In [20]:

```
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 [21]:

```
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/promise5_useful.txt",names=header
data.head()
```

Out[21]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComı
0	91	9	3	2	318	2089.21	0.04	27.68	75.47	57833.24	...	80	
1	109	21	5	18	381	2547.56	0.04	28.37	89.79	72282.68	...	97	
2	505	106	41	82	2339	20696.93	0.01	75.93	272.58	1571506.88	...	457	
3	107	25	7	14	619	4282.78	0.02	52.91	80.95	226588.75	...	103	
4	74	11	1	8	294	1917.93	0.03	28.77	66.66	55178.46	...	60	

5 rows × 22 columns

In [22]:

```
data=pd.DataFrame(data)

data.defects=data.defects.replace(True,1)
data.defects=data.defects.replace(False,0)
```

In [23]:

```
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: 76, zeroes: 76
```

In [24]:

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

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75], dtype=int64),)
```

In [25]:

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

In [26]:

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

In [27]:

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

In [28]:

```
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[28]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
0	0.001570	0.000155	0.000052	0.000035	0.005486	0.036045	6.901112e-07	0.000478	0.001302
1	0.001505	0.000290	0.000069	0.000248	0.005259	0.035167	5.521753e-07	0.000392	0.001239
2	0.000321	0.000067	0.000026	0.000052	0.001486	0.013149	6.352961e-09	0.000048	0.000173
3	0.000471	0.000110	0.000031	0.000062	0.002727	0.018869	8.811351e-08	0.000233	0.000357
4	0.001338	0.000199	0.000018	0.000145	0.005317	0.034683	5.425127e-07	0.000520	0.001205

5 rows × 21 columns

In [29]:

```
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()
```

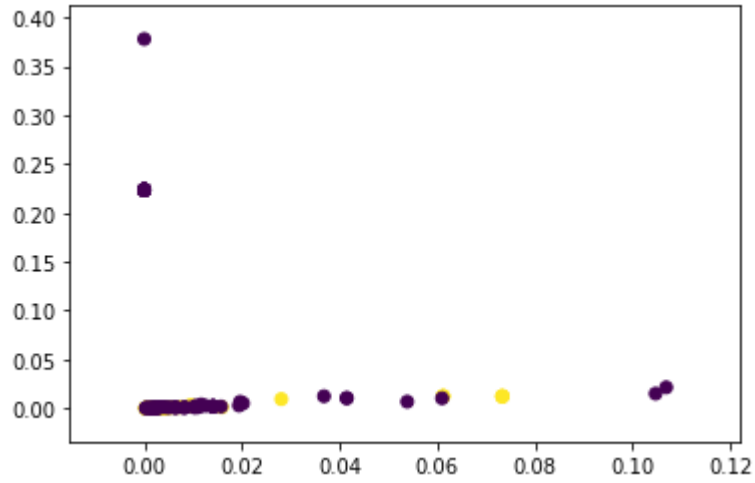
Out[29]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
0	0.001570	0.000155	0.000052	0.000035	0.005486	0.036045	6.901112e-07	0.000478	0.001302
1	0.001505	0.000290	0.000069	0.000248	0.005259	0.035167	5.521753e-07	0.000392	0.001239
2	0.000321	0.000067	0.000026	0.000052	0.001486	0.013149	6.352961e-09	0.000048	0.000173
3	0.000471	0.000110	0.000031	0.000062	0.002727	0.018869	8.811351e-08	0.000233	0.000357
4	0.001338	0.000199	0.000018	0.000145	0.005317	0.034683	5.425127e-07	0.000520	0.001205

5 rows × 22 columns

In [30]:

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



In [31]:

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

In [32]:

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

In [33]:

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

```
(114, 21) (114, 1) (38, 21) (38, 1)
```

In [34]:

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

In [35]:

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

```
ones: 59, zeroes: 55
```

In [36]:

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

```
[4.72280452e-03 5.90350566e-04 5.90350566e-04 5.90350566e-04
 2.89271777e-02 1.40527049e-01 8.26490792e-05 4.15016448e-03
 1.99951737e-02 9.87597461e-01 4.72280452e-05 5.48671816e-02
 4.72280452e-03 2.36140226e-03 0.00000000e+00 3.54210339e-03
 6.49385622e-03 1.06263102e-02 1.53491147e-02 1.35780630e-02
 5.90350566e-04]


[4.71407299e-04 1.10141892e-04 3.08397298e-05 6.16794596e-05
 2.72711325e-03 1.88685397e-02 8.81135138e-08 2.33104301e-04
 3.56639447e-04 9.98276547e-01 6.30011623e-06 5.54597910e-02
 4.53784596e-04 1.40981622e-04 1.76227028e-05 1.71821352e-04
 1.54198649e-04 3.78888109e-04 1.58163757e-03 1.14547568e-03
 1.76227028e-04]


[2.21651913e-03 5.91071767e-04 4.43303825e-04 2.95535884e-04
 1.27080430e-02 6.76319093e-02 1.03437559e-05 2.17662178e-03
 4.59262763e-03 9.96023901e-01 2.21651913e-05 5.53346612e-02
```

BASE PREDICTIORS

1- SVM

In [37]:

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

In [38]:

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

Out[38]:

```
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 [39]:

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

Out[39]:

```
57.89473684210527
```

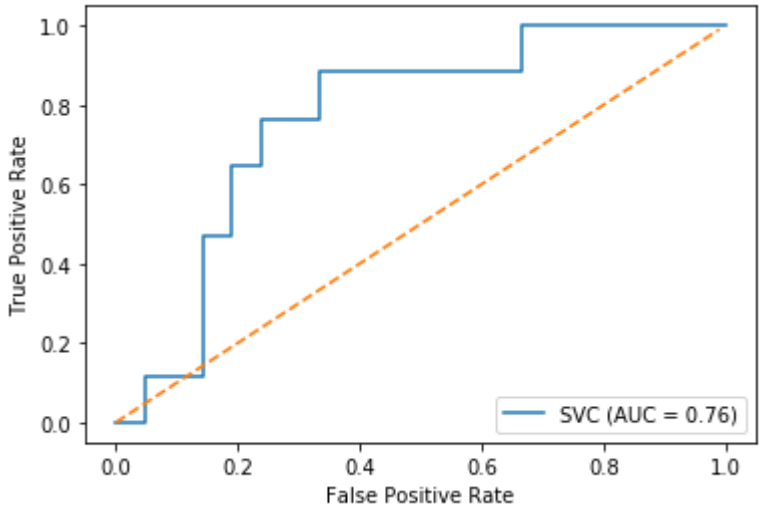
In [40]:

```
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:  57.89473684210527
Probability of detection of defect(Recall, pd):  1.0
Probability of false alarm(pf):  0.0
Probability of correct detection(Precision):  0.5151515151515151


F1-score or FM:  0.6799999999999999
AUC value:  0.6190476190476191


   P    N
[[ 5 16]
 [ 0 17]]


              precision    recall  f1-score   support

         0.0       1.00      0.24      0.38        21
         1.0       0.52      1.00      0.68        17

    accuracy                           0.58        38
   macro avg       0.76      0.62      0.53        38
weighted avg       0.78      0.58      0.52        38
```

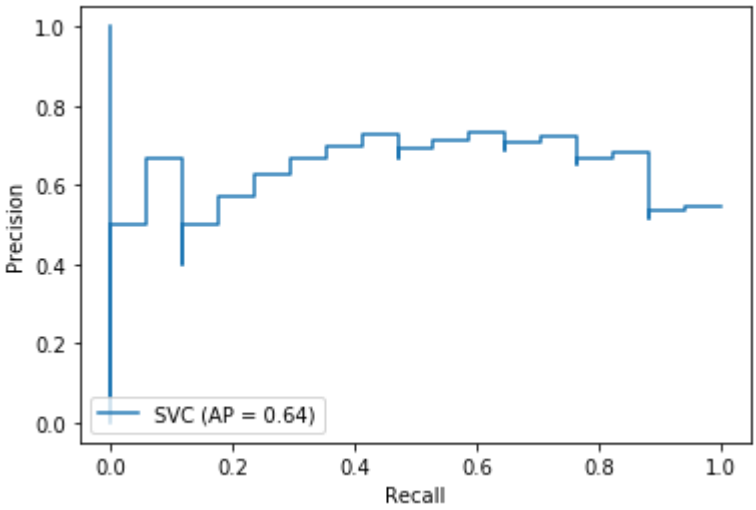
In [41]:

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

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



In [42]:

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



2- KNN

In [116]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=11)
```

In [117]:

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

Out[117]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                     weights='uniform')
```

In [118]:

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

Out[118]:

```
71.05263157894737
```

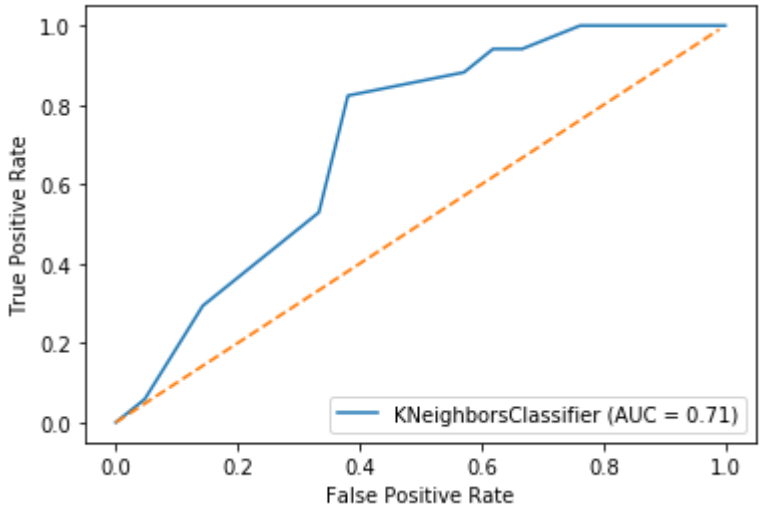
In [119]:

```
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:  71.05263157894737
Probability of detection of defect(Recall, pd):  0.8235294117647058
Probability of false alarm(pf):  0.1875
Probability of correct detection(Precision):  0.6363636363636364


F1-score or FM:  0.717948717948718
AUC value:  0.7212885154061625


[[13  8]
 [ 3 14]]


              precision    recall  f1-score   support

         0.0       0.81      0.62      0.70        21
         1.0       0.64      0.82      0.72        17

    accuracy                           0.71        38
   macro avg       0.72      0.72      0.71        38
weighted avg       0.73      0.71      0.71        38
```

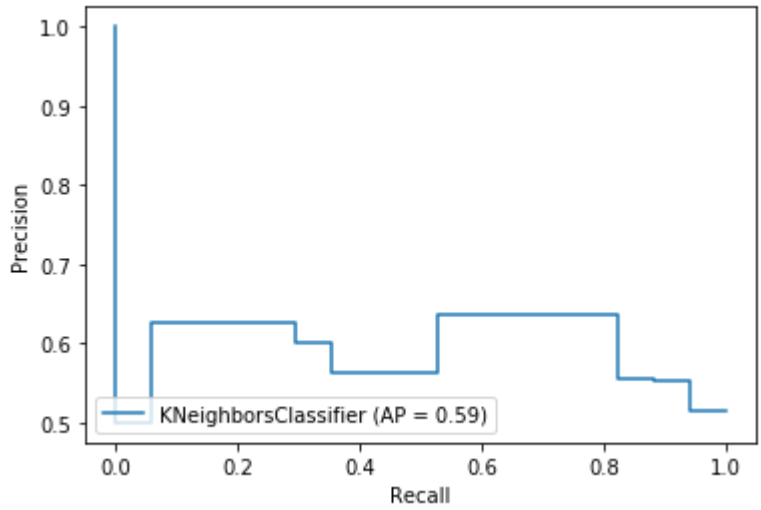
In [120]:

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

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



In [121]:

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



In [122]:

```
# 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)
```

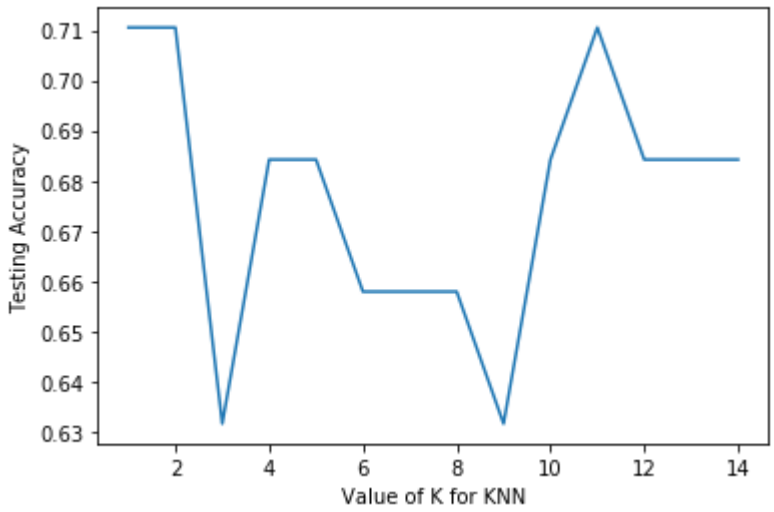
```
[0.7105263157894737, 0.7105263157894737, 0.631578947368421, 0.68421052631578
95, 0.6842105263157895, 0.6578947368421053, 0.6578947368421053, 0.6578947368
421053, 0.631578947368421, 0.6842105263157895, 0.7105263157894737, 0.6842105
263157895, 0.6842105263157895, 0.6842105263157895]
```

In [123]:

```
# 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[123]:

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



3- NAIVE BAYES

In [51]:

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

In [52]:

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

Out[52]:

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

In [53]:

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

Out[53]:

```
73.68421052631578
```

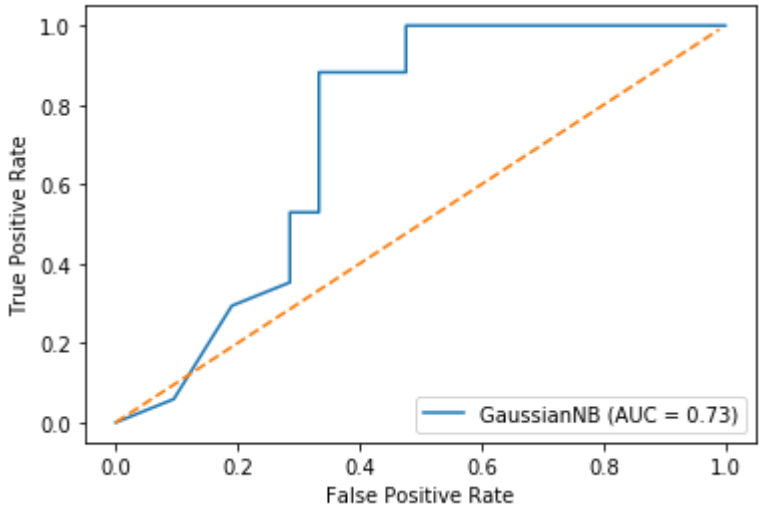
In [54]:

```
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:  73.68421052631578
Probability of detection of defect(Recall, pd):  1.0
Probability of false alarm(pf):  0.0
Probability of correct detection(Precision):  0.6296296296296297


F1-score or FM:  0.7727272727272727
AUC value:  0.7619047619047619


[[11 10]
 [ 0 17]]


              precision    recall  f1-score   support

         0.0       1.00      0.52      0.69        21
         1.0       0.63      1.00      0.77        17

    accuracy                           0.74        38
   macro avg       0.81      0.76      0.73        38
weighted avg       0.83      0.74      0.73        38
```

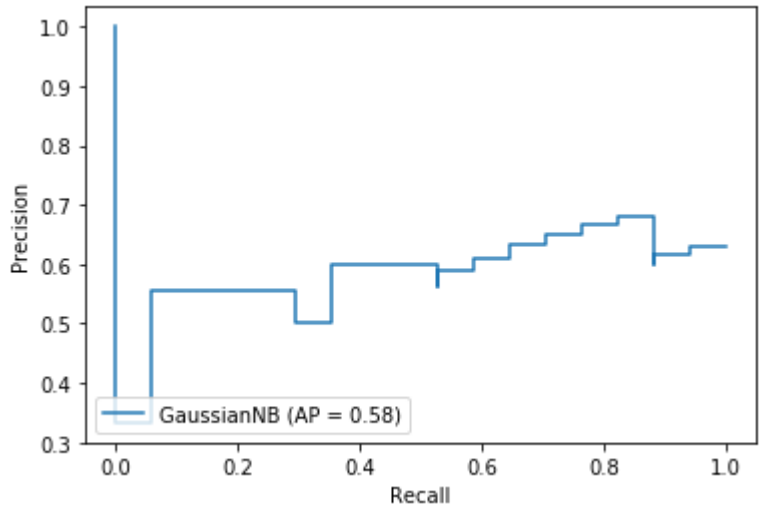
In [55]:

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

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



In [56]:

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



In [57]:

```
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 38 points : 10
```

4- LOGISTIC REGRESSION

In [58]:

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

In [59]:

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

Out[59]:

```
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 [60]:

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

Out[60]:

```
60.526315789473685
```

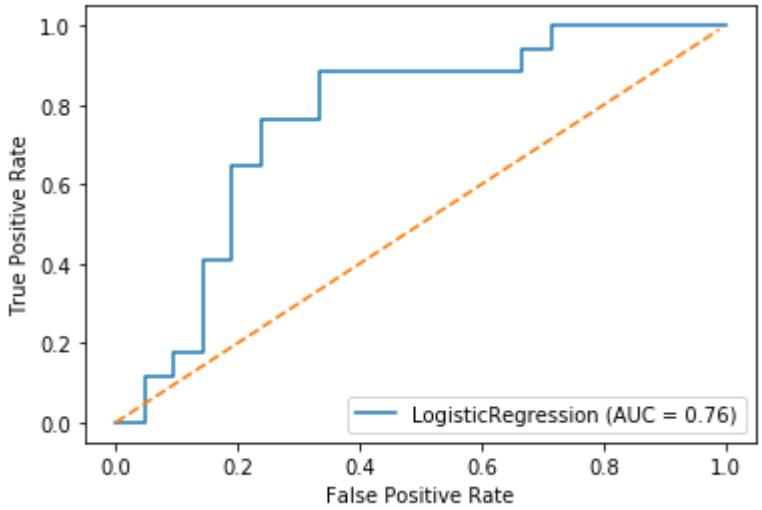
In [61]:

```
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:  60.526315789473685
Probability of detection of defect(Recall, pd):  0.9411764705882353
Probability of false alarm(pf):  0.125
Probability of correct detection(Precision):  0.5333333333333333


F1-score or FM:  0.6808510638297872
AUC value:  0.6372549019607843


[[ 7 14]
 [ 1 16]]


              precision    recall  f1-score   support

         0.0       0.88      0.33      0.48        21
         1.0       0.53      0.94      0.68        17

    accuracy                           0.61        38
   macro avg       0.70      0.64      0.58        38
weighted avg       0.72      0.61      0.57        38
```

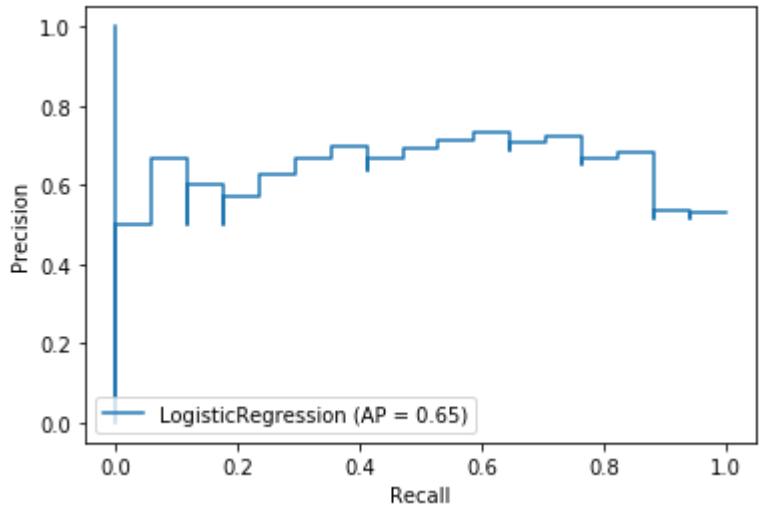
In [62]:

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

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



In [63]:

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



5- MLP

In [64]:

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

In [65]:

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

Out[65]:

```
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 [66]:

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

Out[66]:

```
73.68421052631578
```

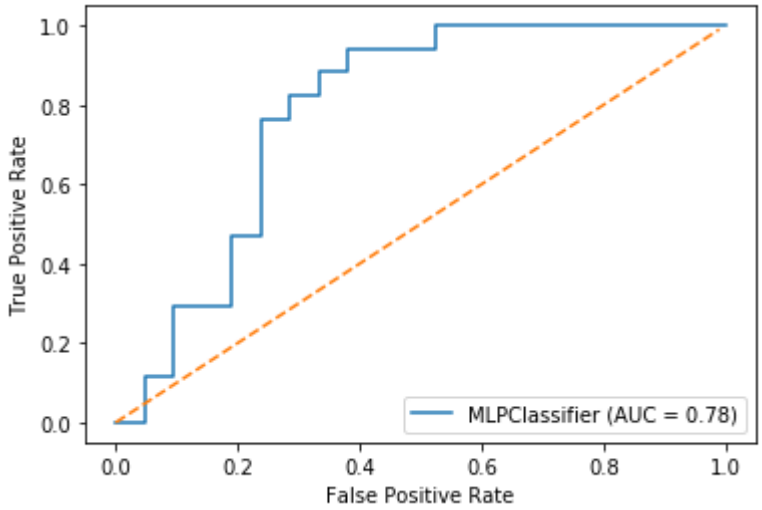
In [67]:

```
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:  73.68421052631578
Probability of detection of defect(Recall, pd):  0.8823529411764706
Probability of false alarm(pf):  0.13333333333333333
Probability of correct detection(Precision):  0.6521739130434783


F1-score or FM:  0.75
AUC value:  0.7507002801120448


[[13  8]
 [ 2 15]]


              precision    recall  f1-score   support

         0.0       0.87      0.62      0.72        21
         1.0       0.65      0.88      0.75        17

    accuracy                           0.74        38
   macro avg       0.76      0.75      0.74        38
weighted avg       0.77      0.74      0.73        38
```

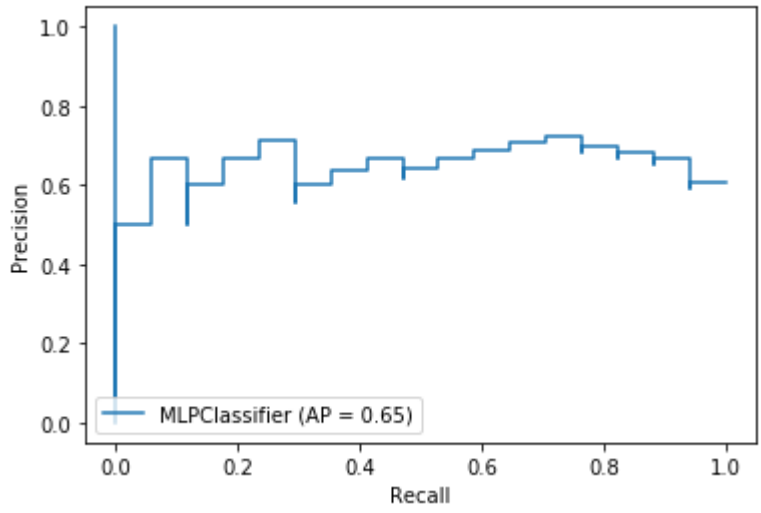
In [68]:

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

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



In [69]:

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



6- DECISION TREE

In [70]:

```
from sklearn import tree
```

In [71]:

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

Out[71]:

```
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 [72]:

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

Out[72]:

```
81.57894736842105
```

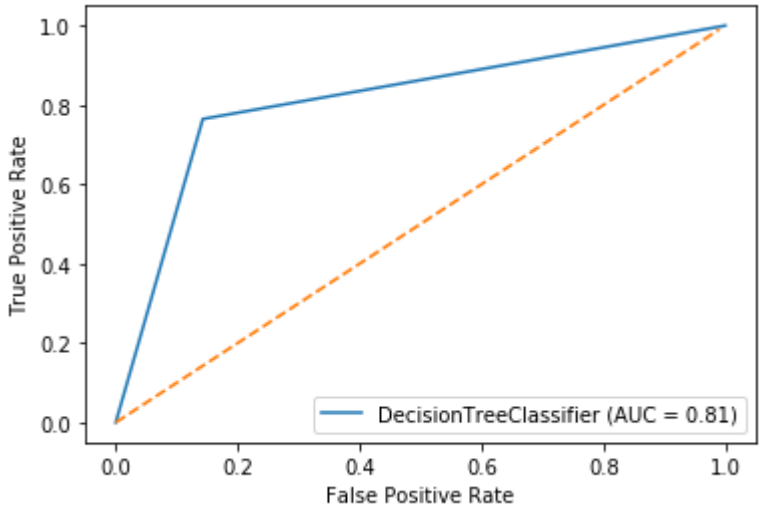
In [73]:

```
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:  81.57894736842105
Probability of detection of defect(Recall, pd):  0.7647058823529411
Probability of false alarm(pf):  0.18181818181818182
Probability of correct detection(Precision):  0.8125


F1-score or FM:  0.787878787878788
AUC value:  0.8109243697478993


[[18  3]
 [ 4 13]]


              precision    recall  f1-score   support

         0.0       0.82      0.86      0.84        21
         1.0       0.81      0.76      0.79        17

    accuracy                           0.82        38
   macro avg       0.82      0.81      0.81        38
weighted avg       0.82      0.82      0.82        38
```

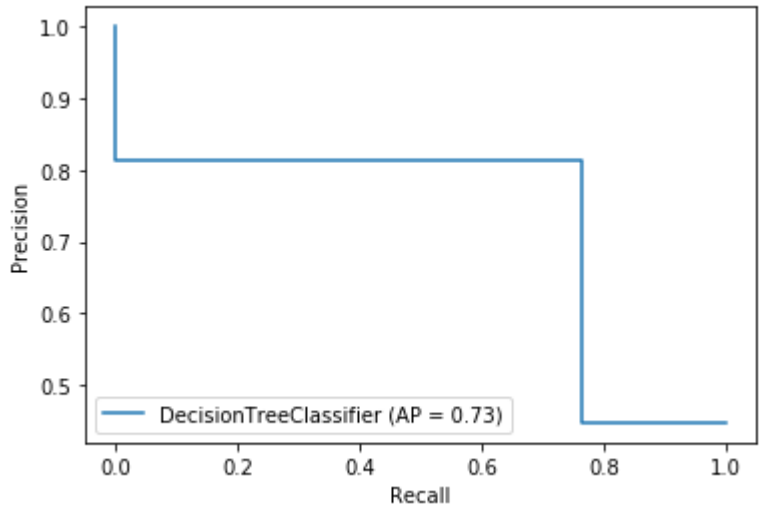
In [74]:

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

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



In [75]:

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



ENSEMBLE PREDICTORS

1- ADABOOST

In [76]:

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

In [77]:

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

Out[77]:

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

In [78]:

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

Out[78]:

```
71.05263157894737
```

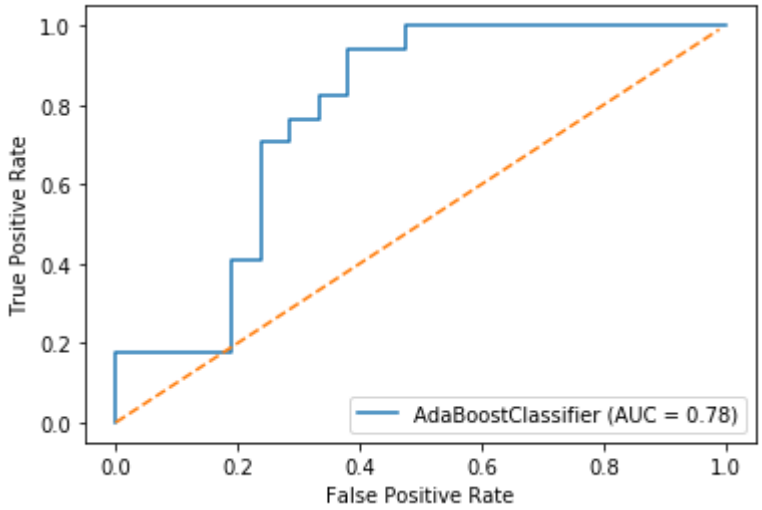
In [79]:

```
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:  71.05263157894737
Probability of detection of defect(Recall, pd):  0.7058823529411765
Probability of false alarm(pf):  0.25
Probability of correct detection(Precision):  0.6666666666666666


F1-score or FM:  0.6857142857142857
AUC value:  0.7100840336134454


[[15  6]
 [ 5 12]]


              precision    recall  f1-score   support

         0.0       0.75      0.71      0.73        21
         1.0       0.67      0.71      0.69        17

    accuracy                           0.71        38
   macro avg       0.71      0.71      0.71        38
weighted avg       0.71      0.71      0.71        38
```

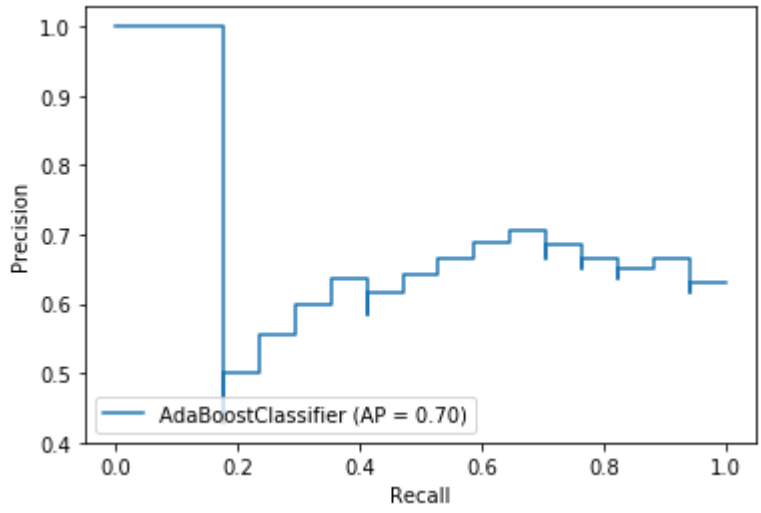
In [80]:

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

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



In [81]:

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



2- BAGGING

In [82]:

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

In [83]:

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

Out[83]:

```
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 [84]:

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

Out[84]:

```
78.94736842105263
```

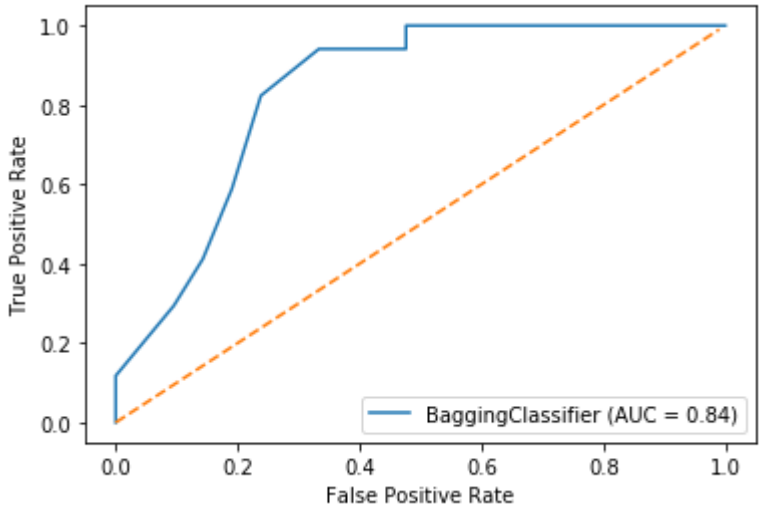
In [85]:

```
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:  78.94736842105263
Probability of detection of defect(Recall, pd):  0.8235294117647058
Probability of false alarm(pf):  0.15789473684210525
Probability of correct detection(Precision):  0.7368421052631579


F1-score or FM:  0.7777777777777778
AUC value:  0.7927170868347339


[[16  5]
 [ 3 14]]


              precision    recall  f1-score   support

         0.0       0.84      0.76      0.80        21
         1.0       0.74      0.82      0.78        17

    accuracy                           0.79        38
   macro avg       0.79      0.79      0.79        38
weighted avg       0.80      0.79      0.79        38
```

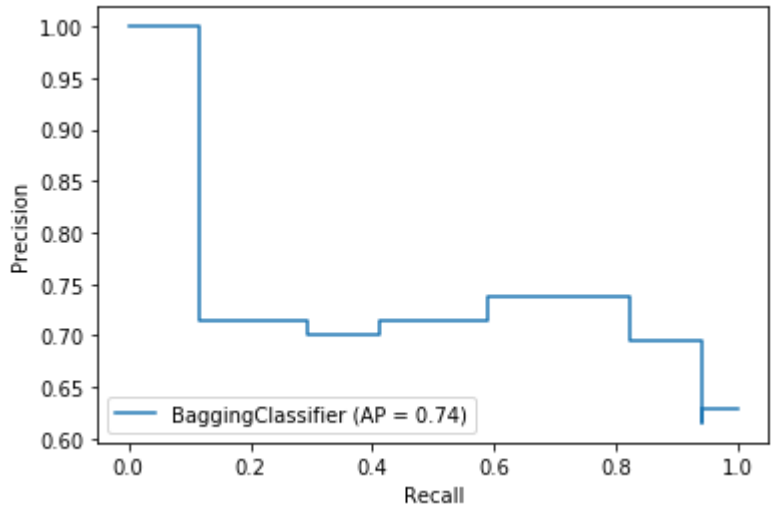
In [86]:

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

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



In [87]:

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



3- Extra_Tree_Classifier

In [88]:

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

In [89]:

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

Out[89]:

```
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 [90]:

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

Out[90]:

```
76.31578947368422
```

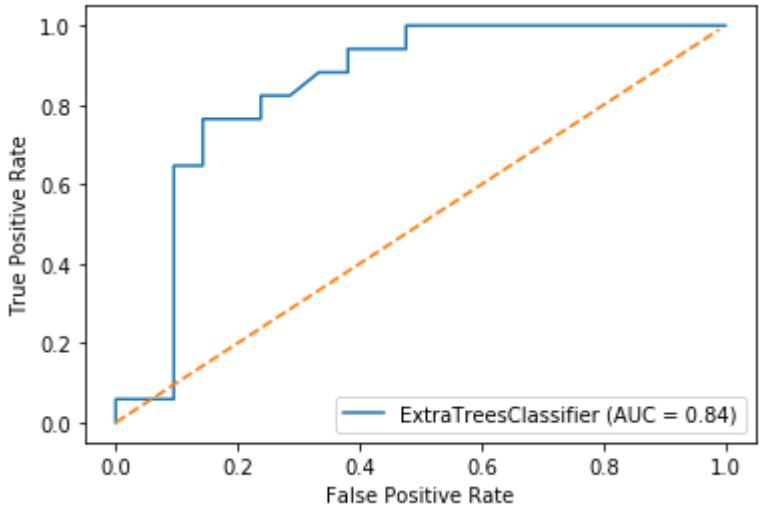
In [91]:

```
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:  76.31578947368422
Probability of detection of defect(Recall, pd):  0.6470588235294118
Probability of false alarm(pf):  0.25
Probability of correct detection(Precision):  0.7857142857142857


F1-score or FM:  0.7096774193548386
AUC value:  0.7521008403361344


[[18  3]
 [ 6 11]]


              precision    recall  f1-score   support

         0.0       0.75      0.86      0.80        21
         1.0       0.79      0.65      0.71        17

    accuracy                           0.76        38
   macro avg       0.77      0.75      0.75        38
weighted avg       0.77      0.76      0.76        38
```

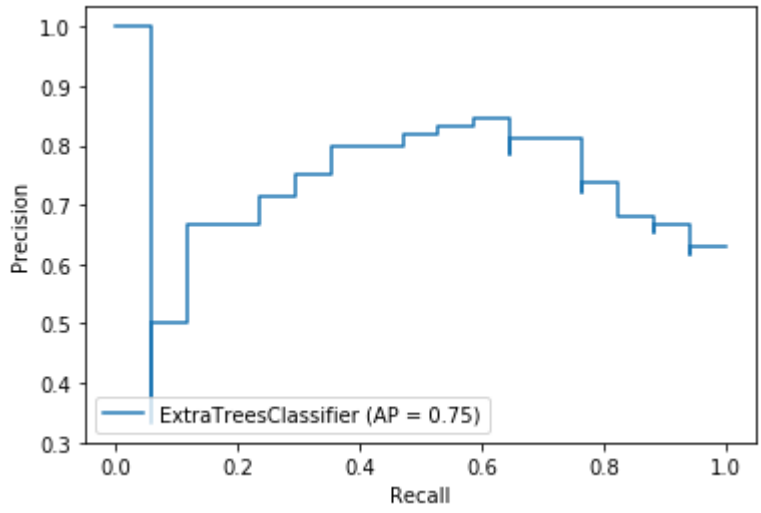
In [92]:

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

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



In [93]:

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



4- Gradient_Boosting_Classifier

In [94]:

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

In [95]:

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

Out[95]:

```
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 [96]:

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

Out[96]:

```
71.05263157894737
```

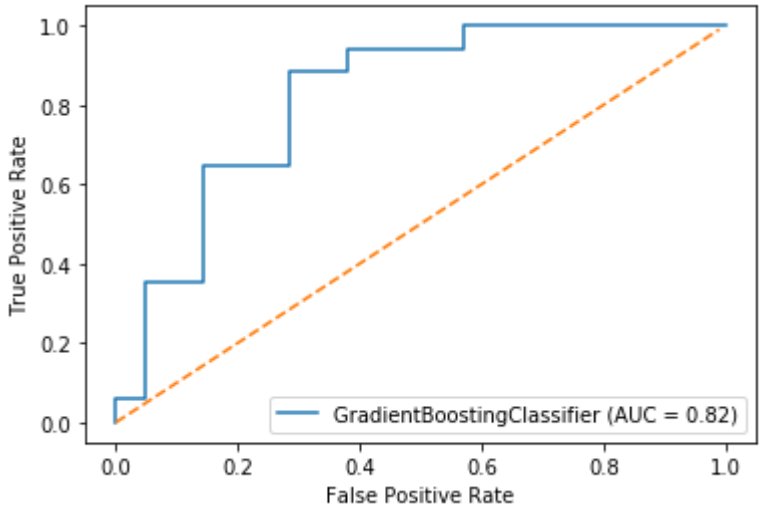
In [97]:

```
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:  71.05263157894737
Probability of detection of defect(Recall, pd):  0.7058823529411765
Probability of false alarm(pf):  0.25
Probability of correct detection(Precision):  0.6666666666666666


F1-score or FM:  0.6857142857142857
AUC value:  0.7100840336134454


[[15  6]
 [ 5 12]]


              precision    recall  f1-score   support

         0.0       0.75      0.71      0.73        21
         1.0       0.67      0.71      0.69        17

    accuracy                           0.71        38
   macro avg       0.71      0.71      0.71        38
weighted avg       0.71      0.71      0.71        38
```

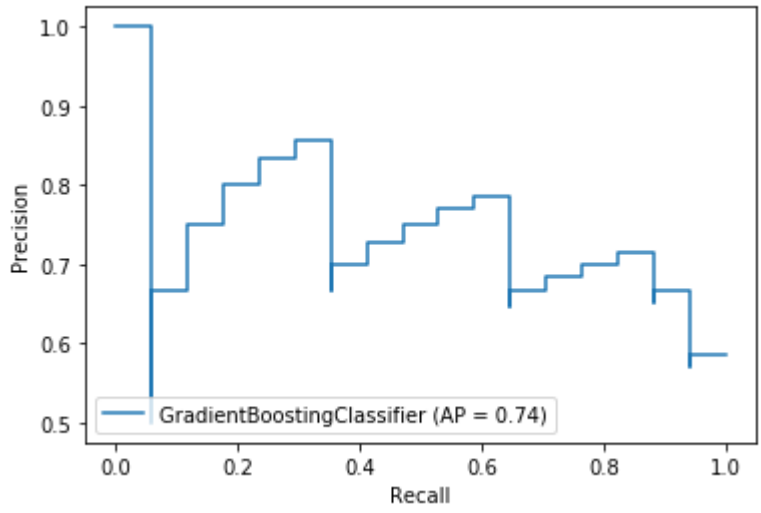
In [98]:

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

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



In [99]:

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



5- Random_Forest_Classifier

In [100]:

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

In [101]:

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

Out[101]:

```
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 [102]:

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

Out[102]:

```
78.94736842105263
```

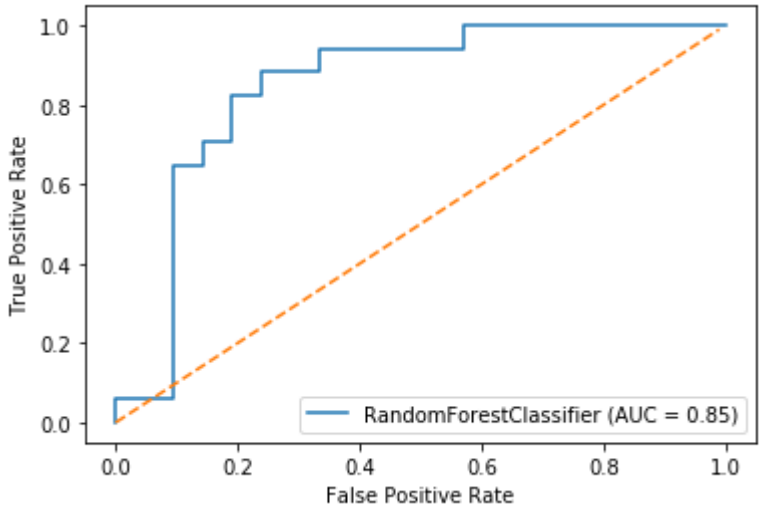
In [103]:

```
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:  78.94736842105263
Probability of detection of defect(Recall, pd):  0.8235294117647058
Probability of false alarm(pf):  0.15789473684210525
Probability of correct detection(Precision):  0.7368421052631579


F1-score or FM:  0.7777777777777778
AUC value:  0.7927170868347339


[[16  5]
 [ 3 14]]


              precision    recall  f1-score   support

         0.0       0.84      0.76      0.80        21
         1.0       0.74      0.82      0.78        17

    accuracy                           0.79        38
   macro avg       0.79      0.79      0.79        38
weighted avg       0.80      0.79      0.79        38
```

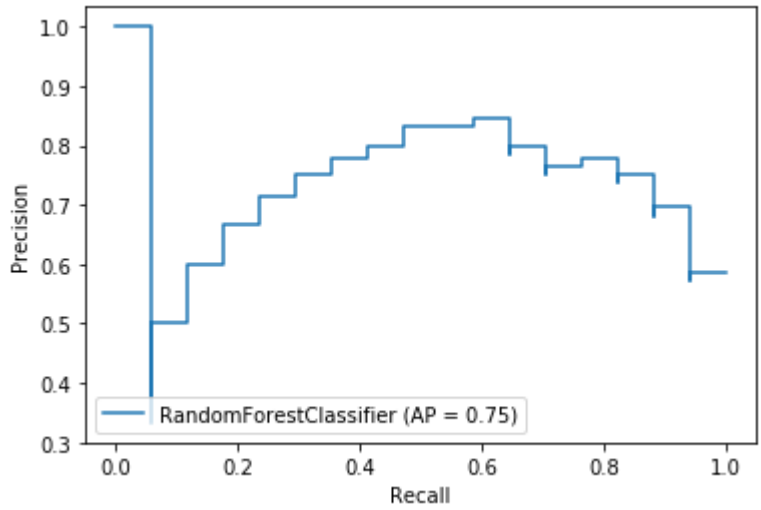
In [104]:

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

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



In [105]:

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



6- Stacking_Classifier

In [106]:

```
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 [107]:

```
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)
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[107]:

```
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 [108]:

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

Out[108]:

```
81.57894736842105
```

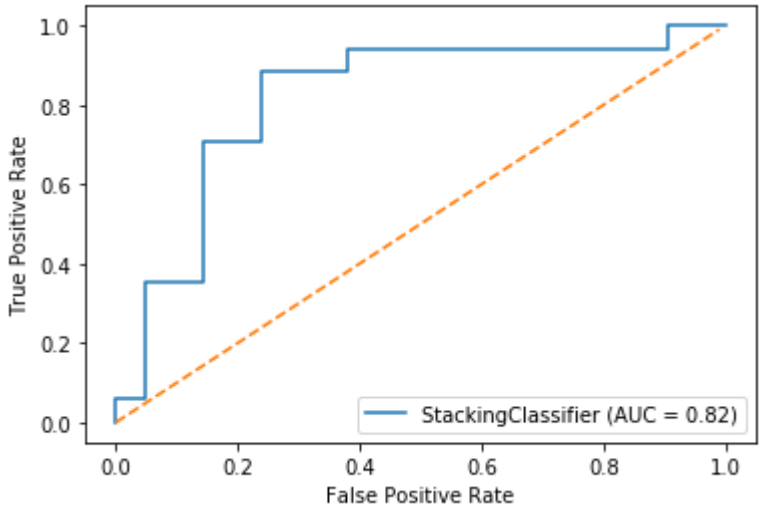
In [109]:

```
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:  81.57894736842105
Probability of detection of defect(Recall, pd):  0.8823529411764706
Probability of false alarm(pf):  0.1111111111111111
Probability of correct detection(Precision):  0.75


F1-score or FM:  0.8108108108108107
AUC value:  0.8221288515406161


[[16  5]
 [ 2 15]]


              precision    recall  f1-score   support

         0.0       0.89      0.76      0.82        21
         1.0       0.75      0.88      0.81        17

    accuracy                           0.82        38
   macro avg       0.82      0.82      0.82        38
weighted avg       0.83      0.82      0.82        38
```

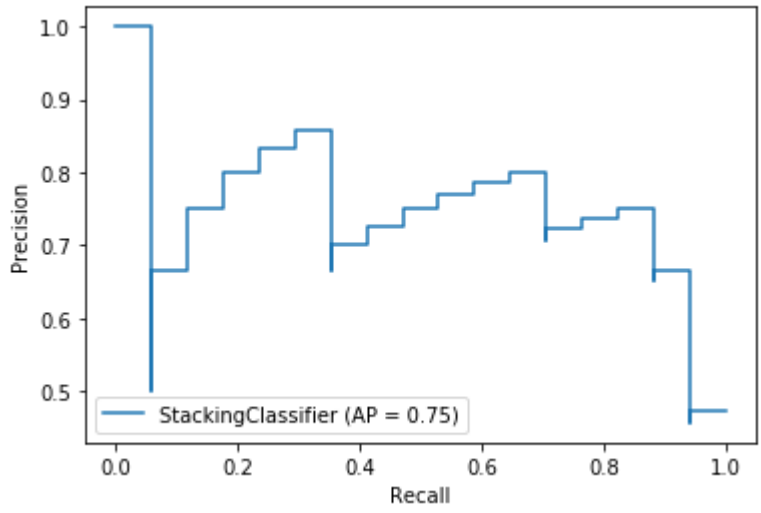
In [110]:

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

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



In [111]:

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



7- Voting_Classifier

In [112]:

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

In [113]:

```
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[113]:

```
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 [114]:

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

Out[114]:

```
73.68421052631578
```

In [115]:

```
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:  73.68421052631578
Probability of detection of defect(Recall, pd):  1.0
Probability of false alarm(pf):  0.0
Probability of correct detection(Precision):  0.6296296296296297


F1-score or FM:  0.7727272727272727
AUC value:  0.7619047619047619


[[11 10]
 [ 0 17]]


              precision    recall  f1-score   support

         0.0       1.00      0.52      0.69        21
         1.0       0.63      1.00      0.77        17

    accuracy                           0.74        38
   macro avg       0.81      0.76      0.73        38
weighted avg       0.83      0.74      0.73        38
```

In [ ]:

In [ ]: