Heart Disease prediction

Introduction

</center

Library Importing

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import plot_confusion_matrix,roc_auc_score, roc_curve, f1_score, accuracy_score
from sklearn.metrics import make_scorer, precision_score, precision_recall_curve, plot_precision_recall_curve
from sklearn.metrics import recall_score, plot_roc_curve

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

In [3]:
```
sns.set_theme(context='notebook',
              style='white',
              palette='deep',
              font='Lucida Calligraphy',
              font_scale=1.5,
              color_codes=True,
              rc=None)

import matplotlib

plt.rcParams['figure.figsize'] = (14,8)
plt.rcParams['figure.facecolor'] = '#F0F8FF'
plt.rcParams['figure.titlesize'] = 'medium'
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.edgecolor'] = 'green'
plt.rcParams['figure.frameon'] = True

plt.rcParams["figure.autolayout"] = True

plt.rcParams['axes.facecolor'] = '#F5F5DC'
plt.rcParams['axes.titlesize'] = 25
plt.rcParams["axes.titleweight"] = 'normal'
plt.rcParams["axes.titlecolor"] = 'Olive'
```

```
plt.rcParams['axes.edgecolor'] = 'pink'
plt.rcParams["axes.linewidth"] = 2
plt.rcParams["axes.grid"] = True
plt.rcParams['axes.titlelocation'] = 'center'
plt.rcParams["axes.labelsize"] = 20
plt.rcParams["axes.labelpad"] = 2
plt.rcParams['axes.labelweight'] = 1
plt.rcParams["axes.labelcolor"] = 'Olive'
plt.rcParams["axes.axisbelow"] = False
plt.rcParams['axes.xmargin'] = .2
plt.rcParams["axes.ymargin"] = .2


plt.rcParams["xtick.bottom"] = True
plt.rcParams['xtick.color'] = '#A52A2A'
plt.rcParams["ytick.left"] = True
plt.rcParams['ytick.color'] = '#A52A2A'

plt.rcParams['axes.grid'] = True
plt.rcParams['grid.color'] = 'green'
plt.rcParams['grid.linestyle'] = '--'
plt.rcParams['grid.linewidth'] = .5
plt.rcParams['grid.alpha'] = .3

plt.rcParams['legend.loc'] = 'best'
plt.rcParams['legend.facecolor'] =  'NavajoWhite'
plt.rcParams['legend.edgecolor'] = 'pink'
plt.rcParams['legend.shadow'] = True
plt.rcParams['legend.fontsize'] = 20


plt.rcParams['font.family'] = 'Lucida Calligraphy'
plt.rcParams['font.size'] = 14

plt.rcParams['figure.dpi'] = 200
plt.rcParams['figure.edgecolor'] = 'Blue'
```

In [4]:
```
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
pd.set_option("display.precision", 2)
```

In [5]:
```
heart1 = pd.read_csv('../input/heart-failure-prediction/heart.csv')
heart1.head().style.set_properties(**{'background-color': 'Olive',
                           'color': 'white',
                           'border-color': 'darkblack'})
```

Out[5]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.00	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.00	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.00	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.50	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.00	Up	0

Attributes:

- **Age:** Age of the patient [years]
- **Sex:** Sex of the patient [M: Male, F: Female]
- **ChestPainType:** Chest Pain Type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- **RestingBP:** Resting blood pressure [mm Hg]
- **Cholesterol:** Serum cholesterol [mm/dl]
- **FastingBS:** Fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- **RestingECG**: Resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- **MaxHR:** Maximum heart rate achieved [Numeric value between 60 and 202]
- **ExerciseAngina:** Exercise-induced angina [Y: Yes, N: No]
- **Oldpeak:** Oldpeak = ST [Numeric value measured in depression]
- **ST_Slope:** The slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

- **HeartDisease:** Output class [1: heart disease, 0: Normal]

In [6]:
```
display(heart1.shape)
display(heart1.size)
```

```
(918, 12)
11016
```

In [7]:
```
heart1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [8]:
```
heart1.describe()
```

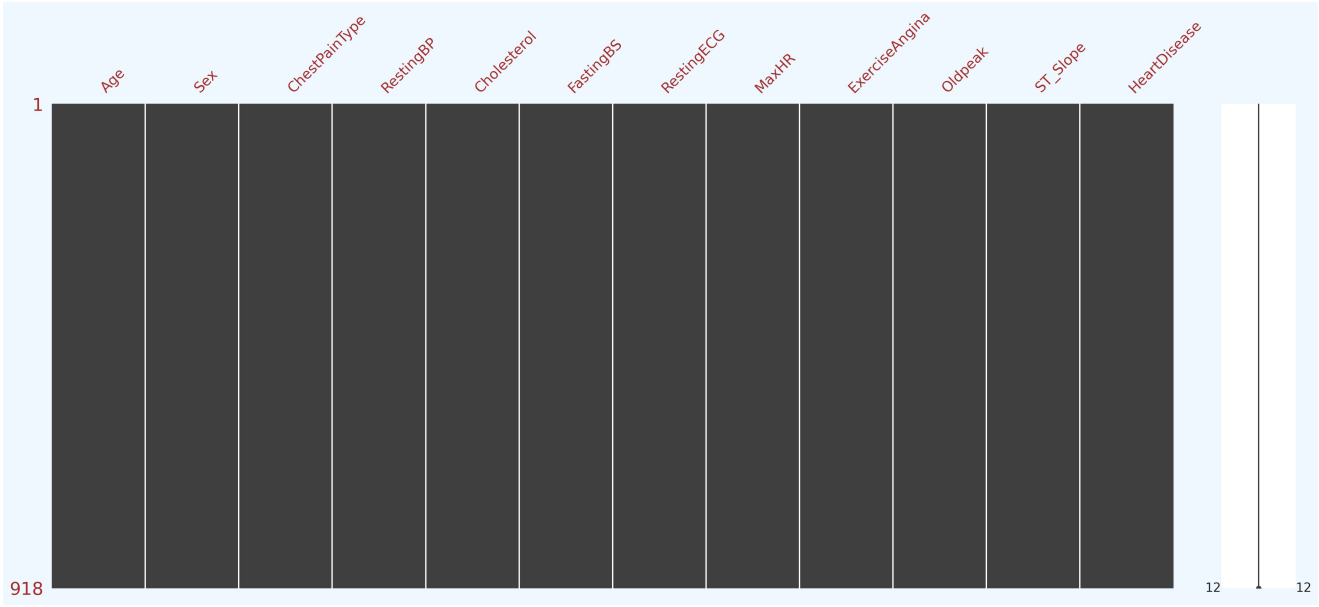
Out[8]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.00	918.00	918.00	918.00	918.00	918.00	918.00
mean	53.51	132.40	198.80	0.23	136.81	0.89	0.55
std	9.43	18.51	109.38	0.42	25.46	1.07	0.50
min	28.00	0.00	0.00	0.00	60.00	-2.60	0.00
25%	47.00	120.00	173.25	0.00	120.00	0.00	0.00
50%	54.00	130.00	223.00	0.00	138.00	0.60	1.00
75%	60.00	140.00	267.00	0.00	156.00	1.50	1.00
max	77.00	200.00	603.00	1.00	202.00	6.20	1.00

In [9]:
```
heart1.isnull().mean()*100
```

Out[9]:
```
Age               0.0
Sex               0.0
ChestPainType     0.0
RestingBP         0.0
Cholesterol       0.0
FastingBS         0.0
RestingECG        0.0
MaxHR             0.0
ExerciseAngina    0.0
Oldpeak           0.0
ST_Slope          0.0
HeartDisease      0.0
dtype: float64
```

In [10]:
```
import missingno as msno
msno.matrix(heart1)
plt.show()
```



In [11]:
```
Women = heart1.loc[heart1['Sex'] == 'F']["HeartDisease"]
rate_women = (Women.sum()/len(Women)).round(2)*100
print("Percentage of Women with probability of HeartDisease:", rate_women,"%")

Men = heart1.loc[heart1['Sex'] == 'M']["HeartDisease"]
rate_men = (Men.sum()/len(Men)).round(2)*100
print("Percentage of Men with probability of HeartDisease  :", rate_men,"%")
```

```
Percentage of Women with probability of HeartDisease: 26.0 %
Percentage of Men with probability of HeartDisease  : 63.0 %
```

In [12]:
```
print(f'We have {heart1.shape[0]} instances with the {heart1.shape[1]-1} features and 1 output variable')
```

```
We have 918 instances with the 11 features and 1 output variable
```

In [13]:
```
pd.DataFrame(abs(heart1.corr()['HeartDisease']))
```

Out[13]:

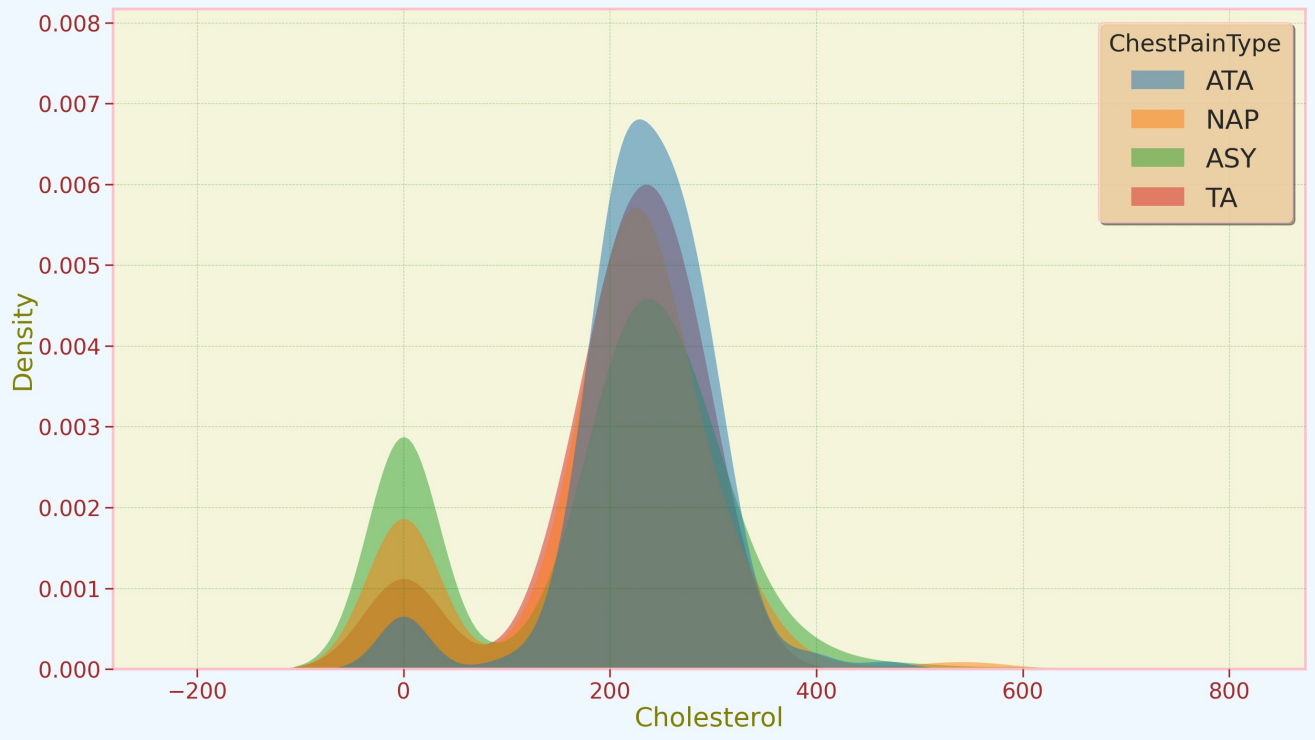
	HeartDisease
Age	0.28
RestingBP	0.11
Cholesterol	0.23
FastingBS	0.27
MaxHR	0.40
Oldpeak	0.40
HeartDisease	1.00

In [14]:
```
## Combining Data
heart1.agg(
    {
        "Age": ["min", "max", "median","mean", "skew", 'std'],
         "RestingBP": ["min", "max", "median", "mean","skew",'std'],
         "Cholesterol": ["min", "max", "median", "mean","skew",'std'],
         "Oldpeak": ["min", "max", "median", "mean","skew",'std'],
         "MaxHR": ["min", "max", "median", "mean","skew",'std']
    }
)
```

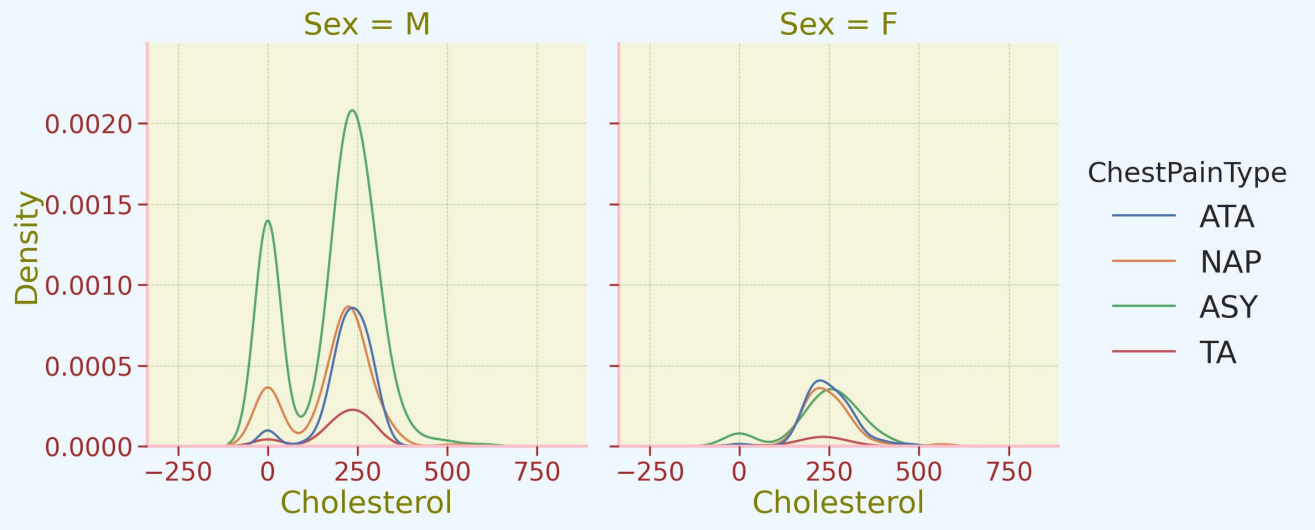
Out[14]:

	Age	RestingBP	Cholesterol	Oldpeak	MaxHR
min	28.00	0.00	0.00	-2.60	60.00
max	77.00	200.00	603.00	6.20	202.00
median	54.00	130.00	223.00	0.60	138.00
mean	53.51	132.40	198.80	0.89	136.81
skew	-0.20	0.18	-0.61	1.02	-0.14
std	9.43	18.51	109.38	1.07	25.46

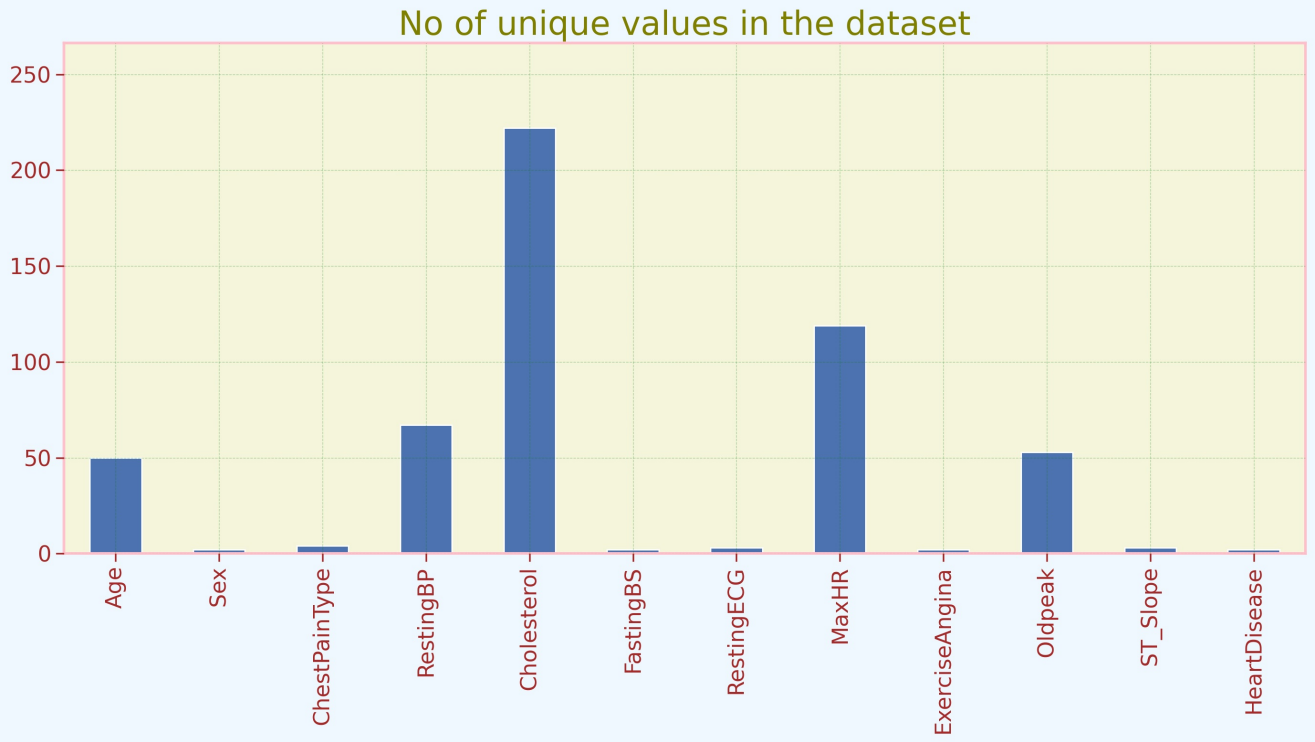
In [15]:
```
sns.kdeplot( data=heart1, x="Cholesterol", hue="ChestPainType", fill=True, common_norm=False, palette="tab10"
```



In [16]:
```
sns.displot(data=heart1, x="Cholesterol", hue="ChestPainType", col="Sex", kind="kde");
```



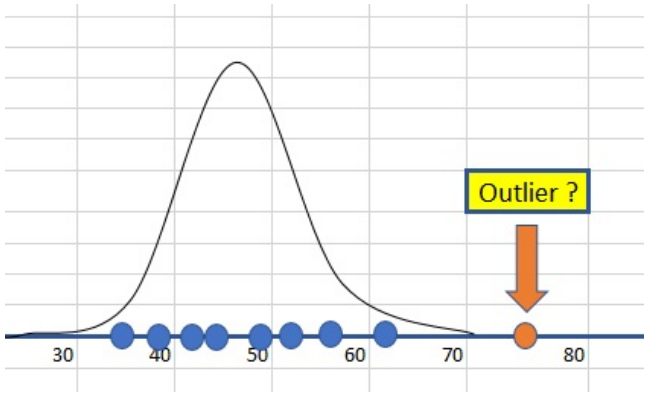
In [17]:
```
heart1.nunique().plot(kind='bar')
plt.title('No of unique values in the dataset')
plt.show()
```



Outliers in Data

Outliers in Data

Outliers are unusual values in your dataset, and they can distort statistical analyses and violate their assumptions. Unfortunately, all analysts will confront outliers and be forced to make decisions about what to do with them. Given the problems they can cause, you might think that it's best to remove them from your data. But, that's not always the case. Removing outliers is legitimate only for specific reasons.

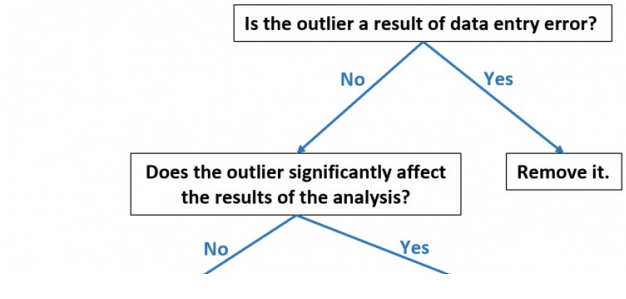


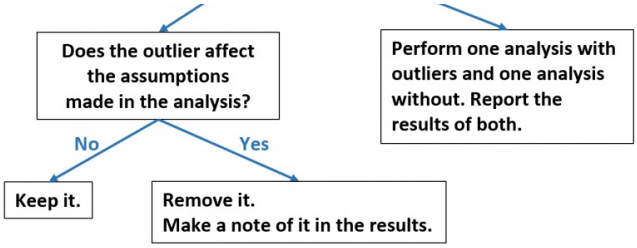
</center

When to Remove Outliers in Data

- An outlier is an observation that lies abnormally far away from other values in a dataset.
- Outliers can be problematic because they can affect the results of an analysis.
- However, they can also be informative about the data you're studying because they can reveal abnormal cases or individuals that have rare traits.
- In any analysis, you must decide to remove or keep outliers.

Fortunately, you can use the following flow chart to help you decide:





</center

Different outlier detection technique.

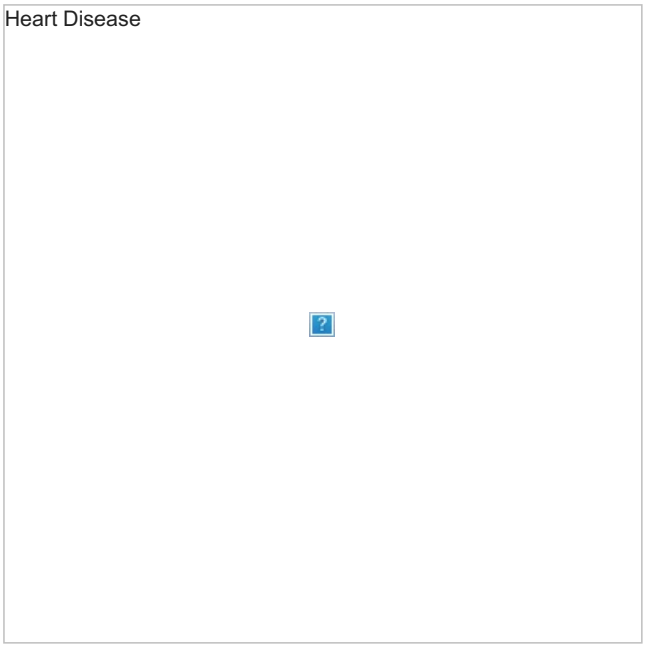
- 1. Hypothesis Testing
- 1. Z-score method
- 1. Robust Z-score
- 1. I.Q.R method
- 1. Winsorization method(Percentile Capping)
- 1. DBSCAN Clustering
- 1. Isolation Forest
- 1. Visualizing the data

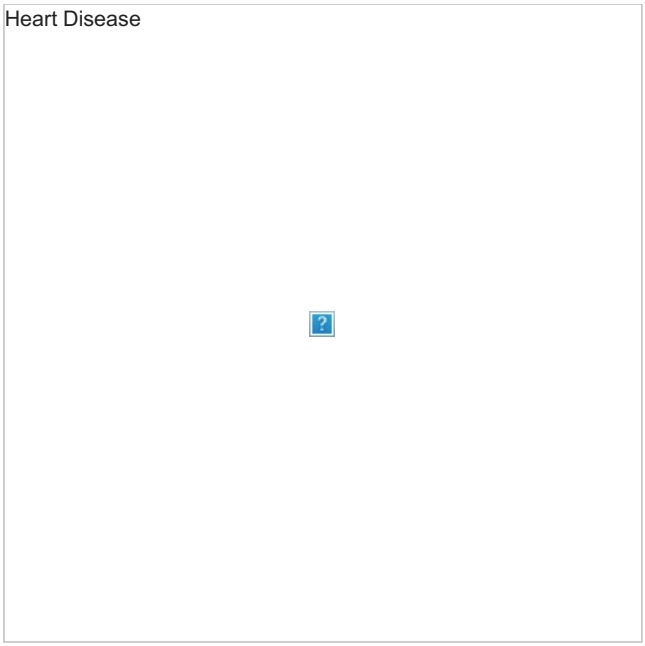
IQR method is used in this project

- In this method by using Inter Quartile Range(IQR), we detect outliers. IQR tells us the variation in the data set.Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR treated as outliers

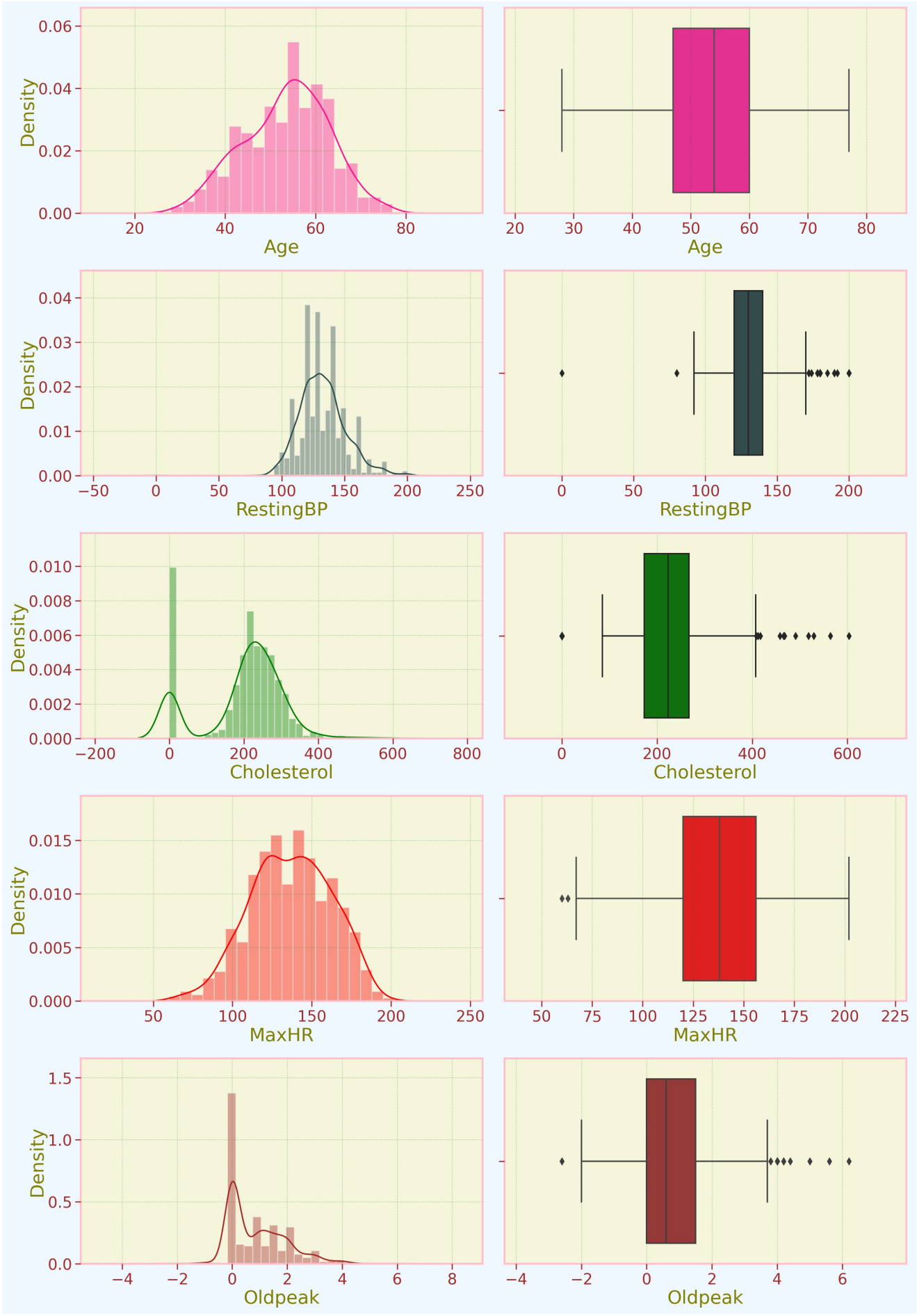
Q1 represents the 1st quartile/25th percentile of the data. Q2 represents the 2nd quartile/median/50th percentile of the data. Q3 represents the 3rd quartile/75th percentile of the data.

- (Q1–1.5*IQR) represent the smallest value in the data set and
- (Q3+1.5*IQR) represnt the largest value in the data set.





In [18]:
```
plt.figure(figsize=(14,20))

plt.subplot(5,2,1)
sns.distplot(heart1['Age'],color='DeepPink')
plt.subplot(5,2,2)
sns.boxplot(heart1['Age'],color='DeepPink')

plt.subplot(5,2,3)
sns.distplot(heart1['RestingBP'],color='DarkSlateGray')
plt.subplot(5,2,4)
sns.boxplot(heart1['RestingBP'],color='DarkSlateGray')

plt.subplot(5,2,5)
sns.distplot(heart1['Cholesterol'],color='Green')
plt.subplot(5,2,6)
sns.boxplot(heart1['Cholesterol'],color='Green')

