Assignment 3.1 [Python & R]

University of San Diego

ADS 502

Dingyi Duan

For Exercises 28–34, work with the churn data set.

28. Partition the data set, so that 67% of the records are included in the training data set and 33% are included in the test data set. Use a bar graph to confirm your proportions.

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
%load_ext rpy2.ipython
```

Python Packages

In [3]:
```
import random
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt



from sklearn.tree import DecisionTreeClassifier, export_graphviz
from statsmodels.stats.proportion import proportions_ztest
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
```

R Package

In [4]:
```
%%R
library(readr)
library(ggplot2)
library(C50)
library(e1071)
```

```
R[write to console]: RStudio Community is a great place to get help:
https://community.rstudio.com/c/tidyverse (https://community.rstudio.com/c/tidy
verse)
```

Python

In [5]:
```
churn_py = pd.read_csv("D:/2021-Spring-textbooks/ADS-502/Website Data Sets/churn'
```

In [6]:
```
churn_py.head()
```

Out[6]:

	State	Account Length	Area Code	Phone	Intl Plan	VMail Plan	VMail Message	Day Mins	Day Calls	Day Charge	...	Eve Charge	Night Mins
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	16.78	244.7
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	16.62	254.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	10.30	162.6
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	5.26	196.9
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	12.61	186.9

5 rows × 22 columns

In [7]:
```
churn_train_py, churn_test_py = train_test_split(churn_py, test_size = 0.33, ranc
```

In [8]:
```
print ('churn_train_py has %s records'% churn_train_py.shape[0],
       '\nchurn_test_py has %s records'% churn_test_py.shape[0])
```

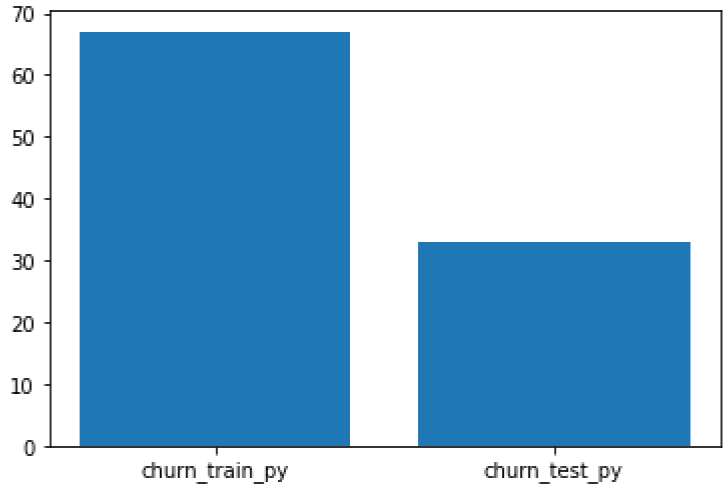
```
churn_train_py has 2233 records
churn_test_py has 1100 records
```

In [9]:
```
print ('The proportion for churn_train_py is %s'% round(churn_train_py.shape[0]*1
       '\nThe proportion for churn_test_py is %s'% round(churn_test_py.shape[0]*1
```

```
The proportion for churn_train_py is 67.0
The proportion for churn_test_py is 33.0
```

In [10]:
```
names_py = ['churn_train_py', 'churn_test_py']
values_pct_py = [round(churn_train_py.shape[0]*100/churn_py.shape[0],0),round(chu
plt.bar(names_py, values_pct_py)
```

Out[10]: <BarContainer object of 2 artists>



R

In [11]:
```
%%R
churn_r <- read.csv(file = "D:/2021-Spring-textbooks/ADS-502/Website Data Sets/ch
set.seed(7)
n <- dim(churn_r)[1]
head(churn_r)
```

```
  State Account.Length Area.Code    Phone Intl.Plan VMail.Plan VMail.Message
1    KS            128       415 382-4657        no        yes            25
2    OH            107       415 371-7191        no        yes            26
3    NJ            137       415 358-1921        no         no             0
4    OH             84       408 375-9999       yes         no             0
5    OK             75       415 330-6626       yes         no             0
6    AL            118       510 391-8027       yes         no             0
  Day.Mins Day.Calls Day.Charge Eve.Mins Eve.Calls Eve.Charge Night.Mins
1    265.1       110      45.07    197.4        99      16.78      244.7
2    161.6       123      27.47    195.5       103      16.62      254.4
3    243.4       114      41.38    121.2       110      10.30      162.6
4    299.4        71      50.90     61.9        88       5.26      196.9
5    166.7       113      28.34    148.3       122      12.61      186.9
6    223.4        98      37.98    220.6       101      18.75      203.9
  Night.Calls Night.Charge Intl.Mins Intl.Calls Intl.Charge CustServ.Calls
1          91        11.01      10.0          3        2.70              1
2         103        11.45      13.7          3        3.70              1
3         104         7.32      12.2          5        3.29              0
4          89         8.86       6.6          7        1.78              2
5         121         8.41      10.1          3        2.73              3
6         118         9.18       6.3          6        1.70              0
  Old.Churn Churn
1    False. False
2    False. False
3    False. False
4    False. False
5    False. False
6    False. False
```

In [12]:
```
%%R
train_ind <- runif(dim(churn_r)[1]) <= 0.67 # Determin training data proportion
```

In [13]:
```
%%R
# Split training and test datasets
churn_train_r <- churn_r[ train_ind, ]
churn_test_r <- churn_r[ !train_ind, ]
```

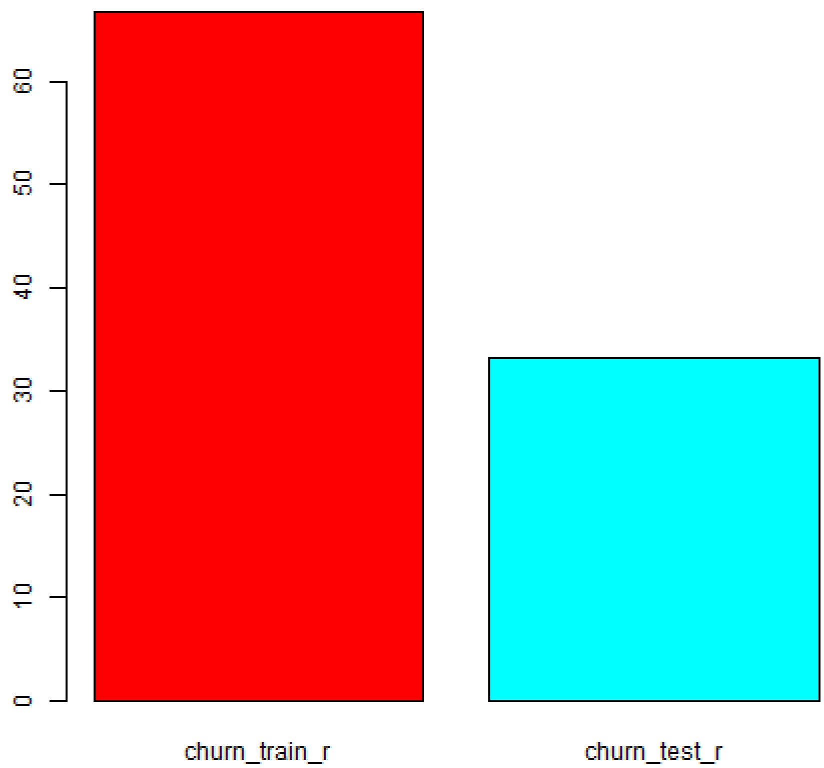
In [14]:
```
%%R
dim(churn_train_r)[1]
```

```
[1] 2227
```

In [15]:
```
%%R
dim(churn_test_r)[1]
```

```
[1] 1106
```

In [16]:
```
%%R
values_pct_r <- c((dim(churn_train_r)[1])*100/dim(churn_r)[1],(dim(churn_test_r)|
names_r <- c("churn_train_r","churn_test_r")
barplot(values_pct_r,names.arg=names_r,col = rainbow(2))
```



29. Identify the total number of records in the training data set and how many records in the training data set have a churn value of true.

Python

In [17]:
```
print('Total number of records in the training data set is ',churn_train_py.shape
```

```
Total number of records in the training data set is  2233
```

In [18]:
```
print('Number of records in the training data set have a churn value of true is '
```

```
Number of records in the training data set have a churn value of true is  320
```

In [19]:
```
# Or
churn_train_py['Churn'].value_counts()
```

Out[19]:
```
False    1913
True      320
Name: Churn, dtype: int64
```

R

In [20]:
```
%%R
n_train_r <- dim(churn_train_r)[1]
print(paste('Total number of records in the training data set is ',n_train_r))
```

```
[1] "Total number of records in the training data set is  2227"
```

In [21]:
```
%%R
a <- table(churn_train_r$Churn)["True"]
print(paste('Number of records in the training data set have a churn value of tru
```

```
[1] "Number of records in the training data set have a churn value of true is
325"
```

In [22]:
```
%%R
# Or
table(churn_train_r$Churn)
```

```
False  True
 1902   325
```

30. Use your answers from the previous exercise to calculate how many true churn records you need to resample in order to have 20% of the rebalanced data set have true churn values.

Python

In [23]:
```
# Ratio of "True" to all Churn in churn_train
churn_train_py['Churn'].sum()*100 / churn_train_py.shape[0]
```

Out[23]:
```
14.33049708911778
```

Toal churn_train has 2233, Churn == 'True' has 320, which is 14.3% of all samles; We need 20%.

In [24]:
```
print('We need %s records with Churn == True to add to our training set' % int((
```

```
We need 158 records with Churn == True to add to our training set
```

R

In [25]:
```
%%R

print(paste('Current ratio of "Churn == True" to all training set is ', a*100/n_t
```

```
[1] "Current ratio of \"Churn == True\" to all training set is  14.593623709025
6"
```

In [26]:
```
%%R

# We need 20%
print(paste('The number of Churn == True we still need:', floor((0.2*(2233)-320)/
```

```
[1] "The number of Churn == True we still need: 158"
```

31. Perform the rebalancing described in the previous exercise and confirm that 20% of the records in the rebalanced data set have true churn values.

Python

In [27]:
```
to_resample_py = churn_train_py.loc[churn_train_py['Churn'] == True]
```

In [28]:
```
# Sample from our records of interest
```

In [29]:
```
our_resample_py = to_resample_py.sample(n = 158, replace = True)
```

In [30]:
```
# Add the resampled records to our original training data set
```

In [31]:
```
churn_train_rebal_py = pd.concat([churn_train_py, our_resample_py], axis=0)
```

In [32]:
```
churn_train_rebal_py['Churn'].sum() #The current Churn == True records
```

Out[32]:
```
478
```

In [33]:
```
# Or
churn_train_rebal_py['Churn'].value_counts()
```

Out[33]:
```
False    1913
True      478
Name: Churn, dtype: int64
```

In [34]:
```
# Proportion of our desired records in the new (rebalanced) train data set
```

In [35]:
```
round(churn_train_rebal_py['Churn'].sum()*100/churn_train_rebal_py.shape[0],2)
```

Out[35]:
```
19.99
```

R

In [36]:
```
%%R
# Identify the record indices we want to resample using which()
to.resample_r <- which(churn_train_r$Churn == 'True')
```

In [37]:
```
%%R
# Randomly sample from the values in to.resample
our.resample_r <- sample(x = to.resample_r, size = 158, replace = TRUE)
```

In [38]:
```
%%R
# Get the records whose record numbers are those in our.resample
our.resample_r <- churn_train_r[our.resample_r,]
```

In [39]:
```
%%R
# Add the resampled records back onto our original training data set
churn_train_rebal_r <- rbind(churn_train_r, our.resample_r)
```

In [40]:
```
%%R
t1 <- table(churn_train_rebal_r$Churn)
#ratio <- t1[2] / sum(t1) * 100
t1
```

```
False  True
 1902   483
```

In [41]:
```
%%R
# Now the new ratio of Churn == True
t1['True']*100/dim(churn_train_rebal_r)[1]
```

```
    True
20.25157
```

32. Which baseline model do we use to compare our classification model performance against? To which value does this baseline model assign all predictions? What is the accuracy of this baseline model?

Answer: For binary classification, we use All Positive Model or All Negative Model.

All Positive Model assigns all predictors to positive values; All Negative Model assigns all predictors to negative values.

The accuracy for All Positive Model will be p which is 14% from our test data set; The accuracy for All Negative Model will be 1-p which will be 86%.

33. Validate your partition by testing for the difference in mean day minutes for the training set versus the test set.

Python

In [42]:
```
# Before rebalance in train data set
mean_day_mins_train_before_py = churn_train_py['Day Mins'].sum() / churn_train_py
mean_day_mins_train_before_py
```

Out[42]:
```
179.75302283922974
```

In [43]:
```
# Mean days min in test set
mean_day_mins_test_py = churn_test_py['Day Mins'].sum() / churn_test_py.shape[0]
mean_day_mins_test_py
```

Out[43]:
```
179.81990909090908
```

Means are very similar, so running t-test

In [44]:
```
# Two samples t test
np.var(churn_test_py['Day Mins']), np.var(churn_train_py['Day Mins'])
```

Out[44]:
```
(3010.977976355374, 2943.552898385955)
```

In [45]:
```
# Perform two sample t-test with equal variances
stats.ttest_ind(a=churn_test_py['Day Mins'], b=churn_train_py['Day Mins'], equal_
```

Out[45]:
```
Ttest_indResult(statistic=0.03333175951101492, pvalue=0.9734120244442176)
```

p-value = 0.973 which is greater than 0.05, we fail to reject the null hypothesis of the test. We do not have sufficient evidence to say that the mean day minutes between the two data sets are different.

R

In [46]:
```
%%R
# Before rebalance in train data set
mean_day_mins_train_before_r <- sum(churn_train_r$Day.Mins) / dim(churn_train_r)[
mean_day_mins_train_before_r
```

```
[1] 180.911
```

In [47]:
```
%%R
# After rebalance in train data set
mean_day_mins_train_rebal_r <- sum(churn_train_rebal_r$Day.Mins) / dim(churn_trai
mean_day_mins_train_rebal_r
```

```
[1] 182.5936
```

In [48]:
```
%%R
# Mean days min in test set
mean_day_mins_test_r <- sum(churn_test_r$Day.Mins) / dim(churn_test_r)[1]
mean_day_mins_test_r
```

```
[1] 177.4879
```

In [49]:
```
%%R
# t-test
t.test(churn_test_r$Day.Mins,churn_train_r$Day.Mins)
```

```

        Welch Two Sample t-test

data:  churn_test_r$Day.Mins and churn_train_r$Day.Mins
t = -1.7248, df = 2259.7, p-value = 0.08471
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -7.315115  0.468881
sample estimates:
mean of x mean of y 
 177.4879  180.9110 
```

p-value = 0.085 which is greater than 0.05, we fail to reject the null hypothesis of the test. We do not have sufficient evidence to say that the mean day minutes between the two data sets are different.

34. Validate your partition by testing for the difference in proportion of true churn records for the training set versus the test set.

Python

In [50]:
```
# Ratio of "True" to all Churn in churn_train before rebalance
print(churn_train_py['Churn'].sum()*100 / churn_train_py.shape[0])
```

```
14.33049708911778
```

In [51]:
```
# Ratio of "True" to all Churn in churn_test
print(churn_test_py['Churn'].sum()*100 / churn_test_py.shape[0])
```

```
14.818181818181818
```

Two proportions are similar, running one proportion z-test

In [52]:
```
# Perform one proportion z-test
proportions_ztest(count=churn_train_py['Churn'].sum(), nobs=churn_train_py.shape
```

Out[52]:
```
(-0.6577180599814599, 0.5107193152727614)
```

p-value = 0.51 which is greater than 0.05, we fail to reject the null hypothesis of the test. We do not have sufficient evidence to say that the proportion of true churn records between the two data sets are different.

R

In [53]:
```
%%R
# Ratio of "True" to all Churn in churn_train before rebalance

table(churn_train_r$Churn)["True"]*100 / dim(churn_train_r)[1]
```

```
    True
14.59362
```

In [54]:
```
%%R
# Ratio of "True" to all Churn in churn_test

table(churn_test_r$Churn)["True"]*100 / dim(churn_test_r)[1]
```

```
    True
14.28571
```

In [55]:
```
%%R
# t-test
prop.test(c(table(churn_test_r$Churn)["True"],table(churn_train_r$Churn)["True"])
```

```
        2-sample test for equality of proportions with continuity correction

data:  c(table(churn_test_r$Churn)["True"], table(churn_train_r$Churn)["True"])
out of c(dim(churn_test_r)[1], dim(churn_train_r)[1])
X-squared = 0.034423, df = 1, p-value = 0.8528
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.02905977  0.02290158
sample estimates:
   prop 1    prop 2
0.1428571 0.1459362
```

p-value = 0.85 which is greater than 0.05, we fail to reject the null hypothesis of the test. We do not have sufficient evidence to say that the proportion of true churn records between the two data sets are different.

For the following exercises, work with the adult_ch6_training and adult_ch6_test data sets. Use R to solve each problem.

Python

In [56]:
```
adult_train_py = pd.read_csv("D:/2021-Spring-textbooks/ADS-502/Website Data Sets/
adult_test_py = pd.read_csv("D:/2021-Spring-textbooks/ADS-502/Website Data Sets/a

print(adult_train_py.head())
print(adult_test_py.head())
```

```
  Marital status Income  Cap_Gains_Losses
0  Never-married  <=50K           0.02174
1       Divorced  <=50K           0.00000
2        Married  <=50K           0.00000
3        Married  <=50K           0.00000
4        Married  <=50K           0.00000
  Marital status Income  Cap_Gains_Losses
0        Married  <=50K          0.000000
1        Married   >50K          0.051781
2  Never-married  <=50K          0.000000
3       Divorced   >50K          0.000000
4        Married   >50K          0.000000
```

R

In [57]:
```
%%R
adult_train_r <- read.csv(file = "D:/2021-Spring-textbooks/ADS-502/Website Data S
adult_test_r <- read.csv(file = "D:/2021-Spring-textbooks/ADS-502/Website Data Se

