#**Heart Disease Prediction Using Logistic Regression**

**Step 1:** Importing libraries

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns  
%matplotlib inline  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.metrics import roc\_auc\_score

**Step 2:** Mounting drive into the colab notebook

from google.colab import drive  
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

**Step 3:** Reading data from the drive

heart=pd.read\_csv('/content/drive/MyDrive/heart.csv')  
heart

Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG \  
0 40 M ATA 140 289 0 Normal   
1 49 F NAP 160 180 0 Normal   
2 37 M ATA 130 283 0 ST   
3 48 F ASY 138 214 0 Normal   
4 54 M NAP 150 195 0 Normal   
.. ... .. ... ... ... ... ...   
913 45 M TA 110 264 0 Normal   
914 68 M ASY 144 193 1 Normal   
915 57 M ASY 130 131 0 Normal   
916 57 F ATA 130 236 0 LVH   
917 38 M NAP 138 175 0 Normal   
  
 MaxHR ExerciseAngina Oldpeak ST\_Slope HeartDisease   
0 172 N 0.0 Up 0   
1 156 N 1.0 Flat 1   
2 98 N 0.0 Up 0   
3 108 Y 1.5 Flat 1   
4 122 N 0.0 Up 0   
.. ... ... ... ... ...   
913 132 N 1.2 Flat 1   
914 141 N 3.4 Flat 1   
915 115 Y 1.2 Flat 1   
916 174 N 0.0 Flat 1   
917 173 N 0.0 Up 0   
  
[918 rows x 12 columns]

##Data Preprocessing

**Step 4:** Analysing the dataset.

1. shape of the graph
2. columns of the graphs.
3. checking null values
4. checking the presence of duplicates
5. checking mean,min,etc. using describe function.

#shape of the graph  
heart.shape

(918, 12)

#columns of the graphs.  
heart.columns

Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',  
 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST\_Slope',  
 'HeartDisease'],  
 dtype='object')

heart.isnull().sum()

Age 0  
Sex 0  
ChestPainType 0  
RestingBP 0  
Cholesterol 0  
FastingBS 0  
RestingECG 0  
MaxHR 0  
ExerciseAngina 0  
Oldpeak 0  
ST\_Slope 0  
HeartDisease 0  
dtype: int64

#checking null values  
heart.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

heart.describe().T

count mean std min 25% 50% 75% max  
Age 918.0 53.510893 9.432617 28.0 47.00 54.0 60.0 77.0  
RestingBP 918.0 132.396514 18.514154 0.0 120.00 130.0 140.0 200.0  
Cholesterol 918.0 198.799564 109.384145 0.0 173.25 223.0 267.0 603.0  
FastingBS 918.0 0.233115 0.423046 0.0 0.00 0.0 0.0 1.0  
MaxHR 918.0 136.809368 25.460334 60.0 120.00 138.0 156.0 202.0  
Oldpeak 918.0 0.887364 1.066570 -2.6 0.00 0.6 1.5 6.2  
HeartDisease 918.0 0.553377 0.497414 0.0 0.00 1.0 1.0 1.0

**Step 5:** Dividing numerical and categorial data

1. Here the value which are unique in the categorical data are atmost 4.
2. So if we calculate the unique value count and get more than 6(let's consider) then we can consider it as numerical data else categorical.

col = list(heart.columns)  
categorical\_features = []  
numerical\_features = []  
for i in col:  
 if len(heart[i].unique()) > 6:  
 numerical\_features.append(i)  
 else:  
 categorical\_features.append(i)  
  
print('Categorical Features :',\*categorical\_features)  
print('Numerical Features :',\*numerical\_features)

Categorical Features : Sex ChestPainType FastingBS RestingECG ExerciseAngina ST\_Slope HeartDisease  
Numerical Features : Age RestingBP Cholesterol MaxHR Oldpeak

**Step 6:** creating a deep copy of our data set and converting all the columns to numerical values

from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()  
deep\_copy\_heart = heart.copy(deep = True)  
  
deep\_copy\_heart['Sex'] = le.fit\_transform(deep\_copy\_heart['Sex'])  
deep\_copy\_heart['ChestPainType'] = le.fit\_transform(deep\_copy\_heart['ChestPainType'])  
deep\_copy\_heart['RestingECG'] = le.fit\_transform(deep\_copy\_heart['RestingECG'])  
deep\_copy\_heart['ExerciseAngina'] = le.fit\_transform(deep\_copy\_heart['ExerciseAngina'])  
deep\_copy\_heart['ST\_Slope'] = le.fit\_transform(deep\_copy\_heart['ST\_Slope'])

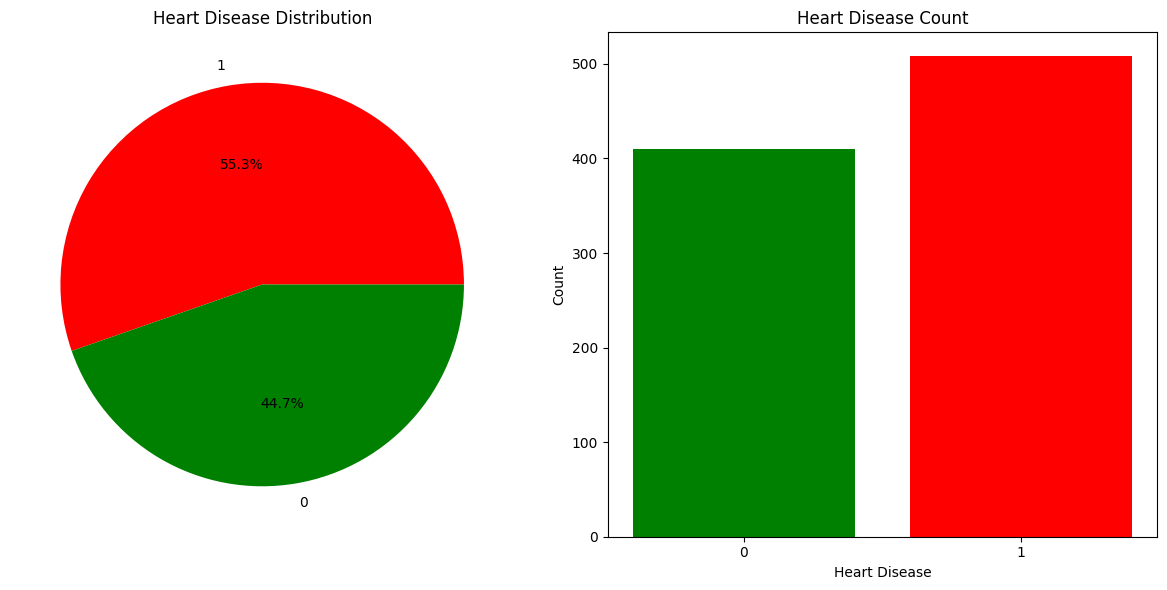
deep\_copy\_heart

Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG \  
0 40 1 1 140 289 0 1   
1 49 0 2 160 180 0 1   
2 37 1 1 130 283 0 2   
3 48 0 0 138 214 0 1   
4 54 1 2 150 195 0 1   
.. ... ... ... ... ... ... ...   
913 45 1 3 110 264 0 1   
914 68 1 0 144 193 1 1   
915 57 1 0 130 131 0 1   
916 57 0 1 130 236 0 0   
917 38 1 2 138 175 0 1   
  
 MaxHR ExerciseAngina Oldpeak ST\_Slope HeartDisease   
0 172 0 0.0 2 0   
1 156 0 1.0 1 1   
2 98 0 0.0 2 0   
3 108 1 1.5 1 1   
4 122 0 0.0 2 0   
.. ... ... ... ... ...   
913 132 0 1.2 1 1   
914 141 0 3.4 1 1   
915 115 1 1.2 1 1   
916 174 0 0.0 1 1   
917 173 0 0.0 2 0   
  
[918 rows x 12 columns]

##Data Visualization

**Step 7:** visualizing the count of the total no. of heart diseased people along with people without heart disease

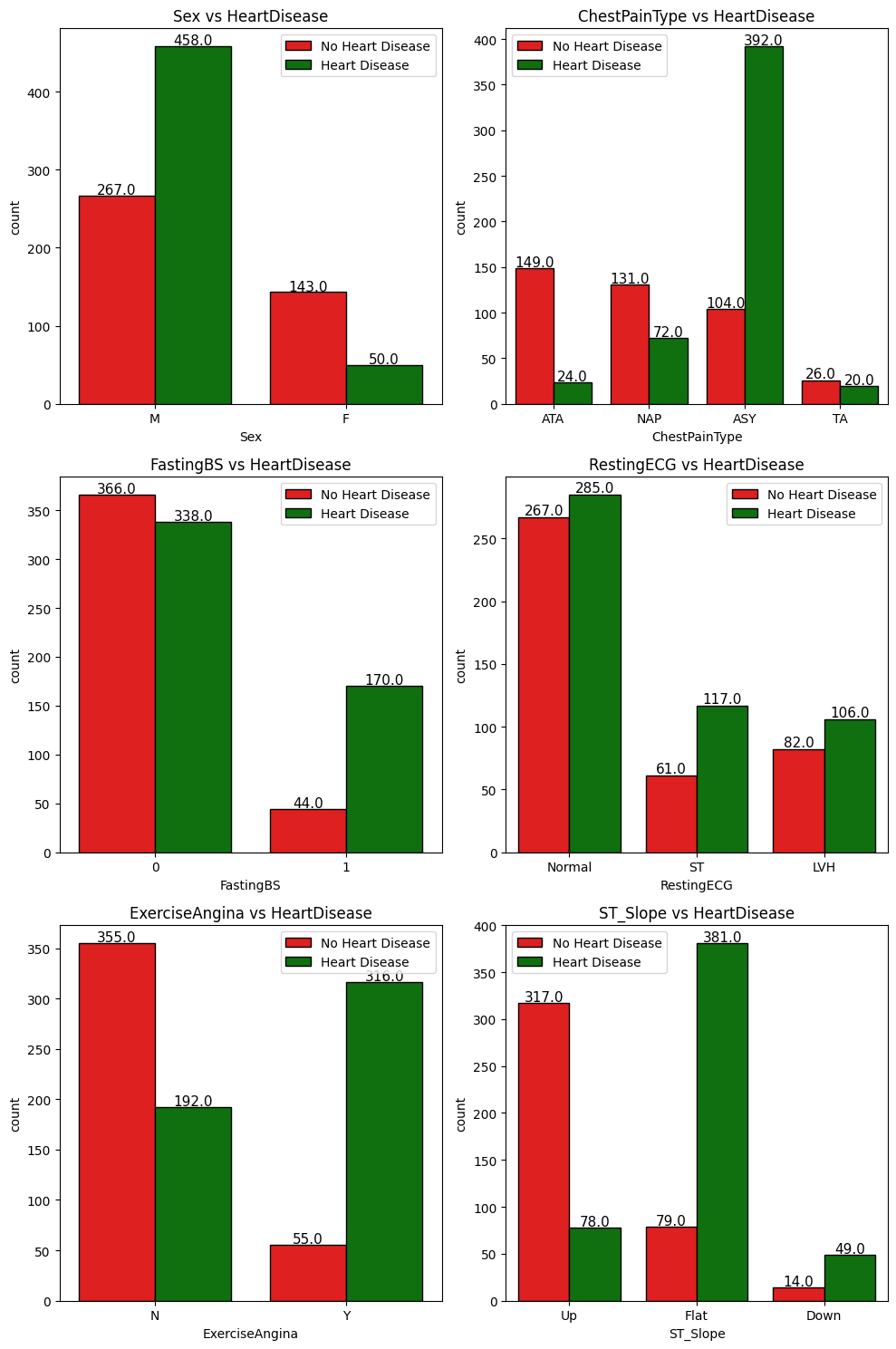
heart\_disease\_counts = heart['HeartDisease'].value\_counts()  
  
# Define the colors. '1' for heart disease is red, '0' for healthy heart is any other color (e.g., green)  
colors = ['red' if label == 1 else 'green' for label in heart\_disease\_counts.index]  
  
fig, axs = plt.subplots(1, 2, figsize=(12, 6)) # 1 row, 2 columns  
  
# Create a pie chart  
axs[0].pie(heart\_disease\_counts, labels = heart\_disease\_counts.index, colors = colors, autopct='%1.1f%%')  
axs[0].set\_title('Heart Disease Distribution')  
  
# Create a bar graph  
axs[1].bar(heart\_disease\_counts.index, heart\_disease\_counts, color = colors)  
axs[1].set\_xlabel('Heart Disease')  
axs[1].set\_ylabel('Count')  
axs[1].set\_title('Heart Disease Count')  
axs[1].set\_xticks(heart\_disease\_counts.index)  
  
plt.tight\_layout()  
plt.show()



**Step 8**:\*\* Aanalysing the data by plotting graph for different scenario.

data=heart

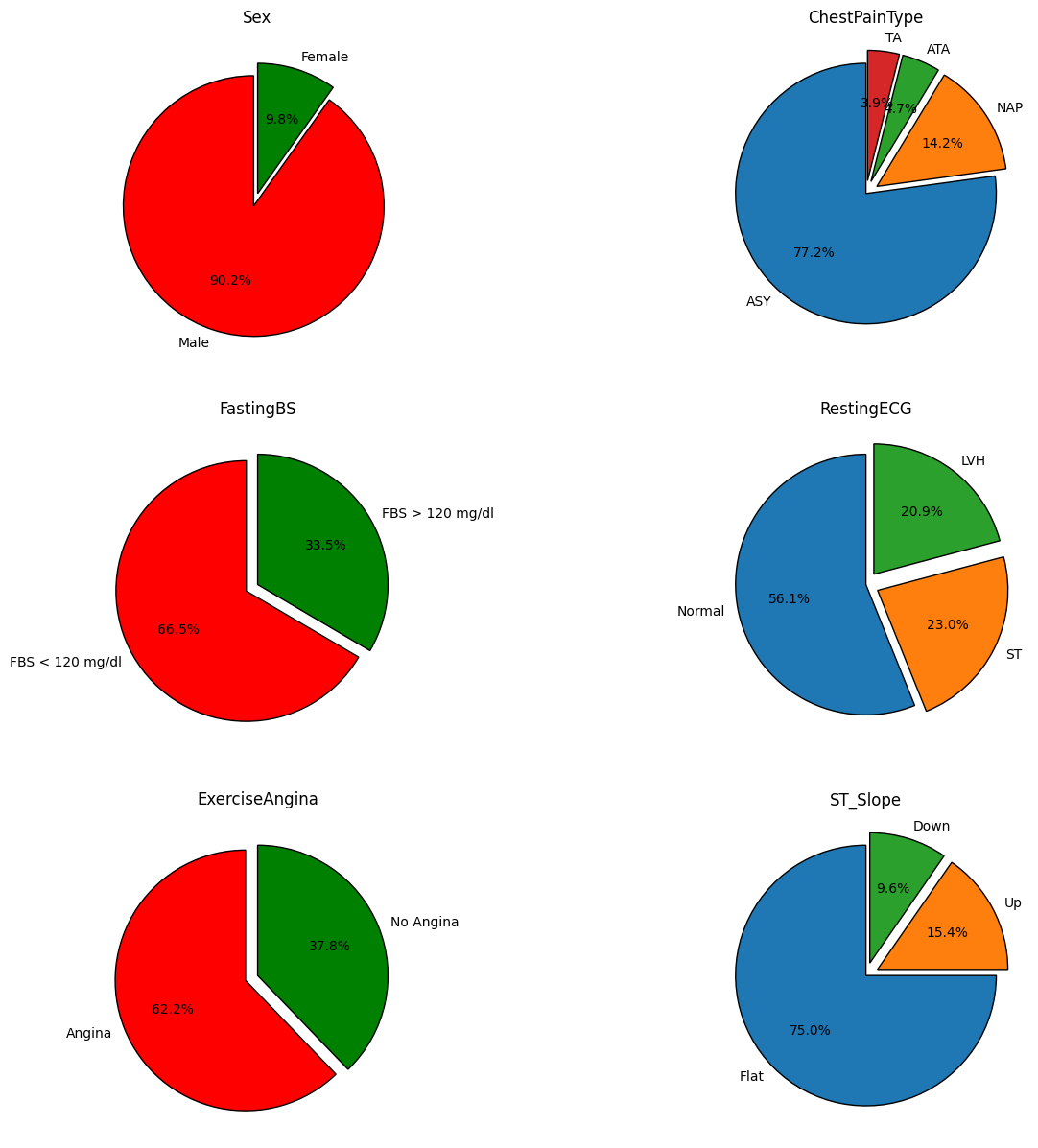
fig, ax = plt.subplots(nrows = 3, ncols = 2, figsize = (10, 15))  
for i in range(len(categorical\_features) - 1):  
 ax = plt.subplot(3, 2, i+1)  
 sns.countplot(x=categorical\_features[i], data=data, hue="HeartDisease", palette=colors, edgecolor='black', ax=ax)  
 for rect in ax.patches:  
 ax.text(rect.get\_x() + rect.get\_width() / 2, rect.get\_height() + 2, rect.get\_height(), horizontalalignment='center', fontsize = 11)  
 title = categorical\_features[i] + ' vs HeartDisease'  
 plt.legend(['No Heart Disease','Heart Disease'])  
 plt.title(title)  
plt.tight\_layout()  
plt.show()



Number Of Male And Female Having Symptoms in Different Scenarios

sex = heart[heart['HeartDisease'] == 1]['Sex'].value\_counts()  
sex = [sex[0] / sum(sex) \* 100, sex[1] / sum(sex) \* 100]  
  
cp = heart[heart['HeartDisease'] == 1]['ChestPainType'].value\_counts()  
cp = [cp[0] / sum(cp) \* 100,cp[1] / sum(cp) \* 100,cp[2] / sum(cp) \* 100,cp[3] / sum(cp) \* 100]  
  
fbs = heart[heart['HeartDisease'] == 1]['FastingBS'].value\_counts()  
fbs = [fbs[0] / sum(fbs) \* 100,fbs[1] / sum(fbs) \* 100]  
  
restecg = heart[heart['HeartDisease'] == 1]['RestingECG'].value\_counts()  
restecg = [restecg[0] / sum(restecg) \* 100,restecg[1] / sum(restecg) \* 100,restecg[2] / sum(restecg) \* 100]  
  
exang = heart[heart['HeartDisease'] == 1]['ExerciseAngina'].value\_counts()  
exang = [exang[0] / sum(exang) \* 100,exang[1] / sum(exang) \* 100]  
  
slope = heart[heart['HeartDisease'] == 1]['ST\_Slope'].value\_counts()  
slope = [slope[0] / sum(slope) \* 100,slope[1] / sum(slope) \* 100,slope[2] / sum(slope) \* 100]  
ax,fig = plt.subplots(nrows = 4,ncols = 2,figsize = (15,15))  
  
plt.subplot(3,2,1)  
plt.pie(sex,labels = ['Male','Female'],autopct='%1.1f%%',startangle = 90,explode = (0.1,0),colors = colors,  
 wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})  
plt.title('Sex');  
  
plt.subplot(3,2,2)  
plt.pie(cp,labels = ['ASY', 'NAP', 'ATA', 'TA'],autopct='%1.1f%%',startangle = 90,explode = (0,0.1,0.1,0.1),  
 wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})  
plt.title('ChestPainType');  
  
plt.subplot(3,2,3)  
plt.pie(fbs,labels = ['FBS < 120 mg/dl','FBS > 120 mg/dl'],autopct='%1.1f%%',startangle = 90,explode = (0.1,0),colors = colors,  
 wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})  
plt.title('FastingBS');  
  
plt.subplot(3,2,4)  
plt.pie(restecg,labels = ['Normal','ST','LVH'],autopct='%1.1f%%',startangle = 90,explode = (0,0.1,0.1),  
 wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})  
plt.title('RestingECG');  
  
plt.subplot(3,2,5)  
plt.pie(exang,labels = ['Angina','No Angina'],autopct='%1.1f%%',startangle = 90,explode = (0.1,0),colors = colors,  
 wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})  
plt.title('ExerciseAngina');  
  
plt.subplot(3,2,6)  
plt.pie(slope,labels = ['Flat','Up','Down'],autopct='%1.1f%%',startangle = 90,explode = (0,0.1,0.1),  
 wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})  
plt.title('ST\_Slope');

<ipython-input-33-47535b4c389c>:20: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.  
 plt.subplot(3,2,1)  
<ipython-input-33-47535b4c389c>:25: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.  
 plt.subplot(3,2,2)  
<ipython-input-33-47535b4c389c>:30: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.  
 plt.subplot(3,2,3)  
<ipython-input-33-47535b4c389c>:35: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.  
 plt.subplot(3,2,4)  
<ipython-input-33-47535b4c389c>:40: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.  
 plt.subplot(3,2,5)  
<ipython-input-33-47535b4c389c>:45: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.  
 plt.subplot(3,2,6)



**Step 9:** Splitting the data as training set and testing set.

X= deep\_copy\_heart.iloc[:, :-1].values  
Y=deep\_copy\_heart.iloc[:, 11].values

features = deep\_copy\_heart[deep\_copy\_heart.columns.drop(['HeartDisease','RestingBP','RestingECG'])].values  
target = deep\_copy\_heart['HeartDisease'].values  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size = 0.20, random\_state = 2)

##Logistic Regression and testing accuracy

predictor= LogisticRegression()

predictor.fit(x\_train, y\_train)

/usr/local/lib/python3.10/dist-packages/sklearn/linear\_model/\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.  
  
Increase the number of iterations (max\_iter) or scale the data as shown in:  
 https://scikit-learn.org/stable/modules/preprocessing.html  
Please also refer to the documentation for alternative solver options:  
 https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression  
 n\_iter\_i = \_check\_optimize\_result(

LogisticRegression()

#accuracy of train data  
predict\_train=predictor.predict(x\_train)  
training\_accuracy = accuracy\_score(predict\_train,y\_train)

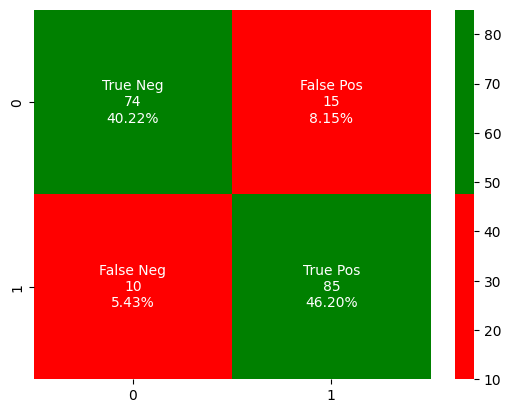
print('Accuracy on the training data is: ', training\_accuracy)

Accuracy on the training data is: 0.8583106267029973

predict\_test=predictor.predict(x\_test)  
testing\_accuracy = accuracy\_score(predict\_test,y\_test)

cm = confusion\_matrix(y\_test,predictor.predict(x\_test))  
names = ['True Neg','False Pos','False Neg','True Pos']  
counts = [value for value in cm.flatten()]  
percentages = ['{0:.2%}'.format(value) for value in cm.flatten()/np.sum(cm)]  
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names,counts,percentages)]  
labels = np.asarray(labels).reshape(2,2)  
sns.heatmap(cm,annot = labels,cmap = colors,fmt ='')

<Axes: >



print('Accuracy on the testing data is: ', testing\_accuracy)

Accuracy on the testing data is: 0.8641304347826086