Travel Insurance Purchase Forecast

Zahra Fatah

2023-11-20

A tour & travels company is offering travel insurance package to their customers. The new insurance package also includes COVID cover. The company wants to know which customers would be interested to buy it based on their database history. The insurance was offered to some of the customers in 2019 and the given data has been extracted from the performance/sales of the package during that period. The data is provided for almost 2000 of its previous customers and the goal is to build a model that can predict if the customer will be interested to buy the travel insurance package.

```
#par( mfrow= c(3,2) )

draw_confusion_matrix <- function(cm) {

  total <- sum(cm$table)
  res <- as.numeric(cm$table)

  # Generate color gradients. Palettes come from RColorBrewer.
  greenPalette <- c("#F7FCF5","#E5F5E0","#C7E9C0","#A1D99B","#74C476","#41AB5D","#238B45","#006D2C","#0
  redPalette <- c("#FFF5F0","#FEE0D2","#FCBBA1","#FC9272","#FB6A4A","#EF3B2C","#CB181D","#A50F15","#670
  getColor <- function (greenOrRed = "green", amount = 0) {
    if (amount == 0)
      return("#FFFFFF")
    palette <- greenPalette
    if (greenOrRed == "red")
      palette <- redPalette
    colorRampPalette(palette)(100)[10 + ceiling(90 * amount / total)]
  }

  # set the basic layout
  layout(matrix(c(1,1,2)))
  par(mar=c(2,2,2,2))
  plot(c(100, 345), c(300, 450), type = "n", xlab="", ylab="", xaxt='n', yaxt='n')
  title('CONFUSION MATRIX', cex.main=2)

  # create the matrix
  classes = colnames(cm$table)
  rect(150, 430, 240, 370, col=getColor("green", res[1]))
  text(195, 435, classes[1], cex=1.2)
  rect(250, 430, 340, 370, col=getColor("red", res[3]))
  text(295, 435, classes[2], cex=1.2)
  text(125, 370, 'Predicted', cex=1.3, srt=90, font=2)
  text(245, 450, 'Actual', cex=1.3, font=2)
  rect(150, 305, 240, 365, col=getColor("red", res[2]))
  rect(250, 305, 340, 365, col=getColor("green", res[4]))
  text(140, 400, classes[1], cex=1.2, srt=90)
```

```r
  text(140, 335, classes[2], cex=1.2, srt=90)

  # add in the cm results
  text(195, 400, res[1], cex=1.6, font=2, col='white')
  text(195, 335, res[2], cex=1.6, font=2, col='white')
  text(295, 400, res[3], cex=1.6, font=2, col='white')
  text(295, 335, res[4], cex=1.6, font=2, col='white')

  # add in the specifics
  plot(c(100, 0), c(100, 0), type = "n", xlab="", ylab="", main = "DETAILS", xaxt='n', yaxt='n')
  text(10, 85, names(cm$byClass[1]), cex=1.2, font=2)
  text(10, 70, round(as.numeric(cm$byClass[1]), 3), cex=1.2)
  text(30, 85, names(cm$byClass[2]), cex=1.2, font=2)
  text(30, 70, round(as.numeric(cm$byClass[2]), 3), cex=1.2)
  text(50, 85, names(cm$byClass[5]), cex=1.2, font=2)
  text(50, 70, round(as.numeric(cm$byClass[5]), 3), cex=1.2)
  text(70, 85, names(cm$byClass[6]), cex=1.2, font=2)
  text(70, 70, round(as.numeric(cm$byClass[6]), 3), cex=1.2)
  text(90, 85, names(cm$byClass[7]), cex=1.2, font=2)
  text(90, 70, round(as.numeric(cm$byClass[7]), 3), cex=1.2)

  # add in the accuracy information
  text(50, 35, names(cm$overall[1]), cex=1.5, font=2)
  text(50, 20, round(as.numeric(cm$overall[1]), 3), cex=1.4)
  #text(70, 35, names(cm$overall[2]), cex=1.5, font=2)
  #text(70, 20, round(as.numeric(cm$overall[2]), 3), cex=1.4)
}
```

```r
data=read.csv("TravelInsuranceData.csv",header=TRUE)
TravelInsuranceTest=read.csv("TravelInsuranceTest.csv",header=T)
```

```r
Insurance_data= data[,-1] # Removing very first column as it was not necessary in the data analysis.
TravelInsuranceTest=TravelInsuranceTest[,-1]
```

```r
Insurance_data$ChronicDiseases=as.factor(Insurance_data$ChronicDiseases)
Insurance_data$Employment.Type= as.factor(Insurance_data$Employment.Type)
Insurance_data$GraduateOrNot= as.factor(Insurance_data$GraduateOrNot)
Insurance_data$FrequentFlyer= as.factor(Insurance_data$FrequentFlyer)
Insurance_data$EverTravelledAbroad= as.factor(Insurance_data$EverTravelledAbroad)
Insurance_data$TravelInsurance= as.factor(Insurance_data$TravelInsurance)
```

```r
TravelInsuranceTest$ChronicDiseases=as.factor(TravelInsuranceTest$ChronicDiseases)
TravelInsuranceTest$Employment.Type= as.factor(TravelInsuranceTest$Employment.Type)
TravelInsuranceTest$GraduateOrNot= as.factor(TravelInsuranceTest$GraduateOrNot)
TravelInsuranceTest$FrequentFlyer= as.factor(TravelInsuranceTest$FrequentFlyer)
TravelInsuranceTest$EverTravelledAbroad= as.factor(TravelInsuranceTest$EverTravelledAbroad)
TravelInsuranceTest$TravelInsurance= as.factor(TravelInsuranceTest$TravelInsurance)
```

1. Before we create a model, do some data cleaning, feature selection and exploratory data analysis.

```r
unique(Insurance_data$ChronicDiseases)
```

```
## [1] 1 0
## Levels: 0 1
```

```r
unique(Insurance_data$Employment.Type)
```

```
## [1] Government Sector            Private Sector/Self Employed
## Levels: Government Sector Private Sector/Self Employed
```

```r
unique(Insurance_data$GraduateOrNot)
```

```
## [1] Yes No
## Levels: No Yes
```

```r
unique(Insurance_data$FrequentFlyer)
```

```
## [1] No  Yes
## Levels: No Yes
```

```r
unique(Insurance_data$EverTravelledAbroad)
```

```
## [1] No  Yes
## Levels: No Yes
```

```r
unique(Insurance_data$TravelInsurance)
```

```
## [1] 0 1
## Levels: 0 1
```

```r
unique(Insurance_data$AnnualIncome)
```

```
##  [1]  400000 1250000  500000  700000 1150000 1300000 1350000 1450000  800000
## [10] 1400000  850000 1500000 1050000  350000  600000  900000  550000  300000
## [19]  750000 1100000 1200000 1000000  950000 1700000 1750000  650000  450000
## [28] 1800000 1550000 1650000
```

```r
unique(Insurance_data$Age)
```

```
##  [1] 31 34 28 25 33 26 32 29 35 30 27
```

```r
unique(Insurance_data$FamilyMembers)
```

```
## [1] 6 7 4 3 8 9 5 2
```

```r
skim_without_charts(Insurance_data)
```

Table 1: Data summary

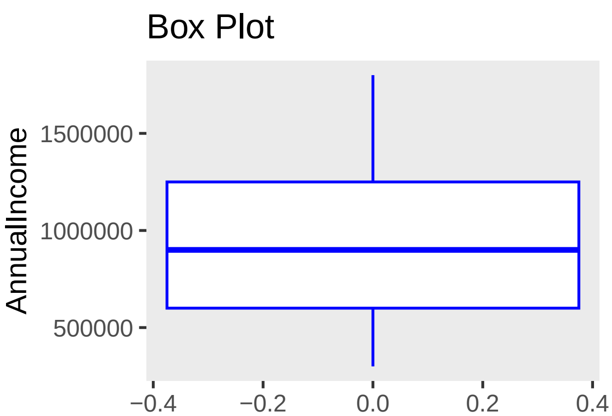
Name	Insurance_data
Number of rows	1887
Number of columns	9
Column type frequency:	
factor	6
numeric	3
Group variables	None

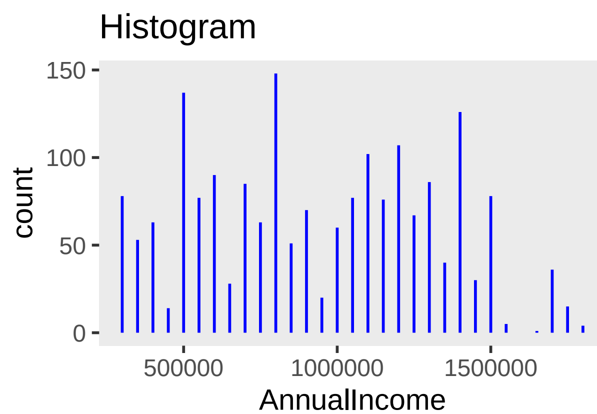
Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Employment.Type	0	1	FALSE	2	Pri: 1352, Gov: 535
GraduateOrNot	0	1	FALSE	2	Yes: 1605, No: 282
ChronicDiseases	0	1	FALSE	2	0: 1359, 1: 528
FrequentFlyer	0	1	FALSE	2	No: 1495, Yes: 392
EverTravelledAbroad	0	1	FALSE	2	No: 1520, Yes: 367
TravelInsurance	0	1	FALSE	2	0: 1206, 1: 681

Variable type: numeric

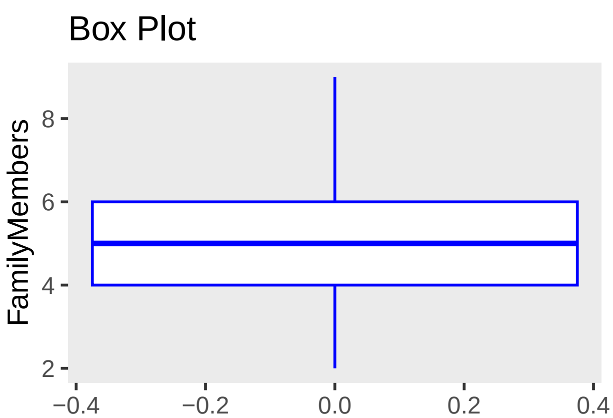
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Age	0	1	29.64	2.92	25	28	29	32	35
AnnualIncome	0	1	936062.53	376418.10	300000	600000	900000	1250000	1800000
FamilyMembers	0	1	4.75	1.62	2	4	5	6	9

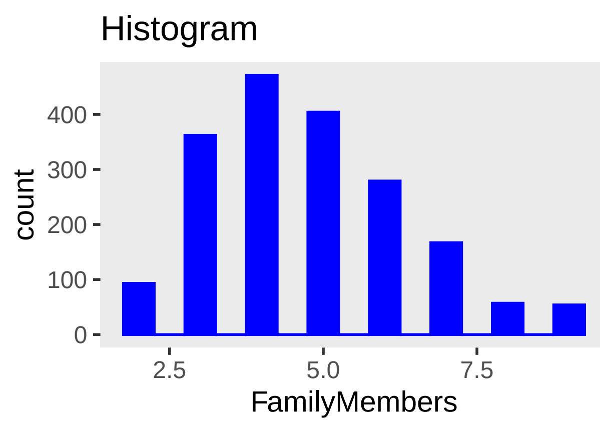




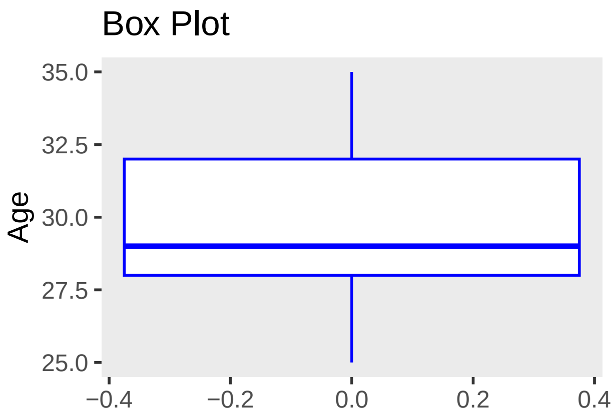
##Annual Income

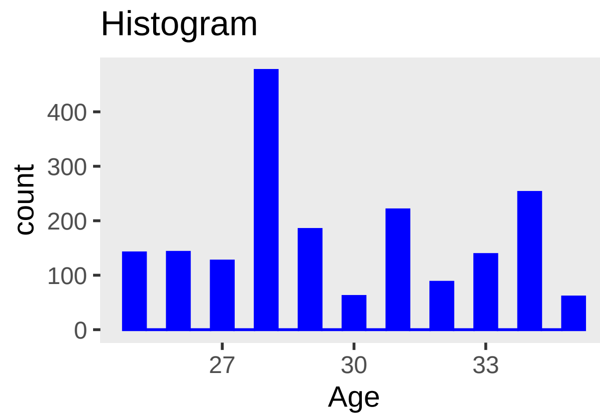
Family members





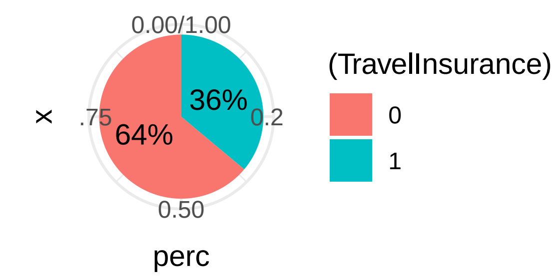
Age

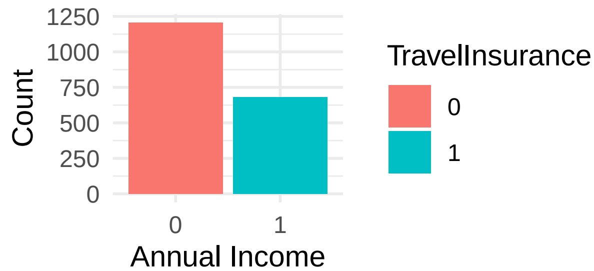


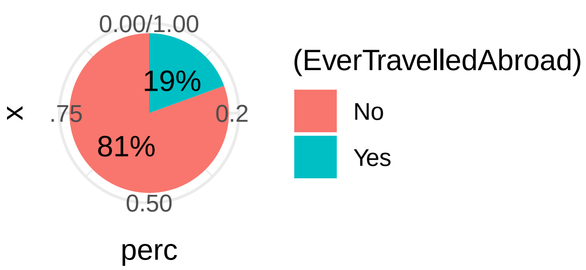


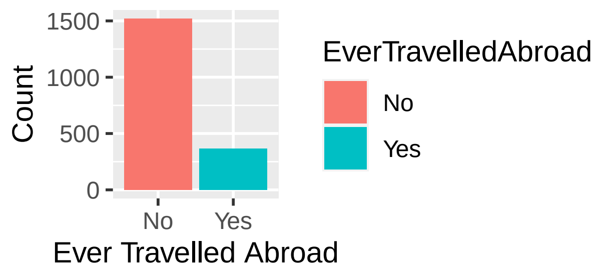
Categorical Variables

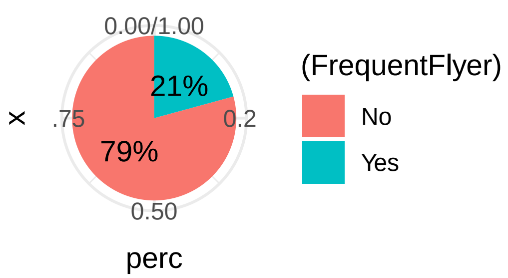
```
## [1] 0 1
## Levels: 0 1
```

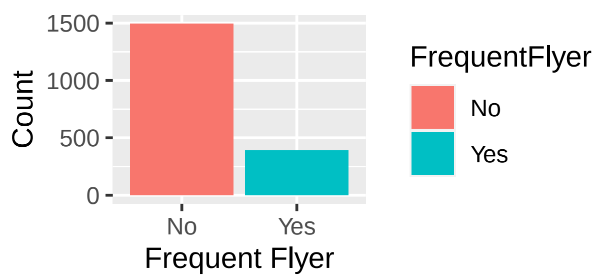








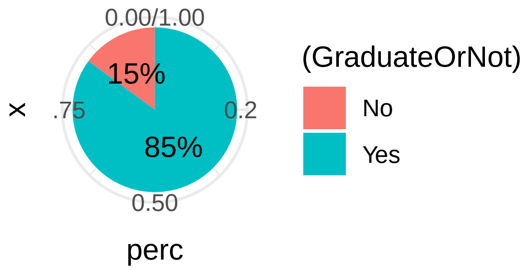


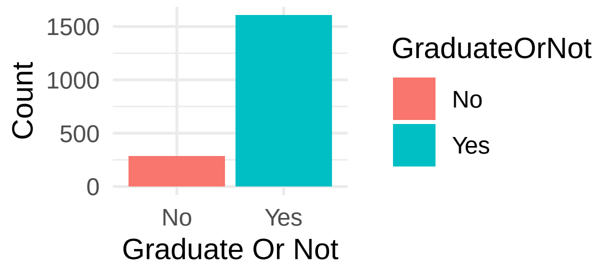


Categorical Variables 2

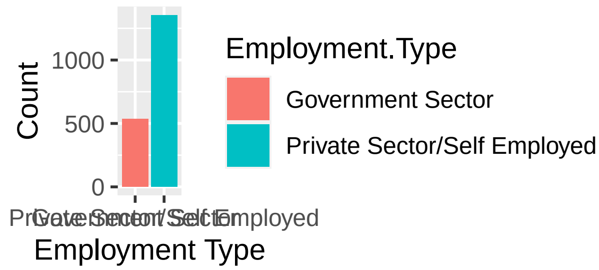
```
## Warning: Unknown or uninitialised column: `(TravelInsurance)`.
```