print(head(adult_train_r))
print(head(adult_test_r))
```

```
  Marital.status Income Cap_Gains_Losses
1  Never-married  <=50K          0.02174
2       Divorced  <=50K          0.00000
3        Married  <=50K          0.00000
4        Married  <=50K          0.00000
5        Married  <=50K          0.00000
6        Married   >50K          0.00000
  Marital.status Income Cap_Gains_Losses
1        Married  <=50K         0.000000
2        Married   >50K         0.051781
3  Never-married  <=50K         0.000000
4       Divorced   >50K         0.000000
5        Married   >50K         0.000000
6        Married  <=50K         0.000000
```

In [58]:
```
%%R
# Change categorical attributes to factors
adult_train_r$Income <- factor(adult_train_r$Income)
adult_train_r$Marital.status <- factor(adult_train_r$Marital.status)
```

23. Using the training data set, create a C5.0 model (Model 1) to predict a customer's Income using Marital Status and Capital Gains and Losses. Obtain the predicted responses.

Python

In [59]:
```
y = adult_train_py[['Income']]
y_test = adult_test_py[['Income']]
X = adult_train_py[['Marital status', 'Cap_Gains_Losses']]
X_test = adult_test_py[['Marital status', 'Cap_Gains_Losses']]
marital_dummy = pd.get_dummies(X['Marital status'])
marital_dummy_test = pd.get_dummies(X_test['Marital status'])
X = pd.concat((X[['Cap_Gains_Losses']], marital_dummy), axis = 1)
X_test = pd.concat((X_test[['Cap_Gains_Losses']], marital_dummy_test), axis = 1)
C5_py = DecisionTreeClassifier(criterion='entropy', min_samples_split=75, max_lea
C5_test_py = DecisionTreeClassifier(criterion='entropy', min_samples_split=75, ma
```

In [60]:
```
# Select a random sample
sample_01 = X.sample()
sample_01
```

Out[60]:

	Cap_Gains_Losses	Divorced	Married	Never-married	Separated	Widowed
5059	0.383838	0	1	0	0	0

In [61]:
```
# Prediction of the selected random sample
pred_single_C5_py = C5_py.predict(sample_01)
pred_single_C5_py
```

Out[61]:
```
array(['>50K'], dtype=object)
```

In [62]:
```
# Compare to its true value

y.iloc[sample_01.index[0]]
```

Out[62]:
```
Income    <=50K
Name: 5059, dtype: object
```

R

In [63]:
```
%%R
# Run the training data set through C5.0 to obtain Model 1. Save the result as C5
C5_r <- C5.0(Income ~ Marital.status + Cap_Gains_Losses, data = adult_train_r)
```

In [64]:
```
%%R
# Subset the predictor variables from the test data set into their own data frame
train.X <- subset(x = adult_train_r, select = c("Marital.status", "Cap_Gains_Loss

ypred <- predict(object = C5_r, newdata = train.X)

# Select a random sample for prediction
sample_r <- adult_train_r[sample(nrow(adult_train_r), size=1), ]
sample_r
```

```
     Marital.status Income Cap_Gains_Losses
4431  Never-married  <=50K                0
```

In [65]:
```
%%R

# Run sample prediction
test.sample_r <- subset(x = sample_r, select = c("Marital.status", "Cap_Gains_Los
ypred_sample <- predict(object = C5_r, newdata = test.sample_r)
ypred_sample
```

```
[1] <=50K
Levels: <=50K >50K
```

24. Evaluate Model 1 using the test data set. Construct a contingency table to compare the actual and predicted values of Income.

Python

In [66]:
```
y_true = adult_test_py[['Income']]
y_pred = C5_test_py.predict(X_test)
confusion_py = confusion_matrix(y_true, y_pred)
confusion_py
```

Out[66]:
```
array([[4627,   47],
       [1141,  340]], dtype=int64)
```

R

In [67]:
```
%%R

test.X <- subset(x = adult_test_r, select = c("Marital.status", "Cap_Gains_Losses
ypred_test <- predict(object = C5_r, newdata = test.X)

# Build a contingency table and compare the acutal Income in test data set with p

t1 <- table(adult_test_r$Income, ypred_test)
row.names(t1) <- c("Actual: 0", "Actual: 1")
colnames(t1) <- c("Predicted: 0", "Predicted: 1")
t1 <- addmargins(A = t1, FUN = list(Total = sum), quiet = TRUE)
t1
```