plt.subplot(5,2,7)
sns.distplot(heart1['MaxHR'],color='Red')
plt.subplot(5,2,8)
sns.boxplot(heart1['MaxHR'],color='Red')

plt.subplot(5,2,9)
sns.distplot(heart1['Oldpeak'],color='Brown')
plt.subplot(5,2,10)
sns.boxplot(heart1['Oldpeak'],color='Brown')

plt.tight_layout()
plt.show()
```

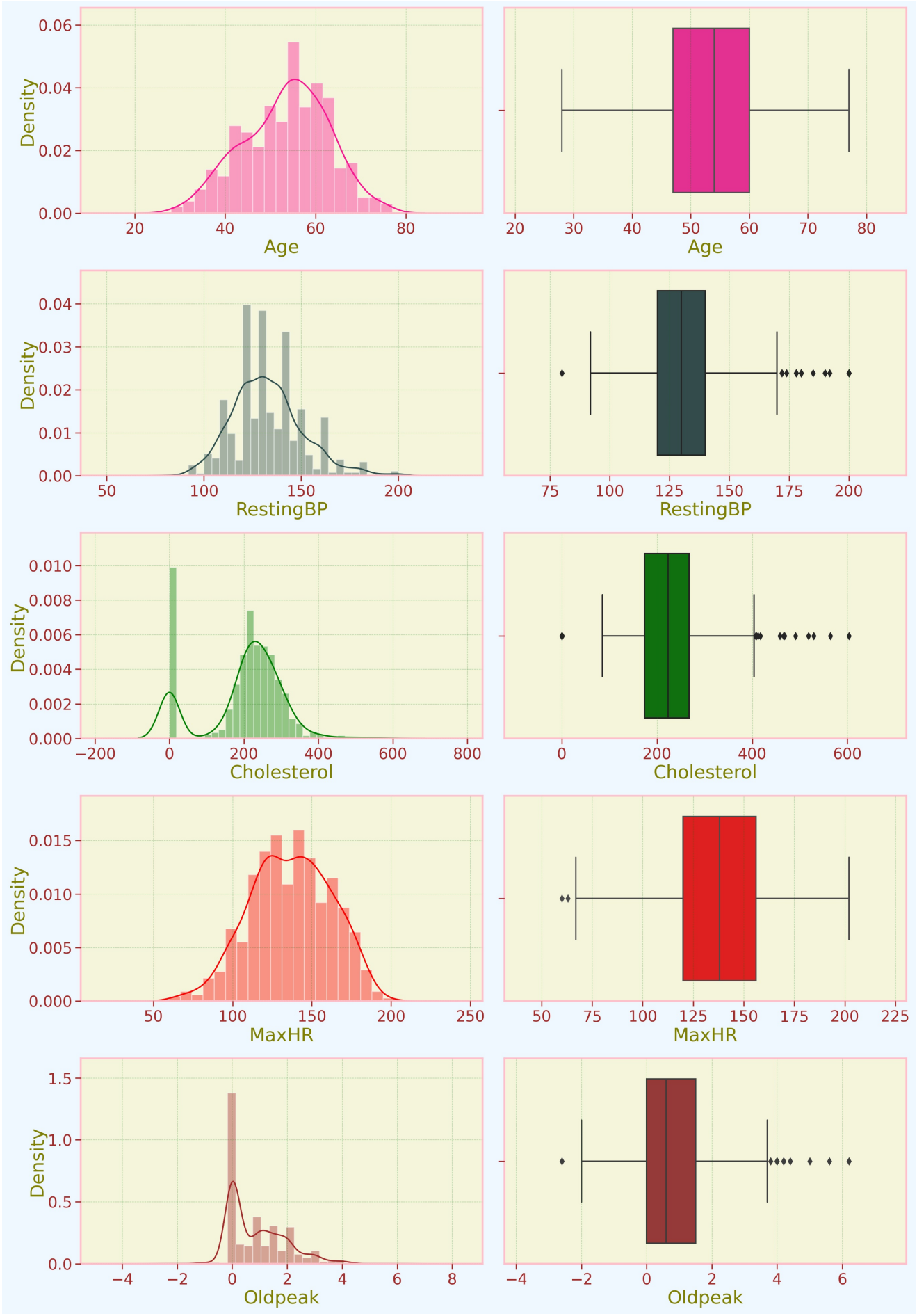


In [19]:
```
Q1 = heart1.quantile(0.05)
Q3 = heart1.quantile(0.95)
IQR = Q3 - Q1
IQR.sort_values(ascending=False)
```

Out[19]:
```
Cholesterol     331.3
MaxHR            82.0
RestingBP        54.0
Age              31.0
Oldpeak           3.0
FastingBS         1.0
HeartDisease      1.0
dtype: float64
```

In [20]:
```
heart = heart1 [~((heart1 < (Q1 - 1.5 * IQR)) |(heart1 > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [21]:
```
plt.figure(figsize=(14,20))

plt.subplot(5,2,1)
sns.distplot(heart['Age'],color='DeepPink')
plt.subplot(5,2,2)
sns.boxplot(heart['Age'],color='DeepPink')

plt.subplot(5,2,3)
sns.distplot(heart['RestingBP'],color='DarkSlateGray')
plt.subplot(5,2,4)
sns.boxplot(heart['RestingBP'],color='DarkSlateGray')

plt.subplot(5,2,5)
sns.distplot(heart['Cholesterol'],color='Green')
plt.subplot(5,2,6)
sns.boxplot(heart['Cholesterol'],color='Green')

plt.subplot(5,2,7)
sns.distplot(heart['MaxHR'],color='Red')
plt.subplot(5,2,8)
sns.boxplot(heart['MaxHR'],color='Red')

plt.subplot(5,2,9)
sns.distplot(heart['Oldpeak'],color='Brown')
plt.subplot(5,2,10)
sns.boxplot(heart['Oldpeak'],color='Brown')

plt.tight_layout()
plt.show()
```



In [22]:
```
# Checking the number of Zeros(0) present in the Cholesterol
Cholesterol = heart[heart['Cholesterol'] == 0]
Cholesterol.shape
```

Out[22]:
```
(171, 12)
```

Cholesterol value cannat be zero , So zeros must be filled with median value

In [23]:
```
## Converting Zero Value of Cholesterol to NaN Value
heart.loc[heart['Cholesterol'] == 0,'Cholesterol'] = np.nan
```

```
heart.head(1)
```

Out[23]:

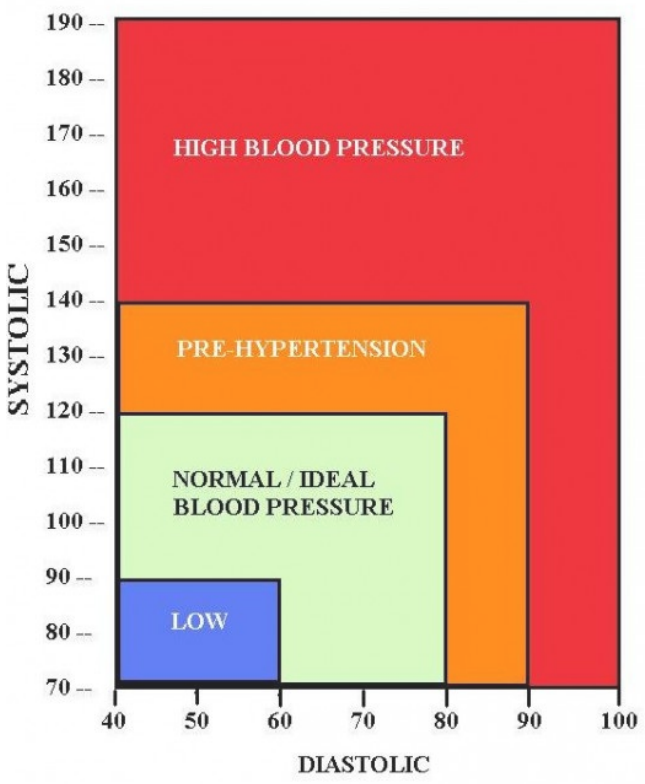
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289.0	0	Normal	172	N	0.0	Up	0

In [24]:
```
## NaN value of Cholesterol is filled with median value
heart["Cholesterol"] = heart["Cholesterol"].fillna(heart["Cholesterol"].median())
```

RestingBP:

A total blood pressure reading is determined by measuring the systolic and diastolic blood pressures.

- Systolic blood pressure, the top number, measures the force the heart exerts on the walls of the arteries each time it beats.
- CrossDiastolic blood pressure, the bottom number, measures the force the heart exerts on the walls of the arteries in between beats.



- In this dataset there is no seprate columns are provided for Systolic and Diastolic Blood Pressure

In [25]:
```
## Checking the presence of Zeros in the RestingBP
RestingBP = heart[heart['RestingBP'] == 0]
RestingBP
```

Out[25]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease

RestingBP represents the blood pressure of the patient. It is not possible to have values equal to Zero(0).There is only one Zero in RestingBP So I'll remove the value Zero(0)

In [26]:
```
heart = heart.drop(heart[(heart['RestingBP'] == 0)].index)
```

In [27]:
```
Systolic_RestingBP = heart[heart['RestingBP'] >= 100]
display(Systolic_RestingBP.value_counts().sum())
```

```
905
```

In RestingBP, number Systolic blood pressure of the patient is 905, this is calculated from the range of Systolic which is from 70 to 190 and above

In [28]:
```
Diastolic_RestingBP = heart[heart['RestingBP'] <= 100]
display(Diastolic_RestingBP.value_counts().sum())
```

In RestingBP, number Diastolic blood pressure of the patient is only 27, this is calculated from the range of Diastolic which is from 40 to 100

In [29]:
```
print(Systolic_RestingBP.shape)
print(Diastolic_RestingBP.shape)
```

```
(905, 12)
(27, 12)
```

In RestingBP, data is totally unbalanced because for a person both Systolic & Diastolic Blood Pressure is measured at a same time. So it is concluded that RestingBP data set is only of Systolic Blood Pressure

In [30]:
```
heart['Diastolic_RestingBP'] = np.where( heart['RestingBP'] <=100, 1,0)
heart.sample(5)
```

Out[30]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	Hear
482	67	M	TA	142	270.0	1	Normal	125	N	2.5	Up	
500	65	M	ASY	136	248.0	0	Normal	140	Y	4.0	Down	
169	58	M	NAP	140	179.0	0	Normal	160	N	0.0	Up	
571	69	M	ASY	140	110.0	1	Normal	109	Y	1.5	Flat	
16	38	M	ASY	110	196.0	0	Normal	166	N	0.0	Flat	

In [31]:
```
heart['RestingBP'].min()
```

Out[31]:
```
80
```

In [32]:
```
heart["RestingBP"].max()
```

Out[32]:
```
200
```

In [33]:
```
heart["RestingBP_Category"]= pd.cut(heart["RestingBP"] ,bins=[0,120, 129 , 139,200] ,labels=["Normal_BP'
print("Value Counts :\n\n",heart['RestingBP_Category'].value_counts())

heart.sample(5)
```

```
Value Counts :

 Hypertension_Stage_2    327
Normal_BP                292
Hypertension_Stage_1     216
Elevated_BP               82
Name: RestingBP_Category, dtype: int64
```

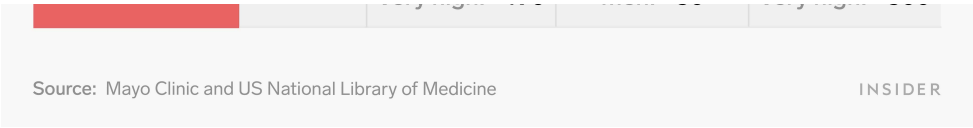
Out[33]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	Hear
879	48	M	ATA	110	229.0	0	Normal	168	N	1.0	Down	
845	61	F	ASY	145	307.0	0	LVH	146	Y	1.0	Flat	
463	51	M	ASY	128	237.0	0	Normal	107	N	0.0	Up	
279	54	F	ATA	130	253.0	0	ST	155	N	0.0	Up	
234	54	F	ATA	140	309.0	0	ST	140	N	0.0	Up	

In [34]:
```
# converting from categorical variable to object type
heart['RestingBP_Category'] = heart['RestingBP_Category'].astype(object)
```

Cholesterol levels for adults, ages 20 and over

AMOUNT (mg/dL)	TOTAL	LDL	HDL	TRIGLYCERIDES
Ideal	<200	<100	>60	<150
Borderline	200–239	130–159	Women: 40–59 Men: 50–59	150–199
Too high or low	>240	High: 160–189 Very high: >190	Women: <40 Men: <50	High: 200–499 Very high: >500



In [35]:
```
heart['Age'].min()
```

Out[35]:
```
28
```

In [36]:
```
heart["Cholesterol_Category"]= pd.cut(heart["Cholesterol"] ,bins=[0, 200, 230 , 500] ,labels=["Normal",'
print("Value Counts :\n\n",heart['Cholesterol_Category'].value_counts())

heart.head()
```

```
Value Counts :

 High          576
Borderline    187
Normal        150
Name: Cholesterol_Category, dtype: int64
```

Out[36]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartD
0	40	M	ATA	140	289.0	0	Normal	172	N	0.0	Up	
1	49	F	NAP	160	180.0	0	Normal	156	N	1.0	Flat	
2	37	M	ATA	130	283.0	0	ST	98	N	0.0	Up	
3	48	F	ASY	138	214.0	0	Normal	108	Y	1.5	Flat	
4	54	M	NAP	150	195.0	0	Normal	122	N	0.0	Up	

In [37]:
```
heart.dtypes
```

Out[37]:
```
Age                      int64
Sex                     object
ChestPainType           object
RestingBP                int64
Cholesterol            float64
FastingBS                int64
RestingECG              object
MaxHR                    int64
ExerciseAngina          object
Oldpeak                float64
ST_Slope                object
HeartDisease             int64
Diastolic_RestingBP      int64
RestingBP_Category      object
Cholesterol_Category    category
dtype: object
```

In [38]:
```
# converting from categorical variable to object type
heart['Cholesterol_Category'] = heart['Cholesterol_Category'].astype(object)
```

In [39]:
```
heart.dtypes
```

Out[39]:
```
Age                      int64
Sex                     object
ChestPainType           object
RestingBP                int64
Cholesterol            float64
FastingBS                int64
RestingECG              object
MaxHR                    int64
ExerciseAngina          object
Oldpeak                float64
ST_Slope                object
HeartDisease             int64
Diastolic_RestingBP      int64
RestingBP_Category      object
Cholesterol_Category    object
dtype: object
```

In [40]:
```
heart['Cholesterol_Category'].value_counts()
```

Out[40]:
```
High          576
Borderline    187
Normal        150
Name: Cholesterol_Category, dtype: int64
```

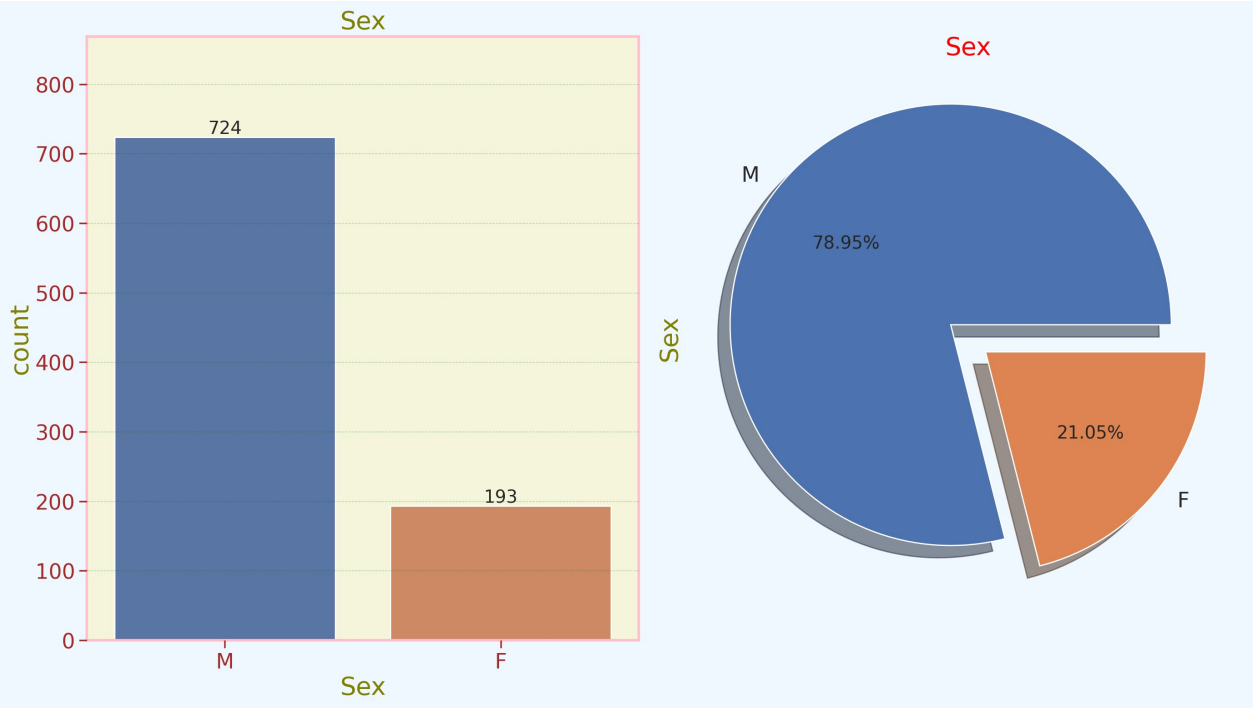
In [41]:
```
numerical = heart.select_dtypes(include=['number']).columns
categorical = heart.select_dtypes(include=['object']).columns

print('Numerical  :',numerical)
print("**********"*10)
print("Categorical:",categorical)
```

```
Numerical  : Index(['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak',
       'HeartDisease', 'Diastolic_RestingBP'],
      dtype='object')
**************************************************************************************************
Categorical: Index(['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope',
       'RestingBP_Category', 'Cholesterol_Category'],
      dtype='object')
```

In [42]:
```
for col in heart[categorical]:
    print(heart[col].value_counts())
    print("****"*7)
```

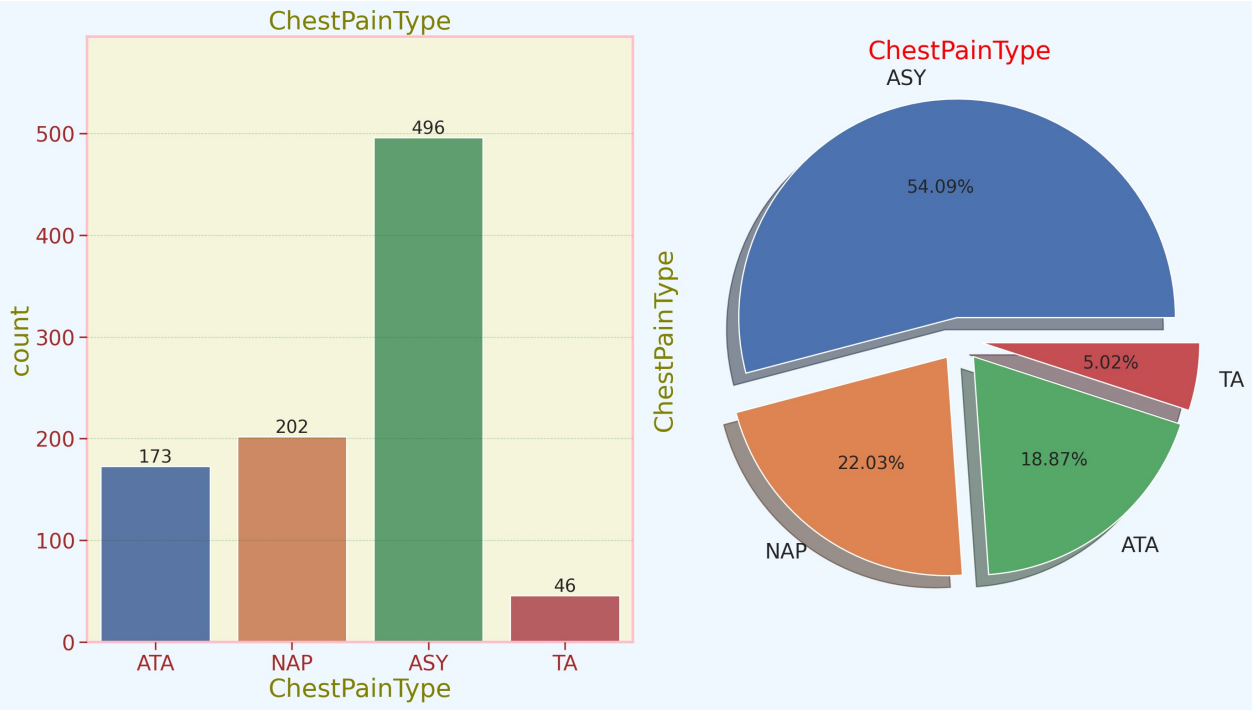
```
M    724
F    193
Name: Sex, dtype: int64
****************************
ASY    496
NAP    202
ATA    173
TA      46
Name: ChestPainType, dtype: int64
****************************
Normal    551
LVH       188
ST        178
Name: RestingECG, dtype: int64
****************************
N    546
Y    371
Name: ExerciseAngina, dtype: int64
****************************
Flat    459
Up      395
Down     63
Name: ST_Slope, dtype: int64
****************************
Hypertension_Stage_2    327
Normal_BP               292
Hypertension_Stage_1    216
Elevated_BP              82
Name: RestingBP_Category, dtype: int64
****************************
High          576
Borderline    187
Normal        150
Name: Cholesterol_Category, dtype: int64
****************************
```

In [43]:
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='Sex', data=heart)
ax.bar_label(ax.containers[0])
plt.title("Sex", fontsize=20)
ax =plt.subplot(1,2,2)
ax=heart['Sex'].value_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "Sex", fontsize = 20,color='Red',font='Lucida Calligraphy');
```

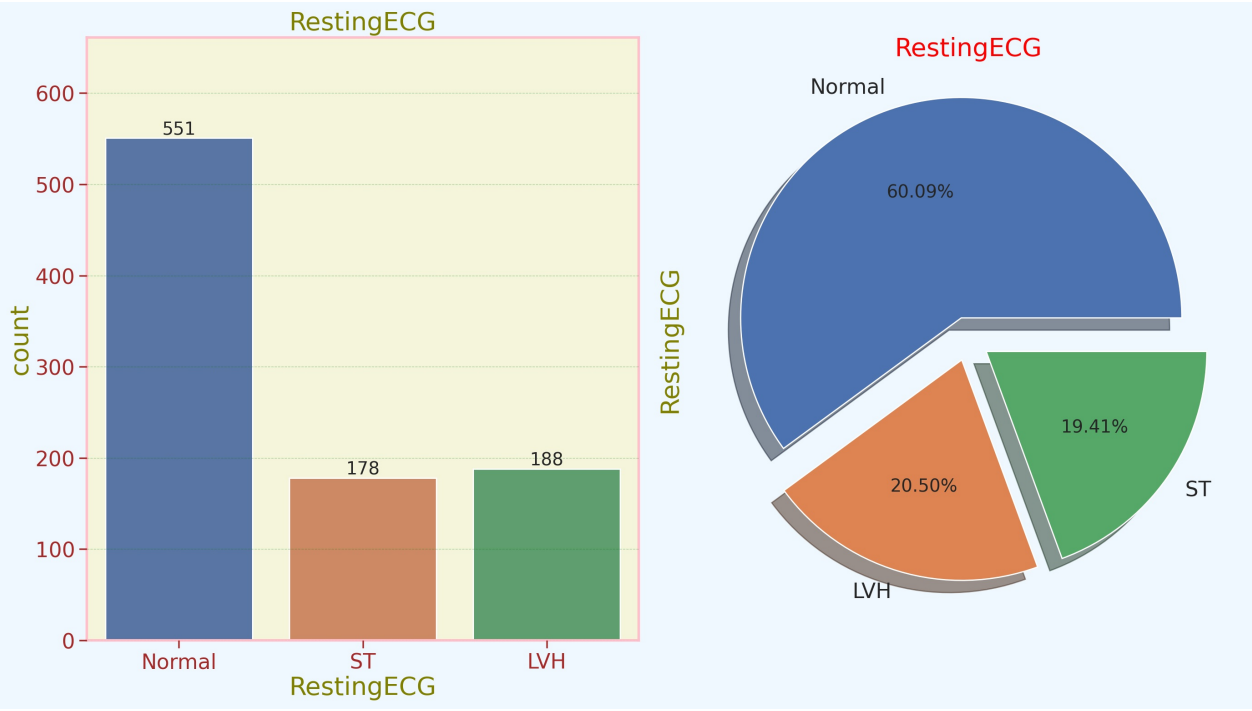


In [44]:
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='ChestPainType', data=heart)
ax.bar_label(ax.containers[0])
```

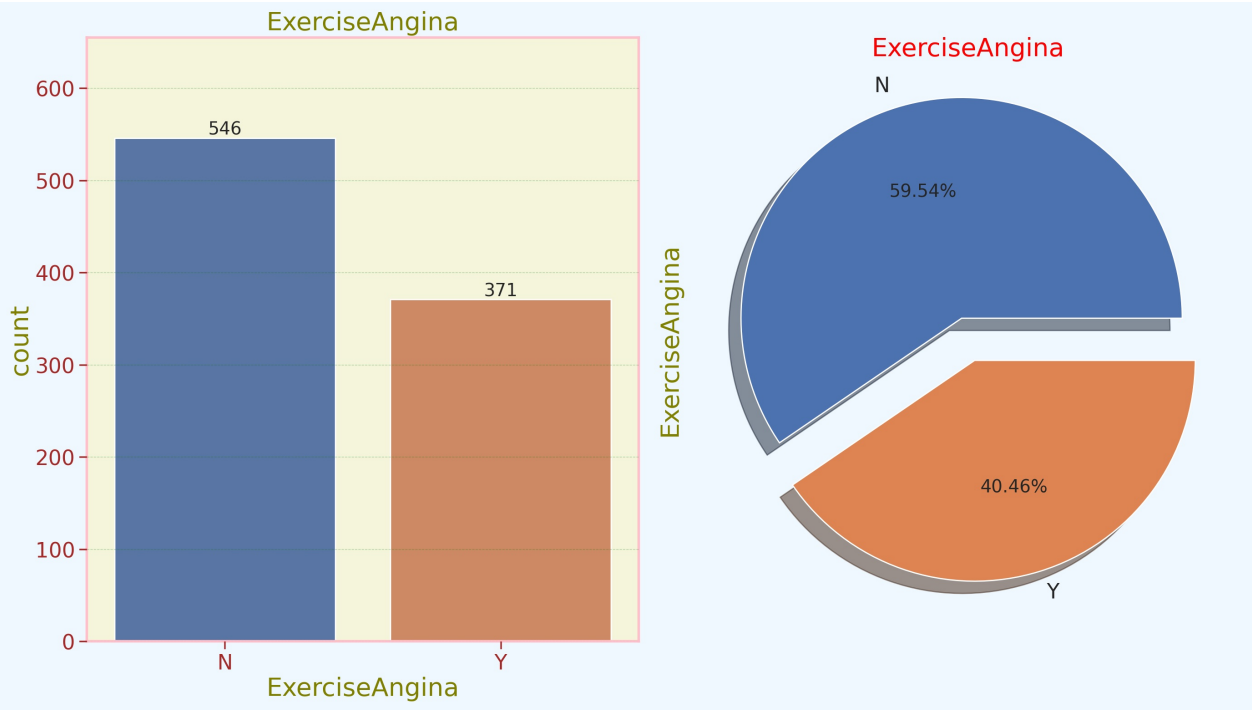
```
plt.title("ChestPainType", fontsize=20)
ax =plt.subplot(1,2,2)
ax=heart['ChestPainType'].value_counts().plot.pie(explode=[0.1, 0.1,0.1,0.1],autopct='%1.2f%%',shadow=Tr
ax.set_title(label = "ChestPainType", fontsize = 20,color='Red',font='Lucida Calligraphy');
```



