# Parkinson's Disease Detection

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

## 1. Library Imports

```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile as zf
import os
import random
import cv2
import imutils
import pickle

from sklearn.metrics import classification_report,confusion_matrix
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder,LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.calibration import calibration_curve
import matplotlib.lines as mlines
from sklearn.metrics import roc_auc_score, roc_curve

from skimage import feature

from keras.utils import to_categorical
from imutils import build_montages,paths
from google.colab.patches import cv2_imshow

sns.set()
```

## 2. Data Loading

```
circle_train_healthy=os.listdir('/content/drive/MyDrive/Dataset/handpd/Circle/Train/Healthy/')
circle_train_park = os.listdir('/content/drive/MyDrive/Dataset/handpd/Circle/Train/Parkinson/')

fp_circle_train_healthy = '/content/drive/MyDrive/Dataset/handpd/Circle/Train/Healthy/'
fp_circle_train_park = '/content/drive/MyDrive/Dataset/handpd/Circle/Train/Parkinson/'

circle_test_healthy = os.listdir('/content/drive/MyDrive/Dataset/handpd/Circle/Test/Healthy/')
circle_test_park = os.listdir('/content/drive/MyDrive/Dataset/handpd/Circle/Test/Parkinson/')

fp_circle_test_healthy = '/content/drive/MyDrive/Dataset/handpd/Circle/Test/Healthy/'
fp_circle_test_park = '/content/drive/MyDrive/Dataset/handpd/Circle/Test/Parkinson/'
```

## 3. Feature Engineering Techniques

Three feature extraction methods are defined for processing the images:

Histogram of Oriented Gradients (HOG): Captures edge direction and intensity. Local Binary Patterns (LBP): Used for texture analysis. Haralick Textures: Extracts statistical texture features.

### Histogram of Oriented Gradients (HOG) technique

```
def quantify_image(image):
  features = feature.hog(image,orientations=9,
              pixels_per_cell=(10,10),cells_per_block=(2,2),transform_sqrt=True,block_norm="L1")

  return features
```

Local Binary Patterns (LBP) for Texture Analysis:

```
from skimage.feature import local_binary_pattern
from skimage import feature

def quantify_image_lbp(image):
    features = local_binary_pattern(image, P=24, R=8, method="uniform")
    (hist, _) = np.histogram(features.ravel(),
                             bins=np.arange(0, 27),
                             range=(0, 26))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
```

Haralick Textures:

```
!pip install mahotas
```

```
     Requirement already satisfied: mahotas in /usr/local/lib/python3.10/dist-packages (1.4.13)
     Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from mahotas) (1.23.5)
```

```
import mahotas as mt

def quantify_image_haralick(image):
    textures = mt.features.haralick(image)
    ht_mean = textures.mean(axis=0)
    return ht_mean
```

## 4. Image Preprocessing and Feature Extraction For Hog FE Technique

```
trainX = []
testX = []
outputs = []
trainY = []
testY = []

for i in circle_train_healthy:
    image_path = (fp_circle_train_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image =cv2.threshold(image, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image(image)
    trainX.append(features)
    trainY.append('healthy')

for i in circle_train_park:

    image_path = (fp_circle_train_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image(image)
    trainX.append(features)
    trainY.append('parkinson')

for i in circle_test_healthy:

    image_path = (fp_circle_test_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image(image)
    testX.append(features)
    testY.append('healthy')

for i in circle_test_park:


    image_path = (fp_circle_test_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image(image)
    testX.append(features)
    testY.append('parkinson')



trainX = np.array(trainX)
testX = np.array(testX)
trainY = np.array(trainY)
testY = np.array(testY)



trainX


     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., 0.],
           [0., 0., 0., ..., 0., 0., 0.]])
```

testX

```
    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., 0.],
           [0., 0., 0., ..., 0., 0., 0.]])
```

trainY

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson'],
          dtype='<U9')
```

testY

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson'], dtype='<U9')
```

```
le = LabelEncoder()
```

```
trainY = le.fit_transform(trainY)
testY = le.transform(testY)
```

trainY

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

testY

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

```
indexes = np.random.randint(0,30,25)
indexes
labels = []

for i in indexes:
  preds = le.inverse_transform(trainY)[i]
  labels.append(trainY)
results = []

for i in range(25):
  image = outputs[i]

  if labels[i] == 'healthy':
    color = (0,255,0)
  else:
    color = (0,0,255)

  text = str(labels[i])

  image = cv2.resize(image,(128,128))
  cv2.putText(image,text,(3,20),cv2.FONT_HERSHEY_SIMPLEX,0.5,color,2)

  results.append(image)
```

```
    <ipython-input-30-f8a98714eab3>:13: FutureWarning: elementwise comparison failed; returning scalar instead, but in the f
      if labels[i] == 'healthy':
```

```
montage = build_montages(results,(128,128),(5,5))[0]

cv2_imshow(montage)
cv2.waitKey(0)
```

```
-1
```

## 5. Machine Learning Models

Several classification models are trained and evaluated:

Random Forest Classifier: An ensemble method for classification.

Decision Tree: A tree-based model for classification.

Support Vector Machine (SVM): A powerful classifier, especially effective in high-dimensional spaces.

Logistic Regression: A linear model for binary classification.

Naive Bayes: A probabilistic classifier based on Bayes' theorem.

Each model is trained on the extracted features and evaluated on the test set. Performance metrics like confusion matrix, sensitivity, specificity, and accuracy are calculated. Also, ROC curves and calibration curves are plotted to evaluate model performance.

## Random Forests

```
model = RandomForestClassifier(n_estimators=100)

model.fit(trainX,trainY)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
preds = model.predict(testX)
preds
```

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

```
cnf = confusion_matrix(testY,preds)
cnf
```

```
    array([[6, 3],
           [1, 7]])
```

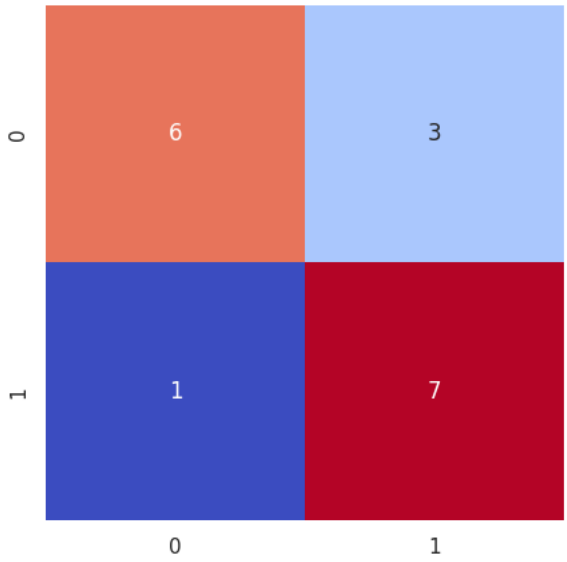
```
sensitivity= 35/(35+9)
sensitivity
```

```
    0.7954545454545454
```

```
specificity= 22/(22+0)
specificity
```

```
    1.0
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```



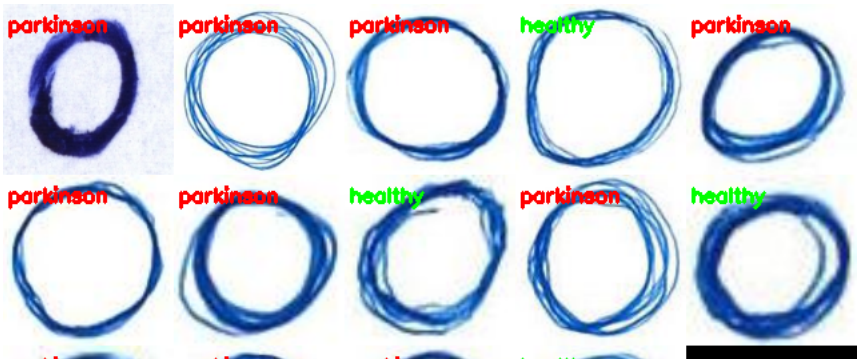

```
acc = metrics.accuracy_score(testY,preds)
acc
```

```
    0.7647058823529411
```

```
indexes = np.random.randint(0,30,25)

indexes
```

```
    array([ 5, 19, 28, 21, 28, 16, 22, 28, 11,  4, 11, 28,  6, 23,  6,  4, 16,
           21,  1, 20, 11, 21, 10, 10,  2])
```

```
montage = build_montages(results,(128,128),(5,5))[0]

cv2_imshow(montage)
cv2.waitKey(0)
```




## Decision Trees

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7)
clf.fit(trainX,trainY)
```

```
DecisionTreeClassifier(max_depth=7)
```

```
preds_dt = clf.predict(testX)
preds_dt
```

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

```
cnf = confusion_matrix(testY,preds_dt)
cnf
```

```
array([[7, 2],
       [4, 4]])
```

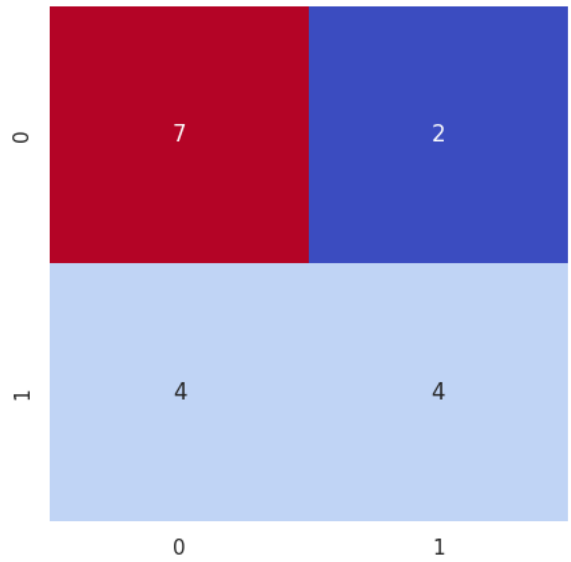
```
sensitivity=29/(29+10)
sensitivity
```

```
0.7435897435897436
```

```
specificity=21/(21+6)
specificity
```

```
0.7777777777777778
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_dt = metrics.accuracy_score(testY,preds_dt)
acc_dt
```

```
0.6470588235294118
```

## Support Vector Machine

```
from sklearn.svm import SVC
svm_mod = SVC(probability=True)
svm_mod.fit(trainX,trainY)
```

```
SVC(probability=True)
```

```
pred_svc = svm_mod.predict(testX)
pred_svc
```

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

```
cnf = confusion_matrix(testY,pred_svc)
cnf
```

```
array([[4, 5],
       [1, 7]])
```

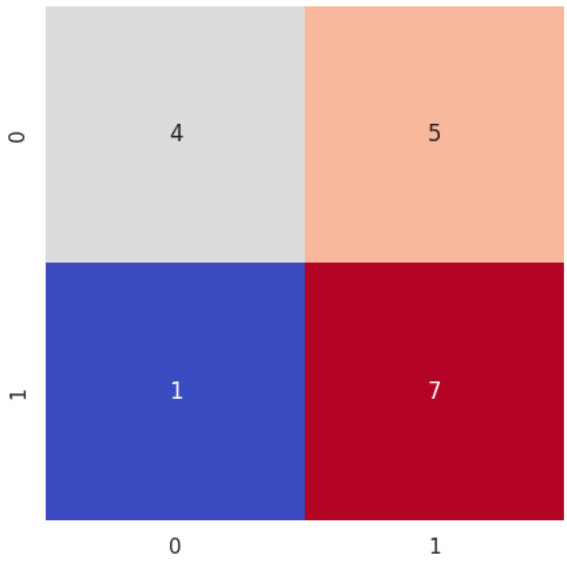
```
sensitivity= 33/(33+4)
sensitivity
```

```
0.8918918918918919
```

```
specificity=27/(27+2)
specificity
```

```
0.9310344827586207
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_svc = metrics.accuracy_score(testY,pred_svc)
acc_svc
```

```
0.6470588235294118
```

```
with open('/content/drive/MyDrive/circle_svm_model_pkl.pkl', 'wb') as files:
    pickle.dump(svm_mod, files)
```

## Logistic Regression

```
lg_mod = LogisticRegression()
lg_mod.fit(trainX, trainY)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred_lg = lg_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_lg)
cnf
```

```
array([[5, 4],
       [2, 6]])
```

```
sensitivity=34/(34+6)
sensitivity
```

```
0.85
```

```
specificity=25/(25+1)
specificity
```

```
0.9615384615384616
```

```
acc_lg = metrics.accuracy_score(testY,pred_lg)
acc_lg
```

```
0.6470588235294118
```

## Naive Bayes

```
gnb_mod = GaussianNB()
gnb_mod.fit(trainX, trainY)
```

```
▾ GaussianNB
GaussianNB()
```

```
pred_nb = gnb_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_nb)
cnf
```

```
array([[4, 5],
       [2, 6]])
```

```
sensitivity=29/(29+15)
sensitivity
```

```
0.6590909090909091
```

```
specificity=16/(16+6)
specificity
```

```
0.7272727272727273
```

```
acc_nb = metrics.accuracy_score(testY,pred_nb)
acc_nb
```

```
0.5882352941176471
```

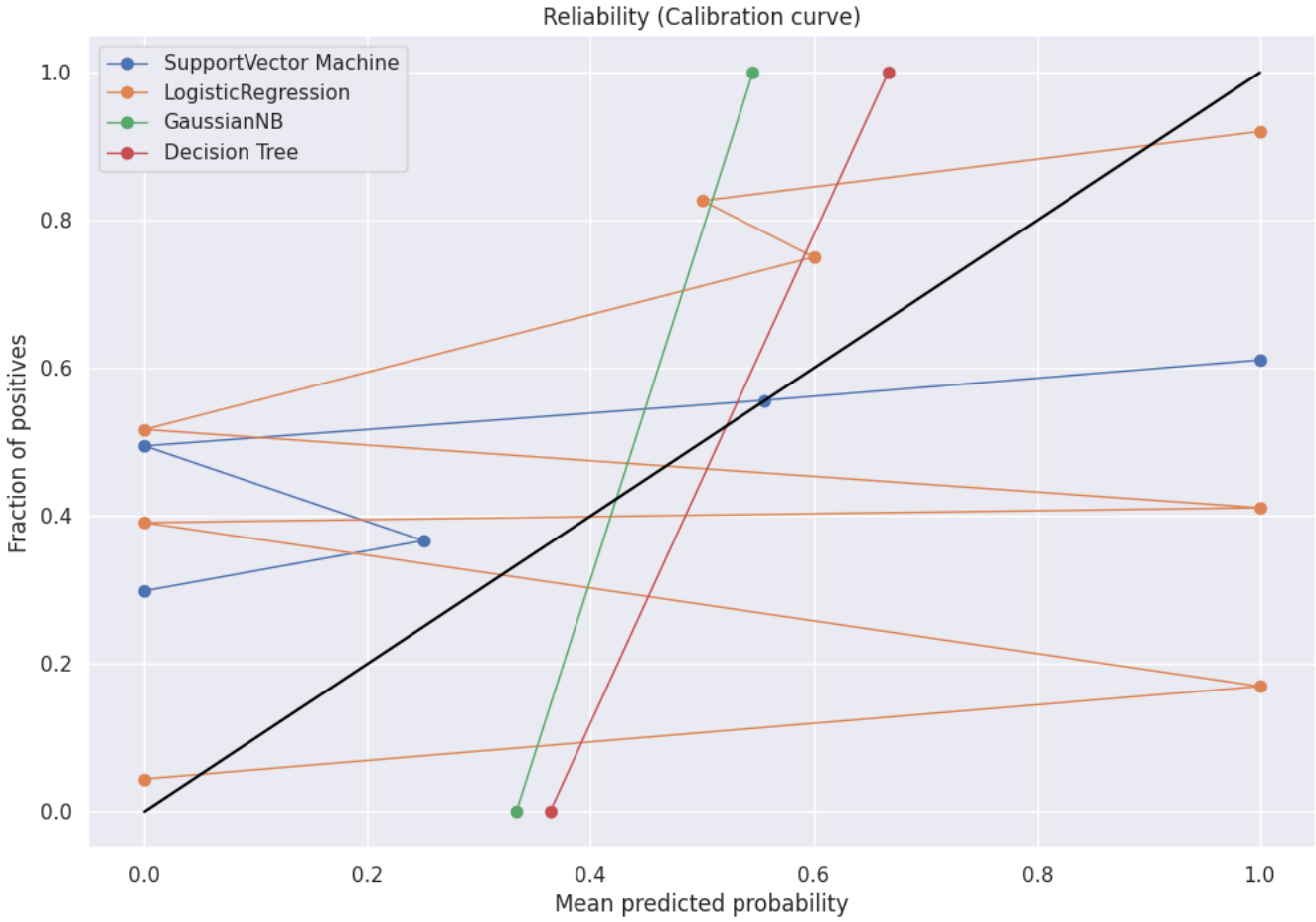
```
no_skill_prob = [0 for _ in range(len(testY))]
no_skill_auc = roc_auc_score(testY, no_skill_prob)
print("No Skill AUC: ", no_skill_auc)
ns_fpr, ns_tpr, _ = roc_curve(testY, no_skill_prob)

dt_prob = clf.predict_proba(testX)[:,-1]
dt_auc = roc_auc_score(testY,dt_prob)
print("DecisionTree AUC: ", dt_auc)
dt_fpr, dt_tpr, _ = roc_curve(testY, dt_prob)

svm_prob = svm_mod.predict_proba(testX)[:,-1]
svm_auc = roc_auc_score(testY,svm_prob)
print("Support Vector Machine AUC: ", svm_auc)
svm_fpr, svm_tpr, _ = roc_curve(testY, svm_prob)

lg_prob = lg_mod.predict_proba(testX)[:,-1]
lg_auc = roc_auc_score(testY, lg_prob)
print("Logistic Regression AUC: ",lg_auc)
lg_fpr, lg_tpr, _ = roc_curve(testY, lg_prob)

nb_prob = gnb_mod.predict_proba(testX)[:,-1]
nb_auc = roc_auc_score(testY, nb_prob)
print("Gaussian Naive Bayes AUC: ", nb_auc)
nb_fpr, nb_tpr, _ = roc_curve(testY, nb_prob)



fig,ax = plt.subplots(figsize=(12,8))
ax.plot(ns_fpr, ns_tpr, linestyle= '--', label= 'Reference')
ax.plot(svm_fpr, svm_tpr, linestyle= '-', label= 'SupportVectorMachine')
ax.plot(lg_fpr, lg_tpr, linestyle='-', label= 'LogisticRegression')
ax.plot(nb_fpr, nb_tpr, linestyle='-', label= 'GaussianNB')
ax.plot(dt_fpr, dt_tpr, linestyle='-', label= 'Decision Tree')

plt.xlabel('False Positve rate')
plt.ylabel('True Positive rate')
plt.title('ROC curve')
```

```
    No Skill AUC:  0.5
```

```
sv_x, sv_y = calibration_curve(testY, svm_prob, n_bins=10)
lg_x, lg_y = calibration_curve(testY, lg_prob, n_bins=10)
nb_x, nb_y = calibration_curve(testY, nb_prob, n_bins=10)
dt_x, dt_y = calibration_curve(testY,dt_prob,n_bins=10)
```

```
fig, ax = plt.subplots(figsize=(12,8))

ax.plot(sv_x, sv_y, marker= 'o', linewidth= '1', label= 'SupportVector Machine')
ax.plot(lg_x, lg_y, marker= 'o', linewidth= '1', label= 'LogisticRegression')
ax.plot(nb_x, nb_y, marker= 'o', linewidth= '1', label= 'GaussianNB')
ax.plot(dt_x, dt_y, marker= 'o', linewidth= '1', label= 'Decision Tree')
```

```
line = mlines.Line2D([0,1],[0,1], color='black')
ax.add_line(line)
ax.legend()
plt.xlabel('Mean predicted probability')
plt.ylabel('Fraction of positives')
plt.title('Reliability (Calibration curve)')
plt.show()
```



```
circle_model = {
    'Random Forest': model,
    'Support Vector Machine': svm_mod,
    'Decision tree':clf,
    'Gaussian Naive Bayes': gnb_mod,
    'Logistic Regression': lg_mod

}

columns = [
    'accuracy'
]

circle_table = {}


for model in circle_model.keys():

    circle_pred = circle_model[model].predict(testX)
    accuracy = circle_model[model].score(testX,testY)
    circle_table[model] = [accuracy]


circle_results = pd.DataFrame.from_dict(circle_table, orient='index')
circle_results.columns = columns
circle_results
```

| | accuracy |
|---|---|
| **Random Forest** | 0.764706 |
| **Support Vector Machine** | 0.647059 |
| **Decision tree** | 0.647059 |
| **Gaussian Naive Bayes** | 0.588235 |
| **Logistic Regression** | 0.647059 |

## Local Binary Patterns

```
trainX = []
testX = []
outputs = []
trainY = []
testY = []

for i in circle_train_healthy:
    image_path = (fp_circle_train_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image =cv2.threshold(image, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_lbp(image)
    trainX.append(features)
    trainY.append('healthy')
    # ... rest of your code ...



for i in circle_train_park:

    image_path = (fp_circle_train_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_lbp(image)
    trainX.append(features)
    trainY.append('parkinson')

for i in circle_test_healthy:

    image_path = (fp_circle_test_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_lbp(image)
    testX.append(features)
    testY.append('healthy')

for i in circle_test_park:


    image_path = (fp_circle_test_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_lbp(image)
    testX.append(features)
    testY.append('parkinson')



trainX = np.array(trainX)
testX = np.array(testX)
trainY = np.array(trainY)
testY = np.array(testY)



trainX
```

```
array([[0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 9.05000e-03,
        8.35325e-01, 3.37500e-03],
```

```
            [1.52500e-03, 3.20000e-03, 1.75000e-04, ..., 0.00000e+00,
             8.24925e-01, 1.65275e-01],
            [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.60000e-03,
             8.39225e-01, 8.21750e-02],
            ...,
            [7.50000e-05, 3.00000e-04, 1.75000e-04, ..., 2.01750e-02,
             7.15800e-01, 1.38400e-01],
            [4.00000e-04, 2.77500e-03, 2.10000e-03, ..., 3.02250e-02,
             6.71425e-01, 2.09675e-01],
            [5.00000e-05, 4.25000e-04, 7.50000e-05, ..., 2.95000e-03,
             6.78500e-01, 2.89200e-01]])
```

testX

```
            2.75000e-04, 3.25000e-03, 8.50000e-03, 1.26000e-02, 9.75000e-03,
             1.00500e-02, 8.45000e-03, 7.62500e-03, 5.90000e-03, 6.37500e-03,
             3.32500e-03, 3.30000e-03, 5.47500e-03, 1.03750e-02, 7.10600e-01,
             1.93175e-01],
            [0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
             0.00000e+00, 0.00000e+00, 5.00000e-05, 5.00000e-05, 1.25000e-04,
             3.25000e-04, 2.42500e-03, 8.67500e-03, 1.37250e-02, 9.97500e-03,
             1.15000e-02, 8.65000e-03, 1.17000e-02, 9.55000e-03, 1.02750e-02,
             6.42500e-03, 9.20000e-03, 1.19000e-02, 2.77750e-02, 6.96475e-01,
             1.61200e-01],
            [0.00000e+00, 0.00000e+00, 2.00000e-04, 0.00000e+00, 0.00000e+00,
             0.00000e+00, 2.50000e-05, 2.50000e-05, 7.50000e-05, 3.00000e-04,
             8.75000e-04, 2.57500e-03, 5.60000e-03, 6.95000e-03, 3.50000e-03,
             3.20000e-03, 1.17500e-03, 1.27500e-03, 1.50000e-04, 2.25000e-04,
             1.25000e-04, 3.25000e-04, 0.00000e+00, 0.00000e+00, 8.69525e-01,
             1.03875e-01],
            [8.00000e-04, 5.60000e-03, 9.00000e-04, 1.00000e-04, 2.00000e-04,
             1.25000e-04, 5.00000e-05, 2.50000e-05, 2.50000e-05, 3.25000e-04,
             3.50000e-04, 1.07500e-03, 1.10000e-03, 1.62500e-03, 1.17500e-03,
             1.57500e-03, 5.50000e-04, 5.00000e-04, 5.00000e-04, 5.50000e-04,
             3.75000e-04, 4.25000e-04, 5.50000e-04, 9.25000e-04, 7.69150e-01,
             2.11425e-01],
            [1.50000e-04, 0.00000e+00, 0.00000e+00, 0.00000e+00, 2.50000e-05,
             1.25000e-04, 1.50000e-04, 2.00000e-04, 1.75000e-04, 4.25000e-04,
             1.27500e-03, 5.87500e-03, 1.06000e-02, 1.69000e-02, 1.35750e-02,
             1.69750e-02, 1.40000e-02, 1.61000e-02, 1.26750e-02, 1.35750e-02,
             8.85000e-03, 1.25000e-02, 9.90000e-03, 1.98500e-02, 7.91375e-01,
             3.47250e-02],
            [0.00000e+00, 0.00000e+00, 1.00000e-04, 5.00000e-05, 1.75000e-04,
             2.75000e-04, 3.50000e-04, 2.25000e-04, 1.75000e-04, 7.00000e-04,
             8.50000e-04, 2.50000e-03, 3.40000e-03, 5.90000e-03, 5.75000e-03,
             7.45000e-03, 6.35000e-03, 6.57500e-03, 6.80000e-03, 8.47500e-03,
             7.75000e-03, 1.16750e-02, 1.35250e-02, 2.00500e-02, 7.40500e-01,
             1.50400e-01],
            [5.75000e-04, 3.00000e-04, 1.25000e-04, 2.50000e-05, 2.50000e-05,
             5.00000e-05, 2.50000e-05, 2.50000e-04, 2.25000e-04, 3.00000e-04,
             1.02500e-03, 4.25000e-03, 1.01000e-02, 1.90000e-02, 1.56000e-02,
             1.76000e-02, 1.43750e-02, 1.90500e-02, 1.22000e-02, 1.19000e-02,
             7.80000e-03, 1.08750e-02, 7.40000e-03, 1.46250e-02, 7.84550e-01,
             4.77500e-02],
            [7.50000e-05, 7.75000e-04, 2.25000e-04, 5.00000e-05, 0.00000e+00,
             0.00000e+00, 1.75000e-04, 1.00000e-04, 1.00000e-04, 5.25000e-04,
             9.50000e-04, 3.65000e-03, 6.17500e-03, 9.17500e-03, 8.12500e-03,
             1.04500e-02, 5.67500e-03, 4.65000e-03, 2.72500e-03, 3.25000e-03,
             1.95000e-03, 1.67500e-03, 1.02500e-03, 1.12500e-03, 7.83025e-01,
             1.54350e-01],
            [5.00000e-05, 1.00000e-04, 6.25000e-04, 1.82500e-03, 7.00000e-04,
             3.50000e-04, 2.50000e-04, 1.50000e-04, 3.00000e-04, 4.25000e-04,
             1.17500e-03, 3.45000e-03, 7.02500e-03, 9.95000e-03, 1.04000e-02,
             1.25250e-02, 1.06250e-02, 1.12750e-02, 1.08500e-02, 1.27750e-02,
             1.02750e-02, 1.38000e-02, 1.04250e-02, 1.21000e-02, 8.00800e-01,
             5.77750e-02],
            [5.50000e-04, 4.00000e-04, 7.25000e-04, 1.00000e-04, 0.00000e+00,
             2.50000e-05, 0.00000e+00, 1.00000e-04, 5.00000e-05, 1.75000e-04,
             3.00000e-04, 1.55000e-03, 3.22500e-03, 8.07500e-03, 5.92500e-03,
             8.17500e-03, 5.82500e-03, 9.47500e-03, 6.37500e-03, 8.25000e-03,
             6.45000e-03, 1.23250e-02, 1.26750e-02, 1.95250e-02, 6.93850e-01,
             1.95875e-01]])
```

trainY

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson'],
          dtype='<U9')
```

testY

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson'], dtype='<U9')
```

```
le = LabelEncoder()
trainY = le.fit_transform(trainY)
testY = le.transform(testY)
indexes = np.random.randint(0,30,25)
indexes
labels = []

for i in indexes:
  preds = le.inverse_transform(trainY)[i]
  labels.append(trainY)
results = []

for i in range(25):
  image = outputs[i]

  if labels[i] == 'healthy':
    color = (0,255,0)
  else:
    color = (0,0,255)

  text = str(labels[i])

  image = cv2.resize(image,(128,128))
  cv2.putText(image,text,(3,20),cv2.FONT_HERSHEY_SIMPLEX,0.5,color,2)

  results.append(image)
```

```
    <ipython-input-100-183257d9622d>:16: FutureWarning: elementwise comparison failed; returning scalar instead, but in the
      if labels[i] == 'healthy':
```

```
montage = build_montages(results,(128,128),(5,5))[0]

cv2_imshow(montage)
cv2.waitKey(0)
```

```
-1
```

## Random Forests

```
model = RandomForestClassifier(n_estimators=100)
```

```
model.fit(trainX,trainY)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
preds = model.predict(testX)
preds
```

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

```
cnf = confusion_matrix(testY,preds)
cnf
```

```
array([[7, 2],
       [3, 5]])
```

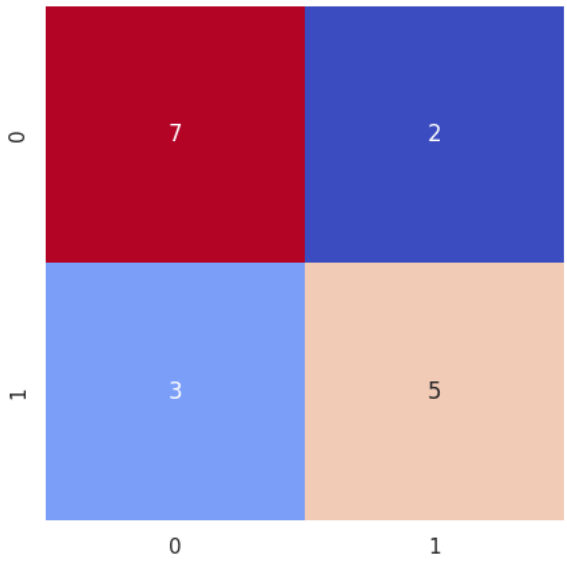
```
sensitivity= 35/(35+9)
sensitivity
```

```
0.7954545454545454
```

```
specificity= 22/(22+0)
specificity
```

```
1.0
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```



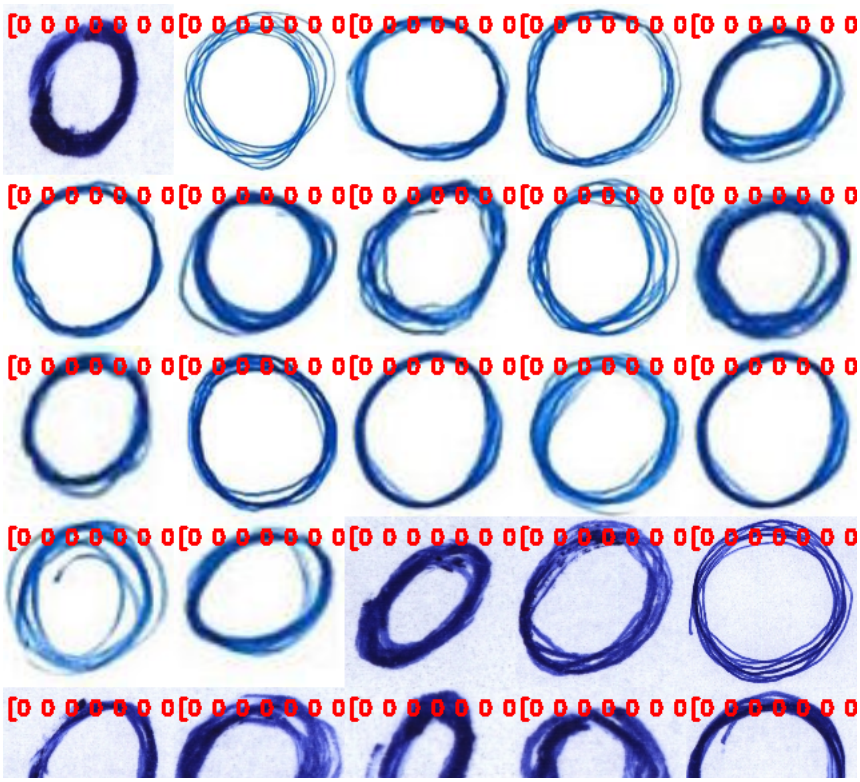

```
acc = metrics.accuracy_score(testY,preds)
acc
```

```
0.7058823529411765
```

```
indexes = np.random.randint(0,30,25)
indexes
```

```
array([28, 24,  3, 24, 20, 24, 25, 19, 18, 21, 12,  3,  3, 22, 19,  0, 21,
        7,  4,  8, 23, 17, 11, 20,  1])
```

```
montage = build_montages(results,(128,128),(5,5))[0]

cv2_imshow(montage)
cv2.waitKey(0)
```



## Decision Trees

-

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7)
clf.fit(trainX,trainY)
```

```
DecisionTreeClassifier(max_depth=7)
```

```
preds_dt = clf.predict(testX)
preds_dt
```

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

```
cnf = confusion_matrix(testY,preds_dt)
cnf
```

```
array([[6, 3],
       [4, 4]])
```

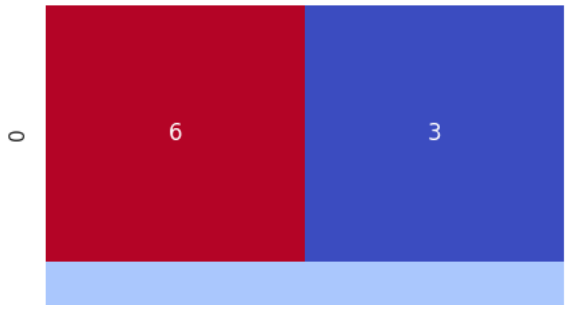
```
sensitivity=29/(29+10)
sensitivity
```

```
0.7435897435897436
```

```
specificity=21/(21+6)
specificity
```

```
0.7777777777777778
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_dt = metrics.accuracy_score(testY,preds_dt)
acc_dt
```

```
0.5882352941176471
```

## Support Vector Machine

```
from sklearn.svm import SVC
svm_mod = SVC(probability=True)
svm_mod.fit(trainX,trainY)
```

```
SVC(probability=True)
```

```
pred_svc = svm_mod.predict(testX)
pred_svc
```

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

```
cnf = confusion_matrix(testY,pred_svc)
cnf
```

```
array([[9, 0],
       [8, 0]])
```

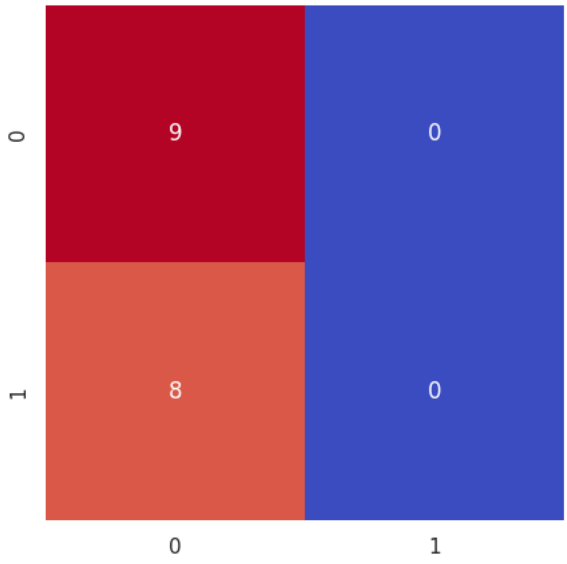
```
sensitivity= 33/(33+4)
sensitivity
```

```
0.8918918918918919
```

```
specificity=27/(27+2)
specificity
```

```
0.9310344827586207
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```



```
acc_svc = metrics.accuracy_score(testY,pred_svc)
acc_svc
```

```
0.5294117647058824
```

```
with open('/content/drive/MyDrive/circle_svm_model_pkl.pkl', 'wb') as files:
    pickle.dump(svm_mod, files)
```

## Logistic Regression

```
lg_mod = LogisticRegression()
lg_mod.fit(trainX, trainY)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred_lg = lg_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_lg)
cnf
```

```
array([[9, 0],
       [8, 0]])
```

```
sensitivity=34/(34+6)
sensitivity
```

```
0.85
```

```
specificity=25/(25+1)
specificity
```

```
0.9615384615384616
```

```
acc_lg = metrics.accuracy_score(testY,pred_lg)
acc_lg
```

```
0.5294117647058824
```

## Naive Bayes

```
gnb_mod = GaussianNB()
gnb_mod.fit(trainX, trainY)
```

```
▾ GaussianNB
GaussianNB()
```

```
pred_nb = gnb_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_nb)
cnf
```

```
array([[7, 2],
       [3, 5]])
```

```
sensitivity=29/(29+15)
sensitivity
```

```
0.6590909090909091
```

```
specificity=16/(16+6)
specificity
```

```
0.7272727272727273
```

```
acc_nb = metrics.accuracy_score(testY,pred_nb)
acc_nb
```

```
0.7058823529411765
```

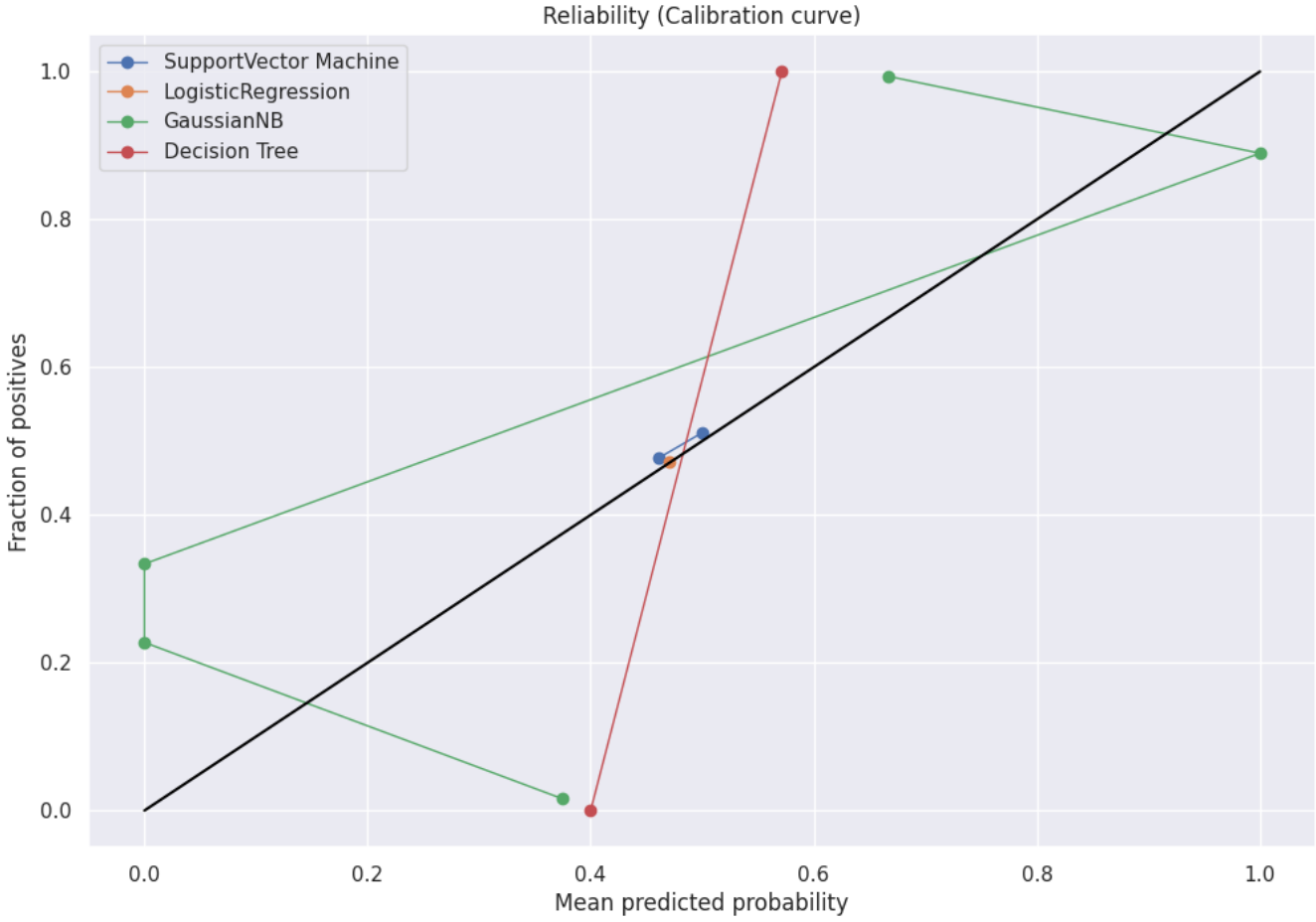
```
no_skill_prob = [0 for _ in range(len(testY))]
no_skill_auc = roc_auc_score(testY, no_skill_prob)
print("No Skill AUC: ", no_skill_auc)
ns_fpr, ns_tpr, _ = roc_curve(testY, no_skill_prob)

dt_prob = clf.predict_proba(testX)[:,-1]
dt_auc = roc_auc_score(testY,dt_prob)
print("DecisionTree AUC: ", dt_auc)
dt_fpr, dt_tpr, _ = roc_curve(testY, dt_prob)

svm_prob = svm_mod.predict_proba(testX)[:,-1]
svm_auc = roc_auc_score(testY,svm_prob)
print("Support Vector Machine AUC: ", svm_auc)
svm_fpr, svm_tpr, _ = roc_curve(testY, svm_prob)

lg_prob = lg_mod.predict_proba(testX)[:,-1]
lg_auc = roc_auc_score(testY, lg_prob)
print("Logistic Regression AUC: ",lg_auc)
lg_fpr, lg_tpr, _ = roc_curve(testY, lg_prob)

nb_prob = gnb_mod.predict_proba(testX)[:,-1]
nb_auc = roc_auc_score(testY, nb_prob)
print("Gaussian Naive Bayes AUC: ", nb_auc)
nb_fpr, nb_tpr, _ = roc_curve(testY, nb_prob)


fig,ax = plt.subplots(figsize=(12,8))
ax.plot(ns_fpr, ns_tpr, linestyle= '--', label= 'Reference')
ax.plot(svm_fpr, svm_tpr, linestyle= '-', label= 'SupportVectorMachine')
ax.plot(lg_fpr, lg_tpr, linestyle='-', label= 'LogisticRegression')
ax.plot(nb_fpr, nb_tpr, linestyle='-', label= 'GaussianNB')
ax.plot(dt_fpr, dt_tpr, linestyle='-', label= 'Decision Tree')

plt.xlabel('False Positve rate')
plt.ylabel('True Positive rate')
plt.title('ROC curve')
```

```
No Skill AUC:  0.5
DecisionTree AUC:  0.5833333333333334
```

```
sv_x, sv_y = calibration_curve(testY, svm_prob, n_bins=10)
lg_x, lg_y = calibration_curve(testY, lg_prob, n_bins=10)
nb_x, nb_y = calibration_curve(testY, nb_prob, n_bins=10)
dt_x, dt_y = calibration_curve(testY,dt_prob,n_bins=10)
```

```
fig, ax = plt.subplots(figsize=(12,8))

ax.plot(sv_x, sv_y, marker= 'o', linewidth= '1', label= 'SupportVector Machine')
ax.plot(lg_x, lg_y, marker= 'o', linewidth= '1', label= 'LogisticRegression')
ax.plot(nb_x, nb_y, marker= 'o', linewidth= '1', label= 'GaussianNB')
ax.plot(dt_x, dt_y, marker= 'o', linewidth= '1', label= 'Decision Tree')
```

```
line = mlines.Line2D([0,1],[0,1], color='black')
ax.add_line(line)
ax.legend()
plt.xlabel('Mean predicted probability')
plt.ylabel('Fraction of positives')
plt.title('Reliability (Calibration curve)')
plt.show()
```



```
circle_model = {
    'Random Forest': model,
    'Support Vector Machine': svm_mod,
    'Decision tree':clf,
    'Gaussian Naive Bayes': gnb_mod,
    'Logistic Regression': lg_mod

}

columns = [
    'accuracy'
]

circle_table = {}


for model in circle_model.keys():

    circle_pred = circle_model[model].predict(testX)
    accuracy = circle_model[model].score(testX,testY)
    circle_table[model] = [accuracy]


circle_results = pd.DataFrame.from_dict(circle_table, orient='index')
circle_results.columns = columns
circle_results
```

| | accuracy |
|---|---|
| **Random Forest** | 0.705882 |
| **Support Vector Machine** | 0.529412 |
| **Decision tree** | 0.588235 |
| **Gaussian Naive Bayes** | 0.705882 |
| **Logistic Regression** | 0.529412 |

## Haralick Textures:

```
!pip install mahotas
```

```
    Requirement already satisfied: mahotas in /usr/local/lib/python3.10/dist-packages (1.4.13)
    Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from mahotas) (1.23.5)
```

```
import mahotas as mt

def quantify_image_haralick(image):
    textures = mt.features.haralick(image)
    ht_mean = textures.mean(axis=0)
    return ht_mean
```

```
trainX = []
testX = []
outputs = []
trainY = []
testY = []

for i in circle_train_healthy:
    image_path = (fp_circle_train_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image =cv2.threshold(image, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_haralick(image)
    trainX.append(features)
    trainY.append('healthy')
    # ... rest of your code ...



for i in circle_train_park:

    image_path = (fp_circle_train_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_haralick(image)
    trainX.append(features)
    trainY.append('parkinson')

for i in circle_test_healthy:

    image_path = (fp_circle_test_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_haralick(image)
    testX.append(features)
    testY.append('healthy')

for i in circle_test_park:


    image_path = (fp_circle_test_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_haralick(image)
    testX.append(features)
    testY.append('parkinson')



trainX = np.array(trainX)
testX = np.array(testX)
trainY = np.array(trainY)
testY = np.array(testY)



trainX
```

```
              8.65690022e-01],
            [ 6.16996356e-01,  2.62820801e+03,  8.82638371e-01,
              1.11939286e+04,  9.59582198e-01,  1.12700296e+02,
              4.21475065e+04,  9.52921090e-01,  9.93339513e-01,
              3.58858893e-03,  2.42847666e-01, -6.96370770e-01,
              8.07949407e-01],
            [ 5.21392271e-01,  1.48540607e+03,  9.49947635e-01,
              1.48355313e+04,  9.77156736e-01,  1.79616498e+02,
              5.78567190e+04,  1.06820807e+00,  1.09105169e+00,
              3.71673070e-03,  1.56587759e-01, -8.34372537e-01,
              8.88851987e-01],
            [ 5.41894584e-01,  1.96134914e+03,  9.29686525e-01,
              1.39441027e+04,  9.69837463e-01,  1.58831239e+02,
              5.38150618e+04,  1.05561720e+00,  1.08578020e+00,
              3.66270171e-03,  1.94410452e-01, -7.86619662e-01,
              8.68840184e-01],
            [ 5.37578102e-01,  1.79421365e+03,  9.36671767e-01,
              1.41630391e+04,  9.72407750e-01,  1.63497605e+02,
              5.48579426e+04,  1.05602368e+00,  1.08361635e+00,
              3.68158788e-03,  1.81429531e-01, -8.02674225e-01,
              8.75061059e-01],
            [ 5.41984974e-01,  2.67668817e+03,  9.01688010e-01,
              1.36097615e+04,  9.58836647e-01,  1.52112642e+02,
              5.17623580e+04,  1.07904733e+00,  1.12021131e+00,
              3.58306830e-03,  2.46585707e-01, -7.25842720e-01,
              8.48657250e-01],
            [ 5.50933209e-01,  1.40424181e+03,  9.49548773e-01,
              1.39137365e+04,  9.78404918e-01,  1.58201772e+02,
              5.42507040e+04,  1.01920108e+00,  1.04079649e+00,
              3.72601919e-03,  1.49874804e-01, -8.35039441e-01,
              8.80288835e-01],
            [ 7.40204651e-01,  1.53041750e+03,  9.00655243e-01,
              7.69992813e+03,  9.76464530e-01,  6.99993062e+01,
              2.92692950e+04,  7.00642794e-01,  7.24178626e-01,
              3.71157268e-03,  1.60336929e-01, -7.44974616e-01,
              7.59134178e-01],
            [ 5.41539409e-01,  1.24861267e+03,  9.56332750e-01,
              1.42937568e+04,  9.80798255e-01,  1.66400894e+02,
              5.59264144e+04,  1.02654632e+00,  1.04574836e+00,
              3.74394604e-03,  1.36491361e-01, -8.52199949e-01,
              8.87798990e-01],
            [ 5.15572446e-01,  2.54022738e+03,  9.12612400e-01,
              1.45313016e+04,  9.60935205e-01,  1.71935920e+02,
              5.55849792e+04,  1.11653371e+00,  1.15559910e+00,
              3.59815932e-03,  2.36852589e-01, -7.46726179e-01,
              8.64280823e-01],
            [ 4.85598298e-01,  3.33104318e+03,  8.90065425e-01,
              1.51478433e+04,  9.48773672e-01,  1.88416191e+02,
              5.72603302e+04,  1.18630606e+00,  1.23753318e+00,
              3.51213645e-03,  2.90048556e-01, -6.97687994e-01,
              8.56334408e-01],
            [ 4.87234332e-01,  5.14247736e+03,  8.20340549e-01,
              1.43089498e+04,  9.20916597e-01,  1.66744539e+02,
              5.20933217e+04,  1.22200885e+00,  1.30109347e+00,
              3.32333838e-03,  3.97275287e-01, -5.73056607e-01,
              8.04081902e-01]])

testX

              5.14566696e+04,  1.05186239e+00,  1.08747917e+00,
              3.62285281e-03,  2.21261444e-01, -7.51913538e-01,
              8.54345886e-01],
            [ 6.25644295e-01,  1.91631718e+03,  9.14797011e-01,
```

```
            3.73271086e-03,  1.44920967e-01, -8.44570912e-01,
            8.88168654e-01],
           [ 5.50621439e-01,  2.17268736e+03,  9.19911738e-01,
            1.35612994e+04,  9.66587406e-01,  1.51174864e+02,
            5.20725103e+04,  1.04951613e+00,  1.08292924e+00,
            3.63889561e-03,  2.10592710e-01, -7.65035728e-01,
            8.59194768e-01],
           [ 5.78213036e-01,  1.59917642e+03,  9.38195293e-01,
            1.29343593e+04,  9.75407123e-01,  1.39728841e+02,
            5.01382608e+04,  9.84044165e-01,  1.00863742e+00,
            3.70380615e-03,  1.65695978e-01, -8.09348979e-01,
            8.63584327e-01],
           [ 6.15364764e-01,  2.86093334e+03,  8.71627874e-01,
            1.11398587e+04,  9.56003240e-01,  1.11942366e+02,
            4.16985014e+04,  9.61863059e-01,  1.00586050e+00,
            3.56284953e-03,  2.59306391e-01, -6.75231201e-01,
            8.00254182e-01],
           [ 5.68096853e-01,  1.49847232e+03,  9.43726853e-01,
            1.33109989e+04,  9.76955797e-01,  1.46460284e+02,
            5.17455231e+04,  9.96083199e-01,  1.01912776e+00,
            3.71527701e-03,  1.57492207e-01, -8.21903926e-01,
            8.70877329e-01],
           [ 5.12491131e-01,  2.97394016e+03,  8.97006755e-01,
            1.44348190e+04,  9.54265368e-01,  1.69644560e+02,
            5.47653358e+04,  1.13395367e+00,  1.17968901e+00,
            3.55090252e-03,  2.66219661e-01, -7.14397586e-01,
            8.53925834e-01]])
```

```
trainY
```

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson'],
          dtype='<U9')
```

```
testY
```

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson'], dtype='<U9')
```

```
le = LabelEncoder()
```

```
trainY = le.fit_transform(trainY)
testY = le.transform(testY)
```

```
trainY
```

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

```
testY
```

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

## Random Forests

```
model = RandomForestClassifier(n_estimators=100)
```

```
model.fit(trainX,trainY)
```

```
    ▾ RandomForestClassifier
    RandomForestClassifier()
```

```
preds = model.predict(testX)
preds
```

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

```
cnf = confusion_matrix(testY,preds)
cnf
```

```
array([[5, 4],
       [3, 5]])
```

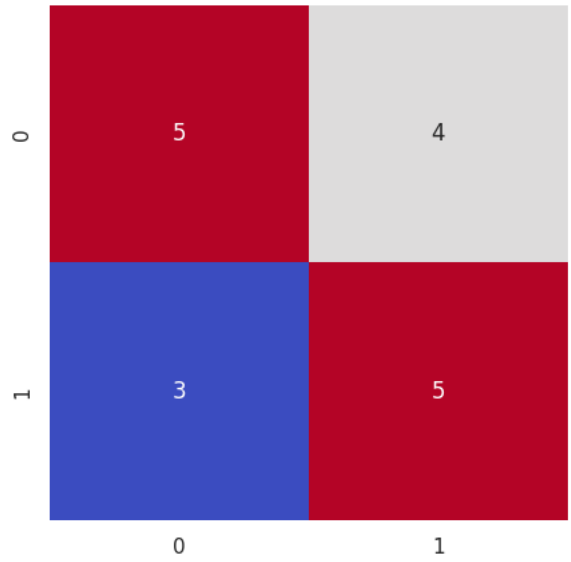
```
sensitivity= 35/(35+9)
sensitivity
```

```
0.7954545454545454
```

```
specificity= 22/(22+0)
specificity
```

```
1.0
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc = metrics.accuracy_score(testY,preds)
acc
```

```
0.5882352941176471
```

```
indexes = np.random.randint(0,30,25)
indexes
```

```
array([27,  0,  1,  5, 24, 13, 15,  2, 13, 25, 14,  2,  4, 25, 16, 28, 28,
       29,  8,  7, 18,  2, 26, 15, 23])
```

## Decision Trees

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7)
clf.fit(trainX,trainY)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7)
```

```
preds_dt = clf.predict(testX)
preds_dt
```

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

```
cnf = confusion_matrix(testY,preds_dt)
cnf
```

```
array([[5, 4],
       [3, 5]])
```

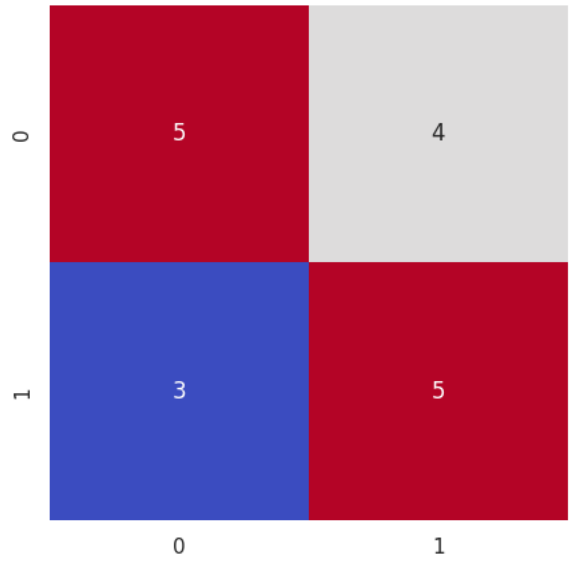
```
sensitivity=29/(29+10)
sensitivity
```

```
0.7435897435897436
```

```
specificity=21/(21+6)
specificity
```

```
0.7777777777777778
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_dt = metrics.accuracy_score(testY,preds_dt)
acc_dt
```

```
0.5882352941176471
```

## Support Vector Machine

```
from sklearn.svm import SVC
svm_mod = SVC(probability=True)
svm_mod.fit(trainX,trainY)
```

```
SVC(probability=True)
```

```
pred_svc = svm_mod.predict(testX)
pred_svc
```

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

```
cnf = confusion_matrix(testY,pred_svc)
cnf
```

```
array([[3, 6],
       [3, 5]])
```

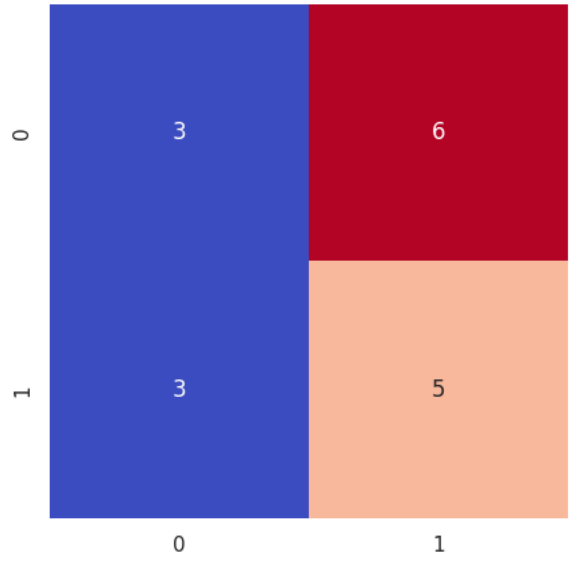
```
sensitivity= 33/(33+4)
sensitivity
```

```
0.8918918918918919
```

```
specificity=27/(27+2)
specificity
```

```
0.9310344827586207
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_svc = metrics.accuracy_score(testY,pred_svc)
acc_svc
```

```
0.47058823529411764
```

```
with open('/content/drive/MyDrive/circle_svm_model_pkl.pkl', 'wb') as files:
    pickle.dump(svm_mod, files)
```

## Logistic Regression

```
lg_mod = LogisticRegression()
lg_mod.fit(trainX, trainY)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred_lg = lg_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_lg)
cnf
```

```
array([[6, 3],
       [5, 3]])
```

```
sensitivity=34/(34+6)
sensitivity
```

```
0.85
```

```
specificity=25/(25+1)
specificity
```

```
0.9615384615384616
```

```
acc_lg = metrics.accuracy_score(testY,pred_lg)
acc_lg
```

```
0.5294117647058824
```

## Naive Bayes

```
gnb_mod = GaussianNB()
gnb_mod.fit(trainX, trainY)
```

```
▾ GaussianNB
GaussianNB()
```

```
pred_nb = gnb_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_nb)
cnf
```

```
array([[4, 5],
       [3, 5]])
```

```
sensitivity=29/(29+15)
sensitivity
```

```
0.6590909090909091
```

```
specificity=16/(16+6)
specificity
```

```
0.7272727272727273
```

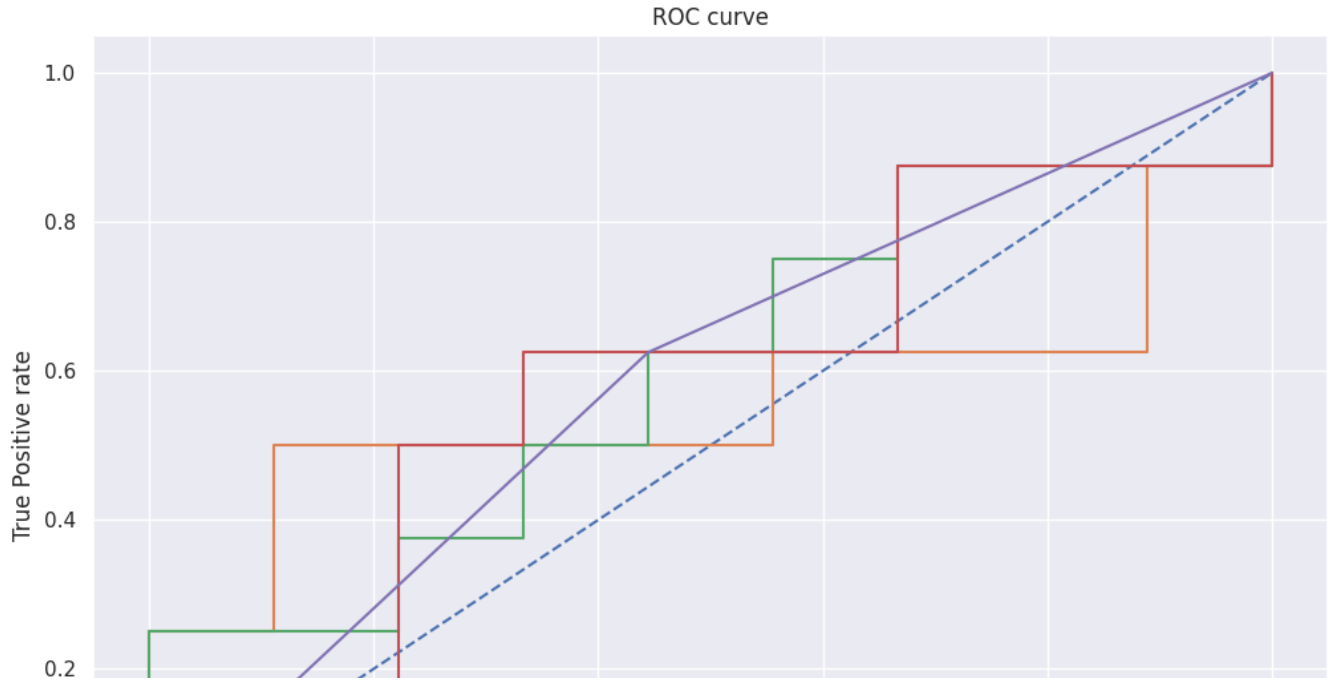
```
acc_nb = metrics.accuracy_score(testY,pred_nb)
acc_nb
```

```
0.5294117647058824
```

```
no_skill_prob = [0 for _ in range(len(testY))]
no_skill_auc = roc_auc_score(testY, no_skill_prob)
print("No Skill AUC: ", no_skill_auc)
ns_fpr, ns_tpr, _ = roc_curve(testY, no_skill_prob)

dt_prob = clf.predict_proba(testX)[:,-1]
dt_auc = roc_auc_score(testY,dt_prob)
print("DecisionTree AUC: ", dt_auc)
dt_fpr, dt_tpr, _ = roc_curve(testY, dt_prob)

svm_prob = svm_mod.predict_proba(testX)[:,-1]
svm_auc = roc_auc_score(testY,svm_prob)
print("Support Vector Machine AUC: ", svm_auc)
svm_fpr, svm_tpr, _ = roc_curve(testY, svm_prob)

lg_prob = lg_mod.predict_proba(testX)[:,-1]
lg_auc = roc_auc_score(testY, lg_prob)
print("Logistic Regression AUC: ",lg_auc)
lg_fpr, lg_tpr, _ = roc_curve(testY, lg_prob)

nb_prob = gnb_mod.predict_proba(testX)[:,-1]
nb_auc = roc_auc_score(testY, nb_prob)
print("Gaussian Naive Bayes AUC: ", nb_auc)
nb_fpr, nb_tpr, _ = roc_curve(testY, nb_prob)
```

```
fig,ax = plt.subplots(figsize=(12,8))
ax.plot(ns_fpr, ns_tpr, linestyle= '--', label= 'Reference')
ax.plot(svm_fpr, svm_tpr, linestyle= '-', label= 'SupportVectorMachine')
ax.plot(lg_fpr, lg_tpr, linestyle='-', label= 'LogisticRegression')
ax.plot(nb_fpr, nb_tpr, linestyle='-', label= 'GaussianNB')
ax.plot(dt_fpr, dt_tpr, linestyle='-', label= 'Decision Tree')

plt.xlabel('False Positve rate')
plt.ylabel('True Positive rate')
plt.title('ROC curve')
```

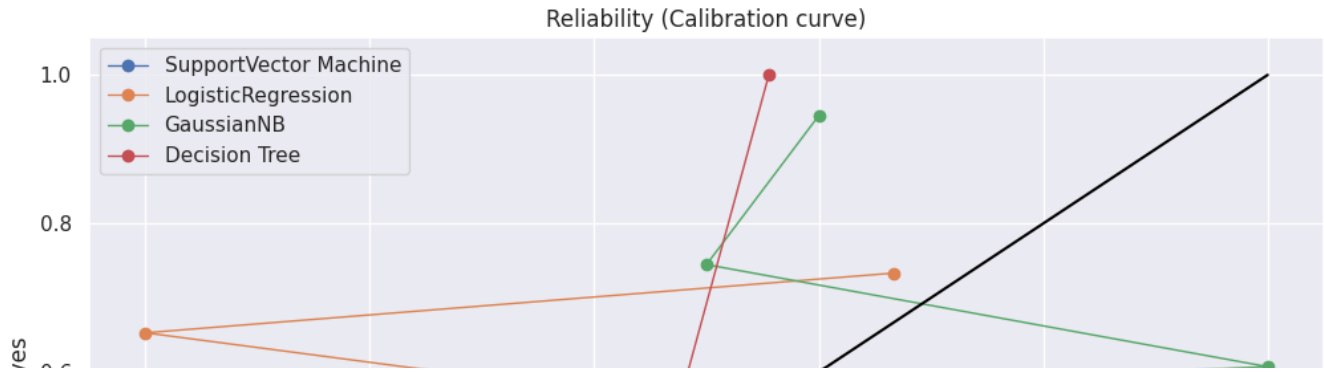
```
No Skill AUC:  0.5
DecisionTree AUC:  0.5902777777777778
Support Vector Machine AUC:  0.5555555555555556
Logistic Regression AUC:  0.5972222222222222
Gaussian Naive Bayes AUC:  0.5833333333333334
Text(0.5, 1.0, 'ROC curve')
```




```
sv_x, sv_y = calibration_curve(testY, svm_prob, n_bins=10)
lg_x, lg_y = calibration_curve(testY, lg_prob, n_bins=10)
nb_x, nb_y = calibration_curve(testY, nb_prob, n_bins=10)
dt_x, dt_y = calibration_curve(testY,dt_prob,n_bins=10)
```

```
fig, ax = plt.subplots(figsize=(12,8))

ax.plot(sv_x, sv_y, marker= 'o', linewidth= '1', label= 'SupportVector Machine')
ax.plot(lg_x, lg_y, marker= 'o', linewidth= '1', label= 'LogisticRegression')
ax.plot(nb_x, nb_y, marker= 'o', linewidth= '1', label= 'GaussianNB')
ax.plot(dt_x, dt_y, marker= 'o', linewidth= '1', label= 'Decision Tree')
```

```
line = mlines.Line2D([0,1],[0,1], color='black')
ax.add_line(line)
ax.legend()
plt.xlabel('Mean predicted probability')
plt.ylabel('Fraction of positives')
plt.title('Reliability (Calibration curve)')
plt.show()
```




```
circle_model = {
    'Random Forest': model,
    'Support Vector Machine': svm_mod,
    'Decision tree':clf,
    'Gaussian Naive Bayes': gnb_mod,
    'Logistic Regression': lg_mod

}

columns = [
    'accuracy'
]

circle_table = {}


for model in circle_model.keys():

    circle_pred = circle_model[model].predict(testX)
    accuracy = circle_model[model].score(testX,testY)
    circle_table[model] = [accuracy]


circle_results = pd.DataFrame.from_dict(circle_table, orient='index')
circle_results.columns = columns
circle_results
```

| | accuracy |
|---|---|
| **Random Forest** | 0.588235 |
| **Support Vector Machine** | 0.470588 |
| **Decision tree** | 0.588235 |
| **Gaussian Naive Bayes** | 0.529412 |
| **Logistic Regression** | 0.529412 |

Double-click (or enter) to edit

```
#Combinied HOG and LBP


from skimage import feature
import numpy as np

def quantify_image_combined(image):
    # Extract HOG features
    hog_features = feature.hog(image, orientations=9, pixels_per_cell=(10, 10),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm="L1")

    # Extract LBP features
    lbp_features = feature.local_binary_pattern(image, P=24, R=8, method="uniform")
    (hist, _) = np.histogram(lbp_features.ravel(), bins=np.arange(0, 27), range=(0, 26))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)

    # Combine HOG and LBP features
    combined_features = np.concatenate((hog_features, hist))

    return combined_features
```

```
trainX = []
testX = []
outputs = []
trainY = []
testY = []

for i in circle_train_healthy:
    image_path = (fp_circle_train_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image =cv2.threshold(image, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_combined(image)
    trainX.append(features)
    trainY.append('healthy')
    # ... rest of your code ...



for i in circle_train_park:

    image_path = (fp_circle_train_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_combined(image)
    trainX.append(features)
    trainY.append('parkinson')

for i in circle_test_healthy:

    image_path = (fp_circle_test_healthy + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_combined(image)
    testX.append(features)
    testY.append('healthy')

for i in circle_test_park:


    image_path = (fp_circle_test_park + i)
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        continue
    outputs.append(image)
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image ,0,255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image_combined(image)
    testX.append(features)
    testY.append('parkinson')
```

Double-click (or enter) to edit

```
trainX = np.array(trainX)
testX = np.array(testX)
trainY = np.array(trainY)
testY = np.array(testY)
```

```
trainX
```

```
    array([[0.      , 0.      , 0.      , ..., 0.00905 , 0.835325, 0.003375],
           [0.      , 0.      , 0.      , ..., 0.      , 0.824925, 0.165275],
           [0.      , 0.      , 0.      , ..., 0.0016  , 0.839225, 0.082175],
           ...,
           [0.      , 0.      , 0.      , ..., 0.020175, 0.7158  , 0.1384  ],
           [0.      , 0.      , 0.      , ..., 0.030225, 0.671425, 0.209675],
           [0.      , 0.      , 0.      , ..., 0.00295 , 0.6785  , 0.2892  ]])
```

```
testX
```

```
    array([[0.      , 0.      , 0.      , ..., 0.001525, 0.819125, 0.09525 ],
           [0.      , 0.      , 0.      , ..., 0.009825, 0.8033  , 0.015275],
           [0.      , 0.      , 0.      , ..., 0.00635 , 0.784525, 0.079675],
           ...,
           [0.      , 0.      , 0.      , ..., 0.001125, 0.783025, 0.15435 ],
           [0.      , 0.      , 0.      , ..., 0.0121  , 0.8008  , 0.057775],
           [0.      , 0.      , 0.      , ..., 0.019525, 0.69385 , 0.195875]])
```

```
trainY
```

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson'],
          dtype='<U9')
```

```
testY
```

```
    array(['healthy', 'healthy', 'healthy', 'healthy', 'healthy', 'healthy',
           'healthy', 'healthy', 'healthy', 'parkinson', 'parkinson',
           'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
           'parkinson'], dtype='<U9')
```

```
le = LabelEncoder()
trainY = le.fit_transform(trainY)
testY = le.transform(testY)
indexes = np.random.randint(0,30,25)
indexes
labels = []

for i in indexes:
  preds = le.inverse_transform(trainY)[i]
  labels.append(trainY)
results = []

for i in range(25):
  image = outputs[i]

  if labels[i] == 'healthy':
    color = (0,255,0)
  else:
    color = (0,0,255)

  text = str(labels[i])

  image = cv2.resize(image,(128,128))
  cv2.putText(image,text,(3,20),cv2.FONT_HERSHEY_SIMPLEX,0.5,color,2)

  results.append(image)
```

```
    <ipython-input-259-183257d9622d>:16: FutureWarning: elementwise comparison failed; returning scalar instead, but in the
      if labels[i] == 'healthy':
```

## Random Forests

```
model = RandomForestClassifier(n_estimators=100)
```

```
model.fit(trainX,trainY)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
preds = model.predict(testX)
preds
```

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

```
cnf = confusion_matrix(testY,preds)
cnf
```

```
array([[6, 3],
       [2, 6]])
```

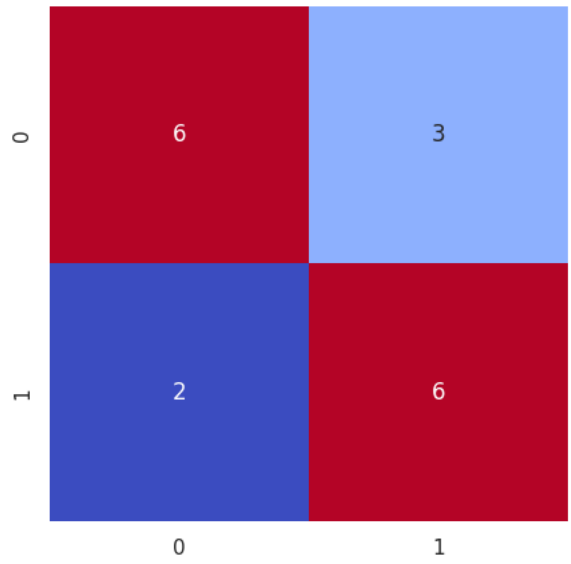
```
sensitivity= 35/(35+9)
sensitivity
```

```
0.7954545454545454
```

```
specificity= 22/(22+0)
specificity
```

```
1.0
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc = metrics.accuracy_score(testY,preds)
acc
```

```
0.7058823529411765
```

```
indexes = np.random.randint(0,30,25)
indexes
```

```
array([27, 14, 18, 16,  0, 10, 29,  2,  3, 26, 26,  2, 21, 24, 23, 17, 22,
        9,  0, 10,  7, 18,  3,  9, 17])
```

## Decision Trees

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7)
clf.fit(trainX,trainY)
```

```
▾      DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7)
```

```
preds_dt = clf.predict(testX)
preds_dt
```

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

```
cnf = confusion_matrix(testY,preds_dt)
cnf
```

```
array([[5, 4],
       [4, 4]])
```

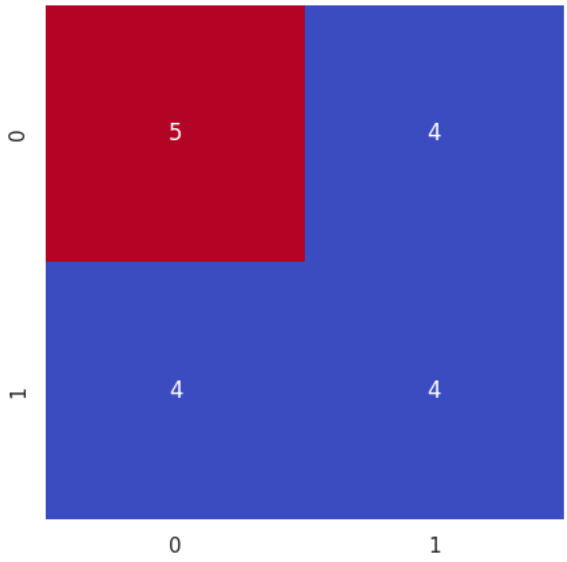
```
sensitivity=29/(29+10)
sensitivity
```

```
0.7435897435897436
```

```
specificity=21/(21+6)
specificity
```

```
0.7777777777777778
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_dt = metrics.accuracy_score(testY,preds_dt)
acc_dt
```

```
0.5294117647058824
```

## Support Vector Machine

```
from sklearn.svm import SVC
svm_mod = SVC(probability=True)
svm_mod.fit(trainX,trainY)
```

```
SVC(probability=True)
```

```
pred_svc = svm_mod.predict(testX)
pred_svc
```

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

```
cnf = confusion_matrix(testY,pred_svc)
cnf
```

```
array([[4, 5],
       [1, 7]])
```

```
sensitivity= 33/(33+4)
```

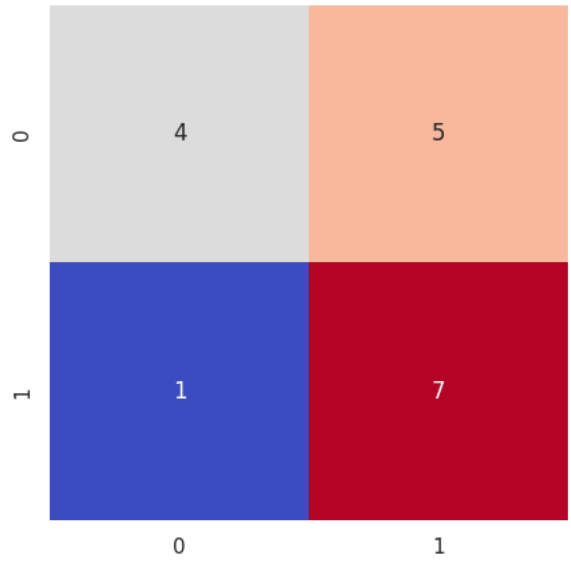
```
sensitivity=33/(33+4)
sensitivity
```

```
0.8918918918918919
```

```
specificity=27/(27+2)
specificity
```

```
0.9310344827586207
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cnf , annot=True , cmap="coolwarm" , cbar=False)
plt.show()
```




```
acc_svc = metrics.accuracy_score(testY,pred_svc)
acc_svc
```

```
0.6470588235294118
```

```
with open('/content/drive/MyDrive/circle_svm_model_pkl.pkl', 'wb') as files:
    pickle.dump(svm_mod, files)
```

## Logistic Regression

```
lg_mod = LogisticRegression()
lg_mod.fit(trainX, trainY)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred_lg = lg_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_lg)
cnf
```

```
array([[5, 4],
       [2, 6]])
```

```
sensitivity=34/(34+6)
sensitivity
```

```
0.85
```

```
specificity=25/(25+1)
specificity
```

```
0.9615384615384616
```

```
acc_lg = metrics.accuracy_score(testY,pred_lg)
acc_lg
```

## Naive Bayes

```
gnb_mod = GaussianNB()
gnb_mod.fit(trainX, trainY)
```

GaussianNB()

```
pred_nb = gnb_mod.predict(testX)
```

```
cnf = confusion_matrix(testY,pred_nb)
cnf
```

```
array([[4, 5],
       [2, 6]])
```

```
sensitivity=29/(29+15)
sensitivity
```

```
0.6590909090909091
```

```
specificity=16/(16+6)
specificity
```

```
0.7272727272727273
```

```
acc_nb = metrics.accuracy_score(testY,pred_nb)
acc_nb
```

```
0.5882352941176471
```

```
no_skill_prob = [0 for _ in range(len(testY))]
no_skill_auc = roc_auc_score(testY, no_skill_prob)
print("No Skill AUC: ", no_skill_auc)
ns_fpr, ns_tpr, _ = roc_curve(testY, no_skill_prob)

dt_prob = clf.predict_proba(testX)[:,-1]
dt_auc = roc_auc_score(testY,dt_prob)
print("DecisionTree AUC: ", dt_auc)
dt_fpr, dt_tpr, _ = roc_curve(testY, dt_prob)

svm_prob = svm_mod.predict_proba(testX)[:,-1]
svm_auc = roc_auc_score(testY,svm_prob)
print("Support Vector Machine AUC: ", svm_auc)
svm_fpr, svm_tpr, _ = roc_curve(testY, svm_prob)

lg_prob = lg_mod.predict_proba(testX)[:,-1]
lg_auc = roc_auc_score(testY, lg_prob)
print("Logistic Regression AUC: ",lg_auc)
lg_fpr, lg_tpr, _ = roc_curve(testY, lg_prob)

nb_prob = gnb_mod.predict_proba(testX)[:,-1]
nb_auc = roc_auc_score(testY, nb_prob)
print("Gaussian Naive Bayes AUC: ", nb_auc)
nb_fpr, nb_tpr, _ = roc_curve(testY, nb_prob)
```

```
fig,ax = plt.subplots(figsize=(12,8))
ax.plot(ns_fpr, ns_tpr, linestyle= '--', label= 'Reference')
ax.plot(svm_fpr, svm_tpr, linestyle= '-', label= 'SupportVectorMachine')
ax.plot(lg_fpr, lg_tpr, linestyle='-', label= 'LogisticRegression')
ax.plot(nb_fpr, nb_tpr, linestyle='-', label= 'GaussianNB')
ax.plot(dt_fpr, dt_tpr, linestyle='-', label= 'Decision Tree')

plt.xlabel('False Positve rate')
plt.ylabel('True Positive rate')
plt.title('ROC curve')
```

```
No Skill AUC:  0.5
DecisionTree AUC:  0.5277777777777778
```