```
           ypred_test
            Predicted: 0 Predicted: 1 Total
  Actual: 0         4658           16  4674
  Actual: 1         1057          424  1481
  Total             5715          440  6155
```

25. For Model 1, recapitulate Table 7.4 from the text, calculating all of the model evaluation measures shown in the table. Call this table the Model Evaluation Table. Leave space for Model 2.

Python

In [68]:
```
TN_py = confusion_py[0,0]
FN_py = confusion_py[1,0]
FP_py = confusion_py[0,1]
TP_py = confusion_py[1,1]
TPN_py = TN_py + FN_py
TPP_py = FP_py + TP_py
TAN_py = TN_py + FP_py
TAP_py = FN_py + TP_py
GT_py = TN_py + FN_py + FP_py + TP_py
Precision_py = TP_py/TPP_py
Recall_py = TP_py/TAP_py
```

In [69]:
```
# Overall model cost and Profit per customer formulas
Cost_FP = 10
Cost_TP = -40
Overall_model_cost = FP_py*Cost_FP + TP_py*Cost_TP
Profit_per_customer = -Overall_model_cost/GT_py
```

In [70]:
```
column_names = ['Formula', 'Value']
row_names    = ['Accuracy', 'Error rate','Sensitivity','Specificity','Precision',
                'Overall Model Cost','Profit Per Customer']
matrix = np.reshape(('(TN+TP)/GT', round((TN_py+TP_py)/GT_py,4),'1-(TN+TP)/GT',ro
                     round(TP_py/TAP_py,4),'TN/TAN', round(TN_py/TAN_py,4),'TP/TP
                     '2*Precision*Recall/(Precision+Recall)',round(2*Precision_py
                     '5*Precision*Recall/((4+Precision)+Recall)',round(5*Precisio
                     '1.25*Precision*Recall/((0.25*Precision)+Recall)',
                     round(1.25*Precision_py*Recall_py/((0.25*Precision_py)+Recal
                    Overall_model_cost,'-Overall_model_cost/GT',round(Profit_per_
Model_Evaluation_Table_py = pd.DataFrame(matrix, columns=column_names, index=row_
Model_Evaluation_Table_py
```

Out[70]:

	Formula	Value
Accuracy	(TN+TP)/GT	0.807
Error rate	1-(TN+TP)/GT	0.193
Sensitivity	TP/TAP	0.2296
Specificity	TN/TAN	0.9899
Precision	TP/TPP	0.8786
F1	2*Precision*Recall/(Precision+Recall)	0.364
F2	5*Precision*Recall/((4+Precision)+Recall)	0.1974
F0.5	1.25*Precision*Recall/((0.25*Precision)+Recall)	0.5612
Overall Model Cost	FP*Cost_FP + TP*Cost_TP	-13130
Profit Per Customer	-Overall_model_cost/GT	2.1332

R

In [71]:
```
%%R
# Define elements in the Model Evaluation Table
TN_r <- t1[1,1]
FP_r <- t1[1,2]
FN_r <- t1[2,1]
TP_r <- t1[2,2]
TPN_r <- t1[3,1]
TPP_r <- t1[3,2]
TAN_r <- t1[1,3]
TAP_r <- t1[2,3]
GT_r <- t1[3,3]
Precision_r <- TP_r/TPP_r
Recall_r <- TP_r/TAP_r
```

In [72]:
```
%%R
# Overall model cost and Profit per customer formulas
Cost_FP = 10
Cost_TP = -40
Overall_model_cost_r = FP_r*Cost_FP + TP_r*Cost_TP
Profit_per_customer_r = -Overall_model_cost_r/GT_r
```

In [73]:
```
%%R
tab <- matrix(c('(TN+TP)/GT', round((TN_r+TP_r)/GT_r,4),'1-(TN+TP)/GT',round(1-(T
                'TN/TAN', round(TN_r/TAN_r,4),'TP/TPP',round(TP_r/TPP_r,4),'2*Pre
                round(2*Precision_r*Recall_r/(Precision_r+Recall_r),4), '5*Precis
                round(5*Precision_r*Recall_r/((4+Precision_r)+Recall_r),4),'1.25*
                round(1.25*Precision_r*Recall_r/((0.25*Precision_r)+Recall_r),4),
                 Overall_model_cost_r,'-Overall_model_cost/GT',round(Profit_per_c
colnames(tab) <- c('Formula','Value')
rownames(tab) <- c('Accuracy','Error rate','Sensitivity','Specificity','Precisior
                    'Profit Per Customer')
tab <- as.table(tab)
Model_Evaluation_Table_r <- tab
Model_Evaluation_Table_r
```

```
                    Formula                                 Value
Accuracy            (TN+TP)/GT                              0.8257
Error rate          1-(TN+TP)/GT                            0.1743
Sensitivity         TP/TAP                                  0.2863
Specificity         TN/TAN                                  0.9966
Precision           TP/TPP                                  0.9636
F1                  2*Precision*Recall/(Precision+Recall)   0.4414
F2                  5*Precision*Recall/((4+Precision)+Recall) 0.2627
F0.5                1.25*Precision*Recall/((0.25*Precision)+Recall) 0.6541
Overall Model Cost  FP*Cost_FP + TP*Cost_TP                 -16800
Profit Per Customer -Overall_model_cost/GT                  2.7295
```

26. Clearly and completely interpret each of the Model 1 evaluation measures from the Model Evaluation Table.

Accuracy (ACC) is calculated as the number of all correct predictions divided by the total number of the dataset. The best accuracy is 1.0, whereas the worst is 0.0.

Error rate (ERR) is calculated as the number of all incorrect predictions divided by the total number of the dataset. The best error rate is 0.0, whereas the worst is 1.0.

Sensitivity (Recall or True positive rate) is calculated as the number of correct positive predictions divided by the total number of positives. It is also called recall (REC) or true positive rate (TPR). The best sensitivity is 1.0, whereas the worst is 0.0.

Specificity (True negative rate) is calculated as the number of correct negative predictions divided by the total number of negatives. It is also called true negative rate (TNR). The best specificity is 1.0, whereas the worst is 0.0.

Precision (Positive predictive value) is calculated as the number of correct positive predictions divided by the total number of positive predictions. It is also called positive predictive value (PPV). The best precision is 1.0, whereas the worst is 0.0.

F1-score is the harmonic mean of the precision and recall. A perfect model has an F-score of 1. A factor in Fβ indicating how much more important recall is than precision. For example, if we consider recall to be twice as important as precision, we can set β to 2. The standard F-score is equivalent to setting β to 1.

27. Create a cost matrix, called the 3x cost matrix, that specifies a false positive is four times as bad as a false negative.

Python

In [74]:
```
# Cost matrix with FP = 4FN
cost_matrix_3x_py = {'<=50K':1, '>50K':4}
```

R

In [75]:
```
%%R
cost_matrix_3x_r = matrix(c(0,4,1,0), byrow = TRUE, ncol = 2 )
dimnames(cost_matrix_3x_r) <- list(c("<=50K",">50K"), c("<=50K",">50K"))
cost_matrix_3x_r
```

```
      <=50K >50K
<=50K     0    4
>50K      1    0
```

28. Using the training data set, build a C5.0 model (Model 2) to predict a customer's Income using Marital Status and Capital Gains and Losses, using the 3x cost matrix.

Python

In [76]:
```
C5_02_py = DecisionTreeClassifier(criterion='entropy', min_samples_split=75, max_
                                  class_weight=cost_matrix_3x_py).fit(X, y)

# Select a random customer from the training set
sample_02 = X.sample()
sample_02
```

Out[76]:

	Cap_Gains_Losses	Divorced	Married	Never-married	Separated	Widowed
18201	0.0	1	0	0	0	0

In [77]:
```
# Select a random customer from the training set
sample_02 = X.sample()
sample_02
```

Out[77]:

	Cap_Gains_Losses	Divorced	Married	Never-married	Separated	Widowed
12042	0.0	1	0	0	0	0

In [78]:
```
# Predict Income for the single random customer
pred_single_C5_02_py = C5_02_py.predict(sample_02)
pred_single_C5_02_py
```

Out[78]:
```
array(['<=50K'], dtype=object)
```

In [79]:
```
# Compare to the true value

y.iloc[sample_02.index[0]]
```

Out[79]:
```
Income    <=50K
Name: 12042, dtype: object
```

R

In [80]:
```
%%R
#Adding cost matrix to C5.0 model
C5.costs_r <- C5.0(Income ~ Marital.status + Cap_Gains_Losses, data = adult_trair
```

In [81]:
```
%%R
# Select a random sample for prediction
sample_02_r <- adult_train_r[sample(nrow(adult_train_r), size=1), ]
sample_02_r
```

```
      Marital.status Income Cap_Gains_Losses
10104        Married  <=50K                0
```

In [82]:
```
%%R
# Run sample prediction
test.sample_r <- subset(x = sample_02_r, select = c("Marital.status", "Cap_Gains_
ypred_sample <- predict(object = C5.costs_r, newdata = test.sample_r)
ypred_sample
```

```
[1] >50K
Levels: <=50K >50K
```

29. Evaluate your predictions from Model 2 using the actual response values from the test data set. Add Overall Model Cost and Profit per Customer to the Model Evaluation Table. Calculate all the measures from the Model Evaluation Table.

Python

In [83]:
```
# Contingency table with Model 2 using test data
y_true = adult_test_py[['Income']]
y_pred_02 = C5_02_py.predict(X_test)
confusion_02_py = confusion_matrix(y_true, y_pred_02)
confusion_02_py
```

Out[83]:
```
array([[3041, 1633],
       [ 149, 1332]], dtype=int64)
```

In [84]:
```
TN_02_py = confusion_02_py[0,0]
FN_02_py = confusion_02_py[1,0]
FP_02_py = confusion_02_py[0,1]
TP_02_py = confusion_02_py[1,1]
TPN_02_py = TN_02_py + FN_02_py
TPP_02_py = FP_02_py + TP_02_py
TAN_02_py = TN_02_py + FP_02_py
TAP_02_py = FN_02_py + TP_02_py
GT_02_py = TN_02_py + FN_02_py + FP_02_py + TP_02_py
Precision_02_py = TP_02_py/TPP_02_py
Recall_02_py = TP_02_py/TAP_02_py
```

In [85]:
```
# Overall model cost and Profit per customer formulas
Cost_FP = 10
Cost_TP = -40
Overall_model_cost_02 = FP_02_py*Cost_FP + TP_02_py*Cost_TP
Profit_per_customer_02 = -Overall_model_cost_02/GT_02_py
```

In [86]:
```
# Model Evaluation Table with Overall model cost and Profit per customer
column_names = ['Formula', 'Value']
row_names    = ['Accuracy', 'Error rate','Sensitivity','Specificity','Precision',
                'Overall Model Cost','Profit Per Customer']
matrix = np.reshape(('(TN+TP)/GT', round((TN_02_py+TP_02_py)/GT_02_py,4),'1-(TN+T
                     round(TP_02_py/TAP_02_py,4),'TN/TAN', round(TN_02_py/TAN_02_
                     '2*Precision*Recall/(Precision+Recall)',round(2*Precision_02
                     '5*Precision*Recall/((4+Precision)+Recall)',round(5*Precisio
                     '1.25*Precision*Recall/((0.25*Precision)+Recall)',
                     round(1.25*Precision_02_py*Recall_02_py/((0.25*Precision_02_
                    Overall_model_cost_02,'-Overall_model_cost/GT',round(Profit_p
Model_Evaluation_Table_02_py = pd.DataFrame(matrix, columns=column_names, index=r
Model_Evaluation_Table_02_py
```

Out[86]:

	Formula	Value
Accuracy	(TN+TP)/GT	0.7105
Error rate	1-(TN+TP)/GT	0.2895
Sensitivity	TP/TAP	0.8994
Specificity	TN/TAN	0.6506
Precision	TP/TPP	0.4492
F1	2*Precision*Recall/(Precision+Recall)	0.5992
F2	5*Precision*Recall/((4+Precision)+Recall)	0.3777
F0.5	1.25*Precision*Recall/((0.25*Precision)+Recall)	0.4992
Overall Model Cost	FP*Cost_FP + TP*Cost_TP	-36950
Profit Per Customer	-Overall_model_cost/GT	6.0032

R

In [87]:
```
%%R
ypred_test <- predict(object = C5.costs_r, newdata = test.X)

# Build a contingency table and compare the acutal Income in test data set with p

t2 <- table(adult_test_r$Income, ypred_test)
row.names(t2) <- c("Actual: 0", "Actual: 1")
colnames(t2) <- c("Predicted: 0", "Predicted: 1")
t2 <- addmargins(A = t2, FUN = list(Total = sum), quiet = TRUE)
t2
```

```
           ypred_test
            Predicted: 0 Predicted: 1 Total
  Actual: 0         3171         1503  4674
  Actual: 1          128         1353  1481
  Total             3299         2856  6155
```

In [88]:
```
%%R

# Define elements in the Model Evaluation Table 02
TN_02_r <- t2[1,1]
FP_02_r <- t2[1,2]
FN_02_r <- t2[2,1]
TP_02_r <- t2[2,2]
TPN_02_r <- t2[3,1]
TPP_02_r <- t2[3,2]
TAN_02_r <- t2[1,3]
TAP_02_r <- t2[2,3]
GT_02_r <- t2[3,3]
Precision_02_r <- TP_02_r/TPP_02_r
Recall_02_r <- TP_02_r/TAP_02_r
```

In [89]:
```
%%R
# Overall model cost and Profit per customer formulas
Cost_FP = 10
Cost_TP = -40
Overall_model_cost_02_r = FP_02_r*Cost_FP + TP_02_r*Cost_TP
Profit_per_customer_02_r = -Overall_model_cost_02_r/GT_02_r
```

In [90]:
```
%%R
tab2 <- matrix(c('(TN+TP)/GT', round((TN_02_r+TP_02_r)/GT_02_r,4),'1-(TN+TP)/GT',
                 'TP/TAP', round(TP_02_r/TAP_02_r,4),
                'TN/TAN', round(TN_02_r/TAN_02_r,4),'TP/TPP',round(TP_02_r/TPP_02
                round(2*Precision_02_r*Recall_02_r/(Precision_02_r+Recall_02_r),4
                round(5*Precision_02_r*Recall_02_r/((4+Precision_02_r)+Recall_02_
                round(1.25*Precision_02_r*Recall_02_r/((0.25*Precision_02_r)+Reca
                 Overall_model_cost_02_r,'-Overall_model_cost/GT',round(Profit_pe
colnames(tab2) <- c('Formula','Value')
rownames(tab2) <- c('Accuracy','Error rate','Sensitivity','Specificity','Precisio
                    'Profit Per Customer')
tab2 <- as.table(tab2)
Model_Evaluation_Table_02_r <- tab2
Model_Evaluation_Table_02_r
```

```
                    Formula                                      Value
Accuracy            (TN+TP)/GT                                   0.735
Error rate          1-(TN+TP)/GT                                 0.265
Sensitivity         TP/TAP                                       0.9136
Specificity         TN/TAN                                       0.6784
Precision           TP/TPP                                       0.4737
F1                  2*Precision*Recall/(Precision+Recall)        0.6239
F2                  5*Precision*Recall/((4+Precision)+Recall)    0.4017
F0.5                1.25*Precision*Recall/((0.25*Precision)+Recall) 0.5242
Overall Model Cost  FP*Cost_FP + TP*Cost_TP                      -39090
Profit Per Customer -Overall_model_cost/GT                       6.3509
```

30. Compare the evaluation measures from Model 1 and Model 2 using the 3x cost matrix. Discuss the strengths and weaknesses of each model.

Python

In [91]:
```
Model_Comparison_py = Model_Evaluation_Table_py.drop(['Formula'], axis=1)
```

In [92]:
```
Model_Comparison_py['Value of Model 2'] = Model_Evaluation_Table_02_py[['Value']]
```

In [93]:
```
Model_Comparison_py
```

Out[93]:

	Value	Value of Model 2
Accuracy	0.807	0.7105
Error rate	0.193	0.2895
Sensitivity	0.2296	0.8994
Specificity	0.9899	0.6506
Precision	0.8786	0.4492
F1	0.364	0.5992
F2	0.1974	0.3777
F0.5	0.5612	0.4992
Overall Model Cost	-13130	-36950
Profit Per Customer	2.1332	6.0032

Compare Model 1 to Model 2 we can see that Model 1 has higher accuracy and lower error rate while Model 2 scores higher on F-scores. Adding the unequal error costs makes Model 2 has decreased overall model cost and increased the Profit Per Customer by 181%. Because every true positive gives us $40, the models which tended to make more positive predictions did better. Sensitivity (recall), the proportion of all the positive responders that the model captured, thus turned out to be more important than specificity.Thus, accuracy is not the proper metric to compare models which have unequal error costs.

R

In [94]:
```
%%R
Model_Evaluation_Table_r <- Model_Evaluation_Table_r[,-1]
```

In [95]:
```
%%R
Model_Comparison_r <- cbind(Model_Evaluation_Table_r,Model_Evaluation_Table_02_r|
```

In [96]:
```
%%R
Model_Comparison_r
```

```
                    Model_Evaluation_Table_r
Accuracy            "0.8257"                  "0.735"
Error rate          "0.1743"                  "0.265"
Sensitivity         "0.2863"                  "0.9136"
Specificity         "0.9966"                  "0.6784"
Precision           "0.9636"                  "0.4737"
F1                  "0.4414"                  "0.6239"
F2                  "0.2627"                  "0.4017"
F0.5                "0.6541"                  "0.5242"
Overall Model Cost  "-16800"                  "-39090"
Profit Per Customer "2.7295"                  "6.3509"
```

Same scenario with Python.

For the following exercises, work with the framingham_nb_training and framingham_nb_test data sets. Use either Python or R to solve each problem.

31. Run the Naïve Bayes classifier to classify persons as living or dead based on sex and education.

Python

In [97]:
```
framingham_nb_train_py = pd.read_csv("D:/2021-Spring-textbooks/ADS-502/Website Da
framingham_nb_test_py = pd.read_csv("D:/2021-Spring-textbooks/ADS-502/Website Dat

framingham_nb_train_py.head()
```

Out[97]:

	Sex	Educ	Death
0	2	3	0
1	2	2	0
2	1	1	0
3	2	1	0
4	2	1	0

In [98]:
```
# Check for if the true repsonse has a good proportion
framingham_nb_train_py['Death'].sum()*100 / framingham_nb_train_py.shape[0]
```

Out[98]:
```
55.0
```

True response rate is 55%, no need to rebalance the data. Move on.

In [99]:
```
# Split the predictors and target in the training set
X_framingham_train_py = framingham_nb_train_py[['Sex', 'Educ']]
y_framingham_train_py = framingham_nb_train_py['Death']
```

In [100]:
```
# Convert categorical data into dummy variables for both 'Sex' and 'Educ' for the
X_framingham_train_processed_py = pd.DataFrame()

for var in X_framingham_train_py.columns:
    dummies = pd.get_dummies(X_framingham_train_py[var])
    X_framingham_train_processed_py = pd.concat([X_framingham_train_processed_py,
```

In [101]:
```
# Run Naive Bayers algorithm using training set
nb = MultinomialNB().fit(X_framingham_train_processed_py, y_framingham_train_py)
```

In [102]:
```
# Split the predictors and target in the test set
X_framingham_test_py = framingham_nb_test_py[['Sex', 'Educ']]
y_framingham_test_py = framingham_nb_test_py['Death']
```

In [103]:
```
# Convert categorical data into dummy variables for both 'Sex' and 'Educ' for the
X_framingham_test_processed_py = pd.DataFrame()

for var in X_framingham_test_py.columns:
    dummies = pd.get_dummies(X_framingham_test_py[var])
    X_framingham_test_processed_py = pd.concat([X_framingham_test_processed_py, d
```

In [104]:
```
predictions_nb_py = nb.predict(X_framingham_test_processed_py)
predictions_nb_py
```

Out[104]:
```
array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 0], dtype=int64)
```

R

In [105]:
```
%%R

framingham_nb_train_r <- read.csv(file = "D:/2021-Spring-textbooks/ADS-502/Websit
framingham_nb_test_r <- read.csv(file = "D:/2021-Spring-textbooks/ADS-502/Website

head(framingham_nb_train_r)
```

```
  Sex Educ Death
1   2    3     0
2   2    2     0
3   1    1     0
4   2    1     0
5   2    1     0
6   2    3     0
```

In [106]:
```
%%R

cols = c('Sex', 'Educ')
framingham_nb_train_r[, cols] <- lapply(framingham_nb_train_r[, cols], as.factor)

# Or we could use this one by one
#framingham_nb_train_r$Sex <- factor(framingham_nb_train_r$Sex)
#framingham_nb_train_r$Educ <- factor(framingham_nb_train_r$Educ)
```

In [107]:
```
%%R
# Run the model

nb <- naiveBayes(formula = Death ~ Sex + Educ, data = framingham_nb_train_r)
```

In [108]:
```
%%R

# Process the test set the same way as training set
framingham_nb_test_r[, cols] <- lapply(framingham_nb_test_r[, cols], as.factor)
```

In [109]:
```
%%R
predictions_nb_r <- predict(nb, framingham_nb_test_r)
predictions_nb_r
```

```
   [1] 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0
0
  [38] 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 0 0 1 1 1 1
1
  [75] 1 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 1
0
 [112] 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 0 1
1
 [149] 0 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1
1
 [186] 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1
0
 [223] 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 1 0
0
 [260] 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1
1
 [297] 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0
 [334] 0 0 0 1 1 1 1 0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1
0
 [371] 1 1 0 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0
1
 [408] 1 0 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1
1
 [445] 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0
1
 [482] 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1
1
 [519] 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0
1
 [556] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
1
 [593] 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 1 0 1 1 0
1
 [630] 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1
 [667] 1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1
1
 [704] 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1
1
 [741] 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1
1
 [778] 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 0 0
1
 [815] 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1
1
 [852] 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1
 [889] 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1
1
 [926] 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1
1
 [963] 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
```

```
0
[1000] 0
Levels: 0 1
```

32. Evaluate the Naïve Bayes model on the framingham_nb_test data set. Display the results in a contingency table. Edit the row and column names of the table to make the table more readable. Include a total row and column.

Python

In [110]:
```python
cm = confusion_matrix(y_framingham_test_py, predictions_nb_py)
```

In [111]:
```python
cm
```

Out[111]:
```
array([[203, 322],
       [105, 370]], dtype=int64)
```

In [112]:
```python
TN_py = cm[0][0]
FP_py = cm[0][1]
FN_py = cm[1][0]
TP_py = cm[1][1]

print('TN: ', TN_py,
      '\nFP: ', FP_py,
      '\nFN: ', FN_py,
      '\nTP: ', TP_py)
```

```
TN:  203
FP:  322
FN:  105
TP:  370
```

In [113]:
```python
col_names = ['Predicted:0', 'Predicted:1', 'Total']
row_names =['Actual:0', 'Actual:1','Total']
matrix = np.reshape((TN_py,FP_py,TN_py+FP_py,FN_py,TP_py,FN_py+TP_py,TN_py+FN_py,
cm_py = pd.DataFrame(matrix,columns=col_names, index=row_names)
cm_py
```

Out[113]:

	Predicted:0	Predicted:1	Total
Actual:0	203	322	525
Actual:1	105	370	475
Total	308	692	1000

R

In [114]:
```
%%R
# Build a contingency table and compare the acutal Income in test data set with p

t3 <- table(framingham_nb_test_r$Death, predictions_nb_r)
row.names(t3) <- c("Actual: 0", "Actual: 1")
colnames(t3) <- c("Predicted: 0", "Predicted: 1")
t3 <- addmargins(A = t3, FUN = list(Total = sum), quiet = TRUE)
```

In [115]:
```
%%R
t3
```

```
           predictions_nb_r
            Predicted: 0 Predicted: 1 Total
  Actual: 0          167          283   450
  Actual: 1          122          428   550
  Total              289          711  1000
```

In [116]:
```
%%R
TN_r = t3[1:1,1:1]
FP_r = t3[1:1,2:2]
FN_r = t3[2:2,1:1]
TP_r = t3[2:2,2:2]
```

In [117]:
```
%%R
cat(paste("TN:", FP_r,"\nFP:", FP_r,"\nFN:", FN_r,"\nTP:", TP_r))
```

```
TN: 283
FP: 283
FN: 122
TP: 428
```

33. According to your table in the previous exercise, find the following values for the Naïve Bayes model:

a. Accuracy

b. Error rate

Python

In [118]:
```
print("Accuracy for NB model is: ",round((TN_py+TP_py)/(TN_py+TP_py+FN_py+FP_py),
     '\nError Rate for NB model is: ', round(1-(TN_py+TP_py)/(TN_py+TP_py+FN_py+F
```

```
Accuracy for NB model is:  0.573
Error Rate for NB model is:  0.427
```

R

In [119]:
```
%%R
cat(paste("Accuracy for NB model is: ", round((TN_r+TP_r)/(TN_r+TP_r+FN_r+FP_r),4
          "\nError Rate for NB model is: ", round(1-(TN_r+TP_r)/(TN_r+TP_r+FN_r+F
```

```
Accuracy for NB model is:  0.595
Error Rate for NB model is:  0.405
```

34. According to your contingency table, find the following values for the Naïve Bayes model:

a. How often it correctly classifies dead persons.

b. How often it correctly classifies living persons.

Python

In [120]:
```
print(round(TN_py*100/(TN_py+FP_py),2),"% chance of correctly classifies dead per
```

```
38.67 % chance of correctly classifies dead persons. This is also the Specifici
ty
```

In [121]:
```
print(round(TP_py*100/(TP_py+FN_py),2),"% chance of correctly classifies living p
```

```
77.89 % chance of correctly classifies living persons. This is also the Sensiti
vity
```

R

In [122]:
```
%%R
cat(round(TN_r*100/(TN_r+FP_r),2),"% chance of correctly classifies dead persons.
```

```
37.11 % chance of correctly classifies dead persons. This is also the Specifici
ty
```

In [123]:
```
%%R
cat(round(TP_r*100/(TP_r+FN_r),2),"% chance of correctly classifies living persor
```

```
77.82 % chance of correctly classifies living persons. This is also the Sensiti
vity
```