In [102]:

```
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 [103]:

```
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/promise3_useful.txt",names=header
data.head()
```

Out[103]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1
2	83.0	11.0	1.0	11.0	171.0	927.89	0.04	23.04	40.27	21378.61	...	65	10
3	46.0	8.0	6.0	8.0	141.0	769.78	0.07	14.86	51.81	11436.73	...	37	2
4	25.0	3.0	1.0	3.0	58.0	254.75	0.11	9.35	27.25	2381.95	...	21	0

5 rows × 22 columns

In [104]:

```
data=pd.DataFrame(data)

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

In [105]:

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

```
(array([  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,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
        92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104,
       105, 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, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
       222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
       235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247,
```

In [197]:

```
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: 326, zeroes: 326
```

In [106]:

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

In [107]:

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

In [108]:

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

In [109]:

```
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[109]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	
0	0.165213	0.210271	0.210271	0.210271	0.195252	0.195252	0.195252	0.195252	0.195252	0
1	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0
2	0.003873	0.000513	0.000047	0.000513	0.007978	0.043292	0.000002	0.001075	0.001879	0
3	0.004006	0.000697	0.000523	0.000697	0.012280	0.067043	0.000006	0.001294	0.004512	0
4	0.010413	0.001250	0.000417	0.001250	0.024159	0.106113	0.000046	0.003895	0.011351	0

5 rows × 21 columns

In [110]:

```
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()
```

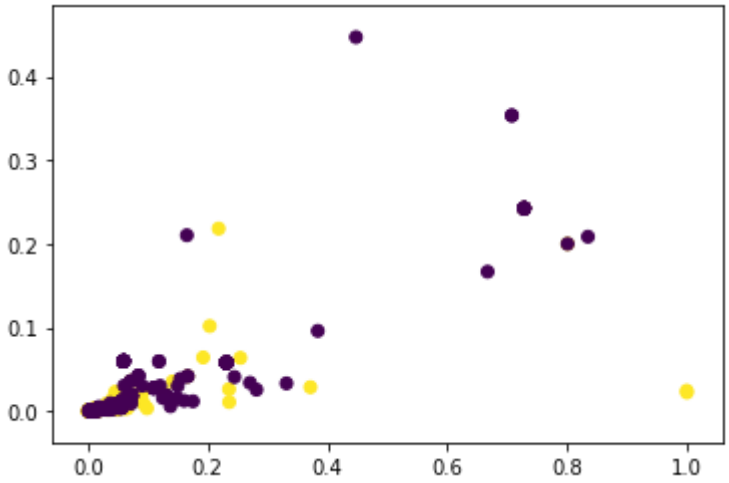
Out[110]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	
0	0.165213	0.210271	0.210271	0.210271	0.195252	0.195252	0.195252	0.195252	0.195252	0
1	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	0
2	0.003873	0.000513	0.000047	0.000513	0.007978	0.043292	0.000002	0.001075	0.001879	0
3	0.004006	0.000697	0.000523	0.000697	0.012280	0.067043	0.000006	0.001294	0.004512	0
4	0.010413	0.001250	0.000417	0.001250	0.024159	0.106113	0.000046	0.003895	0.011351	0

5 rows × 22 columns

In [111]:

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



In [112]:

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

In [113]:

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

In [114]:

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

```
(489, 21) (489, 1) (163, 21) (163, 1)
```

In [115]:

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

In [116]:

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

```
ones: 245, zeroes: 244
```

In [117]:

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

```
[1.16909416e-03 2.01567958e-04 1.69317085e-04 1.45128930e-04
 3.55565878e-03 2.26674456e-02 1.61254366e-07 3.55082115e-04
 5.14723937e-04 9.98192747e-01 7.57895522e-06 5.54551347e-02
 6.53080183e-04 2.41881549e-05 4.67637662e-04 0.00000000e+00
 2.25756113e-04 4.43449507e-04 2.16080851e-03 1.39485027e-03
 3.95073197e-04]


[1.18751645e-02 1.31946272e-03 6.59731360e-04 6.59731360e-04
 2.57295230e-02 1.02918092e-01 6.59731360e-05 6.35981031e-03
 1.06744534e-02 9.92427287e-01 3.29865680e-05 5.51337497e-02
 9.23623904e-03 0.00000000e+00 1.31946272e-03 0.00000000e+00
 5.93758224e-03 4.61811952e-03 1.58335526e-02 9.89597040e-03
 1.97919408e-03]


[4.01685063e-02 2.67790042e-03 2.67790042e-03 2.67790042e-03
 6.96254109e-02 2.72021125e-01 7.76591121e-04 9.37265146e-03
 7.77126701e-02 9.52073936e-01 8.03370126e-05 5.28885333e-02
```

BASE PREDICTIORS

1- SVM

In [118]:

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

In [119]:

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

Out[119]:

```
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 [120]:

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

Out[120]:

```
64.41717791411043
```

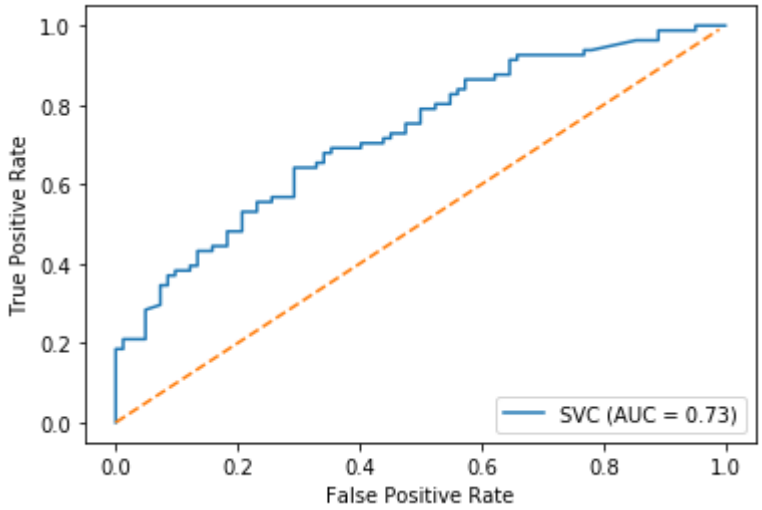
In [121]:

```
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:  64.41717791411043
Probability of detection of defect(Recall, pd):  0.7901234567901234
Probability of false alarm(pf):  0.29310344827586204
Probability of correct detection(Precision):  0.6095238095238096


F1-score or FM:  0.6881720430107527
AUC value:  0.6450617283950617


   P    N
[[41 41]
 [17 64]]


              precision    recall  f1-score   support

         0.0       0.71      0.50      0.59        82
         1.0       0.61      0.79      0.69        81

    accuracy                           0.64       163
   macro avg       0.66      0.65      0.64       163
weighted avg       0.66      0.64      0.64       163
```

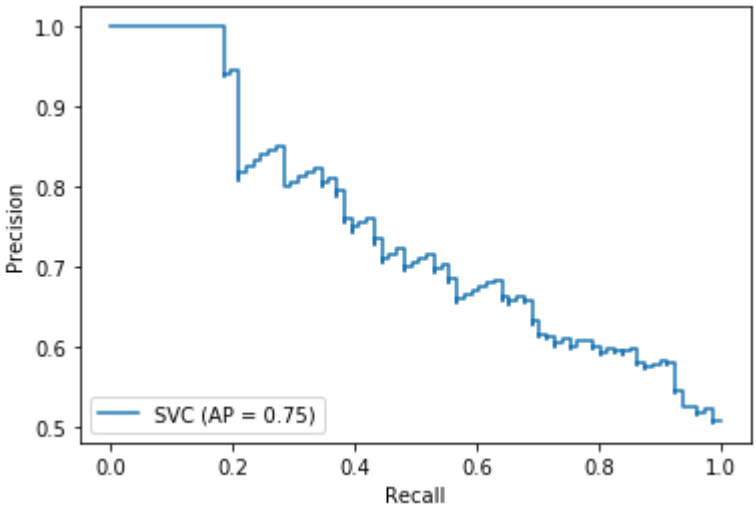
In [122]:

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

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



In [123]:

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



2- KNN

In [124]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=9)
```

In [125]:

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

Out[125]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=9, p=2,
                     weights='uniform')
```

In [126]:

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

Out[126]:

```
60.73619631901841
```

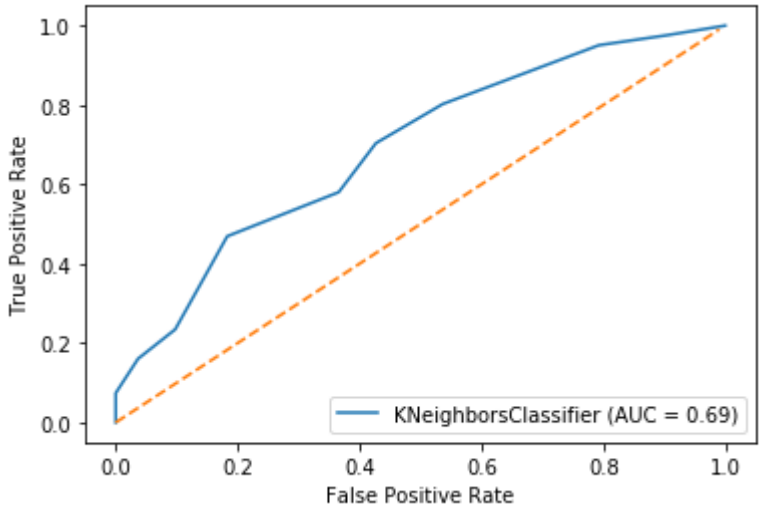
In [127]:

```
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:  60.73619631901841
Probability of detection of defect(Recall, pd):  0.5802469135802469
Probability of false alarm(pf):  0.3953488372093023
Probability of correct detection(Precision):  0.6103896103896104


F1-score or FM:  0.5949367088607596
AUC value:  0.6071966275218308


[[52 30]
 [34 47]]


              precision    recall  f1-score   support

         0.0       0.60      0.63      0.62        82
         1.0       0.61      0.58      0.59        81

    accuracy                           0.61       163
   macro avg       0.61      0.61      0.61       163
weighted avg       0.61      0.61      0.61       163
```

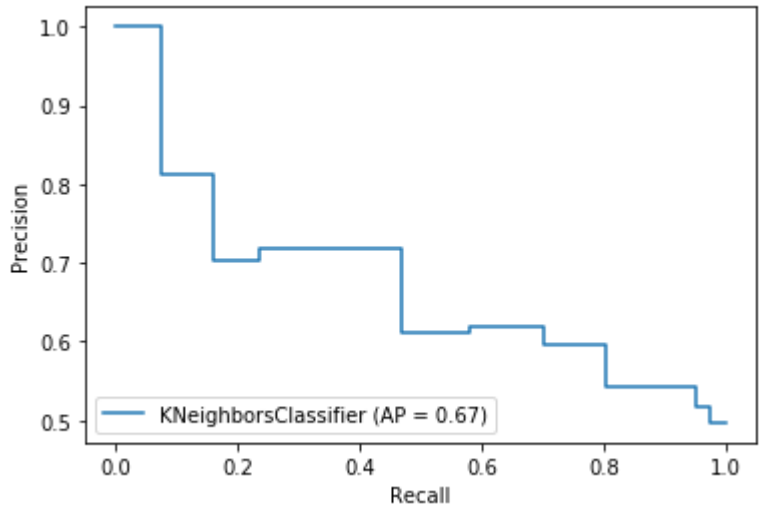
In [128]:

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

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



In [129]:

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



In [130]:

```
# 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)
```

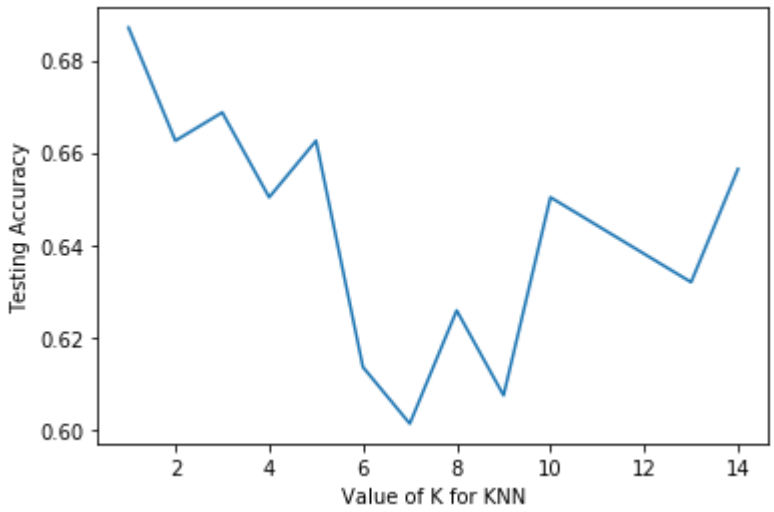
```
[0.6871165644171779, 0.6625766871165644, 0.6687116564417178, 0.6503067484662
577, 0.6625766871165644, 0.6134969325153374, 0.6012269938650306, 0.625766871
1656442, 0.6073619631901841, 0.6503067484662577, 0.6441717791411042, 0.63803
68098159509, 0.6319018404907976, 0.656441717791411]
```

In [131]:

```
# 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[131]:

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



3- NAIVE BAYES

In [132]:

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

In [133]:

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

Out[133]:

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

In [134]:

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

Out[134]:

```
58.895705521472394
```

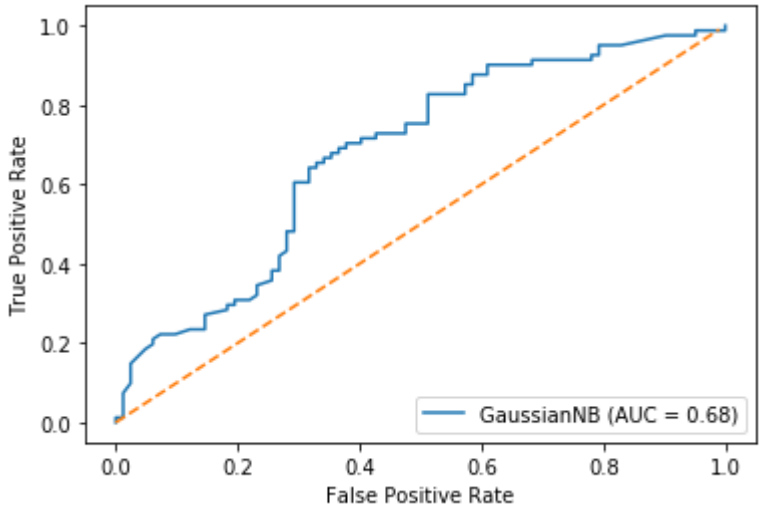
In [135]:

```
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:  58.895705521472394
Probability of detection of defect(Recall, pd):  0.9135802469135802
Probability of false alarm(pf):  0.2413793103448276
Probability of correct detection(Precision):  0.5522388059701493


F1-score or FM:  0.6883720930232559
AUC value:  0.5909364649202047


[[22 60]
 [ 7 74]]


              precision    recall  f1-score   support

         0.0       0.76      0.27      0.40        82
         1.0       0.55      0.91      0.69        81

    accuracy                           0.59       163
   macro avg       0.66      0.59      0.54       163
weighted avg       0.66      0.59      0.54       163
```

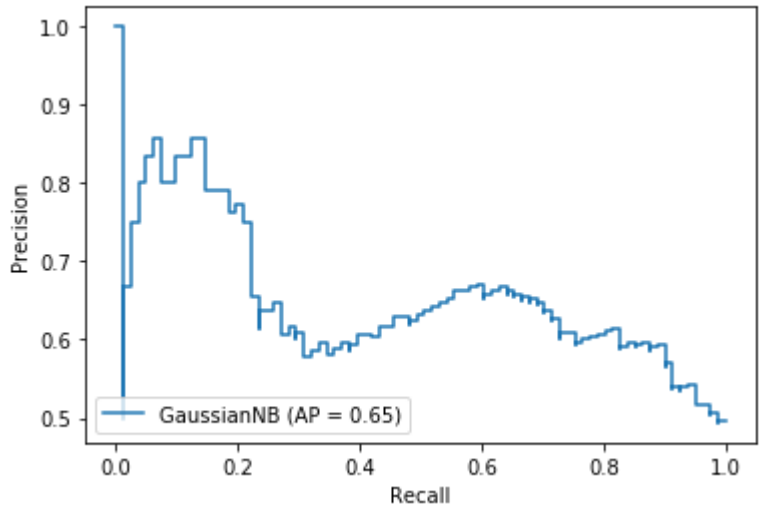
In [136]:

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

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



In [137]:

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



In [138]:

```
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 163 points : 67
```

4- LOGISTIC REGRESSION

In [139]:

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

In [140]:

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

Out[140]:

```
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 [141]:

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

Out[141]:

```
64.41717791411043
```

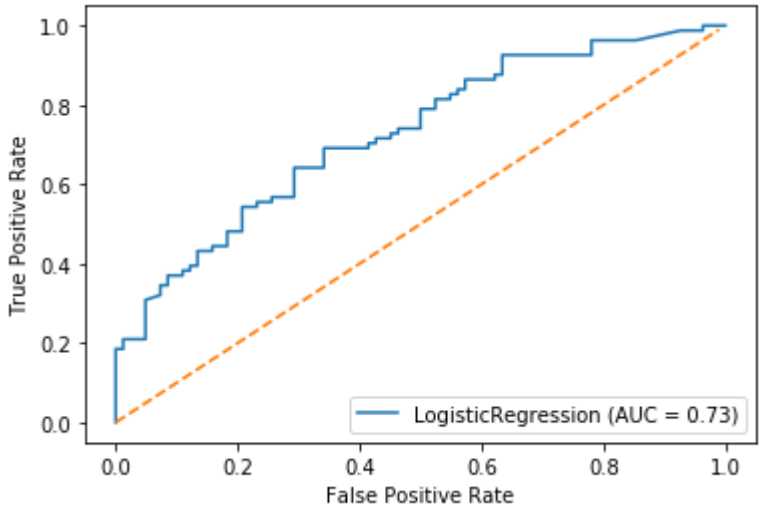
In [142]:

```
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:  64.41717791411043
Probability of detection of defect(Recall, pd):  0.7901234567901234
Probability of false alarm(pf):  0.29310344827586204
Probability of correct detection(Precision):  0.6095238095238096


F1-score or FM:  0.6881720430107527
AUC value:  0.6450617283950617


[[41 41]
 [17 64]]


              precision    recall  f1-score   support

         0.0       0.71      0.50      0.59        82
         1.0       0.61      0.79      0.69        81

    accuracy                           0.64       163
   macro avg       0.66      0.65      0.64       163
weighted avg       0.66      0.64      0.64       163
```

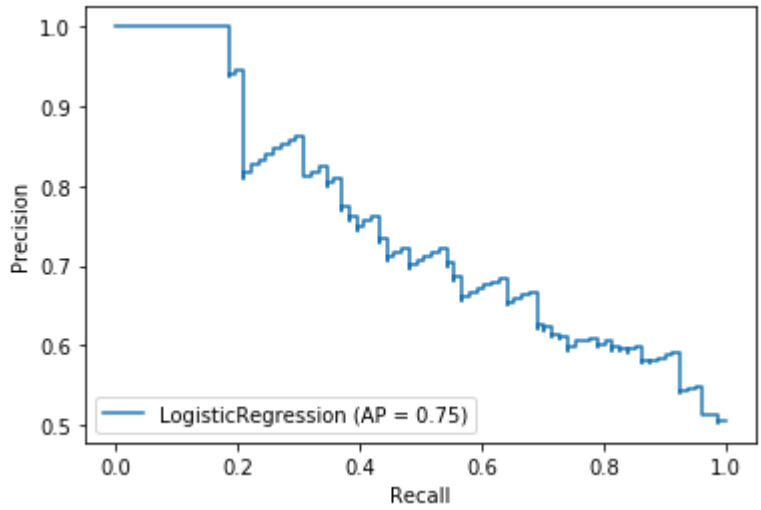
In [143]:

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

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



In [144]:

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



5- MLP

In [145]:

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

In [146]:

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

Out[146]:

```
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 [147]:

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

Out[147]:

```
64.41717791411043
```

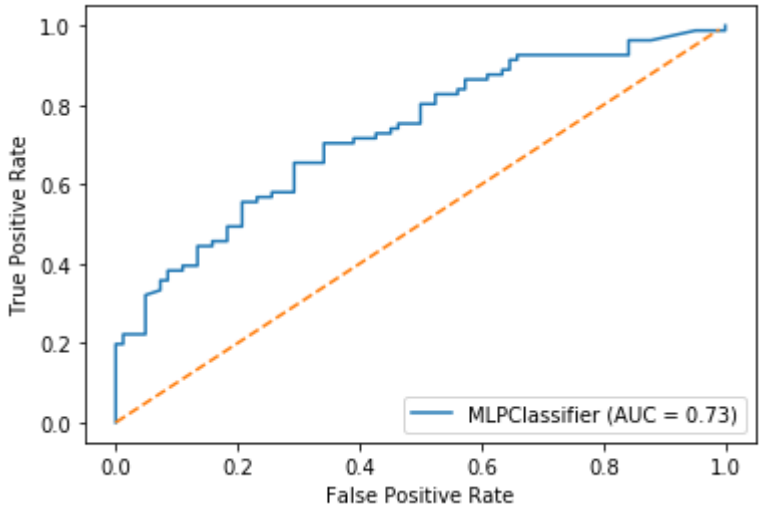
In [148]:

```
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:  64.41717791411043
Probability of detection of defect(Recall, pd):  0.7283950617283951
Probability of false alarm(pf):  0.3235294117647059
Probability of correct detection(Precision):  0.6210526315789474


F1-score or FM:  0.6704545454545455
AUC value:  0.6446853357422464


[[46 36]
 [22 59]]


              precision    recall  f1-score   support

         0.0       0.68      0.56      0.61        82
         1.0       0.62      0.73      0.67        81

    accuracy                           0.64       163
   macro avg       0.65      0.64      0.64       163
weighted avg       0.65      0.64      0.64       163
```

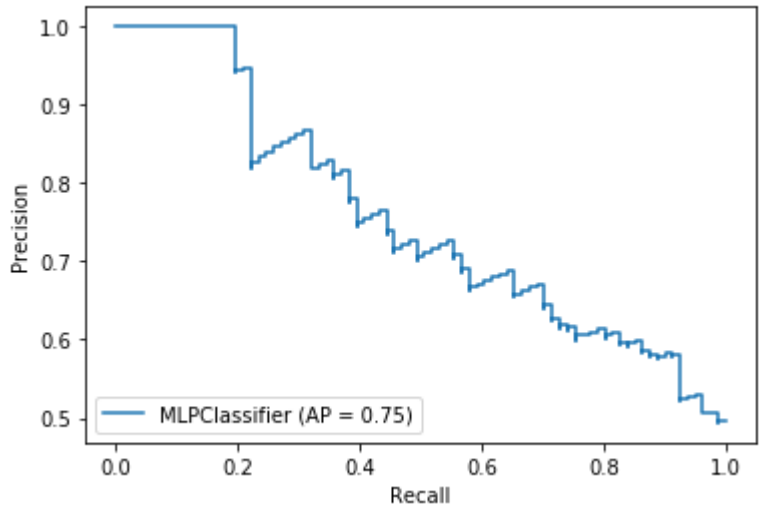
In [149]:

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

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



In [150]:

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



6- DECISION TREE

In [151]:

```
from sklearn import tree
```

In [152]:

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

Out[152]:

```
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 [153]:

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

Out[153]:

```
69.32515337423312
```

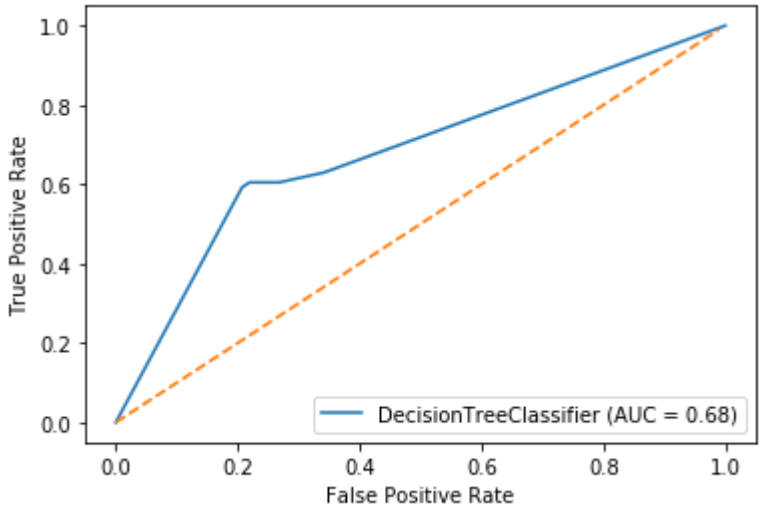
In [154]:

```
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:  69.32515337423312
Probability of detection of defect(Recall, pd):  0.5925925925925926
Probability of false alarm(pf):  0.336734693877551
Probability of correct detection(Precision):  0.7384615384615385


F1-score or FM:  0.6575342465753424
AUC value:  0.6926377597109304


[[65 17]
 [33 48]]


              precision    recall  f1-score   support

         0.0       0.66      0.79      0.72        82
         1.0       0.74      0.59      0.66        81

    accuracy                           0.69       163
   macro avg       0.70      0.69      0.69       163
weighted avg       0.70      0.69      0.69       163
```

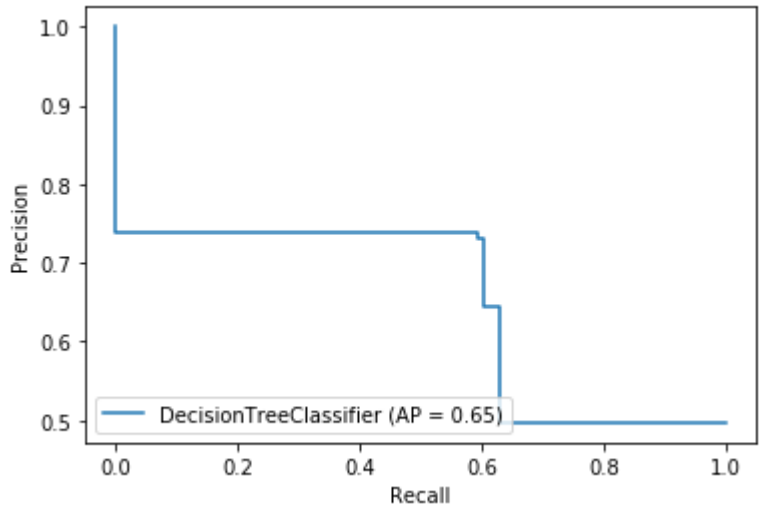
In [155]:

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

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



In [156]:

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



ENSEMBLE PREDICTORS

1- ADABOOST

In [157]:

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

In [158]:

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

Out[158]:

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

In [159]:

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

Out[159]:

```
67.48466257668711
```

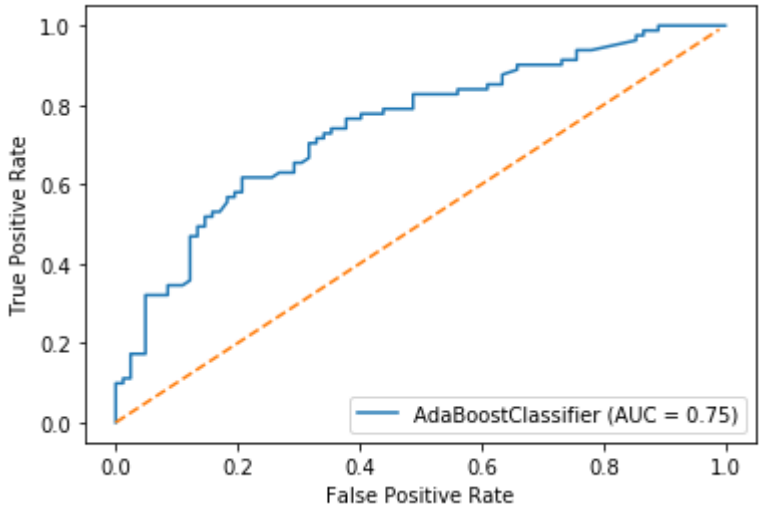
In [160]:

```
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:  67.48466257668711
Probability of detection of defect(Recall, pd):  0.6419753086419753
Probability of false alarm(pf):  0.3333333333333333
Probability of correct detection(Precision):  0.6842105263157895


F1-score or FM:  0.6624203821656051
AUC value:  0.6746461909063535


[[58 24]
 [29 52]]


              precision    recall  f1-score   support

         0.0       0.67      0.71      0.69        82
         1.0       0.68      0.64      0.66        81

    accuracy                           0.67       163
   macro avg       0.68      0.67      0.67       163
weighted avg       0.68      0.67      0.67       163
```

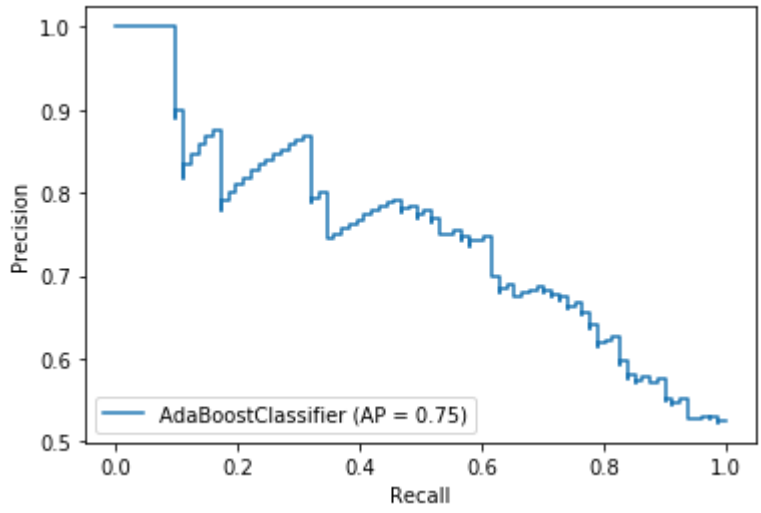
In [161]:

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

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



In [162]:

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



2- BAGGING

In [163]:

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

In [164]:

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

Out[164]:

```
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 [165]:

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

Out[165]:

```
66.87116564417178
```

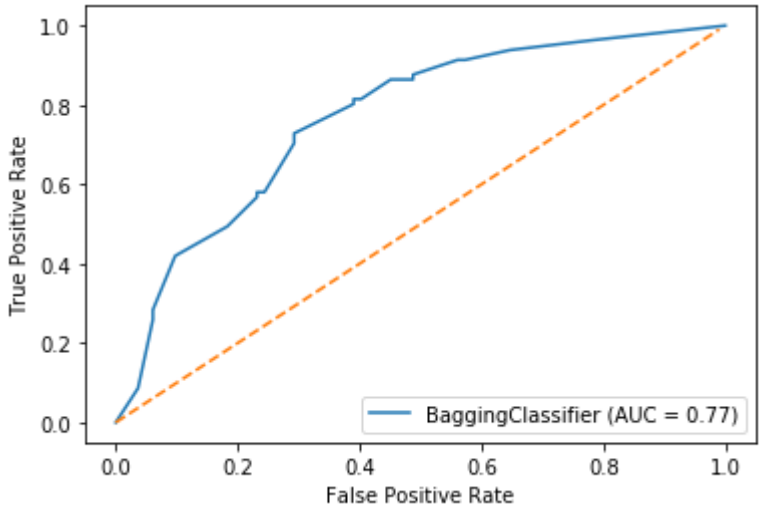
In [166]:

```
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:  66.87116564417178
Probability of detection of defect(Recall, pd):  0.5802469135802469
Probability of false alarm(pf):  0.3541666666666667
Probability of correct detection(Precision):  0.7014925373134329


F1-score or FM:  0.6351351351351352
AUC value:  0.6681722372779283


[[62 20]
 [34 47]]


              precision    recall  f1-score   support

         0.0       0.65      0.76      0.70        82
         1.0       0.70      0.58      0.64        81

    accuracy                           0.67       163
   macro avg       0.67      0.67      0.67       163
weighted avg       0.67      0.67      0.67       163
```

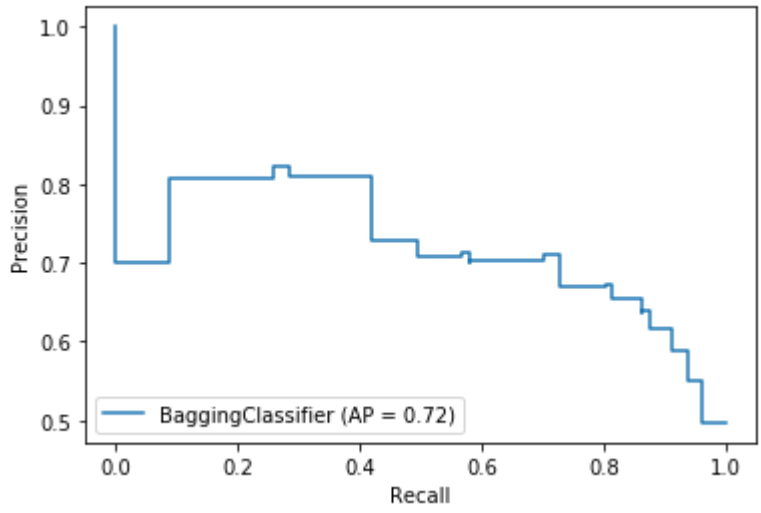
In [167]:

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

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



In [168]:

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



3- Extra_Tree_Classifier

In [169]:

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

In [170]:

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

Out[170]:

```
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 [171]:

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

Out[171]:

```
69.93865030674846
```

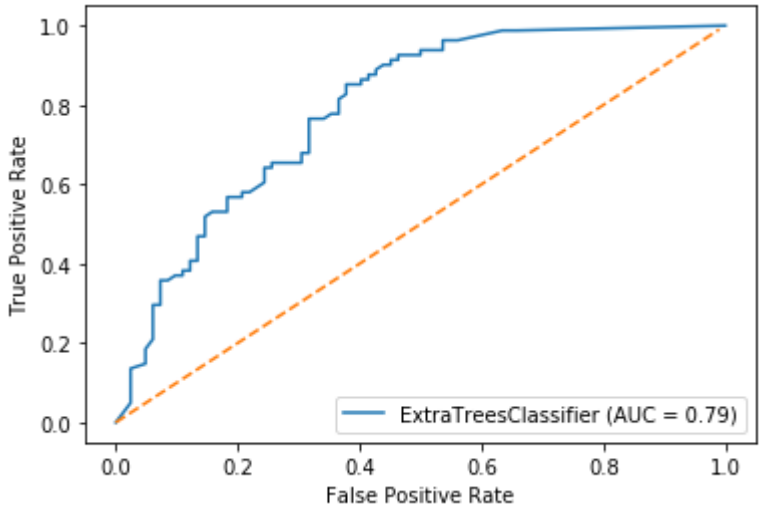
In [172]:

```
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:  69.93865030674846
Probability of detection of defect(Recall, pd):  0.654320987654321
Probability of false alarm(pf):  0.3146067415730337
Probability of correct detection(Precision):  0.7162162162162162


F1-score or FM:  0.6838709677419356
AUC value:  0.6991117133393556


[[61 21]
 [28 53]]


              precision    recall  f1-score   support

         0.0       0.69      0.74      0.71        82
         1.0       0.72      0.65      0.68        81

    accuracy                           0.70       163
   macro avg       0.70      0.70      0.70       163
weighted avg       0.70      0.70      0.70       163
```

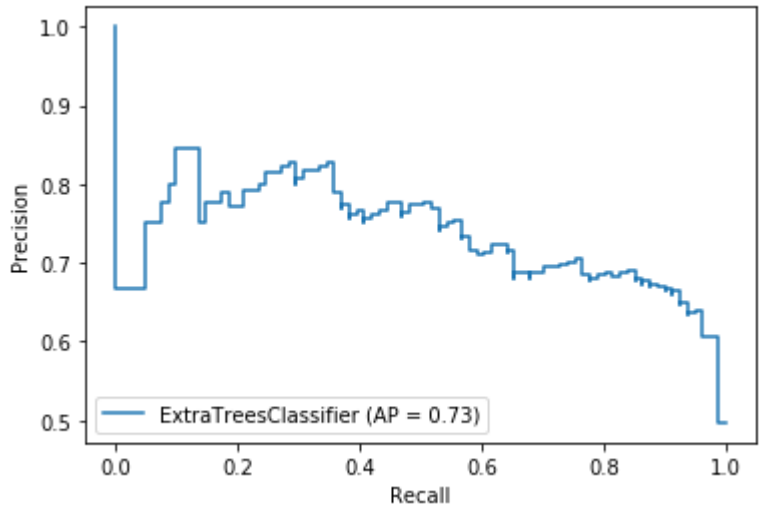
In [173]:

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

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



In [174]:

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



4- Gradient_Boosting_Classifier

In [175]:

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

In [176]:

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

Out[176]:

```
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 [177]:

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

Out[177]:

```
71.16564417177914
```

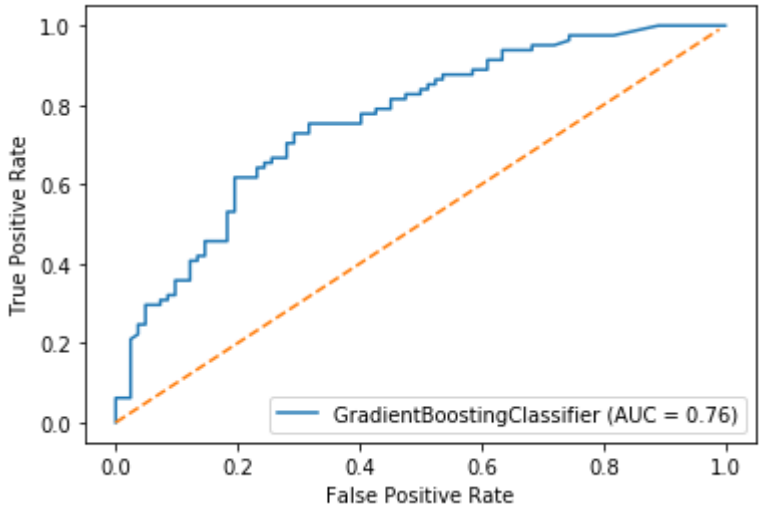
In [178]:

```
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:  71.16564417177914
Probability of detection of defect(Recall, pd):  0.7037037037037037
Probability of false alarm(pf):  0.2891566265060241
Probability of correct detection(Precision):  0.7125


F1-score or FM:  0.7080745341614907
AUC value:  0.7116079494128275


[[59 23]
 [24 57]]


              precision    recall  f1-score   support

         0.0       0.71      0.72      0.72        82
         1.0       0.71      0.70      0.71        81

    accuracy                           0.71       163
   macro avg       0.71      0.71      0.71       163
weighted avg       0.71      0.71      0.71       163
```

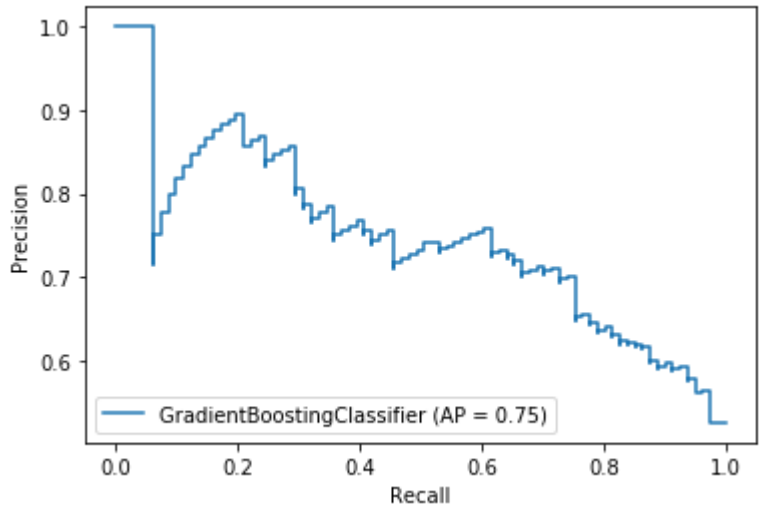
In [179]:

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

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



In [180]:

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



5- Random_Forest_Classifier

In [181]:

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

In [182]:

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

Out[182]:

```
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 [183]:

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

Out[183]:

```
68.71165644171779
```

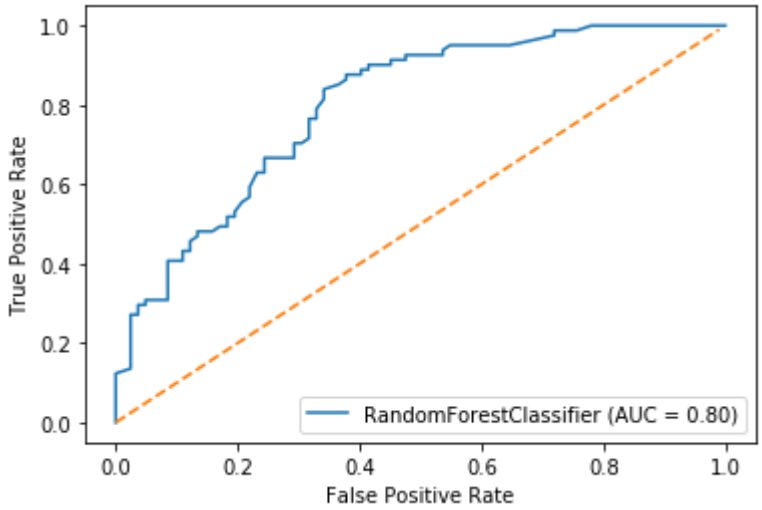
In [184]:

```
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:  68.71165644171779
Probability of detection of defect(Recall, pd):  0.6666666666666666
Probability of false alarm(pf):  0.3176470588235294
Probability of correct detection(Precision):  0.6923076923076923


F1-score or FM:  0.6792452830188679
AUC value:  0.6869918699186991


[[58 24]
 [27 54]]


              precision    recall  f1-score   support

         0.0       0.68      0.71      0.69        82
         1.0       0.69      0.67      0.68        81

    accuracy                           0.69       163
   macro avg       0.69      0.69      0.69       163
weighted avg       0.69      0.69      0.69       163
```

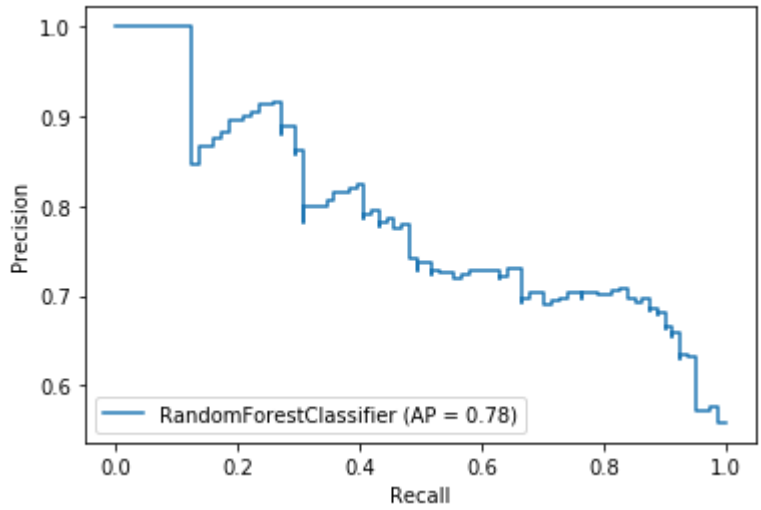
In [185]:

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

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



In [186]:

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



6- Stacking_Classifier

In [187]:

```
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 [188]:

```
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[188]:

```
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 [189]:

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

Out[189]:

```
70.5521472392638
```

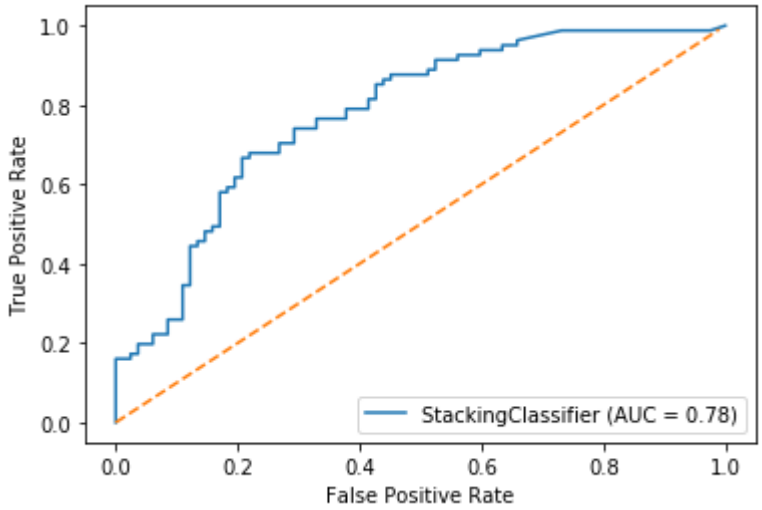
In [190]:

```
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:  70.5521472392638
Probability of detection of defect(Recall, pd):  0.7037037037037037
Probability of false alarm(pf):  0.2926829268292683
Probability of correct detection(Precision):  0.7037037037037037


F1-score or FM:  0.7037037037037037
AUC value:  0.7055103884372177


[[58 24]
 [24 57]]


              precision    recall  f1-score   support

         0.0       0.71      0.71      0.71        82
         1.0       0.70      0.70      0.70        81

    accuracy                           0.71       163
   macro avg       0.71      0.71      0.71       163
weighted avg       0.71      0.71      0.71       163
```

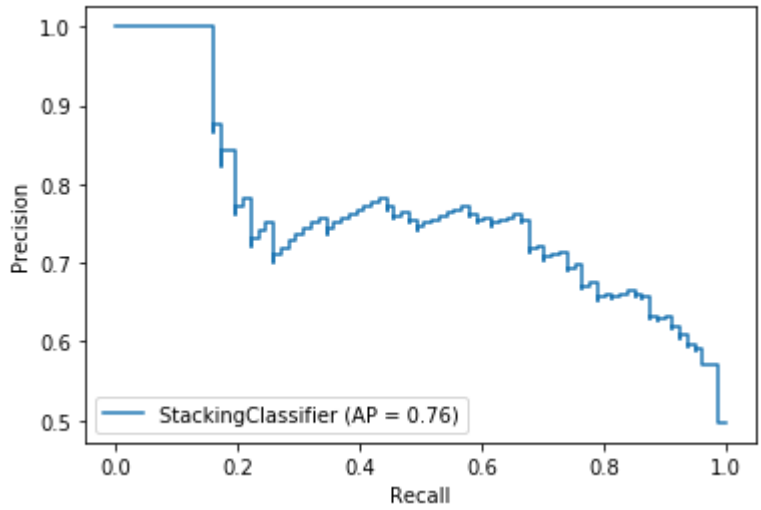
In [191]:

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

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



In [192]:

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



7- Voting_Classifier

In [193]:

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

In [194]:

```
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[194]:

```
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 [195]:

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

Out[195]:

```
63.80368098159509
```

In [196]:

```
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:  63.80368098159509
Probability of detection of defect(Recall, pd):  0.8271604938271605
Probability of false alarm(pf):  0.27450980392156865
Probability of correct detection(Precision):  0.5982142857142857


F1-score or FM:  0.6943005181347149
AUC value:  0.6391900030111413


[[37 45]
 [14 67]]


              precision    recall  f1-score   support

         0.0       0.73      0.45      0.56        82
         1.0       0.60      0.83      0.69        81

    accuracy                           0.64       163
   macro avg       0.66      0.64      0.63       163
weighted avg       0.66      0.64      0.62       163
```

In [ ]:

In [ ]: