In [1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv("breast cancer data.csv")
```

In [3]:
```
data.head(10)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
5	843786	M	12.45	15.70	82.57	477.1	
6	844359	M	18.25	19.98	119.60	1040.0	
7	84458202	M	13.71	20.83	90.20	577.9	
8	844981	M	13.00	21.82	87.50	519.8	
9	84501001	M	12.46	24.04	83.97	475.9	

10 rows × 33 columns

In [4]:
```
data.drop(['Unnamed: 32',"id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
```

In [7]:
```
x_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   radius_mean              569 non-null    float64
 1   texture_mean             569 non-null    float64
 2   perimeter_mean           569 non-null    float64
 3   area_mean                569 non-null    float64
 4   smoothness_mean          569 non-null    float64
 5   compactness_mean         569 non-null    float64
 6   concavity_mean           569 non-null    float64
 7   concave points_mean      569 non-null    float64
 8   symmetry_mean            569 non-null    float64
 9   fractal_dimension_mean   569 non-null    float64
 10  radius_se                569 non-null    float64
 11  texture_se               569 non-null    float64
 12  perimeter_se             569 non-null    float64
 13  area_se                  569 non-null    float64
 14  smoothness_se            569 non-null    float64
 15  compactness_se           569 non-null    float64
 16  concavity_se             569 non-null    float64
 17  concave points_se        569 non-null    float64
 18  symmetry_se              569 non-null    float64
 19  fractal_dimension_se     569 non-null    float64
 20  radius_worst             569 non-null    float64
 21  texture_worst            569 non-null    float64
 22  perimeter_worst          569 non-null    float64
 23  area_worst               569 non-null    float64
 24  smoothness_worst         569 non-null    float64
 25  compactness_worst        569 non-null    float64
 26  concavity_worst          569 non-null    float64
 27  concave points_worst     569 non-null    float64
 28  symmetry_worst           569 non-null    float64
 29  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [9]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
```

In [10]:
```
x_scaled=scaler.fit_transform(x_data)
```

In [12]:
```
x_scaled
```

Out[12]:
```
array([[0.52103744, 0.0226581 , 0.54598853, ..., 0.91202749, 0.59846245,
         0.41886396],
        [0.64314449, 0.27257355, 0.61578329, ..., 0.63917526, 0.23358959,
         0.22287813],
        [0.60149557, 0.3902604 , 0.59574321, ..., 0.83505155, 0.40370589,
         0.21343303],
        ...,
        [0.45525108, 0.62123774, 0.44578813, ..., 0.48728522, 0.12872068,
         0.1519087 ],
        [0.64456434, 0.66351031, 0.66553797, ..., 0.91065292, 0.49714173,
         0.45231536],
        [0.03686876, 0.50152181, 0.02853984, ..., 0.        , 0.25744136,
         0.10068215]])
```

In [17]:
```
scaled_data = pd.DataFrame(x_scaled, columns=x_data.columns)
```

In [18]:
```
scaled_data
```

Out[18]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compact
0	0.521037	0.022658	0.545989	0.363733	0.593753	
1	0.643144	0.272574	0.615783	0.501591	0.289880	
2	0.601496	0.390260	0.595743	0.449417	0.514309	
3	0.210090	0.360839	0.233501	0.102906	0.811321	
4	0.629893	0.156578	0.630986	0.489290	0.430351	
...	...	...	...	...	...	
564	0.690000	0.428813	0.678668	0.566490	0.526948	
565	0.622320	0.626987	0.604036	0.474019	0.407782	
566	0.455251	0.621238	0.445788	0.303118	0.288165	
567	0.644564	0.663510	0.665538	0.475716	0.588336	
568	0.036869	0.501522	0.028540	0.015907	0.000000	

569 rows × 30 columns

splitting train and test samples

In [19]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(scaled_data, y, test_size=0.15,
```

In [20]:
```
print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (483, 30)
x test:  (86, 30)
y train:  (483,)
y test:  (86,)
```

logistic model interpretation

In [23]:
```python
def sigmoid(z):
    y_head = 1/(1+np.exp(-z))
    return y_head
```

In [28]:
```python
def compute_loss(y_true, y_pred):
    # Avoiding log(0) with small addition
    val = 1e-15
    y_pred = np.clip(y_pred, val, 1 - val)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
```

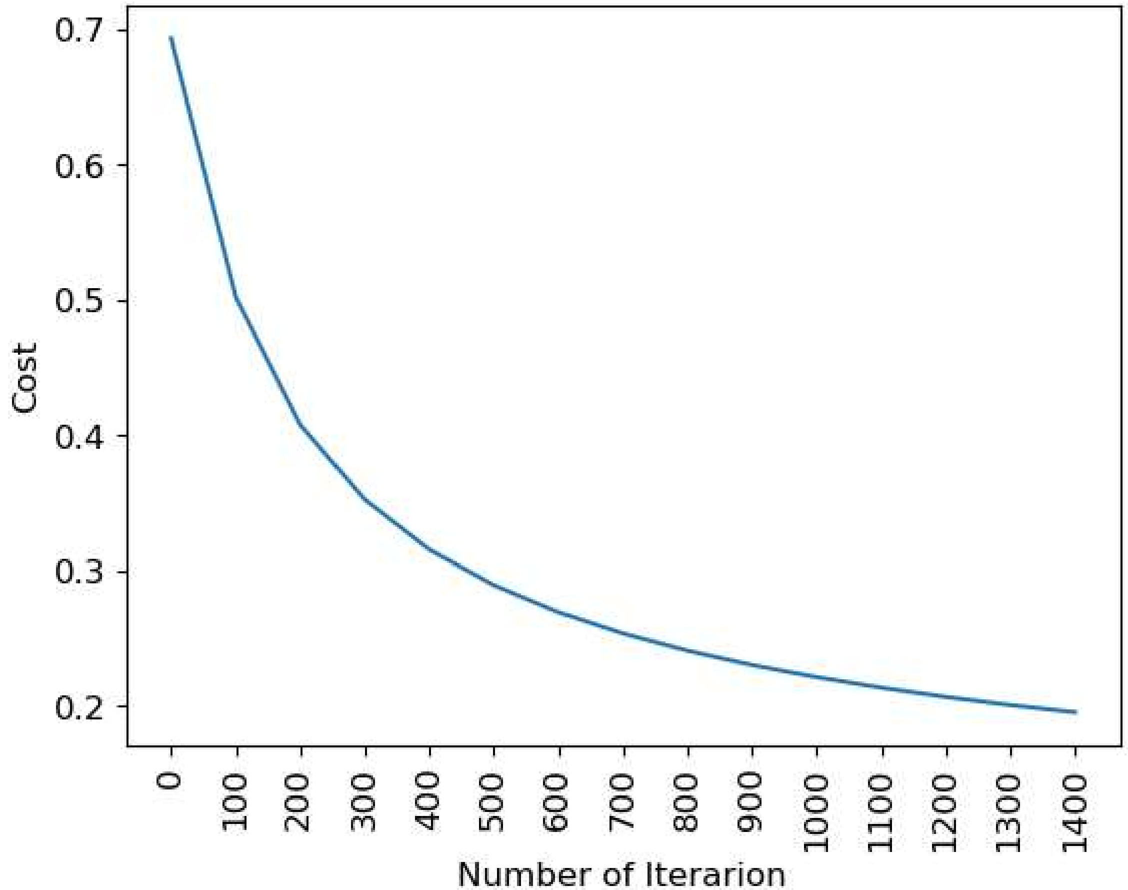
In [47]:
```python
# Gradient descent optimization for logistic regression
def logistic_regression(X, y, learning_rate=0.01, epochs=1000):
    """
    X: Input feature matrix (shape: m x n)
    y: Target labels (shape: m x 1)
    learning_rate: Step size for gradient descent
    epochs: Number of iterations
    """
    # Initialize weights and bias
    m, n = X.shape  # m: number of samples, n: number of features
    weights = np.zeros(n)
    bias = 0
    loss_list = []
    loss_list2 = []
    index = []
    for epoch in range(epochs):
        # Compute linear combination
        z = np.dot(X, weights) + bias

        # Apply sigmoid to get predictions
        y_pred = sigmoid(z)

        # Compute the loss
        loss = compute_loss(y, y_pred)
        loss_list.append(loss)
        if epoch % 100 == 0:
            loss_list2.append(loss)
            index.append(epoch)
            print ("Cost after iteration %i: %f" %(epoch, loss))
        # Gradients calculation
        dw = np.dot(X.T, (y_pred - y)) / m  # Gradient w.r.t. weights
        db = np.sum(y_pred - y) / m         # Gradient w.r.t. bias

        # Update weights and bias
        weights -= learning_rate * dw
        bias -= learning_rate * db

        # Print loss every 100 epochs
```

```python
        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}")

    plt.plot(index,loss_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()

    return weights, bias, loss_list
```

In [59]:
```python
#### model prediction
def predict(X, weights, bias, threshold=0.5):
    probabilities = sigmoid(np.dot(X, weights) + bias)
    return probabilities,(probabilities >= threshold).astype(int)
```

In [93]:
```python
weights, bias, loss = logistic_regression(x_train, y_train, learning_rate=0.1, epoc
```

```
Cost after iteration 0: 0.693147
Epoch 0, Loss: 0.6931
Cost after iteration 100: 0.502415
Epoch 100, Loss: 0.5024
Cost after iteration 200: 0.408058
Epoch 200, Loss: 0.4081
Cost after iteration 300: 0.352446
Epoch 300, Loss: 0.3524
Cost after iteration 400: 0.315645
Epoch 400, Loss: 0.3156
Cost after iteration 500: 0.289314
Epoch 500, Loss: 0.2893
Cost after iteration 600: 0.269406
Epoch 600, Loss: 0.2694
Cost after iteration 700: 0.253726
Epoch 700, Loss: 0.2537
Cost after iteration 800: 0.240986
Epoch 800, Loss: 0.2410
Cost after iteration 900: 0.230376
Epoch 900, Loss: 0.2304
Cost after iteration 1000: 0.221364
Epoch 1000, Loss: 0.2214
Cost after iteration 1100: 0.213583
Epoch 1100, Loss: 0.2136
Cost after iteration 1200: 0.206775
Epoch 1200, Loss: 0.2068
Cost after iteration 1300: 0.200749
Epoch 1300, Loss: 0.2007
Cost after iteration 1400: 0.195362
Epoch 1400, Loss: 0.1954
```



In [94]:
```
y_pred_proba,y_pred=predict(x_test,weights,bias)
```

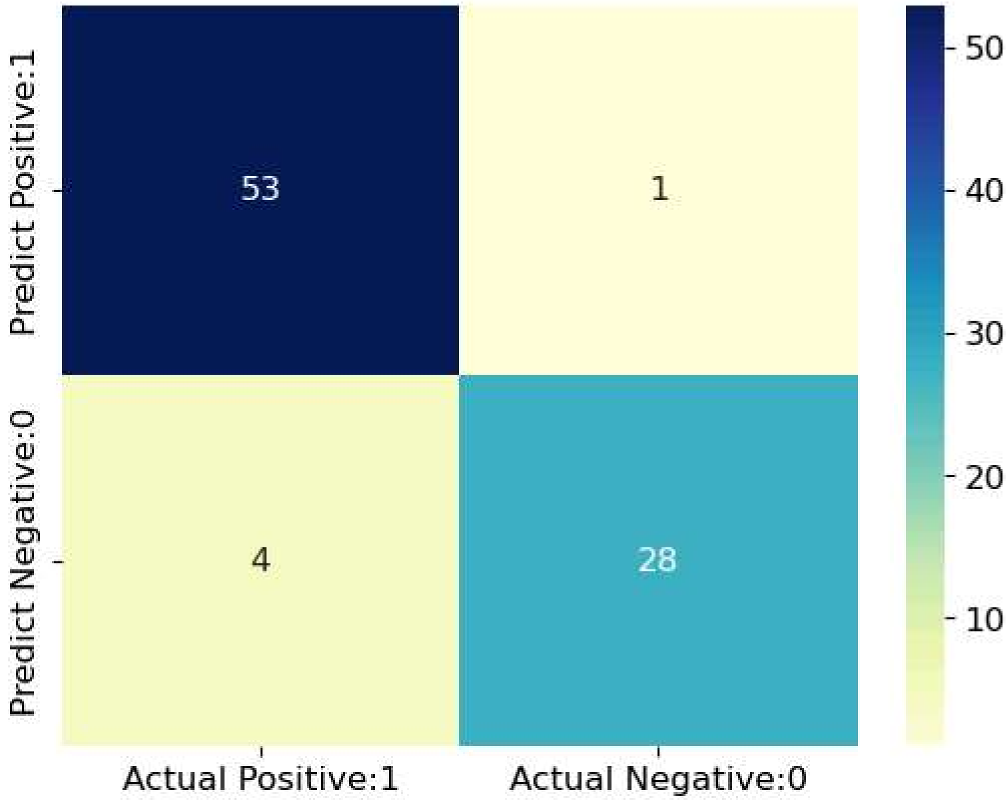
In [95]:
```
y_test,y_pred,y_pred_proba
```

Out[95]:
```
(array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
        1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
        1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1],
       dtype=int64),
 array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
        1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1]),
 array([0.22089955, 0.94302252, 0.77537316, 0.08669849, 0.04877984,
        0.99895021, 0.99728603, 0.72816116, 0.51242851, 0.1337358 ,
        0.13849229, 0.69581088, 0.15348226, 0.53184003, 0.13333444,
        0.91464059, 0.14687007, 0.01525046, 0.00404958, 0.98347057,
        0.40481132, 0.09767545, 0.99699439, 0.02371154, 0.05701281,
        0.07926288, 0.1866041 , 0.07780257, 0.09783395, 0.96707677,
        0.0623139 , 0.06179841, 0.02182269, 0.14569493, 0.02878225,
        0.05823498, 0.46429857, 0.06085171, 0.91885101, 0.20456271,
        0.02744386, 0.68581178, 0.14077995, 0.08375229, 0.08659077,
        0.12328601, 0.01918214, 0.01694625, 0.13857154, 0.19073914,
        0.88334535, 0.9842543 , 0.33266925, 0.14375922, 0.03843487,
        0.21201424, 0.05262118, 0.99968556, 0.68404074, 0.06216004,
        0.13239926, 0.98362705, 0.99138522, 0.18948306, 0.04789653,
        0.28297061, 0.93188638, 0.99531674, 0.04370612, 0.21278713,
        0.56025266, 0.8390829 , 0.12958817, 0.88379436, 0.00891054,
        0.23470442, 0.14118216, 0.44351548, 0.03649441, 0.14869229,
        0.70965004, 0.02809926, 0.4161922 , 0.99317502, 0.68257577,
        0.88181064]))
```

In [96]:
```python
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

In [97]:
```python
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')
```

Out[97]: <Axes: >

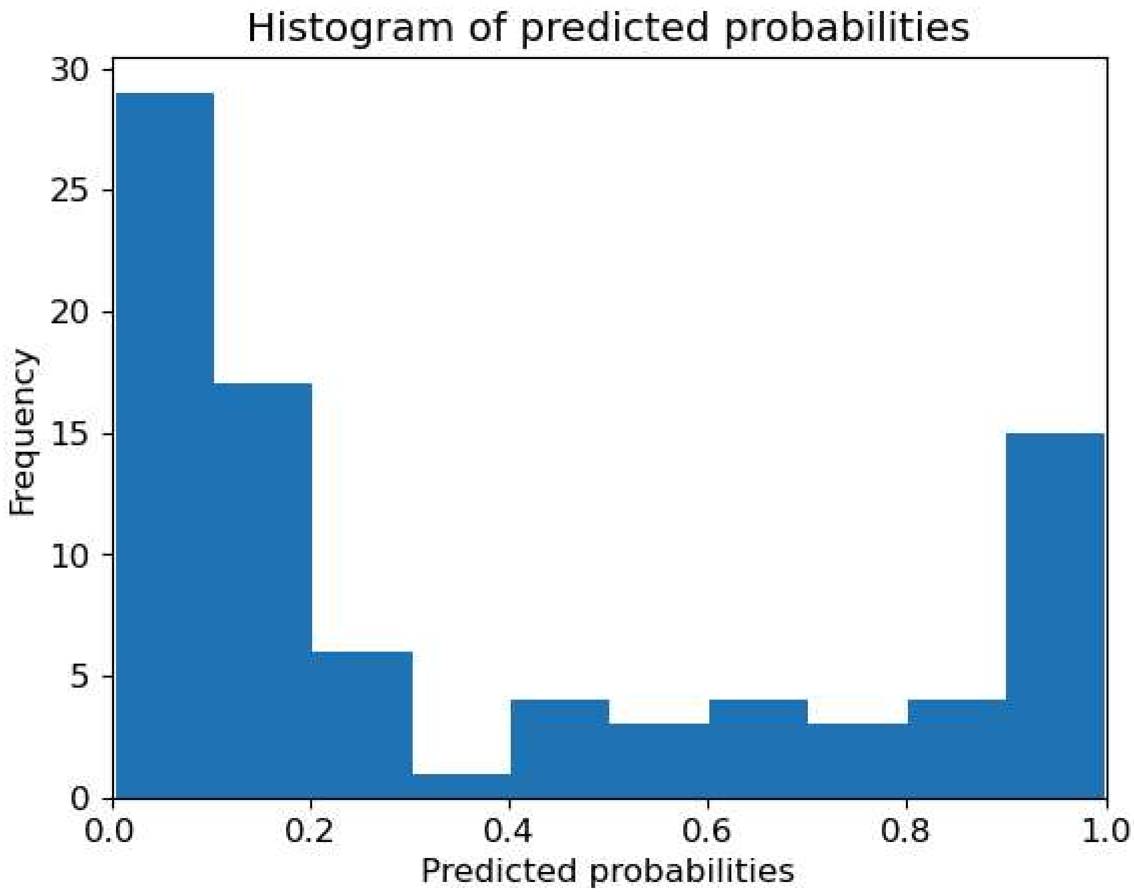


In [98]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        54
           1       0.97      0.88      0.92        32

    accuracy                           0.94        86
   macro avg       0.95      0.93      0.94        86
weighted avg       0.94      0.94      0.94        86
```

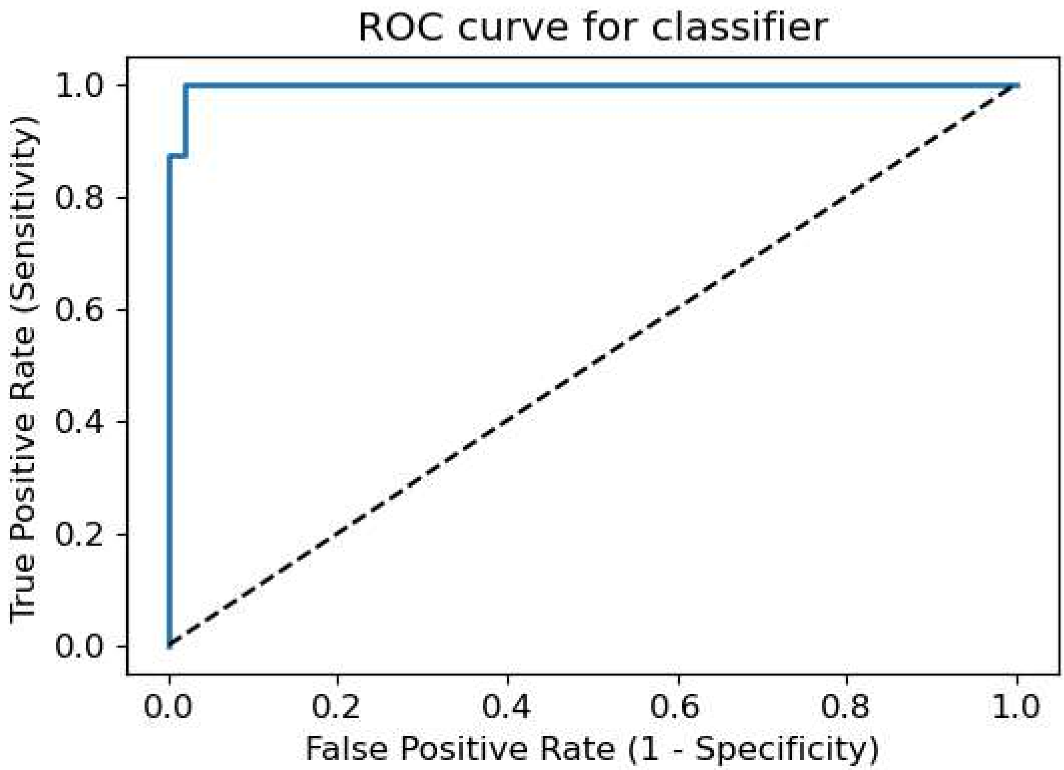
In [99]:
```
plt.hist(y_pred_proba, bins = 10)
# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities')
# set the x-axis limit
plt.xlim(0,1)
# set the title
plt.xlabel('Predicted probabilities')
plt.ylabel('Frequency')
```

Out[99]:
```
Text(0, 0.5, 'Frequency')
```



In [100…

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.show()
print(fpr)
print(tpr)
```




```
[0.         0.         0.         0.01851852 0.01851852 1.        ]
[0.      0.03125 0.875   0.875   1.      1.     ]
```

In [101…

```
from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred)

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

```
ROC AUC : 0.9282
```