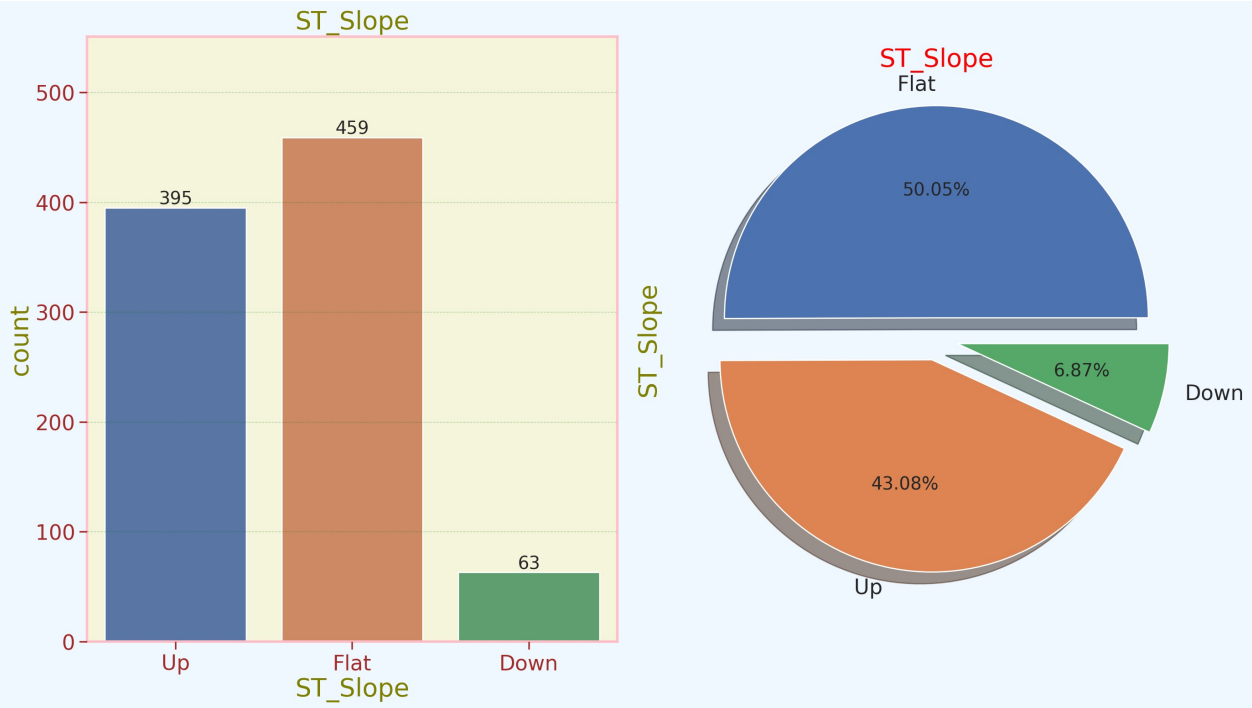
In [45]:
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='RestingECG', data=heart)
ax.bar_label(ax.containers[0])
plt.title("RestingECG", fontsize=20)

ax =plt.subplot(1,2,2)
ax=heart['RestingECG'].value_counts().plot.pie(explode=[0.1, 0.1,0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "RestingECG", fontsize = 20,color='Red',font='Lucida Calligraphy');
```



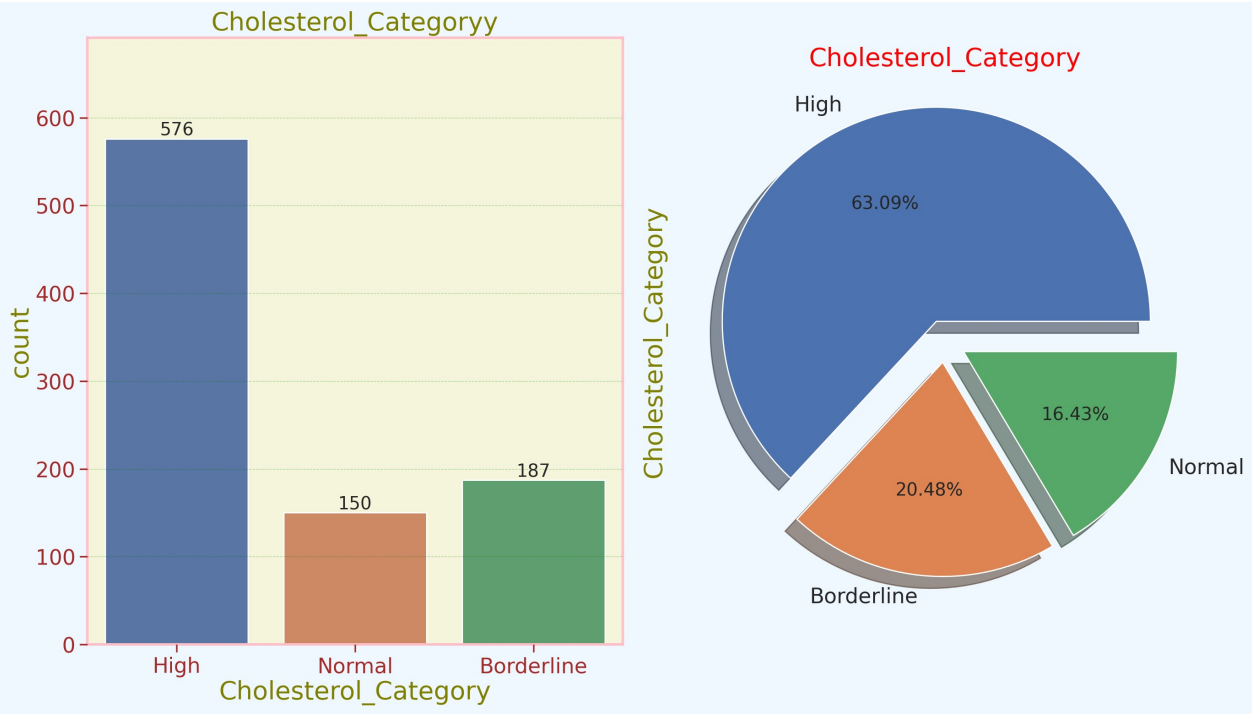
In [46]:
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='ExerciseAngina', data=heart)
ax.bar_label(ax.containers[0])
plt.title("ExerciseAngina", fontsize=20)

ax =plt.subplot(1,2,2)
ax=heart['ExerciseAngina'].value_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "ExerciseAngina", fontsize = 20,color='Red',font='Lucida Calligraphy');
```



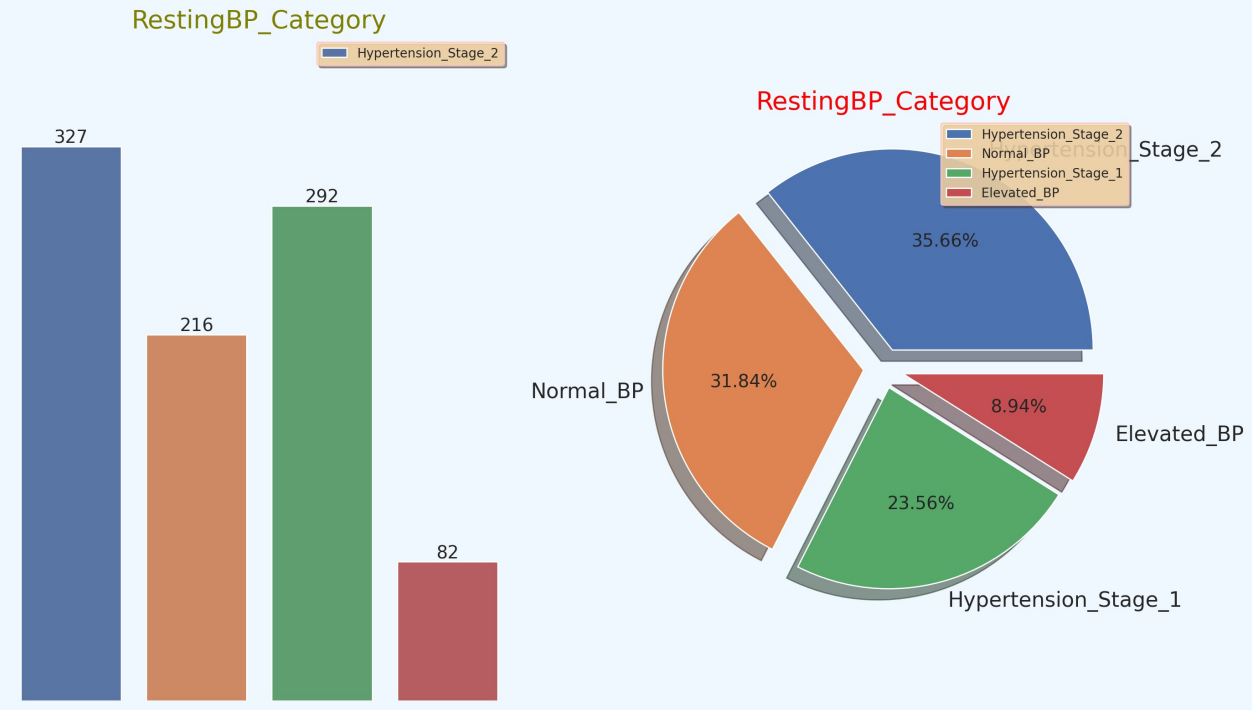
In [47]:
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='ST_Slope', data=heart)
ax.bar_label(ax.containers[0])
plt.title("ST_Slope", fontsize=20)


ax =plt.subplot(1,2,2)
ax=heart['ST_Slope'].value_counts().plot.pie(explode=[0.1, 0.1,0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "ST_Slope", fontsize = 20,color='Red',font='Lucida Calligraphy');
```



In [48]:
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='Cholesterol_Category', data=heart)
ax.bar_label(ax.containers[0])
plt.title("Cholesterol_Categoryy", fontsize=20)

ax =plt.subplot(1,2,2)
ax=heart['Cholesterol_Category'].value_counts().plot.pie(explode=[0.1, 0.1,0.1],autopct='%1.2f%%',shadow
ax.set_title(label = "Cholesterol_Category", fontsize = 20,color='Red',font='Lucida Calligraphy');
```



In [49]:
```
plt.rcParams['legend.fontsize'] = 10

ax = plt.subplot(1,2,1)
ax = sns.countplot(x='RestingBP_Category', data=heart)
ax.bar_label(ax.containers[0])
plt.title("RestingBP_Category", fontsize=20)
plt.legend(labels=['M','F'])
plt.legend(labels=['Hypertension_Stage_2','Normal_BP','Hypertension_Stage_1','Elevated_BP '])
plt.axis('off');

ax =plt.subplot(1,2,2)
ax=heart['RestingBP_Category'].value_counts().plot.pie(explode=[0.1, 0.1,0.1,0.1],autopct='%1.2f%%',shad
ax.set_title(label = "RestingBP_Category", fontsize = 20,color='Red',font='Lucida Calligraphy');
plt.legend(labels=['Hypertension_Stage_2','Normal_BP','Hypertension_Stage_1','Elevated_BP '])
plt.axis('off');
```

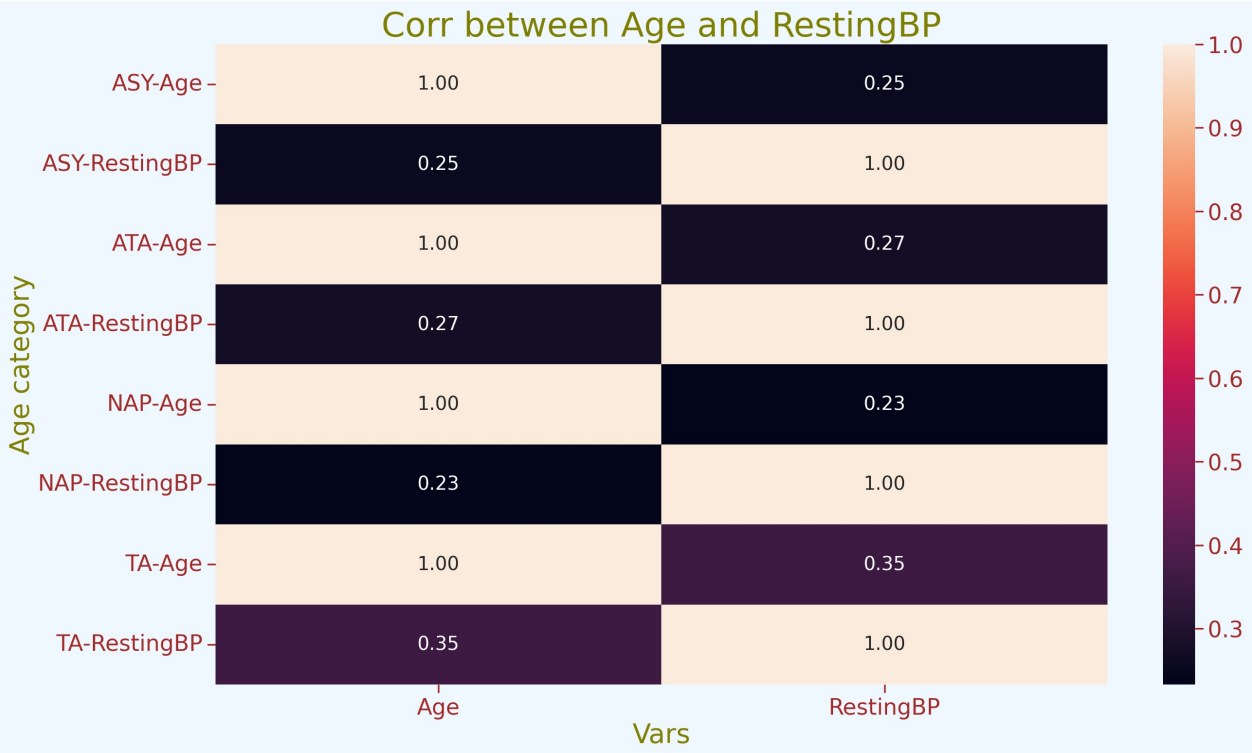


In [50]:
```
corr = heart.groupby(["ChestPainType"])[["Age","RestingBP"]].corr()
corr.head()
```

Out[50]:

ChestPainType		Age	RestingBP
ASY	Age	1.00	0.25
	RestingBP	0.25	1.00
ATA	Age	1.00	0.27
	RestingBP	0.27	1.00
NAP	Age	1.00	0.23

In [51]:
```
sns.heatmap(corr , annot=True , fmt =".2f")
plt.xlabel("Vars")
plt.ylabel("Age category")
plt.title("Corr between Age and RestingBP")
plt.show()
```



In [52]:
```
# Displaying all correlations with ascending sort.
heart.corr().unstack().sort_values(ascending=False).drop_duplicates()
```

Out[52]:
```
Age                 Age                   1.00e+00
Oldpeak             HeartDisease          4.04e-01
HeartDisease        Age                   2.82e-01
FastingBS           HeartDisease          2.68e-01
RestingBP           Age                   2.63e-01
Oldpeak             Age                   2.59e-01
Age                 FastingBS             1.98e-01
Oldpeak             RestingBP             1.74e-01
HeartDisease        RestingBP             1.18e-01
Cholesterol         RestingBP             8.61e-02
HeartDisease        Cholesterol           7.63e-02
FastingBS           RestingBP             6.78e-02
Cholesterol         Oldpeak               5.46e-02
FastingBS           Oldpeak               5.31e-02
Cholesterol         Age                   4.56e-02
Diastolic_RestingBP FastingBS             4.12e-02
FastingBS           Cholesterol           2.49e-02
MaxHR               Diastolic_RestingBP   8.03e-03
HeartDisease        Diastolic_RestingBP   9.34e-04
Cholesterol         MaxHR                -1.51e-03
                    Diastolic_RestingBP  -3.96e-02
Oldpeak             Diastolic_RestingBP  -6.56e-02
Age                 Diastolic_RestingBP  -8.53e-02
MaxHR               RestingBP            -1.10e-01
FastingBS           MaxHR                -1.31e-01
Oldpeak             MaxHR                -1.61e-01
RestingBP           Diastolic_RestingBP  -3.42e-01
Age                 MaxHR                -3.82e-01
HeartDisease        MaxHR                -4.01e-01
dtype: float64
```

In [53]:
```
print(heart['ChestPainType'][heart['Sex']=='F'].value_counts(normalize=True)*100)
```

```
ASY    36.27
ATA    31.09
NAP    27.46
TA      5.18
Name: ChestPainType, dtype: float64
```

In [54]:
```
print('ChestPainType in Females:\n')
print(heart['ChestPainType'][heart['Sex']=='F'].value_counts(normalize=True)*100)
print("********"*5,"\n")
print('ChestPainType in Males:\n')
print(heart['ChestPainType'][heart['Sex']=='M'].value_counts(normalize=True)*100)
```

```
ChestPainType in Females:

ASY    36.27
ATA    31.09
NAP    27.46
TA      5.18
Name: ChestPainType, dtype: float64
****************************************

ChestPainType in Males:

ASY    58.84
NAP    20.58
ATA    15.61
TA      4.97
Name: ChestPainType, dtype: float64
```

In [55]:
```
print('ST_Slope in Females:\n')
print(heart['ST_Slope'][heart['Sex']=='F'].value_counts(normalize=True)*100)
print("********"*5,"\n")
print('ST_Slope in Males:\n')
print(heart['ST_Slope'][heart['Sex']=='M'].value_counts(normalize=True)*100)
```

```
ST_Slope in Females:

Up      57.51
Flat    38.86
Down     3.63
Name: ST_Slope, dtype: float64
****************************************

ST_Slope in Males:

Flat    53.04
Up      39.23
Down     7.73
Name: ST_Slope, dtype: float64
```

In [56]:
```
print('Heart Disease in Females:\n')
print(heart['HeartDisease'][heart['Sex']=='F'].value_counts(normalize=True)*100)
print("********"*5,"\n")
print('Heart Disease in Males:\n')
print(heart['HeartDisease'][heart['Sex']=='M'].value_counts(normalize=True)*100)
```

```
Heart Disease in Females:

0    74.09
1    25.91
Name: HeartDisease, dtype: float64
****************************************

Heart Disease in Males:

1    63.12
0    36.88
Name: HeartDisease, dtype: float64
```

In [57]:
```
print('Heart Disease in RestingECG:\n')
print(heart['RestingECG'][heart['HeartDisease']== 1 ].value_counts(normalize=True)*100)
print("********"*5,"\n")
print('Heart Disease in RestingECG:\n')
print(heart['RestingECG'][heart['HeartDisease']== 0 ].value_counts(normalize=True)*100)
```

```
Heart Disease in RestingECG:

Normal    56.02
ST        23.08
LVH       20.91
Name: RestingECG, dtype: float64
****************************************

Heart Disease in RestingECG:

Normal    65.12
LVH       20.00
ST        14.88
Name: RestingECG, dtype: float64
```

In [58]:
```
print('Heart Disease in RestingBP_Category:\n')
print(heart['RestingBP_Category'][heart['HeartDisease']== 1 ].value_counts(normalize=True)*100)
print("********"*5,"\n")
print('Heart Disease in RestingBP_Category:\n')
print(heart['RestingBP_Category'][heart['HeartDisease']== 0 ].value_counts(normalize=True)*100)
```

```
Heart Disease in RestingBP_Category:

Hypertension_Stage_2    40.43
Normal_BP               28.01
Hypertension_Stage_1    21.50
Elevated_BP             10.06
Name: RestingBP_Category, dtype: float64
****************************************

Heart Disease in RestingBP_Category:

Normal_BP               36.59
Hypertension_Stage_2    29.76
Hypertension_Stage_1    26.10
Elevated_BP              7.56
Name: RestingBP_Category, dtype: float64
```

In [59]:
```
heart_final = heart.drop(['RestingBP','Cholesterol',],axis=1)
heart_final.head()
```

Out[59]:

	Age	Sex	ChestPainType	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease	Diastolic_Restingl
0	40	M	ATA	0	Normal	172	N	0.0	Up	0	
1	49	F	NAP	0	Normal	156	N	1.0	Flat	1	
2	37	M	ATA	0	ST	98	N	0.0	Up	0	
3	48	F	ASY	0	Normal	108	Y	1.5	Flat	1	
4	54	M	NAP	0	Normal	122	N	0.0	Up	0	

Converting Categorial Variable into numerical Variable

In [60]:
```
import category_encoders as ce
encoder = ce.OrdinalEncoder(cols=['ChestPainType','RestingECG','ST_Slope','Cholesterol_Category','Restir
heart_final = encoder.fit_transform(heart_final)

heart_final.head()
```

Out[60]:

	Age	Sex	ChestPainType	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease	Diastolic_Restingl
0	40	M	1	0	1	172	N	0.0	1	0	
1	49	F	2	0	1	156	N	1.0	2	1	
2	37	M	1	0	2	98	N	0.0	1	0	
3	48	F	3	0	1	108	Y	1.5	2	1	
4	54	M	2	0	1	122	N	0.0	1	0	

In [61]:
```
import category_encoders as ce
encoder = ce.OneHotEncoder(cols=['Sex','ExerciseAngina'],drop_invariant=True)
heart_final1 = encoder.fit_transform(heart_final)

heart_final1.head()
```

Out[61]:

	Age	Sex_1	Sex_2	ChestPainType	FastingBS	RestingECG	MaxHR	ExerciseAngina_1	ExerciseAngina_2	Oldpeak	ST_Slope	I
0	40	1	0	1	0	1	172	1	0	0.0	1	
1	49	0	1	2	0	1	156	1	0	1.0	2	
2	37	1	0	1	0	2	98	1	0	0.0	1	
3	48	0	1	3	0	1	108	0	1	1.5	2	
4	54	1	0	2	0	1	122	1	0	0.0	1	

Data train test split

In [62]:
```
X = heart_final1.drop('HeartDisease',axis=1)
```

```
y = heart_final1['HeartDisease']
```

In [63]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=21)
```

Standardizing the data by using MinMaxScaler

In [64]:
```
# scaling dataset with robust scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pd.DataFrame(X_test, columns=X.columns)


display(X_train.head())
display(X_test.head())
```

	Age	Sex_1	Sex_2	ChestPainType	FastingBS	RestingECG	MaxHR	ExerciseAngina_1	ExerciseAngina_2	Oldpeak	ST_Slope
0	0.53	1.0	0.0	0.33	0.0	0.0	0.44	1.0	0.0	0.30	0.0
1	0.53	0.0	1.0	0.00	0.0	0.5	0.56	1.0	0.0	0.30	0.0
2	0.55	1.0	0.0	0.67	0.0	0.0	0.51	0.0	1.0	0.43	0.5
3	0.31	0.0	1.0	0.33	0.0	0.0	0.81	1.0	0.0	0.30	0.0
4	0.45	1.0	0.0	0.67	0.0	0.5	0.56	0.0	1.0	0.86	0.5

	Age	Sex_1	Sex_2	ChestPainType	FastingBS	RestingECG	MaxHR	ExerciseAngina_1	ExerciseAngina_2	Oldpeak	ST_Slope
0	0.59	0.0	1.0	1.00	0.0	0.0	0.27	1.0	0.0	0.41	0.5
1	0.31	1.0	0.0	0.67	0.0	0.0	0.42	0.0	1.0	0.41	0.5
2	0.24	1.0	0.0	0.67	1.0	0.0	0.65	0.0	1.0	0.41	0.5
3	0.31	1.0	0.0	0.67	0.0	1.0	0.42	0.0	1.0	0.58	0.5
4	0.61	1.0	0.0	0.00	1.0	0.0	0.35	0.0	1.0	0.52	0.5

Model Building

LogisticRegression

In [65]:
```
plt.rcParams['figure.figsize'] = (8,6)
plt.rcParams['font.size'] = 20

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
# Logistic Regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_log = logreg.predict(X_test)

log_train = round(logreg.score(X_train, y_train) * 100, 2)
log_accuracy = round(accuracy_score(y_pred_log, y_test) * 100, 2)
log_f1 = round(f1_score(y_pred_log, y_test) * 100, 2)

print("Training Accuracy    :",log_train,"%")
print("Model Accuracy Score :",log_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification Report: \n",classification_report(y_test,y_pred_log))
```

```
Training Accuracy    : 85.13 %
Model Accuracy Score : 83.7 %
-----------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        76
           1       0.86      0.86      0.86       108