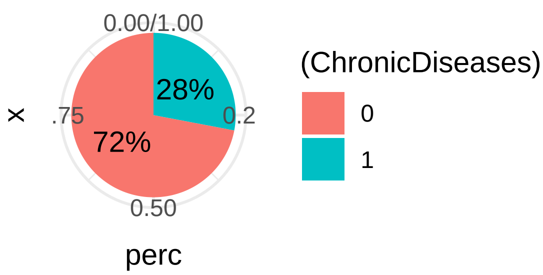
```
## NULL
```

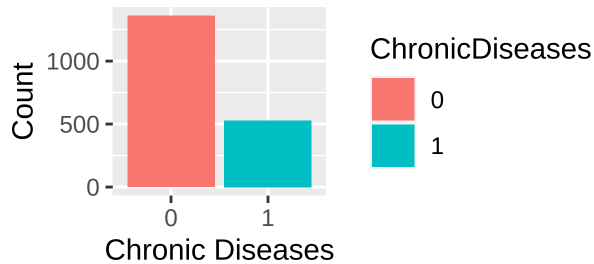

0.00/1.00
15%
85%
x
.75
0.2
0.50
perc
(GraduateOrNot)
No
Yes


1500
1000
500
0
Count
No
Yes
Graduate Or Not
GraduateOrNot
No
Yes

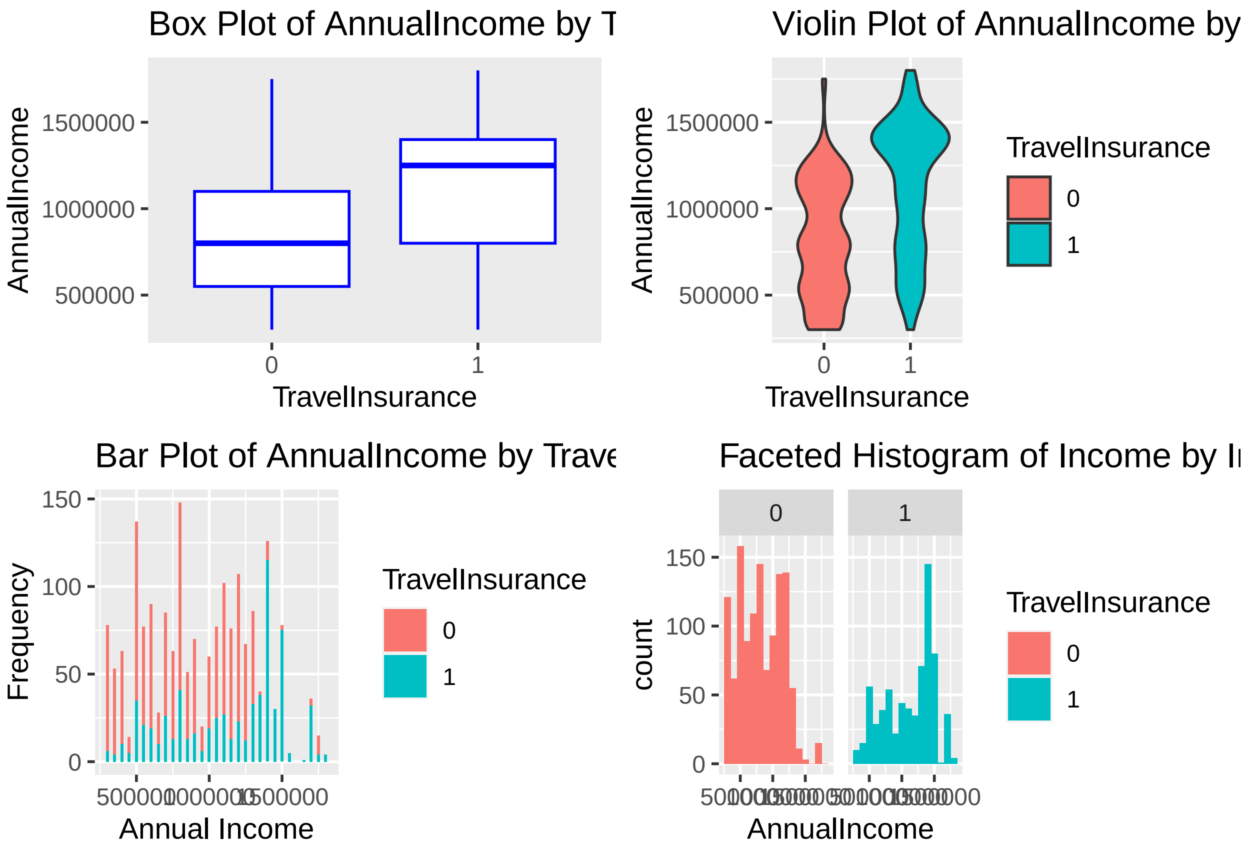

0.00/1.00
28%
72%
x
.75
0.2
0.50
perc
(Employment.Type)
Government Sector
Private Sector/Self Employed


1000
500
0
Count
Government Sector
Private Sector/Self Employed
Employment Type
Employment.Type
Government Sector
Private Sector/Self Employed

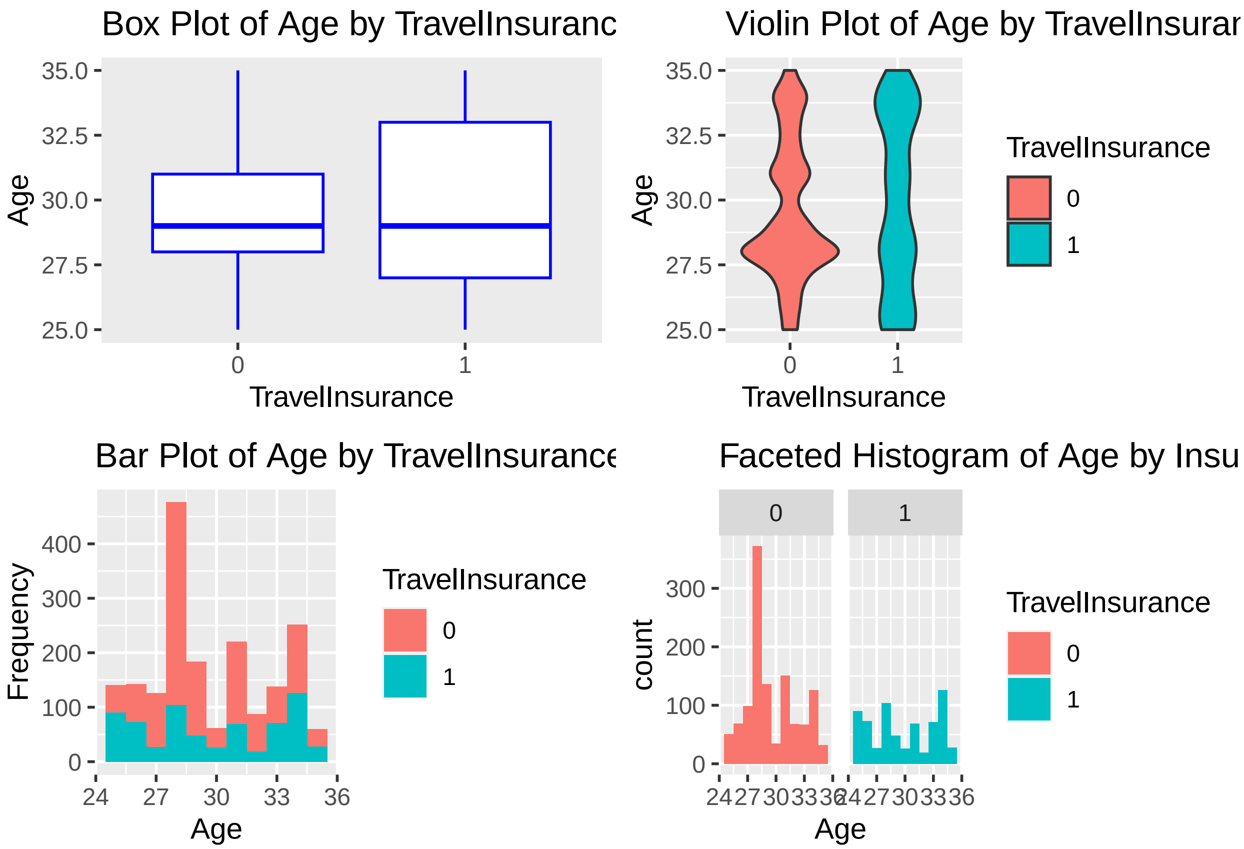

0.00/1.00
28%
72%
x
.75
0.2
0.50
perc
(ChronicDiseases)
0
1


1000
500
0
Count
0
1
Chronic Diseases
ChronicDiseases
0
1

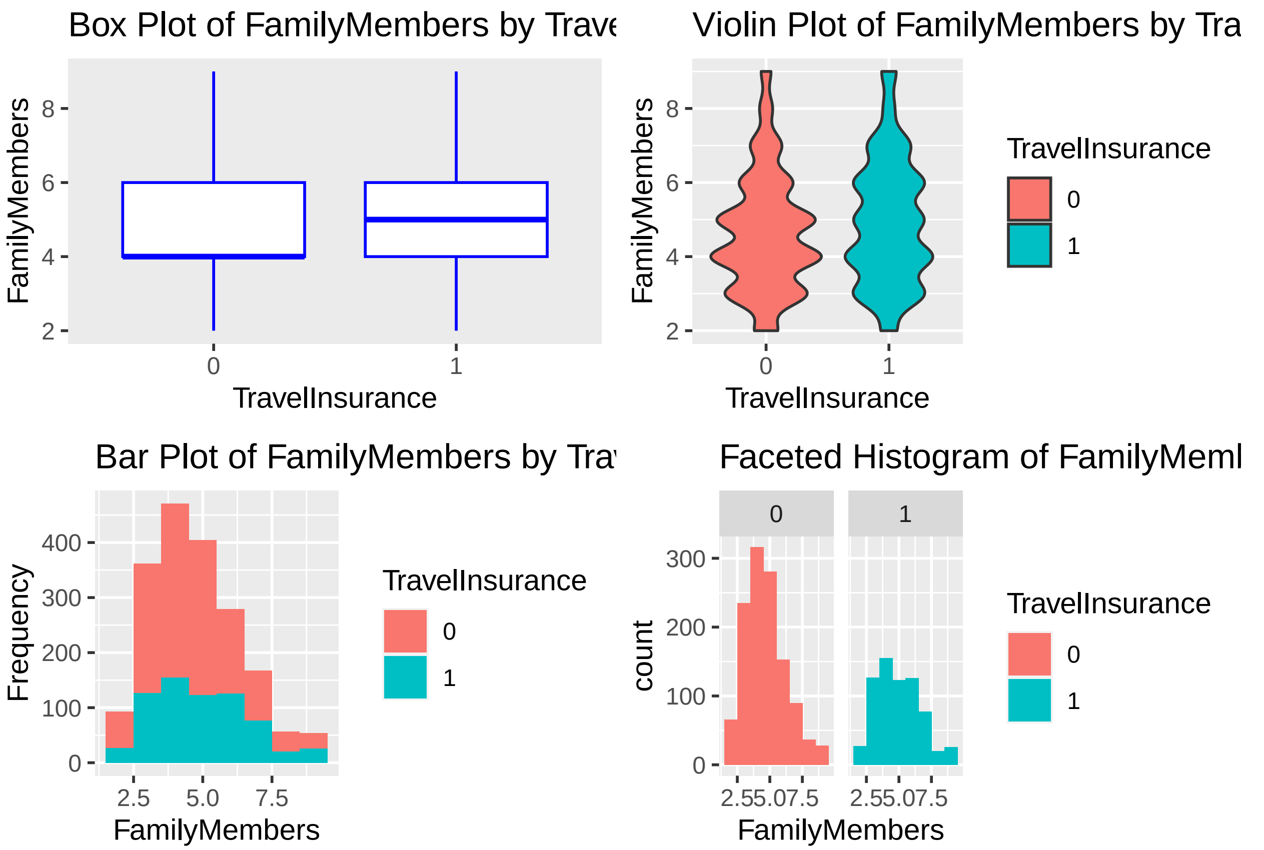
plot of AnnualIncome by Insurance



plot of Age by Insurance



plot of FamilyMembers by Insurance




```
#ggpairs(data = Insurance_data %>% select(TravelInsurance,Age, AnnualIncome,FamilyMembers))
```

2. Come up with a set of candidate methods that is suitable for the data.
3. Fit the models with training data.
4. Reduce the dimension of features by performing feature selection or dimension reduction.
5. Adjust the tuning parameters using cross-validation or model performance criteria such as error rate, AUC, etc.
6. Check the adequacy of the model fits and possibly revise the model.
7. Compare the models and choose your final model base on the prediction accuracy on the test data.

Logestic regression

2) Logistic Model

use all variables as predictor:

```
set.seed(100)

# Step 1: Split the data into training and testing sets
sample_index= sample(1:nrow(Insurance_data), 0.8 * nrow(Insurance_data))
train_data=Insurance_data[sample_index, ]
test_data=Insurance_data[-sample_index, ]
```

```r
# Step 2: Train the logistic regression model
model= glm(TravelInsurance ~ Age + Employment.Type + GraduateOrNot + AnnualIncome + FamilyMembers +Chron
              family = binomial, data = train_data)
summary(model)
```

```
##
## Call:
## glm(formula = TravelInsurance ~ Age + Employment.Type + GraduateOrNot +
##     AnnualIncome + FamilyMembers + ChronicDiseases + FrequentFlyer +
##     EverTravelledAbroad, family = binomial, data = train_data)
##
## Coefficients:
##                                               Estimate Std. Error z value
## (Intercept)                                 -4.996e+00  7.137e-01  -7.001
## Age                                          6.275e-02  2.093e-02   2.997
## Employment.TypePrivate Sector/Self Employed  2.155e-01  1.493e-01   1.444
## GraduateOrNotYes                            -2.093e-01  1.739e-01  -1.203
## AnnualIncome                                 1.506e-06  1.965e-07   7.660
## FamilyMembers                                1.403e-01  3.773e-02   3.719
## ChronicDiseases1                             4.122e-02  1.366e-01   0.302
## FrequentFlyerYes                             4.415e-01  1.547e-01   2.853
## EverTravelledAbroadYes                       1.538e+00  1.720e-01   8.942
##                                             Pr(>|z|)
## (Intercept)                                 2.55e-12 ***
## Age                                          0.00272 **
## Employment.TypePrivate Sector/Self Employed  0.14880
## GraduateOrNotYes                             0.22881
## AnnualIncome                                1.85e-14 ***
## FamilyMembers                                0.00020 ***
## ChronicDiseases1                             0.76276
## FrequentFlyerYes                             0.00433 **
## EverTravelledAbroadYes                       < 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: 1955.0  on 1508  degrees of freedom
## Residual deviance: 1603.1  on 1500  degrees of freedom
## AIC: 1621.1
##
## Number of Fisher Scoring iterations: 4
```

observation

-looking at the logistic model when trained on the training data, still statically significant variables and statically insignificant variables are the same.

Observation -Here Age, AnnualIncome, FamilyMembers, FrequentFlyerYes,EverTravelledAbroadYes are statically significant in determining weather the customer will purchase a travel insurance or not.

-likewise ChronicDiseases1, GraduateOrNotYes, Employment.TypePrivate Sector/Self Employed are not statically significant indicating that they are not important for customer in purchasing the travel insurance.

-for every one unit change in customers age, the log odd of purchasing travel insurance is increased by 7.29e-02 units

-for every one unit change in customers Annual income, the log odd of purchasing travel insurance is increased by 1.56e-06 units

-for every one unit change in customers number of family members in the family, the log odd of purchasing travel insurance is increased by 1.44e-01 units

```
set.seed(100)
# Step 3: Make predictions on the testing set
predictions_glm= predict(model, newdata = test_data, type = "response")
```

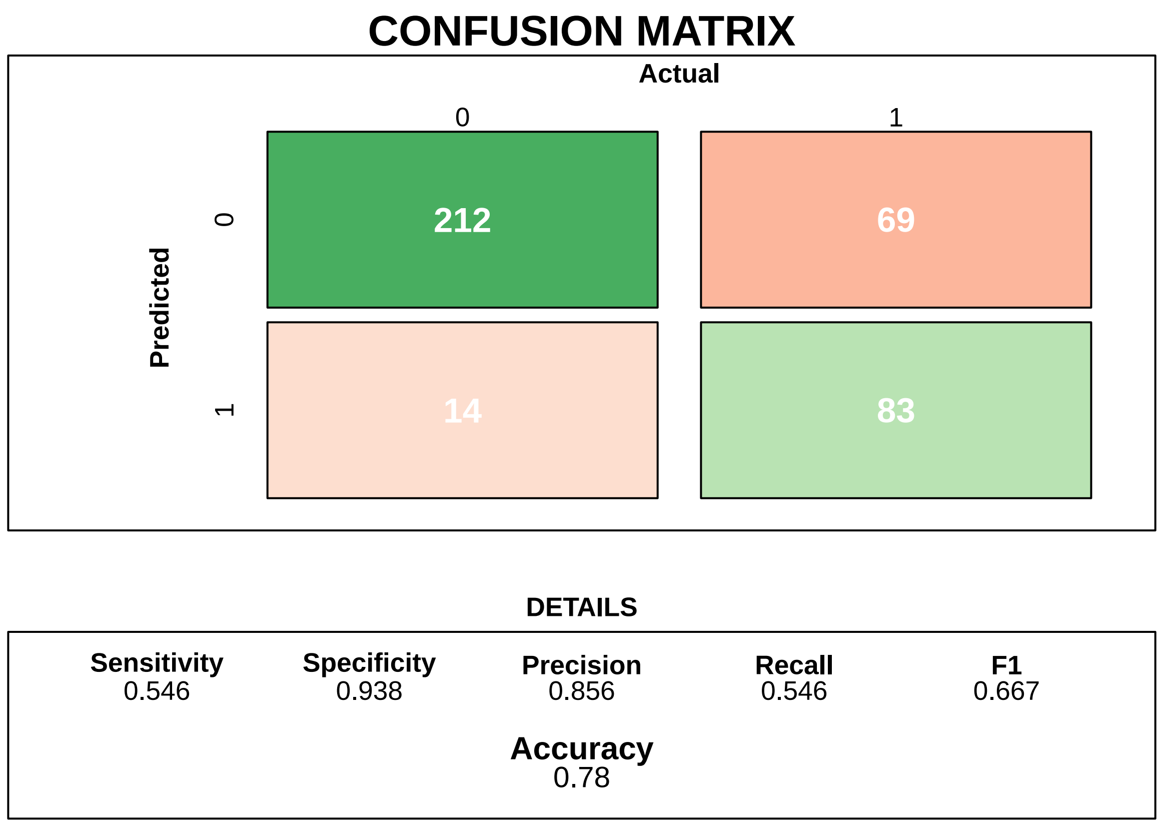
```
set.seed(100)

predicted_labels= ifelse(predictions_glm > 0.5, 1, 0)
```

let's see the confusion matrix

```
#########################
pred=as.factor(predicted_labels)
cm_glm = confusionMatrix(pred,test_data$TravelInsurance,positive = "1")
cm_glm$positive
```

```
## [1] "1"
```

```
draw_confusion_matrix(cm_glm)
```




```
#################################
table(predicted_labels,test_data$TravelInsurance)
```

```
##
## predicted_labels   0   1
```

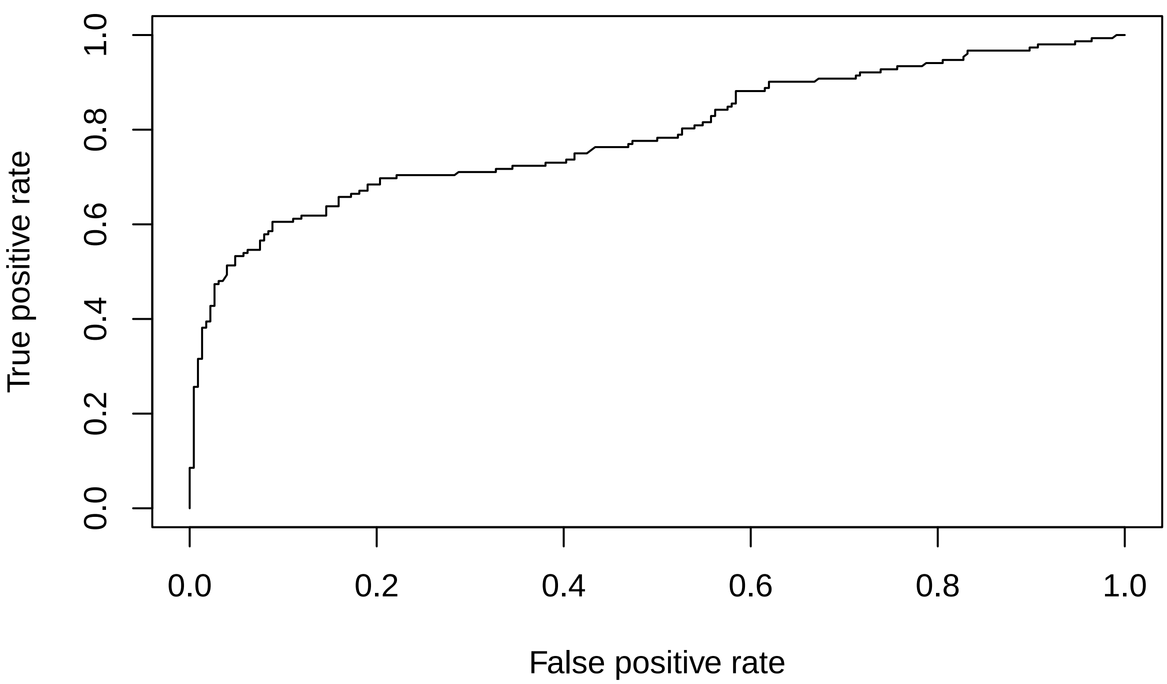
```
##                   0 212   69
##                   1  14   83
```

```
#(accuracy_glm=(cm_glm[1,1]+cm_glm[2,2])/(cm_glm[1,1]+cm_glm[1,2]+cm_glm[2,1]+cm_glm[2,2]))
#(recall= (cm_glm[2,2])/(cm_glm[2,2]+cm_glm[2,1]))
#(precision=(cm_glm[2,2])/(cm_glm[2,2]+cm_glm[1,2]))
```

Therefor the test accuracy of the logistic model is $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ (224+72)/378= 0.7848325 i.e 78.30%

$Precision = \frac{TP}{TP+TN} = \frac{72}{72+68}$ $Recall = \frac{TP}{TP+FN} = \frac{72}{72+14}$

```
########## ROC curve #################

rocplot <- function(pred, truth) {
  predob <- prediction(pred, truth)
  perf <- performance(predob, "tpr", "fpr")
  plot(perf)
}

# Make predictions on the testing set
#predictions.fselected <- predict(model.fselct, newdata = test_data, type = "response")
predictions.fselect= predict(model, newdata = test_data, type = "response")
# Extract the predicted probabilities
fitted <- as.numeric(predictions.fselect)

# Display the ROC plot
rocplot(fitted, test_data$TravelInsurance)
```



Logestic regression:

Feature selection

-Here Age, AnnualIncome, FamilyMembers, FrequentFlyerYes,EverTravelledAbroadYes are statically significant in determining weather the customer will purchase a travel insurance or not.

```r
set.seed(100)
# Step 2: Train the logistic regression model
model.fselect= glm(TravelInsurance ~ Age  + AnnualIncome + FamilyMembers + FrequentFlyer + EverTravelle
summary(model.fselect)
```

```
##
## Call:
## glm(formula = TravelInsurance ~ Age + AnnualIncome + FamilyMembers +
##     FrequentFlyer + EverTravelledAbroad, family = binomial, data = train_data)
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -4.890e+00  6.854e-01  -7.135 9.69e-13 ***
## Age                     5.764e-02  2.074e-02   2.779 0.005455 **
## AnnualIncome            1.542e-06  1.893e-07   8.143 3.86e-16 ***
## FamilyMembers           1.401e-01  3.765e-02   3.720 0.000199 ***
## FrequentFlyerYes        4.606e-01  1.538e-01   2.995 0.002744 **
## EverTravelledAbroadYes  1.540e+00  1.712e-01   8.995  < 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: 1955.0  on 1508  degrees of freedom
## Residual deviance: 1607.4  on 1503  degrees of freedom
## AIC: 1619.4
##
## Number of Fisher Scoring iterations: 4
```

```r
set.seed(100)
# Step 3: Make predictions on the testing set
predictions.fselect= predict(model.fselect, newdata = test_data, type = "response")
predictions.fselect[1:10] #let's look at the first 10 predictions by the logistic model on the test dat
```

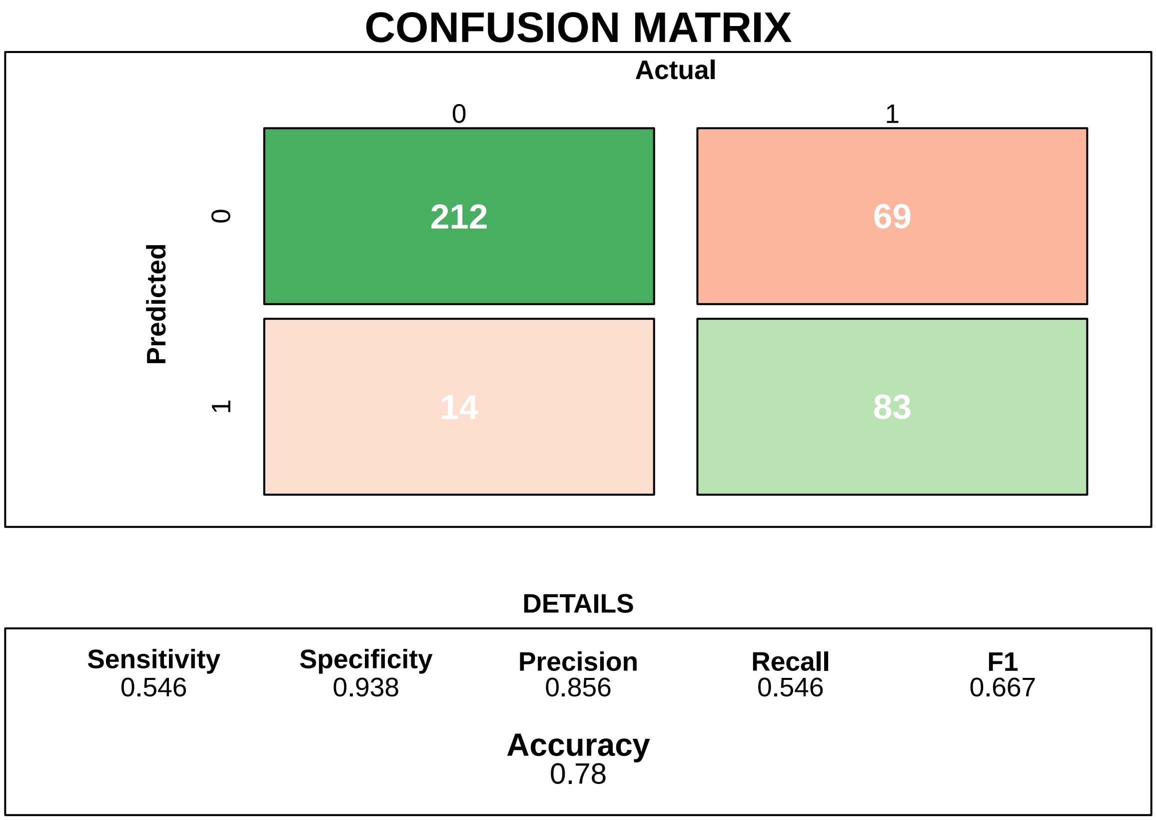
```
##         8         9        10        27        39        40        41        45
## 0.8019987 0.8581790 0.2944717 0.2149270 0.7688422 0.4000402 0.7737694 0.1266537
##        46        49
## 0.7187463 0.7158887
```

```r
set.seed(100)
# let's give the predicted model a good name of labels
# Convert predicted probabilities to binary predictions (0 or 1)
predicted_labels.fselect= ifelse(predictions.fselect > 0.5, 1, 0)
predicted_labels.fselect[1:10]  # looking at the predictive level of first 10 observation by logistic r
```

```
##  8  9 10 27 39 40 41 45 46 49
##  1  1  0  0  1  0  1  0  1  1
```

```r
#########################
pred=as.factor(predicted_labels.fselect)
```

```r
cm_glm.fselect = confusionMatrix(pred,test_data$TravelInsurance,positive = "1")

draw_confusion_matrix(cm_glm.fselect)
```




```r
##################################
set.seed(100)
cm_glm.fselect=table(predicted_labels.fselect,test_data$TravelInsurance)
(accuracy_glm.fselect=(cm_glm.fselect[1,1]+cm_glm.fselect[2,2])/(cm_glm.fselect[1,1]+cm_glm.fselect[1,2
```

```
## [1] 0.7804233
```

```r
(recall_glm.fselect= (cm_glm.fselect[2,2])/(cm_glm.fselect[2,2]+cm_glm.fselect[2,1]))
```

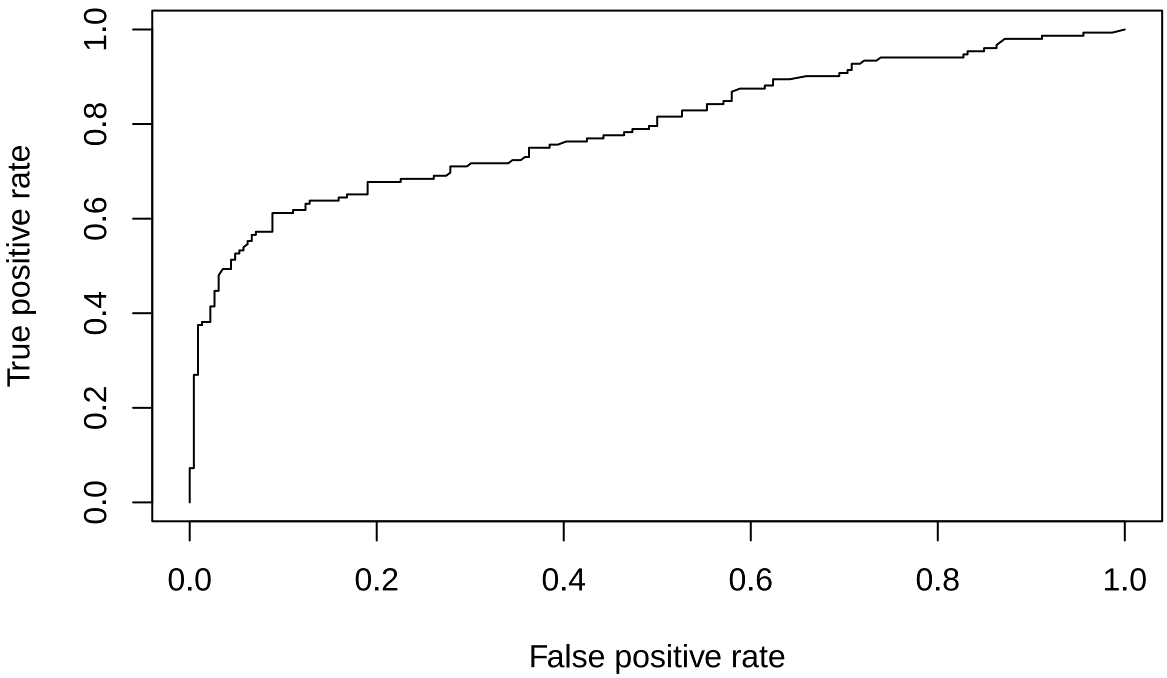
```
## [1] 0.8556701
```

```r
(precision_glm.fselect=(cm_glm.fselect[2,2])/(cm_glm.fselect[2,2]+cm_glm.fselect[1,2]))
```

```
## [1] 0.5460526
```

```r
########## ROC curve #################

rocplot <- function(pred, truth) {
  predob <- prediction(pred, truth)
  perf <- performance(predob, "tpr", "fpr")
  plot(perf)
}

# Make predictions on the testing set
#predictions.fselected <- predict(model.fselct, newdata = test_data, type = "response")
predictions.fselect= predict(model.fselect, newdata = test_data, type = "response")
```

```r
# Extract the predicted probabilities
fitted <- as.numeric(predictions.fselect)

# Display the ROC plot
rocplot(fitted, test_data$TravelInsurance)
```



Logistic regression

cross validation

```r
set.seed(100)
logistic_regression_caret_model = train(
  form = TravelInsurance ~ Age  + AnnualIncome + FamilyMembers + FrequentFlyer + EverTravelledAbroad,
  #tuneLenght=10,
  data = train_data,
  trControl = trainControl(method = "cv", number = 10),
  method = "glm",
  family = "binomial"
)
summary(logistic_regression_caret_model)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -4.890e+00  6.854e-01  -7.135 9.69e-13 ***
## Age                     5.764e-02  2.074e-02   2.779 0.005455 **
## AnnualIncome            1.542e-06  1.893e-07   8.143 3.86e-16 ***
## FamilyMembers           1.401e-01  3.765e-02   3.720 0.000199 ***
```

```
## FrequentFlyerYes        4.606e-01  1.538e-01   2.995 0.002744 **
## EverTravelledAbroadYes  1.540e+00  1.712e-01   8.995  < 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: 1955.0  on 1508  degrees of freedom
## Residual deviance: 1607.4  on 1503  degrees of freedom
## AIC: 1619.4
##
## Number of Fisher Scoring iterations: 4
```

Linear Discriminant Analysis

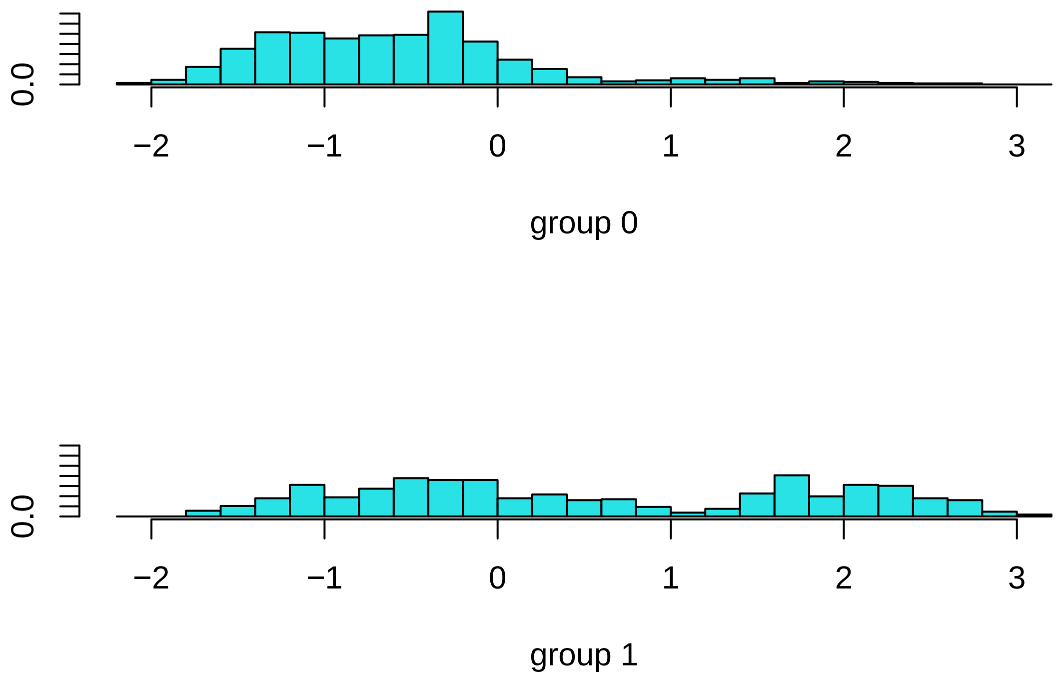
```
set.seed(100)
#install.packages("MASS")
library(MASS)

lda.out=lda(TravelInsurance ~ Age + Employment.Type + GraduateOrNot + AnnualIncome + FamilyMembers + Chr
lda.out
```

```
## Call:
## lda(TravelInsurance ~ Age + Employment.Type + GraduateOrNot +
##     AnnualIncome + FamilyMembers + ChronicDiseases + FrequentFlyer +
##     EverTravelledAbroad, data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.6494367 0.3505633
##
## Group means:
##        Age Employment.TypePrivate Sector/Self Employed GraduateOrNotYes
## 0 29.54388                                    0.655102        0.8428571
## 1 29.80340                                    0.805293        0.8506616
##   AnnualIncome FamilyMembers ChronicDiseases1 FrequentFlyerYes
## 0     820459.2      4.662245        0.2836735        0.1408163
## 1    1120604.9      4.918715        0.2911153        0.3327032
##   EverTravelledAbroadYes
## 0             0.06938776
## 1             0.39508507
##
## Coefficients of linear discriminants:
##                                                      LD1
## Age                                         5.066645e-02
## Employment.TypePrivate Sector/Self Employed  1.521348e-01
## GraduateOrNotYes                           -2.087685e-01
## AnnualIncome                                1.407870e-06
## FamilyMembers                               1.252036e-01
## ChronicDiseases1                            2.917388e-02
## FrequentFlyerYes                            4.179604e-01
## EverTravelledAbroadYes                      1.717812e+00
```

Observation

-The LDA output indicates that 64.09% of the training observation corresponds to customer not taking the travel insurance and 35.90% of the training observation corresponds to the customer taking the travel insurance

```
plot(lda.out)
```




```
set.seed(100)
lda.pred <- predict(lda.out , test_data)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

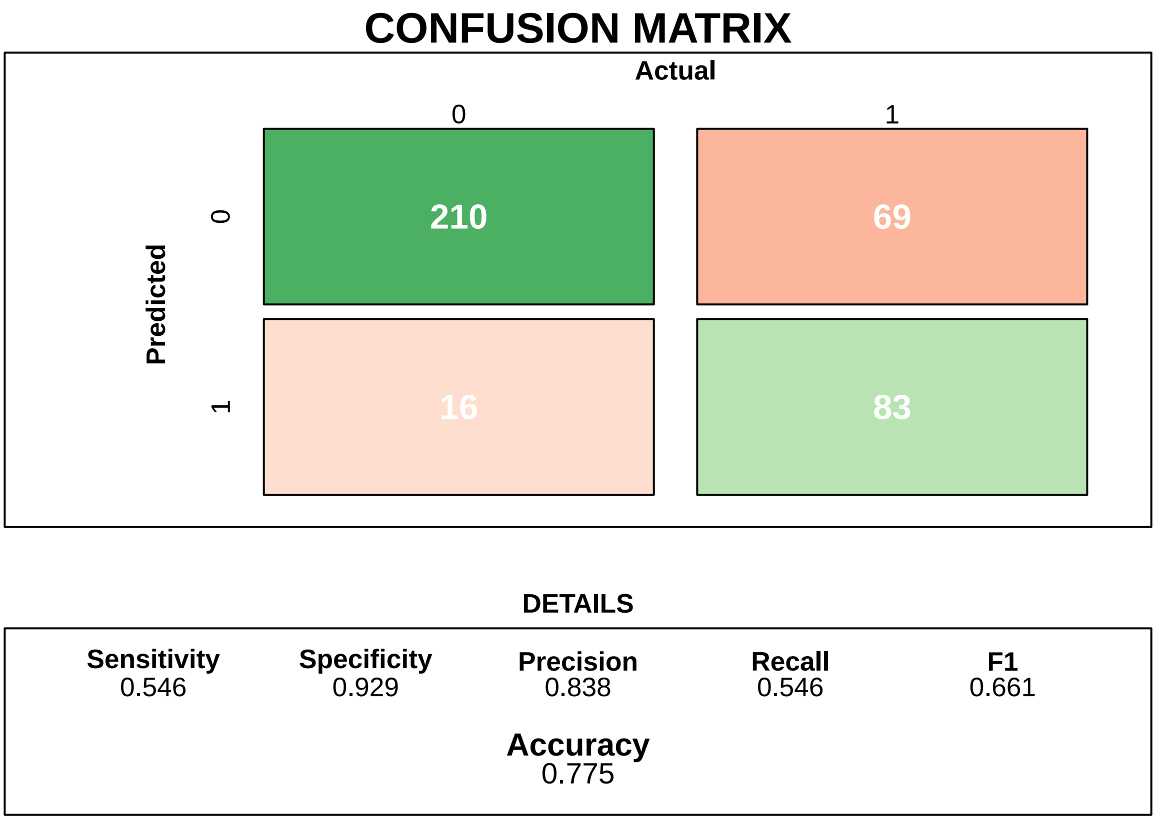
```
lda.pred$class[1:10] # What LDA predict for first 10 observation
```

```
##  [1] 1 1 0 0 1 0 1 0 1 1
## Levels: 0 1
```

```
lda.class <- lda.pred$class
table(lda.class, test_data$TravelInsurance)
```

```
##
## lda.class   0   1
##         0 210  69
##         1  16  83
```

```
#########################
cm_lda = confusionMatrix(lda.class,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_lda)
```




```
#################################
```

For the LDA
$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ (210+83)/378= 0.7751 i.e 77.51%

$Precision = \frac{TP}{TP+TN} = \frac{83}{83+16}$ =0.8384 $Recall = \frac{TP}{TP+FN} = \frac{83}{83+69}$ =0.5355

feature selected

```
lda.select=lda(TravelInsurance ~ Age  + AnnualIncome + FamilyMembers + FrequentFlyer + EverTravelledAbr
lda.out
```

```
## Call:
## lda(TravelInsurance ~ Age + Employment.Type + GraduateOrNot +
##     AnnualIncome + FamilyMembers + ChronicDiseases + FrequentFlyer +
##     EverTravelledAbroad, data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.6494367 0.3505633
##
## Group means:
##        Age Employment.TypePrivate Sector/Self Employed GraduateOrNotYes
## 0 29.54388                                    0.655102        0.8428571
## 1 29.80340                                    0.805293        0.8506616
##   AnnualIncome FamilyMembers ChronicDiseases1 FrequentFlyerYes
## 0     820459.2      4.662245        0.2836735        0.1408163
## 1    1120604.9      4.918715        0.2911153        0.3327032
##   EverTravelledAbroadYes
```

```
## 0            0.06938776
## 1            0.39508507
##
## Coefficients of linear discriminants:
##                                                  LD1
## Age                                     5.066645e-02
## Employment.TypePrivate Sector/Self Employed  1.521348e-01
## GraduateOrNotYes                       -2.087685e-01
## AnnualIncome                            1.407870e-06
## FamilyMembers                           1.252036e-01
## ChronicDiseases1                        2.917388e-02
## FrequentFlyerYes                        4.179604e-01
## EverTravelledAbroadYes                  1.717812e+00
```

```
lda.pred.select <- predict(lda.select , test_data)
names(lda.pred.select)
```

```
## [1] "class"     "posterior" "x"
```

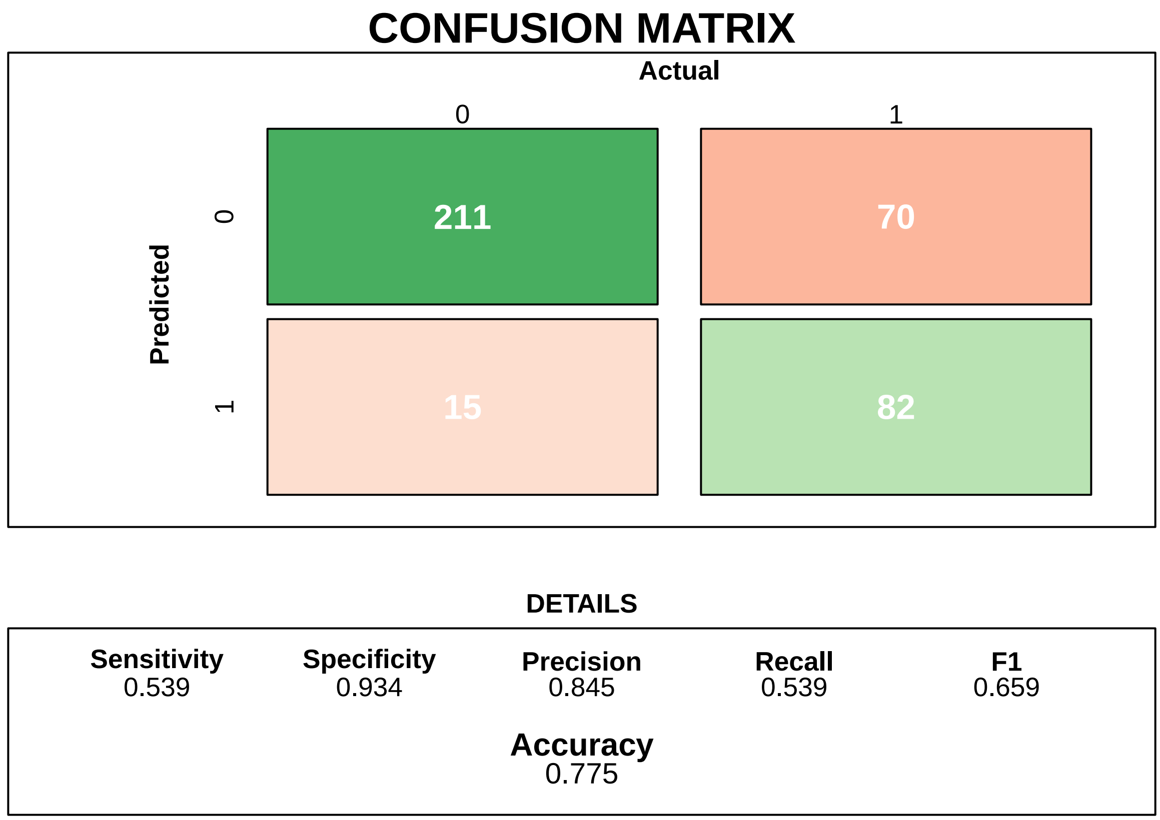
```
lda.pred.select$class[1:10] # What LDA predict for first 10 observation
```

```
##  [1] 1 1 0 0 1 0 1 0 1 1
## Levels: 0 1
```

```
lda.class.select <- lda.pred.select$class
table(lda.class.select, test_data$TravelInsurance)
```

```
##
## lda.class.select   0   1
##                0 211  70
##                1  15  82
```

```
#########################
cm_lda.select = confusionMatrix(lda.class.select,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_lda.select)
```




```
##################################
```

For the LDA

$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ (210+83)/378= 0.7751 i.e 77.51%

$Precision = \frac{TP}{TP+TN} = \frac{82}{82+70}$ =0.5395 $Recall = \frac{TP}{TP+FN} = \frac{82}{82+15}$ =0.8454

4) Quadratic Discriminant Analysis

```
qda.out=qda(TravelInsurance ~ Age + Employment.Type + GraduateOrNot + AnnualIncome + FamilyMembers + Ch
qda.out
```

```
## Call:
## qda(TravelInsurance ~ Age + Employment.Type + GraduateOrNot +
##     AnnualIncome + FamilyMembers + ChronicDiseases + FrequentFlyer +
##     EverTravelledAbroad, data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.6494367 0.3505633
##
## Group means:
##        Age Employment.TypePrivate Sector/Self Employed GraduateOrNotYes
## 0 29.54388                                    0.655102        0.8428571
## 1 29.80340                                    0.805293        0.8506616
##   AnnualIncome FamilyMembers ChronicDiseases1 FrequentFlyerYes
## 0     820459.2      4.662245        0.2836735        0.1408163
## 1    1120604.9      4.918715        0.2911153        0.3327032
##   EverTravelledAbroadYes
## 0             0.06938776
```

```
## 1                0.39508507
```

Observation -The QDA output indicates that 64.09% of the training observation corresponds to customer not taking the travel insurance and 35.90% of the training observation corresponds to the customer taking the travel insurance

```r
qda.pred <- predict(qda.out , test_data)
names(qda.pred)
```

```
## [1] "class"     "posterior"
```

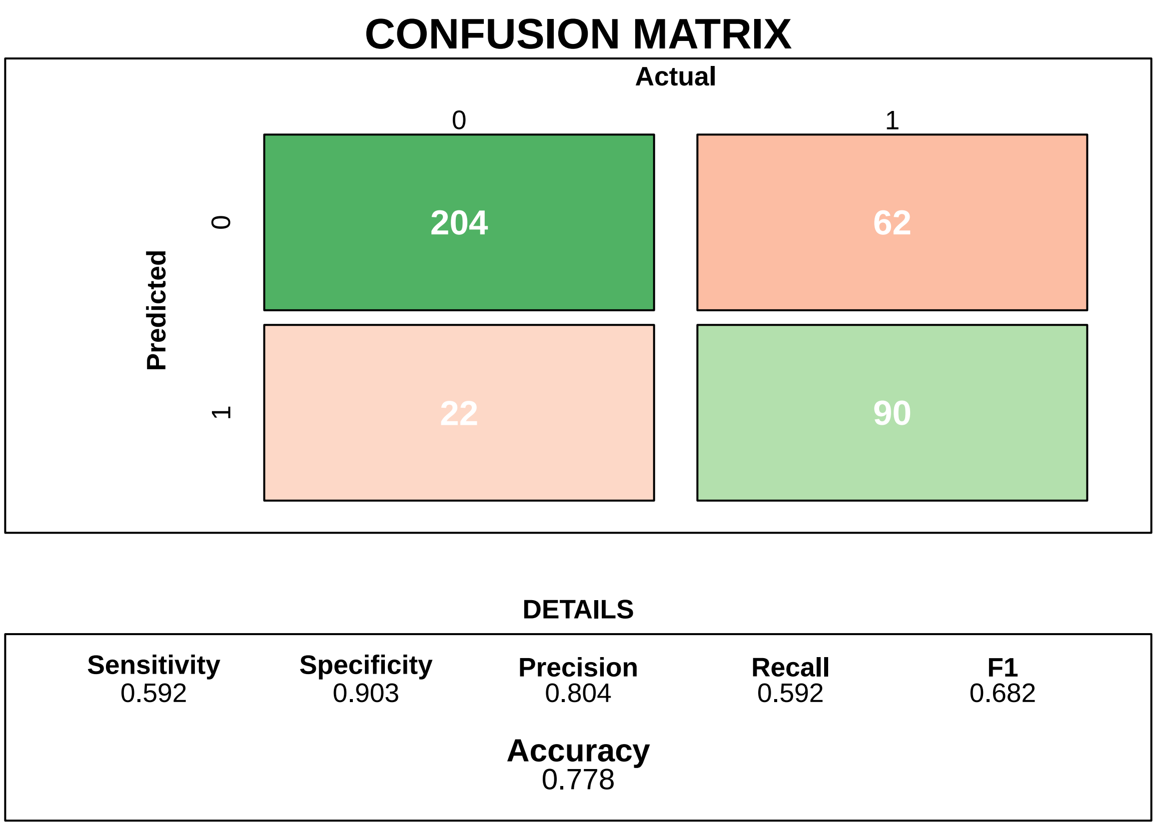
```r
qda.pred$class[1:10] # What QDA predict for first 10 observation
```

```
##  [1] 1 1 0 0 1 0 1 0 1 1
## Levels: 0 1
```

```r
set.seed(100)
qda.class <- qda.pred$class
table(qda.class, test_data$TravelInsurance)
```

```
##
## qda.class   0   1
##         0 204  62
##         1  22  90
```

```r
#########################

cm_qda = confusionMatrix(qda.class,test_data$TravelInsurance,positive = "1")

draw_confusion_matrix(cm_qda )
```



```r
##################################

(204+90)/378
```

```
## [1] 0.7777778
```

```r
90/(90+62)
```

```
## [1] 0.5921053
```

```r
90/(90+22)
```

```
## [1] 0.8035714
```

$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ (204+90)/378= 0.7778 i.e 77.78%

$Precision = \frac{TP}{TP+TN} = \frac{90}{90+62}$ =0.5921 $Recall = \frac{TP}{TP+FN} = \frac{90}{90+22}$ =0.8036

feature selection

```r
qda.select=qda(TravelInsurance ~ Age  + AnnualIncome + FamilyMembers + FrequentFlyer + EverTravelledAbro
qda.select
```

```
## Call:
## qda(TravelInsurance ~ Age + AnnualIncome + FamilyMembers + FrequentFlyer +
##     EverTravelledAbroad, data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.6494367 0.3505633
##
## Group means:
##        Age AnnualIncome FamilyMembers FrequentFlyerYes EverTravelledAbroadYes
## 0 29.54388     820459.2      4.662245        0.1408163             0.06938776
## 1 29.80340    1120604.9      4.918715        0.3327032             0.39508507
```

```r
qda.pred.select <- predict(qda.select , test_data)
names(qda.pred.select)
```

```
## [1] "class"     "posterior"
```

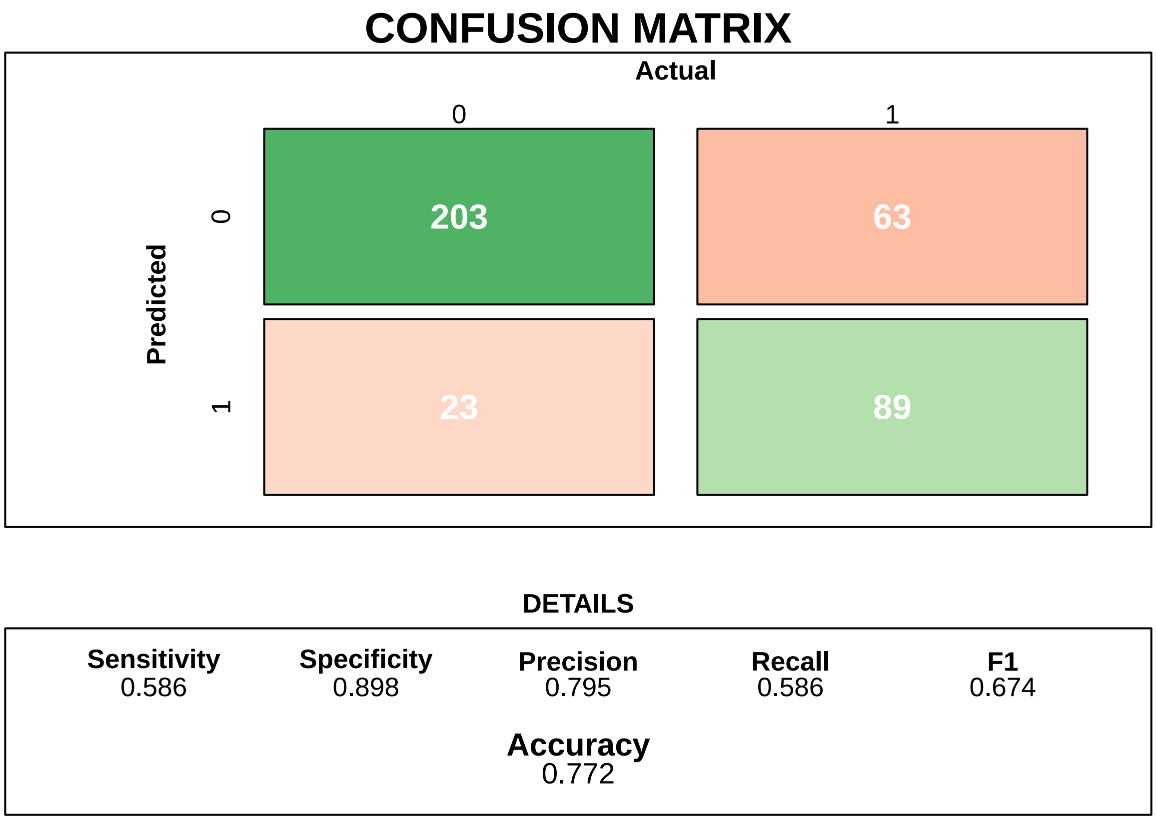
```r
qda.pred.select$class[1:10] # What QDA predict for first 10 observation
```

```
##  [1] 1 1 0 0 1 1 1 0 1 1
## Levels: 0 1
```

```r
set.seed(100)
qda.class.select <- qda.pred.select$class
table(qda.class.select, test_data$TravelInsurance)
```

```
##
## qda.class.select   0   1
##                0 203  63
##                1  23  89
```

```r
 #########################
cm_qda.select = confusionMatrix(qda.class.select,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_qda.select)
```




```r
#################################
(203+89)/378
```

```
## [1] 0.7724868
```

```r
89/(89+63)
```

```
## [1] 0.5855263
```

```r
89/(89+23)
```

```
## [1] 0.7946429
```

$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ (203+89)/378 =0.7725 i.e 77.25%

$Precision = \frac{TP}{TP+TN} = \frac{89}{89+63}$ =0.5855 $Recall = \frac{TP}{TP+FN} = \frac{89}{89+23} = 0.7946$

KNN ## 1) KNN- classifier

```r
set.seed(100)
# cagtegorical variable as factor
Insurance_data$ChronicDiseases=as.factor(Insurance_data$ChronicDiseases)
Insurance_data$Employment.Type= as.factor(Insurance_data$Employment.Type)
Insurance_data$GraduateOrNot= as.factor(Insurance_data$GraduateOrNot)
Insurance_data$FrequentFlyer= as.factor(Insurance_data$FrequentFlyer)
Insurance_data$EverTravelledAbroad= as.factor(Insurance_data$EverTravelledAbroad)
Insurance_data$TravelInsurance= as.factor(Insurance_data$TravelInsurance)
```

```r
# converting all my dataset to numeric for the model setting
Insurance_data_num <- as.data.frame(lapply(Insurance_data[,1:8], as.numeric))
```

```
set.seed(100)
knn_fit = train(
  TravelInsurance ~ .,
  data = train_data,
  method = "knn",
  tuneLength=10,
  trControl = trainControl(method = "cv", number = 10),
  preProcess = c("center", "scale")
)

knn_fit
```

```
## k-Nearest Neighbors
##
## 1509 samples
##    8 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1358, 1358, 1358, 1358, 1358, 1358, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7621148  0.4528058
##    7  0.7720618  0.4654331
##    9  0.7713687  0.4556991
##   11  0.7753687  0.4626138
##   13  0.7786799  0.4676690
##   15  0.7846402  0.4800932
##   17  0.7866402  0.4842670
##   19  0.7839912  0.4807701
##   21  0.7813377  0.4725415
##   23  0.7853157  0.4792685
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 17.
```

```
KNN_pred=predict(knn_fit , test_data)
cm=table(KNN_pred,test_data$TravelInsurance)
(212+94)/378
```

```
## [1] 0.8095238
```

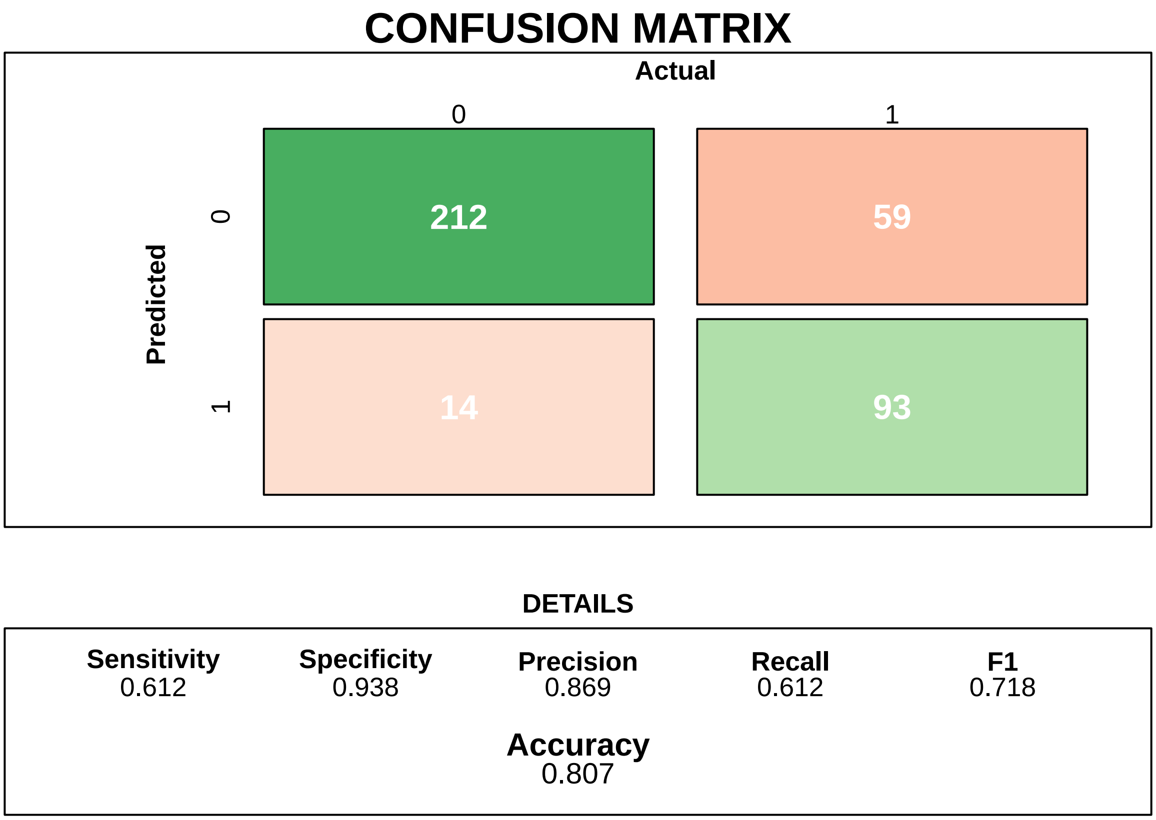
```
94/(94+58)
```

```
## [1] 0.6184211
```

```
94/(94+14)
```

```
## [1] 0.8703704
```

```
#########################
cm_KNN = confusionMatrix(KNN_pred,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_KNN)
```




```
##################################
```

$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ (169+39)/378= 0.8095 i.e 80.95%

$Precision = \frac{TP}{TP+TN} = \frac{94}{94+58}$ =0.6184 $Recall = \frac{TP}{TP+FN} = \frac{94}{94+14}$ =0.8704

using the cross validation for finding the best number of class, k=17

DT and Prued DT

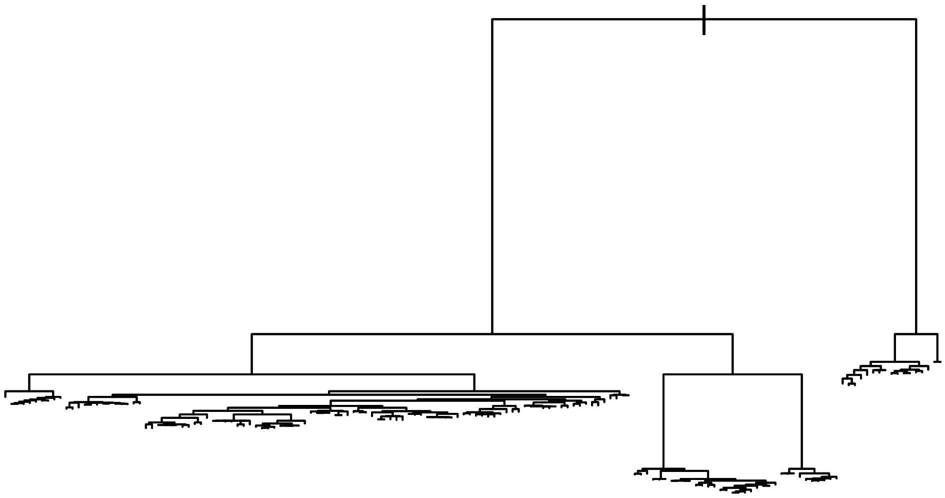
```
tree.d=tree(TravelInsurance~., data=Insurance_data,split="gini",subset= sample_index)
summary(tree.d)
```

```
##
## Classification tree:
## tree(formula = TravelInsurance ~ ., data = Insurance_data, subset = sample_index,
##     split = "gini")
## Number of terminal nodes:  154
## Residual mean deviance:  0.763 = 1034 / 1355
## Misclassification error rate: 0.1637 = 247 / 1509
```

- The Insurance tree has 154 terminal nodes or leaves, which are the endpoints where the classification decisions are made.It is very crowded tree
- The residual mean deviance is 0.763. A lower deviance indicates a better fit of the model to the data.
- The misclassification error rate for this tree is 0.164 , which is calculated as 247 misclassified cases out of a total of 159 cases in train data.

(2) Create a plot of the tree. Pick one of the terminal nodes, and interpret the information displayed.

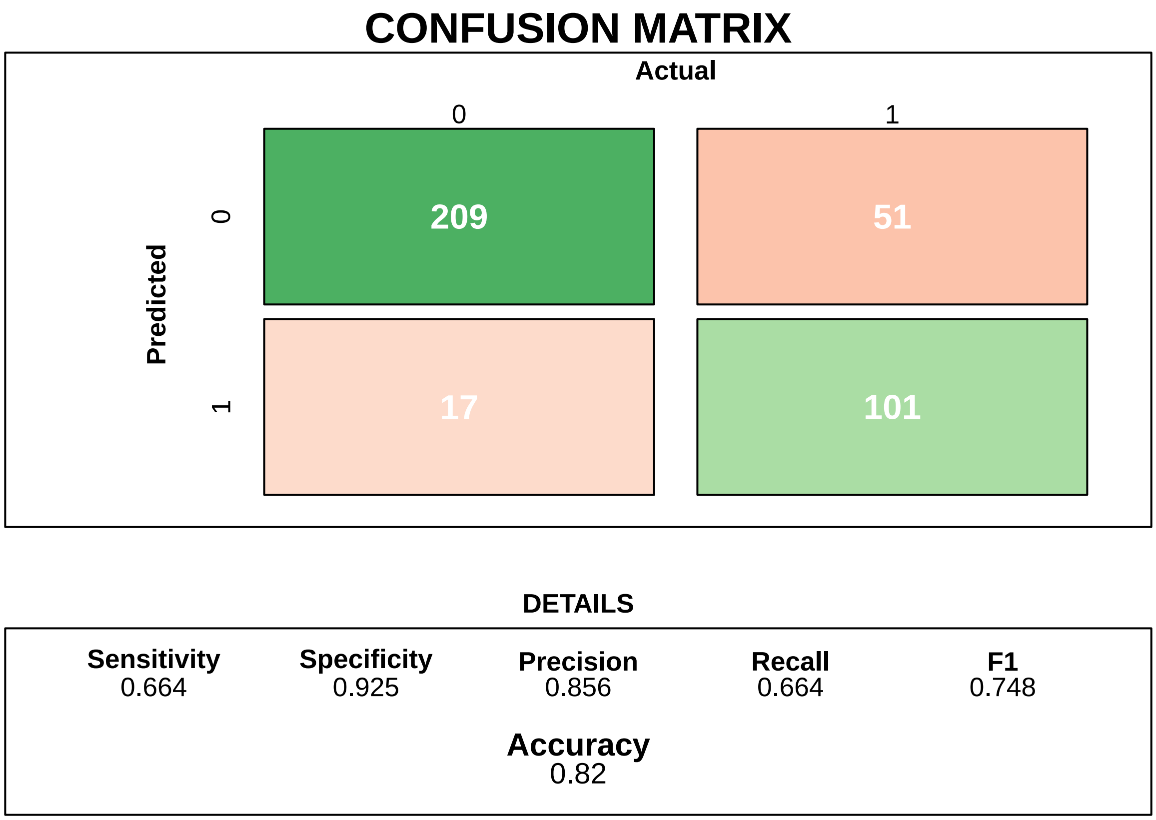
```r
plot(tree.d)
```



```r
#text(tree.d)
#tree.d
```

(3) Predict the labels on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```r
set.seed(100)
pred.d=predict(tree.d,test_data,type="class")

DT.cm=confusionMatrix(pred.d, test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(DT.cm)
```

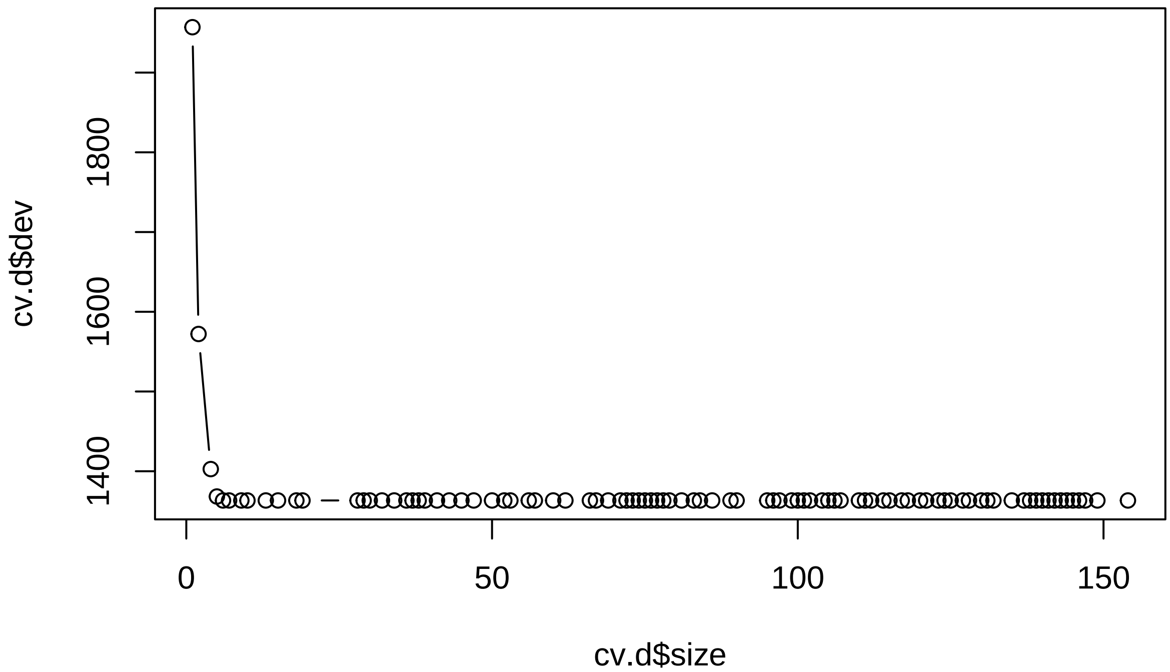



```
table(pred.d,test_data$TravelInsurance)
```

```
##
## pred.d   0   1
##      0 209  51
##      1  17 101
```

(4) Apply the cv.tree() function to the training set in order to determine the optimal tree size. Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis. Which tree size corresponds to the lowest cross-validated classification error rate?

```
#pruning
cv.d=cv.tree(tree.d)
plot(cv.d$size, cv.d$dev, type="b")
```




```
cv.d$dev
```

```
##  [1] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
##  [9] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [17] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [25] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [33] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [41] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [49] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [57] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [65] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [73] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [81] 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359 1363.359
## [89] 1368.298 1402.694 1572.127 1956.913
```

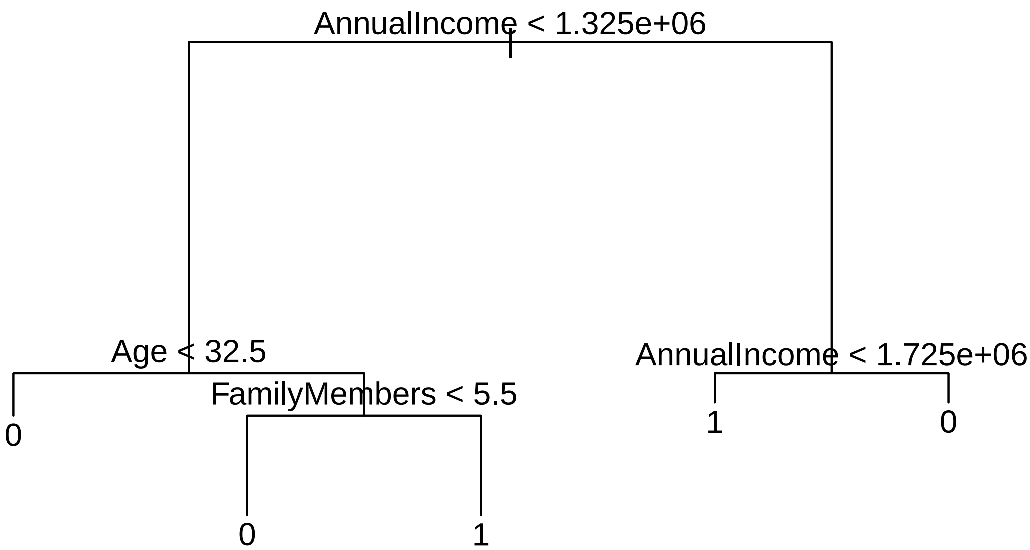
Because deviance error is constant after tree size = 5,I chose tree size = 5.

(5) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
prune.d=prune.tree(tree.d,best=5)
summary(prune.d)
```

```
##
## Classification tree:
## snip.tree(tree = tree.d, nodes = c(7L, 6L, 11L, 10L, 4L))
## Variables actually used in tree construction:
## [1] "AnnualIncome"  "Age"           "FamilyMembers"
## Number of terminal nodes:  5
## Residual mean deviance:  0.9015 = 1356 / 1504
## Misclassification error rate: 0.1723 = 260 / 1509
```

```
plot(prune.d)
text(prune.d)
```



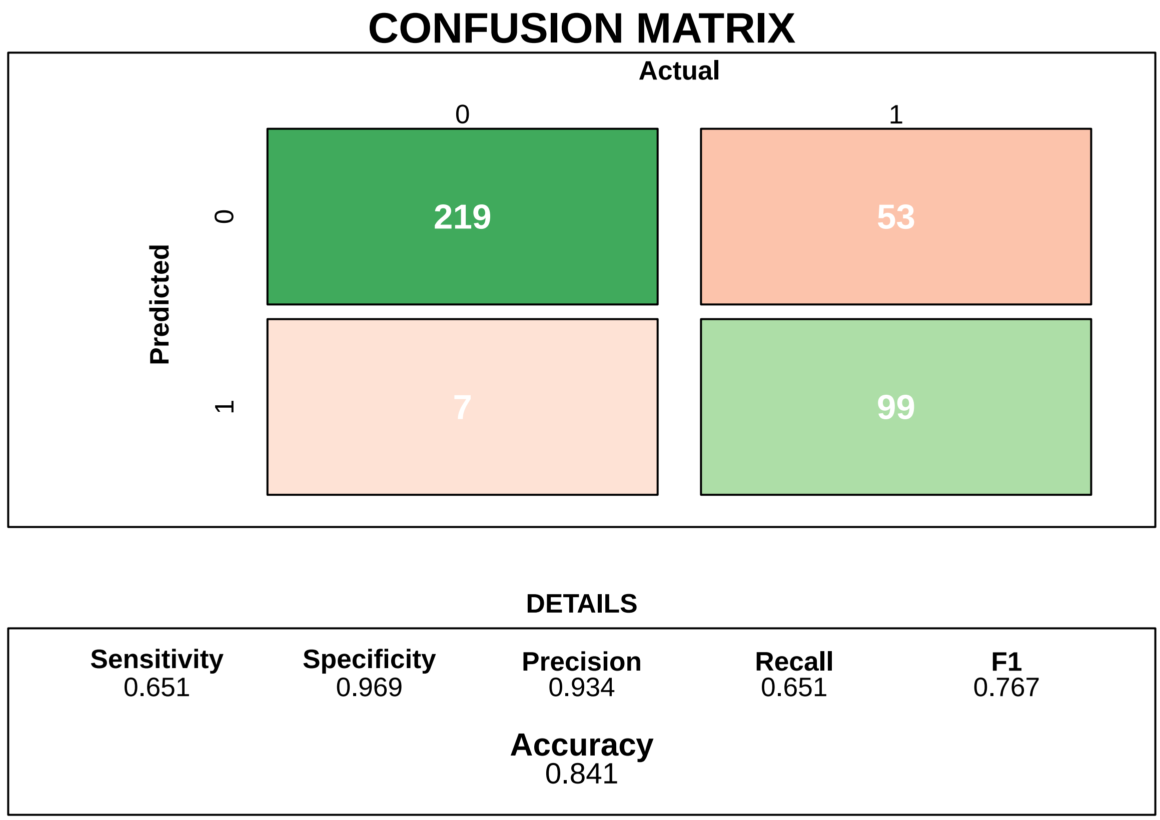
for Best tree size = 5:

Residual mean deviance: 0.901 = 1360 / 1500 Misclassification error rate: 0.172 = 260 / 1509

Both Residual mean deviance and Misclassification error rate are greater for best tree size = 5

(6) Compare the training and test error rates between the pruned and unpruned trees. Which is higher?

```
set.seed(100)
pred.prune=predict(prune.d,test_data,type="class")

prune_DT.cm=confusionMatrix(pred.prune, test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(prune_DT.cm)
```




```
table(pred.d,test_data$TravelInsurance)
```

```
##
## pred.d   0   1
##      0 209  51
##      1  17 101
```

```
(test.error.DT=(42+26)/(270))
```

```
## [1] 0.2518519
```

```
(test.error.prune=(31+32)/270)
```

```
## [1] 0.2333333
```

For DT: Residual mean deviance: 0.6359 = 455.3 / 716 Misclassification error rate: 0.1525 = 122 / 800 test.error.DT = 0.251 = (42+26)/(270)

For Pruned DT: Residual mean deviance: 1.088 = 863.5 / 794 Misclassification error rate: 0.2788 = 223 / 800 test.error.prune= 0.233 (31+32)/270

The test error for pruned DT is less that the test error for unpruned dt which is predictable.

Random Forest

```
set.seed(100)
bag.Insurance_data=randomForest(TravelInsurance~., data=Insurance_data, subset= sample_index, mtry=8, i
bag.Insurance_data # lets take a look at the output
```

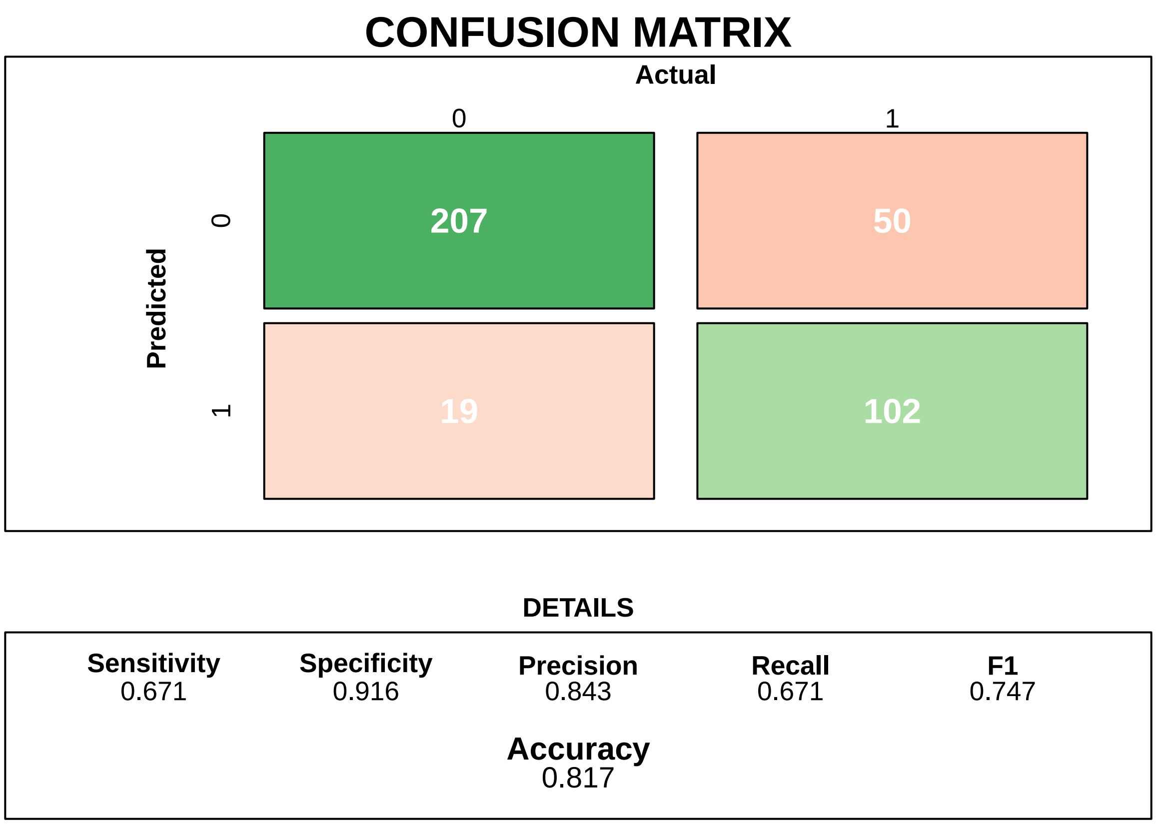
```
##
## Call:
```

```
##  randomForest(formula = TravelInsurance ~ ., data = Insurance_data,      mtry = 8, importance = TRUE
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 21.6%
## Confusion matrix:
##     0   1 class.error
## 0 860 120    0.122449
## 1 206 323    0.389414
```

```
yhat.bag=predict(bag.Insurance_data, newdata = test_data)


table(yhat.bag, test_data$TravelInsurance)
```

```
##
## yhat.bag   0   1
##        0 207  50
##        1  19 102
```

```
bag.cm=confusionMatrix(yhat.bag, test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(bag.cm)
```




```
(mtry=round(sqrt(8),0))
```

```
## [1] 3
```

```
# best mtry =3
```

```r
set.seed(100)


(rf.fit3=randomForest(TravelInsurance~., data=Insurance_data, subset= sample_index, ntree=1000, mtry=3,
```

```
##
## Call:
##  randomForest(formula = TravelInsurance ~ ., data = Insurance_data,      ntree = 1000, mtry = 3, impo
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.42%
## Confusion matrix:
##     0   1 class.error
## 0 930  50  0.05102041
## 1 228 301  0.43100189
```

```r
importance(bag.Insurance_data)
```

```
##                             0          1 MeanDecreaseAccuracy MeanDecreaseGini
## Age                 31.808691  43.947378           51.1346503        83.184216
## Employment.Type     17.209817   3.879430           18.3146677        12.742815
## GraduateOrNot       12.091890   9.076818           15.8103116         8.623758
## AnnualIncome        75.650550 112.331823          124.9789761       286.112323
## FamilyMembers       22.568557  53.404849           47.3448836       132.096367
## ChronicDiseases     -4.193267   6.575596            0.4094072        28.998157
## FrequentFlyer        1.646629   3.999756            3.9853358        19.094321
## EverTravelledAbroad  7.940023   1.606792            7.7624097        11.239449
```

```r
importance(rf.fit3)
```

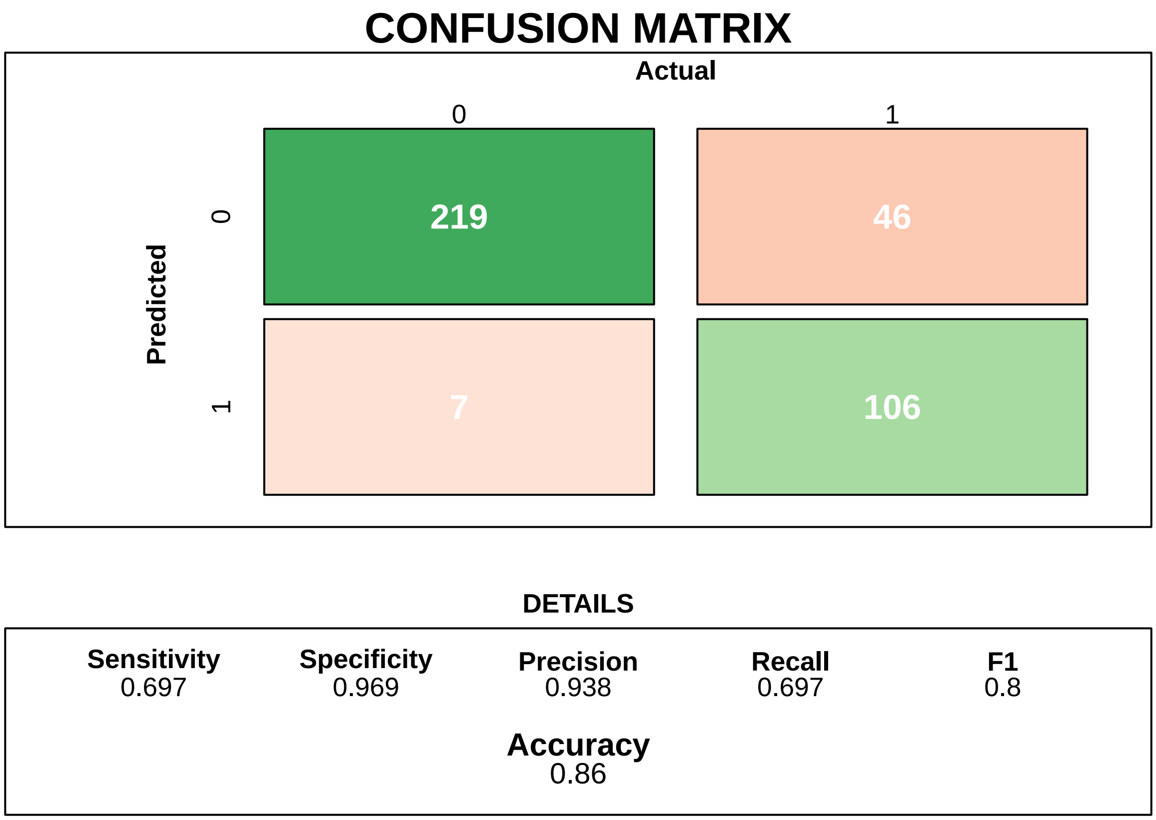
```
##                             0          1 MeanDecreaseAccuracy MeanDecreaseGini
## Age                 47.862392  57.836976            73.699745         79.49270
## Employment.Type     19.183589  12.656292            24.180313         11.92546
## GraduateOrNot        8.011737  13.316647            14.974379          8.75125
## AnnualIncome        40.587780 130.352941           104.434678        198.70659
## FamilyMembers       38.907573  64.885550            66.955196         82.97734
## ChronicDiseases     -4.406492   6.706806             0.222829         16.95500
## FrequentFlyer        9.157400  24.777226            24.339995         18.55546
## EverTravelledAbroad  9.242002  33.824674            35.117362         49.18150
```

AnnualIncome, Age, FamilyMembers, Employment.Type, EverTravelledAbroadYes are important variables in Bagging and RF.

```r
set.seed(100)
yhat.RF=predict(rf.fit3, newdata = test_data)
table(yhat.RF, test_data$TravelInsurance)
```

```
##
## yhat.RF   0   1
##       0 219  46
##       1   7 106
```

```r
RF.cm=confusionMatrix(yhat.RF, test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(RF.cm)
```




```
TravelInsuranceTest$TravelInsurance=rep(0,100)
TravelInsuranceTest$TravelInsurance=as.factor(TravelInsuranceTest$TravelInsurance)
yhat.RF=predict(rf.fit3, newdata = TravelInsuranceTest)
table(yhat.RF)
```

```
## yhat.RF
##  0  1
## 78 22
```

```
TravelInsuranceTest$TravelInsurance=yhat.RF
#write.csv(TravelInsuranceTest,"TravelInsuranceTest_Labeled.csv")
#read.csv("TravelInsuranceTest_Labeled.csv",header = T)
```

XGBoost

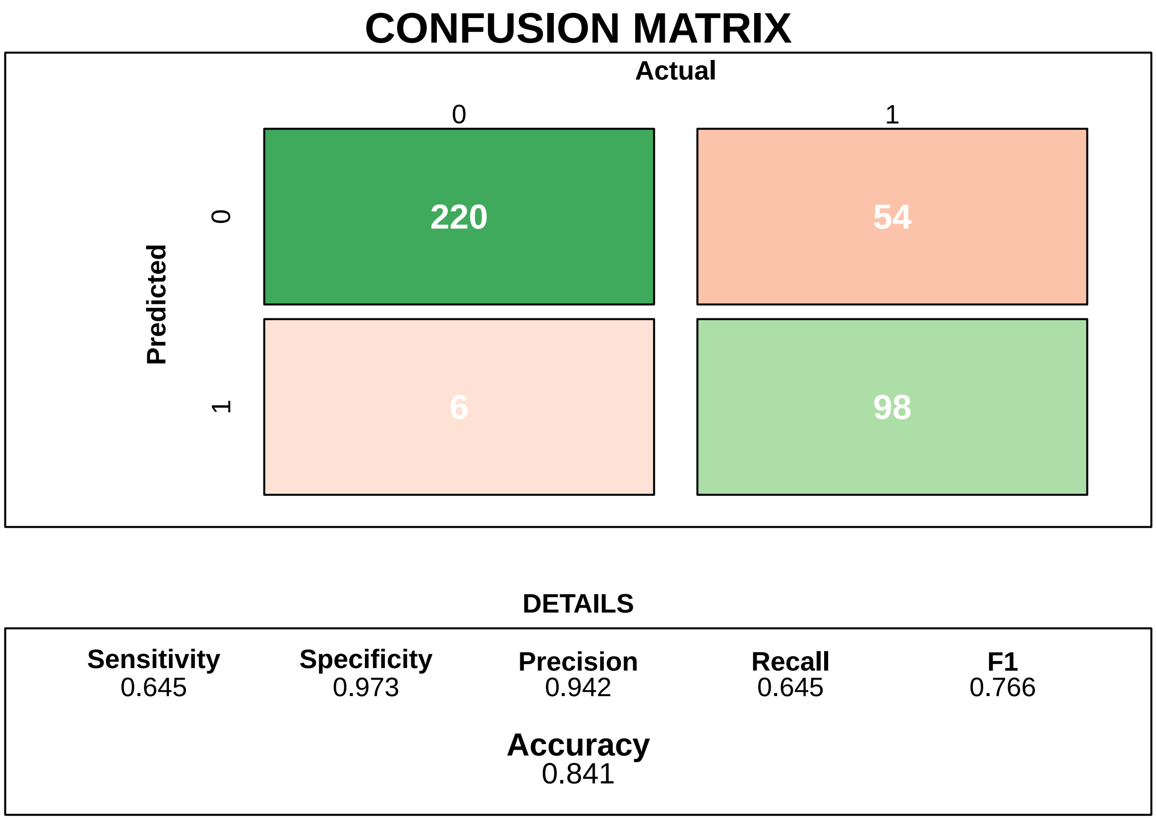
hyperparameter tunning

```
set.seed(100)

grid_gbm = expand.grid(nrounds = c(1,10),
                      max_depth = c(1,4),
                      eta = c(.1,.4),
                      gamma = 0,
                      colsample_bytree = .7,
                      min_child_weight = 1,
                      subsample = c(.8,1))
```

```r
xgboost_hp = train(TravelInsurance~.,
                                  data=train_data,
                                  method="xgbTree",
                                  trControl=trainControl(method = "cv", number = 5),
                                  tuneGrid = grid_gbm)
```

```
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
## [04:42:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instea
```

```r
predicted_xgboost = predict(xgboost_hp ,test_data)

cm_xgboost = confusionMatrix(predicted_xgboost,test_data$TravelInsurance,positive = "1")

draw_confusion_matrix(cm_xgboost)
```



SVM

(d) Tune the linear SVM with various values of cost. Report the cross-validation errors associated with different values of this parameter. Select an optimal cost. Compute the training and test error rates using this new cost value. Comment on your findings.

```
set.seed(100)
tune.out=tune(svm,TravelInsurance~.,data = train_data,kernel="linear",scale=T,
              ranges=list(cost=c(0.001, 0.01, 0.1,0.5, 1,2,5,10,100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.2366269
##
## - Detailed performance results:
##     cost     error dispersion
## 1 1e-03 0.3505784 0.03920048
## 2 1e-02 0.2366269 0.04428797
## 3 1e-01 0.2571567 0.04059054
## 4 5e-01 0.2571567 0.04059054
```

```
## 5 1e+00 0.2571567 0.04059054
## 6 2e+00 0.2571567 0.04059054
## 7 5e+00 0.2571567 0.04059054
## 8 1e+01 0.2571567 0.04059054
## 9 1e+02 0.2571567 0.04059054
```

```
tune.out$best.parameters
```

```
##   cost
## 2 0.01
```

```
tune.out$best.performance
```

```
## [1] 0.2366269
```

```
set.seed(100)
best.fit  = svm(TravelInsurance~.,data = train_data, kernel = "linear", cost = 0.01, scale = TRUE)

# best fit traning error rate
pred_train=predict(best.fit , train_data)
table(pred_train, train_data$TravelInsurance)
```

```
##
## pred_train   0   1
##          0 919 301
##          1  61 228
```

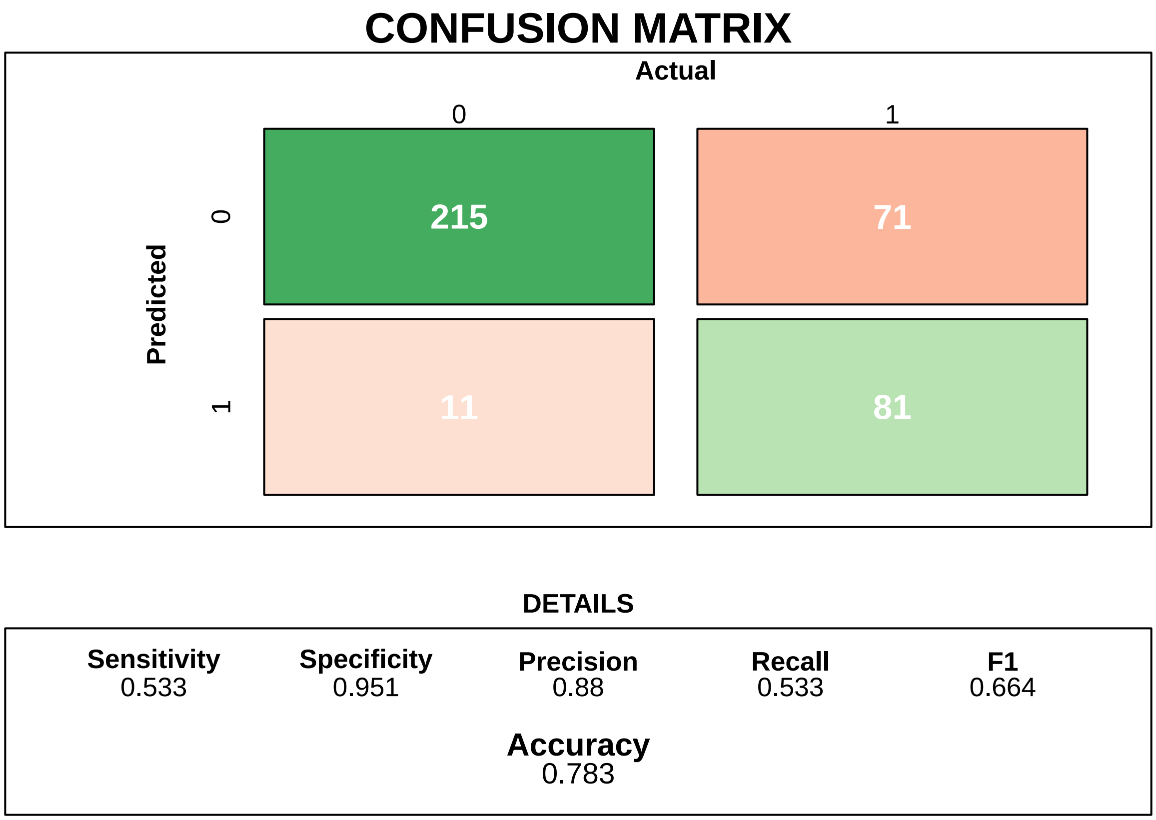
```
 # best performance error : 0.2366
best.train.err=(301+61)/1506

# best fittest error rate
pred_test=predict(best.fit , test_data)
table(pred_test, test_data$TravelInsurance)
```

```
##
## pred_test   0   1
##         0 215  71
##         1  11  81
```

```
(best.test.err = (71+11)/378)
```

```
## [1] 0.2169312
```

```
#0.2169

#########################
cm_svm = confusionMatrix(pred_test,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_svm)
```




```
##################################
```

finding the best cost using crossvalidation and fit the model

we tuned the svm model with 10-fold cross validation, the best parameter for cost =0.01, and best performance error = 0.2366

The error rate in the training data is 0.24045, and in the test data is 0.2169.

(e) Now repeat (d), with radial basis kernels, with different values of gamma and cost. Comment on your results. Which approach seems to give the better results on this data?

```
set.seed(100)
# finding best values of gomma and cost
tune.out=tune(svm,TravelInsurance~.,data = train_data, kernel="radial",ranges=list(cost=c(0.001, 0.01, 
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.1856203
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-03   0.1 0.3505784 0.03920048
```

```
## 2  1e-02   0.1 0.3505784 0.03920048
## 3  1e-01   0.1 0.2200706 0.04793568
## 4  1e+00   0.1 0.1915717 0.04585993
## 5  5e+00   0.1 0.1862781 0.05114341
## 6  1e+01   0.1 0.1862781 0.04748726
## 7  1e-03   0.5 0.3505784 0.03920048
## 8  1e-02   0.5 0.3505784 0.03920048
## 9  1e-01   0.5 0.2081413 0.04608717
## 10 1e+00   0.5 0.1856203 0.05170815
## 11 5e+00   0.5 0.1915585 0.03756448
## 12 1e+01   0.5 0.1895585 0.02890363
## 13 1e-03   1.0 0.3505784 0.03920048
## 14 1e-02   1.0 0.3505784 0.03920048
## 15 1e-01   1.0 0.2445784 0.03919909
## 16 1e+00   1.0 0.1915806 0.04831654
## 17 5e+00   1.0 0.2101060 0.03719872
## 18 1e+01   1.0 0.2273377 0.03336786
## 19 1e-03   2.0 0.3505784 0.03920048
## 20 1e-02   2.0 0.3505784 0.03920048
## 21 1e-01   2.0 0.3293819 0.04404555
## 22 1e+00   2.0 0.2127903 0.04797584
## 23 5e+00   2.0 0.2432539 0.04318160
## 24 1e+01   2.0 0.2512009 0.03840622
## 25 1e-03   3.0 0.3505784 0.03920048
## 26 1e-02   3.0 0.3505784 0.03920048
## 27 1e-01   3.0 0.3419647 0.04064535
## 28 1e+00   3.0 0.2240442 0.04551785
## 29 5e+00   3.0 0.2452362 0.03782538
## 30 1e+01   3.0 0.2465651 0.03935492
## 31 1e-03   4.0 0.3505784 0.03920048
## 32 1e-02   4.0 0.3505784 0.03920048
## 33 1e-01   4.0 0.3479294 0.03715165
## 34 1e+00   4.0 0.2379603 0.04707405
## 35 5e+00   4.0 0.2498852 0.04174338
## 36 1e+01   4.0 0.2498852 0.04266707
```

```r
tune.out$best.parameters
```

```
##    cost gamma
## 10    1   0.5
```

```r
tune.out$best.performance
```

```
## [1] 0.1856203
```

```r
radial.svmfit = svm(TravelInsurance~.,data = train_data, kernel = "radial",gamma=0.1, cost = 10, decisi

# traning error rate
radial.pred_train =predict(radial.svmfit , train_data)
table(radial.pred_train , train_data$TravelInsurance)
```

```
##
## radial.pred_train   0   1
##                 0 943 231
##                 1  37 298
```

```
(radial.train.err = (231+37)/1509)
```

```
## [1] 0.1776011
```

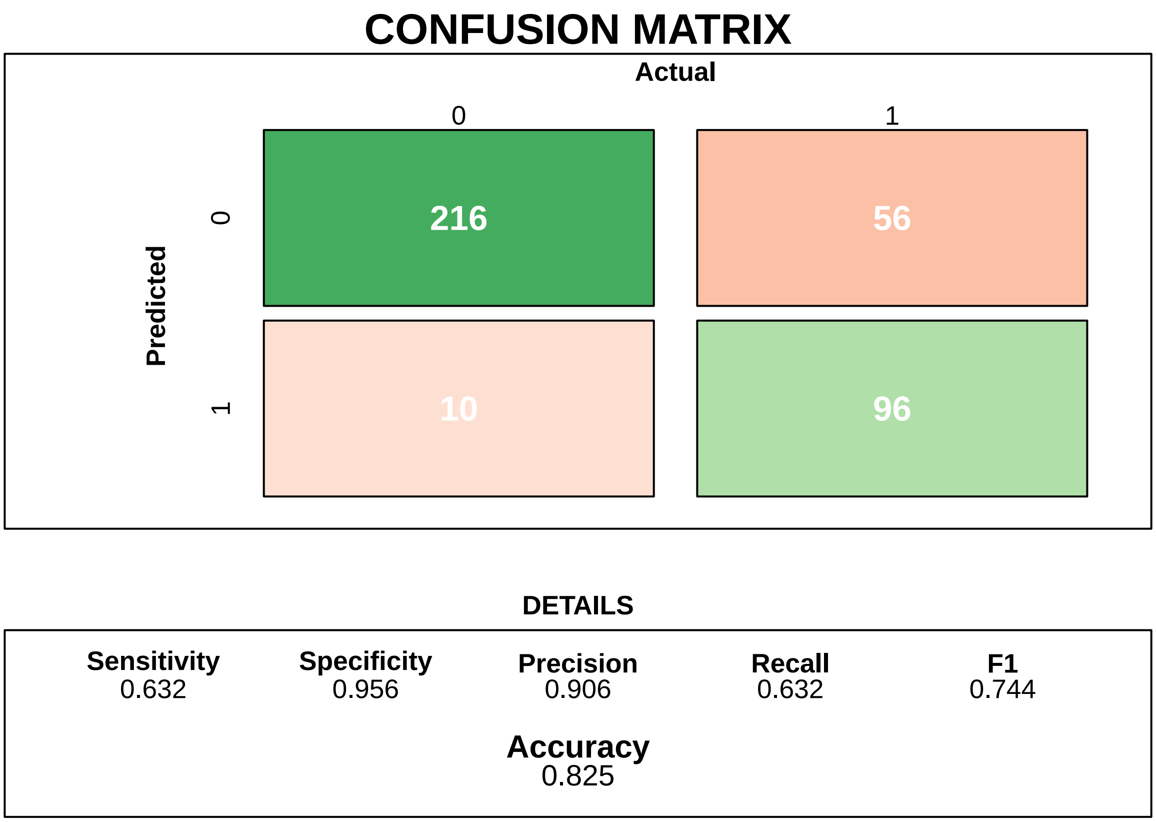
```
# test error rate
radial.pred_test=predict(radial.svmfit, test_data)
table(radial.pred_test, test_data$TravelInsurance)
```

```
##
## radial.pred_test   0   1
##                0 216  56
##                1  10  96
```

```
(radial.test.err = (56+10)/378)
```

```
## [1] 0.1746032
```

```
cm_svm_radial = confusionMatrix(radial.pred_test,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_svm_radial)
```



permormance error: 0.1829 cost=10 gamma =0.1 The training error for the radial kernel (0.1776) is lower than that of the linear kernel (0.24045). However, the test error for the radial kernel (0.1746) less than the linear kernel (0.2169). Therefore, based on these results, it appears that the redial kernel is more effective for our dataset.

(f) Now repeat again, with polynomial basis kernels, with different values of degree and cost. Comment on your results. Which approach (kernel) seems to give the best results on this data?

```
set.seed(100)
# finding best values of gomma and cost
tune.out=tune(svm,TravelInsurance~.,data = train_data, kernel="polynomial",ranges=list(cost=c(0.001, 0.
```

```r
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##     5      3
##
## - best performance: 0.1856159
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-03    0.1 0.3505784 0.03920048
## 2  1e-02    0.1 0.3505784 0.03920048
## 3  1e-01    0.1 0.3505784 0.03920048
## 4  1e+00    0.1 0.3505784 0.03920048
## 5  5e+00    0.1 0.3505784 0.03920048
## 6  1e+01    0.1 0.3505784 0.03920048
## 7  1e+02    0.1 0.3505784 0.03920048
## 8  1e-03    0.5 0.3505784 0.03920048
## 9  1e-02    0.5 0.3505784 0.03920048
## 10 1e-01    0.5 0.3505784 0.03920048
## 11 1e+00    0.5 0.3505784 0.03920048
## 12 5e+00    0.5 0.3505784 0.03920048
## 13 1e+01    0.5 0.3505784 0.03920048
## 14 1e+02    0.5 0.3505784 0.03920048
## 15 1e-03    1.0 0.3505784 0.03920048
## 16 1e-02    1.0 0.3505784 0.03920048
## 17 1e-01    1.0 0.2445784 0.04605799
## 18 1e+00    1.0 0.2571567 0.04059054
## 19 5e+00    1.0 0.2571567 0.04059054
## 20 1e+01    1.0 0.2571567 0.04059054
## 21 1e+02    1.0 0.2571567 0.04059054
## 22 1e-03    2.0 0.3505784 0.03920048
## 23 1e-02    2.0 0.3505784 0.03920048
## 24 1e-01    2.0 0.2233731 0.04642170
## 25 1e+00    2.0 0.2227196 0.04492097
## 26 5e+00    2.0 0.2154305 0.04415458
## 27 1e+01    2.0 0.2134481 0.04459755
## 28 1e+02    2.0 0.2127859 0.04445315
## 29 1e-03    3.0 0.3505784 0.03920048
## 30 1e-02    3.0 0.3505784 0.03920048
## 31 1e-01    3.0 0.2339823 0.05184210
## 32 1e+00    3.0 0.1922428 0.04631032
## 33 5e+00    3.0 0.1856159 0.04905982
## 34 1e+01    3.0 0.1882649 0.04578042
## 35 1e+02    3.0 0.1909007 0.04263200
## 36 1e-03    4.0 0.3505784 0.03920048
## 37 1e-02    4.0 0.3505784 0.03920048
## 38 1e-01    4.0 0.2452362 0.04983317
## 39 1e+00    4.0 0.2008477 0.04549404
```

```
## 40 5e+00    4.0 0.1875938 0.04590359
## 41 1e+01    4.0 0.1862781 0.04758977
## 42 1e+02    4.0 0.1929007 0.05252725
```

```
tune.out$best.parameters
```

```
##    cost degree
## 33    5      3
```

```
tune.out$best.performance
```

```
## [1] 0.1856159
```

```
poly.svmfit = svm(TravelInsurance~.,data = train_data, kernel = "radial",degree=3, cost = 5, decision.va
# traning error rate
poly.pred_train =predict(poly.svmfit , train_data)
table(poly.pred_train , train_data$TravelInsurance)
```

```
##
## poly.pred_train   0   1
##               0 942 233
##               1  38 296
```

```
(poly.train.err = (233+38)/1509)
```

```
## [1] 0.1795891
```

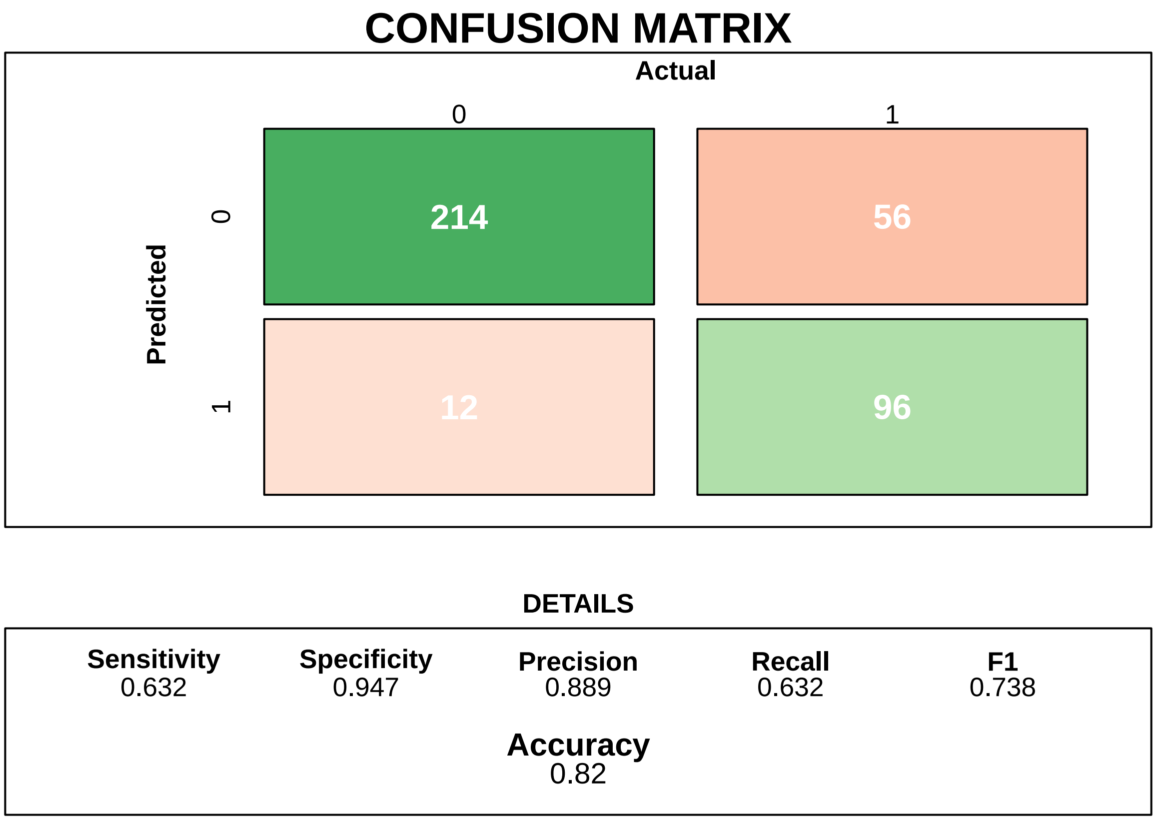
```
# test error rate
poly.pred_test=predict(poly.svmfit, test_data)
table(poly.pred_test, test_data$TravelInsurance)
```

```
##
## poly.pred_test   0   1
##              0 214  56
##              1  12  96
```

```
(poly.test.err = (56+12)/378)
```

```
## [1] 0.1798942
```

```
#########################
cm_svm_poly = confusionMatrix(poly.pred_test,test_data$TravelInsurance,positive = "1")
draw_confusion_matrix(cm_svm_poly)
```




```
##################################
c(best.train.err,best.test.err)
```

```
## [1] 0.2403718 0.2169312
```

```
c(radial.train.err,radial.test.err)
```

```
## [1] 0.1776011 0.1746032
```

```
c(poly.train.err,poly.test.err)
```

```
## [1] 0.1795891 0.1798942
```

cost = 5 degree = 3 performance error = 0.1856 comparing the traning and test error for linear, radial, and polynomial kernels we can see that radial kernel has the best performance.

Neural Network

```
standardize=function(x) {(x-min(x))/(max(x)-min(x))}
std.data=Insurance_data
std.data$AnnualIncome=standardize(std.data$AnnualIncome)
std.data$Age=standardize(std.data$Age)
std.data$FamilyMembers=standardize(std.data$FamilyMembers)
set.seed(100)
ind=sample(1:nrow(std.data), 0.8*nrow(std.data))
train=std.data[ind,]
test=std.data[-ind,]
```

```r
set.seed(100)
fit=nnet(TravelInsurance~., data=train,decay=0.1, size=10, liout=FALSE)
```

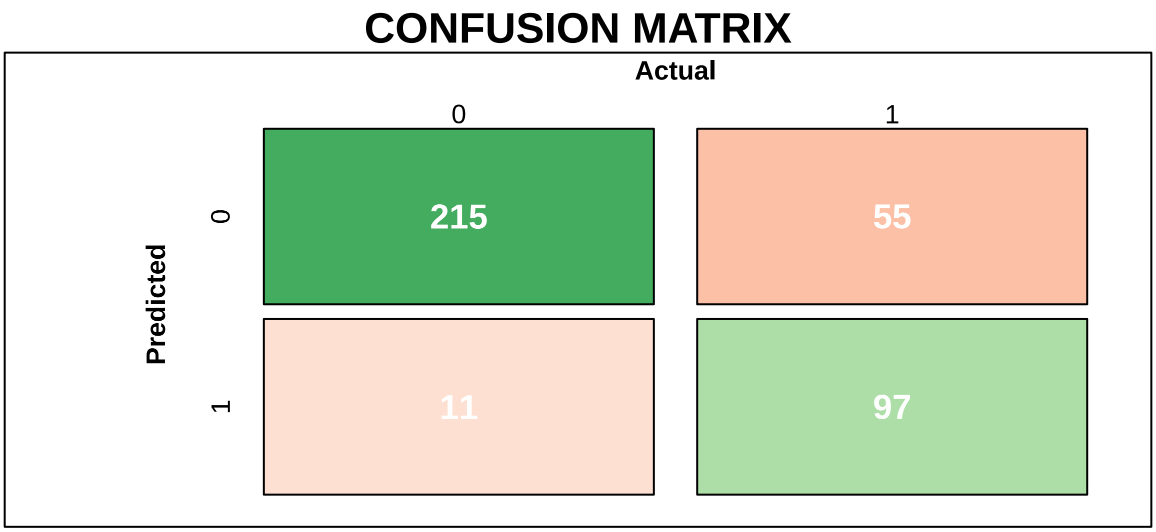
```
## # weights:  101
## initial  value 1122.179933
## iter  10 value 816.650248
## iter  20 value 763.223410
## iter  30 value 739.491369
## iter  40 value 725.620421
## iter  50 value 720.433385
## iter  60 value 717.795530
## iter  70 value 716.821754
## iter  80 value 716.263152
## iter  90 value 715.479459
## iter 100 value 714.797982
## final  value 714.797982
## stopped after 100 iterations
```

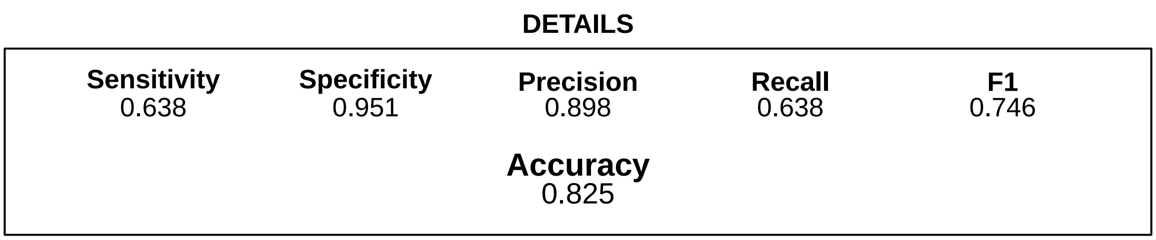
(b) Compare the classification performance of your model with that of linear logistic regression.

```r
set.seed(100)
NN_probs=predict(fit, test)
NN_pred <- rep("No",378)
NN_pred[NN_probs > 0.5] = "Yes"

# The confusion matrix
(cm <- table( NN_pred,test_data$TravelInsurance))
```

```
##
## NN_pred   0   1
##      No  215  55
##      Yes  11  97
```

```r
#drawing confusion matrix
##################################################
NN_predicted_labels= ifelse(NN_probs > 0.5, 1, 0)

pred=as.factor(NN_predicted_labels)
cm_NN = confusionMatrix(pred,test$TravelInsurance,positive= "1")

draw_confusion_matrix(cm_NN)
```






```r
#####################################################
set.seed(100)
mygrid=expand.grid(.decay=c(0.05,0.1),.size=c(3,4,5,6,7,8,9,10,12))
nnetfit=train(TravelInsurance~., data=train, method= "nnet", mmaxit=1000,tuneGrid= mygrid,trace=F)
nnetfit
```

```
## Neural Network
##
## 1509 samples
##    8 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1509, 1509, 1509, 1509, 1509, 1509, ...
## Resampling results across tuning parameters:
##
##   decay  size  Accuracy   Kappa
##   0.05    3    0.7774416  0.4674308
##   0.05    4    0.7826637  0.4820034
##   0.05    5    0.7808932  0.4832054
##   0.05    6    0.7785403  0.4767325
##   0.05    7    0.7784117  0.4786143
##   0.05    8    0.7799334  0.4821469
##   0.05    9    0.7774541  0.4777516
##   0.05   10    0.7810647  0.4874769
##   0.05   12    0.7740846  0.4732448
##   0.10    3    0.7764037  0.4651277
```

```
##   0.10    4    0.7822076  0.4807091
##   0.10    5    0.7839584  0.4872408
##   0.10    6    0.7840119  0.4887891
##   0.10    7    0.7827955  0.4867654
##   0.10    8    0.7794185  0.4797147
##   0.10    9    0.7791144  0.4803317
##   0.10   10    0.7794541  0.4804515
##   0.10   12    0.7797722  0.4828214
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 6 and decay = 0.1.
```

choose decay=0.1 and size=6 accuracy=0.7840

```r
set.seed(100)
fit=nnet(TravelInsurance~., data=train,decay=0.1, size=6, liout=FALSE)
```

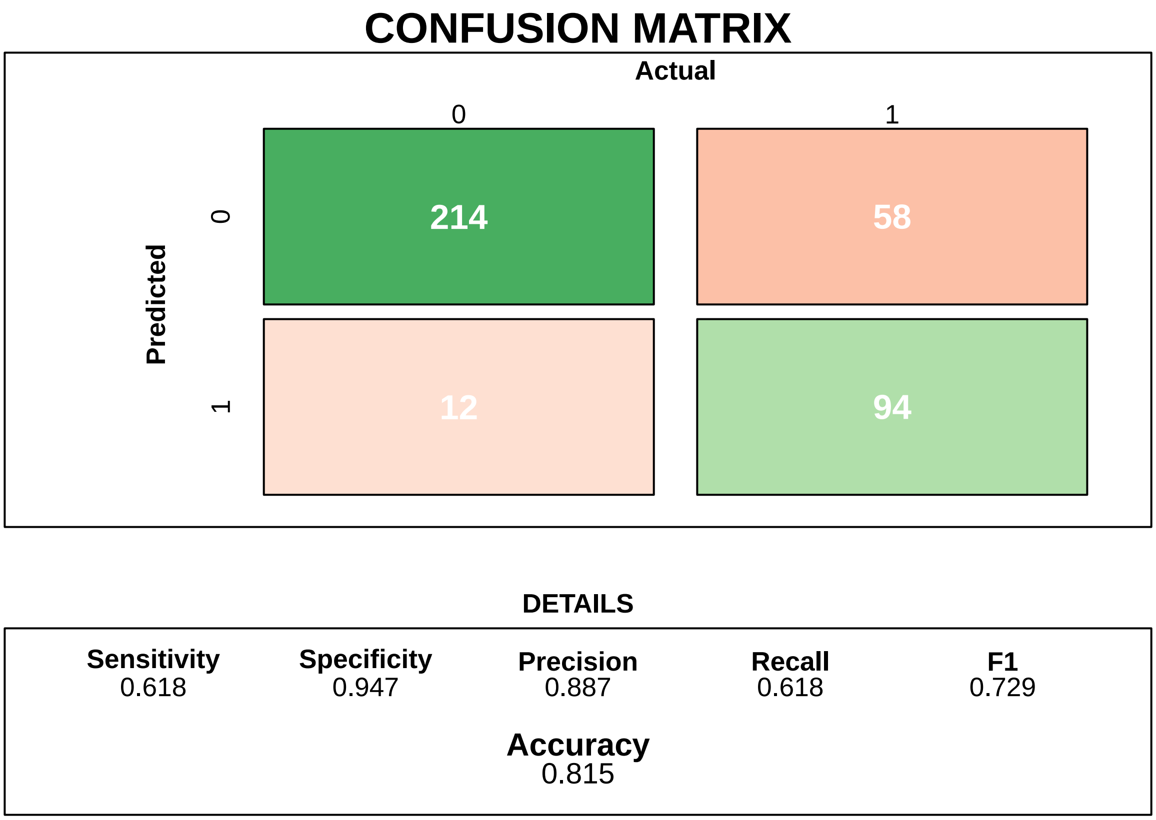
```
## # weights:  61
## initial  value 982.764547
## iter  10 value 810.172714
## iter  20 value 758.474694
## iter  30 value 743.609932
## iter  40 value 734.590877
## iter  50 value 731.003901
## iter  60 value 728.840800
## iter  70 value 727.548934
## iter  80 value 727.253517
## iter  90 value 727.132288
## iter 100 value 727.098660
## final  value 727.098660
## stopped after 100 iterations
```

(b) Compare the classification performance of your model with that of linear logistic regression.

```r
set.seed(100)
NN_probs=predict(fit, test)
NN_pred <- rep("No",378)
NN_pred[NN_probs > 0.5] = "Yes"

# The confusion matrix
(cm <- table( NN_pred,test$TravelInsurance))
```

```
##
## NN_pred   0   1
##     No  214  58
##     Yes  12  94
```

```r
#drawing confusion matrix
####################################################
NN_predicted_labels= ifelse(NN_probs > 0.5, 1, 0)

pred=as.factor(NN_predicted_labels)

cm_NN = confusionMatrix(pred,test$TravelInsurance,positive= "1")

draw_confusion_matrix(cm_NN)
```


CONFUSION MATRIX
Actual
0
1
Predicted
0
1
214
58
12
94
DETAILS
Sensitivity
0.618
Specificity
0.947
Precision
0.887
Recall
0.618
F1
0.729
Accuracy
0.815

```
##################################################
```