JADAVPUR UNIVERSITY

Faculty of Engineering & Technology

MACHINE LEARNING Engg. Laboratory

Name ANKUSH SIL SARMA

Class 4th YEAR 1st SEM Roll No. 002011001042

Date of Experiment 03/08/2023 Date of Submission 13/08/2023

Marks Obtained Signature of Examiner

C O – W O R K E R

NAME	ROLL

Experiment No

Commence at

Name of teacher concerned Completed at

TITLE

OBJECT

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix,roc_curve,auc
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,label_binarize
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import zero_one_loss
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA
```

```
!gdown 1xgdvvKoPpdk1mWCVSNN2UAYe5jnkrsOk

columns = ['Sepal length','Sepal width','Petal length','Petal width','Class labels']
df = pd.read_csv('iris.data',names=columns)

X = df.drop("Class labels",axis=1)
Y = df["Class labels"]

```

```
Downloading...
From: https://drive.google.com/uc?id=1xgdvvKoPpdk1mWCVSNN2UAYe5jnkrsOk
To: /content/iris.data
100% 4.55k/4.55k [00:00<00:00, 18.7MB/s]
```

```
#SVM_linear

linear_best_case = [0,0,0,0,0]
max = 0
linear_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  linear_clf = SVC(kernel='linear',random_state=10)
  linear_clf.fit(X_train,Y_train)
  Y_pred = linear_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    linear_best_case[0] = X_train
    linear_best_case[1] = X_test
    linear_best_case[2] = Y_train
    linear_best_case[3] = Y_test
    linear_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    linear_index = i
  print("--------------------------------------------------")
  print("--------------------------------------------------")

# print(linear_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[12  0  0]
 [ 0 13  2]
 [ 0  1 17]]
Accuracy:
93.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[21  0  0]
 [ 0 17  1]
 [ 0  2 19]]
Accuracy:
95.0
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[16  0  0]
 [ 0 27  4]
 [ 0  0 28]]
Accuracy:
94.66666666666667
--------------------------------------------------
```

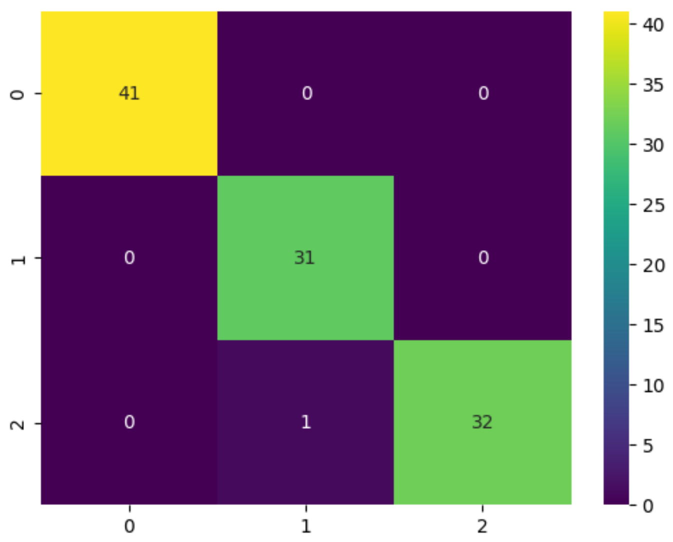
```
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[30  0  0]
 [ 0 28  1]
 [ 0  0 31]]
Accuracy:
98.88888888888889
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[41  0  0]
 [ 0 31  0]
 [ 0  1 32]]
Accuracy:
99.04761904761905
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[linear_index],":")
print(confusion_matrix(linear_best_case[3],linear_best_case[4]))
print("Performance Report:")
print(classification_report(linear_best_case[3],linear_best_case[4]))
```

```
Confusion Matrix for test size 0.7 :
[[41  0  0]
 [ 0 31  0]
 [ 0  1 32]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        41
Iris-versicolor       0.97      1.00      0.98        31
 Iris-virginica       1.00      0.97      0.98        33

       accuracy                           0.99       105
      macro avg       0.99      0.99      0.99       105
   weighted avg       0.99      0.99      0.99       105
```

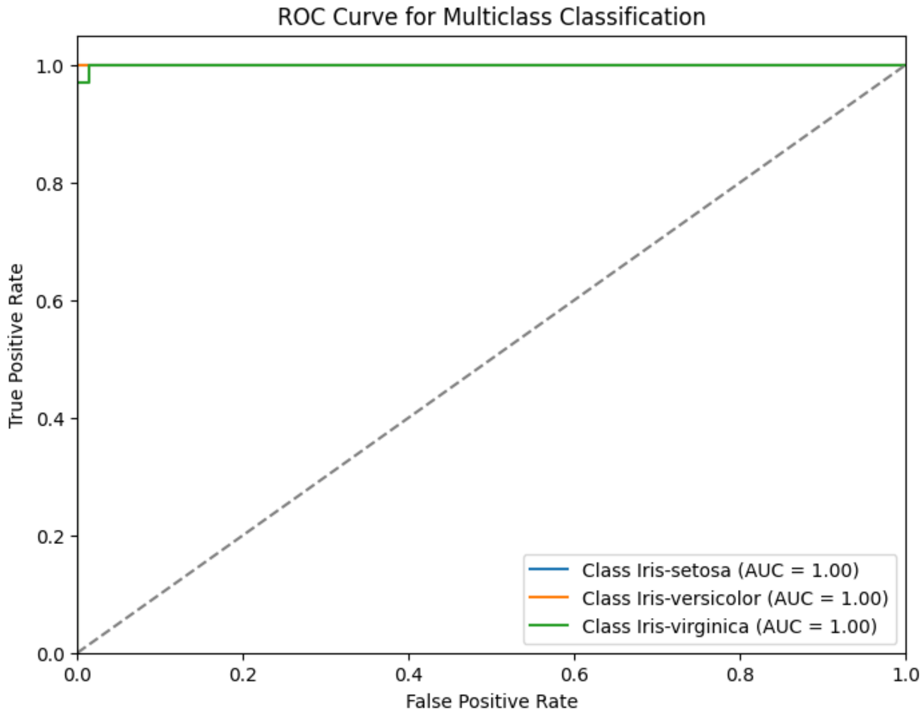
```
cm = confusion_matrix(linear_best_case[3],linear_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = linear_best_case[0],linear_best_case[1],linear_best_case[2],linear_best_case[3]
scores = None

# scores = linear_clf.predict_proba(X_test)
scores = linear_clf.decision_function(X_test)

classes = Y.unique()
n_classes = len(classes)
y_true_bin = label_binarize(y_test, classes=classes)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
```

```
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiclass Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_Polynomial

poly_best_case = [0,0,0,0,0]
max = 0
poly_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  poly_clf = SVC(kernel='poly',random_state=10)
  poly_clf.fit(X_train,Y_train)
  Y_pred = poly_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    poly_best_case[0] = X_train
    poly_best_case[1] = X_test
    poly_best_case[2] = Y_train
    poly_best_case[3] = Y_test
    poly_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    poly_index = i
  print("--------------------------------------------------")
  print("--------------------------------------------------")

# print(poly_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[16  0  0]
 [ 0 15  0]
 [ 0  5  9]]
Accuracy:
88.88888888888889
```

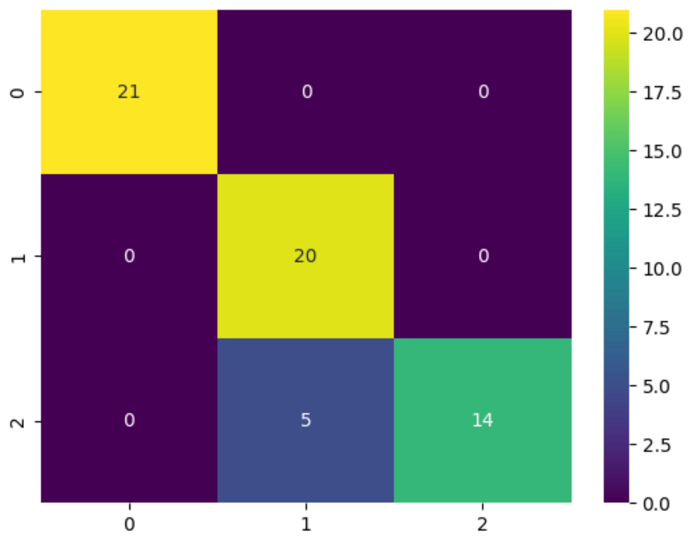
```
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[21  0  0]
 [ 0 20  0]
 [ 0  5 14]]
Accuracy:
91.66666666666666
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[24  0  0]
 [ 0 27  0]
 [ 0  8 16]]
Accuracy:
89.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[32  0  0]
 [ 0 29  1]
 [ 0  8 20]]
Accuracy:
90.0
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[33  0  0]
 [ 0 36  0]
 [ 0  9 27]]
Accuracy:
91.42857142857143
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[poly_index],":")
print(confusion_matrix(poly_best_case[3],poly_best_case[4]))
print("Performance Report:")
print(classification_report(poly_best_case[3],poly_best_case[4]))
```

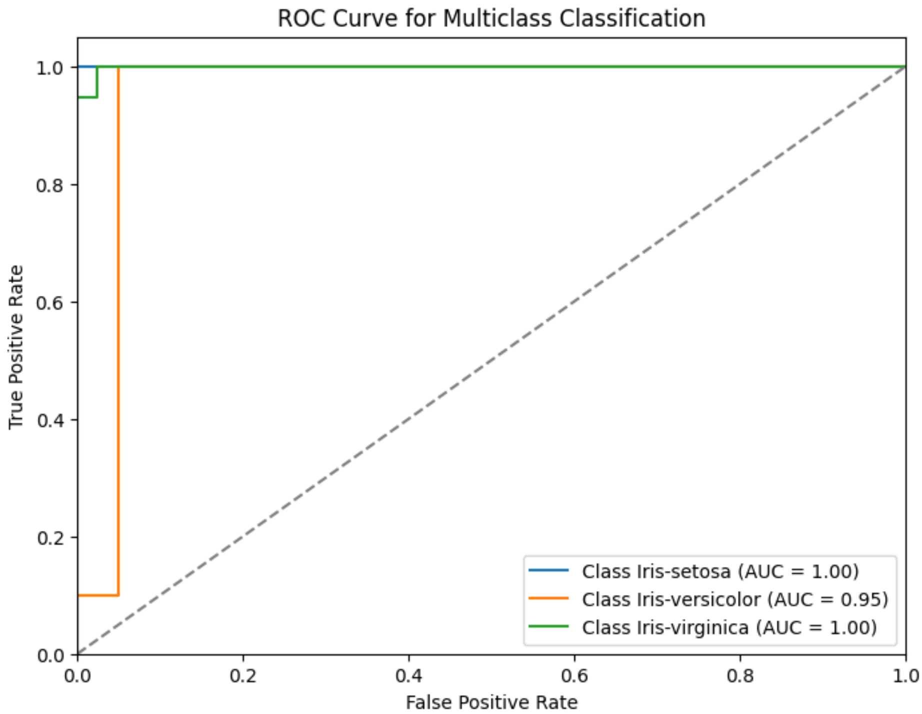
```
Confusion Matrix for test size 0.4 :
[[21  0  0]
 [ 0 20  0]
 [ 0  5 14]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        21
Iris-versicolor       0.80      1.00      0.89        20
 Iris-virginica       1.00      0.74      0.85        19

       accuracy                           0.92        60
      macro avg       0.93      0.91      0.91        60
   weighted avg       0.93      0.92      0.91        60
```

```
cm = confusion_matrix(poly_best_case[3],poly_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



```
X_train,X_test, y_train, y_test = poly_best_case[0],poly_best_case[1],poly_best_case[2],poly_best_case[3]
scores = None

# scores = poly_clf.predict_proba(X_test)
scores = poly_clf.decision_function(X_test)

classes = Y.unique()
n_classes = len(classes)
y_true_bin = label_binarize(y_test, classes=classes)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiclass Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_Gaussian

rbf_best_case = [0,0,0,0,0]
max = 0
rbf_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  rbf_clf = SVC(kernel='rbf',random_state=10)
  rbf_clf.fit(X_train,Y_train)
  Y_pred = rbf_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    rbf_best_case[0] = X_train
    rbf_best_case[1] = X_test
```

```
    rbf_best_case[2] = Y_train
    rbf_best_case[3] = Y_test
    rbf_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    rbf_index = i
  print("------------------------------------------------")
  print("------------------------------------------------")

# print(rbf_best_case[3])
```

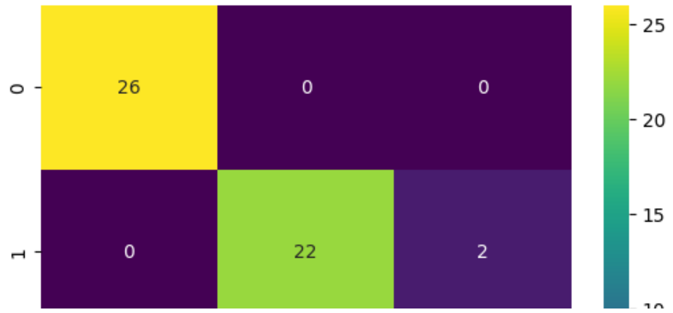
```
Confusion Matrix for test size 0.3 :
[[13  1  0]
 [ 0 14  1]
 [ 0  1 15]]
Accuracy:
93.33333333333333
------------------------------------------------
------------------------------------------------
Confusion Matrix for test size 0.4 :
[[24  0  0]
 [ 0 18  1]
 [ 0  2 15]]
Accuracy:
95.0
------------------------------------------------
------------------------------------------------
Confusion Matrix for test size 0.5 :
[[26  0  0]
 [ 0 22  2]
 [ 0  1 24]]
Accuracy:
96.0
------------------------------------------------
------------------------------------------------
Confusion Matrix for test size 0.6 :
[[32  0  0]
 [ 0 27  1]
 [ 0  3 27]]
Accuracy:
95.55555555555556
------------------------------------------------
------------------------------------------------
Confusion Matrix for test size 0.7 :
[[38  0  0]
 [ 0 29  2]
 [ 0  3 33]]
Accuracy:
95.23809523809523
------------------------------------------------
------------------------------------------------
```

```
print("Confusion Matrix for test size",size[rbf_index],":")
print(confusion_matrix(rbf_best_case[3],rbf_best_case[4]))
print("Performance Report:")
print(classification_report(rbf_best_case[3],rbf_best_case[4]))
```

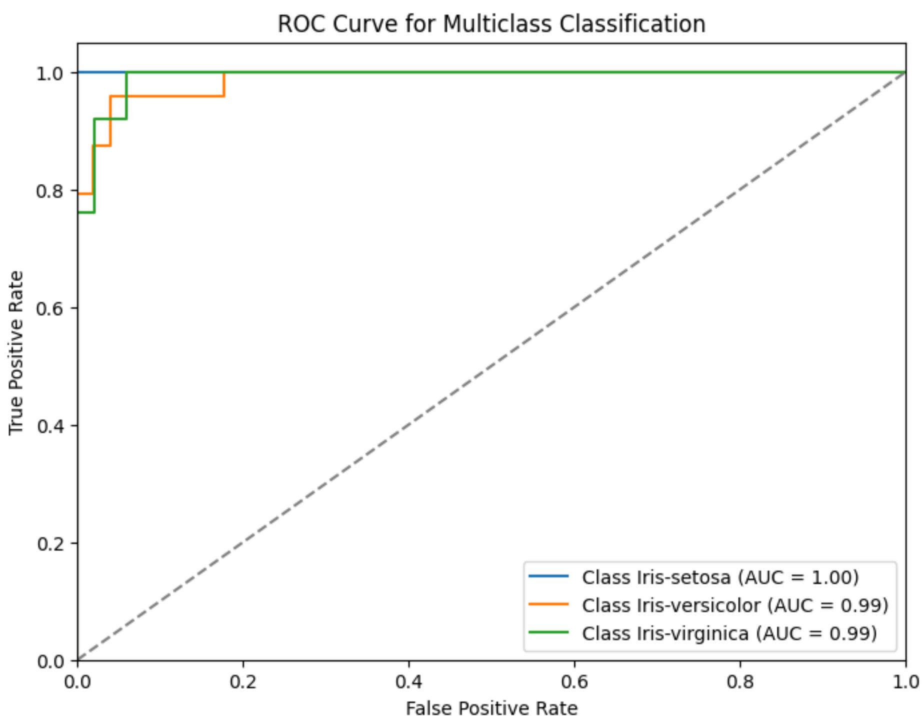
```
Confusion Matrix for test size 0.5 :
[[26  0  0]
 [ 0 22  2]
 [ 0  1 24]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        26
Iris-versicolor       0.96      0.92      0.94        24
 Iris-virginica       0.92      0.96      0.94        25

       accuracy                           0.96        75
      macro avg       0.96      0.96      0.96        75
   weighted avg       0.96      0.96      0.96        75
```

```
cm = confusion_matrix(rbf_best_case[3],rbf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = rbf_best_case[0],rbf_best_case[1],rbf_best_case[2],rbf_best_case[3]
scores = None

# scores = rbf_clf.predict_proba(X_test)
scores = rbf_clf.decision_function(X_test)

classes = Y.unique()
n_classes = len(classes)
y_true_bin = label_binarize(y_test, classes=classes)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiclass Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_sigmoid

sigmoid_best_case = [0,0,0,0,0]
max = 0
sigmoid_index = 0
for i in range(0,len(size)):
```

```
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  sigmoid_clf = SVC(kernel='sigmoid',random_state=10)
  sigmoid_clf.fit(X_train,Y_train)
  Y_pred = sigmoid_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    sigmoid_best_case[0] = X_train
    sigmoid_best_case[1] = X_test
    sigmoid_best_case[2] = Y_train
    sigmoid_best_case[3] = Y_test
    sigmoid_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    sigmoid_index = i
  print("------------------------------------------------")
  print("------------------------------------------------")

# print(sigmoid_best_case[3])
```

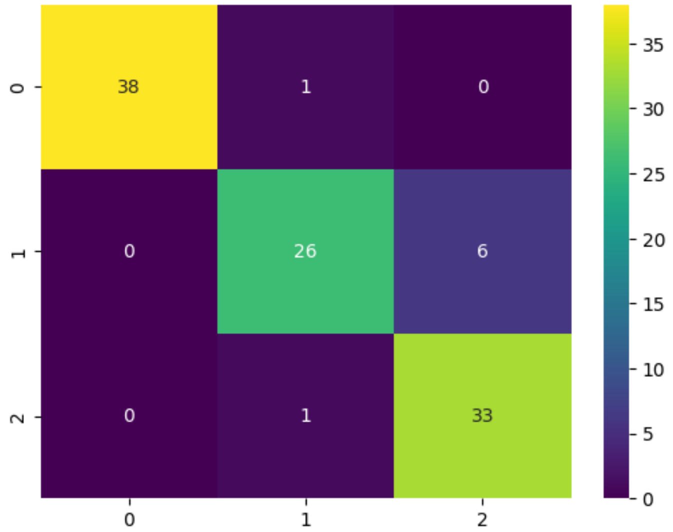
```
Confusion Matrix for test size 0.3 :
[[16  0  0]
 [ 0 11  2]
 [ 0  2 14]]
Accuracy:
91.11111111111111
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[16  0  0]
 [ 0 16  4]
 [ 0  1 23]]
Accuracy:
91.66666666666666
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[27  0  0]
 [ 0 19  1]
 [ 0  6 22]]
Accuracy:
90.66666666666666
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[28  0  0]
 [ 0 22  6]
 [ 0  5 29]]
Accuracy:
87.77777777777777
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[38  1  0]
 [ 0 26  6]
 [ 0  1 33]]
Accuracy:
92.38095238095238
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[sigmoid_index],":")
print(confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4]))
print("Performance Report:")
print(classification_report(sigmoid_best_case[3],sigmoid_best_case[4]))
```

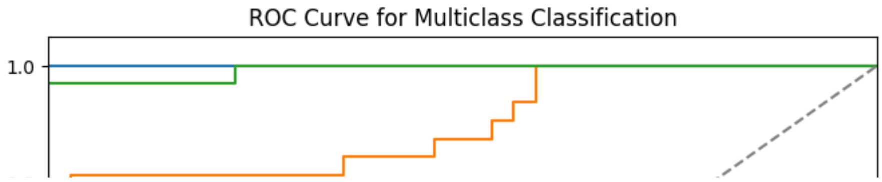
```
Confusion Matrix for test size 0.7 :
[[38  1  0]
 [ 0 26  6]
 [ 0  1 33]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      0.97      0.99        39
Iris-versicolor       0.93      0.81      0.87        32
 Iris-virginica       0.85      0.97      0.90        34

       accuracy                           0.92       105
      macro avg       0.92      0.92      0.92       105
   weighted avg       0.93      0.92      0.92       105
```

```
cm = confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = sigmoid_best_case[0],sigmoid_best_case[1],sigmoid_best_case[2],sigmoid_best_case[3]
scores = None

# scores = sigmoid_clf.predict_proba(X_test)
scores = sigmoid_clf.decision_function(X_test)

classes = Y.unique()
n_classes = len(classes)
y_true_bin = label_binarize(y_test, classes=classes)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiclass Classification')
plt.legend(loc='lower right')
plt.show()
```




```
# MLP

MLP_best_case = [0,0,0,0,0]
max = 0
MLP_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  MLP_clf = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=900)
  MLP_clf.fit(X_train,Y_train)
  Y_pred = MLP_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    MLP_best_case[0] = X_train
    MLP_best_case[1] = X_test
    MLP_best_case[2] = Y_train
    MLP_best_case[3] = Y_test
    MLP_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    MLP_index = i
  print("--------------------------------------------------")
  print("--------------------------------------------------")

# print(MLP_best_case[3])
```

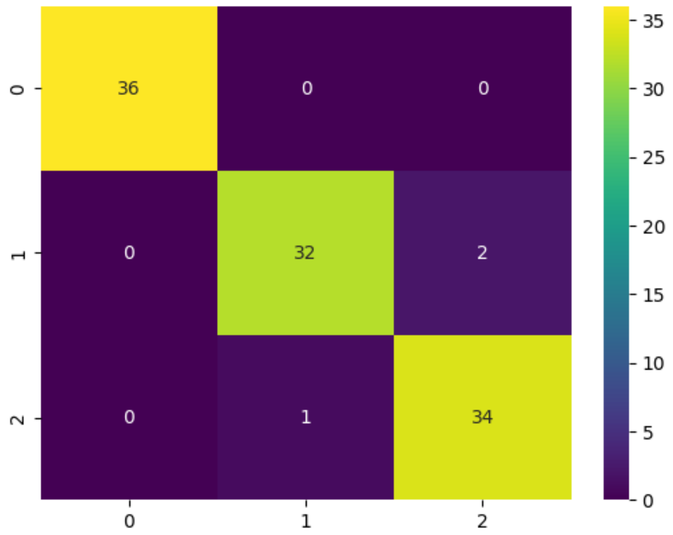
```
Confusion Matrix for test size 0.3 :
[[13  0  0]
 [ 0 16  2]
 [ 0  1 13]]
Accuracy:
93.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[20  1  0]
 [ 0 16  3]
 [ 0  0 20]]
Accuracy:
93.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[24  0  0]
 [ 0 24  1]
 [ 0  3 23]]
Accuracy:
94.66666666666667
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[34  1  0]
 [ 0 28  3]
 [ 0  2 22]]
Accuracy:
93.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[36  0  0]
 [ 0 32  2]
 [ 0  1 34]]
Accuracy:
97.14285714285714
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[MLP_index],":")
print(confusion_matrix(MLP_best_case[3],MLP_best_case[4]))
print("Performance Report:")
print(classification_report(MLP_best_case[3],MLP_best_case[4]))
```

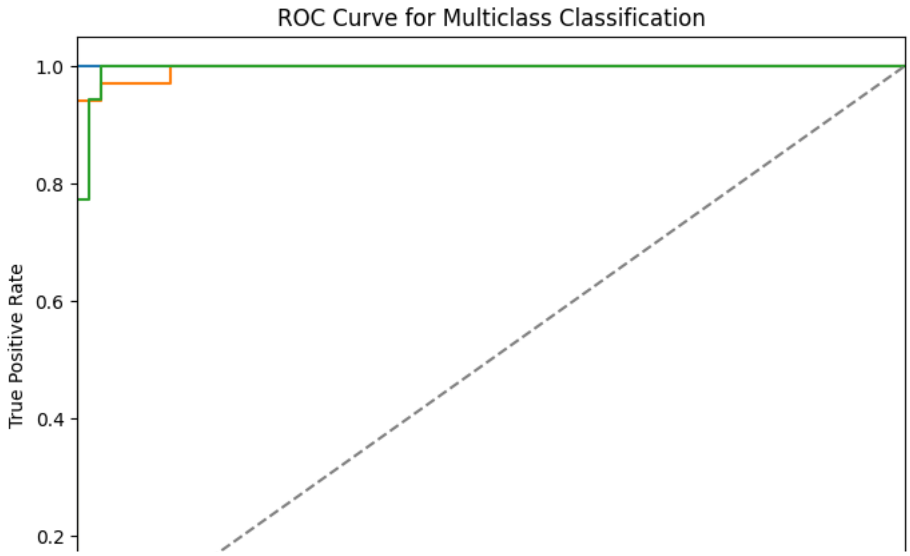
```
Confusion Matrix for test size 0.7 :
[[36  0  0]
 [ 0 32  2]
 [ 0  1 34]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        36
Iris-versicolor       0.97      0.94      0.96        34
 Iris-virginica       0.94      0.97      0.96        35

       accuracy                           0.97       105
      macro avg       0.97      0.97      0.97       105
   weighted avg       0.97      0.97      0.97       105
```

```
cm = confusion_matrix(MLP_best_case[3],MLP_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = MLP_best_case[0],MLP_best_case[1],MLP_best_case[2],MLP_best_case[3]
scores = None

scores = MLP_clf.predict_proba(X_test)
# scores = MLP_clf.decision_function(X_test)

classes = Y.unique()
n_classes = len(classes)
y_true_bin = label_binarize(y_test, classes=classes)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiclass Classification')
plt.legend(loc='lower right')
plt.show()
```




```
# Random Forest

rf_best_case = [0,0,0,0,0]
max = 0
rf_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  RFclf = RandomForestClassifier(n_estimators=20)
  RFclf.fit(X_train,Y_train)
  Y_pred = RFclf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    rf_best_case[0] = X_train
    rf_best_case[1] = X_test
    rf_best_case[2] = Y_train
    rf_best_case[3] = Y_test
    rf_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    rf_index = i
  print("--------------------------------------------------")
  print("--------------------------------------------------")

# print(rf_best_case)
```

```
Confusion Matrix for test size 0.3 :
[[15  0  0]
 [ 0 13  2]
 [ 0  0 15]]
Accuracy:
95.55555555555556
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[21  0  0]
 [ 0 18  0]
 [ 0  3 18]]
Accuracy:
95.0
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[25  0  0]
 [ 0 23  3]
 [ 0  3 21]]
Accuracy:
92.0
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[31  0  0]
 [ 0 25  1]
 [ 0  2 31]]
Accuracy:
96.66666666666667
--------------------------------------------------
```

```
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[40  0  0]
 [ 0 30  1]
 [ 0  5 29]]
Accuracy:
94.28571428571428
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

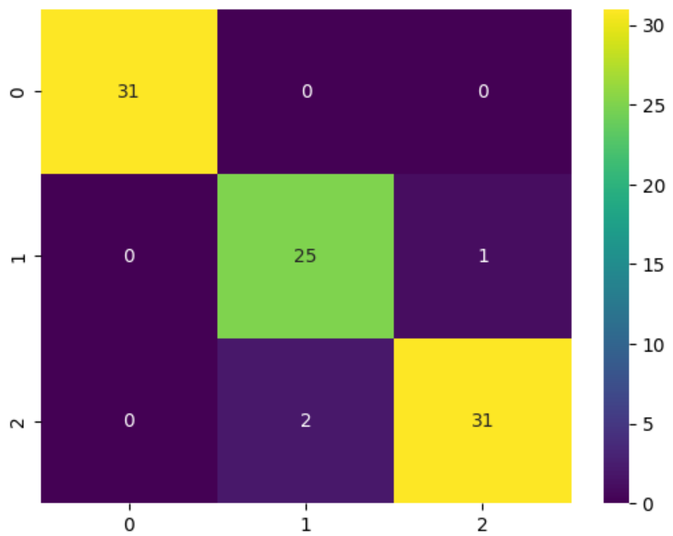
```
Confusion Matrix for test size 0.6 :
[[31  0  0]
 [ 0 25  1]
 [ 0  2 31]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        31
Iris-versicolor       0.93      0.96      0.94        26
 Iris-virginica       0.97      0.94      0.95        33

       accuracy                           0.97        90
      macro avg       0.96      0.97      0.97        90
   weighted avg       0.97      0.97      0.97        90
```


```
print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

```
Confusion Matrix for test size 0.6 :
[[31  0  0]
 [ 0 25  1]
 [ 0  2 31]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        31
Iris-versicolor       0.93      0.96      0.94        26
 Iris-virginica       0.97      0.94      0.95        33

       accuracy                           0.97        90
      macro avg       0.96      0.97      0.97        90
   weighted avg       0.97      0.97      0.97        90
```

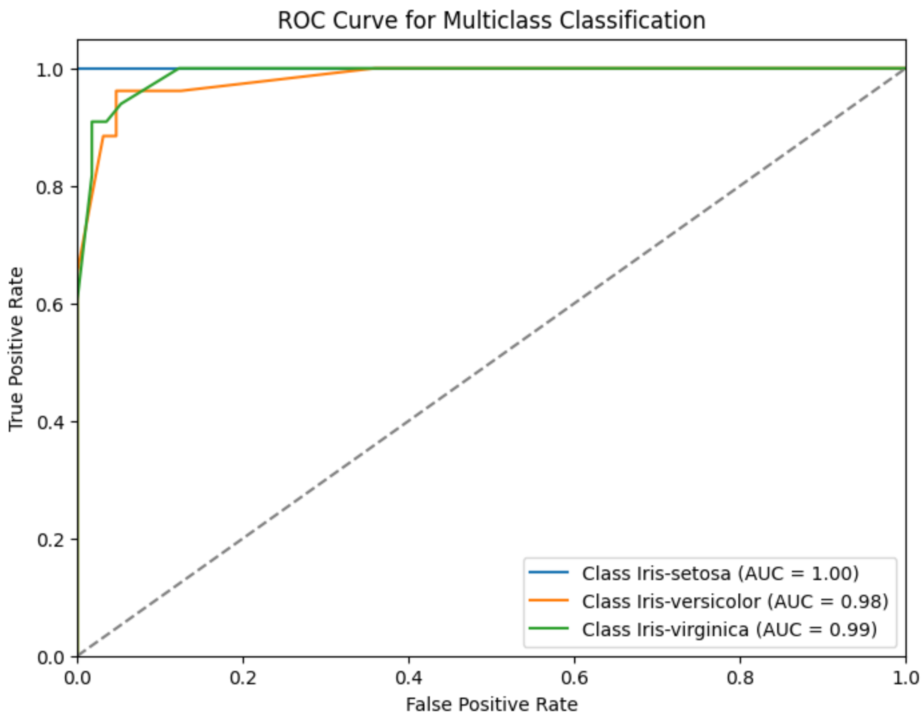

```
cm = confusion_matrix(rf_best_case[3],rf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = rf_best_case[0],rf_best_case[1],rf_best_case[2],rf_best_case[3]
scores = None

```

```
scores = RFclf.predict_proba(X_test)
# scores = RFclf.decision_function(X_test)

classes = Y.unique()
n_classes = len(classes)
y_true_bin = label_binarize(y_test, classes=classes)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiclass Classification')
plt.legend(loc='lower right')
plt.show()
```



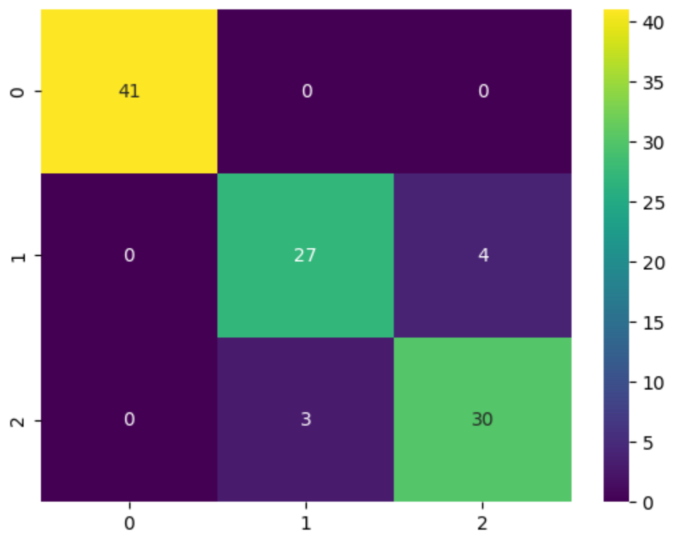
After Principal Component Analysis (PCA) for feature dimensionality reduction

```
#SVM_linear

pca = PCA(n_components = 2)
linear_best_case[0] = pca.fit_transform(linear_best_case[0])
linear_best_case[1] = pca.transform(linear_best_case[1])
linear_clf = SVC(kernel='linear',random_state=10)
linear_clf.fit(linear_best_case[0],linear_best_case[2])
linear_best_case[4] = linear_clf.predict(linear_best_case[1])

print("Confusion Matrix for test size",size[linear_index],":")
print(confusion_matrix(linear_best_case[3],linear_best_case[4]))
print("Performance Report:")
print(classification_report(linear_best_case[3],linear_best_case[4]))
```

```
Confusion Matrix for test size 0.7 :
[[41  0  0]
 [ 0 27  4]
 [ 0  3 30]]
Performance Report:
              precision    recall  f1-score   support
```

```
     Iris-setosa       1.00      1.00      1.00        41
 Iris-versicolor       0.90      0.87      0.89        31
  Iris-virginica       0.88      0.91      0.90        33

        accuracy                           0.93       105
       macro avg       0.93      0.93      0.93       105
    weighted avg       0.93      0.93      0.93       105
```

```
cm = confusion_matrix(linear_best_case[3],linear_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#SVM_Polynomial

pca = PCA(n_components = 2)
poly_best_case[0] = pca.fit_transform(poly_best_case[0])
poly_best_case[1] = pca.transform(poly_best_case[1])
poly_clf = SVC(kernel='poly',random_state=10)
poly_clf.fit(poly_best_case[0],poly_best_case[2])
poly_best_case[4] = poly_clf.predict(poly_best_case[1])

print("Confusion Matrix for test size",size[poly_index],":")
print(confusion_matrix(poly_best_case[3],poly_best_case[4]))
print("Performance Report:")
print(classification_report(poly_best_case[3],poly_best_case[4]))
```

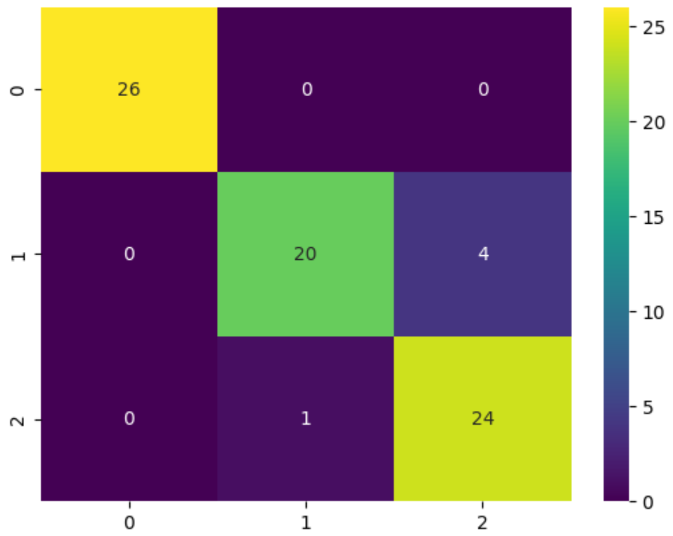
```
Confusion Matrix for test size 0.4 :
[[21  0  0]
 [ 0 20  0]
 [ 0  7 12]]
Performance Report:
                 precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        21
 Iris-versicolor       0.74      1.00      0.85        20
  Iris-virginica       1.00      0.63      0.77        19

        accuracy                           0.88        60
       macro avg       0.91      0.88      0.88        60
    weighted avg       0.91      0.88      0.88        60
```

```
cm = confusion_matrix(poly_best_case[3],poly_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```


```
#SVM_Gaussian

pca = PCA(n_components = 2)
rbf_best_case[0] = pca.fit_transform(rbf_best_case[0])
rbf_best_case[1] = pca.transform(rbf_best_case[1])
rbf_clf = SVC(kernel='rbf',random_state=10)
rbf_clf.fit(rbf_best_case[0],rbf_best_case[2])
rbf_best_case[4] = rbf_clf.predict(rbf_best_case[1])

print("Confusion Matrix for test size",size[rbf_index],":")
print(confusion_matrix(rbf_best_case[3],rbf_best_case[4]))
print("Performance Report:")
print(classification_report(rbf_best_case[3],rbf_best_case[4]))
```

```
Confusion Matrix for test size 0.5 :
[[26  0  0]
 [ 0 20  4]
 [ 0  1 24]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        26
Iris-versicolor       0.95      0.83      0.89        24
 Iris-virginica       0.86      0.96      0.91        25

       accuracy                           0.93        75
      macro avg       0.94      0.93      0.93        75
   weighted avg       0.94      0.93      0.93        75
```

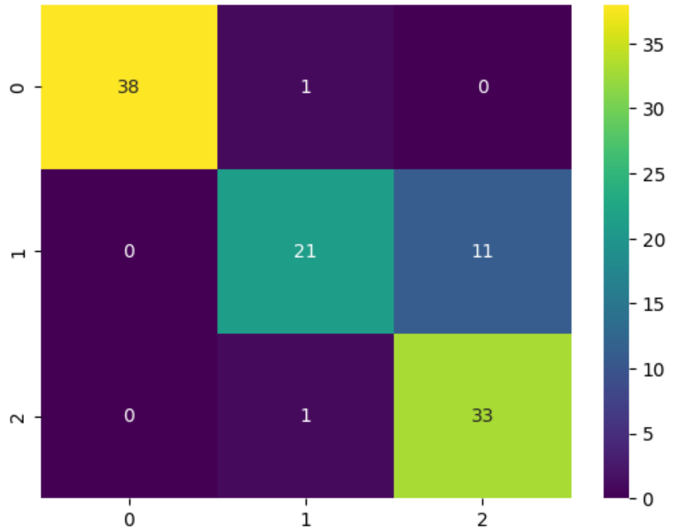
```
cm = confusion_matrix(rbf_best_case[3],rbf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#SVM_Sigmoid

pca = PCA(n_components = 2)
sigmoid_best_case[0] = pca.fit_transform(sigmoid_best_case[0])
sigmoid_best_case[1] = pca.transform(sigmoid_best_case[1])
sigmoid_clf = SVC(kernel='sigmoid',random_state=10)
sigmoid_clf.fit(sigmoid_best_case[0],sigmoid_best_case[2])
sigmoid_best_case[4] = sigmoid_clf.predict(sigmoid_best_case[1])

print("Confusion Matrix for test size",size[sigmoid_index],":")
print(confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4]))
print("Performance Report:")
print(classification_report(sigmoid_best_case[3],sigmoid_best_case[4]))
```

```
Confusion Matrix for test size 0.7 :
[[38  1  0]
 [ 0 21 11]
 [ 0  1 33]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      0.97      0.99        39
Iris-versicolor       0.91      0.66      0.76        32
 Iris-virginica       0.75      0.97      0.85        34

       accuracy                           0.88       105
      macro avg       0.89      0.87      0.87       105
   weighted avg       0.89      0.88      0.87       105
```

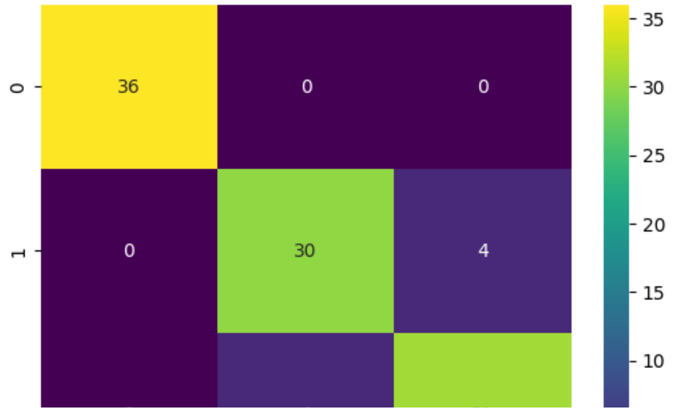
```
cm = confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#MLP

pca = PCA(n_components = 2)
MLP_best_case[0] = pca.fit_transform(MLP_best_case[0])
MLP_best_case[1] = pca.transform(MLP_best_case[1])
MLP_clf = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=900)
MLP_clf.fit(MLP_best_case[0],MLP_best_case[2])
MLP_best_case[4] = MLP_clf.predict(MLP_best_case[1])

print("Confusion Matrix for test size",size[MLP_index],":")
print(confusion_matrix(MLP_best_case[3],MLP_best_case[4]))
print("Performance Report:")
print(classification_report(MLP_best_case[3],MLP_best_case[4]))
```

```
Confusion Matrix for test size 0.7 :
[[36  0  0]
 [ 0 30  4]
 [ 0  4 31]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        36
Iris-versicolor       0.88      0.88      0.88        34
 Iris-virginica       0.89      0.89      0.89        35

       accuracy                           0.92       105
      macro avg       0.92      0.92      0.92       105
   weighted avg       0.92      0.92      0.92       105
```

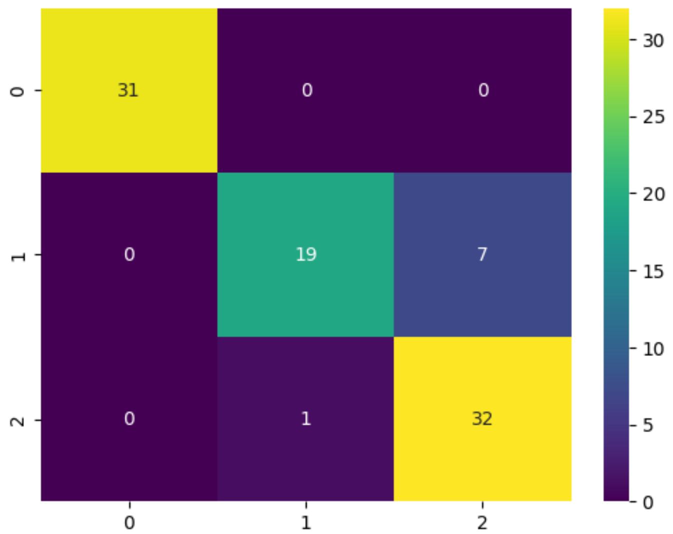
```
cm = confusion_matrix(MLP_best_case[3],MLP_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#RandomForest

pca = PCA(n_components = 2)
rf_best_case[0] = pca.fit_transform(rf_best_case[0])
rf_best_case[1] = pca.transform(rf_best_case[1])
RFclf = RandomForestClassifier(n_estimators=20)
RFclf.fit(rf_best_case[0],rf_best_case[2])
rf_best_case[4] = RFclf.predict(rf_best_case[1])

print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

```
Confusion Matrix for test size 0.6 :
[[31  0  0]
 [ 0 19  7]
 [ 0  1 32]]
Performance Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        31
Iris-versicolor       0.95      0.73      0.83        26
 Iris-virginica       0.82      0.97      0.89        33

       accuracy                           0.91        90
      macro avg       0.92      0.90      0.90        90
   weighted avg       0.92      0.91      0.91        90
```

```
cm = confusion_matrix(rf_best_case[3],rf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix,roc_curve,auc
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,label_binarize
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import zero_one_loss
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA
```

```
!gdown 1cJSgF3wH1ez3oovZODp-rf2-ek6TAv13

df = pd.read_csv('ionosphere.data')

X = df.drop(df.columns[34],axis=1)
Y = df[df.columns[34]]
Y = np.where(Y == 'b', 0, 1)

size=[0.30,0.40,0.50,0.60,0.70]
```

```
Downloading...
From: https://drive.google.com/uc?id=1cJSgF3wH1ez3oovZODp-rf2-ek6TAv13
To: /content/ionosphere.data
100% 76.5k/76.5k [00:00<00:00, 52.4MB/s]
```

```
#SVM_linear

linear_best_case = [0,0,0,0,0]
max = 0
linear_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  linear_clf = SVC(kernel='linear',random_state=10)
  linear_clf.fit(X_train,Y_train)
  Y_pred = linear_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    linear_best_case[0] = X_train
    linear_best_case[1] = X_test
    linear_best_case[2] = Y_train
    linear_best_case[3] = Y_test
    linear_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    linear_index = i
  print("--------------------------------------------------")
  print("--------------------------------------------------")

# print(linear_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[29 10]
 [ 6 60]]
Accuracy:
84.76190476190476
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[39 22]
 [ 0 79]]
Accuracy:
84.28571428571429
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 38 19]
 [  5 113]]
Accuracy:
86.28571428571429
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
```

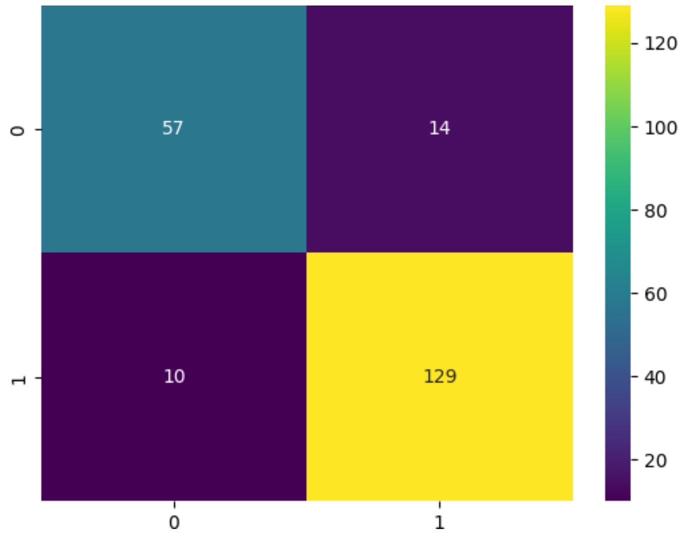
```
[[ 57  14]
 [ 10 129]]
Accuracy:
88.57142857142857
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 54 38]
 [  2 151]]
Accuracy:
83.6734693877551
-------------------------------------------------
-------------------------------------------------
```

```
print("Confusion Matrix for test size",size[linear_index],":")
print(confusion_matrix(linear_best_case[3],linear_best_case[4]))
print("Performance Report:")
print(classification_report(linear_best_case[3],linear_best_case[4]))
```

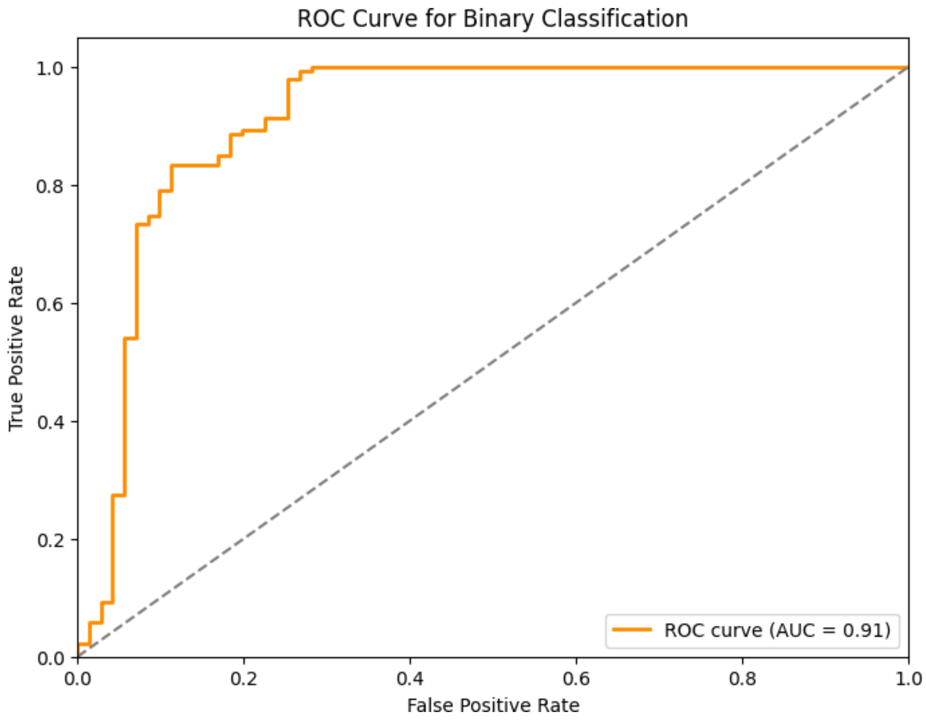
```
Confusion Matrix for test size 0.6 :
[[ 57  14]
 [ 10 129]]
Performance Report:
              precision    recall  f1-score   support

           0       0.85      0.80      0.83        71
           1       0.90      0.93      0.91       139

    accuracy                           0.89       210
   macro avg       0.88      0.87      0.87       210
weighted avg       0.88      0.89      0.88       210
```

```
cm = confusion_matrix(linear_best_case[3],linear_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = linear_best_case[0],linear_best_case[1],linear_best_case[2],linear_best_case[3]
scores = None

# scores = linear_clf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = linear_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_Polynomial

poly_best_case = [0,0,0,0,0]
max = 0
poly_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  poly_clf = SVC(kernel='poly',random_state=10)
  poly_clf.fit(X_train,Y_train)
  Y_pred = poly_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    poly_best_case[0] = X_train
    poly_best_case[1] = X_test
    poly_best_case[2] = Y_train
    poly_best_case[3] = Y_test
    poly_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    poly_index = i
  print("--------------------------------------------------")
  print("--------------------------------------------------")

# print(poly_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[10 27]
 [ 0 68]]
Accuracy:
74.28571428571429
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[10 37]
 [ 0 93]]
Accuracy:
73.57142857142858
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 16 48]
 [  0 111]]
Accuracy:
72.57142857142857
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[ 18 56]
```

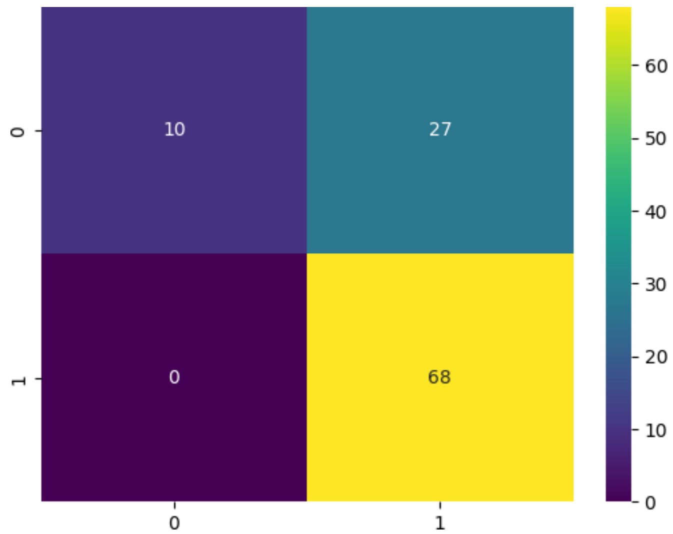
```
 [  0 136]]
Accuracy:
73.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 11 74]
 [  0 160]]
Accuracy:
69.79591836734694
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[poly_index],":")
 print(confusion_matrix(poly_best_case[3],poly_best_case[4]))
print("Performance Report:")
print(classification_report(poly_best_case[3],poly_best_case[4]))
```

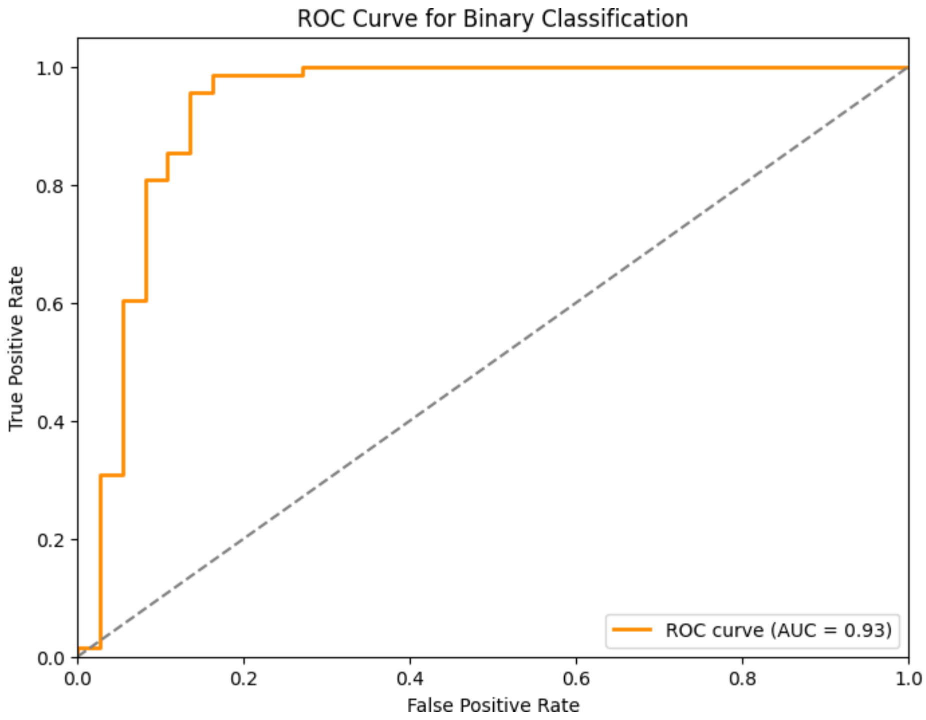
```
Confusion Matrix for test size 0.3 :
[[10 27]
 [ 0 68]]
Performance Report:
              precision    recall  f1-score   support

           0       1.00      0.27      0.43        37
           1       0.72      1.00      0.83        68

    accuracy                           0.74       105
   macro avg       0.86      0.64      0.63       105
weighted avg       0.82      0.74      0.69       105
```

```
cm = confusion_matrix(poly_best_case[3],poly_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = poly_best_case[0],poly_best_case[1],poly_best_case[2],poly_best_case[3]
scores = None

# scores = RFclf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = poly_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
 plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_Gussian

rbf_best_case = [0,0,0,0,0]
max = 0
rbf_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  rbf_clf = SVC(kernel='rbf',random_state=10)
  rbf_clf.fit(X_train,Y_train)
  Y_pred = rbf_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    rbf_best_case[0] = X_train
    rbf_best_case[1] = X_test
    rbf_best_case[2] = Y_train
    rbf_best_case[3] = Y_test
    rbf_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    rbf_index = i
  print(" ------------------------------------------------------------------------------------------------ ")
  print(" ------------------------------------------------------------------------------------------------ ")

# print(rbf_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[31 5]
 [ 2 67]]
Accuracy:
93.33333333333333
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[44 7]
 [ 2 87]]
Accuracy:
93.57142857142857
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 54   9]
 [   2 110]]
Accuracy:
93.71428571428572
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[ 61 14]
 [   3 132]]
```

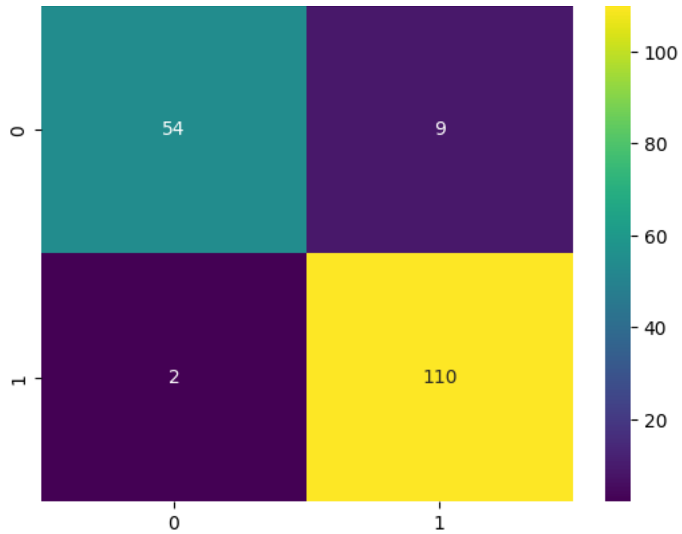
```
Accuracy:
91.9047619047619
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 76 15]
 [  4 150]]
Accuracy:
92.24489795918367
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[rbf_index],":")
print(confusion_matrix(rbf_best_case[3],rbf_best_case[4]))
print("Performance Report:")
print(classification_report(rbf_best_case[3],rbf_best_case[4]))
```

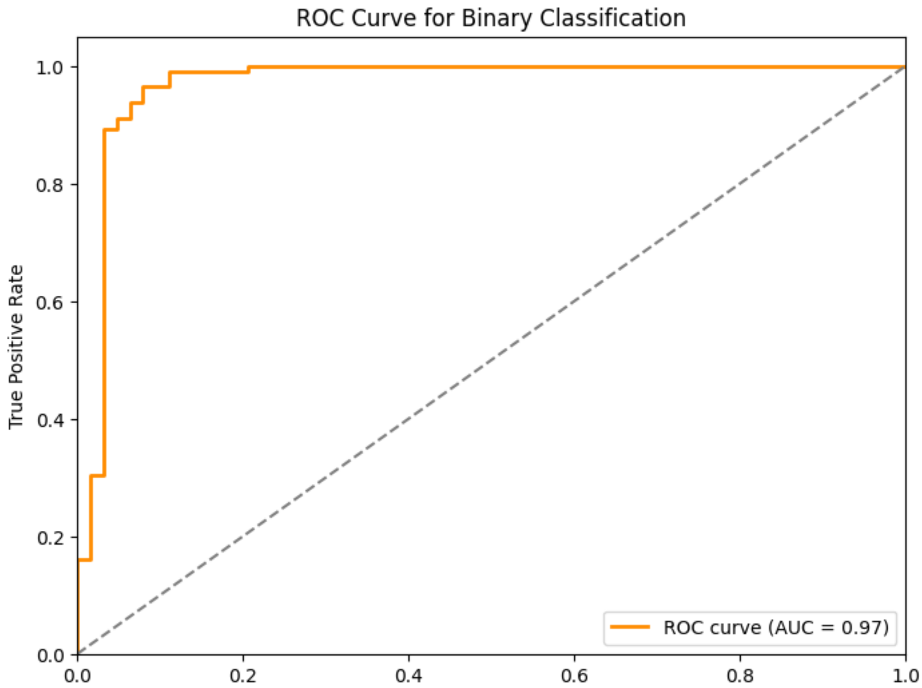
```
Confusion Matrix for test size 0.5 :
[[ 54   9]
 [  2 110]]
Performance Report:
              precision    recall  f1-score   support

           0       0.96      0.86      0.91        63
           1       0.92      0.98      0.95       112

    accuracy                           0.94       175
   macro avg       0.94      0.92      0.93       175
weighted avg       0.94      0.94      0.94       175
```

```
cm = confusion_matrix(rbf_best_case[3],rbf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = rbf_best_case[0],rbf_best_case[1],rbf_best_case[2],rbf_best_case[3]
scores = None

# scores = RFclf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = rbf_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_sigmoid

sigmoid_best_case = [0,0,0,0,0]
max = 0
sigmoid_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  sigmoid_clf = SVC(kernel='sigmoid',random_state=10)
  sigmoid_clf.fit(X_train,Y_train)
  Y_pred = sigmoid_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    sigmoid_best_case[0] = X_train
    sigmoid_best_case[1] = X_test
    sigmoid_best_case[2] = Y_train
    sigmoid_best_case[3] = Y_test
    sigmoid_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    sigmoid_index = i
  print(" --------------------------------------------------------------------------------- ")
  print(" --------------------------------------------------------------------------------- ")

# print(sigmoid_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[18 18]
 [ 1 68]]
Accuracy:
81.9047619047619
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[28 20]
 [ 1 91]]
Accuracy:
85.0
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 38 24]
 [  0 113]]
Accuracy:
86.28571428571429
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[ 45 24]
 [  1 140]]
Accuracy:
```

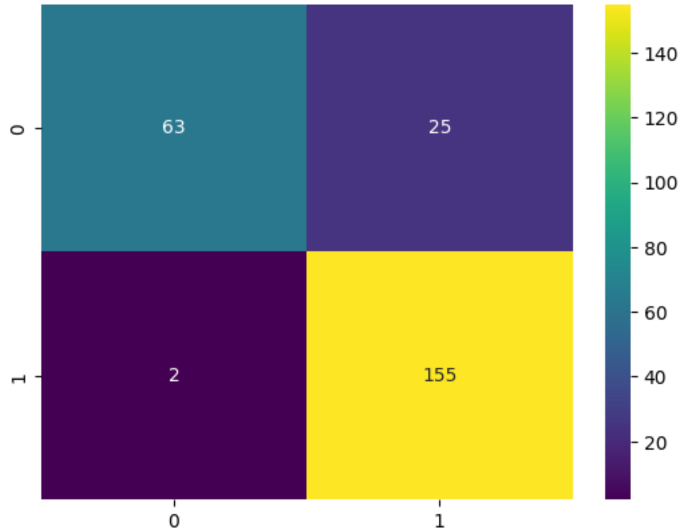
```
88.09523809523809
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 63 25]
 [  2 155]]
Accuracy:
88.9795918367347
--------------------------------------------------
--------------------------------------------------
```

```python
print("Confusion Matrix for test size",size[sigmoid_index],":")
print(confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4]))
print("Performance Report:")
print(classification_report(sigmoid_best_case[3],sigmoid_best_case[4]))
```

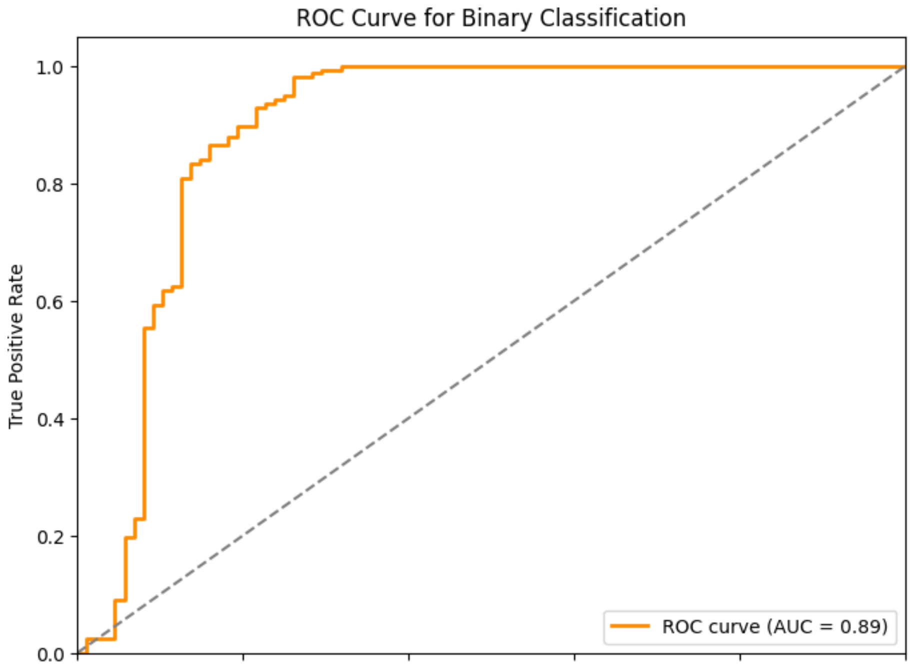
```
Confusion Matrix for test size 0.7 :
[[ 63  25]
 [  2 155]]
Performance Report:
              precision    recall  f1-score   support

           0       0.97      0.72      0.82        88
           1       0.86      0.99      0.92       157

    accuracy                           0.89       245
   macro avg       0.92      0.85      0.87       245
weighted avg       0.90      0.89      0.89       245
```

```python
cm = confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```python
X_train,X_test, y_train, y_test = sigmoid_best_case[0],sigmoid_best_case[1],sigmoid_best_case[2],sigmoid_best_case[3]
scores = None

# scores = RFclf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = sigmoid_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
 plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
# MLP

MLP_best_case = [0,0,0,0,0]
max = 0
MLP_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  MLP_clf = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=900)
  MLP_clf.fit(X_train,Y_train)
  Y_pred = MLP_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    MLP_best_case[0] = X_train
    MLP_best_case[1] = X_test
    MLP_best_case[2] = Y_train
    MLP_best_case[3] = Y_test
    MLP_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    MLP_index = i
  print(" -------------------------------------------------- ")
  print(" -------------------------------------------------- ")

# print(MLP_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[28 11]
 [ 2 64]]
Accuracy:
87.61904761904762
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[44 9]
 [ 1 86]]
Accuracy:
92.85714285714286
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 56 17]
 [  1 101]]
Accuracy:
89.71428571428571
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[ 56 18]
 [  3 133]]
Accuracy:
90.0
```

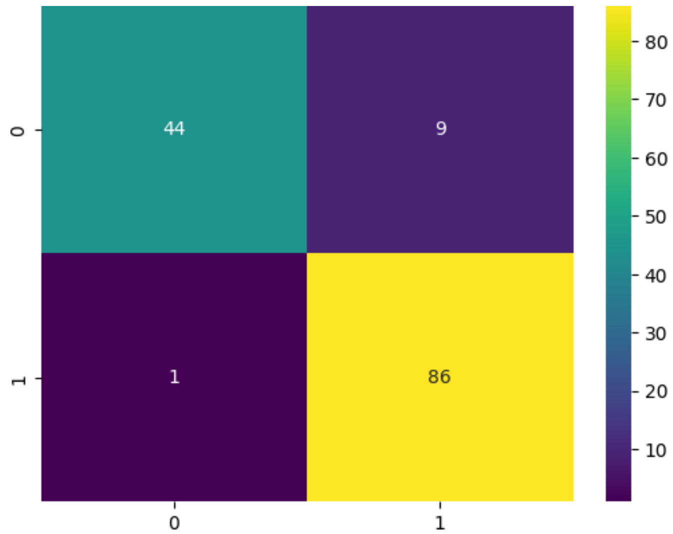
```
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 61 24]
 [ 12 148]]
Accuracy:
85.3061224489796
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[MLP_index],":")
print(confusion_matrix(MLP_best_case[3],MLP_best_case[4]))
print("Performance Report:")
print(classification_report(MLP_best_case[3],MLP_best_case[4]))
```

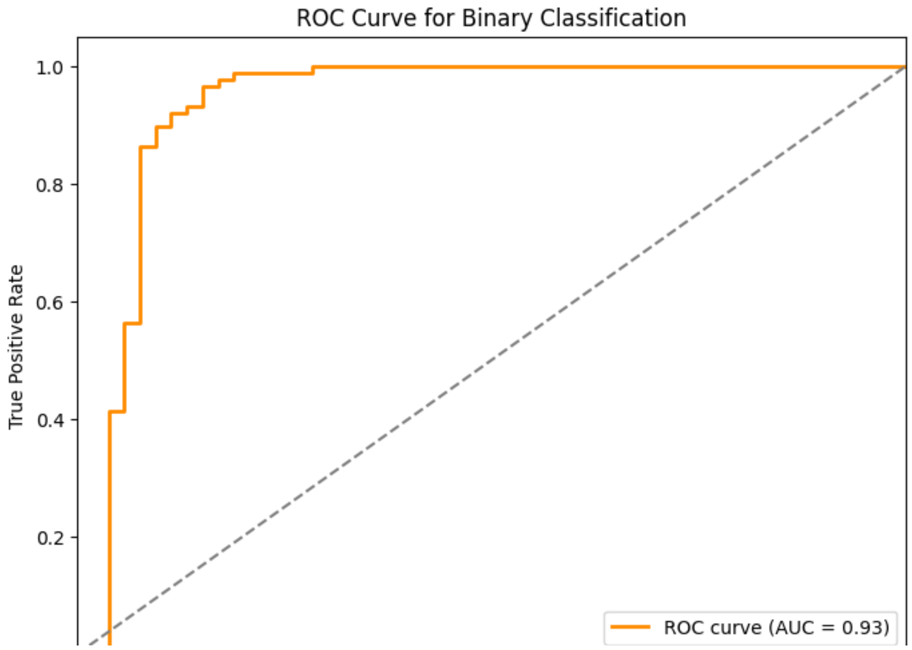
```
Confusion Matrix for test size 0.4 :
[[44  9]
 [ 1 86]]
Performance Report:
              precision    recall  f1-score   support

           0       0.98      0.83      0.90        53
           1       0.91      0.99      0.95        87

    accuracy                           0.93       140
   macro avg       0.94      0.91      0.92       140
weighted avg       0.93      0.93      0.93       140
```

```
cm = confusion_matrix(MLP_best_case[3],MLP_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = MLP_best_case[0],MLP_best_case[1],MLP_best_case[2],MLP_best_case[3]
scores = None

scores = MLP_clf.predict_proba(X_test)
prob_positive_class = scores[:, 1]
# decision_function = RFclf.decision_function(X_test)
# prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
 plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
# Random Forest

rf_best_case = [0,0,0,0,0]
max = 0
rf_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  RFclf = RandomForestClassifier(n_estimators=20)
  RFclf.fit(X_train,Y_train)
  Y_pred = RFclf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    rf_best_case[0] = X_train
    rf_best_case[1] = X_test
    rf_best_case[2] = Y_train
    rf_best_case[3] = Y_test
    rf_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    rf_index = i
  print(" ---------------------------------------------------------------------------------- ")
  print(" ---------------------------------------------------------------------------------- ")

# print(rf_best_case)
```

```
Confusion Matrix for test size 0.3 :
[[38 2]
 [ 3 62]]
Accuracy:
95.23809523809523
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[45 9]
 [ 6 80]]
Accuracy:
89.28571428571429
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 53   5]
 [  9 108]]
Accuracy:
92.0
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.6 :
[[ 60 10]
 [ 12 128]]
Accuracy:
89.52380952380953
--------------------------------------------------
```

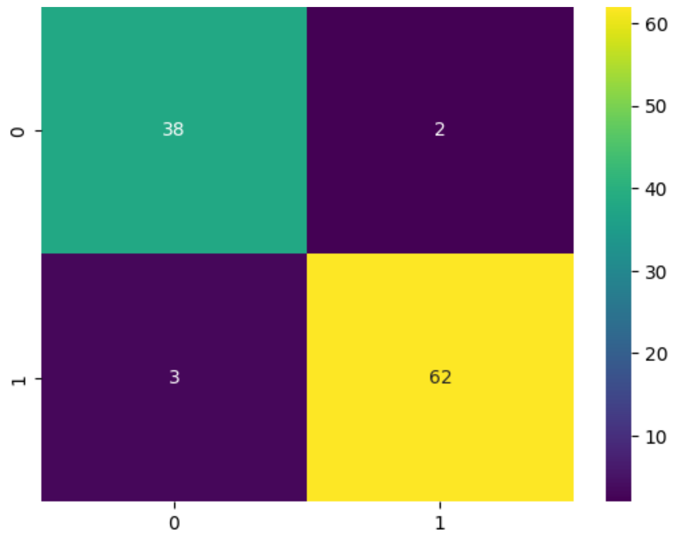
```
--------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 70 11]
 [ 13 151]]
Accuracy:
90.20408163265307
--------------------------------------------------
--------------------------------------------------
```

```
print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

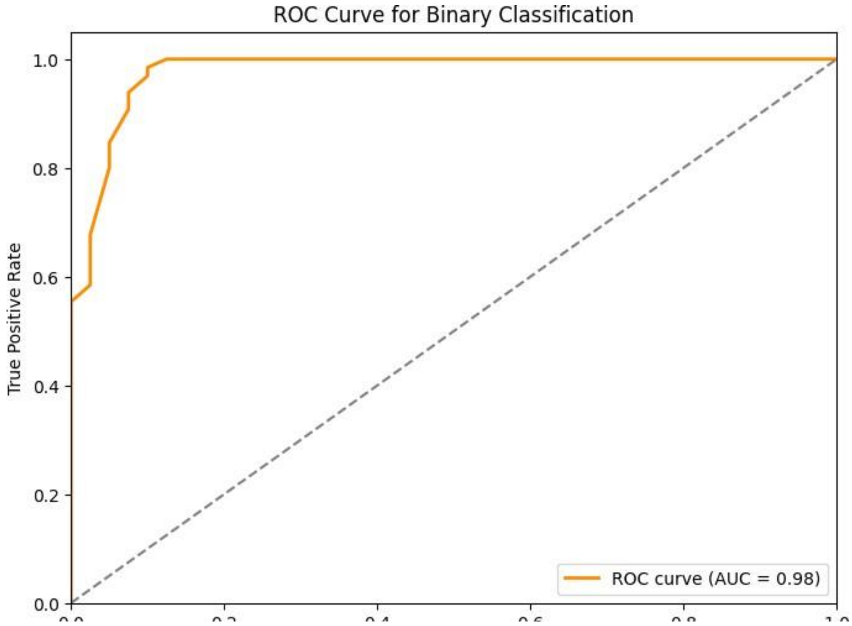
```
Confusion Matrix for test size 0.3 :
[[38 2]
 [ 3 62]]
Performance Report:
              precision    recall  f1-score   support

           0       0.93      0.95      0.94        40
           1       0.97      0.95      0.96        65

    accuracy                           0.95       105
   macro avg       0.95      0.95      0.95       105
weighted avg       0.95      0.95      0.95       105
```

```
cm = confusion_matrix(rf_best_case[3],rf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = rf_best_case[0],rf_best_case[1],rf_best_case[2],rf_best_case[3]
scores = None

scores = RFclf.predict_proba(X_test)
prob_positive_class = scores[:, 1]
# decision_function = RFclf.decision_function(X_test)
# prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```



After Principal Component Analysis (PCA) for feature dimensionality reduction

```
#SVM_linear

pca = PCA(n_components = 2)
linear_best_case[0] = pca.fit_transform(linear_best_case[0])
linear_best_case[1] = pca.transform(linear_best_case[1])
linear_clf = SVC(kernel='linear',random_state=10)
linear_clf.fit(linear_best_case[0],linear_best_case[2])
linear_best_case[4] = linear_clf.predict(linear_best_case[1])
print("Confusion Matrix for test size",size[linear_index],":")
print(confusion_matrix(linear_best_case[3],linear_best_case[4]))
print("Performance Report:")
print(classification_report(linear_best_case[3],linear_best_case[4]))
```

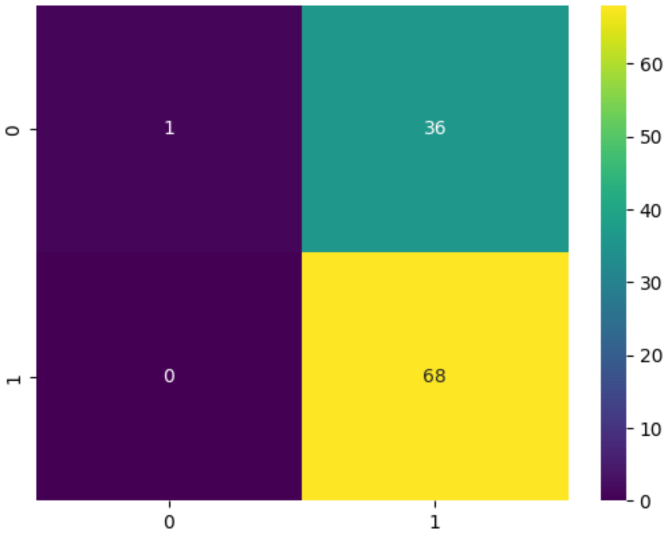
```
Confusion Matrix for test size 0.6 :
[[  0  71]
 [  0 139]]
Performance Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        71
           1       0.66      1.00      0.80       139

    accuracy                           0.66       210
   macro avg       0.33      0.50      0.40       210
weighted avg       0.44      0.66      0.53       210
```

```
cm = confusion_matrix(linear_best_case[3],linear_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```


```
#SVM_Polynomial

pca = PCA(n_components = 2)
poly_best_case[0] = pca.fit_transform(poly_best_case[0])
poly_best_case[1] = pca.transform(poly_best_case[1])
poly_clf = SVC(kernel='poly',random_state=10)
poly_clf.fit(poly_best_case[0],poly_best_case[2])
poly_best_case[4] = poly_clf.predict(poly_best_case[1])

print("Confusion Matrix for test size",size[poly_index],":")
 print(confusion_matrix(poly_best_case[3],poly_best_case[4]))
print("Performance Report:")
print(classification_report(poly_best_case[3],poly_best_case[4]))
```

```
Confusion Matrix for test size 0.3 :
[[ 1 36]
 [ 0 68]]
Performance Report:
              precision    recall  f1-score   support

           0       1.00      0.03      0.05        37
           1       0.65      1.00      0.79        68

    accuracy                           0.66       105
   macro avg       0.83      0.51      0.42       105
weighted avg       0.78      0.66      0.53       105
```

```
cm = confusion_matrix(poly_best_case[3],poly_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```

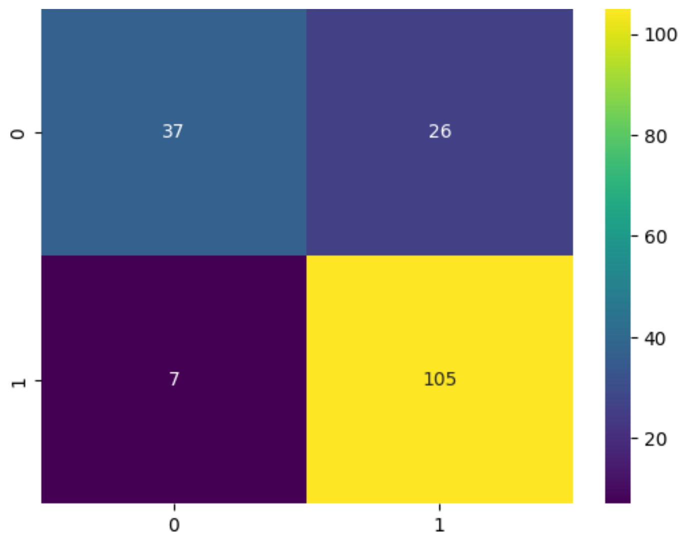



```
#SVM_Gaussian

pca = PCA(n_components = 2)
rbf_best_case[0] = pca.fit_transform(rbf_best_case[0])
rbf_best_case[1] = pca.transform(rbf_best_case[1])
rbf_clf = SVC(kernel='rbf',random_state=10)
rbf_clf.fit(rbf_best_case[0],rbf_best_case[2])
rbf_best_case[4] = rbf_clf.predict(rbf_best_case[1])

print("Confusion Matrix for test size",size[rbf_index],":")
print(confusion_matrix(rbf_best_case[3],rbf_best_case[4]))
print("Performance Report:")
print(classification_report(rbf_best_case[3],rbf_best_case[4]))
```

```
Confusion Matrix for test size 0.5 :
[[ 37  26]
 [  7 105]]
Performance Report:
              precision    recall  f1-score   support

           0       0.84      0.59      0.69        63
```

```
           1       0.80      0.94      0.86       112

    accuracy                           0.81       175
   macro avg       0.82      0.76      0.78       175
weighted avg       0.82      0.81      0.80       175
```

```
cm = confusion_matrix(rbf_best_case[3],rbf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#SVM_Sigmoid

pca = PCA(n_components = 2)
sigmoid_best_case[0] = pca.fit_transform(sigmoid_best_case[0])
sigmoid_best_case[1] = pca.transform(sigmoid_best_case[1])
sigmoid_clf = SVC(kernel='sigmoid',random_state=10)
sigmoid_clf.fit(sigmoid_best_case[0],sigmoid_best_case[2])
sigmoid_best_case[4] = sigmoid_clf.predict(sigmoid_best_case[1])

print("Confusion Matrix for test size",size[sigmoid_index],":")
print(confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4]))
print("Performance Report:")
print(classification_report(sigmoid_best_case[3],sigmoid_best_case[4]))
```

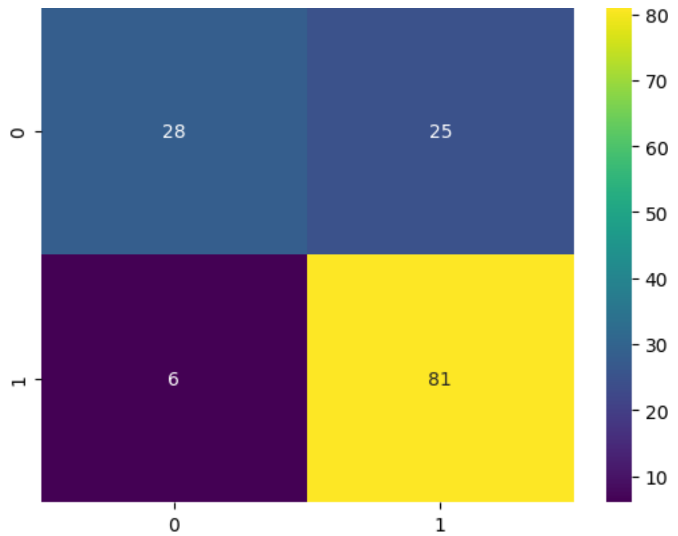
```
Confusion Matrix for test size 0.7 :
[[ 25  63]
 [ 42 115]]
Performance Report:
              precision    recall  f1-score   support

           0       0.37      0.28      0.32        88
           1       0.65      0.73      0.69       157

    accuracy                           0.57       245
   macro avg       0.51      0.51      0.50       245
weighted avg       0.55      0.57      0.56       245
```

```
cm = confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```


```
#MLP

pca = PCA(n_components = 2)
MLP_best_case[0] = pca.fit_transform(MLP_best_case[0])
MLP_best_case[1] = pca.transform(MLP_best_case[1])
MLP_clf = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=900)
MLP_clf.fit(MLP_best_case[0],MLP_best_case[2])
 MLP_best_case[4] = MLP_clf.predict(MLP_best_case[1])

print("Confusion Matrix for test size",size[MLP_index],":")
print(confusion_matrix(MLP_best_case[3],MLP_best_case[4]))
print("Performance Report:")
print(classification_report(MLP_best_case[3],MLP_best_case[4]))
```

```
Confusion Matrix for test size 0.4 :
[[28 25]
 [ 6 81]]
Performance Report:
              precision    recall  f1-score   support

           0       0.82      0.53      0.64        53
           1       0.76      0.93      0.84        87

    accuracy                           0.78       140
   macro avg       0.79      0.73      0.74       140
weighted avg       0.79      0.78      0.77       140
```

```
cm = confusion_matrix(MLP_best_case[3],MLP_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```

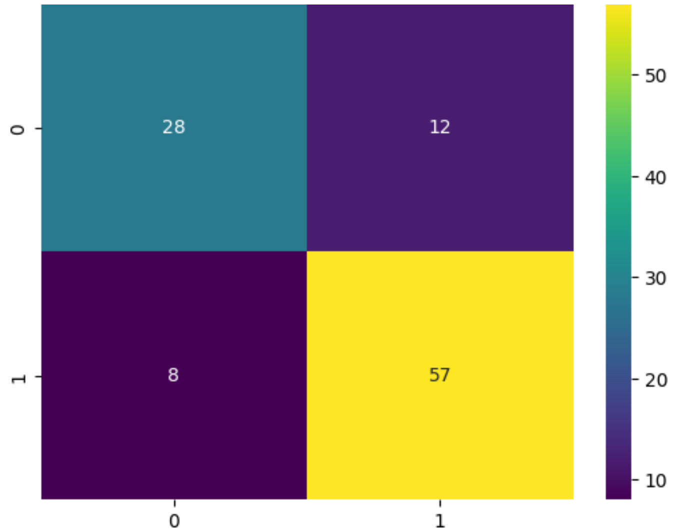



```
#RandomForest

pca = PCA(n_components = 2)
rf_best_case[0] = pca.fit_transform(rf_best_case[0])
rf_best_case[1] = pca.transform(rf_best_case[1])
RFclf = RandomForestClassifier(n_estimators=20)
RFclf.fit(rf_best_case[0],rf_best_case[2])
 rf_best_case[4] = RFclf.predict(rf_best_case[1])

print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

```
Confusion Matrix for test size 0.3 :
[[28 12]
 [ 8 57]]
Performance Report:
              precision    recall  f1-score   support

           0       0.78      0.70      0.74        40
```

```
           1       0.83      0.88      0.85        65

    accuracy                           0.81       105
   macro avg       0.80      0.79      0.79       105
weighted avg       0.81      0.81      0.81       105
```

```
1 cm = confusion_matrix(rf_best_case[3],rf_best_case[4])
2 sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
3 plt.show()
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix,roc_curve,auc
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,label_binarize
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import zero_one_loss
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA
```

```
!gdown 1F5U0IZ8otJ5iU6s6e3x_Cw3HGSLypaZl

column_names = ["ID","Diagnosis",
    "mean radius", "mean texture", "mean perimeter", "mean area", "mean smoothness",
    "mean compactness", "mean concavity", "mean concave points", "mean symmetry", "mean fractal dimension",
    "radius error", "texture error", "perimeter error", "area error", "smoothness error",
    "compactness error", "concavity error", "concave points error", "symmetry error", "fractal dimension error",
    "worst radius", "worst texture", "worst perimeter", "worst area", "worst smoothness",
    "worst compactness", "worst concavity", "worst concave points", "worst symmetry", "worst fractal dimension"
]
df = pd.read_csv('wdbc.data',names = column_names)

X = df.drop(df.columns[[0,1]],axis=1)
Y = df[df.columns[1]]
Y = np.where(Y == 'M', 0, 1)

size=[0.30,0.40,0.50,0.60,0.70]
```

```
Downloading...
From: https://drive.google.com/uc?id=1F5U0IZ8otJ5iU6s6e3x_Cw3HGSLypaZl
To: /content/wdbc.data
100% 124k/124k [00:00<00:00, 99.6MB/s]
```

```
#SVM_linear

linear_best_case = [0,0,0,0,0]
max = 0
linear_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  linear_clf = SVC(kernel='linear',random_state=10)
  linear_clf.fit(X_train,Y_train)
  Y_pred = linear_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    linear_best_case[0] = X_train
    linear_best_case[1] = X_test
    linear_best_case[2] = Y_train
    linear_best_case[3] = Y_test
    linear_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    linear_index = i
  print("---------------------------------------------------")
  print("---------------------------------------------------")

# print(linear_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[ 61   1]
 [  2 107]]
Accuracy:
98.24561403508771
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[ 90   2]
 [  1 135]]
Accuracy:
98.68421052631578
--------------------------------------------------
```

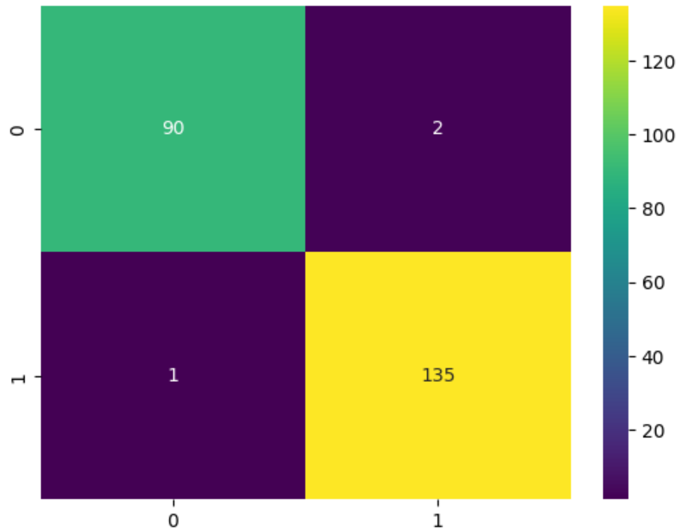
```
-------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 96   6]
 [  4 179]]
Accuracy:
96.49122807017544
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.6 :
[[118  10]
 [  5 209]]
Accuracy:
95.6140350877193
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.7 :
[[139   8]
 [  5 247]]
Accuracy:
96.74185463659147
-------------------------------------------------
-------------------------------------------------
```

```
print("Confusion Matrix for test size",size[linear_index],":")
print(confusion_matrix(linear_best_case[3],linear_best_case[4]))
print("Performance Report:")
print(classification_report(linear_best_case[3],linear_best_case[4]))
```

```
Confusion Matrix for test size 0.4 :
[[ 90   2]
 [  1 135]]
Performance Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98        92
           1       0.99      0.99      0.99       136

    accuracy                           0.99       228
   macro avg       0.99      0.99      0.99       228
weighted avg       0.99      0.99      0.99       228
```

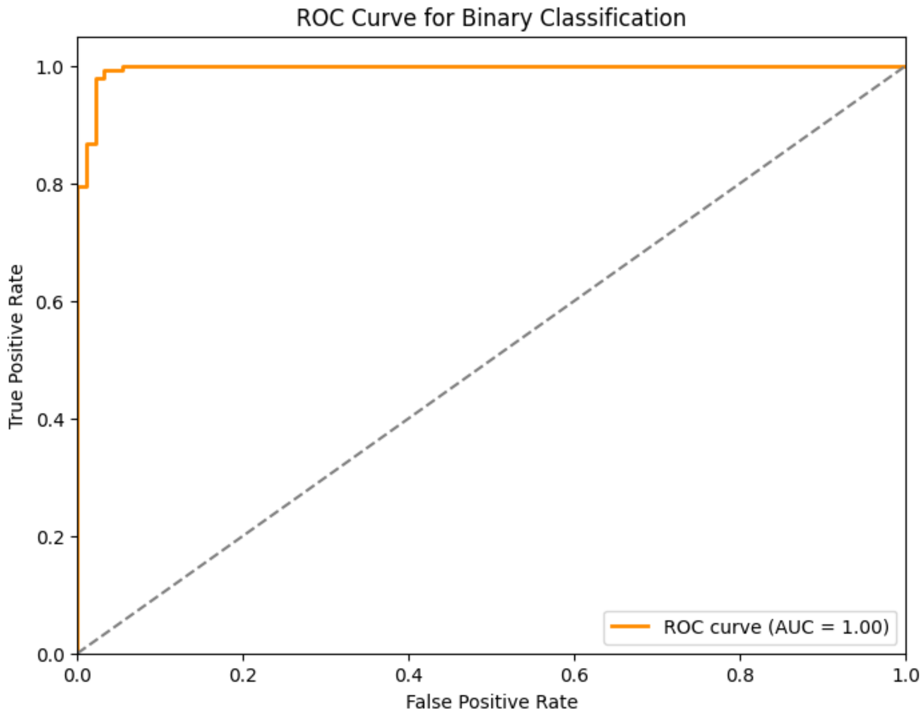
```
cm = confusion_matrix(linear_best_case[3],linear_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = linear_best_case[0],linear_best_case[1],linear_best_case[2],linear_best_case[3]
scores = None

# scores = linear_clf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = linear_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_Polynomial

poly_best_case = [0,0,0,0,0]
max = 0
poly_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  poly_clf = SVC(kernel='poly',random_state=10)
  poly_clf.fit(X_train,Y_train)
  Y_pred = poly_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    poly_best_case[0] = X_train
    poly_best_case[1] = X_test
    poly_best_case[2] = Y_train
    poly_best_case[3] = Y_test
    poly_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    poly_index = i
  print("---------------------------------------------------")
  print("---------------------------------------------------")

# print(poly_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[ 46  20]
 [  0 105]]
Accuracy:
88.30409356725146
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[ 61  30]
 [  0 137]]
Accuracy:
86.8421052631579
--------------------------------------------------
```

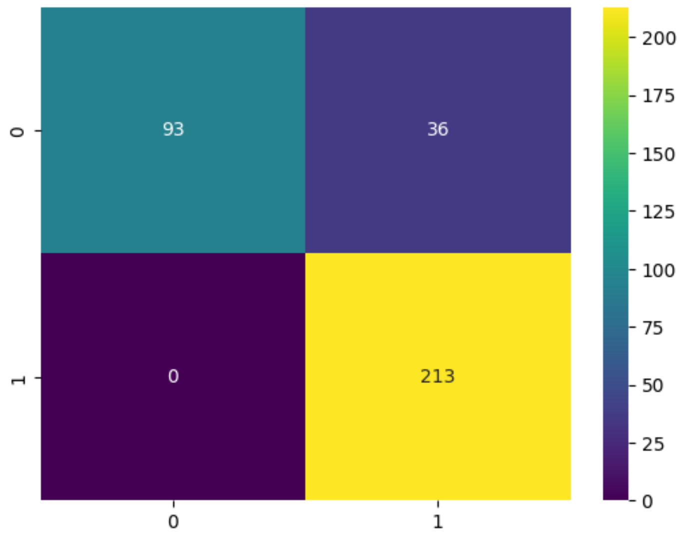
```
-------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 62  33]
 [  1 189]]
Accuracy:
88.0701754385965
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.6 :
[[ 93  36]
 [  0 213]]
Accuracy:
89.47368421052632
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.7 :
[[ 92  57]
 [  1 249]]
Accuracy:
85.46365914786968
-------------------------------------------------
-------------------------------------------------
```

```
print("Confusion Matrix for test size",size[poly_index],":")
print(confusion_matrix(poly_best_case[3],poly_best_case[4]))
print("Performance Report:")
print(classification_report(poly_best_case[3],poly_best_case[4]))
```

```
Confusion Matrix for test size 0.6 :
[[ 93  36]
 [  0 213]]
Performance Report:
              precision    recall  f1-score   support

           0       1.00      0.72      0.84       129
           1       0.86      1.00      0.92       213

    accuracy                           0.89       342
   macro avg       0.93      0.86      0.88       342
weighted avg       0.91      0.89      0.89       342
```

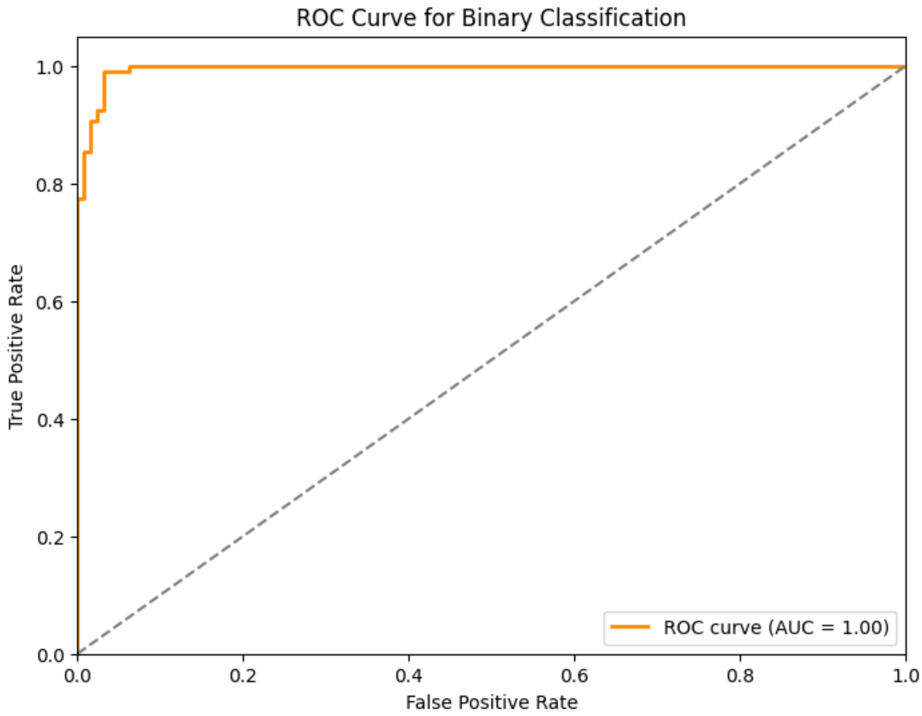
```
cm = confusion_matrix(poly_best_case[3],poly_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = poly_best_case[0],poly_best_case[1],poly_best_case[2],poly_best_case[3]
scores = None

# scores = RFclf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = poly_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_Gussian

rbf_best_case = [0,0,0,0,0]
max = 0
rbf_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  rbf_clf = SVC(kernel='rbf',random_state=10)
  rbf_clf.fit(X_train,Y_train)
  Y_pred = rbf_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    rbf_best_case[0] = X_train
    rbf_best_case[1] = X_test
    rbf_best_case[2] = Y_train
    rbf_best_case[3] = Y_test
    rbf_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    rbf_index = i
  print("---------------------------------------------------")
  print("---------------------------------------------------")

# print(rbf_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[ 59   5]
 [  2 105]]
Accuracy:
95.90643274853801
---------------------------------------------------
---------------------------------------------------
Confusion Matrix for test size 0.4 :
[[ 76   4]
 [  6 142]]
Accuracy:
95.6140350877193
---------------------------------------------------
```

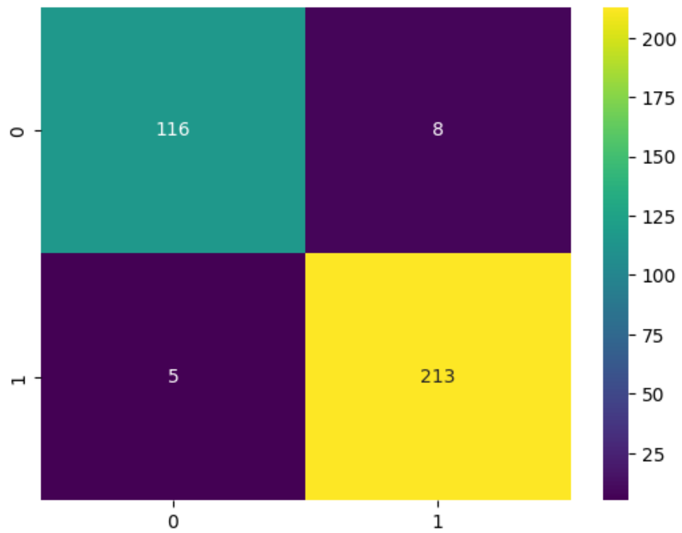
```
-------------------------------------------------
Confusion Matrix for test size 0.5 :
[[ 97   9]
 [  4 175]]
Accuracy:
95.43859649122807
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.6 :
[[116   8]
 [  5 213]]
Accuracy:
96.19883040935673
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.7 :
[[137  14]
 [  7 241]]
Accuracy:
94.73684210526315
-------------------------------------------------
-------------------------------------------------
```

```
print("Confusion Matrix for test size",size[rbf_index],":")
print(confusion_matrix(rbf_best_case[3],rbf_best_case[4]))
print("Performance Report:")
print(classification_report(rbf_best_case[3],rbf_best_case[4]))
```

```
Confusion Matrix for test size 0.6 :
[[116   8]
 [  5 213]]
Performance Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       124
           1       0.96      0.98      0.97       218

    accuracy                           0.96       342
   macro avg       0.96      0.96      0.96       342
weighted avg       0.96      0.96      0.96       342
```

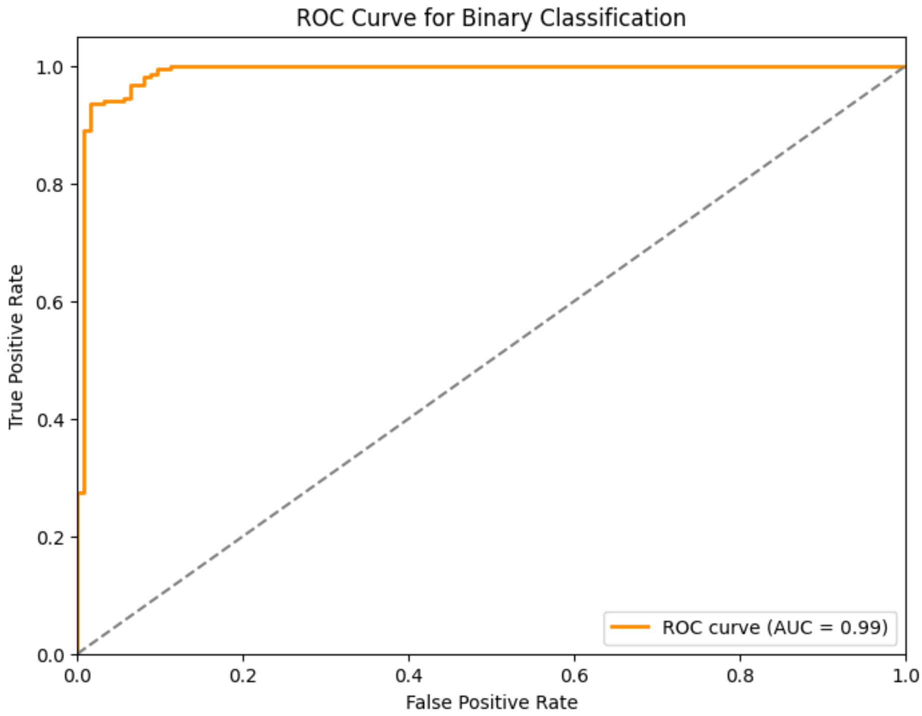
```
cm = confusion_matrix(rbf_best_case[3],rbf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = rbf_best_case[0],rbf_best_case[1],rbf_best_case[2],rbf_best_case[3]
scores = None

# scores = RFclf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = rbf_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
#SVM_sigmoid

sigmoid_best_case = [0,0,0,0,0]
max = 0
sigmoid_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  sigmoid_clf = SVC(kernel='sigmoid',random_state=10)
  sigmoid_clf.fit(X_train,Y_train)
  Y_pred = sigmoid_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    sigmoid_best_case[0] = X_train
    sigmoid_best_case[1] = X_test
    sigmoid_best_case[2] = Y_train
    sigmoid_best_case[3] = Y_test
    sigmoid_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    sigmoid_index = i
  print("---------------------------------------------------")
  print("---------------------------------------------------")

# print(sigmoid_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[ 61   2]
 [  4 104]]
Accuracy:
96.49122807017544
---------------------------------------------------
---------------------------------------------------
Confusion Matrix for test size 0.4 :
[[ 71   8]
 [  2 147]]
Accuracy:
95.6140350877193
---------------------------------------------------
```

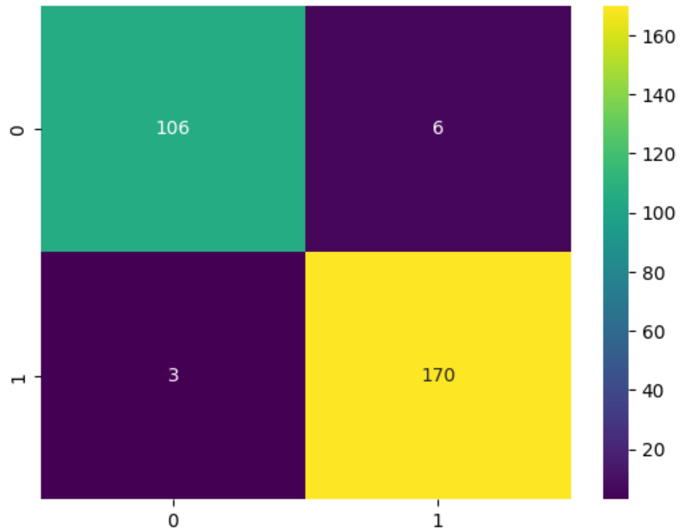
```
------------------------------------------------
Confusion Matrix for test size 0.5 :
[[106   6]
 [  3 170]]
Accuracy:
96.84210526315789
------------------------------------------------
------------------------------------------------
Confusion Matrix for test size 0.6 :
[[118  12]
 [  1 211]]
Accuracy:
96.19883040935673
------------------------------------------------
------------------------------------------------
Confusion Matrix for test size 0.7 :
[[131  14]
 [  3 251]]
Accuracy:
95.73934837092732
------------------------------------------------
------------------------------------------------
```

```
print("Confusion Matrix for test size",size[sigmoid_index],":")
print(confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4]))
print("Performance Report:")
print(classification_report(sigmoid_best_case[3],sigmoid_best_case[4]))
```

```
Confusion Matrix for test size 0.5 :
[[106   6]
 [  3 170]]
Performance Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       112
           1       0.97      0.98      0.97       173

    accuracy                           0.97       285
   macro avg       0.97      0.96      0.97       285
weighted avg       0.97      0.97      0.97       285
```

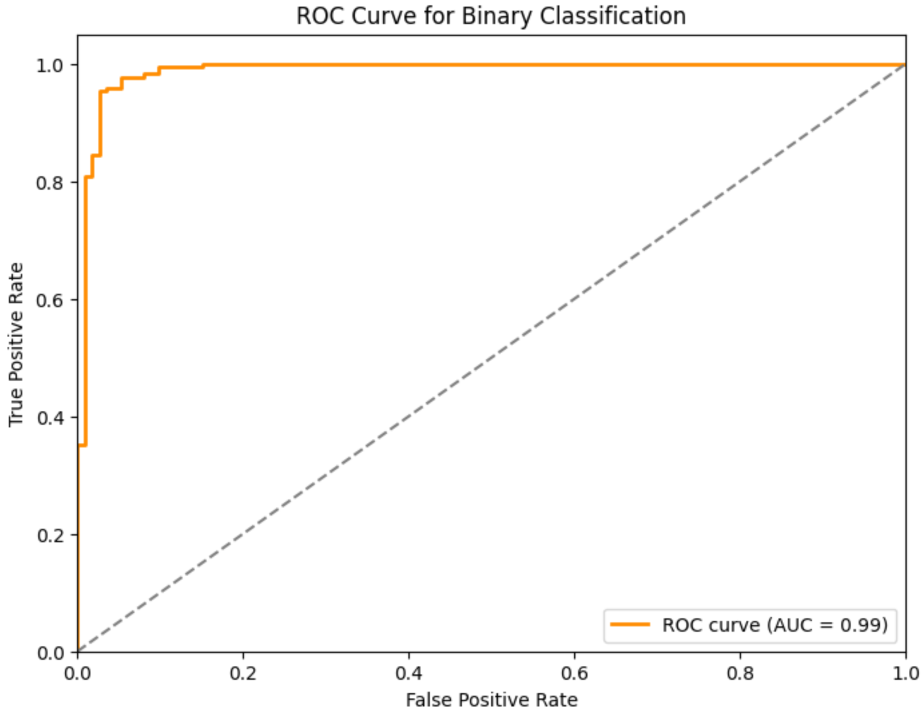
```
cm = confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = sigmoid_best_case[0],sigmoid_best_case[1],sigmoid_best_case[2],sigmoid_best_case[3]
scores = None

# scores = RFclf.predict_proba(X_test)
# prob_positive_class = scores[:, 1]
decision_function = sigmoid_clf.decision_function(X_test)
prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
# MLP

MLP_best_case = [0,0,0,0,0]
max = 0
MLP_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  MLP_clf = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=900)
  MLP_clf.fit(X_train,Y_train)
  Y_pred = MLP_clf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    MLP_best_case[0] = X_train
    MLP_best_case[1] = X_test
    MLP_best_case[2] = Y_train
    MLP_best_case[3] = Y_test
    MLP_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    MLP_index = i
  print("---------------------------------------------------")
  print("---------------------------------------------------")

# print(MLP_best_case[3])
```

```
Confusion Matrix for test size 0.3 :
[[ 65   1]
 [  2 103]]
Accuracy:
98.24561403508771
---------------------------------------------------
---------------------------------------------------
Confusion Matrix for test size 0.4 :
[[ 82   3]
 [  5 138]]
Accuracy:
96.49122807017544
---------------------------------------------------
```

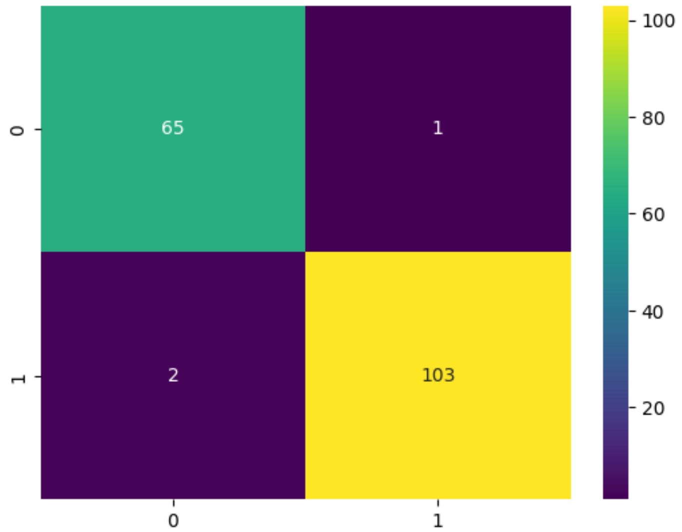
```
-------------------------------------------------
Confusion Matrix for test size 0.5 :
[[101   7]
 [  3 174]]
Accuracy:
96.49122807017544
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.6 :
[[122   5]
 [  5 210]]
Accuracy:
97.07602339181285
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.7 :
[[139  10]
 [  3 247]]
Accuracy:
96.74185463659147
-------------------------------------------------
-------------------------------------------------
```

```
print("Confusion Matrix for test size",size[MLP_index],":")
print(confusion_matrix(MLP_best_case[3],MLP_best_case[4]))
print("Performance Report:")
print(classification_report(MLP_best_case[3],MLP_best_case[4]))
```

```
Confusion Matrix for test size 0.3 :
[[ 65   1]
 [  2 103]]
Performance Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98        66
           1       0.99      0.98      0.99       105

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

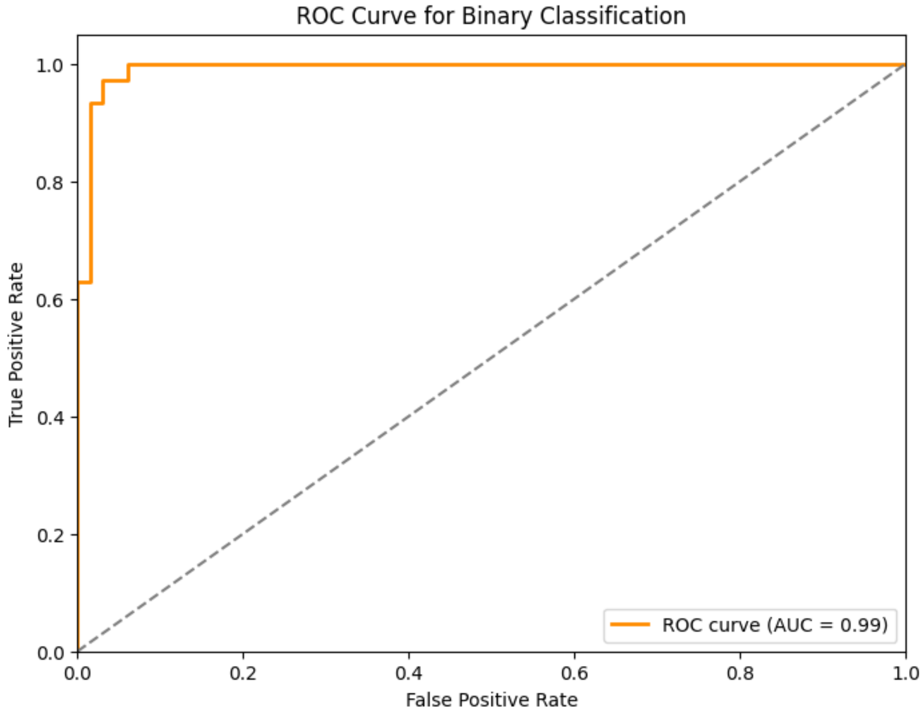
```
cm = confusion_matrix(MLP_best_case[3],MLP_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = MLP_best_case[0],MLP_best_case[1],MLP_best_case[2],MLP_best_case[3]
scores = None

scores = MLP_clf.predict_proba(X_test)
prob_positive_class = scores[:, 1]
# decision_function = RFclf.decision_function(X_test)
# prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```




```
# Random Forest

rf_best_case = [0,0,0,0,0]
max = 0
rf_index = 0
for i in range(0,len(size)):
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=size[i])
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  RFclf = RandomForestClassifier(n_estimators=20)
  RFclf.fit(X_train,Y_train)
  Y_pred = RFclf.predict(X_test)

  print("Confusion Matrix for test size",size[i],":")
  print(confusion_matrix(Y_test,Y_pred))
  print("Accuracy:")
  print(accuracy_score(Y_test,Y_pred)*100)
  if max<accuracy_score(Y_test,Y_pred):
    rf_best_case[0] = X_train
    rf_best_case[1] = X_test
    rf_best_case[2] = Y_train
    rf_best_case[3] = Y_test
    rf_best_case[4] = Y_pred
    max=accuracy_score(Y_test,Y_pred)
    rf_index = i
  print("---------------------------------------------------")
  print("---------------------------------------------------")

# print(rf_best_case)
```

```
Confusion Matrix for test size 0.3 :
[[67  7]
 [ 1 96]]
Accuracy:
95.32163742690058
--------------------------------------------------
--------------------------------------------------
Confusion Matrix for test size 0.4 :
[[ 83   6]
 [  7 132]]
Accuracy:
94.2982456140351
--------------------------------------------------
```

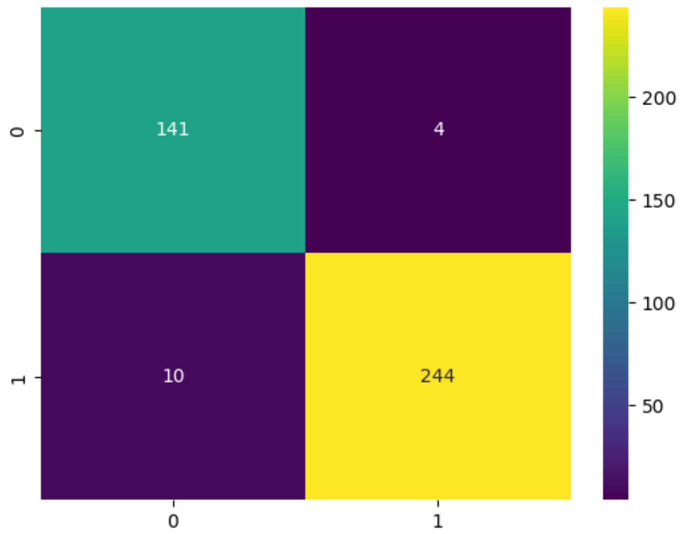
```
-------------------------------------------------
Confusion Matrix for test size 0.5 :
[[105   6]
 [  6 168]]
Accuracy:
95.78947368421052
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.6 :
[[126   7]
 [  7 202]]
Accuracy:
95.90643274853801
-------------------------------------------------
-------------------------------------------------
Confusion Matrix for test size 0.7 :
[[141   4]
 [ 10 244]]
Accuracy:
96.49122807017544
-------------------------------------------------
-------------------------------------------------
```

```
print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

```
Confusion Matrix for test size 0.7 :
[[141   4]
 [ 10 244]]
Performance Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       145
           1       0.98      0.96      0.97       254

    accuracy                           0.96       399
   macro avg       0.96      0.97      0.96       399
weighted avg       0.97      0.96      0.97       399
```

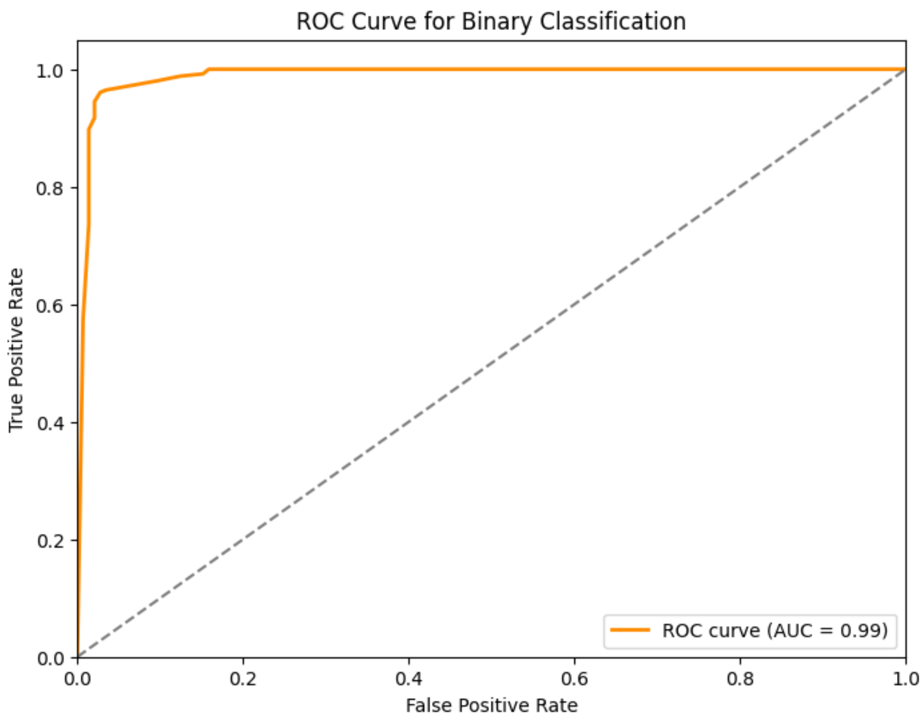
```
cm = confusion_matrix(rf_best_case[3],rf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
X_train,X_test, y_train, y_test = rf_best_case[0],rf_best_case[1],rf_best_case[2],rf_best_case[3]
scores = None

scores = RFclf.predict_proba(X_test)
prob_positive_class = scores[:, 1]
# decision_function = RFclf.decision_function(X_test)
# prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Binary Classification')
plt.legend(loc='lower right')
plt.show()
```



After Principal Component Analysis (PCA) for feature dimensionality reduction

```
#SVM_linear

pca = PCA(n_components = 2)
linear_best_case[0] = pca.fit_transform(linear_best_case[0])
linear_best_case[1] = pca.transform(linear_best_case[1])
linear_clf = SVC(kernel='linear',random_state=10)
linear_clf.fit(linear_best_case[0],linear_best_case[2])
linear_best_case[4] = linear_clf.predict(linear_best_case[1])

print("Confusion Matrix for test size",size[linear_index],":")
print(confusion_matrix(linear_best_case[3],linear_best_case[4]))
print("Performance Report:")
print(classification_report(linear_best_case[3],linear_best_case[4]))
```

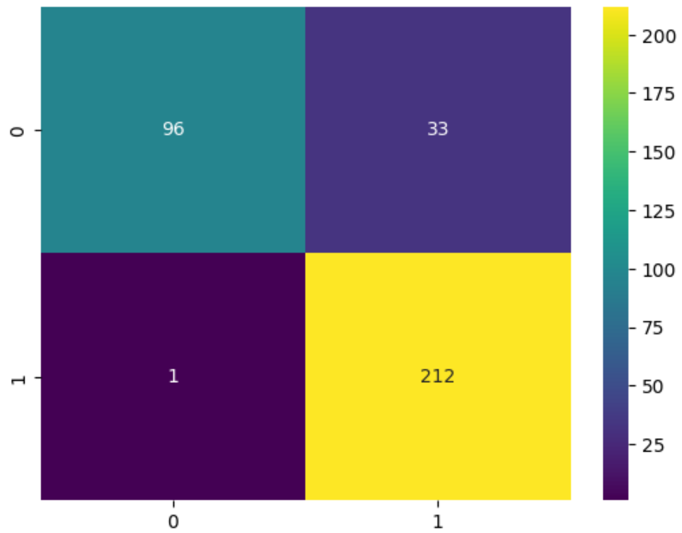
```
Confusion Matrix for test size 0.4 :
[[ 84   8]
 [  5 131]]
Performance Report:
              precision    recall  f1-score   support

           0       0.94      0.91      0.93        92
           1       0.94      0.96      0.95       136

    accuracy                           0.94       228
   macro avg       0.94      0.94      0.94       228
weighted avg       0.94      0.94      0.94       228
```

```
cm = confusion_matrix(linear_best_case[3],linear_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```


```
#SVM_Polynomial

pca = PCA(n_components = 2)
poly_best_case[0] = pca.fit_transform(poly_best_case[0])
poly_best_case[1] = pca.transform(poly_best_case[1])
poly_clf = SVC(kernel='poly',random_state=10)
poly_clf.fit(poly_best_case[0],poly_best_case[2])
poly_best_case[4] = poly_clf.predict(poly_best_case[1])

print("Confusion Matrix for test size",size[poly_index],":")
print(confusion_matrix(poly_best_case[3],poly_best_case[4]))
print("Performance Report:")
print(classification_report(poly_best_case[3],poly_best_case[4]))
```

```
Confusion Matrix for test size 0.6 :
[[ 96  33]
 [  1 212]]
Performance Report:
              precision    recall  f1-score   support

           0       0.99      0.74      0.85       129
           1       0.87      1.00      0.93       213

    accuracy                           0.90       342
   macro avg       0.93      0.87      0.89       342
weighted avg       0.91      0.90      0.90       342
```

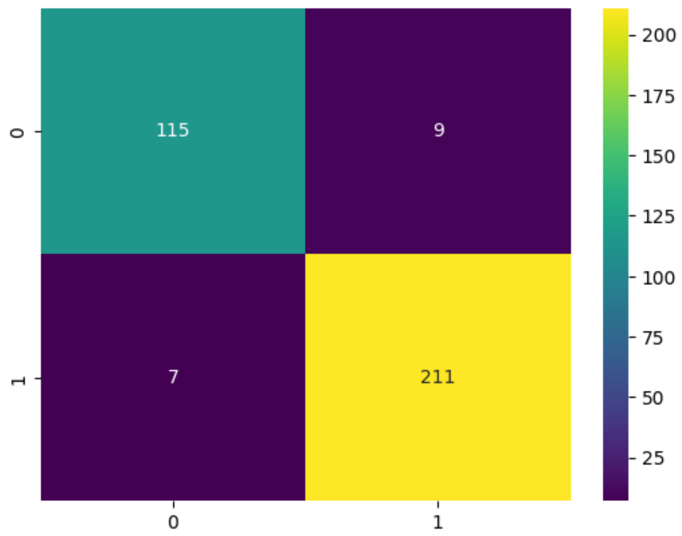
```
cm = confusion_matrix(poly_best_case[3],poly_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#SVM_Gaussian

pca = PCA(n_components = 2)
rbf_best_case[0] = pca.fit_transform(rbf_best_case[0])
rbf_best_case[1] = pca.transform(rbf_best_case[1])
rbf_clf = SVC(kernel='rbf',random_state=10)
rbf_clf.fit(rbf_best_case[0],rbf_best_case[2])
rbf_best_case[4] = rbf_clf.predict(rbf_best_case[1])

print("Confusion Matrix for test size",size[rbf_index],":")
print(confusion_matrix(rbf_best_case[3],rbf_best_case[4]))
print("Performance Report:")
print(classification_report(rbf_best_case[3],rbf_best_case[4]))
```

```
Confusion Matrix for test size 0.6 :
[[115   9]
 [  7 211]]
Performance Report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.93       124
           1       0.96      0.97      0.96       218

    accuracy                           0.95       342
   macro avg       0.95      0.95      0.95       342
weighted avg       0.95      0.95      0.95       342
```

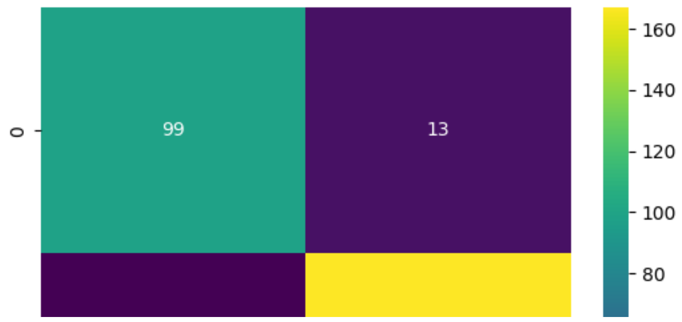
```
cm = confusion_matrix(rbf_best_case[3],rbf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#SVM_Sigmoid

pca = PCA(n_components = 2)
sigmoid_best_case[0] = pca.fit_transform(sigmoid_best_case[0])
sigmoid_best_case[1] = pca.transform(sigmoid_best_case[1])
sigmoid_clf = SVC(kernel='sigmoid',random_state=10)
sigmoid_clf.fit(sigmoid_best_case[0],sigmoid_best_case[2])
sigmoid_best_case[4] = sigmoid_clf.predict(sigmoid_best_case[1])

print("Confusion Matrix for test size",size[sigmoid_index],":")
print(confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4]))
print("Performance Report:")
print(classification_report(sigmoid_best_case[3],sigmoid_best_case[4]))
```

```
Confusion Matrix for test size 0.5 :
[[ 99  13]
 [  6 167]]
Performance Report:
              precision    recall  f1-score   support

           0       0.94      0.88      0.91       112
           1       0.93      0.97      0.95       173

    accuracy                           0.93       285
   macro avg       0.94      0.92      0.93       285
weighted avg       0.93      0.93      0.93       285
```

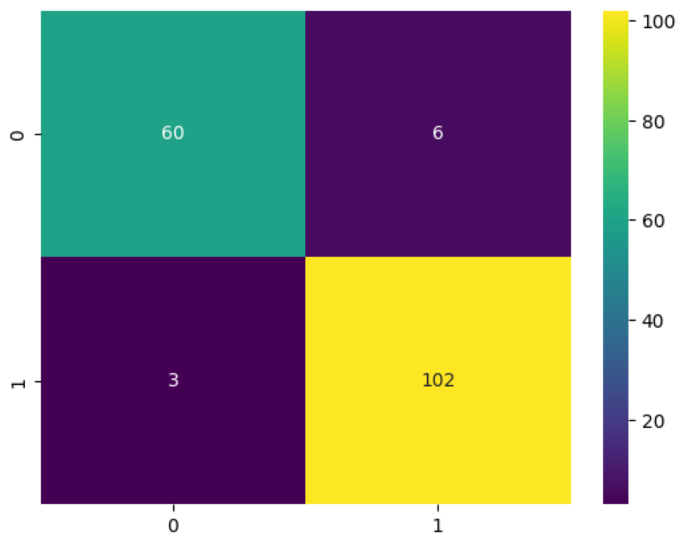
```
cm = confusion_matrix(sigmoid_best_case[3],sigmoid_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#MLP

pca = PCA(n_components = 2)
MLP_best_case[0] = pca.fit_transform(MLP_best_case[0])
MLP_best_case[1] = pca.transform(MLP_best_case[1])
MLP_clf = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=900)
MLP_clf.fit(MLP_best_case[0],MLP_best_case[2])
MLP_best_case[4] = MLP_clf.predict(MLP_best_case[1])

print("Confusion Matrix for test size",size[MLP_index],":")
print(confusion_matrix(MLP_best_case[3],MLP_best_case[4]))
print("Performance Report:")
print(classification_report(MLP_best_case[3],MLP_best_case[4]))
```

```
Confusion Matrix for test size 0.3 :
[[ 60   6]
 [  3 102]]
Performance Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        66
           1       0.94      0.97      0.96       105

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

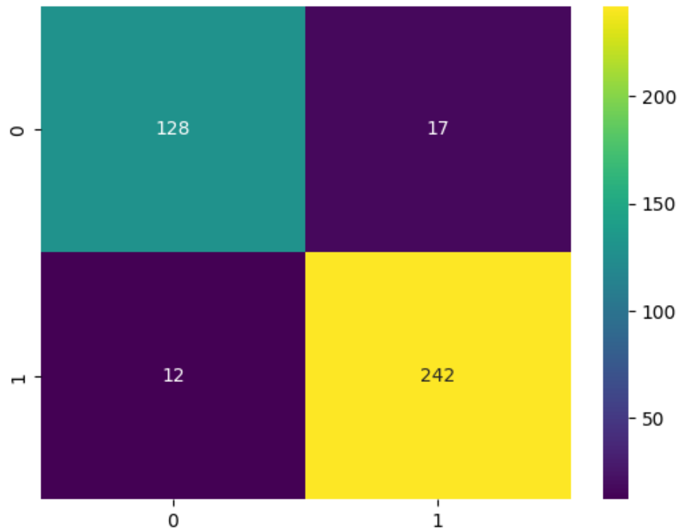
```
cm = confusion_matrix(MLP_best_case[3],MLP_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```




```
#RandomForest

pca = PCA(n_components = 2)
rf_best_case[0] = pca.fit_transform(rf_best_case[0])
rf_best_case[1] = pca.transform(rf_best_case[1])
RFclf = RandomForestClassifier(n_estimators=20)
RFclf.fit(rf_best_case[0],rf_best_case[2])
rf_best_case[4] = RFclf.predict(rf_best_case[1])

print("Confusion Matrix for test size",size[rf_index],":")
print(confusion_matrix(rf_best_case[3],rf_best_case[4]))
print("Performance Report:")
print(classification_report(rf_best_case[3],rf_best_case[4]))
```

```
Confusion Matrix for test size 0.7 :
[[128  17]
 [ 12 242]]
Performance Report:
              precision    recall  f1-score   support

           0       0.91      0.88      0.90       145
           1       0.93      0.95      0.94       254

    accuracy                           0.93       399
   macro avg       0.92      0.92      0.92       399
weighted avg       0.93      0.93      0.93       399
```

```
cm = confusion_matrix(rf_best_case[3],rf_best_case[4])
sns.heatmap(cm,annot=True,fmt="d",cmap='viridis')
plt.show()
```



✓ 0s completed at 5:23 PM