In [ ]:

```
```

In [25]:

```
#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 [26]:

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

In [27]:

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

Out[27]:

	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 [28]:

```
cardioDataset.shape
```

Out[28]:

```
(303, 14)
```

In [29]:

```
cardioDataset.describe()
```

Out[29]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpe
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0000
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 [30]:

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

Out[30]:

```
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 [31]:

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

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

In [32]:

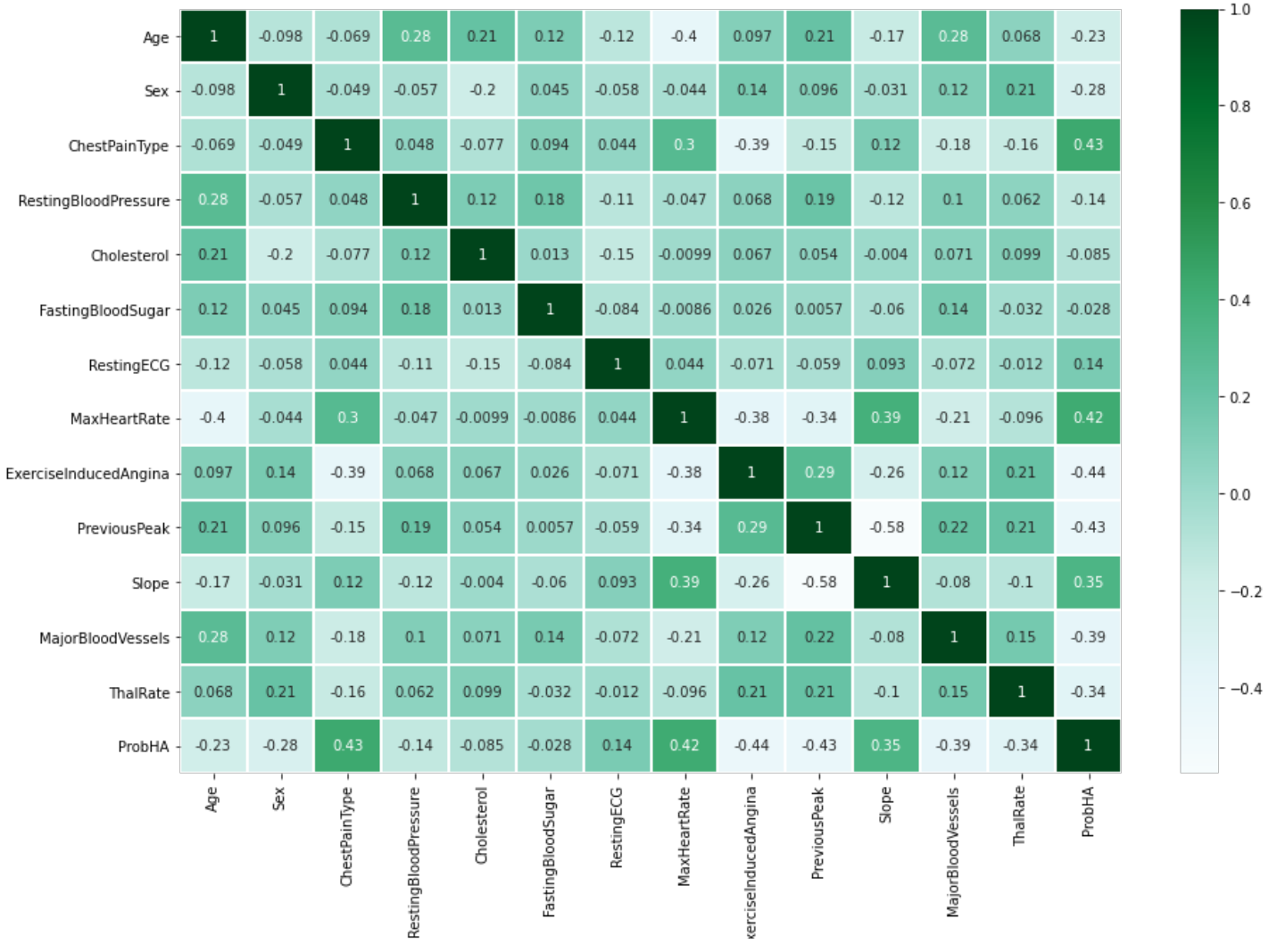
```
cardioDataset.dtypes
```

Out[32]:

```
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 [33]:

```
#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()
```

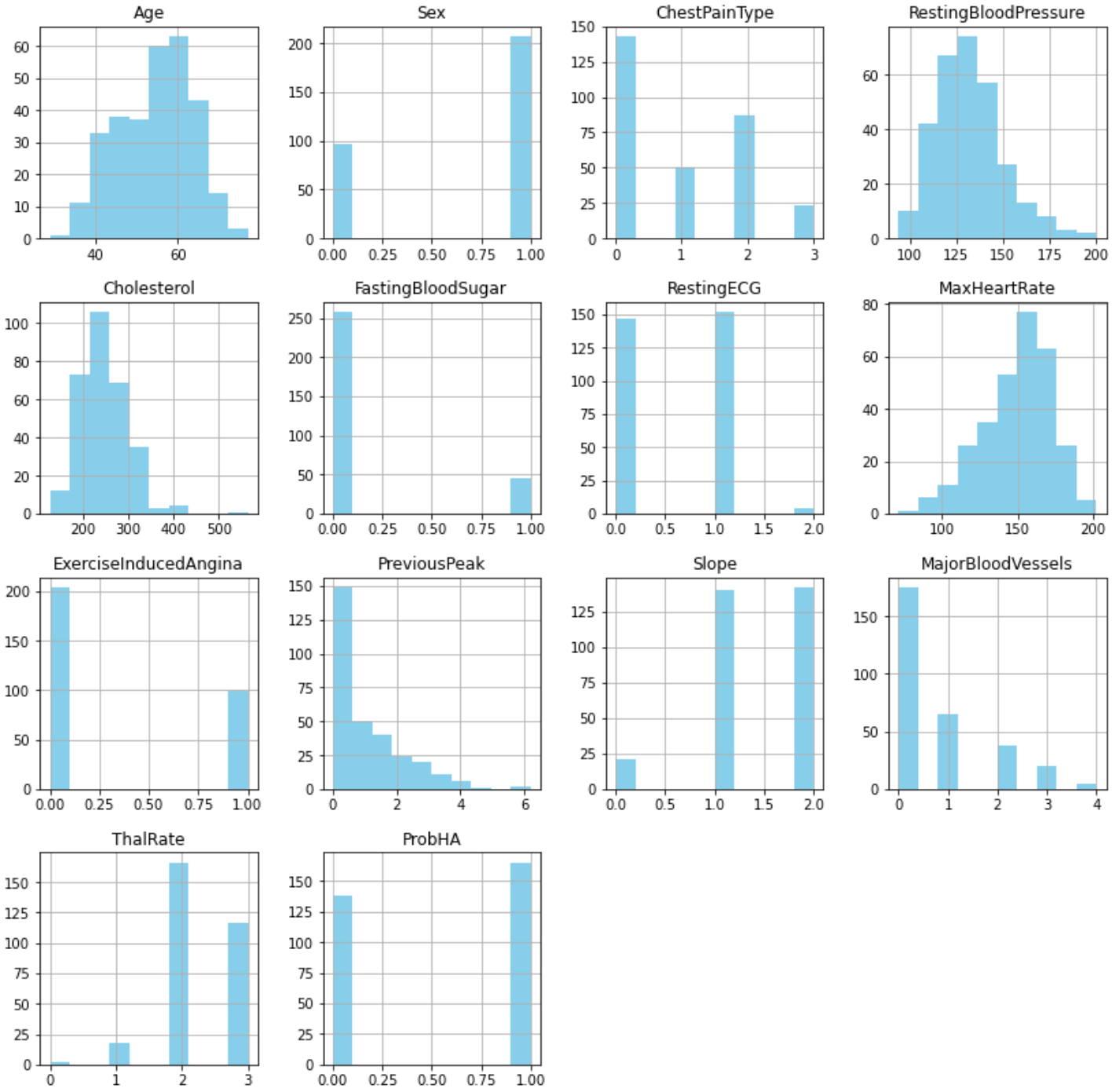


In [34]:

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

Out[34]:

```
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 [35]:

```
cardioDataset.var()
```

Out[35]:

```
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 [36]:

```
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[36]:

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

In [37]:

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

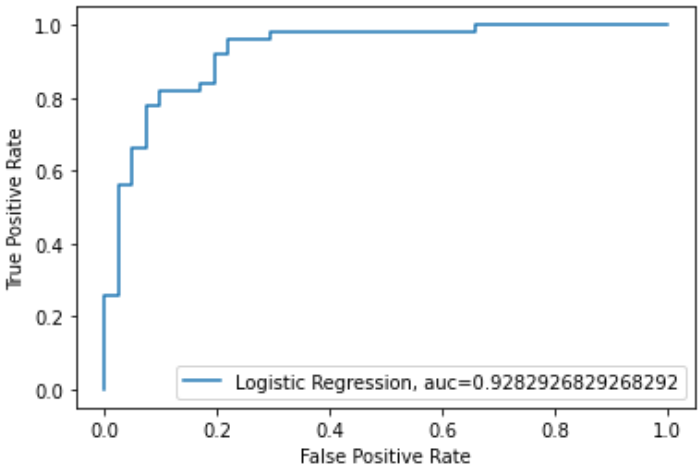
In [38]:

```
#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=0.30, random_state=43)
```

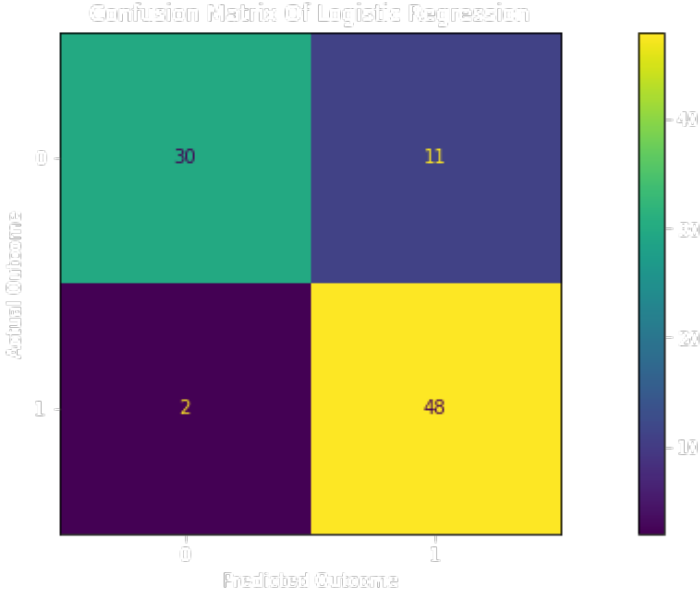
In [39]:

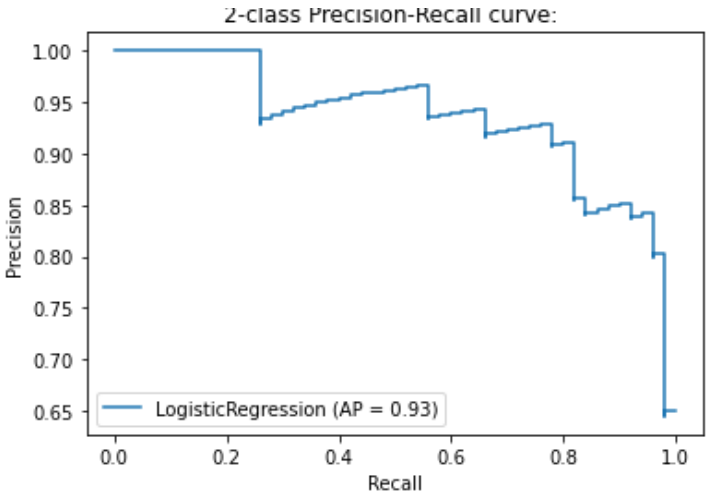
```
#Logistic Regression

accuracies_Of_Algorithms= {}


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_s
core, roc_auc_score, roc_curve,plot_confusion_matrix,precision_recall_curve, plot_precisi
on_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("ClassificationReport", 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()
```



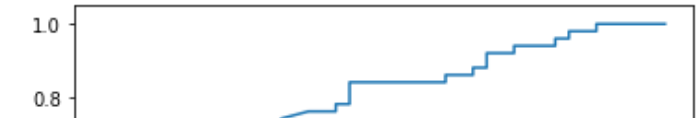

```
Accuracy score of Logistic Regression is: 85.71428571428571 %
Confusion Matrix of Logistic Regression [[30 11]
 [ 2 48]]
ClassificationReport                precision    recall  f1-score   support

           0       0.94      0.73      0.82        41
           1       0.81      0.96      0.88        50

    accuracy                           0.86        91
   macro avg       0.88      0.85      0.85        91
weighted avg       0.87      0.86      0.85        91
```

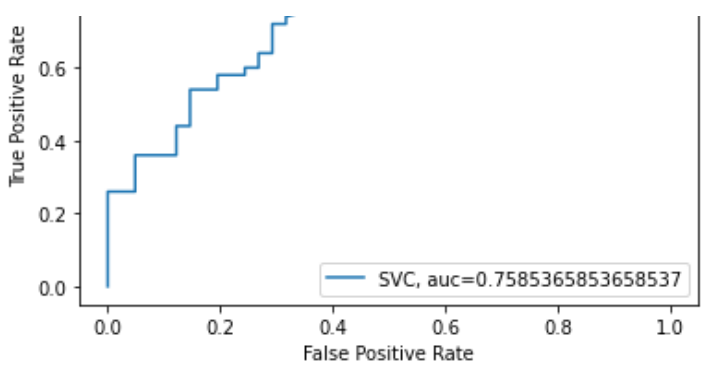




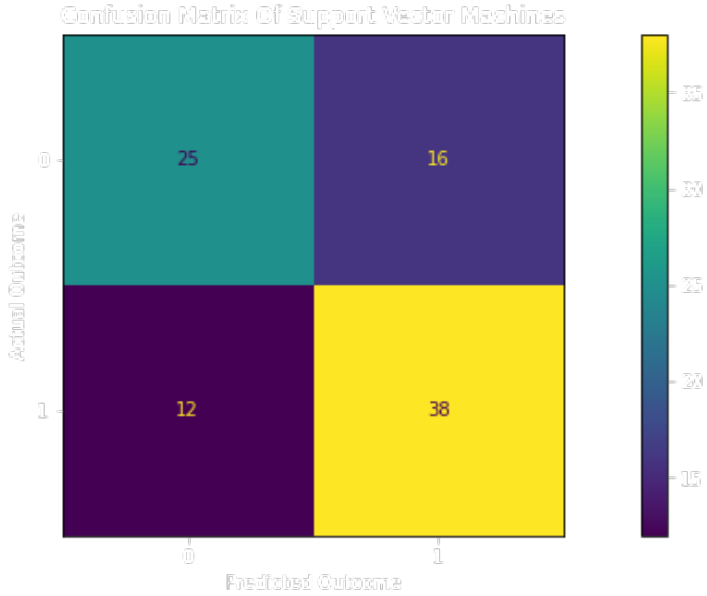
In [40]:

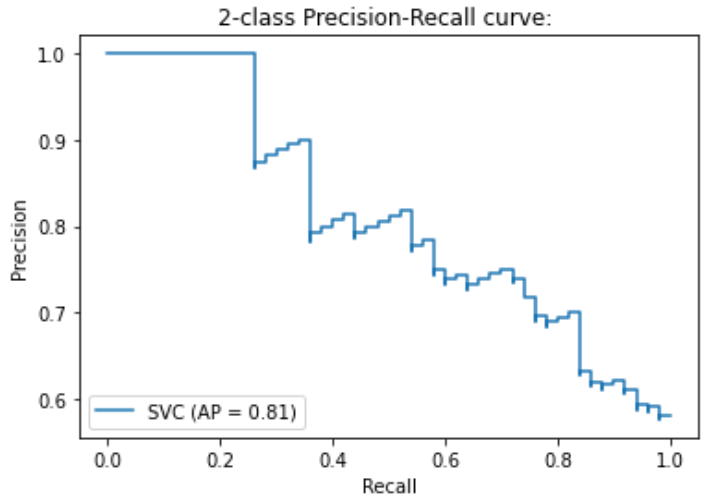
```
#support vector

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: 69.23076923076923 %
Confusion matrix of the model [[25 16]
 [12 38]]
Classification Report                precision    recall  f1-score   support

           0       0.68      0.61      0.64        41
           1       0.70      0.76      0.73        50

    accuracy                           0.69        91
   macro avg       0.69      0.68      0.69        91
weighted avg       0.69      0.69      0.69        91
```

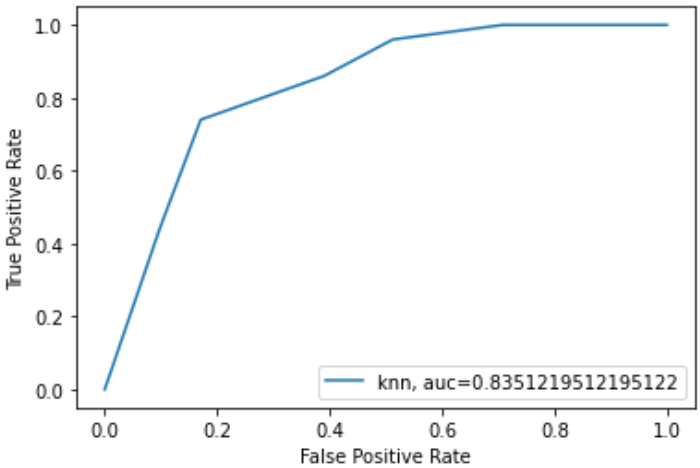




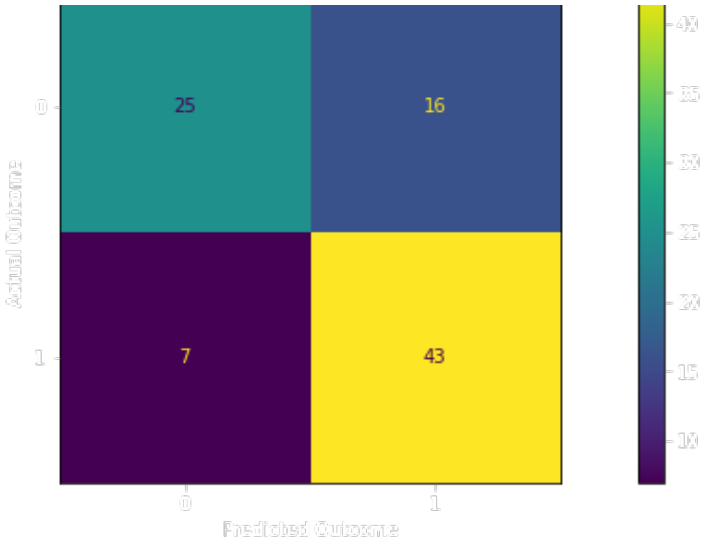
In [41]:

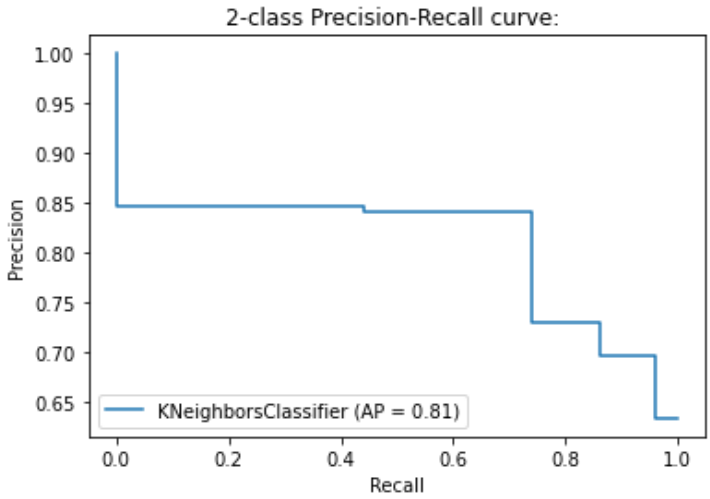
```
#kNearestNeighbours

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()
```




```
Accuracy score of the KNN: 74.72527472527473 %
Confusion matrix of the model [[25 16]
 [ 7 43]]
Classification Report               precision    recall  f1-score   support

           0       0.78      0.61      0.68        41
           1       0.73      0.86      0.79        50

    accuracy                           0.75        91
   macro avg       0.76      0.73      0.74        91
weighted avg       0.75      0.75      0.74        91
```

