

In [ ]:

In [17]:

```
#importing relevant libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plot
import seaborn as sns
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

In [18]:

```
cardioDataset = pd.read_csv('heart.csv')
```

In [19]:

```
cardioDataset.head(30)
```

Out[19]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
10	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
12	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
13	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
15	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
16	58	0	2	120	340	0	1	172	0	0.0	2	0	2	1
17	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
18	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
19	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
20	59	1	0	135	234	0	1	161	0	0.5	1	0	3	1

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
21	44	1	2	130	233	0	1	179	1	0.4	2	0	2	1
22	42	1	0	140	226	0	1	178	0	0.0	2	0	2	1
23	61	1	2	150	243	1	1	137	1	1.0	1	0	2	1
24	40	1	3	140	199	0	1	178	1	1.4	2	0	3	1
25	71	0	1	160	302	0	1	162	0	0.4	2	2	2	1
26	59	1	2	150	212	1	1	157	0	1.6	2	0	2	1
27	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1
28	65	0	2	140	417	1	0	157	0	0.8	2	1	2	1
29	53	1	2	130	197	1	0	152	0	1.2	0	0	2	1

In [20]:

```
cardioDataset.shape
```

Out[20]:

(303, 14)

In [21]:

```
cardioDataset.describe()
```

Out[21]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.0396
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.1610
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.0000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.0000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.8000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.6000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.2000

In [22]:

```
cardioDataset.isnull().sum()
```

Out[22]:

age 0  
sex 0  
cp 0  
trestbps 0  
chol 0  
fbs 0  
restecg 0  
thalach 0  
exang 0  
oldpeak 0  
slope 0  
ca 0  
thal 0  
target 0  
dtype: int64

In [23]:

```
cardioDataset.columns = ['Age', 'Sex', 'ChestPainType', 'RestingBloodPressure', 'Cholest
```

```
erol', 'FastingBloodSugar', 'RestingECG', 'MaxHeartRate',
'ExerciseInducedAngina', 'PreviousPeak', 'Slope', 'MajorBloodVessels', 'ThalRate',
'ProbHA']
```

In [24]:

```
cardioDataset.dtypes
```

Out[24]:

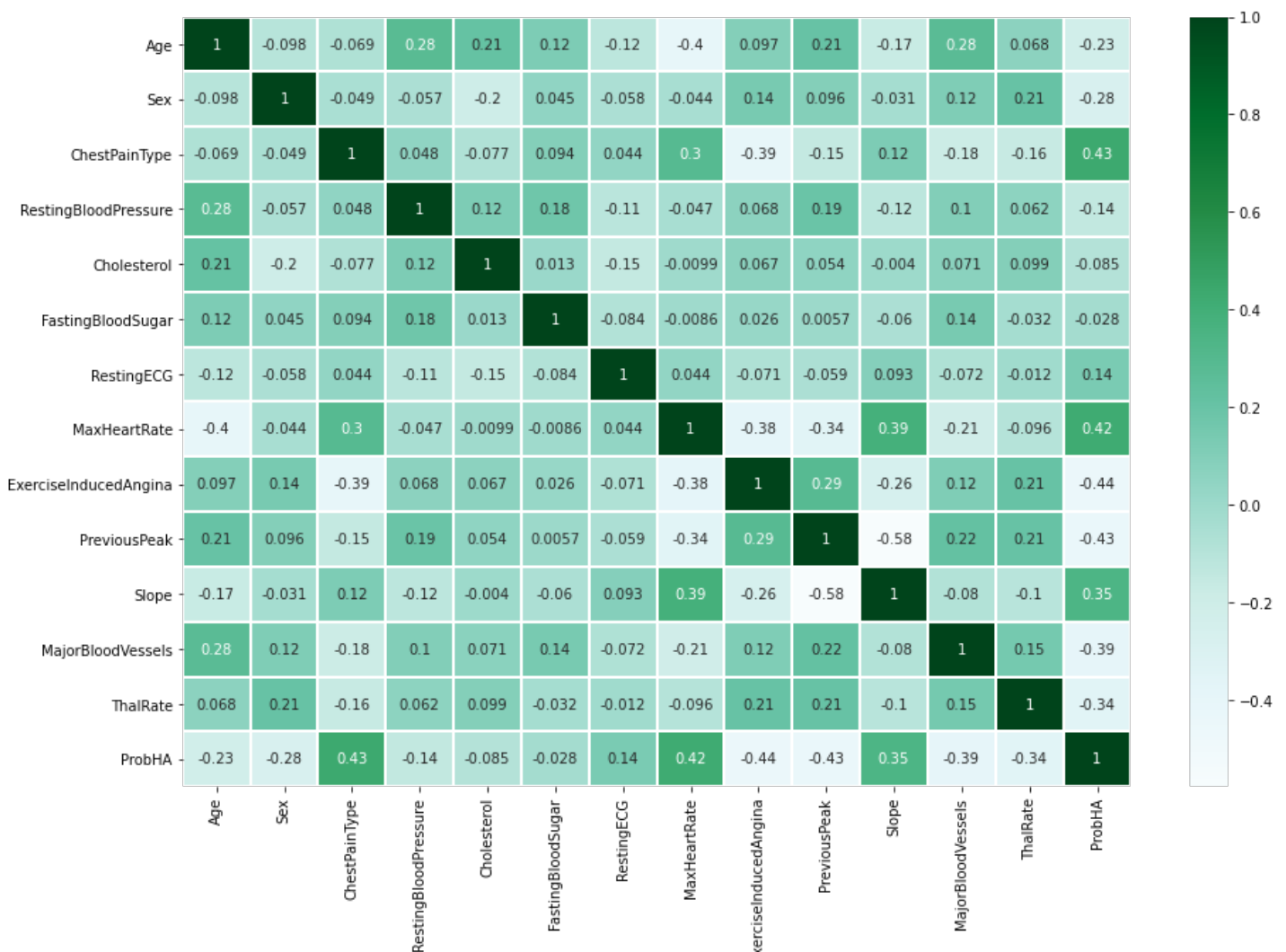
```
Age                int64
Sex                int64
ChestPainType      int64
RestingBloodPressure  int64
Cholesterol        int64
FastingBloodSugar  int64
RestingECG         int64
MaxHeartRate       int64
ExerciseInducedAngina  int64
PreviousPeak       float64
Slope              int64
MajorBloodVessels  int64
ThalRate           int64
ProbHA             int64
dtype: object
```

In [25]:

```
#Correlation matrix
plot.figure(figsize = (15, 10))

correlation = cardioDataset.corr()
mask = np.triu(np.ones_like(correlation, dtype=bool))

sns.heatmap(correlation, annot = True, cmap='BuGn', linewidths=1)
plot.show()
```

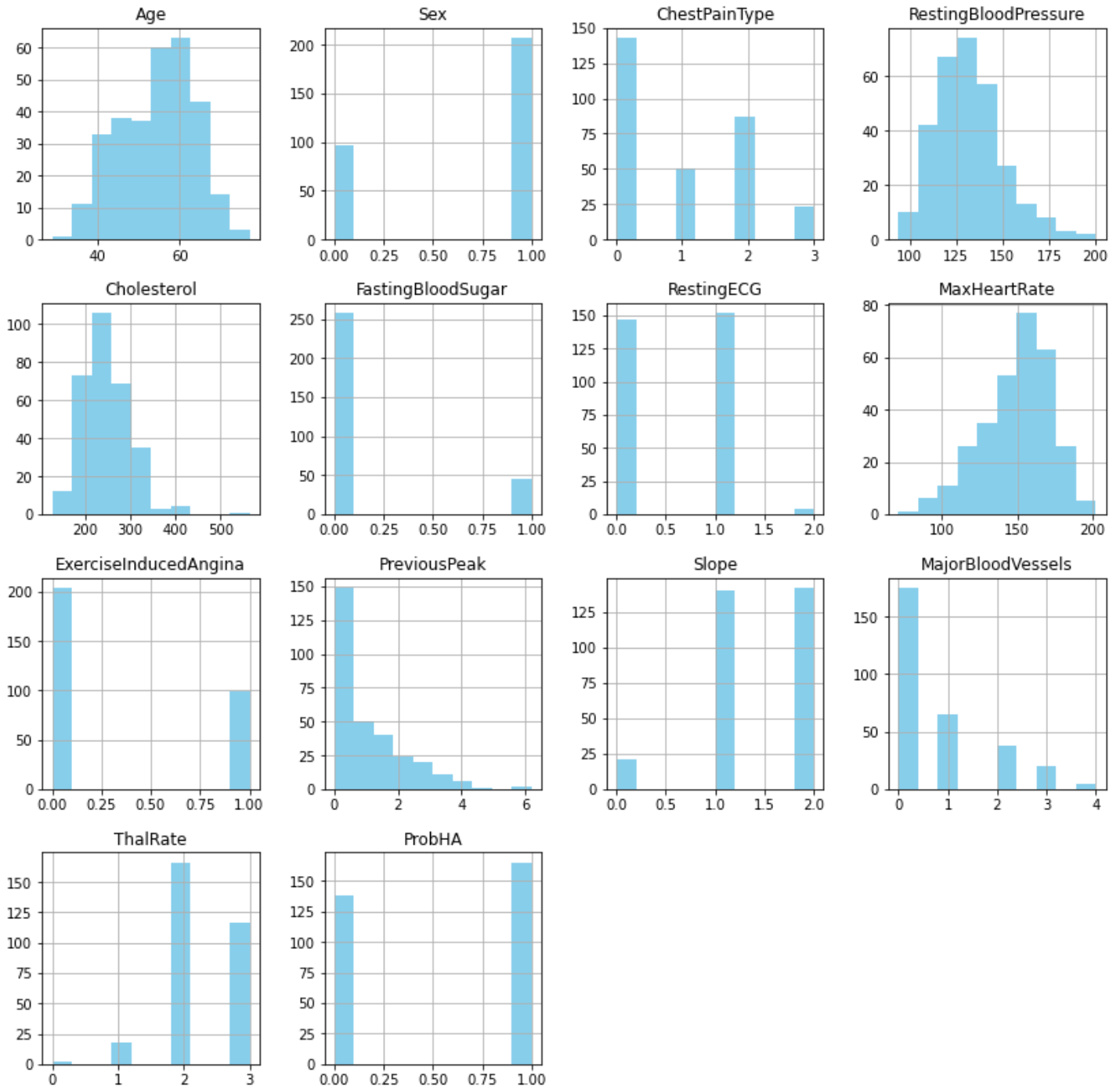


In [26]:

```
chart = plot.figure(figsize = (14, 14))
ax = chart.gca()
cardioDataset.hist(ax=ax, color="skyblue")
```

Out[26]:

```
array([[<AxesSubplot:title={'center':'Age'}>,
       <AxesSubplot:title={'center':'Sex'}>,
       <AxesSubplot:title={'center':'ChestPainType'}>,
       <AxesSubplot:title={'center':'RestingBloodPressure'}>],
      [[<AxesSubplot:title={'center':'Cholesterol'}>,
       <AxesSubplot:title={'center':'FastingBloodSugar'}>,
       <AxesSubplot:title={'center':'RestingECG'}>,
       <AxesSubplot:title={'center':'MaxHeartRate'}>],
      [[<AxesSubplot:title={'center':'ExerciseInducedAngina'}>,
       <AxesSubplot:title={'center':'PreviousPeak'}>,
       <AxesSubplot:title={'center':'Slope'}>,
       <AxesSubplot:title={'center':'MajorBloodVessels'}>],
      [[<AxesSubplot:title={'center':'ThalRate'}>,
       <AxesSubplot:title={'center':'ProbHA'}>, <AxesSubplot:>,
       <AxesSubplot:>]], dtype=object)
```



In [27]:

```
cardioDataset.var()
```

Out[27]:

```
Age                82.484558
Sex                0.217166
ChestPainType      1.065132
RestingBloodPressure  307.586453
Cholesterol        2686.426748
FastingBloodSugar  0.126877
RestingECG         0.276528
MaxHeartRate       524.646406
ExerciseInducedAngina  0.220707
PreviousPeak       1.348095
Slope              0.379735
MajorBloodVessels  1.045724
ThalRate           0.374883
ProbHA             0.248836
dtype: float64
```

In [28]:

```
cardioDataset['RestingBloodPressure']=np.log(cardioDataset['RestingBloodPressure'])
cardioDataset["Cholesterol"] = np.log(cardioDataset["Cholesterol"])
cardioDataset["MaxHeartRate"] = np.log(cardioDataset["MaxHeartRate"])

np.var(cardioDataset[["RestingBloodPressure", "Cholesterol", "MaxHeartRate"]])
```

Out[28]:

```
RestingBloodPressure    0.016894
Cholesterol              0.041401
MaxHeartRate            0.027054
dtype: float64
```

In [29]:

```
x = cardioDataset.drop('ProbHA', axis=1)
y = cardioDataset["ProbHA"]
```

In [30]:

```
#splitting the dataset

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.40, random_state=43)
```

In [40]:

```
#Logistic Regression

accuracies_Of_Algorithms= {}

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, roc_auc_score, roc_curve, plot_confusion_matrix, precision_recall_curve, plot_precision_recall_curve

def logisticModel():
    logisticRegression = LogisticRegression(penalty='l2')
    logisticRegression.fit(x_train, y_train)
```

```

y_pred = logisticRegression.predict(x_test)

y_predProbability = logisticRegression.predict_proba(x_test)[:,1]
falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability)
auc = roc_auc_score(y_test, y_predProbability)

plot.plot(falsePositiveRate, truePositiveRate, label="Logistic Regression, auc="+str
(auc))
plot.legend(loc=4)
plot.ylabel("True Positive Rate")
plot.xlabel("False Positive Rate")
plot.show()

accuracy = accuracy_score(y_test, y_pred)
accuracies_Of_Algorithms["Logistic Regression"] = accuracy*100

print('Accuracy score of Logistic Regression is:' , accuracy_score(y_test, y_pred)*1
00, "%")
print("Confusion Matrix of Logistic Regression", confusion_matrix(y_test, y_pred))

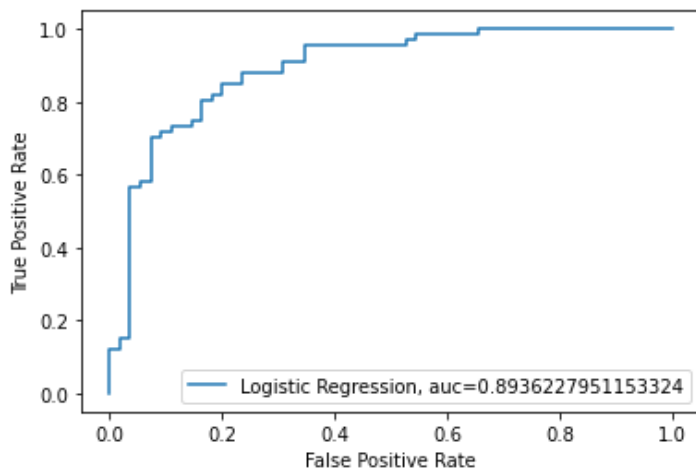
print("ClassReport", classification_report(y_test, y_pred))

plottedMatrix = plot_confusion_matrix(logisticRegression, x_test, y_test)
plottedMatrix.ax_.set_title('Confusion Matrix Of Logistic Regression', color='white')
plot.xlabel("Predicted Outcome", color='white')
plot.ylabel("Actual Outcome", color='white')
plot.gcf().axes[1].tick_params(colors='white')
plot.gcf().axes[0].tick_params(colors='white')
plot.gcf().set_size_inches(15,5)
plot.show()

disp = plot_precision_recall_curve(logisticRegression, x_test, y_test)
disp.ax_.set_title('2-class Precision-Recall curve: '
)

logisticModel()

```



Accuracy score of Logistic Regression is: 81.14754098360656 %

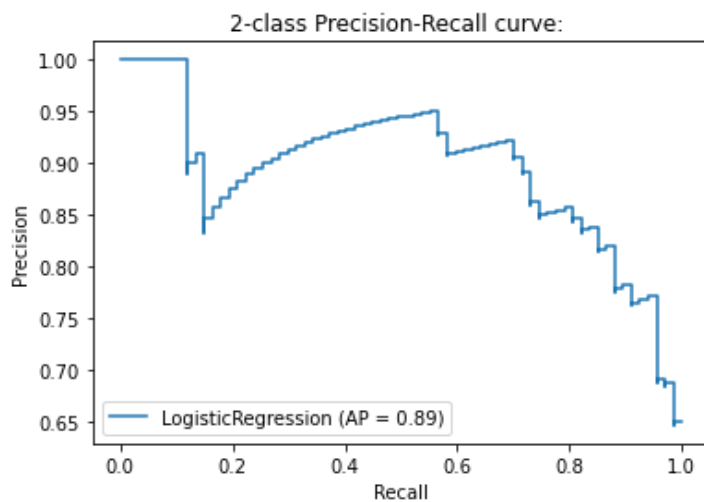
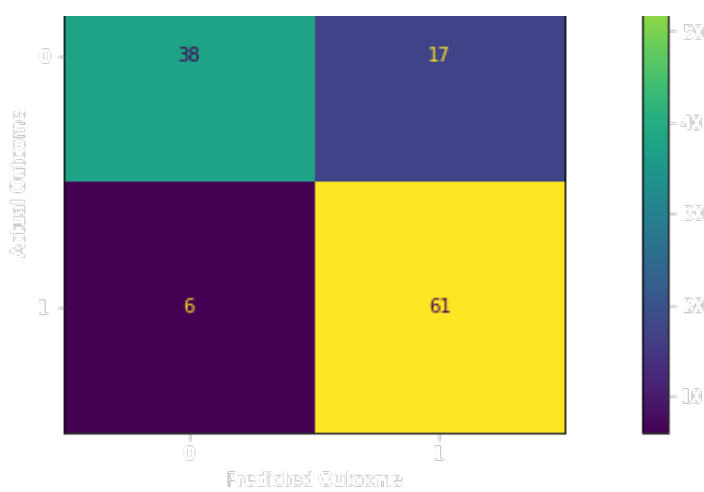
Confusion Matrix of Logistic Regression [[38 17]

[ 6 61]]

ClassReport		precision	recall	f1-score	support
	0	0.86	0.69	0.77	55
	1	0.78	0.91	0.84	67
accuracy			0.81		122
macro avg	0.82	0.80	0.80		122
weighted avg	0.82	0.81	0.81		122

Confusion Matrix Of Logistic Regression





In [32]:

```
#support vector

from sklearn.svm import SVC

def svcClassifier():

    svc = SVC(probability=True)
    svc.fit(x_train, y_train)

    y_pred1 = svc.predict(x_test)

    y_predProbability1 = svc.predict_proba(x_test)[:,1]
    falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability1)
    auc = roc_auc_score(y_test, y_predProbability1)

    plot.plot(falsePositiveRate, truePositiveRate, label="SVC, auc="+str(auc))
    plot.ylabel("True Positive Rate")
    plot.xlabel("False Positive Rate")

    plot.legend(loc=4)
    plot.show()

    accuracy1 = accuracy_score(y_test, y_pred1)
    accuracies_Of_Algorithms['supportVectorMachine'] = accuracy1*100

    accuracy_score(y_train, svc.predict(x_train))

    print("Accuracy score of the model is:", accuracy_score(y_test, y_pred1)*100, "%")
    print("Confusion matrix of the model", confusion_matrix(y_test, y_pred1))

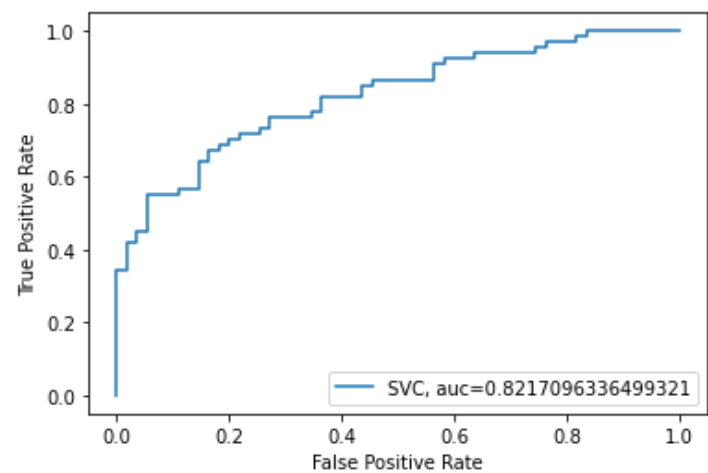
    print("Classification Report", classification_report(y_test, y_pred1))

    plottedMatrix = plot_confusion_matrix(svc, x_test, y_test)
    plottedMatrix.ax_.set_title('Confusion Matrix Of Support Vector Machines', color='whi
```

```
te')
plot.xlabel("Predicted Outcome", color='white')
plot.ylabel("Actual Outcome", color='white')
plot.gcf().axes[1].tick_params(colors='white')
plot.gcf().axes[0].tick_params(colors='white')
plot.gcf().set_size_inches(15,5)
plot.show()

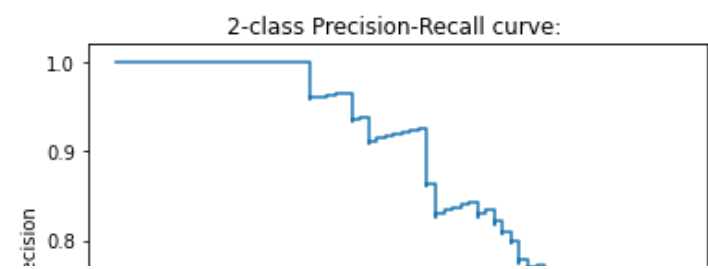
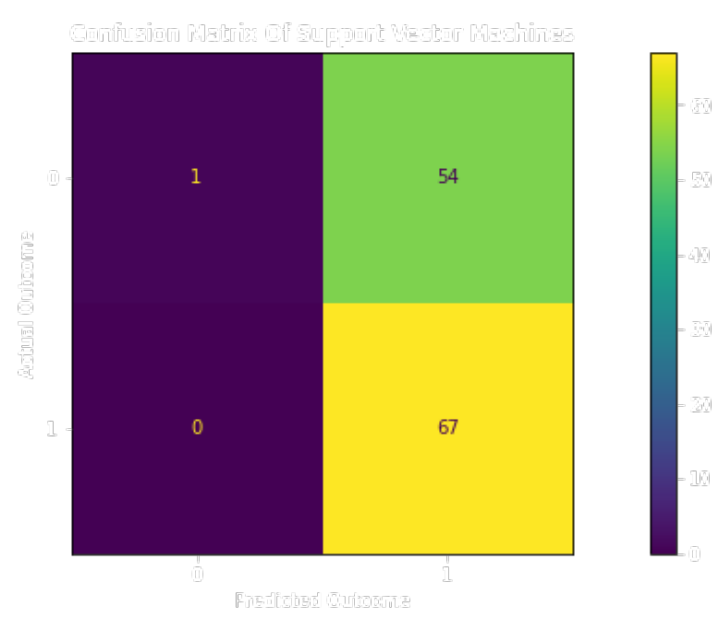
disp = plot_precision_recall_curve(svc, x_test, y_test)
disp.ax_.set_title('2-class Precision-Recall curve: ')

svcClassifier()
```

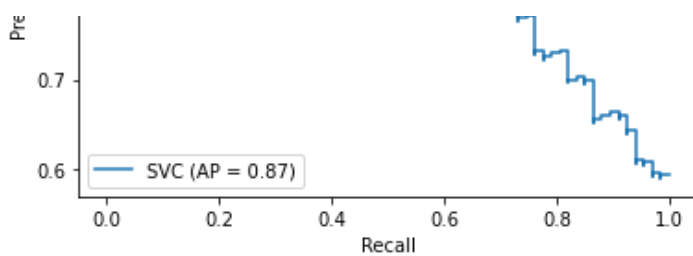


Accuracy score of the model is: 55.73770491803278 %  
 Confusion matrix of the model  $\begin{bmatrix} 1 & 54 \\ 0 & 67 \end{bmatrix}$

Classification Report		precision	recall	f1-score	support
0	1.00	0.02	0.04	55	
1	0.55	1.00	0.71	67	
accuracy			0.56	122	
macro avg	0.78	0.51	0.37	122	
weighted avg	0.75	0.56	0.41	122	







In [33]:

```
#kNearestNeighbours

from sklearn.neighbors import KNeighborsClassifier

def knnClassifier():

    knn = KNeighborsClassifier()

    knn.fit(x_train, y_train)

    y_pred2 = knn.predict(x_test)

    y_predProbability2 = knn.predict_proba(x_test)[::,1]
    falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability2)
    auc = roc_auc_score(y_test, y_predProbability2)

    plot.plot(falsePositiveRate, truePositiveRate, label="knn, auc="+str(auc))
    plot.legend(loc=4)
    plot.ylabel("True Positive Rate")
    plot.xlabel("False Positive Rate")

    plot.show()

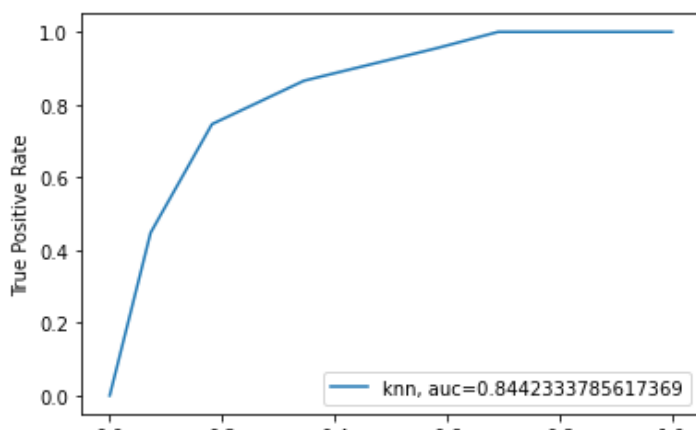
    accuracy2 = accuracy_score(y_test, y_pred2)
    accuracies_Of_Algorithms['KNeighborsClassifier'] = accuracy2*100

    accuracy_score(y_train, knn.predict(x_train))

    print("Accuracy score of the KNN:", accuracy_score(y_test, y_pred2)*100, "%")
    print("Confusion matrix of the model", confusion_matrix(y_test, y_pred2))

    print("Classification Report", classification_report(y_test, y_pred2))
    plottedMatrix = plot_confusion_matrix(knn,x_test, y_test)
    plottedMatrix.ax_.set_title('Confusion Matrix Of knn', color='white')
    plot.xlabel("Predicted Outcome", color='white')
    plot.ylabel("Actual Outcome", color='white')
    plot.gcf().axes[1].tick_params(colors='white')
    plot.gcf().axes[0].tick_params(colors='white')
    plot.gcf().set_size_inches(15,5)
    plot.show()

    disp = plot_precision_recall_curve(knn, x_test, y_test)
    disp.ax_.set_title('2-class Precision-Recall curve: ')
    knnClassifier()
```



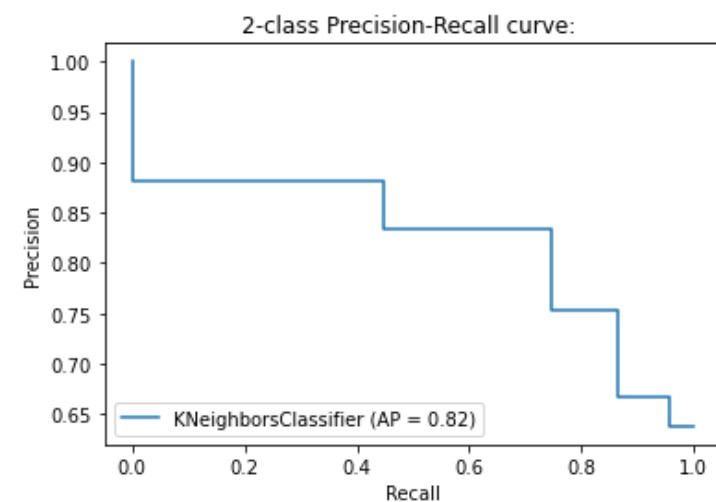
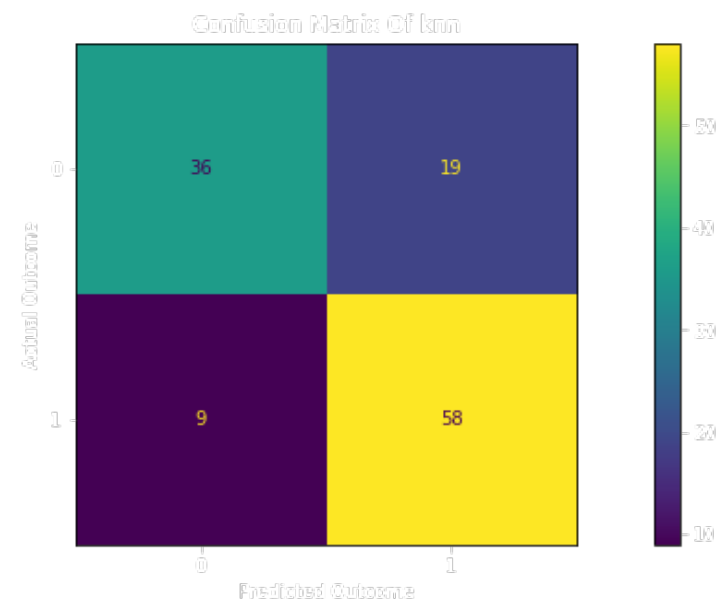
False Positive Rate

Accuracy score of the KNN: 77.04918032786885 %

Confusion matrix of the model [[36 19]

[ 9 58]]

Classification Report		precision	recall	f1-score	support
0	0.80	0.65	0.72	55	
1	0.75	0.87	0.81	67	
accuracy		0.77	122		
macro avg	0.78	0.76	0.76	122	
weighted avg	0.77	0.77	0.77	122	



In [34]:

```
from sklearn.tree import DecisionTreeClassifier

def decisionClassifier():
    dt = DecisionTreeClassifier()
    dt.fit(x_train, y_train)

    y_pred3 = dt.predict(x_test)
    y_predProbability3 = dt.predict_proba(x_test)[:,1]
    falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability3)
    auc = roc_auc_score(y_test, y_predProbability3)

    plot.plot(falsePositiveRate, truePositiveRate, label="Decision Tree, auc="+str(auc))
    plot.legend(loc=4)
    plot.ylabel("True Positive Rate")
    plot.xlabel("False Positive Rate")
    plot.show()
```

```

accuracy3= accuracy_score(y_test, y_pred3)
accuracies_Of_Algorithms['DecisionTreeClassifier'] = accuracy3*100

accuracy_score(y_train, dt.predict(x_train))

print("Accuracy score of the model is:", accuracy_score(y_test,y_pred3)*100, "%")

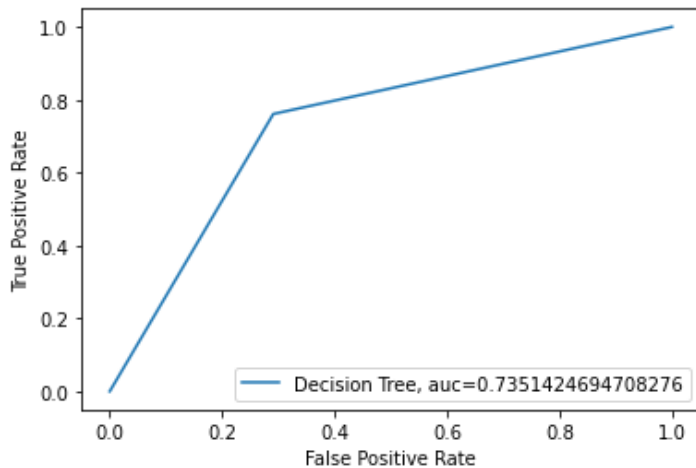
print("Confusion matrix of the model", confusion_matrix(y_test, y_pred3))

print("Classification Report", classification_report(y_test, y_pred3))

plottedMatrix = plot_confusion_matrix(dt, x_test, y_test)
plottedMatrix.ax_.set_title('Confusion Matrix Of Decision Tree', color='white')
plot.xlabel("Predicted Outcome", color='white')
plot.ylabel("Actual Outcome", color='white')
plot.gcf().axes[1].tick_params(colors='white')
plot.gcf().axes[0].tick_params(colors='white')
plot.gcf().set_size_inches(15,5)
disp = plot_precision_recall_curve(dt, x_test, y_test)
disp.ax_.set_title('2-class Precision-Recall curve: ')

```

```
decisionClassifier()
```

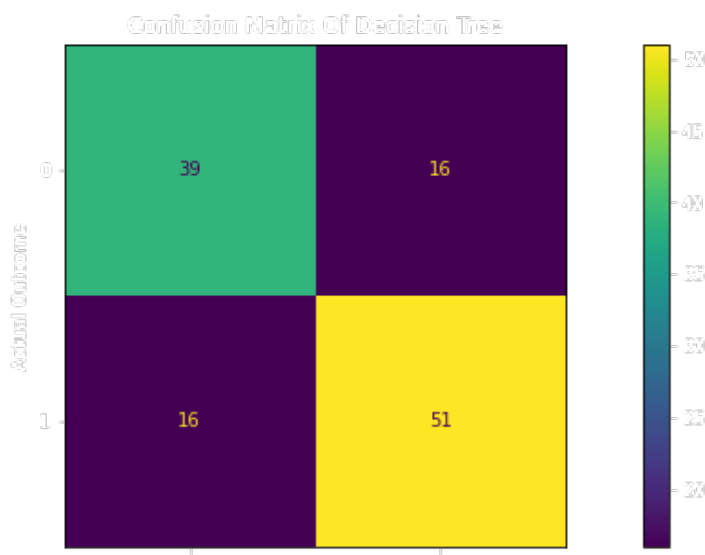


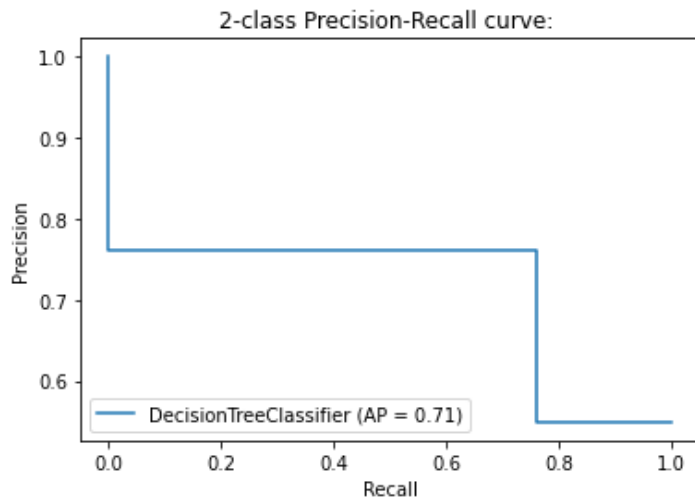
Accuracy score of the model is: 73.77049180327869 %

Confusion matrix of the model [[39 16]

[16 51]]

Classification Report		precision	recall	f1-score	support
0	0.71	0.71	0.71	55	
1	0.76	0.76	0.76	67	
accuracy		0.74	122		
macro avg	0.74	0.74	122		
weighted avg	0.74	0.74	122		





In [35]:

```
from sklearn.ensemble import RandomForestClassifier

def randomForest():
    rfc = RandomForestClassifier(criterion = 'gini', max_depth = 7, max_features = 'sqrt',
                                min_samples_leaf = 2, min_samples_split = 4, n_estimators =
180)
    rfc.fit(x_train, y_train)

    y_pred5 = rfc.predict(x_test)

    y_predProbability5 = rfc.predict_proba(x_test)[::,1]
    falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability5)
    auc = roc_auc_score(y_test, y_predProbability5)

    plot.plot(falsePositiveRate, truePositiveRate, label="Random Forest Classifier, auc="
"+str(auc))
    plot.legend(loc=4)
    plot.ylabel("True Positive Rate")
    plot.xlabel("False Positive Rate")
    plot.show()

    accuracy5 = accuracy_score(y_test, y_pred5)
    accuracies_Of_Algorithms['RandomForestClassifier'] = accuracy5*100

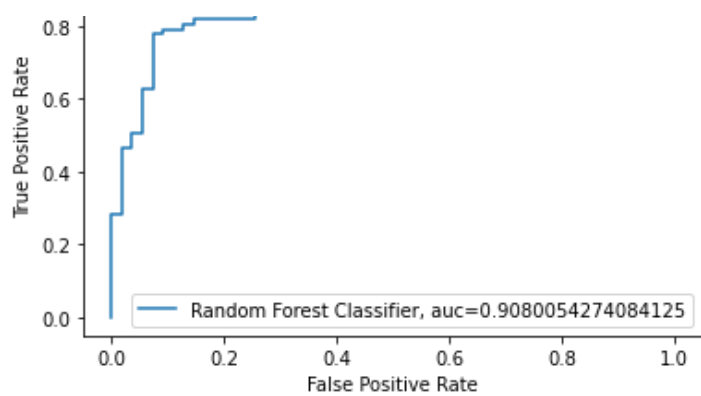
    accuracy_score(y_train, rfc.predict(x_train))
    print("Accuracy score of the model is:", accuracy_score(y_test, y_pred5)*100, "%")
    print("Confusion matrix of the model", confusion_matrix(y_test, y_pred5))

    print("Classification Report", classification_report(y_test, y_pred5))

    plottedMatrix = plot_confusion_matrix(rfc, x_test, y_test)
    plottedMatrix.ax_.set_title('Confusion Matrix Of Random Forest', color='white')
    plot.xlabel("Predicted Outcome", color='white')
    plot.ylabel("Actual Outcome", color='white')
    plot.gcf().axes[1].tick_params(colors='white')
    plot.gcf().axes[0].tick_params(colors='white')
    plot.gcf().set_size_inches(15,5)
    plot.show()
    disp = plot_precision_recall_curve(rfc, x_test, y_test)
    disp.ax_.set_title('2-class Precision-Recall curve: ')

randomForest()
```



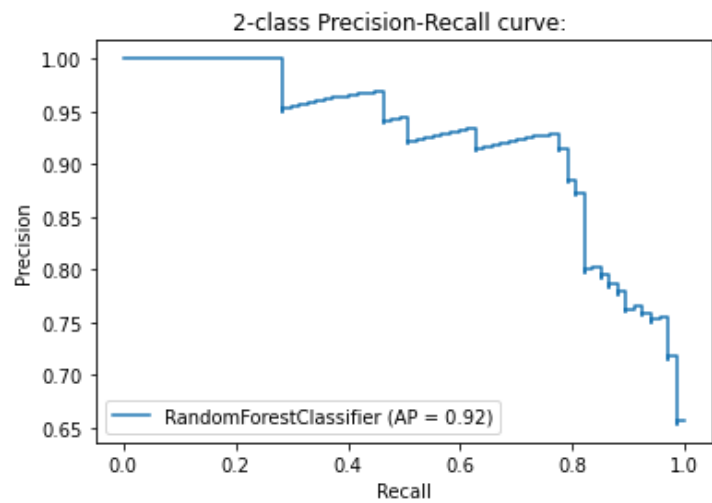
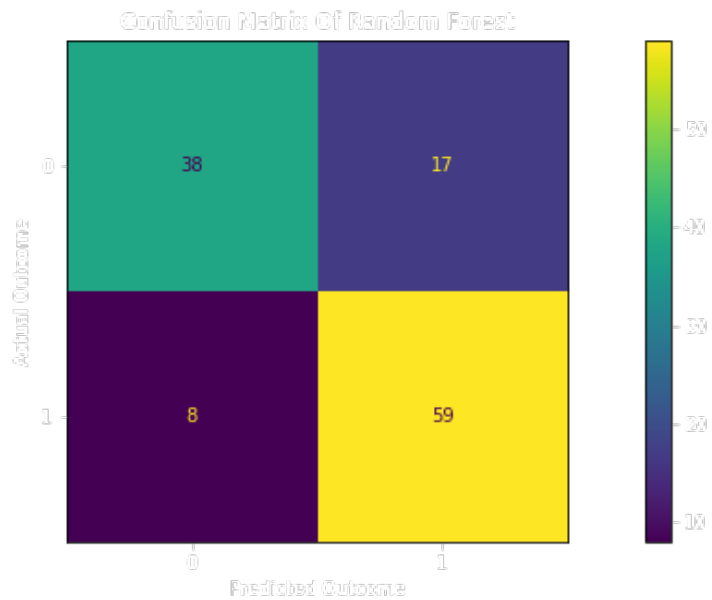


Accuracy score of the model is: 79.50819672131148 %

Confusion matrix of the model [[38 17]

[ 8 59]]

Classification Report		precision	recall	f1-score	support
0	0.83	0.69	0.75	0.72	55
1	0.78	0.88	0.83	0.85	67
accuracy			0.80		122
macro avg	0.80	0.79	0.79	0.79	122
weighted avg	0.80	0.80	0.79	0.79	122



In [36]:

```

from sklearn.ensemble import GradientBoostingClassifier

def GradientBoost():

```

```

gbc = GradientBoostingClassifier()

gbc.fit(x_train, y_train)

y_pred6 = gbc.predict(x_test)

y_predProbability6 = gbc.predict_proba(x_test)[::,1]
falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability6)
auc = roc_auc_score(y_test, y_predProbability6)

plot.plot(falsePositiveRate, truePositiveRate, label="Gradient Boost, auc="+str(auc)
)
plot.legend(loc=4)
plot.ylabel("True Positive Rate")
plot.xlabel("False Positive Rate")
plot.show()

accuracy6=accuracy_score(y_test, y_pred6)

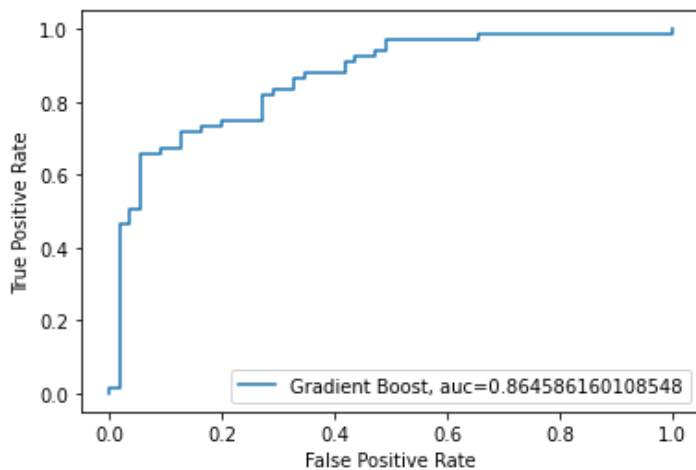
accuracies_Of_Algorithms['GradientBoosting'] = accuracy6*100
print("Accuracy score of the model is:", accuracy_score(y_test, y_pred6)*100, "%")

print("Confusion matrix of the model", confusion_matrix(y_test, y_pred6))

print("Classification Report", classification_report(y_test, y_pred6))

plottedMatrix = plot_confusion_matrix(gbc, x_test, y_test)
plottedMatrix.ax_.set_title('Confusion Matrix Of Gradient Boost', color='white')
plot.xlabel("Predicted Outcome", color='white')
plot.ylabel("Actual Outcome", color='white')
plot.gcf().axes[1].tick_params(colors='white')
plot.gcf().axes[0].tick_params(colors='white')
plot.gcf().set_size_inches(15,5)
plot.show()
disp = plot_precision_recall_curve(gbc, x_test, y_test)
disp.ax_.set_title('2-class Precision-Recall curve: ')
GradientBoost()

```



Accuracy score of the model is: 77.04918032786885 %

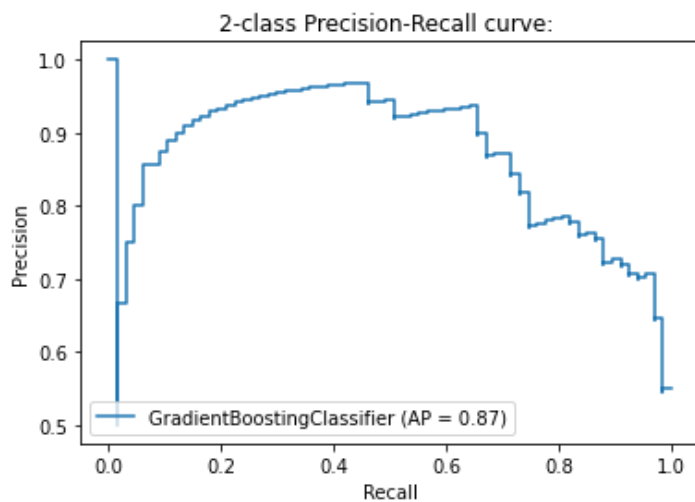
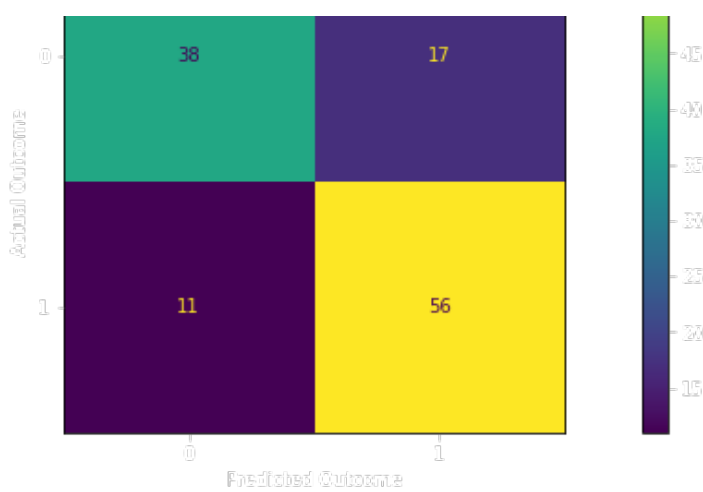
Confusion matrix of the model [[38 17]

[11 56]]

Classification Report		precision	recall	f1-score	support
0	0.78	0.69	0.73	55	
1	0.77	0.84	0.80	67	
accuracy		0.77	122		
macro avg	0.77	0.76	0.77	122	
weighted avg	0.77	0.77	0.77	122	

Confusion Matrix Of Gradient Boost





In [37]:

```
from sklearn.neural_network import MLPClassifier

def multiLayerClassifier():

    multiLayerPerceptron = MLPClassifier(random_state = 10,max_iter=300,activation="relu",
                                         hidden_layer_sizes=(34,34,34))

    multiLayerPerceptron.fit(x_train, y_train)

    y_pred7 = multiLayerPerceptron.predict(x_test)

    y_predProbability7 = multiLayerPerceptron.predict_proba(x_test)[:,1]
    falsePositiveRate, truePositiveRate, _ = roc_curve(y_test, y_predProbability7)
    auc = roc_auc_score(y_test, y_predProbability7)

    plot.plot(falsePositiveRate, truePositiveRate, label="Multi Layer Perceptron, auc="+
str(auc))
    plot.legend(loc=4)
    plot.ylabel("True Positive Rate")
    plot.xlabel("False Positive Rate")
    plot.show()
    accuracy7 = accuracy_score(y_test, y_pred7)

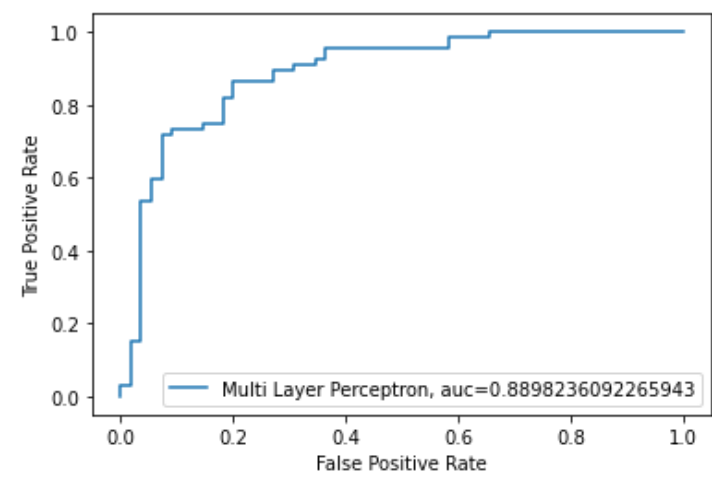
    accuracies_Of_Algorithms['MLPClassifier'] = accuracy7*100

    print("Confusion Matrix of MLPClassifier", confusion_matrix(y_test, y_pred7))
    print("Classification Report", classification_report(y_test, y_pred7))
    plottedMatrix = plot_confusion_matrix(multiLayerPerceptron, x_test, y_test)
    plottedMatrix.ax_.set_title('Confusion Matrix Of MultiLayer PERCEPTRON', color='white')

    plot.xlabel("Predicted Outcome", color='white')
    plot.ylabel("Actual Outcome", color='white')
    plot.gcf().axes[1].tick_params(colors='white')
```

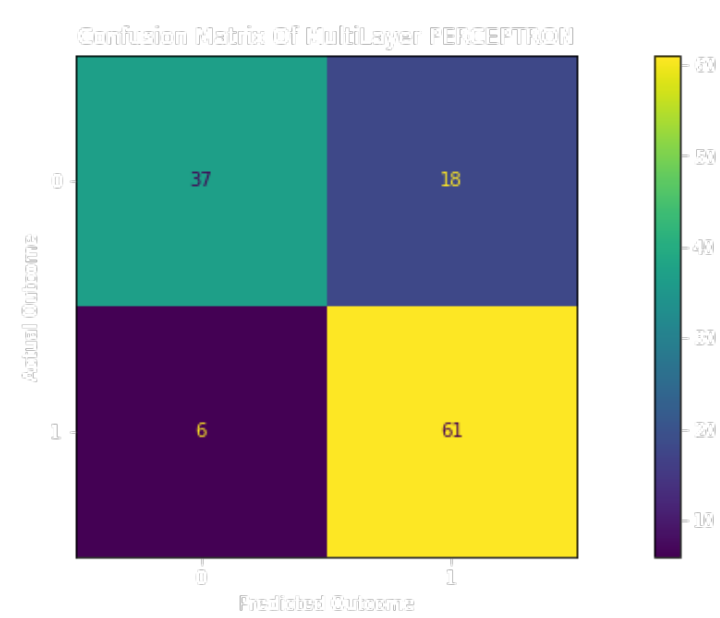
```
plot.gcf().axes[0].tick_params(colors='white')
plot.gcf().set_size_inches(15,5)
plot.show()

accuracy_score(y_test, y_pred7)
print("Accuracy score of the model is:", accuracy_score(y_test, y_pred7)*100, "%")
disp = plot_precision_recall_curve(multiLayerPerceptron, x_test, y_test)
disp.ax_.set_title('2-class Precision-Recall curve: ')
multiLayerClassifier()
```

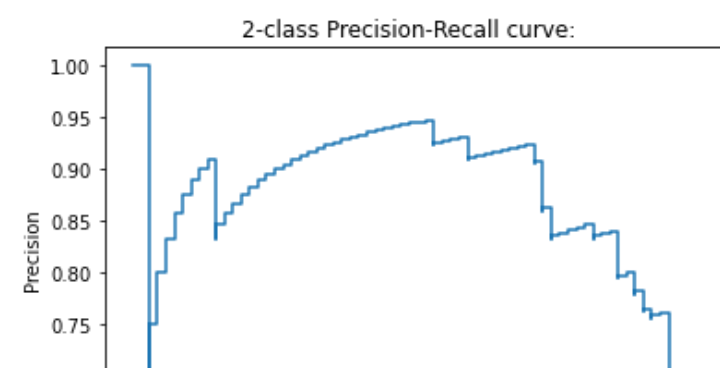


Confusion Matrix of MLPClassifier [[37 18]  
[ 6 61]]

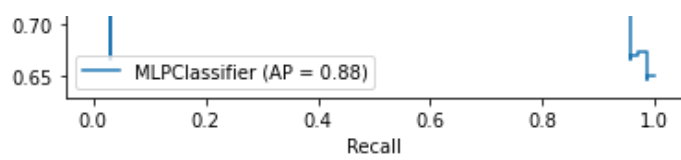
Classification Report		precision	recall	f1-score	support
0	0.86	0.67	0.76	55	
1	0.77	0.91	0.84	67	
accuracy		0.80	122		
macro avg	0.82	0.79	0.80	122	
weighted avg	0.81	0.80	0.80	122	



Accuracy score of the model is: 80.32786885245902 %

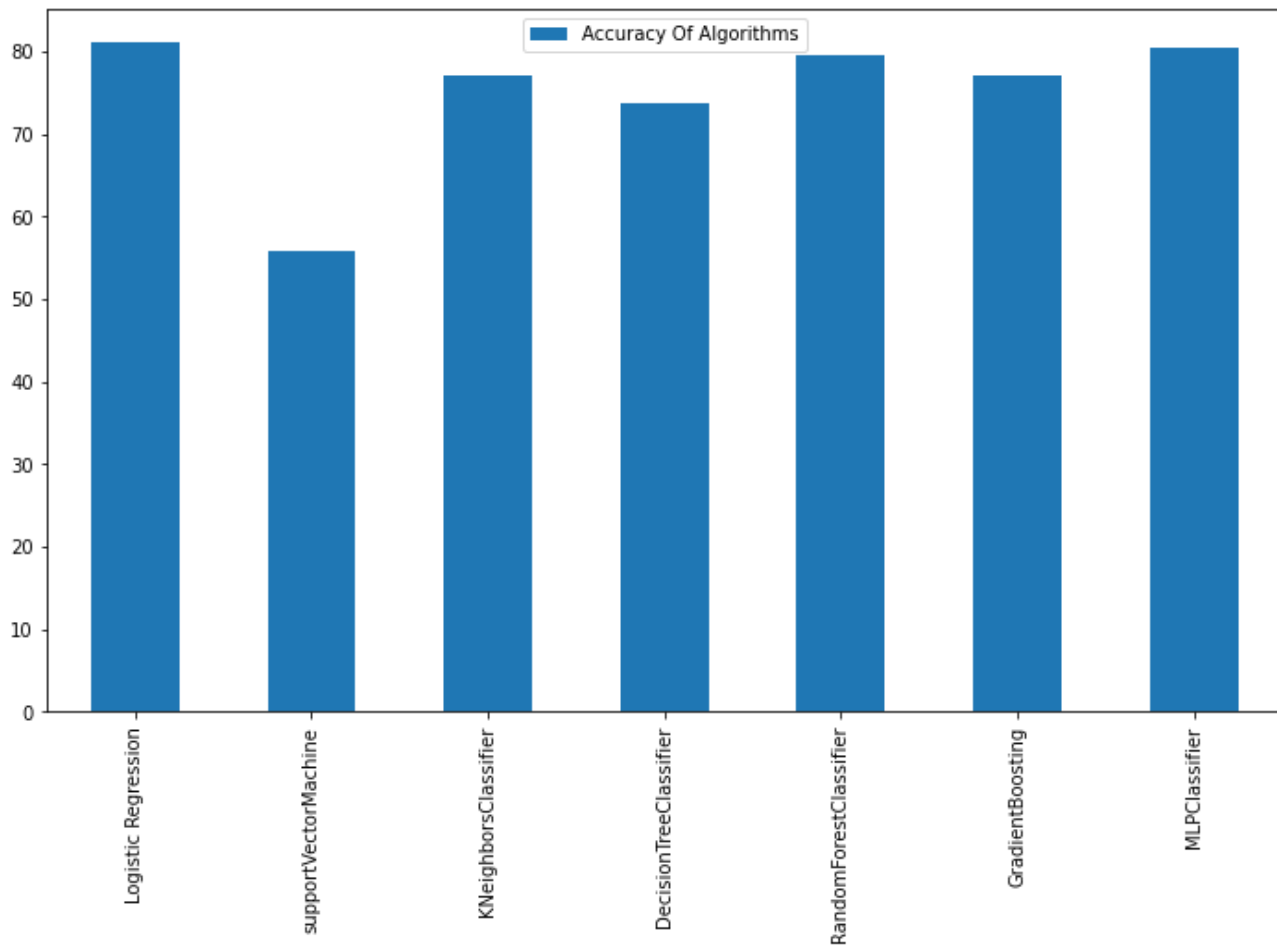






In [38]:

```
cardioBarChart = pd.DataFrame(accuracies_Of_Algorithms.values(),  
                              accuracies_Of_Algorithms.keys(),  
                              columns=["Accuracy Of Algorithms"])  
  
cardioBarChart.plot.bar(figsize = (12,7));
```



In [ ]:

In [ ]:

In [ ]: