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

```
spiral_train_healthy=os.listdir('/content/drive/MyDrive/Dataset/handpd/Spiral/Train/Healthy/')
spiral_train_park = os.listdir('/content/drive/MyDrive/Dataset/handpd/Spiral/Train/Parkinson/')

fp_spiral_train_healthy = '/content/drive/MyDrive/Dataset/handpd/Spiral/Train/Healthy/'
fp_spiral_train_park = '/content/drive/MyDrive/Dataset/handpd/Spiral/Train/Parkinson/'

spiral_test_healthy = os.listdir('/content/drive/MyDrive/Dataset/handpd/Spiral/Test/Healthy/')
spiral_test_park = os.listdir('/content/drive/MyDrive/Dataset/handpd/Spiral/Test/Parkinson/')

fp_spiral_test_healthy = '/content/drive/MyDrive/Dataset/handpd/Spiral/Test/Healthy/'
fp_spiral_test_park = '/content/drive/MyDrive/Dataset/handpd/Spiral/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
```

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
```

4. Image Preprocessing and Feature Extraction For Hog FE Technique

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

for i in spiral_train_healthy:
    image_path = (fp_spiral_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 spiral_train_park:

    image_path = (fp_spiral_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 spiral_test_healthy:

    image_path = (fp_spiral_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 spiral_test_park:


    image_path = (fp_spiral_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', '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', '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',
       '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', 'healthy', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', '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', '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', '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, 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, 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, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

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

HOG

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

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

```
    array([[35,  0],
           [ 7, 24]])
```

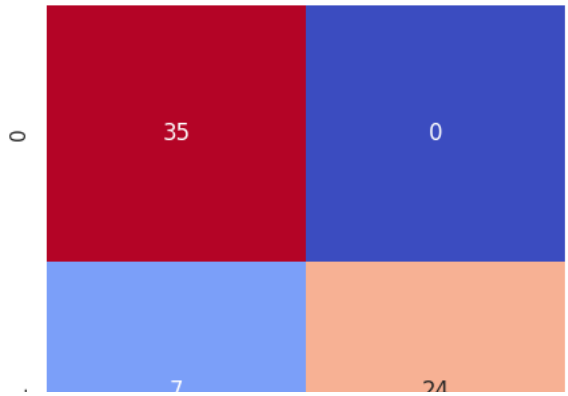
```
sensitivity= 35/(35+9)
sensitivity
```

```
    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
```

```
0.8939393939393939
```

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

```
array([25, 23, 22,  1, 23,  5,  5,  3, 11, 23,  8, 15, 23, 27, 22, 12,  8,
       15,  8, 22,  3, 23, 26, 17,  8])
```

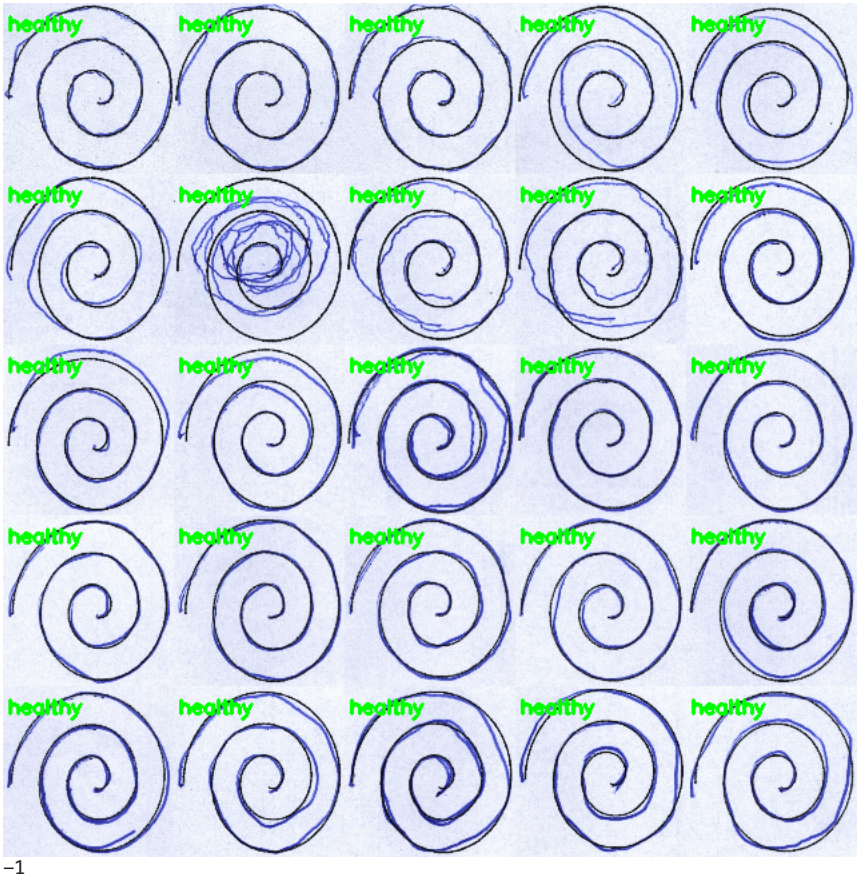
```
labels = []

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

```
labels
```

```
['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']
```

```
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)
```

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

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




```
-1
```

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

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

```
array([[31,  4],
       [13, 18]])
```

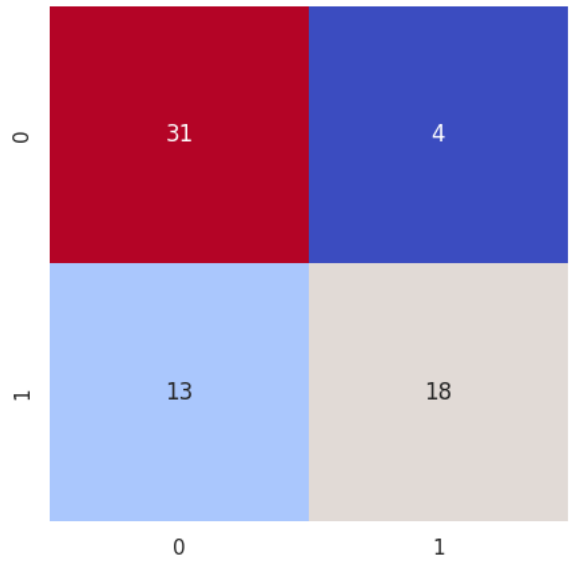
```
sensitivity=29/(29+10)
sensitivity
```

```
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
```

```
0.7424242424242424
```

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

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

```
array([[33,  2],
       [ 4, 27]])
```

