In [20]:
```
# Ian Schenck
# Victor Castellanos

import pandas as pd
import numpy as np
from pathlib import Path
import math
```

In [21]:
```
# load a file that uses , as delimiter
def load_file(path, names):
    if not path.is_file():
        raise FileNotFoundError(str(path))

    data = pd.read_csv(path, sep=",", names=names, header=None)
    return data

# load data given
def load_dfs():
    cols = ["PregnanciesNumber", "GlucosePlasma", "BloodPressureDiast
olic", "SkinThicknessTriceps",
           "Insulin2Hour", "BMI", "DiabetesPedigreeFunction", "Age",
"OutcomeClass"]
    path = Path.cwd() / "data"
    diabetes_file = path / "pima-indians-diabetes.data.csv"
    train_file = path / "train.csv"
    test_file = path / "test.csv"

    diabetes_data = load_file(diabetes_file, cols)
    train_data = load_file(train_file, cols)
    test_data = load_file(test_file, cols)
    return diabetes_data, train_data, test_data

# calculate the mean and standard deviation for each column, separate
d by class
def mean_std_by_class(data):
    data_by_class = data.groupby('OutcomeClass')
    mean = data_by_class.mean()
    std = data_by_class.std()
    false_mean = mean[std.index == 0.0].values[0]
    false_std = std[std.index == 0.0].values[0]
    true_mean = mean[std.index == 1.0].values[0]
    true_std = std[std.index == 1.0].values[0]
    return false_mean, false_std, true_mean, true_std

# calculate normal distribution likelihood
def norm_dist(data, mean, std):
    variance = std**2
    denominator = (2 * math.pi* variance)**(.5)
    numerator = np.exp(-(data - mean)**2 / (2 * variance))
    return numerator / denominator

# calculate class probabilities
def class_probs(data):
    n_false = train_data['OutcomeClass'][train_data['OutcomeClass'] =
= 0].count()
    n_true = train_data['OutcomeClass'][train_data['OutcomeClass'] ==
1].count()
    n_total = train_data['OutcomeClass'].count()

    p_false = n_false / n_total
    p_true = n_true / n_total

    return p_false, p_true
```

In [22]:
```
# load data
diabetes_data, train_data, test_data = load_dfs()

# calculate class probabilities
p_false, p_true = class_probs(train_data)

# calculate mean and std by class
false_mean, false_std, true_mean, true_std = mean_std_by_class(train_
data)

test_data_no_outcome = test_data.drop('OutcomeClass', axis=1)

# calculate probability for false
false_norm = norm_dist(test_data_no_outcome, false_mean, false_std)
false_norm = false_norm.prod(axis=1) * p_false

# calculate probability for true
true_norm = norm_dist(test_data_no_outcome, true_mean, true_std)
true_norm = true_norm.prod(axis=1) * p_true

# assign a predicted outcome to each patient
norm = pd.concat([false_norm, true_norm], axis=1)
norm['diabetes_predicted'] = np.where(norm[1] > norm[0], 1.0, 0.0)
# compare predicted outcome to actual outcome and label as true or fa
lse
merged = pd.concat([norm['diabetes_predicted'], test_data['OutcomeCla
ss']], axis=1)
merged['accurate'] = np.where(merged['diabetes_predicted'] == merged[
'OutcomeClass'], True, False)
```

In [18]:
```
# calculate accuracy
accuracy = merged.mean()['accurate']
accuracy
```

Out[18]:
```
0.7480314960629921
```

In [23]:
```
#adding column true negative, false negative, true positive, and fals
e positive and labeling with 1 if label applies
merged['TN'] = np.where((merged['diabetes_predicted']==0) & (merged[
'OutcomeClass']==0),1,0)
merged['FN'] = np.where((merged['diabetes_predicted']==0) & (merged[
'OutcomeClass']==1),1,0)
merged['FP'] = np.where((merged['diabetes_predicted']==1) & (merged[
'OutcomeClass']==0),1,0)
merged['TP'] = np.where((merged['diabetes_predicted']==1) & (merged[
'OutcomeClass']==1),1,0)


#making dataFrame with TN, FN, FP, TP only columns and find the sum o
f those columns
confusion = merged.drop(['OutcomeClass','diabetes_predicted','accurat
e'],axis=1)
confusion = confusion.sum(axis=0)
#confusion

#getting values from dataframe
TN = confusion[0]
FN = confusion[1]
FP = confusion[2]
TP = confusion[3]
#calculating accuracy, error, senstivity and specificity
#accuracy is the percent of times that we have predicted correct
accuracy = (TP + TN)/(TP + FP + TN + FN)
#error is the percent of times that we have predicted incorrect
error = (FP + FN)/(TP + FP + TN + FN)
#sensitivity tells us for each time that we predicted positive how ma
ny of those times were correct in this case 61.7 percent
sensitivity = (TP)/(FN + TP)
#specificity tells us for each time that we predicted negative how ma
ny of those times were we correct, in this case 82.5percent
specificity = TN/(TN + FP)

#making a datafram to dispaly confusion matrix
displayCMatrix = [[TN,FP],[FN,TP]]
displayCMatrix = pd.DataFrame(displayCMatrix, columns = ['Predicted:
 0','Predicted: 1'], index= ['Actual: 0', 'Actual: 1'])

print(displayCMatrix)
print()
print("accuracy:", accuracy)
print("error:", error)
print("sensitivity:", sensitivity)
print("specificity:", specificity)
```

```
           Predicted: 0  Predicted: 1
Actual: 0           132            28
Actual: 1            36            58

accuracy: 0.7480314960629921
error: 0.25196850393700787
sensitivity: 0.6170212765957447
specificity: 0.825
```

Our interpretation of these values is that sensitivity is more important than specificity in this particular case. If someone does have diabetes, it could be life threatening for them to go undiagnoses. So the fact that our sensitivity is much less than specificity is a negative aspect of this classifier.