## Declare feature vector and target variable

```
X = cc_df.drop(['Biopsy'], axis=1)
y = cc_df['Biopsy']
```

## Split data into separate training and test set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
X_train.shape, X_test.shape
```

```
((668, 35), (167, 35))
```

```
y_train.shape
```

```
(668,)
```

## Engineering outliers

```
def max_value(df3, variable, top):
    return np.where(df3[variable]>top, top, df3[variable])

for df3 in [X_train, X_test]:
    df3['Age'] = max_value(df3, 'Age', 48.5)
    df3['Number_of_sexual_partners'] = max_value(df3, 'Number_of_sexual_partners', 44.5)
    df3['First_sexual_intercourse'] = max_value(df3, 'First_sexual_intercourse', 22.5)
    df3['Num_of_pregnancies'] = max_value(df3, 'Num_of_pregnancies', 6)
```

```
X_train.Age.max(), X_test.Age.max()
```

```
(48.5, 48.5)
```

```
X_train.Number_of_sexual_partners.max(), X_test.Number_of_sexual_partners.max()
```

```
(28.0, 5.0)
```

```
X_train.First_sexual_intercourse.max(), X_test.First_sexual_intercourse.max()
```

```
(22.5, 22.5)
```

```
X_train.Num_of_pregnancies.max(), X_test.Num_of_pregnancies.max()
```

```
(6.0, 6.0)
```

## Feature Scaling

```
X_train.describe()
```

| | Age | Number_of_sexual_partners | First_sexual_intercourse | Num_of_pregnanci |
|---|---|---|---|---|
| count | 668.000000 | 668.000000 | 668.000000 | 668.0000 |
| mean | 26.738024 | 2.600299 | 16.855539 | 2.2320 |
| std | 7.613179 | 1.758373 | 2.289344 | 1.2824 |
| min | 13.000000 | 1.000000 | 10.000000 | 0.0000 |
| 25% | 21.000000 | 2.000000 | 15.000000 | 1.0000 |
| 50% | 26.000000 | 2.000000 | 17.000000 | 2.0000 |
| 75% | 32.000000 | 3.000000 | 18.000000 | 3.0000 |
| max | 48.500000 | 28.000000 | 22.500000 | 6.0000 |

8 rows × 35 columns

```
cols = X_train.columns
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_test)

X_test = scaler.transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns=[cols])
```

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

## Model Training

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (668, 35)
y_train shape: (668,)
```

```
from sklearn.linear_model import LogisticRegression

logreg= LogisticRegression(solver='liblinear', random_state=0)

logreg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

## Predict results

```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
logreg.predict_proba(X_test)[:,0] # probabilities of getting no biopsy
```