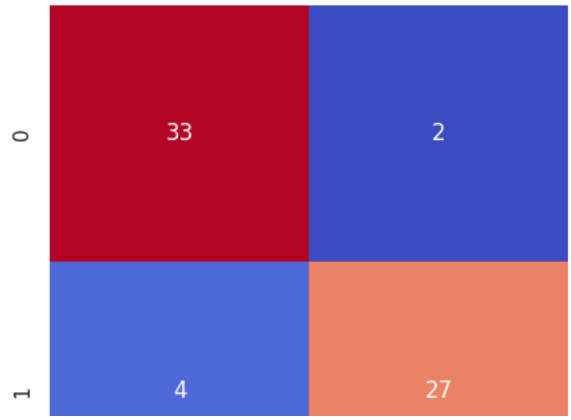
```
sensitivity= 33/(33+4)
sensitivity
```

```
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
```

```
0.9090909090909091
```

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

Logistic Regression

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

```
LogisticRegression()
```

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

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

```
array([[34,  1],
       [ 6, 25]])
```

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

```
0.85
```

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

```
0.9615384615384616
```

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

```
0.8939393939393939
```

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([[29,  6],
           [15, 16]])
```

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

```
    0.6590909090909091
```

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

```
    0.7272727272727273
```

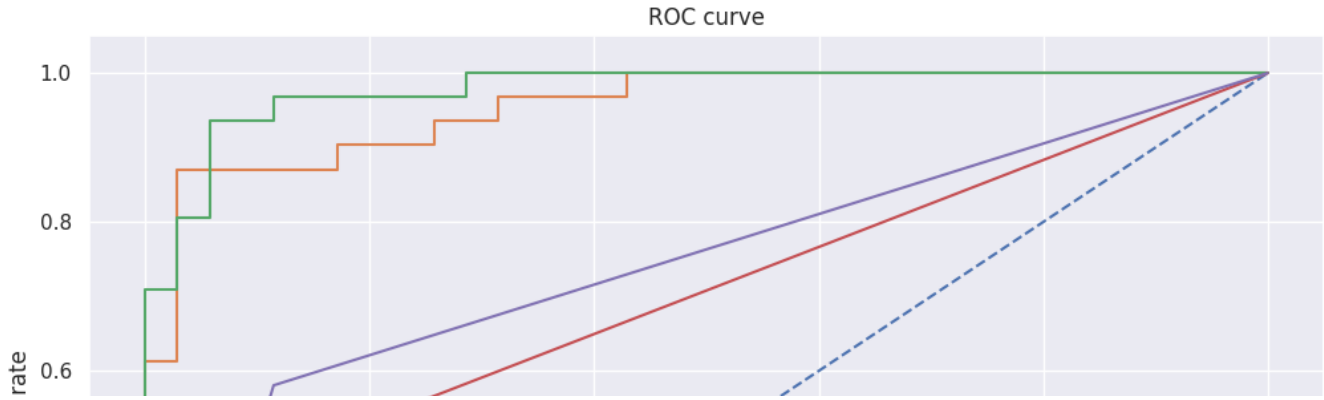
```
acc_nb = metrics.accuracy_score(testY,pred_nb)
acc_nb
```

```
    0.6818181818181818
```

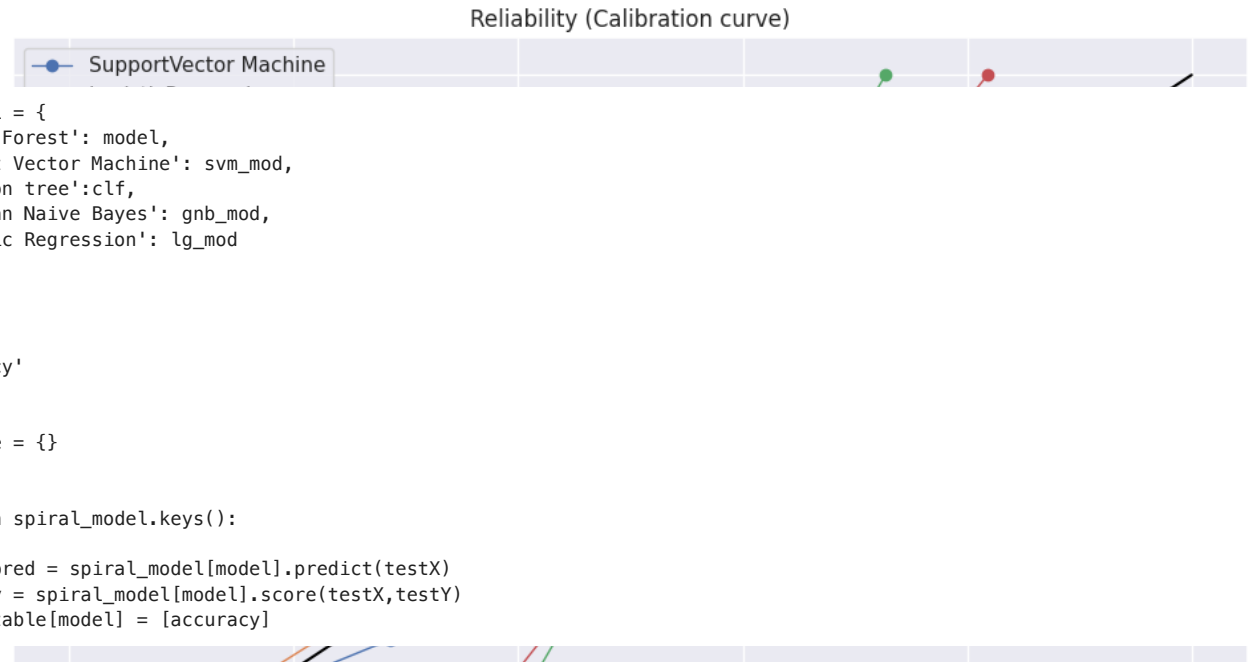
```
no_skill_prob = [0 for _ in range(len(testY))]
no_skill_auc = roc_auc_score(testY, no_skill_prob)
print("No Skill AUC: ", no_skill_auc)
ns_fpr, ns_tpr, _ = roc_curve(testY, no_skill_prob)

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:  0.7331797235023041
Support Vector Machine AUC:  0.9548387096774192
Logistic Regression AUC:  0.9769585253456221
Gaussian Naive Bayes AUC:  0.6723502304147465
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()
```




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

}

columns = [
    'accuracy'
]

spiral_table = {}


for model in spiral_model.keys():

    spiral_pred = spiral_model[model].predict(testX)
    accuracy = spiral_model[model].score(testX,testY)
    spiral_table[model] = [accuracy]
```

```
spiral_results = pd.DataFrame.from_dict(spiral_table, orient='index')
spiral_results.columns = columns
spiral_results
```

	accuracy
Random Forest	0.893939
Support Vector Machine	0.909091
Decision tree	0.742424
Gaussian Naive Bayes	0.681818
Logistic Regression	0.893939

Local Binary Patterns

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

for i in spiral_train_healthy:
    image_path = (fp_spiral_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 spiral_train_park:

    image_path = (fp_spiral_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 spiral_test_healthy:

    image_path = (fp_spiral_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 spiral_test_park:


    image_path = (fp_spiral_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([[7.50000e-05, 4.22500e-03, 2.55000e-03, ..., 0.00000e+00,
        9.00950e-01, 9.18750e-02],
       [3.50000e-04, 3.77500e-03, 3.07500e-03, ..., 0.00000e+00,
        8.89375e-01, 1.02850e-01],
       [3.75000e-04, 4.67500e-03, 2.80000e-03, ..., 0.00000e+00,
        8.96700e-01, 9.48000e-02],
       ...,
       [1.00000e-04, 3.50000e-03, 2.15000e-03, ..., 0.00000e+00,
        9.01925e-01, 9.22250e-02],
       [1.75000e-04, 4.50000e-03, 2.85000e-03, ..., 0.00000e+00,
        9.05725e-01, 8.61250e-02],
       [1.75000e-04, 4.50000e-03, 2.85000e-03, ..., 0.00000e+00,
        9.05725e-01, 8.61250e-02]])
```

testX

```
array([[2.25000e-04, 5.02500e-03, 2.87500e-03, ..., 0.00000e+00,
        9.01250e-01, 9.00750e-02],
       [4.75000e-04, 5.62500e-03, 2.12500e-03, ..., 0.00000e+00,
        8.84225e-01, 1.07025e-01],
       [3.05000e-03, 1.27500e-02, 1.07500e-03, ..., 0.00000e+00,
        8.88525e-01, 9.41250e-02],
       ...,
       [7.50000e-05, 2.62500e-03, 1.60000e-03, ..., 0.00000e+00,
        8.99050e-01, 9.65250e-02],
       [1.00000e-04, 3.50000e-03, 2.15000e-03, ..., 0.00000e+00,
        9.01925e-01, 9.22250e-02],
       [1.56250e-02, 3.56250e-02, 8.50000e-04, ..., 0.00000e+00,
        9.06250e-01, 4.09750e-02]])
```

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', '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', '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',
       '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', 'healthy', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', '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', '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', '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, 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, 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, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1])
```

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

```
array([[27,  8],
       [ 8, 23]])
```

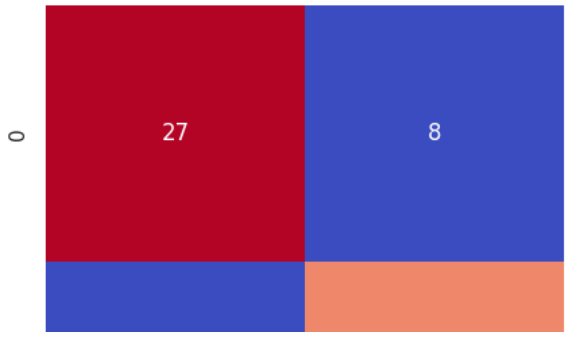
```
sensitivity= 35/(35+9)
sensitivity
```

```
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
```

```
0.7575757575757576
```



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

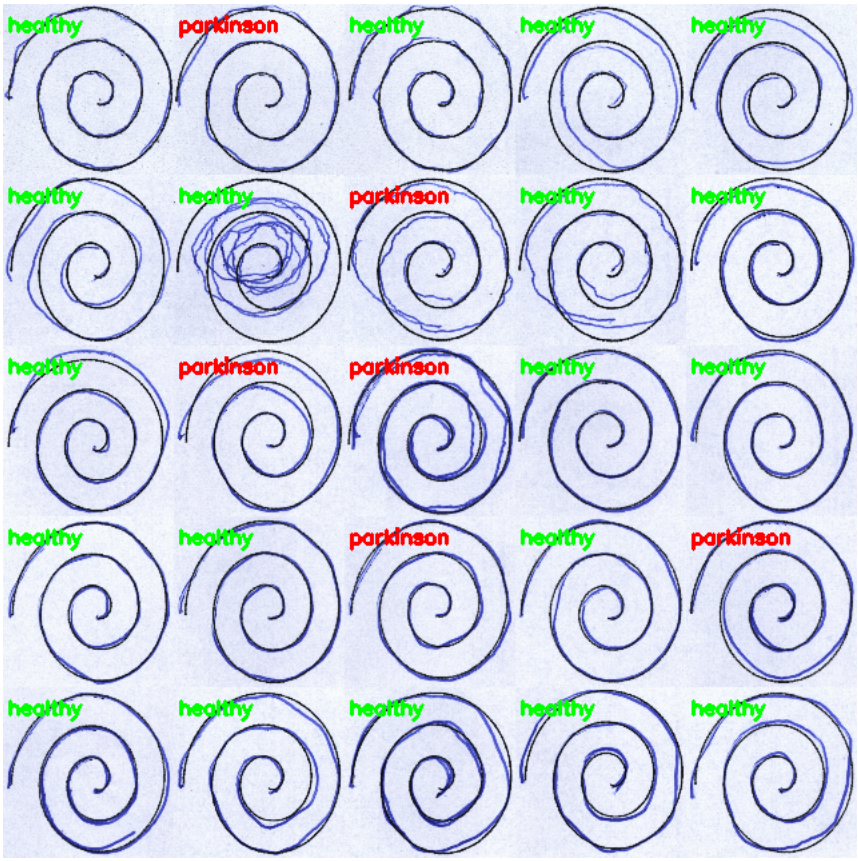
```
array([11,  2, 18, 24, 11,  7, 17, 28, 23, 16,  1, 21, 12, 27, 18, 19, 29,
        2,  7,  5, 11,  1, 15, 24, 11])
```

```
labels = []

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

```
labels
```

```
['healthy',
 'parkinson',
 'healthy',
 'healthy',
 'healthy',
 'healthy',
 'healthy',
 'parkinson',
 'healthy',
 'healthy',
 'healthy',
 'parkinson',
 'parkinson',
 'healthy',
 'healthy',
 'healthy',
 'healthy',
 'parkinson',
 'healthy',
 'parkinson',
 'healthy',
 'healthy',
 'healthy',
 'healthy',
 'healthy']
```

```
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)



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

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



-1

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

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

```
array([[24, 11],
       [ 3, 28]])
```

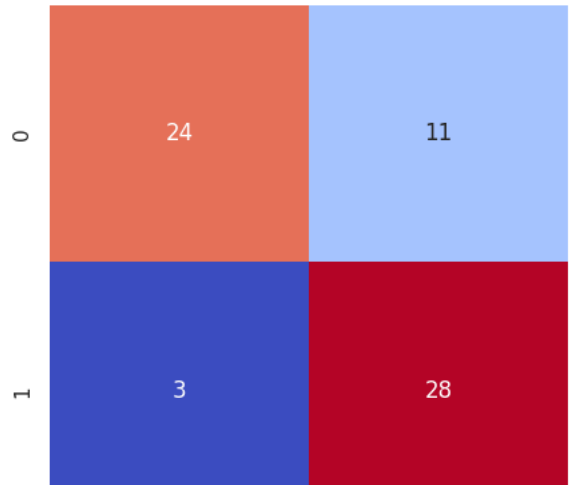
```
sensitivity=29/(29+10)
sensitivity
```

```
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
```

```
0.7878787878787878
```

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, 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])
```

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

```
array([[35,  0],
       [31,  0]])
```

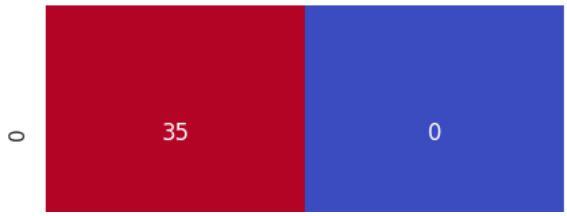
```
sensitivity= 33/(33+4)
sensitivity
```

```
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
```

```
0.5303030303030303
```

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

Logistic Regression

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

```
LogisticRegression()
```

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

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

```
array([[35,  0],
       [31,  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
```

```
0.5303030303030303
```

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([[35,  0],
       [28,  3]])
```

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

```
0.6590909090909091
```

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

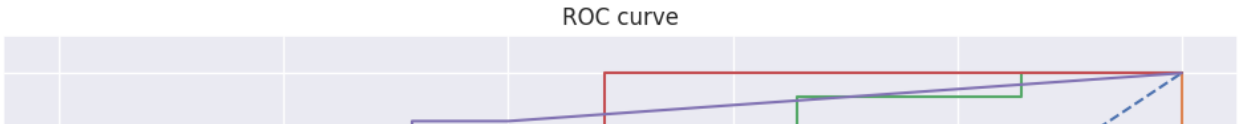
```
0.7272727272727273
```

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

```
0.5757575757575758
```

```
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')
```

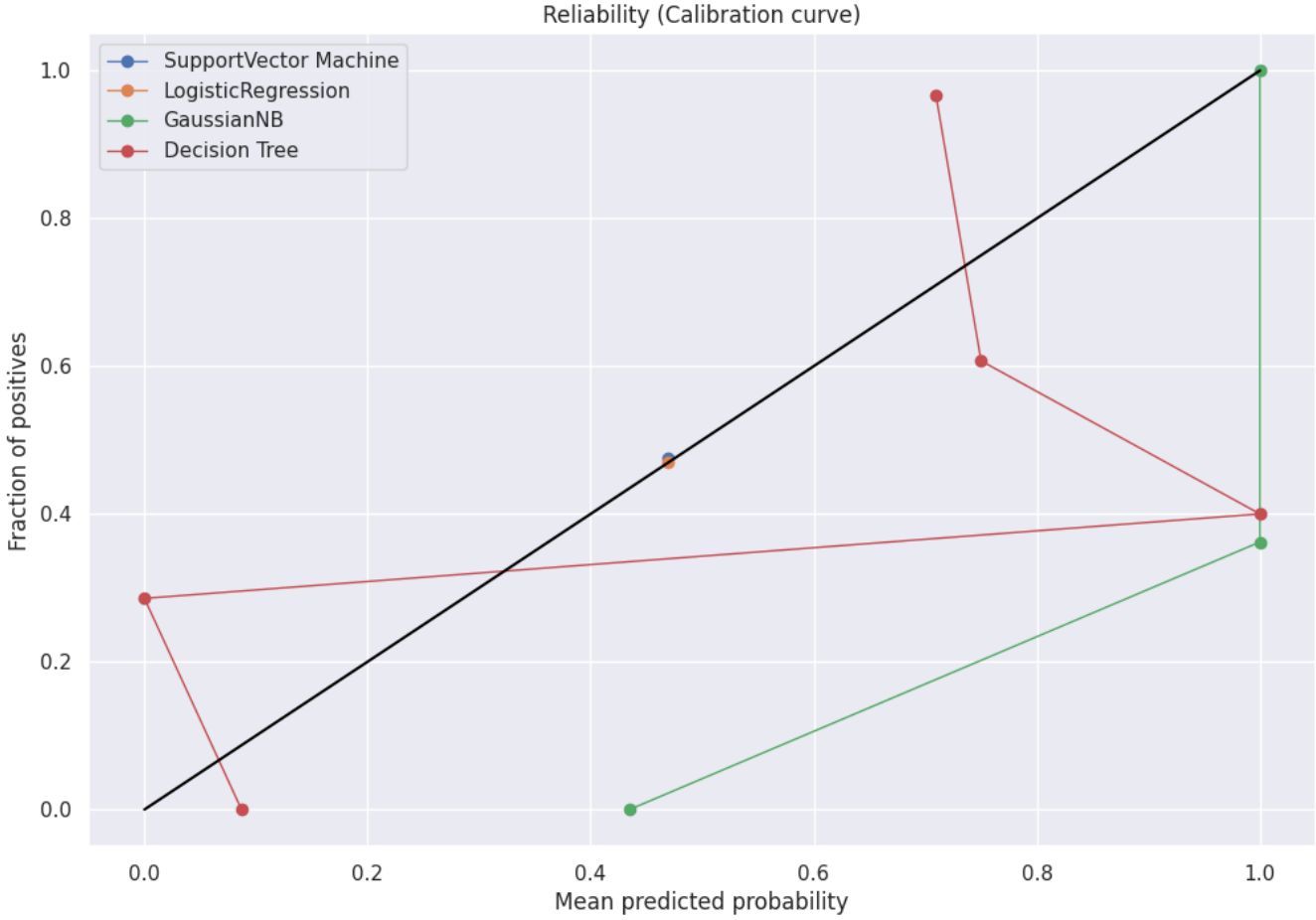
```
No Skill AUC:  0.5
DecisionTree AUC:  0.8055299539170507
Support Vector Machine AUC:  0.3050691244239631
Logistic Regression AUC:  0.6414746543778802
Gaussian Naive Bayes AUC:  0.8147465437788017
Text(0.5, 1.0, 'ROC curve')
```




```
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()
```



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

}

columns = [
    'accuracy'
]

spiral_table = {}


for model in spiral_model.keys():

    spiral_pred = spiral_model[model].predict(testX)
    accuracy = spiral_model[model].score(testX,testY)
    spiral_table[model] = [accuracy]


spiral_results = pd.DataFrame.from_dict(spiral_table, orient='index')
spiral_results.columns = columns
spiral_results
```

	accuracy
Random Forest	0.757576
Support Vector Machine	0.530303
Decision tree	0.787879
Gaussian Naive Bayes	0.575758
Logistic Regression	0.530303