    accuracy                           0.84       184
   macro avg       0.83      0.83      0.83       184
weighted avg       0.84      0.84      0.84       184
```

Confusion Matrix

Confusion Matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

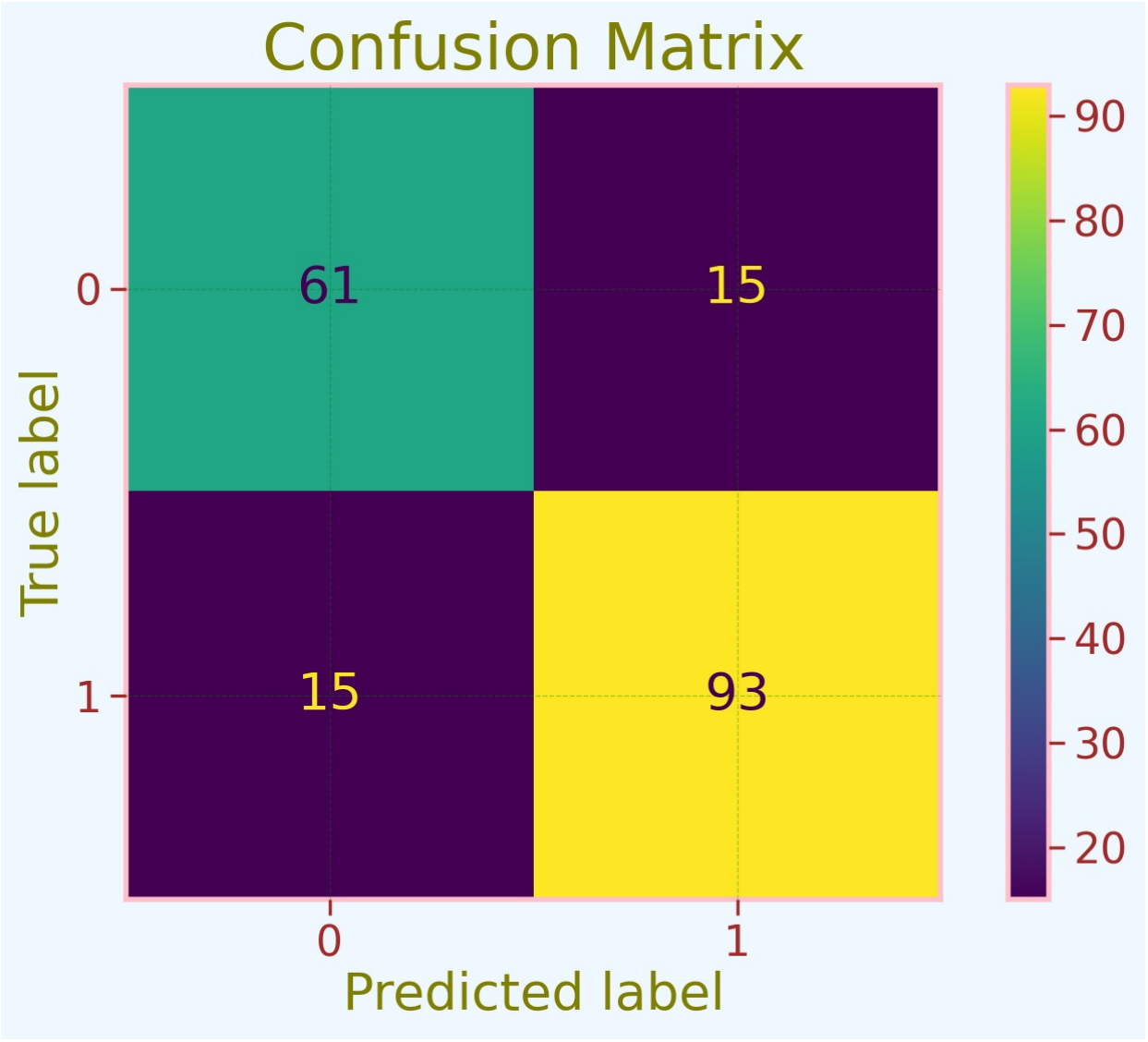
True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error.**

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error.**

- These four outcomes are summarized in a confusion matrix given below:

In [66]:
```
plot_confusion_matrix(logreg, X_test, y_test);
plt.title('Confusion Matrix');
```



ROC Curve

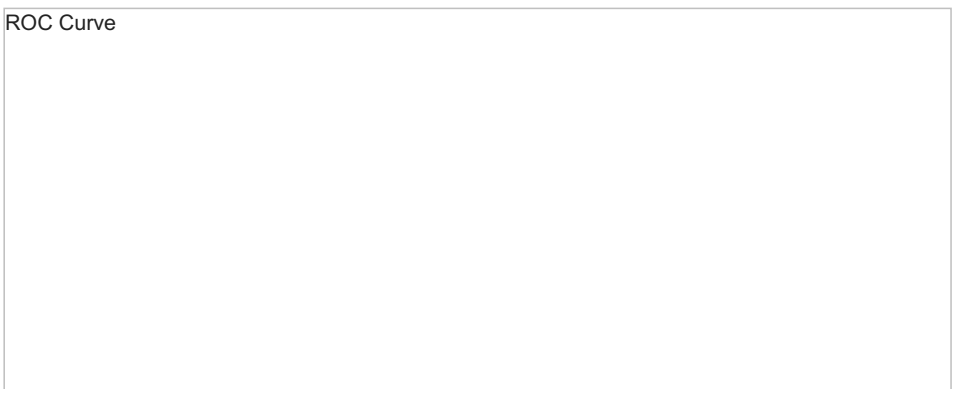
ROC Curve

ROC Curve (Receiver Operating Characteristic)

- The ROC curve tells us how well our classifier is classifying between term deposit suscriptions (True Positives) and non-term deposit suscriptions. The X-axis is represented by False positive rates (Specificity) and the Y-axis is represented by the True Positive Rate (Sensitivity.) As the line moves the threshold of the classification changes giving us different values. The closer is the line to our top left corner the better is our model separating both classes.

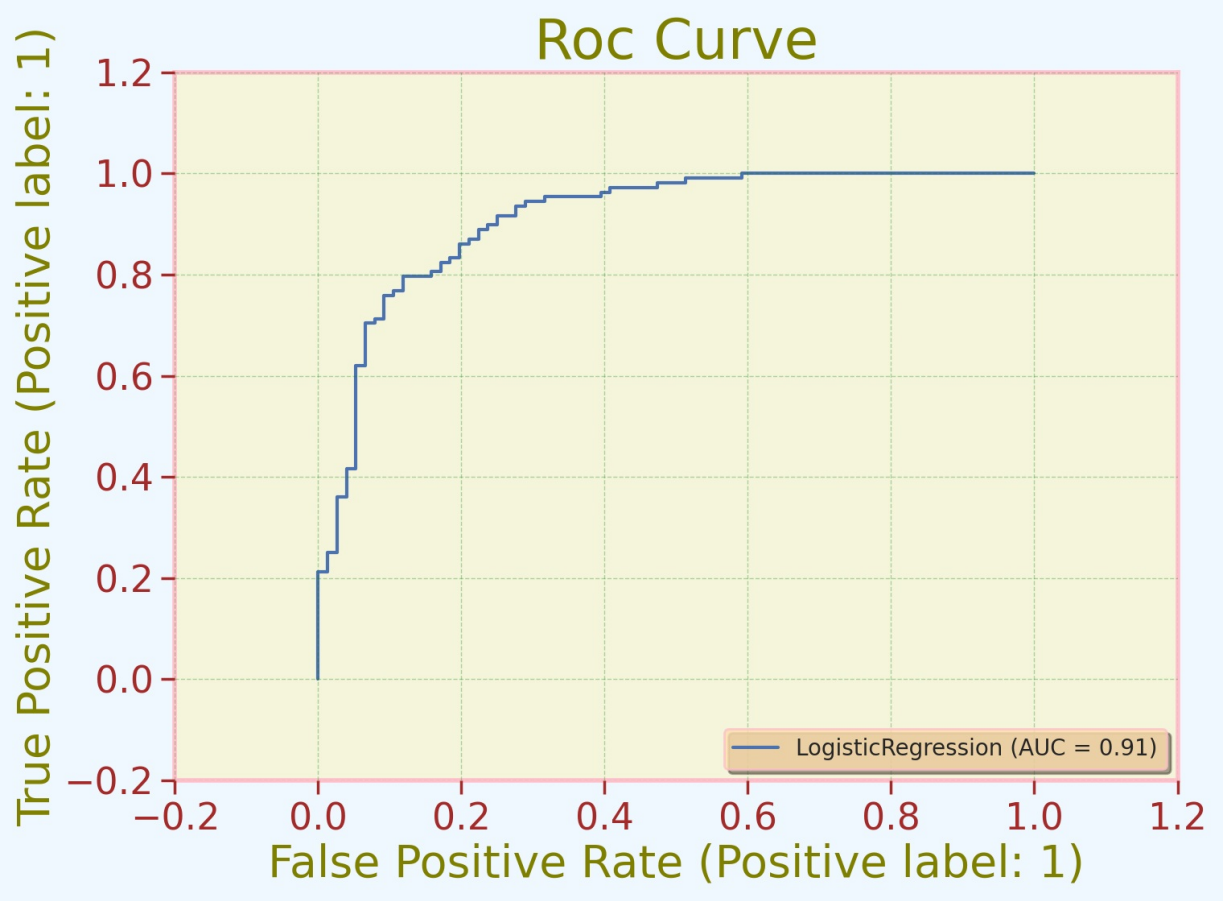
Area under the ROC curve:

- Area under the ROC (Receiver Operating Characteristic) curve : AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease



</center

In [67]:
```
plot_roc_curve(logreg, X_test, y_test);
plt.title('Roc Curve');
```



Precision Recall Curve

Precision Recall Curve

There are numerous ways to evaluate the performance of a classifier. In this article, we introduce the Precision-Recall Curve and further examine the difference between two popular performance reporting methods: Precision-Recall (PR) Curve and Receiver Operating Characteristic (ROC) Curve.

A PR curve is simply a graph with Precision values on the y-axis and Recall values on the x-axis. In other words, the PR curve contains TP/(TP+FN) on the y-axis and TP/(TP+FP) on the x-axis.

- It is important to note that Precision is also called the Positive Predictive Value (PPV)
- Recall is also called Sensitivity, Hit Rate or True Positive Rate (TPR)

Interpreting a PR Curve

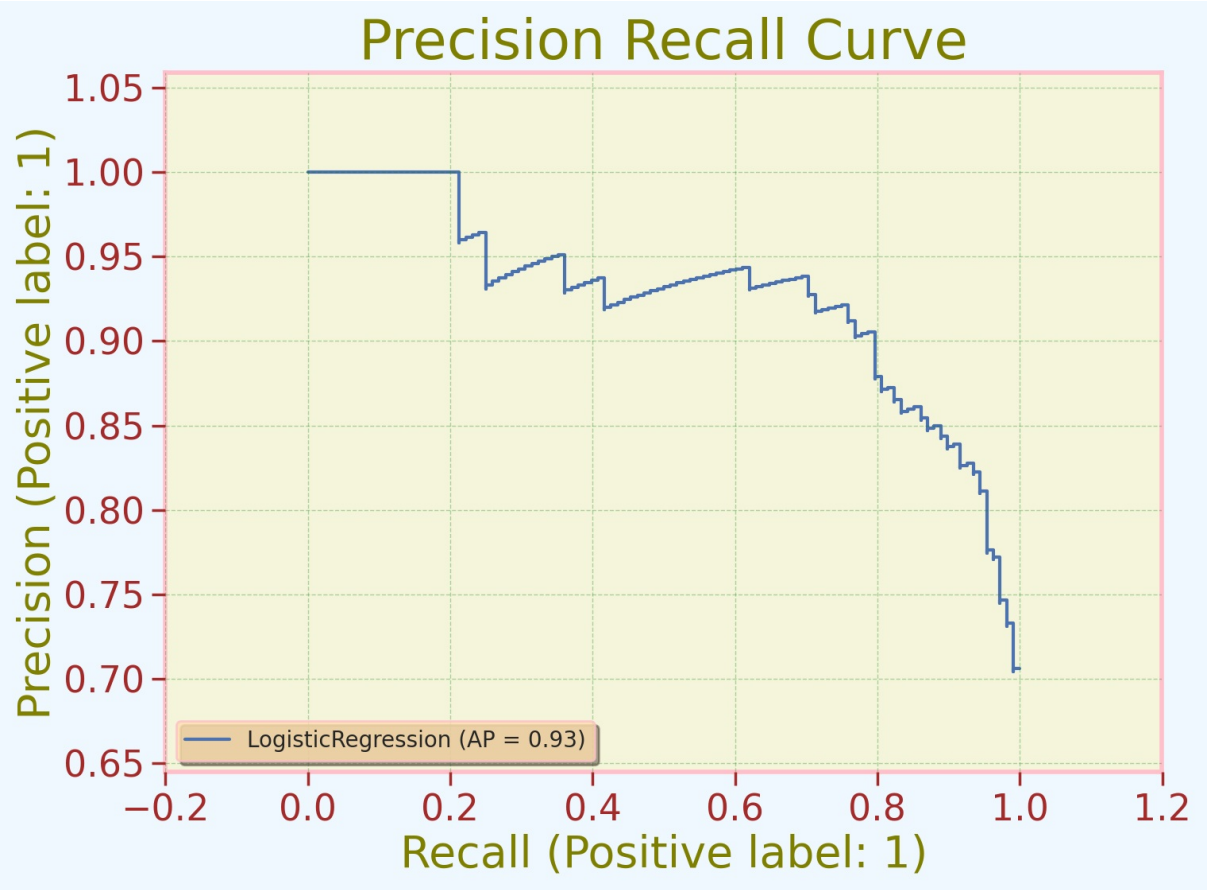
It is desired that the algorithm should have both high precision, and high recall. However, most machine learning

algorithms often involve a trade-off between the two. A good PR curve has greater AUC (area under curve). In the figure above, the classifier corresponding to the Orange line has better performance than the classifier corresponding to the Other lines.It is important to note that the classifier that has a higher AUC on the ROC curve will always have a higher AUC on the PR curve as well.

Precision Recall Curve

</center

In [68]:
```
plot_precision_recall_curve(logreg, X_test, y_test)
plt.title('Precision Recall Curve');
```



Cross val Score

Cross val Score

Cross_val_score is a function in the scikit-learn package which trains and tests a model over multiple folds of your dataset. This cross validation method gives you a better understanding of model performance over the whole dataset instead of just a single train/test split.

The process that cross_val_score uses is typical for cross validation and follows these steps:

- The number of folds is defined, by default this is 5
- The dataset is split up according to these folds, where each fold has a unique set of testing data
- A model is trained and tested for each fold
- A model is trained and tested for each fold
- The mean and standard deviation of these metrics can then be calculated to provide a single metric for the process

What is cross_val_score used for?

- Cross_val_score is used as a simple cross validation technique to prevent over-fitting and promote model generalisation.
- The typical process of model development is to train a model on one fold of data and then test on another. But how do we know that this single test dataset is representative? This is why we use cross_val_score and cross validation more generally, to train and test our model on multiple folds such that we can be sure out model generalises well across the whole dataset and not just a single portion.
- If we see that the metrics for all folds in cross_val_score are uniform then it can be concluded that the model is able to generalise, however if there are significant differences between them then this may indicate over-fitting to certain folds and would need to be investigated further

In [69]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=logreg, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

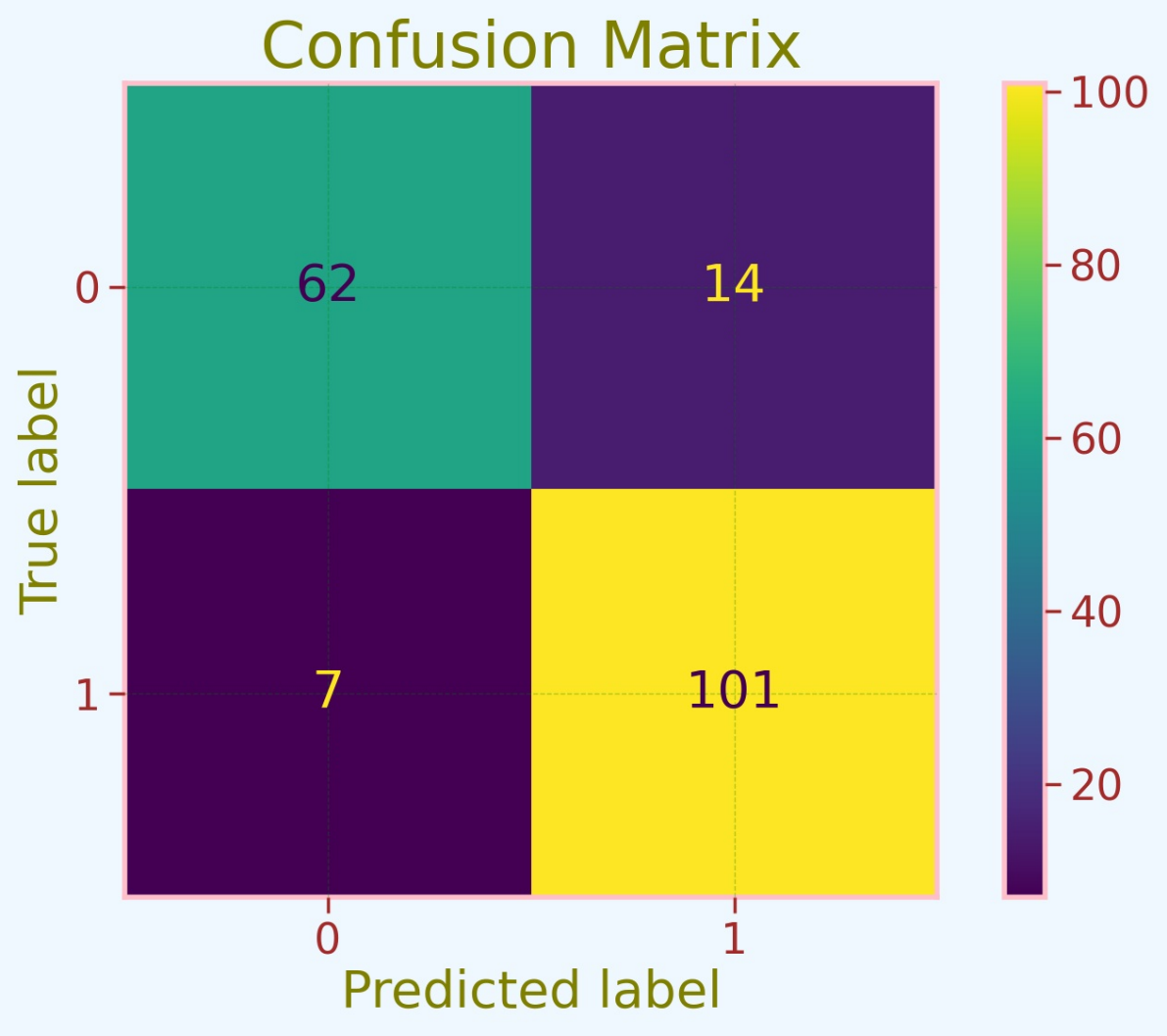
```
Model Accuracy Score: 84.85 %
Std. Dev: 2.36 %
```

Support Vector Machines

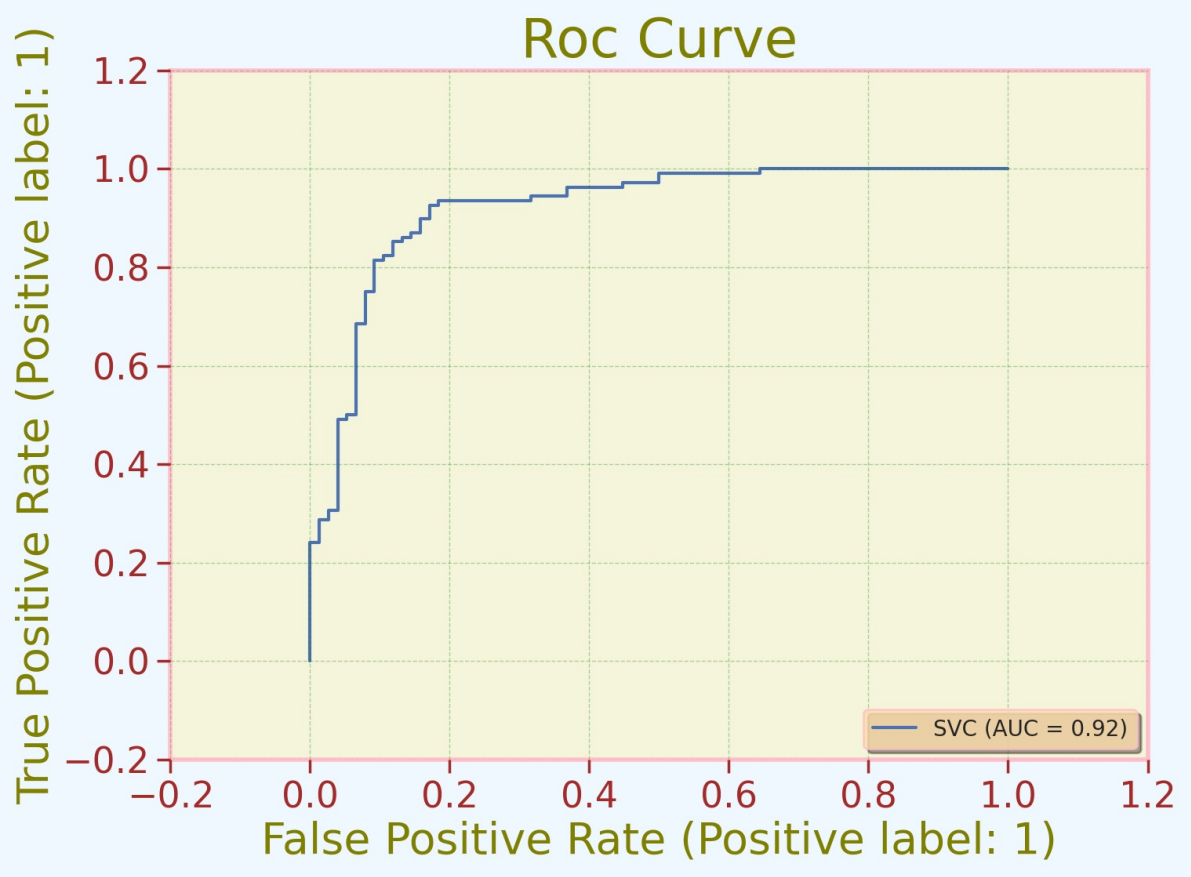
In [70]:
```
# Support Vector Machines
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)

svc_train = round(svc.score(X_train, y_train) * 100, 2)
svc_accuracy = round(accuracy_score(y_pred_svc, y_test) * 100, 2)
svc_f1 = round(f1_score(y_pred_svc, y_test) * 100, 2)

print("Training Accuracy     :",svc_train,"%")
print("Model Accuracy Score  :",svc_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_svc))
```

```
Training Accuracy     : 87.31 %
Model Accuracy Score  : 88.59 %
--------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.90      0.82      0.86        76
           1       0.88      0.94      0.91       108

    accuracy                           0.89       184
   macro avg       0.89      0.88      0.88       184
weighted avg       0.89      0.89      0.88       184
```

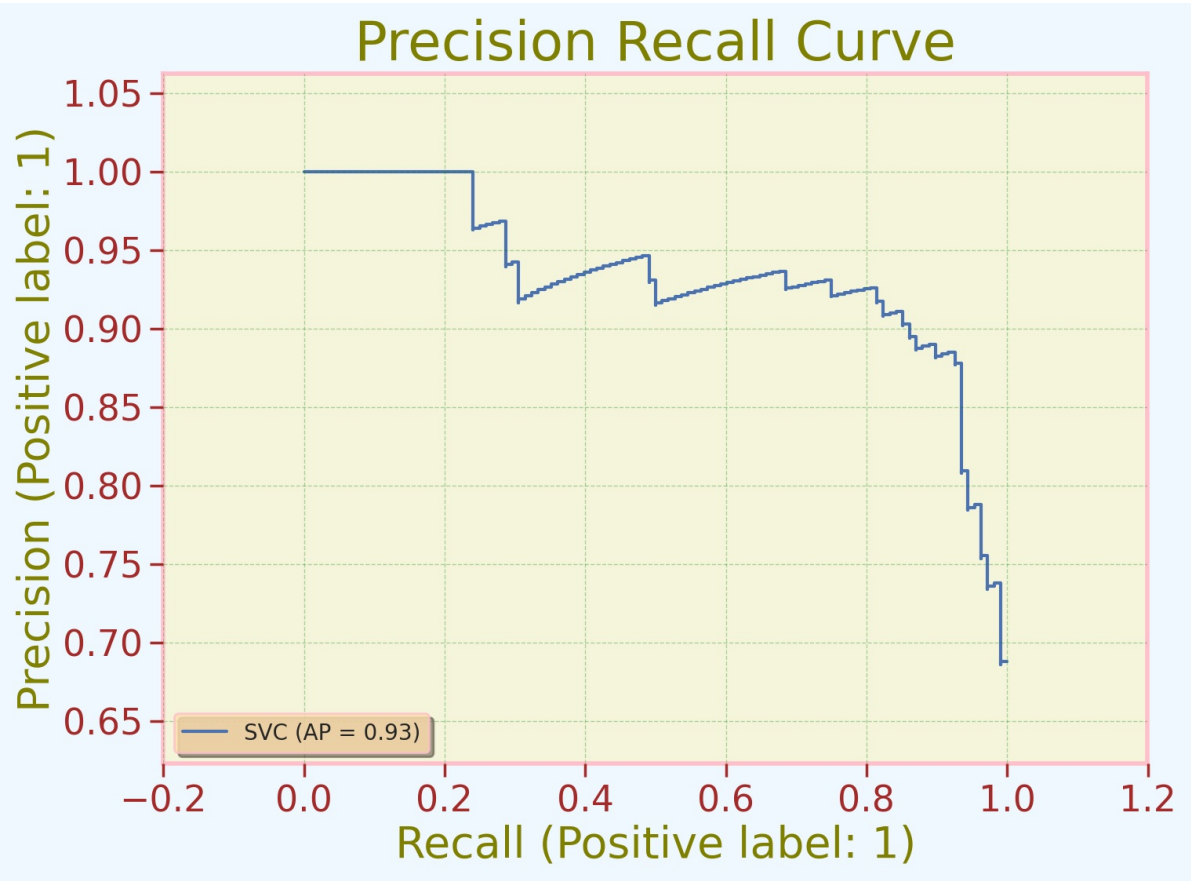
In [71]:
```
plot_confusion_matrix(svc, X_test, y_test);
plt.title('Confusion Matrix');
```



In [72]:
```
plot_roc_curve(svc, X_test, y_test);
plt.title('Roc Curve');
```



In [73]:
```
plot_precision_recall_curve(svc, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [74]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=svc, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

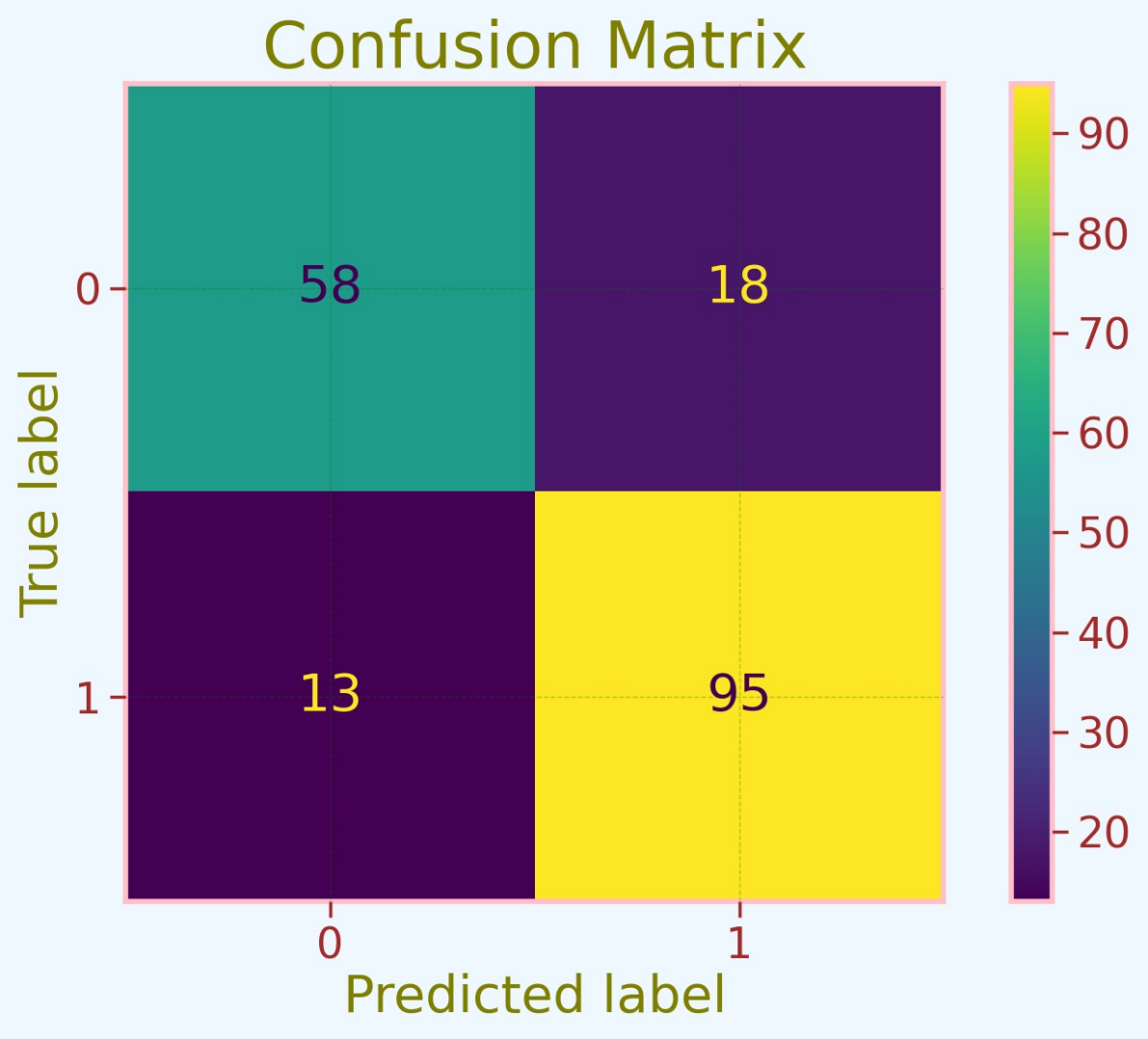
```
Model Accuracy Score: 84.56 %
Std. Dev: 3.93 %
```

KNeighborsClassifier

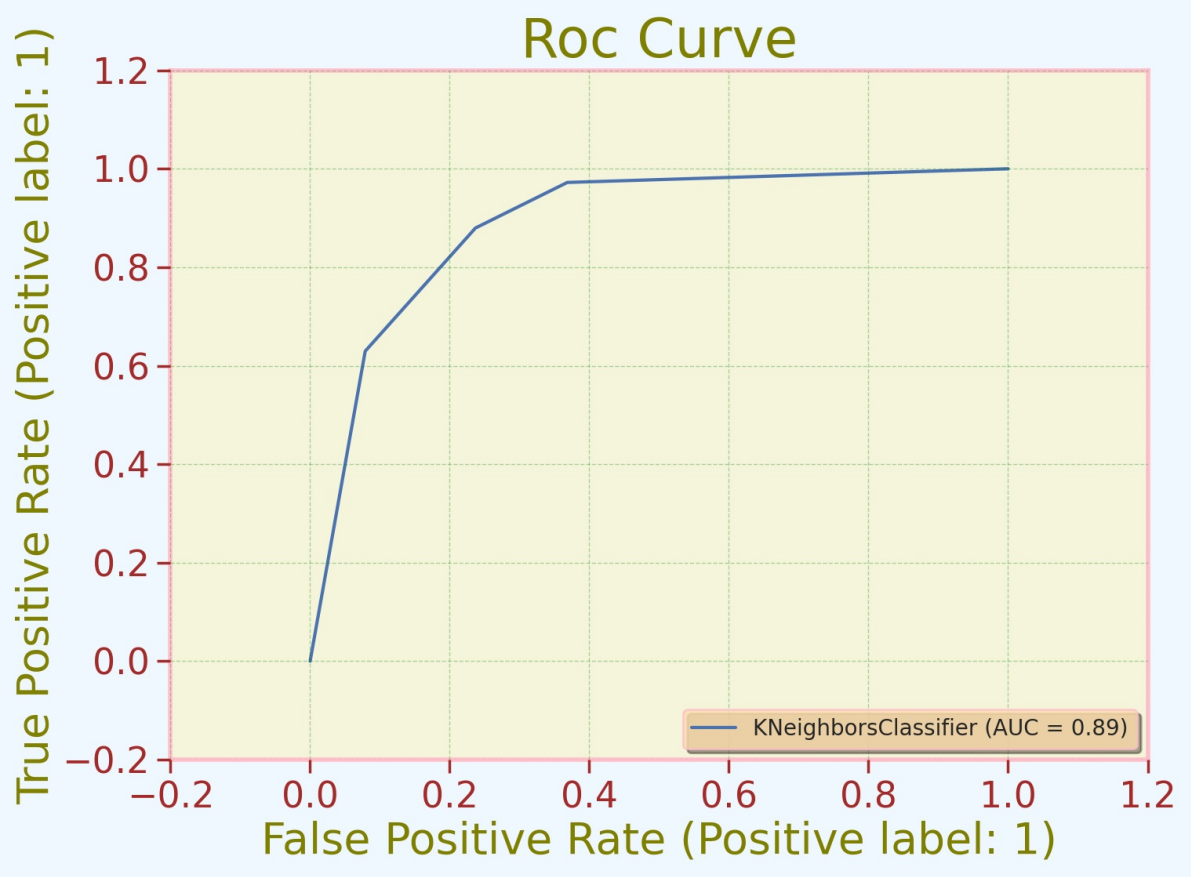
In [75]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

knn_train = round(knn.score(X_train, y_train) * 100, 2)
knn_accuracy = round(accuracy_score(y_pred_knn, y_test) * 100, 2)
knn_f1 = round(f1_score(y_pred_knn, y_test) * 100, 2)

print("Training Accuracy     :",knn_train,"%")
print("Model Accuracy Score  :",knn_accuracy,"%")
print("\033[1m-------------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_knn))
```

```
Training Accuracy     : 89.63 %
Model Accuracy Score  : 83.15 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.82      0.76      0.79        76
           1       0.84      0.88      0.86       108

    accuracy                           0.83       184
   macro avg       0.83      0.82      0.82       184
weighted avg       0.83      0.83      0.83       184
```

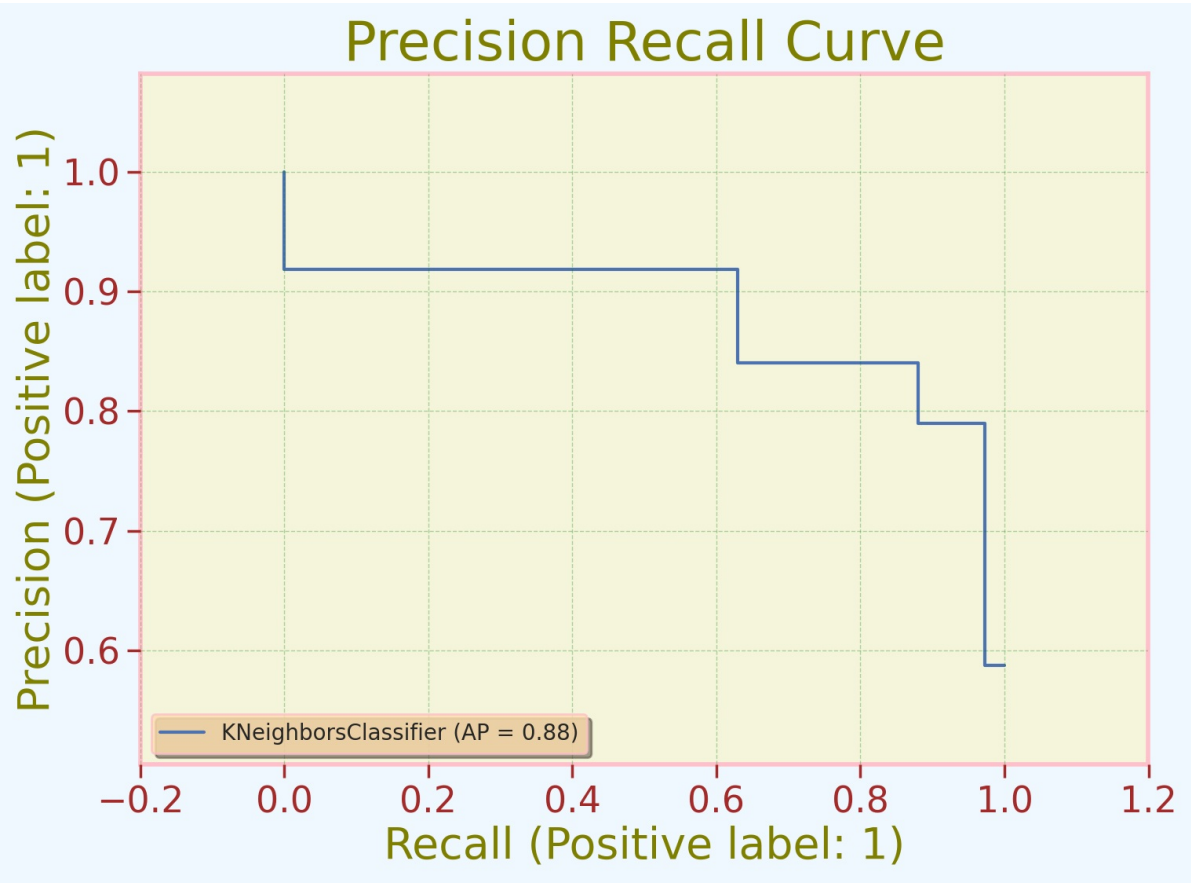
In [76]:
```
plot_confusion_matrix(knn, X_test, y_test);
plt.title('Confusion Matrix');
```



In [77]:
```
plot_roc_curve(knn, X_test, y_test);
plt.title('Roc Curve');
```



In [78]:
```
plot_precision_recall_curve(knn, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [79]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=knn, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

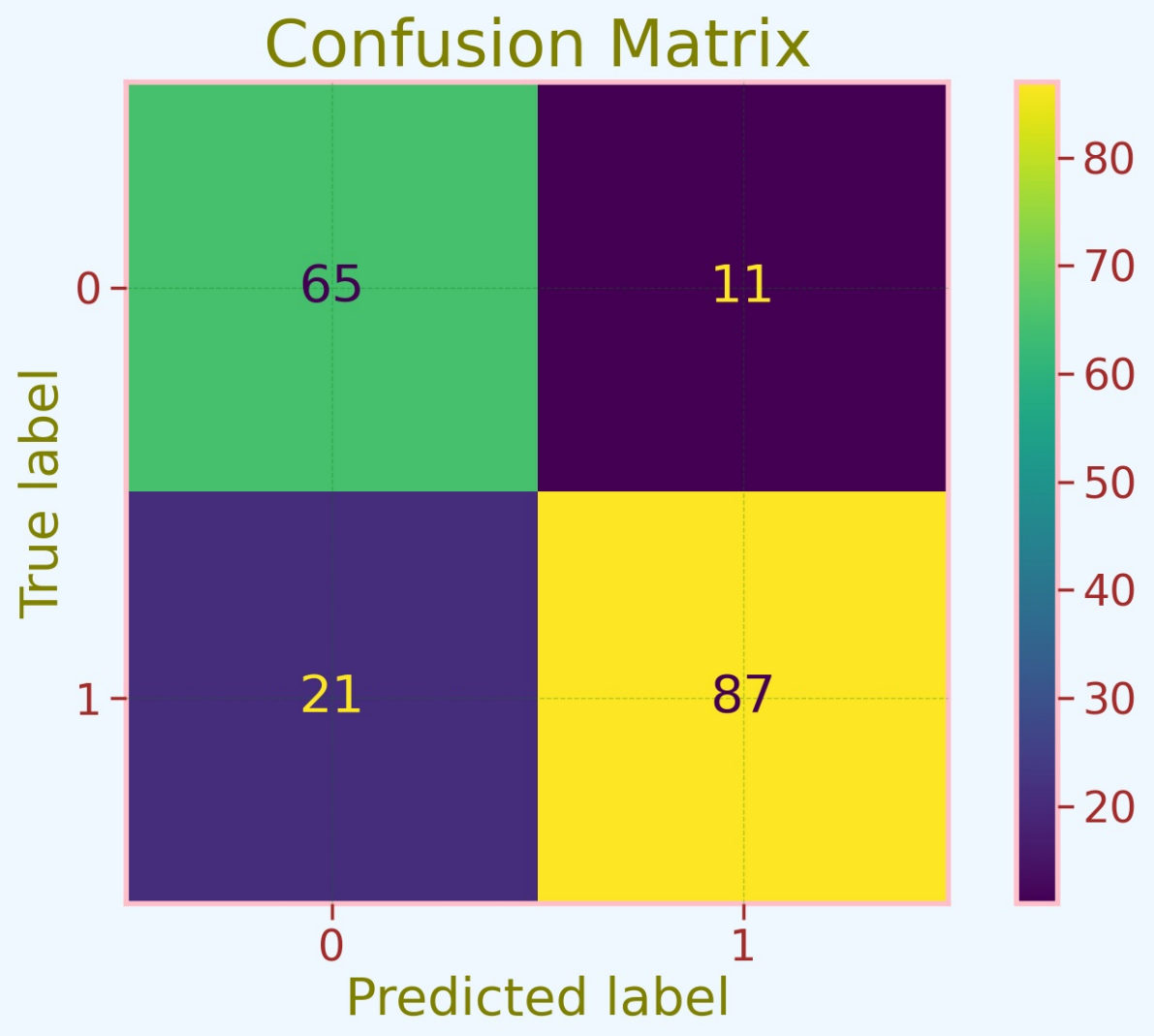
```
Model Accuracy Score: 81.98 %
Std. Dev: 3.33 %
```

GaussianNB

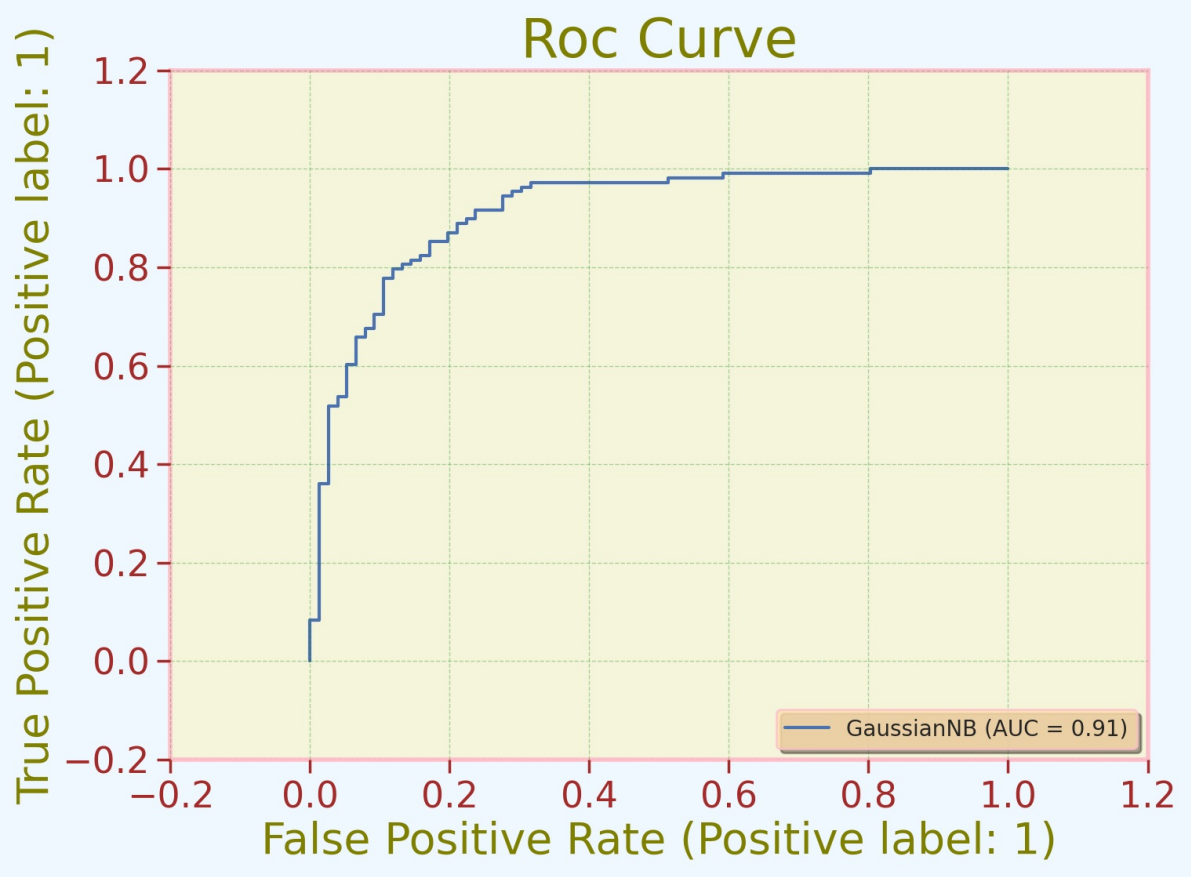
In [80]:
```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
y_pred_NB = gaussian.predict(X_test)

gaussian_train = round(gaussian.score(X_train, y_train) * 100, 2)
gaussian_accuracy = round(accuracy_score(y_pred_NB, y_test) * 100, 2)
gaussian_f1 = round(f1_score(y_pred_NB, y_test) * 100, 2)

print("Training Accuracy     :",gaussian_train,"%")
print("Model Accuracy Score  :",gaussian_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_NB))
```

```
Training Accuracy     : 83.77 %
Model Accuracy Score  : 82.61 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.76      0.86      0.80        76
           1       0.89      0.81      0.84       108

    accuracy                           0.83       184
   macro avg       0.82      0.83      0.82       184
weighted avg       0.83      0.83      0.83       184
```

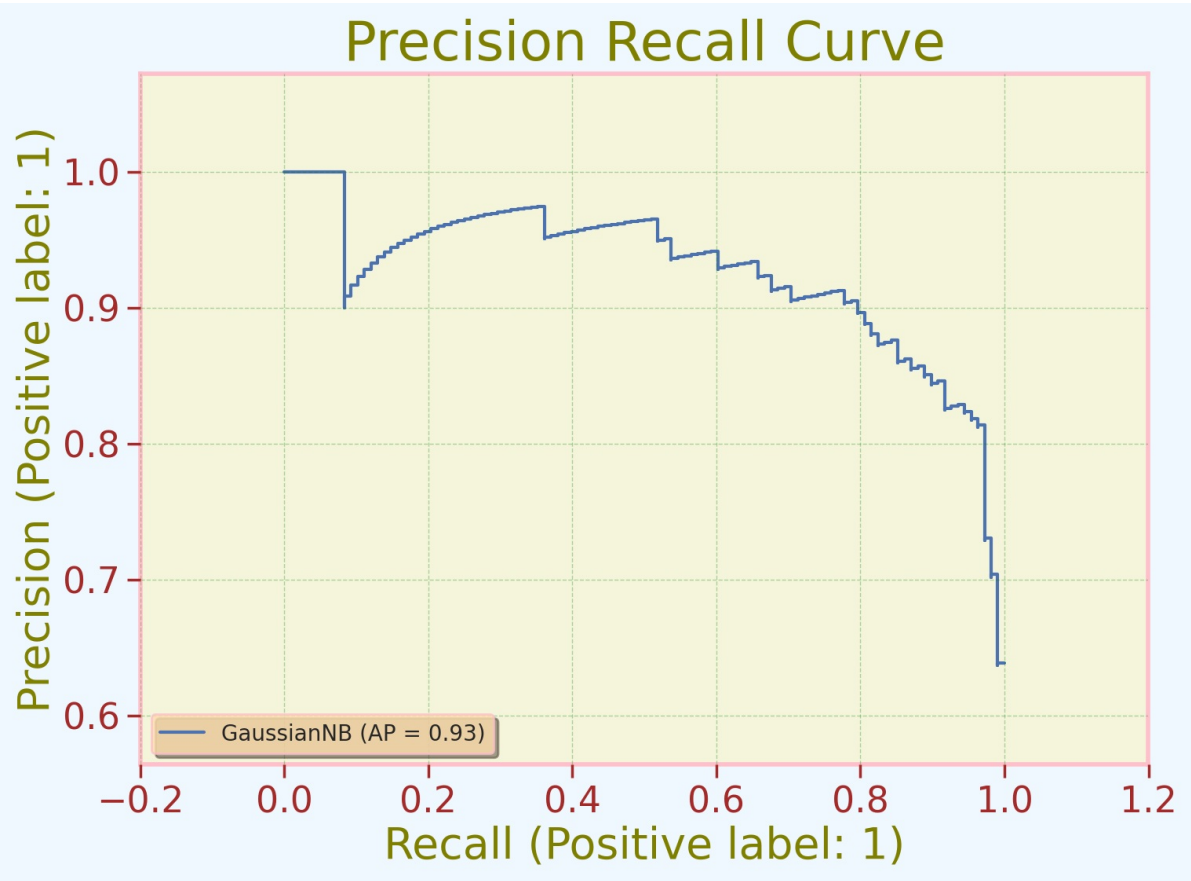
In [81]:
```
plot_confusion_matrix(gaussian, X_test, y_test);
plt.title('Confusion Matrix');
```



In [82]:
```
plot_roc_curve(gaussian, X_test, y_test);
plt.title('Roc Curve');
```



In [83]:
```
plot_precision_recall_curve(gaussian, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [84]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=gaussian, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

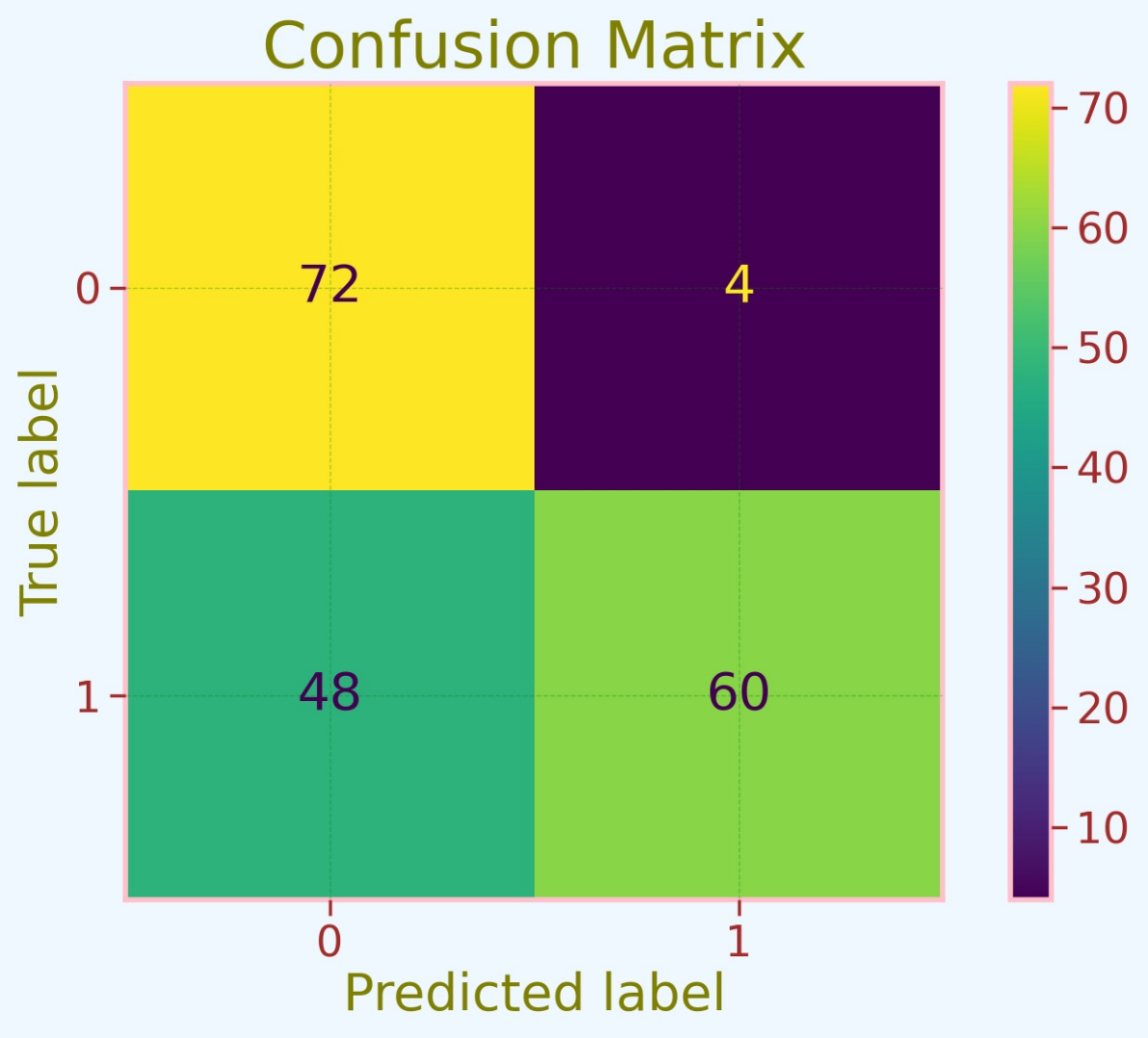
```
Model Accuracy Score: 83.08 %
Std. Dev: 3.39 %
```

Perceptron

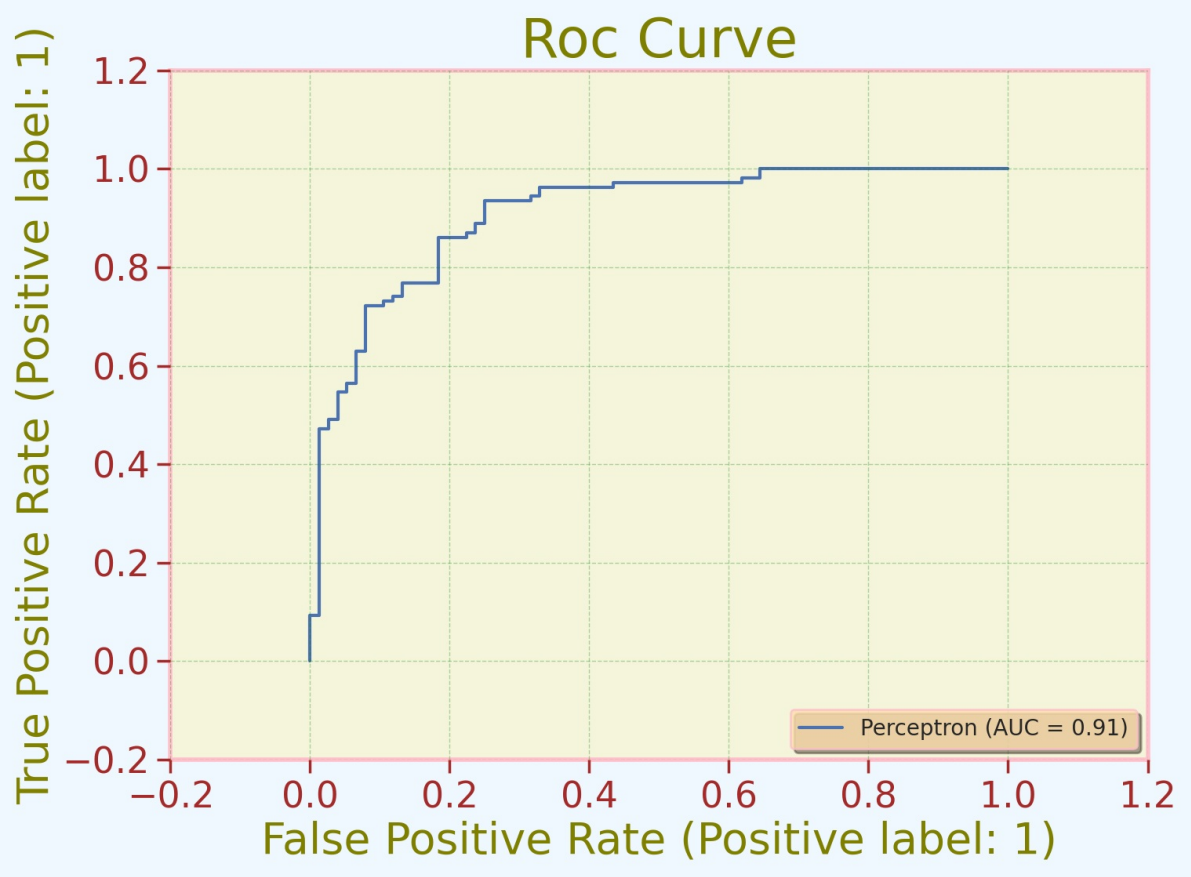
In [85]:
```
# Perceptron

from sklearn.linear_model import Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, y_train)
y_pred_perceptron = perceptron.predict(X_test)


perceptron_train = round(perceptron.score(X_train, y_train) * 100, 2)
perceptron_accuracy = round(accuracy_score(y_pred_perceptron, y_test) * 100, 2)
perceptron_f1 = round(f1_score(y_pred_perceptron, y_test) * 100, 2)

print("Training Accuracy    :",perceptron_train,"%")
print("Model Accuracy Score :",perceptron_accuracy,"%")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_perceptron))
```

```
Training Accuracy    : 71.08 %
Model Accuracy Score : 71.74 %
-----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.60      0.95      0.73        76
           1       0.94      0.56      0.70       108

    accuracy                           0.72       184
   macro avg       0.77      0.75      0.72       184
weighted avg       0.80      0.72      0.71       184
```

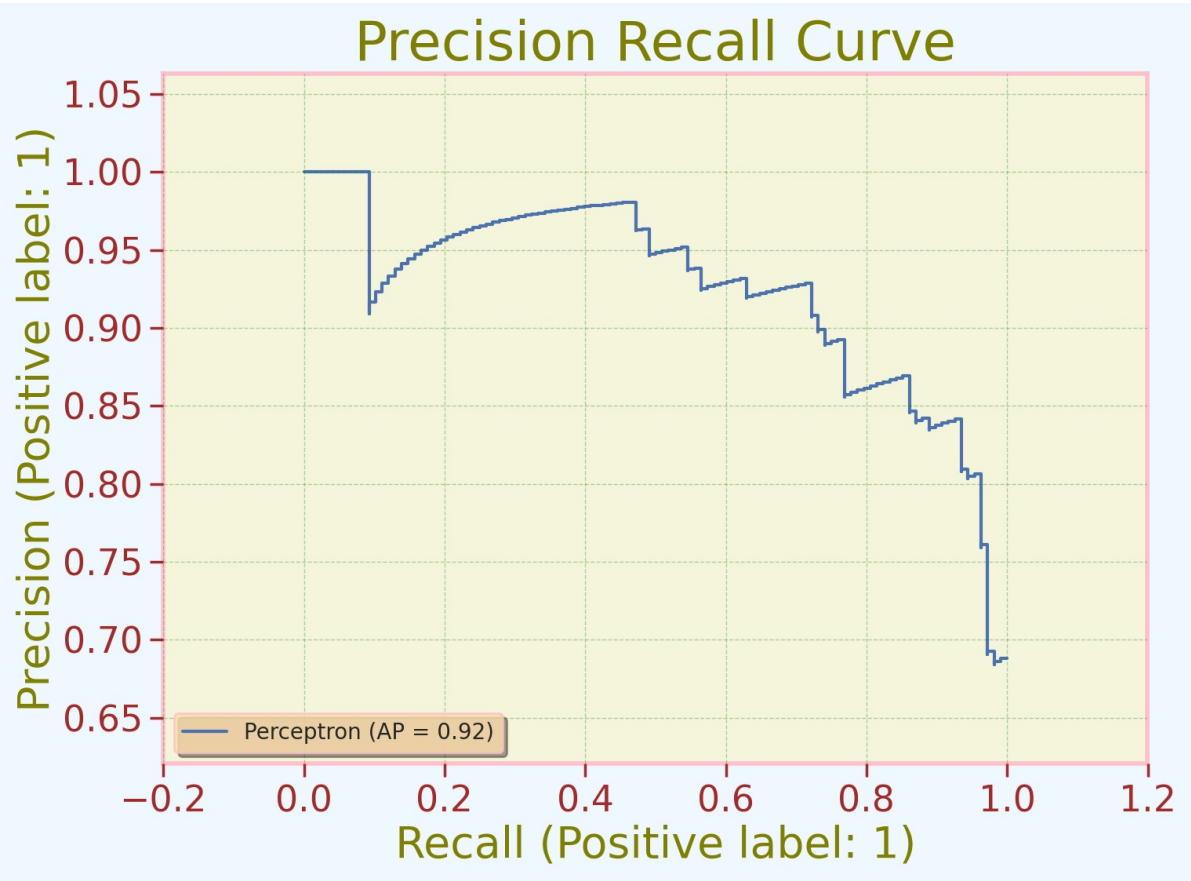
In [86]:
```
plot_confusion_matrix(perceptron, X_test, y_test);
plt.title('Confusion Matrix');
```



In [87]:
```
plot_roc_curve(perceptron, X_test, y_test);
plt.title('Roc Curve');
```



In [88]:
```
plot_precision_recall_curve(perceptron, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [89]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=perceptron, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

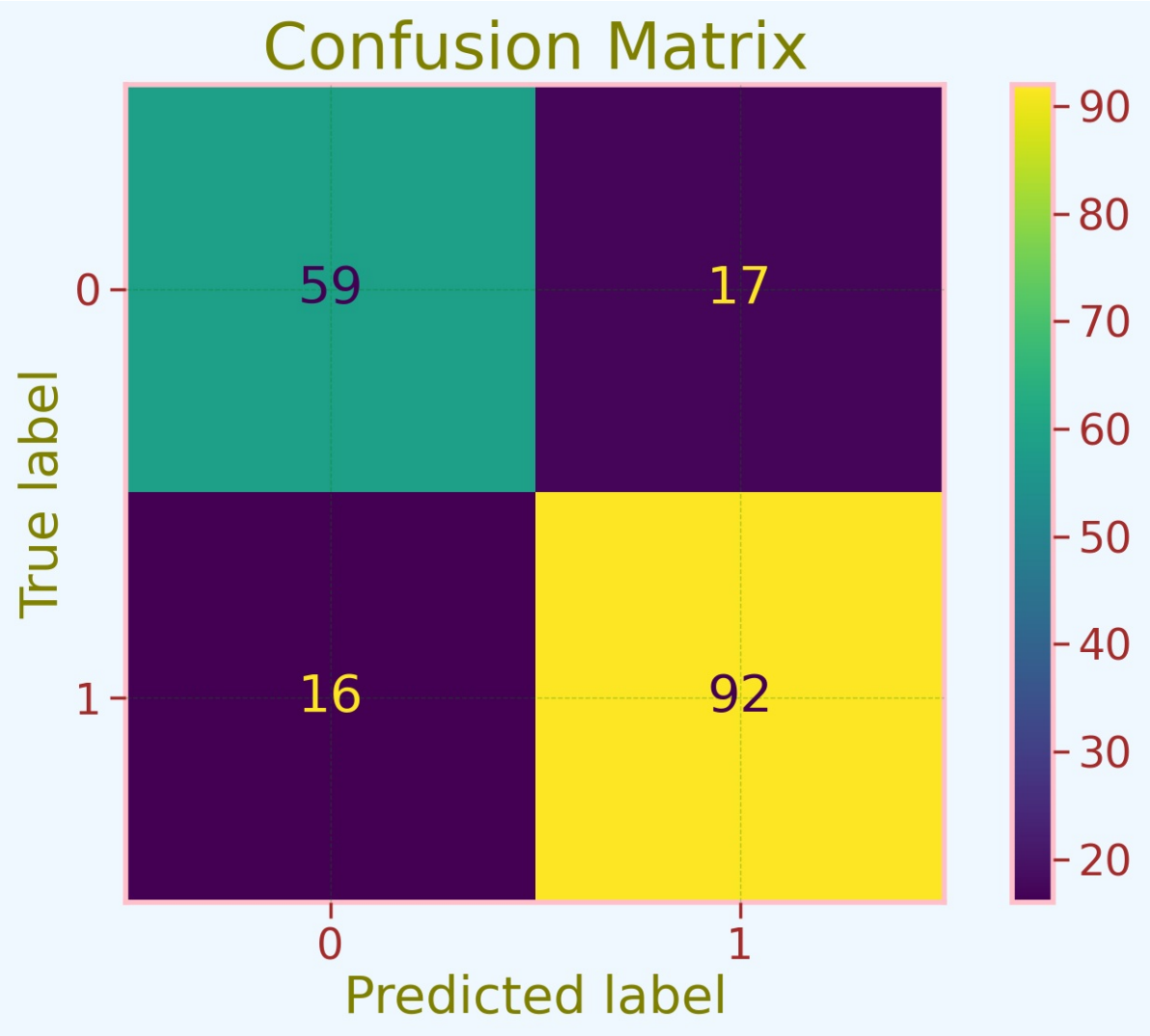
```
Model Accuracy Score: 75.31 %
Std. Dev: 8.66 %
```

LinearSVC

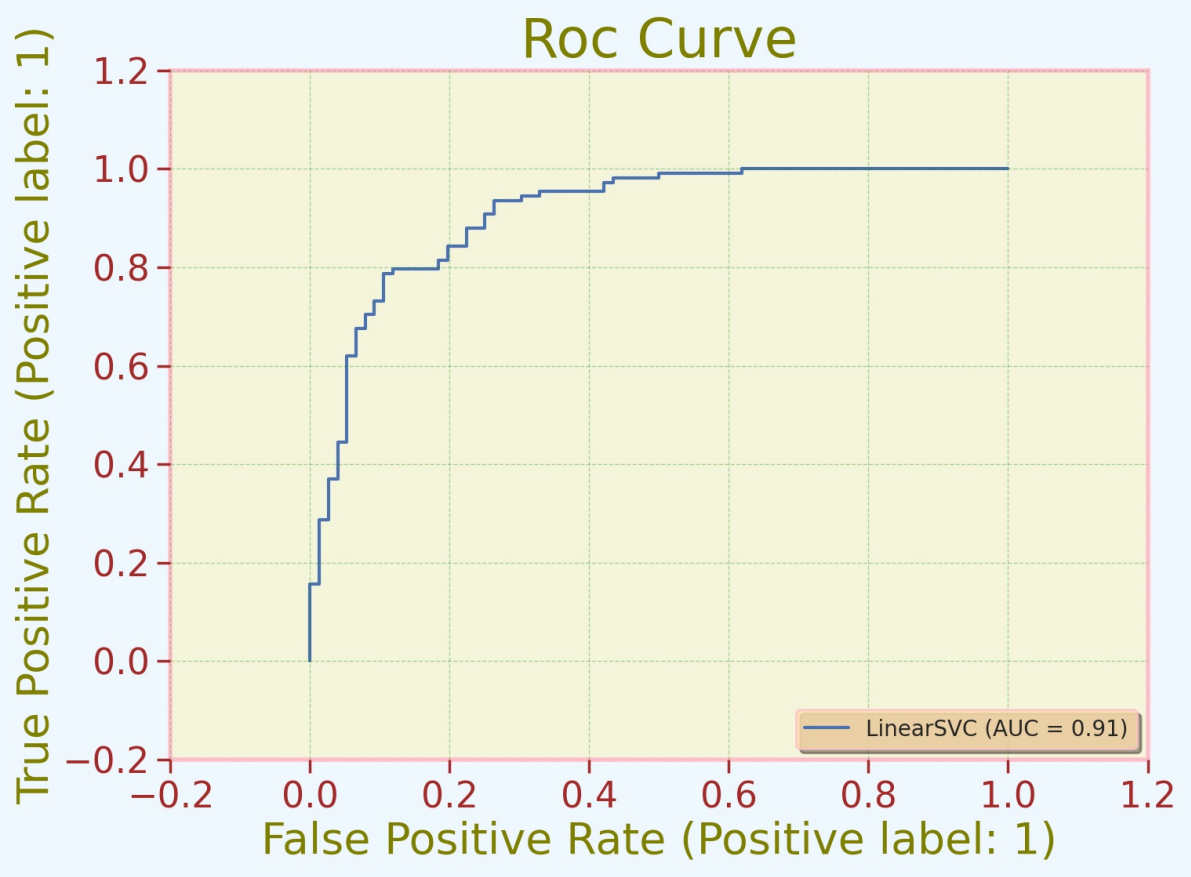
In [90]:
```
# Linear SVC
from sklearn.svm import LinearSVC
linear_svc = LinearSVC()
linear_svc.fit(X_train, y_train)
y_pred_LinearSVC = linear_svc.predict(X_test)

linear_svc_train = round(linear_svc.score(X_train, y_train) * 100, 2)
linear_svc_accuracy = round(accuracy_score(y_pred_LinearSVC, y_test) * 100, 2)
linear_svc_f1 = round(f1_score(y_pred_LinearSVC, y_test) * 100, 2)

print("Training Accuracy    :",linear_svc_train,"%")
print("Model Accuracy Score :",linear_svc_accuracy,"%")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_LinearSVC))
```

```
Training Accuracy    : 85.13 %
Model Accuracy Score : 82.07 %
-----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.79      0.78      0.78        76
           1       0.84      0.85      0.85       108

    accuracy                           0.82       184
   macro avg       0.82      0.81      0.81       184
weighted avg       0.82      0.82      0.82       184
```

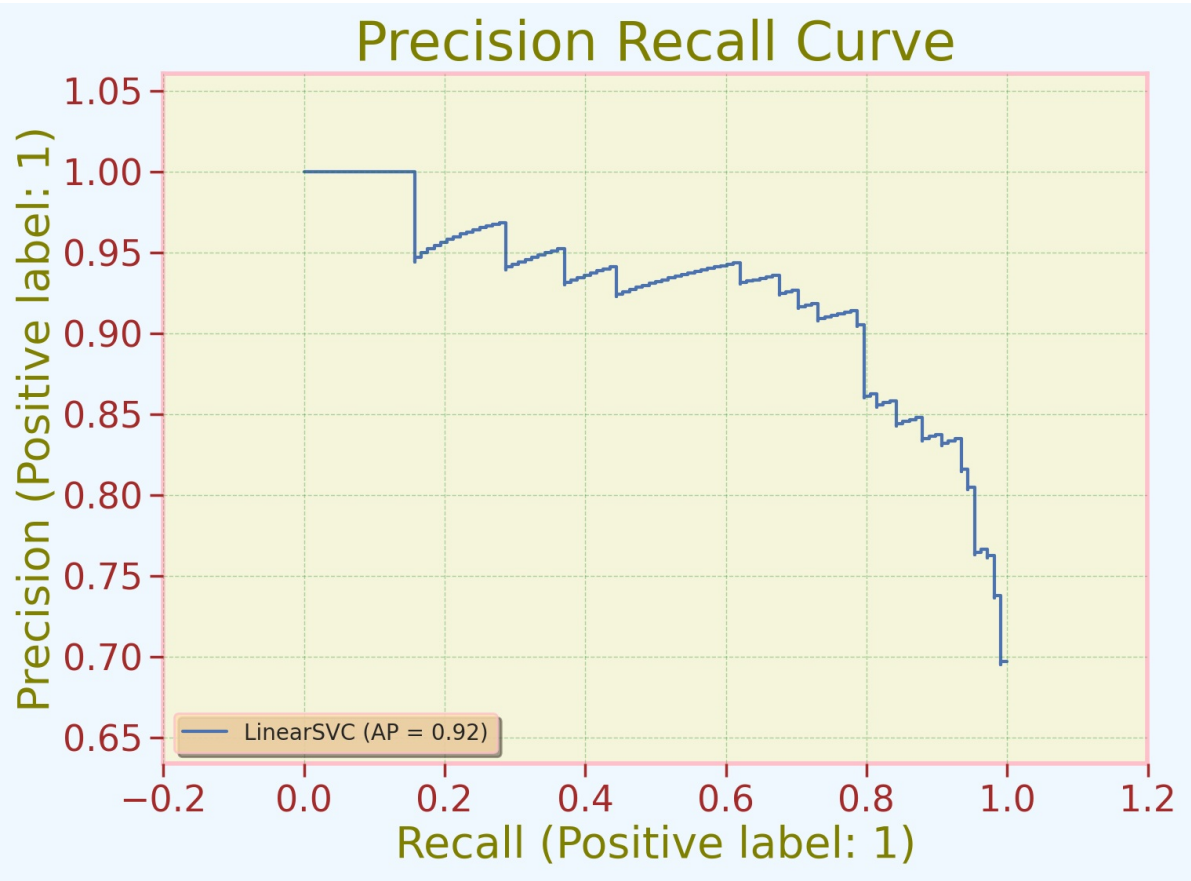
In [91]:
```
plot_confusion_matrix(linear_svc, X_test, y_test);
plt.title('Confusion Matrix');
```



In [92]:
```
plot_roc_curve(linear_svc, X_test, y_test);
plt.title('Roc Curve');
```



In [93]:
```
plot_precision_recall_curve(linear_svc, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [94]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=linear_svc, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

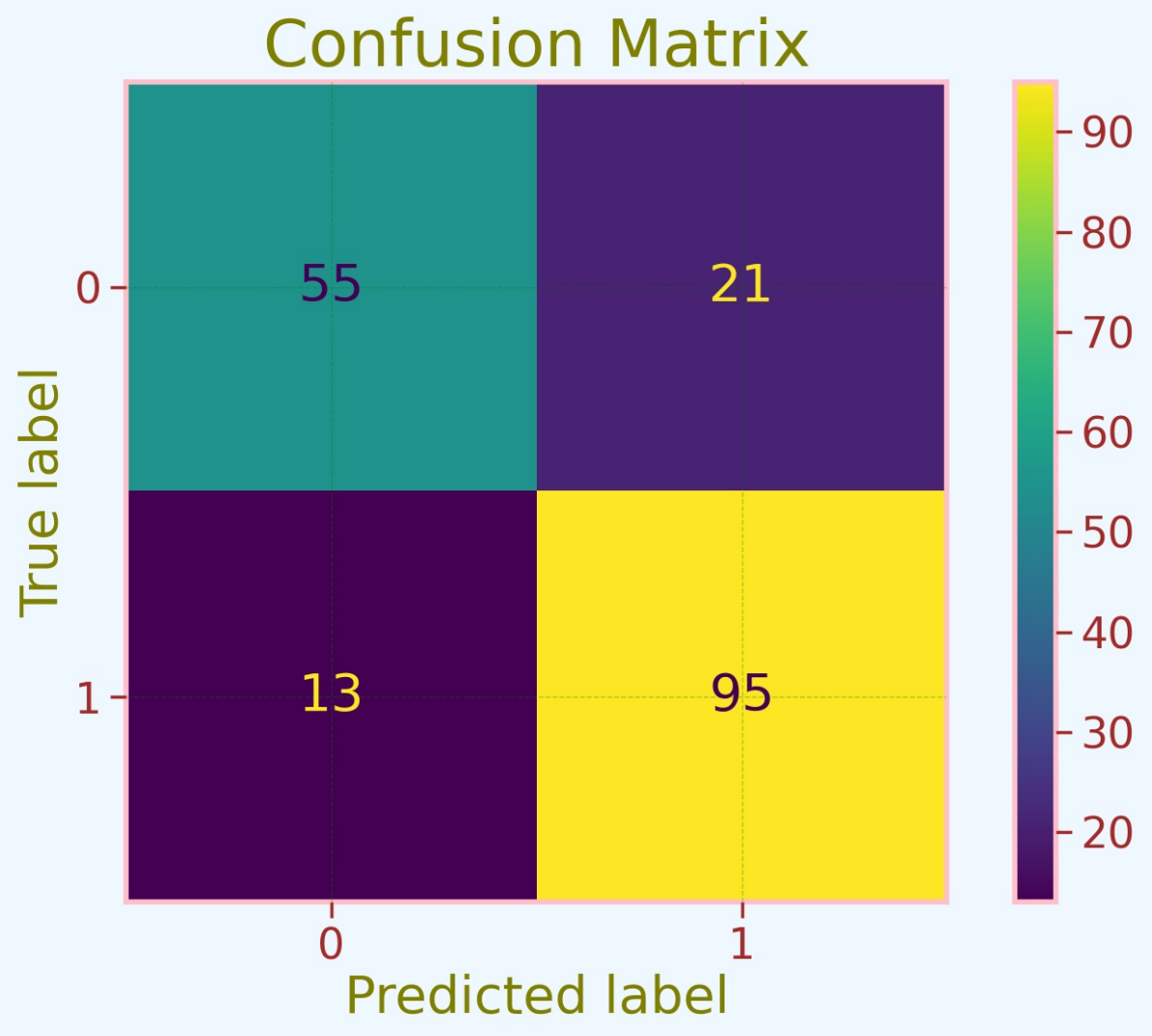
```
Model Accuracy Score: 84.85 %
Std. Dev: 2.10 %
```

SGDClassifier

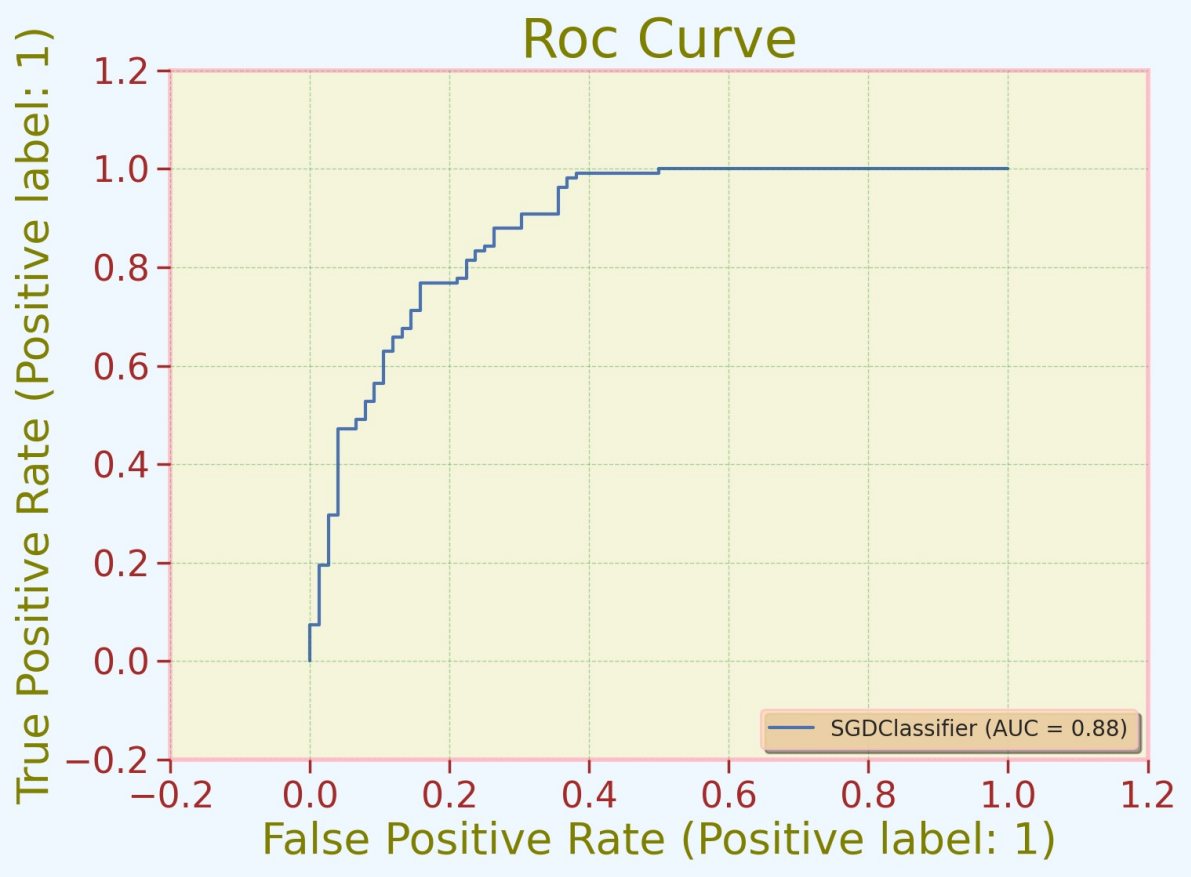
In [95]:
```
# Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd.fit(X_train, y_train)
y_pred_SGD = sgd.predict(X_test)

sgd_train = round(sgd.score(X_train, y_train) * 100, 2)
sgd_accuracy = round(accuracy_score(y_pred_SGD, y_test) * 100, 2)
sgd_f1 = round(f1_score(y_pred_SGD, y_test) * 100, 2)

print("Training Accuracy     :",sgd_train,"%")
print("Model Accuracy Score  :",sgd_accuracy,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_SGD))
```

```
Training Accuracy     : 84.72 %
Model Accuracy Score  : 81.52 %
----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.81      0.72      0.76        76
           1       0.82      0.88      0.85       108

    accuracy                           0.82       184
   macro avg       0.81      0.80      0.81       184
weighted avg       0.81      0.82      0.81       184
```

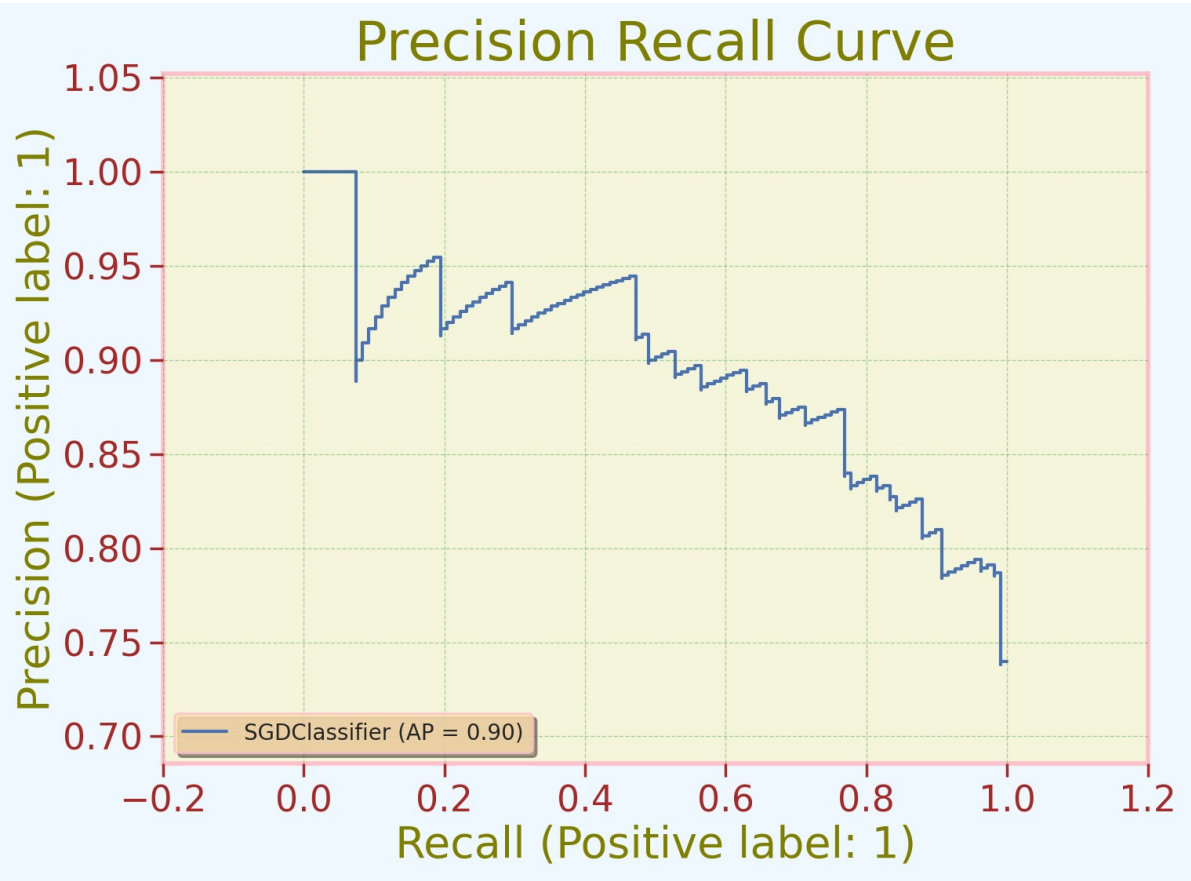
In [96]:
```
plot_confusion_matrix(sgd, X_test, y_test);
plt.title('Confusion Matrix');
```



In [97]:
```
plot_roc_curve(sgd, X_test, y_test);
plt.title('Roc Curve');
```



In [98]:
```
plot_precision_recall_curve(sgd, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [99]:
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=sgd, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 81.70 %
Std. Dev: 4.27 %
```

SDecisionTreeClassifier

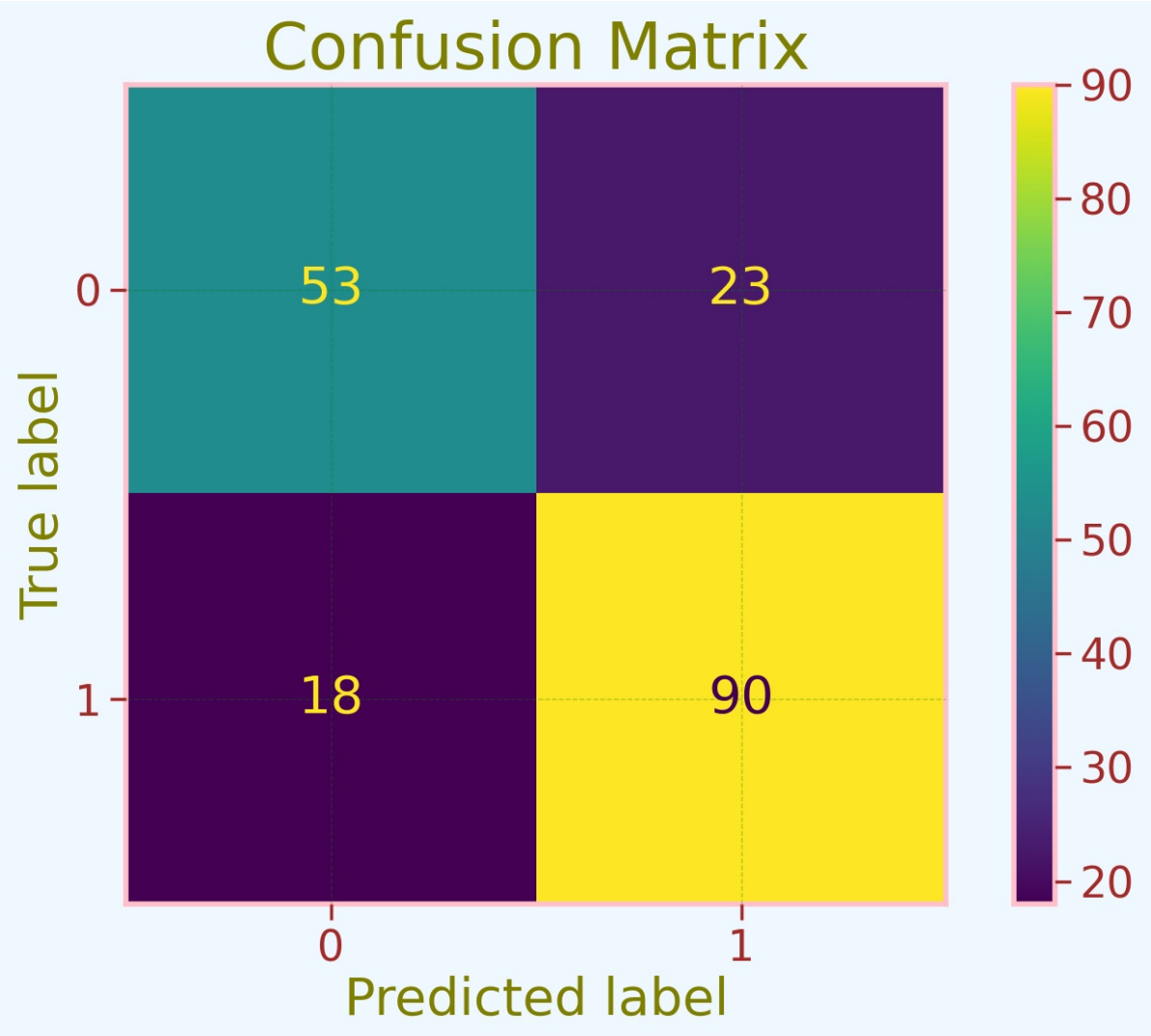
In [100…

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision = DecisionTreeClassifier()
decision.fit(X_train, y_train)
y_pred_Decision = decision.predict(X_test)

decision_train = round(decision.score(X_train, y_train) * 100, 2)
decision_accuracy = round(accuracy_score(y_pred_Decision, y_test) * 100, 2)
decision_f1 = round(f1_score(y_pred_Decision, y_test) * 100, 2)

print("Training Accuracy     :",decision_train,"%")
print("Model Accuracy Score  :",decision_accuracy,"%")
print("\033[1m-------------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_Decision))
```

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 77.72 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.75      0.70      0.72        76
           1       0.80      0.83      0.81       108

    accuracy                           0.78       184
   macro avg       0.77      0.77      0.77       184
weighted avg       0.78      0.78      0.78       184
```

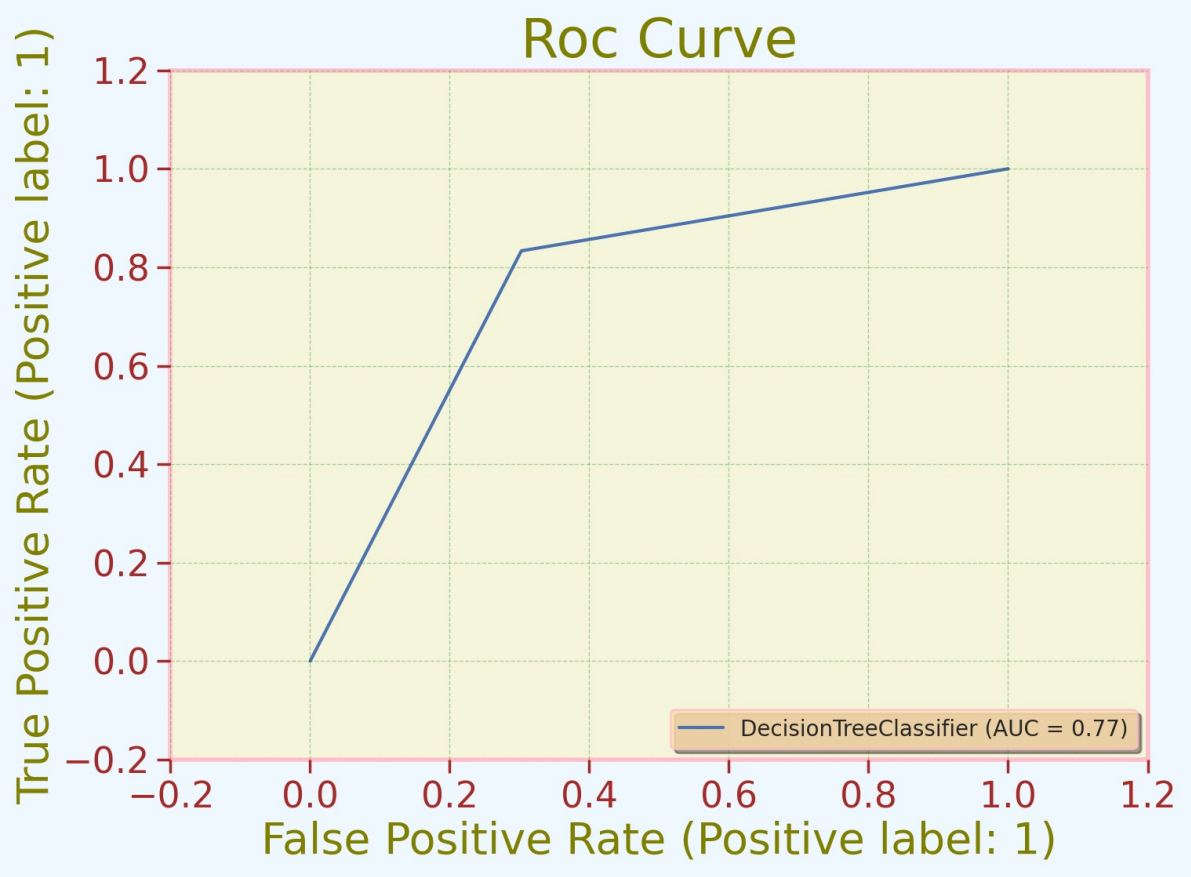
In [101…

```
plot_confusion_matrix(decision, X_test, y_test);
plt.title('Confusion Matrix');
```

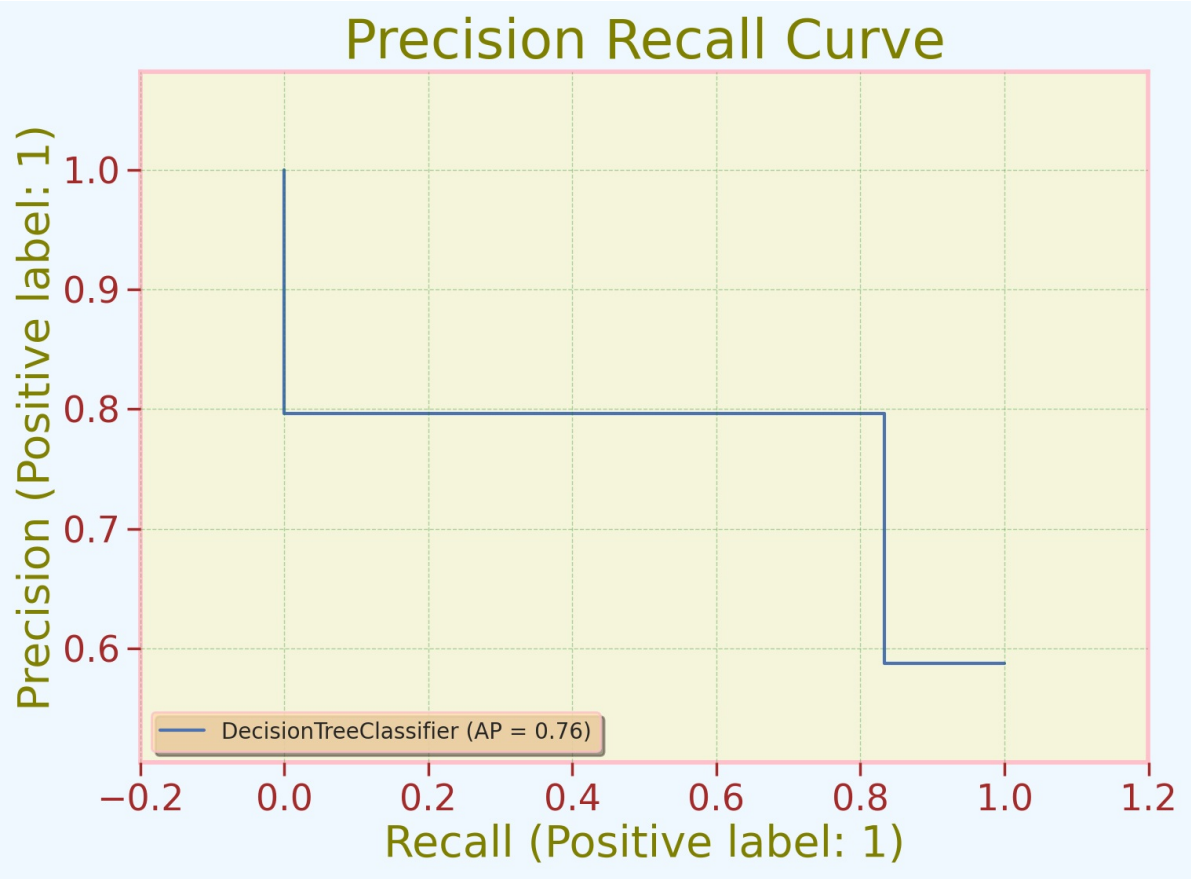


In [102…

```
plot_roc_curve(decision, X_test, y_test);
plt.title('Roc Curve');
```



In [103…
```
plot_precision_recall_curve(decision, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [104…
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=decision, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 79.52 %
Std. Dev: 5.29 %
```

RandomForestClassifier

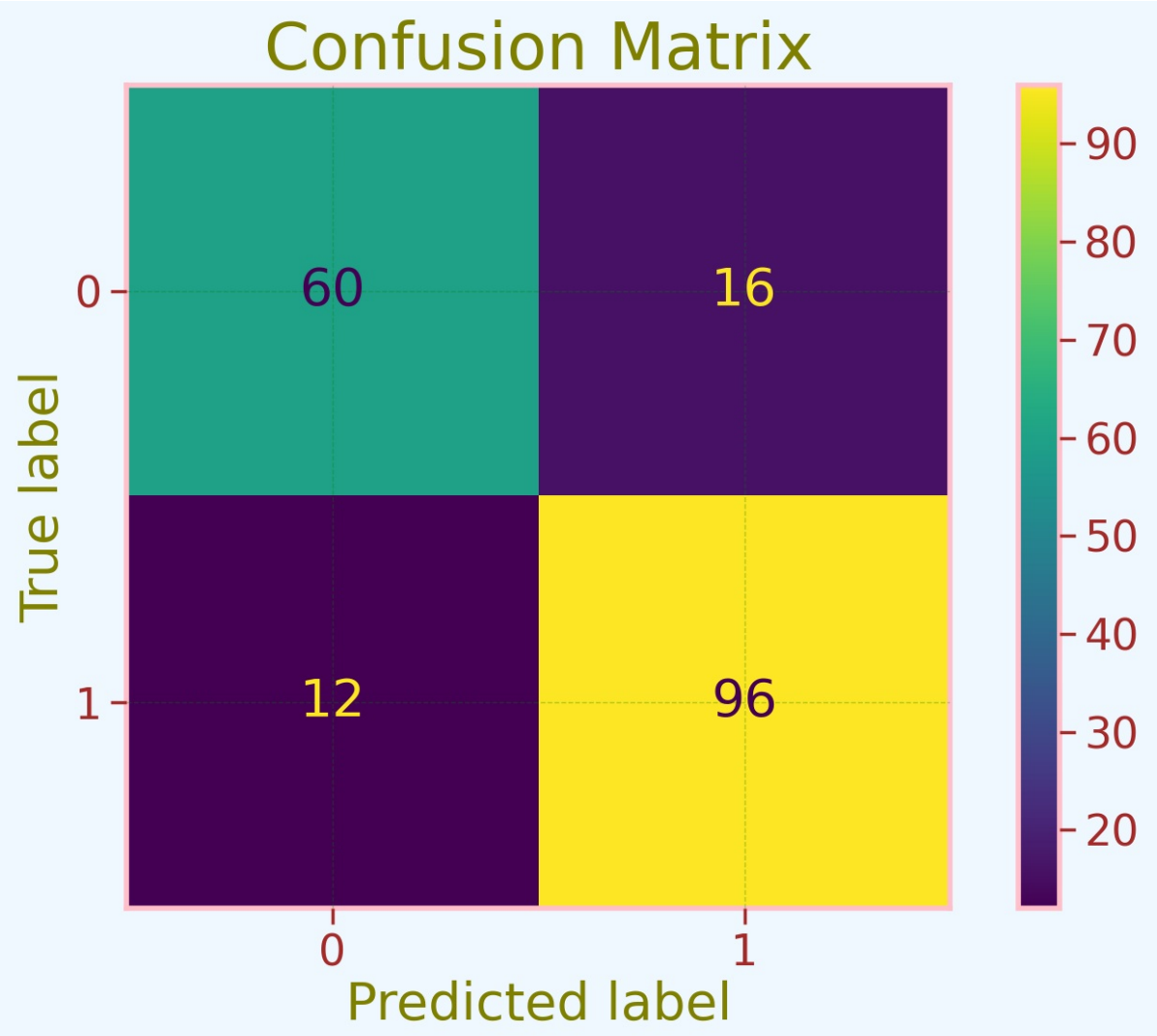
In [105…

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
y_pred_random = random_forest.predict(X_test)
random_forest.score(X_train, y_train)

random_forest_train = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_accuracy = round(accuracy_score(y_pred_random, y_test) * 100, 2)
random_forest_f1 = round(f1_score(y_pred_random, y_test) * 100, 2)

print("Training Accuracy     :",random_forest_train,"%")
print("Model Accuracy Score  :",random_forest_accuracy,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_random))
```

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 84.78 %
----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.83      0.79      0.81        76
           1       0.86      0.89      0.87       108

    accuracy                           0.85       184
   macro avg       0.85      0.84      0.84       184
weighted avg       0.85      0.85      0.85       184
```

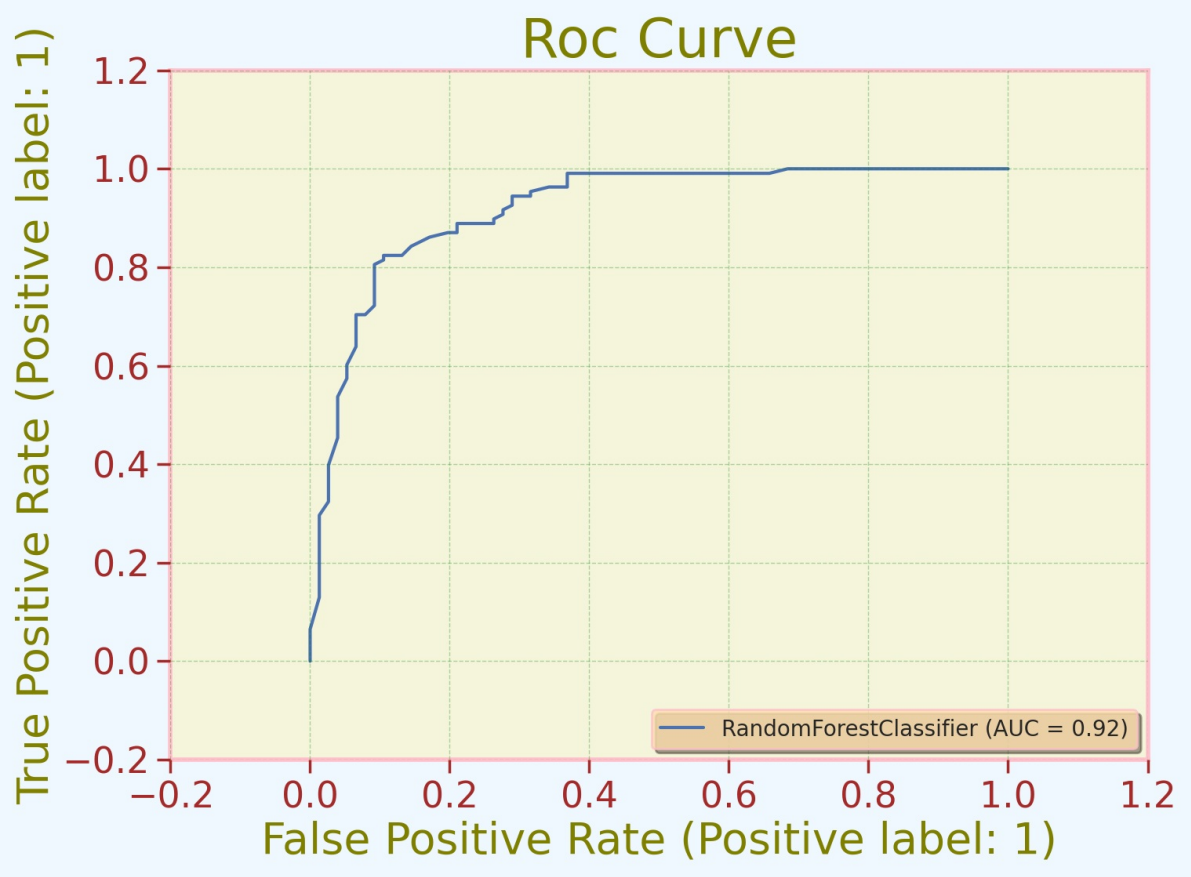
In [106…

```
plot_confusion_matrix(random_forest, X_test, y_test);
plt.title('Confusion Matrix');
```

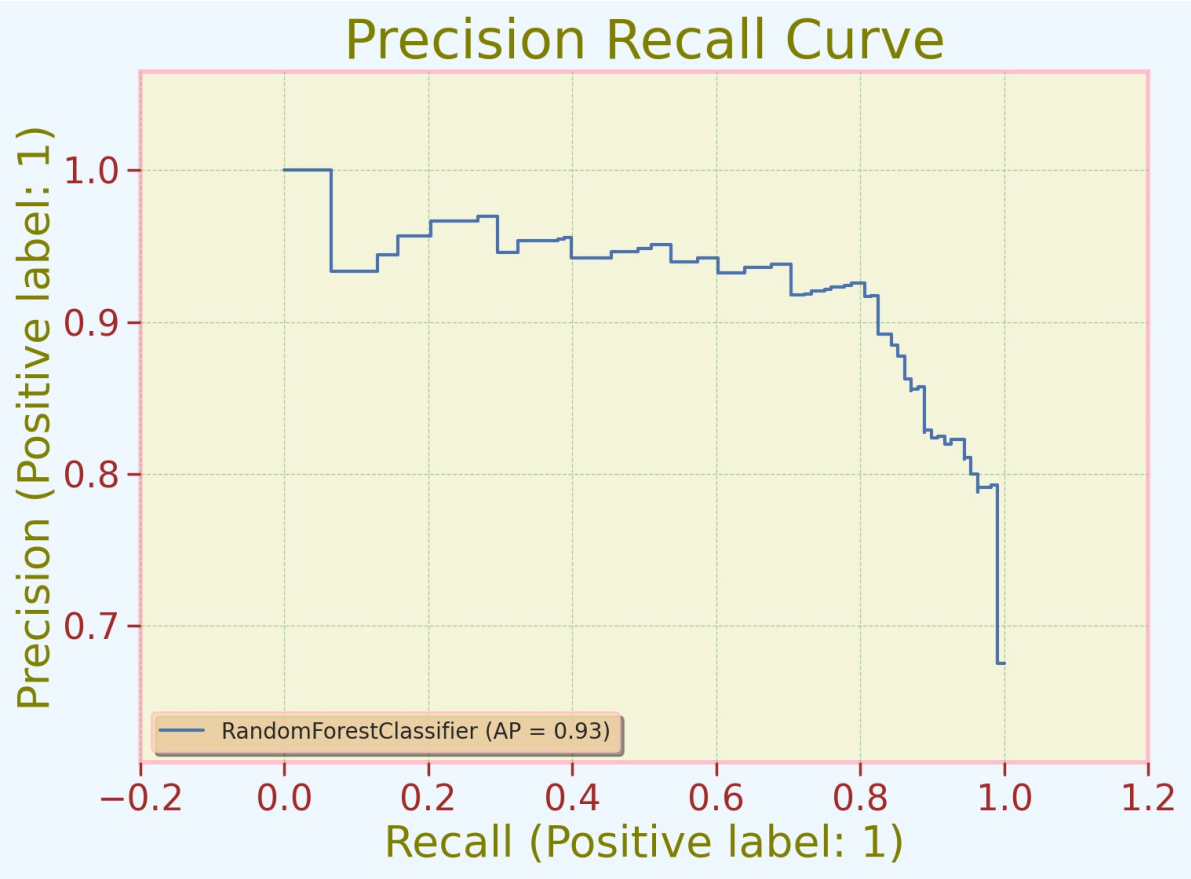


In [107…

```
plot_roc_curve(random_forest, X_test, y_test);
plt.title('Roc Curve');
```



In [108…
```
plot_precision_recall_curve(random_forest, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [109…
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=random_forest, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

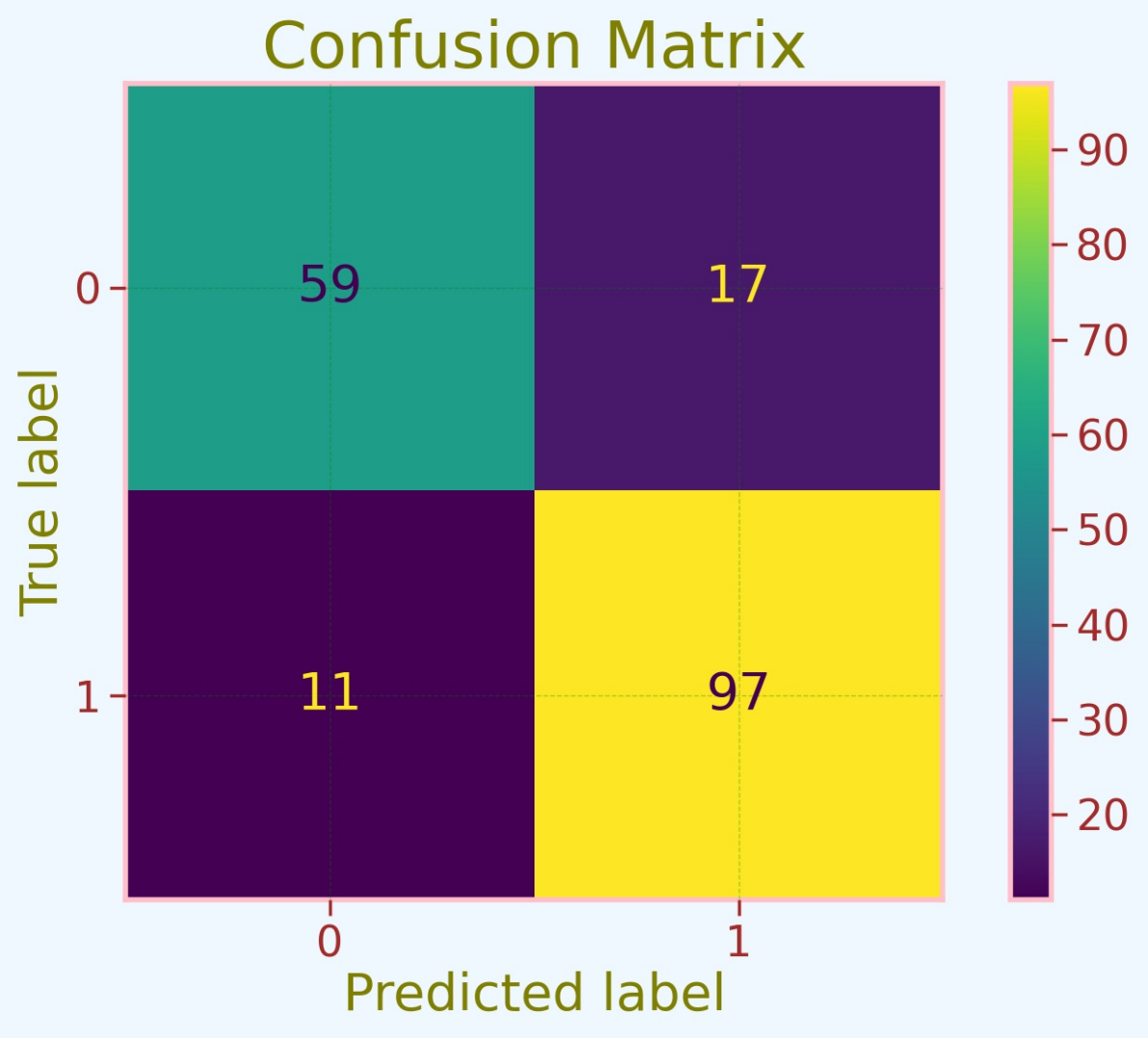
```
Model Accuracy Score: 86.62 %
Std. Dev: 3.21 %
```

MLPClassifier

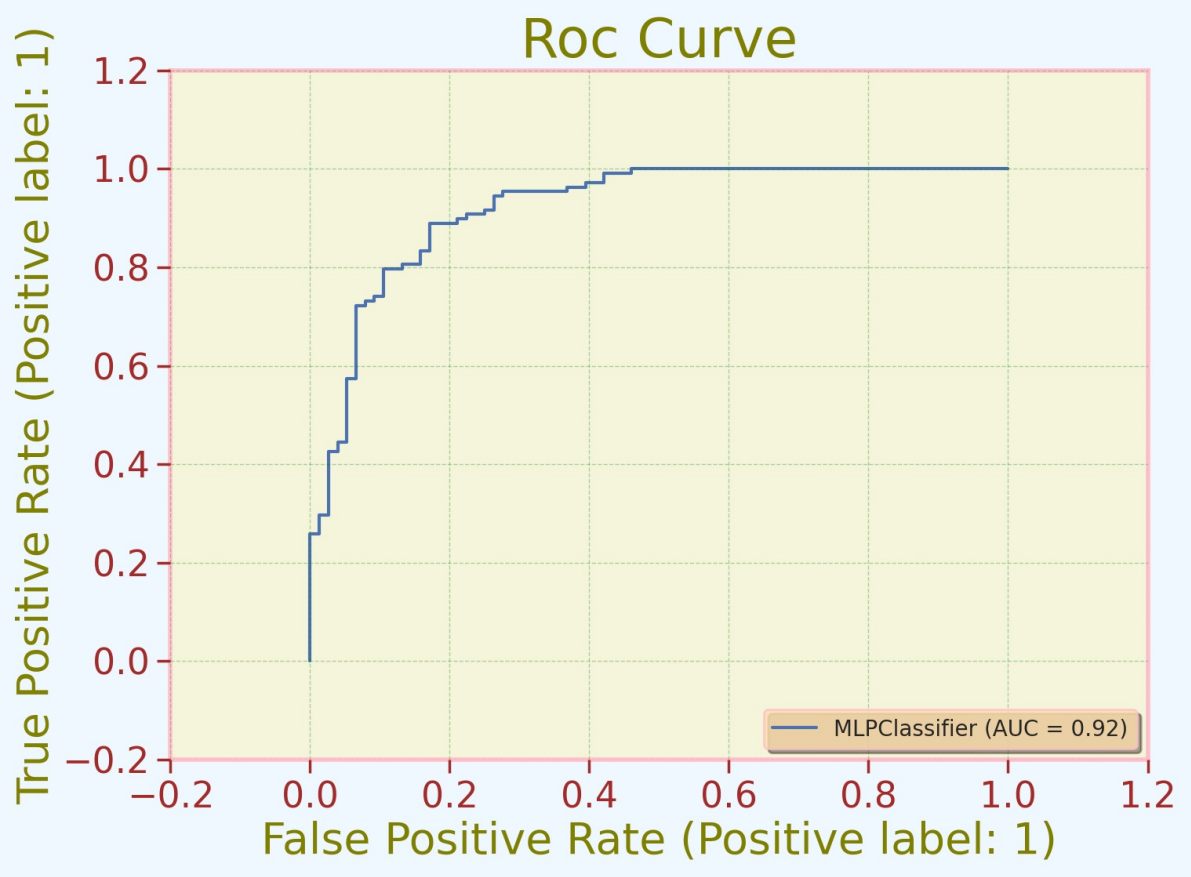
In [110…]
```
# MLPClassifier
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train,y_train)
y_pred_MLP = mlp.predict(X_test)
mlp.score(X_train, y_train)


mlp_train = round(mlp.score(X_train, y_train) * 100, 2)
mlp_accuracy = round(accuracy_score(y_pred_MLP, y_test) * 100, 2)
mlp_f1 = round(f1_score(y_pred_MLP, y_test) * 100, 2)

print("Training Accuracy     :",mlp_train,"%")
print("Model Accuracy Score  :",mlp_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_MLP))
```

```
Training Accuracy     : 87.99 %
Model Accuracy Score  : 84.78 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.84      0.78      0.81        76
           1       0.85      0.90      0.87       108

    accuracy                           0.85       184
   macro avg       0.85      0.84      0.84       184
weighted avg       0.85      0.85      0.85       184
```

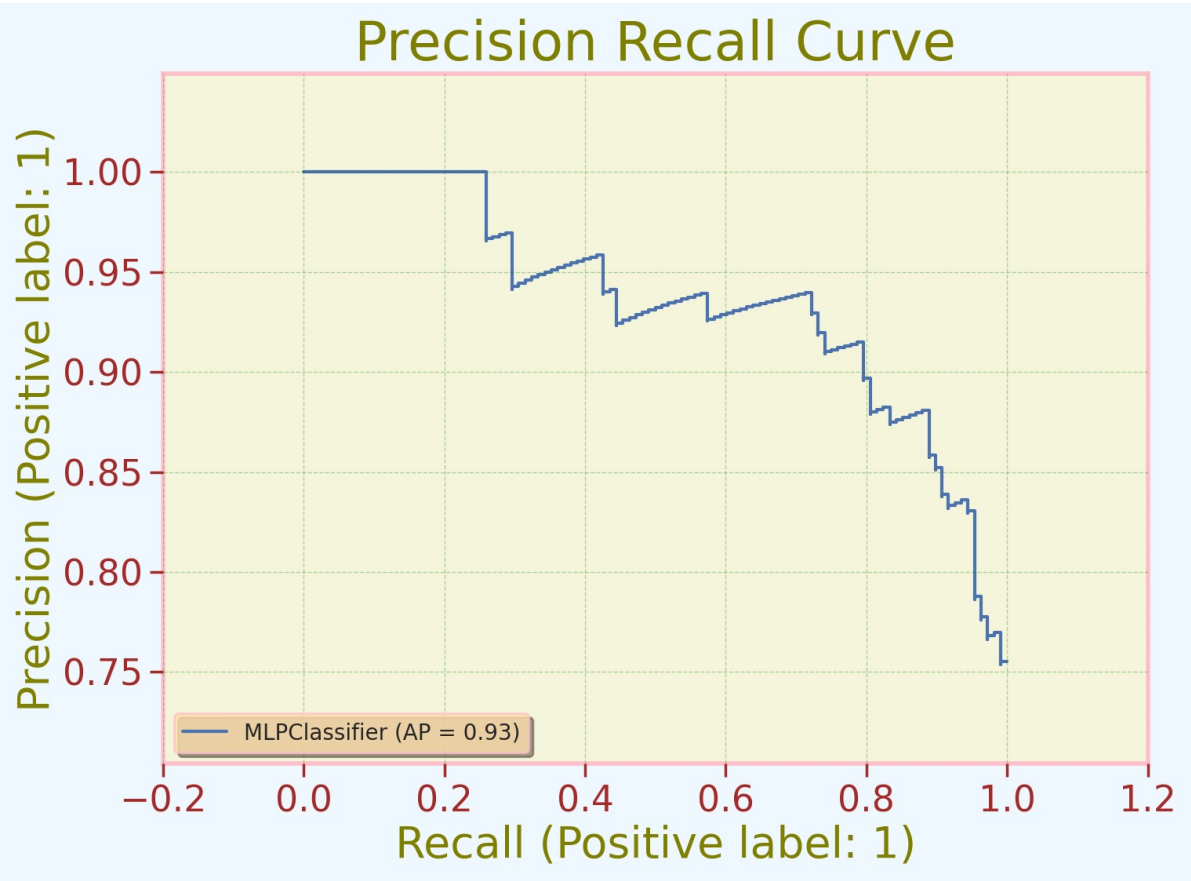
In [111…]
```
plot_confusion_matrix(mlp, X_test, y_test);
plt.title('Confusion Matrix');
```



In [112…]
```
plot_roc_curve(mlp, X_test, y_test);
plt.title('Roc Curve');
```



In [113…
```
plot_precision_recall_curve(mlp, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [114…
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=mlp, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

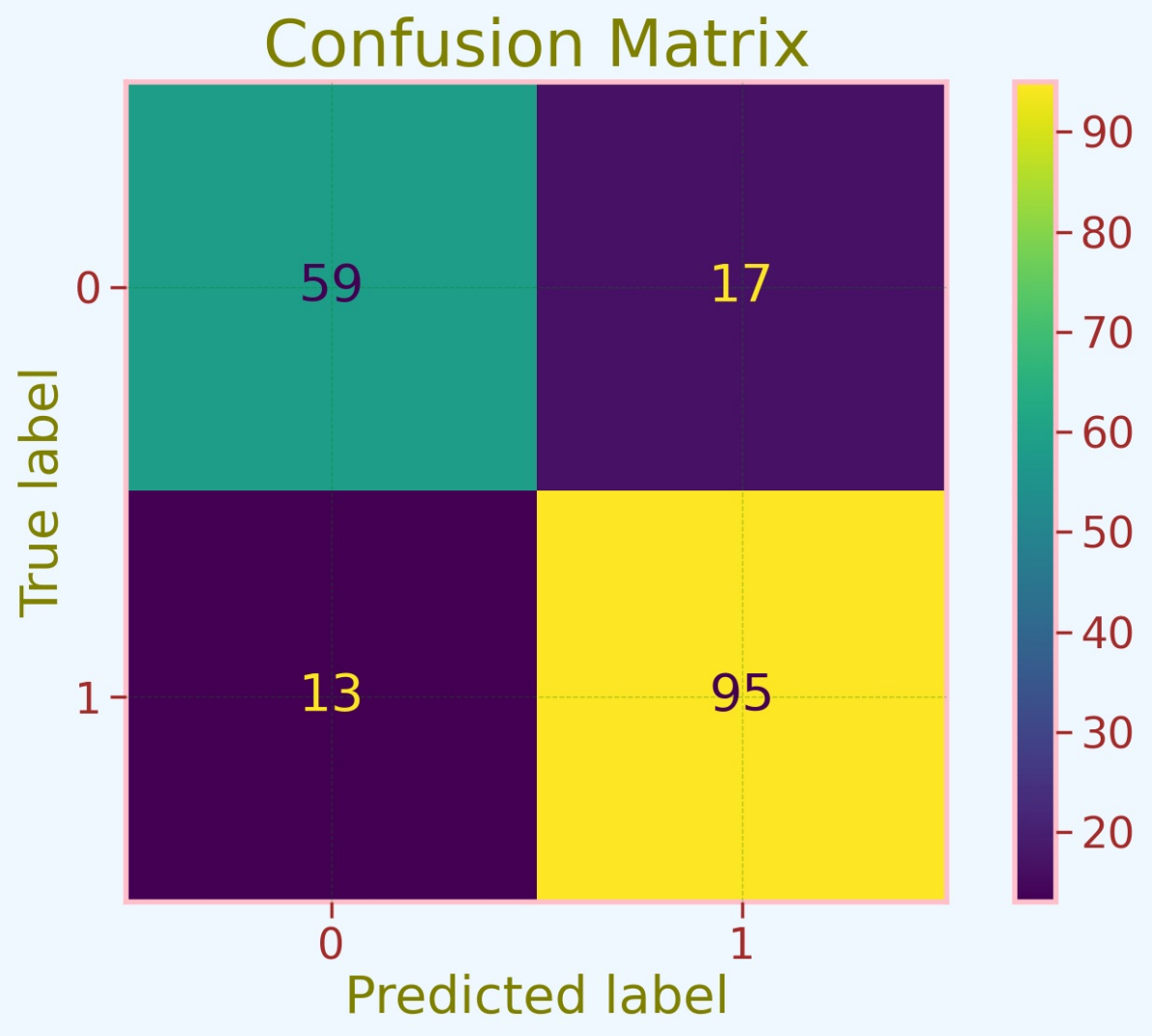
```
Model Accuracy Score: 85.39 %
Std. Dev: 2.65 %
```

XGBClassifier

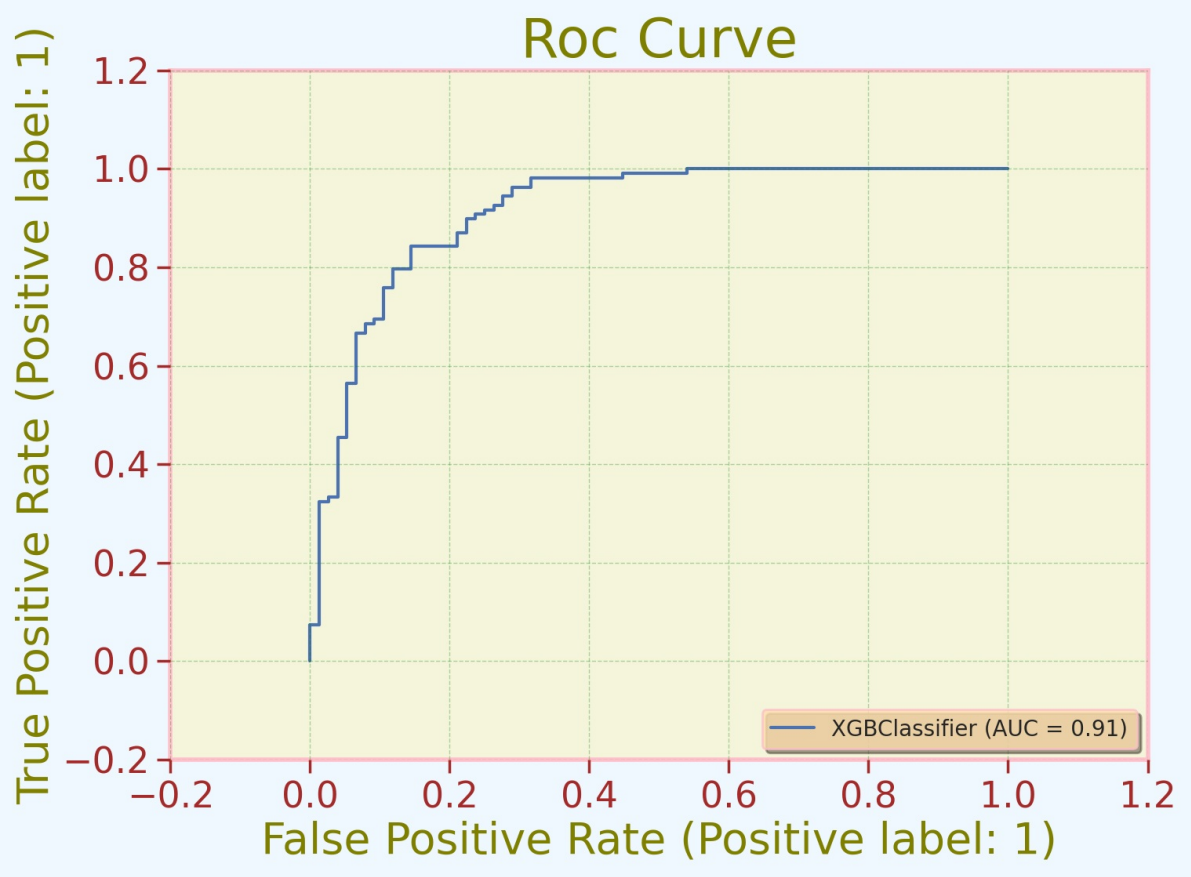
In [115…
```
import xgboost as Xgb
xgb = Xgb.XGBClassifier()
xgb.fit(X_train,y_train)
y_pred_xgb = xgb.predict(X_test)
xgb.score(X_train, y_train)

xgb_train = round(xgb.score(X_train, y_train) * 100, 2)
xgb_accuracy = round(accuracy_score(y_pred_xgb, y_test) * 100, 2)
xgb_f1 = round(f1_score(y_pred_xgb, y_test) * 100, 2)

print("Training Accuracy     :",xgb_train,"%")
print("Model Accuracy Score  :",xgb_accuracy,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_xgb))
```

```
Training Accuracy     : 100.0 %
Model Accuracy Score  : 83.7 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.82      0.78      0.80        76
           1       0.85      0.88      0.86       108

    accuracy                           0.84       184
   macro avg       0.83      0.83      0.83       184
weighted avg       0.84      0.84      0.84       184
```

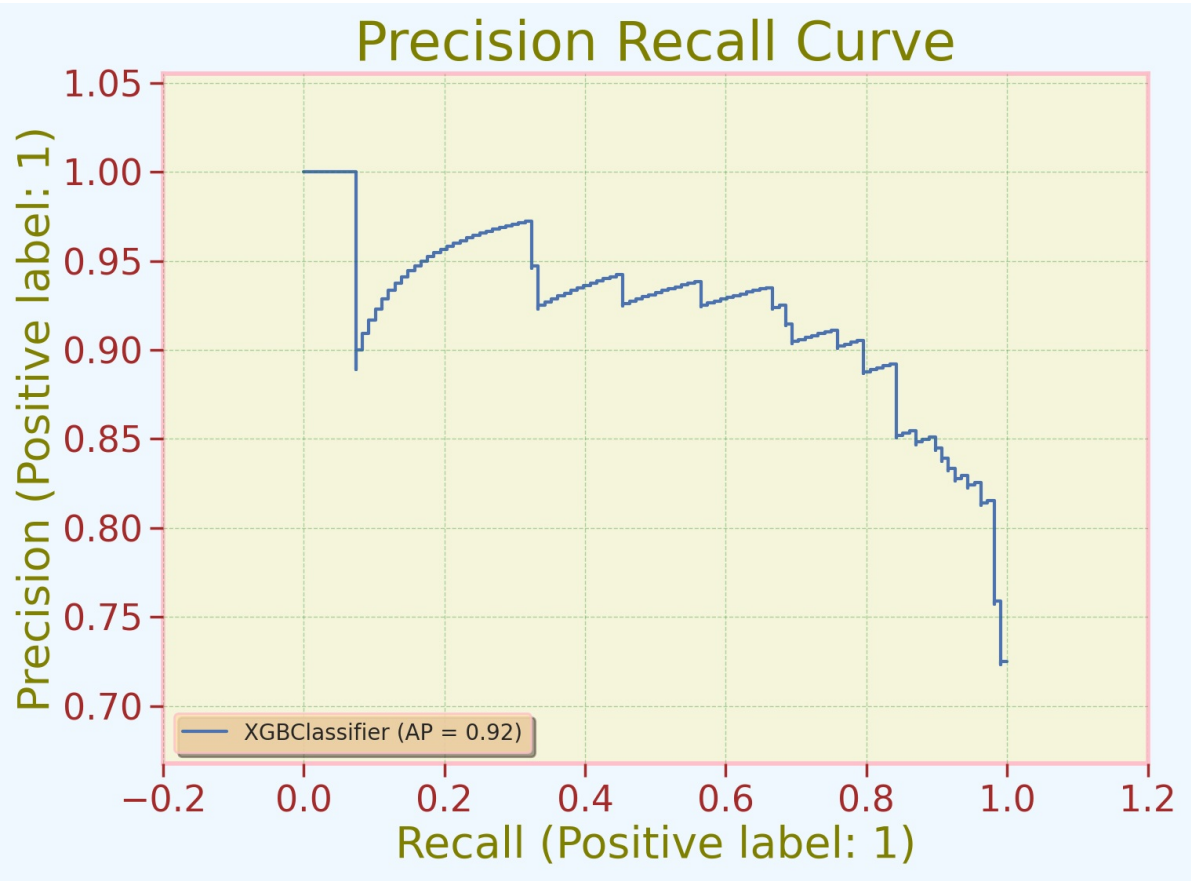
In [116…
```
plot_confusion_matrix(xgb, X_test, y_test);
plt.title('Confusion Matrix');
```



In [117…
```
plot_roc_curve(xgb, X_test, y_test);
plt.title('Roc Curve');
```



In [118…
```
plot_precision_recall_curve(xgb, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [119…
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=xgb, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

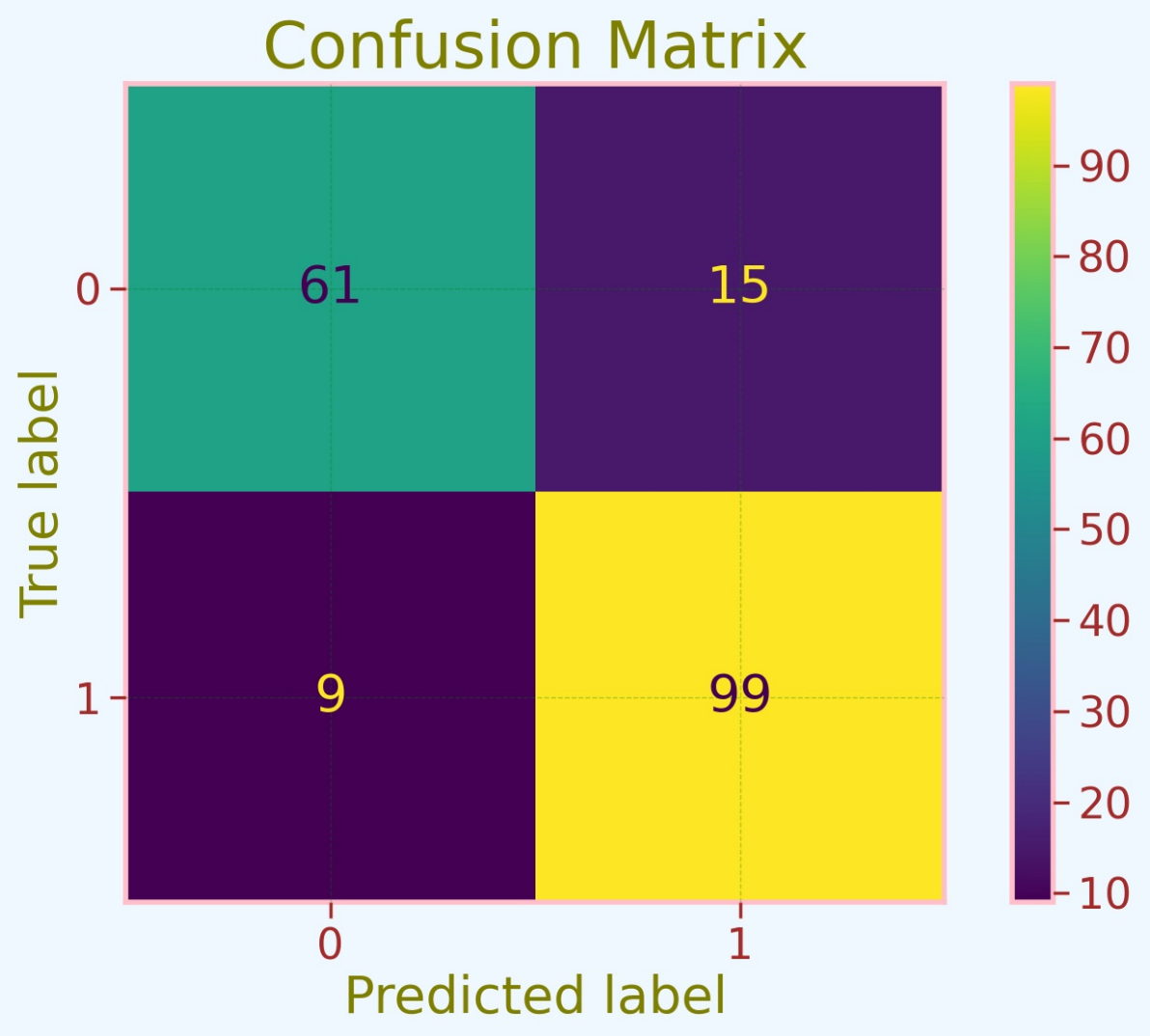
```
Model Accuracy Score: 83.75 %
Std. Dev: 4.43 %
```

GaussianProcessClassifier

In [120…

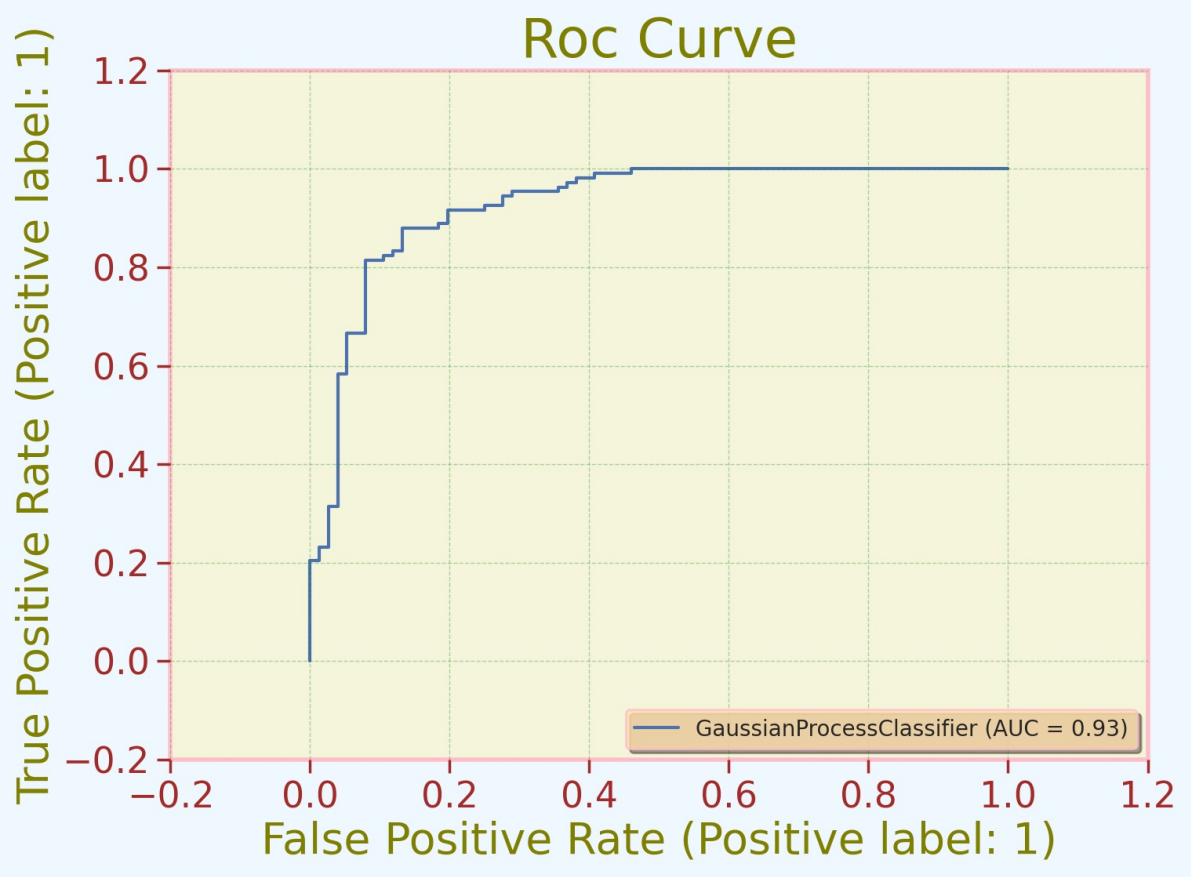
```
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
RBF = GaussianProcessClassifier(1.0 * RBF(1.0))
RBF.fit(X_train,y_train)
y_pred_RBF = RBF.predict(X_test)

RBF_train = round(RBF.score(X_train, y_train) * 100, 2)
RBF_accuracy = round(accuracy_score(y_pred_RBF, y_test) * 100, 2)
RBF_f1 = round(f1_score(y_pred_RBF, y_test) * 100, 2)

print("Training Accuracy    :",RBF_train ,"%")
print("Model Accuracy Score :",RBF_accuracy ,"%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification Report: \n",classification_report(y_test,y_pred_RBF))
print("\033[1m-------------------------------------------------------\033[0m")
plot_confusion_matrix(RBF, X_test, y_test);
plt.title('Confusion Matrix');
```

```
Training Accuracy    : 88.27 %
Model Accuracy Score : 86.96 %
-------------------------------------------------------
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.80      0.84        76
           1       0.87      0.92      0.89       108

    accuracy                           0.87       184
   macro avg       0.87      0.86      0.86       184
weighted avg       0.87      0.87      0.87       184

-------------------------------------------------------
```

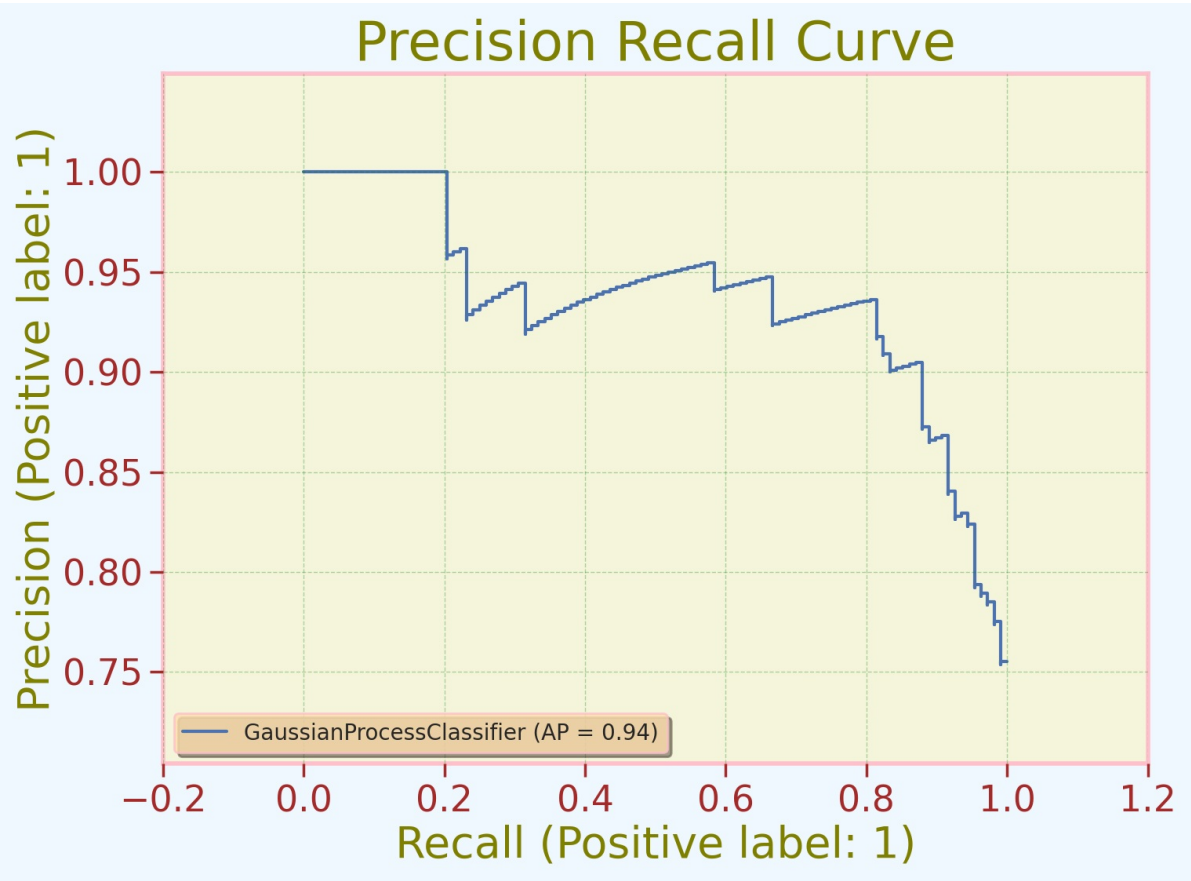


In [121…

```
plot_roc_curve(RBF, X_test, y_test);
plt.title('Roc Curve');
```



In [122…
```
plot_precision_recall_curve(RBF, X_test, y_test)
plt.title('Precision Recall Curve');
```



In [123…
```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=RBF, X = X_train, y=y_train, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 85.39 %
Std. Dev: 2.65 %
```

Comparing Models

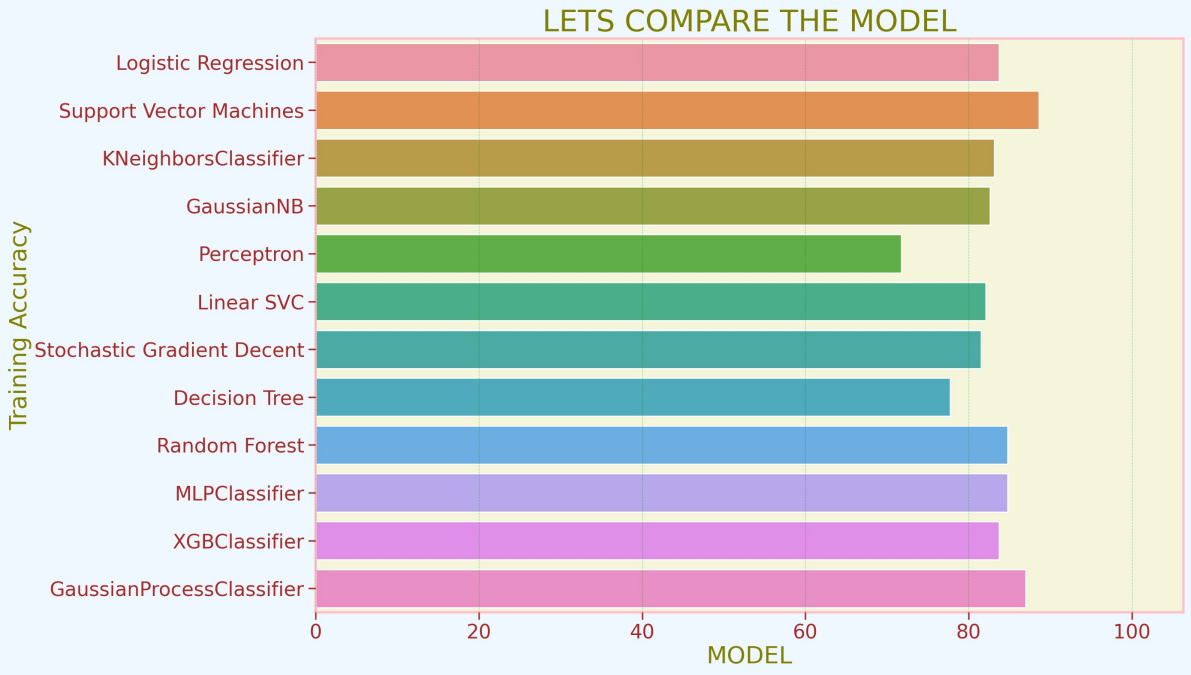
In [124…
```
models = pd.DataFrame({
    'Model': [
        'Logistic Regression','Support Vector Machines', 'KNeighborsClassifier',
        'GaussianNB','Perceptron', 'Linear SVC','Stochastic Gradient Decent',
        'Decision Tree','Random Forest', 'MLPClassifier' ,'XGBClassifier', 'GaussianProcessClassif
    ],
    'Training Accuracy': [
        log_train, svc_train, knn_train, gaussian_train, perceptron_train,
        linear_svc_train, sgd_train, decision_train, random_forest_train,
        mlp_train, xgb_train, RBF_train
    ],
    'Model f1 Score': [
        log_f1, svc_f1, knn_f1, gaussian_f1, perceptron_f1,
        linear_svc_f1, sgd_f1, decision_f1, random_forest_f1,
        mlp_f1, xgb_f1, RBF_f1
    ],
    'Model Accuracy Score': [
        log_accuracy, svc_accuracy, knn_accuracy, gaussian_accuracy, perceptron_accuracy,
        linear_svc_accuracy, sgd_accuracy, decision_accuracy, random_forest_accuracy,
        mlp_accuracy, xgb_accuracy, RBF_accuracy
    ]

})
```

In [125…
```
# Accuracy Comparison Table
models.sort_values(
    by='Model Accuracy Score', ascending=False).style.background_gradient(
        cmap='cool').hide_index().set_properties(**{
            'font-family': 'Lucida Calligraphy',
            'color': 'LigntGreen',
            'font-size': '15px'
        })
```

Out[125]:

Model	Training Accuracy	Model f1 Score	Model Accuracy Score
Support Vector Machines	87.31	90.58	88.59
GaussianProcessClassifier	88.27	89.19	86.96
Random Forest	100.00	87.27	84.78
MLPClassifier	87.99	87.39	84.78
Logistic Regression	85.13	86.11	83.70
XGBClassifier	100.00	86.36	83.70
KNeighborsClassifier	89.63	85.97	83.15
GaussianNB	83.77	84.47	82.61
Linear SVC	85.13	84.79	82.07
Stochastic Gradient Decent	84.72	84.82	81.52
Decision Tree	100.00	81.45	77.72
Perceptron	71.08	69.77	71.74

In [126…
```
plt.rcParams['figure.figsize'] = (14,8)
plt.rcParams['font.size'] = 10
plt.rcParams['axes.xmargin'] = .2
plt.rcParams["axes.ymargin"] = .2


import seaborn as sns

sns.barplot(y= 'Model', x= 'Model Accuracy Score', data= models)
plt.title('LETS COMPARE THE MODEL')
plt.xlabel('MODEL')
plt.ylabel('Training Accuracy');
```



In [127…
```
prediction1 = random_forest.predict(X_test)
print(prediction1)
```

```
[1 1 1 1 1 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 1 0 1 0 0 0 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0
 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 0 0 0
 0 0 1 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 1 1 0 1 0 1 1 0 0
 1 1 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0]
```

In [128…
```
cross_checking = pd.DataFrame({'Actual' : y_test , 'Predicted' : prediction1})
cross_checking.sample(15).style.background_gradient(
        cmap='coolwarm').set_properties(**{
            'font-family': 'Lucida Calligraphy',
            'color': 'LigntGreen',
            'font-size': '15px'
        })
```

Out[128]:

	Actual	Predicted
748	1	1
867	1	0
215	0	0
787	1	1
467	0	0
507	0	0
328	1	1
626	0	0
70	1	1
44	1	1
433	1	1
643	1	1
104	1	1
254	1	1
746	1	1