In [31]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import plot_roc_curve, plot_precision_recall_curve
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import metrics
```

DATA PREPROCESSING

In [32]:

```
header=["loc","v(g)","ev(g)","iv(g)","n","v","l","d","i","e","b","t","lOCode","lOComment","
data=pd.read_csv("D://Downloads/Software/Software Dataset/promise4_useful.txt",names=header
data.head()
```

Out[32]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlan
0	14	2.0	1	2	22	88.00	0.17	5.79	15.21	509.14	...	8	0	
1	10	2.0	1	2	18	64.53	0.14	7.00	9.22	451.71	...	8	0	
2	8	1.0	1	1	10	31.70	0.50	2.00	15.85	63.40	...	3	0	
3	6	1.0	1	1	5	11.61	0.67	1.50	7.74	17.41	...	2	0	
4	14	2.0	1	1	11	36.54	0.27	3.75	9.74	137.03	...	3	9	

5 rows × 22 columns

In [33]:

```
data=pd.DataFrame(data)

data.defects=data.defects.replace('yes',1)
data.defects=data.defects.replace('no',0)
```

In [132]:

```
temp=np.array(data['defects'])
z=0
o=0
for i in temp:
    if(i==1):
        o+=1
    else:
        z+=1
print("ones: %d, zeroes: %d" %(o,z))
```

```
ones: 106, zeroes: 106
```

In [34]:

```
arr=np.array(data.defects)
print(np.where(arr==1))  #use shuffle as 1s and 0s are together
```

```
(array([106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
       119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131,
       132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
       145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157,
       158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
       171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183,
       184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196,
       197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
       210, 211], dtype=int64),)
```

In [35]:

```
for i in range(16,len(header)-1):
    data[header[i]]=pd.to_numeric(data[header[i]], errors='coerce').astype('float32')
```

In [36]:

```
data=data.dropna(axis=0,how='any')
```

In [37]:

```
defects=data.loc[:,'defects']
data=data.drop('defects',axis=1)
```

In [38]:

```
from sklearn.preprocessing import Normalizer
transformer=Normalizer().fit(data)
x_scaled=transformer.transform(data)
data = pd.DataFrame(x_scaled,columns = ["loc","v(g)","ev(g)","iv(g)","n","v","l","d","i","e
data.head()
```

Out[38]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	
0	0.026984	0.003855	0.001927	0.003855	0.042404	0.169615	0.000328	0.011160	0.029316	0
1	0.021836	0.004367	0.002184	0.004367	0.039305	0.140908	0.000306	0.015285	0.020133	0
2	0.107276	0.013409	0.013409	0.013409	0.134095	0.425080	0.006705	0.026819	0.212540	0
3	0.244858	0.040810	0.040810	0.040810	0.204048	0.473800	0.027342	0.061214	0.315866	0
4	0.097039	0.013863	0.006931	0.006931	0.076245	0.253272	0.001871	0.025993	0.067512	0

5 rows × 21 columns

In [39]:

```
data['defects']=defects
#data=data.drop('lOCodeAndComment',axis=1)
#data=data.drop('lOBlank',axis=1)
#data=data.drop('lOComment',axis=1)
data=data.dropna(axis=0,how='any')
data.head()
```

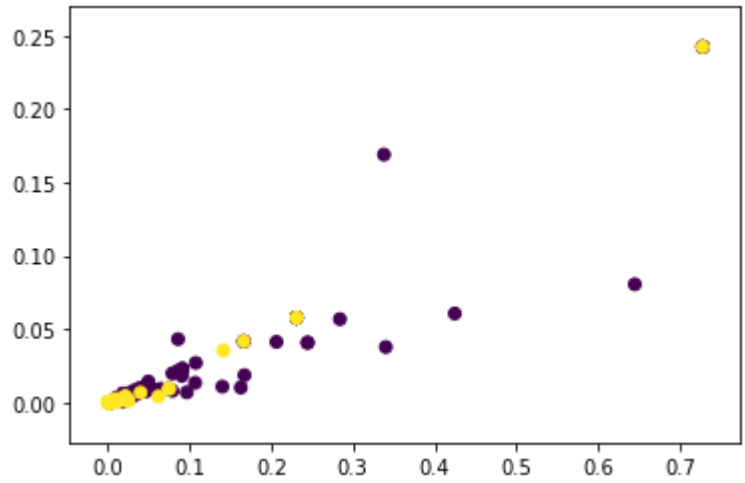
Out[39]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	
0	0.026984	0.003855	0.001927	0.003855	0.042404	0.169615	0.000328	0.011160	0.029316	0
1	0.021836	0.004367	0.002184	0.004367	0.039305	0.140908	0.000306	0.015285	0.020133	0
2	0.107276	0.013409	0.013409	0.013409	0.134095	0.425080	0.006705	0.026819	0.212540	0
3	0.244858	0.040810	0.040810	0.040810	0.204048	0.473800	0.027342	0.061214	0.315866	0
4	0.097039	0.013863	0.006931	0.006931	0.076245	0.253272	0.001871	0.025993	0.067512	0

5 rows × 22 columns

In [40]:

```
x=data['loc']
y=data['iv(g)']
z=data['defects']
plt.scatter(x,y,c=z)
plt.show()
```



In [41]:

```
x=data.drop('defects',axis=1).values
y=data[["defects"]].values
```

In [42]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y)
```

In [43]:

```
print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)
```

```
(159, 21) (159, 1) (53, 21) (53, 1)
```

In [44]:

```
ytrain, ytest=ytrain.flatten(), ytest.flatten()
```

In [45]:

```
z=0
o=0
for i in ytrain:
    if(i==1):
        o+=1
    else:
        z+=1
print("ones: %d, zeroes: %d" %(o,z))
```

```
ones: 80, zeroes: 79
```

In [46]:

```
for i in range(0,len(ytrain)):
    if(ytrain[i]==1):
        print(xtrain[i])
        print("\n")
```

```
[4.71109558e-03 7.06664337e-04 3.92591298e-04 3.92591298e-04
 8.63700856e-03 4.49941035e-02 3.92591298e-06 1.74074982e-03
 2.02969701e-03 9.97368771e-01 1.49184693e-05 5.54095507e-02
 2.98369387e-03 1.33481041e-03 1.57036519e-04 0.00000000e+00
 1.49184693e-03 1.41332867e-03 5.33924166e-03 3.29776691e-03
 1.33481041e-03]


[5.89570755e-03 8.18848271e-04 1.63769654e-04 3.27539308e-04
 1.68682744e-02 8.97719737e-02 1.47392689e-05 1.81293007e-03
 8.10659788e-03 9.94139120e-01 2.94785378e-05 5.52296782e-02
 4.25801101e-03 0.00000000e+00 8.18848271e-04 1.63769654e-04
 2.12900550e-03 4.42178066e-03 9.33487029e-03 7.53340409e-03
 1.47392689e-03]


[8.47929148e-03 8.07551569e-04 2.01887892e-04 6.05663677e-04
 1.57472556e-02 7.94348101e-02 1.61510314e-05 2.52965529e-03
 6.34129870e-03 9.95032742e-01 2.62454260e-05 5.52789238e-02
```

BASE PREDICTIORS

1- SVM

In [47]:

```
from sklearn.svm import SVC
```

In [48]:

```
svm_model=SVC()
svm_model.fit(xtrain,ytrain)
```

Out[48]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [49]:

```
predsvm=svm_model.predict(xtest)
svm_model.score(xtest,ytest)*100
```

Out[49]:

```
69.81132075471697
```

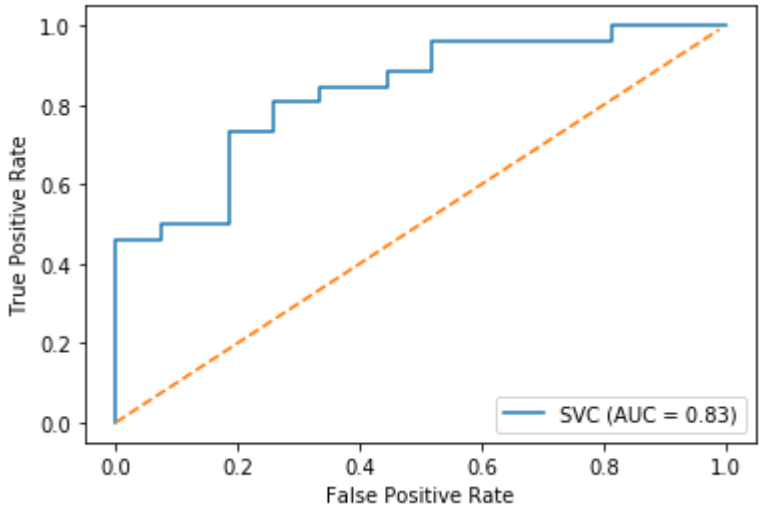
In [50]:

```
accuracy=confusion_matrix(ytest,predsvm)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predsvm, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predsvm))
print("\n")
print("   P    N")
print(confusion_matrix(ytest,predsvm))
print("\n")
print(classification_report(ytest,predsvm))
```

```
Accuracy:  69.81132075471697
Probability of detection of defect(Recall, pd):  0.8461538461538461
Probability of false alarm(pf):  0.21052631578947367
Probability of correct detection(Precision):  0.6470588235294118


F1-score or FM:  0.7333333333333334
AUC value:  0.7008547008547009


   P    N
[[15 12]
 [ 4 22]]


              precision    recall  f1-score   support

           0       0.79      0.56      0.65        27
           1       0.65      0.85      0.73        26

    accuracy                           0.70        53
   macro avg       0.72      0.70      0.69        53
weighted avg       0.72      0.70      0.69        53
```

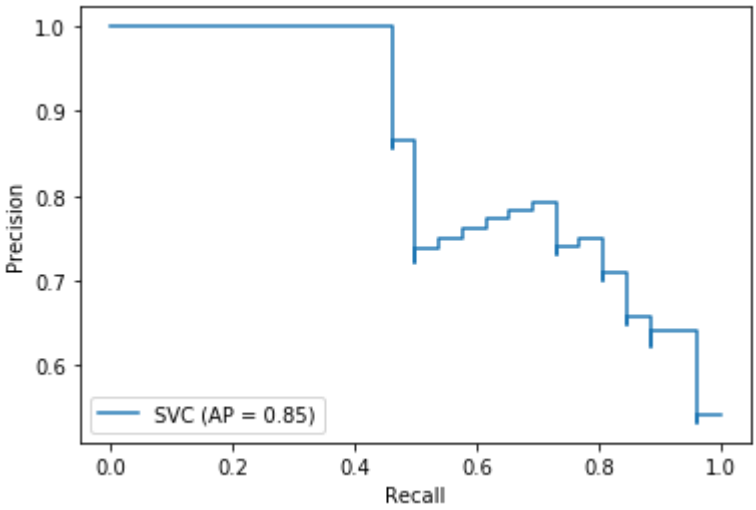
In [51]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(svm_model, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [52]:

```
disp = plot_precision_recall_curve(svm_model, xtest, ytest)
plt.show()
```



2- KNN

In [126]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=2)
```

In [127]:

```
knn.fit(xtrain,ytrain)
```

Out[127]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=2, p=2,
                     weights='uniform')
```

In [128]:

```
predknn=knn.predict(xtest)
knn.score(xtest,ytest)*100
```

Out[128]:

```
77.35849056603774
```

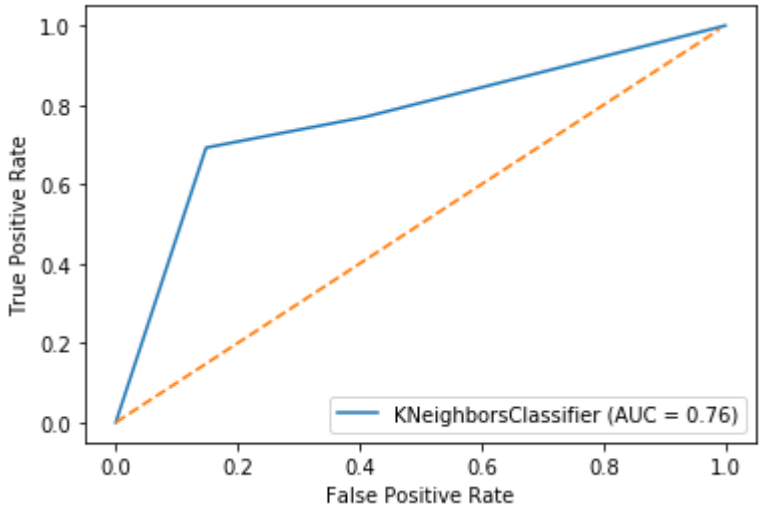
In [129]:

```
accuracy=confusion_matrix(ytest,predknn)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predknn, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predknn))
print("\n")
print(confusion_matrix(ytest,predknn))
print("\n")
print(classification_report(ytest,predknn))
```

```
Accuracy:  77.35849056603774
Probability of detection of defect(Recall, pd):  0.6923076923076923
Probability of false alarm(pf):  0.25806451612903225
Probability of correct detection(Precision):  0.8181818181818182


F1-score or FM:  0.7500000000000001
AUC value:  0.7720797720797721


[[23  4]
 [ 8 18]]


              precision    recall  f1-score   support

           0       0.74      0.85      0.79        27
           1       0.82      0.69      0.75        26

    accuracy                           0.77        53
   macro avg       0.78      0.77      0.77        53
weighted avg       0.78      0.77      0.77        53
```

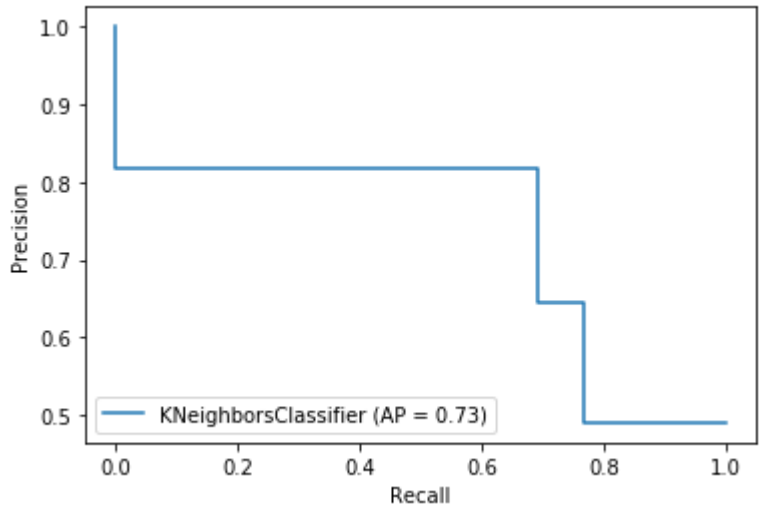
In [130]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(knn, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [131]:

```
disp = plot_precision_recall_curve(knn, xtest, ytest)
plt.show()
```



In [59]:

```
# try K=1 through K=25 and record testing accuracy
k_range = range(1, 15)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 26
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xtrain, ytrain)
    y_pred = knn.predict(xtest)
    scores.append(metrics.accuracy_score(ytest, y_pred))

print(scores)
```

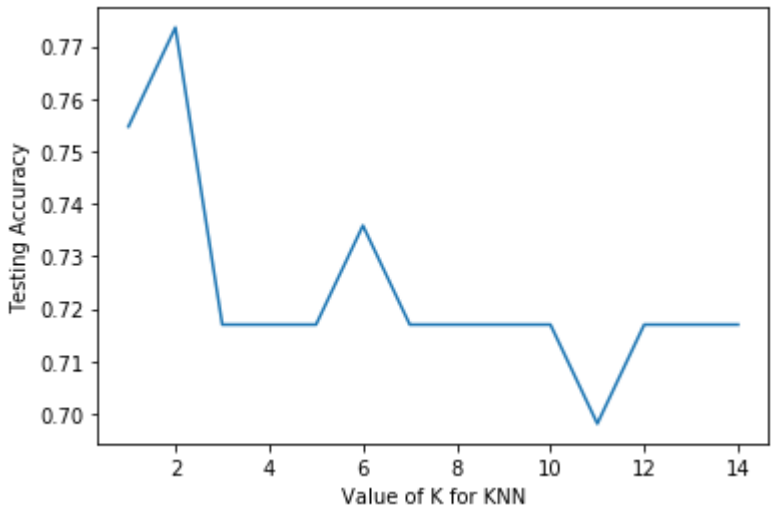
```
[0.7547169811320755, 0.7735849056603774, 0.7169811320754716, 0.7169811320754
716, 0.7169811320754716, 0.7358490566037735, 0.7169811320754716, 0.716981132
0754716, 0.7169811320754716, 0.7169811320754716, 0.6981132075471698, 0.71698
11320754716, 0.7169811320754716, 0.7169811320754716]
```

In [60]:

```
# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Out[60]:

```
Text(0, 0.5, 'Testing Accuracy')
```



3- NAIVE BAYES

In [61]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

In [62]:

```
gnb.fit(xtrain,ytrain)
```

Out[62]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [63]:

```
predg=gnb.predict(xtest)
gnb.score(xtest,ytest)*100
```

Out[63]:

```
62.264150943396224
```

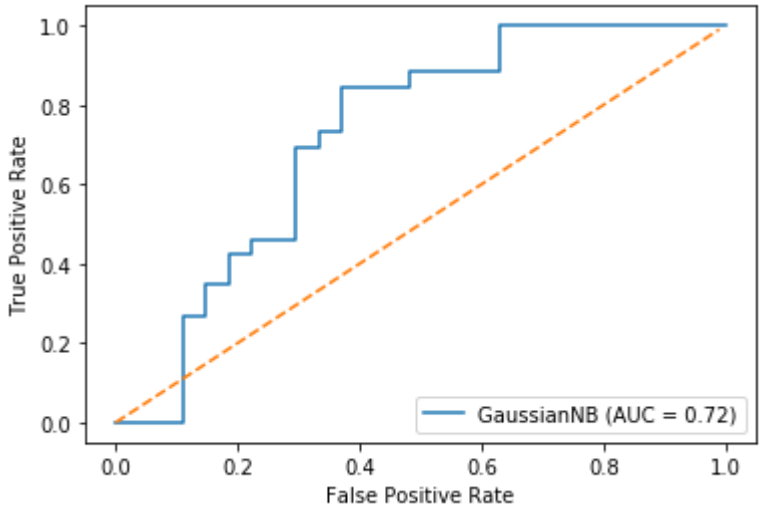
In [64]:

```
accuracy=confusion_matrix(ytest,predg)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predg, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predg))
print("\n")
print(confusion_matrix(ytest,predg))
print("\n")
print(classification_report(ytest,predg))
```

```
Accuracy:  62.264150943396224
Probability of detection of defect(Recall, pd):  0.8846153846153846
Probability of false alarm(pf):  0.23076923076923078
Probability of correct detection(Precision):  0.575


F1-score or FM:  0.696969696969697
AUC value:  0.6274928774928774


[[10 17]
 [ 3 23]]


              precision    recall  f1-score   support

           0       0.77      0.37      0.50        27
           1       0.57      0.88      0.70        26

    accuracy                           0.62        53
   macro avg       0.67      0.63      0.60        53
weighted avg       0.67      0.62      0.60        53
```

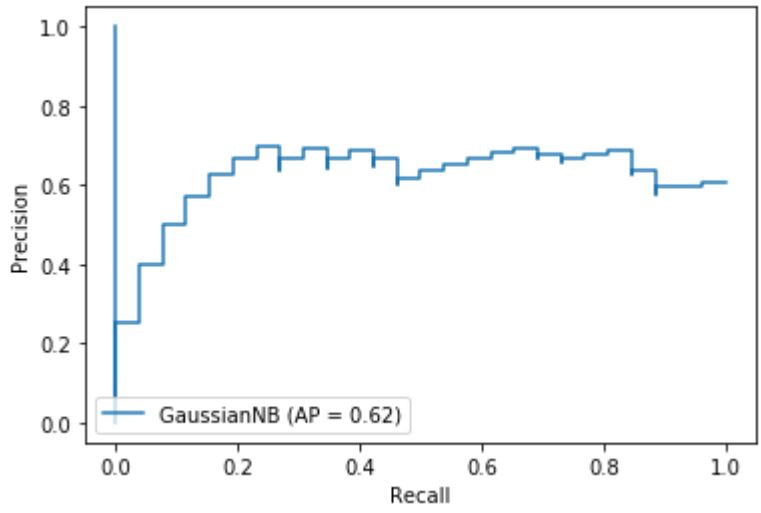
In [65]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(gnb, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [66]:

```
disp = plot_precision_recall_curve(gnb, xtest, ytest)
plt.show()
```



In [67]:

```
c=0
l=len(ytest)
for i in range(0,l):
    if(predg[i]!=ytest[i]):
        c=c+1
print("Number of mislabeled points out of a total %d points : %d" %(l,c))
```

```
Number of mislabeled points out of a total 53 points : 20
```

4- LOGISTIC REGRESSION

In [68]:

```
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()
```

In [69]:

```
logmodel.fit(xtrain,ytrain)
```

Out[69]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [70]:

```
predlog=logmodel.predict(xtest)
logistic_score=logmodel.score(xtest,ytest)*100
logistic_score
```

Out[70]:

```
71.69811320754717
```

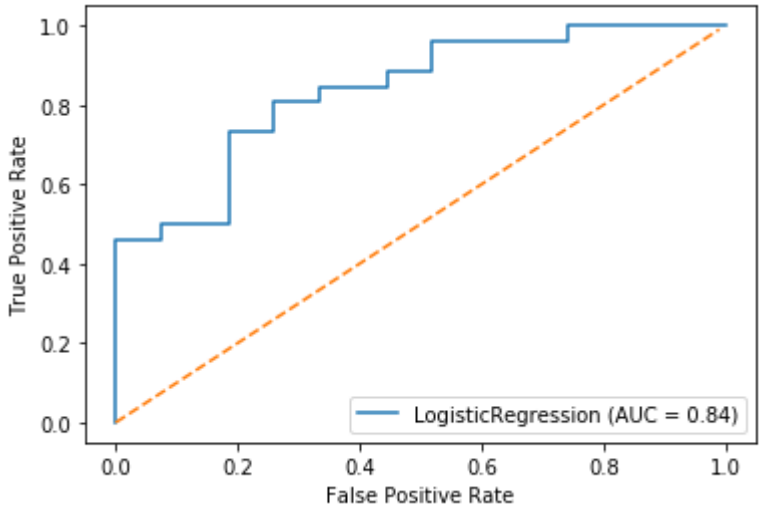
In [71]:

```
accuracy=confusion_matrix(ytest,predlog)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predlog, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predlog))
print("\n")
print(confusion_matrix(ytest,predlog))
print("\n")
print(classification_report(ytest,predlog))
```

```
Accuracy:  71.69811320754717
Probability of detection of defect(Recall, pd):  0.8846153846153846
Probability of false alarm(pf):  0.16666666666666666
Probability of correct detection(Precision):  0.6571428571428571


F1-score or FM:  0.7540983606557377
AUC value:  0.7200854700854701


[[15 12]
 [ 3 23]]


              precision    recall  f1-score   support

           0       0.83      0.56      0.67        27
           1       0.66      0.88      0.75        26

    accuracy                           0.72        53
   macro avg       0.75      0.72      0.71        53
weighted avg       0.75      0.72      0.71        53
```

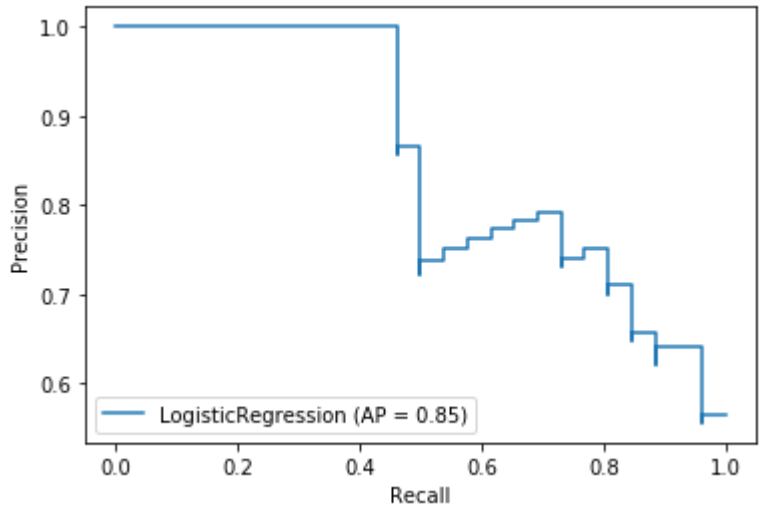
In [72]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(logmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [73]:

```
disp = plot_precision_recall_curve(logmodel, xtest, ytest)
plt.show()
```



5- MLP

In [74]:

```
from sklearn.neural_network import MLPClassifier
```

In [75]:

```
model=MLPClassifier(hidden_layer_sizes=(20,20),max_iter=2000)
model.fit(xtrain,ytrain)
```

Out[75]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.
9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(20, 20), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=2000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

In [76]:

```
predn=model.predict(xtest)
model.score(xtest,ytest)*100
```

Out[76]:

```
75.47169811320755
```

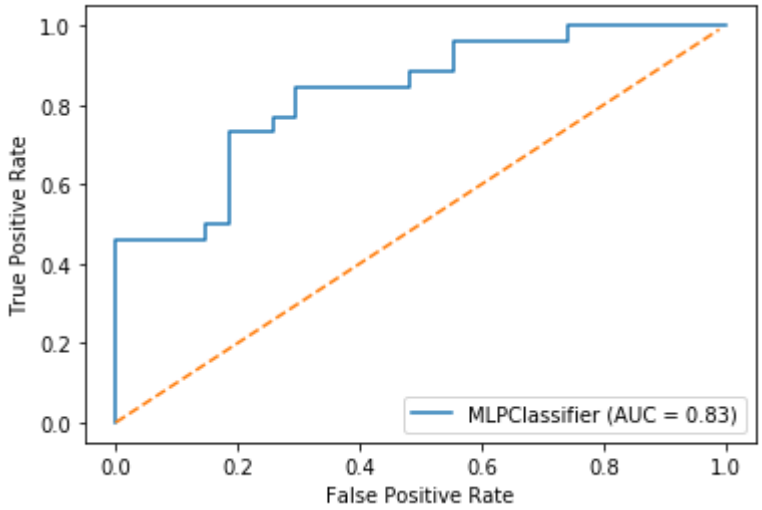
In [77]:

```
accuracy=confusion_matrix(ytest,predn)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predn, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predn))
print("\n")
print(confusion_matrix(ytest,predn))
print("\n")
print(classification_report(ytest,predn))
```

```
Accuracy:  75.47169811320755
Probability of detection of defect(Recall, pd):  0.8076923076923077
Probability of false alarm(pf):  0.20833333333333334
Probability of correct detection(Precision):  0.7241379310344828


F1-score or FM:  0.7636363636363636
AUC value:  0.7556980056980057


[[19  8]
 [ 5 21]]


              precision    recall  f1-score   support

           0       0.79      0.70      0.75        27
           1       0.72      0.81      0.76        26

    accuracy                           0.75        53
   macro avg       0.76      0.76      0.75        53
weighted avg       0.76      0.75      0.75        53
```

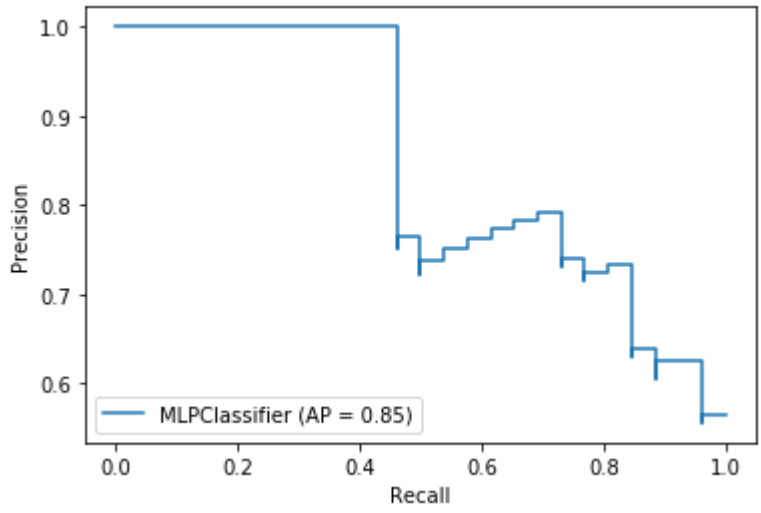
In [78]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(model, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [79]:

```
disp = plot_precision_recall_curve(model, xtest, ytest)
plt.show()
```



6- DECISION TREE

In [80]:

```
from sklearn import tree
```

In [81]:

```
tmodel=tree.DecisionTreeClassifier()
tmodel.fit(xtrain,ytrain)
```

Out[81]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [82]:

```
predt=tmodel.predict(xtest)
tmodel.score(xtest,ytest)*100
```

Out[82]:

```
69.81132075471697
```

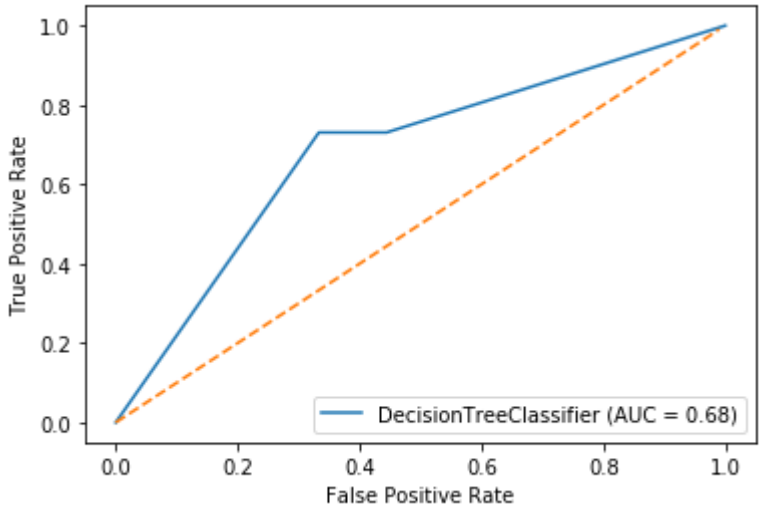
In [83]:

```
accuracy=confusion_matrix(ytest,predt)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predt, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predt))
print("\n")
print(confusion_matrix(ytest,predt))
print("\n")
print(classification_report(ytest,predt))
```

```
Accuracy:  69.81132075471697
Probability of detection of defect(Recall, pd):  0.7307692307692307
Probability of false alarm(pf):  0.28
Probability of correct detection(Precision):  0.6785714285714286


F1-score or FM:  0.7037037037037038
AUC value:  0.6987179487179488


[[18  9]
 [ 7 19]]


              precision    recall  f1-score   support

           0       0.72      0.67      0.69        27
           1       0.68      0.73      0.70        26

    accuracy                           0.70        53
   macro avg       0.70      0.70      0.70        53
weighted avg       0.70      0.70      0.70        53
```

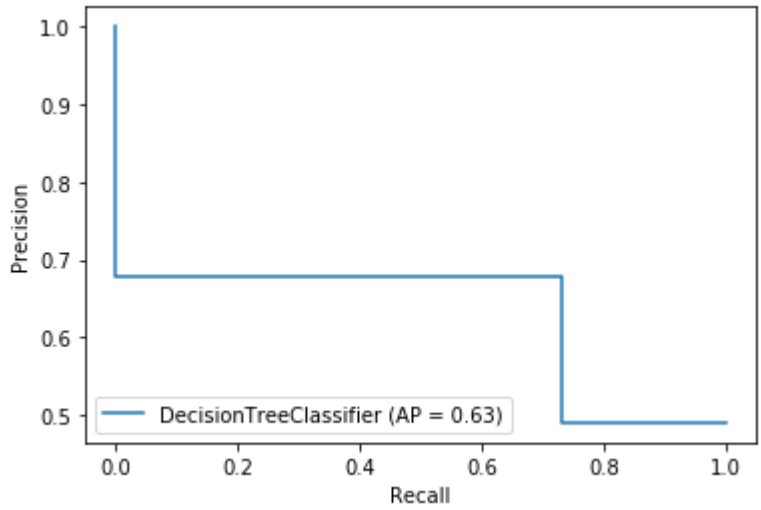
In [84]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(tmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [85]:

```
disp = plot_precision_recall_curve(tmodel, xtest, ytest)
plt.show()
```



ENSEMBLE PREDICTORS

1- ADABOOST

In [86]:

```
from sklearn.ensemble import AdaBoostClassifier
```

In [87]:

```
adamodel = AdaBoostClassifier(n_estimators=100)
adamodel.fit(xtrain,ytrain)
```

Out[87]:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=
1.0,
                   n_estimators=100, random_state=None)
```

In [88]:

```
predada=adamodel.predict(xtest)
adamodel.score(xtest,ytest)*100
```

Out[88]:

```
71.69811320754717
```

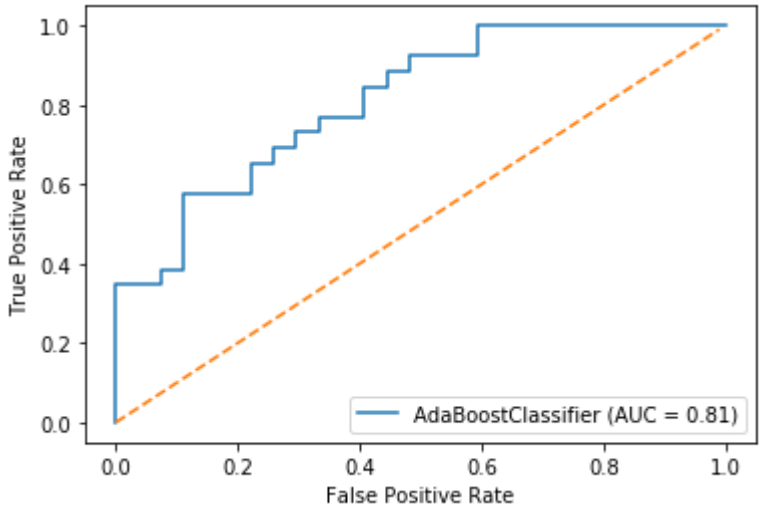
In [89]:

```
accuracy=confusion_matrix(ytest,predada)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predada, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predada))
print("\n")
print(confusion_matrix(ytest,predada))
print("\n")
print(classification_report(ytest,predada))
```

```
Accuracy:  71.69811320754717
Probability of detection of defect(Recall, pd):  0.7307692307692307
Probability of false alarm(pf):  0.2692307692307692
Probability of correct detection(Precision):  0.7037037037037037


F1-score or FM:  0.7169811320754716
AUC value:  0.7172364672364673


[[19  8]
 [ 7 19]]


              precision    recall  f1-score   support

           0       0.73      0.70      0.72        27
           1       0.70      0.73      0.72        26

    accuracy                           0.72        53
   macro avg       0.72      0.72      0.72        53
weighted avg       0.72      0.72      0.72        53
```

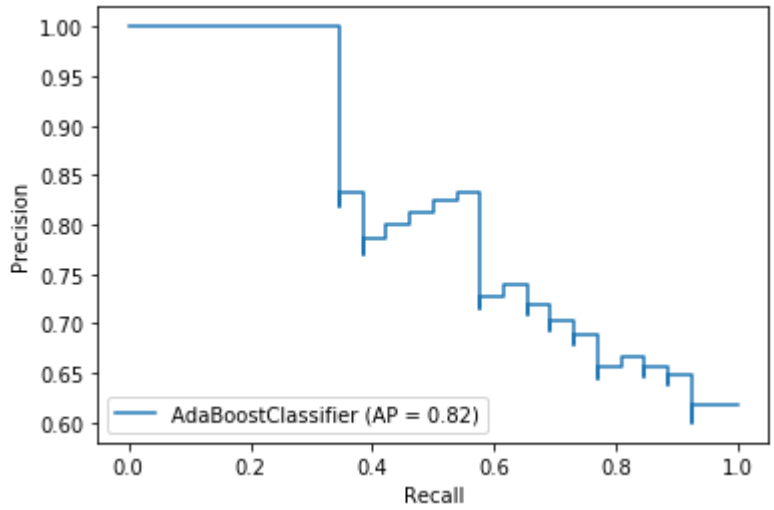
In [90]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(adamodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [91]:

```
disp = plot_precision_recall_curve(adamodel, xtest, ytest)
plt.show()
```



2- BAGGING

In [92]:

```
from sklearn.ensemble import BaggingClassifier
```

In [93]:

```
bagmodel = BaggingClassifier(base_estimator=None, n_estimators=10) #default=decision tree,
bagmodel.fit(xtrain, ytrain)
```

Out[93]:

```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=Fa
lse,
                  max_features=1.0, max_samples=1.0, n_estimators=10,
                  n_jobs=None, oob_score=False, random_state=None, verbose=
0,
                  warm_start=False)
```

In [94]:

```
predbag=bagmodel.predict(xtest)
bagmodel.score(xtest, ytest)*100
```

Out[94]:

```
77.35849056603774
```

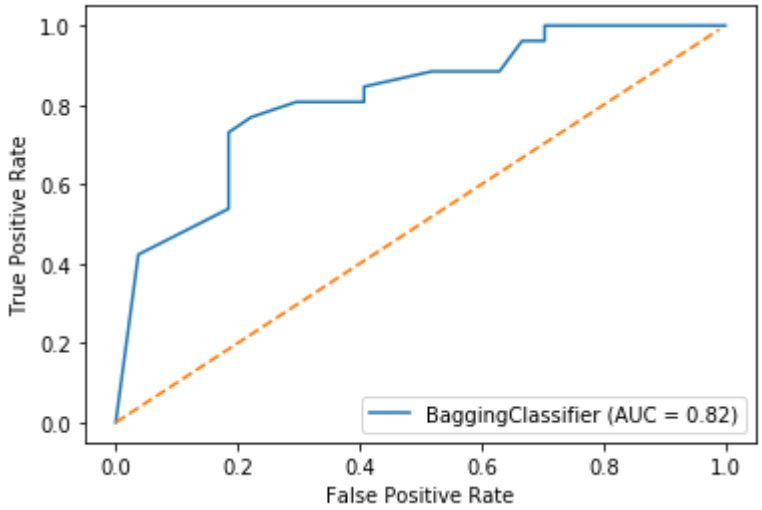
In [95]:

```
accuracy=confusion_matrix(ytest,predbag)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predbag, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predbag))
print("\n")
print(confusion_matrix(ytest,predbag))
print("\n")
print(classification_report(ytest,predbag))
```

```
Accuracy:  77.35849056603774
Probability of detection of defect(Recall, pd):  0.7692307692307693
Probability of false alarm(pf):  0.2222222222222222
Probability of correct detection(Precision):  0.7692307692307693


F1-score or FM:  0.7692307692307693
AUC value:  0.7735042735042735


[[21  6]
 [ 6 20]]


              precision    recall  f1-score   support

           0       0.78      0.78      0.78        27
           1       0.77      0.77      0.77        26

    accuracy                           0.77        53
   macro avg       0.77      0.77      0.77        53
weighted avg       0.77      0.77      0.77        53
```

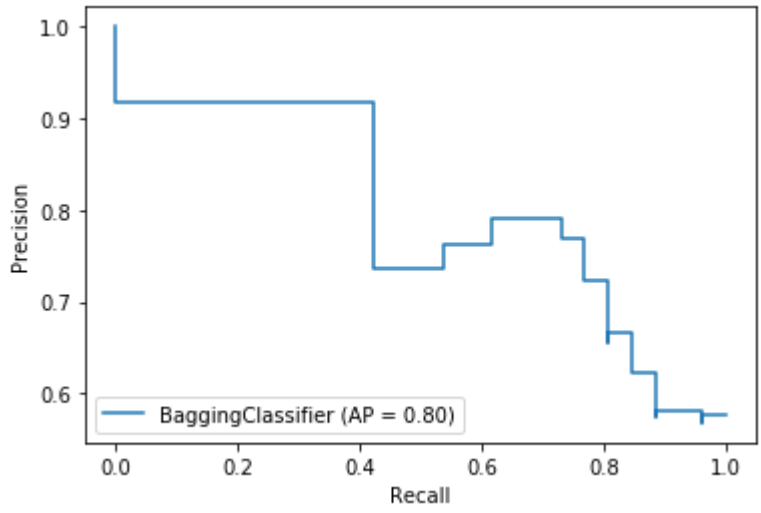
In [96]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(bagmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [97]:

```
disp = plot_precision_recall_curve(bagmodel, xtest, ytest)
plt.show()
```



3- Extra_Tree_Classifier

In [98]:

```
from sklearn.ensemble import ExtraTreesClassifier
```

In [99]:

```
exmodel = ExtraTreesClassifier(n_estimators=100)
exmodel.fit(xtrain, ytrain)
```

Out[99]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

In [100]:

```
predex=exmodel.predict(xtest)
exmodel.score(xtest,ytest)*100
```

Out[100]:

```
75.47169811320755
```

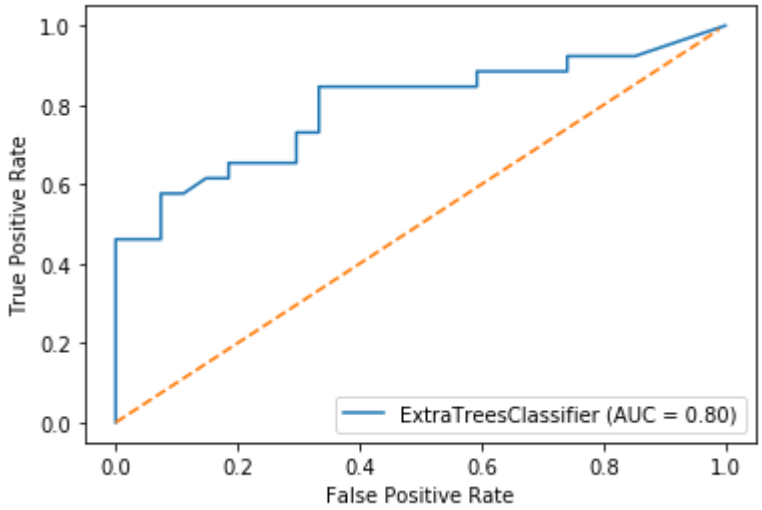
In [101]:

```
accuracy=confusion_matrix(ytest,predex)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predex, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predex))
print("\n")
print(confusion_matrix(ytest,predex))
print("\n")
print(classification_report(ytest,predex))
```

```
Accuracy:  75.47169811320755
Probability of detection of defect(Recall, pd):  0.8461538461538461
Probability of false alarm(pf):  0.18181818181818182
Probability of correct detection(Precision):  0.7096774193548387


F1-score or FM:  0.7719298245614036
AUC value:  0.7564102564102566


[[18  9]
 [ 4 22]]


              precision    recall  f1-score   support

           0       0.82      0.67      0.73        27
           1       0.71      0.85      0.77        26

    accuracy                           0.75        53
   macro avg       0.76      0.76      0.75        53
weighted avg       0.76      0.75      0.75        53
```

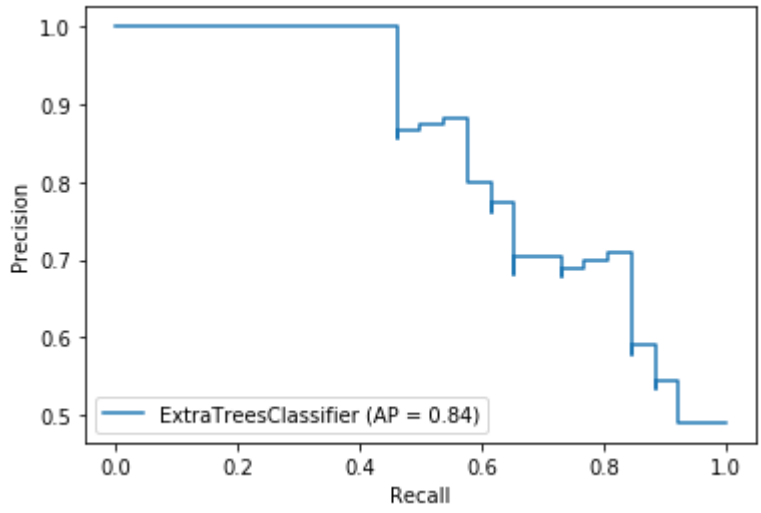
In [102]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(exmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [103]:

```
disp = plot_precision_recall_curve(exmodel, xtest, ytest)
plt.show()
```



4- Gradient_Boosting_Classifier

In [104]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [105]:

```
gradmodel = GradientBoostingClassifier()
gradmodel.fit(xtrain,ytrain)
```

Out[105]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=Non
e,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=Non
e,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [106]:

```
predgrad=gradmodel.predict(xtest)
gradmodel.score(xtest,ytest)*100
```

Out[106]:

```
73.58490566037736
```

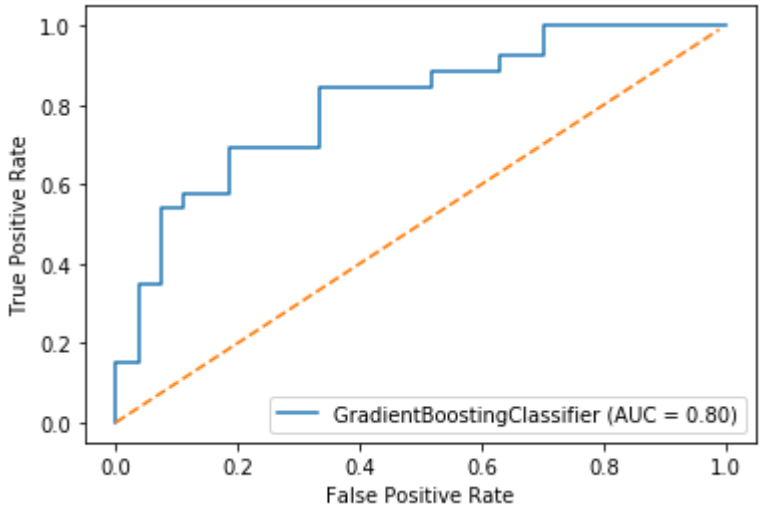
In [107]:

```
accuracy=confusion_matrix(ytest,predgrad)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predgrad, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predgrad))
print("\n")
print(confusion_matrix(ytest,predgrad))
print("\n")
print(classification_report(ytest,predgrad))
```

```
Accuracy:  73.58490566037736
Probability of detection of defect(Recall, pd):  0.8076923076923077
Probability of false alarm(pf):  0.21739130434782608
Probability of correct detection(Precision):  0.7


F1-score or FM:  0.75
AUC value:  0.7371794871794872


[[18  9]
 [ 5 21]]


              precision    recall  f1-score   support

           0       0.78      0.67      0.72        27
           1       0.70      0.81      0.75        26

    accuracy                           0.74        53
   macro avg       0.74      0.74      0.73        53
weighted avg       0.74      0.74      0.73        53
```

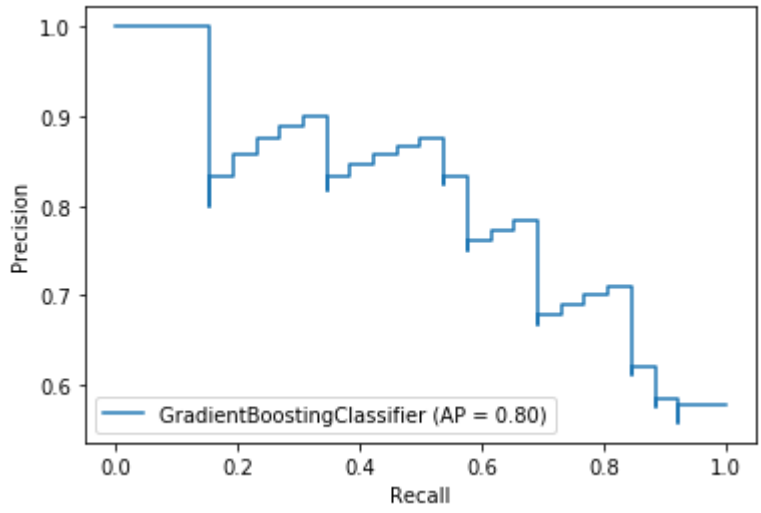
In [108]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(gradmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [109]:

```
disp = plot_precision_recall_curve(gradmodel, xtest, ytest)
plt.show()
```



5- Random_Forest_Classifier

In [110]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [111]:

```
randmodel = RandomForestClassifier()
randmodel.fit(xtrain,ytrain)
```

Out[111]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [112]:

```
predrand=randmodel.predict(xtest)
randmodel.score(xtest,ytest)*100
```

Out[112]:

```
71.69811320754717
```

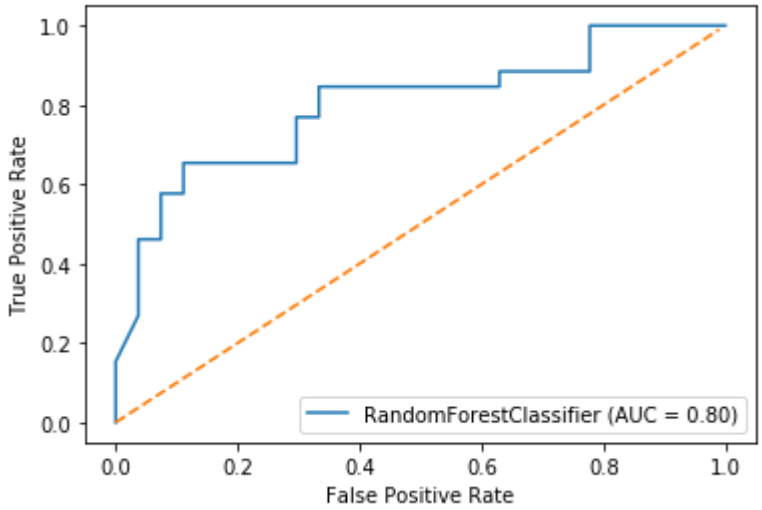
In [113]:

```
accuracy=confusion_matrix(ytest,predrand)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predrand, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predrand))
print("\n")
print(confusion_matrix(ytest,predrand))
print("\n")
print(classification_report(ytest,predrand))
```

```
Accuracy:  71.69811320754717
Probability of detection of defect(Recall, pd):  0.7307692307692307
Probability of false alarm(pf):  0.2692307692307692
Probability of correct detection(Precision):  0.7037037037037037


F1-score or FM:  0.7169811320754716
AUC value:  0.7172364672364673


[[19  8]
 [ 7 19]]


              precision    recall  f1-score   support

           0       0.73      0.70      0.72        27
           1       0.70      0.73      0.72        26

    accuracy                           0.72        53
   macro avg       0.72      0.72      0.72        53
weighted avg       0.72      0.72      0.72        53
```

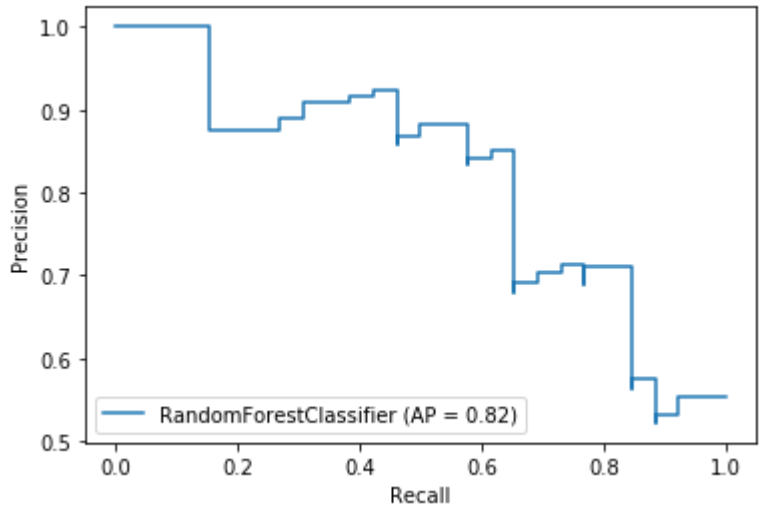
In [114]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(randmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [115]:

```
disp = plot_precision_recall_curve(randmodel, xtest, ytest)
plt.show()
```



6- Stacking_Classifier

In [116]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
```

In [117]:

```
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
              ('svr', make_pipeline(StandardScaler(), LinearSVC(random_state=42)))]

stmodel = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
stmodel.fit(xtrain,ytrain)
```

```
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[117]:

```
StackingClassifier(cv=None,
                   estimators=[('rf',
                                RandomForestClassifier(bootstrap=True,
                                                       ccp_alpha=0.0,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='aut
o',
                                                       max_leaf_nodes=Non
e,
                                                       max_samples=None,
                                                       min_impurity_decrea
se=0.0,
                                                       min_impurity_split=
None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=
2,
                                                       min_weight_fraction
_leaf=0.0,
                                                       n_estimators=10,
                                                       n_jobs=None,...
                                                           tol=0.0001,
                                                           verbose=0))],
                                         verbose=False))],
                   final_estimator=LogisticRegression(C=1.0, class_weight=
```

```
None,
                                                   dual=False,
                                                   fit_intercept=True,
                                                   intercept_scaling=1,
                                                   l1_ratio=None,
                                                   max_iter=100,
                                                   multi_class='auto',
                                                   n_jobs=None, penalty
='l2',
                                                   random_state=None,
                                                   solver='lbfgs',
                                                   tol=0.0001, verbose=
0,
                                                   warm_start=False),
                   n_jobs=None, passthrough=False, stack_method='auto',
                   verbose=0)
```

In [118]:

```
predst=stmodel.predict(xtest)
stmodel.score(xtest,ytest)*100
```

Out[118]:

```
73.58490566037736
```

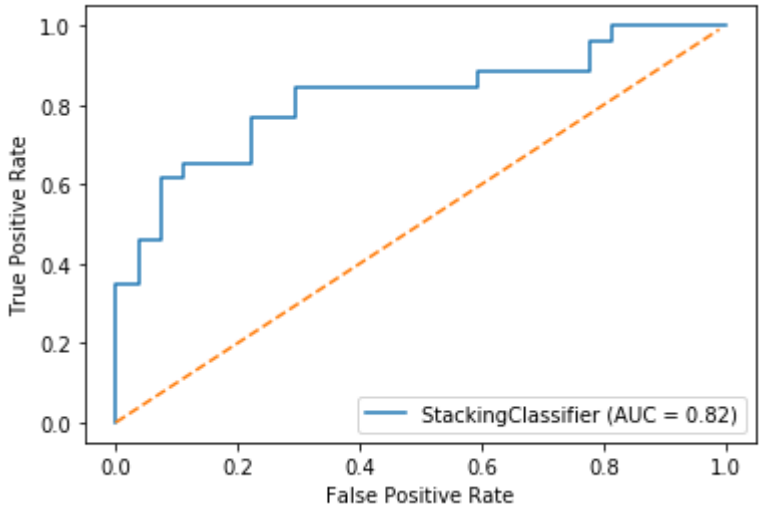
In [119]:

```
accuracy=confusion_matrix(ytest,predst)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predst, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predst))
print("\n")
print(confusion_matrix(ytest,predst))
print("\n")
print(classification_report(ytest,predst))
```

```
Accuracy:  73.58490566037736
Probability of detection of defect(Recall, pd):  0.7692307692307693
Probability of false alarm(pf):  0.24
Probability of correct detection(Precision):  0.7142857142857143


F1-score or FM:  0.7407407407407408
AUC value:  0.7364672364672364


[[19  8]
 [ 6 20]]


              precision    recall  f1-score   support

           0       0.76      0.70      0.73        27
           1       0.71      0.77      0.74        26

    accuracy                           0.74        53
   macro avg       0.74      0.74      0.74        53
weighted avg       0.74      0.74      0.74        53
```

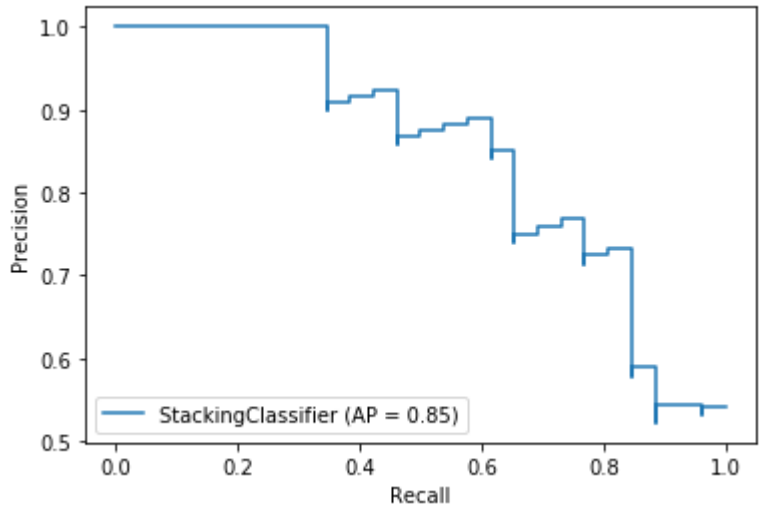
In [120]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(stmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [121]:

```
disp = plot_precision_recall_curve(stmodel, xtest, ytest)
plt.show()
```



7- Voting_Classifier

In [122]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
```

In [123]:

```
clf1 = LogisticRegression()
clf2 = RandomForestClassifier()#n_estimators=50, random_state=1)
clf3 = GaussianNB()
votmodel = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting=
votmodel.fit(xtrain,ytrain)
```

Out[123]:

```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(C=1.0, class_weight=None,
                                                 dual=False, fit_intercept=T
rue,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=10
0,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l2',
                                                 random_state=None,
                                                 solver='lbfgs', tol=0.0001,
                                                 verbose=0, warm_start=Fals
e)),
                             ('rf',
                              RandomForestClassifier(bootstrap=True,
                                                     ccp_alpha=0.0,
                                                     class_weight=None,
                                                     cr...
                                                     max_leaf_nodes=None,
                                                     max_samples=None,
                                                     min_impurity_decrease=
0.0,
                                                     min_impurity_split=Non
e,
                                                     min_samples_leaf=1,
                                                     min_samples_split=2,
                                                     min_weight_fraction_lea
f=0.0,
                                                     n_estimators=100,
                                                     n_jobs=None,
                                                     oob_score=False,
                                                     random_state=None,
                                                     verbose=0,
                                                     warm_start=False)),
                             ('gnb',
                              GaussianNB(priors=None, var_smoothing=1e-0
9))],
                 flatten_transform=True, n_jobs=None, voting='hard',
                 weights=None)
```

In [124]:

```
predvot=votmodel.predict(xtest)
votmodel.score(xtest,ytest)*100
```

Out[124]:

```
71.69811320754717
```

In [125]:

```
accuracy=confusion_matrix(ytest,predvot)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predvot, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predvot))
print("\n")
print(confusion_matrix(ytest,predvot))
print("\n")
print(classification_report(ytest,predvot))
```

```
Accuracy:  71.69811320754717
Probability of detection of defect(Recall, pd):  0.8846153846153846
Probability of false alarm(pf):  0.16666666666666666
Probability of correct detection(Precision):  0.6571428571428571


F1-score or FM:  0.7540983606557377
AUC value:  0.7200854700854701


[[15 12]
 [ 3 23]]


              precision    recall  f1-score   support

           0       0.83      0.56      0.67        27
           1       0.66      0.88      0.75        26

    accuracy                           0.72        53
   macro avg       0.75      0.72      0.71        53
weighted avg       0.75      0.72      0.71        53
```

In [ ]:

```

```

In [ ]:

```

```