```
#
```

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 spiral_train_healthy:
    image_path = (fp_spiral_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 spiral_train_park:

    image_path = (fp_spiral_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 spiral_test_healthy:

    image_path = (fp_spiral_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 spiral_test_park:


    image_path = (fp_spiral_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


     array([[ 7.74926518e-01,  3.10423317e+03,  7.34399264e-01, ...,
              2.75710697e-01, -4.93883567e-01,  6.07174457e-01],
```

```
       [ 7.53550137e-01,  3.31967231e+03,  7.42343849e-01, ...,
         2.89807051e-01, -4.99571966e-01,  6.27775767e-01],
       [ 7.65847882e-01,  3.26719796e+03,  7.30670090e-01, ...,
         2.86486580e-01, -4.86983458e-01,  6.10654912e-01],
       ...,
       [ 7.76841928e-01,  3.07968111e+03,  7.34172639e-01, ...,
         2.74078470e-01, -4.94025059e-01,  6.05639279e-01],
       [ 7.80070177e-01,  3.29268597e+03,  7.05581570e-01, ...,
         2.88012550e-01, -4.59390109e-01,  5.81716308e-01],
       [ 7.80070177e-01,  3.29268597e+03,  7.05581570e-01, ...,
         2.88012550e-01, -4.59390109e-01,  5.81716308e-01]])
```

testX

```
         2.16470424e+04,  6.82163672e-01,  7.30281334e-01,
         3.53370646e-03,  2.77245765e-01, -5.08651508e-01,
         6.24922828e-01],
       [ 6.40741325e-01,  6.43598242e+03,  6.34038352e-01,
         8.79072818e+03,  9.01024476e-01,  8.21936166e+01,
         2.87269303e+04,  9.50167459e-01,  1.04914451e+00,
         3.19662618e-03,  4.63376724e-01, -3.53298010e-01,
         5.98110491e-01],
       [ 6.30293458e-01,  6.65735329e+03,  6.31998208e-01,
         9.04247194e+03,  8.97620132e-01,  8.51322082e+01,
         2.95125345e+04,  9.71523325e-01,  1.07390477e+00,
         3.17549429e-03,  4.74065421e-01, -3.49513016e-01,
         6.00507694e-01],
       [ 7.57876653e-01,  3.97103066e+03,  6.69828224e-01,
         6.01145795e+03,  9.38931648e-01,  5.25669226e+01,
         2.00748011e+04,  6.98496648e-01,  7.59565939e-01,
         3.44390899e-03,  3.30222187e-01, -4.13243806e-01,
         5.69092935e-01],
       [ 7.58436270e-01,  4.09428974e+03,  6.55410494e-01,
         5.93900152e+03,  9.37036113e-01,  5.18523208e+01,
         1.96617164e+04,  6.97413219e-01,  7.60378075e-01,
         3.43087028e-03,  3.37811583e-01, -3.96833851e-01,
         5.57706120e-01],
       [ 7.25447567e-01,  4.32409846e+03,  6.87335806e-01,
         6.91259792e+03,  9.33502007e-01,  6.16749255e+01,
         2.33262932e+04,  7.70523277e-01,  8.37022292e-01,
         3.40709366e-03,  3.51152083e-01, -4.26837440e-01,
         6.01725691e-01],
       [ 7.67801636e-01,  3.06567481e+03,  7.48412863e-01,
         6.09061984e+03,  9.52854630e-01,  5.33505448e+01,
         2.12968045e+04,  6.73073194e-01,  7.20219289e-01,
         3.54046991e-03,  2.73228749e-01, -5.10298157e-01,
         6.22861761e-01],
       [ 7.49212633e-01,  3.37788545e+03,  7.42446892e-01,
         6.55534801e+03,  9.48053310e-01,  5.80137396e+01,
         2.28435066e+04,  7.15142516e-01,  7.67090004e-01,
         3.50691387e-03,  2.93324482e-01, -4.99322728e-01,
         6.30603089e-01],
       [ 7.62189639e-01,  3.23610662e+03,  7.38967801e-01,
         6.19655040e+03,  9.50233651e-01,  5.44039387e+01,
         2.15500950e+04,  6.86400345e-01,  7.36167460e-01,
         3.52208046e-03,  2.84333038e-01, -4.97101302e-01,
         6.19452410e-01],
       [ 7.74332797e-01,  2.92730314e+03,  7.53730088e-01,
         5.94112401e+03,  9.54982574e-01,  5.18732242e+01,
         2.08371929e+04,  6.57663779e-01,  7.02681897e-01,
         3.55557192e-03,  2.63872543e-01, -5.18889605e-01,
         6.22273939e-01],
       [ 7.76841928e-01,  3.07968111e+03,  7.34172639e-01,
         5.79053694e+03,  9.52639235e-01,  5.03959035e+01,
         2.00824666e+04,  6.53306836e-01,  7.00668330e-01,
         3.53898961e-03,  2.74078470e-01, -4.94025059e-01,
         6.05639279e-01],
       [ 7.56315893e-01,  5.13933702e+03,  5.38157803e-01,
         5.56194602e+03,  9.20964891e-01,  4.81734994e+01,
         1.71084471e+04,  6.96880198e-01,  7.75916523e-01,
         3.32370135e-03,  3.97024216e-01, -2.80356438e-01,
         4.67929095e-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', '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', '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',
       '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', 'healthy', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', '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', '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', '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, 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, 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, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 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, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
           0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1])
```

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

```
    array([[24, 11],
           [ 8, 23]])
```

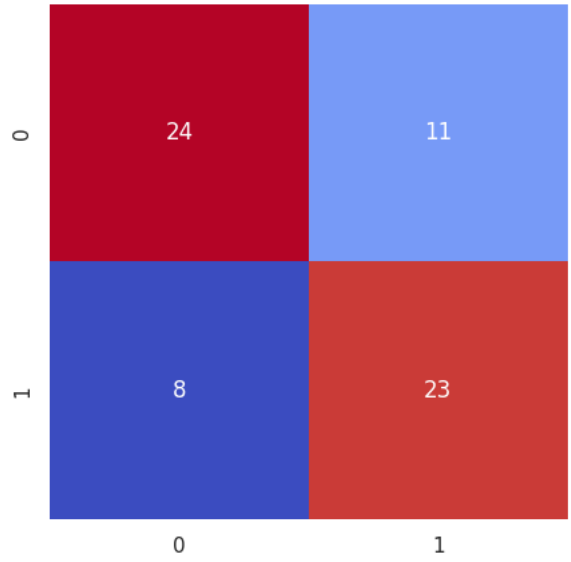
```
sensitivity= 35/(35+9)
sensitivity
```

```
    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
```

```
    0.7121212121212122
```

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

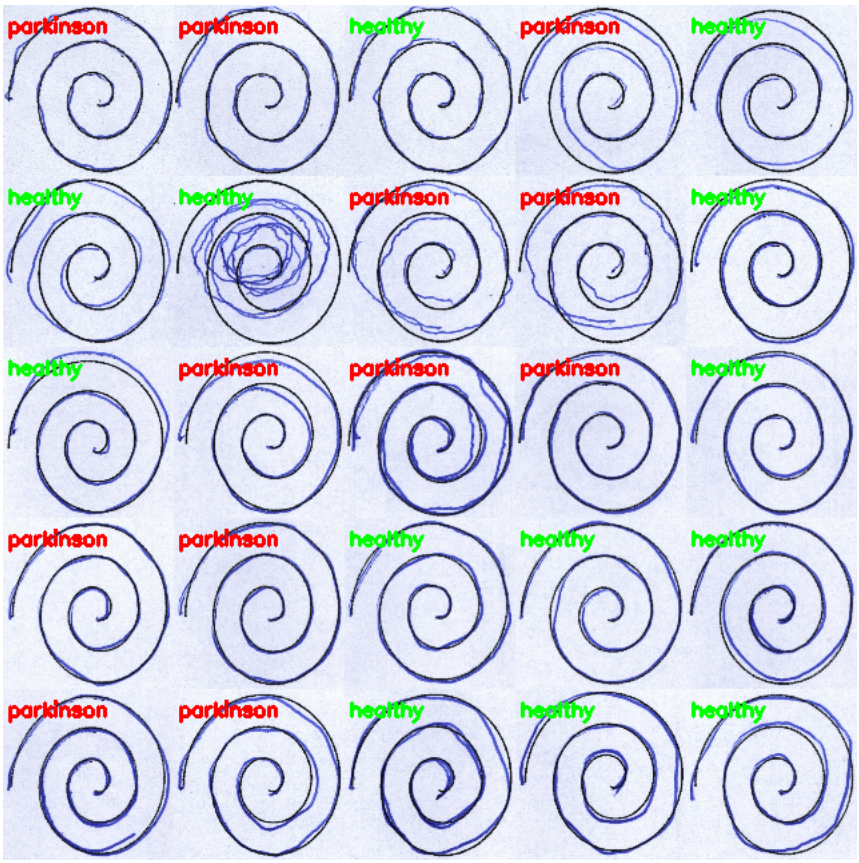
```
    array([21, 28,  9, 12, 27,  5, 22,  8, 16, 27, 15,  3,  7,  7, 20,  8, 16,
           19, 14, 27, 28, 24, 10,  9, 17])
```

```
labels = []

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

```
labels
```

```
    ['parkinson',
     'parkinson',
     'healthy',
     'parkinson',
     'healthy',
     'healthy',
     'healthy',
     'parkinson',
     'parkinson',
     'healthy',
     'healthy',
     'parkinson',
```

```
      'parkinson',
      'parkinson',
      'healthy',
      'parkinson',
      'parkinson',
      'healthy',
      'healthy',
      'healthy',
      'parkinson',
      'parkinson',
      'healthy',
      'healthy',
      'healthy']


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)



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

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




```
-1
```

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

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

```
array([[27,  8],
       [11, 20]])
```

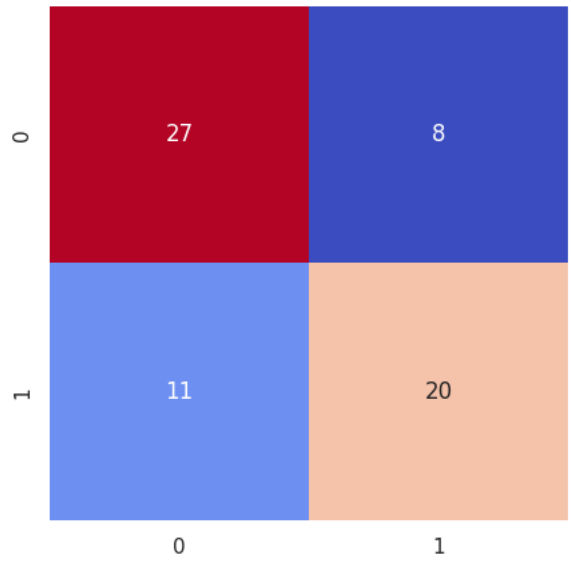
```
sensitivity=29/(29+10)
sensitivity
```

```
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
```

```
0.7121212121212122
```

Support Vector Machine

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

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

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

```
    array([[35,  0],
           [29,  2]])
```

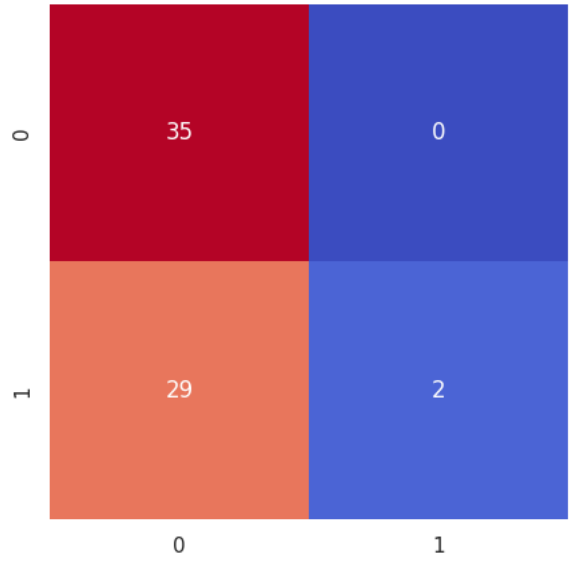
```
sensitivity= 33/(33+4)
sensitivity
```

```
    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
```

```
    0.5606060606060606
```

```
with open('/content/drive/MyDrive/spiral_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([[25, 10],
           [17, 14]])
```

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

```
0.85
```

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

```
0.9615384615384616
```

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

```
0.5909090909090909
```

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([[23, 12],
       [14, 17]])
```

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