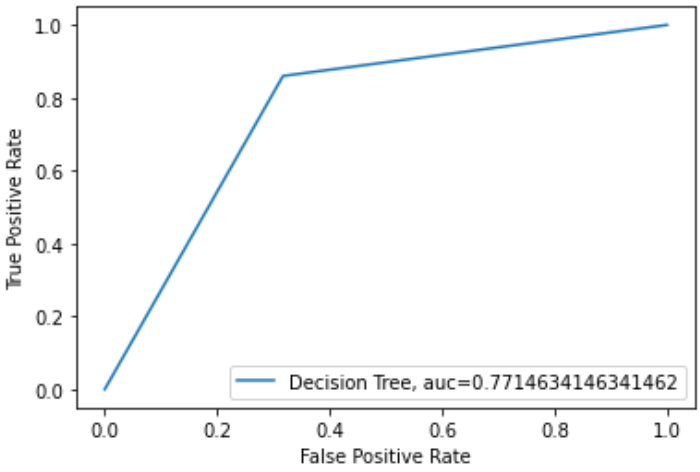




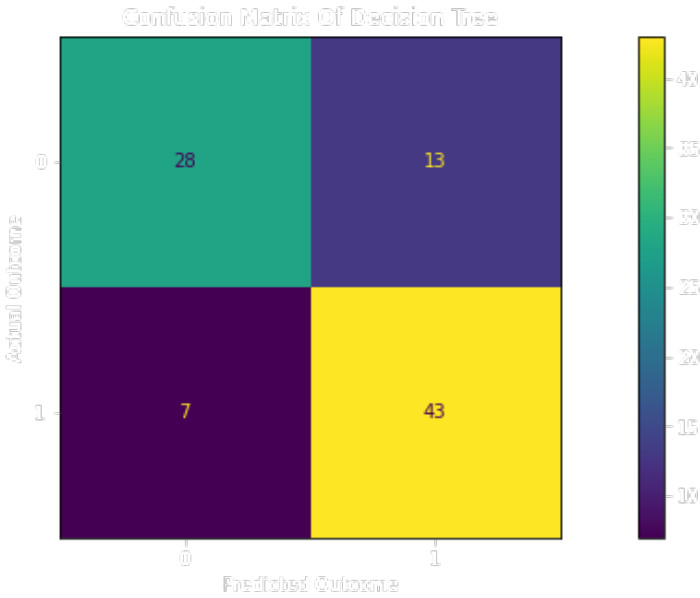
In [42]:

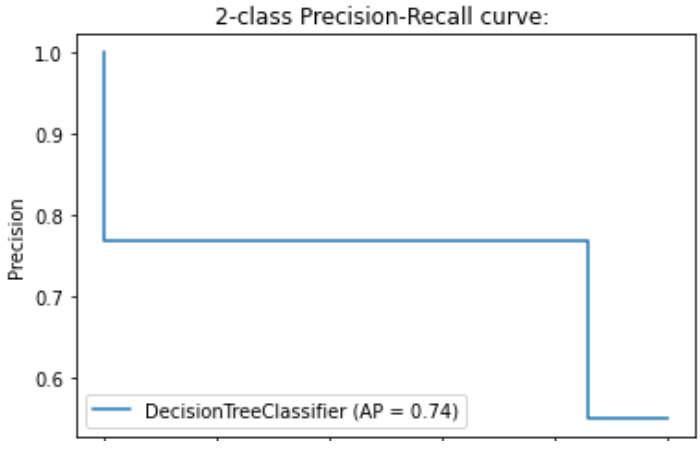
```
from sklearn.tree import DecisionTreeClassifier

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')
```

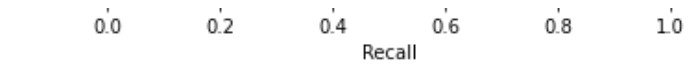
```
    plot.gcf().axes[1].tick_params(colors='white')
    plot.gcf().axes[0].tick_params(colors='white')
    plot.gcf().set_size_inches(15,5)
    disp = plot_precision_recall_curve(dt, x_test, y_test)
    disp.ax_.set_title('2-class Precision-Recall curve: ')

decisionClassifier()
```




```
Accuracy score of the model is: 78.02197802197803 %
Confusion matrix of the model [[28 13]
 [ 7 43]]
Classification Report                precision    recall  f1-score   support

           0       0.80      0.68      0.74        41
           1       0.77      0.86      0.81        50

    accuracy                           0.78        91
   macro avg       0.78      0.77      0.77        91
weighted avg       0.78      0.78      0.78        91
```

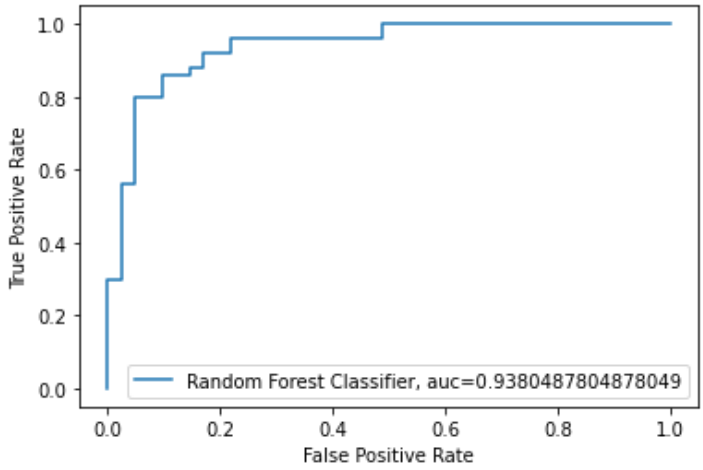






In [43]:

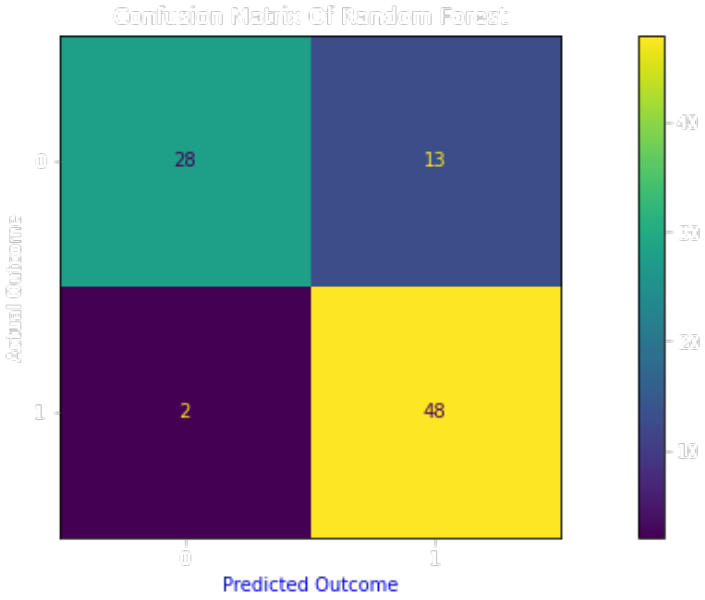
```
from sklearn.ensemble import RandomForestClassifier


def randomForest():

    rfc = RandomForestClassifier(criterion = 'gini', max_depth = 7,  max_features = 'sqr
t',
                            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='blue')
    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()
```

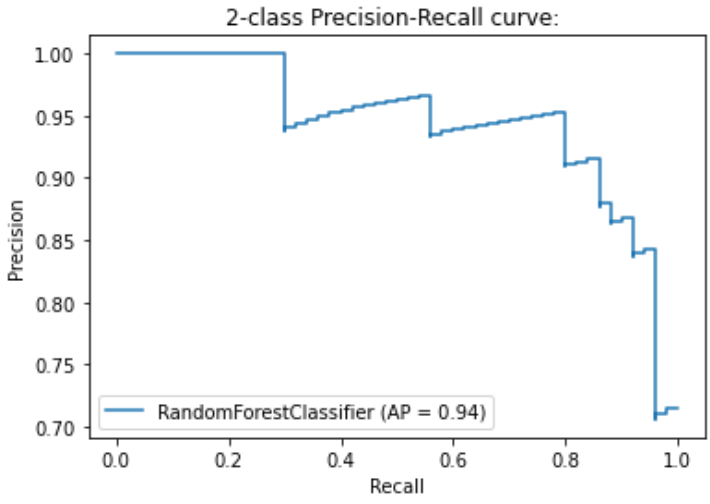



```
Accuracy score of the model is: 83.51648351648352 %
Confusion matrix of the model [[28 13]
 [ 2 48]]
Classification Report              precision    recall  f1-score   support
```

Classification Report precision recall f1-score support

0	0.93	0.68	0.79	41
1	0.79	0.96	0.86	50
accuracy			0.84	91
macro avg	0.86	0.82	0.83	91
weighted avg	0.85	0.84	0.83	91

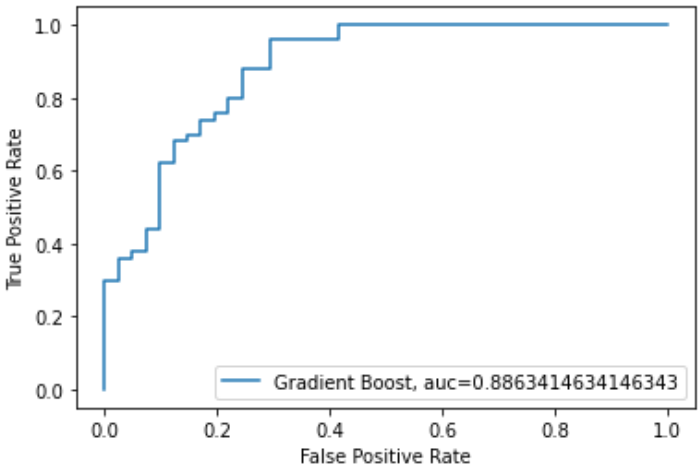




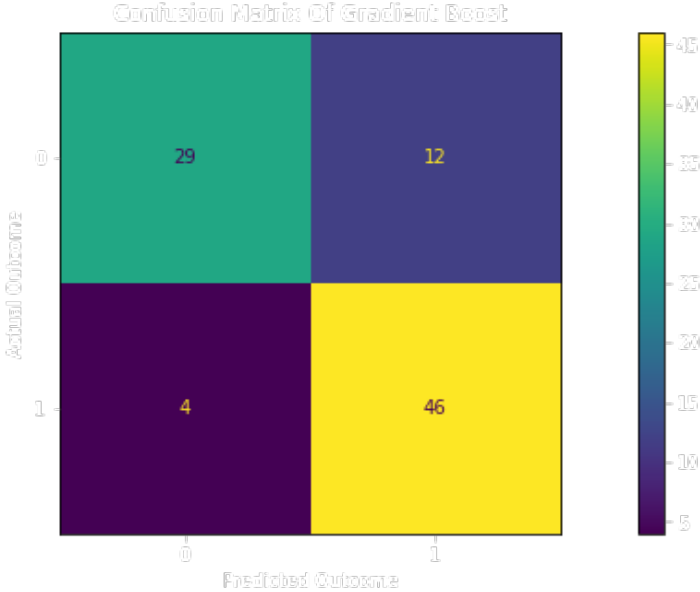
In [44]:

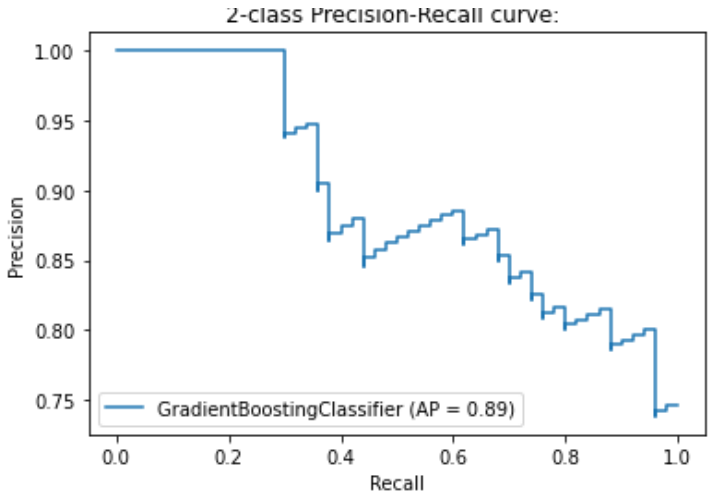
```
from sklearn.ensemble import GradientBoostingClassifier


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()
```



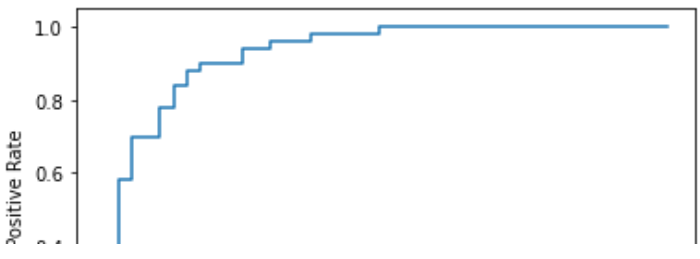

```
Accuracy score of the model is: 82.41758241758241 %
Confusion matrix of the model [[29 12]
 [ 4 46]]
Classification Report                precision    recall  f1-score   support

           0       0.88      0.71      0.78        41
           1       0.79      0.92      0.85        50

    accuracy                           0.82        91
   macro avg       0.84      0.81      0.82        91
weighted avg       0.83      0.82      0.82        91
```

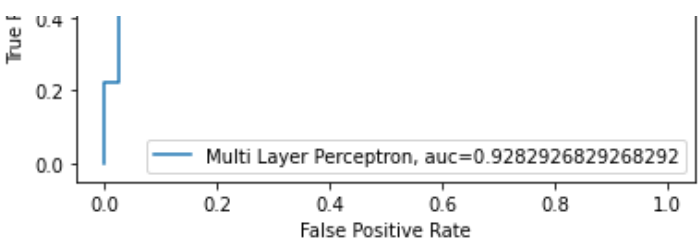




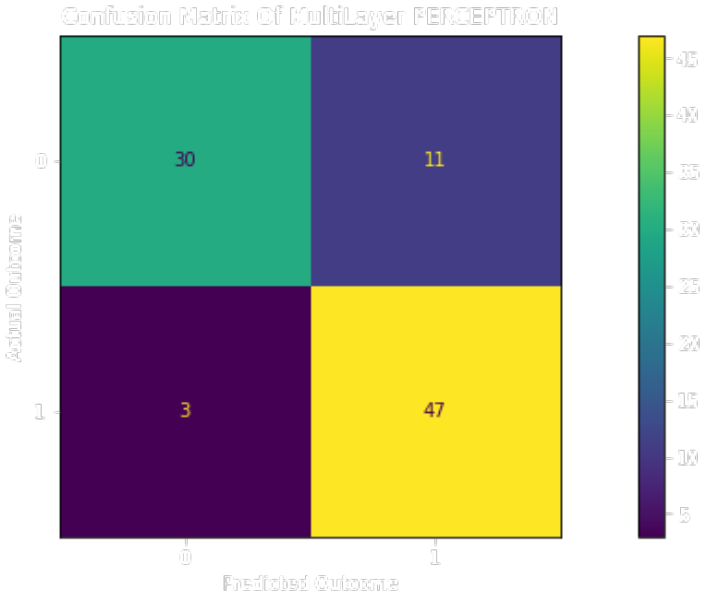
In [45]:

```
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()
```

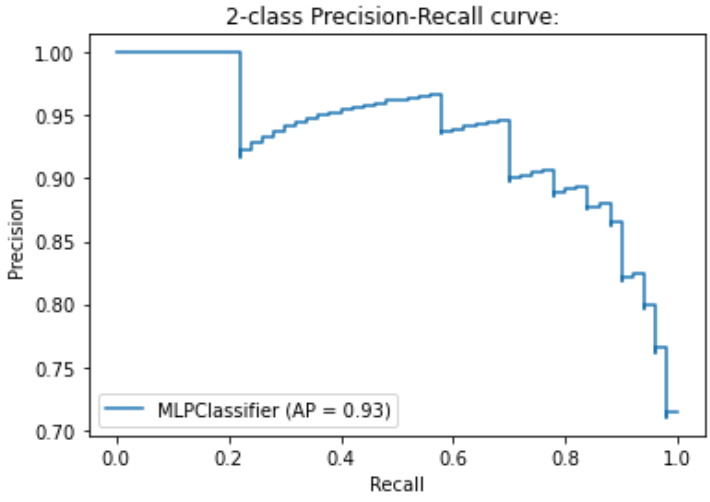





```
Confusion Matrix of MLPClassifier [[30 11]
 [ 3 47]]
Classification Report                precision    recall  f1-score   support

           0       0.91      0.73      0.81        41
           1       0.81      0.94      0.87        50

    accuracy                           0.85        91
   macro avg       0.86      0.84      0.84        91
weighted avg       0.85      0.85      0.84        91
```

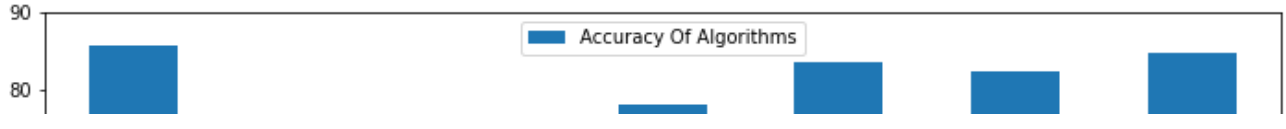


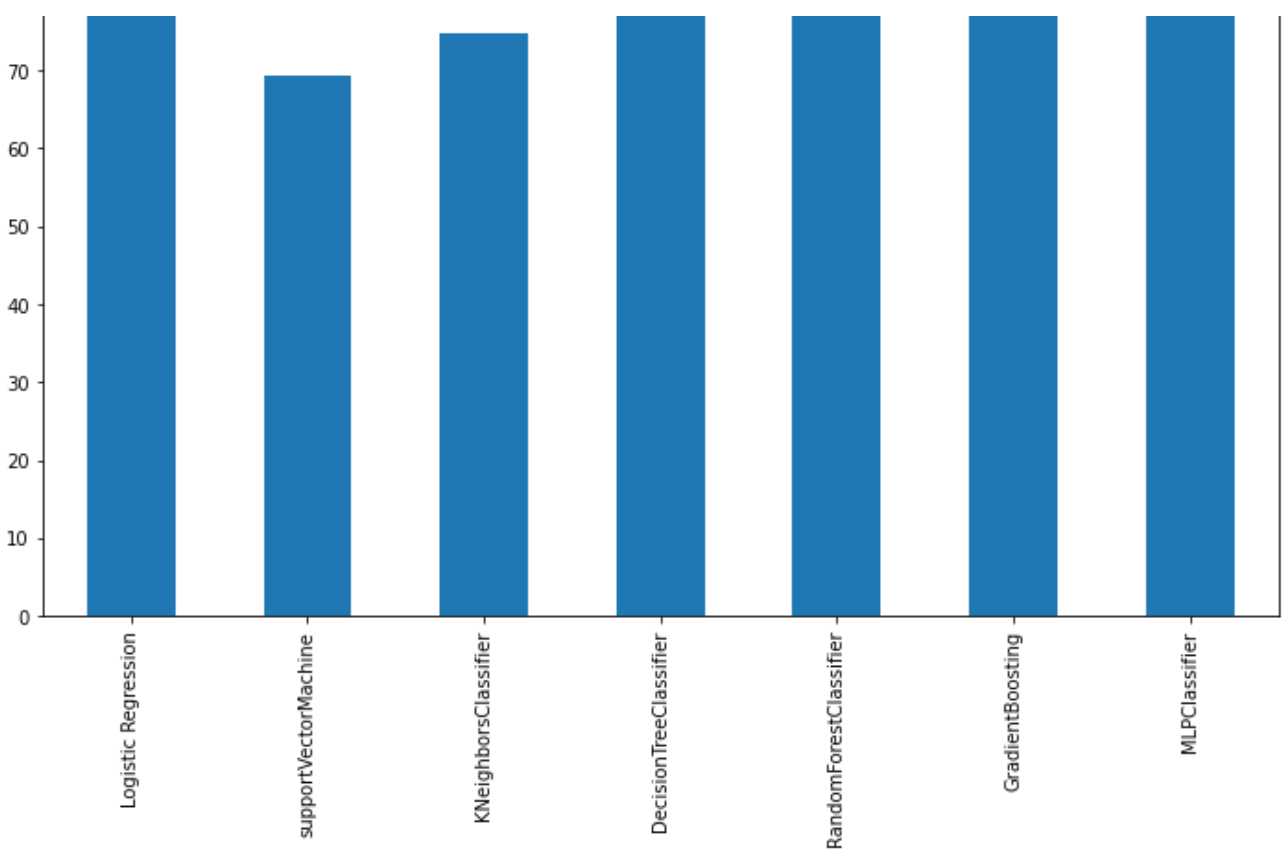

```
Accuracy score of the model is: 84.61538461538461 %
```



In [46]:

```
cardioBarChart = pd.DataFrame(accuracies_Of_Algorithms.values(),
                              accuracies_Of_Algorithms.keys(),
                              columns=["Accuracy Of Algorithms"])

cardioBarChart.plot.bar(figsize = (12,7));
```




70
60
50
40
30
20
10
0
Logistic Regression
supportVectorMachine
KNeighborsClassifier
DecisionTreeClassifier
RandomForestClassifier
GradientBoosting
MLPClassifier

In [ ]:

In [ ]:

In [ ]: