <-
+    predict(svm_model_optimal_radial, newdata = test_set)


> test_error_rate_optimal_radial <-
+    mean(test_predictions_optimal_radial != test_set$TravelInsurance)
> test_error_rate_optimal_radial
[1] 0.1940035
> accuracy <- 1 - test_error_rate_optimal_radial
> accuracy
[1] 0.8059965

> conf_matrix <- table(Actual = test_set$TravelInsurance, Predicted =
test_predictions_optimal_radial)
> print(conf_matrix)
      Predicted
Actual   0   1
     0 341  24
     1  86 116

> precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
> precision
[1] 0.8285714
> recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
>
> recall
[1] 0.5742574
> f1_score <- 2 * (precision * recall) / (precision + recall)
>
> f1_score
[1] 0.6783626
```

Logistic Regression:

```
> set.seed(12)
>
> train_indices <- sample(1:nrow(TravelInsuranceData), 1320)
> train_set <- TravelInsuranceData[train_indices, ]
> test_set <- TravelInsuranceData[-train_indices, ]
> train_set$TravelInsurance <- as.factor(train_set$TravelInsurance)
>
> test_set$TravelInsurance <- as.factor(test_set$TravelInsurance)
> log_model <- glm(TravelInsurance~., data = train_set, family = binomial)

> caret::varImp(log_model)
                                                   Overall
Age                                              2.7708663
```

```
Employment.TypePrivate Sector/Self Employed 0.5815850
GraduateOrNotYes                            0.6790549
AnnualIncome                                6.8090922
FamilyMembers                               3.8497335
ChronicDiseases                             0.8501738
FrequentFlyerYes                            1.2055837
EverTravelledAbroadYes                      9.7406964
> test.predictions <- predict(log_model, newdata = test_set, type =
"response")
>
> predicted.classes <- ifelse(test.predictions > 0.415, 1, 0)
> actual.classes <- test_set$TravelInsurance
> conf_matrix <- table(Actual = actual.classes, Predicted =
+                        predicted.classes)
> print(conf_matrix)
      Predicted
Actual   0   1
     0 313  52
     1  86 116

> accuracy <- sum(diag(conf_matrix))/sum(conf_matrix)
> accuracy
[1] 0.7566138
> precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
> precision
[1] 0.6882353
> recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
> recall
[1] 0.5792079
> f1_score <- 2 * (precision * recall) / (precision + recall)
> f1_score
[1] 0.6290323
> summary(log_model)

Call:
glm(formula = TravelInsurance ~ ., family = binomial, data = train_set)

Coefficients:
                                             Estimate Std. Error z value
Pr(>|z|)
(Intercept)                                 -4.986e+00  7.648e-01  -6.519
7.10e-11 ***
Age                                          6.236e-02  2.250e-02   2.771
0.005591 **
Employment.TypePrivate Sector/Self Employed  9.366e-02  1.610e-01   0.582
0.560846
GraduateOrNotYes                            -1.293e-01  1.905e-01  -0.679
0.497103
AnnualIncome                                 1.456e-06  2.138e-07   6.809
9.82e-12 ***
FamilyMembers                                1.574e-01  4.089e-02   3.850
0.000118 ***
ChronicDiseases                              1.242e-01  1.461e-01   0.850
0.395228
FrequentFlyerYes                             2.091e-01  1.735e-01   1.206
0.227978
EverTravelledAbroadYes                       1.904e+00  1.955e-01   9.741  <
2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1729.3  on 1319  degrees of freedom
Residual deviance: 1383.7  on 1311  degrees of freedom
AIC: 1401.7
```

```
Number of Fisher Scoring iterations: 4
```

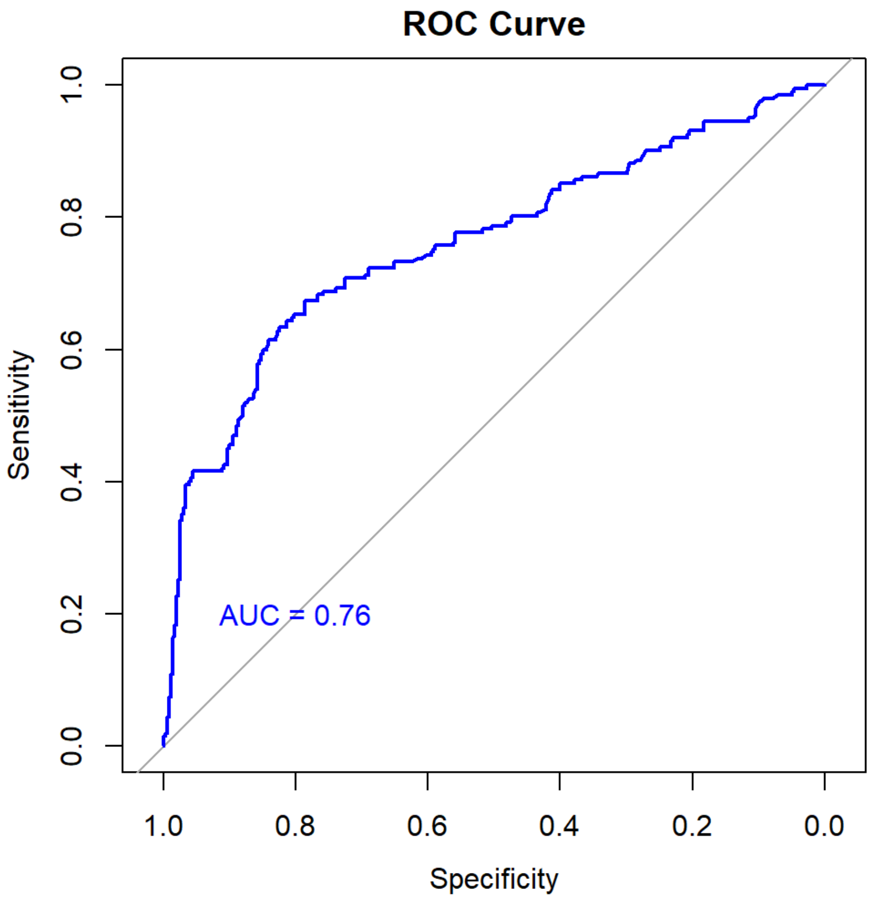
Logistic regression with CV and features reduction

```
> cv_model <- train(TravelInsurance ~ AnnualIncome+FamilyMembers+Age, data =
train_set,trControl = train_control,method = "glm",family = binomial())
> print(cv_model)
Generalized Linear Model

1320 samples
   3 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1189, 1188, 1188, 1187, 1188, 1188, ...
Resampling results:

  Accuracy   Kappa
  0.7386388  0.4042625

> test.predictions.cv <- predict(cv_model, newdata = test_set, type = "raw")

> actual.classes <- test_set$TravelInsurance

> conf_matrix <- table(Actual = actual.classes, Predicted =
test.predictions.cv)
> conf_matrix
      Predicted
Actual   0   1
     0 322  43
     1  96 106
> accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
> accuracy
[1] 0.7548501
```

Optimal cutoff for Logistic regression with ROC

```
> roc_curve <- roc(test_set$TravelInsurance, test.predictions)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
>
> # Plot the ROC curve
> plot(roc_curve, main = "ROC Curve", col = "blue", lwd = 2)
> text(0.8, 0.2, paste("AUC =", round(auc(roc_curve), 2)), col = "blue")
> optimal_cutoff <- coords(roc_curve, "best", best.method =
"youden")$threshold
> cat("Optimal Cutoff:", optimal_cutoff, "\n")
Optimal Cutoff: 0.3412594
> test_predicted_classes <- ifelse(test.predictions > optimal_cutoff, 1, 0)
> conf_matrix <- table(Actual = test_set$TravelInsurance, Predicted =
test_predicted_classes)
> print(conf_matrix)
      Predicted
Actual   0   1
     0 287  78
     1  66 136
>
> # Calculate accuracy
> accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
```

```
> print(accuracy)
[1] 0.7460317
```



Linear discriminant analysis

```
> library(MASS)

> set.seed(12)
>
> train_indices <- sample(1:nrow(TravelInsuranceData), 1320)
> train_set <- TravelInsuranceData[train_indices, ]
> test_set <- TravelInsuranceData[-train_indices, ]
> train_set$TravelInsurance <- as.factor(train_set$TravelInsurance)
> test_set$TravelInsurance <- as.factor(test_set$TravelInsurance)
> lda_model <- lda(TravelInsurance ~ ., data = train_set)
> lda_predictions <- predict(lda_model, newdata = test_set)
>
> lda_predicted_classes <- lda_predictions$class
>
> actual_classes <- test_set$TravelInsurance
> conf_matrix_lda <- table(Actual = actual_classes, Predicted =
+                             lda_predicted_classes)
>
> print(conf_matrix_lda)
      Predicted
Actual   0   1
     0 329  36
     1 114  88
> accuracy_lda <- sum(diag(conf_matrix_lda)) / sum(conf_matrix_lda)
> accuracy_lda
[1] 0.7354497
```

```
> precision_lda <- conf_matrix_lda[2, 2] / sum(conf_matrix_lda[, 2])
> recall_lda <- conf_matrix_lda[2, 2] / sum(conf_matrix_lda[2, ])
> f1_score_lda <- 2 * (precision_lda * recall_lda) / (precision_lda +
recall_lda)
> accuracy_lda <- sum(diag(conf_matrix_lda)) / sum(conf_matrix_lda)
>
> cat("Precision:", precision_lda, "\n")
Precision: 0.7073171
> cat("Recall:", recall_lda, "\n")
Recall: 0.4306931
> cat("F1 Score:", f1_score_lda, "\n")
F1 Score: 0.5353846
```

Linear discriminant analysis with CV and feature redcution

```
> train_control <- trainControl(method = "cv", number = 10)

> cv_model <- train(TravelInsurance ~AnnualIncome+FamilyMembers+Age, data =
train_set,trControl = train_control,method = "lda")
> test.predictions.cv.lda <- predict(cv_model, newdata = test_set, type =
"raw")
> actual.classes <- test_set$TravelInsurance
> conf_matrix_lda_cv <- table(Actual = actual.classes, Predicted =
test.predictions.cv.lda)
> print(conf_matrix_lda_cv)
      Predicted
Actual   0   1
     0 321  44
     1  93 109
> accuracy_lda_cv <- sum(diag(conf_matrix_lda_cv)) / sum(conf_matrix_lda_cv)
> accuracy_lda_cv
[1] 0.7583774

> precision_lda_cv <- conf_matrix_lda_cv[2, 2] / sum(conf_matrix_lda_cv[, 2])
>
> # Recall
> recall_lda_cv <- conf_matrix_lda_cv[2, 2] / sum(conf_matrix_lda_cv[2, ])
>
> # Test Error Rate
> test_error_rate_lda_cv <- 1 - accuracy_lda_cv
>
> # F1 Score
> f1_score_lda_cv <- 2 * (precision_lda_cv * recall_lda_cv) /
(precision_lda_cv + recall_lda_cv)
>
> # Display the results
> cat("Precision:", precision_lda_cv, "\n")
Precision: 0.7124183
> cat("Recall:", recall_lda_cv, "\n")
Recall: 0.539604
> cat("Test Error Rate:", test_error_rate_lda_cv, "\n")
Test Error Rate: 0.2416226
> cat("F1 Score:", f1_score_lda_cv, "\n")
F1 Score: 0.6140845
```

Optimal cutoff for Linear Discriminant Analysis with ROC

```
> library(pROC)
>
> # Fit LDA model
```

```
> lda_model <- lda(TravelInsurance ~ ., data = train_set)
> lda_predictions <- predict(lda_model, newdata = test_set)
>
> # Get predicted probabilities for class 1
> lda_probabilities <- lda_predictions$posterior[, 2]
>
> # Create a data frame with actual and predicted probabilities
> roc_data <- data.frame(actual = test_set$TravelInsurance, predicted =
lda_probabilities)
>
> # Create a ROC curve
> roc_curve <- roc(roc_data$actual, roc_data$predicted)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
>
> # Find the optimal cutoff based on Youden's J statistic
> optimal_cutoff <- coords(roc_curve, "best", best.method =
"youden")$threshold
> cat("Optimal Cutoff:", optimal_cutoff, "\n")
Optimal Cutoff: 0.3347771
>
> # Apply optimal cutoff to convert probabilities to predicted classes
> lda_predicted_classes <- ifelse(lda_probabilities > optimal_cutoff, 1, 0)
>
> # Create a confusion matrix
> conf_matrix_lda <- table(Actual = test_set$TravelInsurance, Predicted =
lda_predicted_classes)
> print(conf_matrix_lda)
      Predicted
Actual   0   1
     0 304  61
     1  75 127
>
> # Calculate accuracy
> accuracy_lda <- sum(diag(conf_matrix_lda)) / sum(conf_matrix_lda)
> print(accuracy_lda)
[1] 0.7601411

> # Precision
> precision_lda <- conf_matrix_lda[2, 2] / sum(conf_matrix_lda[, 2])
>
> # Recall
> recall_lda <- conf_matrix_lda[2, 2] / sum(conf_matrix_lda[2, ])
>
> # F1 Score
> f1_score_lda <- 2 * (precision_lda * recall_lda) / (precision_lda +
recall_lda)
>
> # Test Error Rate
> test_error_rate_lda <- 1 - accuracy_lda
>
> # Display the results
> cat("Precision:", precision_lda, "\n")
Precision: 0.6774194
> cat("Recall:", recall_lda, "\n")
Recall: 0.6237624
> cat("F1 Score:", f1_score_lda, "\n")
F1 Score: 0.6494845
> cat("Test Error Rate:", test_error_rate_lda, "\n")
Test Error Rate: 0.2398589
```

Predicting the target variable using the best model - GBM WITH CV AND SHRINKAGE

```
> set.seed(12)
> train_index <- sample(1:nrow(TravelInsuranceData), 0.7 *
nrow(TravelInsuranceData))
> train_data <- TravelInsuranceData[train_index, ]
>
> test_data <- TravelInsuranceData[-train_index, ]
> library(gbm)
> train_data$Employment.Type <- as.factor(train_data$Employment.Type)
> train_data$GraduateOrNot <- as.factor(train_data$GraduateOrNot)
> train_data$FrequentFlyer <- as.factor(train_data$FrequentFlyer)
> train_data$EverTravelledAbroad <- as.factor(train_data$EverTravelledAbroad)
>
> ctrl <- trainControl(method = "cv", number = 10)
> X_train <- train_data[, -which(names(train_data) == "TravelInsurance")]
> y_train <- as.factor(train_data$TravelInsurance)
> tuneGrid <- expand.grid(n.trees = c(150, 200, 300),interaction.depth =
c(3,4,5),shrinkage = c(0.001, 0.005,0.01, 0.1),n.minobsinnode = c(5,10,15))
> gbm_cv_model <- train(x = X_train, y = y_train, method = "gbm", trControl =
ctrl, tuneGrid = tuneGrid, verbose = FALSE )

> View(TravelInsuranceTest)

> X_test <- TravelInsuranceTest
> predictions <- predict(gbm_cv_model, newdata = X_test, type = "raw", n.trees
= gbm_cv_model$bestTune$n.trees)

> TravelInsuranceTest$Predicted_TravelInsurance <- predictions

> print(TravelInsuranceTest$Predicted_TravelInsurance)
  [1] 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
 [54] 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0 0 0
Levels: 0 1
> label_counts <- table(TravelInsuranceTest$Predicted_TravelInsurance)
> label_counts

 0  1
 81 19

> TravelInsuranceTest$Predicted_TravelInsurance <-
ifelse(TravelInsuranceTest$Predicted_TravelInsurance == 0, "No", "Yes")
>

> print(TravelInsuranceTest$Predicted_TravelInsurance)
  [1] "No"  "No"  "Yes" "Yes" "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "Yes"
"No"  "No"  "No"  "Yes" "No"
 [18] "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "Yes" "No"  "No"  "Yes"
"Yes" "No"  "No"  "No"  "No"
 [35] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "No"
"No"  "No"  "No"  "No"  "Yes"
 [52] "No"  "No"  "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "Yes" "Yes" "No"
"No"  "Yes" "No"  "No"  "Yes"
 [69] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"
"No"  "Yes" "No"  "No"  "No"
 [86] "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"
"No"  "No"  "No"
> label_counts <- table(TravelInsuranceTest$Predicted_TravelInsurance)
> label_counts

 No Yes
 81  19
```