```
0.6590909090909091
```

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

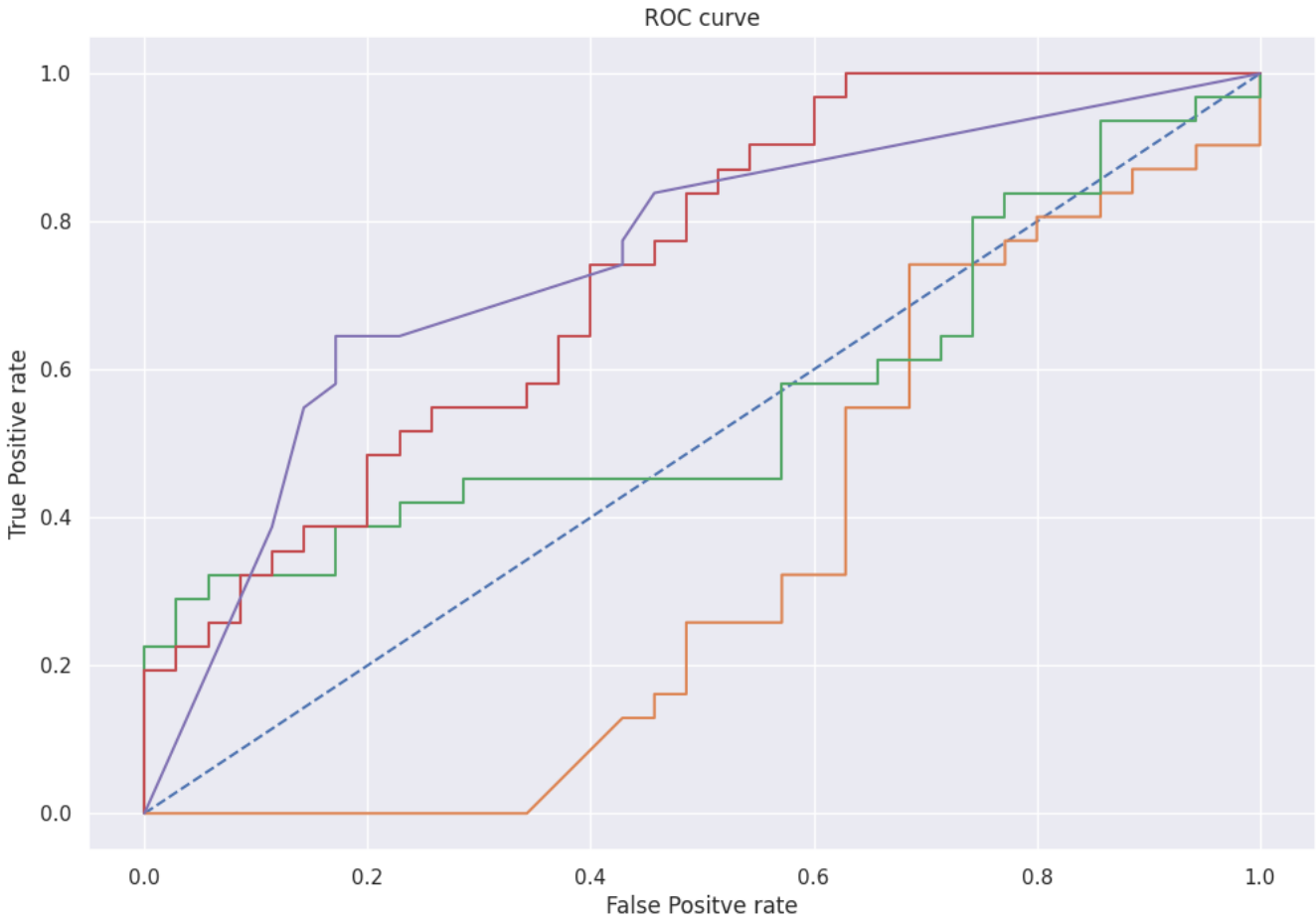
```
0.7272727272727273
```

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

```
0.6060606060606061
```

```
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')
```

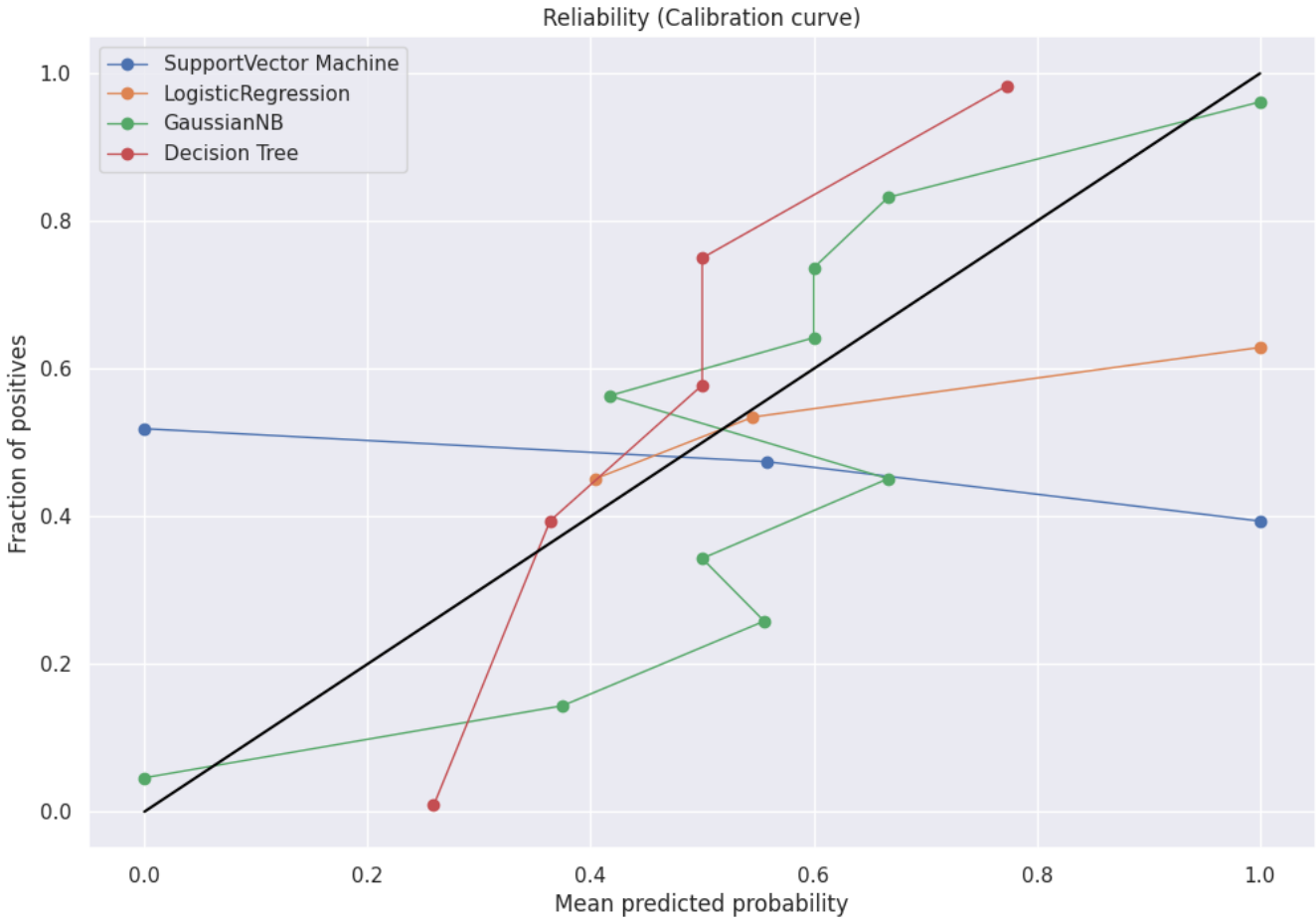
```
No Skill AUC:  0.5
DecisionTree AUC:  0.7493087557603687
Support Vector Machine AUC:  0.34285714285714286
Logistic Regression AUC:  0.56036866359447
Gaussian Naive Bayes AUC:  0.7354838709677419
Text(0.5, 1.0, 'ROC curve')
```



```
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()
```



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

}

columns = [
    'accuracy'
]

spiral_table = {}


for model in spiral_model.keys():

    spiral_pred = spiral_model[model].predict(testX)
    accuracy = spiral_model[model].score(testX,testY)
    spiral_table[model] = [accuracy]


spiral_results = pd.DataFrame.from_dict(spiral_table, orient='index')
spiral_results.columns = columns
spiral_results
```

	accuracy
Random Forest	0.712121
Support Vector Machine	0.560606
Decision tree	0.712121
Gaussian Naive Bayes	0.606061
Logistic Regression	0.590909

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 spiral_train_healthy:
    image_path = (fp_spiral_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 spiral_train_park:

    image_path = (fp_spiral_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 spiral_test_healthy:

    image_path = (fp_spiral_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 spiral_test_park:


    image_path = (fp_spiral_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.      , 0.90095 , 0.091875],
       [0.      , 0.      , 0.      , ..., 0.      , 0.889375, 0.10285 ],
       [0.      , 0.      , 0.      , ..., 0.      , 0.8967  , 0.0948  ],
       ...,
       [0.      , 0.      , 0.      , ..., 0.      , 0.901925, 0.092225],
       [0.      , 0.      , 0.      , ..., 0.      , 0.905725, 0.086125],
       [0.      , 0.      , 0.      , ..., 0.      , 0.905725, 0.086125]])
```

```
testX
```

```
array([[0.      , 0.      , 0.      , ..., 0.      , 0.90125 , 0.090075],
       [0.      , 0.      , 0.      , ..., 0.      , 0.884225, 0.107025],
       [0.      , 0.      , 0.      , ..., 0.      , 0.888525, 0.094125],
       ...,
       [0.      , 0.      , 0.      , ..., 0.      , 0.89905 , 0.096525],
       [0.      , 0.      , 0.      , ..., 0.      , 0.901925, 0.092225],
       [0.      , 0.      , 0.      , ..., 0.      , 0.90625 , 0.040975]])
```

```
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', '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', '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',
       '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', 'healthy', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkinson', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', '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', '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', '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, 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, 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, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1])
```

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

```
array([[35,  0],
       [ 8, 23]])
```

```
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
```

```
0.8787878787878788
```

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

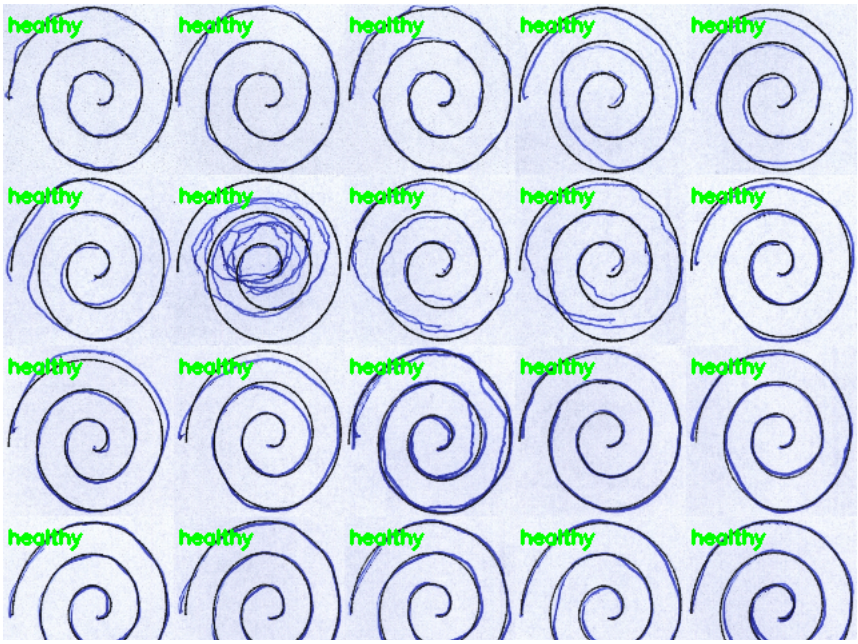
```
array([ 2,  3, 25, 24,  8, 19, 29, 27, 10, 24,  1, 28, 26, 21, 22, 28, 23,
        9, 16, 22,  2, 22, 12,  0, 17])
```

```
labels = []

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

```
labels
```

```
['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']
```

```
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)



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

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

```
array([[29,  6],
       [10, 21]])
```

```
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
```

```
0.7575757575757576
```

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

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

```
array([[33,  2],
       [ 4, 27]])
```

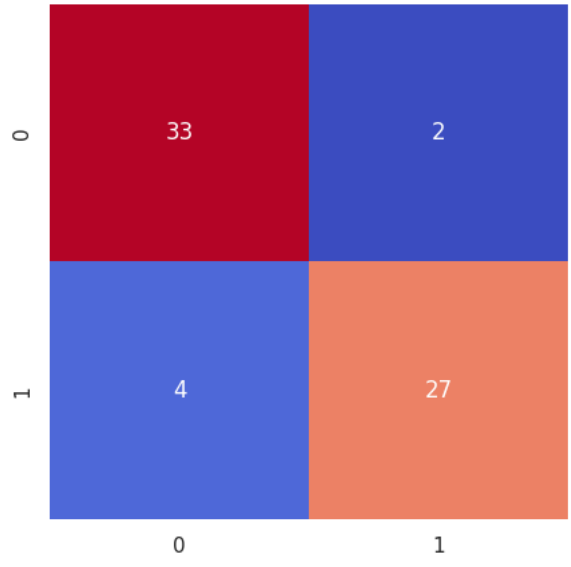
```
sensitivity= 33/(33+4)
sensitivity
```

```
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
```

```
0.9090909090909091
```

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

Logistic Regression

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression

LogisticRegression()

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

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

```
array([[34,  1],
       [ 6, 25]])
```

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

```
0.85
```

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

```
0.9615384615384616
```

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

```
0.8939393939393939
```

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([[29,  6],
       [15, 16]])
```

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

```
0.6590909090909091
```

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

```
0.7272727272727273
```

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

```
0.6818181818181818
```

```
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:  0.752995391705069
Support Vector Machine AUC:  0.9548387096774192
Logistic Regression AUC:  0.9769585253456221
Gaussian Naive Bayes AUC:  0.6723502304147465
Text(0.5, 1.0, 'ROC curve')
```