```
array([0.91992733, 0.98901255, 0.99213392, 0.99338499, 0.96350935,
       0.98492135, 0.68396716, 0.99280932, 0.98619543, 0.9800177 ,
       0.96957308, 0.9884378 , 0.98259612, 0.9885742 , 0.46033742,
       0.9812865 , 0.99320811, 0.98325562, 0.98527004, 0.97930573,
       0.97639997, 0.98044322, 0.91443637, 0.98776232, 0.98894744,
       0.95692314, 0.98609478, 0.98749521, 0.99896489, 0.27489498,
       0.96384501, 0.98325726, 0.88303876, 0.99172243, 0.93884554,
       0.98294622, 0.98942283, 0.98886892, 0.99482286, 0.98490239,
       0.98256677, 0.98364479, 0.9813101 , 0.98599757, 0.99631075,
       0.9879477 , 0.52490908, 0.9746085 , 0.9904325 , 0.98865874,
       0.98557267, 0.98447354, 0.94496033, 0.98596821, 0.99735405,
       0.91022135, 0.97966427, 0.98364959, 0.9874529 , 0.9843784 ,
       0.98908174, 0.96781357, 0.99008667, 0.98170679, 0.98179538,
       0.99195909, 0.97511538, 0.96864293, 0.98391754, 0.98073402,
       0.46809188, 0.98836638, 0.9768906 , 0.97928647, 0.95045374,
       0.9878481 , 0.98824817, 0.57344977, 0.98684004, 0.9918064 ,
       0.98865707, 0.45668094, 0.98791917, 0.9893904 , 0.99117006,
       0.97200749, 0.98954389, 0.98306286, 0.98920591, 0.98310232,
       0.98867978, 0.97865247, 0.97546575, 0.98849014, 0.98212838,
       0.98356008, 0.98980109, 0.98849196, 0.99568845, 0.98961769,
       0.21202714, 0.9654617 , 0.99136161, 0.99135604, 0.98278209,
       0.98105385, 0.415899  , 0.99049651, 0.98217583, 0.97771181,
       0.98954869, 0.99218039, 0.97364164, 0.98029408, 0.27568103,
       0.98888715, 0.25541798, 0.97489767, 0.98355687, 0.99334744,
       0.99244108, 0.99356519, 0.98292325, 0.98145791, 0.98705342,
       0.98095867, 0.98549552, 0.98484295, 0.99117503, 0.98576044,
       0.98715349, 0.42116867, 0.99606284, 0.98257991, 0.99234688,
       0.98484498, 0.26332961, 0.98598453, 0.99339527, 0.98658895,
       0.98471595, 0.98409526, 0.99176459, 0.99494561, 0.99692705,
       0.98928807, 0.98723822, 0.99027939, 0.99476763, 0.98940565,
       0.98304047, 0.96238851, 0.99265516, 0.99630946, 0.98989316,
       0.99517441, 0.98481641, 0.51253167, 0.99027703, 0.60745069,
       0.98894497, 0.97933178, 0.98356808, 0.98392162, 0.97376315,
       0.9851037 , 0.9902956 ])
```

```
logreg.predict_proba(X_test)[:,1] # probabilities of getting biopsy
```

```
array([0.08007267, 0.01098745, 0.00786608, 0.00661501, 0.03649065,
       0.01507865, 0.31603284, 0.00719068, 0.01380457, 0.0199823 ,
       0.03042692, 0.0115622 , 0.01740388, 0.0114258 , 0.53966258,
       0.0187135 , 0.00679189, 0.01674438, 0.01472996, 0.02069427,
       0.02360003, 0.01955678, 0.08556363, 0.01223768, 0.01105256,
       0.04307686, 0.01390522, 0.01250479, 0.00103511, 0.72510502,
       0.03615499, 0.01674274, 0.11696124, 0.00827757, 0.06115446,
       0.01705378, 0.01057717, 0.01113108, 0.00517714, 0.01509761,
       0.01743323, 0.01635521, 0.0186899 , 0.01400243, 0.00368925,
       0.0120523 , 0.47509092, 0.0253915 , 0.0095675 , 0.01134126,
       0.01442733, 0.01552646, 0.05503967, 0.01403179, 0.00264595,
       0.08977865, 0.02033573, 0.01635041, 0.0125471 , 0.0156216 ,
       0.01091826, 0.03218643, 0.00991333, 0.01829321, 0.01820462,
       0.00804091, 0.02488462, 0.03135707, 0.01608246, 0.01926598,
       0.53190812, 0.01163362, 0.0231094 , 0.02071353, 0.04954626,
       0.0121519 , 0.01175183, 0.42655023, 0.01315996, 0.0081936 ,
       0.01134293, 0.54331906, 0.01208083, 0.0106096 , 0.00882994,
       0.02799251, 0.01045611, 0.01693714, 0.01079409, 0.01689768,
       0.01132022, 0.02134753, 0.02453425, 0.01150986, 0.01787162,
       0.01643992, 0.01019891, 0.01150804, 0.00431155, 0.01038231,
       0.78797286, 0.0345383 , 0.00863839, 0.00864396, 0.01721791,
       0.01894615, 0.584101  , 0.00950349, 0.01782417, 0.02228819,
       0.01045131, 0.00781961, 0.02635836, 0.01970592, 0.72431897,
       0.01111285, 0.74458202, 0.02510233, 0.01644313, 0.00665256,
       0.00755892, 0.00643481, 0.01707675, 0.01854209, 0.01294658,
       0.01904133, 0.01450448, 0.01515705, 0.00882497, 0.01423956,
       0.01284651, 0.57883133, 0.00393716, 0.01742009, 0.00765312,
       0.01515502, 0.73667039, 0.01401547, 0.00660473, 0.01341105,
       0.01528405, 0.01590474, 0.00823541, 0.00505439, 0.00307295,
       0.01071193, 0.01276178, 0.00972061, 0.00523237, 0.01059435,
       0.01695953, 0.03761149, 0.00734484, 0.00369054, 0.01010684,
       0.00482559, 0.01518359, 0.48746833, 0.00972297, 0.39254931,
       0.01105503, 0.02066822, 0.01643192, 0.01607838, 0.02623685,
       0.0148963 , 0.0097044 ])
```

## Check accuracy score

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.9401
```

## Compare the train-set and test-set accuracy

```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0])
```

```
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.9671
```

## Check for overfitting and underfitting

```
print('Training set score: {:.4f}'.format(logreg.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.9671
Test set score: 0.9401
```

```
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)

logreg100.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.9746
Test set score: 0.9281
```

```
logreg001 = LogisticRegression(C=0.01, solver='liblinear', random_state=0)

logreg001.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(C=0.01, random_state=0, solver='liblinear')
```

```
print('Training set score: {:.4f}'.format(logreg001.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg001.score(X_test, y_test)))
```

```
Training set score: 0.9341
Test set score: 0.9401
```

## Compare model accuracy with null accuracy

```
y_test.value_counts()
```

```
Biopsy
0    157
1     10
Name: count, dtype: int64
```

```
null_accuracy = (157/(157+10))

print('Null accuracy score: {0:0.4f}'.format(null_accuracy))
```

```
Null accuracy score: 0.9401
```

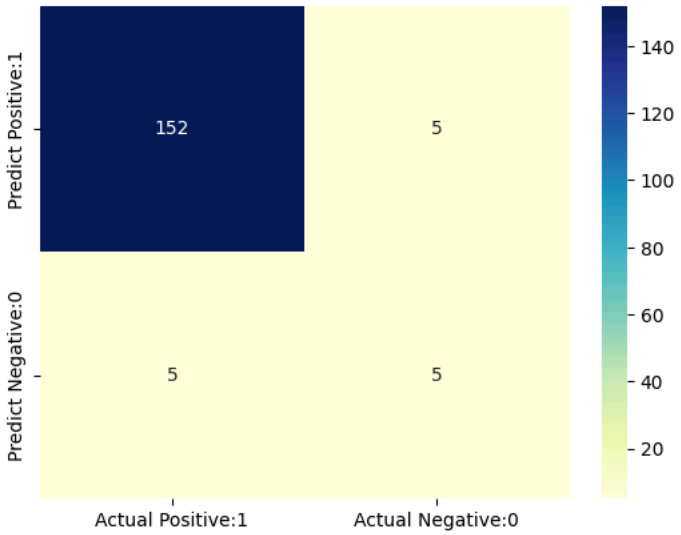
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[152   5]
 [  5   5]]

True Positives(TP) =  152

True Negatives(TN) =  5

False Positives(FP) =  5

False Negatives(FN) =  5
```

## After creating confusion matrix, it could be seen that out of 167 results, thee are only 5 results each for Type 1 and 2 errors.

```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                  index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```




## Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       157
           1       0.50      0.50      0.50        10

    accuracy                           0.94       167
   macro avg       0.73      0.73      0.73       167
weighted avg       0.94      0.94      0.94       167
```

## Classification Accuracy

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]


classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.9401
```

## Classification Error

```
classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0599
```

## Precision

```
precision =  TP / float(TP+FP)

print('Precision: {0:0.4f}'.format(precision))
```

```
Precision: 0.9682
```

## Recall

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9682
```

## True Positive Rate

```
true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.9682
```

## Specificity

```
specificity = TN / (TN+FP)
print('Specifity : {0:0.4f}'.format(specificity))
```

```
Specifity : 0.5000
```

## Adjusting the threshold level

```
y_pred_prob = logreg.predict_proba(X_test)[0:10]

y_pred_prob
```

```
array([[0.91992733, 0.08007267],
       [0.98901255, 0.01098745],
       [0.99213392, 0.00786608],
       [0.99338499, 0.00661501],
       [0.96350935, 0.03649065],
       [0.98492135, 0.01507865],
       [0.68396716, 0.31603284],
       [0.99280932, 0.00719068],
       [0.98619543, 0.01380457],
       [0.9800177 , 0.0199823 ]])
```

```
y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Probability of no Biopsy', 'Probability of a Biopsy'])

y_pred_prob_df
```

| | Probability of no Biopsy | Probability of a Biopsy |
|---|---|---|
| 0 | 0.919927 | 0.080073 |
| 1 | 0.989013 | 0.010987 |
| 2 | 0.992134 | 0.007866 |
| 3 | 0.993385 | 0.006615 |
| 4 | 0.963509 | 0.036491 |
| 5 | 0.984921 | 0.015079 |
| 6 | 0.683967 | 0.316033 |
| 7 | 0.992809 | 0.007191 |
| 8 | 0.986195 | 0.013805 |
| 9 | 0.980018 | 0.019982 |

Next steps: View recommended plots

```
logreg.predict_proba(X_test)[0:10, 1]
```

```
array([0.08007267, 0.01098745, 0.00786608, 0.00661501, 0.03649065,
       0.01507865, 0.31603284, 0.00719068, 0.01380457, 0.0199823 ])
```

```
y_pred1 = logreg.predict_proba(X_test)[:, 1]
```

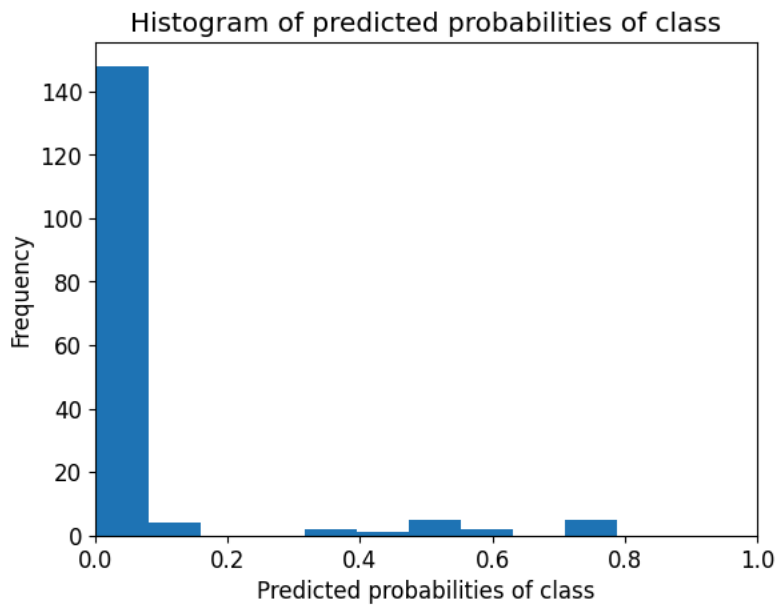
```
from sklearn.preprocessing import binarize

for i in range(1,5):
    cm1=0

    y_pred1 = logreg.predict_proba(X_test)[:,1]

    y_pred1 = y_pred1.reshape(-1,1)

    y_pred2 = binarize(y_pred1, threshold=(i/10))

    cm1 = confusion_matrix(y_test, y_pred2)
    print ('With',i/10, 'threshold the Confusion Matrix is ','\n\n', cm1, '\n\n',
          'with', cm1[0,0]+cm1[1,1], 'correct predictions, ', '\n\n',
          cm1[0,1], 'Type I errors( False Positives), ','\n\n',
          cm1[1,0], 'Type II errors( False Negatives), ','\n\n',
          'Accuracy score: ', (accuracy_score(y_test, y_pred2)), '\n\n',
          'Sensitivity: ',cm1[1,1]/(float(cm1[1,1]+cm1[1,0])), '\n\n',
          'Specificity: ',cm1[0,0]/(float(cm1[0,0]+cm1[0,1])), '\n\n',
          '====================================================', '\n\n')
```

```
 [[149   8]
 [  3   7]]

 with 156 correct predictions,

 8 Type I errors( False Positives),

 3 Type II errors( False Negatives),

 Accuracy score:  0.9341317365269461

 Sensitivity:  0.7

 Specificity:  0.9490445859872612

 ====================================================


With 0.3 threshold the Confusion Matrix is

 [[149   8]
 [  3   7]]

 with 156 correct predictions,

 8 Type I errors( False Positives),
```

```
Accuracy score:  0.9341317365269461

Sensitivity:  0.6

Specificity:  0.9554140127388535

=====================================================
```

```
plt.rcParams['font.size'] = 12

plt.hist(y_pred1, bins = 10)

plt.title('Histogram of predicted probabilities of class')

plt.xlim(0,1)

plt.xlabel('Predicted probabilities of class')
plt.ylabel('Frequency')
```

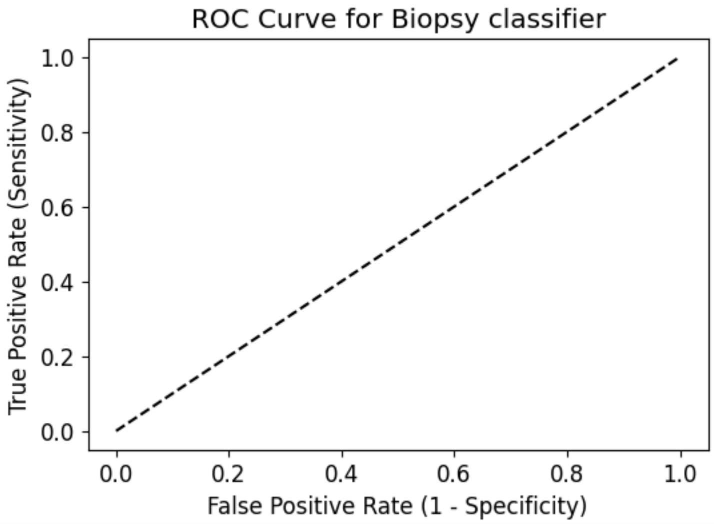
```
Text(0, 0.5, 'Frequency')
```




## ROC Curve

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred1, pos_label = 'Yes')

plt.figure(figsize=(6,4))

plt.plot(fpr,tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--')

plt.rcParams['font.size'] = 12

plt.title('ROC Curve for Biopsy classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_ranking.py:1029: UndefinedMetricWarning: No positive samples in y_true, true pc
  warnings.warn(
Text(0, 0.5, 'True Positive Rate (Sensitivity)')
```




No positive samples in y_true so there is no curve.

## ROC-AUC

```
from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred1)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.9153
```

## Cross validated ROC-AUC

```
from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(logreg, X_train, y_train, cv=5, scoring='roc_auc').mean()

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9434
```

## k-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(logreg, X_train, y_train, cv=5, scoring = 'accuracy')

print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.95522388 0.94776119 0.94776119 0.96240602 0.94736842]
```

```
print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.9521
```

## Hyperparameter Optimization using GridSearch CV

```
from sklearn.model_selection import GridSearchCV

parameters = [{'penalty':[']1',']2']},
          {'C':[1, 10, 100, 1000]}]

grid_search = GridSearchCV(estimator = logreg,
               param_grid = parameters,
               scoring = 'accuracy',
               cv = 5,
               verbose=0)

grid_search.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1160, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter of LogisticRegression must be a str among {'l1', 'elastic

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1160, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter of LogisticRegression must be a str among {'l1', 'elastic

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-fini
  warnings.warn(
```

▸ **GridSearchCV**

▸ **estimator: LogisticRegression**

▸ LogisticRegression

```
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))

print('Parameters that give the best results :','\n\n', (grid_search.best_params_))

print('\n\nEstimator that was chosen by the search :', '\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.9551


Parameters that give the best results :

 {'C': 10}


Estimator that was chosen by the search :

 LogisticRegression(C=10, random_state=0, solver='liblinear')
```

```
print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
GridSearch CV score on test set: 0.9281
```